QUADS Board

Tutorial
A QUICC-Start

1/7/94

Introduction

This version of the tutorial is valid for QUADS boards with revision 0.3 User Interface Software (The revision number is shown at the top of the “360sw” screen).

This guide is intended to make your introduction to the MC68360 a little easier and faster. By now you should have at least glanced at a 68360 User's Manual and be aware that the 68360 has three main sections: a CPU32+ processor; a number of general purpose peripherals such as timers, an interrupt controller and 2 independent DMA channels; and a communications section with 4 serial communication controllers (SCCs) and a Time Slot Assigner. You should also have gained access to a QUADS board. Along with its many other uses, the QUADS board is the fastest way to get acquainted with the part, since it has built-in software that is designed for just this purpose. If you do not have a QUADS board present, you can still learn some things from this tutorial.

You'll notice the QUADS board has 2 QUICC devices on it. One of them is the Master QUICC with its CPU32+ core enabled and the other one is in slave mode (CPU core disabled). The Slave QUICC is used for controlling the ports on the QUADS board such as the RS232 port used in stand alone operation and the parallel port used for communication with the ADI card. The Master QUICC is left completely untouched by the board software and can be used by the user to develop code. In this guide, we will be configuring the Master QUICC to complete a series of exercises.

If you have not used a QUADS board before, don't worry, you will learn the basics from this guide -- just enough

Text Notation

Through this document, you will see various text notations. The following is a list of these conventions:

`Prompt:` **`data`** – The bold type is entered by the user, the non-bold type is text from the machine.

`DOS>` - prefixes a command to be typed at the IBM/PC command line.

`QUICCbug>` - prefixes a command to be typed at the QUICCbug prompt.

<CR> - hit the “enter” or “return” key on your keyboard.

[text] or #text - notes to accompany the operation that is being performed

Initial Setup

You can operate the board with a VT100 compatible terminal (or VT100 terminal emulation program on any computer). This is the easiest way to get started. In this case, the terminal should be plugged into the P5 Terminal port on the board (9600 baud, 2 stop bits, no parity). Then, to enter the debugger, simply reset the board (press the Hard and Soft Reset switches simultaneously - SW1 and SW2).

You can also use a PC or Sun 4 as a host using a separate ADI card. The host software package is used to invoke the board firmware. This software currently exists for both the IBM/PC and SUN4 with the following User Interfaces:

- Command Line Interface
- Graphical User Interface

Use with ADI

IBM-PC version

Follow the instructions in Chapters 1 & 2 of the QUADS Hardware User’s Manual to connect the QUADS board to the host. Then follow the instructions in Chapter 4 of the QUADS Software User’s Manual to install the to start the graphical user interface from an IBM/PC, type "`..\host_gui -s 100 0`" from the "QSTART" directory with an IBM-PC

SUN Version

Follow the instructions in Chapters 1 & 2 of the QUADS Hardware User's Manual to connect the QUADS board to the host. Then follow the instructions in Chapter 4 of the QUADS Software User's Manual to install the software on the PC. Finally, follow the instructions in the README file on how to install the interactive software on the Flash Eprom on the QUADS board.

If the directory "QSTART" did not come on disk with the QUADS Board package, the files can be downloaded from the Motorola BBS - (512) 891-3733.

Remember -- if you use the HOST software, you do not use the red Reset or black Abort buttons on the board, but rather the host software Abort and Reset options, such as ctrl-A and ctrl-Y.

Examples

Debugger Basics

Reset the board. At this point you should get the MC68360 QUICC debugger prompt:

```
QUICC Monitor / Debugger - Version 0.4
   (C) Copyright 1992 by Motorola Inc
Cold start
DRAM SIZE IS 100000 BYTES
QUICCbug>
```

First, lets take a look at a few miscellaneous QUICCbug commands.
Hit <CR> after typing in each command.

```
QUICCbug> rd                          # displays the register set
```
[see the register display]

```
QUICCbug> rm D1
```

