UNIVERSITATEA "POLITEHNICA" TIMIŞOARA
Facultatea de Electronică şi Telecomunicaţii
Departamentul de Electronică Aplicată

TEZĂ DE DOCTORAT

elaborată în vederea obţinerii

Titlului de DOCTOR în specialitatea ELECTRONICĂ

de **Ing. Aurel-Ştefan GONTEAN**

Conducător ştiinţific
Prof. Dr. Ing. TIBERIU MUREŞAN

TIMIŞOARA
1998

CONTRIBUŢII LA PROIECTAREA OPTIMALĂ CU STRUCTURI LOGICE PROGRAMABILE

ANEXA 0

LISTA SIMBOLURILOR ŞI A PRESCURTĂRILOR UTILIZATE

A0.1. LISTA SIMBOLURILOR

B	numărul de biți ce se trunchiază (în capitolul 4)
CL	latura blocului logic, considerat rectangular (în capitolul 2)
C_T	numărul total de conexiuni rutate în FPGA (în capitolul 3)
D	densitatea pistelor în canalul de rutare (în capitolul 3)
F_c	flexibilitatea blocului C (în capitolul 3)
F_s	flexibilitatea blocului S (în capitolul 3)
k	numărul de intrări în LUT (în capitolul 2)
k	indice de frecvență (în capitolul 4)
L	numărul de biți la ieşirea ROM (în capitolul 4)
LC_l	lungimea unei conexiuni (în capitolul 3)
m	numărul de tabele LUT dintr-un bloc logic
M	numărul de biți ce se iau în considerare la intrarea ROM (în capitolul 4)
n	indice temporal (în capitolul 4)
N	numărul de biți ai acumulatorului (în capitolul 4)
N	dimensiunea FPGA (în capitolul 3)
$\overline{R}$	lungimea medie a unei conexiuni (în capitolul 3)
RP	dimensiunea unui switch de rutare (în capitolul 2)
R_p	rutabilitatea obținută practic (în capitolul 3)
R_t	rutabilitatea prezisă teoretic (în capitolul 3)
S_i	evenimentul reprezentat de rutarea cu succes a conexiunii C_i (în capitolul 3)
w	numărul de piste dintr-un canal (în capitolul 3)
X_1, X_2	evenimentele reprezentate de rutarea cu succes a conexiunii inițiale, respectiv a conexiunii finale (în capitolul 3)
ε_r	eroarea relativă de predicție (în capitolul 3)
λ	numărul de conexiuni ale unui bloc logic (în capitolul 3)
λ_g	parametru ce caracterizează distribuția Poisson pentru un canal de rutare (în capitolul 3)
$\Theta(n)$	faza la ieşirea acumulatorului (în capitolul 4)

A0.2. DICŢIONAR DE TERMENI ŞI ACRONIME

A

ABEL	software de dezvoltare pentru SLP produs de firma Data I/O.
AES	Automat Elementar Secvenţial Sincron.
ALU	*Arithmetic Logic Unit*, unitate aritmetică şi logică.
ASCII	*American Standard for Computer Information Interchange*, cod de reprezentare a principalelor caractere şi semne ortografice.
ASIC	*Application-Specific Integrated Circuit*, circuit integrat orientat pe aplicaţie.

B

BIOS	*Binary Input-Output System*, grup de rutine de intrare-ieşire aflate în memoria ROM a oricărui calculator compatibil IBM PC.
BL	Bloc logic.

C

CAD	*Computer Aided Design*, proiectare asistată de calculator.
CAE	*Computer Aided Engineering*, inginerie asistată de calculator.
CAL	*Configurable Array Logic*, matrice configurabilă.
Canal	porţiunea rectangulară dintre două rânduri sau coloane de blocuri logice; un canal constă din mai multe piste.
CC	Convertor de Cod.
CHS	*Configurable Hardware System*, sistem configurabil hardware.
CI	Circuite Integrate.
CIN	Circuite Integrate Numerice.
CLB	*Configurable Logic Block*, bloc logic configurabil.
CLS	*Configurable Logic Software*, resurse logice configurabile.
CNA	Convertor Numeric Analogic.
conexiune	o pereche de pini conectaţi electric.
CPLD	*Complex Programmable Logic Device*, PLD complex.

D

DIP	*Dual In Line Package*, tip de capsulă de plastic pentru circuite integrate.
DRAM	*Dynamic Random Access Memory*, memorie RAM dinamică.
DSP	*Digital Signal Processor*, procesor de semnal.
DUT	*Device Under Test*, circuit testat.

E

EDIF	*Electronic Design Interchange Format*, format special de stocare a informaţiei, referitoare la proiectele electronice.
EEPROM	*Electrically Erasable Programmable ROM*, EPROM ce se poate şterge electric.
EPLD	*Erasable Programmable Logic Device*, SLP ce poate fi şters şi reprogramat electric.
EPROM	*Erasable Programmable ROM*, ROM programabil electric.

F

FPAA	*Field Programmable Analog Array*, arie analogică programabilă.
FPGA	*Field Programmable Gate Array*, arie de porţi programabilă.
FPLA	*Field Programmable Logic Array*, arie logică programabilă.
FSM	*Finite State Machine*, vezi AES.
FSW	*Frequency Setting Word*, cuvânt de memorare a indicelui de frecvenţă *k*. Uneori *k* se substituie chiar cu FSW (în capitolul 4).

G

GAL	*Generic Array Logic*, structură logică programabilă introdusă de firma Lattice.

H

HAL	*Hardware Array Logic*, un GAL programat prin mască la producător.
HDL	*Hardware Definition Language*, limbaj de descriere a schemelor numerice.

I

IC	*Integrated Circuit*, vezi şi CI.
IOB	*Input/Output Block*, bloc de intrare-ieşire.
isp	*In System Programmable*, familie de SLP programabilă în circuit, introdusă de firma Lattice.

J

JEDEC	*Joint Electronic Device Engineering Council*, grup de firme producătoare de circuite şi echipamente ce a propus un format de comunicare între software-ul de dezvoltare şi un programator.
JTAG	Joint Test Action Group, grup de firme producătoare de circuite integrate ce au propus un nou standard de testare pe frontieră, devenit ulterior IEEE 1149.1.

L

LAB	*Logic Array Block*, bloc logic din componenţa CPLD Altera.
LCA	*Logic Cell Array*, bloc logic din componenta FPGA Xilinx.
LCD	*Liquid Crystal Display*, afişaj cu cristale lichide.
LFSR	*Linear Feedback Shift Register*, registru de deplasare cu reacţie liniară.
LSb	*Last Significant bit*, bitul cel mai puţin semnificativ.
LSB	*Last Significant Byte*, octetul cel mai puţin semnificativ.
LSFR	vezi LFSR
LSI	*Large Scale Integration*, integrare pe scară mare.

M

MIPS	*Millions of Instructions Per Second*, unitate de măsură pentru viteza de prelucrare a unui procesor.
MDPA	*Monochrome Display / Printer Adapter*, adaptor video livrat iniţial cu calculatoarele IBM PC; conţinea şi un port paralel de imprimantă.
MPGA	*Mask Programmable Gate Array*, SLP programabil prin mască.
MSb	*Most Significant bit*, bitul cel mai semnificativ.

MSB — *Most Significant Byte*, octetul cel mai semnificativ.
MSI — *Medium Scale Integration*, integrare pe scară medie.
MTBF — *Mean Time Between Failures*, timp mediu de funcţionare între două erori.

N

NMOS — tehnologie mai veche de realizare a circuitelor integrate numerice.

P

PAL — *Programmable Array Logic*, SLP de complexitate redusă cu aria de porţi SI programabilă şi aria de porţi SAU fixă.
PCMCIA — magistrală miniaturală utilizată pentru conectarea la note-book-uri.
PGA — *Pin Grid Array*, tip de capsulă.
pin — o intrare sau ieşire a unui bloc logic.
pistă — o legătură electrică ce traversează lungimea sau lăţimea unui canal; o pistă este compusă din unul sau mai multe segmente de lungimi diferite.
PLA — *Programmable Logic Array*, SLP de complexitate redusă cu aria de porţi SI programabilă şi aria de porţi SAU de asemenea programabilă.
PLCC — Tip de capsulă de plastic.
PLD — *Programmable Logic Device*, structură logică programabilă.
PLICE — *Programmable Low Impedance Circuit Element*, element antifuzibil elaborat de firma Actel.
PPI — *Parallel Programmable Interface*, circuit periferic utilizat ca interfaţă programabilă intrare-ieşire.
POST — *Power On Self Test*, rutine de autotestare aflate în BIOS şi care se execută la punerea sub tensiune a calculatorului.
PREP — *Programmable Electronics Performance Corporation*, grup de firme ce au elaborat un set de teste pentru evaluarea independentă a SLP-urilor.
PROM — *Programmable Read Only Memory*, memorie ROM programabilă.

Q

QFP — *Quad Flat Pack*, tip de capsulă de plastic.

R

RAM — *Random Access Memory*, memorie ce poate fi citită şi scrisă.
R&D — *Research and Development*, cercetare şi dezvoltare.
RISC — *Reduced Instruction Set Computer*, calculator cu un număr redus de instrucţiuni microprogramate.
ROM — *Read Only Memory*, memorie ce poate fi doar citită.

S

SCLK — *Serial Clock*, intrare serială de tact.
SDI — *Serial Data Input*, intrare serială de date.
SDO — *Serial Data Output*, ieşire serială de date.
segment — o legătură directă, componentă a unei conexiuni.
SLP — Structură Logică Programabilă.
SNF — Sinteză (sintetizor) Numeric de Frecvenţă.
SNR — *Signal to Noise Ratio*, raport semnal zgomot.

SRAM	*Static Random Access Memory*, memorie RAM statică.
SPP	*Standard Parallel Port*, portul standard de imprimantă la PC.
SSI	*Small Scale Integration*, integrare pe scară mică.
switch	o resursă logică utilizată pentru conectarea a două segmente.

T

TTL	*Transistor-Transistor Logic*, tehnologie bipolară de realizare a circuitelor integrate numerice.

U

UV	Ultraviolet.

V

VCO	Voltage Controlled Oscillator, oscilator comandat în tensiune.
VHDL	*Very High-Level Hardware Definition Language*, limbaj structurat de descriere funcțională a schemelor electronice.
VLSI	Very Large Scale Integration, integrare pe scară foarte mare.

INTRODUCERE

Denumirea de *Structuri Logice Programabile* (SLP) a fost introdusă pentru prima dată în [Ştefan83] de către dl. Prof. Dr. Ing. Tiberiu Mureşan şi reprezintă terminologia românească cea mai potrivită pentru a desemna o familie de circuite integrate cu resurse ce pot fi interconectate conform dorinţei utilizatorului. Deşi gândite la început doar ca circuite numerice, este tot mai frecventă în literatură prezentarea unor realizări de circuite mixte, care conţin şi resurse analogice. Cu atât mai mult denumirea introdusă anterior îşi menţine actualitatea, deoarece nu face referire la caracterul numeric sau analogic al resurselor oferite.

Evoluţia în sine a vânzărilor de logică programabilă care se apropie în acest an de impresionanta cifră de 3 miliarde de dolari (numai circuitele FPGA depăşind un miliard de dolari) prezintă şi o altă importanţă pentru România. În dorinţa de punere în evidenţă a creativităţii native, de această tehnologie de vârf pot beneficia în primul rând o serie de firme mici, de 2-5 persoane, care pot folosi cu succes un calculator Pentium şi o platformă software accesibilă pentru a implementa proiecte cu SLP. Această tehnologie nu necesită deci investiţii costisitoare, ci experienţă şi inteligenţă, putându-se proiecta circuite complexe pentru nevoile interne, dar mai ales externe, prin aceasta obţinându-se o serie de beneficii evidente.

Pentru învăţământul academic această opţiune oferă suplimentar avantajul demonstrării studenţilor a posibilităţii parcurgerii cu succes în laborator a tuturor etapelor necesare implementării unui proiect, pornind de la idee, schemă, simulare şi continuând cu implementarea fizică şi verificarea după programare, prin aceasta realizându-se efectiv legătura dintre teoria universitară şi practica tehnologiei de nivel foarte ridicat, atât de necesară astăzi.

Principala problemă avută la elaborarea acestei lucrări este consecinţa actualităţii domeniului ales. La redactarea finală a lucrării accentul a glisat spre circuite care nici măcar nu existau în momentul alegerii temei (1990). Noutatea temei este demonstrată şi de absenţa cvasitotală de bibliografie în limba română în această direcţie. Din acest motiv, am preferat un drum mai anevoios, publicând mai multe lucrări în domeniu, pentru a mă putea concentra în final asupra acestei redactări.

Materialul din această teză este reflectat parţial în două cărţi: [Mureşan96] - la care sunt coautor şi [Gontean97] la care sunt autor principal şi de două lucrări publicate în străinătate ([Gontean98a] şi [Gontean98b]) la care sunt unic autor, la prestigioasa manifestare PDS'98 care are drept subiect tocmai SLP.

[Mureşan96] este o carte clasică de aplicaţii a circuitelor numerice, ce umplea un gol în literatura de specialitate din ţara noastră (o asemenea lucrare nu a mai fost publicată de 17 ani). Aici apare în premieră în literatura noastră un capitol dedicat integral structurilor logice programabile, datorat preocupărilor autorului în această direcţie.

[Gontean97] este prima lucrare din literatura română care tratează integral subiectul structurilor logice programabile, fiind în acelaşi timp una din puţinele lucrări de pe mapamond care abordează *simultan* subiectele PLD, CPLD şi FPGA.

Cele mai importante rezultate obţinute de autor în cercetarea sa au fost publicate în sinteză cu ocazia Conferinţei Internaţionale PDS'98 (Programmable Devices and Systems) desfăşurată în Polonia în februarie 1998. Organizată de prestigioasele Universităţi din Gliwice

(Polonia) şi Ostrava (Cehia) şi de Secţiunea Poloneză a IEEE, această manifestare a avut drept unic subiect structurile logice programabile şi aplicaţiile lor.

Implementarea unui proiect complex, testarea, simularea şi verificarea concluziilor presupuse de rutare reprezintă substanţa primei lucrări ([Gontean98a] Gontean A., *Mathematical Model for Predicting Routing Capabilities in FPGAs*, Proceedings of PDS'98, pag.129-134).

Prezentarea realizării originale a unui programator universal de structuri logice programabile este obiectul celei de-a doua lucrări ([Gontean98b] Gontean A., *Low Cost, High Functionality Programmer*, Proceedings of PDS'98, pag.207-212).

* * *

Pentru coerenţă şi fluiditatea înţelegerii, lucrarea este precedată de o listă a simbolurilor şi un index alfabetic al acronimelor şi prescurtărilor utilizate.

Fiecare capitol este încheiat de un set de concluzii personale cu evidenţierea contribuţiilor originale ale autorului şi de comentarea principalelor referinţe bibliografice la care se face apel în cadrul capitolului.

În primul capitol sunt prezentate într-o manieră originală strategiile de implementare a sistemelor numerice. Alături de evidenţierea celor trei abordări posibile de implementare (software, hardware, respectiv logica programabilă) se introduc cu acest prilej patru etape de dezvoltare a circuitelor integrate numerice, delimitate convenţional de autor la o perioadă de 10 ani. O propunere originală de clasificare a structurilor logice programabile continuă natural elementele prezentate anterior.

Capitolul 2 debutează cu o scurtă sinteză originală asupra stadiului SLP moderne. O atenţie deosebită este acordată în mod firesc, circuitelor FPGA moderne, a căror dezvoltare din ultimii 10 ani a determinat o cifră de vânzări comparabilă cu a familiei de microcontrolere 8051. Influenţa arhitecturii blocului logic asupra performanţelor obţinute este un alt subiect ce permite autorului propriul studiu şi obţinerea unor concluzii interesante şi a unor contribuţii originale. Caracteristicile de metastabilitate a SLP fac obiectul unui subcapitol separat şi sunt sprijinite de un montaj experimental original de măsură a performanţelor de metastabilitate. Alături de concluziile finale şi contribuţiile originale, în finalul acestui capitol sunt incluse şi o serie de aprecieri privind evoluţia şi perspectivele FPGA.

Cel de-al treilea capitol este dedicat în întregime unui model matematic de rutare pentru structurile FPGA simetrice. Modelul structural introdus nu restrânge generalitatea rezultatelor la SLP, ci este aplicabil oricăror structuri simetrice care respectă condiţiile de arhitectură impuse. Rezultatele experimentale validează riguros presupunerea de distribuţie matematică Poisson făcută în cadrul acestui capitol pentru densitatea canalului din structură. Prin aceasta se pune la îndemâna utilizatorului o metodă ce prezice posibilitatea implementării unui proiect concret într-o arhitectură dată, fiind cunoscut faptul că acest lucru nu este realizat de nici un ruter disponibil în prezent pe piaţă. În finalul capitolului sunt prezentate o serie de concluzii privind influenţa topologiei resurselor de rutare asupra performanţelor obţinute şi este ilustrată grafic dependenţa rutabilităţii de diverşi parametri.

Capitolul 4 este dedicat implementării originale a unui sintetizor numeric de frecvenţă, prezentând o suită de variante concrete propuse de autor. Pe lângă modernitatea subiectului ales, complexitatea temei abordate permite concluzii obiective care să valideze ipotezele din

capitolul anterior. O caracteristică importantă a FPGA Xilinx permite autorului să elaboreze o schemă de implementare de foarte mare viteză cu aceste circuite. Diferitele variante de ASIC implementate sunt prezentate împreună cu rezultatele simulării. Concluziile şi contribuţiile originale de la sfârşitul capitolului permit introducerea unei sinteze a principalelor rezultate obţinute.

Programatoarele de concepţie originală prezentate în *capitolul 5* reprezintă un alt domeniu de interes abordat de autor. Sunt prezentate două asemenea programatoare, unul dedicat şi altul universal, implementate în cadrul activităţii de cercetare din cadrul Departamentului de Electronică Aplicată. Pe lângă componenta hardware aceste programatoare au fost echipate şi cu o interfaţă utilizator, ce poate rula atât din mediul DOS, cât şi de sub Windows. Ambele programatoare se conectează la PC pe portul paralel, ceea ce conferă acestei opţiuni o serie de avantaje, descrise pe larg în capitol.

Capitolul 6 prezintă sinteza contribuţiilor originale şi redă o serie de concluzii finale, de ansamblu.

Bibliografia extinsă de la sfârşitul tezei poate prezenta un bun punct de plecare pentru studii ulterioare. În cele 143 de titluri prezentate sunt incluse câteva lucrări de referinţă, punctate de fiecare dată la referinţele bibliografice ce încheie fiecare capitol. Cercetarea bibliografică a suferit mutaţii profunde în ultimii 5 ani şi de aceea am considerat utilă prezentarea celor mai utile adrese de Internet care conţin pagini de Web referitoare la subiectul SLP.

Programul de autotestare al programatorului universal, prezentat în *Anexa 1* este doar o mică parte a software-ului realizat, dar prefigurează o direcţie asupra căruia autorul a dorit să insiste: testarea funcţională. Interfaţa grafică utilizator, modulele software de citire şi programare nu au fost incluse în primul rând datorită volumului (peste 7000 de linii de cod).

Variantele de implementare a sintetizorului de frecvenţă discutat în capitolul 4 se regăsesc în detaliu în *Anexa 2*. Aici sunt cuprinse schemele electrice, vectorii de test, simularea funcţională şi principalele etape ale rutării efectuate pe reţeaua Sun a Universităţii din Preston, Anglia.

Lucrarea conţine 6 capitole şi 3 Anexe cuprinzând 217 pagini, 101 figuri şi 46 de planşe în Anexa 2, precum şi 143 referinţe bibliografice, dintre care 15 personale ca unic sau prim autor şi 2 în colaborare, dintre care amintesc cele două cărţi şi cele două lucrări prezentate şi publicate în străinătate.

* * *

Prezenta lucrare reprezintă rodul preocupărilor autorului în domeniul modern şi de mare interes al structurilor logice programabile, manifestate printr-un efort susţinut desfăşurat pe perioada ultimilor 8 ani, în cadrul colectivului de Circuite Integrate Numerice din cadrul Facultăţii de Electronică şi Telecomunicaţii, Universitatea "Politehnica" din Timişoara. Experienţa autorului în acest domeniu a fost sprijinită de colaborarea iniţiată de binecunoscuta firmă Texas Instruments în cadrul programului "Parteneriat pentru Inovare", la care sunt membru din 1994. Cele două burse de trei luni în cadrul unui program Tempus, una în Germania şi cealaltă în Anglia mi-au permis să iau contact nemijlocit cu realizările colegilor din lumea occidentală şi să pot lucra, chiar dacă pe o perioadă relativ scurtă pe un echipament hardware performant şi cu software profesional.

Doresc să mulţumesc şi pe această cale conducătorului ştiinţific, domnul profesor doctor inginer Tiberiu MUREŞAN pentru atenta şi competenta îndrumare pe care mi-a acordat-o pe tot parcursul pregătirii la doctorat. Afirm aici cu toată convingerea că această teză nu ar fi fost posibilă fără sprijinul oferit de Domnia sa, combinat cu imbolduri mobilizatoare în momentele strategice ale pregătirii.

O listă lungă de persoane au avut o contribuţie mai mare sau mai mică la finalizarea acestei lucrări, şi de aceea pentru a nu nedreptăţi pe cineva, voi mulţumi aici tuturor colegilor din Facultatea de Electronică şi Telecomunicaţii care m-au ajutat şi nu în ultimul rând familiei care a ştiut să înţeleagă importanţa acestui moment şi să nu-mi ceară lucruri imposibile în toată această perioadă.

Aurel GONTEAN

Timişoara, 28.03.1998

CAPITOLUL 1

STRATEGII DE IMPLEMENTARE A SISTEMELOR NUMERICE

1.0. INTRODUCERE

În acest prim capitol sunt prezentate succint principalele modalități de implementare a sistemelor numerice, ca preambul necesar pentru introducerea logicii programabile drept alternativă avantajoasă începând cu acest deceniu. Etapele de dezvoltare a circuitelor integrate numerice împreună cu metodele de proiectare în tehnica numerică sunt tratate pe scurt în continuare, cu accent pe implicațiile asupra structurilor logice programabile, în special și a FPGA, în particular. Acest prim capitol se încheie cu o propunere de clasificare a structurilor logice programabile. O asemenea clasificare aduce o serie de clarificări utile pentru inginerul proiectant, înscriindu-se în preocuparea constantă a autorului de a descoperi căile prin care se poate găsi SLP-ul cel mai potrivit unei aplicații date.

1.1. IMPLEMENTAREA SISTEMELOR NUMERICE

În majoritatea situațiilor, un sistem numeric este reprezentat cel mai bine de un *algoritm*, respectiv de o descriere formală a funcționării acestuia. Istoric, au existat două abordări distincte pentru implementarea unui sistem numeric:

- *software*, bazată pe arhitectura de calculator von Neumann, în care algoritmul este transformat într-o secvență de cod și executat de un procesor (în figura 1.1, M este memoria principală, A este acumulatorul, R este un registru general, iar ALU este unitatea aritmetico-logică);
- *hardware*, în care algoritmul este transpus direct "în siliciu".

La acest nivel (maxim) de abstractizare decizia între cele două abordări este de o importanță crucială, afectând performanțele, gabaritul și costul produsului rezultat. De exemplu, alegerea unei arhitecturi pipeline poate aduce îmbunătățiri ale performanțelor de viteză de sute de procente față de varianta clasică, pe când alegerea între diversele tehnologii posibile (o decizie pe câteva niveluri ierarhice de proiectare mai jos) poate suplimenta performanțele cu cel mult câteva zeci de procente.

Pentru evaluarea diferitelor posibilități de implementare se va studia un proiect simplu de însumare a produsului elementelor a doi vectori, conform relației:

$$s = \sum_{i=0}^{n-1} a_i \cdot b_i \qquad (1.1)$$

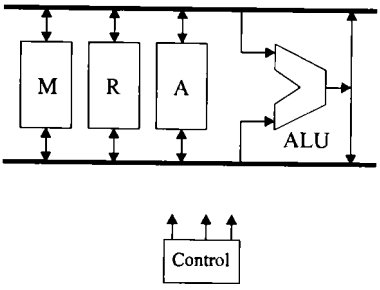


Figura 1.1. Calculator cu arhitectură von Neumann.

1.1.1. ABORDAREA 1: SOFTWARE

Pentru calculul lui *s* trebuie scris un program, de exemplu:

```
s = 0
for i = 0 to n-1 do
    s = s + a[i]*b[i]
```

Compilând programul pe calculatorul din figura 1.1 şi presupunând o metodă de adresare bazată cu deplasament, rezultă o posibilă ordonare a datelor în memorie (figura 1.2).

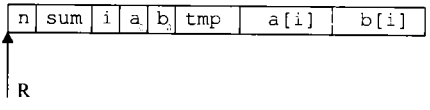


Figura 1.2. Organizarea datelor în memorie.

În urma compilării, poate rezulta o secvență în cod maşină, care transcrisă în pseudolimbaj de asamblare se poate prezenta astfel:

```
        CLR  A
        MOV  R, adresa      ;R=adresa datelor
        MOV  (R+2),A        ;s=0
        MOV  (R+3),A        ;i=0
bucla:  MOV  A,(R+4)        ;A=a[i]
        MOV  (R+6),A        ;tmp=a[i]
        MOV  A, (R+5)       ;A=b[i]
        MLT  A,(R+6)        ;R=b[i]*tmp
        ADD  A,(R+2)        ;A=s+a[i]*b[i]
        MOV  (R+2),A        ;s=s+a[i]*b[i]
        MOV  A,(R+3)        ;A=i
        INC  A              ;A=i+1
        MOV  (R+3),A        ;i=i+1

        CMP  A,(R+1)        ;? i = n
        JNE  bucla          ;dacă NU, salt
```

Nu există nici o restricție deosebită privitoare la mediul de programare sau la performanțele calculatorului utilizat.

1.1.2. ABORDAREA 2: HARDWARE

Pentru implementare (figura 1.3) se pot utiliza registre de memorare pentru cei doi vectori a[i] şi b[i], unităţi aritmetice pentru produs şi sumă, registre pentru rezultatele parţiale şi suma finală. Datele pot fi reprezentate serie sau paralel, în funcţie de viteza de prelucrare necesară şi de costul estimat pentru proiect.

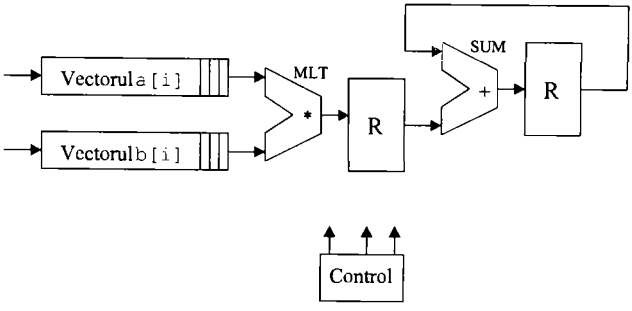


Figura 1.3. Procesarea hardware a sumei de vectori.

1.1.3. ABORDAREA 3: LOGICĂ PROGRAMABILĂ

Chiar din acest exemplu de complexitate redusă se pot deduce caracteristicile celor două abordări prezentate. În abordarea software, s-a stabilit un canal simplu de date într-un calculator modest (memorie, un acumulator, un registru general, ALU) controlat de un mecanism complex subordonat codului compilat. În abordarea hardware, canalul de date este mai complicat, dar este controlat de un AES simplu. Se poate considera abordarea hardware ca un *calculator dedicat aplicaţiei*, care execută rapid un algoritm specific, cu preţul construirii măcar al unui exemplar din aparat. Soluţia software este mult mai lentă, dar ieftină, deoarece utilizează un calculator din producţia de masă (considerat deja un bun de larg consum).

O dată cu apariţia logicii programabile reutilizabile a devenit posibilă *a treia cale*, în care AES dedicate pot fi construite folosind circuite produse în serie mare, deci ieftine. Abordarea logică programabilă îmbină avantajele hardware (viteză ridicată) cu implementarea directă a algoritmului (prin software), utilizând *circuite şi echipamente relativ ieftine*.

1.1.4. COMPARAŢIE ÎNTRE DIFERITELE POSIBILITĂŢI DE ABORDARE

În tabelul 1.1 sunt comparate sintetic principalele caracteristici ale celor trei variante de abordare discutate anterior. În prezent este comună întrepătrunderea celor trei abordări pentru obţinerea unui raport performanţe/preţ cât mai bun. De exemplu, abordarea software se poate îmbunătăţi hardware prin adăugarea unui coprocesor şi sporirea în consecinţă a vitezei de calcul. Odată cu perfecţionarea în continuare a tehnologiei de producţie şi mai ales a suportului pentru proiectare cu logică programabilă, se poate estima că *în mai puţin de 10 ani, această abordare va predomina cel puţin în implementarea aplicaţiilor la cerere*.

Tabelul 1.1

Modalităţi de implementare a sistemelor numerice

	Software	Hardware	Logică programabilă
Cerinţe de spaţiu	Medii, echipament	Reduse, CI	Reduse, echipament
Viteza de lucru	Mică	Mare	Mare
Eficienţa arhitecturii	Fixă	Configurabilă	Configurabilă
Durata implementării	Mică	Mare	Mică-Medie
Preţ	Redus	Ridicat	Redus
Reprezentarea datelor	Lungime fixă	Lungime variabilă	Lungime variabilă
Reutilizare	DA	NU	DA

1.2. PRINCIPALELE ETAPE DE DEZVOLTARE A CIRCUITELOR INTEGRATE NUMERICE

Procesul tehnologic de fabricare al unui echipament parcurge mai multe faze, dintre care două sunt esenţiale: implementarea utilizând componente semiconductoare şi echiparea plăcilor cu circuit imprimat. Studiind dezvoltarea industriei electronice se pot defini *etapele* de dezvoltare pentru circuitele numerice şi implicit pentru sisteme, rezumate în tabelul 1.2. Aceste etape, convenţional desemnate ca durată la un deceniu, se suprapun parţial.

Tabelul 1.2

Etape de dezvoltare a CIN

	1961-1970 **SSI-MSI**	1971-1980 **LSI-VLSI**	1981-1990 **ASIC**	1990-2000 **SLP**
Componente	Familii logice	µP pe 8 biţi, RAM	µP pe 32 biţi, celule standard	CPLD, FPGA
Complexitate (Nr. de porţi echivalente)	$\approx 10^2$	$\approx 10^4$	$\approx 10^5$	$\approx 10^4$
Exemplu tipic	TI TTL CD 4000	I 8080, 4164	I 80486 Arii de porţi	Flex 7000, Xilinx 4000, Act 3
Aria medie a plăcii echipate	< 75 cm^2	< 1500 cm^2	< 1500 cm^2	< 1500 cm^2

1.2.1. ETAPA FAMILIILOR LOGICE

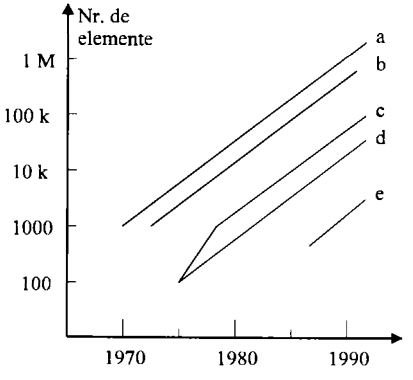
Anii '60 şi o parte din anii '70 au fost profund marcaţi de seriile de circuite integrate TTL introduse de Texas Instruments [TI 88], respectiv de seria 4000 de către National Semiconductor. Aceste circuite SSI şi MSI ofereau resurse logice sub formă de porţi, bistabile, registre, numărătoare, ALU, etc. Dimensiunile tipice ale plăcii echipate se păstrau relativ reduse datorită multiplelor legături între circuite şi tehnologiei de fabricaţie pentru circuitele imprimate.

1.2.2. ETAPA COMPONENTELOR LSI ŞI VLSI

Anii '70 au fost inauguraţi de către inventarea memoriei RAM dinamice o dată cu circuitul 1103 introdus de firma Intel şi a primului microprocesor, Intel 4004 şi s-au încheiat cu răspândirea microprocesoarelor pe 16 de biţi, de exemplu Intel 80286 şi Motorola 68000. Progresul tehnologic a condus la dezvoltarea masivă a celor două domenii majore ale pieţii de circuite integrate şi la proliferarea altor segmente de piaţă, cum ar fi SRAM sau microcontrolere, etc. Componentele din etapa precedentă au continuat să fie utilizate pentru interfaţarea circuitelor complexe, iar plăcile echipate au devenit mai mari, permiţând implementarea unui sistem complet. Prin aceasta se reducea costul, se mărea fiabilitatea şi se eliminau constrângerile legate de interconexiunile din fundul de sertar.

1.2.3. ETAPA ASIC

Introducerea ASIC-urilor la începutul anilor '80 a fost determinată de dorinţa de a înlocui multitudinea de circuite de interfaţă, comandă şi control dintre cipurile VLSI cu resurse logice disponibile unitar. În această etapă sunt utilizate curent celulele standard, MPGA şi primele structuri logice programabile. Plăcile echipate îşi păstrează dimensiunile mari, dar oferă multe facilităţi suplimentare, capacităţi de memorie sporite, etc.



LEGENDĂ

	Tip de circuit	Element
a	DRAM	Biţi
b	SRAM	Biţi
c	CI numeric	Tranzistoare
d	Arii de porţi	Porţi
e	Logică programabilă	Celule

Figura 1.4. Progresul tehnologic după legea Moore.

Pe măsura perfecţionării tehnologiei, circuitele VLSI au evoluat prin creşteri exponenţiale în densitatea de integrare şi liniare în timpii de propagare. Binecunoscuta lege a lui Moore, enunţată la începutul anilor '60 îşi păstrează incredibil de bine valabilitatea şi în prezent: "*numărul de tranzistoare per circuit integrat se dublează la fiecare 12...18 luni*". Circuite ca memoriile, microprocesoarele, ariile de porţi şi FPGA-urile se integrează foarte bine în această lege, de vreme ce mărimi ca numărul de biţi (capacitatea memoriei), numărul de porţi, respectiv de celule a crescut de asemenea exponenţial (figura 1.4).

Este extrem de interesant de subliniat că există un prag (în jurul cifrei de 1000 de elemente utile, tranzistoare, porţi, biţi, celule) pentru care un tip nou de circuit se impune pe piaţă. Exemplele care vin să sprijine această afirmaţie sunt multiple, de pildă ariile de porţi introduse de firma Ferranti (câteva sute de porţi per cip) nu s-au bucurat de succes comercial, pe

când ariile firmei LSI Logic (câteva mii de porți) au antrenat vânzări de peste un miliard de dolari.

1.2.4. ETAPA LOGICII PROGRAMABILE

Dezvoltarea SLP a devenit explozivă odată cu introducerea PLD CMOS de complexitate medie și mai ales după răspândirea FPGA reprogramabile în tehnologie SRAM [Gontean92b, Gontean92c]. În această etapă, un sistem numeric tipic conține un procesor, memorie, SLP pentru comandă și control și eventual un procesor special suplimentar (de exemplu un DSP). Plăcile echipate, de dimensiuni similare cu cele din etapa anterioară oferă posibilități multiple și facilități deosebite cum ar fi testarea pe frontieră [Gontean94a, Gontean94b].

1.3. METODE ȘI TEHNICI DE PROIECTARE

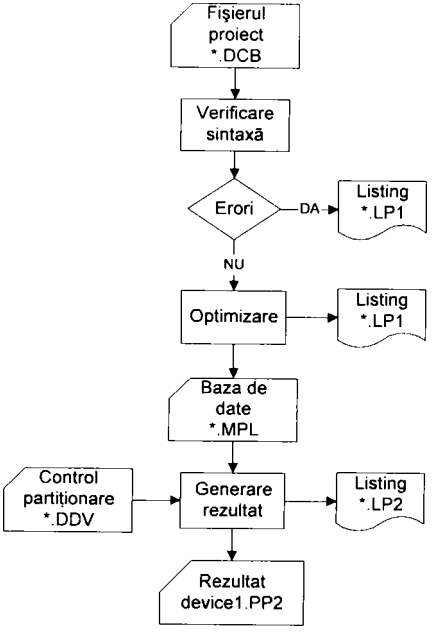


Figura 1.5. Proiectarea tipică cu SLP.

Pachetul de programe LOG/iC, dezvoltat de firma ISData GmbH din Germania, este unul din cele mai perfecționate instrumente CAD de proiectare cu PLD. Facilitățile incluse au sporit considerabil de-a lungul timpului, în prezent LOG/iC asigurând maximum de flexibilitate posibil oferit de un CAD *orientat pe text*. Se pot partiționa *automat* proiectele, fiind permisă introducerea datelor în format standard (ecuații logice, tabele de adevăr), format FSM, format orientat pe niveluri ierarhice (pe blocuri structurate) și opțional scheme electrice din alte CAD-uri prin intermediul listelor de legături. Sunt disponibile module separate pentru:

- testare (generare automată de vectori de test și un simulator performant);
- proiectare cu arii de porți.

În figura 1.5 sunt prezentate etapele de proiectare cu pachetul LOG/iC și principalele fișiere implicate în acest proces. În [Gontean97] am prezentat o serie de aplicații dezvoltate cu acest program.

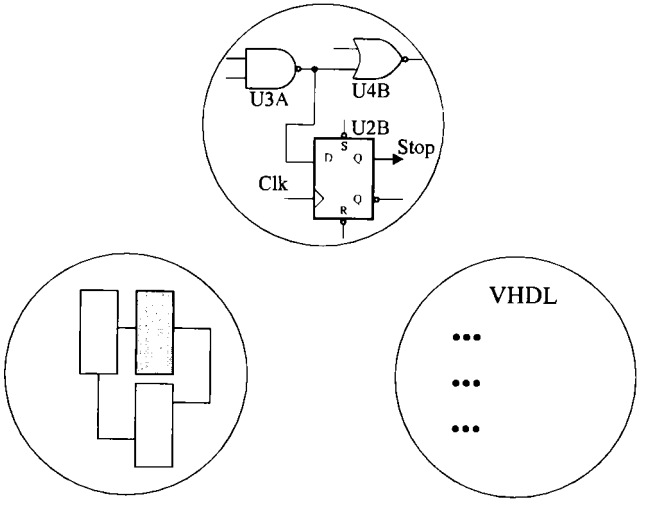
1.3.1. METODE DE DESCRIERE PENTRU UN SISTEM NUMERIC

În prezent sunt utilizate mai multe metode de descriere a unui proiect, la acest subiect persistând unele confuzii. Progresul tehnologic și larga utilizare a suportului CAD au determi-

nat evoluția de la simpla descriere a circuitului, pornind de la elementele sale (tranzistoare, rezistoare, etc) la baze de date care cuprind:

- scheme electrice structurate ierarhic;
- limbaje HDL;
- modele de simulare;
- descrierea amplasamentului;
- vectori şi metode de test.

Descriere structurală
Componente: module, simboluri
Legături: conexiuni, nume
Notații: scheme electronice, HDL



Descriere fizică
Componente: blocuri, celule
Legături: conexiuni
Notații: amplasament, harta fuzibilelor

Descriere funcțională
Componente: module
Legături: nume
Notații: limbaje specializate

Figura 1.6. Descrierea clasică a unui sistem numeric.

Pentru descrierea fără ambiguități a unui proiect datele trebuie organizate în trei *domenii*: structural, funcțional şi fizic, simultan cu separarea clară dintre *logica* şi *notarea* proiectului. În figura 1.6 este prezentată o diagramă clasică care identifică elementele specifice fiecărui domeniu. La nivelul logic, o descriere structurală poate fi alcătuită dintr-o schemă bloc sau electrică, o descriere funcțională printr-un set de ecuații booleene, iar cea fizică prin harta fuzibilelor dintr-un PAL. O memorie ROM este reprezentată ca un bloc în schema electrică (mod de notare), conținutul său fiind cuprins într-un tabel de adevăr (la nivelul logic).

Tradițional, proiectarea în electronică s-a asemănat foarte mult cu principiul *"divide et impera"* în care un sistem amplu era partiționat repetat în mai multe subsisteme mai simple, până când acestea deveneau suficient de simple pentru descrierea cu un anumit nivel de abstractizare [Bell71], [Bell78]. În acest caz, nivelul conceptual de abstractizare include circuite

logice, mecanisme de transfer între registre, comunicația dintre procesor și memorie. Abstractizarea la nivel fizic include componente semiconductoare, plăci de circuit imprimat, conectică de fund de sertar. O reprezentare simplă ca cea din figura 1.6 nu mai satisface cerințele descrierii atâtor niveluri și trebuie de aceea modificată.

O nouă metodă de reprezentare, propusă în [Gajski88], conține un graf relațional structurat pe trei axe, de forma literei Y (figura 1.7). Fiecare braț al grafului este scalat conform nivelului proiectării. În această diagramă, diferitele etape ale proiectării se pot reprezenta ca tranziții de la un punct de pe un braț spre alt punct din alt braț, rezultând în final o curbă spiralată care converge spre originea grafului.

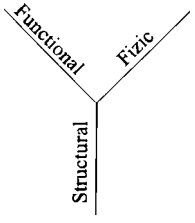


Figura 1.7. Diagrama Gajski-Kuhn.

Domeniul structural descrie modul de interconectare a diferitelor elemente, fiind comun mai multor ramuri ale ingineriei. Un obiectiv mai degrabă vag decât precis este de a crea o arhitectură elegantă care asigură sistemului implementat caracteristici de performanță ridicată la un cost redus printr-o abordare simplă, dar eficace. În cazul FPGA, domeniul structural poate fi implementat cu unelte specifice pentru desenarea schemelor electrice.

Domeniul fizic presupune furnizarea de date suficiente pentru fabricarea și testarea unui proiect. Pentru FPGA aceasta se traduce prin definirea funcțiilor blocurilor logice și a căilor de rutare pentru interconexiuni, similar cu producerea unui amplasament (layout) în VLSI. Uneltele utilizate sunt editoare simbolice, generatoare de vectori de test, software de plasare și rutare.

Domeniul funcțional implică o cale de a descrie modul de funcționare al proiectului. Poate fi cuprins în modelul de simulare, HDL, într-un set de ecuații logice sau într-o tabelă de adevăr. Schemele bloc sau electrice pot deveni descrieri funcționale utilizând convenții simple de desenare. De obicei implementările SLP sunt descrise în domeniul funcțional de limbaje dedicate (Abel, Synopsis, ViewLogic, etc).

1.3.2. PROIECTAREA IERARHICĂ

Proiectarea ierarhică a fost "redescoperită" la sfârșitul anilor '70, ca o necesitate de a rezolva serioasele stagnări apărute la dezvoltarea circuitelor VLSI dedicate. Renunțând la conceptul *"divide et impera"*, au reapărut idei ca modularizarea, proiectarea recursivă și partaja-

rea informațiilor pentru a ușura sarcina proiectării VLSI. Nivelul de complexitate atins de FPGA moderne este comparabil cu cel al circuitelor VLSI din acea perioadă, de aceea metodele de proiectare dezvoltate pentru circuitele VLSI își păstrează actualitatea. O propunere de reprezentare conceptuală spațială care să evidențieze aceste tendințe, pornind de la diagrama Gajski-Kuhn este ilustrată în figura 1.8.

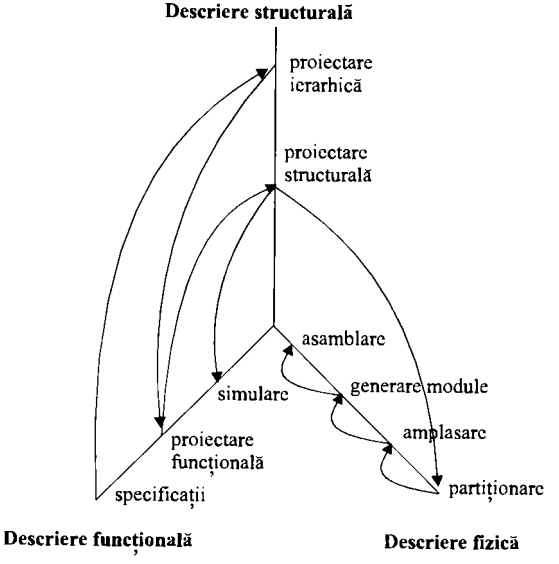


Figura 1.8. Propunere de reprezentare spațială.

1.3.3. PROIECTAREA INDEPENDENTĂ DE TEHNOLOGIE

Ideea ce stă la baza proiectării independente de tehnologie este că ar fi de dorit să se utilizeze o metodă de proiectare bazată pe un limbaj de descriere hardware (HDL) și convertirea acestei descrieri prin intermediul unei sinteze software într-o formă intermediară, care să poată fi direcționată spre diverse stiluri de implementare și circuite provenind de la producători distincți. Abordarea aceasta este analogă cu scrierea de software în limbaje de nivel înalt, codul sursă putând fi *portat* și compilat pe diverse platforme.

Avantajele acestei abordări sunt:

- creșterea productivității prin utilizarea unei descrieri la nivel înalt;
- posibilitatea alegerii unui spectru larg de producători și eliminarea dependenței de un producător *unic*, de obicei și furnizor al unui CAD specific.

Se pot menționa aici și următoarele dezavantajele:

- eficiență scăzută, prin adaptarea mai slabă la circuitul destinație al proiectului;
- pierderea controlului direct asupra arhitecturii proiectului, a principalelor aspecte temporale implicate.

Cele două dezavantaje menționate anterior pot reprezenta aspecte esențiale pentru reușita unui proiect. Probabil că locul de confruntare cel mai acerb între pro-urile și contra-urile

proiectării independente de tehnologie sunt circuitele FPGA, unde utilizând unelte CAD potrivite proiectul se poate implementa *integral* prin intermediul unui limbaj HDL.

Până în prezent, în inginerie în general, indiferent de nivelul sau domeniul la care se realizează proiectarea, circuitele de mare succes au fost întotdeauna cu un ordin de mărime mai performante decât media, mai ales datorită utilizării creatoare de către inginer a anumitor aspecte specifice tehnologiei implicate, adică *dependent de tehnologie*.

1.3.4. METODA MEAD-CONWAY

Lucrare de referință în domeniu, [Mead80] descrie o metodă de proiectare care acoperă toate etapele, de la descrierea funcțională la amplasamentul final, tehnica propusă fiind pe larg preluată de producătorii circuitelor integrate complexe. Metoda Mead-Conway (denumită astfel deoarece nu s-a răspândit sub un alt nume) este o metodă ierarhică care se bazează pe managementul complexității interconexiunilor unui proiect. Acest aspect este deosebit de important deoarece costul asigurării fluxului de date într-un circuit complex este măcar la fel de mare ca și costul prelucrării acestor date în restul circuitului. [Mead80] este una dintre primele lucrări care definind clar acest aspect și propune tehnici eficiente de proiectare, care includ:

- managementul conexiunilor ca obiectiv primar în nivelurile superioare de proiectare;
- utilizarea de reguli simple de proiectare pentru obținerea unui grad ridicat de abstractizare a detaliilor;
- utilizarea de module care au structuri "naturale" elegante de interconectare, ca de exemplu canale dedicate de date sau o arhitectură PLA pe două niveluri (tabelul 1.3).

Tabelul 1.3

Forme naturale de interconectare

Modul	Forma naturală
Funcții logice	Logică pe două niveluri (PLA, ROM)
AES	Logică pe două niveluri (PLA, ROM)
Registru de date	Canal de date
ALU	Canal de date
Multiplicator	Arie combinațională

Metoda Mead-Conway menține un izomorfism între descrierea structurală și cea fizică, evitând prin aceasta dezavantajele păstrării unor ierarhii separate pentru cele două descrieri. Acest lucru se poate obține utilizând un compilator unic pentru a genera ambele ierarhii.

Metoda descrisă poate fi aplicată și în cazul FPGA, care prezintă intrinsec o structură omogenă. Din experiența proprie pot afirma că ea este mai potrivită pentru arhitecturile FPGA cu granularitate fină, care oferă resurse logice la nivel de poartă logică, decât pentru FPGA cu granularitate brută, la care resursele sunt reprezentate la nivel de bloc logic complex.

1.3.5. PROIECTAREA SISTEMELOR SINCRONE

În cazul sistemelor numerice în general și al FPGA în special, decizia privitoare la tipul de elemente de memorare utilizat în blocurile sau celulele logice influențează fundamental performanțele și arhitectura ansamblului. În cazul unui sistem *sincron*, toate celulele sunt co-

mandate de un semnal de tact global. Transferul informației se realizează prin canale de date ordonate, fiecare celulă dispunând de registre pentru stocarea temporară a informației (figura 1.9). La o asemenea abordare principala problemă este întârzierea semnalului de tact, care poate determina pierderea informației din anumite celule. De aceea, unele FPGA moderne au un arbore de distribuție a semnalului de tact cu întârziere redusă (sub 0,5 ns) în tot cipul [Actel92], [Altera93], [Xilinx94]. Utilizarea unei metode sincrone *nu* implică eliminarea hazardului combinațional, care trebuie studiat și eliminat pentru fiecare proiect și circuit în parte.

În cadrul unui sistem sincron, semnalul de tact se poate prezenta în două moduri:

- semnal de tact unic, (*o singură fază*): pentru această abordare moștenită din epoca SSI/MSI, semnalele *Clk 1* și *Clk 2* din figura 1.9 sunt identice; ea se poate utiliza la toate FPGA actuale;
- o schemă de tact cu *două faze*: această metodă, populară în epoca VLSI este descrisă în [Mead80]. Fronturile celor două semnale de tact nu se suprapun, aceasta împiedicând datele să parcurgă mai mult de o celulă într-o perioadă. O variantă a acestei metode este descrisă în [Demian94], (la care sunt coautor), în care este folosit un semnal de tact unic, dar cele două faze corespund frontului crescător și descrescător al semnalului de tact. Se obține astfel simultan posibilitatea achiziției de eșantioane la dublul frecvenței semnalului de tact și o schemă electrică eficientă. Această tehnică poate fi utilizată cu succes și în cazul FPGA apărute în ultimul timp care oferă posibilitatea acționării bistabilelor pe ambele fronturi. Tehnica UltraDMA, devenită operațională din anul 1997 la harddisk-urile IDE performante folosește aceeași schemă de tact, obținându-se dublarea ratei de transfer.

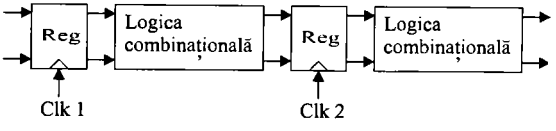


Figura 1.9. Model parțial de sistem sincron cu evidențierea modului de transfer între celule.

În cazul sistemelor *sincronizate intern* (self timed), descrise în [Dillinger88], [Mead80], fiecare celulă generează *explicit* semnalele de sincronizare de tip handshake: "go" și "done". Aceste semnale însoțesc datele pe întreg canalul de comunicație. Marele avantaj al acestei abordări constă în faptul că proiectantul este eliberat de sarcina studiului problemelor de temporizare din timpul proiectării. Dezavantajul acestei metode constă în resursele logice consumate suplimentar, corespunzător logicii de sincronizare asociate fiecărei celule.

1.3.6. TEHNICA PIPELINE

Această tehnică este o extensie a metodologiei de proiectare a sistemelor sincrone oferind în plus avantajul unei viteze sporite. Asemănător unei benzi de montaj din industria de automobile, transferul informației în și din celulele logice se realizează sincron cu un semnal unic de tact, fiecare celulă dispunând suplimentar de o memorie tampon pentru stocarea datelor intermediare. Fiecare operație corespunzătoare unei celule este simplă, putându-se executa

rapid, astfel încât întregul ciclu de prelucrări este mai scurt decât la abordarea sincronă clasică. Asupra acestei tehnici se va reveni la în capitolul 4, la §4.3.2.

1.3.7. CIRCUITE ASINCRONE

În acest caz proiectantul este răspunzător de toate corelațiile temporale din sistem. Cazul tipic este cel în care resursele logice sunt limitate la porți (indiferent de numărul lor), toate modulele logice fiind în final alcătuite din aceste porți. Această metodă este mai rar utilizată datorită dificultăților de proiectare și de asigurare a sincronizării în sistem.

1.4. PROPUNERE DE CLASIFICARE A STRUCTURILOR LOGICE PROGRAMABILE

În literatura de specialitate anglo-americană nu există o terminologie unanim acceptată pentru SLP și cu atât mai puțin o clasificare unitară, ea diferind de la o firmă la alta și de la un autor la altul. La baza acestei situații stau atât rațiuni comerciale, cât și noutatea acestui domeniu, motiv pentru care nu a apărut un consens de opinii între specialiști. Clasificarea este importantă pentru proiectant pentru a putea lua decizii atât la nivel strategic (ce fel de circuit este mai potrivit pentru scopul propus), cât și tactic (care sunt caracteristicile majore ale structurii necesare). Clasificarea propusă în continuare aduce câteva elemente noi față cele prezentate în [Auer95], [Brown92] și [Rose93] în primul rând prin numărul de criterii evidențiate (6 față de maximum 3), dar și prin informațiile furnizate de acestea. Clasificarea originală propusă în continuare corespunde următoarelor criterii:

1. După *gradul de complexitate*, respectiv numărul de porți ȘI-NU echivalente:
   a) SLP de complexitate redusă, sub 500 de porți echivalente (PAL, PLA, GAL);
   b) SLP de complexitate medie, sub 5.000 de porți echivalente (PLD, CPLD);
   c) SLP de complexitate mare, peste 5.000 de porți echivalente (FPGA).

Zonele de separare dintre a), b) și c) se întrepătrund, existând de exemplu FPGA-uri echivalente cu 1.200 de porți ȘI-NU.

Din punctul de vedere al autorului, acesta este criteriul esențial de alegere al SLP. Modalitatea de exprimare ca număr de porți echivalente este în general acceptată de producător și de beneficiar, cu mențiunea că în funcție de tipul aplicației, algoritmii de rutare și plasare utilizați și experiența proiectantului, gradul de utilizare pentru resursele logice ale SLP se situează între 10 și 95%. De aici rezultă clar importanța unei *proiectări optimale*, care să folosească cât mai bine resursele disponibile sau să permită utilizarea SLP de complexitatea cea mai redusă care să satisfacă scopul propus.

2. După *modul de funcționare*:
   a) combinaționale (unele PLA-uri, unele PAL-uri, unele FPGA-uri);
   b) secvențiale.

3. După *locul programării* lor:
   a) la fabricant (HAL, circuitele GaAs Gazelle Microcircuits)
   b) la utilizator, acestea la rândul lor se pot programa:

b1) într-un programator (PAL, EPLD, Actel FPGA);
b2) chiar în montajul în care operează (E^2PLD, Xilinx FPGA, ispLSI).

4. După *tehnologia utilizată*:
   a) bipolare (PLA, PAL);
   b) unipolare (CHMOS);
   c) GaAs (firma Gazelle Microcircuits).

5. După *posibilitatea reprogramării*:
   a) nereprogramabile - toate circuitele bipolare şi unele FPGA-uri;
   b) reprogramabile după ştergere prin expunere la lumină ultravioletă (EPLD);
   c) reprogramabile electric (E^2PLD).

6. După *posibilitatea de testare*:
   a) fără facilităţi de testare;
   b) cu testare pe durata programării;
   c) cu testare funcţională, pe frontieră.

În continuare este introdusă o metodă de ordonare a SLP într-un sistem propriu de coordonate, care poate fi utilizat la estimarea celei mai potrivite structuri pentru o aplicaţie dată.

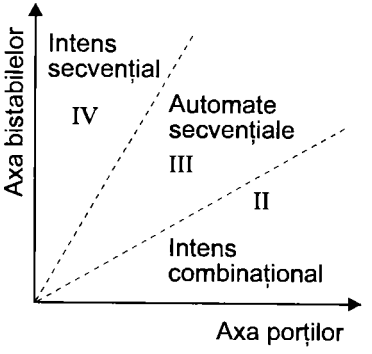


Figura 1.10. Zonele de aplicaţie.

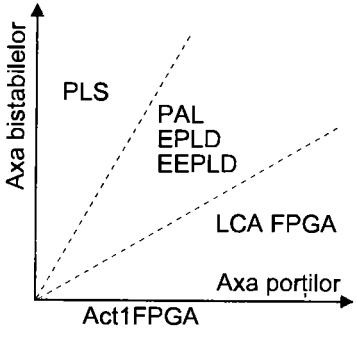


Figura 1.11. Ordonarea SLP.

În figura 1.10, *Oy* este axa bistabilelor, iar *Ox* este axa porţilor. Trisecţionând unghiul drept al primului cadran, se pot delimita cinci zone:

1. Axa *Ox* - Zona aplicaţiilor pur combinaţionale;
2. Zona II - Zona aplicaţiilor intens combinaţionale;
3. Zona III - Zona automatelor elementare cu stări finite;
4. Zona IV - Zona aplicaţiilor intens secvenţiale;
5. Axa *Oy* - Zona aplicaţiilor pur secvenţiale.

Toate structurile logice programabile pot fi ordonate în coordonatele Porţi/Bistabile. Trebuie menţionat că circuitele complexe, în funcţie de configuraţia în care sunt folosite, pot ocupa cel mult una din două zone adiacente (figura 1.11).

1.5. CONCLUZII

În acest capitol au fost prezentate pe scurt principalele elemente legate de implementarea sistemelor numerice, accentuându-se de la bun început avantajele utilizării logicii programabile în comparație cu utilizarea circuitelor integrate numerice clasice.

În paragraful 1.1 au fost identificate pe baza unui exemplu simplu abordările clasice de implementare a unui sistem numeric: hardware și software. Deși nu apare explicit în literatura de specialitate, este posibilă și o a treia variantă, respectiv *logica programabilă*, care îmbină avantajele majore ale abordărilor prezentate anterior. Importanța acestei noi posibilități este cu atât mai mare, cu cât permite implementări directe utilizând circuite și echipamente ieftine, aspect deosebit de important pentru țara noastră. Comparația efectuată de autor în subparagraful 1.1.4 prezintă sintetic aceste avantaje, încheindu-se cu estimarea că în mai puțin de zece ani, logica programabilă va domina cel puțin segmentul de piață al aplicațiilor la cerere.

Studiul amănunțit al bibliografiei extinse în acest domeniu a permis identificarea în paragraful 1.2 a principalelor etape de dezvoltare ale circuitelor integrate numerice. Aceste etape, desemnate convențional la o durată de zece ani, au fost completate cu numirea acestui deceniu ca fiind *etapa SLP*, afirmație motivată de succesele înregistrate pe plan mondial de aceste circuite. Prezentarea sistematică a acestor etape vine să sprijine concluziile enunțate anterior și oferă o perspectivă a acestor evoluții și tendințe.

Redarea unitară, clar structurată, a tuturor metodelor de proiectare a sistemelor numerice, a permis reliefarea principalelor caracteristici ale fiecăreia și implicit analiza comparată cu evidențierea avantajelor și actualității în fiecare caz în parte.

Paragraful 1.3 debutează cu o succintă descriere a pachetului de programe LOG/iC, a cărui aprofundare a fost posibilă pe durata specializării efectuate de autor în Germania în 1995. Acest pachet este unul din cele mai perfecționate instrumente CAD de proiectare cu PLD, permițând dezvoltarea eficientă de aplicații. Prezentarea este continuată firesc cu metodele de descriere pentru un sistem numeric. Această sistematizare a condus la identificarea neajunsurilor diagramei Gajski-Kuhn și propunerea extinderii ei spațiale. Este de asemenea importantă concluzia de la sfârșitul subparagrafului 1.3.3, care indică superioritatea implementărilor dependente de tehnologie față de cele independente tehnologic.

Studiul metodei Mead-Conway conduce la observația că această metodă este aplicabilă la arhitecturile FPGA cu granularitate fină, domeniu al SLP extrem de nou și dinamic, pentru care nu există în prezent o rezolvare completă, în consens, pe plan mondial.

Propunerea de schemă de tact cu două faze, derivată din metoda Mead-Conway, a fost aplicată la realizarea unui analizor logic ieftin și performant, precum și a unui număr de lucrări originale pe această temă.

Clasificarea SLP care încheie primul capitol prezintă o importanță specială. Criteriile prezentate se transformă firesc pentru utilizator în jaloane fundamentale de selecție optimală a unui SLP potrivit aplicației dorite. Ea permite definirea sistemului de coordonate porți-bistabile din figura 1.10 și ordonarea SLP-urilor în zonele de aplicație din figura 1.11.

1.6. REFERINȚE BIBLIOGRAFICE

Detalii referitoare la implementarea sistemelor numerice se găsesc în mai multe surse bibliografice, de exemplu [Hill93], [Mureşan96], [Wakerly90]. În [Tredennick95] se oferă o excelentă perspectivă asupra viitorului imediat al SLP, competiția SLP-urilor cu microprocesoarele. Lucrări de referință privind proiectarea circuitelor integrate VLSI sunt: [Dillinger88], [Gajski88], [Hill93] şi [Mead80].

Dintre lucrările autorului, [Gontean91] abordează unitar în premieră subiectul SLP, detaliind SLP de complexitate redusă şi implementarea utilizarea acestor circuite. [Gontean92a] conține detalii privind implementarea automatelor cu stări finite cu SLP, alegerea SLP potrivit pentru o aplicație dată. [Gontean97] oferă informații suplimentare referitoare la noțiunile fundamentale privind SLP, istoria evoluției acestor circuite şi structura prezentă a pieței în acest domeniu.

CAPITOLUL 2

STRUCTURI LOGICE PROGRAMABILE MODERNE

2.0. INTRODUCERE

Apariția structurilor logice programabile la mijlocul anilor '70 şi mai ales introducerea FPGA în 1986 au determinat o evoluție spectaculoasă a tehnicilor şi mijloacelor de proiectare a sistemelor numerice. În acest context este prezentată în prima parte a acestui capitol o sinteză a principalelor structuri logice programabile disponibile în prezent pe piață, preluată din [Gontean97]. SLP de complexitate redusă au fost amintite pe scurt, deoarece arhitectura introdusă de aceste structuri se păstrează şi în circuitele moderne, fabricate de exemplu de firma Altera [Altera93].

O atenție deosebită este acordată circuitelor FPGA, a căror prezentare succintă continuă cu un model de structură şi un studiu original al influenței granularității blocului logic asupra ariei ocupate de circuit. Studiul teoretic şi practic al metastabilității permite autorului prezentarea unui montaj original de determinare a caracteristicilor de metastabilitate pentru orice tip de circuit numeric şi câteva concluzii practice referitoare la acest subiect.

Capitolul se încheie cu un set de concluzii care includ şi câteva contribuții originale, referitoare la arhitectura blocului logic şi comportarea metastabilă a SLP.

Deşi nu acoperă decât o parte din criteriile posibile de alegere a SLP potrivit unei aplicații date, acest capitol demonstrează din mai multe puncte de vedere - între care unele originale - superioritatea arhitecturii FPGA oferite de firma Xilinx. Această concluzie este validată de ponderea acestor circuite (de exemplu de peste 50% din cota de piață în 1997) care s-a menținut relativ constantă în ultimul deceniu.

2.1. SLP DE COMPLEXITATE REDUSĂ

SLP de complexitate redusă conțin circuite de tip PROM, PLA şi PAL, cu o structură internă pe două niveluri de tip ŞI-SAU. Primul nivel este realizat cu porți ŞI, ale căror intrări se pot conecta la valorile adevărate sau complementare ale intrărilor circuitului. Ieşirea porților ŞI este conectabilă la intrarea nivelului următor, realizat cu porți SAU.

Tabelul 2.1

Niveluri programabile în structurile logice de complexitate redusă

ARIE	PROM	PAL	PLA
ŞI	FIXĂ	PROGRAMABILĂ	PROGRAMABILĂ
SAU	PROGRAMABILĂ	FIXĂ	PROGRAMABILĂ

Arhitectura ŞI-SAU este foarte potrivită pentru implementarea de ecuaţii logice sub forma de sumă de produse. Diferenţele majore dintre SLP-urile amintite constau în posibilitatea de a putea programa un nivel sau altul (tabelul 2.1).

2.1.1. TIPURI FUNDAMENTALE DE ARHITECTURI SLP

Variantele introduse în tabelul 2.1 se regăsesc ca tipuri fundamentale de SLP. Cu toate că memoria PROM nu este o SLP, ea poate fi astfel considerată (şi eventual implementată) şi de aceea am considerat util a trata unitar *toate* aceste trei categorii de arhitectură.

A. Structura internă a unui PROM

Arhitectura unui PROM, *privit ca o SLP*, este prezentată în figura 2.1. PROM-ul are patru intrări, 10 porţi ŞI cu 8 intrări, respectiv 4 porţi SAU cu 10 intrări. Intrările în PROM sunt decodificate *complet* prin intermediul bufferelor şi a celor 10 porţi ŞI. Fiecare combinaţie de la intrare este preluată printr-o poartă ŞI cu câte $2n$ intrări (n este numărul intrărilor în PROM).

Un PROM poate fi folosit pentru implementarea funcţiilor logice multiple, câte o funcţie logică independentă la fiecare ieşire, singura limitare fiind numărul de intrări disponibile. Întrucât toate intrările sunt decodificate complet, aplicaţii de genul generator de caractere, convertoare de cod, generatoare de funcţii sunt foarte potrivite pentru implementarea cu un PROM.

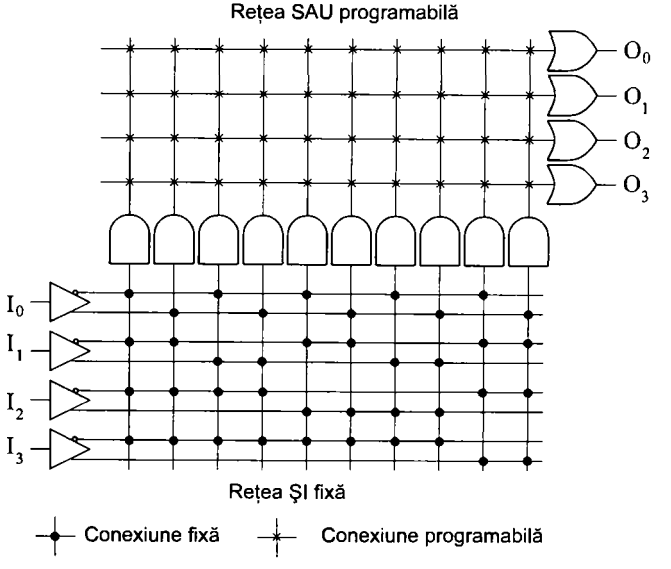


Figura 2.1. Structura internă a unui PROM.

Un PROM are un număr fix de intrări şi ieşiri. De exemplu, un PROM cu capacitatea 4k x 8 biţi are 12 intrări şi 8 ieşiri. Numărul fix de intrări/ieşiri este limitarea majoră a PROM-urilor la implementarea funcţiilor logice. O funcţie necesitând o combinaţie intrări/ieşiri diferită - chiar dacă numărul total intrări + ieşiri este mai mic decât cel oferit de structura PROM-ului - nu va putea fi implementată.

O funcţie cu 13 variabile de intrare şi 6 variabile de ieşire nu va putea folosi un PROM de tipul amintit (4k x 8), chiar dacă numărul de intrări + ieşiri (13 + 6) este mai mic decât cel oferit de PROM (12 + 8).

Sunt disponibile PROM-uri cu bistabile pe ieşiri, la care valoarea la ieşire nu este prezentă decât după activarea unui semnal de sincronizare. Cu aceasta se poate elimina efectul nedorit al hazardului combinaţional. Pentru aplicaţii rapide sunt potrivite circuitele EPROM, cum ar fi CY7C245 oferit de Cypress, de 2k x 8 biţi, ieşire cu bistabile şi cu un timp de acces de 25 ns.

B. Structura internă a unui PLA (figura 2.2)

Caracteristice unui PLA sunt:

- numărul de intrări (n);
- numărul de ieşiri (m);
- numărul de termeni produs (p).

Generic, se poate descrie un astfel de circuit ca un PLA $n \times m$ cu p termeni produs. În general p este mult mai mic decât numărul de mintermeni pentru o funcţie de n variabile; de aceea un asemenea PLA conţine mult mai puţină "logică" decât un ROM cu acelaşi număr de intrări şi ieşiri ($2^n \times m$ ROM) - care poate implementa orice funcţie logică cu n intrări şi m ieşiri. Utilizarea unui PLA se limitează la situaţiile în care funcţiile logice se pot exprima cu p sau mai puţini termeni produs.

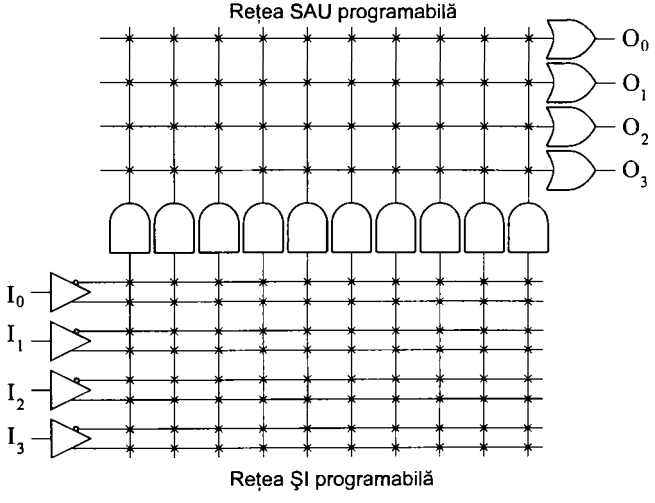


Figura 2.2. Structura internă a unui PLA.

Un PLA $n \times m$ cu p termeni produs conține p porți ȘI, fiecare având $2n$ intrări și m porți SAU, fiecare cu p intrări. Fiecare linie de intrare conține un buffer ce oferă simultan la ieșire complementul și valoarea adevărată a intrării respective. După cum s-a menționat în capitolul 1, în ciuda flexibilității deosebite oferite (atât matricea de porți ȘI, cât și matricea de porți SAU programabile), circuitele PLA au o mică răspândire, deoarece:

- sunt lente - semnalul parcurge *două* niveluri de fuzibile programabile;
- sunt scumpe - aria de siliciu consumată este mare;
- suportul software nu este dezvoltat;
- sunt greu de înțeles de utilizator.

