Last Revision: 12 Oct 2012

RS232IO Controller

Manual

Compiled by: Grant Phillips
Riaan Ehlers

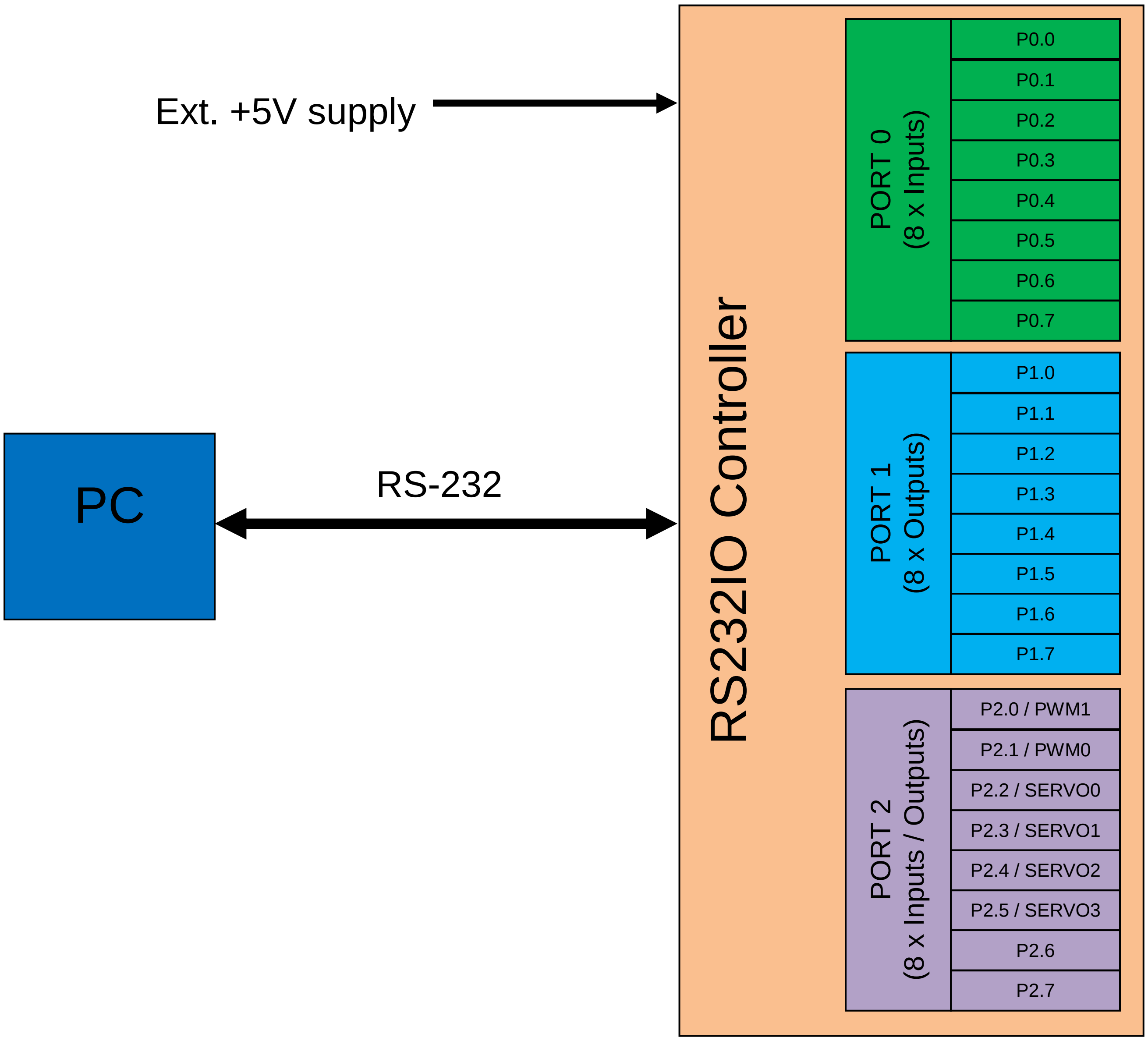
1. Overview

The RS232IO Controller is a device that allows a computer to interface to electronic circuits/devices using a standard RS232 port. It has the following features:

- 1 x 8-bit output port
- 1 x 8-bit input port
- 1 x 8-bit input/output port with PWM and SERVO alternate functions
- 5 x Analog input channels
- 2 x PWM output channels
- 4 x Servo Motor output channels

2. Architecture

The RS232IO Controller architecture can is summarized in the following block diagram:



Port0
Normal 8-bit input port. The inputs are labeled P0.0 (LSB) to P0.7 (MSB). 0V = LOW and +5V is HIGH.

Port1
Normal 8-bit output port. The outputs are labeled P1.0 (LSB) to P1.7 (MSB). The outputs can only provide a 0 (0V) or a 1 (+5V). Each output can source a maximum current of 25mA.

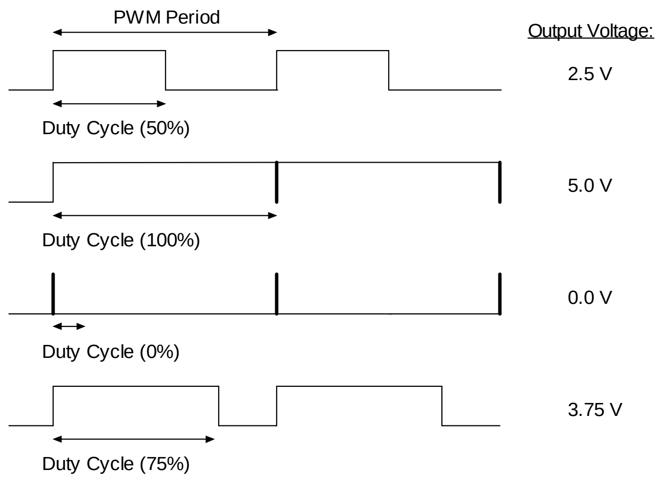
Port2
An 8-bit input/output port. The IO pins are labeled P2.0 (LSB) to P2.7 (MSB). The PWM and SERVO outputs are multiplexed with the P2.0 to P2.5 pins. This means that the “mode” for each pin can be set individually to:

0 = the pin is an output
1 = the pin is an input
2 = the pin takes on the alternate function and is ignored as an input or output pin. E.g. if the mode for P2.0 is set to 2, then it become the PWM1 output and cannot be used as a input or output pin.

The outputs can only provide a 0 (0V) or a 1 (+5V) and each output can also source a maximum current of 25mA.

Analog Channels 0 to 4
An analog channel to indicate the voltage level on its input. The voltage level is indicated as a 10-bit value. If the input value is 0, it indicates that the input voltage is 0V. If the input value is 1023, it means that the input voltage is +5V. This means that for every 1 that the input value changes, it indicates a +/- 5mV change on the input pin.

