Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2023, gehouden op donderdag 30 maart 2023 jl. en georganiseerd door NHL Stenden Hogeschool). Bij elkaar bijna 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op donderdag 27 maart 2025 in Zwolle en wordt dan georganiseerd door Hogeschool Windesheim. Kijk op www.nioc2025.nl voor meer informatie.

Wil je op de hoogte blijven van de ontwikkeling rond Stichting NIOC en de NIOC kennisbank, schrijf je dan in op de nieuwsbrief via

www.nioc.nl/nioc-kennisbank/aanmelden nieuwsbrief

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

SIM-PL, auteursomgeving voor digitale componenten

Ben Bruidegom en Wouter Koolen-Wijkstra, AMSTEL-insituut UvA

Samenvatting
SIM-PL is een auteursomgeving om componenten te construeren en te simuleren voor cursussen zoals Digitale techniek, Computerarchitectuur en Computerorganisatie. SIM-PL is te gebruiken in het hele spectrum van abstracties dat bij deze cursussen aan bod komt dus van poortschakelingen tot pipeline processoren. Er wordt een bibliotheek van herbruikbare voorbeeldcomponenten en schakelingen meegeleverd.
Keywords: Digitale techniek, Computerarchitectuur, Computerorganisatie

Inleiding

SIM-PL is een onderwijsgericht modellerings- en simulatiepakket voor digitale componenten. Het wordt met succes toegepast in het informatica en kunstmatige intelligentie curriculum aan de Universiteit van Amsterdam. Dit artikel is als volgt optgebouwd: Eerst leggen we uit wat digitale componenten zijn, en hoe deze componenten in de computer gerepresenteerd worden. Daarna gaan we in op de specifieke eigenschappen van onze simulator. Vervolgens komt onze toekomstvisie m.b.t. SIM-PL aan bod, en als laatste beschouwen we de sterke punten van SIM-PL, en de daar op natuurlijke wijze uit voortvloeiende toepassingen.

Digitale Componenten

Digitale componenten zijn hardwareschakelingen waarvan alle in/uitgangen slechts de waarden 0 of 1 kunnen aannemen. De waarde van de uitgangen is een functie van de waarden van de ingangen in de tijd. Hierbij speelt de *propagation delay* (het tijdsverschil tussen een verandering in de inputwaarde tot de bijbehorende verandering in de outputwaarde) een belangrijke rol. Digitale componenten worden *hiërarchisch* opgebouwd. Bijv. poort → flipflop → register → registerfile van een processor. SIM-PL geeft inzicht in de werking van digitale componenten door het waardeverloop op ingangen, uitgangen en interne verbindingen exact in de tijd te simuleren.

Componenten[i] Construeren

In SIM-PL zijn er, parallel aan de hiërarchische opbouw van fysieke digitale componenten, twee typen componenten:

- Basiscomponenten. Deze atomaire componenten zijn de functionele bouwblokken van de schakeling
- Complexe componenten. Deze zijn opgebouwd uit subcomponenten en verbindingen

Constructie basiscomponent

De constructie van een *basiscomponent* bestaat uit:

- Tekenen basisfiguur
- Vastleggen plaats inputs en outputs
- Programmeren van de relatie tussen inputs en outputs (zie *Events*)
- Programmeren van een eventuele geheugenfunctie
- Instellen van de propagation delay

Events

Componenten worden geactiveerd door de volgende events:

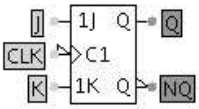
- INIT legt de begintoestand van de component vast
- INPUT-CHANGE reageert op een signaalverandering van één of meer *ingangen*.
- CLOCK-RISING reageert op een positieve klokflank
- CLOCK-FALLING reageert op een negatieve klokflank

Aan elk van deze events kan een programma worden verbonden. (Zie *Interne programmeertaal*) Dit programma wordt uitgevoerd als de bijbehorende event optreedt, en definieert op deze wijze de functionaliteit van de component.

Interne programmeertaal: nBit

Componenten worden in een taal geprogrammeerd met een syntax die sterk op C/C++/Java lijkt. Het basis datatype is een getal van ***n*** bits. Dit wordt gebruikt om een aantal parallelle signaallijnen te representeren.

Als voorbeeld is hieronder een JK–flipflop[ii] weergegeven met een deel van de bijbehorende door de editor gegenereerde xml-code.



