



Column #46, February 1999 by Lon Glazner:

Storing Data With The RAMPack B

A problem commonly faced by Stamp users is how to store data that can sometimes come in overwhelming quantities. We're not talking just a few bytes here and a few bytes there. We're talking about a few hundred bytes — or more — that need to be stored for later use. If you ever wanted to store a text file, and parse the data out to some serial device, or have a string of serial data stored automatically for you, then the RAMPack B may fit the bill.

More importantly, we'll be using this month's RAMPack B (RPB) overview as a stepping stone. Next month, we'll use the serial data storage capacity of the RPB to store graphic bit-map (*.BMP) data files. These files will be downloaded directly from your PC into the RPB. Next month's article will also use the Scott Edwards Electronics, Inc., G12032 Mini Serial Liquid Crystal Display (G12032 LCD). With the RPB and the G12032 LCD, developing graphic displays becomes a relatively simple task. Some limited animation with the graphic display is also possible using these two devices in tandem.

In this article, we will start off covering some of the basic functions of the RPB, briefly discuss data conversion, and finish up with how to use the First-In-First-Out (FIFO) buffer mode to store serial data automatically. But first, an overview of electronic memory "types."

Data Memory Discussion

There are quite a few ways to store data in today's electronic devices. There are typically four areas of concern that need to be addressed when selecting a memory device. These areas are speed, volatility of memory, input/output (I/O) pins required to access the memory, and how often the memory can be written to.

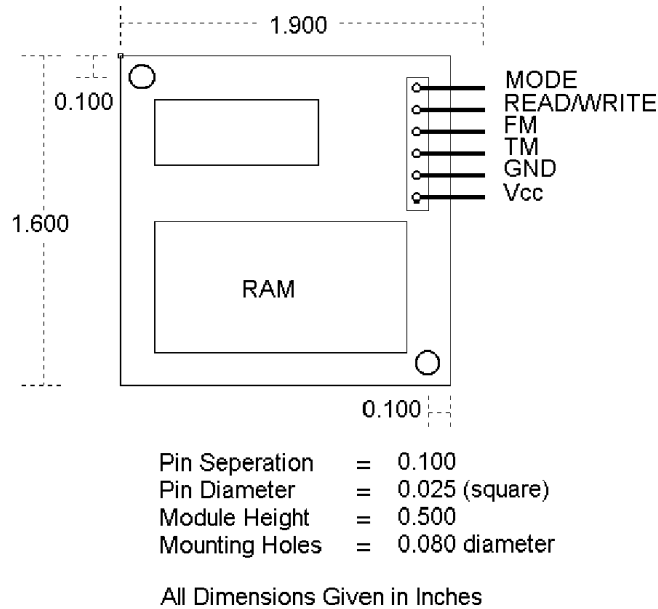
For example, with serial EEPROM (Electrically Erasable PROM), such as Microchip's 24LCxx series, you gain the dual benefits of a low pin count to access the memory and non-volatile memory (stored data is retained after power is removed). With EEPROM, you can usually access memory with two or three I/O pins depending on the type of memory you use. Parallax provides application notes for interfacing a BASIC Stamp to EEPROM. In fact, the BASIC Stamp has EEPROM on board that can be accessed through the PBASIC READ and WRITE commands. EEPROM is usually an excellent vehicle for the storage of data in electronic designs. It is low-cost, easy-to-use, and has memory that is retained even after power is lost.

There are only three downsides to EEPROM. First, EEPROM is usually only guaranteed for one million erase/write cycles. This may seem like a lot, but if you were to update a memory location every three seconds, the EEPROM could fail after only 35 days of use. Secondly, interfacing to EEPROM is not always the easiest thing for beginners to accomplish. Finally, compared to other memory types, access time for EEPROM is considered slow. For the experienced Stamp user, the second shortcoming is of little consequence, and the third isn't entirely relevant because of the operating speed of BASIC Stamps.

