



Academie voor Technology, Innovation &
Society Delft
Academie voor ICT & Media

Digitale System Engineering 2

Week 4 – Datapadsystemen
Jesse op den Brouw
DIGSE2/2016-2017

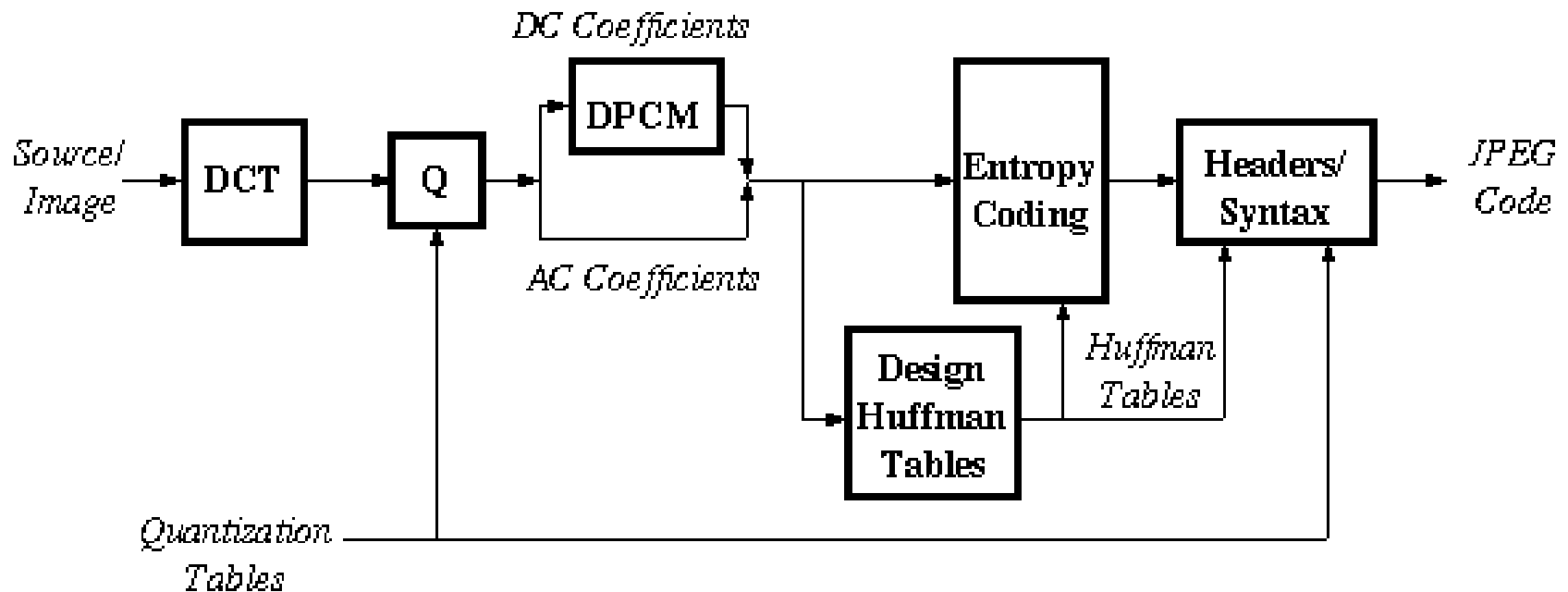
DE HAAGSE
HOGESCHOOL

Complexe systemen

- In principe kan elk sequentiëel systeem beschreven worden met een toestandsdiagram.
- In de praktijk is dat niet handig.
- Het aantal toestanden van zo'n systeem, bijvoorbeeld een microprocessor, kan wel oplopen tot enkele honderden.
- Daarnaast wordt in veel systemen ook *data* verwerkt.
- Daarom is het gebruikelijk om een systeem te *partitioneren* in kleinere deelsystemen.

Complexe systemen

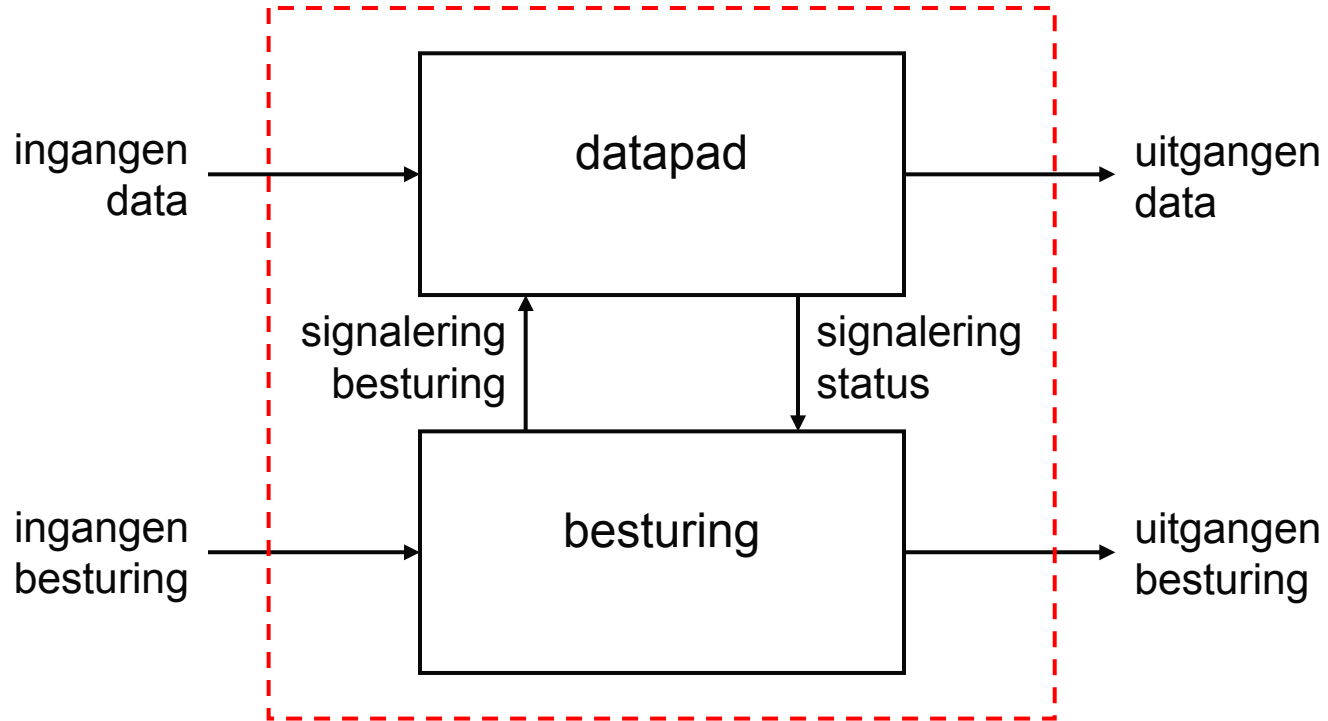
- Vereenvoudigde blokschema JPEG-compressie.



