

CO-DESIGN

JOURNAL TIL CO-DESIGN

E4ISD2/HW

Janus Bo Andersen (JA67494)

Forår 2020

CO-DESIGN OVERBLIK

"80 percent of a product's cost is determined during the first 20 percent of its development cycle."

www.garysmitheda.com

Punkt

Hvad er det?

Detaljer

- **Formål:** Reducere risici og omkostninger i udvikling, opnå hurtigere time-to-market
 - Opnå "optimalt" SW/HW-mix ift. krav og behov
- De fleste komplekse elektroniske produkter indeholder både HW og SW, der skal sam-udvikles
- Co-design er en metode og model for parallelt design af denne hardware og software
 - Systemtænkning og integreret design
 - Kan også benyttes til IC- og SoC-design, osv.

Problemet

- Specifikation af system bestående af HW og SW:
 - Opdele systemet på HW- og SW-komponenter
 - Mappe systemadfærd til systemstruktur.

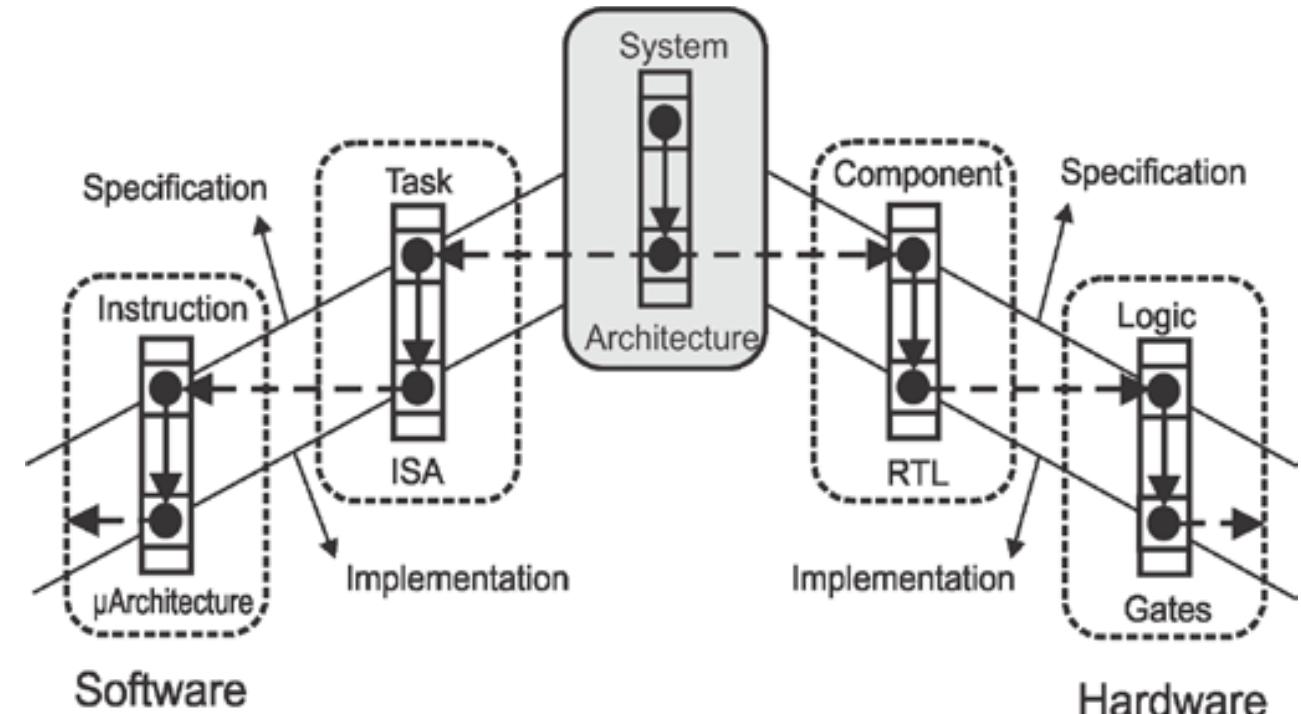
Nøglebegreber

- **Co-:** Concurrency, Coordination; Complexity, Correctness
- Co-design proces = syntese og mapping: Allocation, Binding, Scheduling
- EUDP: Architecture, Modelling og Partitioning

DESIGN-MODEL

Double-roof-modellen

- Systemet er øverste abstraktionsniv.
 - "Double roof": Fra Specifikation (behavioural) til Implementation (strukturel)
- Opdeler systemet i SW- og HW-grene:
- Udviklingskæder til syntese af løsning.
 - Bevæger sig trinvist fra højeste til lavere abstraktionsniveau
 - Implementeringsdetaljer fastlægges / forfines (vertikale pile), dvs. en forfining fra behavioural-beskrivelse til structural-design.
- Fokuseret på løsning af de tre problemer:
allokering, binding og scheduling.



Kilde: Teich (2012)

PROCES OG AKTIVITETER

Synthesis

Aktiviteter

- Oversætte fra *behavioural* specifikation til *strukturel* specifikation.
- For hvert abstraktionsniv.

Allocation

Aktiviteter

- Udvælge el. designe systemressourcer
 - Processorer, IP-blokke (interfaces), memory, osv.
 - Forbindelser; netværk, osv.
- Komponere systemarkitektur
- Allokere ressourcer.

Gentag og forfin iterativt
(test, evaluering, feedback loops, co-simulering, osv)

Binding

Aktiviteter

- Binde *behavioural/funktionelle* krav til strukturelle/arkitektur-objekter.
- Opgaver, processer og funktioner bindes til processeringsressourcer
- Variabler og datastrukturer bindes til hukommelser
- Kommunikation bindes til ruter mellem ressourcer.

Scheduling

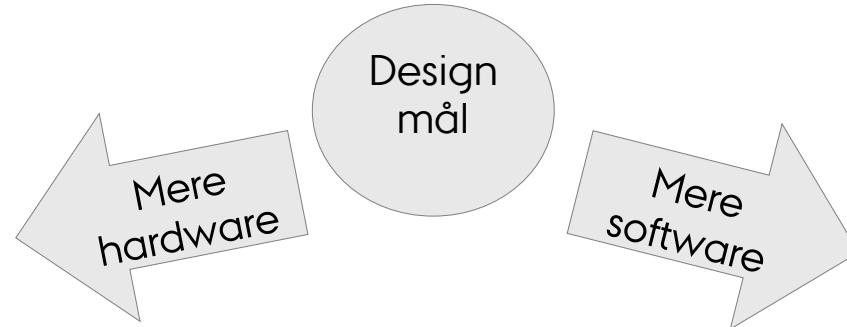
Aktiviteter

- Planlægge hvornår forskellig funktionalitet afvikles på de forskellige ressourcer.
 - Direkte spec. af rækkefølger, eller
 - Specifikation af scheduler for hver CPU
 - Task prioriteter.

Øvrigt:

- Som partitioning i EUDP:
- Afgøre for hvert sub-system om funktionaliteten mest hensigtsmæssigt implementeres i HW eller SW.
- Opnå opdeling, som vil give krævet ydeevne i system
- Omkostninger, størrelse, effekt/energi, hastighed, osv

DESIGN TRADE-OFFS I HW/SW-MIX



Implementering i hardware giver relativt...

- Bedre ydeevne
 - Typisk hurtigere
 - Parallel processesering, forskellige hardware-acceleratorer
 - Garantier for realtid
 - Typisk højere energi-effektivitet.

Implementering i software giver relativt...

- Lavere udviklingsomkostninger
- Kortere udviklingstid
- Lavere designkompleksitet
- Bedre supportmuligheder: Softwaren kan opdateres senere.

FPGA'en er et eksempel hvor der "nemt" foretages et trade-off:

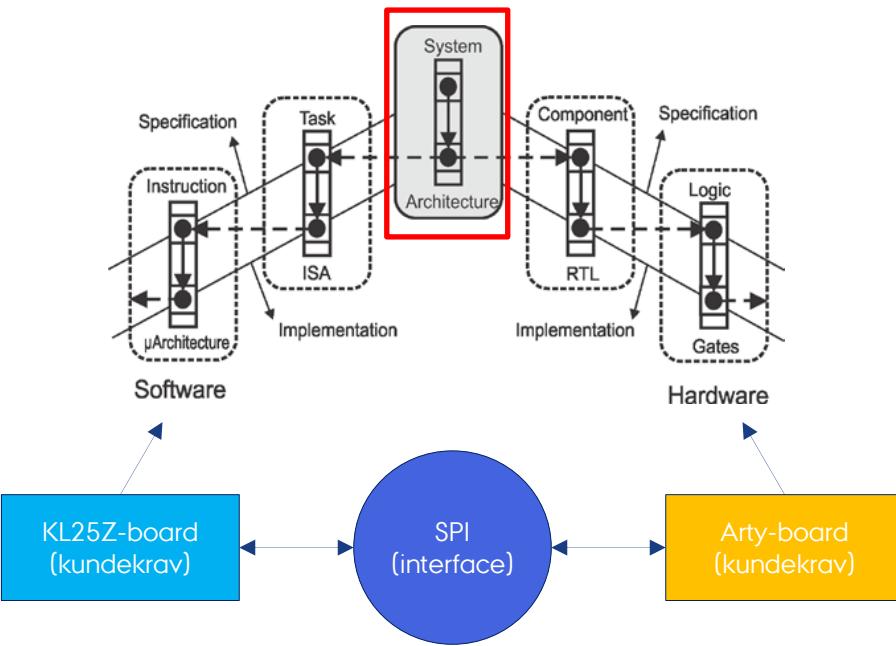
- SoC med syntetiserbar soft-processor.
 - Udvikling af HW, evt. med IP-kerner.
 - Udvikling af SW til afvikling på soft-processor

Tilsvarende skal foretages i fx DSP-applikationer eller udvikling af andre ASIC'er.