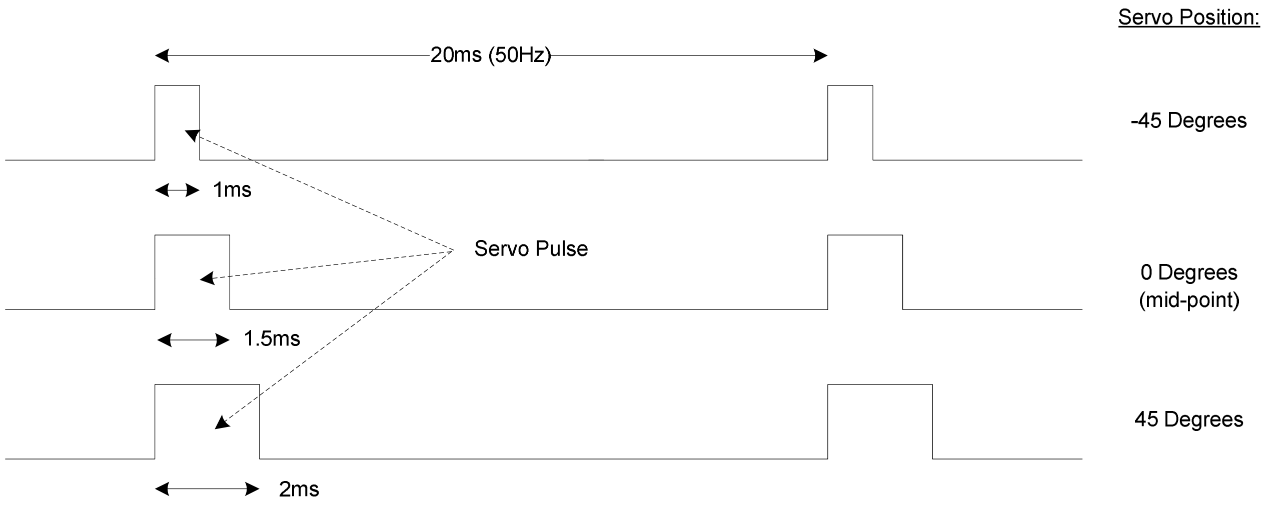
PWM Outputs 0 & 1
A PWM (Pulse Width Modulation) output is used to provide an analog output. E.g. a PWM output could be connected to a light and by changing the PWM duty cycle, the brightness of the light could be varied. When using a PWM output, the user must set the duty cycle which is represented as a 10-bit value of 0 – 1020 (0% - 100%). The operation is explained in the following diagram.



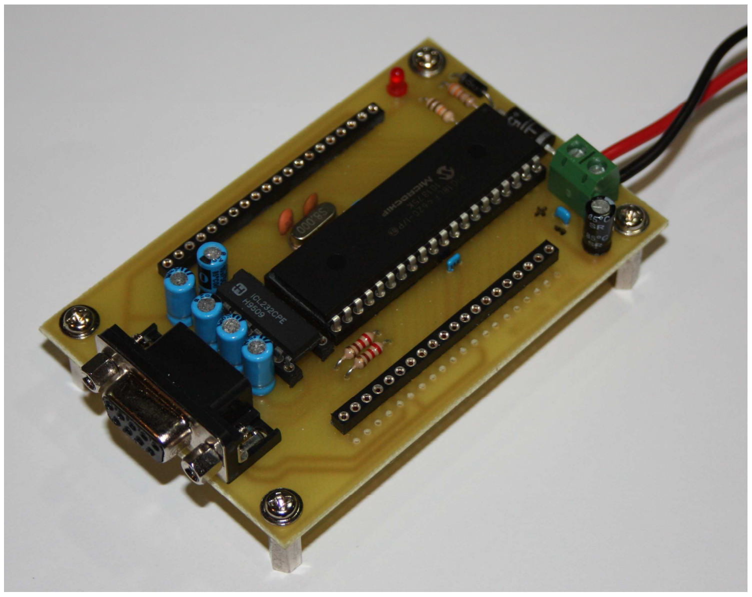
The frequency values for the PWM modules are fixed at 33.3Khz due to hardware limitations.

SERVO Outputs 0 to 3

A servo output is used to control the position of a servo motor. Generally a servo needs a constant frequency input signal of 50Hz (20ms) and then the position of the servo is determined by the duration of the HIGH pulse. When the pulse is 1.5ms, then the servo will be more or less at the midpoint. Generally servo motors can turn through 90˚, which means that a 1ms and a 2ms pulse will move it to the furthest positions. Sometimes these servos can be pushed beyond these theoretical limits and sending it a pulse of less than 1ms or more than 2ms will turn it further than just the 90˚. PLEASE TAKE CARE WHEN TESTING THIS AND TRY TO INCREMENT THE PULSE IN SMALL STEPS TO "TEST" WHERE THE LIMIT IS FOR YOUR PARTICULAR SERVO. It is always safe to start with a 1.5ms pulse which will move it to its mid-position.



