LABORATORY 3: TIVA I/O

Part 1: Tiva exercises.

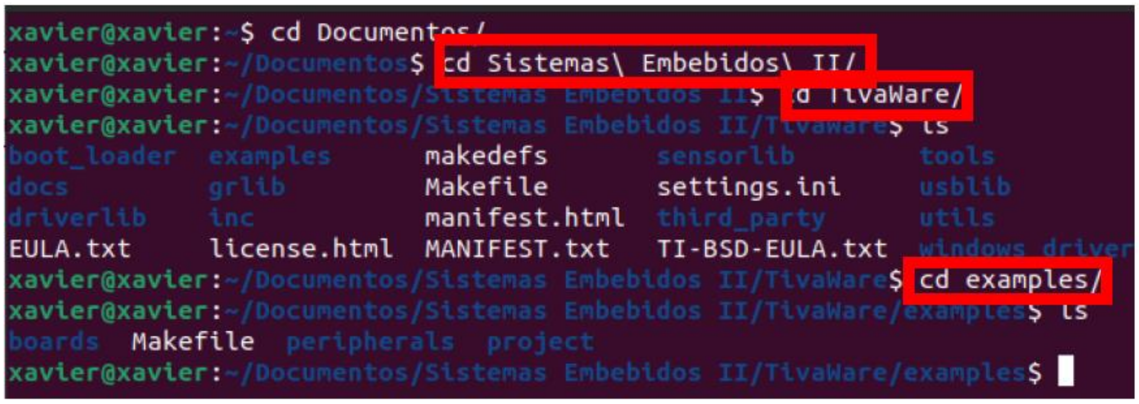
Theory Concepts:

In this laboratory we will learn how to use the TM4C1294XL board in our projects. For that reason, we will use the Tivaware SDK installed during previous laboratories.

Working with TIVA will require a few additional steps in comparison to Raspberry Pi. However, with proper practice it will be a mechanic process and easy to configure.

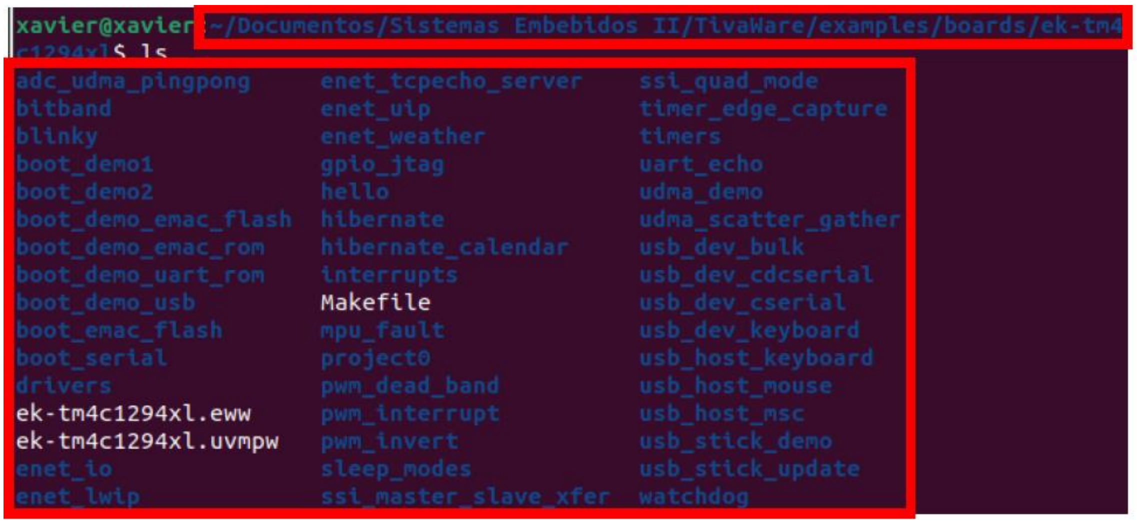
Project creation:

We will create a test project to understand how TIVA manages projects and makes them possible to compile:



```
xavier@xavier:~$ cd Documentos/
xavier@xavier:~/Documentos$ cd Sistemas\ Embebidos\ II/
xavier@xavier:~/Documentos/Sistemas Embebidos II$ cd TivaWare/
xavier@xavier:~/Documentos/Sistemas Embebidos II/TivaWare$ ls
boot_loader  examples      makedefs       sensorlib        tools
docs         grlib         Makefile       settings.ini     usblib
driverlib    inc           manifest.html  third_party      utils
EULA.txt     license.html  MANIFEST.txt   TI-BSD-EULA.txt  windows_driver
xavier@xavier:~/Documentos/Sistemas Embebidos II/TivaWare$ cd examples/
xavier@xavier:~/Documentos/Sistemas Embebidos II/TivaWare/examples$ ls
boards  Makefile  peripherals  project
xavier@xavier:~/Documentos/Sistemas Embebidos II/TivaWare/examples$
```

We will access the folder that contains the TivaWare folder, then access to examples and boards and ek-1294xl.

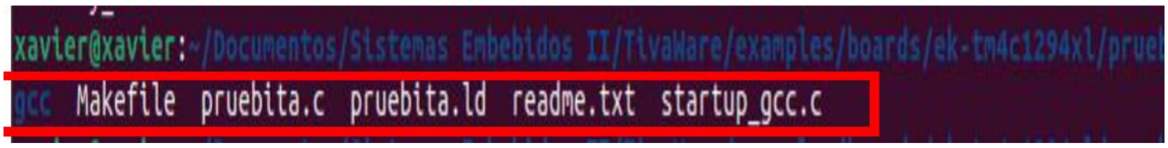


```
xavier@xavier:~/Documentos/Sistemas Embebidos II/TivaWare/examples/boards/ek-tm4c1294xl$ ls
adc_udma_pingpong        enet_tcpecho_server   ssi_quad_mode
bitband                  enet_uip              timer_edge_capture
blinky                   enet_weather          timers
boot_demo1               gpio_jtag             uart_echo
boot_demo2               hello                 udma_demo
boot_demo_emac_flash     hibernate             udma_scatter_gather
boot_demo_emac_rom       hibernate_calendar    usb_dev_bulk
boot_demo_uart_rom       interrupts            usb_dev_cdcserial
boot_demo_usb            Makefile              usb_dev_cserial
boot_emac_flash          mpu_fault             usb_dev_keyboard
boot_serial              project0              usb_host_keyboard
drivers                  pwm_dead_band         usb_host_mouse
ek-tm4c1294xl.eww        pwm_interrupt         usb_host_msc
ek-tm4c1294xl.uvmpw      pwm_invert            usb_stick_demo
enet_io                  sleep_modes           usb_stick_update
enet_lwip                ssi_master_slave_xfer watchdog
```

Inside that folder we will find a lot of different projects that can be used to generate different interactions with TIVA.

