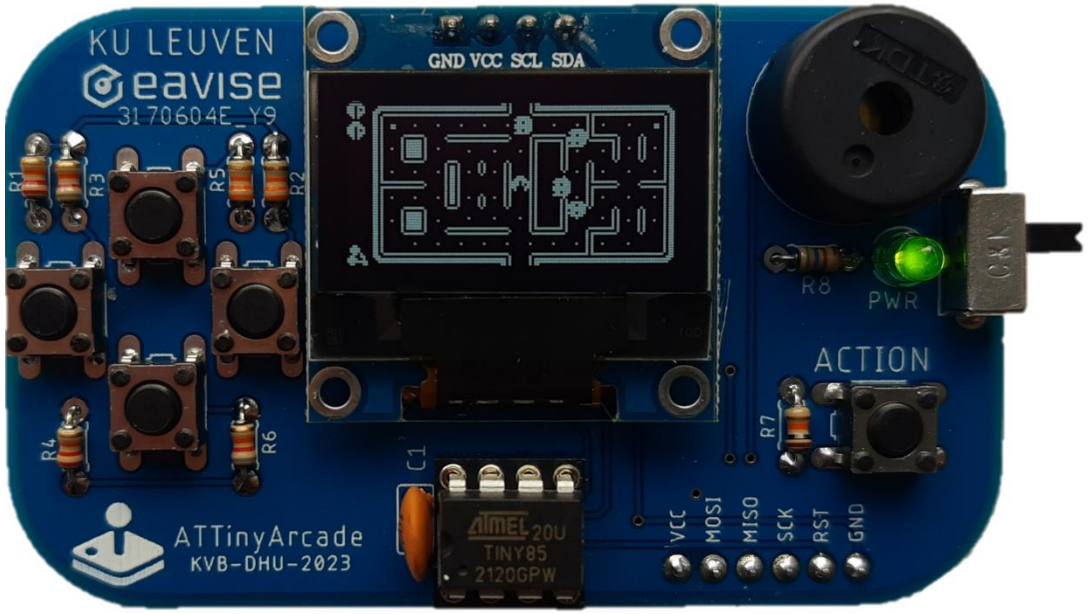
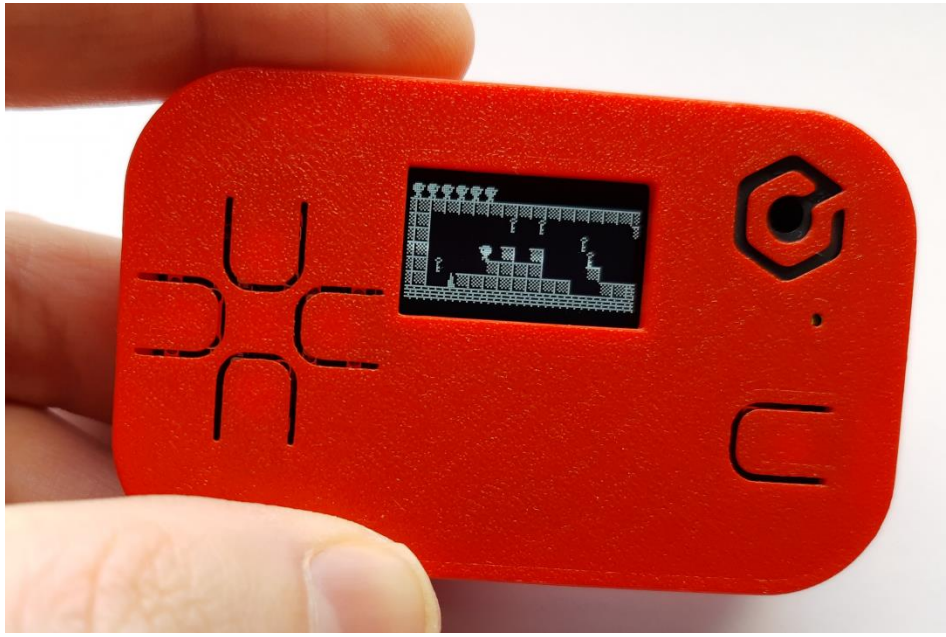
ATTiny85 Arcade workshop

Kristof Van Beeck - Dries Hulens





KU LEUVEN eavise

Overview of today

- 09:00 Introduction

- 09:30 PCB Design intro
- 09:45 PCB Soldering & electrical testing
- 10:45 PCB Test & bootloader
- 11:00 SW intro (OLED, I2C, ADCs, Interrupts)

Hardware

Software

- 12:00 Lunch break

- 12:30 SW intro (OLED, I2C, ADCs, Interrupts)
- 13:30 Develop simple game (Snake)
- 15:30 Download finished games
- 16:00 End

Software

Overview of today

- **09:00 Introduction**

- 09:30 PCB Design intro
- 09:45 PCB Soldering & electrical testing
- 10:45 PCB Test & bootloader
- 11:00 SW intro (OLED, I2C, ADCs, Interrupts)

Hardware

Software

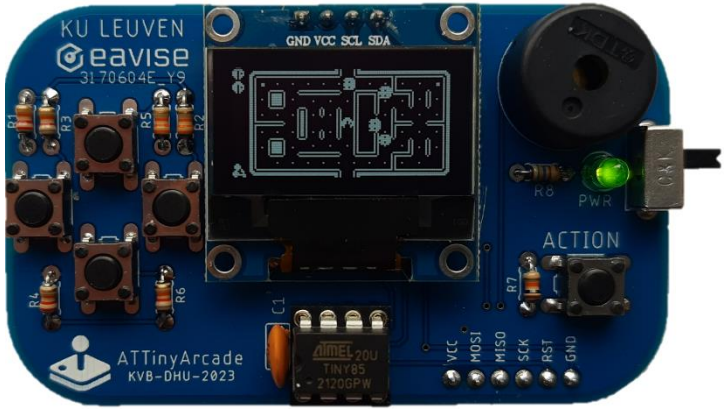
- 12:00 Lunch break

- 12:30 SW intro (OLED, I2C, ADCs, Interrupts)
- 13:30 Develop simple game (Snake)
- 15:30 Download finished games
- 16:00 End

Software

Introduction

- Build a mini hand-held battery power arcade game console from scratch



- Specs:
  - ATTiny85 controller (8K Bytes memory)
    16 MHz
  - OLED screen
  - 5 push buttons (full D-pad + action button)
  - On/off slide switch
  - Piezo buzzer for sound effects
  - CR2032 battery
  - Through-hole PCB

Introduction

- Credits!

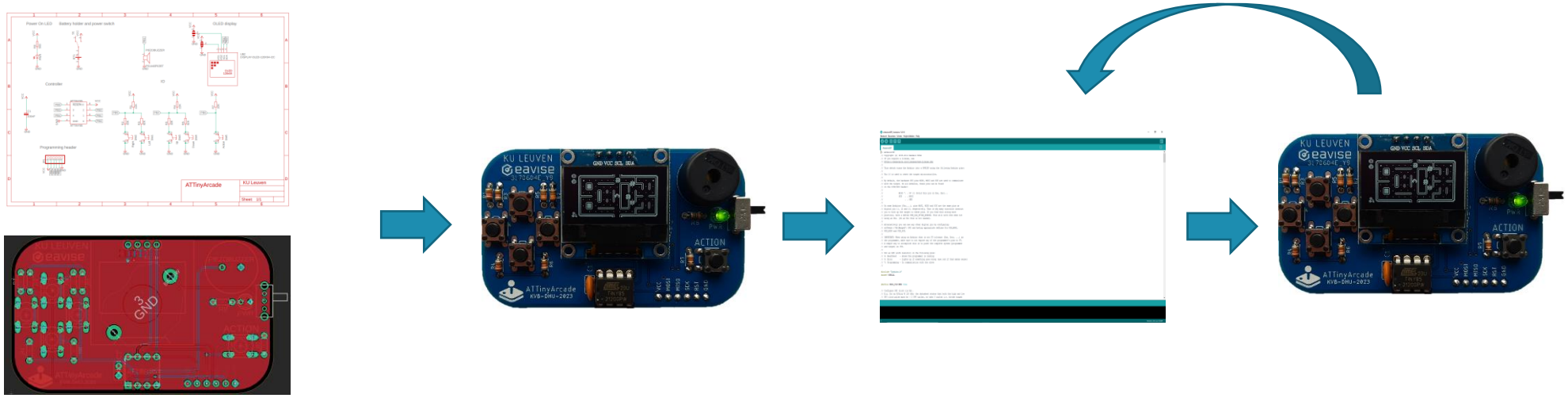
  Original implementation from Electronoobs

  D-pad upgrade & most games from Daniel C.

  Some games created by Andy Jackson

Introduction

- We’ll go over entire development:
  - Schematic ➔ draw and manufacture PCB ➔ Soldering ➔ Electrical test ➔ SW test ➔ Software implementation of simple game



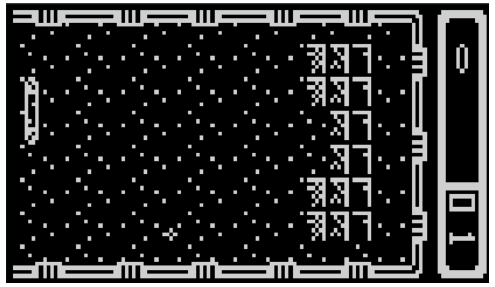
  - PCB Design: **EAGLE Layout Editor**
  - Software is written using **Arduino IDE** and language

Introduction

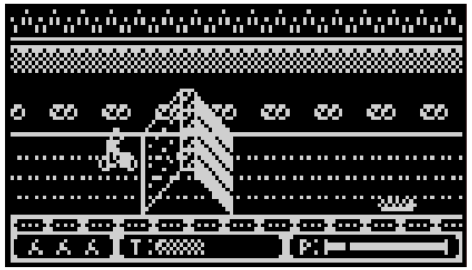
- 17 supported games (one at a time ☺):

Q*bert (1982)

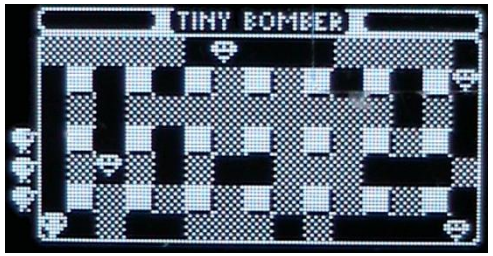
Arkanoid (1986)



Excite Bike (1984)

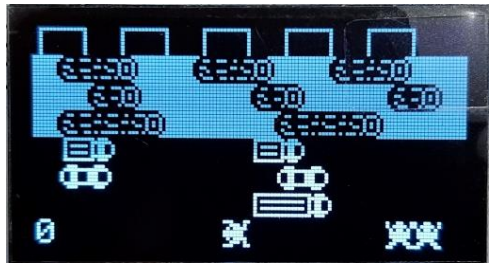


Bomberman (1983)

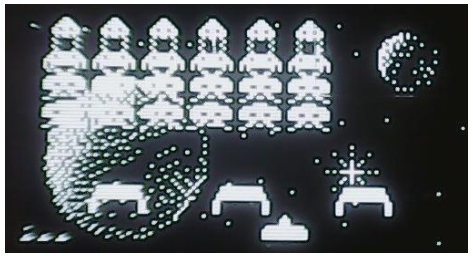


Dugger (1988)

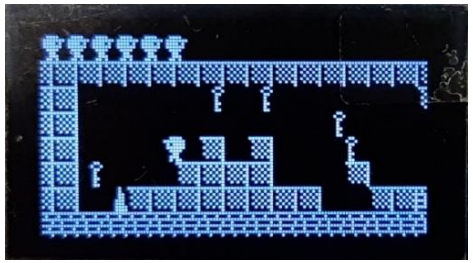
Frogger (1981)



Space Invaders (1978)



Tiny Gilbert (platformer)

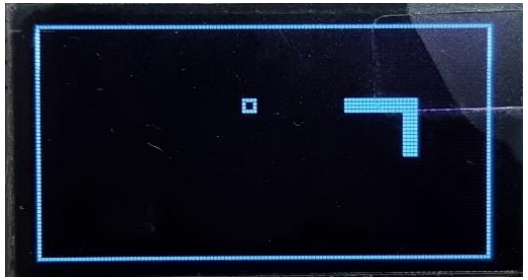


KU LEUVEN eavise

Introduction

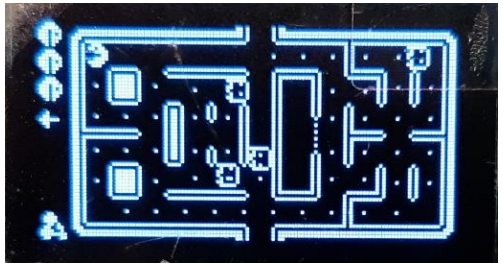
- 17 supported games (one at a time ☺):

Snake (1976)

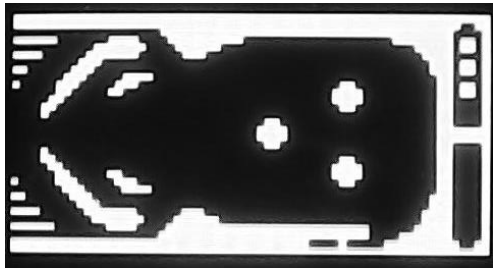


Morpion (tic-tac-toe)

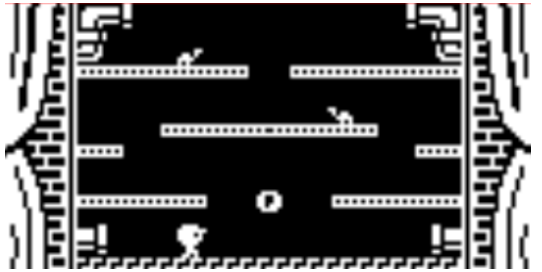
Pacman (1980)



Pinball (1984)

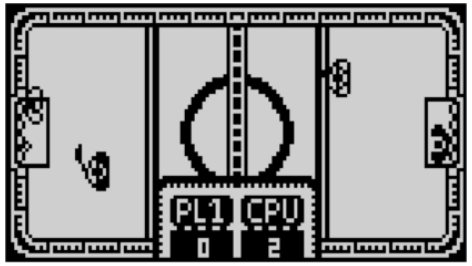


Pipeline (1978)

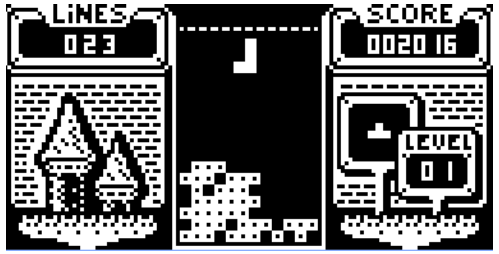


Plaque Attack (1983)

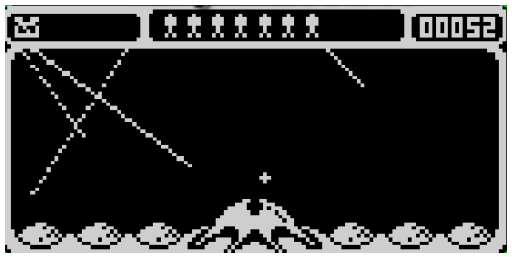
Hat Trick (1988)



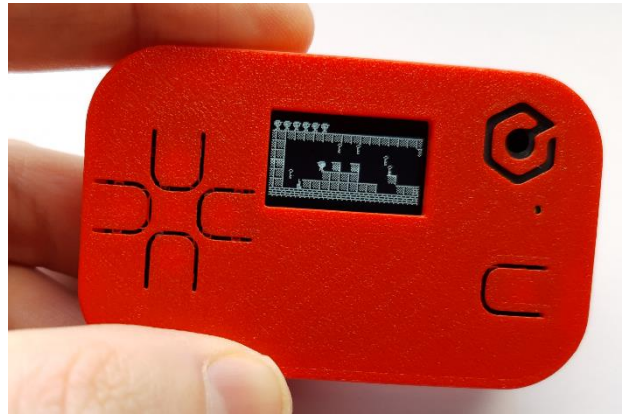
Tetris (1984)



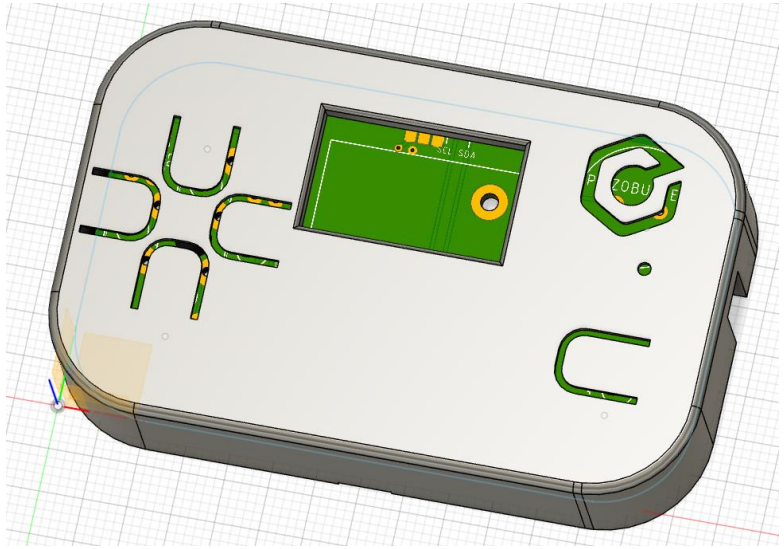
Missile Command (1980)

