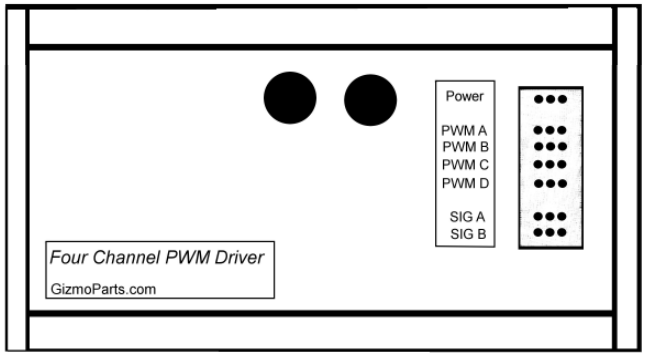
GPUX

Four Channel PWM Driver



USB to R/C PWM Four Channel Driver
With Dual Signal I/O
V1.0

Gizmo Parts
www.gizmoparts.com

Introduction

The GPUX is a converter that connects any four R/C devices to a computer via the USB interface. The converter appears as a standard serial (COM) port to the computer, and as a generic four channel R/C receiver. Any standard R/C device is compatible with the GPUX including servos and motor controllers.

In addition, the GPUX has two digital general purpose I/Os that may be configured independently as either inputs or outputs for interface non-R/C devices.

GPUX Specifications and Features

- Four Independent R/C Channel Outputs
- Two General Purpose I/O's
- USB 2.0 Full Speed Compatible
- Electrically Isolated USB Interface
- Power Input: 4.5-5V

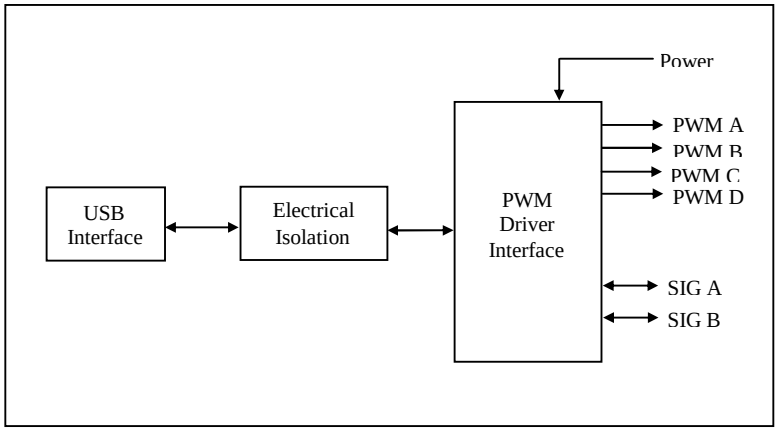


Figure 1 GPUX Block Diagram

The GPUX is composed of three major blocks: USB Interface, Electrical Isolation Circuitry, and PWM Driver Interface. The USB Interface is compatible with any computer equipped with an USB port. When the GPUX is used with a Microsoft operating system it appears as a COM port. Under Linux, the GPUX appears as ttyUSBx, where x is a number.

The Electrical Isolation circuitry separates power and ground signals between the computer and the PWM devices connected to the GPUX. From the computer's perspective, the GPUX is a bus-powered device; even though, no power is used from the computer for the PWM devices. The isolation also helps to protect the computer from possible damage caused by electrical problems on the device interface side.

The PWM Driver Interface has four independently programmable PWM outputs and two general purpose input/output TTL ports. Each PWM output can generate a signal with a pulse width ranging from 0 to 2 mSec.

If for some reason, the converter stops functioning, simply contact Gizmo Parts for a repair estimate and return authorization.

LIMITED WARRANTY

THIS MANUAL AND REFERENCE MATERIALS ARE SOLD "AS IS," WITHOUT WARRANTY AS TO THEIR PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. YOU ASSUME THE ENTIRE RISK AS TO THE RESULT AND PERFORMANCE OF THE CONVERTER.

HOWEVER, TO THE ORIGINAL PURCHASER ONLY, THE MANUFACTURER WARRANTS THE CONVERTER TO BE FREE FROM MANUFACTURING DEFECTS AND FAULTY WORKMANSHIP UNDER NORMAL USE FOR A PERIOD THIRTY DAYS FROM THE DATE OF PURCHASE. IF FAILURE OF THE CONVERTER HAS RESULTED FROM ACCIDENT OR ABUSE THE MANUFACTURER SHALL HAVE NO RESPONSIBILITY TO REPLACE THE CONVERTER UNDER THE TERMS OF THIS LIMITED WARRANTY.

Notes

The two general purpose signals can be configured as either inputs or outputs. The outputs are capable of only supplying a 1 to 2 mA; therefore, additional buffering may be necessary to control components such as relays or other components that require larger sink or source currents.