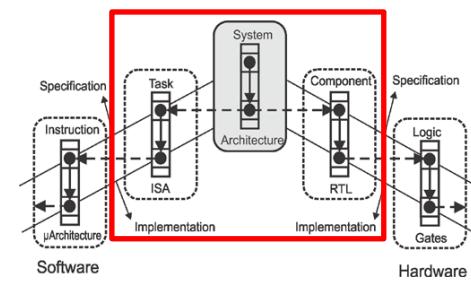
CO-DESIGN PROJEKT FORÅR 2020

Project brief, overordnede "behavioural" krav:

- Lysstyrke på 4 LED'er på Arty-board skal kunne styres via PWM.
- Brugerinterface: Bruger indtaster valg/værdier i konsol til KL25Z-board.
- KL25Z-board sender kommandoer til Arty-board via SPI.
- Arty-board (HW) håndterer stimulering af LED'er via PWM ift. ønsket lysstyrke.

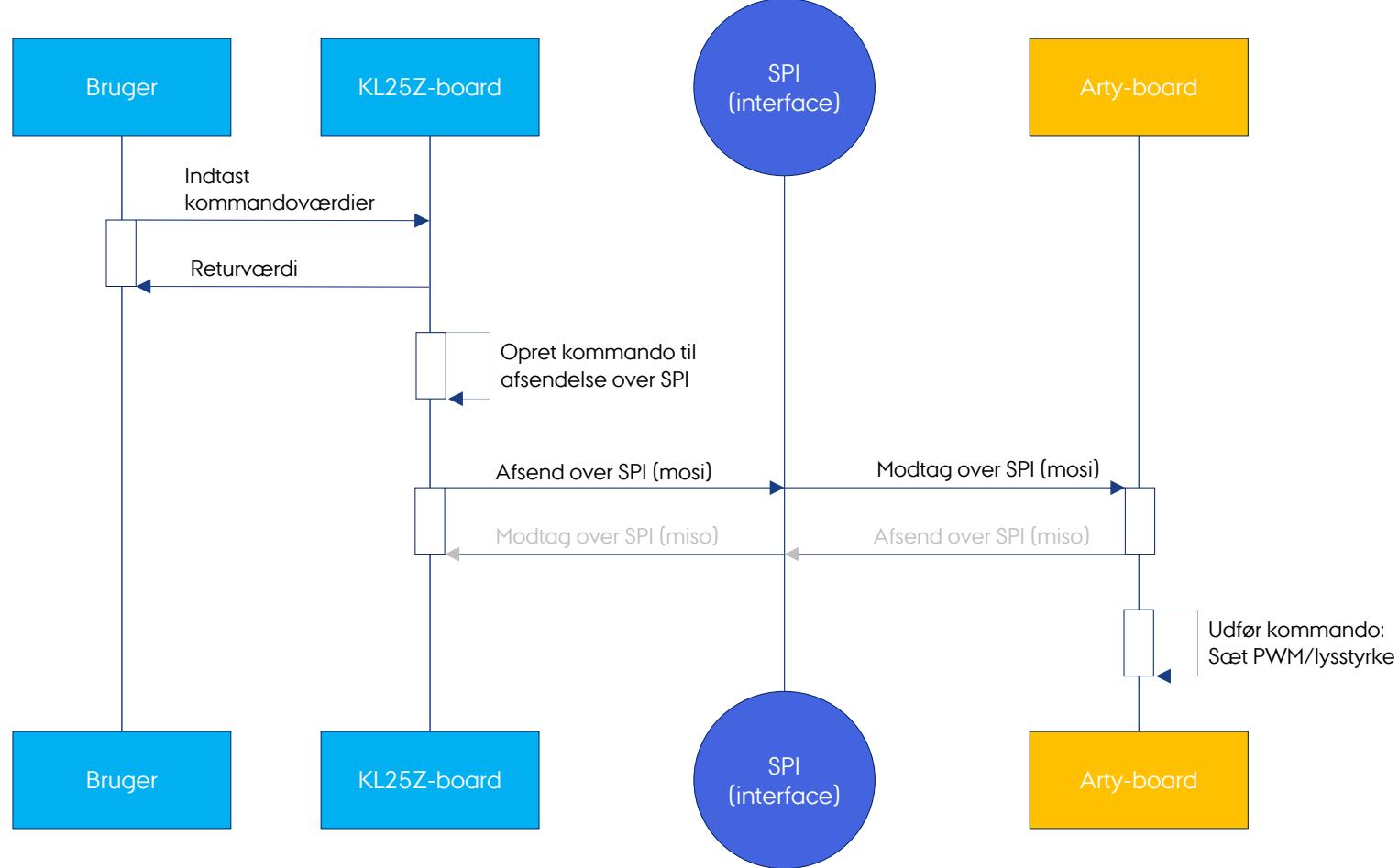


SEKVENSDIAGRAM OVERORDNET SYSTEM: SW+HW

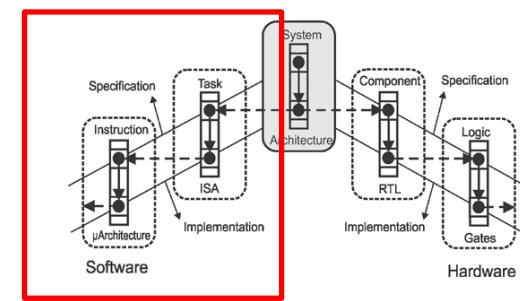


Sekvensdiagrammet viser den overordnede:

- **Allocation**
 - Til platforme / ressourcer
- **Binding**
 - Hvilken ressource udfører hvilken funktionalitet
- **Scheduling**
 - Rækkefølge i afvikling
 - Rækkefølge i kommunikation



SEKVENSDIAGRAM SOFTWARE-SIDE



Sekvensdiagrammet viser den overordnede:

• Allocation

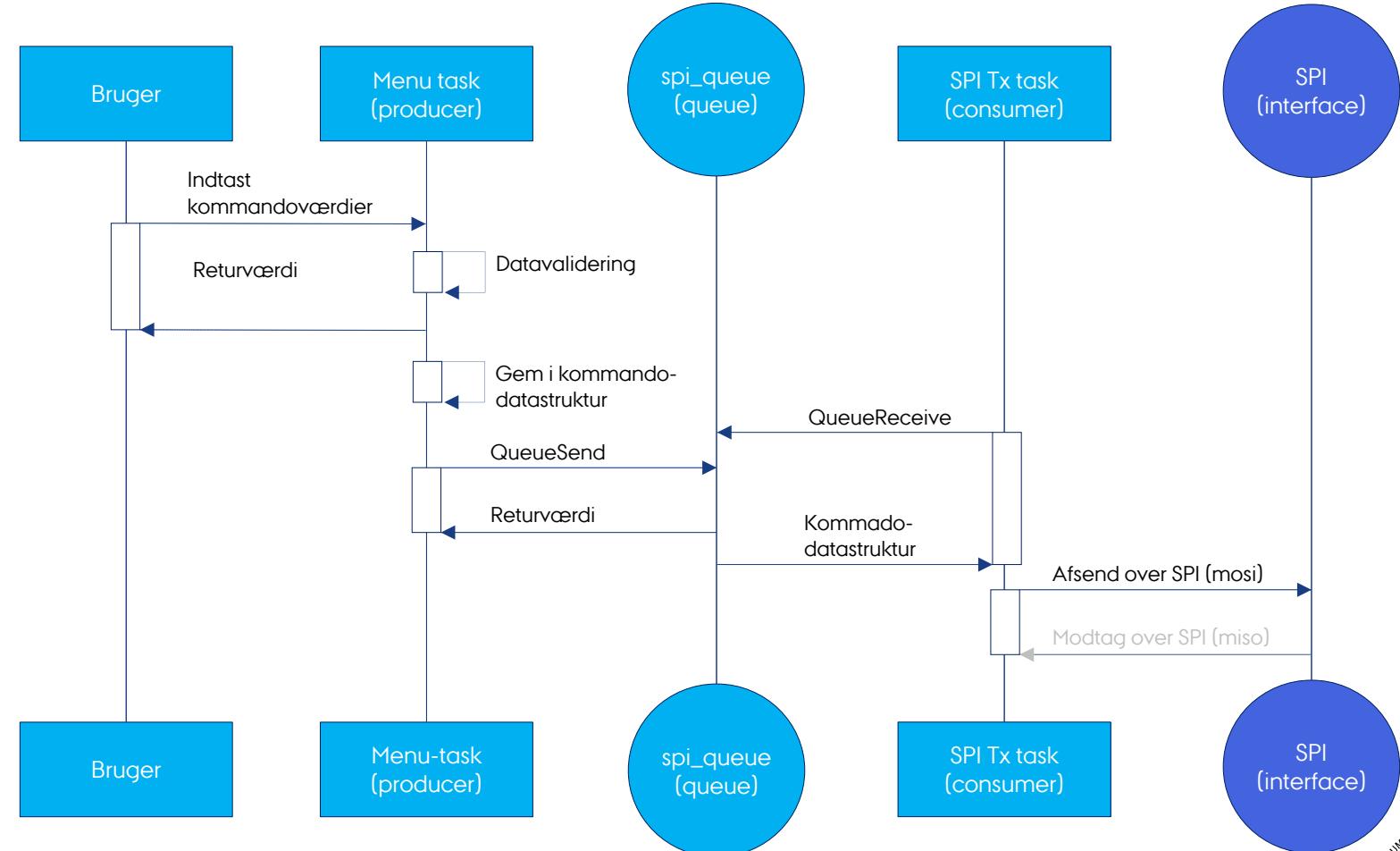
- To RTOS tasks allokeret til at udføre funktionalitet
- En queue til kommunikation mellem tasks

• Binding

- Ansvar i hver task

• Scheduling

- Synkronisering vha. RTOS queues



KOMMUNIKATION: PROTOKOL

En kommando sætter **intensitet** (0-255) for en farve (R, G, B) på en af de 4 RGB-LED'er på Arty.

