

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 device tree genereert voor jouw HPS-systeem om Linux te booten;

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 FPGA-schakeling aanstuurt vanuit Linux;
- een simpele Linux kernel module schrijft die de FPGA-schakeling kan aansturen.

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 voorbeeldprogramma *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 [CSC10 Team](#) of zelf downloaden van Intel³. Installeer dit programma.

² https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/Intel_FPGA_Monitor_Program_ARM.pdf

³ Windows of Linux.

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 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 ingesteld⁵ 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 netwerkverbinding⁶ 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.

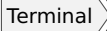
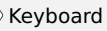

3.5 Zorg ervoor dat je een mini-USB kabel aansluit tussen je computer en de *UART to USB* connector. Open het monitorprogramma PuTTY⁷ op 115200

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 <https://www.chiark.greenend.org.uk/~sgtatham/putty/>

baud naar de juiste COM-poort⁸. We gebruiken PuTTY omdat deze beter om kan gaan met een Linux console dan TeraTerm. Het is hierbij handig om bij de PuTTY instellingen, onder  , 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  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 `temp` als wachtwoord. Je bevind je nu in een zogenoemde bash shell⁹ waarin je Linux-commando's kunt uitvoeren. Mocht 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 ongevolde 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 16 GB. Dit is

⁸ Die vind je in Windows in “Apparaatbeheer” onder “Poorten”.

⁹ 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>.

```

COM5 - PuTTY
5.9.0-csc10 Linux
09:28:18
Tue Aug 24 2021
0 users
ttyS0 on csc10

A simple, lightweight linux distribution.

          CCCCCCCCCCCCCC  SSSSSSSSSSSSSSS  CCCCCCCCCCCCCC
        CCC:::::C SS:::::S  CCC:::::C
       CC:::::CS:::::SSSSS:::::S  CC:::::C
      C:::::CCCCCCCC:::CS:::::S  SSSSSSS  C:::::CCCCCCCC:::C
     C:::::C  CCCCCCS:::::S  C:::::C  CCCCC
    C:::::C      S:::::S  C:::::C
   C:::::C      S:::::SSSS  C:::::C
  C:::::C      SS:::::SSSS  C:::::C
 C:::::C      SSS:::::SS  C:::::C
C:::::C      SSSSS:::::S  C:::::C
C:::::C      S:::::SC:::::C
C:::::C  CCCCC  S:::::S  C:::::C  CCCCC
C:::::CCCCCCCC:::CSSSSSSS  S:::::S  C:::::CCCCCCCC:::C
  CC:::::CS:::::SSSSS:::::S  CC:::::C
  CCC:::::CS:::::SS  CCC:::::C
  CCCCCCCCCCCCCC  SSSSSSSSSSSSSSS  CCCCCCCCCCCCCC

      Welkom op deze CSC10 Linux Image!

  Standaard username:password is [student:tempPWD]

csc10 login: █

```

Figuur 1: Het scherm na opstarten van Linux.

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`¹⁰.

Op regel 22 wordt `/dev/mem` geopend. Op regel 29 wordt het fysiek geheugen-bereik van de LightWeight AXI-bus gekoppeld aan het virtuele geheugen, en een pointer hiernaar wordt teruggegeven.

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 RTS10. De docent heeft voor het gemak de `HWREG(x)` macro toegevoegd.

Sluit de editor (`Ctrl` + `X` voor nano). Compileer de code met bijv. `gcc main.c -o main`. Voer de code uit met `sudo ./main`. Druk KEY0 in en de leds (led1 t/m led8) zullen binair optellen. Als al deze leds branden,

¹⁰ De optie `-l` zorgt ervoor dat je de regelnummers in de kantlijn ziet.

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

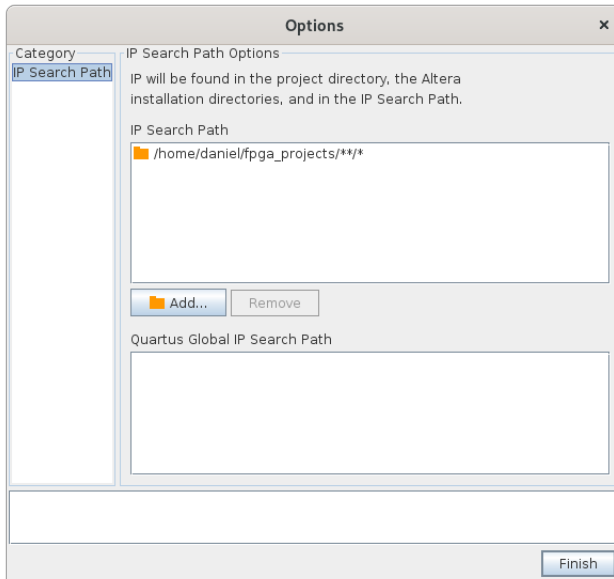
Stap 1: Configureer systeem in Platform Designer

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_CSC10.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 "Aria V/Cyclone V Hard Processor System". Druk op **Add**. Als er een configuratie window open kan je op **Finish** drukken. Zorg ervoor dat de preset window aan staat (**View** > **Presets**) en met hps_0 geselecteerd, **Apply** de preset genaamd *DE1-SoC bordje CSC10*. Nu rest ons om de HPS verder te configureren.

