# Motor Driver

The motor driver board has 4 inputs – 2 for each motor. The inputs are:

IN1 – when high, the left wheel will turn forward (i.e. CCW)

IN2 – when high, the left wheel will turn backward (i.e. CW)

IN3 – when high, the right wheel will turn forward (i.e. CW)

IN4 – when high, the right wheel will turn backward (i.e. CCW)

It is best to use PWMs to drive the four motor control pins. Use a duty cycle of 0.01% for the off position, 100% for full speed, and something in-between for slower speeds.

The PWMs should use a frequency that is fast enough that the motors appear to turn smoothly instead of starting and stopping. A value 100Hz or higher works well.

To give you an idea of the speed, the test firmware uses a 50% duty cycle when driving the motors.

The pin mapping is:

|  |  |
| --- | --- |
| **Motor Driver Pin** | **PSoC Pin Name** |
| IN4 | CYBSP\_D8 |
| IN3 | CYBSP\_D9 |
| IN2 | CYBSP\_D10 |
| IN1 | CYBSP\_D11 |

# Servo Motor

The servo motor used to rotate the ultrasonic sensor has a range of 180˚. It is controlled using a pulse. The pulse width sets the angle as follows:

|  |  |
| --- | --- |
| **Pulse width (µs)** | **Angle (degrees)** |
| 2500 | -90 |
| 2000 | -45 |
| 1500 | 0 |
| 1000 | +45 |
| 500 | +90 |

The easiest way to set the angle is to use a PWM. Set the appropriate pulse width for the PWM to move the servo to the desired position.

You can either use a continuously running PWM (which will hold the servo actively in the desired position) or a one-shot PWM (which will set the servo to the desired position each time it is started and then release it so that it can be moved manually).

You can use the "cyhal\_pwm\_set\_period" function to easily set the desired pulse width. (You can keep the period set at a fixed value such as 5,000us and just adjust the period as necessary).

The pin mapping is:

|  |  |
| --- | --- |
| **Signal** | **PSoC Pin Name** |
| Angle Signal | CYBSP\_A4 |

# Ultrasonic Sensor

To use the ultrasonic sensor, you send out a pulse on the "Trig" line of 10us and look for a return pulse on the "Echo" line. The wider the return Echo pulse is, the greater the distance is.

The Echo line pulse width will vary from ~100us to ~10000us (10ms). A wall 1 foot from the sensor will produce a pulse of about 1500us.

To drive the Trig line, there are (at least) two options:

1. Use a GPIO configured as an output with strong drive. Set the pin high, wait 10us, and then set the pin low.
2. Use a PWM configured as a one-shot with a pulse width of 10us and start it each time you want to make a measurement.
   1. You can use the "cyhal\_pwm\_set\_period" function to set the desired pulse width.

One simple way to measure the Echo line pulse width is to use a timer:

Configure the timer as a one-shot up counter. Configure the Count input of the timer as a level signal and connect it to the Echo line.

Reset the timer to 0 and start it before sending the Trig pulse.

Because the Echo input is connected to the Count input of the timer as a level signal, the timer will count up when the Echo line goes high and will continue to count until the Echo line goes low again.

Once the timer completes its full period, read the count value to determine the width of the Echo pulse. You can either wait long enough for the timer to be done, or you configure the timer to generate an interrupt when it reaches its terminal count.

One option is to use a separate RTOS task to handle all of the distance measurement operations. In the task's infinite loop, you can:

1. Reset and start the timer.
2. Send the Trig pulse.
3. Use an RTOS delay to wait until the timer is done.
4. Read the Count value from the timer.
5. Use an RTOS Queue to send the value where it is needed, or use a Semaphore to inform another task that a new value is ready.
   1. If you use a Queue, you might want to use a Queue with a depth of 1 and force the new value into the Queue even if a previous value hasn't been read yet. That makes the latest value always available to any other thread that needs it.
6. Use an RTOS delay to wait until the next measurement is needed.

The frequency for the timer should be such that an appropriate resolution is obtained on the pulse width. For example, a timer with a clock of 1MHz and a period of 10000 will have a resolution of 1us and will measure a maximum time period of 10ms. You can divide the count by 128 to get a result that is approximately in inches.

The pin mapping is:

|  |  |
| --- | --- |
| **Ultrasonic Sensor Pin** | **PSoC Pin Name** |
| Trig | CYBSP\_D13 |
| Echo | CYBSP\_D12 |

# Rotary Encoders and Optical Sensors

The rotary encoders have 20 slots in them that let light through. The optical sensor has a digital output that is high when light passes through the slots and is low when light is blocked.

Therefore, to measure distance, you just need to monitor the input signal for transitions from 0 to 1 (the opposite would work equally well).

This can be done (at least) two ways

1. Use a digital input pin with a rising edge interrupt. In the interrupt, just increment a variable to keep track of the number of transitions. A count of 20 means the wheel has gone through one complete revolution.
2. Use a counter where the Count input is configured as rising edge triggered. The counter will keep track of the number of transitions which can be read by the firmware whenever a count is needed.

The encoders and optical sensors can be used to move a specific distance – as little as 1/20th of the circumference of the wheels, which is about 1cm.

If you want even finer resolution, you can use an interrupt or count on both edges – that will give you 40 transitions for each revolution of the wheel.

The rotary encoders and optical sensors can also be used to monitor and tune the left and right motors so that the car drives perfectly straight when it is supposed to. Without this, variation in the motors may cause the care to drift to one side.

Note that these encoders and sensors can't tell the difference between forward and backward movement. To do that, we would need a quadrature encoder/decoder. Quadrature encoders have 2 sensors that are 90˚ out of phase so that you can tell which way the wheel is turning. That isn't necessary in our case since we are the ones controlling the wheel so we already know which way it is turning.

The pin mapping is:

|  |  |
| --- | --- |
| **Optical Sensor** | **PSoC Pin Name** |
| Left | CYBSP\_D7 |
| Right | CYBSP\_D2 |