Manier van werken

De hoeveelheid informatie die beschikbaar is over het gebruik van Intel SoC FPGA's is overweldigend. Om de in de cursushandleiding 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 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.

Opdrachten week 2 – Platform Designer component en RTOS

Vorige week heb je een systeem gebouwd op een FPGA dat een Nios II soft core bevat. Daarbij heb je gebruik gemaakt van verschillende IP cores[1] zoals On-Chip Memory, PIO (Parallel I/O), JTAG UART, Interval Timer en Nios II Processor. De documentatie van deze en andere standaard in Platform Designer beschikbare IP-cores kun je als volgt vinden:

- De *Embedded Peripheral IP User Guide*[2] beschrijft de standaard bij Quartus II meegeleverde IP cores.
- De beschrijving van IP cores uit het Intel® FPGA Academic Program kun je op deze webpagina[3] vinden.

Natuurlijk wil je het met Platform Designer gebouwde systeem combineren met je eigen hardware die je beschreven hebt in VHDL. Dit kan op twee manieren:

- Het systeem dat je met Platform Designer hebt gebouwd is 'gewoon' een VHDL **`entity`** en deze kun je als **`component`** op de gebruikelijke manier

[1] Intellectual Property core, zie Wikipedia.

[2] https://bitbucket.org/HR_ELEKTRO/csc10/wiki/Docs/Embedded_Peripherals_IP_User_Guide_18_1.pdf

[3] https://software.intel.com/content/www/us/en/develop/topics/fpga-academic/materials-ip-cores.html

instanstiëren in je eigen VHDL beschrijving. De in Platform Designer geëxporteerde poorten kun je dan verbinden met andere signalen uit je eigen VHDL beschrijving.

- Je kunt ook je eigen Platform Designer component ontwikkelen. Deze component kun je dan gebruiken bij het bouwen van een systeem in Platform Designer.

De eerste methode heb je al toe leren passen bij de cursus HWP01. De tweede methode ga je deze week toepassen.

Tot nu toe heb je bij het ontwikkelen van de software voor de Nios II processor gebruik gemaakt van een BSP. Bij grotere of tijdkritische programma's is het vaak handig om gebruik te maken van een Real-Time OS. Deze week ga je bij het ontwikkelen van de software gebruik maken van het RTOS µC/OS-II.

Je gaat deze week leren hoe je:

- een eigen Platform Designer component kan definiëren met behulp van VHDL;
- deze component kan opnemen in een te bouwen systeem dat geïmplementeerd kan worden in een FPGA;
- hoe je deze component kan aansturen vanuit software met behulp van een RTOS.

Opdrachten eerste les.

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

In week 1 heb je de 7-segment displays rechtstreeks aangestuurd met behulp van een PIO. De omzetting van binaire code naar 7-segmentscode heb je in software geïmplementeerd. Er waren dus 7 bits in het PIO data-out register nodig om één 7-segment display aan te sturen. Omdat het PIO data-out register maximaal 32-bits breed is, kon je maar maximaal vier 7-segment displays aansturen met behulp van één PIO. Om die reden heb je twee PIO's gebruikt. Deze week ga je zelf

een Platform Designer component maken waarmee je de 7-segment displays gaat aansturen. Deze component implementeert de omzetting van binaire code naar 7-segmentscode in hardware. Daardoor zijn in het output register nog maar 4 bits nodig om één 7-segment display aan te sturen. We kunnen dan dus met één 32 bits register maximaal acht 7-segment displays aansturen.

2.1 Download de *Making Platform Designer Components*[4]. Voer deze uit maar maak hierin de volgende *aanpassingen*:

- Maak het On-Chip Memory, dat je toevoegt op pagina 11, 131072 bytes groot. Fixeer het beginadres van het On-Chip Memory op adres `0x0000_0000`.
- Voeg bovenaan pagina 13 ook een JTAG UART en een Interval Timer (Simple periodic interrupt van 1 ms) toe aan het systeem. Figuur 12 op pagina 13 ziet er dus iets anders uit. Noteer het adres van de `reg32_component` in de memory map.
- In figuur 27 op pagina 27 kies je voor VHDL in plaats van Verilog. Het vinkje bij *Create block symbol file* kun je uitvinken.
- De code die gegeven is in figuur 30 op pagina 30 bevat een syntaxisfout en een verkeerde poortnaam.
- Voeg boven aan pagina 32 de tekst toe: schakel de source level debugging uit via het menu Edit 〉 Disable Source Level Debugging.
- Vervang op pagina 32 het adres `0x00000000` met het adres van de `reg32_component` in de memory map.

2.2 Het omzetten van hexadecimaal naar 7-segmentscode wordt in de tutorial in het top level gedaan. Het is echter veel logischer om dit in de zelfgemaakte Platform Designer component te doen. Deze component functioneert dan als een 7-segment hardware driver die je vanuit software kan aansturen.

