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

Digitale Systeem Engineering 2

Week 4 – Datapadsystemen
Jesse op den Brouw
DIGSE2/2020-2021

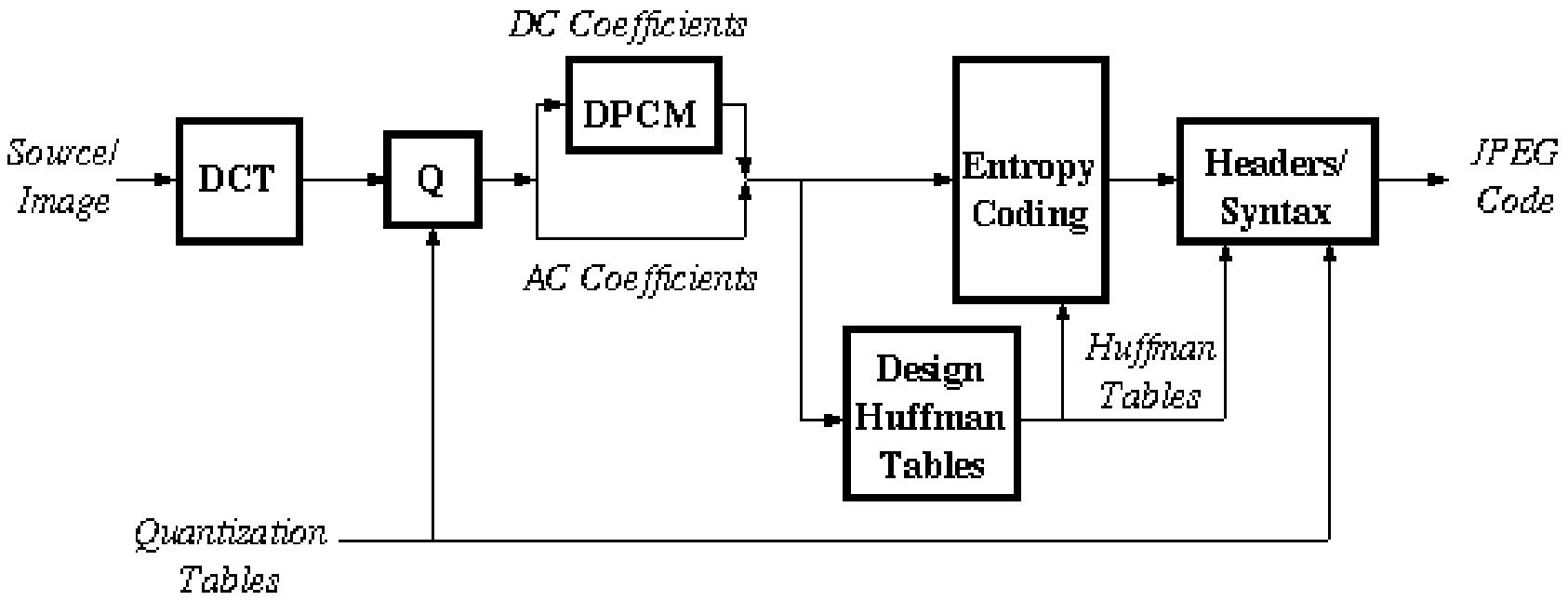
DE HAAGSE HOGESCHOOL

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

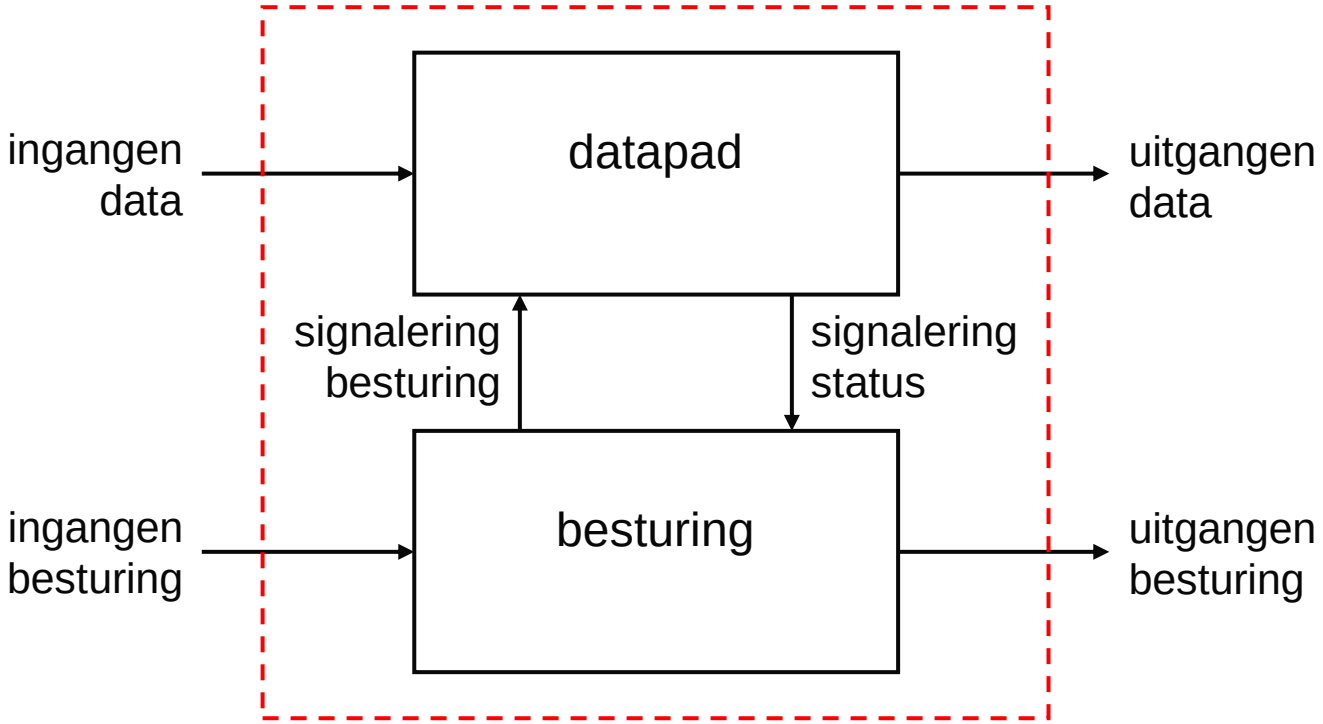
- Vereenvoudigde blokschema JPEG-compressie.



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, not, 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 ← R2 + R1          -- add R1 to R2 and store in R2
R2 ← M(R1)            -- load R2 with RAM pointed by R1
```

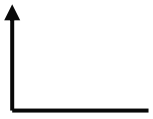
Vermenigvuldigen

- 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:

```
    1101
    1011   x
  ------
    1101
   11010
  000000
 1101000   +
 --------
 10001111
```



Maximaal 4+4 = 8 cijfers

Vermenigvuldigen is eenvoudig:

Vermenigvuldigen met 0 levert 0!

Vermenigvuldigen met 1 levert getal!