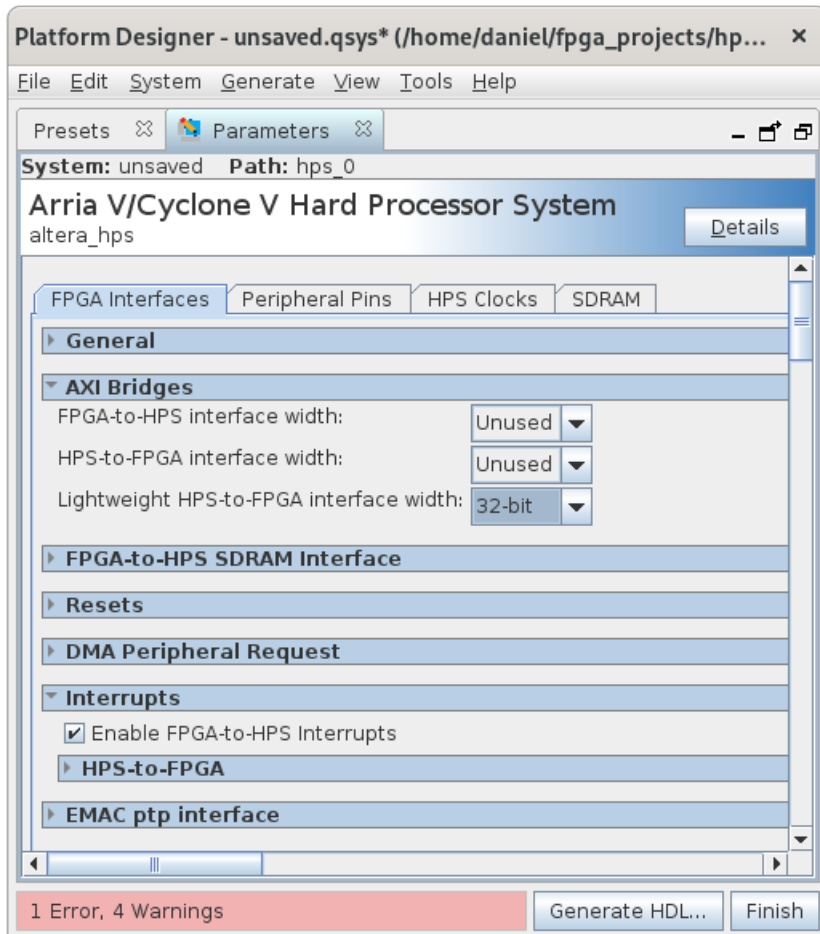


Figuur 2: Ip search path toevoegen binnen Platform Designer

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 weergegeven 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.

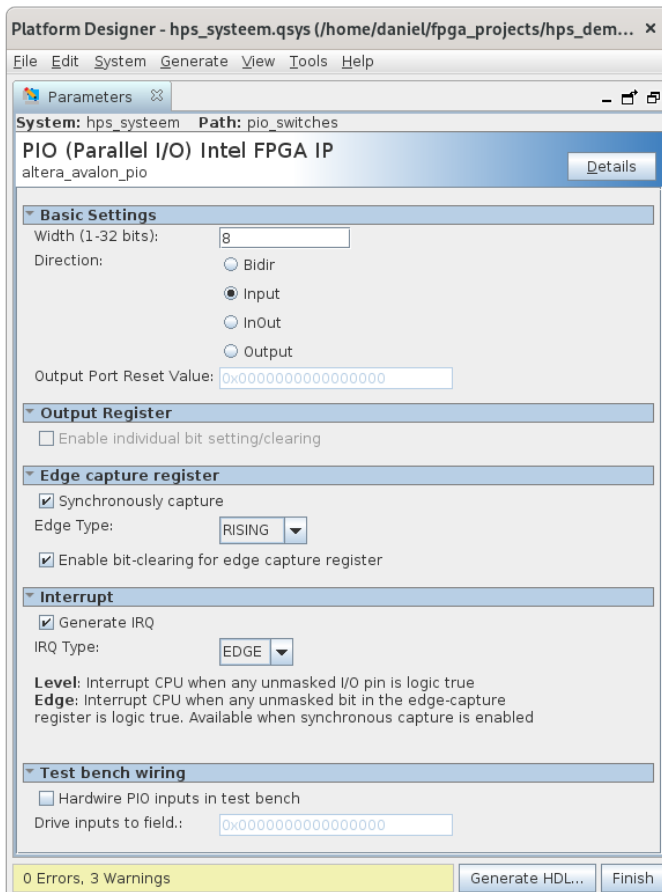
N.B. elke instellingswijziging voor de HPS duurt een paar seconden. Wacht dus even na het klikken voordat je verder gaat.



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.

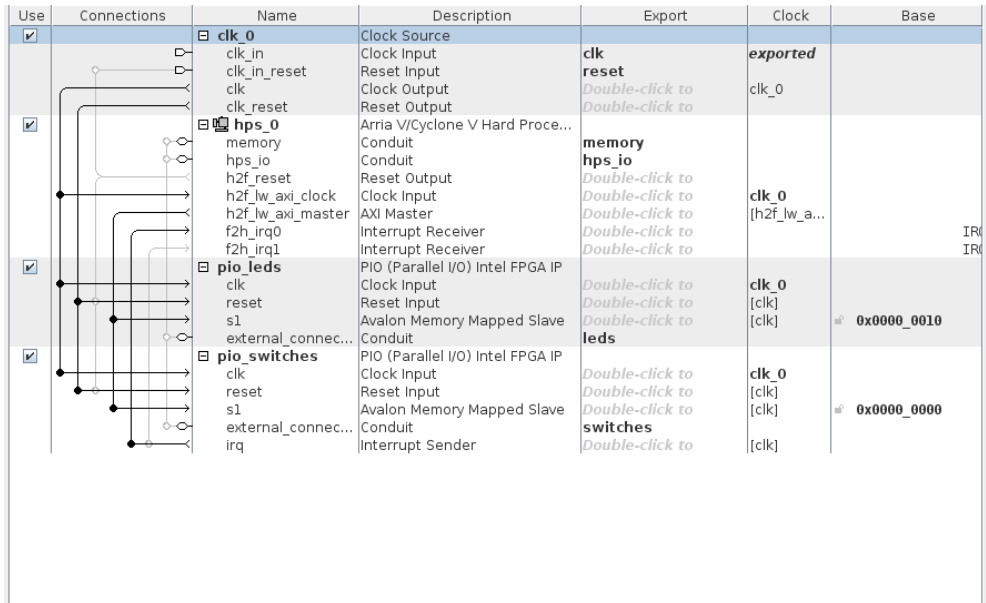
3.13 Sluit de HPS-configuratie en voeg een PIO-module toe aan het systeem. We gaan deze PIO gebruiken om led0 t/m led7 mee aan te sturen. Gebruik de standaard instellingen (8 bits uitgang). Voeg een tweede PIO-module toe met de instellingen van [figuur 4](#). We gaan deze PIO gebruiken om switch0 t/m switch7 mee in te lezen.