SRAM (Static RAM), on the other hand, has no limitation on how many read/write cycles it can be used for, but requires a large number of I/O pins to address memory locations. In fact, with no additional logic buffers, more than 22 I/O pins could be required. The additional problem of volatile memory (data is lost when power is removed) usually precludes SRAM from being an attractive memory storage device for Stamp-based designs. SRAM, and RAM in general, has one additional advantage of fast access time that makes it attractive to designers using much faster processors than the BASIC Stamp.

There are additional types of RAM including Dallas Semiconductor NVRAM (non-volatile RAM). NVRAM provides a back-up lithium power source to retain data in RAM when power is removed from the system. This is an example of an after-market addition to RAM that eliminates one of RAM's inherent shortcomings.

Figure 46.1: RAM Pack B dimensions



The RPB (Figure 46.1) is another such attempt to delete the shortcomings of RAM-based memory. The RPB is kind of like “smart RAM.” With the RPB, access to SRAM can be accomplished with a single I/O pin. The communication format used by the RPB is the very common 8N1 (eight data bits, no parity, one stop bit) serial communication. This is the same format as is used in many devices such as serial printers, PCs, GPS receivers, and the BASIC Stamp. In the RPB, SRAM comes socketed in the module. This SRAM can be removed and replaced with NVRAM. The RPB normally comes with 8Kx8 byte SRAM. 8Kx8 byte SRAM has room to store 8,192 bytes of data. The user can replace the 8Kx8 byte SRAM with 32Kx8 byte SRAM and extend the storage capability of the RPB to 32,768 bytes of data.

I used a Dallas Semiconductor DS1230Y-150 32Kx8 byte NVRAM in the RPB used in this design. I tend to keep one of these lying around for test purposes. There is no reason that the 8Kx8 byte SRAM that the RPB is sold with can't be used for any of the functions described in this article.

Using the RAMPack B WRITE and BYTE_READ Commands

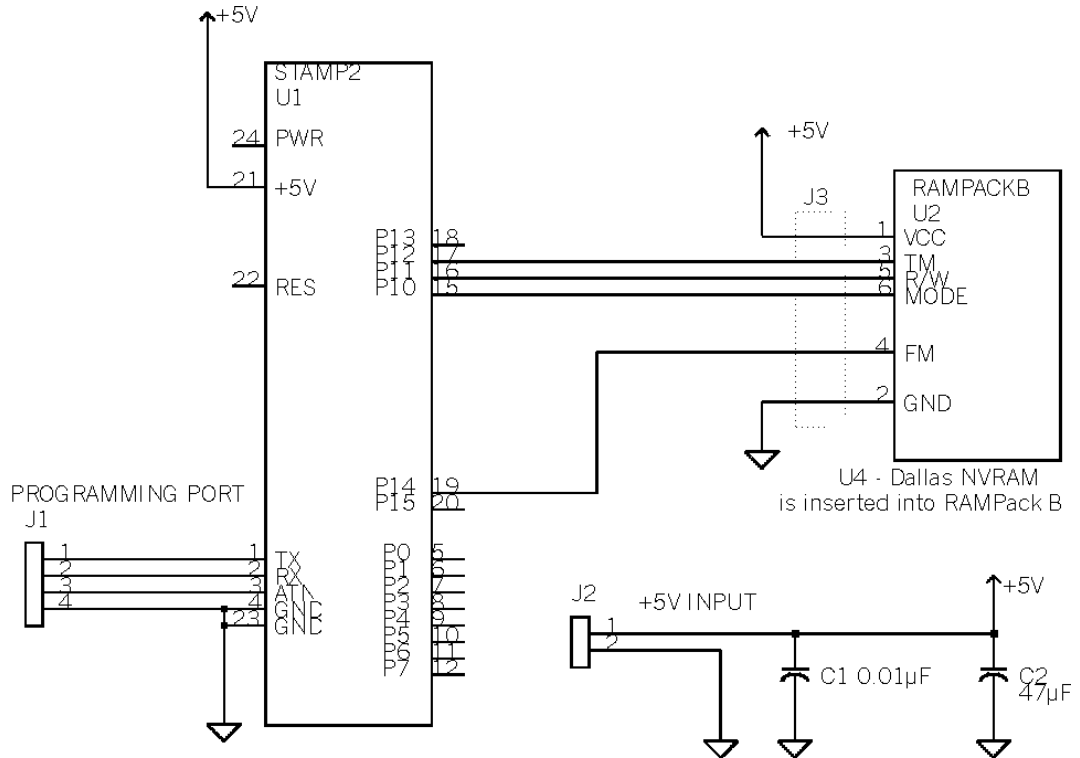
To start with, we'll be taking a look at just the WRITE and BYTE READ commands of the RPB. All commands sent to the RPB must be prefaced with a \$55, or '55'hex. This is a binary '01010101', and is used by the RPB to calculate the baud rate that the Master unit is using to send data. In this design, the Master unit is a BASIC Stamp 2. For more information on the RPB data formats, I would recommend downloading the latest data sheet from "www.solutions-cubed.com."