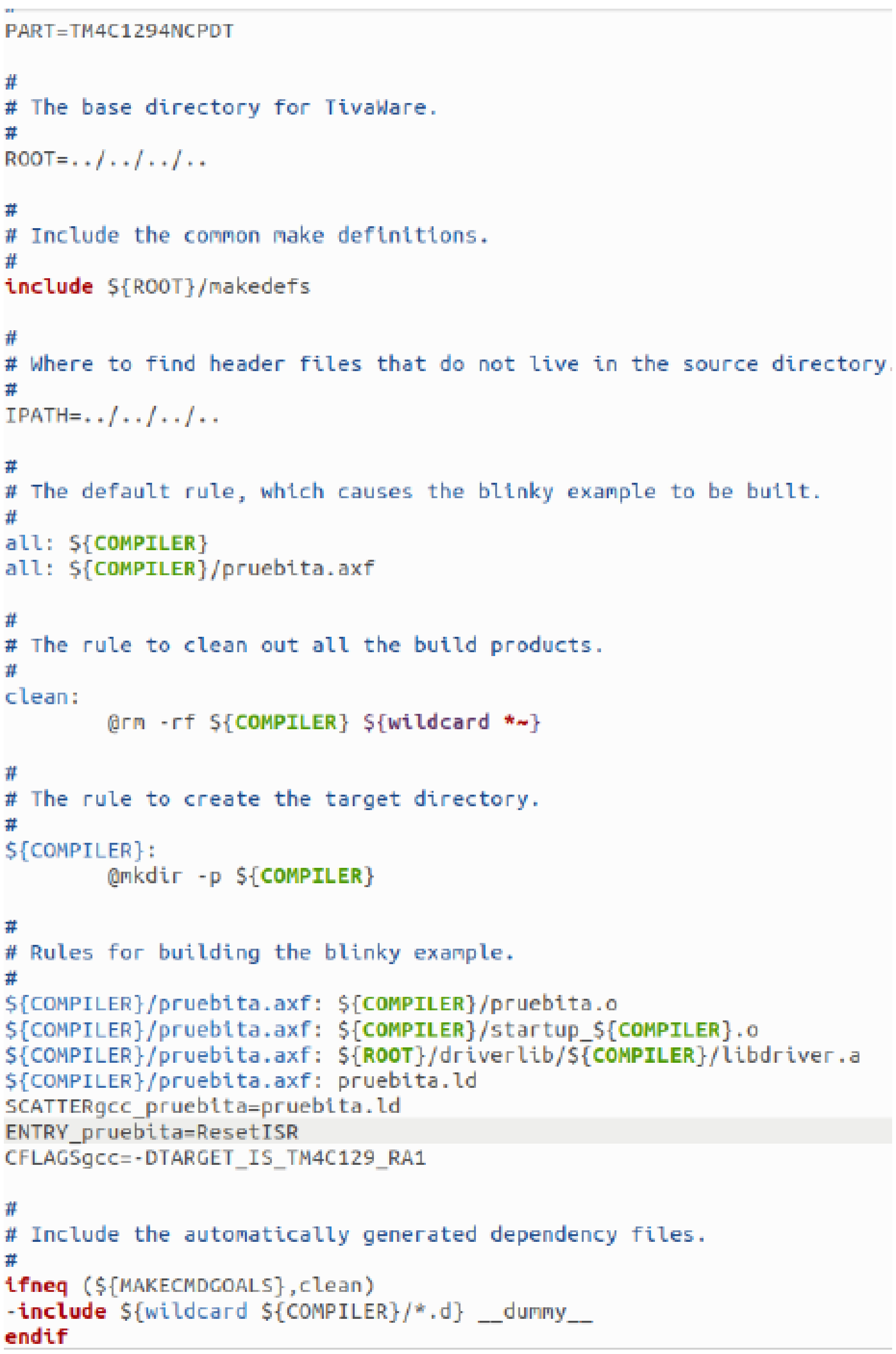
To create a new project a folder with similar configurations must be created. To simplify the new project creation, we will use the “blinky” folder as our base project.

We will create a copy of blinky folder and rename it as “test_project”. Then we will access to the new folder.



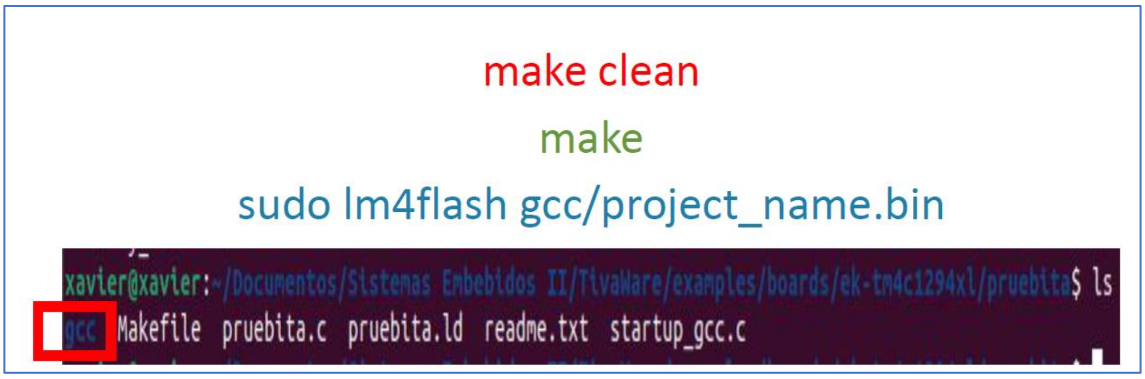
```
xavier@xavier:~/Documentos/Sistemas Embebidos II/TivaWare/examples/boards/ek-tm4c1294xl/prue
gcc  Makefile  pruebita.c  pruebita.ld  readme.txt  startup_gcc.c
```

