Tesi doctoral

Síntesi d'alt nivell de circuits asíncrons

Autora: Rosa M. Badia i Sala

Director: Jordi Cortadella i Fortuny

Barcelona, maig de 1994

UPC

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Departament d'Arquitectura de Computadors

Tesi doctoral presentada per Rosa M. Badia i Sala per tal
d'aconseguir el grau de Doctora en Informàtica per la
Universitat Politècnica de Catalunya

Tribunal

José M. Llaberia, President

Ramon Beivide Secretari

Joan Figueras Vocal 1er

Carme Torras

Carme Torras

Vocal 2on

Elena Valderrama

Elena Valderrama Vocal 3er

Barcelona, 12 de Julio de 1994

Al Sergi

Índex

Llista de Figures

Llista de Taules

Llista d'Algorismes

Resum

A mesura que augmenta el nombre de transistors integrables en un xip, problemes com el desfasament del senyal de rellotge esdevenen cada cop més crítics. Altres avantatges com un consum més baix, una velocitat mitjana de càlcul, un disseny modular o l'adaptació a les constants físiques avalen la realització de circuits asíncrons.

Però els circuits asíncrons tenen un principal inconvenient que rau en la complexitat del seu disseny. Per tal que un circuit asíncron funcioni correctament cal garantir que sigui lliure de riscos i curses, i aquesta no és una tasca fàcil de realitzar.

En aquest treball es presenta una metodologia de síntesi d'alt nivell de circuits asíncrons. La síntesi d'alt nivell té com a objectiu generar descripcions estructurals d'un circuit a partir d'una descripció del seu comportament. La síntesi d'alt nivell és un tema de recerca molt actiu des de l'última dècada, però mai ha estat aplicada al disseny de circuits asíncrons.

Les principals contribucions d'aquest treball són: un model d'arquitectura asíncrona per a la síntesi d'alt nivell de circuits asíncrons, diversos algorismes de planificació i d'assignació per fer síntesi d'alt nivell de circuits asíncrons i un exemple de disseny on es valida la metodologia de disseny proposada.

El model d'arquitectura asíncrona proposat consisteix en un sistema multiprocessador on cada processador té una doble component. D'una banda hi ha el component de càlcul format per elements del camí de dades autotemporitzats. D'altra banda hi ha el controlador local, que se sincronitza amb els elements del càlcul del processador i amb altres controladors. Així doncs, el control està totalment distribuït i la comunicació entre processadors és totalment asíncrona. El comportament dels controladors locals es descriu mitjançant grafs de transició de senyals (STG). A partir de descripcions en STG es poden obtenir circuits asíncrons lliures de riscos amb eines de síntesi existents.

Es presenta una metodologia de planificació d'operacions per a una arquitectura asíncrona basada en llistes d'esdeveniments. Aquesta metodologia es concreta en dos algorismes: *ELS* i *ELLAS*. Ambdós algorismes tenen complexitat polinòmica. Els resultats obtinguts per aquests algorismes són millors o iguals als obtinguts pels algorismes de planificació d'operacions més reconeguts.

En el treball es proposa un algorisme d'associcació de recursos basat en la teoria de grafs. La contribució més important d'aquesta part del treball és la representació de tots els elements a associar en un únic graf de compatibilitat, de manera que totes les tasques d'associació es poden realitzar simultàniament.

També es presenta un algorisme de planificació d'operacions i assignació de recursos. Les dues fases de la síntesi d'alt nivell es realitzen de manera simultània mitjançant un algorisme basat en la tècnica de recuit simulat. El model d'execució utilitzat permet l'encadenament d'operacions per tal de minimitzar el temps d'execució.

Finalment, es presenta un exemple de disseny d'un circuit asíncron utilitzant la metodologia de disseny proposada en el treball. En ell es demostra que les tècniques proposades permeten obtenir dissenys de circuits asíncrons de manera automàtica. El disseny s'ha comparat amb una realització síncrona del mateix circuit. El resultat d'aquesta comparació mostra que els circuits asíncrons encara no són prou ràpids, tot i que segurament en un futur pròxim ho poguin ser si s'apliquen tècniques per augmentar la velocitat dels components del camí de dades.

Agraïments

Primerament, vull agrair al meu director de tesi, el doctor Jordi Cortadella, tot el seu ajut durant aquests últims anys. Sense les seves orientacions, idees i la seva accessibilitat des del primer a l'últim dia aquest treball no hagués estat possible. També vull agrair als meus companys Enric Pastor i Oriol Roig la seva colaboració i suggerències en diversos aspectes d'aquest treball. I a tot el grup de recerca, gràcies pel seu suport.

El meu agraïment al doctor Mateo Valero i al doctor Josep M. Llaberia, i a tots els membres del Departament d'Arquitectura de Computadors que s'han interesat pel meu treball, així com al doctor Jordi Torres, pel seu encoratjament i comprensió.

Vull agrair a la Comissió de Doctorat del Departament d'Arquitectura de Computadors i als altres membres del tribunal de la pre-lectura de la tesi totes les seves preguntes i suggeriments que sens dubte han ajudat a millorar alguns aspectes d'aquesta tesi.

Finalment, la meva gratitud al meu pare i a la meva mare, als meus germans i a la meva familia, i al Sergi. Sense la seva paciència, el seu suport i el seu amor, el treball hagués resultat molt més ingrat.

Aquest treball ha estat subvencionat pel Ministeri d'Educació i Ciència (TIC 0300/89, TIC 1036/91) i per la Comunitat Europea (ACiD-WG Esprit 7225).

Capítol 1

Introducció

L'objectiu d'aquesta tesi és el desenvolupament d'una metodologia de síntesi d'alt nivell de circuits asíncrons. En aquest capítol es presenten les motivacions bàsiques del nostre interès cap a l'automatització del disseny de circuits asíncrons, un resum del treball que s'ha realitzat i l'estructura de la tesi.

1.1 Circuits síncrons i asíncrons

La presència o absència de rellotge determina clarament dues alternatives en el disseny de circuits. Els circuits síncrons es caracteritzen per l'existència del senyal de rellotge. El senyal de rellotge té una determinada freqüència i determina la seqüència d'execució de les operacions, alhora que serveix com a referència de temps. L'inici de les operacions és determinat pel flanc de rellotge i la duració de les operacions és un múltiple del temps de cicle.

En canvi, en els circuits asíncrons no existeix cap rellotge que marqui el pas de les operacions. Les diferents operacions es van executant una després de l'altra en una seqüència prefixada i el final d'una operació determina el principi de la següent [Sei80]. La duració de les operacions depèn les dades d'entrada i el que cal garantir és que s'executin en un determinat ordre. Cap esdeveniment no ha d'ocórrer en un temps determinat, sinó després d'un altre o altres, de la mateixa manera que diferents accions se succeeixen una després d'una altra en una cadena de muntatge.

Si bé el disseny de circuits asíncrons ha estat un tema d'estudi durant les últimes dècades, no ha estat fins recentment que els avenços en la tecnologia han permès realitzar circuits asíncrons més fàcilment. Les raons per les quals el disseny dels circuits asíncrons no era factible amb la tecnologia dels anys 60 o 70 eren:

1. D'una banda, que els circuits asíncrons utilitzen més transistors que els síncrons per a un circuit equivalent, a causa del control i de la lògica utilitzada per fer els blocs combinacionals.

2. D'altra banda, que el disseny dels circuits asíncrons és molt més complex que el dels circuits síncrons. Existeixen problemes com els riscos[1] o les curses[2] [Moo92] que calen ser evitats i, per tant, el procés de disseny és més costós i llarg.

Les raons que fan creure que els circuits asíncrons seran molt utilitzats en un futur proper són:

[1]Trad. de *hazards*

[2]Trad. de *races*

- A mesura que augmenta el nombre de transistors integrables en un xip, el primer dels inconvenients dels circuits asíncrons va esdevenint menys important.
- Problemes com el desfasament del rellotge es van agreujant a mesura que escalem la tecnologia i limiten els circuits síncrons.
- Altres avantatges dels circuits asíncrons com:
  - Temps mig de càlcul enfront d'un retard màxim dels circuits síncrons.
  - Consum de potència menor, que els fa atractius per a dispositius portàtils.
  - Gran modularitat en el disseny

Per aquestes raons, l'automatització del disseny de circuits asíncrons és un tema important de recerca. L'objectiu d'aquest treball és dissenyar eines de síntesi automàtica de circuits asíncrons.

1.2 Disseny de circuits

Durant les últimes tres dècades s'ha plantejat la necessitat d'automatitzar la tasca de disseny dels circuits. Inicialment tot el procés de disseny es realitzava per dissenyadors experts que afrontaven la difícil tasca de realitzar totes les fases de disseny del xip. Dos dels defectes d'aquest procés de disseny realitzat de manera manual eren:

- La seva lentitud.
- Els possibles errors humans introduïts que calia que fossin detectats el més aviat possible per no encarir el procés de disseny.

L'automatització del disseny és important ja que, d'una banda, redueix el cicle de disseny i, de l'altra, permet explorar l'espai de disseny fins que s'obté la configuració més adient en cada cas. Però l'automatització també té un preu, ja que els dissenys obtinguts són sovint més lents o ocupen més àrea.

Sembla important, doncs, obtenir eines tals que, donada una especificació del comportament del circuit desitjat i un conjunt de restriccions de disseny, com ara l'àrea màxima o

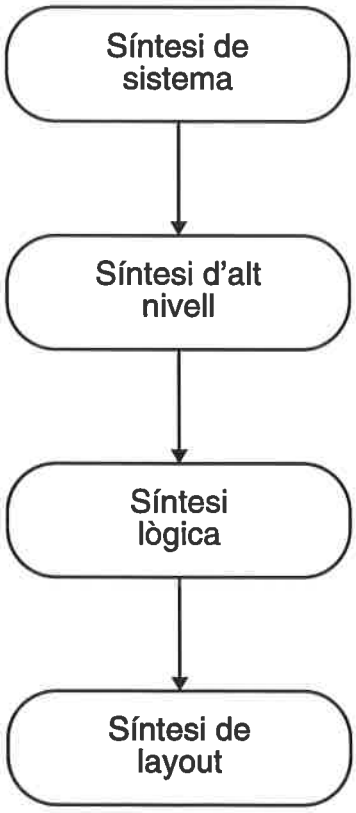


Figura 1.1: Esquema de les fases del procés de disseny d'un circuit

la latència màxima del circuit, generin una estructura que implementi el circuit especificat sota les restriccions prefixades.

A la figura 1.1 es pot veure un esquema de les diferents fases dins del disseny d'un circuit. Com es veurà a continuació, les fases inferiors d'aquest procés estan en un nivell d'automatització molt alt, mentre que les fases superiors encara són un tema punter de recerca.

La primera fase és la síntesi de sistemes. En aquest nivell es prenen decisions com establir el particionat, descriure l'especificació del sistema en diferents xips o separar l'execució en diferents etapes d'un *pipeline*. En la major.a de sistemes, aquesta etapa encara es realitza manualment.

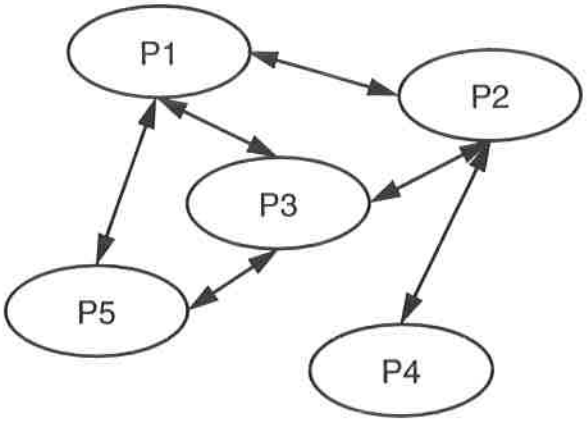
La següent fase és la síntesi d'alt nivell, i l'objectiu és obtenir una descripció estructural d'un circuit a partir d'una descripció del seu comportament. No es tracta d'una simple traducció, sinó que es pretén que s'optimitzi la qualitat del circuit. Com a resultat d'aquesta etapa, s'obté una descripció de la xarxa de mòduls que compondran el camí de dades del circuit i una descripció del control. La síntesi d'alt nivell ha estat tema de recerca durant els darrers 20 anys, però el seu àmbit s'ha restringit als circuits síncrons.

A continuació trobem la síntesi lògica que fa una correspondència entre equacions booleanes i la tecnologia. Aquestes equacions booleanes descriuen el control o els mòduls del camí de dades. En aquesta fase tenen lloc tasques com la minimització d'estats, la codificació d'estats i la projecció a una tecnologia. La síntesi lògica ha estat un tema de recerca intens durant els darrers 30 anys i podem dir que ha arribat a un nivell estable en el qual s'ofereixen eines d'alta qualitat al mercat.

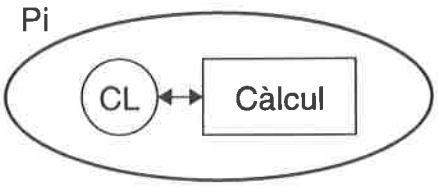
L'última fase d'aquest procés de disseny és la síntesi de layout on s'obté la descripció física del circuit. En aquesta fase es porten a terme tasques com l'escalat dels transistors, la distribució del rellotge, l'emplaçament dels mòduls en el circuit i el connexionat.

1.3 Resum del treball realitzat

El principal objectiu del nostre treball és desenvolupar un entorn de síntesi automàtica per al disseny de circuits asíncrons, és a dir, oferir a la comunitat eines per a l'automatització del disseny de circuits asíncrons.



(a)



CL = Controlador Local

(b)

Figura 1.2: (a) Model d'arquitectura asíncrona basat en un sistema multiprocessador; (b) Cada processador del sistema està compost per un component de càlcul i un controlador local

A [CB92] es presenta un model d'arquitectura asíncrona basat en un sistema multiprocessador totalment asíncron amb pas de missatges. En aquest sistema, cada processador està format per dos components: un component de càlcul i un component de control local. Entre els diferents processadors que formen el sistema, la comunicació és totalment asíncrona. La figura 1.2 mostra un esquema d'aquesta arquitectura asíncrona.

La figura 1.3 mostra l'esquema d'un sistema de síntesi de circuits asíncrons basat en aquest model d'arquitectura. Aquest sistema es compon d'eines de síntesi d'alt nivell i d'eines de síntesi lògica.

El punt de partida és una descripció del comportament del circuit que es vol sintetitzar. Aquesta descripció pot estar escrita en un llenguatge de descripció de harware (HDL). Sobre

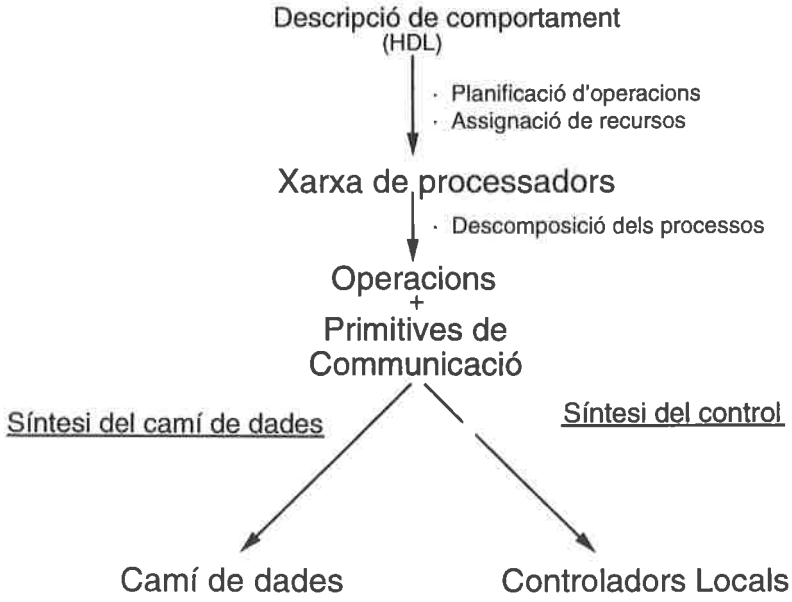


Figura 1.3: Síntesi automàtica de circuits asíncrons

aquesta descripció s'apliquen les fases de planificació d'operacions i assignació de recursos, que són les fases principals de la síntesi d'alt nivell. Com a resultat, s'obté una descripció de la xarxa de processadors que compondran el sistema i de les operacions que es realitzen en cadascun d'ells i en quin ordre.

Anomenarem *procés* al conjunt d'operacions a executar en un processador. Cadascun dels processos es pot descompondre en operacions de càlcul i en primitives de comunicació amb els altres processadors.

A partir d'aquest punt el procés de síntesi es descompon en dos: d'una banda, tenim la síntesi del camí de dades i, de l'altra, la síntesi del control.

La contribució més important d'aquesta tesi és la proposta d'una metodologia de síntesi automàtica de circuits sobre un model d'arquitectura asíncrona. Per tal d'acomplir aquest objectiu es presenta un mòdel d'arquitectura asíncrona basat en un sistema multiprocessador totalment asíncron. Es presenten també diferents algorismes de síntesi d'alt nivell per a circuits asíncrons. Concretament es proposa un algorisme de planificació d'operacions, un algorisme d'assignació de recursos i un algorisme on es realitza de forma simultània la planificació d'operacions i l'assignació de recursos. Finalment, també es proposa una metodologia per a la síntesi del control.

1.4 Estructura del document

Al capítol 2 es fa una introducció a la síntesi d'alt nivell i a les seves etapes més importants: la planificació d'operacions i l'assignació de recursos. Per cadascuna d'aquestes etapes, es defineix el problema a resoldre, es dóna una classificació dels algorismes existents a la literatura i es descriuen alguns dels més rellevants.

Al capítol 3 es parla de circuits asíncrons, dels seus avantatges i inconvenients.. Es descriuen les diferents alternatives existents per construir mòduls autotemporitzats i es fa una revisió de les diferents propostes existe ts per fer síntesi de circuits de control asíncrons. Es descriuen les característiques dels mòduls autotemporitzats amb lógica DCVSL i com es poden encadenar. Finalment, es presenta una revisió de diferents implementacions de circuits asíncrons que s'han descrit a la literatura.

Al capítol 4 es proposa un model d'arquitectura asíncrona basat en mòduls autotemporitzats i control totalment distribuït. Els protocols de senyals per controlar el camí de dades del circuit es descriuen amb grafs de transicions de senyals (STG). D'aquesta descripció es pot sintetitzar el control mitjançant tècniques existents. Es presenten les diferents propostes existents per sintetitzar circuits asíncrons a partir d'STG. Es proposa una metodologia per obtenir els STG que descriuen el control per al model d'arquitectura presentat.

El capítol 5 té com a objectiu definir conceptes bàsics per realitzar la planificació d'operacions en un sistema de síntesi d'alt nivell de circuits asíncrons. Es presenten les estructures de dades i operacions bàsiques que seran necessàries. Es proposen dues estratègies de planificació d'operacions per a sistemes asíncrons basats en llistes d'esdeveniments: *Event-List Scheduling (ELS)* i *Event-List Look-Ahead Scheduling (ELLAS).*

Al capítol 6 es presenta *GLASS*, una nova formulació basada en la teoria de grafs del problema d'assignació de recursos on les tres tasques d'assignació (assignació d'unitats funcionals, assignació de registres i assignació d'unitats d'interconnexió) són executades simultàniament. L'algorisme no es basa en el temps de cicle per fer l'anàlisi dels temps de vida de les variables i operacions, i per tant es pot utilitzar aquesta estratègia tant per fer síntesi de circuits síncrons com per fer síntesi de circuits asíncrons.

Al capítol 7 es presenta una estratègia per planificar operacions i assignar recursos de

forma simultània per fer síntesi d'alt nivell de circuits asíncrons. L'algorisme està basat en la tècnica d'*escalada de cims* anomenada *recuit simulat*[3]. Aquest algorisme permet encadenar operacions i, a l'igual que els algorismes presentats al capítol 5, està basat en llistes d'esdeveniments.

Al capítol 8 es presenta la realització d'un exemple complet utilitzant el model d'arquitectura presentat al capítol 4 i l'algorisme de planificació i assignació descrit al capítol 7 . Per fer aquest disseny, s'ha utilitzat l'eina de disseny de layout OCEAN [GS93].

Finalment al capítol 9 es presenten les conclusions i línies obertes de la tesi.

[3]Trad. de *simulated annealing*

Capítol 2

Síntesi d'Alt Nivell

La síntesi d'alt nivell o síntesi de comportament té com a objectiu obtenir una descripció estructural d'un sistema a partir de la seva descripció de comportament. En aquest capítol es descriuen les diferents fases que formen part d'aquest procés de síntesi i es fa especial èmfasi en les més importants: la planificació d'operacions i l'assignació de recursos.

2.1 Introducció

Les transformacions que efectua la síntesi d'alt nivell per passar d'un nivell d'especificació a un altre no és una simple traducció, sinó que comporta una exploració de l'espai de disseny en busca de les millors configuracions. Aquesta recerca no és trivial, i és per això que s'ha descompost en diferents fases, cadascuna d'elles prou complexa [MPC90]. La figura 2.1 mostra les diferents fases en què es pot descompondre la síntesi d'alt nivell i els resultats que s'obtenen en portar-les a terme.

A continuació, es descriu cadascuna de les fases:

1. **Especificació:** la primera fase especifica el comportament del circuit que es vol sintetitzar en un llenguatge de descripció de hardware com ara VHDL [CST91], HardwareC [KM92], ELLA [Mor84], etc. Aquesta descripció ha de descriure amb detall les entrades i sortides del sistema i la relació que ha d'haver-hi entre elles.

2. **Compilació:** d'aquesta descripció algorítmica es passa a una descripció en forma de graf dirigit mitjançant una fase de traducció. En fer aquesta traducció, és freqüent que s'apliquin dos tipus d'optimitzacions:

   - Optimitzacions típiques de compilació de llenguatges [ASU86]: propagació de constants, eliminació de codi mort, expansió de subprogrames...
   - Optimitzacions més específiques de la síntesi d'alt nivell: transformacions específiques dels recursos (per exemple, substituir una multiplicació per una potència de dos per un desplaçament realitzat seleccionant els bits apropiats), incrementar el nivell de paral·lelisme dels operadors i altres.

   El graf que s'obté de la fase de traducció és el *graf de flux de dades* $DFG = (V, A)$ on cada vèrtex $v \in V$ representa una operació i existeix un arc $(u, v) \in A$, si entre u i v existeix una dependència de dades. Si, a més de dependències de dades, el conjunt d'arcs també representa dependències de control, direm que el graf és un *graf de flux de dades i de control (CDFG)*. La figura 2.2 mostra un exemple de traducció d'una descripció de comportament a un *DFG*.

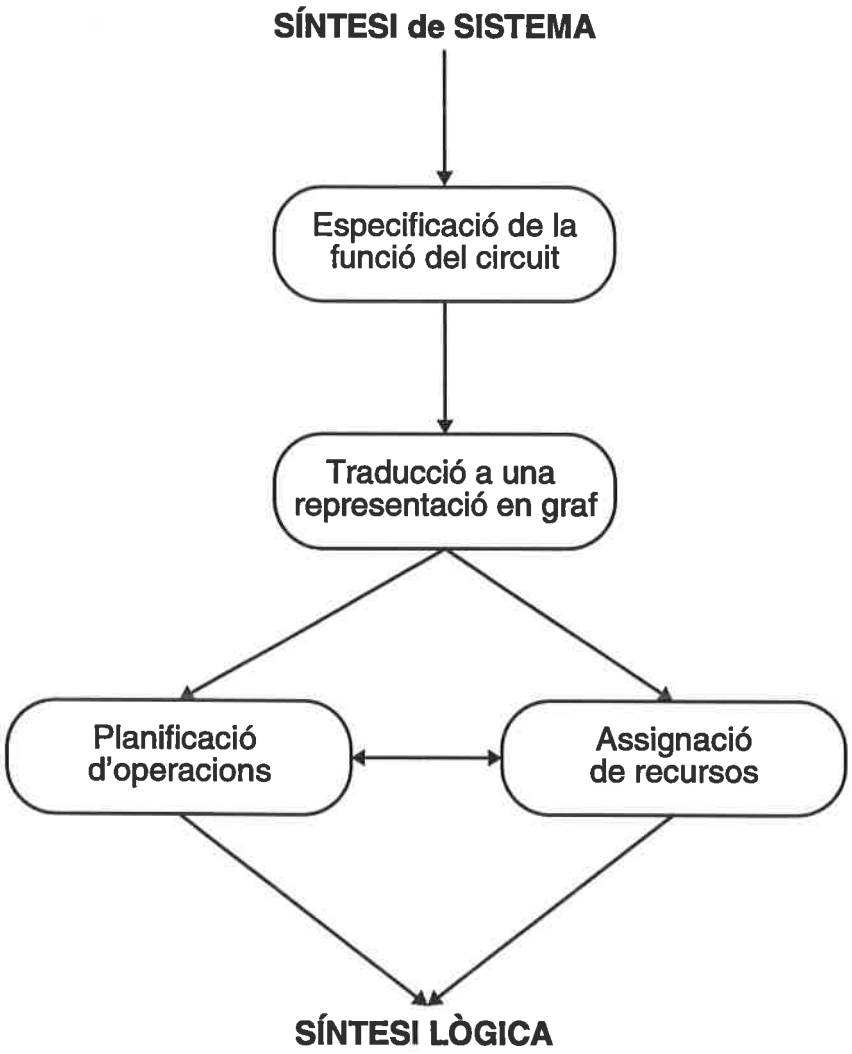


Figura 2.1: Esquema de les fases del procés de síntesi d'alt nivell

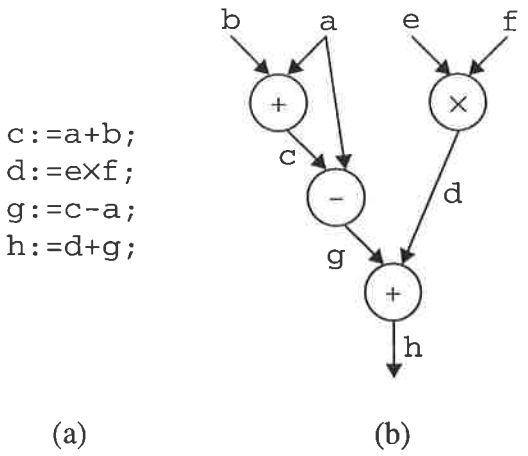


Figura 2.2: (a) Descripció de comportament; (b) graf de flux de dades

A més del graf, hi ha altres dades d'entrada que normalment s'utilitzen a la síntesi d'alt nivell, com per exemple la *llibreria de recursos* i *restriccions* al disseny desitjat.

A la llibreria de recursos es descriuen els diferents components disponibles. Cada component es caracteritza per paràmetres com:

- Àrea que ocupa.
- Llista d'operacions que pot executar.
- Retard per executar cadascuna de les operacions.

Les restriccions més habituals són les d'àrea i temps. Les restriccions d'àrea es poden descriure en termes d'àrea de layout o de nombre de recursos de la llibreria disponibles. Les restriccions de temps es poden donar en nombre de cicles o en temps total d'execució.

3. **Planificació d'operacions i assignació de recursos:** després de la traducció es realitzen les fases més importants de la síntesi d'alt nivell: la *planificació d'operacions*[1] i l'*assignació de recursos*[2]. Mentre la planificació d'operacions distribueix l'execució

[1]Traducció del terme anglès *operation scheduling*

[2]Traducció del terme anglès *hardware allocation*

de les operacions del graf en el temps, l'assignació de recursos associa les operacions a recursos (unitats funcionals, registres...).

La planificació d'operacions i l'assignació de recursos són dues tasques interdependents que poden realitzarse una abans de l'altra o simultàniament. Així per exemple, en sistemes com FACET [TS86], es realitza primer la planificació d'operacions i després l'assignació de recursos. En sistemes com CADDY/DSL [CR89], s'executa primer l'assignació de recursos i després la planificació d'operacions. D'altra banda, en sistemes com SLICER/SPLICER [PG87a] o SCHALLOC[BP90], les dues fases, planificació d'operacions i assignació de recursos, són portades a terme a la vegada.

Després d'aquestes fases, tal com hem vist al capítol 1, podem passar al següent nivell en el procés de síntesi que seria el de síntesi lògica.

La síntesi d'alt nivell obté com a resultat una descripció del camí de dades del circuit i una descripció del seu control. El camí de dades consisteix en un conjunt de components de la llibreria de recursos —unitats funcionals, multiplexors, registres, busos— interconnectats entre si. Segons el model d'arquitectura de camí de dades que es consideri, es pot simplificar o complicar la tasca de síntesi. Per exemple, si el model d'arquitectura considerat és molt senzill, el conjunt de dissenys alternatius es més reduït i per tant se simplifica la tasca de síntesi.

Un dels factors que té una influència més important en l'eficiència del camí de dades és la topologia d'interconnexió. La topologia d'interconnexió defineix com es suportaran les transferències de dades entre els diferents components del camí de dades. Diferenciarem dos models d'arquitectura:

Model d'arquitectura orientat al multiplexor: quan s'utilizen multiplexors en les transferències de dades.

Model d'arquitectura orientat al bus: quan s'utilitzen busos en les transferències de dades.

La figura 2.3.a mostra un exemple d'un camí de dades utilitzant multiplexors i la figura 2.3.b mostra un exemple de camí de dades utilitzant busos per a les interconnexions. En un camí

de dades poden coexistir els dos models d'arquitectura, de manera que el connexionat es realitza amb busos i multiplexors.

A la següent secció es defineixen alguns termes que seran utilitzats a la resta del capítol. A la secció 2.3 i 2.4 es parla amb més profunditat de les fases de planificació d'operacions i d'assignació de recursos. Finalment, es presenten algunes conclusions.

2.2 Definicions

En aquesta secció es descriuen alguns conceptes que seran utilitzats en aquest capítol i en capítols posteriors.

2.2.1 Graf de flux de dades

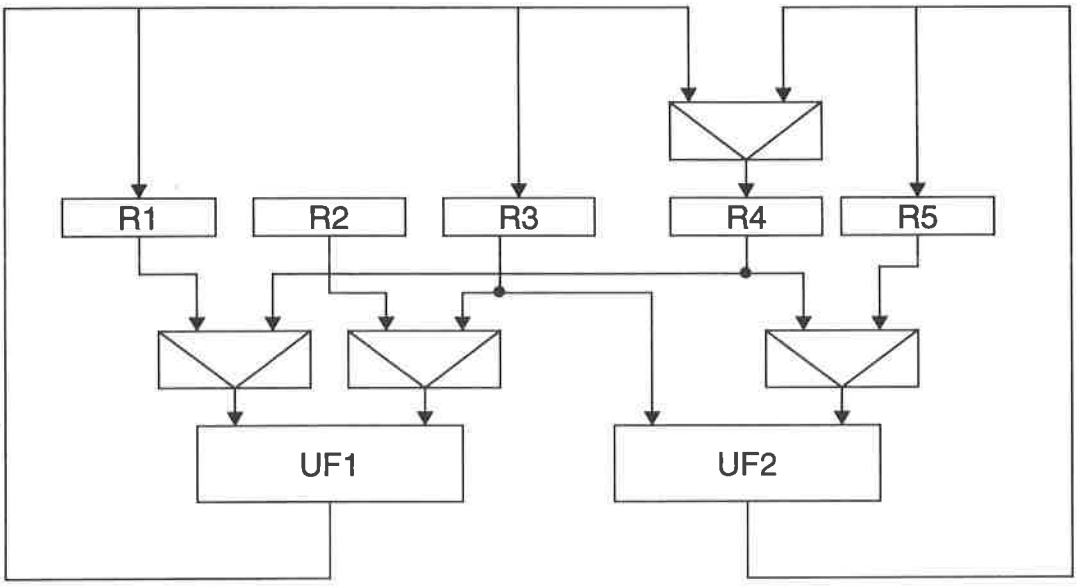
El graf de flux de dades $DFG = (V, A)$ és un graf on cada vèrtex $v \in V$ representa una operació a executar i existeix un arc entre dos vèrtexs si hi ha una dependència de dades entre les operacions representades pels vèrtexs. Si, a més de dependències de dades, el conjunt d'arcs representa dependències de control, parlarem d'un graf de flux de dades i control (CDFG).

2.2.1.1 Vèrtexs font i destí

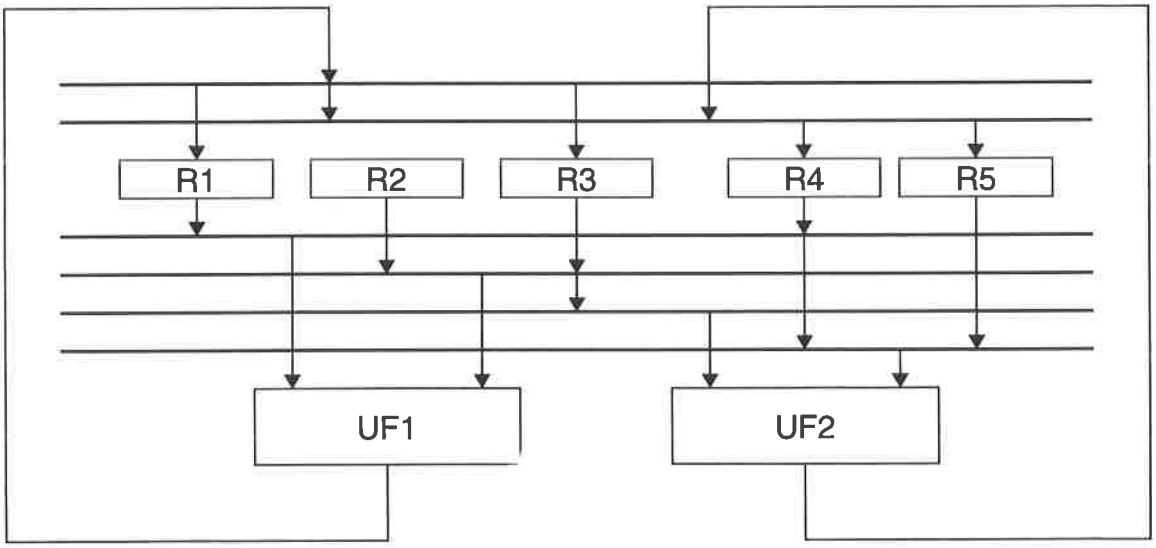
Anomenarem vèrtexs *font* del DFG o CDFG aquells que no tinguin cap predecessor en el graf i anomenarem vèrtexs *destí* aquells que no tinguin cap successor en el graf.

2.2.1.2 Graf de flux de dades planificat

Un cop s'han efectuat les fases de planificació d'operacions i assignació de recursos sobre el graf de flux de dades, anomenarem al graf resultant *graf de flux de dades planificat (SDFG)*. El SDFG és un graf isomorf al DFG on cada vèrtex $v \in V$ continua representant una operació, però ara conté informació sobre la planificació i assignació de recursos.



(a)



(b)

Figura 2.3: (a) Camí de dades amb arquitectura orientada a multiplexors; (b) Camí de dades amb arquitectura orientada a busos

2.3 Planificació d'operacions

En aquesta secció parlarem de la fase de planificació d'operacions dins de la síntesi d'alt nivell. Primerament es defineix el problema a resoldre, després es fa una classificació dels algorismes existents a la literatura i es presenten alguns dels algorismes de planificació més rellevants.

2.3.1 Definició formal del problema

La planificació d'operacions té com a objectiu resoldre el problema d'executar una sèrie de tasques utilitzant un conjunt de recursos, sota una conjunt de restriccions, ja siguin d'àrea o de temps. L'objectiu és minimitzar el temps total o el cost en recursos, depenent del tipus de restriccions d'entrada. Entre els diferents problemes de planificació, el de *planificació amb precedències restringides* [GJ79] és el que més s'ajusta a la planificació d'operacions dins de la síntesi d'alt nivell.

Definició 2.1 Planificació amb precedències restringides [GJ79]

Donat un conjunt de tasques T on cada tasca t té duració 1, un ordre parcial $\prec$ dins de T, un nombre $m \in Z^+$ de processadors, i un temps total $D \in Z^+$, la planificació amb precedències restringides *consisteix a poder contestar amb un* sí *o amb un* no *la següent pregunta: Existeix una planificació $\sigma : T \rightarrow \{0, 1, ..., D\}$ tal que*

$$| \{t \in T : \sigma(t) = s\} | \leq m \ \forall s \in \{0, 1, ..., D\} \quad \wedge \quad t_i \prec t_j \Rightarrow \sigma(t_i) < \sigma(t_j) \ ?$$

Aquest problema es pot relacionar amb la planificació d'operacions dins de la síntesi d'alt nivell si s'identifiquen tasques amb operacions, la duració de les tasques amb el retard de les operacions, els processadors amb les unitats funcionals, el temps total amb el nombre total de cicles necessaris i l'ordre parcial amb l'ordre definit pel graf de flux de dades.

Està demostrat que la planificació amb precedències restringides és un problema NP-complet [GJ79]. La definició 2.1 planteja una pregunta que pot ser resposta amb un "sí" o amb un "no". Aquest tipus de problemes són anomenats problemes de *decisió*. Si considerem el problema d'*optimització*, en lloc d'un problema NP-complet tenim un problema NP-hard. Per tant, com que no existeixen solucions executables en temps polinòmic,

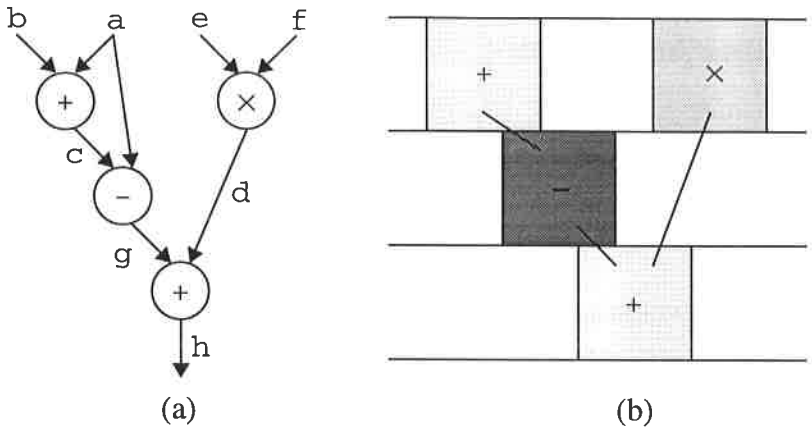


Figura 2.4: (a) Graf de flux de dades; (b) Planificació d'operacions per al graf de flux de dades anterior

haurem de proposar heurístiques que ens permetin solucionar el problema de planificació d'operacions amb grafs grans. Normalment es tracta un problema més general en què la duració de les tasques és més gran que 1.

La figura 2.4.b mostra un exemple de planificació d'operacions per al graf de flux de dades de la figura 2.4.a.

2.3.2 Classificació

Una primera classificació dels algorismes de planificació d'operacions podria obtenir-se segons si intenten optimitzar temps o espai. L'espai es pot mesurar en funció del nombre de recursos necessaris (unitats funcionals, registres, multiplexors...), en funció del nombre de portes necessàries o en funció de l'àrea de layout del xip. El temps, en el cas de disseny de circuits síncrons, es pot mesurar com el nombre de cicles necessaris per executar el graf de flux per a un disseny determinat.

Parlarem de *planificació amb restricció de recursos* si el que s'intenta és optimitzar el temps donada una restricció de recursos. Parlarem de *planificació amb restricció de temps* si s'intenta optimitzar l'espai donat un temps màxim d'execució. Exemples de planificació amb restricció de recursos els podem trobar als algorismes de *planificació per*

llistes[3] [DLSM81], CADDY [GKR91] i de planificació amb restricció de temps a l'algorisme de *planificació dirigida per forces*[4] [PK89a].

També hi ha algorismes que no fan cap tipus de restricció i que intenten optimitzar àrea i espai a la vegada [CBA91a].

D'altra banda també podem classificar els algorismes segons la seva naturalesa.

- *Algorismes transformacionals:* els algorismes transformacionals parteixen d'una configuració inicial i a cada iteració apliquen diferents tipus de transformacions per obtenir noves configuracions. Un exemple d'algorisme transformacional el podem trobar a [DN89] on s'aplica la tècnica de *recuit simulat*[5].

- *Algorismes constructius:* són algorismes en què a cada iteració es planifiquen una o diverses operacions (per exemple, totes les que es planifiquen en un cicle). La majoria dels algorismes són iteratius o constructius. Exemples d'aquest tipus d'algorismes els podem trobar a [DLSM81] i a l'algorisme MAHA [PPM86].

Pel que fa a les característiques de les operacions, els algorismes més antics, com el de planificació per llistes [DLSM81] o MAHA [PPM86], consideren que el temps d'execució de les operacions és un cicle. En propostes posteriors, aquesta restricció es relaxa i es permeten operacions multicicle i encadenament d'operacions dins del cicle, com en els algorismes de *planificació dirigida per forces* [PK89a] o [PG87a].

La majoria d'algorismes només consideren la planificació de blocs bàsics. Una excepció és el sistema HYPER, que primer planifica els blocs bàsics però permet correccions sobre aquesta planificació quan es tracta el graf complet. D'altra banda, cal esmentar l'algorisme de *planificació basada en camins*[6] [Cam91] on s'emfasitza el tractament dels salts condicionals, minimitzant el nombre de cicles per tots els camins existents.

Pel que fa als sistemes asíncrons, hem de dir que no coneixem cap sistema què faci planificació d'operacions dins d'un sistema asíncron. L'única proposta en què es considera

[3]Trad. de *list-scheduling*

[4]Trad. de *Force-Directed Scheduling*

[5]Trad. de *simulated annealing*

[6]Traducció del terme anglès *path-based scheduling*

el fet que hi hagi algunes operacions on el temps d'execució no està fixat és en l'algorisme de *planificació relativa*[7] [KM90].

2.3.3 Exemples

A continuació es descriuran cinc algorismes de planificació d'operacions: algorismes ALAP i ASAP, *planificació per llistes* [DLSM81], *planificació dirigida per forces* [PK89a] i *planificació relativa* [KM90]. Primerament es presenten els algorismes ASAP i ALAP, ja que són utilitzats frenqüentment com a base en altres algorismes. A continuació es descriuen els algorismes de planificació per llistes per la seva simplicitat i importància històrica. El segon és un dels algorismes de planificació que dóna millors resultats per la seva complexitat. Per últim, es presenta l'únic algorisme conegut per nosaltres on es té en compte la possibilitat que hi hagi operacions o esdeveniments asíncrons.

2.3.3.1 Planificació ASAP i ALAP

Els algorismes ASAP i ALAP són dos exemples senzills de planificació d'operacions, on el nombre de recursos disponibles és il·limitat i es restringeix el nombre de cicles de la planificació a la profunditat del camí crític del *DFG*.

Les sigles ASAP provenen de l'anglès *As Soon As Possible*, és a dir, el més aviat possible. L'algorisme ASAP planifica cada operació en el primer cicle en què les dependències de dades ho facin possible [MLD92].

En canvi, l'algorisme ALAP (*As Late As Possible*), planifica les operacions en el últim cicle possible, sempre que el nombre total de cicles no excedeixi la profunditat del camí crític del *DFG*.

L'algorisme ASAP es va utilitzar en algunes propostes inicials, però normalment, tant l'algorisme ASAP com l'ALAP, són utilitzats per calcular paràmetres que seran utilitzats després en un algorisme de planificació més complex.

La figures 2.5.a i 2.5.b mostren un exemple de planificacions ASAP i ALAP.

[7] Traducció del terme anglès *relative scheduling*

planificació per llistes (G){
insertar operacions preparades $(G, L_{FU_0}, L_{FU_1}, ...L_{FU_m})$;
$cicle = 0$;
mentre $L_{FU_0} \neq \emptyset$ *i* $L_{FU_1} \neq \emptyset$ *i* $...L_{FU_N} \neq \emptyset$ **fer**
per $k = 1$ **fins a** m **fer**
per $uf = 1$ **fins a** N_k **fer**
si $L_{FU_k} \neq \emptyset$ **llavors**
planificar operació $(primera(L_{FU_k}, cicle))$;
$L_{FU_k} = esborrar\ (L_{FU_k}, primera(L_{FU_k})$;
fisi
fiper
fiper
insertar operacions preparades $(G, L_{FU_0}, L_{FU_1}, ...L_{FU_N})$;
$cicle = cicle + 1$;
fimentre
}

Algorisme 2.1: Algorisme de planificació per llistes

2.3.3.2 Planificació per llistes

La planificació d'operacions per llistes és un dels mètodes amb restricció de recursos més popular. Aquest algorisme manté una llista d'operacions *preparades*. Es considera que una operació està *preparada* si és una operació *font* o si totes les operacions que li són predecessores en el graf ja estan planificades. En cada iteració i de l'algorisme es planifiquen algunes operacions de la llista de preparades en el cicle i. Per triar les operacions en cada cicle es fa servir una funció de prioritat. Així doncs, les operacions més prioritàries seran planificades en el cicle actual i la resta seran postposades a cicles posteriors. En cada iteració s'ha d'actualitzar la llista de *preparades*.

La qualitat dels resultats obtinguts per aquest algorisme dependrà de la funció de prioritat utilitzada. L'algorisme 2.1 descriu l'algorisme de planificació per llistes.

Atès que hi poden haver diferents tipus d'operacions, la llista d'operacions preparades es descompon en diverses llistes, una per a cada tipus d'operació (L_{FU_i}). En cada cicle es planifiquen per a cada llista (L_{UF_k}) tantes operacions (com a molt) com recursos, que puguin executar aquesta operació, hi hagi (N_k). Es trien aquelles operacions de la llista amb màxima prioritat. Si la llista està ordenada, simplement s'han d'extreure les N_k primeres operacions de la llista.

La funció de prioritat pot ser definida de diferents maneres, i obtenir així resultats diferents. Un exemple és el que pren com a prioritat del vèrtex la *longitud del camí més llarg* des del vèrtex al final del graf. Així, planificarem primer aquelles operacions que tinguin una cadena de successors més llarga, és a dir, aquelles operacions més crítiques, mentre que les que tinguin cadenes de successors curtes, seran postposades als següents cicles [DLSM81].

Una altra és la que es basa en la *mobilitat* de les operacions. La mobilitat de les operacions es defineix com la diferència entre la planificació ALAP i ASAP del vèrtex. La funció de prioritat és inversament proporcional a la mobilitat, assignant a les operacions amb menys mobilitat més prioritat, de tal manera que les operacions del camí crític es planifiquen abans que les altres. Aquesta funció s'utilitza en el sistema SLICER [PG87b].

La complexitat de l'algorisme de planificació per llistes és $O(n)$, on n és el nombre de vèrtexs del *DFG*.

2.3.3.3 Planificació dirigida per forces

El mètode de les forces pot ser utilitzat per fer planificació amb restricció de recursos o per fer planificació amb restricció de temps. Plantejarem primer el cas de la planificació amb restricció de temps per després extendre-ho a la planificació amb restricció de recursos.

Planificació amb restricció de temps (FDS)

En aquest cas, l'objectiu de l'algorisme es minimitzar el nombre de recursos utilitzats (unitats funcionals, registres i busos) per a un temps d'execució màxim (nombre màxim de cicles). L'estratègia es basa en planificar operacions similars en cicles diferents de manera que s'equilibra la concurrència de les operacions que utilitzen el mateix tipus de recurs sense augmentar el temps total d'execució. A més, en equilibrar la concurrència de les

operacions, s'assegura que les unitats funcionals tinguin una alta utilització.

Primer de tot, cal calcular els *límits de temps* de les operacions. Aquest càlcul es fa avaluant les planificacions ASAP i ALAP. La figura 2.5.c mostra els límits de temps per a un exemple. L'amplada dels requadres que contenen cada operació representa la probabilitat que l'operació pugui ésser executada en un cicle determinat. Aquest algorisme suposa que la funció de probabilitat segueix una distribució uniforme i assigna a tots els cicles on l'operació es pot executar la mateixa probabilitat.

A continuació, podem crear el graf de distribució (DG) per a un tipus d'operació o per a un grup de tipus d'operacions. Aquest graf s'obté sumant les probabilitats de les diferents operacions d'un determinat tipus en cada cicle. Per a cada graf de distribució i cada cicle i és calcula:

$$DG(i) = \sum_{tipus\ op} Prob(op, i)$$

on el sumatori s'aplica a totes les operacions d'un mateix tipus o d'un grup de tipus d'operacions (*tipus op*). Per exemple, podem calcular el graf de distribució per a la multiplicació (figura 2.6).

El següent pas és calcular les forces associades a cada assignació d'operació a cicle que siguin factibles. Per a una operació amb límit de temps que vagi del cicle l a l'm la força associada amb la seva assignació al cicle j és:

$$Força(j) = DG(j) - \sum_{i=l}^{m} \left[\frac{DG(i)}{(m-l+1)} \right]$$

És a dir, la força associada amb l'intent d'assignar una operació al cicle j és igual a la diferència entre el valor de la distribució en aquest cicle menys la mitja de la distribució en els cicles que formen part del límit de temps de l'operació.

També es calculen les *forces indirectes* associades als predecessors i successors de l'operació. No es descriu el càlcul d'aquestes forces per no complicar excessivament la descripció del mètode.

Un cop s'han calculat totes les forces per totes les operacions i cicles on poden ser assignades, es tria aquella assignació d'operació a cicle amb força mínima. Es fa la corresponent assignació i es tornen a calcular els límits de temps, graf de distribució i forces. El procés es repeteix fins que no queda cap operació per assignar.

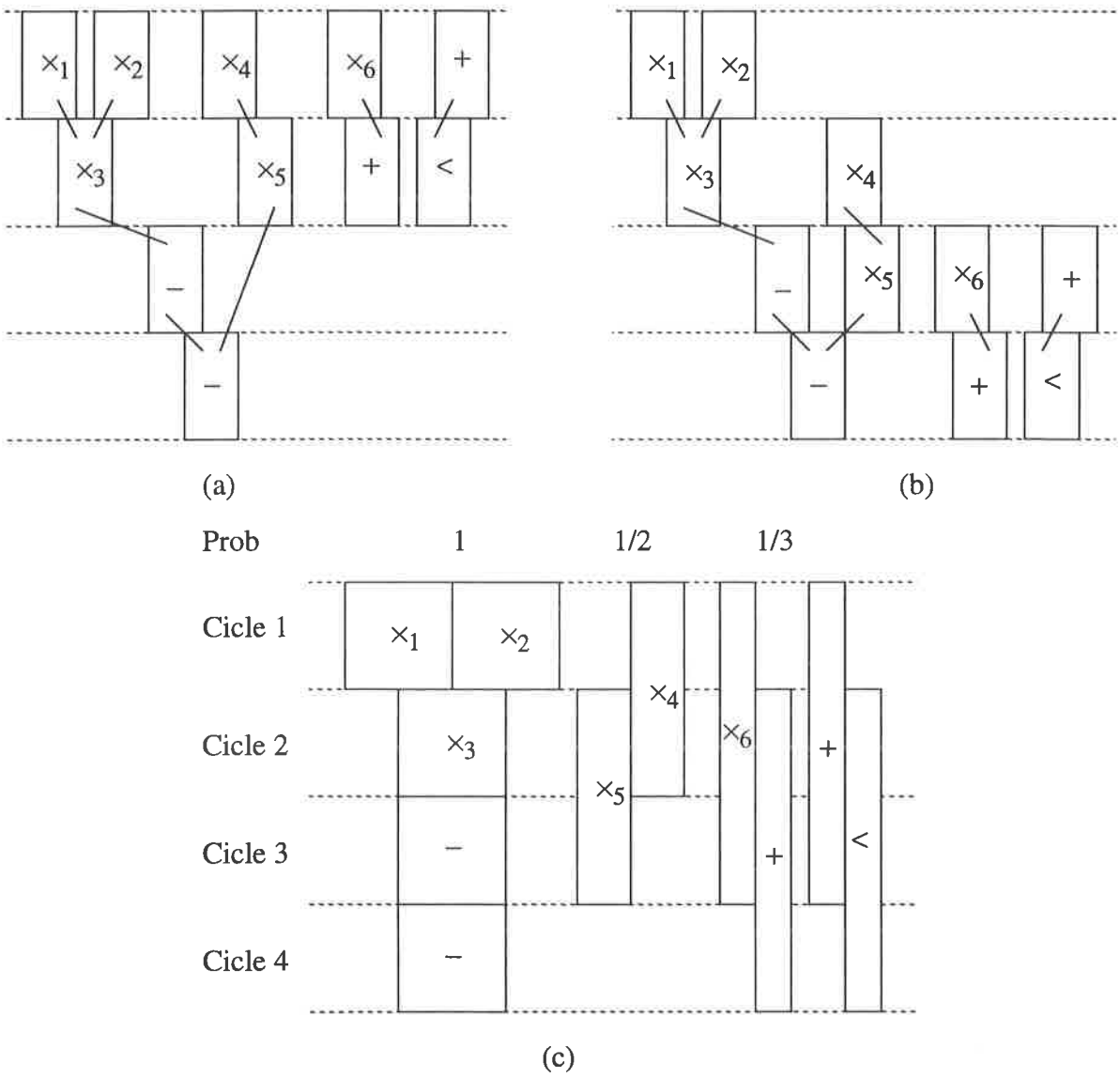


Figura 2.5: (a) Planificació ASAP; (b) Planificació ALAP; (c)Límits de temps per l'exemple anterior

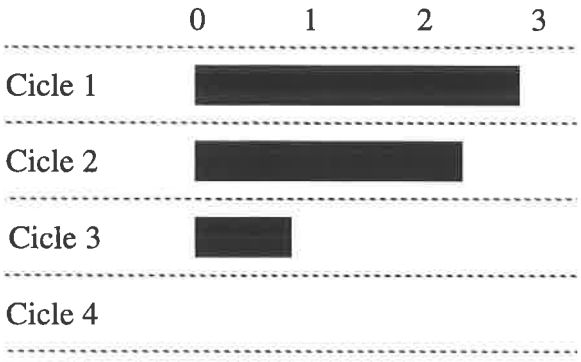


Figura 2.6: Graf de distribució per a la multiplicació

En triar la força amb un valor més petit el que estem fent és balancejar els grafs de distribució, fent que els valors pels diferents cicles s'equilibrin el màxim possible. Això es pot veure a la figura 2.7 on es mostra com es modifica el graf de distribució segons si assignem la operació x_4 de l'exemple anterior al cicle 1 o al cicle 2.

La complexitat de l'algorisme de planificació per forces és de $O(cn^3)$ on c és el nombre de cicles i n el nombre total d'operacions. Aquesta complexitat es pot reduir a $O(n^2)$ aplicant algunes estratègies [PK89a].

Planificació amb restricció de recursos (FDLS)

El mètode de càlcul de les forces també permet planificar les operacions amb restricció de recursos de manera similar al mètode de planificació per llistes, però utilitzant el càlcul de les forces com a funció de prioritat (*planificació per llistes dirigida per forces*[8]).

Com que el càlcul de les forces requereix un temps d'execució màxim, inicialment es fixa aquest temps al determinat pel camí crític. Com a la planificació per llistes, tenim una llista d'operacions preparades. Si totes les operacions *preparades* són del camí crític i no hi ha prou recursos per a totes, caldrà estendre el temps màxim d'execució a un cicle més i tornar a avaluar els límits de temps. Si no podem executar totes les operacions *preparades* en el cicle actual perquè no hi ha prou recursos, es calculen les forces associades al fet de *retardar* les operacions al cicle següent. Això no vol dir que l'operació es planifiqui al cicle següent, sinó que no es planifica en l'actual i que el seu límit de temps es veu reduït.

[8]Traducció del terme anglés *Force-Directed List Scheduling*

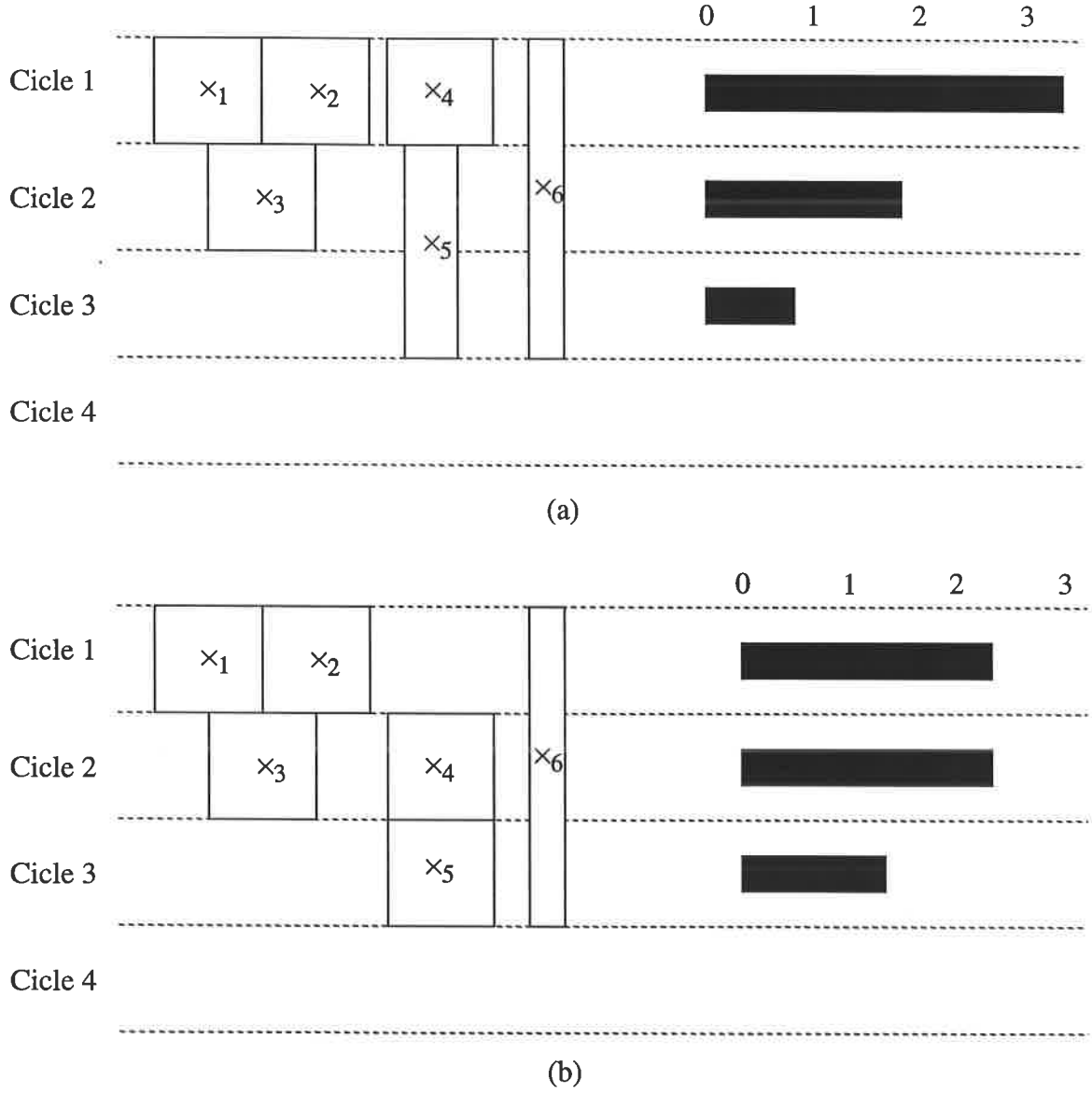


Figura 2.7: (a) Límits de temps i graf de distribució per a les multiplicacions si s'assigna la multiplicació 4 al cicle 1; (b) Límits de temps i graf de distribució per a les multiplicacions si s'assigna la multiplicació 4 al cicle 2

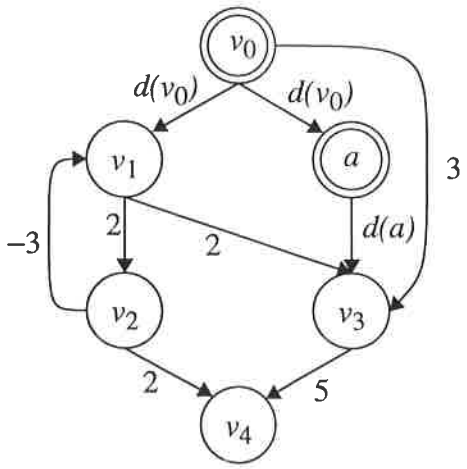


Figura 2.8: Exemple de graf polar

Es *retarden* les operacions amb força mínima i es planifiquen en el cicle actual les altres. Aquest procés es repeteix fins que totes les operacions han estat planificades.

La complexitat d'aquest algorisme és de $O(n^2)$ [PK89a].

2.3.3.4 Planificació relativa

Els algorismes de planificació d'operacions tradicionals suposen que els temps d'execució de les operacions és conegut. Però aquesta suposició no és sempre certa atès que hi ha operacions que poden tenir retards il·limitats. L'algorisme de planificació relativa considera operacions amb retard fix i amb retard il·limitat. Aquest algorisme, a diferència dels anteriors que s'han descrit, suposa que ja s'ha realitzat la fase d'assignació de recursos, de manera que els diferents conflictes que poguessin existir entre operacions assignades al mateix recurs han estat eliminats introduint dependències estructurals entre elles.

L'algorisme parteix d'un *graf polar* $G(V, A)$ amb un únic vèrtex font que és predecessor de tots els altres i un únic vèrtex destí que és successor de tots els altres. Els vèrtexs representen les operacions a planificar i els arcs representen dependències entre elles. Els arcs $(e = (v_i, v_j))$ tenen un pes w_{ij} que equival al retard de l'operació v_i.

La figura 2.8 mostra un exemple de graf utilitzat en aquest sistema. El vèrtex v_0 és el vèrtex font i el vèrtex v_4 és el vèrtex destí. El vèrtex v_1 té un retard de 2 cicles per

executar-se, i per aquest motiu els arcs que van de v_1 a v_2 i v_3 tenen pes 2.

En cas que tots els pesos siguin coneguts, trobar una planificació d'aquest graf equival a trobar un etiquetatge dels vèrtexs $\sigma : V \rightarrow Z^+$ tal que $\sigma(v_j) \geq \sigma(v_i) + w_{ij}$ si existeix un arc entre v_i i v_j amb pes w_{ij}.

L'etiqueta $\sigma(v_i)$ associada al vèrtex v_i representa el temps d'execució respecte a l'inici de la planificació.

L'algorisme contempla que hi pugui haver restriccions de temps. És a dir, permet establir el temps d'execució mínim o màxim d'un vèrtex. Per introduir aquestes restriccions, s'afegeixen arcs al graf i el seu pes es calcula en funció de la restricció de temps que es vol establir. Per exemple, si volem que una operació s'executi com a molt tard 5 cicles després de l'inici de l'execució del graf, cal afegir un arc entre el vèrtex que representa aquesta operació i el vèrtex font amb pes -5. En el cas de la figura 2.8 hi ha un arc amb pes -3 entre v_2 i v_1. Aquest arc indica que v_2 ha d'executar-se com a molt tard 3 cicles després de v_1.

En cas que volguem que una operació s'executi com a molt aviat 3 cicles després d'haver començat l'execució, cal afegir un arc entre el vèrtex font i el vèrtex que representa l'operació amb pes 3. En l'exemple de la figura 2.8 veiem que existeix un arc entre v_0 i v_3 que representa una restricció de temps d'aquest tipus.

L'algorisme permet, com hem dit abans, considerar operacions amb retard il·limitat. Aquestes operacions són anomenades *àncores*. El conjunt d'àncores $A \subseteq V$ d'un graf està format pel vèrtex font v_0 i per totes les operacions amb retard il·limitat. Al graf de la figura tenim dues àncores: els vèrtexs v_0 i a. La planificació quan hi ha àncores en el graf es realitza definint desplaçaments respecte al temps de planificació de l'àncora.

Abans de buscar la planificació relativa, cal assegurar-se que aquesta és factible. Per fer-ho, cal comprovar que totes les restriccions de temps que s'han imposat no depenguin del retard d'execució de cap de les àncores.

La planificació relativa Ω d'un graf $G(V, A)$ és el conjunt de desplaçaments de cada vèrtex $v_i \in V$ respecte a cada àncora del conjunt d'àncores del graf.

2.4 Assignació de recursos

En aquesta secció es presenta una revisió de la fase d'assignació de recursos dins de la síntesi d'alt nivell. Es descriuen les diferents subtasques d'aquesta fase, es donen algunes definicions sobre alguns conceptes importants, es presenta una classificació dels diferents tipus d'algorismes existents i es descriuen els més representatius.

2.4.1 Anàlisi del temps de vida de les variables

2.4.1.1 Temps de vida

El temps de vida d'una variable és l'interval de temps durant el qual el registre associat a aquesta variable està ocupat per aquesta variable [ASU86].

Per exemple, el temps de vida d'una variable comença en l'instant en què s'emmagatzema en un registre i acaba en el seu últim ús. Parlarem de *naixement d'una variable* per indicar l'instant de temps en què comença el seu temps de vida. Parlarem de *mort d'una variable* per indicar l'instant de temps en què el seu temps de vida acaba. Dues variables amb temps de vida disjunt poden compartir un mateix registre.

De manera similar podem parlar del temps de vida d'una operació o del temps de vida d'un transferència de dades.

La figura 2.9.a mostra un graf de flux de dades planificat i la 2.9.b l'anàlisi del temps de vida de les variables que apareixen en aquest graf.

2.4.1.2 Graf de compatibilitat

Direm que dues variables, operacions o transferències de dades són compatibles si els seus temps de vida no se solapen, de manera que poden compartir un mateix recurs. Podem construir el graf de compatibilitat $GC = (V, A)$ a partir del graf de flux de dades $DFG = (V, A')$, on V és el conjunt de vèrtexs del graf de flux de dades i existeix un arc $e = (v, w), e \in A$ si les operacions representades pels vèrtexs v, w són compatibles.

2.4.1.3 Clique [MLD92]

Donat un graf $G = (V, A)$, on V és el conjunt dels vèrtexs i A el conjunt dels arcs, un *clique* és un subconjunt de vèrtexs $V' \subset V$ que forma un subgraf complet, és a dir, que existeix un arc $a \in A$ entre qualsevol parell de vèrtexs del conjunt V'.

Un *cobriment en cliques* del graf G és un conjunt de *cliques* $\{A_1, A_2, A_3, ..., A_c\}$ que cobreix tot el graf $(V = A_1 \bigcup A_2 \bigcup A_3 ... \bigcup A_c)$. Un *cobriment en cliques* del graf s'anomena *mínim* si c és mínim.

Quan un cobriment en *cliques* compleix $A_i \bigcap A_j = \emptyset, \forall i \neq j$ llavors tenim un *particionat en cliques* de G. Es pot obtenir un *particionat en cliques* de G eliminant d'un cobriment en *cliques* els vèrtexs que apareguin en més d'un *clique*.

La figura 2.9.c mostra un exemple de graf de compatibilitat pel mateix exemple anterior i una possible solució al problema de particionat mínim en *cliques*.

2.4.1.4 Graf d'incompatibilitat

Direm que dues variables, operacions o transferències de dades són incompatibles si els seus temps de vida se solapen. El graf d'incompatibilitat $GI = (V, A')$ és el graf dual al de compatibilitat. Dos vèrtexs $u, v \in V$ estan connectats per un arc a si les operacions representades per u i v són incompatibles.

2.4.1.5 Colorejat d'un graf d'incompatibilitat

El problema dual de trobar un particionat en *cliques* mínim d'un graf de compatibilitat és el de trobar un colorejat mínim d'un graf d'incompatibilitat. Trobar un colorejat mínim d'un graf consisteix a trobar una funció $f : V \rightarrow \{1, 2, \ldots, K\}$ tal que $f(u) \neq f(v)$ per qualsevol $(u, v) \in A'$, on K sigui mínim. Solucionar el problema de colorejat és NP-hard [GJ79].

La figura 2.9.d mostra un exemple de graf d'incompatibilitat pel mateix exemple anterior i una possible solució al problema de colorejat mínim.

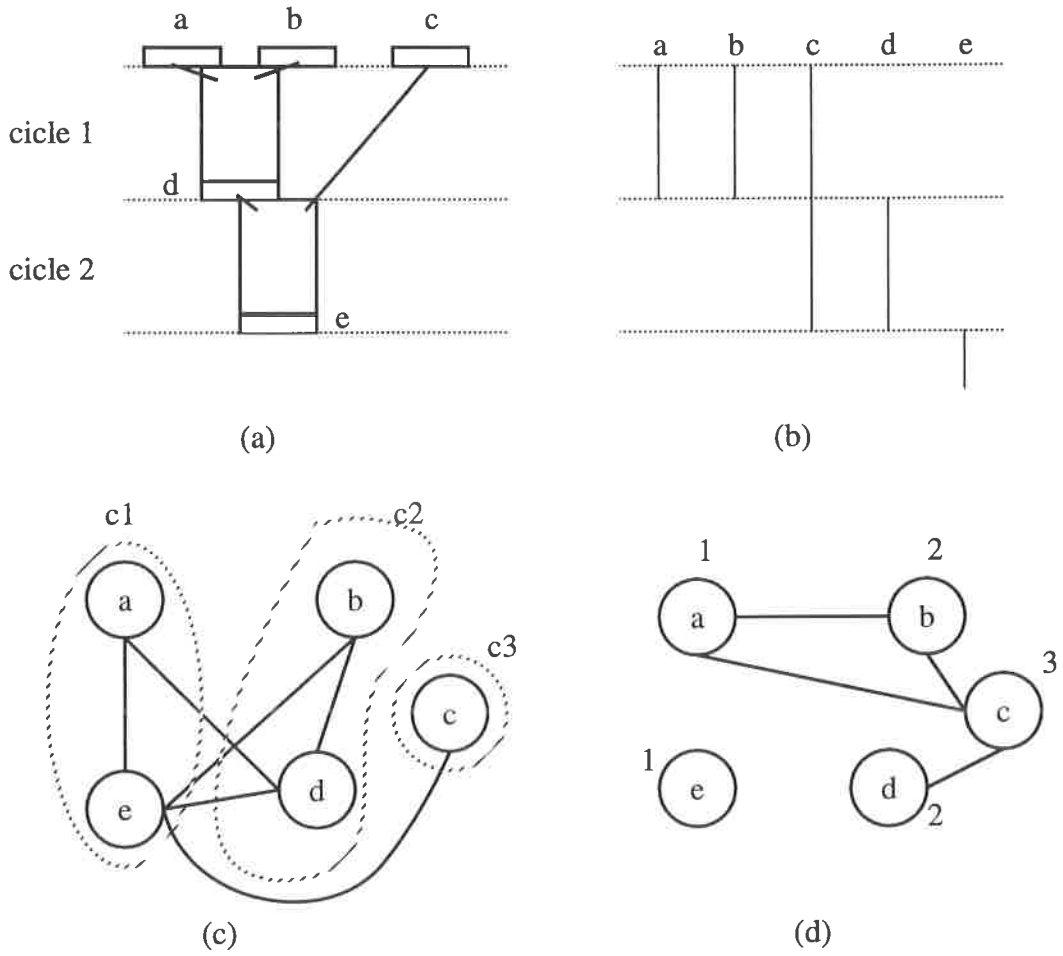


Figura 2.9: (a) Graf de flux de dades planificat ; (b) Anàlisi del temps de vida de les variables; (c) Graf de compatibilitat corresponent i particionat en cliques*; (d) Graf d'incompatibilitat i colorejat*

2.4.2 Introducció

La fase d'assignació de recursos és, juntament amb la de planificació d'operacions, una de les fases més importants de la síntesi d'alt nivell. S'hi decideix quines unitats formaran el camí de dades i quines operacions es realitzaran en cadascuna d'elles. L'assignació de recursos es pot dividir en dues tasques: la selecció i l'associació de recursos. A la primera es tria quins tipus d'unitats funcionals es faran servir i quantes unitats de cada tipus formaran part del camí de dades.

A la segona, s'associa cadascuna de les operacions, variables i transferències de dades a les diferents unitats funcionals, unitats d'emmagatzemament i unitats d' interconnexió.

La fase de selecció es pot realitzar abans o simultàniament a la d'associació. Molts sistemes realitzen la selecció de recursos abans de la planificació d'operacions, de manera que la planificació d'operacions es fa amb restricció de recursos, i l'associació de recursos és l'última tasca que es realitza.

L'associació de recursos encara pot ser dividida en tres subtasques:

- associació d'operacions a unitats funcionals (ALU, sumadors, restadors, multiplicadors...).
- associació de variables a unitats d'emmagatzemament (memòries, registres...).
- associació de transferències de dades a unitats d'interconnexió (busos, multiplexors...).

Alguns sistemes realitzen l'associació d'unitats funcionals abans de la d'emmagatzemament deixant les interconnexions per al final, com el sistema HAL [PK89b] o l'algorisme de correspondència bipartita ponderada [HCLH90]; altres realitzen l'associació d'unitats d'emmagatzemament abans de les funcionals [TS86, HCLH90], i altres realitzen totes les tasques de manera simultània, com al sistema ADAM [KP90b].

2.4.3 Classificació

Els diferents algorismes existents per solucionar el problema de l'associació de recursos es poden classificar en tres grans grups:

- Algorismes orientats al cicle: les diferents subtasques d'associació de recursos es realitzen de manera simultània per cada cicle del graf planificat.

- Algorismes globals: les tres subtasques d'associació de recursos són executades separadament, però de manera global respecte a la planificació d'operacions.

- Algorismes de millora iterativa: donat un camí de dades inicial, es millora la seva qualitat de forma iterativa mitjançant la succesiva reassociació dels diferents mòduls.

El primer grup inclou tots els algorismes constructius: mètodes ambiciosos [KP90b], mètodes de bifurcar i tallar [PG87a, SMT+92] i programació lineal [BM89, RJL92].

El segon grup inclou totes les formulacions de la teoria de grafs: particionat en *cliques* [TS86], colorejat de grafs [HE88, ST91], algorisme del *cantó esquerre*[9] [KP87] i l'algorisme de correspondència bipartita ponderada [HCLH90].

Els algorismes de millora iterativa poden assolir solucions de gran qualitat al preu d'utilitzar tècniques amb ús intensiu de la CPU tal com fan les tècniques d'*escalada de cims* [KN92] com ara el *recuit simulat* o tècniques d'evolució simulada [LEG90].

De totes aquestes tècniques, només els mètodes ambiciosos o iteratius i les formulacions de la teoria de graf poden obtenir solucions per al problema d'associació de recursos a un cost de CPU acceptable. A més, els algorismes orientats al cicle només estan orientats als sistemes síncrons, mentre que un dels nostres interessos principals és la síntesi de circuits asíncrons.

2.4.4 Exemples

A continuació descriurem alguns dels exemples d'algorismes d'associació de recursos existents en la literatura. Aquests exemples són l'algorisme del *cantó esquerre* [KP87], l'heurística de Tseng per fer particionat en *cliques* [TS86] i l'algorisme ambiciós MABAL [KP90b]. Tot i que tots tres mètodes poden ser classificats com a algorismes globals, creiem que és interesant parlar-ne per la seva importància i com a introducció al capítol 6.

[9] Trad. de *left-edge*

2.4.4.1 Algorisme del cantó esquerre

Aquest algorisme és una adaptació del conegut algorisme de connexionat de canal [HS71] per resoldre el problema d'associació de variables a unitats d'emmagatzemament. L'objectiu del connexionat de canal és minimitzar el nombre de pistes utilitzades per connectar punts dins d'un canal.

Dos terminals del canal que pertanyen a una mateix senyal són connectats per dos segments verticals (perpendiculars al canal) que connecten els punts a la pista, i un segment horitzontal (és a dir, paral·lel al canal) —o interval d'expansió del senyal— que viatja per la pista i connecta els dos segments verticals. Dos segments horitzontals poden compartir una mateixa pista si no se solapen.

Com que l'amplada del canal depèn del nombre de pistes utilitzades, l'algorisme intenta agrupar els segments horitzontals que no se solapin en una mateixa pista per tal de minimitzar el nombre de pistes utilitzades. Per fer-ho, es comença associant el segment horitzontal situat més a l'esquerra a la primera pista. Continua amb el següent segment situat a la dreta de l'anterior que no se solapi amb ell i l'associa a la mateixa pista. Aquest procés es repeteix fins que no pot associar cap més segment a la pista. L'algorisme es repeteix per les altres pistes fins que no queda cap segment horitzontal per associar.

