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

© BME-MIT



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:
 - o EFM32GG-BRD2200A-A03-schematic.pdf
 - Board schematic: peripherals and their interconnection
 - o 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



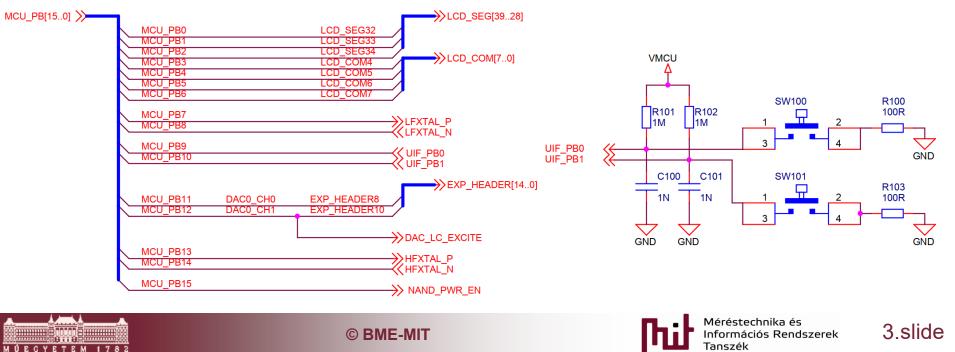




1) Physical connections on the board

- Find the connections between the uC and the buttons based on the schematic
- Buttons: connected to 'Port B' of GPIO peripheral

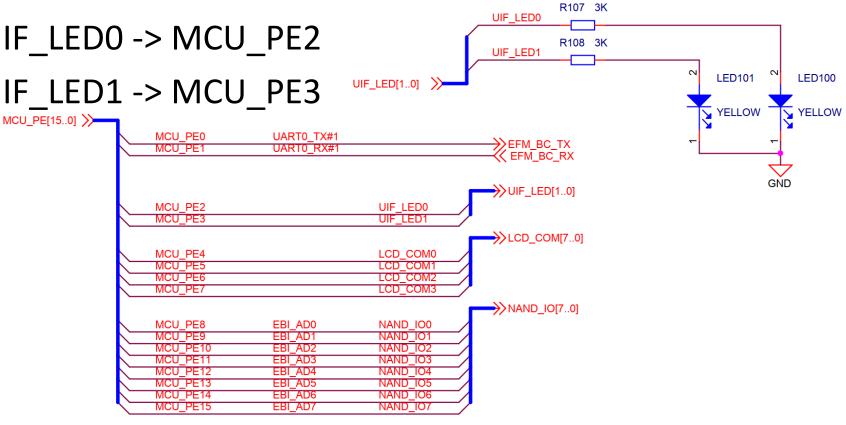
 UIF_PB0 -> MCU_PB9
 UIF_PB1 -> MCU_PB10



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** ○ UIF LEDO -> MCU PE2 **UIF LED1** ○ UIF LED1 -> MCU PE3 UIF LED[1..0]