We will see some files named as “blinky.xx” those files must be renamed as “test_project.xx” respecting the file type. In the image the files were renamed as “pruebita.xx”.



```
PART=TM4C1294NCPDT

#
# The base directory for TivaWare.
#
ROOT=../../../..

#
# Include the common make definitions.
#
include ${ROOT}/makedefs

#
# Where to find header files that do not live in the source directory.
#
IPATH=../../../..

#
# The default rule, which causes the blinky example to be built.
#
all: ${COMPILER}
all: ${COMPILER}/pruebita.axf

#
# The rule to clean out all the build products.
#
clean:
	@rm -rf ${COMPILER} ${wildcard *~}

#
# The rule to create the target directory.
#
${COMPILER}:
	@mkdir -p ${COMPILER}

#
# Rules for building the blinky example.
#
${COMPILER}/pruebita.axf: ${COMPILER}/pruebita.o
${COMPILER}/pruebita.axf: ${COMPILER}/startup_${COMPILER}.o
${COMPILER}/pruebita.axf: ${ROOT}/driverlib/${COMPILER}/libdriver.a
${COMPILER}/pruebita.axf: pruebita.ld
SCATTERgcc_pruebita=pruebita.ld
ENTRY_pruebita=ResetISR
CFLAGSgcc=-DTARGET_IS_TM4C129_RA1

#
# Include the automatically generated dependency files.
#
ifneq (${MAKECMDGOALS},clean)
-include ${wildcard ${COMPILER}/*.d} __dummy__
endif
```

All the “blinky” assignations inside the Makefile. This is because this file determines where the project dependencies are and how to manage them. With the previous preparations we are ready to flash our board with the new project.



```
make clean
make
sudo lm4flash gcc/project_name.bin
```

```
xavier@xavier:~/Documentos/Sistemas Embebidos II/TivaWare/examples/boards/ek-tm4c1294xl/pruebita$ ls
gcc  Makefile  pruebita.c  pruebita.ld  readme.txt  startup_gcc.c
```

Using the image commands we will clear the previous creation of the compilable file, create a new one and flash the TIVA board with the new .bin file.

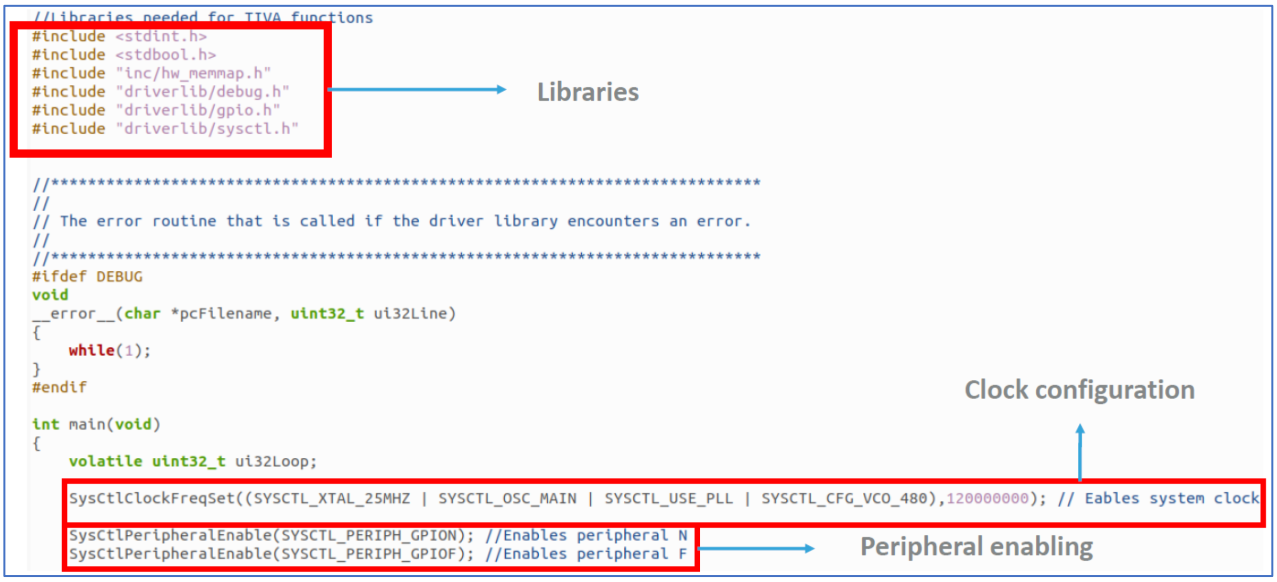
What is a makefile? How does it work?

