6

Ontwerpmethoden

Doelstelling

In dit hoofdstuk maak je kennis met een aantal ontwerpmethodieken en leer je werken met behulp van de methode met een gescheiden dataverwerking en besturing.

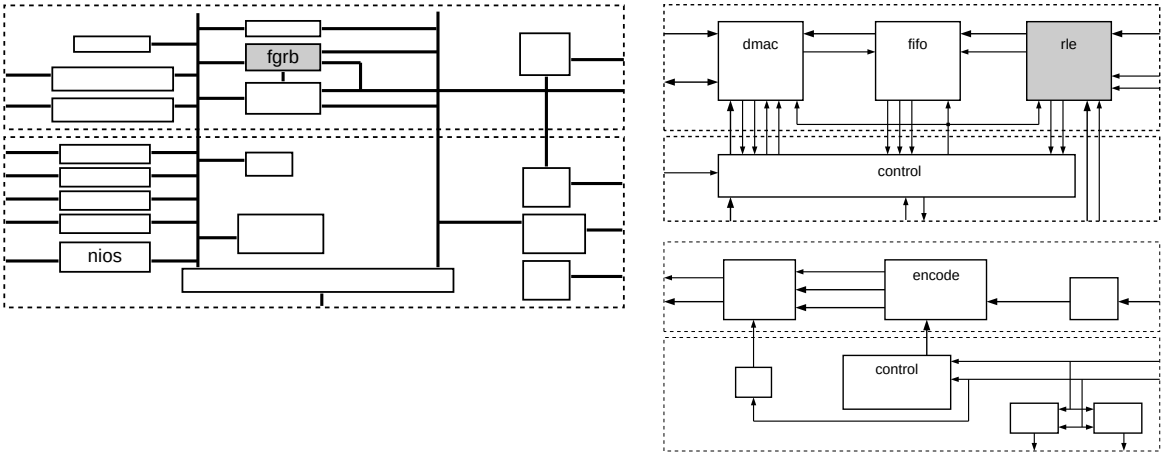
Onderwerpen

De behandelde onderwerpen zijn:

- De architectuur van een complex digitaal systeem.
- De hiërarchie in een complex digitaal systeem.
- De opdeling van een digitaal systeem in een control- of besturingsdeel en een datapad of dataverwerkingsdeel.
- De stuursignalen en statussignalen.
- Twee methoden voor het bepalen van de frequentie: de periodetijdmeting en de frequentiemeting.
- Het ontwerpen met de iteratieve, softwarematige aanpak.
- Het ontwerpen met behulp van de methode met een gescheiden dataverwerking en besturing.
- Het tekenen van het dataverwerkingsgedeelte van een digitaal systeem.
- Het omzetten van een ontwerp met gescheiden dataverwerking en besturing naar een VHDL-beschrijving.
- Het ontwerp met de FSMD-methode.
- Het type `boolean`.
- Toestandsdiagrammen met datapad-aspecten (FSMD).
- Een ASM-chart met datapad-aspecten (ASMD).
- Het twee-processenmethode.

Grote complexe digitale systemen bestaan uit veel verschillende onderdelen, die onderling met elkaar communiceren. Deze onderdelen of subsystemen bestaan zelf weer uit een aantal deelsystemen, die allerlei gegevens aan elkaar doorgeven. Ieder deelsysteem kan op zichzelf weer een compleet digitaal systeem zijn.
Kenmerkend voor een complex digitaal systeem is de hiërarchische opbouw en het parallellisme van al deze deelsystemen. Een digitaal ontwerp heeft een architectuur, die met behulp van blokschema's, stroomdiagrammen en andere grafische tekeningen wordt vastgelegd.
In figuur 6.1 staat een voorbeeld van een complex digitaal systeem. Het hoogste niveau kent achttien deelsystemen. Een van deze onderdelen, het blok `fgrb`, is verder uitgewerkt en bestaat weer uit vier blokken. Daarvan bevat het blok `rlec` op zijn beurt weer zeven onderdelen.

Blokschema's, zoals die in figuur 6.1 zijn getekend, zijn essentieel om het overzicht over het complexe systeem te houden. Een dergelijk groot complex systeem wordt ontworpen door een team van ontwerpers, die allemaal verantwoordelijk zijn voor één of meer subsystemen.



Figuur 6.1 : Een hiërarchisch ontwerp van een complex digitaal systeem. Links staat het blokschema van het hoogste niveau met het complete systeem. Het blokschema van het onderdeel `fgrb` staat rechtsboven en rechtsonder is het onderdeel `rlec` verder uitgewerkt.

Op ieder niveau van het ontwerp is een dataverwerkingsdeel en een besturingsgedeelte te onderscheiden. In de blokschema's van figuur 6.1 bevindt het dataverwerkingsdeel zich steeds bovenaan en de besturing onderaan de tekeningen. Ook de embedded NIOS-processor, linksonder in het blokschema van het hoogste niveau, bevat een dataverwerkingsdeel met een rekeneenheid en dataregisters en een besturingsdeel dat de programmacode vertaalt naar functionele bewerkingen.
Om een complex digitaal systeem te maken, zijn een vaste ontwerpmethodiek, een zorgvuldige documentatie, een systematische aanpak en eenduidige blokschema's essentieel.

In hoofdstuk 2 tot en met hoofdstuk 4 zijn relatief eenvoudige, combinatorische en sequentiële bouwblokken besproken. Dit hoofdstuk behandelt verschillende ontwerpmethoden voor een middelgroot digitaal systeem. Dat is een deelontwerp op het laagste niveau van figuur 6.1. Deze systemen zijn opgebouwd uit zowel combinatorische als sequentiële processen, waarvan de functionaliteit met een paar tekeningen kan worden vastgelegd en waarvan de hoeveelheid VHDL-code niet groter is dan duizend regels.
Niet alles hoeft vanaf niets te worden opgebouwd. Softcores en megafuncties geven vaak complete oplossingen voor deelproblemen. In figuur 6.1 representeert bijvoorbeeld het blok NIOS een complete NIOS-processor.
Dit hoofdstuk richt zich op methodieken om een compleet, nieuw systeem te maken. De besproken methodieken zijn:

- de iteratieve softwarematige aanpak,
- de methode met gescheiden dataverwerking en besturing,
- de FSMD-methode,
- de ASMD-methode,
- de twee-processenmethode.

6.1 Ontwerpvoorbeeld : de elektronische personenweegschaal

Het voorbeeld van de elektronische personenweegschaal is ontleend aan *Ontwerpen van digitale MOS-IC's* van J. van Dijken et. al., dat in 1994 is uitgegeven door Nijgh & Van Ditmar.

Dit hoofdstuk gebruikt één specifiek voorbeeld om al de verschillende ontwerpmethodieken te bespreken.

De specificatie: de eisen aan de elektronische personenweegschaal

Een elektronische personenweegschaal heeft een slimme druksensor, die een pulstrein afgeeft waarvan de frequentie f evenredig is met het gewicht G dat de sensor detecteert:

$$f = k\,G \tag{6.1}$$

Ieder ontwerp begint met een studie naar de mogelijkheden. In dit voorbeeld is dat de meetmethode, maar in een andere situatie is dat een studie van de bestaande toestand of een studie naar een nieuw te gebruiken ontwerptechniek. Hier is de term analyse gebruikt, anderen spreken liever van een onderzoek of vooronderzoek.

De constante k is een evenredigheidsconstante en is in dit voorbeeld 100 Hz/kg. Het bereik van de weegschaal ligt tussen 10 en 150 kg. Het gewicht wordt met een nauwkeurigheid van 0,1 kg op een vier-cijferig digitaal display afgebeeld. De aanduiding op het display moet minimaal iedere halve seconde ververst kunnen worden. De sensor reageert binnen 10 ms op een gewichtsverandering.

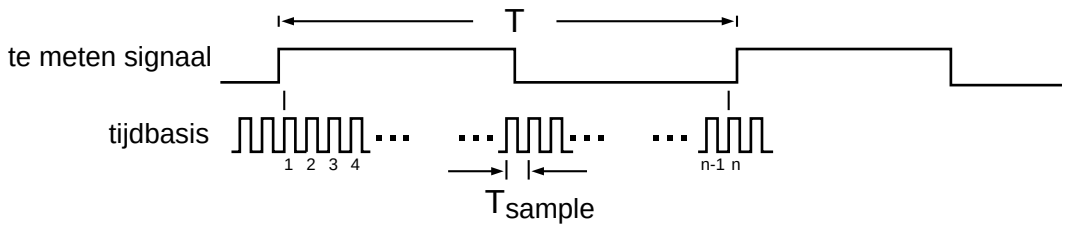
De analyse: het bepalen van een periodetijd of een frequentie

Het bepalen van de periodetijd of de frequentie van een periodiek signaal is in wezen hetzelfde. Als de frequentie f bekend is, is de periodetijd T ook bekend en omgekeerd:

$$f = \frac{1}{T} \tag{6.2}$$

In een digitaal systeem wordt tijd altijd gerelateerd aan het kloksignaal of aan een van de klok afgeleid signaal. Voor het bepalen van de periodetijd en het meten van de frequentie is altijd een teller nodig. Er zijn twee principieel verschillende meetmethoden:

- **Periodetijdmeting**
  Bij deze meting wordt gedurende een periode van het periodieke signaal het aantal klokslagen n geteld.



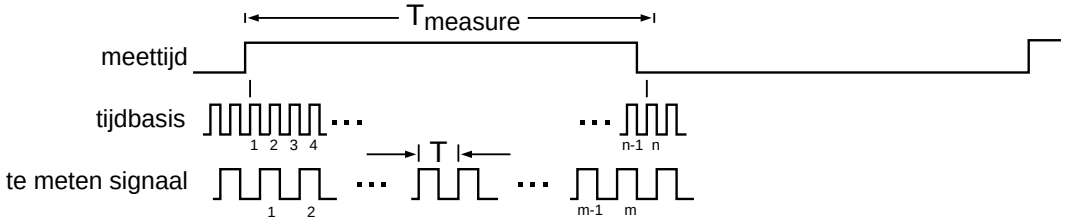
Figuur 6.2 : **De periodetijdmeting.** Het aantal pulsen van de tijdbasis is evenredig met de periodetijd van het te meten signaal.

Als T_{sample} de periodetijd van de systeemklok is of een daarvan afgeleide klok en de periodetijd van het te meten signaal T is, geldt:

$$T = n\,T_{\text{sample}} \tag{6.3}$$

De periodetijd van het te meten signaal is evenredig met het aantal gemeten *samples* of klokslagen

- **Frequentiemeting**
  Bij deze meting wordt gedurende een vooraf vastgestelde meettijd het aantal perioden van het te meten signaal geteld.



Figuur 6.3 : De frequentiemeting. De frequentie van het te meten signaal is evenredig met het aantal samples.

