

# Master SPI Bus Engine



## 1.0 INTRODUCTION

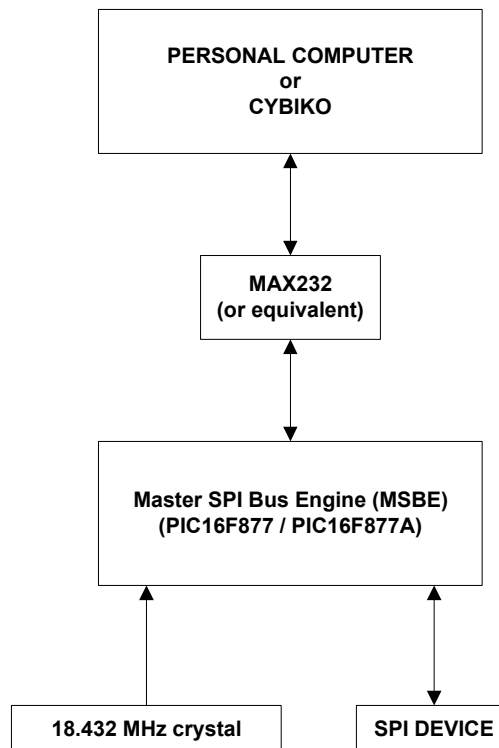
The Master SPI Bus Engine (MSBE) is an asynchronous serial to synchronous SPI (Serial Peripheral Interface) bus hardware bridge that allows users to quickly and easily communicate with SPI devices via any serial port that handles asynchronous serial communications at 115,200 bps with a 8N1 data format (8 data bits, no parity, 1 stop bit). The Master SPI Bus Engine supports ASCII-based human-interactive commands (for use with a serial terminal or serial terminal emulator) for quick and easy communication with SPI devices. In addition, the Master SPI Bus Engine supports a separate binary command set that allows users to build SPI bus Personal Computer (PC) applications with their favorite high-level language such as C/C++, Visual Basic, etc.

Since the MSBE is hosted by a Microchip PIC16F877 or PIC16F877A microcontroller that has an on-board SPI bus engine, the user need not get bogged down with the actual implementation of the SPI bus engine's firmware low-level functions. The MSBE can be configured to use several SCK clock frequencies (288 KHz, 1152 KHz, and 4608 KHz), and any of the four standard SPI bus modes can be selected: Mode 0 (0,0), Mode 1 (0,1), Mode 2 (1,0) and Mode 3 (1,1). All user-configurable parameters are sticky settings (stored in the PICmicro's nonvolatile data EEPROM memory) and thus will be remembered upon subsequent power-up of the Master SPI Bus Engine.

The MSBE allows communication with up to four SPI slave devices due to its four active-low chip select outputs (/CS #1, /CS #2, /CS #3, and /CS #4).

If you ever wanted to have a master SPI bus port on your PC, then the Master SPI Bus Engine is for you!

## 2.0 SYSTEM BLOCK DIAGRAM



**NOTE:** Don't forget the details:

- 1) Supply power to all chips and use adequate power supply decoupling capacitors in close proximity to all chips.
- 2) Use adequate crystal load capacitors.
- 3) The PIC16F877/PIC16F877A pins RB0, RB1, RB2, RB4 are outputs that become SPI slave chip selects /CS #1, /CS #2, /CS #3, and /CS #4, respectively.

The PIC16F877/PIC16F877A microcontroller must be supplied with an 18.432 MHz parallel-cut crystal and appropriate loading capacitors. Consult the crystal manufacturer's data sheet for capacitor values, but values in the range of 12 pF to 33 pF should be a good starting point.

A MAX232 or equivalent RS-232 transceiver IC (integrated circuit) must be used if the host's asynchronous serial port implements the RS-232 electrical specification. Don't forget to connect the host's serial port ground to the Master SPI Bus Engine ground!

### 3.0 PIC16F877/PIC16F877A PIN USAGE

From a hardware perspective, the Master SPI Bus Engine is simple to wire up. The Master SPI Bus Engine allows control of up to four slave SPI devices through its four active-low chip select outputs. The following list shows a PICmicro package-independent pin usage chart to aid in wiring up your Master SPI Bus Engine.

NOTE:  $V_{DD}$  must be 4.5V to 5.5V.  
Don't forget to place decoupling capacitors as close to the PICmicro as possible.  
Keep wiring neat and keep component leads short.

<u>PIN NAME</u>	<u>PIN USAGE</u>
/MCLR/V <sub>PP</sub>	Connect to $V_{DD}$ via a 10K ohm pull-up resistor (or use your favorite reset circuit/reset supervisor IC).
V <sub>DD</sub>	Connect <b>BOTH</b> V <sub>DD</sub> pins to your 4.5V to 5.5V supply.
V <sub>SS</sub>	Connect <b>BOTH</b> V <sub>SS</sub> pins to your power supply ground.
OSC1, OSC2	Connect your 18.432 MHz parallel-cut crystal to these pins. Don't forget to use crystal load capacitors.
TX	Asynchronous serial port transmit line - Connect to your MAX232 (or equivalent) RS-232 transceiver chip.
RX	Asynchronous serial port receive line - Connect to your MAX232 (or equivalent) RS-232 transceiver chip.
SCK	Connects to SPI slave device SCK line (serial clock line) or equivalent (device may use different pin terminology).
SDI	Connects to SPI slave device SDO line (serial data out line) or equivalent (device may use different pin terminology). (Note: Do not allow the SDI pin to float. Certain applications, even if the SDI pin is used, may require a pull-up or pull-down resistor on this pin to prevent excessive current consumption due to the possible oscillation of a floating pin.)
SDO	Connects to SPI slave device SDI line (serial data in line) or equivalent (device may use different pin terminology).
RB0	/CS #1 active-low output (chip select #1 connects to chip select of SPI slave device #1).
RB1	/CS #2 active-low output (chip select #2 connects to chip select of SPI slave device #2).
RB2	/CS #3 active-low output (chip select #3 connects to chip select of SPI slave device #3).
RB4	/CS #4 active-low output (chip select #4 connects to chip select of SPI slave device #4).