What happens if we don't change all the files name?

What is a .bin file?

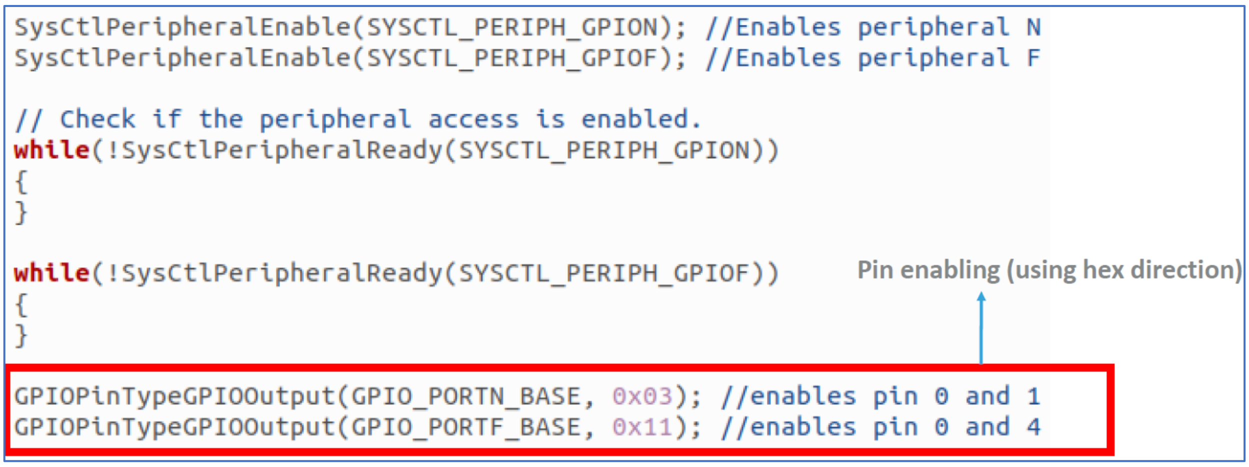
TIVA C File:

We will analyze the blinky C file to see some key elements in the code syntax:



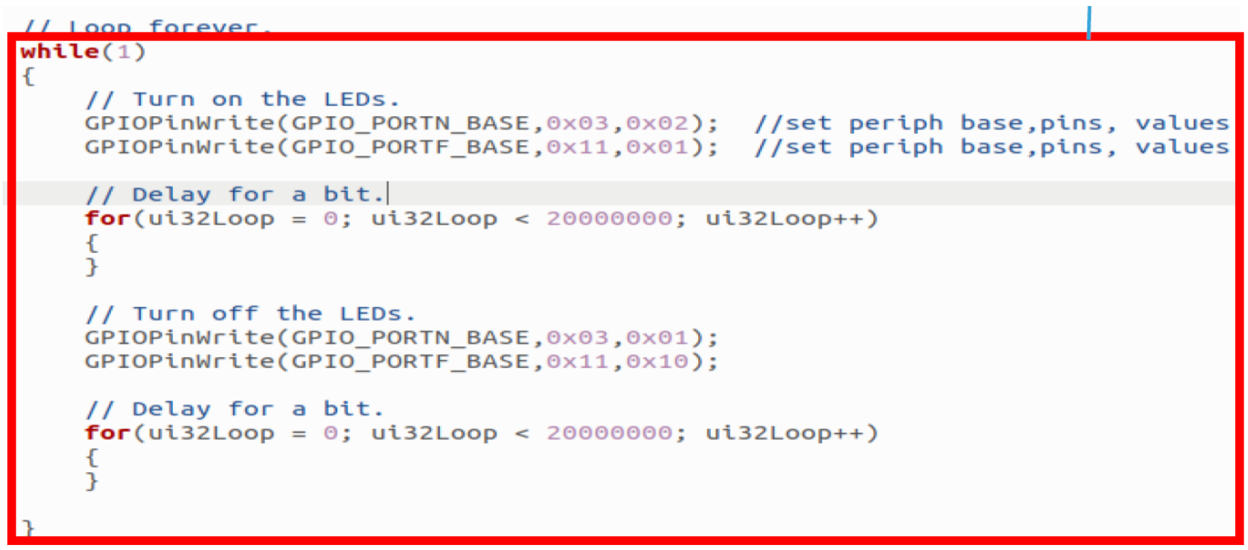
```c
//Libraries needed for TIVA functions
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/sysctl.h"

//*****************************************************************************
//
// The error routine that is called if the driver library encounters an error.
//
//*****************************************************************************
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
    while(1);
}
#endif

int main(void)
{
    volatile uint32_t ui32Loop;

    SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN | SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480),120000000); // Eables system clock

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION); //Enables peripheral N
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //Enables peripheral F
```

To enable peripherals, we will need to know the peripheral letter, this can be seen in the TIVA datasheet. In this first image we are just enabling the whole peripheral to be used, every peripheral contains 8 pins that goes from 0 to 7.



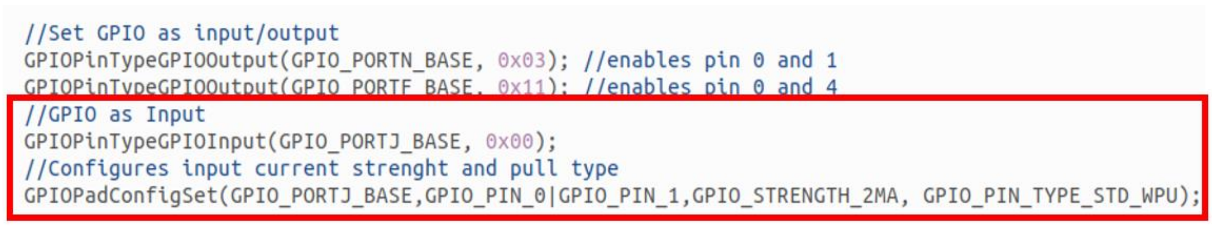
```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION); //Enables peripheral N
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //Enables peripheral F

// Check if the peripheral access is enabled.
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPION))
{
}

while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF))
{
}

GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, 0x03); //enables pin 0 and 1
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, 0x11); //enables pin 0 and 4
```

We enable specific pins and define them as Input or Output. For this laboratory we can use the pin declaration from blinky.



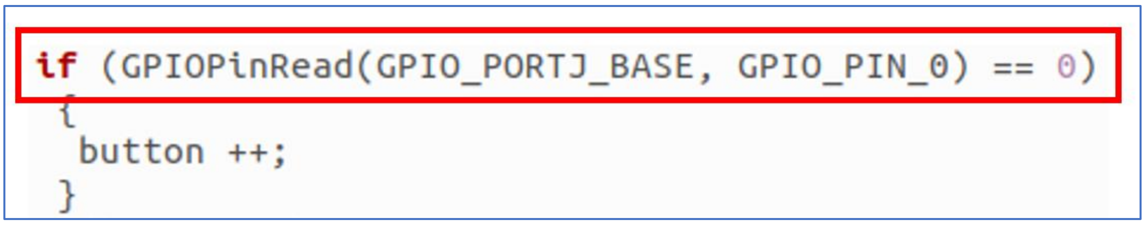
```
// Loop forever.
while(1)
{
    // Turn on the LEDs.
    GPIOPinWrite(GPIO_PORTN_BASE,0x03,0x02);  //set periph base,pins, values
    GPIOPinWrite(GPIO_PORTF_BASE,0x11,0x01);  //set periph base,pins, values

    // Delay for a bit.
    for(ui32Loop = 0; ui32Loop < 20000000; ui32Loop++)
    {
    }

    // Turn off the LEDs.
    GPIOPinWrite(GPIO_PORTN_BASE,0x03,0x01);
    GPIOPinWrite(GPIO_PORTF_BASE,0x11,0x10);

    // Delay for a bit.
    for(ui32Loop = 0; ui32Loop < 20000000; ui32Loop++)
    {
    }
}
```