Datapadsystemen

- Elk deelsysteem kan worden opgedeeld in een *datapad* en een *besturing*.
- Het datapad wordt gebruikt voor opslag en bewerken van data binnen een systeem.
- De besturing bestuurt de werking van het datapad.
- We komen aldus tot het volgende model (zie volgende slide).

Datapadsystemen



Bouwstenen datapad

- Registers/geheugens
 - Met enable, load, shift, clear
 - RAM, ROM
- Tellers
 - Als up, down, updown, met enable, load, clear
- Rekenkundig
 - Operaties als add, subtract, multiply, divide
- Logisch
 - Bitwise (and, or, nor, exor), expressie (groter, kleiner, gelijk, ongelijk, ...)
- Multiplexers
 - Routeren van data langs de datapadonderdelen.

Control, RTL

- De control unit is een toestandsmachine die het datapad zodanig aanstuurt dat het geheel de juiste werking krijgt.
- Registers worden gebruikt voor tussentijdse opslag. De data wordt tussen register getransporteerd, al dan niet bewerkt.
- Er wordt dan ook gesproken over *register transfer*.
- Om in grote systemen de operaties éénduidig weer te geven, wordt er een *register transfer language* (RTL) gebruikt. Voorbeeld:

$R2 \leftarrow R2 + R1$

-- *add R1 to R2 and store in R2*

$R2 \leftarrow M(R1)$

-- *Load R2 with RAM pointed by R1*

Vermenigvuldigen

- Eerder is het ontwerp van een vermenigvuldiger besproken.
- Dat kan heel goed met combinatoriek, maar dat levert veel hardware (lees chipoppervlakte, energieverbruik) op.
- Een vermenigvuldiger kan echter ook als sequentiële machine worden ontworpen, waardoor de hoeveelheid hardware beperkt wordt.
- Het nadeel is wel dat het langer duurt voordat het resultaat beschikbaar is.
- NB: de vermenigvuldiger werkt met *unsigned* getallen.

Vermenigvuldigen

- In het binaire systeem werkt vermenigvuldigen zo:

$$\begin{array}{r} 1101 \\ 1011 \\ \hline 1101 \quad \times \\ 11010 \\ 000000 \\ 1101000 \\ \hline 10001111 \quad + \end{array}$$

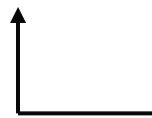
Vermenigvuldigen is eenvoudig:

Vermenigvuldigen met 0 levert 0!

Vermenigvuldigen met 1 levert getal!

0-en schuiven voor tweetal, viertal, ...

Nadeel: multi-input opteller nodig

 Maximaal $4+4 = 8$ cijfers

Vermenigvuldigen

- De multi-input opteller kan vermeden worden door tussentijds op te tellen:

$$\begin{array}{r} A: \quad \quad \quad 1101 \\ B: \quad \quad \quad 1011 \\ \hline PP_1: \quad \quad 1101 \quad \times \\ \\ \quad \quad \quad 11010 \quad + \\ \hline PP_2: \quad \quad 100111 \\ \quad \quad \quad 000000 \quad + \\ \hline PP_3: \quad \quad 100111 \\ \quad \quad \quad 1101000 \quad + \\ \hline P: \quad \quad \quad 10001111 \end{array}$$

standaard
optellers

Vermenigvuldigen

- Nu worden de optelstagen uniform gemaakt en wordt begonnen met een tussenresultaat 0 (PP_0). De rode nullen moeten worden aangevuld om de optelling uniform te maken. De blauwe nullen worden veroorzaakt door het positioneren van A.
- Het is duidelijk te zien dat het bitpatroon van A verschuift.
- Het aantal optelstagen is identiek aan de bitlengte van B.

A:	1101	
B:	1011	x
PP_0 :	00000000	
	00001101	+
PP_1 :	00001101	
	00011010	+
PP_2 :	00100111	
	00000000	+
PP_3 :	00100111	
	01101000	+
P:	10001111	

Vermenigvuldiger

- Voor de sequentiële $n \times n$ -bits vermenigvuldiging is een algoritme op te stellen:

```
P = 0; A = ..; B = ..;  -- Clear intermediate result
for i = 0 to n-1 do    -- Iterate over the n bits of B
  if b(i) = 1 then    -- If i-th bit of B is 1 then ...
    P = P + A;        -- add A to intermediate result
  else                -- But if not ...
    P = P + 0;        -- add zero to intermediate result
  end if;
  left-shift A;      -- Reposition A by shift to left
end for;              -- Ready
```

