EKSAMEN I INDLEJRET SYSTEMDESIGN 1

E3ISD1

Janus Bo Andersen (JA67494)

24. januar 2020 kl. 9.00

EMNER
2 X 10-12 MIN. + VOTERING

Hardware (Morten)

1. Start-up og memory
2. Peripheral I/O
3. Interrupts
4. Timers
5. Seriel kommunikation
6. Analog interfacing

Software (Klaus)

1. C++: Klasser og OOP
2. C++: Funktioner, objekter, polymorphisme
3. C/C++: Make og Makefile
4. Beaglebone: Boot/shutdown, shell-scripts, I/O
5. Beaglebone: Udvikling, (cross-)compiling, debug
6. Beaglebone: Mikroprojekt

HW1: STARTUP OG MEMORY
INDHOLD

1. Globale og lokale variable: Forskelle på variabel scopes og levetid
    1. Memory map for KL25Z og placering af initialisering af variabler
        1. data_init, bss_init, efter init
2. Brug af `volatile` qualifier
3. Stack
    1. Stack pointer

1: Startup og memory

1. Forklar forskellen på en ***global*** og en ***lokal*** variabel. Hvordan initialiseres hhv. globale og lokale variable, når processoren starter op.
2. Hvornår vil du bruge **`volatile`** sammen med oprettelsen af en variabel.
3. Hvad er **stack**, og hvordan bruges denne når du kalder en funktion, f.eks fra **`main()`**.

HW1: STARTUP OG MEMORY
GLOBALE VS LOKALE VARIABLE

Hvad er forskellen?

- Global (statisk) variabel:
  - Bruges til delt data, tilgængelig i alle scopes i en translation unit.
  - Brug `extern`, hvis den deklareres i andre translation units (global på tværs).
  - Lifetime: Statisk, hele programmets levetid.
  - Brug af globale variable *kan* være "code smell", dvs. et tegn på dårligt design.

- Lokal variabel:
  - Lokal til et scope {}, fx en funktions scope f(){…}
  - Indkapslet, interfererer ikke med andre variable i andre scopes.
  - Lifetime: Scope (økonomisk, mindre fejlrisiko).
  - Brug `static`, hvis værdien skal overleve efter scope forlades, men kun være tilgængelig fra scope.

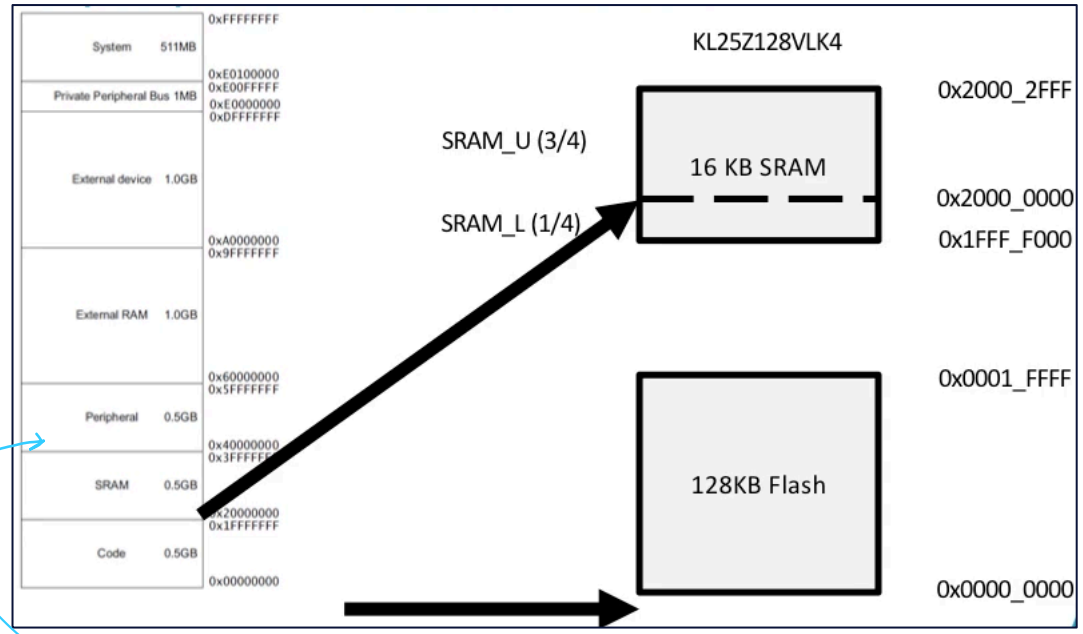
Initialiseres forskelligt. Initialisering -> næste slides (men først kort lidt om memory map for Cortex-M0).

HW1: STARTUP OG MEMORY
MEMORY MAP FOR CORTEX-M0

Struktur:
- 32-bit processor -> 2^32 adresser (4.3e9 -> 4 GB)
- Byte-addressable
  - Word = 32 bit => 1 word = 4 adresser
  - Endianness (!)
- Memory-mapped I/O (peripherals)

Blokke:
- "Code" består af
  - 128 KB Flash (permanent)
  - 4 KB SRAM (flygtigt)
- "SRAM" består af
  - 12 KB SRAM (rest. af de totale 16 KB)
- "Peripheral"
  - Adresser til fx GPIO-registre, osv.
- "Private Peripheral"
  - Til FGPIO registre (direkte adgang ad "privat vej"...)

Behov for lagring:
- Kode
- RO statisk data
- RW statisk data
  - Initialiseret, 0-initialiseret, u-initialiseret.
- Stack
- Heap



Absolute address (hex)	Register name	Width (in bits)	Access
4004_9000	Pin Control Register n (PORTA_PCR0)	32	R/W
4004_9004	Pin Control Register n (PORTA_PCR1)	32	R/W
4004_9008	Pin Control Register n (PORTA_PCR2)	32	R/W
4004_900C	Pin Control Register n (PORTA_PCR3)	32	R/W
4004_9010	Pin Control Register n (PORTA_PCR4)	32	R/W
4004_9014	Pin Control Register n (PORTA_PCR5)	32	R/W
4004_9018	Pin Control Register n (PORTA_PCR6)	32	R/W
4004_901C	Pin Control Register n (PORTA_PCR7)	32	R/W

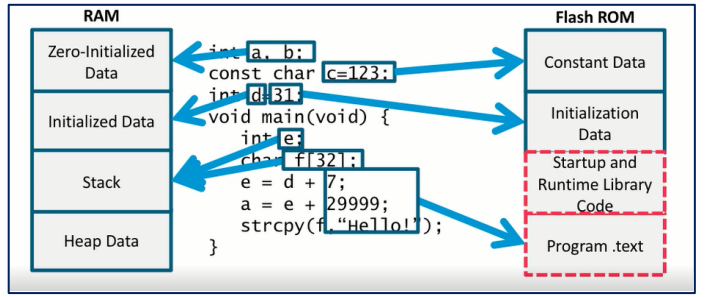
+4

HW1: STARTUP OG MEMORY
INITIALISERING AF VARIABLE

Proces for initialisering ved processorens opstart drives af startup-filen (startup_mkl25z4.c):

- ResetISR(), SystemInit() og så håndteres data…

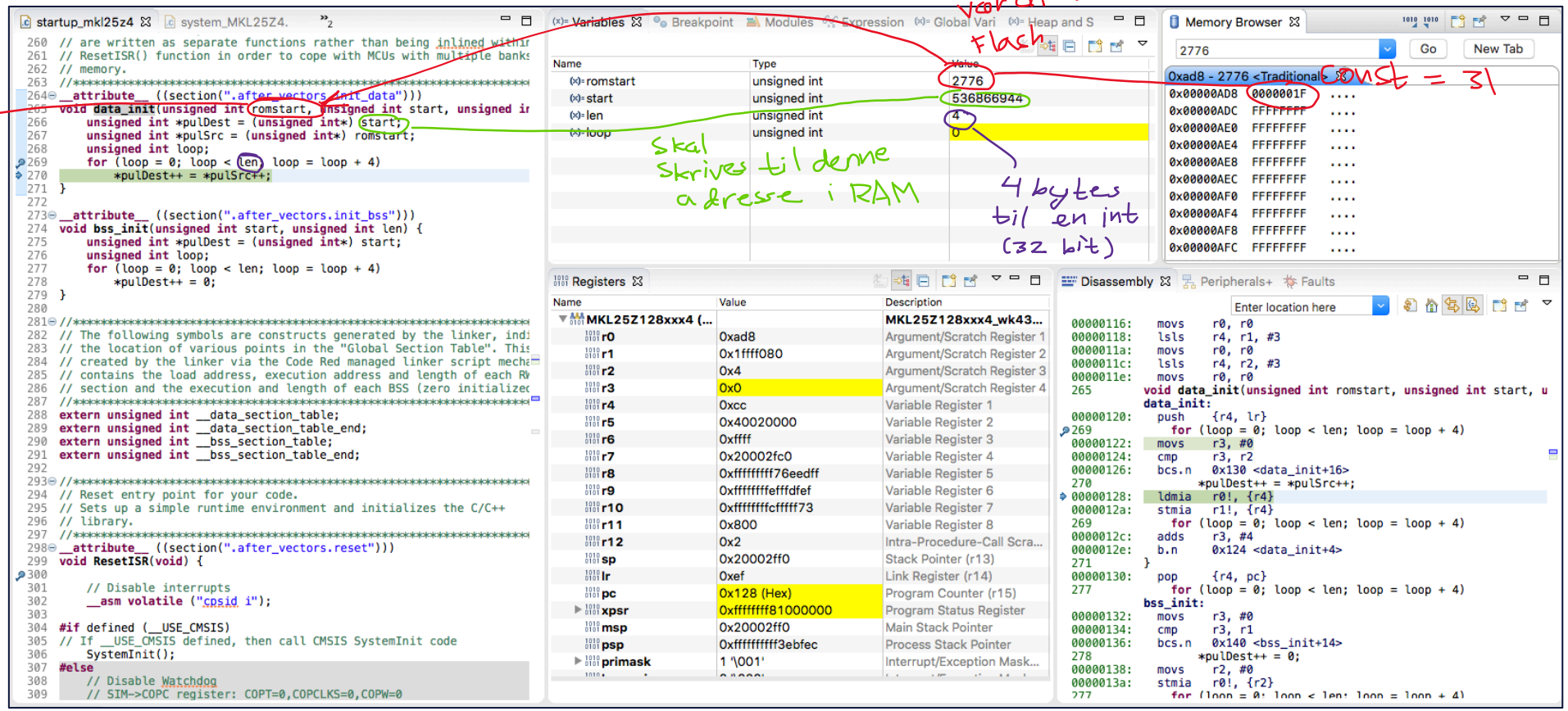
- Eks.: Lokalisering følger C's memory model
  - Globale / lokale statiske:
    - Const variable er gemt i et read-only område (RO).
    - Med initial værdi -> datasektion (RW) ->håndteres med **data_init**
    - Uden initial værdi -> bliver 0-init -> BSS-sektion (RW) -> **bss_init**.
  - Lokale:
    - Automatiske: Stack (RW) (u/værdi)
    - Dynamiske: Heap (RW) (u/værdi)




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

int a, b;           // global, uninitialized

const char c=123;   // global const

int d=31;           // global, initialized

void main(void)

{

   int e;           // local, uninitialized

   char f[32];      // local, uninitialized

   e = d + 7;

   a = e + 29999;

   strcpy(f, "Hello!");

}
```

HW1: STARTUP OG MEMORY
DATA_INIT

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

int a, b;           // global, uninitialized

const char c=123;   // global const

int d=31;           // global, initialized

void main(void)

{

    int e;          // local, uninitialized

    char f[32];     // local, uninitialized

    e = d + 7;

    a = e + 29999;

    strcpy(f, "Hello!");

}
```

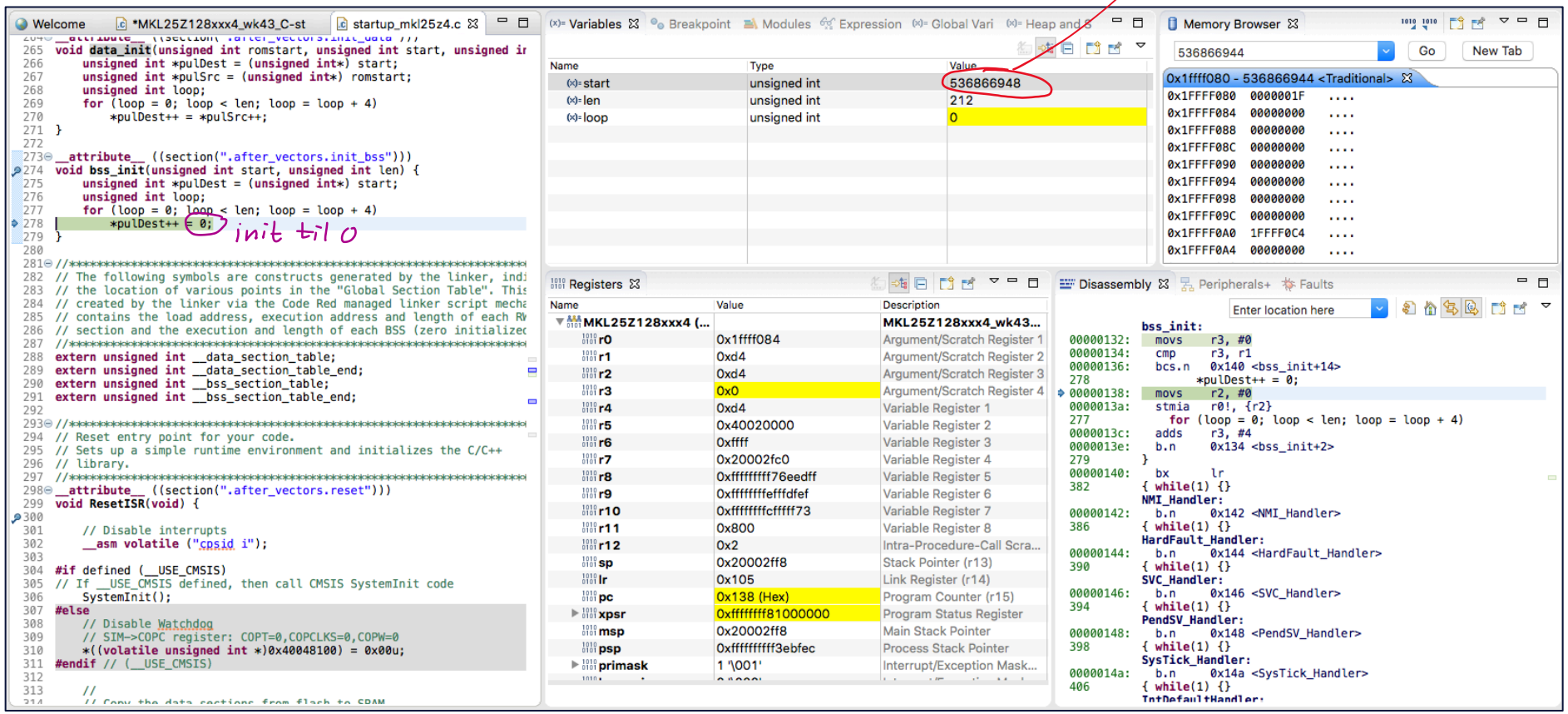



```
2776 = 0x00 00 0A D8 (Flash)
536866944 = 0x1F FF F0 80  (SRAM)
```

HW1: STARTUP OG MEMORY

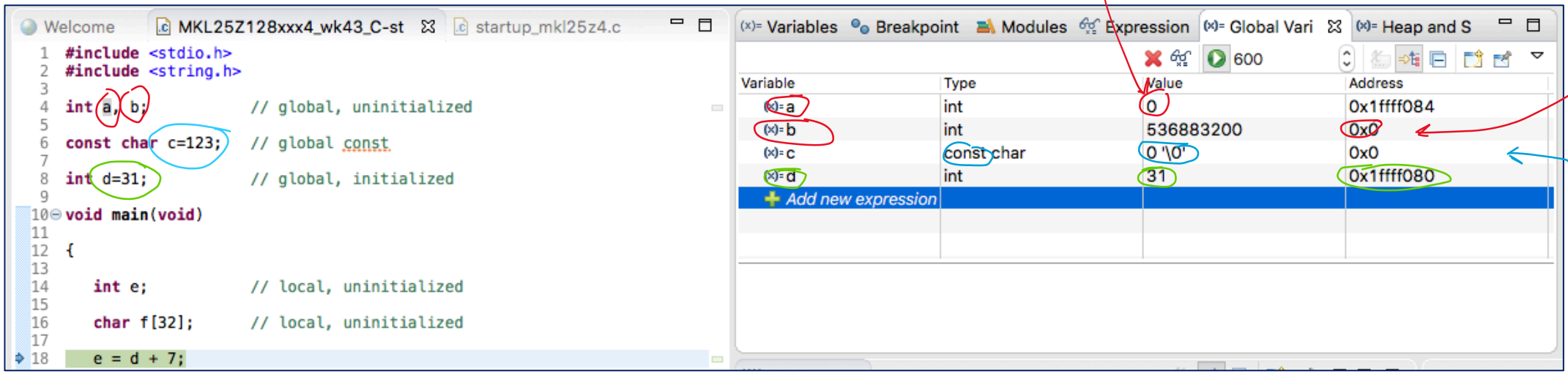
BSS_INIT

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

int a, b;            // global, uninitialized

const char c=123;    // global const

int d=31;            // global, initialized

void main(void)

{

    int e;           // local, uninitialized

    char f[32];      // local, uninitialized

    e = d + 7;

    a = e + 29999;

    strcpy(f, "Hello!");

}
```

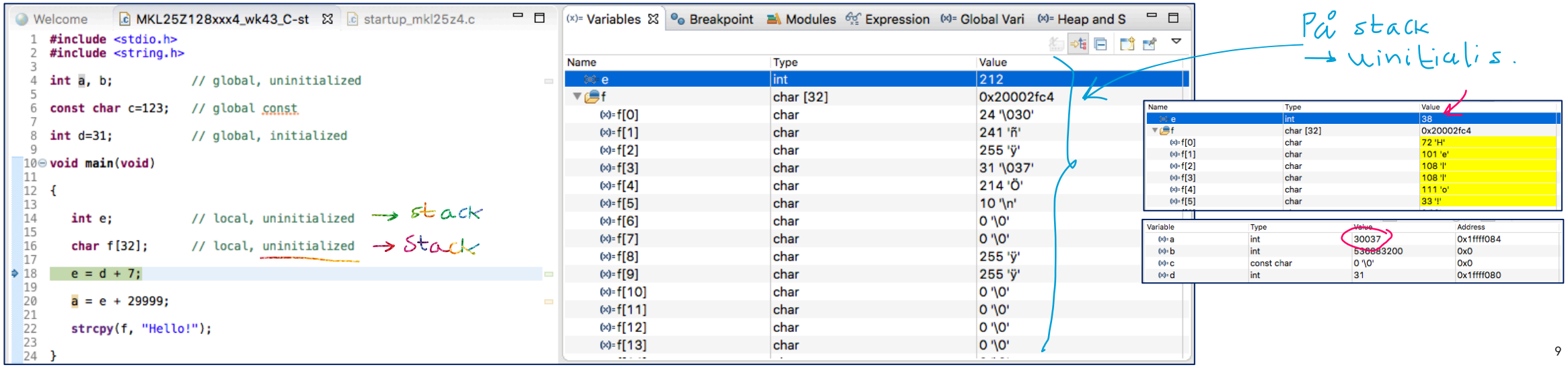


HW1: STARTUP OG MEMORY
EFTER INIT

Globale variable



Lokale variable



HW1: STARTUP OG MEMORY
BRUG AF `VOLATILE` QUALIFIER

Volatile

- Problem:
  - Optimizer prøver at være "smart" og spare load-operationer.
  - Men værdien i hukommelsen kan have ændret sig siden sidste load.
- Løsning
  - Tving compiler til at lave load-operation før hver anvendelse af variabel vha. `volatile` qualifier.
- Anvendelse:
  - Repræsentere hardwareregistre som variable (memory-mapped)
  - Hvor ISR-rutiner kan ændre værdier asynkront
  - Multi-trådet programmering

Kritiske sektioner -> Atomic operations til at undgå race conditions.

- Problem: Operationer i en kritisk sektion må ikke afbrydes. Data races / race conditions på ikke-atomisk data giver risiko for "korrumpering".
- Løsning: Gøre kritiske sektioner interrupt-fri og derved "atomiske".
- Forekomst: Vigtige sammenhængende ikke-atomiske operationer (fx datastrukturer) (sænker dog programmets responsiveness).

HW1: STARTUP OG MEMORY
HVAD ER EN STACK

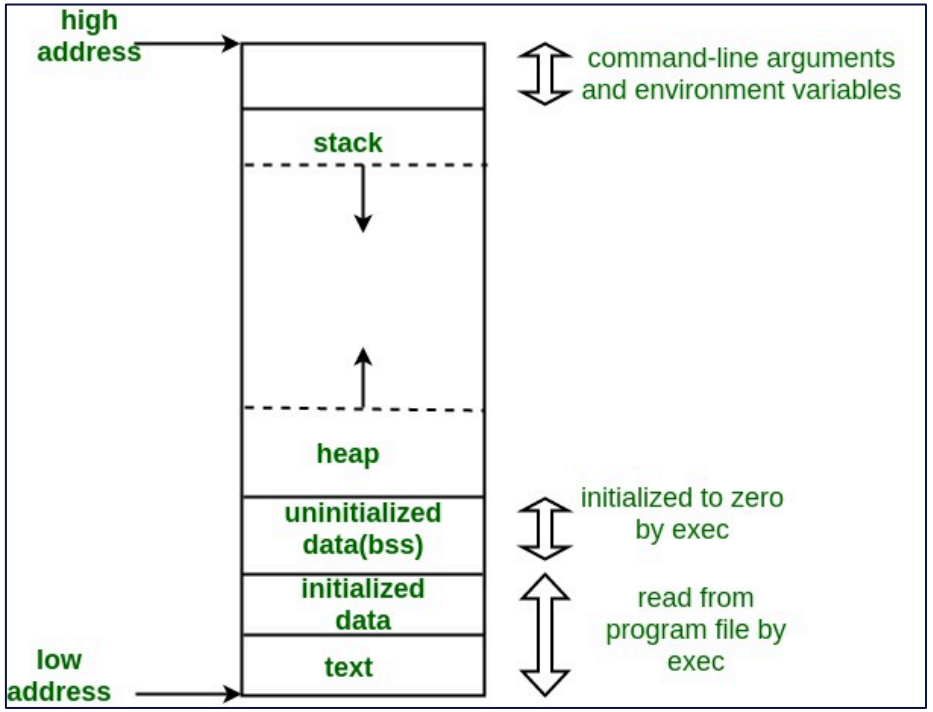
Datastruktur i RAM

Push og pop operationer

- Vokser ”oppe fra og ned”
- push (data på stack) mindsker SP
- pop (data fra stack) øger SP

Bruges bl.a. til

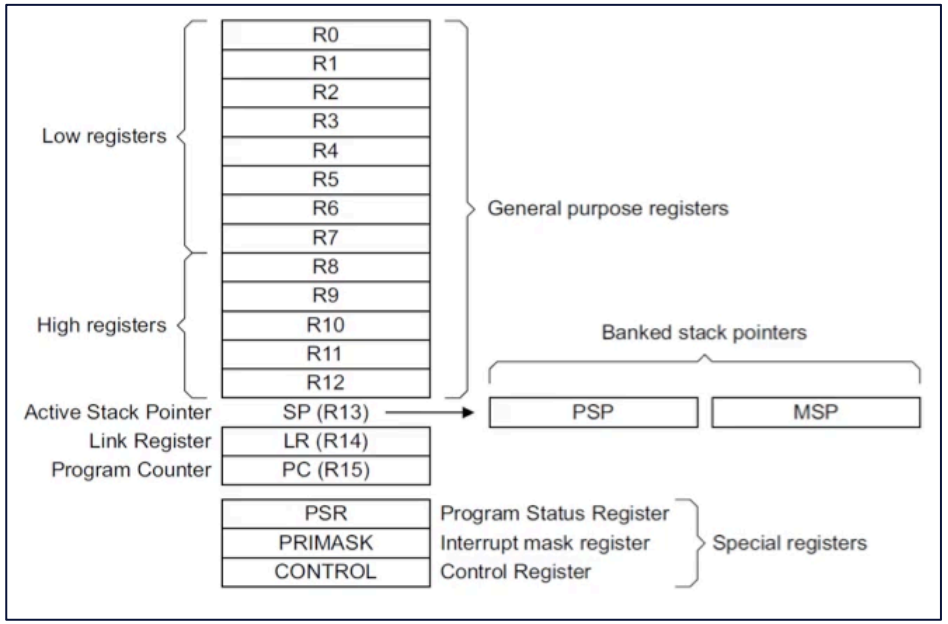
- Funktionkald (-> stack frame/activation record med lokale variable)
- Gemme kontekst (registre, osv.), når et interrupt får CPU’en til at skifte opgave.



HW1: STARTUP OG MEMORY
STACK – SP ER ET REGISTER, DER "PEGER"

Registre i CPU (Core registers i registerfil)

- **SP**: Stack-pointer – peger ud i RAM
  - Stack: Midlertidig lagring af variabler, m.m.
    - MSP (main..) og PSP (process..)
- 13 general purpose registers (R0-R12)
- **PC**: Program counter (R15)
- **LR**: Link (returadresse) (LR14)
- Kontrolregistre, bl.a.
  - PSR: APSR, IPSR, EPSR
  - PRIMASK: Masking af interrupts



HW1: EMNE
OPSUMMERING OG PERSPEKTIVERING

HW2: PERIPHERAL I/O
INDHOLD

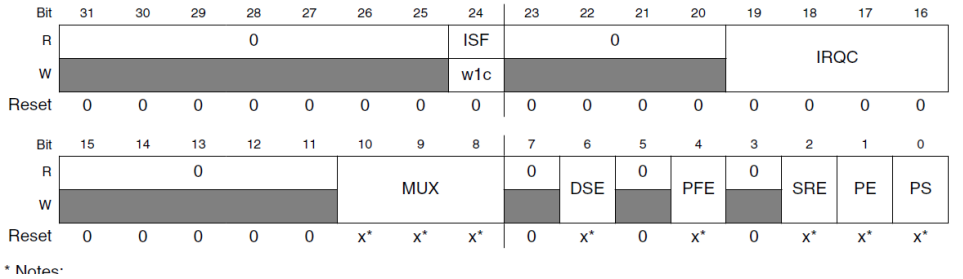
1. Hvordan sættes pins op?
2. Tilgang til et register; pointers og assembly
3. Logiske operatorer

2: Periphereal I/O

1. Forklar hvordan et enkelt ben på processoren sættes op til en given funktion, hvis du f.eks fra en SPI driver vil bruge processorens PORTE3 som SPI1_MISO.
2. Hvordan tilgår processoren et ***register***, f.eks. TPM0_SC, på adresse 0x40038000. Dvs. hvad oversættes dit C program til, og hvordan bruger processoren dette.
3. Hvad sker der i kode eksemplet nedenfor, hav særligt fokus på de logiske operatorer:

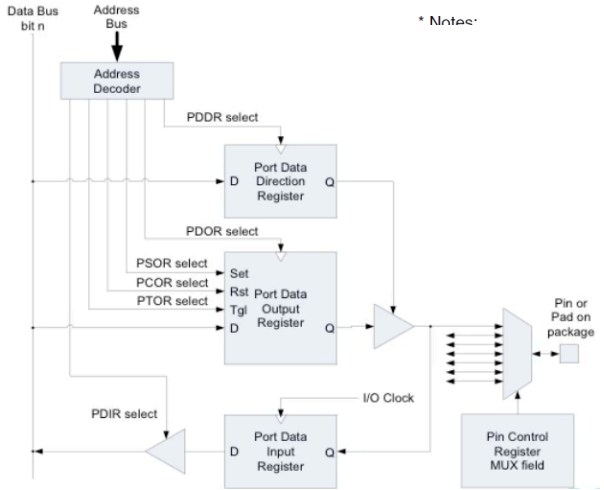
```
/* enable SPI1 & PORTE clock*/
SIM->SCGC4 |= SIM_SCGC4_SPI1_MASK;
SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;

