Exercise 5: I/O in Assembly

Lab IoT

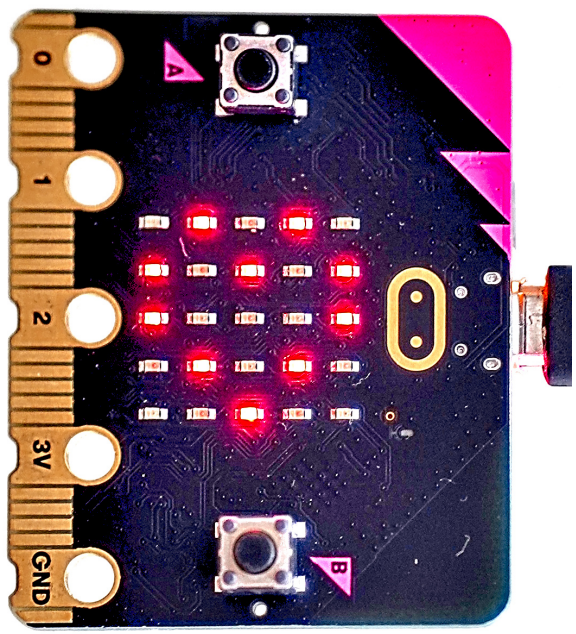
Philipp H. Kindt

Assistant Professorship for Pervasive Computing Systems (PCS)
TU Chemnitz

November 11, 2021

Goals

- The goal of this exercise is to operate the LED matrix using a) low-level C-code and b) assembly.
- We will therefore extend the code from Exercise 4.
- We will develop our code iteratively, starting from short, simple blocks of code to more involved assembly constructs.
- Please write your code in *main.c* of Exercise 4.



The GPIO on a Low Level

- Every peripheral is controlled by registers mapped in the memory space.
- The address of a peripheral register is given by a
  1. A base address that belongs to the corresponding peripheral (here, GPIO0).
  2. An offset for the particular register (e.g., the *OUTSET* register).

Base address	Peripheral	Instance	Description	Configuration	
0x50000000	GPIO	GPIO	General purpose input and output		Deprecated
0x50000000	GPIO	P0	General purpose input and output, port 0	P0.00 to P0.31 implemented	
0x50000300	GPIO	P1	General purpose input and output, port 1	P1.00 to P1.09 implemented	

Table 41: Instances

Register	Offset	Description
OUT	0x504	Write GPIO port

Figure: Address mapping of GPIO registers

Source: Nordic Semiconductor. nRF52833 Product Specification. v.1.4, 2021, `https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.4.pdf`

I/O so far...

We a have previously accessed the GPIO using the *nrf_gpio_pin_write()*-function. It is defined in nrf_gpio.h,by Nordic Semiconductors, line 675 ff:

```
__STATIC_INLINE void nrf_gpio_pin_write(uint32_t pin_number, uint32_t value)
{
        if (value == 0)
        {
                nrf_gpio_pin_clear(pin_number);
        }
        else
        {
                nrf_gpio_pin_set(pin_number);
        }
}
```

I/O so far (2)

Internally, *nrf_gpio_pin_clear(pin_number)* and *nrf_gpio_pin_set(pin_number)* will ...

- Determine the right GPIO (i.e., either GPIO0 or GPIO1) based on *pin_number*.
- Write to the corresponding GPIO register, which is either *OUTSET* for activating a pin or *OUTCLEAR* for deactivating a pin.