Figuur 6.3 geeft een grafische weergave van deze meetmethode. Er zijn twee tellers nodig. Eén teller definieert de meettijd T_{measure}, door bijvoorbeeld n klokslagen te tellen. De andere teller telt het aantal perioden m van het te meten signaal.

$$mT = T_{\text{measure}} \qquad \text{of} \qquad f = m f_{\text{measure}} \tag{6.4}$$

De frequentie van het te meten signaal is evenredig met het aantal gemeten *samples* of klokslagen m.

De periodetijdmeting is snel; de periodetijd is bekend na één periode van het te meten signaal. Een nadeel is dat bij een hoge nauwkeurigheid de samplefrequentie hoog moet zijn. Voor snelle signalen is deze meting minder geschikt.
De frequentiemeting is geschikt voor signalen met een hoge frequentie. In een korte tijd kunnen er dan heel veel perioden geteld worden. Voor langzame signalen — en dat geldt zeker bij nauwkeurige metingen — wordt de meettijd te lang.
Voor signalen, die niet snel en die niet langzaam zijn, kan een variant op de periodetijdmeting worden toegepast. In plaats van over één periode kan er over meerdere perioden worden geteld. De gezochte periodetijd is dan het gemiddelde over de verschillende perioden.

Voor de periodetijdmeting volgt uit formule 6.1, 6.2 en 6.3 dat het gewicht G omgekeerd evenredig is met het aantal getelde klokslagen n:

$$G = \frac{1}{n} \frac{f_{\text{sample}}}{k} \tag{6.5}$$

Naast het gegeven dat de samplefrequentie hoog moet zijn om de gewenste nauwkeurigheid te halen, is er voor de berekening een deling nodig om het gewicht te bepalen. De deling is niet standaard synthetiseerbaar en als deze wel gesynthetiseerd kan worden, is daar veel logica voor nodig.

Bij de frequentiemeting is het gewicht evenredig met het aantal getelde klokslagen. Uit formule 6.1 en 6.4 volgt:

$$G = m \frac{f_{\text{measure}}}{k} \tag{6.6}$$

Bij deze methode is het gewicht evenredig met het aantal getelde klokslagen. Bovendien kan het ontwerp verder vereenvoudigd worden door f_{measure} slim te kiezen. De meettijd kan zo gekozen worden dat het gemeten aantal klokslagen overeenkomt met het gewicht in tiende kilogrammen. Deze situatie doet zich voor als $m = 10G$. Voor een k van 100 Hz/kg is f_{measure} gelijk aan 10 Hz. De meettijd T_{measure} is dan gelijk aan 0,1 s

Het aantal perioden van het ingangssignaal dat in 0,1 s past, geeft het gewicht in tiende kilogrammen. Deze meettijd voldoet aan de eis dat het display minimaal iedere halve seconde ververst moet worden.

6.2 De iteratieve softwarematige aanpak

Deze paragraaf geeft een gedragsbeschrijving voor de elektronische personenweegschaal met behulp van een iteratieve softwarematige aanpak. Deze aanpak beschrijft het gedrag eerst in gewone, Nederlandstalige zinnen. Stap voor stap wordt de beschrijving steeds verder verfijnd en tenslotte leidt dit tot een VHDL-beschrijving van de weegschaal.
De verwerkingseenheid van de elektronische personenweegschaal doet voortdurend twee dingen: het telt het aantal pulsen gedurende de meettijd en zet het resultaat daarna op het display. Een eerste aanzet kan dus zijn:

```
Tel het aantal pulsen gedurende de meettijd van 0,1 seconde
Zet het resultaat op het display
```

De hier gebruikte aanpak heeft als voordeel dat het bedenken van de oplossing vanuit het Nederlands gedaan wordt.

Het achterliggende idee van deze aanpak is dat als je het probleem niet in gewoon Nederlands kunt uitleggen, je het dan zeker niet in een taal als C, Java of VHDL kunt uitleggen.

De Nederlandse teksten zijn schuingedrukt en staan tussen dubbele aanhalingstekens.

De beslissing of er geteld wordt of dat het resultaat op het display gezet wordt, kan gerealiseerd worden met een if-statement:

```
if "meettijd < 100 ms" then
  "tel het aantal pulsen"
else
  "zet resultaat op het display"
  "begin opnieuw met meten"
end if;
```

Een digitaal systeem voor een FPGA zal altijd synchroon zijn. Dit gedrag kan met een proces met een wachtopdracht worden gerepresenteerd. Verder wordt het aantal pulsen alleen verhoogd als er een nieuwe puls is en wordt dit aantal weer nul gemaakt voor er een nieuwe meting uitgevoerd wordt.

```
aanzet_3 : process is
begin
  if "meettijd < 100 ms" then
    if "nieuwe puls" then
      aantalpulsen <= aantalpulsen + 1;
    end if;
  else
    "zet resultaat op het display"
    aantalpulsen <= 0;
    "begin opnieuw met meten"
  end if;
  wait for sample_time;
end process aanzet_3;
```

Het wait-for-statement is — net als andere toewijzingen met een tijdsaspect — niet synthetiseerbaar. Een digitaal systeem heeft altijd een systeemklok nodig voor de definitie van tijd. Het meten van een bepaalde tijdsduur komt altijd overeen met het tellen van een aantal klokslagen. Voor een systeemklok van 100 kHz is een meettijd van 100 ms gelijk aan 10000 klokslagen. In het ontwerp zijn dus twee tellers nodig: één voor het aantal pulsen en één voor het aantal klokslagen. Nadat het resultaat op het display is gezet, worden beide tellers nul gemaakt.
In code 6.1 staat de vierde aanzet voor de gedragsbeschrijving van de elektronische personenweegschaal. Twee aspecten zijn nog niet goed beschreven: het detecteren van een nieuwe puls en het op het display zetten van het resultaat.

De pulsdetectie kan worden beschreven met het attribuut `'last_value`. Dit signaalattribuut geeft de vorige waarde van een signaal. Als de laatste waarde laag en de huidige waarde hoog is, is er een nieuwe puls. De vijfde aanzet uit code 6.2

Code 6.1 : Vierde aanzet voor gedragsbeschrijving van elektronische personenweegschaal.

```
architecture gedrag of epw is
  signal cc          : integer;
  signal aantalpulsen : integer;
begin
  aanzet_4 : process (clk) is
  begin
    if rising_edge(clk) then
      if cc < 10000 then
        if "nieuwe puls" then
          aantalpulsen <= aantalpulsen + 1;
        end if;
        cc <= cc + 1;
      else
        "zet resultaat op het display"
        aantalpulsen <= 0;
        cc <= 0;
      end if;
    end if;
  end process aanzet_4;
end architecture gedrag;
```

bevat een procedure `send_to_display`, die het aantal getelde pulsen op de juiste manier op een 4-cijferig digitaal display afbeeldt. Ook is aan het proces een actief lage asynchrone reset toegevoegd.

Code 6.2 : Vijfde aanzet voor gedragsbeschrijving van elektronische personenweegschaal.

```
 1  architecture gedrag of epw is
 2    signal cc          : integer;
 3    signal aantalpulsen : integer;
 4  begin
 5    aanzet_5 : process (clk, rst_n) is
 6    begin
 7      if rst_n = '0' then
 8          aantalpulsen <= 0;
 9          cc <= 0;
10      elsif rising_edge(clk) then
11        if cc < 10000 then
12          if (puls'last_value = '0') and (puls = '1') then
13            aantalpulsen <= aantalpulsen + 1;
14          end if;
15          cc <= cc + 1;
16        else
17          send_to_display(aantalpulsen);
18          aantalpulsen <= 0;
19          cc <= 0;
20        end if;
21      end if;
22    end process aanzet_5;
23  end architecture gedrag;
```

In code 6.2 heeft de procedure `send_to_display` geen uitgangen. Normaal gesproken zouden dat de aansluitingen van het display, dat het gewicht afbeeldt, moeten zijn. In dit geval schrijft de procedure `send_to_display` bij het simuleren het gewicht als tekst naar het scherm. Deze vijfde aanzet is ook om deze reden niet synthetiseerbaar.

In paragraaf 10.11 wordt het textio-package waarmee tekst naar het scherm geschreven kan worden besproken.

Mits de procedure `send_to_display` uit code 6.2 het aantal pulsen op een juiste wijze converteert naar vier cijfers, beschrijft deze code het gedrag van de elektronische personenweegschaal. Helaas bevat deze code twee lastige problemen. Het

attribuut `last_value` is niet synthetiseerbaar en het afbeelden van een binair getal als BCD-gecodeerd getal is niet triviaal.

Voor de detectie van een nieuwe puls moet de vorige waarde bewaard worden, daar is een extra intern signaal voor nodig. In code 6.3 krijgt bij iedere klokslag het signaal `vorige_puls` de waarde van `puls`. Het signaal `vorige_puls` bevat dus altijd de vorige waarde van `puls`. Hiermee is een nieuw puls eenvoudig te detecteren.
De pulsen worden BCD-gecodeerd geteld. De functie `incrementBCD` verhoogt het aantal pulsen met één en de functie `to_display` zet de vier BCD-cijfers om naar een 28-bits signaal `gewicht` dat de vier 7-segmentsdisplays aanstuurt.

Code 6.3 : Zesde aanzet voor gedragsbeschrijving van elektronische personenweegschaal.

```
12    aanzet_6 : process (clk, rst_n) is
13    begin
14      if rst_n = '0' then
15          aantalpulsen <= (others => '0');
16          cc <= 0;
17      elsif rising_edge(clk) then
18        if cc < 10000 then
19          if (vorige_puls  = '0') and (puls = '1') then
20            aantalpulsen <= incrementBCD(aantalpulsen);
21          end if;
22          cc <= cc + 1;
23        else
24          gewicht <= to_display(aantalpulsen);
25          aantalpulsen <= (others => '0');
26          cc <= 0;
27        end if;
28        vorige_puls <= puls;
29      end if;
30    end process aanzet_6;
31
32    dig_dec  <= gewicht(6 downto 0);
33    dig_unit <= gewicht(13 downto 7);
34    dig_ten  <= gewicht(20 downto 14);
35    dig_hund <= gewicht(27 downto 21);
```

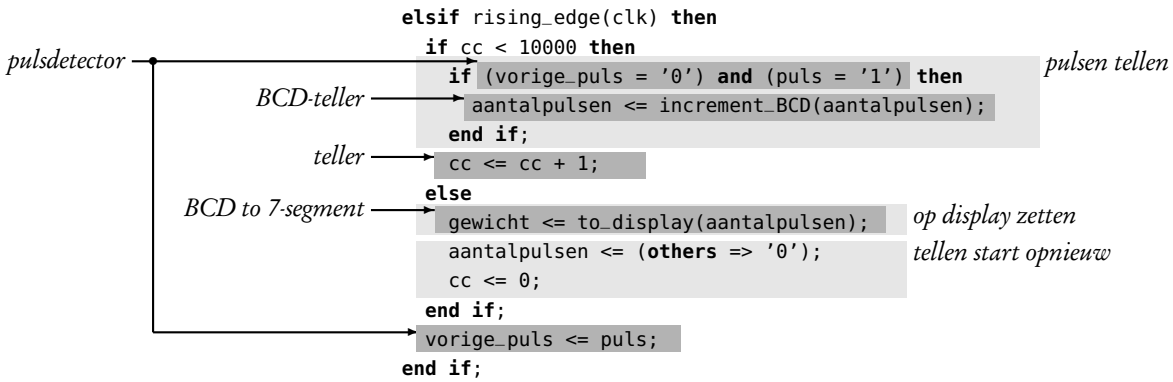
Er is gekozen om het aantal pulsen BCD-gecodeerd te tellen, omdat het omzetten van een binair getal naar een BCD-gecodeerd getal lastig is. Deze conversie kan op twee manieren worden opgelost: parallel en sequentieel. Bij de parallelle methode is er veel logica nodig en bij de sequentiële oplossing zijn er meerdere klokslagen nodig voor de conversie. Bovendien is het niet eenvoudig om de sequentiële variant in het proces te implementeren: functies mogen bijvoorbeeld in VHDL geen wachtopdrachten bevatten. Fundamenteel hierbij is dat het proces iedere klokslag nieuwe waarden voor `vorige_puls` en `cc` moet bepalen en dat er voor de conversie meer klokslagen nodig zijn.