Code Listing 46.1 (RPB1_299.BS2) shows one example of writing and reading data from the RPB using just the WRITE and BYTE READ commands. The WRITE command discussed here should not be confused with the PBASIC WRITE command, which accesses EEPROM on-board the BASIC Stamp. All Solutions Cubed Mini-Mods — of which the RPB is one — can communicate using the PBASIC SEROUT and SERIN commands.

In this program, part of the ASCII character set is written to the RPB with the WRITE command. These ASCII values are then read back to the BASIC Stamp 2 using the BYTE READ command. Figure 46.2 can be used as a hook-up, or wiring diagram, if you want to try this yourself. This connection scheme uses two I/O pins to communicate with the RPB. You could connect the TM and FM pin of the RPB together and run this program using one I/O pin on the BS2. Multiple baud rates are used in this code example to display how the communication baud rate to and from the RPB can be changed on the fly.

The RPB WRITE command follows a simple format. Figure 46.3 displays the serial format in hexadecimal values. The decimal equivalents work for these values as well. The data string for a WRITE command always starts with the sync byte ('55'hex), then the command byte ('00'hex), followed by the high address byte, and low address byte, for the address location that the WRITE is to begin at. For example, to start writing data at address location 1000 ('03E8'hex), you would send '03'hex as the high address byte, and 'E8'hex as the low address byte. After this data is received by the RPB, it will store any subsequent bytes of data sequentially in RAM. The RPB will exit the WRITE subroutine if there is no data received after a period of time. The time it takes for this to occur is based on the baud rate that data is being received at. For 9600 baud, the time the FM line must remain high for a WRITE to end is roughly 2ms.

Figure 46.2: Hookup diagram with RAM Pack B and BASIC Stamp



For the BYTE READ command, the RPB expects the sync byte ('55'hex), then the command byte ('01'hex), and finally the high and low address bytes for the memory location that you would like to read. The RPB will send the data byte requested on the TM line in 500us. Figure 46.3 displays this function with data displayed in hexadecimal format.

Storing Serial Data Automatically With The RAMPack B

While the RPB WRITE and BYTE READ commands are useful, they don't solve a very common problem Stamp users run into. It's difficult to store large amounts of serial data for later use by the Stamp. Using the circuit in Figure 46.4 and the RPB's FIFO mode, you can easily interface to the RPB to store serial data from a PC terminal program.