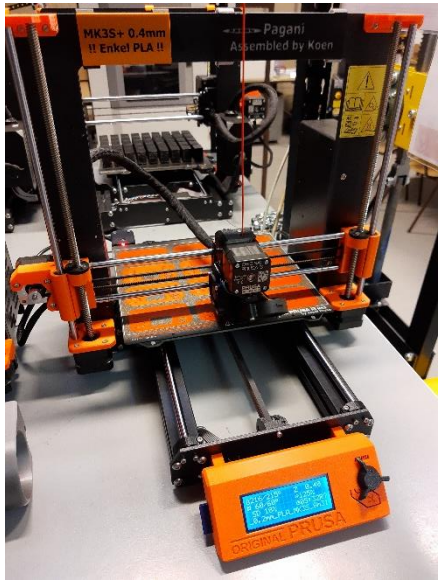


Introduction



- 3D printed housing
- Not part of this workshop
- Fusion360 CAD software (free for students!) – Prusa MK3 printer





Introduction

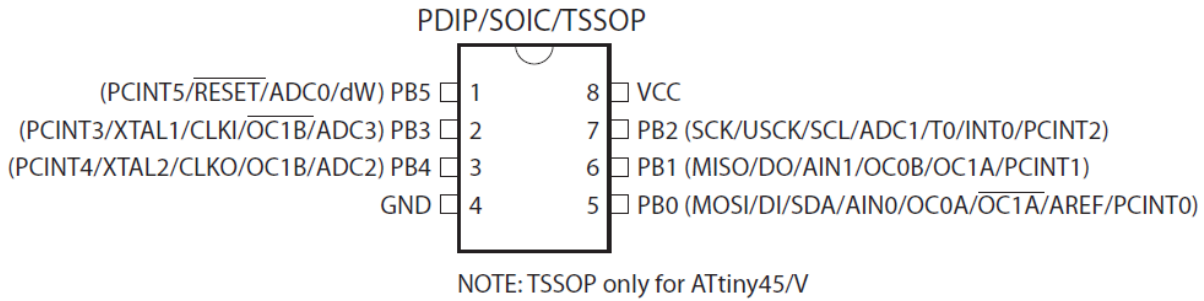
- Location of course files:

https://gitlab.com/EAVISE/workshops/

ATTinyArcadeV2

- **Task 1:**
  - Download ZIP from course URL
    - Contains these slides, PCB, datasheets, games, SW templates,…
  - Extract to D:\ drive

Introduction – ATTiny85 microcontroller



NOTE: TSSOP only for ATtiny45/V

ATtiny25/V / ATtiny45/V / ATtiny85/V
Summary

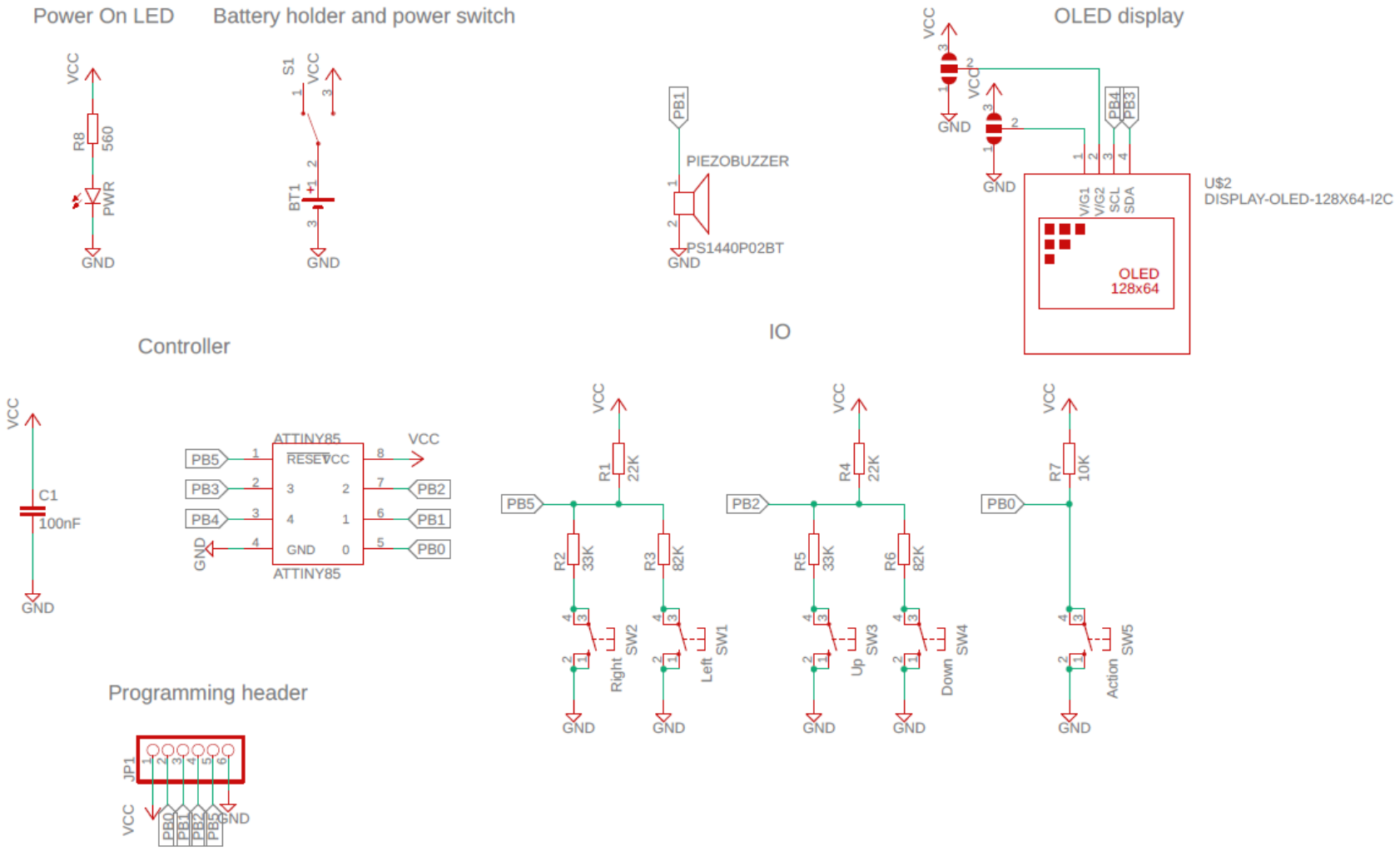
Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 120 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
- Non-volatile Program and Data Memories
  - 2/4/8K Bytes of In-System Programmable Program Memory Flash
    - Endurance: 10,000 Write/Erase Cycles
  - 128/256/512 Bytes In-System Programmable EEPROM
    - Endurance: 100,000 Write/Erase Cycles
  - 128/256/512 Bytes Internal SRAM
  - Programming Lock for Self-Programming Flash Program and EEPROM Data Security
- Peripheral Features
  - 8-bit Timer/Counter with Prescaler and Two PWM Channels
  - 8-bit High Speed Timer/Counter with Separate Prescaler
    - 2 High Frequency PWM Outputs with Separate Output Compare Registers
    - Programmable Dead Time Generator
  - USI – Universal Serial Interface with Start Condition Detector
  - 10-bit ADC
    - 4 Single Ended Channels
    - 2 Differential ADC Channel Pairs with Programmable Gain (1x, 20x)
    - Temperature Measurement
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - debugWIRE On-chip Debug System
  - In-System Programmable via SPI Port
  - External and Internal Interrupt Sources
  - Low Power Idle, ADC Noise Reduction, and Power-down Modes
  - Enhanced Power-on Reset Circuit
  - Programmable Brown-out Detection Circuit
  - Internal Calibrated Oscillator
- I/O and Packages
  - Six Programmable I/O Lines
  - 8-pin PDIP, 8-pin SOIC, 20-pad QFN/MLF, and 8-pin TSSOP (only ATtiny45/V)
- Operating Voltage
  - 1.8 - 5.5V for ATtiny25V/45V/85V
  - 2.7 - 5.5V for ATtiny25/45/85
- Speed Grade
  - ATtiny25V/45V/85V: 0 – 4 MHz @ 1.8 - 5.5V, 0 - 10 MHz @ 2.7 - 5.5V
  - ATtiny25/45/85: 0 – 10 MHz @ 2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V
- Industrial Temperature Range
- Low Power Consumption
  - Active Mode:
    - 1 MHz, 1.8V: 300 µA
  - Power-down Mode:
    - 0.1 µA at 1.8V

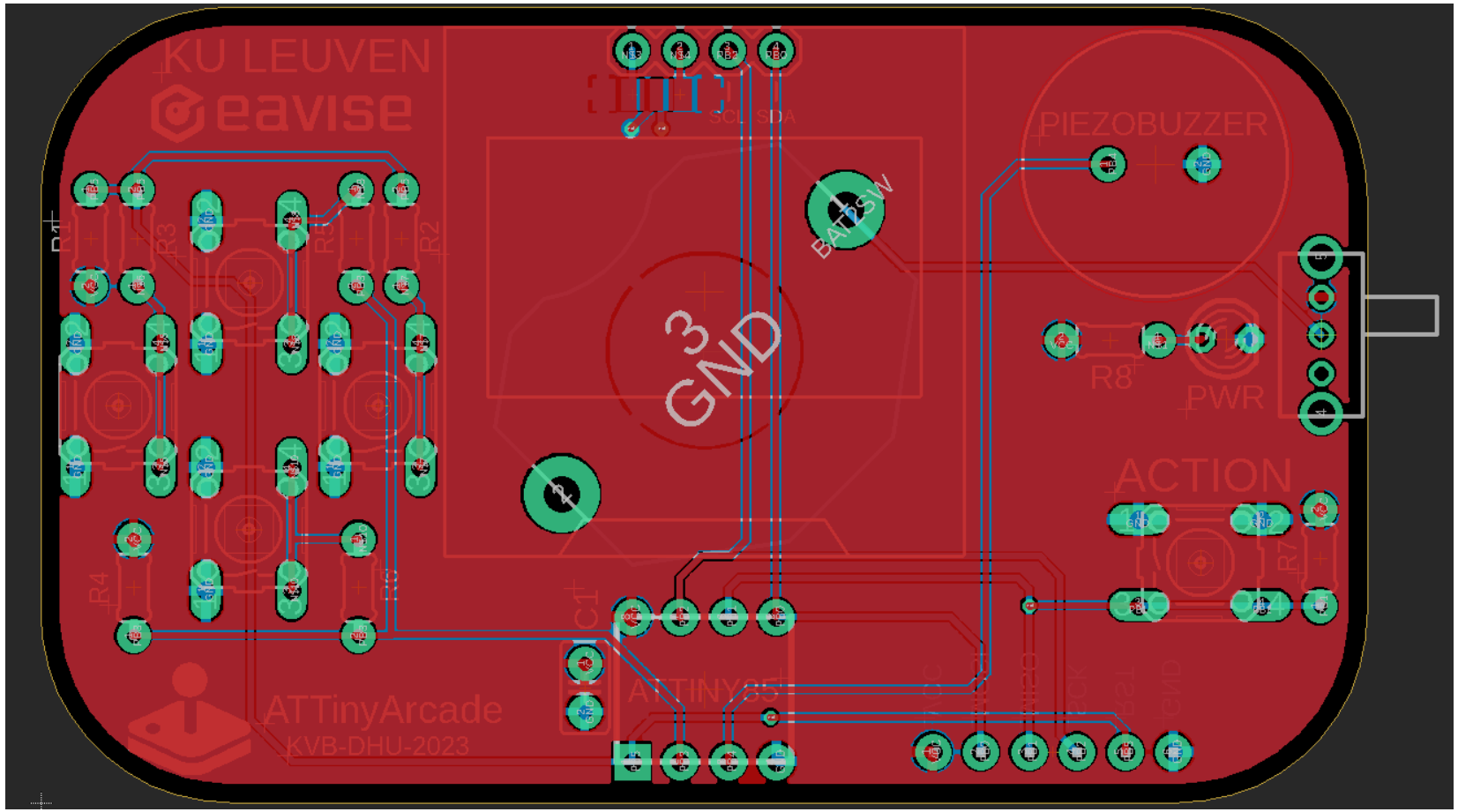
Rev. 2586QS–AVR–08/2013

KU LEUVEN eavise

Introduction - schematic



Introduction - PCB



Introduction

- **Task 2:**
  - Use the datasheets to calculate the battery life of this mini handheld game console!

Overview of today

- 09:00 Introduction

- **09:30 PCB Design intro**
- 09:45 PCB Soldering & electrical testing
- 10:45 PCB Test & bootloader
- 11:00 SW intro (OLED, I2C, ADCs, Interrupts)

Hardware

Software

- 12:00 Lunch break

- 12:30 SW intro (OLED, I2C, ADCs, Interrupts)
- 13:30 Develop simple game (Snake)
- 15:30 Download finished games
- 16:00 End

Software

PCB Design intro

- **Task 3:**
  - Copy folder “ATTinyArcade_PCB” (under hardware) to:

    C:\Users\student\Documents\EAGLE\projects\

    If folder does not exist yet, first open Eagle and login

  - Open Eagle from desktop

    If login is required:

    “elektronicaworkshop@gmail.com”

    pwd: “arduinoide1”
  - Open project (both schematic and PCB – press F5 if not yet visible)

PCB Design intro

- **Task 4:**
  - Add the power LED and resistor to the schematic
  - Position and route both the LED and resistor on the PCB
  - Perform ERC and DRC check
    - https://jlcpcb.com/capabilities/Capabilities

    ➔ Use the **jlcpcb2layer.dru** file
  - Use the CAM generator to generate Gerber files

    ➔ Use the **jlcpcb_2_layer_v9.cam** file
  - Upload Gerber files to your favorite PCB fab

    ➔ e.g. https://jlcpcb.com/
  - View PCB preview and check for errors

Overview of today

- 09:00 Introduction

- 09:30 PCB Design intro
- **09:45 PCB Soldering & electrical testing**
- 10:45 PCB Test & bootloader
- 11:00 SW intro (OLED, I2C, ADCs, Interrupts)

Hardware

Software

- 12:00 Lunch break

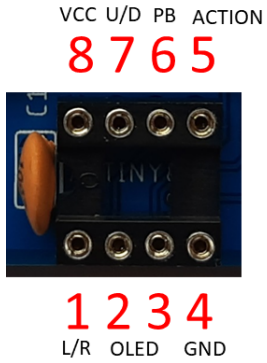
- 12:30 SW intro (OLED, I2C, ADCs, Interrupts)
- 13:30 Develop simple game (Snake)
- 15:30 Download finished games
- 16:00 End

Software

PCB Soldering

- PCB for this workshop has been designed completely through hole
  - Solder from lowest components to highest components
  - Do not forget solder jumpers
  - Note polarities and positions of components!

