

INF3430/4430 Høsten 2007

## **Laboppgave 4**

Bruk av Xilinx EDK

Konstruksjon av System on Chip

## Innledning.

Målet med denne laboppgaven er at dere skal få en innføring i hvordan man kan lage et System-on-Chip (SoC), og få et innblikk i grensesnittet mellom maskinvare og programvare. Tidligere utviklede moduler må tilpasses slik at de passer inn med IP-kjernegrensesnittet til EDK.

Det ferdige produktet til denne oppgaven blir et selvstendig "TV-spill" med input fra tastaturet og output på skjerm (VGA). Spillet styres av et program som kjører på en mikroprosessor-kjerne på FPGAen.

Det er viktig at dere går grundig gjennom tutorialen i deloppgave 1 og gjør det som er beskrevet der før dere setter i gang med å lage systemet. Det er i tillegg mye dokumentasjon tilgjengelig på \EDK\doc\ lokalt på labmaskinene. Spesielt kan *Platform Studio User Guide* og *Embedded System Tools Reference Manual* være nyttige hvis man lurer på noen detaljer. *Les gjennom hver deloppgave grundig før dere starter å gjøre noe.*

## Rapport.

For deloppgave skal dere levere, *i tillegg til* det som nevnes i hver oppgave:

En kortfattet rapport som oppsummerer hva som er gjort, med problemer/utfordringer.

Alle filene i innleveringen skal pakkes ned i en zip-fil med et navn som identifiserer de som har innlevert. I tillegg skal dere demonstrere det ferdige systemet for gruppelærer på laben.

Alle innleverte VHDL-filer skal følge retningslinjene for indentering som beskrevet i kokeboken.

## Oppgave 1. Tutorial

Gå grundig gjennom tutorialen **EDK 8.2 MicroBlaze Tutorial in Spartan 3** (Vi bruker EDK 9.1 men det er ikke stor forskjell) som ligger ved oppgavefilene som EDK82\_MB\_Tutorial\_kommentert.pdf. Denne filen inneholder en del ekstra kommentarer i forhold til den som ligger på nettet hos Xilinx. Hvis dere printer den ut kommer ikke kommentarene med, så sjekk alltid opp i pdf-filen når dere ser et slikt tegn:



Levering:

- Rapporten skal nevne ting som var uklare i tutorialen, og eventuelle problemer som oppstod.

Hint:

- Hvis dere virkelig ikke finner ut hvorfor noe ikke fungerer, prøv å starte på nytt med et helt nytt system, i en ny katalog. Det tar ikke så lang tid å komme tilbake til der man slapp. Det er såpass mange parametre man kan stille på i EDK at det kan være litt vanskelig å vite akkurat hvor noe gikk galt.
- Når et helt system bygges i Platform Studio kommer det gjerne en del advarsler/warnings, men dette trenger man ikke bry seg om. Men man må følge med på advarsler under bygging av "egne" moduler.
- Gi alt nøyaktig samme navn og gjør alt nøyaktig som det foreslås av tutorialen. Det minsker sjansen for at noe går galt, spesielt til simuleringen på slutten.

## Oppgave 2. Konstruksjon av hardware-delen av SoC-systemet

Bygg et system vha. Base System Builder Wizard.

Spesifikasjoner:

- MicroBlaze core
- Reference freq.: 50 MHz, Processor freq.: 50 MHz.
- Local data mem: 8KB
- RS232 @ 57600
- LEDs\_8Bit
- Ikke LED\_7SEGMENT eller Push\_Buttons\_3Bit
- Ikke DIP\_switches eller SRAM
- STDIN/STDOUT: RS232
- Memory test application

Andre detaljer som ikke er nevnt skal ha samme verdier som i tutorialen. Se figur 1. Implementer systemet i hardware – test ut på labkortet med Memory test-applikasjonen. Koble til seriekabel og sjekk at en får output fra denne i terminalprogrammet. Legg merke til at siden det ikke lenger er SRAM i designet, og programmet kjører i resten av minnet, kan ikke programmet i praksis teste noe minne. Det eneste som skrives er "Entering ..." og "Exiting ...".