**Note:** Do not allow the SDI pin to float. Certain applications, even if the SDI pin is used, may require a pull-up or pull-down resistor on this pin to prevent excessive current consumption due to the possible oscillation of a floating pin. Consult the data sheets of any slave SPI devices you will use.

That's it. All other pins not named above may be left unconnected as they are configured as outputs.

**Question:** How will I know that the Master SPI Bus Engine is working?

**Answer:** Upon power-up of the PICmicro that hosts the Master SPI Bus Engine (or by a software reset command), the Master SPI Bus Engine will perform an ASCII dump to its serial port via the TX line (the display may vary somewhat depending on the firmware version or whether sticky settings were changed, but you should see something like the following):

```
Master SPI Bus Engine
v1.1 (C) 2004 Ken Pergola

ASCII Commands: 0 1 2 3 4
S R # _ ^ " * ! { } ?

SPI Bus Mode:      0,0
SCK Frequency:    288 KHz
SCK Idle State:    LOW
Tx on SCK:        ACTIVE->IDLE
Input Sample Phase: MID
```

## 4.0 SERIAL TERMINAL SETUP

Although the Master SPI Bus Engine was primarily designed for the Cybiko classic handheld computer using the VTterm application, it will work with most standard PC-hosted serial terminal emulators such as:

Mtty Serial Terminal  
HyperTerminal  
Etc.

**NOTE:** If you wire up your Master SPI Bus Engine such that it will be used with a PC's serial port without a null modem adapter or cable, the Cybiko will require a null-modem adapter. The converse is true as well: that is, if you wire up your Master SPI Bus Engine such that it will be used with a Cybiko's serial port without a null modem adapter or cable, a connection to a standard PC's serial port will require a null-modem adapter.

### 4.1 USING A PC-HOSTED SERIAL TERMINAL

Terminal UART settings:

115,200 bps  
8N1 (8 data bits, no parity, 1 stop bit)  
No hardware or software flow control  
Local echo off

### 4.2 USING THE CIBIKO WIRELESS INTER-TAINMENT SYSTEM

Cybiko application: VTterm v1.2 (written by Jeff Frohwein: <http://www.devrs.com/cybiko/download.php - apps>)

If the above link does not work, an Internet search of 'VTterm' should allow you to quickly find this file. Download vtterm.zip. Once unzipped, the two files you need to load onto your Cybiko are VTterm.app and ComPort.dl.

If you will be using the Cybiko because of its portability, invoke VTterm (VT100 emulator for Cybiko) and ensure its settings are configured as follows: The settings screen is invoked within VTterm by hitting the Cybiko's 'Select' button.

#### Terminal Settings

Baud Rate	115200
Data Bits	8
Stop Bits	1.0
Parity	None
Handshake	None
Font Size	5x7
Local Echo	No

**IMPORTANT NOTE:** It might be a good idea to disable the power saver mode (suspend mode) on the Cybiko when you are using VTterm. It has been noticed in some cases that after a Cybiko goes into sleep mode and is later awoken from sleep mode, VTterm does not communicate with the Master SPI Bus Engine. While it may appear that the Master SPI Bus Engine is locked up, it is not. What fixes this problem in the short-term is shutting down VTterm and re-starting it. For a long-term fix, just remember to disable the Cybiko's power saver mode: <Fn><F6> (Disables automatic switch to Suspend Mode). Disabling power saver mode can be done at any time (including while VTterm is running).

## 5.0 USER-INTERACTIVE ASCII COMMANDS

Master SPI Bus Engine ASCII COMMAND FUNCTION LIST	ASCII COMMAND
UNASSERT_ALL_CHIP_SELECTS	0
ASSERT_CHIP_SELECT_1	1
ASSERT_CHIP_SELECT_2	2
ASSERT_CHIP_SELECT_3	3
ASSERT_CHIP_SELECT_4	4
SEND_USER_DATA_AND_RECEIVE_BYTE	S
SEND_DUMMY_DATA_AND_RECEIVE_BYTE	R
SET_SCK_FREQUENCY	#
SET_SCK_IDLE_STATE_LEVEL	_
SET_DATA_OUTPUT_ON_SCK_EDGE	^
SET_DATA_INPUT_SAMPLE_PHASE	“
CLEAR_SCREEN	*
SOFTWARE_RESET	!
TEST_MODE_1	{
TEST_MODE_2	}
HELP	?

These commands allow the user to manually communicate with SPI devices interactively via a serial terminal emulator running on a host PC or Cybiko hand-held computer.

**IMPORTANT NOTE:** *Before using the Master SPI Bus Engine to talk to any SPI device, it must be configured properly for the target SPI device (in terms of SPI bus mode, clock frequency, input sample phase, etc.) via the ASCII commands, otherwise communication with the target SPI device may not yield the results expected. Please refer to the SPI device manufacturer's data sheet for details. Also ensure that the chip select line connected to the target SPI device is controlled properly as well.*

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
UNASSERT_ALL_CHIP_SELECTS	0

**Command function description:**

Unasserts all chip select output lines (/CS #1, /CS #2, /CS #3, and /CS #4).

*User types the following character on the keyboard: 0*

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
ASSERT_CHIP_SELECT_1	1

**Command function description:**

Unasserts all chip select output lines and then asserts chip select #1 (/CS #1).

*User types the following character on the keyboard: 1*

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
ASSERT_CHIP_SELECT_2	2

**Command function description:**

Unasserts all chip select output lines and then asserts chip select #2 (/CS #2).

*User types the following character on the keyboard: 2*

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
ASSERT_CHIP_SELECT_3	3

**Command function description:**

Unasserts all chip select output lines and then asserts chip select #3 (/CS #3).

*User types the following character on the keyboard: 3*

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
ASSERT_CHIP_SELECT_4	4

**Command function description:**

Unasserts all chip select output lines and then asserts chip select #4 (/CS #4).

*User types the following character on the keyboard: 4*

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
SEND_USER_DATA_AND_RECEIVE_BYTE	S

**Command function description:**

Sends a user-defined data byte over the SPI bus, and reads/displays the data byte received from the slave device.

