UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TECHNISCHE INFORMATIK

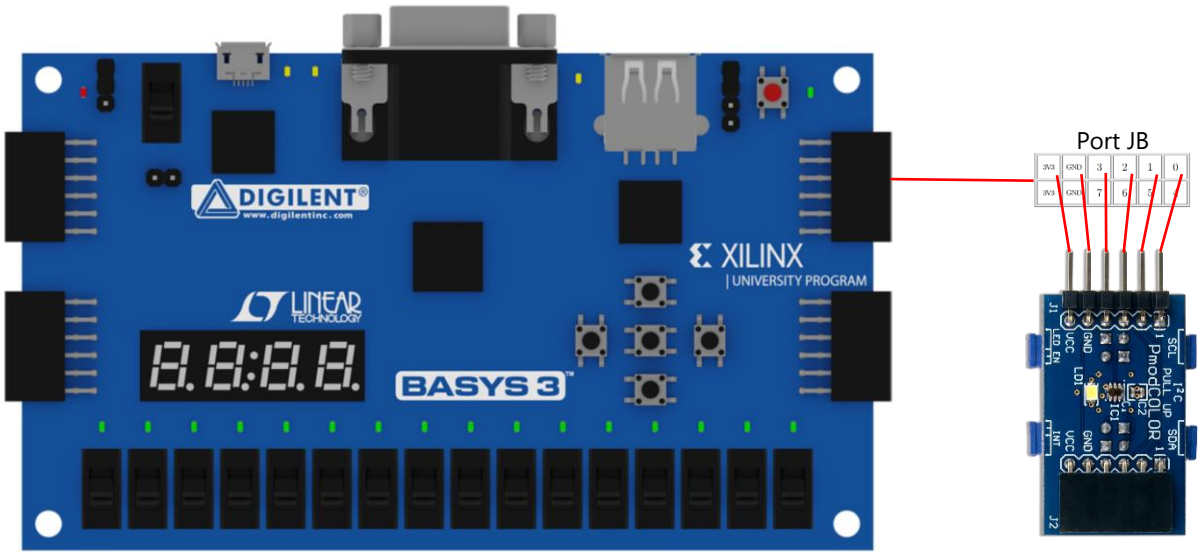
Practical Exercise 1

Implementation of Embedded Systems

Course: *Eingebettete Systeme – CS2101-KP04, CS2101*
Lecturer: *Prof. Dr.-Ing. Mladen Berekovic*
Exercises: *Waiel Aljnabi, Philipp Grothe, Christopher Blochwitz,*

Introduction

The target of this exercise is to create a light measuring station using the EduCore-V as processor and the TCS3472 as the light sensor[1] packaged as an Pmod COLOR module[2].

