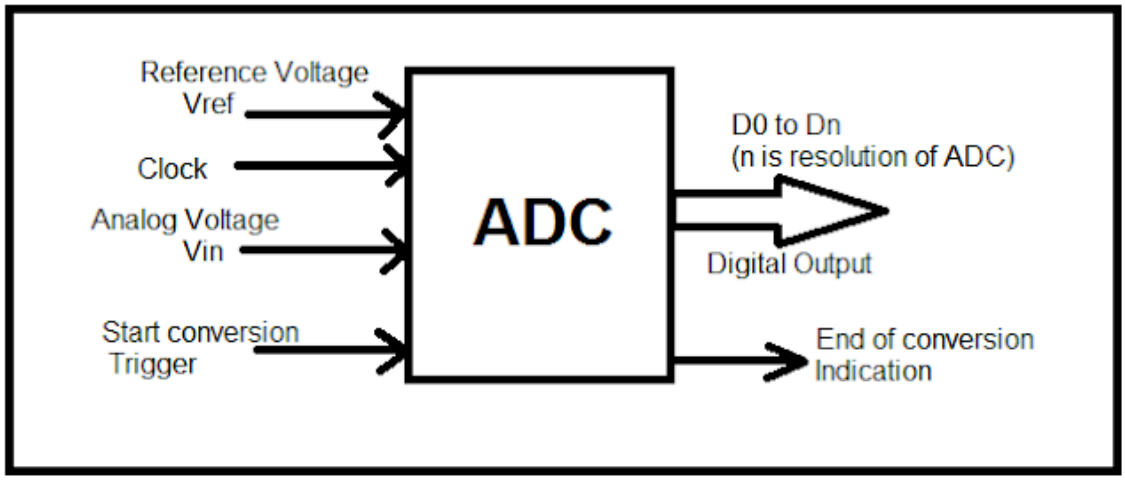
LABORATORY 6: TIVA – RPI Integration II

Part 1: ADC.

Theory Concepts:

Analog to Digital Converter or ADC is a module that allows to convert analog signals into digital ones. This is important when we work with embedded systems because most of the real world signals will be interpreted as analog signals.



UART uses a packet where a start bit is sent, then the data frame, then the parity bit (if used) and finally the stop bit.

How does the start, parity and stop bit works? Explain it with an example.

ADC in TIVA:

To use ADC in TIVA we will need to:

- Import libraries.
- Enable ADC ports and peripherals.
- Select ADC channel.
- Capture converted value.

Import libraries:

```
//Potenciometro mediante ADC//
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c1294ncpdt.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"
#include "driverlib/interrupt.h"
#include "driverlib/adc.h"
```

Enable ADC:

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
// Enable Port and pin for ADC//
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
//User Led pin declaration
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, 0x03);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, 0x11);
//Enables pin
GPIOPinTypeADC(GPIO_PORTK_BASE, 0x08);
```

The previous lines go inside the main function. We must enable the adc peripheral and define the pin type as adc. Which pins are we defining?

ADC configuration:

```
// Enable the first sample sequencer to capture the value of channel 0 when
// the processor trigger occurs. Sequencer 3 captures 1 sample
ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
//Configures the ADC step (interruption, end and channel 19)
ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_IE | ADC_CTL_END | ADC_CTL_CH19);
//Enables ADC Sequence
ADCSequenceEnable(ADC0_BASE, 3);
//Clears the ADC interruption
ADCIntClear(ADC0_BASE, 3);
```

Why do we use the channel 19? To select the channel, we must check the TM4C1294XL datasheet in the ADC section:

AIN12	4	PD3	I	Analog	Analog-to-digital converter input 12.
AIN13	3	PD2	I	Analog	Analog-to-digital converter input 13.
AIN14	2	PD1	I	Analog	Analog-to-digital converter input 14.
AIN15	1	PD0	I	Analog	Analog-to-digital converter input 15.
AIN16	18	PK0	I	Analog	Analog-to-digital converter input 16.
AIN17	19	PK1	I	Analog	Analog-to-digital converter input 17.
AIN18	20	PK2	I	Analog	Analog-to-digital converter input 18.
AIN19	21	PK3	I	Analog	Analog-to-digital converter input 19.

ADC use:

