Practical Exercise

STM32F4 Discovery

Alessandro Palla
alessandro.palla@for.unipi.it

Outline

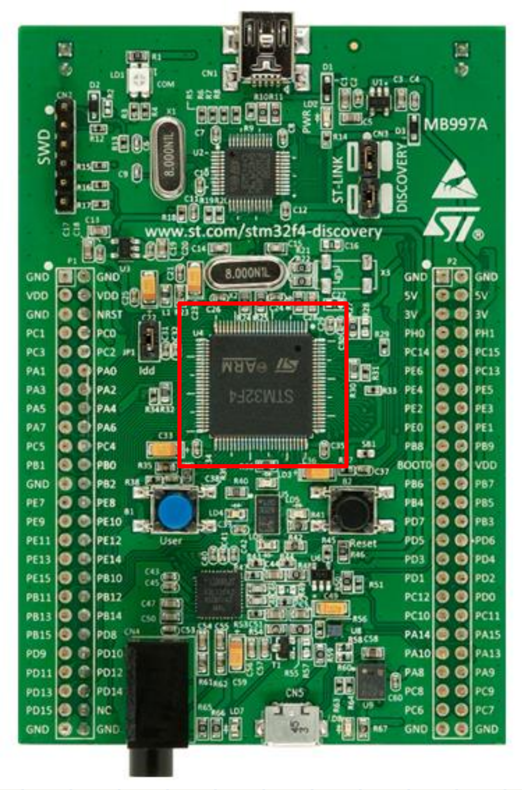
- STM32F4 Discovery
- Application: USB Mouse with accelerometer
- Hardware Configuration
  - Requirements
  - Peripherals Selections
    - Timer
    - GPIO
    - SPI
    - USB
  - Board Pinout
  - Clock Selection
  - Peripheral Configuration

Outline

- Software Design
  - PWM LEDs control
  - Interrupt management
  - Accelerometer Theoretical Background
    - LIS3DSH SPI communication
  - USB
- Put All together
- Conclusion

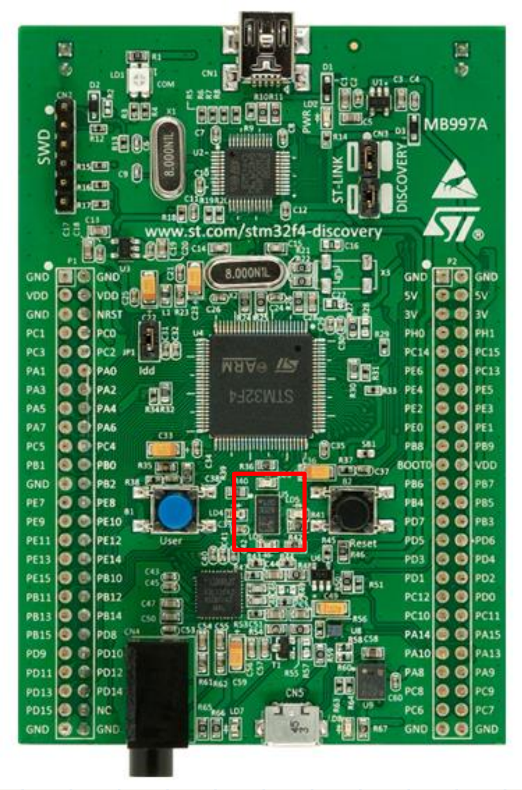
STM32Discovery F4

- STM32F407x Cortex-M4F core, 1MB flash, 192KB RAM, frequency up to 168 MHz



STM32Discovery F4

- STM32F407x Cortex-M4F core, 1MB flash, 192KB RAM, frequency up to 168 MHz
- 3-axis accelerometer

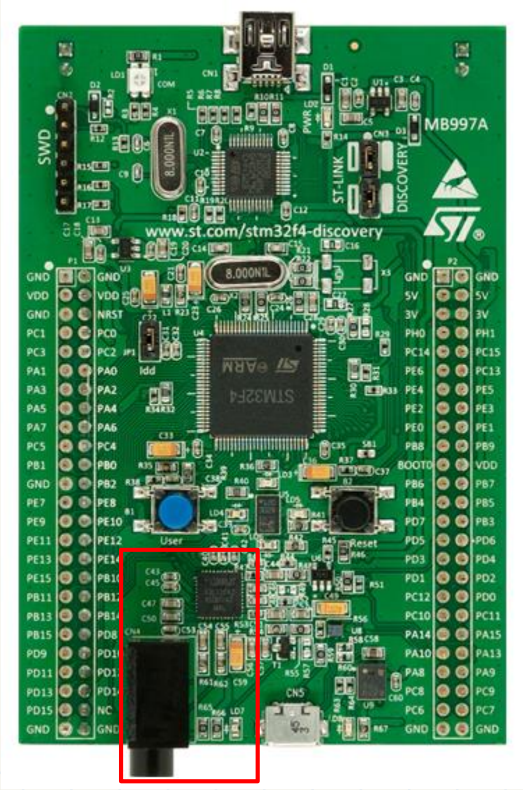


STM32Discovery F4

- STM32F407x Cortex-M4F core, 1MB flash, 192KB RAM, frequency up to $168\ MHz$
- 3-axis accelerometer
- Omnidirectional MEMS Microphone

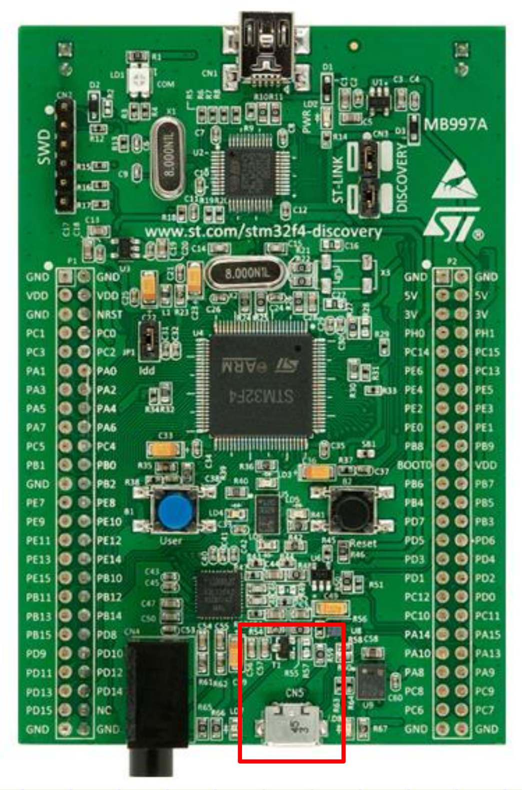
STM32Discovery F4

- STM32F407x Cortex-M4F core, 1MB flash, 192KB RAM, frequency up to 168 MHz
- 3-axis accelerometer
- Omnidirectional MEMS Microphone
- Audio DAC with class D amplifier



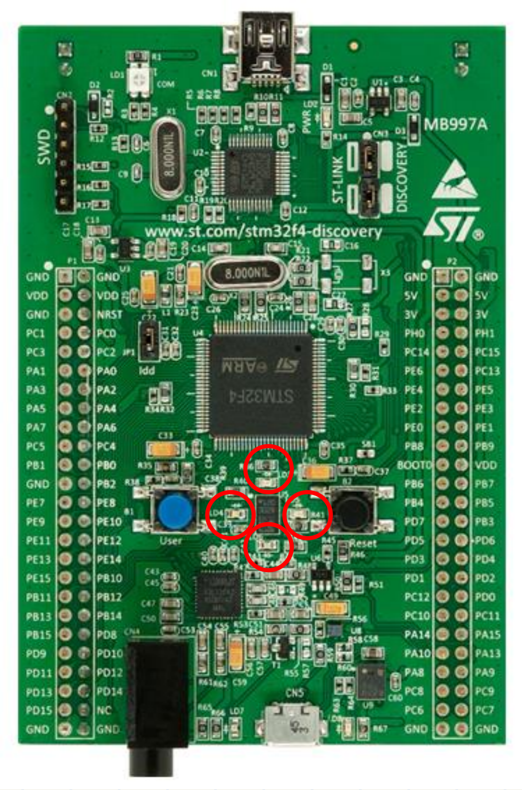
STM32Discovery F4

- STM32F407x Cortex-M4F core, 1MB flash, 192KB RAM, frequency up to 168 MHz
- 3-axis accelerometer
- Omnidirectional MEMS Microphone
- Audio DAC with class D amplifier
- USB FS (Full Speed) with micro-USB connector