3. Hardware



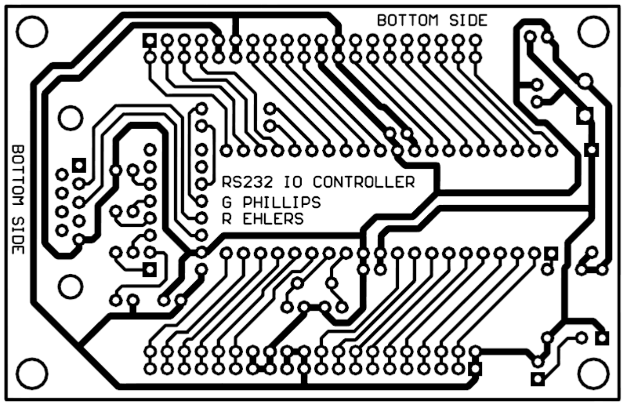
3.1 Component List

The following components are used for the RS232IO Controller:

Qty	Component	Schematic Ref	
1	2-pin PCB mount screw terminal	J1	
1	DB9 Female Right/Angle PCB mount connector	J2	
1	DB9 Female Plug + Cover		*1
1	DB9 Male Plug + Cover		*1
1	+/- 1m multicore cable for RS232 comms		*1
1	3mm/5mm LED	D1	
1	1N6267	D2	*2
1	1N4007	D3	*3
1	330R 1/4W resistor	R1	
1	10K 1/4W resistor	R2	
2	220R 1/4W resistor	R3, R4	
1	470uF Electrolytic Capacitor 16V	C1	
5	10uF Electrolytic Capacitor 16V	C5 - C9	
2	33pF Ceramic Capacitors	C10, C11	
3	100nF Ceramic Capacitors	C2, C3	
1	8Mhz Crystal	X1	*4
1	RS232IO Controller (PIC18F4620)	U1	*5
1	MAX232	U2	
1	DIP40 IC Holder (for RS232IO Controller)		
1	DIP16 IC Holder (for MAX232)		

*1 These are required to make up the RS-232 cable to communicate between the PC and the RS232IO Controller
*2 Optional - only to prevent overvoltage in the event of a incorrect power supply
*3 Optional - only to prevent reverse polarity when power is connected the wrong way around
*4 Must be EXACTLY 8Mhz
*5 You can get this from Mr Ehlers and he will program it with the latest Firmware version