En kommando består af **2 bytes**

- Byte 0: Adresse på LED-farve-kombination
- Byte 1: Intensitet (0-255)

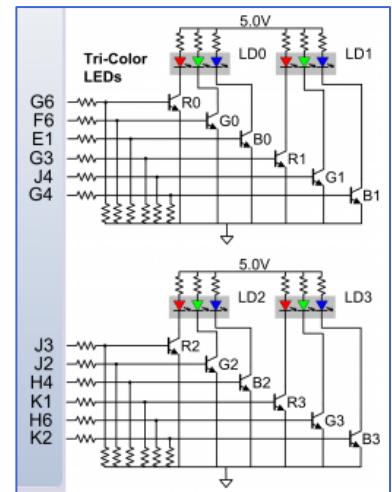
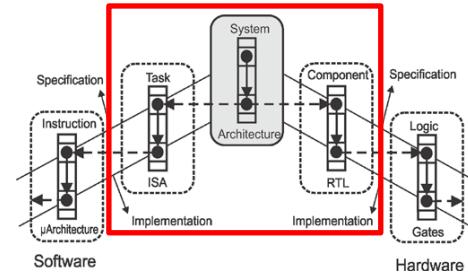
Adressetabel (se th.) i FPGA oversætter fra adresse til enable-signal vha. dekoder.

- Intensiteter (0-255) -> register på adresse

Datastruktur benyttes på KL25Z til at holde en kommando:

```
21 /* Data structure for data transfer */
22 typedef struct {
23     uint8_t addr;    // 1 byte address field
24     uint8_t data;    // 1 byte value field (0-255)
25 } fpga_cmd_t;
```

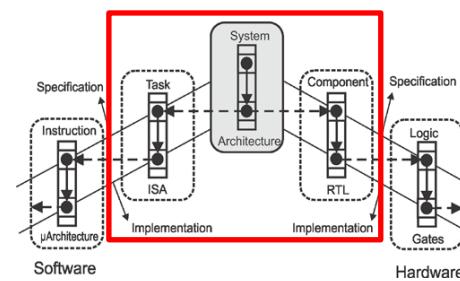
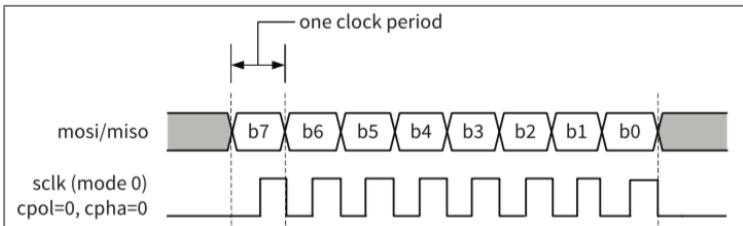
Adresse	LED	Farve	Arty Pin
0	LD0	R0	G6
1	LD0	G0	F6
2	LD0	B0	E1
3	LD1	R1	G3
4	LD1	G1	J4
5	LD1	B1	G4
6	LD2	R2	J3
7	LD2	G2	J2
8	LD2	B2	H4
9	LD3	R3	K1
10	LD3	G3	H6
11	LD3	B3	K2



INTERFACE OG PROTOKOL

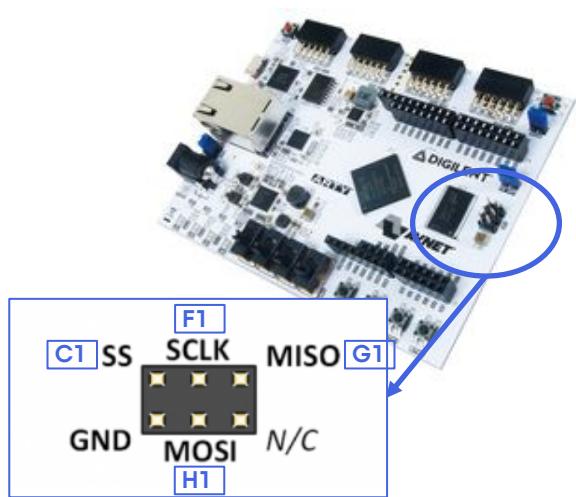
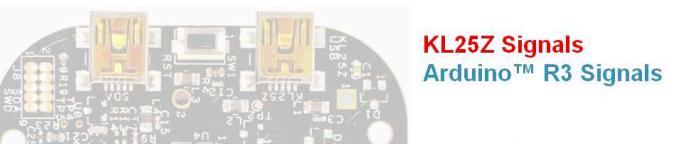
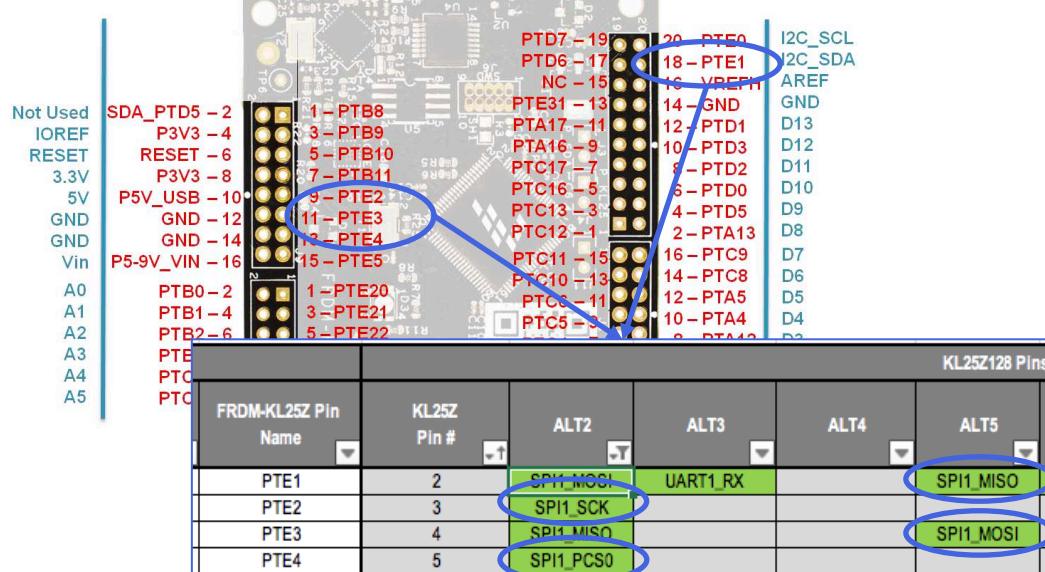
SPI-kommunikation:

- Som standard: 1 byte ad gangen -> 2 datapakker for at overføre en kommando
- Benytter standard (frem for at ændre SPI-implementeringen)

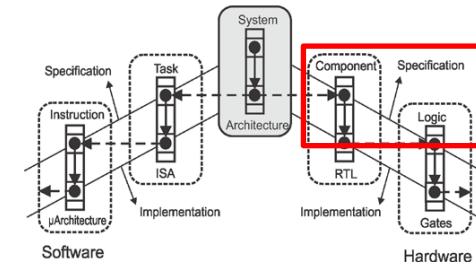


Interface setup:

- KL25Z-board -> *Master: SPI1*
- Arty-board -> *Slave: Custom SPI*
- 3,3 V interfaces
- SPI-indstillinger (se fig.):
 - SPI Mode 0
 - MSB først
 - SPI-clockfrekvens: ca. 46 kHz



HARDWARE-DESIGN



Principper og koncept

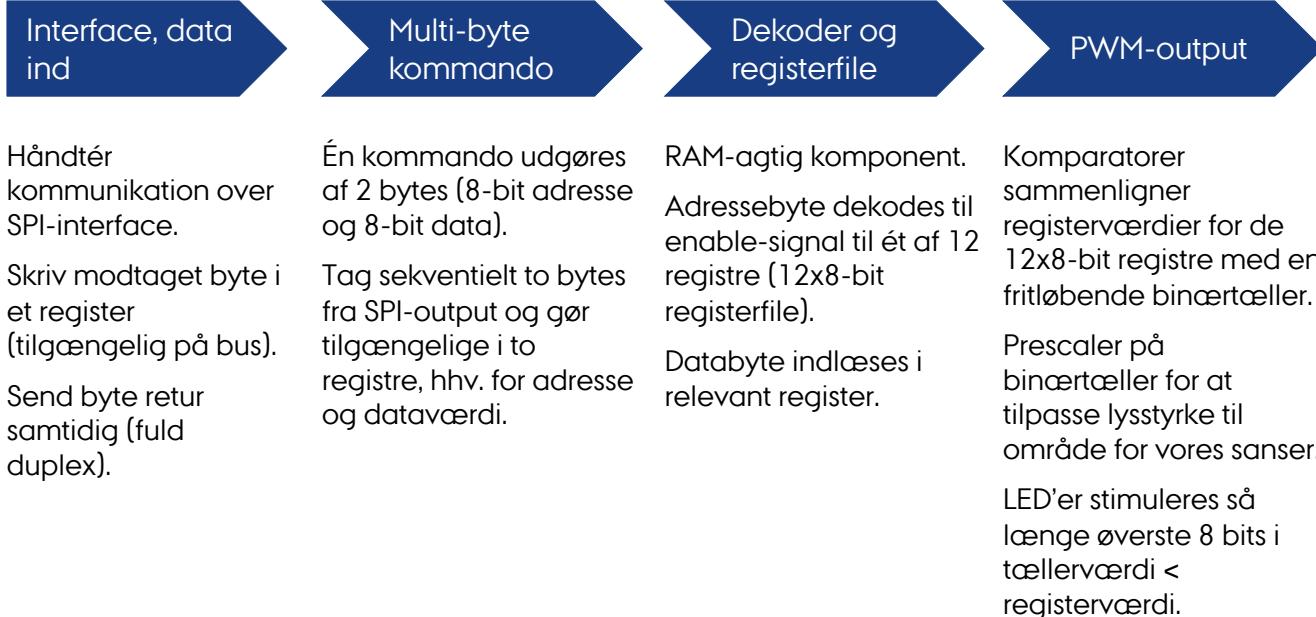
Overordnede design-principper:

- Anvender synkront design og FSM / FSMD'er.
- Modulære komponenter: Hvert HW-kredsløb er en genbrugelig, instantierbar komponent.
- Top-filen instantierer de relevante komponenter.

Overordnet design-koncept:

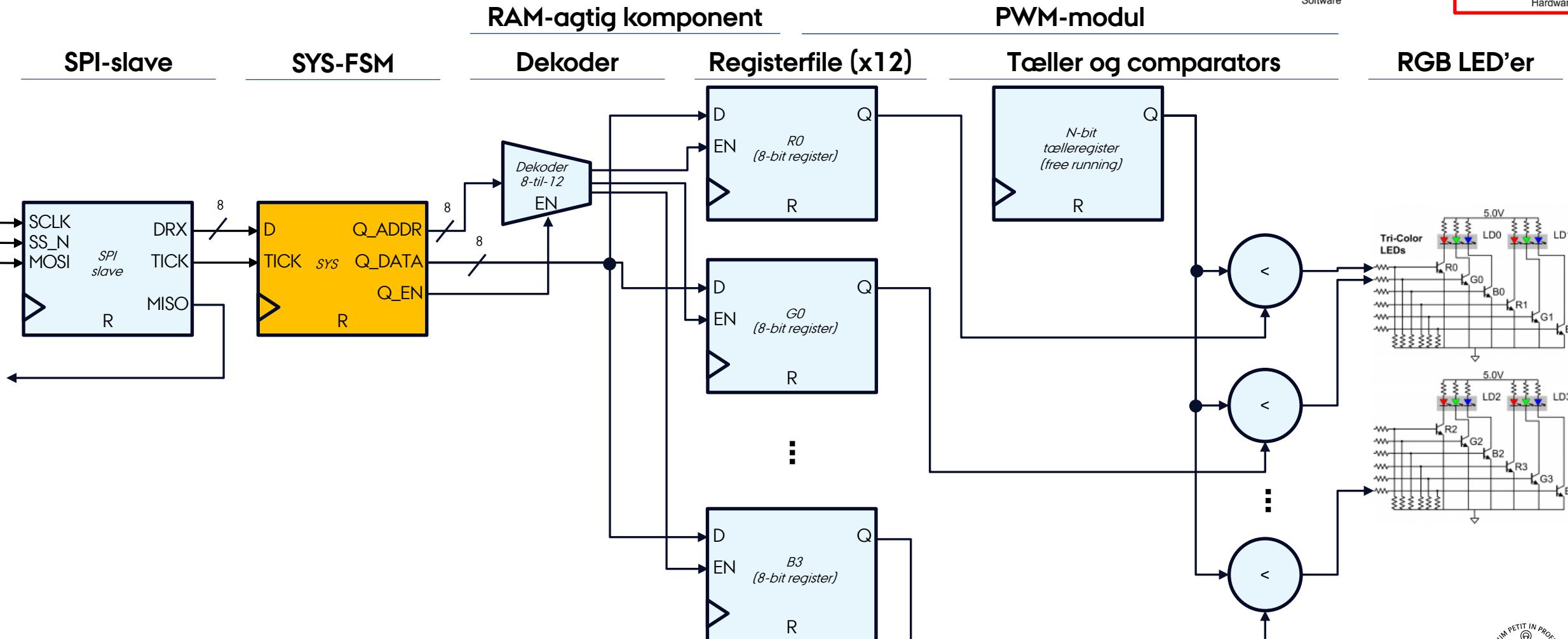
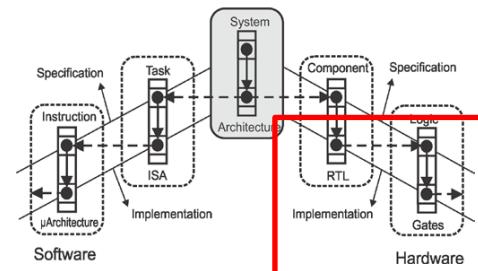
- **System struktureret som en "pipeline":**
 - Data ind -> behandling -> LED output
 - Hver komponent giver 'tick' når data er klar.
 - Application-specific!
 - Frem for et mere generisk system med fælles bus og central processesering.
 - Kunne have valgt at instantiere MicroBlaze MCS core, AXI bus, generisk SPI, osv., og så skrive drivers / kode.

"Pipeline"-struktur (behavioural)

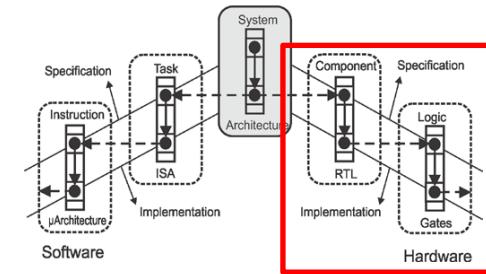


HW: BLOKDIAGRAM

RTL-STRUKTUR FOR SYSTEM



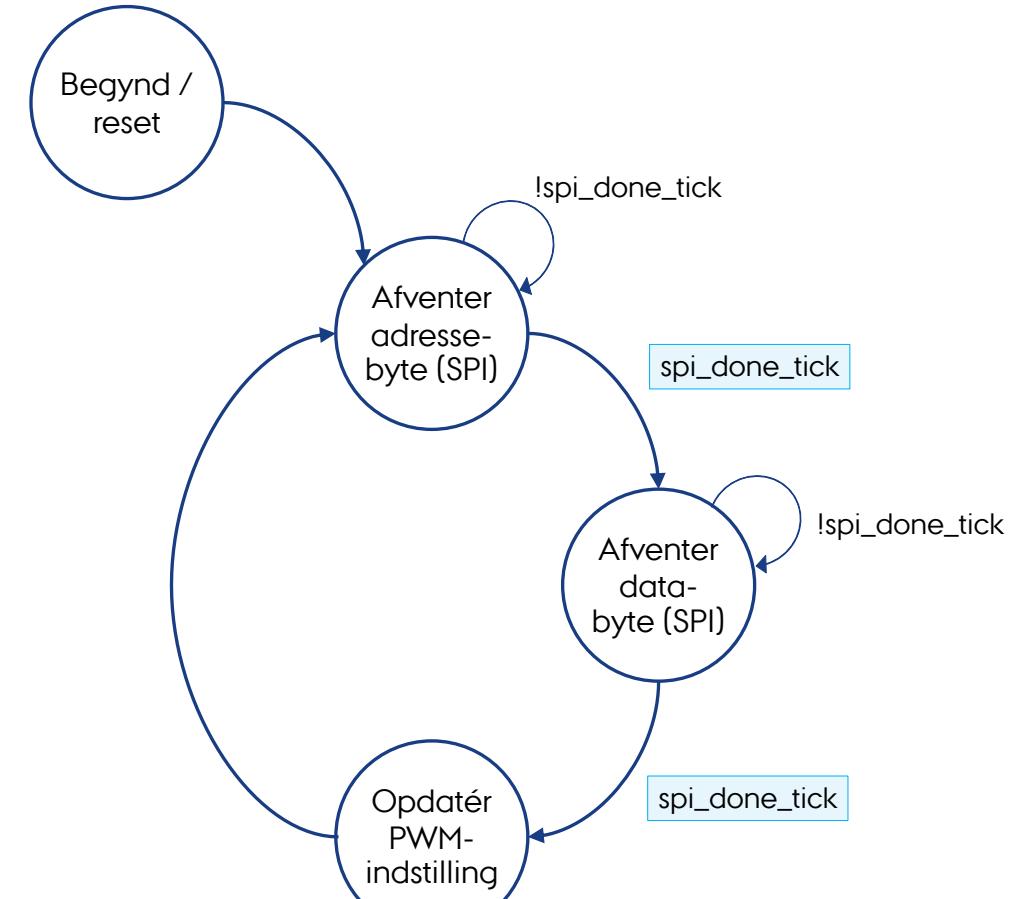
HARDWARE SYS-FSM ER KERNEN I SYSTEMET



Overordnet koncept:

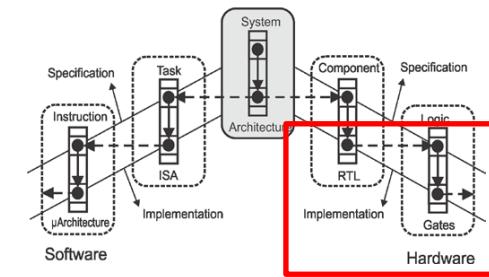
- SYS-FSM er synkront **bindeled** mellem indkommende SPI-transmissioner og PWM-stimulering af LED
- Modelleres som FSM (se t.h.)