Kurdahi i Parker [KP87] varen aplicar l'algorisme del cantó esquerre a l'associació de variables a registres (o, per extensió, a altres unitats d'emmagatzemament). En aquest cas, les pistes són substituïdes per registres i els segments horitzontals pel temps de vida de les variables. L'objectiu de l'algorisme és minimitzar el nombre de registres i, per fer-ho, s'intenta associar variables tals que el seu temps de vida no se solapi al mateix registre —és a dir, variables compatibles són associades al mateix registre—.

En aquest cas, enlloc de començar amb el segment horitzontal situat més a l'esquerra, s'associa al primer registre la variable que *neix* més aviat. A continuació es tria, del conjunt de variables compatibles amb la variable ja associada, aquella que neix primer i s'associa al mateix registre. Aquest procés es repeteix fins que no podem associar cap més variable al registre.

Es procedeix de manera similar per a la resta de registres fins que no queda cap variable per associar. La figura 2.10 mostra un exemple d'associació de variables a registres

utilitzant l'algorisme del cantó esquerre.

Alguns dels trets a favor d'aquest algorisme són que la seva complexitat és polinòmica i que utilitza el nombre mínim de registres, però per altra banda no té en compte l'impacte de l'associació de variables a registres en el cost d'interconnexió (associació de transferències de dades a unitats d'interconnexió).

2.4.4.2 Particionat en cliques

A continuació presentarem un mètode extret de la teoria de grafs aplicable a qualsevol de les tres subtasques d'associació de recursos. Per enllaçar amb el que s'ha explicat en l'apartat anterior, es plantejarà també el cas de l'associació de variables a registres.

Donat un conjunt de variables V construïm un graf de compatibilitats $GC = (V, A)$. Resoldre el problema d'associar variables a registres és equivalent a trobar un particionat en *cliques* mínim del graf on cada *clique* correspon a un registre. Resoldre el particionat mínim en *cliques* és un problema NP-hard [GJ79]. Tseng i Siewiorek [TS86] varen proposar una heurística de cost polinòmic per resoldre aquest problema que, tot i no obtenir resultats òptims, ha esdevingut un clàssic en el tema.

Abans de presentar l'algorisme, cal fer algunes definicions. Definim el *nombre de veïns comuns* d'un arc $e = (v, w)$, $NVC(e)$, com:

$$NVC(e) = \mid \{v' \in V : (v', v) \in A \wedge (v', w) \in A\} \mid$$

És a dir, el nombre de variables que són compatibles amb les dues que uneix l'arc.

Definirem la funció $fusionar(u, v)$ com l'acció de fusionar dos vèrtexs u i v units per un arc $e = (u, v)$ per formar un tercer vèrtex w. Per fer-ho, cal eliminar els vèrtexs u i v i tots els arcs que unien u o w al graf. Després cal afegir el nou vèrtex w i arcs entre ell i els vèrtexs del graf de compatibilitats que eren veïns comuns de u i w. La figura 2.11 mostra un exemple de l'efecte de la funció *fusionar* sobre dos vèrtexs del graf. El significat d'aquesta acció és que les dues variables que són fusionades en un únic vèrtex compartiran un mateix recurs, en aquest cas un registre.

L'algorisme de particionat en *cliques* proposat a [TS86] es mostra a l'algorisme 2.2. Inicialment, l'algorisme calcula el nombre de veïns comuns per a cada arc del graf de compatibilitats ($NVC(e)$, $e \in A$). En cada iteració del bucle es tria un arc $e_{best} = (v, w)$

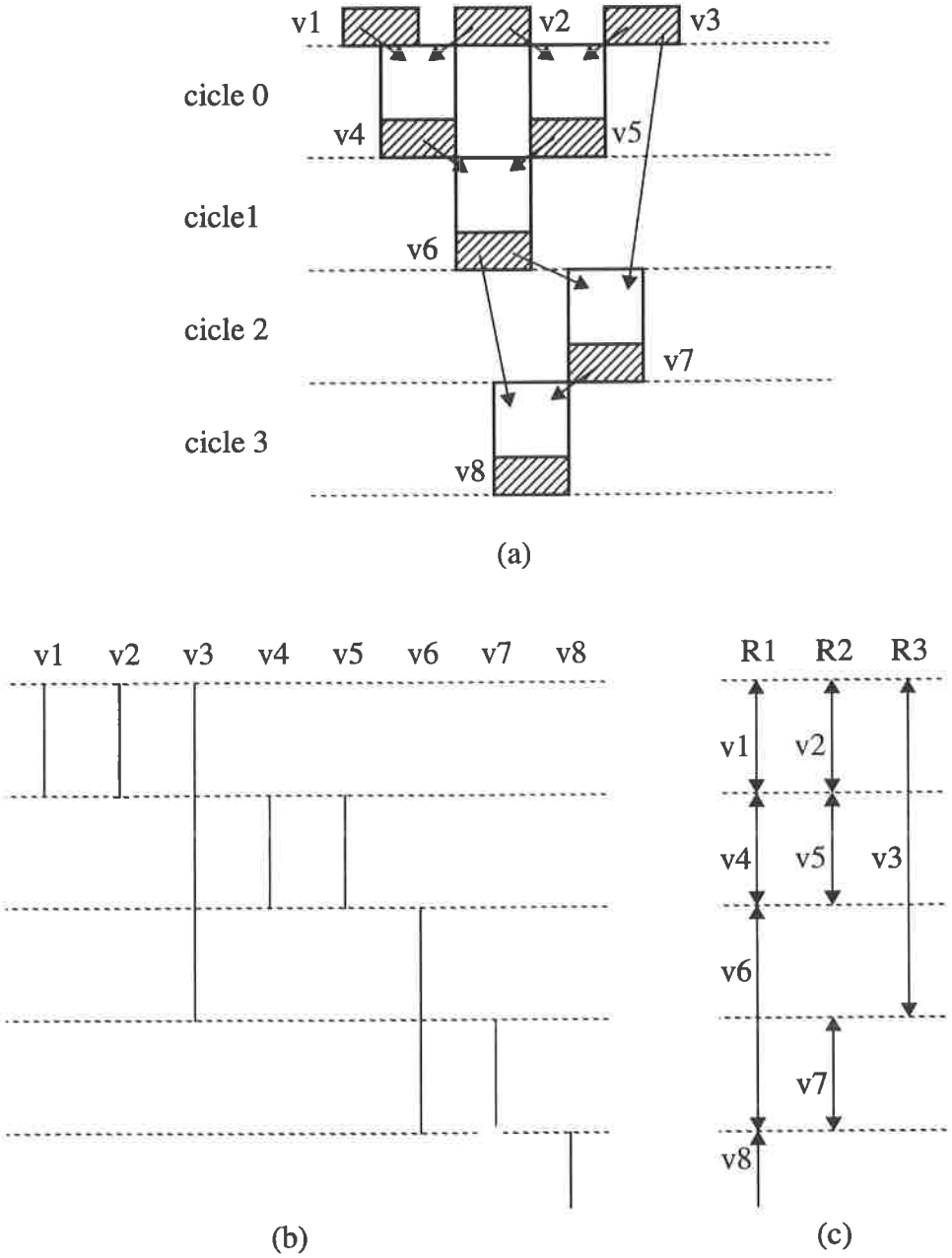


Figura 2.10: (a) Graf planificat inicial; (b) Temps de vida de les variables; (c) Resultat de l'associació de variables a registres

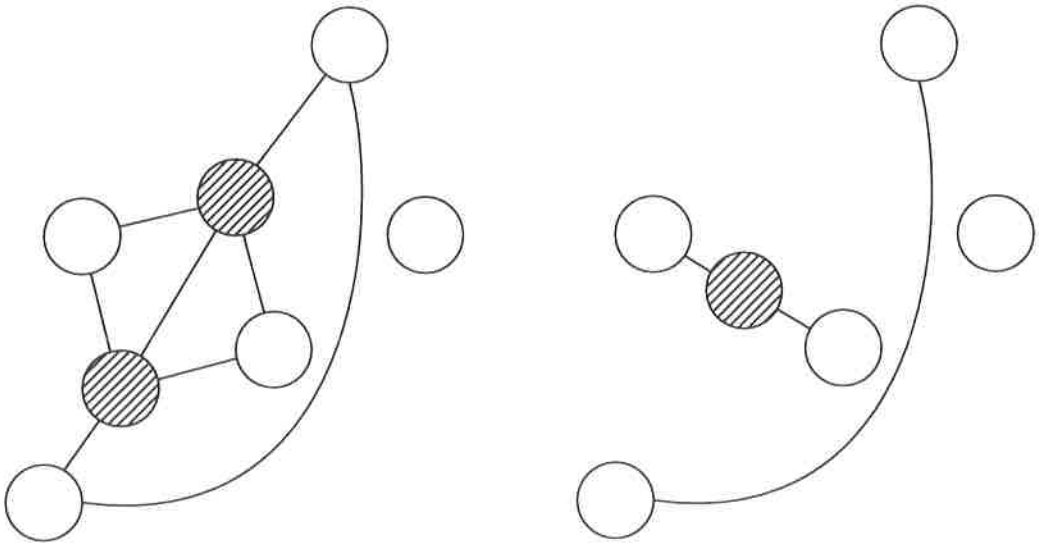


Figura 2.11: Exemple de l'efecte de la funció fusionar *sobre una part del graf de compatibilitats*

tal que el nombre de veïns comuns $NVC(e_{best})$ sigui màxim per a tots els arcs de A. A continuació es *fusionen* els vèrtexs v, w en un únic vèrtex que etiquetarem amb el nom de v. Seguidament, es tria aquell arc $e = (v, w)$ connectat amb el nou vèrtex v que tingui $NVC(e)$ màxim i es fusionen els seus vèrtexs. Aquest pas es repeteix fins que no queda cap arc que connecti v al graf, és a dir, no queda cap variable que pugui compartir el mateix registre amb les que hem anat *fusionant.* L'algorisme acaba quan no queda cap arc per *fusionar.*

L'algorisme tria els arcs amb nombre de veïns comuns màxim, ja que això implica que el vèrtex resultant de fusionar és més probable que es pugui *fusionar* amb d'altres vèrtexs i formar *cliques* més grans. En cas d'empat, s'utilitza un altre paràmetre que és el de *nombre d'arcs a esborrar si els vèrtexs v, w de l'arc e es fusionen.* Es tria aquell arc amb *nombre d'arcs a esborrar* mínim.

La complexitat de l'algorisme és $O(n^2)$, essent n el nombre d'arcs del graf de compatibilitat. Aquest algorisme, a l'igual que el del cantó esquerre no té en compte el cost d'interconnexió alhora d'associar les variables als registres.

Una variant d'aquest algorisme que si té en compte el cost d'interconnexió va ser pre-

particionat en cliques (CG){
  per *tot* $e \in A$ **fer** *calcular* $NVC(e)$;
  repetir
    $e_{best} \leftarrow e = (v, w) : NVC(e) = \max_{A}(NVC)$;
    $merge(v, w)$;
    per *tot* $e \in A$ **fer** *calcular* $NVC(e)$;
    $A_v \leftarrow \{(v, v') \in A\}$;
    mentre $A_v \neq \emptyset$ **fer**
      $e_{best} \leftarrow e = (v, w) : NVC(e) = \max_{A_v}(NVC)$;
      $merge(v, w)$;
      per *tot* $e \in A$ **fer** *calcular* $NVC(e)$;
      $A_v \leftarrow \{(v, v') \in A\}$;
    fifer
  fins que $A = \emptyset$;
}

Algorisme 2.2: Algorisme de particionat en cliques

sentada per Paulin i Knight [PK89b]. En aquesta proposta, s'assignen pesos als arcs del graf de compatibilitat en funció de l'afinitat d'interconnexió existent entre els vèrtexs. Per exemple, s'assigna un pes molt més alt a un arc que connecti dues variables tals que tinguin com a font i destí les mateixes unitats funcionals que a un altre que connecti dues variables que no tinguin cap relació (diferent unitat funcional font i diferent unitat funcional destí).

Un cop calculats els pesos dels arcs, s'aplica l'algorisme anterior al subgraf de compatibilitat $SG = (V, A')$ on els arcs $e' \in A'$ tenen un pes superior a un cert valor llindar.

Una altra formulació basada en la teoria de grafs que també ha estat molt utilitzada és el colorejat de grafs de conflictes [HE88, ST91]. El colorejat de grafs i el particionat en *cliques* són problemes duals que tenen la mateixa complexitat. En el graf de conflictes, a diferència del graf de compatibilitat, dos vèrtexs comparteixen un arc si les operacions que representen no són compatibles. Solucionar el problema d'assignació de recursos consisteix a trobar un colorejat del graf tal que dos vèrtexs adjacents no tinguin el mateix color assignat.

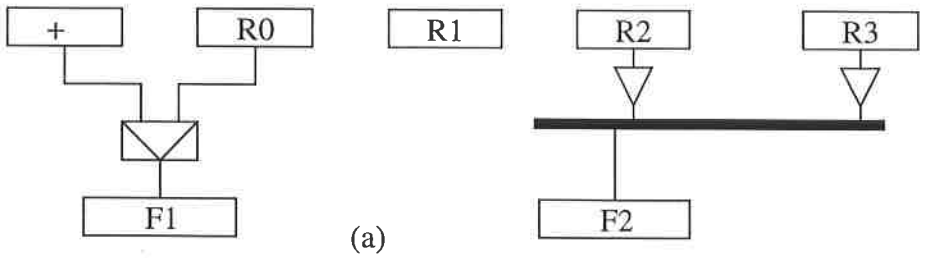
2.4.4.3 MABAL

L'algorisme que es presenta a continuació és un algorisme constructiu que associa pas a pas les operacions, variables i transferències de dades als diferents recursos. Aquest algorisme parteix d'un camí de dades buit i de manera gradual va afegint unitats funcionals, registres i interconnexió a mesura que són necessaris. En cada iteració, es calcula el cost mínim d'afegir cadascuna de les diferents operacions, variables o transferències de dades que encara no estan associades a cap element del camí de dades. De tots els casos es tria el que tingui un cost mínim. El procés es repeteix fins que no queda cap entitat per associar. L'algorisme 2.3 descriu l'algorisme MABAL.

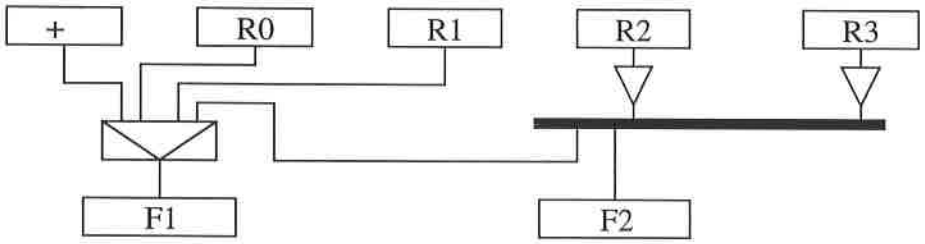
A continuació descriurem com l'algorisme considera les diferents alternatives i com calcula el seu cost pel cas d'una operació que encara no està associada a cap unitat funcional. Suposem que tenim un camí de dades a mig construir (figura 2.12.a) i que l'algorisme vol avaluar el cost d'associar una operació x a una unitat funcional de tipus F al camí de dades actual [KP90a]. Suposarem que l'operació té una entrada que prové de $R1$ i una de $R2$. L'algorisme busca la millor associació per a l'operació i per a la interconnexió des de

```
algorisme MABAL(ENoA){
    Data-path_actual = ∅;
    mentre ENoA ≠ ∅ fer
        CostMinim = ∞;
        per tot enoa ∈ ENoA fer
            Data-path_tmp = AFEGIR(Data-path_actual, enoa);
            Cost_tmp = COST(Data-path_tmp);
            si Cost_tmp < CostMinim llavors
                CostMinim = Cost_tmp;
                MillorEntitat = enoa;
            fisi
        fiper Data-path_actual = AFEGIR(Data-path_actual, MillorEntitat);
        ENoA = ENoA − MillorEntitat;
    fimentre
}
```

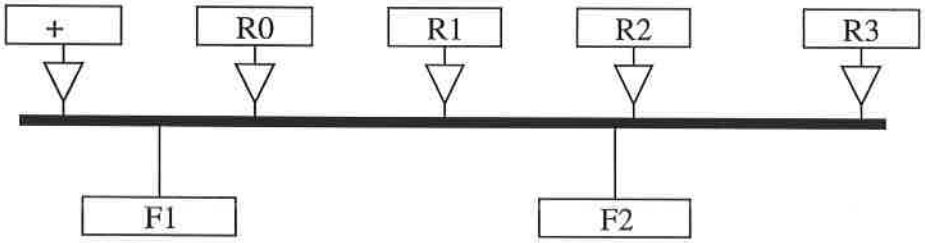
Algorisme 2.3: Algorisme MABAL



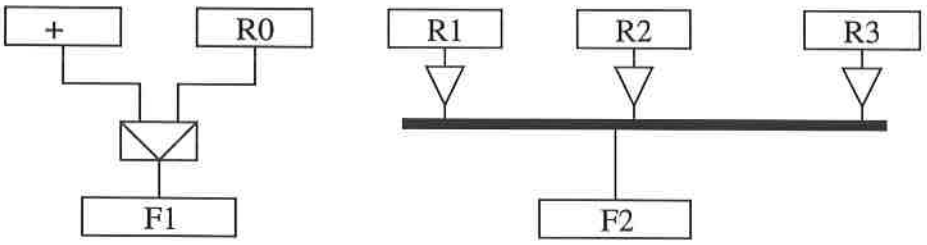
(a)



(b) $\Delta Cost = 2 \cdot C_{mux(2 \rightarrow 1)}$

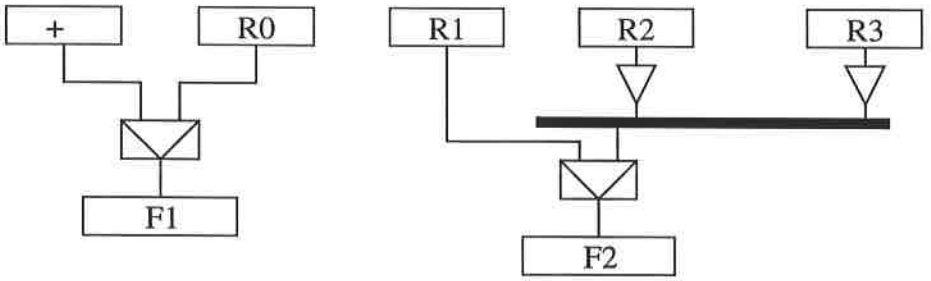


(c) $\Delta Cost = 3 \cdot C_{tsd} - C_{mux(2 \rightarrow 1)}$

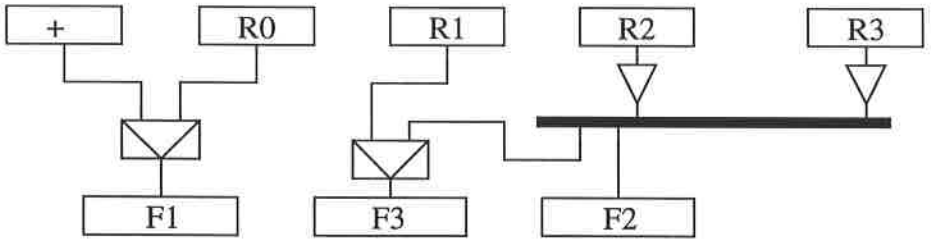


(d) $\Delta Cost = C_{tsd}$

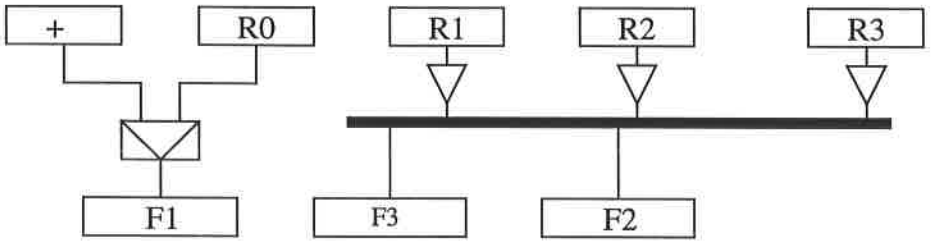
Figura 2.12: (a) Camí de dades inicial al què es vol afegir una operació x *a una unitat funcional de tipus F; (b) Associa l'operació x a F1 i l'interconnexió es realitza amb multiplexors; (c) Associa l'operació a F1 i l'interconnexió es fa amb busos; (d) Associa l'operació a F2 i interconnexió amb bus [KP90a]*



(e) $\Delta Cost = C_{mux(2\rightarrow1)}$



(f) $\Delta Cost = C_{mux(2\rightarrow1)} + C_F$



(g) $\Delta Cost = C_{tsd} + C_F$

Figura 2.13: (e) Associa l'operació a F2 i interconnexió amb multiplexor; (f) Afegeix una nova unitat funcional F3 i interconnexió amb multiplexor; (g) Afegeix una nova unitat funcional F3 i interconnexió amb bus [KP90a]

$R1$ i $R2$. El primer cas que es considera és el d'associar l'operació x a la unitat funcional $F1$ i que la interconnexió necessària es faci amb multiplexors (figura 2.12.b). En aquest cas, l'increment de cost pel fet de prendre aquesta decisió és de dos multiplexors de dues entrades[10].

Una segona alternativa (figura 2.12.c) seria fer la interconnexió amb un únic bus enlloc de fer-la amb multiplexors. En aquest cas, l'increment de cost és la diferència entre el cost de tres elements tres-estats i un multiplexor de dues entrades. La tercera i quarta alternatives considerades (figura 2.12.d i 2.13.e) associen l'operació x a la unitat funcional $F2$. A la primera d'elles s'utilitzen busos per fer l'interconnexió i, a la segona, s'utilitzen multiplexors. El cost incremental d'una i altra opció són el cost d'un element tres-estats i el cost d'un multiplexor de dues entrades, respectivament.

Per últim, es considera el cas d'afegir una nova unitat funcional de tipus F ($F3$) al camí de dades. Les figures 2.13.f i 2.13.g mostren els camins de dades obtinguts per aquesta alternativa depenent de si l'interconnexió es fa amb multiplexors o amb bus. En aquest cas, a més del cost d'interconnexió, cal afegir al cost incremental el cost de la nova unitat funcional.

2.5 Conclusions

Al primer capítol s'ha vist la necessitat d'automatitzar el disseny dels circuits. En aquest capítol, s'ha presentat la síntesi d'alt nivell, s'han descrit les seves fases i s'ha aprofundit en les més importants.

La síntesi d'alt nivell té com a objectiu obtenir una descripció estructural d'un circuit a partir d'una descripció del seu comportament. Atesa la complexitat d'aquest procés s'ha descompost en diferents fases: especificació, traducció, planificació d'operacions i assignació de recursos. Les dues últimes són les més importants de la síntesi d'alt nivell.

Tant la planificació d'operacions com l'assignació de recursos tenen una gran complexi-

[10]Per unificar criteris, se sol comptar el cost d'interconnexió dels multiplexors amb el nombre de multiplexors de dues entrades necessaris per construir el multiplexor que es necessiti. En aquest cas, passem d'un multiplexor de dues entrades a un de quatre. Amb tres multiplexors de dues entrades en podem construir un de quatre i, per tant, el cost incremental és de dos multiplexors de dues entrades

tat i s'han d'utilitzar heurístiques per poder donar resultats propers a l'óptim en un temps de CPU no molt gran. Per a cadascuna d'elles s'ha plantejat quin és el problema a resoldre, s'ha fet una classificació dels algorismes existents i s'han vist diferents formulacions que s'han proposat a la literatura durant els últims anys.

Tot i que la quantitat i qualitat dels algorismes existents és important, cal fer èmfasi en el fet que fins al moment no s'han proposat algorismes orientats a la síntesi d'alt nivell de circuits asíncrons. L'únic algorisme de planificació d'operacions en el que es tracten operacions amb un retard desconegut és l'algorisme de planificació relativa proposat per Ku i De Micheli [KM90].

En els capítols 5, 6 i 7 es presenten diferents propostes per fer planificació d'operacions i assignació de recursos per sistemes asíncrons.

Capítol 3

Sistemes asíncrons

Els circuits asíncrons presenten una alternativa al disseny tradicional síncron. Els circuits asíncrons es caracteritzen per l'absència de senyal de rellotge. Aquest tret fa que tinguin alguns avantatges com l'absència del problema de desfasament de rellotge o una velocitat mitjana de càlcul. Tot i aquests avantatges, els circuit asíncrons han estat descartats durant molts anys a causa de la dificultat del seu disseny. Aquest capítol presenta una introducció al disseny dels circuits asíncrons i fa una revisió de diferents aspectes relacionats amb el seu disseny i característiques.

3.1 Disseny de circuits asíncrons

Tot i que el disseny de circuits asíncrons ha estat present durant les últimes quatre dècades [Huf57, Ung69], no ha estat fins recentment que ha tornat a ser considerat com a alternativa als circuits síncrons.

En un *circuit síncron* podem diferenciar clarament entre els circuits combinacionals i els elements de memòria. Aquesta separació facilita el disseny dels circuits síncrons. El flanc del senyal de rellotge marca l'inici i finalització de les operacions. Els senyals de sortida dels elements combinacionals poden oscil·lar durant el cicle amb l'única condició que siguin estables al final del mateix. Totes les possibles realimentacions de la lògica combinacional queden trencades per elements de memòria.

Un *circuit asíncron* és una interconnexió arbitrària de portes, en què no podem distingir la part combinacional dels elements de memòria. Aquesta llibertat fa que el seu disseny sigui molt més complex, ja que apareixen problemes com els riscos i la metastabilitat. En els darrers anys, els dissenyadors s'han mostrat escèptics a acceptar els circuits asíncrons, tot i els seus avantatges, atesa la complexitat del seu disseny.

A continuació es descriuen aquells motius que fan que el disseny de circuits asíncrons i, per tant l'automatització del seu disseny, sigui desitjable i d'altra banda es descriuen quins són els inconvenients del disseny asíncron.

3.1.1 Motivacions per al disseny de circuits asíncrons

3.1.1.1 Desfasament del rellotge global

A mesura que la tecnologia avança, es redueix l'àrea necessària per integrar els circuits. Cada vegada podem incorporar més circuits en els xips i el procés de fabricació dels sistemes s'ha abaratit. En els circuits síncrons aquest fet ha provocat que el desfasament del senyal de rellotge sigui un problema cada vegada més important.

En circuits que treballen amb un rellotge global, aquest senyal s'ha de distribuir per tot el xip. Com que aquesta línia és considerablement llarga, el flanc de rellotge no arriba en el mateix instant de temps a tots els elements del circuit. El *desfasament del rellotge* és la diferència que hi ha entre els temps d'arribada del flanc de rellotge als diferents elements

de memòria del circuit. Per tal que el circuit funcioni correctament, aquest desfasament no pot superar un valor determinat.

A mesura que augmenta la mida i complexitat dels circuits, augmenta també la complexitat de la distribució del senyal de rellotge. A més, la capacitat de la línia de rellotge té tendència a restar constant a mesura que escalem el circuit fent que el desfasament sigui proporcionalment més important. Tot i que s'han proposat tècniques com la distribució en arbre o la utilització de múltiples fases de rellotge [JM86, Bak90, Tew89], la distribució del senyal de rellotge continua essent un problema crític.

Un exemple dels problemes que comporta la distribució del senyal de rellotge, el trobem en el microprocessador *DEC Alpha* [Dob92]. Aquest processador, dissenyat amb tecnologia CMOS de $0,75\ \mu m$, té 1,68 milions de transistors i la capacitat total del senyal de rellotge és de $3,25\ nF$. Per realitzar la distribució del rellotge s'han utilitzat cinc nivells de *buffering*. D'aquesta manera s'ha obtingut un desfasament del rellotge de 300 *ps*.

En els circuits asíncrons, com que no existeix el senyal de rellotge, no existeix aquest problema.

3.1.1.2 Velocitat de càlcul

En els circuits síncrons, el començament de les operacions ve determinat pel senyal de rellotge. Si considerem el cas d'operacions unicicle, el temps de cicle ve determinat pel retard del mòdul més lent en el seu pitjor cas. Així doncs, en la majoria dels casos estem desaprofitant el temps. En els circuits asíncrons, les operacions poden començar tan aviat com les operacions predecessores hagin acabat, de manera que unes vegades trigarem més i altres trigarem menys en funció de les dades d'entrada, però en general podem considerar un temps mitjà.

Per exemple, si considerem un sumador de n bits amb propagació de *carry*, el temps pitjor és proporcional al temps de propagar el *carry* per tots els bits ($O(n)$) mentre que en mitjana el sumador triga un temps proporcional a propagar el *carry* a $\log n$ bits ($O(\log n)$).

3.1.1.3 Modularitat dels dissenys

En els circuits asíncrons la sincronització entre diferents mòduls es realitza mitjançant intercanvis de senyals de sincronització que implementen un protocol de dues o quatre fases. D'aquesta manera la interconnectivitat és totalment modular, és a dir, podem canviar la implementació d'un dels móduls, però mentre se sincronitzin de manera correcta entre si, el sistema funcionarà correctament (tot i que el seu rendiment pot variar).

En canvi, en els circuits síncrons no és sempre així. Imaginem un sistema format per dos components que es comuniquen entre si governats pel mateix rellotge. Si en algun moment es vol canviar un dels mòduls substituint-lo, per exemple, per un de comportament equivalent però més ràpid, governat per un rellotge de freqüència diferent a l'anterior, haurem d'afegir al circuit un component de sincronització entre els dos components anteriors, ja que ara no estan governats pel mateix rellotge. És per aquest motiu que han tingut gran èxit les interfícies asíncrones, com per exemple el bus VME [Hea88, SL89].

3.1.1.4 Consum

En els circuits síncrons una part molt important del consum es produeix al voltant del flanc del rellotge i en la distribució del senyal de rellotge. A més, els circuits síncrons sempre consumeixen, ja que el senyal de rellotge sempre està actiu, fins i tot quan el circuit no està fent cap tipus de càlcul.

Un exemple del consum en un sistema síncron el trobem de nou en el microprocessador *DEC Alpha*. Amb una alimentació de 3,3 V i una freqüència de rellotge de 200 MHz, té un consum de 30 W. El 40% del consum és degut als *drivers* del rellotge.

En canvi, en els circuits asíncrons només consumeixen aquelles parts del circuit que estan funcionant. Si pensem en tots aquells processadors que estan actius per la nit, executant operacions de tipus NOP, podrem imaginar-nos el gran estalvi que podria suposar que aquests processadors fossin asíncrons.

3.1.2 Inconvenients del disseny de circuits asíncrons

Tot i les raons a favor del disseny de circuits asíncrons que s'han comentat a l'apartat anterior, els circuits asíncrons no han estat acceptats durant molts anys a causa de la

complexitat del seu procés de disseny. Vegem quins són els inconvenients del disseny asíncron i d'on ve la complexitat del seu disseny.

3.1.2.1 Àrea

L'àrea requerida per als circuits asíncrons és més gran que en la dels circuits síncrons per diverses raons. D'una banda, tenim que els components del camí de dades ocupen més espai. Per exemple, si utilitzem la família lògica DCVSL [WF83] per implementar els elements de la llibreria tindrem una penalització en àrea d'aproximadament el 40% per fer el connexionat dels senyals diferencials d'entrada i de sortida (doble via).

També haurem de tenir en compte l'àrea addicional del control que sol ser més gran atès el cost addicional que comporta implementar el control amb circuits amb sincronització explícita.

L'àrea no serà sempre un factor crític ja que cada vegada serà més gran la quantitat de transistors integrables en un xip.

3.1.2.2 Riscos i curses

Un *risc* és el comportament erroni, encara que només sigui temporalment, en un senyal, que pot produir que el funcionament del circuit sigui incorrecte. Podem diferenciar entre dos tipus de riscos: els riscos estàtics i els riscos dinàmics:

Definició 3.1 Risc estàtic

Un risc estàtic és una o més transicions 0-1-0 (o 1-0-1) en un senyal quan el comportament esperat era un 0 estàtic (o un 1 estàtic) [Moo92].

Definició 3.2 Risc dinàmic

Un risc dinàmic és una transició múltiple de 0 a 1 (o de 1 a 0) quan el comportament esperat era que hi hagués una única transició de 0 a 1 (o de 1 a 0) [Moo92].

Definició 3.3 Cursa

Quan dues o més variables canvien al mateix temps es diu que el circuit té un risc seqüencial o una cursa. Si la sortida o l'estat del circuit depèn del resultat de la cursa, per exemple de la seqüència del canvis, es diu que la cursa és crítica [Moo92].

Definició 3.4 Risc essencial

Un risc essencial és una cursa entre entrades i variables d'estat d'un circuit que fa que el circuit quedi en un estat incorrecte [Moo92].

Cal evitar totes les combinacions de riscos, curses crítiques i riscos essencials ja que poden provocar malfuncions en el circuit. Però evitar tots aquests riscos no és una tasca trivial i és per això que el disseny de circuits asíncrons ha estat considerat com a inviable durant molts anys i només s'ha utilitzat en camps molt concrets com el disseny d'interfícies [Hea88, SL89].

La figura 3.1.a mostra un exemple de circuit asíncron amb riscos. Aquest circuit implementa la funció:

$$f = ab + b'f$$

i la seva taula de Karnaugh és la que mostra la figura 3.1.b. Cadascuna de les portes AND es correspon amb un dels cubs del recobriment de tots els valors a 1 de la funció.

En l'estat $a = b = f = 1$ on els nodes interns valen $c = 1$ i $d = 0$, si es produeix una transició del senyal b de 1 a 0, tenim un risc en el circuit. Si suposem que el retard de l'inversor és més gran que el de les portes AND, tenim que la porta AND que estava a 1 es desactiva abans que l'altra s'activi, de manera que la sortida (f) es desactiva, quan hauria de mantenir-se a 1. Si finalment la segona porta s'activa, la sortida tornarà a ser 1. En aquest cas es produiria un risc estàtic.

Però en aquest cas encara pot ser pitjor, ja que el valor f es realimenta i provoca una desactivació permanent de la segona porta, i el circuit roman en l'estat erroni $b = f = 0$ i $a = 1$.

Per solucionar aquest risc pot afegir-se una porta AND redundant que cobreixi la transició del senyal b de 1 a 0, tal com mostra la figura 3.1.c.

3.1.2.3 Metastabilitat

Un altre problema dels circuits asíncrons és la metastabilitat. Un exemple d'aquest problema el podem il·lustrar amb un àrbitre que garanteixi exclusivitat mútua a algun tipus de recurs. Si es produeixen peticions simultànies a l'àrbitre, pot passar que aquest arribi a

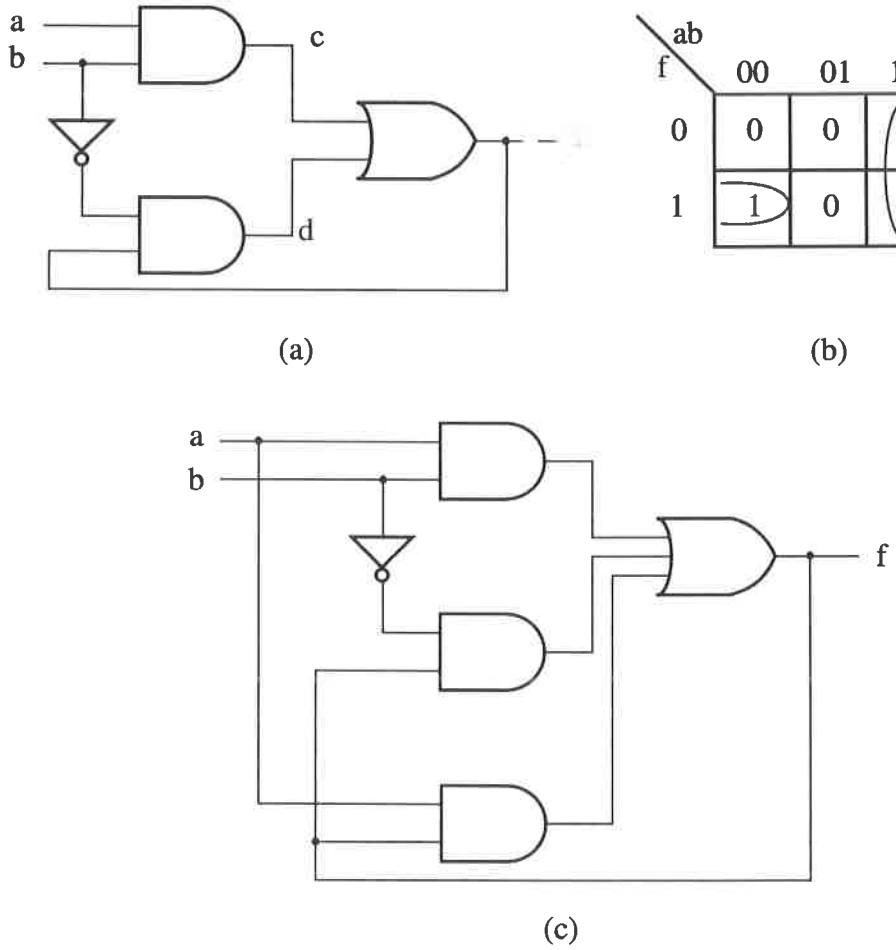


Figura 3.1: (a) Exemple de circuit asíncron amb riscos; (b) Taula de Karnaugh del circuit anterior; (c) Circuit anterior modificat

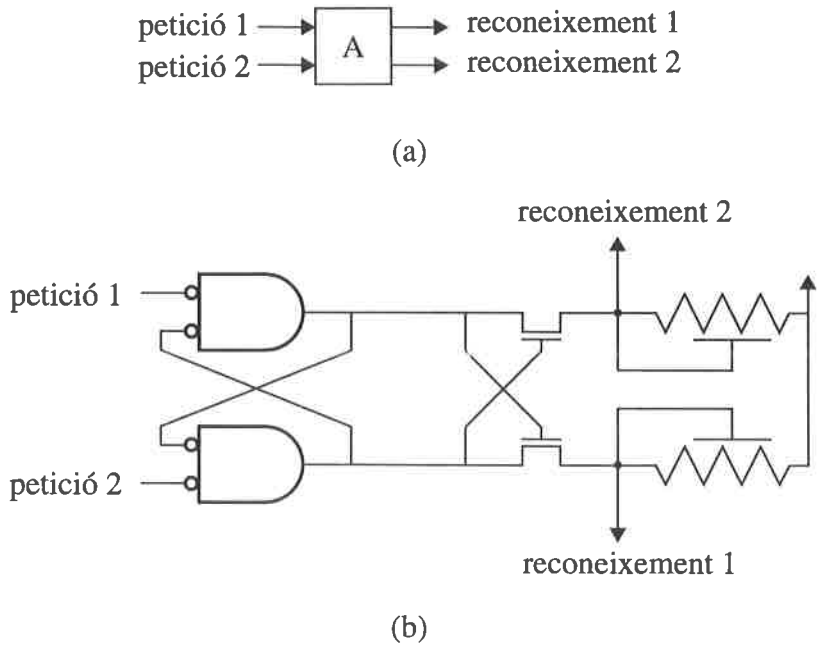


Figura 3.2: (a) Esquema d'un àrbitre; (b) Implementació de l'àrbitre anterior

un estat metastable, on la sortida resta indefinida durant un període de temps indeterminat. La figura 3.2 mostra un exemple d'àrbitre que controla dos enllaços, on les peticions poden ser simultànies, però els reconeixements són mútuament exclusius.

La metastabilitat no és un problema exclusiu dels circuits asíncrons, sinó que també existeix en els circuits síncrons, però en ells l'existència del rellotge permet evitar aquest fenomen més fàcilment.

Com que és evident que el disseny dels circuits asíncrons no és una tasca trivial i que els avantatges dels circuits asíncrons són grans, sembla raonable invertir esforços en automatitzar aquest disseny.

3.2 Terminologia

Al llarg del capítol es faran servir diferents termes amb què s'han classificat els circuits asíncrons. En aquesta secció definirem aquests termes.

Definició 3.5 Circuits autotemporitzats

Els circuits autotemporitzats estan formats per la interconnexió de components elementals. Cadascun d'aquests components inicia un càlcul quan s'activa un senyal de petició i activa un senyal d'acabament per indicar que el càlcul ja ha finalitzat. La succesió dels càlculs en cadascun dels components segueix un ordre preestablert. Els components es comuniquen entre si mitjançant l'intercanvi de senyals de sincronització [Sei80].

Definició 3.6 Circuits insensibles als retards

E's circuits insensibles als retards só una forma restringida de circuits autotemporitzats [vB92]. Un circuit és insensible als retards si el seu comportament extern resta invariable, encara que s'inserti un nombre indeterminat d'elements de retard a les entrades o sortides dels components que el formen, o encara que s'eliminin elements de retard preexistents al circuit [Kel74]

Definició 3.7 Circuits independents de la velocitat

Un circuit és independent de la velocitat si el seu comportament extern és independent del retard dels mòduls que el constitueixen. [Kel74].

Si tornem a l'exemple del circuit de la figura 3.1, hem vist que la transició del senyal b de 1 a 0 està lliure de riscos si no es produeix cap més transició mentre el circuit no s'estabilitzi. Però si es produeix una transició en un altre senyal mentre el circuit no estigui estable, no es garanteix que el resultat sigui el correcte. Això es degut a que el circuit suposa una *velocitat de l'entorn* i per tant no és independent de la velocitat.

3.3 Mòduls autotemporitzats

Els mòduls autotemporitzats es caracteritzen pel fet que l'activació d'un senyal de petició els indica que han de començar a fer un càlcul i indiquen que aquest càlcul ha arribat a la seva fi activant un senyal d'acabament [Sei80]. Els protocols utilitzats per executar una operació poden ser de dues fases ($\{petició^*, acabament^*\}$) o de quatre fases ($\{petició^+, acabament^+, petició^-, acabament^-\}$) [1].

[1] El simbol * darrere un senyal indica una transició en aquest senyal, i els simbols $^-$ i $^+$ indiquen una transició d'aquest senyal de 1 a 0 i de 0 a 1, respectivament

A^T	A^F	Valor codificat
0	0	espai
0	1	FALS
1	0	CERT
1	1	valor no utilitzat

Taula 3.1: Codificació doble via

A	1		0		0		1		1		0	
A^T	0	1	0	0	0	0	0	1	0	1	0	0
A^F	0	0	0	1	0	1	0	0	0	0	0	1

Taula 3.2: Transmissió amb doble via

A continuació veurem quines són les diferents opcions que hi ha per realitzar mòduls autotemporitzats.

3.3.1 Codificació de dades amb doble via

Aquest tipus de mòduls es caracteritzen pel fet que cada bit de dades és codificat amb dos senyals A^T i A^F [Sei80, WHAY87, KKTV94]. La taula 3.1 mostra la codificació més utilitzada d'aquests senyals. Abans de fer una operació en el mòdul, el valor inicial de cada bit de les dades de sortida és la codificació 00, també anomenada *espai*. Quan l'operació acaba, un dels dos senyals que codifiquen cada bit passa a valer 1, de manera que per detectar si l'operació ha acabat, és a dir, per activar el senyal d'acabament d'una operació, n'hi ha prou amb comprovar que cadascun dels parells de senyals tinguin valors complementats.

La codificació 11 no es produirà mai, ja que per passar de 01 a 10 o viceversa sempre passarem per 00.

La taula 3.2 mostra un exemple de transmissió d'informació codificada en doble via.

3.3.2 Mòduls amb dades compactades

En aquest cas el mòdul no té cap característica especial i el senyal de finalització és simplement el senyal d'inici retardat. El retard ve determinat pel temps màxim que pot

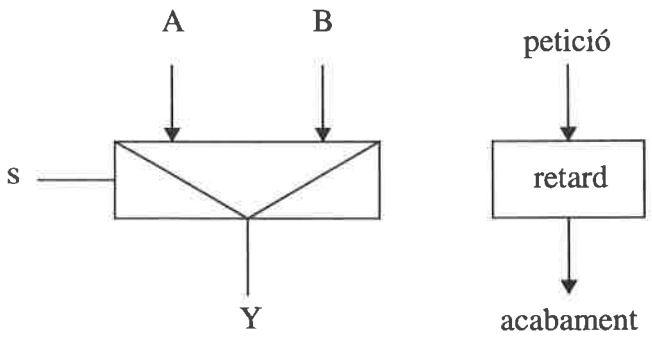


Figura 3.3: Exemple de mòaul amb dades compactades

trigar el mòdul a realitzar una operació. Aquesta tècnica només és adequada quan el temps del mòdul per realitzar operacions es gairebé constant, independentment de les dades d'entrada. Aquesta opció, tot i que no gaudeix d'algunes de les propietats de la tècnica anterior, es utilitzat en alguns casos, ja que la lògica del circuit no és tan complexa [CDS93].

3.3.3 Mòduls autotemporitzats amb lògica DCVSL

En aquest apartat es presenta la família lògica DCVSL (*Differential Cascode Voltage Switch Logic*) [HG84] com a alternativa per construir mòduls autotemporitzats. Els senyals de petició i acabament controlen els mòduls autotemporitzats. El protocol implementat amb aquests senyals pot ser de dues o quatre fases. La figura 3.4 mostra un exemple de cadascun d'aquests protocols.

La família lògica DCVSL és una família completa que utilitza la codificació amb doble via per les dades d'entrada i de sortida, de manera que una mateixa porta genera la funció que es vol calcular i la funció negada. Les portes DCVSL es caracteritzen per tenir en el *pull-up* un parell de transistors pMOS i en el *pull-down* dues xarxes complementàries de transistors nMOS que calculen la funció i la funció negada. En alguns casos, però, alguns dels transistors de les dues xarxes es poden compartir de manera que es redueix el nombre de transistors necessaris. La figura 3.5.a mostra un exemple de porta DCVSL genèrica, la figura 3.5.b mostra una porta XOR en lògica DCVSL en què els transistors de les xarxes nMOS es comparteixen i la figura 3.5.c mostra una porta AND en lògica DCVSL.

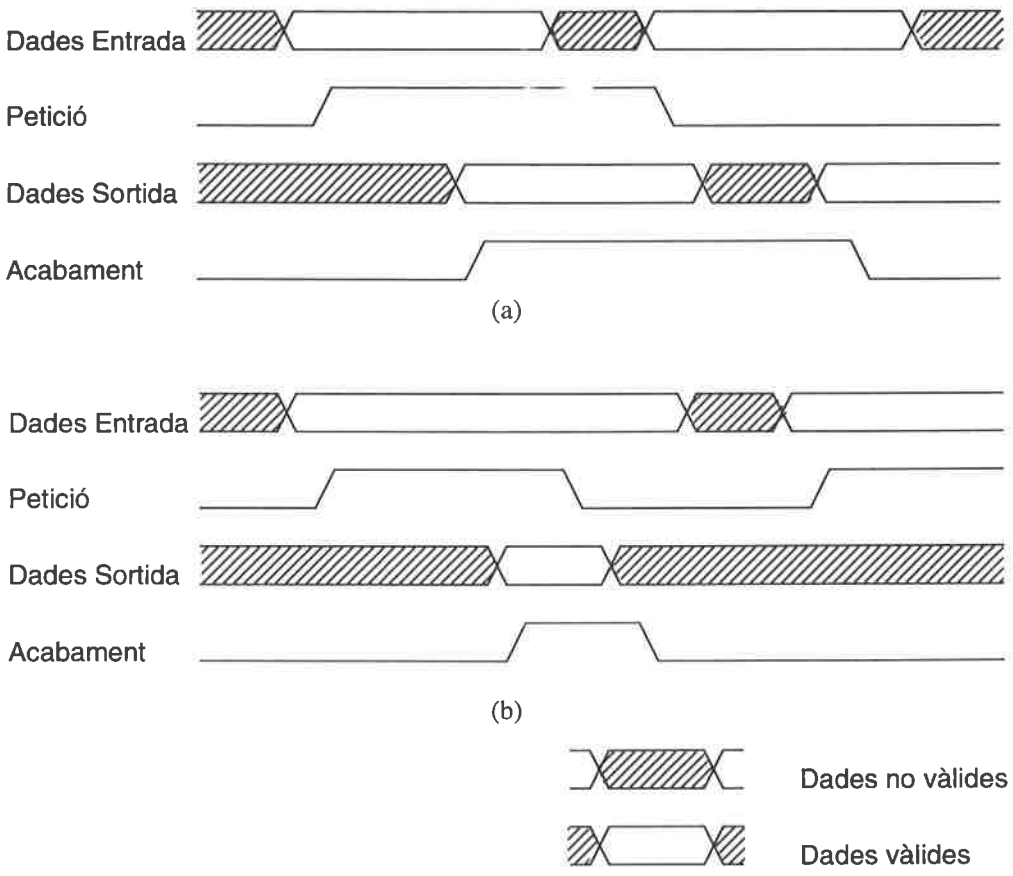


Figura 3.4: (a) Exemple de protocol de dues fases; (b) Exemple de protocol de quatre fases

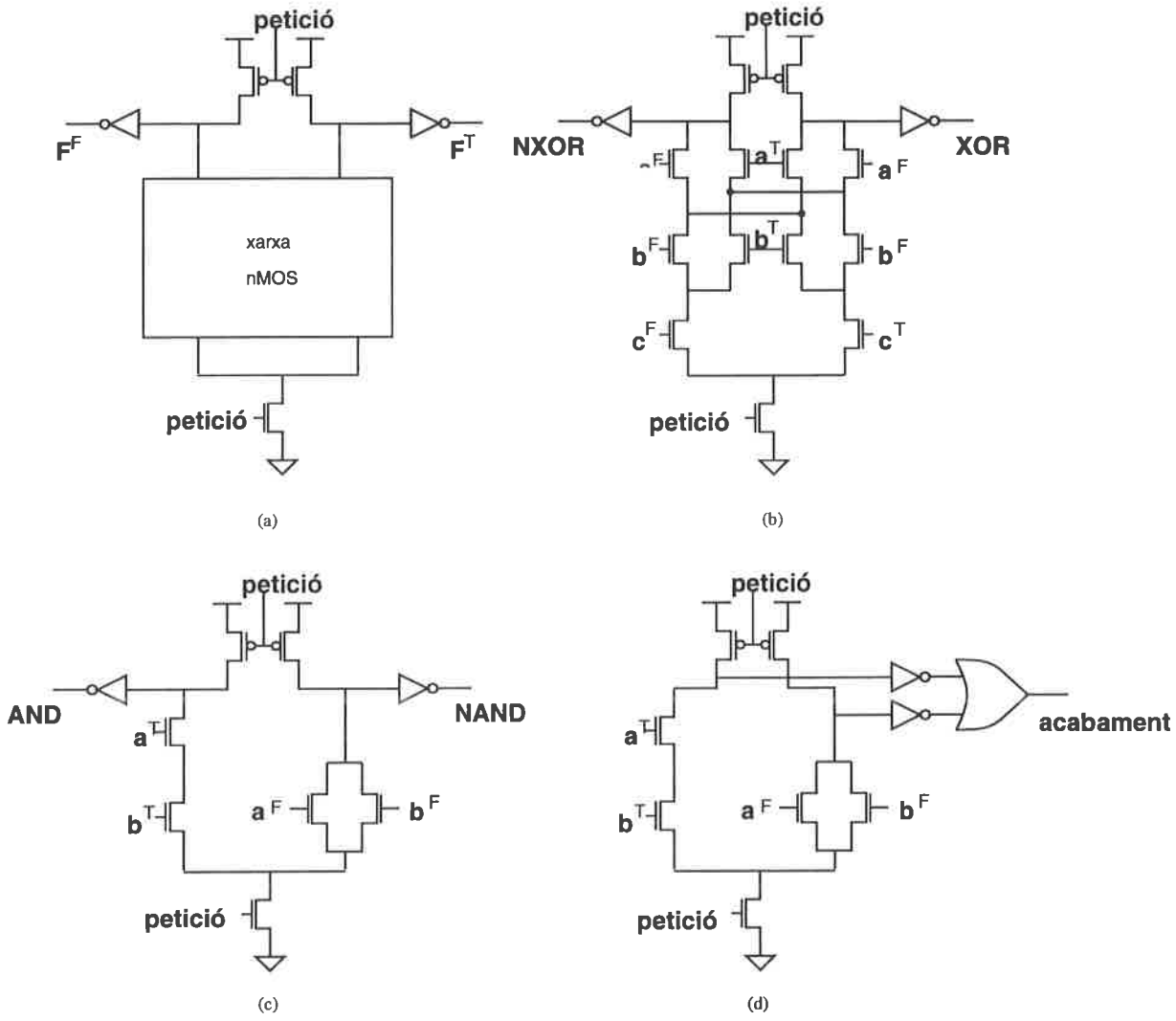


Figura 3.5: (a) Porta genèrica amb lògica DCVSL; (b) Porta XOR amb lògica DCVSL; (c) Porta AND amb lògica DCVSL; (d) Generació del senyal d'acabament

A continuació veurem alguns aspectes importants pel que fa a la construcció de mòduls autotemporitzats amb lògica DCVSL, com són l'avaluació de les funcions i la detecció d'acabament de l'avaluació.

3.3.3.1 Avaluació de funcions amb portes DCVSL

Les portes DCVSL avaluen les funcions en tres fases. A continuació veurem quines són i s'il·lustrarà amb l'exemple concret d'una porta AND (figura 3.5.c).

1. **Fase d'inicialització**: comença quan el senyal de petició val 0. Tant la funció com la funció negada es precarreguen a 0. Les entrades poden ser vàlides o estar inicialitzades (codi 00).

   Per al nostre exemple, imaginem que inicialment les entrades a i b estan inicialitzades i valen el codi 00. La funció AND i la funció NAND passaran a valer 0 (sortides inicialitzades). Si alguna de les dues entrades passés a tenir un valor vàlid mentre s'està fent la inicialització, no afectaria ja que el transistor nMOS del *pull-down*, controlat pel senyal petició, manté tots els camins tallats.

2. **Fase d'avaluació**: comença quan el senyal de petició passa a valer 1. La funció no avaluarà fins que totes les entrades siguin vàlides. Un cop totes les entrades són vàlides, una de les dues subxarxes nMOS es descarrega de manera que una de les dues funcions passa a valer 1.

   En el nostre cas, si qualsevol de les dues entrades val 0 (codi 01) es descarregaria el node intern de la funció NAND, passant a valer 1 i la funció AND es mantindria a 0. En cas que totes dues entrades valessin 1 (codi 10) es descarregaria el node intern de la funció AND. Per continuar amb l'exemple suposarem que el resultat de la funció AND és 0.

3. **Fase de memorització**: aquesta fase només té lloc quan la funció ja s'ha avaluat i si el senyal de petició és manté actiu quan els senyals d'entrada s'inicialitzen, és a dir passen a valer el codi 00. Aquest fet es produeix si la porta que calcula les entrades s'inicialitza fent que les seves sortides quedin inicialitzades. El resultat de la funció

es manté vàlid mentre el senyal de petició està actiu, de manera que tenim la porta actuant com un element de memòria.

En el nostre exemple, si les entrades a i b s'inicialitzen i passen a valer el codi 00, tenim que els camins a terra es tallen. Això no afecta a les sortides, ja que el node intern de la funció AND, que era el que valia 1, ja tenia aquest camí a terra tallat, altrament s'hagués descarregat. Pel que fa al node intern de la funció NAND, com que ja s'ha descarregat, mentre el transistor del *pull-up* es mantingui tallat no es carregarà i mantindrà el resultat de la funció NAND a 1. Quan el senyal petició passi a valer 0 tornarem a ser a la *fase d'inicialització*.

3.3.3.2 Detecció de final d'avaluació: portes C de Muller

Un altre punt important és la detecció del final d'avaluació de la porta. Ja hem dit que els circuits autotemporitzats comuniquen a l'entorn que el càlcul que estaven fent ja està disponible activant el senyal d'acabament. Veurem com generar el senyal acabament amb lògica DCVSL.

Quan la porta només té una sortida, generar el senyal d'acabament consisteix a fer la OR de les dues funcions de sortida (la que avalua la funció i la que avalua la funció negada). La figura 3.5.d mostra com generar el senyal d'acabament per a la porta AND anterior. A la fase d'inicialització, quan totes dues funcions passen a valer 0, el senyal d'acabament també valdrà 0. En la fase d'avaluació, quan una de les funcions passa de valer 0 a 1, la sortida de la OR també canvia de 0 a 1 activant el senyal d'acabament. Un cop es desactiva el senyal de petició i es tornen a inicialitzar les funcions de sortida, la desactivació del senyal d'acabament indica la finalització de la inicialització. D'aquesta manera queda definit un protocol de quatre fases entre els senyals de petició i acabament.

Un element important en els circuits asíncrons és la porta C de Muller. La figura 3.6 mostra el símbol utilitzat per representar una porta C de Muller i una possible implementació en CMOS [Men91, Wil91]. Una porta C de Muller de dues entrades implementa la funció $C = AB + BC_o + AC_o$ on A i B són les entrades, C_o és el valor anterior de la sortida i C és el valor de la sortida actual. Així doncs, la sortida d'una porta C de Muller només canvia el valor de la sortida si totes les entrades valen el mateix, altrament manté el valor

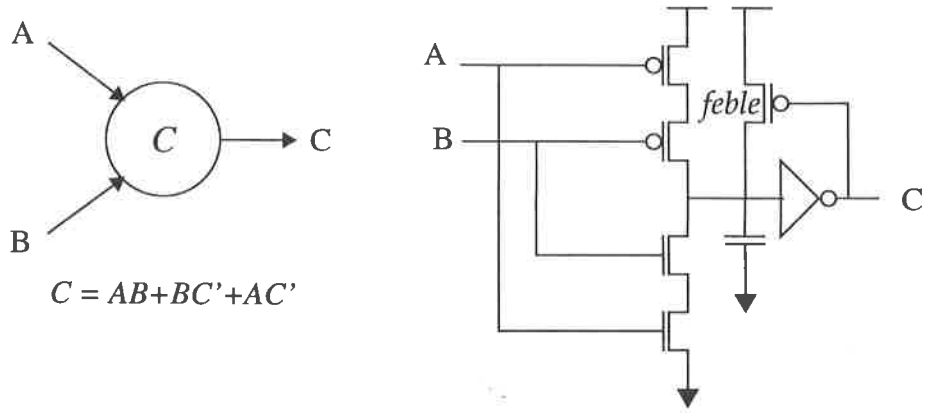


Figura 3.6: C de Muller: Símbol i implementació CMOS

anterior.

En el cas de portes complexes amb més d'una sortida, pot passar que les diferents funcions avaluïn en temps diferents. En aquest cas, caldrà fer la OR de cadascuna de les sortides amb la seva complementària. Els resultats de les OR seran les entrades d'una porta C amb tantes entrades com funcions de sortida tingui la porta. Si el nombre d'entrades és molt gran, es pot utilitzar de manera equivalent un arbre de portes C de dues entrades. La figura 3.7 mostra un exemple de com es genera el senyal d'acabament quan la porta té més d'una funció de sortida utilitzant una única porta C de n entrades o un arbre de portes C de dues entrades.

El funcionament utilitzant una única porta C es detalla a continuació (el cas de l'arbre és equivalent). Quan totes les sortides estan inicialitzades, les sortides de totes les portes OR valen 0. En aquest moment, la porta C val 0, és a dir, el senyal d'acabament val 0. A mesura que les funcions van avaluant, algunes portes OR passen a valer 1, però la porta C no canvia fins que totes valen 1. Quan totes les sortides són vàlides, la sortida de totes les portes OR val 1 i la porta C avalua 1, fent que el senyal d'acabament valgui 1 també.

En fer la inicialització de la porta, el procés és semblant. Quan el senyal de petició és 0, la porta comença a inicialitzar-se. Les sortides passen a valer el codi 00 d'inicialització en moments diferents. Mentre alguna de les sortides no s'hagi inicialitzat, el senyal d'acabament és manté a 1. Quan totes les sortides s'hagin inicialitzat, totes les portes

OR passen a valer 0 i el senyal d'acabament es desactiva.

3.3.4 Graf d'estats d'una porta DCVSL

La figura 3.8 mostra el graf d'estats per a una porta DCVSL genèrica. Aquest graf d'estat es correspon amb el comportament descrit a la secció anterior.

Suposem que partim de l'estat 1 en què el senyal de petició val 0 i les sortides estan inicialitzades. D'aquest estat podem passar a dos de diferents, segons quin dels dos esdeveniments següents es produeixi primer:

1. L'activació del senyal petició (passem a l'estat 3).

2. Que les entrades esdevinguin vàlides (passem a l'estat 2).

En qualsevol dels dos estats (3 o 2) les sortides no varien, ja que mentre les entrades no siguin vàlides la porta no pot avaluar i, mentre el senyal de petició valgui 0 encara que les entrades siguin vàlides, tampoc. Dels estats 3 i 2 passem a l'estat 4 quan el senyal de petició val 1 i quan les entrades són vàlides. En aquest estat, la porta avalua i quan les sortides passen a ser vàlides passem a l'estat 5 (fase d'avaluació).

En l'estat 5 poden tornar a passar dues coses diferents:

1. Que el senyal de petició es desactivi.

2. Que les entrades s'inicialitzin.

En el primer cas, passem de nou a la fase d'inicialització sense passar per la de memorització (estat 6) i en el segon passem a la fase de memorització (estat 7).

En l'estat 7, quan el senyal de petició passa a valer 0, la porta passa a l'estat 9 —fase d'inicialització— i passat un temps de latència quan les sortides s'inicialitzen (codi 00) es passa a l'estat 1.

De l'estat 6, quan les sortides s'inicialitzen, la porta passa a l'estat 8. Quan les entrades s'inicialitzen, es passa a l'estat 1, completant el cicle.

Aquest és el funcionament "correcte" de la porta, però hi ha alguns estats en què l'ocurrència d'algun esdeveniment no esperat pot portar a un funcionament incorrecte de la porta. Aquest és el cas, per exemple, de l'estat 10 en el qual, tot i que les entrades ja

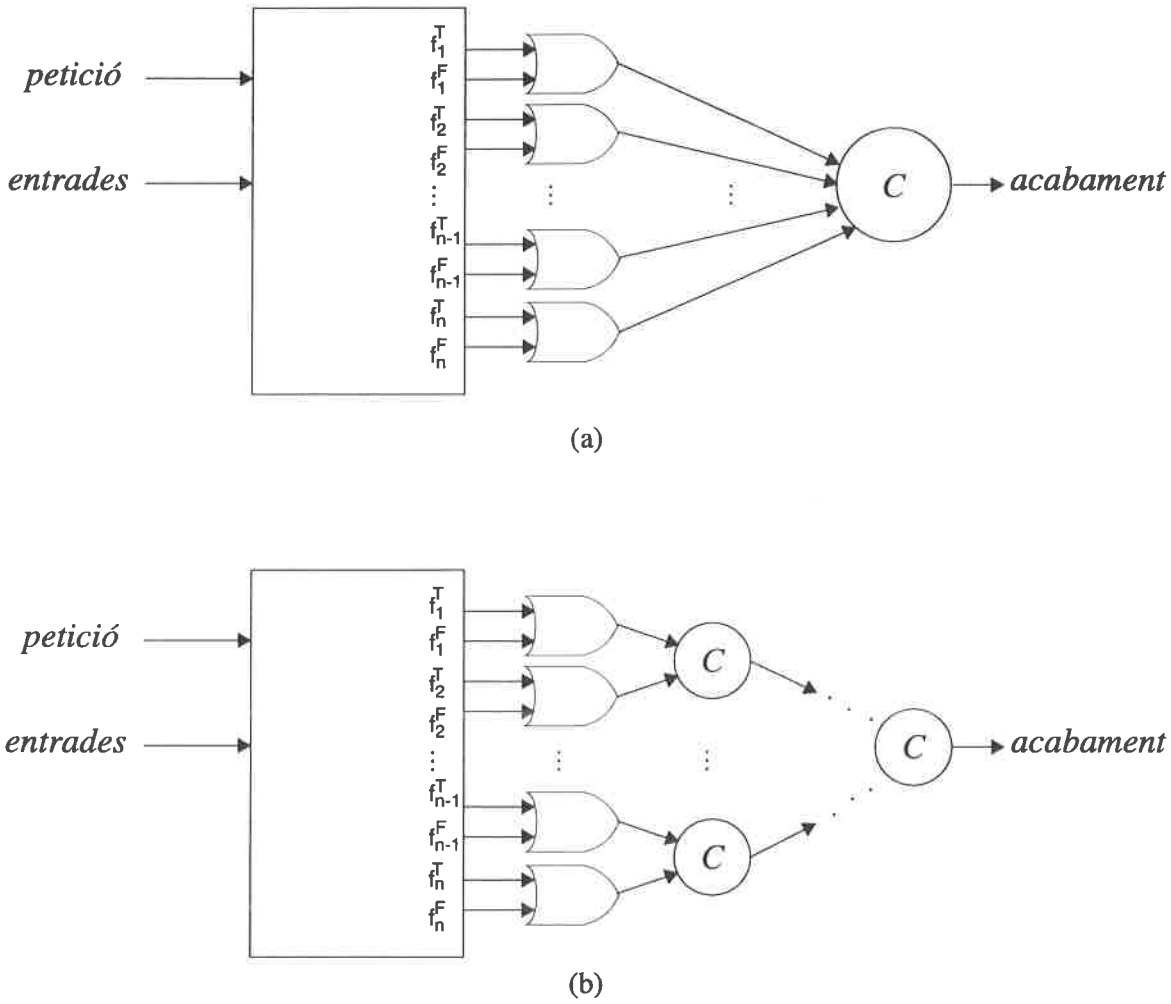


Figura 3.7: (a) Generació del senyal acabament amb una porta C de n entrades; (b) Generació del senyal acabament amb un arbre de portes C de 2 entrades

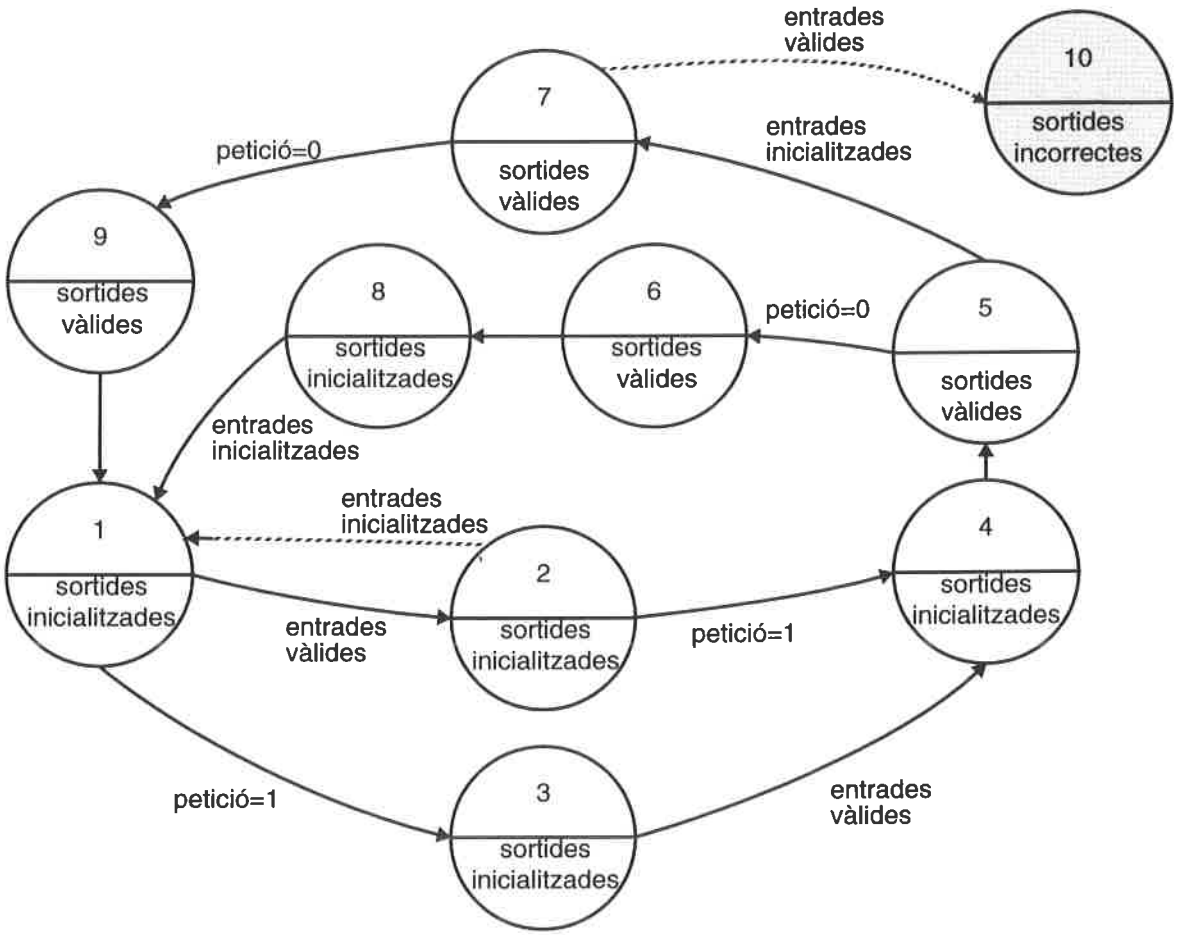


Figura 3.8: Graf d'estats d'una porta DCVSL

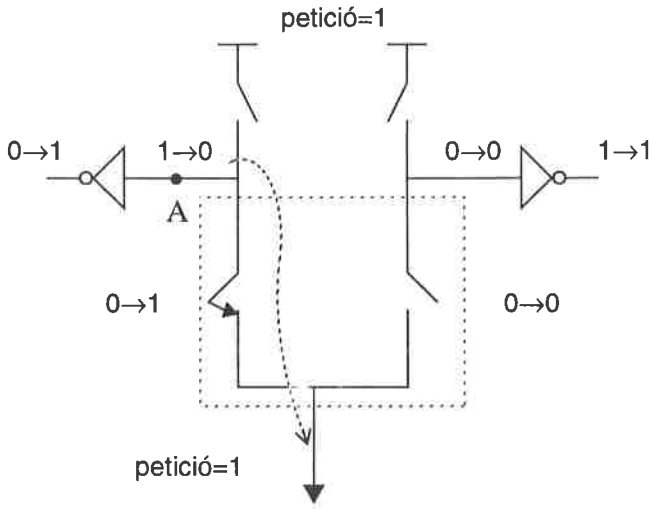


Figura 3.9: Exemple de com una porta pot passar a un estat amb sortides incorrectes si el senyal de petició no s'activa i desactiva correctament

no són vàlides, el resultat es manté. En aquest estat, si per error les entrades passessin a ser vàlides —això podria ocórrer si per exemple la porta predecessora generés un nou valor vàlid abans de temps—, la porta podria passar a un estat de sortides incorrectes (codi 11). Per exemple, en la figura 3.9, si el camí que va del node A a terra passa d'estar tallat a conduir, el node A es descarregaria. En conseqüència, la sortida corresponent passaria de 0 a 1. És evident que aquest comportament és indesitjat i que el control haurà de sincronitzar els diferents mòduls del circuit perquè això no passi.

Un altre cas que no és desitjable però que no es tan crític és el que es produeix a l'estat 2 si les entrades que ja eran vàlides són inicialitzades —la porta predecessora s'inicialitza sense esperar a que el càlcul acabi—. En aquest cas, passem de l'estat 2 a 1.

La taula 3.3 mostra per a cada estat del graf anterior quin valor té el senyal de petició de la porta i si les entrades són vàlides o estan inicialitzades.

3.3.5 Element de memòria amb lògica DCVSL

Una porta que tindrà algunes característiques diferents als altres mòduls és el *latch*. La figura 3.10 mostra l'esquema del *latch* que utilitzarem. En aquest cas, el senyal de petició ac-

Estat	Entrades	*petició*
1	inicialitzades	0
2	vàlides	0
3	inicialitzades	1
4	vàlides	1
5	vàlides	1
6	vàlides	0
7	inicialitzades	1
8	vàlides	0
9	inicialitzades	0
10	vàlides	1

Taula 3.3: Entrades i senyal de petició en els diferents estats d'una porta genèrica DCVSL

tua com a senyal de càrrega i el senyal d'acabament indica que l'operació d'emmagatzemament ha acabat. Aquest *latch* no sols manté la sortida amb el valor emmagatzemat encara que les entrades s'inicialitzin (passen a valer el codi 00), sinó que si després d'inicialitzar-les prenen un altre valor vàlid, si el senyal de petició és actiu, la porta manté el valor de sortida inicial.

Això ens permetrà no sols inicialitzar la porta anterior, sinó que podem continuar amb altres càlculs mentre el *latch* manté el resultat anterior.

La figura 3.11 mostra el graf d'estats del *latch* i podem apreciar que és lleugerament diferent al de les altres portes, atès el seu comportament singular.

En l'estat 1, quan el senyal de petició s'activa, es passa a l'estat 2 si les entrades encara no són vàlides i a l'estat 3 si ja ho són. La transició de l'estat 2 al 3 es produeix quan les entrades són vàlides.

A l'estat 3, després d'un temps d'emmagatzemament les sortides passen a ser vàlides i es passa a l'estat 4. De l'estat 4 es passa al 5 quan les entrades són inicialitzades o al 7 si el senyal de petició passa a ser 0 abans que les entrades canviïn. De l'estat 7 es passa a l'1 quan les sortides s'inicialitzen.

De l'estat 5 es passa al 6 quan el senyal de petició es desactiva, i un cop les sortides

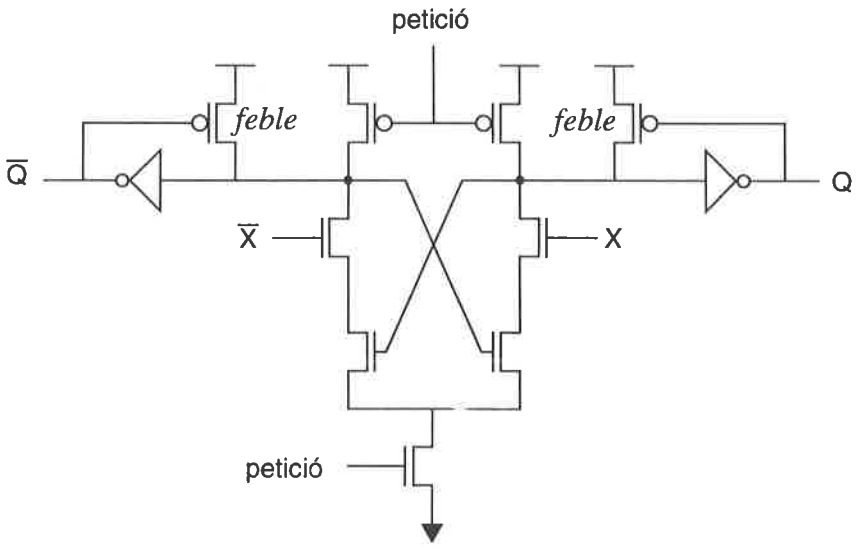


Figura 3.10: Latch *en lògica DCVSL*

s'inicialitzen el *latch* passa a l'estat 1.

3.3.6 Encadenament de mòduls insertant elements de memòria

En aquest apartat veurem com podem encadenar mòduls DCVSL amb *latchs* i la successió d'estats pels quals passen. Veurem un exemple dels possibles estats pels quals podrien passar una porta i un *latch* que estan encadenats (figura 3.12).

Suposarem que inicialment les entrades ja són vàlides i que els senyals de petició d'ambdues portes són 0. A partir d'ara parlarem de p_p per indicar el senyal de petició de la porta i de p_l per indicar el senyal de petició del *latch*. També, per evitar confusions amb els identificadors dels estats, subindexarem amb una p els estats de la porta i amb una l els estats del *latch*. Així, doncs, quan parlem de l'estat 4_l sabrem que es tracta de l'estat 4 del *latch*.

Per les condicions inicials que hem suposat, l'estat inicial de la porta és el 2_p i el del *latch*, l'1_l. Els senyals de petició de les dues portes es poden activar simultàniament, de manera que quan el valor de sortida de la porta sigui vàlid, el *latch* comença a emmagatzemar-lo. En activar p_l, com que les entrades del *latch* encara no són actives, el *latch* passa a l'estat 2_l.