© BME-MIT



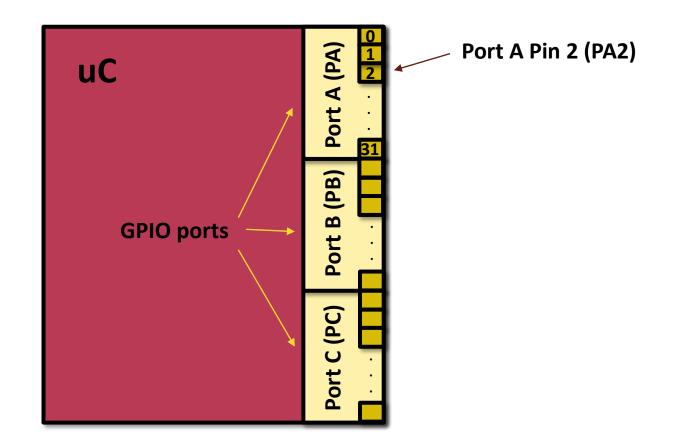
Méréstechnika és

Tanszék

Információs Rendszerek

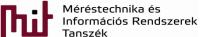


1) Physical connections on the board









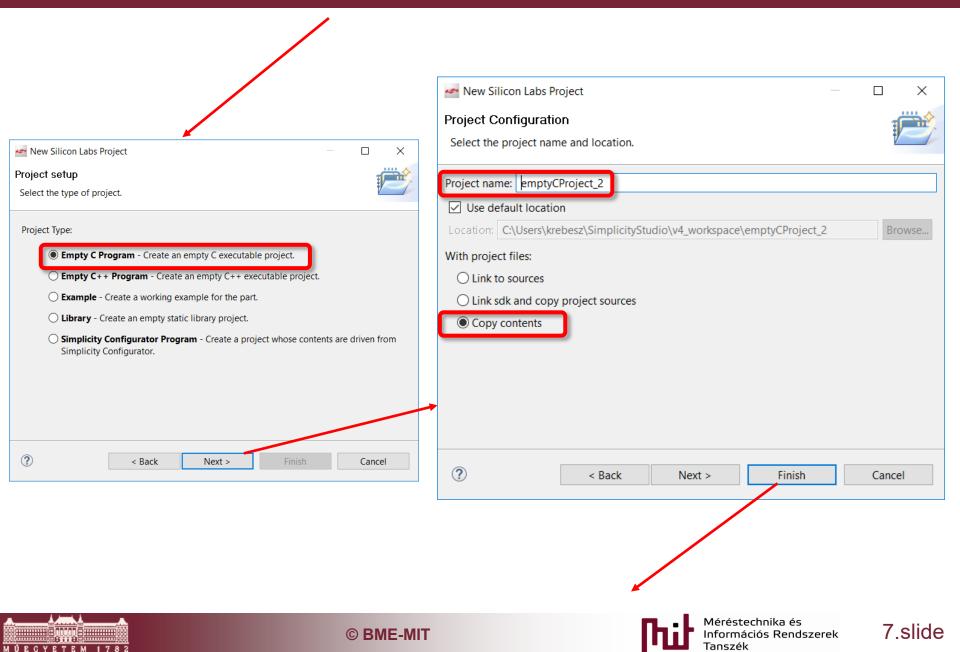
2) Start a new empty project

File->New->Project:

мÚ

New Project		New Silicon Labs Project -	
Select a wizard Create a C/C++/Assembly project targeting Silicon Labs MCUs.		Select the board, part, and SDK for the project.	
Wizards:		Boards: EFM32 Giant Gecko Starter Kit board (BRD2200A Rev A03) × Part:	~
 > > C/C++ > > Simplicity Studio Show All Wizards. 		Search EFM32GG990F1024 SDK: Gecko SDK Suite: MCU 5.8.3.0, Micrium OS Kernel 5.7.0 (v2.6.3) (I:\Simplicity_studio\	v devel
	Cancel		Cancel
© BME-M	ІІТ	Méréstechnika és Információs Rendszerek	6.slide

3) Start a new empty project



3) Empty project created

Simplicity IDE - emptyCProject_2/src/main.c - Simplicity Studio ™

File Edit Source Refactor Navigate Search Project Run Window Help

* ▼ 🗣 ▼ 🛅 ▼ 🗒 🐚 | 🗞 ▼ 🗞 ▼ 🖢 ▼ 🖓 ▼ 🏷 → 🔶 ▼ 🕮

ြဲ Project Explorer 🛛	🖻 🔄 🏹 🗖 🗖 🚺 main.c 🛛
 emptyCProject_2 [GNU ARM v7.2.1 - De includes EMSIS 	bug][EFM32GG990F1024 - Gecko SDK Sui 1 #include "em_device.h" 2 #include "em_chip.h" 3 4@int main(void)
 > imain.c > imain.c 	5 { 6 /* Chip errata */ 7 CHIP_Init(); 8
	9 /* Infinite loop */ 10 while (1) { 11 }
	12 } 13