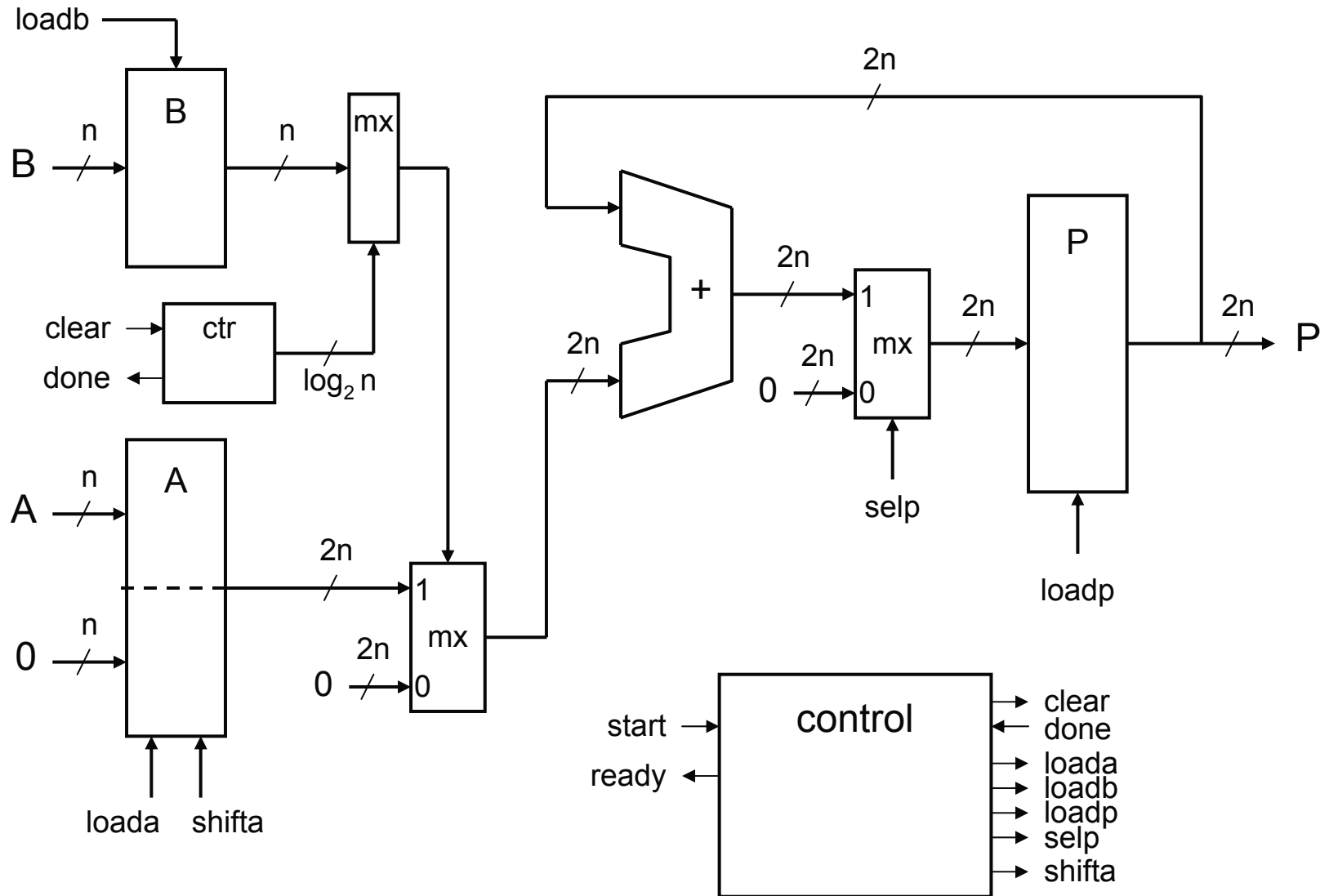
Vermenigvuldiger

- De vermenigvuldiger heeft de volgende digitale bouwstenen nodig:
- Er is een $2n$ -bit schuifregister nodig voor A. De hoogste n bits worden geladen met nullen.
- Er is een n -bit register nodig voor B.
- Er is een $2n$ -bit register nodig voor P.
- Er is een $n \times 1$ multiplexer nodig om de afzonderlijke bits van B te selecteren.
- Er is een teller nodig met $\log_2 n$ telbits om de afzonderlijke bits van B aan te wijzen.
- Alle registers hebben een laad/onthoudmogelijkheid. Register A moet kunnen schuiven.

Vermenigvuldiger

- Vervolg:
- Er is een 2×1 $2n$ -bit multiplexer nodig om A of nullen te selecteren afhankelijk van de waarde van het i^e bit van B.
- Er is een $2n$ -bit opteller nodig om de waarde van A op te tellen bij het tussenresultaat van P. Er is geen c_{in}/c_{out} nodig.
- Er is een 2×1 $2n$ -bit multiplexer nodig om register P te laden met de uitkomst van de opteller of met nullen (wissen).
- Er is een toestandsmachine om het geheel te besturen. Deze heeft een start-ingang om de bewerking te starten en een ready-uitgang om aan te geven dat de bewerking klaar is.

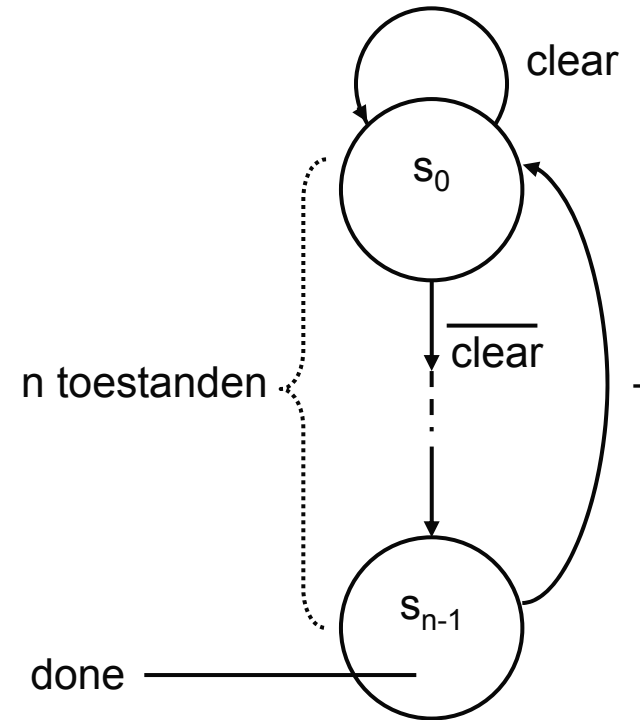
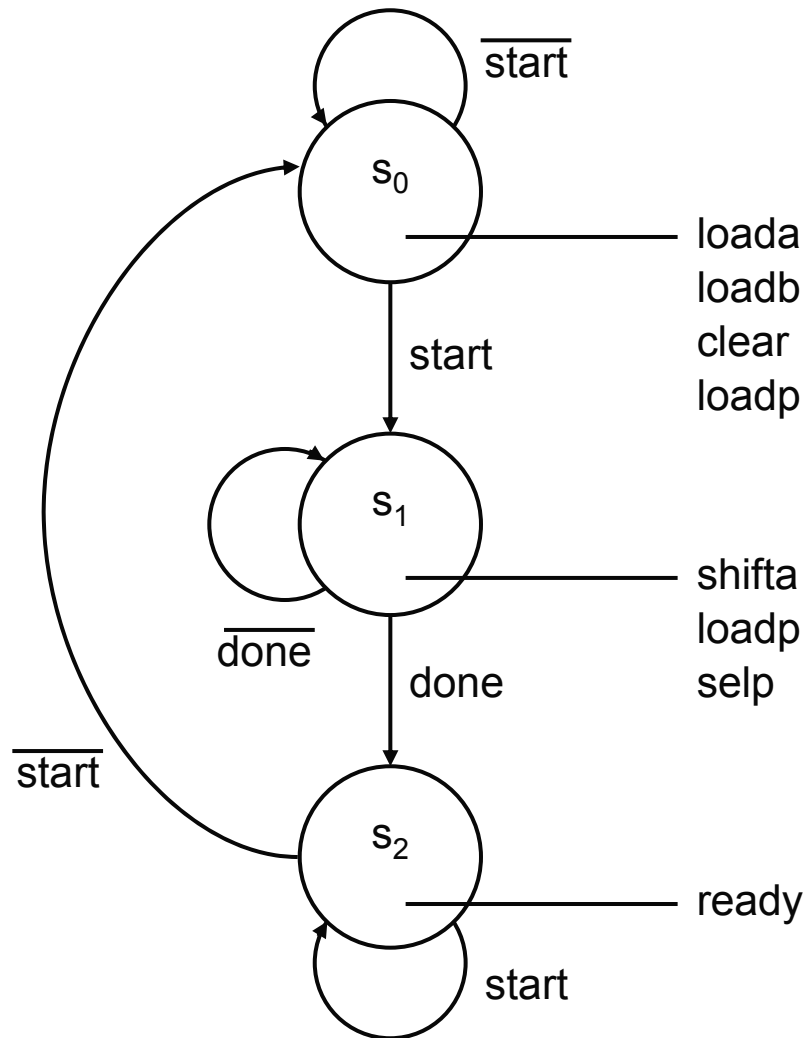
Eerste opzet vermenigvuldiger