System Requirements

The GPUX was tested with both Windows XP and Linux, and normally requires no additional software for either operating system. The USB interface uses FTDI Chip's FT232RL, and if an application requires additional or custom driver software, please visit FTDI Chips' web site: www.ftdichip.com.

Connections and Controls

Figure 2 shows all the GPUX external connections.

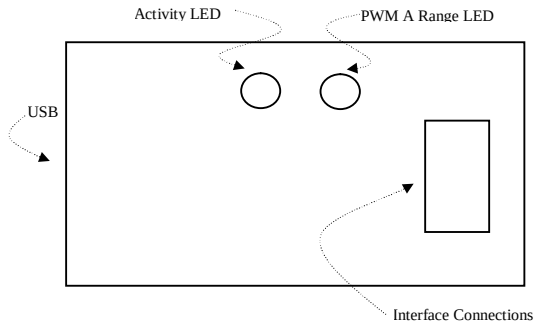


Figure 2 GPUX LEDs and Connections

- *Activity LED* – Blinks at a regular interval to indicate the presence of power and flickers to indicate the presence of serial data between the computer and converter.
- *PWM A Range LED* – Indicates the programmed pulse width for PWM A. If the programmed width is 1.5 mSec or longer, then the LED is ON.

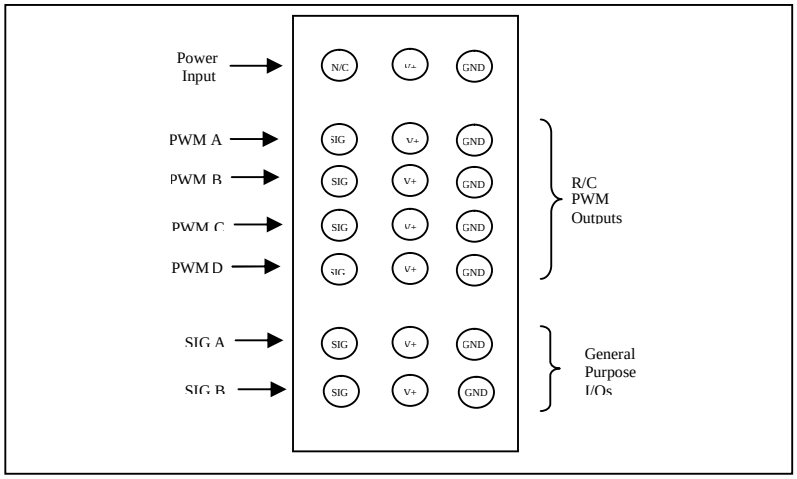


Figure 3 Connector Pinout

Figure 3 shows all the GPUX's interface pins. Normally, power is supplied to the GPUX through the dedicated Power connector, which is located at the top; however, power may be supplied to the GPUX through any of the pins labeled V+ and GND. The next four rows contain the connectors for the PWM drivers. The bottom two connectors are the general purpose I/O connectors.

Not pictured in Figure 2 or Figure 3 is a two position DIP switch. This switch is located inside the GPUX and controls the direction of the general purpose I/Os as shown in Table 1. The GPUX ships with both switches in the OFF position (input mode). To change the settings, remove the four screws located on the bottom of the GPUX and make the appropriate switch changes.

Table 1 General Purpose I/Os Switch Settings

Switch	On	Off
1	Signal A Output	Signal A Input
2	Signal B Output	Signal B Input


```
            }

            Console.Write("COM port({0}): ", defaultPortName);
            portName = Console.ReadLine();

            if (portName == "")
            {
                portName = defaultPortName;
            }
            return portName;
        }
    }
```

```
        }
        else if (stringComparer.Equals("mid", message))
        {
            buffer[0] = (byte)':';
            buffer[1] = 192;
            buffer[2] = 192;
            buffer[3] = 192;
            buffer[4] = 192;
            _serialPort.Write(buffer, 0, 5);
        }
        else if (stringComparer.Equals("max", message))
        {
            buffer[0] = (byte)':';
            buffer[1] = 255;
            buffer[2] = 255;
            buffer[3] = 255;
            buffer[4] = 255;
            _serialPort.Write(buffer, 0, 5);
        }
        else if (stringComparer.Equals("ramp", message))
        {
            for (int i = 127; i < 255; i++)
            {
                buffer[0] = (byte)':';
                buffer[1] = (byte)i;
                buffer[2] = 164;
                buffer[3] = 191;
                buffer[4] = (byte)(255 + 127 - i);
                _serialPort.Write(buffer, 0, 5);

            }
            for (int i = 255; i >= 127; i--)
            {
                buffer[0] = (byte)':';
                buffer[1] = (byte)i;
                buffer[2] = 164;
                buffer[3] = 191;
                buffer[4] = (byte)(255 + 127 - i);
                _serialPort.Write(buffer, 0, 5);

            }
        }
        else
        {
            _serialPort.Write(message);
        }
    }

    readThread.Join();
    _serialPort.Close();
}

public static void Read()
{
    while (_continue)
    {
        try
        {
            _serialPort.Read(rbuffer, 0, 1);
            Console.Write("{0} ", rbuffer[0]);       // print value
            Console.Write("{0}\n", Convert.ToChar(rbuffer[0]));
        }
        catch (TimeoutException) { }
    }
}

public static string SetPortName(string defaultPortName)
{
    string portName;

    Console.WriteLine("Available Ports:");
    foreach (string s in SerialPort.GetPortNames())
    {
        Console.WriteLine("   {0}", s);
```