- After soldering we perform an electrical test



PIN	CONNECTION	Measured voltages, with GND as reference
1	Left and right buttons	3.1V (none), 2.4V (left), 1.8V (right)
2	OLED – SDA	(3.1V)
3	OLED – SCL	(3.1V)
4	GND	(0V)
5	Action button	3.1V (none), 0V (action)
6	Piezzo buzzer	/
7	Up and down buttons	3.1V (none), 2.4V (down), 1.8V (up)
8	Vcc	3.2V

PCB Soldering

See solder guide here:

https://gitlab.com/EAVISE/workshops/attinyarcadev2/-/blob/main/SOLDER_GUIDE.md

PCB Soldering

- Move to lab A111
- Hope to finish around 10:45

Overview of today

- 09:00 Introduction

- 09:30 PCB Design intro
- 09:45 PCB Soldering & electrical testing
- **10:45 PCB Test & bootloader**
- 11:00 SW intro (OLED, I2C, ADCs, Interrupts)

Hardware

Software

- 12:00 Lunch break

- 12:30 SW intro (OLED, I2C, ADCs, Interrupts)
- 13:00 Develop simple game (Snake)
- 15:30 Download finished games
- 16:00 End

Software

PCB Test & bootloader

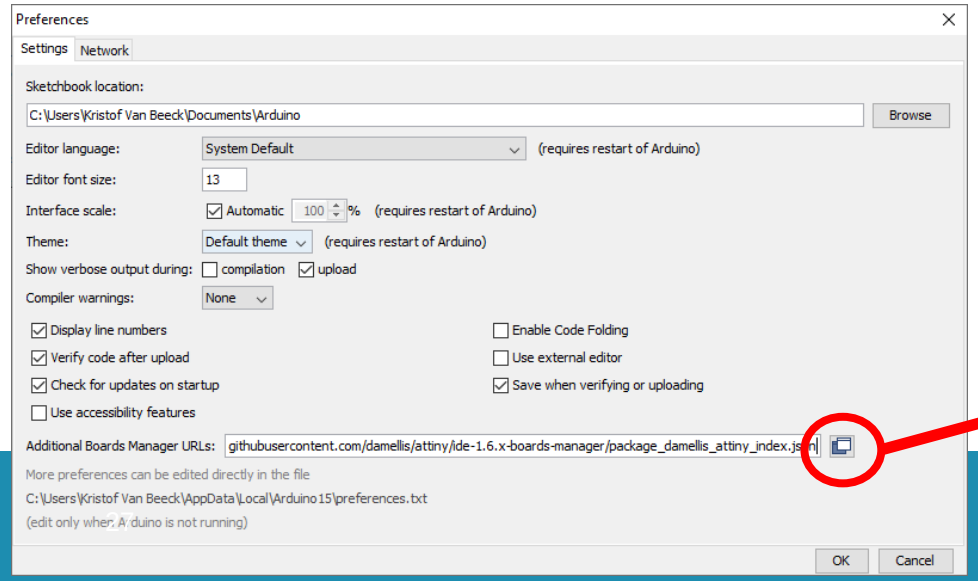
- Let’s now burn the bootloader (set the fuses) and test the hardware of the board!
- **Task 5**

PCB Test & bootloader

- First, install the ATTiny85 libraries
- Start Arduino IDE, go to *File* ➔ *Preferences*

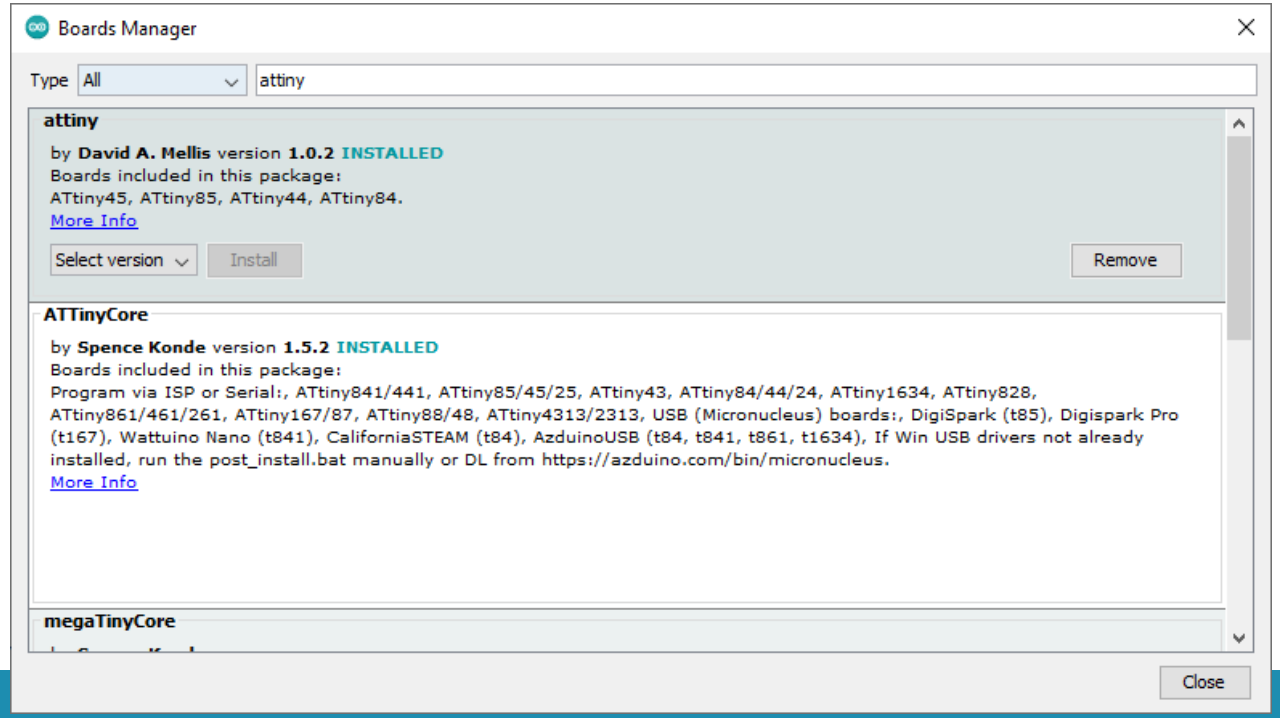
Add an additional board manager URL:

https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json



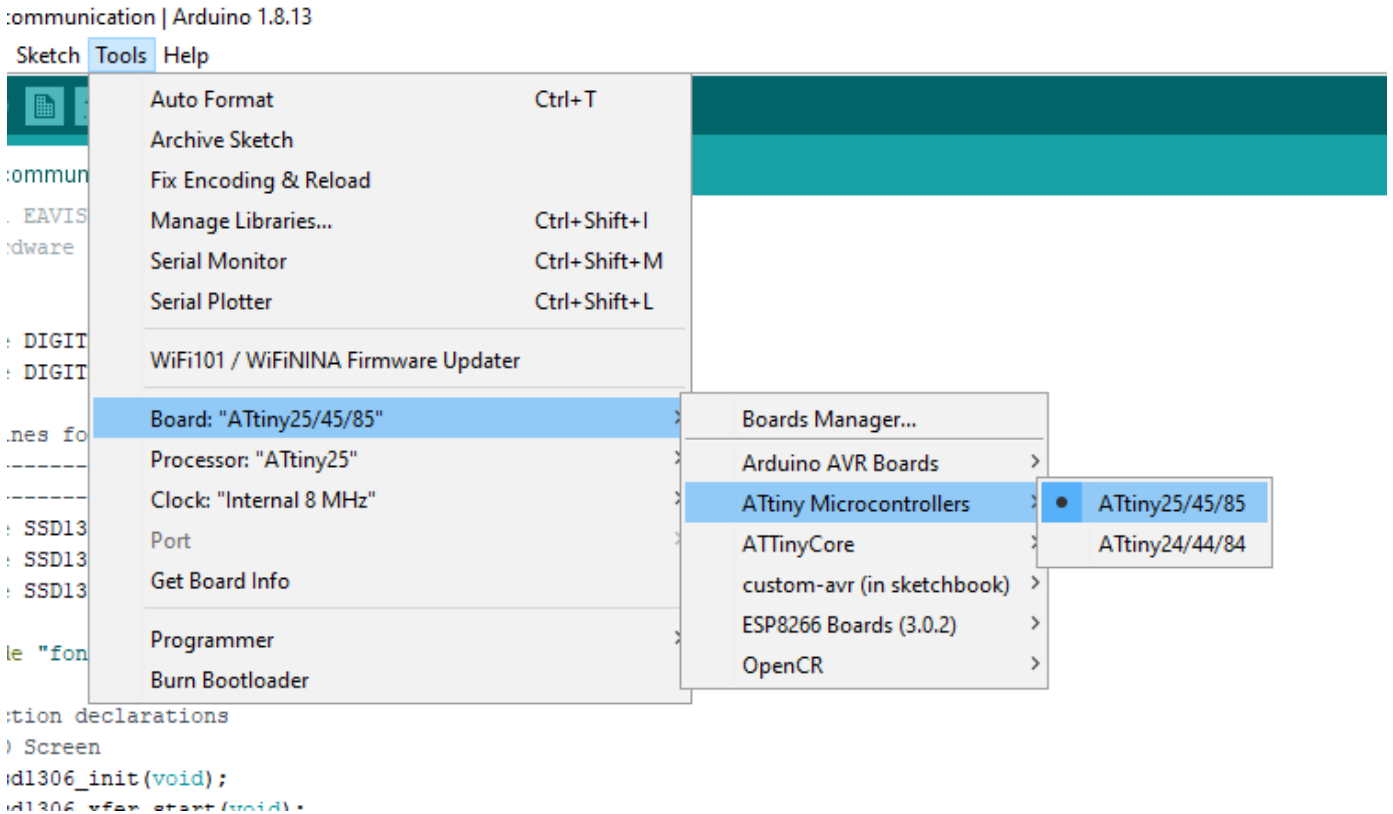
PCB Test & bootloader

- *Tools ➔ Board ➔ Board manager*
- *Search for attiny*, install the board:
  - *attiny (by David A. Mellis - version 1.0.2)*



PCB Test & bootloader

- Under *Tools* ➔ *Board* ➔ *“Attiny Microcontrollers”* should now be available

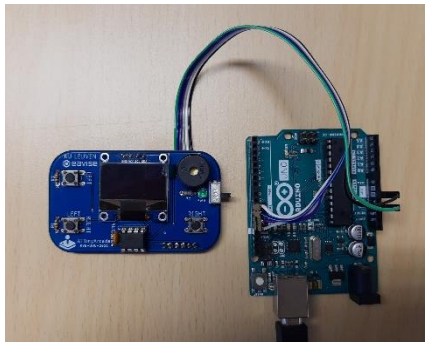


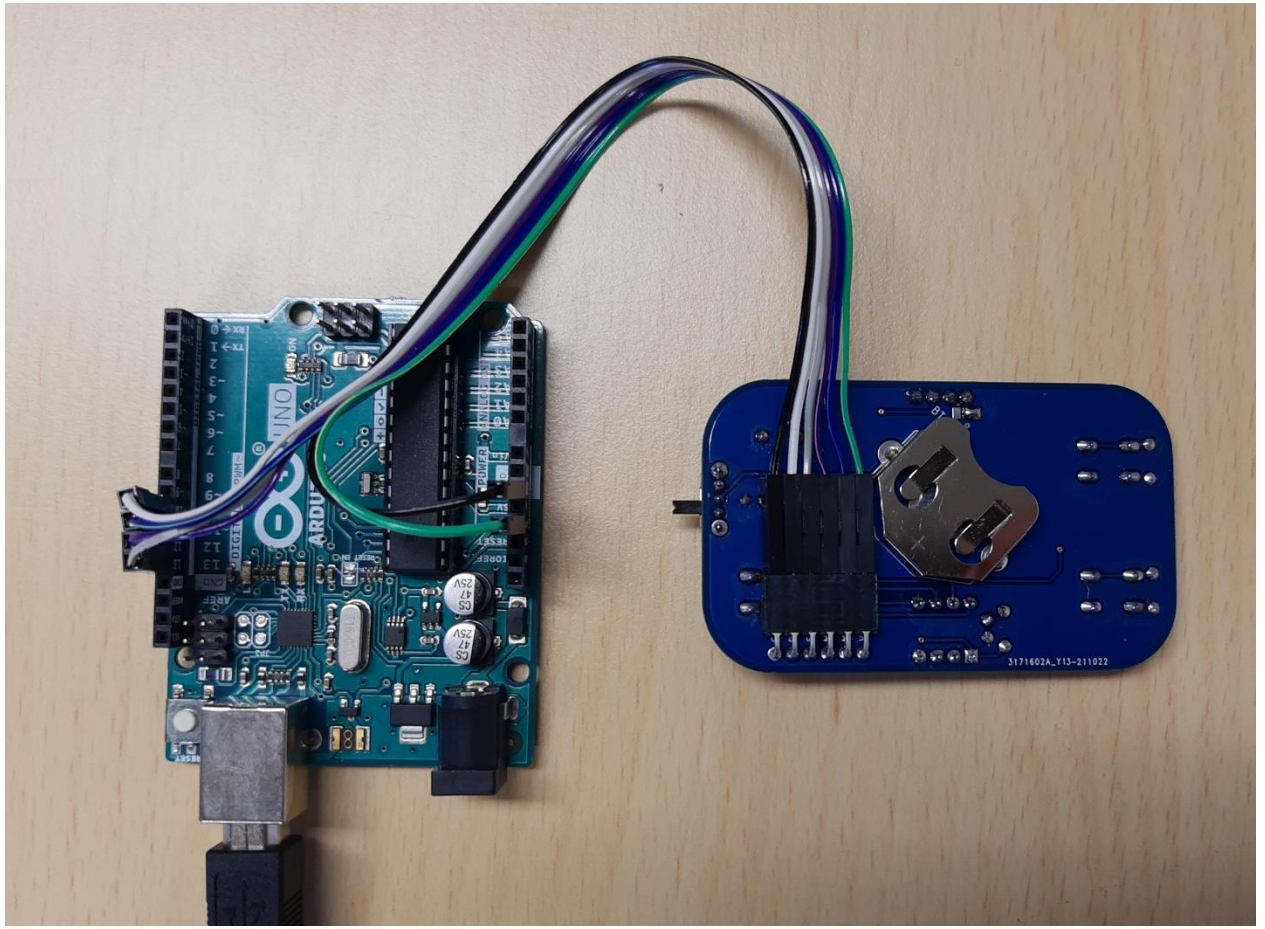
PCB Test & bootloader

- We’re going to use the Arduino Uno as ISP programmer to program the ATTiny85 on our PCB, using the header at the back
- Connect the Arduino Uno to the PCB header as shown on next slide (do not connect the Uno through USB yet)
- Double check connections when finished
- Connect Arduino Uno to USB on desktop

PCB Test & bootloader

Arduino Uno	ATTiny85 Arcade PCB
3V3	VCC
D11	MOSI
D12	MISO
D13	SCK
D10	RESET
GND	GND

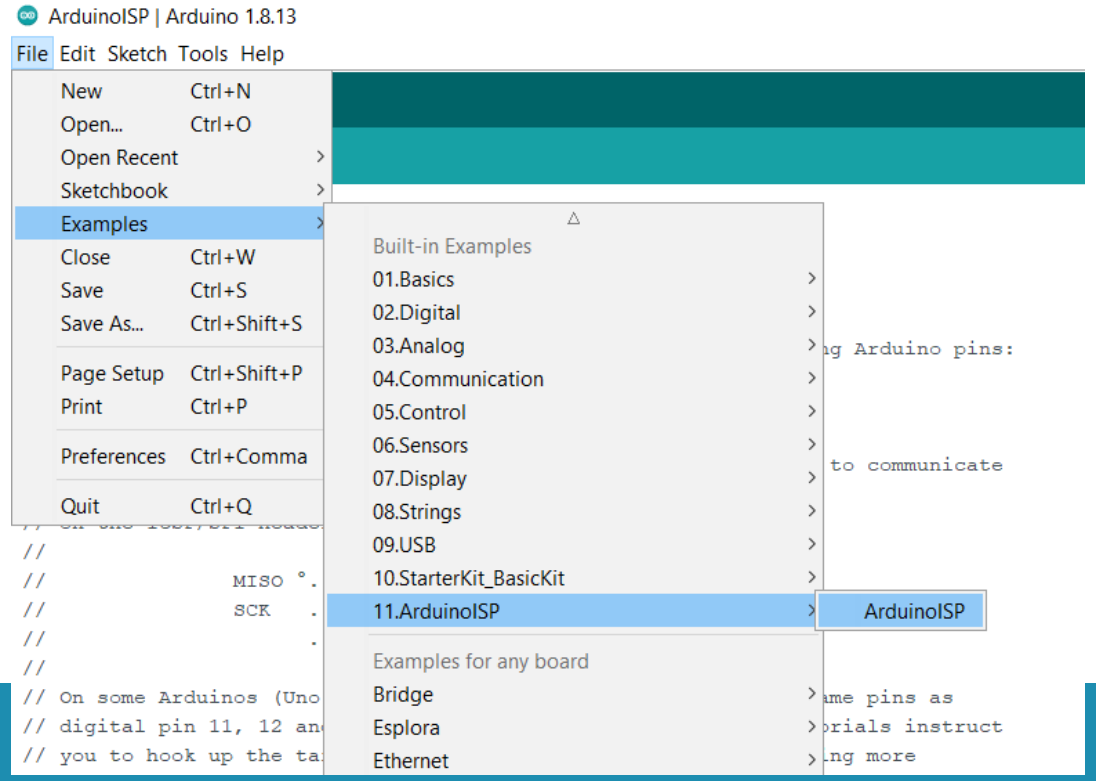




PCB Test & bootloader

- Upload the Arduino ISP code to the Arduino Uno
- First, open the ISP code:

File ➔ Examples ➔ 11. ArduinoISP ➔ ArduinoISP



PCB Test & bootloader

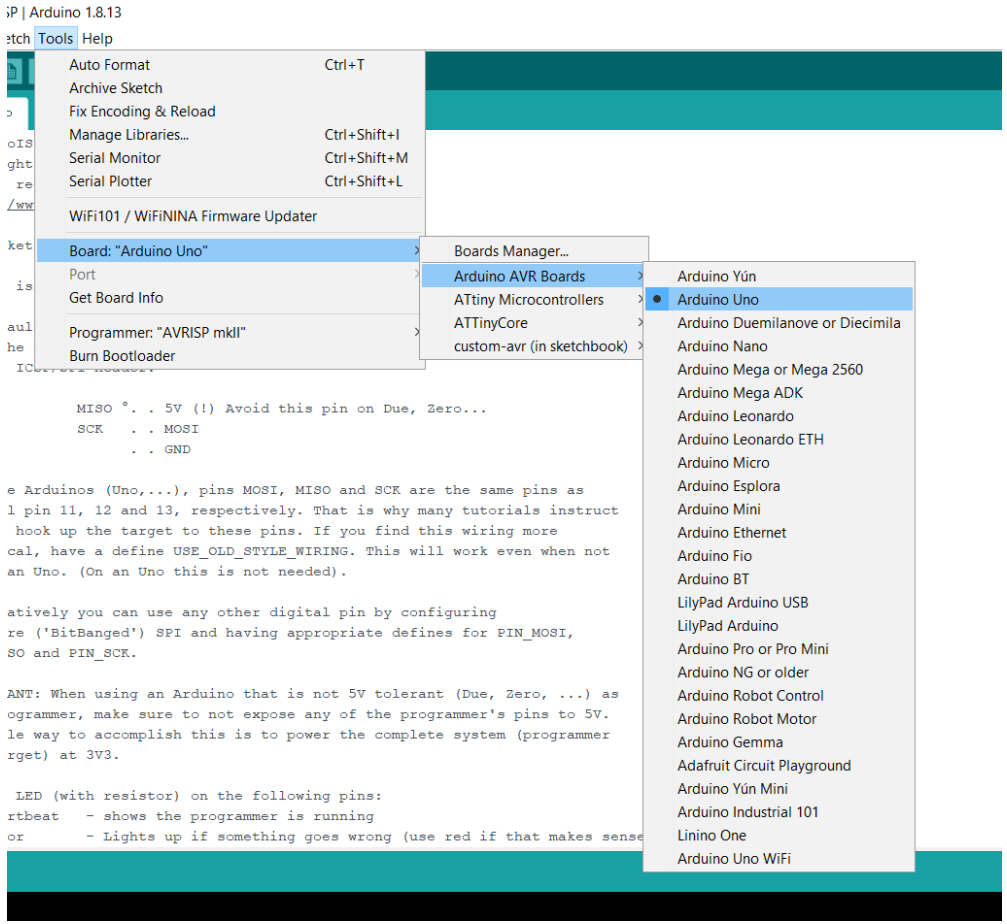
- Select the correct board:

Tools ➔ Board
(set to Arduino Uno)

Tools ➔ Port
(select correct COM port)

Tools ➔ Programmer
(select AVRISP mkII)

- **Upload code to board!**



PCB Test & bootloader

- We can now use the Arduino Uno as ISP to program our ATTiny85 PCB
- First, we need to burn the bootloader (which sets the fuses)

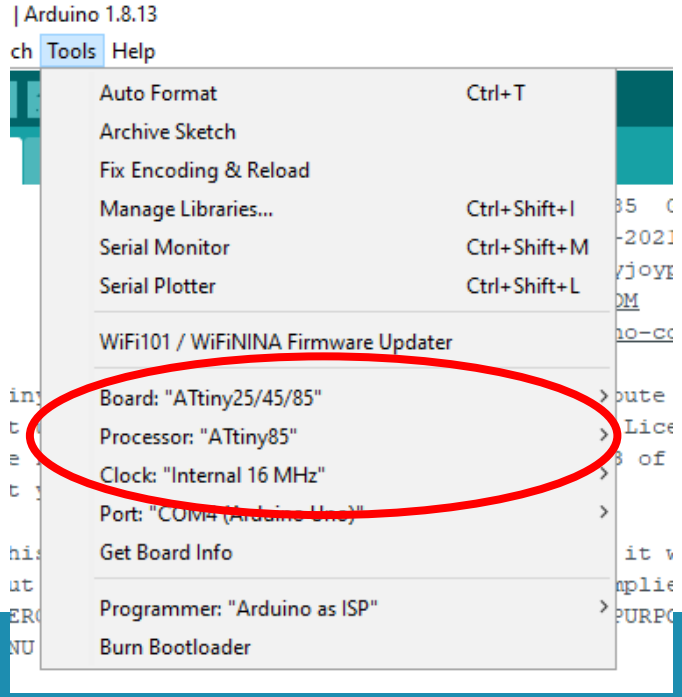
PCB Test & bootloader

- Under *Tools*, make sure to select the following options:

Board: “ATTiny25/45/85”

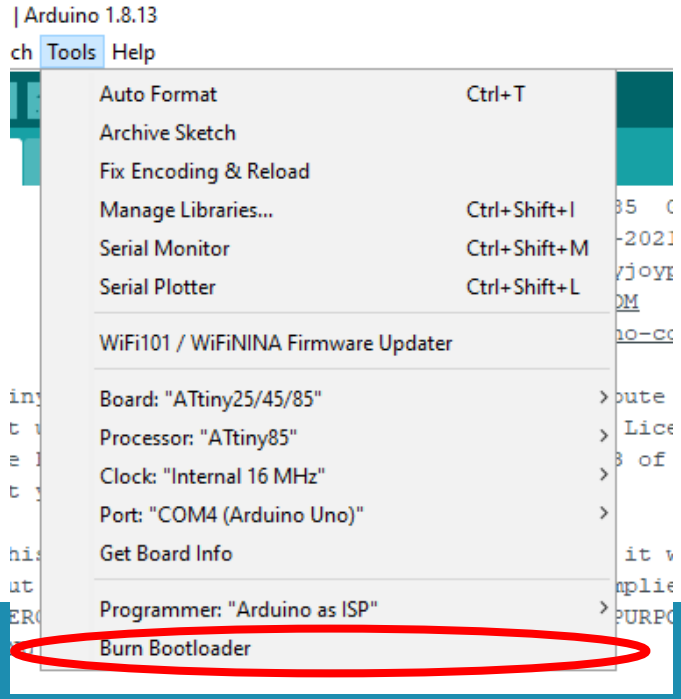
Processor: “ATTiny85”

Clock: 16 MHz internal ➔ EXTREMELY IMPORTANT



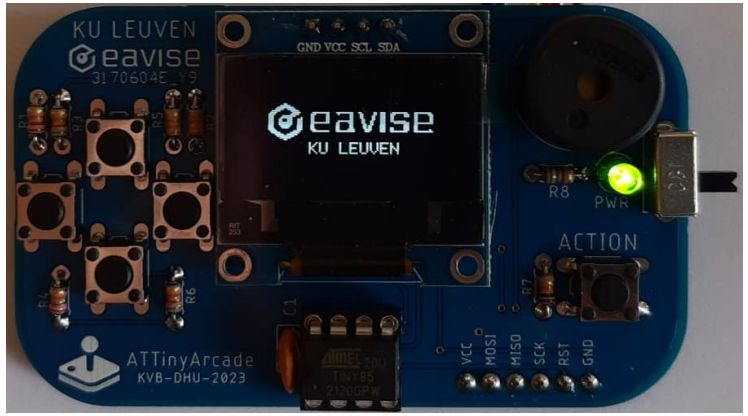
PCB Test & bootloader

- Make sure to change *Tools* ➔ *Programmer* to *Arduino as ISP*
- Click on *Tools* ➔ *Burn bootloader*
- You should get: "*Done burning bootloader*" if everything went fine (and some beep noises ☺), otherwise check connections again



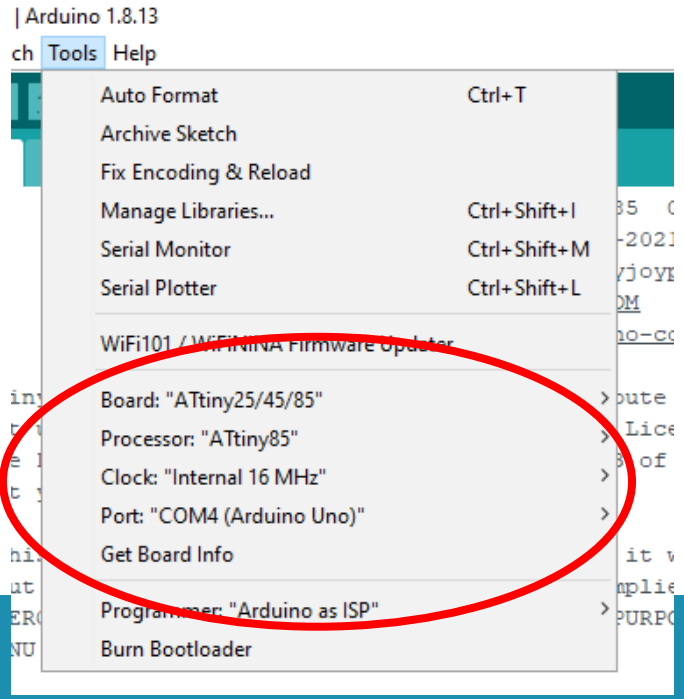
PCB Test & bootloader

- We can now program the ATTiny85 from the Arduino IDE
- First, we run a simple hardware test program
- This runs the following sequence:
  - Initialize ports of ATTiny85
  - Initialize OLED screen
  - Shows EAVISE logo (2 seconds)
  - Plays sound
  - Shows test screen to test D-pad and action button



PCB Test & bootloader

- This sketch is found under “/software/hw_test/”
- Open in Arduino IDE, make sure that *ATTiny85* board is still selected with 16 MHz clock, and the programmer is *Arduino as ISP*
- *Upload the code and test the hardware!*



Overview of today

- 09:00 Introduction

- 09:30 PCB Design intro
- 09:45 PCB Soldering & electrical testing
- 10:45 PCB Test & bootloader
- **11:00 SW intro (OLED, I2C, ADCs, Interrupts)**

Hardware

Software

- 12:00 Lunch break

- 12:30 SW intro (OLED, I2C, ADCs, Interrupts)
- 13:30 Develop simple game (Snake)
- 15:30 Download finished games
- 16:00 End

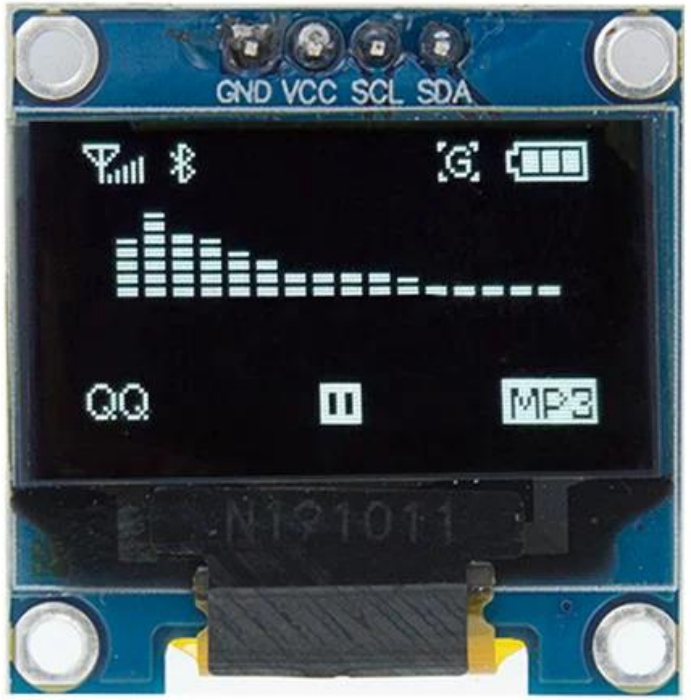
Software

SW intro (OLED, I2C, ACD, Interrupts)

- Let’s now discuss the software side:
  - The OLED screen interfacing
    - I2C interface protocol
  - User input
    - ADCs
    - Interrupts (optional)

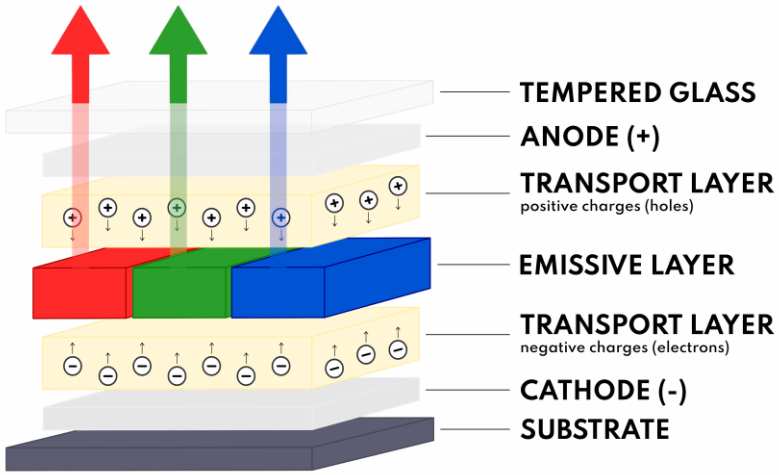
SW intro (OLED)

- OLED screen specs:
  - SSD1306 controller
  - Supports **I2C** & SPI
  - 128 x 64 pixels
  - 0.96-inch diagonal
  - Supply voltage 3.3V – 5V



SW intro (OLED)

- OLED technology:
  - Similar to LED, emissive electroluminescent layer is a film of organic compound that emits light in response to an electric current
  - PMOLED and AMOLED



https://www.unisystem-displays.com/en/news/all-about-oleds.html

SW intro (OLED)

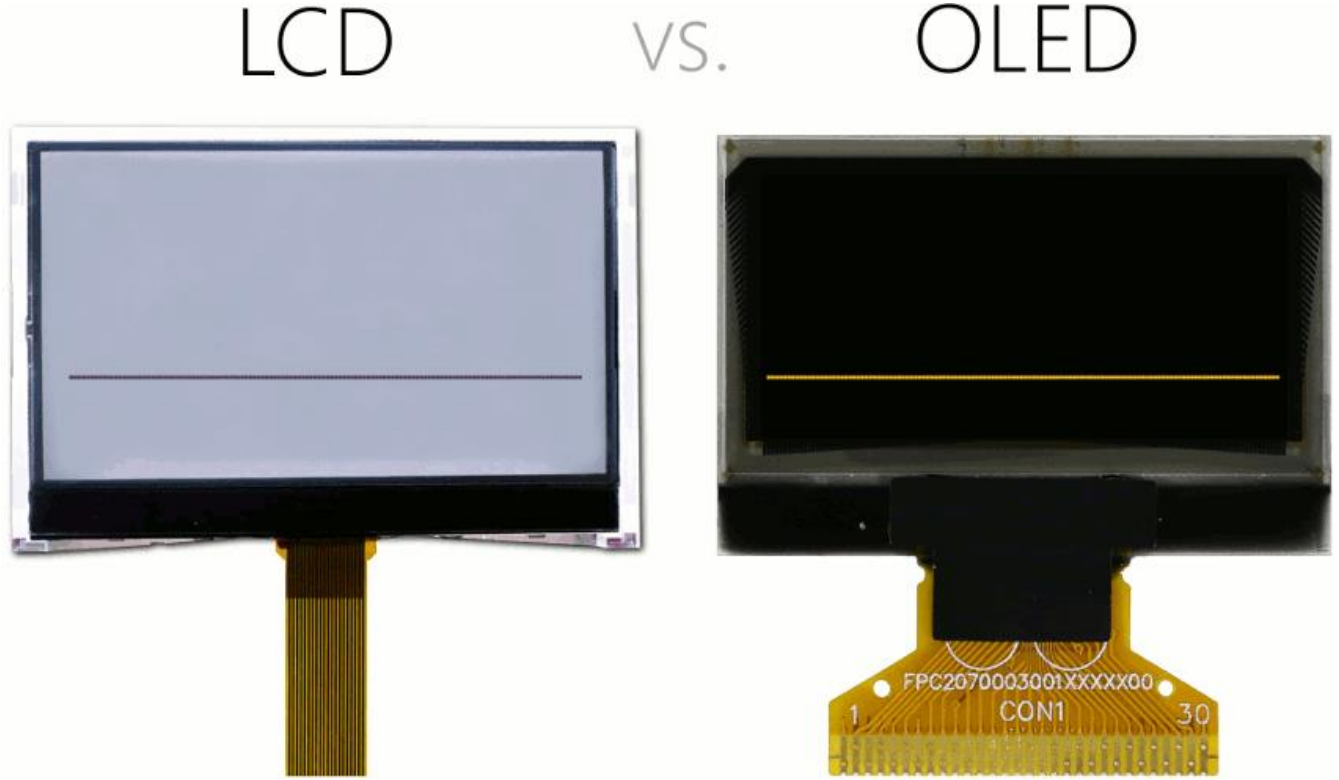
- OLED technology:
  - Each pixel emits visible light (as opposed to an LCD screen)

- Advantages:
  - Excellent brightness and contrast
  - Wide viewing-angle
  - No need for backlight: smaller, lightweight, flexible, uses less power
  - Fast response time

