A white background with black dots

Description automatically generated

**LABORATORY MANUAL**

**Pwm AND TIMERS**

**aY23/24**

## OBJECTIVES

1. Learn the difference between polling and interrupts
2. Configure and implement hardware interrupts
3. Setup and configure the IR-based sensor as a wheel encoder

## EQUIPMENT

1. Computer or laptop that has the Pico C/C++ SDK toolchain installed
2. Raspberry Pi Pico W
3. HC-SR04 Ultrasonic sensor
4. L298N Motor Driver Board
5. DC motors x2
6. Batteries w/ battery pack
7. Micro-USB Cable

## NOTE:

1. Only students wearing fully covered shoes are allowed in the lab due to safety.
2. Bring your laptop with Code Composer Studio installed.
3. For your understanding and better quiz preparation, note the given tasks, especially questions or unexpected code behaviour.

# Introduction

Embedded systems can interact with their surrounding environment in many ways. In the previous lab, we learn there are two primary mechanisms initiating actions: **event-triggered** and **time-triggered**. We covered how to use **event-triggered** events to invoke ISR.

In this lab session, we shall learn how to use **time-triggered** events, also known as **Timers**. A timer is a specialised type of clock that is used to measure time intervals. A timer that counts from zero upwards for measuring the time elapsed is often called a stopwatch. It is a device that counts down from a specified time interval and generates a time delay.

A timer can also be used as a counter that keeps track of the number of times a particular event or process occurred regarding a clock signal. It is used to count the events happening outside the microcontroller. The counter can then be configured to produce a pulse with a specific duty cycle known as **pulse-width modulation (PWM)**. PWM timers are commonly used for motion control and dimming LEDs.

Lastly, we will develop code that incorporates timers for the **ultrasonic sensor** in measuring distance and PWM to adjust the speed of the **motor.**

# Introduction to Timers

Microcontrollers usually feature one or multiple types of timers to begin time-triggered actions after specific time intervals. The different types differ in their power consumption, counter register width and feature set. In this lab, we only use the high-level functions (e.g. alarms and repeating timer) provided by the SDK in “pico\_time” to trigger events.

The central component of a timer is a counter. A counter is a register whose value is incremented with every rising (or falling) edge of the clock input. The timer can be configured to trigger an interrupt every time the counter register rolls over (i.e. when it resets to 0 after reaching the highest counter value). Selecting different clock sources and dividing the clock frequency by a constant factor could achieve larger intervals between roll-overs without increasing the size of the counter register. However, this reduces the granularity since the counter value is incremented less frequently.

The timer peripheral on RP2040 supports the following features:

1. single 64-bit counter, incrementing once per microsecond
2. Latching two-stage read of counter, for race-free read over 32 bit bus
3. Four alarms: match on the lower 32 bits of counter, IRQ on match.

By default the timer uses a one microsecond reference that is generated in the Watchdog (see Section 4.8.2) which is derived from the clk\_ref.

The timer has 4 alarms, and can output a separate interrupt for each alarm. The alarms match on the lower 32 bits of the 64 bit counter which means they can be fired a maximum of 2^32 microseconds into the future. This is equivalent to:

1. 2^32 ÷ 10^6: ~4295 seconds
2. 4295 ÷ 60: ~72 minutes

# Alarms

Alarm functions for scheduling future execution. Alarms are added to alarm pools, which may hold a certain fixed number of active alarms. Each alarm pool utilizes one of four underlying hardware alarms, thus you may have up to four alarm pools. An alarm pool calls (except when the callback would happen before or during being set) the callback on the core from which the alarm pool was created. Callbacks are called from the hardware alarm IRQ handler, so care must be taken in their implementation.

A repeating timer simply creates an additional alarm automatically once the previous alarm is triggered.

There are various sample codes that can be found in the following links. Do look at the different implementation of timers in the Pico to achieve different effects.

Example Alarm implementation

<https://github.com/raspberrypi/pico-examples/blob/master/timer/hello_timer/hello_timer.c>

# Task 1: Configuring the HC-SR04 Ultrasonic Sensor

An ultrasonic sensor is a device that measures the distance to an object using ultrasonic sound

waves. It works by sending out a sound wave at a frequency above the range of human hearing.

This sensor measures the distance of a target by transmitting ultrasonic sound waves and then

converts the reflected sound into an electrical signal. The sensor determines the distance to a

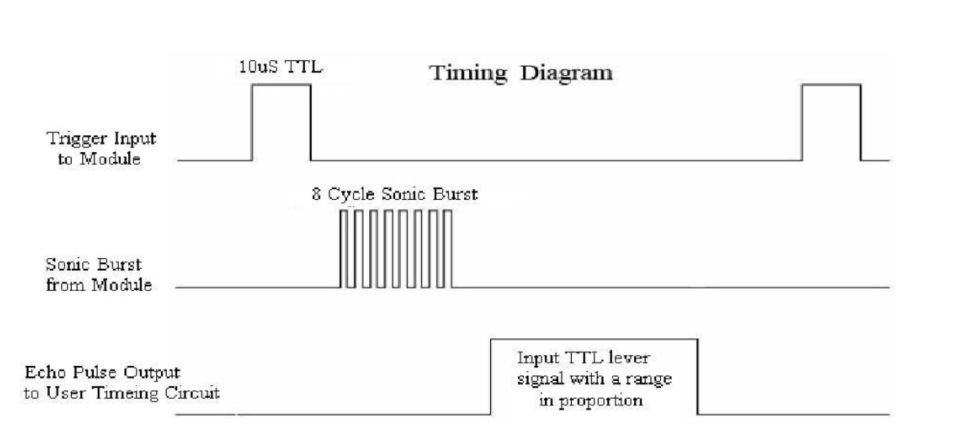
target by measuring time lapses between the sending and receiving of the ultrasonic wave. The

following is a brief feature list of the ultrasonic sensor. Further details can be obtained from

the datasheet on the following webpage.

