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 3 – Hardcore en Linux

Tot nu toe heb je nog geen gebruik gemaakt van de dual core ARM-Cortex A9 hard core die aanwezig is in de Cyclone V SoC FPGA. Daar gaat dit lab verandering in brengen. De hard core wordt door Intel het Hard Processor System (HPS) genoemd.

In week 1 heb je het Monitor Programma van Intel gebruikt om de Nios II processor *bare metal*[1] te programmeren. Deze week gaan we ditzelfde programma gebruiken om de HPS bare metal te programmeren. In eerste instantie gebruiken we hierbij door Intel gedefinieerde hardware de zogenoemde DE1-SoC Computer. Later gaan we ook zelf een systeem bouwen dat een HPS bevat en gaan we Linux op dit systeem draaien. We kunnen het systeem dan in plaats van bare metal ook vanuit het Linux OS programma's laten uitvoeren.

Je gaat deze week leren hoe je:

- de dual core ARM-Cortex A9 bare metal kunt programmeren;
- een Hard Processor System (HPS) configureert binnen Platform Designer;
- een devicetree genereert voor jouw HPS-systeem om Linux te booten;
- een FPGA-schakeling aanstuurt vanuit Linux;
- een simpele Linux kernel module schrijft die de FPGA-schakeling kan aansturen en kan reageren op een interrupt afkomstig van de FPGA-schakeling;
- een character device installeert zodat je vanuit user space (zonder superuser rechten) de leds van het board via de FPGA kunt aansturen door naar een (virtueel) bestand te schrijven;

[1] Als een programma rechtstreeks de hardware aanspreekt, zonder gebruik te maken van een operating systeem, dan wordt dit *bare metal programming* genoemd. Zie https://en.wikipedia.org/wiki/Bare_machine.

- een character device ontwikkeld en installeert zodat je vanuit user space de schakelaars van het board via de FPGA kunt inlezen door uit een (virtueel) bestand te lezen;
- een interrupt afkomstig van de schakelaars kunt afhandelen in user space door het implementeren van een character device dat asynchrone I/O ondersteunt.

Voer de volgende opdrachten tijdens de eerste les uit.

3.1 Download de *Monitor Program Tutorial for the ARM Processor*[2]. Je hebt het Monitor Program in week 1 al gebruikt. Voer de stappen beschreven in de volgende delen van de tutorial uit:

- paragraaf 3.1 tot en met 3.8;
- hoofdstuk 6.

3.2 Schrijf nu zelf een C-programma dat de waarde van een teller die regelmatig opgehoogd wordt weergeeft op de 7-segment displays. Als uitgangspunt kun je het C voorbeeld-programma *Getting Started* gebruiken. Als je een hardware timer wilt gebruiken (dat is niet verplicht), dan kun je het C voorbeeldprogramma *Interrupt Example* gebruiken.

3.3 Later in de opdrachten zullen we de EDS command line gebruiken. Deze zit inbegrepen in het SoC EDS programma van Intel. De betreffende installatiebestanden kun je vinden in het CSC1 Team onder het tabblad *Lesmateriaal* of zelf downloaden van Altera[3]. Installeer dit programma.

Linux

Linux booten van sd-kaart

We gaan de HPS nu niet meer bare metal maar met behulp van Linux programmeren.

3.4 Er is voor dit lab een Linux image opgezet. Voordat we ons eigen HPS-systeem gaan configureren zullen we dit eerst gaan booten en even kijken wat we ermee kunnen. Op de wiki onder *Linux Image Week 3* staan instructies om de linux image zelf te flashen op

[2] https://github.com/HR-ELEKTRO/csc1/wiki/Docs/Intel_FPGA_Monitor_Program_ARM.pdf

[3] Windows of Linux.

een micro-sd-kaartje van tenminste 4 GB.[4] Stop een geflasht sd-kaartje in de DE1-SoC. Voordat we het systeem opstarten moeten eerst de MSEL switches onder het bordje worden ingesteld[5] naar `00000`, doe dit.

Er zijn meerdere manieren om het systeem te bedienen. De meegeleverde image heeft bij de eerste boot geen framebuffer geconfigureerd, dus we zullen niet een monitor aansluiten op het bord. In eerste instantie zullen we verbindingen maken via een UART-connectie, later zou je via een netwerkverbinding[6] een SSH- of bijv. VNC-connectie kunnen opzetten. Dit is niet het doel van dit lab dus we beperken ons nu tot de UART-verbinding.

3.5 Zorg ervoor dat je een mini-USB kabel aansluit tussen je computer en de *UART to USB* connector, en dat eventuele drivers zijn geïnstalleerd[7]. Open het monitorprogramma PuTTY[8] op 115200 baud naar de juiste COM-poort[9]. We gebruiken PuTTY omdat deze beter om kan gaan met een Linux terminal dan andere terminalprogramma's zoals TeraTerm. Het is hierbij handig om bij de PuTTY instellingen, onder Terminal » Keyboard , Linux te selecteren voor de function keys. Tevens is het standaard venstergrootte van PuTTY 80 × 24, verander dit naar iets groters, bijvoorbeeld 120 × 30, onder het Window menu. Je kunt deze instellingen opslaan, zodat je ze in het vervolg met één klik kunt laden.

Druk op de rode knop om de DE1-SoC in te schakelen. Je ziet in PuTTY de bootloader (U-Boot) starten en deze zal de Linux kernel opstarten waarna je een scherm zal zien zoals in figuur 1.

Login met `student` als gebruikersnaam en `temppwd` als wachtwoord. Je bevind je nu in een zogenoemde bash shell[10] waarin je Linux-commando's kunt uitvoeren. Mocht

[4] Mocht je zelf geen kaartje of sd-kaart-lezer hebben dan kun je een voorgeprogrammeerd kaartje ophalen in de docentkamer om te lenen. Ook zijn er een aantal sd-kaart-lezers beschikbaar op uitleenbasis.

[5] Met MSEL op `00000` kan U-Boot de FPGA configureren

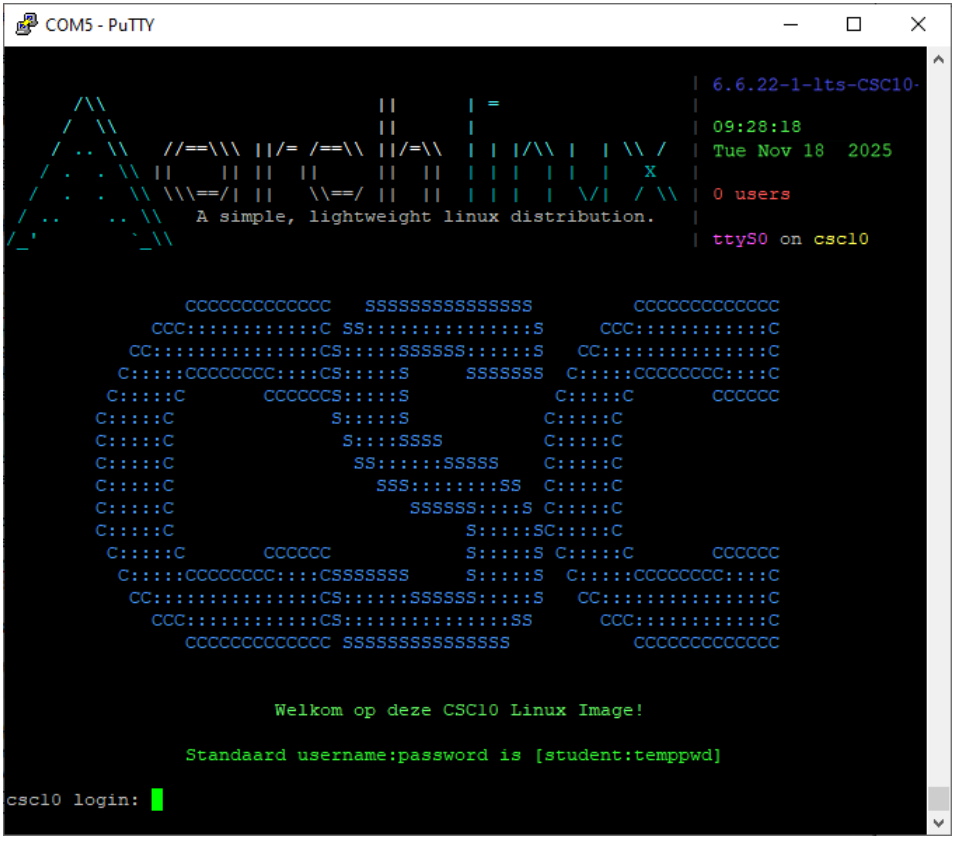
[6] De kernel is gecompileerd met support voor de meeste USB wifi-adapters, naast de optie voor een bedrade verbinding.

[7] Voor borden vanaf nummer 50 (revisie H) heb je deze driver nodig: https://www.silabs.com/software-and-tools/usb-to-uart-bridge-vcp-drivers. Lagere nummers hebben deze driver niet nodig.

[8] Zie https://www.chiark.greenend.org.uk/~sgtatham/putty/

[9] Die vind je in Windows in "Apparaatbeheer" onder "Poorten".

[10] bash heeft vele mogelijkheden en is niet alleen een command interpreter maar ook een programmeertaal. Zie eventueel https://www.gnu.org/software/bash/manual/bash.pdf.



Figuur 1: Het scherm na opstarten van Linux.

je geen ervaring met Linux hebben; in bijlage A kun je een lijstje met standaard commando's terugvinden die je in een Linux-omgeving kunt gebruiken.

3.6 Via UART zal de terminalgrootte niet automatisch meeveranderen wanneer je de venstergrootte van PuTTY verandert. Dit kan later problemen opleveren, dus het is verstandig de terminal iets groter te maken. De docent heeft hiervoor een kleine bash functie al geïntegreerd. Voer `rsz` uit en de venstergrootte van de uart terminal zal gelijk worden aan de grootte van PuTTY.

3.7 Om de image zo klein mogelijk te houden is alle ongevulde ruimte uit de root-partitie verwijderd. Nu de image op de sd-kaart staat is het handig om deze root-partitie te

vergroten van 3 GB naar de beschikbare ruimte op de sd-kaart (bijvoorbeeld 16 GB). Dit is een eenmalige actie.