- Disadvantages:
  - Expensive technology
  - Limited lifetime of organic material
  - Prone to environmental factors (e.g. moisture)

https://www.unisystem-displays.com/en/news/all-about-oleds.html

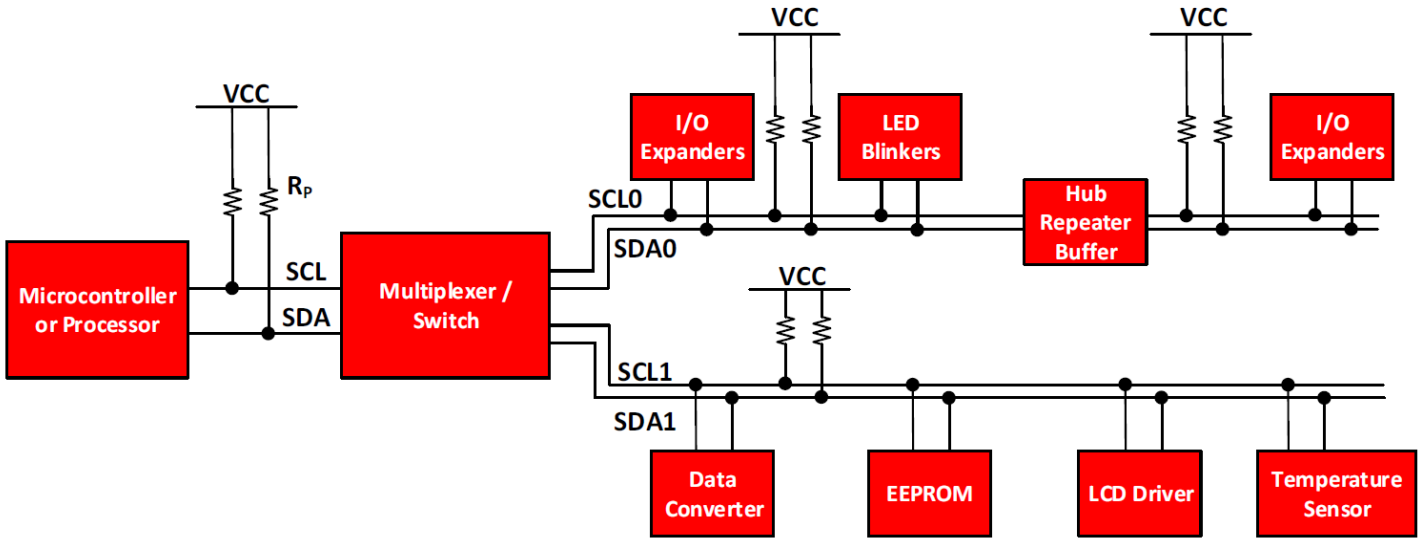
SW intro (OLED)



https://www.unisystem-displays.com/en/news/all-about-oleds.html

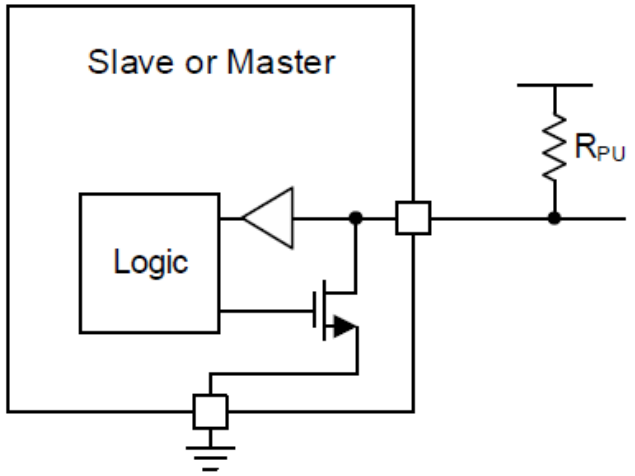
SW intro (I2C)

- Our screen uses I2C for communication with the SSD1306 controller:
  - Inter-Integrated circuit bus
  - Two wire interface: SCK & SDA: bidirectional and open-collector or open-drain (pull-up required)



Texas Instruments - Application Report SLVA704–June 2015 Understanding the I2C Bus

SW intro (I2C)

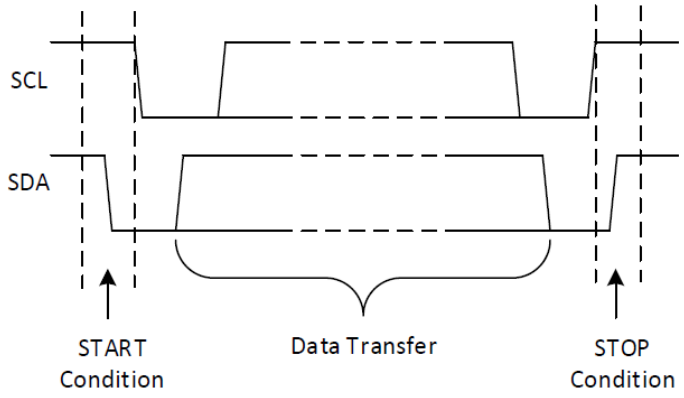


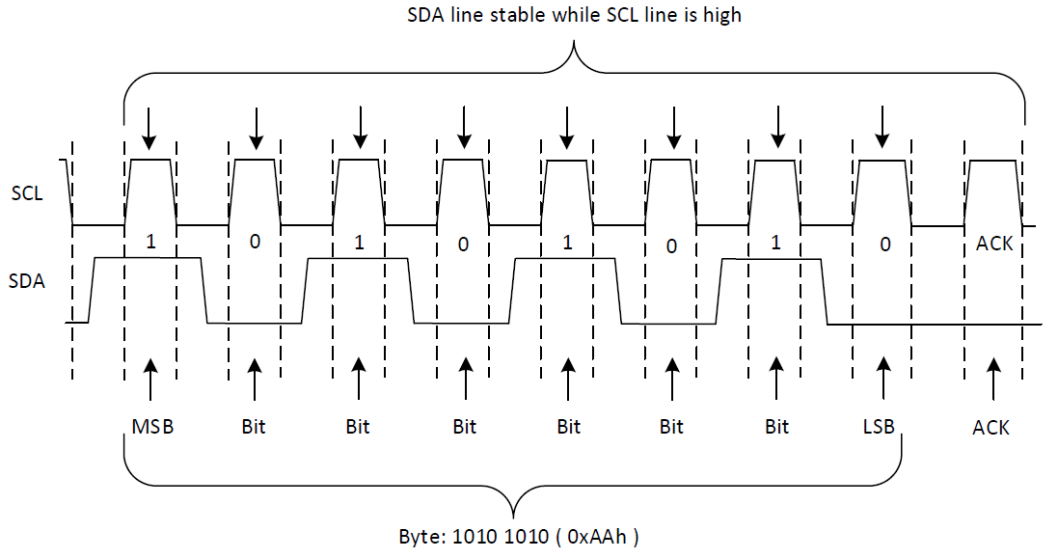
- I2C
  - Controller (master) communicates with slave devices
  - Slave may not transmit data unless addressed
  - Each device on the I2C bus has a specific address

  - Many slave devices require configuration upon startup to set behavior
  - Typically done through the slave's internal register maps
  - A device can have one or multiple registers where data is stored, written or read

Texas Instruments - Application Report SLVA704–June 2015 Understanding the I2C Bus

SW intro (I2C)

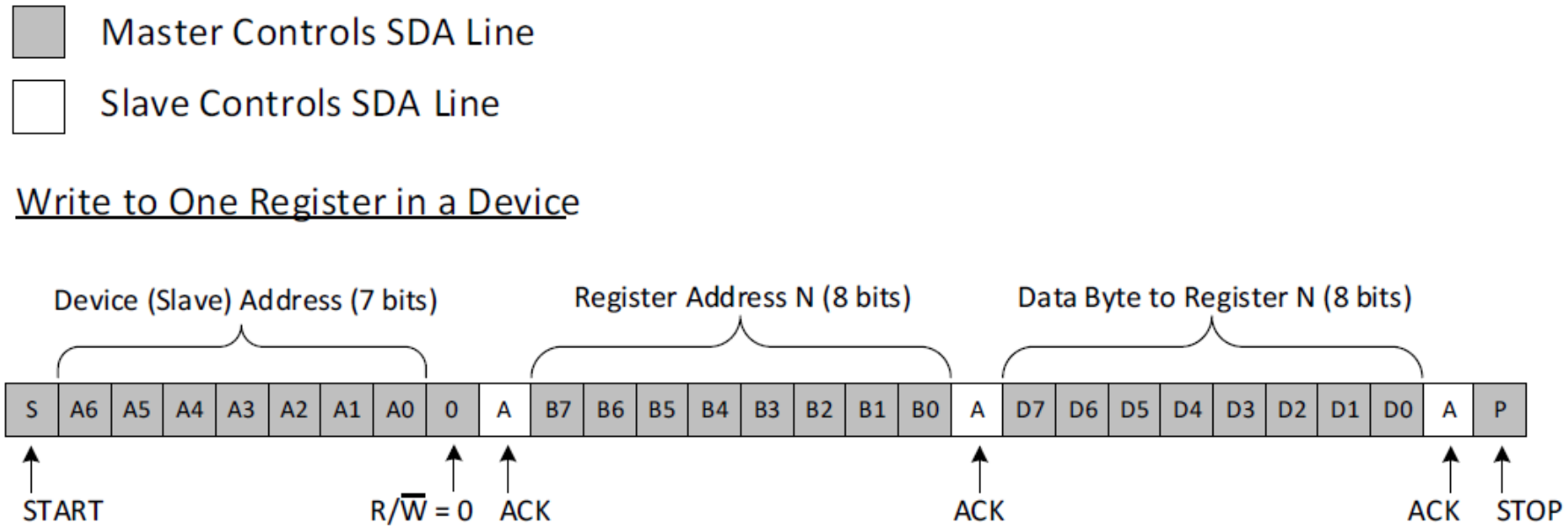




Texas Instruments - Application Report SLVA704–June 2015 Understanding the I2C Bus

SW intro (I2C)

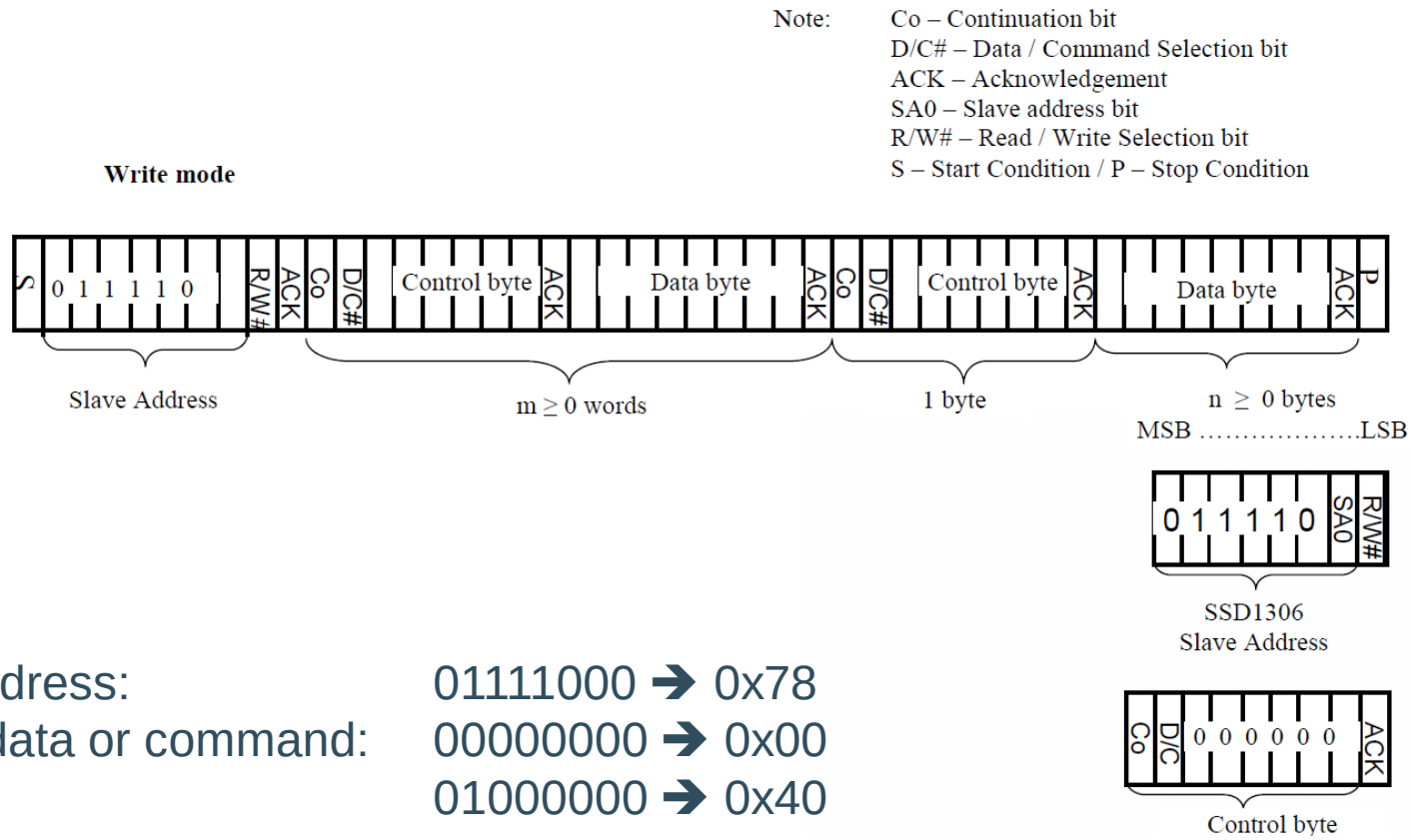
- I2C message



Texas Instruments - Application Report SLVA704–June 2015 Understanding the I2C Bus

SW intro (I2C)

- OLED screen - SSD1306 implementation: datasheet p. 20



Note:

I2C address: 01111000 ➔ 0x78

Write data or command: 00000000 ➔ 0x00

01000000 ➔ 0x40

SW intro (OLED)

- Now we know how to send data
- What should we send to do what? ➔ Command table (p. 28 – 32)

9 COMMAND TABLE

Table 9-1: Command Table

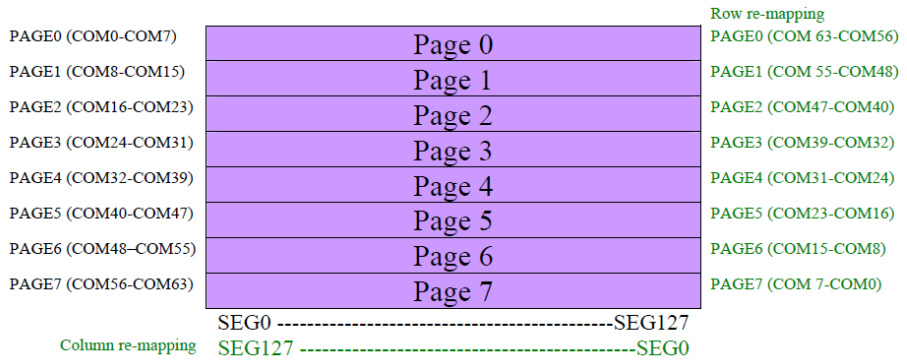
(D/C#=0, R/W#(WR#) = 0, E(RD#=1) unless specific setting is stated)

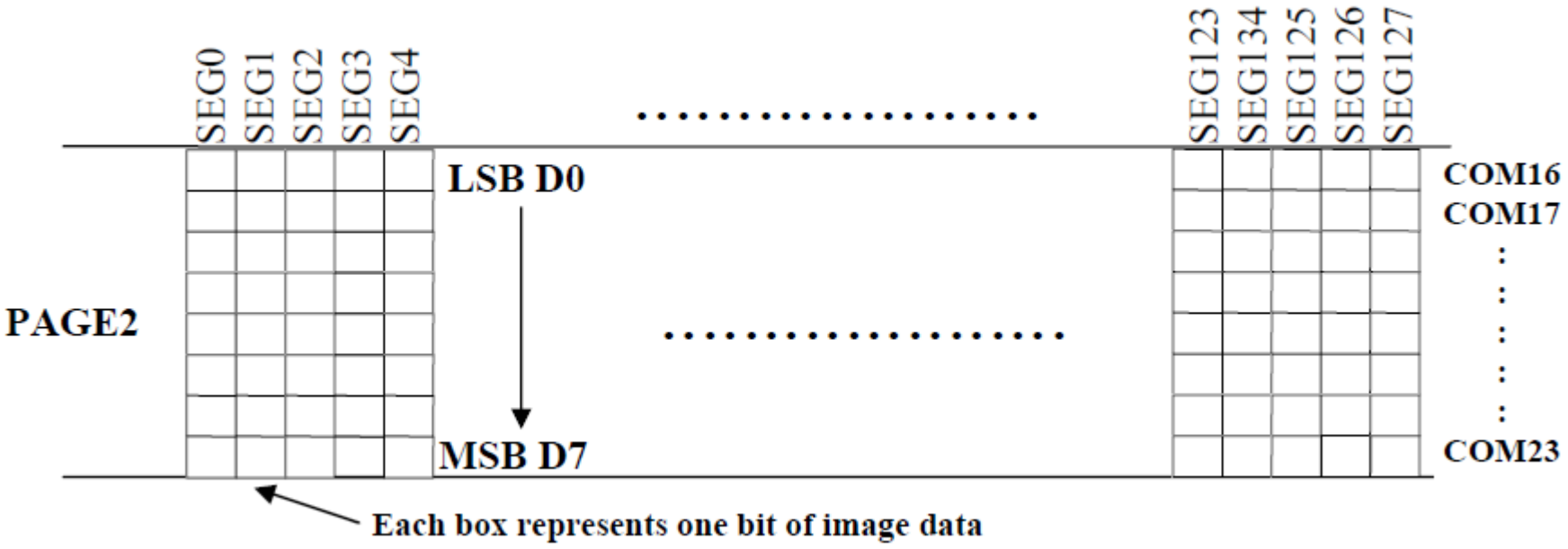
1. Fundamental Command Table											
D/C#	**Hex**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**Command**	**Description**
0 0	81 A[7:0]	1 A_7	0 A_6	0 A_5	0 A_4	0 A_3	0 A_2	0 A_1	1 A_0	Set Contrast Control	Double byte command to select 1 out of 256 contrast steps. Contrast increases as the value increases. (RESET = 7Fh)
0	A4/A5	1	0	1	0	0	1	0	X_0	Entire Display ON	A4h, X_0=0b: Resume to RAM content display (RESET) Output follows RAM content A5h, X_0=1b: Entire display ON Output ignores RAM content
0	A6/A7	1	0	1	0	0	1	1	X_0	Set Normal/Inverse Display	A6h, X[0]=0b: Normal display (RESET) 0 in RAM: OFF in display panel 1 in RAM: ON in display panel A7h, X[0]=1b: Inverse display 0 in RAM: ON in display panel 1 in RAM: OFF in display panel
0	AE AF	1	0	1	0	1	1	1	X_0	Set Display ON/OFF	AEh, X[0]=0b:Display OFF (sleep mode) (RESET) AFh X[0]=1b:Display ON in normal mode

- Different types: fundamental, scrolling, addressing, HW configuration, timing

SW intro (OLED)

- How does the screen work?