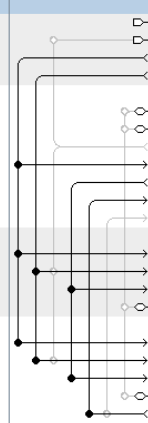


Figuur 4: Instellingen van de tweede PIO-module

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 Addresses`.

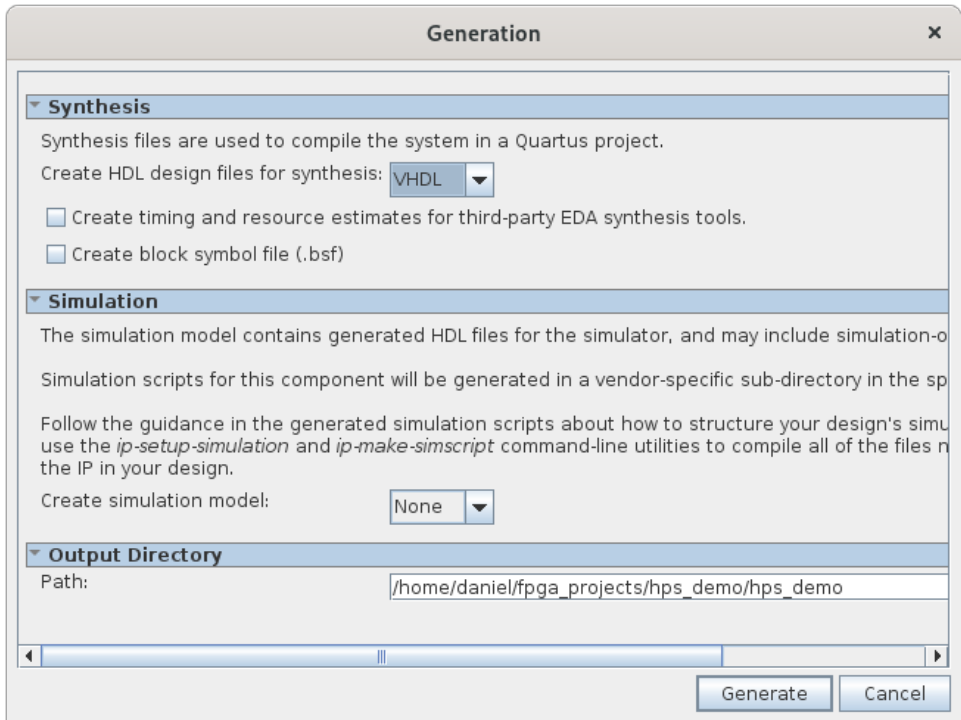


Use	Connections	Name	Description	Export	Clock	Base	
<input checked="" type="checkbox"/>		clk_0	Clock Source				
		clk_in	Clock Input				
		clk_in_reset	Reset Input				
		clk	Clock Output		<code>clk</code>	<code>exported</code>	
		clk_reset	Reset Output		<code>reset</code>	<code>clk_0</code>	
<input checked="" type="checkbox"/>		hps_0	Arria V/Cyclone V Hard Proce...				
		memory	Conduit		<code>memory</code>		
		hps_io	Conduit		<code>hps_io</code>		
		h2f_reset	Reset Output				
		h2f_lw_axi_clock	Clock Input			<code>clk_0</code>	
		h2f_lw_axi_master	AXI Master			<code>[h2f_lw_a...</code>	
		f2h_irq0	Interrupt Receiver				IRQ
		f2h_irq1	Interrupt Receiver				IRQ
<input checked="" type="checkbox"/>		pio_leds	PIO (Parallel I/O) Intel FPGA IP				
		clk	Clock Input		<code>clk</code>	<code>clk_0</code>	
	reset	Reset Input		<code>reset</code>	<code>[clk]</code>	<code>0x0000_0010</code>	
	s1	Avalon Memory Mapped Slave					
	external_connec...	Conduit		<code>leds</code>			
<input checked="" type="checkbox"/>	pio_switches	PIO (Parallel I/O) Intel FPGA IP					
	clk	Clock Input		<code>clk</code>	<code>clk_0</code>		
	reset	Reset Input		<code>reset</code>	<code>[clk]</code>	<code>0x0000_0000</code>	
	s1	Avalon Memory Mapped Slave					
	external_connec...	Conduit		<code>switches</code>			
	irq	Interrupt Sender			<code>[clk]</code>		

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 druk op `Generate`.

Voordat we Platform Designer sluiten is het handig om de *Component Instantiation Template* te kopiëren. Ga naar `Generate` > `Show Instantiation 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.



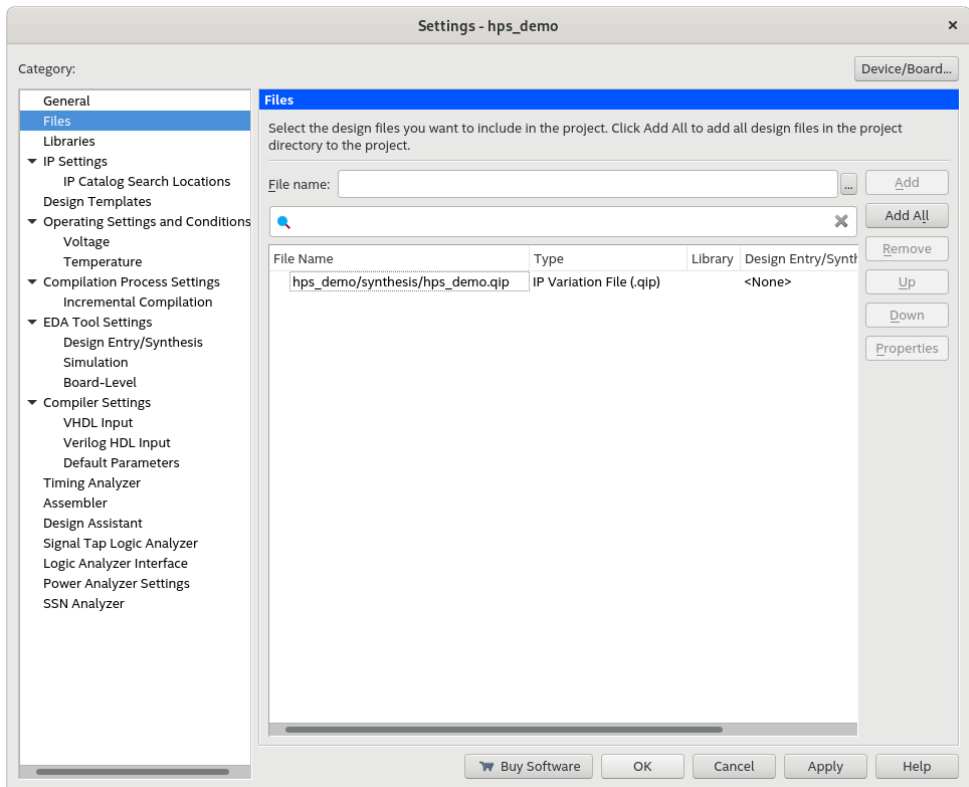
Figuur 6: Instellingen HPS configuratie.