/* disable SPI1 */
SPI1->C1 &= ~SPI_C1_SPE_MASK;

/* Setup pins */
/* E2 SPI1_SCK, alt2 */
PORTE->PCR[2] = (2 << 8);
```

HW2: PERIPHERAL I/O
OPSÆTNING AF PIN



Enkelt pins (ikke bus)

- Mux sætter fysisk forbindelse
- Pin control register MUX bits styrer funktion
- Forskellige pins har forskellige "muligheder" -> pinouts



MUX (bits 10-8)	Configuration
000	Pin disabled (analog)
001	Alternative 1 – GPIO
010	Alternative 2
011	Alternative 3
100	Alternative 4
101	Alternative 5
110	Alternative 6
111	Alternative 7

For GPIO pins er der forskellige veje

- Normal GPIO (med DMA-adgang)
- FGPIO (FasterGPIO)
  - Sparer CPU-cycles. Lader I/O tage en kortere vej
  - Ingen DMA

FRDM-KL25Z Pins				KL25Z128 Pins									
On-board Usage	I/O Header & Pin Num	Arduino™ R3 Pin Name	FRDM-KL25Z Pin Name	KL25Z Pin #	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	Reset State/Function
--	J2 20	D14	PTE0	1		PTE0		UART1_TX	RTC_CLKOUT	CMP0_OUT	I2C1_SDA		DISABLED
--	J2 18	D15	PTE1	2		PTE1	SPI1_MOSI	UART1_RX		SPI1_MISO	I2C1_SCL		DISABLED
--	J5 09	--	PTE2	3		PTE2	SPI1_SCK						DISABLED
--	J5 11	--	PTE3	4		PTE3	SPI1_MISO			SPI1_MOSI			DISABLED
--	J5 13	--	PTE4	5		PTE4	SPI1_PCS0						DISABLED
--	J5 15	--	PTE5	6		PTE5							DISABLED
Power	--	--	VDD	7	VDD								VDD
Power	--	--	VSS	8	VSS								VSS
USB	--	--	USB0_DP	9	USB0_DP								USB0_DP
USB	--	--	USB0_DM	10	USB0_DM								USB0_DM
2.2uF cap	--	--	VOUT33	11	VOUT33								VOUT33
USB VBUS (5V)	--	--	VREGIN	12	VREGIN								VREGIN
--	J10 01	--	PTE20	13	ADC0_DP0/ADC0_SE0	PTE20		FTM1_CH0	UART0_TX				ADC0_DP0/ADC0_SE0
--	J10 03	--	PTE21	14	ADC0_DM0/ADC0_SE4a	PTE21		FTM1_CH1	UART0_RX				ADC0_DM0/ADC0_SE4a
--	J10 05	--	PTE22	15	ADC0_DP3/ADC0_SE3	PTE22		FTM2_CH0	UART2_TX				ADC0_DP3/ADC0_SE3
--	J10 07	--	PTE23	16	ADC0_DM3/ADC0_SE7a	PTE23		FTM2_CH1	UART2_RX				ADC0_DM3/ADC0_SE7a
Power	--	--	VDDA	17	VDDA								VDDA
Power	J2 16	AREF*	VREFH	18	VREFH								VREFH
Power	--	--	VREFL	19	VREFL								VREFL
Power	--	--	VSSA	20	VSSA								VSSA
--	J10 09	--	PTE29	21	CMP0_IN5/ADC0_SE4b	PTE29		FTM0_CH2	FTM_CLKIN0				CMP0_IN5/ADC0_SE4b
--	J10 11	--	PTE30	22	DAC0_OUT/ADC0_SE23/CMP0_IN4	PTE30		FTM0_CH3	FTM_CLKIN1				DAC0_OUT/ADC0_SE23/CMP0_IN4

HW2: PERIPHERAL I/O
OPSÆTNING AF PIN

Eksempel

- LCD-øvelse
- 5 keys til 5 funktioner

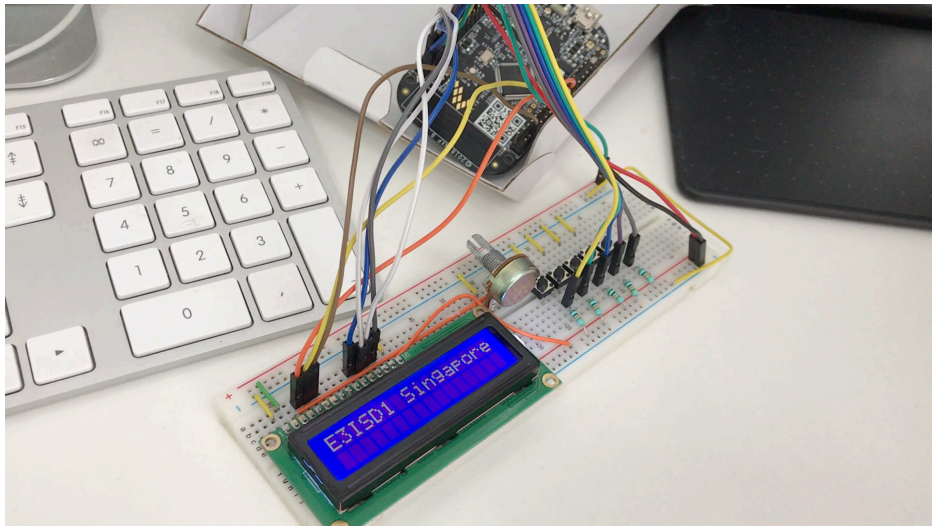
Sætter de 5 keys op

- Clock
- Clear MUX
- Sætter alternativ 1 (GPIO)
- Sætter data direction til input

Aftastning:

- Inverterer (aktiv lav)
- Tjekker ift mask
- Opdaterer array

```
18  /* All switches are on port E */
19  #define SW_UP_POS (21)
20  #define SW_DN_POS (29)
21  #define SW_LT_POS (30)
22  #define SW_RT_POS (23)
23  #define SW_CR_POS (22)
24
25  #define NUM_KEYS (5)
26
27  typedef enum {SWUp=0, SWDn, SWLt, SWRt, SWCr} sw_t;
28
29  /* Define macro to read switches */
30  #define READ_SWITCHES (PTE->PDIR)
31
32  void setup_keys(void) {
33
34      // Enable clock for GPIO keys
35      SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK; // Connect clock to Port E
36
37      // Set 5 switches pins as GPIO
38      PORTE->PCR[SW_UP_POS] &= ~PORT_PCR_MUX_MASK;
39      PORTE->PCR[SW_UP_POS] |= PORT_PCR_MUX(1);
40
41      PORTE->PCR[SW_DN_POS] &= ~PORT_PCR_MUX_MASK;
42      PORTE->PCR[SW_DN_POS] |= PORT_PCR_MUX(1);
43
44      PORTE->PCR[SW_LT_POS] &= ~PORT_PCR_MUX_MASK;
45      PORTE->PCR[SW_LT_POS] |= PORT_PCR_MUX(1);
46
47      PORTE->PCR[SW_RT_POS] &= ~PORT_PCR_MUX_MASK;
48      PORTE->PCR[SW_RT_POS] |= PORT_PCR_MUX(1);
49
50      PORTE->PCR[SW_CR_POS] &= ~PORT_PCR_MUX_MASK;
51      PORTE->PCR[SW_CR_POS] |= PORT_PCR_MUX(1);
52
53
54      /* Clear each SW to be input */
55      PTE->PDDR &= ~MASK(SW_UP_POS);
56      PTE->PDDR &= ~MASK(SW_DN_POS);
57      PTE->PDDR &= ~MASK(SW_LT_POS);
58      PTE->PDDR &= ~MASK(SW_RT_POS);
59      PTE->PDDR &= ~MASK(SW_CR_POS);
60  }
```




```
 99  /* writes to the passed array */
100  void read_keys(unsigned key[NUM_KEYS]) {
101
102      // switches are active low
103      // get the 32-bit input register for PORT E and negate.
104      // Should have 1s where switches are pressed.
105      unsigned switch_code = ~READ_SWITCHES;
106
107      //Detect pressed keys
108      key[SWUp] = switch_code & MASK(SW_UP_POS) ? 1 : 0;
109      key[SWDn] = switch_code & MASK(SW_DN_POS) ? 1 : 0;
110      key[SWLt] = switch_code & MASK(SW_LT_POS) ? 1 : 0;
111      key[SWRt] = switch_code & MASK(SW_RT_POS) ? 1 : 0;
112      key[SWCr] = switch_code & MASK(SW_CR_POS) ? 1 : 0;
113
114
115  }
```

```
29  /* Define macro to read switches */
30  #define READ_SWITCHES (PTE->PDIR)
```

HW2: PERIPHERAL I/O
TILGÅ REGISTER

Processor-manual viser adresser i memory-mapped I/O

Defineret i MKL25Z4.h:

- TPM0_SC status and control register er på adressen 0x40038000
  - Laver en pointer til dette sted, som har en struktur som TPM_Type
  - Først 32 bits er SC
  - TPM0_SC->SC er pointer dereferencing

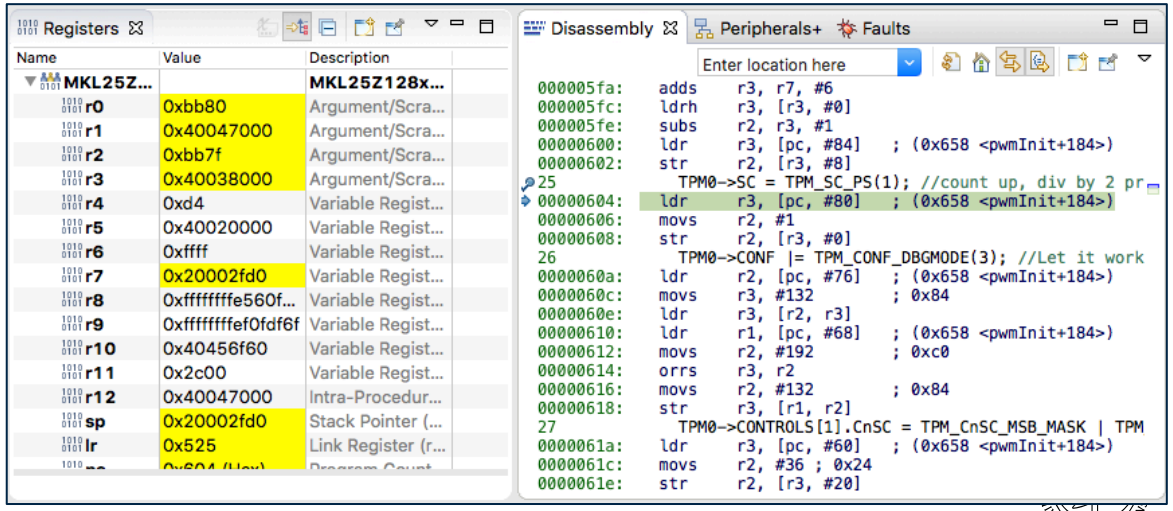
  - Assembly:
    - R3 holder ønsket adresse:
      - LDR (load with immediate offset) en værdi fra mem, pc + 80
    - Gem "1" i R2
    - STR (store with imm. offset): Gemmer registerværdi R2 i hukommelseslokation, som findes i register R3 med offset "0".

TPM memory map

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
4003_8000	Status and Control (TPM0_SC)	32	R/W	0000_0000h	31.3.1/552
4003_8004	Counter (TPM0_CNT)	32	R/W	0000_0000h	31.3.2/553
4003_8008	Modulo (TPM0_MOD)	32	R/W	0000_FFFFh	31.3.3/554

```
4309  /* TPM - Peripheral instance base addresses */
4310  /** Peripheral TPM0 base address */
4311  #define TPM0_BASE                                (0x40038000u)
4312  /** Peripheral TPM0 base pointer */
4313  #define TPM0                                     ((TPM_Type *)TPM0_BASE)
```

```
4169  /** TPM - Register Layout Typedef */
4170  typedef struct {
4171    __IO uint32_t SC;                                /**< Status and Control, offset: 0x0 */
4172    __IO uint32_t CNT;                               /**< Counter, offset: 0x4 */
4173    __IO uint32_t MOD;                               /**< Modulo, offset: 0x8 */
4174    struct {                                         /* offset: 0xC, array step: 0x8 */
4175      __IO uint32_t CnSC;                              /**< Channel (n) Status and Control, array offset:
4176      __IO uint32_t CnV;                               /**< Channel (n) Value, array offset: 0x10, array
4177    } CONTROLS[6];
4178        uint8_t RESERVED_0[20];
4179    __IO uint32_t STATUS;                            /**< Capture and Compare Status, offset: 0x50 */
4180        uint8_t RESERVED_1[48];
4181    __IO uint32_t CONF;                              /**< Configuration, offset: 0x84 */
4182  } TPM_Type;
```



HW2: PERIPHERAL I/O
"AFKODNING" AF EKSEMPEL

SPI1-modulet forbundet til System Clock Gating Control 4 - Vi skal have tændt for clock til det (slukket for at spare energi)

Bitwise samtidig OR og tilskrivning
hs = hs | vs
Så flipper bit højt, uanset om den var høj i forvejen

Mask makrovariabel med bit på 1 lokation

Port E pins forbundet til SCGC5 Ligesom med SPI skal der tændes for clock for at portmodulet virker.

Mask makrovariabel med bit på 1 lokation

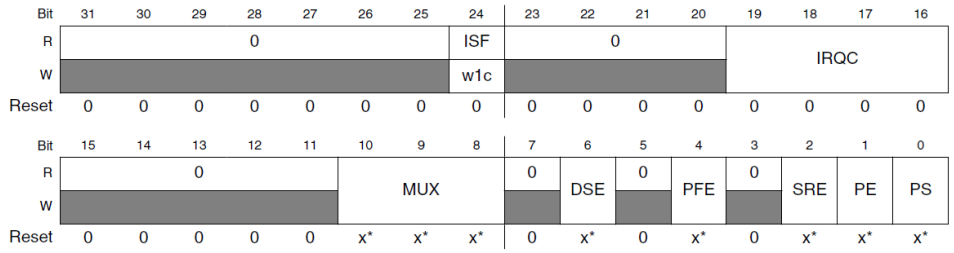
```
/* enable SPI1 & PORTE clock*/
SIM->SCGC4 |= SIM_SCGC4_SPI1_MASK;
SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;

/* disable SPI1 */
SPI1->C1 &= ~SPI_C1_SPE_MASK;

/* Setup pins */
/* E2 SPI1_SCK, alt2 */
PORTE->PCR[2] = (2 << 8);
```

Denne operation inverterer mask, så fx ~(00..010..00) -> (11..101..11)
Bitwise AND med tilskrivning søger for at alt andet bliver som det var før (1&1=1, 0&1=0), kun C1 i registret bliver sat til 0.

PTE2 MUX-bits bliver sat til værdien 2 (0b010), det er alternativ 2, altså SPI1_SCK, som er SPI1-clock...

Hvorfor? Nok for at kunne måle på den via et oscilloskop / analog discovery...

Shifter værdien 2 (0b010) 8 pladser "til venstre)... Svarer til eksponentiering 2^8. Således bliver bits 10, 9 og 8 sat



	KL25Z128 Pins				
FRDM-KL25Z Pin Name	KL25Z Pin #	ALT0	ALT1	ALT2	ALT3
PTE2	3		PTE2	SPI1_SCK	

HW2: PERIPHERAL I/O
OPSUMMERING OG PERSPEKTIVERING

HW3: INTERRUPTS
INDHOLD

1. Hvorfor interrupts? Hvorfor ikke bare polling?
2. Race conditions
3. Kort ISR – et eksempel

3: Interrupts

1) Hvad er ***race-conditions***, og hvordan tages hensyn til disse ?
2) Antag at du har en digital indgang, GPIO, hvor du ønsker at koble en trykknap på. Forklar forskellen på at anvende hhv. ***Polling*** og ***Interrupts*** til at reagere på et knaptryk.
3) Hvorfor ønsker vi at en *Interrupt Service Routine* - ***ISR*** skal være kort / hurtig afviklet

HW3: INTERRUPTS
HVORFOR INTERRUPTS? IKKE POLLING?

Responsive systemer

- Absolutte deadlines (sikkerhed)
- Usability og UX

Polling er busy waiting med meget høj CPU overhead, spilder bare clock cycles, energi, processor-tid på at vente

Efficiens: Concurrency, dele processorens (tid), scheduling

- Aflaste software med hardware -> event-baseret, interrupts
    - Responsiveness (~15 clock cycles fra IRQ til ISR-afvikling i teorien)

Løsninger:

- Hardware: Interrupts og event-triggering
- Software: Scheduling (statisk/kooperativt og dynamisk m/u task-level preemption)

HW3: INTERRUPTS
RACE CONDITIONS, DATA RACES

Overordnet: Flere operationer fra forskellige sektioner af et program prøver at modificere / læse på data.

- Fx en ISR, der afbryder afvikling af anden kode. Måske ikke atomisk. Måske ved den anden kode ikke, at forudsætninger er ændret.

Løsning: Kritiske sektioner -> Atomic operations til at undgå race conditions.

- Problem: Operationer i en kritisk sektion må ikke afbrydes. Data races / race conditions på ikke-atomisk data giver risiko for "korrumpering".
- Løsning: Gøre kritiske sektioner interrupt-fri og derved "atomiske".
- Forekomst: Vigtige sammenhængende ikke-atomiske operationer (fx datastrukturer) (sænker dog programmets responsiveness).

Volatile *kan* være en anden løsning...

- Problem:
  - Optimizer prøver at være "smart" og spare load-operationer.
  - Men værdien i hukommelsen kan have ændret sig siden sidste load.
- Løsning
  - Tving compiler til at lave load-operation før hver anvendelse af variabel vha. `volatile` qualifier.
- Anvendelse:
  - Repræsentere hardwareregistre som variable (memory-mapped)
  - Hvor ISR-rutiner kan ændre værdier asynkront
  - Multi-trådet programmering

HW3: INTERRUPTS
INTERRUPT LATENCY

Øvelse uge 40:

- Fra tryk på knap til ISR kører, ca. 690 ns.
- @48 MHz svarer det til ca. 33 clock cycles.
- Noget længere end de 15-16 cycles nævnt.

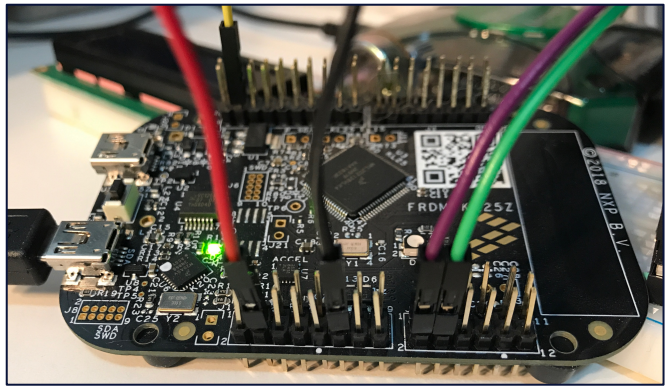
Mulige forklaringer

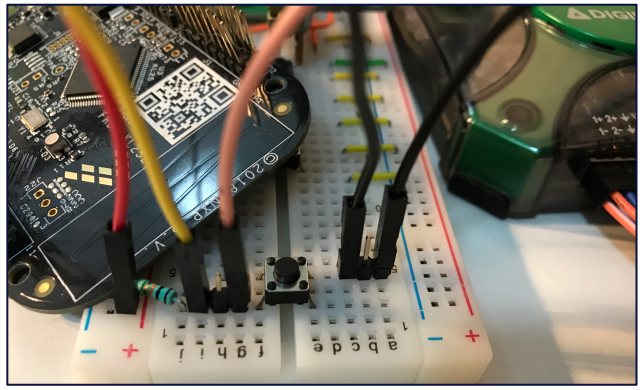
- Leadtime fra ISR startes til GPIO-signal aktivt.
- Ventetid på at gemme kontekst på stack.
- CPU/bus, mv. kører ikke 48 MHz, ej optimeret.

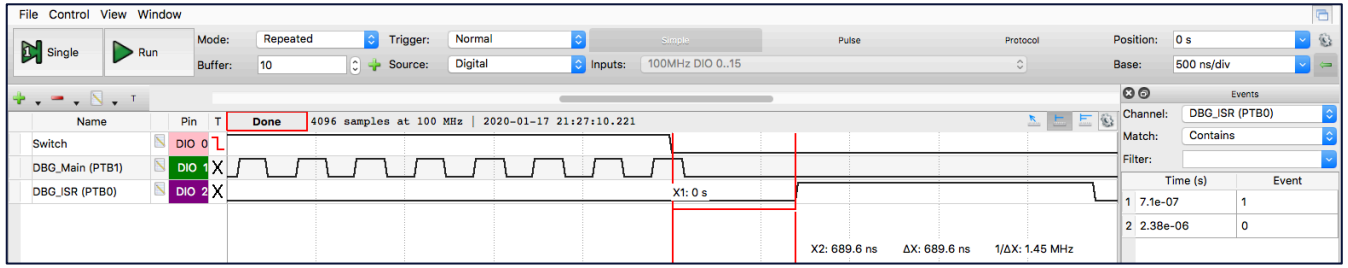
Når kode optimeres (-O3) forbedres med 77 ns.

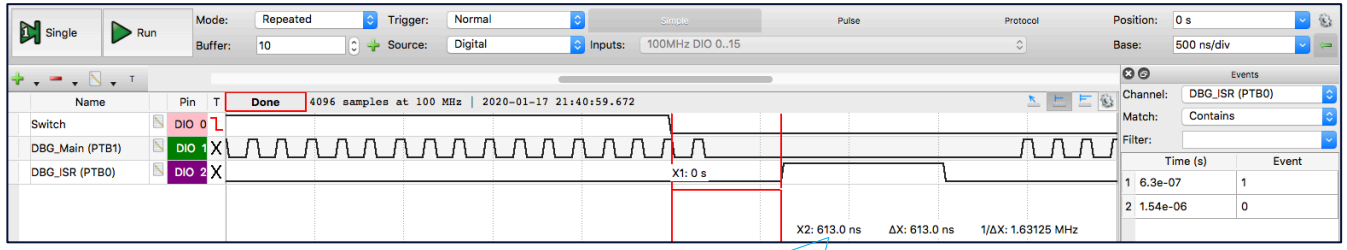
- Bemærk også ændret frekvens på DBG_Main og kortere ISR-tid.

Måske behøver vi ikke afbryde? Brug en hardwaretimer/counter i baggrunden til at tælle events...









Med –O3 forbedres med 77 ns

HW3
KORT ISR

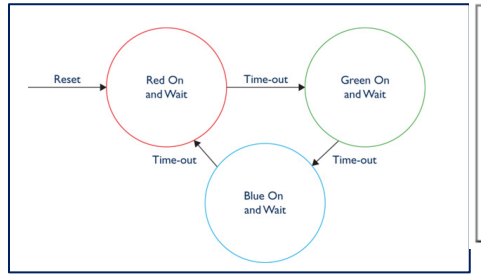
Kort ISR

- Lang ISR sænker "responsiveness" af resten af koden
  - Plads til nye interrupts - kan forsinke start af andre ISR'er (samtidigt opståede nu i prioritetslisten)
  - Performance/responsiveness vs. complexity
  - Evt. nestede interrupts / med prioriteter.
- Bliver nemt til spaghetti-kode
  - Svært at forstå, vedligeholde, udvikle på

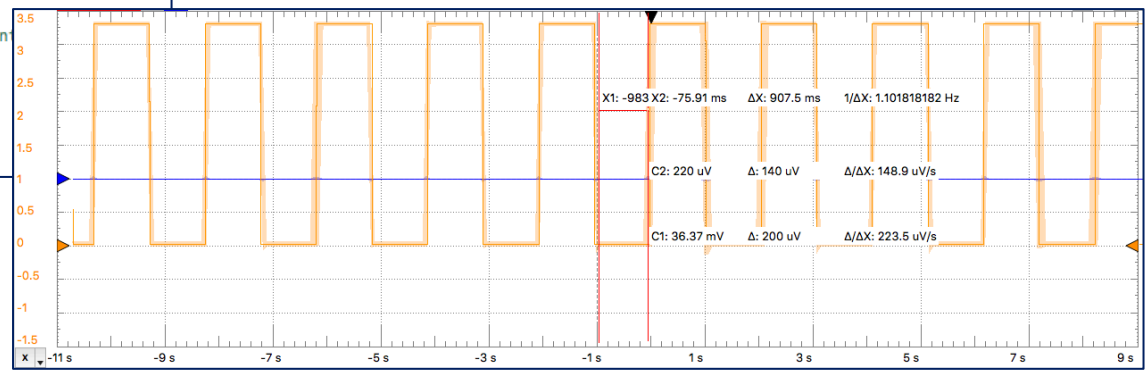
Eksempel: En nok lidt for lang ISR fra uge 38

- Burde flytte noget kode ud
- Rammer næsten den ønskede frekvens, men der er lidt forsinkelse...

