

Alien shooter exercise on an LED matrix

Toni Jääskeläinen

COMP.CE.100 Introduction to Embedded Systems

Exercise main goals

- Learn the basics of embedded C-programming and how the LED matrix works
- Learn how to read information from datasheets or from other documentation
 - How to utilize this information in code
- Learn how to use the Xilinx SDK and its features (debugger, software loading to board etc..)

General info

- Updated exercise files will be added on Moodle.
- Check MapV2.pdf to find TC219 (if you don't know where it is)
 - You will get access to rights to TC219 (need to read NDA and sign it somehow)
- Can also do remotely at home
 - Loan board and install Xilinx SDK
- Find yourself an **exercise pair/pairs** (or do solo)
 - Three is max group size
 - Use Moodle group tool to form groups!
- Deadline is on 22.4.2022!
 - Source codes must be returned by end of that day
 - Show to assistant or return project to Moodle

Guidance times

- Guidance times available sometime after exam week and last until deadline
 - Local guidance at TC219 and remotely at Zoom and Mattermost
 - Invite link to Mattermost:
https://mattermost.tut.fi/signup_user_complete/?id=yf3x7e9xo3f4jy5um9pyr9a35y
 - Student channel:
<https://mattermost.tut.fi/studentsetstaff/channels/compce100-introduction-to-embedded-systems-students>
- If there is any change on the times, we will give out a notification.

Borrowing PYNQ

- You need to borrow PYNQ with LED matrix to complete this project work, if doing remotely
 - You need your own Micro-USB cable
 - Borrowing times available on Moodle

Installing the software

- Download Xilinx SDK 2019.1
 - Quick install guide and link will become available on Moodle
 - ***If installer asks to install drivers, just do it***
- Windows or Linux is required with ~40 GB of free space
 - Size is large but SW can be used in other courses, like Digital Design

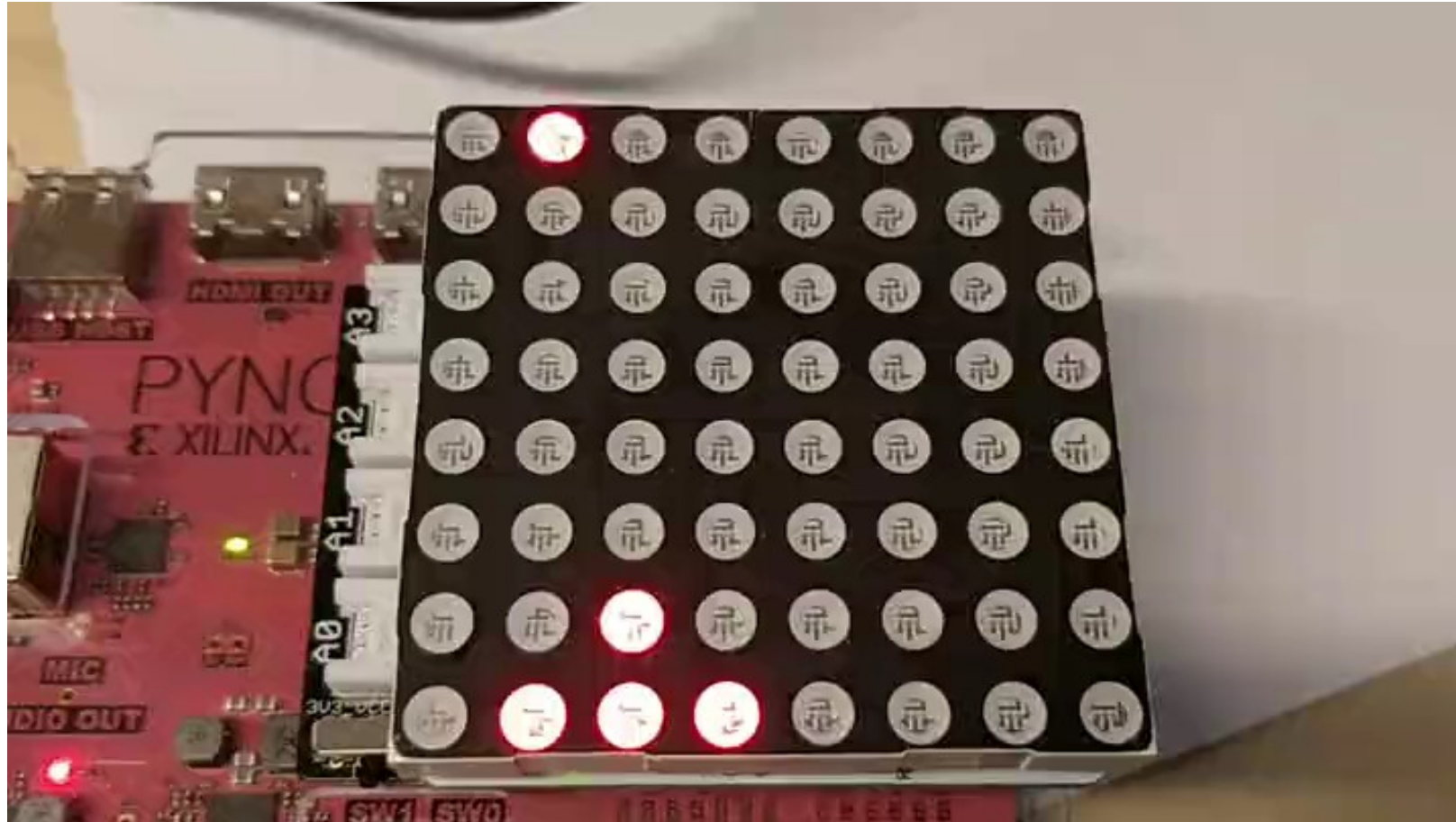
Exercise pass requirements, 5 p

- LED matrix is working
 - LED matrix LEDs can be set active any amount and anywhere with any color.

Exercise pass requirements, 10 p

- Code LED matrix to functional state and use that code to code the game
- "ship" can be moved using buttons
- Alien is moving
- Ship can fire moving bullets
- Hitting alien increases score or/and missing increases miss score
- Game has ending
- Score needs to been shown somewhere
- Game can be restarted
- Coding style is free of choice

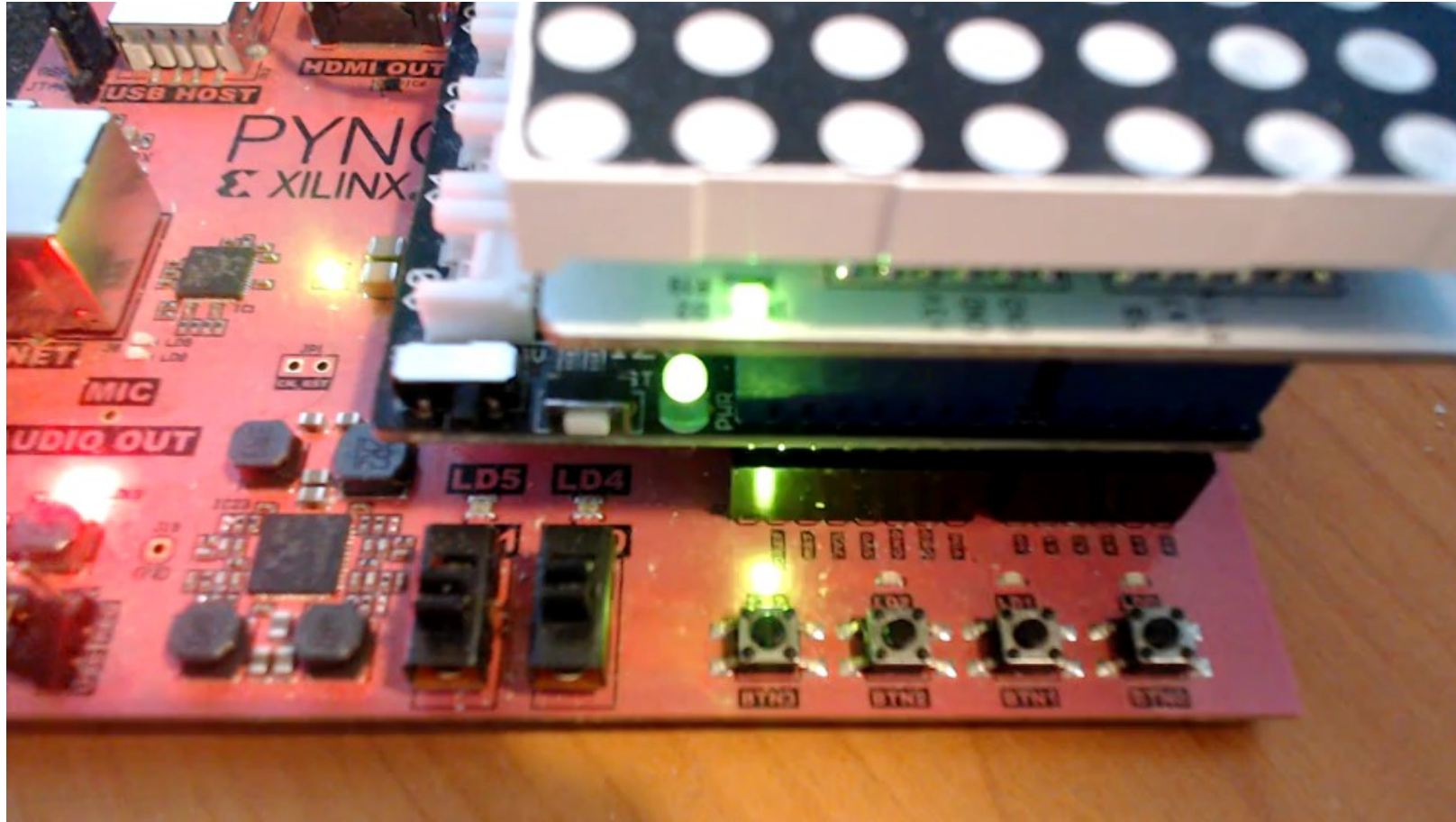
Example game screen



”extra”, 2 p

- Make button-LEDs (normal LEDs near buttons: LD0-LD3) go left->right->left unlimited times
 - LED shifting from left to right and back must be visible with human eyes!
- This needs to be written with assembler
 - Write code into given ASM file (blinker.S)
- This 2 points can be summed to both 10 and 5 points
 - Meaning total points would be 12 or 7