C. Structura internă a unui PAL

În cazul PAL, aria de porți ȘI este programabilă, iar aria de porți SAU este fixă (figura 2.3). Spre deosebire de PROM-uri, la PAL-uri intrările nu sunt decodificate complet. Pentru PAL-ul prezentat în figura 2.3, la 6 intrări sunt prezente doar 16 porți ȘI (*și nu 2^6 porți ȘI*). Faptul că decodificarea nu este completă permite mărirea numărului de intrări, ceea ce nu implică o creștere substanțială a numărului de fuzibile. Mărind de exemplu numărul de intrări de la 6 la 10, aria de siguranțe crește de la 12 x 16 la 20 x 16.

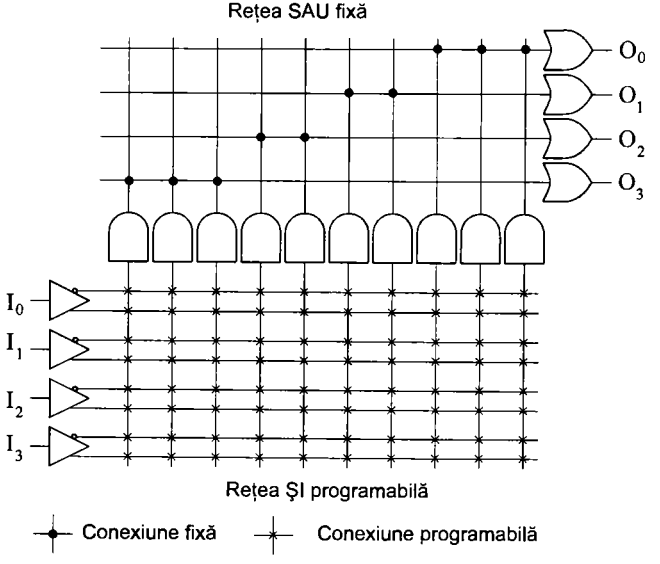


Figura 2.3. Structura internă a unui PAL.

Deoarece pentru un PAL o ieșire furnizează suma unui număr limitat de termeni (proveniți de la porțile ȘI), numărul de porți ȘI necesare pentru o implementare nu trebuie să depășească acest număr.

Pe lângă numărul mare de intrări, circuitele PAL prezintă facilități deosebite care le fac potrivite pentru implementarea funcțiilor logice:

- pini de intrare/ieșire programabili;
- ieșiri combinaționale sau cu bistabile cu reacție spre aria programabilă;
- ieșiri cu polaritatea programabilă.

2.2. CIRCUITE DE COMPLEXITATE MARE (FPGA)

Structurile FPGA oferă un grad ridicat de integrare, accesibil anterior doar ariilor de porți programabile prin mască (MPGA, **M**ask **P**rogrammable **G**ate **A**rrays) împreună cu o durată redusă a ciclului proiectare-producție, specifică pentru PLD. De la introducerea lor de către firma Xilinx în 1986, structurile FPGA au reprezentat unul din segmentele din industria electronică cu cea mai rapidă dezvoltare, cu aplicații variind de la înlocuirea logicii tradiționale (echivalentul a câtorva mii de porți), până la proiecte vizând substituirea de microprocesoare speciale (echivalentul a câtorva zeci de mii de porți). Se estimează că această tendință va continua în următorii ani, cu un volum de vânzări ce va depăși un miliard de dolari la sfârșitul acestui deceniu.

Pe lângă succesul incontestabil de piață, FPGA-urile deschid noi domenii de cercetare în:

- proiectarea VLSI;
- tehnologii și limbaje de programare;
- CAD;
- supercomputere cu utilizare specială.

Începând din 1990, s-au organizat mai multe workshop-uri în acest domeniu,inițial în SUA și apoi în Europa și Asia. În facultăți se introduc cursuri de proiectare cu FPGA. Rolul FPGA în educație a fost recunoscut de National Science Foundation SUA, care a sponzorizat o serie de workshop-uri și conferințe pe această temă. Beneficiind de sprijinul firmei Texas Instruments (unde a fost școlarizat în domeniul FPGA), autorul a susținut trei asemenea workshop-uri: unul la Facultatea de Electronică și Telecomunicații din București în 1993 și două la Facultatea de Electronică și Telecomunicații din Timișoara în 1994.

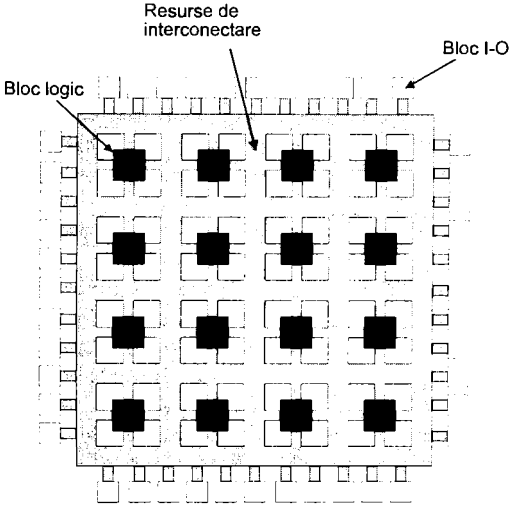


Figura 2.4. Arhitectura FPGA.

Arhitectura FPGA (figura 2.4) este similară cu cea a circuitelor MPGA, constând dintr-o matrice de blocuri logice care pot fi interconectate prin programare pentru a obține funcționarea dorită. Diferența majoră între FPGA și MPGA este aceea că circuitele MPGA sunt programate la fabricant prin mască, în timp ce FPGA-urile sunt programate la utilizator asemenea PLD tradiționale. Față de PLD-uri, circuitele FPGA oferă niveluri mult mai mari de integrare, arhitecturi și variante de rutare complexe. Rutarea PLD este foarte simplă, dar și extrem de ineficientă, datorită stabilirii unor legături de tip intrare-ieșire prin intermediul unui singur switch; în cazul FPGA rutarea se execută prin intermediul a mai multor switch-uri, ceea ce permite o mai bună utilizare a resurselor logice.

La PLD, logica este implementată mai ales cu scheme de tip ȘI-SAU, conținând porți ȘI cu multiple intrări. În contrast, circuitele FPGA utilizează porți cu un număr mai mic de intrări, structurate pe mai multe niveluri, o abordare mult mai compactă pentru majoritatea aplicațiilor.

Pentru FPGA, blocul logic poate fi simplu ca un tranzistor sau să prezinte complexitatea unui microprocesor, fiind capabil în cazul tipic de a implementa mai multe funcții combinaționale și secvențiale.

La FPGA actuale, blocurile logice sunt formate din:

- tranzistoare;
- porți simple cu două intrări, de exemplu ȘI-NU respectiv SAU-EXCLUSIV;
- multiplexoare;
- tabele de căutare LUT (**L**ook **U**p **T**ables);
- structuri ȘI-SAU cu multe intrări.

Rutarea în FPGA poate fi o simplă legare cu cel mai apropiat vecin sau comparabilă cu cea dintr-un microprocesor. O variantă tipică de rutare constă din segmente de legătură având lungimea variabilă, interconectabile prin switch-uri programabile electric. Alegerea numărului *optim* de segmente pentru legături este un procedeu dificil care nu este definitiv pus la punct. Alegând un număr nepotrivit de segmente de legătură, numai o mică parte din blocurile logice pot fi utilizate, ceea ce duce la irosirea inutilă a resurselor oferite de circuit.

Lungimea segmentelor de legătură joacă de asemenea un rol important în stabilirea performanțelor circuitului. Alegând de exemplu numai segmente lungi pentru interconectare, abordarea se dovedește costisitoare în ceea ce privește aria ocupată și timpul de propagare ridicat (frecvență scăzută de lucru). Pe de altă parte, dacă toate segmentele sunt scurte, interconexiunile între blocuri logice depărtate se realizează prin intermediul unui număr mare de switch-uri legate în cascadă, ceea ce duce la un timp de propagare nepermis de ridicat.

Implementarea switch-urilor programabile se realizează prin mai multe tehnologii, dintre care trei sunt majore:

- SRAM, în care switch-ul este un tranzistor controlat de un bit de memorie;
- Antifuzibil, care programat devine un contact electric de rezistență redusă;
- EPROM, unde se utilizează un tranzistor cu grilă flotantă care poate fi blocat prin injectarea de sarcină pe grilă.

În toate aceste tehnologii, switch-ul ocupă o arie mai mare și prezintă o rezistență în conducție mai ridicată decât orice alt element de interconectare utilizat la MPGA. Este necesară suplimentar o zonă pentru logica de programare. Drept rezultat, densitatea de integrare a circuitelor FPGA actuale este cu un ordin de mărime mai redusă decât a MPGA-urilor fabricate în aceeași tehnologie.

Efectele negative ale dimensiunii și rezistenței parazite ale switch-urilor pot fi reduse prin arhitecturi evoluate. Alegând dimensiunea și structura potrivită pentru un bloc logic,

proiectând varianta de rutare pentru obținerea unui grad ridicat de utilizare a resurselor se pot optimiza atât densitatea de integrare, cât și performanțele. Cea mai bună alegere din punct de vedere al arhitecturii interne depinde din păcate de tipul de proiect ce urmează a fi implementat, cu alte cuvinte nu există "cea mai potrivită arhitectură" care să corespundă *simultan* tuturor tehnologiilor și tuturor proiectelor.

Complexitatea actuală a FPGA a depășit momentul proiectării manuale. În prezent performanțele implementărilor cu FPGA depind esențial de software-ul de proiectare, de cel de plasare și de rutare utilizat.

2.2.1. TEHNOLOGII DE PROGRAMARE

A. SRAM

Tehnologia SRAM este utilizată în circuite fabricate de Xilinx, Plessey, Algotronix, Concurrent Logic și Toshiba. Valorile logice memorate în SRAM controlează switch-urile, respectiv elemente ca multiplexoare (figura 2.5).

În figura 2.5.a, atunci când în celula SRAM este memorat un "1", tranzistorul MOS conduce, comportându-se ca un comutator închis și poate fi utilizat pentru interconectarea a două segmente de legătură. În cazul multiplexorului, valorile stocate în SRAM comandă care intrare a multiplexorului se va conecta la ieșirea sa (figura 2.5.b).

Deoarece memoriile SRAM sunt volatile, FPGA-ul trebuie configurat la fiecare alimentare. Aceasta conduce la necesitatea unui mecanism de memorare a configurării, materializat într-un PROM, EPROM, EEPROM sau date pe (hard)disk.

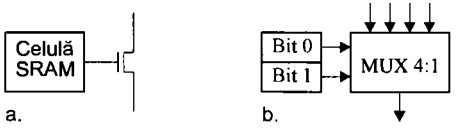


Figura 2.5. Comanda cu SRAM a elementelor logice.

Principalul dezavantaj al tehnologiei SRAM este suprafața mare ocupată. Sunt necesare cel puțin cinci tranzistoare pentru implementarea unei celule SRAM de un bit și cel puțin încă un tranzistor ca switch programabil. Avantajele acestei tehnologii sunt în schimb majore:

- reprogramare rapidă;
- reprogramare dinamică (în circuit);
- proces standard de fabricație.

B. Antifuzibil

Tehnologia antifuzibil este utilizată în circuite fabricate de Actel, Quicklogic și Crosspoint. Un antifuzibil (antisiguranță) este un element de circuit cu două terminale care prezintă în starea neprogramată o rezistență foarte mare între terminale. Atunci când o tensiune de programare (între 11...20 V, în funcție de tipul de antifuzibil) este aplicată între terminale, au loc o serie de procese care conduc în final la modificarea rezistenței la o valoare redusă. Acest tip de legătură este *permanent*.

În prezent se utilizează pentru fabricarea antifuzibilului un dielectric oxigen-azot-azot (ONO) sau siliciu amorf între două straturi conductoare, de obicei metalice. Această tehnologie implică resurse logice suplimentare pentru a putea suporta tensiunea relativ mare de programare şi curenţi de programare de 5 mA sau mai mult.

Avantajul oferit de această tehnologie este dimensiunea redusă a antifuzibilului, cu puţin mai mare decât conexiunea internă a două segmente metalice. Din păcate, acest avantaj este redus de dimensiunea ridicată a tranzistoarelor de programare, de prezenţa tranzistoarelor de izolare care trebuie să protejeze tranzistoarele ce operează la tensiuni reduse de tensiunile relativ ridicate din timpul programării. Un alt avantaj al acestei tehnologii este rezistenţa redusă a antifuzibilului: 300...500 Ω la ONO, respectiv 50...100 Ω în celălalt caz. În plus, capacitatea parazită a unei astfel de legături este semnificativ mai redusă faţă de cea a celorlalte tehnologii.

C. EPROM

Această tehnologie este folosită în circuite fabricate de Altera şi Plus Logic (abordare EPROM), respectiv de AMD şi Lattice (EEPROM).

În acest caz switch-ul este un tranzistor care poate fi blocat permanent (figura 2.6). Aceasta se realizează prin injectarea de sarcină pe grila flotantă (grila 2 din figura 2.6), prin intermediul unei tensiuni ridicate aplicate între grila 1 şi drena tranzistorului. Sarcina suplimentară creşte tensiunea de prag a tranzistorului, blocându-l. Sarcina poate fi îndepărtată prin expunere la lumină ultravioletă, prin aceasta reducându-se tensiunea de prag şi tranzistorul funcţionează normal. Un aspect interesant este că în locul utilizării tranzistorului direct ca un switch programabil, se preferă folosirea sa pentru legarea la masă a unei linii de bit, atunci când linia de cuvânt este SUS. Pe lângă faptul că o asemenea abordare permite o legătură logică simplă între liniile de cuvânt şi cele de bit, ea permite implementarea de logică ŞI-cablat - prin aceasta se realizează atât *funcţii logice, cât şi de rutare*.

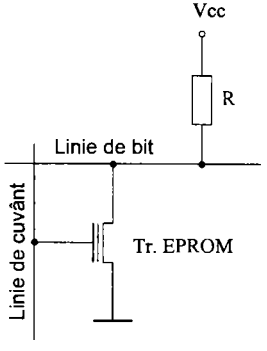


Figura 2.6. Un tranzistor cu grilă flotantă folosit ca switch.

Analog cu tehnologia SRAM, şi aceste circuite sunt reprogramabile. Un avantaj faţă de abordarea SRAM este că nu mai este necesar un circuit extern de memorare a configurării. Printre dezavantaje se pot enumera:

- rezistenţa ridicată în conducţie a tranzistorului;
- consum ridicat de putere, chiar în regim static, datorită rezistenţei de pull-up (figura 2.6).

Tehnologia EEPROM este foarte asemănătoare, cu deosebirea că îndepărtarea sarcinii se poate realiza electric, în circuit. Prin aceasta se adaugă avantajul programării structurii la distanță sau a utilizării aceloraşi resurse în două scopuri diferite. Celula EEPROM este de aproape două ori mai mare decât cea EPROM.

D. Rezumatul principalelor tehnologii

Tabelul 2.3

Comparație între principalele tehnologii

Tehnologie	Volatil	Reprogramabil	Arie	R_{ON} [Ω]	C_P [fF]	Etape supl.
SRAM	DA	DA, în circuit	Mare	0.5 - 2k	10-20	0
Antifuzibil ONO	NU	NU	Antifuzibil redus Tr. Programare mare	300-500	5	3
Antifuzibil (amorf)	NU	NU	Antifuzibil redus Tr. Programare mare	50-100	1.1-1.3	3
EPROM	NU	DA, în afara circuitului	Redusă	2-4 k	10-20	3
EEPROM	NU	DA, în circuit	2 x EPROM	2-4 k	10-20	> 5

Tabelul 2.3 oferă o comparație între alternativele descrise anterior. În toate cazurile se presupune o tehnologie CMOS de 1,2 μm. În tabel se dau date diferite corespunzător celor două variante de antifuzibil. Coloana a 5-a şi a 6-a a tabelului se referă la rezistența în conducție a switch-ului, respectiv la capacitatea parazită în stare de blocare (fără a ține cont de alte capacități parazite suplimentare datorate legăturii sau tranzistoarelor de programare). Pentru comparație, capacitatea unui segment de 10 μm cu lățimea minim posibilă în cadrul procesului de 1,2 μm este de 0,6 fF. Ultima coloană prezintă numărul de etape suplimentare necesare în procesul de fabricație, comparativ cu tehnologia CMOS standard.

2.2.2. ARHITECTURA BLOCULUI LOGIC

Blocurile logice din FPGA actuale se deosebesc profund în dimensiune şi capabilități de implementare a funcțiilor logice. Blocul logic din FPGA-ul produs de firma Crosspoint este format din două tranzistoare şi poate implementa o inversare, pe când blocul de tip LUT al firmei Xilinx din seria 3000 poate implementa orice funcție de cinci variabile, dar este semnificativ mai mare. Pentru a putea compara blocurile logice se introduce noțiunea de *granularitate*, ca măsură a complexității blocului logic.
Granularitatea poate fi definită în mai multe feluri:

- după numărul de funcții booleene ce se pot implementa;
- după numărul de porți ŞI-NU echivalente;
- după numărul total de tranzistoare;
- după aria totală sau normalizată ocupată;
- după numărul de intrări şi/sau de ieşiri.

Problemele sunt şi mai complexe deoarece la unele FPGA, cum sunt cele produse de AMD şi ALTERA, partea de logică şi cea de rutare sunt parțial întrepătrunse, fiind dificil de separat contribuția fiecăreia în cadrul structurii. Pentru scopul acestei lucrări, se vor descrie doar două categorii de granularitate: fină şi grosieră.

Criteriul de comparație va fi cel menționat în [Gontean97] și anume modalitatea de implementare a unei funcții de trei variabile (figura 2.7.a), $f = a \cdot b + \bar{c}$, la care realizarea cu două porți ȘI-NU este prezentată în figura 2.7.b.

Figura 2.7. O funcție de trei variabile (a) și implementarea cu porți ȘI-NU (b).

A. FPGA cu granularitate fină

Principalul avantaj pentru acest tip de arhitectură este gradul ridicat de utilizare al blocurilor logice care se poate obține. În plus, tehnicile de sinteză sunt foarte apropiate de cele pentru MPGA, respectiv celule standard.

Pe de altă parte, dezavantajul major al acestei arhitecturi este numărul mare de segmente de legătură și de switch-uri necesare, ceea ce conduce la o arie mare de siliciu consumată și la un timp de propagare ridicat.

1) FPGA Crosspoint

Pentru această familie de FPGA, blocul logic constă dintr-o pereche de tranzistoare (figura 2.8). Implementarea funcției f este prezentată în figura 2.9, unde datorită modului specific de conectare pe rânduri a tranzistoarelor, *Tx* și *Ty* din zona haşurată sunt blocate pentru a izola porțile ȘI-NU ce implementează funcția. În afară de perechea de tranzistoare complementare, acest tip de FPGA mai conține un alt tip de bloc logic, bazat pe o memorie RAM, care în afară de utilizarea primară ca memorie cu acces aleator, mai poate fi folosit pentru a implementa funcții logice.

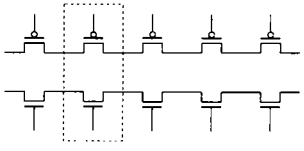


Figura 2.8. Perechi de tranzistoare complementare în FPGA.

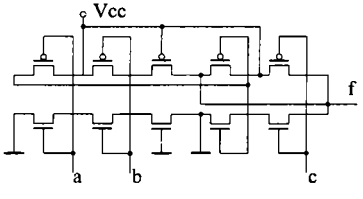


Figura 2.9. Implementarea funcției $f = a \cdot b + \bar{c}$ într-un FPGA Crosspoint.

2) FPGA Plessey

În acest caz blocul logic este reprezentat de o poartă ŞI-NU cu două intrări (figura 2.10). Implementarea dorită pentru funcţia f este realizată conform figurii 2.7.b. Latch-ul D este configurat transparent, cu aceasta blocul logic din figura 2.10 devine combinaţional.

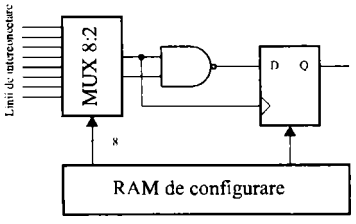


Figura 2.10. Blocul logic într-un FPGA Plessey.

B. FPGA cu granularitate grosieră

O caracteristică importantă a blocului logic este funcţionalitatea, definită în [Brown92] ca numărul de funcţii logice distincte pe care un bloc logic îl poate implementa. În cazul unei porţi ŞI-NU, funcţionalitatea este 5 ($f = \overline{a \cdot b}$, $f = \bar{a}$, $f = \bar{b}$, $f = 0$, $f = 1$). Un LUT cu trei intrări are funcţionalitatea mult mai mare, respectiv $2^{2^3} = 256$.

Există posibilităţi variate de alegere a arhitecturii blocului logic, aşa cum s-a văzut la §2.2.2. Fiecare dintre aceste variante oferă funcţionalităţi şi performanţe diferite, după cum funcţionalitatea blocului logic va influenţa resursele necesare rutării.

1) FPGA Actel

Blocul logic din această familie de FPGA se bazează pe proprietatea unui multiplexor de a implementa diverse funcţii logice prin conectarea intrărilor sale la variabilele de intrare sau la "0", respectiv "1" logic (mai multe exemple despre implementări de funcţii cu multiplexoare se găsesc în [Mureşan96].

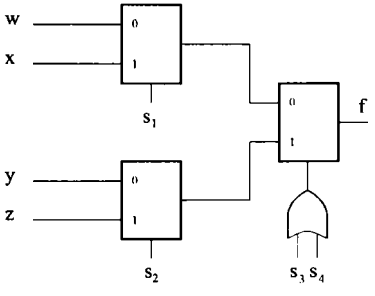


Figura 2.11. Blocul logic din familia Act-1.

Blocul logic din familia Act-1 (figura 2.11) implementează funcţia:

$$f = \left(\overline{s_3 + s_4}\right) \cdot \left(\overline{s_1} w + s_1 x\right) + \left(s_3 + s_4\right) \cdot \left(\overline{s_2} y + s_2 z\right).$$

Pentru cele 8 intrări sunt posibile 702 de combinații diferite. Funcția $f = a \cdot b + \overline{c}$ se implementează setând: $w = 1$, $x = 1$, $y = 0$, $z = a$, $s_1 = 0$, $s_2 = b$, $s_3 = c$, $s_4 = 0$. Blocul logic din familia Act-2 este asemănător (figura 2.12), permițând implementarea a 766 funcții diferite.

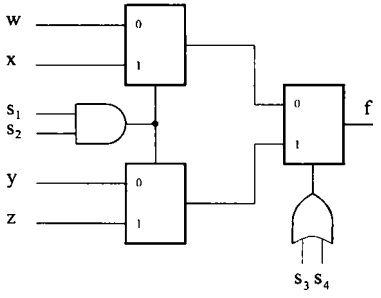


Figura 2.12. Blocul logic din familia Act-2.

2) FPGA Quicklogic

În figura 2.13 este prezentat blocul logic din familia FPGA a firmei Quicklogic. Se utilizează un multiplexor 4:1, cu fiecare intrare provenind de la o poartă ŞI. A doua intrare la porțile ŞI este inversată, prin aceasta nu mai este necesar câte un bloc logic suplimentar.

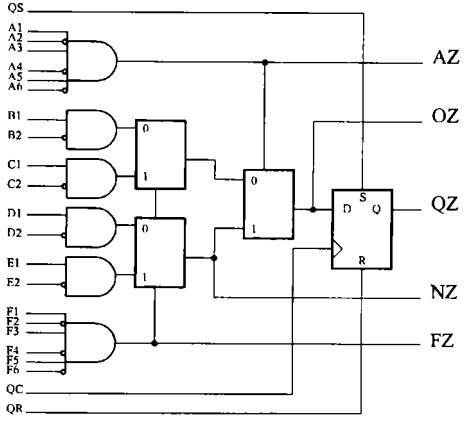


Figura 2.13. Blocul logic din familia FPGA Quicklogic.

Arhitecturile de FPGA bazate pe multiplexoare prezintă avantajul unei flexibilități deosebite, în contrast cu numărul relativ redus de tranzistoare implicat. Totuşi, numărul mare de intrări (8 în cazul Actel şi 14 în cazul Quicklogic) implică cerințe severe pentru resursele de

rutare şi switch-uri. De aceea aceste blocuri logice sunt mai potrivite pentru utilizarea de switch-uri de dimensiuni reduse, de exemplu antifuzibile.

3) FPGA Xilinx

Pentru această familie se utilizează o memorie SRAM care funcţionează ca LUT (figura 2.14). Tabelul de adevăr pentru o funcţie având k variabile este memorat într-un SRAM de capacitate $2^k \times 1$ biţi. Liniile de adresă în SRAM reprezintă variabilele de intrare, iar ieşirea SRAM - funcţia de implementat. Pentru tabelul de adevăr al funcţiei f, memoria SRAM va conţine "1" la adresa 0, "0" la adresa 1, etc. Principalul avantaj al LUT este funcţionalitatea, un LUT cu k intrări poate implementa *toate* funcţiile de k variabile (sunt 2^{2^k} asemenea funcţii). Dezavantajul - sunt inacceptabil de mari pentru mai mult de cinci intrări, deoarece numărul de celule de memorie pentru k intrări este 2^k. De aceea este relativ frecventă situaţia în care resursele oferite de LUT sunt slab utilizate.

a	b	c	d
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

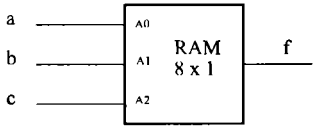


Figura 2.14. Un SRAM configurat ca LUT.

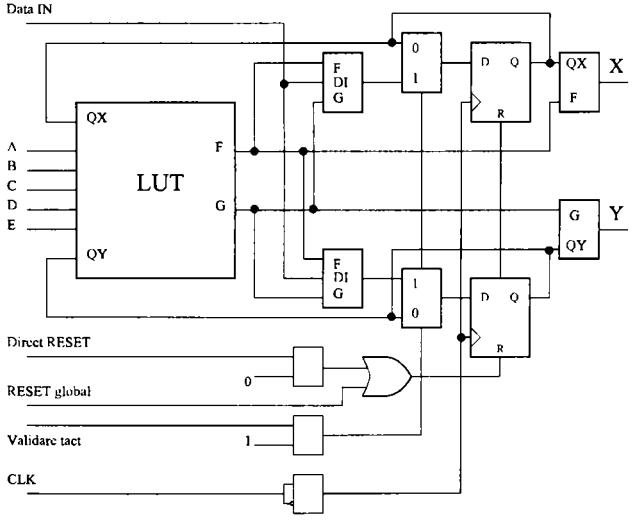


Figura 2.15. Blocul logic în familia Xilinx 3000.

Blocurile logice din familia Xilinx 3000 (care se bucură de o mare popularitate) conţin un LUT cu cinci intrări şi o ieşire, ce poate fi configurat în două LUT cu patru intrări, cu constrângerea că în total nu se vor utiliza mai mult de cinci intrări distincte. Prin aceasta se asigu-

ră o mai mare flexibilitate şi o bună utilizare a resurselor logice, deoarece în majoritatea situaţiilor funcţiile de implementat nu prezintă mai mult de cinci variabile. După cum se poate observa în figura 2.15, blocul logic mai conţine două bistabile şi multiplexoare pentru a putea realiza configuraţii diverse.

Familia Xilinx 4000 prezintă o nouă arhitectură pentru blocul logic (figura 2.16). În acest caz sunt utilizate două LUT cu patru intrări, ale căror ieşiri alimentează un LUT cu trei intrări. Structura eterogenă astfel creată asigură un echilibru mai bun între performanţe şi utilizarea resurselor logice. O altă abordare interesantă este faptul că legătura dintre LUT-urile cu patru intrări şi cel cu trei intrări este *neprogramabilă*, deci rapidă. Dezavantajul constă în faptul că la unele implementări acest LUT rămâne nefolosit.

Familia Xilinx 4000 oferă suplimentar:

- posibilitatea folosirii directe a LUT ca SRAM;
- circuite suplimentare de transport rapid (fast carry), utile pentru implementarea de sumatoare rapide.

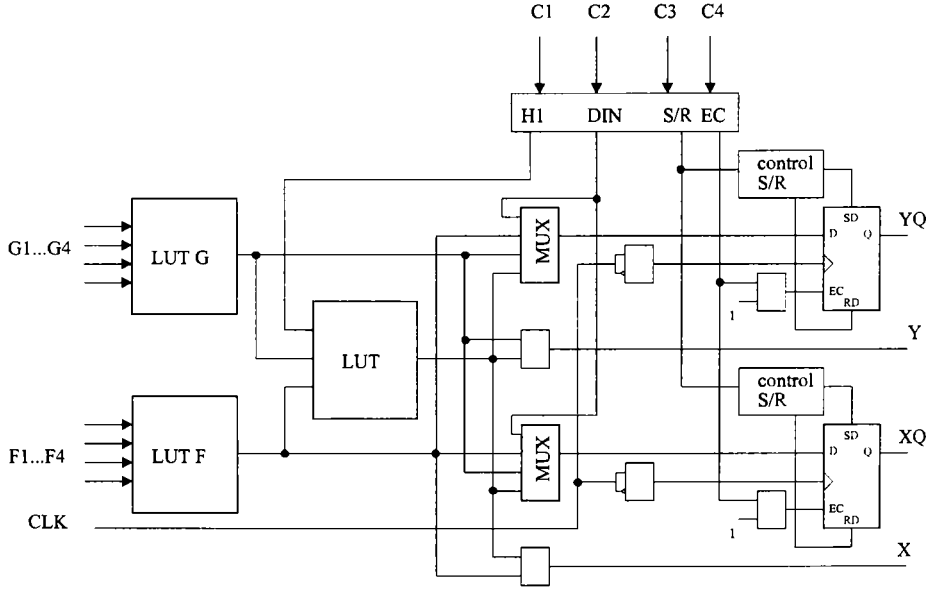


Figura 2.16. Blocul logic în familia Xilinx 4000.

4) FPGA Altera

Original bazate pe structura PAL-clasică, circuitele Altera au evoluat spre structuri ŞI-SAU având un număr mare de intrări (de la 20 la peste 100) în porţile ŞI, respectiv cu 3...8 intrări în porţile SAU. Arhitectura blocului logic din familia MAX 5000 este prezentată în figura 2.17.a.

Utilizând ca switch tranzistorul cu grilă flotantă din figura 2.6, orice segment vertical poate fi conectat la o intrare a unei porţi ŞI. Trei termeni produs sunt aduşi la intrările porţii SAU. Ieşirea poate fi inversată prin intermediul porţii SAU-EXCLUSIV, prin aceasta asigurându-se o mai mare flexibilitate. În plus, fiecare semnal de intrare este disponibil la

porţile ŞI atât în forma adevărată, cât şi cea complementată. Implementarea funcţiei f este prezentată în figura 2.17.b, în care punctele sunt legături de tip ŞI-cablat, descrise anterior.

Avantajul arhitecturii cu porţi ŞI cu un număr mare de intrări constă în faptul că funcţiile logice pot fi implementate pe un număr mic de niveluri de blocuri logice, cu aceasta reducându-se necesarul de interconexiuni programabile. Pe de altă parte, este dificil a folosi toate intrările sau măcar majoritatea lor, ceea ce conduce la irosirea resurselor logice. Pierderea nu este chiar aşa de severă precum apare la prima vedere datorită spaţiului redus utilizat de funcţiile ŞI-cablat, care suplimentar sunt folosite şi ca elemente de rutare. De aceea dezavantajul major al acestei arhitecturi rămâne puterea consumată, care este mare datorită rezistenţelor de pull-up. Pentru a compensa această pierdere, în familia MAX 7000 fiecare poartă ŞI poate fi programată la un consum de putere redus cu 60%, timpul de propagare corespunzător crescând în schimb cu circa 40%. Această facilitate poate fi utilizată cu succes în cazurile în care frecvenţa nu este critică.

Familia MAX 5000 oferă suplimentar un alt bloc logic, numit *expandor*, format dintr-o poartă ŞI-NU cu un număr ridicat de intrări, a cărei ieşire poate fi conectată la strucura ŞI-SAU a blocului logic din figura 2.17.a. Expandorul poate fi utilizat pentru mărirea numărului de termeni produs corespunzători unei structuri ŞI-SAU, ocupând o arie mică şi prezentând acelaşi timp de propagare ca blocul logic.

Blocul logic din familia MAX 7000 oferă suplimentar doi termeni produs şi facilitatea interesantă de "împrumut" a termenilor produs, valabilă pentru blocurile logice învecinate.

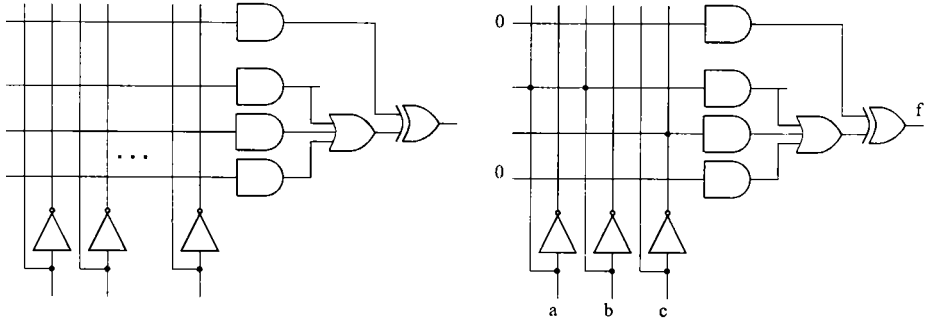


Figura 2.17. Blocul logic în familia Altera 5000 (a) şi implementarea funcţiei f (b).

2.3. EFECTUL GRANURALITĂŢII BLOCULUI LOGIC ASUPRA DENSITĂŢII DE INTEGRARE ŞI PERFORMANŢELOR SLP

O caracteristică importantă a blocului logic este funcţionalitatea sa, definită ca numărul de funcţii logice pe care un asemenea bloc îl poate implementa. De exemplu, o poartă ŞI-NU poate implementa 5 funcţii logice ($f = \overline{ab}$, $f = \overline{a}$, $f = \overline{b}$, $f = 0$, $f = 1$), pe când un bloc logic bazat pe un LUT cu 3 intrări poate implementa $2^{2^3} = 256$ funcţii logice. Aria totală ocupată în siliciu de un proiect implementat într-un FPGA este suma dintre aria totală a tuturor blocurilor logice şi aria totală ocupată de resursele de rutare. Deoarece tipic între 70 şi 90% din aria totală revine resurselor de rutare, este evident că arhitectura blocului logic are un impact

profund asupra ariei ocupate. La blocurile cu funcționalitate ridicată, numărul de pini este mare, dar numărul de conexiuni scade, distanța dintre blocuri scade (scăzând în același timp și timpul de propagare). Pe de altă parte un bloc complex ocupă o arie de siliciu importantă și adeseori este doar parțial folosit. Din această prezentare sumară rezultă că este posibil să existe un optim al dimensiunii blocului logic, pentru care aria ocupată să fie minimă. Acest subcapitol tratează modelarea și calculul acestui optim.

2.3.1. MODEL DE CALCUL AL ARIEI OCUPATE

Modelul propus (figura 2.18) se referă la FPGA cu structură simetrică, cu blocuri logice constituite din LUT cu k intrări (și o singură ieșire), fiind caracterizat de câțiva parametri:

- A_{bit} este aria necesară pentru a implementa un bit de memorie într-un LUT, având circa 400 μm^2 în tehnologia CMOS de 1,2 μm.
- RP este dimensiunea unui switch de rutare, respectiv a unei piste pentru figura 2.19;
- w este numărul de piste dintr-un canal;
- m este numărul de tabele LUT dintr-un bloc logic ($m = 1$) în cazul studiat;
- k este numărul de intrări în LUT.

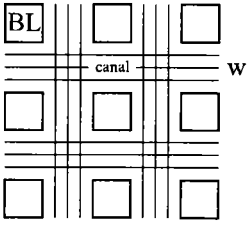


Figura 2.18. Model de conectare pentru FPGA.

Am preferat această alegere deoarece parametrul fundamental studiat - aria blocului logic - se poate foarte bine caracteriza printr-un singur parametru k. Concluziile deduse pot fi extinse și pentru celelalte arhitecturi ale blocului logic prezentate anterior.

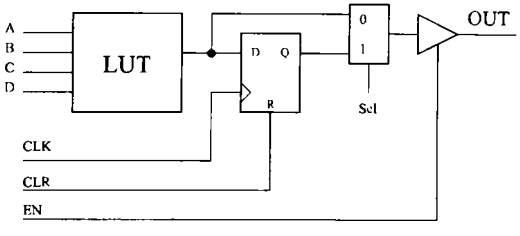


Figura 2.19. Model de arhitectură pentru blocul logic.

Modelul schematic considerat este imaginat integral de autor și conține toate elementele necesare studiului: un LUT cu k intrări, un bistabil D care poate fi șuntat prin intermediul unui multiplexor 2:1 și un etaj de ieșire trei-stări. Față de modelele similare propuse în literatură,

de exemplu în [Brown92] şi [Rose93], schema propusă în figura 2.19 reflectă mai exact tendinţele actuale de structurare a blocului logic din FPGA moderne. Prin LUT se poate implementa orice funcţie logică de 4 variabile, iar aranjamentul cu multiplexor şi buffer trei stări neinversor de la ieşire este o replică fidelă a etajelor de ieşire din FPGA actuale. Prin aceasta au fost cuprinse într-un model simplu, uşor de simulat, principalele caracteristici ale blocului logic din FPGA actuale.

Pentru blocul logic din figura 2.19, aria ocupată constă dintr-o componentă fixă, aferentă bistabilului, multiplexoarelor, logicii aleatoare şi una variabilă (dependentă de k) corespunzătoare LUT. În acest caz, aria ocupată de resursele logice din cadrul unui bloc este:

$$A_{bloc\_logic} = A_{fixă} + m \cdot A_{bit} \cdot 2^k \tag{2.1}$$

Pentru un proces CMOS de 1,2 um, aria fixă este de 5100 μm^2 dacă blocul logic conţine bistabilul reprezentat în figura 2.19 şi de 2100 μm^2 în caz contrar.

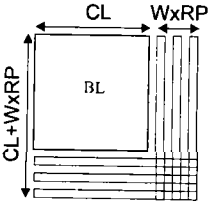


Figura 2.20. Arhitectura modelului blocului logic.

Conform reprezentării din figura 2.20 se poate scrie aria totală ocupată de un bloc, luând în considerare şi resursele folosite pentru rutare:

$$A_{bloc} = (CL + w \cdot RP)^2 = \left(\sqrt{A_{bloc\_logic}} + w \cdot RP\right)^2 \tag{2.2}$$

iar în final, aria totală pentru un circuit dat devine:

$$A_{totală} = N_{bloc} A_{bloc} \tag{2.3}$$

2.3.2. REZULTATE EXPERIMENTALE

Pentru sintetizorul de frecvenţă prezentat detaliat în capitolul 4 am studiat dependenţa ariei blocului logic de valoarea lui k în diverse variante de implementare. Un prim rezultat este redat în figura 2.21. Aria blocului logic (ABL) este măsurată în mii de μm^2 şi creşte după cum este şi firesc cu creşterea lui k. Numărul de blocuri logice este cu atât mai redus cu cât k este mai mare.

Produsul celor două curbe este graficul cel mai interesant, acesta reprezentând aria totală (este vorba de aria *logicii*) ocupată în funcţie de k. Această dependenţă a fost reprezentată alături de aria blocului logic în figura 2.22, respectiv de numărul de blocuri logice în figura 2.23. Ambele *grafice indică prezenţa unui minim* pentru k cuprins între 3 şi 4.

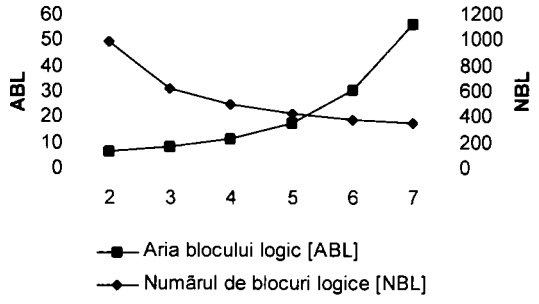


Figura 2.21. Aria blocului logic şi numărul de blocuri logice în funcţie de *k*.

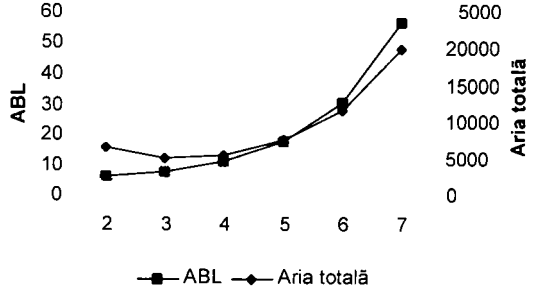


Figura 2.22. Aria totală şi aria blocului logic în funcţie de *k*.

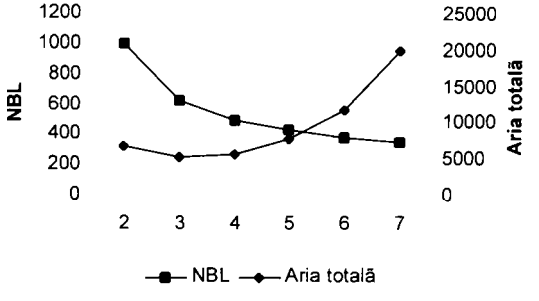


Figura 2.23. Aria totală şi numărul de blocuri logice în funcţie de *k*.

Interesant este şi rezultatul prezentat în figura 2.24, care redă dependenţa dintre aria totală ocupată de un bloc logic (ţinând cont de resursele de rutare) în funcţie de *k*. Se confirmă

minimul pentru $k = 4$, cu mențiunea că pentru cazul $k = 3$ diferența este nesemnificativă (4%).

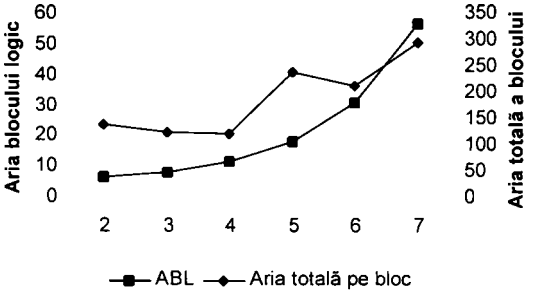


Figura 2.24. Aria totală și numărul de blocuri logice în funcție de *k*.

În finalul acestui subcapitol mai trebuie subliniate două aspecte importante. În primul rând, deși modelul introdus în figura 2.19 este convenabil de studiat, putând fi modelat cu un număr mic de parametri, FPGA bazate pe LUT domină net acest segment de piață (preluând aici enumerarea din [Gontean97], distribuția cotelor de piață în 1993 era de 66% pentru Xilinx cu o arhitectură a blocului logic bazată pe LUT, de 15% pentru Actel și de 9% pentru AT&T, ceilalți competitori deținând o pondere practic neglijabilă).

În al doilea rând, este interesantă orientarea principalului producător mondial de FPGA de la LUT cu 5 intrări în cazul familiei Xilinx 3000 la LUT cu 4 intrări în cazul familiei mai noi Xilinx 4000. Deși nu este motivată în [Xilinx94], această migrarre demonstrează clar justețea studiului efectuat și a rezultatelor obținute.

Concluziile prezentate validează schema din figura 2.19 ca un bloc versatil ce poate fi utilizat ca bază arhitecturală pentru FPGA simetrice în tehnologie SRAM. Acest etaj este de complexitate mai mare decât toate arhitecturile de granularitate fină prezentate în cadrul acestui capitol, dar mai puțin complex decât blocul logic al seriei Xilinx 3000.

Acest studiu va fi continuat în viitor pentru LUT cu mai multe ieșiri și pentru blocuri logice cu mai multe LUT (cazul $m \neq 1$).

2.4. CARACTERISTICI DE METASTABILITATE A SLP

La interconectarea a două sisteme numerice care funcționează pe baza unor semnale de tact diferite trebuie implementat un mecanism de sincronizare între cele două sisteme, orice intrare trebuind sincronizată cu semnalul de tact al respectivului sistem. Soluția clasică este de a utiliza un bistabil D, ca în figura 2.25. Folosirea acestei scheme presupune implicit că în toate cazurile sunt satisfăcute condițiile impuse de timpii de stabilire și de menținere, ceea ce nu este neapărat adevărat. Semnalul la intrarea D se poate modifica chiar în vecinătatea tranziției active a semnalului de tact, ceea ce poate duce la plasarea bistabilului într-o a treia stare, numită *metastabilă*, în care tensiunea de ieșire este undeva între nivelul de tensiune SUS și cel JOS.

Teoretic, ieşirile Q şi $\overline{Q}$ pot rămâne un timp nedefinit în această stare (*comportare metastabilă*), fiind de asemenea posibil ca cele două ieşiri să oscileze de câteva ori între nivelurile SUS şi JOS, înainte de stabilizare (*comportare oscilatorie*).

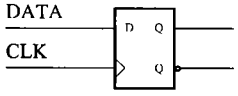


Figura 2.25. Un bistabil D folosit pentru sincronizare.

Comportarea metastabilă se referă la circuite având un timp de propagare redus comparat cu timpul de ridicare (figura 2.26), iar cea oscilatorie este comună circuitelor cu un timp de propagare ridicat faţă de timpul de ridicare (figura 2.27). Această discuţie se va limita la comportarea metastabilă (figura 2.28), fiind cunoscute şi publicate puţine date despre comportarea oscilatorie. Denumirea "metastabilă" provine de la faptul că orice perturbaţie poate comuta bistabilul în una din cele două stări stabile.

O dată ce un bistabil a ajuns în starea metastabilă, nu există un timp de stabilire finit după care să se poată garanta (cu probabilitatea 1) că ieşirea a ajuns într-o stare validă. În aceste condiţii sarcina circuitului de sincronizare este de a reduce rata de erori la un nivel acceptabil, aceasta semnificând frecvenţa cu care intrările asincrone determină comportări metastabile după un anumit timp de stabilire. Nivelurile de ieşire nedeterminate pot produce rezultate necontrolabile dacă este permisă propagarea lor prin sistem. Probabilitatea unui asemenea eveniment depinde de lăţimea ferestrei şi de frecvenţa semnalului de sincronizat.

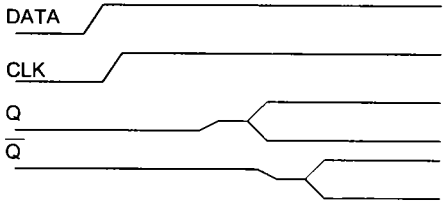


Figura 2.26. Circuit de sincronizare cu comportare metastabilă.

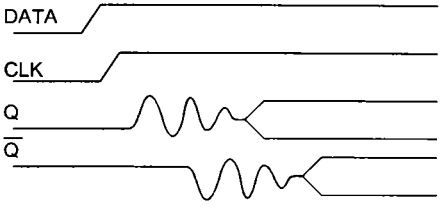


Figura 2.27. Circuit de sincronizare cu comportare oscilatorie.

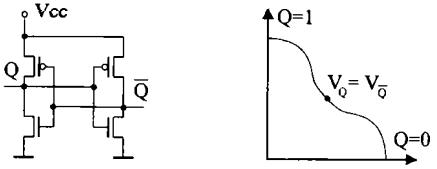


Figura 2.28. Caracteristica de metastabilitate la un bistabil CMOS.

Ecuația de bază pentru MTBF este:

$$\frac{1}{MTBF} = f_{CLOCK} \cdot f_{DATA} \cdot C_1 \cdot e^{-C_2 \Delta T}, \tag{2.4}$$

în care:

- ΔT reprezintă timpul de stabilire;
- constantele C_1 și C_2 reflectă caracteristici particulare ale circuitului și (mai important) procesul tehnologic folosit la fabricare;
- f_{CLOCK} și f_{DATA} reprezintă frecvența semnalului de tact, respectiv a datelor de la intrarea de sincronizare.

Reprezentând ecuația de mai sus într-un sistem de coordonate semilogaritmic, raportul $MTBF / \Delta T$ apare ca o linie dreaptă (figura 2.29). Constanta C_1 scalează liniar ecuația MTBF (cu cât mai mică este C_1, cu atât este mai mare este MTBF). C_1 afectează curba $MTBF / \Delta T$ într-un sens absolut, translatând-o de-a lungul axei MTBF (Oy). Constanta C_2 afectează panta curbei $MTBF / \Delta T$, deci este o măsură (într-un sens relativ) a timpului în care bistabilul iese din starea metastabilă. Cu cât panta este mai abruptă, cu atât mai bună este stabilitatea. Ecuația indică că MTBF este o funcție liniară cu frecventa de tact și f_{DATA}.

În [Gontean93] este prezentat un montaj original utilizat pentru evaluarea constantelor din relația (2.1), prin determinarea numărului de evenimente metastabile în unitatea de timp. În acest scop se folosesc de obicei comparatoare de mare viteză, greu accesibile și mai ales scumpe.

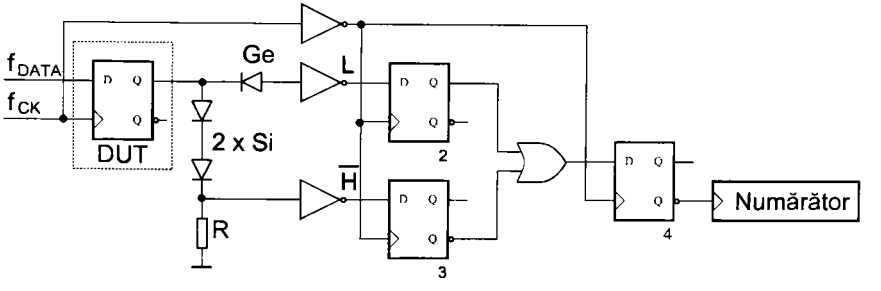


Figura 2.29. Sistemul de măsură original propus.

Originalitatea aranjamentului din figura 2.29 constă tocmai în utilizarea tehnicii TTL (ieftine şi larg răspândite) pentru depistarea evenimentelor metastabile. În cadrul acestui experiment se consideră că circuitul testat (DUT) se găseşte într-o stare metastabilă atunci când tensiunea la ieşirea sa Q este mai mare de V_{ILMax} şi mai mică decât V_{IHmin}, pentru o perioadă de timp mai mare decât durata unei tranziţii între două niveluri logice. În cazul din figura 2.29, aceasta se traduce prin $0{,}8\ V < V_Q < 2{,}0\ V$.

Compararea nivelurilor de tensiune se obţine cu o schemă simplă dar eficientă, cu diode şi inversoare care mută pragul de comutare al inversoarelor la 0,8, respectiv 2 V. Prin aceasta se obţine o precizie satisfăcătoare (sub 0,1 V eroare de nivel determinată experimental) la o întârziere de câteva nanosecunde, greu de atins altfel chiar cu cele mai rapide comparatoare. Prin aplicarea semnalului de tact inversat la intrările bistabilelor 2 şi 3, se obţine un semnal de validare a comparării tensiunii V_Q cu cele două praguri amintite anterior. Un numărător extern contorizează numărul de evenimente metastabile pe durata experimentului.

Rezultatele experimentale privind metastabilitatea SLP conduc la următoarele concluzii:

- caracteristicile de metastabilitate ale componentelor dintr-o familie dată sunt virtual identice, din moment ce ele depind mai mult de tehnologie decât de circuitul individual;
- componentele MAX au caracteristici de metastabilitate substanţial mai bune decât cele ale seriei EP pentru că sunt fabricate cu o tehnologie rapidă de 0,8μm;
- EPM5000 de MAX EPLD-uri au o metastabilitate echivalentă cu familia FAST-TTL. Într-un asemenea EPLD, bistabilele pot fi folosite ca sincronizatoare cu caracteristici echivalente;
- în cazul în care se cere timp de stabilire foarte mic, familia AS-TTL furnizează caracteristici superioare EPLD-urilor;
- integrând niveluri logice multiple într-un singur EPLD se poate adesea reduce necesitatea unor timpi de stabilire individuali mici. Opţiunea aceasta este disponibilă când performanţele circuitului nu sunt critice.

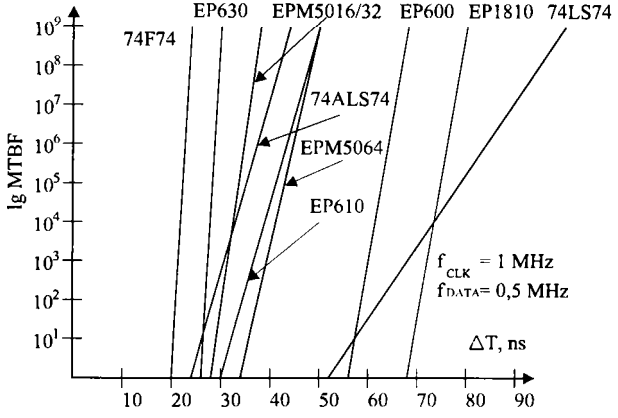


Figura 2.30. Funcţia MTBF şi efectul parametrilor.

Tabelul 2.4 sintetizează valorile C_1 şi C_2 pentru EPLD-uri şi circuite TTL. De la aceste valori, calcularea MTBF poate fi făcută pentru o combinaţie particulară a frecvenţei tactului şi a datelor de la intrare.

Următorul exemplu presupune folosirea EPM5032 într-o aplicaţie de sincronizare şi necesită un MTBF de un an ($\approx 3 \cdot 10^7$ s). Frecvenţa tactului este de 10MHz. Pentru a calcula timpul minim de stabilire permis şi a aprecia MTBF, constantele din tabel sunt folosite pentru rezolvarea ecuaţiei de metastabilitate. Timpul de stabilire necesar este:

$$\Delta T = \frac{\ln[MTBF \cdot F_{CLOCK} \cdot F_{DATA} \cdot C_1]}{C_2}.$$ Înlocuind rezultă: $\Delta T = 39\ ns$.

Pentru a obţine un MTBF de un an cu un EPM5032, aplicaţia ar trebui să permită un timp de stabilire de aproximativ 39 ns înainte ca ieşirea macrocelulei de sincronizare a EPM5032 să fie utilizată în altă parte în sistem. Calcule similare pot fi făcute pentru orice combinaţie de MTBF-uri şi frecvenţe de date/tact. În conformitate cu relaţia logaritmică între MTBF şi timpul de stabilire, modificări esenţiale în MTBF pot fi obţinute pentru modificări mici ale ΔT. De exemplu, dacă MTBF necesar este crescut de la un an la 10 ani, ΔT trebuie modificat numai cu 2 ns (41 ns). Creşterea marginii de siguranţă la proiectare este necostisitoare şi trebuie examinată pentru fiecare aplicaţie în parte.

Tabelul 2.4

Constantele C_1 şi C_2 în funcţie de circuit

Circuit	C_1	C_2
EP630	3.081E+52	5.415
EP610	4.567E+12	1.919
EP600	1.916E+22	1.340
EP1810	3.529E+38	2.068
EPM5016/EPM5032	4.981E+20	2.482
EPM5064	4.747E+11	1.641
74F74	6.100E+21	4.000
74LS74	5.270	0.5081

În finalul acestui subcapitol trebuie menţionat că un montaj concret de evaluare a comportării metastabile depăşeşte cadrul teoretic al unei discuţii academice - el oferă un criteriu clar, cantitativ de selecţie a celui mai potrivit SLP pentru o aplicaţie dată, bineînţeles din punctul de vedere al metastabilităţii.

2.5. CONCLUZII

Toate sintezele arhitecturale prezentate în cadrul acestui capitul sunt originale. Pentru SLP de complexitate redusă se evidenţiază posibilitatea integrării memoriilor PROM în cadrul mai larg al SLP.

O atenţie deosebită este acordată în acest capitol structurilor complexe FPGA. Este propusă o clasificare a FPGA în funcţie de arhitectura blocului logic, pe criteriul posibilităţii de implementare într-un singur bloc a unei funcţii de trei variabile.

Posibilitățile actuale de utilizare a FPGA sunt amintite alături de câteva perspective viitoare de folosire a acestor structuri, dintre care unele semnificative sunt descrise în continuare:

- în viitorul apropiat se întrevede o creştere a numărului de arhitecturi pentru FPGA;
- dezvoltarea FPGA orientate pe aplicație, în domenii cum ar fi transmisiile de date, DSP, AES, microprocesoare; interesantă aici este direcția prefigurată în [Gavrilescu97];
- apariția de sisteme hardware şi software de emulare şi de testare pentru FPGA provenind de la mai mulți producători;
- dezvoltarea unor arhitecturi de blocuri logice neomogene; acestea promit obținerea unui compromis mai bun între suprafața ocupată şi performanțele obținute;
- proliferarea variantelor de rutare segmentate, care oferă avantaje majore față de variantele nesegmentate, cu doar una sau două lungimi posibile pentru segmente;
- programele CAD pentru dezvoltare cu/de FPGA vor deveni tot mai sofisticate şi mai performante, favorizând dezvoltarea în continuare a arhitecturilor şi sporirea performanțelor;
- este posibil ca succesul FPGA în domeniul circuitelor numerice să fie reprodus în domeniul circuitelor analogice de FPAA (**F**ield **P**rogrammable **A**nalog **A**rray).

Modelul original de structură introdus în figura 2.19 a fost utilizat ca bază de calcul pentru determinarea ariei totale ocupate de blocul logic. S-a demonstrat existența unui minim (cu diferențe minore între cele două cazuri) pentru $k = 3$, respectiv 4. Cazul studiat se referă la FPGA simetrice la care blocul logic este realizat cu un LUT cu k intrări şi o ieşire. El poate fi extins în primul rând în cazul LUT cu ieşiri multiple, respectiv la blocuri logice alcătuite din mai multe LUT.

Pentru studiul metastabilității a fost imaginat un montaj experimental simplu şi eficient care poate fi utilizat la studiul comportării metastabile a oricărui sistem numeric. Pe baza acestui montaj au fost măsurate alături de circuitele TTL o serie de SLP pe care le-am avut la dispoziție.

Dispozitivele fabricate în aceeaşi tehnologie au caracteristici similare de metastabilitate, iar "trucurile" şi optimizările de proiectare nu au un efect marcant.

Un circuit de sincronizare bine proiectat trebuie să ofere margini de siguranță adecvate. Un circuit poate avea un MTBF de 10 ani şi totuşi să greşească în primele minute de funcționare. În aplicații la care riscul minim este primordial, metastabilitatea nu poate fi niciodată neglijată.

Seria MAX EPM5000 prezintă caracteristici de metastabilitate comparabile cu bistabilele ALS (Advanced-Low-Power-Schottky) până la FAST (Fairchild-Advanced-Schottky TTL). Seria EP prezintă o metastabilitate mai bună decât cea a bistabilelor LS.

2.5.1. EVOLUȚII ŞI PERSPECTIVE FPGA

Facilitatea de reconfigurare cu consecința imediată de a corecta o eroare sau de a schimba o configurație este foarte importantă. Autorul consideră însă că logica programabilă

reconfigurabilă va avea un impact mult mai profund legat de posibilitatea *reconfigurării dinamice* a structurii, pentru a se adapta mai bine necesităților. FPGA actuale dispun de câteva zeci de mii de porți echivalente, dar se poate estima sporirea acestui număr la milioane de porți în următorii ani. Ne putem aştepta la dezvoltarea impetuoasă a procesării paralel şi a tehnicii pipe-line implementate în SLP. De exemplu, un proces grafic poate fi prelucrat de trei procesoare (pentru calcule tridimensionale), iar sistemul se poate reconfigura dinamic în două procesoare utilizate pentru modelare bidimensională sau calcule legate de afişarea pe ecran, un singur FPGA putând fi utilizat pentru întreaga implementare. Cu toate că asemenea posibilități apar ca futuristice în prezent, ele doresc să sublinieze că SLP în general şi FPGA în special, nu sunt numai o nouă familie de circuite VLSI, ci un *concept* care deschide perspectiva unor posibilități nebănuite acum câțiva ani.

2.5.2. EVALUAREA ARHITECTURII SLP

Nu există în prezent o arhitectură universală pentru SLP în general şi FPGA în special. Alegând complexitatea drept criteriu principal de clasificare (Capitolul 1, §1.4) se pot emite aprecieri pertinente asupra celei mai performante arhitecturi în cadrul fiecărei categorii de SLP. Astfel, pentru SLP de complexitate redusă, cea mai potrivită alegere este formată de circuitele GAL, datorită avantajelor prezentate:

- consum redus;
- reprogramare electrică şi simplă (programarea se face serial, utilizând trei pini);
- timp de propagare redus.

Pentru SLP de complexitate medie, alegerea este de asemenea relativ uşor de făcut. În acest caz, circuitele CPLD Altera oferă avantaje asupra altor circuite similare, datorită eleganței arhitecturii oferite şi posibilității de reprogramare. Principalul competitor direct, reprezentat de familiile Actel Act1 şi Act 2 deşi oferă un timp de propagare mai redus nu pot fi reprogramate şi deci nu sunt potrivite pentru prototipuri şi serii mici de producție.

Pentru FPGA, alegerea este relativ dificilă, deoarece deşi deosebirile arhitecturale sunt pregnante, utilizatorului îi sunt oferite un mare număr de resurse logice, fiecare variantă oferind posibilitatea implementării proiectului dorit. De aceea, autorul propune în acest caz alte criterii de departajare, şi anume posibilitatea reconfigurării dinamice a FPGA. Modificarea structurii hardware a unui asemenea circuit prin comenzi software va putea duce într-un viitor apropiat la înlocuirea upgrade-ului fizic cu o simplă reconfigurare software, cel mai probabil de realizat prin Internet.

Din experiența proprie, comparând diversele arhitecturi disponibile, am ajuns la concluzia că cea mai performantă arhitectură se bazează pe LUT cu 5 sau 4 intrări, implementate în seriile XC3000 şi XC4000 ale firmei Xilinx (care sunt de asemenea şi reconfigurabile). Această structură se apropie cel mai mult de ceea ce am putea încerca a defini ca FPGA-ul ideal.

2.6. REFERINȚE BIBLIOGRAFICE

Cu excepția capitolului 1 (care tratează de altfel un subiect de interes general pentru implementările cu circuite integrate numerice), acest capitol este cel mai bine reprezentat în lite-

ratură. Din motive evidente, cataloagele firmelor producătoare sunt generoase cu detalii referitoare la SLP-urile proprii. Aici se pot menționa [AMD88,93], [Lattice88], [Philips91] și [TI93a] pentru SLP de complexitate mică și medie. Pentru FPGA se pot consulta suplimentar [Actel92a,b], [Altera91,93], [Intel88,93], [TI93b,c] și [Xilinx94]. Toate aceste cataloage pentru FPGA conțin și exemple concrete de implementare, articole republicate, note de aplicații, fiind un ghid excelent pentru cititorul doritor de cunoștințe suplimentare.

[Ștefan83] este prima lucrare din literatura română în care apare subiectul PLA în capitolul redactat de Dl. Profesor Tiberiu Mureșan tratează. Tot aici Domnia Sa propune în premieră denumirea de "Structuri logice programabile" pe care am preluat-o consecvent. Descrierea arhitecturii și funcționării SLP este tratată și în alte câteva cărți, dintre care, din păcate, unele au devenit deja parțial depășite: [Bostock87], [Alford89], [Bolton90], [Broesch91].

La nivel mediu se prezintă [Blank92]; o altă carte din literatura germană [Auer95] nu spune multe lucruri suplimentare față de cataloagele mai sus menționate.

Capitole despre PLD se găsesc în [Wakerly90], [Hill93] și [Mureșan96]. Nu am găsit nici o referință care să trateze *simultan* subiectul PLD și FPGA, acesta fiind unul din motivele pentru redactarea [Gontean97].

Un articol important de sinteză privind arhitectura FPGA este [Rose93], text suprapus parțial cu [Brown93]. Ambii autori sunt profesori la Universitatea din Toronto și fac parte dintr-una din cele mai puternice echipe de specialiști în domeniul FPGA.

Metastabilitatea circuitelor numerice este tratată didactic într-o serie de lucrări ca de exemplu în [Wakerly90], [Dillinger88] și [Hill93]. În [Altera91] este prezentată schema bloc pentru un montaj practic de determinare a MTBF, iar în [Gontean93] este prezentat un montaj experimental complet de evaluare a metastabilității și principalele rezultate experimentale obținute.

Rezumând principalele contribuții bibliografice ale autorului referitoare la acest capitol, ele sunt:

- [Gontean92b] și [Gontean92c] pentru evoluțiile semnificative pe piața SLP și perspectivele dezvoltării în continuare ale acestui domeniu de mare interes;
- [Gontean93a] prezintă principalele aspecte referitoare la un subiect mai puțin tratat în literatură, și anume testarea structurilor logice programabile. Acest subiect este dezvoltat ulterior de autor în [Gontean94a] și [Gontean94b].
- [Gontean93b] introduce montajul experimental de determinare a caracteristicilor de metastabilitate pentru orice circuit numeric;
- [Gontean94c] prezintă rezultatele determinărilor parametrilor de metastabilitate pentru SLP și concluziile desprinse de aici.
- [Gontean96a] este un model de proiectare cu FPGA a unui controler de memorii folosind un registru LSFR, tratat în [Mureșan96];
- [Gontean97] este cartea ce tratează simultan problemele PLD și FPGA și introduce o serie de aplicații.

CAPITOLUL 3

CONTRIBUȚII LA STUDIUL RUTĂRII ÎN FPGA

3.0. INTRODUCERE

Rutarea este un proces iterativ, care necesită resurse sporite de calcul, care sunt cu atât mai mari cu cât este mai complex circuitul ce urmează a fi rutat. Făcând apel doar la unul din multiplele exemple recente din literatură, în [Lienig97] este prezentat un caz în care o rutare necesită o rețea de 14 calculatoare Sun, ce rulează același program un timp relativ ridicat. O asemenea abordare este evident foarte scumpă și nu garantează reușita operației de rutare. În caz de nereușită, de multe ori se alege un FPGA mai complex, care este folosit doar parțial.

De aceea apare ca deosebit de convenabilă punerea la punct a unei metode de evaluare a șanselor de rutare cu succes a unui proiect dat *înainte* ca această rutare să fie efectiv pornită. Această analiză calitativă, al cărei rezultat este o probabilitate de rutare cu succes a proiectului, dar care nu se preocupă de modul în care această rutare poate fi efectuată, necesită o putere de calcul substanțial scăzută și un timp de calcul neglijabil.

În acest capitol autorul studiază un model de structură pentru FPGA simetrice ce poate fi utilizat pentru prezicerea rutabilității. Sunt deduse relații probabilistice de efectuare a unor conexiuni și este verificată pe un caz concret o presupunere de distribuire probabilistică a pistelor ocupate într-un canal de legătură. Implementarea fizică pe care au fost efectuate toate experimentele este prezentată în detaliu în capitolul 4.

La sfârșitul capitolului anterior am desemnat FPGA Xilinx ca fiind cea mai performantă structură din clasa sa. Modelul studiat se referă la FPGA simetrice, având ca principal exponent structurile produse de firma Xilinx.

3.1. DEFINIȚII

Datorită complexității implicate, în scopul rutării unui proiect se utilizează de obicei etapele propuse în [Lorenzetti89]:

1. partiționare;
2. utilizarea unui ruter global;
3. utilizarea unui ruter local.

Prin *partiționare* se urmărește divizarea resurselor de rutare în zone potrivite pentru utilizarea algoritmilor de rutare existenți. *Ruterul global* creează un nou set de probleme de rutare cu restricțiile aferente, fără a asigna canale sau segmente de legătură. Ruterul global supraveghează utilizarea canalelor de legătură între resursele logice partiționate anterior.

Prin utilizarea unui *ruter local* se aleg segmentele şi switch-urile pentru fiecare conexiune, în cadrul restricțiilor stabilite de ruterul global.

Prin *varianta de rutare* se înțelege modalitatea prin care switch-urile programabile şi segmentele de legătură se interconectează pentru a realiza accesul la blocurile logice. FPGA produse de firme diferite conțin propriile variante specifice de rutare, diferite între ele.

Se defineşte *flexibilitatea blocului C de conectare* F_c ca fiind numărul de piste adiacente blocului la care se poate conecta fiecare pin al unui bloc logic. În figura 3.1.a, au fost reprezentate două blocuri logice interconectate printr-un bloc C, la care $F_c = 2$, deoarece fiecare pin al blocului logic se poate conecta la două piste verticale. Valoare minimă pentru F_c este 1.

Flexibilitatea F_s *a blocului S de conectare* reprezintă numărul de piste la care se poate lega orice pistă ce intră în blocul de conectare. Valoarea minimă pentru F_s este 2, deşi în practică nu se produc circuite cu $F_s < 3$ pentru flexibilitatea procesului de rutare.

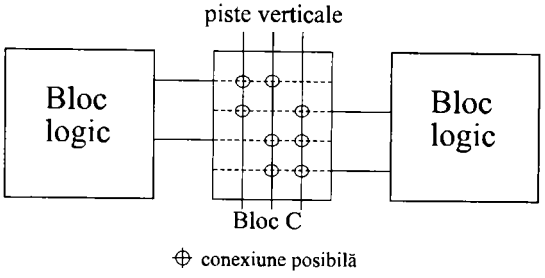


Figura 3.1.a. Bloc C de conectare cu flexibilitatea $F_c = 2$.

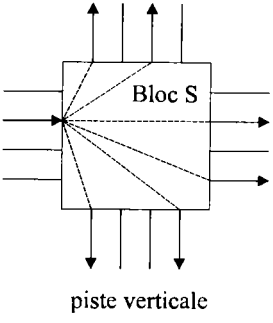


Figura 3.1.b. Bloc S de conectare cu flexibilitatea $F_s = 6$.

Blocul S din figura 3.1.b are flexibilitatea F_s = 6, aceasta însemnând că *fiecare* pistă de intrare se poate conecta cu 6 piste de ieşire. Pentru simplitate au fost reprezentate cele 6 conexiuni posibile doar pentru o singură pistă de intrare.