```
void SysTick_Handler(void)
{
    // Variables for the state machine
    static state_t state = RED;  //Initialize as RED
    static state_t state_next;
    static unsigned long count = 0;

    // The LEDs are connected so high=off, low=on.
    switch (state) {
    case RED:

        PTB->PCOR = MASK(RED_LED_SHIFT);    // red LED on
        PTD->PSOR = MASK(BLUE_LED_SHIFT);   // blue LED off

        count++;                            // counter for times in case

        // Run the red light for 3 cycles of 100 ms / cycle = 300 ms
        if (count < 3) {
            state_next = RED;               // Go to this case again next
        } else {
            state_next = GREEN;
            count = 0;                      // Reset counter
        }

        break;

    case GREEN:
        PTB->PCOR = MASK(GREEN_LED_SHIFT);  // set the green LED
        PTB->PSOR = MASK(RED_LED_SHIFT);    // clear the red LED

        count++;

        // Run the green light for 6 cycles = 600 ms
        if (count < 6) {
            state_next = GREEN;             // Go to this case again next
        } else {
            state_next = BLUE;
            count = 0;                      // Reset counter
        }

        break;

    case BLUE:
        PTD->PCOR = MASK(BLUE_LED_SHIFT);   // set the blue LED
        PTB->PSOR = MASK(GREEN_LED_SHIFT);  // clear the green LED
        count++;

        // Run the blue light for 9 interrupt cycles = 900 ms
        if (count < 9) {
            state_next = BLUE;              // Go to this case again next
        } else {
            state_next = RED;
            count = 0;                      // Reset count
        }

        break;
    }

    state = state_next;
}
```



STATE	LED farve	TimeOut
RED	Rød	300[mS]
GREEN	Grøn	600[mS]
BLUE	Blå	900[mS]d



HW3: EMNE
OPSUMMERING OG PERSPEKTIVERING

HW4: TIMERS
INDHOLD

1. Perifere enheder med timer-funktioner
2. Interrupts med en timer + håndtering
    1. SysTick-eksempel (uge 46)
    2. Eksempel med FSM-håndtering (uge 38)
3. Opsætning af TPM
    1. Oversigt over registre
    2. PWM-eksempel (uge 46)
4. Opsummering og perspektivering

4: Timers

1) Redegør for hvordan et timermodul, som KL25's TPM (Timer/PWM Module) kan anvendes, f.eks. til ***periodisk interrupt***, ***PWM*** eller ***input capture***.
2) Giv et overblik over hvilke perifer-enheder (Periphereals) i KL25 processoren der indeholder timer-funktioner
3) Forklar hvordan en timer kan bruges til at generere periodiske interrupts, og hvordan disse kan håndteres, f.eks. hvis du skal bruge en 10ms tællevariabel.

HW4: TIMERS
PERIFERE TIMERENHEDER PÅ KL25Z-BOARD

Formål:

- Hardware aflaster software
- Tælle tid / events
  - Clocked synkront / asynkront
- Beregninger på signal, IRQ, PWM, osv.

Perifere enheder:

- TPM: Timer/PWM
- PIT: Intervaller
- LPTMR: Low-power TMR
- RTC: Real-time Clock
- Watchdog: Fejldetektion. Holder øje med CPU ”heartbeat” -> reset vektor
- SysTick (Cortex-M0)

HW4: TIMERS
INTERRUPTS MED EN TIMER

SysTick driver uge 46

Bruger Cortex-M0+'s SysTick til at generere interrupts med 1500 ms interval

- initSystickTmr
- ISR, toggler
- Beregning af loadværdi
- Resten er standard / loop.

Med Analog Discovery dok. interval på 1500 ms.

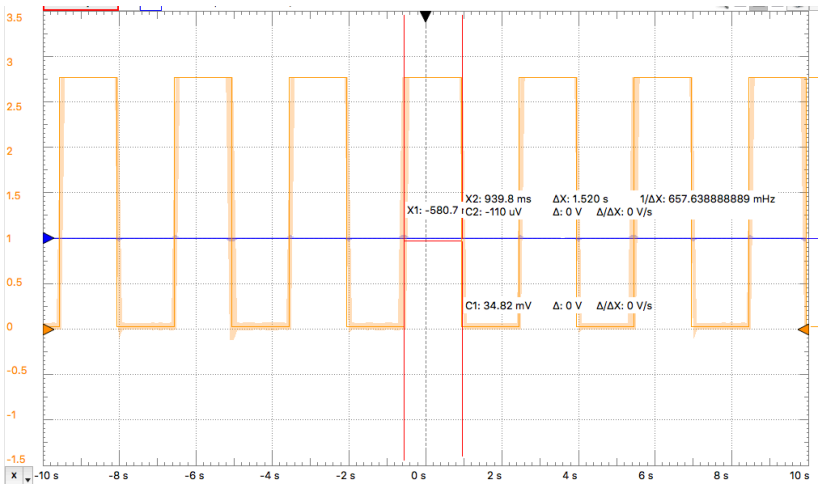
10 ms? Genberegn LDV:

- LDV = 10e-3 / 3e-6 -1
- LDV = 30e3 - 1

```c
/**
 * @file SysTick.c
 * @brief E3ISD1 Week 46: Set up SysTick timer to run at 3 MHz and load
 * compare register
 * @date 2019
 * @author Janus Bo Andersen
 * Function for setting up SysTick and corresponding interrupt,
 * also with 'ticks' valued as trigger threshold for triggering interrupt
 *
 * @note timer is started after calling this function
 * @param ticks number of ticks, counting at 3 MHz, before triggering input
 * @return 0 on success, -1 if ticks outside range
 *
 */

#include "SysTick.h"

int initSystickTmr(unsigned int ticks) {
    /* Based on Dean p. 189 */

    // Check valid input, already guaranteed unsigned, must fit in 24-bit (2^24-1)
    if (ticks == 0 || ticks > 16777215) {
        return -1;
    }

    // LDV = T_target / T_counter -1.
    SysTick->LOAD = (ticks);

    // Set interrupt
    NVIC_SetPriority(SysTick_IRQn, 3);

    // Start at zero so it initially reloads
    SysTick->VAL = 0;

    // Note: We use the alternate clock source
    // (48 MHz/16 = 3 MHz), so CLKSOURCE bit is 0
    SysTick->CTRL = SysTick_CTRL_TICKINT_Msk | // Set exception on reaching zero
                    SysTick_CTRL_ENABLE_Msk;   // Enable the SysTick

    return 0;

}
```

```c
void SysTick_Handler() {
    // Toggle the blue LED
    PTD->PTOR = MASK(BLUE_LED_SHIFT);
}
```

```c
    // LDV = T_target / T_counter -1 = 1.5s / (1 / 3MHz) = 4.5e6.
    const unsigned int LDV = 4500000L-1L;

    if ( !initSystickTmr(LDV) ) {
        //error occurred
        return -1;
    }
```



HW4: TIMERS
HÅNDTERING AF TIMERINTERRUPTS MED FSM

FSM ISR uge 38

SysTick hver 10 ms

SysTick ISR indeholder FSM

- 3 tilstande
- Hver tilstand har forskellig varighed
- Designet kan forbedres!
  - Ryk kode ud af ISR...

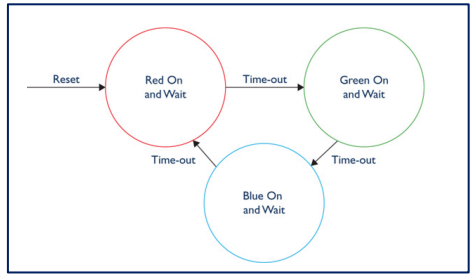
Benyttede ikke egen driver, men den fra core_cm0plus.h

- CLKSRC=1, så der burde benyttes processorclock (48 MHz), men den kører faktor-2 langsommere

```c
void SysTick_Handler(void)
{
    // Variables for the state machine
    static state_t state = RED;  //Initialize as RED
    static state_t state_next;
    static unsigned long count = 0;

    // The LEDs are connected so high=off, low=on.
    switch (state) {
    case RED:

        PTB->PCOR = MASK(RED_LED_SHIFT);    // red LED on
        PTD->PSOR = MASK(BLUE_LED_SHIFT);   // blue LED off

        count++;                            // counter for times in case

        // Run the red light for 3 cycles of 100 ms / cycle = 300 ms
        if (count < 3) {
            state_next = RED;               // Go to this case again next
        } else {
            state_next = GREEN;
            count = 0;                      // Reset counter
        }

        break;

    case GREEN:
        PTB->PCOR = MASK(GREEN_LED_SHIFT);  // set the green LED
        PTB->PSOR = MASK(RED_LED_SHIFT);    // clear the red LED

        count++;

        // Run the green light for 6 cycles = 600 ms
        if (count < 6) {
            state_next = GREEN;             // Go to this case again next
        } else {
            state_next = BLUE;
            count = 0;                      // Reset counter
        }

        break;

    case BLUE:
        PTD->PCOR = MASK(BLUE_LED_SHIFT);   // set the blue LED
        PTB->PSOR = MASK(GREEN_LED_SHIFT);  // clear the green LED
        count++;

        // Run the blue light for 9 interrupt cycles = 900 ms
        if (count < 9) {
            state_next = BLUE;              // Go to this case again next
        } else {
            state_next = RED;
            count = 0;                      // Reset counter
        }

        break;
    }

    state = state_next;
}
```

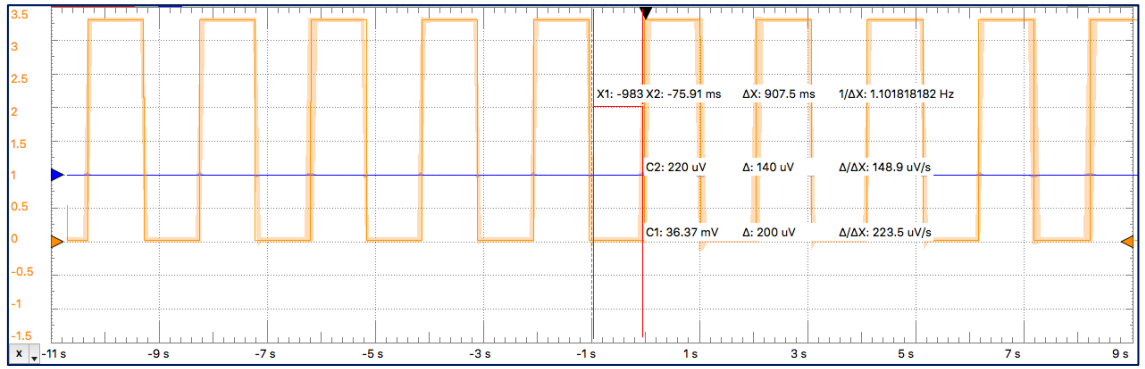
```c
typedef enum {
    RED,
    GREEN,
    BLUE
} state_t;
```

```c
    /*100 ms = 10 interrupts pr. second. Fcpu / 10 = sysTickConfigValue */
    /* Was 4800000, but there is some factor clock scaling going on here */
    SysTick_Config(2400000);
```



STATE	LED farve	TimeOut
RED	Rød	300[mS]
GREEN	Grøn	600[mS]
BLUE	Blå	900[mS]d

```c
__STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks)
{
  if ((ticks - 1UL) > SysTick_LOAD_RELOAD_Msk)
  {
    return (1UL);                                           /* Reload value impossible */
  }

  SysTick->LOAD  = (uint32_t)(ticks - 1UL);                 /* set reload register */
  NVIC_SetPriority (SysTick_IRQn, (1UL << __NVIC_PRIO_BITS) - 1UL); /* set Priority for Systick Interrupt */
  SysTick->VAL   = 0UL;                                     /* Load the SysTick Counter Value */
  SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
                   SysTick_CTRL_TICKINT_Msk   |
                   SysTick_CTRL_ENABLE_Msk;                 /* Enable SysTick IRQ and SysTick Timer */
  return (0UL);                                             /* Function successful */
}

#endif
```



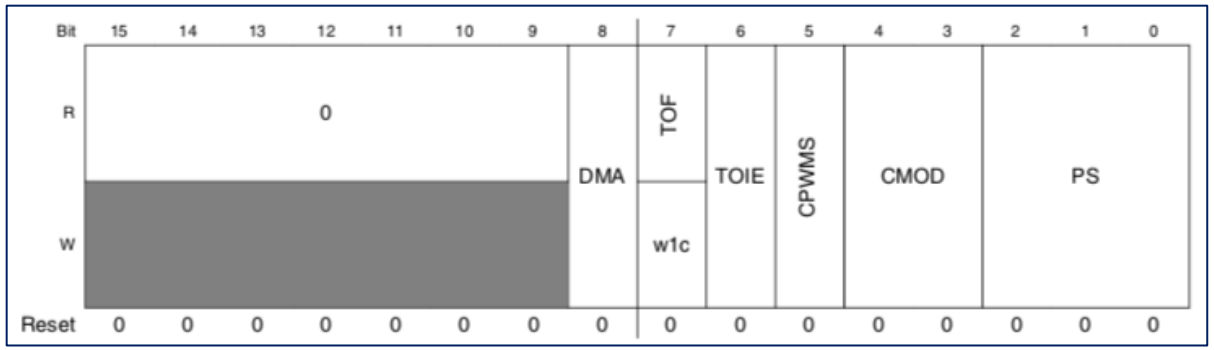
HW4: TIMERS
OPSÆTNING AF TPM

Konfiguration, der skal sættes for **TPM-modulet**

- Clock source (CMOD)
- Interrupt ved overflow (TOIE ved TOF)
- Tælleretning (CPWMS)
- Prescaling (PS)
- Free-running eller modulo tæller (TPMx_MOD)

Kanal-indstillinger – næste slide...

TPMx_SC (TPM0, TPM1, TPM2)

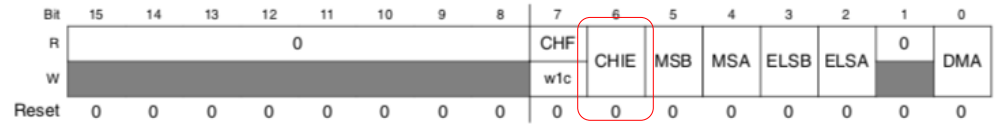


6 TOIE	Timer Overflow Interrupt Enable Enables LPTPM overflow interrupts. 0 Disable TOF interrupts. Use software polling or DMA request. 1 Enable TOF interrupts. An interrupt is generated when TOF equals one.

HW4: TIMERS
OPSÆTNING AF TPM

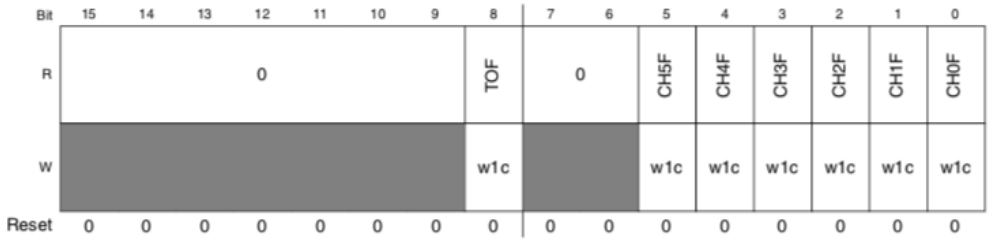
Konfiguration, der skal sættes for **kanalen**

- TPM0: 6 kanaler, TPM1 og TPM2: Hver 2 kanaler
- Register TPMx_C*n*SC:



- Interrupt på kanalen (CHIE)
- Mode (MSB, MSA)
- Edge select (ELSB, ELSA)

Alle flag kan ses i TPMx_STATUS



Konfiguration af en kanal

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
X	00	00	None	Channel disabled
X	01/10/11	00	Software compare	Pin not used for LPTPM
0	00	01	Input capture	Capture on Rising Edge Only
		10		Capture on Falling Edge Only
		11		Capture on Rising or Falling Edge
	01	01	Output compare	Toggle Output on match
		10		Clear Output on match
		11		Set Output on match
	10	10	Edge-aligned PWM	High-true pulses (clear Output on match, set Output on reload)
		X1		Low-true pulses (set Output on match, clear Output on reload)
	11	10	Output compare	Pulse Output low on match
		X1		Pulse Output high on match
1	10	10	Center-aligned PWM	High-true pulses (clear Output on match-up, set Output on match-down)
		X1		Low-true pulses (set Output on match-up, clear Output on match-down)

HW4: TIMERS
EKSEMPEL PÅ OPSÆTNING AF PWM

Uge 46:

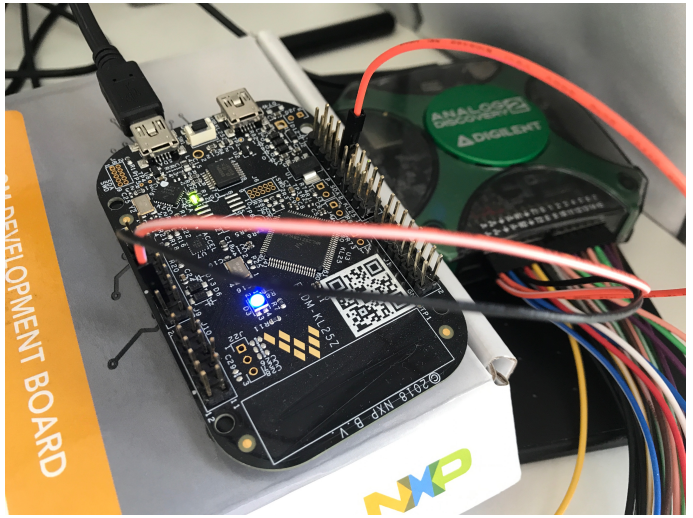
- Skrue op og ned for lysstyrke i LED
- Duty-cycle: CnV/MOD

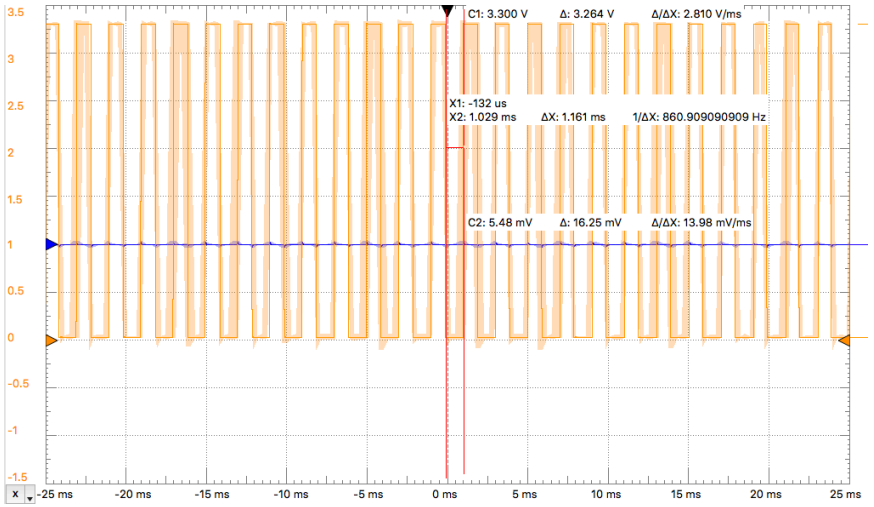
Puls ift. edge: start

Måler med Analog Discovery på D1 (blå diode)

```c
/**
 * @file pwm.c
 * @brief E3ISD1 Week 46: Set up PWM to manage LED brightness
 * @date E3ISD1 2019
 * @author Janus Bo Andersen
 * @note uint16_t because TPM MOD counter only supports 16 bit
 * @param period number of clock ticks in a period
 * @return 0 on success, -1 if ticks outside range
 *
 */
#include "pwm.h"

int pwmInit(uint16_t period) {

    SIM->SCGC5 |= SIM_SCGC5_PORTD_MASK; // Clock PTD1 (LED)
    SIM->SCGC6 |= SIM_SCGC6_TPM0_MASK;  // Clock TPM

    //Let's use FAST GPIO
    PORTD->PCR[BLUE_LED_POS] &= ~PORT_PCR_MUX_MASK;
    PORTD->PCR[BLUE_LED_POS] |= PORT_PCR_MUX(4);    //Alt. 4 FGPIO

    //Config TPM0
    SIM->SOPT2 |= (SIM_SOPT2_TPMSRC(1) | SIM_SOPT2_PLLFLLSEL_MASK); //48 MHz
    TPM0->MOD = period-1; // MOD counter mod
    TPM0->SC = TPM_SC_PS(1); //count up, div by 2 prescaler
    TPM0->CONF |= TPM_CONF_DBGMODE(3); //Let it work in debug mode
    TPM0->CONTROLS[1].CnSC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSA_MASK; //Pwm edge align
    TPM0->CONTROLS[1].CnV = 0; //initial duty cycle
    TPM0->SC |= TPM_SC_CMOD(1); //START :)

    return 0;
}
```

```c
    pwmInit(PWM_PERIOD);    //Initialize PWM using our driver
    while(1) {
        //First brighten the LED
        for (i=0; i<PWM_PERIOD; i++) {
            TPM0->CONTROLS[1].CnV = i;

            //wait a while
            for (delay=0; delay<100; delay++) {
                //pass
            }
        }
        //Then fade the LED
        for (i=PWM_PERIOD-1; i>0; i--) {
            TPM0->CONTROLS[1].CnV = i;

            //wait a while
            for (delay=0; delay<100; delay++) {
            //pass
            }
        }