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

In de top-level module hoef je dan alleen de component `embedded_system` nog maar te instantiëren en met de juiste pinnen te verbinden. Pas de in opdracht 2.1 gemaakte hardware aan zodat de omzetting van hexadecimaal naar 7-segmentscode in de zelfgemaakte component gebeurt.

2.3 Pas ook de software die je bij opdracht 1.10 hebt gemaakt aan zodat het werkt met de bij opdracht 2.2 gebouwde hardware. Test deze software met behulp van *Nios II Software Build Tools for Eclipse*.

Het systeem dat je bij opdracht 2.2 hebt gebouwd stuurt slechts vier 7-segments displays aan. Het is nu een koud kunstje om de zelfgemaakte component alle zes de 7-segments displays aan te laten sturen. We hebben namelijk nog 16 bits over in het 32-bits register.

Om de opdracht leerzamer te maken kiezen we echter een andere aanpak. De zelfgemaakte component heeft een *Avalon Memory-Mapped Interface*[5]. Deze interface bevat ook een adres en dit adres kan gebruikt worden om verschillende registers in de component te adresseren. Je gaat er nu voor zorgen dat je de overige twee 7-segment displays via een tweede register kan aansturen.

2.4 Breid de in opdracht 2.2 aangemaakte component uit met één adreslijn en extra signalen in de *Conduit signals* en zorg ervoor dat de component 2 registers bevat. De vier meest rechtse 7-segment displays moeten (via binaire codes) aan te sturen zijn via het eerste register en de twee meest linkse 7-segment displays moeten (via binaire codes) aan te sturen zijn via het tweede register.

2.5 Pas ook de software aan zodat alle zes 7-segment displays aangestuurd kunnen worden.

[5] Zie https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf.

2.6 Test deze software met behulp van *Nios II Software Build Tools for Eclipse*.

Opdrachten tweede les.

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

Tot nu toe heb je bij het ontwikkelen van de software voor de Nios II processor gebruik gemaakt van een BSP. Bij grotere of tijdkritische programma's is het vaak handig om gebruik te maken van een Real-Time OS. Je gaat bij de volgende opdrachten gebruik maken van het RTOS µC/OS-II.

2.7 Bestudeer hoofdstuk 11 van het *Nios II Software Developer's Handbook*[6].

2.8 We maken gebruik van het boek: Jean J. Labrosse. *µC/OS II The Real-Time Kernel – User's Manual*. Micriμm Press, 2015. URL: https://bitbucket.org/HR_ELEKTRO/csc10/wiki/Boeken/100-uC-OS-II-003.pdf. Zorg dat je begrijpt hoe je een taak kan starten in dit RTOS en hoe taken met elkaar kunnen communiceren.

2.9 Maak in Nios II Software Build Tools for Eclipse een nieuw project aan met het menu File > New > Nios II Application and BSP from Template. Selecteer de bij opdracht 2.4 gebouwde SOPC Information File. Kies voor de *Hello MicroC/OS-II* project template. Dit project kun je als basis gebruiken om je eigen software met behulp van het RTOS MicroC/OS-II te ontwikkelen.

Laad het bij opdracht 2.4 gebouwde systeem op het DE1-SoC board en test het bij opdracht 2.9 aangemaakte project. Verbeter eventuele fouten.

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

2.10 Maak nog een project aan dat gebruik maakt van de hierboven aangemaakte BSP en MicroC/OS-II. Codeer een applicatie twee toestanden kent 'lopen' en 'stilstaan' en die bestaat uit 3 taken:

- Een periodieke taak die, in de toestand lopen, elke milliseconde een 16-bits teller met 1 verhoogt en de waarde van deze teller in hexadecimale code weergeeft op de meest rechtse vier 7-segment displays. In de toestand stilstaan behoud de teller zijn waarde. De teller moet starten op 0.
- Een periodieke taak die, in de toestand lopen, elke seconde een 8-bits teller met 1 verhoogt en de waarde van deze teller in hexadecimale code weergeeft op de meest linkse twee 7-segment displays. In de toestand stilstaan behoud de teller zijn waarde. De teller moet starten op 0.
- Een taak die wacht tot er een karakter binnenkomt op de JTAG UART. Als het karakter `'0'` wordt ontvangen, moet de applicatie in de toestand stilstaan komen en als het karakter `'1'` wordt ontvangen, moet de applicatie in de toestand lopen komen.

Ken prioriteiten toe met behulp van Rate Monotonic Priority Assignment zoals je bij de cursus RTS10[7] hebt geleerd. Bereken daartoe ook de maximale periodetijd van de sporadic task (de derde taak). Zorg ervoor dat dit project zonder fouten gecompileerd kan worden.

Laad het bij opdracht 2.4 gebouwde systeem op het DE1-SoC board en debug het bij opdracht 2.10 aangemaakte project totdat het werkt volgens de gegeven specificaties.

[7] Zie eventueel: https://bitbucket.org/HR_ELEKTRO/rts10/wiki.