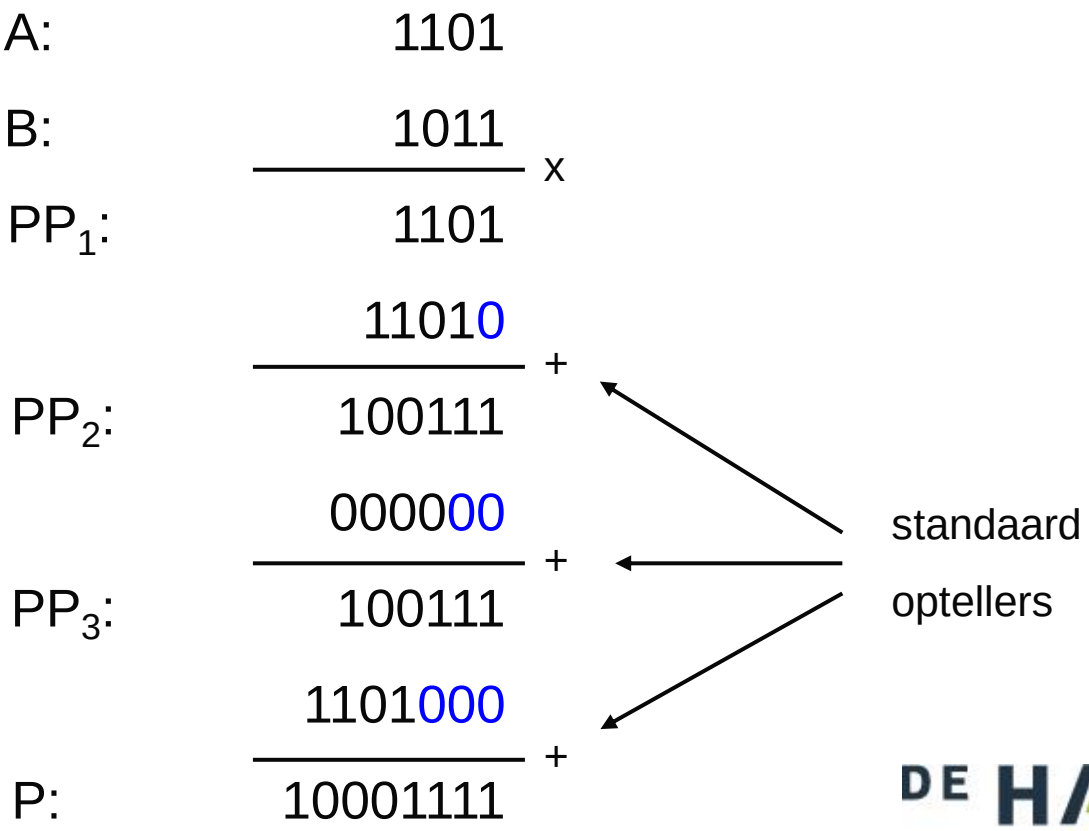
0-en schuiven voor tweetal, viertal, …

Nadeel: multi-input opteller nodig

DE HAAGSE HOGESCHOOL

Vermenigvuldigen

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



Vermenigvuldigen

- Nu worden de optelslagen 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 optelslagen 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 nxn-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-bits register nodig voor B.
- Er is een 2n-bits register nodig voor P.
- Er is een nx1-bits 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.