Voer de volgende stappen uit en let heel goed op of je het correct hebt ingetypt:

```
# Laat grootte partities zien:
$ df -h

# Vergroot root partitie:
$ sudo parted /dev/mmcblk0 resizepart 2 100%

# Vergroot filesystem:
$ sudo resize2fs /dev/mmcblk0p2

# Check nieuwe partitie grootte:
$ df -h
```

Nu is de root-partitie groot genoeg gemaakt voor toekomstig werk. Type als laatste `sudo reboot now` om het Linux-systeem te herstarten.

Als het goed is zijn de 7-segment displays nu aan en knippert led0. Deze led wordt aangestuurd vanuit de hardware en vanuit de software kunnen we alleen led1 t/m led9 aansturen.

Om te zien of de FPGA-hardware bereikbaar is kunnen we het volgende doen: `cat /sys/class/fpga_bridge/br*/name` om de HPS-FPGA bruggen te laten zien en `cat /sys/class/fpga_bridge/br*/state` om te kijken welke actief zijn. Je ziet, als het goed is, de namen van alle vier de bruggen verschijnen.

3.8 Ga naar de directory `CSC10_Development/test_pio/` hier is door de docent een klein programma neergezet. Open `main.c` met bijv. `nano -l main.c`[11].

Op regel 18 wordt `/dev/mem`[12] geopend. Op regel 24 wordt het fysiek geheugenbereik van de LightWeight AXI-bus gekoppeld aan het virtuele geheugen met behulp van de functie `mmap`[13] en een pointer hiernaar wordt teruggegeven.

[11] De optie `-l` zorgt ervoor dat je de regelnummers in de kantlijn ziet.

[12] Zie https://man7.org/linux/man-pages/man4/mem.4.html.

[13] Zie https://pubs.opengroup.org/onlinepubs/9799919799/functions/mmap.html.

De offset naar de registers van de led PIO-module is in deze configuratie `0x10040`, deze tellen we op bij het adres van de LW-bus. Vervolgens kunnen we de leds aansturen met registers zoals bij RTS1. De docent heeft voor het gemak de `HWREG(x)` macro toegevoegd.

Sluit de editor (Ctrl + X voor nano). Compileer de code met `gcc main.c -o main`. Voer de code uit met `sudo ./main`[14]. Druk KEY0 in en de leds (led1 t/m led8) zullen binair optellen. Als al deze leds branden, dan wordt het programma beëindigd. Je kunt het programma ook eerder afbreken door op Ctrl + C te drukken.

Gefeliciteerd! Je hebt een koppeling tussen de HPS en FPGA gebruikt! Nu is het natuurlijk leerzamer, en noodzakelijk, om zelf een HPS-systeem te configureren en hier je eigen hardware aan toe te voegen. De volgende opdrachten zullen je helpen een HPS-systeem op te zetten.

Eigen HPS/Linux systeem opzetten

Stap 1: Configureer systeem in Platform Designer

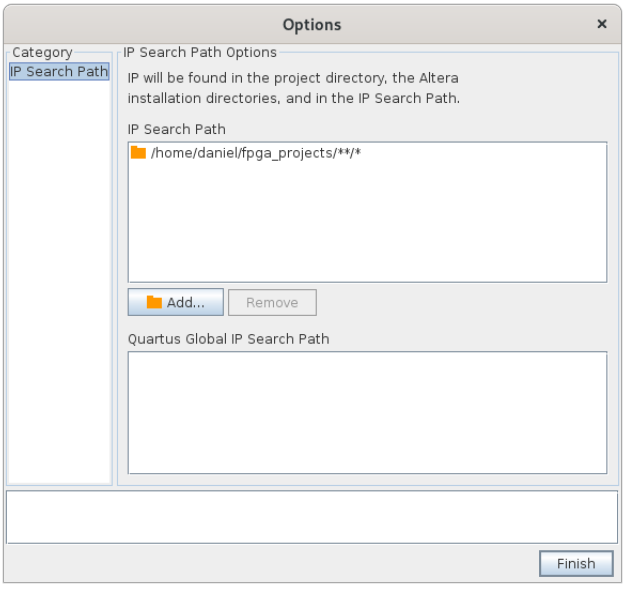
3.9 Om dadelijk de HPS makkelijk te kunnen configureren in Platform Designer, heeft de docent een standaard preset-bestand gemaakt. Dit bestand beschrijft alvast alle klok- en timings-parameters van het DDR3 geheugen. Download `DE1-SoC_bordje_CSC1.qprs` en plaats het in een locatie waar je het dadelijk terug kunt vinden, bijv. in je FPGA projecten folder.

Open nu Quartus en maak een nieuw project aan voor het DE1-SoC board. In deze opdracht heeft de docent `hps_demo` gebruikt. Je kunt een andere naam gebruiken maar zorg ervoor dat je in volgende stappen `hps_demo` vervangt met jouw bestandsnaam.

Binnen dit project wordt Platform Designer gebruikt om de HPS te configureren, open Platform Designer. Om bovengenoemde presets toe te voegen ga je naar het menu Tools > Options en voeg je het directory toe waar je het *.qprs bestand hebt opgeslagen door op Add te drukken. Zie figuur 2.

3.10 Zoek naar HPS in de IP Catalog en selecteer "Arria V/Cyclone V Hard Processor System". Druk op Add. Als er een configuratie window opent kun je op Finish drukken. Zorg

[14] Het programma moet met superuser rechten worden uitgevoerd omdat het (virtuele) bestand `/dev/mem` alleen toegankelijk is voor `root`.



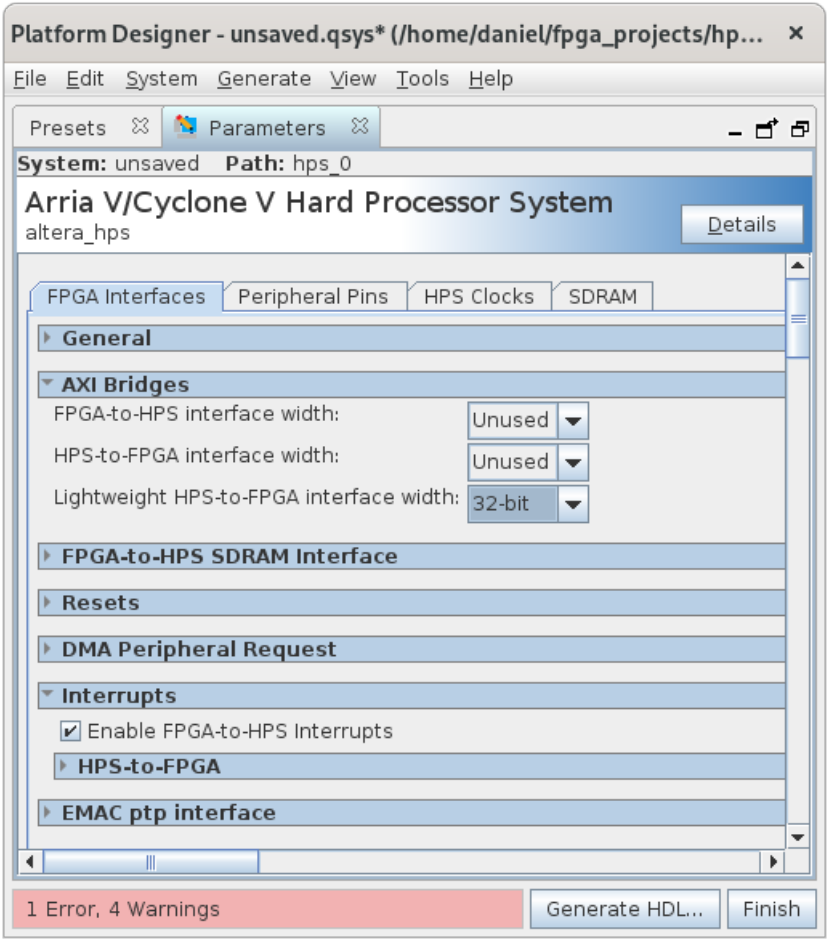
Figuur 2: Ip search path toevoegen binnen Platform Designer

ervoor dat de preset window aan staat (View > Presets) en met `hps_0` geselecteerd, Apply de preset genaamd *DE1-SoC bordje CSC1*. Nu rest ons om de HPS verder te configureren.

Het configureren van de HPS bestaat grofweg uit twee delen; Het instellen van de bussen en de peripheral pin multiplexer van de HPS. Je kunt ervoor kiezen om een peripheral pin te koppelen aan de FPGA (LoanIO), aan een externe chip (zoals SPI NAND) of te gebruiken als directe IO m.b.v. de interne GPIO-module van de HPS. Voor ons eerste systeem houden we de bussen simpel.

3.11 Dubbelklik op `hps_0` om de HPS te configureren. Configureer de FPGA Interfaces-tab zoals weergeven in figuur 3. De ingeklapte velden hebben alle features uit staan. De standaard configuratie heeft alle fpga2hps en hps2fpga bruggen aan staan, deze zetten we voor nu uit.

N.B. elke instellingswijziging voor de HPS duurt een paar seconden. Wacht dus even na het klikken voordat je verder gaat.

