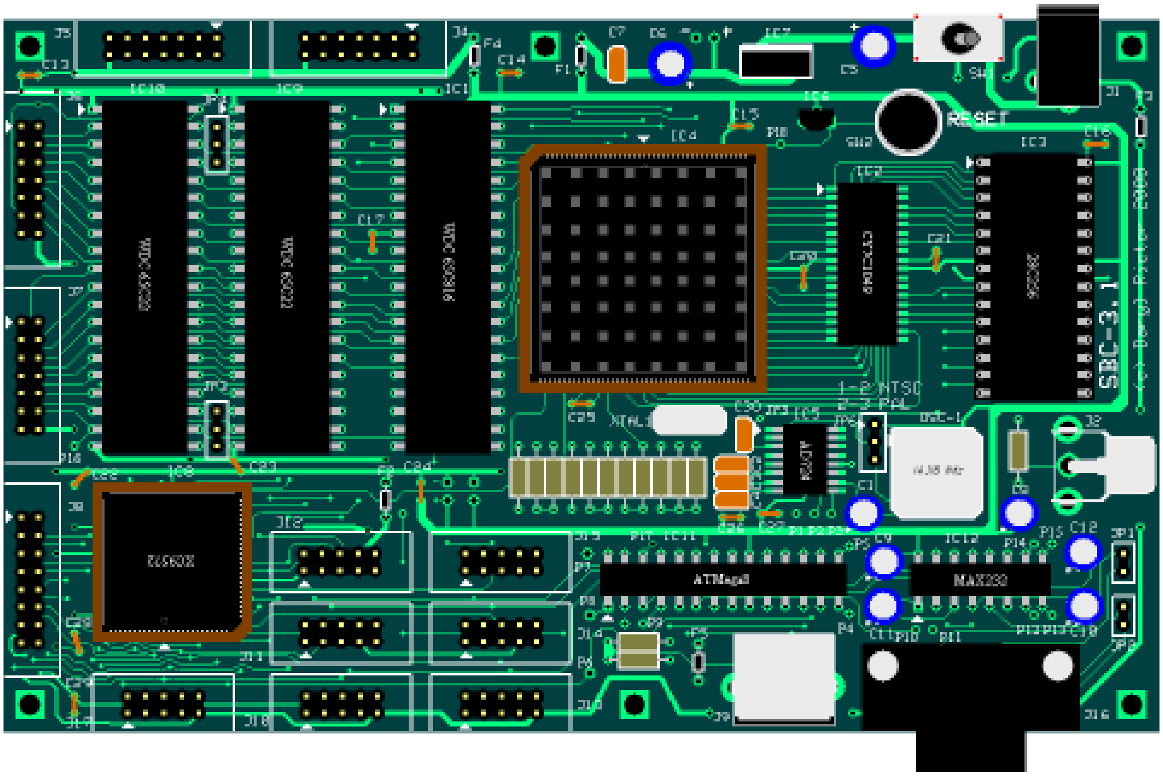
SBC-3 User's Guide



Written by

Daryl Rictor

V1.0
Nov 9, 2008

CONTENTS

DISCLAIMER AND COPYRIGHT NOTICE

This computer design, firmware, software, and its associated documentation are provided for your personal use only and appear here exclusively by permission of the copyright holder. No commercial reproduction is authorized without prior written consent of the copyright holder. Please contact the copyright holder before re-distributing, re-publishing or disseminating this copyrighted work. This work is not GPL or in the public domain. Please respect the author's copyright.

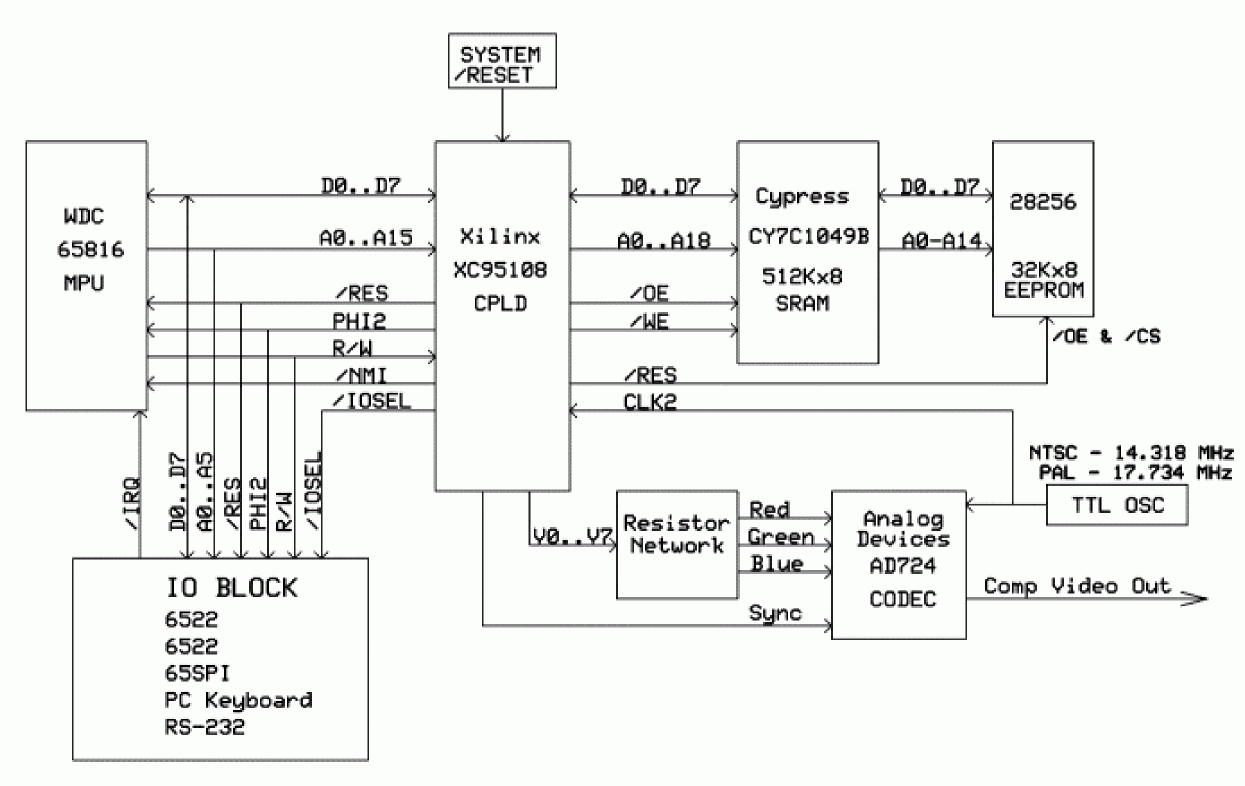
No warranty, either expressed or implied, is given. I assume no liability for its use in any project or device. Use this work at your own risk.