Note

This command prints out memory locations starting with address $20000. Refer to the QUADS memory map on page 15 of the M68360QUADS User's Manual. Notice that location $20000 is the starting address for the Master QUICC's internal Memory. This value is pointed to by the Master QUICC's MBAR register. The software programs MBAR to $20000 for proper configuration of the memory map. Now if you want, you can begin to modify the RAM and registers of the 68360. We will just display some of them for now:

```
QUICCbug> md 20000:70          # displays $70 words from some of the 68360 Dual-Port RAM
QUICCbug> md 21000,21070       # displays $70 bytes from some of the 68360 Registers
QUICCbug> md 2160e             # displays the 68360 DSR register for SCC1
```

Now let's try a memory modify command:

```
QUICCbug> mm 2160e
0002160e 7e7e ?  0800 .        # sets the location to $0800 and the period terminates the input
```

Note

This command reads the memory location before you can modify it, and as a word (16 bit) by default. Some bit in registers (like SCCE, event register) are cleared by writing “ones”. In this case, you can use the “MS” command (Memory Set) if you don't want to read the register first. In this example, you should type:

```
QUICCbug> ms 2160e 0800
```

[This changed the contents of location $2160e from $7e7e to $0800]

Other useful commands

```
QUICCbug> he                   # Help - gives a list of the commands
QUICCbug> he mm                # Help on a specific command
```

QUICCbug has a single line assembler. It is an option of the memory modify command.

```
QUICCbug> mm 403000;di
```

[Now you can type “NOP” to insert a no-op instruction, for instance. Once again a “.” takes you out]

```
00403000    00000000    ORI.B    #$0,D0   ? NOP
00403002    00000000    ORI.B    #$0,D0   ? .
```

Note

Other commonly used instructions

- `t` - trace, to step one (or more) instructions at a time
- `gn` - to set a breakpoint at the next instruction's address. Useful for skipping subroutines.

You are now familiar with the basics of the QUICCbug debugger on the QUADS board. For more information on additional commands refer to the CPU32Bug Debug Monitor User's Manual.

Trying the User Interface Software

Reset the board as described above.

```
QUICCbug> 360sw
```

[This starts up the User Interface Software. This command is the same as "`g b0000`"]

You are now in the 68360 menu. This is menu A. Take a quick glance at all the other menus (b,c,d,e,f,g). Hit <CR> after each command. You can use <del> to erase the character if you make a mistake.

```
Enter option: b                   # Driver menu (68360 chip drivers)
Enter option: c                   # LAPB menu (layer 2 protocol)
Enter option: d                   # LAPD menu (layer 2 protocol)
Enter option: e                   # X.25 menu (layer 3 protocol)
Enter option: f                   # BISYNC menu (part of the layer 2 protocol)
Enter option: g                   # File transfer menu (an example layer 7 protocol)
Enter option: a                   # now go back to menu a
```

These menus control the software modules that are included in the EPROMs on the QUADS board. The menus are detailed in the 68360 Software User's Manual and the modules themselves are detailed in the 68360 Software Programmer's Manual. (Some of the modules like LAPB and X.25 may not be implemented in your FLASH EPROMs, but must be downloaded into RAM before their menus can be accessed).

In this exercise we will only deal with menu "a", the 68360 menu. This menu gives you an easy way to modify the 68360 registers directly.

```
Enter option: 25                  # Choose memory display to see what DSR is now.
Enter :... 2160e 10               # To get $10 bytes from memory location $2160e
```

[Notice that location $2160e (the DSR) has been automatically written with 7e7e upon RESET. This value "$7e7e" is of significance only in that the HDLC protocol uses this as a synchronization sequence]

```
Enter option: 25
Enter :...21600 70
```
[Notice the $7e7e sequence is repeated four times. Those four sets of registers are for the 4 SCCs. We will look an SCC register set in detail shortly. Hit <CR> to continue]

Now try someother options. Notice that the selected registers on the 68360 are now displayed in a more readable format. Hit <CR> to page through the screens:

```
Enter option: 15                # Interrupt Controller
Enter option: 17                # Timer registers
Enter option: 20                # SCC registers
Enter option: 11                # System Integration Module (SIM) registers
Enter option: 00                # To exit the User Interface Software menus.
```