```
while(1){
    // Trigger the sample sequence.
    ADCProcessorTrigger(ADC0_BASE, 3);
    // Wait until the sample sequence has completed.
    while(!ADCIntStatus(ADC0_BASE, 3, false))
    {
    }
    //Clear the ADC interruption
    ADCIntClear(ADC0_BASE, 3);
    // Read the value from the ADC.
    ADCSequenceDataGet(ADC0_BASE, 3, &ui32Value);
    //Operates with the ADC values
    if (ui32Value > 2047)
    {
        GPIOPinWrite(GPIO_PORTN_BASE,0x03,0x03);
        GPIOPinWrite(GPIO_PORTF_BASE,0x11,0x11);
    }else{
        GPIOPinWrite(GPIO_PORTN_BASE,0x03,0x00);
        GPIOPinWrite(GPIO_PORTF_BASE,0x11,0x00);
    }
}
```

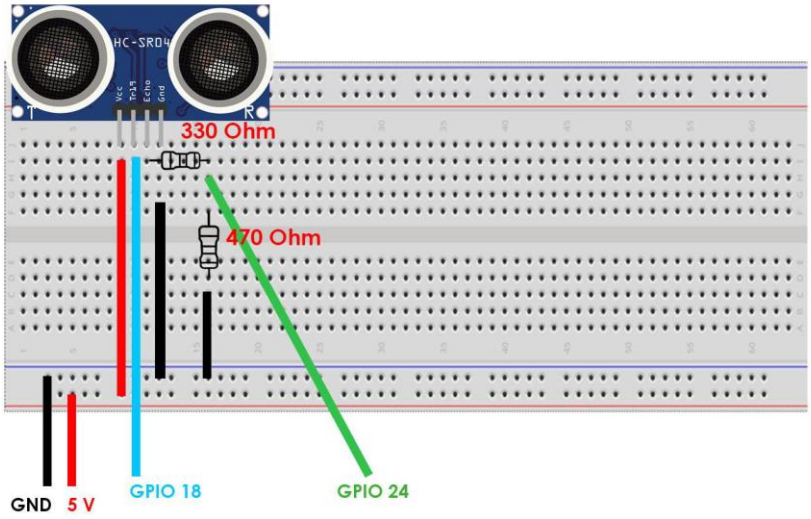
To use adc we need to know the TIVA resolution. With that resolution we can capture the converted value and use it.

- Which is the TIVA resolution?
- Can you give 3 examples of ADC use?

Exercises:

- **Connect a potentiometer to TIVA and make it vary the FS from a custom sequence using timer 0.**
- **Detect the distance of an object using the HC-SR04 ultrasonic sensor in Raspberry Pi. Then send the value of the distance to the Tiva with serial communication using UART. Finally follow the next instructions:**
    - **If the value of distance is above 10 cm, turn of all the user LEDs in the TIVA.**
    - **If the value of distance is between 10 – 8 cm, turn on the user LED PN1.**
    - **If the value of distance is between 8 – 6 cm, turn on the user LEDs PN1 and PN0.**
    - **If the value of distance is between 6 – 4 cm, turn on the user LEDs PN1, PN0 and PF4.**
    - **If the value of distance is below 4 cm, turn on all the user LEDs in the TIVA.**

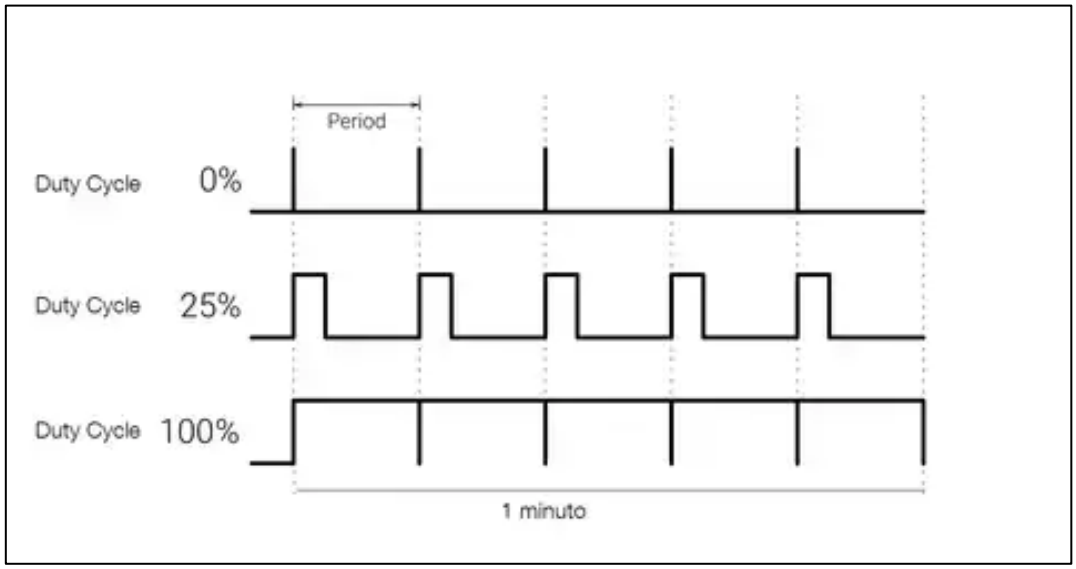
You can use the following diagram to connect HC-SR04:



Part 2: PWM.

Theory concepts:

PWM or pulse width modulation is a technique that help us to simulate analog values as output. This is very useful because it allows us to modify the standard behavior from different electronic components such as motors, LEDs, buzzers, etc.



PWM in TIVA:

To use pwm in tiva we must:

- Import pwm libraries.
- Enable peripherals.

- Define the clock tick radius.
- Implement pwm.

To import the pwm library we need:

```
//PWM//
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/pwm.h"

uint32_t reloj;
volatile uint32_t width;
```

Then we must enable the peripherals to work with pwm:

```
// Enable the PWM0 peripheral
SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2);
// Configure the PWM function for this pin.
GPIOPinConfigure(GPIO_PF1_M0PWM1);
GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);
// Wait for the PWM0 module to be ready.
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_PWM0))
{
}
```

Similarly, to ADC, we need to check TIVA datasheet PWM pins. We will find a table like this:

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type	Description
M0FAULT0	46	PF4 (6)	I	TTL	Motion Control Module 0 PWM Fault 0.
M0FAULT1	61	PK6 (6)	I	TTL	Motion Control Module 0 PWM Fault 1.
M0FAULT2	60	PK7 (6)	I	TTL	Motion Control Module 0 PWM Fault 2.
M0FAULT3	81	PL0 (6)	I	TTL	Motion Control Module 0 PWM Fault 3.
M0PWM0	42	PF0 (6)	O	TTL	Motion Control Module 0 PWM 0. This signal is controlled by Module 0 PWM Generator 0.
M0PWM1	43	PF1 (6)	O	TTL	Motion Control Module 0 PWM 1. This signal is controlled by Module 0 PWM Generator 0.
M0PWM2	44	PF2 (6)	O	TTL	Motion Control Module 0 PWM 2. This signal is controlled by Module 0 PWM Generator 1.

Is needed to configure PWM in TIVA, for that we have to calculate the clock tick calculus, we can use the following formula:

Clock tick calculus = target period * clock freq