3.2. VARIANTA DE RUTARE Xilinx

Cele mai cunoscute FPGA produse de Xilinx sunt familiile 3000 şi 4000. Datorită tehnologiei SRAM utilizate, dimensiunea fiecărei legături este mare, iar în Xilinx 3000 blocul de conectare leagă tipic fiecare pin al unui bloc logic la două sau trei (din cele cinci) piste din vecinătatea unui bloc, aşa cum este în figura 3.2. Pe toate cele patru laturi ale unui bloc logic se găsesc blocuri de conectare care conectează un număr de 11 pini ai blocului logic la segmentele de legătură. Conexiunile sunt implementate cu tranzistoare de legătură (trecere) pentru pinii de ieşire, respectiv cu multiplexoare pentru pinii de intrare. Utilizarea multiplexoarelor reduce numărul de celule SRAM necesare pentru fiecare pin. Blocul de switch-uri realizează conexiuni între segmentele de legătură în punctele de intersecţie dintre canalele verticale şi cele orizontale. După cum se observă din figura 3.2, fiecare segment de legătură se poate conecta la un subset al segmentelor de legătură de pe laturile opuse ale blocului de conectare. Fiecare segment se poate conecta uzual la cinci sau şase (din 15) segmente de legătură de pe laturile opuse, acest număr fiind limitat de dimensiunea şi capacitatea parazită a switch-urilor din tehnologia SRAM.

În familia Xilinx 3000 sunt disponibile patru tipuri de segmente de legătură:

- de uz general, constând din segmente de legătură care trec prin blocul S;
- interconexiune directă, constând din segmente de legătură care conectează fiecare ieşire a blocului logic, direct la cele mai apropiate patru blocuri logice vecine, aşa cum sugerează liniile negre groase ce ies din blocul logic central din figura 3.2;
- linii lungi, care traversează lungimea sau lăţimea cipului, asigurând ieşiri cu fan-out ridicat şi întârziere uniformă;
- o linie de tact, materializată printr-o singură reţea ce străbate întreg circuitul, comandată de un driver. Această linie este conectată doar la intrările de tact ale bistabilelor şi asigură un defazaj redus al semnalului de tact.

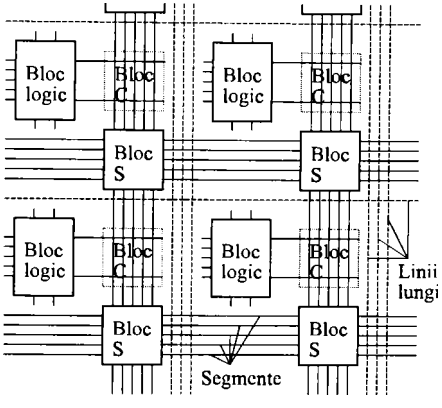


Figura 3.2. Varianta Xilinx 3000.

Familia Xilinx 4000 este asemănătoare, cu următoarele deosebiri:

- sunt 18 piste de legătură de uz general pentru fiecare canal în loc de 5;
- posibilitățile de conectare a blocurilor logice sunt sporite, fiecare bloc logic putându-se poate lega la aproape orice pistă;
- patru dintre piste trec prin switch-uri numai la fiecare al doilea bloc de switch-uri (figura 3.3), prin aceasta asigurându-se o densitate mai mare de integrare.

În figura 3.3, S este un switch de rutare, iar segmentele de lungime unitară nu au mai fost reprezentate.

Pentru familia 3000, catalogul [Xilinx94] specifică parametrii de rutare: $F_c = 0{,}6 \cdot W$ şi $F_s = 6$. În [Brown92] este prezentat un rezultat extrem de interesant, care arată că mărind F_c la $0{,}8 \cdot W$ se obține rutarea cu număr minim de piste posibil. Acest rezultat este studiat în continuare în acest capitol, observând influența creşterii F_c asupra rutabilității.

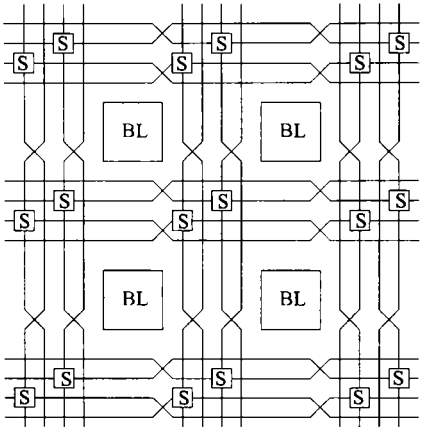


Figura 3.3. Varianta de rutare Xilinx 4000 cu segmente de lungime dublă.

3.3. MODEL DE STRUCTURĂ PENTRU UN FPGA SIMETRIC

Modelul general prezentat în figura 2.4 a fost adaptat pentru a satisface cerințele studiului abordat. *Blocul C* de conectare leagă intrările/ieşirile blocurilor logice la canalele de rutare. *Blocul S de switch-uri* interconectează segmente orizontale şi verticale. S-au făcut următoarele presupuneri:

- se studiază un FPGA simetric, cu N blocuri logice pe fiecare latură;
- atât blocurile C cât şi S conțin switch-uri;
- pistele traversează blocurile C (fără a trece prin switch-uri);
- un switch este obligatoriu pentru traversarea unui bloc S.
- switch-urile din blocurile C sunt folosite pentru conectarea pinilor la segmente, iar cele din blocurile S pentru interconectarea segmentelor.

Un proiect cu un număr total de C_t conexiuni se rutează într-un FPGA N x N. Lungimea fiecărei conexiuni se supune unei legi de repartiție, presupusă geometrică, cu lungimea medie

$\overline{R}$ [Heller84] și are următoarea reprezentare fizică: o conexiune se va termina într-un bloc C cu probabilitatea $\frac{1}{\overline{R}}$ și va continua către alt bloc C cu probabilitatea $1-\frac{1}{\overline{R}}$.

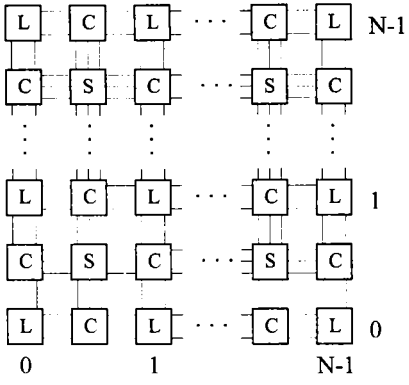


Figura 3.3. Model de structură pentru un FPGA simetric.

Conexiunile C_T se notează cu $C_1, C_2, ..., C_{C_T}$ și probabilitatea ca fiecare conexiune să poată fi rutată este $P(R_{C_1}), P(R_{C_2}), ..., P(R_{C_{C_T}})$. În aceste condiții rutabilitatea se scrie:

$$Rutabilitatea = \frac{1}{C_T}\sum_{i=1}^{C_T} P(R_{C_i}). \tag{3.1}$$

În [Gamal81] este prezentat un model stocastic utilizat la prezicerea necesarului de resurse de legare în circuite integrate Master Slice, care prezintă o arie bidimensională de celule identice cu canale de rutare între rândurile și coloanele de celule. Se presupune că numărul de conexiuni la o celulă se supune repartiției Poisson cu parametrul λ, unde λ este definit ca fiind raportul dintre numărul total de conexiuni din circuit și numărul total de celule. Lungimea medie a unei conexiuni (reprezentată de numărul de celule traversat) este $\overline{R}$. Celelalte restricții prezente în lucrare nu sunt necesare pentru cazul studiat în acest capitol.

Concluzia la articolul citat este că într-o arie de N x N canale, densitatea segmentelor urmează repartiția Poisson, cu densitatea medie $\frac{\lambda\overline{R}}{2}$, pentru $\overline{R} < \infty$ și este *independentă* de N. Cu toate că rezultatele din [Gamal81] se referă la circuite Master Slice, ele se extind și la FPGA datorită similarității arhitecturii. Prin urmare este convenabil a prezice densitatea canalelor de rutare în FPGA folosind rezultatele descrise anterior. Acuratețea predicțiilor se poate verifica comparând repartiția Poisson cu media $\frac{\lambda\overline{R}}{2}$ cu distribuția densității canalelor de rutare într-un FPGA real. În figura 3.7 este prezentată o asemenea comparație, între o implementare dintr-un FPGA real și predicția ce rezultă din aplicarea rezultatelor lui Gamal. După cum se poate observa din figură, realitatea se apropie extrem de mult de distribuția Poisson ideală.

Se va da în continuare o demonstrație fizică a utilizării repartiției Poisson.

Presupunând că FPGA-ul are W piste în fiecare canal, iar pentru fiecare din cele W piste fie p_1 probabilitatea ca pista să fie ocupată în cadrul unui proiect rutat în FPGA. Dacă $W = 1$, fie p_1 probabilitatea ca pista să fie ocupată de o conexiune. Dacă $W = 2$, p_2 este probabilitatea ca fiecare din cele două piste să fie utilizate pentru conexiuni. Evident $p_2 < p_1$. Rezultă în cazul general $W = n$, probabilitatea p_n ca fiecare pistă să fie ocupată şi $p_n < ... < p_2 < p_1$. Pentru $n \to \infty$, $p_n \to 0$.

Deoarece p_n este mică, evenimentul ca o pistă să fie ocupată este *rar*, iar numărul acestor evenimente (densitatea) poate fi aproximat de legea de repartiţie Poisson:

$\Psi(k) = \frac{\mu^k e^{-\mu}}{k!}$, unde μ este media (sau dispersia).

Pe de altă parte într-un FPGA real, o pistă poate traversa mai multe blocuri logice (şi nu doar unul ca în modelul de mai sus). Aici trebuie menţionat că o pistă lungă atrage după sine o capacitate parazită importantă şi o diafonie crescută, cu alte cuvinte cade în sarcina programului de rutare să minimizeze numărul de astfel de asemenea conexiuni (sunt frecvente proiectele *fără* nici o astfel de linie lungă [Actel92b]). De aceea consider că presupunerea că toate pistele din FPGA sunt constituite din segmente scurte nu restrânge generalitatea rezultatelor din acest capitol.

3.4. CALCULUL PROBABILITĂŢII DE RUTARE

3.4.1. PROBABILITATEA RUTĂRII UNEI CONEXIUNI

Conexiunea C_i din figura 3.4 este realizată între blocurile logice de coordonate (x_1, y_1) şi (x_2, y_2). Conform [Gamal81], lungimea conexiunii este:

$$LC_i = |x_2 - x_1| + |y_2 - y_1|. \qquad (3.2)$$

Numărul de blocuri S traversate de conexiunea C_i este $LC_i - 1$. Fie $LC_i = n + 1$. Evenimentul "conexiunea C_i a fost rutată cu succes" va fi notat cu L_{n+1}. În acest context:

- X_1 este evenimentul ca pinul blocului logic asociat cu C_i la (x_1, y_1) să se poată conecta cu măcar o pistă a primului bloc C. (Prin definiţie sunt F_c piste care se pot conecta la pinul respectiv, dar o parte din ele pot fi deja ocupate)
- S_1, S_2, ..., S_n sunt evenimentele ca C_i să fie rutată cu succes o pistă a primului, celui de-al doilea, al n-lea bloc S. Sunt $LC_i - 1$ asemenea evenimente pentru conexiunea C_i.
- X_2 este evenimentul ca ultimul bloc C să poată fi conectat la blocul logic de coordonate (x_2, y_2).

Evident:

$$R_{C_i} | L_{n+1} = X_1 \cap S_1 \cap S_2 ... \cap S_n \cap X_2. \qquad (3.3)$$

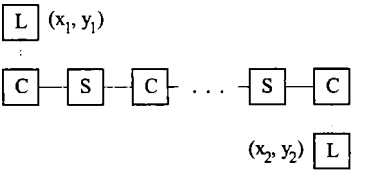


Figura 3.4. O conexiune tipică între două blocuri logice.

În acest caz,

$$P(R_{C_i}|L_{n+1}) = P(X_1 \cap S_1 \cap S_2 ... \cap S_n \cap X_2) = \\ = P(X_1)P(S_1|X_1)P(S_2|S_1 \cap X_1)..P(X_2|S_n \cap ... \cap S_1 \cap X_1) \quad (3.4)$$

În relațiile de mai sus, am păstrat convenția de notare a evenimentelor condiționate separându-le cu o bară verticală. Deoarece evenimentele X_1, S_1, S_2, ...S_n, X_2 nu sunt independente trebuie deduse formule pentru fiecare termen din relația de mai sus. Pentru aceasta trebuie luați în considerare principalii parametri ce descriu geometria FPGA:

- Fc - flexibilitatea blocului C;
- Fs - flexibilitatea blocului S;
- W - numărul de piste dintr-un canal;
- D - densitatea pistelor în canalul de rutare.

Aşa cum am demonstrat deja anterior, densitatea unui canal de rutare este aproximată conform distribuției Poisson, caracterizată de parametrul $\lambda_g = \frac{\lambda \overline{R}}{2}$, unde λ este numărul de conexiuni ale unui bloc logic şi $\overline{R}$ lungimea medie a unei conexiuni.

3.4.2. PROBABILITATEA EVENIMENTULUI X_1

Blocul logic din figura 3.5 este adiacent unui canal de rutare cu W piste. Un pin se poate conecta prin switch-uri de rutare la F_c piste. Un număr de D piste sunt deja rutate de conexiuni anterioare. Evenimentul X1 poate fi privit ca un proces aleator în care un pin al blocului logic se poate conecta la una din pistele libere, dacă există şi un switch corespunzător. Se respectă în continuare presupunerea demonstrată anterior că D este distribuit conform legii de repartiție Poisson şi că legea de repartiție Poisson este infinit divizibilă. Probabilitatea evenimentului contrar lui X_1 se supune de asemenea legii Poisson, dar are media $\frac{\lambda_g F_c}{W}$. Rezultă pentru probabilitatea evenimentului X_1 expresia:

$$P(X_1) = 1 - p\left(\frac{\lambda_g F_c}{W}, F_c\right). \quad (3.5)$$

Efectul aproximației din ecuația de mai sus poate fi neglijată (distribuția Poisson este infinită), deoarece se limitează doar pentru valorile $F_c \leq W$.

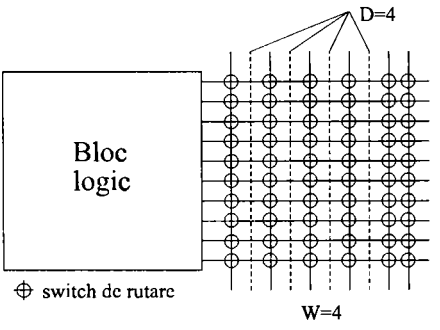


Figura 3.5. Model de calcul pentru evenimentul X_1.

O altă modalitate de calcul a probabilității evenimentului X_1 este mai directă: se definesc evenimentele $X_1\_A_1$, $X_1\_A_2$, ..., $X_1\_A_{Fc}$ pentru care X_1 apare la exact 1, 2 ..., F_c piste. Aceasta înseamnă că:

$X_1 = X_1\_A_1 \cup X_1\_A_2 \cup ... \cup X_1\_A_{F_c}$ și deoarece evenimentele $X_1\_A_1$, $X_1\_A_2$, ..., $X_1\_A_{Fc}$ sunt independente, pentru probabilitatea evenimentului X_1 rezultă relația:

$$P(X_1) = P(X_1\_A_1) + P(X_1\_A_2) + ... + P(X_1\_A_{F_c}). \tag{3.6}$$

În cazul general, în care X_1 apare cu exact *a piste*, evenimentul asociat este $X_1\_A_a$. În acest caz, $D = F_c - a$ piste sunt deja ocupate, iar:

$$P(X_1\_A_a) = p\left(\frac{\lambda_g F_c}{W}, F_c - a\right). \tag{3.7}$$

Se observă că ecuația (3.5) este un caz particular al ecuației de mai sus pentru $a = 0$. În final se obține:

$$P(X_1) = \sum_{a=1}^{F_c} P\left(\frac{\lambda_g F_c}{W}, F_c - a\right). \tag{3.8}$$

3.4.3. PROBABILITATEA EVENIMENTULUI S_1

Pentru ușurința demonstrației se va considera pentru început cazul $F_s = 3$. În acest caz, oricărui segment ce intră într-un bloc S îi corespunde câte un segment pe fiecare latură a blocului S. Acesta este cazul cel mai simplu de studiat, dar el poate fi extins ulterior la valori mai mari ale lui F_s.

Și în acest caz un model pentru evenimentul $S_1|X_1$ este ilustrat în figura 3.6. Segmentele de legătură X_1_A sunt desenate îngroșat la intrarea blocului S, iar cele D segmente ocupate de la ieșirea sa sunt reprezentate punctat. Și în acest exemplu am considerat $W = 10$. Cazul $X_1\_A = 3$ corespunde evenimentului $X_1\_A_3$, definit în subcapitolul anterior. Switch-urile din interio-

rul blocului S sunt reprezentate punctat şi aici au semnificaţia că orice segment de intrare a blocului poate fi conectat cu orice segment de ieşire. În acest context, acest eveniment poate fi considerat ca un eveniment aleator în care oricare dintre cele X_1_A piste de intrare se pot conecta la oricare dintre cele $D-W$ piste de ieşire.

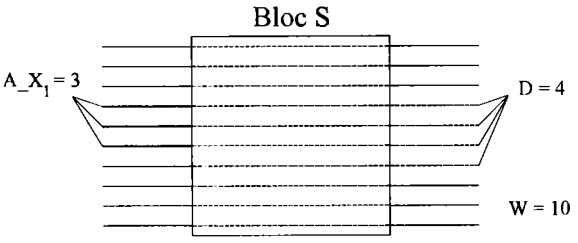


Figura 3.6. Model de calcul pentru evenimentul $S_1|X_1$.

Evenimentul $S_1|X_1$ poate interveni la una sau mai multe piste de ieşire. Se definesc în continuare evenimentele $S_1\_A_1$, $S_1\_A_2$, ..., $S_1\_A_{Fc}$ ca fiind evenimentele pentru care $S_1|X_1$ se produce cu exact 1, 2 ..., F_c piste de ieşire. Deoarece:

$$S_1|X_1 = S_1\_A_1 \cup S_1\_A_2 \cup ... \cup S_1\_A_{F_c},$$

iar evenimentele $S_1\_A_1$, $S_1\_A_2$, ..., $S_1\_A_{Fc}$ sunt independente, se poate scrie:

$$P(S_1|X_1) = P(S_1\_A_1) + P(S_1\_A_2) + ... + P(S_1\_A_{F_c}). \tag{3.9}$$

În cazul general, evenimentul $S_1|X_1$ apare cu exact k piste disponibile la ieşire. Evenimentul corespunzător este $S_1\_A_k$. Probabilitatea de realizare a lui $S_1\_A_k$ depinde de numărul de piste de intrare, date de X_1_A şi de valoarea lui D. Fie X_1_A = a. Deoarece X_1 s-a produs, este evident că X_1 a apărut cu exact a piste disponibile. Evenimentul corespunzător acestei presupuneri se scrie $X_1\_A_a|X_1$. Dacă sunt disponibile k piste, atunci k - a piste sunt deja utilizate şi deci $D = k - a$. Pentru calculul probabilităţii $P(S_1\_A_k|(X_1\ A_a|X_1))$ se va utiliza deja metoda consacrată în subcapitolele anterioare. Se presupune că D respectă legea de distribuţie Poisson. În acest caz, singurele piste de ieşire ce prezintă interes sunt cele *a piste care* pot fi atinse de cele a piste de pe partea de intrare. În consecinţă, noua distribuţie Poisson aferentă acestui caz va avea media $\frac{\lambda_g a}{W}$, şi deci:

$$P(S_1\_A_k|(X_1\ A_a|X_1)) = p\left(\frac{\lambda_g a}{W}, a-k\right). \tag{3.10}$$

În continuare se vor considera pentru toate valorile posibile ale lui X_1_A evenimentele $X_1\_A_1|X_1$,..., $X_1\_A_{F_c}|X_1$. Deoarece îndeplinirea $S_1\_A_k$ implică exact unul dintre $X_1\_A_1|X_1$,..., $X_1\_A_{F_c}|X_1$, atunci:

$$S_1\_A_k = (S_1\_A_k \cap (X_1\_A_1|X_1)) \cup \ldots \cup (S_1\_A_k \cap (X_1\_A_{F_c}|X_1)),$$

şi deoarece evenimentele $X_1\_A_1|X_1$,...., $X_1\_A_{F_c}|X_1$ sunt independente, rezultă:

$$P(S_1\_A_k) = P(S_1\_A_k \cap (X_1\_A_1|X_1)) + \ldots + P(S_1\_A_k \cap (X_1\_A_{F_c}|X_1)) \quad (3.11)$$

Este cunoscut că $P(X \cap Y) = P(X)P(Y|X)$, iar înlocuind în relația anterioară rezultă:

$$\begin{aligned} P(S_1\_A_k) = {} & P(X_1\_A_1|X_1)P(S_1\_A_k|(X_1\_A_1|X_1)) + \ldots \\ & + P(X_1\_A_{F_c}|X_1)P(S_1\_A_k|(X_1\_A_{F_c}|X_1)) \end{aligned} \quad (3.12)$$

Termenii $P(S_1\_A_k|(X_1\_A_a|X_1))$ sunt dați de relația (3.10), așa că:

$$P(S_1\_A_k) = \sum_{a=1}^{F_c} P(X_1\_A_a|X_1)\, p\left(\frac{\lambda_k a}{W}, a-k\right). \quad (3.13)$$

După cum s-a menționat anterior, termenii $P(X_1\_A_a|X_1)$ exprimă probabilitatea ca, dată fiind apariția evenimentului X_1, X_1 a apărut la exact a piste disponibile. Fiecare dintre termenii $P(X_1\_A_a|X_1)$ se poate exprima [Şabac65] utilizând formula lui Bayes:

$$P(X_1\_A_a|X_1) = \frac{P(X_1\_A_a)}{\sum_{j=1}^{F_c} P(X_1\_A_j)}, \quad (3.14)$$

unde $P(X_1\_A_1)$,... , $P(X_1\_A_{Fc})$ sunt date de ecuația 3.8. Substituind (3.13) şi (3.14) în (3.9), rezultă:

$$P(S_1|X_1) = \sum_{k=1}^{F_c} P(S_1\_A_k) = \sum_{k=1}^{F_c}\sum_{a=1}^{F_c} \frac{P(X_1\_A_a)}{\sum_{j=1}^{F_c} P(X_1\_A_j)} p\left(\frac{\lambda_k a}{W}, a-k\right) \quad (3.15)$$

3.4.4. GENERALIZARE PENTRU CAZUL $F_s > 3$

Ecuația anterioară presupune o valoare fixă pentru Fs (Fs=3). Se va generaliza în continuare rezultatul oferit de relația (3.15) pentru cazul general, în care Fs nu este 3. În ecuația (3.10), luând de exemplu cazul Fs=6, parametrul a trebuie înlocuit cu 2a. Analog, în cazul general, ecuația (3.10) devine:

$$P(S_1\_A_k|(X_1\_A_a|X_1)) = p\left(\frac{\lambda_g \alpha a}{W}, \alpha a - k\right), \tag{3.16}$$

în care α este un factor de scalare. Evident, α depinde de *Fs*, dar este influenţat şi de cazul în care într-un bloc *S* o conexiune îşi păstrează direcţia sau efectuează un unghi de 90°. Fie Z_1 evenimentul în care conexiunea nu-şi schimbă sensul în blocul *S* şi Z_2 evenimentul contrar (evident P(*Z*2) = 1 - P(Z_1)) şi α_1 şi α_2 coeficienţii corespunzători de scalare. Atunci, $P(S_1|X_1) = P(Z_1)P((S_1|X_1)|Z_1) + P(Z_2)P((S_1|X_1)|Z_2)$ şi utilizând relaţiile (3.15) şi (3.16) se poate scrie:

$$P(S_1|X_1) = P(Z_1)\sum_{k=1}^{W}\sum_{a=1}^{F_c}\frac{P(X_1\_A_a)}{\sum_{j=1}^{F_c}P(X_1\_A_j)}p\left(\frac{\lambda_g \alpha_1 a}{W}, \alpha_1 a - k\right) +$$

$$+ P(Z_2)\sum_{k=1}^{W}\sum_{a=1}^{F_c}\frac{P(X_1\_A_a)}{\sum_{j=1}^{F_c}P(X_1\_A_j)}p\left(\frac{\lambda_g \alpha_2 a}{W}, \alpha_2 a - k\right). \tag{3.17}$$

Schimbarea limitei de însumare în ecuaţia 3.17 de la *k* la *W* (faţă de 3.15) este necesară deoarece poate fi posibil a se conecta la toate cele W piste dintr-un canal pentru valori *Fs*>3.

3.4.5. PROBABILITATEA EVENIMENTELOR S_i

Până în prezent am tratat doar evenimentul $S_1|X_1$. Rezultatele obţinute pot fi însă extrapolate la un caz general, pentru evenimentul S_m $(S_m|S_{m-1} \cap ... \cap S_1 \cap X_1)$. Ambele sume în acest caz vor avea limita *W*.

Probabilităţile $P(X_1\_A_1)$,... , $P(X_1\_A_{Fc})$ din ecuaţia (3.17) se vor înlocui cu $P(S_{m-1}\_A_1)$,... , $P(S_{m-1}\_A_W)$, rezultând:

$$P(S_m\_A_k) = \sum_{a=1}^{W}P(S_{m-1}\_A_a|S_{m-1}\cap ... \cap S_1 \cap X_1)\,p\left(\frac{\lambda_g a}{W}, a - k\right) \tag{3.18}$$

Cu aceasta, ecuaţia 3.17 devine:

$$P(S_m|S_{m-1}\cap ... \cap S_1 \cap X_1) = P(Z_1)\sum_{k=1}^{W}\sum_{a=1}^{W}\frac{P(S_{m-1}\_A_a)}{\sum_{j=1}^{F_c}P(S_{m-1}\_A_j)}p\left(\frac{\lambda_g \alpha_1 a}{W}, \alpha_1 a - k\right) +$$

$$+ P(Z_2)\sum_{k=1}^{W}\sum_{a=1}^{W}\frac{P(S_{m-1}\_A_a)}{\sum_{j=1}^{F_c}P(S_{m-1}\_A_j)}p\left(\frac{\lambda_g \alpha_2 a}{W}, \alpha_2 a - k\right). \tag{3.19}$$

3.4.6. PROBABILITATEA EVENIMENTULUI X_2

Modelul utilizat pentru calculul probabilității de apariție a evenimentului X_2 este prezentat în figura 3.7. La ieşirea blocului C sunt disponibile $S_n\_A = 4$ piste (ceea ce corespunde cu evenimentul $S_n\_A_4$ din subcapitolul anterior). Evenimentul X_2 poate fi privit ca un proces aleator în care pinul blocului logic poate fi conectat la oricare dintre cele S_n_A piste unde se găsesc switch-uri. Se consideră evenimentul contrar lui X_2, notat NX_2.

$$P(NX_2|S_n \cap ...S_1 \cap X_1) = 1 - P(X_2|S_n \cap ...S_1 \cap X_1).$$

Pentru calculul lui $P(NX_2|S_n \cap ...S_1 \cap X_1)$ se alege o valoare specifică a pentru S_n_A, putându-se calcula în acest caz probabilitatea condiționată

$$P((NX_2|S_n \cap ... \cap S_1 \cap X_1)|S_n\_A_a) = \frac{C_{W-F_c}^{a}}{C_W^a}. \qquad (3.20)$$

membrul drept al ecuației anterioare exprimă raportul dintre numărul de moduri în care *toate* cele a piste pot nimeri în cadrul celor W-F_c piste, astfel încât conexiunea să nu poată fi realizată şi numărul total de modalități în care a piste se pot distribui de-a lungul a W piste.

În ecuația (3.20) se presupune că fiecare din cele F_c switch-uri pot fi distribuite pe oricare din cele W piste, ceea ce este desigur pesimist în raport cu realitatea. Un bloc C cu topologia avansată asigură că toate pistele conectabile la un pin al blocului logic acoperă alți pini conectați la blocul logic sau la alte blocuri logice.

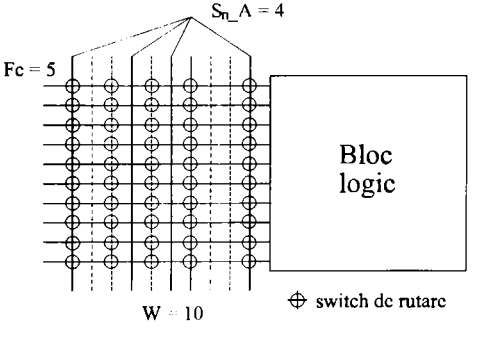


Figura 3.7. Model de calcul pentru evenimentul X_2.

Considerând evenimentele $S_n\_A_1$, $S_n\_A_2$,..., $S_n\_A_W$, se poate scrie:

$$P(NX_2|S_n \cap ... \cap S_1 \cap X_1) = \\ = \sum_{a=1}^{W} P(S_n\_A_a|S_n \cap ... \cap S_1 \cap X_1) P((NX_2|S_n \cap ... \cap S_1 \cap X_1)|S_n\_A_a),$$

în care fiecare $P(S_n \_ A_a | S_n \cap ... \cap S_1 \cap X_1)$ este dat de formula lui Bayes, şi deci:

$$P(X_2|S_n \cap ... \cap S_1 \cap X_1) = 1 - \sum_{a=1}^{W} \frac{P(S_n \_ A_a)}{\sum_{j} P(S_n \_ A_j)} \cdot \frac{C_{W-F_c}^{a}}{C_W^a} \qquad (3.21)$$

3.4.7. PROBABILITATEA EVENIMENTULUI RC_i

În acest moment, relația (3.1) poate fi evaluată pentru valori date ale LC_i=n+1. Se poate scrie deci:

$$P(R_{C_i}|L_{n+1}) = P(X_1 \cap S_1 \cap S_2 \cap ... \cap S_n \cap X_2) = \qquad (3.22)$$
$$= P(X_1)P(S_1|X_1)P(S_2|S_1 \cap X_1) \cdots P(S_n|S_{n-1} \cap ... \cap S_1 \cap X_1)P(X_2|S_n \cap ... \cap S_1 \cap X_1)$$

Considerând $LC_i = l_{IMax}$ lungimea maximă a oricărei conexiuni şi L_{lMax} evenimentul asociat. Fie evenimentele $L_1,...L_{lMax}$ corespunzătoare valorilor posibile ale lui LC_i. Deoarece apariția unui eveniment RC_i implică exact unul dintre $L_1,..., L_{lMax}$, rezultă:

$$P(R_{C_i}) = \sum_{l=0}^{l_{Max}} P(L_l) \cdot P(R_{C_i}|L_l) \qquad (3.23)$$

în care $P(L_l)$ este legea de distribuție, iar fiecare factor $P(R_{C_i}|L_l)$ se calculează cu relația (3.22). După cum am menționat anterior, legea de distribuție este presupusă a fi geometrică, cu media $\overline{R}$. În aceste condiții,

$$P(L_l) = pq^{l-1} = \frac{1}{R}\left(1 - \frac{1}{R}\right)^{l-1} \qquad (3.24)$$

3.5. REZULTATE EXPERIMENTALE

Parametrii de rutare F_c şi F_s au un efect predominant asupra performanțelor obținute. Studiind exemplele concrete introduse în capitolul 4 şi prezentate în detaliu în Anexa 2, rezultă că cea mai bună performanță de rutare se obține pentru $F_c = W$, fapt demonstrat şi în tabelul 3.1.

În figura 3.8 este prezentată comparația între o implementarea dintr-un FPGA real [Brown92] şi predicția ce rezultă din aplicarea rezultatelor lui Gamal. După cum se poate observa din figură, cazul real se apropie extrem de mult de distribuția Poisson ideală.

Relația (3.25) definește eroarea relativă ε_r a predicției de rutare în funcție de rutabilitatea teoretică R_t şi cea practică R_p. În figura 3.9 este ilustrată această dependență, cu F_c ca parametru:

$$\varepsilon_r = \frac{R_p - R_t}{R_t} \cdot 100 . \qquad (3.25)$$

Tabelul 3.1

Numărul minim W de piste într-un canal pentru rutare completă

Circuit	Densitatea canalului	*W* minim necesar pentru rutare completă
Dds1	9	10
Snfag	10	10
Dds	11	12
Dds2	11	13

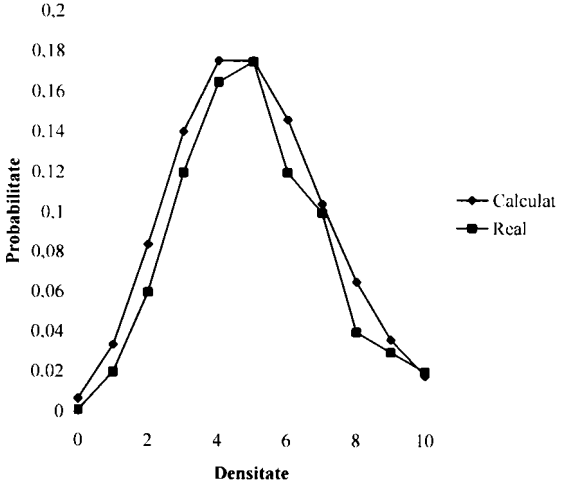


Figura 3.7. Densitatea canalelor în FPGA - predicţie şi realitate.

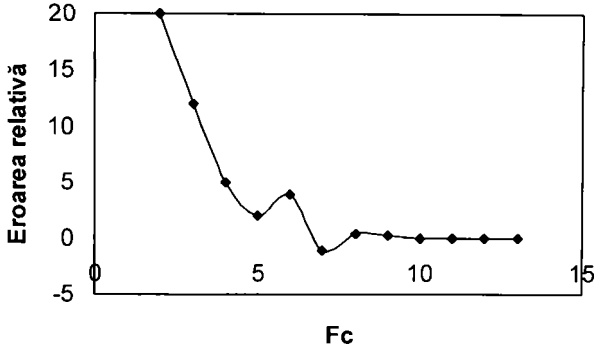


Figura 3.8. Eroarea relativă a predicţiei de rutare.

Studiind graficul din figura 3.8 rezultă o concluzie importantă: cu o unică excepție, rutabilitatea prezisă teoretic este mai pesimistă decât cea obținută practic. Acest rezultat este cu atât mai important cu cât este cunoscut faptul că prin rulări repetate sau redefinirea unor parametri se pot îmbunătăți (în jurul a maxim 10% după experiența autorului) rezultatele de rutare oferite de software-ul respectiv.

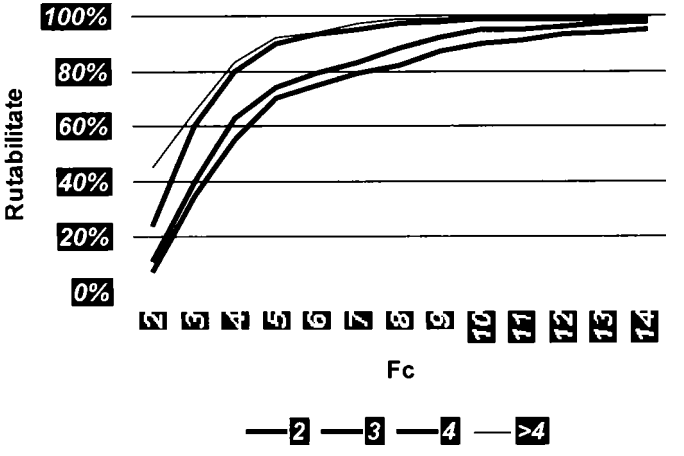


Figura 3.9. Rutabilitatea în funcție de *Fc*.

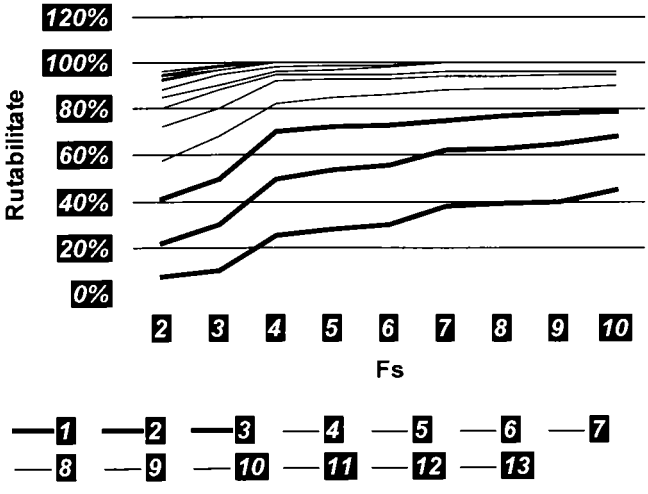


Figura 3.10. Rutabilitatea în funcție de *Fs*.

Graficul din figura 3.8 a fost realizat pentru $F_s = 6$ (motivarea acestei alegeri va fi dată în continuare). Este interesant de studiat cum se modifică rutabilitatea pentru diferite valori

ale lui F_s. Acest lucru este realizat în figura 3.9 pentru F_s = 2, 3, 4 şi > 4. Şi în acest caz se pot desprinde câteva concluzii interesante:

- pentru cazurile studiate, rutabilitate 100% se obține pentru F_c>=10. Acest rezultat este consistent cu o concluzie anterioară, care afirma că performanțe optime de rutare se obțin pentru F_c = 0,6...1 .W;
- mărirea valorii lui F_s peste 4 nu aduce nici o îmbunătățire majoră din punctul de vedere al rutării. Acesta este şi motivul pentru care am ales F_s = 6 în figura 3.8.

Este natural a studia acum dependența rutabilității de F_s, cu F_c luat ca şi parametru. Această dependență este ilustrată în figura 3.10 şi menține concluziile enunțate anterior cu privire la valorile lui F_s. Apare astfel idea unei reprezentări sintetice, care să țină cont *simultan* de variația ambilor parametri de rutare (F_c şi F_s). O asemenea reprezentare este propusă în figura 3.11.

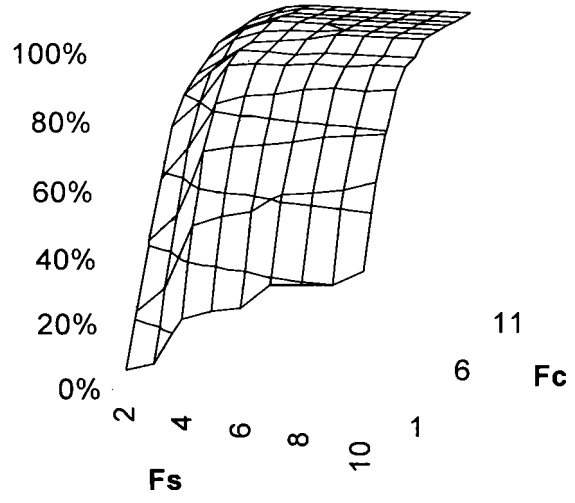


Figura 3.11. Reprezentarea tridimensională a rutabilității.

3.6. CONCLUZII

Studiul consecvent al modelului de rutare introdus în acest capitol permite elaborarea a câtorva concluzii importante. Ca un preambul la enumerarea acestor concluzii, trebuie menționate două argumente în favoarea acreditării ideii că o arhitectură ideală de FPGA nu a fost încă descoperită:

- numărul mare de articole apărute pe tema rutării FPGA şi comparațiile efectuate de autori între metode demonstrează că rutarea în special şi arhitectura FPGA în general nu au fost încă deplin rezolvate.

- pe piață nu s-a impus clar o anumită arhitectură în favoarea alteia (altora). Deși Xilinx ocupă o parte importantă a acestui segment, și alți competitori sunt prezenți, deci domeniul FPGA este în continuă dezvoltare arhitecturală și tehnologică.

În preambulul acestui capitol au fost prezentate un set coerent de definiții pentru principalele mărimi ce caracterizează procesul de rutare. Pe parcursul demonstrațiilor efectuate, se degajă idea prezicerii rutabilității *înaintea* inițierii procesului de rutare propriu-zis.

Cea mai importantă concluzie (și contribuție originală) este verificarea într-un cadru propriu a respectării distribuției Poisson pentru densitatea canalelor de rutare. Această confirmare a făcut posibilă modelarea structurală propusă în acest capitol.

În sprijinul acestei ipoteze, autorul a prezentat și o demonstrație fizică, intuitivă prin care a arătat că evenimentul ca o pistă dintr-un canal să fie ocupată este *rar*, prin aceasta validând suplimentar presupunerea inițială.

Introducerea unei etape intermediare, înaintea rutării propriu-zise este aparent o complicație suplimentară. Această abordare ieftinește însă evident proiectarea prin eliminarea unor încercări repetate de rutare în circuite acolo unde acest lucru este imposibil. În plus, datorită caracterului pesimist al prezicerii date de relația 3.1, este o măsură a gradului în care merită de continuat efortul de plasare și rutare a unui proiect într-un FPGA dat, pentru care acest lucru nu a fost reușit din primele încercări. În acest caz economia provine din alegerea unui FPGA mai puțin complex (și deci mai ieftin) pentru un proiect dat.

Studiul rutării din acest capitol se concentrează asupra influenței parametrilor Fc și Fs asupra performanțelor obținute. Prezentarea dependenței rutabilității de aceşti parametri și mai ales ilustrarea tridimensională a acestei dependențe reprezintă un instrument deosebit de util pentru proiectant.

Studiul erorii relative de rutabilitate introduse de autor (formula 3.25) demonstrează faptul că modelul structural este unul pesimist (cu alte cuvinte rutabilitatea prezisă poate fi sigur atinsă). Acesta este de asemena un rezultat important, căci permite inițierea unui studiu al rutabilității aprioric rutării propriu-zise (rutare pentru care sunt necesare, așa după cum este cunoscut, importante resurse de calcul).

3.6.1. DIRECȚII VIITOARE DE STUDIU

Modelul arhitectural introdus în acest capitol prezintă câteva limitări ce au simplificat considerabil studiul. Eliminând secvențial aceste constrângeri se poate continua acest studiu pentru alte arhitecturi, generalizând rezultatele obținute. Concret, cele două direcții principale de cercetare în viitor sunt:

1. considerarea unei arhitecturi rectangulare M x N (renunțarea la limitarea arhitecturii pătrate N x N. Această direcție este aplicabilă în primul rând pentru FPGA produse de firma Actel;
2. pentru FPGA cu ratarea bazată pe canale segmentate, considerarea segmentelor de lungimi diferite pentru efectuarea conexiunilor. Această direcție este potrivită pentru toate FPGA moderne, la care tendința este de a utiliza resurse de rutare bazate pe segmente cu lungimi diferite.

3.7. REFERINŢE BIBLIOGRAFICE

Pentru redactarea acestui capitol am folosit pe lângă literatura inginerească dedicată şi câteva cărţi de matematică. Între acestea, [Şabac65] este o lucrare excelentă, cunoscută inginerilor timişoreni. Am folosit de aici capitolul de probabilităţi şi am găsit foarte utilă formula lui Bayes, dată la pagina 565.

În ultimii ani au apărut tot mai multe articole dedicate rutării. Lucrare de referinţă [Gamal82], este citată în majoritatea articolelor moderne despre rutare şi introduce pentru prima dată idea unui model matematic de rutare pentru circuitele VLSI. [Brown92] este o altă lucrare de excepţie ce tratează subiectul FPGA şi are un capitol dedicat unui model stohastic de rutare în FPGA.

O lucrare recentă din domeniul rutării - [Lienig97] - este extrem de interesantă, introducând ca principal element de noutate reducerea diafoniei apărute între segmentele lungi paralele. Datorită cantităţii mari de calcule implicate, algoritmul propus rulează paralel pe o reţea de calculatoare.

Dintre lucrările autorului în domeniu, cea mai reprezentativă este [Gontean98a], susţinută la prestigioasa conferinţa internaţională PDS'98 în Polonia. Aici este introdus modelul de structură redat în figura 3.3 şi demonstraţia distribuţiei Poisson pentru densitatea pistelor din canalul de rutare.

CAPITOLUL 4

SINTEZA NUMERICĂ DE FRECVENȚĂ CU STRUCTURI LOGICE PROGRAMABILE

4.0. INTRODUCERE

În capitolul anterior am prezentat o modalitate de a estima probabilitatea rutării unui proiect într-un FPGA simetric. Pentru a valida rezultatele deduse, am considerat în acest capitol mai multe variante de implementare a unui exemplu concret, respectiv a unui sintetizor numeric de frecvență. Un prim motiv pentru această opțiune este complexitatea suficient de mare a implementării, suficientă pentru a putea aplica cu succes statistica matematică. Un al doilea motiv este modernitatea subiectului ales, aspect demonstrat de multitudinea de referințe bibliografice actuale referitoare la sinteza și sintetizoarele numerice de frecvență.

În prezent se utilizează mai multe metode de sinteză de frecvență:

1. *Metoda analogică directă*, în care frecvența de ieșire se obține din frecvența de referință prin multiplicare, divizare, mixare și filtrare;
2. *Sinteza indirectă* sau metoda PLL, în care frecvența de ieșire se obține dintr-un oscilator secundar (de obicei VCO), calat în fază (sau mai rar în frecvență) cu oscilatorul de referință;
3. *Sinteza numerică directă* (SNF), în care frecvența de ieșire se obține direct din frecvența de referință.

Principalele criterii avute în vedere la compararea diferitelor variante de implementare trebuie să includă:

1. Raportul semnal/zgomot;
2. Domeniul de frecvență posibil de obținut;
3. Viteza de comutare de la o frecvență de ieșire la alta;
4. Rezoluția frecvenței de ieșire;
5. Posibilitățile de modulare;
6. Continuitatea fazei.

SNF este o tehnică care s-a îmbunătățit spectaculos în ultimii ani. În aceasta metodă, frecvența de ieșire este creată și controlată într-o schemă sincronă și de aceea atât faza cât și amplitudinea sunt cunoscute exact. În plus nu există o derivă în timp sau cu temperatura. În cadrul proiectului nu am considerat influența CAN, aceasta depășind sfera domeniului abordat.

Un alt avantaj al SNF este că semnalul poate fi foarte ușor modulat. În plus forma semnalului de ieșire este stocată într-o memorie (al cărei conținut poate fi simplu modificat), flexibilitatea acestei abordări fiind extremă. Continuitatea de fază se obține implicit, sarcină ce nu se putea îndeplini în celelalte două cazuri decât pentru modificări foarte reduse de frecvență.

Pe durata stagiului efectuat la Universitatea Central Lancashire din Preston, Marea Britania, din anul 1996 am proiectat, implementat şi simulat câteva variante de sintetizoare numerice de frecvență. Acest lucru a fost posibil şi datorită sprijinului obținut de la Dl. Prof. Dr. Phil Holifield, al cărui cont pe rețeaua de stații Sun l-am folosit în diferitele modelări şi simulări efectuate. Toate variantele de SNF propuse sunt originale, fiecare fiind adusă la stadiul de simulare post-rutare.

4.1. FUNCȚIONAREA UNUI SNF

Un SNF se compune în principal dintr-un subansamblu numeric, unde se generează o anumită secvență de numere şi un CNA pentru transformarea acestei secvențe în semnal analogic. Prima parte este compusă dintr-un acumulator, un convertor de cod şi un registru pentru memorarea FSW.

Principiul de funcționare al unui SNF este extrem de simplu, presupunând calculul (în timp real) al unei faze care creşte liniar, obținerea unei serii de valori numerice corespunzătoare formei de variație dorită şi convertirea şi filtrarea acestei serii, rezultând un semnal analogic având parametrii impuşi inițial. SNF utilizează o singură frecvență de referință, care guvernează funcționarea întregii scheme. Implementarea convertorului de cod se poate realiza prin mai multe metode, fiind posibil a genera o gamă extrem de largă de forme de undă periodice.

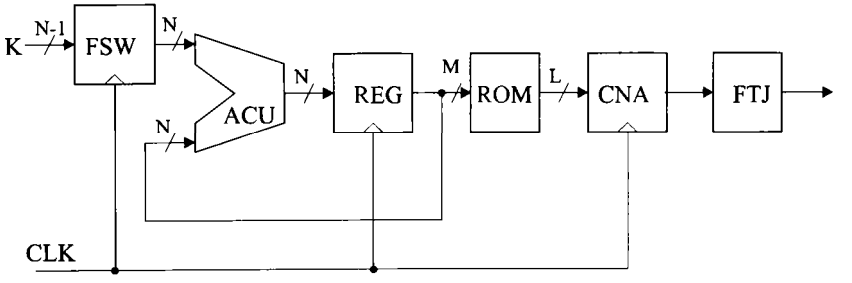


Figura 4.1. Schema de principiu a unui SNF.

În figura 4.1 este prezentată schema de principiu a unui SNF sinusoidal [Tierney71], [Crawford94]. Semnalul ce se obține la ieşire este:

$$x(t) = \sin(2\pi f_{out} t + \phi). \tag{4.1}$$

Notând cu F_{ref} frecvența de referință, $F_{ref} = \frac{1}{T_{ref}}$, la momentele $t = nT_{ref}$, ($n \in \mathrm{N}$) la ieşirea CNA se obțin eşantioane de amplitudine condiționată de personalizarea convertorului de cod. Fiecare eşantion $x(nT_{ref})$ este calculat utilizând o fază $\Phi(n) = 2\pi f_{out} nT_{ref}$ în care $f_{out} = kf_{res}$, unde k este un indice de frecvență. Prin f_{res} s-a notat rezoluția frecvenței la ieşire (şi în acelaşi timp frecvența minimă ce poate fi generată), egală cu $\frac{F_{ref}}{2^N}$. Gama frecvențelor posibile de ieşire este dată de ecuația de acordare a SNF:

$$f_{out} = k\frac{F_{ref}}{2^N} \tag{4.2}$$

Pentru $\phi = 0$ şi $t = nT_{ref}$, relaţia (4.1), devine:

$$x\left(nT_{ref}\right) = \sin\Phi\left(nT_{ref}\right) = \sin 2\pi\left(\frac{kF_{ref}}{2^N}nT_{ref}\right) = \sin\left(\frac{2\pi}{2^N}kn\right) = \sin\left(2\pi\frac{\Theta(n)}{2^N}\right) \tag{4.3}$$

unde $\Theta(n)$ este secvenţa generată de acumulator, fiind egală cu $\langle n\cdot k\rangle$ în care $\langle\cdot\rangle$ reprezintă modulo 2^N. În relaţia de mai sus, n este un indice temporal, iar k de frecvenţă.

În figura 4.2 este prezentat modul de generare a două frecvenţe diferite, $f_{out,min} = f_{res}(k=1)$ şi $f_{out} = 2\cdot f_{res}(k=2)$.

Acumulatorul este format dintr-un sumator pe N biţi urmat de un registru de memorare, organizat de asemenea pe N biţi. În acumulator se adună periodic aceeaşi valoare k (notată uneori şi cu FSW); la depăşire (atunci când suma este mai mare de 2^N-1), MSb se pierde (ceea ce corespunde unei scăderi de 2^N) şi procesul de adunare continuă. Prin această comportare se simulează periodicitatea cu 2π rad a funcţiei sinus.

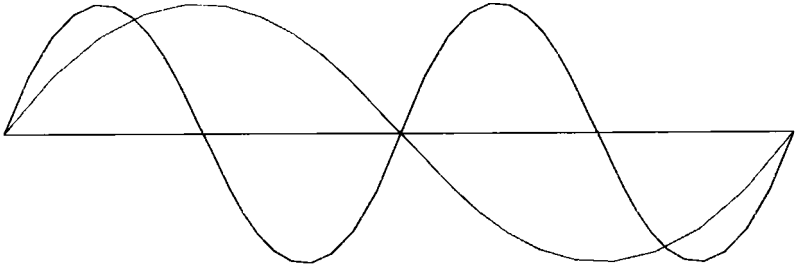


Figura 4.2. Modul de obţinere a frecvenţelor f_{res} şi $2f_{res}$.

Faza $\Theta(n)$ obţinută la ieşirea acumulatorului se poate extrapola la un semnal dinte de fierăstrău continuu, cu panta $\frac{\Delta\Theta(n)}{\Delta t} = k$, amplitudinea 2^N şi perioada $\frac{2^N}{k}$ (figura 4.3). Este evident că şi funcţia sinus generată va avea perioada $\frac{2^N}{k}$.

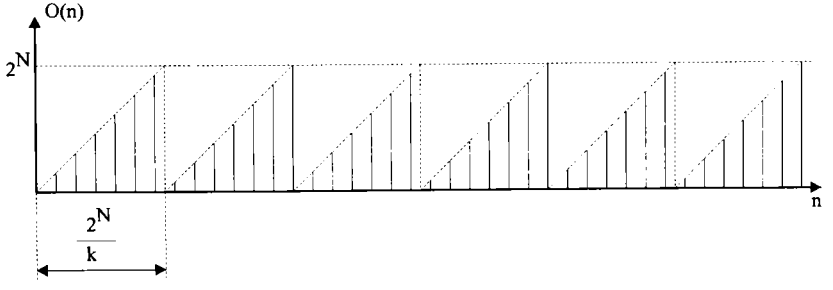


Figura 4.3. Faza la ieşirea acumulatorului.

Secvența $\Theta(n)$ are o perioadă numerică P, definită ca valoarea minimă pentru care $\Theta(n) = \Theta(n+P)$, și este în general egală cu $\frac{2^N}{(k,2^N)}$, unde $(k,2^N)$ reprezintă cel mai mare divizor comun al lui 2^N și k. De aceea P este egală cu perioada semnalului dinte de fierăstrău idealizat numai în cazul particular în care $k = 2^a$, cu $a \in N$.

De exemplu pentru k = 3 și N = 4, perioada semnalului dinte de fierăstrău este 16/3=5.33, iar P = 16. Secvența de la ieșirea convertorului de cod va avea perioada P, iar de aici rezultă o primă consecință importantă, și anume că spectrul semnalului de la intrarea CNA va fi discret, caracterizat de $\frac{2^N}{(k,2^N)}$ puncte [Nicholas87].

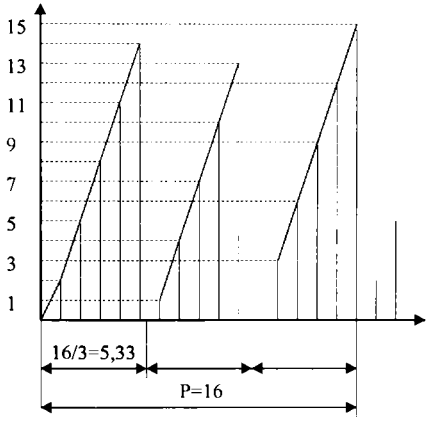


Figura 4.4. Perioada și perioada numerică.

Așa cum s-a menționat deja, sunt posibile mai multe modalități de implementare a convertorului de cod:

1. utilizând un algoritm de calcul, de exemplu o aproximare prin calcul polinomial;
2. folosind un ROM ca LUT (tabel de căutare);
3. dacă este acceptabilă o formă de undă triunghiulară la ieșirea CNA, se poate utiliza un bloc de inversoare.

Prima variantă este cea clasică, dar cerințele tot mai mari de frecvențe ridicate au determinat dezvoltarea celei de-a doua variante, în care calculele sunt minimizate pe seama stocării *directe* a valorilor amplitudinii în ROM. Descreșterea prețului memoriilor ROM a încurajat această dezvoltare. În ultimul timp au fost propuse suplimentar combinații ale celor două metode, determinate atât de necesitatea de frecvență de operare cât mai ridicată, dar și de dorința de a utiliza un ROM de capacitate cât mai mică (și implicit mai rapid). De exemplu, DSP-ul PDSP16350 al firmei Gec-Plessey utilizează un acumulator pe 34 de biți, folosind un algoritm de calcul al amplitudinii.

Cea de-a treia variantă este foarte puțin studiată în literatură și oferă avantajul unei extreme simplități și posibilitatea implementării directe în ASIC sau SLP, cu dezavantajul unui spectru de ieșire mai puțin performant.

În continuare se vor efectua câteva considerații sumare asupra variantei de implementare cu ROM. Rezoluția de frecvență depinde de N, numărul de biți ai acumulatorului, ($f_{res} = \frac{F_{ref}}{2^N}$ și $\phi_{res} = \frac{2\pi}{2^N}$), iar rezoluția amplitudinii depinde de L, numărul de biți de la ieșirea ROM. Dacă *toți* biții acumulatorului sunt dirijați spre ROM, atunci pentru a obține o cuantă fină de frecvență este necesar un ROM de capacitate mare. Acest efect este mai evident pentru rezoluția de fază decât pentru rezoluția de amplitudine, deoarece dimensiunea ROM crește exponențial cu numărul de biți de adresă și liniar cu numărul de biți de la ieșirea sa.

Memoriile ROM de capacitate mare oferă o rezoluție bună de frecvență, dar limitează viteza SNF datorită timpului de acces ridicat. Pe de altă parte este relativ dificil de implementat arii ROM într-un ASIC. De aceea nu toți biții acumulatorului sunt dirijați spre ROM, ci doar un număr M, iar $B = N - M$ biți mai puțin semnificativi se trunchiază. Prin trunchierea fazei se înrăutățește însă puritatea semnalului la ieșirea SNF. În literatură sunt propuse metode suplimentare de reducere a dimensiunii ROM, din care unele vor fi prezentate în continuare.

4.2. SNF ÎMBUNĂTĂȚITE

4.2.1. REDUCEREA DIMENSIUNII MEMORIEI ROM

Cea mai simplă metodă de compresie este de a folosi simetria funcției sinus, memorând doar valorile din primul cadran. Biții MSb și MSb-1 sunt utilizați pentru decodificarea cadranului, ceilalți biți fiind utilizați pentru adresarea unui ROM ce conține valorile funcției sinus din primul cadran. Bitul MSb stabilește semnul funcției, iar bitul MSb-1 prima derivată (semnalul crește sau scade). Dacă amplitudinea trebuie să scadă pentru creșterea fazei, adresa ROM este inversată; pentru un semnal negativ se inversează tot cuvântul de ieșire (figura 4.5).

Dimensiunea ROM poate fi redusă în continuare prin compresia valorilor $\sin\Phi(n)$ pentru $0 < \Phi < \frac{\pi}{2}$, așa cum se propune de exemplu în [Gontean86], [Holtkamp94] și [Pucio94].

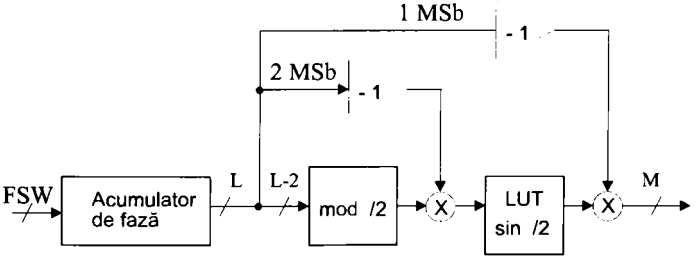


Figura 4.5. ROM SNF în primul cadran.

O primă posibilitate este utilizarea unei ierarhii de două memorii ROM, una pentru valori brute și alta pentru valori fine. De exemplu se poate înlocui un ROM de 2^{A+B+C} cuvinte cu

două ROM de dimensiuni mai reduse, respectiv de dimensiune 2^{A+B} și 2^{A+C}, ale căror ieșiri sunt însumate pentru a reconstrui funcția sinus (figura 4.6).

De fapt, $\sin\Phi$ poate fi scris ca $\sin(\alpha+\beta+\gamma)$ cu $\alpha<\frac{\pi}{2}$, $\beta<\frac{\pi}{2}2^{-A}$ și $\gamma<\frac{\pi}{2}2^{-(A+B)}$. Deoarece $\sin(\alpha+\beta+\gamma)\approx\sin(\alpha+\beta)+\cos\alpha\cdot\sin\gamma$, conținutul primei memorii ROM va fi $\sin(\alpha+\beta)$ iar pentru cel de-al doilea $\cos\alpha\cdot\sin\gamma$. De exemplu, pentru N=12, și 11 biți la ieșire, A=4, B=4 și C=4. Față de un ROM de dimensiune $2^{12}\cdot 11$ se utilizează un ROM $2^8\cdot 11$ și unul $2^8\cdot 4$, adică o economie de $\left(1-\frac{2^8\cdot 11+2^8\cdot 4}{2^{12}\cdot 11}\right)\cdot 100\%=91.5\%$. În cel de-al doilea ROM sunt memorați doar 4 biți pentru fiecare cuvânt, deoarece cei mai semnificativi 7 biți sunt întotdeauna zero. Degradarea raportului semnal/zgomot datorată acestei compresii este uzual mai mică de ±1.5 LSb.

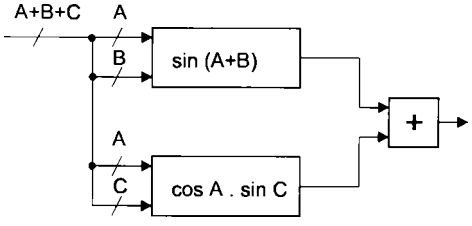


Figura 4.6. Arhitectura Sunderland pentru compresia funcției $\sin\Phi(n)$.

O altă posibilitate, independentă de aproximațiile trigonometrice este propusă în [6], având suplimentar avantajul unui raport semnal zgomot mai bun.

4.2.2. ÎMBUNĂTĂȚIREA RAPORTULUI SEMNAL/ZGOMOT

O primă metodă propusă în [O'Leary91] se poate aplica pentru frecvențele de ieșire mult mai reduse decât frecvența de referință utilizând tehnica "noise shaping". Eroarea datorată trunchierii se poate corecta printr-un proces de acumulare. În final această eroare rezultă ca o eroare de adresare a ROM. În acest fel se interpolează adresele consecutive din ROM. Arhitectura propusă este prezentată în figura 4.7. În această tehnică se presupune că există implicit o supraeșantionare atunci când $f_{out} << F_{ref}$.

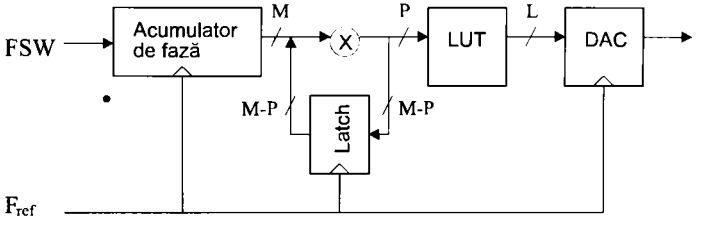


Figura 4.7. Reducerea zgomotului.

O altă abordare este propusă în [Nicholas87]; ea se bazează pe observația că valorile K ale FSW pentru care $(K,2^B)=2^{B-1}$ dau amplitudinea maximă ($B = N - M$ este numărul de biți trunchiați iar $(K,2^B)$ este cel mai mare divizor comun între K și 2^B). De aceea asemenea valori pentru FSW trebuie evitate, de exemplu luând doar valori impare pentru FSW. Prin aceasta se dublează însă rezoluția SNF, chiar cu o îmbunătățire de 3.9 dB în SNR.

Pe de altă parte pentru FSW impar, bitul LSb al acumulatorului oscilează între 0 și 1 la fiecare tact, ca și când acumulatorul ar avea un bit suplimentar.

De aceea frecvența se dublează prin alegerea unui FSW impar și se înjumătățește prin ultima modificare, care se poate realiza simplu cu un bistabil D, ca în figura 4.8. Este de menționat că performanțele de viteză nu sunt modificate deoarece calea critică nu este afectată.

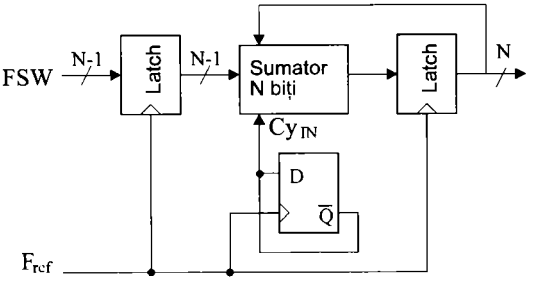


Figura 4.8. Acumulator modificat.

O altă posibilitate este injectarea unui semnal numeric aleator (dither) în SNF, prin aceasta întrerupându-se periodicitatea erorii. Aceasta se poate realiza prin însumarea unei secvențe pseudoaleatoare la intrarea CNA, așa cum este prezentat în figura 4.9. Amplitudinea acestui semnal este de obicei ½ LSb. Hazardul obținut va crește puțin nivelul general de zgomot la ieșire (overall output noise floor).

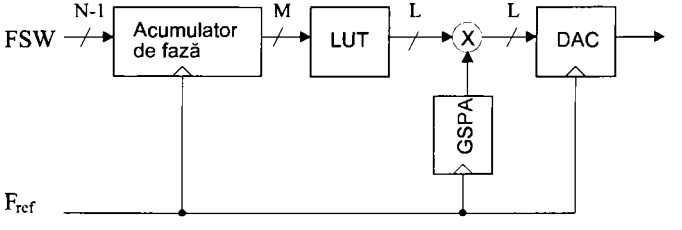


Figura 4.9. Reducerea zgomotului prin utilizarea unui semnal aleator.

4.3. POSIBILITĂȚI DE IMPLEMENTARE A UNUI SNF

La implementarea de tip SLP sau ASIC există o mare libertate de opțiune în privința arhitecturii folosite pentru diferitele blocuri componente. Constrângerile sunt legate de aria de siliciu ocupată, de maparea logicii în siliciu, dar mai ales de plasarea și rutarea conexiunilor.

Un SNF prezintă două componente critice pentru viteză: acumulatorul și memoria ROM. Ambele trebuie să proceseze corect informația în intervalul $T_{ref} = \frac{1}{F_{ref}}$, deoarece la fiecare T_{ref} un nou eșantion este emis către CNA. În tehnologia actuală, ROM-ul este componenta cea mai lentă. Lungimea L a cuvântului de memorie este un parametru important datorită influenței sale asupra erorii de cuantizare. Pe de altă parte o rezoluție ridicată de fază și frecvență conduce la F_{ref} redusă.

4.3.1. SCHEMA BLOC

Cea mai simplă variantă de implementare a convertorului de cod pe care am ales-o folosește inversoare în locul memoriei ROM (figura 4.10). Detalii complete privind schema electrică, simularea și rutarea se găsesc în Anexa 2. Utilizarea inversoarelor conduce și la avantajul reducerii resurselor utilizate deoarece un ROM *intern* este mai dificil de implementat, mai ales la capacități mari. Timpul de propagare t_{inv} pentru inversor este mai redus decât timpul t_{rac} de acces la ROM.

Dezavantajul acestei abordări este descreșterea raportului semnal/zgomot la ieșire datorită prezenței în spectru a armonicilor frecvenței utile. În acest caz componenta critică devine acumulatorul, iar din acesta sumatorul este partea cea mai lentă.

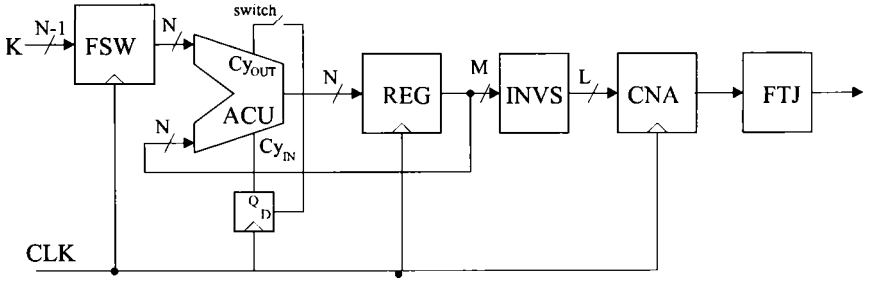


Figura 4.10. SNF cu inversoare - schema bloc (notat dds1 în Anexa 2).

Pe frontul crescător al semnalului de tact, registrul de intrare FSW încarcă un cuvânt pe N-1 biți. După registrul FSW se găsește acumulatorul, care adună în fiecare perioadă de tact cuvântul de control stocat în FSW. Intrarea *switch* permite acumulatorului să numere până la 2^{12}-1 sau 2^{12}. Cei M biți mai semnificativi din acumulator sunt dirijați spre blocul inversor, unde se poate realiza și o eventuală modulare. Cei L biți ce rezultă din inversor sunt memorați în registrul final și aplicați CNA.

4.3.2. STUDIUL ARHITECTURII PENTRU ACUMULATOR

În cazul SNF, acumulatorul este format în principiu dintr-un sumator și un registru de memorare. În acumulator se adună repetat aceeași valoare FSW, cu ignorarea constantă a bitului rezultat în urma depășirii. Așa cum s-a menționat acumulatorul este componenta critică a

SNF, iar sumatorul cea mai lentă parte a sa. Creşterea vitezei de operare depinde de propagarea bitului de transport *carry*.

Din etapa TTL sunt binecunoscute o serie de tehnici de sporire a vitezei acumulatorului, dar pentru 12 biți sau mai mult, aceste metode sunt fie nesatisfăcătoare ca suplimentare a vitezei, fie nu sunt optime din punct de vedere al ariei de siliciu ocupate. Cea mai bună abordare este găsirea unei *arhitecturi globale* pentru acumulator şi nu doar pentru sumator.

Tehnica pipeline de procesare paralel aduce o sporire a performanțelor de viteză, cu dezavantajul unei întârzieri (numită latență - LATENCY şi exprimată în perioade de tact), cu care apare la ieşire rezultatul final. Din momentul în care datele de intrare sunt valide, rezultatul corect apare la ieşire doar atunci când *toate* celulele componente ale structurii au procesat datele corespunzătoare. În acest caz blocul elementar din componența acumulatorului este un sumator pe un bit. Latența s-ar putea îmbunătăți prin divizarea în continuare a acestui sumator pe un bit, ceea ce conduce însă la necesitatea utilizării logicii dinamice şi la o strategie specifică de aplicare a semnalului de tact.