    }
```





HW4: TIMERS
OPSUMMERING OG PERSPEKTIVERING

HW5: SERIEL KOMMUNIKATION
INDHOLD

1. Synkron vs. asynkron kommunikation
    1. Eksempel
2. Overblik over seriel kommunikation (væsentlige karakteristika ved I2C, SPI og UART
3. Forskelle på fuld duplex og halv duplex
    1. Eksempel
4. Opsummering og perspektivering

5: Seriel kommunikation

1) Hvad er de væsentlige forskelle der adskiller hhv. synkron/asynkron kommunikation, giv et eksempel.
2) Hvad er de væsentlige forskelle på full- og half- duplex, giv et eksempel.
3) Redegør for væsentlige karakteristika ved hhv. SPI/I2C/UART.

HW5: SERIEL KOMMUNIKATION
SYNKRON VS ASYNKRON

Synkron:

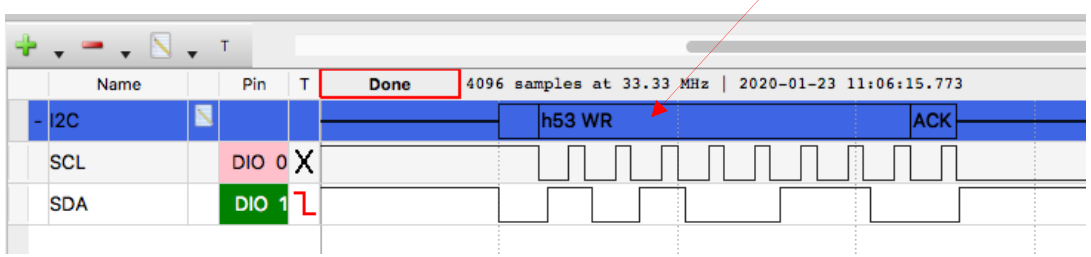
- Fælles clock-signal
- Koordineret af master eller ekstern clock
- Eksempel: SPI, I2C

Asynkron:

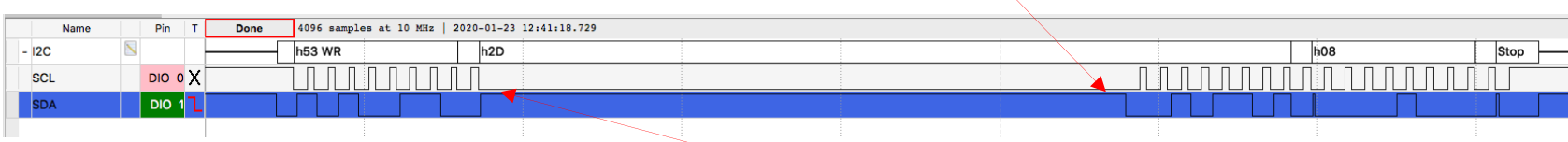
- Kommunikationshastighed aftalt på forhånd (baud)
- Krav til nogenlunde ens intern clockfrekv.
- Eller sat som standardhastighed
- Eksempel: UART (selvom der findes USART)
- Forskellige protokoller

Begge typisk aktiv lav

Eksempel: Kommunikation med et ADXL345-modul
- SCL er clock
- Master trækker SDA lav for at initiere
- 0x53 addresseres



Eksempel: Kommunikation med et ADXL345-modul
- Initiering og bit overføres
- Registre aflæses



Eksempel: Når clock stopper og SDA går høj, så er kommunikationen slut

HW5: SERIEL KOMMUNIKATION

SERIEL KOMMUNIKATION

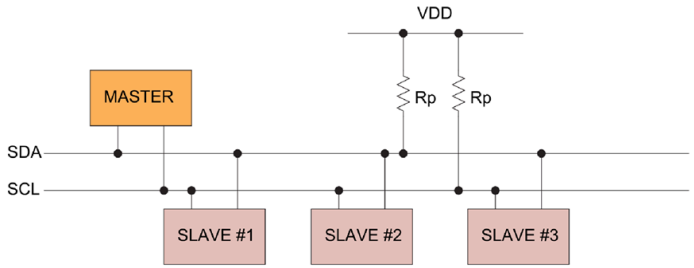
I2C

Seriel-bus, hardware og protokol (half-duplex)

- 2 linjer: SDA, SCL (aktiv lav)
- Synkron (clocked, SCL)
- Philips / NXP specifikation
- Hurtigere end UART, langsommere end SPI

Master-slaves

- 7-bit-adresser (128 enheder, men...)
- Clock-synkronisering fra master
  - Clock-stretching fra slave
- Multi-master via "collision detection" og "arbitration".



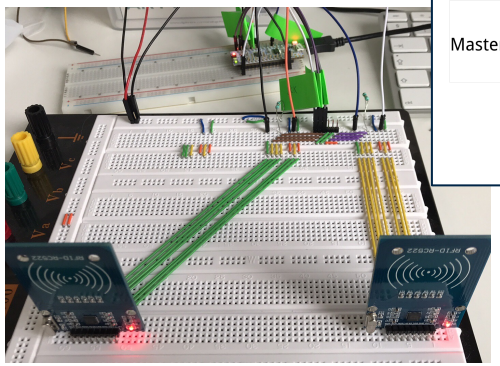
SPI

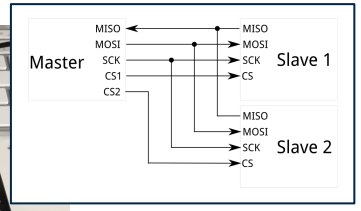
Seriel-bus, hardware og protokol (full-duplex)

- 2+N linjer: MOSI, MISO, SCK, *N x* SS/CS
- Synkron (clocked, SCK)
- Master-slave(s), kun 1 master
- Hurtigste af I2C, UART og SPI

Eksempel:

- Øvelse 20b
- Pro2, flere RFID-læsere i "samme hus"

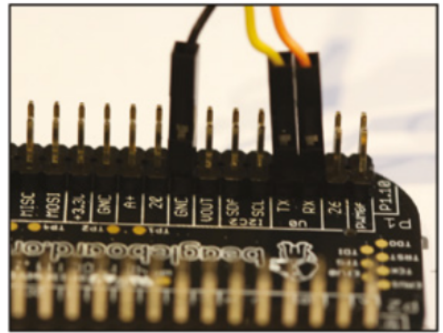




UART

Seriel-forbindelse (half-duplex)

- Point-to-point (UART1 <-> UART2)
  - Rx, Tx linjer på hvert punkt
  - Asynkron: Ingen fælles clock (intern baud rate)
    - Hastighed (<115200 baud) skal være ens I hver ende
    - Start/stop-bits, og maks. længde af databits (5-9) -> typisk 8 og paritet
  - Hardware: Parallel databus -> skifteregister -> Tx -> serielforb. -> Rx, osv...
  - Oversampling til fx støjreduk.
- Kan implementere forskellige kommunikationsprotokoller, xx RS-232
- Langsommest af de tre
- Men simpel!



HW5: SERIEL KOMMUNIKATION
HALV / FULD DUPLEX

Halv duplex

Kun en kanal, kan benyttes af en ad gangen

- Enten Tx eller Rx

Eksempler:

- UART med ét skifteregister:
  - Skifteregister bruges til en Tx eller Rx, FIFO-buffer afkobler
- I2C:
  - Kun en bus, og kun 1 enhed kan addresseres ad gangen

Simplex:

- Kun en kan overføre på kanalen, den anden kan kun lytte.

Fuld duplex

To kanaler eller forskellige frekvensbånd

- Samtidig Tx og Rx på to forskellige kanaler/bånd

Eksempel:

- UART med to skifteregistre
  - Et til Tx og et til Rx, to FIFO buffers afkobler
- SPI: Fuld-duplex, fordi der er både MOSI- og MISO-kanaler
  - Dog kun 1 slave ad gangen (kun 1 CS/SS)
- ADSL:
  - Frekvensopdeling
- Oplagt eksempel:
  - Telefoner: Både digitale/mobiler og de gamle analoge

Ikke-serielle:

- Parallel databus

HW5: SERIEL KOMMUNIKATION
EKSEMPEL SPI

Serielt: En linje til hver dataretning (ind/ud)

Synkront (clock)

- SPI: SCK

Databus

- SPI: MISO, MOSI

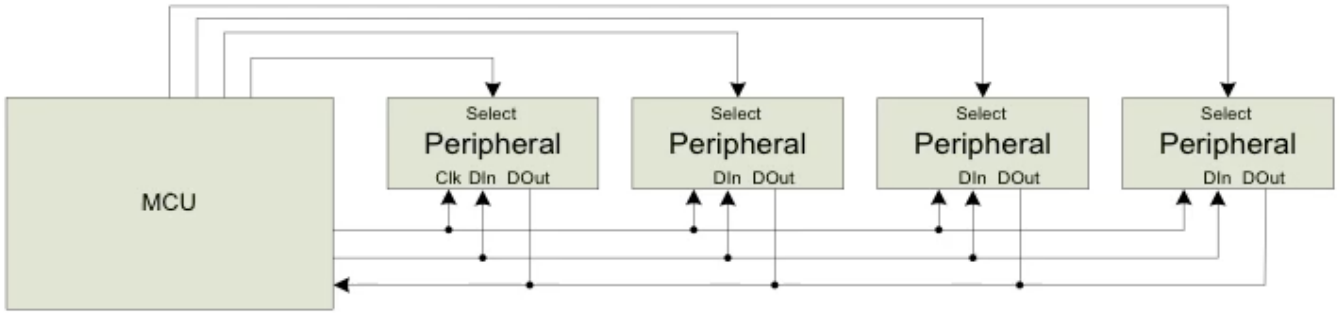
Select:

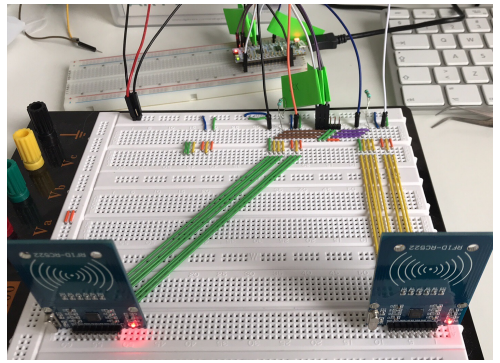
- SPI: SS/CS

Antal ben/linjer med N tilsluttede enheder:

- 3 + N

Færre pins ud af IC end parallelforb.... Mindre pakke, mindre energi.



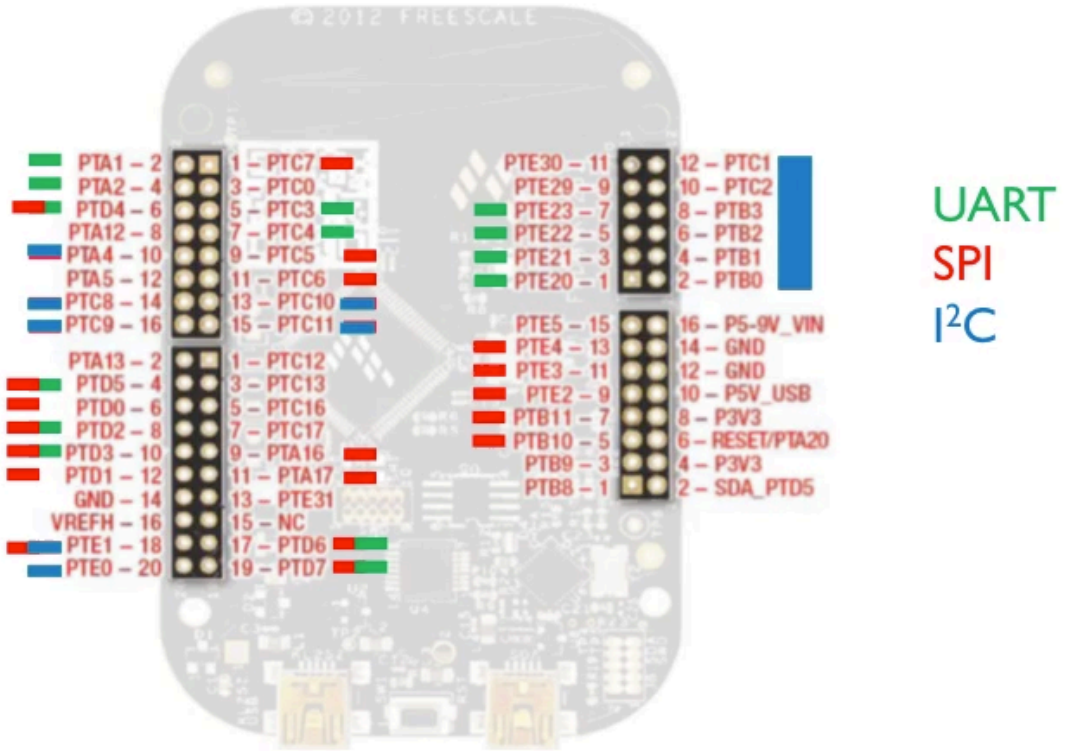


HW5: SERIEL KOMMUNIKATION
KL25Z

KL25Z har mulighed for UART, SPI og I2C

- 3x UART (UART0 er dog optaget)
- 2x SPI
- 2x I2C
- USB

Hvis vi skal bruge CAN, så er det på PocketBeagle ;)



HW5: SERIEL KOMMUNIKATION
KL25Z UART INITIALISERING

UART2 clock fra SCGC4

UART2 Control register 2
Slukker for TX (TE) og RX (RE)

Baud rate generator (prescaler)
Clockes ift. bus clock på 24 MHz
BDH og BDL er er High Low bits i modulo divisor

UART2 data register
-> Nu skriver vi endelig ud på linjen...

```
void Init_UART2(uint32_t baud_rate) {
        uint32_t divisor;
        // enable clock to UART and Port A
        SIM->SCGC4 |= SIM_SCGC4_UART2_MASK;
        SIM->SCGC5 |= SIM_SCGC5_PORTE_MASK;

        // connect UART to pins for PTE22, PTE23
        PORTE->PCR[22] = PORT_PCR_MUX(4);
        PORTE->PCR[23] = PORT_PCR_MUX(4);
        // ensure tx and rx are disabled before configuration
        UART2->C2 &=  ~(UARTLP_C2_TE_MASK | UARTLP_C2_RE_MASK);

        // Set baud rate to 4800 baud
        divisor = BUS_CLOCK/(baud_rate*16);
        UART2->BDH = UART_BDH_SBR(divisor>>8);
        UART2->BDL = UART_BDL_SBR(divisor);

        // No parity, 8 bits, two stop bits, other settings;
        UART2->C1 = UART2->S2 = UART2->C3 = 0;

        // Enable transmitter and receiver
        UART2->C2 = UART_C2_TE_MASK | UART_C2_RE_MASK;
}
```

```
void UART2_Transmit_Poll(uint8_t data) {
        // wait until transmit data register is empty
        while (!(UART2->S1 & UART_S1_TDRE_MASK))
                ;
        UART2->D = data;
}
```

Vil outputte på PTE22 og PTE23, sæ tænder for clock til portmodul E

PTE22 alt. 4: UART2_TX
PTE23 alt. 4: UART2_RX

8 høje bits og 8 lave bits i baud rate generator

UART2 Control register 2
Tænder for TX (TE) og RX (RE) igen

Pos. for flaget, der går op, når data register er klar til at blive skrevet til

HW5: SERIEL KOMMUNIKATION
OPSUMMERING OG PERSPEKTIVERING

Sende/modtage på interrupts

- Buffer -> cirkulær FIFO til hver retning

Message parsing og protokoller/specs (fx NMEA til sonarer, TSIP til GPS-modtagere, osv.)

- endianness (network byte order big-endian)
- binær til struktureret data (ascii, utf-8, tal, osv...)

Fejlhåndtering, støjimmunitet

- Differentielt signal, højere spændinger

Andre kommunikationstyper: CAN-bus 4. semester ☺

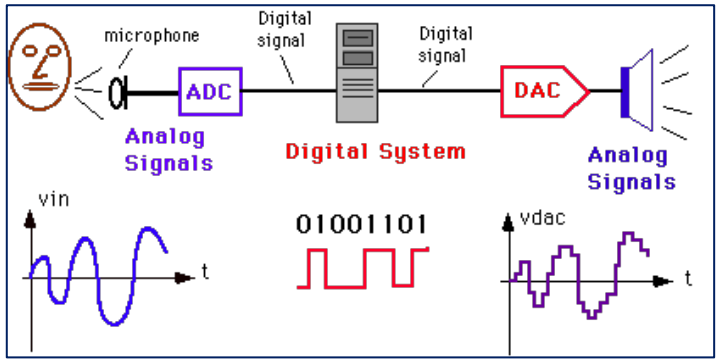
HW6: ANALOG INTERFACING
INDHOLD

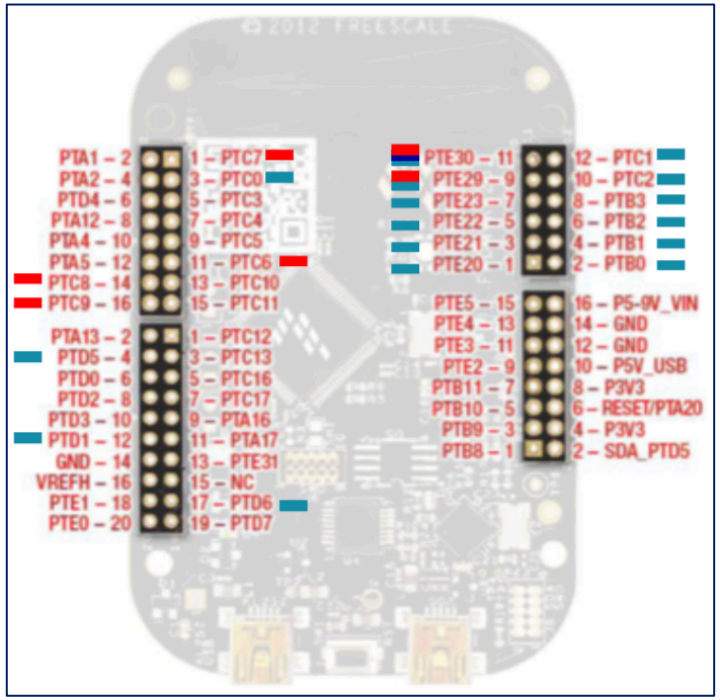
1. Formål med analogt interfacing
2. ADC'en i KL25Z
3. KL25Z eksempel på analogt input via ADC
    1. Overvejelser omkring konverteringstid
4. Hardware triggering
5. Opsætning af pins (pinmux)
6. Opsummering og perspektivering

6: Analog interfacing

1) Giv et kort overblik over Analog til Digital Converteren (ADC) i KL25, suppler evt. med et kodeeksempel, der viser hvordan converteren anvendes.
2) Forklar hvordan ***Hardware triggering*** kan anvendes i forhold til ADC'en.
3) Hvordan sættes KL25 processorens Ben (Pins) op når de skal anvendes til f.eks. Analogt input eller som GPIO.

HW6: ANALOG INTERFACING
ANALOG CONVERSION KL25 (ADC)





Formål med analog interfacing:

- Oversætte mellem digital og analog elektronik.
  - Måle på / interagere med den fysiske verden via analog elektronik
  - Fysiske fænomener som en kontinuert/analog spændingssignal (kontinuert i tid og skala):
    - Temperatur, tryk, lysstyrke/intensistet, lydtryk, acceleration/vibration, udsvingning, osv.
- Ind: Oversætte kontinuert spænding til et diskret/digitalt signal (sampling + kvantisering); ADC, CMP.
- Ud: Generere analoge signaler; DAC.

Hardware til rådighed:

- Ind: 1x **ADC** (14 kanaler: 14xSE, 4xDI), 1x Comparator (6 inputs)
- Ud: 1x DAC (1 output)

HW6: ANALOG INTERFACING
ANALOG CONVERSION KL25 (ADC)

Konversion A/D

- Vælg +/- referencer
- Vælg opløsning = præcision
- Vælg clockfrekvens
- Single-ended vs. differentialsig.
- Digital kode repræsenterer analogt interval
- Interpolation til at udregne det analoge interval for fysisk param.

Andre overvejelser:

- Type er SAR-konverter så
  - Binærsøgning tager N steps.
  - vs. kvantiseringsfejl sfa. N
- Samplingsfrekvens ift. maks. frek.
  - Anti-aliaseringsfiltre

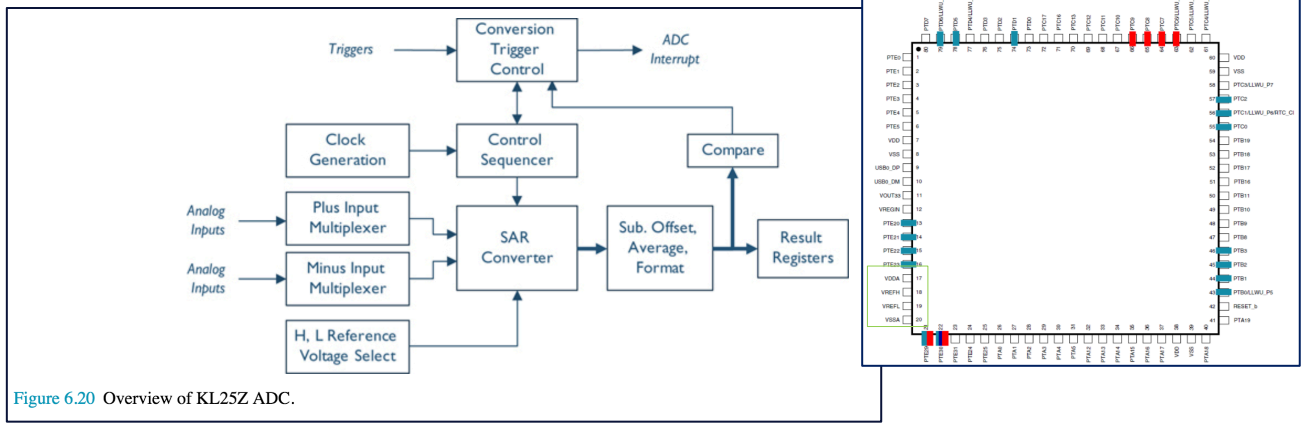


Figure 6.20 Overview of KL25Z ADC.

General Equation

n = converted code

V_{in} = sampled input voltage

V_{+ref} = upper voltage reference

V_{-ref} = lower voltage reference

N = number of bits of resolution in ADC

$$n = \left\lfloor \frac{(V_{in} - V_{-ref})\,2^N}{V_{+ref} - V_{-ref}} + 1/2 \right\rfloor$$

$$n = \left\lfloor \frac{(V_{in})\,2^N}{V_{+ref}} + 1/2 \right\rfloor$$

$$n = \left\lfloor \frac{3.30\,v\;2^{10}}{5\,v} + 1/2 \right\rfloor = 676$$

$\lfloor X \rfloor = I$ *floor function: nearest integer I such that I <= X*

floor(x+0.5) rounds x to the nearest integer

General Equation

n = converted code

V_{in_min} = minimum input voltage for code n

V_{in_max} = maximum input voltage for code n

V_{+ref} = upper voltage reference

V_{-ref} = lower voltage reference

N = number of bits of resolution in ADC

$$V_{in\_min} = \frac{n - \frac{1}{2}}{2^N}(V_{+ref} - V_{-ref}) + V_{-ref}$$

$$V_{in\_max} = \frac{n + \frac{1}{2}}{2^N}(V_{+ref} - V_{-ref}) + V_{-ref}$$

Simplification with V_{-ref} = 0 V

$$V_{in\_min} = \frac{n - \frac{1}{2}}{2^N}(V_{+ref})$$

$$V_{in\_max} = \frac{n + \frac{1}{2}}{2^N}(V_{+ref})$$

HW6: ANALOG INTERFACING
ANALOG CONVERSION KL25 (ADC)

Steps...

Initialisering

- Clock til porte og ADC
- Vælg generelle indstillinger
- Konfigurér conv.clock (ADCK)
- Vælg Vref
- Vælg trigger (HW/SW)
- Konfigurer inputkanal/pin
- *Burde kalibrere her...*

Konvertering

- SW trigger pba. mask
- Poll COCO-bit (bedre: IRQ)
- Udlæs resultat, cast til def. bitlænge

Resultat:

- Koder mellem ~350 (støj) og 65536 (2^16-1)

Initialisering

```c
/*
 * @author Janus Bo Andersen
 * @date 2019-2020
 * @param channel adc_channel (corresp. to book p171 and manpg 464, pinmap)
 * @return void
 * @brief this function initializes the ADC.
 */

void adcInit(adc_channel channel)
{

    /* Clock source for port B & C */
    SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK;
    SIM->SCGC5 |= SIM_SCGC5_PORTC_MASK;

    /* Connect clock source for ADC */
    SIM->SCGC6 |= SIM_SCGC6_ADC0_MASK;

    /* Setup the converter part 1 */
    ADC0->CFG1 = ADC_CFG1_ADLPC_MASK |          // Low power config
                 ADC_CFG1_ADLSMP_MASK |        // Long sample time
                 ADC_CFG1_MODE(3) |            // 16-bit, single-ended
                 ADC_CFG1_ADICLK(0);           // ADCK is Bus Clock

    /* Setup the converter part 2 (redundant but explicit method) */
    ADC0->SC2 = ADC_SC2_REFSEL(0);             // Voltage ref. is VREFH, VREFL
    ADC0->SC2 &= ~ADC_SC2_ADTRG_MASK;          // Software trigger
    ADC0->SC2 &= ~ADC_SC2_ACFE_MASK;           // Compare func. disabled
    ADC0->SC2 &= ~ADC_SC2_DMAEN_MASK;          // DMA disabled

    /* Set up input for ADC input */
    switch (channel) {
    case AD15: //PTC1
        PORTC->PCR[1] = 0x00000000;            // Totally clear PCR (precaution)
        PORTC->PCR[1] &= ~PORT_PCR_MUX_MASK;   // Clear MUX for PTC1
        PORTC->PCR[1] |= PORT_PCR_MUX(0);      // Set up for analog input
        break;
    }
}
```

Konvertér og udlæs

```c
 * @author Janus Bo Andersen
 * @date 2019-2020
 * @param channel adc_channel (corresp. to book p171 and manpg 464, pinmap)
 * @brief this function from a channel on ADC
 * @return unsigned short with the 16-bit code
 */
unsigned short adcRead(adc_channel channel)
{
    /* Warning: This implementation requires that adcInit(channel) has been run */

    unsigned short res;         // Result code from conversion
    uint32_t start_mask = 0UL;  // Mask used to trigger conversion