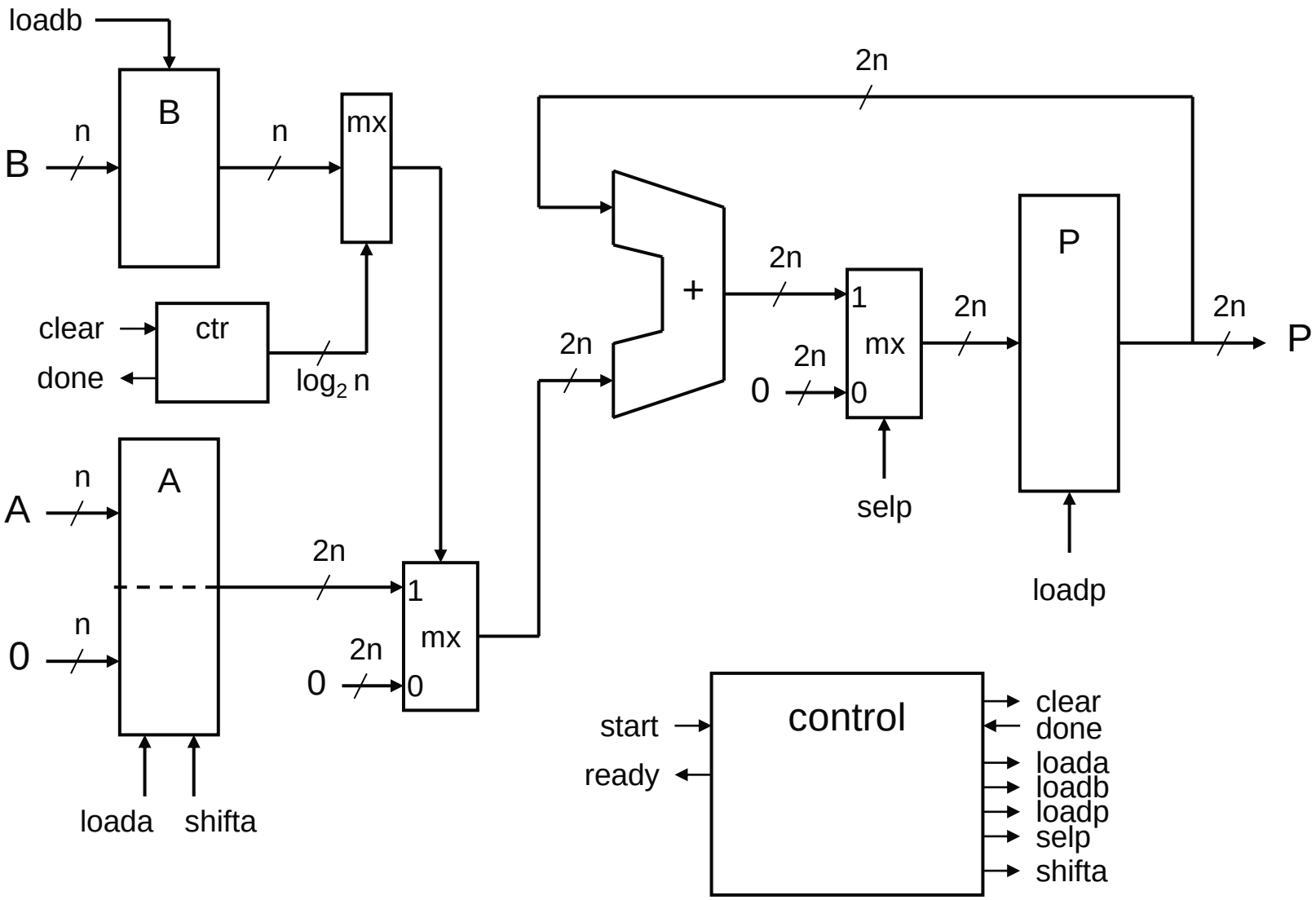
Vermenigvuldiger

- Vervolg:

- Er is een 2x1 2n-bits multiplexer nodig om A of nullen te selecteren afhankelijk van de waarde van het i^e bit van B.
- Er is een 2n-bits 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 2x1 2n-bits 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.

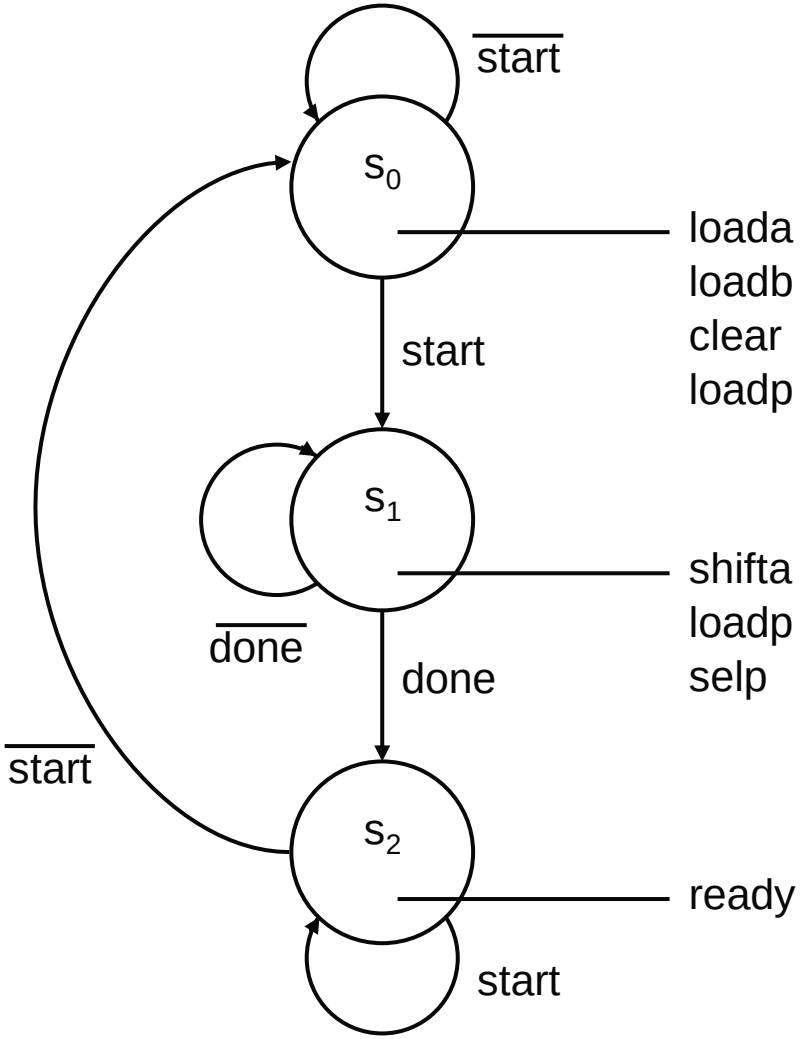
Eerste opzet vermenigvuldiger

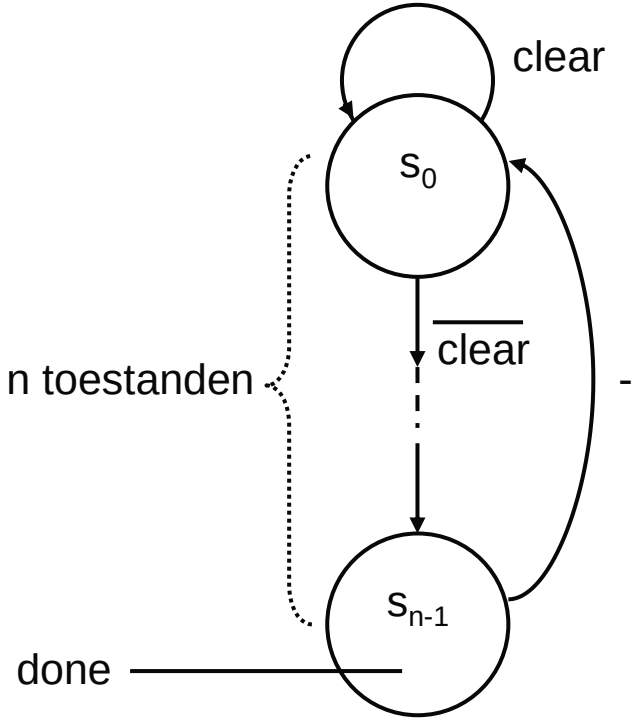


Operatievolgorde

- 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:

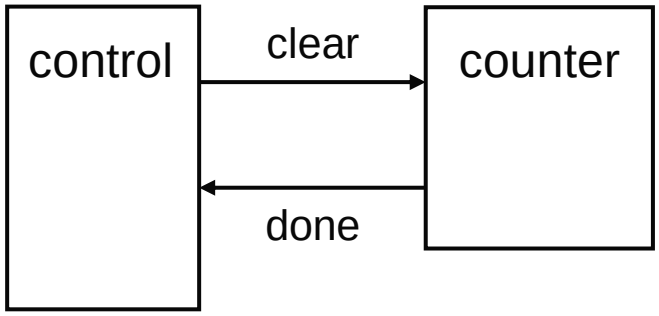
- Bij de overgangen staan nu de signaalnamen, normaal of geinverteerd.
- Uitgangen zijn alleen actief indien genoemd bij een toestand. Ontbreekt een naam bij een uitgang dan is deze dus inactief.

- 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 (uiteraard) op dezelfde klokflank geklokt worden.

Vermenigvuldiger 2.0

- De eerste vermenigvuldiger kan vereenvoudigd worden. Merk op:

- B wordt een schuifregister zodat de nx1-bits 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 2x1 2n-bits mux vervallen.
- De 2x1 2n-bits mux om P te wissen wordt in P geplaatst. Dat levert iets minder hardware op, en minder tekenwerk.