Your use of this computer design and the associated programs indicates your acceptance of all license terms.

OVERVIEW

SBC-3 is my latest design. It is an expansion on the SBC-2 board and incorporates two Complex Programmable Logic Devices (CPLDs). The CPLDs replace dozens of discrete logic chips and increase performance and features.

This is the System Block Diagram:



CPU

The microprocessor is the 65C02's big brother, the WDC 65C816. This processor is a 16-bit processor and can directly address 16MB of memory space. It is compatible with 65C02 code and includes more powerful 16-bit instructions as well. While a 65C02 can address more than 65k using bank switching techniques, the 65C816 can address SBC-3’s entire 512k directly. SBC-3 uses the 40-pin DIP package.

RAM

The RAM is a Cypress CY7C1049D 512k x 8 static RAM with an access speed if 15ns. It is packaged in a 36-pin SOP. It is used for the microprocessor program and data storage, and for the video display buffers. The Video system and the microprocessor access the RAM using an alternating access scheme that allows both to have full access without causing either to wait on the other. This is done using the PHI2 clock signal. When the clock is low, the video system accesses RAM, and when it’s high, the 65C816 can access RAM.

The RAM is located in the first 8 blocks of the 16MB space, from $00000 to $7FFFF. There are not any provisions to expand this on the board. This was chosen to prevent bus loading from interfering with critical system timing. Additional temporary storage could be implemented through the SPI interface.

This is the System memory map:

$00000-$0000F - VIA 1 I/O
$00010-$0001F - VIA 2 I/O
$00020-$00027 - SPI I/O
$00028-$0002F – unused I/O Space
$00030 - Video Display Register
$00031-$07FFF - System RAM
$08000-$0FFFF - System RAM loaded with contents of the EEPROM during RESET. Software-selectable Write Protection through Video Data Register.
$10000-$1FFFF - System RAM - Default Video display block
$20000-$7FFFF - System RAM

EEPROM

The EEPROM is a standard 28C256 32k x 8 EEPROM in a 28-pin DIP package. It contains the boot code and System Monitor which is copied into RAM during the RESET cycle of the XC95108. After the reset is released, the EEPROM is deselected and stays in low-power standby until another reset. Users can create their own boot code to suit their individual needs. The EEPROM is not writable in-system, therefore must be programmed externally. **It is highly recommended to place this IC in a socket to allow changes easily.**

SYSTEM RESET

The Dallas Semiconductor DS-1813 econo-reset device handles the system-reset functions. This 3-pin package provides the power-on reset and also de-bounces the reset button. No external components are required. Pressing the Reset switch will re-initialize the system also. The reset signal goes to the XC95108 CPLD, which sets the 65C816's reset low and keeps it low, inhibiting it. The XC95108 copies the contents of the 32k EEPROM into RAM, from address $08000 to $0FFFF, initializes the video system, and releases the reset to the 65C816. The 65C816 will pull a RESET vector from $0FFFC and $0FFFD and start executing instructions from that address. This allows the system to boot up into the System Monitor. The upper 32k of RAM in Block 0 ($08000-$0FFFF) is defined as read-only after RESET to prevent changes to the boot code. Under software control, it can be changed to read-write access, allowing the user to change the RESET and Interrupt vectors as needed. See the Video Display Register description below for more details.

POWER SUPPLY

The power supply is a standard 9vdc wall transformer capable of delivering up to 1.25A. This feeds the onboard 5vdc voltage regulator. This is an L7805 capable of providing up to 1.5A of regulated power. I have included 5 fuses on the board to help prevent an overload to the power supply. There is a 1A fuse feeding the output of the 5v regulator. There are three 375mA fuses providing 5v power to the I/O ports: one for the SPI ports, one for the keyboard jack, and one for the 65C22 ports. There is also a fuse for the 9v pads on the SPI ports. The fuses are optional but I recommend using them, especially the I/O port fuses, as a short could damage the power traces on the board.