```
// Configure the PWM generator for count down mode with immediate updates to the parameters.
PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
// Set the period. For a 50 KHz frequency, the period = 1/50,000, or 20 microseconds.
//For a 20 MHz clock, this translates to 400 clock ticks. Use this value to set the period.
PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, 400);
// Set the pulse width of PWM1 for a 75% duty cycle(width = 300).
width = 300;
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, width);
// Start the timers in generator 0.
PWMGenEnable(PWM0_BASE, PWM_GEN_0);
// Enable the outputs.
PWMOutputState(PWM0_BASE, (PWM_OUT_1_BIT), true);
```

In the example we calculate: 20MHz * 20u sec

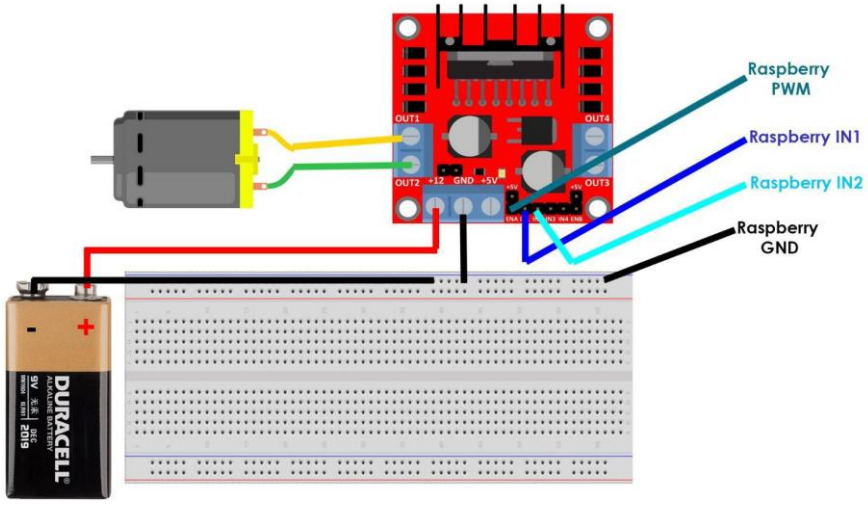
Finally, is important to manipulate the clock tick as our pulse width where the maximum pulse will be the total value of the clock tick.

```
while(1){
    for(width=0;width<=399;width++){
        GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2, GPIO_PIN_2);
        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, width);
        SysCtlDelay(9000000);
    }
}
```

How can we use PWM in raspberry pi? Explain the steps.

Exercises:

- **Using Pulse Width Modulation (PWM) change the speed of a DC motor following Figure 1. For this purpose, set the duty cycle to 75% and test the motor. Repeat this process with the following duty cycles: 25%, 45% and 50%. You can use the following schema for motor connection to raspberry Pi.**



- **Create a sequence using PWM to speed up a DC motor. The duty cycle of the motor must be initiated at 0% and every half of a second the value must increase by 1%. When the duty cycle is at his maximum, the value should return to 0.**
- **Using a potentiometer generate a PWM output and increase the intensity of an external LED (do not use any of the user LEDs) every half of a second. Finally, change the LED for a DC motor.**

Rasp – TIVA project

For the project mentioned in Lab 5:

- Add **HC-SR04** control so the robot moves until it is 3 cm near any object. Use TIVA to control **HC-SR04.**
- Modify the code so the robot motors are controlled by TIVA and the speed is managed using UART to communicate Rasp with TIVA.
- Create a repository for the project in your local Github. This repo must have 3 branches: main, roboV1 and roboV2. Then push it to your remote repository.
- Robov1 will contain all the development done for lab 5 and robov2 the changes applied from lab 6.
- Merge roboV1 in main, then try and merge robov2 in master, solve any conflicts if needed.
- Create a new branch called roboX from robov1, in this branch add changes for the LEDs, make any custom changes you prefer. Then, use the rebase command in roboV2 and roboX to change the base of the newest branch.
- What is rebase? When it should be used?