

# RS485 Half Duplex

Wednesday, May 12 2004 @ 01:00 AM EDT



RS485 Half Duplex with Dios as Master and Athena485 as slave.

## What is RS485?

The RS485 standard defines the electrical considerations for the line not the protocol. Generally RS485 is a half duplex system. You can however implement full duplex systems as well with multiple drivers. With RS485 you use two physical lines for each signal. In this article we will concentrate on a half duplex system. The actual half duplex signal will use 2 physical lines. The two lines for each are referred to as balanced differential and are complementary of each other. For instance to transmit a high the A lead will be high and the B lead will go low. To transmit a low the A lead will be low and the B lead will be high. This system makes the signals much more immune to interference. The RS485 system also calls for a common as well. This can be transmitted via the cable or you can tie each node (slaves and master) to GND.

One of the other benefits of RS485 is that you can connect multiple devices to a single bus. How you do this depends upon the driver and the protocol used. The specifications says you can have upto 32 nodes on the bus. This is based on a 12k impedance of each node. However there are drivers that have 1/2, 1/4, and 1/8 loading so more nodes can be connected.

To interface our TTL level microcontroller we use a RS485 driver. There are single channel and multiple channel drivers available from various manufactures. I have found that the single channel drivers seem to be interchangeable. They all have the same pin out. In order to go full duplex with a single channel driver you will need two.

In this article we are going to do a half duplex connection. Half duplex has a couple advantages.

- | The hardware is less complicated.
- | It makes it easier to have multiple masters on the Bus.
- | You only need 2 wires for the bus. This means you can use normal telephone wire for the signal lines

The driver chips have control leads to set the direction of data flow. So in order to communicate with these driver chips in a half duplex operation we will need 3 leads.

- | Tx
- | Rx
- | Control (or Direction)

The control lead will need to be brought high or low depending on the direction of the communications. Its even more complex then that. When the master is transmitting it is in transmit mode the the slaves will need to be in receive mode. If ether is in the wrong mode nothing will happen and no chip will receive data.

The Athena485 (and Perseus) have a special command called apinit. This command allow you to set up a low level protocol that will control the transmit, receive and control leads on the slave chip.

## Low Level Protocol

If all we needed to do was to have one chip send to another chip we could hardwire the direction control on the driver chips and just transmit away. No need for a protocol. However we want to send commands and get responses from multiple chips on the bus. Lets look at how the AP (Address Protocol) protocol works in half duplex mode.

Master:

- | The master places the internal UART into 9 bit mode.
- | The 9th bit is set to 1.
- | The driver is placed into transmit mode by the master (note that all the slaves are in off line receive mode by default)
- | A byte 0-255 is transmitted.
- | Once all bytes in the packet (address + data bytes) have been sent the master goes into receive mode by pulling the control lead low.

On the slave end the byte is received by the UART on all slaves on the bus. The UART on all the slaves detects the 1 in the 9th bit and causes an internal IRQ to fire. Each slave internally checks the received 8bit byte to check to see if it matches its address. (set with the apinit command) If the address does not match the slave chip remains off line and the ignores all data until another address byte comes in. If the address does match the slave chip is placed into normal online mode. In this mode the chip will receive data normally until an address byte is received. At this point the slave uses one of the high level protocols to accept commands. In this example we will use the getpacket protocol. If the command sent is a request for data the slave will force the driver into transmit mode by pulling the control lead high. The requested data is then transmitted. Once all the data has been transmitted the slave goes back off line and forces the driver chip into receive mode.

## High Level Protocol

Lets take a closer look at the getpacket protocol used on the slave end (Athena485). In this protocol we tell the command how many data pieces we need to have populated. we keep track of them with a command index variable. Once all data pieces have been received the command falls through so we can process the data. If all the pieces have not been received we have provided a label to jump to.

```
dim cmdidx,data1,data2
loop:
  getpacket loop,cmdidx,data1,data2
  ..... handle data here
```

In the above example the program will continue to loop from the getpacket command until two bytes have been received. These two bytes will be placed into the data1 and data2 variables. In most cases you will use a branch command to jump to an appropriate piece of code to handle the data received.