State machine til system-tilstand (SYS-FSM)



SYSTEM-FSM (SYS_FSM)

SYNKRONT SYSTEM



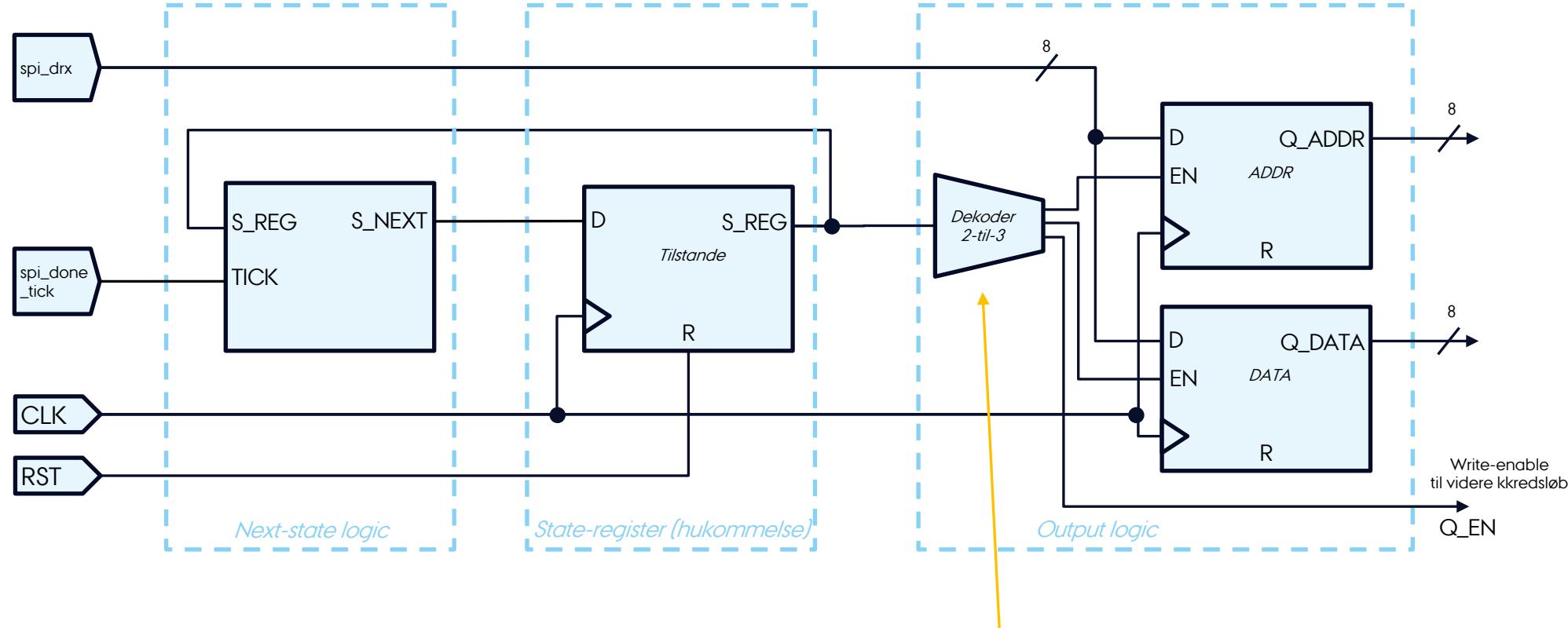
RTL-struktur for state machine til system-tilstand (SYS-FSM)

Mealy-type FSM

- Indholder data path
- Tilstande avancerer ved SPI-done tick
- Output-logik afhængigt af input

Opdaterer registre i rækkefølge

- ADDR-register får første modtagne byte
- DATA-register får anden modtagne byte
- Endelig giver dekoder et højt write-enable signal til PWM registerfile



Idé:
 $S_REG == "00" \rightarrow Y(0:2) = "100"$,
 $S_REG == "01" \rightarrow Y(0:2) = "010"$,
 $S_REG == "10" \rightarrow Y(0:2) = "001"$.
Men syntetiseret til 3 Mux'er.

SYS-FSM-MODUL

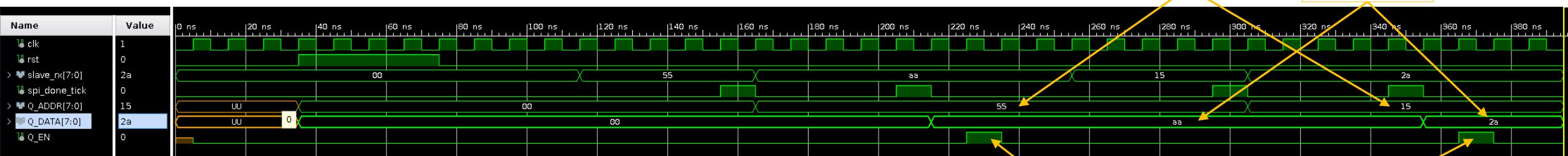
IMPLEMENTERING OG TEST

Implementeret som FSMD.

- Bytes fra SPI (simuleret) -> **SYS-FSM** -> Q_ADDR og Q_DATA (aflæses)

Testbench OK:

- Reset-knappen virker
- Der kan modtages flere kommandoer:
 - To bytes kommer ind fra SPI per hver kommando.
 - Kommando 1 := [55, AA]. Kommando 2 := [15, 2A].
 - Efter hvert *spi_done_tick*='1' ses at,
 - Q_ADDR* opdateres først, dernæst opdateres *Q_DATA*.
 - Efter adresse-bus og data-bus er opdateret, sendes et enable-tick til det videre kredsløb.



HARDWARE

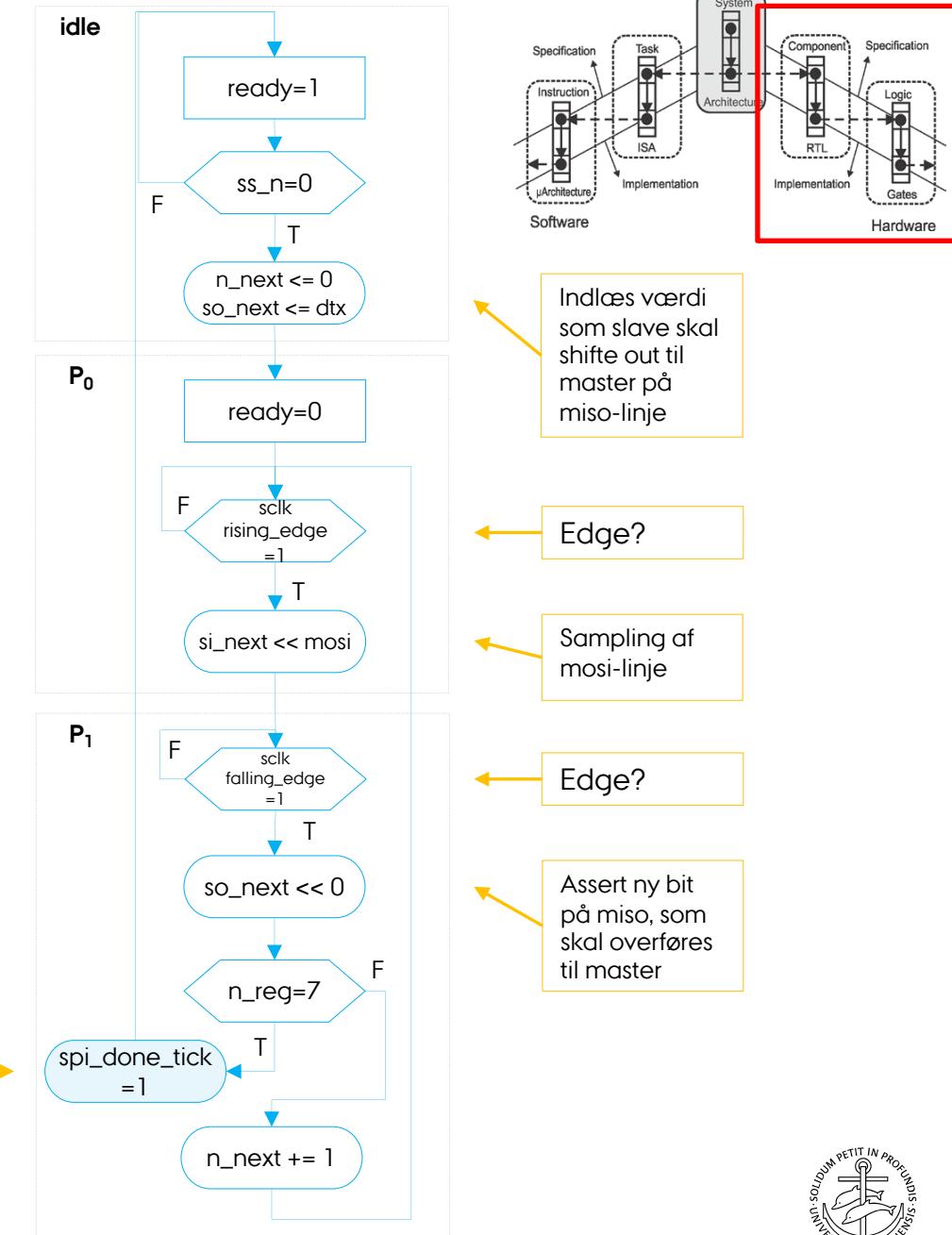
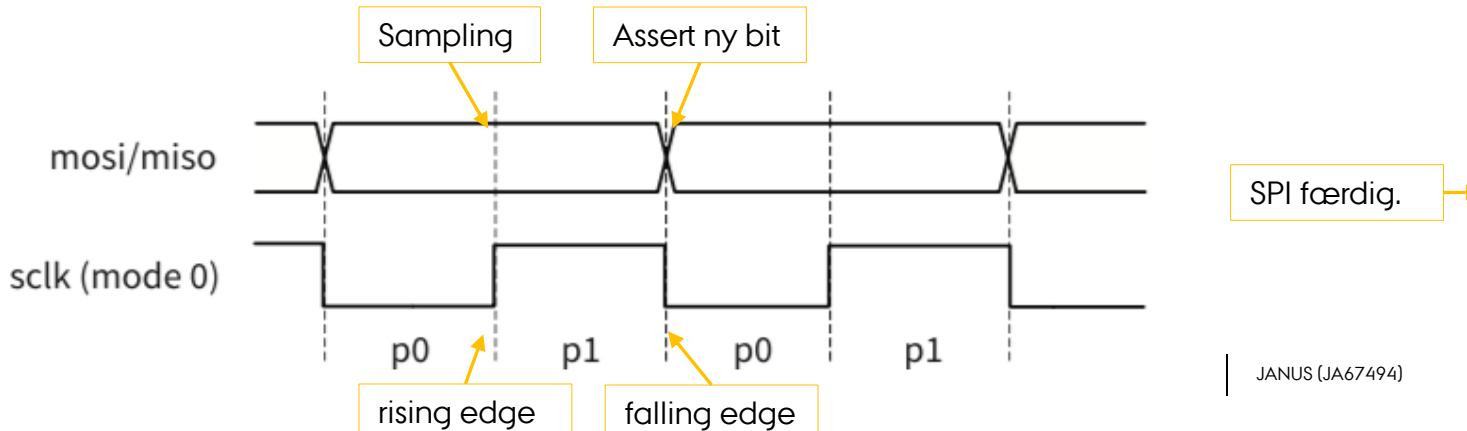
SPI SLAVE