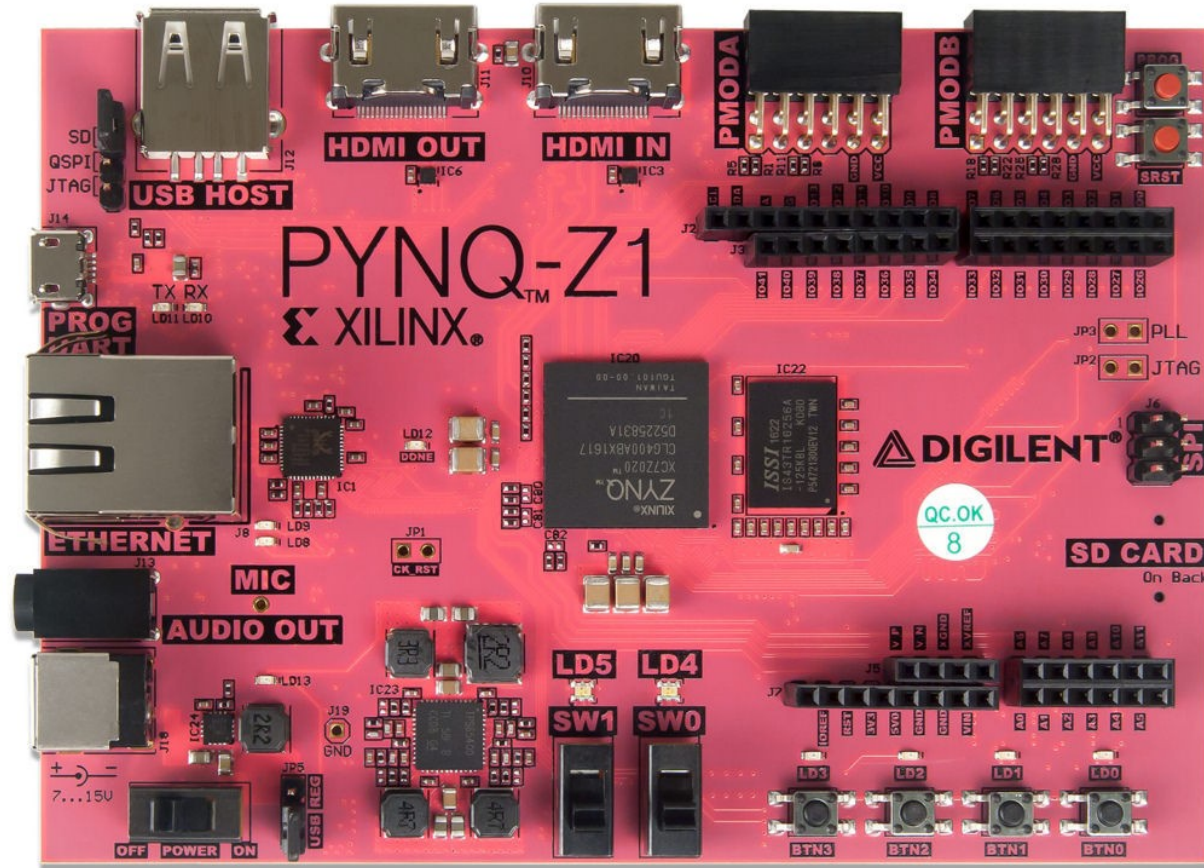
Button-LED blinker example



Points summary

- 5 points to get LED matrix working
- 5 points to get game working
- 2 points to get button-LED shifter working using assembler

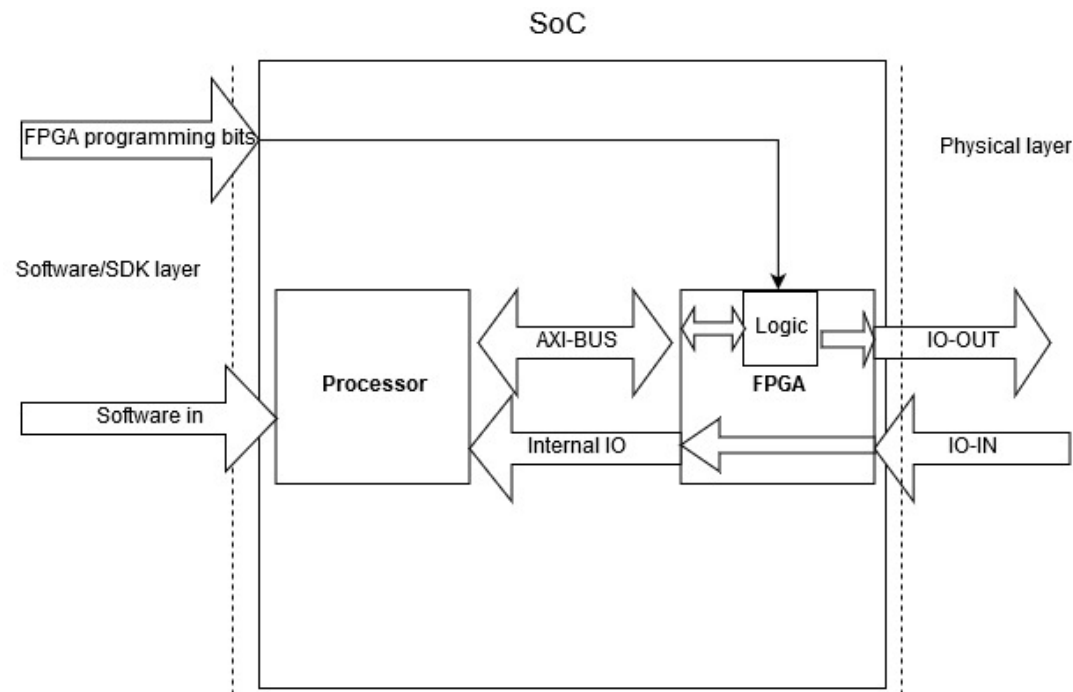
PYNQ-Z1 Development board



Features

- Zynq-7000 SoC chip
 - ARM Cortex-A9 Processor (Dual core)
 - Artix-7 family FPGA
- Arduino Compatibility: can use Arduino shields (like LED matrix in this exercise)
 - Does not support Arduino codes
- Supports OS and **baremetal programming**.