Protocol Specification

Data is transferred between the computer and the GPUX using a simple communications protocol. The protocol consists of four command types:

1. Version – Used to request the firmware version.
2. PWM – Used to set the PWM output pulse widths.
3. Signal Input – Used to read the signal levels of the general purpose I/Os.
4. Signal Output – Used to write the signal levels of the general purpose I/Os.

The tables listed in this section show the expected formats for all the supported commands. If the GPUX fails to recognize a command, it will respond with a NACK, 78 (0x4E), to indicate it has received a character which it does not understand.

Table 2 Version Command

Computer Data	GPUX Data	Comment
0x56		'V' - Version Command
	0x41	'A' - Acknowledge
	G	GPUX Message
	P	
	U	
	X	
	0x20	(space)
	V	
	1	
	.	
	0	
	0x10	New Line

The PWM Command is used to output widths of the PWM signals. The width is determined by the following equation:

Pulse Width = 2.0 mSec * (Byte X) / 255.

Table 3 PWM Command

Computer Data	GPUX Data	Comment
0x3a		':' - PWM Command
(Byte 0)		PWM A Setting
(Byte 1)		PWM B Setting
(Byte 2)		PWM C Setting
(Byte 3)		PWM D Setting
	0x41	'A' - Acknowledge

Table 4 Signal Input Command

Computer Data	GPUX Data	Comment
0x49		'I' – Input Command
	0x41	'A' - Acknowledge
	(Byte)	Bit 0 – Signal A Bit 1 – Signal B Bits 2-7 = 0

Table 5 Signal Output Command

Computer Data	GPUX Data	Comment
0x4F		'O' – Input Command
	0x41	'A' - Acknowledge
(Byte)		Bit 0 – Signal A Bit 1 – Signal B Bits 2-7 = 0

Example Program

Listing 1 Example C# GPUX Control Program

```
using System;
using System.IO.Ports;
using System.Threading;

public class PortChat
{
    static bool _continue;
    static SerialPort _serialPort;

    static byte[] rbuffer;

    public static void Main(string[] args)
    {
        byte [] buffer;
        string message;
        StringComparer stringComparer = StringComparer.OrdinalIgnoreCase;
        Thread readThread = new Thread(Read);

        // Create a new SerialPort object with default settings.
        _serialPort = new SerialPort();
        buffer      = new byte[5];
        rbuffer     = new byte[5];

        // Allow the user to set the appropriate properties.
        _serialPort.PortName = SetPortName(_serialPort.PortName);

        _serialPort.BaudRate  = 9600;
        _serialPort.Parity    = Parity.None;
        _serialPort.DataBits  = 8;
        _serialPort.StopBits  = StopBits.One;
        _serialPort.Handshake = Handshake.None;

        // Set the read/write timeouts
        _serialPort.ReadTimeout  = 500;
        _serialPort.WriteTimeout = 500;

        _serialPort.Open();
        _continue = true;
        readThread.Start();

        Console.WriteLine("Type V    to read GPUX Version.");
        Console.WriteLine("Type MIN  to set all ports to 1.0 mSec.");
        Console.WriteLine("Type MID  to set all ports to 1.5 mSec.");
        Console.WriteLine("Type MAX  to set all ports to 2.0 mSec.");
        Console.WriteLine("Type RAMP to ramp up and down ports A and D.");
        Console.WriteLine("Type QUIT to exit.");

        while (_continue)
        {
            message = Console.ReadLine();
            message.Trim();

            if (stringComparer.Equals("quit", message))
            {
                _continue = false;
            }
            else if (stringComparer.Equals("min", message))
            {
                buffer[0] = (byte)':';
                buffer[1] = 128;
                buffer[2] = 128;
                buffer[3] = 128;
                buffer[4] = 128;
                _serialPort.Write(buffer, 0, 5);
```