Finally, we can use our output pins to generate a write statement. The GPIOPinWrite function will have the base peripheric, pins enabled and values to activate. We can use for loops to generate delays, or a function called SysCtlDelay.



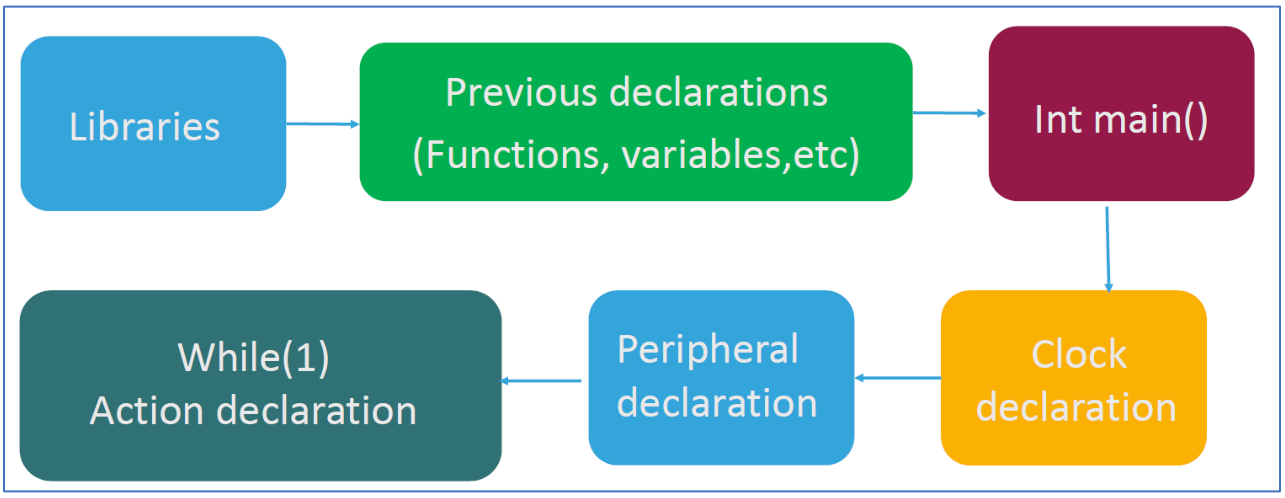
```
//Set GPIO as input/output
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, 0x03); //enables pin 0 and 1
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, 0x11); //enables pin 0 and 4
//GPIO as Input
GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, 0x00);
//Configures input current strenght and pull type
GPIOPadConfigSet(GPIO_PORTJ_BASE,GPIO_PIN_0|GPIO_PIN_1,GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
```

If we want to use a pin as Input, we can follow the image definition. Is important to mention that we must define the input pull type and strength.



```
if (GPIOPinRead(GPIO_PORTJ_BASE, GPIO_PIN_0) == 0)
 {
  button ++;
 }
```

If we want to review if the pin is pressed, we can compare it's value with 0.


Libraries
Previous declarations (Functions, variables,etc)
Int main()
Clock declaration
Peripheral declaration
While(1) Action declaration

To understand better the pin definition and learn additional functions that can be useful is important to review the documentation provided in class and read the datasheet and driverlib from TIVA.

What happens if we use driverlib for delays?