De keuze voor BCD-gecodeerd tellen is bij het ontwikkelen van hardware een logische keuze. Een BCD-teller is een standaard component die deel uitmaakt van iedere logische familie. Een gedragsbeschrijving van een BCD-teller is niet eenvoudig, maar BCD-tellers zijn binnen veel ontwikkelomgevingen wel beschikbaar als standaard bouwsteen. De hier gebruikte functie `incrementBCD` komt in paragraaf 8.3 aan de orde en de BCD-teller in paragraaf 8.4.

Bespreking softwarematige aanpak

Een volgende, logische stap in het ontwerpproces zou een nieuwe opzet kunnen zijn, die gebruik maakt van een aparte BCD-teller, pulsdetector en displayeenheid. Dit voortschrijdend inzicht ontstaat bijna altijd als geprobeerd wordt een gedragsbeschrijving met één proces te maken. Omdat het ontwerp uiteindelijk als hardware gerealiseerd wordt, is het verstandig het ontwerp direct in functionele blokken in te delen.
Als er bij de zesde aanzet gekozen was om binair te tellen en het resultaat te converteren naar een BCD-gecodeerd signaal, kan de beschrijving zeer ingewikkeld worden. Het is bijna ondoenlijk om de seriële BCD-conversie in deze oplossing in te bouwen. Dat is een fundamenteel probleem. Een digitaal systeem bevat altijd aspecten die te maken hebben met de verwerking van gegevens en aspecten die deze verwerking besturen. De verwerking van gegevens bevat typisch parallellisme. De tellers voor het tellen van het aantal klokslagen en het aantal pulsen voeren hun taken gelijktijdig uit. De besturing is daarentegen juist sequentieel. Na het tellen van de pulsen wordt het resultaat op het display gezet en start het tellen opnieuw. In veel gevallen is de besturing niet meer dan een toestandsmachine.

De dataverwerking van de elektronische personenweegschaal bestaat uit een pulsdetector, een BCD-teller voor het aantal pulsen en een omzetting van de BCD-waarde naar de vier 7-segmentsdisplays. De besturing bestaat uit drie acties: het tellen van de pulsen, het op het display zetten van het resultaat en het opnieuw starten van de meting. In code 6.3 staan deze twee aspecten — namelijk de dataverwerking en de besturing — allemaal door elkaar. In figuur 6.4 is dit gevisualiseerd. Onderdelen van de dataverwerking zijn over de hele code verdeeld en vaak in verschillende stukken geknipt. De pulsdetector bestaat uit twee stukken: het testen op de flank staat bij het if-statement en het toekennen van signaal `puls` aan `volgende_puls` staat aan het einde van de code.



Figuur 6.4 : Dataverwerking en besturing bij de softwarematige aanpak. De verschillende aspecten staan door elkaar en overlappen elkaar. De dataverwerkingsdelen hebben een donker grijze achtergrond. De besturingsdelen hebben een licht grijze achtergrond.

Voor beginnende VHDL-ontwerpers is het moeilijk deze twee aspecten te onderscheiden. Zeker in combinatie met de parallelle en sequentiële constructies van de taal levert dat vaak een onleesbare, moeilijk te begrijpen code op. Ervaren VHDL-ontwerpers kunnen hier vaak wel goed mee omgaan en maken wel leesbare beschrijvingen met grote complexe processen.

Het voordeel van grote geïntegreerde blokken is dat de simulatietijd kort kan zijn. Een beschrijving, die uit veel kleine processen bestaat, simuleert langzamer. Er zijn dan veel interne signalen. Voor ieder signaal heeft de simulator een databasestructuur nodig waarin allerlei attributen worden bijgehouden. Voor een beschrijving met een klein aantal grote processen zijn minder interne signalen nodig. De interne databasestructuur is dan veel eenvoudiger en de simulatie zal sneller zijn.

6.3 De methode met gescheiden dataverwerking en besturing

Het voorbeeld met de elektronische personenweegschaal laat zien dat een digitaal systeem bestaat uit onderdelen, die typisch te maken hebben met dataverwerking en uit onderdelen bestaan die vooral te maken hebben met besturing.

Het dataverwerkingsdeel wordt ook *data path* of datapad genoemd.

De dataverwerking bestaat uit verschillende blokken die tegelijkertijd naast elkaar hun taak uitvoeren. De personenweegschaal bevat een pulsdetector, een BCD-teller en een omzetter voor het gemeten resultaat. Deze blokken bestaan naast elkaar en voeren hun taken parallel uit.

In dit boek wordt meestal dataverwerking of dataverwerkingsdeel gebruikt omdat de uitspraak en de betekenis van *path* en pad verwarring geeft.
Het Engelse woord *path* betekent in het Nederlands pad in betekenis van route of traject. Het Engelse woord *pad* betekent kussentje.

De besturing zorgt ervoor dat de gegevens op een correcte manier door het dataverwerkingsdeel worden geleid. De besturing is meestal een toestandsmachine die op het juiste moment de stuursignalen van de dataverwerking hoog maakt. Het besturingsdeel kenmerkt zich door het sequentiële karakter van de toestanden die het digitale systeem moet doorlopen.

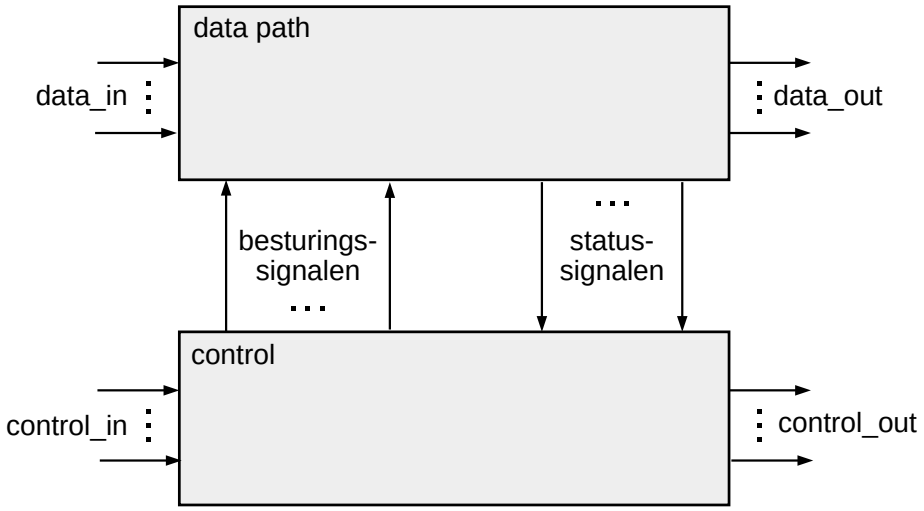
Omdat dataverwerking vooral parallelle kenmerken heeft en het besturingsdeel vooral sequentieel is, ligt het voor de hand deze twee te scheiden. Deze paragraaf behandelt de methode met een gescheiden dataverwerking en besturing. Bij deze methode wordt eerst het dataverwerkingsdeel getekend en daarna het bijbehorende toestandsdiagram. Bij het tekenen van het dataverwerkingsdeel worden alle tijdsaspecten buiten beschouwing gelaten. In de tekening van de dataverwerking staan alleen de registers waarmee de verschillende gegevens bewaard worden en de bewerkingen, die met deze gegevens worden uitgevoerd. Naderhand worden de verschillende scenario's dit het systeem kan doorlopen opgesteld en wordt dit verder uitgewerkt in een toestandsdiagram.

Het Engelse woord voor besturing is *control*. Het Nederlandse woord controle is afgeleid van controleren, dat naast beheersen ook nagaan, nazien of checken kan betekenen.
In dit boek wordt meestal besturing gebruikt, omdat dit in het Nederlands meer eenduidig is.

Model voor gescheiden dataverwerking en besturing

In figuur 6.5 staat een blokschema van een digitaal systeem dat opgebouwd is uit een dataverwerkingsdeel en een besturingsdeel. Aan de linkerzijde staan de ingangssignalen van het systeem. Eén deel van deze signalen bevat de gegevens die door het dataverwerkingsdeel bewerkt worden, het andere deel stuurt het besturingsdeel van het systeem aan. Aan de rechterzijde staan de uitgangssignalen van het systeem. Een deel van de uitgangssignalen bevat het resultaat van de dataverwerking. Het andere deel van de uitgangssignalen geeft informatie over de toestand van het systeem.
De besturingssignalen of stuursignalen van het besturingsdeel naar het dataverwerkingsdeel zorgen dat de enable-signalen van registers en tellers en de selectiesignalen van multiplexers op het juiste moment actief zijn. Informatie over

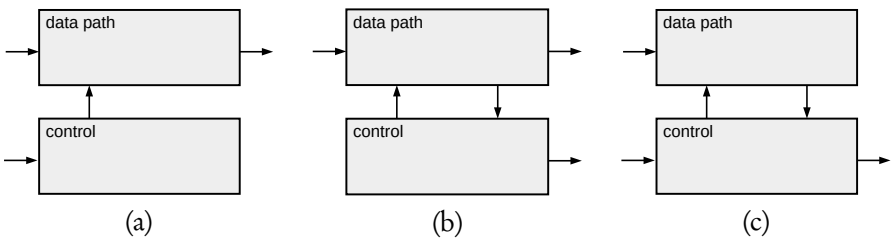
De methode met gescheiden dataverwerking en besturing wordt ook de *data path/control-* of *datapad/control*-methode genoemd.



Figuur 6.5 : Een digitaal systeem is opgebouwd uit een dataverwerkingsdeel of *data path* en een besturingsdeel of *control*.
Naast de ingang- en uitgangssignalen zijn er besturingssignalen van het besturingsdeel naar het dataverwerkingsdeel en kunnen er statussignalen zijn van het dataverwerkingsdeel naar het besturingsdeel.

