Inleiding

Er komen steeds meer configureerbare system-on-chips (SoC's) op de markt, waarin een microprocessor geïntegreerd is met een stuk programmeerbare hardware (FPGA). In deze cursus leer je hoe je deze complexe chips kunt programmeren en configureren zodat je ze toe kunt passen in een embedded systeem en op deze manier een totaal geïntegreerde oplossing voor een high-performance en/of real-time toepassing kunt realiseren.

Op een FPGA kun je op twee manieren een microprocessor integreren:

- Je kunt de HDL-code (VHDL of Verilog) van een processor in je FPGA-project integreren. Zo'n processor wordt een *soft core* genoemd omdat deze uit HDL-code bestaat, die desgewenst aangepast kan worden. Het is mogelijk om je eigen processorarchitectuur te ontwerpen, maar dit is niet aan te raden omdat je dan ook de benodigde softwareontwikkeltools, zoals assembler en compiler zelf moet ontwikkelen. Je kunt dan beter een bestaande processorarchitectuur gebruiken. Deze kun je zelf implementeren, maar het is eenvoudiger om gebruik te maken van een open-source implementatie. Er zijn open-source implementaties van vele bekende processorarchitecturen beschikbaar[1]. In deze cursus maken we gebruik van de Nios II soft core van Intel[2].
- Je kunt ook gebruik maken van een processor die al door de fabrikant van de FPGA in silicium is geïntegreerd. Veelal wordt in dit geval een ARM architectuur gebruikt. Zo'n processor wordt een *hard core* genoemd omdat deze uit vaste hardware bestaat en dus niet meer aangepast kan worden. Intel noemt een FPGA die voorzien is van een hard core een SoC FPGA[3]. In deze cursus zul je gebruik maken van een Cyclone® V SoC FPGAs die

[1] Zie bijvoorbeeld: https://opencores.org/projects?expanded=System%20on%20Chip%2CProcessor.

[2] Zie eventueel: https://www.intel.com/content/www/us/en/products/details/fpga/nios-processor/ii.html.

[3] Zie eventueel: https://www.intel.com/content/www/us/en/products/details/fpga.html.

een dual-core ARM Cortex-A9 bevat. Intel noemt deze hard core een hard processor system (HPS). AMD/Xilinx gebruikt de namen Zynq en Versal voor FPGA's met een hard core[4]. Microsemi gebruikt net als Intel de naam SoC FPGA[5].

Natuurlijk kun je beide mogelijkheden combineren en in één applicatie zowel een soft core als een hard core gebruiken. Op de hard core zou je dan een *General Purpose OS* (bijvoorbeeld Linux) kunnen draaien waarop de niet tijdkritische softwaretaken van de applicatie geïmplementeerd kunnen worden. Op de soft core zou je dan een *Real-Time OS* kunnen draaien waarop de tijdkritische softwaretaken van de applicatie geïmplementeerd kunnen worden.

Je bent na het volgen van deze cursus in staat om:

- in VHDL een hardware module te ontwerpen en implementeren met een memory bus interface zodat deze module, vanuit een soft of hard core processor, memory mapped te programmeren is;
- een embedded systeem op een FPGA te ontwerpen en implementeren bestaande uit een soft core, software, bestaande hardware modules en zelf in VHDL geïmplementeerde hardware modules;
- op dit embedded systeem een RTOS toe te passen;
- een embedded systeem op een FPGA te ontwerpen en implementeren bestaande uit een hard core, software die draait onder Linux, bestaande hardware modules en zelf in VHDL geïmplementeerde hardware modules;
- te beslissen of bepaalde functionaliteit van een embedded applicatie beter op een soft core, op een hard core of in hardware geïmplementeerd kan worden;
- verschillende vormen van High Level Syntheses met de voor- en nadelen van deze vormen te benoemen.

Zie voor verdere details betreffende deze cursus de cursushandleiding.

[4] Zie eventueel: https://www.xilinx.com/products/silicon-devices/soc.html.

[5] Zie eventueel: https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas.