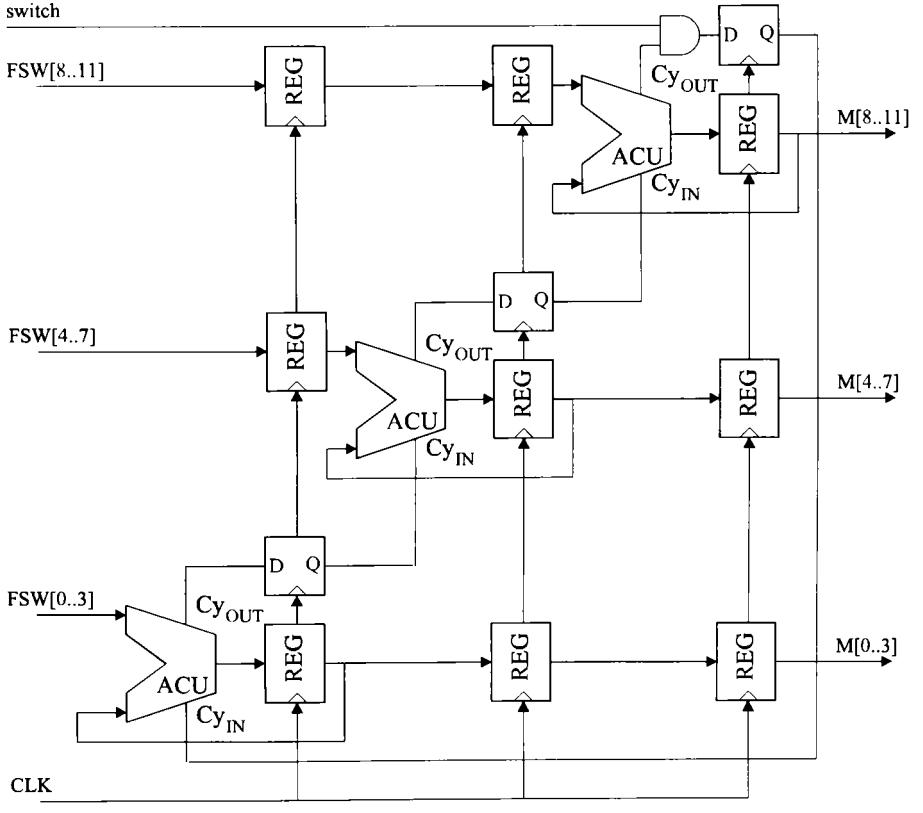


Figura 4.11. Acumulator cu trei etaje (notat dds în Anexa 2).

O primă posibilitate de implementare a acumulatorului este prezentată în figura 4.11. Pentru reducerea ariei de siliciu ocupate, în acest caz au fost implementate trei etaje pe patru

biți, intrările acumulatorului fiind:

- cei N-1 biți pentru FSW;
- CLK;
- switch.

Pentru cazul $N = 12$, cei patru biți mai puțini semnificativi ai FSW se aplică la intrările primului sumator, iar celelalte 4 intrări sunt ieşirile registrului ce urmează primul sumator, (aceştia sunt chiar rezultatul din precedenta adunare). Ceilalți biți ai FSW sunt memorați în partea ce rămâne din primul registru, pentru ca în următoarele cicluri să poată fi adunați în următoarele sumatoare. Aceste două sumatoare funcționează similar cu primul sumator, adunând 4 biți ai registrului anterior cu cei 4 din rezultatul fostei proprii adunări. De asemenea se memorează în registru ieşirea carry a primului sumator. De aceea la următorul semnal de tact carry este livrat spre următorul sumator. Astfel semnalul carry este întârziat pentru a avea alinierea temporală corectă. Dacă cuvântul de control din FSW are 11 biți, la o organizare pe 12 biți a acumulatorului, bitul cel mai semnificativ va fi setat la nivelul 0. Intrarea switch are această funcție, când este JOS (ca un comutator deschis) ieşirea carry a ultimului sumator nu mai este adusă la intrarea carry a primului. În caz contrar această legătură este stabilită. Astfel acumulatorul numără până la 4095 (switch = 1 sau închis) sau până la 4096 (deschis).

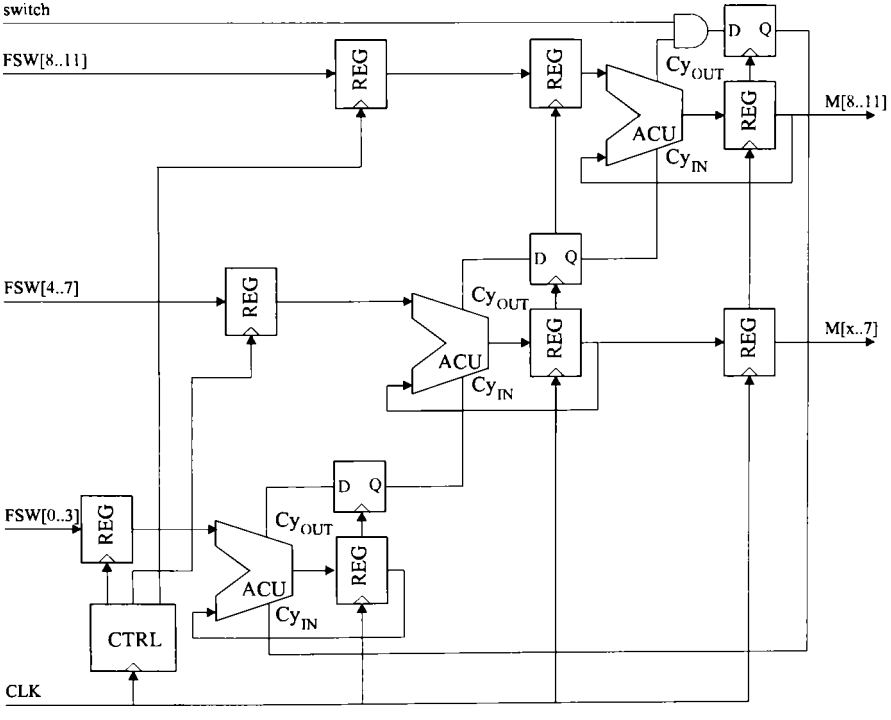


Figura 4.12. Acumulator cu trei etaje, varianta a II-a (notat dds1 în Anexa 2).

Această arhitectură prezintă avantajul că dacă FSW este modificat acumularea continuă, asigurându-se *continuitatea de fază*. Aceasta caracteristică este asigurată de registrele de me-

morare care întârzie biții de date înainte de adunare (7 bistabile din primul registru și 3 din cel de-al doilea).

Cei 4 biți mai semnificativi ai celui de-al doilea registru și cei 8 ai celui de-al treilea sunt întârziați pentru alinierea lor cu biții mai semnificativi. Numărul acestor registre poate fi micșorat ținând cont de faptul că nu toți biții acumulatorului sunt folosiți mai departe. O micșorare suplimentară a ariei ocupate poate fi obținută considerând faptul că în cele două registre de memorare de la intrare datele nu sunt procesate. Acestea doar întârzie ieșirea pentru a avea alinierea corectă atunci când FSW se modifică. Dacă FSW este același de la o perioadă de tact la alta, aceste registre nu mai sunt necesare.

Modificând registrul FSW și anume prin gruparea în trei blocuri a bistabilelor din registru, fiecare cu propriul semnal de tact, datele pot fi obținute prin întârzierea tactului pentru aceste blocuri. Pentru aceasta este necesar un bloc suplimentar de control, care va genera 3 semnale de tact întârziate cu o perioadă de tact unul de altul. Arhitectura pentru acest tip de acumulator este prezentată în figura 4.12.

Cuvântul FSW trebuie să fie stabil la intrarea registrului FSW pentru cel puțin 3 perioade de tact. Avantajele modificărilor propuse sunt mai pronunțate pentru un număr ridicat de etaje pentru acumulator.

A treia schemă propusă (figura 4.13) este un acumulator pe N biți, complet pipeline. Problema acestui tip de acumulator este latența ridicată, care este de N perioade de tact. Un număr mare de bistabile este necesar dacă nu sunt aplicate cele două modificări propuse la schema din figura 4.12.

De exemplu pentru N=12, sunt necesare 12x12+1=145 bistabile (considerând FSW de 11 biți și câte un bistabil pentru memorarea carry în fiecare etaj). Economia de bistabile de memorare este 10+9+...+1=55, iar economia de bistabile de deplasare este de 1+2+3+4+5+6+ ...=56 bistabile. Cu numai 34 de bistabile se poate implementa un acumulator.

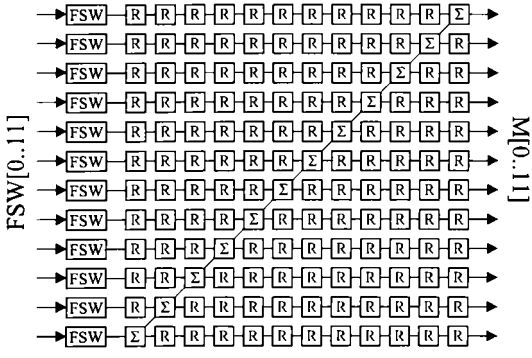


Figura 4.13. Acumulator pipeline, schema bloc.

Cu această schemă circuitul de control trebuie să furnizeze 11 semnale de tact întârziate cu o perioadă unul față de altul. Deși circuitul de control se poate implementa cu un AES cu 4 bistabile și porți, o abordare mai eficientă în cazul SLP este utilizarea unui registru de deplasare cu 11 bistabile D, având atât avantajul vitezei de funcționare mai ridicate, cât și pe cel al

eliminării complete a logicii combinaționale. Ieşirile bistabilelor sunt semnale de tact pentru bistabilele FSW (figura 4.14).

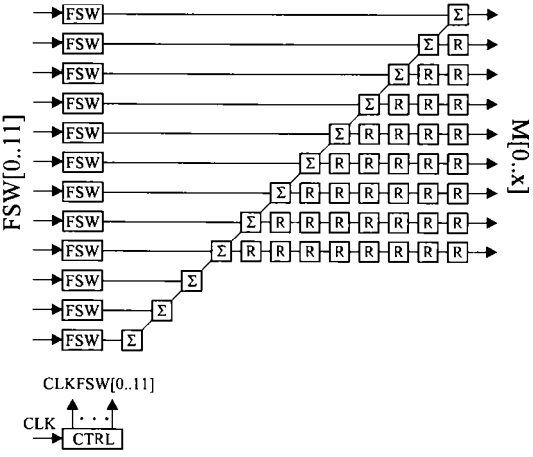


Figura 4.14. Acumulator pipeline modificat, schema bloc (notat dds2 în Anexa 2).

4.3.3. IMPLEMENTAREA ACUMULATORULUI CU FPGA Xilinx

Din studiul SLP disponibile în prezent rezultă că arhitectura familiei XC3000 (şi familia următoare, XC4000) dezvoltată de firma Xilinx este cea mai potrivită pentru implementarea unui acumulator pipeline, care pot funcționa la frecvențe de peste 50 MHz.

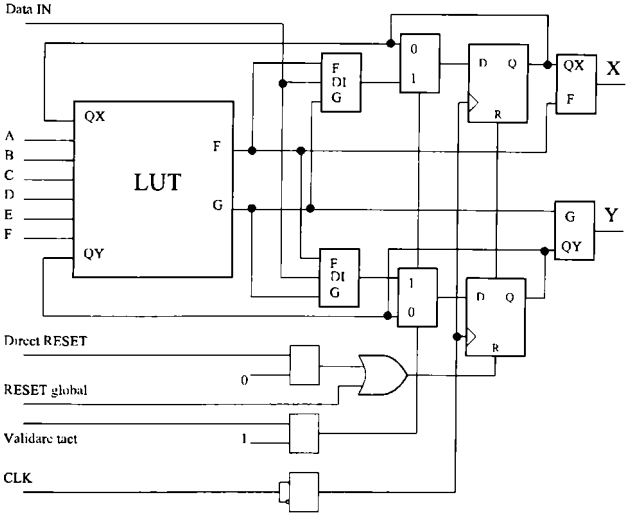


Figura 4.15. Blocul logic în familia Xilinx XC3000.

Blocul logic al familiei XC3000, prezentat în figura 4.14 poate fi configurat pentru a implementa un sumator complet pe un bit, conform figurii 4.15.

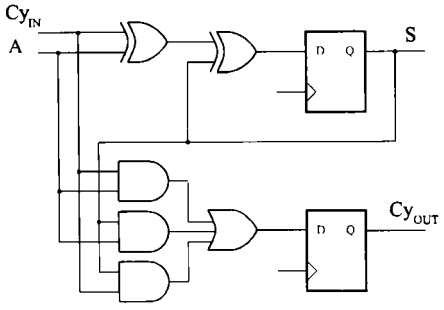


Figura 4.16. Sumator pe un bit într-un CLB Xilinx XC 3000.

4.3.4. SUMATORUL

Un sumator complet este o celulă cu trei intrări (de date a_i și b_i și *carry* c_i), iar ca ieșiri suma s_i și carry c_{i+1}, conform următorului tabel de adevăr.

Tabelul 4.1

Funcționarea sumatorului complet

a	b	c_i	sum	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	1	0	1
0	1	0	1	0
1	1	0	0	1
1	1	1	1	1
1	0	1	0	1
1	0	0	1	0

Tabelul 4.2.

Funcționarea sumatorului propus

a	b	c_i	$\overline{sum}$	$\overline{c_{i+1}}$
0	0	0	1	1
0	0	1	0	1
0	1	1	1	0
0	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	0	1	1	0
1	0	0	0	1

Pentru un sumator standard (ripple carry adder) pe mai mulți biți, sumatorul pe un bit furnizează la ieșire rezultatul corect doar după ce sumatorul anterior a determinat valoarea bitului *carry*. De aceea creșterea numărului de biți duce la mărirea timpului de propagare. Au fost dezvoltate tehnici de a micșora acest timp, tratate foarte bine în literatură[1]. Câteva posibilități sunt prezentate în cele ce urmează:

- *sumatoare cu transport anticipat* (CARRY LOOK AHEAD), biții de carry sunt calculați în paralel; în acest mod celula *i* nu mai este nevoită să aștepte rezultatul furnizat de celula *i*-1.
- *sumatoare cu selecție după carry* (CARRY SELECT ADDERS) sunt divizate în două blocuri - unul calculează suma cu *carry in* = 0, celălalt cu *carry in* =1. Cele

[1]Faptul că acest subiect este atât actual cât și de larg interes este demonstrat și de faptul că IEEE publică și în 1995 articole pe acest domeniu

două ieşiri sunt aduse la intrările unui MUX 2:1 cu *carry* intrarea de selecție, astfel încât suma corectă este selectată.

- *sumatoare* (CARRY SKIP ADDERS) sunt de asemenea împărțite în blocuri. Ideea este de a recunoaşte dacă *carry* se va propaga printr-un bloc şi de a utiliza un multiplexor care să conducă transportul direct la blocul următor. Dacă în cadrul blocului se generează transport, atunci acesta se va transmite blocului următor prin multiplexor.

Este evident că oricare dintre cele trei variante propuse utilizează resurse logice suplimentare față de sumatorul cu transport serie; în plus performanțele de viteză nu se îmbunătățesc efectiv decât pentru un sumator având mai mult de 8 biți, vezi de exemplu [Bruma93].

4.4. SIMULARE ŞI REZULTATE EXPERIMENTALE

Pachetul de programe Solo 1400, furnizat de European Silicon Structures (ES^2) este un software performant, pe care l-am utilizat pe perioada stagiului la Universitatea Central Lancashire, Preston pentru proiectarea şi simularea unui sintetizor numeric de frecvență, în mai multe variante. Etapele parcurse la proiectarea cu Solo respectă algoritmul general din figura 1.5, cu mențiunea că este posibilă şi o simulare post-rutare (post layout simulation) în condiții foarte apropiate de cele reale de funcționare.

Numărul mare de fişiere implicate, complexitatea calculelor, mărimea bibliotecilor şi importanța rezultatelor au determinat implementarea acestui program pe maşini RISC, respectiv pe stații grafice SUN.

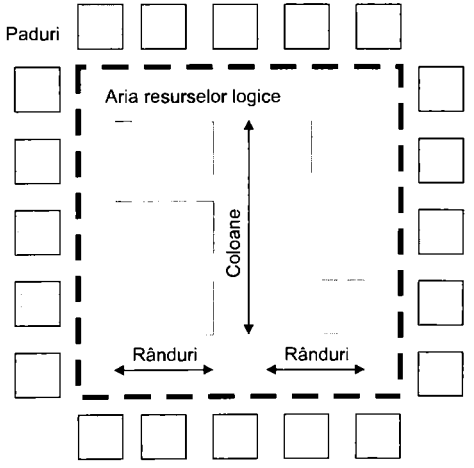


Figura 4.17. Layout-ul pentru un circuit proiectat cu Solo 1400.

Layout-ul prezentat în figura 4.17 este practic identic cu modelul folosit în studiul din capitolul 2, figura 2.4, cu deosebirea că permite suplimentar proiecte cu layout dreptunghiular (unde numărul de coloane este diferit de numărul de rânduri).

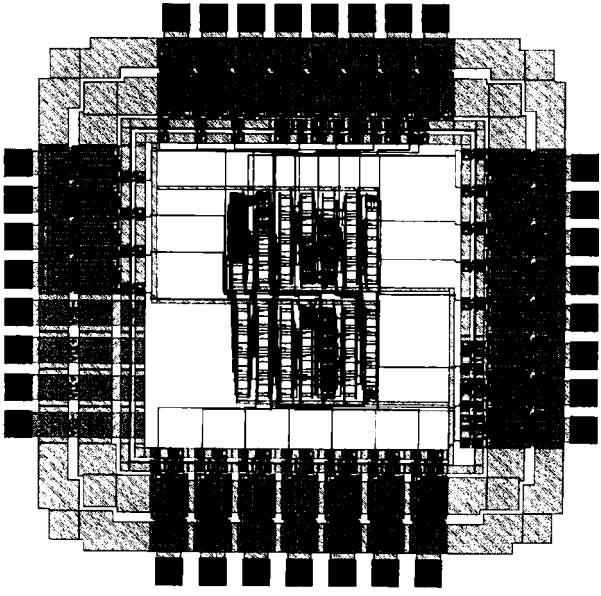


Figura 4.18. Asic dds.

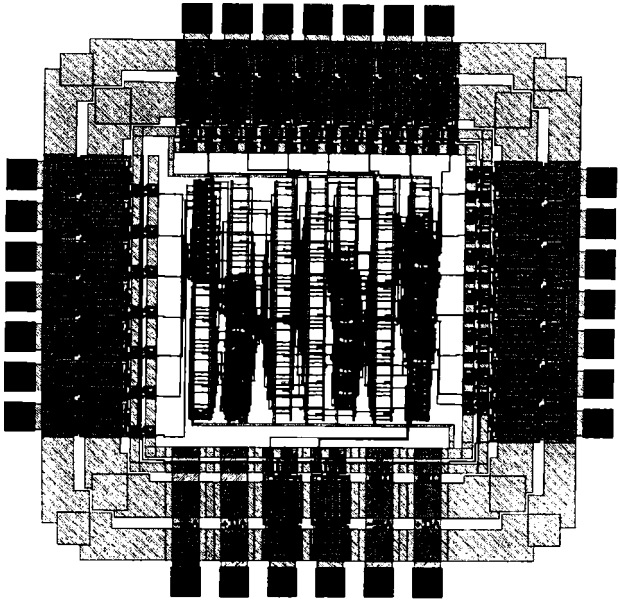


Figura 4.19. Asic dds1.

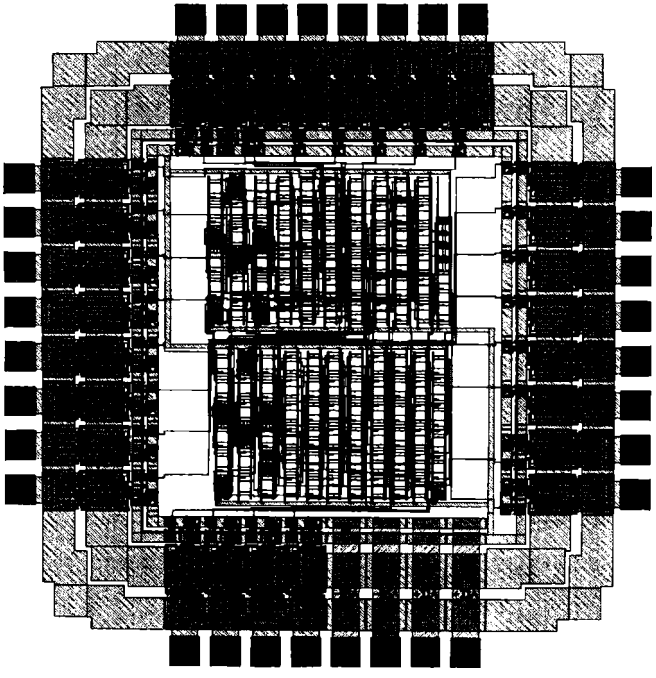


Figura 4.20. Asic dds2.

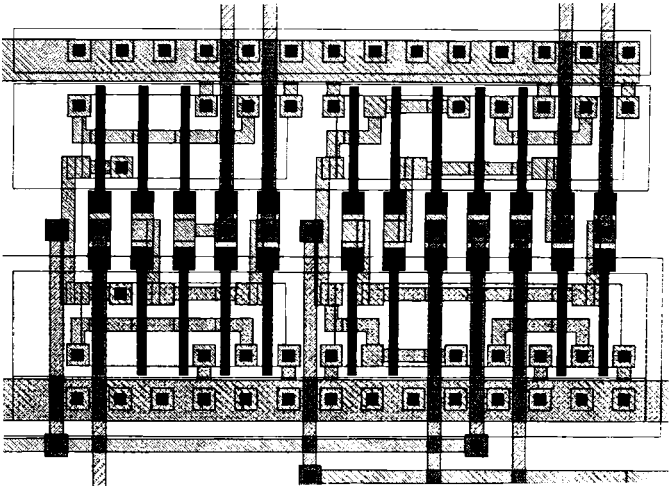


Figura 4.21. Asic dds2 - detaliu.

Tabelul 4.1

Sinteza caracteristicilor pentru variantele implementate

	Criteriul de comparație	Varianta de implementare		
		dds	dds1	dds2
1.	Terminale	31	27	32
2.	Aria activă ocupată	5,62 mm^2	4,98 mm^2	5,88 mm^2
3.	Aria ocupată de resursele logice	1,46 mm^2	1,13 mm^2	1,59 mm^2
4.	Aria globală ocupată	6,66 mm^2	5,96 mm^2	6,95 mm^2
5.	Timp de calcul normat CPU	3,35	4,29	3,9
6.	Porți utilizate în implementare	91	100	146
7.	Etaje fizice necesare	854	788	2035
8.	Etaje fizice nefolosite	42	31	77
9.	Frecvența maximă [MHz]	40	45	75

Pentru o evaluare cât mai cuprinzătoare a performanțelor diferitelor variante implementate, am dedus în tabelul 4.2 dependența simultană dintre tensiunea de alimentare, temperatura internă și întârzierea normată introdusă de circuit.

Tabelul 4.2

Întârzierea normată în funcție de temperatură și tensiunea de alimentare

Întârziere normată		Temperatura internă [C]											
		-60	-40	-20	0	20	40	60	80	100	120	140	160
Tensiunea de alimentare [V]	3	1,281	1.491	1,68	1.869	2.058	2.268	2.457	2.646	2.835	3.045	3.234	3.423
	3,3	1,1529	1,3419	1,512	1,6821	1,8522	2.0412	2.2113	2.3814	2.5515	2.7405	2.9106	3.0807
	3,4	1.0492	1.2212	1,376	1,5308	1,6856	1.8576	2.0124	2.1672	2.322	2.494	2.6488	2.8036
	3,6	0.9577	1.1147	1.256	1,3973	1,5386	1.6956	1.8369	1.9782	2.1195	2.2765	2.4178	2.5591
	3,8	0.8723	1.0153	1.144	1.2727	1.4014	1.5444	1.6731	1.8018	1.9305	2.0735	2.2022	2.3309
	4	0.8113	0.9443	1.064	1.1837	1.3034	1.4364	1.5561	1.6758	1.7955	1.9285	2.0482	2.1679
	4,2	0.7625	0.8875	1	1.1125	1.225	1.35	1.4625	1.575	1.6875	1.8125	1.925	2.0375
	4,4	0.7137	0.8307	0.936	1.0413	1.1466	1.2636	1.3689	1.4742	1.5795	1.6965	1.8018	1.9071
	4,6	0.6466	0.7526	0.848	0.9434	1.0388	1.1448	1.2402	1.3356	1.431	1.537	1.6324	1.7278
	4,8	0.6405	0.7455	0.84	0.9345	1.029	1.134	1.2285	1.323	1.4175	1.5225	1.617	1.7115
	5	0.61	0.71	0.8	0.89	0.98	1.08	1.17	1.26	1.35	1.45	1.54	1.63
	5,2	0.5917	0.6887	0.776	0.8633	0.9506	1.0476	1.1349	1.2222	1.3095	1.4065	1.4938	1.5811
	5,4	0.5734	0.6674	0.752	0.8366	0.9212	1.0152	1.0998	1.1844	1.269	1.363	1.4476	1.5322
	5,6	0.5673	0.6603	0.744	0.8277	0.9114	1.0044	1.0881	1.1718	1.2555	1.3485	1.4322	1.5159

Această dependență este cel mai bine observată în reprezentarea tridimensională din figura 4.22, care poate constitui un punct de plecare pentru o evaluare rapidă a performanțelor de viteză a circuitului studiat.

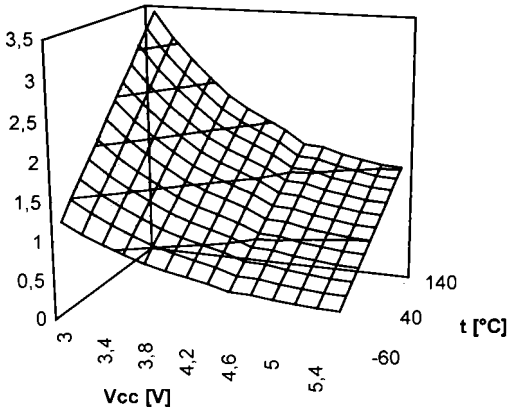


Figura 4.22. Întârzierea normată în funcție de tensiunea de alimentare Vcc și temperatură.

4.5. CONCLUZII

Pornind de la analiza posibilităților de implementare a unui SNF prezentată în prima parte a acestui capitol, autorul dezvoltă un studiu original al modalităților concrete de realizare a unui sintetizor de frecvență, oferind mai multe variante de rezolvare. Au fost evidențiate principalele metode de reducere a dimensiunii memoriei ROM din componența LUT, dezvoltând o idee prezentată în [Gontean86] la o sesiune de comunicări științifice studențești.

O posibilitate interesantă este realizarea unei scheme de sintetizor numeric de frecvență cu inversoare (figura 4.10), subiect abordat foarte puțin în literatură. Această variantă oferă cel mai redus timp de propagare la o astfel de schemă, cu dezavantajul unui spectru înrăutățit al semnalului de ieșire, comparat cu varianta în care LUT este implementat cu ROM..

Autorul menționează acumulatorul ca fiind componenta critică a sintetizorului. Un studiu detaliat privind arhitectura acumulatorului demonstrează avantajele metodei pipe-line de implementare, în cadrul acestui capitol propunându-se trei variante diferite de implementare (figurile 4.11, 4.12, 4.13).

Îmbunătățirea suplimentară a arhitecturii din figura 4.13 realizată pe baza înțelegerii intime a modului de operare al sintetizorului a făcut posibilă o reducere cu 40% a resurselor logice implicate cu 40%, rezultând varianta prezentată în figura 4.14.

Bazat pe cercetarea efectuată asupra diverselor arhitecturi de acumulator și ținând cont de sinteza structurilor logice programabile efectuată în capitolul 2, autorul propune utilizarea particulară a unei celule dintr-un FPGA Xilinx XC3000 pentru implementarea unui acumulator complet pipe-line pe un bit (figura 4.16). Deși literatura nu menționează explicit această posibilitate, o asemenea implementare beneficiază de structura intimă a logicii din această familie de FPGA, permițând totodată frecvențe ridicate de lucru (peste 50 MHz la familia XC3000).

O altă contribuție originală este legătura făcută de autor între arhitectura ASIC cu layout dreptunghiular suportată de programul ES^2 și cea a unui FPGA simetric ca și caz particular.

Consultarea atentă a datelor de catalog furnizate de producător a permis stabilirea grafică și analitică a dependenței dintre întârzierea normată și temperatură, respectiv tensiunea de alimentare. Această dependență este redată într-o formă convenabilă de prezentare (figura 4.22).

Trei dintre variantele de sintetizor de frecvență prezentate succint în acest capitol au fost implementate, testate și rutate de autor cu ajutorul programului Solo 1400 pe rețeaua Sun a Universității Preston din Anglia. Lay-out-ul final pentru aceste variante este redat în figurile 4.18-4.20, iar un detaliu de rutare în figura 4.21. Toate aceste variante, implementări, verificări și rutări sunt bineînțeles, complet originale.

Pentru variantele implementate, autorul prezintă o sinteză comparativă a principalelor caracteristici. Această analiză reflectă performanțele de viteză ale variantei complet pipe-line îmbunătățite din figura 4.14.

Autorul demonstrează pe un caz concret o afirmație din capitolul 1, și anume că o abordare dependentă de tehnologie atent gândită și elaborată, poate oferi performanțe excelente. Implementarea schemei din figura 4.14 oferă o viteză maximă de funcționare cu 67% mai mare decât cea mai performantă variantă clasică, la un consum global de porți crescut cu doar 46% față de aceasta. Acesta este un rezultat absolut remarcabil.

4.6. REFERINȚE BIBLIOGRAFICE

Literatura citată în cadrul acestui capitol acoperă mai multe domenii distincte: sinteza numerică de frecvență, îmbunătățirea performanțelor arhitecturilor utilizate, reducerea zgomotelor, proiectare și implementare de circuite VLSI.

Primul SNF descris și implementat în tehnologie TTL apare în [Tierney71], lucrare de referință citată constant în ultimele două decenii. În [Tierney75] sunt prezentate principalele aspecte legate de sinteza de frecvență și implementările posibile cu tehnologia disponibilă până în acel moment. Un studiu atent al zgomotelor inerente sintezei numerice de frecvență este prezentat în [Mehrgardt83], împreună cu mijloace specifice de a reduce aceste efecte nedorite. În același context se înscriu [Nicholas87] și [Nicholas88], lucrări care tratează efectele trunchierii lungimii cuvântului în acumulator, respectiv efectul global al trunchierilor efectuate în întregul sintetizor. [O'Leary91] se ocupă de asemenea de reducerea zgomotelor, propunând o arhitectură îmbunătățită de sintetizor. O altă lucrare de referință, [Sunderland84] prezintă pentru prima dată o arhitectură care să reducă dimensiunea memoriei ROM ocupate.

Lucrare mai mult teoretică, [Crawford94] sintetizează majoritatea cunoștințelor până în acel moment și este mai ales utilă la capitolele extra, cum ar fi reducerea dimensiunii memoriei ROM, reducerea zgomotelor, etc. [Bruma93] prezintă tehnici moderne de îmbunătățire a vitezei sumatoarelor. [Hatamian87] descrie o structură pipe-line de procesare de semnal, una din primele abordări care prezintă o soluție pipe-line de viteză. [Holtkamp94] prezintă o structură de sintetizor cu spectru larg utilizabil în comunicații. Un sintetizor cu o componentă spectrală îmbunătățită este prezentat în [O'Leary91].

Manualul oferit de producătorul programelor pe Sun, [Solo91], prezintă concis toate aspectele legate de utilizarea programelor respective, dar oricum nu poate suplini experiența în sinteza circuitelor numerice, care nici nu este trecută în bibliografie.

Proiectarea generală de VLSI este tratată în [Dillinger88], [Hill93], [Hu85], [Naish88], [Weste93]. Arhitectura Xilinx este prezentată în cataloagele de firmă, din care am citat doar [Xilinx94].

Dintre lucrările personale, [Gontean96] prezintă realizările autorului în timpul stagiului efectuat în Anglia, la Universitatea Central Lancashire din Preston. În [Gontean97] sunt reluate succint câteva dintre aceste chestiuni, iar rezultatele şi concluziile personale privind implementările sunt prezentate în [Gontean98a].

CAPITOLUL 5

PROGRAMATOARE

5.0. INTRODUCERE

Programarea este o verigă esențială în dezvoltarea de proiecte cu SLP. Tipic, această etapă se execută în programatoare, deși există și SLP care se pot programa în montaj, fără un alt dispozitiv suplimentar. În continuare se va efectua o prezentare a aspectelor esențiale programării SLP și a principalelor preocupări ale autorului în domeniul programatoarelor.

Realizarea unui programator performant la un preț redus este o sarcină dificilă care impune rezolvarea unor probleme diverse, pornind de la electronica analogică, continuând cu logica programabilă și încheind cu programele de comandă, respectiv interfața software utilizator. Este prezentat și un studiu pertinent asupra posibilităților de implementare a unui programator, reliefându-se avantajele fiecărei alternative în parte.

În acest capitol sunt prezentate două programatoare realizate practic de autor, unul specializat pe un anumit tip de circuite și altul universal. Raportat la momentul realizării lor (perioada 1991-1993), pe piață au apărut circuite specializate care permit o implementare cu un număr mai redus de cipuri, principiile de realizare propuse păstrându-și valabilitatea.

Se poate afirma că prezența acestui capitol încheie natural cercul preocupărilor autorului în domeniul proiectării optimale cu SLP, deoarece orice proiect, oricât de atractiv ar fi pe hârtie sau în memoria calculatorului, rămâne un exercițiu academic în absența implementării sale fizice, pentru care programarea este indispensabilă.

5.1. PROGRAMAREA SLP

Pentru coerența materialului prezentat, am selectat tipurile reprezentative de SLP din punct de vedere al modului de programare. De aceea sunt prezentate trei tipuri distincte de SLP: PAL, EPLD și ispLSI. Pentru PAL-urile clasice, procesul de programare este ireversibil și el se materializează prin arderea fuzibilelor ce urmează a fi programate. SLP bazate pe tehnologia EPROM sau EEPROM se pot șterge prin expunere la radiație ultravioletă sau electric. FPGA Actel (bazate pe antifuzibile ca element de interconectare) sunt programabile o singură dată.

O mențiune specială trebuie făcută pentru FPGA Xilinx. Așa cum am arătat în capitolul 2, soluția oferită de acest producător se bazează pe două circuite: FPGA-ul propriu-zis și un EPROM de dimensiuni reduse de configurare. De aceea, este cu atât mai interesant de implementat un programator combinat pentru SLP și EPROM, care să poată deservi și FPGA Xilinx. Prin aceasta se asigură o flexibilitate sporită și o independență sporită față de necesitatea unui programator dedicat sau universal scump, respectiv de dispozitive de adaptare cu socluri diverse, așa cum se va vedea la §5.2.4.

5.1.1. PROGRAMAREA FAMILIEI PAL 16xxx

Fiecare fuzibil poate fi programat cu o secvență simplă de tensiuni aplicate la doi pini de control (pinii 1 și 11) și pe ieșirea care se programează. Adresarea ariei de 2048 de elemente fuzibile se realizează cu ajutorul unor semnale TTL aplicate la cele 8 intrări (fuzibilele de pe o coloană sunt selectate cu ajutorul a 5 biți; cu restul de 3 biți se selectează poarta ȘI din cele opt corespunzătoare unei celule). Pe durata programării, tensiunea de alimentare V_{CC} este menținută la + 5 V, iar curentul consumat de capsulă este $I_{CC} = 50 \div 200$ mA.

Cronogramele programării valabile pentru tipurile 16L8, 16R8, 16R6, 16R4, 16H8, 16LD8 și 16 HD8 sunt prezentate în figura 5.1. După ce programarea s-a încheiat, întreaga arie de fuzibile se verifică la $V_{CCL} = 4{,}3$ V și $V_{CCH} = 5{,}7$ V. Verificarea poate fi făcută citind toate ieșirile simultan sau fiecare ieșire pe rând. Un fuzibil de protecție permite împiedicarea oricărei încercări de copiere (toate fuzibilele vor apărea ca fiind programate).

Principalele caracteristici electrice ale procesului de programare sunt:

- curenții de programare la pinii 1, 11: $10 \div 40$ mA;
- $\frac{dV_{11}}{dt} = 100 \div 1000$ V/μs;
- $R = 2$ kΩ;
- $t_P = 50$ μs $\div$ 5 ms;
- $t_D = 100 \div 1000$ ns.

Pentru a programa fuzibilul de protecție se procedează astfel:

1. Se alimentează circuitul la $V_{CCP} = 5{,}2$ V;
2. Se ridică nivelul tensiunii la pinul 1 la 11 V;
3. Se furnizează la ieșirile Out0 $\div$ Out7 un impuls cu durata de 50 μs de amplitudine $V_{OP} = 20$ V;
4. Se verifică dacă operațiunea s-a încheiat cu succes (test la V_{CCL}, V_{CCH}).

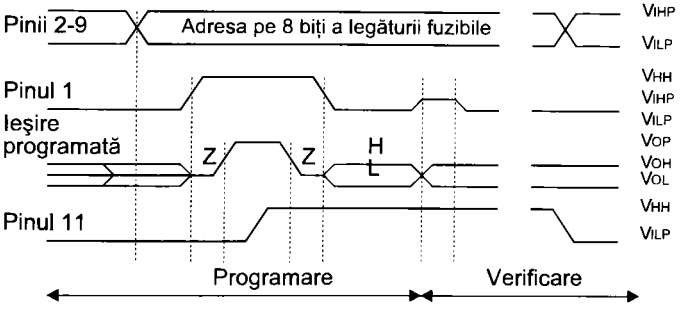


Figura 5.1. Cronogramele programării unui PAL.

Schema simplificată folosită la programarea unui PAL 16xx este prezentată în figura 5.2.

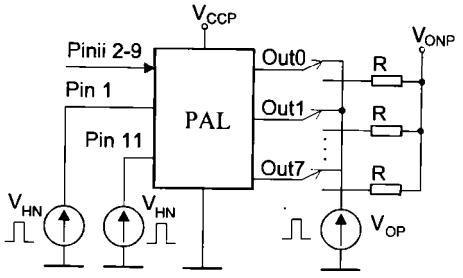


Figura 5.2. Schema simplificată a programării unui PAL.

5.1.2. PROGRAMAREA FAMILIEI ALTERA EP6x0

Locul conexiunilor fuzibile din PAL-urile bipolare a fost luat în cazul EPLD-urilor de celule echipate cu tranzistoare NMOS; o asemenea celulă este foarte asemănătoare cu o celulă de EPROM uzual (tehnologie 0,8 microni) şi a fost perfecţionată suficient pentru a se putea prezice statistic o rată de defectare 1/100.000 ani de funcţionare continuă la 70°C. Pentru EPLD-urile din seria EP6x0, cronogramele de programare sunt cele din figura 5.3, corespunzător următorilor paşi:

1. potenţialul pinului de programare (13) este adus la V_{PP} = 12,5 V;
2. adresele de rând şi coloană sunt aduse la pinii 2, 11, 15 ÷ 23;
3. octetul în vederea programării este aplicat pinilor 3 ÷ 10;
4. algoritmul de programare este analog celui de programare rapidă a memoriilor EPROM (cu impulsuri având durata de 100 μs), urmate de cicluri separate de verificare;
5. pentru a mări siguranţa datelor înscrise pot fi furnizate impulsuri adiţionale de programare.

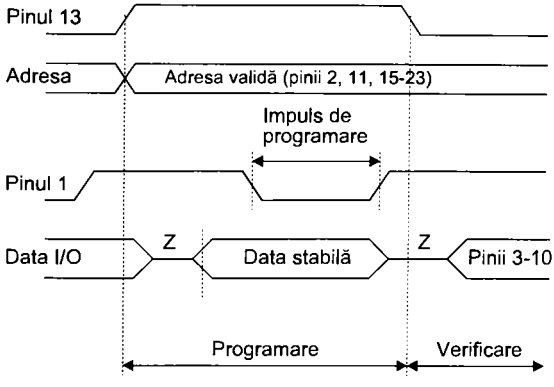


Figura 5.3. Cronogramele de programare pentru circuitele din seria EP6x0.

Este oferit un bit de protecție programabil (Security Bit), care împiedică orice tentativă de interogare ulterioară a circuitului, protejând prin aceasta proiectarea efectuată.

Datorită apariției tranzistoarelor bipolare parazite în procesul de integrare al dispozitivelor CMOS, este posibil uneori să se producă fenomenul de agățare. Proiectarea circuitului se face astfel încât în condiții normale de funcționare, respectivele tranzistoare bipolare să nu ajungă niciodată în conducție.

Pentru a reduce șansa apariției fenomenului de agățare, utilizatorul trebuie să asigure la conectare secvența următoare de aplicare a tensiunilor de alimentare și de intrare:

1. V_{SS} (GND); 2. V_{CC} (+5 V); 3. Intrări.

La deconectare, secvența se parcurge invers. În cazul unui EPLD frecvent manipulat într-un soclu, se recomandă circuitul de protecție din figura 5.4.

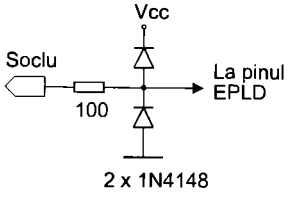


Figura 5.4. Circuit de protecție pentru EPLD-uri montate în soclu.

5.1.3. PROGRAMAREA FAMILIEI ispLSI

Familia isp (**I**n **S**ystem **P**rogrammable) oferită de firma Lattice se programează simplu, practic fără un programator dedicat, existând posibilitatea programării în montaj a circuitului (figura 5.5.a).

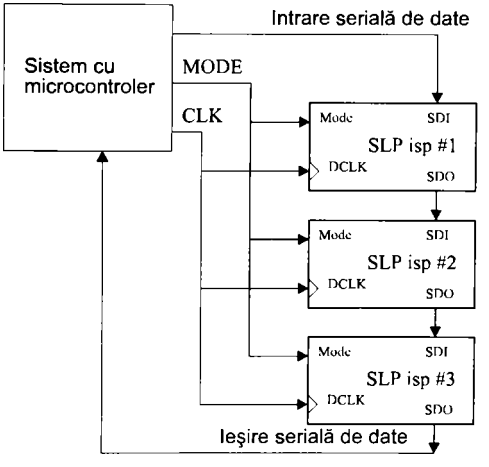


Figura 5.5.a. Programarea familiei ispLSI - schema bloc.

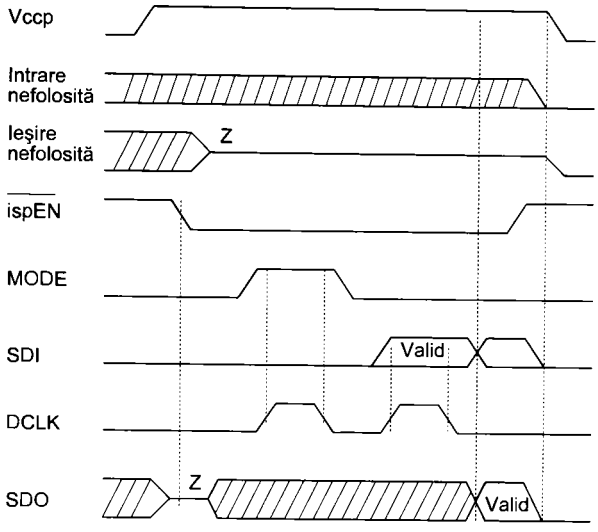


Figura 5.5.b. Programarea familiei ispLSI - cronograme.

Datele se înscriu în SLP prin intermediul liniei SDI şi se citesc pe linia SDO. SCLK asigură caracterul sincron al operaţiilor, iar $\overline{ispEN}$ este intrarea de validare a circuitului. Diagramele de timp corespunzătoare programării sunt prezentate în figura 5.5.b.

5.2. PROGRAMATOARE INDUSTRIALE DISPONIBILE PE PIAŢĂ

În prezent există un mare număr de programatoare universale oferite de diverse firme. În continuare sunt prezentate succint trei tipuri reprezentative, aparţinând unor firme de renume în domeniu.

5.2.1. CLASIFICAREA PROGRAMATOARELOR

O clasificare a programatoarelor este nu numai utilă, dar şi necesară în acest punct al prezentării. Am clasificat programatoarele după o serie de criterii, după cum urmează:

1. După *destinaţie*:
   a) dedicate;
   b) universale.

2. După *structura internă*:
   a) programatoare independente;
   b) programatoare ce folosesc un calculator gazdă.

3. După *modul de comunicare* cu calculatorul:
   a) utilizând interfața serială RS232;
   b) folosind un slot de extensie în calculator;
   c) utilizând interfața paralel (portul de imprimantă);
   d) folosind alte interfețe.

4. După *numărul de circuite programate* la un moment dat:
   a) un singur circuit;
   b) mai multe circuite.

Un *programator dedicat* acționează asupra unui singur tip de circuit programabil (PROM, EPROM, PLA, PAL, EPLD). Este simplu de utilizat, dar limitările sale sunt evidente. Se mențin pe piață mai ales programatoarele dedicate destinate SLP-urilor complexe livrate chiar de compania producătoare a SLP-ului respectiv.

Un *programator universal* este mai scump, dar prezintă o flexibilitate sporită și limitată doar de software-ul aferent. În plus, datorită necesității de a multiplexa pe aceeași linie niveluri TTL (sau CMOS) cu tensiuni ridicate (pentru programare), este foarte simplă utilizarea unui asemenea programator pentru testarea circuitelor logice și/sau a memoriilor semiconductoare.

Un *programator independent* este cel mai scump din lista de mai sus (poate ajunge la 4000 $). Este necesar un microsistem dedicat, o memorie operativă de 1 - 4 Mocteți, posibilități de comunicare cu un calculator. Motivul principal al alegerii unui astfel de programator este că el nu este dependent de o arhitectură dată, ale cărei resurse le partajează. O asemenea investiție nu se justifică decât în foarte puține cazuri și de aceea se preferă un programator ce folosește un calculator gazdă.

5.2.2. COMUNICAREA ÎNTRE PROGRAMATOR ȘI CALCULATOR

Legătura dintre programator și calculator este cel mai des realizată folosind interfața serială sau cea paralelă, modalitate recomandată de majoritatea producătorilor de circuite programabile în cazul unui programator independent. Pentru un programator ce utilizează resursele unui calculator gazdă, această metodă are avantajul că nu ocupă un slot de extensie (care s-ar putea să nu existe) și nu utilizează nici adrese I/O suplimentare. Dezavantajul constă în ocuparea unui port serial sau paralel. În continuare vor fi discutate pe scurt variantele de comunicare.

A. Placă de interfață în calculator

Această opțiune este cea care permite accesul direct la resursele calculatorului. De obicei, interfața este proiectată pentru o magistrală de 8 biți, de tip XT. Sunt disponibile tensiunile de alimentare +5V (alimentarea logicii interfeței și a programatorului), +12V (este posibilă folosirea unui convertor cc/cc pentru generarea tensiunilor mari [14 - 25V]), -12V (pentru eventuala parte analogică) și -5V (de obicei nefolosit).

Este posibilă generarea de stări WAIT, accesul la un canal DMA, utilizarea interfețelor serie și paralel. Schema electrică este deosebit de simplă, necesitând o logică de selecție (cu spațiu de adresare selectabil), buffere și latch-uri. Este necesar un slot de extensie liber, ceea ce constituie principalul dezavantaj al acestei abordări, deoarece nu întotdeauna acest slot există (de

exemplu, în cazul note-book-urilor) sau nu este liber (în cazul unui sistem cu magistrală PCI, eventual conectat la o rețea, cu scanner, placă de sunet și cartelă modem cele 3 sau 4 sloturi ISA se ocupă foarte repede).

B. Comunicația prin interfața serială V24

Deși a fost cea mai utilizată interfață pentru programatoare, de la începutul acestui deceniu locul ei a fost luat de comunicația pe portul paralel. Avantajele și dezavantajele ei sunt prezentate în tabelul 5.1.

Tabelul 5.1

Interfața RS232

AVANTAJE	DEZAVANTAJE
• Legătura simplă (3...5 fire) • Lungime mare a firelor de legătură • Este des utilizată și recomandată • Este independentă de calculatorul folosit	• Viteza redusă - maxim 38400 baud • Necesită un microsistem dedicat (programatorul este mai scump).

C. Comunicația prin interfața paralelă de imprimantă

Alegerea portului de imprimantă (gândit inițial ca mijloc de ieșire a datelor din calculator) a fost intens sprijinită de autor încă din 1991 [Gontean91], moment în care nici pe plan local și nici mondial nu se manifestau asemenea tendințe. Deoarece portul de imprimantă dispune de 8 linii de date unidirecționale, și doar 5 linii de stare de intrare, complicarea montajului electronic este evidentă. Abia după 1993 se remarcă generalizarea utilizării portului de imprimantă, cu toate că și în prezent firmele renumite producătoare de programatoare folosesc în continuare portul serial, comportament dictat de specificațiile firmelor producătoare de circuite programabile (Intel, Motorola, AMD, etc).

ROM-BIOS acceptă până la 3 porturi paralele de imprimantă, care sunt asignate drept LPT1-LPT3. În timpul execuției POST (**P**ower **O**n **S**elf **T**est), BIOS-ul testează existenta acestor porturi în următoarea ordine:

1. 3BCH - port aflat pe placa MDPA (**M**onochrome **D**isplay/ **P**rinter **A**dapter);
2. 378H - interfața de imprimantă #1;
3. 278H - interfața de imprimantă #2,

iar în cazul în care le găsește, le asignează ca LPT1-LPT3. Adresa de bază a porturilor care răspund testelor se memorează la 0040:0008, în cadrul variabilelor ROM-BIOS. Pentru a putea utiliza (prin intermediul Int17) și o a patra interfață de imprimantă, adresa ei trebuie memorată la adresa 0040:0010.

Descrierea semnificației porturilor folosite într-o interfață paralelă de imprimantă (unde cu X s-a notat adresa de bază a interfeței, care poate fi 3BCH, 378H, 278H sau alta, după cum s-a văzut mai sus) este prezentată în continuare:

- X+0: Printer Data Latch, la *scriere* , trimite octet la imprimantă;
  la *citire*, citește ultimul octet trimis.

Datele scrise în portul X+0 pot fi citite cu ajutorul unei instrucțiuni IN făcute la aceeași adresă; aceasta permite o verificare ușoară a existenței interfeței de imprimantă cu adresa de bază X. O altă utilizare este diagnosticarea interfeței și a circuitului de intrare al imprimantei: o defecțiune se poate sesiza ușor făcând un OUT la adresa X+0, apoi un IN de la aceeași adresă, operații urmate de un XOR între octetul înscris și cel citit. Orice bit pe 1 semnifică existența unei defecțiuni.

♦ X+2: Printer Controls

La acest port se poate face atât scriere, cât și citire; semnificația biților citiți/scriși este:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
+Strobe	+AUTO Line Feed	-INT	+Select Input	+IRQ Enable	0	0	0

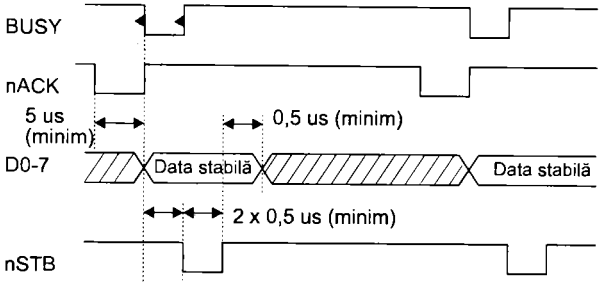


Figura 5.6. Comunicația pe portul paralel.

Liniile + Strobe, + AUTO Line Feed și + Select Input sunt negate, astfel încât la cuplă se vor găsi semnalele - Strobe, - AUTO Line Feed, - Select Input. Semnalele sunt inversate la citire încă o dată, așa că un octet trimis la portul X+2 va fi citit (cu biții 5-7, însă, pe 0).

Datele sunt citite, cu excepția lui BUSY (care este inversat) direct din cuplă. Modalitatea în care se realizează transferul datelor se poate observa din figura 5.6.

♦ X+1: *Doar citire*

Semnificația biților citiți de la acest port este următoarea:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
-BUSY	-ACK.	+P. END	+SELECT	-ERROR	0	0	0

Tabelul 5.2

Interfața paralelă

DEZAVANTAJE	AVANTAJE
• Legătură complexă (25 fire) • Lungime redusă a firelor de legătură • Necesită 5-6 operații I/O pentru un octet transferat	• Viteză ridicată de comunicare • NU necesită un microsistem dedicat (programatorul este mai ieftin). • Este independentă de tipul de PC folosit

Conectorul cu 25 de contacte la care se atașează imprimanta este prezentat în figura 5.7. Principalele caracteristici ale interfeței paralel, aleasă ca mijloc de comunicare pentru un programator sunt rezumate în tabelul 5.2.

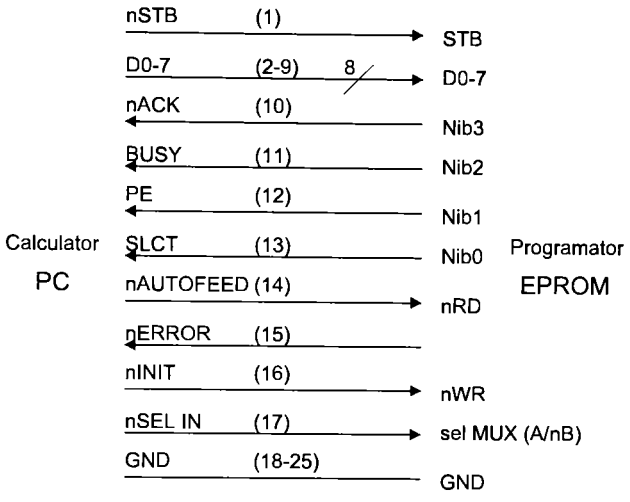


Figura 5.7. Semnalele la portul paralel.

În tabelul 5.3. sunt rezumate principalele opțiuni pentru conectarea unui calculator de tip PC la un programator (cu +++ s-a notat un avantaj maxim pentru varianta respectivă). Studiul tabelului indică avantajele și dezavantajele fiecărei metode.

Dorind implementarea unui programator ieftin și evitarea deschiderii calculatorului (de exemplu datorită pierderii garanției), se impune alegerea interfeței paralele ca mijloc de comunicare.

În final mai trebuie menționat faptul că PC-urile produse recent dispun de un port paralel *complet* bidirecțional, pe care viteza de transfer a informației este substanțial îmbunătățită.

Tabelul 5.3

Caracteristicile comunicației programator - PC

Caracteristica	**Placă în calculator**	**RS232**	**Interfața paralelă**
Viteza de lucru	+++	+	++
Slot de extensie	DA	NU	NU
Microsistem dedicat	NU	DA	DA
Portabilitate	+	+++	++
Alimentare separată	NU	DA	DA
Fiabilitate	+	++	+++
Adrese I-O	DA	NU	NU

5.2.3. FORMATE PENTRU INTERSCHIMBAREA INFORMAȚIEI

Producătorii de SLP au dorit să evite situația întâlnită la programarea memoriilor EPROM, unde sunt utilizate cel puțin cinci formate de codificare a informației între calculator și dispozitivul de programare. Preocupările în acest sens au condus la dezvoltarea standardului JEDEC și au continuat cu dorința de a implementa un standard de interschimbare a informației la nivel de *proiect*, eforturi concretizate în formatul EDIF.

A. Formatul JEDEC

Un fișier în format JEDEC este un fișier ASCII care poate fi ușor vizualizat și editat sau tipărit. Fișierul JEDEC începe cu caracterul 02H (STX, start-of-text) și se încheie cu 03H (ETX, end-of-text). Fișierul este divizat în câmpuri. Fiecare câmp, cu excepția primului (numit câmp de identificare), începe cu un identificator de câmp și se încheie cu un asterisc (*). Identificatorul de câmp este un caracter (o literă); diferite subcâmpuri pot fi definite prin caractere suplimentare. Câmpurile au priorități diferite, unele fiind generate automat de compilator, altele sunt destinate special dispozitivului de programare sau testorului și în fine, unele sunt opționale. Câmpurile și subcâmpurile definite de standardul JEDEC sunt listate în tabelul 5.4.

Tabelul 5.4

Identificatorii de câmp conform standardului JEDEC 3-A

Identificator	Descriere
(nimic)	Identificarea proiectului (Design Specification)
N	Comentarii (Note)
QF	Număr de fuzibile în circuit (Number of Fuses in Device)
QP	Numărul de pini în vectorii de test (Number of Pins in Test Vectors)
QV	Numărul maxim de vectori de test (Maximum Number of Test Vectors)
F	Starea implicită a fuzibilelor (Default Fuse State)
L	Lista fuzibilelor (Fuse List)
C	Suma de control (Fuse Checksum)
X	Condiția de testare implicită (Default Test Condition)
V	Vectori de test (Test Vectors)
P	Pinii circuitului (Pin sequence)
D	Circuit (Device, obsolete identifier)
G	Fuzibil de siguranță (Security Fuse)
R	Signature Analysis, Resulting Vector
S	Signature Analysis, Starting Vector
T	Signature Analysis, Test Cycles
A	Timpul de acces (Access Time)

Câmpul de identificare începe imediat după caracterul STX și se încheie obligatoriu cu un asterisc. Standardul JEDEC recomandă ca acest câmp să conțină:

1. numele utilizatorului și firma;
2. data, numărul de cod al proiectului, revizia;
3. numele PLD-ului;
4. alte informații.

Câmpul de comentarii este ignorat de programator. Subcâmpurile QF, QP şi QV furnizează dispozitivului de programare valori numerice, de exemplu QV este destinat a asista dispozitivul de programare în alocarea memoriei pentru vectorii de test.

Câmpurile F, L şi C definesc modul de programare al fuzibilelor, numerotate liniar de la 0 la *n* -1. Câmpul F defineşte starea implicită a fuzibilelor care nu sunt descrise în continuare, (întotdeauna 0 sau 1). Lista fuzibilelor din câmpul L începe întotdeauna cu un număr de ordine zecimal al fuzibilului, un spaţiu (sau <**CR**>) şi un şir de "0" sau "1". Suma de control din câmpul C permite detectarea eventualelor erori la comunicarea cu dispozitivul de programare. Următoarele câmpuri sunt folosite pentru testare (definire vectori de test). Câmpul marcat prin X indică condiţia implicită pentru vectorii de test care nu vor fi descrişi în continuare sau condiţiile de test de tip "don't care".

Vectorii de test definiţi de câmpul V se execută în ordine numerică, indiferent de succesiunea din fişierul JEDEC. Câmpul V începe cu un număr de ordine zecimal al vectorului de test, urmat de un spaţiu şi respectiv de condiţiile de test pentru fiecare pin în parte. Condiţiile de test care se pot specifica sunt:

0	aplică nivel "0" logic la pin, considerat intrare;
1	aplică nivel "1" logic la pin, considerat intrare;
X	aplică nivelul logic implicit la pin;
C	aplică un semnal de tact ($0 \rightarrow 1 \rightarrow 0$) la pin, considerat intrare;
K	aplică un semnal de tact ($1 \rightarrow 0 \rightarrow 1$) la pin, considerat intrare;
2-9	aplică tensiuni speciale (super-voltage);
N	pinul respectiv nu se testează (alimentare, masă etc);
L	testează pinul pentru nivel "0";
H	testează pinul pentru nivel "1";
Z	testează pinul pentru starea HiZ;
P	preîncarcă bistabilele;
B	preîncarcă bistabilele ascunse (burried).

Tensiunile speciale sunt utilizate pentru a plasa SLP într-un mod particular de test. Preîncărcarea poate fi executată numai la anumite tipuri de SLP (şi în acest caz este necesară furnizarea unei tensiuni speciale pentru SLP).

Condiţiile de test sunt implicit aplicate în ordinea numerică specificată, de la stânga la dreapta. Prin intermediul câmpului P se poate altera această ordine. Câmpul A specifică întârzierea (propagation delay) pentru vectorii de test, în incremenţi de o nanosecundă. Acest câmp este în general nefolosit, fiind destinat unor testoare avansate (şi scumpe). Pentru testoarele care permit analiza semnăturii sunt utilizate câmpurile R, S şi T. Câmpul G specifică dacă se va programa fuzibilul de siguranţă (numai pentru SLP care *au* acest fuzibil).

B. Formatul EDIF

Pentru a realiza un standard de interschimbare a informaţiei CAE, şase producători din domeniu (Daisy Systems, Motorola, Mentor Graphics, National Semiconductor, Tektronix, şi Texas Instruments) şi-au unit eforturile în perioada 1983-1987 pentru implementarea unui nou standard, numit EDIF (**E**lectronic **D**esign **I**nterchange **F**ormat).

Majoritatea producătorilor de software pentru SLP au adoptat acest standard. Marele avantaj astfel obţinut este că diferitele proiecte cu SLP vor deveni portabile, indiferent de limbajul sau platforma pe care au fost create.

5.2.4. EXEMPLE DE PROGRAMATOARE INDUSTRIALE

EPROM-1 (figura 5.8) este un programator rapid, fiabil şi uşor de utilizat, produs de firma International Microsystems Inc. din S.U.A. În operarea ca programator dedicat, este utilizată tastatura şi doi digiţi de afişare. Programatorul utilizează algoritmi aprobaţi de producătorii circuitelor; un soclu de 28 de pini permite inserarea directă a celor mai uzuale circuite EPROM. Folosind adaptoare adecvate, alte circuite pot fi programate, ca de exemplu microcontrolere din familia 8748 şi 8751.

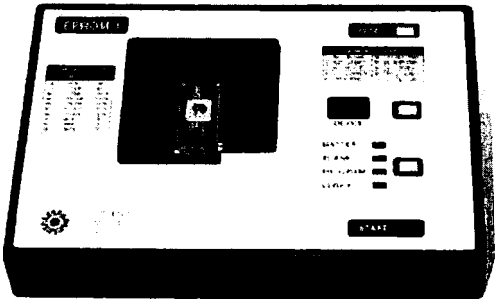


Figura 5.8. Programatorul dedicat EPROM-1 - vedere generală.

Familia de programatoare ChipLab, produse ale firmei Data I/O (numărul 1 mondial în domeniu) includ produsele ChipLab Memory Micro (destinate memoriilor şi microcontrolerelor), ChipLab Logic Plus (destinat PLD-urilor şi memoriilor) şi ChipLab Professional (destinat memoriilor, PLD-urilor şi microcontrolerelor). ChipLab poate programa practic orice memorie sau microcontroler până la 48 pini în capsulă DIP. Folosind adaptoare adecvate, se pot programa şi capsule PLCC, SOIC, QFP şi TSOP.

Algoritmii utilizaţi sunt nu numai aprobaţi, dar şi certificaţi de producătorii circuitelor, cu care DATA I/O păstrează o strânsă legătură.

Circuite programabile:

- EPROM, EEPROM, Flash EPROMs până la 48 pini;
- PROM bipolare;
- Microcontrolere Intel, Motorola, şi Philips;

Interfaţa software

- Interfaţă grafică tip Windows ce urmează standardele SAA/CUA;
- on-line help extins;
- Built-in editor pentru memorii EPROM, cu date hexa sau ASCII;
- Tratează standardele JEDEC, Intel (Intellec 8/MDS, MCS-86, Hex-32), Motorola (S1-S3) şi formatele binare.

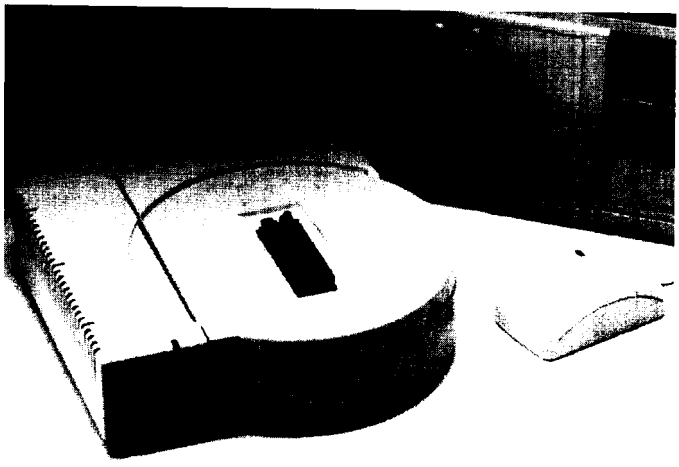


Figura 5.9. Programatorul universal ChipLab - vedere generală.

Accesorii standard:

- interfața software;
- cablul paralel;
- adaptorul DIP48-1.

Seria de programatoare independente universale M4000 include modelele M4008 şi M4016. M4008 poate programa opt circuite, utilizând două adaptoare; M4016 (figura 5.10) poate programa şaisprezece circuite cu patru adaptoare. Gama circuitelor ce pot fi programate este foarte variată, pornind de la memorii EPROM, până la cartele PCMCIA.

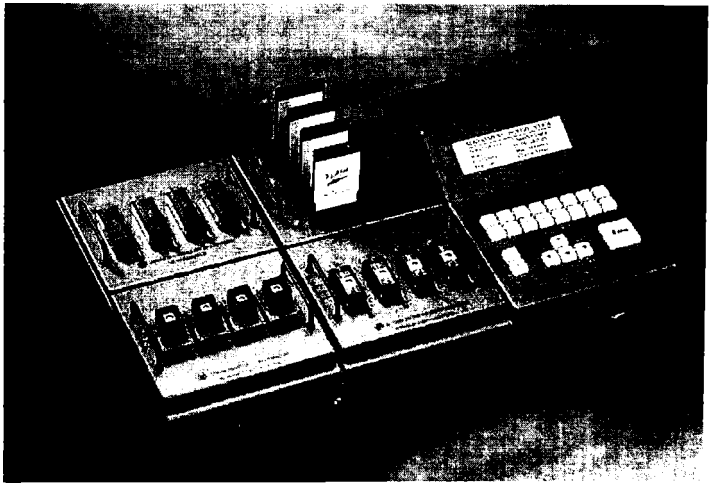


Figura 5.10. Programatorul independent M4016 - vedere generală.

Adaptoarele pe M4016 se pot instala diferit, de exemplu două de EPROM pe 32 de pini, una de microcontroler 8751 şi altul de cartelă PCMCIA. Operarea manuală se realizează prin intermediul unei tastaturi proprii şi a unui ecran LCD. Conţinutul introdus poate fi editat prin funcţii adecvate.

O inovaţie interesantă la M4008 şi M4016 constă în utilizarea pentru firmware de memorii Flash, care se pot reprograma uşor (contra cost) prin postul serial, direct de la PC. Seria M4000 se interfaţează cu calculatorul gazdă prin intermediul portului serial la maxim 19,200 bauds. Opţional, prin intermediul portului paralel se pot obţine rate de transfer mai mari, de până la 30,000 octeţi pe secundă.

5.3. PROGRAMATOR PENTRU FAMILIA SLP DE TIP GAL

După cum am arătat în introducerea acestui capitol, am implementat practic două programatoare:

- un programator dedicat pentru circuite de tip GAL, conectabil la un PC pe portul paralel;
- un programator universal, conectabil la un PC de asemenea prin intermediul poetului paralel.

În primul caz sarcina proiectării este sensibil uşurată de particularităţile de programare serială a familiei GAL, 8 linii de semnal de la PC fiind suficiente pentru implementarea programatorului. programator. Pentru interfaţarea cu portul paralel s-a utilizat un circuit 74LS373 (figura 5.11).

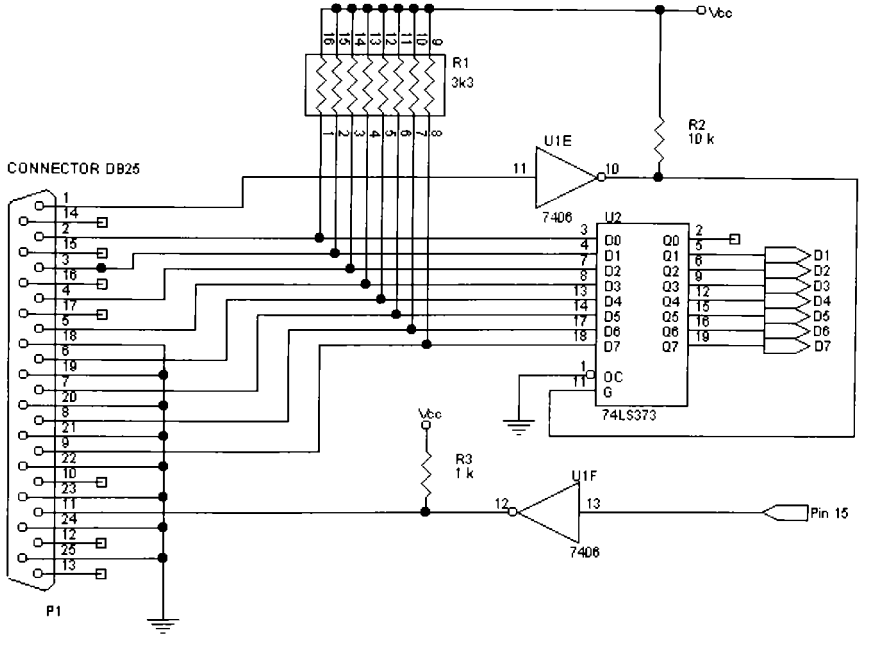


Figura 5.11. Interfaţă pentru programarea familiei GAL.

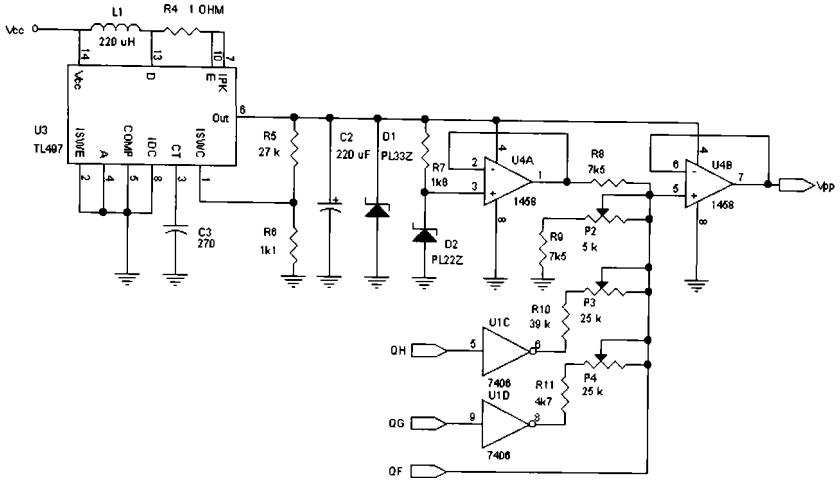


Figura 5.12. Convertor cc/cc pentru programator.

