KU Leuven

Ma Ingenieurswetenschappen:
Computerwetenschappen

Digitale Elektronica en Processoren

Lesnota's

Author:
Helena Brekalo

2015-2016

Contents

Chapter 1

Les 1

1.1 Slides: 1_Intro

Slide 6: Die meerdere cores zorgen voor een grotere verwerkingskracht. Voorbeeld van de smartphone: het blokschema is hier van belang. Je hebt een aantal dingen die contact maken met de buitenwereld. In het blauw: de telecommunicatie, het echte gsm-gebeuren. Je hebt ook ergens iets draadloos (WiFi) (bruin). Het is dus heel specifieke hardware die een bepaalde taak vervult, die is geoptimaliseerd om éé ding heel goedkoop, eenvoudig en compact te doen. Je hebt ook geheugen en powermanagement. De donkere blokjes in het midden zijn een algemene processor (rechts) die redelijk flexibel is en een specifieke processor (links) die alle signaalverwerking doet.

Slide 7: SoC: alles op 1 enkele chip. Heel wat verschillende dingen zijn weer aanwezig. 3 grote stukken: opgelijst met tweekleurige pijltjes:

- Programmeerbare processor: die kan vanalles en nog wat doen.
- Niet-programmeerbare processoren: specifieke processoren voor beeldverwerking bv. Ze zijn meestal beperkt programmeerbaar, om éé specifieke familie van problemen op te lossen.
- Controle-eenheden.

Waarom die tweede groep? → Veel goedkoper! Je hebt ongeveer 1000 keer zoveel energie nodig op een programmeerbare processor dan bij een niet-programmeerbare. Bv i.p.v. 1W 1kW.

Slide 8: We vertrekken van een doel: we hebben een algoritme of beschrijving op gedragsniveau. Dat is een combinatie van software en hardware. Sommige dingen doe je via software omwille van flexibiliteit, andere dingen doen we via hardware. Dat is allemaal afhankelijk van elkaar. De keuze van welk programma je gaat gebruiken zal impact zal hebben op de hardware en de hardware die je hebt zal bepalen hoe programmeerbaar het is. Men spreekt daarom van hardware-software codesign. hardware gaat ook interrageren met wat als algoritme gebruikt kan worden: sommige dingen zijn makkelijk te implementeren, andere moeilijker. Het kan zinvol zijn om de algoritmen eventueel wat aan te

passen.
Bij de hardware heb je een digitaal en een analoog gedeelte. Bij een smartphone: alle telecommunicatie is analoog: continue signalen. Wat in het midden stond was meer digitaal: meer berekeningen. Beiden moeten geïmplementeerd worden, gewoonlijk op dezelfde componenten, momenteel gebeurt dat met transistoren en dat moet vertaald worden naar een chip (geïntegreerde schakeling). Wij gaan het hier enkel hebben over het linkerdeel. Dit kan nog opgesplitst worden in 3 lagen. Poorten liggen het dichtst aan bij transistoren, die kunnen gebruikt worden om basiscomponenten te maken (bv. basisprocessoren) die samengezet kunnen worden om systemen te bouwen.

Slide 9: Het gaat in eerste instantie over het ontwerp van elektronische schakelingen. We gaan daar ook voorbeelden van zien. Hardware kan je beschrijven via een schema, maar ook via een taal: hardware-beschrijvingstalen. We gaan die ook bekijken. We gaan dit ook toepassen in oefeningen en labo's.

Slide 10: Drie grote blokken:

1. Basis: principes van digitaal ontwerp en hoe je dat doet met elektronica. Veel van de dingen die we gaan zien kan ook met andere dingen dan elektornica.

2. Dan gaan we zien hoe we daar schakelingen mee kunnen maken. Eerst zonder geheugen en dan met geheugen.

3. Dan gaan ingewikkeldere algoritmen implementeren en dan kijken hoe dat geïmplementeerd kan worden.

Slide 11: Vereiste voorkennis.

Examen: FSM is schriftelijk, rest is ook mondeling. Eerst wordt de theorie ondervraagd, dan de vertaling van een algoritme (vraag 1).