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 ¹¹.

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

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`

¹¹ Kijk in je logboek als je niet meer weet hoe dat moet.



Figuur 7: qip Bestand toevoegen.

plakken we de eerder gekopieerde tekst. Vergeet niet BEGIN te plaatsen tussen de component en de port map van de component.

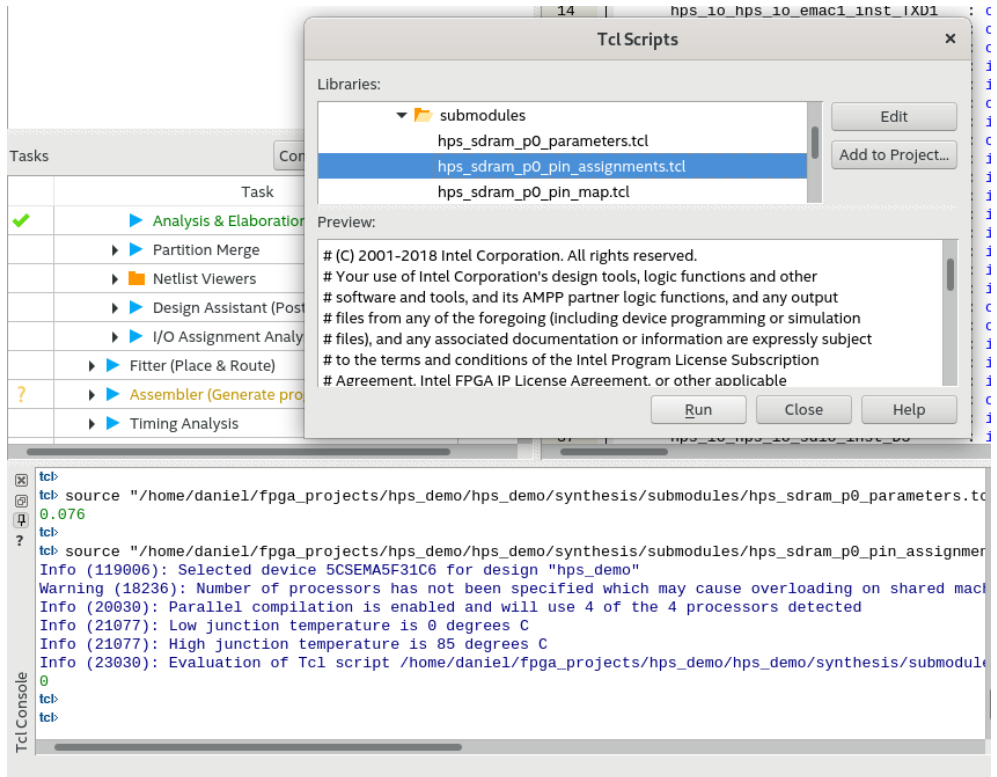
Kopieer de PORT()-beschrijving van de hps_systeem component en gebruik deze voor de hps_demo entity. Verander clk_clk, reset_reset_n, leds_export en switches_export naar de betreffende namen uit het geïmporteerde .qsf-bestand.

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.

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` \gg `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.



Figuur 8: TCL-scripts uitvoeren.

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

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 bootloader¹².
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 Device Tree Blob (dtb) en het laat Linux weten hoe dit platform is opgebouwd en waar welke hardware zit¹³.
4. Linux start en initialiseert alle hardware om het OS te kunnen uitvoeren.

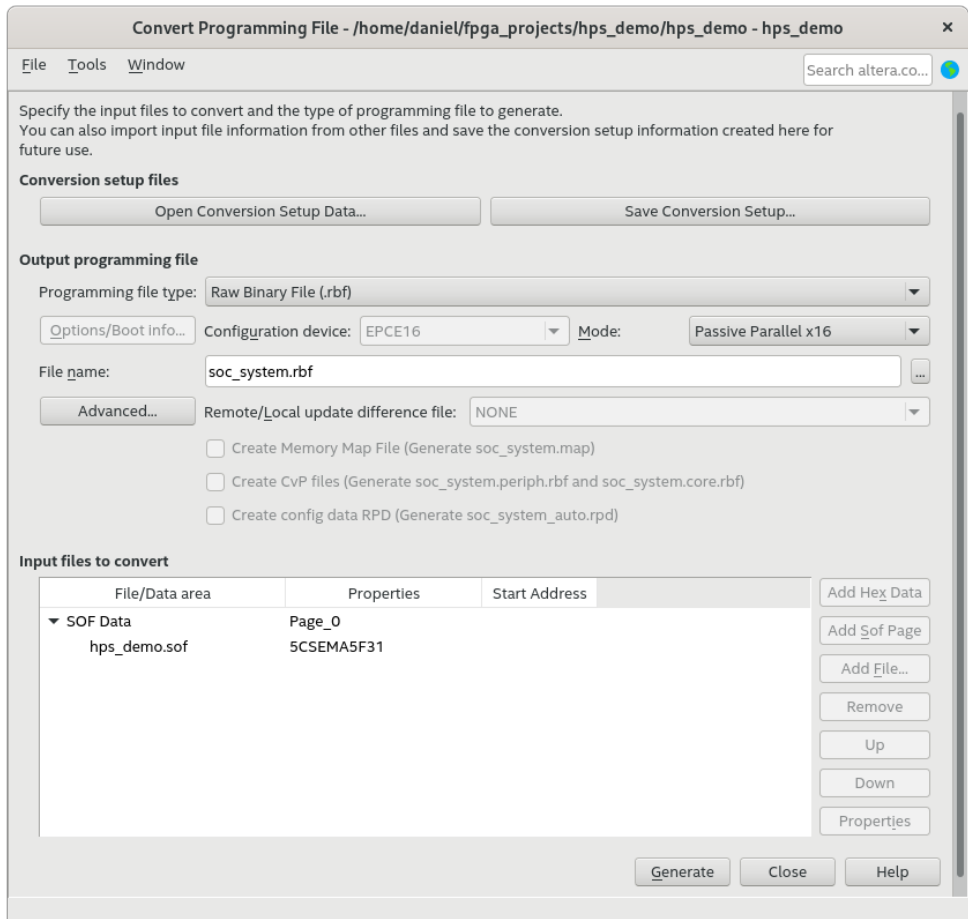
Tot nu toe hebben we enkel een `.sof`-bestand vanuit Quartus gekregen. Dat is het bestand dat je in eerdere weken in de FPGA hebt geladen via het Monitorprogramma of Quartus. U-Boot wil een binaire versie van het `.sof`-bestand dat het zonder problemen direct kan schrijven naar de FPGA. De binaire versie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