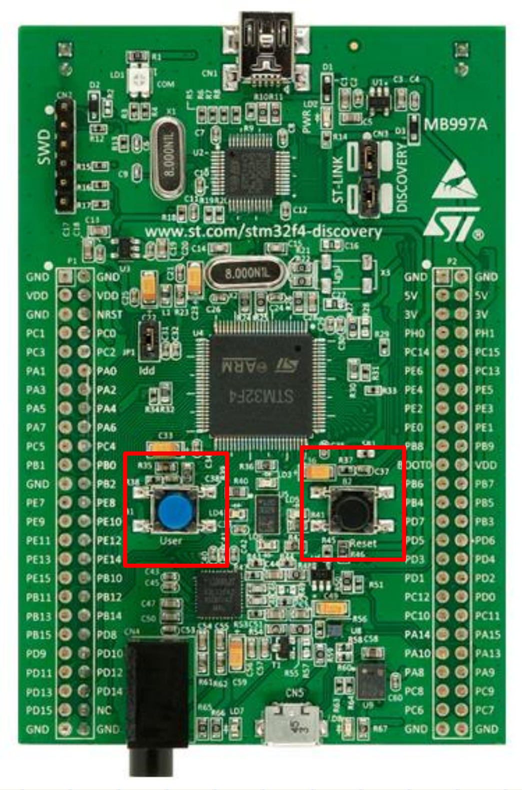
STM32Discovery F4

- STM32F407x Cortex-M4F core, 1MB flash, 192KB RAM, frequency up to 168 MHz
- 3-axis accelerometer
- Omnidirectional MEMS Microphone
- Audio DAC with class D amplifier
- USB FS (Full Speed) with micro-USB connector
- 4 user led (green, blue red and orange)



STM32Discovery F4

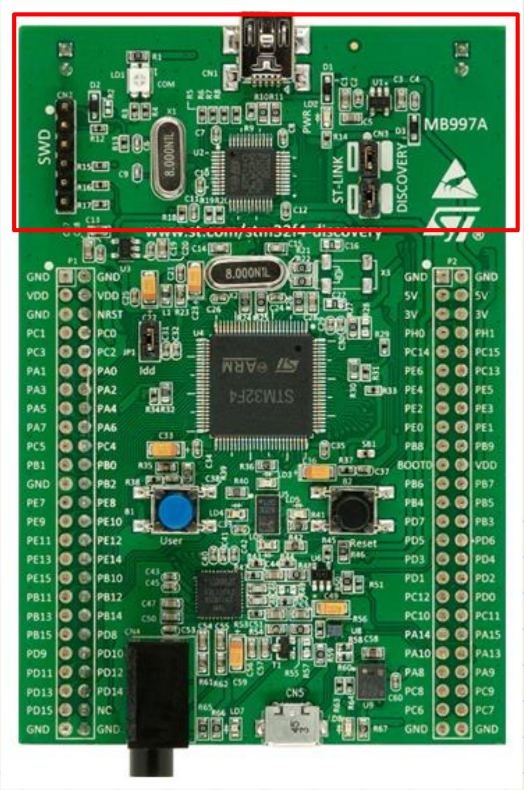
- STM32F407x Cortex-M4F core, 1MB flash, 192KB RAM, frequency up to 168 MHz
- 3-axis accelerometer
- Omnidirectional MEMS Microphone
- Audio DAC with class D amplifier
- USB FS (Full Speed) with micro-USB connector
- 4 user led (green, blue red and orange)
- One user button and one reset



STM32Discovery F4

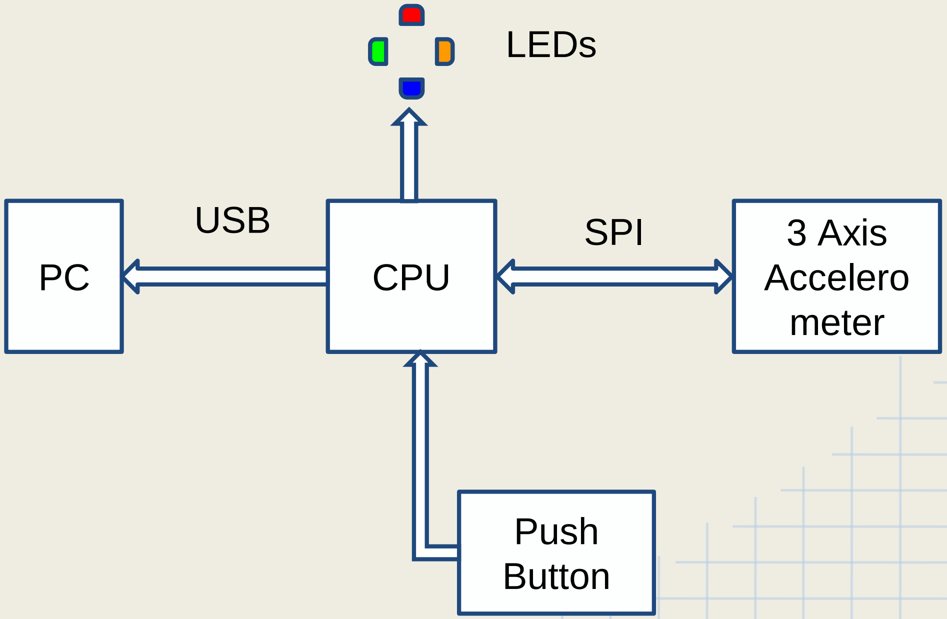
- STM32F407x Cortex-M4F core, 1MB flash, 192KB RAM, frequency up to 168 MHz
- 3-axis accelerometer
- Omnidirectional MEMS Microphone
- Audio DAC with class D amplifier
- USB FS (Full Speed) with micro-USB connector
- 4 user led (green, blue red and orange)
- One user button and one reset

- Programmation via USB with ST Link (chip above)



System Architecture

- Single Button USB Mouse
- Controlled by tilts on y and x axes
- Latency = $10\ ms$
  - It means that CPU has to polls accelerometer each millisecond
- Visual feedback using pulsing led.
  - Pulse frequency = $10\ Hz$



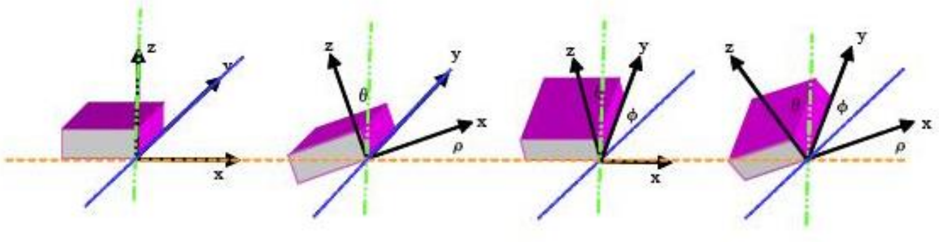
How it Works

- Tilt is measured by projection of g (gravity acceleration vector) on x and y axes (A_x, A_y)
- If A_x, A_y << A_z

