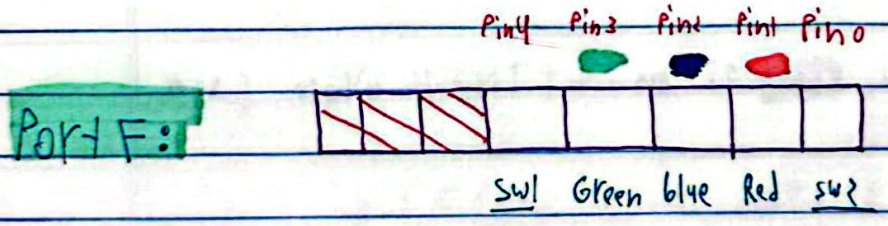
- Lab 6: TM4C introduction

TM4C micro controller has 6 Ports

Port A
Port B
Port C
Port D
Port E
Port F which we will use

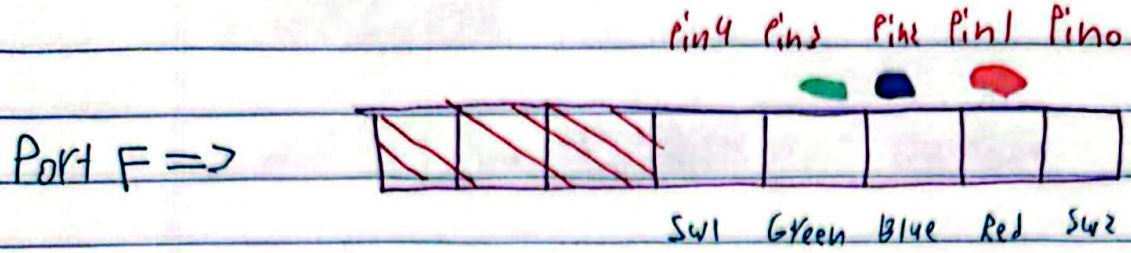
each port is associated with multiple Registers /will be explained later/
and each port has 8 bits (Pins)
/except Port F has 5/



/Sw: switch/

Each Register will do operations on the Pins, like make the Pin as input or output (like make the lights Pin1,2,3 as output and the switches as output), OR make it on/off, etc...

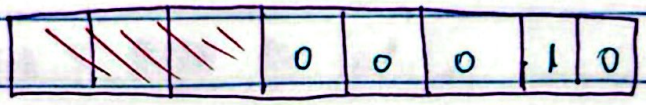
- Registers



1- GPIO DATA

is Responsible for setting the data

ex) GPIo_PortF_DATA_R = 0X02;



the light Red is on

note: for switches if the switch is Pushed sw=0 else sw=1

ex) make the red turn on if sw2 is Pushed

```
if (GPIo_PortF_DATA_R & 0x01 == 0)
    GPDR = 0X02;
```

if Sw2 is zero then turn on the Red

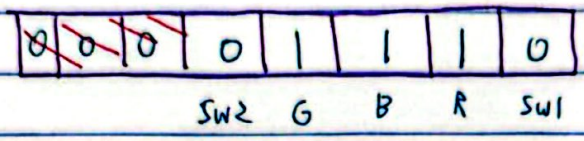
↓

Pin0

2- GPIODIR

will set the Inputs AND outputs

ex) GPIO_PortF_DIR_R = 0x0E



~~1: inPut~~ 0: inPut 1: outPut

So SW1 & SW2 inputs

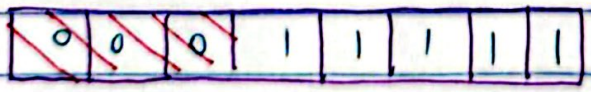
Green, Blue, Red outputs

3- GPIODEN

for if the Pin is digital or Analog (like in communication)

