

In Part 2 we are going to expand upon the slave IOproc that we created in Part 1. There are several things we have to do to create the ability to place multiple chips on the same transmit and receive lines.

The first requirement is that we must turn off the transmit port when not in use. We use the **cptxoff** command to do this. This allows us to connect all the transmit leads of the chips together. If we did not do this there would be contention on the port and all that would get transmitted is garbage.

The transmitter will come back on line automatically each time we use the **cptxmt** command. The problem is that after we transmit our data we need to issue the **cptxoff** command again. While this works in theory it does not work in reality because the transmitter may still be transmitting bits when we issue the **cptxoff** command. We could use a pause command to place a small delay but what we really need is a way to tell when the transmit buffer is empty. Luckily one exists. The following subroutine is called each time we want to place the transmitter offline.

```
offline:
    if TXSTA & 2 = 2 then
        cptxoff
        goto loop
    endif
    goto offline
```

The **if TXSTA & 2 = 2 then** statement tests to see if bit 1 on the internal register TXSTA is set. If it is we know the transmit buffer is empty.

This solves the problem of placing the transmitter offline. You can now connect all the transmit leads together and all the receive leads together. What you have done is created a bus that you can place as many chips as you want.

On the hardware side of things you must do one more thing. You must pull the transmit side of the bus high with a 10K resistor. This keeps the Port in the idle state even with all the transmitters offline. When a chip goes online the resistor is weak enough to be overridden by the chip that goes online.

Now all we need is a way to identify each chip so that only the chip we want will respond to commands. We do this with an additional byte of data in our packet. The first byte in the packet will be the target chip's ID.

After the packet has been received we will compare the target ID with the ID of the chip. If they do not match we will simply ignore the packet and continue back to the main loop.

Step A: Program and Test the Slave

The following program adds all the items we just discussed to the IO proc demo in Part 1.

```

'Chip To Chip 2A
Nemesis
'MultiChip IO Proc Demo 1
  dim atodflags,temp,owbyte,x,owport
  dim ID
  dim cmdidx,target,cmd,data1
  clearall

  ID=11 'This chips ID

'Commands
'CMD ID,0,port  Set port as input
'CMD ID,1,port  set port as output

'CMD ID,2,port  Read and return port
'CMD ID,3,0     Read and return all ports

'CMD ID,4,port  Set port low
'CMD ID,5,port  Set port high
'CMD ID,6,data  Set All Ports

'CMD ID,7,port  Set port as Analog
'CMD ID,8,0     Set allports as Digital
'CMD ID,9,port  Read and return Analog value (8 bit)
'cmd ID,10,port Read and return Analog value (16 bit)

cptxoff

'-----
'  Main Loop
'-----
loop:
  if RCSTA & 4 <> 0 then
    temp = RCREG 'Clear Frame Error
    cmdidx = 0
    cptxoff
  endif
  getpacket loop,cmdidx,target,cmd,data1
  if target <> ID then
    goto loop
  endif
  branch cmd,doinput,douput,doread,doreadall,dosetlow,dosethigh,
  dowriteall,dosetatod,dosetdig,readatod,readatod16
  goto loop

'-----
'  Set Port State (input)
'-----
doinput:
  gosub checkport
  input data1
  goto loop

'-----
'  Set Port State (output)
'-----
douput:

```

```

    gosub checkport
    output data1
    goto loop

'-----
' Read port
'-----
doread:
    gosub checkport
    onportgoto data1,-,readhigh
    cplxmt 0
    goto offline
readhigh:
    cplxmt 1
    goto offline

'-----
' Read all port
'-----
doreadall:
    portget data1
    cplxmt data1
    goto offline

'-----
' Set Port Low
'-----
dosetlow:
    gosub checkport
    low data1
    goto loop

'-----
' Set Port High
'-----
dosethigh:
    gosub checkport
    high data1
    goto loop

'-----
' Set All Ports
'-----
dowriteall:
    portset data1
    goto loop

'-----
' Set Port to AtoD mode
'-----
dosetatod:
    lookup temp,data1,1,2,4,16,32,64,0,8,0,0,0
    atodflags = atodflags ^ temp
    atodinit atodflags,0
    goto loop

'-----
' Set Port to digital
'-----
dosetdig:

```

```

atodinit 0
goto loop

'-----
'Send AtoD 8-bit
'-----
readatod:
  ATODMODE=0
  atod data1,temp
  cptxmt temp
  goto offline

'-----
'Send AtoD 16-bit
'-----
readatod16:
  ATODMODE=128
  atod data1,temp
  cptxmt temp
  temp = ADRESL
  cptxmt temp
  goto offline

'-----
'Utilities
'-----
checkport:
  if data1 > 14 then
    returnto loop
  else
    return
  endif

offline:
  if TXSTA & 2 = 2 then
    cptxoff
    goto loop
  endif
  goto offline

```

To test the code we programmed 3 Nemesis chip. Notice the ID variable. We set the ID variable to a different value in each chip. In the case of our test program they were set to 10,11, and 12.



Program and test each chip individually. To test the chip you will have to use its ID as shown above. In this case we are addressing the chip with the ID of 12 and requesting the port 0 status. When you send this packet the transmitter of the target chip will go online transmit the port state then go back offline again.

Now it's time to talk to three chips. We connected an RS232 driver to three chips by connecting their transmit ports (Nemesis Port 4) and their receive ports (Nemesis Port 12). You will have to connect the ATN leads on all the chips as well as the Vss and Vdd pins as well.



Again we will use the debug terminal to test the target chips. Just change the ID (First Byte) for each chip you want to talk to.

Here is a list of the commands for the new slave procs.

In this implementation we set up a 3 byte packet. The first byte is the ID, the second is the command and the third is used as a data byte. For demonstration purposes we are only accessing the first 8 ports but you could support all the ports if you wish.

CMD ID,0,portSet port as input
CMD ID,1,portset port as output
CMD ID,2,portRead and return port
CMD ID,3,0 Read and return all ports
CMD ID,4,portSet port low
CMD ID,5,portSet port high
CMD ID,6,dataSet All Ports
CMD ID,7,portSet port as Analog
CMD ID,8,0 Set allports as Digital
CMD ID,9,portRead and return Analog value (8 bit)
CMD ID,10,portRead and return Analog value (10 bit) Sends 2 bytes

You can set ports 0-7 as input, output or AtoD input. Note that the Nemesis uses port 4 for communications so access to that port is not possible.

Step B: Test the Master

First you need to connect the Master chip to the Slave chips. We are going to use ports 0 and 1 as a software serial interface on the Master to connect to the Slave.

Connect:

Master Port 0 to all the Slave Ports pin 12

Master Port 1 to all the Slave Ports pin 4

Connect a 10K resistor between the Slave Ports pin 4 and Vdd

Master Vss to Slave Vss

Note that if you are using chips other than the Nemesis you will need to connect Port 0 on the Master to program RX on the Slave and Port 1 on the Master to Program TX on the Slave.

You wont need your PC connected to the Slave since you have already programmed and tested it. Connect PC to the Master so that you can program it.

Enter the following code into the Master.

```
`Chip To Chip 2B
Nemesis

  dim Chip10,Chip11,Chip12
  output 0
  low 0
  pause 25 'Need this to sync Slave
  Setbaud SBAUD9600

loop:
  serout 0,10,2,0
  serin -,1,Chip10

  serout 0,11,2,0
  serin -,1,Chip11

  serout 0,12,2,0
  serin -,1,Chip12

  print Chip10," ",Chip11," ",Chip12
  goto loop
```

The program retrieves the state of port 0 from each chip then displays the data.

Final Thoughts

The key issue with this protocol is that all chips on the bus must use the same packet length. If you design a chip that uses 5 packets you will have to increase the size of all the chips to 5 and just ignore the extra data.

You can go one further by creating a variable packet protocol and in a future addition to this series we will do just that.

Parts

- | | |
|-------------------------|---|
| Athena Microcontroller | Kronos Robotics #16276 |
| Nemesis Microcontroller | Kronos Robotics #16406 |
| Athena WorkBoard Deluxe | Kronos Robotics #16457 |
| Athena Compiler | http://www.kronosrobotics.com/downloads/AthenaSetup.exe |