Embedded and ambient systems 2023.10.11.

Practice 3

Peripheral handling at register level

Méréstechnika és Információs Rendszerek Tanszék

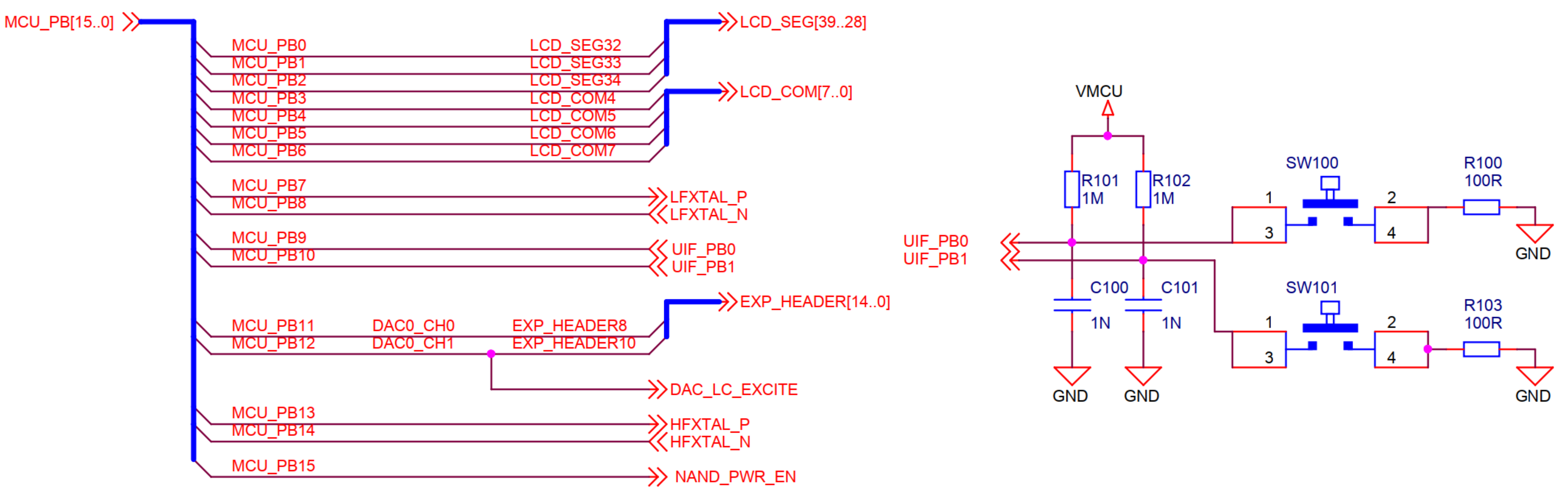
Budapest University of Technology and Economics
Department of Measurement and Information Systems

1) Peripheral handling at low level

- Useful to see how peripherals work at a register level (hidden by the high-level functions)
- See the LED-blinking-by-button project built up from empty code at a low level
- Source files needed:
  - EFM32GG-BRD2200A-A03-schematic.pdf
    - Board schematic: peripherals and their interconnection
  - EFM32GG-RM.pdf (RM=reference manual)
    - Use it as a reference, i.e., the necessary chapters are needed only to be read
    - It is a good way to understand general topics, e.g., communication (e.g. UART) used by the uC

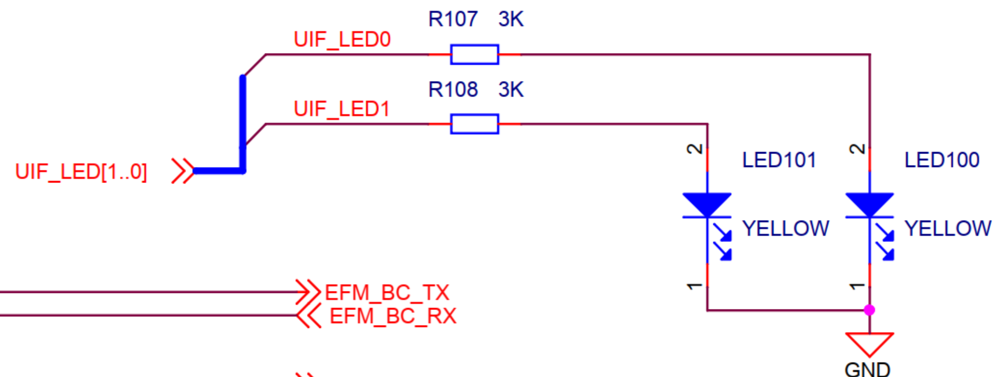
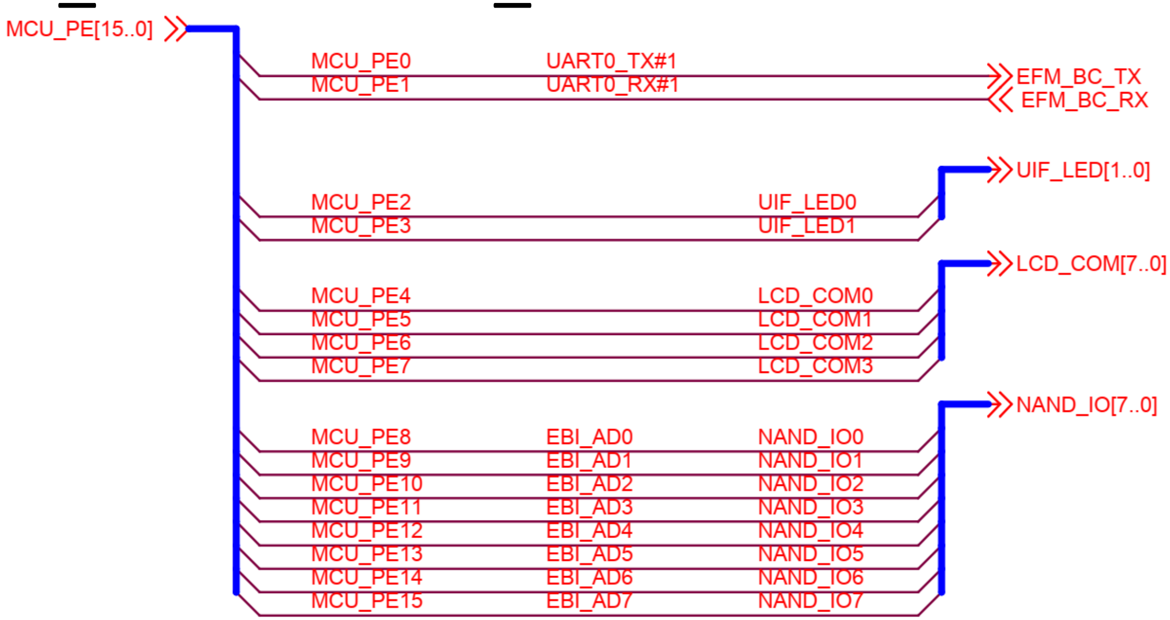
1) Physical connections on the board

- Find the connections between the uC and the buttons based on the schematic
- Buttons: connected to ‘Port B’ of GPIO peripheral
  - UIF_PB0 -> MCU_PB9
  - UIF_PB1 -> MCU_PB10