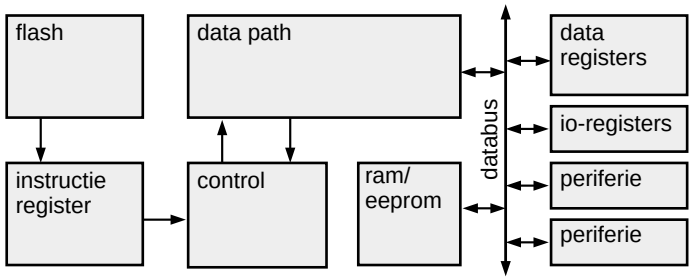
de status van de bewerkingen wordt via de statussignalen aan het besturingsdeel doorgegeven. Voorbeelden zijn de overflow bij de optelfunctie en de overflow van een teller.

Niet alle signalen hoeven in ieder digitaal systeem aanwezig te zijn. Figuur 6.5 is een algemeen model die in veel varianten voorkomt. In figuur 6.6.a staat een voorbeeld van een systeem zonder statussignalen en met een besturing zonder uitgangssignalen. Figuur 6.6.b is een systeem met een autonome besturing. Het besturingsdeel heeft geen externe ingangssignalen Het systeem uit figuur 6.6.c heeft geen uitgangssignalen bij de dataverwerking.



Figuur 6.6 : Drie varianten op het digitale systeem met gescheiden dataverwerking en besturing.

Een microprocessor en een microcontroller voldoen ook aan het model met een gescheiden dataverwerking en besturing. In figuur 6.7 staat een vereenvoudigd schema van een microcontroller. Het dataverwerkingsdeel van de microcontroller bevat onder andere de ALU of rekeneenheid en het besturingsdeel krijgt de opdrachten uit het instructieregister. De dataverwerking is bij een microcontroller aangesloten op de databus, die ook is verbonden met de dataregisters, het datageheugen, de io-registers en allerlei perifere blokken, zoals een UART en ADC.



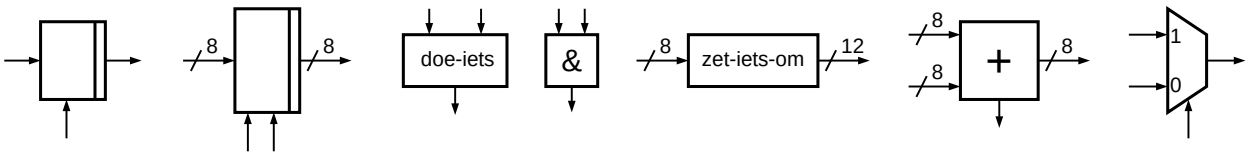
Figuur 6.7 : **Vereenvoudigd schema van een microcontroller.** Een microcontroller bevat een dataverwerkingsdeel en een besturing. De dataverwerking omvat ondermeer de ALU en de besturing krijgt de instructies uit het instructieregister.

6.4 Tekenen dataverwerkingsdeel

Het dataverwerkingsdeel bestaat uit allerlei sequentiële componenten, zoals dataregisters, schuifregisters, tellers en flipfloppen en uit combinatorische componenten, zoals multiplexers, optellers, vermenigvuldigers en andere rekenkundige en logische bewerkingen.
Van het ontwerp van het dataverwerkingsdeel wordt eerst een tekening gemaakt, die aan de volgende eisen voldoet:

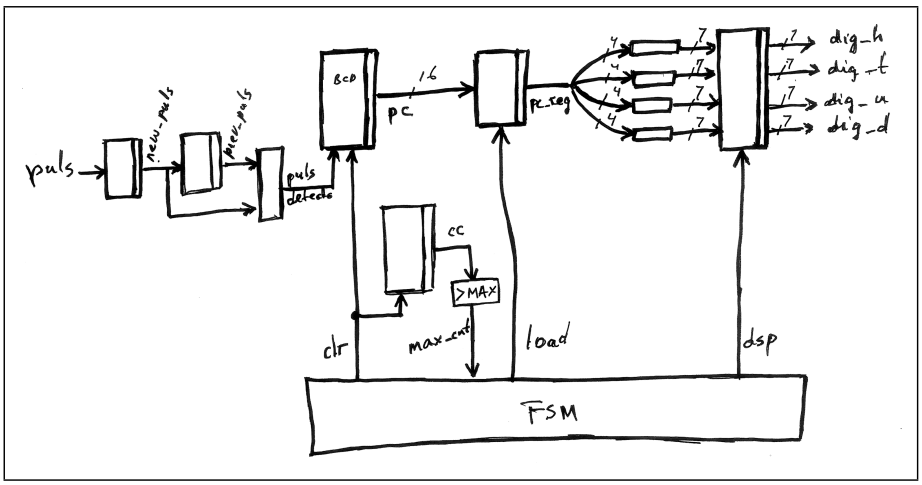
- de tekening bevat alleen functionele signalen;
- het kloksignaal en de globale asynchrone reset worden niet getekend;
- sequentiële componenten krijgen aan de rechter zijde een verticale streep;
- de tekening bevat simpele blokken en symbolen;
- bij de blokken en symbolen staat eventueel een beknopte toelichting;
- alle signalen hebben een naam;
- voor alle signalen is de busbreedte aangegeven.

De tekening, blokken en symbolen moeten eenvoudig zijn. Het is een hulpmiddel bij het ontwerp. De tekening hoeft niet volledig te zijn. Figuur 6.8 geeft een aantal voorbeelden van symbolen.



Figuur 6.8 : **Symbolen voor een tekening van de dataverwerking.** Rechts staan een flipflop en een 8-bits dataregister. Daarnaast staan vijf verschillende combinatorische functies.

Met behulp van potlood en papier is een schets van de dataverwerking snel te maken. In figuur 6.9 staat de schets voor de dataverwerking van de elektronische personenweegschaal. Samen met het toestandsdiagram beschrijft deze tekening het hele ontwerp. Zonder dat er over VHDL gesproken wordt, kunnen belangrijke ontwerpbeslissingen genomen worden. Bij het tekenen is besloten een BCD-teller voor het aantal pulsen te gebruiken. Dit is expliciet gemaakt door in het symbool van de teller *bcd* te zetten.



Figuur 6.9 : Een schets van de dataverwerking voor de elektronische personenweegschaal.

Er is een register `pc_reg` toegevoegd om het aantal getelde pulsen te bewaren. Strikt genomen is dit register overbodig. De reden om dit register toe te voegen is dat hierna een flink stuk combinatoriek volgt, namelijk vier BCD-to-7segment-converters. Zonder register `pc_reg` veranderen de converters voortdurend van waarde en wordt er voortdurend energie gedissipeerd. Met register `pc_reg` veranderen de ingangen alleen als er een nieuwe waarde naar het display wordt geschreven.

De entity is hier weggelaten. Deze bevat naast de ingang `puls` en de vier 7-bits uitgangen `dig_hund`, `dig_ten`, `dig_unit`, `dig_dec` een kloksignaal `clk` en een asynchrone reset `rst_n`.

Tijdens het ontwerptraject wordt de schets voortdurend aangepast of moet zelfs opnieuw getekend worden. Alle tussenstappen worden in het projectdossier bewaard. Pas bij het documenteren is het nodig een nette tekening van de dataverwerking te maken. In figuur 6.10 staat het definitieve datapad.

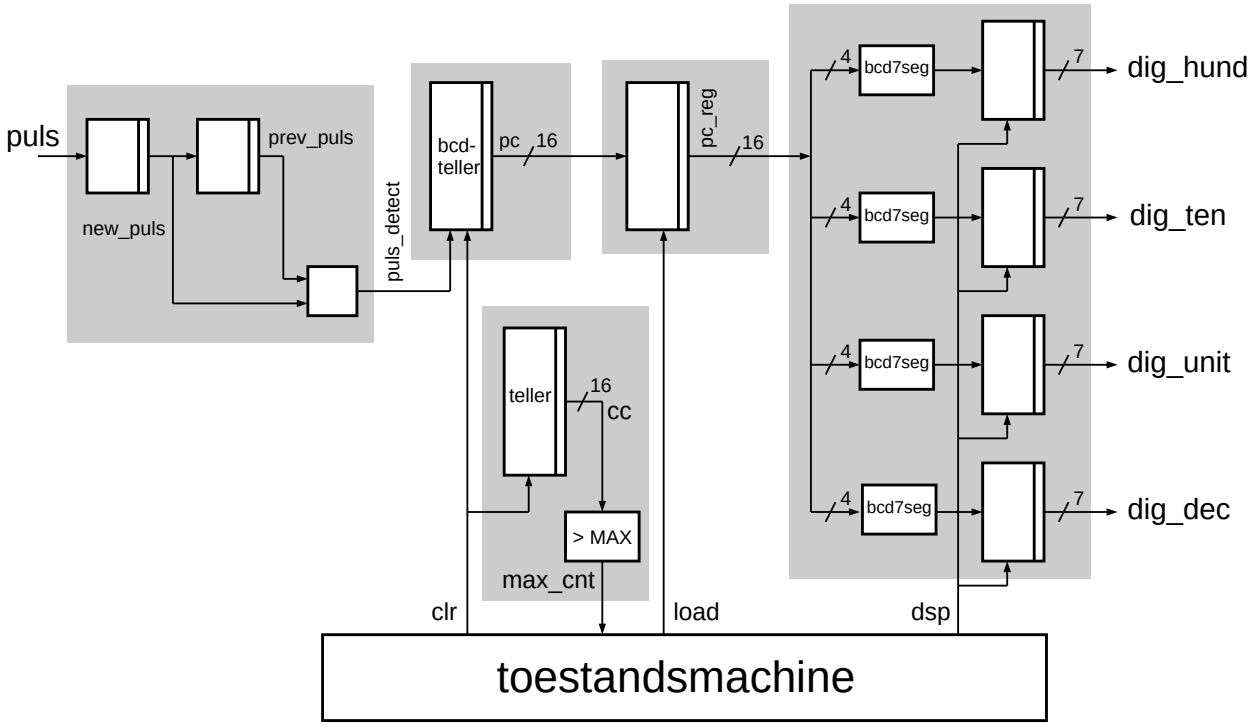
Code 6.4 : De interne signalen van het dataverwerkingsdeel.

```
30   signal pc_reg        : std_logic_vector(15 downto 0);
31   signal pc            : unsigned(15 downto 0);
32   signal cc            : unsigned(15 downto 0);
33   signal new_puls      : std_logic;
34   signal prev_puls     : std_logic;
35   signal puls_detect   : std_logic;
36   signal clr           : std_logic;
37   signal dsp           : std_logic;
38   signal load          : std_logic;
39   signal max_cnt       : std_logic;
```

6.5 Uitwerking dataverwerkingsgedeelte in VHDL

De onderdelen uit het dataverwerkingsdeel kunnen één op één naar VHDL vertaald worden. De tekening bevat tien interne signalen, onder andere de drie stuursignalen `clr`, `dsp` en `load` die van de toestandsmachine komen en het statussignaal `max_cnt` dat naar de toestandsmachine wijst. Samen met de zes andere interne signalen staan deze in het fragment van code 6.4.