Comment out CHIP_Init(); function









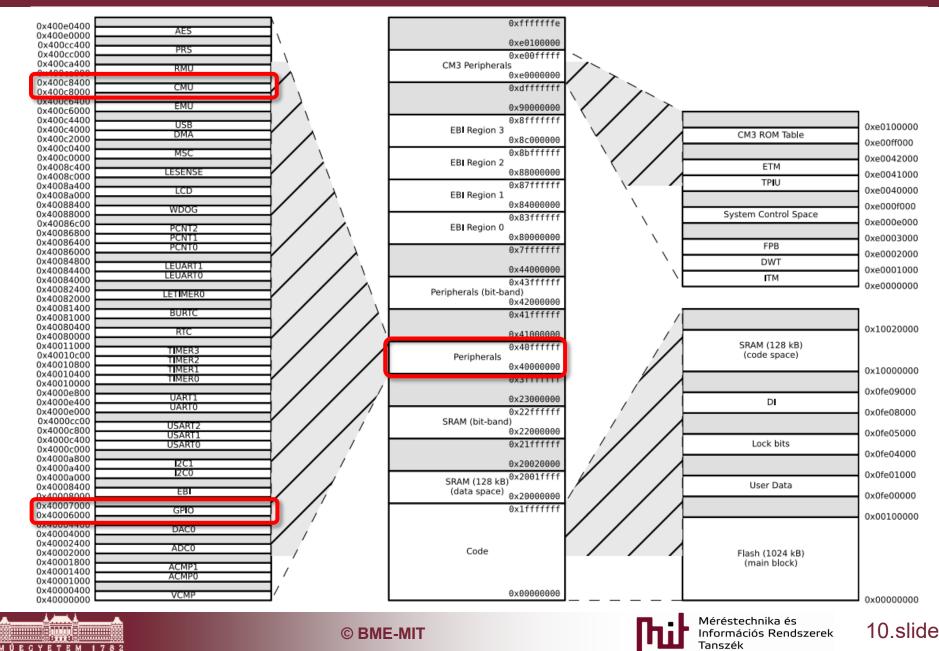
4) Get to know necessary peripherals

- 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)

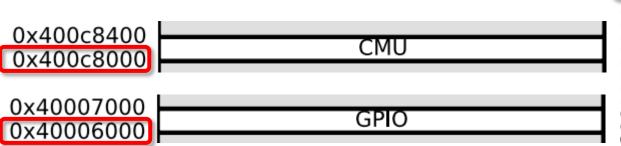




4) Memory map (full)



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

	-	
0x400c8400	CMU	
0x400c8000	CMU	
0x40000400	EMU	
0x400c6000	EMO	
0x400c4400	USB	
0x400c4000	DMA	
0x400c2000	DMA	
0x400c0400	MSC	
0x400c0000	MSC	
0x4008c400	LESENSE	
0x4008c000	LESENSE	
0x4008a400	LCD	
0x4008a000		
0x40088400	WDOG	
0x40088000	WDOG	ł
0x40086c00	PCNT2	
0x40086800	PCN12 PCN11	
0x40086400	PCNTI	
0x40086000	PCNIU	
0x40084800	LEUART1	
0x40084400	LEUARTO	
0x40084000	LEUARTO	
0x40082400	LETIMERO	
0x40082000	LETIMERO	ŕ
0x40081400	BURTC	
0x40081000	BORIC	
0x40080400	RTC	
0x40080000	Ric	/
0x40011000	TIMER3	
0x40010c00	TIMER2	
0x40010800	TIMER1	
0x40010400	TIMERO	
0x40010000	TIMERO	(
0x4000e800	UART1	
0x4000e400	UARTO	
0x4000e000	UARTU	
0x4000cc00	USART2	ł
0x4000c800	USARTI	
0x4000c400	USARTO	
0x4000c000	USANIU	
0x4000a800	12C1	
0x4000a400	201	
0x4000a000	200	
0x40008400	EBI	
0×40008000	EB	
0x40007000	GPIO	
0x40006000	GPIO	
	Méréstechnika és	
_ ha ; h	Információs Rendszerek	11.slide
	T	

Tanszék

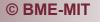


5) Base address aliases in code

 Avoid memorizing memory addresses using aliases in the code (use <u>Tab</u> instead of Space)

```
🖻 main.c 🖾
 1 #include "em device.h"
   #include "em chip.h"
  3
   #define CMU BASE ADDR
                           0x400c8000
    #define GPIO BASE ADDR 0x40006000
  6
 7⊖int main(void)
 8
      /* Chip errata */
 9
      //CHIP Init();
10
11
12
   /* Infinite loop */
13
    while (1) {
14
15 }
16
```









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
 - bbbbbbb |= 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
 - bbbbbbb &=~ 00100000 -> bbbbbbbb &= 11011111
 bb0bbbbb