Operatievolgorde

- Om het geheel in goede banen te leiden moet het volgende worden gedaan.
- Register A en B moeten worden geladen met de te vermenigvuldigen getallen, de teller moet op 0 gezet worden, register P moet op 0 gezet worden.
- Zolang de teller nog niet klaar is, moet P vermeerderd worden met A als het betreffende bit van B logisch 1 is, anders moet P vermeerderd worden met 0. Register A moet één plaats naar links geschoven worden.
- Als de teller klaar is, signaleer dat de berekening klaar is.

Toestandsdiagrammen control en teller

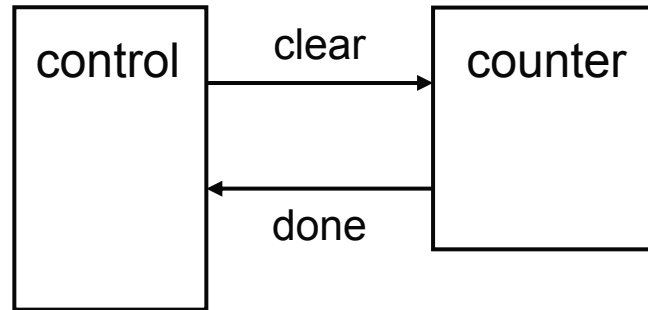


Toestandsdiagrammen control en teller

- Opmerkingen bij de toestandsdiagrammen:
- Bij de overgangen staan nu de signaalsnamen, normaal of geïnverteerd.
- Uitgangen zijn alleen actief indien genoemd bij een toestand. Ontbreekt een naam bij een uitgang dan is deze dus inactief.
- De teller heeft n toestanden. Alleen in toestand s_{n-1} is uitgang done actief.
- De telcyclus wordt doorlopen zolang clear inactief is. Vanuit elke toestand wordt naar s_0 gesprongen als clear actief is.
- De teller geeft de telstand via uitgangen af.

Gekoppelde toestandsmachines

- In het eerste ontwerp zitten twee toestandsmachines, de teller en de besturing, die informatie met elkaar uitwisselen.



- Dit worden gekoppelde toestandsmachines genoemd. Merk op dat beide machines (uiteindelijk) op dezelfde klokflank geklokt worden.

Vermenigvuldiger 2.0

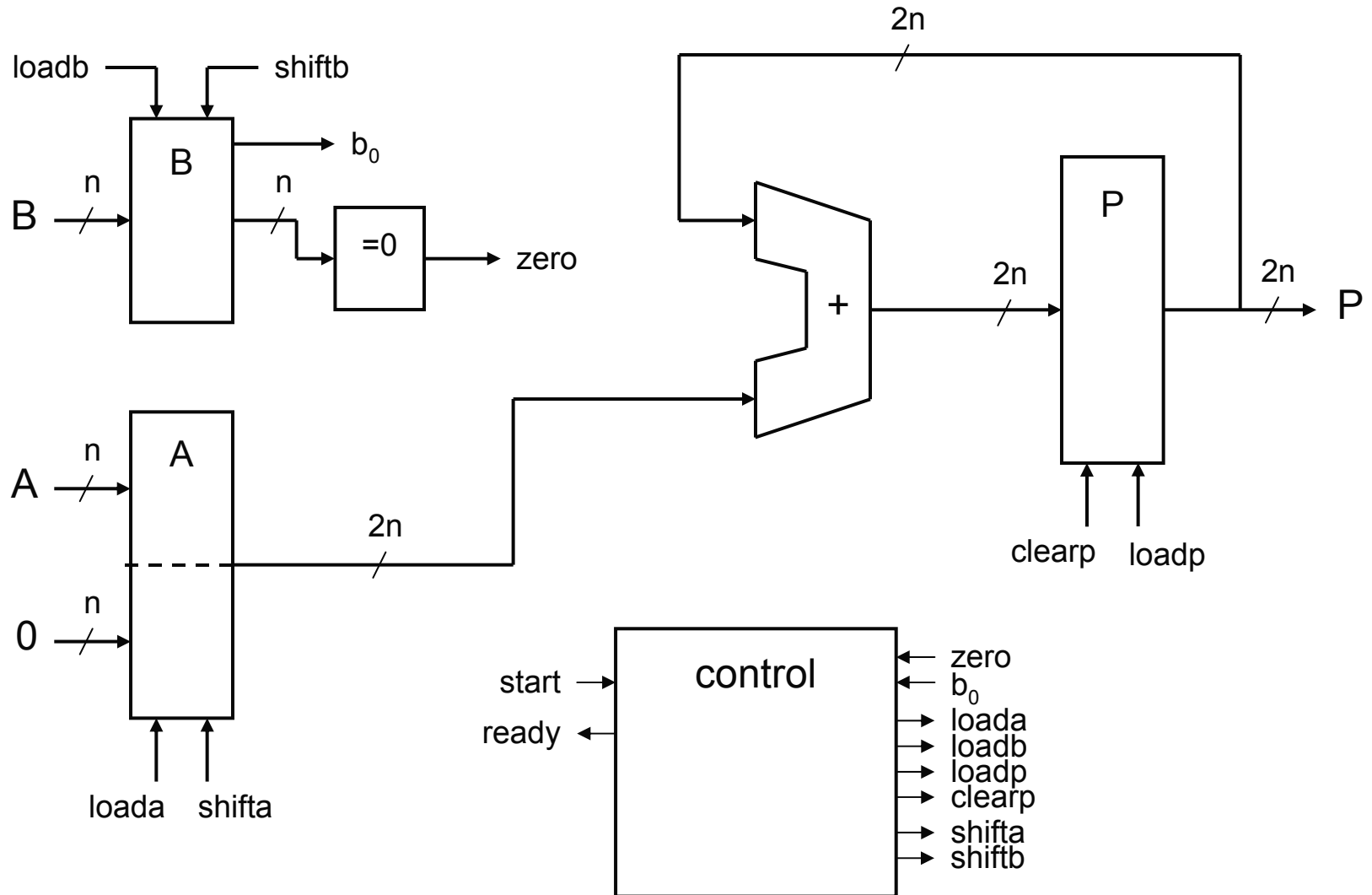
- De eerste vermenigvuldiger kan vereenvoudigd worden. Merk op:
- B wordt een schuifregister zodat de $n \times 1$ multiplexer overbodig is.
- Als B vóór schuiven 0 is, dan hoeft er niet meer opgeteld te worden en is het resultaat bekend. Testen op 0 kan eenvoudig, waardoor de teller komt te vervallen.
- Het optellen van 0 bij P kan vervangen worden door P op dat moment niet te laden. Hierdoor kan een 2×1 $2n$ -mux vervallen.
- De 2×1 $2n$ -mux om P te wissen wordt in P geplaatst. Dat levert iets minder hardware op, en minder tekenwerk.
- De besturing heeft nu meer stuuruitgangen en register B is nu een schuifregister geworden. Dat leidt tot meer logica. Verder zijn register A en de opteller nog steeds $2n$ -bits breed.