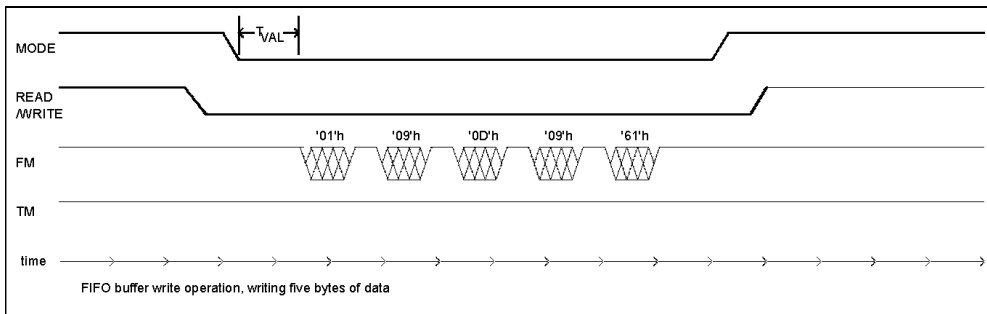
Some explanation of the circuit in Figure 46.4 is required. The transistor Q1 — along with D1, R1, and R4 — acts to invert and limit data signals from the PC serial port to the BS2 and RPB. This could be considered a “poor-man’s” RS-232 converter. Data received at the RPB’s FM pin from the PC will be at roughly TTL levels. R2 and D2 just provide a visual indication when data is being transmitted. The red LED, D2, will flash when data is present at the FM pin.

One of the trickier aspects to using the RPB FIFO mode in this configuration lies in the need to send commands to the RPB. Because the Master unit (BS2) has to share the FM pin with the serial data source (PC in this case), the BS2 must have the ability to seize the serial data line. This is accomplished by connecting the P15 (FMEN in code) of the BS2 to the emitter of Q1. If P15 (FMEN) of the BS2 is an output high (5VDC), then Q1 is effectively removed from the circuit. In this configuration, P14 (FMBS2) can be used to send commands to the RPB.

Current can flow from the emitter of Q1 to the collector with this kind of switch. So it is possible to damage an I/O pin if R1 and R4 are not large enough. This would happen if Q1 were to be turned on while its emitter was held high (5VDC) and the BS2 was outputting a logic low (0VDC). The current that the BS2 is required to sink for the circuit in Figure 46.4 was only 6mA. This value agreed with a SPICE simulation of the same circuit. In the lab, I reduced the values of R1 and R4 to 1.0K ohm resistors. This increased the current that the BS2 had to sink to about 30mA. If you use this circuit, I would recommend sticking with the values in Figure 46.4. You should also write your code to minimize the possibility of the BS2 writing to the RPB while your serial data source is sending data.

It is also necessary to give the serial data source (PC) access to the RPB. When the BS2 outputs a logic low (0VDC) at P15 (FMEN), it provides a ground path for the emitter of Q1. This effectively places Q1 in the circuit. When P15 (FMEN) is low, then P14 (FMBS2) should be configured as an input pin. P11 (R_W) and P10 (MODE) are also used to place the RPB in FIFO mode. In this application, the FIFO write operation is used to automatically store data sequentially in RAM with the RPB. Figure 46.5 shows the proper pin configurations for the RPB MODE and READ/WRITE pins for a FIFO write operation. Tval is the time from executing a FIFO write operation until serial data will be accepted. The maximum time to expect here is 20us.

Figure 46.5: RAM Pack B FIFO Write Operation



Code Listing 46.2 (RPB2_299.BS2) stores data from a text terminal into the RAM residing on the RAMPack B. This routine waits for the ASCII characters “START” to be received by the BS2 before executing a FIFO write operation. Any text typed into the data terminal after the “START” characters are received will be stored sequentially in RAM. When the ASCII characters “STOP” are received, the BS2 will read the data stored in the RPB and display it via the Debug command. The RPB baud rate is set to 9600 baud just after the “START” is received. Up to 8,192 characters could be stored in RAM using this routine and the SRAM that an RPB is sold with.

I should note a sticky point in this design. The “START” that is received by the BS2 is also received by the RPB on its FM pin. Since these characters do not meet the typical command structure that the RPB expects (sync byte, then command byte, etc.), the RPB will execute a system reset. This reset returns the RPB to a default condition which has the baud rate set to 2400. Sending any valid command from the BS2 to the RPB at 9600 baud prior to executing a FIFO write operation resets the internal baud rate of the RPB to 9600 baud.