Software

We maken tijdens deze cursus onder andere gebruik van de volgende software (de benodigde installatiebestanden kun je ook vinden in het CSC10 Team):

- Quartus Prime Lite Edition versie 18.1. Let op! Dit is niet de laatste versie, maar we gebruiken deze versie omdat deze versie Nios II EDS ondersteund zonder dat Windows Subsystem for Linux nodig is en zonder dat Eclipse handmatig apart geïnstalleerd moet worden. Bovendien heeft deze versie nog support voor VHDL-2008 iets wat Intel in latere (gratis) versies niet meer ondersteund[6].
- Cyclone V device support.
- Intel FPGA Academic Program. Dit programma moet in het directory worden geïnstalleerd waar Quartus is geïnstalleerd. Bij het installeren moet bij "Computer Systems" alleen de DE1-SoC aangevinkt worden.

Hardware

Bij deze cursus maken we gebruik van het DE1-SoC board van Terasic[7]. Je hebt dit board in bruikleen gekregen van Hogeschool Rotterdam en kunt alle opdrachten dus thuis uitvoeren.

Manier van werken

De hoeveelheid informatie die beschikbaar is over het gebruik van Intel SoC FPGA's is overweldigend. Om de hierboven beschreven leerdoelen te behalen maken we gebruik van door Intel beschikbaar gestelde tutorials. Bedenk goed dat het uitvoeren van deze tutorials geen doel op zich is. Zorg dat je begrijpt wat je doet

[6] Zie eventueel: https://community.intel.com/t5/Intel-Quartus-Prime-Software/Is-Intel-really-cutting-VHDL-2008-support-from-Quartus-Prime/td-p/699141.

[7] Zie: https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836.

en hou de leerdoelen in het oog. Vraag indien nodig je docent om extra uitleg. Wij adviseren je om een logboek bij te houden, zodat je e.e.a. snel terug kunt zoeken.

In de eerste vijf weken van deze cursus werk je aan weekopdrachten die niet beoordeeld, maar wel afgetekend (voor gedaan), zullen worden. Er wordt in deze weken in tweetallen gewerkt, maar bij het thuiswerken heeft elke student wel een eigen bordje. Vraag zelf om feedback. Als de docent jullie van feedback heeft voorzien zal hij de weekopdracht aftekenen. Alle weekopdrachten moeten afgetekend zijn voordat je aan de eindopdracht mag beginnen.

In de laatste vier weken werk je aan een eindopdracht waarmee je aantoont de leerdoelen behaald te hebben. Deze eindopdracht mag naar keuze individueel of samen met een andere student worden uitgevoerd. Je schrijft ook een kort verslag over deze eindopdracht. Ja mag deze eindopdracht zelf bedenken. De opdrachtomschrijving moet worden ingeleverd in het begin van week 5 en goedgekeurd worden door de docenten. Bedenk dat je slechts 8 werkdagen hebt om deze eindopdracht uit te voeren.

Opdrachten week 1 – Soft core

Je gaat deze week leren hoe je:

- een Nios II soft core kan integreren in een Cyclone V FPGA met behulp van *Intel's Platform Designer Tool For Quartus*;
- software voor deze soft core kan ontwikkelen in assembler en C en hoe je deze software kan testen met behulp van het *Intel FPGA Monitor Program*;
- software voor deze soft core kan ontwikkelen in C door gebruik te maken van een Board Support Package (BSP) gegenereerd door de *Nios II Embedded Design Suite (EDS)*.

Opdrachten eerste les.

De opdrachten behorende bij de eerste les zouden ongeveer één dag (8 uur) werk moeten kosten.