Starting a Timer

Reset the board as before

```
QUICCbug> 360sw                 # to start the user interface software
```

To be able to modify any registers, you must first select the “flip read/write” option:

```
Enter option: 3                 # should put a “+” on the screen. Allows register writes.
Enter option: 17                # Look at the Timer registers
Enter field number: 1           # selects the Timer Mode Register (TMR1) for timer 1
```

At this point there are 2 options. You can enter the value to be placed in the register or you can hit ‘?’ to display the bit fields of the registers. Let’s try the first one.

```
Enter value... 0002             # selects timer 1's clock source to be the master clock
```
[Note that the value in TMR1 has been modified.]

Now to try the second option...

```
Enter field number: 1           # selects the Timer Mode Register (TMR1) for timer 1
Enter value... ?
```
[The bit fields are now displayed on the screen. The down-arrow key (‘j’ in Terminal Mode) scrolls you down until the field that you would like to alter is highlighted. Scrolling down to the ICLK bits would allow us to change

```
Enter option: 17
```

[The count should indeed have changed, since timer 1 is running. Every time you enter option 17 you will be taking a “snap shot” of the values of the timer registers. Try this by leaving this option (with <CR>s), and coming back in again.]

Now let’s turn timer 1 off from inside option 17.

```
Enter field number: 0              # selects TGCR
Enter value... 0000                # Now timer 1 is turned off.
```

Optional

Start up timer 2 by writing $0032 to the TMR2 register. Now enable timer 2 by writing $0010 to the TGCR. This will cause the timer to toggle the TOUT2* pin whenever the reference value is reached (TRR2). To see this on a scope first note that Port A pin 11 is a dual function pin with the TOUT2* signal. The M68360QUADS User's Manual states that PA11 (TOUT2*) may be found on A12 of the PD6 connector. But wait! Before this will work, you have to configure this pin from a Port A I/O pin to a timer pin. This is done in option 16 of the main menu. First set PAODR (Port A open drain register) to $0000 by selecting 2 in this menu. Now program PAPAR (Port A parameter register) to $0800 by selecting 1. Finally, select 0 and program PADIR (Port A direction register) to $0800 enabling the TOUT2* pin as an output. Now the timer pulses should appear on the scope.

Trying out an SCC the Easy Way!

When the User Interface Software starts up, it configures all 4 SCCs into the HDLC protocol. This can be easily changed in the SCC registers. However, for this example the HDLC protocol and loopback (internally connecting the transmit and receive data pins) will work just fine. Regardless of how an SCC is configured, there is a simple way to send a message through that SCC:

Reset the board as before

```
QUICCbug> 360sw                    # to start the user interface software
```

To be able to modify any registers, you must first select the “flip read/write” option:

```
Enter option: 3                    # should put a “+” on the screen. Allows register writes.
```

If you are using a terminal based system (i.e. not using the ADI host software), enable the driver module to

```
Enter field number: 4           # select SCC Mask Register (SCCM)
Enter value.. 001f              # Enable all Interrupts (for the Driver Software)
```

Exit the SCC Registers section by hitting <CR>

```
Enter option: 19                # to access the Baud Rate Generator Registers
Enter field number: 0           # we want to use  BRGC1
Enter value..00010144           # Enable BRG count for a clock to SCC1
Enter field number: <cr>        # exit
```

The last piece of setup is to initialize the SDMA Control Register

```
Enter option: 14                # to access the SDMA Control Registers
Enter field number: 1           # to change SDCR
Enter value..0740               # This register is described on page 7-64 of the UM
Enter field number: <cr>        # exit
```

Try the send-a-telegram option. You will have to pick an SCC for this test (pick 0, which is what the higher layer software modules use to refer to SCC1).

```
Enter option: 27                # Send a Telegram
Enter id:  (0) 0                # selects SCC1
Enter string: hello             # Send the string "hello"
```