    /* Set up for ADC input */
    switch (channel) {
    case AD15: //PTC1
        start_mask = 15UL;                      // Channel 15 (0b01111)
        start_mask &= ~ADC_SC1_DIFF_MASK;       // Single-ended mode (disable DIFF)
        start_mask &= ~ADC_SC1_AIEN_MASK;       // No interrupts (disable AIEN)
        break;
    }

    /* Software-triggering consists of writing to SC1A */
    ADC0->SC1[0] = start_mask;

    /* Poll - busy waiting until COCO (Conv. Completed) */
    /* TODO: Implement IRQ approach */
    while (!( ADC0->SC1[0] & ADC_SC1_COCO_MASK)) {
        // zzzz
    }

    /* result: the 32-bit register, casted to unsigned short,
     * as unused bits 16:31 are cleared by HW */
    res = (unsigned short) ADC0->R[0];

    return res;
}
```

HW6: ANALOG INTERFACING
OVERVEJELSER OM KONVERTERINGSTID

Som udgangspunkt:

- 16-bit for bedst præcision
- single-ended eller diff. afhængigt af behov
- Ekstra features efter behov

Hvis hastighed er en faktor:

- Undgå low-power conversion
- Justér præcision
- Vælg single-ended
- Undgå at bruge adders
- Undgå at bruge averaging

Precision (bits)	Mode	Base Conversion Time BCT
8	single-ended	17 ADCK cycles
9	differential	27
10	single-ended	20
11	differential	30
12	single-ended	20
13	differential	30
16	single-ended	25
16	differential	34

Eksempel: Differential, 16-bit:
f_{ADC} = 12 MHz => T_{ADC} = 83 ns/cycle
T_{BCT} = 34 cycles x 83 ns / cycle = 2.8 µs

2.8 µs ~ 136 CPU cycles @ 48 MHz.

HW6: ANALOG INTERFACING
HARDWARE TRIGGERING

Hvordan kan hardware triggering anvendes?

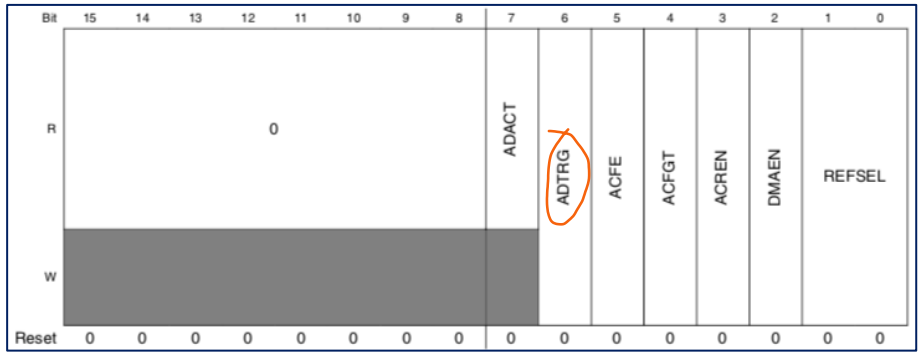
Trigger: udløser en ADC-måling. HW eller SW.

Muligheder for HW-signaler som triggers:

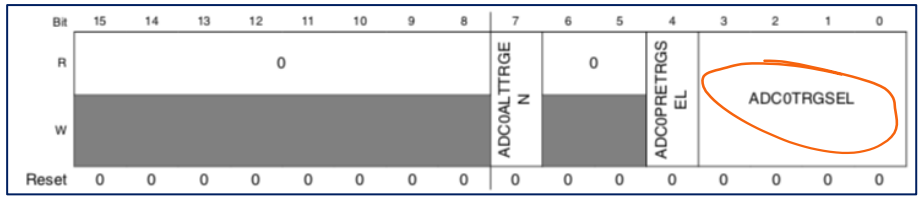
- Ekstern udløser
- Tid: TPM, PIT, RTC, LPTMR
- Comparator

Metode:

- Vælg HW-trigger i feltet ADTRG i ADC'ens Status and Control Register 2 (ADC0_SC2)
- Vælg trigger-kilde i feltet ADC0TRGSEL fra Systems Options 7 (SIM_SOPT7)



6 ADTRG	Conversion Trigger Select Selects the type of trigger used for initiating a conversion. Two types of trigger are selectable: • Software trigger: When software trigger is selected, a conversion is initiated following a write to SC1A. • Hardware trigger: When hardware trigger is selected, a conversion is initiated following the assertion of the ADHWT input after a pulse of the ADHWTSn input. 0 Software trigger selected. 1 Hardware trigger selected.



3–0 ADC0TRGSEL	ADC0 trigger select Selects the ADC0 trigger source when alternative triggers are functional in stop and VLPS modes. . 0000 External trigger pin input (EXTRG_IN) 0001 CMP0 output 0010 Reserved 0011 Reserved 0100 PIT trigger 0 0101 PIT trigger 1 0110 Reserved 0111 Reserved 1000 TPM0 overflow 1001 TPM1 overflow 1010 TPM2 overflow 1011 Reserved 1100 RTC alarm 1101 RTC seconds 1110 LPTMR0 trigger 1111 Reserved

HW4: ANALOG INTERF.
OPSÆTNING AF PIN

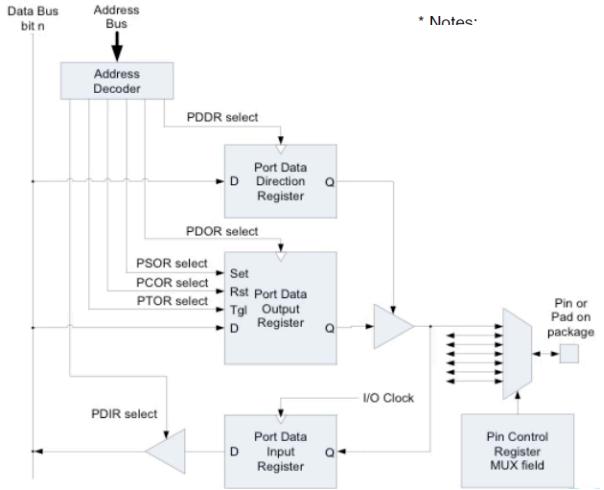
Enkelt pins (ikke bus)

- Mux sætter fysisk forbindelse
- Pin control register MUX bits styrer funktion
- Forskellige pins har forskellige "muligheder" -> pinouts
- Analog/ADC:
  - Alternativ 0 (0b0)

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0							ISF	0				IRQC			
W								w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0					MUX			0	DSE	0	PFE	0	SRE	PE	PS
W																
Reset	0	0	0	0	0	x*	x*	x*	0	x*	0	x*	0	x*	x*	x*

* Notes:



MUX (bits 10-8)	Configuration
000	Pin disabled (analog)
001	Alternative 1 – GPIO
010	Alternative 2
011	Alternative 3
100	Alternative 4
101	Alternative 5
110	Alternative 6
111	Alternative 7

FRDM-KL25Z Pins				KL25Z128 Pins									
On-board Usage	I/O Header & Pin Num	Arduino™ R3 Pin Name	FRDM-KL25Z Pin Name	KL25Z Pin #	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	Reset State/Function
--	J2 20	D14	PTE0	1		PTE0		UART1_TX	RTC_CLKOUT	CMP0_OUT	I2C1_SDA		DISABLED
--	J2 18	D15	PTE1	2		PTE1	SPI1_MOSI	UART1_RX		SPI1_MISO	I2C1_SCL		DISABLED
--	J5 09	--	PTE2	3		PTE2	SPI1_SCK						DISABLED
--	J5 11	--	PTE3	4		PTE3	SPI1_MISO			SPI1_MOSI			DISABLED
--	J5 13	--	PTE4	5		PTE4	SPI1_PCS0						DISABLED
--	J5 15	--	PTE5	6		PTE5							DISABLED
Power	--	--	VDD	7	VDD								VDD
Power	--	--	VSS	8	VSS								VSS
USB	--	--	USB0_DP	9	USB0_DP								USB0_DP
USB	--	--	USB0_DM	10	USB0_DM								USB0_DM
2.2uF cap	--	--	VOUT33	11	VOUT33								VOUT33
USB VBUS (5V)	--	--	VREGIN	12	VREGIN								VREGIN
--	J10 01	--	PTE20	13	ADC0_DP0/ADC0_SE0	PTE20		FTM1_CH0	UART0_TX				ADC0_DP0/ADC0_SE0
--	J10 03	--	PTE21	14	ADC0_DM0/ADC0_SE4a	PTE21		FTM1_CH1	UART0_RX				ADC0_DM0/ADC0_SE4a
--	J10 05	--	PTE22	15	ADC0_DP3/ADC0_SE3	PTE22		FTM2_CH0	UART2_TX				ADC0_DP3/ADC0_SE3
--	J10 07	--	PTE23	16	ADC0_DM3/ADC0_SE7a	PTE23		FTM2_CH1	UART2_RX				ADC0_DM3/ADC0_SE7a
Power	--	--	VDDA	17	VDDA								VDDA
Power	J2 16	AREF*	VREFH	18	VREFH								VREFH
Power	--	--	VREFL	19	VREFL								VREFL
Power	--	--	VSSA	20	VSSA								VSSA
--	J10 09	--	PTE29	21	CMP0_IN5/ADC0_SE4b	PTE29		FTM0_CH2	FTM_CLKIN0				CMP0_IN5/ADC0_SE4b
--	J10 11	--	PTE30	22	DAC0_OUT/ADC0_SE23/CMP0_IN4	PTE30		FTM0_CH3	FTM_CLKIN1				DAC0_OUT/ADC0_SE23/CMP0_IN4

HW6: ANALOG INTERFACING
OPSUMMERING OG PERSPEKTIVERING

SW1: C++ KLASSER OG OOP
INDHOLD

1. Klasser (eks. fra uge 38)
2. Nedarvning
3. Access specifiers
4. Constructors/destructors
    1. Ansvar for at rydde op?

1	C++	Forklar • hvad klasser er, • hvordan constructor og destructor fungerer og • access specifiers (private, protected og public) • hvorledes klasser kan arve, single inheritance,

SW1: C++ KLASSER OG OOP
HVAD ER KLASSER

Objektorienteret koncept: Software-objekter, der modsvarer "virkelighedens" objekter...

- Indkapsling, "metoder", nedarvning

C++ typesystem:

- Brugerdefinerede komplekse typer: Klasser og structs
- Adskille interface og implementation
- Beskytte variable/datastrukturer

Vores ArgParser "Options" fra uge 38

Options.h (definition af type og interface)

```
5   #ifndef U38_2_OPTIONS_H
6   #define U38_2_OPTIONS_H
7
8   #include <iostream>
9   #include <string>
10  #include <vector>
11
12  class Options {
13  public:
14      Options() {} //default constructor
15      Options(int argc, char** argv) : argc(argc), argv(argv) {} //member initialization list synta
16      ~Options() {}
17      std::string getopt(); //return valid options given, one by one
18      int numopt(); //return number of given options in argv
19      int valopt(); //return number of valid options in argv
20      bool operator== (const Options& other) const; //non-mutating/inspection operator
21      void setOptPossible(std::vector<std::string> optPossible);
22
23  private:
24      void parseValidOpts(); //parse the given options for only valid ones
25
26      //Pure virtual, must be implemented in subclasses
27      virtual void parseGivenOpts() = 0; //parse the argv list for options given
28
29  protected:  //make accessible in derived class
30      int argc = 0;
31      char** argv = nullptr;
32      int _numopt = 0; //num of options in argv
33      int _valopt = 0; //num of given options that were valid
34
35      //These types are now defined in subclasses as they vary
36      std::vector<std::string> optPossible;  // defined by user, options program can respond to
37      std::vector<std::string> optValid; //constructed from intersection of Given options and optstring
38      std::vector<std::string> optGiven; //constructed from argv
39  };
```

typenavn

public interface

constructor-metoder

destructor

public metoder (get)

overload operator

private interface

pure virtual metode

protected interface

Indkapslede variable (member variables)

SW1: C++ KLASSER OG OOP
IMPLEMENTERING

Implementering

- Implementerer ind i klassen som et namespace
- Interface skal være stabilt
- Implementering kan ændres, optimeres

Hvis man har overloadet sine constructors, så bør man implementere muligheder for at "nå til samme tilstand"

- fx Options() og Options(argc, argv)...

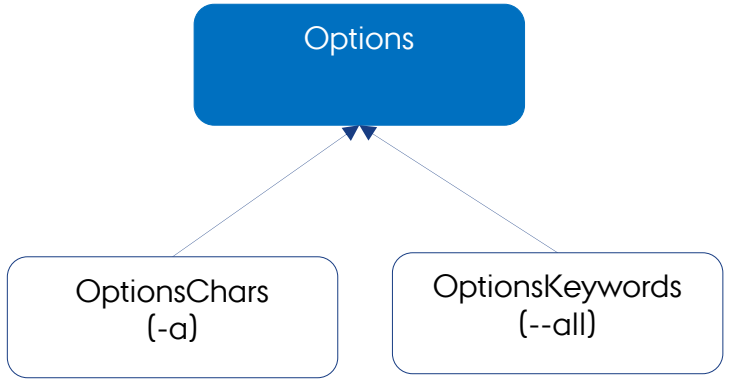
Options.cpp (udklip fra implementering)

```
5   #include "Options.h"
6   #include <regex>
7   #include <algorithm>
8   #include <stdexcept>
9
10  //sets a vector that defines valid options, then calls the parser for valid options
11  void Options::setOptPossible(std::vector<std::string> optPossible) {
12      this->optPossible = optPossible; //set private member
13      parseValidOpts(); //now parse and build set of valid options
14      return; //done
15  }
16
17  //Parses given options and finds the valid ones
18  void Options::parseValidOpts() {
19
20      //iterate over the options (allowed opts), and the ones found in Given are put in Valid vector
21      std::vector<std::string>::iterator it; //iterator for find algo
22
23      for (auto option: optPossible) {
24          //check if the option was given at the command line
25          it = std::find(begin(optGiven), end(optGiven), option); //where the c was found
26          if (it != end(optGiven)) //if find did not reach end of vector, c was found
27              optValid.push_back(option); // add the option as it was found in the Given
28      }
29
30      _valopt = optValid.size(); //set the number of Given options that were Valid
31
32      return;
33  }
```

Definerer en tidligere erklæret public metode

Arbejder på private variable

SW1: C++ KLASSER OG OOP
NEDARVNING



Nedarvning

- Single inheritance: nedarver metoder, variable fra en anden klasse
  - *Hvordan* nedarvningen sker afhænger af **nedarvningstypen** i den nye klasse (public, private, protected)
  - *Hvad der er adgang til* af variable og metoder fra base class afhænger af **access specifiers** i base class (senere slides)
- Multiple inheritance: arve fra flere klasser på en gang...

Eksemplet:

- To specifikke options-typer arver ”publically” fra Options

Ny derived class

Arver fra Options

```
14  class OptionsChars : public Options {
15  public:
16      OptionsChars() {}; // default derived class constructor
17      OptionsChars(int argc, char** argv) : Options(argc, argv) { parseGivenOpts(); } //call base class ctor
18
19  private: //Pure virtual, must be implemented in subclasses
20      void parseGivenOpts() override; //parse the argv list for options given
21  };
```

Laver nye constructors til denne klasse

Denne klasse implementerer en pure virtual method

```
14  class OptionsKeywords : public Options {
15  public:
16      OptionsKeywords() {}; // default derived class constructor
17      OptionsKeywords(int argc, char** argv) : Options(argc, argv) { parseGivenOpts(); } //call base class ctor
18
19  private: //Pure virtual, must be implemented in subclasses
20      void parseGivenOpts() override; //parse the argv list for options given
21  };
```

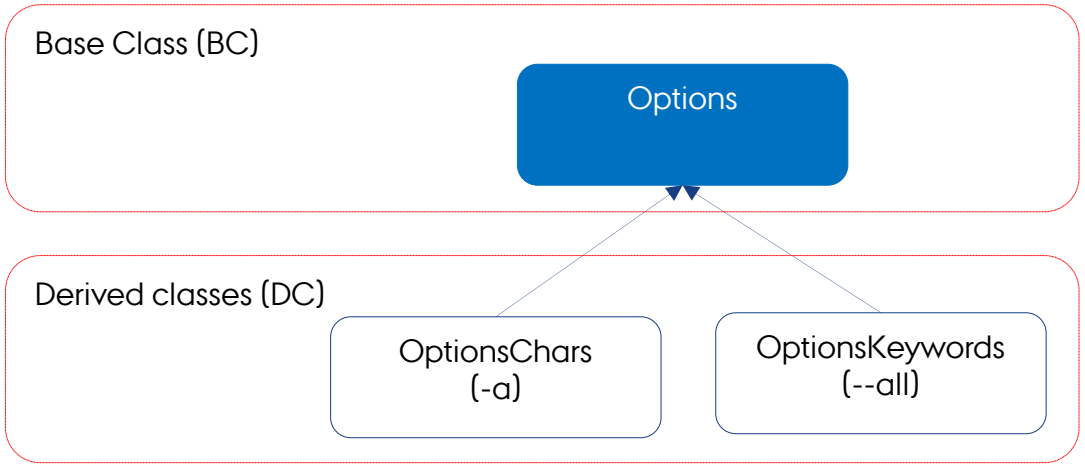
SW1: C++ KLASSER OG OOP
ACCESS SPECIFIERS

Access Specifiers:

- Adgang til data/metoder i klassen
- `public`: kan tilgås af "alle"
- `protected`: kan tilgås i DC
- `private`: kan kun tilgås fra klassen selv
- Friendship ☺: En klasse defineret som `friend class <navn>` må gerne tilgå private/protected. Den behøver ikke nedarve.

Type af nedarvning:

- Hvordan den nye klasse bygges:
- `public`: nedarver fra base class sp
  - public fra BC som public i DC
  - protected fra BC som protected i DC
  - private kan jo ikke tilgås...
- `protected`:
  - public og protected fra BC bliver protected i DC.
- `private`:
  - public og protected fra BC bliver private i DC.



SW1: C++ KLASSER OG OOP
CONSTRUCTORS OG DESTRUCTORS

Constructors:

- Definerer, hvad der skal ske, når en ny instans (objekt) af en klasse skal oprettes
- Copy constructor giver tit mening!

Destructors:

- Definerer, hvordan objektet skal afvikles: Oprydning, delete

Hvornår? Lifetime, ansvar for at rydde op...

- `new` / `delete`, versus smartpointers (`sharedptr`, `uniqueptr`)...

Kritiske systemer:

- C++ guideline/krav til flysystemer: Ingen delete/free efter takeoff...

Embedded systemer

- Måske undgå heap/freestore, pga. hukommelsesfragmentering

Argumenter til instantiering af nyt objekt

Initializer list: Benytter BC constructor til at initialisere

Kalder denne funktion så objektet færdig-initialiseres

```
17    OptionsKeywords(int argc, char** argv) : Options(argc, argv) { parseGivenOpts(); } //call base class ctor
```

```
16    ~Options() {}
```

C++'s spec/regler for hvad der sker, når et objekt destrueres, går ud af scope

```
43    std::vector<std::string> optChars {"a", "b", "o"}; //only accept -a -b -o options
44    std::vector<std::string> optKeywords {"alpha", "beta", "omega"}; //only accept --alpha --beta --omega options
45
46    // Instantiate two different Options-derived objects (for demo)
47    OptionsChars myOptionsChars(argc, argv);
48    OptionsKeywords myOptionsKeywords(argc, argv);
49
50    // We can now access them via polymorphism
51    std::vector<Options*> opt(2); //vector holds two pointers to options objects
52    opt[0] = &myOptionsChars;
53    opt[1] = &myOptionsKeywords;
54
55    // We are now working through an Options object pointer (base class)
56    opt[0]->setOptPossible(optChars);
57    opt[1]->setOptPossible(optKeywords);
```

Instantierer nye objekter

Arbejder polymorfisk på objekterne

SW1: C++ KLASSER OG OOP
OPSUMMERING OG PERSPEKTIVERING

SW2: C++ OVERLOADING, OOP, POLYMORF.
INDHOLD

1. Overloading af funktioner og metoder
2. Instantiering af objekter
3. Hvad betyder polymorfisme
4. Templates
5. Opsummering og perspektivering

2	C++	Forklar • function overloading, • object instantiation • hvad polymorphism er og hvorledes det kan udnyttes • hvad templates er og hvordan det kan anvendes i programmeringen

SW2: C++ OVERLOADING, OOP, POLYMORF.
OVERLOADING AF FUNKTIONER

Ideen:

- Forskellige metode-/funktionsdefinitioner for forskellige "argumentlister" med med samme funktionsnavn
  - Virker for *bruger* af API som samme funktion, men forskellige typer/antal af argumenter resulterer i kald til forskellige funktioner
- Endnu bedre, hvis man også har overloadet sine operatorer

Hvis man har overloadet sine constructors, så bør man implementere muligheder for at "nå til samme tilstand"

- fx Options() og Options(argc, argv)...

Eksempler:

- Overloaded constructor
- Overloaded operator
- Vores argparser "Options" fra uge 37/38

```
12  class Options {
13  public:
14      Options() {} //default constructor
15      Options(int argc, char** argv) : argc(argc), argv(argv) {} //member initialization list syntax
```

To forskellige metoder til at instantiere et objekt

```
3   // Find the maximum of two integers
4   int max(int a, int b) {
5       return (a > b) ? a : b;
6   }
7
8   // Find the maximum of two doubles
9   double max(double a, double b) {
10      return (a > b) ? a : b;
11  }
12
13  int main() {
14      //Determine some maxima and give to the user
15      std::cout << max(2,1) << " and " << max(2.1,2.2) << std::endl;
16      return 0;
17  }
```

Rigtig fedt at kunne beregne max uden at skulle tænke alt for meget over hvilken type... Implicit/explicit casting er også muligheder

```
94  //non-mutating/inspection operator
95  //compares all data items for this and other: this==other
96  bool Options::operator== (const Options& other) const {
97
98      //comparison list is very redundat, only first three needed... Others are then invariants.
99      if (    this->argc == other.argc            // same command line argument length
100         && this->argv == other.argv              // same command line argument mem location
101         && this->optString == other.optString    // started with identical strings
102         && this->optGiven == other.optGiven      // command line options have been parsed (always true if above)
103         && this->optValid == other.optValid      // cmd line options have been filtered with optstring (same)
104         && this->_numopt == other._numopt        // this should always be true as a consequence of above
105         && this->_valopt == other._valopt) {     // same...
106         return true; //similar
107     } else {
108         return false; //not similar
109     }
110 }
```

Tilbage i options-eksemplet... Overloader operator

SW2: C++ OVERLOADING, OOP, POLYMORF.
INSTANTIERING

Instantiering:

- Kald til constructor
- Valg af allokering
  - lokal/automatisk variabel
  - heap/freestore-variabel

Scope, lifetime, memory leaks, ansvar for oprydning

- `new` / `delete`, versus smartpointers (`sharedptr`, `uniqueptr`)…

Kritiske systemer:

- C/C++ guideline/krav til flysystemer: Ingen delete/free efter takeoff…

Embedded systemer

- Måske undgå heap/freestore, pga. hukommelsesfragmentering

Argumenter til instantiering af nyt objekt

Initializer list: Benytter BC constructor til at initialisere

Kalder denne funktion så objektet færdig-initialiseres

```
17    OptionsKeywords(int argc, char** argv) : Options(argc, argv) { parseGivenOpts(); } //call base class ctor
```

```
16    ~Options() {}
```

C++'s spec/regler for hvad der sker, når et objekt destrueres, går ud af scope

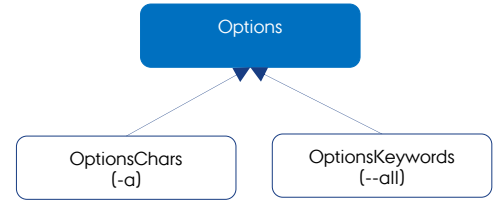
```
8   #include <iostream>
9   #include <string>
10  #include "Options.h"
11
12  int main(int argc, char* argv[]) {
13
14      std::string optString {"abo"}; //only accept -a -b -o options
15
16      //allocate object on the heap, modern C++ style
17      auto myOptions = std::make_unique<Options>(argc, argv); //make a unique smart pointer (RAII)
18
19      myOptions->setOptstring(optString); //set the allowed options in the object, also parses them
20
21      std::cout << "Number of GIVEN options in command line: " << myOptions->numopt() << std::endl;
22      std::cout << "Number of VALID options in command line: " << myOptions->valopt() << std::endl;
23
24      //popping the options one by one...
25      for (unsigned i = 0; i < myOptions->valopt(); ++i) {
26          std::cout << "Valid option was set: " << myOptions->getopt() << std::endl;
27      }
28
29      return 0;
30  }
```

pointer-typen bliver også automatisk infereret

Smartptr tilgang til at instantiere med allokering på heap. Det er nu C++'s problem, at der bliver ryddet op… RAII-binding

RAII garanterer de-allokering her!

SW2: C++ OVERLOADING, OOP, POLYMORF.
POLYMORFISME



Polymorfisme

- Options-klassen er polymorf til de to derived klasser (de nedarver fra Options)
  - Den kan altså *antage* form som enten det ene eller det andet.
  - Vi har implementeret pure virtual-funktioner – som skal implementeres i en derived klasse.
  - Options er i dette tilfælde en *abstract base class*. Dens interface skal implementeres fuldt ud af en derived klasse.

- Når vi tilgår Options-klassen, så benyttes nedarvnings-hierarkiet til at afgøre, hvilken "form", der er tale om...

