
RC Radio Interface

December 2006 of Servo Magazine

Pick up an issue at
<http://www.servomagazine.com>



Recently I built a couple of projects where I interfaced a PlayStation controller to a microcontroller. While this works fine in many situations there are times when you want to use an RC radio for its extended range or reliability. In this article we are going to create a very accurate and reliable interface to any RC radio with up to 6 channels.

To perform this interface we will create a Coprocessor using a microcontroller. Once complete we can then connect other devices or even a PC to this interface.

The RC Radio

An RC receiver has multiple connectors that allow you to connect to servos or speed controllers. Each of these connectors has three leads. Two of the leads provide power and the third is a signal lead. The signal lead provides a positive pulse once every 20 milliseconds. It is the width of this pulse that each servo or speed controller uses to determine its position or speed. On average the pulse will range from 1000 to 2000 microseconds. The neutral or center position is at, or close to 1500 microseconds.

As you change the positions of the joysticks or knobs on the transmitter the pulse width will change in proportion to the amount of movement.

When it is all said and done it is these pulse widths that we are going to measure with our interface.

The Objective

When I started the interface project I came up with a list of objectives.

1. Must support 1 to 6 channels
2. Must cover the pulse range from 1000us to 2000us at a resolution of 10us.
3. The interface must support the PC and microcontrollers.

Since the pulse repeats one every 20 milliseconds you will never be able to measure the pulses any faster than that. If we take the time to measure each pulse independently one after another it would take as long as 120 milliseconds to measure 6 pulses. This is much too slow. What we have to do is measure all the pulses at once. We need a real fast chip to do this. This is where the DiosPro comes in. In order to obtain the resolution we need for 6 channels we need to use assembly language. The DiosPRO chip supports inline assembly language with the KRAssembler.

KRAssembly resembles basic in many ways. For instance here is a simple loop to toggle an I O port.

```
Loop:  
  toggle 5  
  goto Loop
```

This short example is exactly the same in the Dios Language as it is in KRAssembly. The difference is that in KRAssembly it runs at up to 10 million instructions per second. This is the kind of speed you need for a project like this.

The DiosPro chip also has a built-in UART so we can add a serial interface that just about any chip, device or PC can talk to.

Take a look at Figure 2. Here I have piggybacked a small 4-channel receiver and the whole thing weighs less than 20g. While our final goal is to create the smallest payload possible I recommend starting your interface with a larger board to allow prototyping and perfecting your program.