*User types the following character on the keyboard: S*

*User will then be prompted to enter a byte (in hexadecimal) to send to the SPI slave device.*

**EXAMPLE:**

*If you want to send 0x0C to the slave device, type the following at the keyboard: S0C*

**NOTE:** Uppercase 'S' and lowercase 's' are permissible.

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
SEND_DUMMY_DATA_AND_RECEIVE_BYTE	R

**Command function description:**

Sends a 'dummy' data byte over the SPI bus, and reads/displays the data byte received from the slave device. This command is used when you are interested in receiving data from the SPI slave device, but the byte sent is a 'DO NOT CARE' or 'dummy' byte.

*User types the following character on the keyboard: R*

**NOTE:** Uppercase 'R' and lowercase 'r' are permissible.

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
SET_SCK_FREQUENCY	#

**Command function description:**

Allows the selection of the SCK (serial clock) frequency.

*User types the following character on the keyboard: #*

*The 'SCK Frequency' will be displayed to the user and will cyclically toggle between 288 KHz, 1152 KHz, and 4608 KHz (upon subsequent invocations of this command).*

**NOTE:** This is a sticky setting that will be remembered and loaded upon Master SPI Bus Engine power-up.

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
SET_SCK_IDLE_STATE_LEVEL	_

**Command function description:**

Allows the setting of the SCK idle state level (referred to as the clock polarity **CKP** bit in Microchip's documentation, or **CPOL** in Motorola's SPI documentation). The idle state of the SCK signal is either low or high, and the active state of SCK is the opposite of the idle state of SCK.

*User types the following character on the keyboard: \_ (underscore character)*

*The 'SCK Idle State' mode will be displayed to the user and will cyclically toggle between LOW and HIGH (upon subsequent invocations of this command).*

*Since this parameter affects the SPI Bus Mode, the following parameters will be displayed to the user:*

**SPI Bus Mode**  
**SCK Idle State** (parameter that this command toggles)  
**Tx on SCK**

**NOTE:** This is a sticky setting that will be remembered and loaded upon Master SPI Bus Engine power-up.

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
SET_DATA_OUTPUT_ON_SCK_EDGE	^

**Command function description:**

Allows the ability to select on which SCK edge that the master will transmit data to the slave (referred to as the clock edge **CKE** bit in Microchip's documentation, or **CPHA** in Motorola's SPI documentation). The master SPI data can be transmitted to the slave on either the **active-to-idle** edge of SCK, or on the **idle-to-active** edge of SCK.

*User types the following character on the keyboard: ^*

*The 'Tx on SCK' mode (transmit data on SCK edge state) will be displayed to the user and will cyclically toggle between ACTIVE->IDLE and IDLE->ACTIVE (upon subsequent invocations of this command).*

*Since this parameter affects the SPI Bus Mode, the following parameters will be displayed to the user:*

**SPI Bus Mode**  
**SCK Idle State**  
**Tx on SCK** (parameter that this command toggles)

**NOTE:** This is a sticky setting that will be remembered and loaded upon Master SPI Bus Engine power-up.

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
SET_DATA_INPUT_SAMPLE_PHASE	"

**Command function description:**

Allows the ability to set at which point in the data output phase that the master will sample the input data (referred to as the sample bit **SMP** in Microchip's documentation). The master can sample the input data at either the middle of the data output phase or at the end of the data output phase.

*User types the following character on the keyboard: "*

*The 'Input Sample Phase' mode will be displayed to the user and will cyclically toggle between MID and END (upon subsequent invocations of this command).*

**NOTE:** This is a sticky setting that will be remembered and loaded upon Master SPI Bus Engine power-up.



Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
<b>CLEAR_SCREEN</b>	<b>*</b>

**Command function description:**

Clears the serial terminal's screen.

(On the Cybiko, it may take up to 2 seconds for the screen display to refresh after receiving this command.)

**User types the following character on the keyboard: \***

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
<b>SOFTWARE_RESET</b>	<b>!</b>

**Command function description:**

Generates a software reset on the target microcontroller that hosts the Master SPI Bus Engine.

The software reset will reset and re-initialize the Master SPI Bus Engine, and display its current settings.

(On the Cybiko, it may take up to 4 seconds for the screen display to refresh after receiving this command.)

**User types the following character on the keyboard: !**

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
<b>TEST_MODE_1</b>	<b>{</b>

**Command function description:**

Initiates a master SPI bus write transaction sequence in an infinite loop. Useful for troubleshooting and observing SPI bus signals (SCK and SDO signal relationships under various bus modes) with an oscilloscope. The sequence that is performed in an infinite loop is as follows:

Assert chip select #1 (/CS #1)  
Send byte 0xAA (alternating ones and zeros: 10101010)  
Unassert chip select #1 (/CS #1)  
Time delay

**User types the following character on the keyboard: {**

**WARNING: This command will execute an SPI test transaction sequence in an infinite loop that will preclude any subsequent communication attempts. The only way to break out of this test mode is to cycle power to the Master SPI Bus Engine.**

Master SPI Bus Engine ASCII COMMAND FUNCTION	ASCII COMMAND
<b>TEST_MODE_2</b>	<b>}</b>

**Command function description:**

Initiates a master SPI bus write transaction sequence in an infinite loop. Useful for troubleshooting and observing SPI bus signals (SCK and SDO signal relationships under various bus modes) with an oscilloscope. The sequence that is performed in an infinite loop is as follows:

Assert chip select #1 (/CS #1)  
Send byte 0x55 (alternating zeros and ones: 01010101)  
Unassert chip select #1 (/CS #1)  
Time delay

**User types the following character on the keyboard: }**

**WARNING: This command will execute an SPI test transaction sequence in an infinite loop that will preclude any subsequent communication attempts. The only way to break out of this test mode is to cycle power to the Master SPI Bus Engine.**

Master SPI Bus Engine ASCII COMMAND FUNCTION		ASCII COMMAND
HELP		?

**Command function description:**

Displays the Master SPI Bus Engine's current configuration settings and the ASCII command list.  
(On the Cybiko, it may take up to 4 seconds for the screen display to refresh after receiving this command.)