In code 6.5 staat de VHDL-beschrijving van het deel met de pulsdetector. De



Figuur 6.10 : De dataverwerking voor de elektronische personenweegschaal.
Er zijn vijf onderdelen te onderscheiden: een pulsdetector, een pulsteller, een register voor het aantal getelde pulsen, een teller voor de meettijd en een deel dat het resultaat op het display afzet.

signalen `new_puls` en `prev_puls` staan beide achter de klokflank van regel 73. Voor beide signalen wordt daarom een flipflop gebruikt. De conditionele signaaltoewijzing van regel 79 beschrijft het combinatorische blok dat uit de signalen `new_puls` en `prev_puls` detecteert dat er een nieuwe puls is.

Code 6.5 : De pulsdetector uit het datapad van de elektronische personenweegschaal.

```
68  pulsdetector : process (clk, rst_n) is
69  begin
70    if rst_n = '0' then
71      new_puls  <= '0';
72      prev_puls <= '0';
73    elsif rising_edge(clk) then
74      new_puls  <= puls;
75      prev_puls <= new_puls;
76    end if;
77  end process pulsdetector;
78
79  puls_detect <= '1' when (new_puls = '1') and (prev_puls = '0') else '0';
```

In code 6.6 en code 6.7 staan de VHDL-beschrijvingen van het register `pc_reg` en de teller `cc`. De teller en de verschillende dataregisters krijgen allemaal een eigen proces. Alleen als de voorwaarde, waarop de signalen een andere waarde krijgen, hetzelfde is, is het handig om de signalen in één proces te plaatsen. De flipfloppen

new_puls en prev_puls krijgen allebei iedere klokslag een nieuwe waarde. Daarom zijn deze in één proces geplaatst.
De teller wordt nul gemaakt als signaal clr hoog is en het signaal pc_reg verandert alleen als load hoog is. Omdat deze voorwaarden verschillend zijn, krijgen deze signalen een eigen sequentieel proces. Signaal max_cnt wordt hoog als het aantal getelde klokpulsen groter is dan tienduizend.

Code 6.6 : Het dataregister pc_reg uit het datapad.

```
106  pc_register : process (clk,rst_n)  is
107  begin
108    if rst_n = '0' then
109      pc_reg <= (others =>'0');
110    elsif rising_edge(clk) then
111      if load = '1' then
112        pc_reg <= std_logic_vector(pc);
113      end if;
114    end if;
115  end process pc_register;
```

Code 6.7 : De teller voor de klokslagen.

```
117  clockcounter : process (clk, rst_n) is
118  begin
119    if rst_n = '0' then
120      cc <= (others => '0');
121    elsif rising_edge(clk) then
122      if clr = '1' then
123        cc <= (others => '0');
124      else
125        cc <= cc + 1;
126      end if;
127    end if;
128  end process clockcounter;
129
130  max_cnt <= '1' when cc > MAX  else '0';
```

De BCD-teller en de uitgangsregisters hebben verschillende stuursignalen en zijn dus weer een aparte processen, die respectievelijk in code 6.8 en 6.9 staan. Het algoritme voor het incrementeren van een BCD-getal is te complex voor deze uitleg van de implementatie van het model met gescheiden dataverwerking en besturing. In paragraaf 8.3 wordt dit algoritme besproken en in code 8.16 staat het ontbrekende deel van het proces pulscounter. In plaats van een eigen BCD-teller te gebruiken, kan de ontwerper ook een BCD-teller uit de bibliotheek van de ontwikkelomgeving toepassen door de betreffende component in de code aan te roepen.

Code 6.8 : De 4-digit BCD-teller.

```
81   pulscounter : process (clk,rst_n)  is
82     variable c : std_logic;
83   begin
84     if rst_n = '0' then
85       pc <= (others =>'0');
86     elsif rising_edge(clk) then
87       if clr = '1' then
88         pc <= (others => '0');
89       elsif puls_detect = '1' then
           ⋮ het algoritme voor
           ⋮ met één ophogen van
           ⋮ BCD-gecodeerde signaal pc
102      end if;
103    end if;
104  end process pulscounter;
```

Code 6.9 : Het uitgangsregister van het datapad.

```
132  outputregisters : process (clk, rst_n) is
133  begin
134    if rst_n = '0' then
135      dig_dec  <= (others => '0');
136      dig_unit <= (others => '0');
137      dig_ten  <= (others => '0');
138      dig_hund <= (others => '0');
139    elsif rising_edge(clk) then
140      if dsp = '1' then
141        dig_dec  <= bcd7seg(pc_reg(3 downto 0));
142        dig_unit <= bcd7seg(pc_reg(7 downto 4));
143        dig_ten  <= bcd7seg(pc_reg(11 downto 8));
144        dig_hund <= bcd7seg(pc_reg(15 downto 12));
145      end if;
146    end if;
147  end process outputregisters;
```

Voor de conversie van de vier BCD-gecodeerde signalen naar de vier signalen, die

de 7-segmentsdisplays aansturen, wordt in het proces `outputregisters` een functie `bcd7seg` gebruikt. Deze functie staat in code 6.10.

In het boek van van Dijken wordt niet altijd het gemeten resultaat afgebeeld.

Als het gewicht kleiner is dan 10 kg wordt er niets afgebeeld. De weegschaal lijkt dan uit. Als het gewicht groter is dan 150 kg toont de weegschaal de tekst `err` en als het gewicht kleiner is dan 100 kg wordt het honderdtal niet weergegeven.

Om de letters `e`, `r` en een leeg display weer te geven zijn deze tekens toegevoegd aan `bcd7seg`.

Het algoritme voor deze correcties kan in proces `pc_reg` plaats vinden. In code 6.6 kan regel 112 door deze pseudocode vervangen worden:

```
if pc >= 150 then
 pc_reg <= errE;
elsif pc < 10 then
 pc_reg <= EEEE;
else
 pc_reg <= pc;
 if pc < 100 then
  pc_reg(15 downto 12)<= E;
 end if;
end if;
```

Deze functionaliteit van de weegschaal is voor de uitleg van de ontwerpmethodieken niet heel relevant en is daarom weggelaten.

Wel is het interessant dat het extra register `pc_reg` hier voor gebruikt wordt. Zonder dit register, zou er in het datapad op deze plaats een combinatorisch blok geplaatst moeten worden met dit algoritme.

Code 6.10 : De functie `bcd7seg`.

```
43   function bcd7seg (bcd : in std_logic_vector(3 downto 0))
44                                    return std_logic_vector is
45   begin
46     case bcd is
47       -- digits
48       when "0000" => return "1111110";
49       when "0001" => return "0110000";
50       when "0010" => return "1101101";
51       when "0011" => return "1111001";
52       when "0100" => return "0110011";
53       when "0101" => return "1011011";
54       when "0110" => return "1011111";
55       when "0111" => return "1110000";
56       when "1000" => return "1111111";
57       when "1001" => return "1111011";
58       -- characters
59       when "1100" => return "1101111";  -- e
60       when "1101" => return "1000110";  -- r
61       when "1111" => return "0000000";  -- empty
62       when others => return "0000000";
63     end case;
64   end function bcd7seg;
```

Conclusie over ontwerp en implementatie dataverwerking

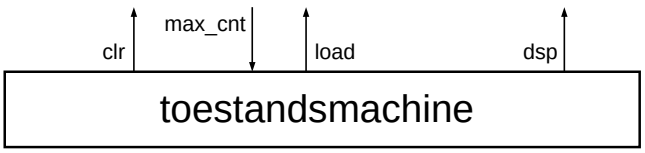
Het ontwerp van het dataverwerkingsdeel van een digitaal systeem bestaat uit eenvoudige digitale componenten. Een schets of tekening van het dataverwerkingsdeel is snel gemaakt en is goed bespreekbaar binnen een ontwerpteam. Belangrijke ontwerpbeslissingen kunnen al in een vroeg stadium van het ontwerptraject gemaakt worden.

De tekening kan één op één naar een VHDL-beschrijving vertaald worden. Het levert een beschrijving op die uit een groot aantal, relatief eenvoudige processen bestaat. Processen van lastig te beschrijven onderdelen kunnen eenvoudig worden vervangen door een aanroep van een bibliotheekcomponent met de gewenste functionaliteit.

Tijdens het hele ontwerp van het dataverwerkingsdeel is alleen vastgelegd welke functies nodig zijn en hoe deze functies met elkaar verbonden zijn. Nergens is ter sprake gekomen wanneer de tellers nul gemaakt moeten worden of wanneer register `pc_reg` zijn nieuwe waarde krijgt. Er is gefocust op de datastructuur en de bewerkingen. De toestandsmachine levert de signalen waarmee de registers en tellers worden aangestuurd. De toestandsmachine bepaalt de volgorde waarin dat gebeurt en regelt daarmee hoe de gegevens verwerkt worden.

6.6 Ontwerp van de toestandsmachine

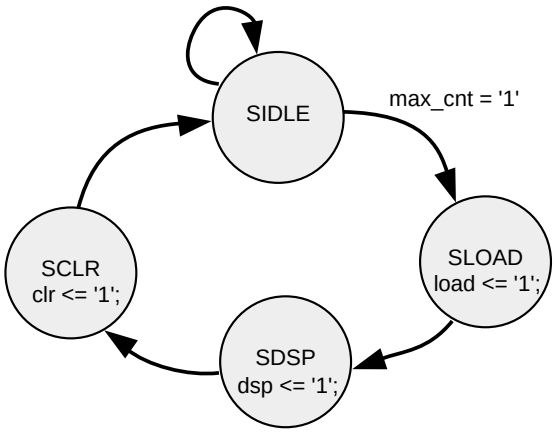
In het voorbeeld van de elektronische personenweegschaal is de toestandsmachine autonoom. Er zijn geen externe besturingssignalen. Figuur 6.11 toont een detail van figuur 6.10 met de toestandsmachine.



Figuur 6.11 : De in- en uitgangssignalen van de toestandsmachine.

De toestandsmachine heeft slechts één ingangssignaal: het statussignaal `max_cnt` waarmee de dataverwerking aangeeft dat de meettijd verstreken is. De toestandsmachine genereert de drie stuursignalen `load`, `dsp`, en `clr`.

In figuur 6.12 staat het toestandsdiagram voor de elektronische personenweegschaal. Er zijn vier toestanden. Samen met de tekening van het datapad uit figuur 6.10 legt het toestandsdiagram uit figuur 6.12 het gedrag van de elektronische personenweegschaal volledig vast.



Figuur 6.12 : Het toestandsdiagram voor de elektronische personenweegschaal.

In de toestand `SIDLE` worden de pulsen van het signaal `puls` geteld. Nadat de meettijd verstreken is, als `max_cnt = '1'`, wordt in toestand `SLOAD` signaal `load` hoog gemaakt. Het aantal getelde pulsen wordt in register `pc_reg` gezet. Eén klokslag later wordt `dsp` hoog en krijgen de uitgangsregisters de nieuwe waarden. Tenslotte wordt in toestand `SCLR` signaal `clr` hoog gemaakt en worden de tellers `cc` en `pc` nul. Dit maakt de huidige meettijd en het aantal getelde pulsen nul.