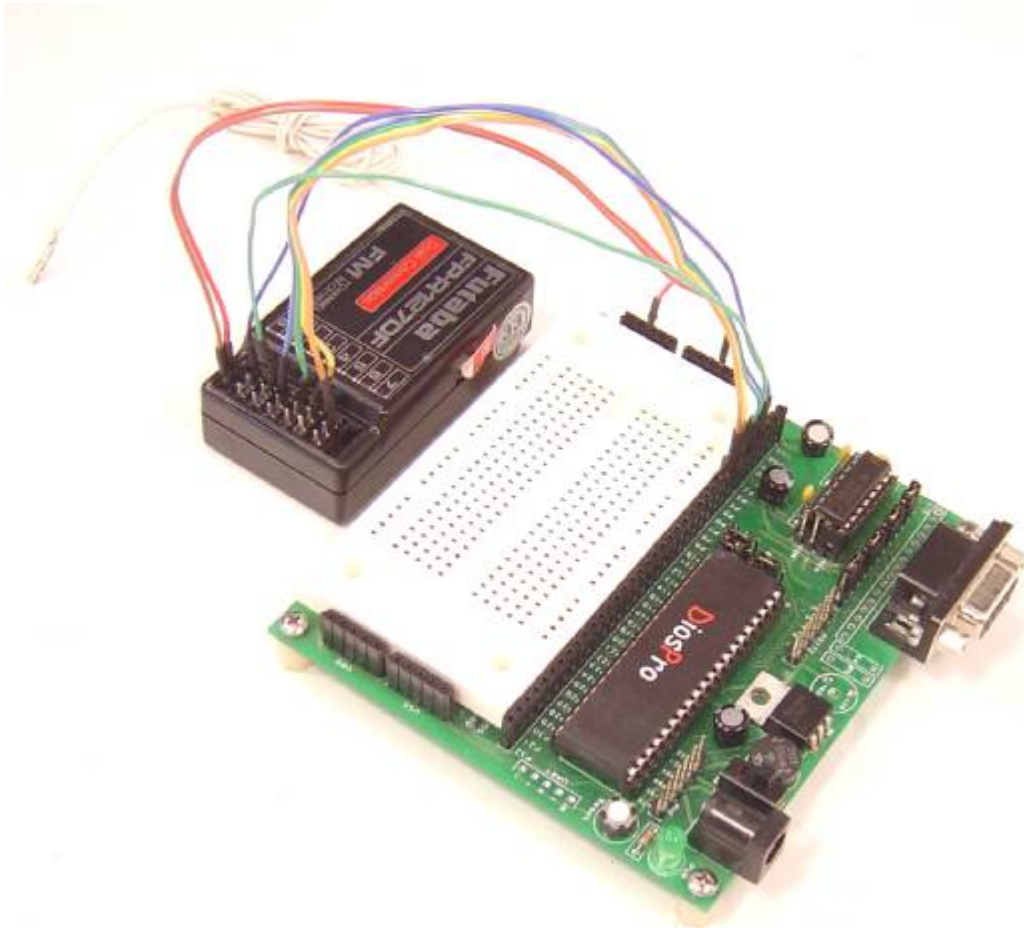


Figure 3

The Dios Workboard deluxe shown in Figure 3 is a perfect way to prototype this project. It contains a built-in regulator and RS232 interface. The built-in interface can be used to program the DiosPro as well as to provide an interface to the PC. If you have a sizable junk box you may want to build your own DiosWorkboard. For that I have made the PCB available as well and it will be listed at the end of the article.

The Firmware

The program that we program into a microcontroller is often called firmware. The firmware for this project is called RCRadio.txt. It is available along with the other source and executables from the Kronos Robotics website.

The RCRadio.txt program gives a range of 10us to 2550us in 10us units. It will return a reading of 150 for a radio pulse of 1500us. The actual accuracy is within the 20us we specified earlier with an accuracy of 10us being more typical.

Due to space considerations the code won't be presented here but, there are plenty of comments if you want to make modifications to the code.

We use a couple of high level Dios commands to set things up then jump into the KRAssembler with the startasm command.

The bulk of the program takes place within a very tight and controlled loop. During each iteration we test each of the 6 ports and if it is high we increment a counter. Once a channel goes high we set a corresponding bit in the flag variable called f0.

During the loop we take a peek at the UART to see if any data has been loaded. If it has it is quickly removed and loaded into the tcmd variable which we will use later. This small peek is carefully timed so that the same amount of time is spent whether we receive data on the UART or not.

Once the UART is checked the program then checks the flag variable f0 to see if all the pulses on the indicated channels have been measured. If all channels have been measured we check to see if the tcmd variable contains a value of 128. If one was made we jump to the appropriate routine to transmit the indicated measurements. We then clean up and wait for all the ports to go low state before we reenter the main loop and start measuring again.

Note that even if we are only measuring 1 port we still cycle through all 6. This keeps the numbers and timing consistent.

The baud rate is set to 115200 baud 8N1. A slower baud rate can be used if you plan on connecting a microcontroller that does not support speeds that high.

To load the firmware into the DiosPro chip you have two choices. You can use the free Dios compiler software available on the Kronos Robotics website or you can use the ZeusPRo software to program the precompiled ZPU file. You will need an RS232 driver or a DiosWorkboard as well.

Connecting the RC Receiver

The connections from the Dios chip to the RC receiver will make or break your project. The following is the most reliable method I have found in doing so.