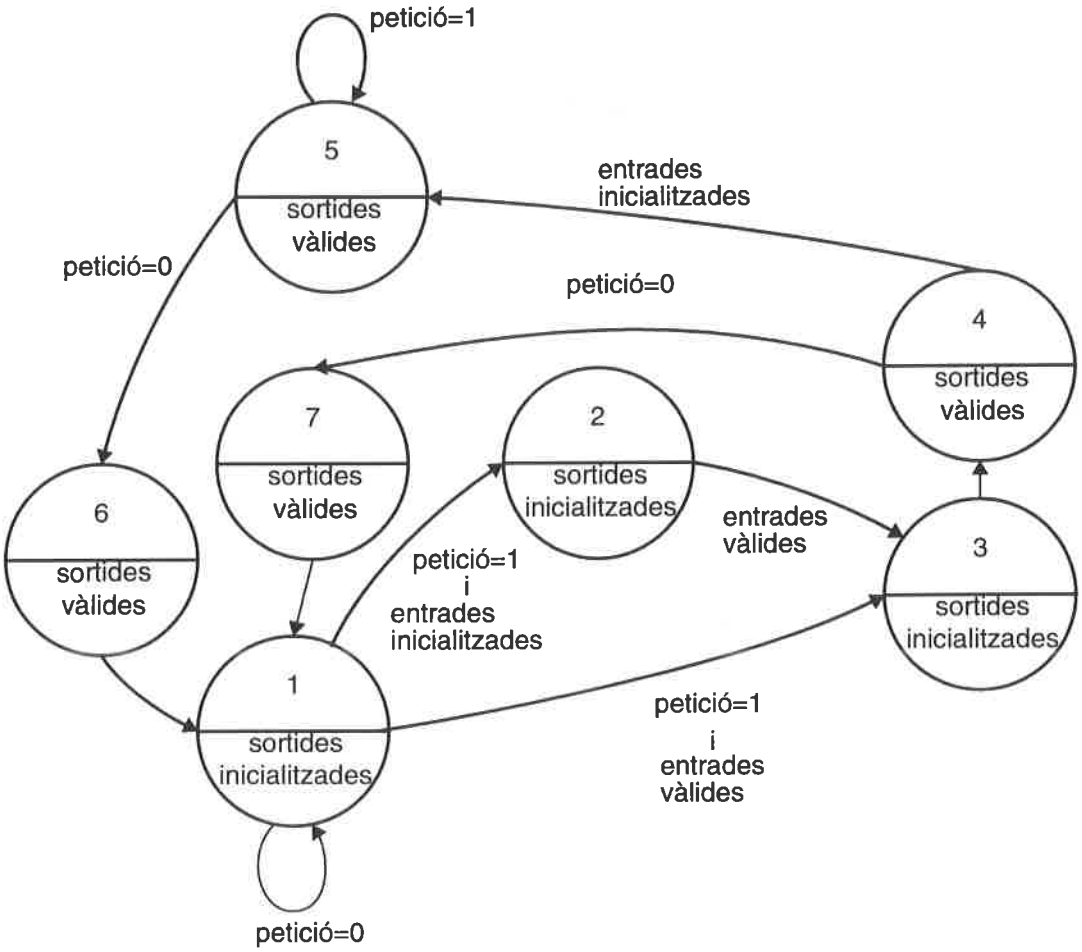


Figura 3.11: Graf d'estats del latch

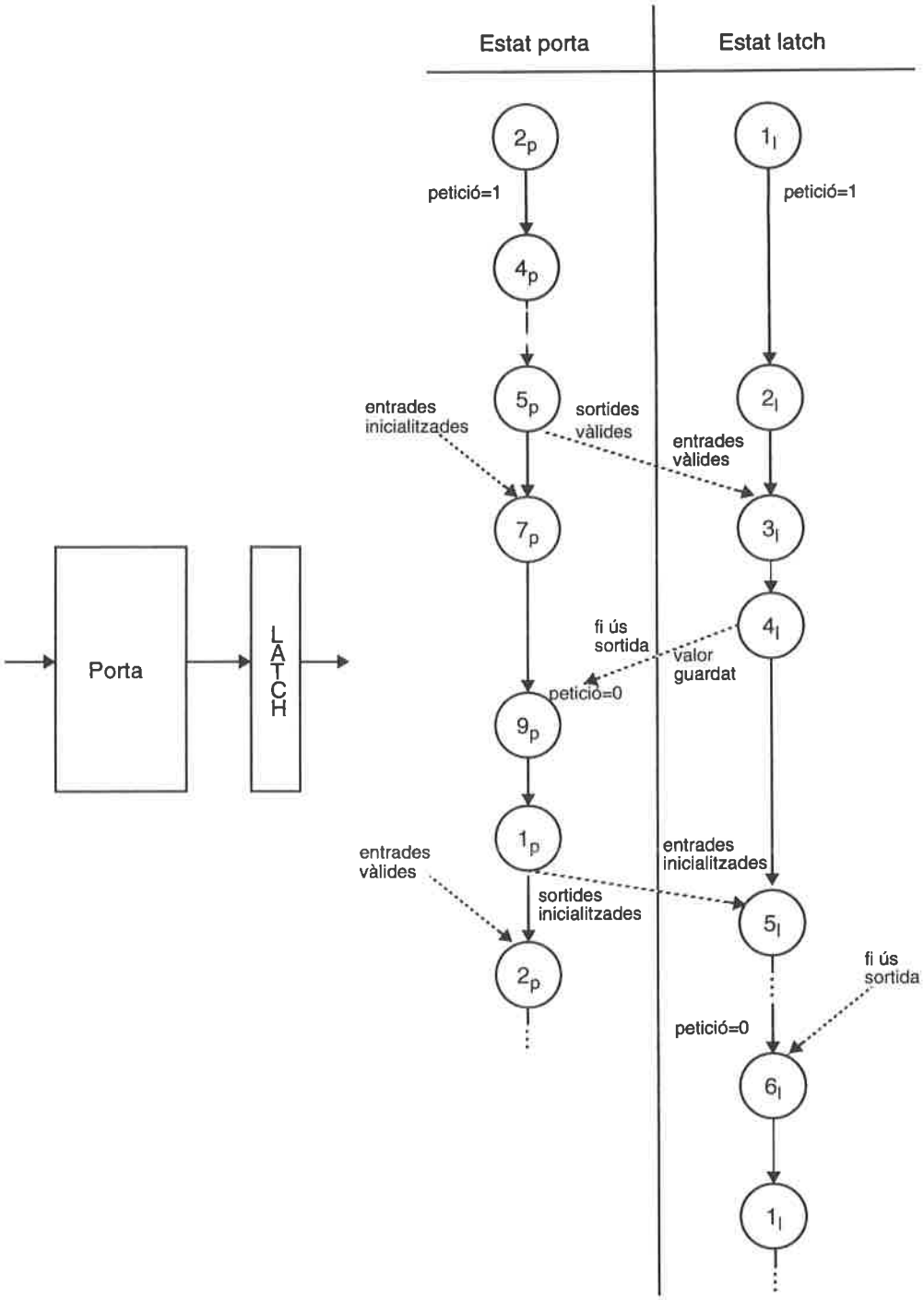


Figura 3.12: Exemple de successió d'estats en l'encadenament d'una porta genèrica i un latch

En activar el p_p, la porta passa a l'estat 4_p i, passat el temps de càlcul (depèn de les dades d'entrada), les sortides seran vàlides i passa a l'estat 5_p. El fet que les sortides de la porta siguin vàlides significa que les entrades del *latch* també ho són, ja que es tracta del mateix senyal. Això fa que el *latch* passi a l'estat 3_l i emmagatzemi el resultat del càlcul de la porta. Un cop s'ha emmagatzemat es passa a l'estat 4_l.

Si el predecessor de la porta és inicialitzat, les entrades s'inicialitzen, de manera que es passa a l'estat 7_p en què es memoritza el resultat.

Quan el resultat ja està emmagatzemat, podem inicialitzar la porta i utilitzar-la per a altres càlculs, ja que el resultat del càlcul anterior està emmagatzemat en el *latch*. En desactivar el senyal p_p, la porta passa a l'estat 9_p d'inicialització, i quan les sortides s'han inicialitzat passa a l'estat 1_p. Des d'aquest estat, si les entrades tornessin a ser vàlides, passaria a l'estat 2_p i tornaria a començar un nou cicle.

Pel que fa al *latch*, quan la porta passa a l'estat 1_p, les sortides (que són les entrades del *latch*) passen a estar inicialitzades, fet que provoca la transició a l'estat 5_l, en què les sortides continuen vàlides. El *latch* es manté en aquest estat, encara que les entrades oscil·lin. Quan el valor emmagatzemat en el *latch* ja no es necessita, podem inicialitzar-lo posant a 0 el senyal p_l de manera que tornem a l'estat inicial del *latch* (1_l) després de passar per l'estat 6_l.

3.3.7 Memorització sense latchs

A continuació veurem com podem encadenar portes DCVSL sense insertar *latchs* aprofitant la seva fase de memorització [Wil91]. De nou ho farem sobre un exemple. En aquest cas, suposarem que tenim dues portes DCVSL que volem encadenar. Suposarem que la segona porta utilitza els resultats de la primera com a entrades i que aquestes són les úniques dades d'entrada que utilitza. Inicialment les entrades de la primera porta són vàlides i els senyals de petició d'ambdues portes (p_1 i p_2) valen 0. Subindexarem els estats de les portes amb un 1 o amb un 2 segons es tracti de la primera o de la segona porta.

L'estat inicial de la primera porta és el 2_1 (entrades vàlides i $p_1 = 0$) i el de la segona 1_2 (entrades inicialitzades i $p_2 = 0$). El senyal de petició de les dues portes s'activa a la vegada —això permet reduir el temps de càlcul, ja que el temps d'encadenar dues operacions sol ser

inferior al temps total que trigarien separadament les dues portes—. Com que la primera porta ja té les entrades vàlides, passa a la fase d'avaluació i obté uns resultats (estats 4_1 i 5_1).

En canvi, la segona porta passa a l'estat 3_2, ja que les seves entrades encara no són vàlides. Quan la primera porta té les sortides vàlides (que són les entrades de la segona porta) passa a l'estat 4_2 i després d'avaluar, al 5_2.

Com que la primera porta ja té els resultats estables, pot passar que les seves entrades siguin inicialitzades, de manera que passarà a l'estat 7_1 de memorització. Això mateix li pot passar a la segona porta. En aquest cas, les seves entrades s'inicialitzen perquè el senyal p_1 de la primera porta és desactivat i la primera porta s'inicialitza. Com a resultat, tenim que la primera porta després de l'estat 9_1 passa a l'estat 1_1 i la segona a l'estat 7_2.

En aquest punt, pot passar que a la primera porta li arribin noves dades d'entrada vàlides, cosa que faria que la porta passés a l'estat 2_1 de nou, però el senyal de p_1 no pot ser activat encara. El motiu pel qual no podem activar el senyal p_1 és que provocaria que la porta generés unes noves sortides vàlides. Aquestes noves sortides vàlides serien entrades vàlides de la segona porta que podrien provocar que la segona porta passés a l'estat erroni 10_2, en què les sortides que genera són incorrectes. Per tant, la primera porta s'ha d'esperar que la segona estigui a l'estat 1_2 en què no hi ha cap problema si les entrades són vàlides.

A la vegada, la segona porta s'inicialitza desactivant el senyal p_1 (estat 9_2) quan detecta que la sortida generada ja no es necessita més. En aquest punt es podria activar el senyal p_1 de nou, tornant a iniciar un nou cicle de càlcul.

Amb la lògica DCVSL podem encadenar diverses portes sense necessitat d'insertar *latchs* per guardar resultats. Això és degut a la capacitat de memorització que tenen aquestes portes. A la figura 3.14 veiem un exemple de com se succeirien els diferents esdeveniments en un sistema com aquest.

Tot i que podem activar els senyals de petició de dues o més portes a la vegada, sempre acabarà abans la primera porta que la segona, ja que aquesta utilitza els valors calculats a l'anterior per completar el seu càlcul i així successivament. Quan la segona porta acaba el seu càlcul, podem inicialitzar la primera sense que això afecti a la segona, però abans de tornar a activar el petició de la primera hem d'esperar que la segona també s'hagi inicialitzat.

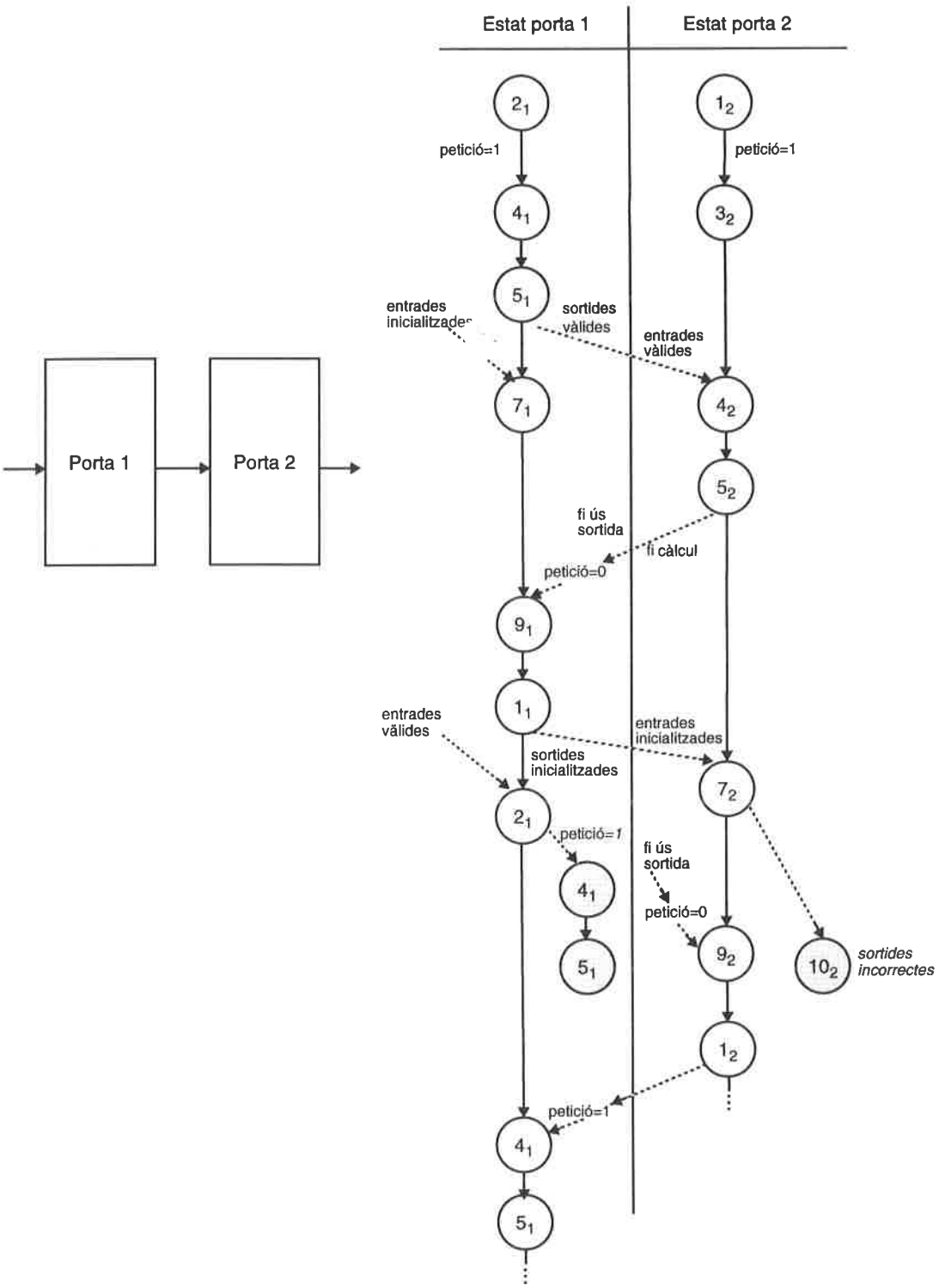


Figura 3.13: Exemple de successió d'estats en l'encadenament de dues portes genèriques

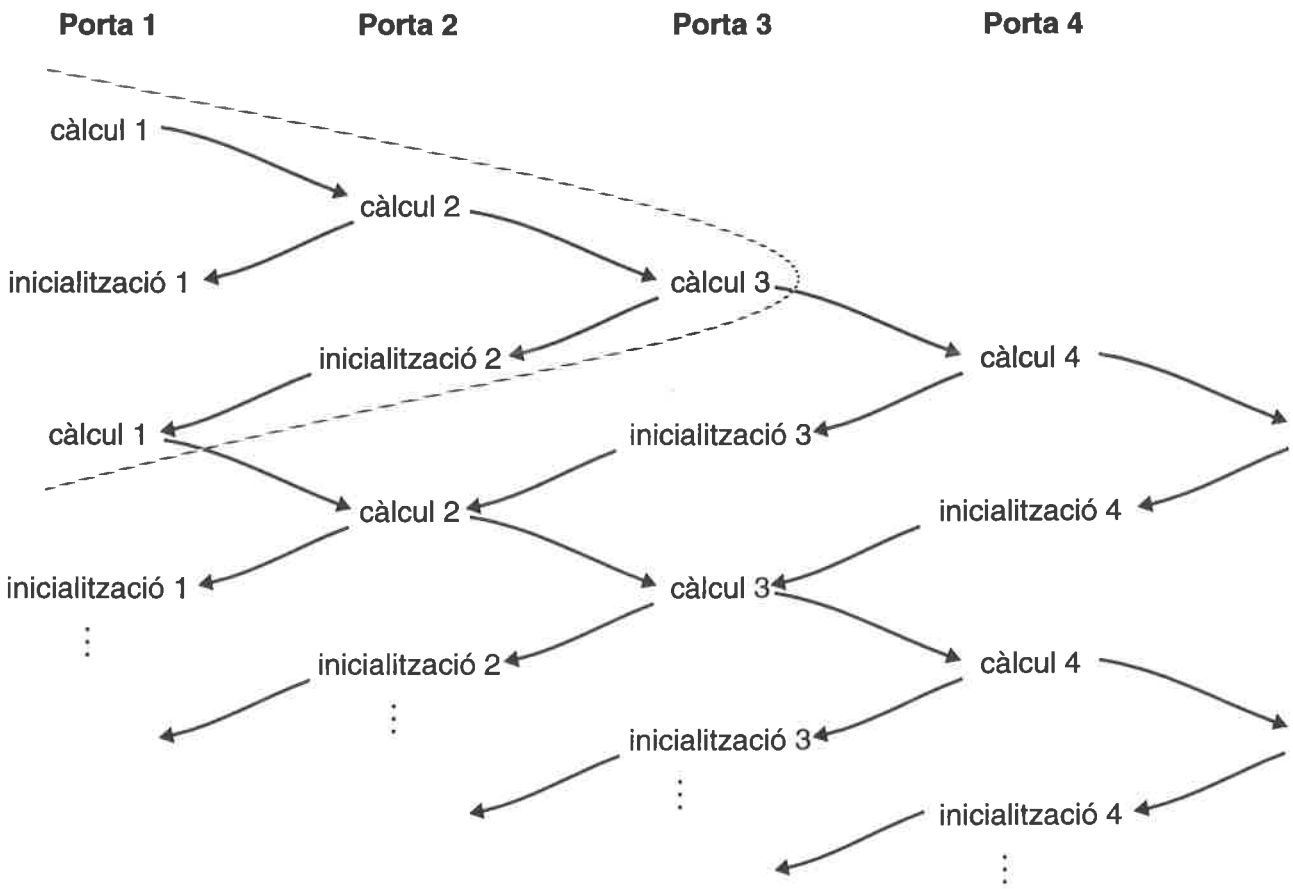


Figura 3.14: Exemple de com podem encadenar quatre portes DCVSL i fer un conjunt de càlculs sense necessitat d'insertar latchs *per emmagatzemar-ne els resultats*

A la figura podem veure que la inicialització de la segona porta depèn del fet que el càlcul de la tercera porta acabi i, per tant, podem tenir una cadena de tres o més portes fent càlculs sense necessitat d'insertar *latchs*.

3.4 Síntesi de circuits de control

Els treballs existents en l'àrea de disseny de circuits asíncrons s'orienten bàsicament a la síntesi de circuits de control. Pel nostre coneixement, només hi ha una proposta de model d'arquitectura asíncrona presentat a [Hir87] on el sistema compila especificacions en ISPS

a circuits amb arquitectura en bus.

El disseny de circuits asíncrons va ser popular als anys 60 i 70 quan la complexitat dels circuits encara no era gaire gran, però va ser desestimada durant molt de temps a causa dels diferents impediments que hi havia a l'hora de la seva realització.

Històricament s'han utilitzat dos models de circuit asíncron:

- **Model de Huffman**, on els circuits es descomponen en un circuit combinacional i un conjunt de senyals realimentats. En aquest model s'associa un retard limitat a cada interconnexió entre portes diferents [Huf54]. Si el valor de més d'un senyal canvia en un mateix instant, poden produir-se curses en els senyals de sortida dels circuits combinacionals. Aquestes curses es poden evitar afegint lògica (tots els implicants primers) i fent que només pugui canviar un senyal en cada instant [Mil65]. En circuits seqüencials, les curses poden portar a errors permanents causant malfuncionaments i sortides incorrectes. Si tots els elements tenen retard limitat, aquests errors es poden evitar afegint lògica redundant i fent que els elements de retard en els senyals realimentats suficientment grans, de manera que dóna temps als circuits combinacionals a estabilitzar-se abans de canviar d'estat [Mil65, Mag71, Ung71].

- **Model de Muller**, on el circuits es descomponen en portes amb interconnexió arbitrària. S'assigna un retard il·limitat, però finit, a cada sortida de les portes [MB59]. La síntesi de circuits asíncrons utilitzant aquest model fa servir detectors de dades i espaiadors [AFM69]. S'utilitzen esquemes de codificació per les dades, de manera que els detectors de dades indiquen quan les sortides són estables. Aquestes propostes redueixen l'eficiència al 50%, ja que gairebé la meitat del temps s'ha de dedicar a inicialitzar els elements del sistema.

Es pot fer una classificació dels diferents estils de disseny de circuits asíncrons segons la naturalesa de l'especificació del control. Hi trobem quatre tendències: metodologies basades en màquines d'estats, micropipelines, metodologies basades en llenguatges de programació i metodologies basades en xarxes de Petri. A continuació, farem una revisió de les diferents propostes existents per a aquestes quatre tendències.

3.4.1 Mètodes basats en màquines d'estat

Tot i que els mètodes basats en màquines d'estat han estat molt populars en les darreres dècades [Huf57, MB59, Ung71, MFR85], últimament són més habituals els mètodes basats en xarxes de Petri o els mètodes basats en llenguatges de programació. Així i tot, encara hi ha algunes propostes recents, com la de Nowick i Dill [ND91] que cal destacar.

L'objectiu d'aquest mètode és definir un estil de disseny el més proper possible al dels sistemes síncrons, però amb els avantatges dels circuits asíncrons. Per fer-ho, es dissenyen màquines d'estat localment síncrones amb cicles de duració indefinida que determinen els canvis d'estat. Aquest mètode de disseny permet que hi hagi més d'un canvi a les entrades (*ràfega d'entrada*) i a cada canvi d'estat es canvien els valors d'un conjunt de les sortides (*ràfega de sortida*).

La metodologia comença definint un diagrama d'estats de la màquina de control que es vol dissenyar. Cada arc de transició entre estats s'etiqueta amb un conjunt d'entrades (*ràfega d'entrada*) i amb un conjunt de sortides (*ràfega de sortida*). Des d'un estat determinat, quan totes les entrades de la ràfega d'entrada han canviat, el sistema genera la corresponent ràfega de sortida i canvia d'estat.

Tot i que les transicions en les entrades poden venir en qualsevol ordre, tenim la restricció que només poden produir-se canvis en els senyals indicats en la ràfega d'entrada. Altres restriccions són que cap ràfega d'entrada d'un estat pot ser un subconjunt d'un altra ràfega d'entrada i que cada estat té un únic punt d'entrada.

Les màquines d'estats que s'implementen amb aquesta metodologia es componen de lògica combinacional, elements de memòria controlats pel rellotge i variables d'estat que es realimenten a la lògica combinacional. El rellotge local s'utilitza per eliminar alguns riscos possibles a part de controlar el canvi d'estat. La síntesi es fa construint taules d'estats per al senyal del rellotge, per a les variables d'estat i per a les sortides, i s'utilitzen tècniques per minimitzar el nombre d'estats i nombre de canvis d'estat.

3.4.2 Micropipelines

El mètode de *micropipelines* va ser proposat per Sutherland [Sut89]. Es basa en la implementació dels circuits amb elements de càlcul segmentats, utilitzant protocols de comuni-

cació de dues fases. S'utilitza el model de dades compactades. L'inconvenient d'aquesta proposta rau en el fet que s'ha de fer un estudi detallat dels retards de les unitats funcionals. D'altra banda, els circuits combinatoris són molt eficaços, ja que la seva implementació es basa en els models síncrons tradicionals.

3.4.3 Mètodes basats en llenguatges de programació

En aquest apartat es descriuen aquelles metodologies de disseny de circuits asíncrons basades en la traducció dirigida per la sintaxi Aquestes metodologies no poden considerar-se síntesi d'alt nivell ja que no intenten optimitzar el disseny, sinó que simplement tradueixen d'una descripció en un lleguatge determinat. Aquests mètodes, tot i que són molt elegants i robustos, tenen l'inconvenient que el circuit resultant pot ser pitjor que l'obtingut amb altres mètodes, ja que no s'optimitza. El circuit resultant depèn linealment de l'especificació donada. Els llenguatges que s'utilitzen són lleguatges de comunicació de processos com ara CSP [Hoa89], Tangram [vBKR+91] o Occam. La primera d'aquestes metodologies que es descriu és la proposada per Martin a [Mar86] i la segona és la que proposa van Berkel a [vBKR+91].

3.4.3.1 Compilació de CSP a circuits (Martin 1986)

En aquest mètode el comportament del circuit es descriu amb un conjunt de processos comunicants amb la notació proposada a [Mar85], que és molt semblant al CSP de Hoare [Hoa89]. El circuit final es compon d'una xarxa d'operadors, cadascun d'aquest són elements del conjunt d'operadors del que en direm el *codi objecte.* Aquest conjunt d'operadors podria estar compost, per exemple, per una porta AND, una porta OR, l'element C de Muller, un cable i un *fork.*

El mètode de compilació es pot descompondre en quatre fases:

- Descomposició de processos: es reemplacen els processos de la descripció inicial per diversos processos equivalents. Es pretén obtenir una descripció en què la banda dreta de cada guarda sigui una única línia de programa, és a dir, consisteixi únicament en assignacions simples i primitives de comunicació compostes per comes i punts i comes. Per fer-ho, s'aplica una regla de descomposició.

- Expansió en protocols de quatre fases: en aquesta fase es reemplaça cadascun dels canals de comunicació existents en la descripció dels processos per un protocol de quatre fases.

- Expansió en regles de producció: a continuació es compila l'expansió en protocols de quatre fases a un conjunt de regles de producció on tots els seqüenciaments explícits són eliminats.

- Reducció d'operadors: l'última fase del procés consisteix a identificar les regles de producció amb aquelles que descriuen la semàntica dels operadors, de manera que el programa es tradueix a una xarxa d'operadors.

Com que els circuits són correctes per construcció i les guardes de les regles de producció són estables per construcció, els circuits no tenen riscos.

3.4.3.2 Compilació de Tangram a circuits

El llenguatge Tangram desenvolupat a Philips Research està basat en el CSP de Hoare [Hoa89] i la tècnica que es descriu a continuació és molt semblant a la descrita a l'apartat anterior. La idea bàsica és construir circuits a partir de components elementals partint d'una descripció en llenguatge Tangram.

El procés de traducció es descompon en dues fases:

- Traducció de Tangram a circuits amb sincronització explícita: per fer-ho, es defineix una funció de traducció C del domini dels programes en Tangram al domini dels circuits amb sincronització explícita. Per a cada regla de producció definida en Tangram, la funció C descriu un regla de traducció. Així doncs, aquesta traducció està clarament dirigida per la sintaxi. Els circuits amb sincronització explícita obtinguts són insensibles als retards per construcció.

- Traducció de circuits amb sincronització explícita a circuits VLSI: abans de fer la traducció en si es fan algunes optimitzacions i refinaments per simplificar-los. Després es descompon l'especificació en primitives VLSI disponibles com puguin ser inversors o portes NAND.

3.4.4 Mètodes basats en xarxes de Petri

Un altre grup són aquelles metodologies de disseny basades en xarxes de Petri i, més concretament, en *grafs de transicions de senyals* (STG). Els STG són una subclasse de xarxes de Petri [Pet62] interpretades que varen ser proposades simultàniament per Chu [Chu87] i per Rosenblum *et al.* [RY85]. Els STG permeten descriure relacions de causalitat i concurrència entre els senyals de manera semblant a com es fa en un diagrama de temps.

Aquestes tècniques utilitzen STG per descriure el comportament dels circuits asíncrons mitjançant la descripció de la seqüència de transicions dels seus senyals.

Com es veurà en el proper capítol, la metodologia de disseny proposada en aquest treball utilitza STG per descriure el control. Per aquest motiu, en el següent capítol es descriuen amb detall les tècniques basades en xarxes de Petri.

3.5 Experiències en el disseny de circuits asíncrons

A continuació farem una revisió dels circuits asíncrons que s'han realitzat i que s'han descrit en la literatura.

3.5.1 Circuit divisor

A [WHAY87], Williams i altres presenten un xip per fer divisions en punt flotant utilitzant l'algorisme de divisió SRT. El xip es dissenya amb una estratègia intermitja entre un disseny totalment iteratiu, que seria barat en àrea però amb un temps d'execució llarg, i un disseny combinacional, on es requereix molta àrea, però el temps d'execució és molt curt. La proposta implementada és un petit vector dels elements aritmètics necessaris per cada etapa disposats en forma d'anell. Cada element és un mòdul autotemporitzat amb les dades codificades amb doble via i els senyals de control necessaris són generats internament. El vector itera diverses vegades per calcular el quocient amb la precisió desitjada. Si l'anell té suficients etapes no serà necessari afegir registres ja que les dades de l'última etapa seran vàlides durant el temps suficient per tornar a alimentar la primera etapa.

El xip és implementat amb tecnologia CMOS de 3 μm amb dos nivells de metall. L'àrea del xip és de $6,6 \times 1,4$ mm sense els *pads* i té 13.000 transistors. El xip és capaç de generar

en mitjana un bit del quocient cada 22 ns. Una versió millorada en tecnologia de 2 μm genera en mitjana un bit de quocient cada 10 ns.

3.5.2 Filtres lattice

Meng i altres presenten a [MBM90] el disseny d'un conjunt de xips per implementar filtres adaptatius *lattice* d'alta freqüència. La idea és implementar un conjunt reduït de xips que permetin implementar filtres adaptatius de qualsevol longitud. Segons la longitud del vector del filtre que es vulgui implementar i la freqüència a què hagi d'anar, s'haurà d'utilitzar un nombre determinat de xips de cada tipus. El conjunt de xips es compon de quatre xips de càlcul i un cinquè xip que implementa dos *pipelines* de registres de longitud variable. Cadascun dels xips és implementat amb lògica DCVSL (doble via). Els xips són implementats en tecnologia CMOS de $1,6$ μm. Els autors argumenten que un disseny síncron seria indubtablement més ràpid que el que es proposa a l'article, però d'altra banda el disseny d'aquests xips asíncrons és molt més senzill que en el cas del síncron, ja que l'absència de rellotge simplifica molt el disseny d'aquests circuits.

3.5.3 Comparació síncron/asíncron

A [ART93], Auletta i altres presenten una comparació en el disseny de màquines d'estats finits amb camí de dades (FSMD) síncrones i asíncrones. Per fer aquesta comparació es dissenya amb la mateixa especificació, la mateixa llibreria de *standard cells* i les mateixes eines de disseny de *layout* i de simulació dos dissenys, un de síncron i un d'asíncron. Els xips són implementacions d'un algorisme per factoritzar nombres de 16 bits. Ambdós dissenys són construïts a partir de descripcions de màquines d'estat en forma algorítmica. El disseny asíncron s'implementa amb mòduls amb codificació de doble via i utilitzant la metodologia de Martin [Mar92]. El resultat de la comparació és que el disseny asíncron és inferior, tant si es fa un disseny optimitzant en àrea com si es fa un disseny optimitzant en temps d'execució. Els autors tenen com a objectiu futur determinar com canviaria la comparació per dissenys més grans on els retards del control representin un percentatge inferior en el temps d'execució de cada iteració.

3.5.4 Memòria cache

A [NDDH93], Nowick i altres presenten la implementació d'una cache de segon nivell d'un sistema basat en un processador RISC asíncron utilitzant la metodologia descrita a [ND91]. Aquesta metodologia consisteix a dissenyar el control dels circuits amb màquines d'estat amb rellotge local. El rellotge local és utilitzat per eliminar possibles riscos, però els canvis d'estat no depenen d'aquest rellotge local sinó de canvis en les dades d'entrada. Apliquen aquesta metodologia per realitzar el controlador de la cache com una màquina d'estat localment síncrona. Aquest treball demostra que és possible realitzar circuits asíncrons *grans*. El disseny final obté una millora d'aproximadament un 100% sobre realitzacions síncrones equivalents.

3.5.5 Post Office

Coates i altres presenten a [CDS93] el disseny d'un xip anomenat Post Office per suportar comunicacions en el sistema paral·lel Mayfly. El disseny del xip es va fer, com en el cas anterior, utilitzant la metodologia descrita a [ND91] basada en la utilització de màquines d'estat localment síncrones. El Post Office ha estat fabricat utilitzant tecnologia CMOS de $1,2\ \mu m$. El circuit conté 300.000 transistors i ocupa una àrea de $11 \times 8,3\ \text{mm}^2$. El circuit està compost de 96 màquines d'estat que ocupen el 19% de l'àrea del xip, un camí de dades construït amb mòduls autotemporitzats utilitzant codificació compactada de les dades que ocupa el 45% de l' àrea i el connexionat i els *pads* que ocupen el 33% de l'àrea. La importància d'aquest exemple, igual que l'anterior, rau en el fet que s'ha demostrat que es poden construir circuits asíncrons d'altes prestacions.

3.5.6 Circuits dissenyats des de Tangram

Als laboratoris de Philips Research han desenvolupat diferents circuits a partir d'especificacions en Tangram. Un d'aquests és el *descodificador d'errors* per al *Digital Compact Cassette (DCC)* [Kes93]. Una primera aproximació d'aquest circuit es pot trobar a [KvBB+92].

La descripció del circuit consisteix en 430 línies de codi en Tangram. El programa s'ha traduït a un circuit insensible als retards on totes les comunicacions estan basades

en protocols de quatre fases i les dades es codifiquen amb doble via. El circuit té 44.000 transistors i ocupa $11,5\ mm^2$ en tecnologia CMOS de 1 μm. Aquest circuit ha estat dissenyat per a un consum d'energia baix, que en el pitjor dels casos és de $8,3\ mW$. En mitjana, el consum és de $2,5\ mW$. El circuit va ser millorat posteriorment per obtenir un consum encara més baix. El resultat final és un estalvi d'un 40% d'energia ($1,5\ mW$ en mitjana) a un cost d'un increment en àrea del 5%.

3.5.7 Processador ARM

Aquest processador ha estat desenvolupat a la Universitat de Manchester pel grup AMULET. Es tracta d'una versió asíncrona del processador ARM, dissenyat per ACORN computers. La versió síncrona del ARM va ser dissenyada per ACORN Computers els anys 1983-84. Les seves principals característiques són:

- Disseny senzill (25.000 transistors)
- Arquitectura RISC de 32 bits
- Econòmic
- Baix consum (100 Mips/Watt)

La versió asíncrona es basa en el disseny amb micropipelines de Sutherland. La primera versió d'aquest processador, AMULET1, està acabada. Actualment el grup de Manchester està millorant el seu rendiment sobre una segona versió, l'AMULET2.

3.5.8 Processador RISC en GaAs

A [TMB93] es presenten diferents tècniques per al disseny de circuits asíncrons en arseniur de gali utilitzant la metodologia de disseny presentada per Martin [Mar86]. Aquestes tècniques s'han utilitzat per dissenyar un processador RISC de 16 bits. Aquest processador és una modificació del processador en tecnologia CMOS presentat a [MBL+89]. El processador té 16 registres de propòsit general amb quatre busos, dos per lectura i dos per escriptura. El processador es va definir inicialment com un conjunt de processos comunicants. El processador obtingut aplicant el procés de traducció descrit a [MBL+89] té les

següents característiques: 200 Mips amb una dissipació de potència de 2 Watts fabricat amb el procés HGAAS III.

3.6 Conclusions

Els circuits asíncrons es caracteritzen per l'absència de rellotge. En ells, a diferència dels circuits síncrons, no es poden distingir clarament els elements de càlcul dels de memòria, ja que estan interconnectats de forma arbitrària.

Els circuits asíncrons tenen alguns avantatges que els fa atractius:

- Absència del problema de desfasament de rellotge.
- Velocitat mitjana de càlcul.
- Modularitat en el disseny.
- Consum baix.
- Adaptació a les constants físiques.

Però també tenen alguns inconvenients, com són:

- Àrea dels circuits més gran.
- Existència de riscos i curses que compliquen el seu disseny.
- Metastabilitat.

El principal motiu pel qual els dissenyadors han preferit el disseny síncron a l'asíncron és que la tasca d'eliminar tots els possibles riscos i curses d'un circuit és complexa. Per aquesta raó creiem que és important desenvolupar eines de síntesi automàtica de circuits asíncrons.

Els circuits autotemporitzats es caracteritzen per estar governats per dos senyals: el senyals de *petició* i el senyal d'*acabament*. El primer indica que es requereix que el mòdul executi una operació i el segon indica que l'operació ha acabat. Per realitzar els mòduls autotemporitzats existeixen diferents alternatives, com els mòduls amb dades compactades

o el mòduls amb codificació de les dades amb doble via. S'ha presentat una familia lògica que permet construir mòduls autotemporitzats amb codificació de les dades amb doble via: la familia DCVSL. En capítols posteriors utilitzarem aquesta familia per a construir la llibreria de recursos d'un sistema de síntesi d'alt nivell de circuits asíncrons.

Fins ara, la recerca en el camp del disseny automàtic de circuits asíncrons s'ha orientat a la síntesi lògica. S'ha presentat una revisió de les tendències més importants existents a la literatura. També s'ha presentat una revisió de les experiències existents en el disseny de circuits asíncrons.

Capítol 4

Model d'arquitectura asíncrona

En aquest capítol es presenta un model d'arquitectura asíncrona per a la síntesi d'alt nivell de circuits asíncrons. Aquest model es basa en un sistema multiprocessador en què cada processador executa un procés i en què la comunicació entre processadors és totalment asíncrona. Els elements del camí de dades en aquest model són mòduls autotemporitzats. El control és totalment distribuït. Cada element de control es modela amb xarxes de Petri, més concretament amb grafs de transicions de senyals (STG). Dels STG podem passar a circuits asíncrons lliures de riscos mitjançant tècniques existents.

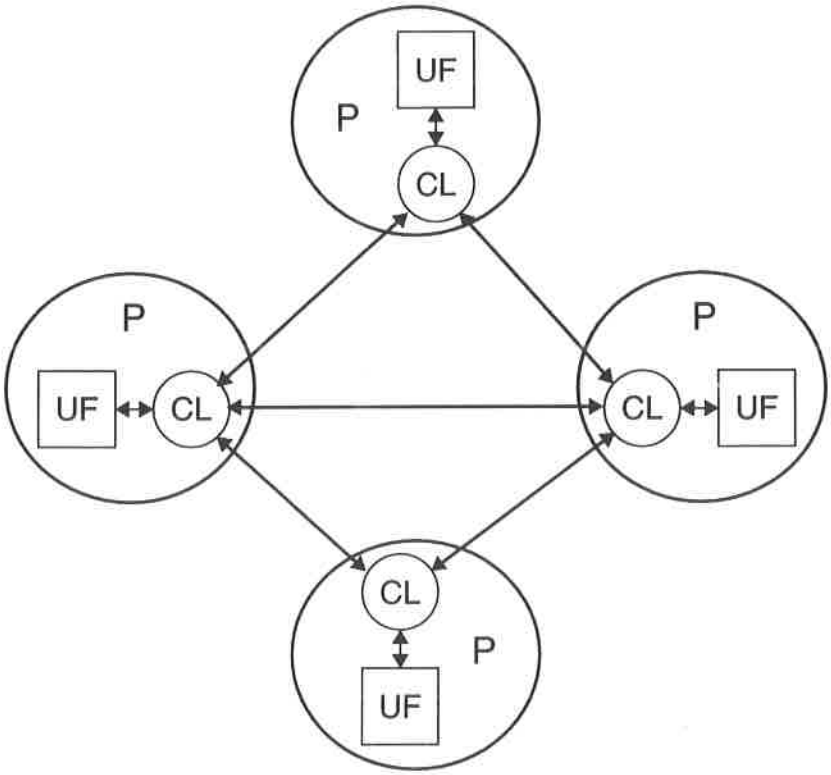


Figura 4.1: Esquema del model d'arquitectura asíncrona

4.1 Introducció

Atès que els algorismes d'alt nivell obtenen una descripció estructural dels circuits, fa necessari establir un model d'arquitectura que permeti realitzar aquesta estructura. Com ja s'ha dit en altres capítols, la síntesi d'alt nivell de circuits asíncrons és una àrea de recerca inexplorada fins al moment i tampoc no s'han definit models d'arquitectura asíncrones que donin suport als seus algorismes.

En aquest capítol es presenta el model d'arquitectura asíncrona publicat a [CB92], basat en un sistema multiprocessador amb pas de missatges, on el control està totalment distribuït en diferents processadors. Cada processador del sistema té dos components: un de càlcul i un de control.

El *component de càlcul* està compost per un o més elements del camí de dades: unitats funcionals, registres, multiplexors, etc.

El *component de control* o *controlador local* (CL), d'una banda controla el component de càlcul i de l'altra es sincronitza amb altres *controladors locals* d'altres processadors quan calen transferències de dades entre ells. La figura 4.1 mostra un esquema d'aquest model d'arquitectura.

El processador executa un procés compost per un conjunt d'operacions. Per exemple, si l'element de càlcul del processador és un sumador, el procés que s'executi en aquest processador serà un conjunt de sumes amb una seqüència preestablerta. El controlador local és l'encarregat de fer que aquesta seqüència de sumes s'executi en l'ordre correcte. També ha de comunicar-se amb altres controladors locals quan les dades d'entrada de les sumes provinguin d'altres processadors o quan el resultat de les sumes s'hagi d'utilitzar per altres càlculs en altres processadors.

Si enlloc del control distribuït s'utilitzés un model amb control central s'haurien desaprofitat alguns dels atractius dels circuits asíncrons. D'una banda, amb un control centralitzat tindríem senyals globals que s'haurien de distribuir a tots els components del circuit. Aquests senyals globals introduirien retards que reduirien el paral·lelisme inherent als sistemes asíncrons. D'altra banda, el nombre d'estats d'una unitat de control creix exponencialment amb el nombre de senyals de control [Chu87].

Els elements del camí de dades són mòduls autotemporitzats. Per construir aquests mòduls en els nostres exemples hem optat per la lògica DCVSL (vegeu capítol 3).

A la següent secció es descriu com es defineixen els controladors locals. A la secció 4.3 es descriuen les propostes existents per fer síntesi lògica de circuits asíncrons a partir de descripcions en STG. A continuació és presenta una metodologia per descriure els controladors locals en STG a partir de descripcions en alt nivell i com podem manipular aquests STG. Finalment es presenten algunes conclusions.

4.2 Control distribuït

Ja s'ha dit amb anterioritat que el model d'arquitectura que es proposa està basat en un control distribuït on cada mòdul o conjunt reduït de mòduls del camí de dades està controlat per un controlador local. Els controladors locals tracten amb dos tipus de senyals de control:

- **Senyals locals:** són aquells senyals necessaris per controlar el mòdul o conjunt de mòduls del processador (senyals de petició, acabament i altres, com per exemple els senyals de selecció d'un multiplexor o el codi d'operació d'una ALU).

- **Senyals globals:** senyals generats per a la sincronització entre diferents controladors locals quan són necessàries transferències de dades entre mòduls de diferents processadors (senyals com per exemple, *sortida vàlida* per indicar que el resultat d'una operació està disponible o *entrada consumida* per indicar que una determinada dada d'entrada ja no és necessària).

Per descriure el comportament dels controladors locals i les transicions dels diferents senyals de control utilitzarem *grafs de transicions de senyals (STG)*. Un cop tinguem tots els controladors locals descrits amb STG podem obtenir un circuit lliure de *riscs* aplicant diferents tècniques existents de síntesis de circuits asíncrons a partir d'STG.

Els STG dels controladors locals no poden generar-se si no s'han realitzat les fases de planificació d'operacions i assignació de recursos. Un cop s'han planificat les operacions i assignat els recursos sabem quina és la seqüència d'operacions a realitzar en cada mòdul de camí de dades i quines dependències de dades existiran entre els diferents mòduls del camí de dades. Amb aquesta informació podrem generar els STG dels controladors locals de tot el circuit.

4.3 Síntesi a partir d'STG

En aquest apartat es presenten aquelles metodologies de disseny basades en els grafs de transicions de senyals o STG. Els STG són una subclasse de xarxes de Petri [Pet62] interpretades que varen ser proposades simultàniament per Chu [Chu87] i per Rosenblum *et al.* [RY85].

Els STG permeten descriure el comportament dels circuits asíncrons mitjançant la descripció de la seqüència de transicions dels seus senyals. La figura 4.2.a mostra un exemple d'STG on x i y són senyals del circuit i on un signe $^+$ darrere del senyal indica una transició de 0 a 1 del senyal i un signe $^-$ darrere del senyal indica una transició de 1 a 0 del senyal. Anomenarem *marca* al cercle negre que etiqueta algun dels arcs.

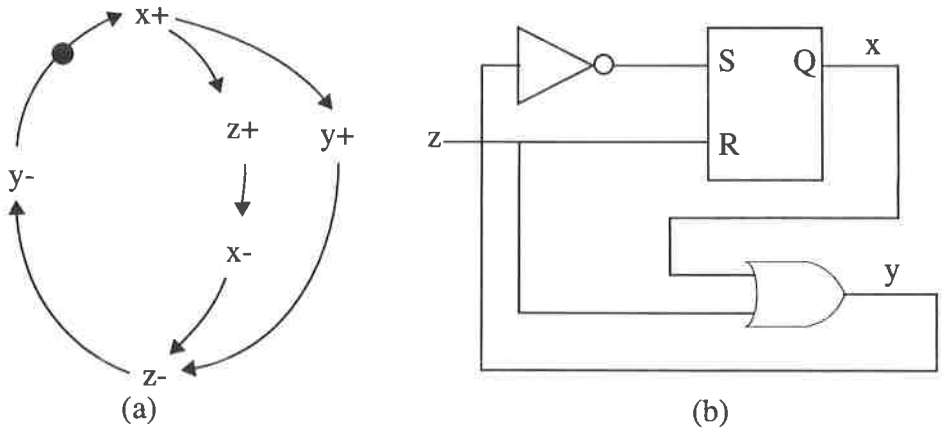


Figura 4.2: (a) Exemple d'STG; (b) Circuit amb comportament equivalent al descrit per l'STG;

De l'STG podem construir un graf d'estats. Els estats s'obtenen de disparar les transicions dels senyals. Una transició es pot disparar quan té una marca a tots els arcs entrants. Un cop s'ha disparat, s'eliminen les marques dels arcs entrants i es propaguen a tots els arcs sortints de la transició, de manera que les transicions successores puguin disparar-se. La figura 4.2.b mostra el graf d'estats per l'STG anterior. A cada estat li podem assignar un codi binari format pel valor dels senyals en l'estat.

Per tal que un STG sigui realitzable en un circuit lliure de riscos cal que el graf d'estats corresponent tingui la propietat de codificació única dels estats (USC). Un STG té aquesta propietat quan qualsevol parell d'estats del graf té codi binari diferent [MSB91].

Una altra propietat menys restrictiva és la codificació completa dels estats (CSC). Un graf d'estats té aquesta propietat si [MSB91]:

- Té la propietat d'USC.

- O bé, si dos estats tenen un mateix codi binari, les transicions de senyals de sortida permeses en els dos estats són idèntiques

A continuació es defineixen alguns termes que seran necessaris a la resta de la secció. Després es descriuen alguns dels mètodes basats en STG.

4.3.1 Definicions

La majoria de les següents definicions han estat extretes de [Moo92] i de [Lav92].

Definició 4.1 Xarxa de Petri

Una xarxa de Petri *és un quartet* $\mathcal{P} = (T, P, F, M)$ *on* T *és un conjunt de transicions no buit,* P *és un conjunt de llocs no buit,* F *és la relació de flux* $F \subseteq (T \times P) \cup (P \times T)$ *entre les transicions i els llocs i* M *és el marcatge inicial.*

Definició 4.2 Marcatge d'una xarxa de Petri

Un marcatge *és una funció* $m : P \rightarrow \{0, 1, 2, \ldots\}$, *on* $m(p), p \in P$ *és el nombre de marques que hi ha a* p *en el marcatge* m.

Definició 4.3 Graf marcat

Una xarxa de Petri és un graf marcat *si cada lloc té un únic predecessor i un únic successor.*

Definició 4.4 Màquina d'estats

Una xarxa de Petri és una màquina d'estats *si cada transició té unicament un predecessor i un successor.*

Definició 4.5 Xarxa de Petri de lliure elecció

Una xarxa de Petri és de lliure elecció *respecte a un subconjunt de transicions* T' *si per a cada dues transicions* t_1 *i* t_2, *tals que com a mínim una d'elles pertany a* T' *i tals que tenen un lloc predecessor comú,* t_1 *i* t_2 *tenen un únic predecessor.*

Una xarxa de Petri és de lliure elecció —xarxa FC— si és de lliure elecció respecte a tot el conjunt de transicions.

Definició 4.6 Xarxa de Petri viva

Un marcatge m *d'una xarxa de Petri és* viu *si per a cada marcatge* m' *que es pot aconseguir des de* m, *per a cada transició* t *existeix un marcatge* m'' *que es pot aconsegir des de* m' *que permet que la transició* t *es dispari.*

Una xarxa de Petri és viva *si el seu marcatge inicial és viu.*

Definició 4.7 Xarxa de Petri segura

Una xarxa de Petri és k-limitada *si existeix un enter k tal que per a cada lloc p i per a cada marcatge que es pot aconseguir,* $m(p) \leq p$.

Una xarxa de Petri és segura *si és 1-limitada.*

4.3.2 Revisió de les propostes basades en STG

Són molts els mètodes per dissenyar circuits asíncrons que utilitzen d'una manera o altra els STG com a estructura base. Fem un repàs d'algunes d'aquestes tècniques.

A [Lav92], Lavagno proposa descompondre el procés de síntesi en diverses fases:

- Codificació d'estats: es mira si l'STG té la propietat de CSC. Per fer-ho, es construeix el graf d'estats i s'interpreta com un màquina d'estats finita (FSM). S'apliquen tècniques clàssiques de minimització i codificació d'estats. D'aquesta FSM codificada se n'extreu informació suficient per saber quants senyals d'estat calen i on s'han de insertar les seves transicions. Aquesta tècnica té complexitat exponencial, ja que la mida del graf d'estats pot ser exponencial.

- Síntesi inicial: de l'STG es deriva una implementació a dos nivells.

- Anàlisi de riscos: es comprova que l'STG i la implementació inicial no tinguin riscos. Per fer-ho, s'assignen retards a les connexions. Existeix un risc si en assignar un retard determinat a una connexió alguna seqüència de transicions especificada a l'STG no es compleix.

- Síntesi lògica restringida: s'apliquen una sèrie de transformacions al circuit inicial per obtenir una implementació multinivell.

- Eliminació de riscos: un risc pot ser eliminat utilitzant síntesi lògica per balancejar els retards o augmentant algun retard determinat.

A [VGCM92], Vanbekbergen resol el problema de mirar si un *graf marcat* té la propietat d'USC. Un STG té la propietat d'USC si tots els senyals de l'STG formen part d'una *classe tancada.* La idea intuïtiva és la següent: si existeix un subconjunt de senyals de l'STG

tals que tant la seva transició positiva com negativa poden ser disparades sense haver de disparar cap altra transició de senyals que no estiguin en el subconjunt, tindrem dos estats amb el mateix codi.

Imaginem el subconjunt $\{a, b\}$ i el conjunt de transicions $\{a^+, b^+, a^-, b^-\}$. Si totes les transicions del conjunt es poden disparar sense haver de disparar-ne cap altra de fora el conjunt, vol dir que l'estat d'abans de disparar el conjunt de transicions i l'estat de després de disparar-les tenen el mateix codi i per tant l'STG no té la propietat d'USC. L'algorisme que comprova que tots els senvals de l'STG pertanyin a una classe tancada té cost polinòmic ($O(n^4)$), però només és aplicable a grafs marcats, on no es considera cap possible comportament condicional.

A [VLGM92], Vanbekbergen i altres proposen un altre mètode per obtenir un graf d'estats amb la propietat de CSC partint d'un STG inicial. No es demana cap característica especial a l'STG —no cal que sigui *viu*, ni *segur*—. Simplement cal que sigui un graf connex, finit i que tingui una assignació d'estats consistent.

Es construeix un graf d'estats inicial Φ derivat de l'STG amb n senyals, que són els senyals de l'STG. D'aquest graf d'estats es construeix un nou graf d'estats Φ' que té la propietat de CSC. Aquest nou graf d'estats té q senyals on $q = n+m$, on m és el nombre de senyals que cal afegir per tal que el graf sigui CSC. Aquest nombre m de senyals se suposa prefixat. De cada estat de Φ es deriva un o més estats del graf Φ' aplicant un procediment de traducció que assegura que el nou graf es CSC. Aquest mètode té complexitat exponencial.

4.3.3 Algorismes polinòmics

A diferència de les tècniques descrites en els apartats anteriors, les que es proposen a [PC93a] i a [PC93b] no requereixen generar el graf d'estats per detectar conflictes de codificació. Es tracten xarxes FC.

A [PC93a] es resolen els possibles conflictes que hi hagi d'USC mitjançant l'anàlisi d'un subconjunt de màquines d'estat que cobreixin l'STG. La figura 4.3 mostra un exemple de com es pot descompondre un STG en màquines d'estat.

Definició 4.8 *Per a cada lloc p_i de una xarxa FC viva i segura, definim $\mathcal{V}_i$ com el conjunt de vectors binaris dels marcatges que tenen una marca a p_i.*

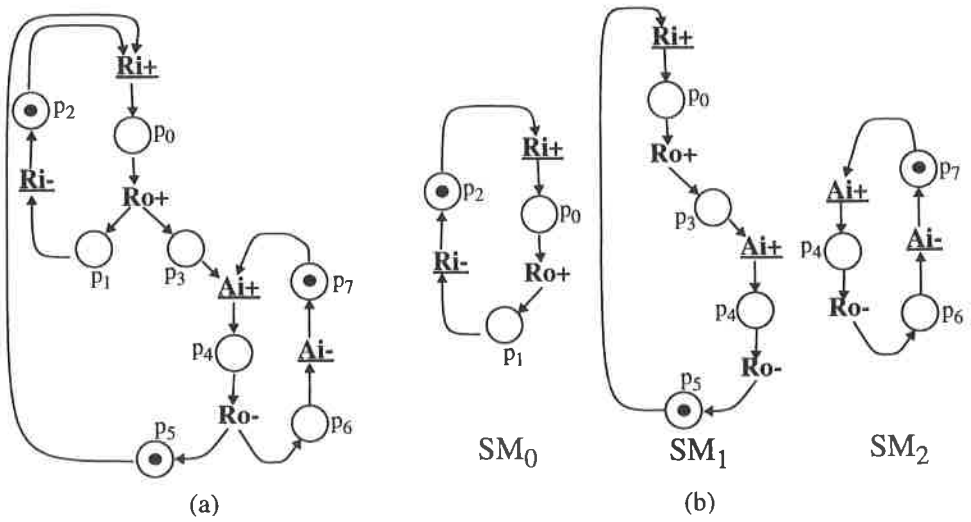


Figura 4.3: (a) STG; (b) Màquines d'estat per a l'STG anterior;

Per detectar els conflictes d'USC d'un STG cal trobar un conjunt de màquines d'estat que cobreixi de manera irredundant l'STG, tal que si es fa la intersecció de qualsevol dels conjunts $\mathcal{V}_i$ de cada màquina d'estat entre si , la intersecció sigui buida.

Enlloc de calcular els conjunts $\mathcal{V}_i$ que requeriria un algorisme amb complexitat exponencial, es busquen els cubs $\mathcal{C}_i$ tals que cobreixen tots els vectors binaris de $\mathcal{V}_i$. En utilitzar els cubs enlloc dels conjunts per detectar si l'STG té la propietat d'USC, estem fent més restrictiva la propietat d'USC, de manera que pot passar que es detectin conflictes que realment no existeixen. En contrapartida, el fet d'utilitzar cubs ens porta a un algorisme amb complexitat polinòmica.

Un cop s'han detectat els conflictes, s'inserten variables per eliminar-los obtenint un STG amb la propietat d'USC i, per tant, el circuit que sintetitzem serà lliure de riscos.

A [PC93b] es proposa un algorisme polinòmic per resoldre el problema de CSC basat en el mètode anterior i per sintetitzar circuits lliures de riscos.

4.4 De SDFG a STG

El primer pas que cal considerar és com podem passar d'un graf de flux de dades planificat (vegeu capítol 2) en el que les operacions estan planificades i assignades a un recurs a un conjunt de processos comunicants on cada procés és executat en un processador.

Després caldrà traduir aquest graf d'estats a un STG. Del SDFG cal extreure'n dos tipus d'informació:

- Quines operacions s'executen a cada mòdul del camí de dades (i en quin ordre).
- Quines transferències de dades hi haurà entre els diferents mòduls (i en quin ordre).

De moment suposarem que en cada processador hi ha un únic component del camí de dades.

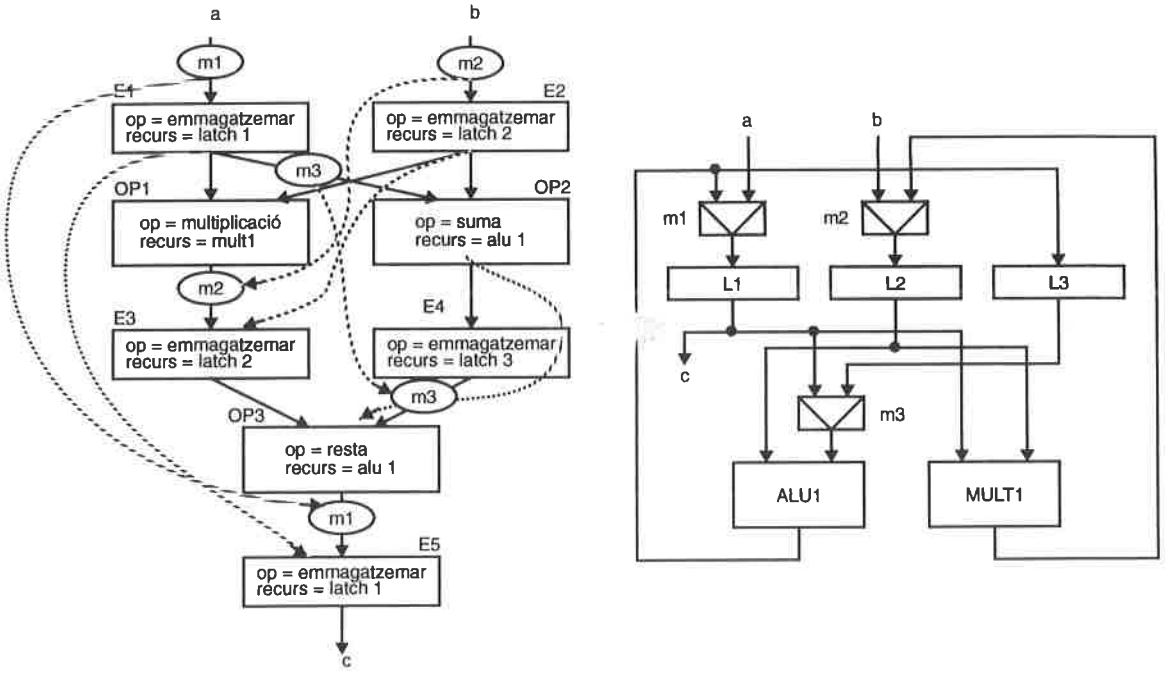
La figura 4.4.a mostra un exemple d'un SDFG. Cada vèrtex del graf conté informació sobre quina operació executa i en quin recurs. Els arcs entre vèrtexs indiquen dependències entre ells.

Dependències de dades Els arcs amb línia contínua representen dependències de dades, és a dir, entre els dos controladors locals dels recursos que han estat assignats a aquests vèrtexs hi haurà com a mínim una sincronització per poder realitzar una transferència d'informació (la indicada per l'arc). Quan siguin necessaris multiplexors o busos per efectuar la transferència d'informació, l'arc corresponent contindrà informació sobre quin recurs s'utilitzarà. En aquest cas s'ha optat per una camí de dades amb arquitectura orientada a multiplexors tal com mostra la figura 4.4.b.

Dependències estructurals Els arcs amb línia discontínua representen dependències estructurals. Les dependències estructurals existeixen entre vèrtexs que han estat assignats a un mateix recurs. Com que un recurs no pot executar dues operacions simultàniament cal establir un ordre d'execució entre aquests vèrtexs. La planificació d'operacions determina les dependències estructurals. Les dependències estructurals ens permetran establir l'ordre en què s'executaran les operacions representades pels vèrtexs als diferents processadors.

La figura 4.4.c mostra com podem descompondre el SDFG en processos. Cada procés es compon d'operacions i primitives de pas de missatges *enviar* i *rebre*.

Hem triat un exemple senzill on la planificació d'operacions i l'assignació de recursos són trivials, però suficient pels nostres propòsits. Podem veure que cada vèrtex del SDFG és descompost en una o diverses primitives de comunicació i en una operació en el recurs.



Procés ALU1	Procés MULT1	Procés L1	Procés L2	Procés L3	Procés m1	Procés m2	Procés m3
		E1			rebre a	rebre b	
		rebre m1	rebre m2		selecciona 1	selecciona 0	
		emmagatzema	emmagatzema		envia L1	envia L2	
		envia m3					rebre L1
rebre L2,m3	rebre L1,L2	envia MULT1	envia ALU1,MULT1				selecciona 0
suma	multiplica						envia ALU1
envia L3	envia m2			rebre ALU1		rebre MULT1	
			rebre m2	emmagatzema		selecciona 1	
			emmagatzema			envia L2	
OP3				envia m3			rebre L3
rebre L2, m3			envia ALU1				selecciona 1
resta							envia ALU1
envia m1					rebre ALU1		
		rebre m1			selecciona 0		
		emmagatzema			envia L1		
		envia c					

(c)

Figura 4.4: (a) Graf de flux de dades planificat; (b) Camí de dades corresponent a l'SDFG anterior; (c) Descomposició del graf anterior en operacions i primitives de comunicació

Per exemple, el vèrtex E1 es descompon en una primitiva de rebre del multiplexor 3 i en una operació d'emmagatzemament. Un altre exemple és el vèrtex OP3 que es descompon en dues primitives de rebre dels multiplexors 1 i 2, en una operació de resta a l'ALU 1 i en una primitiva d'enviar al multiplexor 3.

4.4.1 Descomposició dels processos en senyals de control

Per a cada primitiva de comunicació el controlador local del mòdul haurà de generar senyals de control globals per sincronitzar-se amb altres controladors. Per a cada operació sobre el recurs el controlador haurà d'intercanviar senyals locals amb el mòdul (bàsicament petició i acabament).

Es generen dos tipus de senyals de control global segons si el controlador s'ha de sincronitzar per rebre dades d'entrada o per enviar dades de sortida.

Dades d'entrada: anomenarem *entrada vàlida (ev)* al senyal global d'entrada a un CL que quan s'activa indica al controlador que s'ha rebut una determinada dada d'entrada. Anomenarem *entrada consumida (ec)* al senyal global de sortida que activa un CL per indicar que una determinada dada d'entrada ja ha estat utilitzada i que no es necessita més. Aquest dos senyals s'intercanvien entre CL amb un protocol de quatre fases. Per a cada dada d'entrada tindrem un parell de senyals (*entrada vàlida*, *entrada consumida*).

Dades de sortida: anomenarem *sortida vàlida (sv)* al senyal global de sortida que activa un CL per indicar que una determinada dada de sortida ja és estable. Anomenarem *sortida consumida (sc)* al senyal global d'entrada que quan és activat indica al CL que el rep que no cal mantenir per més temps aquesta sortida estable perquè ja ha estat utilitzada. Igual que amb els senyals anteriors, aquests dos senyals s'intercanvien amb un protocol de quatre fases. Per a cada dada de sortida tindrem un parell de senyals (*sortida vàlida*, *sortida consumida*).

Són aquests senyals, juntament amb els de petició i acabament i la seva composició amb protocols de quatres fases, que ens permetran descriure el comportament dels controladors amb STG. Així, per exemple, una primitiva *rebre* es traduirà en un intercanvi de senyals

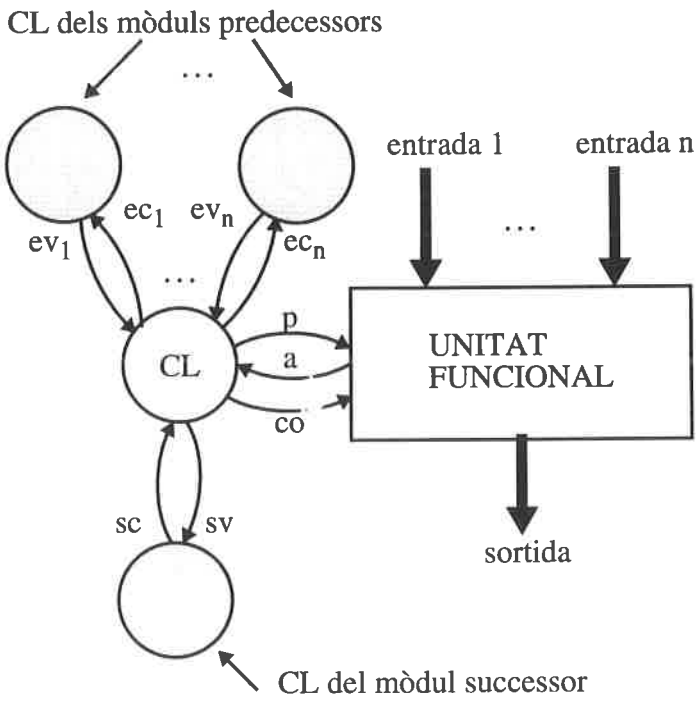


Figura 4.5: Processador format per una unitat funcional i el seu corresponent controlador local

(*entrada vàlida*, *entrada consumida*) i una primitiva *enviar* en un intercanvi de senyals (*sortida vàlida*, *sortida consumida*). A continuació veurem un exemple d'STG per quatre tipus de recursos diferents: unitat funcional, *latch*, multiplexor d'entrada a una unitat funcional i multiplexor d'entrada a un *latch*.

4.4.2 STG per a una unitat funcional

Vegem com seria l'STG per a una operació en una unitat funcional que pot efectuar diferents operacions amb n entrades i una sortida (vegeu figura 4.5). Al llarg d'aquest apartat i dels següents, parlarem de mòduls *predecessors* per indicar els mòduls que generen les entrades d'un mòdul determinat i de mòduls *successors* per indicar els mòduls que utilitzen el resultat d'un mòdul determinat.

En els processos, podem identificar una operació a executar en una unitat funcional com el conjunt de primitives de comunicació i d'operacions de càlcul:

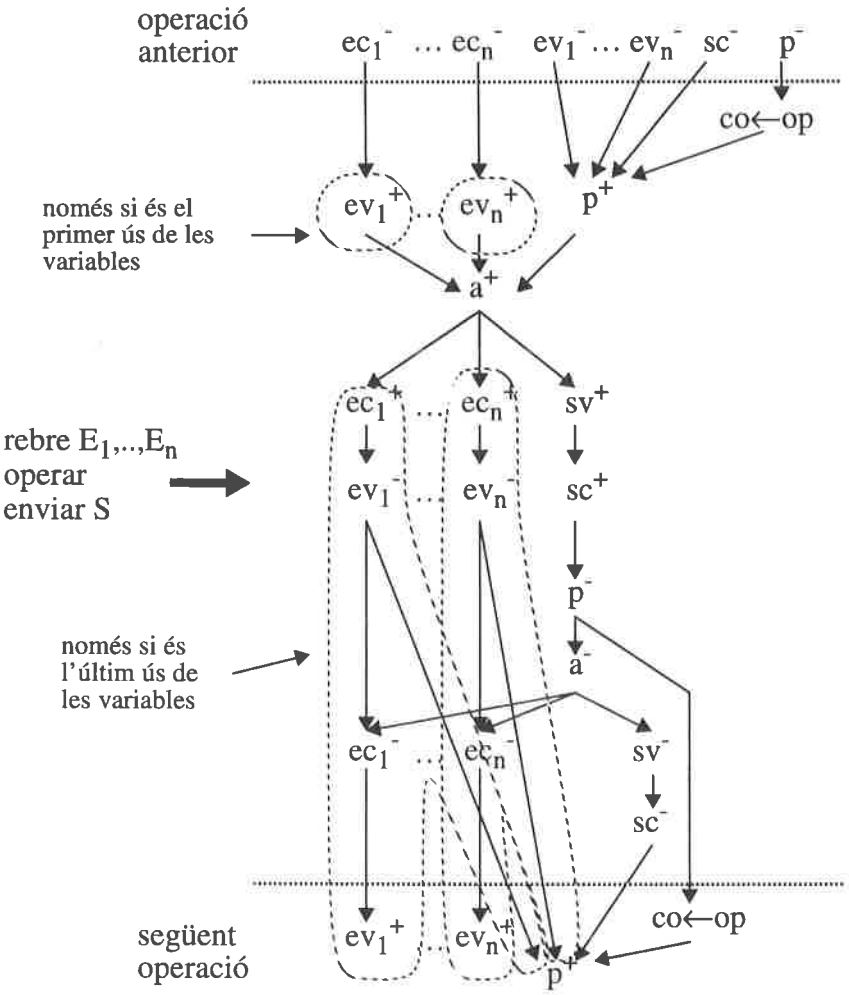


Figura 4.6: STG per a una unitat funcional

rebre $E_1, \ldots, E_n$

operar

enviar S

Les primitives rebre es traduiran en l'STG per protocols de quatre fases de les variables (ev, ec) i les d'enviar per protocols de les variable (sv, sc) tal com ja hem dit.

La figura 4.6 mostra la combinació d'aquests protocols i la dels senyals de petició i d'acabament per formar l'STG complet.

Aquest STG concorda amb el comportament de les portes que s'ha descrit en aquest

capítol:

- Partint d'un estat en què les entrades i sortides estan inicialitzades, pot passar o bé que les entrades passin a ser vàlides (estat 4) o que activem el senyal de petició p (estat 3). Si alguna de les variables ja s'havia utilitzat a la unitat per algun altre càlcul, no haurem d'esperar que s'activi el senyal de ev_i ja que ja s'havia activat previament. En unitats multifunció, abans d'activar el senyal p, cal que s'hagi seleccionat l'operació que es vol executar donant el valor adequat al codi d'operació co. El codi d'operació serà un vector de bits $co = (co_1, \ldots, co_l)$ de manera que la unitat pot realitzar com a molt 2^l tipus d'operacions diferents. Només s'han d'activar o desactivar aquells bits del codi d'operació que tinguin un valor diferent al que ens interessa. Això ho indiquem amb la transició $co \leftarrow op$.

- Quan les entrades són vàlides i el senyal p s'ha activat, la porta calcula la funció. El control s'espera fins que s'activa el senyal d'acabament a que indica que el càlcul ha acabat.

- A continuació poden activar-se diferents senyals simultàniament. D'una banda, com que el resultat ja es vàlid, podem comunicar-ho a d'altres CL que necessiten aquest valor activant el senyal de sortida vàlida (sv^+). A més, com que les portes tenen la capacitat de memorització, podem deixar que les entrades siguin inicialitzades. Comuniquem aquest fet a d'altres CL activant els senyals d'entrada consumida (ec^+). Un cas en què no s'activarà ec_i és quan la variable i sigui utilitzada en més d'una operació. En aquest cas, només s'activarà ec_i quan es tracti de l'últim ús de la variable.

- Abans d'inicialitzar la porta, cal esperar que s'activi el senyal sc que indica que el mòdul successor ja no necessita més el resultat. En cas que el resultat l'utilitzessin més d'un mòdul successor, hi hauria tants parells de senyals (sv, sc) com mòduls successors, i caldria esperar que s'activessin tots els senyals sc abans d'inicialitzar la porta. Per inicialitzar la porta cal desactivar el senyal p.

- Quan les entrades ja estan inicialitzades ($ev_i^-, \forall i$) i la porta ja ha estat inicialitzada (c^-), podem desactivar els senyals d'entrada consumida de manera que els mòduls

predecessors poden passar a fer un altre càlcul. Els arcs que van de la transició c^- al les transicions ec_i^- eviten que els mòduls predecessors passin a fer cap càlcul si no s'ha inicialitzat la porta actual. Recordem que si es generessin noves entrades vàlides abans que la porta s'hagués inicialitzat, podríem passar a l'estat 7 (vegeu figura 3.8).

- D'altra banda, un cop s'ha acabat la inicialització de la porta, també es desactiva el senyal de *sv* per indicar que les sortides estan inicialitzades. Abans de poder executar una altra operació en aquest mòdul, cal esperar que el senyal de *sc* es desactivi (en cas que hi hagi més d'un mòdul successor, cal esperar que es desactivin tots els sc_i). Igual que abans, si es generessin unes noves sortides abans que el mòdul successor s'hagués inicialitzat, podria portar el mòdul successor a l'estat 7.

Aquesta seqüència de transicions es repeteix per a cada operació que es realitza en la unitat funcional, de manera que l'STG complet per a cada unitat funcional consisteix en una seqüència d'STG com el de la figura.