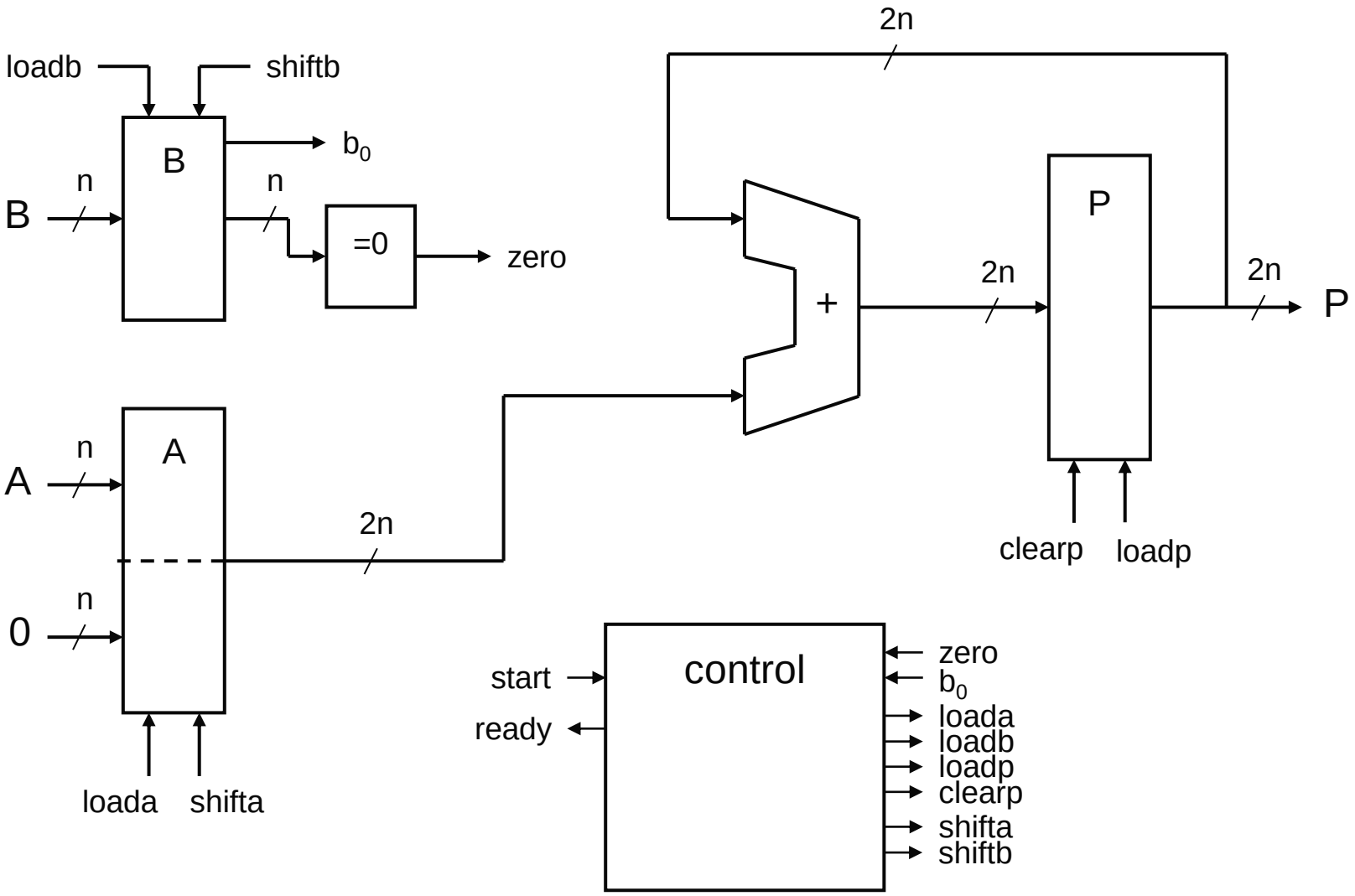
- De besturing heeft nu meer stuuruitgangen en register B is nu een schuifregister geworden. Dat leidt tot meer logica. Verder zijn register A en de opteller nog steeds 2n-bits breed.