1.1 Download de *Introduction to the Platform Designer Tool*[8]. Voer deze uit maar maak hierin de volgende *aanpassingen*:

- In figuur 19 op pagina 19 kies je voor VHDL in plaats van Verilog. Het vinkje bij *Create block symbol file* kun je uitvinken.
- Vervang in de twee laatste regels op pagina 19 Verilog voor VHDL en `nios_systeem.v` voor `nios_systeem.vhd`.
- Figuur 20 op pagina 20 geeft een deel van de Verilog-code. De overeenkomstige VHDL-code (**`entity`** `nios_systeem`) kun je vinden in het bestand `nios_systeem.vhd`.
- 5.1.1 op pagina 21 slaan we over omdat we VHDL gebruiken (zie 5.1.2).
- Het bestand `DE1_SoC.qsf` dat wordt genoemd op pagina 23 kun je hier[9] downloaden.
- Voer paragraaf 7.3 twee maal uit om zowel de assemblercode uit figuur 23 als de C-code uit figuur 24 te testen. Bij het testen van het C-programma kun je Edit » Enable Source Level Debugging kiezen om stap voor stap door je C-code te kunnen lopen. Als je daarna weer een assemblerprogramma wilt testen, dan moet je Edit » Disable Source Level Debugging kiezen anders loopt het Monitor programma vast.

[8] https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/Introduction_to_the_Qsys_Tool.pdf

[9] https://software.intel.com/content/www/us/en/develop/articles/fpga-academic-boards.html

1.2 De meest linkse twee leds blijven altijd gedoofd[10]. Hoe komt dit? *Tip:* het probleem zit niet alleen in de hardware maar ook in de software. Breid het hardware systeem uit en pas de software aan zodat alle tien de schakelaars en leds gebruikt kunnen worden. De stand van de schakelaars moet op de leds worden weergegeven.

1.3 Lees nu de tutorial *Intel FPGA Monitor Program Tutorial for Nios II*[11] door zodat je op de hoogte bent van alle mogelijkheden die het monitorprogramma biedt. Bestudeer ook paragraaf 8.2 en het daarbij behorende *voorbeeldprogramma* zodat je een idee hebt hoe je interrupts in een C programma op een Nios II processor kunt implementeren. Deze code kun je vinden in: `C:\intelFPGA_lite\18.1\University_Program\Computer_Systems\sample_programs\nios2\c\interrupt_example`.

1.4 Voeg in *Platform Designer* twee PIO poorten toe aan de hardware en verbind die met de zes 7-segment displays. Hoe de 7-segment displays zijn aangesloten kun je vinden in de *DE1-SoC User Manual*[12]. De oudere borden (nummer 1 t/m 15) hebben revisie C en de nieuwere borden (nummer 16 t/m 50) hebben revisie G. Voeg tevens een Interval Timer toe aan de hardware die elke ms een interrupt genereert. Voeg in de software een teller toe die het aantal interrupts telt en geef de stand van deze teller weer op de zes 7-segment displays. Je mag zelf kiezen of je de waarde decimaal of hexadecimaal weergeeft. Maak gebruik van de code die is gegeven in het voorbeeld dat je in opdracht 1.3 hebt bestudeerd. De verschillende registers

[10] Als deze twee leds zwak branden dan ben je vergeten om in Quartus de *Unused pins* aan te passen naar "As input tri-stated", zie opdracht 1.1.

[11] https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/tutorials/Intel_FPGA_Monitor_Program_NiosII.pdf

[12] https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=836&PartNo=4

van de Interval Timer kun je vinden in *Embedded Peripheral IP User Guide*[13]. Een C-functie die een 4-bits (hex) digit omzet naar (active low) 7-segments code is gegeven in listing 1.

```
int hex_to_7_seg(int hex_digit) {
    if (hex_digit == 0x0) return 0x40;
    if (hex_digit == 0x1) return 0x79;
    if (hex_digit == 0x2) return 0x24;
    if (hex_digit == 0x3) return 0x30;
    if (hex_digit == 0x4) return 0x19;
    if (hex_digit == 0x5) return 0x12;
    if (hex_digit == 0x6) return 0x02;
    if (hex_digit == 0x7) return 0x78;
    if (hex_digit == 0x8) return 0x00;
    if (hex_digit == 0x9) return 0x10;
    if (hex_digit == 0xA) return 0x08;
    if (hex_digit == 0xB) return 0x03;
    if (hex_digit == 0xC) return 0x46;
    if (hex_digit == 0xD) return 0x21;
    if (hex_digit == 0xE) return 0x06;
    if (hex_digit == 0xF) return 0x0E;
    return 0x7F;
}
```

Listing 1: hex_to_7_seg.c: C-functie die een 4-bits digit omzet naar 7-segments code.