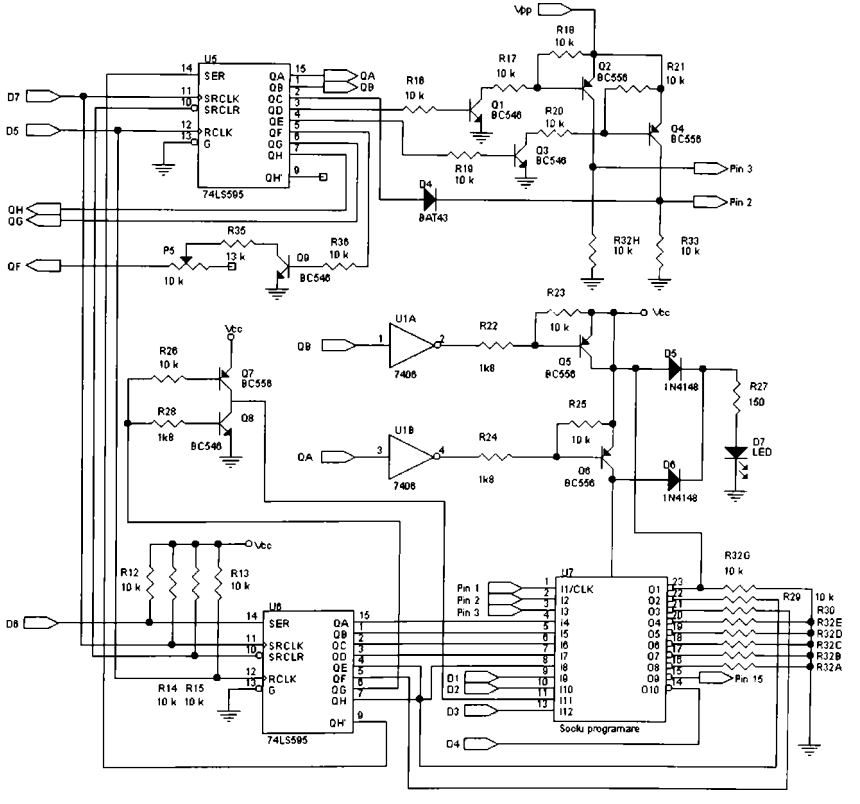


Figura 5.13. Programatorul propriu-zis.

Alimentarea programatorului de la conectorul de tastatură este o soluție originală, pe care am utilizat-o și în alte situații, de exemplu [Gontean94]. Pentru furnizarea tensiunii de programare, a fost realizat un convertor cc/cc, implementat în jurul circuitului TL497. Acesta furnizează la pinul 6 o tensiune: $1{,}2\ \text{V} \cdot \frac{R_5 + R_6}{R_6} \approx 30{,}65\ \text{V}$, de unde se va obține Vpp, conform schemei din figura 5.12.

Deoarece cele opt linii de date ale portului paralel nu sunt suficiente pentru a ocupa toți pinii GAL-ului în modul EDIT, trei linii de date (D5, D6, D7) sunt direcționate spre registrul de deplasare de 16 biți 74LS673.

Am utilizat un singur soclu TEXTOOL pentru întreaga familie GAL (figura 5.13), circuitele GAL16V8 fiind introduse decalat în soclu.

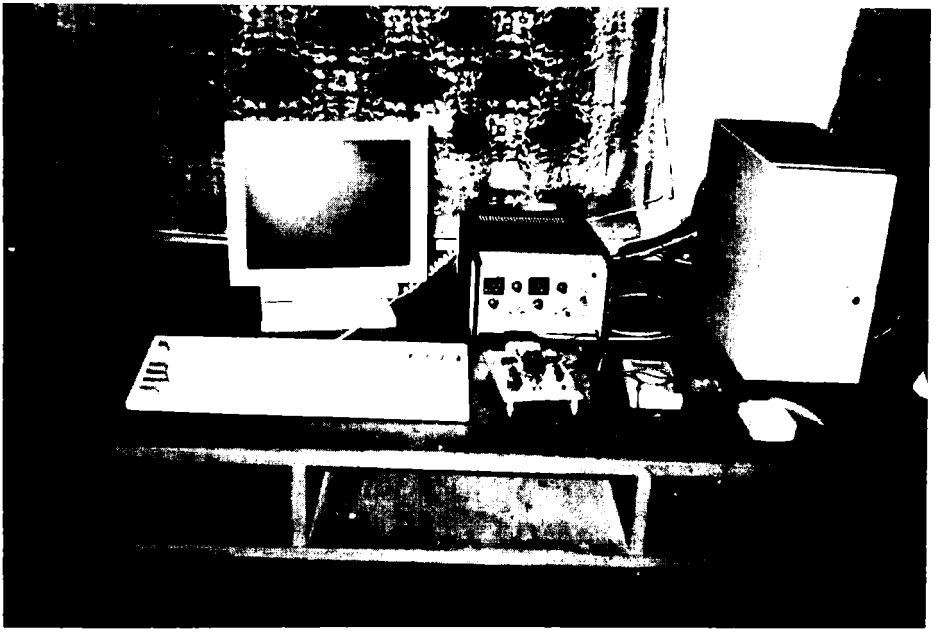


Figura 5.14. Programatorul realizat - vedere de ansamblu.

5.4. PROGRAMATORUL UNIVERSAL APROMMER

Cel de-al doilea programator implementat este un programator universal, pentru circuite EPROM/SLP, [Gontean92a], [Gontean97], [Gontean98b]. Cele mai importante etape în dezvoltarea acestui echipament au constat în:

1. elaborarea unui studiu bibliografic amănunțit al caracteristicilor de programare ale circuitelor disponibile pe piață;
2. elaborarea unei scheme bloc potrivită scopului dorit;

3. implementarea unui sistem de comunicație bidirecțional pe portul paralel;
4. proiectarea unor surse de tensiune programabile, fiabile și precise;
5. elaborarea unor scheme pentru electronica de pin de tip EPROM, respectiv de tip SLP;
6. implementarea unui set coerent și cuprinzător de rutine de autotestare;
7. elaborarea unui pachet de programe pentru utilizarea programatorului.

Elementele esențiale referitoare la proiectarea și funcționarea programatorului vor fi descrise în continuare. Toate aspectele prezentate în continuare sunt originale.

5.4.1. SCHEMA BLOC

Schema bloc a unității centrale este redată în figura 5.15. Existența unei magistrale unidirecționale de date implică folosirea:

- unui registru de adresare în care să fie memorat cipul selectat și contorul/portul din cadrul cipului;
- unui tampon pentru magistrala de date.

Prezența a (doar) 5 linii de intrare la cupla de imprimantă impune folosirea unui multiplexor pentru citirea datelor de pe magistrala internă de date.

Trei circuite PPI 8255 sunt folosite pentru comanda surselor programabile de tensiune, a liniilor bidirecționale de date și a liniilor unidirecționale de adrese. Un circuit timer 8253 este utilizat pentru generarea diferitelor temporizări. Se mai remarcă prezența unui circuit de protecție, a electronicii de pin și a modulelor de extensie.

Semnalele de la/spre 8255 au fost împărțite în 4 categorii:

a. Semnale de intrare: IN_0, IN_1;
b. Semnale de ieșire: RA_8 - RA_{23}, SA_0 - SA_7;
c. Semnale bidirecționale: RA_0 - RA_7, DQ_0 - DQ_{15};
d. Semnale de control: $\overline{EN\,V_{PP}}$, V_{PP1} - V_{PP6}, $\overline{EN\,V_{CC}}$, V_{cc1} - V_{cc3}, G_0 - G_2, OUT_0, OUT_1.

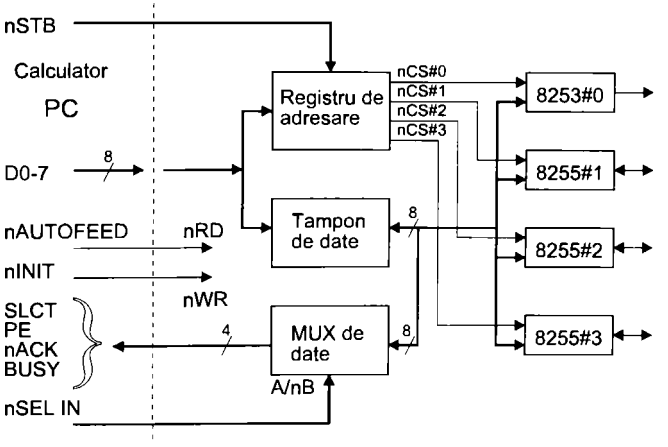


Figura 5.15. Schema bloc a programatorului realizat.

Sunt disponibile 24 de linii bidirecționale și 16 linii de ieșire, toate de uz general; opt linii de comandă (SA0- SA7), două linii de intrare (IN0 și IN1) și 4 linii nefolosite PC4-7, 8255#2, adică un total de 54 de semnale programabile, în afara liniilor folosite pentru comanda surselor de tensiune programabile sau a timer-ului 8253.

Programatorul este integral comandat prin intermediul portului paralel. Circuitele PPI 8255 sunt programate conform operației concrete de efectuat (citire, programare, verificare). Un program separat a fost pus la punct pentru dezasamblarea conținutului PAL-urilor cu 20 de pini.

5.4.2. SURSE PROGRAMABILE DE TENSIUNE

Studiul atent al comportării structurilor logice programabile (dar și al circuitelor EPROM, EEPROM) în timpul programării conduce la necesitatea a două clase de surse de tensiune programabile:

a) o sursă de tensiune de valoare relativ mare, destinată programării circuitului;
b) o sursă de tensiune de valoare relativ mică, destinată alimentării circuitului.

A. Sursa *eVpp*

Este destinată generării tensiunii de programare a circuitului; din analiza foilor de catalog pentru circuitele actuale disponibile pe piață rezultă o gamă de tensiuni de programare între 11 V - 25 V. Pentru un increment de 0,25 V rezultă că sunt suficienți 6 biți pentru codificarea numerică a tensiunii programate.

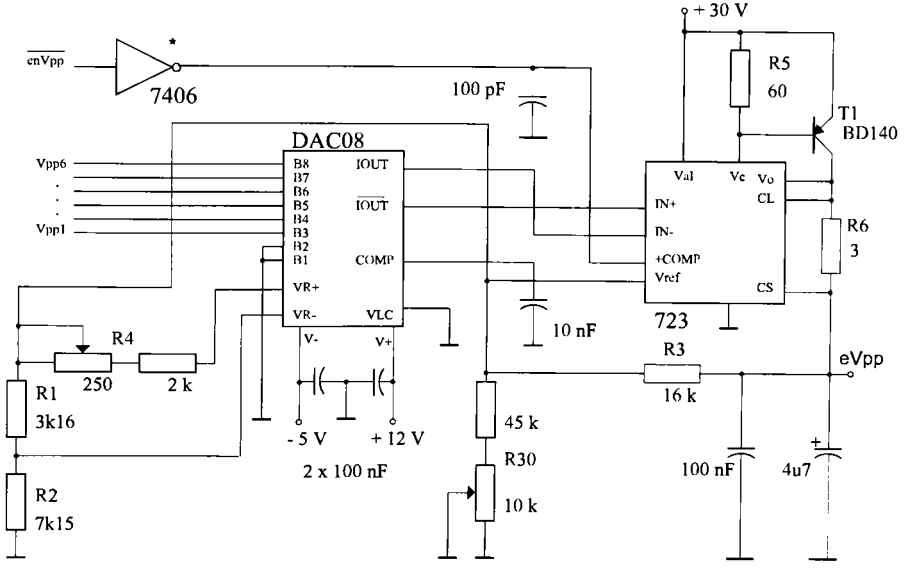


Figura 5.16. Sursa de tensiune programabilă *eVpp*.

Varianta (originală) pentru sursa de tensiune propusă este prezentată în figura 5.16. Circuitul 723 este folosit ca generator de tensiune de referință pentru DAC 08, ca amplificator de eroare și ca driver pentru tranzistorul de putere T_1. Sursa poate fi inhibată dacă $\overline{en\,V_{PP}} = 1$.

Notând cu I curentul de ieșire din DAC 08 corespunzător unei combinații binare oarecare (V_{PP1} ... V_{PP6}), cu I_0 treapta unitară de curent corespunzătoare la 1 LSb, se poate scrie expresia tensiunii de ieșire:

$$V_{PP} = V_{REF}\left(1+\frac{R_3}{R_{30}}\right) + I_0 R_3 \text{. În plus } I_0 R_3 = 0{,}25\text{ V}. \qquad (5.1)$$

Alegând I_{FS} = 1 mA, rezultă I_0 = 4/256 mA și R_3 = 16 kΩ. Se obține:

$$V_{REF}\left(\frac{R_3}{R_{30}}+1\right) = 9{,}5\text{ V}. \qquad (5.2)$$

Valorile semireglabilelor au fost alese astfel încât să fie compensată valoarea imprecisă a V_{REF} pentru 723 (6,8 ÷ 7,6 V). Procedura de reglare a sursei constă din următoarele etape:

a) se comandă V_{PP1} =...= V_{PP6} = $\overline{EN\,V_{PP}}$ = 0;
b) din R_{30} se reglează $eVpp$ = 9,5 V;
c) se comandă V_{PP1} =...= V_{PP6} = 1;
d) din R_2 se reglează $eVpp$ = 25,25 V.

B. Sursa *eVcc*

La prima vedere, o structură ca în figura 5.17 (propusă în [Ristea83], pagina 206) satisface necesitățile, ținând cont de domeniul și numărul de valori necesare tensiunilor de alimentare mult mai reduse decât la *eVpp*. Schema necesită două circuite integrate (minim) și prezintă trei inconveniente majore:

a) reglarea este dificilă, pentru n valori ale *eVcc* fiind necesari n + 1 semireglabili;
b) pentru n tensiuni comandate sunt necesare n linii de comandă, reducerea acestui număr de linii implicând utilizarea unui decodificator;
c) *eVcc* nu poate fi inhibată (doar redusă la 1,2 V).

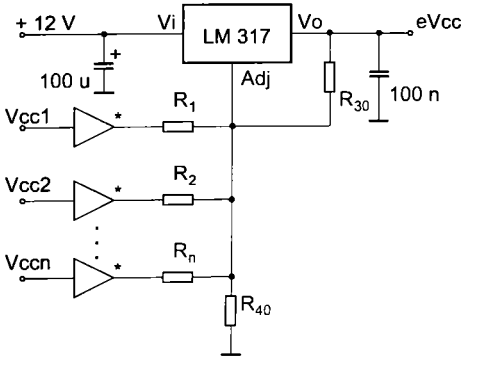


Figura 5.17. Propunere de sursă *eVcc*.

De aceea se propune o altă structură pentru sursa programabilă *eVcc*, (figura 5.18). În figura 5.19. este prezentată schema simplificată pentru sursa *eVcc*. După echivalarea Thevenin, rezultă:

$$V_0 = V_1\left(1+\frac{R_3}{R_{30}}\right) + I_0 R_3 . \qquad (5.3)$$

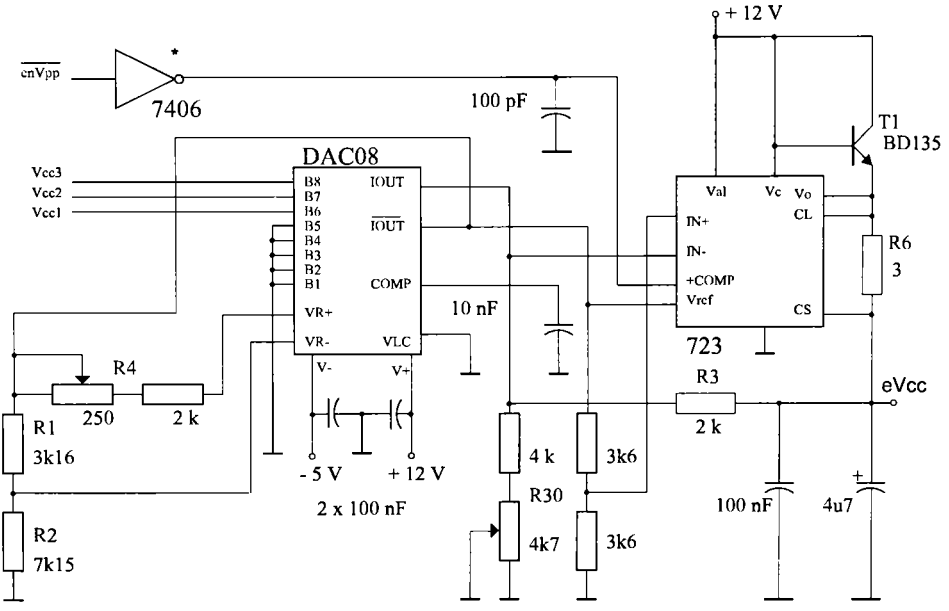


Figura 5.18. Sursa de tensiune programabilă *eVcc*.

Alegând (cu R_4) curentul capăt de scală pentru DAC 08, I_{FS} = 1 mA, rezultă incrementul de curent (pentru această schemă):

$$I_0 = \frac{32}{256} = \frac{1}{8}\text{mA} \text{ şi } I_0 R_3 = 0{,}25\text{ V}, \text{ adică } R_3 = 2\text{ k}\Omega,$$

şi impunând $eVcc_{minim}$ = 4,75 V, se obţin celelalte elemente ale schemei.

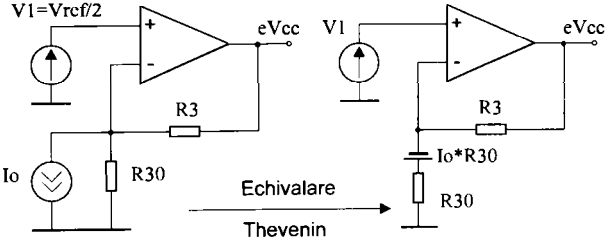


Figura 5.19. Schema simplificată pentru sursa *eVcc*.

Ansamblul (DAC 08 + LM 723) din figurile 5.16 şi 5.18 poate fi alimentat de la o singură sursă [Bodea85]. În acest caz este însă necesară o translatare de nivel cu rezistenţe şi diode (câte 2 rezistenţe şi 1 diodă pentru fiecare bit comandat [MSb ... LSb]). De aceea am preferat varianta cu 2 surse de alimentare. Verificarea condiţiilor de bună funcţionare pentru DAC 08, conduce la legarea pinului I_0 (nefolosit) la V_{REF} (723), iar pentru tensiunea de alimentare pozitivă (în cazul sursei *eVpp*) la o valoare mai mare de 10 V. Prin aceasta se elimină funcţionarea la o tensiune de alimentare aproape de valoarea limită absolută (36 V).

5.4.3. ELECTRONICA DE PIN

Este destinată multiplexării tensiunilor relativ ridicate destinate programării (în general 11 V - 25 V) cu nivelurile TTL obişnuite LOW şi HIGH. Curentul la programare variază între 30 mA (circuite CMOS) şi maxim 200 mA (PAL).

Pentru circuitele de tip EPROM şi EPLD-uri cu structură programabilă tot de tip EPROM, am implementat următoarea schemă pentru electronica de pin. Montajul din figura 5.20 *nu* protejează electronica de pin la o comandă greşită software.

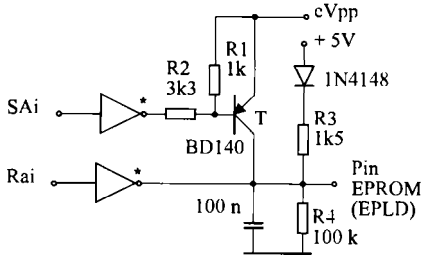


Figura 5.20. Electronica de pin de tip EPROM.

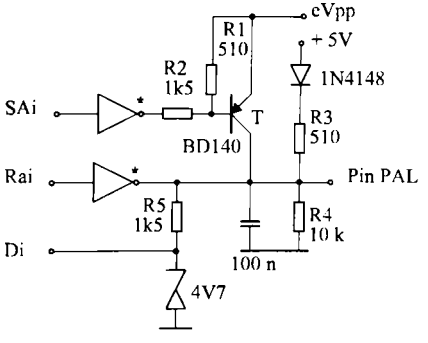


Figura 5.21. Electronica de pin de tip SLP.

Liniile de comandă de validare a tensiunii de programare (*eVpp*) SA_i şi de forţare nivel zero (RA_i) sunt active JOS. Software-ul trebuie să asigure interdicţia combinaţiei (0,0) pe cele 2 intrări.

La circuitele PAL, pinul programat *trebuie şi citit*; de aceea a fost prevăzut un circuit suplimentar de limitare cu diodă Zener (figura 5.21). Acest circuit poate fi folosit şi ca electronică de pin în scopuri de testare a circuitelor numerice.

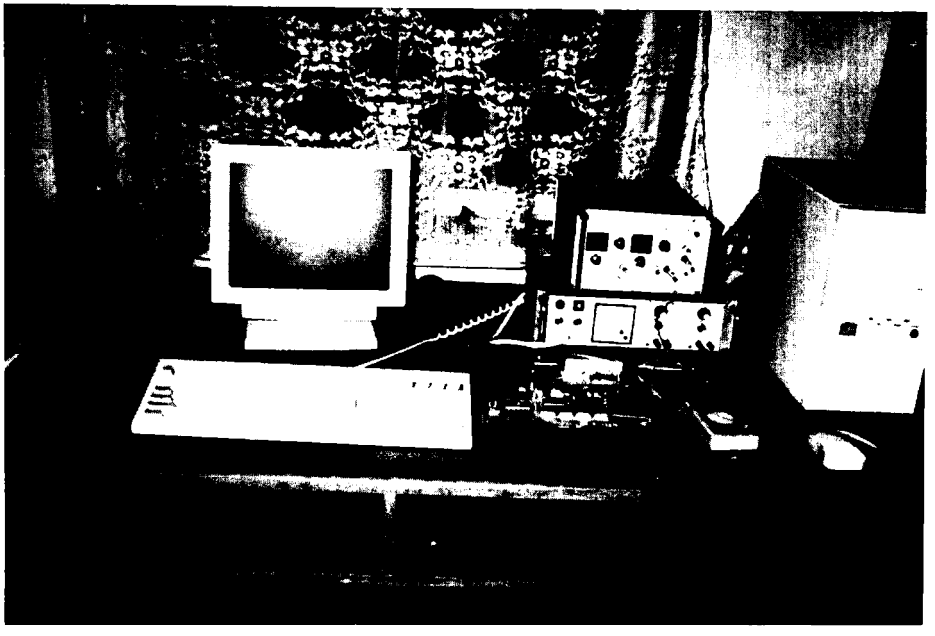


Figura 5.22. Programatorul realizat - vedere de ansamblu.

5.5. SOFTWARE PENTRU PROGRAMATOR

Este unanim recunoscută în zilele noastre ponderea deosebită a codului scris pentru interfaţa grafică utilizator, raportată la lungimea codului util scopului ales. În cadrul elaborării acestei lucrări au fost implementate mai multe module de program, în principal aliniate la trei direcţii majore:

- module de program de autotestare programator;
- implementarea interfeţei grafice utilizator;
- implemenatrea algoritmilor de programare.

5.5.1. AUTOTESTAREA PROGRAMATORULUI

Considerând complexitatea unui programator modern, autotestarea apare ca o soluţie naturală atât în perioada de punere în funcţiune, cât şi în exploatare. Verificarea este eficientă dacă este în acelaşi timp cuprinzătoare şi uşor de utilizat. În perioada de punere în funcţiune a programatorului am scris mai multe proceduri de testare ale principalelor blocuri ale programato-

rului. După încheierea acestei etape, testele au fost rafinate, pentru a putea servi scopului inițial - determinarea funcționării dispozitivului și a încadrării în specificațiile tehnice. Programul astfel rezultat a fost astfel conceput încât să nu necesite pentru rulare un PC performant. Toate mesajele sunt implementate în mod text.

După afișarea logo-ului de introducere, programul determină în procedura det_LPTi adresa fizică a portului de imprimantă, utilizată în continuare. În procedura meniu_principal au fost incluse zece teste reprezentative. Este important de subliniat că absența firmware-ului din programator nu conduce la complicații suplimentare la upgrade-ul softului de testare. Se obține astfel eficient testarea hardware-ului prin software, integral comandat de PC. Cele zece teste sunt:

1. Test port de date;
2. Test port de comandă;
3. Test general magistrala interna de date;
4. Test citire magistrala internă de date și MUX;
5. Test 8253 funcționare ca astabil;
6. Test 8253 funcționare ca monostabil;
7. Test programare 8255;
8. Test sursă de tensiune programabilă eVpp;
9. Test sursa de tensiune proframabilă eVcc;
10. Test citire circuit programat (EPROM sau SLP).

Primele două teste se bazează pe emisia octeților 00 și 0FFH pe portul de date, respectiv de comandă. Magistrala internă de date este verificată prin alternarea clasică a octeților 55H și 0AAH.

Pentru a depista existența unui scurtcircuit în montaj sau o componentă defectă, testul 4, de tip "walking zero", inspectează multiplexorul și magistrala internă de date. Bitul pe care s-a determinat o neconcordanță între valoarea citită efectiv și cea calculată este afișat cu mesaj de eroare.

Testele 5, 6, 7 verifică dacă circuitele 8253 și 8255 pot fi programate, iar pentru 8253 dacă acesta funcționează ca monostabil, respectiv ca astabil.

Testul 8 verifică sursa eVpp între 9,5 V și 25,25 V, în incremenți de 0,25 V. Sursa eVcc este verificată între 4,75 V și 6,5 V (de asemenea în incremenți de 0,25 V) în cadrul testului 9.

Ultimul test permite verificarea parțială a electronicii de pin prin citirea conținutului unui EPROM sau SLP și scrierea sa într-un fișier. Programul de test autocomentat este listat în Anexa 1. Nu am considerat necesare explicații suplimentare referitoare la acest subiect deoarece rutinele de autotestare sunt comentate intensiv, tocmai pentru a asigura o înțelegere cât mai clară din partea utilizatorului.

5.5.2. INTERFAȚA GRAFICĂ UTILIZATOR

Principalele deziderate urmărite la realizarea interfeței grafice au fost: ușurința utilizării, și compatibilitatea cu modul text și Windows. A fost astfel creat un mediu grafic care poate rula direct sub DOS sau Windows (figura 5.23), acționabil cu mouse-ul, tastatura sau direct prin shortcut-uri. Prezența unui tabel ASCII de simboluri și a unui calculator simplu sporește versatilitatea interfeței implementate.

5.5.3. ALGORITMI DE PROGRAMARE IMPLEMENTAȚI

Pentru circuitele EPROM au fost implementați practic toți algoritmii cunoscuți de programare: standard (n * 50 ms), rapid (m * n * 1 ms) și fast (p * n * 150 μs). Prezența timer-ului 8253 este o dată în plus justificată de temporizările reduse (și stabile) ce trebuiau implementate, comparabile cu rata de transmisie pe portul paralel, imposibil de realizat soft.

Diversitatea extremă a PLD-urilor comparate cu memoriile EPROM a făcut posibilă doar implementarea algoritmilor de programare prezenți în literatură (PAL, GAL, EPxxx). Structura hardware a programatorului nu limitează extensiile software de algoritmi de programare, pe măsură ce aceștia vor deveni cunoscuți.

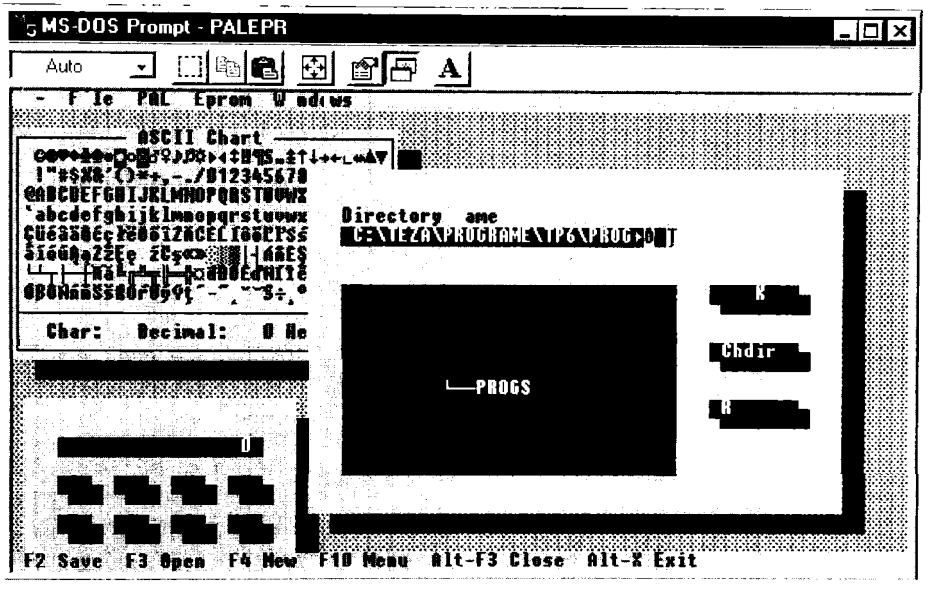


Figura 5.23. Ecranul interfeței grafice.

5.6. COMPARAȚIE ÎNTRE PROGRAMATOARELE PREZENTATE

5.6.1. SINTEZA CARACTERISTICILOR PROGRAMATORULUI REALIZAT

Circuite programabile:

- EPROM, EEPROM, Flash EPROMs până la 40 pini, PROM bipolare;
- Capsule DIP - alte tipuri numai prin adaptor.

Interfața software

- Interfață grafică tip text ce poate rula și sub Windows;
- Editor incorporat pentru memorii EPROM, cu date hexa sau ASCII;
- Tratează standardele JEDEC, Intel și formatele binare.

Accesorii standard:

- interfața software;
- cablul paralel;
- sursa de tensiune de alimentare externă.

Tabelul 5.5

Comparație între programatoarele prezentate

Caracteristică	ChipLab	M4016	Progr. propus
Număr de circuite programate simultan	1 -	16 + +	1 -
Programare microcontrolere	DA + + +	DA + + +	NU - - -
Viteza de transmisie date	++++	+	+ + +
Interfață grafică Win	DA ++++	DA ++++	NU -
Alte capsule decât DIP	DA, cu adaptor	DA, cu adaptor	DA, cu adaptor
Testare CI numerice și memorii	DA	DA	DA
Prezența firmware	NU ++++	DA +	NU ++++
Preț	+++	++	+++++

5.7. CONCLUZII

Acest capitol demonstrează posibilitatea realizării unui programator ieftin și fiabil, utilizând componente accesibile. Este analizată în detaliu comunicația dintre programator și calculator, cu avantajele și dezavantajele fiecărei metode implicate. Această analiză permite formularea clară a avantajelor utilizării portului paralel, metodă propusă de autor încă din anul 1991 și care între timp s-a generalizat. După studiul unor produse similare în domeniu sunt înfățișate aspectele practice legate de programator. Alegerea unei variante de programator fără firmware este o soluție elegantă, care nu necesită rescrierea unor rutine pentru un procesor intern. Studiul atent al surselor programabile de tensiune demonstrează că se pot obține performanțe excelente folosind circuite nepretențioase, în scheme atent gândite. Schemele propuse pentru electronica de pin permit interfațarea finală atât pentru circuite de tip EPROM, cât și pentru SLP.

Componentă modernă și esențială a electronicii moderne, testarea constituie pentru programatorul universal realizat un capitol distinct în eforturile autorului. Rutinele de test implementate oferă o modalitate eficientă de punere la punct a prototipului realizat, dar și o modalitate convenabilă de automatizare a verificărilor în cadrul unei eventuale fabricări în serie.

Implementarea interfeței grafice utilizator a reprezentat un efort cel puțin la fel de mare ca punerea la punct a hardware-ului. Cele aproximativ 7000 de lini de cod ale programului sursă au format în final un mediu ce poate rula sub sistemele de operare majore existente (DOS, Windows 3.x, Windows 95, Windows NT), simplu de utilizat și de extins.

Sintetizând afirmațiile de mai sus și ținând cont de particularitățile prezentate în tabelul 5.5, rezultă câteva avantaje pentru Aprommer față de celelalte programatoare discutate, cum ar fi:

- o portabilitate totală şi o viteză mare de transmisie date datorate comunicării exclusive pe portul paralel;
- este uşor de manipulat, chiar şi de personal având o calificare medie;
- implementează o schemă hardware simplă, cu circuite de larg consum, de mare fiabilitate, la preţ redus;
- absenţa oricărui firmware implică doar upgrade-uri software şi elimină problemele de compatibilitate, versiune.

Ultimele două avantaje sunt esenţiale pentru soluţia oferită. Deşi realizat ca prototip, implementarea programatorului Aprommer a permis autorului pe lângă acumularea inerentă de cunoştinţe şi experienţă, desprinderea unor concluzii referitoare la aceste echipamente. Este evident interesul pentru dezvoltarea ulterioară, descrisă prin următoarele etape:

- extinderea hardware, prin realizarea de adaptoare pentru un număr cât mai mare de SLP;
- extinderea software, prin adăugarea de noi algoritmi de programare şi eventual a unor rutine de testare a circuitelor numerice şi memoriilor semiconductoare, domeniu în care autorul are de asemenea experienţă şi realizări [Gontean93a], [Gontean94b].

5.8. REFERINŢE BIBLIOGRAFICE

Dacă referinţele bibliografice pentru SLP sunt în general sărace, pentru programarea SLP ele sunt remarcabil de firave. Trecută este epoca cataloagelor cu memorii EPROM, bine documentate, după care fiecare îşi putea construi un programator. Firmele producătoare păstrează detaliile de programare, permiţând accesul la informaţia detaliată numai anumitor companii producătoare de dispozitive de programare.

Programatoare de SLP relativ simple se găsesc în reviste de specialitate, de exemplu [Elektor94] sau în [Lattice88,94].

[Miclăuş94] prezintă un programator dedicat realizat în cadrul unui proiect de diplomă coordonat de autor.

CAPITOLUL 6

CONTRIBUȚII ORIGINALE ȘI CONCLUZII FINALE

Gândită ca o contribuție la strategia de aliniere creatoare și perseverentă a nivelului tehnic autohton la posibilitățile actuale de realizare a echipamentelor electronice numerice, teza propusă reprezintă o noutate atât din punct de vedere al domeniului abordat cât și al soluțiilor oferite. Domeniul abordat al *Structurilor Logice Programabile* a cunoscut o dezvoltare spectaculoasă pe plan mondial în ultimul deceniu, perioadă ce coincide în mare cu cea de pregătire la doctorat. În acest interval cunoștințele acumulate de autor s-au sedimentat, experiența a crescut, permițând obținerea de rezultate originale și oferind baza pentru redactarea acestei lucrări. În scopul declarat al obținerii de rezultate performante, autorul s-a străduit să aducă contribuții originale atât în partea teoretică prin clasificări, sinteze și comparații, introducând o metodă originală de predicție a rutabilității cât și în cea practică, verificând rezultatele prezise teoretic, implementând în trei variante diferite un sintetizor numeric de frecvență și prezentând două variante de programatoare de SLP.

6.1. CONTRIBUȚII ORIGINALE

Pe parcursul celor cinci capitole ale tezei de doctorat se evidențiază următoarele contribuții originale mai importante:

CAPITOLUL 1

1.1. În paragraful 1.1 au fost identificate pe baza unui exemplu simplu abordările clasice de implementare a unui sistem numeric: hardware și software. Deși nu apare explicit în literatura de specialitate, este posibilă și o a treia variantă, respectiv *logica programabilă*, care îmbină avantajele majore ale abordărilor prezentate anterior. Importanța acestei noi posibilități este cu atât mai mare, cu cât permite implementări directe utilizând circuite și echipamente ieftine, aspect deosebit de important pentru țara noastră. Comparația efectuată de autor în subparagraful 1.1.4 prezintă sintetic aceste avantaje, încheindu-se cu estimarea că în mai puțin de zece ani, logica programabilă va domina cel puțin segmentul de piață al aplicațiilor la cerere.

1.2. Studiul amănunțit al bibliografiei extinse în acest domeniu a permis identificarea în paragraful 1.2 a principalelor etape de dezvoltare ale circuitelor integrate numerice. Aceste etape, desemnate convențional la o durată de zece ani, au fost completate cu numirea acestui deceniu ca fiind *etapa SLP*, motivat de succesele înregistrate pe plan mondial de aceste circuite. Prezentarea sistematică a acestor etape vine să sprijine concluziile enunțate anterior și oferă o perspectivă a acestor evoluții și tendințe.

1.3. Redarea unitară, clar structurată, a tuturor metodelor de proiectare a sistemelor numerice, a permis reliefarea principalelor caracteristici ale fiecăreia și implicit analiza comparată cu evidențierea avantajelor și actualității în fiecare caz în parte.

1.4. Paragraful 1.3 debutează cu o succintă descriere a pachetului de programe LOG/iC, a cărui aprofundare a fost posibilă pe durata specializării efectuate de autor în Germania în 1995. Acest pachet este unul din cele mai perfecționate instrumente CAD de proiectare cu PLD, permițând dezvoltarea eficientă de aplicații. Prezentarea este continuată firesc cu metodele de descriere pentru un sistem numeric. Această sistematizare a condus la identificarea neajunsurilor diagramei Gajski-Kuhn și la propunerea extinderii ei spațiale.

1.5. Este de asemenea importantă concluzia de la sfârșitul subparagrafului 1.3.3, care indică superioritatea implementărilor dependente de tehnologie față de cele independente tehnologic.

1.6. Studiul metodei Mead-Conway conduce la observația că această metodă este aplicabilă la arhitecturile FPGA cu granularitate fină, domeniu al SLP extrem de nou și dinamic, pentru care nu există în prezent o rezolvare completă, în consens, pe plan mondial.

1.7. Propunerea de schemă de tact cu două faze, derivată din metoda Mead-Conway, a fost aplicată la realizarea unui analizor logic ieftin și performant, precum și a unui număr de lucrări originale pe această temă.

1.8. Clasificarea SLP care încheie primul capitol prezintă o importanță specială. Criteriile prezentate se transformă firesc pentru utilizator în jaloane fundamentale de selecție optimală a unui SLP potrivit aplicației dorite. Ea permite definirea sistemului de coordonate porți-bistabile din figura 1.10 și ordonarea SLP-urilor în zonele de aplicație din figura 1.11.

CAPITOLUL 2

2.1. Prezentarea sintetică a arhitecturii SLP de complexitate redusă.

2.2. Pentru SLP de complexitate redusă se evidențiază posibilitatea încadrării memoriilor PROM în cadrul mai larg al SLP.

2.3. Este propusă o clasificare a FPGA în funcție de arhitectura blocului logic, alegând drept criteriu de clasificare posibilitatea de implementare într-un singur bloc a unei funcții de trei variabile.

2.4. Prezentarea sintetică a principalelor tehnologii de programare, criteriu original de clasificare suplimentară a FPGA.

2.5. Decelarea și prezicerea unor evoluții viitoare pentru FPGA, dintre care câteva sunt descrise în continuare:

- 2.5.1. creșterea numărului de arhitecturi pentru FPGA;
- 2.5.2. dezvoltarea de FPGA orientate pe aplicație, în domenii cum ar fi transmisiile de date, DSP, AES, microprocesoare; interesantă aici este direcția prefigurată în [Gavrilescu97];
- 2.5.3. apariția de sisteme hardware și software de emulare, testare pentru FPGA provenind de la mai mulți producători;
- 2.5.4. dezvoltarea de arhitecturi de blocuri logice neomogene. Acestea promit obținerea unui compromis mai bun între suprafața ocupată și performanțele obținute;

2.5.5. proliferarea variantelor de rutare segmentate, care oferă avantaje majore faţă de variantele nesegmentate, cu doar una sau două lungimi posibile pentru segmente;

2.5.6. succesul FPGA în domeniul circuitelor numerice va fi reprodus în domeniul circuitelor analogice de FPAA (Field Programmable Analog Array).

2.6. Elaborarea unui model original de structură blocului logic dintr-un SLP (figura 2.19), utilizat ca bază de calcul pentru determinarea ariei totale ocupate de blocul logic.

2.7. Simularea dependenţei ariei blocului logic de valoarea lui k în diferite variante de implementare.

2.8. S-a demonstrat existenţa unui minim pentru $k = 3$, respectiv 4.

2.9. Cazul studiat (referitor la FPGA simetrice la care blocul logic este realizat cu un LUT cu k intrări şi o ieşire) poate fi extins în primul rând la cazul LUT cu ieşiri multiple, respectiv la blocuri logice alcătuite din mai multe LUT.

2.10. Evaluarea celor mai performante arhitecturi pentru FPGA, cu indicarea clară a celei mai bune opţiuni în fiecare caz.

2.11. Elaborarea unui montaj experimental eficient ce poate fi utilizat la studiul comportării metastabile a oricărui sistem numeric. Pe baza acestui montaj au fost măsurate alături de circuitele TTL, o serie de SLP pe care le-am avut la dispoziţie.

2.12. Verificarea afirmaţiei că tehnologia de fabricaţie prezintă elementul primordial în descrierea comportării metastabile a unui circuit, relevând efectele minore obţinute prin optimizări de proiectare.

2.13. Seria MAX EPM5000 prezintă caracteristici de metastabilitate comparabile cu cea a bistabilelor ALS până la FAST. Seria EP prezintă o metastabilitate mai bună decât cea a bistabilelor LS.

CAPITOLUL 3

3.1. Elaborarea unui set coerent de definiţii pentru principalele mărimi ce caracterizează procesul de rutare.

3.2. Studiul pertinent al arhitecturii Xilinx, ca fiind cea mai adecvată elaborării modelului de structură pentru calculul probabilităţii de rutare.

3.3. Demonstrarea pe baze fizice a presupunerii respectării legii de distribuţie Poisson pentru densitatea pistelor din canalul de rutare.

3.4. Idea prezicerii rutabilităţii înaintea iniţierii procesului derutare propriu-zis.

3.5. Studiul dependenţei rutabilităţii de gradul de flexibilitate Fc şi prezentarea convenabilă a rezultatelor (figura 3.9).

3.6. Studiul dependenţei rutabilităţii de gradul de flexibilitate Fs şi prezentarea convenabilă a rezultatelor (figura 3.10).

3.7. Reprezentarea tridimensională a rutabilităţii (figura 3.11).

3.8. Studiul numărului minim de piste într-un canal pentru o rutare completă (pornind de la cazuri concrete (tabelul 3.1).

3.9. Demonstrarea superiorităţii performanţelor de rutare pentru Fc cuprins în intervalul 0,6...1 .W, cu 0,8 .W ca valoarea cea mai convenabilă.

3.10. Studiul erorii relative de rutabilitate.

3.11. Demonstrarea faptului că modelul structural este unul pesimist (cu alte cuvinte rutabilitatea prezisă poate fi *sigur* atinsă).

CAPITOLUL 4

4.1. Prezentarea concisă a principalelor metode de sinteză de frecvenţă şi a criteriilor de comparare a diferitelor variante de implementare a sintetizoarelor de frecvenţă.
4.2. Evidenţierea metodelor de reducere a dimensiunii memoriei ROM din componenţa LUT, dezvoltând o idee prezentată de autor la o sesiune de comunicări ştiinţifice studenţeşti.
4.3. Elaborarea unei scheme de sintetizor numeric de frecvenţă cu inversoare (figura 4.10), subiect abordat foarte puţin în literatură.
4.4. Elaborarea unui studiu amănunţit privind arhitectura acumulatorului, componenta critică a sintetizorului. Acest studiu demonstrează avantajele metodei pipe-line de implementare, autorul propunând trei variante diferite de implementare (figurile 4.11, 4.12, 4.13).
4.5. Studiul amănunţit al comportării arhitecturii din figura 4.13 a permis simplificarea resurselor logice implicate cu 40%, rezultând varianta din figura 4.14.
4.6. Studiul arhitecturii blocului logic al familiei Xilinx XC300 (figura 4.15) a permis demonstrarea pretării acestei arhitecturi la o implementare de sintetizor de mare viteză (figura 4.16).
4.7. Demonstrarea similitudinilor dintre layout-ul pentru un circuit proiectat cu programul profesional Solo 1400 şi cel al unui FPGA simetric, pentru validarea concluziilor rezultate în urma rutării.

Trei din variantele de sintetizor de frecvenţă prezentate succint în acest capitol au fost implementate, testate şi rutate de autor cu ajutorul programului Solo 1400 pe reţeaua Sun a Universităţii Preston din Anglia. Lay-out-ul final pentru aceste variante este redat în figurile 4.18-4.20, iar un detaliu de rutare în figura 4.21. Toate aceste variante, implementări, verificări şi rutări sunt bineînţeles, complet originale.

4.8. Implementarea celor trei variante de sintetizor menţionate anterior.
4.9. Elaborarea unei sinteze comparative pentru variantele de implementare prezentate, care reflectă performanţele de viteză ale variantei complet pipe-line îmbunătăţite din figura 4.14.
4.10. Demonstrarea pe un caz concret a afirmaţiei din capitolul 1, şi anume că o abordare *dependentă de tehnologie* atent gândită şi rafinată de autor, poate oferi performanţe excelente. Implementarea schemei din figura 4.14 oferă o viteză maximă de funcţionare cu 67% mai mare decât cea mai performantă variantă clasică, la un consum global de porţi crescut cu doar 46%. Acesta este un rezultat absolut remarcabil.
4.11. Deducerea dependenţei întârzierii normate în funcţie de temperatură şi tensiunea de alimentare (tabelul 4.2) şi redarea rezultatului într-o formă convenabilă de prezentare (figura 4.22).

CAPITOLUL 5

5.1. Este elaborată o clasificare unitară a programatoarelor, după patru criterii sintetice. Această clasificare permite decelarea principalelor atribute pe care trebuie să le prezinte un programator pentru a corespunde dorinţei utilizatorului. Clasificarea este

urmată de explicații sintetice privind diversele opțiuni posibile, inclusiv din punct de vedere al costurilor implicate.

5.2. Este analizată în detaliu comunicația dintre programator și calculator, cu avantajele și dezavantajele fiecărei metode implicate. Această analiză pertinentă oferă posibilitatea evidențierii avantajelor oferite de comunicația pe portul paralel, metodă propusă de autor încă din 1991 [Gontean91] și care între timp a devenit universală.

5.3. Indicarea sintetică a aspectelor principale privind formatele de interschimbare a informației între programator și calculator. Aceste formate standardizate oferă o platformă comună fără de care nu se poate concepe o proiectare cu SLP.

5.4. O analiză critică a câtorva tipuri de programatoare existente pe piață permite sintetizarea principalelor lor caracteristici și o bază pentru comparațiile ulterioare.

În capitolul 5 sunt prezentate două programatoare originale, realizate de utilizator. Ambele se conectează pe portul paralel la un calculator gazdă. În ambele cazuri portul paralel este utilizat în modul SPP, prin aceasta fiind posibilă comunicația cu orice tip de PC. Primul programator este unul dedicat, al doilea fiind un programator universal.

5.5. Elaborarea unei scheme electrice pentru un programator dedicat (figurile 5.11-5.13). O vedere de ansamblu a programatorului realizat este redată în figura 5.14.

5.6. Propunerea de alimentare a programatorului direct de la conectorul de tastatură. Această opțiune oferă două avantaje majore: simplitate și cost redus.

5.7. Elaborarea unei scheme bloc pentru un programator universal care să conțină elemente accesibile, fiabile și ușor extensibile (figura 5.15).

5.8. Alegerea unei variante de programator fără firmware, care nu necesită modernizări hardware sau rescrierea rutinelor pentru un procesor intern. Pe infrastructura implementată, orice modificare se realizează simplu și rapid, în modulul de programare de pe PC.

5.9. Rezolvarea problemelor de comunicație bidirecțională pe portul paralel, care oferă doar 5 linii de intrare (figurile 5.6, 5.7).

5.10. Studiul atent al surselor programabile de tensiune necesare demonstrează necesitatea a două clase de asemenea surse: *surse de programare*, notate eVpp, de tensiune relativ ridicată (în intervalul 11-25 V) și *surse de alimentare*, notate eVcc, de tensiune relativ scăzută (în intervalul 4,75 V-6,25 V).

5.11. Folosirea combinației DAC08-ßA723 pentru realizarea unei surse programabile de tensiune. Pe lângă aspectul preț, care nu este de loc de neglijat se obțin o serie de avantaje, cum ar fi: stabilitatea termică a schemei (ßA723 este compensat termic), simplitatea schemei (amplificatorul operațional din ßA723 este folosit cu DAC08 pentru conversia curent-tensiune. Schema rezultată este protejată nativ la scurtcircuit la ieșire, și poate fi oricând inhibată folosind intrarea $\overline{enVpp}$. Performanțe superioare la un preț mai mic sunt dificil de obținut chiar și astăzi, schema propusă inițial în [Gontean91] reprezentând rodul unui efort de combinare a unei idei inovatoare cu circuite integrate industriale.

5.12. Evidențierea neajunsurilor unui aranjament pentru o sursă de tensiune de alimentare ca cel propus curent în literatură (figura 5.17).

5.13. Imaginarea unei scheme electrice care să permită în aceleași condiții de calitate evidențiate la punctul 5.10, obținerea unei tensiuni de ieșire inferioare tensiunii de referință interne pentru ßA723. Schema originală propusă în figura 5.18 este validată de transformarea Thévénin din figura 5.19.

5.14. Verificarea experimentală a relațiilor 5.1-5.3 a validat schemele discutate anterior. Experiența de utilizare a demonstrat suplimentar că se pot obține rezultate excepționale în scheme simple, atent gândite.
5.15. Elaborarea a două variante pentru electronica de pin. Prima variantă este destinată circuitelor de tip EPROM, iar cea de-a doua este destinată SLP. Propunerea de a folosi montajul din figura 5.21 ca electronică de pin standardizată și în scopul testării circuitelor numerice.
5.16. Evidențierea și implementarea a trei module software necesare unei bune funcționări a programatorului: module de autotestare, interfața grafică utilizator și modulele corespunzătoare algoritmilor de programare.
5.17. Elaborarea ordinogramelor și a modulelor de program necesare autotestării programatorului.
5.18. Utilizarea unor metode de testare specifice memoriilor semiconductoare pentru validarea magistralei interne de date a programatorului.
5.19. Includerea unui test complex, de citire EPROM sau SLP, prin care se poate opera integral programatorul fără utilizarea interfeței grafice.
5.20. Proiectarea și implementarea interfeței grafice utilizator. Interfața realizată funcționând sub modurile DOS, Windows 3.x, Windows 95, Windows NT nu restrânge sub nici un fel generalitatea programatorului proiectat.
5.21. Proiectarea programatorului pentru a utiliza cât mai puține resurse hardware și software din PC-ul gazdă.
5.22. Prezentarea sintetică a unei comparații pertinente între programatoarele redate în capitolul 5.

6.2. CONCLUZII FINALE

Teza de doctorat abordează aspectele moderne ale proiectării optimale cu structuri logice programabile. În sintezele teoretice efectuate, clasificările prezentate, în metoda utilizată dar și în implementările de sintetizor și programatoare realizate, autorul aduce o serie de contribuții originale. Preocupările autorului în acest domeniu sunt în consens cu reorientarea actuală spre tehnologii performante, ce cuprind un grad ridicat de inteligență la un consum material redus. Principalele obiective ale cercetării urmărite în lucrare au vizat în principal următoarele aspecte:

- Studiul efectului arhitecturii blocului logic asupra performanțelor SLP;
- Elaborarea unui model de conectare pentru FPGA pentru calculul ariei ocupate;
- Studiul comportării metastabile a SLP și elaborarea unui montaj de evaluare cantitativă a acestei comportări;
- Elaborarea unui model pentru studiul rutării în FPGA simetrice;
- Verificarea pe proiecte practice rutate a presupunerii distribuției Poisson pentru densitatea canalului;
- Implementarea unui sintetizor de frecvență în mai multe variante pentru a putea compara performanțele obținute;
- Studierea celei mai potrivite arhitecturi a blocului logic pentru implementarea unui sintetizor de frecvență într-un FPGA;
- Realizarea unui programator universal de SLP, hardware și software, performant și ieftin.

Rezultatele cele mai importante ale tezei au la bază 17 referinţe bibliografice publicate de autor, dintre care se remarcă: 2 cărţi tehnice, 2 lucrări ştiinţifice prezentate şi publicate în străinătate şi 13 lucrări ştiinţifice publicate la manifestări naţionale şi internaţionale din ţară.

În continuare va fi prezentată succint reflectarea principalelor obiective ale cercetării în lucrările autorului, prezentate cronologic în continuare:

1. [Gontean86] prezintă un set de rutine utilizate pentru implementarea unei biblioteci matematice rapide în sisteme cu microprocesor. Proprietăţile de simetrie şi periodicitate ale funcţiei sinus au făcut posibile reducerea considerabilă a dimensiunii tabelului de valori memorate pentru funcţia sinus, metodă analogă cu cea descrisă în capitolul 4.
2. [Gontean91] este primul referat redactat în cadrul pregătirii la doctorat şi reprezintă sinteza stadiului de dezvoltare a SLP la acel moment, împreună cu tendinţele din acest domeniu.
3. [Gontean92a], al doilea referat redactat în cadrul pregătirii la doctorat pune accentul asupra principalelor metode moderne de analiză şi proiectare a sistemelor numerice folosind structuri logice programabile. Cu acest prilej se prezintă şi programatorul universal dezvoltat de autor.
4. [Gontean92b] prezintă noile modalităţi de proiectare cu SLP ce au apărut în ultimul deceniu prin dezvoltarea SLP.
5. [Gontean92c] este o continuare la lucrarea anterioară şi prezintă aspecte specifice, mai puţin cunoscute ale proiectării cu SLP şi posibilităţile noi oferite de aceste circuite.
6. [Gontean93a] este o sinteză a principalelor metode de testare pentru structurile logice programabile.
7. [Gontean93b] prezintă montajul experimental de determinare a caracteristicilor de metastabilitate a circuitelor numerice.
8. [Demian94] introduce posibilitatea reducerii costurilor de producţie pentru un analizor logic utilizând SLP. Rezultat al unui contract încheiat cu firma AXA SRL, la proiectarea acestui analizor s-a folosit în premieră comanda schemei pe ambele fronturi ale semnalului de tact.
9. [Gontean94a] continuă preocupările autorului în domeniul SLP, introducând elementele esenţiale ale standardului IEEE 1149.1 şi modalităţile de implementare în FPGA.
10. [Gontean94b] este o sinteză a metodelor de testare pe frontieră cu aplicaţii în cazul FPGA.
11. [Gontean94c] tratează aspectele metastabilităţii SLP şi prezintă rezultatele experimentale ale comportării metastabile ale SLP în raport cu circuitele TTL standard.
12. [Gontean96a] tratează o modalitate originală de implementare a unui controler de memorie RAM cu FPGA, utilizând registre de deplasare cu reacţie liniară, prezentate în [Mureşan96]
13. [Gontean96b] este sinteza prezentată în Colectiv de Catedră a variantelor de implementare a unui sintetizor numeric de frecvenţă, efectuată cu ocazia bursei la Universitatea Central Lancashire din Preston, Anglia.
14. [Mureşan96] este o carte clasică de aplicaţii de circuite numerice, la care subiectul SLP este tratat de autor într-un capitol separat şi o anexă.
15. [Gontean97] este cartea autorului în domeniul SLP. Conţine 8 capitole şi o anexă, (1. Noţiuni fundamentale asupra SLP, 2. Proiectarea unui sistem numeric,

3. SLP actuale, 4. Proiectarea cu SLP, 5. Software utilizat pentru proiectarea cu SLP, Programarea SLP. Programatoare, 7. Aplicații, 8. Analiza comparată a costurilor în producerea sistemelor digitale). Această carte este prima ce tratează integral acest subiect în literatura română, iar contribuția autorului la elaborarea ei este de 88%.

16. [Gontean98a] introduce modelul de structură pentru FPGA simetric; este publicată la prestigioasa manifestare internațională PDS'98 în Polonia.
17. [Gontean98b] prezintă sinteza realizărilor autorului în domeniul programatoarelor universale. Și această lucrare a fost publicată în străinătate, la Conferința Internațională PDS'98.

* * *

Teza de doctorat prezentată a fost definitivată în perioada 1994-1998, dar reprezintă rodul preocupărilor autorului în domeniul structurilor logice programabile pe durata ultimilor 8 ani.

Elaborarea tezei de doctorat a avut loc sub îndrumarea competentă și generoasă a conducătorului științific, dl. Prof. Dr. Ing. Tiberiu MUREȘAN, față de care autorul nutrește cele mai sincere sentimente de respect și apreciere. Pentru discuțiile utile, sfaturile competente, sugestiile novatoare și încurajările stimulatoare care au contribuit substanțial la finalizarea acestei teze îi sunt profund recunoscător.

ANEXA 1

PROGRAM DE AUTOTESTARE AL PROGRAMATORULUI Aprommer

```
program teste;
uses crt;

const
         inactiv   : Byte = $0D; {-A/B=0, -WR=1, -RD=1, STB=1}
var
         adr       : word;
         mask_adr  : byte;
         cw        : byte;   {cuvint de cda 8253}
         tw        : byte;   {durata temporizare 8253}
         sel       : byte;   {Selector 8255/8253}
         I         : byte;   {contor de uz general}
         Ch        : Char;   {Caracter citit de la tastatura}
         pd,pc,ps  : word;   {printer port date, comanda, stare}
         p         :^word;   {pointer catre portul imprimantei}
         x,y       : byte;   {folositi pentru scriere (x) si citire (y)}
         oldx,oldy : byte;   {back up pentru x si y}
         numar_8255: byte;
         port_8255 : byte;
         contor    : integer; {reface continutul contorului 8253}
         eVpp,eVcc : real;   {valoarea eVpp, eVcc}
         tip_E     : byte;
         nume_fis  : string;
         f         : file of byte;
         m         : word;
         lungime_E : word;

{    sel      | 8255#1 |  8255#2 | 8255#3 | 8253#0
     ------------------------------------------------
     Port A  |   $04  |   $08   |  $0C   |  $00
     Port B  |   $05  |   $09   |  $0D   |  $01
     Port C  |   $06  |   $0A   |  $0E   |  $02
     Cuv. cda|   $07  |   $0B   |  $0F   |  $03
}

procedure beep_scurt;
begin
     sound(1000);
     delay(50);
     nosound;
end;

procedure press_ESC;
begin
     GotoXY(1,25);
     write('                Apasati <ESC> pentru a iesi din program');
end;

procedure press_CR;
begin
     GotoXY(1,25);
     write('                Apasati <Enter> pentru a continua      ');
```

```
     repeat until ReadKey=#13;
end;

procedure press_space;
begin
     GotoXY(1,22);
     write('                    Apasati <Space> pentru a schimba
valoarea');
end;

procedure blank_space;
begin
     GotoXY(1,22);
     write('                                                        ');
end;


procedure antet;
begin
     ClrScr;
     GotoXY(1,10);
     HighVideo;
     Writeln('                                  Univ. Politehnica TM');
     LowVideo;
     Writeln('                                 ====================');
     GotoXY(1,18);
     Writeln('                 Program test Aprommer - Ver. 1.30');

     writeln('                         @ A. Gontean Februarie 1992');
     press_CR;
end;

procedure det_LPTi;     {determina adresa fizica a LPT3}
begin
     p:=ptr($0000,$040C);
     pd:=p^;
     ps:=pd+1;          {initializeaza pc si ps}
     pc:=pd+2;
end;

procedure scrie_x(x:byte);
begin
     port[pd]:=x;
     port[pc]:=$09;{-WR=0, -RD=1, STB=0}
     port[pc]:=$0D;{-WR=1, -RD=1, STB=0}
end;

procedure scrie_sel_in_U2(sel:byte);
begin
     port[pc]:=$0C;     {-A/B MUX=0, -WR=1, -RD=1, STB=1,
                         {latch-ul U2 devine transparent}
     port[pd]:=sel;     {Select. 8253/8255 si port cerut}
     port[pc]:=$0D;     {-A/B MUX=0, -WR=1, -RD=1, STB=0}
                        {Latch-ul U2 memoreaza selectia}
end;

procedure citeste_825x(sel:byte);
begin
     scrie_sel_in_U2(sel);   {selectie 8255i, 8253}
     port[pc]:=$0F;          {SEL INP=-A/B=0, INIT=-WR=1}
                             {AUTO FD=-RD=0, STB=0}
     y:=port[ps];            {citire nibble inferior}
     y:=y shr 4;
     port[pc]:=$07;          {SEL INP=-A/B=1, INIT=-WR=1}
                             {AUTO FD=-RD=0, STB=0}
     y:=y + (port[ps] AND $F0);   {nibble superior si rezultat}
```

```
     port[pc]:=inactiv;       {SEL INP=-A/b=0, INIT=-WR=1}
                              {AUTO FD=-RD=1, STB=0}
     y:=y XOR $88;            {complementare bitii 3 si 7}
end;

procedure citeste_date(sel:byte);
{citeste magistrala interna de date; NU 825x}
begin
     scrie_sel_in_U2(sel);   {selectie 8255i, 8253}
     port[pc]:=$0D;      {SEL INP=-A/B=0, INIT=-WR=1, AUTO FD=-RD=1, STB=0}
     y:=port[ps];        {citire nibble inferior}
     y:=y shr 4;
     port[pc]:=$05;      {SEL INP=-A/B=1, INIT=-WR=1, AUTO FD=-RD=1, STB=0}
     y:=y + (port[ps] AND $F0);   {nibble superior si rezultat}
     y:=y XOR 136;       {complementare bitii 3 si 7}
end;

procedure scrie_825x(x,sel:byte);
{Octetul x este inscris in circuitul selectat cu sel}
begin
     scrie_sel_in_U2(sel);
     port[pd]:=x;        {Emisie pe liniile ext. de date}
     port[pc]:=$09;      {-A/B MUX=0, -WR=0, -RD=1, STB=0}
                         {Se comanda scrierea in 8255-ul sau 8253 cerut}
     port[pc]:=$0D;      {-A/B MUX=0, -WR=1, -RD=1, STB=0}
                         {Incheierea operatiei de scriere}
end;

procedure initializeaza;     {initializeaza Aprommer}
begin
     sel:=$07;           {8255#1, cuv de cda}
     x:=$80;             {8255, toate liniile de iesire MOD 0}
     scrie_825x(x,sel); {8255#1 e in modul 0}
     sel:=$04;           {8255#1, portul A}
     x:=$FF;
     scrie_825x(x,sel);
     sel:=$05;           {8255#1, portul B}
     scrie_825x(x,sel);
     sel:=$06;           {8255#1, portul C}
     x:=$81;             {-ENVcc=1, Vcc1,2,3=0,G0,G1,G2=0, FREE1=1}
     scrie_825x(x,sel);

     sel:=$0B;           {8255#2, cuv de cda}
     x:=$89;             {8255, MOD 0, Port A,B, Port C, PC0-PC7 intrare}
     scrie_825x(x,sel); {8255#2 e in modul 0}
     sel:=$08;           {8255#2, portul A}
     x:=$FF;
     scrie_825x(x,sel);
     sel:=$09;           {8255#2, portul B}
     scrie_825x(x,sel);

     sel:=$0F;           {8255#3, cuv de cda}
     x:=$80;             {8255, toate liniile de iesire MOD 0}
     scrie_825x(x,sel); {8255#3 e in modul 0}
     sel:=$0C;           {8255#3, portul A}
     x:=$FF;
     scrie_825x(x,sel);
     sel:=$0D;           {8255#3, portul B}
     scrie_825x(x,sel);
     sel:=$0E;           {8255#3, portul C}
     x:=$FF;             {toate liniile de comanda inactive}
     scrie_825x(x,sel);
end;

procedure scrie_3(x:byte);
begin
```

```
     port[pd]:=x;
     port[pc]:=$09;      {INIT=-WR=0, AUTO FD=-RD=1, STB=0}
                         {ATENTIE! INIT este singurul semnal}
                         {din portul de comanda neinversat la cupla!}
                         {celelalte sint inversate}
     GotoXY(1,i);
     writeln('Test 3: Emisie ',x,' pe magistrala interna de date
Aprommer');
     press_CR;
end;

procedure scrie_eroare_8255;
begin
     numar_8255:= ((sel AND $0C) SHR 2);
     port_8255:=    sel AND $03;
     case port_8255 of
     0: Ch:='A';
      1: Ch:='B';
      2: Ch:='C';
  end;

  writeln('Eroare la 8255 #',numar_8255,'  Portul ',Ch,'; s-a scris ',x,';
s-a citit ',y);
  delay(50);
end;

procedure scrie_OK_8255;
begin
     numar_8255:= ((sel AND $0C) SHR 2);
     port_8255:=    sel AND $03;
     case port_8255 of
     0: Ch:='A';
     1: Ch:='B';
     2: Ch:='C';
  end;
  writeln('       OK: 8255 #',numar_8255,'  Portul ',Ch);
     delay(50);
end;

procedure mesaj_DP;     {mesaj numar disp. periferic}
begin
     ClrScr;
     writeln('Numarul dispozitivului periferic ?');
     writeln('1. 8255 #1');
     writeln('2. 8255 #2');
     writeln('3. 8255 #3');
     writeln;
     repeat
          begin
               Ch:=ReadKey;
                    case Ch of
                         '1': sel:=4;
                         '2': sel:=8;
                         '3': sel:=12;
                    else beep_scurt;
                end;
        end;
     until (Ch='1') OR (Ch='2') OR (Ch='3');
    press_CR;
    GotoXY(1,8);

     writeln('Portul din 8255 A,B,C,Comanda ?');
     writeln('1. Portul A');
     writeln('2. Portul B');
     writeln('3. Portul C');
     writeln('4. Cuvint de comanda');
```

```
     writeln;
     repeat
     begin
               Ch:=ReadKey;
                    case Ch of
                         '1': sel:=sel+0;
                         '2': sel:=sel+1;
                         '3': sel:=sel+2;
                         '4': sel:=sel+3;
                       else beep_scurt;
                end;
        end;
    until (Ch='1') OR (Ch='2') OR (Ch='3') OR (Ch='4');
    press_CR;
end;

procedure scrie_test_8255(x:byte);
begin
     mesaj_DP;
     scrie_sel_in_U2(x);
end;

procedure test_8255_x;
{testeaza cipul 8255 si portul specificate prin sel}
begin
     x:=Lo(Random(255));
     scrie_825x(x,sel);
     citeste_825x(sel);
     if (x<>y) then scrie_eroare_8255
     else scrie_OK_8255;
end;

procedure prog_8253_monostabil;
{Programeaza contorul 0,1,2 din 8253 pentru un impuls de 65536 * Ttact =
65536 * 900 ns = 58.98 ms}
begin
     scrie_825x(cw,sel);
     scrie_825x(tw,sel);
end;

procedure prog_8253_astabil;
begin
     scrie_825x(cw,sel);
     scrie_825x(tw,sel);
end;

procedure pregateste_soclu;
begin
     initializeaza;
     writeln('Introduceti circuitul in soclu');
     writeln;
     press_CR;
end;

procedure init_EPROM;
begin
     sel:=$06;
     x:=$85;            {eVcc inhibat, programat pentru 5.25 V}
     sel:=$08;
     x:=$81;            {eVpp inhibat, programat pentru 9.50 V}
end;

procedure inhiba_adr_E;
begin
     scrie_825x($FF,$04);
     scrie_825x($FF,$05);
```

```
end;

procedure emite_adr_E;
begin
     sel:=$04;
     scrie_825x(LO(adr),sel);
     sel:=$05;
     scrie_825x(HI(adr) + mask_adr,sel);
end;

procedure init_adr;
begin
     adr:=0;
     case tip_E of
          1: mask_adr:=$F8;
          2: mask_adr:=$F0;
          3: mask_adr:=$E0;
          4: mask_adr:=$C0;
          5: mask_adr:=$80;
          6: mask_adr:=$00;
     end;
     emite_adr_E;
end;

procedure meniu_E;
begin
     ClrScr;
     writeln('Alegeti tipul de EPROM:');
     writeln(' 1. 2716');
     writeln(' 2. 2732');
     writeln(' 3. 2764');
     writeln(' 4. 27128');
     writeln(' 5. 27256');
     writeln(' 6. 27512');
          begin
               Ch:=ReadKey;
               Case Ch of
                    '1' : tip_E:=1;
                    '2' : tip_E:=2;
                    '3' : tip_E:=3;
                    '4' : tip_E:=4;
                    '5'  :    tip_E:=5;
                    '6'  : tip_E:=6;
               else beep_scurt;
               end;
          end;
          lungime_E:=1024;
          for i:=1 to tip_E do lungime_E:=lungime_E*2;
          writeln;
          writeln(lungime_E,' octeti');
          press_CR;
end;

procedure citeste_E;
begin
  emite_adr_E;
  sel:=$0E;
  x:=$CC»                    {-CE=0, PC6, 8255#3}
  scrie_825x(x,sel);
  x:=$CC;                    {-OE=0, PC4, 8255#3}
  sel:=$0C;
  citeste_825x(sel);         {y contine octetul citit}
  sel:=$0E;
  x:=$FF;
  scrie_825x(x,sel);         {SB0-SB7 inactive}
  inhiba_adr_E;
```

```
end;

procedure save_EPROM;
begin
     ClrScr;
     pregateste_soclu;
     GotoXY(1,5);
     write('Numele fisierului in care salvati EPROM-ul?');
     readln(nume_fis);
     assign(f,nume_fis);
     rewrite(f);
    init_adr;
     for m:=1 to lungime_E do
     begin
          citeste_E;
          write(f,y);
          adr:=adr+1;
     end;
     close(f);
end;

procedure test_1;
{      Test 1 - Verificare generala port de date
       ========================================
}
begin
     ClrScr;
     writeln('Test 1: Emisie  00 pe portul de date');
     x:=$00;
     press_CR;
     port[pd]:=x;
     GotoXY(1,2);
     writeln('Test 1: Emisie $FF pe portul de date');
     x:=$FF;
     port[pd]:=x;
     press_CR;
end;

procedure test_2;
begin
ClrScr;
{      Test 2 - Verificare generala port de comanda
       ============================================
}
     x:=$00;
     port[pc]:=x;
     writeln('Test 2: Emisie  00 pe portul de comanda');
     press_CR;
     GotoXY(1,2);
     writeln('Test 2: Emisie $FF pe portul de comanda');
     x:=$FF;
     port[pc]:=x;
     press_CR;
end;

procedure test_3;
{Test 3 - Verificare generala magistrala interna de date Aprommer
=================================================================
}
begin
     ClrScr;
     i:=1;
     scrie_3($0AA);
     i:=2;
     scrie_3($55);
end;
```

```
procedure test_4;  {se testeaza U4 - 74LS157}
begin
{      Test 4 - Citire magistrala interna de date si test MUX
     ============================================================
}
  ClrScr;
  writeln('Test Walking zero pe magistrala interna de date');
  x:=$80;       { 1000 0000 masca test}
  i:=0;         {contor deplasari dreapta}
  repeat
     begin
          port[pd]:=x;
          port[pc]:=$09;{SEL INP=-A/B=0, INIT=-WR=0, AUTO FD=-RD=1, STB=0}
          y:=port[ps];
          y:=y shr 4;
          port[pc]:=$01;{SEL INP=-A/B=1, INIT=-WR=0, AUTO FD=-RD=1, STB=0}
          y:=y+(port[ps] AND $F0);
          y:=y XOR 136;
     end;
 writeln('    Test bitul ',i);
 delay(500);
 if NOT (x=y) then writeln('                EROARE pe bitul ',i);
 x:=x SHR 1;
 i:=i+1;
 until x=0;
 press_CR;
end;

procedure test_5;
{               Test 5 - Test 8253 astabil
                ============================}

begin
     ClrScr;
     initializeaza;
     sel:=$03»            {cuv. cda 8253}
     cw:=$26;             {Contor 0, mod 3}
     tw:=$D5;
     prog_8253_astabil;
     cw:=$66;             {Contor 1, mod 3}
     prog_8253_astabil;
     cw:=$A6;             {Contor 2, mod 3}
     prog_8253_astabil;

     sel:=$06»            {8255#1, Port C}
     x:=$F1;              {G0, G1, G2 =1}
     scrie_825x(x,sel);
     press_CR;
     x:=$81;
     scrie_825x(x,sel); {G0,G1,G2=0}
end;

procedure test_6;
begin
{          Test 6 - Test functionare 8253 ca monostabil
          ==============================================}

     ClrScr;
     initializeaza;
     sel:=$03;
     cw:=$22;
     prog_8253_monostabil;
     sel:=$0B;          {8255#2, cuv de cda}
     x:=$81;            {8255, MOD 0, Port A,B, PC4-PC7 iesire}
                        {Port C, PC0-PC3 intrare}
```

```
     scrie_825x(x,sel); {8255#2 e in modul 0}

     sel:=$0A;          {8255 #2, portul C}
     citeste_825x(sel); {citesc portul C, 8255 #2}
     y:=y AND $02;      {masca selectie Out0, 8253 #0}
     if (y=0) then
          begin
            beep_scurt;
            writeln('Eroare: Iesirea 8253 Out 0 nu este 1');
            press_CR;
          end;
     sel:=$06;          {8255#1, portul C}
     x:=$91;            {invalidare eVcc, setare G0=1, G1 si G2=0 (8253#0)}
     scrie_825x(x,sel); {declansare monostabil}
     sel:=$03;
     x:=$02;            {citire 8253 contor 0}
     scrie_825x(x,sel);
     delay(10);
     citeste_825x(sel);
     contor:=y;
     GotoXY(5,5);
     write('contor=',contor);
     sel:=$0A;          {8255 #2, portul C}
     delay(20);
     citeste_825x(sel); {citessc portul C, 8255 #2}
     y:=y AND $01;      {masca selectie Out0, 8253 #0}
     GotoXY(1,2);
     if (y=1) then
          begin
               beep_scurt;
               writeln('Eroare: 8253 Out 0 nu a declansat');
               press_CR;
          end;
     delay(100);        {astept sa termine de numarat}
     citeste_825x(sel); {citessc portul C, 8255 #2}
     y:=y AND $01;      {masca selectie Out0, 8253 #0}
     GotoXY(1,3);
     if (y=0) then
        begin
          beep_scurt;
          writeln('Eroare: 8253 Out 0 nu a terminat de numarat');
          press_CR;
        end
          else begin
               GotoXY(5,5);
               HighVideo;
               writeln('     8253 a trecut testul ca monostabil');
               LowVideo;
               press_CR;
          end;
     sel:=$06;
     x:=$81;
     scrie_825x(x,sel);
end;

procedure test_7;
{         Test 7 Programare si functionare 8255 (#1, #2, #3
         ======================================================
}

begin
     ClrScr;
     initializeaza;
     sel:=$04;      {8255#1, portul A}
     test_8255_x;
     sel:=$05;      {8255#1, portul B}
```