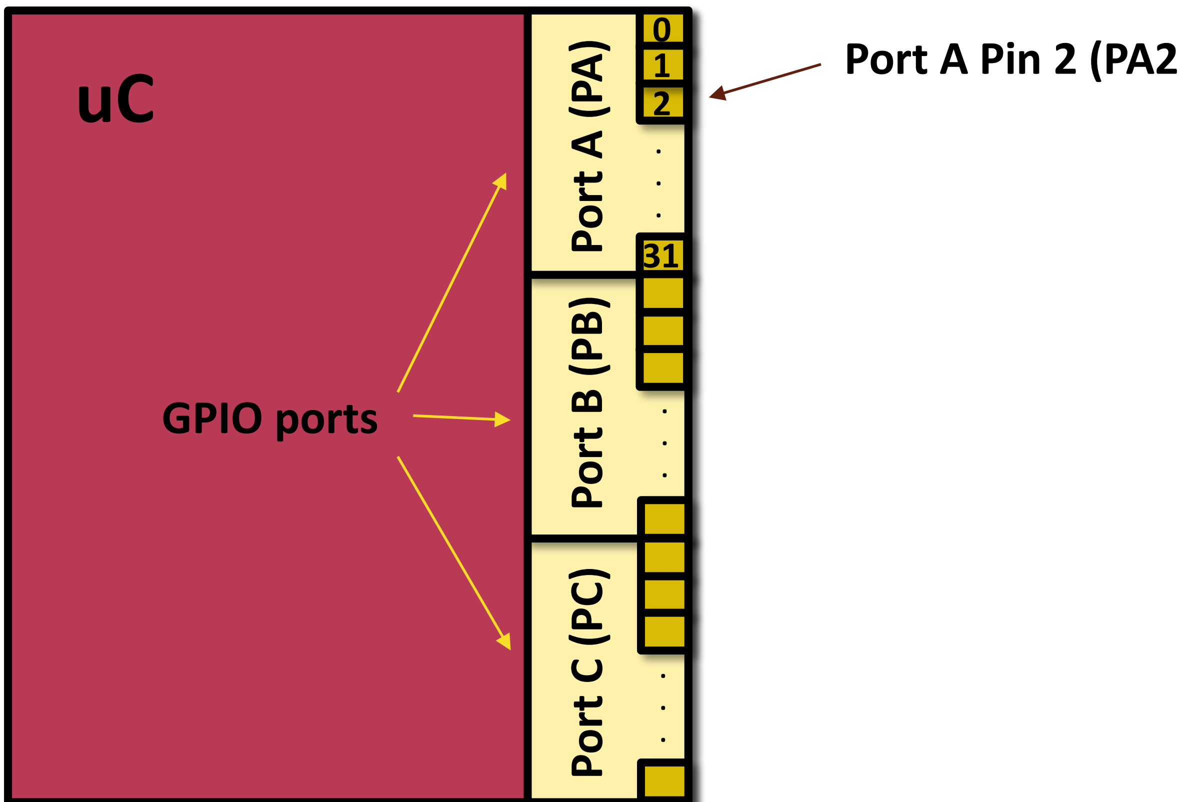


1) Physical connections on the board

- Find the connections between the uC and the LEDs based on the schematic
- LEDs: connected to ‘Port E’ of GPIO
  - UIF_LED0 -> MCU_PE2
  - UIF_LED1 -> MCU_PE3