SYSTEM TIMING

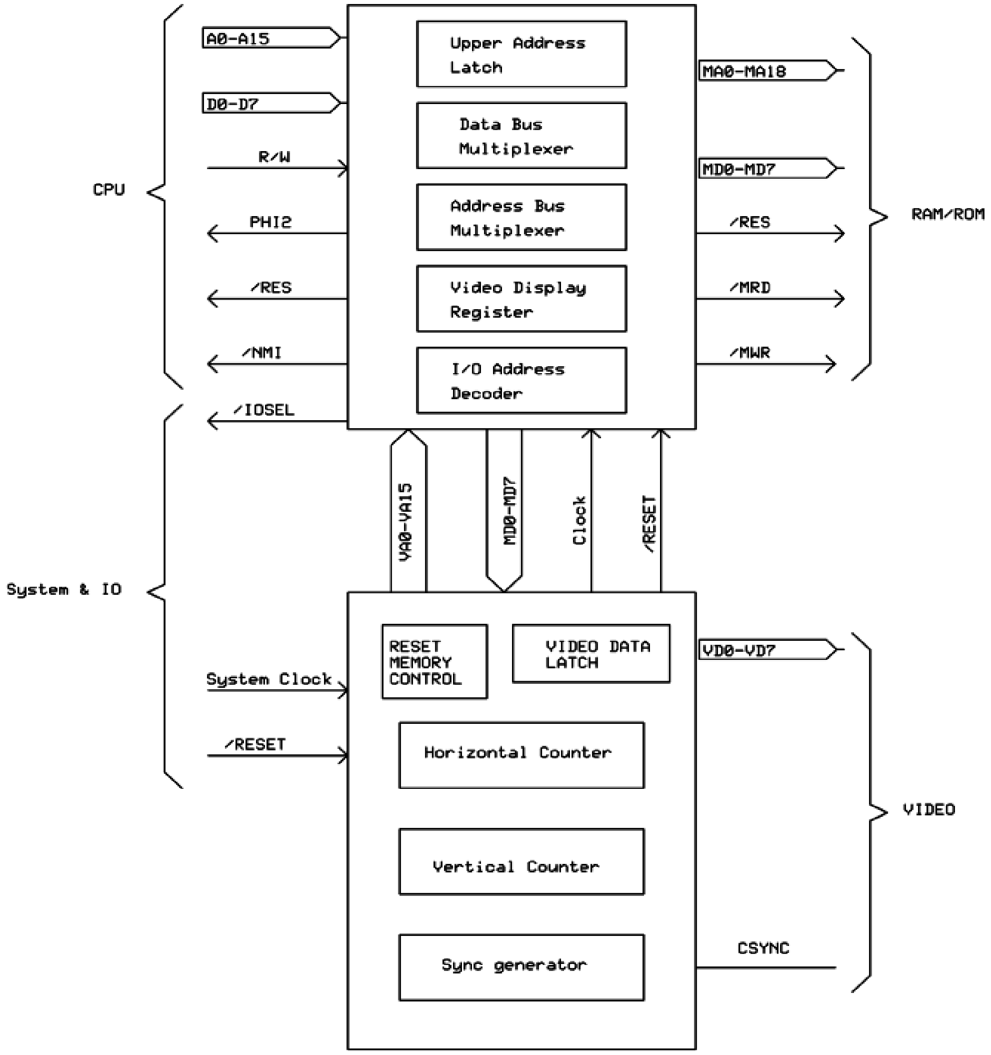
NTSC System – A 14.318 MHz TTL oscillator is used to clock the XC95108 and AD724. The XC95108 divides the clock in half and feeds the 65C816 with a 7.158 MHz PHI2 clock.

PAL System – A 16 MHZ TTL Oscillator is used to clock the XC95108. The XC95108 divides the clock in half and feeds the 65C816 with an 8 MHz PHI2 clock. The AD724 color subcarrier is 4.43 MHz and is provided from a crystal and capacitor located next to it. Alternatively, a 17.734 MHz TTL oscillator could be used for both XC95018 and AD724, just like the NTSC system.

XC95108 CPLD

The large CPLD is a Xilinx XC95108 unit in an 84-pin PLCC package. It essentially ties the whole computer together and generated the video display. There are two firmware versions for this device: one for NTSC video, the other is for PAL video.

Here is a functional block diagram:



The memory management section contains the upper address latch, bus multiplexers, I/O decoder and the Video Data Register. The upper address latch fetched the upper 3 address lines from the 65C816 data bus during the PHI2 low clock cycle. This is how the 65C816 can address more than 64k of memory. The address bus multiplexer used PHI2 to select which device has access to RAM. When the clock is low, the video system accesses RAM, and when it’s high, the 65C816 can access RAM. The data bus multiplexer gates RAM data to and from the 65C816, from RAM to Video data latch, and from the Video Display Register (VDR) to and from the 65C816.

The I/O decoder is used to select the address space from $00000-$0002F for use by the I/O devices. The VDR’s functions are described below. The non-maskable interrupt from the vertical sync circuits is also fed through the VDR and connects to the 65C816’s NMI input pin.