**User types the following character on the keyboard: ?**

**NOTE:** Due to the PICC Lite's limited program memory space for the PIC16F877/PIC16F877A microcontroller, there simply was no room left to display detailed descriptive information on the ASCII commands or any additional help information.

## 6.0 LOW-LEVEL BINARY COMMANDS

Master SPI Bus Engine BINARY COMMAND FUNCTION LIST	BINARY COMMAND (hex)
UNASSERT_ALL_CHIP_SELECTS	0x80
ASSERT_CHIP_SELECT_1	0x90
ASSERT_CHIP_SELECT_2	0x91
ASSERT_CHIP_SELECT_3	0x92
ASSERT_CHIP_SELECT_4	0x93
SEND_USER_DATA_AND_RECEIVE_BYTE	0xA0
SEND_DUMMY_DATA_AND_RECEIVE_BYTE	0xB0
TEST_MODE_1	0xC0
TEST_MODE_2	0xC1
SOFTWARE_RESET	0xF1

These binary commands (non-ASCII) allow the user to create high-level SPI device drivers and software applications via software control of the host's serial port. The MSBE ActiveX DLL (MSBE.dll) wraps these functions in order to simplify creating PC software-based SPI bus communication applications. The MSBE ActiveX DLL provides the following device drivers in v1.0.0:

**Microchip Technology SPI serial EEPROMs:**

25XX010A  
25XX020A  
25XX040/25XX040A  
25XX080/25XX080A/25XX080B  
25XX160/25XX160A/25XX160B  
25XX320/25XX320A  
25XX640/25XX640A  
25XX128  
25XX256

**Maxim MAX7219 8-digit LED display driver**

**Linear Technology LTC1661 10-bit, dual-channel DAC**

**Linear Technology LTC1665 8-bit, eight-channel DAC**

**IMPORTANT NOTES:** *The Master SPI Bus Engine does not allow queuing of commands. After an initial command and any applicable parameter has been sent to the Master SPI Bus Engine, subsequent commands must not be sent until the expected response code and any associated data (if any) is received from Master SPI Bus Engine, or until a communication-timeout period of 250 mS has expired.*

*For example, once the binary command and the data byte are sent (back-to-back) for the SEND\_USER\_DATA\_AND\_RECEIVE\_BYTE command, the host must wait for the response PASS/FAIL code and for the byte received before issuing any subsequent binary command. If there is no response after 250 mS, a communication timeout has occurred and issuing another command is permissible.*

*Before using the Master SPI Bus Engine to talk to any SPI device, it must be configured properly for the target SPI device (in terms of SPI bus mode, clock frequency, input sample phase, etc.) via the ASCII commands, otherwise communication with the target SPI device may not yield the results expected. Please refer to the SPI device manufacturer's data sheet for details. Also ensure that the chip select line connected to the target SPI device is controlled properly as well.*

Master SPI Bus Engine BINARY COMMAND FUNCTION	BINARY COMMAND
UNASSERT_ALL_CHIP_SELECTS	0x80

**Command function description:**

Unasserts all chip select output lines (/CS #1, /CS #2, /CS #3, and /CS #4).

**Host sends:**

Byte #1: 0x80 (COMMAND FUNCTION)

**SPI Bus Engine response:**

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: **PASS** – All chip select lines were unasserted successfully.

Any response in the range of 0x01 – 0xFF: **FAIL** – Attempt to unassert all chip select lines failed.

Master SPI Bus Engine BINARY COMMAND FUNCTION	BINARY COMMAND
ASSERT_CHIP_SELECT_1	0x90

**Command function description:**

Unasserts all chip select output lines and then asserts chip select #1 (/CS #1).

**Host sends:**

Byte #1: 0x90 (COMMAND FUNCTION)

**SPI Bus Engine response:**

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: **PASS** – Chip select #1 was asserted successfully.

Any response in the range of 0x01 – 0xFF: **FAIL** – Attempt to assert chip select #1 failed.

Master SPI Bus Engine BINARY COMMAND FUNCTION	BINARY COMMAND
ASSERT_CHIP_SELECT_2	0x91

**Command function description:**

Unasserts all chip select output lines and then asserts chip select #2 (/CS #2).

**Host sends:**

Byte #1: 0x91 (COMMAND FUNCTION)

**SPI Bus Engine response:**

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: **PASS** – Chip select #2 was asserted successfully.

Any response in the range of 0x01 – 0xFF: **FAIL** – Attempt to assert chip select #2 failed.

Master SPI Bus Engine BINARY COMMAND FUNCTION	BINARY COMMAND
<b>ASSERT_CHIP_SELECT_3</b>	<b>0x92</b>

**Command function description:**

Unasserts all chip select output lines and then asserts chip select #3 (/CS #3).

**Host sends:**

Byte #1: 0x92 (COMMAND FUNCTION)

**SPI Bus Engine response:**

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: **PASS** – Chip select #3 was asserted successfully.

Any response in the range of 0x01 – 0xFF: **FAIL** – Attempt to assert chip select #3 failed.

Master SPI Bus Engine BINARY COMMAND FUNCTION	BINARY COMMAND
<b>ASSERT_CHIP_SELECT_4</b>	<b>0x93</b>

**Command function description:**

Unasserts all chip select output lines and then asserts chip select #4 (/CS #4).

**Host sends:**

Byte #1: 0x93 (COMMAND FUNCTION)

**SPI Bus Engine response:**

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: **PASS** – Chip select #4 was asserted successfully.

Any response in the range of 0x01 – 0xFF: **FAIL** – Attempt to assert chip select #4 failed.

Master SPI Bus Engine BINARY COMMAND FUNCTION	BINARY COMMAND
<b>SEND_USER_DATA_AND_RECEIVE_BYTE</b>	<b>0xA0</b>

**Command function description:**

Sends a user-defined data byte over the SPI bus, and reads/returns the data byte received from the slave device.

**Host sends:**

Byte #1: 0xA0 (COMMAND FUNCTION)

Byte #2: 0xXX (where XX is the byte to send)

**SPI Bus Engine response:**

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: **PASS** – Byte was successfully sent to the SPI slave device.