By default we want all of them Digital

GPIO_PortF_DEN_R = 0x1F



Scanned with CamScanner

4- RCGCGPIO

This Register used to enable the Port clock

1: enable 0: disable

ex) SYSCTL_RCGCGPIO_R = 0x20

1	0	0	0	0	0
Port F	Port E	Port D	Port C	Port B	Port A

this code will enable Port E only

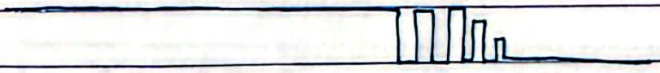
Scanned with CamScanner

- Lab 7:

Problem: in the Process of using the switches you might face a Problem were the switch do more than one click that's because of the noise

exPected:

Reality



Hardware solution: connect it with capaceter in Parallel to make L.P.F / low Pass filter/

Software solution: make delay of a Perfect time for take just the first bounce

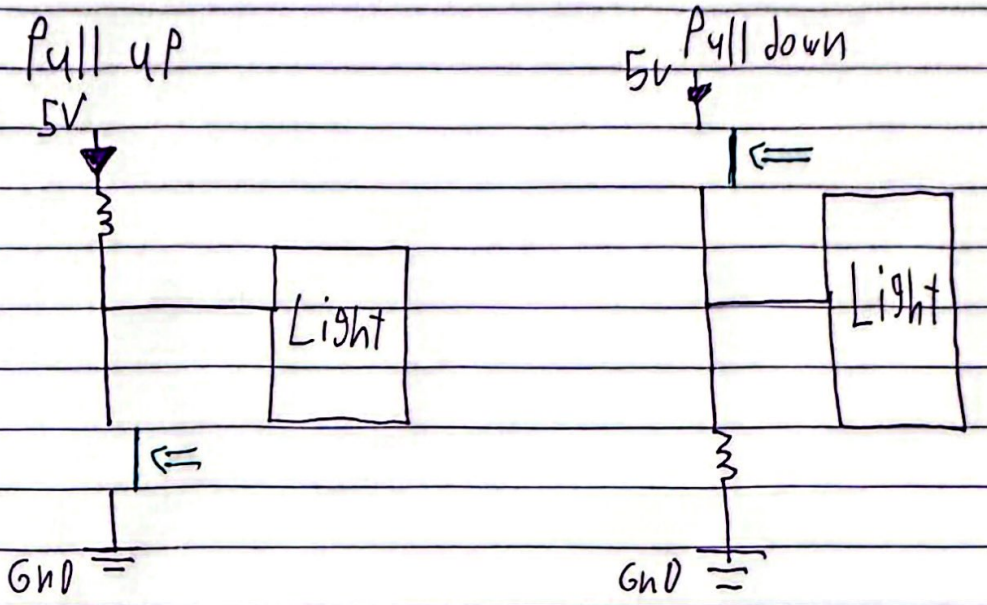
5- GPIOPUR:

Make the switches as Pull up Registers

1: enable Pull up 0: disable Pull up

6- GPIOPDR

Make the switches as Pull down



Pull up : if you Push go to the ground (short circuit)

Push the button : light is off

NoT " " " : light is on

- Interrupts

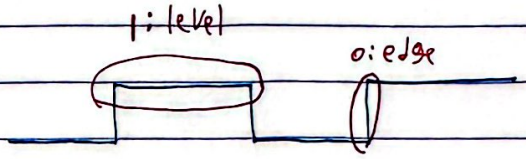
- Interrupt setup Registors

1- GPIOIS:

use to determine edge or level interrupt sensetive

0: edge

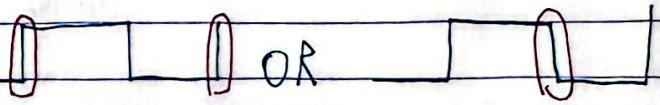
1: level



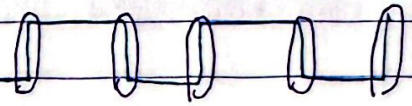
2- GPIOBE: ~~/* Just if we use edge sensetive */~~