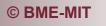


Check p.128 Fig.11.1 of EFM32GG-RM.pdf

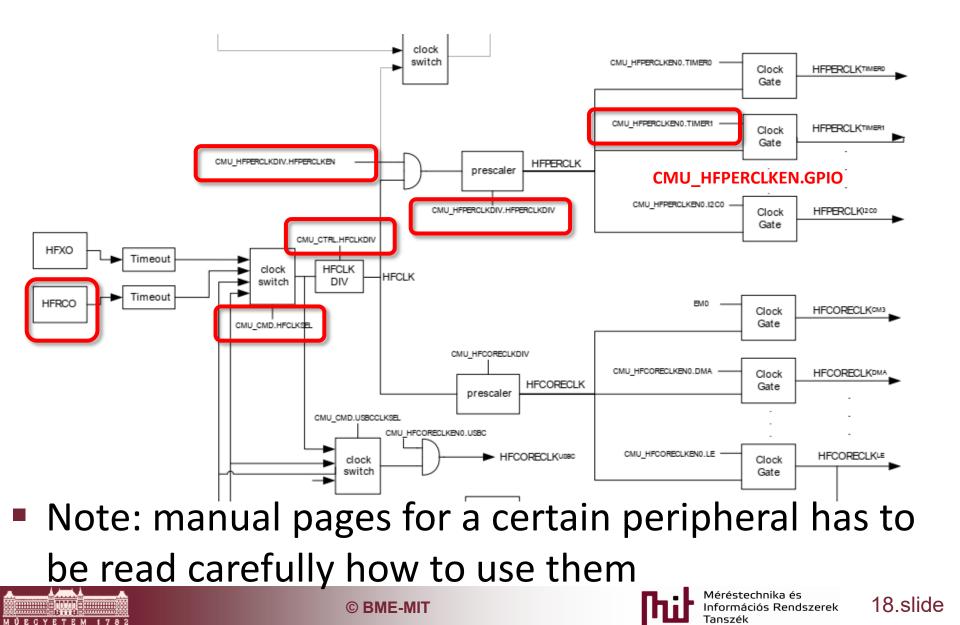
- Clock distribution network is shown
- Clock has to be provided for the peripherals
 - This is uC dependent but always has to be take care of providing CLK for the peripherals and enabling the peripherals
- Find HFRCO: high-frequency RC osc
 - Not too much precise but readily available -> no external CLK source is needed

Check the signal path toward the GPIO peripheral









- 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	Туре	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
	© BME-MIT	1	Méréstechnika és Információs Rendszerek 19.slid

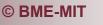
Use copy-paste to put the register addresses into the code

- 0x008 CMU_HFPERCLKDIV_OFFS
- 0x044 CMU_HFPERCLKEN0_OFFS

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

Offset	Name	Туре	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









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

© BME-MIT

11.5.1 CMU_CTRL - CMU Control Register

Offset															Bi	t Po	siti	on					•									
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	5	10	6	8	7	9	5	4	з	2	-	0
Reset		0		0				0×0			0×0			0x3	0		0×0		0	, c	nxn	ć	0X3		0	0x1			223	2		
Access		RW		RW				RW			RW		I	RW	RW		RW		RW				2 Y		RW	RW			/V/C			
Name		HFLE		DBGCLK				CLKOUTSEL1			CLKOUTSELO			LFXOTIMEOUT	LFXOBUFCUR		HFCLKDIV		LFXOBOOST				HFXUIIMEUUI		HFXOGLITCHDETEN	HFXOBUFCUR			HEYOROOST			
Bit	Na	me						Re	eset			A	cc	ess		De	scri	iptio	on													
22:20	CLK	(00	ΓSE	L0				0x0)			R	W			Clo	ock (Outp	out	Sel	ect ()										
	Con	trols	the	cloc	k ou	itpu	t mı	ultipl	exer	: To	actu	ally	ou	tput (on th	ne pi	n, se	et CL	KC	DUT	0PE	N in	СМ	J_R	DUT	E.						
	Valu	ue			M	ode									Descr	iptior	ı															
	0				H	FRC	0							F	IFRC	CO (d	irectl	y froi	m o	scilla	tor).											
	_																									érés						

21.slide

nációs Rendszerek



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

11.5.3 CMU_HFPERCLKDIV - High Frequency Peripheral Clock Division Register

Offset															Bit P	ositi	ion														
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	8	17 16	15	4	13	12	7	10	6	8	2	9	5	4	ю	~ ~	-	0
Reset																						_	-						0×0		
Access																							RW						RW		
Name																							HFPERCLKEN						HFPERCLKDIV		
Bit	Na	me						Re	set			A	cce	ss	D	escr	ipti	on													
31:9	Re	serve	ed					То	ensu	re co	omp	atibil	ity w	vith	future o	devic	es, a	alwa	ays v	vrite	e bits	to 0.	More	e inf	form	atio	n in S	Secti	on 2.1	(р.	3)
8	HF	PER	CLK	EN				1				R۷	V		н	PER	RCLI	ΚE	nabl	le											

Set to enable the HFPERCLK.