You will need several 3-pin headers, one for each channel. These are created by snapping (cutting) off the appropriate number of pins from a 36-pin header as shown in Figure 4.

Connect one of the end pins to a length of wire. Add a bit of 1/16" heat shrink as reinforcement. At the other end of the wire attach a single pin pulled from a male header then add some heat shrink to that as shown in Figure 5. You will need one of these for each channel on the radio you are going to connect to the DiosPro.

If your receiver has a separate battery connector you will need to attach two leads to a header as shown in Figure 6. Note that some receivers use BEC only and you must connect the power to one of the servo connectors. Refer to your receiver's user manual.

Once your connectors have been created it is a simple matter to connect the appropriate connector to the Dios or Dios Carrier. You will need to refer to your receiver's manual to determine the appropriate connector. You can also look at one of the servos that came with your radio. The red wire is +, the black wire is -, and the white wire is the signal lead for the servo. On this Futaba radio there is a small key slot for each connector. The key is located next to the signal lead.



Figure 4



Figure 5

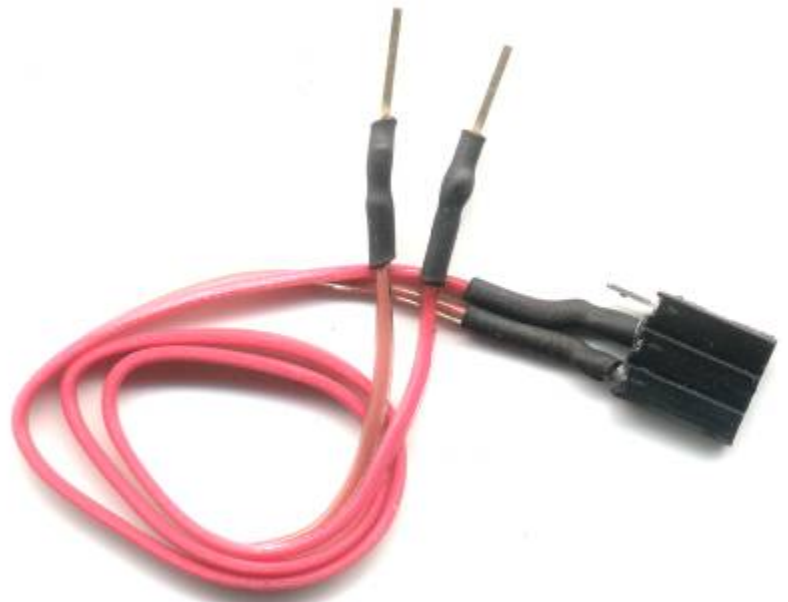


Figure 6

Testing with a Microcontroller

To test the interface with a microcontroller I will use another Dios and a program called ReadRCRadioTest.txt shown in Program 1. The program simply opens a port, sends the request command, reads the 6 channels and displays the results. The output is shown in Figure 7.

Notice the tight loop. This is where you would actually process the received data and use it to control your world. You might be asking, "Why not just place any processing in the same chip that is measuring the pulses in the first place?" The answer is quite simple. The DiosPro runs at 40Mhz and at 10Mips, and we need almost 100% of this to accurately measure the 6 pulses from the receiver. By offloading the processing of the receiver data we not only get accurate readings we also divide and conquer. Once the interface chip has been debugged we no longer have to worry about any of the complexities of this RC radio interface and can concentrate solely on our application. It's the hardware equivalent to modular programming.

```
DiosPro
func main()

  hsersetup baud,HBAUD115200,txon,start,inwait,25
  pause 50
  hserout 6 'Set the number of channels
  pause 50

Loop:
  getreadings()
  pause 100
  goto Loop

endfunc

func getreadings()
```

```
dim a,b,c,d,e,f
hsersetup clear
hserout 128

hserin timeout,a,b,c,d,e,f
print a," ",b," ",c," ",d," ",e," ",f

exit

timeout:
  print "Timeout"
endfunc
```