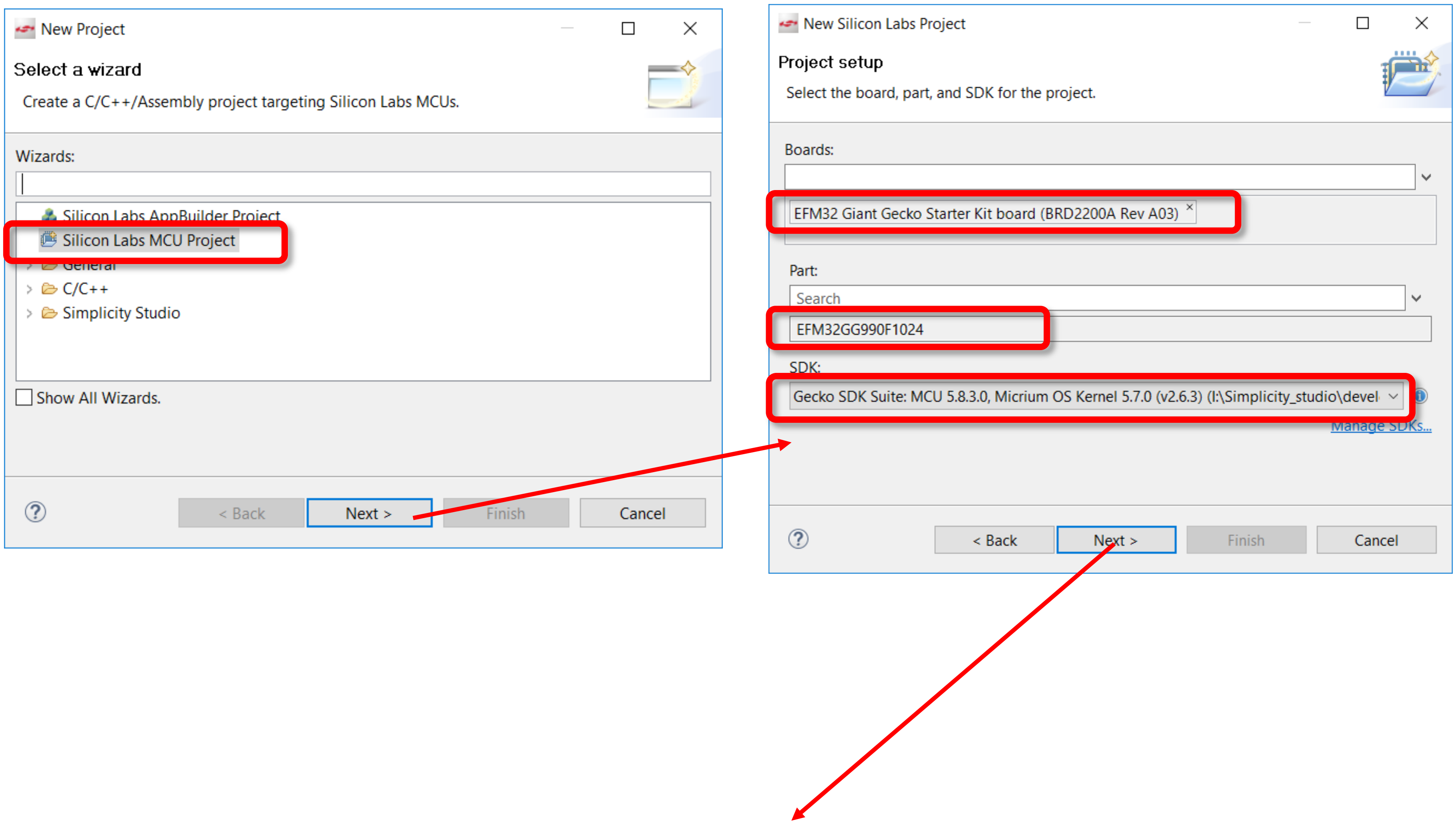



1) Physical connections on the board

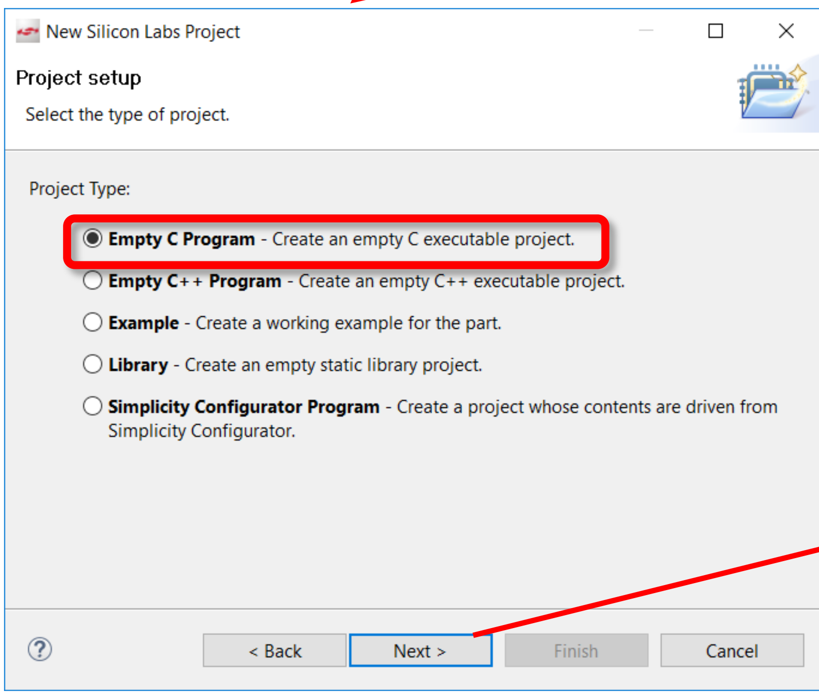


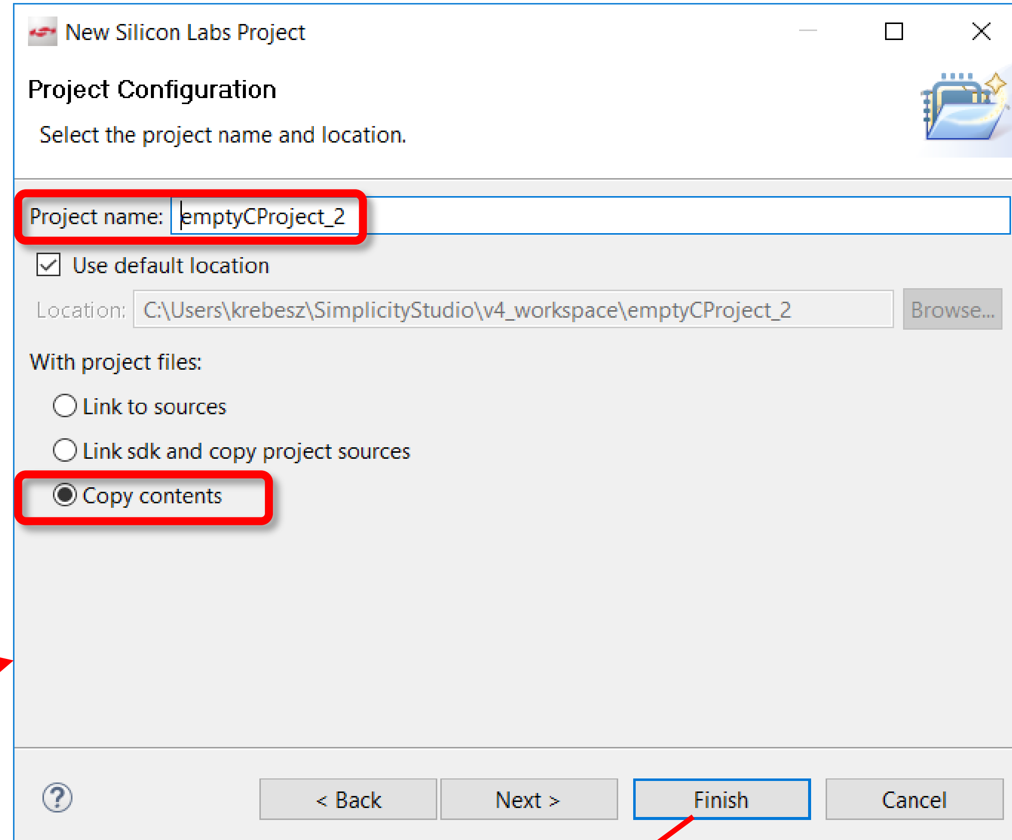
2) Start a new empty project

- File->New->Project:

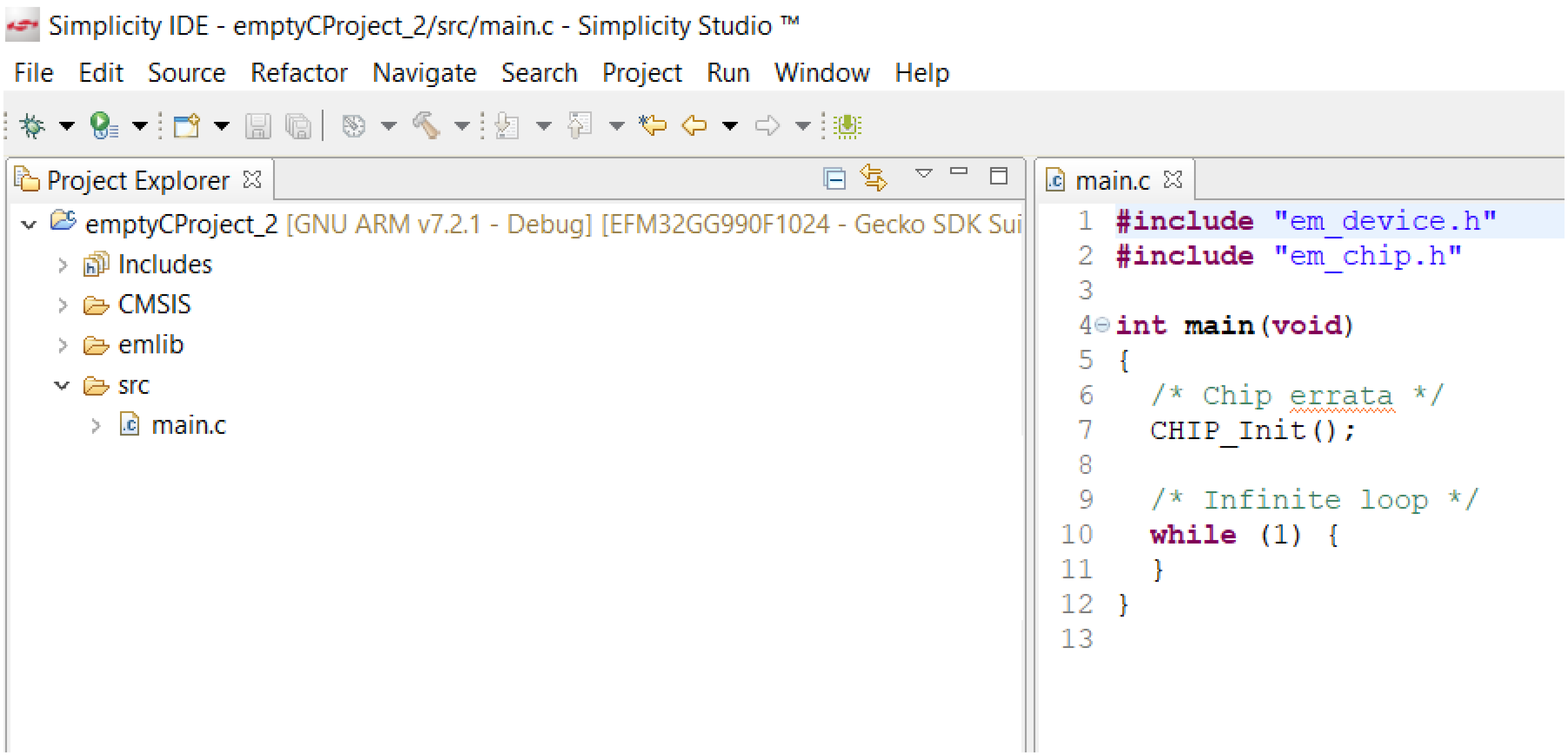


3) Start a new empty project





3) Empty project created

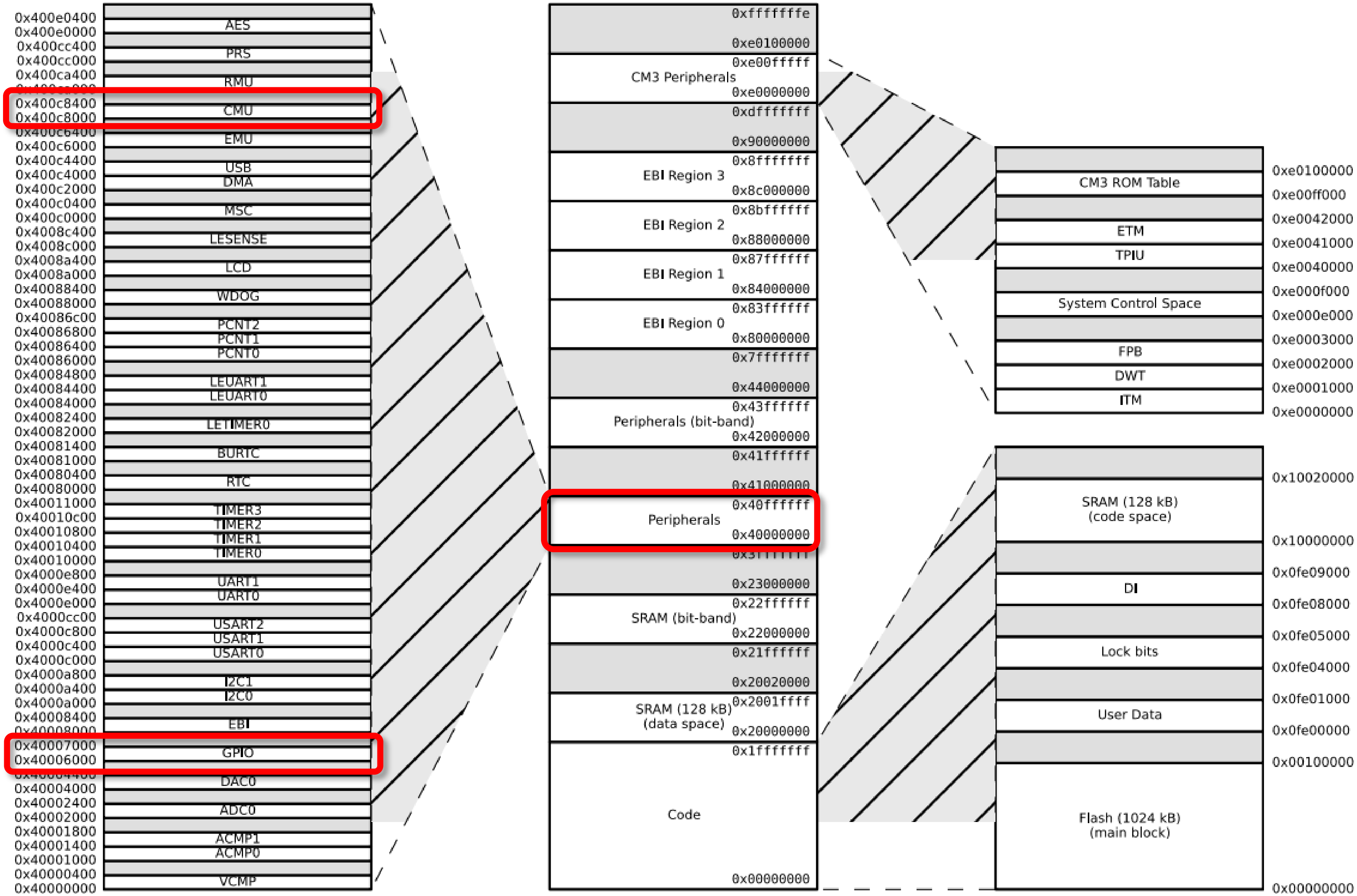


- Comment out CHIP_Init(); function

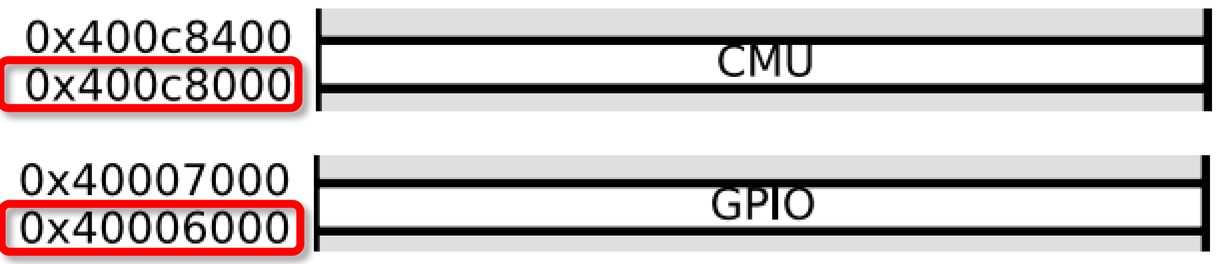
4) Get to know necessary peripherals

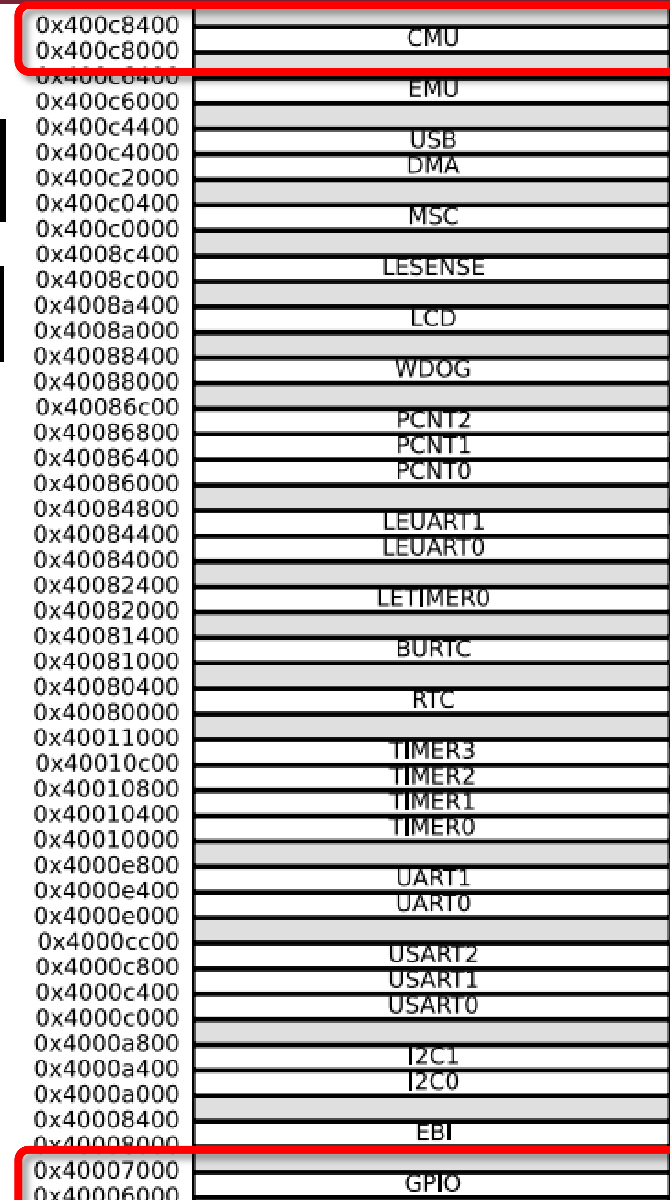
- Two peripherals are needed
  - General Purpose Input Output (GPIO)
  - Clock Management Unit (CMU)
- Check p.17 Fig.5.2 of EFM32GG-RM.pdf
  - Memory map of the system
  - 32-bit uC -> 4GB addressable memory theoretically but only a small part is physically available
  - Obviously only the physically available amount of memory is shown in the map
  - (see next slide for the map)

4) Memory map (full)



4) Memory map (CMU and GPIO regs.)





- The address space is important
  - Starts from bottom and increasing to the top
  - Base addresses of peripheral registers have to be determined
    - CMU base address: 0x400c8000
    - GPIO base address: 0x40006000

5) Base address aliases in code

- Avoid memorizing memory addresses using aliases in the code (use Tab instead of Space)

main.c

```c
#include "em_device.h"
#include "em_chip.h"

#define CMU_BASE_ADDR	0x400c8000
#define GPIO_BASE_ADDR	0x40006000

int main(void)
{
  /* Chip errata */
  //CHIP_Init();

  /* Infinite loop */
  while (1) {
  }
}

```

6) Accessing registers using base addr.

- Base address is only the start address of a certain kind of register array, like CMU registers
- To access a specific register (e.g. register for REG_A of a register array) an offset address have to be used relative to the base address
  - The address of a specific register is the base + offset address
    - e.g. REG_A -> 0x400c8000+0x044
- Note, that registers usually contain configuration bits to be set (see later)

6) Explanation for setting reg. content

- 32-bit registers are addressed
- Memory address is determined to store data there
  - Remember: base address + offset = memory address
    - Problem: this is a number for the compiler not an address
    - Solution: to turn this number into a memory address it has to be converted into a **pointer** (use * to mark a pointer)
      - In C, pointer is a variable type that points to a certain part of the memory (to a memory address where e.g. a register store data)
      - Turning a number into a pointer means forcing the change of variable type, called **cast**ing
- The way to refer to a certain register is uC dependent, its implementation has to be checked via examples, description, manual, etc.

6) Explanation for setting reg. content

- In our case a pointer can be given by:
  - (*(volatile long unsigned int *)(0x400c8000+0x044))
    - First *: a value is to be written into the memory (register) at the given address
    - volatile: avoid to be optimized out
    - long unsigned int: type of the pointer (note: 32-bit reg.)
    - Second *: this is a pointer
    - 0x400c8000+0x044 : this is the known memory address
  - The pointer itself:
    - (volatile long unsigned int *)(0x400c8000+0x044)
    - To give a value for this pointer the first * is used