used to determine if bothe edges or one edge interrupt sensetive

0: one edge/level



1: both edges/level

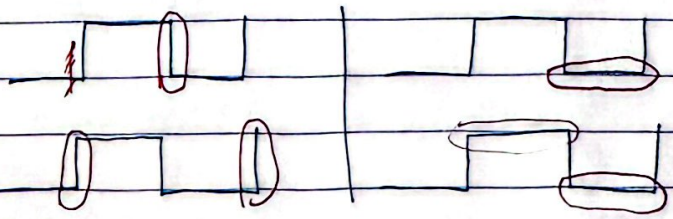


3- GPIOIEV: /* Just if we use one edge or level */

used to determine which edge, Rise or Fall

0: Falling

1: Rising



4- GPIOIM:

used to enable/disable switch as interrupt

0: Disable

1: Enable

5- GPIOICR:

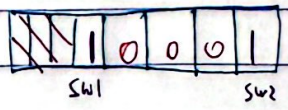
used to clear the previous interrupts to start a new one

ex) GPIOF-ICR = 0X10 (clear sw1 last interrupt)

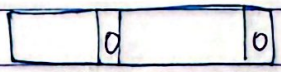
- examples on interrupt set-ups

1). creat an interrupt that, work at sw1, sw2 on falling edge

GPIOF → IM = 0X11 (PortF)

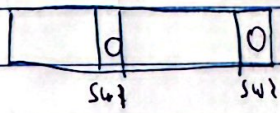


edge sensitive GPIOF → IS = 0X00



One edge GPIOF → IBE = 0X00

falling edge GPIOF → IEV = 0X00

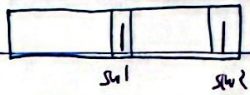


clear last interrupt GPIOF → ICR = 0X11

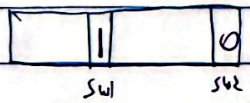
NVIC → ISER[0] = (1<<30) will be explained later

2) Create an Interrupt that, work on sw1, sw2
where:
sw1: level sensetive
sw2: Rising edge sensetive

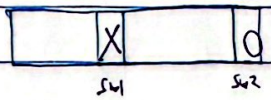
GPIOF->IM = 0X11



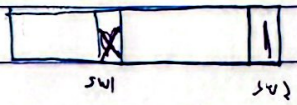
GPIOF->IS = 0X10



GPIOF->IBE = 0X00



GPIOF->IEV = 0X01



to clear previous Interrupts => GPIOF->ICR = 0X11

NVIC->ISER[0] |= (1<<30) (will be explained later)

- Read from Interrupt

~~GPIOMIS~~

GPIOMIS:

1: there's Interrupt
0: // No //

ex) if (GPIOF->MIS & 0x01) {
Sw1 Pressed
GPIOF->ICR = 0x10
else if (GPIOF->MIS & 0x10) {
Sw2 Pressed
GPIOF->ICR = 0x01

NVIC_EN0: Nested Vector Interrupt controller

it's an array that contain multiple interrupts controller that need to be enabled.

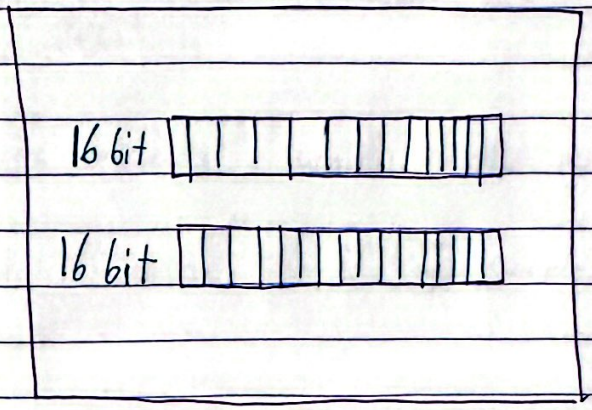
the interrupt we want to enable here is Port E interrupt which is in

```
NVIC->ISER[0] |= (1<<30)
```