The video system consists of a horizontal and vertical counter, sync generator, video data latch, and reset memory controller. The counters are used to provide the necessary timing signals for the video data, blank space, and horizontal and vertical line timing. The sync generator extracts timing pulses from the counters and generated the composite sync pulses. The video data latch holds the display data taken from RAM and sends it to the digital to analog converters used to generate the color picture data. The reset memory control is used for two purposes. During reset, it provides addressing for the RAM and EEPROM and controls the RAM write pin to copy the EEPROM data to RAM. After reset, it is used to sequentially step through the video data in the RAM. It used the 3 bits from the VDR to select the memory block to display.

The video data output leaves the CPLD and goes through a simple resister-ladder analog-to-digital converter. It then is conditioned and enters the Analog Devices AD724 RGB to Composite Video converter. The CPLD’s composite sync pulse is also sent to the AD724. This chip does all the mixing of the video signals and outputs an NTSC (or PAL) composite video signal. The AD724 also has S-Video output. I did not add the conditioning components or jack, but there are pads near the AD724 to add a pigtail if S-Video is desired. The AD724 datasheet contains details on the conditioning components needed.

VIDEO DISPLAY

SBC-3 has a 320x200 pixel, 256-color composite video display supporting the analog NTSC or PAL format. The screen is stored in a 64,000-byte array in RAM. The upper left corner pixel is located at address $x0000. The “x” is a user selectable block number, ranging from $0 to $7.

The screen array is formatted like this:

	1	2	3		318	319	320
1	$x0000	$x0001	$x0002	...	$x013D	$x013E	$x013F
2	$x0140	$x0141	$x0142	...	$x027D	$x027E	$x027F
3	$x0280	$x0281	$x0282	...	$x03BD	$x03BE	$x03BF
	...	...	...	...	...	...	...
198	$xF640	$xF641	$xF642	...	$xF77D	$xF77E	$xF77F
199	$xF780	$xF781	$xF782	...	$xF8BD	$xF8BE	$xF8BF
200	$xF8C0	$xF8C1	$xF8C2	...	$xF9FD	$xF9FE	$xF9FF

On startup, the video display uses block 1 of RAM, or in other words, the display base address is set to $10000.

The display's color is generated using a simple binary encoding for the Red, Green, and Blue components. It uses 3 bits for Red (00000111b), 3 bits for Green (00111000b), and 2 bits for Blue (11000000b). Without going into a lot of theory, Blue is a stronger color than Red and Green, so we can get away with using just 2 bits. This allows us to store the color information for one pixel in one byte of memory.

Here is a small representation of the color patterns available:

Hex	bbgggrrr	Color	
$00	00000000	Black	⬛
$07	00000111	Red	🟥
$38	00111000	Green	🟩
$C0	11000000	Blue	🟦
$3F	00111111	Yellow	🟨
$C7	11000111	Violet	🟪
$F8	11111000	Cyan	🟦
$27	00100111	Orange	🟧
$B5	10101101	Gray	◼
$FF	11111111	White	⬜

Text is rendered using software to apply a character font to the graphic area. In other words, there is no hardware-driven text generation.

The System Monitor includes a simple white on black text generator. The resolution is 40x25 characters using an 8x8 pixel font. Text data is stored just above the graphics window, from $xFA00 to $xFDE7. Software generates the display using a font stored in RAM (loaded during reset from the EEPROM). The user can create custom text generation code to allow for color text or different sized fonts.

Even though the display starts in block 1, that doesn't mean you are limited to using just it. Many games use double buffering to generate video. This is simply displaying the current scene from one block (or buffer) while drawing the next scene in another block. Once the next scene is drawn, you switch the display to the new block and start drawing the next one in the first block. This method prevents flicker and other video artifacts from degrading the display. SBC-3 can use any of the 8 blocks available to display the picture. Block 0 will most likely be used for code and is not recommended. Block 1 is the default and blocks 2 through 7 can be used for multiple buffering or caching of images.

Video Display Register

The Video Display Register (VDR) is used to select which block to display, and to control interrupt and memory protection functions. It is accessed at address $00030. The actual register is located inside of the XC95108 CPLD.

These are the bit descriptions of the VDR:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
NMI	VSI	VIE	MP	VM	B2	B1	B0
		0	1		0	0	1

```
Bit 7: NMI – Non-maskable Interrupt:          (Read Only)
        0= no NMI, 1=NMI active
Bit 6: VSI – Vertical blanking active:        (Read Only)
       0=inactive, 1=active
Bit 5: VIE – Vertical Interrupt Enable:      (read/write)
       0=disable, 1=enable
Bit 4: MP  - Memory Protect:                 (read/write)
       0=read/write, 1=read only
Bit 3: Video Mode  0=NTSC, 1=PAL              (Read Only)
Bit 3: 0 - not used
Bit 2-0: Display Block # (0-7)               (read/write)
```

Power on and reset values for read/write flags are shown in the bottom row.

Bits 0-2 can be updated at any time. The current display block will continue to be displayed until after the next vertical sync has completed. The new block number is loaded as the next screen refresh begins. You can poll the NMI bit to know when the block has switched, or enable the VIE bit and use the NMI interrupt on the CPU to update the status. You do not need to worry about this function if you are not using the double-buffering technique described above.