After sending the message, you should get an immediate response that says "`Received Telegram:hello`" (your host-based system will not show this message -- instead, the message shows up in the QUICC.EV file in the directory you ran the host software from). The message shows that the driver module software has detected that a frame has been received, and has processed it. The processing also includes removing it from the receive queue of buffer descriptors (BD's). Let's look at the Transmit BD tables.

```
Enter option: 5                 # This will show the TX BD's.
Enter channel...1               # hit return or 1 + return to select channel 1 (SCC1)
Enter First...<CR>              # First TX BD address
0:1c00 0007 <address>           # Contents of the First BD
```

Notice that for SCC1, there is something in the first TX BD. The first value is a status (1c00) longword which has the bit fields and settings displayed. The second is the data length (0007) and the last value is a 32-bit pointer to the buffer location. Write down the value for this pointer, it will be used later.

Use the memory display option to look at the buffer containing “hello” by looking at the location pointed to by the TX data buffer pointer.

```
Enter option: 25
Enter:  address...address 30     # to see the data buffer for BD#1
```

It might be helpful to stop here for a moment and see what was actually transmitted through the SCC1 as configured in HDLC:

Table 1. HDLC Frame Transmission

Hex	Character	Meaning
7e	.	Opening FLAG of HDLC
aa	.	Address Byte 1
68	h	Address Byte 2
65	e	Control Byte
6c	l	Data Byte 1
6c	l	Data Byte 2
6f	o	Data Byte 3
00	.	Data Byte 4
14	.	CRC Byte 1
e9	.	CRC Byte 2
7e	.	Closing FLAG for HDLC

The opening and closing flags and CRC bytes are automatically provided and removed by the 68360, hence they do not appear in the buffer in memory. Notice that YOU are responsible for providing the control, data and address bytes in the data buffer. Upon reception, the 68360 knows nothing about the control and data bytes. Since the driver merely adds an ‘aa’ on to the front of the telegram, it appears that the “aa” and the “h” are actually the address and control bytes of the frame. The 360 will, if told to, do address recognition, but in this case has been told to receive all frames, so no address checking is done. (The zero insertion/deletion function is not shown in the above). Now exit to main menu.

Let’s now look at the RX BD's.

In order to see the message as it comes in memory, we need to first disable the driver's receive process. Exit option 6 (by hitting <CR>) and then:

```
Enter option: b                 # Go to driver menu
Enter option: 6                 # Option 6
Enter child id: 0               # Chooses SCC1
a(ctivate),...: d               # Disables the receive process
```

Note

The (e) doesn't mean that the process was previously enabled, but rather indicates the default value if you hit return alone.

```
Enter option: a                 # Return to a menu
```

Try sending another telegram. Choose a different message this time. Note that this time you will not see the Received Telegram message if you saw it previously, because that part of the driver software is no longer enabled.

```
Enter option: 27                # Send a Telegram
Enter id:  (0) 0                # selects SCC1
Enter string: world             # Send the string "world"
```

Now go look at the RX BDs.

```
Enter option:6
Enter channel...1               # hit return or 1 + return to select channel 1 (SCC1)
Enter First...<CR>              # First RX BD address
```

Notice there is a buffer descriptor for SCC1 with a status field of 0c00. This is the second BD in the list for SCC1 (because the 360 steps through the BDs one at a time). It should have a length of 0009. Why is it 9 bytes long rather than 7 as before? Answer: a 2-byte CRC has been transmitted with the message.

Note the value of the buffer pointer (the last 2 words in that line), and verify the received message (and CRC) using the memory display option 25. Exit and try:

```
Enter option:25
Enter:  address...address 30    # Type in the address of the buffer you found above
[you should see the message "world"]
```

Reset the board as before

```
QUICCbug> 360sw                       # to start the user interface software

Enter option: 3                       # to allow register writes.
```

First we set up the source area of our DMA transfer

```
Enter option: 26                      # Option 26, memory modify
Enter address... 403000               # Start at location 403000
403000 0
403001 1
403002 2                              # hex values 0..f
...
40300f f
403010 !                              # note the "!" is used to quit rather than "."
```

You just used the memory modify option 26 to initialize 16 bytes in the external RAM. A return will not modify an existing value, and a "!" will exit this option. Now we will copy this block of 16 values from $403000 to $403020 using IDMA1.