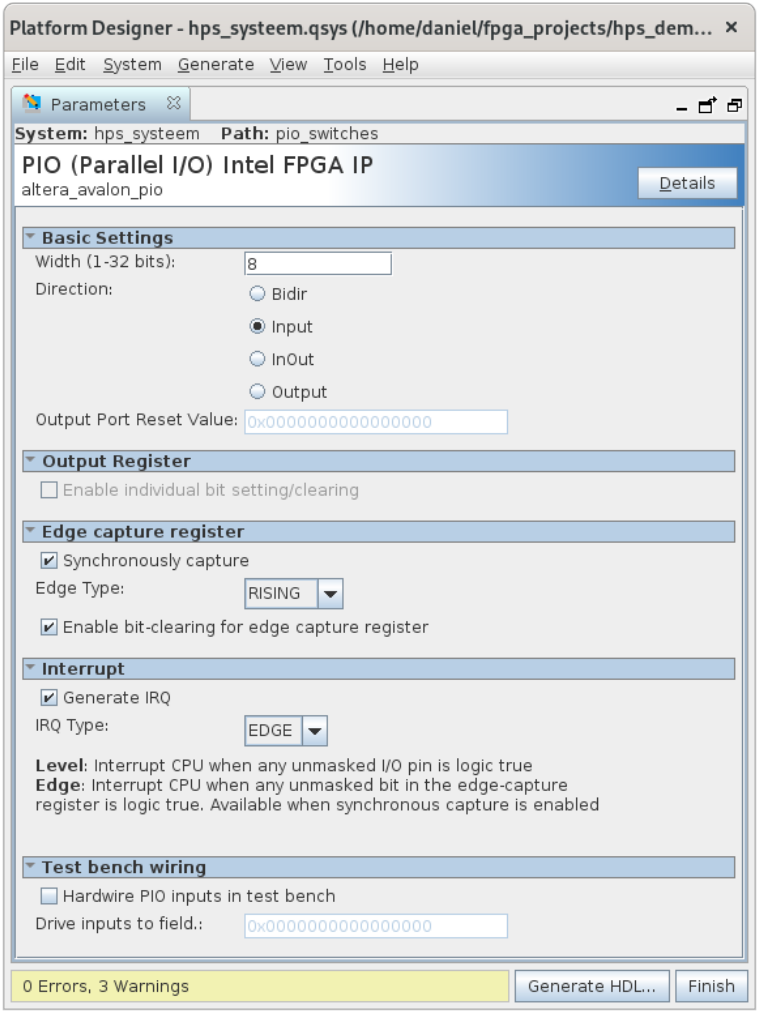


Figuur 3: Interface instellingen voor HPS in Platform Designer

3.12 De instellingen van de Peripheral Pins-tab is grotendeels ingevuld door onze preset. Er moeten nog wel een aantal GPIO-pinnen gekoppeld worden. Zoek in de DE1-SoC User Manual op welke knop en welke led direct aan de HPS zijn gekoppeld. Koppel deze vervolgens aan de GPIO-module door de betreffende GPIO-knop in de *Peripherals Mux Table* in te drukken.

3.13 Sluit de HPS-configuratie en voeg een PIO-module toe aan het systeem. We gaan deze PIO gebruiken om led0 t/m led7 mee aan te sturen. Gebruik de standaard instellingen

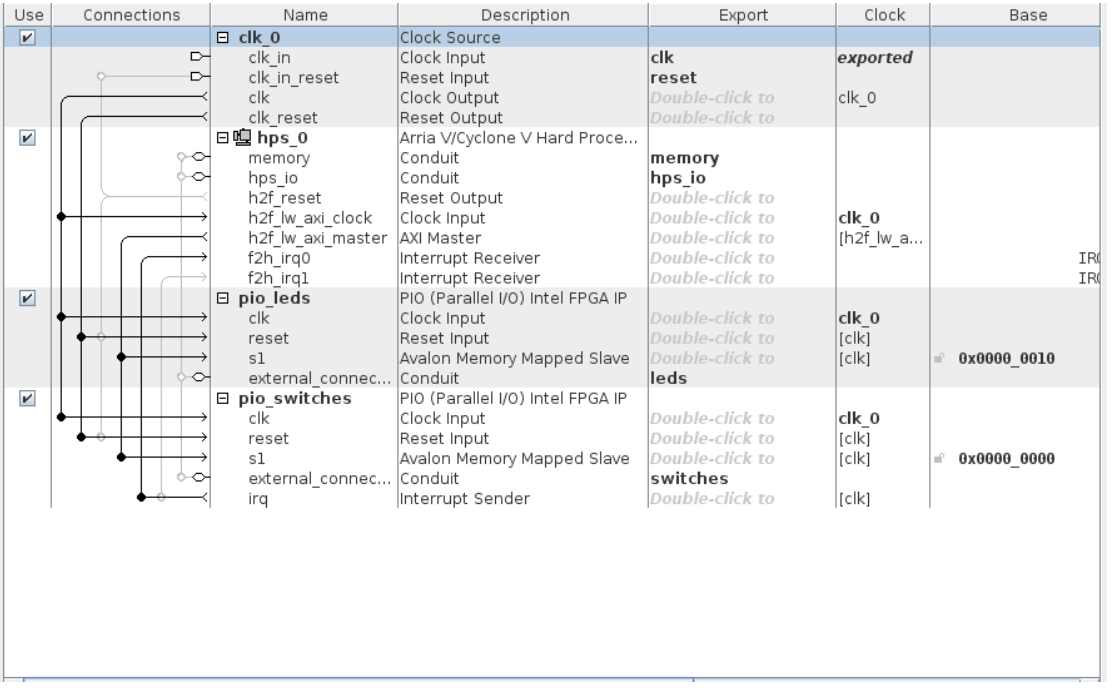
(8 bits uitgang). Voeg een tweede PIO-module toe met de instellingen van figuur 4. We gaan deze PIO gebruiken om switch0 t/m switch7 mee in te lezen.



Figuur 4: Instellingen van de tweede PIO-module

3.14 Koppel de signalen aan elkaar zoals aangegeven in figuur 5. Zorg er ook voor dat de juiste signalen worden geëxporteerd: `hps_io`, `memory`, `leds` en `switches`. `hps_io` en `memory` zullen automatisch worden gekoppeld, `leds` en `switches` moeten we later koppelen door de gegenereerde code aan te passen.

Ken als laatste de basisadressen toe via System » Assign Base Adresses.



Figuur 5: Koppelingen IP blocks in platform designer

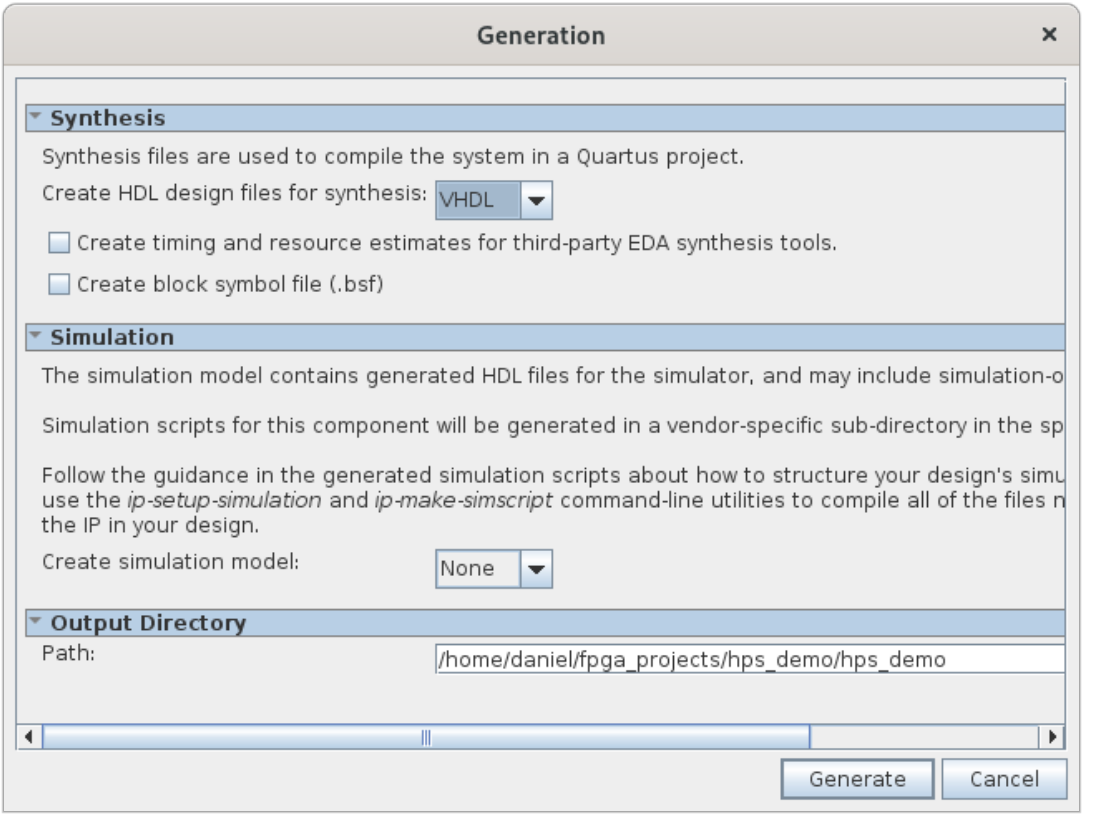
3.15 We zijn nu klaar met configureren van ons eerste HPS-systeem. Sla het systeem op met bijvoorbeeld de naam `hps_systeem` en klik op Generate HDL. Gebruik de instellingen van figuur 6 en drukop Generate.

Voordat we Platform Designer sluiten is het handig om de *Component Instantiation Template* te kopieren. Ga naar Generate » Show Instatiation Template... . Selecteer VHDL als taal, en druk op Copy. Dit zullen we bij de volgende stap gebruiken om de top-level file op te zetten.

3.16 Ga terug naar Quartus, klik op Project » Add/Remove files from project en voeg het `.qip`-bestand toe en druk op OK (figuur 7). Vergeet niet de pin-assignments te importeren met het `.qsf`-bestand [15].

Maak een nieuw bestand aan en sla deze op als `hps_demo.vhd`. Dit wordt ons top-level bestand.

[15] Kijk in je logboek als je niet meer weet hoe dat moet.