When using the Non-Maskable Interrupt, the NMI bit in the VDR will remain set until a read of the VDR is performed. In other words, enabling VIE will cause a single interrupt during the next vertical sync period. The NMI output will remain active-low until the VDR is read. If repeated interrupts are desired, then be sure to add a read of the VDR into your NMI service routine.

The System Monitor includes a simple NMI service routine that increments location $000FF on each NMI and reads the VDR. You can use this as a simple timer or other application.

Input/Output

Communication with the outside world is accomplished using the three peripheral devices attached to the 65C816. There are two 65C22 Versatile Interface Adapters (VIA) and one 65SPI Serial Peripheral Interface adapter.

65C22 VIA

The 65C22 VIA's are 40-pin DIP packages that include two 8-bit data ports, each with 2 handshake lines. These lines can be individually programmed as inputs or outputs. They can also be combined into a single 16-bit parallel data port, with handshake. Two of the handshake lines can also form a serial data channel. There are two timers that can be used for interrupt generation, serial shift clocks, pulse width modulation, and many other applications. The VIA's provide a versatile path for adding your own I/O devices. See the device data sheet for a more detailed description of the available functions.

SBC-3's System Monitor initializes these VIA's on reset. It disables interrupts, sets the port to inputs, disables the timers and serial shift registers. The only resource used by the system is the CB1 input from VIA1, used by the ATMega8 for handshaking. All other resources in the VIA's are available for user applications.

This is the memory map for the VIA1 registers:

$00000 – VIA1PRB : port B data
$00001 – VIA1PRA : port A data
$00002 - VIA1DDRB : port B Data direction register
$00003 - VIA1DDRA : port A Data direction register
$00004 - VIA1T1CL : Timer 1 counter low
$00005 - VIA1T1CH : Timer 1 counter high
$00006 - VIA1T1LL : Timer 1 latch low
$00007 - VIA1TALH : Timer 1 latch high
$00008 - VIA1T2CL : Timer 2 counter low
$00009 - VIA1T2CH : Timer 2 counter low
$0000A - VIA1SR : Shift Register
$0000B - VIA1ACR : Auxiliary control register
$0000C - VIA1PCR : Peripheral control register
$0000D - VIA1IFR : Interrupt flag register
$0000E - VIA1IER : interrupt enable register
$0000F - VIA1PRA1 : port A data (no handshake)

This is the memory map for the VIA2 registers:

$00010 – VIA2PRB : port B data
$00011 – VIA2PRA : port A data
$00012 – VIA2DDRB : port B Data direction register
$00013 – VIA2DDRA : port A Data direction register
$00014 – VIA2T1CL : Timer 1 counter low
$00015 – VIA2T1CH : Timer 1 counter high
$00016 – VIA2T1LL : Timer 1 latch low
$00017 – VIA2TALH : Timer 1 latch high
$00018 – VIA2T2CL : Timer 2 counter low
$00019 – VIA2T2CH : Timer 2 counter low
$0001A – VIA2SR : Shift Register
$0001B – VIA2ACR : Auxiliary control register
$0001C – VIA2PCR : Peripheral control register
$0001D – VIA2IFR : Interrupt flag register
$0001E – VIA21IER : interrupt enable register
$0001F – VIA2PRA1 : port A data (no handshake)

65SPI

The 65SPI is the other CPLD on the board. It is a Xilinx XC9572 CPLD in a 44-pin PLCC package. It is a modified version of my 65SPI device. It has all the features of the standard package but also uses a few pins to decode the 65C22 address space and provide interrupt management back to the 65C816. As a trade-off for pins, the modified 65SPI does not have a dedicated external clock pin. Instead, it uses the Slave Select 7 (SS7) pin, under software control, to act as the external clock. The SS6 pin is dedicated to the on-board keyboard and RS-232 port controller. If you decide you don't want the onboard I/O, the SS6 pin is available on the SPI expansion port. In either case, SS0 through SS5 are available on dedicated ports for SPI devices. These ports provide the MOSI, MISO, SCLK, SSx; and 5vdc, 9vdc, and ground pins. The SPI expansion port has all of those pins and includes all 8 of the SS pins. This will allow you to provide extra decoding on an expansion board to yield up to 255 SPI devices. The 65SPI sits from $00020-$0027 in the 65C816's memory map.

The 65SPI Register map is here:

$00020 – SPI data port
$00021 – SPI status and control
$00022 – SCLK divisor
$00023 – Slave Select register
$00024 – Interrupt status
$00025 – Interrupt status (repeated)
$00026 – Interrupt status (repeated)
$00027 – Interrupt status (repeated)

A detailed description of the SPI Master's functions and registers can be found in the 65SPI Datasheet.

The data port is used to read and write data to the SPI. The status and control provides status from the SPI and control to the SPI. The SCLK divisor controls the SPI clock speed. It uses the lower 6 bits to control the speed. The upper 2 bits are not used. The fastest speed is with 0 loaded. It will be equal to 1/2 of the selected clock speed. The slowest is with $3F loaded. This is equal to selected clock divided by 128. The slave select is an 8-bit output port used to select the Slave SPI devices. The interrupt status register provided Interrupt status from the SPI and each of the 6522 VIA's. The CPU can use this to find which device caused an interrupt.