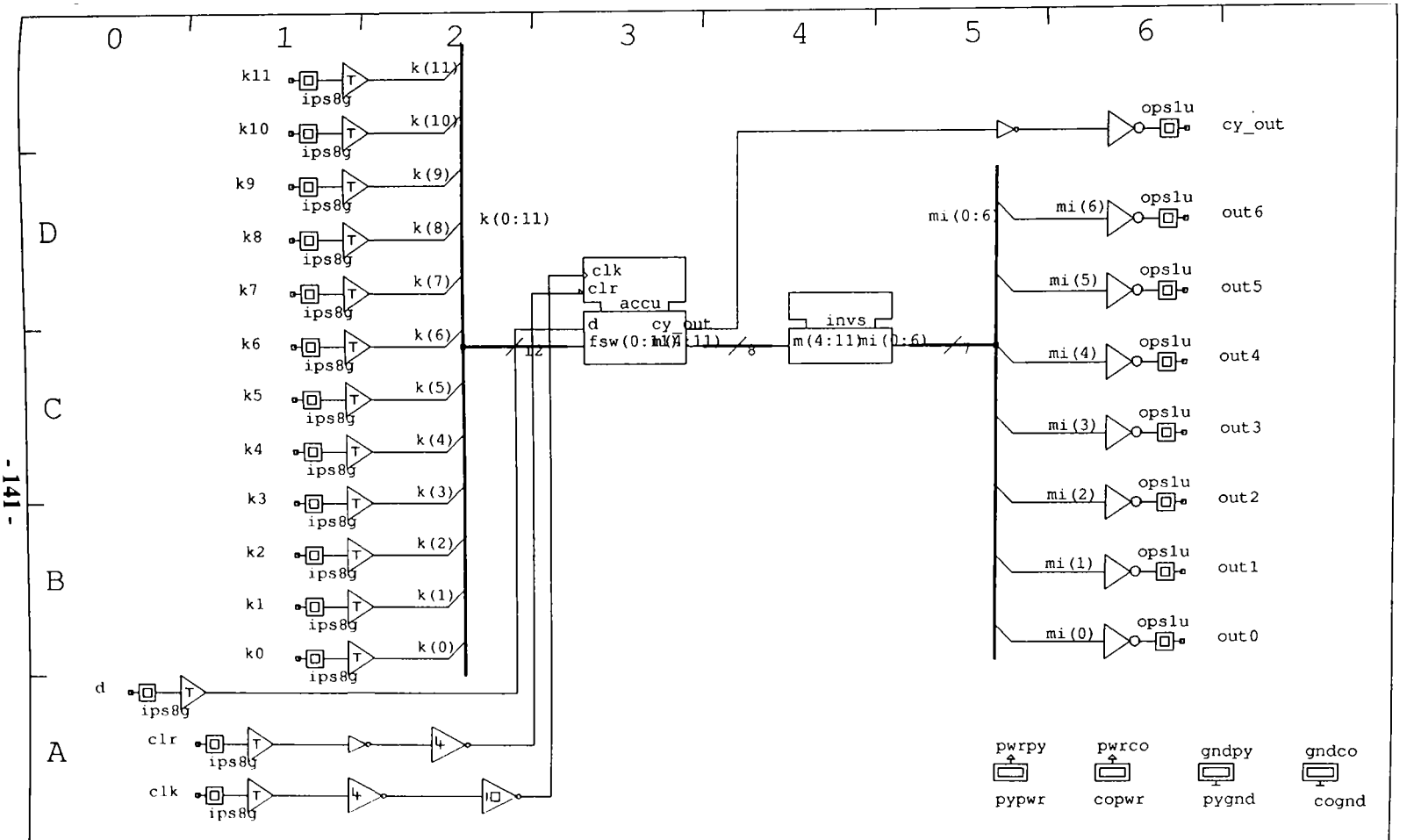
```
     test_8255_x;
     sel:=$06;      {8255#1, portul C}
     test_8255_x;
     writeln;

     sel:=$08;      {8255#2, portul A}
     test_8255_x;
     sel:=$09;      {8255#2, portul B}
     test_8255_x;
     writeln;

     sel:=$0C;      {8255#3, portul A}
     test_8255_x;
     sel:=$0D;      {8255#3, portul B}
     test_8255_x;
     sel:=$0E;      {8255#3, portul C}
     test_8255_x;

     writeln;
     writeln;
     writeln('Pe liniile 8255 #1,2,3 se gaseste combinatia 0101..');
     press_CR;
end;

procedure test_8;
{
          Test 8 - Test eVpp Programabilitate, reglaj
          ===========================================
}

begin
     ClrScr;
     Initializeaza;
     sel:=$08;
     x:=$81;
     scrie_825x(x,sel);
     writeln('Sursa eVpp inhibata');
     press_CR;

     ClrScr;
     x:=$80;
     GotoXY(1,4);
     scrie_825x(x,sel);
     write('Sursa eVpp livreaza Vpp_minim =  9.50 V');
     press_CR;

     ClrScr;
     eVpp:=9.5;
     while (eVpp < 25.25) do
          begin
               gotoXY(1,8);
               write('Verificati eVpp=',eVpp:5:2);
               scrie_825x(x,sel);
               eVpp:=eVpp+0.25;
               x:=x+2;
               press_space;
               repeat until ReadKey=' ';
          end;

     x:=$FE;
     GotoXY(1,8);
     scrie_825x(x,sel);
     writeln('Sursa eVpp livreaza Vpp_maxim = 25.25 V');
     blank_space;
     press_CR;
```

```
     x:=$81;            {inhib sursa eVpp}
     scrie_825x(x,sel);
end;

procedure test_9;
{
              Test 9 Test sursa eVcc
             ========================
}

begin
     ClrScr;
     initializeaza;
     sel:=$06;
     x:=$81;
     scrie_825x(x,sel);
     writeln('Sursa eVcc inhibata');
     press_CR;

     ClrScr;
     x:=$80;
     GotoXY(1,4);
     scrie_825x(x,sel);
     write('Sursa eVcc livreaza Vcc_minim = 4.75 V');
     press_CR;

     ClrScr;
     eVcc:=4.75;
     while (eVcc < 6.50) do
          begin
               gotoXY(1,8);
               write('Verificati eVcc=',eVcc:5:2);
               scrie_825x(x,sel);
               eVcc:=eVcc+0.25;
               x:=x+2;
               press_space;
               repeat until ReadKey=' ';
          end;

     x:=$8E;
     GotoXY(1,8);
     scrie_825x(x,sel);
     writeln('Sursa eVcc livreaza Vcc_maxim = 6.50 V');
     blank_space;
     press_CR;

     x:=$81;            {inhibare sursa eVcc}
     scrie_825x(x,sel);
end;

procedure test_0;
{         Test 0  Citire EPROM si scriere intr-un fisier
          ================================================
}
begin
     meniu_E; {stabileste tip EPROM si-i calculeaza lungimea}
     init_EPROM;     {eVcc si eVpp setate, inhibate}
     save_EPROM;
  init_EPROM;      {eVcc si eVpp inhibate}
     beep_scurt;
     press_CR;
end;

procedure scrie_meniu;
begin
     ClrScr;
```

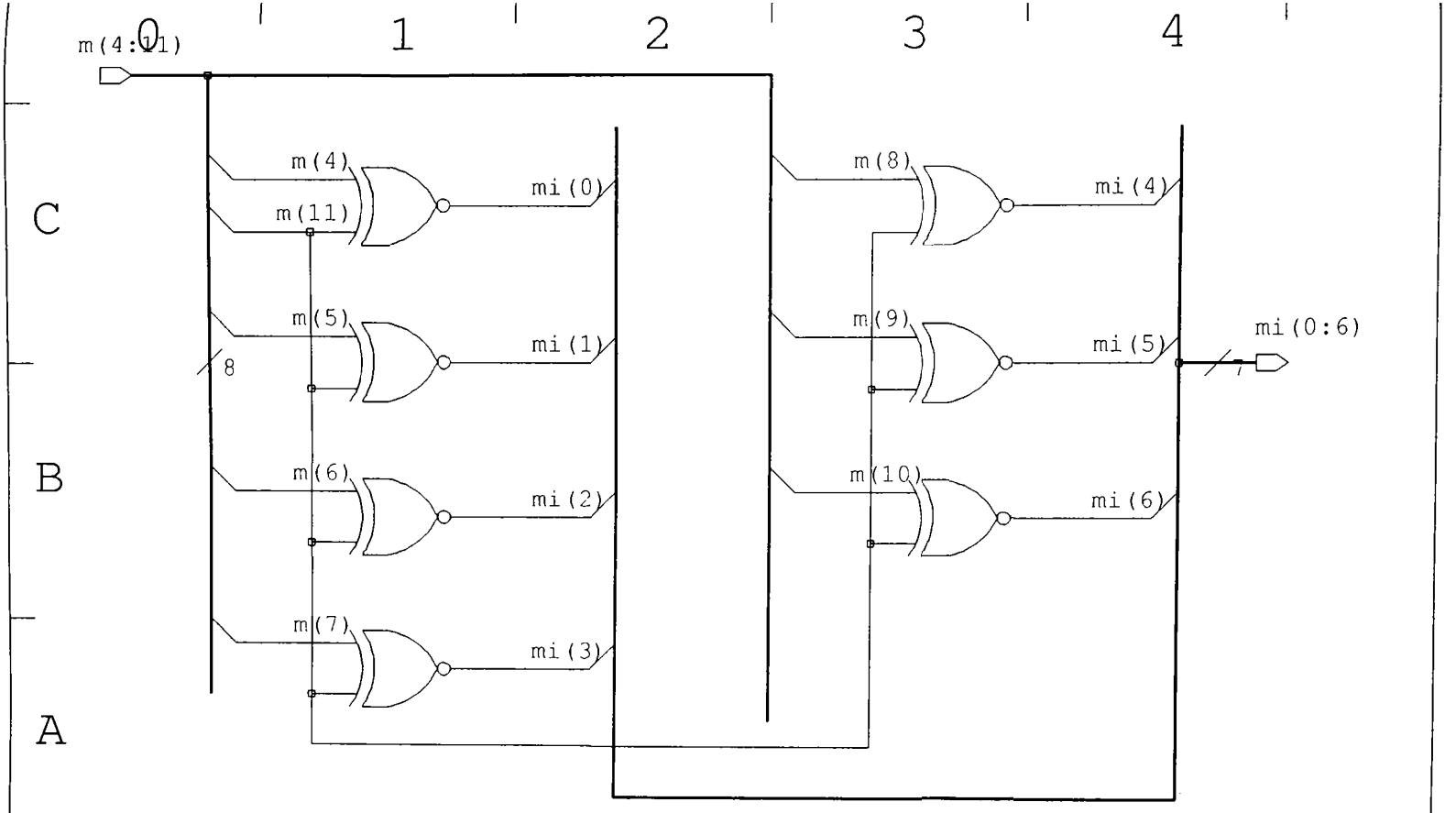
```
     GotoXY(1,3);
     writeln('       Selectati optiunea tastind numarul curent:');
     writeln;
     writeln('   1. Test port de date');
     writeln('   2. Test port de comanda');
     writeln('   3. Test general magistrala interna de date');
     writeln('   4. Test citire magistrala interna de date si MUX');
     writeln('   5. Test 8253 astabil');
     writeln('   6. Test 8253 monostabil');
     writeln('   7. Test programare 8255');
     writeln('   8. Test sursa programabila eVpp');
     writeln('   9. Test sursa programabila eVcc');
     writeln('   0. Test citire EPROM');
     press_ESC;
end;

procedure meniu_principal;
begin
     ClrScr;
     scrie_meniu;
     repeat
          begin
               Ch:=ReadKey;
               Case Ch of
                    '1' : test_1;
                    '2' : test_2;
                    '3' : test_3;
                    '4' : test_4;
                    '5' : test_5;
                    '6' : test_6;
                    '7' : test_7;
                    '8' : test_8;
                    '9' : test_9;
                    '0' : test_0;
               else beep_scurt;
               end;
          end;
     scrie_meniu;
     until Ch=#27;
ClrScr;
end;

begin {program principal}
     Antet;
     ClrScr;
     det_LPTi;
     meniu_principal;
end.
```

ANEXA 2

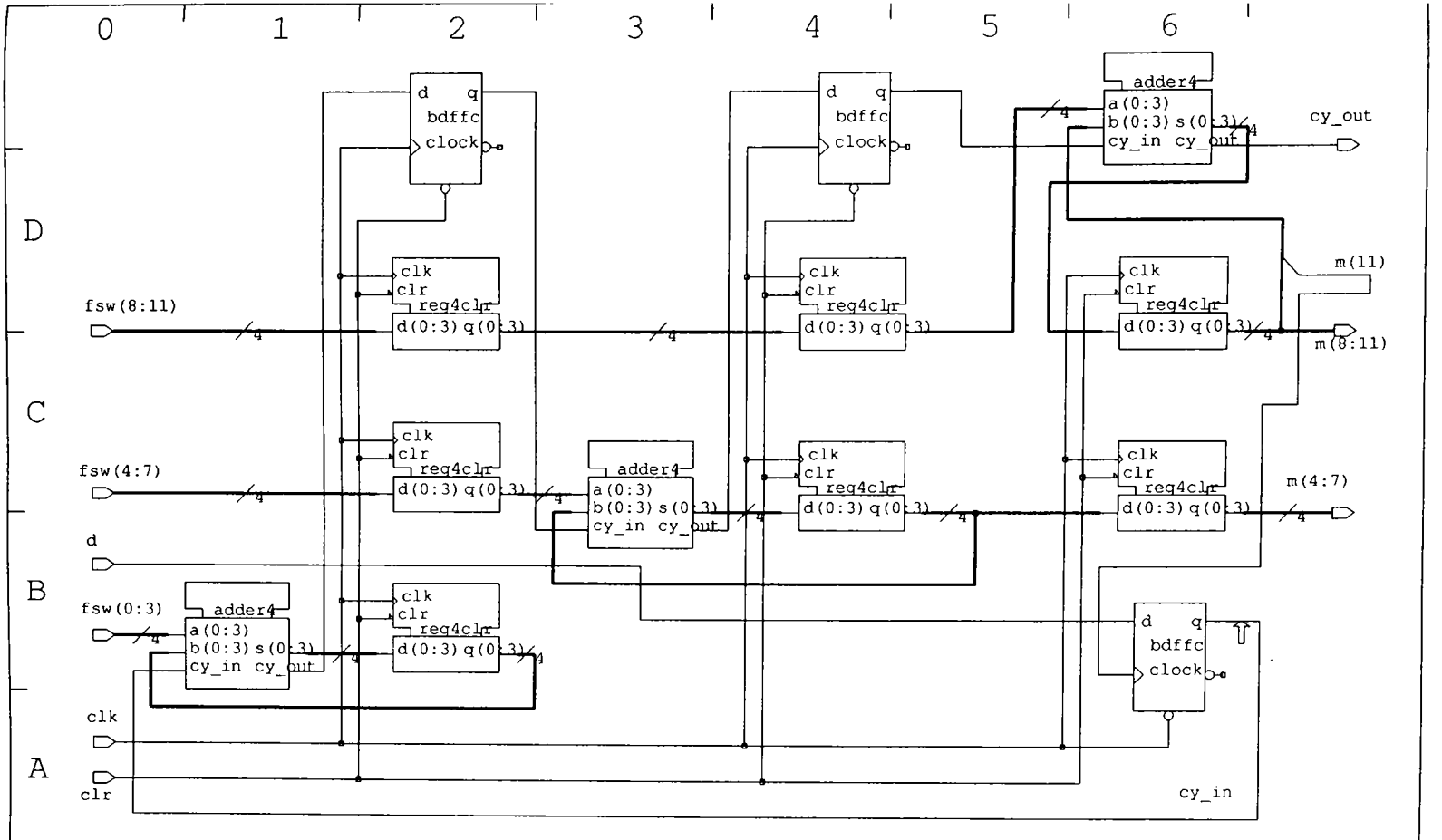
VARIANTE DE IMPLEMENTARE A SINTETIZORULUI DE FRECVENȚĂ



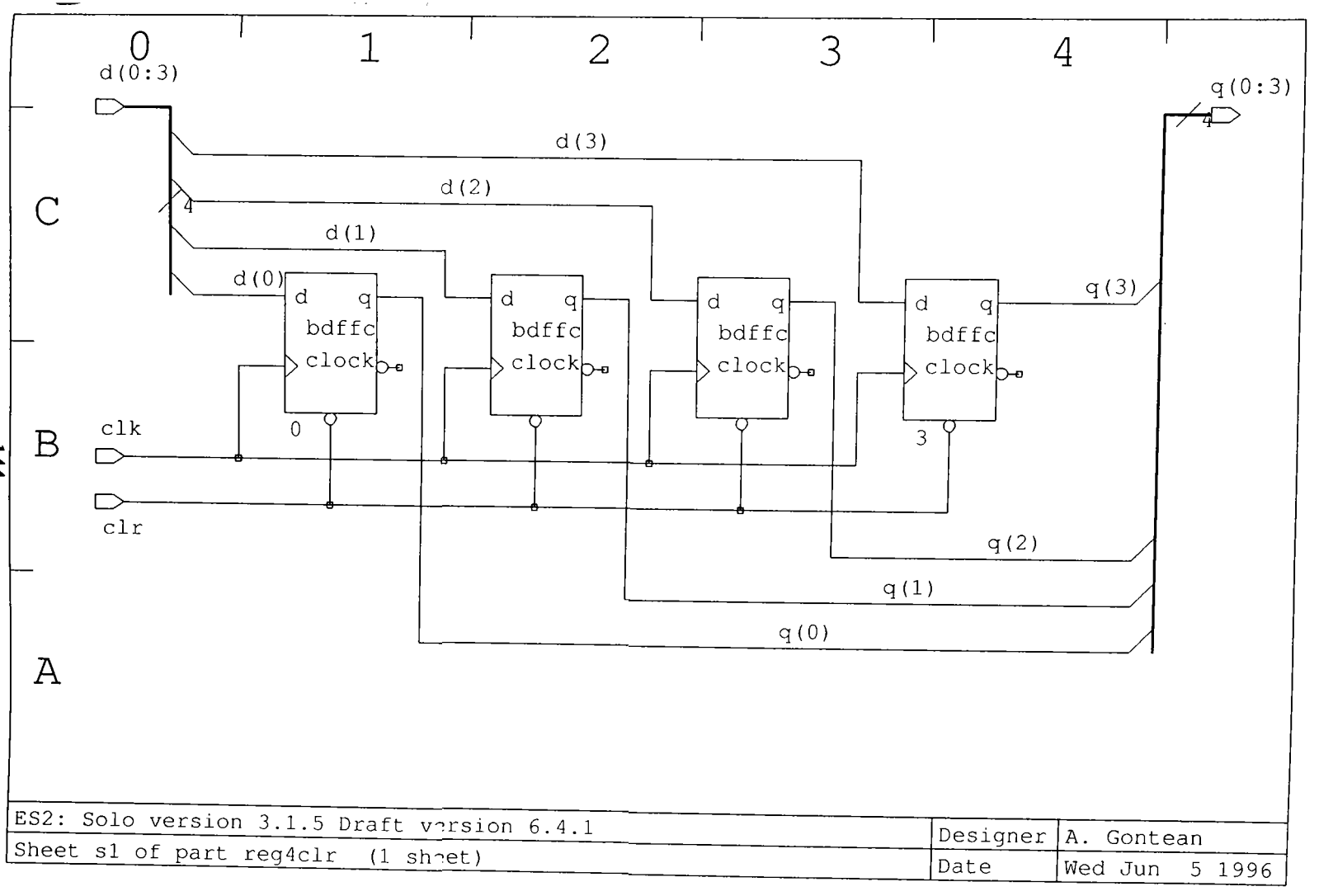
ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part dds1 (1 sheet)	Date	Wed Jun 5 1996



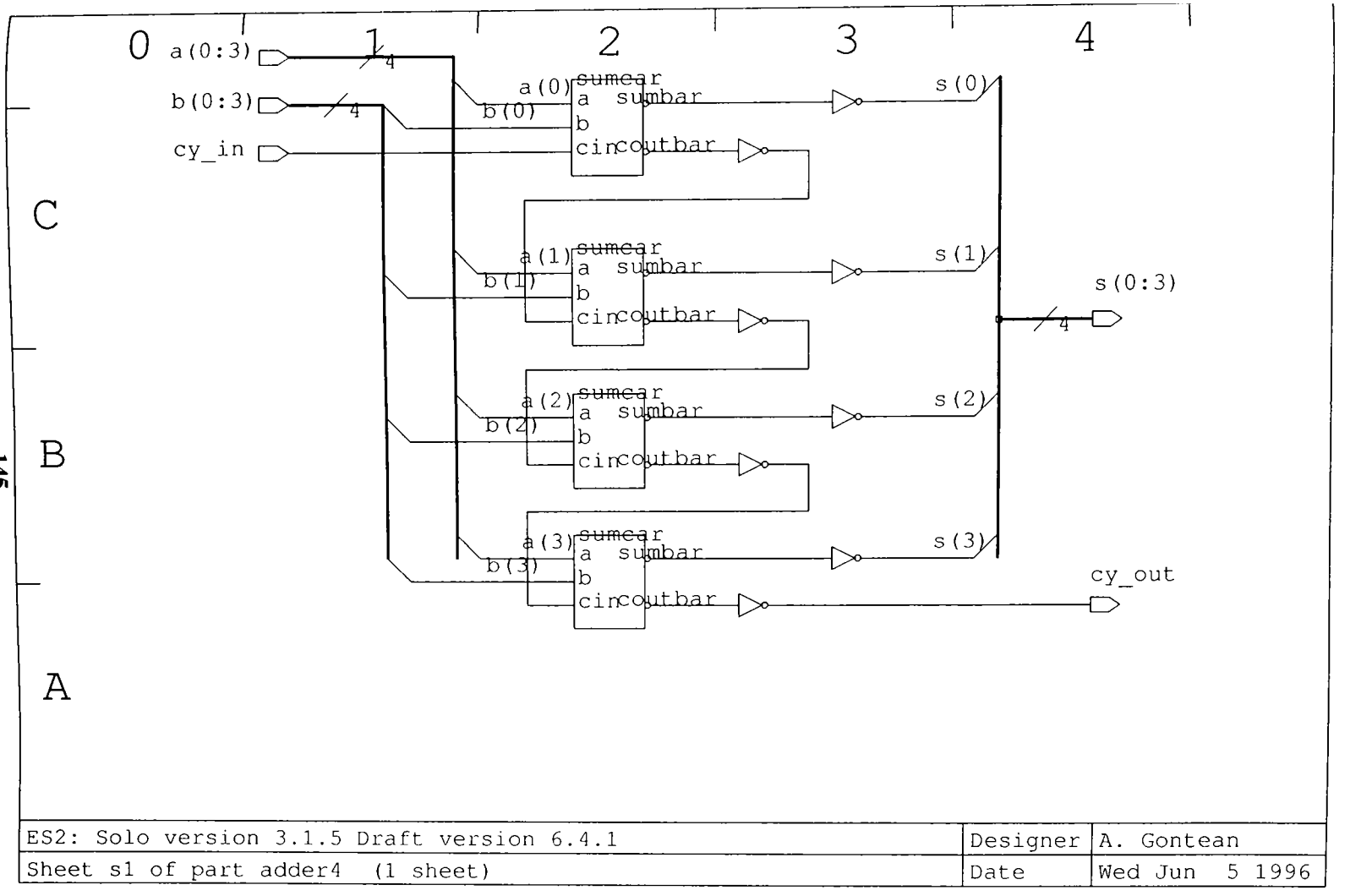
ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part invs (1 sheet)	Date	Wed Jun 5 1996



ES2: Solo version 3.1.5 Draft version 6.4.1		Designer	A. Gontean
Sheet s1 of part accu (1 sheet)		Date	Wed Jun 5 1996

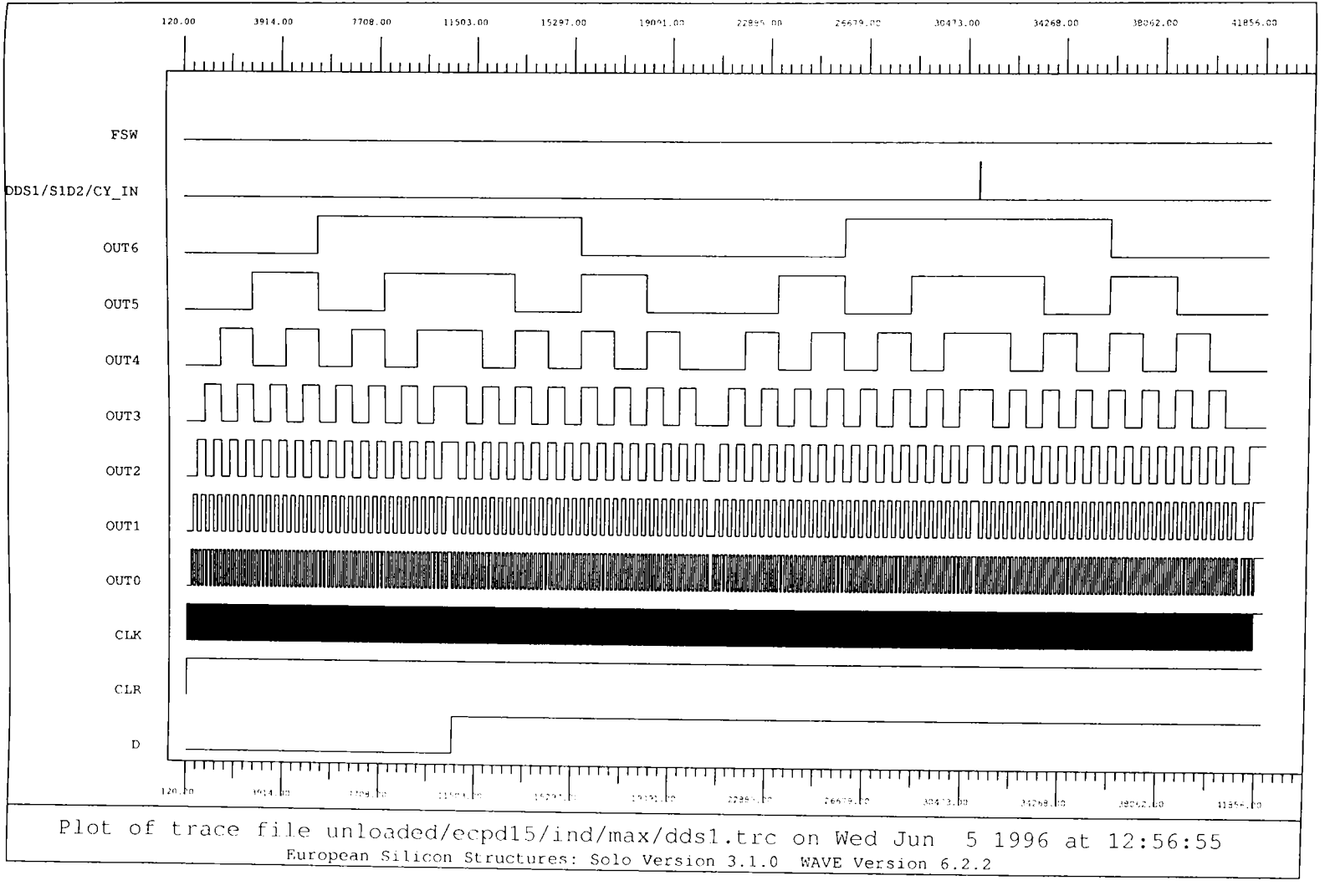


ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part reg4clr (1 sheet)	Date	Wed Jun 5 1996

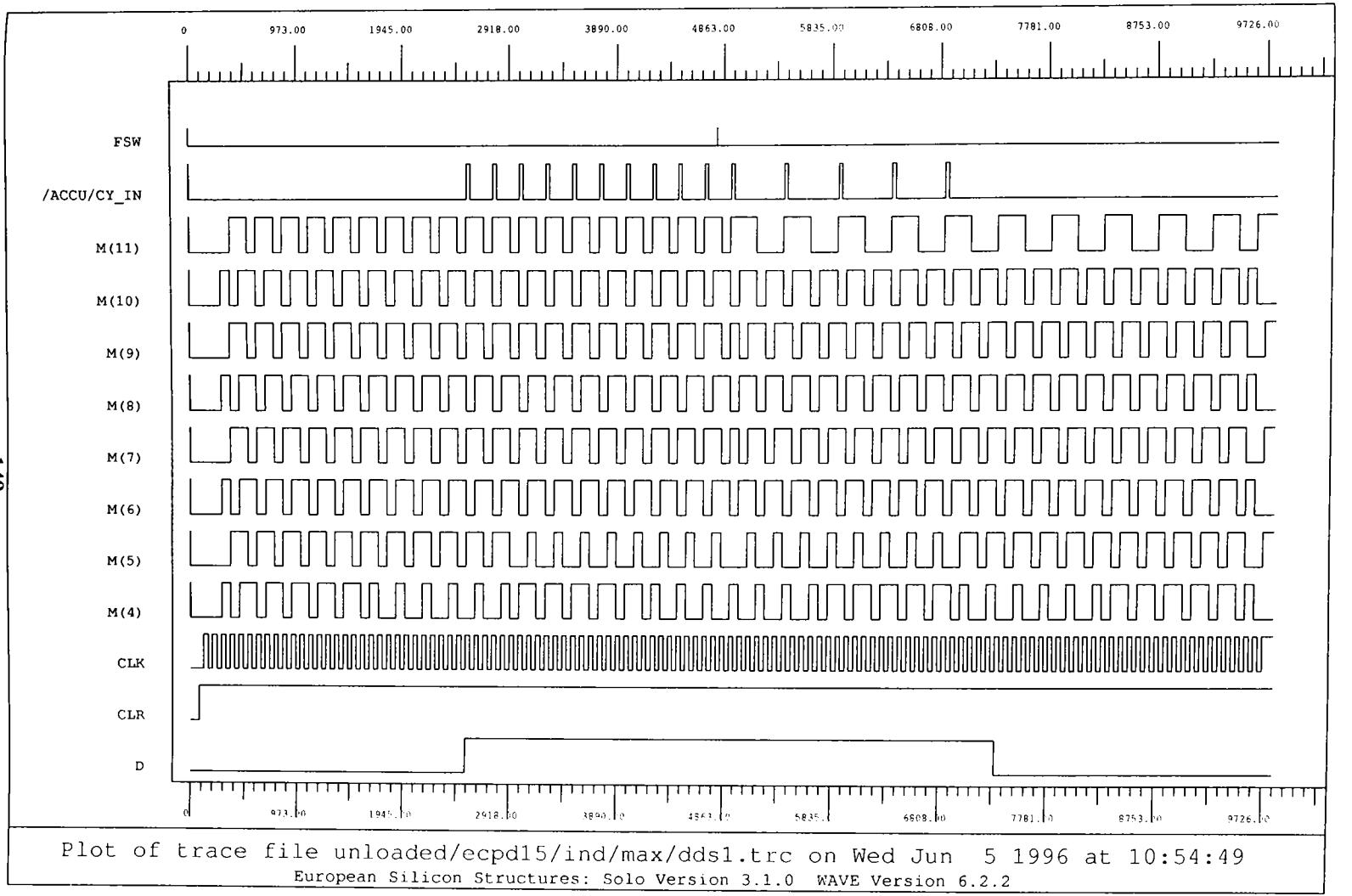


ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part adder4 (1 sheet)	Date	Wed Jun 5 1996

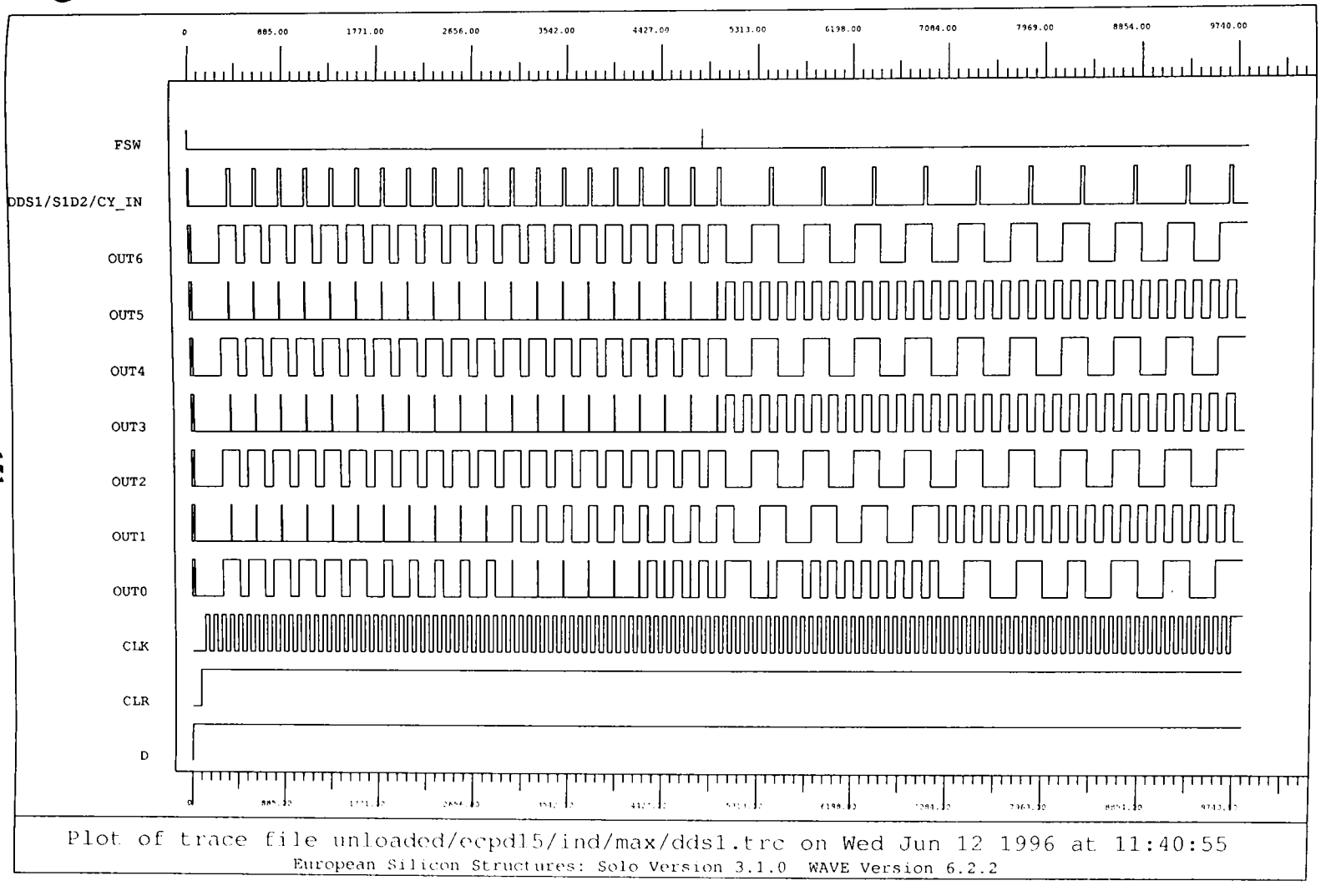
```
#step = 40 (cuanta de timp)
d = L step*3 =L step*262 (521 = H step*1042 =L)
k0  = L (H step *120 = H)
k1  = L (step *120 = H)
k2  = L (=H step *120 = L)
k3  = L (step *120 = H)
k4  = H (H step *120 = L)
k5  = L (step *120 = H)
k6  = L (H step *120 = L)
k7  = L (step *120 = H)
k8  = L (H (step *120 = L)
k9  = L (step *120 = H)
k10 = L (H step *120 = L)
k11 = L
clr = L step*2 =H
clk = L step*3 [=H step =L step]*520 (semnal de *ac*)
out0
out1
out2
out3
out4
out5
out6
```


120.00
3914.00
7708.00
11503.00
15297.00
19091.00
22885.00
26679.00
30473.00
34268.00
38062.00
41856.00
FSW
DDS1/S1D2/CY_IN
OUT6
OUT5
OUT4
OUT3
OUT2
OUT1
OUT0
CLK
CLR
D
Plot of trace file unloaded/ecpd15/ind/max/dds1.trc on Wed Jun 5 1996 at 12:56:55
European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

```
#step   40 {cuanta de timp}
d    L step*3  L step*60   H step*120  L
fsw(0:11)    1365 step*120  683 {stabilirea frecventei generate}
clr    L step*2  H
clk    L step*3 [ H step  L step]*120 {semnal de tact}
m(4:11)
```


0
973.00
1945.00
2918.00
3890.00
4863.00
5835.00
6808.00
7781.00
8753.00
9726.00
FSW
/ACCU/CY_IN
M(11)
M(10)
M(9)
M(8)
M(7)
M(6)
M(5)
M(4)
CLK
CLR
D
Plot of trace file unloaded/ecpd15/ind/max/dds1.trc on Wed Jun 5 1996 at 10:54:49
European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

```
#step = 40 {cuanta de timp}
d    = H {L step*3 =L step*262 521 = H step*1042 =L}
k0   = H step *120 = H
k1   = L step *120 = H
k2   = H step *120 = L
k3   = L step *120 = H
k4   = H step *120 = L
k5   = L step *120 = H
k6   = H step *120 = L
k7   = L step *120 = H
k8   = H step *120 = L
k9   = L step *120 = H
k10  = H step *120 = L
k11  = L
clr  = L step*2 =H
clk  = L step*3 [=H step =L step]*120 {semnal de tact}
out0
out1
out2
out3
out4
out5
out6
```


0
885.00
1771.00
2656.00
3542.00
4427.00
5313.00
6198.00
7084.00
7969.00
8854.00
9740.00
FSW
DDS1/S1D2/CY_IN
OUT6
OUT5
OUT4
OUT3
OUT2
OUT1
OUT0
CLK
CLR
D
Plot of trace file unloaded/ecpd15/ind/max/dds1.trc on Wed Jun 12 1996 at 11:40:55
European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

```
{       Position file for MODEL file dds1.idl
{-------------------------------------------

MODEL version 6.2.3     Wed Jun  5 1996     12:03:03
PLACE version 6.3       Wed Jun  5 1996     13:10:14

Number of columns  :    1

Column  1 -  number of rows:    7
             -  stages per row:  117

{ P = Position, from start of row (   direction stated).
{ S = Size, in terms of stages.
{ H = Height, in terms of rows. By (   ault 1 row.
{ T = Type, the type of the library element.
{ N = Nets, the net numbers connected to the part.

{ The Diagram below shows the Floor Plan Parts.
{
{ Scale is set to 5 Stages per 3 characters
{
{    +-------------------------------------------------+--+--+--------+--+
{ 7  |                                                 | c|**|    f   | g|
{    +                                                 +--+--+--------+--+
{ 6  |                                                                     |
{    +                                                                     +
{ 5  |                                                                     |
{    +                                                                     +
{ 4  |                                 b                                   |
{    +                                                                     +
{ 3  |                                                                     |
{    +                                                                     +
{ 2  |                                                                     |
{    +-------------------------------+                                     +
{ 1  |              a                |                                     |
{    +-------------------------------+-------------------------------------+
{
{ Index of Parts shown:
{
{       Instance Name                                 Model Name
{  a - /DDS1/S1D4Y2                                   invs
{  b - /DDS1/S1D2                                     accu
{  c - /DDS1/S1E5                                     not
{  f - /DDS1/S1A2Y4                                   buffer10
{  g - /DDS1/S1A2Y3                                   buffer4
{
{
{ Index of Parts not shown:
{
{       Instance Name                                 Model Name
{  d - /DDS1/S1A2                                     buffer4
{  e - /DDS1/S1A2Y2                                   not
{
{ Note: Any areas with the name '\\' consist of blank stages,
{       whilst any areas named '**' are too small to resolve
{

{ Column 1 Row 1,  L -> R, wastage  0 stages

/DDS1/S1D4Y2                                          { P   1, S  49, T invs, N
                                                      {  55 56 57 58 59 60 61 62
                                                      {  63 64 65 66 67 68 69

{ Column 1 Row 2,  R -> L, wastage  1 stage

{ Column 1 Row 3,  L -> R, wastage  5 stages

{ Column 1 Row 4,  R -> L, wastage  5 stages

{ Column 1 Row 5,  L -> R, wastage  13 stages

{ Column 1 Row 6,  R -> L, wastage  5 stages

{ Column 1 Row 7,  L -> R, wastage  2 stages

/DDS1/S1D2                                            { P  50, S  712, T accu, N
                                                      {  41 42 43 44 45 46 47 48
                                                      {  49 50 51 52 38 40 39 55
                                                      {  56 57 58 59 60 61 62 54
/DDS1/S1E5                                            { P  91, S  2, T not, N  54
                                                      {  53
/DDS1/S1A2                                            { P  93, S  5, T buffer4, N
                                                      {  36 40
/DDS1/S1A2Y2                                          { P  98, S  2, T not, N  37
                                                      {  36
/DDS1/S1A2Y4                                          { P 100, S  13, T buffer10
                                                      { , N  35 39
/DDS1/S1A2Y3                                          { P 113, S  5, T buffer4, N
                                                      {  34 35
```

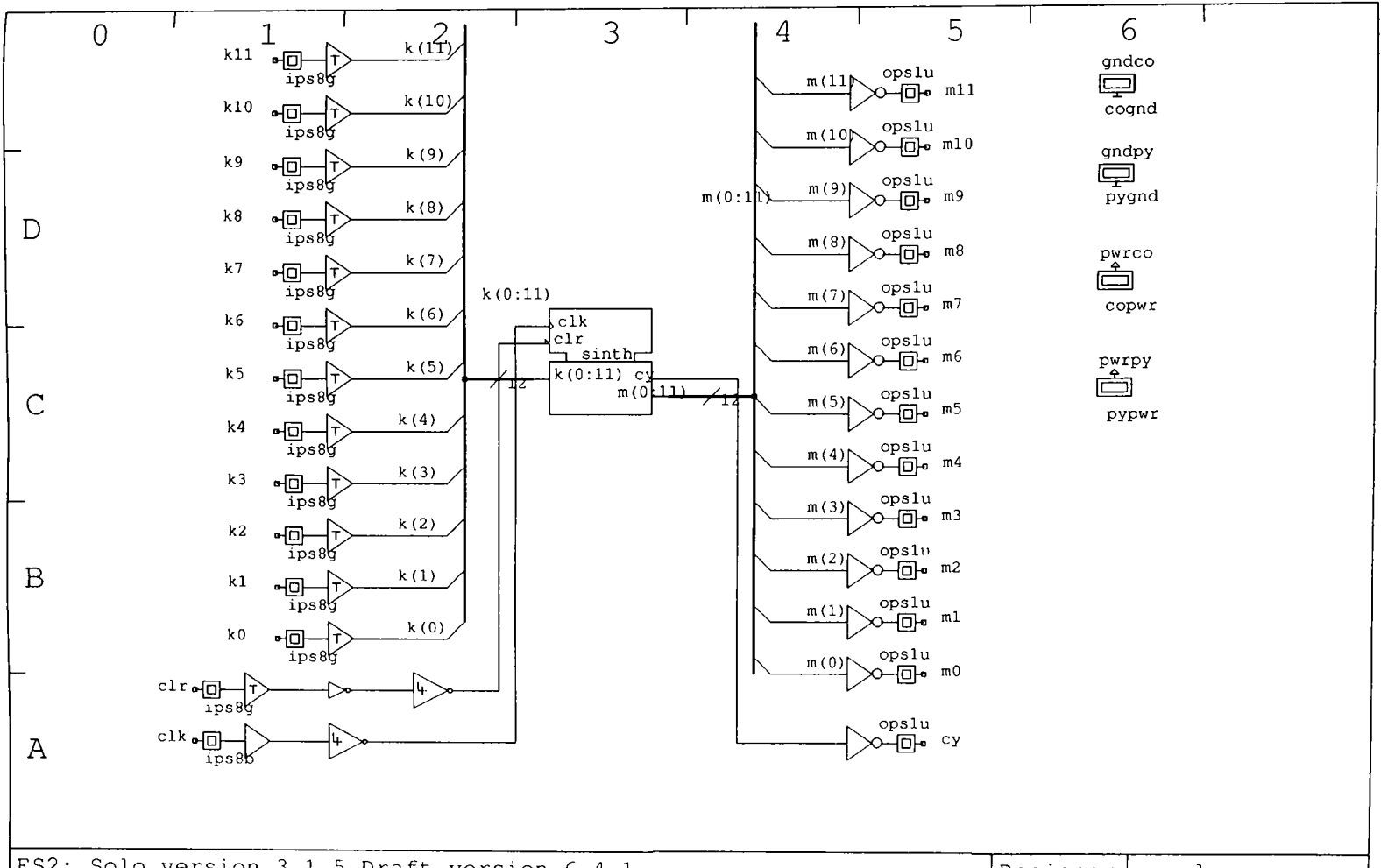
*2 Solo version 3.1; PLACE version 6.3 Wed Jun 5 1996 13:10:15

```
    Inputs  :    IDL file : dds1.idl
     PRIVATE PROCESS file : /apts/solo1400/ecpd15/process.ppf
      PUBLIC PROCESS file : /apts/solo1400/ecpd15/indl.prc

    Outputs :    PLC file : dds1.plc
                 PEX file : NONE
                 POS file : dds1.pos

    Options : /TWOLAYER/POSITION/LEVEL = 1/VARY/FEED/AUTO

    cif_file_version, ['(ES2) ecpd15 hardcell.cif v3.1*']

Column 1 Row  1            20 gates implemented in 117 stages
---------------

Column 1 Row  2            17 gates implemented in 116 stages
---------------

Column 1 Row  3             7 gates implemented in 112 stages
---------------

Column 1 Row  4             7 gates implemented in 112 stages
---------------

Column 1 Row  5            18 gates implemented in 104 stages
---------------

Column 1 Row  6             7 gates implemented in 112 stages
---------------

Column 1 Row  7            24 gates implemented in 115 stages
---------------

Number of columns   =   1
Rows in column 1    =   7
Size of row         = 117

Physical stages required          =  788
Total number of unused stages     =   31

Cpu time = 1.30 sec
```

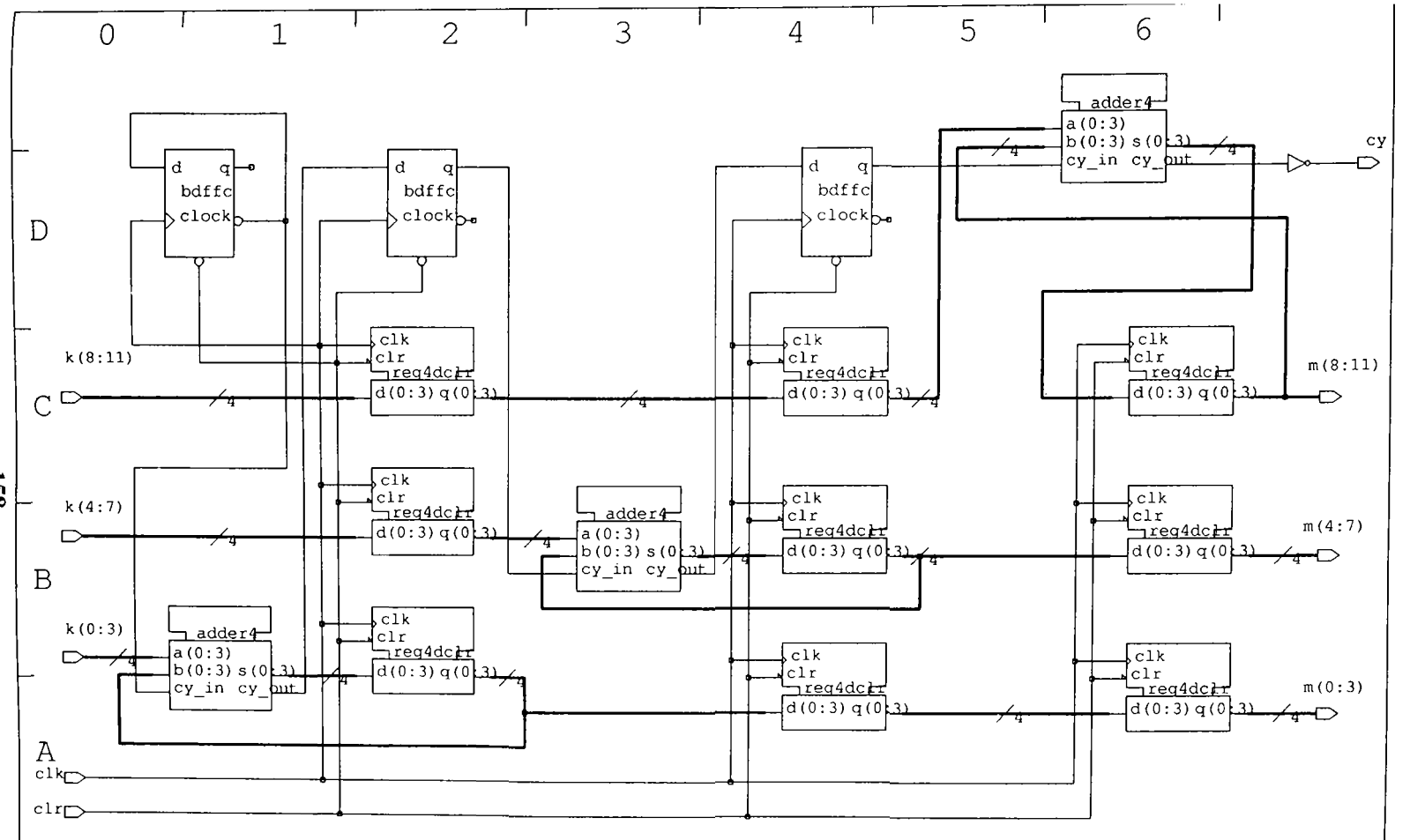
*3 Solo version 3.1; GATE version 6.2 Wed Jun 5 1996 13:10:58

```
    Inputs       PLC file : dds1.plc

    Outputs :    GAT file : dds1.gat

    Options : /OPTIMISED

    cif_file_version, ['(ES2) ecpd15 hardcell.cif v3.1*']
Cpu time = 0.300 sec
```

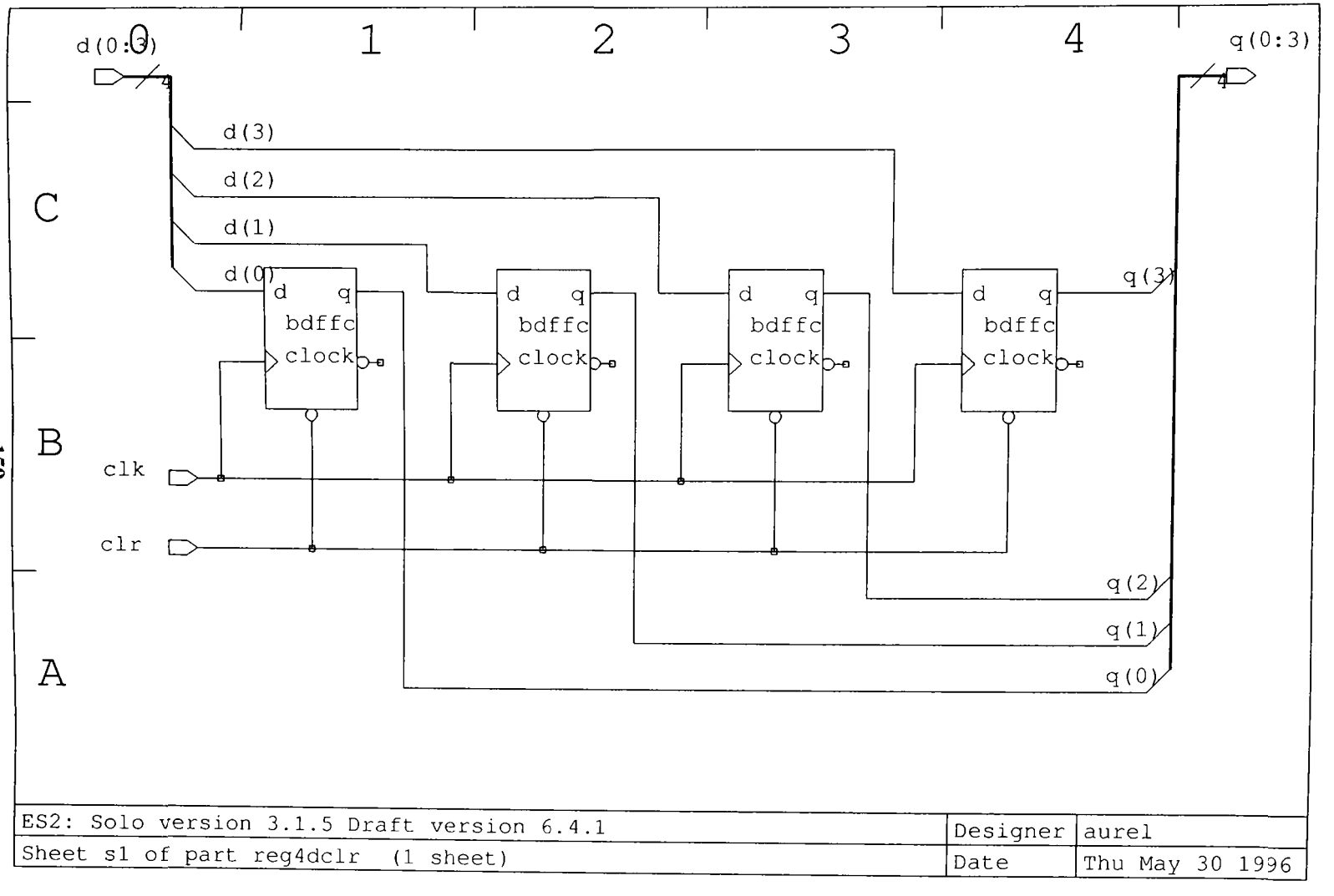
ddsl AG 5 iunie 1996 - Vedere generala

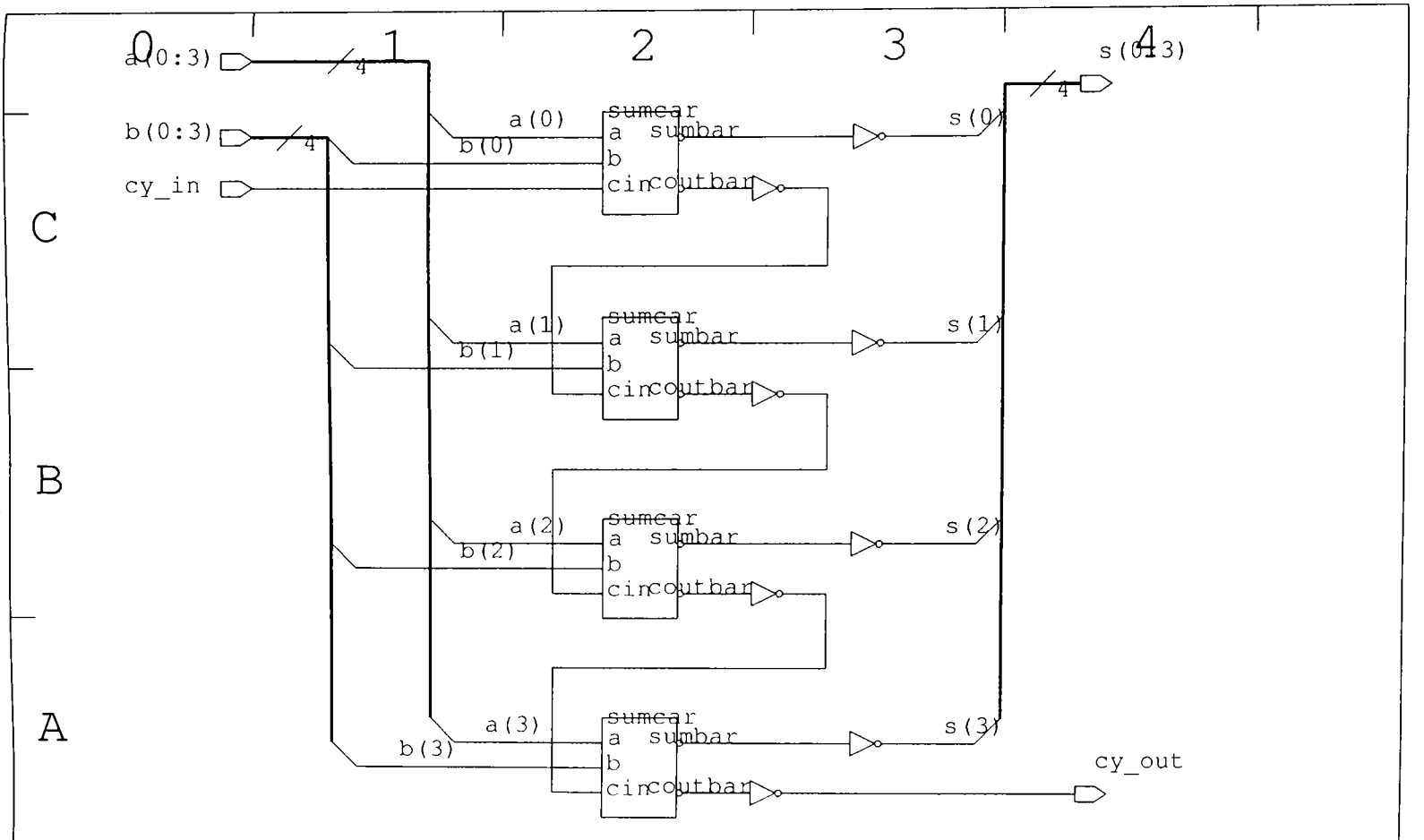


ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	aurel
Sheet s1 of part final (1 sheet)	Date	Thu May 30 1996



ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	aurel
Sheet s1 of part sinth (1 sheet)	Date	Thu May 30 1996


0
1
2
3
4
d(0:3)
q(0:3)
4
d(3)
d(2)
d(1)
d(0)
C
B
A
d q
bdffc
clock
q(3)
q(2)
q(1)
q(0)
clk
clr
ES2: Solo version 3.1.5 Draft version 6.4.1
Designer
aurel
Sheet s1 of part reg4dclr (1 sheet)
Date
Thu May 30 1996



ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	aurel
Sheet s1 of part adder4 (1 sheet)	Date	Thu May 30 1996

```
{       Position file for MODEL file dds.idl
{-----------------------------------------

MODEL version 6.2.3     Wed May 29 1996     15:00:17
PLACE version 6.3       Thu May 30 1996     10:47:15

Number of columns  :    2

Column  1 -  number of rows:    7
          -  stages per row:   64
Column  2 -  number of rows:    7
          -  stages per row:   64


{ P = Position, from start of row (in direction stated).
{ S = Size, in terms of stages.
{ H = Height, in terms of rows. By default 1 row.
{ T = Type, the type of the library element.
{ N = Nets, the net numbers connected to the part.

{ The Diagram below shows the Floor Plan Parts.
{
{ Scale is set to 6 Stages per 3 characters
{
{   +--------------------------------++--------------------------------+
{ 7 |                                ||                                |
{   +                                ++                                +
{ 6 |                                ||                                |
{   +                                ++                                +
{ 5 |                                ||                                |
{   +                                ++                                +
{ 4 |                c               ||                c               |
{   +                                ++                                +
{ 3 |                                ||                                |
{   +                                ++                                +
{ 2 |                                ||                                |
{   +--+--+                          ++                          +--+  +
{ 1 | a|**|                          ||                          | d|  |
{   +--+--+--------------------------++--------------------------+--+  +
{
{ Index of Parts shown:
{
{      Instance Name                          Model Name
{  a - /FINAL/S1B2Y4                          not
{  c - /FINAL/S1C3                            sinth
{  d - /FINAL/S1A2                            buffer4
{
{ Index of Parts not shown:
{
{      Instance Name                          Model Name
{  b - /FINAL/S1A2Y3                          buffer4
{
{ Note: Any areas with the name '\\' consist of blank stages,
{       whilst any areas named '**' are too small to resolve
{

{ Column 1 Row 1,  L -> R, wastage  7 stages

/FINAL/S1B2Y4                                 { P   1, S  2, T not, N  38
                                              {  39
/FINAL/S1A2Y3                                 { P   3, S  5, T buffer4, N
                                              {  39 40

{ Column 1 Row 2,  R -> L, wastage  0 stages

{ Column 1 Row 3,  L -> R, wastage  6 stages

{ Column 1 Row 4,  R -> L, wastage  2 stages

{ Column 1 Row 5,  L -> R, wastage  0 stages

{ Column 1 Row 6,  R -> L, wastage  0 stages

{ Column 1 Row 7,  L -> R, wastage  0 stages

{ Column 2 Row 7,  L -> R, wastage  0 stages

{ Column 2 Row 6,  R -> L, wastage  8 stages

{ Column 2 Row 5,  L -> R, wastage  0 stages

{ Column 2 Row 4,  R -> L, wastage  2 stages

{ Column 2 Row 3,  L -> R, wastage  6 stages

{ Column 2 Row 2,  R -> L, wastage  0 stages

{ Column 2 Row 1,  L -> R, wastage  11 stages

/FINAL/S1C3                                   { P  10, S  842, T sinth, N
                                              {  56 57 58 59 60 61 62 63
                                              {  64 65 66 67 40 42 44 45
                                              {  46 47 48 49 50 51 52 53
                                              {  54 55 43
```

/FINAL/S1A2

[P 99, S 5, T buffer4, N
[41 42

```
*2  Solo version 3.1;  PLACE version 6.3    Thu May 30 1996    10:47:15

    Inputs  :    IDL file : dds.idl
     PRIVATE PROCESS file : /apts/solo1400/ecpd12/process.ppf
      PUBLIC PROCESS file : /apts/solo1400/ecpd12/indl.prc

    Outputs :    PLC file : dds.plc
                 PEX file : NONE
                 POS file : dds.pos

    Options : /TWOLAYER/POSITION/LEVEL = 1/VARY/FEED/AUTO

    cif_file_version, ['(ES2) hardcell.cif ecpd12  v3.1*']

Column 1 Row  1            6 gates implemented in  57 stages
----------------

Column 1 Row  2            4 gates implemented in  64 stages
----------------

Column 1 Row  3           14 gates implemented in  58 stages
----------------

Column 1 Row  4            5 gates implemented in  62 stages
----------------

Column 1 Row  5            4 gates implemented in  64 stages
----------------

Column 1 Row  6            4 gates implemented in  64 stages
----------------

Column 1 Row  7            4 gates implemented in  64 stages
----------------

Column 2 Row  7           12 gates implemented in  64 stages
----------------

Column 2 Row  6            7 gates implemented in  56 stages
----------------

Column 2 Row  5            4 gates implemented in  64 stages
----------------

Column 2 Row  4           10 gates implemented in  62 stages
----------------

Column 2 Row  3            9 gates implemented in  58 stages
----------------

Column 2 Row  2            4 gates implemented in  64 stages
----------------

Column 2 Row  1            4 gates implemented in  53 stages
----------------

Number of columns    =   2
Rows in column 1     =   7
Size of row          =  64

Rows in column 2     =   7
Size of row          =  64

Physical stages required          =  854
Total number of unused stages     =   42

Cpu time = 0.910 sec
```

*3 Solo version 3.1; GATE version 6.2 Thu May 30 1996 10:47:24

Inputs PLC file : dds.plc

Outputs : GAT file : dds.gat

Options : /OPTIMISED

cif_file_version, ['(ES2) hardcell.cif ecpd12 v3.1*']
Cpu time = 0.260 sec

*4 Solo version 3.1.5; ROUTE version 6.2.4 Thu May 30 1996 10:52:24

Inputs : GAT file : dds.gat
Manual data : none

Outputs : RTE file : dds.rte
SIG file : dds.sig

Options : /OPTIMISED

Private process file : /apts/solo1400/ecpd12/process.ppf
Public process file : /apts/solo1400/ecpd12/ind1.prc
Default pad file : PAD_FILE
Pad gap specified : 0 hundredths of microns
POWER_SCALE : 2.50

cif_file_version, ['(ES2) allpads.cif ecpd12 v3.1*']
cif_file_version, ['(ES2) hardcell.cif ecpd12 v3.1*']
cif_file_version, ['(ES2) ecpd12 plastic.cif v3.1*']

Channel requirements
====================
Column 1 minor 1 - 7 channels
minor 2 - 8 channels
minor 3 - 8 channels
minor 4 - 7 channels
minor 5 9 channels
minor 6 - 7 channels
Column 2 minor 1 - 7 channels
minor 2 - 9 channels
minor 3 - 8 channels
minor 4 - 8 channels
minor 5 9 channels
minor 6 - 12 channels
Column 1 bottom highway - 68 channels
Column 2 bottom highway - 67 channels
Column 1 top highway - 69 channels
Column 2 top highway - 69 channels
Central highway 1 - 8 channels
Left outer highway - 33 channels
Right outer highway - 32 channels
Bottom extension - 1 channel
Top extension - 1 channel

TOP pad assignment

	ID	Type	Terminal	Signal
1	pvssdins	GNDPY		
2	copwrins	PWRCO		
3	pvpwrins	PWRPY		
4	cogndins	GNDCO		
5	m11ins	OPS10	a	m(11)
6	m10ins	OPS10	a	m(10)
7	m9ins	OPS10	a	m(9)