```
43      std::vector<std::string> optChars {"a", "b", "o"}; //only accept -a -b -o options
44      std::vector<std::string> optKeywords {"alpha", "beta", "omega"}; //only accept --alpha --beta --omega options
45
46      // Instantiate two different Options-derived objects (for demo)
47      OptionsChars myOptionsChars(argc, argv);
48      OptionsKeywords myOptionsKeywords(argc, argv);
49
50      // We can now access them via polymorphism
51      std::vector<Options*> opt(2); //vector holds two pointers to options objects
52      opt[0] = &myOptionsChars;
53      opt[1] = &myOptionsKeywords;
54
55      // We are now working through an Options object pointer (base class)
56      opt[0]->setOptPossible(optChars);
57      opt[1]->setOptPossible(optKeywords);
```

Instantierer nye objekter som lokale variabler

Pointer til base class!

Arbejder polymorfisk på objekterne

```
25
26      //Pure virtual, must be implemented in subclasses
27      virtual void parseGivenOpts() = 0; //parse the argv list for options given
```

pure virtual metode i Options-klassen
Skal implementeres!

```
1   /* E3ISD1 Week 38/2 -- Exercise 8
2    * Parse and shows valid options from command line with pattern like "-h" and "--help" (GNU style)
3    *
4    * This week we work with Polymorphism and Exception handling.
5    *
6    * Polymorphism
7    * - A class that declares or inherits a virtual function is called a polymorphic class.
8    *   -> So the Options class is polymorphic to its two derived classes.
9    *   -> It declares pure virtual functions that are only implemented in derived classes.
10   *   -> The Options base class is an _abstract base class_.
11   * - To demonstrate polymorphism, we access the derived classes through pointers to a base class type.
```

Nogen har været søde at huske at skrive en forklaring i koden... ☺

SW2: C++ OVERLOADING, OOP, POLYMORF.
TEMPLATES

Templates

- Generisk definition, mange anvendelser
- Både funktioner og klasser
- C++ compileren udleder hvilke funktionskopier, der skal generes

Anvendelser

- Over det hele i moderne C++, STL er dybt afhængigt af Templates
- "Generic programming", ikke-typeafhængige algoritmer og datastrukturer
  - Fx et generisk binærtræ, der kan indeholde alt muligt...
- Don't repeat yourself...

Ulemper:

- Aggressiv anvendelse kan give stor kode (mange kopier af templatekode med type fyldt ud...)

Containeren std::vector er defineret generisk, vi bruger template-argumentet til at angive hvilken type/klasse der skal indeholdes

```
43      std::vector<std::string> optChars {"a", "b", "o"}; //only accept -a -b -o options
```

Definerer nu max-funktionen som en template
- Kan fungere for alle typer med operatoren '>' implementeret
- Operator overloading vil altså give mulighed for at anvende denne på i princippet alle typer.

```
3   // Find the maximum of operands with > operator
4   template <typename T>
5   T max(T a, T b) {
6       return (a > b) ? a : b;
7   }
8
9   int main() {
10      //Determine some maxima and give to the user
11      std::cout << max(2,1) << ", " << max("Kaffe", "Kage") << std::endl;
12      return 0;
13  }
```

C++ compileren udleder selv hvilken funktion, der skal kaldes

SW2: C++ OVERLOADING, OOP, POLYMORF.
OPSUMMERING OG PERSPEKTIVERING

SW3: MAKE
INDHOLD

1. Formål og metode med Make
2. Eksempel på automatisering
3. Eksempel på mere avanceret Makefile
4. CMake til at lave cross-platform Makefiles
5. Opsummering og perspektivering

3	make	Forklar hvordan make og en makefile fungerer

SW3: MAKE
FORMÅL OG METODE

Ideen:

- Automatisk build
  - Compiling, linking
- Definér targets, , include libraries, compiler, compiler-flag osv. én gang
  - Definer evt. midlertidige compile targets
- Kør simpel kommando for at gentage noget kompliceret (jeg har vist det overdrevet kompliceret!)
- Nemt at dele/distribuere

Tilgang:

- make er GNU utility (Eclipse bruger også make)
- Makefile er tekst-definition på build (husk! skal være tabs!)

Anvendelser

- Behøver ikke at være compiling, kan også være opgaveafvikling...

Makefile

```
1  #Makefile for exercise E3ISD1 wk 38-2
1  CC=g++
2  CFLAGS=-std=c++14 -Wall
3
4  exercise: main.o Options.o OptionsChars.o OptionsKeywords.o
5    $(CC) $(CFLAGS) main.o Options.o OptionsChars.o OptionsKeywords.o -o exercise
6
7  main.o:
8    $(CC) $(CFLAGS) -c main.cpp
9
10 Options.o:
11   $(CC) $(CFLAGS) -c Options.cpp
12
13 OptionsChars.o:
14   $(CC) $(CFLAGS) -c OptionsChars.cpp
15
16 OptionsKeywords.o:
17   $(CC) $(CFLAGS) -c OptionsKeywords.cpp
18
19
20 .PHONY clean:
21   rm -rf *.o
```

Compiler og flag som variable

Target

Afhængig af disse objektfiler er bygget

Linker dem sammen

Bygger objektfiler

Ryd op

```
~/git/E3ISD1-SW/u38_2(master*) » make                    janusboandersen@Januss-MacBook-Pro
g++  -std=c++14 -Wall -c main.cpp
g++  -std=c++14 -Wall -c Options.cpp
g++  -std=c++14 -Wall -c OptionsChars.cpp
g++  -std=c++14 -Wall -c OptionsKeywords.cpp
g++  -std=c++14 -Wall main.o Options.o OptionsChars.o OptionsKeywords.o -o exercise
```

SW3: MAKE
AUTOMATISERING

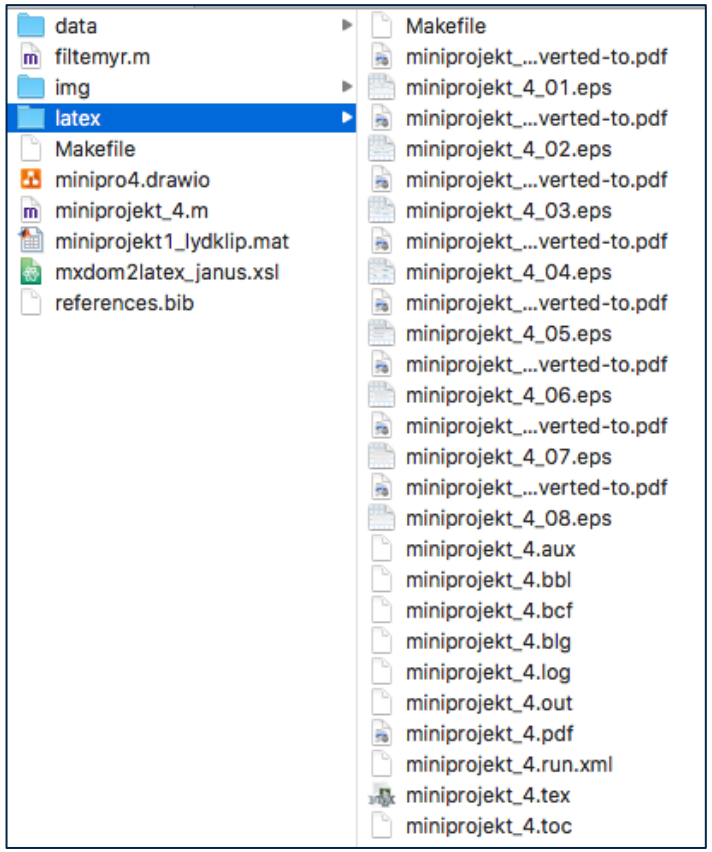
Eksemplet

- Automatisering af LaTeX afvikling
- Skal køre flere gange pga. indholdsfortegnelse, bibliografi, citater, osv.

- Makefile, der referer til Makefile

Makefile
```
1  all:
1    $(MAKE) -C latex all
```

Makefile
```
1  all:
1    lualatex miniprojekt_4.tex
2    biber miniprojekt_4.bcf
3    lualatex miniprojekt_4.tex
4    lualatex miniprojekt_4.tex
5    open miniprojekt_4.pdf
6
7  clean:
```



SW3: MAKE
LIDT MERE AVANCERET

Eksemplet

- Kan fuldautomatisere byggeproces
- Benytte kommandoer til at afvikle shell-funktioner
- Bruges med en mere ”best-practice” biblioteksstruktur

```
Makefile
41 # ----- Target, build directory and source directory -----
40 # set target executable file. It will be compiled into the build dir
39 TARGET_EXEC ?= a.out
38
37 # Set directories for placing built files and intermediate object files
36 BUILD_DIR ?= ./build
35
34 # Set directory to search for source code
33 SRC_DIRS ?= ./src
32
31
30 # ----- Compilers and versions -----
29 # These need likely not be explicitly defined, but are done so here for stability
28 # Using GNU compilers as they seem to produce similar results across platforms
27 # On Mac OS, gcc and g++ are wrapped by Clang
26
25 # C compiler and flags
24 CC := gcc
23
22 # C++ compiler and flags
21 CXX := g++
20 # CXX_FLAGS := -Wall -Werror -std=c++11
19
18
17 # ----- Build dependency / prequisites tree -----
16
15 # Source code filenames to map in dependency tree and to compile
14 SRCS := $(shell find $(SRC_DIRS) -name *.cpp -or -name *.c -or -name *.s)
13
12 # Object file names to map in dependency tree and to give to linker
11 # Prefixes the build_dir, and postfixes .o (main.cpp -> build/main.cpp.o)
10 OBJS := $(SRCS:%=$(BUILD_DIR)/%.o)
 9
 8 # Dependency list files, one dependency file per source file (main.cpp.o -> main.cpp.d)
 7 DEPS := $(OBJS:.o=.d)
 6
 5
 4 # ----- Pre-processor -----
 3 # Specify include directories to search for header files during pre-processing
 2 INC_DIRS := $(shell find $(SRC_DIRS) -type d)
 1
67 # Build the flag for all custom include directories ( -I<dir1> -I<dir2> -I<dir3> ... )
NORMAL Makefile                                                    make
```

Prøver at have en fornuftig biblioteksstruktur: /src, /build osv

Variabel defineres så højresiden evalueres

% er wildcards
Finder all objektfiler i build_dir

Omdøber ud fra skabelone

SW3: MAKE
CMAKE

CMake:

- Nyere cross-platform Makefile-maker
- Definerer projektet som et objekt
  - Sætte attributter
- Fordel:
  - Forskellige platforme kan have dependencies forskellige steder, forskellige navne, osv
  - Makroer (defineret af folkene bag Cmake og andre contributors), definerer hvordan en hver platform fungerer
- CMakeLists.txt definerer
- -> cmake laver en Makefile
- -> make afvikler build

Options.cpp | Options.h | main.cpp | CMakeLists.txt

```
1  cmake_minimum_required(VERSION 3.14)
2  project(u38_2)
3
4  set(CMAKE_CXX_STANDARD 17)
5  add_library(Options Options.cpp)
6  add_library(OptionsChars OptionsChars.cpp)
7  add_library(OptionsKeywords OptionsKeywords.cpp)
8
9  add_executable(u38_2 main.cpp)
10 target_link_libraries(u38_2 Options OptionsChars OptionsKeywords)
```

SW3: MAKE
OPSUMMERING OG PERSPEKTIVERING

Opsummering

Andre buildsystemer

- Rake -> Ruby-baseret automatisering
- Autotools -> GNU build-system

SW4: BEAGLEBONE STARTUP, I/O, M.V.
INDHOLD

1. Boot-up & Shutdown
2. Shell scripts
    1. Anvendelse og eksempel med GPIO
    2. Øvrigt eksempel
3. I/O
    1. Overblik over I/O muligheder på PB
    2. I/O: GPIO og PWM
    3. I/O: I2C og ADX345
    4. Overblik over seriel kommunikation
4. Opsummering og perspektivering

4	BeagleBone Black/PocketBeagle (forkortet BBB)	Forklar • hvad der sker når en BBB booter - og når den lukkes rigtigt ned • hvad shell-scripts er og hvor de anvendes gerne med eksempler • hvordan skabes der adgang til IO (gpio, i2c, can, pwm, etc.)

SW4: BEAGLEBONE STARTUP, I/O, M.V.

BOOT-UP

Hovedbudskaber

Der kører en opstartsrutine inden man kan få adgang til PB/BB via seriel/SSH

- "Dum" processer/hardware -> initialiseret hardware -> kernel loadet i hukommelse -> userspace vi kan interagere med...
- Ingen UEFI/BIOS på BB ->
- Resetvektor: AM335x's startup ROM (clock+hukom.) -> Bootloader
  - -> U-Boot bootstrapper via filer på PB'ens SD-kort
    - Initialisering og test af hardware (device tree*)
    - Loader tidligt filsystem til hukommelse (ramdisk)
    - Loader kernen til hukommelsen
      - -> systemd tager så over...

Hvorfor er det interessant for os?

- Vi kan modificere startup-proces/hardware/miljø via uEnv.txt
- Vi kan modificere boot config for device tree (pins standardfkt.)
- Vi kan lave device tree overlays, hvis vi permanent tilføjer ny HW
- Hvis noget går galt, kan vi måske reparere det (læsbar debugsymb.)
- Vigtigste filer er i partition uden for userspace rækkevidde (fx u-boot.img)

*) Device tree: IEEE-standard til definition af hardware. Samme kerne på flere systemer – FDT er specifik til harware; undgår hardcoding i source.

```
janus@beaglebone:/boot$ ls -la
totalt 17840
drwxr-xr-x  4 root   root      4096 aug  4 02:12 .
drwxr-xr-x 21 root   root      4096 aug  4 02:10 ..
-rw-r--r--  1 root   root    161266 jul 31 00:16 config-4.14.108-ti-r113
drwxr-xr-x  3 root   root      4096 aug  4 02:03 dtbs
-rw-r--r--  1 root   root   4696442 aug  4 02:12 initrd.img-4.14.108-ti-r113
-rw-r--r--  1 root   root       542 aug  4 03:27 SOC.sh
-rw-r--r--  1 root   root   3447799 jul 31 00:16 System.map-4.14.108-ti-r113
drwxr-xr-x  2 root   root      4096 aug  4 02:10 uboot
-rw-r--r--  1 debian debian    2299 aug  4 03:27 uEnv.txt
-rwxr-xr-x  1 root   root   9929216 jul 31 00:16 vmlinuz-4.14.108-ti-r113
```

```
janus@beaglebone:/boot/dtbs/4.14.108-ti-r113$ ls -la | grep pocket
-rw-r--r-- 1 root root 124423 jul 31 00:16 am335x-pocketbeagle.dtb
-rw-r--r-- 1 root root 119573 jul 31 00:16 am335x-pocketbeagle-gamepup.dtb
-rw-r--r-- 1 root root 126463 jul 31 00:16 am335x-pocketbeagle-techlab.dtb
```

4 stadier i opstarten

- 1: Primary Program Loader (AM335x ROM): Gør det muligt at læse fra SD-kortet
- 2: U-Boot Secondary Program Loader: Initialiserer pin-muxing, clocks, hukommelse. Loader U-Boot.
- 3: U-Boot (regular): Filsystem, uEnv.txt, loader og giver kontrol til Linux-kernen
- 4: Linux-kernen i hukommelse, perifere enheder konfigureres (USB, osv.). Mounter endeligt filsystem.

SW4: BEAGLEBONE STARTUP, I/O, M.V.
SHUT-DOWN

Hovedbudskaber

Lad være med bare at tage strømmen...

Brug:

- Power-knappen til "soft shutdown"
- `sudo reboot` eller `sudo shutdown –h now`

Hvorfor?

- Fordi Derek Molloy siger det...

- Kørende processer skal have en chance for afslutte, gemme
- Filsystem skal unmountes
- Power management skal opdateres

Til højre: Soft shutdown

```
beaglebone login: [▯[0;32m  OK  ▯[0m] Closed Load/Save RF Kill Switch Status /dev/rfkill Watch.
         Stopping Session 1 of user ISD.
[▯[0;32m  OK  ▯[0m] Stopped target System Time Synchronized.
[▯[0;32m  OK  ▯[0m] Stopped target Timers.
[▯[0;32m  OK  ▯[0m] Stopped Daily Cleanup of Temporary Directories.
[▯[0;32m  OK  ▯[0m] Stopped target Host and Network Name Lookups.
         Stopping dnsmasq - A lightweight DHCP and caching DNS server...
         Stopping User Manager for UID 1001...
         Unmounting /run/user/1001...
[▯[0;32m  OK  ▯[0m] Unmounted /run/user/1001.
[▯[0;32m  OK  ▯[0m] Reached target Unmount All Filesystems.
[▯[0;32m  OK  ▯[0m] Stopped target Local File Systems (Pre).
[▯[0;32m  OK  ▯[0m] Stopped Remount Root and Kernel File Systems.
[▯[0;32m  OK  ▯[0m] Stopped Create Static Device Nodes in /dev.
[▯[0;32m  OK  ▯[0m] Reached target Shutdown.
[▯[0;32m  OK  ▯[0m] Reached target Final Step.
         Starting Power-Off...
[   82.361044] reboot: Power down

[▯[0;32m  OK  ▯[0m] Reached target Shutdown.
[ 8812.284326] watchdog: watchdog0: watchdog did not stop!
[ 8812.537959] reboot: Restarting system
```

SW4: BEAGLEBONE STARTUP, I/O, M.V.
SHELL-SCRIPTS

Shell scripts

Script som afvikles af/i shell

- bash, Zsh, etc.
  - En unix/posix-ting (virker kun måske i Powershell/Win)
  - Der kan være forskellig syntax i forskellige shells
- Automatisering af linux-kommandoer
- Ikke bedste valg til mere avancerede scripts (brug Python eller Lua i stedet)
  - *Kun* linux-kommandoer, ingen libraries
  - Ikke OOP, ikke test-/udviklingsværktøjer
- Spawner en ny shell (proces) ved afvikling
- Kræver execute-rettigheder (chmod ugo+x), og evt. sudo
- Kan placeres, hvor det bliver auto-executed ved opstart
  - /etc/init.d/

Eksempel 1

bashLED.sh

```
1  #!/bin/bash
1  LED3_PATH=/sys/class/leds/beaglebone:green:usr3
2
3  function removeTrigger
4  {
5    echo "none" >> "$LED3_PATH/trigger"
6  }
7
8  echo "Shell-script til håndtering af diode 3"
9
10 if [ $# != 1 ]; then
11   echo "Du må kun give et argument: "
12   echo -e "bashLED <command> \n"
13   echo -e "hvor <command>=on, off, flash, status"
14   exit 2
15 fi
16 echo "Du har givet kommandoen: $1"
17
18 if [ "$1" == "on" ]; then
19   echo "Tænder diode 3"
20   removeTrigger
21   echo "1" >> "$LED3_PATH/brightness"
22 elif [ "$1" == "off" ]; then
23   echo "Slukker for diode 3"
24   removeTrigger
25   echo "0" >> "$LED3_PATH/brightness"
26 elif [ "$1" == "flash" ]; then
27   echo "Blinker med diode 3"
28   echo "timer" >> "$LED3_PATH/trigger"
29   echo "50" >> "$LED3_PATH/delay_on"
30   echo "50" >> "$LED3_PATH/delay_off"
31 elif [ "$1" >> "status" ]; then
32   cat "$LED3_PATH/trigger";
33 fi
34 echo "Scriptet er slut"
```

Afvikles af ...

Variabel

Funktion

Control flow if/then / elif/ else/ fi

Input-args

Exit-val (fejl 2, incorrect usage)

[] er en test substitution... Bash's [[ ]] er bedre!

Afslutning af if-struktur

SW4: BEAGLEBONE STARTUP, I/O, M.V.
SHELL-SCRIPTS

Eksempel fra Pro3 - testafvikling

s8-tc.sh

```
#!/usr/bin/env bash

# This script runs test cases S8.TC-1, TC-2, TC-3 and TC-4
# Only tested on MacOS so far.
# I don't think TC-4 is compatible on Windows, and I am not sure
and grep...
# Janus 21 Nov 2019

MQTT_PORT="1883"
WEBSOCK_PORT="8081"
URL="auteam2.mooo.com"
TMP_FILE="./test-cases/s8-tc-tmp.txt"
TEST_MSG="This is a test message sent from publisher to subscriber

# Attempting to determine the system, and set the correct commands
# Prepare
if [[ "$(uname)" =~ "MING"? ]];then
  USER_OS="Windows"
  DOCKERCOMPOSEUP="docker-compose -f docker-compose-win.yml up -d
  LOCALHOST="192.168.99.100"
  PORTSCANNER="nc -z"
  WORDSEARCH="grep -w"

elif [[ "$(uname)" =~ "Darwin"? ]];then
  USER_OS="macOS"
  DOCKERCOMPOSEUP="docker-compose up -d --build"
  LOCALHOST="localhost"
  PORTSCANNER="nc -z"
  WORDSEARCH="grep -w"

elif [[ "$(uname)" =~ "Linux"? ]];then
  USER_OS="Linux"
  DOCKERCOMPOSEUP="docker-compose up -d --build"
  LOCALHOST="localhost"
  PORTSCANNER="nc -z"
  WORDSEARCH="grep -w"

else
  USER_OS="unknown"
  DOCKERCOMPOSEUP="docker-compose up -d --build"
  LOCALHOST="localhost"
```

s8-tc.sh

```
service_running () {
  if [[ -z "$(docker-compose ps | ${WORDSEARCH} ".*mqtt.*Up.*")" ]]; then
    false # the service mqtt does not have Up in the status
  else
    true
  fi
}

listening_on () {
  if [[ -z "$( ${PORTSCANNER} "$1" "$2" 2>&1 | ${WORDSEARCH} "succeeded" )" ]]; the
    false #the word succeeded was not found, so the service is not listening on the
  else
    true
  fi
}

get_credentials () {
  if [[ -z "${MOSQUITTO_USER}" ]] || [[ -z "${MOSQUITTO_PASSWORD}" ]]; then
    echo "Enter credentials for the Mosquitto server:"
    read -p "Enter User: " MOSQUITTO_USER
    read -s -p "Enter Password: " MOSQUITTO_PASSWORD
    echo ""
    export MOSQUITTO_USER
    export MOSQUITTO_PASSWORD
  fi
}



#Font colours
RED='\033[1;31m'          # Bold red
GREEN='\033[0;32m'        # Bold green
YELLOW='\033[0;33m'       # Bold yellow
HEAD='\033[1;37;44m'
NC='\033[0m' # No Color

#### Info to users ###
echo -e "${HEAD}Running all testcases for deliverable S8${NC}"

#### TEST-CASE TC-1 ####
TC="S8.TC-1"
```

s8-tc.sh

```
#### TEST-CASE TC-1 ####
TC="S8.TC-1"
echo -e "${HEAD}*** Test-case ${TC} ***${NC}"
echo "Verifies that the Mosquitto server is successfully running in docker-compose"

if service_running; then
  echo -e "${TC}: ${GREEN}OK${NC}. Service is running."
else
  echo -e "${YELLOW}Service not started${NC}. Attempting to build and start."
  $DOCKERCOMPOSEUP

  if service_running; then
    echo -e "${TC}: ${GREEN}OK${NC}. Service is running."
  else
    echo -e "${TC}: ${RED}FAILED${NC}. Service could not be started."
    exit 1
  fi
fi

#### TEST-CASE TC-2.01 ####
TC="S8.TC-2.01"
echo -e "${HEAD}*** Test-case ${TC} ***${NC}"
echo "Verifies that the Mosquitto-server is listening on

if listening_on "${LOCALHOST}" "${MQTT_PORT}"  && listen
  echo -e "${TC}: ${GREEN}OK${NC}. Service is listening.
else
  echo -e "${TC}: ${RED}FAILED${NC}. Service is not list
fi

#### TEST-CASE TC-2.02 ####
TC="S8.TC-2.02"
echo -e "${HEAD}*** Test-case ${TC} ***${NC}"
echo "Verifies that the Mosquitto-server is listening on

if listening_on "${URL}" "${MQTT_PORT}"  && listening_on
  echo -e "${TC}: ${GREEN}OK${NC}. Service is listening.
else
  echo -e "${TC}: ${RED}FAILED${NC}. Service is not list
fi

NORMAL  s8-tc.sh
```

s8-tc.sh

```
#### TEST-CASE TC-3 ####
TC="S8.TC-3"
echo -e "${HEAD}*** Test-case ${TC} ***${NC}"
echo "Verifies that a message can be sent to the Mosquitto server on $LOCALHOST port $MQTT_PORT."