The SPI Status and Control register is configured like this:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
TC	IER	BSY	FRX	TMO	ECE	CPOL	CPHA	(Read Status)
	IER		FRX	TMO	ECE	CPOL	CPHA	(Write Control)
	0		0	0	1	0	0	(power up default)

TC	Transmission Complete – This flag is set when the last bit has been shifted and is cleared when the SPI Data register is read.
IER	Interrupt Enable – Interrupts are enabled when this is set to 1 and disabled when set to 0.
BSY	SPI Busy – This is 1 when data is written to the SPI data register and will stay high until the last bit is shifted.
FRX	Fast Receive mode – When set to 1, fast receive mode triggers shifting upon reading or writing the SPI Data register. When set to 0, shifting is only triggered by writing the SPI data register.
TMO	Tri-state MOSI - When set to 1, the MOSI pin will be tri-stated. When set to 0, the MOSI pin will have an active output. Tri-state will allow some three-wire interfaces to work properly.
ECE	External Clock Enable – This flag displays the selected shift clock source. 0 = PHI2 and 1 = external Shift clock pin (14).
CPOL	Clock Polarity – This flag displays the shift clock polarity. 0 = Rising edge; 1 = Falling edge
CPHA	Clock Phase – This flag displays the shift clock phase. 0 = Leading edge; 1 = Trailing edge

The Interrupt Status register is configured like this:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SPI	VIA1	VIA2					

Bit 7: SPI Interrupt - 0 = not IRQ, 1=IRQ active
Bit 6: VIA 1 Interrupt - 0 = no IRQ, 1= IRQ active
Bit 5: VIA 2 Interrupt - 0 = no IRQ, 1= IRQ active

* The changes for the Modified SPI are below:

Bit 2 of the Status and Control registers ($00021) is the External Clock Enable (ECE) flag. When this bit is 0, SS7 is an active output. When ECE is high, SS7 is tri-stated and you can put an external SCLK signal on it. The reset state is ECE = 1. This prevents an input clock from trying to drive against an active output. **If you plan to use the internal clock, then you will need to write a 0 to this bit before the SPI will work.** The System Monitor does this after a Reset is performed.

SPI Operation

The following is a guideline for initiating SPI communications.

1. If interrupt-driven transmission is desired, ensure an IRQ handler is enabled for the SPI interface.
2. If the TC bit in the SPI Status register is 1, then read the SPI Data register to clear it.
3. Enable Interrupts if desired, and set the Clock mode bits by writing to the SPI Control register.
4. Write the clock divisor value to the SCLK Divisor register.
5. Enable the appropriate Slave Select line by writing the Slave Select register.
6. Write first data byte to SPI Data register.
7. The TC bit will be set in the SPI Status register when shifting is complete.
8. Read the SPI Data register to get the incoming byte. The TC flag will clear after byte is read.
9. Process the incoming byte as required.
10. If you have more data to send or receive, repeat steps 6-9
11. After the last byte is received, de-select the Slave device.

Note 1. For fast transmit without polling and without receiving data, you can perform step 6 repeatedly without doing steps 7-9 as long as the SCLK rate is fast enough to keep up. When finished, do step 8 and 11 to clear the controller for the next operation.

Note 2. For fast receive without polling or sending data, Set the FRX bit in the control register and do steps 1 – 5. Now, just read the SPI data port, store the value, and repeat until you are done. The controller will automatically send the last byte written after each read. When done, set the FRX bit to 0 and do steps 8 and 11 to clear the controller for the next operation.

Note 3. For three wire interfaces, connect the MOSI and MISO pins together. Use the TMO bit in the control register to tri-state the MOSI pin during data reads, using the fast receive procedures in note 2.

Atmel ATMega8 Micro-controller

The Atmel ATMega8 micro-controller is a 28-pin DIP package that provides the interface to the PC keyboard and an RS-232 serial port via SPI port 6. The PC keyboard decoder takes care of converting the keyboard scan codes into ASCII character codes. As an option, the scan codes can also be sent directly to the 65C816 for enhanced user program control. The RS-232 port is provided for general data transfer. The default port set-up is 9600 baud, 8 data bits, no parity, and one stop bit. The user is able to select from the following options: 2400, 4800, 9600, 19200, 38400 baud; 7 or 8 data bits; odd, even, or no parity; and 1 or 2 stop bits. The ATMega8 provides a handshake line to 65C816 through VIA1. JP3 and JP4 are used to select either CA1 or CB1. This line can be polled or an Interrupt can be enabled on VIA1. The System Monitor uses polling on CB1, so JP3 should be strapped on 2-3 and JP4 strapped on 1-2. **This chip cannot be programmed in-system so it is highly recommended that this chip be installed in a socket for ease of removal.**

The fastest speed recommended for the SCLK when communicating with the ATMega8 is 2MHz. Therefore, the SCLK Divisor register in the 65SPI must be 1 or higher. For PAL systems, a 2 is recommended.

The System Monitor will use these ports to provide basic Input/Output terminal functions, including the ability to load and save programs and data via XMODEM file transfers to a connected RS-232 device.