SW intro (OLED)

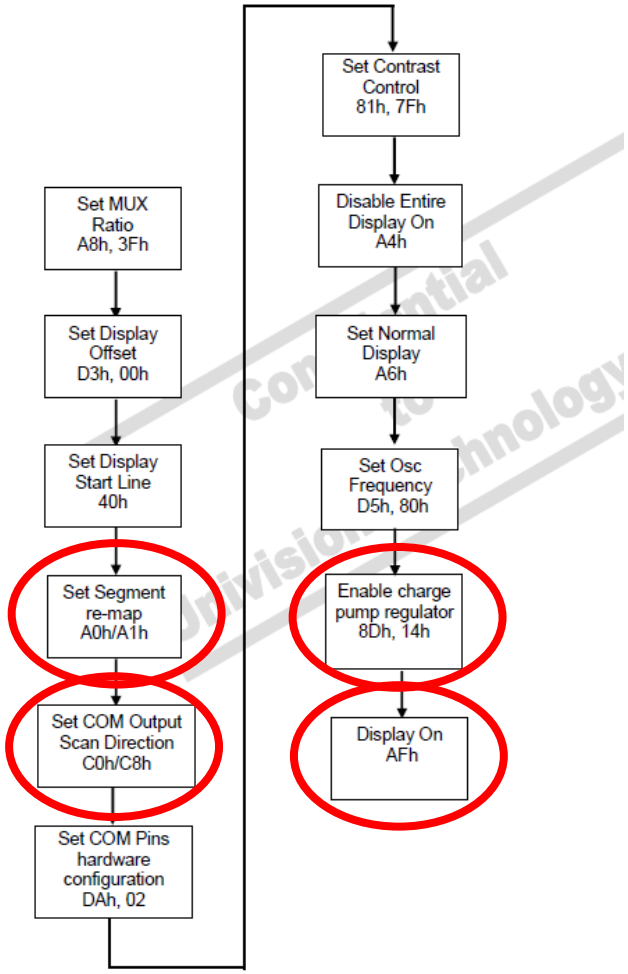
- Set address pointer for GDDRAM

3. **Addressing Setting Command Table**											
D/C#	**Hex**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**Command**	**Description**
0	00~0F	0	0	0	0	X_3	X_2	X_1	X_0	Set Lower Column Start Address for Page Addressing Mode	Set the lower nibble of the column start address register for Page Addressing Mode using X[3:0] as data bits. The initial display line register is reset to 0000b after RESET. **Note** (1) This command is only for page addressing mode
0	10~1F	0	0	0	1	X_3	X_2	X_1	X_0	Set Higher Column Start Address for Page Addressing Mode	Set the higher nibble of the column start address register for Page Addressing Mode using X[3:0] as data bits. The initial display line register is reset to 0000b after RESET. **Note** (1) This command is only for page addressing mode
0	B0~B7	1	0	1	1	0	X_2	X_1	X_0	Set Page Start Address for Page Addressing Mode	Set GDDRAM Page Start Address (PAGE0~PAGE7) for Page Addressing Mode using X[2:0]. **Note** (1) This command is only for page addressing mode

SW intro (OLED)

- Startup sequence

Figure 2 : Software Initialization Flow Chart



SW intro (OLED)

- Let's have a look how this is programmed in our Arduino IDE

- **Task 6:**
  - Open "*oled_driver.ino*" in Arduino IDE
  - Examine code, try to understand how it works
  - Use the loop function to:
    1. Initialize the screen
    2. Clear the screen
    3. Draw a one-pixel border around the screen
    4. Write your name in the middle of the screen, using:

       ```
       void ssd1306_char_f6x8(uint8_t x, uint8_t y, const char ch[])
       ```
    5. Optional: move or scroll it across the screen? Experiment yourself!

Overview of today

- 09:00 Introduction

- 09:30 PCB Design intro
- 09:45 PCB Soldering & electrical testing
- 10:45 PCB Test & bootloader
- 11:00 SW intro (OLED, I2C, ADCs, Interrupts)

Hardware

Software

- **12:00 Lunch break**

- 12:30 SW intro (OLED, I2C, ADCs, Interrupts)
- 13:30 Develop simple game (Snake)
- 15:30 Download finished games
- 16:00 End

Software

Overview of today

- 09:00 Introduction

- 09:30 PCB Design intro
- 09:45 PCB Soldering & electrical testing
- 10:45 PCB Test & bootloader
- 11:00 SW intro (OLED, I2C, ADCs, Interrupts)

Hardware

Software

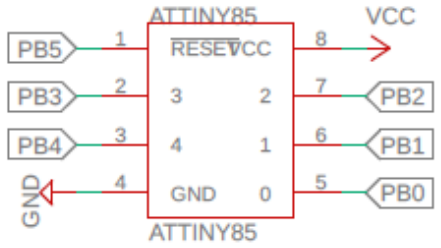
- 12:00 Lunch break

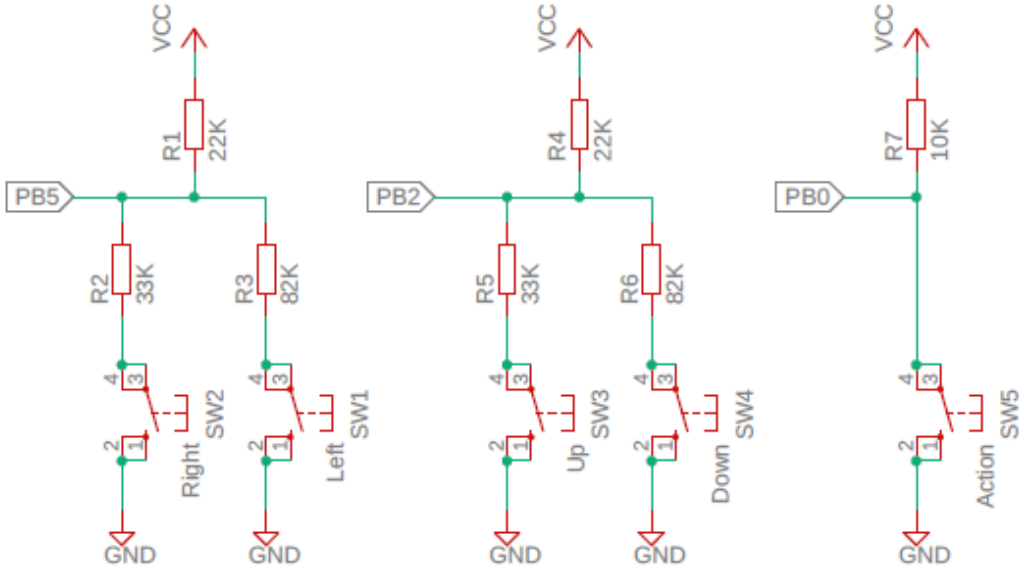
- **12:30 SW intro (OLED, I2C, ADCs, Interrupts)**
- 13:30 Develop simple game (Snake)
- 15:30 Download finished games
- 16:00 End

Software

SW intro (User Input - ADC)

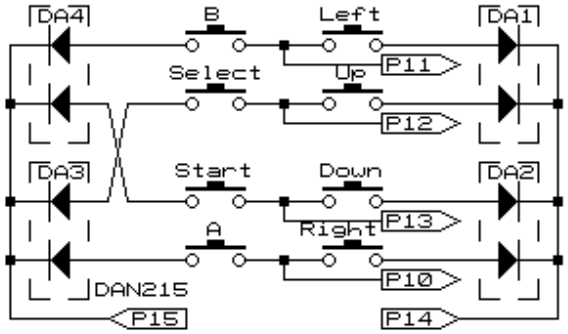
- Now we are going to read the user input buttons
- How many user IOs are there? How many pins on microcontroller?
- Schematic:

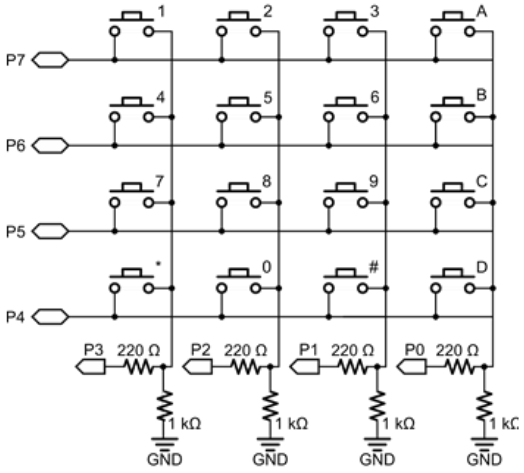


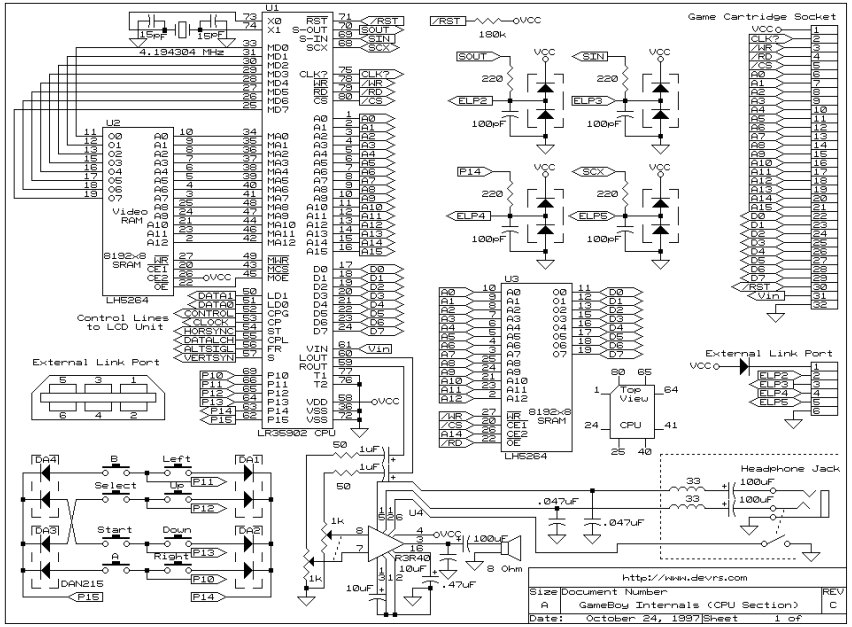


SW intro (User Input - ADC)

- We use a trick to reduce the required IO ports
  - Often used in hardware design

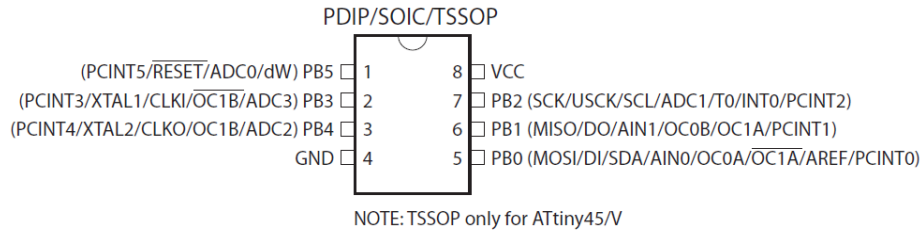






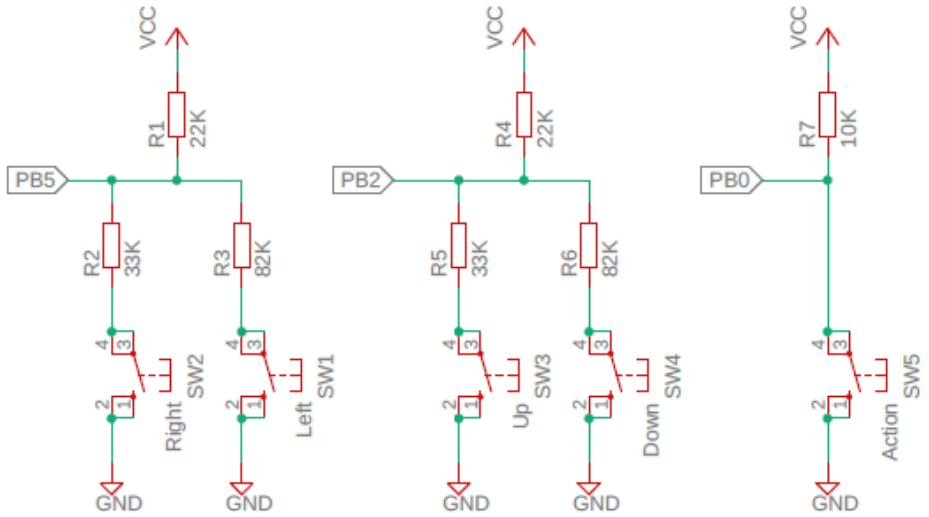
KU LEUVEN eavise

SW intro (User Input - ADC)

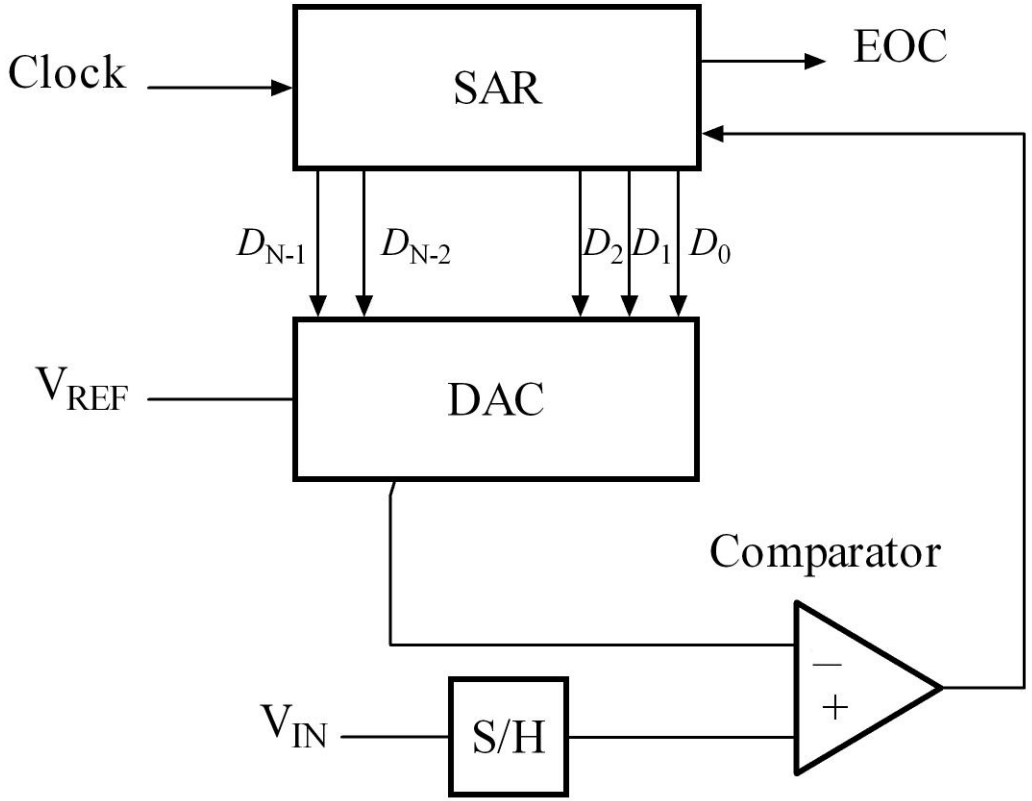


- We use different analog values through voltage dividers on the analog inputs of the ATTiny85
  - ADCs to determine button press!