4.4.3 STG per a un multiplexor d'entrada a una unitat funcional

A continuació, presentarem un STG per a un multiplexor d'entrada a una unitat funcional (vegeu figura 4.7). Tot i que els multiplexors poden tenir n entrades, en cada operació només en cal una: la corresponent a l'entrada que es vol seleccionar. En un procés, podem identificar un transferència de dades en un multiplexor com la seqüència de primitives de comunicació i operacions:

rebre E_i
seleccionar i
enviar S

Si una variable s'utilitza en més d'una operació en el mòdul successor, només rebrem la dada una vegada, però l'enviarem al mòdul successor tantes vegades com es necessiti. Per exemple, imaginem que una mateixa variable seleccionada per l'entrada i del multiplexor s'utilitza en dos càlculs diferents —no necessàriament consecutius— en la unitat successora

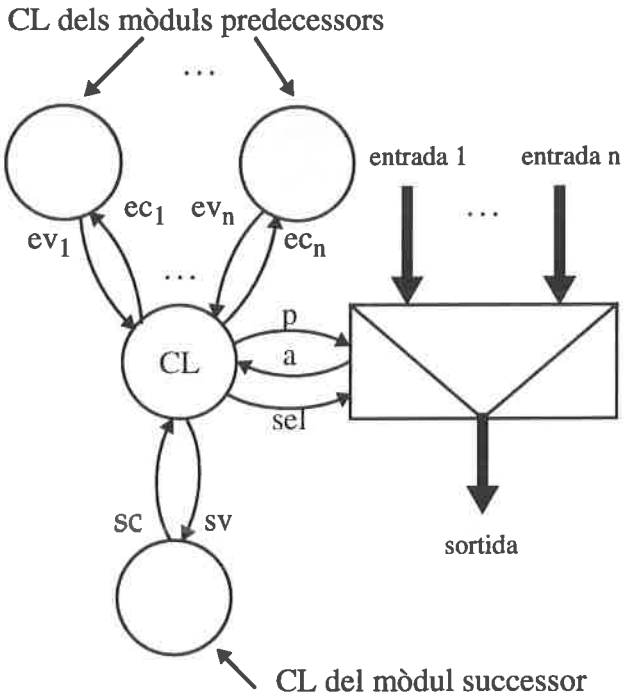


Figura 4.7: Processador format per un multiplexor i el seu controlador local

(essent igual el valor de la variable d'entrada). Tindríem una seqüència de primitives de comunicació i operacions semblants a:

rebre E_i
seleccionar i
enviar S
...
seleccionar i
enviar S

En l'STG de la figura 4.8, que representa un STG per a una transferència de dades amb un multiplexor d'entrada a una unitat funcional, s'hi veu reflectit aquest fet que acabem d'esmentar i d'altres que comentarem a continuació:

- Partint d'un estat en què les entrades i les sortides estan inicialitzades, podem passar a diferents estats segons si el que s'activa primer és el senyal ev_i o el senyal p. De manera semblant al que passava abans amb el codi d'operació, abans d'activar el senyal p cal que els senyals de selecció siguin estables. En un multiplexor de n entrades tindrem un senyal de selecció format per un vector de $\log_2 n$ bits. Només s'han de generar transicions en aquells bits en què el valor actual sigui diferent del que ens interessa per seleccionar l'entrada i. Això s'indica amb la transició $sel \leftarrow i$. En cas que no sigui el primer ús de la variable d'entrada, no caldrà esperar que s'activi el senyal ev_i.

- Quan la entrada és vàlida i s'ha activat el senyal p, el multiplexor selecciona l'entrada corresponent fent que s'obtingui a la sortida. Quan el multiplexor activa el senyal a vol dir que la sortida ja és estable.

- A continuació poden activar-se dos senyals globals simultàniament: el senyal ec per indicar que no es necessita més el valor d'entrada (només si es tracta de l'últim ús de la variable en qüestió) i el senyal sv per indicar que la sortida ja és estable.

- En cas que s'hagi activat el senyal ec_i, el mòdul predecessor serà inicialitzat. Aquest fet és comunicat pel CL del mòdul predecessor al CL del multiplexor mitjançant la

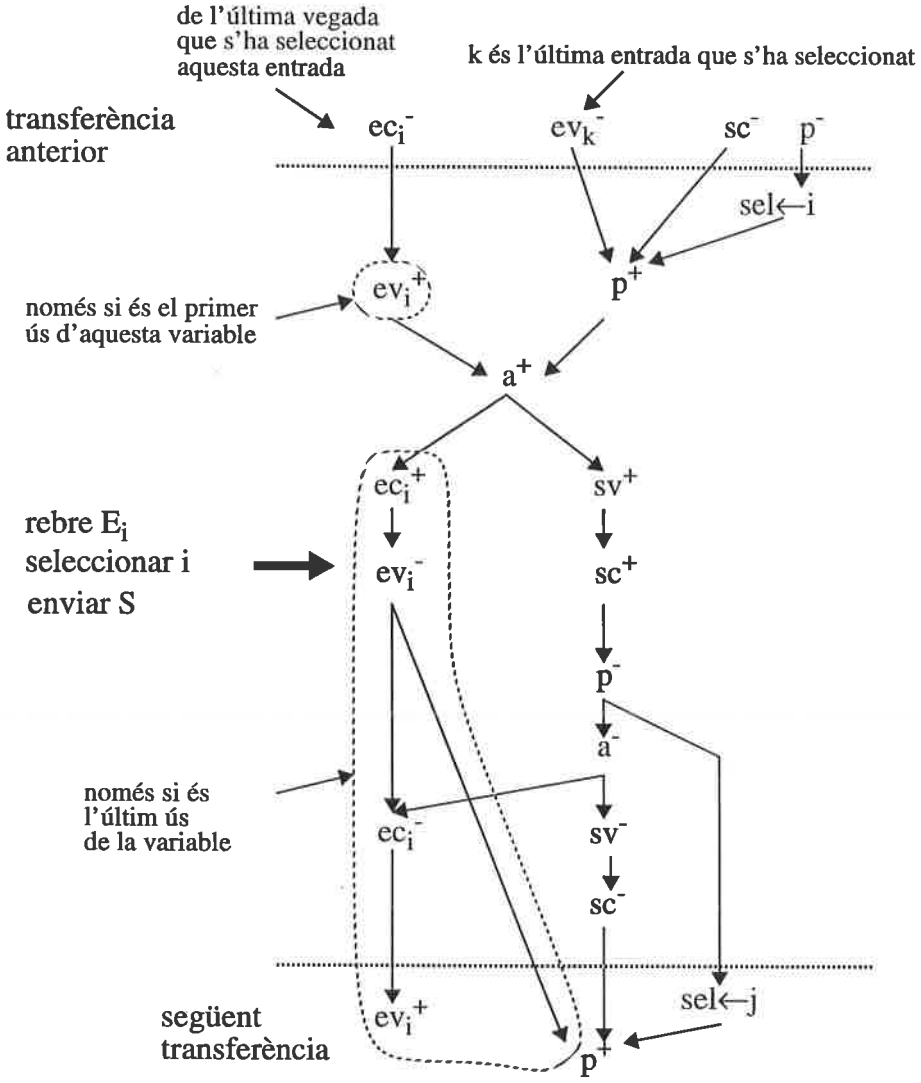


Figura 4.8: STG per a un multiplexor d'entrada a una unitat funcional

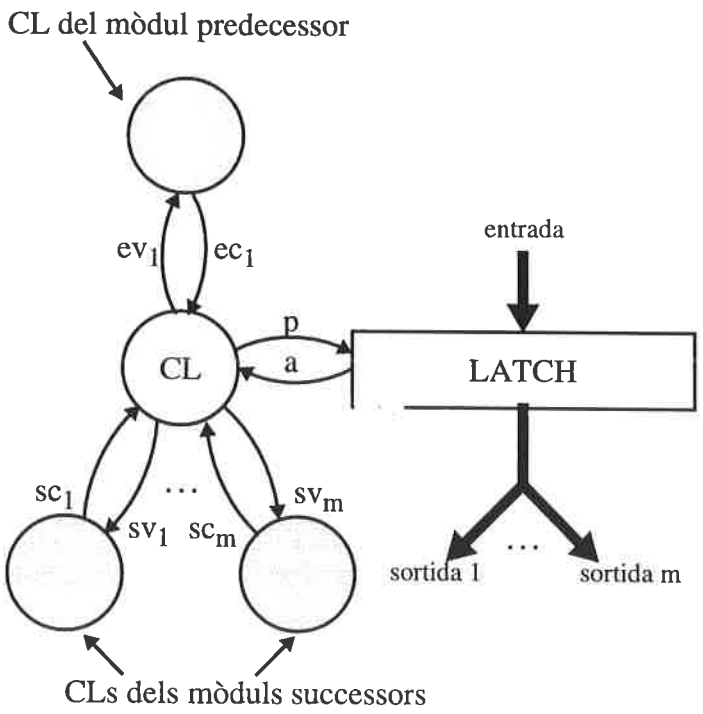


Figura 4.9: Processador format per a un latch *i el seu controlador local*

desactivació del senyal ev_i. Després que aquesta transició s'hagi realitzat, cal esperar que el multiplexor hagi estat inicialitzat per desactivar el senyal ec_i per les mateixes raons que s'han esmentat a l'apartat anterior.

- D'altra banda, quan el mòdul successor activa el senyal sc podem inicialitzar el multiplexor (p^-). Quan el multiplexor està inicialitzat es desactiva el senyal a i es poden acabar els protocols que estaven pendents (ec_i^- i sv^-, sc^-).

4.4.4 STG per a un latch

En aquest apartat presentarem la part de l'STG corresponent al CL d'un *latch* on es controla l'emmagatzemament d'una variable (vegeu figura 4.9). Suposarem que el *latch* rep la variable a emmagatzemar d'un multiplexor o una unitat funcional i que l'envia a un o més mòduls successors. Considerarem que el *latch* fa un únic ús de la variable, ja que tot i que seria possible emmagatzemar dues vegades la mateixa informació en el mateix

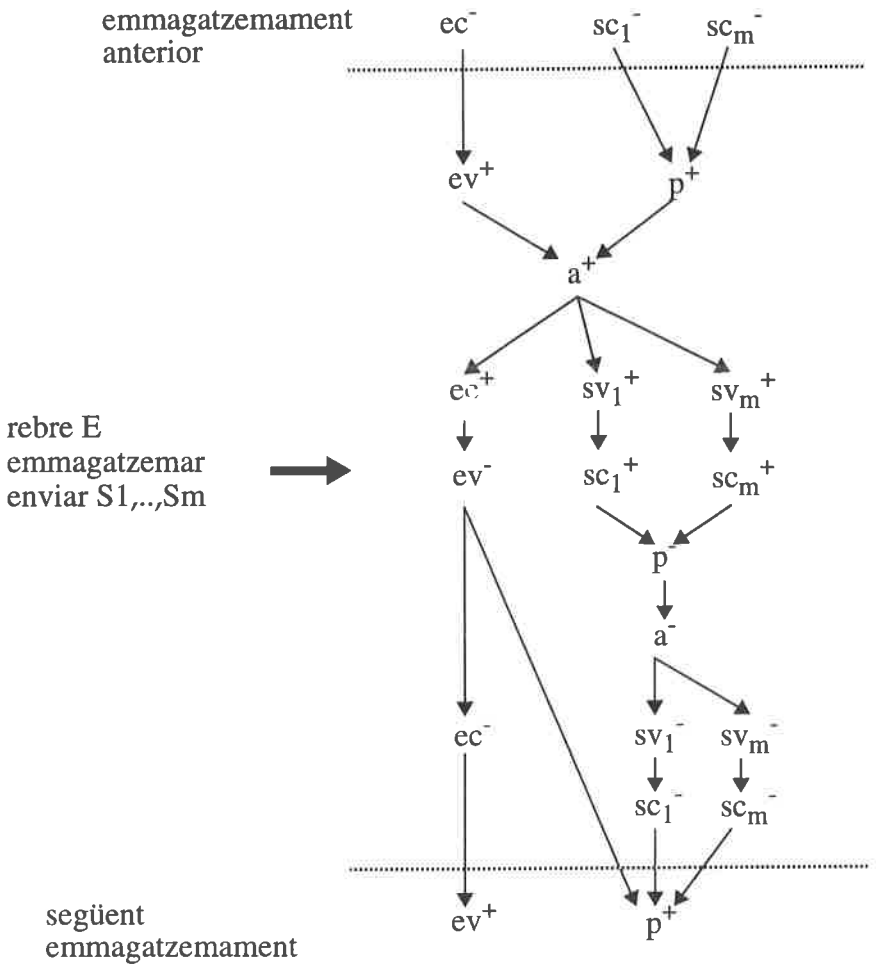


Figura 4.10: STG per a un latch

latch no sembla lògic. Així doncs, la seqüència de primitives de comunicació i operacions corresponents a l'emmagatzemament d'una variable en un *latch* tenen el següent aspecte:

rebre E

emmagatzemar

enviar $S_1, \ldots, S_m$

La figura 4.10 correspon a la traducció a STG del tros de procés anterior.

La seqüència de transicions corresponent a aquest STG és la següent:

- El controlador, a la vegada que espera que les entrades siguin vàlides (ev^+), activa el senyal de *petició* (p^+).

- El controlador espera que el *latch* indiqui que la informació ja està emmagatzemada. Aquest esdeveniment s'indica amb l'activació del senyal *acabament* (a^+).

- A continuació poden activar-se diferents senyals simultàniament. Es pot fer la transició ec^+ per indicar que la variable ja està emmagatzemada. El mòdul predecessor pot inicialitzar-se i després pot passar a efectuar algun altre càlcul, ja que això no afectarà al *latch*. També es poden efectuar les transicions sobre els senyals de *sortida vàlida* ($sv_1^+, \ldots, sv_m^+$) per indicar als mòduls successors que la sortida és estable.

- Les transicions sobre els senyals de *sortida consumida* $sc_1^+, \ldots, sc_m^+$ indiquen que els mòduls successors ja han utilitzat la variable i per tant podem inicialitzar el *latch* (p^-).

- La transició a^- indica al CL que el *latch* s'ha inicialitzat. A continuació, el CL desactiva els senyals de *sortida vàlida* ($sv_1^-, \ldots, sv_m^-$). Les transicions $sc_1^-, \ldots, sc_m^-$ indiquen que els mòduls successors han acabat d'incialitzar-se i per tant ja podem emmagatzemar un altra variable al *latch*. Cal esperar aquestes transicions abans que el senyal de *petició* s'activi de nou, ja que si la sortida tornés a ser vàlida abans que tots els mòduls successors estiguin en fase d'inicialització, algun d'ells podria passar a l'estat 7.

4.4.5 STG per a un multiplexor d'entrada a un latch

La figura 4.11 mostra una part d'un STG que descriu el comportament d'un CL d'un multiplexor d'entrada a un *latch*. Aquesta part d'STG correspon a la seqüència de primitives de comunicació i operacions:

rebre E_i
seleccionar i
enviar S

Com es pot veure, aquest graf és gairebé igual al corresponent a un multiplexor d'entrada a una unitat funcional. La diferència principal està en el fet que en les unitats funcionals és

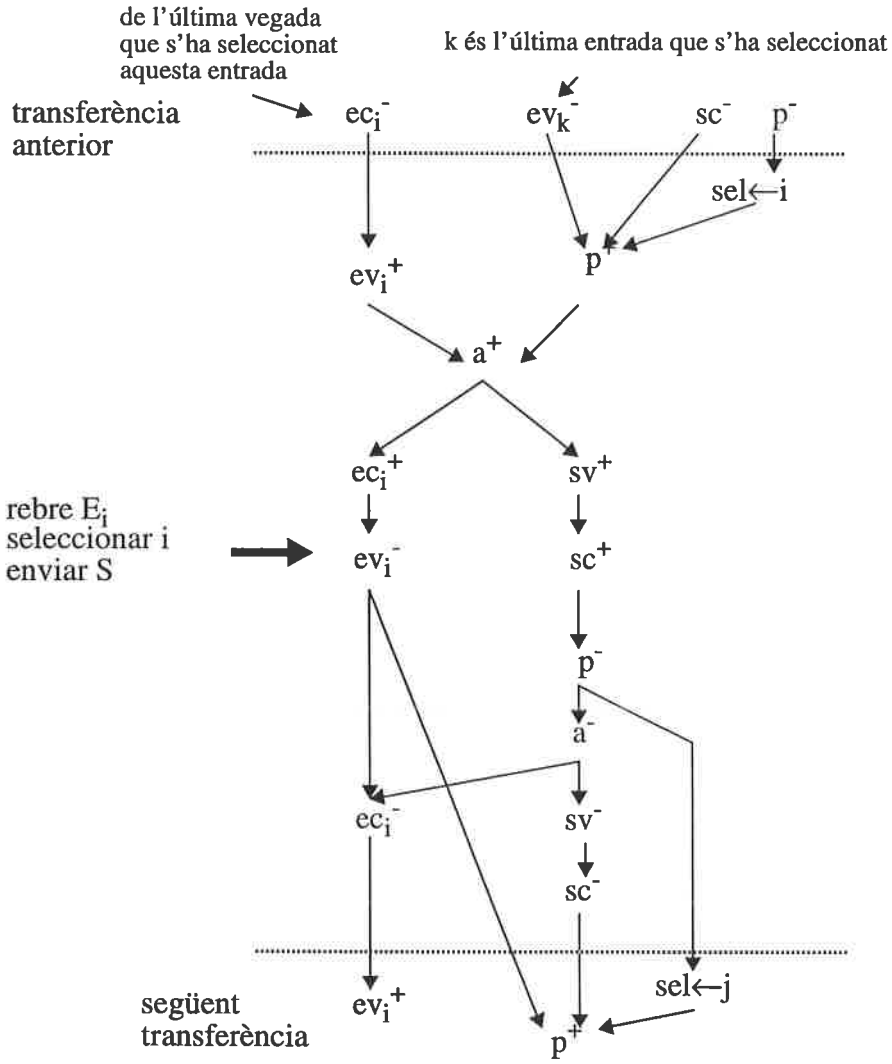


Figura 4.11: STG per a un multiplexor d'entrada a un latch

raonable que una variable s'utilitzi en més d'un càlcul. En el cas del *latchs*, ja hem comentat a l'apartat anterior que tot i que això seria possible, no seria lògic, ja que comportaria emmagatzemar dues —o més— vegades el mateix valor en el *latch*. Aquest fet afecta a la part del graf corresponent a les entrades, és a dir, al protocol dels senyals (ev_i, ec_i). En el cas del multiplexor d'entrada a una unitat funcional, si no es tractava del primer ús de la variable, no calia esperar la transició ev_i^+ i si no es tractava de l'últim ús de la variable no calia acabar el protocol dels senyals d'entrada. En el cas del multiplexor d'entrada al *latch*, com que considerem un únic ús de la variable, sempre esperarem que s'activi aquest senyal i sempre acabarem el protocol dels senyals d'entrada.

4.4.6 Exemple concret

A continuació veurem com es generen els STG dels CL per a un exemple concret. Utilitzarem l'exemple mostrat a la figura 4.4, on ja hem vist com podem passar d'un SDFG a un conjunt de processos que descriuen el comportament dels CL i un camí de dades. Com que descriure tots els STG seria llarg i tediós, ens limitarem a veure la connectivitat dels CL i l'STG per la ALU1.

La figura 4.12 mostra un esquema dels CL per a aquest exemple i quina connectivitat tenen. Els arcs entre els diferents CL indiquen intercanvis de senyals globals entre ells. Podem veure que existeixen arcs entre els CL que controlen components del camí de dades que tenen transferències de dades entre ells, per exemple, el CL del multiplexor 1 amb el CL del *latch* 1.

Aquells grups de CL que tenen una interconnectivitat més gran són canditats a ser agrupats en un únic processador, de manera que el component de càlcul d'aquest processador estaria format pels diferents components del camí de dades que formaven els processadors anteriors i el component de control seria un únic CL. La següent secció estudia com podem agrupar processadors.

A la figura 4.4.c podem veure que el procés que executa el CL de la ALU1 consisteix a fer una suma i una resta de les dades que li arriben del *latch* 2 i del multiplexor 3 i enviar el resultat a *latch* 3 i al multiplexor 1 respectivament. Si suposem que aquest procés s'executa infinitament, l'STG resultant és el que es mostra a la figura 4.13.

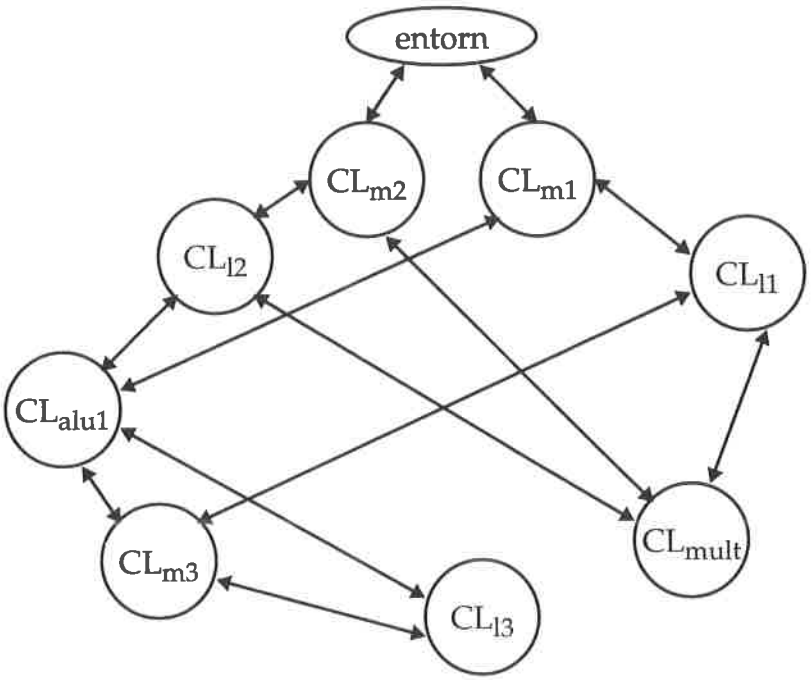


Figura 4.12: Connectivitat dels Controlador Locals per l'exemple de la figura 4.4

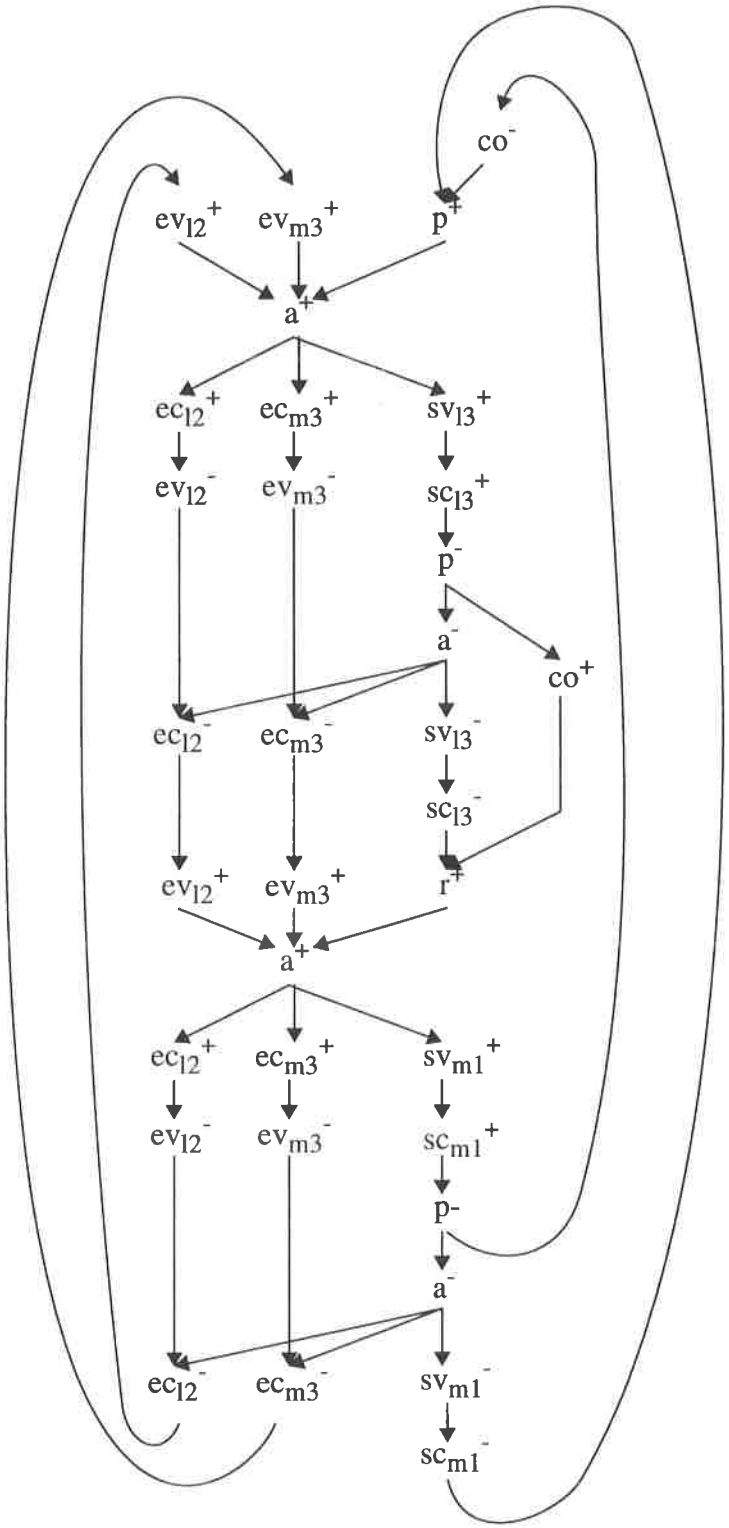


Figura 4.13: STG per a l'ALU 1 per a l'exemple de la figura 4.4

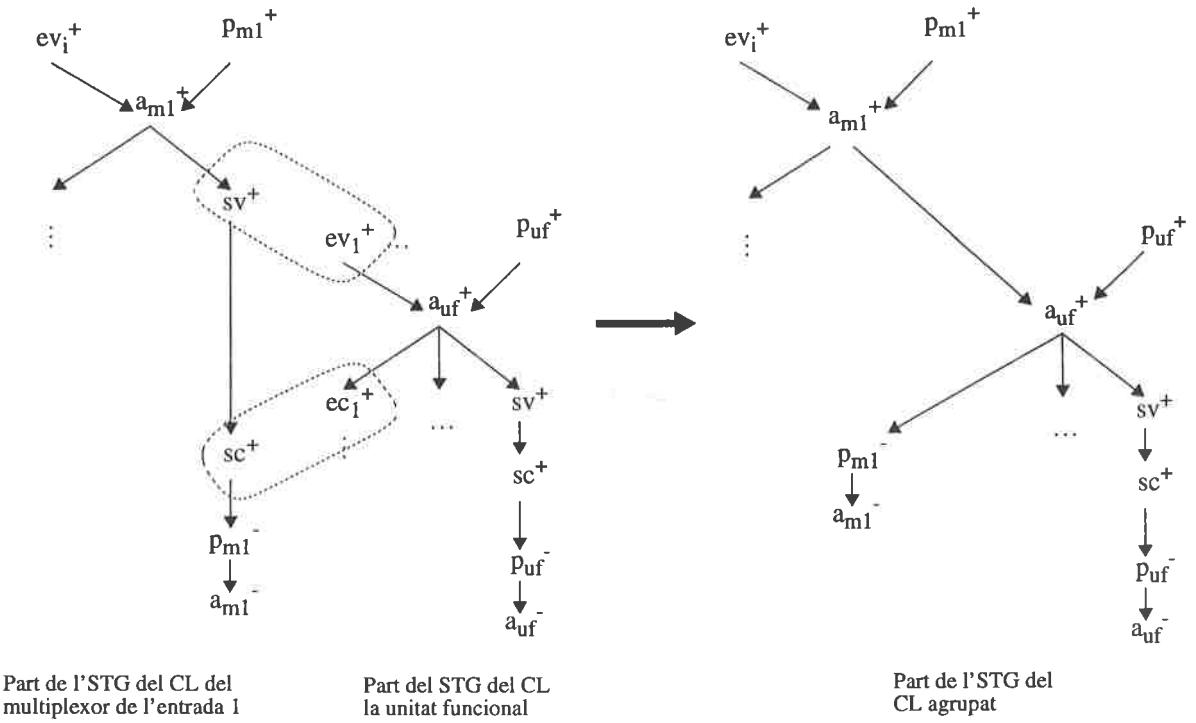


Figura 4.14: Exemple de com podem agrupar l'STG del controlador local d'un multiplexor amb el d'una unitat funcional

4.4.7 Agrupació de controladors locals

Fins ara hem suposat que per a cada component del camí de dades hi hauria un controlador local, però en alguns casos és convenient tenir un únic controlador local per dos o més components del camí de dades si aquests components estan molt relacionats i si hi ha moltes transferències de dades entre ells.

Pensem, per exemple, en una unitat funcional i els multiplexors que pugui tenir a l'entrada. Totes les transferències que fan els multiplexors van a parar a la unitat funcional, de manera que sembla interessant agrupar aquests components en un únic processador i un únic controlador local. D'aquesta manera, tot i que tindrem un controlador local lleugerament més complex que els d'abans, el rendiment d'aquests components del camí de dades es millorarà, ja que s'ha reduït la sobrecàrrega que provoquen els senyals de control en eliminar-ne alguns.

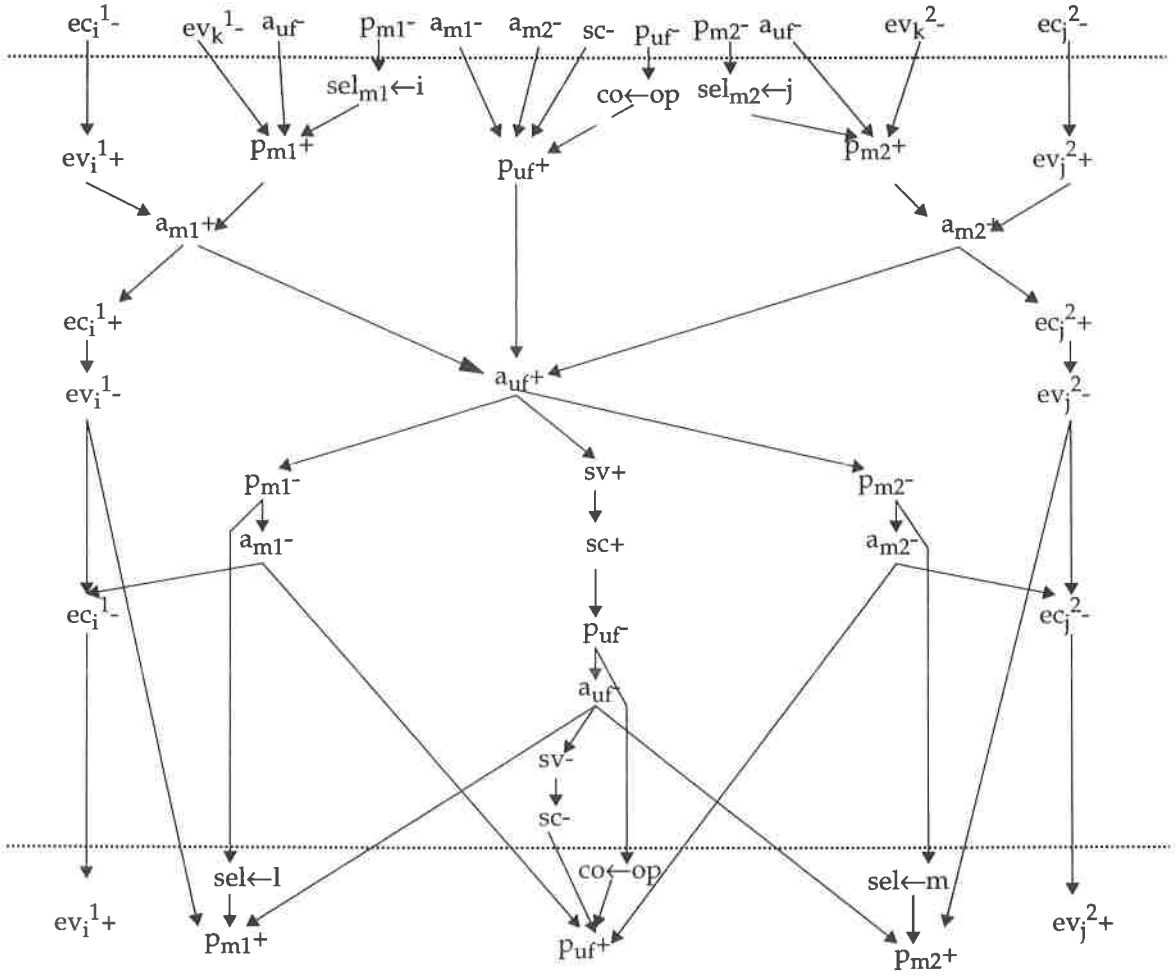


Figura 4.15: Part de l'STG per al CL d'una unitat funcional i dos multiplexors d'entrada

L'agrupació de controladors locals per a aquests casos és senzilla, ja que simplement consisteix a eliminar els parells de transicions positives o negatives dels senyals duals dels controladors i a substituir-los per un arc. Aquests senyals duals són, per exemple, el parell (sv, ev) on sv és el senyal de *sortida vàlida* d'un mòdul predecessor i ev és el senyal d'*entrada vàlida* d'un mòdul successor, o el parell (sc, ec) on sc és el senyal *sortida consumida* i ec és el senyal *entrada consumida* dels mateixos mòduls.

Per exemple, en el cas que hem esmentat abans, caldrà eliminar les transicions positives del senyal global sv del multiplexor de l'entrada 1 i les transicions positives del senyal ev del la unitat funcional i substituir-los per un arc, tal com mostra la figura 4.14. És a dir, com que ara el controlador local tindrà accés als senyals de *petició* i *acabament* dels multiplexors i de la unitat funcional, no té sentit generar i llegir senyals que servien per comunicar quin valor tenien aquells senyals.

Abans d'agrupar, el controlador local del multiplexor es comunica amb el controlador local de la unitat funcional amb el senyal sv (que el CL de la unitat funcional rep en forma del senyal ev_1) per indicar si la sortida del multiplexor és vàlida (sv^+) o inicialitzada (sv^-). El CL del multiplexor sap l'estat de la sortida del multiplexor pel senyal d'*acabament*. Quan detecta la transició a^+ activa el senyal sv i quan detecta la transició a^- desactiva el senyal sv.

D'altra banda, el CL de la unitat funcional, quan detecta la transició ev_1^+, sap que l'entrada 1 és vàlida i per tant pot prosseguir amb el càlcul si totes les altres entrades són vàlides i, quan detecta la transició ev_1^-, sap que l'entrada 1 està inicialitzada. L'altre parell de senyals que intercanvien aquests CL són (sc, ec). El controlador local de la unitat funcional activa el senyal ec_1 per indicar al controlador local del multiplexor que ja ha acabat el càlcul. El CL del multiplexor espera la transició sc^+ i, en detectar-la, passa a inicialitzar el multiplexor (p^-). El CL de la unitat funcional desactiva el senyal ec_1 quan la unitat funcional ja s'ha inicialitzat. El CL del multiplexor espera la transició sc^- que indica aquest fet i, en detectar-la, permet que es continuï amb una altra operació.

Tot aquest protocol ara no té sentit, de manera que és substituït per arcs en aquells punts on hi havia un parell de transicions dels senyals duals. La figura 4.15 mostra com quedaria l'STG del CL d'una unitat funcional i dos multiplexors d'entrada.

$p_{m1}, a_{m1}, p_{m2}, a_{m2}, p_{uf}$ i a_{uf} són els senyals de *petició* i *acabament* dels multiplexors

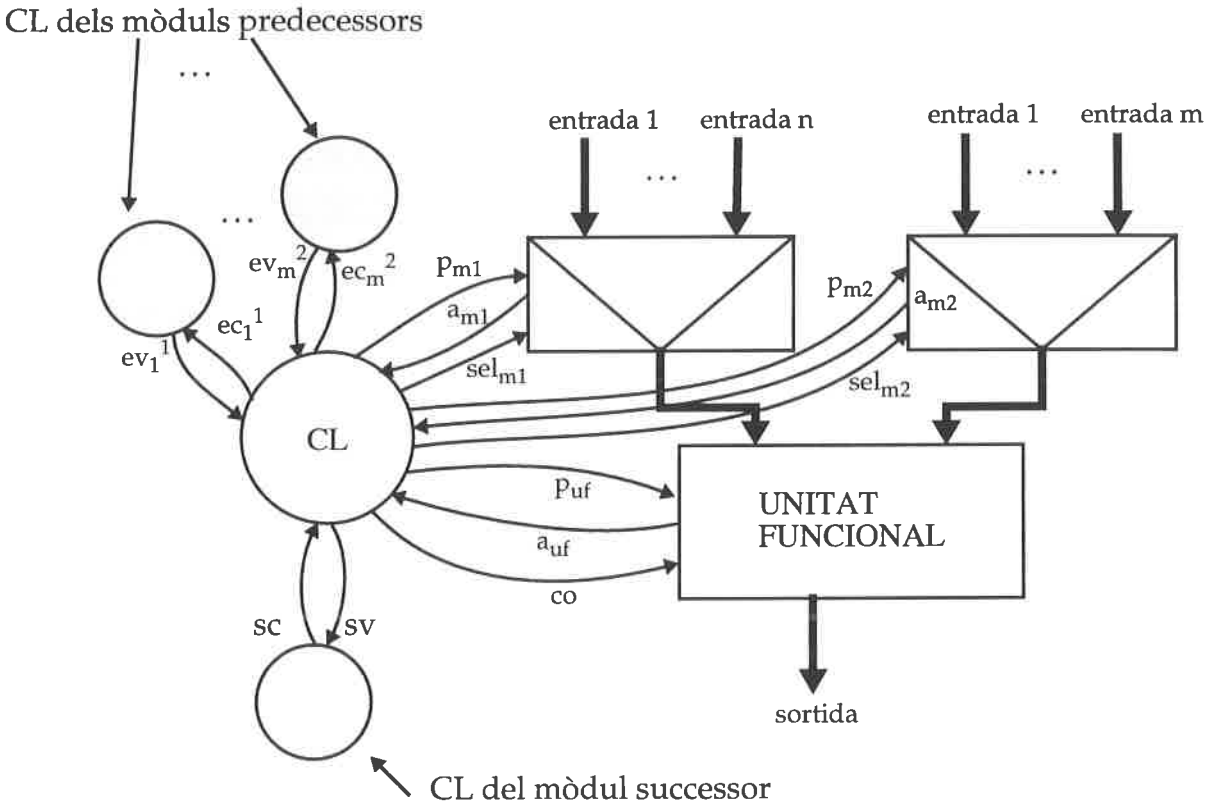


Figura 4.16: Processador format per dos multiplexors i una unitat funcional i el seu corresponent CL

i de la unitat funcional. Els senyals ev_i^1 i ev_j^2 representen els senyals d'*entrada vàlida* que arriben al controlador per indicar si l'entrada i del primer multiplexor i l'entrada j del segon multiplexor són vàlides o no. De la mateixa manera, el controlador genera els senyals ec_i^1 i ec_j^2 per indicar als corresponents controladors locals si aquestes entrades han estat consumides.

La figura 4.16 mostra l'esquema del processador format pels dos multiplexors i la unitat funcional amb el seu controlador local.

4.5 Conclusions

En tot sistema de síntesi de circuits, cal definir un model d'arquitectura on poder realitzar els dissenys obtinguts. En aquest capítol s'ha presentat un model d'arquitectura per a la síntesi d'alt nivell de circuits asíncrons basat en un sistema multiprocessador amb comunicació totalment asíncrona. En aquest sistema els components del camí de dades són realitzats amb lògica DCVSL i el control és totalment distribuït en controlador locals.

En un sistema asíncron no té sentit pensar en un control centralitzat, ja que això faria disminuir el seu rendiment i en complicaria el disseny.

Els controladors locals es sincronitzen amb altres controladors mitjançant senyals de control global. Aquestes sincronitzacions es produeixen quan s'han de realitzar transferències de dades entre els diferents processadors. A la vegada, els controladors locals es sincronitzen amb els components del camí de dades del propi processador mitjançant senyals de control local. Aquestes sincronitzacions controlen l'inici i finalització de les operacions.

El comportament dels controladors locals es pot descriure amb grafs de transicions de senyals (STG). Dels STG podem obtenir circuits asíncrons lliures de riscos utilitzant tècniques existents de síntesi a partir d'STG. En aquest capítol s'han descrit d'aquestes tècniques, fent especial èmfasi en les tècniques desenvolupades en el grup d'investigació.

S'ha proposat una metodologia per obtenir els STG que descriuen el comportament dels controladors locals. Aquesta metodologia es pot aplicar després de les fases de planificació d'operacions i assignació de recursos de la síntesi d'alt nivell. Després d'aquestes fases, queda totalment definit quines operacions s'executaran en cada component i en quin ordre, de manera que es coneix quin ha de ser el comportament dels controladors. S'han

caracteritzat els STG dels diferents tipus de components del camí de dades.

Una extensió de la metodologia presenta com agrupar un conjunt de processadors per obtenir-ne un de sol. El component de càlcul del nou processador estarà format pels components del camí de dades de tots els que s'agrupen. La descripció del component de control del nou processador s'obté agrupant els STG inicials, eliminant alguns senyals de sincronització global.

Capítol 5

ELS i ELLAS: planificació d'operacions

La planificació d'operacions és una de las fases principals de la síntesi d'alt nivell. S'hi distribueixen les operacions en el temps, buscant un compromís òptim entre el temps d'execució i el nombre de recursos utilitzats. Aquest capítol presenta una metodologia de planificació d'operacions basada en llistes d'esdeveniments, orientada a la síntesi d'alt nivell de circuits asíncrons.

5.1 Introducció

Com s'ha vist al capítol 4, el temps d'execució de les operacions en sistemes asíncrons no és un temps conegut, sinó que depèn de les dades d'entrada. Aquest punt és fonamental en el moment de plantejar-se la planificació d'operacions de sistemes asíncrons. La planificació d'operacions en aquest cas no estableix un temps de planificació exacte, sinó un temps estimat, ja que:

- No existeix temps de cicle i, per tant, les operacions poden començar i acabar en qualsevol instant de temps.
- La duració de les operacions és variable.

El que és important no és en si aquest temps de planificació, sinó la relació entre les execucions de les operacions. Per tant, la planificació d'operacions en sistemes asíncrons ha d'establir un ordre parcial en l'execució de les operacions.

En aquest capítol es presenta una metodologia per fer planificació d'operacions, presentant noves estructures de dades i funcions que seran bàsiques en la planificació d'operacions de sistemes asíncrons. Els algorismes presentats (*ELS* i *ELLAS*) fan planificació d'operacions basada en llistes d'esdeveniments.

A la següent secció es defineix el problema a resoldre i quines entrades i sortides tindran els algorismes proposats. La secció 5.3 defineix estructures de dades i funcions bàsiques per a la planificació d'operacions. La secció 5.4 presenta dos algorismes de planificació d'operacions basats en llistes d'esdeveniments. La secció 5.5 presenta els resultats obtinguts amb aquests algorismes i, per últim, es presenten algunes conclusions.

5.2 Definició del problema

5.2.1 Entorn del sistema

La figura 5.1 mostra les entrades i sortides dels algorismes de planificació proposats en aquest capítol. Una de les entrades és el graf de flux de dades (*DFG*), $G = (V, A)$ on cada vèrtex representa una operació i els arcs entre vèrtexs indiquen dependències de dades

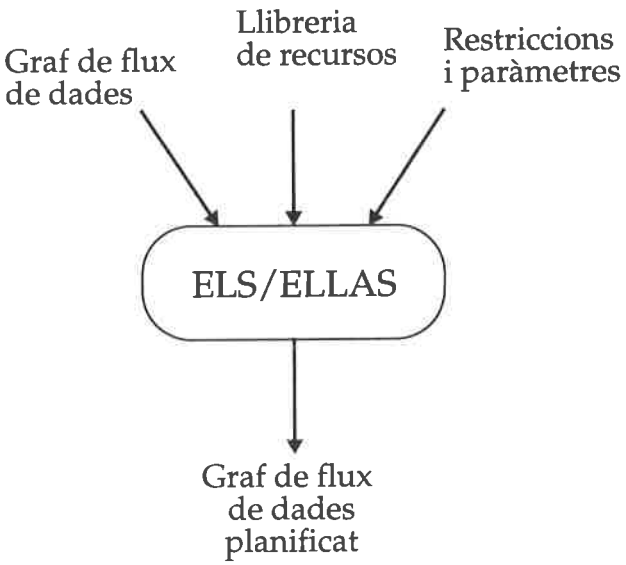


Figura 5.1: Entorn dels algorismes ELS *i* ELLAS

entre ells. Una altra entrada és la llibreria de recursos que conté informació sobre quins recursos estan disponibles, quines operacions és capaç d'executar cada recurs i quin és el temps mitjà d'execució de les operacions en cada recurs. Per últim, el sistema també tindrà com a entrades algunes restriccions i paràmetres. Les restriccions són d'àrea i s'indiquen com el nombre de recursos de cada tipus disponibles per fer la planificació.

5.2.2 Nomenclatura

Per a cada vèrtex $v \in V$ del *DFG*, $G = (V, A)$ definirem la següent terminologia:

v_o : operació executada al vèrtex v

v_i, v_a : temps d'*inici* i *acabament* esperats (després de planificar)

v_u : tipus d'unitat funcional que executa v_o

$pred(v) = \{u \mid (u, v) \in A\}$

$succ(v) = \{u \mid (v, u) \in A\}$

Unitat Funcional	*retard mitjà*					
	+	-	*and*	*or*	×	/
RCA	80	80	∞	∞	∞	∞
CLA	40	40	∞	∞	∞	∞
ALU	60	60	20	20	∞	∞
MUL	∞	∞	∞	∞	200	∞
DIV	∞	∞	∞	∞	∞	400

Taula 5.1: Llibreria de recursos representada en forma de matriu de retards

5.2.2.1 Matriu de retards

La llibreria de recursos es representa mitjançant una *matriu de retards.* Els recursos de la llibreria poden ser multifunció i un tipus d'operació pot ésser executat per diferents tipus de recursos. $\delta_{u,o}$ representa un element de la *matriu de retards* δ on u és el tipus de recurs i o el tipus d'operació. $\delta_{u,o}$ és el retard mitjà del recurs u per executar una operació del tipus o.

La taula 5.1 mostra un exemple de llibreria de recursos representada en forma de *matriu de retards* δ. Les files representen recursos disponibles i les columnes representen operacions possibles. Cada element de la matriu conté el temps mitjà estimat d'execució de l'operació *columna* al recurs *fila.* Si aquest valor és ∞ significa que el recurs no pot executar l'operació.

Definim $UF(o)$ com el conjunt d'unitats funcionals que poden executar operacions del tipus o, és a dir:

$$UF(o) = \{u | \delta_{u,o} < \infty\} \tag{5.1}$$

Les restriccions d'àrea es representaran en forma d'un *vector de recursos,* $R =< |u_1|, |u_2|, \ldots, |u_n| >$, on $|u_i|$ representa el nombre de recursos disponibles del tipus u_i. Suposarem que $|u_i| > 0$ (si $|u_i| = 0$, el recurs u_i pot ser eliminat de la llibreria).

Donat un *vector de recursos* R, definirem el *temps estimat d'execució d'una operació* o,

$\overline{\delta_o}$, com:

$$\overline{\delta_o} = \frac{\sum_{u_i \in UF(o)} |u_i| \delta_{u_i,o}}{\sum_{u_i \in UF(o)} |u_i|} \tag{5.2}$$

És a dir, $\overline{\delta_o}$ és una estimació del que pot trigar (en mitjana) a executar-se una operació de tipus o suposant que totes les unitats funcionals u, tals que $u \in F(v_o)$, tenen la mateixa probabilitat de ser assignades al vèrtex v.

Per exemple, si el vector de recursos per la matriu de retards de la taula 5.1 és $R =< 1, 2, 1, 1, 1 >$, el temps estimat d'execució de la suma és:

$$\overline{\delta_+} = (80 + 2 \times 40 + 60)/(1 + 2 + 1) = 55$$

5.2.3 Model d'execució

El model d'arquitectura en què es basen les estratègies de planificació descrites en aquest capítol és el que s'ha presentat en el capítol anterior. Pel que fa al model d'execució, restringirem l'encadenament d'operacions de manera que les unitats funcionals llegeixen les dades de registres i el resultat de les operacions s'emmagatzema en registres. Per simplificar, considerarem que el retard de les lectures i escriptures de registres és un retard independent de les dades i inclourem aquests retards en els de les operacions.

5.2.4 Formulació del Problema

Donat un *DFG*, una llibreria de recursos disponibles representada per δ i un conjunt de recursos disponibles representat per R, el problema que volem resoldre és trobar una planificació asíncrona de les operacions representades en el graf, tal que l'estimació del seu retard sigui mínima.

Trobar una planificació asíncrona significa establir un ordre parcial dels vèrtexs. La planificació defineix un temps de planificació estimat per a cada operació. Per poder definir aquesta planificació, les operacions són assignades a un tipus d'unitat funcional, amb l'objectiu de minimitzar el temps total d'execució.

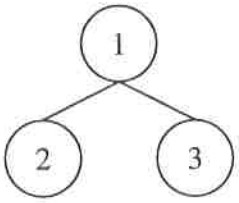


Figura 5.2: Subgraf

i	1	2	3	4	5
Prob(i)	0,2	0,2	0,2	0,2	0,2
i	2	4	6	8	10
Prob(i)	0,2	0,2	0,2	0,2	0,2

Taula 5.2: Distribució del retard de les operacions

En una fase posterior, l'associació de recursos definirà en quin recurs concret s'executarà cada operació. Un cop s'han realitzat les fases de planificació i d'associació, el comportament dels diferents *controladors locals* queda determinat, així com les sincronitzacions que existiran entre operacions. Les sincronitzacions entre els *controladors locals* dels mòduls existiran quan hi hagi dependències de dades entre les operacions.

Per fer una estimació del temps total d'execució d'una planificació, utilitzarem els *retards mitjans* de les operacions continguts a la *matriu de retards* δ. Aquesta aproximació no és del tot correcte, però es farà servir per simplificar l'algorisme. En la majoria dels casos ens acostarem a la realitat, tot i que de vegades la nostra predicció serà optimista.

Per exemple, el temps mitjà d'execució d'un subgraf com el mostrat a la figura 5.2 s'avalua com la suma del retard mitjà d'execució de l'operació del vèrtex 1 més el retard mitjà d'execució de les operacions dels vèrtexs 2 i 3 en paral·lel. Aquest últim retard es correspon amb el de l'operació que acaba més tard.

Imaginem que el retard de les operacions dels vèrtexs 2 i 3 pren valors discrets entre 1 i 5 i segueix una funció de distribució uniforme, tal com mostra la primera fila de taula 5.2. En aquest cas, el retard mitjà d'execució de cadascuna de les operacions de manera independent és 3 i el retard mitjà d'execució de les dues operacions en paral·lel és 3,8. L'error

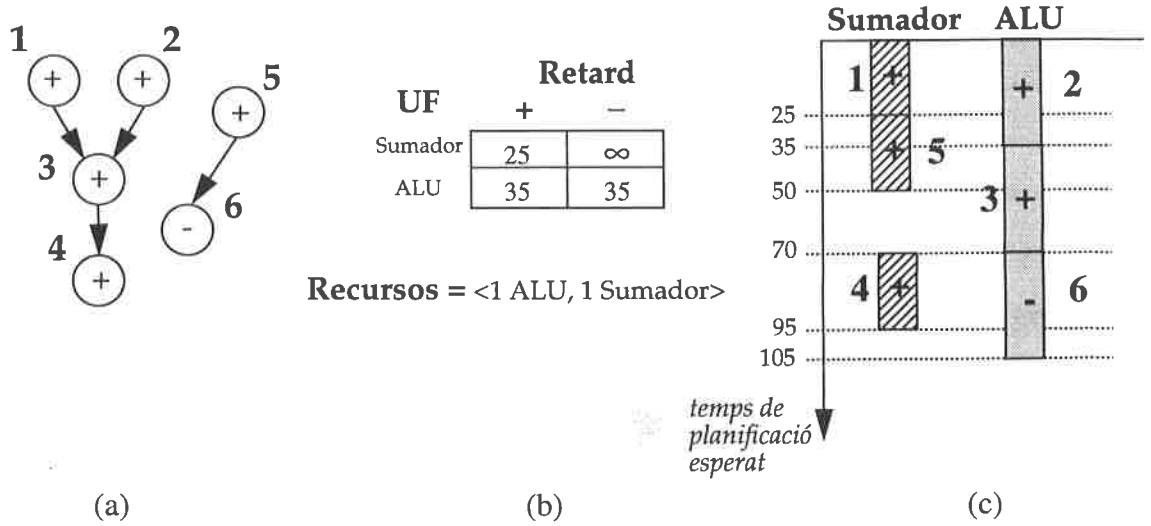


Figura 5.3: (a) Graf de flux de dades; (b) Llibreria i vector de recursos*; (c) Graf de flux de dades planificat*

comés en aquest cas si estimem el retard d'execució de les dues operacions en 3 és d'un 27%.

Aquest és el pitjor dels casos (quan les dues operacions tenen el mateix retard), però si considerem altres casos l'error comés és inferior. Per exemple, si l'operació del vèrtex 3 ara té un retard mitjà de 6 unitats i segueix una distribució com la que es mostra a la fila 2 de la taula 5.2, el retard mitjà d'execució de les dues operacions en paral·lel és de 6,28. En aquest cas si estimem el retard en 6 es comet un error del 4,7%.

Si es tractessin aplicacions en temps real, s'hauria de considerar el retard en el cas pitjor de les unitats funcionals.

Per planificar l'operació representada per un vèrtex $v \in V$, cal definir els temps d'inici i d'acabament esperats d'aquesta operació (v_i, v_a) i el tipus d'unitat funcional en què s'executarà. La figura 5.3 mostra un exemple del que es vol aconseguir.

5.3 Estructures de dades i funcions

En aquesta secció es definiran algunes estructures de dades i funcions que seran fonamentals en els algorismes *ELS* i *ELLAS*.

5.3.1 Estructures de dades

Les estructures de dades que descriurem són la *taula de reserva* i la *llista d'esdeveniments.*

5.3.1.1 Taula de reserva

La *taula de reserva d'un tipus d'unitat funcional* u_i *(*TR_{u_i}*)* és una estructura de dades que conté informació sobre el nombre d'unitats del tipus que són actives en cada instant de temps. El fet que un recurs estigui actiu en un moment donat, significa que està executant alguna operació, és a dir, que està ocupat. Cada vegada que planifiquem una operació del *DFG* i l'assignem a un tipus de recurs u_i, la TR_{u_i} és actualitzada. És evident que el valor de la TR_{u_i} no pot ser més gran que $|u_i|$ per a cap instant de temps.

5.3.1.2 Llista d'esdeveniments

Podem representar una taula de reserva TR_{u_i} com una *llista d'esdeveniments* LE_{u_i}. La llista d'esdeveniments LE_{u_i} és una llista de parells $< temps_i, nufs_i >$ ordenada per temps, on $nufs_i$ és el nombre de recursos disponibles del tipus u_i en l'interval de temps que comença a $temps_i$ i acaba a $temps_{i+1}$. Inicialment, cada llista LE_{u_i} conté un esdeveniment: $< 0, |u_i| >$, és a dir que de temps 0 a temps infinit tenim $|u_i|$ recursos de tipus u_i disponibles. Les figures 5.4.a i 5.4.b mostren la *taula de reserva* i la *llista d'esdeveniments* corresponent al DFG parcialment planificat de la figura 5.4.c.

5.3.2 Funcions per gestionar la llista d'esdeveniments

A continuació descriurem dues funcions per a la gestió de les *llistes d'esdeveniments*: *trobar_interval_lliure* i *reservar_interval.*

5.3.2.1 trobar_interval_lliure

L'especificació de la funció és la següent:

`temps_inici = buscar_interval_lliure (`LE_{u_i}`, temps_inici_mínim, retard)`

Aquesta funció busca en una llista d'esdeveniments LE_{u_i} el primer interval de duració *retard* amb $temps\_inici \geq temps\_inici\_mínim$ tal que hi hagi almenys una unitat funcional

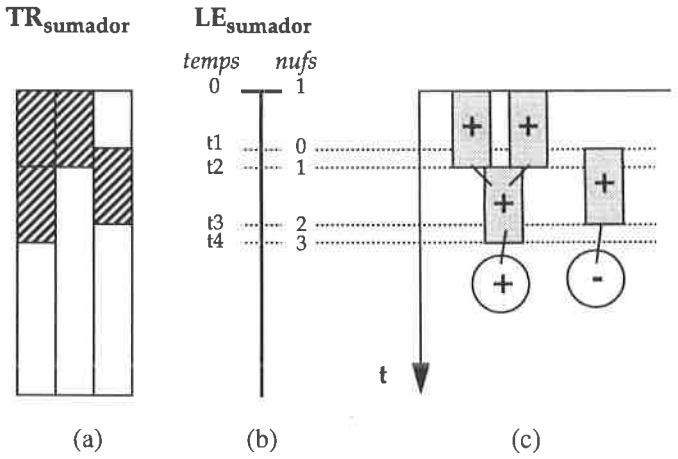


Figura 5.4: (a) Taula de Reserva del tipus d'unitat funcional sumador*; (b) Llista d'esdeveniments del tipus* sumador*; (c)* DFG *parcialment planificat*

lliure de tipus u_i (la figura 5.5 en mostra un exemple). Aquesta funció es pot implementar com una combinació dels algorismes de *cerca binària* (per trobar el primer esdeveniment on començar la recerca) i *first fit* (per trobar el forat de mida *retard*). La complexitat de la funció `buscar_interval_lliure` és $O(\log n)$ on n és el nombre de vèrtexs del *DFG*, ja que tot i que el *first fit* fa una recerca local seqüencial, la *cerca binària* domina la funció [CLR90].

5.3.2.2 reservar_interval

La funció *reservar_interval* s'especifica a continuació:

`reservar_interval (`LE_{u_i}`, temps_inici, retard)`

Aquesta funció reserva un interval de duració *retard* i inici a *temps_inici*. La llista d'esdeveniments LE_{u_i} és actualitzada de manera que un dels recursos de tipus u_i esdevingi actiu en l'interval de temps $< temps\_inici, temps\_inici+retard >$, tal com mostra la figura 5.6. Aquesta funció executa dues *cerques binàries* i, per tant, també té complexitat $O(\log n)$.

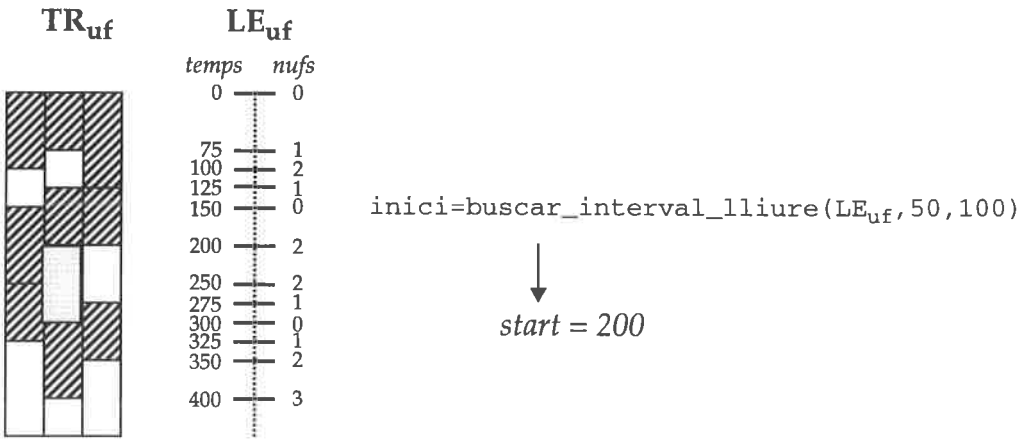


Figura 5.5: Execució de la funció buscar_interval_lliure

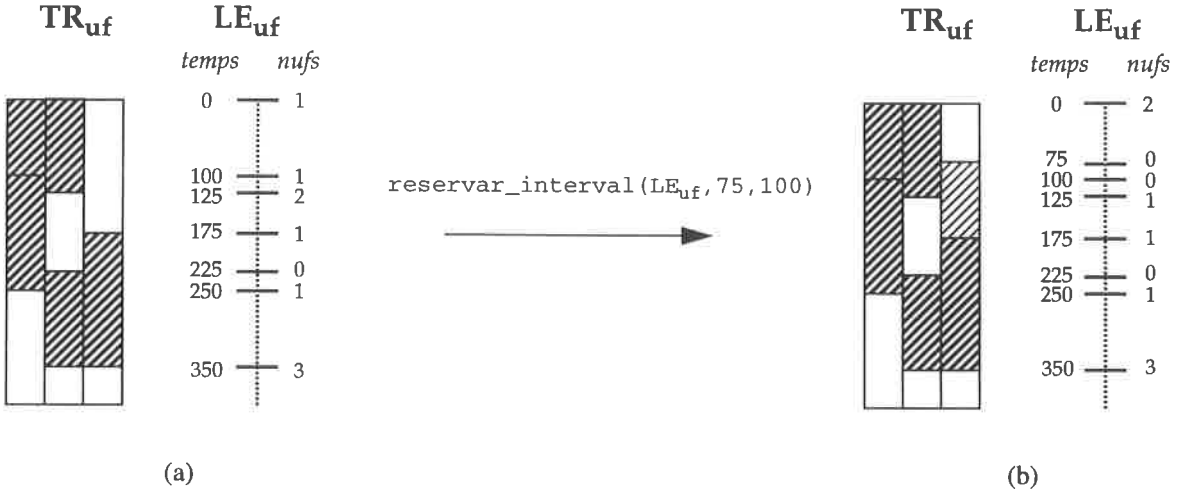


Figura 5.6: Efecte de l'execució de la funció reservar_interval *en una llista d'esdeveniments*

5.4 Algorismes basats en llistes d'esdeveniments

Els algorismes de planificació d'operacions que es presenten a continuació estan basats en l'estructura de *llistes d'esdeveniments* presentada en la secció anterior. Els vèrtexs del graf són triats per planificar-los segons una funció de prioritat.

La diferència principal entre *ELS* i *ELLAS* es troba en la funció de prioritat utilitzada. Mentre *ELS* utilitza una funció de la longitud del camí més llarg des del vèrtex fins al final del graf, *ELLAS* utilitza una funció dinàmica avaluda en cada iteració com a resultat d'una planificació *look-ahead.*

Durant l'execució dels algorismes de planificació, el conjunt V de vèrtexs del *DFG* pot ser particionat en tres conjunts disjunts:

- Conjunt de *Vèrtexs Planificats (VPl)*: conté tots els vèrtexs que ja han estat planificats.
- Conjunt de *Vèrtexs Preparats (VP)*: conté tots els vèrtexs que estan preparats. Un vèrtex $u \in V$ es considera preparat si per tot $u \in pred(v)$, $u \in VPl$, és a dir, tots els predecessors del vèrtexs han estat planificats.
- Conjunt de *Vèrtexs No-Planificats (VNPl)*: conté la resta dels vèrtexs que no pertanyen ni a *VP* ni a *VPl.*

L'algorisme de planificació bàsic de planificació amb llistes d'esdeveniments es descriu a l'algorisme 5.1. Aquest algorisme realitza el següent procés:

1. Inicialment totes les llistes d'esdeveniments LE_u són inicialitzades amb el nombre corresponent de recursos fixat pel *vector de recursos* ($|u|$).
2. Es calculen les prioritats dels vèrtexs i s'inicialitzen els diferents conjunts de vèrtex (*VP, VPl, VNPL*). Com que un vèrtex no pot ser planificat fins que els seus predecessors ja ho siguin, inicialment el conjunt *VP* és el conjunt de vèrtexs *font* del graf.
3. El bucle exterior es repeteix fins que el conjunt *VP* sigui buit, és a dir, tots els vèrtexs han estat planificats. Dins d'aquest bucle, es tria aquell vèrtex v del conjunt *VP* amb prioritat màxima.

planificació amb llistes d'esdeveniments $(G(V,A),\ \delta,\ R)$ {

per a cada u de la llibreria **fer** inicialitzar_llista_esdeveniments $(LE_u,\ |u|)$;

calcular_prioritats $(G,\ \delta,\ R)$;

$VP =$ vèrtexs_font(G);

$VNPl = V - VP;\ VPl = \emptyset$;

mentre $VP \neq \emptyset$ **fer**

$v =$ vèrtex_prioritat_màxima(VP);

temps_inici_mínim $= \max_{u \in pred(v)} u_a$;

per a cada $u \in UF(v_o)$ **fer**

temps_inici_mínim$_u$ = buscar_interval_lliure $(LE_u,$ temps_inici_mínim, $\delta_{u,v_o})$;

acabament$_u$ = temps_inici_mínim$_u + \delta_{u,v_o}$;

fiper

$u_{min} = u$ **tal que** (acabament$_{u_{min}} = \min_{u \in UF(v_o)}$ acabament$_u$);

reservar_interval ($LE_{u_{min}}$, temps_inici_mínim$_{u_{min}}$, δ_{u_{min},v_o});

$v_i =$ temps_inici_mínim$_{u_{min}}$; $v_a =$ acabament$_{u_{min}}$; $v_u = u_{min}$;

$VPl = VPl \cup \{v\}$;

nous_preparats $= \{v \in VNPl \mid \forall u \in \text{pred}(v), u \in VPl\}$;

$VP = (VP - \{v\}) \cup$ nous_preparats; $VNPl = VNPl -$ nous_preparats;

fimentre

}

Algorisme 5.1: Algorisme de planificació d'operacions ELS

4. A continuació, es calcula el temps mínim en què es pot iniciar l'execució d'aquest vèrtex pel que fa a les dependències de dades (*temps_inici_mínim*).

5. Seguidament, per a cada unitat funcional u tal que és capaç d'executar operacions del tipus u_o, es calcula el temps mínim d'inici. Aquest càlcul es realitza mitjançant la funció `buscar_interval_lliure`. També es calcula el temps d'acabament de l'operació.

6. D'aquest conjunt d'unitats funcionals, es tria aquell tipus tal que el temps d'acabament calculat sigui mínim.

7. En aquest punt, es reserva l'interval corresponent per a aquesta operació a la *LE*, es planifica el vèrtex (assignant els valors corresponents a v_i i v_a) i s'assigna el tipus d'unitat funcional al vèrtex.

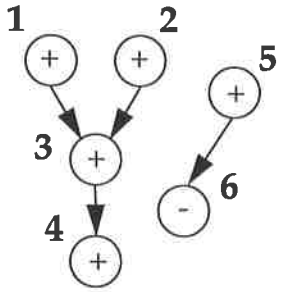
8. Els conjunts de vèrtexs *VP, VPl* i *VNPl* són actualitzats. El vèrtex seleccionat passa a formar part del conjunt *VPl*. A més, alguns dels vèrtexs del conjunt *VNPl* poden passar al conjunt *VP*. La condició que han de complir aquests vèrtexs és que tots els seus predecessors han d'estar planificats.

5.4.1 ELS

La diferència entre els algorismes *ELS* i *ELLAS* es troba en la funció de prioritat que s'utilitza en cada cas per seleccionar el vèrtex a planificar.

L'algorisme *ELS* assigna primer una prioritat a cada vèrtex del *DFG*. Aleshores, els vèrtexs del conjunt *VP* són planificats en ordre segons la seva prioritat. La prioritat de cada vèrtex v, v_p, és calculada com la longitud del camí més llarg des del vèrtex fins al final del *DFG*, suposant que cada vèrtex v és executat en temps $\overline{\delta_{v_o}}$. El càlcul de v_p es pot fer de manera recursiva des dels vèrtexs *destí* fins als vèrtexs *font* tal com es descriu a continuació:

$$v_p = \max_{u \in succ(v)} u_p + \overline{\delta_{v_o}} \tag{5.3}$$



(a)

UF	Retard +	Retard −
Sumador	25	∞
ALU	35	35

(b)

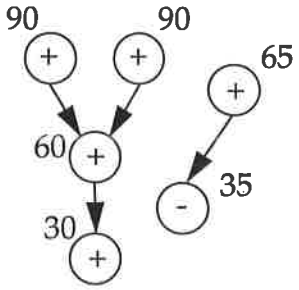
R = < 1 sumador, 1 ALU >

(c)

$$\overline{\delta}_{+} = \frac{1 \times 25 + 1 \times 35}{1 + 1} = 30$$

$$\overline{\delta}_{-} = \frac{1 \times 35}{1} = 35$$

(d)



(e)

Figura 5.7: (a) Graf de flux de dades; (b) Matriu de retards; (c) Vector de recursos; (d) Prioritats dels vèrtexs del DFG anterior

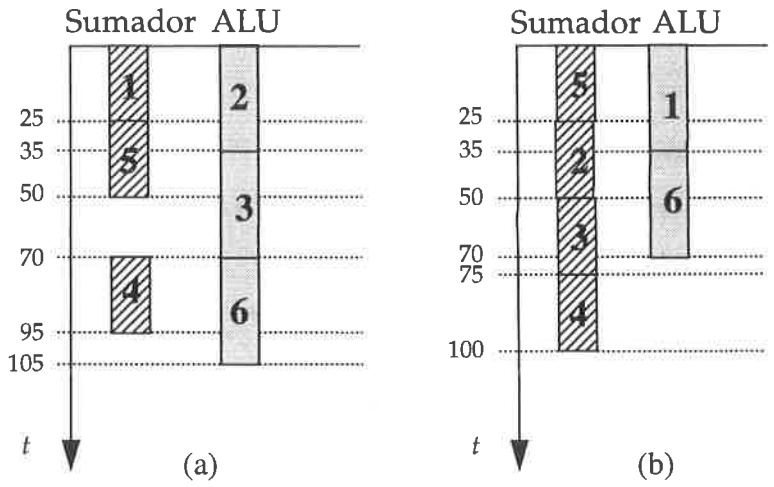


Figura 5.8: (a) Planificació obtinguda per ELS*; (b) Planificació obtinguda per* ELLAS

La figura 5.7.d mostra un exemple dels valors que prenen les prioritats dels vèrtexs d'un *DFG* per una determinada *matriu de retards* i vector de recursos R. Es pot apreciar com els vèrtexs del camí crític prenen valors de prioritat més grans que els altres.

La figura 5.8.a mostra la planificació obtinguda per *ELS* per al *DFG*, matriu de retards i vector de recursos de la figura 5.3.

ELS utilitza una estratègia ambiciosa per seleccionar el següent vèrtex a planificar: tria el vèrtex amb prioritat màxima. Aquesta estratègia pot obtenir resultats pitjors als que s'obtindríem si es considerés informació més global. A la següent secció es millora l'algorisme de *ELS* amb una funció de *look-ahead* que calcula la prioritat dels vèrtexs de manera dinàmica.

5.4.2 ELLAS

L'algorisme *ELS* selecciona a cada iteració el vèrtex amb prioritat màxima. En canvi, *ELLAS* utilitza una funció de *look-ahead* per seleccionar el vèrtex del conjunt *VP* a planificar. Aquesta funció, per a cada vèrtex de *VP*, fa una estimació del temps de planificació que s'obtindria si es planifiqués aquest vèrtex primer, de manera que es tria aquell vèrtex tal que l'estimació és mínima. L'algorisme *ELLAS* és idèntic al d'*ELS*, excepte en la funció `vèrtex_prioritat_màxima`, que és reemplaçat per la funció `prioritat_look_ahead`.

Aquesta funció es descriu a continuació:

prioritat_look_ahead (VP) {
  per a cada $v \in VP$ **fer**
    temps_planificació$_v$ = look_ahead_ELS(v);
  fiper
  $v_{min} = v$ **tal que** (temps_planificació$_{v_{min}} = \min_{v \in VP}$ temps_planificació$_v$);
  retornar(v_{min});
}

La funció `look_ahead_ELS` planifica els vèrtexs de *DFG* que queden per planificar, començant pel vèrtex v. Per seleccionar els vèrtexs a planificar, utilitza l'estratègia *ELS*. Retorna el temps de planificació obtingut. Com que es tracta d'una funció de *look-ahead*, totes les transformacions que es fan en el *DFG* i en les llistes d'esdeveniments han de ser desfetes fins que quedin com estaven abans de la crida a la funció.

La figura 5.8.b mostra la planificació obtinguda per *ELLAS* pel mateix exemple presentat per *ELS*. Com es pot apreciar, el temps de planificació ha estat reduït en utilitzar *ELLAS*. El canvi més significant es produeix pel fet de planificar primer el vèrtex 5 abans dels vèrtexs 1 i 2. Aquesta decisió permet que el vèrtex 6 sigui planificat immediatament després de l'1 obtenint un temps de planificació mínim.

5.4.3 Complexitat d'ELS i ELLAS

A *ELS* el bucle extern és executat n vegades, essent n el nombre de vèrtexs de *DFG*. Si m és el nombre de tipus d'unitats funcionals que poden executar una operació, aleshores la complexitat de cada iteració és $O(m \log n)$, ja que la funció `buscar_interval_lliure` té complexitat $O(\log n)$. Per tant, *ELS* pot ser executat en temps $O(mn \log n)$.

En el cas de *ELLAS*, la complexitat està dominada per la funció de *look-ahead*, on es fa una planificació utilitzant l'estratègia de *ELS*. Si r és el nombre mitjà de vèrtexs del conjunt *VP*, la complexitat de *ELLAS* és $O(rmn^2 \log n)$. El valor de r està directament relacionat amb el paral·lelisme inherent del *DFG*.

Unitat Funcional	*retard*			
	+	-	<	×
$ALU(\diamond)$	50	50	50	∞
$sumador(\oplus)$	35	∞	∞	∞
$mult(\otimes)$	∞	∞	∞	85

Taula 5.3: Matriu de retards utilitzada per als exemples

Si considerem que m és una constant i que el cas pitjor de r és n, tenim que la complexitat de *ELS* és $O(n \log n)$ i la de *ELLAS* $O(n^3 \log n)$.

ELS i *ELLAS* han estat codificats en lleguatge C i ocupen unes 1.300 línies cadascun.

5.5 Resultats

A continuació presentarem alguns resultats obtinguts amb *ELS* i *ELLAS*. S'han triat quatre exemples per obtenir aquestes dades: l'equació diferencial [PK87], el filtre AR-lattice [JMP88], el filtre el·líptic de cinquè ordre [DDN85] i la transformada discreta del cosinus [Wou, KN92]. Primerament, es presenten els resultats que s'han obtingut quan es considera un model d'arquitectura asíncrona. Després, s'utilitza un model d'arquitectura síncrona, per tal de poder comparar els resultats de *ELS* i *ELLAS* amb altres algorismes existents.

5.5.1 Resultats per a sistemes asíncrons

La llibreria utilitzada pels tres primers exemples és la mateixa i es descriu a la *matriu de retards* de la taula 5.3.

En aquests retards, a més del retard del recurs per realitzar l'operació, se suposa inclòs el temps de llegir els operands de registres, el temps de guardar el resultat en un registre i el temps d'inicialitzar el recurs.

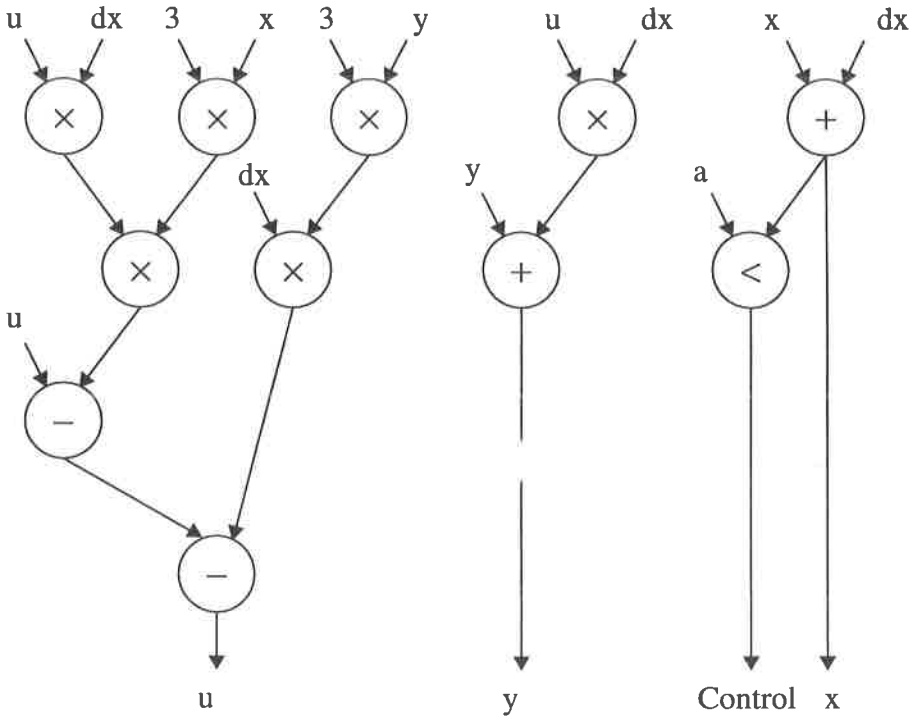


Figura 5.9: Graf de flux de dades de l'equació diferencial

5.5.1.1 Equació diferencial

Aquest exemple va ser proposat per primera vegada a [PKG86] i des d'aleshores ha estat molt utilitzat en planificació d'operacions i assignació de recursos. L'exemple parteix de l'equació diferencial:

$$y'' + 3xy' + 3y = 0$$

que pot ser solucionada amb el següent algorisme:

mentre $x < a$ fer
$x = x + dx;$
$u = u - (3 * x * u * dx) - (3 * y * dx);$
$y = y + (u * dx);$
fimentre

D'aquest algorisme es pot extreure un DFG que és el que mostra la figura 5.9.