Komponenten skal håndtere modtagelse (og afsendelse) af 8 bits over SPI-interface

- Interface er "Mode 0"
- Skal præsentere modtaget byte i et output-register (på bus)

Koncept:

- Modelleret som FSM (se t.h.):
 - Skifter tilstand i takt med SS og SCLK
 - SS og SCLK er oversamplet med sys_clk (100 MHz)
 - Gør systemet synkront
 - Synkron edge detector på SCLK.
 - Giver output tick hver gang 8 bits er overført og klar i register.



SPI SLAVE-MODUL

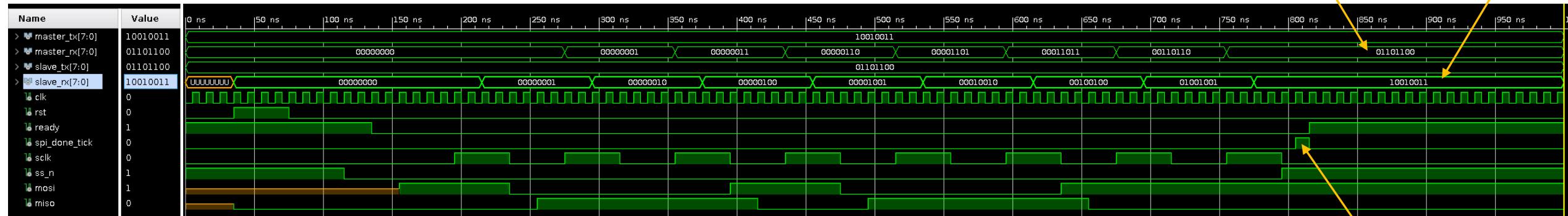
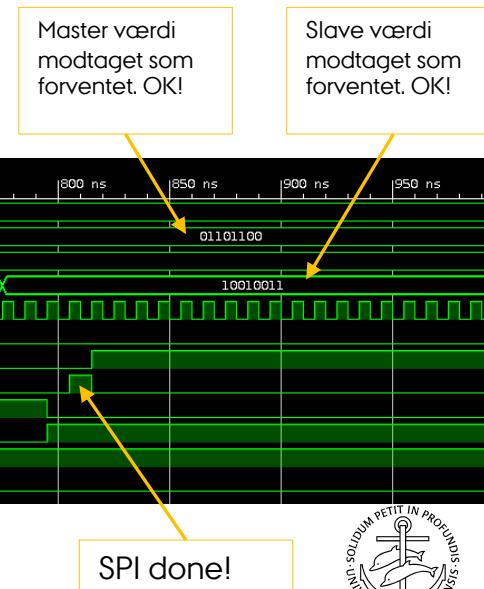
IMPLEMENTERING OG TEST

Implementeret som FSM og med fuld-duplex (shifter ind og ud samtidig med to skifteredistre).

- Simulerede værdier over SPI-interface (SS, SCLK, MOSI, MISO) i tråd med SPI-specifikation (mode 0) -> **SPI-modul** -> registrerværdier (aflæses).

Testbench OK:

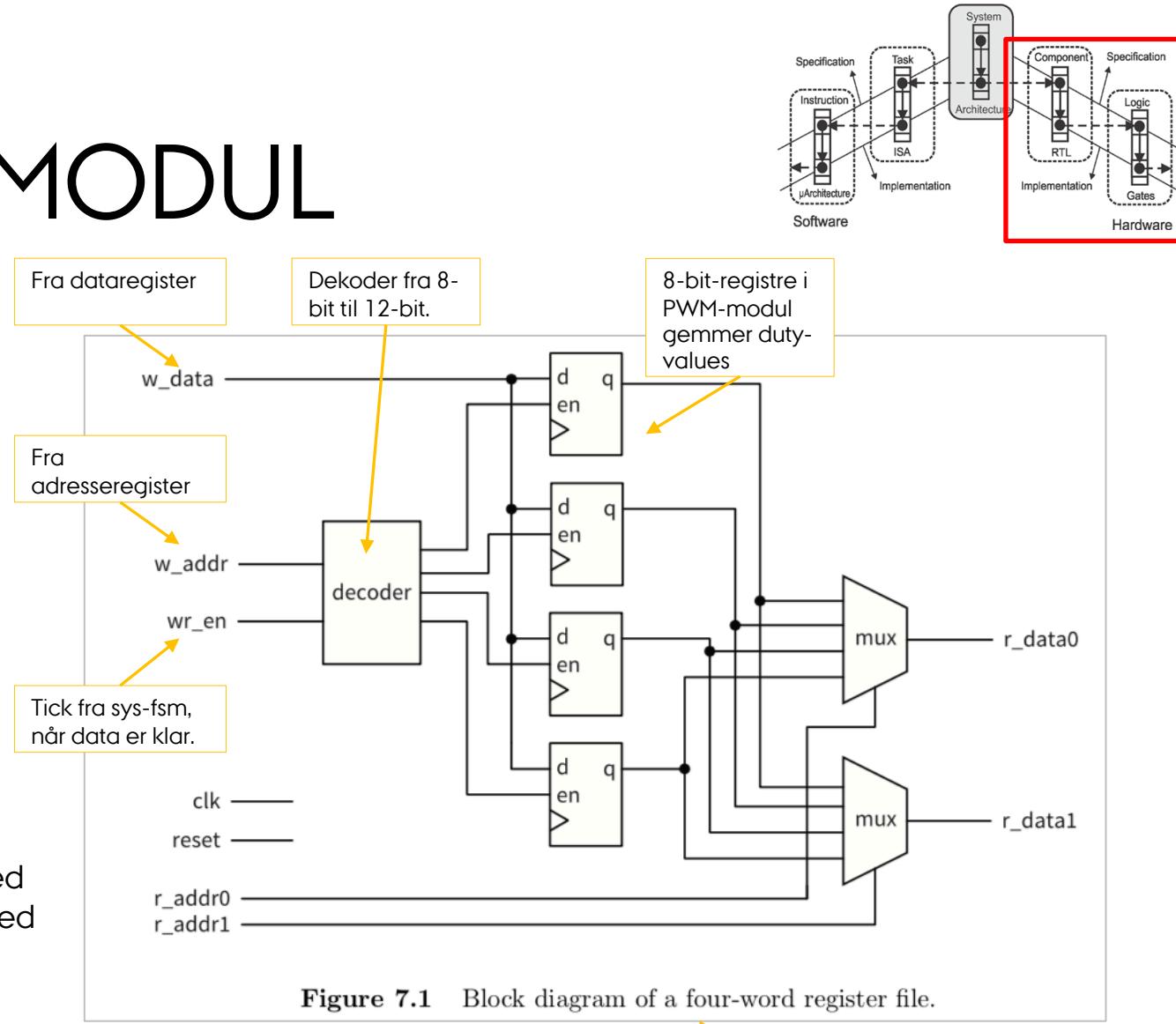
- Reset-knappen virker
- Master sender *master_tx* <= 0b10010011, slave sender *slave_tx* <= 0b01101100
- Efter *spi_done_tick*='1' ses at,
 - *slave_rx* indeholder nu masters besked
 - *master_rx* indeholder ni slaves besked



HARDWARE DEKODER OG PWM-MODUL

Design-koncept:

- Dekoder og registre i PWM-modul fungerer sammen som en RAM-agtig komponent.
 - Inspiration: Chu s. 147 ff. (se fig. 7.1 t.h.)
- **Dekoder** skal oversætte en 8-bit (`uint8_t`) adresseværdi til enable-signaler til de 12 PWM-registre (registerfile).
- **PWM-modul** skal stimulere LED'er med defineret duty cycle.
 - Gemmer duty-value i 12x8-bit registre
 - Én fritløbende binærtæller avancerer synkront på clk.
 - Comparator sammenligner øverste 8 bits i binærtæller med registrerværdier, og stimulerer MOSFET til tilhørende LED med 3,3V eller 0V.
 - Baseret på eksempel med 1 RGB-diode fra MOJ / SMM.