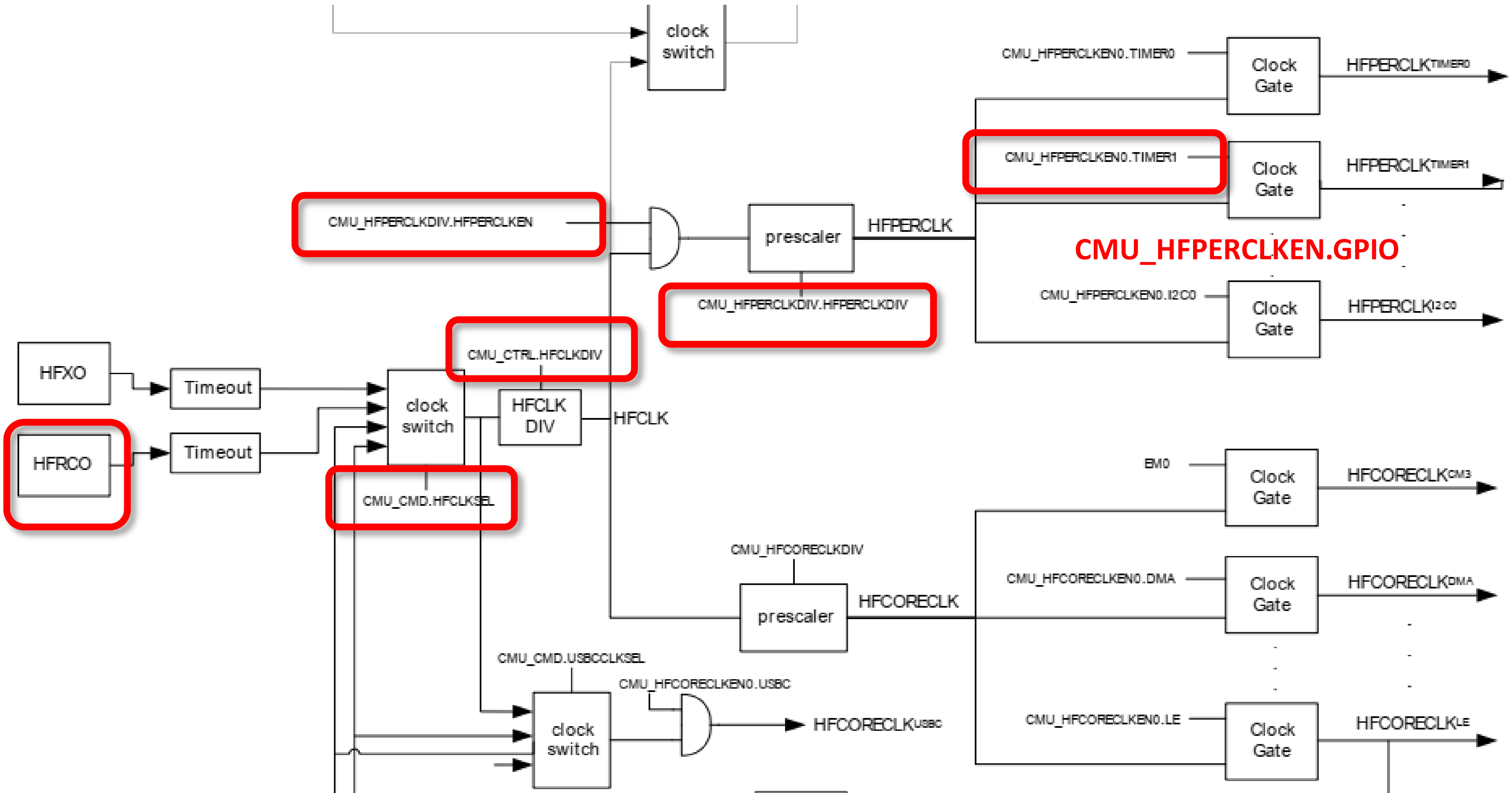
6) Explanation for setting reg. content

- Useful to make it more structured looking
  - #define REG_A (*(volatile long unsigned int *)(0x400c8000+0x044))
- Setting a bit, e.g., set Bit13
  - REG_A |=1<<13
    - |= : bitwise OR used for setting a bit
    - bbbbbbbb |= 00100000 results bb1bbbb
      where b is either 0 or 1
- Clearing a bit, e.g., clear Bit13
  - REG_A &=~(1<<13)
    - &=~ : bitwise AND of inverted values used for clearing
    - bbbbbbbb &=~ 00100000 -> bbbbbbbb &= 11011111
      -> bb0bbbbb

7) Peripheral handling - CMU

- Check p.128 Fig.11.1 of EFM32GG-RM.pdf
  - Clock distribution network is shown
  - Clock has to be provided for the peripherals
    - This is uC dependent but always has to be take care of providing CLK for the peripherals and enabling the peripherals
  - Find HFRCO: high-frequency RC osc
    - Not too much precise but readily available -> no external CLK source is needed
  - Check the signal path toward the GPIO peripheral

7) Peripheral handling - CMU



- Note: manual pages for a certain peripheral has to be read carefully how to use them

7) Peripheral handling - CMU

- Check p.136 of EFM32GG-RM.pdf
  - Registers of CMU peripheral are shown with brief description
  - Register addresses are given relative to the base address
    - E.g. CMU_CTRL addr: from 0x000 to the next register starting 0x004, which is 4bytes, i.e. 32 bits

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	CMU_CTRL	RW	CMU Control Register
0x004	CMU_HFCORECLKDIV	RW	High Frequency Core Clock Division Register
0x008	CMU_HFPERCLKDIV	RW	High Frequency Peripheral Clock Division Register
0x00C	CMU_HFRCOCTRL	RW	HFRCO Control Register
0x010	CMU_LFRCOCTRL	RW	LFRCO Control Register

7) Peripheral handling - CMU

- Use copy-paste to put the register addresses into the code
  - 0x008 CMU_HFPERCLKDIV_OFFS
  - 0x044 CMU_HFPERCLKEN0_OFFS

The offset register address is relative to the registers base address.

Offset	Name	Type	Description
0x000	CMU_CTRL	RW	CMU Control Register
0x004	CMU_HFCORECLKDIV	RW	High Frequency Core Clock Division Register
0x008	CMU_HFPERCLKDIV	RW	High Frequency Peripheral Clock Division Register
0x00C	CMU_HFRCOCTRL	RW	HFRCO Control Register
0x010	CMU_LFRCOCTRL	RW	LFRCO Control Register

7) Peripheral handling - CMU

- Check p.137 of EFM32GG-RM.pdf
  - Bit-level description of CMU registers
  - Check default values: values after Reset

11.5.1 CMU_CTRL - CMU Control Register

Offset	Bit Position																															
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset		0		0		0x0				0x0			0x3		0	0x0			0	0x0		0x3			0	0x1			0x3		0x0	
Access		RW		RW		RW				RW			RW		RW	RW			RW	RW		RW			RW	RW			RW		RW	
Name		HFLE		DBGCLK		CLKOUTSEL1				CLKOUTSEL0			LFXOTIMEOUT		LFXOBUFCUR	HFCLKDIV			LFXOBOOST	LFXOMODE		HFXOTIMEOUT			HFXOGLITCHDETEN	HFXOBUFCUR			HFXOBOOST		HFXOMODE	

Bit	Name	Reset	Access	Description
22:20	CLKOUTSEL0	0x0	RW	**Clock Output Select 0**

Controls the clock output multiplexer. To actually output on the pin, set CLKOUT0PEN in CMU_ROUTE.

Value	Mode	Description
0	HFRCO	HFRCO (directly from oscillator).

7) Peripheral handling - CMU

- Check p.140 of EFM32GG-RM.pdf
  - Enable CLK

11.5.3 CMU_HFPERCLKDIV - High Frequency Peripheral Clock Division Register

Offset	Bit Position																															
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																								1					0x0			
Access																								RW					RW			
Name																								HFPERCLKEN					HFPERCLKDIV			

Bit	Name	Reset	Access	Description
31:9	*Reserved*	*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*		
8	HFPERCLKEN	1	RW	**HFPERCLK Enable**
	Set to enable the HFPERCLK.			

7) Peripheral handling - CMU

- Check p.150 of EFM32GG-RM.pdf
  - Enable CLK for GPIO

11.5.18 CMU_HFPERCLKEN0 - High Frequency Peripheral Clock Enable Register 0

Offset	Bit Position																														
0x044	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset															0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access															RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Name															DAC0	ADC0	PRS	VCMP	GPIO	I2C1	I2C0	ACMP1	ACMP0	TIMER3	TIMER2	TIMER1	TIMER0	UART1	UART0	USART2	USART1	USART0