Verify that the data has been written at $403000 and has not yet been copied to $403020:

```
Enter option: 25
Enter address...403000  50
```

Now before we start manipulating the IDMA1 registers, we will first have to modify a port pin. Why? There are input pins to the 68360 called DONE1* and DONE2* that are used to terminate IDMA operations. For our example, DONE1* will terminate the IDMA1 operation after the first word is transferred, unless it is kept continuously high. There are two ways to make sure this pin stays high. 1) We could go to the PD6 connector on the QUADS board, find DONE1* (B7 on the connector) and pull it high in hardware. 2) Alternatively, we can reconfigure this pin as a general purpose I/O pin, causing the DONE1* signal to be pulled high internal to the chip. We will opt for this software method.

```
Enter option: 22                      #  Enter the Parallel Interface Port (PIP) registers
Enter field number:0                  #  program the PIPC for general purpose I/O pins
Enter value: 0000
Enter field number: 3                 #  program the PBPAR to assign the pin as a general purpose I/O
```

To begin the IDMA operation, the channel mode register (CMR) register needs to be written. We will set the DMA to transfer from memory-to-memory, no interrupt will occur on completion of the transfer, the transfer will occur at a maximum rate (100% of the available bus bandwidth), the source and destination pointers will increment by 2 after each transfer, and the operand size is a longword.

```
Enter field number: 1              # select CMR
Enter value: 0301                  # starts the IDMA transfer
```

[Notice that although the strt bit was set, is shown as 0 rather than 1, because the transfer completes very quickly. Hit <CR> to exit]

```
Enter option: 25                   #  Memory Display
Enter address...403000 50          #  Notice that the data has been copied to 403020
```

Now look back at the DMA registers again to see how they have changed

```
Enter option: 13
Enter channel... 1                 #  select IDMA1
```

[Notice the source and destination pointers are at their final positions. The DMA byte count is now zero, since it counts down from 10. The DMA status register (also known as CSR) is now 01, indicating that the transfer is complete with no errors.]

A Detailed HDLC Example

The following sequence will transmit 4 HDLC frames in loopback mode using SCC1. You may easily modify this sequence to produce many other variations of HDLC transfer (for instance multiple buffers per frame, or transmitting to another board). It is much the same as the "send-a-telegram" option done previously, except that you now have complete control over the SCC.

If you are on a host-based system using the ADI host software, you may turn on the "history record" option and store the following commands in a file for later reuse or editing. This is a powerful feature of the ADS software that allows a special hardware configuration to be set up very quickly. The file is stored on disk, and may be edited as an ASCII file. It is run again using the history run option. The final stage, of course, would be to translate these commands into code residing on the QUADS board, download the code to the board and execute it.

Reset the board as before

```
Enter option: 14                    # the Serial Channel DMA registers
Enter field number: 1               # modify the SDCR
Enter value... 0740                 # to the recommended value
Enter field number: <cr>            # exit

Enter option: 19                    # the Baudrate Generator registers
Enter field number: 0               # use BRGC1
Enter value... 00010144             # for 9600bps
Enter field number: <CR>            # exit

Enter option: 23                    # the Serial Interface registers
Enter field number: <CR>            # to go to the next screen
Enter field number: 2               # modify the SICR
Enter value... 00000000             # to route BRG1 to SCC1
Enter field number: <CR>            # exit

Enter option: 18                    # the Communication Processor
Enter field number: 0               # use the CR
Enter value... 0001                 # to issue the INIT RX&TX Parms Command
[notice that CR is turns into 0000 immediately because the CP automatically clears the register]
Enter field number: <CR>            # exit

Enter option: 7                     # now modify the HDLC SCC Parameter RAM
Enter channel... 1                  # use SCC1
Enter field number: 4               # set MRBLR
Enter value... 10
```