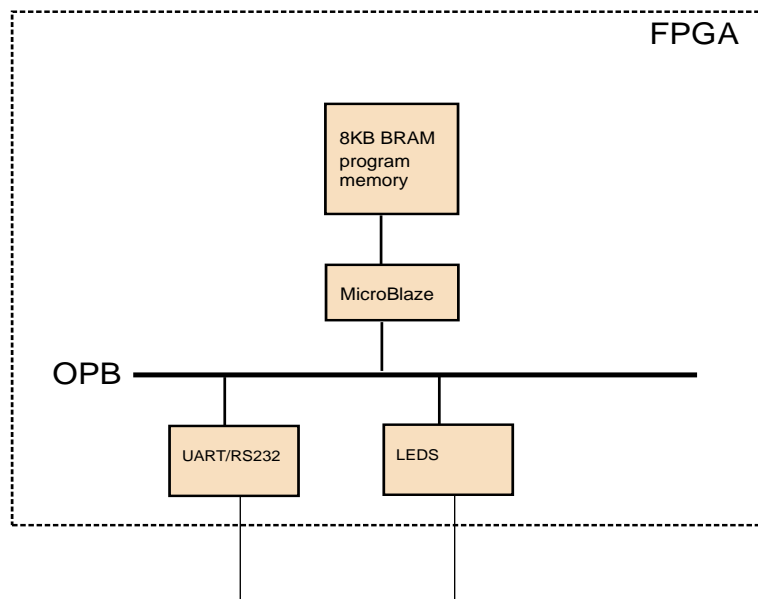


Figure 1: Enkelt blokkdiagram av systemet

Levering:

- Design Report (*Project* → *Generate and View Design Report*. Denne legges i katalogen report/).

Hint:

- Man trenger ikke å gå veien om ISE for å implementere systemet – i stedet kan en velge *Hardware* → *Generate Bitstream* i XPS. For å legge programmet til bitfila velger man *Device Configuration* → *Update Bitstream*, og for å laste ned på kortet velger en *Device Configuration* → *Download Bitstream*. Legg merke til at dette fungerer som en *make*-fil, altså hvis en velger *Configuration* → *Download Bitstream* så vil alle andre oppdateringer

og kompileringer som trengs før dette gjøres automatisk hvis nødvendig. Etter oppgradering til EDK 9.1 har det oppstått noen problemer ved programmering av kortet. Legg til den uthevede linjen under i filen `etc/download.cmd` for at det skal bli mer stabilt:

```
setMode -bscan
setCable -p auto
identify
assignfile -p 1 -file implementation/download.bit
setPreference -pref UseHighz:True
program -p 1
quit
```

### Oppgave 3. VGA-kontroller

I denne deloppgaven skal det legges en ny IP-kjerne til systemet. Vi trenger ikke å konsentrere oss om vhdl-koden til denne, bare tilkoblingen og programmeringsgrensesnittet. Kjernen er en VGA-kontroller som kobles til VGA-porten på testkortet. Denne muliggjør output til en VGA-skjerm i 8 farger, i tekstmodus. Se figur 2. Selve IP-kjernen, i EDK-format, ligger i `opb_color_video_ctrl_v1_00_a/`. Driverfiler ligger i `vga.h` og `vga.c`.