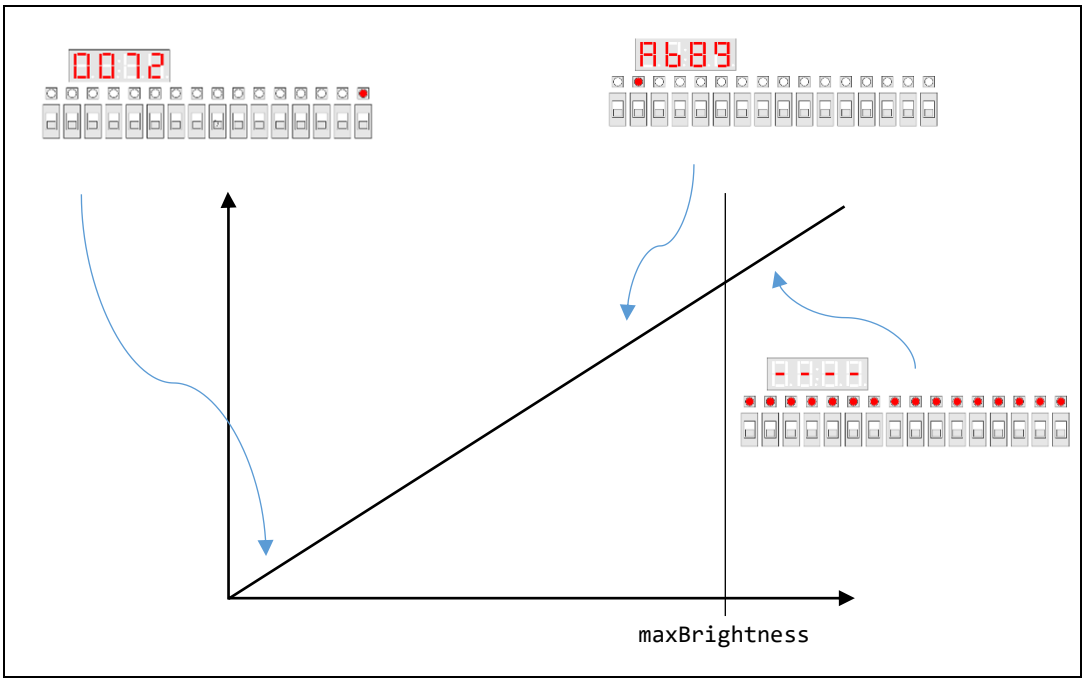


The main behaviour of the station could be described as followed: The processor requests the current brightness from the light sensor over I²C in an endless loop. If the current value is below a threshold `maxBrightness` display the current value on a seven-segment display as a hexadecimal number. Also turn on one of 16 LEDs to display how close the current brightness is to the threshold. If the current brightness is greater than the threshold `maxBrightness` the seven-segment display should change to `"----"` and every LED should turn on. The threshold `maxBrightness` can be set with 8 switches.

We will use the LED on the Pmod COLOR module to simulate various intensities. The intensity of the LED can be changed with another set of 8 switches.

The following figure illustrates the basic behaviour of the system:

[1] https://ams.com/documents/20143/36005/TCS3472_DS000390_3-00.pdf
[2] https://reference.digilentinc.com/reference/pmod/pmodcolor/start



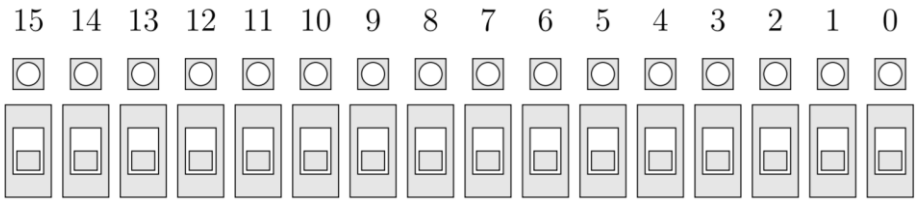
Desired behaviour of the implementation

EduCore-V: Documentation

The EduCore-V is a RISC-V softcore developed at the *Institut für Technische Informatik* and is implemented on the Basys 3 Evaluation Board[3]. The periphery of the Basys 3 Board is mapped to the address space of the EduCore-V and is available through special memory mapped registers.

LED & Switches

The registers **`mmio->led`** and **`mmio->sw`** are 16-bit registers, that are directly mapped to the LEDs and switches of the Basys 3 Board. While `mmio->led` is a read/write register, `mmio->sw` is a read-only register. Writing to this register will be ignored. The bits of these registers are mapped as shown in the figure below. A bit value of zero corresponds to a turned off LED/pulled down switch while a value of one corresponds to the opposite state. The signals of the switches are debounced before they are mapped into the memory space.



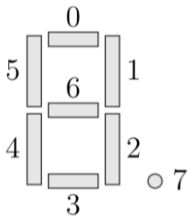
As an example, the following code would turn on the LEDs the corresponding switches of which are also in the on-state:

```
mmio->led = mmio->sw;
```

[3] https://reference.digilentinc.com/reference/programmable-logic/basys-3/start

Seven-Segment Display

The registers **`mmio->seg[0]`** to **`mmio->seg[3]`** are 8-bit read/write registers mapped to each digit of the seven-segment display from right to left. Each bit in these registers represents one segment of the digit as shown in the figure below. Setting the bit to one turns the corresponding segment on, setting it to zero turns the segment off.