The HC-SR04 ultrasonic range sensor features:

1. Input Voltage: 3.3V
2. Current Draw: 20mA (Max)
3. Digital Output: 3.3V
4. Digital Output: 0V (Low)
5. Working Temperature: -15°C to 70°C
6. Sensing Angle: 30° Cone
7. Angle of Effect: 15° Cone
8. Ultrasonic Frequency: 40kHz
9. Range: 2cm - 400cm

Figure 1: Timing diagram for HC-SR04

To begin obtaining the sensor data, the MSP432 needs to give a **trigger** signal of at least 10us pulse (top waveform), as seen in Figure 5. This triggers the sensor to transmit the **sonic-burst** pulses (middle waveform). Once the sensor receives the response, it will reflect the **echo** pin (bottom waveform). The width of the echo pulse correlates with the distance to the object in front of the sensor. Therefore, the longer the pulse, the farther the distance and vice-versa. Once we obtain the width of the pulse, the distance can be calculated using the following formula. This is possible as we know the velocity of sound in the air (340m/s).

Distance to an object (cm) = ((velocity of sound) x pulse-time-in-microseconds) / 2 = pulse-time-in-microseconds / 58

The following link offers a more straightforward illustration of the workings of the HC-SR04 ultrasonic sensor: <https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/>

# Task 1: Configuring the ultrasonic sensor

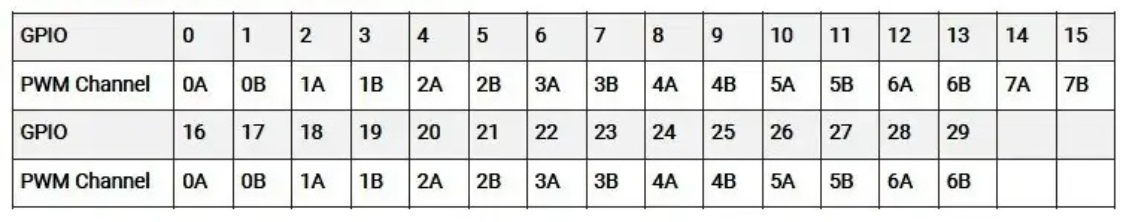
Obtaining sensor data is initiated by sending at least a 10-microsecond pulse via the Trigger pin to GPIO 2. **This can be accomplished by setting the output to high, and at a same time, create an alarm with a callback function where the output is set to low in 10-microsecond.**

Once the trigger pulse has been sent, the echo pulse needs to be captured and measured to obtain the pulse width. **This can be accomplished with an interrupt on the Echo Pin (GPIO 3), taking the time delta of the rising edge and falling edge. The current time can be simply retrieved via APIs provided by “pico\_time”**.

# Introduction to PWM

In the case of the Pico, the PWM lines are implemented using special PWM hardware. It has eight PWM generators, each capable of two PWM outputs. Any of the GPIO lines can be used as PWM lines and this means you can have up to 16 PWM lines in operation at any given time. Things are a little more complicated in that each pair of outputs has the same frequency which means you have eight, independently set, pairs of outputs. In addition, one of the outputs can be used as an input and this reduces the number of outputs available.

The PWM generators are assigned to GPIO pins in a fixed order:



For example, GPIO 0 is assigned to PWM slice 0 Channel A.

Through manipulation of the clock divider, wrap and channel level, one can create a PWM signal of varying frequencies and duty cycles. This is accomplished by using the hardware APIs provided by “hardware\_pwm”.

Refer to the following link that explains how each of the parameters affects the resultant waveform.

<https://www.i-programmer.info/programming/hardware/14849-the-pico-in-c-basic-pwm.html>

Example PWM implementation:

<https://github.com/raspberrypi/pico-examples/blob/master/pwm/hello_pwm/hello_pwm.c>

<https://github.com/raspberrypi/pico-examples/blob/master/pwm/led_fade/pwm_led_fade.c>

# Task 2: Configuring the Motor Driver

The L298N module is a high power motor driver module for driving DC and stepper motors. This module consists of an L298 motor driver IC and a 78M05 5V regulator. This module can control up to two DC motors with directional and speed control. The datasheet of the module can be found at the following link.

A diagram of a circuit board

Description automatically generated

Figure 2: Incorporating the motor driver and motor to the Pico.

There are two ways for this module to control the speed of the motor. An easier way would be to use a jumper across ENA and ENB to fix the voltage to the maximum. However, removing the jumper and connecting the pin to a PWM source would facilitate controlling the motor’s speed.

# Submission

Configure the setup to create a precision PWM with an specific duty cycle. The PWM period is 5ms (200Hz) and is output on GPIO 0 and 1 respectively for the Left and Right motor. Set the duty cycle on the GPIO 0 to be 80% while the output on GPIO 1 to be 40%. The left motor should spin twice as fast as the right motor.

# API documentations

<https://www.raspberrypi.com/documentation/pico-sdk/hardware.html#hardware_irq>

<https://www.raspberrypi.com/documentation/pico-sdk/hardware.html#hardware_pwm>

<https://www.raspberrypi.com/documentation/pico-sdk/high_level.html#pico_time>

<https://www.raspberrypi.com/documentation/pico-sdk/high_level.html#alarm>

<https://www.raspberrypi.com/documentation/pico-sdk/high_level.html#timestamp>