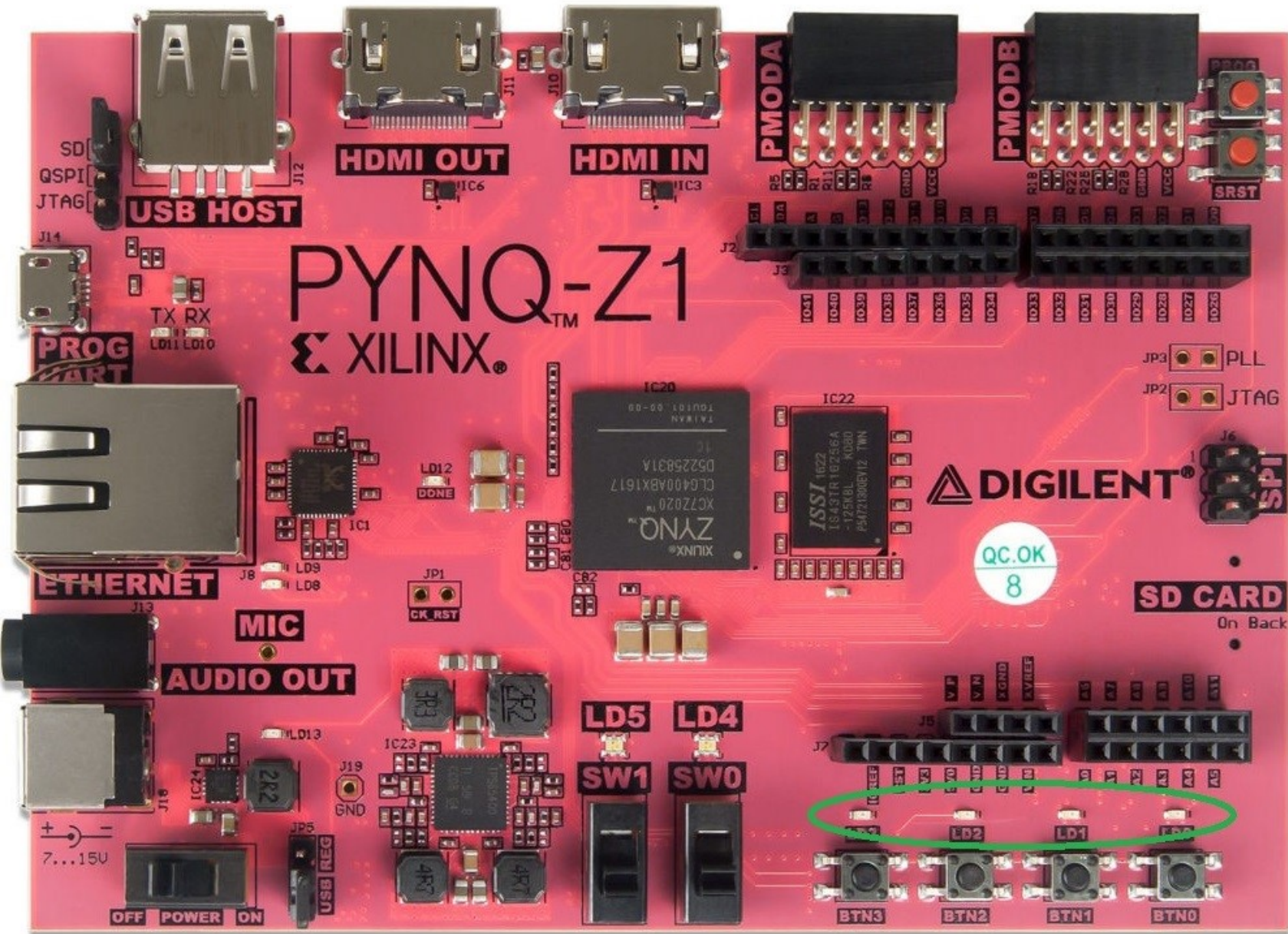
Simplified block diagram for PYNQ



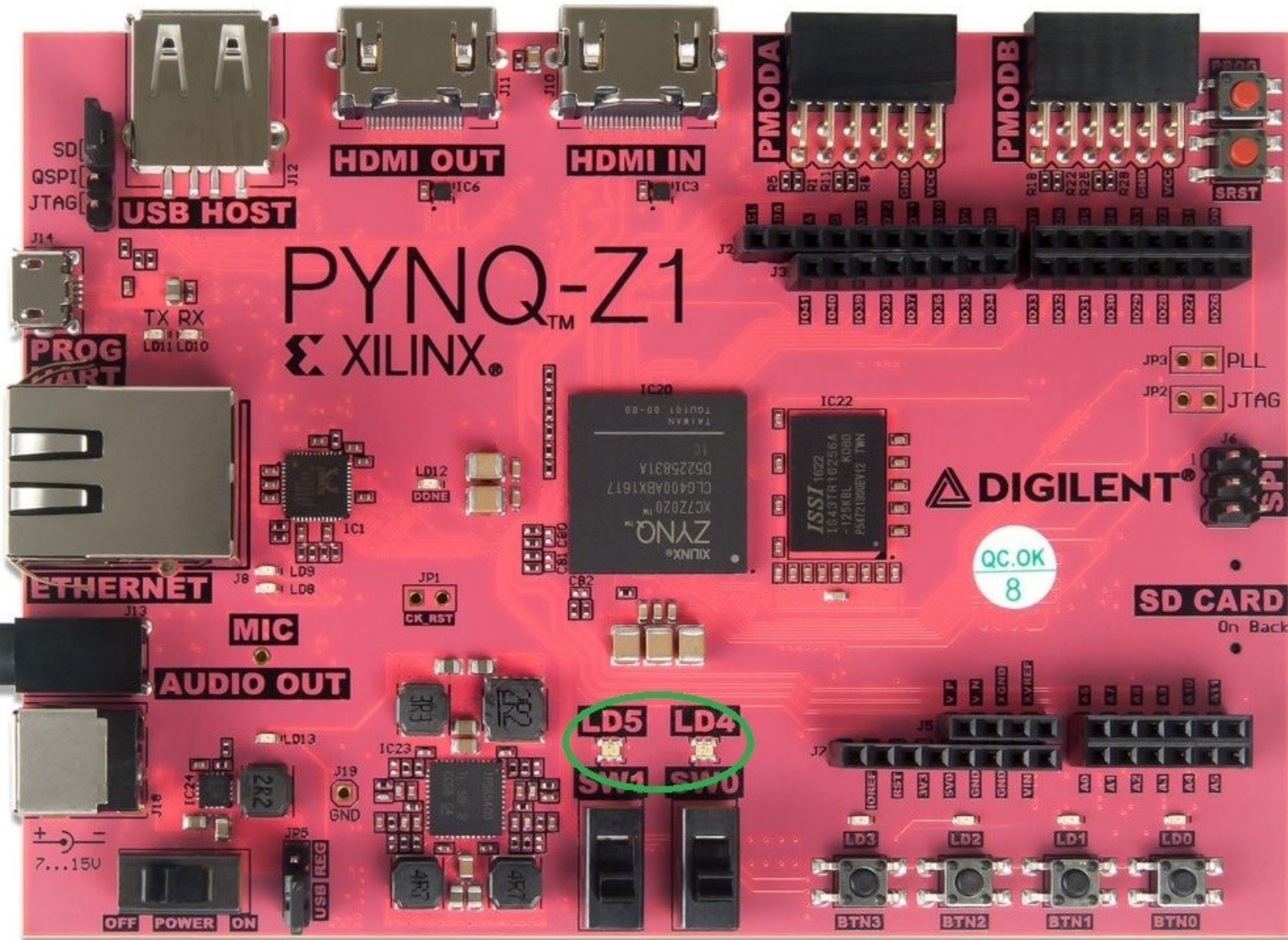
- FPGA is only used to make interconnection between physical I/O's and processor
- In this course, we only program the processor.

Channel							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
C7	C6	C5	C4	C3	C2	C1	C0
<u>Memory Address=0x41220000</u>							
Control_signals							
Bit4	Bit3	Bit2	Bit1	Bit0			
SDA	SCK	SB	LAT	RSTn			
<u>Memory Address=0x41220008</u>							
Inputs							
Bit5	Bit4	Bit3	Bit2	Bit1	Bit0		
SW1	SW0	BTN3	BTN2	BTN1	BTN0		
<u>Memory Address=0xE000A068</u>							
Leds							
Bit3	Bit2	Bit1	Bit0				
LD3	LD2	LD1	LD0				
<u>Memory Address=0x41200000</u>							
RGB leds							
Bit5	Bit4	Bit3	Bit2	Bit1	Bit0		
LD5.R	LD5.G	LD5.B	LD4.R	LD4.G	LD4.B		
<u>Memory Address=0x41240000</u>							

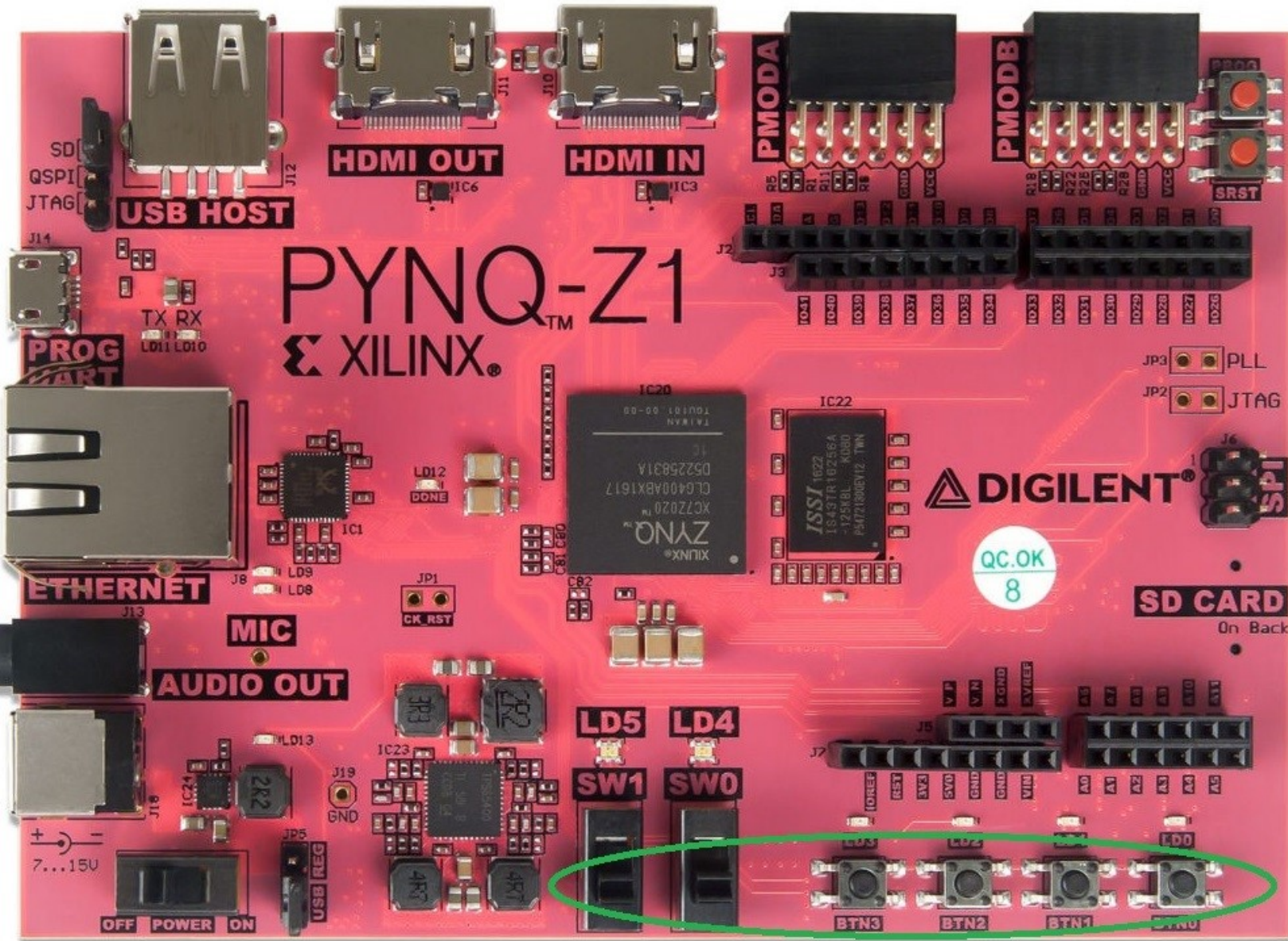
NOTE that reset is active when low (n suffix)!