0x01: **FAIL** – Attempt to send byte to the SPI slave device failed.

Byte #2: Byte read from slave device (received byte is valid only if previous command function response code is PASS)

Master SPI Bus Engine BINARY COMMAND FUNCTION	BINARY COMMAND
SEND_DUMMY_DATA_AND_RECEIVE_BYTE	0xB0

**Command function description:**

Sends a 'dummy' data byte over the SPI bus, and reads/returns the data byte received from the slave device. This command is used when you are interested in receiving data from the SPI slave device, but the byte sent is a 'DO NOT CARE' or 'dummy' byte.

**Host sends:**

Byte #1: 0xB0 (COMMAND FUNCTION)

**SPI Bus Engine response:**

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: **PASS** – 'Dummy' byte was successfully sent to the SPI slave device.

0x01: **FAIL** – Attempt to send 'dummy' byte to the SPI slave device failed.

Byte #2: Byte read from slave device (received byte is valid only if previous command function response code is PASS)

Master SPI Bus Engine BINARY COMMAND FUNCTION	BINARY COMMAND
TEST_MODE_1	0xC0

**Command function description:**

Initiates a master SPI bus write transaction sequence in an infinite loop. Useful for troubleshooting and observing SPI bus signals (SCK and SDO signal relationships under various bus modes) with an oscilloscope. The sequence that is performed in an infinite loop is as follows:

Assert chip select #1 (/CS #1)  
Send byte 0xAA (alternating ones and zeros: 10101010)  
Unassert chip select #1 (/CS #1)  
Time delay

**WARNING: This command will execute an SPI test transaction sequence in an infinite loop that will preclude any subsequent communication attempts. The only way to break out of this test mode is to cycle power to the Master SPI Bus Engine.**

**Host sends:**

Byte #1: 0xC0 (COMMAND FUNCTION)

**SPI Bus Engine response:**

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE

0x00: **PASS** – Initiation of test mode was successful.

Any response in the range of 0x01 – 0xFF: **FAIL** – Attempt to initiate test mode failed.

Master SPI Bus Engine BINARY COMMAND FUNCTION	BINARY COMMAND
<b>TEST_MODE_2</b>	<b>0xC1</b>

**Command function description:**

Initiates a master SPI bus write transaction sequence in an infinite loop. Useful for troubleshooting and observing SPI bus signals (SCK and SDO signal relationships under various bus modes) with an oscilloscope. The sequence that is performed in an infinite loop is as follows:

Assert chip select #1 (/CS #1)  
Send byte 0x55 (alternating zeros and ones: 01010101)  
Unassert chip select #1 (/CS #1)  
Time delay

**WARNING: This command will execute an SPI test transaction sequence in an infinite loop that will preclude any subsequent communication attempts. The only way to break out of this test mode is to cycle power to the Master SPI Bus Engine.**

**Host sends:**

Byte #1: 0xC1 (COMMAND FUNCTION)

**SPI Bus Engine response:**

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE  
0x00: **PASS** – Initiation of test mode was successful.  
Any response in the range of 0x01 – 0xFF: **FAIL** – Attempt to initiate test mode failed.

Master SPI Bus Engine BINARY COMMAND FUNCTION	BINARY COMMAND
<b>SOFTWARE_RESET</b>	<b>0xF1</b>

**Command function description:**

Generates a software-reset on the target microcontroller that hosts the Master SPI Bus Engine.  
The software reset will reset and re-initialize the Master SPI Bus Engine, and the ASCII power-up screen will be dumped to the serial port.

**Host sends:**

Byte #1: 0xF1 (COMMAND FUNCTION)

**SPI Bus Engine response:**

Byte #1: COMMAND FUNCTION PASS/FAIL RESPONSE  
0x00: **PASS** – Software reset was successful.  
Any response in the range of 0x01 – 0xFF: **FAIL** – Software reset failed.

**NOTE:** After a PASS code is received, the microcontroller that hosts the Master SPI Bus Engine will output its ASCII power-up screen to the serial port. Therefore, any PC host software should take this into account and perform the following steps with regard to this command:

- 1) **Issue SOFTWARE\_RESET command**
- 2) **Wait for the PASS code from the Master SPI Bus Engine**
- 3) **Delay 500 milliseconds**
- 4) **Clear the PC's UART receive buffer**

## 7.0 ActiveX DLL: MSBE.dll

The Master SPI Bus Engine can be controlled with an ActiveX DLL that wraps the functionality of the binary command set as well as provides specific device drivers for various SPI Bus devices. The MSBE ActiveX DLL (MSBE.dll) was created in Visual Basic 6.0. Users may explore it within the Visual Basic Object Browser and must set a reference to it (Project menu > References) before using it in their client programs.

**IMPORTANT NOTE:** *Before using the Master SPI Bus Engine to talk to any SPI device, it must be configured properly for the target SPI device (in terms of SPI bus mode, clock frequency, input sample phase, etc.) via the ASCII commands, otherwise communication with the target SPI device may not yield the results expected. Please refer to the SPI device manufacturer's data sheet for details. Also ensure that the chip select line connected to the target SPI device is controlled properly as well.*

### 7.1 AVAILABLE CLASSES IN MSBE.DLL (DLL version v1.0.0)

#### CORE CLASSES:

clsMaster_SPI_Bus_Engine_v100	(wraps the functionality of the low-level Master SPI Bus Engine binary command set)
clsSerialPort_v100	(provides serial port control on the host PC)
clsApp_v100	(provides application information)

#### DEVICE DRIVER CLASSES:

##### SPI SERIAL EEPROMs:

cls25XX010A\_v100  
cls25XX020A\_v100  
cls25XX040\_25XX040A\_v100  
cls25XX080\_25XX080A\_v100  
cls25XX080B\_v100  
cls25XX160\_25XX160A\_v100  
cls25XX160B\_v100  
cls25XX320\_25XX320A\_v100  
cls25XX640\_25XX640A\_v100  
cls25XX128\_v100  
cls25XX256\_v100

##### DIGITAL-TO-ANALOG CONVERTERS:

clsLTC1661\_v100  
clsLTC1665\_v100

##### LED DISPLAY DRIVERS:

clsMAX7219\_v100

### 7.2 IN-PROCESS COMPONENT BASE ADDRESS

The base address for the MSBE.dll is: 0x1100000



### 7.3 DLL EXCEPTION ERROR CODES (DLL version v1.0.0)

I would be remiss in my duties if I did not publish the unique exception error codes that I have created for this DLL. Don't forget that your client code must have proper error handling to accommodate these exceptions in order to avoid run-time errors that crash the client application.