- How do they work on the ATTiny85?
  - 4 channels, single 10-bit ADC
  - Successive approximation

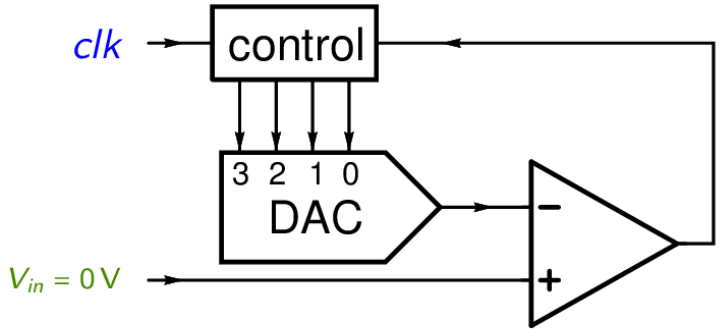


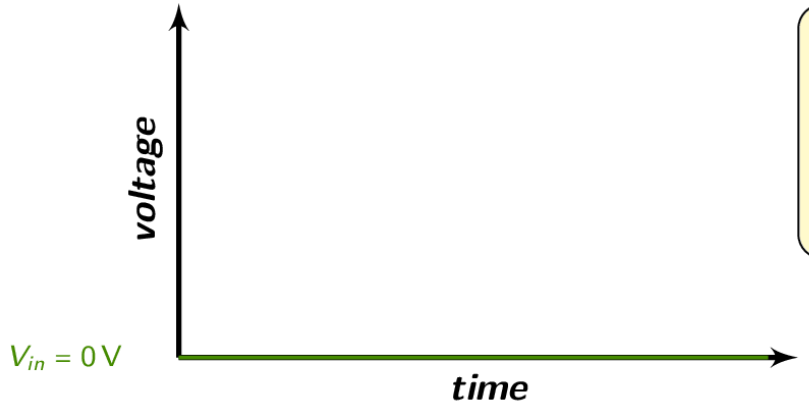
SW intro (User Input - ADC)



SW intro (User Input - ADC)

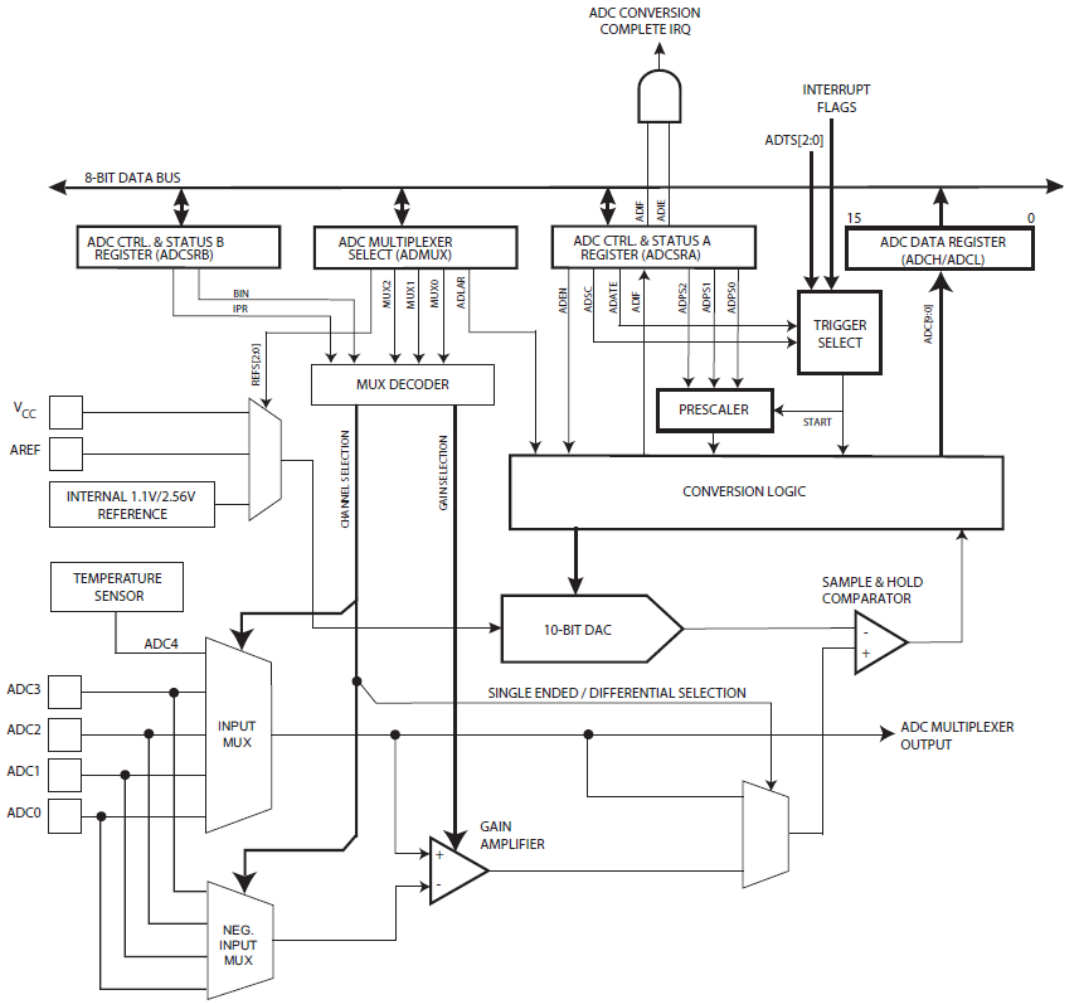
Successive Approximation – example of a 4-bit ADC





Resolution:	
5 V × 1/2	2.5000 V
5 V × 1/4	1.2500 V
5 V × 1/8	0.6250 V
5 V × 1/16	0.3125 V
...	
5 V × 1/1024	0.0049 V

SW intro (User Input - ADC)



SW intro (User Input - ADC)

- A lot of settings!
  - MUX, reference voltage, enables, pre-scaler, interrupts, how 10-bit is stored in 8-bit registers, mode of operation,…
  - 5 registers!

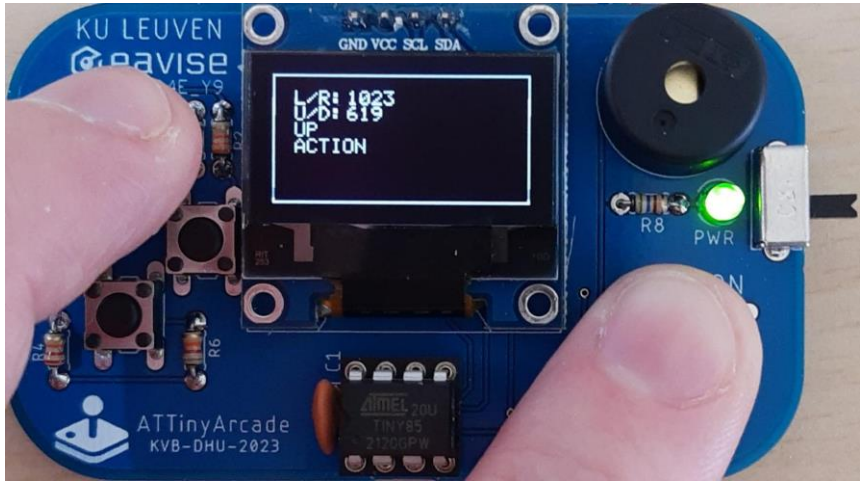
- Luckily, Arduino libraries perform lots of abstraction:

```
analogReadResolution(10) ;
analogReference(DEFAULT);
analogRead(A0);
```

SW intro (User Input - ADC)

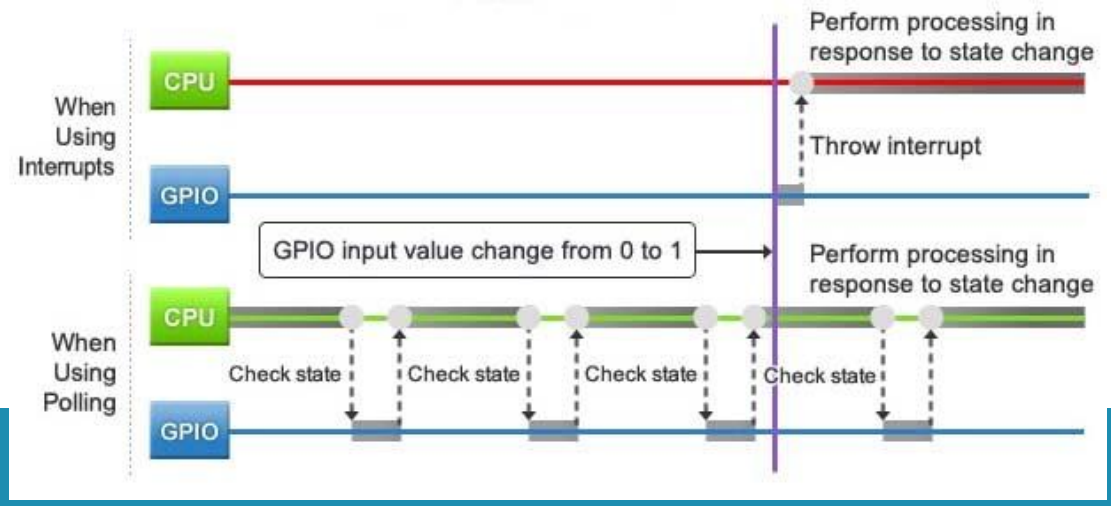
- **Task 7:**
  - Calculate the expected ADC values for each of the 4 buttons (up, down, left, right)
  - Open "*user_input.ino*" in Arduino IDE
  - Examine code, try to understand how it works
  - Extend the code to:
    1. Initialize the user IOs
    2. Show the ADC values on the OLED screen

       tip: `itoa();` to convert int to char
    3. Show which buttons were pressed



SW intro (Interrupts - optional)

- Reading buttons this way is not efficient
  - Sometimes OK, when SW routine is very predictable (e.g. game console)
- Interrupt the running code to do something else
- (Very) short routines (no delays)!
- Internal or external
- Examples: timer, watchdog, keyboard, sensor, switch



SW intro (Interrupts - optional)

- ACTION button is on PCINT0 pin
- Two types in ATTiny85
- INT0 triggers a **direct dedicated interrupt** routine
  - Rising or falling is set through register
- PCINT (**Pin Change INTerrupt**) triggers an interrupt routine if one of the PCINT pins are triggered. You have to check which pin and what change yourself in the interrupt routine

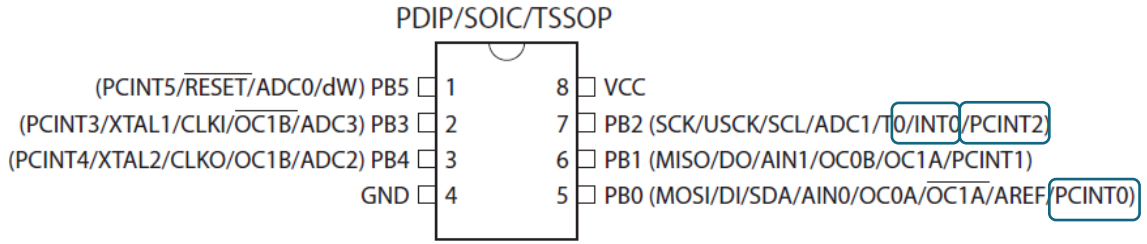


Table 9-1. Reset and Interrupt Vectors

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset
2	0x0001	INT0	External Interrupt Request 0
3	0x0002	PCINT0	Pin Change Interrupt Request 0
4	0x0003	TIMER1_COMPA	Timer/Counter1 Compare Match A
5	0x0004	TIMER1_OVF	Timer/Counter1 Overflow
6	0x0005	TIMER0_OVF	Timer/Counter0 Overflow

SW intro (Interrupts - optional)

- INT0

Bit	7	6	5	4	3	2	1	0	
0x3B	–	INT0	PCIE	–	–	–	–	–	GIMSK
Read/Write	R	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	BODS	PUD	SE	SM1	SM0	BODSE	ISC01	ISC00	MCUCR
	R	R/W	R/W	R/W	R/W	R	R/W	R/W	
	0	0	0	0	0	0	0	0	

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

```
attachInterrupt(digitalPinToInterrupt(PIN_NUMBER), routine, RISING);

void routine() {

        YOUR CODE HERE
}
```

Note: not supported by all ATTiny-cores

SW intro (Interrupts - optional)

- PCINT0_vect

Bit	7	6	5	4	3	2	1	0	
0x15	–	–	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 5:0 – PCINT[5:0]: Pin Change Enable Mask 5:0**

Each PCINT[5:0] bit selects whether pin change interrupt is enabled on the corresponding I/O pin.

Bit	7	6	5	4	3	2	1	0	
0x3B	–	INT0	PCIE	–	–	–	–	–	GIMSK
Read/Write	R	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Set registers yourself in setup() of Arduino code!

```
PCMSK |= bit (PCINT0);
GIMSK |= 0b00100000;
```

```
ISR(PCINT0_vect) {

    YOUR CODE HERE (check which pin and change)

}
```

SW intro (Interrupts - optional)

- Finally, enable all interrupts

Bit	7	6	5	4	3	2	1	0	
0x3F	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- `sei();` or `interrupts();`
- `cli();` or `noInterrupts();`

SW intro (Interrupts - optional)

- Let’s have a look how this is programmed in our Arduino IDE

- **Task 8 - optional:**
  - Extend “*user_input.ino”* with interrupt routine for the ACTION button
    - Uncomment the previous code for ACTION button read

Overview of today

- 09:00 Introduction

- 09:30 PCB Design intro
- 09:45 PCB Soldering & electrical testing
- 10:45 PCB Test & bootloader
- 11:00 SW intro (OLED, I2C, ADCs, Interrupts)

Hardware

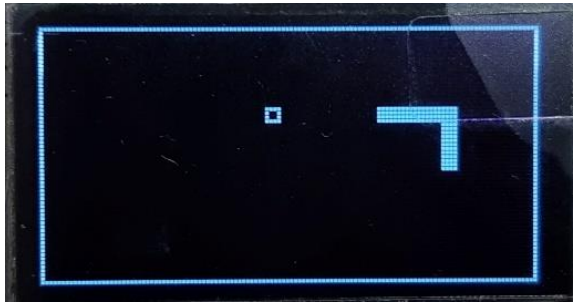
Software

- 12:00 Lunch break

- 12:30 SW intro (OLED, I2C, ADCs, Interrupts)
- **13:30 Develop simple game (Snake)**
- 15:30 Download finished games
- 16:00 End

Software

Develop simple Snake game



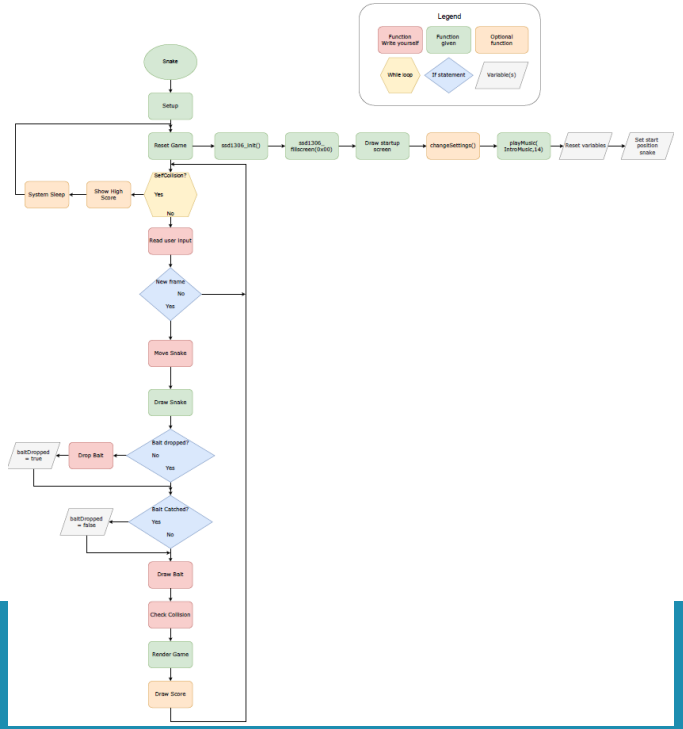
- **Final task 9:**
  - program a simple game: Snake

- Rules:
  - Snake moves in one of four directions at each new timeframe
  - Direction controlled by D-pad
  - Bait randomly positioned on screen
  - If bait is catched, length of the snake, score and speed increases
  - At borders, snake reappears at the opposite side of the game grid
  - Game over when snake collides with itself

Develop simple Snake game

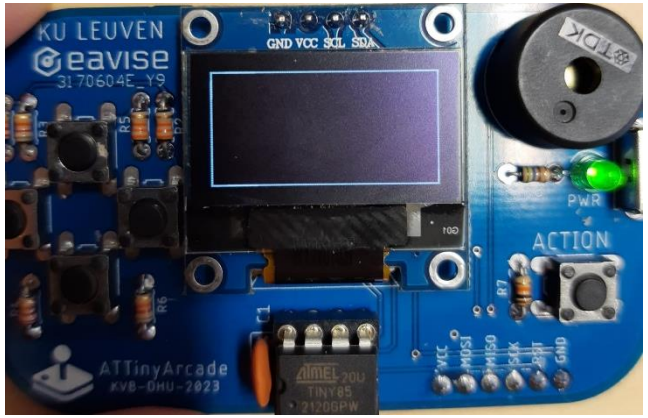
- How to start?
  - ➔Develop flowchart of game

- This has already been done for you! Let's have a look.