## Generic Slave Program

Lets look at a generic slave program that lets the master control its IO ports. In order to test the program we will comment out the apinit command so that the low level protocol is turned off. This will allow us to use the debug form to test the program.

Athena485

'Athena485 RS485 IO slave

'CMD 0,port Set port as input

'CMD 1,port set port as output

'CMD 2,port Read and return port

'CMD 3,0 Read and return all ports

'CMD 4,port Set port low

'CMD 5,port Set port high

'CMD 6,data Set All Ports

'CMD 7 miniad port 0

'CMD 8 miniad port 1

dim atodflags,temp,owbyte,x,owport

dim cmdidx,data1,data2,data3,data4,data5,data6,data7,data8,data9,data10,data11,data12

clearall

'Will add back when in production

'apinit 100,RS485HALF 'Half Duplex Via RS485

offline:

apoffline

loop:

getpacket loop,cmdidx,data1,data2

branch data1,doinput,dooutput,doread,doreadall,dosetlow,dosethigh,dowriteall,dominiad0,\_,  
dominiad1

goto loop

'-----  
' Set Port State (input/output)  
'-----

doinput:

gosub checkport

input data2

goto offline

dooutput:

gosub checkport

output data2

goto offline

'-----  
' Read ports  
'-----

doread:

gosub checkport

```

onportgoto data2,-,readhigh
cptxmt 0
goto offline
readhigh:
cptxmt 1
goto offline

doreadall:
portget data2
cptxmt data2
goto offline

```

```

'-----
' Set Ports (High or Low)
'-----

```

```

dosetlow:
gosub checkport
low data2
goto offline

```

```

dosethigh:
gosub checkport
high data2
goto offline

```

```

dowriteall:
portset data2
goto offline

```

```

'-----
'miniad
'-----

```

```

dominiad0:
miniad 0,temp
cptxmt temp
goto offline

```

```

dominiad1:
miniad 1,temp
cptxmt temp
goto offline

```

```

'-----
'Utilities
'-----

```

```

checkport:
if data2 > 10 then
returnto offline
else
return
endif

```

Notice the comments at the beginning of the program. It shows the 9 commands the program will support. (0-8)