#### MASTER SPI BUS ENGINE CORE ERROR CODES

**COMMUNICATION\_TIMEOUT** = 0x80040300  
(Serial communication timeout.)

**UNASSERT\_ALL\_CHIP\_SELECTS\_FAILURE** = 0x80040301  
(Attempt to unassert all chip select lines failed.)

**ERROR\_MESSAGE\_ASSERT\_CHIP\_SELECT\_1\_FAILURE** = 0x80040302  
(Attempt to unassert chip select #1 failed.)

**ERROR\_MESSAGE\_ASSERT\_CHIP\_SELECT\_2\_FAILURE** = 0x80040303  
(Attempt to unassert chip select #2 failed.)

**ERROR\_MESSAGE\_ASSERT\_CHIP\_SELECT\_3\_FAILURE** = 0x80040304  
(Attempt to unassert chip select #3 failed.)

**ERROR\_MESSAGE\_ASSERT\_CHIP\_SELECT\_4\_FAILURE** = 0x80040305  
(Attempt to unassert chip select #4 failed.)

**SEND\_BYTE\_FAILURE** = 0x80040306  
(Attempt to send byte to slave failed.)

**TEST\_MODE\_1\_FAILURE** = 0x80040307  
(Attempt to enter TEST MODE 1 failed.)

**TEST\_MODE\_2\_FAILURE** = 0x80040308  
(Attempt to enter TEST MODE 2 failed.)

**SOFTWARE\_RESET\_FAILURE** = 0x80040309  
(Attempt to software reset the Master SPI Bus Engine failed.)

**ILLEGAL\_CHIP\_SELECT\_NUMBER** = 0x8004030A  
(Attempt to set the chip select property failed due to an illegal chip select number.  
Legal chip select numbers for this property are: 1, 2, 3, or 4.)

#### DEVICE DRIVER ERROR CODES: MAXIM MAX7219 8-DIGIT LED DISPLAY DRIVER

**ILLEGAL\_DECODE\_MODE\_VALUE** = 0x80040320  
(Attempt to set the MAX7219 decode mode failed due to an illegal decode mode value.)

**ILLEGAL\_DIGIT\_SCAN\_LIMIT\_VALUE** = 0x80040321  
(Attempt to set the MAX7219 digit scan limit failed due to an illegal digit scan limit value.)

**ILLEGAL\_DIGIT\_REGISTER\_ADDRESS** = 0x80040322  
(Attempt to write to a digit register failed due to an illegal digit register address.  
Legal digit register addresses are: 1, 2, 3, 4, 5, 6, 7, and 8.)

#### DEVICE DRIVER ERROR CODES: SPI SERIAL EEPROMs

**SERIAL\_EEPROM\_WRITE\_TIMEOUT** = 0x80040340  
(Attempt to initiate an EEPROM write operation failed due to a write timeout condition.)

**ILLEGAL\_EEPROM\_ADDRESS** = 0x80040341  
(The address argument is an illegal address for the target EEPROM device.)

**ILLEGAL\_PAGE\_ADDRESS\_ARGUMENT** = 0x80040342  
(The address argument is an illegal page address for the target EEPROM device.)

**SERIAL\_EEPROM\_SINGLE\_BYTE\_WRITE\_VERIFICATION\_ERROR** = 0x80040343  
(The single byte write operation failed verification.)

**SERIAL\_EEPROM\_PAGE\_WRITE\_VERIFICATION\_ERROR** = 0x80040344  
(The page write operation failed verification.)

**DEVICE DRIVER ERROR CODES: LINEAR TECHNOLOGY LTC1661 10-BIT, DUAL-CHANNEL DAC**

**LTC1661\_ILLEGAL\_CONTROL\_CODE** = 0x80040360

(Attempt to write to the DAC failed due to an illegal control code.

Legal control codes are: 0 through 15 decimal.)

**LTC1661\_ILLEGAL\_DATA\_CODE** = 0x80040361

(Attempt to write to the DAC failed due to an illegal data value.

Legal data values are: 0 through 1023 decimal.)

**DEVICE DRIVER ERROR CODES: LINEAR TECHNOLOGY LTC1665 8-BIT, EIGHT-CHANNEL DAC**

**LTC1665\_ILLEGAL\_CONTROL\_CODE** = 0x80040380

(Attempt to write to the DAC failed due to an illegal control code.

Legal control codes are: 0 through 15 decimal.)

## 7.4 USING THE MSBE DLL IN VISUAL BASIC 6

Before using the MSBE DLL in Visual Basic, you must first set a reference to it by Selecting the **Project** menu, then the **References...** menu item. Then you just browse to where the MSBE.dll file is located, click **Open** in the **Add Reference** dialog box, then click **OK** in the main **References** dialog box. Now you are ready to use the DLL. The following are some snippets to get you started. It's not easy to see because these are just example snippets, but typically you will only declare and create the objects only once, say within a form's load event, and clear objects once during a form unload event.