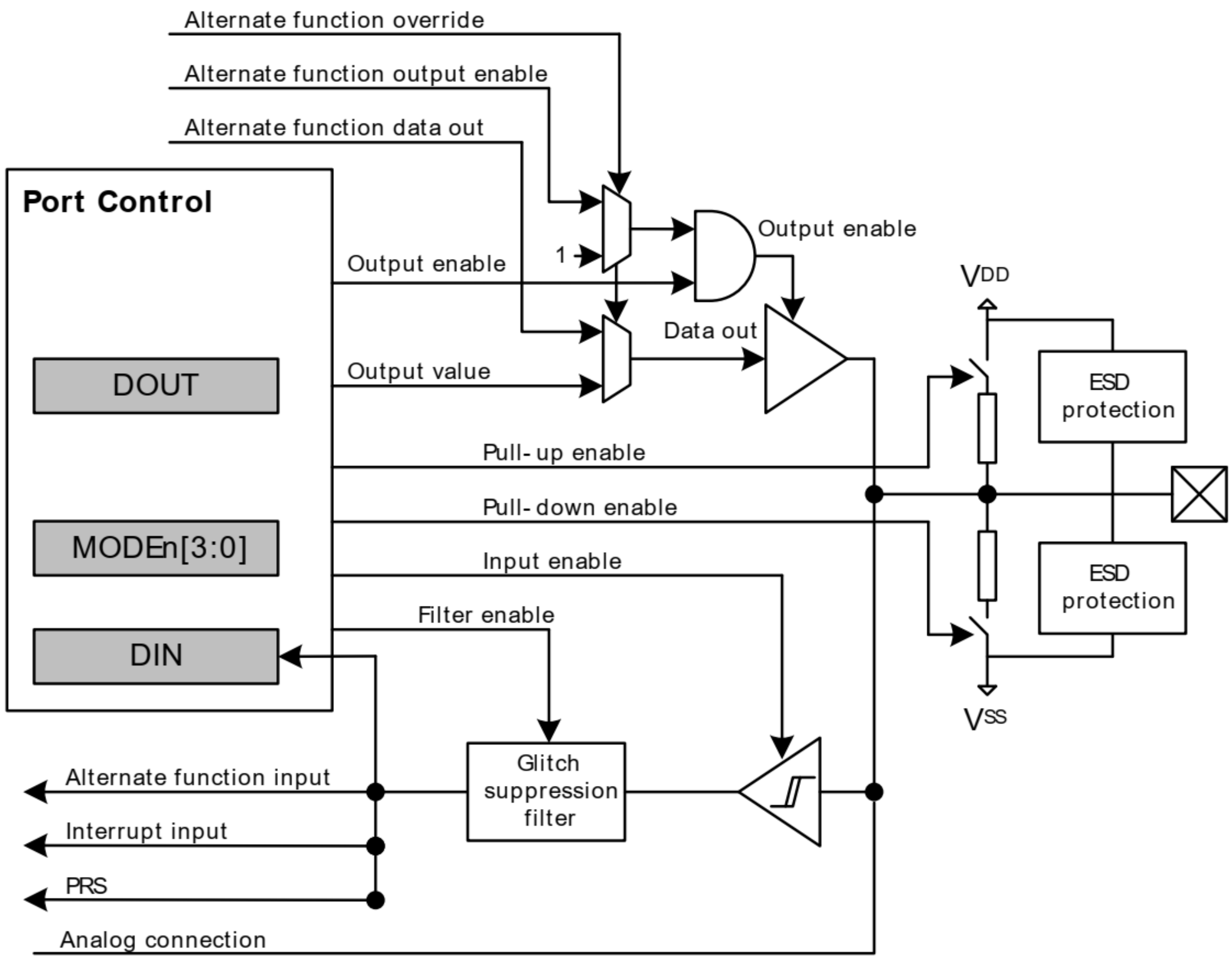
Bit	Name	Reset	Access	Description
13	GPIO	0	RW	**General purpose Input/Output Clock Enable**
	Set to enable the clock for GPIO.			

7) Peripheral handling - CMU

- New #define for pointer to get access to CMU register -> enabling
  - #define CMU_ HFPERCLKDIV (*(volatile unsigned long int*)(0x400c8000 + 0x008))
    - Note: CMU_BASE_ADDR+ CMU_HFPERCLKDIV_OFFS
  - #define CMU_HFPERCLKEN0 (*(volatile unsigned long int*)(0x400c8000 + 0x044))
    - Note: CMU_BASE_ADDR+CMU_HFPERCLKEN0_OFFS
- In the *main* function: CMU_HFPERCLKDIV |= (1<<8);
  CMU_HFPERCLKEN0 |= (1<<13);
- Comment out all #include not to cause any trouble
- Check for errors by compiling

8) Peripheral handling - GPIO

- See pp.756-758 and Fig. 32.1 of EFM32GG-RM.pdf



8) Peripheral handling - GPIO

- Register map of GPIO (see p.764): offsets only!

Offset	Name		Type	Description
0x02C	GPIO_PB_MODEH	Register for push button	RW	Port Pin Mode High Register
0x040	GPIO_PB_DIN	Register for push button	R	Port Data In Register
0x094	GPIO_PE_MODEL	Register for push LED	RW	Port Pin Mode Low Register
0x09C	GPIO_PE_DOUT	Register for LED	RW	Port Data Out Register

- Use #define again
  - #define GPIO_PB_MODEH_OFFS 0x02C
  - #define GPIO_PB_DIN_OFFS 0x040
  - #define GPIO_PE_MODEL_OFFS 0x094
  - #define GPIO_PE_DOUT_OFFS 0x09C

8) Peripheral handling - GPIO

- Pointers to be used have to be created in the same way as in case of CMU
  - #define GPIO_PB_MODEH (*(volatile long unsigned int *)(GPIO_BASE_ADDR+GPIO_PB_MODEH_OFFS))
  - #define GPIO_PB_DIN (*(volatile …*)(…+…_OFFS))
  - #define GPIO_PE_MODEL (*(…
  - #define GPIO_PE_DOUT (*(…

8) Peripheral handling - GPIO

- Check pp. 765-766, the GPIO_Px_CTRL (port control) register: drive modes can be set

Offset	Bit Position																															
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0x0	
Access																															RW	
Name																															DRIVEMODE	

Bit	Name	Reset	Access	Description
31:2	*Reserved*			*To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)*
1:0	DRIVEMODE	0x0	RW	**Drive Mode Select**

Select drive mode for all pins on port configured with alternate drive strength.

Value	Mode	Description
0	STANDARD	6 mA drive current
1	LOWEST	0.1 mA drive current
2	HIGH	20 mA drive current
3	LOW	1 mA drive current

8) Peripheral handling in general- GPIO

- Check pp. 766, the GPIO_Px_MODEL register

Offset	Bit Position																															
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x0				0x0				0x0				0x0				0x0				0x0				0x0				0x0			
Access	RW				RW				RW				RW				RW				RW				RW				RW			
Name	MODE7				MODE6				MODE5				MODE4				MODE3				MODE2				MODE1				MODE0			

Bit	Name	Reset	Access	Description
31:28	MODE7	0x0	RW	**Pin 7 Mode**

Configure mode for pin 7. Enumeration is equal to MODE0.

Value	Mode	Description **4bits->16 different modes**
0	DISABLED	Input disabled. Pullup if DOUT is set.
1	INPUT	Input enabled. Filter if DOUT is set
2	INPUTPULL	Input enabled. DOUT determines pull direction
3	INPUTPULLFILTER	Input enabled with filter. DOUT determines pull direction
4	PUSHPULL	Push-pull output

8) Peripheral handling push button- GPIO

- Check pp. 767, the GPIO_Px_MODEH register

Offset	Bit Position																															
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x0				0x0				0x0				0x0				0x0				0x0				0x0				0x0			
Access	RW				RW				RW				RW				RW				RW				RW				RW			
Name	MODE15				MODE14				MODE13				MODE12				MODE11				MODE10				MODE9				MODE8			

11:8	MODE10	0x0	RW	**Pin 10 Mode**
	Configure mode for pin 10. Enumeration is equal to MODE8.			
7:4	MODE9	0x0	RW	**Pin 9 Mode**

Push buttons are conncted to pins 9 and 10 -> GPIO_Px_MODEH should be used

- Note: the MODEs are the same as before

8) Peripheral handling push button - GPIO

- Push button
  - Pins has to be set as inputs
  - Use MODEH register of port B
- After CLK enable, use GPIO_PB_MODEH |= ?
  - Pin 9 (10) can be configured by bit group [7:4] [11:8]
  - INPUT -> value is 1

Value	Mode	Description
0	DISABLED	Input disabled. Pullup if DOUT is set.
1	INPUT	Input enabled. Filter if DOUT is set
2	INPUTPULL	Input enabled. DOUT determines pull direction
3	INPUTPULLFILTER	Input enabled with filter. DOUT determines pull direction
4	PUSHPULL	Push-pull output

  - Use GPIO_PB_MODEH|=(1<<4); //PB9 conf as input
  - Use GPIO_PB_MODEH|=(1<<8); //PB10 conf as input

8) Peripheral handling LED - GPIO

- Check pp. 767, the GPIO_Px_MODEL register

Offset	Bit Position							
0x004	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
Reset	0x0	0x0	0x0	0x0	0x0	0x0	0x0	0x0
Access	RW	RW	RW	RW	RW	RW	RW	RW
Name	MODE7	MODE6	MODE5	MODE4	MODE3	MODE2	MODE1	MODE0

15:12	MODE3	0x0	RW	**Pin 3 Mode**
	Configure mode for pin 3. Enumeration is equal to MODE0.			
11:8	MODE2	0x0	RW	**Pin 2 Mode**
	Configure mode for pin 2. Enumeration is equal to MODE0.			