The following Code displays the number three on the leftmost segment display:

```
mmio->seg[0] = 0b01001111;
```

I/O Ports

The ports **mmio->port_ja** to **mmio->port_jc** employ multiplexers to route different functions to the pins. Each pin can be used as either a digital I/O pin, a PWM output or a pin for I2C communication. The following code defines pin0 in mmio->port_jb as an PWM output:

```
mmio->port_jb.io_mux.pin0 = IO_MUX_PWM;
```

The next line defines it as the clock pin for the I^2C connection:

```
mmio->port_jb.io_mux.pin0 = IO_MUX_SCL;
```

The pin mapping of the port is shown in the figure below:

3V3	GND	3	2	1	0
3V3	GND	7	6	5	4

For the use of PWM or I^2C, the ports have dedicated controllers. These utilize additional controllers. For the PWM, this is mainly the register to set the duty cycle:

```
mmio->port_jb.pwm.duty_cycle = 0;
```

It defines the duty cycle of the PWM signal, where a value of 0 corresponds to a duty cycle of 0% (= 0/255) and a value of 255 a duty cycle of 99.6% (= 255/256).

For the I2C communication, multiple additional registers are available:

```
mmio->port_jb.i2c.ctrl // Control register
mmio->port_jb.i2c.rdata // Register for reading data
mmio->port_jb.i2c.wdata // Register for writing data
```

Additionally, in the control registers the bits have predefined functions. The following are relevant for this exercise:

```
I2C_CTRL_ENABLE // setting this bit starts a read or write operation
I2C_CTRL_BUSY // controller is busy (high) or ready (low)
I2C_CTRL_RCV_ACK // high if an ACK was received
/* The CMD bits are combined with the ENABLE bit */
I2C_CTRL_CMD_START // starts transaction
I2C_CTRL_CMD_STOP // stops transaction
I2C_CTRL_CMD_WRITE // writes to the bus
I2C_CTRL_CMD_READ_ACK // reads and sends ACK
I2C_CTRL_CMD_READ_NACK // reads and sends NACK
```

For example, the following line reads from the bus and then sends an ACK:

```
mmio->port_jb.i2c.ctrl = I2C_CTRL_CMD_READ_NACK | I2C_CTRL_ENABLE;
```

Compiling your Project

As mentioned before, the Basys Boards is not the MCU. We need to load a bitstream onto it to turn the FPGA into an MCU. For this, a USB stick is connected to the board containing the bitstream and a jumper next to the USB socket is set to the lowest position, indication that the board boots from USB. On boot, an LED left of the USB socket indicates if the bitstream is still being processed. This will be prepared in advance but if you have errors not caused by your implementation, check this setup for errors, too.

Everything you need for compiling your code and running it on the EduCore on the Basys Board is supplied with the template for the exercise. Additionally, a "hello world" project is supplied for you to test the workflow without relying on your implementation. For your implementation, do not change the folder structure and edit only the main.c file for the workflow to correctly function. But also check the other files for useful additional information.

To compile the C code, execute the build.bat script. A window will open showing if you code compiled correctly or if there were errors.

If your code compiled successfully, you can start the upload.bat. This will upload your program to the EduCore. Watch the window contents but also the display of the Basys Board. After the upload a serial connection will be kept open to the EduCore and in the Window the output of print statements in your code will be displayed. This I very useful for debugging.

If you close this window, your program will still be running. You can use the connect.bat script to connect to the serial terminal again to see the outputs as before.

If you get a refused connection error while uploading or connecting, it is easiest to unplug and replug the Basys boards from/to the PC. Remember to wait for the bitstream to be ready after this.

Implementation

These steps will guide you through the implementation. You don't have to write the code into the boxes below. This is only necessary if you want to request written comments on your code.

Task 1: Port Setup

The Pmod COLOR module is connected to the top row of port JB. Based on the I/O Port definition and the datasheet of the module configure the PWM-Controller and the I^2C-Controller to use the correct pins. Make sure to also set the PWM output to 0 and top any I^2C transactions that could still be open.

```
void port_setup(void)
{









}
```

Task 2: I^2C helper methods

To simplify the I^2C transactions we implement the helper functions `i2c_start` and `i2c_stop` to initiate and terminate a transaction and the two functions `i2c_read` and `i2c_write` to read from/write to the bus.