La taula 5.4 mostra els resultats obtinguts per a l'equació diferencial utilitzant diferents *vectors de recursos*. A causa de la petitesa d'aquest exemple, les planificacions obtingudes

Recursos	ELS		ELLAS	
	Planificació	temps CPU	Planificació	temps CPU
$\otimes \otimes \otimes \oplus \diamond$	270 ns	0,01 s	270 ns	0,015 s
$\otimes\otimes \oplus \diamond$	305 ns	0,01 s	305 ns	0,015 s
$\otimes \oplus \diamond$	545 ns	0,01 s	545 ns	0,01 s

Taula 5.4: Resultats per l'equació diferencial (temps de CPU en un DECsystem 5100)

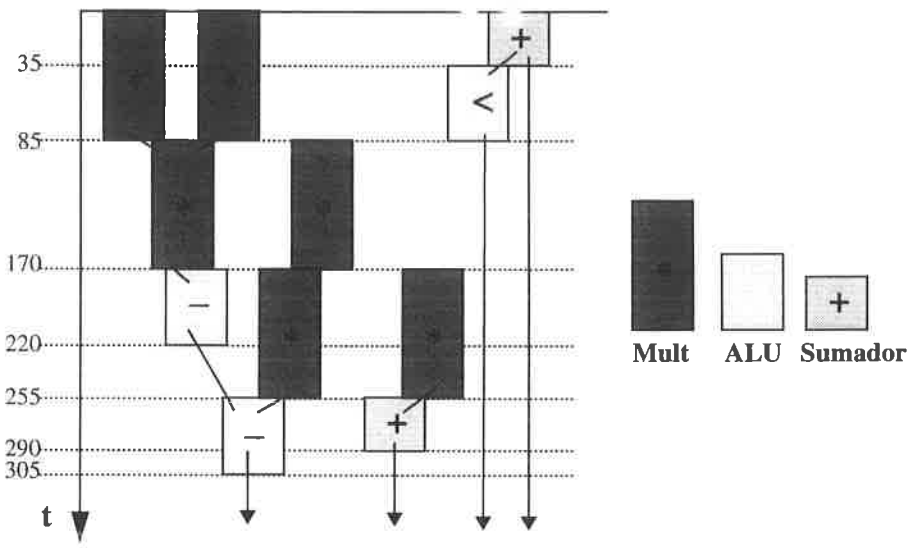


*Figura 5.10: Planificació per a l'*equació diferencial *(Recursos: $\otimes \otimes \oplus \diamond$)*

per *ELS* i *ELLAS* són iguals i els temps de CPU són similars. La figura 5.10 mostra la planificació obtinguda es un dels casos.

5.5.1.2 Filtre AR-lattice

El DFG del filtre AR-lattice és el que mostra la figura 5.11 [JMP88].

La taula 5.5 descriu diferents resultats que s'han obtingut per a aquest exemple utilitzant diferents restriccions de recursos. En aquest cas, com que el graf ja comença a ser una mica més gran, els resultats de *ELS* són pitjors que els de *ELLAS* per alguns casos. Per exemple, si restringim els recursos a un sumador i dos multiplicadors, el temps de

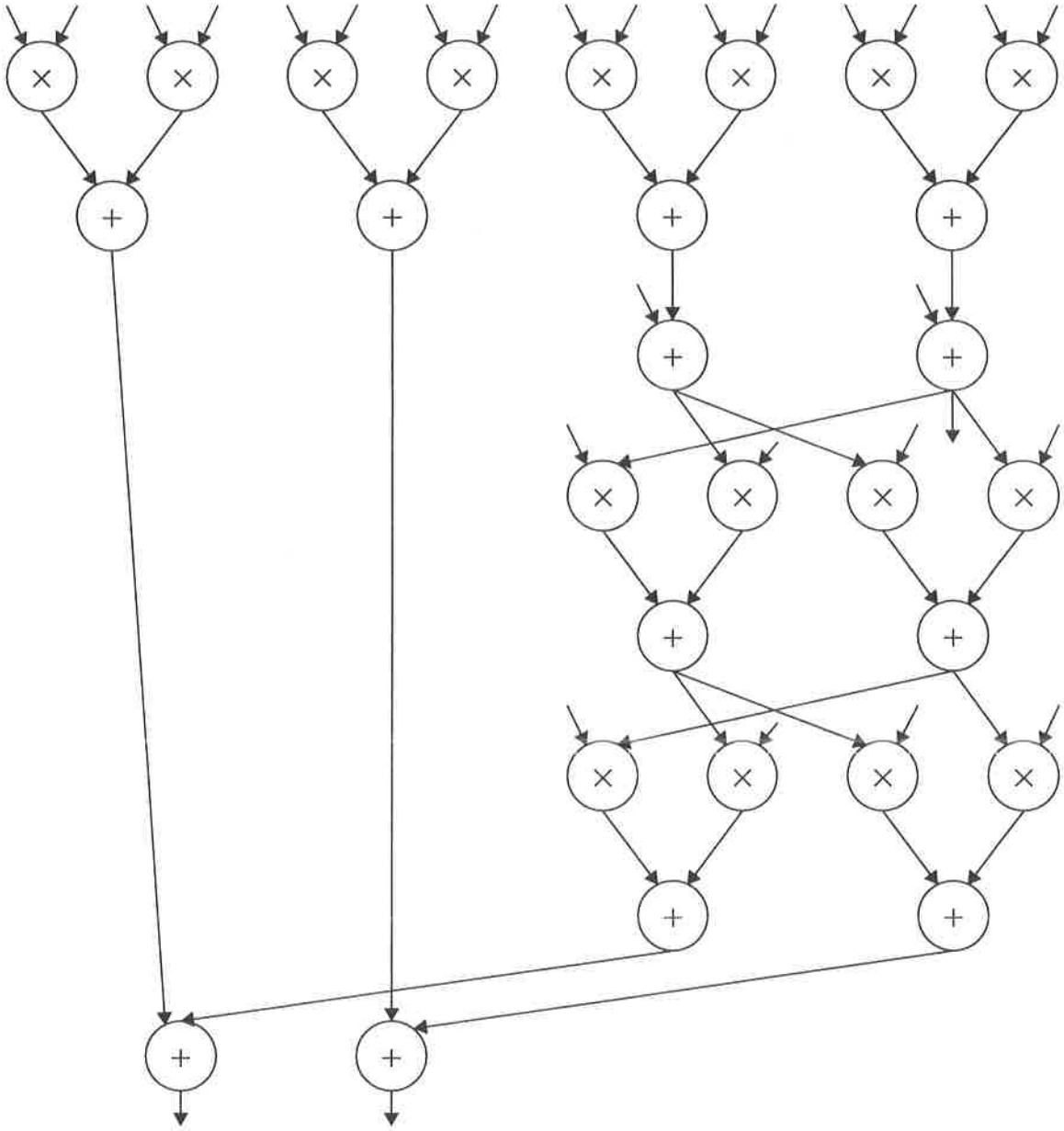


Figura 5.11: Graf de flux de dades del filtre AR-lattice

Recursos	ELS		ELLAS	
	Planificació	temps CPU	Planificació	temps CPU
⊗⊗ ⊕	820	0,016 s	750	0,083 s
⊗⊗ ⊕ ◇	780	0,016 s	750	0,116 s
⊗ ⊗ ⊗ ⊕⊕	650	0,016 s	600	0,083 s
⊗ ⊗ ⊗⊗ ⊕⊕	480	0,016 s	480	0,083 s
⊗ ⊗ ⊗ ⊗ ⊗ ⊕⊕	430	0,016 s	430	0,099 s

Taula 5.5: Resultats per al Filtre AR- ıttice si es considera un model d'arquitectura asíncrona (temps de CPU per una SUN SPARCstation 10)

planificació estimat de *ELS* és 820 i el de *ELLAS* 750, fet que significa una millora del 8%.

5.5.1.3 Filtre el·líptic

El següent exemple que es considera és el filtre el·líptic de cinquè ordre [DDN85]. Aquest exemple ha estat utilitzat per comparar diferents mètodes en múltiples ocasions. El graf de flux de dades per a aquest exemple és el que mostra la figura 5.12. El DFG representa el cos d'un bucle que es repeteix infinitament, de manera que les sortides d'una iteració són les entrades de la següent. El graf conté 34 operacions (26 sumes i 8 multiplicacions).

La taula 5.6 mostra els resultats que s'han obtingut. Els resultats obtinguts per *ELLAS* són millors que els de *ELS* en molts dels casos, al cost d'un temps de CPU més alt (tot i que molt moderat encara). La figura 5.13 compara les planificacions obtingudes per *ELS* i *ELLAS* en un dels casos.

5.5.1.4 Transformada discreta del cosinus

En aquest apartat es presenta un exemple més gran: la *transformada discreta del cosinus* (DCT) [Wou, KN92]. La DCT s'utilitza en codificació d'imatges i en aplicacions de compressió. La figura 5.14 mostra el DFG de la *transformada discreta del cosinus.*

En aquest cas, s'ha utilitzat la matriu de retards que mostra la taula 5.7. La taula 5.8 mostra els resultats obtinguts per aquest exemple.

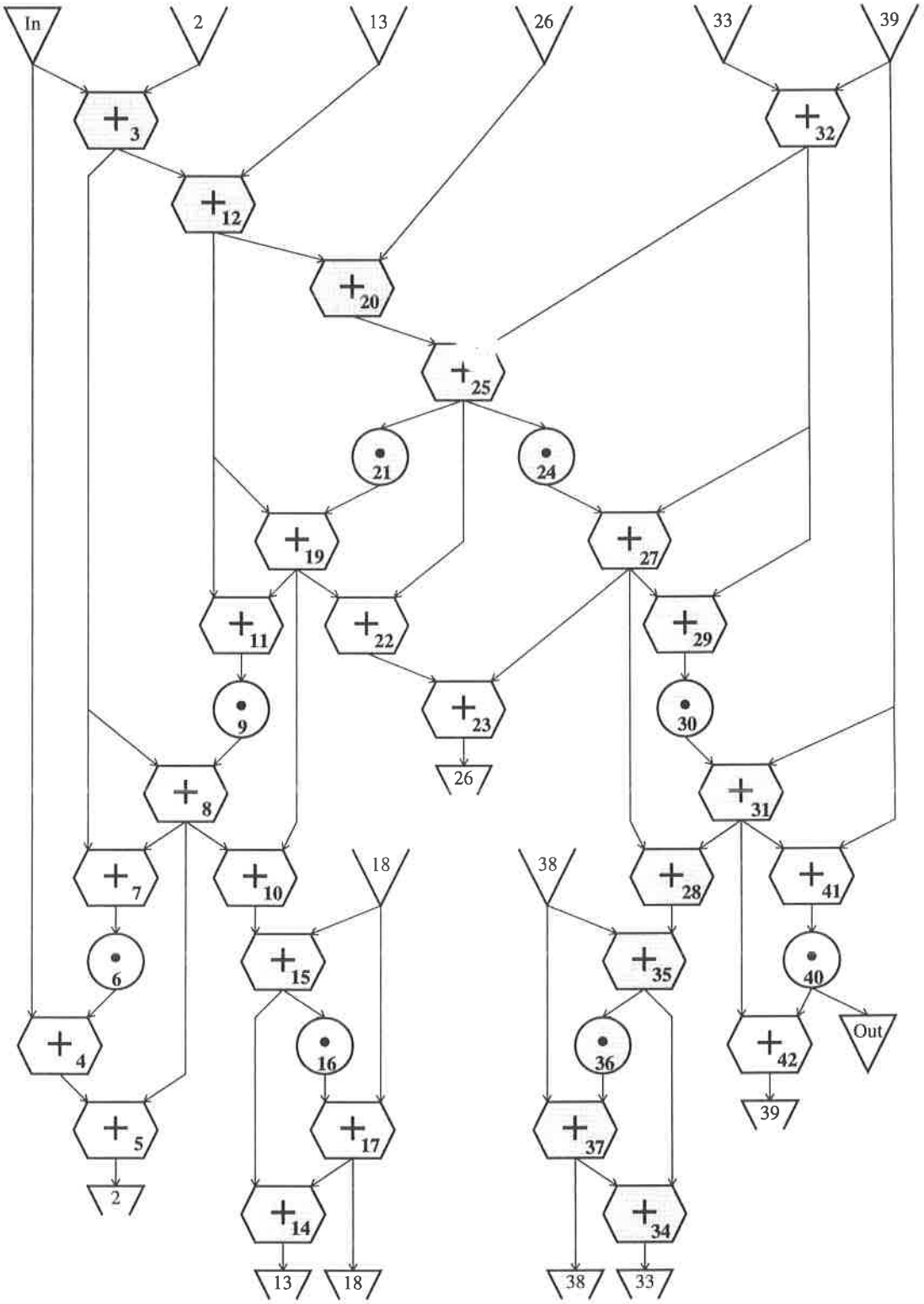


Figura 5.12: Graf de flux de dades del filtre el·líptic de cinquè ordre

Recursos	ELS		ELLAS	
	Planificació	temps CPU	Planificació	temps CPU
⊗⊗ ⊕ ◇	755 ns	0,01 s	755 ns	0,13 s
⊗⊗ ⊕ ◇◇	740 ns	0,01 s	705 ns	0,12 s
⊗⊗ ⊕⊕ ◇	705 ns	0,01 s	690 ns	0,12 s
⊗ ⊗ ⊗ ⊕ ◇	755 ns	0,01 s	745 ns	0,12 s
⊗ ⊗ ⊗ ⊕ ◇◇	720 ns	0,01 s	700 ns	0,12 s
⊗ ⊗ ⊗ ⊕ ◇◇◇	720 ns	0,01 s	685 ns	0,13 s
⊗ ⊗ ⊗ ⊕⊕ ◇	675 ns	0,01 s	670 ns	0,13 s
⊗ ⊗ ⊗ ⊕⊕ ◇◇	675 ns	0,01 s	655 ns	0,12 s
⊗ ⊗ ⊗⊗ ⊕ ◇◇◇	705 ns	0,01 s	690 ns	0,13 s
⊗ ⊗ ⊗⊗ ⊕ ⊕ ⊕ ◇	640 ns	0,01 s	640 ns	0,12 s

Taula 5.6: Resultats per al filtre el·líptic (temps de CPU en un DECsystem 5100)

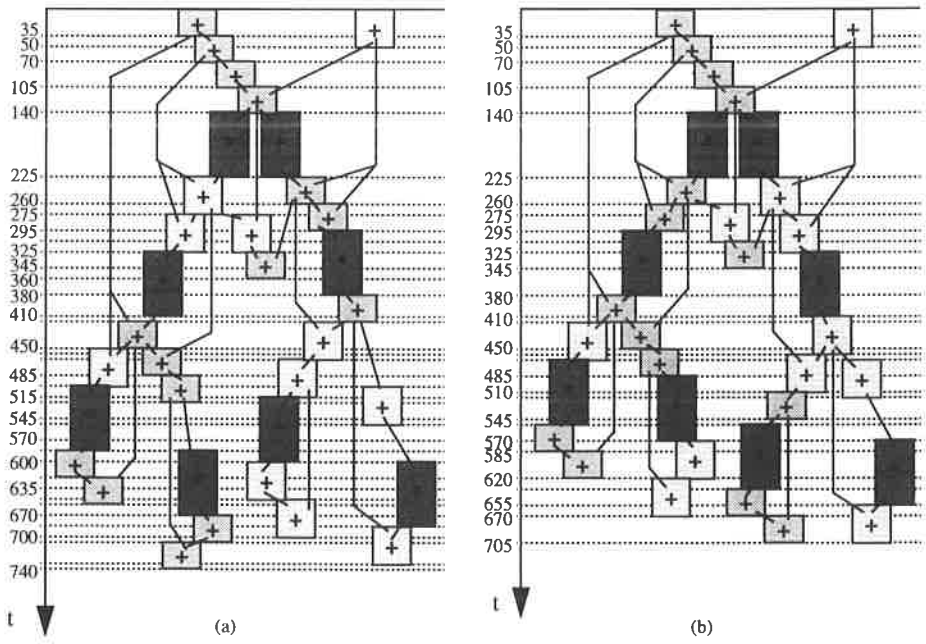


Figura 5.13: (a) Planificació del filtre el·líptic obtinguda per ELS*; (b) Planificació obtinguda per* ELLAS

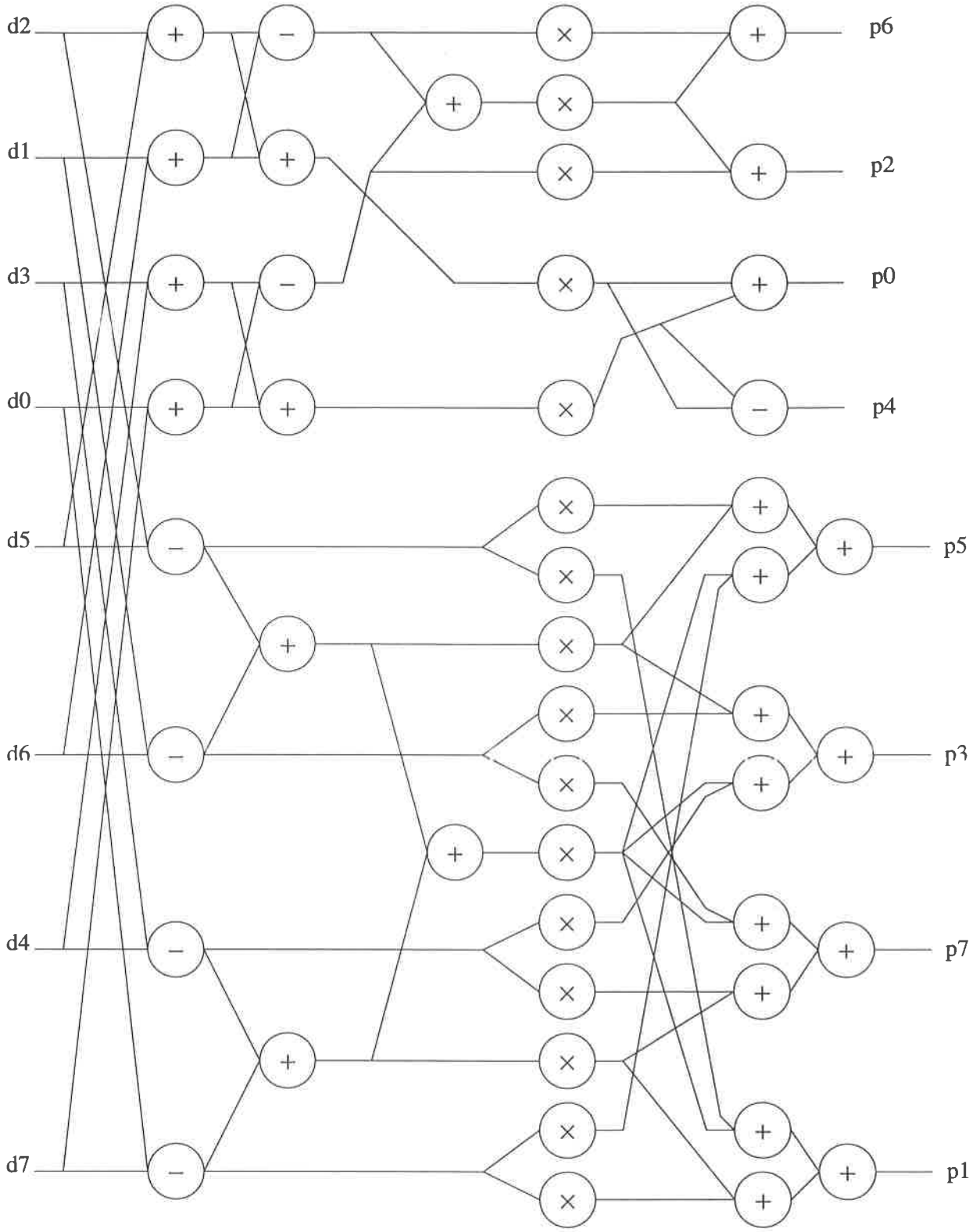


Figura 5.14: Graf de flux de dades de la transformada discreta del cosinus

Unitat Funcional	*retard*		
	+	-	×
$ALU(\diamond)$	20	25	∞
$mult(\otimes)$	∞	∞	45

Taula 5.7: Matriu de retards utilitzada per l'exemple de la transformada discreta del cosinus

Recursos	ELS		ELLAS	
	Planificació	temps CPU	Planificació	temps CPU
2 $\diamond$, 3 $\otimes$	350 ns	0,016 s	350 ns	0,6 s
3 $\diamond$, 3 $\otimes$	315 ns	0,016 s	315 ns	0,63 s
3 $\diamond$, 4 $\otimes$	280 ns	0,016 s	260 ns	0,6 s
4 $\diamond$, 4 $\otimes$	250 ns	0,016 s	240 ns	0,68 s
4 $\diamond$, 3 $\otimes$	315 ns	0,016 s	315 ns	0,6 s
5 $\diamond$, 4 $\otimes$	240 ns	0,016 s	225 ns	0,62 s
5 $\diamond$, 5 $\otimes$	225 ns	0,016 s	225 ns	0,62 s
4 $\diamond$, 5 $\otimes$	225 ns	0,016 s	225 ns	0,65 s
4 $\diamond$, 6 $\otimes$	205 ns	0,016 s	205 ns	0,66 s
4 $\diamond$, 8 $\otimes$	200 ns	0,016 s	195 ns	0,6 s
6 $\diamond$, 6 $\otimes$	185 ns	0,016 s	185 ns	0,61 s
6 $\diamond$, 8 $\otimes$	175 ns	0,016 s	175 ns	0,6 s
8 $\diamond$, 8 $\otimes$	160 ns	0,016 s	155 ns	0,66 s

Taula 5.8: Resultats asíncrons per a la transformada discreta del cosinus (temps de CPU en una SUN SPARCstation 10)

Unitat Funcional	*retard*			
	+	-	×	<
$ALU(\diamond)$	1	1	∞	1
$mult(\otimes)$	∞	∞	2	∞
$sumador(\oplus)$	1	∞	∞	∞
$restador(\ominus)$	∞	1	∞	∞
$comparador(\lhd)$	∞	∞	∞	1

Taula 5.9: Matriu de retards utilitzada per l'exemple de l'equació diferencial

5.5.2 ELS i ELLAS per a planificació síncrona

Atès que no existeixen algorismes de planificació d'operacions per a sistemes asíncrons, en aquesta secció es comparen els resultats dels algorismes *ELS* i *ELLAS* amb altres algorismes de planificació d'operacions per a sistemes síncrons. Perquè la comparació sigui realista, s'han utilitzat els algorismes *ELS* i *ELLAS* per fer planificacions d'operacions per a sistemes síncrons.

Per poder fer aquesta comparació, s'ha utilitzat una matriu de retards on els retards són múltiples de cicle. En aquest cas, els retards dels recursos no són retards estimats, sinó el retard exacte. Així, doncs, els esdeveniments ocorriran en temps múltiples de cicle i obtindrem una planificació síncrona.

A constinuació es presenten els resultats obtinguts per els exemples de l'equació diferencial [PK87] i del filtre el·líptic de cinquè ordre [DDN85].

5.5.2.1 Equació diferencial

A continuació anem a presentar els resultats obtinguts per *ELS* i *ELLAS* en l'exemple de l'equació diferencial si es considera un model d'arquitectura síncrona. En aquest cas s'ha utilitzat una llibreria de recursos composta per una ALU, un multiplicador, un sumador, un restador i un comparador. La matriu de retards es descriu a la taula 5.9. Els retards s'expressen en nombre de cicles.

La taula 5.10 mostra els resultats obtinguts per *ELS* i *ELLAS* per aquest cas. S'han comparat aquests resultats amb els de diferents algorismes de planificació: planificació

Recursos	Planificacions obtingudes (en cicles)					
	Planificació relativa	FDS	ALPS	Fast and Near	ELS	ELLAS
$\otimes \oplus \ominus \lhd$	13	–	–	–	13	13
$\otimes \oplus \oplus \ominus \lhd$	13	–	–	–	13	13
$\otimes \otimes \oplus \ominus \lhd$	7	–	–	–	7	7
$\otimes \otimes \oplus \oplus \ominus \lhd$	7	–	–	–	7	7
$\otimes \otimes \oplus \oplus \ominus \lhd$	7	–	–	–	7	7
$\otimes \otimes \otimes \oplus \oplus \ominus \lhd$	6	–	–	–	6	6
$\otimes \otimes \diamond\diamond$	–	7	7	7	7	7
$\otimes \otimes \otimes \diamond\diamond$	–	6	6	6	6	6

Taula 5.10: Resultats per a l'exemple de l'equació diferencial si es considera un model d'arquitectura síncrona

relativa [KM90], planificació dirigida per forces (FDS) [PK89a], ALPS [LHL89] i l'algorisme presentat a [PK91] (*Fast and Near*).

Atesa la senzillesa de l'exemple, tots els algorismes obtenen els mateixos resultats en tots els casos. Els temps de CPU requerits per aquest exemple es mostra a la taula 5.11.

5.5.2.2 Filtre el·líptic de cinquè ordre

En aquest apartat es presenten els resultats obtinguts per al filtre el·líptic. S'han comparat els algorismes *ELS* i *ELLAS* amb els algorismes de planificació dirigida per forces (FDLS) [PK89a], planificació relativa (PR) [KM90] i *CASCH* [GKR91].

Els recursos utilitzats són un sumador que realitza sumes amb un retard d'un cicle, un multiplicador que realitza multiplicacions en dos cicles i una ALU que realitza sumes en un cicle i multiplicacions en dos cicles.

La taula 5.12 mostra els resultats obtinguts en diferents experiments. L'algorisme *ELS* obté en tots els casos, excepte un, els mateixos resultats que els algorismes *FDLS* i *CASCH*. *ELS* supera els resultats de l'algorisme de planificació relativa en alguns casos.

Es pot observar com *ELLAS* obté, en els casos que s'han comparat, els mateixos resultats que *FDLS*. Cal destacar que l'algorisme *FDLS* està considerat com un dels millors

Algorisme	Temps de CPU	Màquina
Planificació relativa	–	–
FDS	15–35 s	Xerox 1108 LISP machine
ALPS	0,17–0,28 s	VAX-11/8800
Fast and Near	0,016-0,033 s	SUN4/280
ELS	0,01 s	SUN SPARCstation 10
ELLAS	0,01–0,016 s	SUN SPARCstation 10

Taula 5.12: Temps de CPU requerits pels diferents algorismes a l'exemple de l'equació diferencial

algorismes de planificació atesa la seva complexitat. Els resultats de *ELLAS* són millors en alguns casos que els obtinguts per *CASCH* i per l'algorisme de planificació relativa, i mai són pitjors.

Els temps de CPU requerits per cadascun dels algorismes es mostra a la taula 5.13.

5.6 Millores en el model d'execució d'ELS/ELLAS

Una de les limitacions dels algorismes de llistes d'esdeveniments rau en el model d'execució utilitzat. Aquest model implica que les dades d'entrada es llegeixen de registres i que les dades de sortida també s'emmagatzemen en registres. Així, en executar una operació, es considera el retard de la porta per exectutar l'operació i el retard d'un registre per guardar el resultat. A més d'aquests retards, també s'inclou el retard necessari per inicialitzar la porta (vegeu figura 5.15).

Aquest model, tot i que és correcte, és millorable, sobretot si tenim en compte que la fase d'inicialització d'una porta es pot solapar amb la fase de càlcul si la porta no és la mateixa. Aquesta millora s'ha afegit a *ELLAS*, de manera que ara les fases d'inicialització se solapen amb les fases d'execució de les operacions tal com mostra la figura 5.16. És evident que per a sistemes síncrons aquesta modificació no té sentit.

L'algorisme bàsicament és el mateix, el que canvia és el model d'execució utilitzat. El resultat de les operacions es passa a les operacions successores un cop el resultat ja s'ha

Recursos (retard en cicles)			Planificacions obtingudes (en cicles)				
Sumador (1 cicle)	Mul (2 cicles)	ALU (+, *) (1 cicle, 2 cicles)	CASCH	FDLS	PR	ELS	ELLAS
1	1		-	-	28	28	28
1	2		28	-	-	28	28
2	2		18	18	19	19	18
3	2		18	18	18	18	18
		2	23	-	–	23	23
1		2	19	-	–	19	18
2		2	18	-	–	18	18
3	1		-	-	23	21	21
3	3		-	17	17	17	17
2	1		-	21	–	21	21

Taula 5.12: Resultats per a l'exemple del filtre el·líptic si es considera un model d'arquitectura síncrona

Algorisme	Temps de CPU	Màquina
Planificació relativa	–	–
FDLS	1–2 min	Xerox 1108 LISP machine
CASCH	1,3–3,6 s	HP/Apollo DN 4500
ELS	0,01 s	SUN SPARCstation 10
ELLAS	0,06–0,08 s	SUN SPARCstation 10

Taula 5.13: Temps de CPU requerits pels diferents algorismes a l'exemple de l'equació diferencial

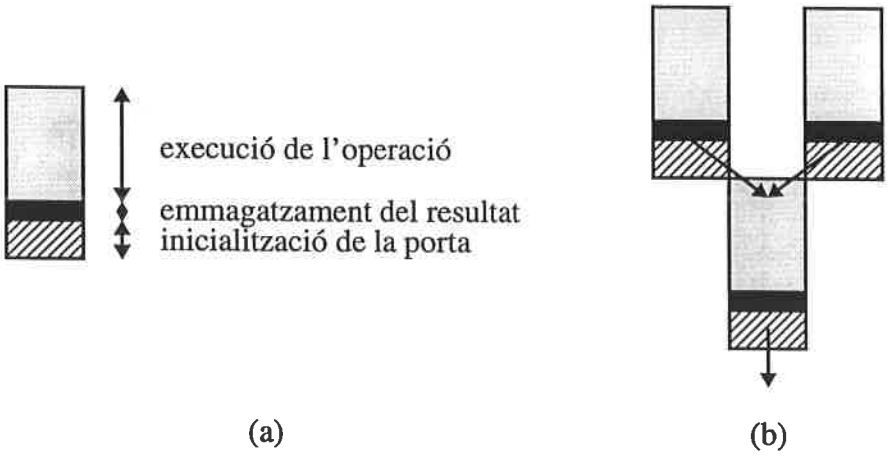


Figura 5.15: (a) Model d'execució dels algorismes ELS *i* ELLAS*; (b) Seqüenciament de les operacions*

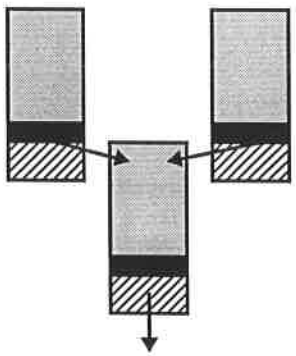


Figura 5.16: Seqüenciament de les operacions a la millora

Recursos	ELLAS		ELLAS modificat temps inicialització 5 ns		ELLAS modificat temps inicialització 10 ns	
	Planificació	CPU	Planificació	CPU	Planificació	CPU
$\otimes \otimes \otimes \oplus \diamond$	270 ns	0,01 s	265	0,017 s	260	0,017 s
$\otimes\otimes \oplus \diamond$	305 ns	0,01 s	300	0,01 s	295	0,017 s
$\otimes \oplus \diamond$	545 ns	0,01 s	540	0,017 s	535	0,01 s

Taula 5.14: Resultats per l'equació diferencial (temps de CPU en una SUN SPARCstation 10)

Recursos	ELLAS		ELLAS modificat temps inicialització 5 ns		ELLAS modificat temps inicialització 10 ns	
	Planificació	CPU	Planificació	CPU	Planificació	CPU
$\otimes\otimes \oplus$	750 ns	0,083 s	805	0,15 s	790	0,13 s
$\otimes\otimes \oplus \diamond$	750 ns	0,116 s	745	0,15 s	740	0,12 s
$\otimes \otimes \otimes \oplus\oplus$	600 ns	0,083 s	575	0,12 s	560	0,13 s
$\otimes \otimes \otimes\otimes\oplus\oplus$	480 ns	0,083 s	460	0,13 s	450	0,15 s
$\otimes \otimes \otimes \otimes \otimes\oplus\oplus$	430 ns	0,099 s	415	0,17 s	400	0,15 s

Taula 5.15: Resultats pel filtre AR-lattice (temps de CPU en una SUN SPARCstation 10)

emmagatzemat, sense esperar a que l'unitat funcional s'hagi inicialitzat. Únicament cal esperar a que l'unitat funcional s'inicialitzi si no hi ha cap recurs lliure.

D'aquesta manera se solapa l'execució de les operacions amb la fase d'inicialització de les unitats, sempre i quan hi hagin suficients recursos lliures. A continuació es mostren els resultats obtinguts amb el nou programa i es comparen amb els obtinguts per *ELLAS*. Les matrius de retards utilitzades són les mateixes de la secció de resultats.

La taula 5.14 mostra els resultats per a l'equació diferencial. En un cas es fixa el temps d'inicialització a 5 ns i en l'altra a 10 ns. Aquest temps d'inicialització se suposa igual per tots els recursos.

La taula 5.15 mostra els resultats obtinguts per al filtre AR-lattice i la 5.16 els del filtre el·líptic de cinquè ordre. Podem observar que per gairebé tots els casos els temps

Recursos	ELLAS		ELLAS modificat temps inicialització 5 ns		ELLAS modificat temps inicialització 10 ns	
	Planificació	CPU	Planificació	CPU	Planificació	CPU
⊗⊗ ⊕ ◇	755 ns	0,07 s	710	0,1 s	680	0,18 s
⊗⊗ ⊕⊕ ◇	690 ns	0,07 s	640	0,1 s	590	0,12 s
⊗ ⊗ ⊗ ⊕ ◇◇	700 ns	0,08 s	660	0,12 s	625	0,12 s
⊗ ⊗ ⊗⊕⊕ ◇	670 ns	0,07 s	620	0,12 s	575	0,12 s
⊗ ⊗ ⊗⊗⊕◇◇◇	690 ns	0,08 s	655	0,1 s	615	0,12 s

Taula 5.16: Resultats pel filtre el·líptic de cinquè ordre (temps de CPU en una SUN SPARCstation 10)

de planificació es veu reduït. Aquest temps es guanya de solapar les inicialitzacions dels recursos amb l'execució de les operacions si hi ha suficients recursos lliures.

5.7 Conclusions

Aquest capítol presenta una primera proposta a la planificació d'operacions sota restricció de recursos en un sistema de síntesi d'alt nivell de circuits asíncrons. La planificació d'operacions en un model asíncron significa definir un ordre d'execució de les operacions. Establir aquest ordre consisteix en seqüencialitzar el DFG segons el nombre de recursos que fixin les restriccions.

S'han definit estructures i funcions bàsiques per a la gestió de llistes d'esdeveniments. S'han presentat dos algorismes: *ELS* i *ELLAS*. Aquests algorismes prenen com a punt de partida un model d'execució en el que les dades d'entrada de les operacions estan en registres i els resultats s'emmagatzemen en registres.

Per fer estimacions del temps de planificació, s'ha utilitzat el temps mitjà d'execució de les operacions. Aquesta estimació no és del tot exacta, però és vàlida pels nostres propòsits. Si es volguessin fer aplicacions en temps real s'hauria de considerar el retard més gran de les unitats funcionals per executar les operacions.

Atès que no existeixen algorismes de planificació d'operacions per sistemes asíncrons, s'han comparat els algorismes *ELS* i *ELLAS* amb altres algorismes de planificació per

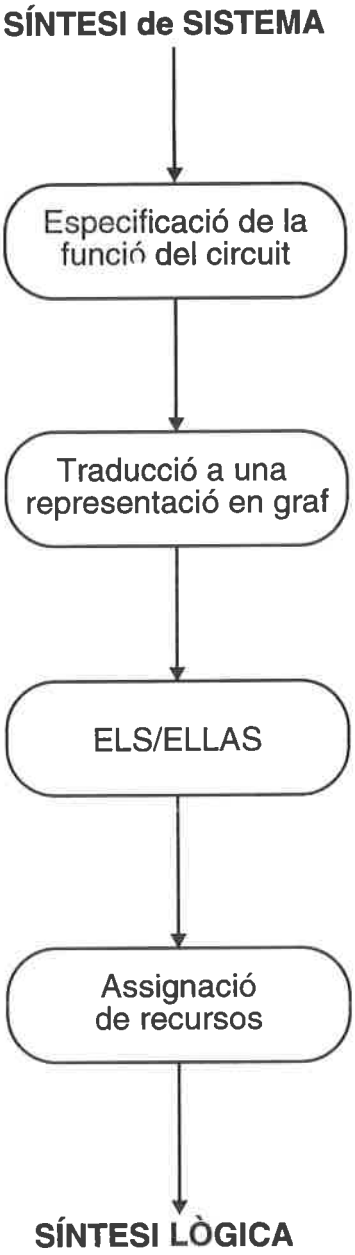


Figura 5.17: ELS/ELLAS *en un sistema de síntesi d'alt nivell*

sistemes síncrons. Per fer-ho, s'han fixat els retards dels recursos a múltiples de cicle, de manera que s'obté una planificació síncrona. Els resultats obtinguts mostren que *ELLAS* és un algorisme de planificació comparable als millors algorismes de planificació per sistemes síncrons.

La figura 5.17 mostra on estaríen situats els algorismes de planificació per llistes en un sistema d'alt nivell per sistemes asíncrons. En aquest cas, la planificació d'operacions es realitza després de la traducció i abans de l'assignació de recursos.

Una característica que limita els algorismes *ELS* i *ELLAS* és que no es permet l'encadenament de mòduls. S'ha proposat una millora que redueix l'efecte d'aquest inconvenient. A la millora es permet el solapament de la fase d'inicialització dels recursos amb la d'avaluació si hi ha suficients recursos disponibles. Amb aquesta millora els temps de planificació es redueixen de l'ordre d'un 5%, depenent del nombre de recursos que s'utilitzin i de l'estructura del graf de flux de dades. En el capítol 7 es planteja un nou model d'execució en el que les operacions es poden encadenar.

Fins al moment només s'ha estudiat la planificació de blocs bàsics. En un futur es planteja l'extensió d'aquests algorismes a paral·lelització de bucles.

Capítol 6

GLASS: Associació de Recursos

Com hem vist al capítol 2, l'assignació de recursos és una de les dues tasques més importants dins de la síntesi d'alt nivell. Fins ara, els algorismes d'assignació de recursos han estat orientats a arquitectures síncrones. GLASS[1] *és un algorisme d'assignació de recursos no basat en el temps de cicle, característica que el fa apte per a sistemes síncrons i asíncrons. Aquest algorisme realitza totes les tasques d'associació de manera simultània, buscant un compromís entre el nombre d'unitats funcionals, registres i unitats d'interconnexió utilitzades.*

[1] De l'anglès, GLobal ASSignment

6.1 Introducció

En el capítol 2 s'han vist les diferents tendències existents dins de l'associació de recursos. El principal objectiu de l'associació de recursos és minimitzar l'àrea del camí de dades. En alguns casos, l'àrea total del circuit pot estar clarament dominada per l'interconnexió. Per exemple, en els circuits asíncrons, l'àrea d'interconnexió és més crítica si es codifiquen les dades en *dual-rail*. Com veurem més endavant amb un exemple, en alguns casos es poden obtenir dissenys amb menys cost d'àrea utilitzant més unitats funcionals ja que l'àrea d'interconnexió es veu reduïda de manera significativa.

Com s'ha vist al capítol 2, els algorismes d'associació de recursos poden ser classificats en tres grups:

- Algorismes orientats al cicle: les diferents subtasques d'associació són realitzades de manera simultània per cada cicle.
- Algorismes globals: les tres subtasques d'associació s'executen de manera separada, però de manera global respecte a la planificació.
- Algorismes de millora iterativa: les tres subtasques s'executen de manera simultània a partir d'una configuració inicial que és millorada de forma iterativa mitjançant la succesiva reassociació dels diferents mòduls.

D'entre els algorismes globals, cal destacar els basats en la teoria de grafs. Fins ara, totes les propostes d'associació de recursos basades en la teoria de grafs presentaven algorismes on les diferents tasques d'associació (associació d'operacions a unitats funcionals, associació de variables a unitats de memòria i associació de transferències de dades a unitats de transferència) són executades una després de l'altra. Normalment, utilitzen la mateixa formulació per a les tres tasques, però cadascuna es realitza de manera independent. L'inconvenient d'aquestes propostes és que en executar les diferents tasques separadament, quan es prenen decisions no es té en compte quina repercussió tindrà cada decisió en els altres tipus de recursos. Per exemple, una decisió d'agrupar dues operacions, pot ser que redueixi el nombre d'unitats funcionals, però també que provoqui un increment important en l'àrea a causa de la interconnexió.

	#mult	#sumador	#registres	#mux 2:1	#bus	#entrades bus	àrea
Solució 1	1	2	11	20	7	28	821
Solució 2	1	3	12	9	4	17	743

Taula 6.1: Resultats pel filtre el·líptic [DDN85]

En aquest capítol es presenta una nova proposta per fer associació de recursos, *GLASS*, on totes les tasques d'associació són executades simultàniament, de manera que es té en compte el cost global en àrea de tots els recursos. L'algorisme resol el problema de particionat en *cliques* d'un *graf de compatibilitat global* en què els vèrtexs representen operacions, variables o transferències de dades. Atès que el problema de particionat en *cliques* és NP-hard [GJ79], cal buscar heurístiques que ens permetin solucionar el problema per grafs grans en un temps de CPU acceptable. *GLASS* utilitza l'heurística proposada per Tseng [TS86] que permet solucionar el problema de particionat en *cliques* en un temps de CPU polinòmic.

La principal aportació d'aquest treball és la representació de les operacions, variables i transferències de dades com a vèrtexs d'un mateix graf de compatibilitat, de manera que les diferents decisions són entrellaçades, amb l'objectiu d'obtenir un cost mínim d'àrea global.

En tenir un graf de compatibilitat global, es poden resoldre les tres tasques d'associació simultàniament. A diferència de les propostes anteriors, decisions, tals com fusionar dues operacions perquè comparteixin una mateixa unitat funcional o fusionar dues variables en un únic registre, poden ser entrellaçades i poden ocórrer en qualsevol moment. D'aquesta manera, s'evitaran decisions que, per exemple, haurien donat bons resultats en l'associació d'unitats funcionals però que haurien incrementat molt el cost d'interconnexió.

Per il·lustrar la validesa d'aquesta proposta, presentem un exemple senzill (vegeu taula 6.1). La primera solució s'ha obtingut restringint *GLASS* de tal manera que primer s'associen operacions a unitats funcionals, després variables a registres i finalment, transferències de dades a busos. La segona solució s'ha obtingut deixant que *GLASS* entrellaci les diferents decisions d'associació. Tot i que la primera solució és millor en termes d'unitats

funcionals i de registres, és pitjor que la segona en termes d'unitats d'interconnexió. Si considerem totes les unitats, el resultat obtingut a la segona solució és millor. Així ho veiem a l'estimació d'àrea. Per fer aquest càlcul de l'àrea total, s'han utilitzat estimacions de l'àrea ocupada per cada tipus d'unitat funcional en una llibreria construïda amb el conjunt d'eines OCEAN [GS93].

6.2 Graf de compatibilitat global

6.2.1 Definicions

L'algorisme d'associació de recursos que es proposa, suposa que el punt de partida és un $SDFG = (V, A)$ on s'han planificat les operacions i s'ha seleccionat el tipus d'unitat funcional on s'executaran les operacions. Per a cada vèrtex $v \in V$ farem servir la terminologia següent:

v_t : tipus de vèrtex: operació (`OP`), variable (`VAR`), o transferència de dades (`TD`)

v_o : operació representada pel vèrtex v

v_f : tipus d'unitat funcional seleccionada per executar l'operació v_o

$pred(v)$ = $(pred_1(v), \cdots, pred_n(v))$ és la llista de predecessors del vèrtex $v(\mid pred(v) \mid = n)$

$succ(v)$ = llista de successors del vèrtex v

Suposarem que els elements de $pred(v)$ (operands d'entrada de v) són ordenats i $pred_i(v)$ indicarà el predecessor i-èssim de v ($i = 1..|pred(v)|$). Si una operació és commutativa, l'ordre dels predecessors pot ser intercanviat. És evident que, en el cas de variables i transferències de dades, $|pred(v)| = 1$.

El graf de compatibilitat que es construeix, $GC = (\hat{V}, \hat{A})$, inicialment té el mateix nombre de vèrtexs que el *SDFG*. Existeix un arc $\hat{e} \in \hat{A}$ entre dos vèrtexs $\hat{u}, \hat{v} \in \hat{V}$ si:

- Les dues operacions executades en els vèrtexs són compatibles.
- Han estat assignades al mateix tipus de recurs.

Com ja hem dit al capítol 2, dos vèrtexs són compatibles si els temps de vida de les operacions que representen no se solapen. El mateix es pot aplicar a les variables i a les transferències de dades.

Definició 6.1 Compatibilitat

Donats dos vèrtexs $u, v \in V$ *definirem* $\mathrm{comp}(u, v)$ *com:*

$$\mathrm{comp}(u,v) = \begin{cases} \text{cert} & \textit{si } u \textit{ i } v \textit{ són compatibles} \\ \text{fals} & \textit{altrament} \end{cases} \tag{6.1}$$

La figura 6.1.b mostra un exemple de graf de compatibilitat per al *SDFG* de la figura 6.1.a. Es pot apreciar que el graf de compatibilitat està format per diferents subgrafs disconnexes, on cadascun d'ells correspon a un tipus de recurs: multiplicador, sumador, restador, comparador i registre. És evident que si dos vèrtexs han estat assignats a tipus de recursos diferents, encara que el seu temps de vida no se solapi, no poden compartir un mateix recurs. Per aquest motiu no hi haurà cap arc entre ells en el *GC*.

En el graf de la figura 6.1 podem apreciar que no existeixen vèrtexs representant les transferències de dades. Això es degut al fet que s'ha optat per un model d'arquitectura orientat al multiplexor. Si s'hagués triat un model d'arquitectura orientat al bus, tindríem un subgraf més al *GC*. En aquest subgraf els vèrtexs representarien transferències de dades i els arcs entre ells la seva relació de compatibilitat per compartir un mateix bus.

Els arcs del *GC* són etiquetats amb un pes. La funció de pes $W : \hat{A} \rightarrow \mathbb{R}$ es defineix per cada arc $\hat{e} = (\hat{u}, \hat{v}) \in \hat{A}$ del *GC* segons l'afinitat d'interconnexió dels vèrtexs $\hat{u}, \hat{v}$. L'afinitat d'interconnexió de dos vèrtexs representa el guany en cost d'interconnexió quan els dos vèrtexs comparteixen un mateix recurs.

Durant l'execució de l'algorisme de particionat en *cliques*, parells de vèrtexs del *GC* connectats per un arc són *fusionats* en un únic vèrtex. És a dir, el *GC* és modificat dinàmicament de manera que cada vèrtex del *GC* representa un conjunt de vèrtexs del *SDFG*.

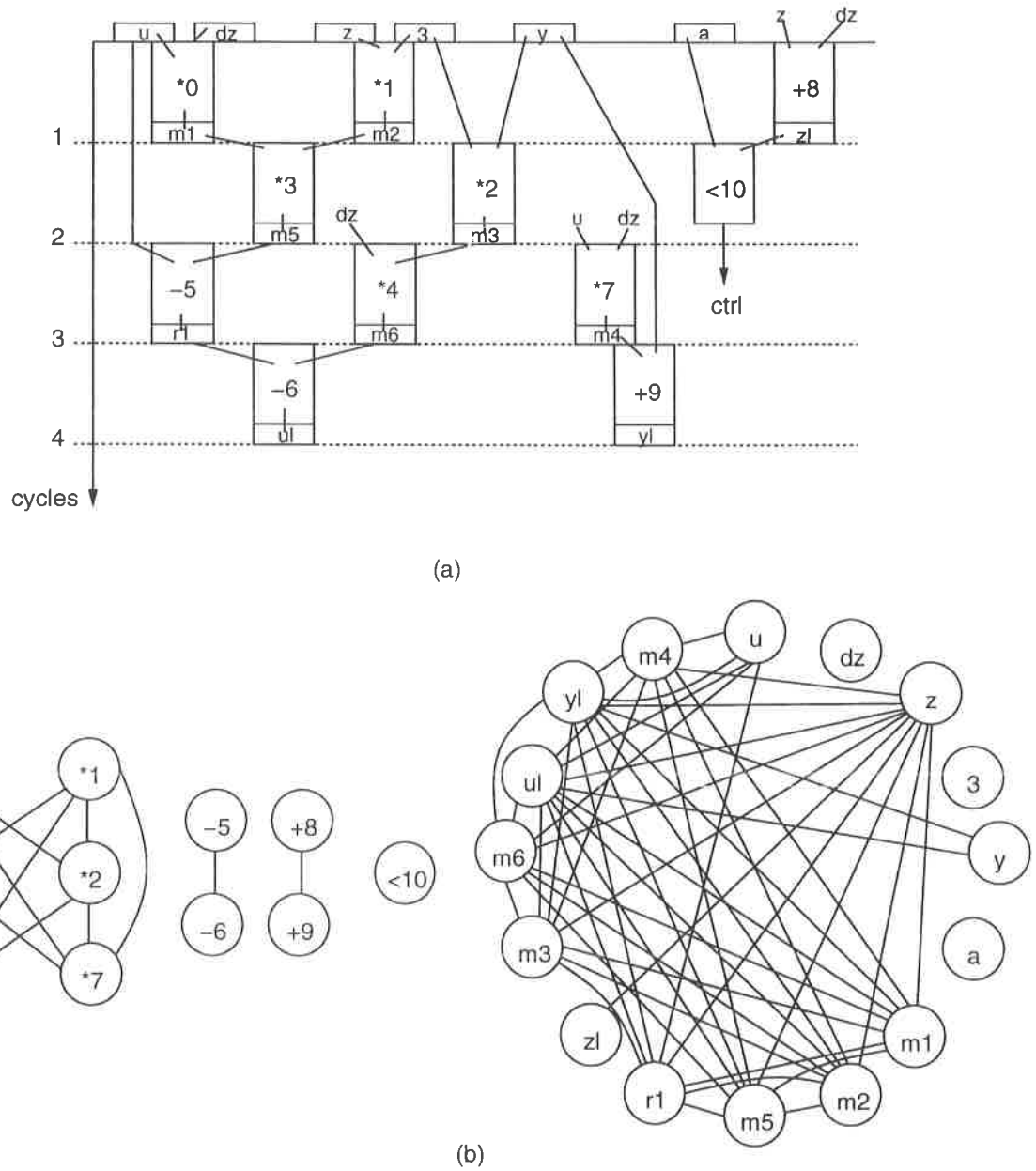


Figura 6.1: (a) Graf de flux de dades planificat; (b) Graf de compatibilitat global pel SDFG *anterior*

Definició 6.2 Conjunt de vèrtexs fusionats

El conjunt de vèrtexs fusionats $\mathrm{CVF}(\hat{v})$ *és el conjunt* $\{v_1, \ldots, v_n\}$, $v_i \in V$ *que han estat fusionats en el vèrtex* $\hat{v} \in \hat{V}$. *Com ja hem dit, el* GC *inicial té tants vèrtexs com el* SDFG *i, per tant, inicialment* $\mathrm{CVF}(\hat{v}) = v$.

Definició 6.3 Fusió

La funció fusionat *es defineix tal com segueix:*

$$\mathrm{fusionat}(u, v) = \begin{cases} \text{cert} & si\ \exists \hat{w} \in \hat{V}\ tal\ que\ u, v \in \mathrm{CVF}(\hat{w}) \\ \text{fals} & altrament \end{cases} \tag{6.2}$$

Es pot deduir fàcilment que si $u = v$ aleshores $fusionat(u, v) = cert$. Si dos vèrtexs $\hat{u}, \hat{v}$ es fusionen en el vèrtex $\hat{w}$, aleshores el *conjunt de vèrtexs fusionats* del vèrtex resultant és la unió dels *conjunts de vèrtexs fusionats* dels vèrtexs inicials, és a dir:

$$CVF(\hat{w}) = CVF(\hat{u}) \cup CVF(\hat{v}) \tag{6.3}$$

Un característica dels vèrtexs del *SDFG* que volem mantenir accessible és la propietat commutativa de les operacions.

Definició 6.4 Propietat commutativa

La funció $\mathrm{comm}(u), u \in V$ *avalua* cert *si* u_o *és una operació commutativa i* fals *altrament.*

Definició 6.5 Propietat commutativa global

Definirem la funció $\mathrm{GCcomm}(\hat{u}), \hat{u} \in \hat{V}$*:*

$$\mathrm{GCcomm}(\hat{u}) = \begin{cases} \text{cert} & si\ \forall u \in \mathrm{CVF}(\hat{u}),\quad \mathrm{comm}(u) = \text{cert} \\ \text{fals} & altrament \end{cases} \tag{6.4}$$

És a dir, un vèrtex $\hat{u} \in \hat{V}$ té la propietat commutativa global si tots els vèrtexs $u \in CVF(\hat{u})$ representen operacions commutatives.

fusionar$(\hat{u},\hat{v})\{$
  afegir vèrtex $\hat{w}$ *a* $\hat{V}$;
  per a cada $\hat{z} \in \hat{V}$ t. q. $(\hat{u},\hat{z}),(\hat{v},\hat{z}) \in \hat{A}$ **fer**
    afegir arc $(\hat{w},\hat{z})$;
  per a cada $(\hat{u},\hat{z}) \in \hat{A}$ **fer** *eliminar arc* $(\hat{u},\hat{z})$;
  {Això també elimina l'arc $(\hat{u},\hat{v})$ }
  per a cada $(\hat{v},\hat{z}) \in \hat{A}$ **fer** *eliminar arc* $(\hat{v},\hat{z})$;
  eliminar vèrtex $\hat{u}$;
  eliminar vèrtex $\hat{v}$;
}

Algorisme 6.1: Funció fusionar

6.2.2 Fusió de vèrtexs

Com ja s'ha dit anteriorment, l'algorisme de particionat en *cliques* va *fusionant* vèrtexs connectats per un arc en un únic vèrtex. La funció *fusionar*$(\hat{u},\hat{v})$ es descriu a l'algorisme 6.1.

La figura 6.2 mostra un exemple de l'efecte que té la funció *fusionar* sobre el graf de compatibilitat. Els vèrtexs $\hat{u}$ i $\hat{v}$, com que estan connectats per un arc, poden ser *fusionats*. Com a resultat s'obté un nou GC on els vèrtexs $\hat{u}$ i $\hat{v}$ han estat reemplaçats per un nou vèrtex $\hat{w}$. Aquells vèrtexs que en el graf inicial tenien un arc que els connectava amb ambdós vèrtexs $\hat{u}$ i $\hat{v}$, tindran un arc que els connecti amb el nou vèrtex $\hat{w}$. Tots els arcs que unien els vèrtexs $\hat{u}$ i $\hat{v}$ amb el GC seran eliminats, així com els mateixos vèrtexs $\hat{u}$ i $\hat{v}$.

El resultat de *fusionar* dos vèrtexs en un és el fet que les operacions representades per aquests vèrtexs seran executades pel mateix recurs, és a dir, compartiran el mateix recurs. És imprescindible que dos vèrtexs que són *fusionats* siguin compatibles i hagin estat assignats al mateix tipus de recurs. Altrament no podran compartir aquest recurs. Quan no queda cap arc en el GC vol dir que no podem *fusionar* cap més parell de vèrtexs i cada vèrtex $\hat{u} \in \hat{V}$ del graf representa un component del camí de dades. Aquest component és del tipus assignat als vèrtexs $v \in CVF(\hat{u})$. A més, les operacions v_o del conjunt de vèrtexs

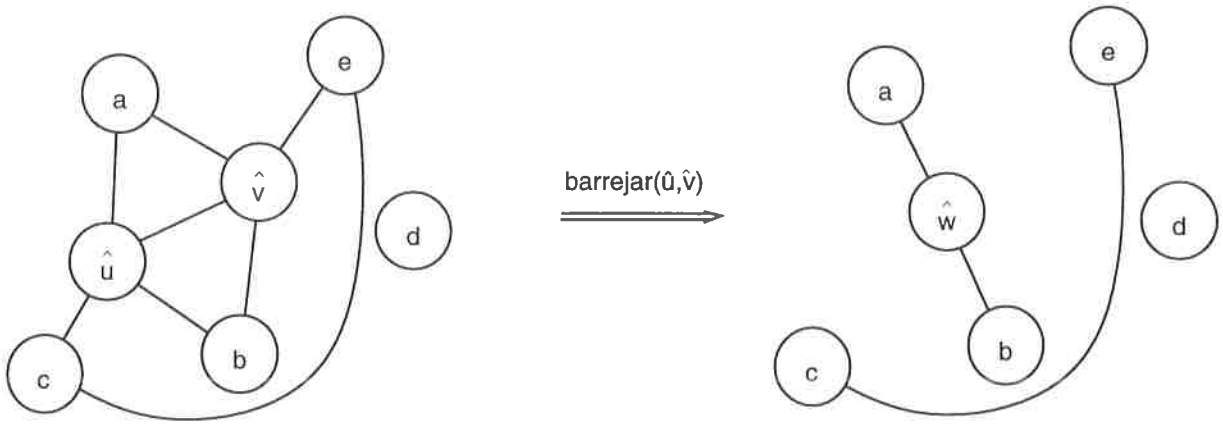


Figura 6.2: Exemple de com fusionar dos vèrtexs

$v \in CVF(\hat{u})$ són les operacions que han estat associades a aquest component.

6.2.3 Pesos dels arcs

Els arcs del GC són etiquetats amb un pes W que representa l'afinitat d'interconnexió dels vèrtexs connectats per aquest arc. Com més gran és el pes d'un arc, més interessant és que els dos vèrtexs connectats per aquest arc siguin fusionats i comparteixin un únic recurs. De la mateixa manera que el GC canvia dinàmicament a mesura que fusionem diferents vèrtexs del graf, també canvien els pesos dels arcs. El pes dels arcs ens servirà per decidir quins vèrtexs cal fusionar per obtenir una configuració amb un cost d'interconnexió menor.

Vegem amb un exemple què volem mesurar amb el pes dels arcs. Imaginem que volem avaluar el pes d'un arc $\hat{e} = (\hat{u}, \hat{v})$. Per simplificar, suposarem que els *conjunts de vèrtexs fusionats* de $\hat{u}$ i $\hat{v}$ tenen un únic element ($CVF(\hat{u}) = \{u\}$ i $CVF(\hat{v}) = \{v\}$). La figura 6.3.a mostra aquesta situació inicial on $u_t = v_t =$ `OP` i les dades d'entrada i sortida de u i v són variables emmagatzemades en registres.

El pitjor cas és quan totes les variables d'entrada i de sortida no tenen cap tipus de relació (no estan fusionades ni són compatibles). Aquest fet implica un increment de l'àrea d'interconnexió si fusionem aquests dos vèrtexs. La figura 6.3.b descriu aquesta situació. S'assigna un pes baix a l'arc ($W(\hat{e})$) per evitar que es fusionin aquests dos vèrtexs si hi ha alternatives millors.

El millor cas possible és quan les variables d'entrada —per exemple x i y— són variables

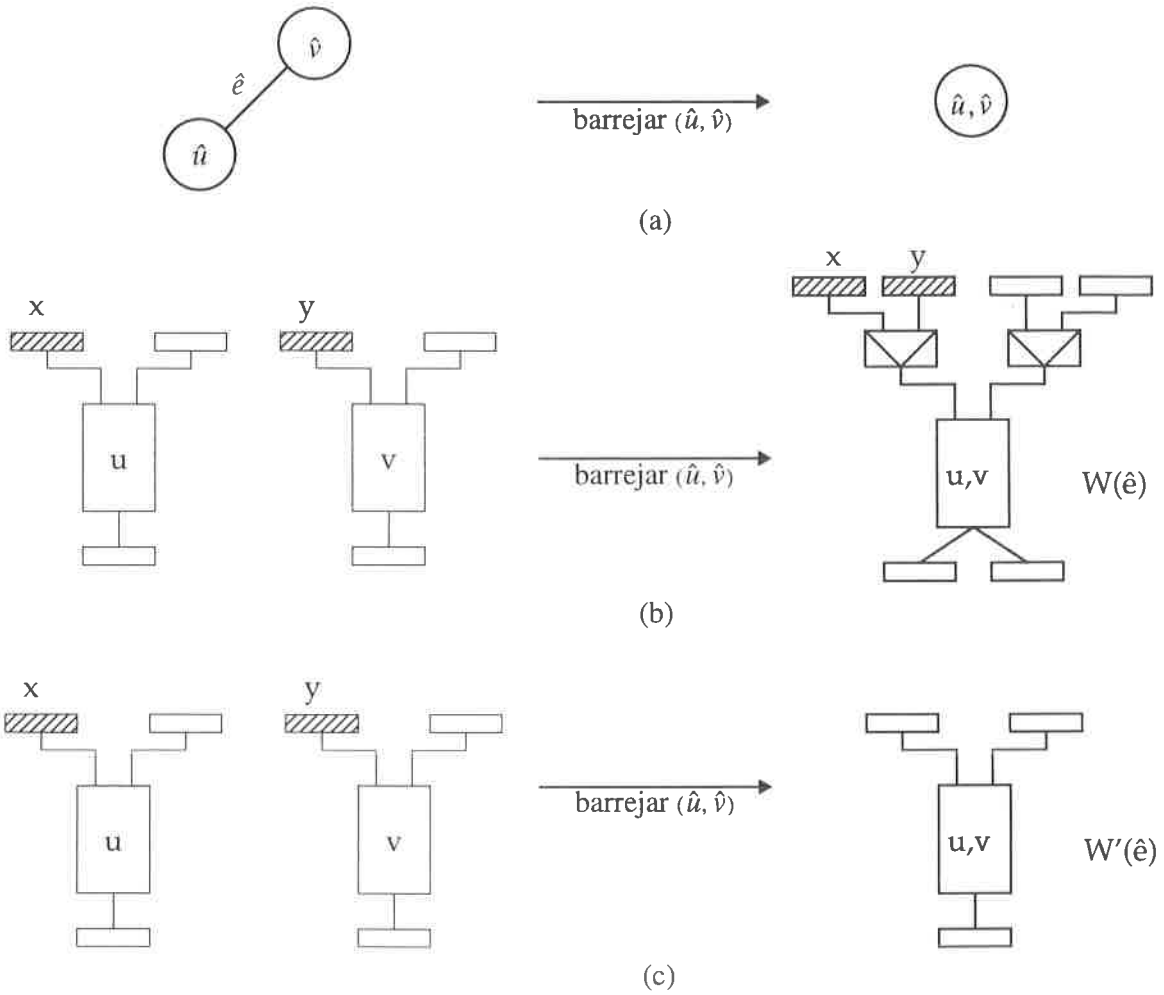


Figura 6.3: (a) Efecte de fusionar els vèrtexs $\hat{u}, \hat{v}$ sobre el GC*; (b) Efecte de fusionar els vèrtexs $\hat{u}, \hat{v}$ sobre el camí de dades si u i v tenen una afinitat d'interconnexió molt petita; (c) Efecte de fusionar els vèrtexs $\hat{u}, \hat{v}$ sobre el camí de dades si u i v tenen una afinitat d'interconnexió molt gran*

fusionades, és a dir, que $fusionat(x, y) = cert$. Així, l'efecte de fusionar els vèrtexs $\hat{u}$ i $\hat{v}$ és favorable, ja que el fet que comparteixin un recurs no incrementa el cost d'interconnexió de la configuració actual. La figura 6.3.c mostra aquesta situació. Atès que l'afinitat d'interconnexió entre $\hat{u}$ i $\hat{v}$ és alta, assignarem un pes ($W'(\hat{e})$) alt a l'arc $\hat{e}$, per afavorir que es fusionin aquests dos vèrtexs.

Un cas intermedi és quan algunes de les entrades i sortides estan fusionades però no totes, o quan algunes de les entrades i sortides són compatibles, ja que és possible que en un futur siguin fusionades.

El pes d'un arc $\hat{e} = (\hat{u}, \hat{v}) \in \hat{A}$, $W(\hat{e})$ es defineix com:

$$W(\hat{e}) = \alpha_t(W_{in}(\hat{e}) + W_{out}(\hat{e})) \tag{6.5}$$

on $W_{in}(\hat{e})$ i $W_{out}(\hat{e})$ són els pesos parcials de $\hat{e}$ respecte a l'afinitat d'interconnexió de les entrades i de les sortides. α_t és un paràmetre del programa que pot prendre diferents valors segons quins tipus de vèrtexs representin $\hat{u}$ i $\hat{v}$ (`OP`, `VAR`, or `TD`). Per exemple α_{OP} és el factor que s'aplica quan l'arc connecta dos vèrtexs on v_t = `OP`.

Si donem un valor alt a α_{OP}, els arcs que connectin vèrtexs que representen operacions tindran un pes més gran i, per tant, seran fusionats abans que altres vèrtexs. Si donem pesos molt diferents a les diferents α_t, podem simular l'execució de les diferents tasques d'associació separadament. Per exemple, si donem un valor molt alt a α_{OP}, un valor mitjà a α_{VAR} i un valor molt petit a α_{TD}, *GLASS* fusionarà primer vèrtexs que representen operacions, després vèrtexs que representen variables i finalment vèrtexs que representen transferències de dades.

Els algorismes 6.2 i 6.3 descriuen com calcular W_{in} i W_{out}.

Suposem que volem calcular el pes d'entrada d'un arc $\hat{e} = (\hat{u}, \hat{v}), \hat{e} \in \hat{A}$, $W_{in}(\hat{e})$.

1. Per a cada parell de vèrtexs $u \in CVF(\hat{u})$, i $v \in CVF(\hat{v})$ mirem la relació entre cada predecessor i-èssim de u i v.

2. Si $pred_i(u)$ i $pred_i(v)$ estan fusionats (pertanyen al mateix *conjunt de vèrtexs fusionats*) se suma el factor de pes W_{f_in} al pes d'entrada total.

3. En cas que $pred_i(u)$ i $pred_i(v)$ siguin compatibles (poden ser fusionats en un futur) se suma el factor de pes W_{c_in} al pes d'entrada total.

$W_{in}(\hat{e})$ {
  { $\hat{e} = (\hat{u}, \hat{v})$ }
  $W = 0$;
  per a cada $u \in CVF(\hat{u})$ **fer**
    per a cada $v \in CVF(\hat{v})$ **fer**
      per $i = 1$ **fins a** $|pred(u)|$ **fer**
        si $fusionat(pred_i(u), pred_i(v))$ **llavors**
          $W := W + \frac{W_{f\_in}}{|pred(u)|}$;
        sinó si $comp(pred_i(u), pred_i(v))$ **llavors**
          $W := W + \frac{W_{c\_in}}{|pred(u)|}$;
      fiper
    fiper
  fiper
  $W := \frac{W}{|CVF(\hat{u})| \cdot |CVF(\hat{v})|}$;
  retorna(W);
}

Algorisme 6.2: Càlcul de W_{in}

4. El pes es pondera segons el nombre de predecessors dels vèrtexs u i v i segons la cardinalitat dels conjunts $CVF(\hat{u})$ i $CVF(\hat{v})$.

Els factors de pes W_{f_in} i W_{c_in} són els pesos parcials que se sumen quan el predecessor (entrada) de $u \in CVF(\hat{u})$ i el predecessor de $v \in CVF(\hat{v})$ són vèrtexs fusionats o compatibles respectivament.

- El fet que els predecessors de u i v siguin **fusionats** vol dir que no caldrà cap unitat d'interconnexió extra per aquesta entrada, ja que són entrades que provenen del mateix recurs (si han estat fusionats en un únic vèrtex vol dir que compartiran el mateix recurs).

- Si els predecessors són **compatibles** vol dir que es possible que siguin fusionats en algun moment de l'execució de l'algorisme. Si aquesta *fusió* es produís, tampoc no caldria cap unitat d'interconnexió extra per a aquesta entrada.

És evident que és més favorable el cas en què ja estan fusionats que quan són compatibles, ja que no tenim la seguretat que siguin fusionats, per tant, el factor de pes W_{f_in} serà més gran o igual que el factor de pes W_{c_in}. En el nostre cas els valors d'aquests factors han estat $W_{f\_in} = 3$ i $W_{c\_in} = 1$.

Si donem valors més grans a W_{f_in}, l'algorisme es torna més ambiciós. Si s'assigna el mateix valor a tots dos factors, la informació continguda en el GC no s'utilitza totalment, ja que es dóna el mateix valor al fet que dos vèrtexs estiguin fusionats que al fet que siguin compatibles.

El càlcul de W_{out} és equivalent al de W_{in}:

1. Per a cada parell de vèrtexs $u \in CVF(\hat{u})$ i $v \in CVF(\hat{v})$ es mira la relació entre els successors. Mentre que el nombre de predecessors dels vèrtexs u i v és el mateix ($|pred(u)| = |pred(v)|$), el nombre de successors pot ser diferent. Pensem, per exemple, en el cas de les variables a emmagatzemar en registres. El nombre de predecessors és sempre 1, però el nombre de successors depèn del nombre d'operacions que utilitzin aquella variable i, per tant, pot ser diferent en cada cas. Així, mentre que abans miràvem només la relació entre els predecessors i-èssims, ara mirarem la relació que hi ha entre cada parell $u' \in succ(u)$ i $v' \in succ(v)$.

$W_{out}(\hat{e})$ {
  { $\hat{e} = (\hat{u}, \hat{v})$ }
  $W = 0$;
  per a cada $u \in CVF(\hat{u})$ **fer**
    per a cada $v \in CVF(\hat{v})$ **fer**
      $temp := 0$;
      per a cada $u' \in succ(u)$ **fer**
        per a cada $v' \in succ(v)$ **fer**
          si $fusionat(u', v')$ **llavors**
            $temp := temp + W_{f\_out}$;
          sinó si $comp(u', v')$ **llavors**
            $temp := temp + W_{c\_out}$;
        fiper
      fiper
      $W := W + \frac{temp}{|succ(u)| \cdot |succ(v)|}$;
    fiper
  fiper
  $W := \frac{W}{|CVF(\hat{u})| \cdot |CVF(\hat{v})|}$;
  retorna(W);
}

Algorisme 6.3: Càlcul de W_{out}

2. En el cas que u' i v' siguin fusionats, se suma el factor de pes W_{f_out} al pes de sortida total.

3. En cas que u' i v' siguin compatibles, se suma el factor de pes W_{c_out}.

4. El pes es pondera segons el nombre de successors dels vèrtexs i segons la cardinalitat dels conjunts de vèrtexs fusionats dels vèrtexs $\hat{u}$ i $\hat{v}$.

Si u' i v' són fusionats, vol dir que en fusionar els vèrtexs $\hat{u}$ i $\hat{v}$ n'hi haurà prou amb una connexió per a aquesta sortida. En cas que siguin compatibles, és possible que només calgui una sortida. W_{f_out} i W_{c_out} són els mateixos factors de pes que W_{f_in} i W_{c_in} aplicats a les sortides. Hem donat a aquests factors els valors $W_{f\_out} = 2$ i $W_{c\_out} = 1$.

6.2.4 Propietat commutativa dels operands

L'algorisme explota la propietat commutativa que tenen algunes operacions com la suma o la multiplicació. En cas que estiguem avaluant el pes d'un arc $\hat{e} = (\hat{u}, \hat{v})$ i algun dels dos vèrtexs $\hat{u}$ o $\hat{v}$ tingui la propietat $GCcomm$, s'aplica el següent procés per calcular el pes de l'arc.

1. Es calcula el pes de l'arc utilitzant l'algorisme explicat a l'apartat anterior.

2. Es tria un dels dos vèrtexs $\hat{u}$ o $\hat{v}$ que tingui la propietat $GCcomm$, per exemple, $\hat{u}$.

3. Per a tots els vèrtexs $u \in CVF(\hat{u})$ es procedeix a intercanviar els seus operands. Per exemple, si es tracta d'operacions binàries, com la suma de dos números, s'intercanvia $pred_1(u)$ per $pred_2(u)$.

4. Es torna a calcular el pes de l'arc.

5. Se selecciona el més gran d'ambdós pesos i es guarda informació sobre quina de les dues configuracions (sense intercanviar operands o intercanviant-ne) té el pes màxim ($\hat{e}_{girat}$).

Aquest procés es descriu a l'algorisme 6.4.

$W(\hat{e})$ {
  { $\hat{e} = (\hat{u}, \hat{v})$ }
  $w1 = \alpha_t(W_{in}(\hat{e}) + W_{out}(\hat{e}))$;
  $w2 = 0$;
  si $GCcomm(\hat{u}) = cert$ **llavors**
    per a cada $u \in CVF(\hat{u})$**fer** *intercanviar operands(u);*
    $w2 = \alpha_t(W_{in}(\hat{e}) + W_{out}(\hat{e}))$;
  sino si $GCcomm(\hat{v}) = cert$ **llavors**
    per a cada $v \in CVF(\hat{v})$ **fer** *intercanviar operands(v);*
    $w2 = \alpha_t(W_{in}(\hat{e}) + W_{out}(\hat{e}))$;
  fisi
  si $w2 > w1$ **llavors**
    $\hat{e}_{girat} = cert$;
    retorna(w2);
  sino
    $\hat{e}_{girat} = fals$
    retorna(w1);
  fisi
}

Algorisme 6.4: Càlcul del pes dels arcs en funció de la propietat commutativa de les operacions

6.3 Algorisme

Donat un graf de compatibilitats GC, resoldre el problema d'associació de recursos consisteix a trobar un particionat en *cliques* del graf. A [PK89b] es proposa una extensió de l'algorisme de particionat en *cliques* presentat a [TS86], on s'assigna un pes als arcs per minimitzar el cost en àrea. L'algorisme selecciona parells de vèrtexs connectats per un arc, segons una funció de pes dels arcs i de les funcions *nombre de veïns comuns* i el *nombre d'arcs a esborrar.*

Al capítol 2 hem definit la funció *nombre de veïns comuns* d'un arc $\hat{e} = (\hat{v}, \hat{w}), \hat{e} \in \hat{A}$ com el nombre de vèrtexs del GC que estan connectat a ambdós vèrtexs que connecta l'arc.

$$NVC(\hat{e}) =\mid \{\hat{v}' \in \hat{V} : (\hat{v}', \hat{v}) \in \hat{A} \wedge (\hat{v}', \hat{w}) \in \hat{A}\} \mid$$

Ara definim el *nombre d'arcs a esborrar* d'un arc $\hat{e} = (\hat{v}, \hat{w}), \hat{e} \in \hat{A}$ quan els vèrtexs de l'arc són fusionats en un únic vèrtex com:

$$NAE(\hat{e}) =\mid \{(\hat{v}, \hat{v}') \in \hat{A} : (\hat{w}, \hat{v}') \notin \hat{A}\} \cup \{(\hat{w}, \hat{v}') \in \hat{A}\} \mid$$

El *conjunt d'arcs a esborrar* és format per aquells arcs que s'hauran d'eliminar de graf de compatibilitat si fusionem els vèrtexs $\hat{v}$ i $\hat{w}$. Aquest arcs són, d'una banda, aquells que no estan connectats amb ambdos vèrtexs, i de l'altra, un de cada dels dos arcs que connecten un vèrtex que és veí de $\hat{v}$ i $\hat{w}$, ja que ara $\hat{v}$ i $\hat{w}$ seran un únic vèrtex.

L'algorisme de particionat en *cliques* dirigit pels pesos dels arcs es descriu a l'algorisme 6.5. El bucle exterior selecciona l'arc $\hat{e}_{best} = (\hat{v}, \hat{w})$ amb pes màxim (τ) i NVC màxim (si més d'un arc té el mateix NVC, se selecciona aquell amb NAE mínim). Els vèrtexs $\hat{v}$ i $\hat{w}$ són fusionats en el vèrtex $\hat{v}$. A continuació s'actualitzen els valors de les funcions $W(\hat{e})$, $NVC(\hat{e})$ i $NAE(\hat{e})$ per als arcs afectats. El bucle interior selecciona en cada iteració un vèrtex adjacent a $\hat{v}$ amb pes màxim i el fusiona amb $\hat{v}$. El procés es repeteix fins que no queda cap arc en el GC (si no hi ha cap arc vol dir que cap vèrtex és compatible amb cap altre i per tant no podem fusionar-los perquè comparteixin recursos).