Vermenigvuldiger 2.0

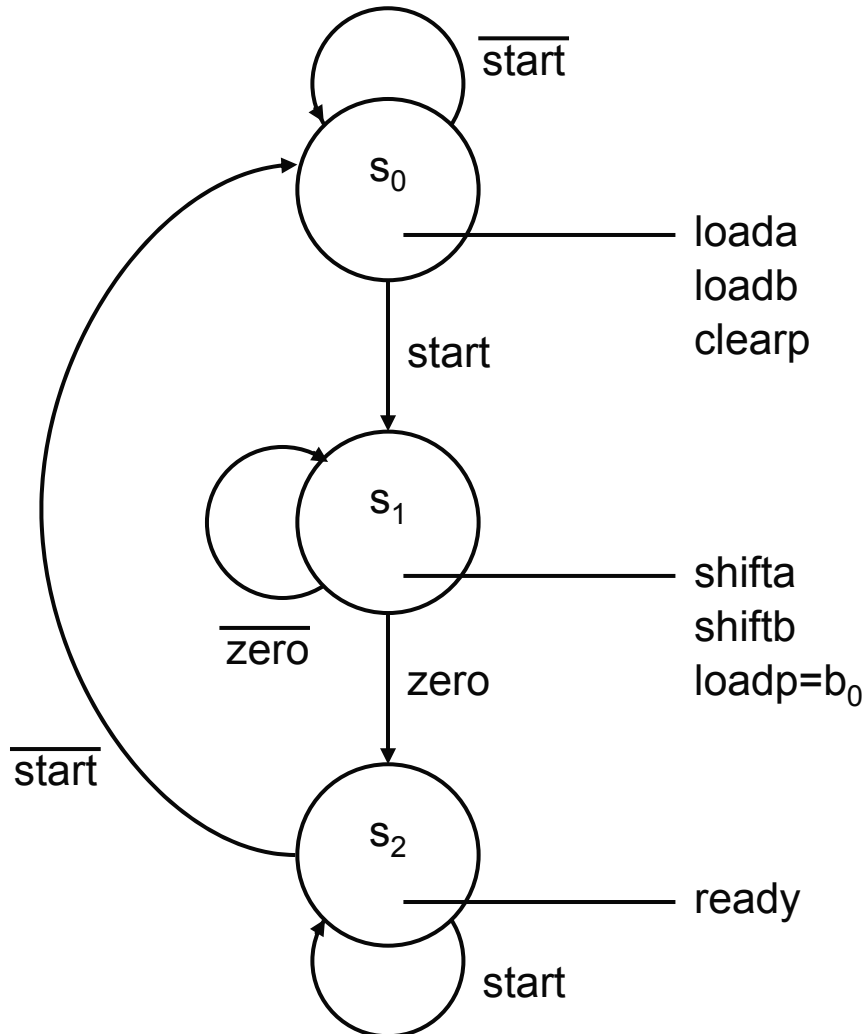
- Voor de sequentiële $n \times n$ -bits vermenigvuldiging is een algoritme op te stellen:

```
P = 0; A = ..; B = ..; -- Clear intermediate result
while B ≠ 0 do          -- Iterate while B not equal to 0
  if b(0) = 1 then     -- If bit 0 of B is 1 then ...
    P = P + A;        -- add A to intermediate result
  end if;
  left-shift A;       -- Reposition A by shift to left
  right-shift B;      -- Set new bit available
end while;            -- Ready
```

Tweede opzet vermenigvuldiger



Toestandsdiagram control



- Merk op:

loada , loadb en clearp zijn tegelijk actief.

shiftd en shiftd zijn tegelijk actief.

loadp is afhankelijk van b_0 (Mealy-uitgang).

VHDL-beschrijving entity en interne signalen

```
entity seq_mult_2 is
    generic (n : integer := 4);
    port (clk : in std_logic;
          areset : in std_logic;
          start : in std_logic;
          dataa : in unsigned(n-1 downto 0);
          datab : in unsigned(n-1 downto 0);
          resultp : out unsigned(2*n-1 downto 0);
          ready : out std_logic);
end entity seq_mult_2;
```

```
architecture rtl of seq_mult_2 is
    constant n_zeros : unsigned(n-1 downto 0) := (others => '0');
    signal rega : unsigned(2*n-1 downto 0);
    signal regb : unsigned(n-1 downto 0);
    signal regp : unsigned(2*n-1 downto 0);
    signal shifta, shiftb, loada, loadb, loadp, clearp : std_logic;
    signal b0, zero : std_logic;
    type state_type is (s0, s1, s2);
    signal state : state_type;
```


VHDL-beschrijving control

- Hieronder de toestandsdefiniëring en de NSL mét register ineen.

```
architecture fsm of control is
type state_type is (s0, s1, s2);
signal state : state_type;
begin
  nsl_reg: process (clk, areset) is
  begin
    if areset = '1' then
      state <= rest;
    elsif rising_edge(clk) then
      case state is
        when s0 => if start = '1' then state <= s1; end if;
        when s1 => if zero = '1' then state <= s2; end if;
        when s2 => if start = '0' then state <= s0; end if;
        when others => null;
      end case;
    end if;
  end process;
```