This will allow frames > 16 bytes in length to overflow the current buffer and wrap to the next. This does not actually happen in our example, because our frames are less than 16 in length, but it could easily be changed below.

```
Enter field number: 11              # set the C_MASK
Enter value... 0000f0b8
Enter field number: 12              # and C_PRES
Enter value... 0000ffff             # to standard default values
Enter field number: 1a              # set RFTHR
Enter value... 0001                 # so that RXF is generated after each frame
Enter field number: <CR>            # exit
```

Now that the QUICC has been set up to allow HDLC transmission, we now prepare the BDs so that it can send and receive.

```
Enter option: 5                      # see SCC Tx BDs
Enter channel... 1                   # use SCC1
Enter First... 40                    # BDs start at DPRBASE+$40
```

We set the BDs ready, with First and Last set so that it transmits the opening flag and the closing flag and CRC.

```
Enter field number: 0
Enter value... 8c00 0006 0040 5000
Enter field number: 1
Enter value... 8c00 0007 0040 5010
Enter field number: 2
Enter value... 8c00 0008 0040 5020
Enter field number: 3
Enter value... ac00 0009 0040 5030
```

The wrap bit tells the 68360 to go back to using the first buffer after this buffer is used.

```
Enter field number: <CR>             to exit

Enter option: 6                      # see SCC Rx BDs
Enter channel... 1                   # use SCC1
Enter First... 0                     # BDs start at DPRBASE+$00
Enter field number: 0
Enter value... 8000 0000 0040 5040
Enter field number: 1
Enter value... 8000 0000 0040 5050
Enter field number: 2
Enter value... 8000 0000 0040 5060
Enter field number: 3
Enter value... a000 0000 0040 5070
Enter field number: <CR>             to exit
```

Next we fill memory locations 405000 to 405030 with data. Portions of this data (as specified in the TX BDs) are going to be transmitted in loopback and received into the receive data buffers located at 405040 to 40508 0.

```
Enter option: 26                     # Memory Modify
Enter address... 405000
```

[At this point, four HDLC frames are being transmitted and received in loopback on SCC1.]

```
Enter field number: <cr>        # to get to the next screen
Enter field number: <cr>        # exit

Enter option: 5                 # SCC Tx
Enter channel... 1              # use SCC1
Enter First... 40               # BDs start at DPRBASE+$40
```

[Inspect the Tx BDs. The value of the status word 8c00 has been changed to 0c00 (ready bit cleared), meaning that the buffer has been transmitted.]

```
Enter option: 6                 # SCC Rx
Enter channel... 1              # use SCC1
Enter First... 0                # BDs start at DPRBASE+$00
```

[Inspect the Rx BDs. The value of the status word 8000 has been changed to 0c00, meaning that data has been received. Note that both FIRST and LAST bits are set showing that each transmitted frame fits in one Rx buffer.]

```
Enter option: 25                # Memory display
Enter address...405000 80       # to inspect the Receive data buffers.
```

[Inspect receive data, note that a 16 bit CRC was added to each frame.]

What is the status of SCC1 now? Well, the receiver and transmitter portions of the SCC are still enabled. The SCC is polling TX BD 0 in the table, waiting for its Ready bit to be set. (Why is it polling TX BD 0 rather than TX BD 4? Ans: because the wrap bit was set in TX BD 3). If this buffer descriptor is set up and its Ready bit is set, it will transmit that buffer, and now use the first RX BD in the table to decide where the data is to go.

A Detailed UART Example

The following sequence will transmit 3 UART frames in loopback mode using SCC1. You may easily modify this sequence to produce many other variations of UART transfers. It follows the same format as the HDLC example above.

If you are on a host-based system using the ADI host software, you may turn on the “history record” option and store the following commands in a file for later reuse or editing. The file is stored on disk, and may be edited as an ASCII file. It is run again using the history run option.