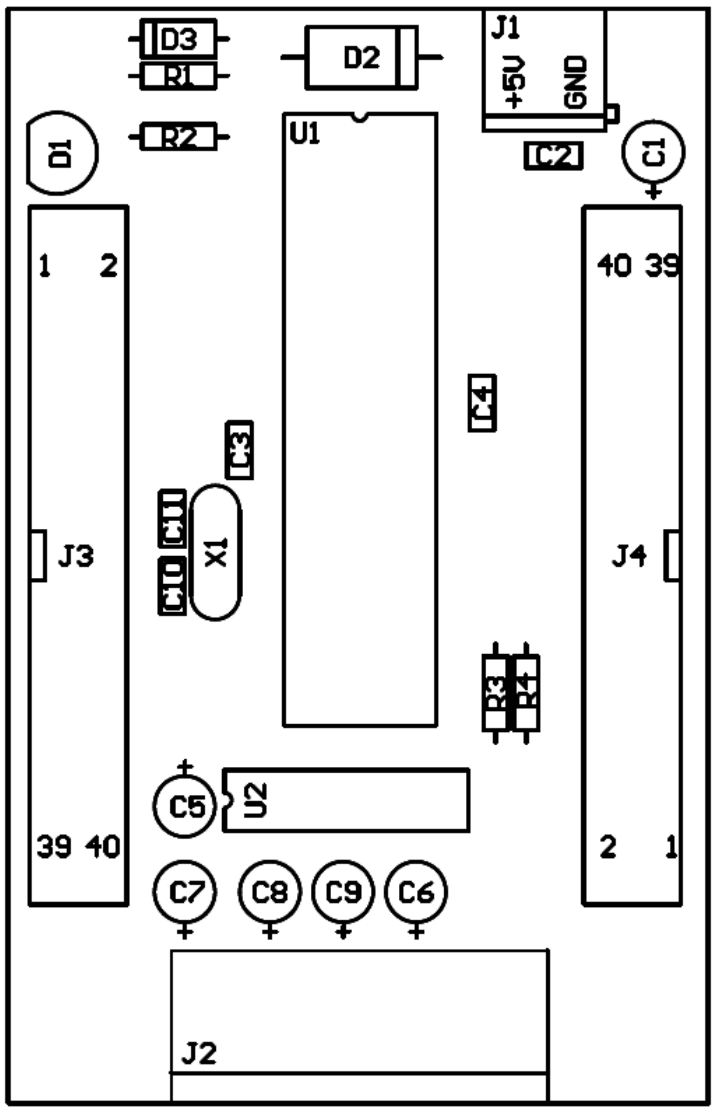
3.2 PCB



See the “RS232IO Controller PCB.pdf” document which contains the actual size of the PCB which can be taken to any PCB manufacturer. The drill sizes for the holes are +/- 1mm, so it is probably best to ask the manufacturers to drill it for you too.

3.3 PCB Layout

The components are arranged as follows on the PCB (seen from the top):



Make sure to place the Diodes the correct way around and take special care when placing the capacitors to have the + sides the correct way around too.

Note that connectors J3 and J4 provide the connections to your external circuits. Also note that PIN 1 for J3 is AT THE TOP, where PIN 1 for J4 is AT THE BOTTOM.

3.4 PCB Layout

The connections for J3 and J4 are as follows:

J3

+5V out	1	2	+5V out
ANALOG0	3	4	ANALOG0
ANALOG1	5	6	ANALOG1
ANALOG2	7	8	ANALOG2
ANALOG3	9	10	ANALOG3
N/A	11	12	N/A
ANALOG4	13	14	ANALOG4
P2.5 / SERVO3	15	16	P2.5 / SERVO3
P2.6	17	18	P2.6
P2.7	19	20	P2.7
+5V out	21	22	+5V out
+5V out	23	24	+5V out
Gnd	25	26	Gnd
Gnd	27	28	Gnd
N/A	29	30	N/A
P2.0 / PWM1	31	32	P2.0 / PWM1
P2.1 / PWM0	33	34	P2.1 / PWM0
P2.2 / SERVO0	35	36	P2.2 / SERVO0
P1.0	37	38	P1.0
P1.1	39	40	P1.1

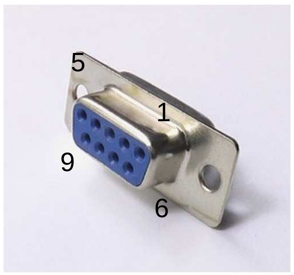
J4

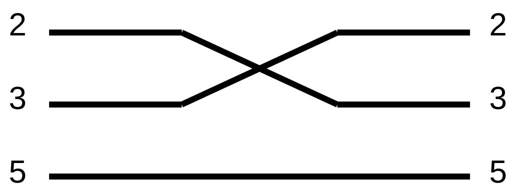
P0.7	40	39	P0.7
P0.6	38	37	P0.6
P0.5	36	35	P0.5
P0.4	34	33	P0.4
P0.3	32	31	P0.3
P0.2	30	29	P0.2
P0.1	28	27	P0.1
P0.0	26	25	P0.0
+5V out	24	23	+5V out
Gnd	22	21	Gnd
P1.7	20	19	P1.7
P1.6	18	17	P1.6
P1.5	16	15	P1.5
P1.4	14	13	P1.4
+5V out	12	11	+5V out
Gnd	10	9	Gnd
P2.4 / SERVO2	8	7	P2.4 / SERVO2
P2.3 / SERVO1	6	5	P2.3 / SERVO1
P1.3	4	3	P1.3
P1.2	2	1	P1.2

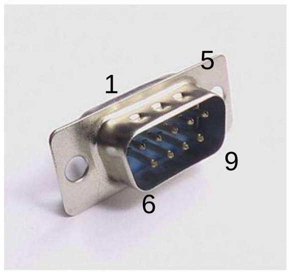
The +5V out pins can supply your external circuits with +5V from the supply that is connected to the RS232IO Controller – do not connect a +5V supply from another source to these pins! If you have external circuits with different power supplies, try to connect one of the Gnd pins to those power supplies' GND pins, i.e. all the GND lines for the supplies must be common.