$$\rho \approx \frac{A_x}{A_z} \approx \frac{A_x}{g}$$

$$\phi \approx \frac{A_y}{A_z} \approx \frac{A_y}{g}$$

- For low deviation acceleration of x and y axes are proportional to tilts



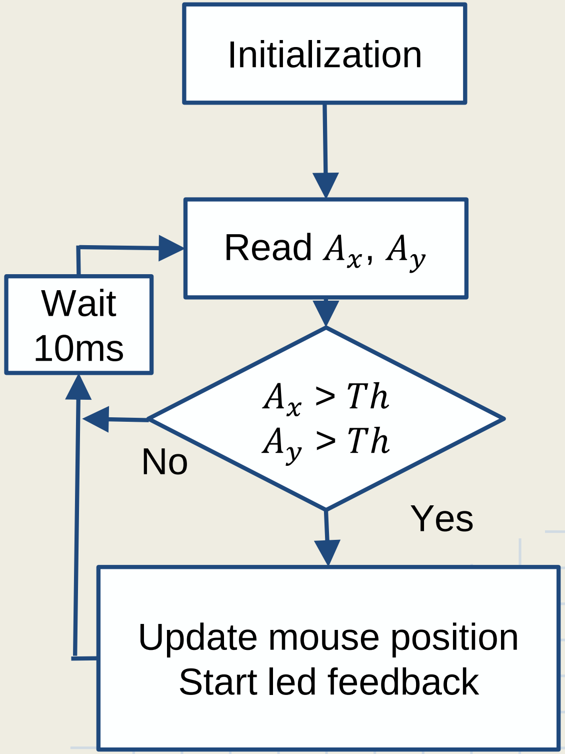
Three Axis for Measuring Tilt

$$\rho = \arctan\left(\frac{A_X}{\sqrt{A_Y^2 + A_Z^2}}\right)$$

$$\phi = \arctan\left(\frac{A_Y}{\sqrt{A_X^2 + A_Z^2}}\right)$$

System flow

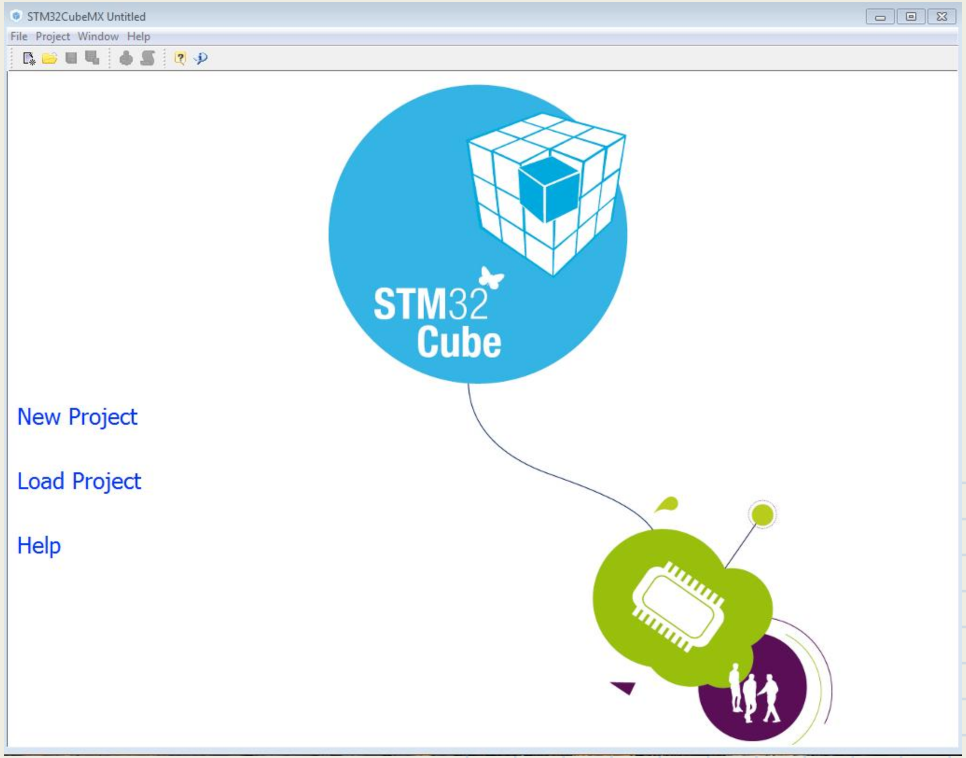
- If A_x or A_y is greater than a threshold, firmware update mouse cursor position
- Leds blink depending on tilts direction



STM32Cube MX

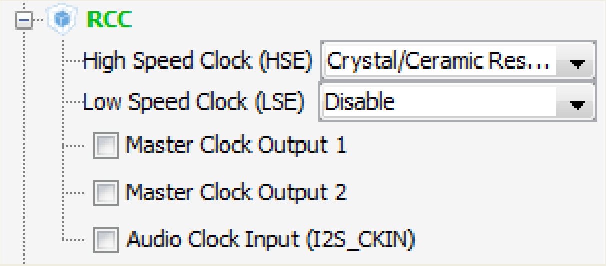
- Developing starting point
- GPIO configuration
- Peripherals selection
- Clock management
- Peripherals and middleware configuration
- Power Calculator

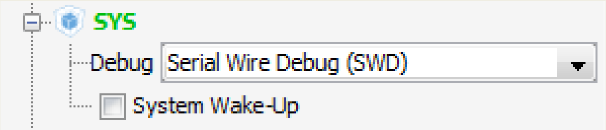
- Big number of library (USB Host and Device, TCP/IP Stack, SSL, FAT FileSystem, FreeRTOS operative system)



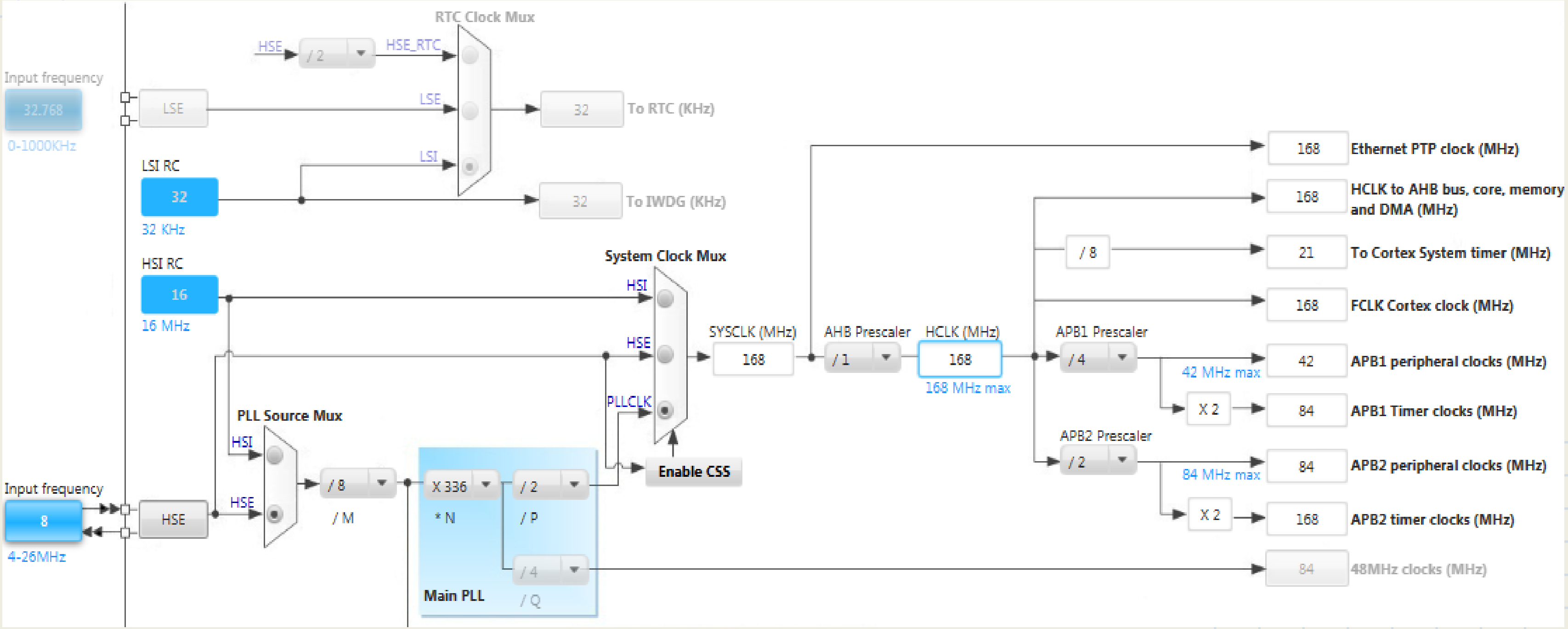
First Step: Debug and Clock

- RCC (Real-Time Clock Control), HSE (High speed clock) connected to $8\ MHz$ Crystal
- STLink connected via SWD (Serial Wire Debug) , a simplified JTAG