In code 6.11 staat de VHDL-beschrijving van het toestandsdiagram. Bij het coderen is gebruikt gemaakt van het model voor een toestandsmachine met drie processen uit paragraaf 5.7. Er zijn aparte processen voor het toestandsregister en voor de toestandsdecoder. De uitgangsdecoder bestaat uit drie conditionele signaaltoewijzingen.

Code 6.11 : De toestandsmachine voor de elektronische personenweegschaal.

```
27    type state_type is (SIDLE, SLOAD, SDSP, SCLR);
28    signal pres_state, next_state : state_type;
```

```
149   state_reg : process (clk,rst_n) is
150   begin
151     if rst_n = '0' then
152       pres_state  <= SIDLE;
153     elsif rising_edge(clk) then
154       pres_state  <= next_state;
155     end if;
156   end process state_reg;
157
158   next_state_decoder: process (pres_state, max_cnt) is
159   begin
160     case pres_state is
161     when SIDLE =>
162       if max_cnt = '1' then
163         next_state <= SLOAD;
164       else
165         next_state <= SIDLE;
166       end if;
167     when SLOAD =>
168       next_state <= SDSP;
169     when SDSP =>
170       next_state <= SCLR;
171     when SCLR =>
172       next_state <= SIDLE;
173     end case;
174   end process next_state_decoder;
175
176   load <= '1' when pres_state = SLOAD else '0';
177   dsp  <= '1' when pres_state = SDSP  else '0';
178   clr  <= '1' when pres_state = SCLR  else '0';
```

De code van de dataverwerking en de besturing staan in één architectuur en die hoort bij de entity van de personenweegschaal.

Het is niet handig om een aparte entity voor de dataverwerking en een aparte entity voor de besturing te maken. Het testen van het dataverwerkingsdeel zonder besturing is lastig omdat er dan een testbench nodig is die alle besturingssignalen genereert.

De testbench voor een complete personenweegschaal is relatief eenvoudig. Naast het kloksignaal en de reset is er alleen een signaal `puls` nodig.

6.7 Statussignalen en booleans

De manier waarop de status van het dataverwerkingsdeel bij de methode met gescheiden dataverwerking en besturing wordt vastgelegd, vinden sommige ontwerpers onhandig. Deze paragraaf geeft hiervoor een aantal alternatieven.

Bij de methode met gescheiden dataverwerking en besturing zijn alle statussignalen van het type `std_logic`. De status van de teller uit code 6.7 wordt op regel 130 met de conditionele signaaltoewijzing doorgegeven aan de toestandsmachine:

```
130   max_cnt <= '1' when cc > MAX  else '0';
```

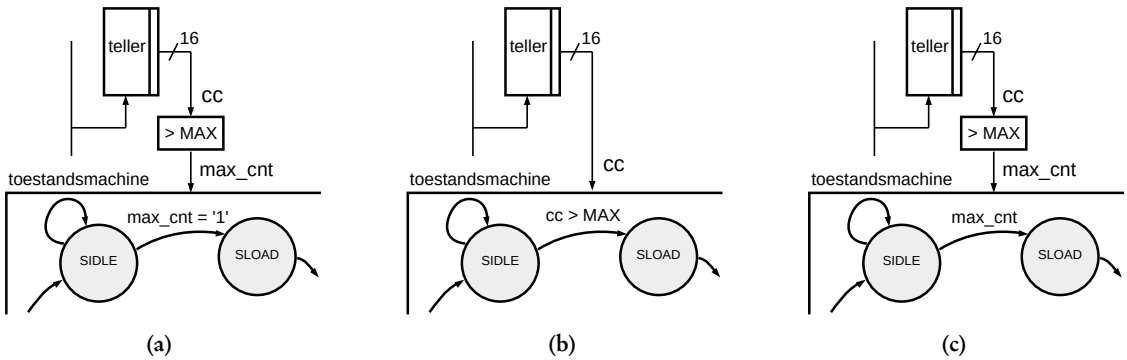
Vanaf VHDL-2008 is er een andere notatie met de operator `??` mogelijk:

```
if ?? max_cnt then
```

Meer informatie over de nieuwe operator `??` staat in bijlage C.6.

In figuur 6.13a staat een detail uit figuur 6.10 met de teller en dit statussignaal. De bijbehorende conditie bij de overgang van toestand `SIDLE` naar toestand `SLOAD` uit de beschrijving van de toestandsmachine van code 6.11 is dan:

```
162       if max_cnt = '1' then
```



Figuur 6.13 : Drie oplossingen voor een statussignaal: voorbeeld a gebruikt een 1-bits signaal `max_cnt`, in voorbeeld b gebruikt de toestandsmachine het 16-bits signaal `cc` en bij voorbeeld c is het signaal van het type `boolean`.

In figuur 6.13b gaat de waarde van de teller `cc` naar de toestandsmachine. De voorwaarde bij de toestandsovergang is dan gelijk aan:

```
162     if cc > MAX then
```

Een andere aanpak is om een signaal van het type `boolean` te gebruiken:

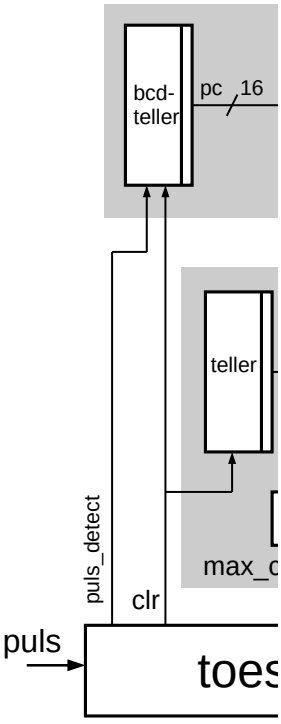
```
        signal max_cnt : boolean;
             ⋮
        max_cnt <= cc > MAX;
```

De waarde van `max_cnt` is afhankelijk van de waarde die signaal `cc` heeft gelijk aan `false` of `true`. De voorwaarde bij de toestandsovergang is dan:

```
162     if max_cnt then
```

Figuur 6.13c toont deze aanpak.

Ondanks het feit dat er geen principieel verschil is tussen een `std_logic` en een `boolean` en de toewijzing aan `max_cnt` en de overgangsvoorwaarde beknopter is, levert het gebruik van booleans geen beter leesbare code op. Integendeel, het is een voordeel als alle signalen `std_logic`, of een daarvan afgeleide type als `signed`, `unsigned` of `std_logic_vector` zijn.



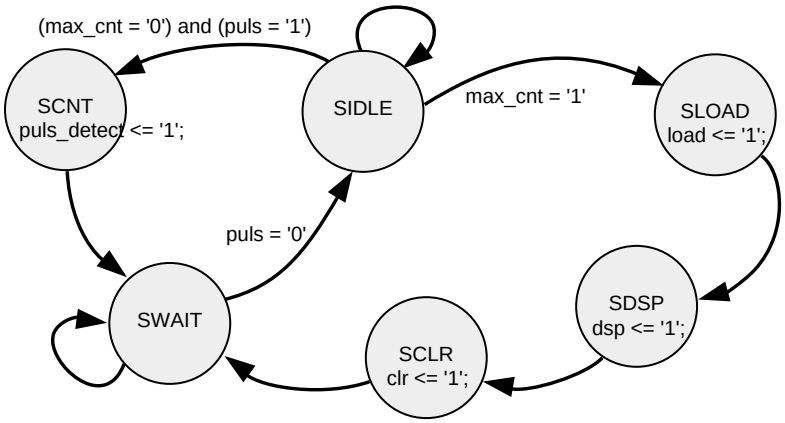
Figuur 6.14 : Detail van alternatief datapad.
Signaal `puls` is een ingang voor de toestandsmachine. Signaal `puls_detect` is hoog als er een nieuwe puls is.

6.8 Alternatieve dataverwerking en besturing

Bij het tekenen van het datapad is besloten het signaal `puls` als een datasignaal te beschouwen. Dat hoeft niet per se. Dit signaal kan ook een ingang van het besturingsdeel zijn. In dat geval moet uit figuur 6.10 het deel met de pulsdetector verwijderd worden. Het signaal `puls_detect` komt dan uit de toestandsmachine, zoals dat in figuur 6.14 is getekend.

Het toestandsdiagram voor deze alternatieve versie staat in figuur 6.15. De toestandsmachine kent nu twee scenario's. Ten eerste moeten de pulsen geteld worden en ten tweede moet het resultaat op het display gezet worden als de meettijd verstreken is. In toestand `SIDLE` doet het systeem niets. Iedere keer als signaal `puls` hoog en de meettijd nog niet verstreken is, gaat de toestandsmachine naar

toestand SCNT. In deze toestand wordt signaal puls_detect hoog. Bij de volgende klokslag komt de machine in toestand SWAIT en wacht totdat signaal puls laag is.



Figuur 6.15 : Het toestandsdiagram voor de alternatieve beschrijving. Signaal puls is hier een ingang van de toestandsmachine.

Nadat de meettijd verstreken is, worden achtereenvolgens de signalen load, dsp en clr hoog. Ook bij dit scenario komt de machine in toestand SWAIT. Dit zorgt ervoor dat een puls nooit meerdere keren geteld wordt.

Als de besturing van een digitaal systeem meer taken doet, wordt de toestandsmachine vaak complexer en lastiger te overzien. Bij de alternatieve beschrijving moeten nu twee dingen tegelijkertijd worden gedaan: het tellen van de pulsen en het afbeelden van de pulsen. Bij de oorspronkelijke beschrijving kent de toestandsmachine slechts één scenario. Het datapad van de oorspronkelijk beschrijving is complexer. Het omvat ook de pulsdetectie, wel zijn in het dataverwerkingsdeel de verschillende functies eenvoudiger te onderscheiden.

6.9 De FSMD-methode

Het is mogelijk om meer functionaliteiten van het dataverwerkingsdeel naar het besturingsdeel te verplaatsen. Zelfs alle functies uit het datapad kunnen in de beschrijving van de toestandsmachine opgenomen worden. Men spreekt dan van een toestandsmachine met dataverwerking oftewel een FSMD, een *finite state machine with a data path*.
In het algemeen geeft een FSMD een meer compacte beschrijving met minder processen dan een ontwerp met een gescheiden dataverwerking en besturing. Een ander voordeel is dat de simulatietijd korter zal zijn. De nadelen van een FSMD zijn dat dit ontwerp minder flexibel is, dat het moeilijker is typisch hardware oplossingen te gebruiken, dat er gemakkelijk foute VHDL-constructies worden toegepast, dat het een slecht leesbare code geeft en dat er geen goede grafische mogelijkheden zijn.