1.2 Slides: 2_Digitaal_Ontwerp

Slide 3: Documentatie is heel belangrijk: als handleiding en om bij te houden wat geprobeerd is (en wat dus wel en niet werkt).

Slide 4: Specificatie: beschrijving, heel dikwijls in natuurlijke taal, van welke berekeningen je wilt/hebt. Specificaties zijn in de meeste gevallen te algemeen en te onvolledig om er een duidelijke implementatie van te maken omdat het in veel gevallen te weinig specifieert. Er kunnen heel veel verschillende algoritmes gebruikt worden om een probleem op te lossen. Langs de andere kant gaat het soms al zaken vastleggen waarvan je niet de bedoeling had dat het vastgelegd werd (bv. vaste of vlottende komma).

Slide 5: Het geheel is dikwijls iteratief (vandaar de lus). Synthese: schakeling maken. Het is niet het bouwen van de schakeling maar een iets algemenere beschrijving maken. Normaal doe je geen synthese van een algemene beschrijving om onmiddelijk op een geïntegreerde schakeling uit te komen. Je gaat daarom in niveau's werken: in stapjes oplossen: eerst op het hoogste niveau kijken en een heel algemene beschrijving doen: ik wil een FFT implementeren, wat houdt dat in, wat voor verwerking heb ik nodig?
Vervolgens ga je een niveau lager: welke blokken heb je allemaal nodig? Die ga je verder verfijnen tot op een niveau waarop je blokken krijgt waarvan je ongeveer weet hoe die geïmplementeerd moeten worden. Je gaat dat dan nog verder opsplisten (in poorten en flip flops). Dan komt er nog een nivau onder: hoe maak ik een poort (met transistoren)?
Door dat in stukjes te doen en elk stukje per keer op te lossen is het doenbaar: je maak ter hapklare brokken van die je kan verwerken. Wij gaan de twee middenste niveau's vooral bekijken.

Slide 6: Belangrijk is het gebruik van bibliotheken: de kennis van anderen, dingen die je zelf niet meer moet doen. We gaan componenten hergebruiken. Als iemand iets nog niet gedaan heeft ga je het zelf maken, het in de bibliotheek steken zodat het hergebruikt kan worden. Die bibliotheken komen op ieder van de niveau's terug. Waarom zo belangrijk? → Wet van Moore.
We krijgen dus heel wat mogelijkheden jaarlijks erbij. Uiteindelijk zal dat wel gaan vastlopen. Wat ontstaat er nu? We krijgen elk jaar meer en meer mogelijkheden. We kunnen dit jaar een twee keer zo complexe schakeling maken dan 2 jaar geleden. Een meer complexe schakeling ontwerpen duurt meer dan twee keer zo lang want je raakt volledig het overzicht kwijt. De processoren aan de rechterkant teken je niet zomaar op 1 blad, dat duurt eindeloos lang. De enige manier om dat doenbaar te krijgen is door zoveel mogelijk te hergebruiken. Dit betekent dat het ontwerpen altijd achterloopt op de mogelijkheden die we hebben.

Slide 7: Blauw: wat we als verwerkingskracht kunnen doen, dit is wat ons interesseert. Meer cores op 1 chip: kost qua ontwerpinspanning niet zoveel werk. In de toekomst gaan we dus veel meer in parallel moeten laten gebeuren. Vermogen (Typical Power): is belangrijk want je kan dat niet eindeloos laten toenemen. Het vermogengebruik hangt af van de spanning in het kwadraat (V), de grootte van de chip (C) en de frequentie (f). Hoe hoger de frequentie, hoe meer verwerkingskracht je hebt. Oorspronkelijk zaten we aan minder dan een Watt voor een geïntegreerde schakeling zitten we nu aan 100 Watt.
We willen het vermogen dus constant houden maar dat heeft zijn impact op de frequentie en de verwerkingskracht.