Check p.150 of EFM32GG-RM.pdf

Enable CLK for GPIO

11.5.18 CMU_HFPERCLKEN0 - High Frequency Peripheral Clock Enable Register 0

Offset							·								Bi	t Po	ositi	on														
0x044	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	5	10	6	8	7	9	5	4	ю	2	-	0
Reset															0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Access															RV	RV	RW	RV	RW	RW	RV	RV	RV	RV	RW	RW	RW	RW	RW	RW	RW	RV
Name															DACO	ADC0	PRS	VCMP	GPIO	12C1	12C0	ACMP1	ACMP0	TIMER3	TIMER2	TIMER1	TIMERO	UART1	UARTO	USART2	USART1	USARTO
Bit	Na	ame						Re	set			A	4cc	cess	•	De	scr	ipti	on													
13	G	iPIO						0				R	W			Ger	nera	l pu	rpos	se Ir	nput/	/Out	put (Cloc	k En	nabl	e	1				_
	S	et to	enat	ole th	ne cl	ock	for (GPIC).																							







- 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_HFPERCLKENO |= (1<<13);</p>
- Comment out all #include not to cause any trouble

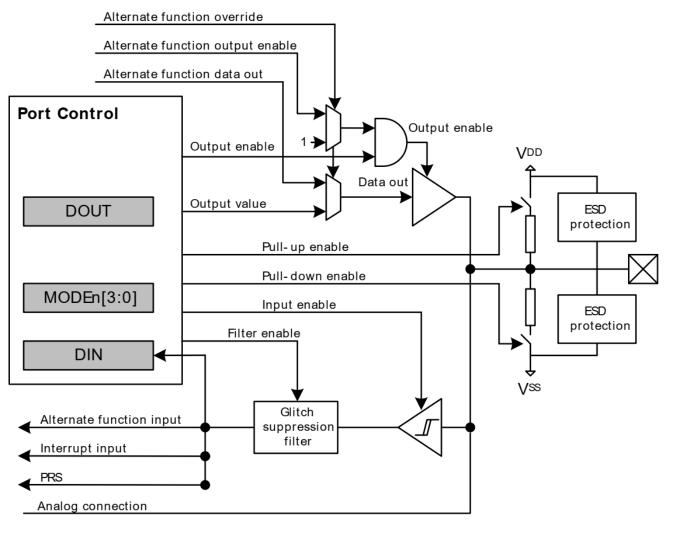
© BME-MIT

Check for errors by compiling



echnika és ociós Rendszerek 24.slide

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





© BME-MIT



Méréstechnika és Információs Rendszerek Tanszék

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

Offset	Name		Туре	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

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









- 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))
 - o #define GPIO_PB_DIN (*(volatile ...*)(...+.._OFFS))
 - o #define GPIO_PE_MODEL (*(...
 - o #define GPIO_PE_DOUT (*(...









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

Offset															Bi	t Po	siti	on														
0x000	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	4	13	12	11	10	ი	8	~	9	5	4	ю	7	-	0
Reset																															0	
Access																															Ma	
Name			-						-							-																

- Bit
 Name
 Reset
 Access
 Description

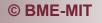
 31:2
 Reserved
 To ensure compatibility with future devices, always write bits to 0. More information in Section 2.1 (p. 3)

 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







Méréstechnika és Információs Rendszerek Tanszék



8) Peripheral handling in general- GPIO

Check pp. 766, the GPIO_Px_MODEL register

Offset															Bi	t Po	ositi	on														
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	7	10	ი	80	7	9	5	4	ю	2	-	0
Reset			nxn				nxn				nxn				0×0			0x0				020	000				nxn			020		
Access		1010				א צ								RW			RW				1010				1010				10/0			
Name		MODE7 R\ MODE6 R\									MOUES				MODE4			MODE3					MODE									