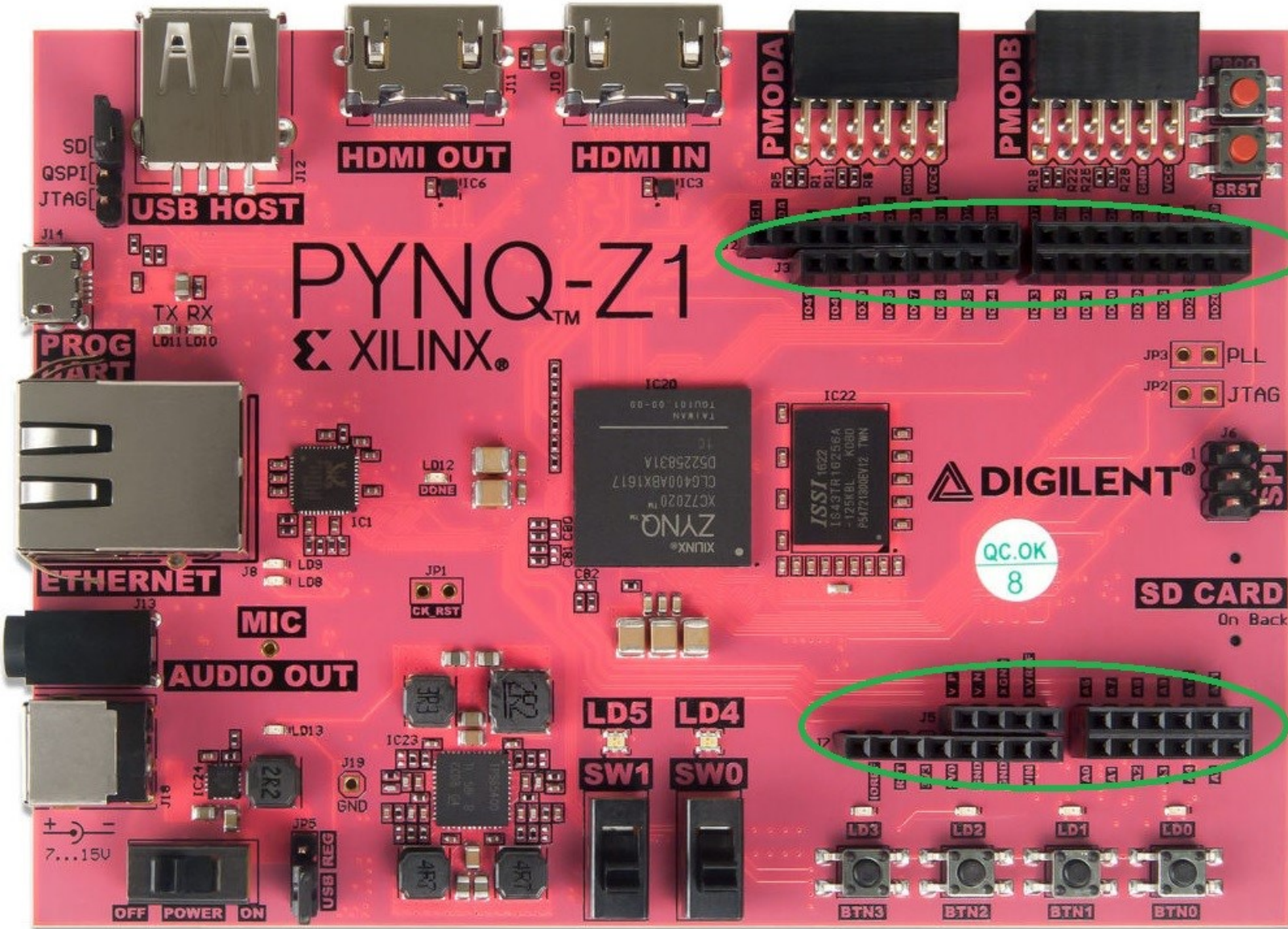
Leds
Memory
address:0x41200000



RGB leds
Memory
Address:0x41240000



Inputs
Memory
address:0xE000A068



Channel
Memory
address:0x41220000

Control_signals
Memory
address:0x41220008

How to use memory addresses in C?

Example: setting LED **LD0** on

```
uint8_t *led = 0x41200000;  
*led |= 0x01;
```

Example: read input

```
uint8_t *btns = 0xE000A068;  
uint8_t a = 0;  
a = *btns;
```

How to use memory addresses in C?

- Led on:

```
*(uint8_t *) 0x41200000 /= 0x01;
```

- Read button state to variable:

```
uint8_t btn = *(uint8_t *) 0xE000A068;
```

How to use memory addresses in C?

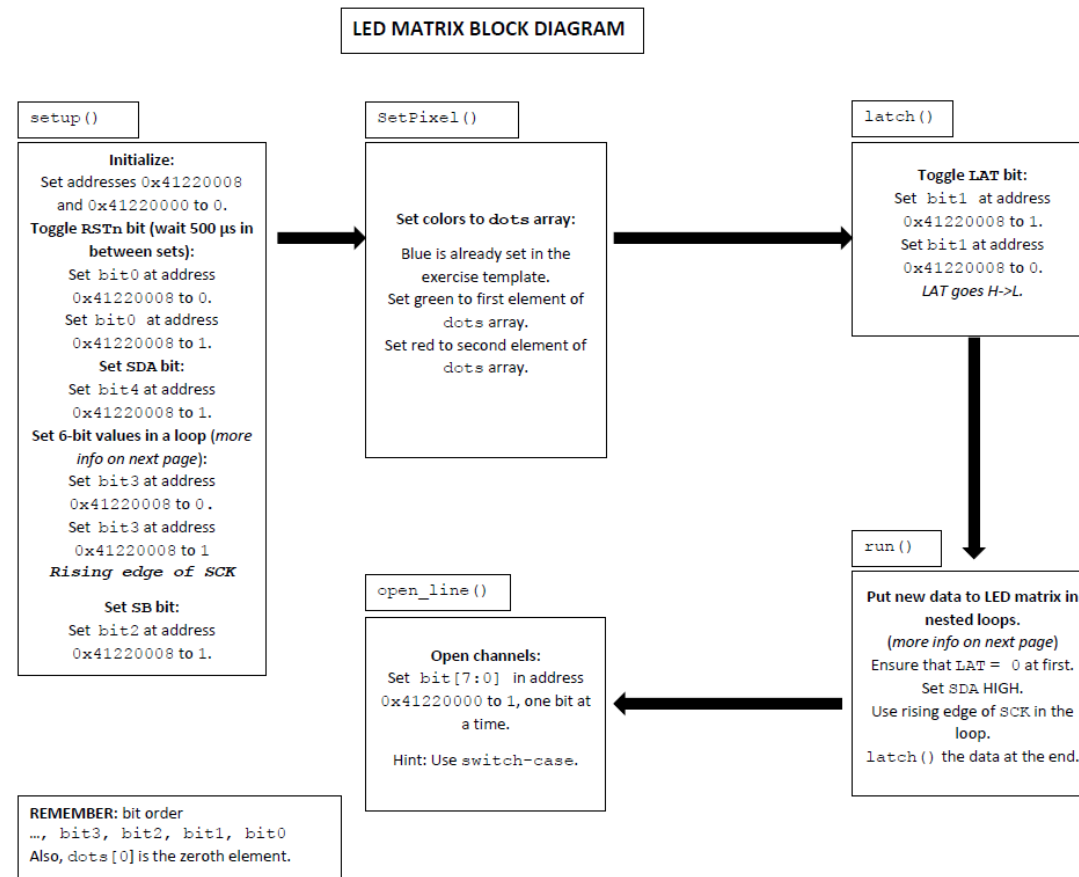
- One good way could be also to define address with some name
- Example, define in global:

```
#define led *(uint8_t *) 0x41200000
```

- Now you can use led to modify data at that address, following will turn LED **LD3** on:

```
led |= 0x08;
```

Block diagram of LED matrix functions



Interrupt handlers

INTERRUPT HANDLERS

BUTTON/SWITCH INTERRUPT

`ButtonHandler ()`

Check Status variable for
button presses and switch
toggles:

Buttons are `bit[3:0]`.

Switches are `bit[5:4]`.

Hint: Use if-statement.

REMEMBER: bit order
`..., bit3, bit2, bit1, bit0`

TIMER INTERRUPTS

`TickHandler ()`

Refresh the LED matrix screen:

The channels should be turned
ON individually.

Check if channels are inside
the screen.

Close all channels.