For the `i2c_start` function, set the respective command bit and start an operation. Then wait for the controller to finish.

```
void i2c_start(void)
{









}
```

For the `i2c_stop` function, set the respective command bit and start an operation. Then wait for the controller to finish.

```
void i2c_stop(void)
{









}
```

The read function only reads from the bus. This is not a full read transaction specific device or register addresses. The Boolean **ack** is passed to the function to decide whether an ACK (ack==true) or a NACK (ack==false) is sent after reading. Read and send the (N)ACK, wait for the operation to finish and return the value that was read.

```
uint8_t i2c_read(bool ack)
{









}
```

Similarly, the write functions writes the contents of **value** to the bus without a specific target. For this, the contents must first be written into the designated register. After that, the operation can be started and waited for its completion. A Boolean must be returned, indicating whether an ACK was received.

```
bool i2c_write(uint8_t value)
{









}
```

Task 3: Sensor Communication

The file `'tcs34725.h'` declares the device address and the register addresses of the light sensor and their values and flags. Using this file and the documentation of the TCS34725, familiarise yourself with the registers ID, ATIME, CONTROL, ENABLE, CDATAL and CDATAH.

The documentation of the TCS34725 states in page 21 that the address register (Command Register) holds additional information (CMD, TYPE). When transmitting the address of the targeted register set these bitfields according to the documentation.

To simplify the communication with the sensor, implement the wrapper functions. Use the device address specified in the file `'tcs34725.h'` and modify the register address according to the documentation.

The device address can be hard-coded into the function but the register is passed along with the value to be written. Don't forget to start and stop the transactions.

```
void pmod_write(uint8_t reg_address, uint8_t value) {









}
```

```
uint8_t pmod_read(uint8_t reg_address) {









}
```

Next, we must implement the initialization routine of the sensor. First call the previously implemented `port_setup()` function. Then read the ID register of the sensor and print it to the console using `printf`. Also check if the ID equals `0x44` and exit the function returning `false` if that is not the case. Then set the integration time and gain (ATIME, CONTROL). Use the lowest settings first and experiment with different values later. Then set the PON flag in the ENABLE register to power on the sensor and wait 3ms using the `_delay_ms` function. Then also set the AEN flag in the ENABLE register to enable the Analog-Digital-Converter inside the

sensor and again wait 3ms. Use the previously implemented `pmod_read()` and `pmod_write()` functions and the constants in the `'tcs34725.h'` file to communicate with the sensor:

```
bool pmod_init(void)
{









}
```

Call the `pmod_init()` function from your `main()` and check if everything was implemented correctly.

Task 4: Implementing the main loop

- Read the state of the upper 8 switches and set the brightness of the LED on the Pmod COLOR module.
- Read the state of the lower 8 switches and calculate the `maxBrightness` by scaling the value by 256.
- Read in the clear channel data low and high byte of the sensor and combine both bytes to a 16-bit integer.
- Display the 16-bit value on the seven-segment display. For this use the given lookup table.
- Map the sensor value from the range `[0, maxBrightness]` to `[0, 15]` and turn the corresponding LED on. If the sensor value is greater than `maxBrightness` turn on every LED and display `"----"` on the seven-segment display.

```c
int main(void)
{
    // initialize the sensor
    pmod_init();

    while (1)
    {
        // set the switch state as the duty cycle of the pwm pin


        // set the max brightness depending on the switch states
        uint16_t maxBrightness =

        // read in the values from the sensor
        uint8_t value_l =
        uint8_t value_h =

        // calculate the combined sensor value
        uint16_t value =

        // print the value to the 7-seg display
        uint8_t digits[] = {
            0b00111111, 0b00000110, 0b01011011, 0b01001111,
            0b01100110, 0b01101101, 0b01111101, 0b00000111,
            0b01111111, 0b01101111, 0b01110111, 0b01111100,
            0b00111001, 0b01011110, 0b01111001, 0b01110001
        };




        // map value from [0, maxBrightness] to [0, 15] ...
        uint8_t mappedValue =
        // ... and turn on the corresponding LED


        // check if the read value is greater that maxBrightness
        // and change the LEDs and 7-seg display accordingly









        // wait a little bit before reading in the next values
        _delay_ms(3);
    }
}
```