	Bit	Na	me	Reset		Access	Description	
	31:28	МО	DE7	0x0	I	RW	Pin 7 Mode	
		Cor	figure mode for pin 7.	Enumeration i	s equal	to MODE0.		
	alue		Mode		Descrip	tion 4bits->	16 different r	nodes
0			DISABLED		Input dis	sabled. Pullup i	f DOUT is set.	
1			INPUT		Input en	abled. Filter if	DOUT is set	
2			INPUTPULL		Input en	abled. DOUT o	determines pull dire	ection
3			INPUTPULLFILTER		Input en	abled with filte	r. DOUT determine	es pull direction
4			PUSHPULL		Push-pu	Ill output		
4			FUSHFULL		Pusn-pu			



© BME-MIT



stechnika és nációs Rendszerek

8) Peripheral handling push button- GPIO

Check pp. 767, the GPIO_Px_MODEH register

Offset															Bi	t Po	siti	on															
0x008	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	7	10	ი	ω	7	9	5	4	ю	2	-	0	
Reset	0×0 0×0														0×0	,		0×0									nxn		0×0				
Access	RX RX								RV						ЯŇ			RW							≷ Y		RW						
Name		313000	11				MODE 14			MODE13					MODE12			MODE11									MODES		MODE8				

11:8	MODE10	0x0	RW	Pin 10 Mode
	Configure mode fo	r pin 10. Enumeration i	s equal to MOD)E8.
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

O Use GPIO_PB_MODEH = (1<<4); //PB9 conf as input</p>

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



© BME-MIT



stechnika és mációs Rendszerek zék

Check pp. 767, the GPIO_Px_MODEL register

Offset															Bit	t Po	siti	on														
0x004	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	7	10	6	8	7	9	5	4	ю	2	-	0
Reset			nxn			ç	nxn		0×0					(0×0			0×0				020	nxn				0×0			0	nxn	
Access			NV N				Х Х		RŴ					i	RW			RW			RX						RW				Х Х	
Name			MUDE				MODE6			MODE5					MODE4			MODE3					MODEZ				MODE1				MODEO	

15:12MODE30x0RWPin 3 ModeConfigure mode for pin 3. Enumeration is equal to MODE0.11:8MODE20x0RWPin 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







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 |= ?
 O 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

O Use GPIO_PE_MODEL |= (4<<12); //PE3 conf as output</p>





- 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.
 - O GPIO_PE_DOUT|=(1<<2); //LED0 set</pre>
 - O GPIO_PE_DOUT|=(1<<3); //LED1 set</pre>







9) Operation at a code level

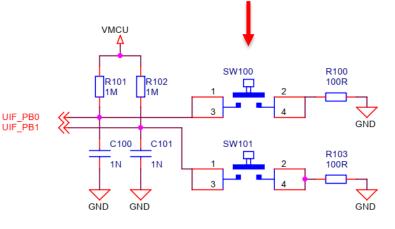
- What should be written in the while loop?
 - Read the status of the button (pushed/released) from the corresponding register bit and control the LED based on button state (on/off)

```
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;
    }
}</pre>
```

© BME-MIT

}

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	4	13	12	7	10	6	∞	~	- 4	2 10	4	e	2	- 0
Reset																															0×0
Access	Bit Name Reset Access Description																		RW												
	31:2 Reserved To ensure compatibility with future devices, always write bits to 0														0.			DE													
Name					1:0	:0 DRIVEMODE 0x0 RW Drive Mode Select Select drive mode for all pins on port configured with alternate drive strength.																	DRIVEMODE								
							V	alue			M	lode								Desc	riptio	n							_		
							0				S	TANI	DAR	D						6 mA	drive	curr	rent								
							1				L	OWE	ST							0.1 n	nA dri	ve cu	urrent								
							2				Н	IGH								20 m	A driv	e cu	rrent								
							3				L	OW								1 mA	drive	curr	rent								







11) Reference code

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

#define CMU_BASE_ADDR 0x400c8000 #define GPIO_BASE_ADDR 0x40006000

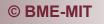
#define CMU_HFPERCLKDIV (*(volatile unsigned long int*)(0x400c8000 + 0x008))
#define CMU_HFPERCLKEN0 (*(volatile unsigned long int*)(0x400c8000 + 0x044))

#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</pre>
```







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 {</pre>
```

```
GPIO_PE_DOUT |= 1 << 2;
```