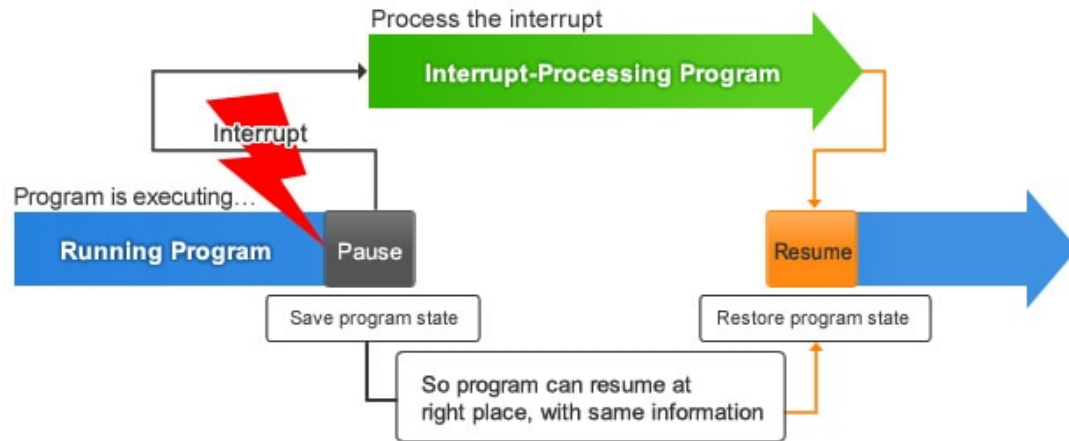
Call `run ()` and
`open_line ()` on channel.
Increment channel.

`TickHandler1 ()`

Handle alien and shooting:

Call functions for alien
movement and shooting here,
according to your game logic.

Interrupts



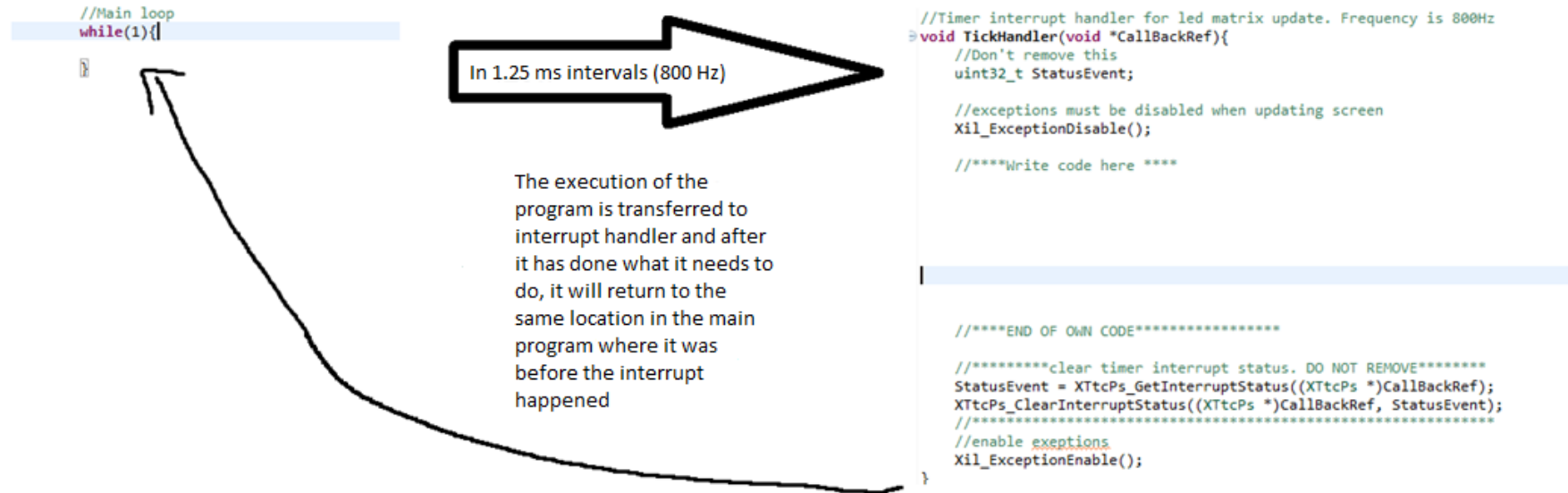
Source: <https://www.renesas.com/eu/en/support/technical-resources/engineer-school/mcu-programming-peripherals-04-interrupts.html>

- Interrupts cause an interrupt in the main program and jump to run the interrupt subroutine.
- In exercise, there are two interrupt methods
 - Timer and GPIO edge interrupt

Timer interrupts

- **Very simply:** Timer interrupts are interrupts that are happening within certain intervals
 - In this exercise, timer interrupts are happening with 10 Hz and 800 Hz frequency.

Timer interrupts



GPIO Edge interrupt

- Interrupt happens on voltage level change of I/O. Usually on low to high or high to low change -> on rising or falling edge.
 - In this exercise, buttons cause an interrupt on falling edge and switches on both edges

GPIO Edge interrupt

```
//Main loop  
while(1){
```

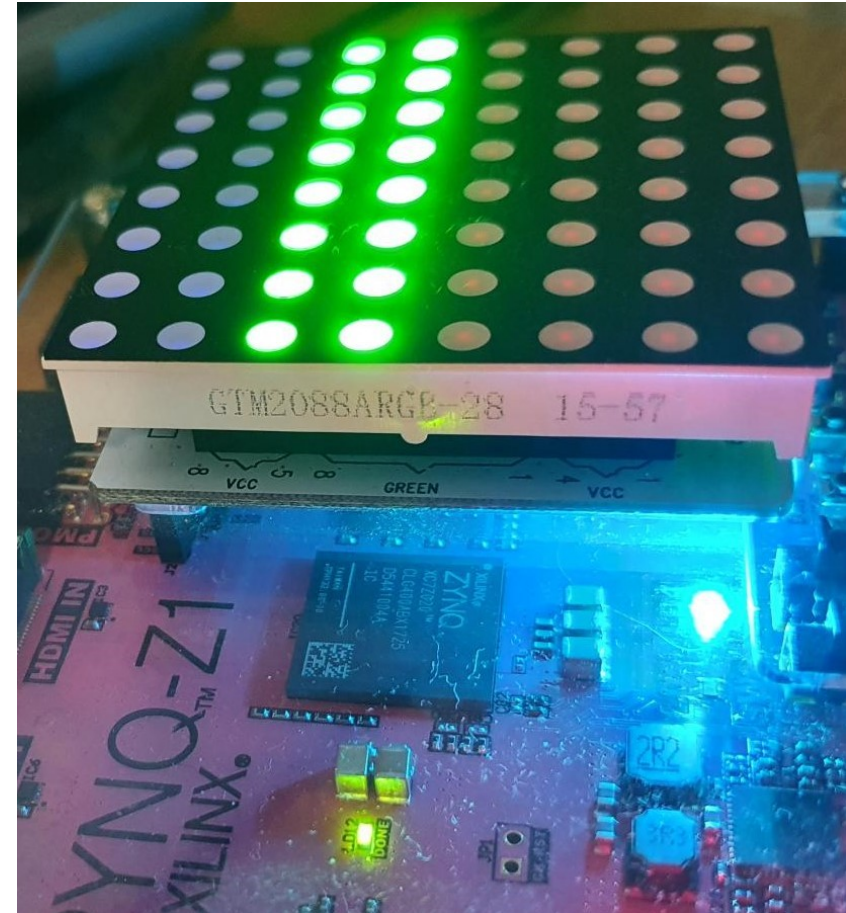
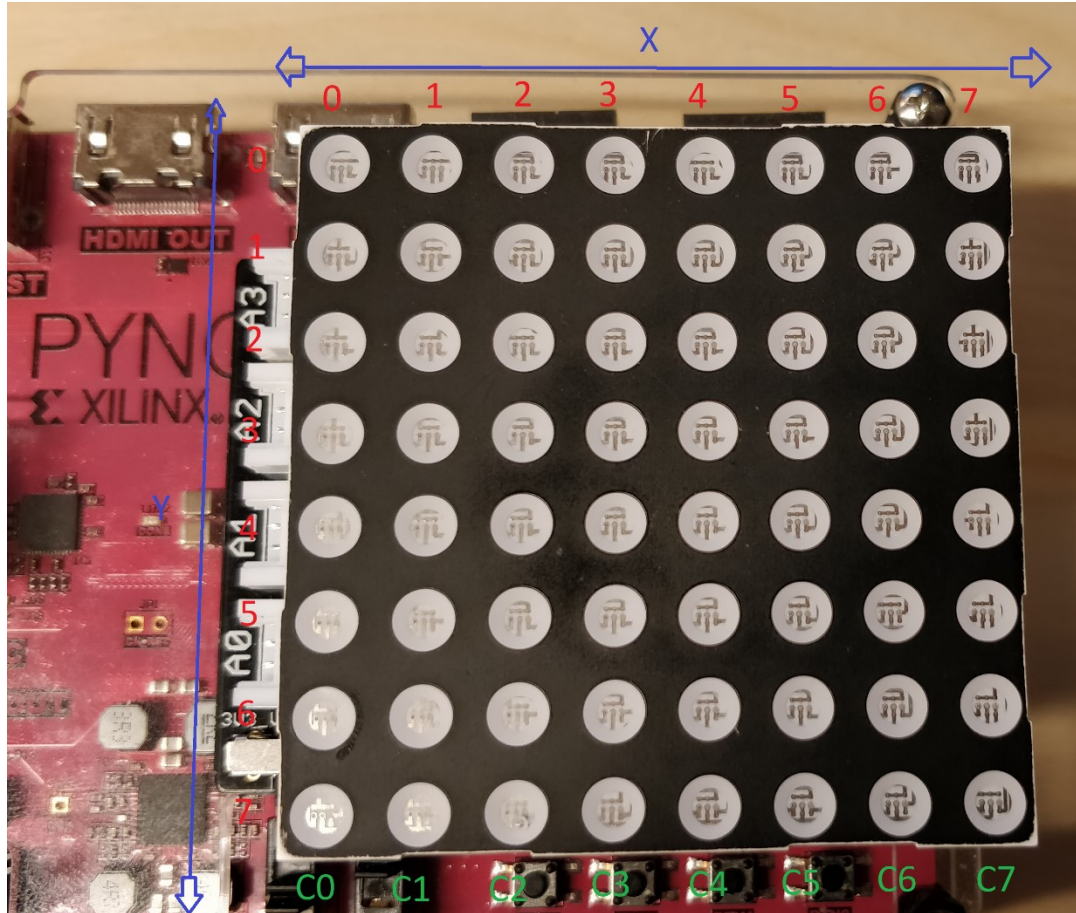
A button is pressed