### VISUAL BASIC CODE SNIPPET: WORKING WITH THE LOW-LEVEL BINARY COMMAND SET TO READ FROM A SPI EEPROM

```
On Error GoTo ErrorHandler

Dim bytDataByte As Byte

' Declare an object variable reference to the serial port class
Dim SerialPort As clsSerialPort_v100

' Create instance of object
Set SerialPort = New clsSerialPort_v100

' Declare an object variable reference to the Master SPI Bus Engine class
Dim SPI_Engine As clsMaster_SPI_Bus_Engine_v100

' Create instance of object
Set SPI_Engine = New clsMaster_SPI_Bus_Engine_v100

' Easy way to open serial port COM 2
Call SerialPort.CommunicationPort_OpenByPortNumber(2)

' Alternate method for opening serial port COM 2:
' SerialPort.CommunicationPortNumber = 2
' Call SerialPort.CommunicationPort_OpenExistingPort

' Assert the /CS #1 line
Call SPI_Engine.ChipSelect_Assert_CS_1

' Send READ command
Call SPI_Engine.SendMasterUserByteAndReceiveSlaveByte(&H3)

' Send ADDRESS high byte
Call SPI_Engine.SendMasterUserByteAndReceiveSlaveByte(0)

' Send ADDRESS low byte
Call SPI_Engine.SendMasterUserByteAndReceiveSlaveByte(0)

' Read byte at previously specified address
bytDataByte = SPI_Engine.SendMasterDummyByteAndReceiveSlaveByte

' Un-assert all chip select lines
Call SPI_Engine.ChipSelect_UnassertAll

' Close the active serial port
Call SerialPort.CommunicationPort_Close

' Clear object variable references
Set SerialPort = Nothing
Set SPI_Engine = Nothing

Exit Sub

ErrorHandler:

MsgBox Err.Number & vbCrLf & _
    Err.Source & vbCrLf & _
    Err.Description & vbCrLf & _
    Err.LastDllError
```

## VISUAL BASIC CODE SNIPPET: WORKING WITH A SPI EEPROM DEVICE DRIVER – MUCH EASIER!

```
On Error GoTo ErrorHandler

Dim bytDataByte As Byte

' Declare an object variable reference to the serial port class
Dim SerialPort As clsSerialPort_v100

' Create instance of object
Set SerialPort = New clsSerialPort_v100

' Declare an object variable reference to the 25XX640 SPI serial EEPROM class
Dim EEPROM As cls25XX640_25XX640A_v100

' Create instance of object
Set EEPROM = New cls25XX640_25XX640A_v100

' Easy way to open serial port COM 2
Call SerialPort.CommunicationPort_OpenByPortNumber(2)

' Alternate method for opening serial port COM 2:
' SerialPort.CommunicationPortNumber = 2
' Call SerialPort.CommunicationPort_OpenExistingPort

' Set the /CS # that will be used with the EEPROM
EEPROM.ChipSelectNumber = 1

' Read byte at address 0x0000
bytDataByte = EEPROM.SingleByte_Read(&H0)

' Close the active serial port
Call SerialPort.CommunicationPort_Close

'Clear object variable references
Set SerialPort = Nothing
Set EEPROM = Nothing

Exit Sub

ErrorHandler:

MsgBox Err.Number & vbCrLf & _
    Err.Source & vbCrLf & _
    Err.Description & vbCrLf & _
    Err.LastDllError
```

## VISUAL BASIC CODE SNIPPET: WORKING WITH THE LTC1665 DEVICE DRIVER (no LTC1661 code snippet will be shown, but it is very similar to the LTC1665)

```
On Error GoTo ErrorHandler

' Declare an object variable reference to the serial port class
Dim SerialPort As clsSerialPort_v100

' Create instance of object
Set SerialPort = New clsSerialPort_v100

' Declare an object variable reference to the LTC1665 digital-to-analog converter class
Dim LTC1665 As clsLTC1665_v100

' Create instance of object
Set LTC1665 = New clsLTC1665_v100

' Easy way to open serial port COM 2
Call SerialPort.CommunicationPort_OpenByPortNumber(2)

' Alternate method for opening serial port COM 2:
' SerialPort.CommunicationPortNumber = 2
' Call SerialPort.CommunicationPort_OpenExistingPort

' Set the /CS # that will be used with the LTC1665 device
LTC1665.ChipSelectNumber = 1

' Set all DAC channels to approximately half of its full range
Call LTC1665.DAC_A_B_C_D_E_F_G_H_WriteWithSameCode(128)

MsgBox "Measure all DAC channels with a digital multimeter."

' You can also access the 8 DAC channels individually like the following:
Call LTC1665.DAC_A_Write(0)
Call LTC1665.DAC_B_Write(32)
Call LTC1665.DAC_C_Write(64)
Call LTC1665.DAC_D_Write(96)
Call LTC1665.DAC_E_Write(128)
Call LTC1665.DAC_F_Write(160)
Call LTC1665.DAC_G_Write(192)
Call LTC1665.DAC_H_Write(224)

MsgBox "Measure all DAC channels with a digital multimeter."

' Close the active serial port
Call SerialPort.CommunicationPort_Close

' Clear object variable references
Set SerialPort = Nothing
Set LTC1665 = Nothing

Exit Sub

ErrorHandler:

MsgBox Err.Number & vbCrLf & _
    Err.Source & vbCrLf & _
    Err.Description & vbCrLf & _
    Err.LastDllError
```

## VISUAL BASIC CODE SNIPPET: WORKING WITH THE MAX7219 DEVICE DRIVER

```
On Error GoTo ErrorHandler

' Declare an object variable reference to the serial port class
Dim SerialPort As clsSerialPort_v100

' Create instance of object
Set SerialPort = New clsSerialPort_v100

' Declare an object variable reference to the MAX7219 LED display driver class
Dim MAX7219 As clsMAX7219_v100

' Create instance of object
Set MAX7219 = New clsMAX7219_v100

' Easy way to open serial port COM 2
Call SerialPort.CommunicationPort_OpenByPortNumber(2)

' Alternate method for opening serial port COM 2:
SerialPort.CommunicationPortNumber = 2
' Call SerialPort.CommunicationPort_OpenExistingPort

' *** SEND '65' to the lower 2 seven segment displays

' Set the /CS # that will be used with the MAX7219
MAX7219.ChipSelectNumber = 1

' Put the MAX7219 into shutdown mode
Call MAX7219.ShutdownMode_Enter

' Use code B decoding for all digits
Call MAX7219.SetDecodeMode(CODE_B_DECODE_FOR_DIGITS_7_THROUGH_0)

' Display only 2 digit
Call MAX7219.SetDigitScanLimit(DISPLAY_DIGITS_0_1_ONLY)

' Maximum brightness
Call MAX7219.SetIntensity_X(DUTY_CYCLE_31_32_MAXIMUM)

' Send '6' to the tens column digit
Call MAX7219.SetDigit_1(6)

' Send '5' to the ones column digit
Call MAX7219.SetDigit_0(5)

' Put the MAX7219 into normal mode
Call MAX7219.ShutdownMode_Exit

' Close the active serial port
Call SerialPort.CommunicationPort_Close

' Clear object variable references
Set SerialPort = Nothing
Set MAX7219 = Nothing

Exit Sub

ErrorHandler:

MsgBox Err.Number & vbCrLf & _
    Err.Source & vbCrLf & _
    Err.Description & vbCrLf & _
    Err.LastDllError
```