Now we set up SCC1 in UART mode...

```
Enter option: 14                # the Serial Channel DMA registers
Enter field number: 1           # modify the SDCR
Enter value... 0740             # to the recommended value
Enter field number: <cr>        # exit

Enter option: 19                # the Baudrate Generator registers
Enter field number: 0           # use BRGC1
Enter value... 00010144         # for 9600bps
Enter field number: <CR>        # exit

Enter option: 23                # the Serial Interface registers
Enter field number: <CR>        # to go to the next screen
Enter field number: 2           # modify the SICR
Enter value... 00000000         # to route BRG1 to SCC1
Enter field number: <CR>        # exit

Enter option: 18                # the Communication Processor
Enter field number: 0           # use the CR
Enter value... 0001             # to issue the INIT RX&TX Parms Command
Enter field number: <CR>        # exit

Enter option: 20                # change SCC's Registers
Enter channel... 1              # use SCC1
Enter field number: 0           # set GSMR_L
Enter value... 00028044         # set SCC into UART Loopback Mode
```

Note

In UART mode, an additional divide-by-16 is used in baud rate generation.

```
Enter field number: <CR>        # to get to the next screen
Enter field number: 0           # set PSMR
Enter value... b000             # to set 8/N/1 UART
Enter field number: 3           # clear the SCCE
Enter value... ffff             # by writing all 1s to it
Enter field number: 4           # clear the SCCM
Enter value... 0000             # so no interrupts are generated
Enter field number: <cr>        # exit
```

Now that the QUICC has been set up to allow UART transmission, we now prepare the BDs so that it can send and receive.

```
Enter option: 5                 # see SCC Tx BDs
Enter channel... 1              # use SCC1
Enter First... 40               # BDs start at DPRBASE+$40
```

The BD Status word has Ready set so that the CP knows it is ready to transmit. The “I” bit is set to so that the transmission of that buffer is seen in the SCCE.

```
Enter field number: 0
Enter value... 9000 0006 0040 5000
Enter field number: 1
Enter value... 9000 0008 0040 5000
Enter field number: 2
Enter value... b000 000a 0040 5000
```

There is absolutely no reason why more than one Tx BD cannot point to the same data buffer area, as shown here.

```
Enter field number: <CR>        to exit
```

Now we set up the Receive BDs

```
Enter option: 6                 # see SCC Rx BDs
Enter channel... 1              # use SCC1
Enter First... 0                # BDs start at DPRBASE+$00
Enter field number: 0
Enter value... 8000 0000 0040 5020
Enter field number: 1
Enter value... 8000 0000 0040 5040
Enter field number: 2
Enter value... a000 0000 0040 5060
Enter field number: <CR>        to exit
```

Next we fill memory locations 405000 to 40500f with data. Portions of this data (as specified in the TX BDs) are going to be transmitted in loopback and received into the receive data buffers located at 405020 to 405070.

```
Enter option: 26                # Memory Modify
Enter address... 405000
Enter field number: <cr>          # to get to the next screen
Enter field number: <cr>          # exit


Enter option: 5                   # SCC Tx
Enter channel... 1                # use SCC1
Enter First... 40                 # BDs start at DPRBASE+$40
```

[Inspect the Tx BDs. The value of the status word 9000 has been changed to 1000 (ready bit cleared), meaning that the buffer has been transmitted.]

```
Enter option: 6                   # SCC Rx
Enter channel... 1                # use SCC1
Enter First... 0                  # BDs start at DPRBASE+$00
```

[Inspect the Rx BDs. Notice that the first two transmit buffers fit entirely in the first receive buffer. The third transmit buffer is split receive buffers one and two. The third receive buffer is still empty since it was never used. Notice that buffer two has a different status than buffer one. The "1" in 0100 means that the second buffer was closed due to the line being IDLE for a certain amount of time. (That time is programmable with the 68360 -- the Maximum IDLE Characters value in the UART Specific Parameter RAM).]

```
Enter option: 25                  # Memory display
Enter address...405000 80         # to inspect the Receive data buffers.
```

[Inspect receive data, note that a 16 bit CRC was added to each frame.]

What is the status of SCC1 now? Well, the receiver and transmitter portions of the SCC are still enabled. The SCC is polling the first TX BD (TX BD 0) in the table (since the wrap bit was set in BD 3), waiting for its Ready bit to be set. If this buffer descriptor is set up and its Ready bit is set, it will transmit that buffer, and now use the third RX BD (RX BD 2) in the table to decide where the data is to go.