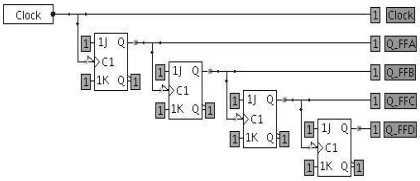

```
<MEMORY>
<STORAGE NAME="m" BITS="1" SIZE="1"/>
<STORAGE NAME="clock" BITS="1" SIZE="1"/>
</MEMORY>
<INTERNALS DELAY="2">
<ACTION EVENT="INPUT_CHANGE">
{
  if( !CLK && clock[0] ) {  // negative edge
  if( J && !K ) m[0]= 1;
  if( !J && K ) m[0]= 0;
  if( J && K ) m[0]= !m[0];
}
 clock[0] = CLK;
 Q= m[0];
 NQ=!m[0];
}
</ACTION>
```

Constructie complexe component

De constructie van een *complexe component* bestaat uit:

- Ophalen en configureren van de subcomponenten van de schakeling.
- Vastleggen plaats inputs en outputs van de gehele schakeling .
- Trekken van de verbindingen tussen de diverse inputs en outputs van de subcomponenten onderling en met de in- en outputs van de schakeling. Op een verbinding kan een 'waypoint/probe' worden geplaatst die de toestand van het signaal op dat punt tijdens de simulatie weergeeft.

Als voorbeeld is hieronder een asynchrone teller samengesteld uit JK-flipflops weergegeven.



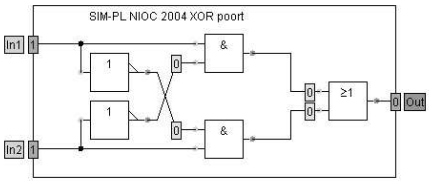
Componenten Simuleren

SIM-PL simuleert het exacte signaalverloop door een schakeling in de tijd. De gebruiker kan de logische waarden op iedere uitgang en verbinding bekijken en vrije ingangen aansturen. Voor het aanleveren van complexe signalen maakt SIM-PL gebruik van twee compilers[iii]:

- De universele compiler
- De generieke assembler compiler

Universele compiler

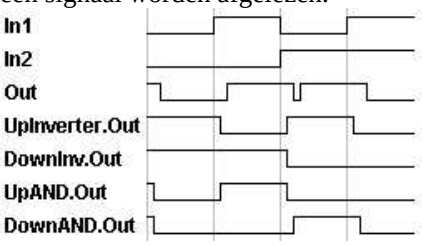
De universele compiler vertaalt laag niveau code in een multikanaal discreet signaal voor willekeurig welke gesimuleerde component. Dit is in het bijzonder handig voor het genereren van waarheidstabellen. Een voorbeeld hiervan is de onderstaande component, een exclusive or-poort.



De voor dit doel ingebouwde optie "generate truth table" produceert onderstaande code. [iv]

```
00: In1= 0;
00: In2= 0;
10: In1= 1;
10: In2= 0;
20: In1= 0;
20: In2= 1;
30: In1= 1;
30: In2= 1;
```

In de volgende figuur is het tijdvolgordediagram weergegeven van dit "programma". De signalen op de in- en uitgangen worden standaard in de tijd weergegeven. Door op een verbinding te klikken tussen twee componenten wordt ook het signaal van deze verbinding weergegeven. De propagatietijd van alle poorten is op 1 gesteld. De initiële waarde van alle ingangen en uitgangen is 1. Met de muis kan de waarde en het tijdstip van een signaal worden afgelezen.



Generieke assembler compiler

De *generieke assembler compiler* vertaalt – gegeven een componentspecifieke instructieset definitie - een assemblerprogramma voor de bijbehorende processorcomponent.

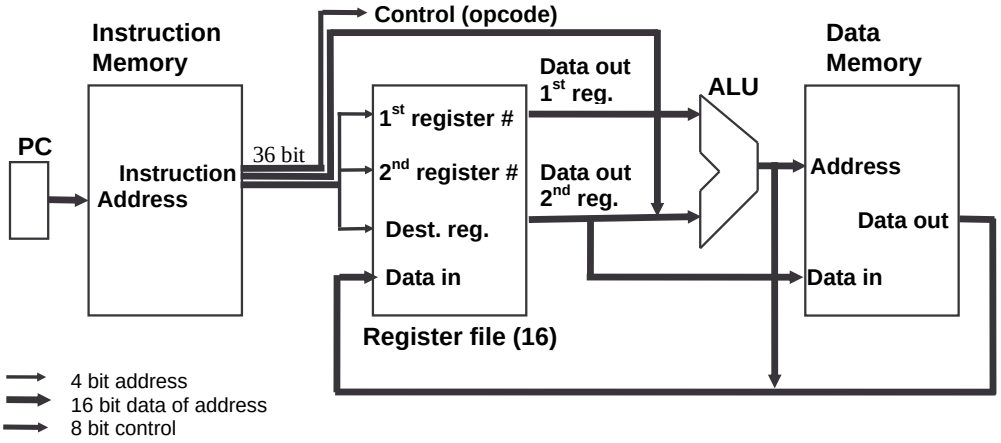
Als voorbeeld is op de volgende bladzijde een sterk vereenvoudigd model van een "Harvard machine[v]" (zie referentie) weergegeven. Deze architectuur bestaat uit vijf hoofdcomponenten: Program Counter (PC), Instruction Memory, Register file, ALU en Data Memory. Iedere instructie wordt in één clockcycle uitgevoerd. Eronder ziet u de realisatie van dit model in SIM-PL. De simulator laat de status zien na het uitvoeren van de instructie LI $1, 0x01FD (Load Immediate register 1 met het getal $01FD_{HEX}$).