Scanned with CamScanner

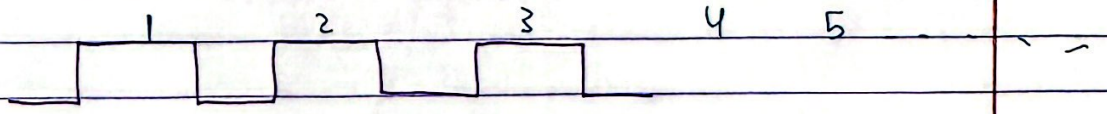
- Lab 8

Timer x



- timer X where X can be 1, 2, 3, ...

is a timer that contain two Registers 16 bit that count from 0 - 2^{16} with adding one each clock pulse



or you can concatinate them in one big Register to count from 0 - 2^{32} /* code will be explained */

where clock steed is 16 MHz/s or 16×10^6 Hz per second

You can conclude that

$$\text{timer} = \frac{\text{count number (between } 0 - 2^{16}) \text{ OR } (0 + 2^{32})}{16 \times 10^6}$$

- So if you chose to cont from $0-2^{14}$ the timer will be

$$\text{timer} = \frac{2^{14}}{16\times10^{6}} = 1.024\times10^{-3} \text{ second}$$

if from $0-2^{16}$ which is the max in 16 bit mode

$$\text{timer} = \frac{2^{16}}{16\times10^{6}} = 4.096\times10^{-3} \text{ second}$$

You can decrease the clock speed by dividing it by number from 0-255 (Prescaller)

Final equation

$$\text{timer} = \frac{\text{counter time}}{\frac{16\times10^{6}}{X}}$$

counter time: numbe from $0-2^{16}$ in 16 bit mod
number from $0-2^{32}$ in 32 bit mod
] IRS

16×10^{6} : clock speed

X: the number divides clock speed from 0-255
also named: Prescaller

ex) make the timer count to 500 ms

Method 1: let X be 255 (max)

$$\text{timer} = \frac{\text{count\_number}}{\frac{16 \times 10^6}{X}}$$

$$500 \times 10^{-3} = \frac{\text{count\_number}}{\frac{16 \times 10^6}{255}}$$

$$\text{count\_number} = 31372.55$$

Method 2: let count_number be 2^{16} (max in 16 mod)

$$500 \times 10^{-3} = \frac{2^{16}}{\frac{16 \times 10^6}{X}}$$

$$X = 122$$

So in order to obtain 500ms you can do

count_number = 31372.55 , X = 255

count_number = 2^{16} , X = 122

X max is 255

Remember: count_number max is 2^{16} in 16 mod

2^{32} in 32 mod

Scanned with CamScanner

ex) Make the timer count to 10 sec

$$\text{timer} = \frac{\text{count\_number}}{\frac{16 \times 10^6}{X}}$$

let X be 255 (Prescaller)

$$10 = \frac{\text{count\_number}}{\frac{16 \times 10^6}{255}}$$

$$\text{count\_number} = 627450 > 2^{16}$$

So you will need 32 bit mod

* There's no Prescaller (X) in 32 bit Mod

- Re solve the Problem

$$10 = \frac{\text{count\_number}}{16 \times 10^6}$$

$$\text{count\_number} = 160 \times 10^6 \leq 2^{32} \checkmark$$

Scanned with CamScanner

- Timer code setup

1. SYSCTL -> RCGCTimer

to enable a specific timer

ex) SYSCTL -> RCGCTimer1 = (1<<1) enable timer 1

// // |= (1<<2) enable timer 2

2- Timer1 -> CTL = 0 disable the timer

= 1 enable // //

3- Timer1 -> CFG

32 0x0 bit mode (32)

16 0x4 bit mode (16)

4- Timer1 -> TAMR

0x02 periodic (keep counting)

0x01 one shot (count just one time)

5- Timer1 -> TAPR:

Represent x (0-255)

Scanned with CamScanner

6- Timer1 -> TAILR

Represent "counter_time"

ex) Timer1 -> TAILR = 64000

7- Timer1 -> ICR

clear previous interrupt

ex) Timer1 -> ICR = 0x1

8- Timer1 -> IMR

enable timer interrupt

last don't forget to enable timer port

NVIC -> ISER[0] |= (1<<21)

- Lab 10

- in this experiment our primary goal is to read analog values and convert it to digital.

short ex)
```
if(analog_value >= 2048)
    GPIOF->data = Red;

else
    GPIOF->data = !Red;
```

the analog value "analog_value" is converted from 0-4095 to either Red or !Red.
for example

* TM4C123 (our microcontroller)

take Voltage up to 3.3V with 12 bit resolution

so the Volt 0 - 3.3
the resolution $0 - 2^{12}$

$0 \rightarrow 3.3V$

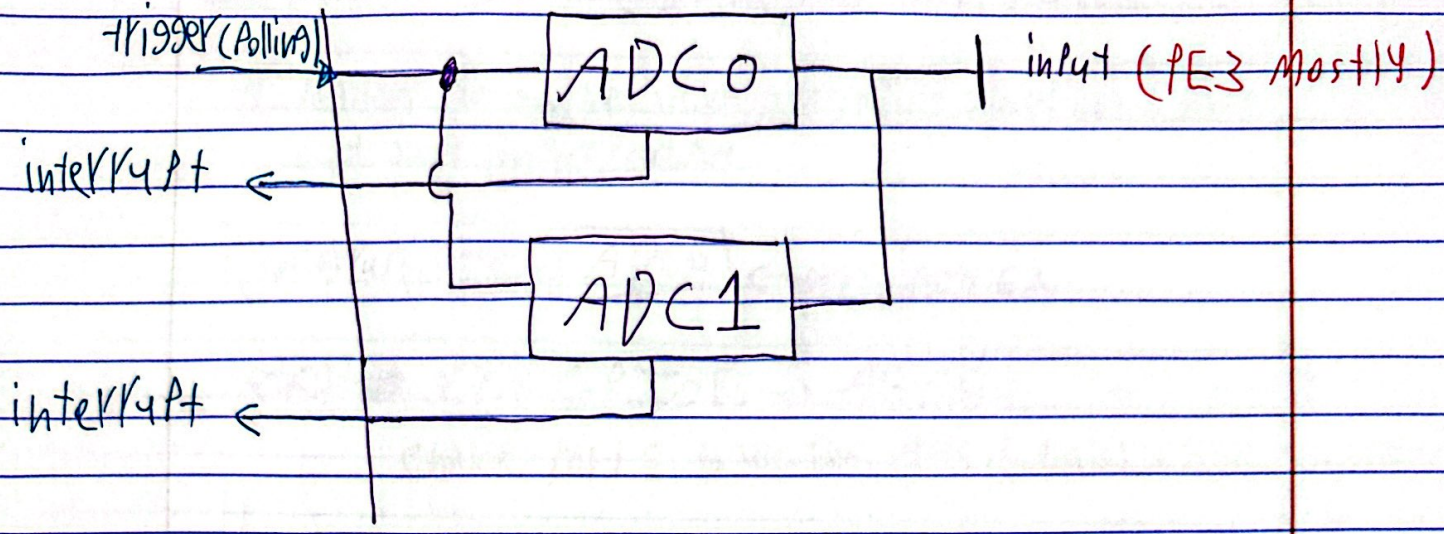
$0 \rightarrow 4095$

so if you want to take value 2.2 V for example

$$\left(\frac{4095}{3.3}\right)(2.2) = 2730$$

ex)
```
if(analog_value >= 2730)
    GPIOF = Blue;
```

Scanned with CamScanner



- So we got Analog value from the input (0-3.3v) and then the ADC convert it to (0-4095)