In code 6.12 staat een FSMD-beschrijving van de elektronische personenweegschaal. De opzet is identiek aan de toestandsmachine van de alternatieve beschrijving uit figuur 6.15. De signalen cc, pc, pc_reg en alle uitgangssignalen zijn aan het

Code 6.12 : FSMD-beschrijving van de elektronische personenweegschaal.

```
27    type state_type is (SIDLE, SCNT, SWAIT, SLOAD, SDSP, SCLR);
28    signal state : state_type;
```

```
89    fsmd : process (clk,rst_n) is
90    begin
91      if rst_n = '0' then
92        state     <= SIDLE;
93        cc        <= (others => '0');
94        pc        <= (others => '0');
95        pc_reg    <= (others => '0');
96        dig_hund  <= (others => '0');
97        dig_ten   <= (others => '0');
98        dig_unit  <= (others => '0');
99        dig_dec   <= (others => '0');
100     elsif rising_edge(clk) then
101       cc <= cc + 1;
102       case state is
103       when SIDLE =>
104         if cc > MAX then
105           state <= SLOAD;
106         elsif puls = '1' then
107           state <= SCNT;
108         else
109           state <= SIDLE;
110         end if;
111       when SCNT =>
112         pc <= incrementBCD(pc);
113         state <= SWAIT;
114       when SWAIT =>
115         if puls = '0' then
116           state <= SIDLE;
117         else
118           state <= SWAIT;
119         end if;
120       when SLOAD =>
121         pc_reg <= pc;
122         state <= SDSP;
123       when SDSP =>
124         dig_dec  <= bcd7seg(pc_reg(3 downto 0));
125         dig_unit <= bcd7seg(pc_reg(7 downto 4));
126         dig_ten  <= bcd7seg(pc_reg(11 downto 8));
127         dig_hund <= bcd7seg(pc_reg(15 downto 12));
128         state <= SCLR;
129       when SCLR =>
130         pc <= (others => '0');
131         cc <= (others => '0');
132         state <= SWAIT;
133       end case;
134     end if;
135   end process fsmd;
```

proces fsmd toegevoegd. Omdat deze signalen achter de klokflank staan worden bij synthese hiervoor registers gebruikt.
De functie bcd7seg is identiek aan die uit code 6.10. De functie incrementBCD verhoogt het BCD-gecodeerde signaal pc met één. Deze functie is net als de beschrijving van de BCD-teller uit code 6.8 behoorlijk complex. Het schrijven van de functie is iets lastiger dan de BCD-teller en wordt ook in paragraaf 8.4 besproken. Bij het ontwerp met een gescheiden dataverwerking en besturing ligt het voor de hand om in plaats van een eigen BCD-teller te schrijven, een component uit de bibliotheek te gebruiken. Bij de FSMD-methode ziet de ontwerper niet zo snel dat hier een standaard BCD-teller gebruikt kan worden.

Een nadeel van het FSMD-model is dat er eenvoudig fouten in de VHDL ontstaan. In figuur 6.16 staan twee verschillende versies van code 6.12. In de linker versie wordt signaal cc opgehoogd en daarna volgt het case-statement. In de rechter versie staat eerst het case-statement en daarna wordt het signaal cc opgehoogd.

```
  elsif rising_edge(clk) then
    cc <= cc + 1;
    case state is
    when SIDLE =>
          ⋮
    when SCLR =>
      pc <= (others => '0');
      cc <= (others => '0');
      state <= SWAIT;
    end case;

  end if;
end process fsmd;
```

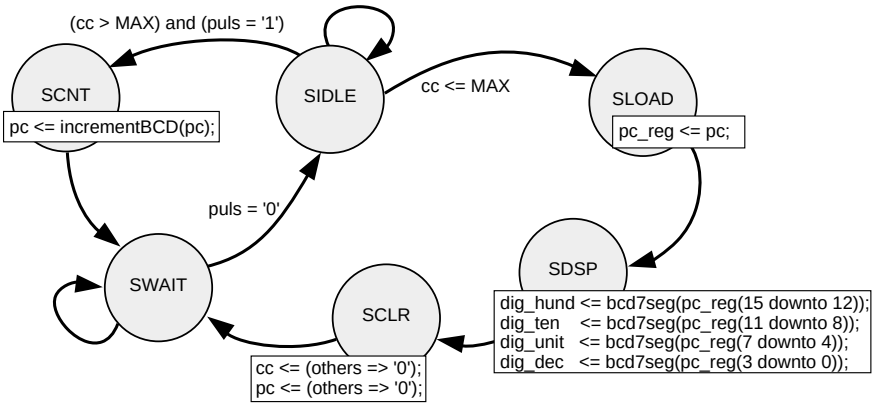
```
  elsif rising_edge(clk) then

    case state is
    when SIDLE =>
          ⋮
    when SCLR =>
      pc <= (others => '0');
      cc <= (others => '0');
      state <= SWAIT;
    end case;
    cc <= cc + 1;
  end if;
end process fsmd;
```

Figuur 6.16 : **Mogelijke onduidelijkheden bij FSMD-model.** Links staat de beschrijving van code 6.12. Het signaal cc wordt voor het case-statement opgehoogd. Rechts staat een alternatief, waarbij het ophogen na het case-statement komt. Hierdoor wordt signaal cc nooit nul gemaakt.

Bij de rechter beschrijving heeft het nul maken van signaal cc in toestand SCLR geen effect. De nieuwe waarde voor het signaal cc wordt in toestand SCLR nul gemaakt. Omdat dit een signaaltoewijzing is, wordt deze waarde niet direct toegekend. Signaal cc heeft nog de oude waarde. Deze waarde wordt gebruikt bij het ophogen. De nieuwe waarde voor signaal cc is de oude waarde van cc, die met één is opgehoogd. Na de evaluatie van alle processen krijgt signaal cc die waarde en dus niet 1 wat een onervaren VHDL-ontwerper misschien verwacht.
Bij de linker versie wordt signaal cc eerst opgehoogd. De nieuwe waarde voor het signaal cc is dan de huidige waarde plus één. In toestand SCLR wordt deze nieuwe waarde overruled met nul. Na de evaluatie van alle processen krijgt signaal cc dan de waarde nul.

Om deze problemen te omzeilen, gebruikt men in plaats van signalen variabelen. Er is niets tegen het gebruik van variabelen. Integendeel, bij het simuleren is voor variabelen geen interne datastructuur nodig en zullen de simulaties sneller zijn. Het nadeel van variabelen is dat het effect bij synthese niet eenduidig is. Signalen achter een klokflank leveren altijd een register op. Variabelen achter

een klokflank kunnen een register opleveren. Vaak leidt dit tot een ingewikkelde beschrijving met variabelen en signalen en extra toestanden om het tijdsgedrag in orde te krijgen.



Figuur 6.17 : Het FSMD-model voor de elektronische personenweegschaal.

In figuur 6.17 staat een tekening van het FSMD-model voor de elektronische personenweegschaal. De dataverwerkingsaspecten staan als signaaltoewijzingen bij de betreffende toestanden. Bij complexere systemen kan dit zeer onoverzichtelijk worden.

6.10 De ASMD-methode

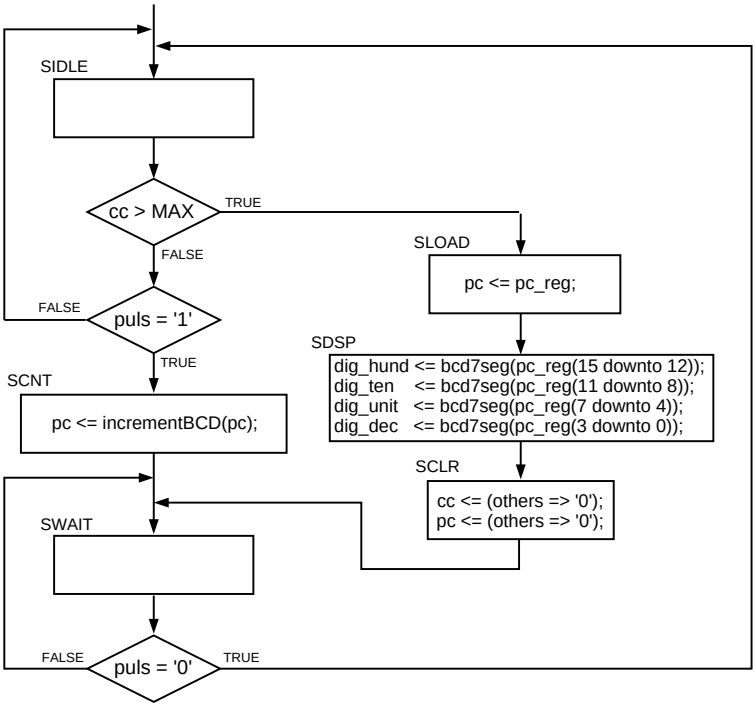
Het FSMD-model kan ook als ASM-chart worden getekend. In plaats van een toestandsdiagram kan de ontwerper ook een ASM-chart met dataverwerking tekenen oftewel een ASMD-chart. De FSMD-methode zou men evengoed de ASMD-methode kunnen noemen.

Voor deze ASMD-methode gelden dezelfde voor- en nadelen als voor de FSMD-methode, alleen levert dit een tekening op die beter leesbaar is. In figuur 6.18 staat de ASMD-chart voor de elektronische personenweegschaal.

Veel ontwikkelomgevingen voor FPGA's bevatten grafische programma's waarmee toestandsdiagrammen en ASM-charts getekend kunnen worden. Aan deze tekeningen kunnen meestal ook dataverwerkingsaspecten worden toegevoegd. Een beginnend VHDL-ontwerper is bij het gebruik van deze programma's snel geneigd alles met het toestandsdiagram of met het ASM-chart op te lossen. Men vergeet dan om een BCD-teller voor het ophogen van de te tellen pulsen te gebruiken.

6.11 De twee-processenmethode

VHDL voert de processen en de parallelle signaaltoewijzingen niet uit in de volgorde waarin deze opgeschreven zijn. De volgorde wordt bepaald door de gebeurtenissen. Als een signaal verandert triggert dat andere signaaltoewijzingen of processen. Deze wijze van uitvoeren staat dicht bij de hardware, maar maakt de interpretatie bij veel processen en toewijzingen moeilijk.

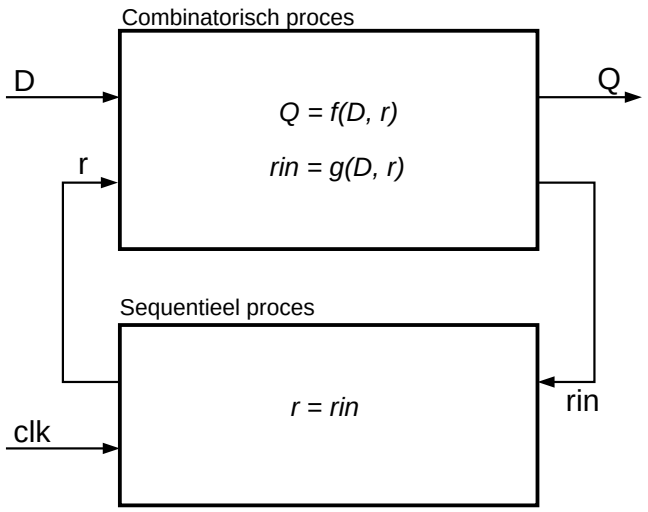


Figuur 6.18 : De ASMD-chart voor de elektronische personenweegschaal.