DEKODER-MODUL

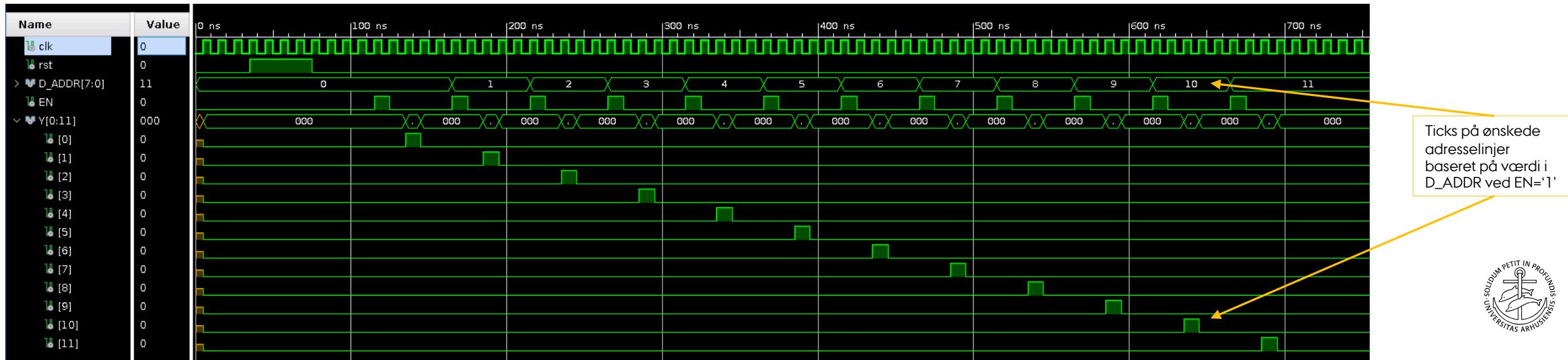
IMPLEMENTERING OG TEST

Implementeret som synkront system med synkron sampling af D_ADDR og EN, hvilket giver 2 clock cycles delay.

- SYS-FSM adresseregister (simuleret) -> **Dekoder** -> Enable-signaler til PWM-registre (aflæst)
- Skal give et enable-tick til PWM-registerfiles, så de indlæser ny data.
- Kunne evt. forbedres med ren kombinatorisk dekoder

Testbench OK:

- Giver tick på ønsket adresselinje, baseret adresse-værdi modtaget fra SPI



PWM-MODUL

IMPLEMENTERING OG TEST

Denne test køres med en 4-bit tæller og 2-bit PWM-værdi (generic mapping), da simuleringen ellers vil tage alt for mange clock cycles at gennemføre.

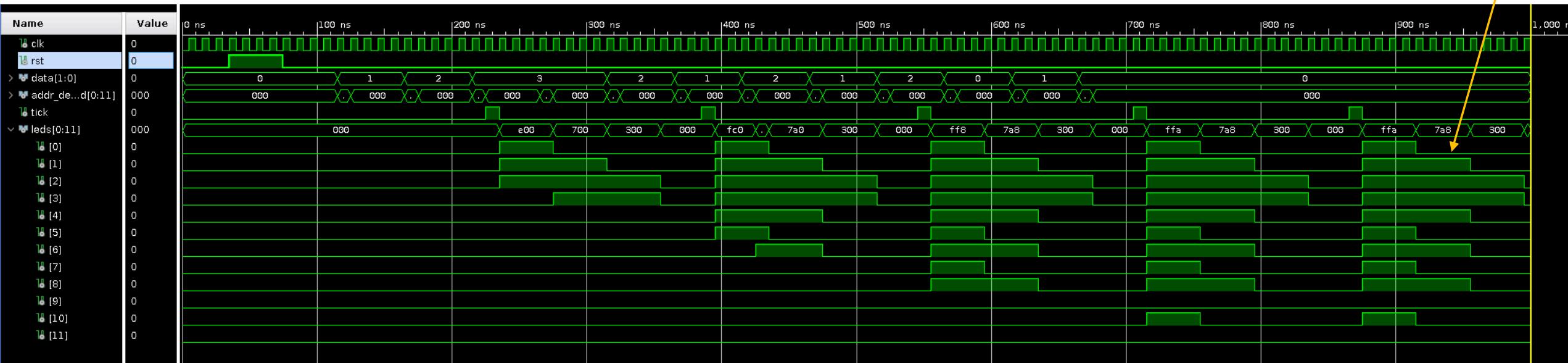
Implementeret 12-LED (4 x RGB) udvidelse til MOJ's skabelon via én binærtæller, registerfile med 12 registre og 12 comparators.

- Enable fra dekoder (simuleret) og data fra SYS-FSM (simuleret) -> **PWM-modul** -> duty-cycles til LED'er (afslæst)

Testbench OK:

- Binærtæller generisk mappet til $N=4$ bits -> overflow med tick hver 16 clock-cycles.
- Dataværdi generisk mappet til $DV=2$ bits bredde -> 4 mulige indstillinger for PWM duty cycle (0%, 25%, 50%, 75%).
 - I HW-implementering laves binærtæller $N=16$ og $DV=8$, så der er 255 mulige indstillinger, de laveste 8 bits er prescaling.
- PWM-registre kan loades med forskellige værdier, *addr[0:11]* er enable signaler, *data[0:DV-1]* indeholder PWM værdi.
- Testeksempel: Indsat forskellige duty cycles på de 12 LED'er, *leds[0:11]*

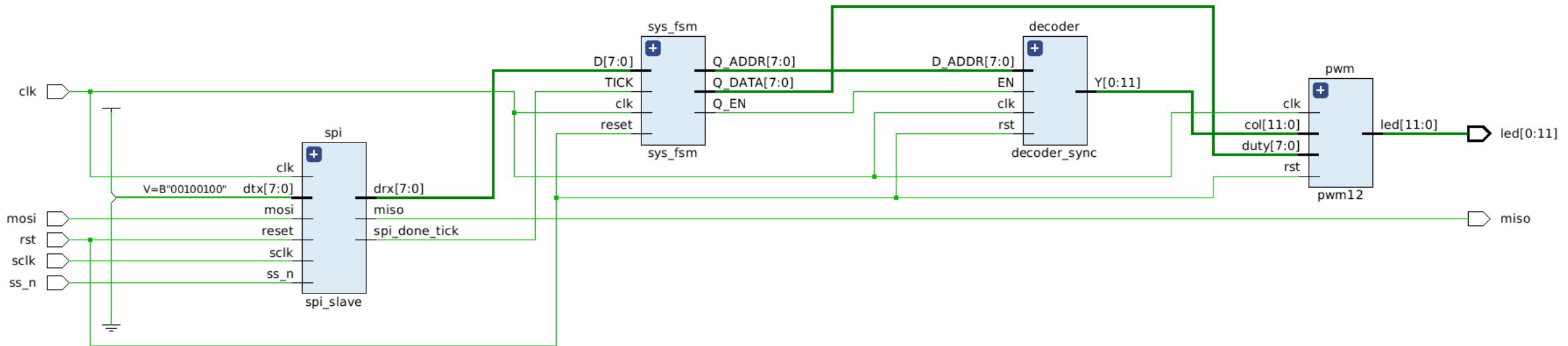
Ønskede PWM
duty cycles for
de 12 LED'er.
OK!



SAMMENSÆTNING AF HW-SYSTEM

Design og implementering:

- Én top-fil:
 - Instantierer og forbinder udviklede komponenter.
- Constraints:
 - Forbinder til I/O: SPI-interface pins, on-board clock (100 MHz), reset-knap og de 4 RGB-LED'er.