Program 1

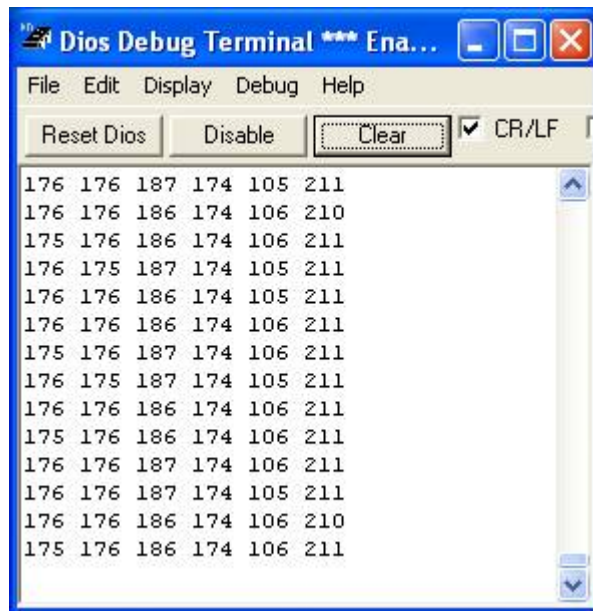


Figure 7

Notice that just after we have opened the port we send the command

hserout 6

This tells the radio interface that we want all 6 channels. If you only want 2 or 4 channels you will need to change this command as well as the hserin command.

One thing to keep in mind is that if you request 6 channels you must have 6 channels connected to the radio. If a requested channel has no data then no data will be transmitted until signal returns. For this reason the host program should test for a timeout. If no data is received after 20ms then the transmitter is off or a connection is broken.

For the best reliability you should tie the unused channels to Vss.

Testing with PC

In order to connect a PC or Pocket PC to the interface you will need an RS232 driver chip. The Dios Workboards all have built-in RS232 drivers that can be used for programming or as an interface. Refer to the Dios Workboard manuals for setup instructions.

The program called RCRadioTest.txt shown in Program 2 is a Zeus program. This program will work with the Free version of Zeus Lite. Notice that this program, while not identical, is very similar to the Dios program presented earlier.

```

func main()
  dim x as integer

  x=ComOpen(1,baud=115200,port=1)
  print "Open Status = ";x
  ComDTR(1,0)
  pause(50)
  ComSettings(1,Timeout = 1,Priority = 10)
  ' ComBGSuspend(1,0)

  pause(50)
  ' Tell RCRadio Chip we want 6 channels
  ComOutput(1,chr(6))
  pause(50)

Loop:
  getreadings()
  pause(10)
  goto Loop

endfunc

'-----
' Get 6 readings
'-----

```

```

func getreadings()

  dim a,b,c,d,e,f,cmdidx
  ComPurge(1)
  doevents()
  ComOutput(1,chr(128))

  cmdidx = -1
Loop:
  ComGetPacket(1,Loop,Timeout,50,cmdidx,a,b,c,d,e,f)
  print a," ",b," ",c," ",d," ",e," ",f

  exit()

TimeOut:
  print "TimeOut"
  exit()
endfunc

```

Program 2

Going further

I wanted to take the PC test program a step further. In the program RCRadioTestForm.txt, I display the actual joystick positions as shown in Figure 1. This program requires ZeusPro but I have also included a compiled exe of the program as well.

In the August Issue of Servo Magazine there was an article called FaceWalker. FaceWalker used a DiosPro to interface to a wireless Playstation controller. It would be a very simple task to replace that interface with the one presented here. Simply replace the UART interface to one that tests port 14.

Sources

I have provided the part numbers for the carrier PCB's as well as the EZ232 PCB. This way you can build your own kits. For a list of the parts needed to build your own kits just download the manuals for the appropriate board. These can be found on the website.