¹² https://en.wikipedia.org/wiki/Das_U-Boot

¹³ https://en.wikipedia.org/wiki/Device_tree#Usage_in_Linux



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

Stap 3: Device Tree Blob genereren (socfpga.dtb)

3.19 Download [hps_common_board_info.xml](#) en [soc_system_board_info.xml](#)¹⁴ en kopieer deze bestanden naar jouw projectmap.

¹⁴ Deze bestanden kun je ook vinden het de1_soc_GHRD voorbeeld project op de CD-ROM die je kunt [downloaden](#) bij TerasIC.

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 `socfpga.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](#). Doe dit.



```
109         gpio-controller;
110     }; //end gpio@0x100000010 (pio_leds)
111
112     pio_switches: gpio@0x100000000 {
113         compatible = "altr,switches";
114         reg = <0x00000001 0x00000000 0x00000010>;
115         interrupt-parent = <&hps_0_arm_gic_0>;
116         interrupts = <0 40 1>;
117         clocks = <&clk_0>;
118         altr, gpio-bank-width = <8>; /* embeddedsw.dts.params.altr, gpio-bank-
```

Figuur 10: Label switches definiëren voor kernel module.

Nu genereren we de devicetree-blob met het commando:

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

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 device-tree-blob `socfpga.dtb` moeten op de boot-partitie van de SD-kaart komen te staan.

Voordat je de originele 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 . 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 driver schrijven voor ons systeem

3.23 Omdat we ook een IRQ-sigitaal hebben gekoppeld aan het `hps_demo` systeem, kun je 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 device tree is gedefinieerd zoals in [figuur 10](#). Het voorbeeldproject op de SD-kaart bevat een `makefile` en een `csc_module.c` bestand.

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.

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
$lsmmod

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

Voer nu het commando uit om de kernel module te laden. Als het goed is, stuurt de kernel module telkens een melding naar het console als je een van de switches SW0 t/m SW3 hoog maakt.

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 knoppen
```

Als alles naar behoren werkt kun je de kernel module weer verwijderen.

3.24 Pas nu de kernel module aan zodat ook een melding naar het console 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?

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.

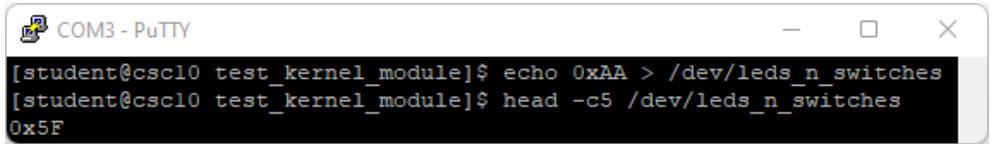
User-space programma's draaien niet in de kernel maar maken gebruik van verschillende interface-lagen die de kernel aanbiedt om indirect met hardware te communiceren. Zo'n user-space programma kan wel nog steeds root rechten hebben.

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 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 bestandssysteem. Door onze driver een character device aan te laten maken wordt er onder de /dev map een 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 kunnen koppelen aan de leds en switches van je systeem. Met als gevolg dat je leds aan kunt sturen door te schrijven naar deze 'file' en switches uit kunt lezen door te lezen uit deze 'file'. Zie [figuur 11](#). Nadat je onderstaande opdracht ([opdracht 3.26](#)) hebben uitgevoerd, kun je bijvoorbeeld met het commando `echo de waarde 0xAA naar de leds sturen door naar het bestand /dev/leds_n_switches te schrijven`. Als je met het commando `head de file /dev/leds_n_switches uitleest`, dan lees je de stand van de switches uit.

De voorbeeldcode die is gegeven voor de kernel module roept onderaan `module_platform_driver()` aan. Dit is een macro die onder water onder andere `module_init()` aanroept. Het begin van onze module is de `init_handler()` functie. Dit wordt eenmalig uitgevoerd en dit is dus de plek waar we alles instellen en ook het character device zullen registreren.