# Ask and set credentials if not inherited from parent process
get_credentials
if mosquitto_pub -h ${LOCALHOST} -p ${MQTT_PORT} -t "Test/topic" -m "Test message" --id TestPublisher -u "${MOSQUI
TTO_USER}" -P "${MOSQUITTO_PASSWORD}"; then
  echo -e "${TC}: ${GREEN}OK${NC}. Service can accept publish messages."
else
  echo -e "${TC}: ${RED}FAILED${NC}. Service declined the operation."
fi


#### TEST-CASE TC-4 ####
TC="S8.TC-4"
echo -e "${HEAD}*** Test-case ${TC} ***${NC}"
echo "Verifies that a topic can be subscribed to and that a message can received from the Mosquitto server on $LOC
ALHOST port $MQTT_PORT."
get_credentials

# Start the subscriber as a background process and direct the output to a file
mosquitto_sub -h ${LOCALHOST} -p ${MQTT_PORT} -t "Test/topic" --id TestSubscriber -u "${MOSQUITTO_USER}" -P "${MOS
QUITTO_PASSWORD}" > "${TMP_FILE}" & PROC_ID=$!
disown
sleep 1 # Wait to ensure it's running

# Send a publish message
mosquitto_pub -h ${LOCALHOST} -p ${MQTT_PORT} -t "Test/topic" -m "${TEST_MSG}" --id TestPublisher -u "${MOSQUITTO_
USER}" -P "${MOSQUITTO_PASSWORD}"

# Wait for message handling, kill the subscribe process
sleep 1
kill -1 ${PROC_ID} 2>/dev/null

# Read back the output from the received message and compare to what was sent
RCV_MSG=$(tr -d '\0' < $TMP_FILE )
if [ "${RCV_MSG}" = "${TEST_MSG}" ]; then
  echo "Received the message: ${RCV_MSG}"
  echo -e "${TC}: ${GREEN}OK${NC}. Messages on subscribed topics can be received from the service."
NORMAL  s8-tc.sh                                    sh  utf-8[unix]  95% ≡  164/171 ln :  1
```

SW4: BEAGLEBONE STARTUP, I/O, M.V.

I/O: OVERBLIK

I/O-systemet

Pins har 8 modes (mux bits 0-2)

- Brug ikke de "røde" pins i headertabellerne

Single-wire (GPIO, mode 7)

- Digital I/O
- Elekt. begrænsninger! 3.3V, src:4-6mA, snk:8mA

Seriel bus (SPI, I2C, CAN) eller UART

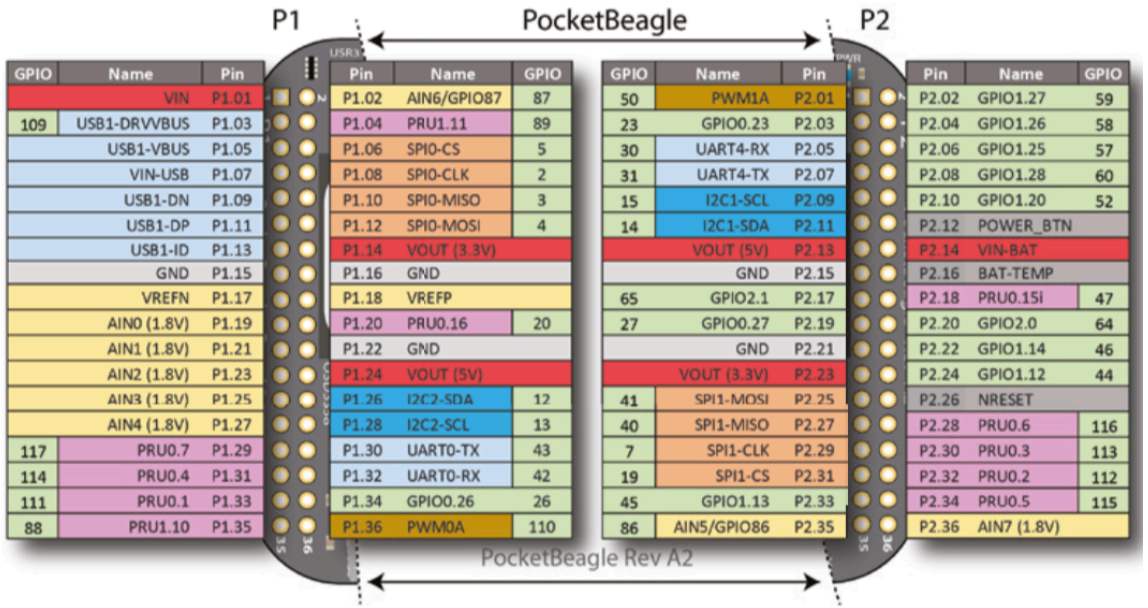
- Kommunikation til komplekse sensorer eller moduler
- Seriel kommunikation

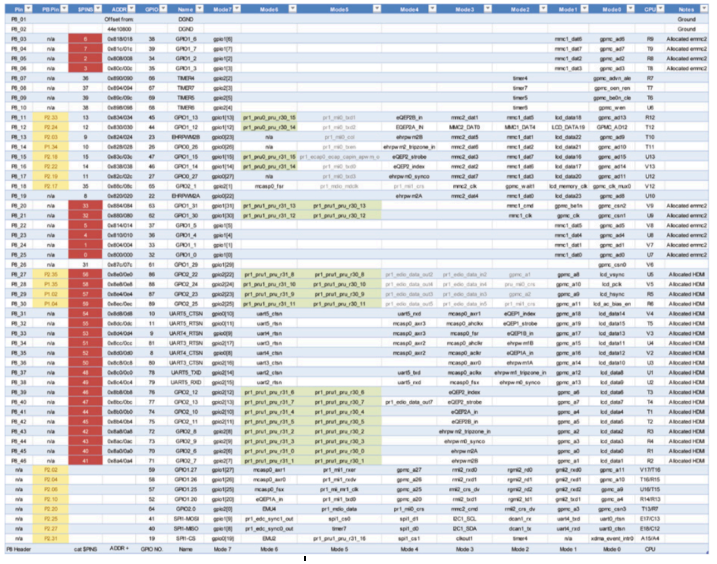
Analog interfacing

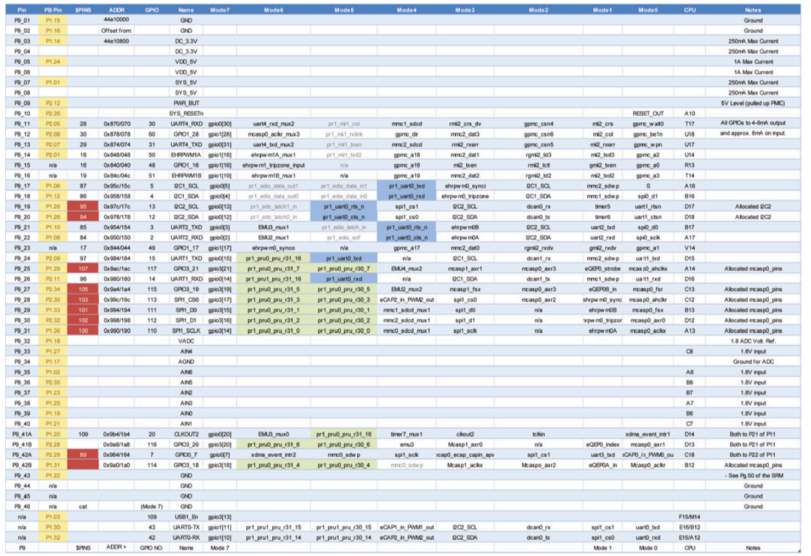
- Ind: ADC, ud: PWM

USB-devices

- Hvis der er Linux-drivers







SW4: BEAGLEBONE STARTUP, I/O, M.V.

I/O: *ALTING ER EN FIL*

```
janus@beaglebone:~$ groups
janus dialout cdrom floppy sudo audio video plugdev users i2c spi pwm gpio
```

I/O linux tools -> alt er en fil

config-pin tool:

- mode, direction (+/- pullup/down) value

```
janus@beaglebone:~/projects/E3ISD1/ch6$ sudo usermod -a -G gpio janus
```

```
janus@beaglebone:~/projects/E3ISD1/ch6$ config-pin -l p2.08
default gpio gpio_pu gpio_pd gpio_input
```

```
janus@beaglebone:~/projects/E3ISD1/ch6$ config-pin -q p2.08
P2_08 Mode: gpio Direction: in Value: 0
```

Mapping ift. AM335x fire GPIO-banker á 32 pins:

- PB 2.08 -> GPIO1_28 => 32+28 = GPIO60

-> Sysfs:

```
janus@beaglebone:~/projects/E3ISD1/ch6$ cat /sys/class/gpio/gpio60/direction
in
```

-> Alt er en fil

```
janus@beaglebone:/sys/class/gpio/gpio60$ config-pin -a p2.08 out
janus@beaglebone:/sys/class/gpio/gpio60$ echo 1 > value
janus@beaglebone:/sys/class/gpio/gpio60$ config-pin -q p2.08
P2_08 Mode: gpio Direction: out Value: 1
```

Men: Langsom metode, høj CPU laod (polling sysfs).

- Hurtig switching -> PWM!

C++ tools (gpio.h)

PWM-eksempel

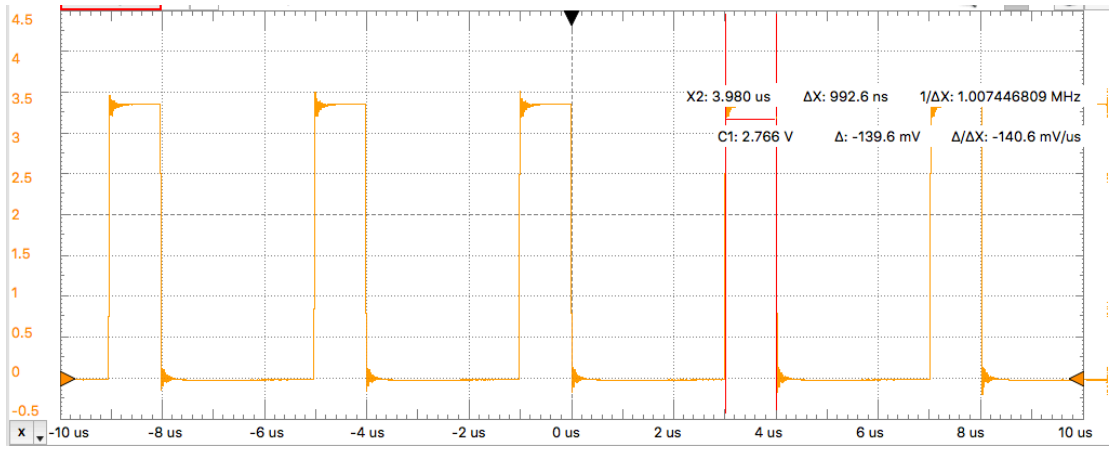
pwm.sh

```
1  #!/bin/bash
1  PWM_PATH="/sys/class/pwm/pwm-0:0"
2  if [ "$1" == "on" ]; then
3    echo 4000 > ${PWM_PATH}/period
4    echo 1000 > ${PWM_PATH}/duty_cycle
5    echo 1 > ${PWM_PATH}/enable
6    echo "PWM Period is now $(cat ${PWM_PATH}/period) ns."
7    echo "PWM Duty-cycle is now $(cat ${PWM_PATH}/duty_cycle) ns out of total period."
8  elif [ "$1" == "off" ]; then
9    echo 0 > ${PWM_PATH}/enable;
10 fi
```

BBB CHIP[3]	CHANNEL	BBB PINS	PB CHIP	BP PINS
pwmchip0	0A	P9_22/ P9_31	pwmchip0	P1.08/ P1.36
pwmchip0	0B	P9_21/ P9_29	pwmchip0	P1.10/ P1.33

```
janus@beaglebone:~/projects/E3ISD1/pwm$ ./pwm.sh on
PWM Period is now 4000 ns.
PWM Duty-cycle is now 1000 ns out of total period.
```

```
janus@beaglebone:/sys/class/pwm/pwm-0:0$ ls -la
totalt 0
drwxrwxr-x 3 root pwm    0 jan  1  2000 .
drwxrwxr-x 5 root pwm    0 jan  1  2000 ..
-r--r--r-- 1 root pwm 4096 jan  1  2000 capture
lrwxrwxrwx 1 root pwm    0 jan 23 00:07 device -> ../../pwmchip0
-rw-rw-r-- 1 root pwm 4096 jan  1  2000 duty_cycle
-rw-rw-r-- 1 root pwm 4096 jan  1  2000 enable
-rw-rw-r-- 1 root pwm 4096 jan  1  2000 period
-rw-rw-r-- 1 root pwm 4096 jan  1  2000 polarity
drwxrwxr-x 2 root pwm    0 jan  1  2000 power
lrwxrwxrwx 1 root pwm    0 jan 23 00:07 subsystem -> ../../../../../../../../class/pwm
-rw-rw-r-- 1 root pwm 4096 jan  1  2000 uevent
```



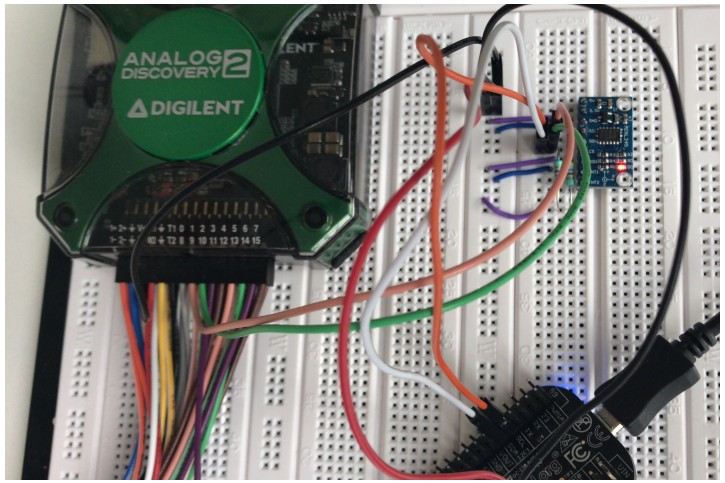
SW4: BEAGLEBONE STARTUP, I/O, M.V.

I/O: LINUX TOOLS, I2C – ADXL345

ADXL345

Accelerometer, tri-axial MEMS sensor

- I2C, SPI; her forbundet via I2C (øv. 20a)
- SDA og SCL pulled-up (aktiv lav)
- Interrupts 1 og 2 ikke benyttet
- På I2C-bus på adresse 0x53, DEVID 0xe5
- Start-bit (SDA), så clock, så 7-bit adresse



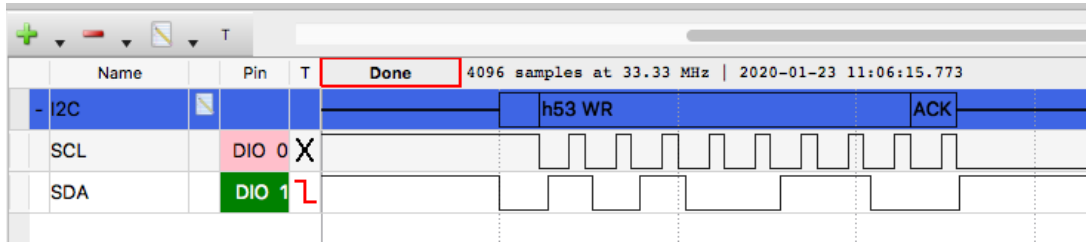
Linux tools

```
janus@beaglebone:~$ i2cdetect -y -r 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- 53 -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

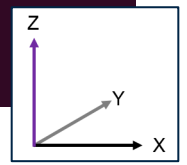
```
janus@beaglebone:~$ i2cget -y 1 0x53 0x00
0xe5
```

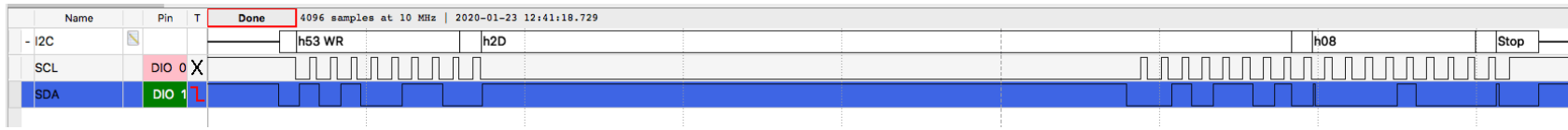
```
janus@beaglebone:~/projects/E3ISD1/ADXL345$ i2cdump -y 1 0x53 b
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: e5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4a    ?..............J
10: 82 00 30 00 00 ff 01 3e 00 00 00 93 00 00 00 00    ?.0...?>...?....
20: 00 00 00 00 00 00 00 00 00 00 00 00 0a 08 00 00    ............??..
30: 82 00 fc ff 06 00 f9 00 00 00 00 00 00 00 00 00    ?.?.?.?.........
40: e5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4a    ?..............J
50: 80 00 30 00 00 fe 02 3e 00 00 00 93 00 00 00 00    ?.0..??>...?....
60: 00 00 00 00 00 00 00 00 00 00 00 00 0a 08 00 00    ............??..
70: 82 00 fb ff 07 00 f8 00 00 00 00 00 00 00 00 00    ?.?.?.?.........
80: e5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4a    ?..............J
90: 00 00 30 00 00 fe 01 3e 00 00 00 93 00 00 00 00    ..0..??>...?....
a0: 00 00 00 00 00 00 00 00 00 00 00 00 0a 08 00 00    ............??..
b0: 02 00 fc ff 07 00 f9 00 00 00 00 00 00 00 00 00    ?.?.?.?.........
c0: e5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4a    ?..............J
d0: 00 00 30 00 00 ff 02 3e 00 00 00 93 00 00 00 00    ..0...?>...?....
e0: 00 00 00 00 00 00 00 00 00 00 00 00 0a 08 00 00    ............??..
f0: 02 00 fc ff 09 00 f8 00 00 00 00 00 00 00 00 00    ?.?.?.?.........
```

Outputs




```
janus@beaglebone:~/projects/E3ISD1/ADXL345$ ./ADXL345
Starting the ADXL345 sensor application
The Device ID is: e5
The POWER_CTL mode is: 08
The DATA_FORMAT is: 00

X=-5 Y=7 Z=249 sample=24
```





SW4: BEAGLEBONE STARTUP, I/O, M.V.

I/O: SERIEL KOMMUNIKATION

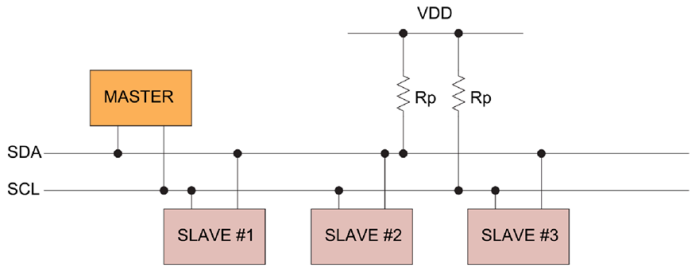
I2C

Seriel-bus, hardware og protokol

- 2 linjer: SDA, SCL (aktiv lav)
- Synkron (clocked, SCL)
- Philips / NXP specifikation
- Hurtigere end UART, langsommere end SPI

Master-slaves

- 7-bit-adresser (128 enheder, men...)
- Clock-synkronisering fra master
  - Clock-stretching fra slave
- Multi-master via "collision detection" og "arbitration".



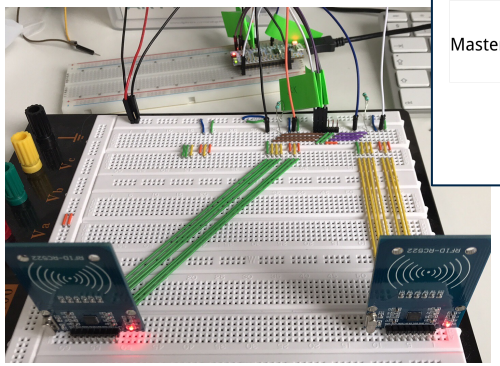
SPI

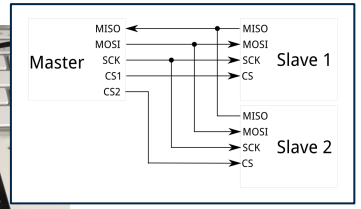
Seriel-bus, hardware og protokol

- 2+N linjer: MOSI, MISO, SCK, *N x* SS/CS
- Synkron (clocked, SCK)
- Master-slave(s), kun 1 master
- Hurtigste af I2C, UART og SPI

Eksempel:

- Øvelse 20b
- Pro2, flere RFID-læsere i "samme hus"

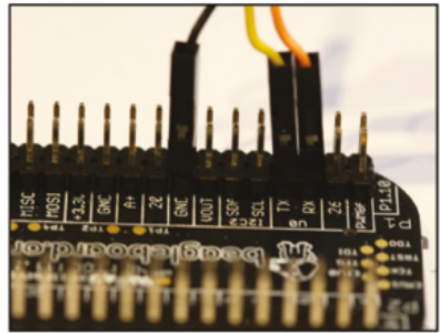




UART

Seriel-forbindelse, kun hardware

- Point-to-point (UART1 <-> UART2)
  - Rx, Tx linjer på hvert punkt
  - Asynkron: Ingen clock
    - Hastighed (<115200 baud) skal være ens I hver ende
    - Start/stop-bits, og maks. længde af databits (5-9)
  - Hardware: Parallel databus -> skifteregister -> Tx -> serielforb. -> Rx, osv...
- Kan implementere forskellige kommunikationsprotokoller, xx RS-232
- Langsommest af de tre
- Men simpel!



SW4: BEAGLEBONE STARTUP, I/O, M.V.
OPSUMMERING OG PERSPEKTIVERING

SW5: BEAGLEBONE UDVIKLING
INDHOLD

1. Overvejelser inden valg af sprog/teknologi
2. Scripting-eksempler
3. Cross-compiling
4. Remote debugging
5. Øvrige muligheder
6. Opsummering og perspektivering

5	Programudvikling og BBB	Forklar • hvordan man kan udvikle programmer der skal afvikles på BBB • hvordan man kan fejlfinde et program, som kører på en BBB

SW5: BEAGLEBONE UDVIKLING
UDVIKLING – OVERORDNEDE VALG

Scripting og fortolkede sprog

Stor udvalg

- Shell, Python, Lua, js/Node.js, Java

Metode:

- Hvis sproget ikke allerede er understøttet, så installér vha. debians package manager
- Installer krævede libs vha. sprogets egen package manager
- Åben en editor i terminalen (ViM, Emacs, nano, …) og gå i gang…

Fordele:

- Kræver som regel blot at der build-tools samt at sprogets package manager kan hente pakker

Ulemper:

- Performance er sjældent lige så god som compilede sprog
- Debugging-værktøjer ikke så gode
- Koden kan læses og ændres af alle

Compilede sprog

Udvalg:

- C/C++, Rust, osv.