Slide 8: Analyse: nagaan of het voldoet aan de specificaties. Dat doe je niet helemaal op het einde: het ontwerp gebeurt in stapjes en ga je telkens testen of dat het wel voldoet aan zijn specificaties op dat niveau. Dat betekent meer dan alleen maar testen of het doet wat het moet doen. Het functionele is eigenlijk maar een aspect ervan, er hoort veel meer bij: het vermogenverbruik bv. mag niet eindeloos oplopen. De snelheid waaraan dat kan werken, de

kostprijs,... Dat soort zaken moet ook nagekeken worden. De testbaarheid is ook een belangrijke factor.

Slide 10: Schakelingen gebaseerd op logische werking. We werken met discrete signalen en in de praktijk met bits: de waarden worden enkel 0 of 1.

Slide 11: Er is een verband tussen het al dan niet ingedrukt zijn van de knop en de uitgang. Je kan ook andere functies hebben: een complementaire die in rusttoestand een verbinding maakt: in rusttoestand is er een verbinding en brandt de lamp. Druk je de knop in, gaat de lamp uit. Er zijn heel wat verschillende manieren om een logische functie aan te duiden, afhankelijk van welk boek/welke beschrijving je neemt: onderaan: verschillende manieren om NOT te schrijven. Wij gebruiken normaal altijd een accent (').

Slide 12: Andere logische functies:

- AND: schakelaars in serie zetten en dan brandt de lamp alleen als beide knoppen ingedrukt zijn.
- OR: schakelaars in parallel zetten: de lamp brandt als een van de twee schakelaars ingedrukt is of ze alletwee ingedrukt zijn.

We kunnen complexere schakelaars maken: je kan een lamp bedienen vanop twee plaatsen, dat is ook een logische functie: XOR. De lamp zal branden als de ene of de andere knop ingedrukt is, maar niet als ze alletwee ingedrukt zijn of alletwee niet ingedrukt zijn. Voor andere logische functies kan je ze herleiden tot de twee eersten (AND en OR): XOR is een parallelschakeling van een serieschakeling: je hebt alleen maar een parallel- en een serieschakeling nodig en de inverter. We hebben dus maar 3 logische schakelingen nodig: het inverse (NOT), een AND en een OR, daarmee kunnen we alle logische functies bepalen.

Slide 13: Hoe beschrijven we de functionaliteit van zo'n logische schakeling? $\rightarrow$ Met een waarheidstabel: een opsomming van alle mogelijke combinaties aan de ingang en daarbij wordt aangegeven wat de uitgang is. Vermits een waarheidstabel eigenlijk de functionaliteit beschrijft zijn twee implementaties met eenzelfde waarheidstabel volledig equivalent, die doen exact hetzelfde. Je kan dus ook zeggen dat als je een waarheidstabel hebt die je op verschillende manieren kan maken, je aan de hand van die waarheidstabellen verschillende schakelingen kan maken. Het is dus niet zo dat bij een functionaliteit altijd maar 1 implementatie mogelijk is, er zijn meerdere mogelijkheden.

Slide 14: Logische poorten: meer dan de inverter, de AND en de OR-poort heb je in principe niet nodig, hoewel men dikwijls gebruik maakt van complexe poorten. In CMOS is het bv. goedkoper om met complexe poorten te werken dan met de basispoorten. Een logische schakeling is dan een combinatie van die poorten. Je kan dit met een schema doen of een programma.