```
COM3 - PuTTY
[student@csc10 test_kernel_module]$ echo 0xAA > /dev/leds_n_switches
[student@csc10 test_kernel_module]$ head -c5 /dev/leds_n_switches
0x5F
```

Figuur 11: Resultaat van opdracht 3.26.

3.26 Deze [blog-post](#) laat zien hoe je een character device kunt aanmaken in een kernel module. Gebruik de uitleg en voorbeeldcode van de blogpost om zelf een device te laten registreren met de naam `leds_n_switches`. Vanuit user-space moet je via het device de leds kunnen aansturen als je ernaar toe schrijft en de schakelaarstanden kunnen uitlezen als je er van leest, zie [figuur 11](#).

Er zijn een aantal tips en aandachtspunten:

- De init en exit functies van de kernel module heten in de blog anders dan in de `csc_module.c` voorbeeldcode, zet de inhoud van de init en exit functie uit de blog dus in de bestaande functies van de voorbeeldcode.
- In de blog worden twee minor device nummers gebruikt. Jij gebruikt maar één minor device nummer. Het is dus niet nodig om `-0` aan de naam toe te voegen.
- Een character device moet, zoals de naam al aangeeft, karakters ontvangen. Met het commando `echo`, zie [figuur 11](#), wordt bijvoorbeeld de karakterstring `"0xAA"` naar de character device verstuurd. De device ontvangt deze karakterstring in de functie `mychardev_write`. Daar moet deze karakterstring omgezet worden naar een integer zodat deze waarde naar de leds gestuurd kan worden. Je kunt daarbij handig gebruik maken van de functie `sscanf`¹⁵.
- Een character device moet, zoals de naam al aangeeft, karakters versturen. Met het commando `head -c5`, zie [figuur 11](#), worden bijvoorbeeld vijf karakters uit de character device gelezen. De device verstuurd

¹⁵ Zie https://www.tutorialspoint.com/c_standard_library/c_function_sscanf.htm.

deze karakterstring in de functie `mychardev_read`. Daar moet dus de integer waarde die vanaf de schakelaars wordt ingelezen (bijvoorbeeld 17) omgezet worden naar een karakterstring die de hexadecimale waarde weergeeft (bijvoorbeeld "`0x11`"). Je kunt daarbij handig gebruik maken van de functie `sprintf`¹⁶.

3.27 In de vorige opdracht heb je karakters naar het character device `leds_n_switches` gestuurd met behulp van het commando `echo`. Je kunt dit device natuurlijk ook vanuit een C-programma gebruiken (als file) om op een hele eenvoudige manier de leds aan te sturen. Je kunt daarbij gebruik maken van de functies `fopen`¹⁷, `fclose`¹⁸ en `fprintf`¹⁹ die gedeclareerd zijn in `stdio.h`.

Schrijf een C-programma dat de waarden 0 t/m 15 naar de leds stuurt en elke waarde 1 seconde laat staan. Maak gebruik van het in [opdracht 3.26](#) gemaakte character device. Om 1 seconde te wachten kun je de functie `sleep`²⁰ gebruiken, die gedeclareerd is in `unistd.h`.

3.28 In [opdracht 3.26](#) heb je karakters van de character driver `leds_n_switches` gelezen met behulp van het commando `head`. Je kunt dit device natuurlijk ook vanuit een C-programma gebruiken (als file) om op een hele eenvoudige manier de switches uit te lezen. **Let op!** Deze opdracht is bedoeld om je te leren waarom iets *niet werkt!* Je kunt daarbij gebruik maken van de functies `fopen`, `fclose` en `fscanf`²¹ die gedeclareerd zijn in `stdio.h`.

¹⁶ Zie https://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm.

¹⁷ Zie https://www.tutorialspoint.com/c_standard_library/c_function_fopen.htm.

¹⁸ Zie https://www.tutorialspoint.com/c_standard_library/c_function_fclose.htm.

¹⁹ Zie https://www.tutorialspoint.com/c_standard_library/c_function_fprintf.htm.

²⁰ Zie <https://pubs.opengroup.org/onlinepubs/9699919799/functions/sleep.html>.

²¹ Zie https://www.tutorialspoint.com/c_standard_library/c_function_fscanf.htm.

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 3.26](#) gemaakte character device. **Je ziet dat dit *niet* werkt!** De waarde van de switches wordt in eerste instantie wel correct weergegeven, maar als je de switches bedient terwijl het programma draait, dan wordt de weergegeven waarde niet aangepast! Dit komt omdat de stdio-library buffering toepast en bij de eerste de beste fscanf meteen 4096 karakters inleest. De stand van de schakelaars op dat moment wordt dus vele malen ingelezen en in het buffer geplaatst. Als we vervolgens na elke seconde opnieuw een fscanf uitvoeren, dan lezen we steeds (de oude waarde) uit het buffer. Dit zou je op kunnen lossen door het buffer, dat gebruikt wordt om het character device uit te lezen, aan te passen en exact groot genoeg te maken om de switches één keer uit te kunnen lezen. Maar dit lijkt meer op hacken dan op programmeren²².

Het is beter om in dit geval de stdio-library niet te gebruiken maar het character device uit te lezen met behulp van de onderliggende system calls. Je kunt daarbij gebruik maken van de functies open²³, close²⁴ en read²⁵. Pas het C-programma dat gedurende 10 seconden de waarden van de switches elke seconde naar de terminal print nu aan, zodat het gebruik maakt van bovenstaande system calls. Als het goed is, wordt de waarde die wordt weergegeven nu wel 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

²² Voor de echte hackers: dit kun je doen m.b.v. de functie setvbuf, zie https://www.tutorialspoint.com/c_standard_library/c_function_setvbuf.htm.

²³ Zie <https://pubs.opengroup.org/onlinepubs/9699919799/functions/open.html>.

²⁴ Zie <https://pubs.opengroup.org/onlinepubs/9699919799/functions/close.html>.

²⁵ Zie <https://pubs.opengroup.org/onlinepubs/9699919799/functions/read.html>.

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.

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 user-space. Om dit voor elkaar te krijgen moeten we aan de kernel module de PID (Process ID) van het user-space programma doorgeven.

De twee stappen zijn:

- Registreer de user-space applicatie bij de kernel module.
- Stuur een signaal naar user-space applicaties in geval van de hardware interrupt.

3.30 Het doorgeven van de user-space PID ga je doen met `ioctl`. Een van de functiepointers die je in [opdracht 3.26](#) bent tegengekomen was `unlocked_ioctl`. Deze functie wordt gebruikt wanneer een user-space applicatie de `ioctl`-functie aanroept. De functie `ioctl` is een groot beest om te gebruiken²⁶, wij zullen het minimaal gebruiken en gaan ervan uit dat slechts één user-space applicatie het signaal wil ontvangen. Als eerste configureren we de kernel module om een PID te kunnen ontvangen en signalen te kunnen versturen; hiervoor moeten een aantal dingen worden aangepast:

- Maak een globale variabele aan om de user-space taak/applicatie te kunnen bewaren:

```
static struct task_struct *taak = NULL;
```

- De definitie van de `unlocked_ioctl` functie is:

```
static long mijn_ioctl_functie(struct file *file, ↵  
    ↵ unsigned int cmd, unsigned long arg)
```

²⁶ Zie <https://www.kernel.org/doc/html/latest/driver-api/ioctl.html>.

We kunnen `cmd` gebruik om te zien welk commando de user-space applicatie heeft gebruikt. De `_IOW` macro geeft aan dat het om user naar kernel communicatie gaat.