KRMicros

<http://www.krmicros.com/>

ZeusLite

<http://www.krmicros.com/Development/Pocket%20Zeus%20Lite/PocketZeusLite.htm>

Kronos Robotics

<http://www.kronosrobotics.com>

Dios Compiler

<http://www.kronosrobotics.com/downloads/DiosSetup.exe>

Project Downloads

<http://www.kronosrobotics.com/Projects/RCRadioInterface.shtml>

Other Parts

These parts can be found on the Kronos Robotics website.

DiosPro28 Chip	#16429
DiosPro40 Chip	#16428
DiosCarrier1	#16170
Carrier1 PCB	#16151
Dios Workboard Basic	#16453
Dios Workboard Deluxe	#16452
36-Pin Female Header	#16291
40-Pin Male Header	#16290
EZRS232 Driver	#16167
EZRS232 PCB	#16149

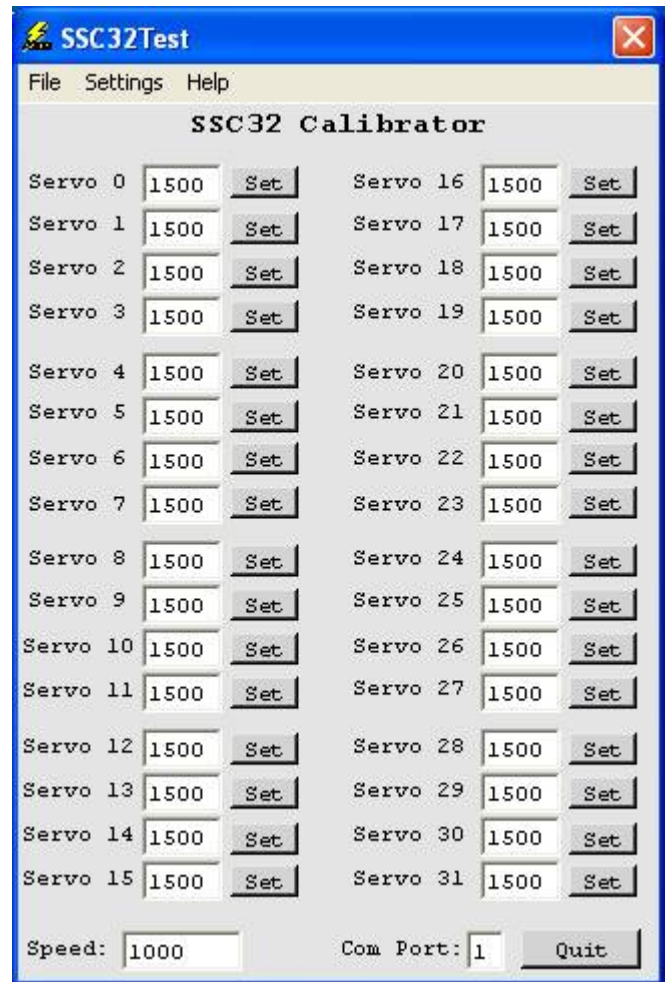


Figure 21

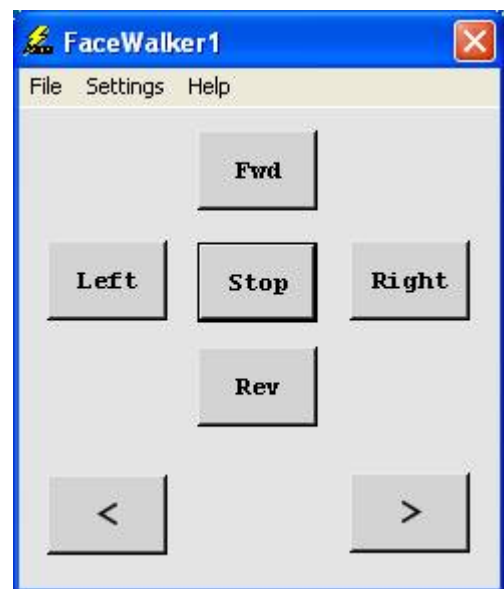


Figure 22