The execution of the program is transferred to interrupt handler and after it has done what it needs to do, it will return to the same location in the main program where it was before the interrupt happened

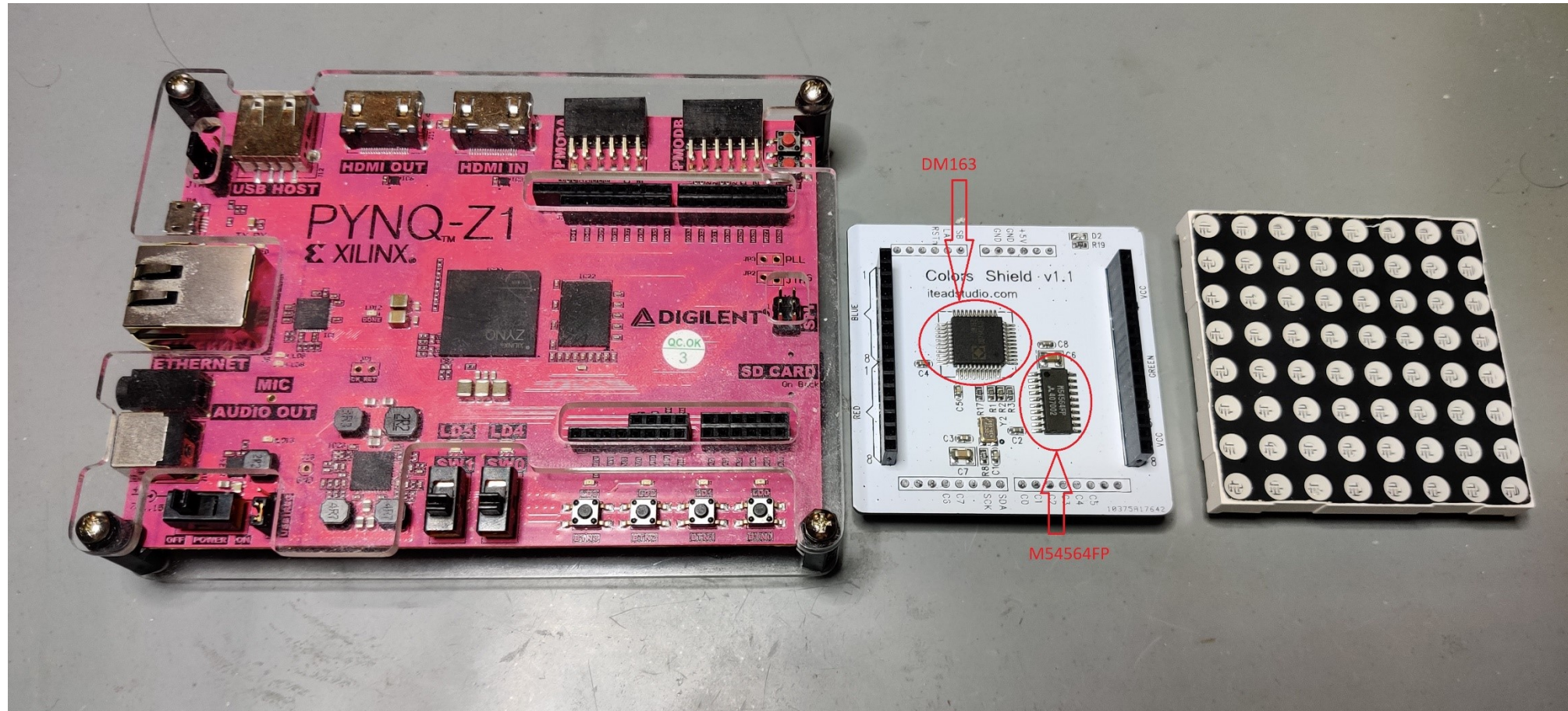
```
//Interrupt handler for switches and buttons. Connected buttons and switches are at bank2. Reading Status will tell which button or switch was used  
void ButtonHandler(void *CallBackRef, u32 Bank, u32 Status)  
{  
    //****Write code here ****  
  
    //BTN0 is clicked  
    if(Status==0x01) {  
  
    }  
  
    //BTN1 is clicked  
    else if(Status==0x02){  
  
    }  
    //BTN2 is clicked  
    else if(Status==0x04){  
  
    }  
    //BTN3 is clicked  
    else if(Status==0x08){  
  
    }  
  
    //SW0 position is changed. 0xE000A068 address needs to be read if you want to know in which position slider is  
    else if(Status==0x10){  
  
    }  
  
    //SW1 position is changed. 0xE000A068 address needs to be read if you want to know in which position slider is  
    else if (Status==0x20){  
  
    }  
  
    //****END OF OWN CODE*****  
}
```

Led matrix

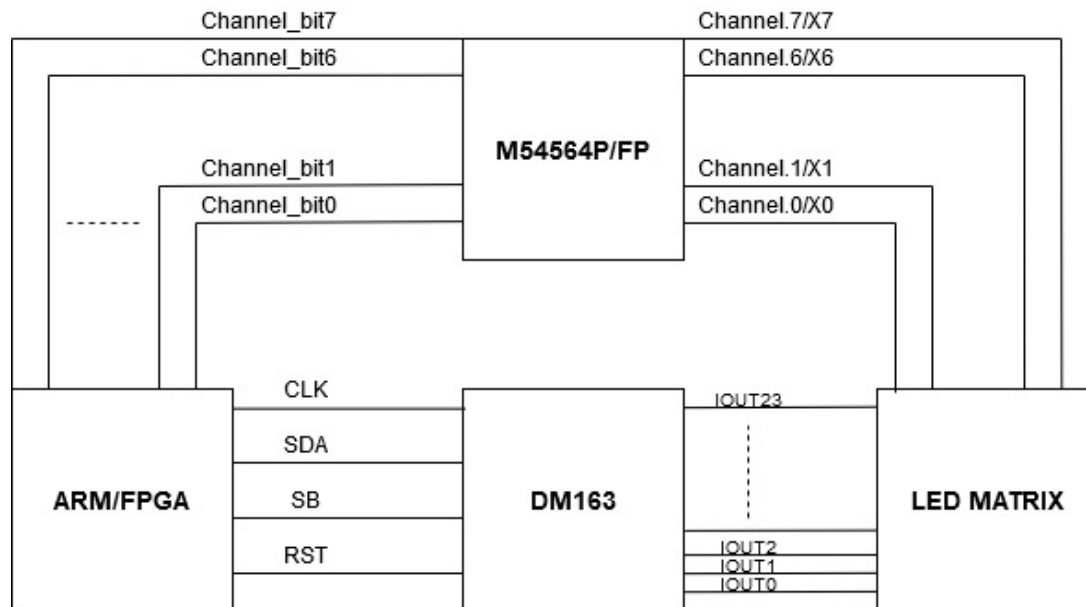
- If you get some random pixels when running the project, turn the matrix 180 degrees and try again!