In RPB2_299.BS2, this is done by resetting the RPB’s internal address pointer with the WRITE POINTER command ('05'hex). This command resets the internal address pointer to '0000'hex, as well as setting the RPB baud rate to 9600.

The hassle of dealing with the effects that the “START” characters have on the RPB can be easily removed. The BS2 could simply execute a FIFO write operation, and then wait for the “STOP” characters. In this manner, all received data would be loaded into the RPB. Since this design interfaces to a human typist, through the PC, adding additional Debug routines and commands can be accommodated. The Debug commands used in this routine might cause unnecessary delays and can be removed in designs where timing is critical.

In instances where data is received as one long string — such as from GPS receivers — it is best to execute a FIFO write operation immediately and wait until a particular end-of-string message is received. In this design, the “STOP” constitutes the end-of-string message.

Figure 46.6: Screen Capture From HP-54645D MSO of RPB2_299.BS2

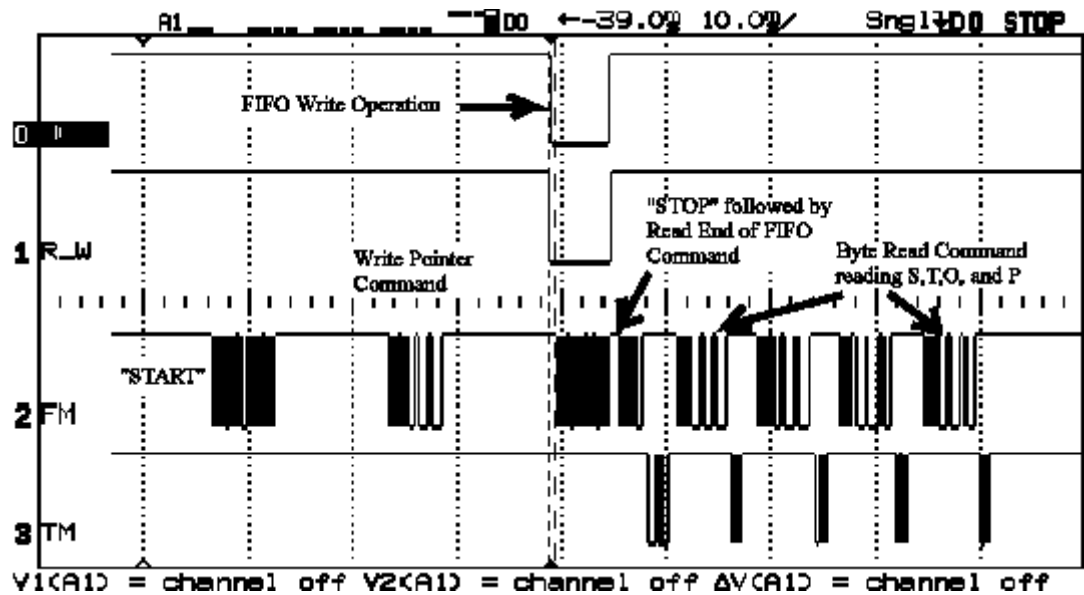


Figure 46.6 displays a screen capture from an HP-54645D MSO (Mixed Signal Oscilloscope). This image is of a slightly-modified version of RPB2_299.BS2. From it, you can see how the data flows from the serial data source to the RPB, and then from the RPB to the BS2. I should note that my serial data source was a PC utilizing a common terminal program found on most IBM-style computers available today. The communication parameters that I used were 8N1 serial data, 9600 baud, no-handshaking, and local-echo “on.”

Column #46: Storing Data With The RAMPack B

In Closing

Most of the difficulty in interfacing the RAMPack B's FIFO mode to the BS2 concern the pin sharing requirements for TM and FM pins. These difficulties can usually be overcome with a minimum of additional electronic parts. Often times, the BASIC Stamp is expected to operate as kind of a "traffic director" for serial data. In these instances, the RAMPack B can be effectively used to buffer serial data.