Clock Configuration



Clock Configuration

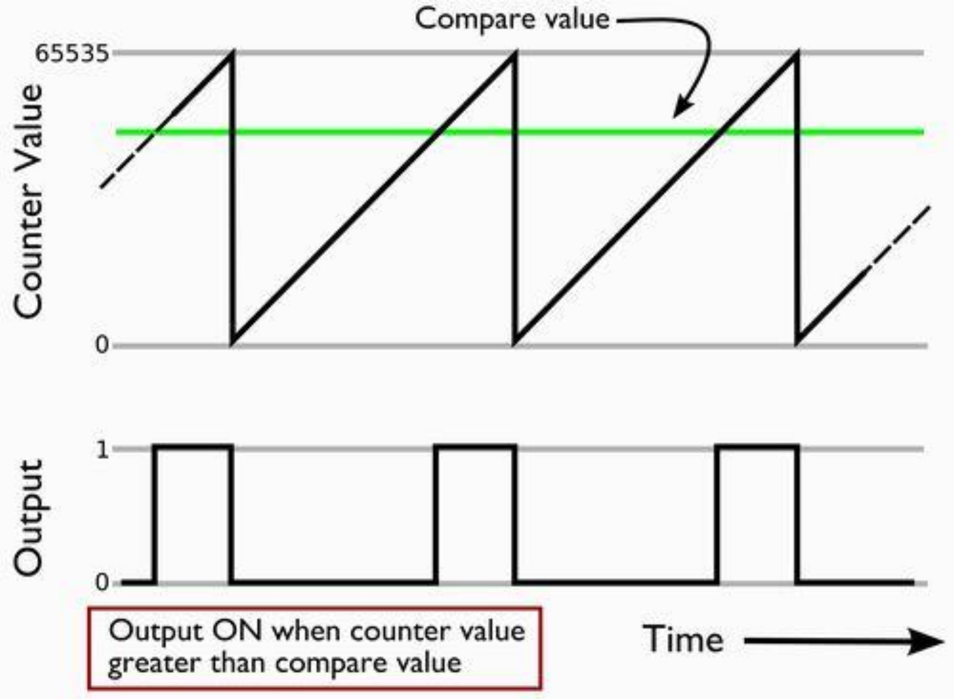
- Input Crystal Frequency: $8\ MHz$
- Crystal is more accurate than HSI (High Speed Internal oscillator), so is suggested to use it in order to improve performances.

- PLL (Phased Locked Loop) is an electronic system that can increase frequency of signals. In this case $f_{out} = f_{in} \frac{N}{M * P}$
- System Clock and HCLK (AHB Bus clock) are set to $168\ MHz$
- Each APB bus has a different clock speed: $42\ MHz$ APB1, $84\ MHz$ APB2
- Indeed timer has greater frequency: $84\ MHz$ APB1, $168\ MHz$ APB2

Pulse Width Modulation

- Adjusting duty cycle of signal we can control it’s average
- If digital signal’s frequency is greater than system bandwidth, we can approximate output with signal mean

- Can be used also to generate a fixed duty cycle waveform

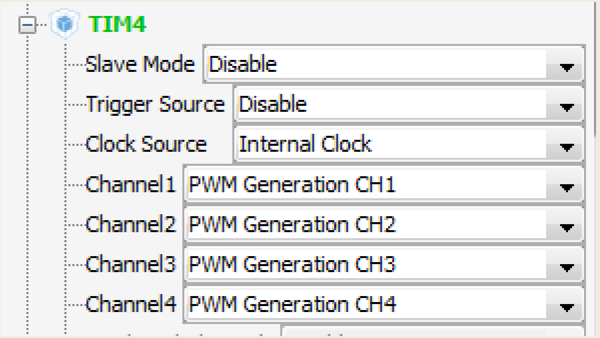


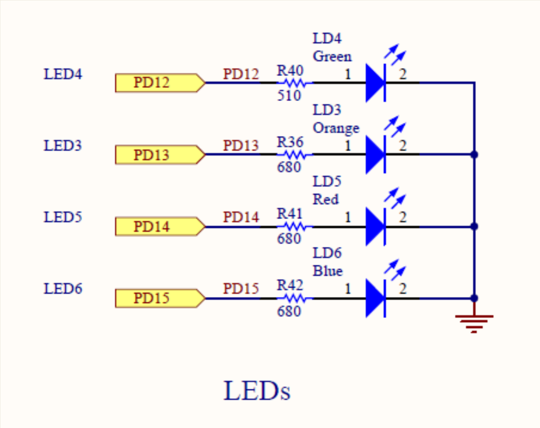
Pulse Width Modulation

$$f_{PWM} = \frac{f_{Timer}}{COUNTER\_MAX} = 10\ Hz$$

$$\delta\ (Duty\ Cycle) = \frac{OCR}{COUNTER_{MAX}} = \ 50\%\ \text{(square wave)}$$

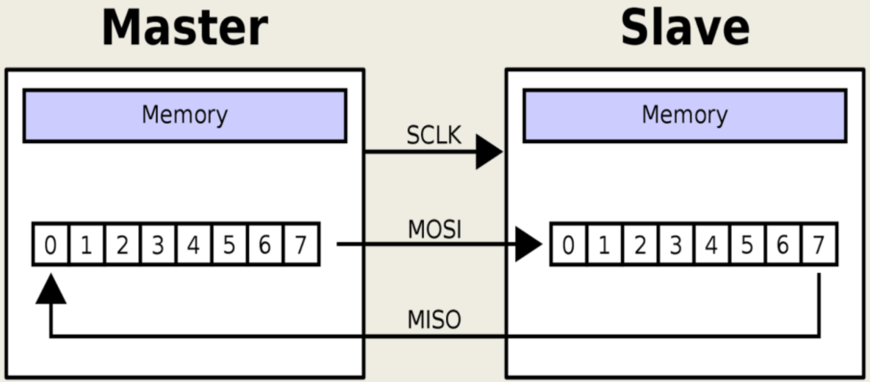
- All Timer4 channels are connected to LEDs
- Timer4 is connected to APB1 ($84\ MHz$)

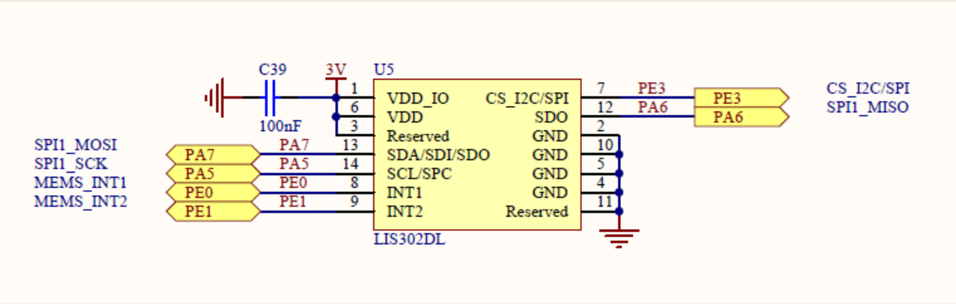


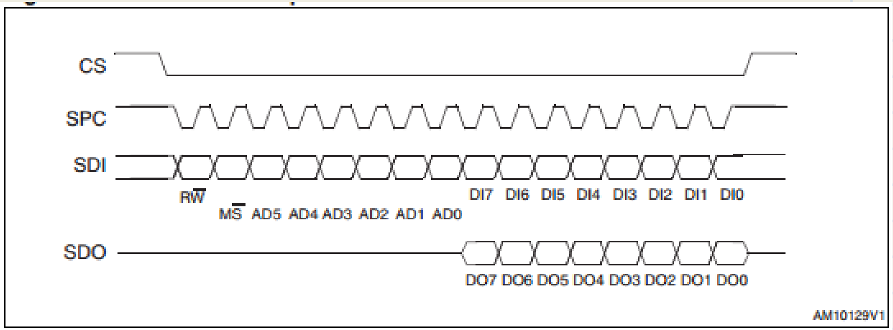


Serial Peripheral Interface

- Full Duplex synchronous serial data link
- MOSI: Master Out Slave IN
- MISO Master IN Slave Out