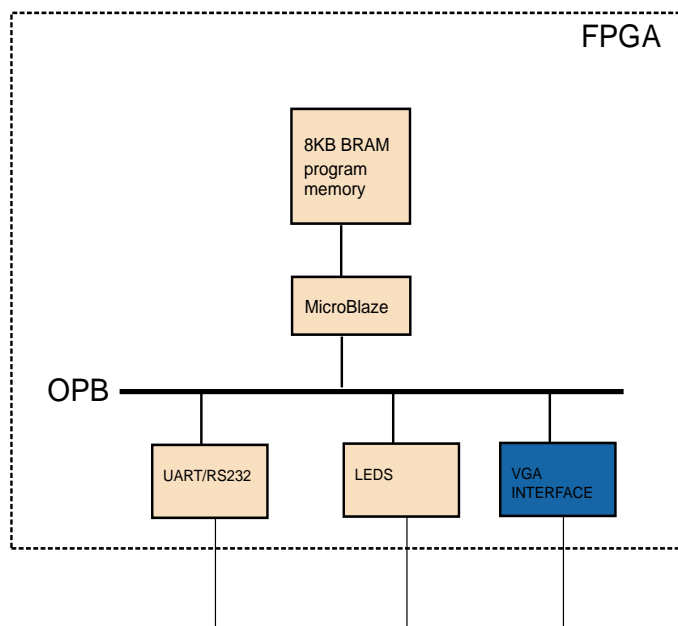


Figure 2: Systemet med VGA-kjerne tilkoblet

### Hardware

Kopier katalogen `opb_color_video_ctrl_v1_00_a` til `pcores/` i prosjektet.

Project → Rescan User IP Directories

Gå så til *IP Catalog*, *Project Local pcores*, og legg til `opb_color_video_ctrl`.

Følg så samme prosedyre som i tutorialen.

Husk å koble til de ekstra portene:

Følgende signaler kan kobles til den allerede eksisterende `sys_clk_s`-linjen: `OPB_CLOCK` og `PIXEL_CLOCK`.

Resten av VGA-kontrollerens porter skal kobles slik at de går ut av systemet, men det er ikke satt opp porter for det ennå. Den enkleste måten å sette opp dette på er å velge *Make External* under *Ports*-perspektivet. Da velges portnavn automatisk. Hvis det blir noe kluss her kan det være lurt å feilsøke

i

system.mhs.

I tillegg til dette må systemets .ucf-fil oppdateres med nye porter som tilsvarer "Port names" som akkurat ble lagt til. Dette gjøres manuelt ved å editere filen, følg mønsteret til de andre portene. FPGA-pinnenavn finner dere i kapittel 5 i Spartan-3 Starter Kit Board User Guide. Bygg så hardware-delen av systemet for å teste at modulen er lagt til riktig. Tools→Generate Bitstream.

## Software

For å bruke VGA-kontrolleren er det en stor fordel å bruke driverkildetekoden som følger med oppgaven. Dette gjør at vi kan styre skjerm-output på et høyere nivå enn om vi skulle snakket direkte med VGA-kontrolleren. Et nytt softwareprosjekt, som også kan brukes i de neste oppgavene, må lages.

*Software* → *Add SW Application Project*. Kall det nye prosjektet "spill" eller noe lignende.

For at bare denne applikasjonen skal lastes ned på kortet, høyreklikk på TestApp\_Memory og velg *Make Project Inactive*. I tillegg bør man høyreklikke på det nye prosjektet og sjekke at *Mark to Initialize BRAMs* er valgt.

I tillegg må vi ha et "linker script". Høyreklikk på det nye prosjektet, velg *Set Compiler Options* Velg *Use Custom Linker Script* og bruk det samme scriptet som brukes til TestApp\_Memory (TestApp\_Memory/src/TestApp\_Memory\_LinkScr.ld).

Nå må VGA-driverfilene legges til prosjektet slik at de blir kompilert og linket inn med applikasjonen. Lag en katalog i XPS-prosjektkatalogen som heter det samme som software-prosjektet. Kopier inn vga.h og vga.c. Høyreklikk på "sources" i software-prosjektet, og legg til begge filene.

For å teste hardwaren og driveren følger det med et veldig enkelt testprogram, vga\_test.c. Dette inneholder main-funksjonen, startpunktet til et C-program. Legg det til prosjektet på samme måte som driverfilene. Tools→Build All User Applications for å teste at det kompilerer riktig. Gjør deretter Tools→Download for å laste bitfilen med HW-konfigurasjon og programmet ned på FPGAen. Legg merke til at dette kjører både build og update bitstream hvis det er nødvendig.

Se på den ekstra skjermen eller vha. VGA-switchen hva som har kommet ut av kortet.

Undersøk testprogrammet og prøv å forandre litt på det. Se i vga.h/c for forklaring på de forskjellige funksjonene som er tilgjengelige for driveren.