Next month, we'll take a look at pin sharing with the TM pin, as well as the FM pin. Sharing both pins allows the BASIC Stamp and RPB to act as traffic director and data buffer in a myriad of configurations. One such configuration will be covered in the March installment. Next month, we'll trap serial data from a PC, and then send the data to the Scott Edwards Electronics G12032 LCD. The same circuitry we used in Figure 46.4 will be added to, to accommodate the sharing of the TM pin, and to convert from non-inverted serial data to inverted serial data. The end result will be a device that records bit-maps from your PC and can then display these bit-maps to a graphic LCD.

```
' Program Listing 46.1. RPB1_299.BS2 ASCII Loop Program
' RPB1_299.BS2 - ASCII Loop Program: this Basic Stamp2 program writes part
' of the ASCII character set to the RAMPack B at 4800 baud. It then reads
' the data back in reverse order at 2400 baud. This data is displayed
' using the Debug command. Finally, the data is read back at 9600 baud in
' the order in which it was stored, this data is also displayed via the
' Debug command.
'
'*****
'Program Variables
'
Address VAR      Word
'Stores address in RAM to write or read from
Addr_lo VAR     Address.lowbyte
'Address is broken into bytes for sending to the RPB
Addr_hi VAR     Address.highbyte
'Address is broken into bytes for sending to the RPB
Baud   VAR      Word
'A word variable is used to store the data rate
DataByte VAR     Byte
'Used to hold data to be sent or received
A      VAR      Byte
'For...Next variable
'*****
'Program Constants
'
FMBS2  CON      14      'From Master pin: sends data/commands to the RPB
TMBS2  CON      12      'To Master pin: receives data from the RPB
B_Write CON     $00     'Write command for the RPB
B_Read  CON     $01     'Byte read command for the RPB
```

Column #46: Storing Data With The RAMPack B

```
'*****
'Initialization of registers
  Pause100           'Allow some time for RPB to power up
  Address = 0        'Default address is 0
  DataByte = 33     'Start data at ASCII "!"
'*****
MainProgram:
'Write data to RPB at 4800 baud
'
  Baud = 188         'Write data at 4800 baud
  For A = 1 to 58   'Send ASCII "!" through "Z"
    Serout FMBS2,Baud,[$55,B_Write,Addr_hi,Addr_lo,DataByte]
    Pause 20        'Allow time for WRITE to end
    DataByte = DataByte + 1 'Point to next ASCII character
    Address = Address + 1 'Point to next address
  Next
'
'Read data from RPB at 2400 baud, reads data in reverse order of how
'data was written to RAM
Baud = 396          'Read data at 2400 baud
For A = 1 to 58
  Address = Address - 1
'Read starting at last byte written
  Serout FMBS2,Baud,[$55,B_Read,Addr_hi,Addr_lo]
  Serin TMBS2,Baud,[DataByte]
  Debug DataByte, " " 'Display byte received
Next
'
'Read data from RPB at 9600 baud, reads data in the same order as data was
'written to RAM

Baud = 84           'Read data at 9600 baud
Debug CR,CR        'Place a blank line between data sets
For A = 1 to 58
  Serout FMBS2,Baud,[$55, B_Read,Addr_hi,Addr_lo]
  Serin TMBS2,Baud,[DataByte]
  Debug DataByte, " " 'Display byte received
  Address = Address + 1 'Read data start at first byte written
Next
END:               'End of program
```