Slide 15: Als je zo'n schema hebt kan je ook makkelijk de waarheidstabel opstellen. Je gaat gewoon op alle tussenliggende draden kijken wat er gebeurt voor elke combinatie van ingangen aan de uitgangen: dus aan a en b en daarna f en op die manier kan je de functionaliteit gaan beschrijven van wat die schakeling doet.
Dat is een ding: je hebt een schakeling en je weet wat die doet.
Meestal wil men het omgekeerde: men wil het omgekeerde, wat is er de schakeling voor? Maar zelfs dit is een onvolledige beschrijving van wat die schakeling doet want het beschrijft alleen de logische functie en in praktijk is een schakeling meer dan alleen zijn logische functie.
Een schakeling wordt altijd nog aangevuld met een tijdsgedrag. Het gaat er hier niet over hoe traag die poorten werken, het is niet dat als er bij x iets verandert, er onmiddelijk iets bij b verandert, dat gebeurt nooit ogenblikkelijk; ogenblikkelijk bestaat niet: het duurt altijd een zekere tijd. Dat betekent dat als we iets veranderen bij x bv., dat een tijdje later a gaat veranderen. y veranderen gaat een tijdje later b veranderen. Als a en b veranderd zijn, zal f een tijdje later veranderen. Wat je dan ziet verschijnen zijn vertragingen: het ogenblik dat de ingang verandert en dat de uitgang verandert, in elke reële schakeling is dat zo. Die vertraging gaat meespelen in hoe efficiënt de schakeling werkt. Het is dus belangrijk om die te kennen, je gaat dat nooit terugvinden in een logische functie, dat is bijkomende informatie.
Het tijdsgedrag nakijken is ook heel belangrijk om problemen te detecteren, problemen die je niet gaat zien als je alleen maar naar de functionaliteit gaat kijken. Bv. a en b veranderen "tegelijkertijd" (dat bestaat niet! De ene gebeurtenis zal altijd iets voor de andere plaatsvinden.) $\rightarrow$ dat kan zijn gevolgen hebben: als b iets vroeger verandert dan a, dan krijgen we heel kortstondig een waarde 0, dan zie je aan de uitgang plots een piekje verschijnen, wat er niet zou mogen zijn. Als je gewoon naar de logica kijkt, zou die er niet mogen zijn, maar in praktijk is dat er wel en dit kan voor problemen zorgen. Dat piekje dat er is, dat we niet verwachten, kan opeens gevolgen hebben in de verdere verwerking van de schakeling, dus we moeten ons daar bewust van zijn. We moeten dus niet alleen de logische functies controleren, maar het tijdsgedrag is even belangrijk.

Slide 16: Kijken we dan naar het omgekeerde: we hebben een functionaliteit, hoe kunnen we dat implementeren? Er staan twee implementaties, als je die gaat uitrekenen, dan zie je dat die identiek zijn, die doen juist hetzelfde.
Als we dit gedaan hebben kunnen we kiezen voor een van de implementaties en vanuit logisch standpunt maakt het geen enkel verschil. Het is dan niet zo dat je eender welke realisatie zomaar mag kiezen: de linkse heeft meer poorten dan de rechtse: het gaat meer kosten (meer transistoren, een grotere chip,...). De linkse gaat ook trager werken want als je van x naar de uitgang moet ga je door twee ingangen telkens moeten gaan waar dat rechts niet het geval is.
Als we dan toch kunnen kiezen willen we het zo goedkoop mogelijk en zo snel mogelijk hebben. Hoe sneller die schakeling is, hoe hoger de verwerkingskracht: we kunnen maar aan de volgende bewerking beginnen als de vorige gedaan is. Als die schakeling dus heel snel een resultaat levert kan die heel snel aan een volgende bewerking beginnen. Vandaar dat hoe sneller de schakeling werkt, hoe hoger de verwerkingskracht. Als we dan toch kunnen kiezen willen we die schakeling met de minimale kostprijs en die schakeling met de hoogste verwerk-

ingskracht (die zo snel mogelijk het resultaat geeft). We gaan daarvoor gebruik maken van de formules die erbij staan.
De snelheid is evenredig met het aantal ingangen, dus de linkse zal trager werken dan de rechtse. Op die manier kan men tussen de twee schakelingen kiezen en de beste kiezen.

Slide 18: Boole algebra kan gebruikt worden omdat we ook hier met twee (logische) waarden werken. In de Boole algebra vertrekt men van een aantal axioma's. De duale uitdrukkingen zijn hier ook steeds geldig: je mag 0 door 1 vervangen en omgekeerd en + door . en omgekeerd.

Slide 20: Wij gaan vooral gebruik maken van de wet van De Morgan: de inverse van het product is de som van de inversen en het inverse van de som is het product van de inversen.
In dit vak zal een distributiviteit voor de vermenigvuldiging ten opzichte van de optelling en distributiviteit van de optelling ten opzichte van de vermenigvuldiging gebruikt worden.