Hint:

- Husk å generere nye adresser etter at VGA-kjernen er lagt til.

Levering:

- Det modifiserte testprogrammet

## Oppgave 4. Tilpassing av tastaturgrensesnitt

Tastaturgrensesnittet fra laboppgave 3 skal tilpasses slik at det blir en passende IP-kjerne for bruk i systemet. Det skal kobles som en slave på OPB-bussen, og skal designes som en OPB IPIF core (for å forenkle tilkoblingen). Se figur 3. For å gjøre det enkelt skal det kun ha et registergrensesnitt og en oppførsel lignende den i laboppgave 3. Altså ett register som sier om det er trykket en ny tast, og ett register man skriver til for å signalisere at man har lest inn scan-koden (fra et tredje register). Dette innebærer at man blir nødt til å sjekke registrene ofte for å finne ut av om det har blitt trykket en knapp. Dette kan by på problemer hvis programmet ikke sjekker tastaturkontrolleren ofte nok ifm. lange skjermoppdateringstider eller andre programdeler som tar lang tid. *Pass på at man ikke gjør noe i programmet som "blokkerer" lenge slik at man ikke får med seg alle tastetrykk.* Legg også merke til at filteret som ble brukt i laboppgave 3 *ikke lenger skal brukes*. Alle signalene/kodene som kommer fra tastaturgrensesnittet skal tolkes av programmet.