Structure of the OUTSET Register:

Bit number				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
ID				f	e	d	c	b	a	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
Reset 0x00000000				**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
ID	Acce	Field	Value ID	Value								Description																							
A-f	RW	PIN[i] (i=0..31)										Pin i																							
			Low	0								Read: pin driver is low																							
			High	1								Read: pin driver is high																							
			Set	1								Write: a '1' sets the pin high; a '0' has no effect																							

Figure: nRF52 OUTSET register

Source: Nordic Semiconductor. nRF52833 Product Specification. v.1.4, 2021, `https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.4.pdf`

Writing to the GPIO Registers

For activating a GPIO pin, the *OUTSET* register needs to be written:

```
NRF_P0->OUTSET = 1<<pin_number;
```

For deactivating a GPIO pin, the *OUTCLR* register needs to be written:

```
NRF_P0->OUTCLR = 1<<pin_number;
```

$\Rightarrow$ We can use the GPIO (and every other peripheral) without using Nordic's functions! The Nordic SDK is just there fore convenience.

Task 5.1 - GPIO on a Low Level

- Replace the calls to *nrf_gpio_pin_write()* for activating/ disactivating *the display's row signals* by direct register writes.
- Since all row signals are controlled by GPIO0, there is no need to distinguish between both GPIOs. All writes need to be directed to the registers of GPIO0.
- In Eclipse, go through the assembly code belonging to *NRF_P0->OUTCLR = 1«pin_number*
  step by step using the debugger in the *instr. stepping mode*.
- The assembly code will look up the *OUTCLR* member of the *NRF_P0* structure. At the end, there will be a store instruction, which represents the actual write to the GPIO register. To which address does the CPU write?
- Note: the instruction *LDR.w* is an ordinary load instruction. The ".w"-suffix ensures that a 32-bit instruction is generated. Its meaning is equal to that of *LDR*. You will also spot a *LDRB* instruction. It loads a byte instead of a 32-bit word from a memory location.

Task 5.2 - GPIO on the Lowest Level

- Look up the addresses of the *OUTSET* and *OUTCLR* registers of GPIO0 from the nRF52833 product specification.
- Replace the code *NRF_P0->OUTCLR = 1«pin_number* and *NRF_P0->OUTSET = 1«pin_number* by a direct write to this address.
- Towards this, create two pointers of type *uint32_t**. Assign the address of *OUTSET* (without using Nordic's macros) to one of them, and the address of *OUTCLR* to the other one.
- Test your code and step through it in the *instruction stepping mode*. Which assembly instructions does the compiler generate for this memory write?

Task 5.3: GPIO in Assembly

- We now access the GPIO in assembly language.
- Your main.c contains three function declarations:

```
void gpio_on(uint32_t pin);
void gpio_off(uint32_t pin);
void gpio_write_assembly(uint32_t pin, uint32_t value);
```

- These functions already exist as assembly code in *myGPIODriver.S*, which is already in your project folder.
- These functions currently only contain the code for returning to the caller.
- Goal of this task is to implement the functionality of these functions in assembly.

Task 5.3.1: GPIO gpio_on() and gpio_off()

- Implement *GPIO gpio_on()* (activate a GPIO pin) and *gpio_off()* (disactivate a GPIO pin).
- This function should have one parameter, which is pin-number to be switched on or off.
- Main idea: Put the GPIO register addresses you have looked up from the product specification into a CPU register. Then, write to these addresses for accessing the peripheral registers.
- You need to shift a 1 to the right position of the appropriate GPIO pin. You can use the *LSL* (logical shift left) for this.
- Replace *nrf_gpio_pin_write()* in main.c by a call to your newly written assembly function. Only do this for controlling the display rows. You can hence again assume that we will only write to GPIO0.

Task 5.3.2: gpio_write_assembly()

- Implement the *gpio_write_assembly()* function. It should have exactly the same interface as *nrf_gpio_pin_write(uint32_t pin_number, uint32_t value)*, and also implement the same functionality.
- An if...then...else construct is needed.
- For creating a first version, copy&paste parts of your code from Task 5.4.1 into your newly written function.
- Once you successfully tested your code, replace the code you pasted from Task 5.4.1 by a function call to *GPIO gpio_on()* or *gpio_off()*, respectively.
- Test your code by calling it from main.c instead of *nrf_gpio_pin_write()*.
- Only use it for writing to the display rows, such that all writes can be addressed to GPIO0.