PYNQ LED matrix shield

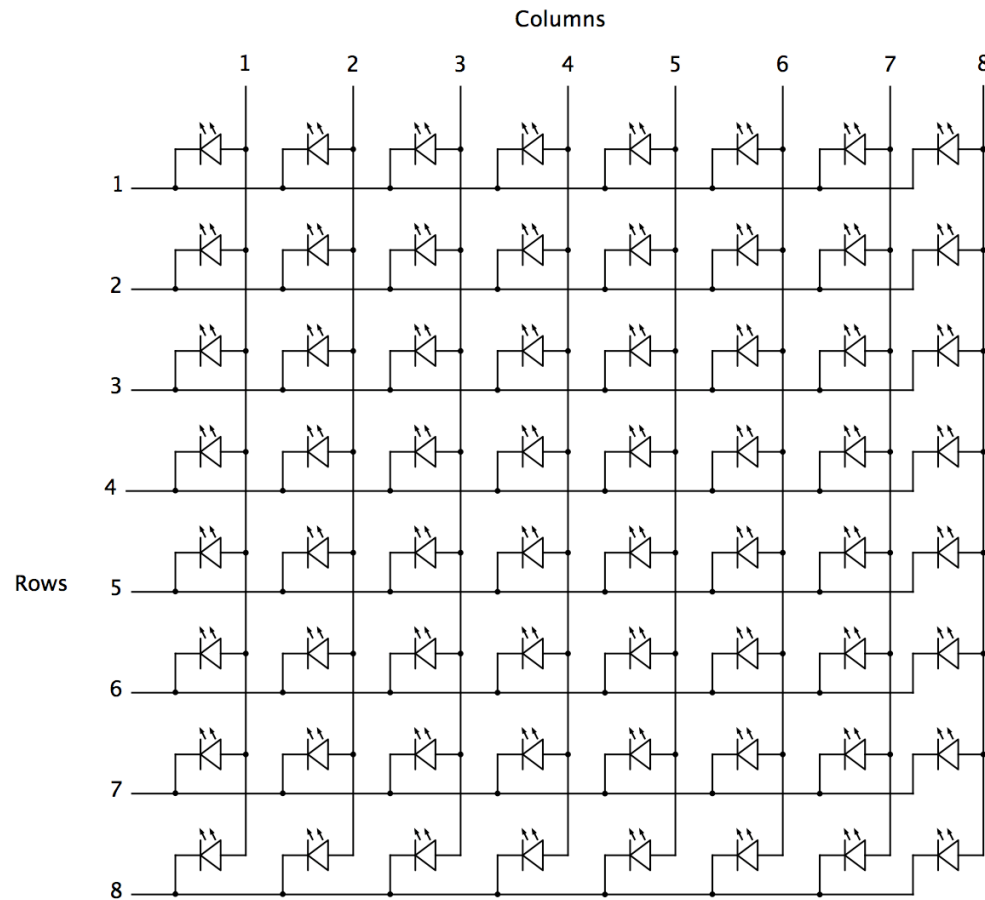


How LED matrix is connected



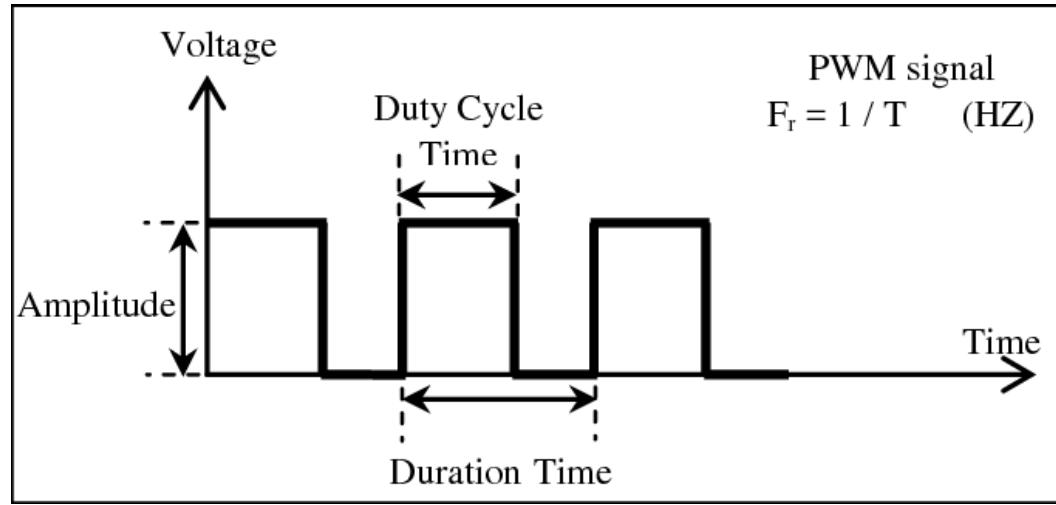
- ARM controls LED matrix indirectly
 - M54564 works as current source and ARM controls which channel is active
 - DM163 works as current sink

How LED matrix works in general



- This picture shows general idea how LED matrix works
 - Note this picture serves only as an example for one color of LED matrix
- You control vertical columns with channel bits and DM163 controls horizontal row inputs
 - Setting channel bit to '1' makes that column active since current can then flow through LEDs

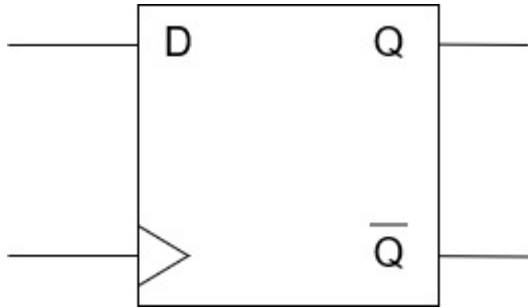
PWM






Source: https://www.researchgate.net/figure/PWM-signal-with-its-two-basic-time-periods_fig4_271437313

- PWM (Pulse Width Modulation) is continuing pulse with certain period of ON and OFF time.
- Most common usage is to control power like in this exercise.

D-flip-flop



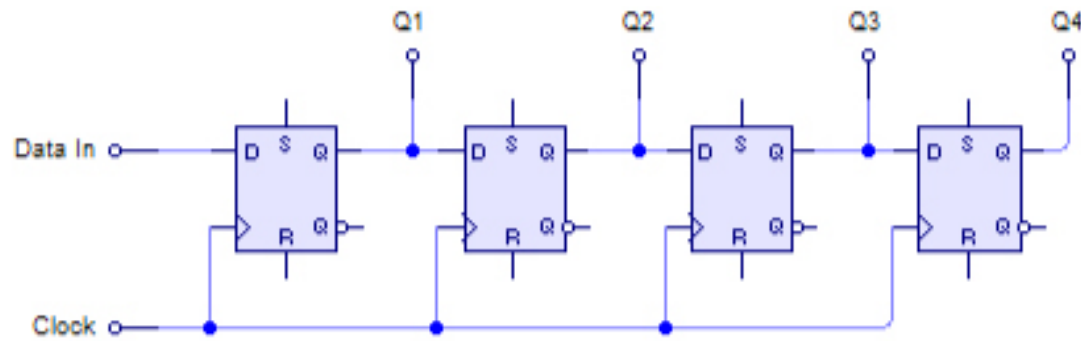
D Flip Flop Truth Table

CL (Note 1)	D	Q	\overline{Q}
	0	0	1
	1	1	0
	x	Q	\overline{Q}
x	x	0	1
x	x	1	0
x	x	1	1

No Change
x = Don't Care Case

- Data out (Q) changes only on rising edge of the clock
 - Q takes value of input D on rising edge
- D-flip-flop is basic component in digital systems
 - Used as synchronizer and in many other applications
 - In this exercise, DM163 chip uses D-flip-flop to form shift registers