VHDL-beschrijving control

- De uitgangslógica

```
o1: process (state, b0) is
begin
  loada <= '0'; loadb <= '0'; loadp <= '0'; shifta <= '0';
  shiftb <= '0'; clearp <= '0'; ready <= '0';
  case state is
    when s0 => loada <= '1'; loadb <= '1'; clearp <= '1';
    when s1 => shifta <= '1'; shiftb <= '1';
                if b0 = '1' then loadp <= '1'; end if;
    when s2 => ready <= '1';
    when others => null;
  end case;
end process;

end fsm;
```

VHDL-beschrijving datapad

```
datapad: process (clk, areset, regb) is
  begin
    if areset = '1' then
      rega <= (others => '0'); regb <= (others => '0');
      regp <= (others => '0');
    elsif rising_edge(clk) then
      -- Register A.
      if loada = '1' then rega <= n_zeros & dataa;
      elsif shifta = '1' then rega <= rega(2*n-2 downto 0) & '0';
      end if;
      -- Register B.
      if loadb = '1' then regb <= datab;
      elsif shiftb = '1' then regb <= '0' & regb(n-1 downto 1);
      end if;
      -- Register P.
      if clearp = '1' then regp <= (others => '0');
      elsif loadp = '1' then regp <= regp + rega; end if;
    end if;
  end if;
```

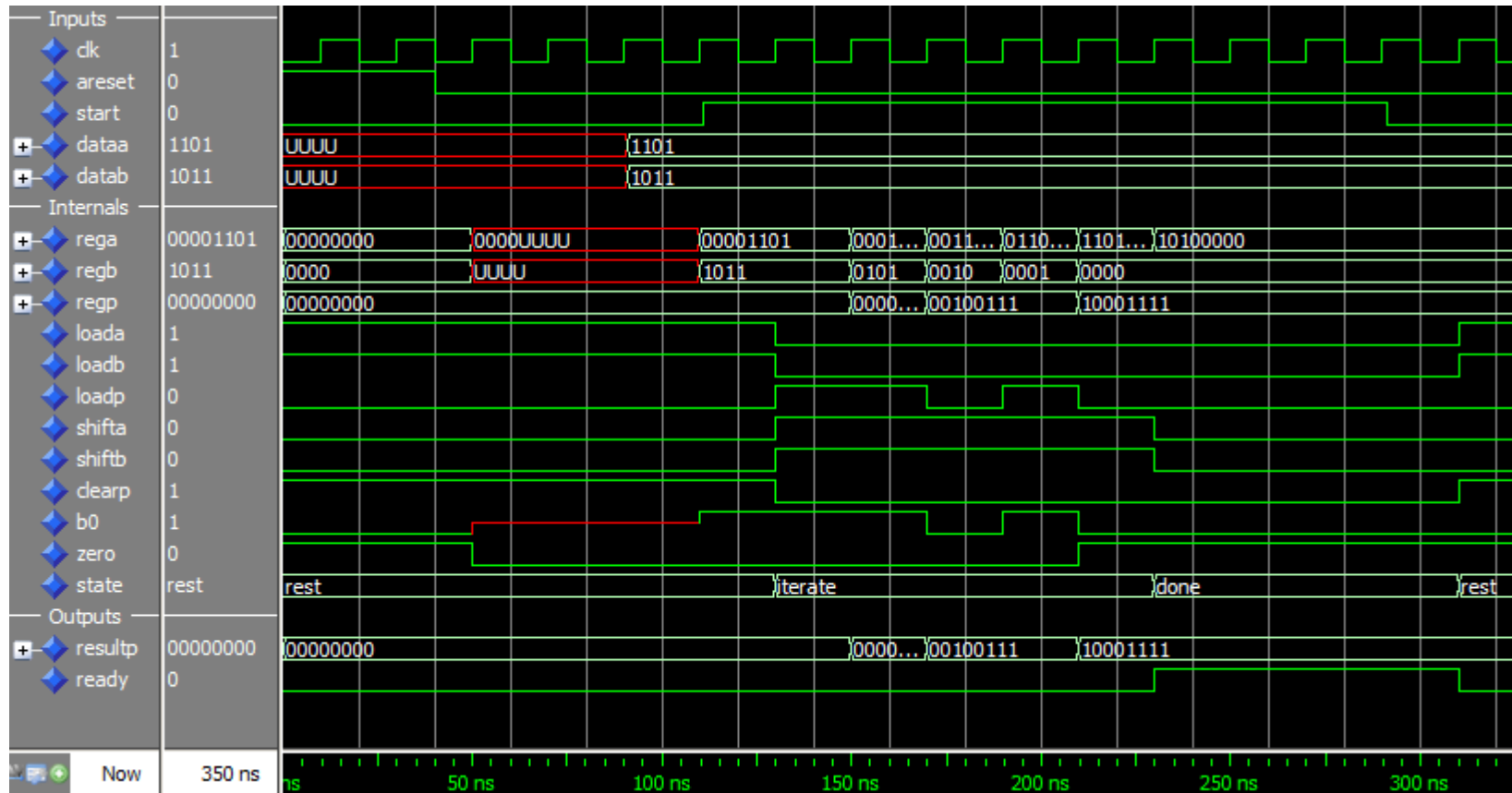
VHDL-beschrijving datapad

```
-- The combinatorial outputs zero and b0
  if regb = n_zeros then zero <= '1'; else zero <= '0'; end if;
  b0 <= regb(0);
end process;

resultp <= regp;
end architecture rtl;
```

Simulatie

- Hieronder de simulatie van de vermenigvuldiging van 13x11.



Opgaven

- Register A is $2n$ bits breed. Ontwerp een derde versie van de vermenigvuldiger waarbij voor A slechts n bits nodig zijn. Hint: P schuiven in plaats van A .
- De opteller is $2n$ bits breed. Ontwerp een vierde versie van de vermenigvuldiger waarbij voor de opteller minder bits nodig zijn.
- Een nadeel van de ontworpen control is dat P wordt gewist in de rusttoestand. Pas het toestandsdiagram aan zodat P behouden blijft.
- Herontwerp de toestandsmachine voor de twee versie van de vermenigvuldiger als een *Mealy*-machine.