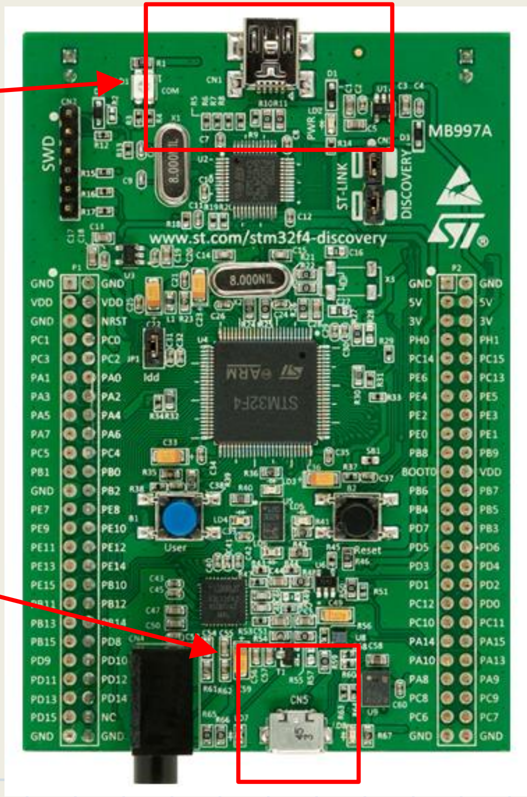
USB

- Asymmetric communication: one Host and multiple Devices (up to 127)
- USB can supply embedded devices:
  - Voltage supply: 5V
  - Current: up to 1A
  - Power up to 5W, enough for most embedded devices
- First version of standard (1996): USB 1.0, speed 1.5 Mbit/s
- USB 1.1 introduces USB FS (Full Speed), speed 12 Mbit/s
- USB 2.0 => USB HS (High Speed), theoretical speed of 480 Mbit/s

- STM32F4
  - Can be Host, Device or OTG (on-the-go, can switch between Host and Device)
  - It supports both USB FS and HS,

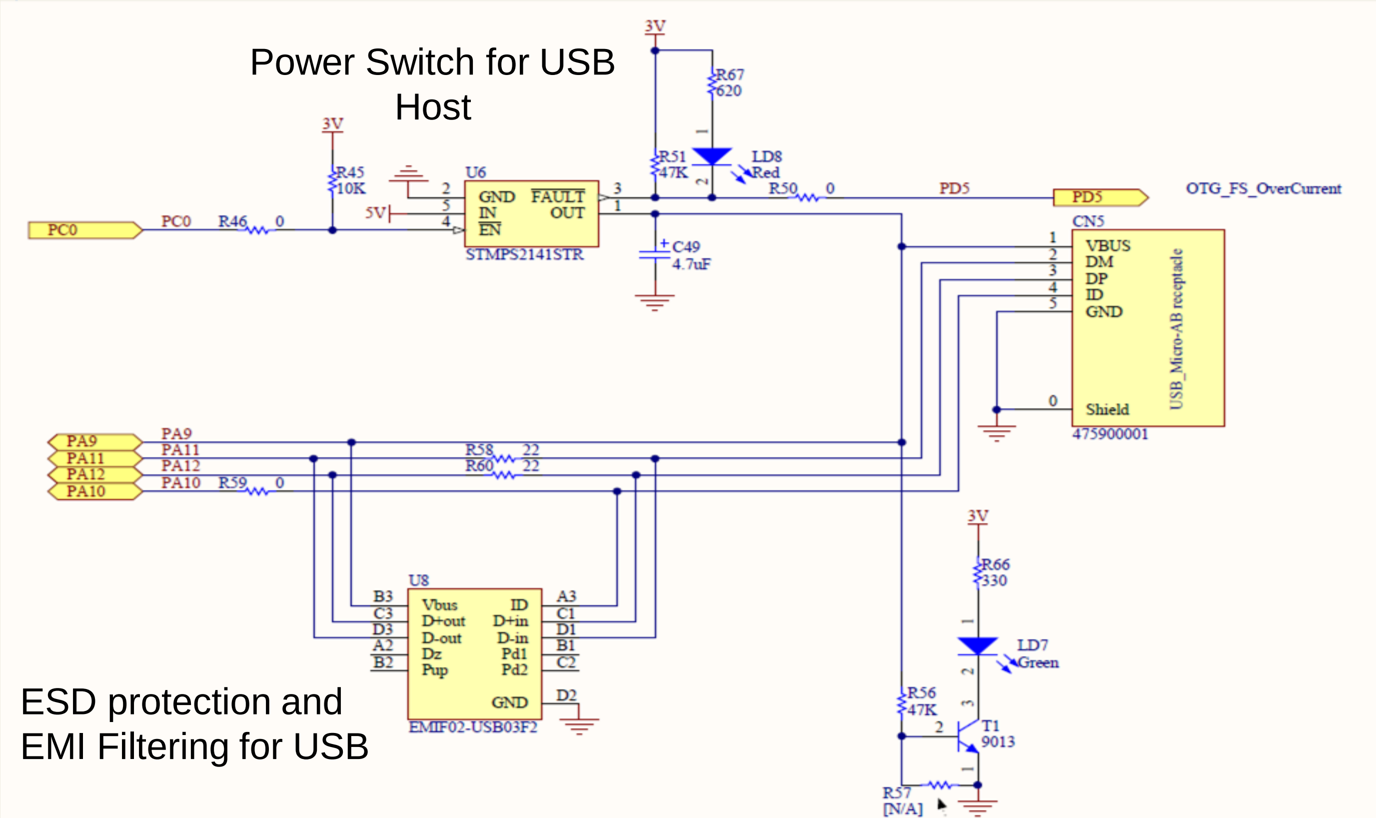
STM32F4 Discovery USB

- Mini USB connector for power supply, debugging and programming CPU
- Micro USB connector for communication
- In our application:
  - USB Device, PC is the Host
  - We don’t need high speed => USB FS



Discovery USB Schematic

Power Switch for USB Host



ESD protection and EMI Filtering for USB

USB HID

- Human Interface Device, a class of USB specification designed to interacts directly with humans
- A lot of devices can use this technology:
  - Mice
  - Keyboards
  - Game Controller
  - Custom Device (driver developing on host side)
- Also other standards have HID class, like Bluetooth (Bluetooth HID, for wireless mice and keyboards)