3.5 RS-232 Cable

The serial cable between the PC and the RS232IO Controller should be wired as follows:







4. RS232IO.dll

The RS232IO Control is a DLL (Dynamic Linked Library) that is used in Visual Basic to give a user access to the RS232IO Controller. The user will then use this control to write to outputs, read inputs, read analogs, and write PWM and SERVO values to the ports of the RS232IO controller.

4.1 Installing the DLL and integrate it into Visual Basic

To install the RS232IO.dll and use it in Visual Basic, do the following steps:

1. Create a "c:\RS232IO" folder.

2. Copy the contents of the "RS232IODLL.zip" file to the "c:\RS232IO" folder.

3. In the General area of the Toolbox (at the bottom of the list), right-click

4. Select "Choose Items" (this may take a while to bring up the next window)

5. Click "Browse"

6. Go to the "c:\RS232IO" folder and select "RS232IO.dll"

7. Click "Open"

8. Select the RS232IO control under General in the Toolbox and add to the form like you would normally add other controls.

4.2 Using the RS232IO control in Visual Basic

The following section summarizes the methods that can be used with the control.

General methods:

1. **`StartIOConnection(ComPortNumber, P2_7_mode, P2_6_mode, P2_5_mode, P2_4_mode, P2_3_mode, P2_2_mode, P2_1_mode, P2_0_mode,)`**
   Start communication with the RS232IO Controller on a specific COM port and sets the mode for the individual port pins on Port 2.

Parameters:	ComPortNumber	-	**0** to **20** (usually **1**)
	P2_7_mode	-	**0** = pin is an output
			1 = pin is an input

P2_6_mode	-	**0** = pin is an output
		1 = pin is an input
P2_5_mode	-	**0** = pin is an output
		1 = pin is an input
		2 = pin is used for SERVO3
P2_4_mode	-	**0** = pin is an output
		1 = pin is an input
		2 = pin is used for SERVO2
P2_3_mode	-	**0** = pin is an output
		1 = pin is an input
		2 = pin is used for SERVO1
P2_2_mode	-	**0** = pin is an output
		1 = pin is an input
		2 = pin is used for SERVO0
P2_1_mode	-	**0** = pin is an output
		1 = pin is an input
		2 = pin is used for PWM0
P2_0_mode	-	**0** = pin is an output
		1 = pin is an input
		2 = pin is used for PWM1

2. **`GetVersion()`**

Reads the current Firmware version from the RS232 IO Controller.

Returns: A **string** value (e.g. "v.2.0.1")

IOPort methods:

1. **`WriteByteIOPort(PortNumber, Value)`**

Writes a byte (8-bit) value to an output port. The pins on Port 2 that are configured as inputs or as their alternate functions, will be ignored when writing the byte to this port.

Parameters:	PortNumber	-	**1** (for Port 1) or **2** (for Port 2)
	Value	-	**byte** value (0 – 255)

2. **`WriteBitIOPort(PortNumber, Bit, Value)`**

Writes to a specific bit on an output port. The pins on Port 2 that are configured as inputs or as their alternate functions, will be ignored when writing to that bit.

Parameters:	PortNumber	-	**1** (for Port 1) or **2** (for Port 2)
	Bit	-	**0** (LSB) to **7** (MSB)
	Value	-	**0** (LOW) or **1** (HIGH)

3. **`ToggleBitIOPort(PortNumber, Bit)`**

Toggles a specific output bit between HIGH and LOW on an output port. The pins on Port 2 that are configured as inputs or as their alternate functions, will be ignored when toggling that bit.

Parameters:	PortNumber	-	**1** (for Port 1) or **2** (for Port 2)
	Bit	-	**0** (LSB) to **7** (MSB)

4. **`ReadByteIOPort(PortNumber)`**

Reads a byte (8-bit) value from an input port. The pins on Port 2 that are configured as outputs or as their alternate functions, will be ignored when reading from this port.