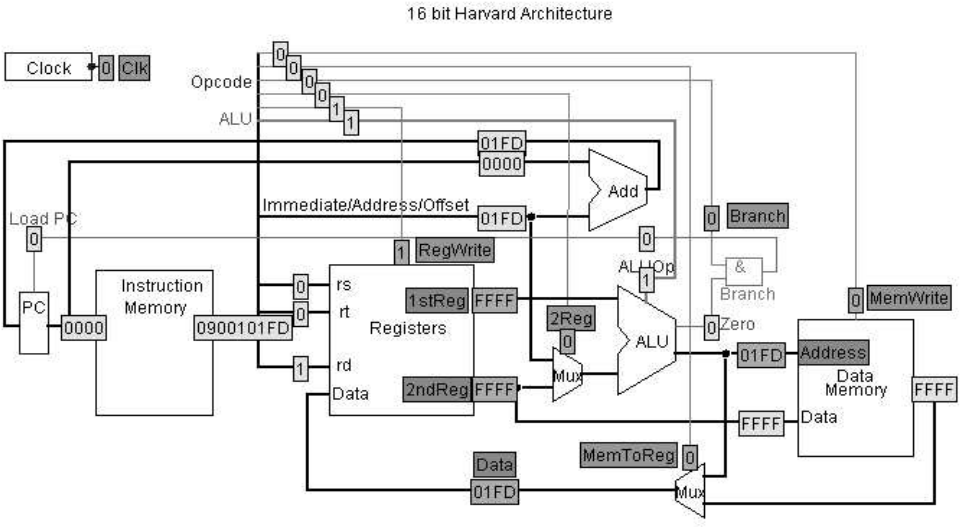
Door op één van de componenten te klikken wordt de status van deze component weergegeven. Zo kan men bijvoorbeeld de status van de Register file of het Data Memory bekijken tijdens het executeren van het programma.

Op de volgende pagina ziet u een voorbeeldsectie uit de instructiesetdefinitie, en daaronder een simpel programma voor deze architectuur. Het programma slaat 5 getallen op in het datageheugen

en telt deze daarna op. De universele compiler wordt aangeroepen op dit programma. De eerste regel vertelt hem waar het instructiesetdefinitiebestand te vinden is. De inhoud van dit bestand bepaalt hoe de rest van het programma wordt geïnterpreteerd. Door het aanpassen van de instructiesetdefinitie kan hetzelfde programma op meerdere processorarchitecturen worden gebruikt.