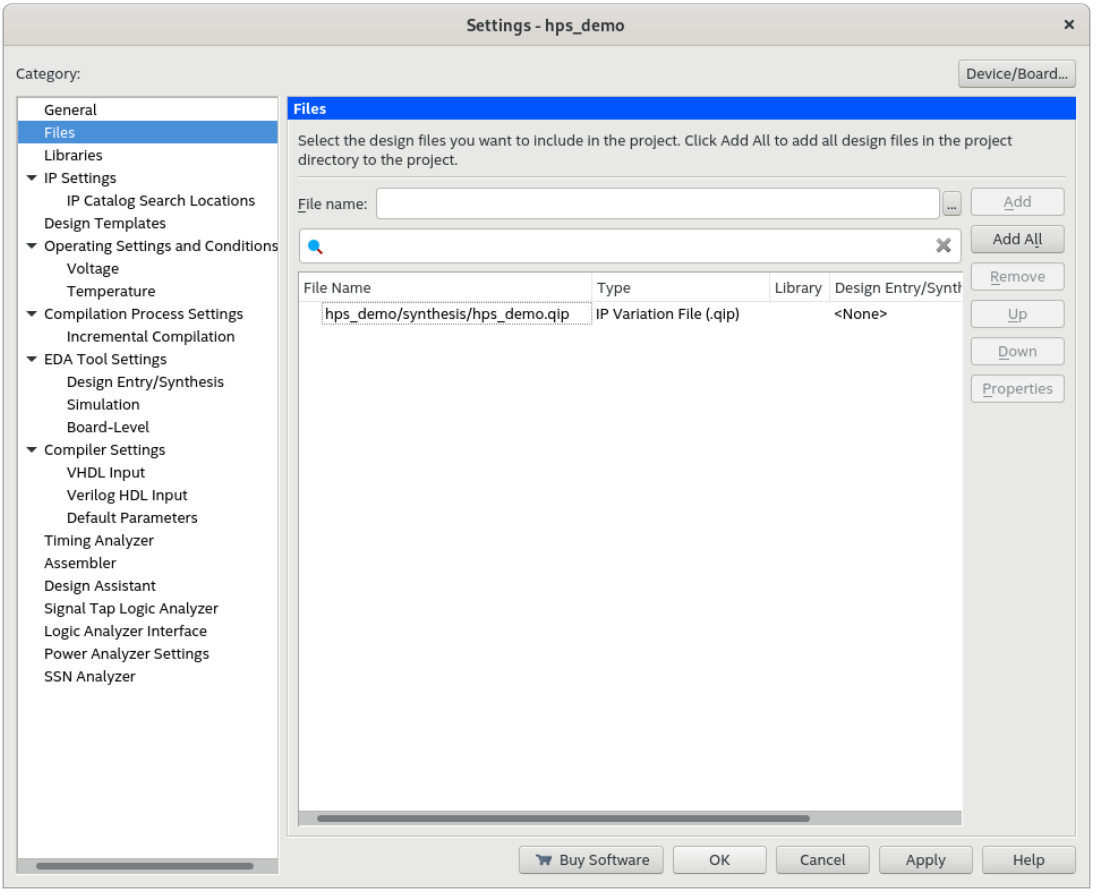
Figuur 6: Instellingen HPS configuratie.

Maak in het VHDL-bestand een nieuwe entity aan genaamd `hps_demo`. De `PORT()`-declaratie mag nu leeg blijven. In de *Architecture* van `hps_demo` plakken we de eerder gekopieerde tekst. Vergeet niet `BEGIN` te plaatsen tussen de component en de port map van de component.

Kopieer de `PORT()`-beschrijving van de `hps_systeem` component en gebruik deze voor de `hps_demo` entity. Verander `clk_clk`, `reset_reset_n`, `leds_export` en `switches_export` naar de betreffende namen uit het geïmporteerde `.qsf`-bestand.

De port-map van `hps_systeem` heeft `CONNECTED_TO_` voor elk signaal staan. Haal deze weg met zoek en vervang.

De gewijzigde signaal namen van de entity moeten nog worden gekoppeld aan de component via de portmap. Doe dit.

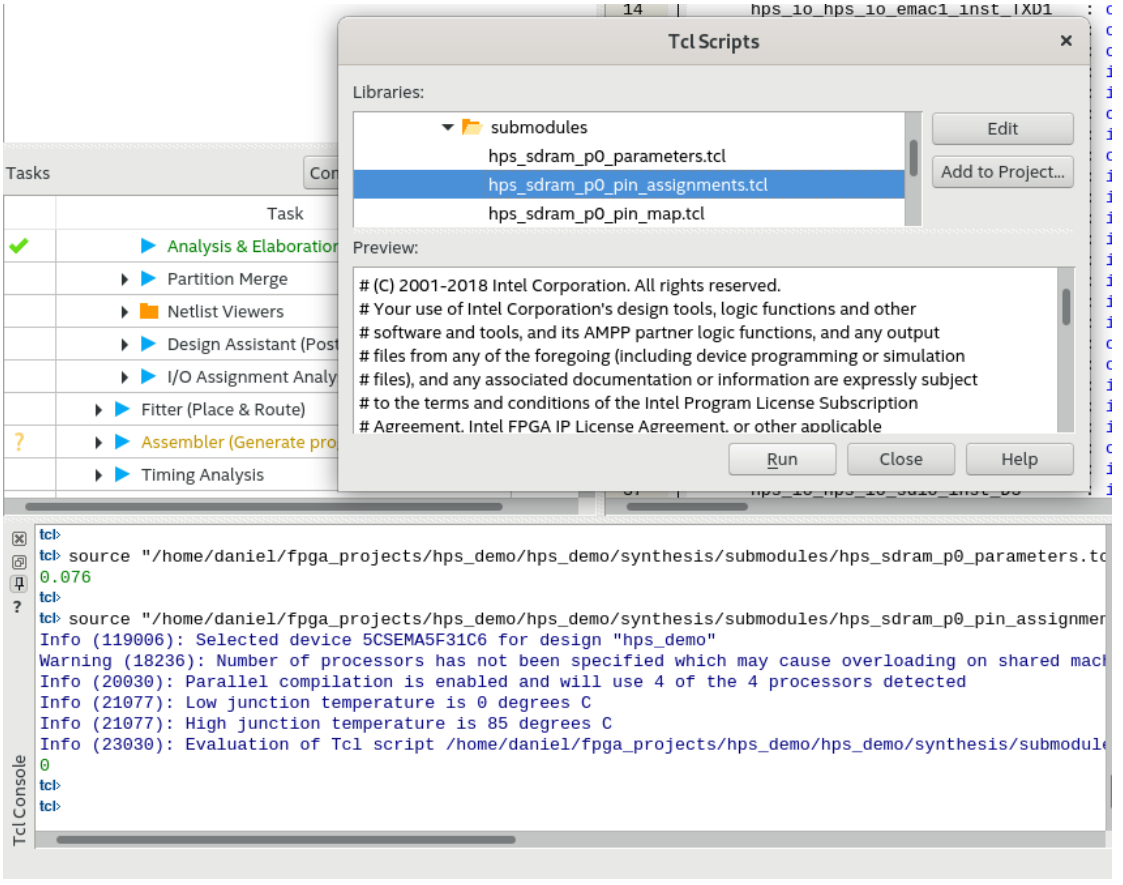


Figuur 7: qip Bestand toevoegen.

Stel het `hps_demo.vhd` bestand in als top-level entity.

3.17 Voer de *Analysis and Synthesis* uit. Dit duurt zo'n 5–15 minuten dus haal even een bakje koffie! Als deze stap klaar is moeten we TCL-scripts uitvoeren om de RAM-instellingen goed toe te passen. Ga naar Tools > TCL Scripts. Voer de eerste twee tcl-bestanden uit (parameters en pin assignments), zie figuur 8. Als dit is toegepast zonder fouten kun je het project *Compileren* en nog eens pauze houden.

Nu zijn we klaar met het opzetten van het systeem. Het opstartproces van de DE1-SoC is als volgt:



Figuur 8: TCL-scripts uitvoeren.

1. De SoC zoekt de preloader op de SD-kaart. Deze staat op een vaste plek in de eerste 1 MB.
2. De preloader zoekt de FAT boot-partitie en voert het bestand `u-boot.img` uit. Dit bestand bevat de code voor de U-Boot bootloader[16].
3. U-Boot configureert het FPGA gedeelte met het bestand `soc_system.rbf` en start vervolgens Linux. Het geeft hierbij het bestand `socfpga.dtb` aan de Linux-kernel. Dit is een devicetree blob (dtb) en het laat Linux weten hoe dit platform is opgebouwd en waar welke hardware zit[17].
4. Linux start en initialiseert alle hardware om het OS te kunnen uitvoeren.

[16] Zie https://en.wikipedia.org/wiki/Das_U-Boot

[17] Zie https://docs.kernel.org/6.6/devicetree/usage-model.html

Tot nu toe hebben we enkel een `.sof`-bestand vanuit Quartus gekregen. Dat is het bestand dat je in eerdere weken in de FPGA hebt geladen via het Monitorprogramma of Quartus. U-Boot wil een binaire versie van het `.sof`-bestand dat het zonder problemen direct kan schrijven naar de FPGA. De binaire versie is een `.rbf`-bestand (Raw Binary File). De volgende stappen leggen uit hoe je dit `.rbf`-bestand kan genereren en hoe je de devicetree blob maakt voor de Linux-kernel.

Stap 2: Binary van HPS systeem genereren (soc_system.rbf)

3.18 Binnen Quartus, ga naar File > Convert Programming Files. Gebruik de *exacte* instellingen van figuur 9 en druk op Generate. Het bestand `soc_system.rbf` wordt later naar de SD-kaart gekopieerd.

Stap 3: Devicetree blob genereren (socfpga.dtb)

3.19 Download `hps_common_board_info.xml` en `soc_system_board_info.xml`[18] en kopieer deze bestanden naar jouw projectmap.

Start de *SoC EDS Command Shell*. Dit is onderdeel van het SoC EDS pakket dat in opdracht 3.3 is geïnstalleerd. Navigeer naar jouw projectmap.

Eerst genereer je het leesbare devicetree bestand `socfpfa.dts` op basis van jouw `.sopcinfo` bestand met dit commando:

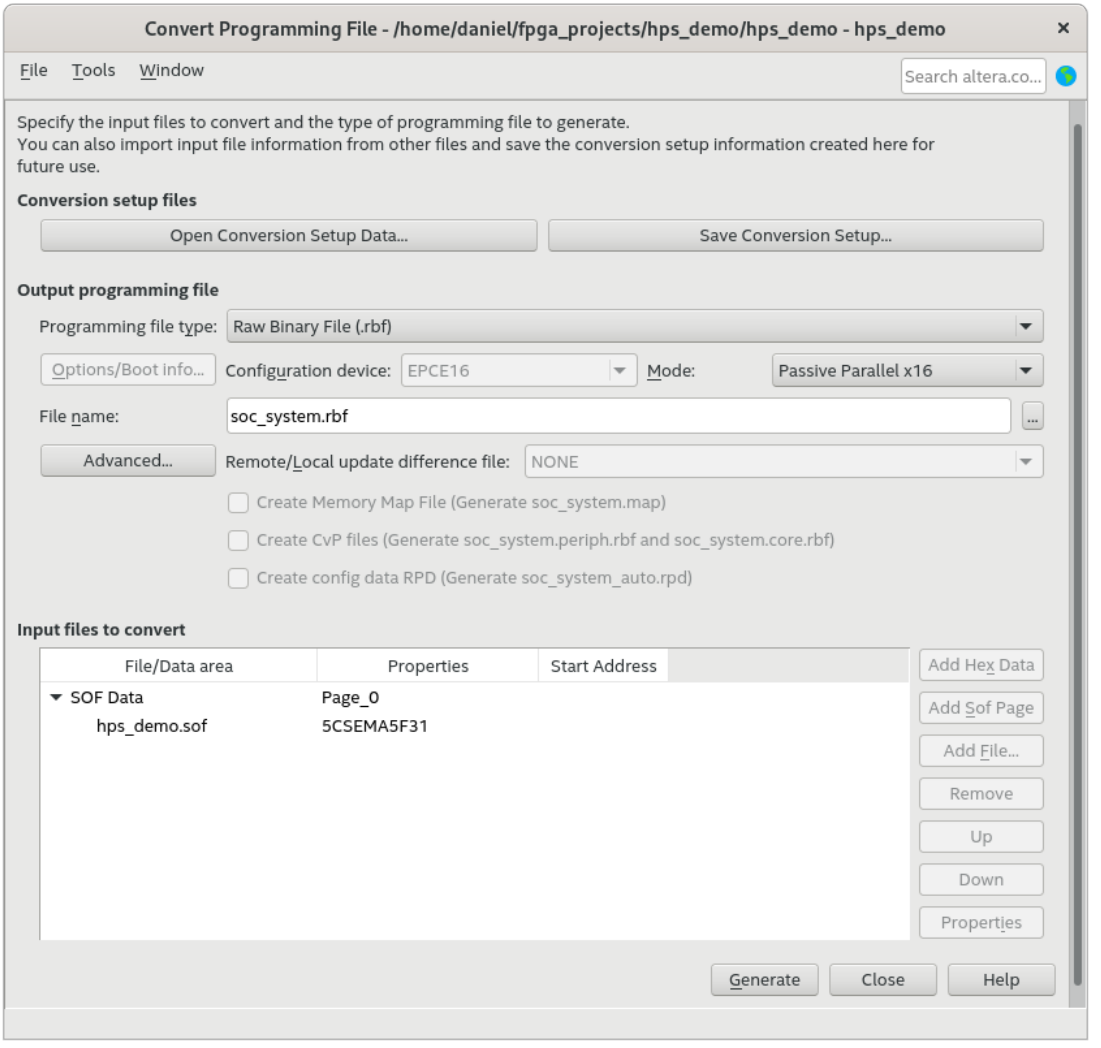
```
sopc2dts -b soc_system_board_info.xml -b ↩
 ↪ hps_common_board_info.xml -c --bridge-removal all --input ↩
 ↪ hps_systeem.sopcinfo --output socfpga.dts
```

Open `socfpga.dts` met een tekstverwerker. We willen het *compatible* veld van onze switches een herkenbare naam geven zodat we later makkelijk een kernel module kunnen koppelen aan de interrupt van ons switches device, zie figuur 10. Verander de naam `"altr,switches"` in `"switches"`. Herhaal dit voor de leds pio en verander het compatible veld in `"leds"`.

Nu genereren we de devicetree blob met het commando:

```
dtc -I dts -O dtb --out socfpga.dtb socfpga.dts
```

[18] Deze bestanden kun je ook vinden het de1_soc_GHRD voorbeeld project op de CD-ROM die je kunt downloaden bij TerasIC.



Figuur 9: Raw Binary File genereren voor U-Boot.

Dit commando zou geen foutmelding moeten geven en het bestand `socfpga.dtb` genereren.

Stap 4: Linux booten met het nieuwe systeem

3.20 Het nieuwe FPGA-bestand `soc_system.rbf` en de nieuwe devicetree blob `socfpga.dtb` moeten op de boot-partitie van de SD-kaart komen te staan. Voordat je de originele

Figuur 10: Label switches definiëren voor kernel module.

bestanden overschrijft is het handige deze eerst een andere naam te geven, zodat je ze later eventueel terug kunt zetten. Dit alles kun je doen door bijvoorbeeld de SD-kaart in je computer te stoppen. Let er hierbij wel op dat Windows 2 van de 3 partities of *schijven* niet kan lezen en je zal vragen om het te formatteren. Doe dit *niet* en klik op Annuleren. De boot-partitie zul je wel kunnen benaderen.

Eventueel kan het ook via Linux op de FPGA. Koppel daarvoor de 1e partitie op de SD-kaart aan `/boot/sdcard_boot` met

```
$ sudo mount /dev/mmcblk0p1 /boot/sdcard_boot/
$ cd /boot/sdcard_boot
```

Geef het originele `socfpga.dtb` en `soc-system.rbf`-bestand op de boot-partitie een andere naam zodat je deze later kunt terugzetten. Zie bijlage A. Kopieer nu de gegenereerde `soc_system.rbf` en `socfpga.dtb` bestanden naar de boot-partitie met behulp van een usb-stick. Stop de USB-stick in de DE1-SOC en mount de stick met

```
$ cd
$ sudo mount /dev/sda1 usb/
```

Onder de map `~/usb` kun je nu de USB-stick benaderen.

3.21 Als de originele bestanden zijn hernoemd en de nieuwe bestanden zijn gekopieerd kun je Linux proberen te starten. Type `sudo reboot now` en kijk of Linux wil booten.

Mocht Linux niet willen opstarten, dan moeten we de originele `socfpga` en `soc_system` bestanden weer terugzetten en Linux opnieuw opstarten.

Nu kun je op zoek gaan naar de fout.

3.22 Kopieer het programma van opdracht 3.8 en maak het werkend op ons nieuwe systeem. Let op dat de adressen voor de PIO-modules nu anders zijn. Je gebruikt nu switch0 in plaats van key0. Zie eventueel de documentatie voor beschikbare registers van de PIO-module.

Voer de volgende opdrachten tijdens de tweede les uit.

Stap 5: Linux kernel module schrijven voor ons systeem

3.23 We hebben een IRQ-signaal gekoppeld aan het `hps_demo` systeem en je gaat nu een kernel module schrijven die reageert op deze interrupt. Dit moet in een kernel module gedaan worden omdat het onder Linux niet mogelijk is om in user space een hardware interrupt af te handelen. Bekijk de voorbeeldcode in `~/CSC10_Development/test_kernel_module/csc_module.c`. De voorbeeldcode gaat ervan uit dat het *compatible* veld van de switches in de devicetree is gedefinieerd als `"switches"` zoals gevraagd in opdracht 3.19. Het voorbeeldproject op de SD-kaart bevat naast het `csc_module.c` bestand ook een `Makefile`.

Gebruik de makefile door `make all` in te typen om de kernel module te compileren. **N.B.** voer dit commando nooit uit met sudo rechten[19].

Als je een waarschuwing krijgt over `clock-skew` dan komt dit omdat de tijd bij elke reboot zich reset. Sommige bestanden kunnen dus een tijd in de toekomst hebben. Een `make clean` voor het compileren zal dit oplossen.

Kernel modules kun je inladen met het commando `insmod` en verwijderen met `rmmod`:

```
#Een kernel module (.ko bestand) laden
$ sudo insmod csc_module.ko

#De geladen kernel modules tonen
$ lsmod

#Een kernel module weghalen
$ sudo rmmod csc_module
```

[19] Voer het commando `sudo pacman -U ~/*.xz` uit als je dit per ongeluk toch gedaan hebt.

Voer nu het commando uit om de kernel module te laden. Als het goed is, stuurt de kernel module telkens een melding met behulp van `printk`[20] als je een van de switches SW0 t/m SW3 hoog maakt. De laatste 5 meldingen kun je bekijken met het commando `dmesg | tail -5`[21]. Als alles naar behoren werkt kun je de kernel module weer verwijderen.

N.B. als na het invoegen van de kernel module niets gebeurt, check dan even of het *compatible*-veld van jouw switches overeenkomt. Zo niet, dan moet je opdrachten 3.19 tot en met 3.21 opnieuw uitvoeren, of je gebruikt onder Linux de commandos `fdtdump`, `fdtget` en `fdtput` om `socfpga.dtb` op de boot-partitie direct te manipuleren. Voorbeelden gebruik:

```
# mount boot-partitie
$ cd ~
$ sudo mount /dev/mmcblk0p1 mnt/
$ cd mnt/
# hele tree bekijken
$ fdtdump socfpga.dtb | less
# enkele waarde bekijken
$ fdtget socfpga.dtb ↩
 ↪ /sopc\@0/bridge\@0xff200000/gpio\@0x100000000 compatible
# enkele waarde manipuleren, type string
$ sudo fdtput --type s socfpga.dtb ↩
 ↪ /sopc\@0/bridge\@0xff200000/gpio\@0x100000000 compatible ↩
 ↪ switches
# unmount boot-partitie
$ cd ..
$ sudo umount mnt/
```

Na het aanpassen van `socfpga.dtb` moet je Linux opnieuw opstarten met het commando `sudo reboot now`.

3.24 Pas nu de kernel module aan zodat ook een melding wordt verstuurd als je een van de switches SW4 t/m SW7 hoog maakt.

3.25 Pas nu de kernel module aan zodat telkens als een switch hoog wordt gemaakt de leds binair optellen. Waarom wordt de teller (bijna) steeds met meer dan één verhoogd?

[20] Zie https://docs.kernel.org/6.6/core-api/printk-basics.html.

[21] Zie https://www.man7.org/linux/man-pages/man1/dmesg.1.html.

Linux deelt programma's op tussen kernel-space en user space. Onze kernel module draait in kernel space en heeft direct toegang tot de hardware en vrijwel alle mogelijke rechten.

User space programma's draaien niet in de kernel maar maken gebruiken van verschillende interface-lagen die de kernel aanbiedt om indirect met hardware te communiceren. Zo'n user space programma kan wel nog steeds root rechten hebben.

De gewenste situatie is dat onze kernel module een extra abstractielaag vormt naar de hardware. In ons geval dus een abstractielaag tussen de PIO-modules van ons FPGA-systeem en user space. Om vanuit user space programma's gebruikt te kunnen maken van onze kernel module zonder superuser rechten, kunnen we onze module registreren als een character device.

In Linux is alles benaderbaar als bestanden. Dit is je misschien al opgevallen toen we de boot-partitie of usb-stick koppelden aan een map op het bestandssysteeem. Door onze driver een character device aan te laten maken wordt er onder de `/dev` map een virtuele file aangemaakt waar een user space programma of terminal naar kan schrijven en lezen. De kernel module ziet deze schrijf- en leesacties en kan hierop reageren.