Parameters:	PortNumber	-	**0** (for Port 0) or **2** (for Port 2)

Returns: A **byte** value (0 – 255)

5. **`ReadBitIOPort(PortNumber, Bit)`**

Reads the status (HIGH or LOW) of a specific bit on an input port. The pins on Port 2 that are configured as outputs or as their alternate functions, will be ignored when reading from that bit.

Parameters:	PortNumber	-	**0** (for Port 0) or **2** (for Port 2)
	Bit	-	**0** (LSB) to **7** (MSB)

Returns: **0** (LOW) or **1** (HIGH)

Analog Input methods:

1. **`ReadAnalogChannel(Channel)`**

Reads an integer value (10-bit) from an Analog input channel.

Parameters:	Channel	-	**0** to **4**

Returns: **integer** value (**0** to **1023**) – represents a voltage of 0V to 5V

PWM methods:

1. **`PWM0_SetDutyCycleAsPercentage(DutyCyclePercentage)`**

Sets the duty cycle for the PWM0 module as a percentage value.

Parameters:	DutyCyclePercentage	-	**0** (0%) to **100** (100%)

2. **`PWM0_SetDutyCycleAs10bitValue(DutyCycle10bitValue)`**

Sets the duty cycle for the PWM0 module as a 10-bit value to produce a more accurate pulse.

Parameters:	DutyCycle10bitValue	-	**0** (0%) to **1020** (100%)

3. **`PWM1_SetDutyCycleAsPercentage (DutyCyclePercentage)`**
   Sets the duty cycle for the PWM1 module as a percentage value.

   Parameters: DutyCyclePercentage - **0** (0%) to **100** (100%)

4. **`PWM1_SetDutyCycleAs10bitValue(DutyCycle10bitValue)`**
   Sets the duty cycle for the PWM1 module as a 10-bit value to produce a more accurate pulse.

   Parameters: DutyCycle10bitValue - **0** (0%) to **1020** (100%)

5. **`PWM0_Start()`**
   Starts (enables) the PWM0 output pulse.

6. **`PWM1_Start()`**
   Stops (disables) the PWM0 output pulse.

7. **`PWM0_Stop()`**
   Starts (enables) the PWM1 output pulse.

8. **`PWM1_Stop()`**
   Stops (disables) the PWM1 output pulse.

SERVO methods:

1. **`SERVO0_SetPulse(PulseWidthInMicroseconds)`**
   Sets the HIGH pulse duration for the SERVO0 output. The parameter represents a microsecond delay value.

   Parameters: PulseWidthInMicroseconds - **200** to **2800**

2. **`SERVO1_SetPulse(PulseWidthInMicroseconds)`**
   Sets the HIGH pulse duration for the SERVO1 output. The parameter represents a microsecond delay value.

   Parameters: PulseWidthInMicroseconds - **200** to **2800**

3. **`SERVO2_SetPulse(PulseWidthInMicroseconds)`**
   Sets the HIGH pulse duration for the SERVO2 output. The parameter represents a microsecond delay value.

   Parameters: PulseWidthInMicroseconds - **200** to **2800**

4. **`SERVO3_SetPulse(PulseWidthInMicroseconds)`**
Sets the HIGH pulse duration for the SERVO3 output. The parameter represents a microsecond delay value.

Parameters: PulseWidthInMicroseconds - **200** to **2800**

5. **`SERVO0_Start()`**
Starts (enables) the SERVO0 output pulse.

6. **`SERVO1_Start()`**
Starts (enables) the SERVO1 output pulse.

7. **`SERVO2_Start()`**
Starts (enables) the SERVO2 output pulse.

8. **`SERVO3_Start()`**
Starts (enables) the SERVO3 output pulse.

9. **`SERVO0_Stop()`**
Stops (disables) the SERVO0 output pulse.

10. **`SERVO1_Stop()`**
Stops (disables) the SERVO1 output pulse.

11. **`SERVO2_Stop()`**
Stops (disables) the SERVO2 output pulse.

12. **`SERVO3_Stop()`**
Stops (disables) the SERVO3 output pulse.