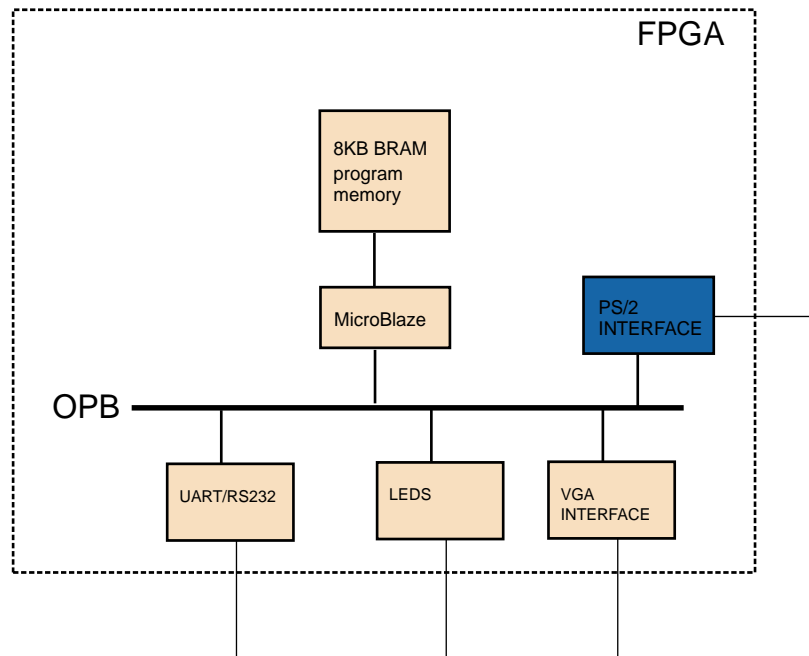


Figure 3: Systemet med tastaturgrensesnitt

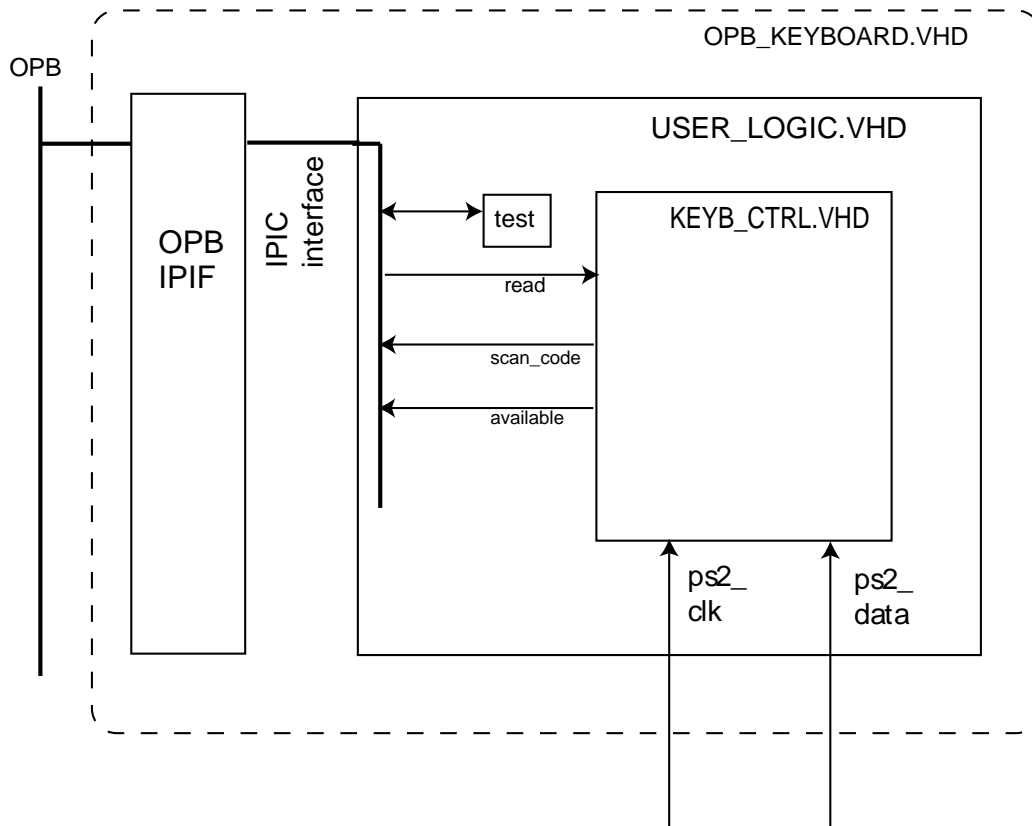
## Hardware

Registre som skal brukes:

Relativ Adresse (Hex)	Registernavn	Read/Write
00	Scancode	R
04	Available	R
08	Read	W
0C	Test	R/W

Testregisteret er der bare for at man skal kunne sjekke at det går an å kommunisere med kjernen – og dermed bekrefte at den er riktig koblet til. Read skal man bare skrive til en gang for å signalisere at man har lest `scan_code`.

Følg framgangsmåten som vist i seksjon IV i tutorialen fra deloppgave 1. Kall modulen `opb_keyboard`. Kun *User logic S/W register support* skal brukes. I motsetning til tutorialen, *ikke* fjern haken ved "Generate ISE..." i skritt 10 i "Create peripheral" (men man trenger ikke "Generate template driver...").



**Figure 4: En oversikt over komponenter i tastaturkjernen**

Så kan dere bygge på i `user_logic.vhd`. Det er enklest å bruke tastaturkontrollerfilen `keybctrl.vhd` fra laboppgave 3 som en modul i `user_logic.vhd`. For å forenkle registerkoden litt kan man ignorere BE (Byte Enable)-signalet. Innholdet i process-en `SLAVE_REG_WRITE_PROC` kan da i første omgang byttes ut med dette:

```

if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
  if Bus2IP_Reset = '1' then
    slv_reg0 <= (others => '0');
    slv_reg1 <= (others => '0');
    slv_reg2 <= (others => '0');
    slv_reg3 <= (others => '0');
  else
    case slv_reg_write_select is
      when "1000" =>
        slv_reg0 <= Bus2IP_Data;
      when "0100" =>
        slv_reg1 <= Bus2IP_Data;
      when "0010" =>
        slv_reg2 <= Bus2IP_Data;
      when "0001" =>
        slv_reg3 <= Bus2IP_Data;
      when others => null;
    end case;
  end if;
end if;

```

Husk å legge til `ps2_clk` og `ps2_data` som ekstra porter til entiteten. Disse må også følges opp i `mpd`-filen som ligger i data-katalogen i `opb_keyboard`. Se i VGA-kontrollerens `mpd`-fil for å se hvordan det er gjort med “ekstrasignalene” der. `Pao`-filen må også oppdateres med en ekstra linje hvis det legges til en ekstra `vhdl`-fil som skal kompileres, f.eks: `lib opb_keyboard_v1_00_a keybctrl vhdl` som må komme før `user_logic` kompileres på grunn av avhengigheter. I tillegg må også `opb_keyboard.vhd` oppdateres med de ekstra portene. Søk på “user ports” i filen for å finne ut hvor det skal legges til.