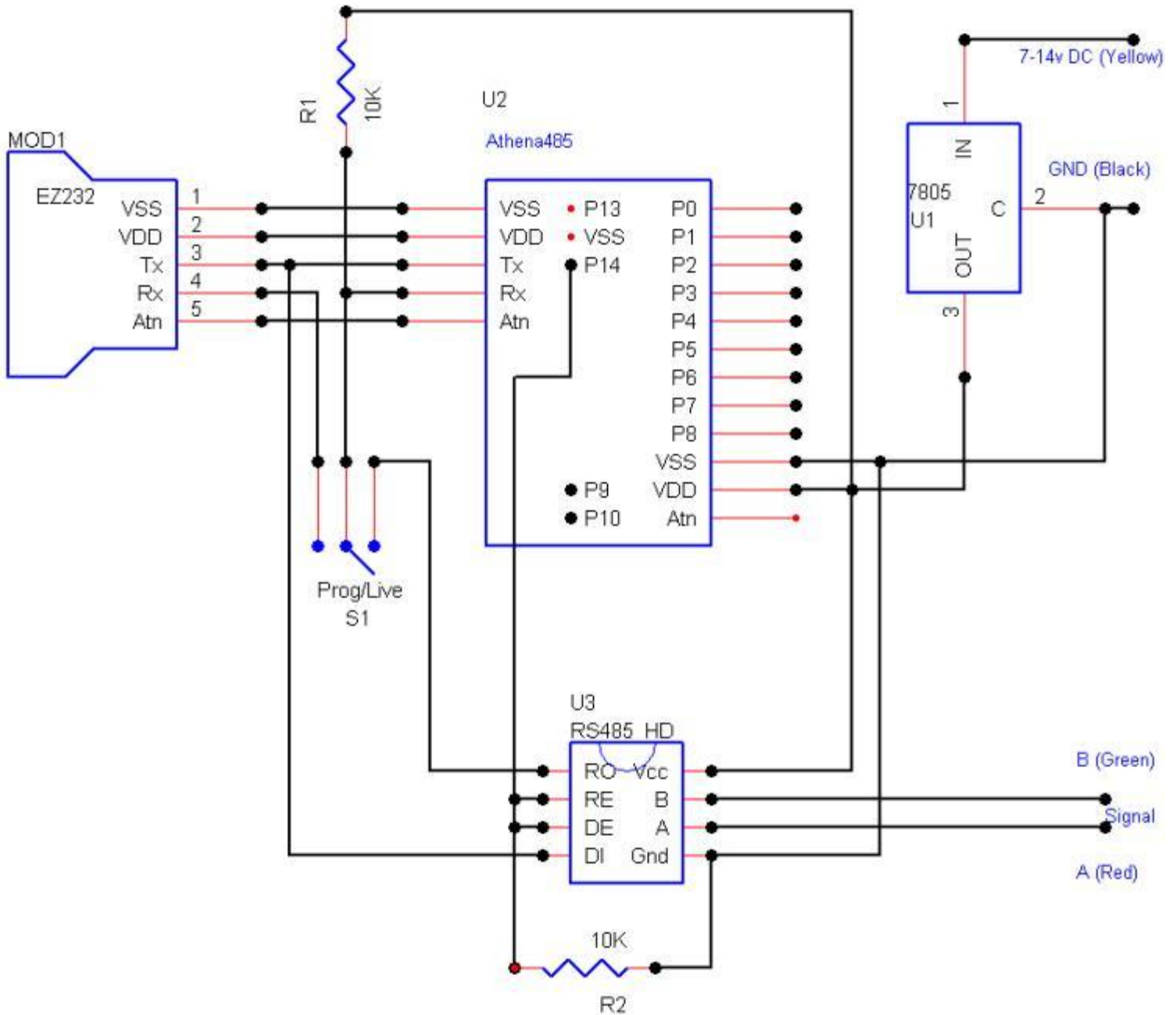


The first byte is always the command and the second byte extra data. In this case we placed a 2 and 3 into the data fields shown on the debug form example. Note that you have to have the Difficulty mode set to Advanced. Hit the send button and the 2 bytes will be transmitted.

Notice that the Athena485 sends back a 0. If you check the code you will see that command 2 tells the Athena485 to transmit the status of a input port. In this case port 3.

Because we use the UART to program the Athena485 it must be remove to do actual live RS485 testing. This is why it is important to test your code as much as possible before live testing.