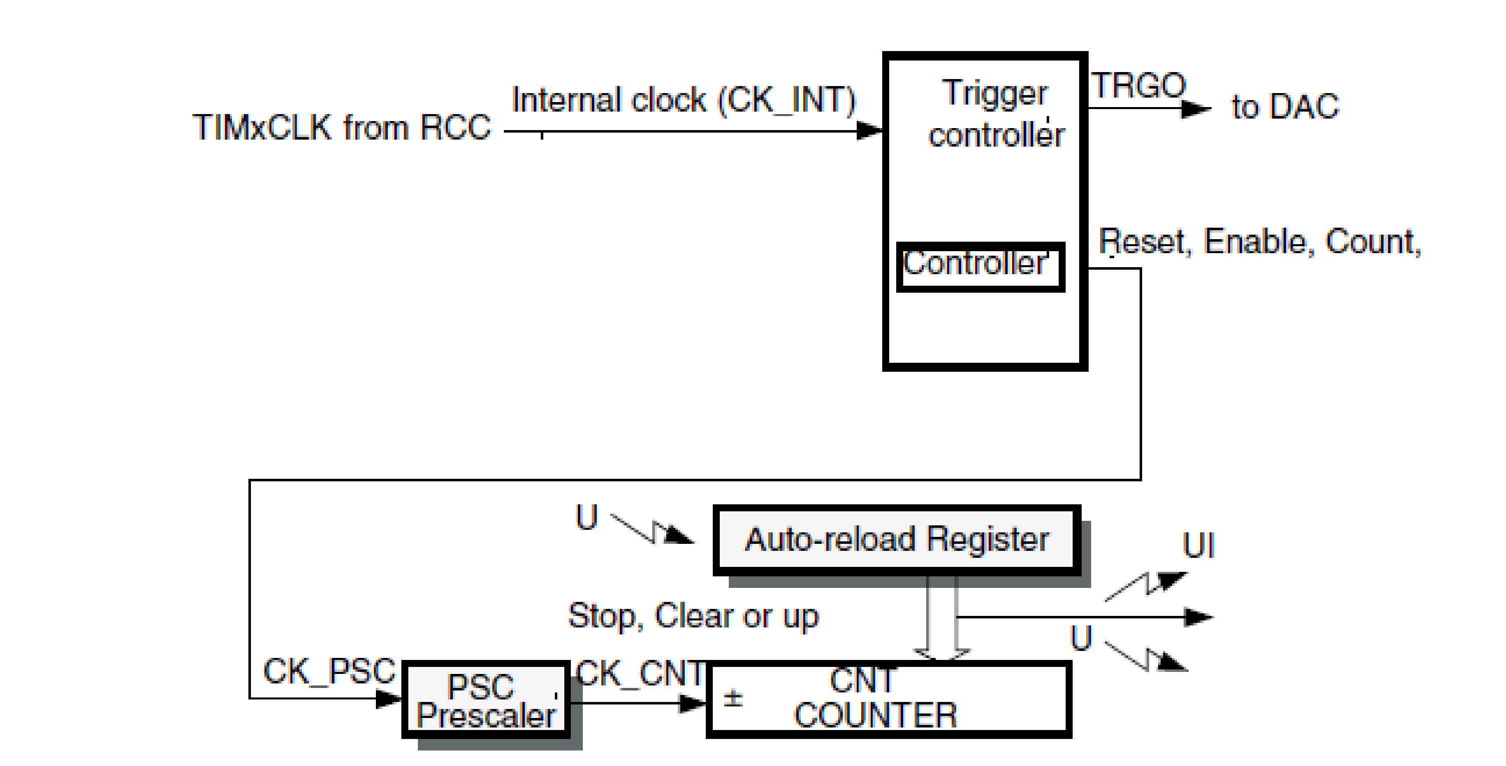
- Latency is more important than throughput in those devices.
- Devices communicate with Host send non-periodic reports (later…)

Time Base Generation

- Timer6 has to generate an $10\ ms$ period time base, and his clock frequency is $84\ MHz$.
- Problem n1: How can we generate $100\ Hz$ signal from $84\ MHz$ one?
- Timer can be configured to generate an interrupt on counter overflow.
- If it counts from 0 to CNT-1, time between 2 interrupts is

$$t = \frac{CNT}{f_{clk}} \Longrightarrow CNT = 840000$$

Timer Block Diagram



Time Base Generation

- Problem n2: Timer6 is a 16 bit timer, and $840000 > 65535\ (2^{16} - 1)$
- Solution: Clock Prescaler!

$$t = CNT * \frac{PSC}{f_{clk}}$$

- Prescaler is also a useful to decrease power consumption
  (remember dynamic power consumption: $Pd = C * f * {V_{dd}}^2$)
- It also decrease resolution of counter:
  - Resolution without prescaler: $\Delta t = \frac{1}{f_{clk}}$
  - Resolution with prescaler: $\Delta t = \frac{PSC}{f_{clk}}$

- In STM32F4 register $PSC_{reg} = PSC - 1$

Software design flow

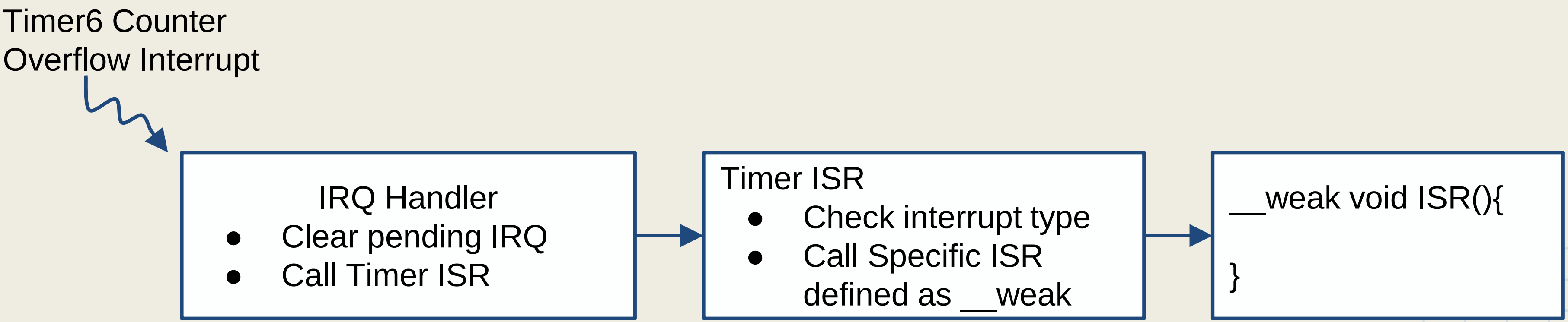
- Software Design
  - PWM LEDs control
  - Interrupt management
  - SPI
    - LIS3DSH Register Map
  - USB HID
- Put All together
- Conclusion

Led Blinking

- Cube Software has already initialize selected peripherals, so user don't need to do it
- Two simple function to start/stop PWM:
  - HAL_TIM_PWM_Start
  - HAL_TIM_PWM_Stop
- Parameters:
  - TIM_HandleTypeDef* htim: a pointer to a Timer Structure, you can find it's declaration in tim.c
  - uint32_t Channel: a macro (defined in STM32F4xx_hal_tim.h) to select channel. TIM_Channel_x, where x goes from 1 to 4

Interrupt management

- HAL_TIM_Base_Start_IT(&htim6); starts Timer6 in interrupt mode.
- How we can personalize ISR (Interrupt Service Routine) in order to perform required task?