Hint:

- Noen av registrene skal det bare leses fra. Andre bare skrives til. Dette må gjenspeiles i registerprosessene. De blir en del enklere.
- Siden `scan_code` og `available` allerede er registre i `keyb_ctrl`, er det ikke nødvendig å ha et ekstra register for dem i `user_logic`. Man trenger bare å sende signalene fra disse rett ut på bussen når de blir valgt. Tilsvarende med `read`-signalet – dette kan sendes rett til `keyb_ctrl`. Se figur 4.
- `bus2ip_data` er for skrijving fra bussen til `user_logic`, og `ip2bus_data` er for skrijving fra `user_logic` til bussen.
- Siden alle registrene skal være 32-bits, må man legge en del 0-er til signalene som kommer fra `keyb_ctrl`. Gjør det slik at MSBene (bitene til venstre) fylles ut med 0.
- I EDK er signaler definert som `(0 to X-1)` i stedet for `(X-1 downto 0)`. Dette må man være veldig obs på når man kobler sammen ting.
- Siden det allerede er skrevet et enkelt driverprogram for modulen, må ikke rekkefølgen/adresseringen på registrene forandres. Relativ adresse i registertabellen ovenfor har et tilsvarende signal i `Bus2IP_WrCE` eller `Bus2IP_RdCE`.

Selve implementasjonsprosessen gjøres enklere ved at det automatisk er opprettet et ISE-prosjekt i katalogen `dev1/` under katalogen til tastaturkjernen. Fra ISE kan man så syntetisere for å teste at det fungerer. Det følger også med noen simuleringsfiler i `opb_keyb_sim.zip` som hjelper dere med å simulere `user_logic.vhd` (`vhdl`-filene som ligger utenfor `user_logic` går vi ut i fra at fungerer). For en oversikt over lese- og skrivetransaksjoner over IPIC (grensesnittet som `user_logic` bruker mot resten av systemet), se helt til venstre på figur 26 og 27 i `\EDK\hw\XilinxProcessorIPLib\pcores\opb_ipif_v3_01_a\doc\opb_ipif.pdf`. Det er bare helt enkle `read` og `write` som er støttet i eksempelkoden som ble generert, og det holder for oss.

Det enkleste er å starte simuleringen fra `Modelsim`, men `use proc_common...-linjene` i `vhdl`-filen må kommenteres vekk under simuleringen. I tillegg kan det tenkes at filnavnene i `do`-filen må forandres.

Når alt fungerer i ISE/`Modelsim` kan man koble til tastaturkjernen i XPS, på samme måte som VGA-kjernen. Også her må vi legge til to eksterne porter, for `ps2_clk` og `ps2_data`.

## Software

På samme måte som i forrige oppgave er det også enklest om tastaturet har en driver. Driverfiler har vi laget ferdig for dere, og ligger i `keyboard.c/h`. Disse må legges til samme SW-prosjekt som sist. Så følger det også med `keyboard_test.c` som er en enkel test av tastaturet. For at programmet skal kjøre med denne må `vga_test.c` fjernes fra prosjektet og `keyboard_test.c` legges til. Prøv å eksperimentere litt med denne, og test flere taster. Legg

merke til funksjonen `keyboard_wait_and_poll` som kalles fra hovedløkken i programmet. Sjekk i `keyboard.c` og observer at denne kontinuerlig sjekker om det er trykket noen nye taster.