## APPENDIX A – TROUBLESHOOTING AND FAQ (FREQUENTLY ASKED QUESTIONS)

**Question:** *I can't communicate with my target SPI device. What do you think I'm doing wrong?*

**Answer:**

First and foremost, please ensure you are using v1.1 of the Master SPI Bus Engine HEX file. The original v1.0 version of the Master SPI Bus Engine HEX file only worked correctly with very old versions of PIC16F877 microcontrollers due to MSSP silicon errata. Please read the 'readme.txt' file that accompanies this PDF file for more details on this issue.

There could be a lot of other reasons as well, but if you are certain your wiring is correct it could be that the Master SPI Bus Engine is improperly configured for the target SPI device. Please ensure that you carefully read the manufacturer's data sheet with regard to the target SPI device. The Master SPI Bus Engine must be configured for the proper SPI Bus Mode (0, 1, 2, or 3), SCK Frequency, and Input Sample Phase (if applicable) in accordance with the target SPI device – otherwise data that is sent/received during SPI transactions will not be correct or will be marginal.

**Question:** *I see 'TEST MODE' displayed on my screen and any ASCII commands I try to issue do not work. Any ideas?*

**Answer:**

You have invoked (possibly accidentally) one of the Master SPI Bus Engine's test modes. Once a test mode has been invoked, the only way to exit the test mode is to cycle the Master SPI Bus Engine's power. For more details on the test modes and their behavior, please refer to the **TEST\_MODE\_1** and **TEST\_MODE\_2** ASCII command description in this document.

**Question:** *Will you release versions that will accept other crystals besides the 18.432 MHz crystal?*

**Answer:**

I'm sorry, but I do not plan on supporting any other crystals besides the 18.432 MHz crystal.

The 18.432 MHz crystal was selected for several reasons:

- A) Keeping a single configuration so that the end-user has a better chance of getting things working.
- B) Simplicity of configuration management, documentation and testing.
- C) I wanted a crystal as close to 20 MHz as possible for maximum horsepower (the maximum  $F_{OSC}$  on the PIC16F877/877A is 20 MHz).
- D) The 18.432 MHz crystal provides an exact 115,200 bps baud rate (for all practical purposes, of course).

**Question:** *Why didn't you choose a newer PIC instead of the PIC16F877/PIC16F877A for this project?*

**Answer:**

Because the HI-TECH PICC Lite free C compiler supports the PIC16F877 and PIC16F877A PICmicros. I wanted to use a PICmicro from the PIC18 architecture, but currently the HI-TECH PICC Lite C compiler does not support any PIC18 devices. I also wanted to use a C compiler that was completely free and one that did not have any 30 or 60-day honor system usage agreement requirement.

**Question:** *Can I use a PIC16F876 or PIC16F876A PICmicro to host the Master SPI Bus Engine?*

**Answer:**

Yes, the PIC16F876 or PIC16F876A will be able to host the Master SPI Bus Engine and you can use the PIC16F877/877A hex files to program them, but I would recommend using the PIC16F877/877A PICmicros – especially the PIC16F877A.

**Question:** *I'm using the Master SPI Bus Engine with the Cybiko classic and the Master SPI Bus Engine does not appear to work anymore once the Cybiko comes out of suspend mode – but this does not happen all the time. Any ideas?*

**Answer:**

It might be a good idea to disable the power saver mode (suspend mode) on the Cybiko when you are using VTterm. It has been noticed in some cases that after a Cybiko goes into sleep mode and is later awoken from sleep mode, VTterm does not communicate with the Master SPI Bus Engine. While it may appear that the Master SPI Bus Engine is locked up, it is not. What fixes this problem in the short-term is shutting down VTterm and re-starting it. For a long-term fix, just remember to disable the Cybiko's power saver mode: <Fn><F6> (Disables automatic switch to Suspend Mode). Disabling power saver mode can be done at any time (including while VTterm is running).

## APPENDIX B – DOCUMENT REVISION HISTORY

I hope the Master SPI Bus Engine project is simple enough for people to build and use from the information contained in this document. However, if you have any questions or need to contact me, you can usually find me on the PICList mailing list ([www.PICList.com](http://www.PICList.com)) or the Microchip Forums (<http://forum.microchip.com>, user name: Ken\_Pergola). You can also e-mail me privately at: [no\\_spam@localnet.com](mailto:no_spam@localnet.com) -- that is not a typo -- that's my real e-mail address -- at least for now (don't forget the underscore '\_' character between the 'no' and 'spam'). If you find any mistakes or find parts of this document that are confusing, please let me know. My wish is that you find the Master SPI Bus Engine as useful as I have found it to be (especially when used in conjunction with the Cybiko classic). Get on the SPI bus and have fun!

### **v1.0.1 – 06-11-2004 – Kenneth Michael Pergola (KMP)**

Added a note in the FAQ section about a bug in the original v1.0 version of the Master SPI Engine HEX file firmware. More details are available in the 'readme.txt' file that accompanies this PDF file in the ZIP file release package. Please ensure you use the v1.1 version of the Master SPI Bus Engine HEX file. Special thanks goes to András Pilinyi, Project Manager of PowerStar Kft., of Budapest, Hungary for discovering and reporting this problem.

Added a note about not leaving the Master SPI Bus Engine's SDI pin floating.

Minor grammatical/spelling edits.

### **v1.0.0 – 04-03-2004 – Kenneth Michael Pergola (KMP)**

Initial preliminary release of the Master SPI Bus Engine document.