Opgaven

- Toon aan dat een vermenigvuldiging van twee n -bits getallen altijd past in $2n$ bits.
- Een slimme student opperde dat signaal b_0 direct verbonden kan worden aan signaal $loadp$. Werkt de vermenigvuldiger dan nog correct? Motiveer het antwoord.
- Ontwerp een systeem dat van de inhoud van een 8-bits register het aantal enen bepaalt dat in het register is opgeslagen. Ontwerp zowel het datapad als de besturing.

FSM with datapad

- In de vorige slides is duidelijk een scheiding te zien tussen datapad en control.
- Control genereert besturingssignalen, datapad genereert statussignalen.
- In VHDL is het mogelijk om control en datapad te combineren in één beschrijving (bv. een proces).
- Dit wordt *FSM with datapad* genoemd, FSMd.

Vermenigvuldiger 2.0 FSMd

```
-- This architecture uses signals for internal storage
architecture rtl_sig of seq_mult_2_fsmd is
type state_type is (rest, iterate, done);
signal state : state_type;
signal rega : unsigned (2*n-1 downto 0);
signal regb : unsigned (n-1 downto 0);
signal regp : unsigned (2*n-1 downto 0);
constant n_zeros : unsigned (n-1 downto 0) := (others => '0');
begin

    -- Here comes the process, see next slide
    ...

    -- Signal ready (combinatorial process)
    ready <= '1' when state = done else '0';
    -- Output the result
    resultp <= regp;

end architecture rtl_sig;
```

Vermenigvuldiger 2.0 FSMd

```
-- The next state logic and register in one process
-- together with the data processing
process (clk, areset) is
begin
  -- Asynchronous reset, set reset state, clear all registers
  if areset = '1' then
    state <= rest;
    rega <= (others => '0');
    regb <= (others => '0');
    regp <= (others => '0');
    -- On positive edge, hence flipflops
  elsif rising_edge(clk) then

    -- Here comes the processing of the states, see next slide
    ...

  end if;
end process;
```

Vermenigvuldiger 2.0 FSMd

```
case state is
  -- Wait for start to be asserted, then goto iterate
  -- Meanwhile, clear all registers
  when rest => if start = '1' then state <= iterate; end if;
                rega <= n_zeros & dataa;
                regb <= datab;
                regp <= (others => '0');

  -- As long as B is not 0, stay iterating, else we're done
  -- Increment P if lsb of B is 1, left shift A, right shift B
  when iterate => if regb = n_zeros then state <= done; end if;
                  if regb(0) = '1' then regp <= regp + rega; end if;
                  rega <= rega(2*n-2 downto 0) & '0';
                  regb <= '0' & regb(n-1 downto 1);

  -- Wait for start to be de-asserted, then goto rest
  when done => if start = '0' then state <= rest; end if;

  -- If something goes horrible wrong, do nothing...
  when others => null;
end case;
```

FSM with datapad

- Het is heel goed mogelijk om tellers en schuifregisters te integreren in een FSMd.
- Door gebruik te maken *generics* en constanten kan zo een herbruikbaar ontwerp worden beschreven.
- Voorwaarde is wel dat alle toekenningen onder klokflanksturing plaatsvinden.

```
entity rs232_module is
    generic (sys_freq : integer := 50000000;
            baudrate : integer := 9600);
    ...
end entity;
```

FSM with datapad

- Door het gebruik van de generics kunnen diverse constanten worden uitgerekend.

```
-- number of system clock pulses per bit time
constant bittime : integer := sys_freq / baudrate;
-- counter to count the bit time
signal bittime_counter : integer range 0 to bittime-1;
...
signal receive_shifter : std_logic_vector(9 downto 0);
...
-- The state of the FSMd
type state_type is (load_timer, read_data_bit, next_bit, ...);
signal state : state_type;
```

FSM with datapad

- Het is heel goed mogelijk om tellers te integreren in een FSMd.

```
when load timer :
    bittime_counter <= bittime - 1;
    state <= read data bit;
when read data bit :
    if bittime_counter > 0 then
        -- bit time not expired, so update ...
        bittime_counter <= bittime_counter - 1;
    else
        -- bit timer expired, so restore bit time and shift in
        bittime_counter <= bittime - 1;
        receive_shifter <= input & receive_shifter(9 downto 1);
        state <= next bit;
    end if;
when next bit :
    ...
```

Algoritme voor vermenigvuldiging

- Eerder is een algoritme gegeven voor de vermenigvuldiging.
- Dit algoritme is niet zonder meer te vertalen naar hardware.
- Zo heeft de machine interactie met de omgeving; er is een start-ingang en een ready-uitgang.
- Daarnaast 'stopt' hardware niet als het algoritme klaar is.
- De pseudo-code moet vertaald worden naar een algoritme dat direct kan worden omgezet in hardware.

Algoritme voor vermenigvuldiging

- Het algoritme is als volgt:

```
ready := 0;                -- (reset) beginnen met niet-klaar
forever do                -- hardware werkt continu
  if start = '1' then    -- start! dus initialiseer alles
    A := input-A; B := input-B; P:= 0; ready := 0;
  elsif B ≠ 0 then      -- B heeft nog 1-en ...
    if b(0) = '1' then  -- een 1 gevonden dus ...
      P := P + A;      -- A erbij optellen
    end if;
    left-shift A; right-shift B; -- A en B positioneren
  else                  -- B heeft geen 1-en meer ...
    ready := 1;        -- dus klaar
  end if;
end for;
```


Algoritme voor vermenigvuldiging

- Helaas is dit algoritme nog niet correct.
- Als start logisch 0 is, wordt uitgang ready altijd 1, ook als er nog geen uitvoering is geweest of als de start eerder weer logisch 0 wordt.
- Er wordt een signaal running geïntroduceerd dat aangeeft of de machine bezig is met de vermenigvuldiging.

Algoritme voor vermenigvuldiging

- Het algoritme is als volgt:

```
ready := 0; running := 0;           -- beginnen met niet-klaar
A := 0; B := 0; P := 0;           -- (reset)
forever do                          -- hardware werkt continu
  if start = 1 and running = 0 then -- start! dus initialiseer
    A := input-A; B := input-B; P:= 0; ready := 0; running := 1;
  elsif B ≠ 0 and running = 1 then -- B heeft nog 1-en ...
    if b(0) = '1' then
      P := P + A;
    end if;
    left-shift A; right-shift B;
  else                                -- B heeft geen 1-en meer
    if start = 0 then running := 0; end if;
    if running = 1 and B = 0 then ready := 1;
                                else ready := 0; end if;
  end if;
end for;
```

Algoritme voor vermenigvuldiging

- Merk op dat alle registers flankgevoelig zijn.
- Alle toekenningen gebeuren dus op de klokflank, ook voor signaal ready.
- Om er voor te zorgen dat ready een combinatorische uitgang wordt en geen synchrone uitgang, moet ready buiten de klokflanksturing worden beschreven*.
- Op de volgende slides staan de uiteindelijke VHDL-beschrijving.