Levering:

- `user_logic.vhd`
- `mpd` og `pao`-filene
- modifisert `keyboard_test.c`
- `mhs`-filen til systemet
- `ucf`-filen til systemet

## Oppgave 5. "TV"-spill

Lag et spill eller lignende. Kravene er som følger:

- Man skal gi input med keyboardet og at det skal gi utslag i noe på skjermen.
- Det skal være et slags mål med det – f.eks enten ved at man må klare noe spesielt, eller at man får poeng etter hvor lenge man klarer å holde på.

Utover dette trenger det ikke å være komplisert. Spillet skal programmeres i C. En start på noe som kan bli et spill kan finnes i filen `spill_test.c`. Logikken er den samme som i java, men man har ikke tilgang på mange biblioteksfunksjoner – dette skal bli et lite program som passer inn i et veldig lite system!

Forslag til spill:

- Pong – En ball spretter rundt på skjermen, men for at den ikke skal sprette ut på siden, må man være på riktig sted med en "strek" man styrer. Kan lages som enkeltspiller, tospiller eller tospiller mot datamaskinen.
- Lander – Land et romskip trygt ned på en platform ved å gi gass oppover/til siden. Tyngdekraften sørger for at det går nedover. Skipet må ikke lande for fort.
- Snake (litt vanskeligere) – Alle kjenner vel dette. Styr en slange rundt på skjermen. Slangen vokser ved å spise matbiter, og må ikke krasje i seg selv.
- Kortspill eller lignende "statiske" spill (Blackjack, yatzy, ...)
- Bruk fantasien! ☺

Hint:

- En grunnleggende oppskrift på en spilløkke blir: les tastestatus – oppdater posisjon – rens gamle ting på skjermen – tegn nye ting – vent en stund (mens man sjekker taster).
- Når man har forandret på kildekoden, holder det å kjøre Device Configuration→Download for å oppdatere FPGAen med det nye programmet.
- Husk at det bare er 8KB minne tilgjengelig for programmet. Dette er en utfordring. Programmet blir fort for stort til å passe inn i minnet. Da får man en feilmelding under build. Ting som kan gjøres for å hjelpe på dette:
  - Prøv å fjerne overflødig kode, og kode som kan gjenbrukes bør legges i funksjoner.
  - `xil_printf`-funksjonen for å skrive til serieporten tar stor plass, så hvis den ikke er absolutt nødvendig kan den fjernes.
  - Linker script (som sier hvor mye minne som skal brukes til program og data) kan justeres litt på. (F.eks. kan kanskje stack size gjøres mindre uten at det blir noe

problem for programmet – prøv å forandre `_STACK_SIZE` : 0x400 til et lavere tall)

- Hvis man trenger mye dataminne kan man koble til SRAM IP (gjøres i BSB) som en modul som kan holde programdata. Legg merke til at programmet fremdeles må ligge i BRAM for at det skal kunne lastes ned sammen med bitfilen. (Det går an å lage større programmer som lagres i PROM og lastes inn i SRAM vha. en bootloader, hvis det er noen som har lyst til å prøve seg på det).
- Det er ikke noe lurt å skrive hele spillet først på en annen platform. Prøv dere fram gradvis på kortet og legg gradvis til funksjonalitet, hvis ikke kan en få overraskelser i form av for lite minne eller at andre ting ikke fungerer som forventet.
- Portering av spillkildekode skrevet for en annen platform er heller ingen god idé med mindre man har skrevet spillet selv og/eller har full kontroll.

Levering:

- Kildekode til spillet
- Demonstrasjon
- Bitfilen til systemet (inkludert programmet) (`implementation/download.bit`).

## Oppgave 6 (valgfritt).

Legg til noe ekstra hardware til systemet. Her er noen forslag:

- Tilpass syvsegmentskontrolleren fra laboppgave 2 slik at det blir en IP-kjerne som kan kobles på OPB. Vis poengsum eller noe annet på denne.
- Legg til lyd via høyttaler, slik at man kan ha lydeffekter eller musikk. Labveilederne eller Kyrre kan levere ut høyttaler og kildefiler som kan være et utgangspunkt for en lyd-IP.
- Ta evt. kontakt med Kyrre for å høre om det er andre moduler/hardware som kan kobles til.