Jiri Gaisler van Gaisler Research AB was eerder werkzaam bij het ESA, European Space Agency. Als systeemontwerper heeft hij daar gewerkt aan de ontwikkeling van fouttolerante microprocessoren voor de ruimtevaart. In het document "Fault-tolerant Microprocessors for Space Applications- uit 2008 "stelt hij in hoofdstuk 5, "A structured VHDL design method- de twee-processenmethode voor.

Om de eenduidigheid en de leesbaarheid van de VHDL te verbeteren, heeft Jiri Gaisler de twee-processenmethode voorgesteld. Iedere entity heeft bij dit model slechts twee processen: een combinatorisch proces dat alle combinatorische functies bevat en een sequentieel proces dat alle registers beschrijft. Het combinatorische proces bestaat uit variabele- en signaaltoewijzingen die op een sequentiële manier doorlopen worden.

Voor de lucht- en ruimtevaart is het maken van soft- en hardware zonder fouten essentieel. Er worden in dat vakgebied extreem hoge eisen gesteld aan de betrouwbaarheid en aan de fouttolerantie. Het is dus niet vreemd dat er vanuit dit vakgebied veel aandacht is voor methodieken die tot minder ontwerpfouten leiden.



Figuur 6.19 : Het algemene blokschema voor de twee-processenmethode.

Verwar de twee-processenmethode niet met het twee-processenmodel voor een toestandsmachine uit figuur 5.32.
De twee-processenmethode stemt wel overeen met het model uit figuur 5.36.

Figuur 6.19 toont het blokschema voor deze aanpak. De ingangen `D` van de entity zijn verbonden met het combinatorische proces. De ingangen van het sequentiële

proces zijn de interne signalen `rin` die van het combinatorische proces komen. Het sequentiële proces kopieert bij de actieve klokflank deze interne signalen naar de interne signalen `r`, die weer ingangen zijn van het combinatorische proces. De uitgangssignalen van de entity zijn uitgangssignalen van het combinatorische proces. Twee vergelijkingen leggen de functionaliteit van het combinatorische proces vast:

$$\texttt{Q} = f(\texttt{D}, \texttt{r}) \tag{6.7}$$

$$\texttt{rin} = g(\texttt{D}, \texttt{r}) \tag{6.8}$$

Op pagina 156 staat in code 6.14 het voorbeeld van een 8-bits teller dat Jiri Gaisler bij zijn voorstel gebruikt. Het combinatorische proces uit deze beschrijving heeft twee uitgangen, namelijk de signaaltoewijzing aan `rin` op regel 28 en de signaaltoewijzing aan `q` op regel 29. Signaal `q` hangt alleen van signaal `r` af. De waarde van `rin` hangt af van de signalen `load`, `count`, `d` en `r`. Om problemen met meervoudige signaaltoewijzingen te voorkomen, is er een hulpvariabele `tmp` gebruikt.

Code 6.13 geeft de beschrijving van de elektronische personenweegschaal met behulp van de twee-processenmethode. Het proces `sequential` beschrijft in feite een hele verzameling dataregisters. Bij de actieve klokflank worden de nieuw berekende waarden in deze registers gezet.
Alle ingangssignalen van het proces `combinational` worden bij het begin van het proces, op regel 103 tot en met 109, aan variabelen toegekend. Afhankelijk van de toestand waarin het systeem zich bevindt, past het case-statement op regel 110 tot en met 141 deze variabelen aan. Aan het eind van het proces worden de variabelen aan de uitgangssignalen van dit proces toegekend. Op deze manier staan alle veranderingen sequentieel achter elkaar. Er zijn in het process geen afhankelijke meervoudige signaaltoewijzingen en bovendien is de combinatoriek gescheiden van de registers. Proces `combinational` is een proces dat sequentieel doorlopen wordt.

6.12 Keuze methodiek

Formeel lijkt het twee-processenmethode de juiste strategie voor het beschrijven van digitale systemen, alleen voor hardware-ontwerpers is deze aanpak nogal geforceerd. Verschillende functionaliteiten van de specifieke onderdelen zijn nu verdeeld over twee processen. Van de pulsteller staat de eigenschap dat het een register is in proces `sequential` en staan het ophogen op regel 120 en het nul maken op regel 138 van proces `combinational`. Dit kan voor hardware-ontwerpers zeer verwarrend zijn.

De FSMD- en de ASMD-methode geven een compacte beschrijving, maar lenen zich minder goed voor typische hardware oplossingen. Bovendien worden er gemakkelijk fouten in de beschrijvingen gemaakt. De sequentiële en parallelle aspecten van VHDL komen op een onduidelijke manier bij elkaar te staan.

De methode met een gescheiden dataverwerking en besturing levert een relatief grote beschrijving met veel processen op. Het voordeel van deze methode is dat met twee tekeningen — de schets van de dataverwerking en het toestandsdiagram — het complete ontwerp vastligt. Bovendien zijn deze tekeningen één op één over te zetten naar synthetiseerbare VHDL. Voor beginnende VHDL-ontwerpers, mits ze de basis kennen van de digitale techniek, geeft deze methode

```
91 combinational : process (puls, curr_state,
92                          curr_cc, curr_pc, curr_pc_reg,
93                          curr_dig_hund, curr_dig_ten,
94                          curr_dig_unit, curr_dig_dec) is
95   variable v_cc       : unsigned(15 downto 0);
96   variable v_pc       : std_logic_vector(15 downto 0);
97   variable v_pc_reg   : std_logic_vector(15 downto 0);
98   variable v_dig_dec  : std_logic_vector(6 downto 0);
99   variable v_dig_unit : std_logic_vector(6 downto 0);
100  variable v_dig_ten  : std_logic_vector(6 downto 0);
101  variable v_dig_hund : std_logic_vector(6 downto 0);
102 begin
103   v_cc       := curr_cc + 1;
104   v_pc       := curr_pc;
105   v_pc_reg   := curr_pc_reg;
106   v_dig_dec  := curr_dig_dec;
107   v_dig_unit := curr_dig_unit;
108   v_dig_ten  := curr_dig_ten;
109   v_dig_hund := curr_dig_hund;
110   case curr_state is
111   when SIDLE =>
112     if curr_cc > MAX then
113       next_state <= SLOAD;
114     elsif puls = '1' then
115       next_state <= SCNT;
116     else
117       next_state <= SIDLE;
118     end if;
119   when SCNT =>
120     v_pc    := incrementBCD(curr_pc);
121     next_state <= SWAIT;
122   when SWAIT =>
123     if puls = '0' then
124       next_state <= SIDLE;
125     else
126       next_state <= SWAIT;
127     end if;
128   when SLOAD =>
129     v_pc_reg := curr_pc;
130     next_state  <= SDSP;
131   when SDSP =>
132     next_dig_dec  <= bcd7seg(curr_pc_reg(3 downto 0));
133     next_dig_unit <= bcd7seg(curr_pc_reg(7 downto 4));
134     next_dig_ten  <= bcd7seg(curr_pc_reg(11 downto 8));
135     next_dig_hund <= bcd7seg(curr_pc_reg(15 downto 12));
136     next_state    <= SCLR;
137   when SCLR =>
138     v_pc    := (others => '0');
139     v_cc    := (others => '0');
140     next_state <= SWAIT;
141   end case;
142   next_cc       <= v_cc;
143   next_pc       <= v_pc;
144   next_pc_reg   <= v_pc_reg;
145   next_dig_hund <= v_dig_hund;
146   next_dig_ten  <= v_dig_ten;
147   next_dig_unit <= v_dig_unit;
148   next_dig_dec  <= v_dig_dec;
149 end process combinational;
```

Code 6.13 : De twee-processenmethode voor de elektronische personenweegschaal. Links staat het combinatorische proces dat uit de huidige waarden de nieuwe waarden berekent. Hieronder staat het sequentiële proces dat de nieuwe waarden toekent aan de huidige waarden.

```
151 sequential : process (clk,rst_n) is
152 begin
153   if rst_n = '0' then
154     curr_state     <= SIDLE;
155     curr_cc        <= (others => '0');
156     curr_pc        <= (others => '0');
157     curr_pc_reg    <= (others => '0');
158     curr_dig_hund  <= (others => '0');
159     curr_dig_ten   <= (others => '0');
160     curr_dig_unit  <= (others => '0');
161     curr_dig_dec   <= (others => '0');
162   elsif rising_edge(clk) then
163     curr_state     <= next_state;
164     curr_cc        <= next_cc;
165     curr_pc        <= next_pc;
166     curr_pc_reg    <= next_pc_reg;
167     curr_dig_hund  <= next_dig_hund;
168     curr_dig_ten   <= next_dig_ten;
169     curr_dig_unit  <= next_dig_unit;
170     curr_dig_dec   <= next_dig_dec;
171   end if;
172 end process sequential;
173
174 dig_hund      <= curr_dig_hund;
175 dig_ten       <= curr_dig_ten;
176 dig_unit      <= curr_dig_unit;
177 dig_dec       <= curr_dig_dec;
```

Code 6.14 : Een 8-bits teller volgens de twee-processenmethode.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity count8 is
6    port (
7      clk   : in  std_logic;
8      load  : in  std_logic;
9      count : in  std_logic;
10     d     : in  std_logic_vector(7 downto 0);
11     q     : out std_logic_vector(7 downto 0));
12 end entity count8;
13
14 architecture twoproc of count8 is
15   signal r, rin : unsigned(7 downto 0);
16 begin
17
18   combinational : process (load, count, d, r) is
19     variable tmp : unsigned(7 downto 0);
20   begin
21     if load = '1' then
22       tmp := unsigned(d);
23     elsif count = '1' then
24       tmp := r + 1;
25     else
26       tmp := r;
27     end if;
28     rin <= tmp;
29     q   <= std_logic_vector(r);
30   end process combinational;
31
32   sequential : process (clk) is
33   begin
34     if rising_edge(clk) then
35       r <= rin;
36     end if;
37   end process sequential;
38
39 end architecture twoproc;
```

het beste resultaat. Het grootste voordeel van deze methode is dat in een vroeg stadium het ontwerp besproken kan worden aan de hand van tekeningen, zonder dat er al een VHDL-code is.

Het programma Ease van HDL Works is wel heel interessant voor de methode met een gescheiden dataverwerking en besturing.

Met de meeste grafische ontwerpprogramma's kunnen toestandsdiagrammen en blokschema's getekend worden. Mits de flexibiliteit groot genoeg is, zijn deze programma's bruikbaar bij de FSMD-methode. Voor de methode met een gescheiden dataverwerking en besturing zijn deze programma's meestal minder geschikt. Het tekenen van blokken, processen en verbindingen kost veel tijd. Bovendien geeft het toevoegen van de VHDL-code aan de verschillende processen meer problemen als het schrijven van code in één enkel VHDL-bestand.