GLASS utilitza aquest algorisme per fer associació de recursos. El tret diferencial més important, però, es troba en el fet que el graf de compatibilitat que s'utilitza a *GLASS* és un graf de compatibilitat global. En el graf de compatibilitat global tots els tipus de

particionat en cliques *dirigit pels pesos (GC)*{
per *tot* $\hat{e} \in \hat{A}$ **fer** *calcular* $W(\hat{e})$;
repetir
$\tau \leftarrow \max_{\hat{e} \in \hat{A}}(W(\hat{e}))$
$\hat{A}_{red} = \{\hat{e} \in \hat{A} | W(\hat{e}) = \tau\}$
per *tot* $\hat{e} \in \hat{A}_{red}$ **fer** *calcular* NVC($\hat{e}$) *and* NAE($\hat{e}$);
repetir
$\hat{e}_{best} \leftarrow$ seleccionar_vèrtex(τ);
$fusionar(\hat{v}, \hat{w})$;
per *tot* $\hat{e} \in \hat{A}$ **fer** *calcular* $W(\hat{e})$;
$\hat{A}_{red} = \{\hat{e} \in \hat{A} | W(\hat{e}) \geq \tau\}$
per *tot* $\hat{e} \in \hat{A}_red$ **fer**
calcular NVC($\hat{e}$) *i* NAE($\hat{e}$);
mentre $\{\hat{e} = (\hat{v}, \hat{v}'), \hat{e} \in \hat{A} : W(\hat{e}) \geq \tau\} \neq \emptyset$ **fer**
$\hat{e}_{best} \leftarrow$ seleccionar_vèrtex_adjacent(τ, v);
$fusionar(\hat{v}, \hat{w})$;
per *tot* $\hat{e} \in \hat{A}$ **fer** *calcular* $W(\hat{e})$;
$\hat{A}_{red} = \{\hat{e} \in \hat{A} | W(\hat{e}) \geq \tau\}$
per *tot* $\hat{e} \in \hat{A}_{red}$ **fer**
calcular NVC($\hat{e}$) *i* NAE($\hat{e}$);
fimentre
fins que $\hat{A}_{red} = \emptyset$;
fins que $A = \emptyset$;
}

Algorisme 6.5: Algorisme de particionat en cliques *dirigit pel pes dels arcs*

vèrtexs del *SDFG* estan representats i per tant les diferents decisions d'associació poden ser entrellaçades. En totes les altres propostes anteriors, es construia un graf de compatibilitat per a cadascun dels tipus de vèrtexs (operacions, variables i transferències de dades) i les diferents tasques d'associació s'executaven de manera seqüencial sense tenir en compte la relació entre les diferentes decisions d'associació.

6.3.1 Complexitat i codi de GLASS

La complexitat de *GLASS* està dominada per les fusions i per les operacions que s'executen al voltant de les fusions (càlcul dels pesos, càlcul del nombre de veïns comuns...). Si n és el nombre de vèrtexs del graf, en el pitjor cas es realitzaran $n-1$ fusions (el pitjor cas és quan el graf és totalment connex). Per a cada fusió es realitzen les següents accions:

- seleccionar un vèrtex
- fusionar
- recalcular els pesos
- calcular el nombre de veïns comuns i el nombre d'arcs a esborrar

Si es manté un punter al vèrtex més prioritari —aquest punter s'actualitza quan es recalculen els pesos, *NVC* i *NAE*— la funció de seleccionar té cost constant.

El cost de fusionar està dominat pels arcs veïns que s'han d'esborrar. Si el grau màxim del graf de compatibilitat és k, $k < n$ (grau de compatibilitat), la complexitat de fusionar seria com a molt $O(k^2)$. En el pitjor cas, quan el graf és totalment connex, el grau seria $n-1$ i la complexitat de fusionar, $O(n^2)$.

La complexitat de recalcular pesos, calcular *NVC* i calcular *NAE* també depèn del grau del graf. Com abans, en cas que el grau sigui k, tenim complexitat $O(k^2)$ i $O(n^2)$ en el cas pitjor.

Així doncs, la complexitat de *GLASS* és $O(n^3)$ en el cas pitjor i $O(nk^2)$ si el grau del graf és k i n és el nombre de vèrtexs del graf de compatibilitat.

GLASS ha estat programat en llenguatge C ocupant un total de 2.400 línies de codi.

6.4 Resultats

Aquesta secció presenta alguns resultats obtinguts amb *GLASS*. *GLASS* és una metodologia d'associació de recursos no basada en el cicle, tret que la fa apta per a sistemes síncrons i asíncrons. Els resultats que es mostren en aquesta secció corresponen primer a exemples asíncrons i després a sistemes síncrons, fet que ens permet comparar-los amb sistemes existents.

Les planificacions utilitzades com a SDFG d'entrada han estat obtingudes amb els algorismes *ELS* i *ELLAS*. Aquestes planificacions s'obtenen restringint el nombre màxim d'unitats funcionals de cada tipus a utilitzar. En aplicar *GLASS* a aquestes planificacions, pot passar que el nombre d'unitats funcionals de la solució final sigui més gran que el fixat a priori. Això es degut al fet que *GLASS* intenta minimitzar l'àrea total de la solució, independentment del nombre d'unitats de cada tipus que s'utilitzin.

Per a tots els resultats obtinguts es fa una estimació de l'àrea necessària per a cada solució. Aquest càlcul s'obté a partir d'una estimació de l'àrea dels diferents recursos que s'utilitzen.

Tots els temps de CPU que es donen en aquesta secció són d'una SUN SPARCstation 10.

6.4.1 Resultats per sistemes asíncrons

El fet que no existeixin sistemes de síntesi d'alt nivell asíncrons ens impedeix comparar aquesta metodologia quan el model d'arquitectura utilitzat és asíncron. En aquesta secció es presenten els resultats obtinguts quan es considera un model d'arquitectura asíncron per dos exemples: el filtre el·líptic de cinquè ordre [DDN85] i la transformada discreta del cosinus [Wou, KN92].

Per poder comparar les diferents solucions obtingudes en termes d'àrea s'han fet unes estimacions. En elles se suma l'àrea de les unitats que componen la solució però no es té en compte el connexionat. Els valors d'àrea utilitzats provenen en part d'una llibreria que teniem a l'abast i en altres casos s'ha extrapolat d'aquests valors l'àrea d'unitats no disponibles a la llibreria, com és el cas dels multiplicadors. La taula 6.2 mostra els valors

sumador	ALU	multiplicador	registre	mux 2:1
42	84	714	10	14

Taula 6.2: Valors utilitzats a les estimacions d'àrea del resultats asíncrons

sumadors	ALU	multiplicadors	registres	multiplexors 2:1	temps CPU	estimació àrea
1	2	2	11	28	0,47 s	2140
1	2	2	12	27	0,45 s	2136
1	3	2	11	29	0,42 s	2238
1	3	2	12	26	0,42 s	2206

Taula 6.3: Taula de resultats per al filtre el·líptic de cinquè ordre si es considera un model d'arquitectura asíncron

utilitzats per als resultats asíncrons.

6.4.1.1 Filtre el·líptic de cinquè ordre

Aquesta secció mostra els resultats que es poden obtenir quan apliquem *GLASS* a la planificació de la figura 6.4 obtinguda amb el programa ELS (vegeu capítol 5). Aquesta planificació s'ha obtingut fixant un vector de recursos amb dues ALU, dos multiplicadors i un sumador. Els retards estimats d'aquestes unitats són els que mostra la taula 5.3. La planificació té un retard estimat de 740 unitats de temps.

Una possible solució que obté *GLASS* és la que es mostra a la figura 6.5. Aquesta configuració utilitza dues ALU, dos multiplicadors, un sumador, 11 registres i 13 multiplexors (equivalents a 28 multiplexors de dues entrades).

La taula 6.3 mostra diferents resultats que s'han obtingut amb *GLASS*, fixant els paràmetres d'entrada del programa a valors diferents. Podem observar com *GLASS* busca un compromís entre l'àrea total de la solució (àrea de les unitats funcionals, àrea de registres i àrea d'interconnexió). Utilitzant alguna unitat funcional o registre més, l'àrea d'interconnexió es redueix. Podem veure com l'estimació d'àrea de totes les solucions és

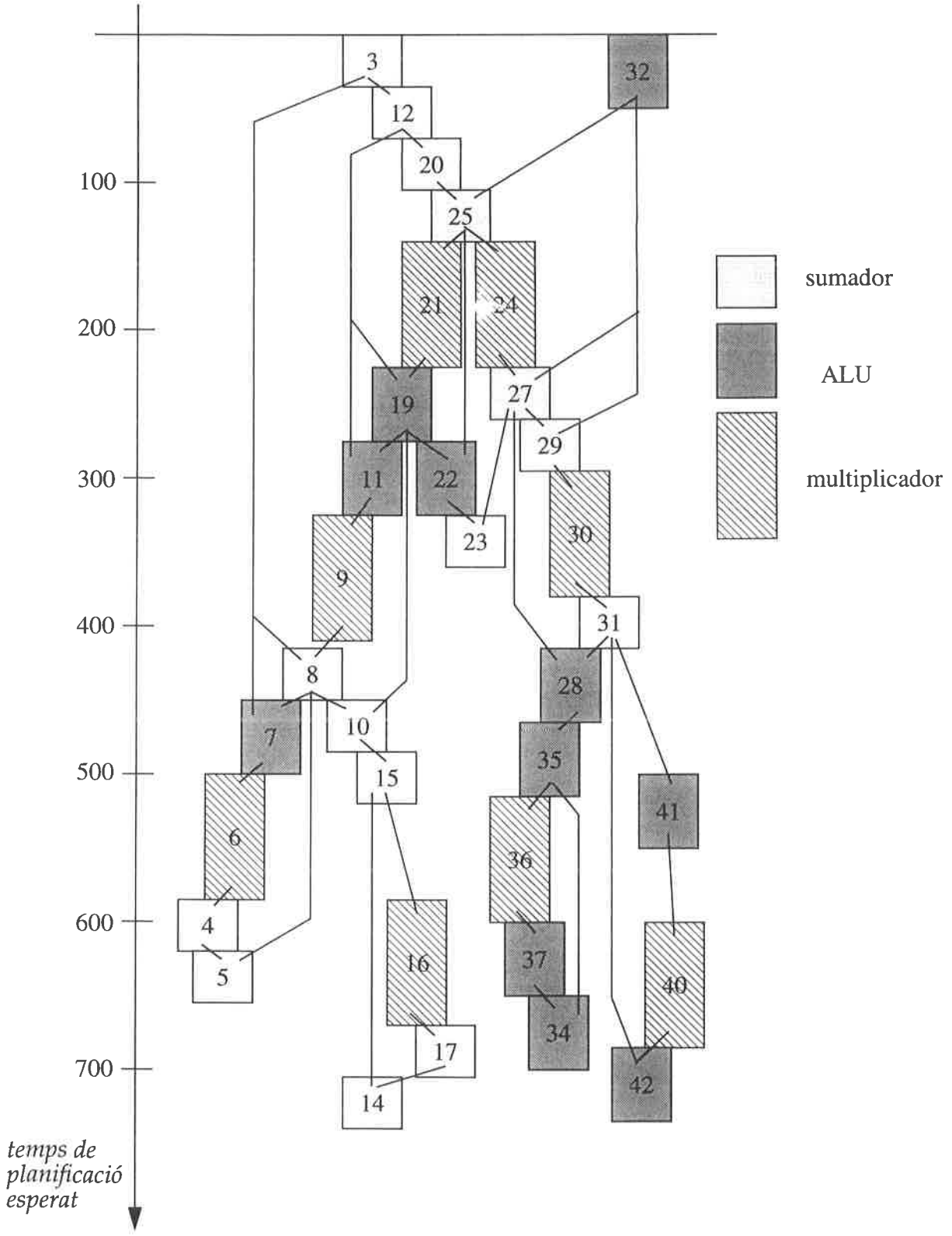


Figura 6.4: Planificació asíncrona del filtre el·líptic utilitzant ELS

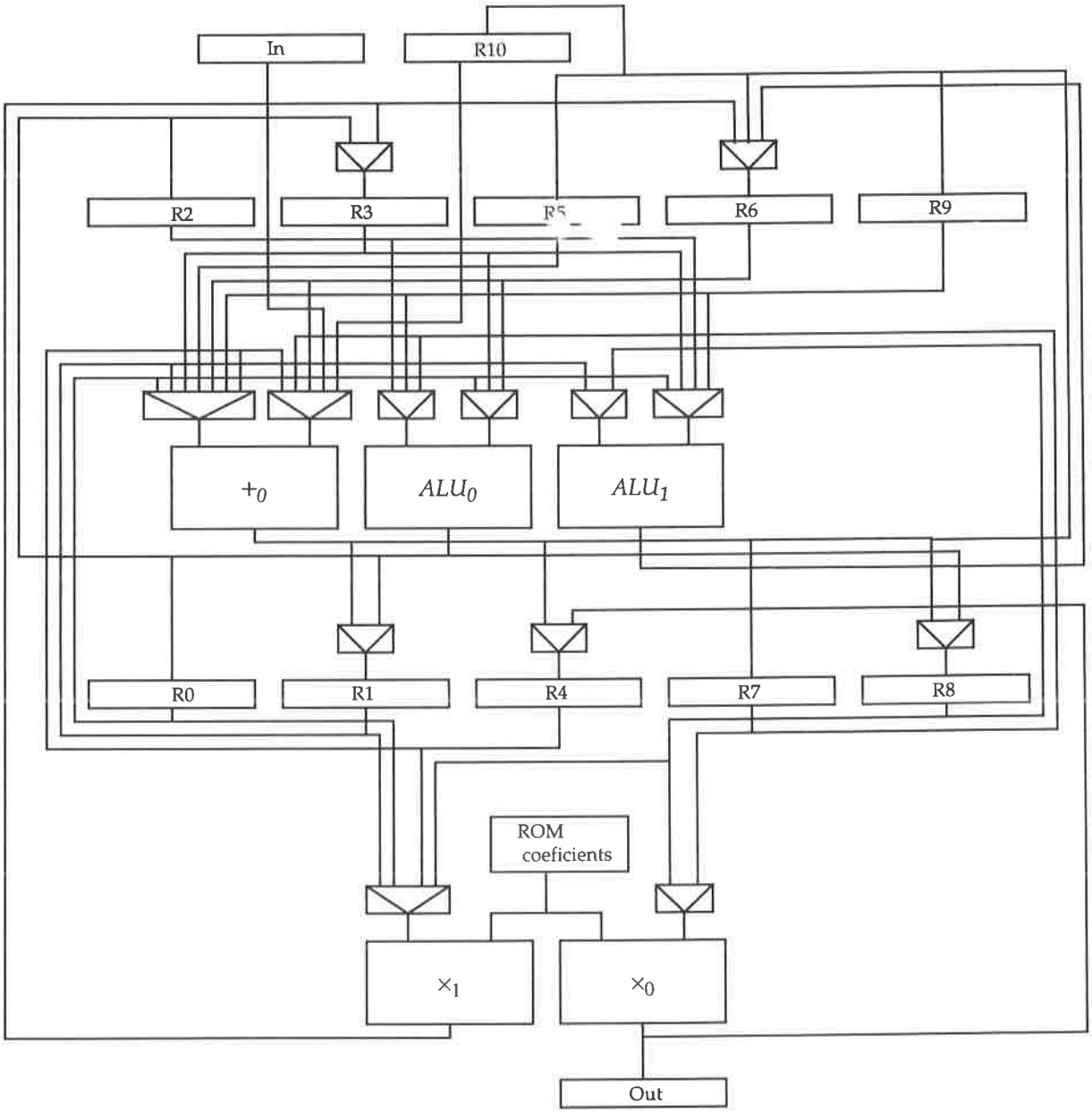


Figura 6.5: Camí de dades obtingut per GLASS *per la planificació de la figura 6.4*

ALU	multiplicadors	registres	multiplexors 2:1	temps CPU	estimació àrea
2	3	18	48	1,8 s	3162
2	3	19	46	1,8 s	3144
2	3	20	45	1,8 s	3140
3	3	19	44	1,9 s	3200

Taula 6.4: Taula de resultats per a la transformada discreta del cosinus quan es considera un model d'arquitectura asíncron

més o menys equivalent.

El problema d'associar recursos en sistemes asíncrons és molt més complex que en sistemes síncrons a causa que en sistemes síncrons els intervals de temps de vida dels diferents vèrtexs del *SDFG* sempre comencen i acaben al final (o principi, depèn de com es miri) d'un cicle. En sistemes asíncrons, com que no hi ha cicles, els intervals de temps de vida comencen i acaben en instants de temps qualssevol. Aquest fet provoca que vèrtexs del *SDFG* que podrien ser compatibles no ho són per un interval de temps molt petit (vegeu figura 6.6). Això es podria arreglar modificant lleugerament la planificació d'operacions o utilitzant una eina que realitzi simultàniament la planificació de les operacions i l'associació dels recursos, tal com veurem al següent capítol.

6.4.1.2 Transformada discreta del cosinus

Per aquest exemple s'ha considerat una planificació obtinguda per ELLAS fixant una restricció de recursos de dues ALU i tres multiplicadors. El retard estimat de la planificació és de 350 unitats de temps. La matriu de retards d'aquestes unitats funcionals es mostra la taula 5.7.

La taula 6.4 mostra diferents solucions que s'han obtingut amb *GLASS* fixant els seus paràmetres d'entrada a valors diferents. Igual que en l'exemple anterior, *GLASS* intenta equilibrar el cost d'àrea buscant un compromís entre l'àrea corresponent a cada tipus de recurs. Així, solucions que utilitzin un nombre mínim d'unitats funcionals tindran un cost d'interconnexió més alt que si s'utilitza alguna unitat més. L'estimació d'àrea ens mostra

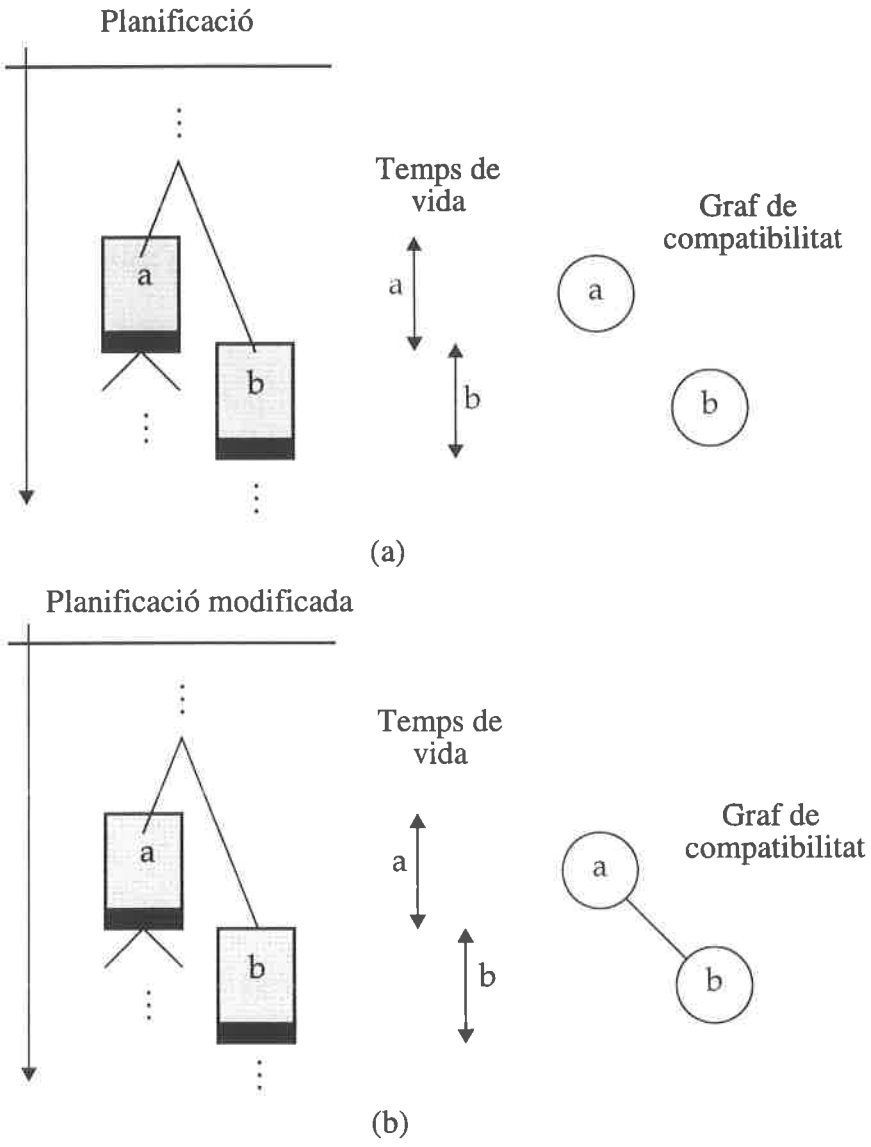


Figura 6.6: (a) Planificació de dos operacions on el temps de vida se solapa i no poden compartir recurs; (b) Modificació de la planificació inicial per tal que puguin compartir un recurs

sumador	ALU	multiplicador	restador	comparador	registre	mux 2:1	tres-estats
22	44	374	25	25	15	7	3,5

Taula 6.5: Valors utilitzats a les estimacions d'àrea del resultats síncrons

com solucions que utilitzen més unitats funcionals, tenen un cost en àrea equivalent a les que n'utilitzen menys.

6.4.2 Resultats per sistemes síncrons

En aquesta secció es presenten els resultats obtinguts per *GLASS* quan es considera un model d'arquitectura síncron. En primer lloc es presenten els resultats obtinguts per dos exemples senzills, l'equació diferencial [PK89a] i l'exemple proposat a FACET [TS86]. A continuació es presenten resultats per a exemples més complexes, com el filtre el·liptic de cinquè ordre [DDN85] o la transformada discreta del cosinus [Wou, KN92].

Com en els resultats obtinguts per sistemes asíncrons, s'han fet estimacions de l'àrea que ocuparan les solucions. La taula 6.5 mostra els valors d'àrea utilitzat en aquest cas.

6.4.2.1 Equació diferencial

Tot i la seva simplicitat, l'exemple de l'equació diferencial ha estat molt utilitzat per comparar resultats des que va ser proposat per primera vegada a [PKG86]. A la planificació s'han restringit els recursos a dos multiplicadors, un sumador, un comparador i un restador. S'han comparat els resultats de *GLASS* amb els que han obtingut els sistemes HAL [PK89a], SPLICER [Pan88] i ARYL/LYRA [HCLH90].

La taula 6.6 mostra aquests resultats. Per a cada entrada de la taula s'ha comptabilitzat el nombre de multiplicadors, registres, multiplexors 2:1, entrades a multiplexors i nombre total de multiplexors utilitzats a la configuració. També es presenta el temps de CPU invertit per obtenir cadascuna de les solucions. El nombre de multiplexors 2:1 ens permet comparar entre les diferents solucions, ja que un multiplexor genèric de n entrades pot ésser construit com un arbre de $n-1$ multiplexors 2:1. El nombre de multiplexors 2:1

	mult	reg	mux 2:1	entrades	mux	CPU	estimació àrea
HAL	2	10	7	13	6	140 s	1019
SPLICER	2	10	6	11	5	1245 s	1012
LYRA	2	9	5	9	4	0,17 s	990
ARYL	2	9	5	9	4	0,04 s	990
GLASS	2	8	7	13	6	0,03 s	989

Taula 6.6: Resultats per a l'equació diferencial utilitzant una planificació de quatre cicles i una restricció de recursos de dos multiplicadors, un sumador, un restador i un comparador

necessaris pot calcular-se com la diferència entre el nombre d'entrades a multiplexors i el nombre total de multiplexors.

Els resultats obtinguts per *GLASS* superen clarament els obtinguts per HAL i SPLICER. En el cas de ARYL i LYRA, els resultats obtinguts per *GLASS* són comparables, ja que en alguns casos *GLASS* utilitza més interconnexió i menys unitats funcionals i en altres més unitats funcionals o registres i menys interconnexió. Podem veure com l'estimació d'àrea dóna avantage a *GLASS*, encara que sigui per molt poc. Els algorismes ARYL i LYRA intenten que les unitats funcionals i registres comparteixin els multiplexors de les entrades per tal de minimitzar el cost en interconnexió. Aquest objectiu és fàcil d'assolir en determinats exemples, com el de l'equació diferencial, però no sempre és així.

La figura 6.7 mostra el camí de dades per a un dels resultats obtinguts per *GLASS*.

6.4.2.2 Exemple del FACET

Aquest exemple va ser presentat per primera vegada en el sistema FACET [TS86] i és conegut per aquest motiu com l'exemple del FACET. Per fer la planificació d'aquest exemple s'utlitzen tres recursos: una ALU1 capaç de realitzar sumes, restes i AND, una ALU2 capaç de fer sumes, multiplicacions i OR i un divisor. La planificació necessita quatre cicles, tal com mostra la figura 6.8.

Els resultats per a aquest exemple s'han comparat amb els obtinguts pels sistemes FACET, SPLICER, ARYL i LYRA. La taula 6.7 mostra per a cada cas el nombre de

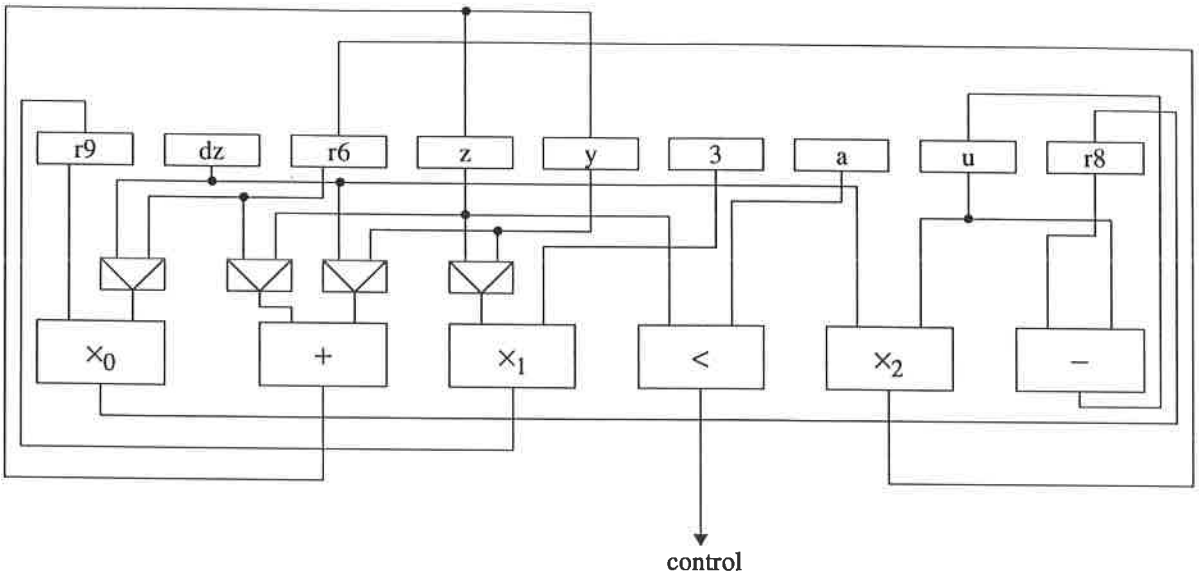


Figura 6.7: Camí de dades per a l'exemple de l'equació diferencial

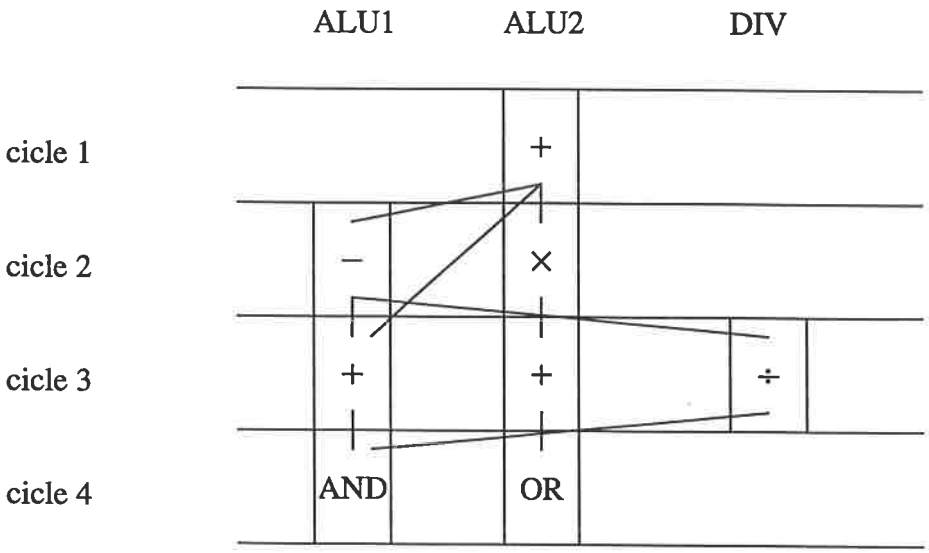


Figura 6.8: Planificació de l'exemple del FACET

	reg	mux 2:1	entrades	mux	CPU	estimació àrea
FACET	8	5	9	4	–	661
SPLICER	7	4	8	4	1.4 s	631
LYRA	7	4	8	4	0,12 s	631
ARYL	7	4	8	4	0,02 s	631
GLASS	7	4	8	4	0,02 s	631

Taula 6.7: Resultats per a l'exemple del FACET

registres, multiplexors 2:1, entrades a multiplexors, nombre total de multiplexors i temps de CPU. En tots els casos, el nombre d'unitats funcionals utilitzades és: una ALU1, una ALU2 i un divisor.

Els resultats obtinguts per *GLASS* per a aquest exemple són millors que els del sistema FACET i iguals als dels sistemes SPLICER, ARYL i LYRA.

6.4.2.3 Filtre el·líptic de cinquè ordre

A continuació veurem els resultats que s'obtenen per al *filtre el·líptic* quan es considera un model d'arquitectura síncron. Per fer l'assignació de recursos s'han partit de planificacions obtingudes amb els programes *ELS* i *ELLAS*. Per a aquest exemple, com que és el més referenciat a la literatura, es consideren diferents casos i diferents models d'arquitectura: model d'arquitectura orientat a multiplexors i a busos.

Els resultats obtinguts amb el filtre el·liptic de cinquè ordre [DDN85] són comparats amb els resultats obtinguts a MABAL [KP90b], ARYL/LYRA [HCLH90], SALSA [KN92], ELF [LEG90], SPAID [HE88] i SCHALLOC [BP90].

La taula 6.8 mostra els resultats obtinguts utilitzant un multiplicador de dos cicles de latència i dos sumadors amb latència d'un cicle i una planificació d'operacions de 21 cicles. La taula representa: nombre de registres, nombre de multiplexors 2:1, nombre d'entrades de multiplexor, nombre total de multiplexors, temps de CPU i tècnica utilitzada.

GLASS millora els resultats obtinguts amb propostes anteriors polinòmiques i obté resultats comparables (encara que pitjors) als obtinguts amb tècniques amb ús intensiu de

	reg	mux 2:1	entrades	mux	CPU	Tècnica	àrea
HAL	12	20	26	6	6 min	Particionat en *cliques*	738
SPLICER†	–	34	43	9	55 s	bifurcar & tallar	–
MABAL	10	30	42	12	10,52 s	Algorisme Heurístic	778
ARYL	10	22	30	8	0,65 s	Correspondència Bipartida Ponderada	722
LYRA	10	23	31	8	3,38 s	Correspondència Bipartida Ponderada	729
SALSA	10	15	–	–	8-10 min	Millora Iterativa	673
SALSA	11	16	–	–	8-10 min	Millora Iterativa	695
ELF	11	16	24	8	1 hora	Simulació de l'Evolució	695
GLASS	10	21	27	6	0,53 s	Particionat en *cliques*	715
GLASS	11	19	25	6	0,48 s	Particionat en *cliques*	716

Taula 6.8: Resultats per al filtre el·liptic † (No utilitza ROM de coeficients)

CPU (SALSA i ELF). Aquestes propostes obtenen solucions millors en àrea a un cost de un temps de CPU molt més alt.

La primera solució obtinguda per *GLASS* utilitza deu registres i 21 multiplexors 2:1. La segona utilitza 11 registres i 19 multiplexors 2:1. Podem veure que l'estimació en àrea de les solucions obtingudes per *GLASS* no s'allunyen gaire de les obtingudes pels sistemes SALSA o ELF (entre un 2% i 6%). Un punt que cal fer ressaltar és que el temps de CPU requerit per *GLASS* és molt reduït, de l'ordre de mig segon.

La taula 6.9 descriu els resultats obtinguts utilitzant dos sumadors i un multiplicador segmentat amb una planificació de 19 cicles. En aquest cas els resultats obtinguts per *GLASS* són gairebé pitjors en tots els casos. Això és degut al fet que només es compara a un algorisme heurístic (MABAL), que és el que es millora. Tots els altres són algorismes de millora iterativa, amb un cost de CPU molt alt, però que obtenen bones solucions. En aquest cas, l'estimació en àrea de les solucions obtingudes per *GLASS* varia entre un 2% i un 8% respecte a les solucions obtingudes pels sistemes SALSA i ELF.

Les taules 6.10 i 6.11 presenten els resultats obtinguts quan s'utilitza una arquitectura orientada al bus. Per ambdues taules, *GLASS* millora els resultats obtinguts amb propostes

	reg	mux 2:1	entrades	mux	CPU	Tècnica	àrea
MABAL	16	32	45	13	7,48 s	Algorisme Heurístic	882
SALSA	10	19	–	–	8-10 min	Millora Iterativa	701
SALSA	11	16	–	–	8-10 min	Millora Iterativa	695
SALSA	12	18	–	–	8-10 min	Millora Iterativa	724
ELF	11	19	30	11	1 hora	Simulació de l'Evolució	716
GLASS	11	23	29	6	0,43 s	Particionat en *cliques*	744
GLASS	12	22	31	9	0,45 s	Particionat en *cliques*	752

Taula 6.9: Resultats per al filtre el·líptic

	reg	bus	entr. bus	mux 2:1	entr. mux	CPU	Tècnica	àrea
MABAL	10	2	11	22	32	14,46 s	Algorisme Heurístic	761
SCHALLOC	13	9	–	40	53	179 s	Bifurca & Talla	893*
GLASS	12	5	19	8	11	4,43 s	Part. en *cliques*	721
GLASS	12	3	10	13	19	4,45 s	Part. en *cliques*	724

Taula 6.10: Resultats per al filtre el·líptic utilitzant un model d'arquitectura orientat al bus.
**(sense comptar l'àrea de les connexions a busos)*

	regs	bus	entr. bus	mux 2:1	entr. mux	CPU	Tècnica	àrea
MABAL	11	2	14	30	43	16,70 s	Algorisme Heurístic	1216
SPAID	19	6	18	–	26		Recerca *Best First*	1252
GLASS	12	3	11	16	24	5,87 s	Particionat en *cliques*	1123

Taula 6.11: Resultats per al filtre el·líptic

anteriors. La taula 6.10 s'ha obtingut a partir d'una planificació de 21 cicles. La planificació s'ha obtingut fixant una restricció de recursos dos sumadors i un multiplicador. Aquesta taula representa: nombre de registres, nombre de busos, nombre d'entrades a bus, nombre de multiplexors 2:1, nombre d'entrades a multiplexors, temps de CPU i tècnica utilitzada.

En aquest cas, l'estimació d'àrea és de gran ajuda per decidir quina solució és millor o pitjor, ja que el nombre de recursos que apareixen a la taula és major. En el cas del sistema SCHALLOC, l'àrea estimada no té en compte el cost de les connexions a busos, ja que aquesta informació no està disponible. Així i tot, el cost en àrea d'aquesta solució és més gran que els altres. La solució obtinguda per MABAL, tot i que requereix menys busos, utilitza més multiplexors. Per aquest motiu, l'àrea total d'aquesta solució és més gran que la de les solucions obtingudes per *GLASS*.

La taula 6.11 mostra els resultats obtinguts si utilitzem una planificació de 18 cicles, amb una restricció de recursos de dos sumadors i dos multiplicadors. En aquest cas, *GLASS* millora als altres dos algorismes amb què s'ha comparat.

6.4.2.4 Transformada discreta del cosinus

Per aquest exemple s'han utilitzat planificacions generades per *ELLAS* i es compara amb els resultats obtinguts pel sistema SALSA [KN92]. La taula 6.12 mostra els resultats obtinguts per dues planificacions diferents. Els recursos utilitzats són una ALU que pot realitzar sumes i restes en un cicle i un multiplicador que realitza multiplicacions en dos cicles.

El primer dels exemples s'obté fixant les restriccions d'àrea a tres ALU i tres multiplicadors. Les planificacions obtingudes pel sistema SALSA i per ELLAS utilitzen 14 cicles. El segon s'obté amb unes restriccions de dues ALU i tres multiplicadors. El sistema SALSA

	cicles	ALU	mult	reg	mux 2:1	CPU	àrea
SALSA	14	3	3	14	29	10-17 min	1667
GLASS	14	4	3	18	32	1,8 s	1792
SALSA	19	2	3	14	30	10-17 min	1630
GLASS	16	2	3	18	36	1,9 s	1732
GLASS	16	3	3	19	34	1,9 s	1777

Taula 6.12: Resultats per a la transformada dis cta del cosinus en un model d'arquitectura síncron

obté una planificació de 19 cicles, mentre que la d'*ELLAS* és de 16 cicles. Els resultats obtinguts per *GLASS* en aquest exemple, tot i que no milloren els resultats del programa SALSA, demostren que el sistema és capaç de tractar amb DFG grans i obtenir resultats bons en un temps de CPU molt curt.

6.5 Conclusions

En aquest capítol s'ha presentat un proposta d'associació de recursos basada en la teoria de grafs on totes les tasques d'associació són executades de forma simultània. Els vèrtexs del graf de compatibilitat representen diferents tipus de vèrtexs (operacions, variables o transferències de dades) de manera que parells de vèrtexs connectats per un arc poden ser *fusionats* en qualsevol instant independentment de quin tipus d'operació representin.

Els arcs del graf de compatibilitat són etiquetats amb un pes que indica l'afinitat d'interconnexió dels vèrtexs connectats per aquest arc. Els pesos dels arcs serveixen per guiar el procés cap a una configuració amb un cost d'interconnexió baix.

Tot i que els resultats obtinguts són inferiors als obtinguts amb tècniques amb ús intensiu de la CPU, milloren els obtinguts per tècniques polinòmiques anteriors.

La representació compacta d'un únic graf de compatibilitat sembla un bon punt de partida per a altres algorismes, com per exemple, tècniques de millora iterativa.

La figura 6.9 presenta l'esquema d'un sistema de síntesi d'alt nivell. En aquest esquema

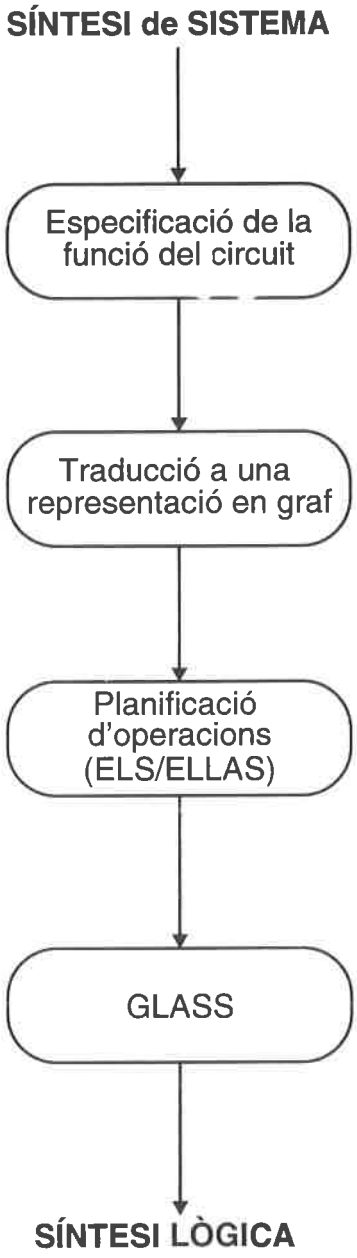


Figura 6.9: GLASS *en un sistema de síntesi d'alt nivell*

veiem que *GLASS* s'executaria després de la fase de planificació d'operacions, que podria ser executada amb un dels algorismes presentats al capítol anterior (*ELS* o *ELLAS*).

Capítol 7

Planificació i assignació simultània

La planificació d'operacions en sistemes asíncrons és una tasca més complexa que en el cas síncron. Un dels motius d'aquesta complexitat és la no existència de cicles que fixin un marc per a l'inici i la finalització de les operacions. Un altre és que el temps d'execució de les operacions depèn en general de les dades d'entrada i, per tant, s'han de fer estimacions dels retards per fer la planificació. A més tenim la complexitat inherent a la majoria de mòduls autotemporitzats, on cal tenir en compte etapes d'inicialització entre operació i operació. En aquest capítol es presenta un algorisme que realitza de manera simultània la planificació d'operacions i l'assignació de recursos. L'algorisme està basat en la tècnica de recuit simulat. Es permet l'encadenament d'operacions amb l'objectiu de minimitzar el temps de càlcul en els camins crítics.

7.1 Introducció

La planificació d'operacions en sistemes asíncrons és més complexa que en els síncrons per diversos motius:

- El temps d'execució de les unitats funcionals depèn de les dades d'entrada.
- Les sincronitzacions entre els mòduls són explícites.
- En molts casos la planificació ha de tenir en compte fases d'inicialització dels mòduls (per exemple, si construïm el camí de dades amb mòduls autotemporitzats amb lògica DCVSL).
- Si es permet l'encadenament d'operacions, l'interval d'ocupació de les unitats funcionals per part de les operacions no és conegut fins que es planifiquen les operacions successores.

Aquests motius, sobretot el darrer, suggereixen que un algorisme transformacional és més adequat per a la planificació d'operacions en sistemes asíncrons, i encara més si es fa l'assignació de recursos de manera simultània. Els algorismes transformacionals parteixen d'una configuració inicial i en cada iteració apliquen diferents transformacions per obtenir noves configuracions. En tot moment es disposa d'una configuració on cada vèrtex del graf té una planificació i una assignació a un recurs. D'aquesta manera els intervals d'ocupació de les unitats funcionals és conegut per totes les operacions i en tot moment.

La majoria d'algorismes transformacionals són formulacions d'un algorisme genèric aplicable a diferents camps. Dins d'aquests algorismes podem destacar diferents propostes [WLL88]: genètics, simulació de l'evolució, recuit simulat...

S'ha triat l'algorisme de *recuit simulat* [KGV83, vLA89], entre altres motius, perquè ja teníem una certa experiència en aquest algorisme [CBA91b]. L'algorisme de recuit simulat parteix d'una configuració inicial qualsevol i explora l'espai de configuracions intentant de no caure en mínims locals. Per fer-ho, es mou a configuracions que incrementen el cost de la solució amb un certa probabilitat que depèn d'un paràmetre intern: la *temperatura*. A temperatures altes, el procés es mou per tot l'espai de configuracions, encara que el cost

s'incrementi. A temperatures baixes, el procés es mou a configuracions que millorin el cost o que no l'incrementin molt.

L'algorisme fa una analogia amb el procés físic de recuit dels sòlids, en què l'objectiu és reorganitzar l'estructura i eliminar-ne les tensions internes. El procés consisteix en un escalfament fins a una temperatura adequada, seguit de refredaments controlats. D'aquesta manera, les partícules del sòlid es reorganitzen de manera aleatòria, fins a arribar a un estat d'energia mínima [vLA89].

Aquest capítol presenta un algorisme que permet fer planificació d'operacions i assignació de recursos de manera simultània. L'algorisme està basat en el recuit simulat. El model d'execució utilitzat permet encadenar operacions. L'algorisme treballa sota restriccions del nombre màxim d'unitats funcionals a utilitzar.

L'estructura del capítol és la següent: la secció 7.2 presenta l'algorisme de recuit simulat, la secció 7.3 presenta l'entorn del programa. A continuació es presenta el model d'execució. A la secció 7.5 es presenten diferents aspectes relacionats amb l'algorisme desenvolupat. Per acabar, a la secció 7.6 es presenten els resultats obtinguts amb el programa, i a la 7.7, les conclusions del capítol.

7.2 L'algorisme de recuit simulat

A l'hora de buscar heurístiques per a problemes en què no es coneixen tècniques de resolució polinòmiques per a trobar l'òptim, podem optar entre dues alternatives:

- Algorismes fets a mida: són aquells que s'adapten a un problema específic.
- Algorismes genèrics: són aquells aplicables a una gran varietat de problemes, tot i que cal personalitzar-los per a cada cas.

El *recuit simulat* és un algorisme genèric d'optimització combinatòria, basat en tècniques d'aleatorització, però que també incorpora nombrosos aspectes relacionats amb els algorismes de millora iterativa. L'algorisme de recuit simulat va ser presentat per Kirkpatrick i altres [KGV83] com a generalització del mètode de *Monte Carlo* proposat per Metropolis i altres [MRR+53].

```
recuit simulat() {
    S := configuració inicial ();
    T := temperatura inicial ();
    mentre no es satisfaci el criteri de parada fer {
        mentre no s'hagi arribat a l'equilibri fer {
            S' := generar nova configuració (S);
            ΔC := cost (S') − cost (S);
            si acceptar(ΔC, T) llavors S := S'
        fimentre
    T:=actualitzar (T)
    fimentre
}
```

Algorisme 7.1: Algorisme de recuit simulat

Tal com mostra l'algorisme 7.1, el recuit simulat comença amb una configuració inicial arbitrària. Mitjançat l'aplicació de petites perturbacions, s'obtenen noves configuracions. La nova configuració és acceptada o no segons una funció d'acceptació que depèn de l'increment de cost de la configuració i d'un paràmetre de control.

En el que queda de secció, parlarem de diferents aspectes de l'algorisme, com per exemple, de la funció d'acceptació, del paràmetre de control T —també anomenat *temperatura*— i dels criteris de parada.

7.2.1 Funció d'acceptació

La funció d'acceptació depèn d'una banda, de l'increment de cost de la nova configuració (ΔC), i de l'altra, de la temperatura (T). L'algorisme 7.2 presenta la funció *acceptar*.

- Si l'increment de cost és negatiu (la nova configuració té un cost menor que l'antiga), sempre s'accepta la nova configuració.

```
acceptar (ΔC, T) {
    si ΔC ≤ 0 llavors retorna (cert);
    sinó si e^{-ΔC/T} > random[0,1) llavors retorna (cert);
        fisi
    fisi
    retorna (fals);
}
```

Algorisme 7.2: Funció acceptar

- Si l'increment de cost és positiu, la nova configuració s'accepta amb una certa probabilitat, segons una funció de Boltzmann:

$$p = e^{-\Delta C/T}$$

Cal destacar que si l'increment de cost tendeix a infinit, la probabilitat d'acceptació tendirà a zero, és a dir, les configuracions que incrementin molt el cost no seran acceptades. També, si la temperatura tendeix a infinit, la probabilitat d'acceptar tendeix a 1, és a dir, quan la temperatura és alta hi ha molta probabilitat d'acceptar qualsevol nova configuració, encara que l'increment de cost sigui alt. En canvi, quan la temperatura tendeix a 0, la probabilitat d'acceptació tendeix a zero, de manera que quan la temperatura arriba a zero, l'algorisme té un comportament ambiciós i només accepta aquells moviments que decrementin el cost de la configuració.

7.2.2 Temperatura

El recuit simulat té un paràmetre de control que és conegut també amb el nom de *temperatura* per la analogia amb el procés físic. La *temperatura* té un valor inicial alt que va disminuint molt lentament al llarg del procés. A temperatures altes, la funció d'acceptació té més probabilitat d'acceptar noves configuracions, encara que l'increment de cost sigui alt. En canvi, a temperatures baixes, difícilment s'acceptaran noves configuracions que incrementin el cost.

7.2.2.1 Temperatura inicial

Hi ha diversos mètodes per calcular el valor inicial de la temperatura [vLA89, Sec88].

- Kirkpatrick i altres [KGV83] proposen seguir la regla següent: triar un valor alt per la temperatura T_0 i provar un conjunt de transicions. Si la proporció de configuracions acceptades χ, és menor que un cert valor χ_0 (a [KGV83] $\chi_0 = 0,8$), doblar el valor de T_0. Repetir aquest procés fins que la proporció de configuracions acceptades sigui més gran que χ_0.

- Johnson i altres [JAMY87] determinen la temperatura inicial calculant l'increment de cost mitjà ($\overline{\Delta C}$) per a un nombre aleatori d'iteracions i resolent la següent fórmula:

$$\chi_0 = e^{-\overline{\Delta C}/T_0} \tag{7.1}$$

  El resultat d'aïllar T_0 de la fórmula anterior és:

$$T_0 = \frac{\overline{\Delta C}}{\ln(\chi_0^{-1})} \tag{7.2}$$

- A [Whi84], White argumenta que una temperatura inicial adequada ha de complir la següent inequació:

$$\sigma \ll T_0 \tag{7.3}$$

  essent σ la desviació estàndard de la funció de distribució del cost. σ es calcula fent un nombre indeterminat d'iteracions i la temperatura inicial es calcula segons la fórmula següent:

$$T_0 = k\sigma \tag{7.4}$$

  El valor de k es determina suposant que la funció de distribució del cost segueix una normal i, que per a una temperatura suficientment alta, tenim probabilitat P d'acceptar una configuració tal que el seu cost és 3σ vegades pitjor que l'actual. Això porta a la següent expressió de k:

$$k = -\frac{3}{\ln P} \tag{7.5}$$

Fórmules semblants a les dues últimes són proposades a [LL85, LWL85, GS86, MS86, AvL85, LM86, OvG84]. A la nostra implementació, s'ha utilitzat l'última alternativa de les exposades.

7.2.2.2 Actualització de la temperatura

En cada iteració del bucle extern de l'algorisme, el valor de la temperatura s'actualitza, decrementant-lo. Generalment s'utilitza la regla següent per decrementar el valor de la temperatura:

$$T_{i+1} = \lambda T_i \tag{7.6}$$

on λ és una constant molt propera a 1, tot i que més petita. Kirkpatrick i altres [KGV83] varen proposar aquesta regla per primera vegada amb un valor de $\lambda = 0,95$, però ha estat utilitzada per altres propostes [JAMY87, LL85, MS86, SSV85].

A la nostra implementació, el valor de λ és un paràmetre del programa. S'ha fixat a un valor entre $0,9$ i $0,98$ segons els casos.

A [GS86], el bucle extern s'executa K vegades. Per calcular la temperatura, es divideix l'interval $[0, T_0]$ en K subintervals, de manera que la temperatura de cada iteració T_i és:

$$T_i = \frac{K - i}{K} T_0 \tag{7.7}$$

7.2.3 Execució del bucle intern: cadenes de Markov

Donat un espai de configuracions, podem considerar que una configuració és veïna d'una altra si podem passar d'una a l'altra aplicant una petita modificació. El *recuit simulat* pot ser considerat un algorisme que contínuament intenta transformar una configuració en una de les seves veïnes.

Aquest mecanisme pot ser matemàticament descrit mitjançant una *cadena de Markov*: una seqüència d'intents, en què el resultat de cada intent depèn únicament del resultat de l'intent anterior [Fel50].

En el cas del recuit simulat, els intents representen moviments i és clar que el resultat d'un moviment depèn només del resultat del moviment anterior [vLA89].

Podem diferenciar entre cadenes de Markov homogènies i no-homogènies. Parlarem de cadenes homogènies si la probabilitat de que el resultat d'un moviment sigui una determinada configuració no depèn de en quina iteració estem i altrament de cadenes no-homogènies.

Així, es diferencia entre dues formulacions diferents de l'algorisme:

- Algorisme homogeni: quan és descrit per una seqüència de cadenes de Markov homogènies. Cada cadena de Markov es genera amb un valor fix de la temperatura. La temperatura es decrementa entre cadenes de Markov consecutives.

- Algorisme no-homogeni: quan és descrit per una única cadena de Markov no-homogènia. El valor de la temperatura es decrementa entre moviments consecutius.

En el nostre cas considerarem el model d'algorisme homogeni. Queda per decidir quina és la millor longitud de la cadena de Markov. La longitud de la cadena de Markov es correspon amb el nombre de moviments que s'apliquen pera a cada valor de la temperatura. En la nostra implementació, la longitud de la cadena de Markov depèn de la mida del problema. Concretament, per a un DFG, $G = (V, A)$, s'ha fixat la longitud de la cadena de Markov a $|V| \cdot |A|$. Aquesta alternativa ha estat utilitzada en altres propostes [BL84, SSV85].

7.2.4 Criteri de parada

El criteri de parada, que a més determina el valor final de la temperatura, és determinat de diferents maneres:

- Fixant un nombre màxim de valors de la temperatura, K, que determina el nombre d'iteracions del bucle exterior que realitza el programa [NSS85, BL84].

- Si les últimes configuracions de cadenes de Markov consecutives són idèntiques per a un nombre determinat de cadenes, l'algorisme acaba [MS86, SSV85].

- L'últim criteri és refinat a [JAMY87]: a més de la condició anterior es requereix que la proporció de configuracions acceptades sigui menor que un cert valor χ_f.

En la nostra implementació, es compara el cost de l'última configuració de cada cadena de Markov. Si durant 20 iteracions el cost no varia de manera significativa (en un 1 per mil), el procés acaba.

7.2.5 Funció de cost i moviments

Per poder passar d'una configuració a una altra, cal un conjunt de moviments. També es necessita una funció de cost que permeti avaluar la qualitat de les configuracions. Per poder caracteritzar l'algorisme de recuit simulat per a un problema concret, s'ha de definir una funció de cost i un conjunt de moviments.

7.3 Entorn del programa

L'algorisme que es presenta té com a entrades un graf de flux de dades, una llibreria de mòduls autotemporitzats, unes restriccions d'àrea i uns paràmetres que ens permeten guiar l'execució del programa.

Les restriccions d'àrea s'especifiquen com el nombre màxim d'unitats funcionals de cada tipus que es poden utilitzar. El nombre de registres i multiplexors no es restringeixen, de manera que són variables del programa a optimitzar. Tampoc no es fixa cap restricció sobre el temps de planificació.

El programa també utilitza alguns factors que són obtinguts com a paràmetres d'entrada, que permeten guiar l'execució del programa.

7.3.1 Graf de flux de dades

El graf de flux de dades $G = (V, A)$ és un graf dirigit on els vèrtexs $v \in V$ representen operacions a executar i els arcs $a \in A$ representen dependències de dades entre les operacions. Per a cada vèrtex $v \in V$, definim:

v_o : tipus d'operació executada en el vèrtex v (suma, resta ...)

v_f : tipus de recurs en el qual s'executarà l'operació representada pel vèrtex v

v_r : recurs concret associat a l'operació representada pel vèrtex v

v_t : tipus de vèrtex: operació (`OP`) o variable (`VAR`)

$pred(v)$: conjunt de vèrtexs predecessors de v $(\{u \in V | (u, v) \in A\})$

$succ(v)$: conjunt de vèrtexs successors de v $(\{u \in V | (v, u) \in A\})$

Quan parlem de v_f ens referim a un tipus de recurs genèric, com pugui ser una ALU o un sumador i, quan parlem de v_r, ens referim a una realització concreta d'un recurs que s'ha associat a l'operació representada pel vèrtex v.

A més, definirem les funcions $ascendent(u, v)$ i $descendent(u, v)$ de dos vèrtexs $u, v \in V$ com:

$$ascendent(u,v) = \begin{cases} cert & \text{si existeix un camí en el DFG des de } u \text{ a } v \\ fals & \text{altrament} \end{cases} \tag{7.8}$$

$$descendent(u,v) = \begin{cases} cert & \text{si existeix un camí en el DFG des de } v \text{ a } u \\ fals & \text{altrament} \end{cases} \tag{7.9}$$

La funció *profunditat d'un camí entre dos vèrtexs u i v*, $profunditat(u, v)$ es defineix com:

$$profunditat(u,v) = \begin{cases} \text{nombre d'arcs del camí més curt de } u \text{ a } v \\ \qquad\qquad \text{si } ascendent(u,v) \text{ o } descendent(u,v) \\ \infty \qquad\qquad \text{altrament} \end{cases} \tag{7.10}$$

Els arcs representen dependències de dades. Per a cada dependència de dades caldrà una transferència de dades entre els vèrtexs predecessor i successor de l'arc. El model d'arquitectura utilitzat és orientat a multiplexors, de manera que les transferències de dades podran necessitar un multiplexor o no, segons el cas.

Definim la funció $MUX(a)$ per a un arc $a = (u, v) \in A$ com:

$$MUX(a) = \begin{cases} cert & \text{si és necessari un multiplexor per realitzar} \\ & \text{la transferència entre els vèrtexs } u \text{ i } v \\ fals & \text{altrament} \end{cases} \tag{7.11}$$

Per a cada arc $a \in A$, definim:

a_f : tipus de multiplexor en el qual es realitzarà la transferència

de dades representada per a si $MUX(a)$

a_r : multiplexor concret en què es realitzarà la transferència
de dades representada per a si $MUX(a)$

$v\_pred\ (a)$: vèrtex predecessor de l'arc

$v\_succ\ (a)$: vèrtex successor de l'arc

a_{ordre} : enter que indica l'ordre d'entrada de la transferència de dades representada per l'arc en el vèrtex successor de l'arc

Com que l'ordre d'entrada dels operands en els vèrtexs és important en considerar l'interconnexió, a_{ordre} indica quina entrada representa l'arc a l'operació representada pel vèrtex successor.

7.3.2 Llibreria de mòduls autotemporitzats

Els mòduls que formaran el camí de dades són mòduls autotemporitzats. Concretament s'ha optat pel model de mòduls en els quals els senyals de *petició* i *acabament* implementen un protocol de quatre fases realitzat amb lògica DCVSL.

La informació representada a la llibreria per cada tipus de recurs tr és la següent:

$\delta^{e}_{tr,op}$: retard necessari per executar l'operació op
en un recurs de tipus tr si l'operació està encadenada amb una altra

$\delta^{ne}_{tr,op}$: retard necessari per executar l'operació op
en un recurs de tipus tr si l'operació no està encadenada amb una altra

δ^{i}_{tr} : retard necessari per inicialitzar un recurs de tipus tr

a_{tr} : àrea que ocupa un recurs de tipus tr

7.3.2.1 Retard d'executar una operació

El temps d'executar una operació de tipus op en un recurs de tipus tr si l'operació no està encadenada ($\delta^{ne}_{tr,op}$), és el retard existent entre l'activació del senyal de *petició* i l'activació del senyal d'*acabament* si es compleixen les següents condicions:

- Les dades d'entrada de l'operació estan emmagatzemades en registres.
- Les dades d'entrada ja són vàlides quan s'activa el senyal de *petició*.

7.3.2.2 Retard d'executar dues operacions encadenades

Si la sortida d'una unitat funcional alimenta directament una entrada d'una altra unitat, direm que les unitats estan encadenades físicament. En aquest cas, podem encadenar l'execució de dues operacions.

Suposem dues operacions, totes dues de t[illegible], *op*, tals que el resultat de la primera és una dada d'entrada de la segona. Suposem, també, que les dues operacions s'executen en unitats funcionals de tipus *tr* i que ambdues unitats estan encadenades físicament. El temps d'executar dues operacions encadenades de tipus *op*, en dos recursos diferents de tipus *tr* físicament encadenats ($\delta^{2e}_{tr,op}$), és el retard existent entre l'activació del senyal de *petició* de la primera unitat i l'activació del senyal d'*acabament* de la segona unitat si es compleixen les següents condicions:

- Les dades d'entrada de la primera operació estan emmagatzemades en registres.
- Les dades d'entrada de la segona operació, excepte la que prové de la primera, estan emmagatzemades en registres.
- Les dades d'entrada de la primera i segona operació —excepte la que és el resultat de la primera— ja són vàlides quan s'activen els senyals de *petició*.

Generalment aquest retard és més petit que si suméssim dues vegades el retard d'executar les operacions si no estan encadenades, ja que a mesura que els resultats de la primera operació es van fent vàlids, la segona comença a calcular.

7.3.2.3 Retard d'executar una operació encadenada

El temps d'executar una operació de tipus *op* en un recurs de tipus *tr* si l'operació està encadenada ($\delta^{e}_{tr,op}$), es calcula de la següent manera:

$$\delta^{e}_{tr,op} = \delta^{2e}_{tr,op} - \delta^{ne}_{tr,op} \tag{7.12}$$

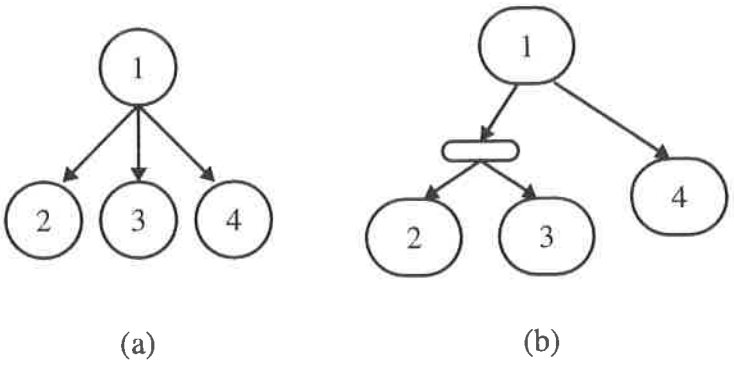


Figura 7.1: Exemple d'encadenament d'operacions: (a) Graf de flux de dades; (b) Graf de flux de dades planificat

7.4 Model d'execució

El programa pot utilitzar el nombre de registres i multiplexors. En aquesta secció es presenten diferents aspectes relacionats amb el model d'execució que s'utilitzarà en l'algorisme.

7.4.1 Encadenament de les operacions

El model d'execució, en aquest cas, permet l'encadenament d'operacions. D'aquesta manera, una operació pot llegir les dades d'entrada d'unitats funcionals o de registres i el resultat pot ser emmagatzemat en un registre o alimentar directament una altra unitat funcional.

En cas que una variable resultat sigui utilitzada per a més d'una operació, el resultat pot ser, d'una banda, emmagatzemat en un registre i, de l'altra pot alimentar directament una (o més) unitats funcionals. La figura 7.1 mostra un exemple d'un comportament d'aquest tipus. El resultat de l'operació 1 és utilitzat per tres operacions destí (2, 3 i 4). En la planificació, les dues primeres llegeixen aquesta variable d'un registre i l'altra està encadenada directament a l'operació 1.

7.4.2 Dependències estructurals

Les dependències estructurals són un conjunt d'arcs, E, que cal afegir al DFG per obtenir l'SDFG, $G = (V, A \bigcup E)$. Les dependències estructurals permeten establir l'ordre d'execució de les operacions associades a un mateix recurs. Durant l'execució de l'algorisme caldrà detectar aquestes dependències estructurals per poder decidir quina operació s'ha d'executar abans en cas de conflicte. En una secció posterior es descriu com determinar la direcció de les dependències estructurals entre vèrtexs en conflicte.

7.4.3 Esdeveniments en l'execució d'una operació

Si els mòduls que executaran les operacions són mòduls autotemporitzats, on els senyals de *petició* i *acabament* implementen un protocol de quatre fases per a cada operació, podem diferenciar els següents esdeveniments en l'execució d'una operació $v \in V$:

1. **Petició de càlcul:** transició positiva del senyal de *petició* ($p^+(v)$). Indica que la fase de càlcul de l'operació pot començar.

2. **Fi de càlcul**: transició positiva del senyal d'*acabament* ($a^+(v)$). Indica que el resultat de l'operació és estable.

3. **Petició d'inicialització**: transició negativa del senyal de *petició* ($p^-(v)$). Indica que la fase d'inicialització del mòdul pot començar.

4. **Fi d'inicialització**: transició negativa del senyal d'*acabament* ($a^-(v)$). Indica que les sortides del mòdul estan inicialitzades.

5. **Alliberació de recurs**: instant a partir del qual es pot tornar a activar el senyal de *petició*, és a dir, el mòdul queda alliberat per executar una altra operació ($ar(v)$).

A continuació es presenta quines són les dependències entre els esdeveniments d'una operació i els dels seus predecessors i successors si considerem mòduls autotemporitzats DCVSL com els del capítol 3. Suposarem que les operacions ja estan assignades al recurs en què seran executades.

Es fan servir estimacions del retard mitjà per als retards de les unitats funcionals per executar operacions i per inicialitzar-se.

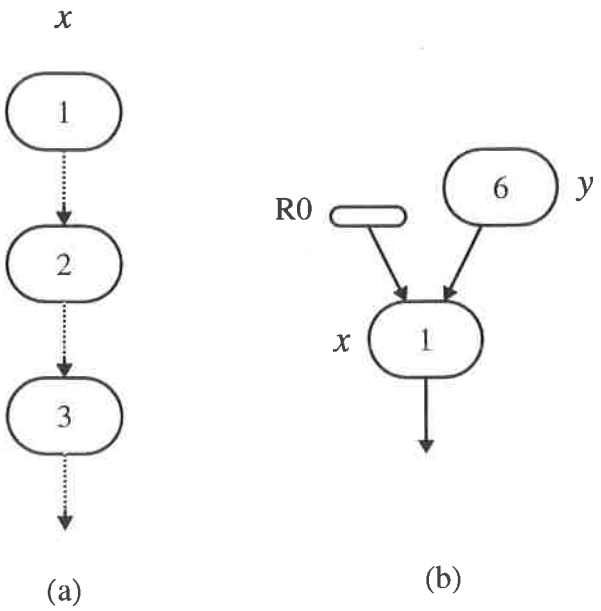


Figura 7.2: (a) Dependències estructurals entre les operacions assignades a un mateix recurs; (b) Tros de DFG planificat

7.4.3.1 Petició de càlcul ($p^+(v)$)

Aquest event pot produir-se si l'operació anterior executada en aquest mòdul ha alliberat el recurs. Per tant, cal que les dependències estructurals entre vèrtexs estiguin definides.

Per exemple, imaginem que volem avaluar quan pot produir-se l'execució de l'event *petició de càlcul* de l'operació representada pel vèrtex 2 de la figura 7.2. L'operació 2 ha estat assignada al recurs x i l'operació que s'executa abans en aquest recurs és l'operació representada pel vèrtex 1. En aquest cas, per poder començar l'operació 2, l'operació 1 ha d'haver alliberat el recurs.

7.4.3.2 Fi de càlcul ($a^+(v)$)

Per fer una estimació de quin instant es pot produir aquest event, cal tenir en compte si l'execució de l'operació està encadenada a altres i en quin moment les entrades són vàlides.

Si dues operacions estan encadenades, el temps d'execució serà probablement menor que si s'executéssin de manera independent, ja que a mesura que les dades de sortida de

l'operació predecessora són vàlides, poden ser utilitzades per l'operació successora. El fet que l'execució d'una operació estigui encadenada amb un altre depèn de dos factors:

- Que les unitats funcionals en què s'executen les operacions estiguin físicament encadenades (no hi ha cap registre enmig).
- Que l'execució de les operacions se solapi.

Si, per exemple, considerem les operacions de la figura 7.3, tenim que les unitats funcionals en les quals s'executen estan físicament encadenades, ja que el resultat de la primera operació alimenta directament la segona operació sense emmagatzemar-lo en un registre. Si l'execució de les operacions se solapa (tal com mostra la figura 7.3.b) direm que l'execució de les operacions està encadenada. Però si l'execució de les operacions no se solapa (figura 7.3.c), direm que l'execució de les operacions no està encadenada, encara que les unitats funcionals sí que ho estaven. Com que el temps d'acabament de la segona operació és desconegut (és el que estem calculant) per decidir en quin cas estem, ens fixarem en l'event *petició de càlcul.*

Definició 7.1 Execució encadenada

Dos vèrtexs, un predecessor de l'altre, $v_p, v_s \in V$, tals que les operacions que representen s'executen en unitats funcionals físicament encadenades, tenen la seva execució encadenada *si:*

$$p^+(v_p) + \varepsilon \geq p^+(v_s)$$

essent $p^+(v_p)$ i $p^+(v_s)$ els esdeveniments de petició de càlcul *del vèrtex predecessor i successor respectivament i ε és un retard positiu. ε és un paràmetre d'entrada del programa.*

El paràmetre ε depèn de la tecnologia i representa l'interval de temps que pot passar entre l'activació dels dos senyals de petició si volem que les operacions s'executin de manera encadenada (vegeu figura 7.3).

Definició 7.2 Encadenat(v)

Direm que un vèrtex v està encadenat, *si l'operació executada en v té l'execució encadenada amb alguna de les operacions dels vèrtexs predecessors de v.*

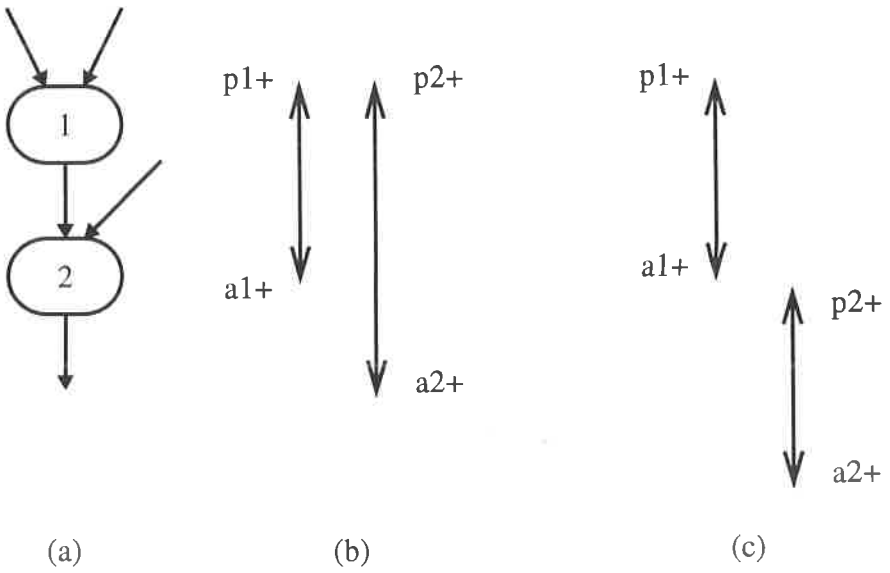


Figura 7.3: (a) Tros de Graf de flux de dades planificat; (b) Execució encadenada de les operacions 1 i 2; (c) Execució no encadenada de les operacions 1 i 2

En calcular el temps de l'event *fi de càlcul*, cal mirar si l'execució de l'operació està encadenada amb alguna de les operacions predecessores.

A continuació es mira en quin moment totes les dades d'entrada del vèrtex (v) són vàlides ($t_{ev}(v)$).

Definició 7.3 Temps d'entrades vàlides d'un vèrtex

El temps d'entrades vàlides d'un vèrtex v *és l'instant en què totes les dades d'entrada de l'operació executada en* v *ja són vàlides. Una dada d'entrada provinent d'un predecessor* $u \in \text{pred}(v)$ *és vàlida quan el resultat del vèrtex predecessor ja és vàlid, és a dir, quan l'operació ja ha acabat* ($a^+(u)$). *En general calcularem el* temps d'entrades vàlides d'un vèrtex v *amb la fórmula:*

$$t_{ev}(v) = \max_{u \in \text{pred}(v)} a^+(u) \tag{7.13}$$

Finalment, el càlcul de l'event *fi de càlcul* de l'operació executada en un vèrtex $v \in V$

es defineix com:

$$a^+(v) = \max(p^+(v),\ t_{ev}(v)) + \begin{cases} \delta^e_{v_f,v_o} & \text{si } encadenat(v) \\ \delta^{ne}_{v_f,v_o} & \text{altrament} \end{cases} \tag{7.14}$$

L'inici de l'operació pot ser, o bé l'event de petició d'inici de càlcul (si les entrades ja són vàlides) o bé el *temps d'entrades vàlides*. A partir d'aquest instant, sumem el retard de la unitat funcional per executar l'operació. Aquest retard pot tenir un valor o un altre segons si el vèrtex està encadenat o no.

7.4.3.3 Funció d'inicialització ($p^-(v)$)

La inicialització del mòdul pot començar quan totes les operacions dels vèrtexs successors de v han acabat els seus càlculs, és a dir:

$$p^-(v) = \max_{u \in succ(v)} a^+(u) \tag{7.15}$$

7.4.3.4 Fi d'inicialització ($a^-(v)$)

L'event de *fi d'inicialització* es produeix quan el mòdul ja s'ha inicialitzat. Podem estimar l'instant de temps en què es produirà aquest event amb la fórmula següent:

$$a^-(v) = p^-(v) + \delta^i_{v_f} \tag{7.16}$$

7.4.3.5 Alliberació de recurs ($ar(v)$)

Una operació executada en un vèrtex pot alliberar el recurs a què està assignat quan el fet que el recurs generi noves sortides vàlides no afecti a les operacions dels vèrtexs successors en el SDFG.

En cas que el vèrtex successor $u \in succ(v)$ representi una operació d'emmagatzemar en un registre, es podrà alliberar el recurs quan l'operació d'emmagatzemar hagi acabat ($a^+(u)$).

En cas que el vèrtex successor u representi una operació qualsevol diferent d'emmagatzemar, es podrà alliberar el recurs quan el mòdul que executa l'operació successora ja s'hagi inicialitzat ($a^-(u)$).

Definim el *temps en què no afecta si les entrades canvien*, $t_{ec}(u)$, per a un vèrtex $u \in V$, com:

$$t_{ec}(u) = \begin{cases} a^+(u) & \text{si } u_t = \text{VAR} \\ a^-(u) & altrament \end{cases} \tag{7.17}$$

En general, estimarem l'instant de temps en què es pot produir l'event *alliberació de recurs* amb la fórmula següent:

$$ar(v) = \max_{u \in succ(v)} t_{ec}(u) \tag{7.18}$$

En el cas que els successors no siguin variables emmagatzemades en registres, l'event d'alliberació de recurs es pot produir quan els successors dels successors han acabat el seu càlcul (a^+), tal com mostra la figura 7.4. Per aquest motiu, moltes condicions depenen del fet que hi hagi un camí de profunditat més gran o igual que 3, si no hi ha cap registre en el camí.

7.4.4 Direcció de les dependències estructurals

En aquesta secció es descriu com determinar el sentit dels arcs de dependències estructurals quan hi ha dos vèrtexs associats a un mateix recurs. Aquesta questió surgeix quan els dos vèrtexs no tenen cap relació de parentesc que determini quina operació s'ha d'executar primer.

Suposem dos vèrtexs, $v1, v2 \in V$, que estan associats a un mateix recurs , $v1_r = v2_r$. Existirà una dependència estructural entre $v1$ i $v2$, $(v1, v2)$ si es compleixen les condicions següents:

- Existeix un vèrtex $v3 \in DFG$ tal que $v1$ i $v2$ en són ascendents: $ascendent(v1, v3)$ i $ascendent(v2, v3)$).
- En el camí de $v2$ a $v3$ no existeix cap vèrtex $v4 \in V$ tal que $v4_t =$ `VAR` i $profunditat(v2, v3) < 3$.
- En el camí de $v1$ a $v3$ existeix un vèrtex $v4 \in V$ tal que $v4_t =$ `VAR` (figura 7.5.b) o $profunditat(v2, v3) \geq 3$ (figura 7.5.c).