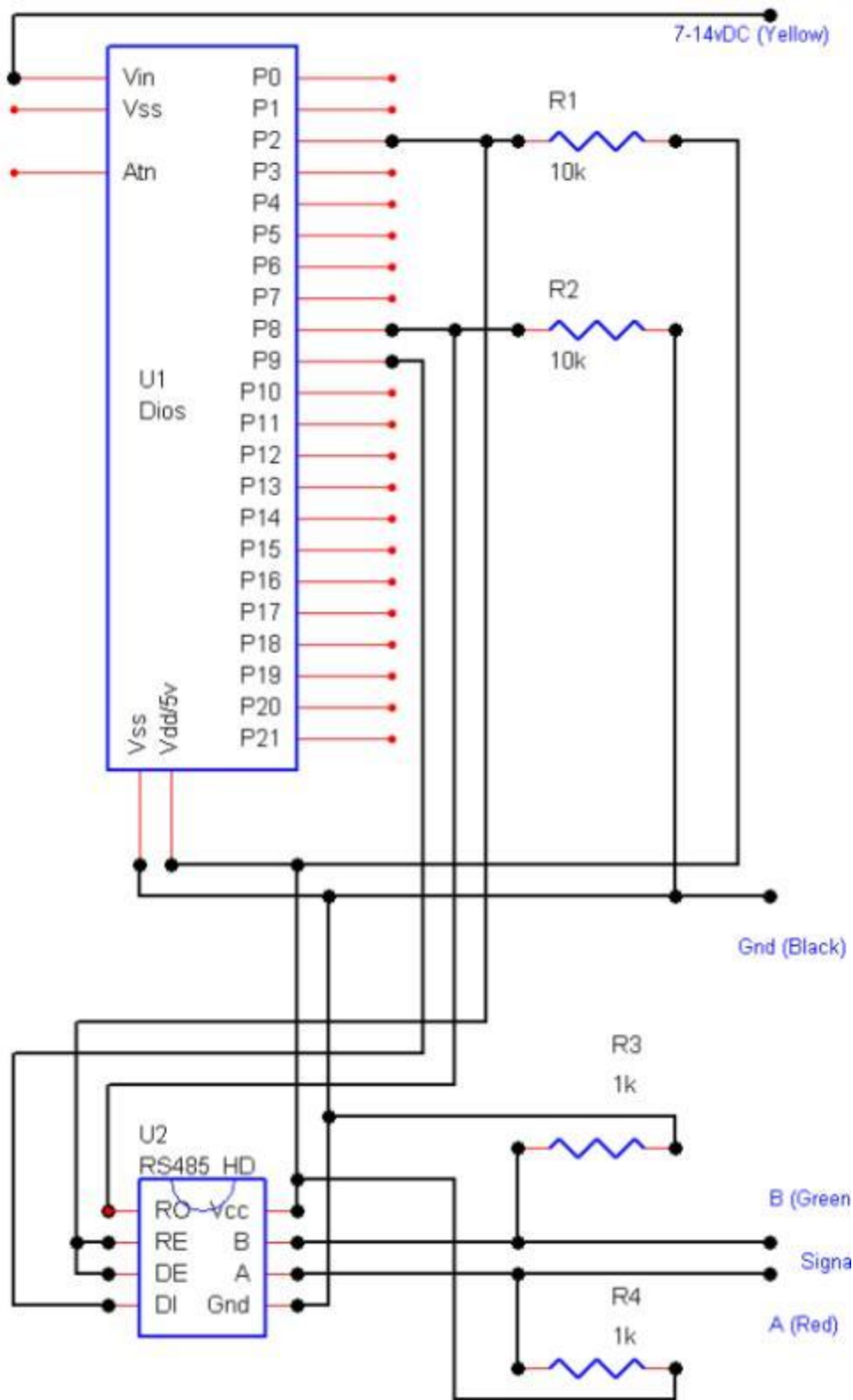
## **Slave Hookup**



We have added S1 to allow us to easily switch between program and live mode. This makes testing the slave with the master a snap. To make a change to the program just place the switch in program mode. Switch back to test with the master. The program above has been fully tested so you should not have to do much debugging. Just make sure you remove the comment from the apinit command and set the address to one of your choice.

A quick comment on power before moving on to the master. You can provide power to the slave any way you wish. It is common to use a 2 more wires in the telephone cable to provide power. If you plan on several devices on the bus you should use a 7.5 volt or better power source and regulate it at each slave. For high current loads its recommended you supply the power externally.

### Master Hookup



We are going to use the master to provide the unregulated power to the slaves. These will be placed on a 2nd pair of wires. Notice the colors indicated in the schematics. If your cable is not colored as indicated make sure you create your own color code system.

We are using a Dinos as a master and it has the advantage of not using the UART to program the chip. This makes programming and testing very easy.