Generel fordel

- Mere ”performant” kode
- Mulighed for build-system til at håndtere komplekse builds (Make, Cmake)

Metode 1: Compiling direkte på BB

- **+** Skriv kode og benyt toolchain/compiler i BB (GNU), samt evt. GDB og Valgrind
- **–** Ikke samme IDE-hjælp som med cross-compiling

Metode 2: Cross-compiling og cross-debugging

- Cross-compiler tool-chain (Linaro)
- **+** Grafisk IDE + Debugger (Eclipse CDT + GDB/GDBserver)
- **+** Debugger på den faktiske hardware + peripherals
- **–** Kan være vanskeligt med dynamisk linking til mange biblioteker
  - Multi-arkitektur tools og libraries…

SW5: BEAGLEBONE UDVIKLING
SCRIPTING-EKSEMPLER

Shell-script (fra øvelser)

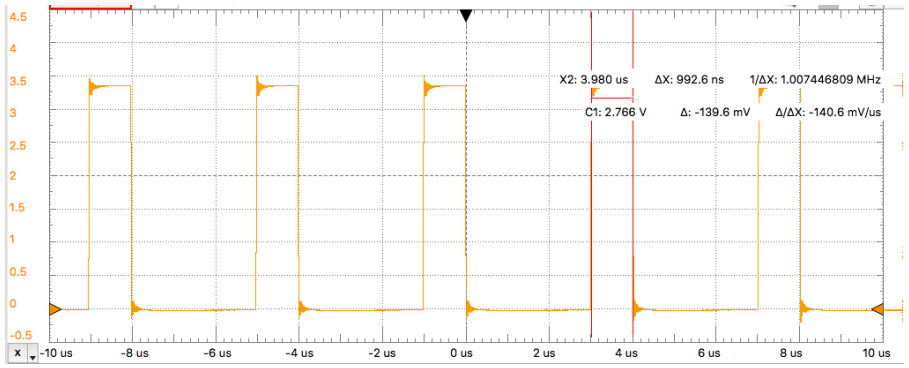
bashLED.sh

```
#!/bin/bash
LED3_PATH=/sys/class/leds/beaglebone:green:usr3

function removeTrigger
{
  echo "none" >> "$LED3_PATH/trigger"
}

echo "Shell-script til håndtering af diode 3"

if [ $# != 1 ]; then
  echo "Du må kun give et argument: "
  echo -e "bashLED <command> \n"
  echo -e "hvor <command>=on, off, flash, status"
  exit 2
fi
echo "Du har givet kommandoen: $1"

if [ "$1" == "on" ]; then
  echo "Tænder diode 3"
  removeTrigger
  echo "1" >> "$LED3_PATH/brightness"
elif [ "$1" == "off" ]; then
  echo "Slukker for diode 3"
  removeTrigger
  echo "0" >> "$LED3_PATH/brightness"
elif [ "$1" == "flash" ]; then
  echo "Blinker med diode 3"
  echo "timer" >> "$LED3_PATH/trigger"
  echo "50" >> "$LED3_PATH/delay_on"
  echo "50" >> "$LED3_PATH/delay_off"
elif [ "$1" >> "status" ]; then
  cat "$LED3_PATH/trigger";
fi
echo "Scriptet er slut"
```

pwm.sh

```
#!/bin/bash
PWM_PATH="/sys/class/pwm/pwm-0:0"
if [ "$1" == "on" ]; then
  echo 4000 > ${PWM_PATH}/period
  echo 1000 > ${PWM_PATH}/duty_cycle
  echo 1 > ${PWM_PATH}/enable
  echo "PWM Period is now $(cat ${PWM_PATH}/period) ns."
  echo "PWM Duty-cycle is now $(cat ${PWM_PATH}/duty_cycle) ns out of total period."
elif [ "$1" == "off" ]; then
  echo 0 > ${PWM_PATH}/enable;
fi
```

```
janus@beaglebone:~/projects/E3ISD1/pwm$ ./pwm.sh on
PWM Period is now 4000 ns.
PWM Duty-cycle is now 1000 ns out of total period.
```



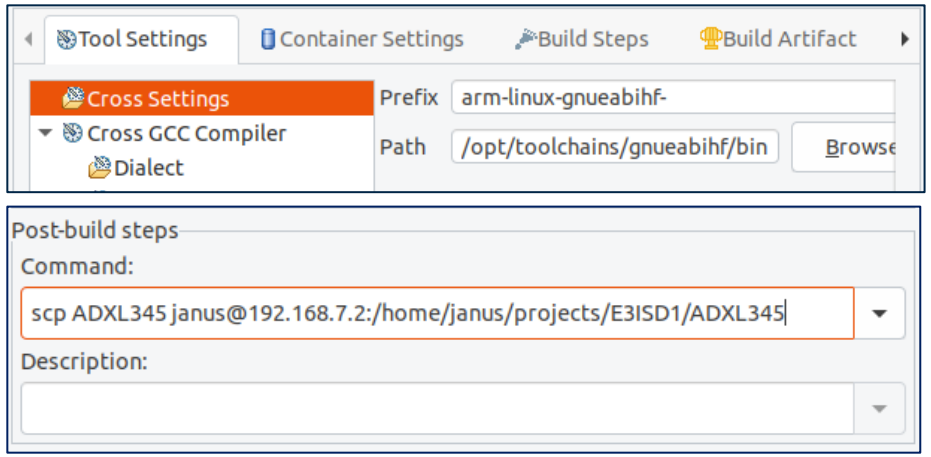
Python (eksempel fra mikroprojekt)

handler.py

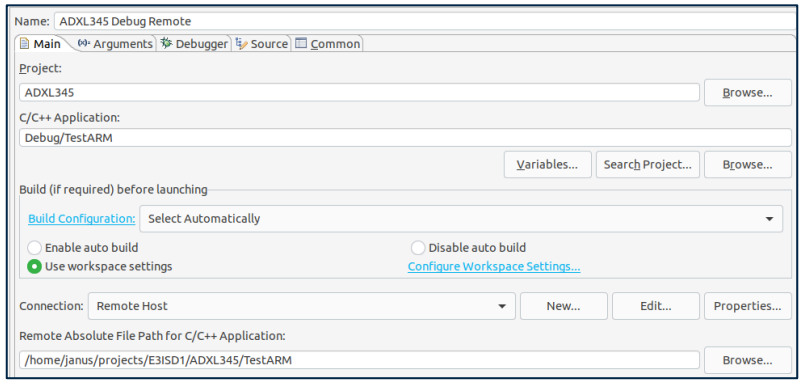
```
from client import MqttClient
import Adafruit_BBIO.GPIO as GPIO

def setup():
    GPIO.setup("USR3", GPIO.OUT)


def on_message_callback(client, userdata, message):
    # The received MQTT message is binary and must be decoded
    msg = message.payload.decode("utf-8")
    print(msg)
    if msg == "ON":
        GPIO.output("USR3", GPIO.HIGH)
    elif msg == "OFF":
        GPIO.output("USR3", GPIO.LOW)
    else:
        print("No action taken!")


def handler_sub_main():
    # Define a subscriber that listens to all subjects
    subscriber = MqttClient("MessageHandler1", on_message_callback)

    # Only subscribe to relevant inbound messages
    subscriber.subscribe("mikroprojekt/inbound")

    print("Starting listening loop")
    subscriber.loop()

def handler_pub_main():
    #Sends off a single message and quits
    publisher = MqttClient("MessagePublisher", on_message_callback)
    publisher.publish("mikroprojekt/outbound", "Hello World!")

if __name__ == '__main__':
    setup()
    handler_sub_main()
```

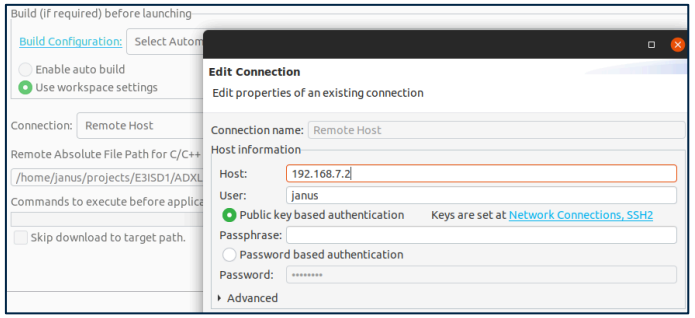
SW5: BEAGLEBONE UDVIKLING
CROSS-COMPILING

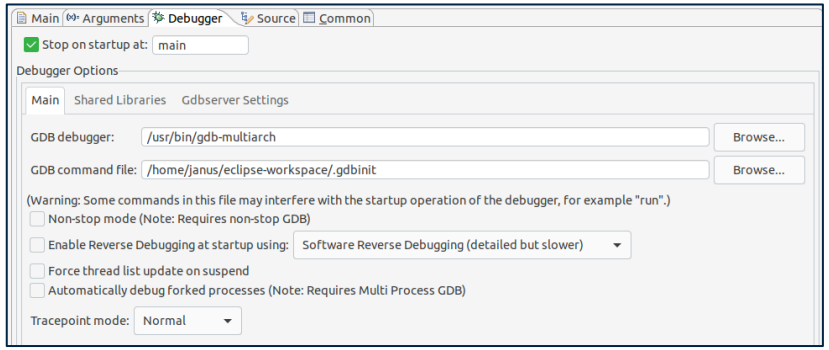
Compiler-setup (ADXL345 over I2C)



Debugger-setup (ADXL345-eksempel)

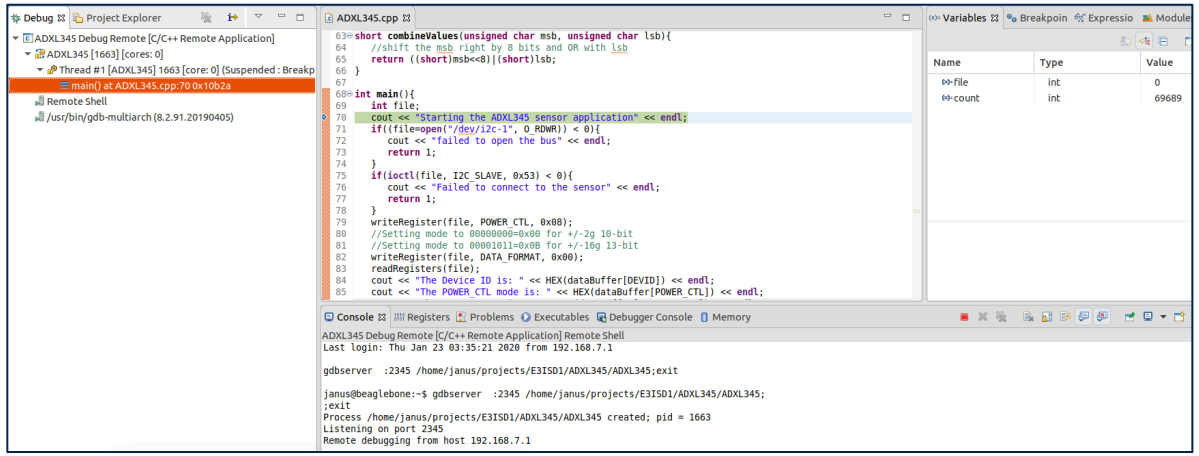


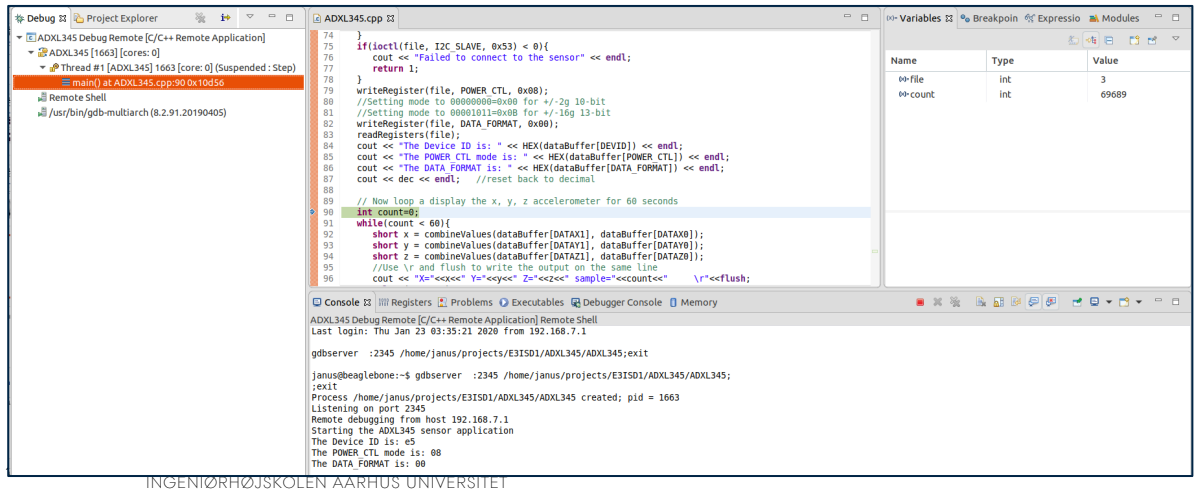




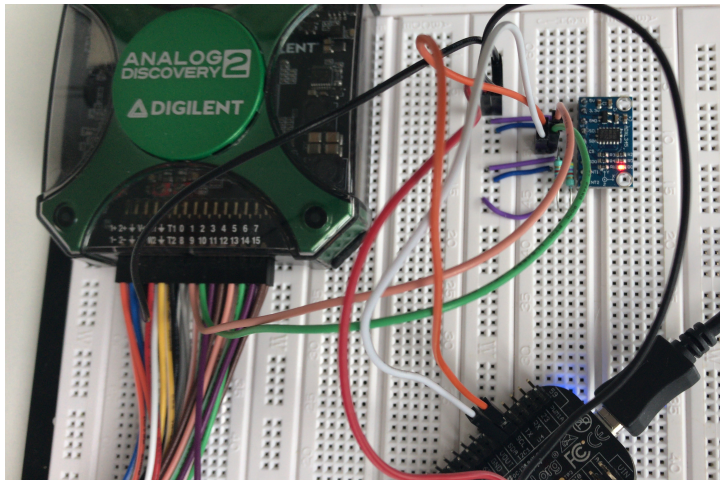
SW5: BEAGLEBONE UDVIKLING
REMOTE DEBUGGING

Debugging-flow (ADXL345 over I2C)

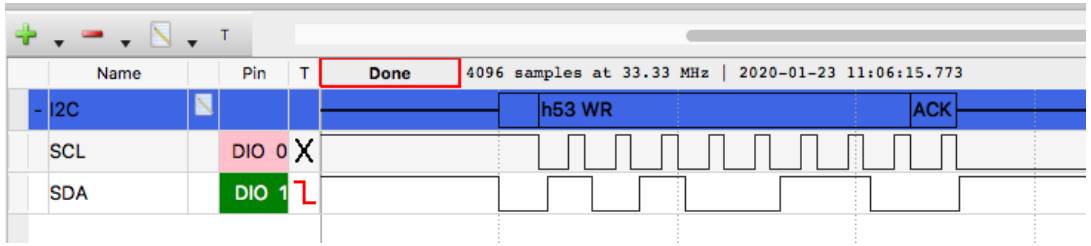




Hardware




```
janus@beaglebone:~/projects/E3ISD1/ADXL345$ ./ADXL345
Starting the ADXL345 sensor application
The Device ID is: e5
The POWER_CTL mode is: 08
The DATA_FORMAT is: 00

X=-5 Y=7 Z=249 sample=24
```



SW5: BEAGLEBONE UDVIKLING
ØVRIGE MULIGHEDER

Øvrige muligheder

- Debugging på target: gdb
- Profiling, memory-debugging: Valgrind (både på PB og via CDT)
- Emulering af ARM-processor med QEMU (qemu-static)

SW5: EMNE
OPSUMMERING OG PERSPEKTIVERING

SW6: MIKROPROJEKT
INDHOLD

Teamprojekt: Marc, Tommy, Janus

1. Krav / idé / formål
2. Analyse og design
3. Klargørelse af PocketBeagle
4. Implementering/løsning
5. Demo
6. Opsummering og perspektivering

6	Projektopgave	Gennemgå projektopgaven. Forklar • hvilke(t) krav du vil opfylde, • analysen der bringer dig frem til den valgte løsning, • det design du er nået frem til og endelig • demonstrér din løsning

SW6: MIKROPROJEKT
KRAV

Punkt	Detaljer
Formål	Vi vil benytte Beaglebone til at kommunikere med "Webinterface" specificeret i PRO3: Proof-of-concept, prototype og dokumentation • Tager vores egen medicin ift. Pro3-specs...
Målsætning	Vi indsætter Beaglebone-platformen som en "Prøvestand" = klient, og på klienten implementerer vi funktionalitet som nedenstående med Python (interpreted).
Funktionalitet	Ambitionen er at implementere 1-3 i mikroprojektet 1. MQTT Publisher 1. Klient skaber netværksforbindelse til en MQTT-broker. 2. Klient sender "publish"-beskeder til broker. 2. MQTT Subscriber 1. • Klient "subscriber" på et emne hos broker. 2. • Klient modtager beskeder via broker 3. GPIO-interaktion (beskeder via MQTT benyttes til at interagere med GPIO) 1. Tænd lysdiode på kommando

SW6: MIKROPROJEKT
ANALYSE OG DESIGN

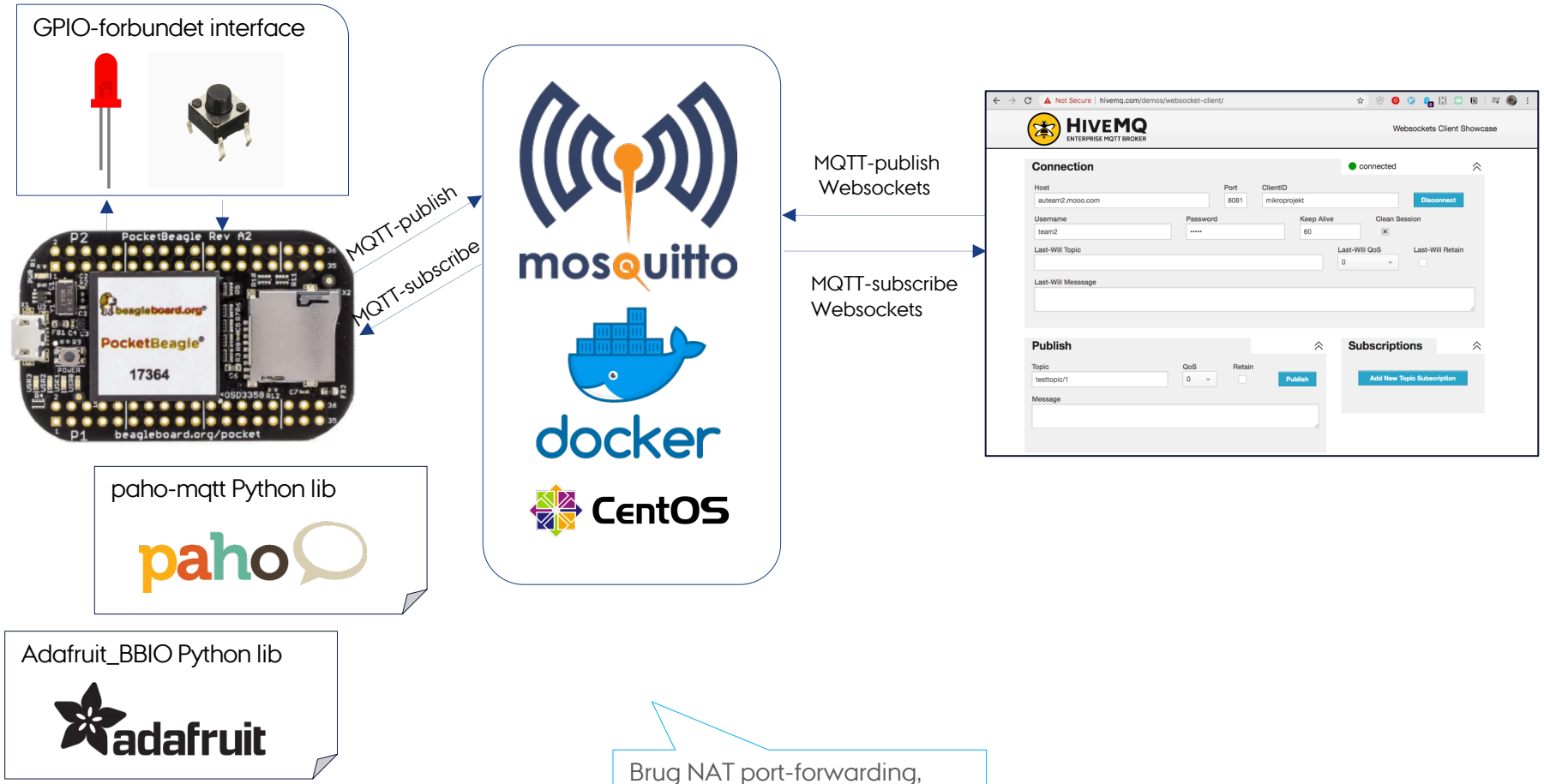
Analyse

Idé, protokol, m.v. udvalgt i Pro3

- Vi er her "på den anden side" af vores egen Pro3-implementering.

Funktionalitet udvikles på Beaglebone (Debian) i dette mikroprojekt, så andre teams har noget at læne sig op ad.

- Kommunikerer vha. CentOS-Docker-Mosquitto stack fra Pro3



https://adafruit-beaglebone-io-python.readthedocs.io/en/latest/

SW6: MIKROPROJEKT
ANALYSE OG DESIGN

Design-valg

Sprog

Vi benytter Python, fordi:

- Kodegenbrug
- Ingen høje performancekrav, så Python er OK
- Ingen behov for at compile libmosquitto, eller cross-compile
- Ingen constraints ift. hvad de andre projektteams vil benytte af sprog til deres implementering
  - Portabilitet for Python er høj (nemt at få op at køre andre steder)

Interface

GPIO (men kunne også have været I2C, osv)...

- Vi vælger at blinke med en LED, fordi det er nemt at demo, og beviser at vi kan bruge GPIO
- GPIO-input ville have være næste lige så nemt

SW6: MIKROPROJEKT
KLARGØRELSE AF VÆRKTØJER

Forberedelse

Build tools

```
janus@beaglebone:~/projects/E3ISD1$ sudo apt-get install build-essential python3-dev python3-setuptools
python3-pip python3-smbus
```

Libraries

requirements.txt

```
1  paho-mqtt
2  adafruit_BBIO
```

```
janus@beaglebone:~/projects/E3ISD1/mikroprojekt$ vim requirements.txt
janus@beaglebone:~/projects/E3ISD1/mikroprojekt$ sudo python3 -m pip install -r requirements.txt
```

Test

blink_pocket.py

```
1  import Adafruit_BBIO.GPIO as GPIO
2  import time
3
4  def setup():
5      for i in ["USR0", "USR1", "USR2", "USR3"]:
6          GPIO.setup(i, GPIO.OUT)
7
8  def pulse():
9      setup()
10     counter = 0
11
12     while True:
13         GPIO.output("USR3", GPIO.HIGH)
14         print("High")
15         time.sleep(0.5)
16         GPIO.output("USR3", GPIO.LOW)
17         print("Low")
18         time.sleep(0.5)
19         counter += 1
20
21 if __name__ == '__main__':
22     pulse()
```

```
janus@beaglebone:~/projects/E3ISD1/mikroprojekt$ sudo python3 blink_pocket.py
High
Low
High
Low
```

Observér, at USR3-dioden blinker i takt

En "svaghed" ved Adafruit_BBIO er at det umiddelbart skal køres som *sudo*
- medmindre man fikser nogle systemsettings...
- udev-rules, setuid, osv.

SW6: MIKROPROJEKT
IMPLEMENTERING / LØSNING

MQTT-klient

client.py

```
"""
This module implements a custome MQTT client, which can be of two types:
- Subscriber: Listens for messages on a given topic
- Publisher: Sends messages on a given topic
"""

import paho.mqtt.client as mqtt

class MqttClient():

    # These should be gotten from the environment ideally
    broker_address = "auteam2.mooo.com"
    broker_port = "1883"
    username = "team2"
    password = "team2"

    # This method is the same for all instances of the class
    @staticmethod
    def on_connect(client, userdata, flags, rc):
        #print("Connected with result code " + str(rc))
        pass

    # For outputting log messages to console
    @staticmethod
    def on_log(client, userdata, level, buf):
        pass
        #print("log: ", buf)

    # By default, do nothing on_message
    @staticmethod
    def on_message_default(client, userdata, message):
        pass

```

```
    #Initialize the client, straight on create
    def __init__(self, name, on_message, will_message="Logging off"):
        """ __init__ Handles all setup and connection when object is initialized.
        @:param: name is the name of the client, as will be shown on the server
        @:param: on_message is the callback used when this client receives a message
        @:param: will_message is the "Last Will" message sent when client loses conn
        """
        self.client = mqtt.Client(client_id=name,
                                  clean_session=True,
                                  userdata=None,
                                  transport="tcp")
        self.client.username_pw_set(MqttClient.username, MqttClient.password)
        self.client.on_connect = MqttClient.on_connect
        self.client.on_message = on_message

        # In production, let's consider disabling logging or routing to a file
        #self.client.on_log = MqttClient.on_log
        #self.client.enable_logger()

        # This ensures, that there is some sort of goodbye on losing connection
        self.client.will_set(name, will_message)

        # Connect immediately
        self.client.connect(MqttClient.broker_address)

    def publish(self, topic, payload):
        return self.client.publish(topic, payload)

    def subscribe(self, topic):
        return self.client.subscribe(topic)

    def loop(self):
        return self.client.loop_forever(retry_first_connection=False)

    def disconnect(self):
        return self.client.disconnect()
NORMAL  client.py                    python  utf-8[unix]  49% ☰  34/
```

Handler

handler.py

```
from client import MqttClient
import Adafruit_BBIO.GPIO as GPIO

def setup():
    GPIO.setup("USR3", GPIO.OUT)


def on_message_callback(client, userdata, message):
    # The received MQTT message is binary and must be decoded
    msg = message.payload.decode("utf-8")
    print(msg)
    if msg == "ON":
        GPIO.output("USR3", GPIO.HIGH)
    elif msg == "OFF":
        GPIO.output("USR3", GPIO.LOW)
    else:
        print("No action taken!")


def handler_sub_main():
    # Define a subscriber that listens to all subjects
    subscriber = MqttClient("MessageHandler1", on_message_callback)

    # Only subscribe to relevant inbound messages
    subscriber.subscribe("mikroprojekt/inbound")

    print("Starting listening loop")
    subscriber.loop()

def handler_pub_main():
    #Sends off a single message and quits
    publisher = MqttClient("MessagePublisher", on_message_callback)
    publisher.publish("mikroprojekt/outbound", "Hello World!")

if __name__ == '__main__':
    setup()
    handler_sub_main()
```

SW6: MIKROPROJEKT

DEMO ☺

https://youtu.be/p4kX3HWeck0

https://github.com/AUTeam2/E3ISD1_mikroprojekt

SW6: MIKROPROJEKT
OPSUMMERING OG PERSPEKTIVERING

Mulige udvidelser

- Aftaste en knap via GPIO
  - Setup er `GPIO.setup("pin", GPIO.IN)`
  - Aftastning er fx
    - `GPIO.wait_for_edge("pin", GPIO.RISING)` eller
    - `GPIO.add_event_detect("pin", GPIO.FALLING)`, samt `if GPIO.event_detected("pin")`
    - `add_event_callback("pin", callback_func)`
- Analog interfacing via
  - Input: PocketBeagle's 12-bits ADC
  - Output: PocketBeagle's PWM
- Måle accelerometer via bus-IO I2C, SPI, osv...

AARHUS
UNIVERSITET