ARM __weak keyword

- A weak function can be redefined in another source code
- If linker find two function with the same name, it uses the one without weak keyword. It is useful to separate application from drivers
- main.c

driver.c

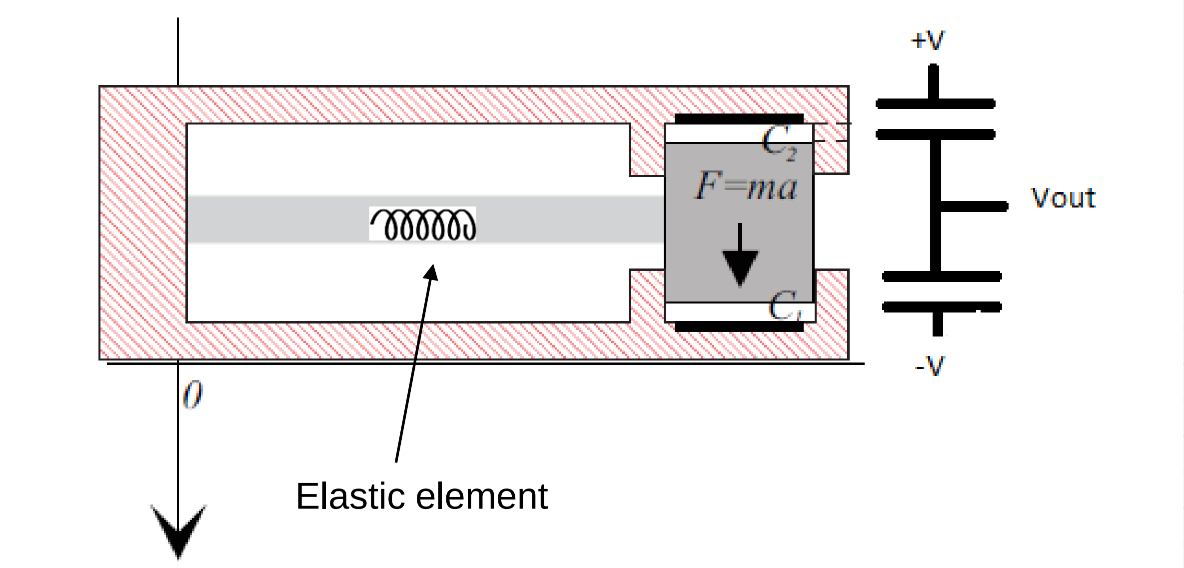
```
//user callback at application
//level
void callback(){
//do stuffs

}
```

```
void ISR(){
//previous stuff
callback();
}

//unused function
__weak void callback(){

}
```

- Timer callback function is HAL_TIM_PeriodElapsedCallback

Accelerometer background

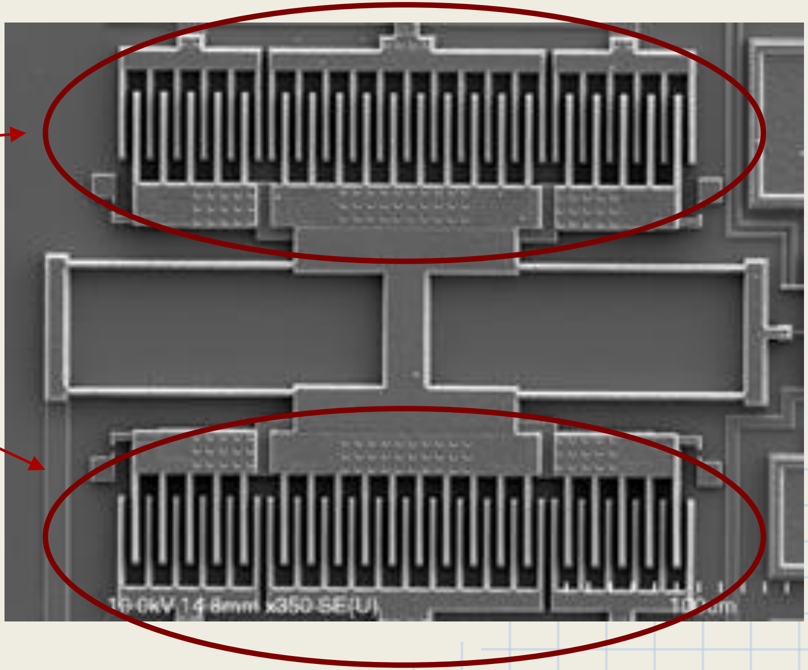
- Measure: displacement of mobile mass change a capacitance of the two electrode
- $\frac{V_u}{V} = \frac{x}{d_0}$
  - V = Drive voltage
  - x = Displacement
  - d_0 = Rest distance between electrode



Accelerometer in real world

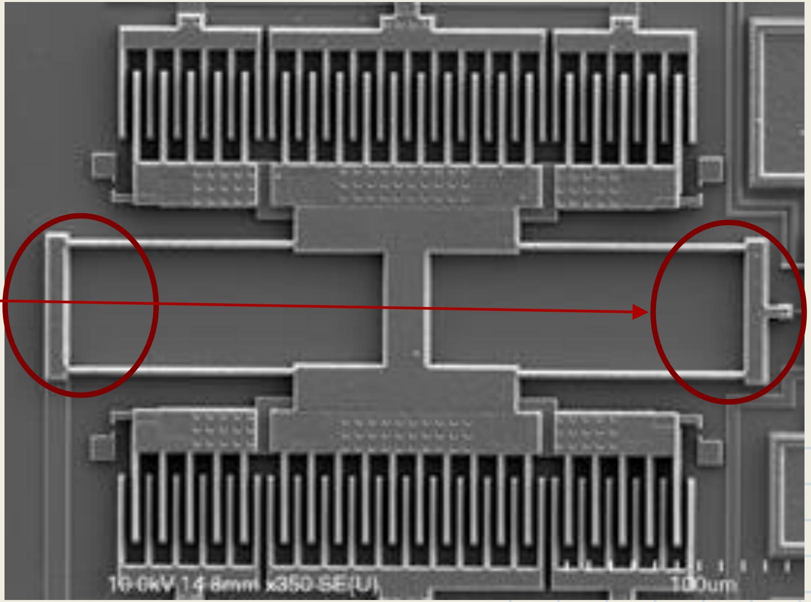
- Simple 1-axis accelerometer in this example

- Capacitors



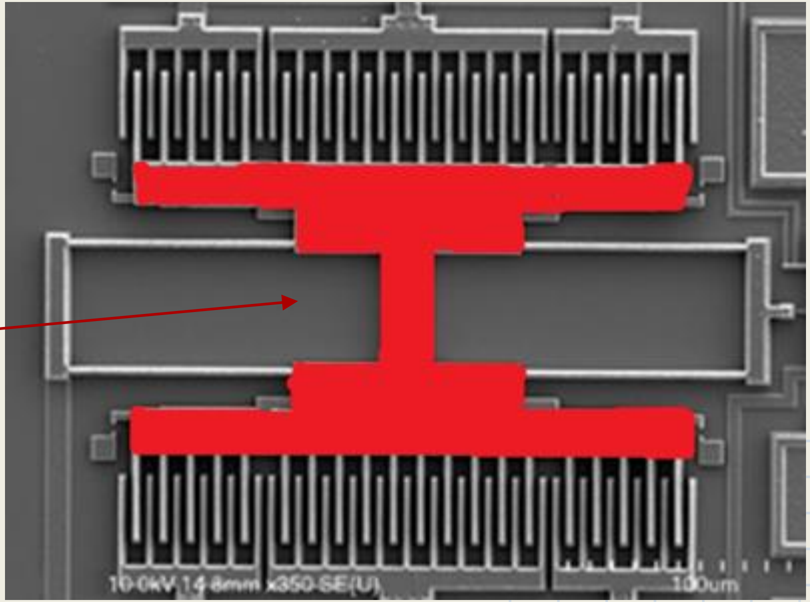
Accelerometer in real world

- Simple 1-axis accelerometer in this example
- Capacitors
- Spring



Accelerometer in real world

- Simple 1-axis accelerometer in this example
- Capacitors
- Spring
- Mobile Mass

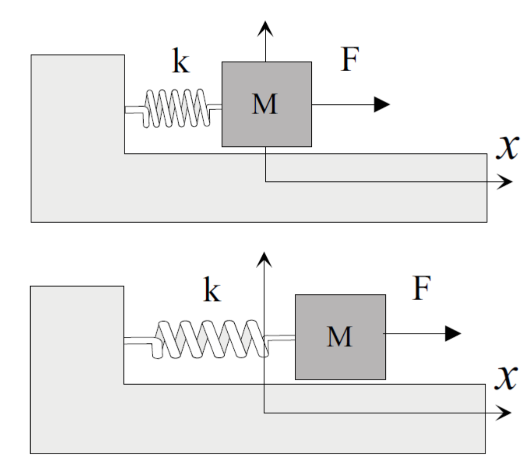


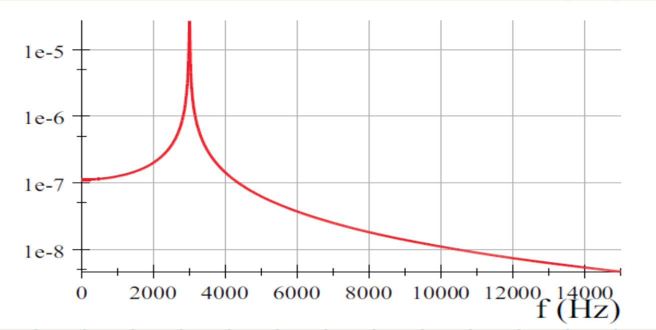
Just a little bit more...