Notice resistors R3 and R4 these are bias resistors and are used to hold the signal leads in a know state when all drivers are in receive mode. This particular state happens for short intervals when the master switches from transmit mode to receive mode. If these resistors were not used all nodes would received garbled data during this interval. We only need these on the master and the schematic shows 1k resistors. This should work for most networks. For networks with high noise and many nodes you may want to change them to 5K resistors.

## Master Program

We have created a couple functions for returning various pieces of information from the remote device. In this example we will read the 4bit AtoD port.

Make sure the apinit command has been uncommented and that the address in the master program matches that on your slave.

```
'Dios Master to control remote Athena485 satalite via RS485 half
' Duplex and ap protocol.
func main()

    dim dat

    hsersetup baud,HBAUD9600,start,txon,clear
    TXSTA.bit(6)=1 'Set up for 9 bit mode

loopy:

    dat = apreadminiad0(100)
    print dat
    goto loopy

endfunc

'-----
'Places the remote slave port 0 in miniad mode
' and returns the reading on this pin (0-15)
'-----
func apreadminiad0(addr)
    dim dat
    apsend(addr,7,0)
    dat=apgetbyte()
    exit dat
endfunc

'-----
'Places the remote slave port 1 in miniad mode
' and returns the reading on this pin (0-15)
'-----
func apreadminiad1(addr)
    dim dat
    apsend(addr,8,0)
    dat=apgetbyte()
    exit dat
endfunc

'-----
'Returns the port state of a given IO port 0-10
'-----
func apgetport(addr,tport)
    dim dat
```

```
    apsend(addr,2,tport)
    dat=apgetbyte()
    exit dat
endfunc
```

```
'=====
'Low Level Routines
'=====
```

```
'-----
'Get byte from remote device
'-----
```

```
func apgetbyte()
    dim dat,ct
    ct = 0
    global apgetbytetimeout
    apgetbytetimeout = 0
```

```
Rloop:
    ct = ct + 1
    if ct > 2000 then
        apgetbytetimeout = 1
        exit 0
    endif
    hserin Rloop,dat
    exit dat
endfunc
```

```
'-----
'wake up chip and send commands
'-----
```

```
func apsend(addr)
    dim args(10)
    dim argct,x
    argct = OPP8
    if argct<1 then exit 0

    output 2
    high 2
```

```
'-----
    TXSTA.bit(0)=1
    hserout addr
    while TXSTA.bit(1)=0
        wend
    TXSTA.bit(0)=0

    if argct < 2 then exit 0
```

```
'-----
'Now send bytes
```

```
    for x = 2 to argct
        hserout args(x - 2)
    next

    while TXSTA.bit(1)=0
        wend
    output 2
    low 2

    pause 1
```

endfunc

The real heart of the master program is the `apsend` command. This command places the UART transmitter into 9 bit mode and sets the 9th bit to 1 while transmitting the slave address. Once sent it then goes into normal transmit mode and transmits the remaining bytes in the packet. Once transmitted it goes into receive mode until then next command packet is sent.



Here is a picture of a slave located in my barn. It is about 400' away from the master.



I added a surplus motion sensor that will return a low state when activated.

### Notes

The RS485 calls for 120 ohm termination resistors located across the A and B leads at the farthest points in the network. I have found at 9600 baud these are not needed. If you want to add them you will also need to change the values of the bias resistors.

The [PCB8](#) works very well for remote slaves. You can use the small proto area for the RS485 Driver.

---

---

### Related Products

- Athena485..... <http://kronosrobotics.com/xcart/customer/product.php?productid=16381>
- EZRS232..... <http://kronosrobotics.com/xcart/customer/product.php?productid=16167>
- PCB8..... <http://kronosrobotics.com/xcart/customer/product.php?productid=16150>
- Dios Carrier 3.... <http://kronosrobotics.com/xcart/customer/product.php?productid=16386>
- Dios 28 pin chip.. <http://kronosrobotics.com/xcart/customer/product.php?productid=16169>
- RS485 Driver chip. <http://kronosrobotics.com/xcart/customer/product.php?productid=16388>

[0 comments](#)

---