The ATMega8’s available functions include:

$00 – null command (used to read back data)
$01 - Read Chip Status
$02 - Read RS-232 Receive Data
$03 - Send RS-232 Transmit data
$04 - Get RS-232 parameters
$05 - Set RS-232 parameters
$06 - Read Keyboard data
$07 - Send Keyboard command
$08 – Reset the RS-232 port
$09 – Reset the PK Keyboard

The micro-controller status byte provides the following information:

bit 7 - KB data avail – 1 = data available, 0 = no data available
bit 6 - RX data avail – 1 = data available, 0 = no data available
bit 5 - TX ready – 1 = ok to send TX data, 0 = wait to send TX data
bit 4 - RX Frame error – 1 = RX frame error, 0 = no error
bit 3 - RX overrun – 1 = RX overrun error, 0 = no error
bit 2 - RX parity error – 1 = RX parity error, 0 = no error
bit 1 - KB missing – 1 = keyboard missing, 0 = no error
bit 0 – not used

You can get the status using the System Monitor’s “Get_Status” routine. See Page 20 for more information. The following code shows the Monitor’s subroutine to get the status of the ATMega8:

```
SPI_STATUS     lda   #$01             ; GET ATM8 STATUS command
               sta   SPIDR            ; send get Status cmd
SPI_STAT1      lda   Via1_IFR         ; test VIA1’s CB1 for high-low change
               and   #$10             ; CB1 mask
               beq   SPI_STAT1        ; wait for ATM8 to signal ready
               lda   SPIDR            ; clear 65SPI flag
               lda   Via1_ORB         ; clear Via1 CB1 flag
               lda   #$00             ; null cmd (for returning data)
               sta   SPIDR            ; send data to ATM8
SPI_STAT2      lda   SPISR            ; read 65SPI status reg
               bpl   SPI_STAT2        ; wait for tx to end
               lda   SPIDR            ; read SPI status byte
               rts
```

Study the System Monitors routine located in the “SPI.INC” for more details on ATMega8 communications.

WARNING:
Since the System Monitor is continually monitoring the ATMega8 for keyboard activity, trying to access the 65SPI from the Monitor’s command line is not advisable and may cause the Monitor to hang. You should place test code in RAM and run that from the Monitor to access the 65SPI interface, or you can call one of the System Monitor’s built-in functions from the command line.

PC keyboard

The keyboard interface defaults to use the built-in Scan code to ASCII converter. You can issue a command to revert to raw scan code mode if desired. Issuing the send keyboard command followed by $00 will set the ASCII converter on. Issuing the send keyboard command followed by $01 will set the ASCII converter off. Issuing the send keyboard command followed by data > $02 will pass the data onto the keyboard, allowing you to send direct keyboard commands. A few options include:

> $ED = Set Status LED's - This command can be used to turn on and off the Num Lock, Caps Lock & Scroll Lock LED's. After Sending ED, keyboard will reply with ACK (FA) and wait for another byte which determines their Status. Bit 0 controls the Scroll Lock, Bit 1 the Num Lock and Bit 2 the Caps lock. Bits 3 to 7 are ignored.
>
> $EE = Echo - Upon sending a Echo command to the Keyboard, the keyboard should reply with a Echo (EE)
>
> $F0 = Set Scan Code Set. Upon Sending F0, keyboard will reply with ACK (FA) and wait for another byte, 01-03 which determines the Scan Code Used. Sending 00 as the second byte will return the Scan Code Set currently in Use
>
> $F3 = Set Typematic Repeat Rate. Keyboard will Acknowledge command with FA and wait for second byte, which determines the Typematic Repeat Rate.
>
> $F4 = Keyboard Enable - Clears the keyboards output buffer, enables Keyboard Scanning and returns an Acknowledgment.
>
> $F5 = Keyboard Disable - Resets the keyboard, disables Keyboard Scanning and returns an Acknowledgment.
>
> $FE Resend - Upon receipt of the resend command the keyboard will re- transmit the last byte sent.
>
> $FF Reset - Resets the Keyboard.

ASCII Conversion Table

Key	Numlock off Unshifted	Numlock off Shifted	Numlock on	Control
A	$61	$41		$01
B	$62	$42		$02
C	$63	$43		$03
D	$64	$44		$04
E	$65	$45		$05
F	$66	$46		$06
G	$67	$47		$07
H	$68	$48		$08
I	$69	$49		$09
J	$6A	$4A		$0A
K	$6B	$4B		$0B
L	$6C	$4C		$0C
M	$6D	$4D		$0D
N	$6E	$4E		$0E
O	$6F	$4F		$0F
P	$70	$50		$10
Q	$71	$51		$11
R	$72	$52		$12
S	$73	$53		$13
T	$74	$54		$14
U	$75	$55		$15
V	$76	$56		$16
W	$77	$57		$17
X	$78	$58		$18
Y	$79	$59		$19
Z	$7A	$5A		$1A
1 !	$31	$21		$11
2 @	$32	$40		$12
3 #	$33	$23		$13
4 $	$34	$24		$14
5 %	$35	$25		$15
6 ^	$36	$5E		$16
7 &	$37	$26		$17
8 *	$38	$2A		$18
9 (	$39	$28		$19
0)	S30	$29		$10
- _	$2D	$5F		$0D
= +	$3D	$2B		$1D
` ~	$60	$7E		$00
[{	$5B	$7B		$1B
] }	$5D	$7D		$1D
\ |	$5C	$7C		$1C