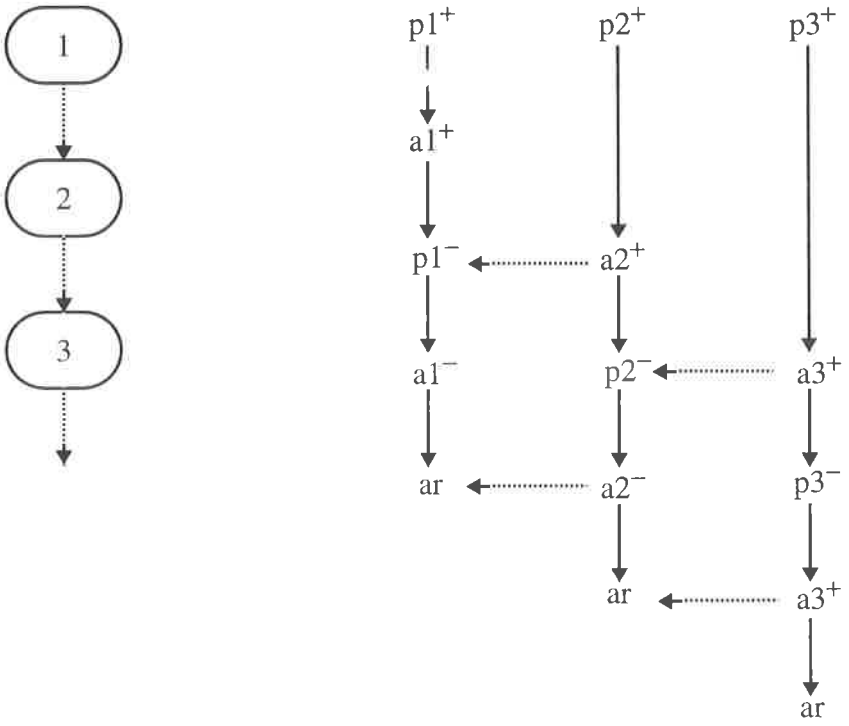


Figura 7.4: (a) Operacions amb execució encadenada; (b) Dependència entre els diferents esdeveniments

Per exemple, a la figura 7.5.b, podem veure que el resultat de l'operació del vèrtex 1 s'emmagatzema en el registre $R0$. En canvi, el resultat de l'operació del vèrtex 2 alimenta directament l'operació 3. S'ha d'executar primer l'operació del vèrtex 1, de manera que, un cop s'ha emmagatzemat el resultat en el registre, es pot utilitzar el recurs per executar la operació del vèrtex 2.

La figura 7.5.c presenta un cas equivalent. Encara que el resultat de l'operació del vèrtex 1 no s'emmagatzema en cap registre, atesa la capacitat d'emmagatzematge dels mòduls realitzats en lògica DCVSL, el recurs és alliberat pels seus successors i es pot executar la segona operació sense conflictes (vegeu secció anterior i figura 7.4.b).

En alguns casos, la dependència estructural no queda definida per cap d'aquestes condicions, com en el cas de la figura 7.5.a. En aquests casos es definirà la dependència segons les existents entre els vèrtexs predecessors o successors dels actuals.

7.5 Algorisme de planificació i assignació

A continuació es presenta l'algorisme de planificació i assignació que s'ha implementat. Primer es presenta una estructura de dades que s'utiltiza en el programa: la llista d'esdeveniments. A continuació, es descriu el mètode que s'ha seguit per construir la configuració inicial. Seguidament es presenten els elementes que personalitzen l'algorisme de recuit simulat per al problema concret que es vol resoldre: els moviments i la funció de cost. Finalment es fan alguns comentaris.

7.5.1 Llista d'esdeveniments

Durant l'execució de l'algorisme, es manté una llista ordenada dels diferents esdeveniments de cada operació. Aquesta llista està ordenada per temps, és a dir, segons l'ordre en qué estan planificades les operacions. Per a cada entrada de la llista es té la següent informació:

- Identificador del vèrtex o arc al que correspon l'event.
- Recurs associat al vèrtex.
- Tipus d'event (p^+, a^+ ...).

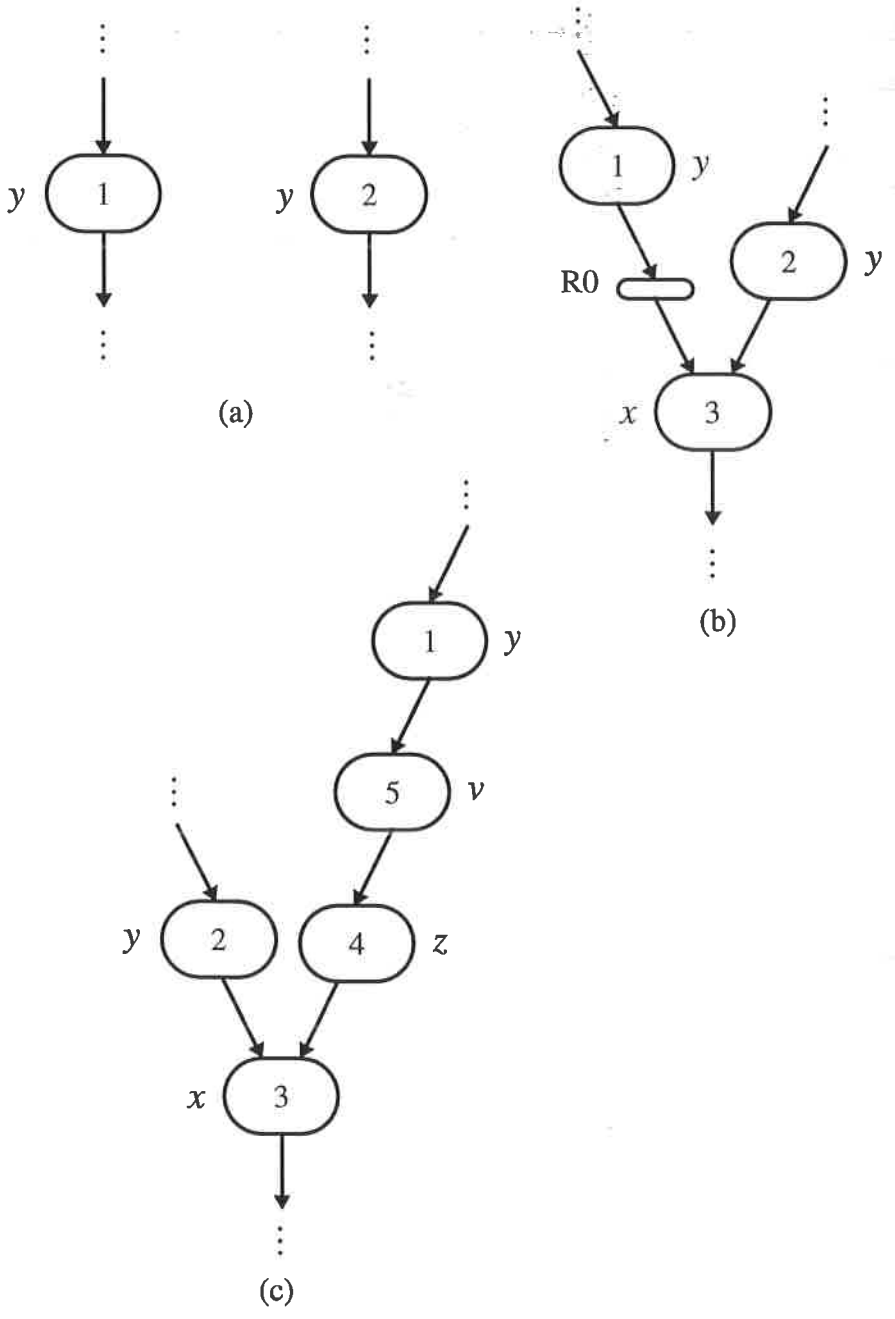


Figura 7.5: Dependències estructurals entre operacions associades al mateix recurs

- Temps estimat de planificació d'aquest event.

La llista d'esdeveniments serà consultada per alguns moviments i serveix per comprovar de forma fàcil algunes propietats de la planificació.

7.5.2 Configuració inicial

L'algorisme de recuit simulat parteix d'una configuració inicial qualsevol. Aquesta configuració inicial ha de definir tant la planificació de les operacions com l'associació de recursos. A continuació descriurem com es construeix la configuració inicial en el nostre cas, tot i que no és l'única opció.

Per simplificar la construcció de la configuració inicial, s'ha optat per un model que no permet l'encadenament d'operacions, és a dir, s'inserten registres a totes les entrades i sortides. Les dades d'entrada i de sortida del DFG també se suposen emmagatzemades en registres.

Els recursos en què s'executaran les operacions són triats aleatòriament. Això afecta a tots els tipus de recursos (unitats funcionals, registres, multiplexors).

Aleshores, la configuració és planificada seguint l'algorisme d'ASAP (vegeu capítol 2). Cal fer ressaltar que, atès que els arcs representen transferències de dades que podrien ser realitzades en multiplexors, els arcs del SDG també són planificats i tenen els seus corresponents esdeveniments. La figura 7.6 mostra un exemple de configuració inicial obtinguda en el cas de l'equació diferencial.

Un cop es té la configuració inicial, es construeix la llista d'esdeveniments. Per a cada vèrtex del graf i per a cada arc que tingui planificació, es crea una entrada a la llista d'event per a cadascun dels diferents esdeveniments de l'execució. La llista s'ordena per temps.

7.5.3 Moviments

Els moviments ens permeten canviar d'una configuració a configuracions veïnes, explorant l'espai de configuracions.

S'han definit quatre moviments que exploren diferents aspectes de les configuracions:

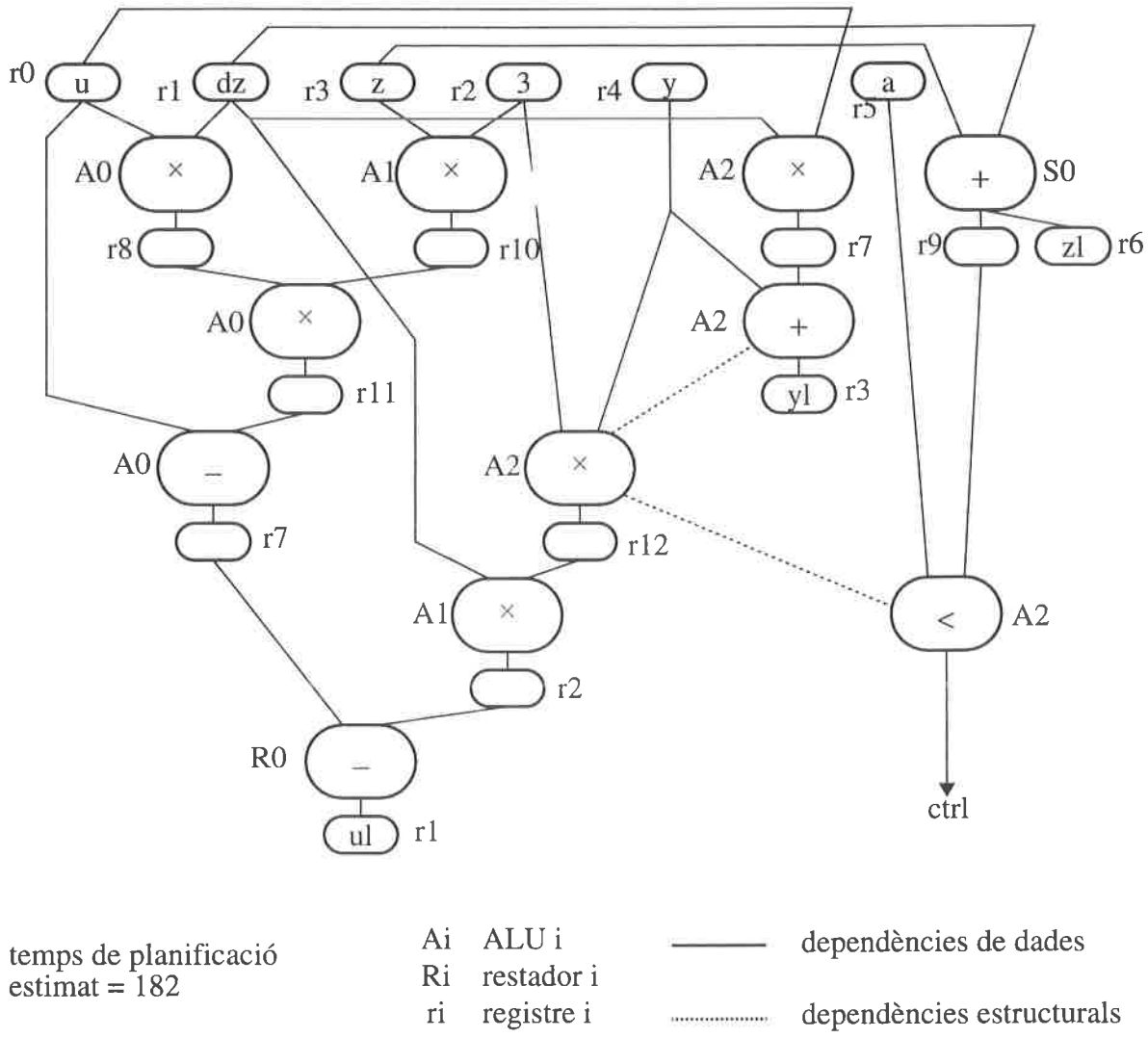


Figura 7.6: Exemple de configuració inicial obtinguda per a l'equació diferencial

1. **Reassociació de recurs**: explora l'espai de configuracions pel que fa a l'associació de recursos.

2. **Intercanvi d'operands**: explora l'espai de configuracions amb l'objectiu de minimitzar el cost d'interconnexió. Afecta bàsicament a l'associació de transferències a multiplexors.

3. **Insertar/Eliminar registre**: explora l'espai de configuracions pel que fa a l'encadenament o no d'operacions. Afecta bàsicament a la planificació d'operacions.

4. **Avançar/Retardar operació**: explora l'espai de configuracions pel que fa a la planificació d'operacions.

A continuació, es descriuen amb detall cadascun d'aquests moviments.

7.5.3.1 Reassociació de recurs

Aquest moviment s'aplica en general als vèrtexs del SDFG, tant si són operacions com variables emmagatzemades en registres.

Es tria aleatòriament el vèrtex $v \in V$ al qual s'aplicarà el moviment. També es tria aleatòriament el nou recurs que s'associarà a l'operació executada en el vèrtex.

El recurs es tria entre tot el conjunt de recursos que puguin executar l'operació amb algunes restriccions:

- **Cas en què** $v_t = \texttt{OP}$:

  1. Si existeix algun ascendent o descendent u, tal que qualsevol dels camins existents entre u i v tingui $profunditat(u, v) < 3$ i que en el camí no hi hagi cap vèrtex w tal que $w_t = \texttt{VAR}$, el nou recurs no pot ser igual a u_r. Si el camí entre u i v té profunditat més gran que 3, el recurs assignat a ja s'haurà alliberat, tal com s'ha vist a la secció 7.4.3.5. Igualment, si hi ha un registre insertat en el camí, el recurs assignat a quedarà lliure en emmagatzemar-se la informació en el registre.

2. Donat un descendent u tal que $profunditat(u, v) < 3$ i que no hi hagi cap vèrtex w en el camí tal que $w_t = \texttt{VAR}$, si existeix un ascendent de u, $v' \neq v$, tal que $profunditat(u, v) < 3$, el nou recurs no pot ser igual a v'_r.

- **Cas en què** $v_t = \texttt{VAR}$:

1. Si el vèrtex triat és una variable de sortida del graf, el recurs triat no pot estar associat a una altra variable de sortida del graf, ja que no podem emmagatzemar dues variables de sortida en un mateix registre.
2. Si el vèrtex triat és una variable d'entrada del graf, el recurs triat no pot estar associat a una altra variable d'entrada del graf.
3. Les mateixes condicions 1 i 2 del cas en què $v_t = \texttt{OP}$.

Un cop s'ha seleccionat el recurs, es canvia l'associació del vèrtex a aquest recurs i es replanifiquen els esdeveniments de l'operació executada en el vèrtex, excepte el de *petició de càlcul.*

Com a resultat d'aquest moviment, la configuració resultant pot ser *il·legal.* Direm que una configuració és *il·legal* si alguna dependència de dades o estructural és violada.

En aquest cas es poden produir violacions en tots dos tipus de dependències. Les violacions en les dependències de dades es poden produir atès que es canvia la replanificació del vèrtex escollit però no les dels vèrtexs descendents. Les dependències estructurals poden ser violades, ja que en replanificar el vèrtex no es té en compte els altres vèrtex associats al recurs triat.

Les configuracions il·legals són permeses ja que si es replanifiquessin tots els vèrtexs en cada moviment el cost del programa augmentaria molt. Les violacions són penalitzades a la funció de cost, tal com es veurà.

La figura 7.7 mostra un exemple d'aquest moviment. En aquesta figura els rectangles més petits representen variables emmagatzemades en registres i, les més grans, operacions associades a diferents recursos. L'alçada dels rectangles representa el retard dels recursos per realitzar els càlculs, és a dir, el retard existent entre l'ocurrència de l'event *petició de càlcul* i *fi de càlcul.* Els altres esdeveniments no es representen per no complicar la figura.

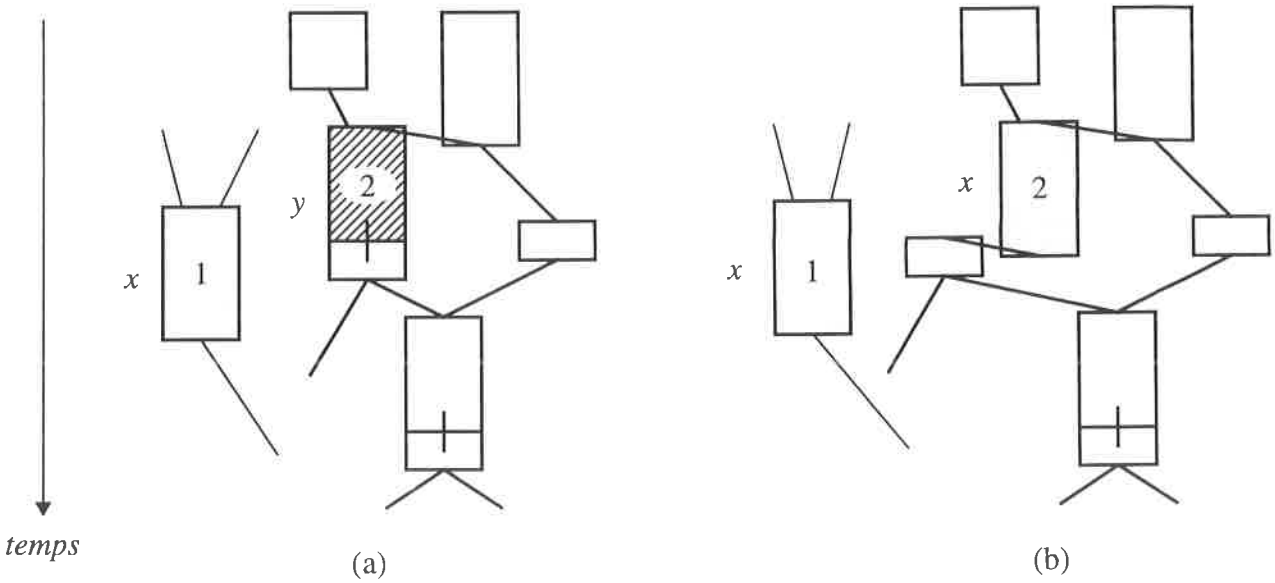


Figura 7.7: (a)

La figura 7.7.a mostra una part del SDFG abans d'aplicar el moviment. El vèrtex escollit és el vèrtex 2, que està associat al recurs y. El recurs que es tria per reassociar al vèrtex 2, és el recurs x, que ja està associat a altres vèrtexs, per exemple el vèrtex 1.

La figura 7.7.b mostra la part del SDFG després d'aplicar el moviment. Com que el retard del recurs x és més gran que el del y, es produeix una violació de la dependència existent entre el vèrtex 2 i el vèrtex que n'emmagatzema el resultat. Per resoldre aquesta violació, s'haurà de retardar l'execució de l'operació d'emmagatzematge. A més, com que la planificació del vèrtex 1 se solapa amb la del vèrtex 2, tenim una violació d'una dependència estructural. Aquesta violació es resol posposant la planificació d'un dels vèrtexs.

7.5.3.2 Intercanvi d'operands

Aquest moviment s'aplica als vèrtexs del SDFG que representen operacions commutatives. Aquesta propietat permet intercanviar l'ordre dels operands de l'operació sense que el resultat es vegi alterat. Les operacions unàries són considerades no commutatives.

Es tria un vèrtex $v \in V$ d'entre els que representen operacions commutatives, de manera aleatòria. A continuació es trien dos operands d'entre els que tingui l'operació, és a dir, es

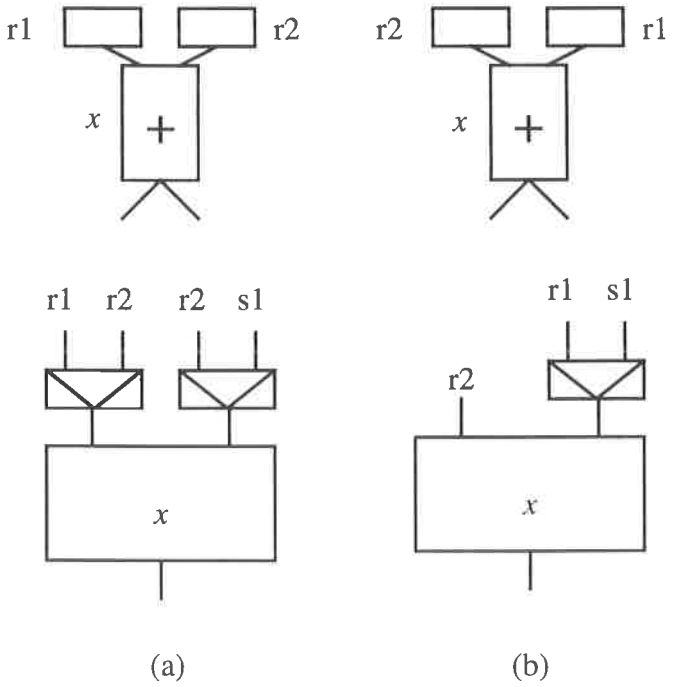


Figura 7.8: Exemple de moviment intercanvi d'operands

trien dos arcs $a1, a2 \in A$ incidents en el vèrtex.

S'intercanvia l'ordre d'entrada de les dades representades pels arcs ($a1_{ordre}$ i $a2_{ordre}$), així com les informacions referents a l'associació de recursos ($a1_r$, $a1_f$, $a2_r$ i $a2_f$).

Com a conseqüència d'aquests canvis, pot canviar el valor de les funcions $MUX(a1)$ i $MUX(a2)$, de manera que el cost d'interconnexió pot variar. En cas que el valor de $MUX(a1)$ sigui *cert* es replanifica l'arc $a1$. El mateix passa amb l'arc $a2$.

La figura 7.8 mostra un exemple d'execució d'aquest moviment i les seves possibles conseqüències sobre el cost d'interconnexió. En aquest cas, el vèrtex a triar ha de representar una operació commutativa, per exemple una suma. En aquest cas, el fet d'intercanviar l'ordre d'entrada dels operands reduiria el cost d'interconnexió en un multiplexor de dues entrades.

De nou, la configuració resultant pot resultar il·legal.

7.5.3.3 Insertar/Eliminar registre

Aquest moviment permet l'encadenament o no de les operacions de l'SDFG. S'aplica sobre els arcs $a \in A$ de l'SDFG, ja que són aquests els que representen les transferències de dades, o variables.

El primer que es decideix, de manera aleatòria, és si es farà una eliminació de registre o una inserció. Segons quin sigui el cas, el comportament del moviment és diferent.

Insertar registre

Es tria un arc $a = (u, v) \in A$ aleatòriament, tal que $u_t = v_t = \texttt{OP}$, és a dir, un arc que uneixi dues operacions encadenades. Segons la topologia del graf, el comportament del moviment és diferent. Si existeix un arc tal que el seu vèrtex predecessor és u i el seu vèrtex successor w és tal que $w_t = \texttt{VAR}$, aleshores s'elimina l'arc a i s'afegeix un arc $a' = (w, v)$. És a dir, com que ja existia un registre que emmagatzemava el resultat de l'operació representada pel vèrtex u, no cal afegir-ne cap altre. El que canvia és que ara l'operació del vèrtex v llegeix la variable d'entrada del registre w.

Si no existeix cap vèrtex que emmagatzemi el resultat de u, cal insertar un vèrtex r entre els dos vèrtexs u i v. Aquest vèrtex r serà tal que $r_t = \texttt{VAR}$, és a dir, representarà una operació d'emmagatzematge. A continuació es tria de manera aleatòria el registre que s'associarà a l'operació del vèrtex r. En aquesta tria hi ha algunes restriccions, que són les mateixes que si el vèrtex representés una variable en el moviment de reassociació.

En tots dos casos es replanifiquen els vèrtexs u, v i el vèrtex que w o r, segons el cas. Si els arcs existents entre ells són transferències a través de multiplexors, també són replanificats.

El fet que aquest moviment inserti registres en els diferents arcs de manera independent permet d'una banda, que el resultat d'una operació sigui emmagatzemada en un registre i, de l'altra, que alimenti directament l'entrada d'una altra operació, permetent-ne l'encadenament.

La figura 7.9.b mostra un exemple d'inserció d'un registre entre dues operacions. La figura 7.9.c mostra el comportament del moviment si posteriorment s'aplica sobre un altre arc que transfereix la mateixa variable.

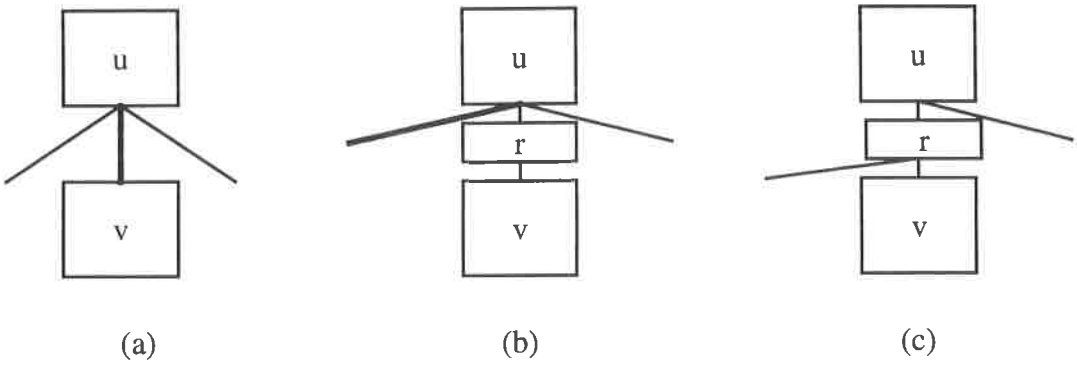


Figura 7.9: Exemple de moviment insertar registre

Eliminar registre

Es tria un arc $a = (r, v)$ tal que r_t = VAR, és a dir, un arc que uneix una variable amb una operació. El vèrtex r ha de ser tal que existeixi un arc (u, r). Aquesta tria té també algunes restriccions ja que, per exemple, no poden quedar dues operacions encadenades assignades al mateix recurs. Aquestes restriccions consisteixen a comprovar que, un cop eliminat el registre, no es produeix cap dels casos descrits en el moviment de reassociar quan el vèrtex representa una operació. S'elimina l'arc a i s'afegeix un arc a' que uneixi el vèrtex u amb el v. Si com a resultat d'aquests canvis el vèrtex r queda sense cap successor ($|succ(r)| = 0$), s'elimina el vèrtex r del graf i l'arc que uneix u amb r.

Després es replanifiquen els vèrtexs u i v i l'r si no s'ha eliminat, així com els arcs que els connecten si representen transferències de dades a través de multiplexors. Com en els moviments anteriors, la configuració resultant pot ser il·legal.

7.5.3.4 Avançar/Retardar operació

Aquest moviment s'aplica sobre vèrtexs de l'SDFG, tant si representen operacions com variable. Els únics vèrtexs dels quals no es permet canviar la planificació són els que representen variables d'entrada del graf, és a dir, els vèrtexs *font* del graf.

Primer es tria el vèrtex al qual s'aplicarà el moviment. Com sempre, aquesta tria es fa de manera aleatòria. A continuació es tria el nombre m d'esdeveniments en què es retardarà o avançarà la planificació del vèrtex. m és el nombre d'esdeveniments que anirem endavant

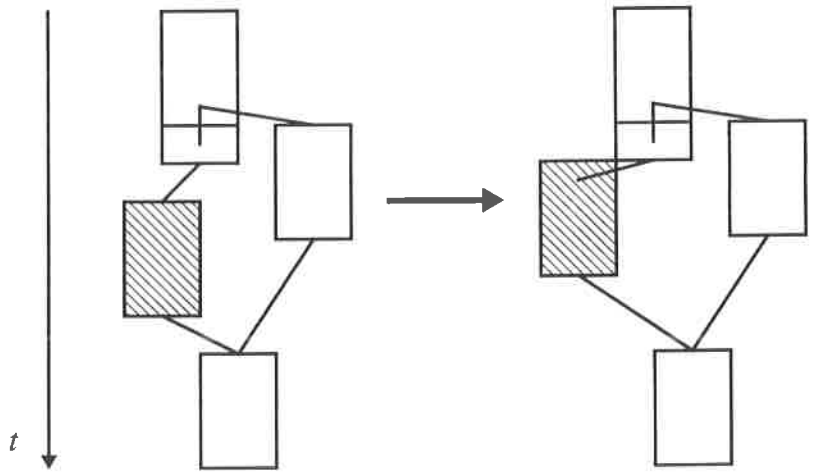


Figura 7.10: Exemple de moviment retardar vèrtex

o enrere a la llista d'esdeveniments. El valor màxim d'm (M) és un paràmetre d'entrada del programa i el nombre mínim és 1. m es tria de manera aleatòria entre 1 i n. n depèn de la temperatura i es calcula amb la següent fórmula:

$$n = \left\lceil \frac{T}{T_o} M \right\rceil \tag{7.19}$$

essent T_o la temperatura inicial i T la temperatura actual. Així doncs, inicialment, el nombre d'esdeveniments en què es retarda o avança la planificació del vèrtex és potencialment més gran que quan la temperatura es baixa.

Es decideix de manera aleatòria si s'avançarà o retardarà la planificació del vèrtex. Per saber quin és el nou inici de l'operació, ens movem en la llista d'esdeveniments m posicions, endavant o enrere, segons quin sigui el cas. El temps de l'event fixarà el nou temps de planificació de la *petició de planificació* del vèrtex. La resta d'esdeveniments del vèrtex es calculen en funció d'aquest.

Com que no es replanifiquen els altres vèrtexs, la configuració resultant pot ser il·legal. La figura 7.10 mostra un exemple del moviment d'avançar un vèrtex.

7.5.4 Funció de Cost

La funció que avalua el cost de les configuracions es defineix amb la següent fórmula:

$$C = \alpha \times temps + \beta \times àrea + \gamma \times penalització \tag{7.20}$$

El terme *temps* avalua el cost de la configuració pel que fa al temps estimat de planificació. El terme *àrea* avalua el cost en àrea de la configuració. Finalment, el terme *penalització* avalua el grau d'il·legalitat de les configuracions.

α, β i γ són paràmetres d'entrada del programa que ens permeten guiar-lo cap al tipus de solucions desitjades. Per exemple, si donem un valor alt a α i un de baix a β, el temps de la planificació serà més penalitzat que l'àrea, de manera que el programa intentarà optimitzar les configuracions en temps de planificació més que en àrea.

El factor γ ens permet penalitzar amb més o menys grau les configuracions il·legals. Si el valor de γ és alt, l'algorisme tindrà més llibertat per exeplorar l'espai de disseny, però tindrem el risc d'obtenir una configuració il·legal. En canvi, si el valor de γ és alt, les configuracions il·legals estaran més penalitzades, de manera que correm el risc que algunes configuracions il·legals que ens portarien a una solució bona, no siguin acceptades.

7.5.4.1 Àrea

Atès que el programa fixa com a restricció d'entrada el nombre d'unitats funcionals de cada tipus a utilitzar en la solució final, el cost en àrea es calcula en funció del cost d'interconnexió i de registres.

El cost d'interconnexió s'avalua com el nombre de multiplexors necessaris multiplicats per l'àrea del multiplexor. Per això cal mirar quines transferències de dades es fan a través de multiplexors (funció *MUX*) i comptar el nombre d'entrades que té cada multiplexor necessari. El nombre de multiplexors es calcula com el nombre de multiplexors de dues entrades necessaris. Per obtenir una estimació de l'àrea que ocuparien els multiplexors es multiplica aquest nombre per l'àrea d'un multiplexor de dues entrades.

El cost dels registres s'avalua com el nombre de registres necessaris a la configuració multiplicat per l'àrea d'un registre.

7.5.4.2 Temps

El cost en temps s'avalua com el temps estimat de la planificació, si les operacions representades pels vèrtexs s'executen en el temps mitjà definit per la llibreria. Aquesta estimació es fixa amb el temps de l'últim event de la llista d'esdeveniments.

7.5.4.3 Penalització

El terme de penalització avalua el grau d'il·legalitat de les configuracions en tots dos tipus de violacions:

- Violacions de les dependències de dades.
- Violacions de les dependències estructurals.

Les violacions de les dependències de dades s'avaluen per a cada vèrtex $v \in V$. Es calcula el temps *legal* de l'event *fi de càlcul* del vèrtex v si es tenen en compte les dependències de dades amb els seus predecessors ($a_l^+(v)$). Es calcula la diferència entre aquest valor i el valor al qual s'ha planificat l'event *fi de càlcul* ($a^+(v) - a_l^+(v)$). Si la diferència és positiva, no hi ha violació de la dependència, ja que l'event està planificat més tard del mínim que fixen les precedències. En canvi, si la diferència és negativa, si que s'està violant la dependència i s'acumula aquest valor sobre el terme *penalització*.

Les violacions de les dependències estructurals s'avaluen mirant si la planificació utilitza un mateix recurs per dos vèrtexs en un mateix instant de temps. Per quantificar aquesta violació, en cas que dos vèrtexs associats a un mateix recurs tinguin la planificació solapada, es calcula l'interval de temps en què estan solapades les seves planificacions. D'aquesta manera, la penalització és proporcional al grau de solapament. Aquesta informació s'obté consultant la llista d'esdeveniments.

7.5.5 Comentaris

En aquesta secció es presenten alguns aspectes concrets de la realització del programa que s'ha fet.

Quan un moviment és refusat, hem de tornar a la configuració anterior, per tant, cal establir un mecanisme que ens permeti tornar a aquesta. Podríem optar entre diferents

	sumador	ALU	restador	registre	multiplexor
+	15	20	∞	∞	∞
−	∞	25	20	∞	∞
×	∞	35	∞	∞	∞
<	∞	25	∞	∞	∞
emmagatzemar	∞	∞	∞	3	∞
transferència	∞	∞	∞	∞	1

Taula 7.1: Matriu de retards dels recursos per executar una operació no encadenada

alternatives, com per exemple, duplicar la informació, de manera que tindríem la configuració antiga i la nova, o definir els moviments inversos. En el nostre cas, s'ha optat per guardar informació sobre les modificacions que s'efectuïn en les diferents estructures de dades, i en cas que el moviment sigui refusat restaurar els canvis.

Si la configuració final és una configuració il·legal (existeix alguna violació de dependència), es transformada en legal per un procés posterior. Aquest procés el que fa és posposar aquelles operacions que violen alguna dependència fins que no ho facin. El procés es repeteix mentre quedi alguna operació en estat il·legal.

El programa s'ha escrit en lleguatge C i ocupa un total d'11.000 línies de codi.

7.6 Resultats

Aquesta secció presenta alguns resultats obtinguts amb el programa per diferents exemples: l'equació diferencial [PK89a], el filtre AR-lattice [JMP88] i el filtre el·líptic de cinquè ordre [DDN85].

7.6.1 Equació diferencial

Les taules 7.1, 7.2, 7.3 i 7.4 descriuen les característiques de la llibreria de recursos utilitzada per aquest exemple. A la taula 7.1, es descriu per a cada recurs i cada operació, el retard mitjà esperat del recurs per executar l'operació no encadenada ($\delta^{ne}_{tr,op}$). Si el retard

	sumador	ALU	restador	registre	multiplexor
+	10	15	∞	∞	∞
−	∞	20	15	∞	∞
×	∞	25	∞	∞	∞
<	∞	20	∞	∞	∞
emmagatzemar	∞	∞	∞	3	∞
transferència	∞	∞	∞	∞	1

Taula 7.2: Matriu de retards dels recursos per executar una operació encadenada

sumador	ALU	restador	registre	multiplexor
5	5	5	3	1

Taula 7.3: Vector de retards dels recursos per inicialitzar-se

sumador	ALU	restador	registre	multiplexor
42	168	42	10	14

Taula 7.4: Vector de l'àrea dels recursos

Temps	#multiplexors 2:1	#registres	Àrea	α	β	γ	temps CPU
145	4	12	806	100	1	10	5,0 min
156	8	10	842	5	1	2	6,4 min
178	6	10	814	100	5	10	4,6 min

Taula 7.5: Resultats de l'equació diferencial

val ∞, significa que l'operació no pot ser executada en aquest tipus de recurs. A la taula 7.2 es descriu, per a cada recurs el retard d'executar l'operació de manera encadenada ($\delta^e_{tr,op}$). A la taula 7.3, es descriu el retard de cada tipus de recurs per inicialitzar-se (δ^i_{tr}) i a la taula 7.4, l'estimació de l'àrea que ocupa el tipus de recurs (a_{tr}).

7.6.1.1 Primer exemple

En aquest cas s'han fixat les restriccions d'entrada a tres ALU, un sumador i un restador. Els resultats obtinguts es mostren a la taula 7.5. Per a cada solució es descriu el temps estimat de la planificació, el nombre de multiplexors 2:1 equivalents que són necessaris pel camí de dades, el nombre de registres utilitzats i una estimació de l'àrea ocupada pel camí de dades. En aquest cas, els valors dels paràmetres α, β i γ varen ser diferents. La taula mostra també els valors d'aquests paràmetres en cada cas.

Per fer l'estimació de quina àrea ocupa cada tipus de recurs, s'ha utilitzat una llibreria en DCVSL dissenyada per a una tecnologia *sea-of-gates*.

La figura 7.11 mostra una de les planificacions i assignacions obtingudes amb aquesta restricció de recursos. S'ha representat el graf de flux de dades i per cada operació s'indica el recurs que se li ha assignat. També s'indica si el resultat de l'operació s'emmagatzema en registre, i en cas afirmatiu, en quin registre.

Les línies discontínues representen dependències estructurals entre operacions associades al mateix recurs. No s'han dibuixat les dependències estructurals existents entre registres ni els multiplexors existents per no complicar la figura. Podem apreciar com el programa encadena algunes operacions per tal de minimitzar el temps de planificació. Per exemple, s'encadenen les diferències i una de les multiplicacions del final del camí crític.

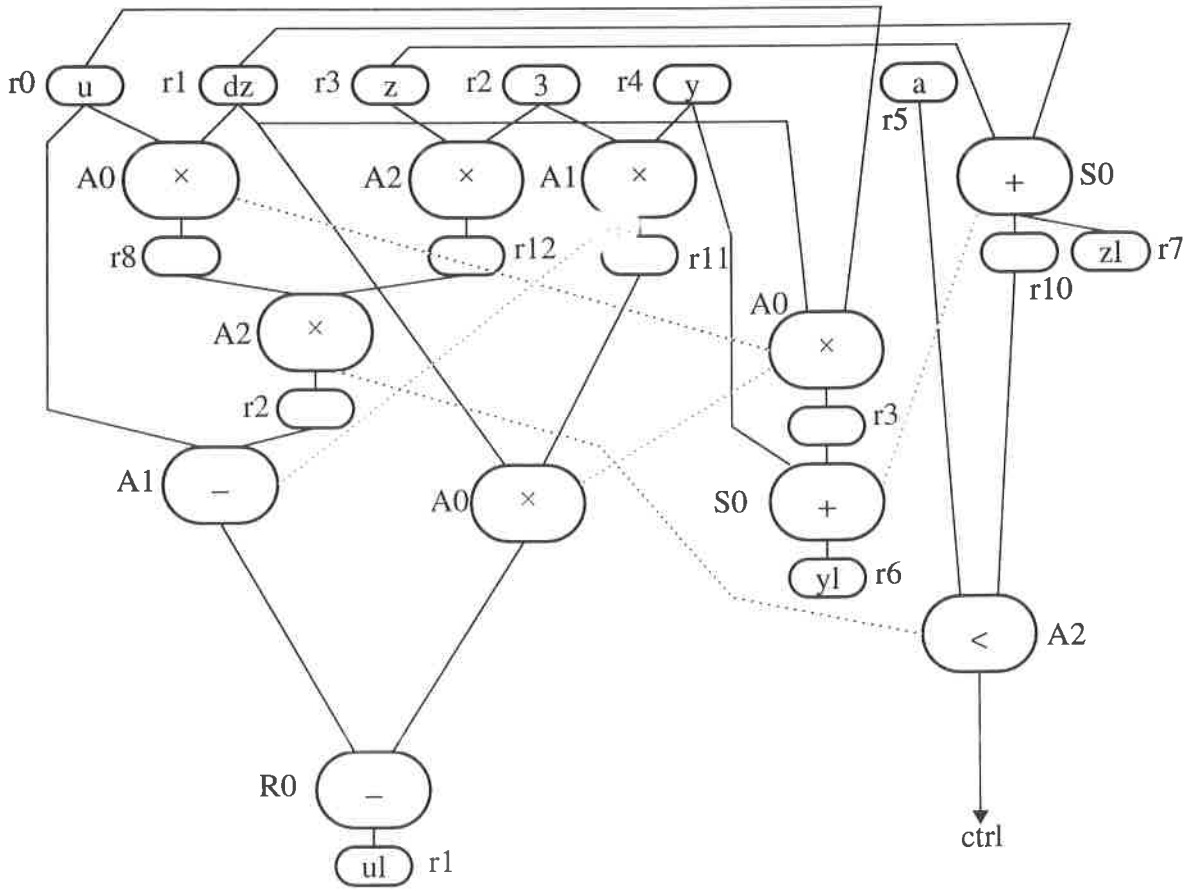


Figura 7.11: Configuració obtinguda per l'equació diferencial utilitzant tres ALU, un sumador i un registre

Temps	#multiplexors 2:1	#registres	Àrea	temps CPU
248	9	13	718	8,4 min
252	9	11	698	6,2 min
294	6	11	656	5,7 min

Taula 7.6: Resultats de l'equació diferencial

7.6.1.2 Segon exemple

En aquest apartat es presenten els resultats obtinguts pel programa si es fixa una restricció de recursos de dos ALU, un sumador i un restador. En aquest cas els diferents resultats s'han obtingut fixant els paràmetres α, β i γ a un mateix valor (α=100, β=1 i γ=10). Podem veure com el programa busca un compromís entre el temps estimat de la planificació i l' àrea total ocupada pel camí de dades. Així, la solució amb una estimació de temps de la planificació més gran és la que té una estimació d'àrea més petita i la que té una estimació de temps més petita és la que ocupa més àrea.

7.6.2 Filtre AR-lattice

En aquesta secció presentarem els resultats obtinguts per al filtre *AR-lattice*. El graf de flux de dades ha estat modificat de manera que no apareixen les multiplicacions. Aquesta modificació s'ha fet sota la suposició que les multiplicacions són per potències de 2 i, per tant, poden ser realitzades en temps 0 (amb desplaçaments). La figura 7.12 mostra el DFG d'aquest filtre amb les modificacions.

La llibreria utilitzada en aquest cas és la mateixa que en el cas anterior (taules 7.1, 7.2, 7.3 i 7.4).

7.6.2.1 Primer exemple

En aquest apartat es descriuen els resultats obtinguts per aquest exemple si es fixen les restriccions de recursos a quatre sumadors.

La taula 7.7 mostra els resultats que s'han obtingut sota aquestes condicions. Es pot

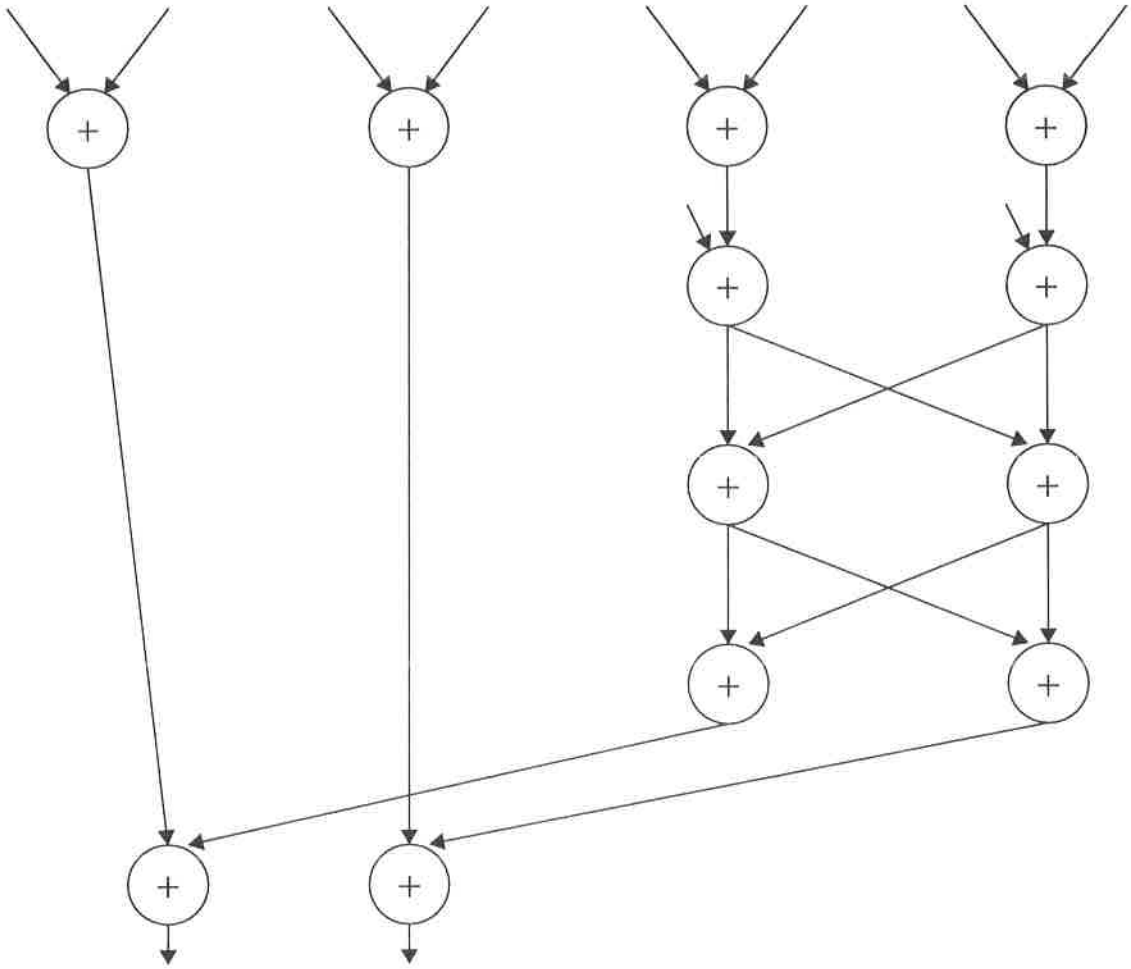


Figura 7.12: Graf de flux de dades del filtre AR-lattice *sense multiplicacions*

Temps	#multiplexors 2:1	#registres	Àrea	α	β	γ	CPU
131	14	10	464	100	1	10	5,6 min
133	11	10	422	100	1	10	7,3 min
143	11	9	412	10	1	100	5,7 min
162	10	9	398	2	1	10	4,6 min

Taula 7.7: Resultats per al filtre AR-lattice *utilitzant quatre sumadors*

Temps	#multiplexors 2:1	#registres	Àrea	α	β	γ	CPU
73	8	10	506	2	1	10	6,7 min
79	6	9	468	10	1	100	5,4 min
87	6	10	478	10	1	10	5,1 min
95	6	10	478	2	1	10	5,7 min
101	6	9	468	1	10	10	5,2 min

Taula 7.8: Resultats per al filtre AR-lattice *utilitzant set sumadors*

observar que si fixem el valor d'α a valors més alts que β s'obtenen resultats millors en temps, però pitjors en termes d'àrea. En canvi, si fixem el valor d'α a valors més baixos, s'obtenen solucions amb una àrea més petita, encara que amb un temps de planificació més gran.

7.6.2.2 Segon exemple

En aquest apartat es descriuen els resultats obtinguts per aquest exemple si es fixen les restriccions de recursos a set sumadors. En aquest cas, tot i que no es tant clar com en l'anterior, veiem com l'àrea disminueix a mesura que la solució obtinguda requereix un temps de planificació anterior.

La figura 7.13 mostra una de les planificacions obtingudes en la que es pot apreciar clarament com el programa encadena les operacions del camí crític per obtenir un temps de planificació inferior. Això és possible ja que s'ha fixat el nombre de recursos disponibles

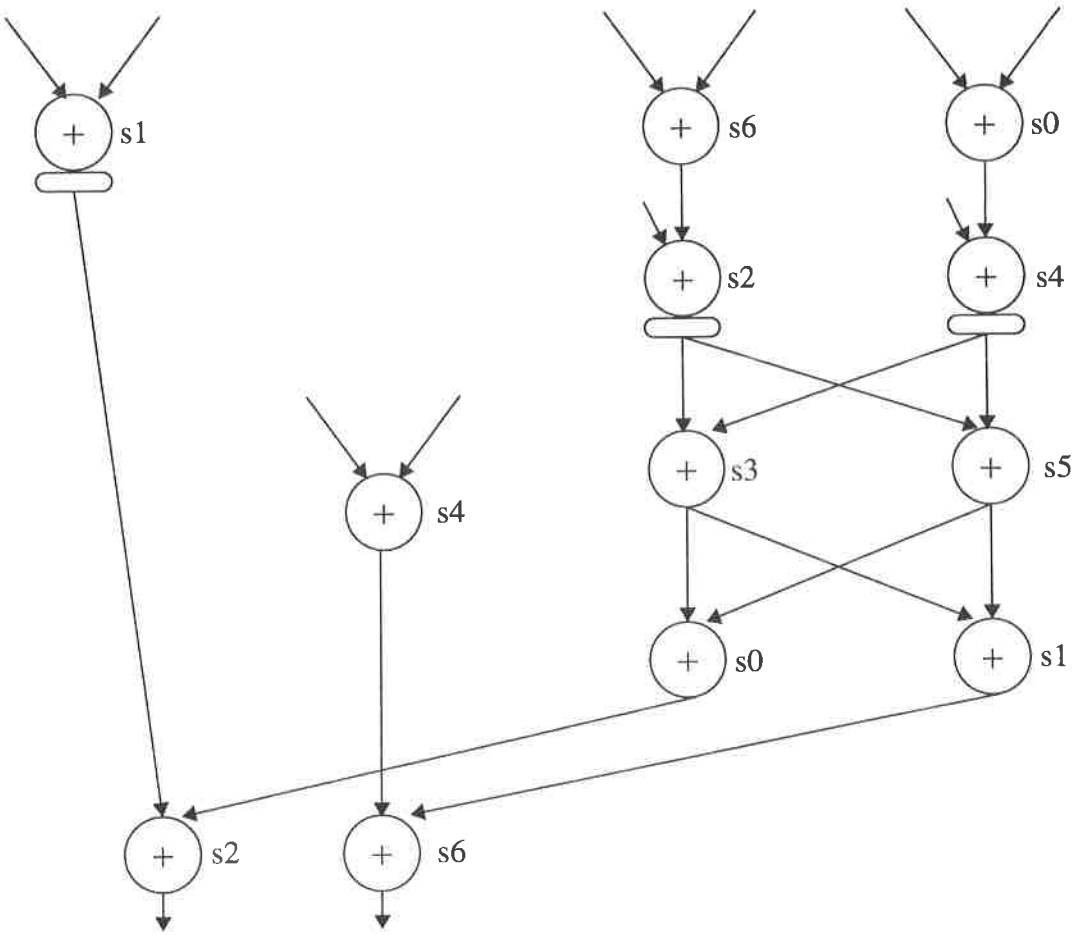


Figura 7.13: Planificació obtinguda per al filtre AR-lattice *si es fixen les restriccions a set sumadors*

a un nombre força gran. El temps de planificació estimat per a aquesta solució és de 73 ns.

7.6.3 Filtre el·líptic

A continuació es presenten els resultats obtinguts amb el programa per al *filtre el·líptic de cinquè ordre* [DDN85]. En aquest cas s'ha modificat lleugerament el DFG, ja que s'han eliminat les multiplicacions. S'ha considerat que les multiplicacions són per potències de 2 i que es poden executar per *hardware* en temps 0 (amb desplaçaments). La llibreria utilitzada és la que mostren les taules 7.9, 7.10, 7.11 i 7.12.

	sumador	registre	multiplexor
+	22	∞	∞
emmagatzemar	∞	10	∞
transferència	∞	∞	11

Taula 7.9: Matriu de retards dels recursos per executar una operació no encadenada

	sumador	registre	multiplexor
+	17	∞	∞
emmagatzemar	∞	7	∞
transferència	∞	∞	7

Taula 7.10: Matriu de retards dels recursos per executar una operació encadenada

sumador	registre	multiplexor
11	8	8

Taula 7.11: Vector de retards dels recursos per inicialitzar-se

sumador	registre	multiplexor
42	10	14

Taula 7.12: Vector de l'àrea dels recursos

Temps	#multiplexors 2:1	#registres	Àrea
456	33	19	862
473	33	18	852
486	33	18	852
499	32	17	828
500	31	16	804
538	21	29	794

Taula 7.13: Resultats per al filtre el·líptic utilitzant cinc sumadors

A continuació es presenten els resultats que s'han obtingut fixant diferents restriccions de recursos.

7.6.3.1 Resultats utilitzant cinc sumadors

La taula 7.13 mostra els resultats obtinguts per al filtre el·líptic quan es fixa la restricció de recursos a cinc sumadors. Aquests resultats han estat obtinguts fixant $\alpha = 2$, $\beta = 1$ i $\gamma = 10$. El temps de CPU per aquest exemple varia entre 15 i 31 minuts.

Com en els exemples anteriors, el programa intenta buscar un compromís entre el temps de planificació i l'àrea requerida pel camí de dades. Així, podem veure com les planificacions amb un cost d'àrea més gran en contrapartida són més ràpides.

La figura 7.14 mostra una configuració final obtinguda pel programa. La figura conté informació sobre la planificació i assignació. Els requadres amb diferent color representen operacions assignades a recursos diferents. L'alçada del requadre representa el temps transcorregut des que s'ha activat el senyals de petició fins que el senyal d'acabament és actiu. Per aquest motiu les alçades dels requadres són diferents, ja que encara que s'activi el senyal de petició, si les entrades no són vàlides el càlcul no comença. En el cas d'operacions encadenades s'ha dibuixat un requadre més petit, que representa el temps transcorregut des de la transició positiva del senyal d'acabament de l'operació predecessora fins a la transició positiva del senyal d'acabament de la propia operació.

La figura mostra també els intervals d'ocupació de cada operació en els diferents suma-

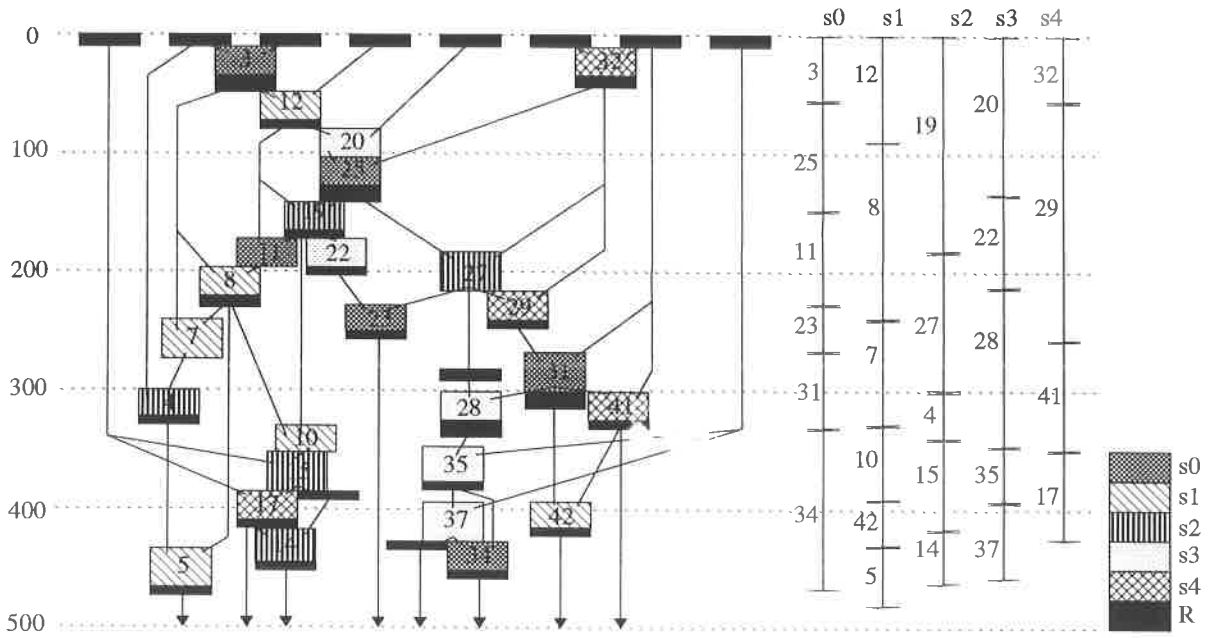


Figura 7.14: Configuració obtinguda per al filtre el·líptic si es fixen les restriccions a cinc sumadors

dors ($s0 \ldots s4$). Els requadres negres representen variables emmagatzemades en registres. El temps estimat d'aquesta planificació és de 473 ns. Pel que fa a recursos, utilitza 33 multiplexors de dues entrades i 18 registres.

Es pot apreciar com el programa intenta encadenar les operacions per tal de minimitzar el temps de planificació. Per exemple, encadena les operacions 20 i 25, que són del camí crític, i també l'11 amb la 8.

7.6.3.2 Resultats utilitzant vuit sumadors

La taula 7.14 mostra els resultats obtinguts per al filtre el·líptic utilitzant una restricció de recursos de vuit sumadors. En utilitzar més unitats funcionals permet al programa encadenar més les operacions i obtenir un temps de planificació més petit. El cost en àrea d'aquestes solucions, tot i que el nombre d'unitats funcionals s'ha doblat, no és molt més gran que l'obtingut a la secció anterior. Això es degut a que el cost en àrea dels multiplexors i registres és molt gran en comparació al de les unitats funcionals. Si comparem

Temps	#multiplexors 2:1	#registres	Àrea	α	β	γ
383	36	20	1040	5	1	10
398	33	20	998	2	1	2
436	32	20	984	2	1	2
443	30	20	956	2	1	2

Taula 7.14: Resultats per al filtre el·líptic utilitzant vuit sumadors

la solució més ràpida obtinguda amb quatre sumadors amb la més ràpida obtinguda amb vuit sumadors, veiem que per un increment en l'àrea del 21% obtenim una millora en el temps de planificació del 16%.

7.7 Conclusions

La planificació d'operacions i l'assignació de recursos són les fases més importants de la síntesi d'alt nivell. Aquestes dues fases poden executar-se una abans de l'altra —per exemple, *ELS* o *ELLAS* i després *GLASS*— o simultàniament, tal com s'ha proposat en aquest capítol. La figura 7.15 mostra l'esquema d'un sistema de síntesi d'alt nivell. En ell, l'algorisme de planificació i assignació que proposem s'executaria després de la fase de traducció i abans de la síntesi lògica. L'algorisme presentat resol aquestes dues fases utilitzant com a base la tècnica d'escalada de cims anomenada recuit simulat. El recuit simulat és un algorisme genèric de tipus iteratiu que permet trobat solucions properes a l'òptim.

El model d'execució utilitzat en aquest cas és més complex que el presentat al capítol 5. Les operacions poden ser encadenades, de manera que poden llegir les variables d'entrada d'altres unitats funcionals i els resultats poden no ser emmagatzemats en registres. Aquesta característica, juntament amb el fet que es tracten recursos d'una llibreria de mòduls autotemporitzats, complica el càlcul del temps de vida de les operacions.

S'ha suposat que els mòduls que compondran la llibreria seran mòduls construits en lògica DCVSL. Aquests mòduls es caracteritzen, entre altres coses, pel fet que l'execució

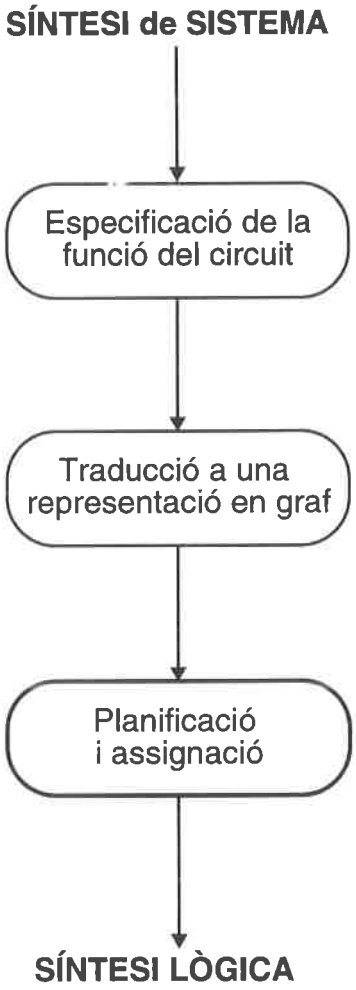


Figura 7.15: Esquema d'un sistema de síntesi d'alt nivell

d'una operació en un component d'aquest tipus es compon de tres fases: fase de càlcul, fase de memorització i fase d'inicialització.

En el capítol s'ha presentat com calcular l' inici i finalització de cadascuna d'aquestes fases, tenint en compte el model d'execució utilitzat.

L'algorisme de recuit simulat parteix d'una configuració inicial i explora l'espai de disseny passant d'una configuració a una altra. Les noves configuracions són acceptades amb una certa probabilitat segons el valor d'una funció de cost i d'un paràmetre intern: la temperatura. S'han definit una funció de cost i un conjunt de moviments per personalitzar l'algorisme de recuit simulat per al problema que es vol resoldre. La funció de cost avalua la qualitat de les configuracions. Els moviments ens permeten passar d'una configuració a una altra veïna.

S'ha presentat un conjunt de resultats que, tot i que no es poden comparar amb altres degut a l'inexistència d'algorismes que resolguin aquest mateix problema, demostren la qualitat del programa.

Capítol 8

Exemple de síntesi

Aquest capítol presenta un exemple complet de disseny d'un circuit asíncron utilitzant la metodologia presentada en el treball. El circuit implementa el filtre el·líptic de cinquè ordre. *La planificació d'operacions i assignació de recursos s'ha realitzat amb la metodologia descrita al capítol 7. Els mòduls que componen el camí de dades s'han construït utilitzant un conjunt d'eines de síntesi de* layout, *basades en una tecnologia de* Sea-of-Gates *(SoG). El control s'ha sintetitzat mitjançant la descripció dels diferents* controladors locals *amb STG, utilitzant la metodologia descrita al capítol 4. D'aquesta descripció en STG es poden obtenir circuits asíncrons lliures de riscos mitjançant tècniques existents. Per poder comprovar la qualitat de la solució obtinguda, s'ha comparat amb un disseny síncron del mateix exemple.*

8.1 Introducció

Els darrers capítols han presentat diferents propostes per realitzar diverses fases de la síntesi d'alt nivell de circuits asíncrons. L'objectiu d'aquest capítol és comprovar que aquestes propostes són vàlides i que permeten el disseny de circuits asíncrons. També es compara el resultat amb un disseny síncron del mateix circuit. El disseny asíncron que s'obté és més lent que el síncron ja que els elements del camí de dades asíncron encara no són prou ràpids per contrarestar el temps que es perd amb les sincronitzacions del control.

L'exemple que s'ha implementat és el *filtre el·líptic de cinquè ordre* [DDN85]. Per simplificar el circuit, s'ha suposat que les multiplicacions són per potències de 2 i que poden realitzar-se en temps 0 (amb desplaçaments), de manera que no caldran multiplicadors. El DFG sota aquestes suposicions és el que mostra la figura 8.1.

El model d'arquitectura del disseny és el que es descriu al capítol 4. Recordem que aquest model es basa en un sistema multiprocessador, on cada processador executa un procés i el control és totalment distribuït.

La planificació d'operacions i l'assignació de recursos s'ha efectuat utilitzant l'eina descrita al capítol 7. El model d'execució d'aquest programa permet l'encadenament de les operacions amb l'objectiu de minimitzar el temps total d'execució.

S'ha construït una petita llibreria de mòduls autotemporitzats amb lògica DCVSL utilitzant les eines de síntesi de *layout* del sistema OCEAN [GS93]. Aquestes eines permeten dissenyar el *layout* dels circuits utilitzant l'estil de disseny SoG amb tecnologia CMOS de 1,6 μm i dos nivells de metall.

SoG és un estil de disseny en què es treballa sobre una *imatge* prefabricada de transistors MOS. Aquests transistors estan disposats en fileres alternatives de transistors n i p. La tasca del dissenyador (o de l'eina, segons quin sigui el cas), és connectar aquests transistors amb línies de metall.

El conjunt d'eines permet dissenyar el *layout* de cel·les a mà i fer emplaçament i connexionat de manera automàtica a partir d'una descripció de la interconnexió dels mòduls.

Aquestes eines també disposen d'un simulador (*Simeye*) que permet comprovar el comportament lògic i de temps dels circuits [vGdG87].

S'han utilitzat aquestes eines ja que permeten definir una llibreria de cel·les amb tec-

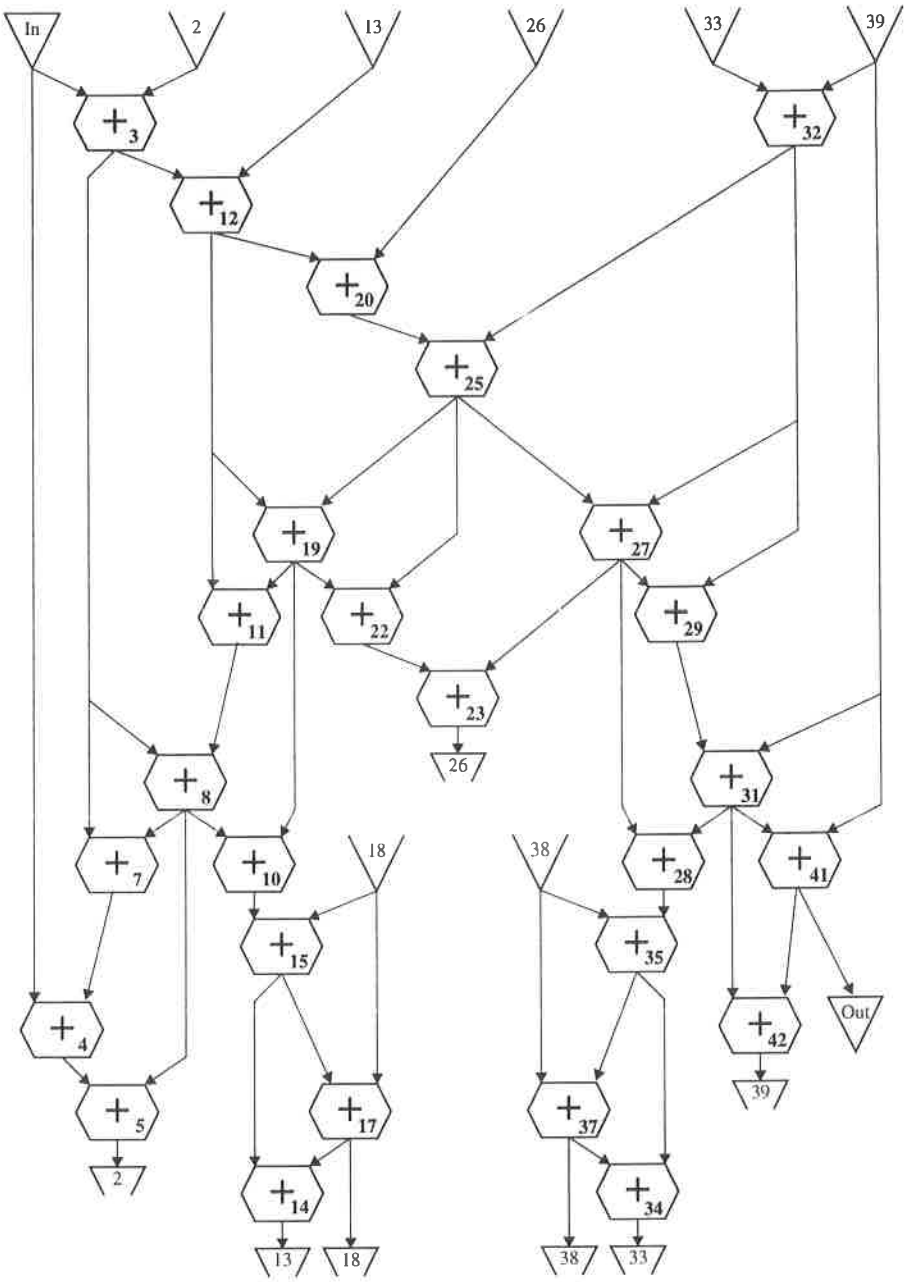


Figura 8.1: Graf de flux de dades del filtre el·líptic sense les multiplicacions

nologia SoG de forma senzilla.

Un cop s'han efectuat les fases de planificació d'operacions i assignació de recursos, queda definit el camí de dades i, també els processos que s'executaran en cada recurs del mateix. Aleshores, es poden dissenyar els STG que descriuen el comportament dels *controladors locals.*

Per comprovar la qualitat del circuit obtingut, es compara amb una realització síncrona del mateix. En aquest cas, la planificació i l'assignació de recursos s'ha fet a mà.

A continuació, es descriu el disseny asíncron, tant pel que fa al camí de dades com al control. A la secció 8.3 es presenta el disseny síncron. Finalment, a la secció 8.4 es fa una comparació de les dues solucions obtingudes i es presenten algunes conclusions.

8.2 Exemple asíncron

En aquesta secció es descriu com s'ha realitzat el disseny asíncron. L'arquitectura d'aquest exemple és la descrita al capítol 4. Primerament es descriu la llibreria de mòduls asíncrons que compondran el camí de dades. A continuació, es descriu la planificació i assignació de recursos que s'ha obtingut amb l'eina descrita al capítol 7. Seguidament, es descriu com s'han obtingut els STG que descriuen el comportament dels diferents *controladors locals.* Finalment, es presenta un resum de les dades més importants.

8.2.1 Llibreria

S'ha construït una llibreria de mòduls asíncrons en lògica DCVSL. Aquesta llibreria es compon de portes elementals, com NAND, NOR o portes C de Muller, que han permès construir components més complexos, com el sumador o els multiplexors. Aquests mòduls s'han dissenyat utilitzant les eines de disseny de *layout* OCEAN.

Les cel·les bàsiques s'han dissenyat a mà. Els altres components s'han generat de manera automàtica a partir de descripcions de la interconnexió de cel·les bàsiques. L'emplaçament automàtic es realitza amb el programa *Madonna* i el connexionat amb el programa *Trout.*

A continuació es descriuen alguns dels components d'aquesta llibreria. Els retards dels

mòduls per realitzar les diferents estimacions són estimacions obtingudes amb el simulador *Simeye*.

8.2.1.1 Sumador

El sumador que s'ha construït és un sumador de 16 bits amb propagació de *carry*. La cel·la elemental del sumador és el *full-adder*. La figura 8.2 mostra l'esquema en transistors del *full-adder* (generació del *carry* i de la suma) i la 8.3, el *layout*. La detecció del final de suma s'ha realitzat amb un arbre de portes C de dues i tres entrades.

El retard del sumador en mitjana és de 22 ns i en el seu cas pitjor és de 40,7 ns. Part d'aquest retard prové de la lògica de detecció d'acabament (3 ns). El temps addicional —en mitjana— si encadenem dues sumes és de 17 ns. La fase d'inicialització del sumador és d'11 ns. L'àrea del sumador és de 2,35 mm^2.

8.2.1.2 Multiplexor

S'han construït dos multiplexors: un de dues entrades i un de quatre; ambdós de 16 bits. La figura 8.4 mostra el esquema en transistors de la cel·la bàsica del multiplexor de dues entrades. i la 8.5, el *layout*.

El retard del multiplexor de dues entrades és de 10,8 ns en mitjana. Si encadenem amb l'unitat funcional anterior és de 7,2 ns, també en mitjana. El retard de detectar l'acabament és 3,6 ns. Aquest retard ja s'ha inclos en els retards anteriors. La fase d'inicialització requereix 8 ns.

El de quatre entrades té un retard de 16,8 ns en mitjana i si s'encadena, 11,4 ns. La detecció d'acabament triga 3,6 ns com en el multiplexor de dues entrades. El retard necessari per l'inicialització és de 9 ns.

L'àrea del multiplexor de dues entrades és de 0,44 mm^2 i la del de quatre, 0,99 mm^2.

8.2.1.3 Element de memòria

S'ha dissenyat un element de memòria de 16 bits a partir de *latch* d'un bit presentat al capítol 3. El *layout* del *latch* es mostra a la figura 8.6.