Opdrachten tweede les.

De opdrachten behorende bij de tweede les zouden ongeveer één dag (8 uur) werk moeten kosten.

Tot nu toe heb je gewerkt met het Monitorprogramma om code voor de Nios II te ontwikkelen en te testen. Deze manier van werken is vrij primitief omdat je

[13] https://bitbucket.org/HR_ELEKTRO/csc10/wiki/Docs/Embedded_Peripherals_IP_User_Guide_18_1.pdf#page=258

alles zelf moet doen. Deze manier van werken geeft echter de minste overhead in de uiteindelijke code. Als alternatief kun je ook gebruik maken van *Intel's Nios II Software Build Tools for Eclipse* (die meegeleverd wordt met Quartus 18.1). Met behulp van deze tools kun je een zogenoemd *Board Support Package* (BSP) voor je zelf met *Platform Designer* gebouwde systeem genereren. Vervolgens kun je de Nios II processor programmeren met behulp van dit BSP. Je kunt dan gebruik maken van veel bekende standaard C functies (zoals bijvoorbeeld `usleep` en `printf`). Het gebruik van de BSP zorgt er wel voor dat je meer programmageheugen nodig hebt.

1.5 Bestudeer het *Nios II Software Developer's Handbook*[14]. Lees in ieder geval:

- **3.1.** Getting Started with Nios II Software in Eclipse
- **6.2.** HAL Architecture for Embedded Software Systems
- **7.3.** The system.h System Description File
- **7.5.** UNIX-Style Interface
- **7.9.** Using Timer Devices
- **9.2.** Nios II Interrupt Service Routines

1.6 Pas de hardware die je voor opdracht 1.4 hebt gebouwd aan zodat het systeem 128 kB onchip geheugen bevat.

1.7 Genereer een BSP voor dit hardware systeem met behulp van *Nios II Software Build Tools for Eclipse*.

1.8 Pas de software die je hebt geschreven voor opdracht 1.4 aan zodat het kan draaien op dit systeem. Maak daarbij zoveel mogelijk gebruik maakt van de BSP. Als voorbeeld is in listing 2 de C-code gegeven om de waarde van de schakelaars naar de leds te kopiëren. De verschillende registers van de

[14] https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/archives/n2sw_nii5v2gen2-18-1.pdf

Interval Timer kun je vinden in *Embedded Peripheral IP User Guide*[15]. De beschikbare defines die je kunt gebruiken om de Interval Timer te programmeren vind je in de file altera_avalon_timer_regs.h. In paragraaf 9.2.4 van het *Nios II Software Developer's Handbook*[16] kun je lezen hoe je de ISR (Interrupt Service Routine) kunt registreren. Wij gebruiken een IIC (Internal Interrupt Controller). De betreffende functie is gedeclareerd in sys/alt_irq.h.

```
#include <system.h>
#include <altera_avalon_pio_regs.h>

int main(void)
{
    while (1)
    {
        int SW_value = ↩
 ↪ IORD_ALTERA_AVALON_PIO_DATA(SWITCHES_BASE);
        IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE, SW_value);
    }
    return 0;
}
```

Listing 2: lights_BSP.c: C-code die gebruik maakt van de BSP.

1.9 Test de software die je hebt geschreven voor opdracht 1.8 met behulp van *Nios II Software Build Tools for Eclipse*.

1.10 Breid het systeem nu uit zodat de Interval Timer gestart en gestopt kan worden door respectievelijk het commando "stop" of "start" via de JTAG

[15] https://bitbucket.org/HR_ELEKTRO/csc10/wiki/Docs/Embedded_Peripherals_IP_User_Guide_18_1.pdf#page=258

[16] https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/archives/n2sw_nii5v2gen2-18-1.pdf#page=257

UART naar het systeem te sturen. Maak ook nu weer zoveel mogelijk gebruik van de BSP.