8 m8ins OPS1U a m(8)

BOTTOM pad assignment

	ID	Type	Terminal	Signal
1	clkins	IPS8B	y	sla1x1
2	k11ins	IPS8G	y	k(11)
3	k0ins	IPS8G	y	k(0)
4	m0ins	OPS1U	a	m(0)
5	m1ins	OPS1U	a	m(1)
6	m2ins	OPS1U	a	m(2)
7	m3ins	OPS1U	a	m(3)
8	m4ins	OPS1U	a	m(4)

LEFT pad assignment

	ID	Type	Terminal	Signal
1	clrins	IPS8G	y	sla1x2
2	k5ins	IPS8G	y	k(5)
3	k6ins	IPS8G	y	k(6)
4	k7ins	IPS8G	y	k(7)
5	k8ins	IPS8G	y	k(8)
6	k9ins	IPS8G	y	k(9)
7	k10ins	IPS8G	y	k(10)

RIGHT pad assignment

	ID	Type	Terminal	Signal
1	m7ins	OPS1U	a	m(7)
2	m6ins	OPS1U	a	m(6)
3	cyins	OPS1U	a	slc3x2
4	k4ins	IPS8G	y	k(4)
5	k3ins	IPS8G	y	k(3)
6	k2ins	IPS8G	y	k(2)
7	k1ins	IPS8G	y	k(1)
8	m5ins	OPS1U	a	m(5)

Cpu time = 2.460 sec

* 5 Solo version 3.1.2; DRAW version 6.3.2 Thu May 30 1996 10:52:41

1430412760

Inputs RTE file : dds.rte

Outputs : CIF file : dds.cif
LDA file : dds.lda

Options : /LOAD/OPTIMISED/CIF/PADS/TWOLAYER

Private process file : /apts/solo1400/ecpd12/process.ppf
Public process file : /apts/solo1400/ecpd12/indl.prc

cif_file_version, ['(ES2) hardcell.cif ecpd12 v3.1*']
cif_file_version, ['(ES2) allpads.cif ecpd12 v3.1*']
Pad coordinate data

pygndins	GNDPY	Top	(57390,183600)	
copwrins	PWRCO	Top	(72390,183600)	
pypwrins	PWRPY	Top	(87390,183600)	
cogndins	GNDCO	Top	(102390,183600)	
m11ins	OPS1U	Top	(117390,183600)	a 55
m10ins	OPS1U	Top	(132390,183600)	a 54
m9ins	OPS1U	Top	(147390,183600)	a 53
m8ins	OPS1U	Top	(162390,183600)	a 52
clkins	IPS8B	Bottom	(71790,56100)	y 41
k11ins	IPS8G	Bottom	(86790,56100)	y 67
k0ins	IPS8G	Bottom	(101790,56100)	y 56
m0ins	OPS1U	Bottom	(116790,56100)	a 44
m1ins	OPS1U	Bottom	(131790,56100)	a 45
m2ins	OPS1U	Bottom	(146790,56100)	a 46
m3ins	OPS1U	Bottom	(161790,56100)	a 47
m4ins	OPS1U	Bottom	(176790,56100)	a 48
clrins	IPS8G	Left	(56100,166230)	y 38
k5ins	IPS8G	Left	(56100,148665)	y 61
k6ins	IPS8G	Left	(56100,131100)	y 62
k7ins	IPS8G	Left	(56100,113535)	y 63
k8ins	IPS8G	Left	(56100,95970)	y 64
k9ins	IPS8G	Left	(56100,78405)	y 65
k10ins	IPS8G	Left	(56100,60840)	y 66
m7ins	OPS1U	Right	(178260,178860)	a 51
m6ins	OPS1U	Right	(178260,163860)	a 50
cyins	OPS1U	Right	(178260,148860)	a 43
k4ins	IPS8G	Right	(178260,133860)	y 60
k3ins	IPS8G	Right	(178260,118860)	y 59
k2ins	IPS8G	Right	(178260,103860)	y 58
k1ins	IPS8G	Right	(178260,88860)	y 57
m5ins	OPS1U	Right	(178260,73860)	a 49

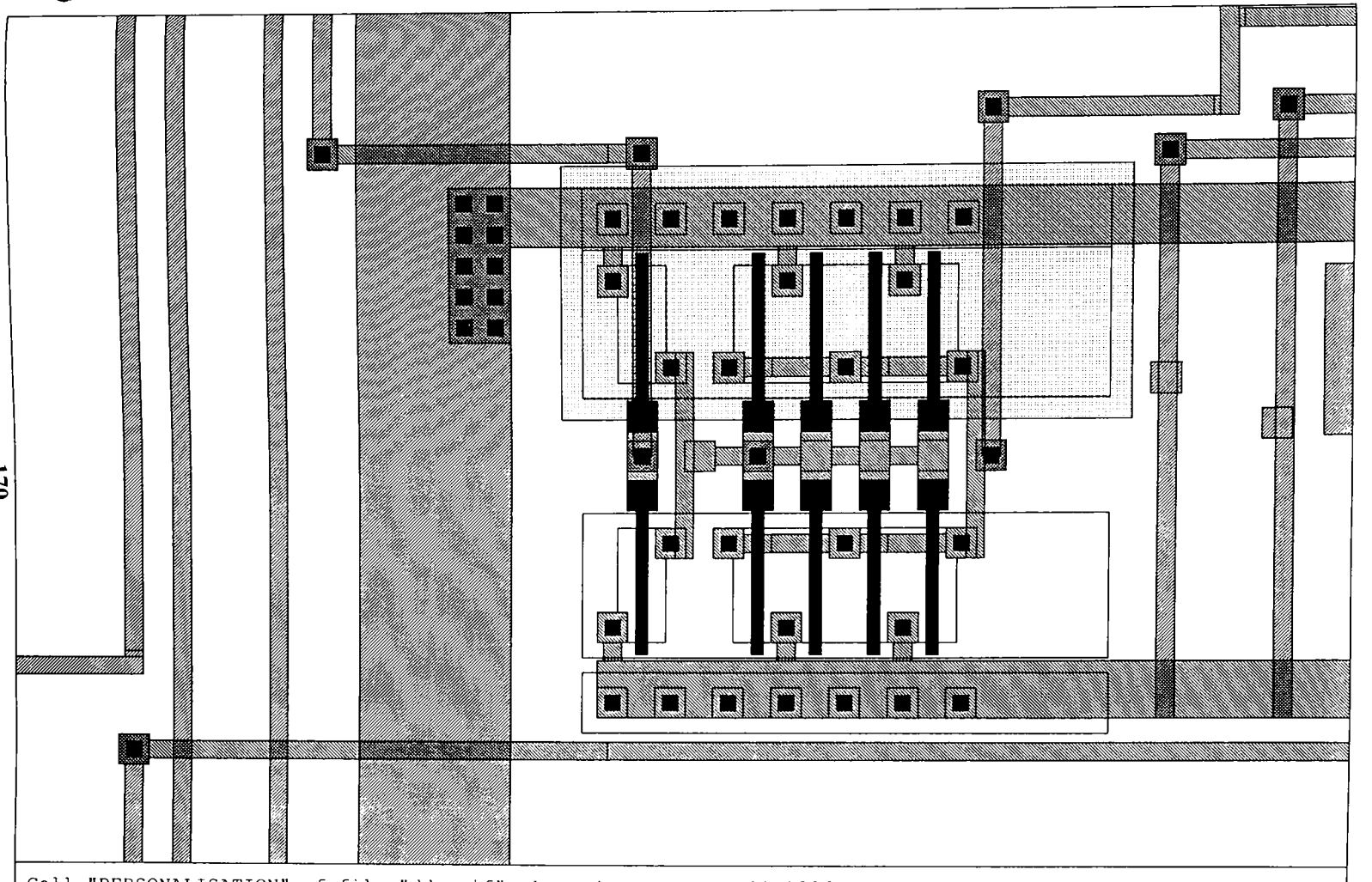
Chip Image Data
==========
Number of pads = 31
Array area (mm) = 1.19 x 1.23 = 1.46 sq.mm
Array area (mil) = 46.68 x 48.40 = 2259.24 sq.mil
Active chip area (mm) = 2.34 x 2.40 = 5.62 sq.mm
Active chip area (mil) = 92.27 x 94.37 = 8707.28 sq.mil
Die size (mm) = 2.55 x 2.61 = 6.66 sq.mm

```
        Die size (mil)    = 100.54 x 102.64   =  10318.69 sq.mil
Cpu time = 3.350 sec
```

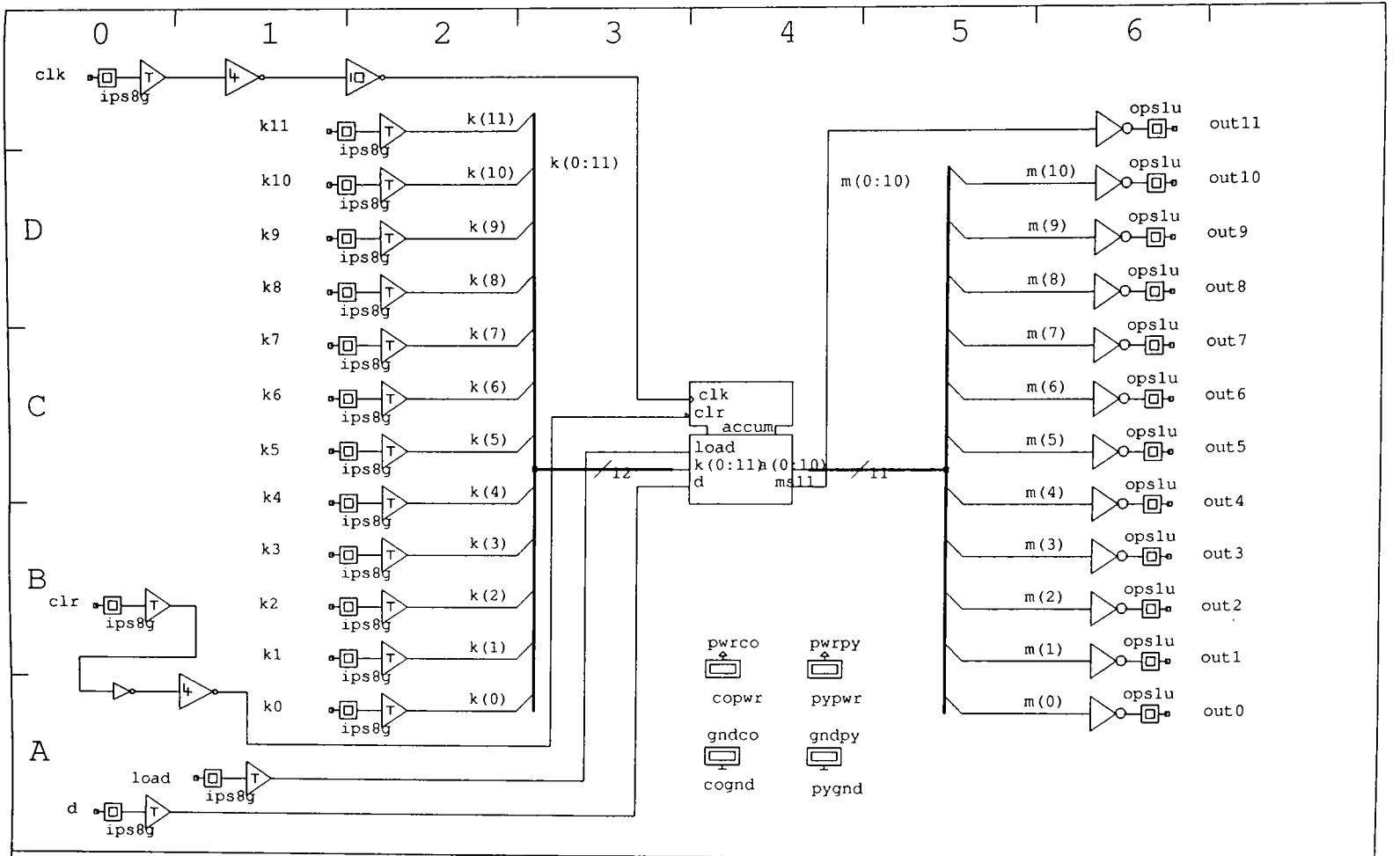
VSS
VCC
VCC
VSS
Cell "PERSONALISATION" of file "dds.cif" plotted on Thu May 30 1996 at 11:27:26

Cell "PERSONALISATION" of file "dds.cif" plotted on Tue Jun 11 1996 at 11:31:54

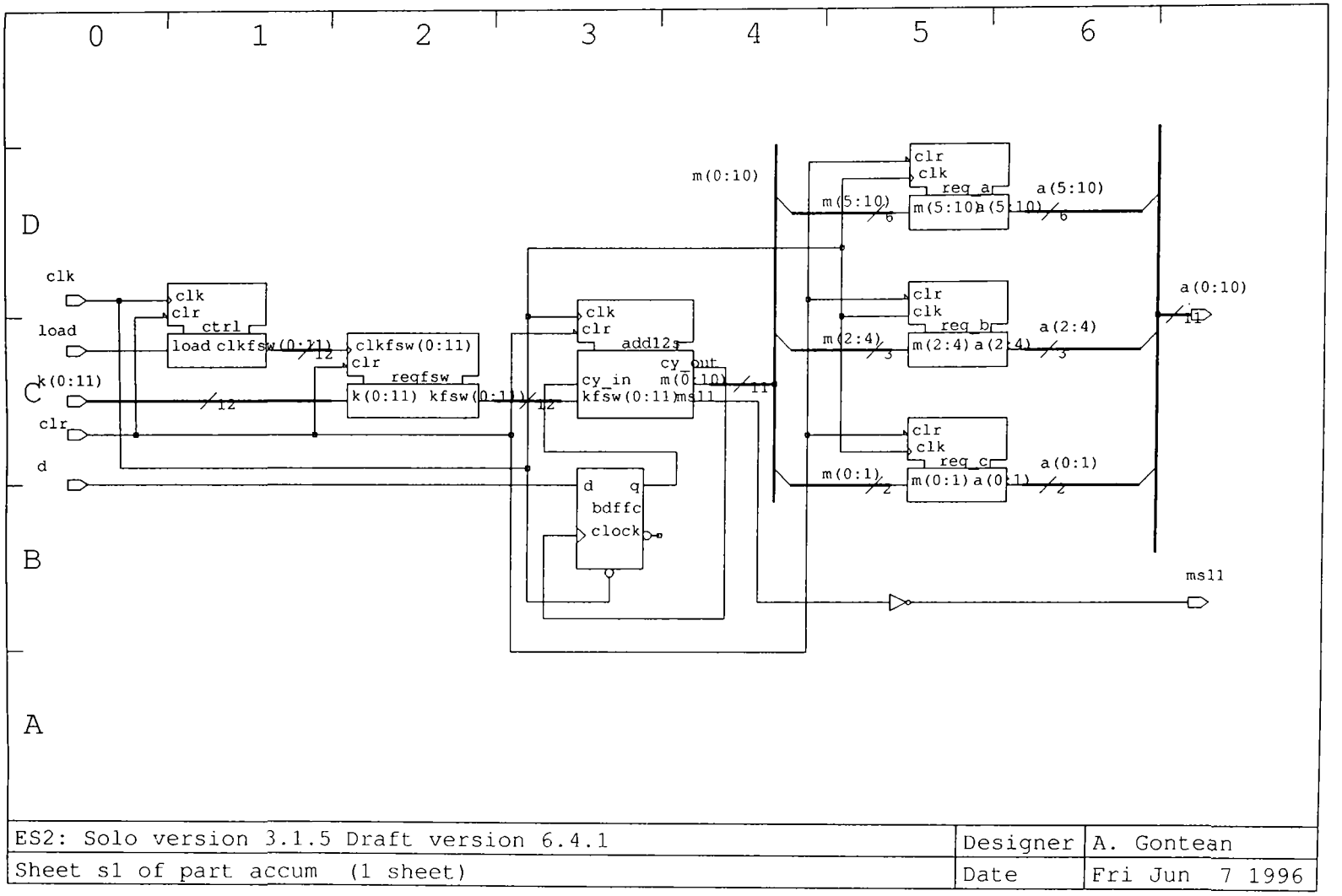
Cell "PERSONALISATION" of file "dds.cif" plotted on Tue Jun 11 1996 at 11:35:07



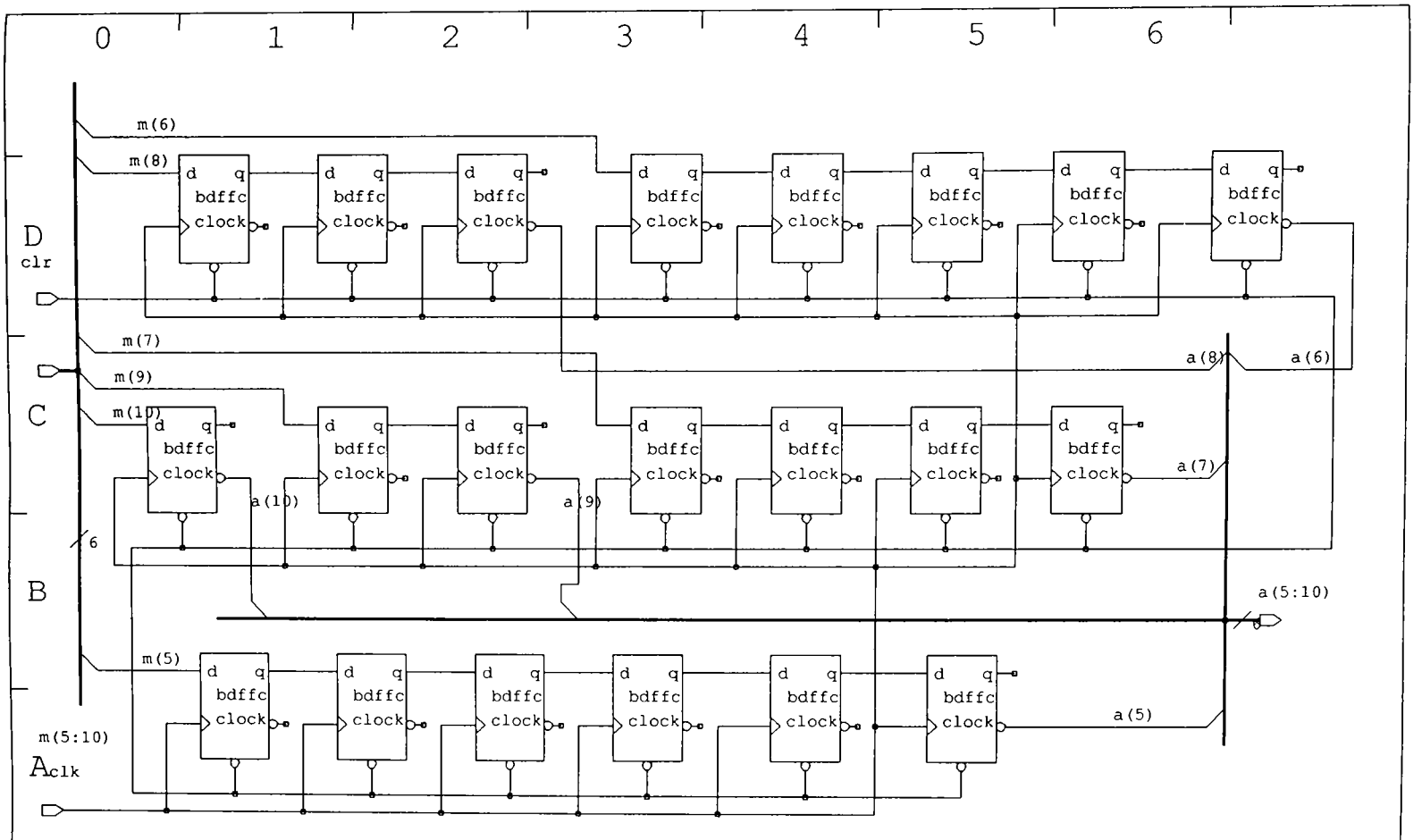
Cell "PERSONALISATION" of file "dds.cif" plotted on Tue Jun 11 1996 at 11:42:52



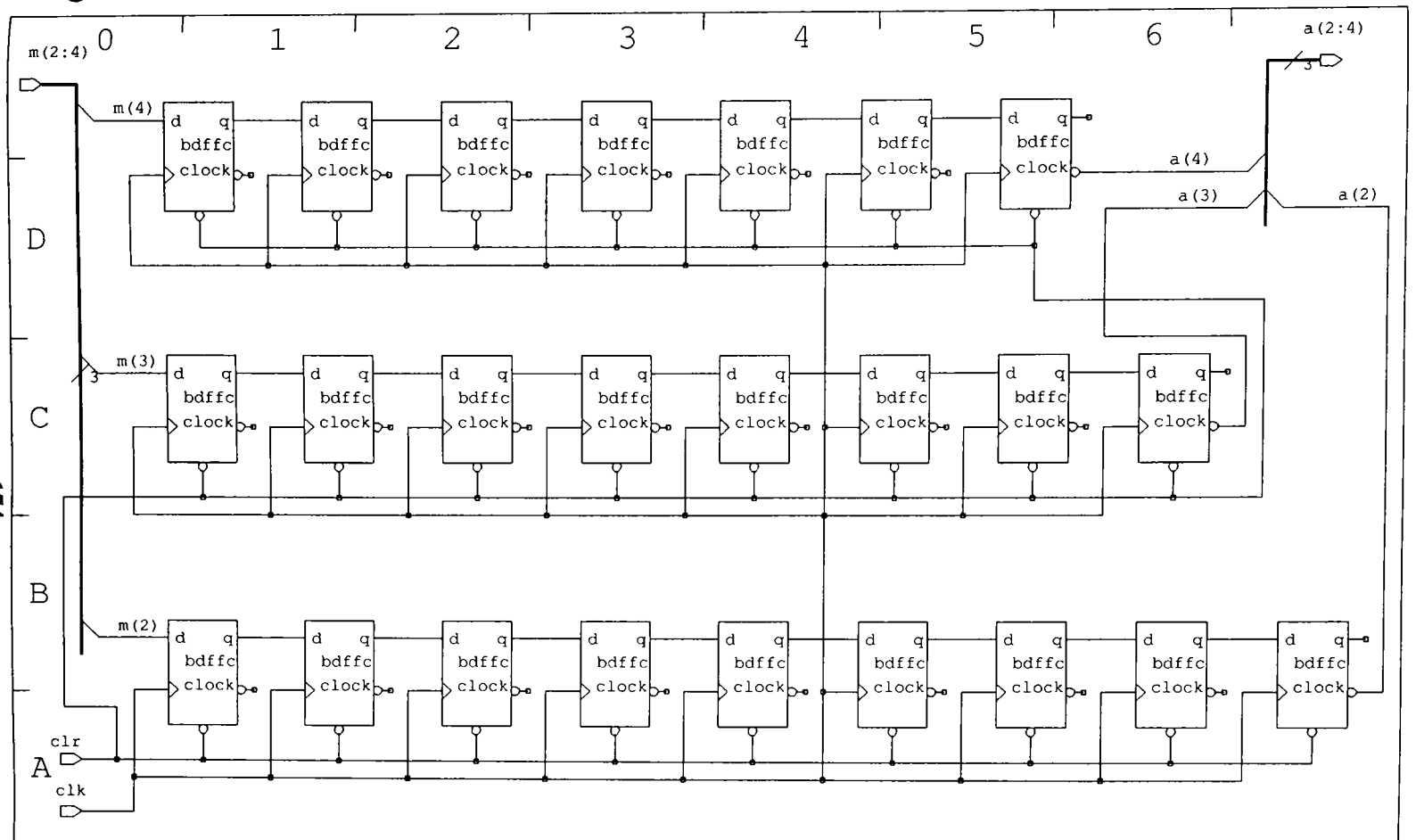
ES2: Solo version 3.1.5 Draft version 6.4.1	Desig er	A. Gontean
Sheet s1 of part dds2 (1 sheet)	Date	Fri Jun 7 1996



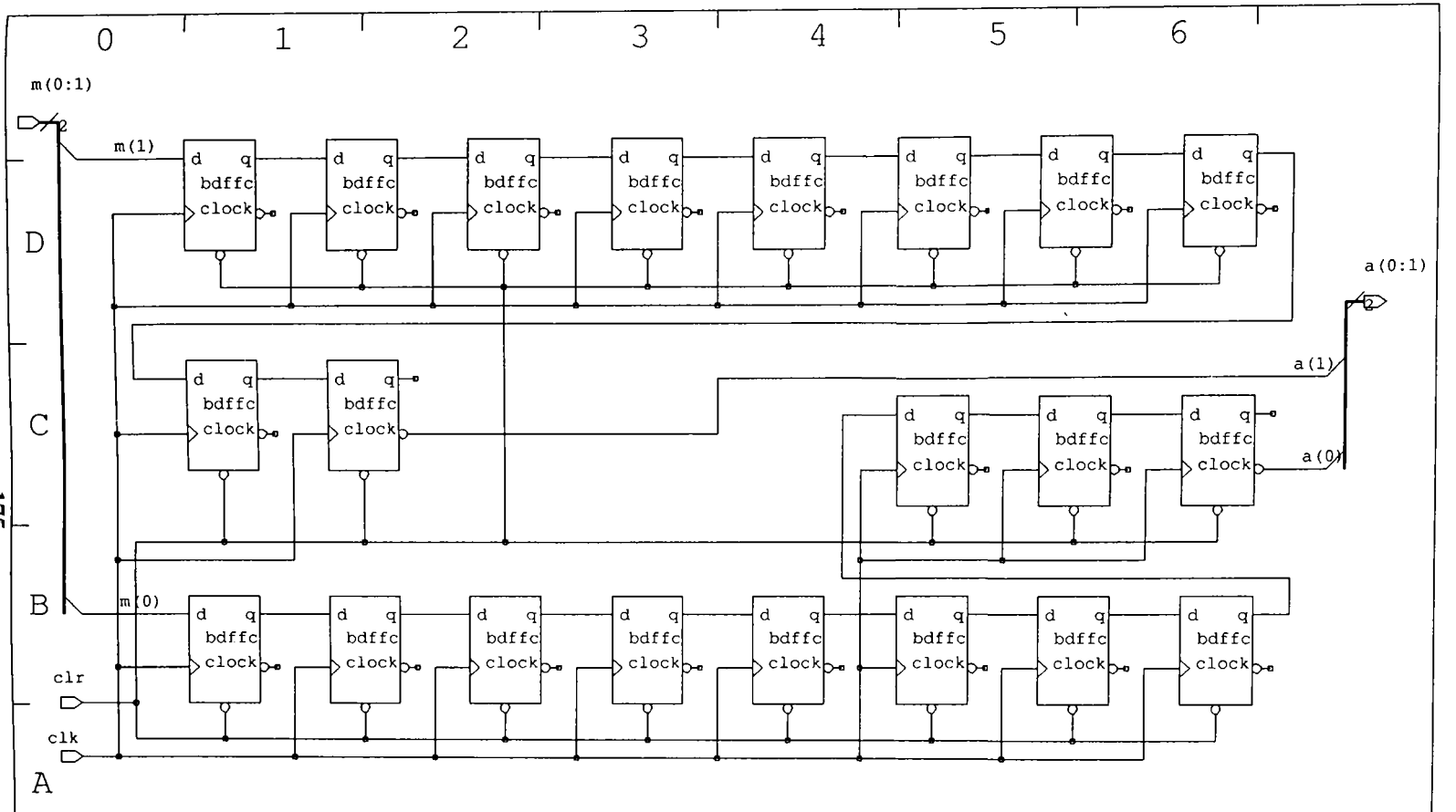
ES2: Solo version 3.1.5 Draft version 6.4.1		Designer	A. Gontean
Sheet s1 of part accum (1 sheet)		Date	Fri Jun 7 1996



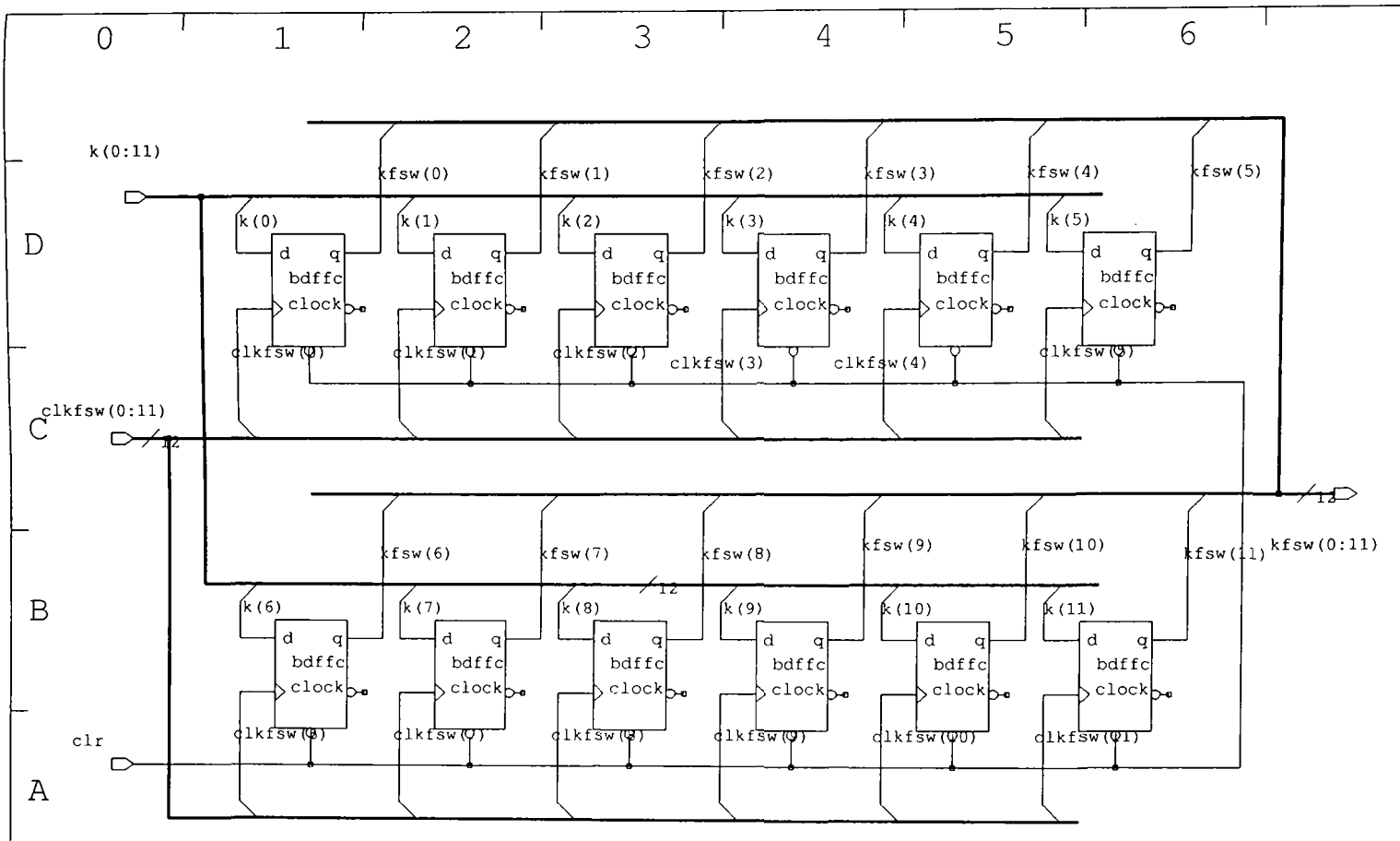
ES2: Solo version 3.1.5 Draft version 6.4.1		Designer	A. Gontean
Sheet s1 of part reg a (1 sheet)		Date	Fri Jun 7 1996



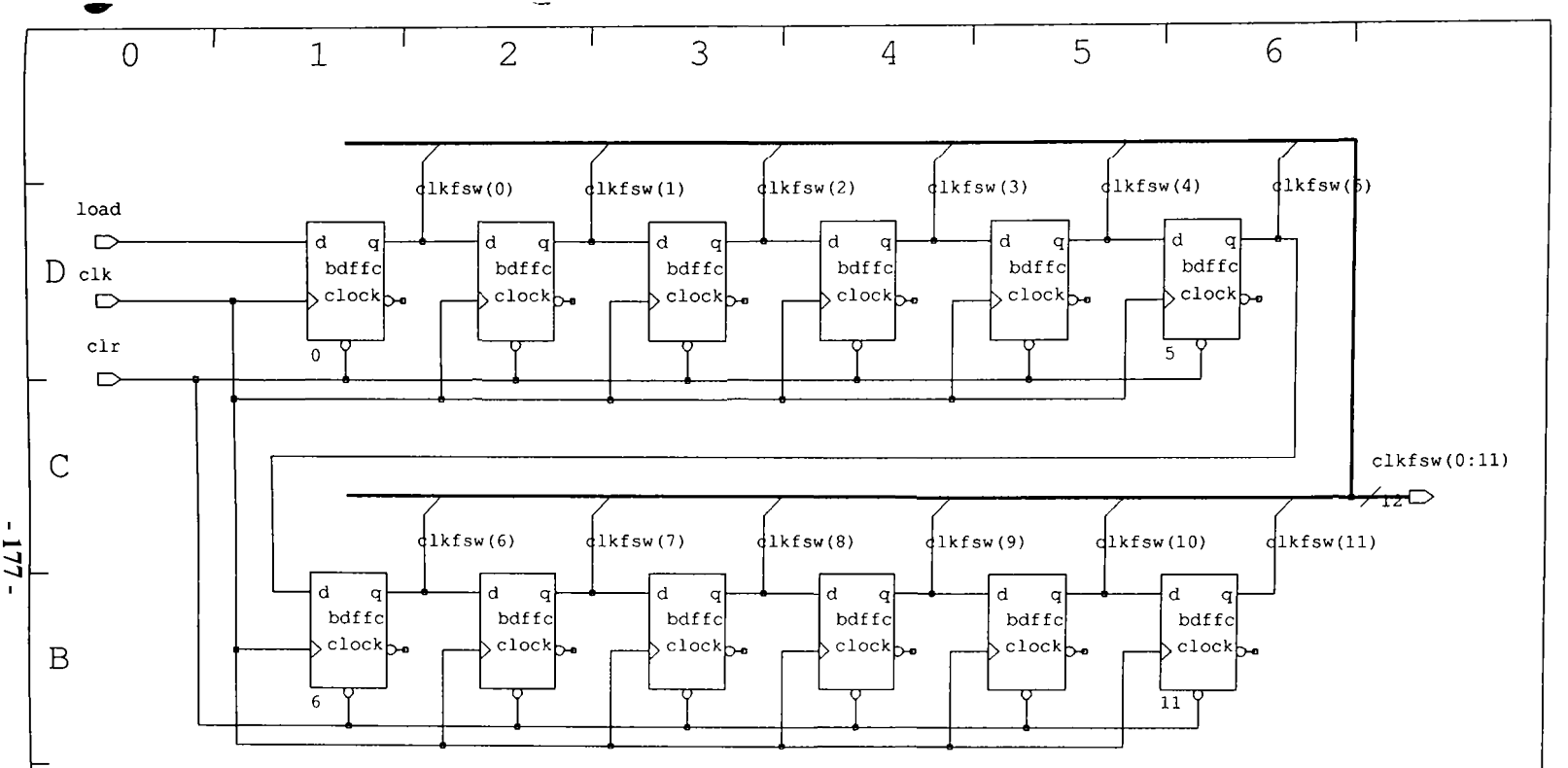
ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part reg_b (1 sheet)	Date	Fri Jun 7 1996



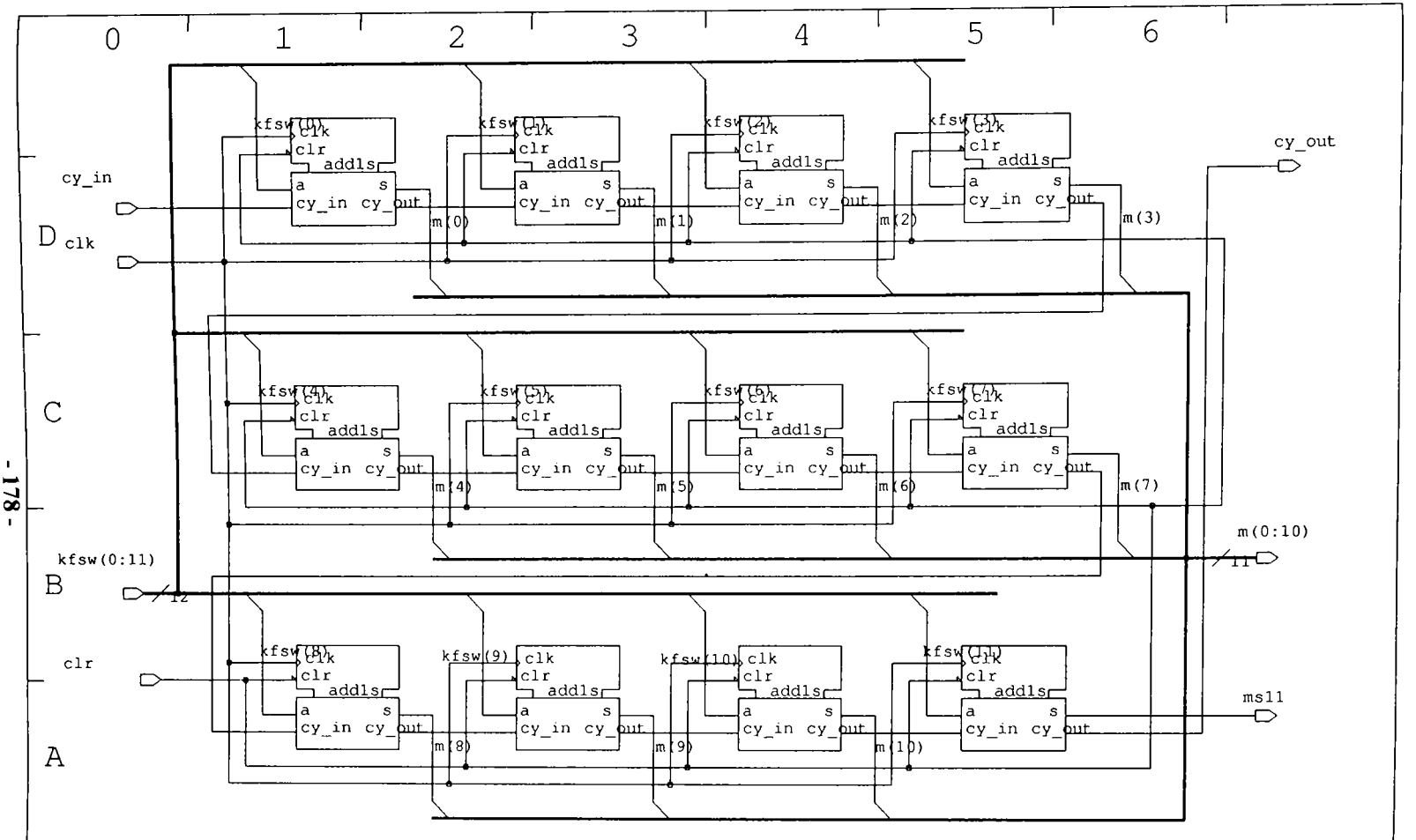
ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part reg c (1 sheet)	Date	Fri Jun 7 1996



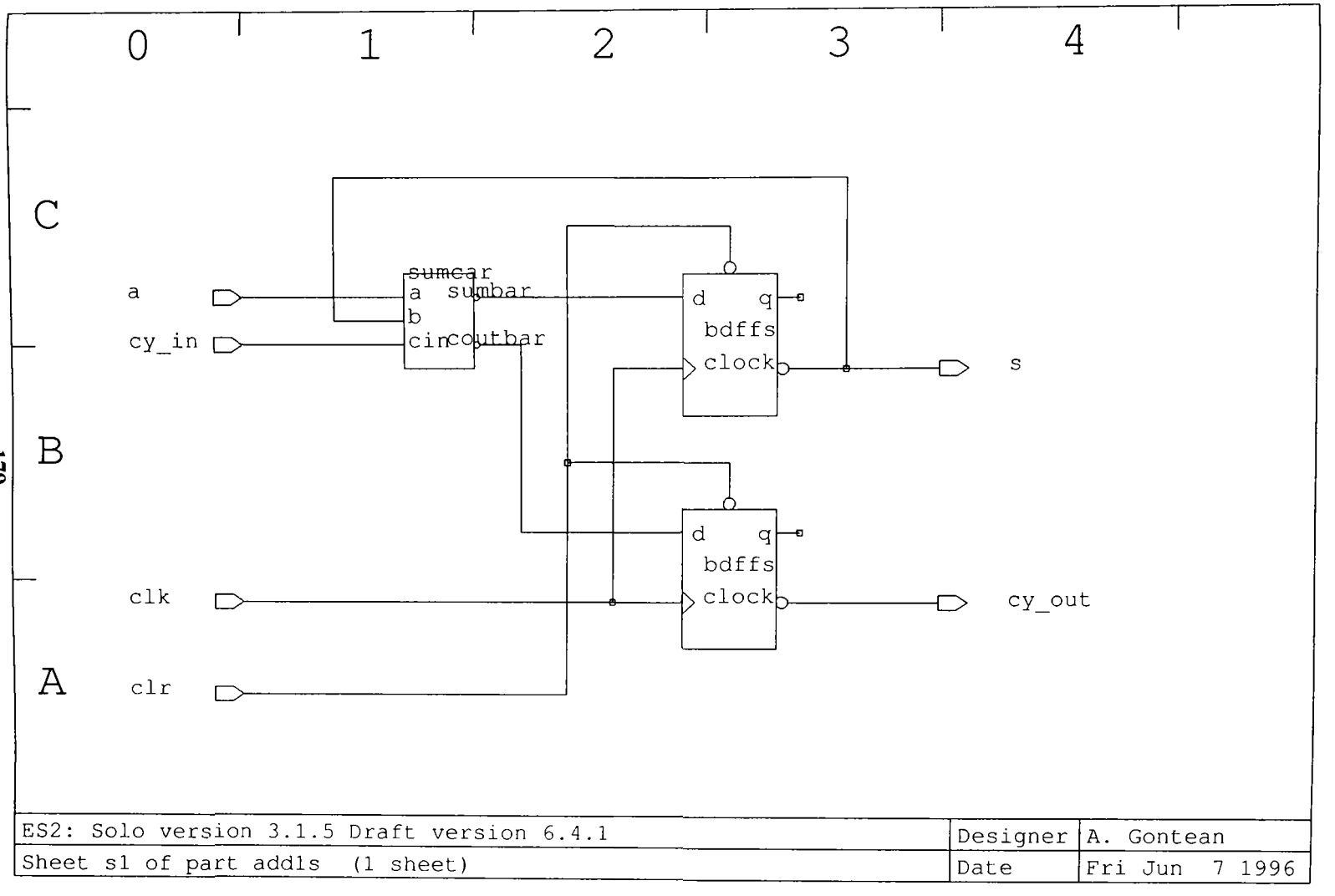
ES2: Solo version 3.1.5 Draft version 6.4.1		Designer	A. Gontean
Sheet s1 of part regfsw (1 sheet)		Date	Fri Jun 7 1996



ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part ctrl (1 sheet)	Date	Fri Jun 7 1996

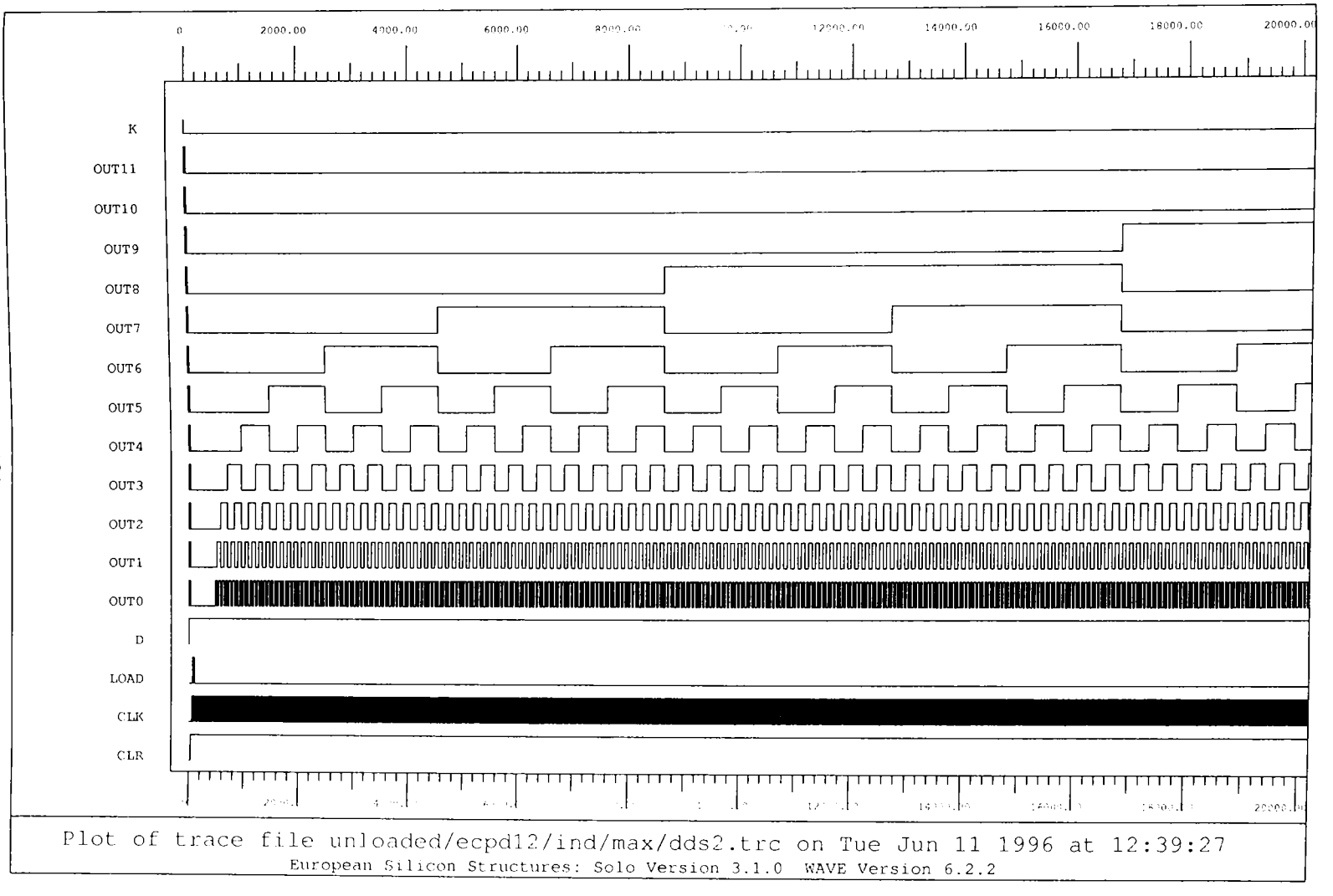


ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part add12s (1 sheet)	Date	Fri Jun 7 1996

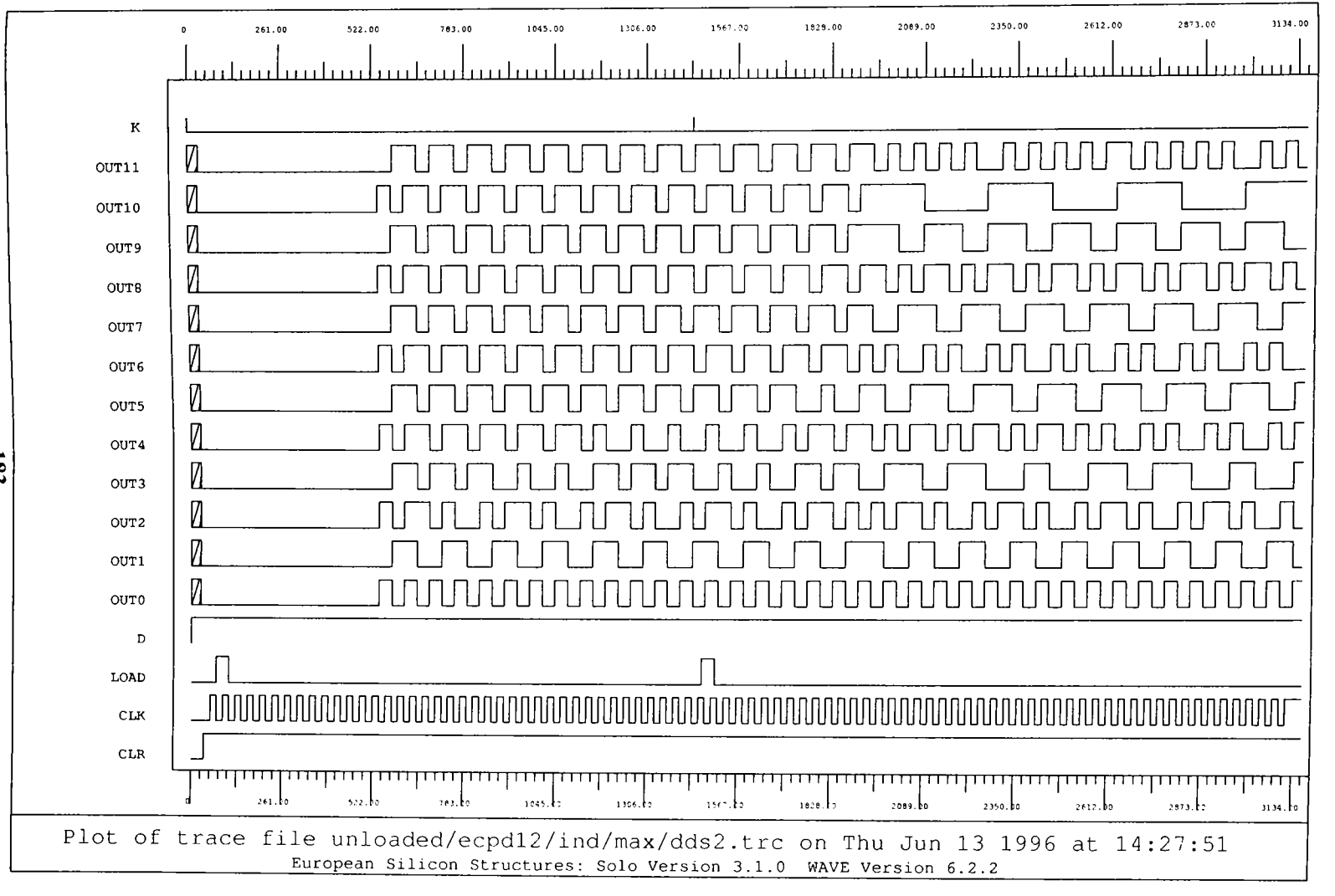


ES2: Solo version 3.1.5 Draft version 6.4.1		Designer	A. Gontean
Sheet s1 of part add1s (1 sheet)		Date	Fri Jun 7 1996

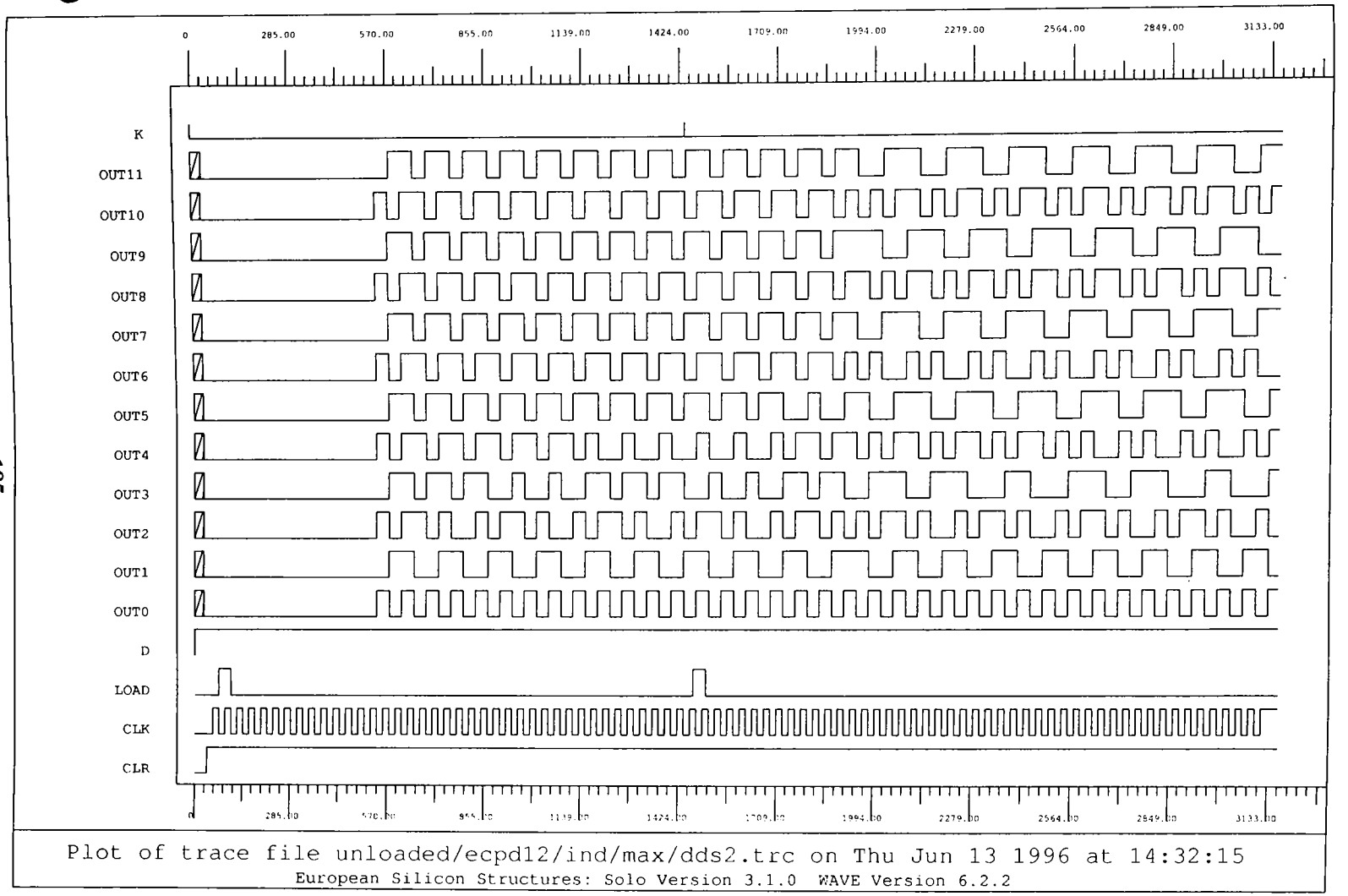
```
#step = 1s (cuanta de tiempo)
clk   = 1 step*2 -H
clk   = 1 step*1 [-H step -1 step]*4150 (semnal de test)
load = 1 step *4 -H step*2 -L
A     = H

A0   = H (step *120    H)
A1   = L (step *120 - H)
A2   = L (step *120 - L)
A3   = L (step *120 - H)
A4   = L (step *120 - L)
A5   = L (step *120 - H)
A6   = L (step *120 - L)
A7   = L (step *120    H)
A8   = L (step *120    L)
A9     L (step *120    H)
A10    L (step *120    L)
A11    L

out0
out1
out2
out3
out4
out5
out6
out7
out8
out9
out10
out11
```


0
2000.00
4000.00
6000.00
8000.00
12000.00
14000.00
16000.00
18000.00
20000.00
K
OUT11
OUT10
OUT9
OUT8
OUT7
OUT6
OUT5
OUT4
OUT3
OUT2
OUT1
OUT0
D
LOAD
CLK
CLR
Plot of trace file unloaded/ecpd12/ind/max/dds2.trc on Tue Jun 11 1996 at 12:39:27
European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

```
#step = 18 {cuanta de timp}
clr  = L step*2 =H
clk  = L step*3 [=H step -L step]*8 {semnal de test}
load = L step *4 =H step*2 -L step*75+10 =H step*2  L
d    = H
k0  = H step *80 = H
k1  = L step *80 = H
k2  = H step *80 = L
k3  = L step *80 = L
k4  = H step *80 = H
k5  = L step *80 = H
k6  = H step *80 = L
k7  = L step *80 = L
k8  = H step *80 = H
k9  = L step *80 = H
k10 = H step *80 = L
k11 = L

out0
out1
out2
out3
out4
out5
out6
out7
out8
out9
out10
out11
```


0
261.00
522.00
783.00
1045.00
1306.00
1567.00
1828.00
2089.00
2350.00
2612.00
2873.00
3134.00
K
OUT11
OUT10
OUT9
OUT8
OUT7
OUT6
OUT5
OUT4
OUT3
OUT2
OUT1
OUT0
D
LOAD
CLK
CLR
Plot of trace file unloaded/ecpd12/ind/max/dds2.trc on Thu Jun 13 1996 at 14:27:51
European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

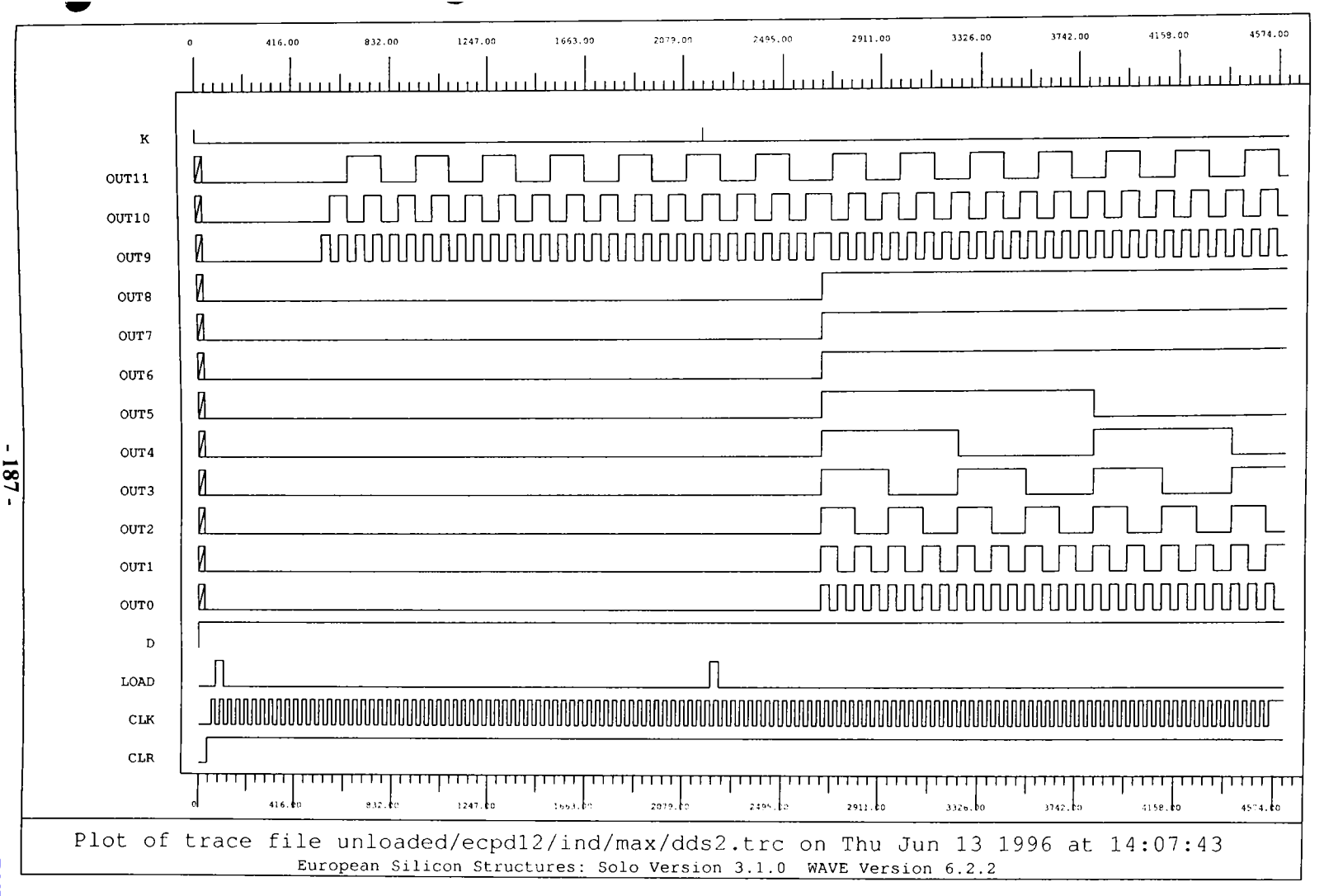
```
#step = 18 (cuanta de timp)
clr   = L step*2 =H
clk   = L step*3 [=H step =L step]*85 (semnal de tact)
load = L step *4 =H step*2 =L step*75+10 =H step*2 =L
d     = H
k0   = H step *80 = H
k1   = L step *90 = H
k2   = H step *80 = L
k3   = L step *80 = L
k4   = H step *80 = H
k5   = L step *80 = H
k6   = H step *80 = L
k7   = L step *80 = L
k8   = H step *80 = H
k9   = L step *80 = H
k10 = H step *80 = H
k11 = L

out0
out1
out2
out3
out4
out5
out6
out7
out8
out9
out10
out11
```



Plot of trace file unloaded/ecpd12/ind/max/dds2.trc on Thu Jun 13 1996 at 14:32:15

European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

```
#step = 18 (cuanta de timp)
clr   = L step*2 =H
clk   = L step*3 [=H step =L step]*125 (semnal de tact)
load  = L step *4 =H step*2 =L step*116 =H step*2 =L
d     = H
k0   = L step *120 = H
k1   = L step *120 = H
k2   = L step *120 = H
k3   = L step *120 = H
k4   = L step *120 = H
k5   = L step *120 = H
k6   = L step *120 = H
k7   = L step *120 = H
k8   = L step *120 = H
k9   = H step *120 = L
k10  = L
k11  = L

out0
out1
out2
out3
out4
out5
out6
out7
out8
out9
out10
out11
```



Plot of trace file unloaded/ecpd12/ind/max/dds2.trc on Thu Jun 13 1996 at 14:07:43

European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

```
{       Position file for MODEL file dds2.idl
{-------------------------------------------

MODEL version 6.2.3     Fri Jun  7 1996    12:40:43
PLACE version 6.3       Fri Jun  7 1996    12:41:34

Number of columns       2

Column  1 -  number of rows:   11
          -  stages per row:   96
Column  2 -  number of rows:   11
          -  stages per row:   96


{ P = Position, from start of row (in direction stated).
{ S = Size, in terms of stages.
{ H = Height, in terms of rows. By default 1 row.
{ T = Type, the type of the library element.
{ N = Nets, the net numbers connected to the part.

{ The Diagram below shows the Floor Plan Parts.
{
{ Scale is set to 8 Stages per 3 characters
{
{    +-----------------------------------++- -----------------------------------+
{ 11 |                                   ||                                   |
{    +                                   ++                                   +
{ 10 |                                   ||                                   |
{    +                                   ++                                   +
{  9 |                                   ||                                   |
{    +                                   ++                                   +
{  8 |                                   ||                                   |
{    +                                   ++                                   +
{  7 |                                   ||                                   |
{    +                                   ++                                   +
{  6 |                 c                 ||                 c                 |
{    +                                   ++                                   +
{  5 |                                   ||                                   |
{    +                                   ++                                   +
{  4 |                                   ||                                   |
{    +                                   ++                                   +
{  3 |                                   ||                                   |
{    +                                   ++                                   +
{  2 |                                   ||                                   |
{    +  +                                ++                  ---+---+         +
{  1 |**|                                ||                   d  | e |         |
{    +  +-----------  -------------------++-                     +   +         +
{
{ Index of Parts shown:
{
{      Instance Name                          Model Name
{  c   /DDS2/S1C3                             accum
{  d - /DDS2/S1E2Y4                           buffer10
{  e   /DDS2/S1E1                             buffer4
{
{
{ Index of Parts not shown:
{
{      Instance Name                          Model Name
{  a   /DDS2/S1B0                             net
{  b   /DDS2/S1A1                             buffer4
{
{ Note: Any areas with the name '\\' consist of blank stages,
{       whilst any areas named '**' are too small to resolve
{

{ Column 1 Row 1,  L -> R, wastage  7 stages

/DDS2/S1B0                                    { P   1, S  2, T not, N  42
                                              {  43
/DDS2/S1A1                                    { P   3, S  5, T buffer4, N
                                              {  43 44

{ Column 1 Row 2,  R -> L, wastage  0 stages

{ Column 1 Row 3,  L -> R, wastage  0 stages

{ Column 1 Row 4,  R -> L, wastage  0 stages

{ Column 1 Row 5,      -> R, wastage  0 stages

{ Column 1 Row 6,  R   -> L, wastage  0 stages

{ Column 1 Row 7,  L -> R, wastage  0 stages

{ Column 1 Row 8,      -> L, wastage  0 stages

{ Column 1 Row 9,  L -> R, wastage  4 stages

{ Column 1 Row 10,     -> L, wastage  4 stages

{ Column 1 Row 11,  L  -> R, wastage  4 stages

{ Column 2 Row 11,  L -> R, wastage  4 stages

  Column 2 Row 10,  R -> L, wastage  4 stages
```

{ Column 2 Row 9, L -> R, wastage 4 stages

{ Column 2 Row 8, R -> L, wastage 0 stages

{ Column 2 Row 7, L -> R, wastage 0 stages

{ Column 2 Row 6, R -> L, wastage 0 stages

{ Column 2 Row 5, L -> R, wastage 0 stages

{ Column 2 Row 4, R -> L, wastage 0 stages

{ Column 2 Row 3, L -> R, wastage 0 stages

{ Column 2 Row 2, R -> L, wastage 0 stages

{ Column 2 Row 1, L -> R, wastage 46 stages

/DDS2/S1C3 { F 9, S 2010, T accum
{ , N 46 59 60 61 62 63 64
{ 65 66 67 68 69 70 45 44
{ 39 47 48 49 50 51 52 53
{ 54 55 56 57 58

/DDS2/S1E2Y4 { P 41, S 13, T buffer10
{ , N 40 39

/DDS2/S1E1 { P 66, S 5, T buffer4, N
{ 41 40

1	k9ins	IPS8G	y	k(9)
2	k8ins	IPS8G	y	k(8)
3	k7ins	IPS8G	y	k(7)
4	k6ins	IPS8G	y	k(6)
5	k5ins	IPS8G	y	k(5)
6	k4ins	IPS8G	y	k(4)
7	k3ins	IPS8G	y	k(3)
8	k2ins	IPS8G	y	k(2)

BOTTOM pad assignment

	ID	Type	Terminal	Signal
1	loadins	IPS8G	y	slc3x1
2	dins	IPS8G	y	slc3x2
3	outllins	OPS1U	a	slc3x5
4	out0ins	OPS1U	a	m(0)
5	outlins	OPS1U	a	m(1)
6	out2ins	OPS1U	a	m(2)
7	out3ins	OPS1U	a	m(3)
8	out4ins	OPS1U	a	m(4)

LEFT pad assignment

	ID	Type	Terminal	Signal
1	k10ins	IPS8G	y	k(10)
2	k11ins	IPS8G	y	k(11)
3	clrins	IPS8G	y	slblxl
4	clkins	IPS8G	y	slelxl
5	cogndins	GNDCO		
6	pygndins	GNDPY		
7	pypwrins	PWRFY		
8	copwrins	PWRCO		

RIGHT pad assignment

	ID	Type	Terminal	Signal
1	klins	IPS8G	y	k(1)
2	k0ins	IPS8G	y	k(0)
3	out10ins	OPS1U	a	m(10)
4	out9ins	OPS1U	a	m(9)
5	out8ins	OPS1U	a	m(8)
6	out7ins	OPS1U	a	m(7)
7	out6ins	OPS1U	a	m(6)
8	out5ins	OPS1U	a	m(5)

Cpu time = 4.330 sec

* 5 Solo version 3.1.2; DRAW version 6.3.2 Fri Jun 7 1996 12:42:32

1430412760

Inputs RTE file : dds2.rte

Outputs : CIF file : dds2.cif
LDA file : dds2.ida

Options : /LOAD/OPTIMISED/CIF/PADS/TWOLAYER

Private process file : /apts/solo1400/ecpd12/process.ppf
Public process file : /apts/solo1400/ecpd12/indl.prc

cif_file_version, ['(ES2) hardcell.cif ecpd12 v3.1*']
cif_file_version, ['(ES2) allpads.cif ecpd12 v3.1*']
Pad coordinate data