INTEGRATIONSTEST

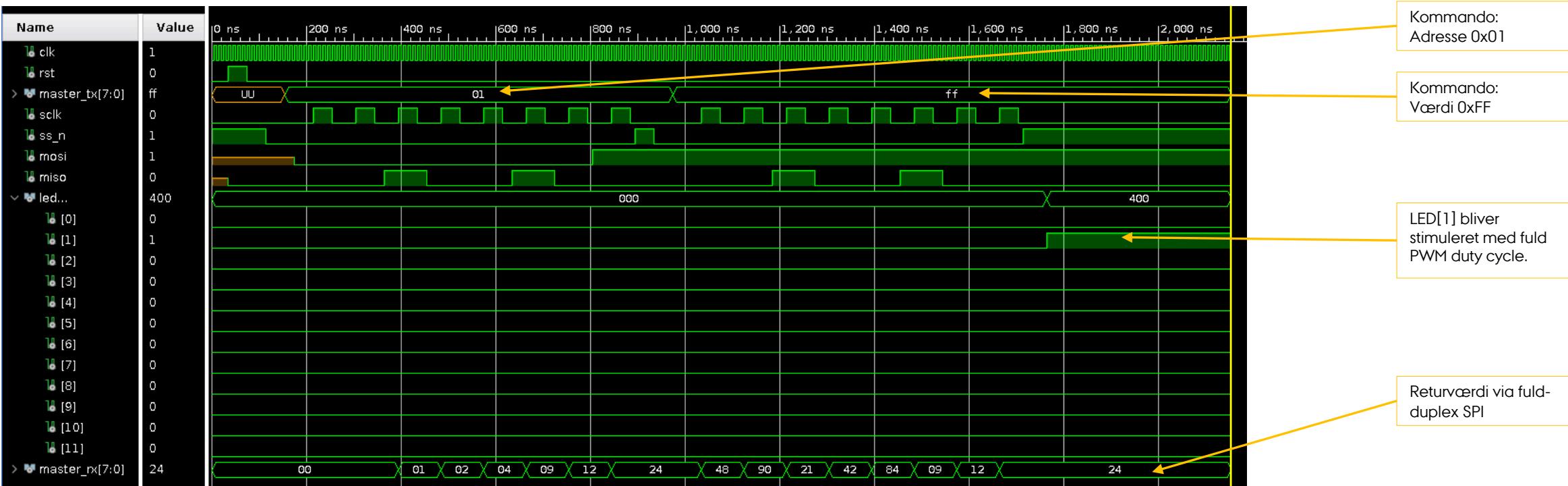
TEST AF 4 MODULER SAMMEN

Sammensætning af hele HW-systemet.

- Tester top-filen, dvs. integreret HW-system.
 - Dataflow: SPI-interface (simuleret) -> **SPI-modul** -> **SYS-FSM** -> **Dekoder** -> **PWM-modul** -> LED'er (aflæst)

Testbench OK!

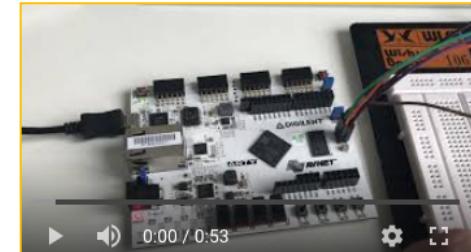
- Sender kommando for at sætte LED[1] (LD0, grøn) til højeste duty cycle => (byte1, byte2) = (0x01, 0xFF)
 - Resultat, LED[1] bliver stimuleret som ønsket.



HARDWARE-TEST

Test OK!:

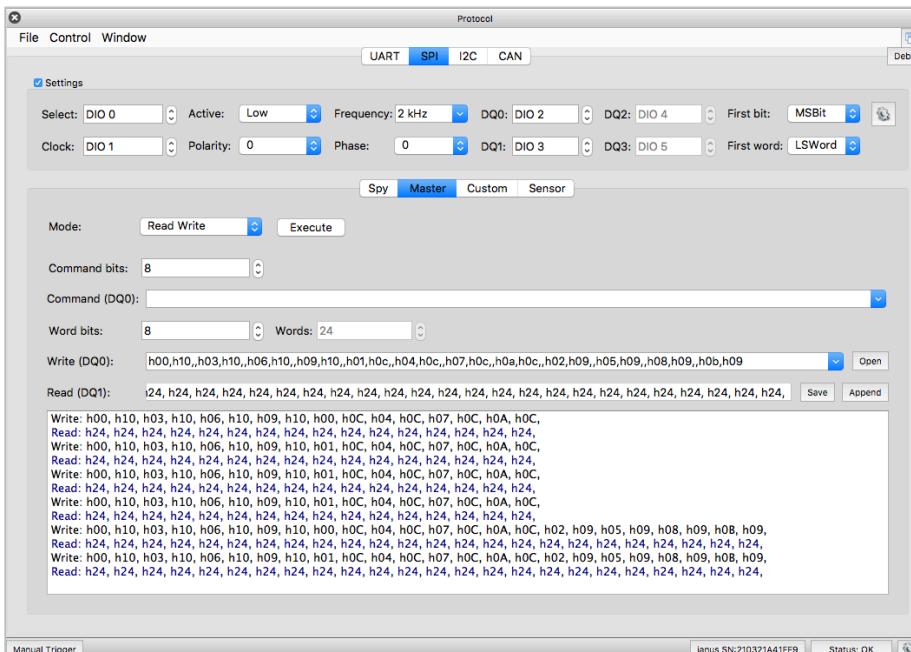
- Kommandoer til at fire RGB'er overføres og virker som forventet.
 - SPI Slave returnerer den hardcodede returværdi (0x24)
 - Reset virker.



https://youtu.be/jqSOHixNE_w

Overfør kommandoer via SPI for alle fire RGB'er

Analog Discover er SPI Master, og overfører til Arty SPI Slave



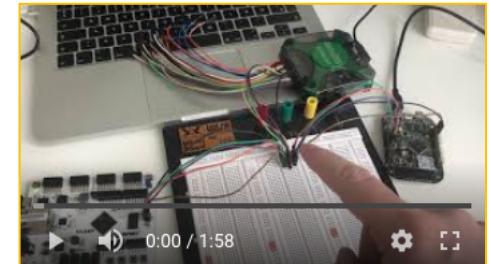
SAMLET SYSTEM



SAMLET SYSTEMTEST

Systemtest OK!:

- Kommandoer overføres fra KL25Z-board til Arty-board via SPI-interface.
- Stimulering på LED'er modsvarer brugerens ønsker / indtastning.
 - Værdi 0 slukker dioden, værdi 255 giver maksimal lysstyrke.
- Alle 4 RGB LED'er (12 forskellige dioder/adresser) kan stimuleres.



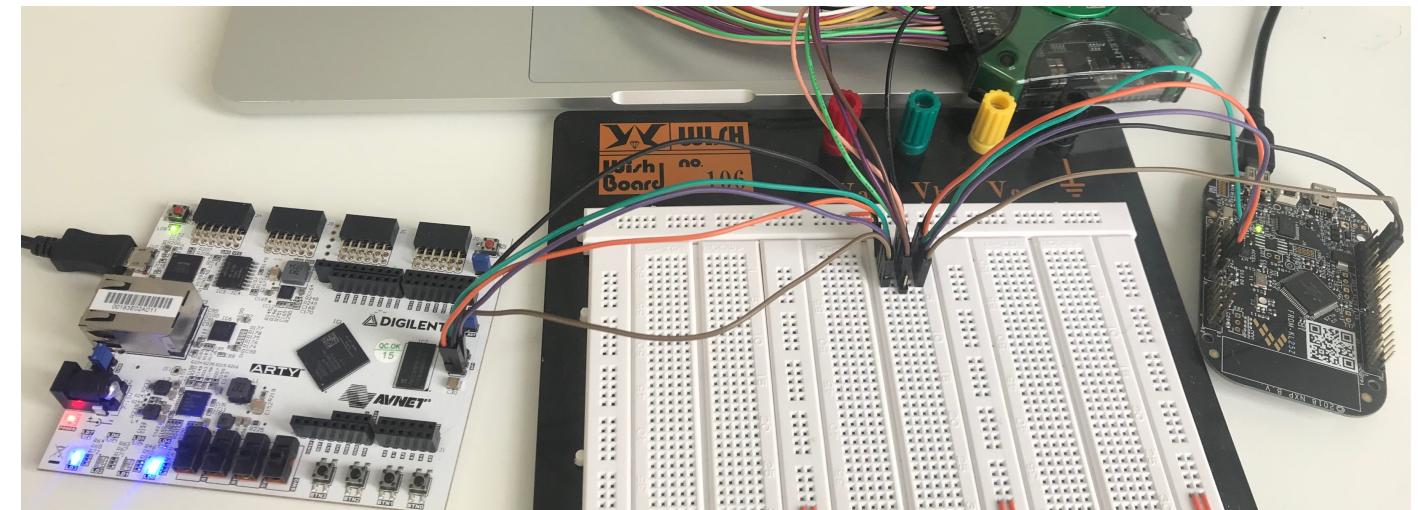
<https://youtu.be/TKfKw5NFccQ>

Bruger indtaster kommandoer i konsol

```
E4ISD2_co-design PE Debug [GDB PEMicro Interface Debugging] Semihosting Console
P&E Semihosting Console

Hello. Co-design Spring 2020.
Enter LED number (address) 0-11:
0
Entered value: 0
Enter PWM value 0-255:
255
Entered value: 255
Sending.
Sent over SPI.
Enter LED number (address) 0-11:
2
Entered value: 2
Enter PWM value 0-255:
255
Entered value: 255
Sending.
Sent over SPI.
```

Opstilling – kommandoer overføres over SPI-interface



KONKLUSION OG EVALUERING

CO-DESIGN-ØVELSE FORÅR 2020

Resultat:

- Vellykket udvikling af HW+SW-system, der implementerer kundens minimumskrav.
- Forsøgt at følge double-roof-modellen i udviklingen.

Læringer:

- Tidlige analyser og design-beslutninger meget vigtige:
 - Interfaces og funktionelle bindinger "varer ved" og er svære at ændre undervejs. Do it right the first time!
- STOR betydning af krav/allokering til platforme:
 - MEGET kortere vej til målet, hvis al funktionalitet var allokeret, bundet og implementeret i software, eller allokeret til to forskellige software-programmérbare platforme (fx KL25Z og Arduino).
- Udvikling og test af HW tager faktor 10 (eller mere) ift. udvikling af software/C-kode.

Konklusion:

- Sjov øvelse! Meget tidkrävende.
- Banker betydning og behov for co-design (og en god udviklingsproces) fast.



AARHUS
UNIVERSITET