Je zou dus een character device genaamd `/dev/leds` kunnen koppelen aan de leds en een andere character device `/dev/switches` aan de switches van je systeem. Met als gevolg dat je leds aan kunt sturen door te schrijven naar de 'file' `/dev/leds` en switches uit kunt lezen door te lezen uit de 'file' `/dev/switches`.

De docenten hebben een kernel module geschreven die een character device genaamd `/dev/leds` aanmaakt waar vanuit user space naar toegeschreven kan worden om de leds aan te sturen. Aan jullie de taak om na het bestuderen en uitproberen van deze code zelf een kernel module te schrijven die een character devide genaamd `/dev/switches` aanmaakt waar vanuit user space kan worden gelezen om de toestand van de schakelaars uit te lezen.

3.26 De door de docenten geschreven kernel module kun je vinden in het directory `~/CSC-10_Development/led_driver`. Deze module maakt een character device genaamd `/dev/leds` aan. Compileer en laad deze kernel module:

```
$ cd ~/CSC10_Development/led_driver
$ make
$ sudo insmod led_module.ko
```

Met het commando `ls -l /dev/leds` kun je zien dat het character device is aangemaakt. Je kunt nu een karakter naar het device schrijven met het commando echo en de ASCII-code van dit kararakter zal op de leds verschijnen.

```
# Zet de waarde 15 (0x0F) op de leds:
$ echo -n -e "\x0F" > /dev/leds
```

De optie `-e` van het echo-commando[22] zorgt ervoor dat de escape-sequentie `\x0F` wordt geïnterpreteerd en `-n` zorgt ervoor dat er geen nieuwe regelkarakter wordt toegevoegd.

Vanuit de kernel module worden kernel messages gestuurd om te later zien welke operaties uitgevoerd worden. Je kunt de laatste 10 kernel messages bekijken met het commando `dmesg`[23]:

```
$ dmesg | tail -10
```

Omdat je de leds nu aan kunt sturen door simpelweg naar een bestand te schrijven (zonder superuser rechten), kun je dit ook vanuit een C-programma doen.

In het directory `led_driver` vind je een C-programma genaamd `leds.c` dat laat zien hoe je dit kunt doen:

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    FILE *fp = fopen("/dev/leds", "w");
    if (fp != NULL) {
        int i;
        for (i = 0; i < 17; i++) {
            printf("Send 0x%02X to the leds\n", i);
            fputc(i, fp);
            fflush(fp);
            sleep(1);
        }
        fclose(fp);
    }
    else {
        printf("Error: can not open /dev/leds\n");
    }
```

[22] Zie https://man7.org/linux/man-pages/man1/echo.1.html

[23] Zie https://man7.org/linux/man-pages/man1/dmesg.1.html

```
    return 0;
}
```

De functie `fopen`[24] opent het character device `/dev/leds` als een bestand voor schrijven (`"w"`). Met de functie `fputc`[25] wordt een karakter naar het device geschreven. De functie `fflush`[26] zorgt ervoor dat de buffer wordt geleegd, zodat de waarde direct naar de leds wordt gestuurd. Met de functie `fclose`[27] wordt het bestand weer gesloten. Al de bovenstaande functies zijn gedeclareerd in `stdio.h`. Na het versturen van een karakter wordt steeds 1 seconde gewacht door de functie `sleep`[28] aan te roepen, die gedeclareerd is in `unistd.h`. Compileer en voer het programma uit:

```
$ gcc leds.c -o leds
$ ./leds
```

In het directory `led_driver` vind je ook het C-programma genaamd `leds_try_read.c` waarin geprobeerd wordt om te lezen van het character device `/dev/leds`. Compileer en voer het programma uit:

```
$ gcc leds_try_read.c -o leds_try_read
$ ./leds_try_read
```

Je ziet dat het character device `/dev/leds` niet geopend kan worden om te lezen (`"r"`).

Bestudeer nu de code in `led_module.c`. Het beste kun je onderaan in het bestand beginnen. De macro `module_platform_driver`[29] heb je al eerder gezien in opdracht 3.23. Deze macro registreert de platform driver genaamd `MYLEDDEV` bij de Linux-kernel. De struct `mijn_module_driver` bevat pointers naar functies die worden aangeroepen als de driver wordt geladen (`init_handler`) en verwijderd (`clean_handler`). Deze functies worden aangeroepen door de kernel wanneer een device wordt gevonden in de devicetree die overeenkomt met de driver (op basis van het *compatible* veld).

In de functie `init_handler` wordt het character device `/dev/leds` aangemaakt en wordt het adres van de PIO-module opgeslagen in de globale pointer `PIO_ptr`. In

[24] Zie https://www.tutorialspoint.com/c_standard_library/c_function_fopen.htm.

[25] Zie https://www.tutorialspoint.com/c_standard_library/c_function_fputc.htm.

[26] Zie https://www.tutorialspoint.com/c_standard_library/c_function_fflush.htm.

[27] Zie https://www.tutorialspoint.com/c_standard_library/c_function_fclose.htm.

[28] Zie https://pubs.opengroup.org/onlinepubs/9799919799/functions/sleep.html.

[29] Zie eventueel https://elixir.bootlin.com/linux/v6.6/source/include/linux/platform_device.h#L302.

de functie `clean_handler` wordt het character device weer verwijderd. De functies `my_dev_open`, `my_dev_release` en `my_dev_write` worden aangeroepen als een user space programma het character device opent, sluit, of beschrijft. Deze koppelingen worden gemaakt in de struct `my_dev_fops`[30]. Er zijn nog meer koppelingen mogelijk, zoals lezen (`.read`), maar deze zijn in deze character device niet geïmplementeerd. Een uitgebreide uitleg van de code in `led_module.c` vind je hier: https://github.com/HR-ELEKTRO/csc1/wiki/led-module.

3.27 Schrijf nu een kernel module genaamd `switch_module.c` die een character device genaamd `/dev/switches` aanmaakt. Met behulp van dit character device moet de toestand van de schakelaars SW0 t/m SW7 kunnen worden uitgelezen.

Maak hiervoor een nieuw directory genaamd `switch_driver` en kopieer de makefile uit het directory `led_driver` naar dit nieuwe directory. Pas de gekopieerde makefile aan zodat deze de kernel module `switch_module.c` compileert. Je kunt eventueel bijlage A bekijken voor de benodigde commando's. Een paar opmerkingen:

- Je kunt de class naam `switches` gebruiken.
- De `write` en `read` functies hebben dezelfde prototypen, alleen is het tweede parameter niet **`const`** bij de `read` functie.
- De tegenhanger van `copy_to_user` is `copy_from_user`[31].

Je kunt het commando `hexdump` gebruiken om de hex waarde van de switches af te drukken in de terminal[32]:

```
$ hexdump -n 1 -e '"%02X\n"' /dev/switches
```

De optie `-n 1` geeft aan dat je één karakter (byte) wilt lezen en de optie `-e '"%02Xn"'` zorgt ervoor dat het karakter als hexadecimaal getal wordt weergegeven gevolgd door een nieuwe regel teken.

3.28 In opdracht 3.27 heb je karakters van de character driver `/dev/switches` gelezen met behulp van het commando `hexdump`. Je kunt dit device natuurlijk ook vanuit een C-programma gebruiken (als file) om op een hele eenvoudige manier de switches uit

[30] Deze struct is van het type `file_operations`, zie https://elixir.bootlin.com/linux/v6.6/source/include/linux/fs.h#L1852.

[31] Zie https://elixir.bootlin.com/linux/v6.6.22/source/include/linux/uaccess.h#L180.

[32] Zie https://man7.org/linux/man-pages/man1/hexdump.1.html.

te lezen. Schrijf een C-programma dat gedurende 10 seconden de waarden van de switches elke seconde naar de terminal print. Maak gebruik van het in opdracht 3.27 gemaakte character device. Je kunt één karakter (byte) lezen met de functie `fgetc`[33]. Als het goed is, wordt de waarde die wordt weergegeven aangepast als de schakelaars bediend worden terwijl het programma uitgevoerd wordt.

3.29 Schrijf nu een C-programma waarin een teller op de leds wordt weergegeven die elke seconde met één wordt verhoogd, zolang SW0 hoog is en SW1 laag is. Als SW0 laag is, blijft het programma draaien maar pauzeert de teller tot SW0 weer hoog wordt. Als SW1 hoog is, stopt het programma. Maak gebruik van de in opdracht 3.26 gegeven `/dev/leds` en in opdracht 3.27 gemaakte `/dev/switches` character devices.

De reden waarom we in eerste instantie een kernel module gebruikten, was om de interrupt van onze PIO-module te kunnen registreren. In sommige situaties is het wenselijk om de interrupt ook door te kunnen geven aan een programma dat draait in user space (zonder superuser rechten). Een van de mogelijkheden om dit te realiseren is door gebruik te maken van een character device die asynchronous I/O[34] ondersteunt. Het character device kan dan een POSIX signal[35] sturen naar een user space programma wanneer er een interrupt optreedt.

Het user space programma dat wil reageren op de interrupt moet dan twee dingen doen:

- zijn proces id (PID) registreren bij het character device;
- asynchronous notification inschakelen op het character device;
- een signal handler registreren om het POSIX signal te kunnen opvangen.

In de onderstaande twee opdrachten ga je eerst de kernel module aanpassen om een POSIX signal te kunnen sturen naar een user space programma. Vervolgens ga je een user space programma gebruiken en uitbreiden dat zijn PID registreert, asynchronous notification inschakelt en een signal handler registreert om het POSIX signal op te vangen.

[33] Zie https://www.tutorialspoint.com/c_standard_library/c_function_fgetc.htm.

[34] Zie https://davmac.org/davpage/linux/async-io.html.

[35] Meer informatie over POSIX signals vind je op https://en.wikipedia.org/wiki/Signal_(IPC).

3.30 Je gaat nu stap voor stap een nieuwe kernel module genaamd `async_switch_module.c` schrijven die een character device genaamd `/dev/async_switches` aanmaakt dat asynchronous I/O ondersteunt.

- Maak hiervoor een nieuw directory genaamd `async_switch_driver` en kopieer de makefile en het bestand `switch_module.c` uit het directory `switch_driver` naar dit nieuwe directory. Hernoem het gekopieerde bestand `switch_module.c` naar `async_switch_module.c`. Pas de gekopieerde makefile aan zodat deze de kernel module `async_switch_module.c` compileert.
- Open het bestand `async_switch_module.c` en voeg de volgende includes toe:

```
#include <linux/interrupt.h>
```

- Voeg een globale variabele toe om de IRQ-nummer van de PIO-module te bewaren:

```
static int irq_number;
```

- Voeg een globale variabele toe om de lijst met processen die asynchroon geïnformeerd moeten worden te bewaren:

```
static struct fasync_struct *my_async_queue = NULL;
```

- Voeg een irq handler toe die wordt aangeroepen wanneer een interrupt optreedt. Bekijk indien nodig de voorbeeldcode in `~/CSC10_Development/test_kernel_module/csc_module.c`. In deze irq handler kun je het POSIX signal `SIGIO` sturen naar de user space taken die zich hebben geregistreerd door de functie `kill_fasync()` aan te roepen:

```
    kill_fasync(&my_async_queue, SIGIO, POLL_IN);
```

- In de **`struct`** `file_operations` moet een functiepointer voor `.fasync` worden toegevoegd:

```
    .fasync = my_dev_fasync,
```

- Implementeer de `fasync` functie die de lijst met user space taken bijhoudt die geïnformeerd moeten worden bij een interrupt:

```
static int my_dev_fasync(int fd, struct file *filp, int on)
{
    printk(KERN_INFO DEVNAME ": Device fasync\n");

    int err = fasync_helper(fd, filp, on, &my_async_queue);
    if (err < 0) {
```

```
            printk(KERN_ALERT DEVNAME ": ERROR: fasync_helper ↩
    ↪ failed\n");
            return err;
        }

        static int on_counter = 0;
        if (on) {
            on_counter++;
            if (on_counter == 1) {
            // registreer hier de interrupt met request_irq en
            // enable de interrupt in de PIO module
            }
        }
        else {
            on_counter--;
            if (on_counter == 0) {
            // disable hier de interrupt in de PIO module en
            // geef de interrupt weer vrij met free_irq
            }
        }
        return 0;
    }
```

- In de `init_handler` functie moet het IRQ-nummer van de PIO-module worden opgehaald uit de devicetree en opgeslagen worden in de globale variabele `irq_number`. Bekijk indien nodig de voorbeeldcode in `~/CSC10_Development/-test_kernel_module/csc_module.c`. Vergeet niet de interrupts aan te zetten in de PIO-module zelf.

3.31 Je gaat nu een user space programma gebruiken dat zijn PID registreert, asynchronous notification inschakelt en een signal handler registreert om het POSIX signal op te vangen. Dit programma kun je vinden in `async_switches.c`.

Breid dit programma nu uit zodat bij elke interrupt van een van de acht schakelaars de waarde van de schakelaars op de leds wordt gezet. Maak gebruik van het character device `/dev/leds` uit opdracht 3.26 om de leds aan te sturen.

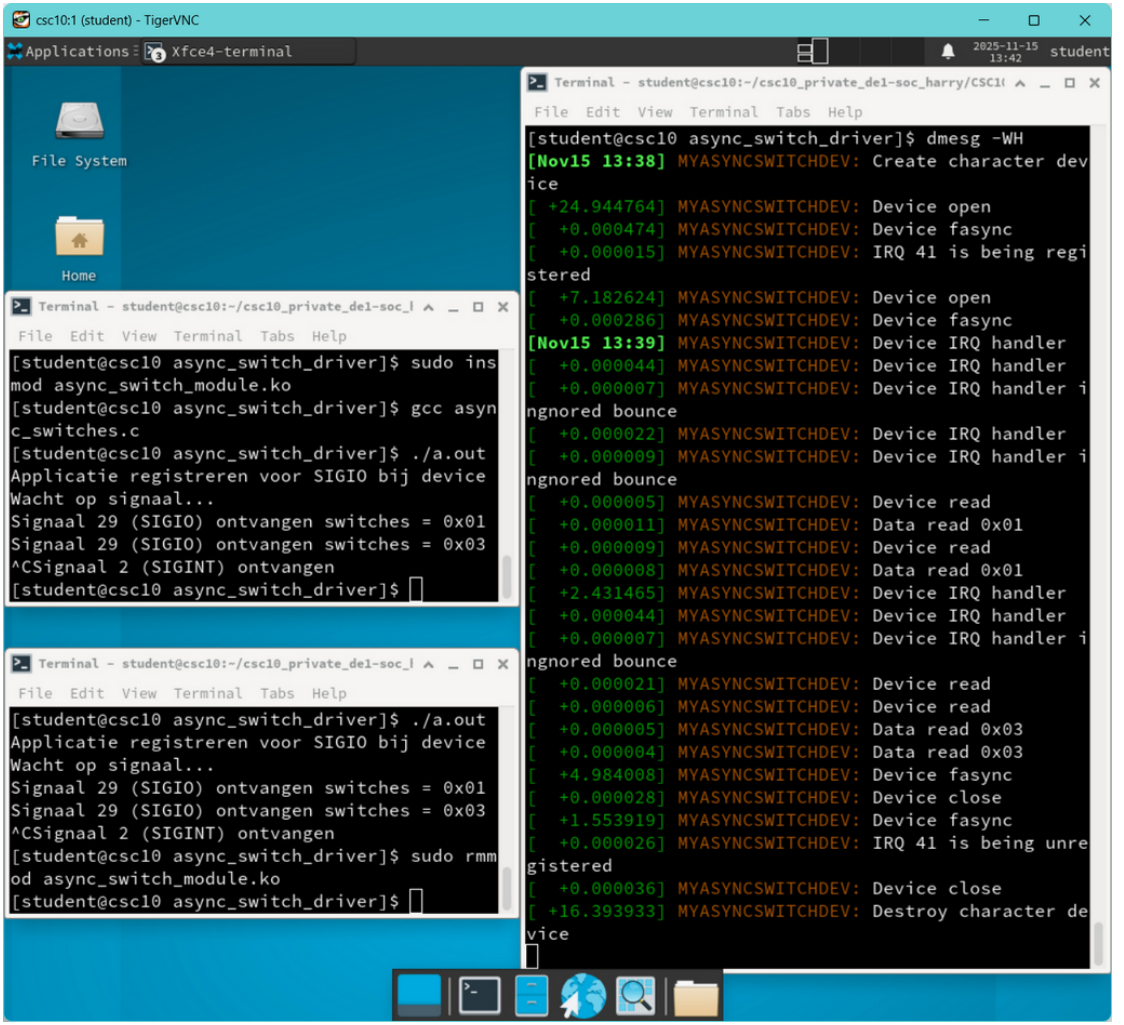
3.32 Deze opdracht is optioneel.

Deze opdracht kun je alleen uitvoeren als je een netwerkverbinding hebt met het bord. Zie bijlagen B en C voor meer informatie. Open meerdere terminals (consoles) op het bord via SSH of Remote Desktop en start in een console het commando `dmesg -WH` om de kernel messages live te volgen. Start het programma `async_switches.c` in een aantal andere consoles. Als je nu de schakelaars bedient zul je, als het goed is in alle consoles waar het programma draait de meldingen zien verschijnen. Zie figuur 11. In dit geval zijn de schakelaars in de kernel module debounced, waardoor bij het hoog maken van de schakelaar slechts één SIGIO signaal wordt gegenereerd.

3.33 Deze opdracht is optioneel en mag ook als (deel van) een eindopdracht gebruikt worden.

Maak nu een nieuw HPS-systeem waarbij de systeemtijd weergegeven wordt op de 7-segment displays. Maak hierbij gebruik van de component die je in week 2 hebt gemaakt om de 7-segment displays aan te sturen. Implementeer een character device waarmee je deze component vanuit user space kunt aansturen. De omzetting van tijd (hh:mm:ss) naar BCD[36] mag je in software doen. Maak gebruik van de hardware timer (periodic timer IP) om elke seconde een interrupt te generen. Implementeer een tweede character device zodat je deze interrupt in user space kunt afhandelen. In een user space applicatie kun je dan bij elk IOSIG signaal de systeemtijd opvragen en schrijven naar de displays. Laat elke 10 seconden ook de datum zien op de 7-segment displays, gedurende 1 seconde.

[36] BCD = Binary-Coded Decimal, zie https://en.wikipedia.org/wiki/Binary-coded_decimal.



Figuur 11: Meerdere instanties van het async_switches programma die gelijktijdig interrupt meldingen ontvangen.

A Veelgebruikte Linux-commando's

Unix/Linux Command Reference

FOSSwire.com

File Commands	System Info
ls – directory listing	**date** – show the current date and time
ls -al – formatted listing with hidden files	**cal** – show this month's calendar
cd *dir* - change directory to *dir*	**uptime** – show current uptime
cd – change to home	**w** – display who is online
pwd – show current directory	**whoami** – who you are logged in as
mkdir *dir* – create a directory *dir*	**finger *user*** – display information about *user*
rm *file* – delete *file*	**uname -a** – show kernel information
rm -r *dir* – delete directory *dir*	**cat /proc/cpuinfo** – cpu information
rm -f *file* – force remove *file*	**cat /proc/meminfo** – memory information
rm -rf *dir* – force remove directory *dir* *	**man *command*** – show the manual for *command*
cp *file1 file2* – copy *file1* to *file2*	**df** – show disk usage

B Secure Shell (SSH)

Mocht je een netwerkverbinding aansluiten op het bord dan is het mogelijk om via het netwerk contact te leggen. Er staat een SSH-server aan op poort 22. Het ip-adres van het bord kun je achterhalen met het commando:.

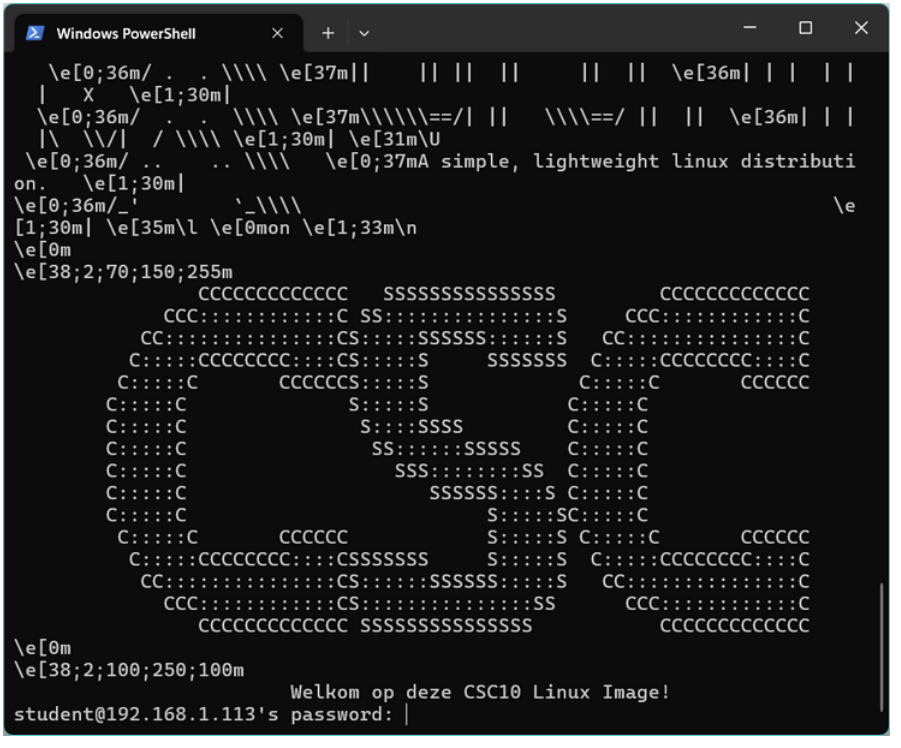
```
$ hostname -i
```

Je kunt nu met een SSH-client op je pc verbinding maken met een remote Linux systeem (op het DE1-SoC bord), zoals uitgelegd in dit filmpje: https://www.youtube.com/watch?v=3hbJZZ4c1io.

Als het ip-adres van het bord bijvoorbeeld 192.168.1.113 is, dan kun je met de het volgende commando in een Power Shell of Command window verbinding maken:

```
C:\Users\Gebruiker> ssh student@192.168.1.113
```

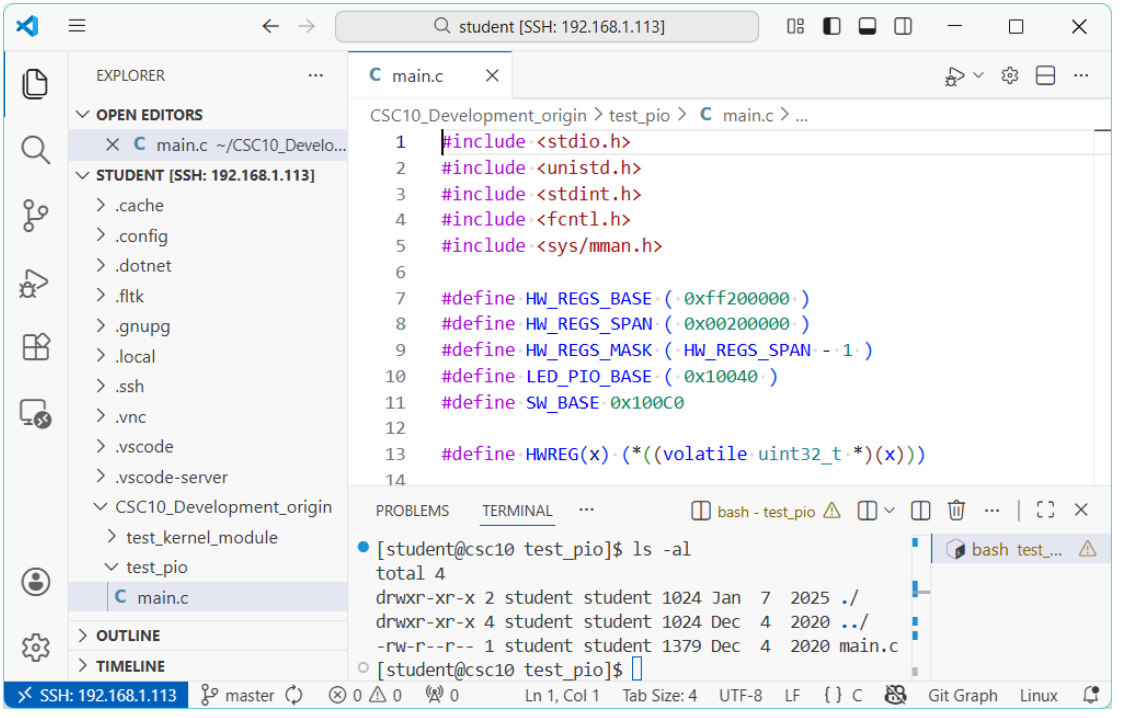
Het standaard wachtwoord is `temppwd`. Zie figuur 12.



Figuur 12: SSH verbinding maken met Power Shell.

Aanbevolen:

Het is ook mogelijk om via VSCode verbinding te maken met het bord. Dan kun je direct in VSCode bestanden bewerken op het bord. Een korte uitleg vind je hier: https://www.databasemart.com/kb/connect-to-linux-server-via-vs-code. Stap 1 kun je vervangen door het bovenstaande filmpje. Zie figuur 13.



Figuur 13: SSH verbinding maken met VSCode.

C Remote Desktop

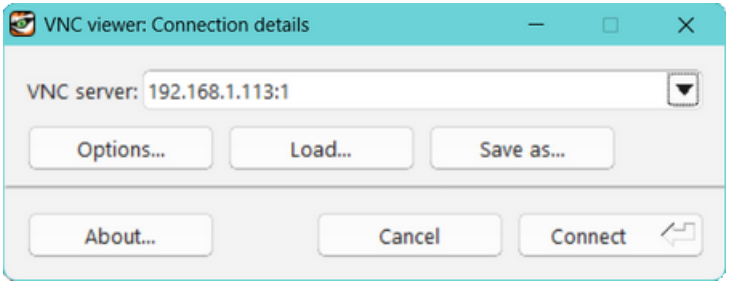
Op de image is ook de desktopomgeving XFCE en de tigervnc-server geïnstalleerd, zie figuur 15. Met de volgende commando's kun je de standaard instelling aanpassen en de server starten. Dit opent een server op poort 5901 met de standaard logingegevens.

```
#Pas eventueel desktop en/of resolutie aan
$ nano ~/.vnc/config

#start vnc server
$ sudo systemctl start vncserver@:1
```

Nadat de server is gestart kun je met een VNC-client op je pc verbinding maken met het bord. Je kunt bijvoorbeeld de TigerVNC viewer gebruiken.

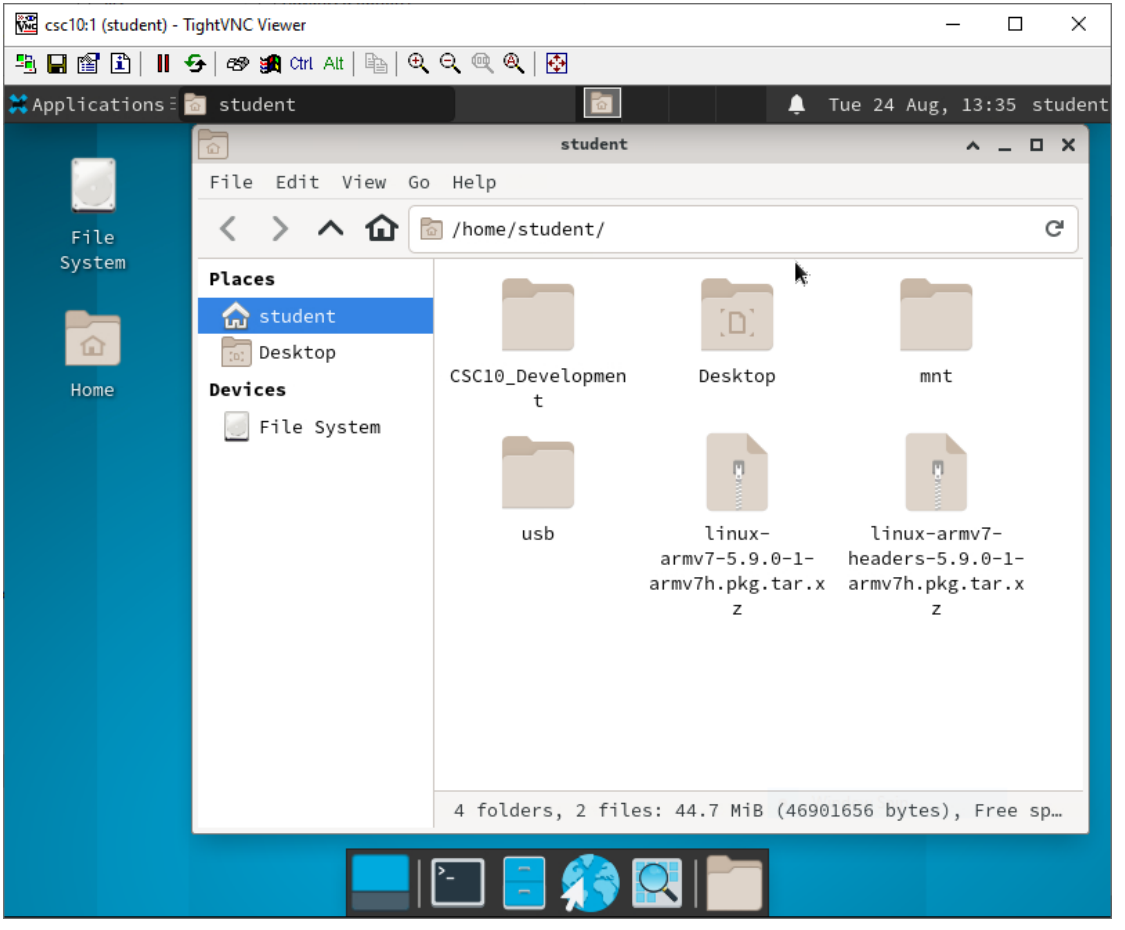
Start de VNC-client en maak verbinding met het IP-adres van het bord, gevolgd door de poort (5901). Je kunt in plaats van het poortnummer ook poortnummer − 5900 (in dit geval dus (5901 − 5900 = 1) gebruiken. Zie figuur 14.



Figuur 14: VNC verbinding maken met TigerVNC viewer.

Mocht je een andere desktop willen gebruiken dan kun je pacman gebruiken om deze te installeren:

```
#Update database en installeer lxqt, vervang eventueel met een ↩
 ↪ desktop naar keuze.
#Let er wel op dat niet elke desktop even 'zuinig' is met de ↩
 ↪ resources.
$ sudo pacman -Sy lxqt
```



Figuur 15: De XFCE omgeving