Vermenigvuldiger 2.0

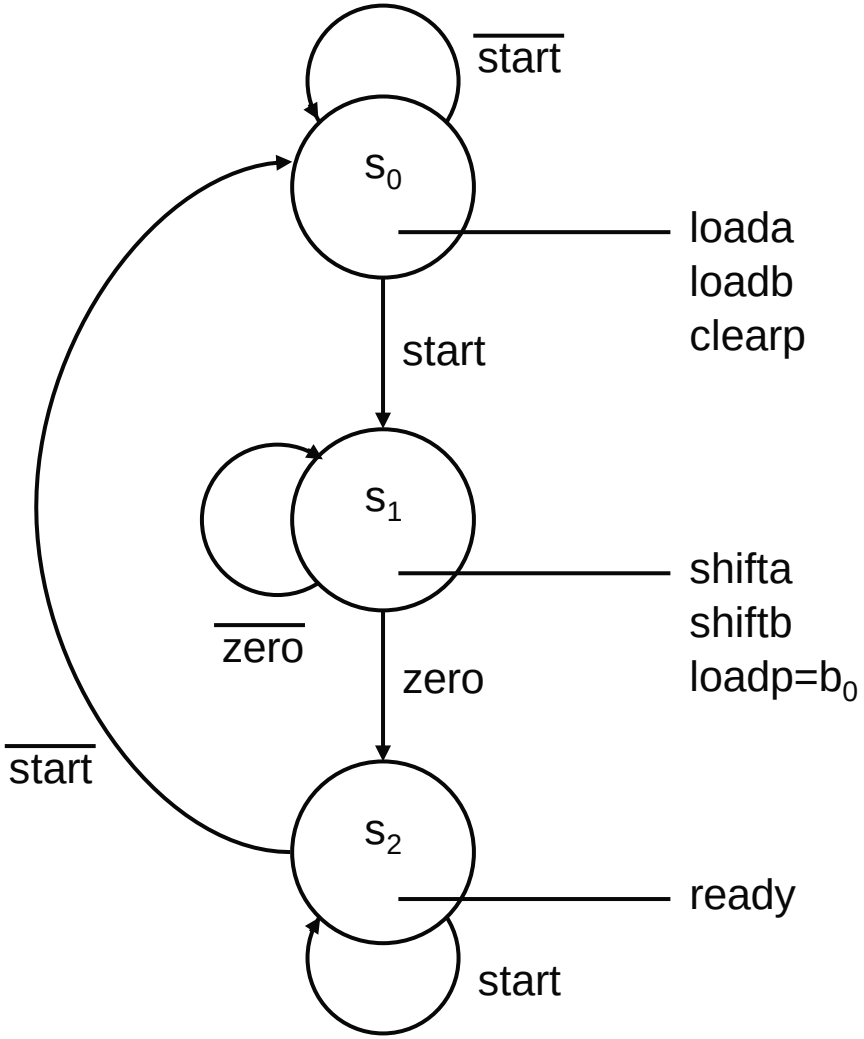
- Voor de sequentiële nxn-bits vermenigvuldiging is een algoritme op te stellen:

```
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.

  shifta en shiftb 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;
```

DE HAAGSE HOGESCHOOL

VHDL-beschrijving control

- Hieronder de toestandsdefiniëring en de NSL mét register ineen.