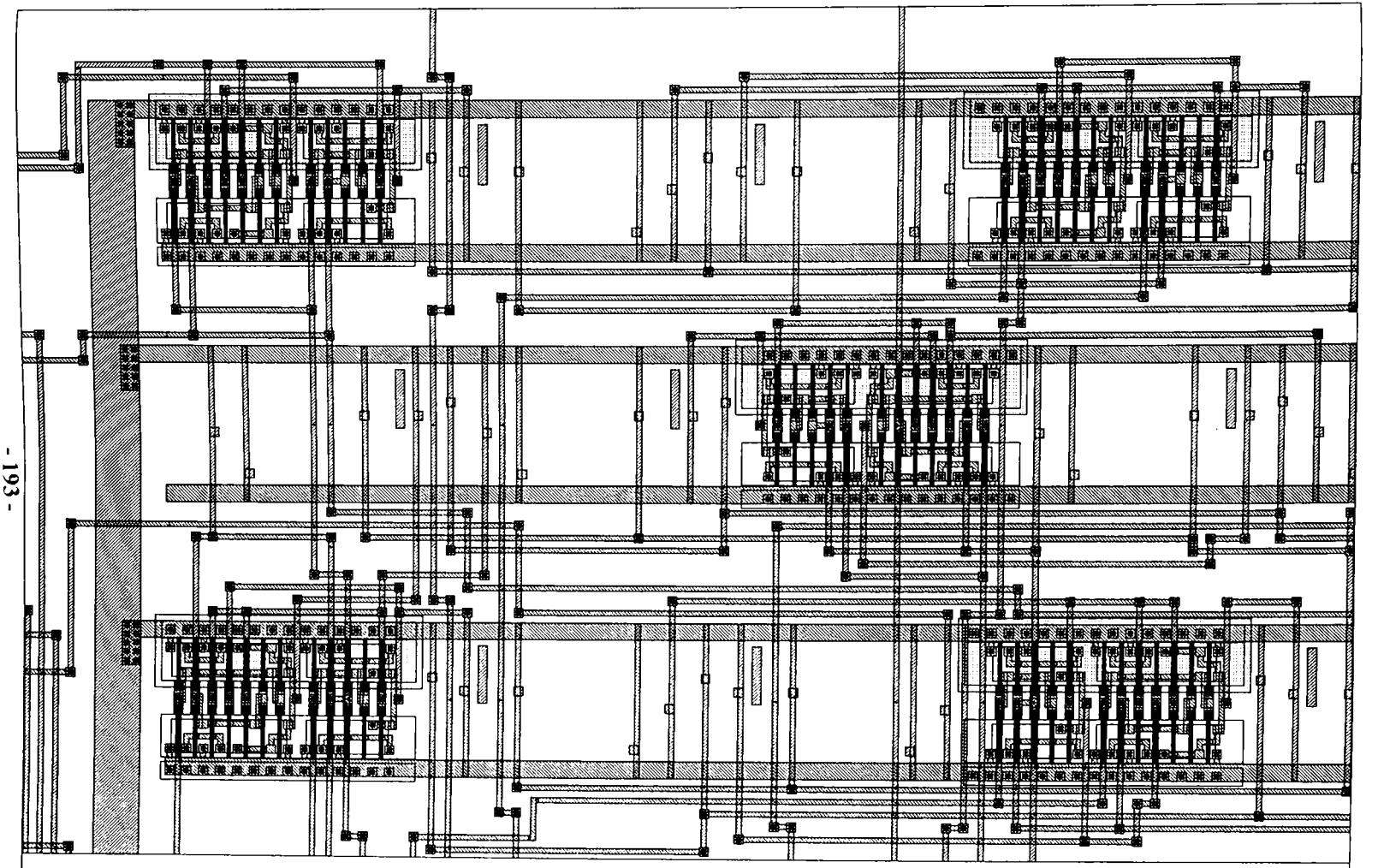
k9ins	IPS8G	Top	(57810,183600)	y 68
k8ins	IPS8G	Top	(74280,183600)	y 67
k7ins	IPS8G	Top	(90765,183600)	y 66
k6ins	IPS8G	Top	(107250,183600)	y 65
k5ins	IPS8G	Top	(123735,183600)	y 64
k4ins	IPS8G	Top	(140220,183600)	y 63
k3ins	IPS8G	Top	(156705,183600)	y 62
k2ins	IPS8G	Top	(173190,183600)	y 61
loadins	IPS8G	Bottom	(72210,56100)	y 45
dins	IPS8G	Bottom	(88680,56100)	y 46
outllins	OPS1U	Bottom	(105165,56100)	a 47
out0ins	OPS1U	Bottom	(121650,56100)	a 48
outlins	OPS1U	Bottom	(138135,56100)	a 49
out2ins	OPS1U	Bottom	(154620,56100)	a 50
out3ins	OPS1U	Bottom	(171105,56100)	a 51
out4ins	OPS1U	Bottom	(187590,56100)	a 52
k10ins	IPS8G	Left	(56100,166230)	y 69
k11ins	IPS8G	Left	(56100,151230)	y 70
clrins	IPS8G	Left	(56100,136230)	y 42
clkins	IPS8G	Left	(56100,121230)	y 41
cogndins	GNDCO	Left	(56100,106230)	
pygndins	GNDPY	Left	(56100,91230)	
pypwrins	PWRFY	Left	(56100,76230)	
copwrins	PWRCO	Left	(56100,61230)	
klins	IPS8G	Right	(189300,178860)	y 60
k0ins	IPS8G	Right	(189300,163860)	y 59
out10ins	OPS1U	Right	(189300,148860)	a 58
out9ins	OPS1U	Right	(189300,133860)	a 57
out8ins	OPS1U	Right	(189300,118860)	a 56
out7ins	OPS1U	Right	(189300,103860)	a 55
out6ins	OPS1U	Right	(189300,88860)	a 54
out5ins	OPS1U	Right	(189300,73860)	a 53

Chip Image Data

Number of pads = 32
Array area (mm) = 1.30 x 1.23 = 1.59 sq.mm
Array area (mil) = 51.02 x 48.40 = 2469.61 sq.mil
Active chip area (mm) = 2.45 x 2.40 = 5.88 sq.mm
Active chip area (mil) = 96.61 x 94.37 = 9117.45 sq.mil

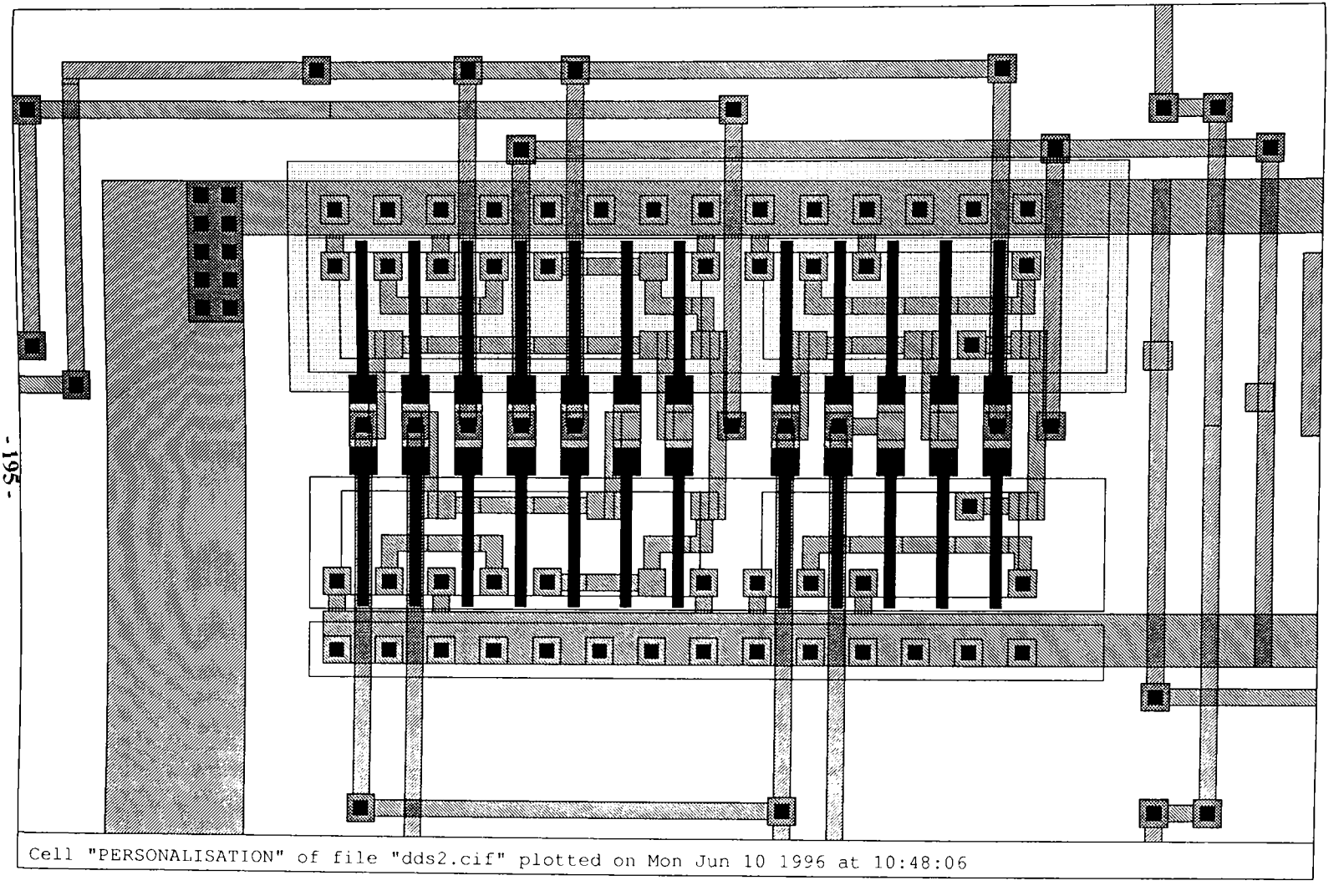
```
        Die size (mm)      = 2.66 x 2.61   =  6.95 sq.mm
        Die size (mil)       104.88 x 102.64   =  10764.80 sq.mil
Cpu time = 3.900 sec
```

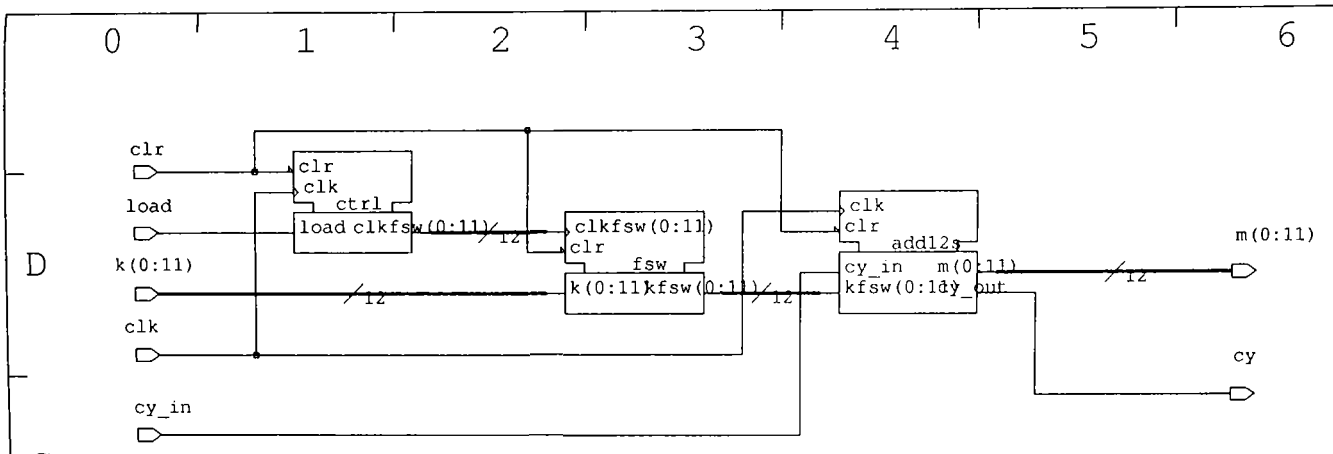
VSS-
VSS-
VCC+
VCC+
Cell "PERSONALISATION" of file "dds2.cif" plotted on Fri Jun 7 1996 at 14:16:57



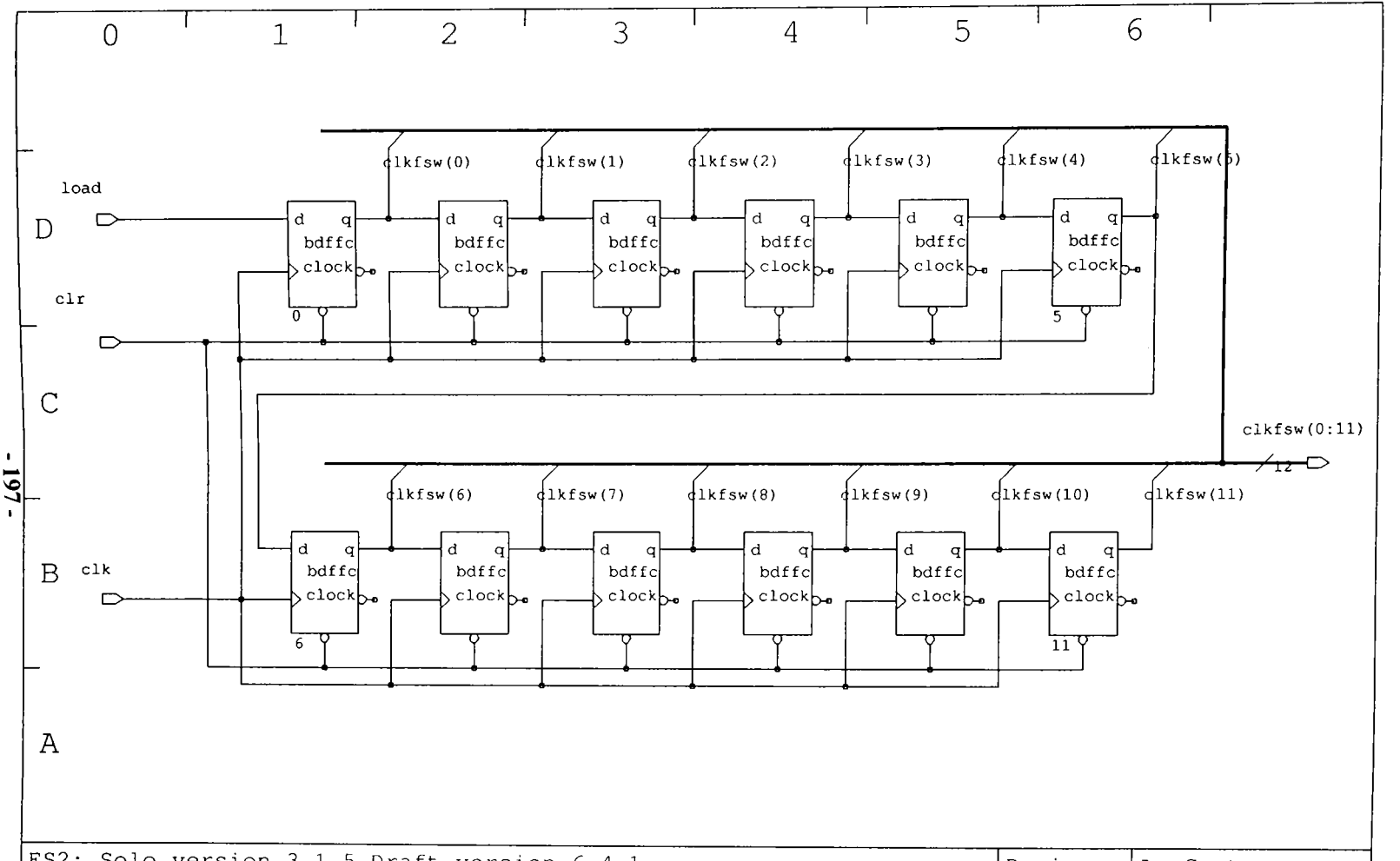
Cell "PERSONALISATION" of file "dds2.cif" plotted on Mon Jun 10 1996 at 10:44:09

Cell "PERSONALISATION" of file "dds2.cif" plotted on Fri Jun 7 1996 at 14:37:32

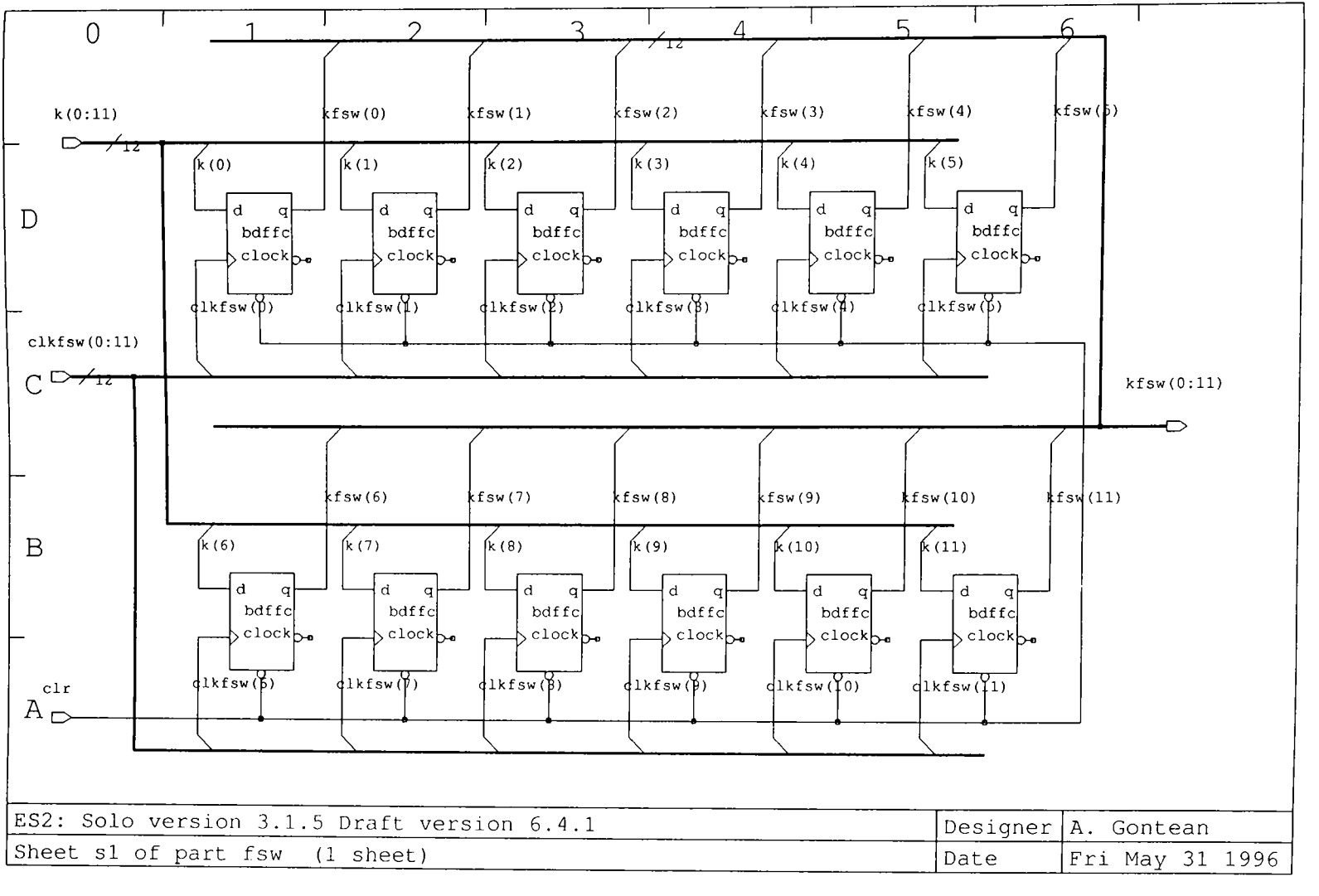

Cell "PERSONALISATION" of file "dds2.cif" plotted on Mon Jun 10 1996 at 10:48:06



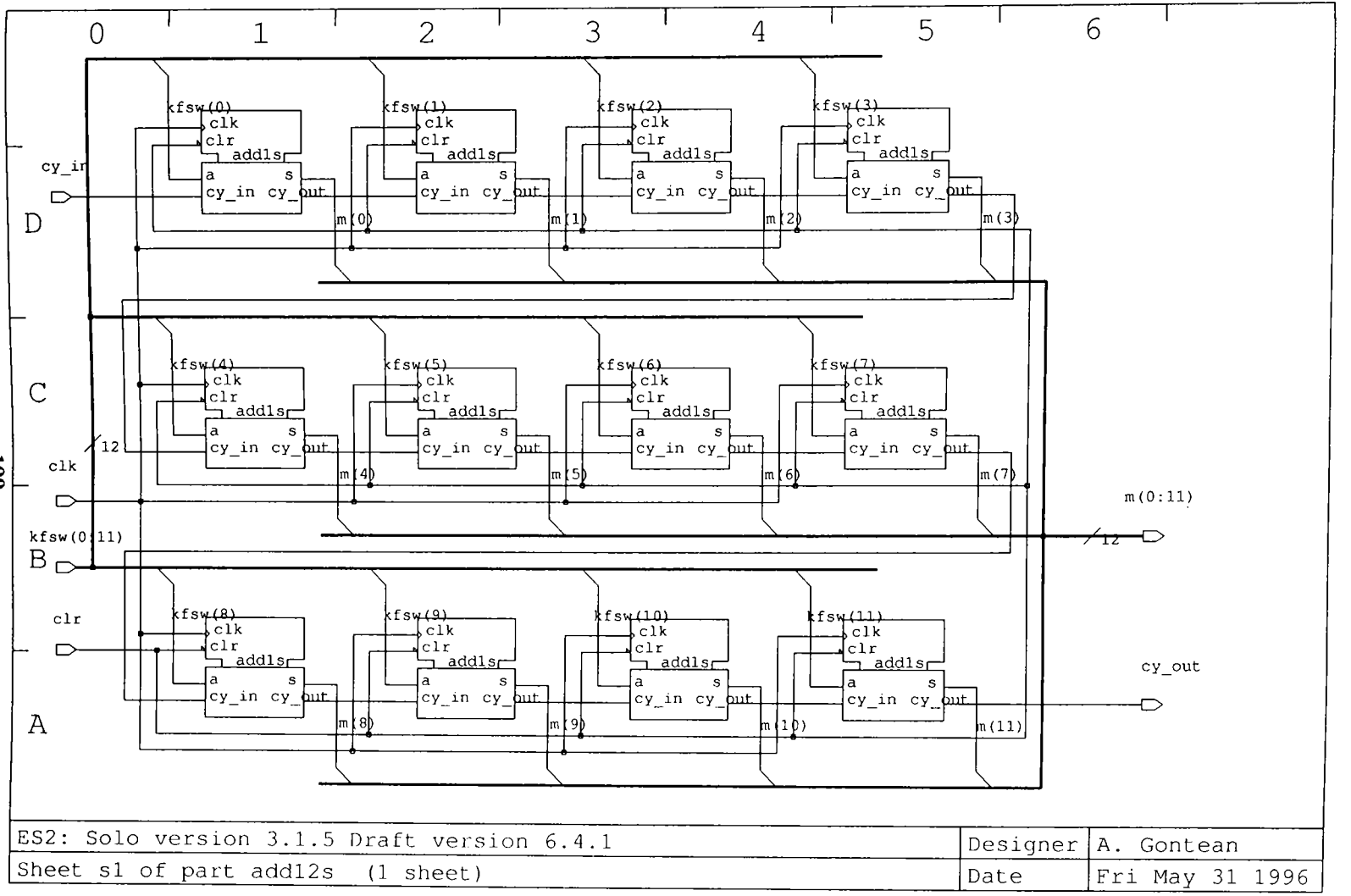
ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part accu (1 sheet)	Date	Fri May 31 1996



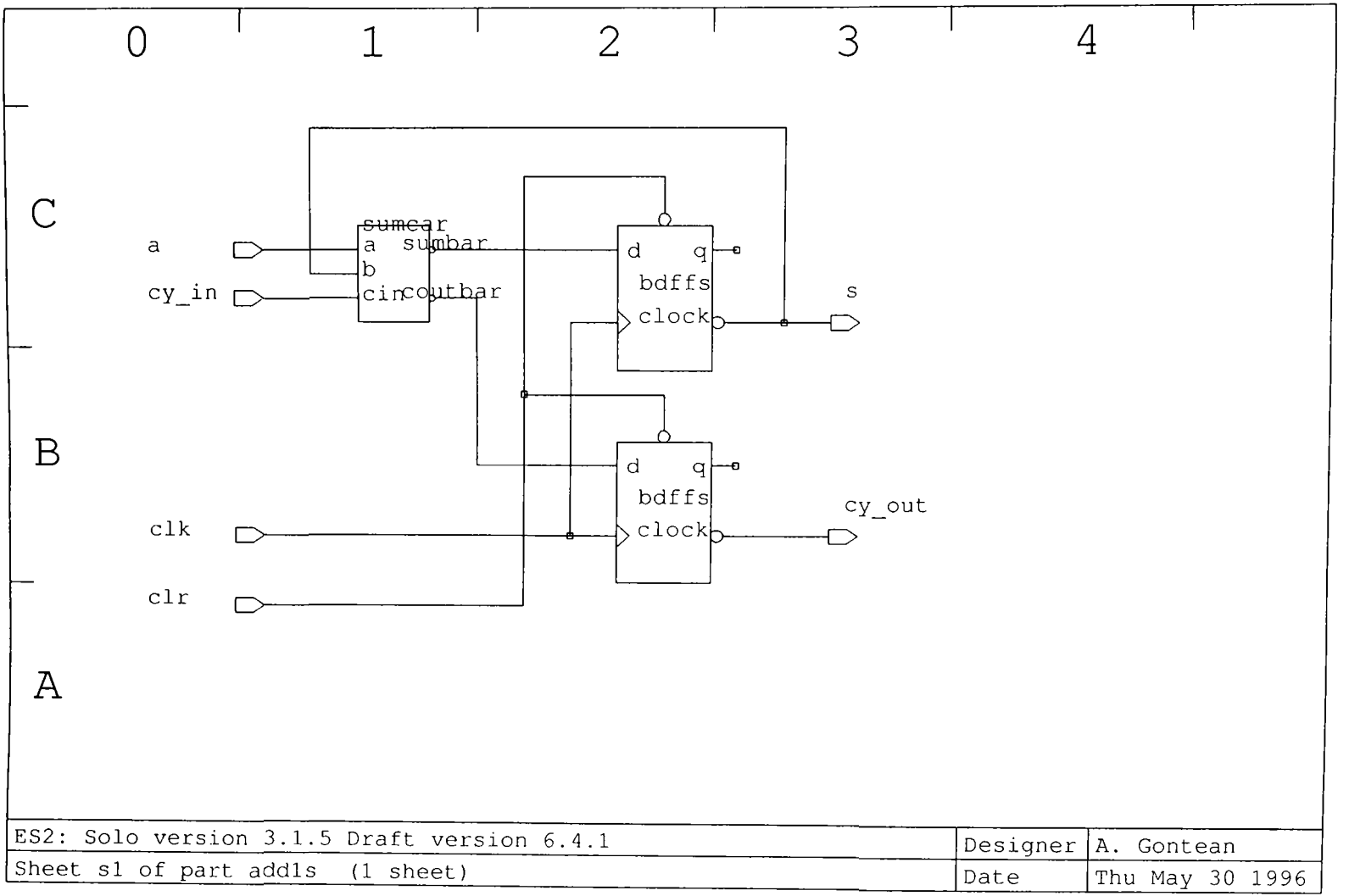
ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part ctrl (1 sheet)	Date	Fri May 31 1996



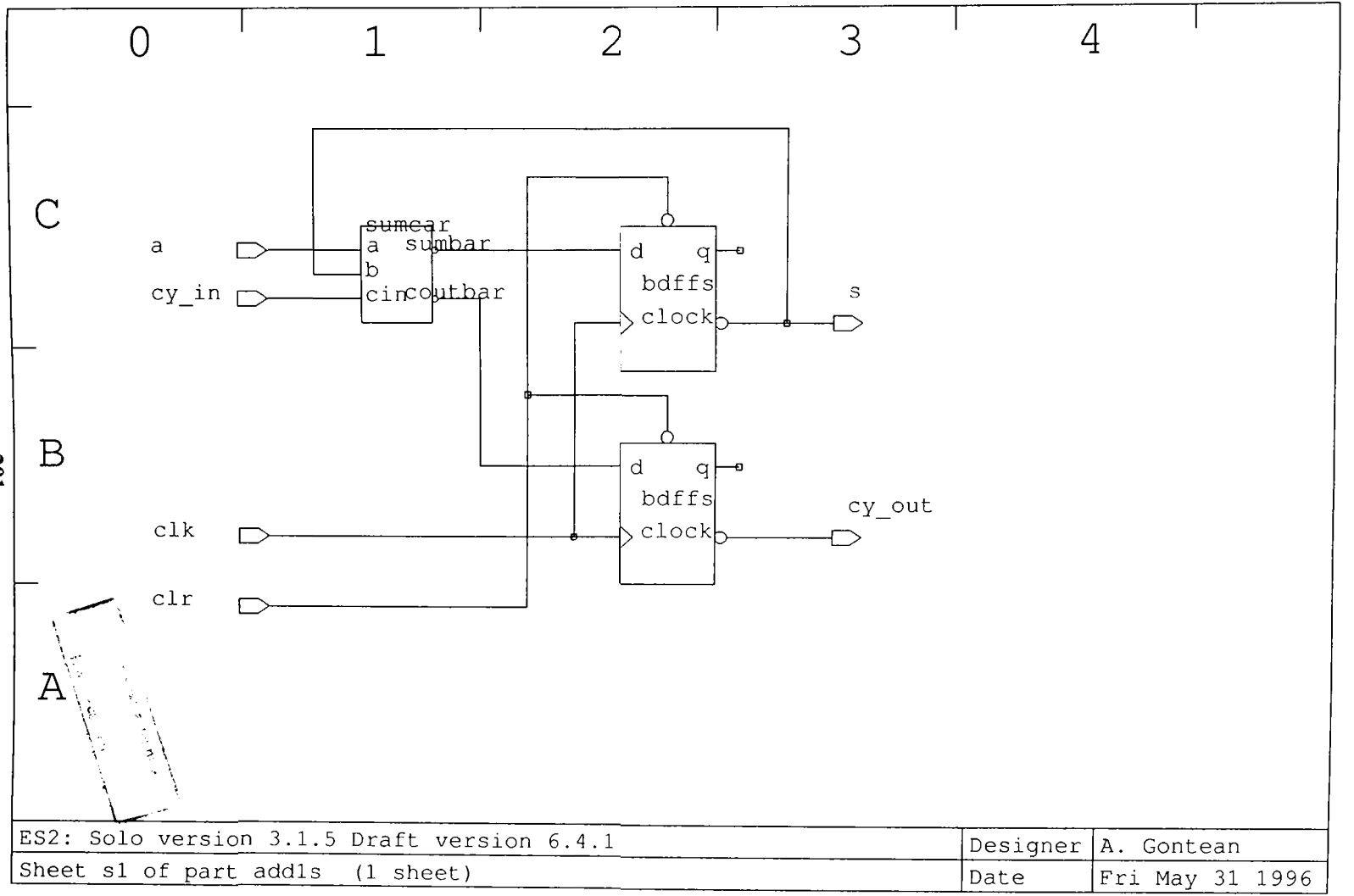
ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part fsw (1 sheet)	Date	Fri May 31 1996



ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part add12s (1 sheet)	Date	Fri May 31 1996

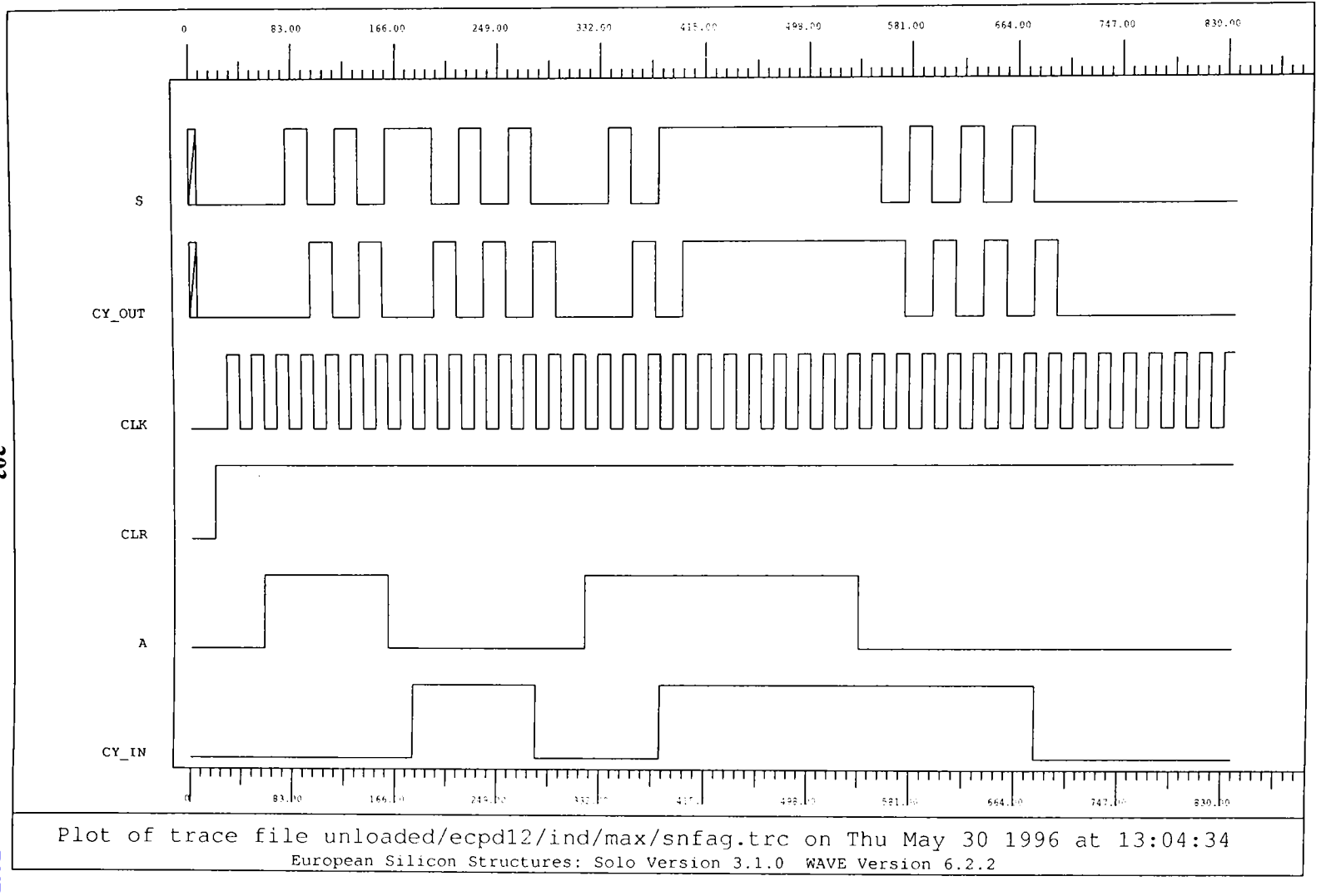


ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part add1s (1 sheet)	Date	Thu May 30 1996

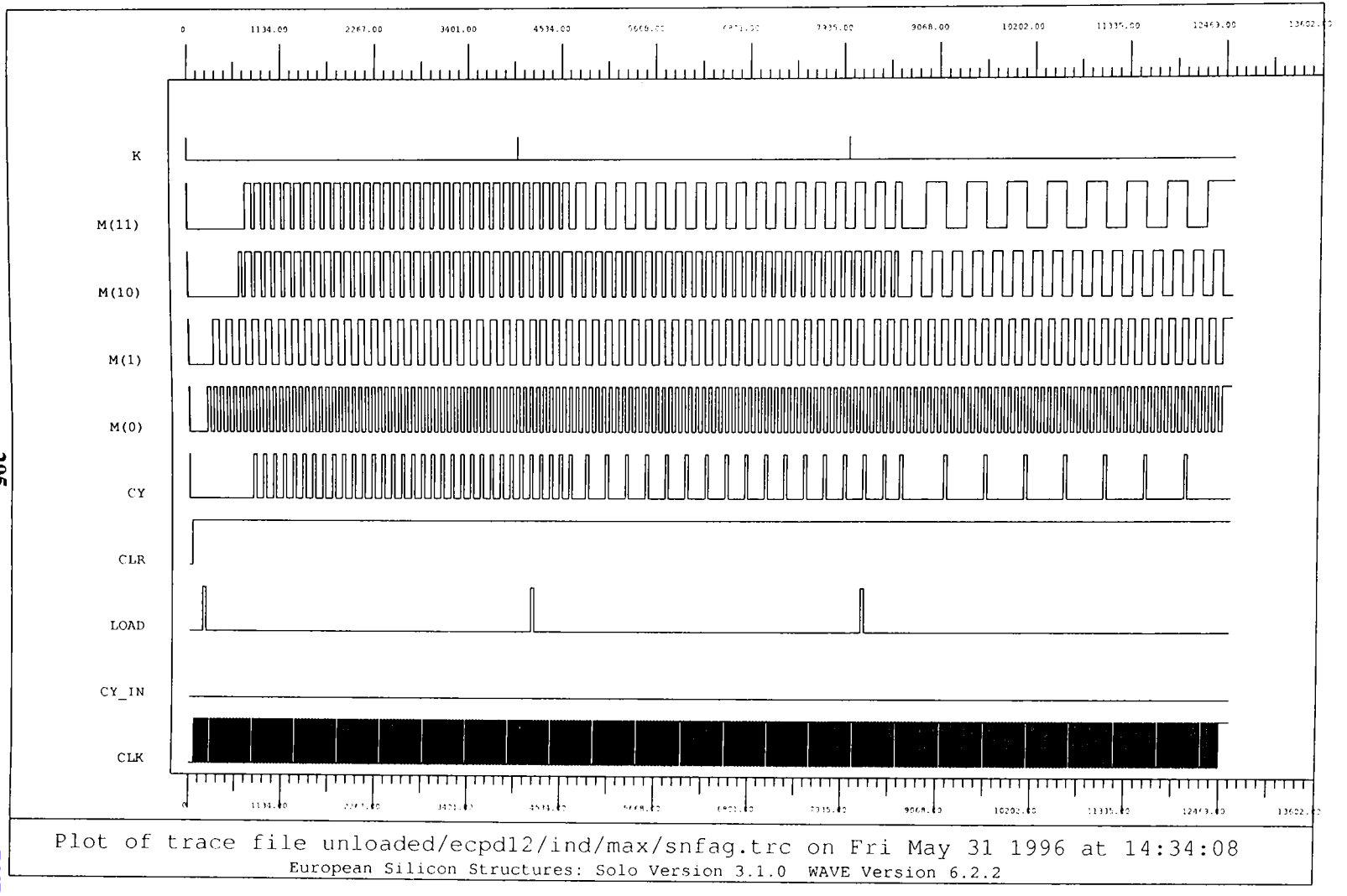


ES2: Solo version 3.1.5 Draft version 6.4.1	Designer	A. Gontean
Sheet s1 of part add1s (1 sheet)	Date	Fri May 31 1996

```
#step = 10
cy_in =L step*18 =H step*10 =L step*10 =H step*30 =L
a     =L step* 6 =H step*10 =L step*16 =H step*22 =L
clr   =L step*2
clk   =L step*3 [=H step =L step]*40
cy_out
s
```


0
83.00
166.00
249.00
332.00
415.00
498.00
581.00
664.00
747.00
830.00
S
CY_OUT
CLK
CLR
A
CY_IN
Plot of trace file unloaded/ecpd12/ind/max/snfag.trc on Thu May 30 1996 at 13:04:34
European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

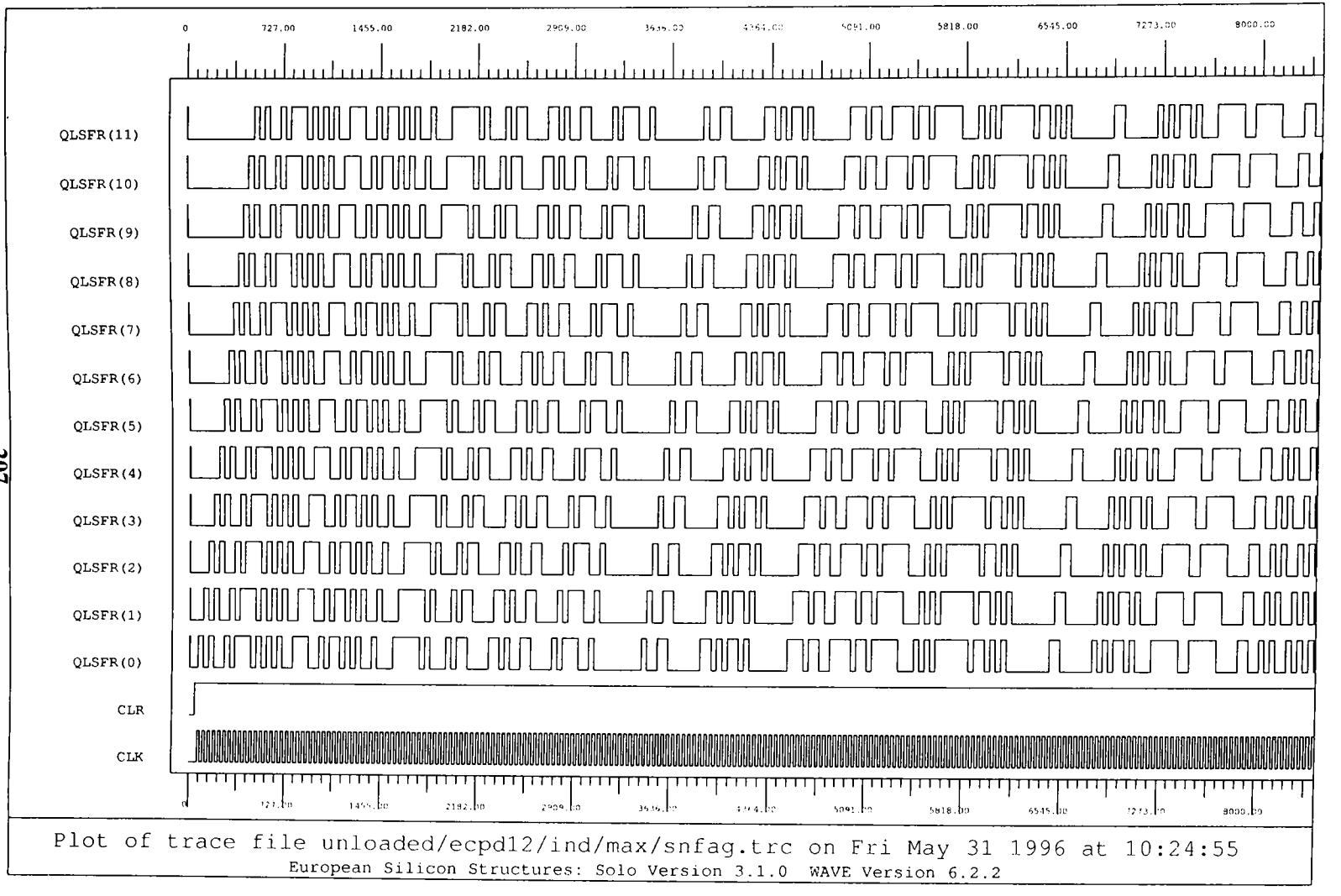
```
#step = 12 {cuanta de timp}
k(0:11) =1365 step*200 =683 step*200 =341
clk   = L step*3    [=H step =L step]*310
cy_in = L
load  = L step*8+5 [=H step*2 =L step*198]*3 =L
clr   = L step*2     =H
cy
m(0)
m(1)
m(10)
m(11)
```



Plot of trace file unloaded/ecpd12/ind/max/snfag.trc on Fri May 31 1996 at 14:34:08

European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

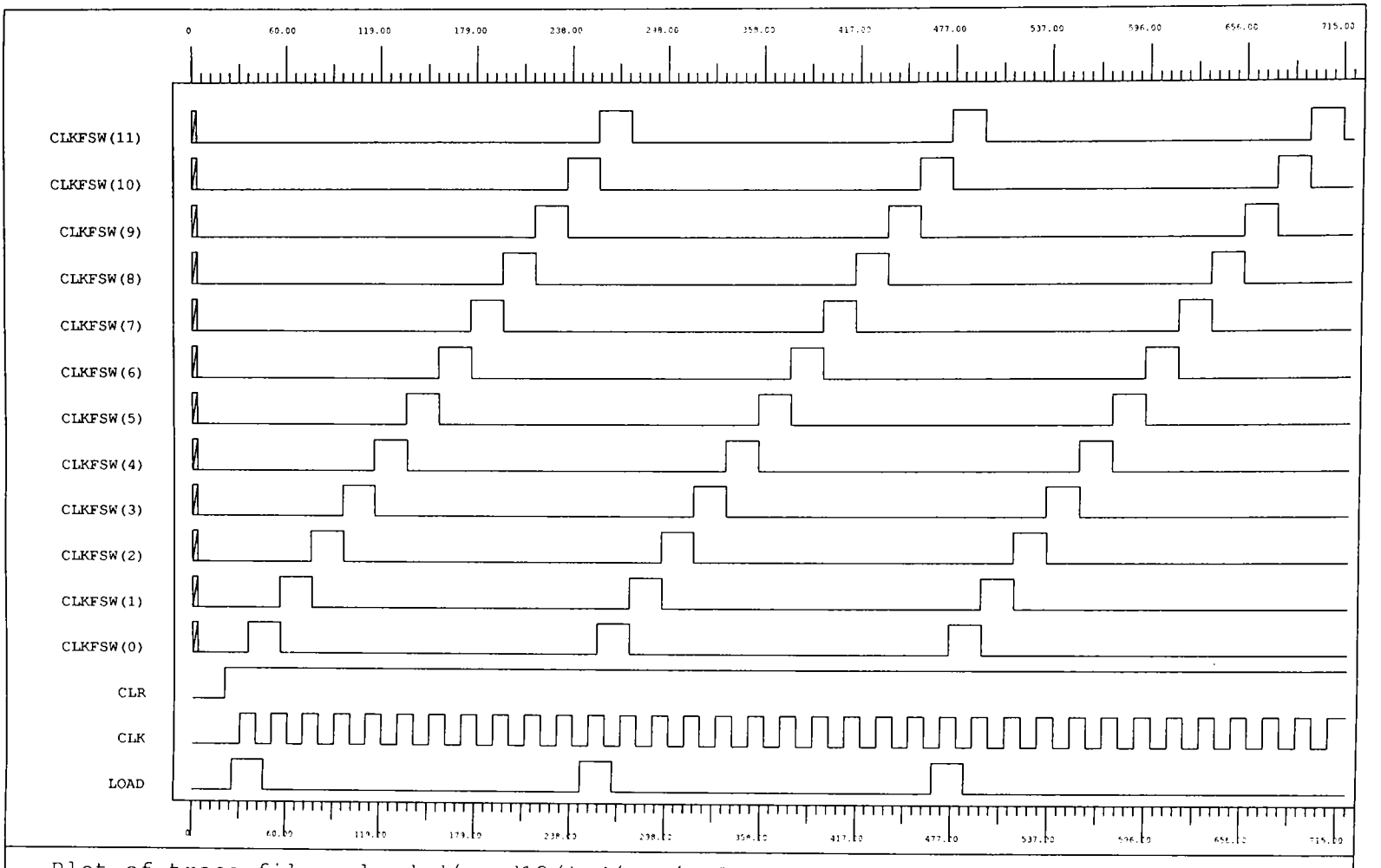
```
#step = 20
clk = L step*3 [=H step =L step]*5000
clr = L step*2 =H
qlsfr(0:11)
```



Plot of trace file unloaded/ecpd12/ind/max/snfag.trc on Fri May 31 1996 at 10:24:55

European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

```
#step = 10
load = L step*2 + 5 [=H step*2 =L step*20]*3 =L
clk = L step*3 [=H step =L step]*34
clr = L step*2 =H
clkfsw(0:11)
```



Plot of trace file unloaded/ecpd12/ind/max/snfag.trc on Fri May 31 1996 at 11:32:39

European Silicon Structures: Solo Version 3.1.0 WAVE Version 6.2.2

BIBLIOGRAFIE

1. [Abouzeid93] Abouzeid, P., ş.a., *Input-Driven Partitioning Methods and Application to Synthesis on Table-Lookup-Based FPGAs*, IEEE Transactions on Computer-Aided Design, Vol. 12, Nr. 7, 1993, pag. 913-925.
2. [Acan92] Acan, A., Ünver, Z., *Switchbox routing by Simulated Annealing: SAR*, Proc. IEEE Int. Symp. Circuits and Systems, vol. 4, 1992, pag. 1985-1988.
3. [Actel92a] Actel, *ACT^{TM} Family Field Programmable Gate Array Data Book*, 1992.
4. [Actel92b] Actel, *FPGA Design Guide II*, 1992.
5. [Ahrens90] Ahrens, M., ş.a, *An FPGA Family Optimized for High Densities and Reduced Routing Delay*, Proceedings of Custom Integrated Circuits Conference, Mai 1990, pag. 31.5.1-31.5.4.
6. [Auer95] Auer, A., Rudolf, D., *FPGA. Feldprogrammierbare Gate Arrays*, Hüthig Buch Verlag Heidelberg, 1995.
7. [AD77] *Analog Digital conversion notes*, Analog Devices, June 1977.
8. [Altera91] *Metastability Characteristics of EPLDs*, Application Note 9, Data Book, Altera, 1991.
9. [Altera93] *Data Book*, Altera, 1993.
10. [AMD90] *Mach Devices High Density EE Programmable Logic Data Book*, AMD, 1990.
11. [Antoniewicz98] Antoniewicz, R., Krasniewski, A., *Testability Measures for Decomposed Structures Intended for FPGA Implementation*, Proceedings International Conference on Programmable Devices and Systems, PDS'98, Gliwice, Polonia, 24-25.02.1998, pag.107-114.
12. [Blank92] Blank, H.J., *Logikbaustaine - Grundlagen, Programmierung und Anwendungen*, Markt&Technik Verlag AG, München, 1992.
13. [Bell71] Bell, C.G., Newell, A., *Computers Structures: Readings and Examples*, Adisson-Wesley, 1971.
14. [Bell78] Bell, C.G., Mudge, J.C., McNamara, J.E., *Computer Engineering: A DEC View of Hardware Systems Design*, Digital Press, 1978.
15. [Bell91] Bell, C.G., McNamara, J.E., *High-Tech Ventures - The Guide for Entrepreneurial Success*, Adisson Wesley, 1991.
16. [Benn89] Benn, H.P., Jones, W.J., *A Fast Hopping Frequency Synthesizer*, Second International Conference on Frequency Control and Synthesis, 10-13 Aprilie 1989, University of Leicester, Anglia, pag. 69-72.
17. [Bobancu74] Bobancu V., ş.a., *Dicţionar de matematici generale*, Ed. Enciclopedică Română, Bucureşti, 1974.
18. [Bodea85] Bodea, M., ş.a., *Circuite integrate liniare. Manual de utilizare. Vol. IV*, Ed. Tehnică, Bucureşti, 1985.
19. [Bolton90] Bolton M., *Digital Systems Design with Programmable Logic*, Addison-Wesley, 1990.

20. [Brown92] Brown, S., Francis, R. a.o., *Field Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.
21. [Bruma93] Bruma C., *Carry skip adders*, Tech. rep., TU Delft, 1993.
22. [Chen95] Chen, C-D., ş.a., *Tracer-fpga: A Router for RAM-Based FPGAs*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol 14, Nr. 3, Martie 1995, pag. 371-374.
23. [Chung92] Chung, K., Rose, J., *TEMPT: Technology Mapping for Exploration of FPGA Architectures with Hard-Wired Connections*, Proceedings of 29th Design Automation Conference, Iunie 1992, Anaheim, CA, pag. 361-367.
24. [Crawford94] Crawford, J. A., *Frequency Synthesiser Handbook*, Artech House, 1994.
25. [Damiani95] Damiani, M., Yang, J.C.-Y., De Micheli,G., "Optimization of Combinational Logic Circuits Based on Compatibles Gates", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 11, November 1995, pag. 1316-1327.
26. [Demian94] Demian P., Gontean, A., *Including PLDs in Logic Analyzer Design Speeds Up Scheme and Cuts Costs*, Proceedings of the Symposium on Electronics and Telecommunications Timişoara, 29-30 sept. 1994, Volume I, pag. 183-186.
27. [DeMicheli94] DeMicheli, G., *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.
28. [Dillinger88] Dillinger T.E., *VLSI Engineering*, Prentice Hall, 1988.
29. [Dresig95] Dresig, F., ş.a., *Der FPGA-Report*, Elektronik, Nr. 9, 1995, pag. 125-136.
30. [Dutt90] Dutt, N.D., Gajski, D.D., *Design Synthesis and Silicon Compilers. Design and Test of Computation*, Design and Test of Computers, Vol. 7, Dec. 1990, pag. 8-23.
31. [Elektor94] * * * ispLSI: *Logik-Schaltkreise Selbstgemacht. Programmieren ohne Programmiergerät*, Elektor, Noiembrie, 1994, pag. 62-65.
32. [Fawcett95] Fawcett, B., Schillinger, P., *Alternative zu Gate Arrays*, Design&Elektronik, Nr. 6, Martie 1995, pag. 39-42.
33. [Gajski88] Gajski, D.D., *Silicon Compilation*, Adisson Wesley, 1988.
34. [Gamal82] Gamal, El A., *Two Dimensional Stochastic Model for Interconnections in Master Slice Integrated Circuits*, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. CAS-28, Nr. 2, February, 1982.
35. [Gamal91] Gamal, El A., ş.a., *Segmented Channel Routing Is Nearly as Efficient as Channel Routing (And Just as Hard)*, Advanced Research in VLSI, Universitatea California, Santa-Cruz, 25-27 Martie 1991.
36. [Gavrilescu97] Gavrilescu, C., *Echipamente pentru prelucrarea numerică a semnalelor*, Referatul Nr. 1 în cadrul pregătirii la doctorat, Timişoara, Decembrie 1997.
37. [Gellert75] Gellert, W., editor, ş.a., *Mică enciclopedie matematică*, Ed. Tehnică, Bucureşti, 1980.
38. [Glas93] Glas, J., *Spread Spectrum Communications*, Tech. Rep., TU Delft, June 1993.
39. [Golomb67] S. W. Golomb, *Shift Register Sequences*, San Francisco, California: Holden-Day Inc., 1967.
40. [Gontean86] Gontean, A., Gavrilescu C., *Bibliotecă matematică rapidă pentru procese în timp real*, A XXVII-a Sesiune de comunicări ştiinţifice studenţeşti, Timişoara, 1986, pag. 68.
41. [Gontean91] Gontean, A., *Tendinţe actuale în structurarea circuitelor logice programabile*, Referat Nr. 1 în cadrul pregătirii la doctorat, Timişoara, 1991.
42. [Gontean92a] Gontean, A., *Metode moderne de analiză şi proiectare a sistemelor numerice folosind structuri logice programabile. Programatoare.* , Referat Nr. 2 în cadrul pregătirii la doctorat, Timişoara, 1992.

43. [Gontean92b] Gontean, A., Râsteiu, M., *New Choices in Digital Design using PLDs*, Lucrările Ştiinţifice ale Universităţii Tehnice Petroşani, Vol. XXIV, Fasc. 1/1992, pag. 47-52.
44. [Gontean92c] Gontean, A., Râsteiu M., *New Trends in Logic Design using PLDs in the 90's*, Lucrările Ştiinţifice ale Universităţii Tehnice Petroşani, Vol. XXIV, fasc. 1-1992, pag. 41-56.
45. [Gontean93a] Gontean, A., Băbăiţă, M., *Metode de testare pentru structurile logice programabile*, Analele Universităţii din Oradea, 1993, pag. 206-212.
46. [Gontean93b] Gontean A., Băbăiţă M., *Metastability Characterstics of TTL Families*, Buletinul Ştiinţific şi Tehnic al Universităţii Tehnice din Timişoara, Tom 38 (52), fasc. 1-2, 1993, pag. 167-170.
47. [Gontean94a] Gontean, A., Băbăiţă M., *Metode de testare pe frontieră. Standardul IEEE 1149.1*, Seminar ETc.
48. [Gontean94b] Gontean, A., Băbăiţă M., *Modern Self-Test Techniques for Digital Circuits*, Sesiunea de comunicări cu participare internaţională OPTIM'94, Universitatea "Transilvania" Braşov, 12-14 mai 1994, pag. 219-222.
49. [Gontean94c] Gontean, A., Băbăiţă M., *Metastability Characteristics of Programmable Logic Devices*, Proceedings of the Symposium on Electronics and Telecommunications Timişoara, 29-30 Sept. 1994, Volume I, pag. 195-198.
50. [Gontean96a] Gontean A., Băbăiţă, M., *Designing High Capacity High Speed RAM based FIFOs*, Proceedings of the International Symposium on Systems Theory. Robotics, Computers&process Informatics, SINTES'8, 8th Edition, Craiova, 6-7 Iunie 1996, pag. 112-116.
51. [Gontean96b] Gontean A., "*Sintetizor numeric de frecvenţă. Varianta ASIC şi FPGA*", Seminarul ştiinţific al Catedrei de Electronică Aplicată, Facultatea de Electronică şi Telecomunicaţii, Universitatea Politehnica, Timişoara, 8 iulie 1996.
52. [Gontean97] Gontean, A. Băbăiţă, M., *Structuri logice programabile. Aplicaţii*, Editura de Vest, Timişoara, 1997.
53. [Gontean98a] Gontean A., *Mathematical Model for Predicting Routing Capabilities in FPGAs*, Proceedings of International Conference Programmable Devices and Systems PDS'98, Gliwice, Polonia, 24-25.02.1998, pag.129-134.
54. [Gontean98b] Gontean A., *Low Cost, High Functionality Programmer*, Proceedings of International Conference Programmable Devices and Systems PDS'98, Gliwice, Polonia, 24-25.02.1998, pag.207-212.
55. [Gorgon98] Gorgon, M., Wiatr, K., *Implementation and Optimization of FPGA Systems Designing in Real-Time Vision Systems*, Proceedings International Conference on Programmable Devices and Systems, PDS'98, Gliwice, Polonia, 24-25.02.1998, pag.135-142.
56. [Greene90] Green, J., ş.a., *Segmented Channel Routing*, Proc. 27th Design Automation Conference, Iunie 1990, pag. 567-572.
57. [Greene93] Green, J., ş.a., *Antifuse Field Programmable Gate Arrays*, Proc. Of The IEEE, Vol. 81, Nr. 7, Iulie, 1993, pag. 1042-1056.
58. [Haskard88] Haskard, M.R., *Analog VLSI Design*, Prentice Hall, 1988.
59. [Hasun88] Hassun, R., Kovalick, A.W., *An Arbitrary Waveform Szntheszizer for DC to 50 MHz*, Hewlett-Packard Journal, Aprilie, 1988, pag. 69-77.
60. [Hatamian87] Hatamian P., Cash, G., "*Parallel bit-level Pipelined VLSI Design for High Speed Signal Processing*", Proceedings of the IEEE, Vol 75, 9 Sept 1987.
61. [Heller84] Heller, W.R., ş.a., "Wirability - Designing Wiring Space for Chips and Chip Packages", *IEEE Design and Test of Computers*, August 1984.

62. [Hesener95], Hesener, A., *Designtips für den FPGA-Entwurf*, Design&Elektronik, Nr. 6, Martie 1995, pag. 46-50.
63. [Hill93] Hill, J.F., Peterson, G.R., *Computer Aided Logical Design with Emphasizes on VLSI*, John Wiley, 1993.
64. [Hoffman92] Hoffman M., Hack U., *Das GAL Buch*, Elektor Verlag GmbH, Aachen, 1992.
65. [Holtkamp94] M. Holtkamp, *Direct digital synthesizers voor fast frequency hopping spread spectrum communicatie*, TU Delft, April 1994.
66. [Hu85] Hu, T.C., Kuh, E.S., *VLSI Circuit Layout: Theory and Design*, IEEE Press, 1985.
67. [Intel93] Intel, *Programmable Logic*, 1993.
68. [JEDEC86] * * * JEDEC Standard no. 3-A: *Standard Data Transfer Format Between Data Preparation System and Programmable Logic Device Programmer*, Washington D.C., Electronic Industries Association, May 1986.
69. [Jone95] Jone, W-B, Papachristou, A., *A Coordinated Circuit Partitioning and Test Generation Method for Pseudo-Exhaustive Testing of VLSI Circuits*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, nr. 3, Martie 1995, pag. 374-379.
70. [Joy92] Joy, D., Ciesielski, M.J., *Layer Assignment foe Printed Circuit Boards and Integrated Circuits*, proc. of The IEEE, Vol. 80, Nr. 2, Februarie 1992, pag. 311-331.
71. [Kleeman87] Kleeman, L., Cantoni, A., Metastable Behaviour in Digital Systems, IEEE Design&Test of Computers, Decembrie 1987.
72. [Kouloheris92] Kouloheris, J., Gamal, El A., *FPGA Performance versus Cell Granularity-PLA Cells*, Custom Integrated Circuits Conference, CICC'92, Mai 1992.
73. [Kuh90] Kuh, E., Ohtsuki, T., *Recent Advances in VLSI Layout*, Proc. of The IEEE, Vol. 78, Nr. 2, Februarie, 1990, pag. 237-263.
74. [Lala90] Lala, P.K., *Digital System Design Using Programmable Logic Devices*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.
75. [Lattice92], Lattice Semiconductor, *pLSI and ispLSI Arhitectural Description*, Data Book, 1992.
76. [Liu95] Liu, L-T., ş.a., *A Replication Cut for Two-Way Partitioning*, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 14, Nr. 5, Mai 1995, pag. 623-648.
77. [O'Leary91] P. O'Leary and F. Maloberti, "A DDS with Improved Spectral Performance", *IEEE Trans. on Communications*, July 1991, Vol. 24, No. 7, pag. 1046-1048.
78. [Onozawa95] Onozawa, A., Chaudhary, K., Kuh, E.S., *Performance Driven Spacing Algorithms using Attractive and Repulsive Constraints for Submicron Lsi's*, IEEE Transactions on Computer Aided Design, vol. 14, Nr. 6, pag. 707-719, 1995.
79. [Lorenzetti89], Lorenzetti, M.J., Baeder, D.S., *Physical Design Automation of VLSI Systems*, Benjamin/Cummings, 1989.
80. [Lienig97] Lienig, J., "A Parallel Genetic Algorithm for Performance-Driven VLSI Routing", IEEE Trans. On Evolutionary Computation, Vol.1, No. 1, pag. 29-39, April 1997.
81. [Lynn92] Lynn, P., *Processors and Noise*, MacMillan, 1992.
82. [Marx85] Marx, E., *EDIF: The Standard for Workstation Intercommunication*, IEEE Micro, October 1985, pag. 68-75.
83. [Mead80] Mead, C.A., Conway, L.M., *Introduction to VLSI Design*, Adisson Wesley, 1980.

84. [Mehrgardt83] S. Mehrgardt, "*Noise spectra of digital sine-generator using the table look-up method*", IEEE Trans. on acoustic, speech and signal processing, Vol. ASSP-31, Nr. 4, August 1983.
85. [Michel92] Michel, P., ş.a., *The Synthesis Approach to the Digital System Design*, Kluwer Academic Publishers, 1992.
86. [Miclăuş94] Miclăuş, M., *Programator de circuite PLD*, Proiect de diplomă, Universitatea Tehnică Timişoara, Facultatea de Electronică şi Telecomunicaţii, 1994.
87. [Mureşan81] Mureşan, T., ş.a., *Microprocesorul 8080 în aplicaţii*, Ed. Facla, Timişoara, 1981.
88. [Mureşan91] Mureşan, T., *Frequency Multiplier*, Electronic Engineering, Nov. 1991, pag. 31.
89. [Mureşan96] Mureşan, T., Gontean A., ş.a., *Circuite integrate numerice. Aplicaţii*, Editura de Vest, Timişoara, 1996.
90. [Mureşan98] Mureşan, T., *Circuite Integrate Numerice. Note de curs*, 1998.
91. [Murgai91] Murgai, R., ş.a., *Improved Logic Synthesis Algorithm for Table Look Up Architectures*, Proceedings IEEE International Conf. On Computer Aided Design, 1991, pag. 564-567.
92. [Naish88] Naish, P., Bishop, P., *Designing ASIC's*, John Willey, 1988.
93. [Nicholas87] H. Nicholas, H. Samueli, "*An analysis of the output spectrum of ddfs in the presence of phase-accumulator truncation*", 41st Annual Frequency Control Symposium, 1987.
94. [Nicholas88] H. Nicholas, H. Samueli, B. Kim, "*The optimisation of ddfs performance in the presence of finite word length effects*", 42nd Annual Frequency Control Symposium, 1988.
95. [Oldfield95] Oldfield, J.V., Dorf, R.C., *Field Programmable Gate Arrays. Reconfigurable Logic for Rapid Prototyping and Implementation of Digital Systems*, John Wiley, 1995.
96. [Philips93] * * * *A Metastability Primer*, Application Note 219, Programmable Logic Data Book, Philips, 1993, pag. 282-285.
97. [Philips97] Philips, *Complex Programmable Logic Devices*, Data Handbook IC27, 1997.
98. [Plessey89] Plessey Semiconductor, *ERA60100 Preliminary Data Sheet*, Swindon, England, 1989.
99. [PrahdalaRao93] Prahdala Rao, B.B., Hansdah, R.C., *Extended Distributed Genetic Algorithm for Channel Routing*, Proc. IEEE Symp. Parallel and Distributed Proc., 1993, pag. 726-733.
100. [PrahdalaRao95] Prahdala Rao, B.B., ş.a., *An Extended Evolutionary Programming Algorithm for VLSI Channel Routing*, Proc. 4th Annu. Conf. Evolutionary Programming, Cambridge MA, MIT Press, 1995.
101. [Puccio94] G. Puccio, *Layout Design of a Direct Digital Frequency Synthesiser as a Frequency Dehopper for a Spread Spectrum Communication System on Sea-of-Gates*", Delft University of Technology, July 1994, http:// olt.et.tudelft.nl/~wissce/reports/giaco/main_html.html.
102. [Pucher98] Pucher, K., *Effective Use of Digital Complementary Line in PLD Circuits*, Proceedings International Conference on Programmable Devices and Systems, PDS'98, Gliwice, Polonia, 24-25.02.1998, pag.149-157.
103. [Ristea83] Ristea, i., Popescu, C.A., *Stabilizatoare de tensiune*, Ed. Tehnică, Bucureşti, 1983.
104. [Roberts78] R.A.Roberts and C. Mullis, *Digital Signal Processing*, Addison-Wesley, 1978.

105. [Rottner95] Rottner, E., *komplexe Bausteinstrukturen beherrschen*, Design&Elektronik, Nr. 6, Martie 1995, pag. 52-54.
106. [Rose93] Rose, J. ş.a., *Architecture of Field-Programmable Gate Arrays*, proceedings of the IEEE, Vol. 81, Nr. 7, Iulie 1993, pag. 1013-1029.
107. [Sait95] Sait, S.M., Youssef, H., *VLSI Physical Design Automation. Theory and Practice*, McGraw-Hill, England, 1995.
108. [Sangiovanni-Vincentelli93] Sangiovanni-Vincentelli, A., ş.a., *Synthesis Methods for Field Programmable Gate Arrays*, Proceedings of the IEEE, Vol. 81, Nr. 7, Iulie 1993, pag. 1057-1083.
109. [Shin95] Shin, H., Kim, C., *Performance-Oriented Technology Mapping for LUT-Based FPGAs* IEEE rans. On VLSI Systems, Vol. 3, Nr. 2, June 1995, pag. 323-327.
110. [Solo91] *Solo 1400 Reference Manuals*, Solo 1400 Release 3.0, European Silicon Structures Limited, 1991.
111. [Sunderland84] D. Sunderland et al., "*Cmos/sos frequency synthesiser LSI circuit for spread spectrum communications*", IEEE Journal on Solid-State circuits, August 1984.
112. [Şabac65] Şabac, Gh., *Matematici speciale*, Vol. II, Ed. Didactică şi Pedagogică, Bucureşti, 1965.
113. [Ştefan83] Ştefan, Gh., Drăghici, I., ş.a., *Circuite integrate digitale*, Ed. Didactică şi Pedagogică Bucureşti, 1983.
114. [TI88] *TTL Logic Data Book*, Texas Instruments, 1988.
115. [TI93a] *FPGA. Data Manual*, Texas Instruments, 1993.
116. [TI93b] *FGGA. Applications Handbook*, Texas Instruments, 1993.
117. [TI94a] *TI-ALS for Windows*, Texas Instruments, 1994.
118. [TI94a] *TI-ALS for Windows. PC Viewlogic Environment*, Texas Instruments, 1994.
119. [Tierney71] J. Tierney, C. Rader, and B. Gold, "*A digital frequency synthesiser*", IEEE Trans. Audio Electroacoustic, March 1971, Vol.AU+19, No. 1, pag. 48-57.
120. [Tierney75] J. Tierney et al., *Frequency Synthesis: Techniques and Application*, IEEE Press, 1975.
121. [Tredennick95] Tredennick, N., Technology and business: Forces Driving Microprocessor Evolution, Proceedings of the IEEE, Vol. 83, No. 12, December 1995, pag.1641-1652.
122. [Trimberger93] Trimberger, S., *A Reprogrammable Gate Array and Applications*, Proc. Of The IEEE, Vol. 81, Nr. 7, Iulie 1993, pag. 1030-1041.
123. [Wakerly90] Wakerly, J., *Digital Design. Principles and Practices*, Prentice Hall. Englewood Cliffs, New Jersey, 1990.
124. [Weste93] Weste, N., Eshragian, K., *Principles of CMOS VLSI Design*, Addison Wesley, 1993.
125. [Xilinx92] *The XACT User Guide*, 1992.
126. [Xilinx94] *The Programmable Logic Data Book*, Xilinx, 1994.
127. [Xilinx96] *The Programmable Logic Data Book*, Xilinx, 1996.
128. [Xu95] Xu, Y., ş.a., Graph-based Output Phase Assignment for PLA Minimization, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, Nr. 5, Mai 1995, pag. 613-622.
129. [Wu96] Wu, Y-L., ş.a.., *Graph Based Analysis of 2-D FPGA Routing*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, Nr. 1, Ianuarie 1996, pag. 33-44.
130. [Yoshimura82] Yoshimura, T., Kuh, E.S., *Efficient Algorithms for Channel Routing*, IEEE Transactions on Computer Aided Design, vol. CAD-1, Nr. 1, pag. 25-35, 1982.

ADRESE DE INTERNET UTILIZATE

131. Actel, http://www.actel.com.
132. Advanced Micro Devices, http://www.amd.com.
133. Aldec Inc., http://www.aldec.com.
134. Altera, http://www.altera.com.
135. BP Microsystems, http://www.bpmicrosystems.com.
136. Cypress, http://www.cypress.com.
137. Data I/O, http://www.dataio.com.
138. Lattice Semiconductor, http://www.lattice.com.
139. Mentat, http://www.cs.virginia.edu/~mentat.
140. SGS-Thomson Microelectronics, http://www.st.com.
141. Texas Instruments, http://www.ti.com.
142. Viewlogic Systems, http://www.viewlogic.com.
143. Xilinx, http://www.xilinx.com.