- Relation between displacement and acceleration
- $F_{tot} = ma = m\ddot{x}$
- $F_{tot} = F - \beta\dot{x} - kx = m\ddot{x} \qquad (F = ma)$
  - F_n = Force applied by acceleration
  - β = Viscous friction coefficient
  - k = Elastic coefficient

Second order differential equation in x(t) ⟹ Laplace!

- $x(s) = \frac{F/m}{s^2 + \frac{\beta}{m}s + \frac{k}{m}}$
- If $s << \omega_0$ ⟹ x = $\frac{F}{K}$ ⟹ Hooke Law
- Second order Low Pass filter
  - $\omega_0 = \sqrt{\frac{k}{m}}, Q = \frac{\sqrt{km}}{\beta}$





Noise

- Strength components: $F = ma + F_n$
  - a is the MEMS acceleration
  - F_n is the force caused by Brownian mote of air, this is a noise source

$$F_n \approx \sqrt{4K_B T \beta}$$

System Sensitivity:

$$G_0 = \frac{1}{{\omega_0}^2}$$

We can report this noise to a “noise acceleration” dividing noise’s displacement by sensitivity:

$$a_n = \frac{F_n}{K} {G_0}^{-1} = \sqrt{\frac{4K_B T}{mQ} \omega_0}$$

Great bandwidth increase input noise

Fast System ⟺ Low Resolution

Dynamic Range

Dynamic Range: $DR = 20\log(\frac{a_{FS}}{a_n})$

Symbol	Parameter	Test conditions	Min.	Typ.(1)	Max.	Unit
FS	Measurement range(2)	FS bit set to 000		±2.0		g
		FS bit set to 001		±4.0		g
		FS bit set to 010		±6.0		g
		FS bit set to 011		±8.0		g
		FS bit set to 100		±16.0		g
So	Sensitivity	FS bit set to 000		0.06		mg/digit
		FS bit set to 001		0.12		mg/digit
		FS bit set to 010		0.18		mg/digit
		FS bit set to 011		0.24		mg/digit
		FS bit set to 100		0.73		mg/digit

Example: LIS3DSH

- 16 bit ⟹ $DR = 20\log(2^{16}) = 96.3\ dB$
- If FS bit is set to 000, $a_{FS} = \pm 2\ g = 4\ g$
- Input noise: $a_n = \frac{4\ g}{2^{16}} \approx 0.061\ mg$

Great Range ⟺ Low Resolution

Our Application

Requirements:

- $\omega_0 = 100\ Hz$,
- $a_{FS} = \pm 2\ g$
- Resolution 8 bit (USB HID)

Derived Specifications:

- Anti-aliasing filter: $f_{LP} \leq 50\ Hz$
- Truncation of 16 bit registers to 8 bit. We have to read only Most Significant Byte

How to send and receive data from accelerometer?

SPI Communication

Figure 7. SPI read protocol

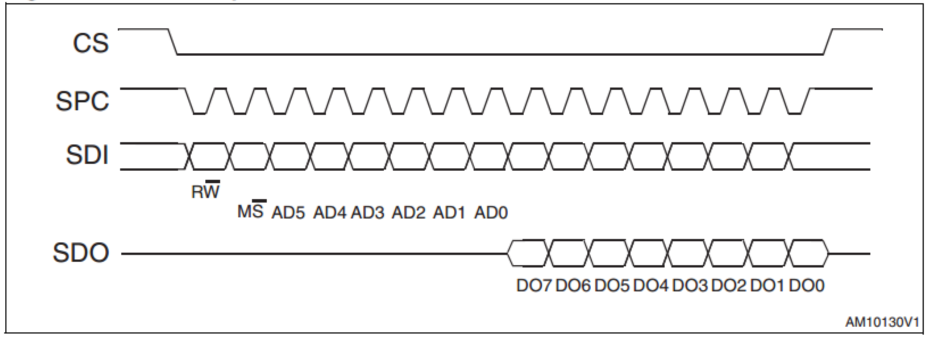
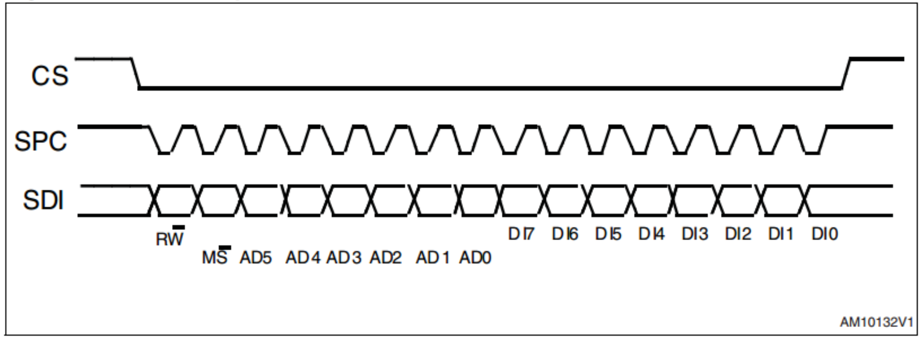


Figure 9. SPI write protocol



Steps:

- Put CS low
- Send register address. MSB is 1 for read or 0 for write operation
- Receive/Send a byte
- Put High CS

HID Report

HID USB Report Structure

- Defines structure of USB report and data fields
- Crazy to understand
- Standard descriptor (usually supported by Operative Systems) are provided by USB consortium.

Report Descriptor

```
USAGE_PAGE (Generic Desktop)              05 01
USAGE (Mouse)                             09 02
COLLECTION (Application)                  A1 01
  USAGE (Pointer)                         09 01
  COLLECTION (Physical)                   A1 00
    USAGE_PAGE (Button)                   05 09
    USAGE_MINIMUM (Button 1)              19 01
    USAGE_MAXIMUM (Button 3)              29 03
    LOGICAL_MINIMUM (0)                   15 00
    LOGICAL_MAXIMUM (1)                   25 01
    REPORT_COUNT (3)                      95 03
    REPORT_SIZE (1)                       75 01
    INPUT (Data,Var,Abs)                  81 02
    REPORT_COUNT (1)                      95 01
    REPORT_SIZE (5)                       75 05
    INPUT (Cnst,Var,Abs)                  81 03
    USAGE_PAGE (Generic Desktop)          05 01
    USAGE (X)                             09 30
    USAGE (Y)                             09 31
    LOGICAL_MINIMUM (-127)                15 81
    LOGICAL_MAXIMUM (127)                 25 7F
    REPORT_SIZE (8)                       75 08
    REPORT_COUNT (2)                      95 02
    INPUT (Data,Var,Rel)                  81 06
  END_COLLECTION                          C0
END_COLLECTION                            C0
```

	Bit 8		Bit 2	Bit 1	Bit 0
Byte 0	Unused		Center	Right	Left
Byte 1	X Axis				
Byte 2	Y Axis				
Byte 3	Unused				

Send Report

Very simple using ST USB Device Middleware

```
if((HID_Buffer[0] != 0) |(HID_Buffer[1] != 0) ||(HID_Buffer[2] != 0)){
    USBD_HID_SendReport(&hUsbDeviceFS, HID_Buffer, 4);
  }
```