Er zijn meerdere processormodellen geïmplementeerd, de ingewikkeldste is een MIPS processor met 5 pipeline stages. Deze wordt gebruikt in het lesprogramma voor de informaticastudenten.





```
# bestand: "16bitHarvard.asm.txt"

# unary arithmetical operators
#format   ARITH1  OP:0d5, ALU:0d3, RS:0d4, RT:0d4, RD:0d4, IMM:0d16 | rd:0d4, rt:0d4
#def NOT  ARITH1  " OP = 0b00011; ALU = 0x0; RS = 0; RT = rt; RD = rd; IMM = 0;"
#def MOVE ARITH1  " OP = 0b00011; ALU = 0x1; RS = 0; RT = rt; RD = rd; IMM = 0;"

# binary arithmetical operators
#format   ARITH2  OP:0d5, ALU:0d3, RS:0d4, RT:0d4, RD:0d4, IMM:0d16 | rd:0d4, rs:0d4,
rt:0d4
#def SUB  ARITH2  " OP = 0b00011; ALU = 0x2; RS = rs; RT = rt; RD = rd; IMM = 0;"
#def ADD  ARITH2  " OP = 0b00011; ALU = 0x3; RS = rs; RT = rt; RD = rd; IMM = 0;"
#def XOR  ARITH2  " OP = 0b00011; ALU = 0x4; RS = rs; RT = rt; RD = rd; IMM = 0;"
#def OR   ARITH2  " OP = 0b00011; ALU = 0x5; RS = rs; RT = rt; RD = rd; IMM = 0;"
#def AND  ARITH2  " OP = 0b00011; ALU = 0x6; RS = rs; RT = rt; RD = rd; IMM = 0;"

# transfer data to and from data memory to registers
#format    LOADSTORE  OP:0d5,  ALU:0d3,  RS:0d4,  RT:0d4,  RD:0d4,  OFFS:0d16|  rd:0d4,
offs:0d16, rs:0d4
#def LW  LOADSTORE "OP=0b01001; ALU=0x3; RS = rs; RT = 0; RD = rd; OFFS = offs;"
#def SW  LOADSTORE "OP=0b10000; ALU=0x3; RS = rs; RT = rd; RD = 0; OFFS = offs;"
```

```
#include "16bitHarvard.asm.txt"

# Add 5 values stored in memory locations
5..9 and store the result in
# the first free location 10

        LI $5, 0x5  # 5 values
        LI $6, 0    # Start at element 0
        LI $0, 0    # Clear $0

# Store 0d10 in address 0d5, 0d11 in
address 0d6, etc.
        LI $4, 0d10
        SW $4, 0d5($0)
        LI $4, 0d11
        SW $4, 0d6($0)
        LI $4, 0d12
        SW $4, 0d7($0)
        LI $4, 0d13
        SW $4, 0d8($0)
        LI $4, 0d14
        SW $4, 0d9($0)

loop:   LW $1, 0d5($6)
        ADD $0, $0, $1
        ADDI $6, $6, 1
        BEQ $6, $5, end
        BRA loop

end: SW $0, 0d5($6)
```

Verbeteringen, uitbreidingen

SIM-PL is nog in ontwikkeling. De volgende wensen zijn geuit:

- Gebruiksvriendelijker Editor (bijv. undo-optie).
- Implementatie om Micro-programmeren mogelijk te maken.
- Toevoegen C-compiler om aansluiting te maken bij het vak “Operating Systems”.
- Toevoegen van Componenten en Architecturen uit de meest gebruikte boeken.
- Geschikt maken voor het middelbaar onderwijs.

Visie om dit te realiseren

Vooral vanuit het HBO maar ook uit het VO is interesse getoond om SIM-PL te gaan gebruiken. De auteurs zoeken dan ook partners om bijv. in Digitale Universiteit-verband SIM-PL verder te realiseren.

Onderwijsrelevantie

De krachtige ontwerpmethodiek van SIM-PL sluit naadloos aan bij het conceptueel denken over schakelingen van verschillende abstractieniveaus. SIM-PL maakt het mogelijk exact het gewenste detail te representeren en te simuleren. Het softwarepakket simuleert schakelingen van een paar poorten tot complete processoren met geheugen-hiërarchieën.

Als voorbeeldtoepassing werd een 16 bit Harvard model gepresenteerd. Studenten kunnen hiervoor assembler-code schrijven, de code assembleren en executeren voor dit door de docent geconstrueerde computermodel met instructieset. Dit tot zijn essenties gereduceerde model van een computer is o.a. uitgeprobeerd met VWO-leerlingen. In korte tijd kon een groot deel van deze leerlingen met het model omgaan en werd de werking ervan begrepen.

Andere redenen zijn:

- SIM-PL is een auteursomgeving geschikt voor docenten, studenten en scholieren.
- SIM_PL dicht het “gat” tussen de vakken “Digitale techniek” waarin poorten, flipflops, tellers etc. worden behandeld en “Computerarchitectuur en computerorganisatie” waarin complete (pipeline)processoren aan de orde komen.
- SIM-PL is beschikbaar onder GPL licentie (Free Software) en is te downloaden via www.science.uva.nl/~benb/SIM-PL

Referentie

Boek: David A. Patterson, John L. Hennessy, Computer Organisation And Design Chapter 5.4, Elsevier

[i] Vanaf dit moment wordt met ‘component’ de representatie in de simulatie bedoeld, dit i.t.t. het fysieke circuit.

[ii] Deze component kan worden opgebouwd uit subcomponenten. Om te laten zien dat het insteekniveau van basiscomponenten volledig variabel is, is dat hier niet gedaan.

[iii] Met ‘compiler’ bedoelen we een sofware component die instructies in een specifieke taal omzet in signalen voor de op dat moment gesimuleerde component. Het abstractieniveau van de taal hangt natuurlijk af van de component.

[iv] Links staat het tijdstip waarop de ingangen hun signalen krijgen aangeboden.

[v] Een Harvard machine heeft twee geheugens: één voor instructies en één voor data.