```
if (cmd == _IOW('a','a',int32_t*)) {
    printk(KERN_INFO "Taak registreren\n");
    //huidige taak die registratie aanvraagt
    taak = get_current();
}
return 0;
```

- Binnen onze hardware interrupt wil je een signaal doorgeven naar de taak die het wil horen. Dit doe je door een `siginfo` structure te vullen en door te sturen met `send_sig_info()`:

```
struct kernel_siginfo info;

//info 0 maken
memset(&info, 0, sizeof(struct kernel_siginfo));

//signaal instellen
info.si_signo = 10; //nummer 10 voor CSC10 :)
info.si_code = SI_QUEUE;
// zie sigqueue in Linux man pages
info.si_int = 1;
//sturen een 1.

if (taak != NULL) {
    printk(KERN_INFO "Signaal sturen naar ↔
↔ applicatie\n");
    if (send_sig_info(10, &info, taak) < 0) {
        printk(KERN_INFO "Signaal niet kunnen ↔
↔ sturen\n");
    }
}
```

- Als laatste, wanneer de user-applicatie klaar is geeft hij het bestand vrij. Bij deze *release* file operatie willen we de taak uit de lijst halen.

```
struct task_struct *ref_taat = get_current();
printk(KERN_INFO "Device bestand vrijgegeven\n");