If we want to evaluate if the button is not pressed, what should we use?

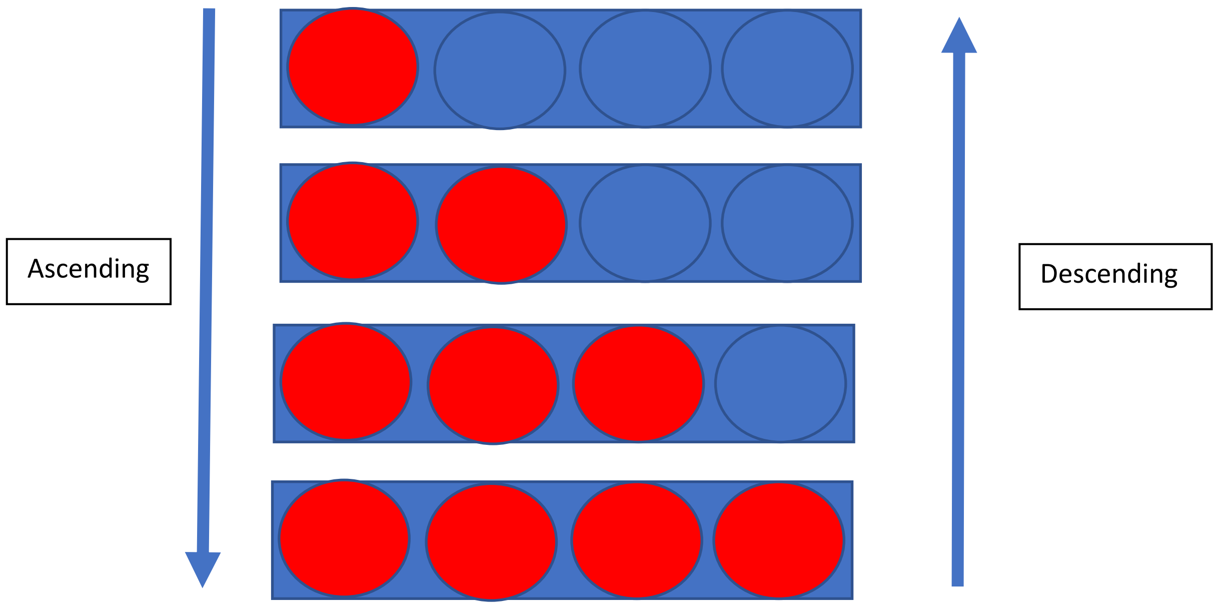
Short exercises:

- Example 1: Write a code that turns on the PN0 led, delays 1 second and turns on the PN1 led, then turns off both leds.
- Example 2: Write a code to turn on the PN0 and PF0 leds, then Turn on the PN1 and PF4 leds, then turn off the leds in the same sequence.
- Example 3: Write a code that turns on the PN1 user led if the PJ1 user switch is pressed.

Is it possible to change the code you made to use functional programming? Why or why not?

Exercise 1. GPIO Tiva Exercise (User LEDS)

Using the 4 user LEDs and 2 user Switch make a simple LED sequence. With the first button make the LEDs to change their state in ascending order and with the second button change the LEDs in descending order.



Exercise 2. GPIO Tiva Exercise (User LED sequence)

Using the 4 user LEDs generate a loop sequence with 4 states minimum. Each state must be active for 2 seconds before it changes, use SysCtlDelay. How did you change the state duration? Is there a more efficient way to do this? (Investigate timers in TIVA)

Exercise 3. GPIO Tiva Exercise (Binary Counter)

Using the 4 user LEDs and 2 user Switch integrated on Tiva develop a binary counter with all the values showed in table 1 following the next features:

- When switch 1 is pressed, binary counter increases.
- When switch 2 is pressed, binary counter reduces.
- All 4 user LEDs represent the binary values.
- When counter reaches value 15 it shouldn’t keep increasing.
- When counter goes to value 0 it shouldn’t keep decreasing.
- Store the decimal value in an integer variable called “counter”.

Is it possible to change the code you made to use functional programming? Why or why not?