LEDs are conncted to pins 2 and 3 -> GPIO_Px_MODEL should be used

- Note: the MODEs are the same as before

8) Peripheral handling LED - GPIO

- LEDs
  - Pins has to be set as outputs: pin 2 and 3 in Port E
  - Use MODEL reg of port E
- After CLK enable, use GPIO_PE_MODEL |= ?
  - Pin 2 (3) can be configured by bit group [11:8] [15:12]
  - Pushpull mode has to be used whose value is a 4

Value	Mode	Description
0	DISABLED	Input disabled. Pullup if DOUT is set.
1	INPUT	Input enabled. Filter if DOUT is set
2	INPUTPULL	Input enabled. DOUT determines pull direction
3	INPUTPULLFILTER	Input enabled with filter. DOUT determines pull direction
4	PUSHPULL	Push-pull output

  - Use GPIO_PE_MODEL |= (4<<8); //PE2 conf as output
  - Use GPIO_PE_MODEL |= (4<<12); //PE3 conf as output

8) Peripheral handling - GPIO

- LEDs' default value should be set
  - Check p.768
    - GPIO_Px_DOUT
      - Data output on port
    - GPIO_Px_DOUTSET
      - Write bits to 1 to set corresponding bits in GPIO_Px_DOUT. Bits written to 0 will have no effect.
    - GPIO_Px_DOUTCLR
      - Write bits to 1 to clear corresponding bits in GPIO_Px_DOUT. Bits written to 0 will have no effect.
  - GPIO_PE_DOUT|=(1<<2); //LED0 set
  - GPIO_PE_DOUT|=(1<<3); //LED1 set

9) Operation at a code level

- What should be written in the while loop?
  - Read the status of the button (pushed/released) from the corresponding register bit and control the LED based on button state (on/off)

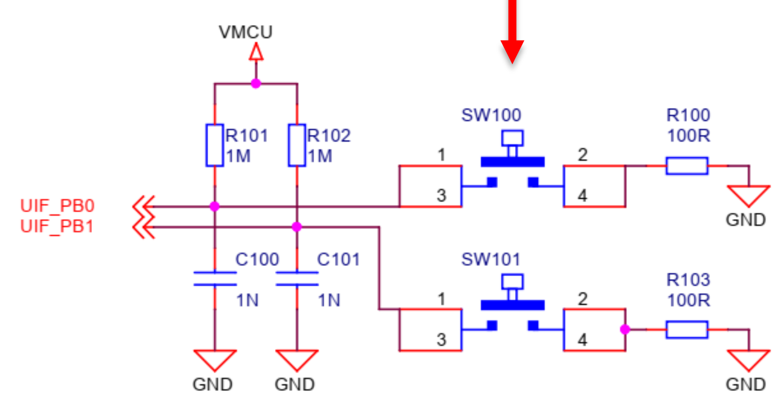
```
while (1) {

        if (GPIO_PB_DIN & (1<<9)){
                GPIO_PE_DOUT &= ~(1 << 3);
        } else {
                GPIO_PE_DOUT |= 1 << 3;
        }

        if (GPIO_PB_DIN & (1<<10)){
                GPIO_PE_DOUT &= ~(1 << 2);
        } else {
                GPIO_PE_DOUT |= 1 << 2;
        }

}
```

Here it should be checked based on the schematic that what is the value of the push button when
-pushed (->low)
-released (->high)



10) Some extra

- Using GPIO_Px_CTRL register the drive strength can be set to control the luminance of the LED
  - Check p. 767

32.5.1 GPIO_Px_CTRL - Port Control Register

Offset	Bit Position																															
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset																															0x0	
Access																															RW	
Name																															DRIVEMODE	

Bit	Name	Reset	Access	Description
31:2	*Reserved*	*To ensure compatibility with future devices, always write bits to 0.*		
1:0	DRIVEMODE	0x0	RW	**Drive Mode Select**

Select drive mode for all pins on port configured with alternate drive strength.

Value	Mode	Description
0	STANDARD	6 mA drive current
1	LOWEST	0.1 mA drive current
2	HIGH	20 mA drive current
3	LOW	1 mA drive current

11) Reference code

```
//#include "em_device.h"
//#include "em_chip.h"

#define CMU_BASE_ADDR 0x400c8000
#define GPIO_BASE_ADDR 0x40006000

#define CMU_HFPERCLKDIV (*(volatile unsigned long int*)(0x400c8000 + 0x008))
#define CMU_HFPERCLKEN0 (*(volatile unsigned long int*)(0x400c8000 + 0x044))

#define GPIO_PB_MODEH (*(volatile unsigned long int*)(0x40006000 + 0x02C))
#define GPIO_PB_DIN   (*(volatile unsigned long int*)(0x40006000 + 0x040))

#define GPIO_PE_MODEL (*(volatile unsigned long int*)(0x40006000 + 0x094))
#define GPIO_PE_DOUT  (*(volatile unsigned long int*)(0x40006000 + 0x09C))

int main(void)
{
  /* Chip errata */
  //CHIP_Init();
    CMU_HFPERCLKDIV |= 1 << 8; // periferal clk enable
    CMU_HFPERCLKEN0 |= 1 << 13; // GPIO clk enable
```

11) Reference code (cont'd)

```
//
    GPIO_PE_MODEL |= 4 << 8; // port E pin 2: pushpull output: page 766
    GPIO_PE_MODEL |= 4 << 12;// port E pin 3: pushpull output

    GPIO_PE_DOUT |= 1 << 2; // port E pin 2: high
    GPIO_PE_DOUT |= 1 << 3; // port E pin 3: high

    GPIO_PB_MODEH |= 1 << 4; // port B pin 9: input: page 67
    GPIO_PB_MODEH |= 1 << 8;// port B pin 10: input


  /* Infinite loop */
  while (1) {

        if (GPIO_PB_DIN & (1<<9)){
            GPIO_PE_DOUT &= ~(1 << 3);
        } else {
            GPIO_PE_DOUT |= 1 << 3;
        }

        if (GPIO_PB_DIN & (1<<10)){
            GPIO_PE_DOUT &= ~(1 << 2);
        } else {
            GPIO_PE_DOUT |= 1 << 2;
        }

  }
}
```