//Taat 0 maken
if(ref_taat == taak) {
    taak = NULL;
}
return 0;
```

Nu de kernel kant is geprogrammeerd kunnen we ook kijken naar de user-applicatie kant. Gebruik `siginfo_user.c` voor de user kant.

Implementeer met bovenstaande informatie een user-space programma dat bij elke interrupt de corresponderende led aanzet. Dit geldt voor alle acht switches. Dus een interrupt van switch 5 moet alleen led 5 aanzetten. Gebruik voor het schakelen van de leds hetzelfde character device als van [opdracht 3.26](#). Let op dat in het user-space programma niet `fopen()` is gebruikt, maar `open()`. Gebruik dus ook de corresponderende `write()`²⁷ om naar het character device te schrijven vanuit jouw user-space programma.

3.31 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. De omzetting van tijd (hh:mm:ss) naar BCD²⁸ mag je in software doen.

Maak gebruik van de hardware timer (periodic timer IP) om elke seconde een interrupt te generen. In een kernel module koppel je die interrupt zodat je in de ISR de systeemtijd kan opvragen en schrijven naar de displays. Laat

²⁷ <https://pubs.opengroup.org/onlinepubs/9699919799/functions/write.html>

²⁸ BCD = Binary-Coded Decimal, zie https://en.wikipedia.org/wiki/Binary-coded_decimal.

elke 10 seconden ook de datum zien op de 7-segment displays, gedurende 1 seconde.

A Veelgebruikte Linux-commando's

Unix/Linux Command Reference

FOSSwire.com

File Commands	System Info
<p>ls - directory listing</p> <p>ls -al - formatted listing with hidden files</p> <p>cd <i>dir</i> - change directory to <i>dir</i></p> <p>cd - change to home</p> <p>pwd - show current directory</p> <p>mkdir <i>dir</i> - create a directory <i>dir</i></p> <p>rm <i>file</i> - delete <i>file</i></p> <p>rm -r <i>dir</i> - delete directory <i>dir</i></p> <p>rm -f <i>file</i> - force remove <i>file</i></p> <p>rm -rf <i>dir</i> - force remove directory <i>dir</i>*</p> <p>cp <i>file1 file2</i> - copy <i>file1</i> to <i>file2</i></p> <p>cp -r <i>dir1 dir2</i> - copy <i>dir1</i> to <i>dir2</i>; create <i>dir2</i> if it doesn't exist</p> <p>mv <i>file1 file2</i> - rename or move <i>file1</i> to <i>file2</i> if <i>file2</i> is an existing directory, moves <i>file1</i> into directory <i>file2</i></p> <p>ln -s <i>file link</i> - create symbolic link <i>link</i> to <i>file</i></p> <p>touch <i>file</i> - create or update <i>file</i></p> <p>cat > <i>file</i> - places standard input into <i>file</i></p> <p>more <i>file</i> - output the contents of <i>file</i></p> <p>head <i>file</i> - output the first 10 lines of <i>file</i></p> <p>tail <i>file</i> - output the last 10 lines of <i>file</i></p> <p>tail -f <i>file</i> - output the contents of <i>file</i> as it grows, starting with the last 10 lines</p>	<p>date - show the current date and time</p> <p>cal - show this month's calendar</p> <p>uptime - show current uptime</p> <p>w - display who is online</p> <p>whoami - who you are logged in as</p> <p>finger <i>user</i> - display information about <i>user</i></p> <p>uname -a - show kernel information</p> <p>cat /proc/cpuinfo - cpu information</p> <p>cat /proc/meminfo - memory information</p> <p>man <i>command</i> - show the manual for <i>command</i></p> <p>df - show disk usage</p> <p>du - show directory space usage</p> <p>free - show memory and swap usage</p> <p>whereis <i>app</i> - show possible locations of <i>app</i></p> <p>which <i>app</i> - show which <i>app</i> will be run by default</p>
Process Management	Compression
<p>ps - display your currently active processes</p> <p>top - display all running processes</p> <p>kill <i>pid</i> - kill process id <i>pid</i></p> <p>killall <i>proc</i> - kill all processes named <i>proc</i>*</p> <p>bg - lists stopped or background jobs; resume a stopped job in the background</p> <p>fg - brings the most recent job to foreground</p> <p>fg <i>n</i> - brings job <i>n</i> to the foreground</p>	<p>tar cf <i>file.tar files</i> - create a tar named <i>file.tar</i> containing <i>files</i></p> <p>tar xf <i>file.tar</i> - extract the files from <i>file.tar</i></p> <p>tar czf <i>file.tar.gz files</i> - create a tar with Gzip compression</p> <p>tar xzf <i>file.tar.gz</i> - extract a tar using Gzip</p> <p>tar cjf <i>file.tar.bz2</i> - create a tar with Bzip2 compression</p> <p>tar xjf <i>file.tar.bz2</i> - extract a tar using Bzip2</p> <p>gzip <i>file</i> - compresses <i>file</i> and renames it to <i>file.gz</i></p> <p>gzip -d <i>file.gz</i> - decompresses <i>file.gz</i> back to <i>file</i></p>
File Permissions	Network
<p>chmod <i>octal file</i> - change the permissions of <i>file</i> to <i>octal</i>, which can be found separately for user, group, and world by adding:</p> <ul style="list-style-type: none"> • 4 - read (r) • 2 - write (w) • 1 - execute (x) <p>Examples:</p> <p>chmod 777 - read, write, execute for all</p> <p>chmod 755 - rwx for owner, rx for group and world</p> <p>For more options, see man chmod.</p>	<p>ping <i>host</i> - ping <i>host</i> and output results</p> <p>whois <i>domain</i> - get whois information for <i>domain</i></p> <p>dig <i>domain</i> - get DNS information for <i>domain</i></p> <p>dig -x <i>host</i> - reverse lookup <i>host</i></p> <p>wget <i>file</i> - download <i>file</i></p> <p>wget -c <i>file</i> - continue a stopped download</p>
SSH	Installation
<p>ssh <i>user@host</i> - connect to <i>host</i> as <i>user</i></p> <p>ssh -p <i>port user@host</i> - connect to <i>host</i> on port <i>port</i> as <i>user</i></p> <p>ssh-copy-id <i>user@host</i> - add your key to <i>host</i> for <i>user</i> to enable a keyed or passwordless login</p>	<p>Install from source:</p> <pre>./configure make make install</pre> <p>dpkg -i <i>pkg.deb</i> - install a package (Debian)</p> <p>rpm -Uvh <i>pkg.rpm</i> - install a package (RPM)</p>
Searching	Shortcuts
<p>grep <i>pattern files</i> - search for <i>pattern</i> in <i>files</i></p> <p>grep -r <i>pattern dir</i> - search recursively for <i>pattern</i> in <i>dir</i></p> <p><i>command</i> grep <i>pattern</i> - search for <i>pattern</i> in the output of <i>command</i></p> <p>locate <i>file</i> - find all instances of <i>file</i></p>	<p>Ctrl+C - halts the current command</p> <p>Ctrl+Z - stops the current command, resume with fg in the foreground or bg in the background</p> <p>Ctrl+D - log out of current session, similar to exit</p> <p>Ctrl+W - erases one word in the current line</p> <p>Ctrl+U - erases the whole line</p> <p>Ctrl+R - type to bring up a recent command</p> <p>!! - repeats the last command</p> <p>exit - log out of current session</p>
	<p>* use with extreme caution.</p> 

B Remote Desktop

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.

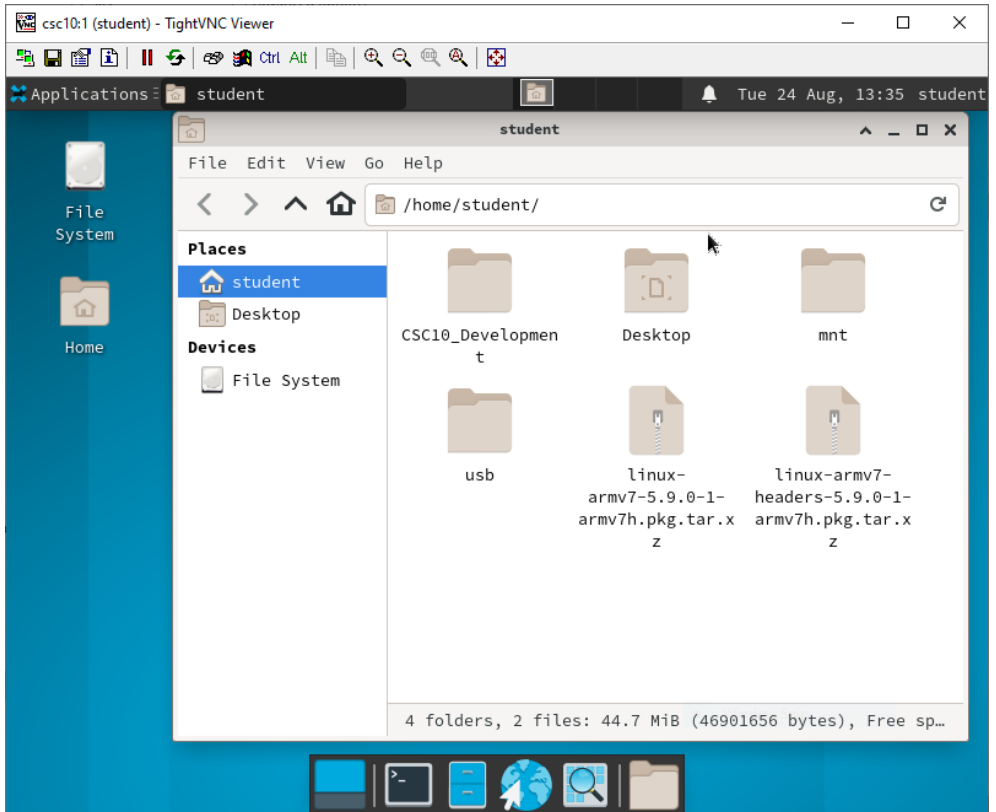
Op de image is ook de desktopomgeving XFCE en de tigervnc-server geïnstalleerd, zie [figuur 12](#). Met de volgende commando's kun je de standaard instelling aanpassen en de server starten. Dit opent een server op poort 5901 met de standaard logingegevens.

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

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

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 12: De XFCE omgeving