* we can get the value through Polling or interrupt

- There is just 12 Pins that can read Analog

	AN0	AN1	AN2	... etc...	AN11
Pin name	PE3	PE2	PE2		PB5
Pin number	6	7	8		57

- Set up Registers

1) RCGC GPIO:

a Register we already know is used to enable Ports (check exp6)

ex) SYSCTL -> RCGCGPIO |= (1<<5);
SYSCTL -> RCGCGPIO |= (1<<4);

enable PortF and PortE

F	E	D	C	B	A
5	4	3	2	1	0

2) RCGC ADC

a Register to enable ADC (1 OR 0)

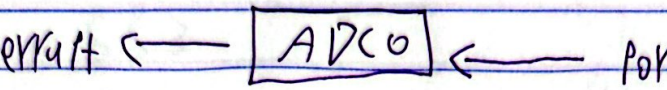
ex) SYSCTL -> RCGCADC |= (1<<0)

enable ADC 0

Remember: we will mostly use PE3 (pin E3) on PORT E

3) AFSEL (GPIOE -> AFSEL)

a Register to enable using alternative function (ADC) not just input/output



ex) GPIOE -> AFSEL |= (1<<3);

enable PORT E to use the ADC (Analog digital converter)

4) DEN (GPIOE -> DEN)

another register we already know (exp. 6) to enable/disable digital

ex) GPIOE -> DEN &= ~(1<<3);

GPIOF -> DEN = 0X1F;

disable the digital on PE3 (will read analog)

enable on PF 0,1,2,3,4,5 (will turn red, Blue, Green) in digital value)

5) AMSEL (GPIOE -> AMSEL)

register to enable/disable the analog value 4 usec

ex) GPIOE -> AMSEL |= (1<<3);

enable Analog on PORT E3

Scanned with CamScanner

- Samplers

Sample sequencer	number of samples
SS3	1
SS2	4
SS1	4
SS0	8

* Mostly we will use SS3
* if you didn't take communications don't worry about its meaning

6) ACTSS (Activate ss)

for activating the sampler sequencer (sss)

ex) ADC0 -> 8= ~(1<<3)

: code
: code

ADC0 -> 1= (1<<3)

disable ss3 while setup then activate it

Scanned with CamScanner

7) ADCEMUX

ex) ADC0->EMUX &= ~0XF000

enable trigger (don't worry about it, just copy from the manual)

8) SSMUXA

Registor to connect the analog channel (0-11) lik (PE3, PE2, PB5) with the chosen SS

ex) ADC0->SSMUX3 = 0;

connect SS3 (MUX3) with ANO (PE3)

SS3 PE3

9) PSSI

tell the ADC to start sampling (convert to dig);

ex) ADC0->PSSI |= (1<<3)

usually in the while loop to keep converting

Scanned with CamScanner

10) RIS

a Register to tell if converting operation is finished

(7+1) = eight (1) - ثمانية

ex) while ((ADC0 -> RIS & 8) == 0)

wait until ADC0 is finished (when 8)

1
7+1

(better to use interrupt)

11) ISC

Register to clear Flag bit

ex) ADC0 -> ISC = 8 ;

12) SSFIFO3

read the Value From SS3

~~ex) ADC0 ->~~ ex) int analog_Value = ADC0 -> SSFIFO3

~~ex) int Adc_Value = ADC0 -> SSFIFO3~~

read the Value From SS3

* Voltage = adc_Value * 0.0008

$$\left(\frac{3.3}{4095}\right)$$

Scanned with CamScanner

* instead of Register IO (RIS) it's better to use interrupt

```
int adc = 0; general variable
Void ADC0SS3_Handler(Void){

    adc_value = ADC0->SSFIFO3;
    ADC0->ISC = 8;
    ADC0-> PSSI |= (1<<3);

}
```

and you will need to enable SS3 interrupt

```
NVIC->ISER[0] |= 0x00010000;
```

From manual

Scanned with CamScanner