* ready mag wel een synchrone uitgang zijn, maar waarom dan toch combinatorisch?

Algoritme voor vermenigvuldiging

- De entity is hetzelfde als bij voorgaande machines

```
architecture algorithmic of seq_mult_2_algo is
begin
```

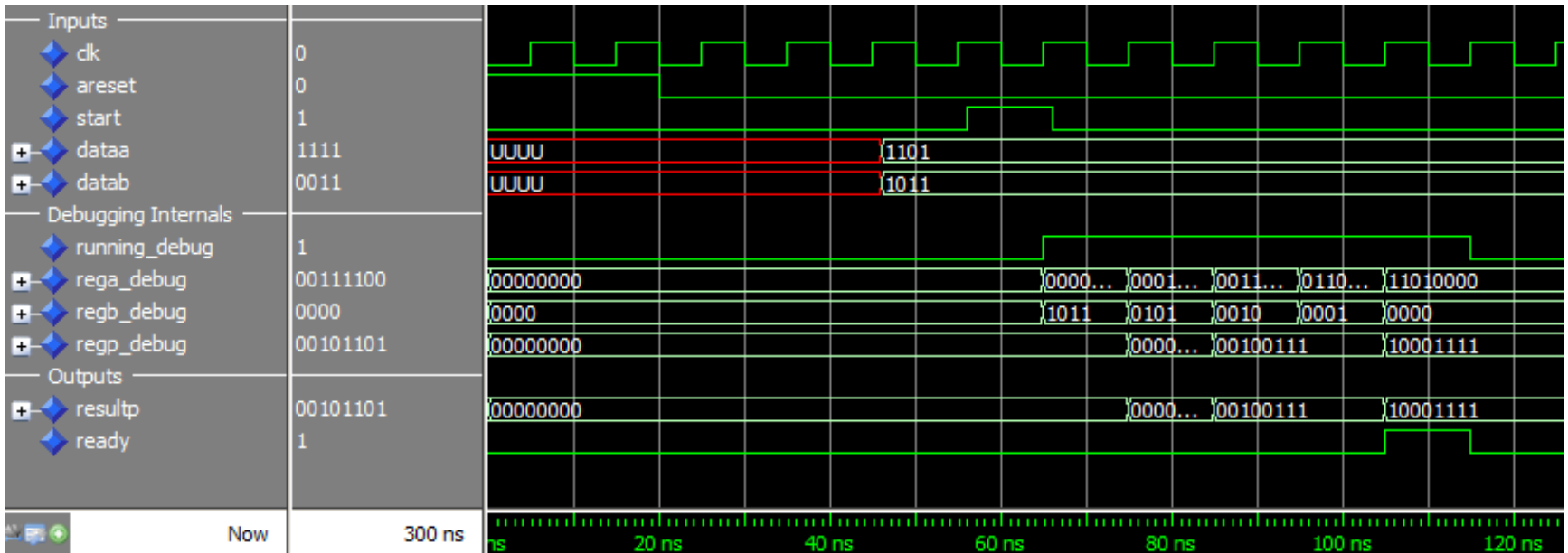
```
    process (clk, areset) is
        -- Register A, B and P
        variable rega : std_logic_vector (2*n-1 downto 0);
        variable regb : std_logic_vector (n-1 downto 0);
        variable regp : std_logic_vector (2*n-1 downto 0);
        -- Keep track of running or not
        variable running : std_logic;
        -- Create a vector with n zero's
        constant n_zeros : std_logic_vector (n-1 downto 0)
            := (others => '0');
```

Algoritme voor vermenigvuldiging

```
begin
  if areset = '1' then                                     -- asynchronous reset
    running := '0'; rega := (others => '0');
    regb := (others => '0'); regp := (others => '0');
  elsif rising_edge(clk) then
    if start = '1' and running = '0' then               -- start pressed
      rega := n_zeros & dataa;                          -- so load and clear
      regb := datab;
      regp := (others => '0');
      running := '1';                                   -- flag we're running
    elsif regb /= n_zeros and running = '1' then       -- B has one's
      if regb(0) = '1' then regp := regp + rega; end if; -- add
      rega := rega(2*n-2 downto 0) & '0';              -- shift left A
      regb := '0' & regb(n-1 downto 1);                -- shift right B
    else                                                -- B has no 1's
      if start = '0' then running := '0'; end if;     -- stop if start released
    end if; -- clock edge
  end if; -- async reset
  if running = '1' and regb = n_zeros then             -- flag we're ready
    ready <= '1'; else ready <= '0'; end if;
  resultp <= regp;                                     -- output result
end process;
end algorithmic;
```

Algoritme voor vermenigvuldiging

- Hieronder het simulatieresultaat. Er zijn extra signals toegevoegd voor het volgen van enkele variabelen. Merk op dat deze signalen alleen aan het eind van een simulatiecyclus wordt bijgewerkt.



Literatuur

Boeken op het gebied van datapad, controllers en multipliers:

- Fundamentals of Digital Logic with VHDL Design – Stephen Brown, 3rd Ed, 2008
- The Designer's Guide To VHDL – Peter J. Ashenden, 2nd Ed, 2002
- Digitale Techniek deel 2 – A.P. Thijssen, 5^e druk, 2000
- Contemporary Logic Design – Randy H. Katz, 2nd Ed, 2005
- Digital System Design With VHDL – Mark Zwoliński, 2nd Ed, 2004
- Logic and Computer Fundamentals – M. Morris Mano, 3rd Ed, 2004
- http://www.ddpp.com/DDPP3_mkt/c05samp3.pdf

Er zijn zeker nog meer mogelijkheden om sequentiële vermenigvuldigers te maken:

- <http://faculty.kfupm.edu.sa/COE/mimam/files/COE200experiment13.pdf>
- <http://www.uoguelph.ca/~ggrewal/3120/Lab11.pdf>
- http://cseweb.ucsd.edu/classes/wi10/cse140L/labs/lab4/lab4_sol.pdf



Academie voor Technology, Innovation &
Society Delft
Academie voor ICT & Media

De Haagse Hogeschool, Delft
+31-15-2606311
J.E.J.opdenBrouw@hhs.nl
www.dehaagsehogeschool.nl

DE HAAGSE
HOGESCHOOL