Column #46: Storing Data With The RAMPack B

```
' Program Listing 46.2. RPB2_299.BS2 Storing Serial Data Automatically
' with the RPB
'RPB2_299.BS2 - Storing Serial Data Automatically with the RPB:
'This program waits for the characters "START" to be received. Once
'received the RPB has its internal baud rate reset to 9600 by the WRITE
'POINTER command. This command also has the effect of pointing any
'subsequent FIFO write operations to address location '0000'hex. Data is
'then loaded into the RPB while it is in FIFO write mode until the
'BS2 receives the characters "STOP". The READ END OF FIFO command is used
'to find the last address pointed to by the FIFO write operation. This
'value is always the same as the number of bytes stored during
'the FIFO write operation. The last byte of data written during a FIFO
'write operation will be at one less than the value returned by the READ
'END OF FIFO command.
'
'*****
'Program Variables
'
Address      VAR      Word
'Stores address in RAM to write or read from
Addr_lo      VAR      Address.lowbyte
'Address is broken into bytes for sending to the RPB
Addr_hi      VAR      Address.highbyte
'Address is broken into bytes for sending to the RPB
DataByte     VAR      Byte
'Used to hold data to be sent or received
End_Addr     VAR      Word
'Stores Read End of FIFO pointer
End_Addr_lo  VAR      End_Addr.lowbyte
'Pointer broken into bytes for receiving from the RPB
End_Addr_hi  VAR      End_Addr.highbyte
'Pointer broken into bytes for receiving from the RPB
'*****
'Pin Assignments
'
FMEN         CON      15  'Enables/disables data from serial data source
FMBS2        CON      14  'From Master pin: sends data/commands to the RPB
TMBS2        CON      12  'To Master pin: receives data from the RPB
R_W          CON      11  'Read/Write pin: selects type of FIFO operation
MODE         CON      10  'Mode pin: enables/disables FIFO mode
'*****
'Communication Constants
'
Sync         CON      $55  'Sync byte
B_Read       CON      $01  'Byte Read command
W_Pointer    CON      $05  'Write Pointer command
EOF_Pointer  CON      $07  'Read End of FIFO command
Baud         CON      84   '9600 baud
'*****
'Initialization of registers
```

Column #46: Storing Data With The RAMPack B

```
Initialize:
  High  FMEN      'Disable data from serial data source
  High  MODE      'Disable FIFO mode
  High  R_W       'Default to FIFO read
  Pause 500       'Allow RPB time to power up
  debug CR,CR,"Initialized",CR
*****
MainProgram:
'Wait for "START" from serial data source, display message when received
'
  Low   FMEN      'Enable data from serial data source
  Serin FMBS2,Baud,[wait("START")]
  debug "START Received",CR
  High  FMEN      'Disable data from serial data source
'
'At this point the RPB has reset itself due to the "START" data and has
'returned to a default baud rate of 2400. The Write Pointer command is
'used to reset the address that the FIFO write operation will start at,
'and reset the RPB baud rate to 9600 for further data.
  Address= 0      'Pointer will be reset to '0000'hex
  Serout FMBS2,Baud,[Sync,W_Pointer,Addr_hi,Addr_lo]
  Input  FMBS2    'Make FMBS2 an input pin
  debug  "Resetting FIFO Pointer",CR
  debug  "Entering FIFO Buffer Mode",CR
  Pause  100
'
'Enable FIFO write operation and wait for "STOP" to be received
  Low   FMEN      'Enable data from serial source
  Low   R_W       'Select FIFO write
  Low   MODE      'Enable FIFO buffer
  Serin FMBS2,Baud,[waitSTOP)]
  High  MODE      'Disable FIFO buffer after "STOP"
  High  R_W       'Default to FIFO read
  High  FMEN      'Disable data from serial source
  Debug "STOP received",CR,CR
'
'The Read End of FIFO command is used to determine how many bytes of data
'were stored during the FIFO write operation. This value is used to set up
'a For-Next loop to read back all of the data and display it using the
'Debug command.
  Serout FMBS2,Baud,[Sync,EOF_Pointer]
  Serin  TMBS2,Baud,[End_Addr_hi,End_Addr_lo]
  Debug  "Number of bytes in message = ",Dec End_Addr,CR
  Debug  "Data received after START is...",CR
  For    Address = 0 to End_Addr-1
' Read data from '0000'hex to (End_Addr - 1)
  Serout FMBS2,Baud,[Sync,B_Read,Addr_hi,Addr_lo]
  Serin  TMBS2,Baud,[DataByte]
  Debug  DataByte 'Display data read
```

Column #46: Storing Data With The RAMPack B

```
Next  
Goto   Initialize   'Start over  
END:  
END:   'End of program
```