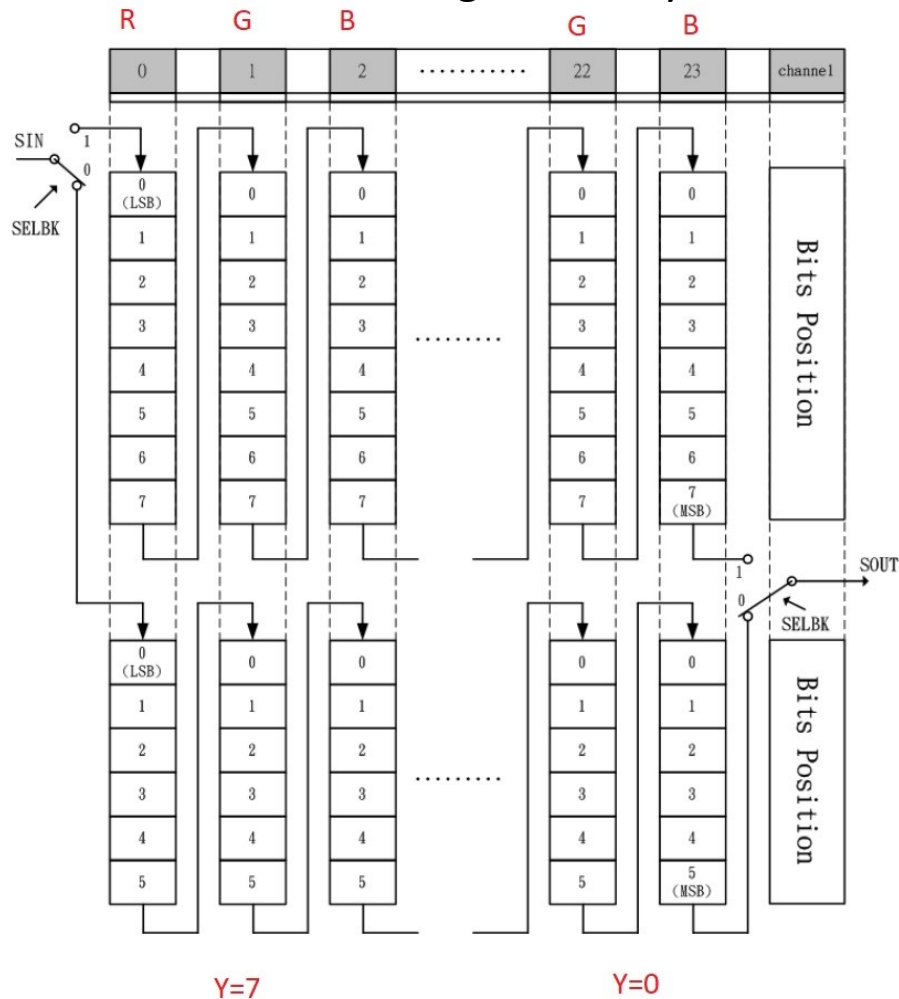
Shift register



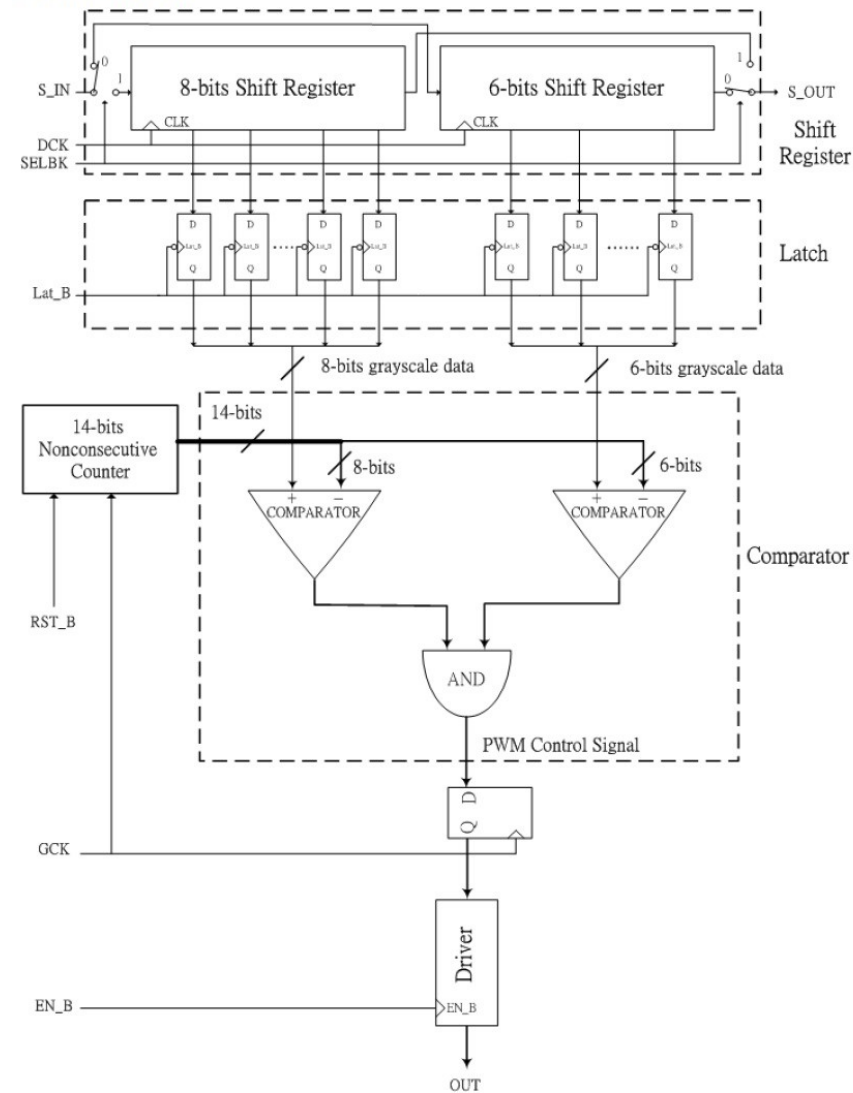
- Shift register is basically multiple D-flip-flops in series.
- Can be used to convert serial data to parallel form.

Shift register and block diagram

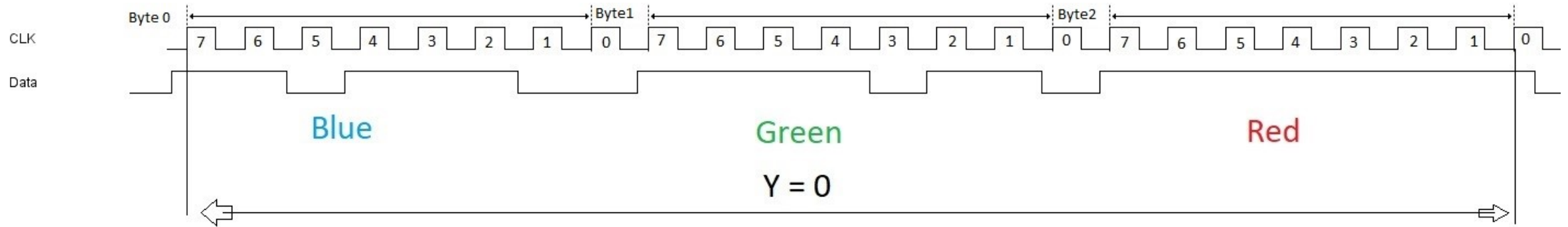
NOTE: Most Significant Byte is Blue



Block Diagram



Serial data



NOTE: Bytes transmitted as FIFO (first in, first out)

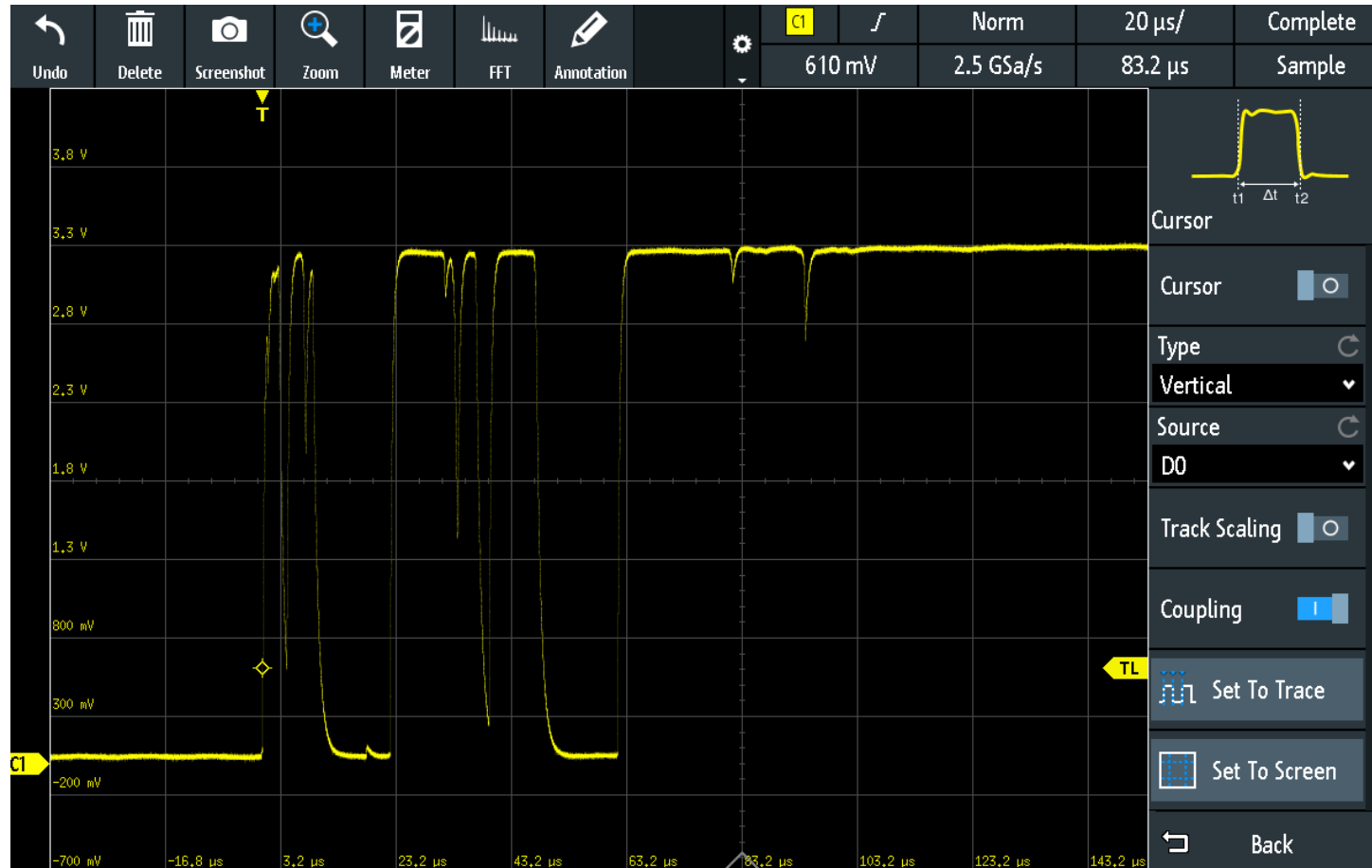
Other important stuff

- SB-bit chooses if data is send to 6 or 8-bit register.
- Reset must be set to inactive state before sending data to LED matrix
 - Set it to '1', since reset is active low
- 6-bit register banks must have values higher than 0
 - Because pixel brightness (if simplified) = 6-bit register*8-bit register
- Channel bits (C0-C7) are used to set active column

Common problems

- Ghostpixels
 - Screen refresh cycle is a bit faulty (dim pixel right/leftside of "real" pixel)
 - This is most likely bad programming (Channels active in wrong places)
- Sometimes program reads button to be pressed multiple times with one click
 - Switching vibration-> nothing can be done in this exercise, since it is more of an electrical feature.
 - Easy fix would have been adding a low-pass filter.
 - Another easy fix would be adding debounce circuit (software or hardware).
- In blinker.S, don't use ; to comment stuff, use //. IDE might show with ; that stuff is commented, in reality it isn't.
- Some micro-usb cables do not have datalines (programming PYNQ is impossible)

Button vibration measured with oscilloscope



In case of questions

- If any questions related to exercise
 - Ask in Mattermost
 - Send email: toni.jaaskelainen@tuni.fi

END