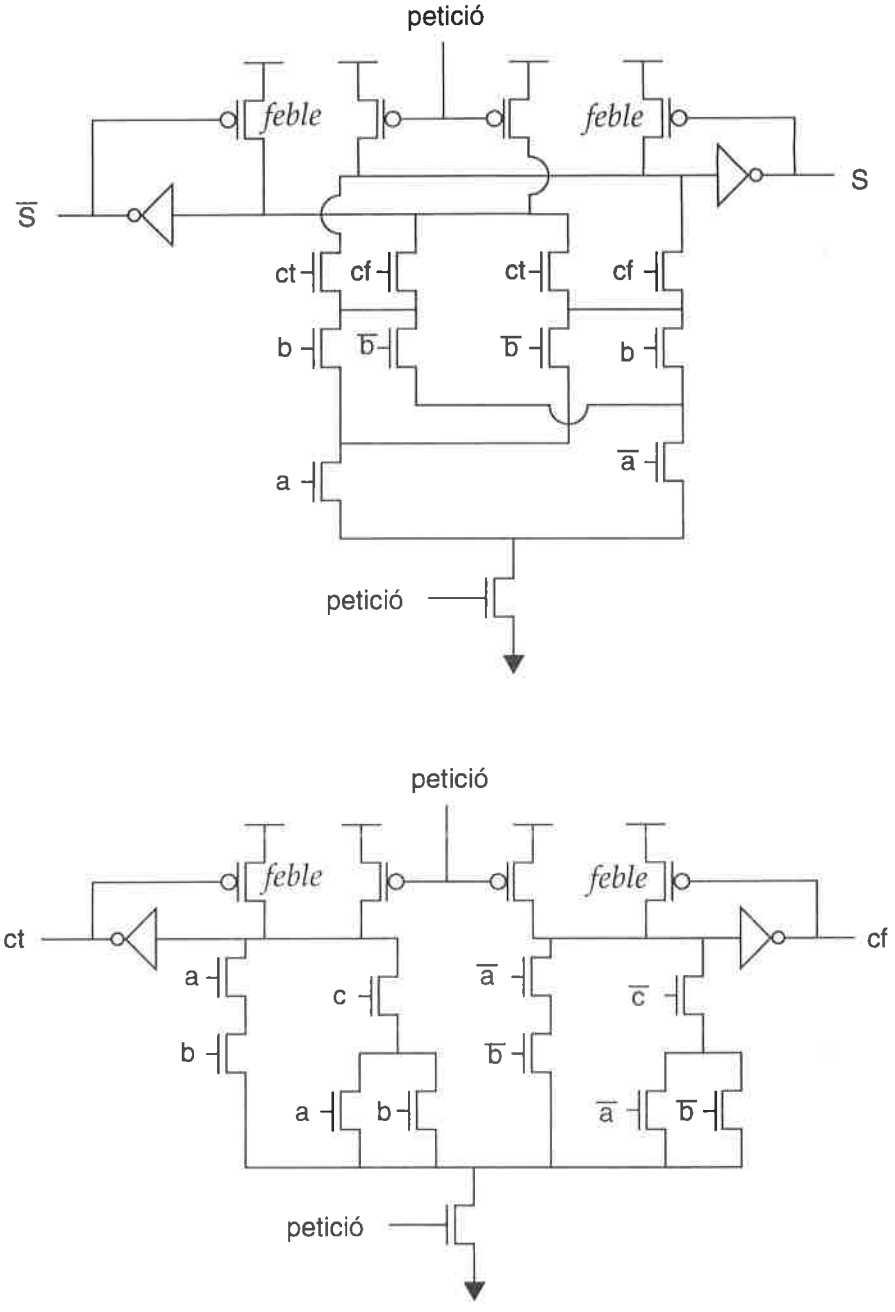


Figura 8.2: Esquema en transistors del full-adder *asíncron amb lògica DCVSL*

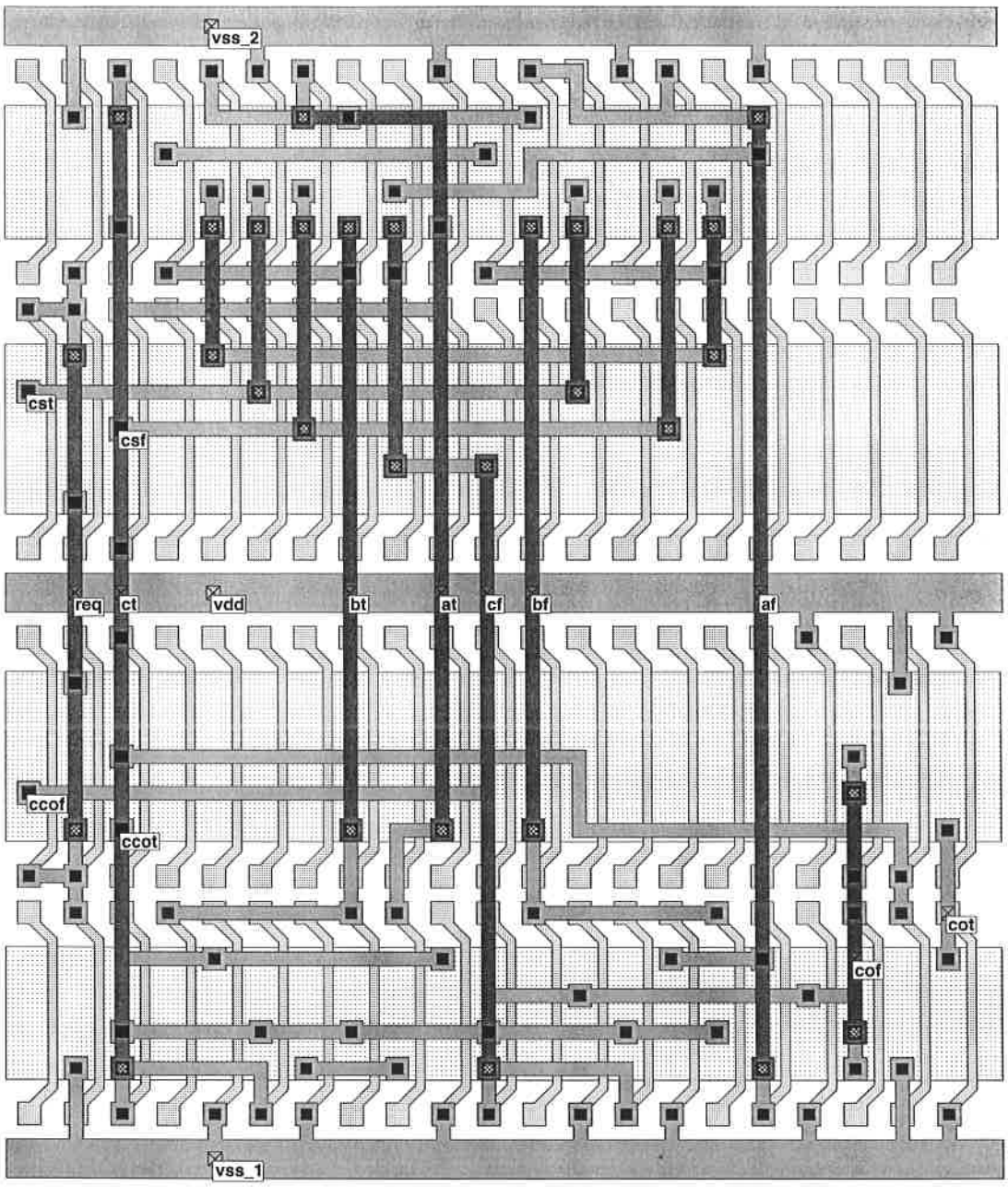


Figura 8.3: Layout *del* full-adder *asíncron amb lògica DCVSL*

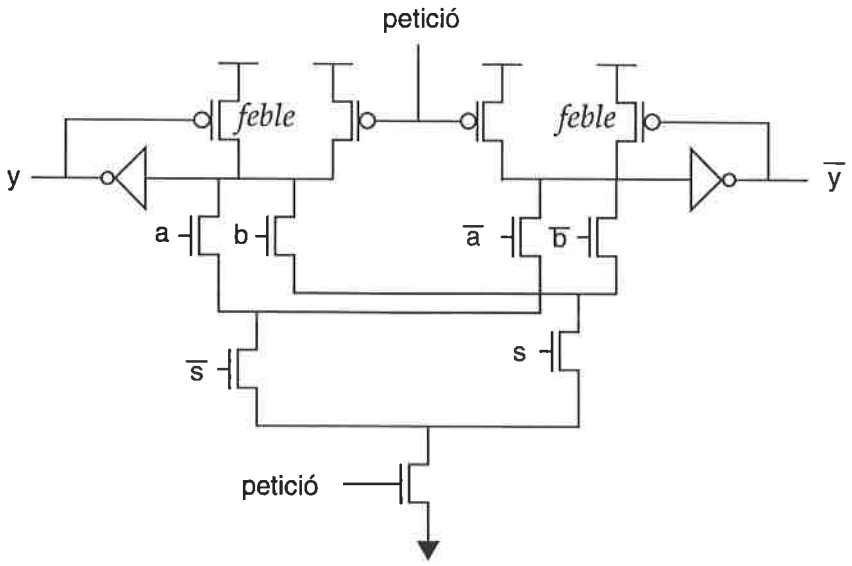


Figura 8.4: Esquema en transistors del multiplexor de dues entrades d'un bit asíncron amb lògica DCVSL

El retard del *latch* per emmagatzemar és en mitjana de 10 ns. Si encadenem amb l'unitat funcional predecessora aquest temps es redueix a 7 ns. L'inicialització requereix 8 ns.

L'àrea de l'element de memòria és de 0,41 mm^2.

8.2.2 Planificació d'operacions i assignació de recursos

La planificació d'operacions i assignació de recursos s'ha realitzat de manera automàtica, utilitzant el programa descrit al capítol 7. Les restriccions de recursos s'han fixat a cinc sumadors.

La figura 8.7 mostra la planificació i assignació obtinguda. Per cada operació s'indica el sumador que se li ha associat. Els arcs discontinuus representen dependències estructurals entre operacions associades al mateix recurs. L'assignació de recursos també decideix el nombre de multiplexors i registres a utilitzar i els associa a transferències i a variables emmagatzemades. En la figura no es reflexa tota aquesta informació per no complicar-la.

El retard estimat de la planificació en el seu cas mitjà és de 477 ns. Aquest retard

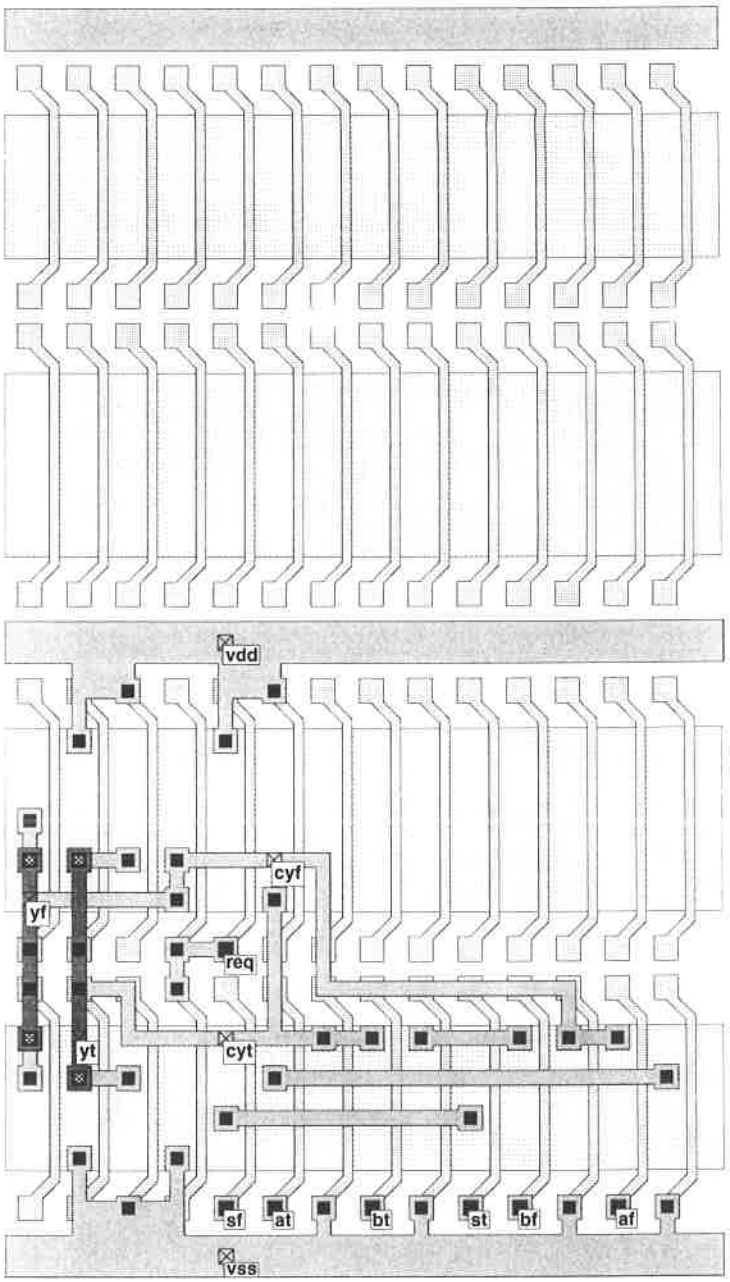


Figura 8.5: Layout *del multiplexor de dues entrades d'un bit asíncron amb lògica DCVSL*

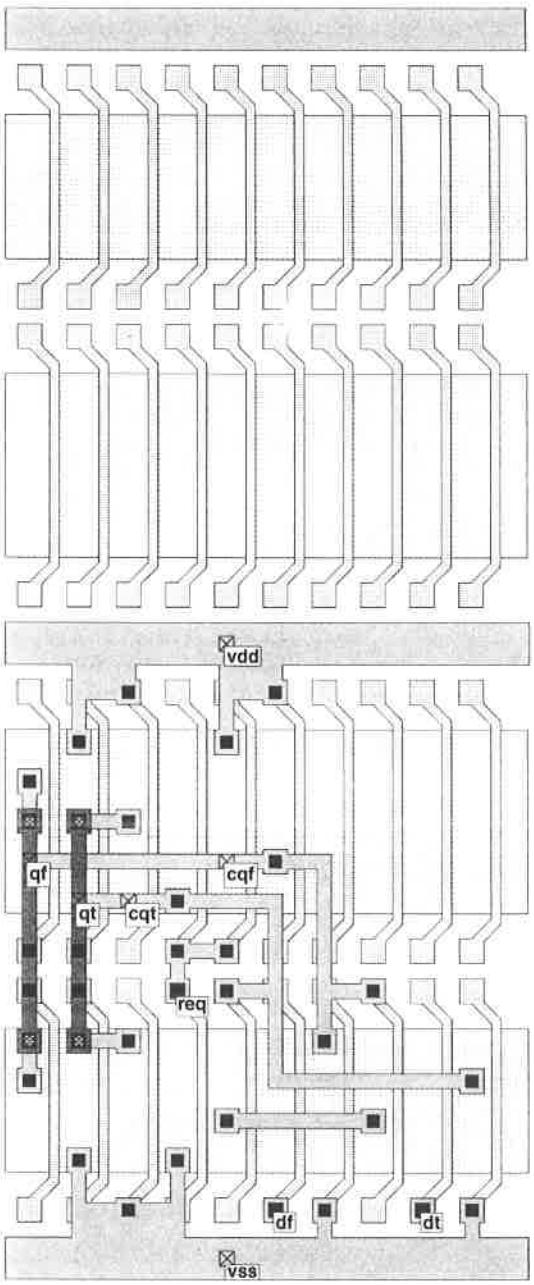


Figura 8.6: Layout *de l'element de memòria d'un bit asíncron amb lògica DCVSL*

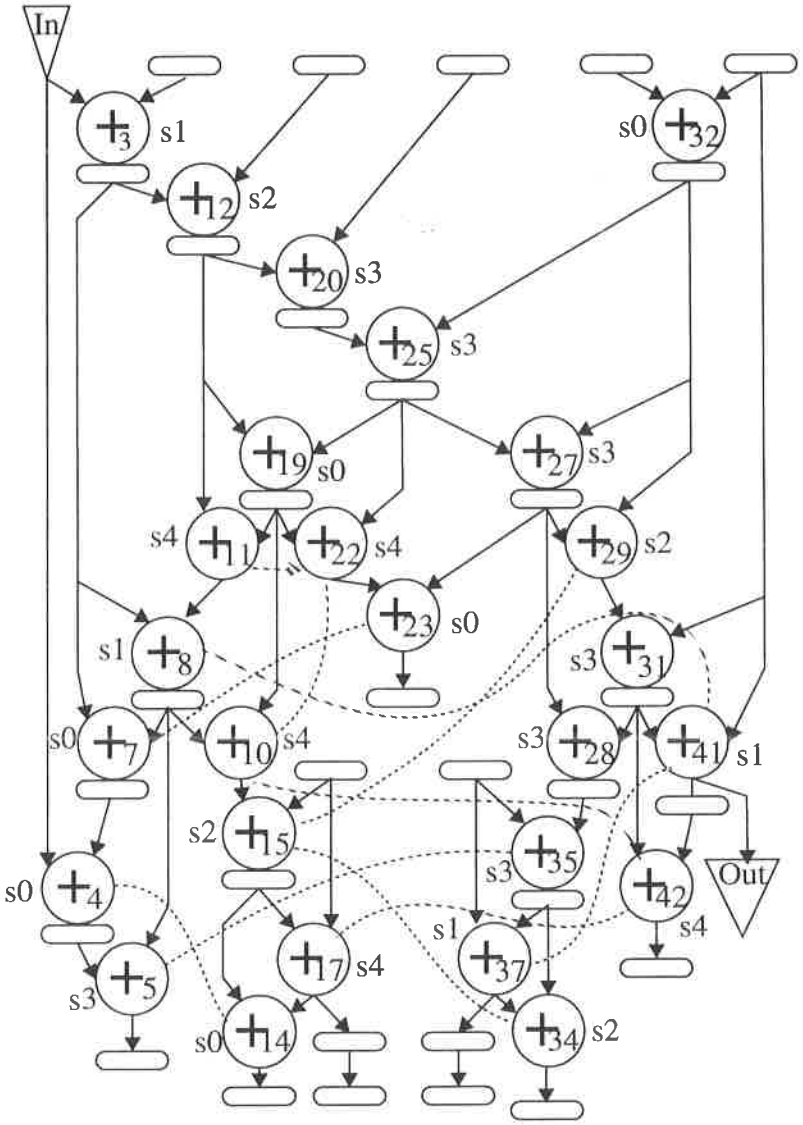


Figura 8.7: Planificació d'operacions i assignació de recursos per a l'exemple asíncron

s'obté sumant les estimacions dels retards de les diferents operacions que formen el camí crític de la planificació. Aquest retard no inclou el temps addicional que afegeixen les sincronitzacions de control.

L'associació de recursos que realitza el programa obté un camí de dades amb 14 registres. El connexionat necessita un total de 32 multiplexors de dues entrades que són realitzats amb 8 multiplexors de dues entrades i 9 multiplexors de quatre entrades.

8.2.3 Síntesi del control

En aquest cas s'ha suposat una granularitat mínima, de manera que tenim un *controlador local* per a cada component de camí de dades. Això implica un total de 37 controladors locals: 5 per als sumadors, 16 per als registres i 16 per als multiplexors. Hi ha un controlador menys dels esperats per als multiplexors, ja que hi ha dos multiplexors (un de quatre entrades i un de dues) que construeixen un únic multiplexor de cinc entrades i és per això que comparteixen el controlador local.

Tots aquests controladors locals han estat descrits amb un STG. Per construir els STG s'ha seguit la metodologia descrita al capítol 4. Aquests STG s'han dissenyat a mà, ja que aquesta part no està automatitzada encara.

La figura 8.8.b mostra l'STG corresponent al sumador 1. La figura 8.8.a mostra la seqüència d'operacions que s'executen en aquest sumador i les diferents sincronitzacions que es produeixen amb controladors locals d'altres mòduls. La part del camí de dades corresponent al sumador 1 es mostra a la figura 8.8.c. Aquest sumador està connectat als multiplexors 3 i 8 per les dues entrades i la sortida es connecta als multiplexors 2, 4, 24 i 25 i als registre 8 i 9.

$ev_{m3}, ev_{m8}, ec_{m3}$ i ec_{m8} representen els senyals d'entrada vàlida i entrada consumida dels multiplexors 3 i 8, respectivament. També hi han els corresponents senyals de *sortida vàlida* i *sortida consumida* per a cadascun dels mòduls que es connecta amb la sortida del sumador. L'estructura de l'STG és la mateixa que la descrita al capítol 4.

Els STG generats poden ser sintetitzats utilitzant tècniques existents i obtenir circuits asíncrons lliures de riscos. Per exemple, l'STG que descriu el comportament del controlador local del sumador 0 pot ser descrit amb diverses equacions booleanes que en total estan

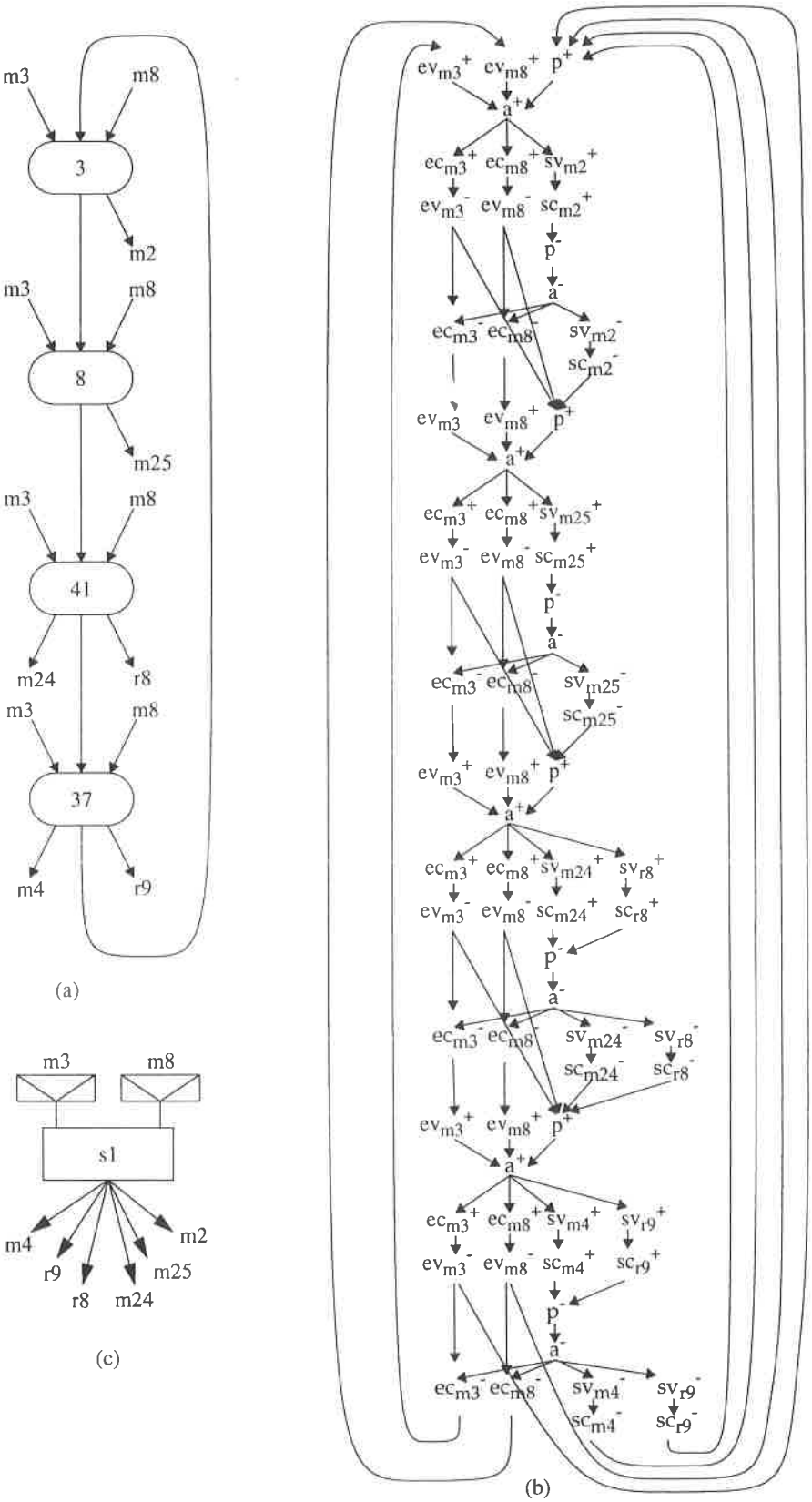


Figura 8.8: STG del controlador local del sumador 1

compostes per 257 literals de 22 variables. Aquestes funcions es poden realitzar amb un circuit format per 12 portes C de Muller i una xarxa a dos nivells de 42 portes NAND. Aquesta implementació s'ha obtingut considerant el model de circuits independents de la velocitat. Si es consideressin retards acotats, s'obtindria una solució millor, però encara no tenim eines per fer-ho.

S'ha calculat que de totes les sincronitzacions que realitza cada controlador local amb altres controladors i amb el component que controla per cada operació, n'hi han dues que s'afegeixen al camí crític. El camí crític de la planificació que s'ha realitzat té deu operacions, nou emmagatzemaments en registres i 17 transferències a través de registres, de manera que hi hauran 72 sincronitzacions que caldrà afegir al temps de planificació.

Per cada sincronització s'ha estimat que es poden trigar uns 2 ns. Per tant, caldria afegir uns 144 ns a la planificació en concepte de sincronitzacions.

8.2.4 Resum de l'exemple asíncron

L'exemple asíncron s'ha generat parcialment de manera automàtica i parcialment de manera manual. La planificació i assignació s'ha fet de manera automàtica utilitzant el programa presentat al capítol 7. Els STG que descriuen el control s'han generat manualment, però utiltizant la metodologia descrita al capítol 4. A partir dels STG podem generar els circuits de control de manera automàtica utilitzant propostes existents [PC93b].

El *layout* del camí de dades s'ha generat de manera automàtica utilitzant les eines OCEAN. Per problemes amb les eines no s'ha pogut obtenir tot el camí de dades complet d'una sola vegada, sino que s'ha partit en diverses parts. S'ha fet separadament emplaçament i connexionat de cadascuna de les parts per poder obtenir una estimació de l'àrea que ocuparà el camí de dades.

La taula 8.1 mostra un resum de les dades de temps i d'àrea de l'exemple que s'ha realitzat.

El control s'ha descrit amb STG que poden ser sintetizats amb eines existents. S'ha estimat que el control afegirà uns 144 ns al temps de la planificació en concepte de sincronitzacions.

Tot l'exemple complet (camí de dades i control) no s'ha pogut simular, ja que no s'ha

CAMÍ de DADES						
Recurs	Àrea (mm^2)			Temps (ns)		
	recursos	àrea recurs	total	retard mitjà	retard encad.	inicialització
multiplexors 2:1	8	0,44	3,49	10,8	7,2	8
multiplexors 4:1	9	0,99	8,94	16,8	11,4	9
sumadors	5	2,35	11,77	22	17	11
registres	14	0,41	5,67	10	7	8
Àrea total obtinguda (estimació)			91,38	Temps mitjà esperat (estimat) de planificació		477
CONTROL						
Estimació del retard del control				144 ns		

Taula 8.1: Resum de l'exemple asíncron

pogut acabar d'integrar totes les parts per problemes amb les eines de CAD.

L'estimació del temps total d'execució per cada iteració d'aquest exemple és de 621 ns.

8.3 Exemple síncron

En aquesta secció es presenta un disseny del mateix filtre el·líptic per a una arquitectura síncrona. Per equiparar amb l'exemple asíncron, s'ha permès l'encadenament de les operacions. S'ha fet una planificació d'operacions *a mà* amb restricció de recursos. Aquesta restricció, com en el cas anterior, és de cinc sumadors. L'associació de recursos s'ha fet també a mà.

Per poder construir el camí de dades, s'ha dissenyat una petita llibreria de mòduls amb lògica CMOS, utilitzant les mateixes eines d'OCEAN.

8.3.1 Llibreria síncrona

La llibreria es compon de quatre mòduls: un sumador amb propagació de *carry*, un registre, un multiplexor de dues entrades i un multiplexor de quatre entrades, tots ells de 16 bits.

Aquesta llibreria s'ha dissenyat amb les mateixes eines d'OCEAN i els retards dels mòduls s'han obtingut amb el simulador *Simeye*.

8.3.1.1 Sumador

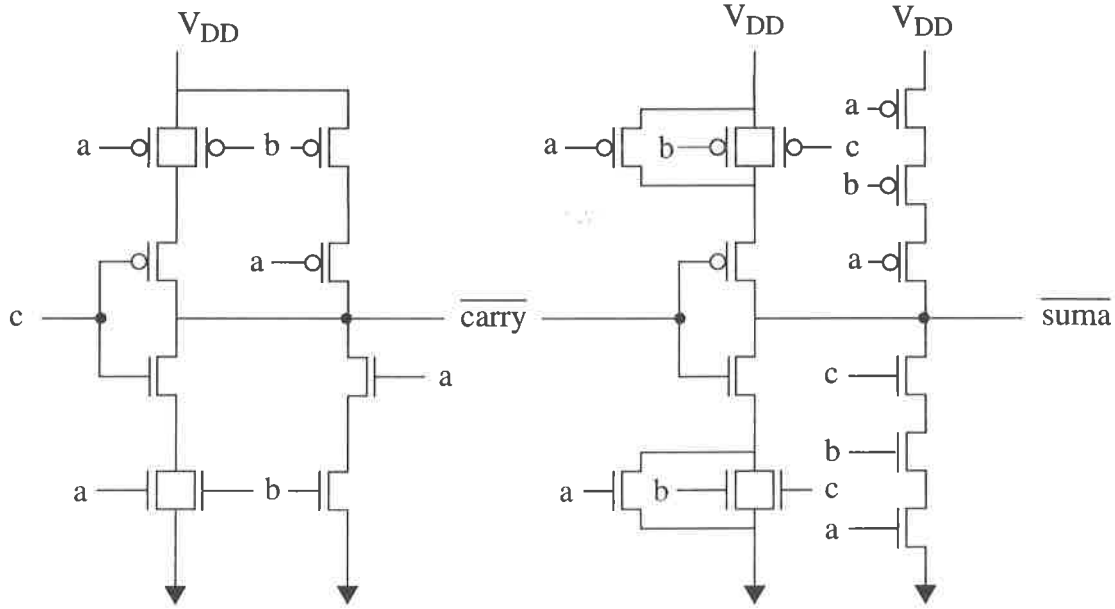


Figura 8.9: Esquema en transistors del full-adder *síncron*

S'ha dissenyat un sumador amb propagació de *carry* a partir d'una cel·la bàsica: un *full-adder*. L'esquema en transistors del *full-adder* es mostra a la figura 8.9 [WE85].

El *layout* de la cel·la és el que mostra la figura 8.10. El temps de càlcul del sumador és de 32 ns en el pitjor cas (temps de simulació). Atès que el temps de suma del pitjor cas és quan el *carry* es propaga per tots els bits, el temps d'encadenar una altra suma serà com a molt el temps de suma d'un bit més el temps de propagar un bit. Aquest temps és de 2 ns.

L'àrea total del sumador és de 0,47 mm^2.

8.3.1.2 Multiplexors

Els multiplexors s'han construït a partir de les cel·les *mu111* i *mu210* ja existents a la llibreria que proporcionen el paquet OCEAN [GS93]. Aquestes cel·les són multiplexors

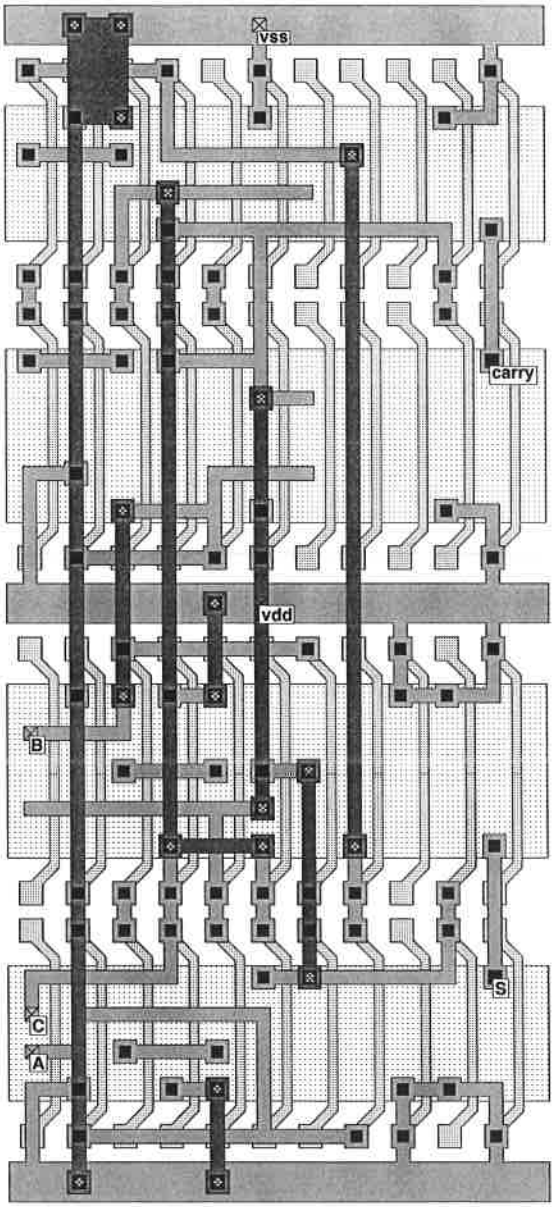


Figura 8.10: Layout *del* full-adder *síncron*

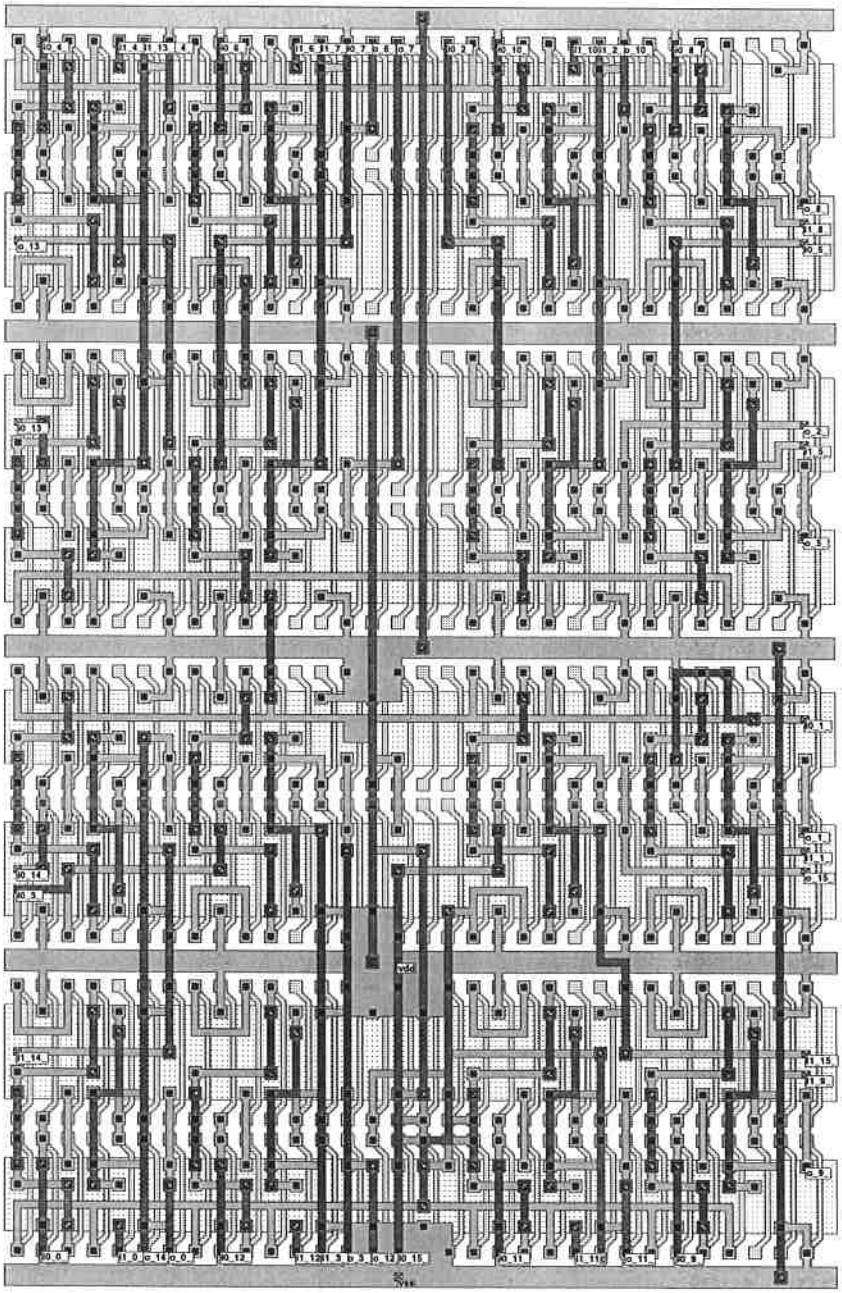


Figura 8.11: Layout *del multiplexor de dues entrades de 16 bits síncron*

d'un bit de dues i de quatre entrades respectivament.

La figura 8.11 mostra el *layout* del multiplexor de dues entrades de 16 bits síncron. El retard que afegeix el multiplexor de dues entrades és de 8,2 ns en el pitjor dels casos i el de quatre entrades és de 12,8 ns.

L'àrea total del multiplexor de dues entrades és de 0,11 mm^2 i la del de quatre entrades de 0,33 mm^2.

8.3.1.3 Registre

El registre, igual que els multiplexors, s'ha construït a partir d'una cel·la bàsica existent a la llibreria: la *dfr11* que implementa un *flip-flop* de tipus D amb possibilitat d'inicialització a zero. El retard del registre per emmagatzemar la informació és de 2 ns en el pitjor dels casos. L'àrea total del registre és de 0,32 mm^2.

8.3.2 Planificació d'operacions i assignació de recursos

La planificació d'operacions s'ha realitzat manualment. S'ha permès l'encadenament d'operacions per equiparar la comparació. El nombre màxim d'unitats funcionals a utilitzar en la planificació és de cinc sumadors, com en el cas asíncron. La figura 8.12 mostra la planificació que s'ha generat. Podem veure com les diferents operacions s'han distribuït en sis cicles. En cada cicle, hi ha algunes operacions encadenades amb d'altres.

L'associació de recursos s'ha realitzat també a mà. El connexionat utilitza 27 multiplexors equivalents de dues entrades. La configuració obtinguda utilitza, a part dels cinc sumadors, nou registres, vuit multiplexors de dues entrades i vuit multiplexors de quatre entrades.

8.3.3 Anàlisi de temps

Un cop s'han definit la planificació i l'assignació, se sap quins recursos han d'operar en cada cicle. Per a cada cicle podem calcular quin és el retard màxim de les operacions executades. Es pren com a temps de cicle, el pitjor dels casos.

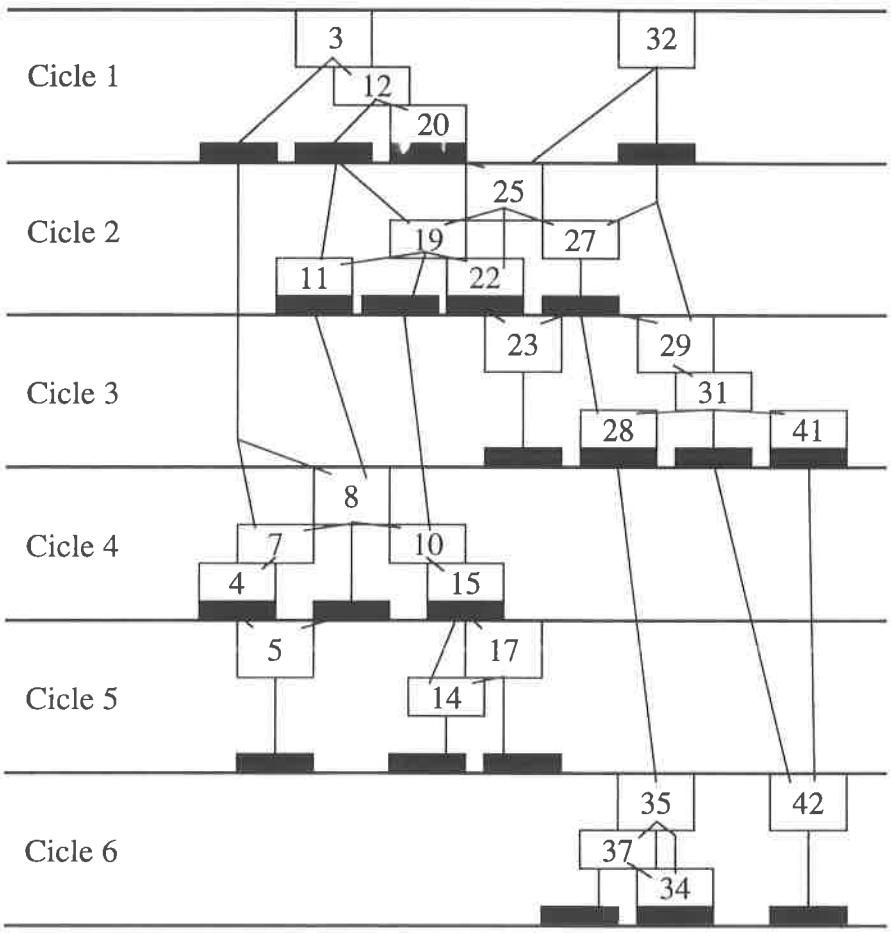


Figura 8.12: Planificació d'operacions per a l'exemple síncron

El càlcul del pitjor dels casos s'obté amb la següent expressió:

$$t_{cicle} = 4t_{mux} + t_{suma} + 2t_{enc} + t_{reg} \tag{8.1}$$

on t_{mux} és el temps de pas per a un multiplexor de quatre entrades, t_{suma} és el temps de càlcul d'un sumador, t_{enc} és el temps addicional que tenim si dues sumes estan encadenades i t_{reg} és el temps necessari per emmagatzemar una dada en un registre.

És a dir, en el pitjor dels casos tindrem tres sumes encadenades (un temps de suma i dos temps de suma encadenada), el resultat de les sumes es guarden en registres i cadascun dels sumadors i registre llegeixen les dades d'entrada d'un multiplexor de quatre entrades. Aquest cas es produeix, per exemple, en el cicle 1. Substituint cada valor pel retard corresponent, tenim un temps de cicle de 89,2 ns.

Atès que la planificació requereix sis cicles, tenim un temps total de 535,2 ns.

8.3.4 Control

El control s'ha descrit amb *kiss2* [SSL+92] i s'ha sintetitzat amb l'eina *kissis* del mateix paquet OCEAN. *Kissis* és una interfície de *SIS* que a més de fer minimització lògica d'estats fa projecció del resultat sobre la llibreria disponible a OCEAN. *SIS* [SSL+92] és una eina de síntesi i optimització de sistemes seqüencials desenvolupada a la Universitat de California de Berkeley.

El circuit que implementa el control es compon de 3 *flip-flops*, 13 inversors, 8 portes NOR i 10 portes NAND. S'ha obtingut el *layout* del control síncron que ocupa 0,36 mm^2.

S'ha considerat que el retard que afegeix el control a la planificació en cada cicle és el temps d'un flip-flop. Aquest temps és de 2,3 ns en el seu cas pitjor. En total cal afegir un temps de 13,8 ns en concepte de control.

8.3.5 Resum

El camí de dades, com en el cas de l'exemple asíncron, s'ha generat de forma automàtica utilitzant les eines d'emplaçament i connexionat del paquet OCEAN.

L'àrea total del camí de dades de l'exemple síncron és de 42,32 mm^2 i la del control és de 0,37 mm^2.

El temps total d'execció en el cas síncron és de 535,2 ns. La taula 8.2 mostra un resum de les dades més importants del camí de dades del exemple síncron.

CAMÍ de DADES					
Recurs	Área (mm^2)			Temps (ns)	
	recursos	àrea recurs	total	retard màxim	retard encad.
multiplexors 2:1	8	0,11	0,88	8,2	–
multiplexors 4:1	8	0,33	2,66	12,8	–
sumadors	5	0,47	2,34	32	2
registres	9	0,32	2,86	2	–
Àrea total obtinguda			42,32	Temps de planificació	535,2
CONTROL					
Àrea del control			0,36	Temps del control	13,8

Taula 8.2: Resum de l'exemple síncron

8.4 Conclusions

La taula 8.3 fa una comparació dels resultats obtinguts per als dos exemples.

	Síncron	Asíncron
Área (mm^2)	42,32	91,38
Temps camí de dades (ns)	535,2	477
Temps control (ns)	13,8	144
Temps total (ns)	549	621

Taula 8.3: Comparació de les dues solucions

Com es pot observar, tot i els temps mitjans de les unitats de càlcul, els circuits asíncrons encara no són més ràpids que els síncrons. Això és a causa d'un excés de sincronitzacions,

pels retards afegits, per la detecció d'acabament i pel temps d'inicialització de les unitats. Si enlloc de sumadors amb propagació de *carry* s'haguessin utilitzat sumadors *CLA* probablement encara hi hauria hagut més diferència entre el disseny síncron i l'asíncron.

A continuació es proposen un conjunt de possibles millores per tal de poder reduir l'impacte d'aquests factors:

- Augmentar la granularitat: d'aquesta manera es reduiria la sobrecàrrega del control. En aquest cas, cada controlador local estaria lligat a un conjunt de components del camí de dades, enlloc d'un de sol.

- Utilitzar un sobreexcés d'unitats de càlcul: aquesta proposta segueix la línia del divisor proposat per Williams [WHAY87]. Aquest és el divisor més ràpid que s'ha dissenyat. Les dades flueixen sense necessitar registres. Mentre una unitat funcional està inicialitzant-se, una altra està calculant i una altra està actuant com a element de memòria. L'inconvenient d'aquesta proposta és que l'àrea augmenta significativament.

- Utilitzar una codificació de dades compactada: en aquest cas, les estimacions del retard són més depenents de la tecnologia. Una solució intermedia és utilitzar codificació de doble via per als camins crítics. D'aquesta manera es reduiria el temps de detecció d'acabament.

La conclusió positiva d'aquest experiment és que la metodologia de disseny presentada en aquest treball funciona i permet obtenir circuits asíncrons de manera gairebé automàtica. Però encara queda molt camí per fer en el disseny de circuits asíncrons si volem que siguin més eficients que els síncrons.

Capítol 9

Conclusions i línies obertes

Aquest capítol presenta les conclusions de les diverses contribucions del treball: un model d'arquitectura asíncrona, algorismes de síntesi d'alt nivell i un exemple de disseny. També es presenten les línies obertes que deixa la tesi: integració de les diferents eines que s'han desenvolupat en un sistema de síntesi d'alt nivell i resolució d'altres problemes oberts, com són l'estudi de tècniques per realitzar circuits aritmètics asíncrons més ràpids, síntesi lògica i verificació de circuits asíncrons.

9.1 Introducció

Els circuits asíncrons es caracteritzen per l'absència de senyal de rellotge i, per tant, també desapareix l'execució en cicles de les operacions. Els atractius dels circuits asíncrons són molts i es poden resumir en els següents:

- Baix consum: en els circuits síncrons la major part de consum prové del senyal de rellotge. En els circuits asíncrons, totes les transicions de senyals són útils, de manera que no es desaprofita l'energia.

- Modularitat: les sincronitzacions entre components asíncrons d'un mateix circuit es fan mitjançant un intercanvi de senyals que segueixen un cert protocol. Aquesta característica fa que el disseny de circuits asíncrons sigui molt modular i independent de la velocitat de l'entorn.

- Desfasament del senyal de rellotge: aquest és un problema cada vegada més important en el disseny dels circuits síncrons. En els circuits asíncrons, com que no existeix aquest senyal, tampoc no tenim aquest problema.

- Velocitat mitjana de càlcul: en els circuits síncrons, s'ha de considerar el retard més llarg de les unitats funcionals a l'hora de fixar el temps de cicle. En els circuits asíncrons, el temps de càlcul de les unitats funcionals depèn de les dades d'entrada. Així, en uns casos tindrem un temps més gran i en altres més curt, de manera que podem considerar que hi ha un temps mitjà de càlcul.

A pesar de tots aquests motius, el disseny de circuits asíncrons ha estat poc extès a causa d'alguns inconvenients:

- Àrea: els circuits asíncrons ocupen més àrea que circuits equivalents síncrons per diferents motius, com la sobrecàrrega dels circuits de control i la codificació de les dades en doble via.

- Riscos i curses: el disseny dels circuits asíncrons és més complex, ja que cal garantir que no existeixin ni riscos ni curses.

- Metastabilitat: tot i que aquest és un problema que també existeix en els circuits síncrons, en els circuits asíncrons és més difícil de solucionar.

Per aquests motius creiem que és important que existeixin eines de disseny automàtic de circuits asíncrons, que permetin aprofitar-ne els avantatges sense haver de passar per la feina feixuga del seu disseny.

Pel nostre coneixement, fins ara, totes les contribucions existents dins del camp de l'automatització dels circuits asíncrons han estat orientades a la síntesi lògica o síntesi de circuits de control.

Aquesta tesi presenta la primera proposta d'un model d'arquitectura asíncrona orientat a la síntesi d'alt nivell de circuits asíncrons. També es presenten per primera vegada algorismes de planificació d'operacions i assignació de recursos per a sistemes asíncrons.

9.2 Contribucions del treball

Les contribucions d'aquest treball són les següents:

- Model d'arquitectura asíncrona: s'ha proposat aquest model com a base per a la síntesi d'alt nivell [CB92, CBPP92]. Consisteix en un sistema multiprocessador on cada processador executa un procés i el control està completament distribuït. Els components del camí de dades són mòduls autotemporitzats. El comportament del control es realitza amb STG. A partir de les descripcions en STG es poden obtenir circuits asíncrons lliures de riscos amb eines de síntesi existents.

- Algorismes de síntesi d'alt nivell:

  - Planificació d'operacions basada en llistes d'esdeveniments: s'ha presentat una metodologia de planificació d'operacions basada en llistes d'esdeveniments [BC93b]. Aquesta metodologia es concreta amb dos algorismes: *ELS* i *ELLAS*, ambdós de cost polinòmic. Els resultats obtinguts amb aquests algorismes mostren que la seva qualitat és comparable als millors algorismes de planificació d'operacions per a sistemes síncrons.

  - Associació global de recursos (*GLASS*): s'ha presentat un algorisme d'associació de recursos basat en la teoria de grafs, on les tres subtasques d'associació es realitzen de manera simultània [BC93a]. El cost de l'algorisme és polinòmic. L'estructura bàsica d'aquest algorisme és el *graf de compatibilitat global.* Aquesta estructura sembla un bon punt de partida per a altres tècniques d'associació de recursos.
  - Planificació i assignació simultània: s'ha presentat un algorisme que permet realitzar la planificació d'operacions i l'assignació de manera simultània. El model d'execució permet l'encadenament de les operacions amb l'objectiu de reduir el temps de la planificació. El model d'arquitectura en què es basa aquest algorisme utilitza mòduls autotemporitzats i, en concret, mòduls construïts amb lògica DCVSL. Una característica d'aquests mòduls és que cal una fase d'inicialització entre càlcul i càlcul que cal tenir en compte a la planificació. L'algorisme està basat en la tècnica de *recuit simulat.* S'ha caracteritzat l'algorisme proposant un conjunt de moviments per passar d'una configuració a una altra i, una funció de cost que permet avaluar la qualitat de les configuracions.

- Realització d'un exemple: s'ha presentat un exemple complet de disseny d'un circuit asíncron utilitzant el model d'arquitectura presentat al capítol 4. La planificació i assignació de recursos s'ha fet de manera automàtica utilitzant el programa presentat al capítol 7. El control s'ha generat seguint la metodologia proposada al capítol 4. Cadascun dels controladors locals s'ha descrit amb un STG. Dels STG podem passar a circuits asíncrons lliures de riscos utilitzant tècniques existents. La solució obtinguda s'ha comparat amb un exemple síncron realitzat a mà.

9.3 Línies obertes

Com a línies obertes d'aquest treball queden:

- Paral·lelització de bucles: el problema de planificació d'operacions només s'ha tractat a nivell de blocs bàsics. Dins del grup de recerca ja s'està treballant en el camp de la paral·lelització de bucles per a sistemes síncrons i properament s'estendrà a sistemes

asíncrons. Aquest treball serà molt útil per a aplicacions de processat de senyal. L'objectiu és integrar el treball que ja s'ha realitzat amb paral·lelització de bucles amb el de planificació d'operacions de circuits asíncrons. Una possible estrategia per aconseguir aquest objectiu seria integrar els algorismes de paral·lelització de bucles *RCPL* [SC93] amb els de planificació d'operacions per sistemes asíncrons per tal de poder aprofitar l'experiència del grup de recerca en ambdós temes.

- Disseny de circuits aritmètics asíncrons: cal dissenyar circuits aritmètics asíncrons més ràpids i de baix consum. Per obtenir el primer objectiu es poden explotar tècniques com la de detectar el final de les operacions amb sensors de corrent [DDH91]. Quan una operació acaba, deixen de produir-se transicions en el circuit, de manera que es pot detectar el final de les operacions pel fet que deixa de passar corrent entre tensió i terra. Amb aquesta tècnica es redueix el temps de detecció de final de les operacions.

  Un altra objectiu del grup es el disseny de circuits asíncrons de baix consum utilitzant tècniques de síntesi lògica. Amb aquestes tècniques, el nombre de transicions no útils són reduïdes o eliminades.

- Síntesi lògica d'STG amb un gran nombre de transicions de senyals: la majoria de propostes existents són de cost exponencial amb el nombre de vèrtexs de l'STG. Dins del grup ja s'està treballant en algorismes de cost polinòmic que permetin la síntesi a partir d'STG grans.

  Fins ara s'han presentat propostes per fer síntesi a partir de xarxes de Petri de lliure elecció. Amb aquest tipus de xarxa no podem descriure el comportament de qualsevol sistema, per tant, és interesant proposar tècniques que permetin fer síntesi a partir de qualsevol tipus de xarxa de Petri.

- Verificació de circuits asíncrons: la verificació és necessària ja que altrament les eines de síntesi no seran acceptades per la comunitat dissenyadora. Dins del grup també s'han iniciat ja treballs en l'àrea de verificació. Aquests treballs es basen en el fet que el conjunt de marcatges d'una xarxa de Petri són isomorfs amb amb les funcions booleanes de n variables, en què n és el nombre de llocs de la xarxa [PRCB94].

Fins ara s'han verificat circuits independents de la velocitat. En un futur es vol ampliar aquest treball a circuits amb retards acotats.

- Disseny de llibreries de portes per a mòduls autotemporitzats: és important disposar de llibreries de portes i móduls autotemporitzats com a base de la síntesi d'alt nivell. Aquest treball ja s'ha començat utilitzant les eines de síntesi de *layout* OCEAN, però s'ha d'ampliar i millorar.

- Afegir possibles millores als circuits asíncrons: al capítol 8 s'han proposat diverses millores per tal d'augmentar l'eficiència dels circuits asíncrons. Per exemple, s'ha proposat d'utilitzar un sobreexcés d'unitats de càlcul o de buscar tècniques per reduir el temps de detecció d'acabament, utilitzant una codificació de dades compacte.

Bibliografia

[AFM69] D. B. Armstrong, A. D. Friedman, i P. R. Manon. Design of asynchronous circuits assuming unbounded gate delays. *IEEE Trans. on Computers*, Vol. C-18, Dec 1969.

[ART93] R. Auletta, B. Reese, i C. Traver. A comparison of synchronous and asynchronous fsmd design. In *Int. Conf. on Computer Design*, pages 178–182, 1993.

[ASU86] A.V. Aho, R. Sethi, i J.D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[AvL85] E.H.L. Aarts i P.J.M. van Laarhoven. Statistical cooling: A general approach to combinatorial optimization problems. *Philips J. of Research*, (40):193–226, 1985.

[Bak90] H. B. Bakoglu. *Circuits, Interconnections and Packaging for VLSI*. Addison-Wesley, 1990.

[BC93a] R.M. Badia i J. Cortadella. GLASS: a graph-theoretical approach for global binding. *Microprocessing and Microprogramming*, 38(1-5):775–782, 1993.

[BC93b] R.M. Badia i J. Cortadella. High-level synthesis of asynchronous systems: Scheduling and process synchronization. In *Proc. of EDAC-93*, 1993.

[BL84] E. Bonomi i J.-L. Lutton. The n-city travelling salesman problem: Statistical mechanics and the metropolis algorithm. *SIAM Rev.*, (26):551–568, 1984.

[BM89] M. Balakrishnan i P. Marwedel. Integrated scheduling and binding: a synthesis approach for design space exploration. In *Proc. of the 26th ACM/IEEE Design Automation Conference*, pages 68–74, 1989.

[BP90] N. Berry i B.M. Pangrle. Schalloc: and algorithm for simultaneous scheduling and connectivity binding in a data path synthesis system. In *Proc. of EDAC-90*, pages 78–82, 1990.

[Cam91] R. Camposano. Path-based scheduling for synthesis. *IEEE Trans. on CAD*, Vol. 10(1), January 1991.

[CB92] Jordi Cortadella i Rosa M. Badia. An asynchronous architecture model for behavioral synthesis. In *Proc. of the European Design Automation Conference*, pages 307–311, Març 1992.

[CBA91a] J. Cortadella, R.M. Badia, i E. Ayguadé. Scheduling in a continuous area-time design space. *Microprocessing and Microprogramming*, 32(1-5):199–206, 1991.

[CBA91b] J. Cortadella, R.M. Badia, i E. Ayguadé. Scheduling in a continuous area-time design space: a simulated annealing-based approach. In *ACM Fifth International Workshop on High-Level Synthesis*, Març 1991.

[CBPP92] J. Cortadella, R.M. Badia, E. Pastor, i A. Pardo. Achilles: A high-level synthesis system for asynchronous circuits. In *Sixth International Workshop on High-Level Synthesis*, pages 87–94, Novembre 1992.

[CDS93] B. Coates, A. Davis, i K. Stevens. The post office experience: Designing a large synchronous chip. *Integration, the VLSI journal*, 15(3):341–366, October 1993.

[Chu87] Tam-Anh Chu. *Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications.* Phd thesis, MIT, june 1987.

[CLR90] T.H. Cormen, C.H. Leiserson, i R.L.Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[CR89] R. Camposano i W. Rosenstiel. Synthesizing circuits from behaviorals descriptions. *IEEE Trans. on CAD*, 8:171–180, 1989.

[CST91] R. Camposano, L.F. Saunders, i R.M. Tabet. VHDL as input for high-level synthesis. *IEEE Design & Test of Computers*, pages 43–49, March 1991.

[DDH91] M.E. Dean, D.L. Dill, i M. Horowitz. Self-timed logic using current-sensing completion detection (CSCD). In *Proc. of the Int. Conference on Computer Design*, pages 187–191, 1991.

[DDN85] P. Dewilde, E. Deprettere, i R. Nouta. *Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms.* T. Kailath, 1985.

[DLSM81] S. Davidson, D. Landskov, B. D. Shriver, i P. W. Mallet. Some experiments in local microcode compaction for horizontal machines. *IEEE Trans. on Computers*, Vol. C-30, Jul 1981.

[DN89] S. Devadas i A.R. Newton. Algorithms for hardware allocation and data path synthesis. *IEEE Trans. on CAD*, 8(7):768–781, 1989.

[Dob92] Dobberpuhli, D.W. et al. A 200-mhz 64-bit dual-issuc CMOS microprocessor. *IEEE Journal of Solid-State Circuits*, 27(11):1555–1567, Novembre 1992.

[Fel50] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. Wiley, New York, 1950.

[GJ79] M.R. Garey i D.S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, New York, 1979.

[GKR91] P. Gutberlet, H. Krämer, i W. Rosenstiel. CASCH - a scheduling algorithm for high-level synthesis. In *Proc. of the European Conference on Design Automation*, pages 311–315, 1991.

[GS86] B.L. Golden i C.C. Skiscim. Using simulated annealing to solve routing and location problems. *Naval Logistics Research Quaterly*, (33):261–279, 1986.

[GS93] P. Groeneveld i P. Stravers. *Ocean: the Sea-of-Gates Design Style.* Delft University of Technology, Faculty of Electrical Engineering, 1993.

[HCLH90] C.Y. Huang, Y.S Chen, Y.L. Lin, i Y.C. Hsu. Data path allocation based on bipartite weighted matching. In *Proc. 27th ACM/IEEE Design Automation Conference*, pages 499–504, 1990.

[HE88] B.S. Haroun i M.I. Elmasry. Automatic synthesis of a multi-bus architecture for dsp. In *Proc. of ICCAD-88*, pages 44–47, 1988.

[Hea88] S. Heath. *VMEbus user's handbook.* CRC press, 1988.

[HG84] L.G. Heller i W. R. Griffin. Cascode voltage switch logic: A differential CMOS logic family. In *IEEE ISSCC Digest of Technical Papers*, Febrer 1984.

[Hir87] Masaharu Hirayama. A silicon compiler system based on asynchronous architecture. *IEEE Trans. on CAD*, Vol. CAD-6(3):297–304, May 1987.

[Hoa89] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall International, 1989.

[HS71] A. Hashimoto i J. Stevens. Wire routing by optimizing channel assignment within large apertures. In *Proc. of the 8th ACM/IEEE Design Automation Conference*, pages 155–169, 1971.

[Huf54] D.A. Huffman. *The Synthesis of Sequential Switching Circuits.* Number 257. J. Franklin Institute, March 1954.

[Huf57] D.A. Huffman. The design and use of hazard-free switching networks. *Journal of the Association for Computing Machinery*, 41(4):47–62, Gener 1957.

[JAMY87] D.S. Johnson, C.R. Aragon, L.A. McGeoch, i M. Yannakakis. Optimization by simulated annealing: an experimental evaluation. Parts I and II, AT & T Bell Laboratories, 1987. preprint.

[JM86] C. Jesshope i W. Moore, editors. *Wafer-Scale Integration.* Adam Hilger, 1986.

[JMP88] R. Jain, M. J. Mlinar, i A. Parker. Area-time model for synthesis of non-pipelined designs. In *Proc. ICCAD-88*, pages 48–51, 1988.

[Kel74] Robert M. Keller. Towards a theory of universal speed-independent modules. *IEEE Trans. on Computers*, Vol. C-23(1), January 1974.

[Kes93] J. Kessels. VLSI programming of a low-power error decoder for the dcc player. Technical Report 93/24, Universitat Politècnica de Catalunya/Dept. d'Arquitectura de Computadors, 1993. Proceedings of the ACiD-WG Workshop on Digital Signal Processing.

[KGV83] S. Kirkpatrick, C.D. Gelatt, i M.P. Vecchi. Optimization by simulated annealing. *Science*, pages 671–680, 1983.

[KKTV94] M. Kishinevsky, A. Kondratyev, A. Taubin, i V. Varshavsky. *Concurrent Hardware: The Theory and Practice of Self-timed Design.* Wiley series in parallel computing. Wiley, 1994.

[KM90] D. Ku i G. De Micheli. Relative scheduling under timing constraints. In *Proc. 27th. Design Automation Conference*, volume Vol. 78, pages 59–64, Jun 1990.

[KM92] D.C. Ku i G. De Micheli. *High Level Synthesis of ASICs Under Timing and Synchronization Constraints.* Kluwer Academic Publishers, 1992.

[KN92] G. Krishnamoorthy i J.A. Nestor. Data path allocation using an extended binding model. In *Proc. 29th ACM/IEEE Design Automation Conference*, pages 279–284, 1992.

[KP87] F.J. Kurdahi i A.C. Parker. REAL: A program for register allocation. In *Proc. of the 24th ACM/IEEE Design Automation Conference*, pages 210–215, 1987.

[KP90a] K. Küçükçakar i A.C. Parker. Data path tradeoffs using MABAL. In *Proc. 27th ACM/IEEE Design Automayion Conference*, pages 511–516, 1990.

[KP90b] K. Küçükçakar i A.C. Parker. MABAL: A software package for module and bus allocation. *International Journal of Computer Aided VLSI Design*, pages 419–436, 1990.

[KvBB+92] J. Kessels, K. van Berkel, R. Burgess, M. Roncken, i F. Schalij. An error decoder for the compact disc player as an example of VLSI programming. In *Proceedings of EDAC-92*, 1992.

[Lav92] Luciano Lavagno. *Synthesis and Testing of Bounded Wire Delay Asynchronous Circuits from Signal Transitions Graphs.* Phd thesis, University of California at Berkeley, 1992.

[LEG90] T.A. Ly, W. L. Elwood, i E.F. Girczyc. A generalized interconnect model for data path synthesis. In *Proc. 27th ACM/IEEE Design Automation Conference*, pages 168–173, 1990.

[LHL89] J-H. Lee, Y-C. Hsu, i Y-L. Lin. A new integer linear programming formulation for the scheduling program in data path synthesis. In *Proc. of the Int. Conference on Computer Aided Design*, pages 20–23, 1989.

[LL85] H.W. Leong i C.L. Liu. Permutation channel routing. In *Proc. of the Int. Conference on Computer Design*, pages 579–584, Octubre 1985.

[LM86] M. Lundy i A. Mees. Convergence of an annealing algorithm. *Math. Prog.*, (34):111–124, 1986.

[LWL85] H.W. Leong, D.F. Wong, i C.L. Liu. A simulated annealing channel router. In *Proc. of the Int. Conference on Computer Aided Design*, pages 226–229, Novembre 1985.

[Mag71] G. Magó. Realization methods for asynchronous sequential circuits. *IEEE Trans. on Computers*, c-20(3):290–297, 1971.

[Mar85] A.J. Martin. The probe: an addition to communication primitives. *Information Processing Letters*, (20):125–130, 1985.

[Mar86] Alain J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributing Computing*, Vol. 1:226–234, 1986.

[Mar92] A.J. Martin. Asynchronous datapaths and the design of an asynchronous adder. *Formal Methods in System Design*, 1(1):117–137, July 1992.

[MB59] D.E. Muller i W.C. Bartky. A theory of asynchronous circuits. *Annals of Computing Laboratory of Harvard University*, pages 204–243, 1959.

[MBL+89] A.J. Martin, S.M. Burns, T.K. Lee, D. Borkovic, i P.J. Hazewindus. The design of an asynchronous microprocessor. In C.L. Seitz, editor, *Advanced Research in VLSI: Proc. of the Decennial Caltech Conference on VLSI*, pages 351–373. MIT Press, 1989.

[MBM90] T.H.-Y. Meng, R.W. Brodersen, i D.G. Messerschmitt. A clock-free chip set for high-sampling rate adaptive filters. *Journal of VLSI Signal Processing*, (1):345–365, 1990.

[Men91] Teresa H. Meng. *Synchronization Design for Digital Systems.* Kluwer Academic Publishers, 1991.

[MFR85] C.E. Molnar, T.-P. Fang, i F.U. Rosenberger. Synthesis of delay-insensitive modules. In *Chapel Hill Conference on VLSI*, pages 67–86, Rockville, 1985. Computer Science Press.

[Mil65] R.E. Miller. *Switching Theory*, volume Vol. 2. Wiley and sons, 1965.

[MLD92] P. Michel, U. Lauther, i P. Duzy, editors. *The Synthesis Approach to Digital System Design.* Kluwer Academic Publishers, 1992.

[Moo92] Cho Woo Moon. *Synthesis and Verification of Asynchronous Circuits from Graphical Spedicication.* Phd thesis, University of California at Berkeley, 1992.

[Mor84] J.D. Morison. ELLA: Hardware description or specification. In *Proc. of the ICCAD*, pages 54–56, June 1984.

[MPC90] M. C. McFarland, A. C. Parker, i R. Camposano. The high-level synthesis of digital systems. In *Proc. of the IEEE*, volume Vol. 78, pages 301–318, Feb 1990.

[MRR+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, i E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, (21):1087–1092, 1953.

[MS86] C.A. Morgenstern i H.D. Shapiro. Chromatic number approximation using simulated annealing. Technical Report CS86-1, Department of Computer Science, The University of New Mexico, Albuquerque, 1986.

[MSB91] Cho W. Moon, Paul R. Stephan, i Robert K. Brayton. Synthesis of hazard-free asynchronous circuits from graphical specifications. In *Proc. of ICCAD-91*, pages 322–325, 1991.

[ND91] S.M. Nowick i D.L. Dill. Synthesis of asynchronous state machines using a local clock. In *Int. Conf. on Computer Design*, pages 192–196, 1991.

[NDDH93] S.M. Nowick, M.E. Dean, D.L. Dill, i M. Horowitz. The design of a high-performance cache controller: a case study in asynchronous synthesis. *Integration, the VLSI journal*, 15(3):241–262, October 1993.

[NSS85] S. Nahar, S. Sahni, i E. Shragowitz. Experiments with simulated annealing. In *Proc. of the 22nd Design Automation Conference*, pages 748–752, Juny 1985.

[OvG84] R.H.J.M Otten i L.P.P.P. van Ginneken. Floorplan design usign simulated annealing. In *Proc. of the Int. Conference on Computer Aided Design*, pages 96–98, Novembre 1984.

[Pan88] B.M. Pangrle. Splicer: A heuristic approach to connectivity binding. In *Proc. 25th ACM/IEEE Design Automation Conference*, pages 536–541, 1988.

[PC93a] E. Pastor i J. Cortadella. An efficient unique state coding algorithm for signal transitions graphs. In *Int. Conf. on Computer Design*, pages 174–177, 1993.

[PC93b] E. Pastor i J. Cortadella. Polynomial algorithms for the synthesis of hazard-free circuits from signal transitions graphs. In *Int. Conf. on Computer Aided Design*, pages 250–254, 1993.

[Pet62] C.A. Petri. *Kommunication mit Atomaten.* Phd thesis, Institut für Instrumentelle Mathematik, 1962.

[PG87a] B.M. Pangrle i D.D. Gajski. Design tools for intelligent silicon compilation. *IEEE Transactions on Computer-Aided Design*, vol. CAD-6(no. 6):1098–1112, 1987.

[PG87b] B.M. Pangrle i D.D. Gajski. Slicer: A state synthesizer for intelligent silicon compilation. In *Proc. IEEE Int. Conf. Computer Design*, Octubre 1987.

[PK87] P. G. Paulin i J. P. Knight. Force-directed scheduling in automatic data-path synthesis. In *Proc. 24th ACM/IEEE Design Automation Conference*, pages 195–202, Jun 1987.

[PK89a] P. G. Paulin i J. P. Knight. Force-directed scheduling for the behavioral synthesis of asic's. *IEEE Trans. on CAD*, pages 661–679, Jun 1989.

[PK89b] P. G. Paulin i J. P. Knight. Scheduling and binding algorithms for high-level synthesis. In *Proc. 26th ACM/IEEE Design Automation Conference*, pages 1–6, 1989.

[PK91] I-C. Park i C-M. Kyung. Fast and near scheduling in automatic data path synthsis. In *Proc. of the 28th. Design Automation Conference*, pages 680–685, June 1991.

[PKG86] P. G. Paulin, J. P. Knight, i E. F. Girczyc. Hal: A multi-paradigm approach to automatic data path synthesis. In *Proc. 23th ACM/IEEE Design Automation Conference*, pages 263–270, Jul 1986.

[PPM86] A.C. Parker, J. Pizarro, i M. Milnar. MAHA: a program for datapath synthesis. In *Proc. of the 23th Design Automation Conference*, pages 461–466, 1986.

[PRCB94] E. Pastor, O. Roig, J. Cortadella, i R.M. Badia. Analysis of petri nets using boolean manipulation. In *Int. Conf. on Petri Nets*, 1994.

[RJL92] M. Rim, R. Jain, i R. De Leone. Optimal allocation and binding in high-level synthesis. In *Proc. 29th ACM/IEEE Design Automation Conference*, pages 120–123, 1992.

[RY85] L.Ya. Rosenblum i A.V. Yakovlev. Signal graphs: From self-timed to timed ones. In *International Workshop on Petri Nets*, pages 199–206, 1985.

[SC93] F. Sánchez i J. Cortadella. Resource-constrained pipelining based on loop transformations. *Microprocessing and Microprogramming*, 38(1-5):429–436, 1993.

[Sec88] C. Sechen. *VLSI Placement and Global Routing Using Simulated Annealing.* Kluwer Academic Publishers, 1988.

[Sei80] C. L. Seitz. *System Timing, Chapter 7 in Introduction to VLSI Systems.* Mead & Conway, Addison-Wesley, 1980.

[SL89] K. Shankar i D. Lee. Build a VMEbus interface with PAL devices. *Electronic Design*, 37(15):55–62, July 1989.

[SMT+92] J. Septién, D. Mozos, F. Tirado, R. Hermida, i M. Fernández. Heuristics for branchd-and-bound global allocation. In *Proc. EURODAC 92*, pages 334–340, 1992.

[SSL+92] E.M. Santovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, i A. Sangiovanni-Vicentelli. SIS: A system for sequential circuit synthesis. Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720, 1992.

[SSV85] C. Sechen i A.L. Sangiovanni-Vincentelli. The timberwolf placement and routing package. *IEEE J. Solid State Circuits*, (20):510–522, 1985.

[ST91] D. Springer i D. E. Thomas. New methods for coloring and clique partitioning in data path allocation. *Integration, The VLSI journal*, Vol. 12(3), December 1991.

[Sut89] I.E. Sutherland. Micropipelines. *Comunications of the ACM*, Juny 1989.

[Tew89] S.K. Tewksbury. *Wafer-level integrated systems: implementation issues.* Kluwer Academic Publishers, 1989.

[TMB93] J.A. Tierno, A.J. Martin, i D. Borkovic. An asynchronous microprocessir in gallium arsenide. Technical Report cs-tr-93-38, California Institute of Technology, Novembre 1993.

[TS86] C.-J. Tseng i D.P Siewiorek. Automated synthesis of data paths in digital systems. *IEEE Transactions on Computer-Aided Design*, vol. CAD-5(no. 3):379–395, 1986.

[Ung69] S. H. Unger. *Asynchronous Sequential Switching Circuits.* Wiley Interscience, 1969.

[Ung71] S.H. Unger. Asynchronous sequential switching circuits with unrestricted input changes. *IEEE Trans. on Computers*, Vol. C-20(12), December 1971.

[vB92] Kees van Berkel. *Handshake circuits: an intermediary between communicating processes and VLSI.* Phd thesis, Technische Universiteit Eindhoven, May 1992.

[vBKR+91] K. van Berkel, J. Kessels, M. Ronckep, R. Saeijs, i F. Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. *Proc. of the European Conference on Design Automation*, pages 384–389, Feb 1991.

[VGCM92] P. Vanbekbregen, G. Goossens, F. Catthoor, i H.J. De Man. Optimized synthesis of asynchronous control circuits from graph-theoretic specifications. *IEEE Trans. on CAD*, November 1992.

[vGdG87] A. van Genderen i S. de Graaf. *SLS: Switch-Level Simulator User's Manual.* Dep. of Electrical Engineering, Delft University of Technology, 1987.

[vLA89] P.J.M. van Laarhoven i E.H.L. Aarts. *Simulated Annealing: Theory and Applications.* Kluwer Academic Publishers, 1989.

[VLGM92] P. Vanbekbergen, B. Lin, G. Goossens, i H. De Man. A generalized state assignment theory for transformations on signal transitions graphs. In *Int. Conf. on Computer Aided Design*, pages 112–117, 1992.

[WE85] N. Weste i K. Eshraghian. *Principles of CMOS VLSI Design. A System Perspective.* Addison-Wesley Publishing Company, 1985.

[WF83] D.F. Wann i M.A. Franklin. Asynchronous and clocked control structures for VLSI based interconnection network. *IEEE Trans. on Computers*, C-32, March 1983.

[WHAY87] T.E. Williams, M. Horowitz, R.L. Alverson, i T.S. Yang. A self-timed chip for division. In *Advanced Research in VLSI. Proceedings of the 1987 Stanford Conference*, pages 75–95, 1987.

[Whi84] S. White. Concepts of scale in simulated annealing. In *Proc. of the Int. Conference on Computer Design*, page 646, 1984.

[Wil91] T. Williams. Self-timed rings and their application to division. Technical Report CSL-TR-91-482, Computer Systems Laboratory, Stanford University, Maig 1991.

[WLL88] D.F. Wong, H.W. Leong, i C.L. Liu. *Simulated Annealing for VLSI Design.* Kluwer Academic Publishers, 1988.

[Wou] Woudsma, R. et al. One-dimensional linear picture transformer. US Patent No. 4,881,192.

Apèndix A

Taula de traduccions

Termes anglesos	Equivalència catalana
bundled data	*dades compactades*
control data flow graph	*graf de flux de dades i de control*
data flow graph	*graf de flux de dades*
data-path	*camí de dades*
delay insensitive	*insensible al retard*
dual-rail	*doble via*
force-directed scheduling	*planificació dirigida per forces*
graph colouring	*colorejat d'un graf*
hardware allocation	*assignació de recursos*
hardware selection	*selecció de recursos*
hazard	*risc*
hill-climbing	*escalada de cims*
layout synthesis	*síntesi de* layout
left-edge	*cantó esquerre*
list scheduling	*planificació per llistes*
operation scheduling	*planificació d'operacions*

Termes anglesos	Equivalència catalana
partitioning	*particionat*
precedence-constrained scheduling	*planificació amb precedències restringides*
placement	*emplaçament*
races	*curses*
relative scheduling	*planificació relativa*
resource binding	*associació de recursos*
resource allocation	*assignació de recursos*
routing	*connexionat*
self-timed	*autotemporitzat*
signal transition graph	*graf de transicions de senyals*
simulated annealing	*recuit simulat*
simulated evolution	*evolució simulada*
speed independent	*independent de la velocitat*
transistors sizing	*escalat de transistors*
technology mapping	*projecció a una tecnologia*
tristate	*tres-estats*

Apèndix B

Taula de sigles

Sigles	Nom complet
ALAP	*As Late As Possible*
ALU	*Arithmetic Logic Unit*
ASAP	*As Soon As Possible*
CDFG	*Control Data Flow Graph*
CL	*Controlador Local*
CLA	*Carry Look-ahead Adder*
CSC	*Complete State Coding*
DCVSL	*Differential Cascode Voltage Switch Level*
DFG	*Data Flow Graph*
ELS	*Event List Scheduling*
ELLAS	*Event List Look-Ahead Scheduling*
FC	*Free Choice*
FDLS	*Force Directed List Scheduling*
FDS	*Force Directed Scheduling*
FSM	*Finite State Machine*
GC	*Graf de Compatibilitat*
GI	*Graf d'Incompatibilitat*

Sigles	Nom complet
GLASS	*GLobal ASSignment*
NAE	*Nombre d'Arcs a Esborrar*
NVC	*Nombre de Veïns Comuns*
RCA	*Ripple Carry Adder*
SC	*Subgraf de Compatibilitat*
SDFG	*Scheduled Data Flow Graph*
SoG	*Sea of Gates*
STG	*Signal Transition Graph*
USC	*Unique State Coding*