Develop simple Snake game

- A skeleton code framework has already been developed for you
- You need to write specific functions
  - Some are optional
- Let’s look at the Arduino code
- Download the skeleton code and run
  - Generates startup
  - Gives heartbeat and draws border



Develop simple Snake game

- *Render_game()* explanation
  - Game grid is 16 x 32

    Stored in *unsigned long screenBuffer[16];*

    = 4 bytes = 32 bits

  - Screen is 64 x 128

    Each game pixel is a 4 x 4 pixel on the OLED screen

Develop simple Snake game

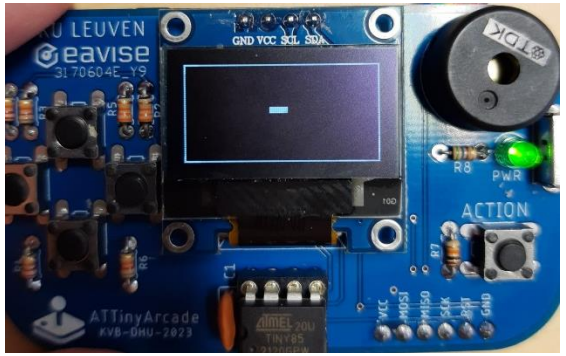
- *Render_game()* converts the *screenBuffer[16]* variable to OLED pixels

- You only need do two things:
  - Set a ‘1’ at each position in the *screenBuffer* for each segment of the **snake** and the single **bait**
  - Indicate the position of the bait with two variables
    - int baitX
    - int baitY

  - **Note: allowed snake and bait positions are X: 0-31 and Y: 0-15**

Develop simple Snake game

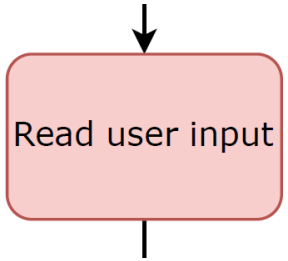
- **Let's try this:**
  - Uncomment *draw_snake()* and upload



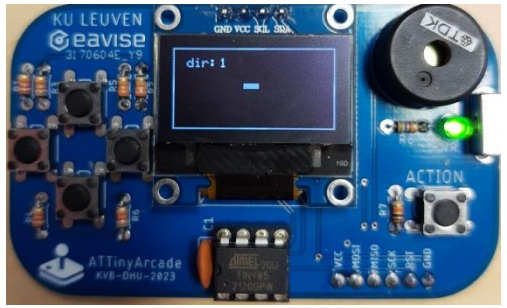
- Now it's up to you!

- **Task 9 begins:** you are now going to implement the missing functions yourself
  - One by one: upload after each (partly) adjusted function
  - We'll walk you through each function (don't worry ☺)

Develop simple Snake game (1)

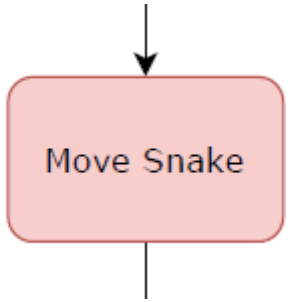


- First, define a global variable to hold the current **dir**ection
  - Byte: 0: up, 1: left, 2: down, 3: right
  - Don’t forget to reset this variable in `resetGame();`
- Function checks if U/D/L/R was pressed, and changes the direction if needed
  - Move in opposite direction not allowed!
- To test, print the current direction as text on the screen



- **Upload to test if it works! ➔ any remaining issues?**

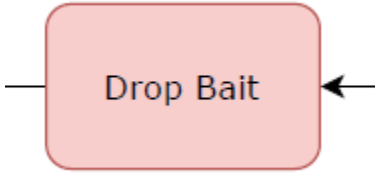
Develop simple Snake game (2)



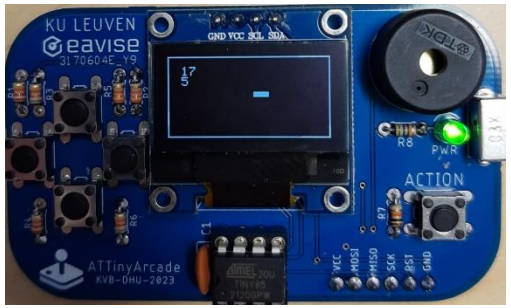
- Move each segment of the snake in the right direction
- Use variables *xPos[]*, *yPos[]* and *len*
- Tip: use case-statement since direction is unambiguous
- Don’t forget the borders of the screen!
- Note: valid xPos = 0 – 31, valid yPos = 0-15

- **Upload!** The snake should now move over the game grid

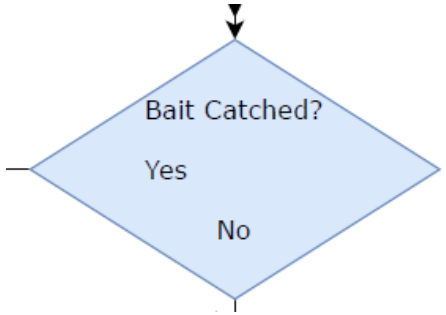
Develop simple Snake game (3)



- Generate random positions for a new bait (preferably not on the edges)
  - baitX: 1 – 30, baitY: 1 – 14
- Test if bait is not dropped on snake itself!
- Print baitX and baitY somewhere on the screen as test for now



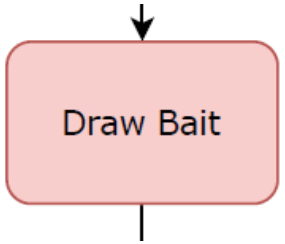
Develop simple Snake game (4)



- See if the snake catched the bait with its head
- Return Boolean *true* or *false*
- Increment *score* and *len* in this function
- Don’t forget to limit the *len* to *maxLen*
- You can play a sound if the bait is catched

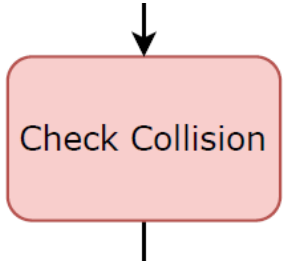
  playMusic(BaitMusic, 4);
- Do not upload yet, first fix next function

Develop simple Snake game (5)



- Draw the bait in the screen buffer
- Set a ‘1’ at the correct location in the buffer
- Remember: *render_game()* knows the difference between bait and snake based on baitX and baitY
- Use function *draw_snake()* as example
- Extremely short function
- **Upload and test! You should be able to play the game (without collision detection)**

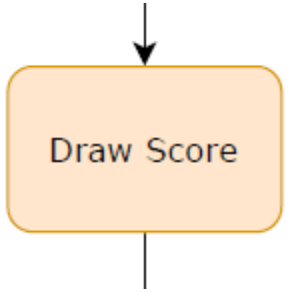
Develop simple Snake game (6)



- See if the head of the snake collided with a segment of itself
- Return Boolean *true* or *false*

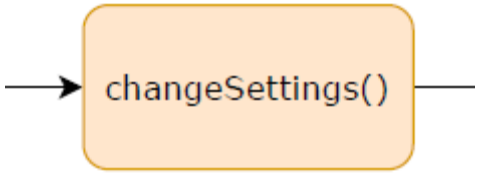
Develop simple Snake game

- That’s it! The game is finished. Well done.
- We have some optional functions left **(simple – intermediate – advanced expert):**



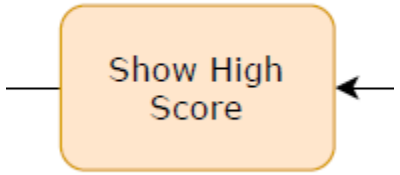
- **Simple!** Draw the current score each frame in one of the corners of the screen

Develop simple Snake game



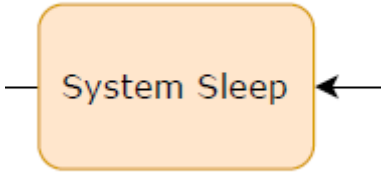
- **Intermediate!**
- At startup of the game see if e.g. UP was held pressed for specific time (e.g. 2 seconds)
- Mute or unmute sound if this was the case
  - Through global variable
  - Use as condition in *beep*-function
- Display *mute* or *sound* on top of the screen at startup

Develop simple Snake game



- **Advanced!** Use EEPROM to store previous high score
- After game over, load the high score from EEPROM
- Compare with current score
  - If higher, display on screen and write new high score to EEPROM
- Tips: EEPROM.read(), EEPROM.write()
  - EEPROM is read/written per byte. Int is 2 bytes long

Develop simple Snake game



- **Expert!**
- After game over, put ATTiny85 in sleep mode
- Wake up after interrupt (PCINT0)
- Steps: clear and switch off OLED screen, switch ADC off, then enable sleep mode and sleep. On power up from sleep switch ADC and OLED screen back on

Develop simple Snake game

- **Develop your own variations!**
  - Game over when you go out of the screen?
  - You could for example use the ACTION button to let the snake jump over itself!

Overview of today

- 09:00 Introduction

- 09:30 PCB Design intro
- 09:45 PCB Soldering & electrical testing
- 10:45 PCB Test & bootloader
- 11:00 SW intro (OLED, I2C, ADCs, Interrupts)

Hardware

Software

- 12:00 Lunch break

- 12:30 SW intro (OLED, I2C, ADCs, Interrupts)
- 13:30 Develop simple game (Snake)
- **15:30 Download finished games**
- 16:00 End

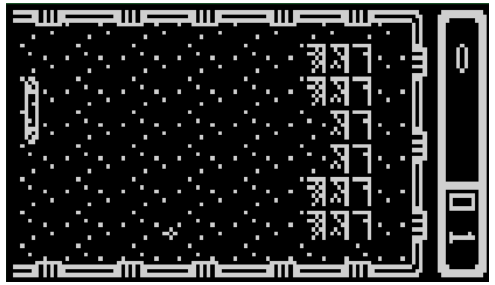
Software

Download finished games

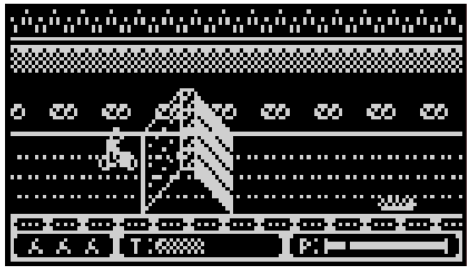
- Feel free to download any of the following finished games, found in the finished_games folder:

Q*bert (1982)

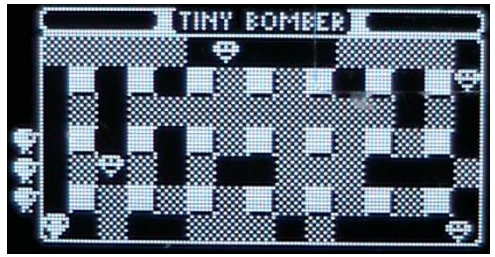
Arkanoid (1986)



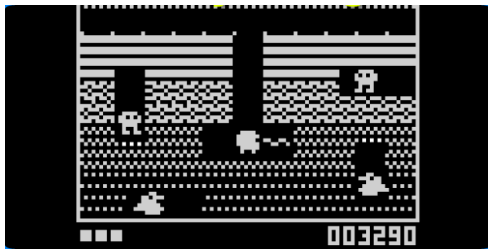
Excite Bike (1984)



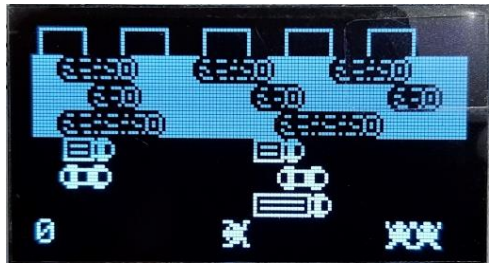
Bomberman (1983)



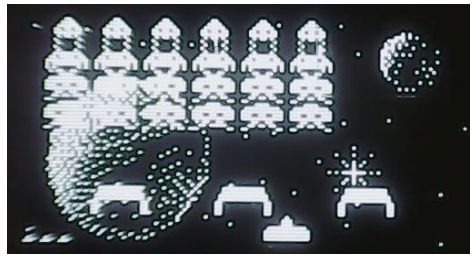
Dugger (1988)



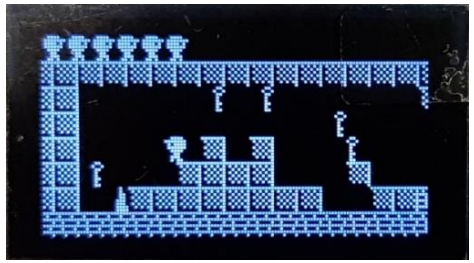
Frogger (1981)



Space Invaders (1978)



Tiny Gilbert (platformer)



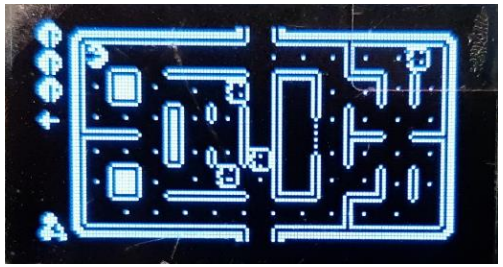
Snake (1976)

Introduction

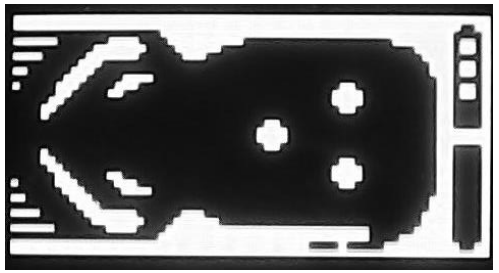
- Feel free to download any of the following finished games, found in the finished_games folder:

Morpion (tic-tac-toe)

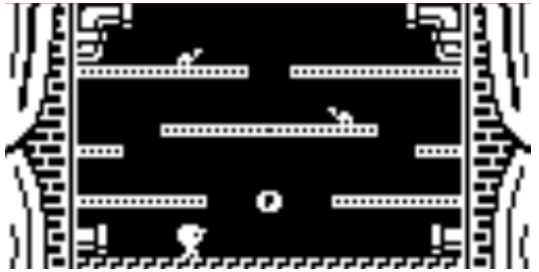
Pacman (1980)



Pinball (1984)

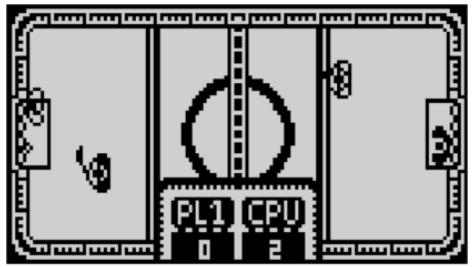


Pipeline (1978)

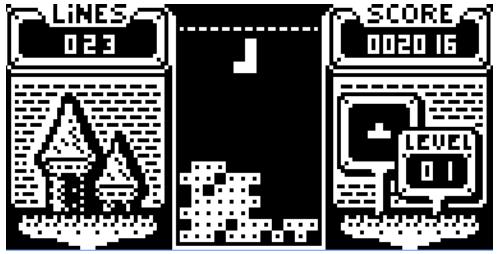


Plaque Attack (1983)

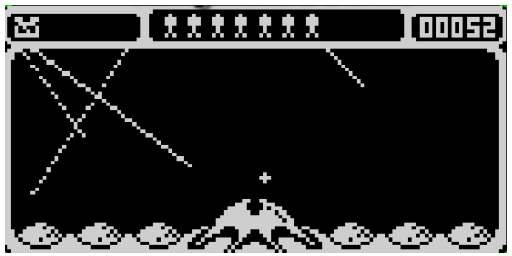
Hat Trick (1988)



Tetris (1984)



Missile Command (1980)



Download finished games

- A few online sources for other games:
  - https://github.com/andyhighnumber/Attiny-Arduino-Games
  - https://github.com/webboggles/AttinyArcade
  - https://www.tinyjoypad.com/tinyjoypad_attiny85
  - https://www.tinyjoypad.com/arduboy
  - …

➔ Some require code modification or additional libraries (e.g. different pins for OLED, coded for fewer buttons, etc.)

Overview of today

- 09:00 Introduction

- 09:30 PCB Design intro
- 09:45 PCB Soldering & electrical testing
- 10:45 PCB Test & bootloader
- 11:00 SW intro (OLED, I2C, ADCs, Interrupts)

Hardware

Software

- 12:00 Lunch break

- 12:30 SW intro (OLED, I2C, ADCs, Interrupts)
- 13:30 Develop simple game (Snake)
- 15:30 Download finished games
- **16:00 End**

Software

The end

- We hope you enjoyed this workshop!