```
begin
  nsl_reg: process (clk, areset) is
  begin
    if areset = '1' then
      state <= s0;
    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 uitgangslogica

```
  ol: 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

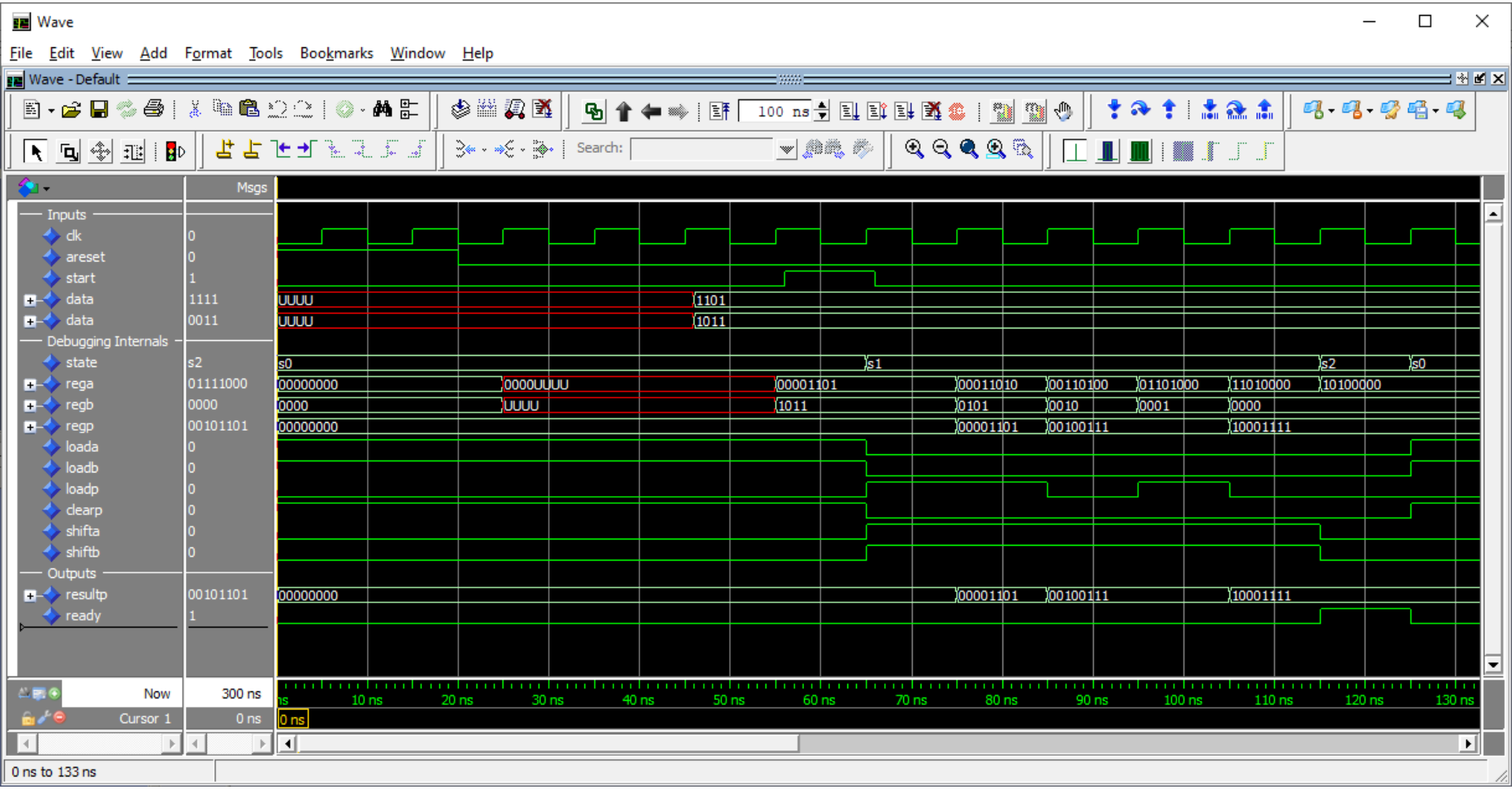
```
datapad: process (clk, areset, regb) is
   begin
       if areset = '1' then
           rega <= (others => '0'); regb <= (others => '0');
           regp <= (others => '0');
       elsif rising_edge(clk) then
           -- Register A.
           if loada = '1' then rega <=  n_zeros & dataa;
           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;
```

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


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
Inputs
clk 0
areset 0
start 1
data 1111
data 0011
Debugging Internals
state s2
rega 01111000
regb 0000
regp 00101101
loada 0
loadb 0
loadp 0
clearp 0
shifta 0
shiftb 0
Outputs
resultp 00101101
ready 1
Now 300 ns
Cursor 1 0 ns
10 ns 20 ns 30 ns 40 ns 50 ns 60 ns 70 ns 80 ns 90 ns 100 ns 110 ns 120 ns 130 ns
0 ns to 133 ns

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;
```

DE HAAGSE HOGESCHOOL

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 and datapad components,
   -- see next slide
   ...

   end if;
end process;
```

DE HAAGSE HOGESCHOOL

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 :
   ...
```

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, 5e 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://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