; :	$3B	$3A		$1B
‘ “	$27	$22		$07
, <	$2C	$3C		$0C
. >	$2E	$3E		$0E
/ ?	$2F	$3F		$0F
Esc	$1B	$1B		$0B
F1	$81	$C1		$01
F2	$82	$C2		$02
F3	$83	$C3		$03
F4	$84	$C4		$04
F5	$85	$C5		$05
F6	$86	$C6		$06
F7	$87	$C7		$07
F8	$88	$C8		$08
F9	$89	$C9		$09
F10	$8A	$CA		$0A
F11	$8B	$CB		$0B
F12	$8C	$CC		$0C
Backspace	$08	$08		$08
Insert	$90	$90		$10
Home	$97	$97		$17
PageUp	$99	$99		$19
Delete	$7F	$7F		$1F
End	$91	$91		$11
PageDown	$93	$93		$13
Up Arrow	$98	$98		$18
Left Arrow	$94	$94		$14
Right Arrow	$96	$96		$16
Down Arrow	$92	$92		$12
PrintScreen	$8F	$CF		$0F
Scroll Lock	$8D	$CD		$0D
PauseBreak	$03	$03		$02
NP /	$2F	$2F	$2F	$2F
NP *	$2A	$2A	$2A	$0A
NP -	$2D	$2D	$2D	$0D
NP 7 Home	$97	$97	$37	$17
NP 8 up	$98	$98	$38	$18
NP 9 Pgup	$99	$99	$39	$19
NP 4 Left	$94	$94	$34	$14
NP 5	$95	$95	$35	$15
NP 6 Right	$96	$96	$36	$16
NP 1 End	$91	$91	$31	$11
NP 2 Down	$92	$92	$32	$12
NP 3 Pgdn	$93	$93	$33	$13
NP +	$2B	$2B	$2B	$0B

NP Enter	$0D	$0D	$0D	$0D
NP 0 Ins	$90	$30	$30	$10
NP . Del	$7F	$7F	$2E	$0E
Left Window	$A1	$E1		$01
Left Alt	$A0	$A0	Alt Release	= $E0
Space	$20	$20		$00
Right Alt	$A0	$A0	Alt Release	= $E0
Right Window	$A2	$E2		$02
Right Menu	$A3	$E3		$03
Power	$A4	$E4		$04
Sleep	$A5	$E5		$05
Wake	$A6	$E6		$06

Control key note: It is being assumed that if you hold down the ctrl key, you are going to press an alpha key (A-Z) with it (except break key defined below). If you press another key, its ASCII code's lower 5 bits will be send as a control code. For example, Ctrl-1 sends $11, Ctrl-; sends $2B (Esc), Ctrl-F1 sends $01. Ctrl-Pause/Break is set to return $02.

The **Alt key** is decoded as a hold down (like shift and ctrl) but does not alter the ASCII code of the key(s) that follow. Rather, it sends a Alt key-down code and a separate Alt key-up code. The user program will have to keep track of it if they want to use Alt keys.

An Example byte stream of the Alt-F1 sequence: A0 81 E0. If Alt is held down longer than the repeat delay, a series of A0's will precede the 81 E0. i.e. A0 A0 A0 A0 A0 A0 81 E0.

RS-232 Serial Port

Use the Get or Set RS-232 parameters commands to see or change the port configuration. The data byte format is:

Bit 7 - not used
0

Bit 6 - stop bits
0 = 1 stop bit (default)
1 = 2 stop bits

Bits - Parity
5 4
0 0 = no parity (default)
0 1 = no parity
1 0 = even parity
1 1 = odd parity

Bit 3 - Data bits
0 = 7 data bits
1 = 8 data bits (default)

Bits - Baud Rate
2 1 0
0 0 0 = 2400
0 0 1 = 4800
0 1 0 = 9600 (default)
0 1 1 = 19200
1 x x = 38400 (x x = don't care)

Default mode is 9600 8N1, = 00001010 = $0A

Flow control provisions are not included with this serial port. If high performance RS-232 is needed, a direct SPI to RS-232 device should be used.

MAX232 Level Shifter

The MAX232 IC is a 16-pin DIP package that provides the voltage conversion from 5v logic to +/- 12v RS-232 levels. The DB9 socket is wired as a DTE device. A modem could be connected to this port or, by using a standard null-modem cable, you can connect to another computer or PC. The System Monitor supports data transfer via XMODEM CRC.

SYSTEM MONITOR COMMANDS

The monitor syntax is as follows:
{} = required [] = optional
HHHH = hex address DD = hex data

The commands are:

[HHHHHH][HHHHHH]{Return}	Dump single or multiple addresses
[HHHHHH]{.HHHH}{Return}	Dump a range of addresses
[HHHHHH]{:DD}[DD]{Return}	Modify memory at address, 1 or more bytes
[HHHHHH]{G}{Return}	Execute program at address
{HHHH.HHHH>HHHH{I}{Return}	Move range from 1st HHHH to 2nd HHHH to 3rd HHHH, start at the 2nd HHHH moving down
[HHHHHH]{L}{Return}	Disassemble 20 lines of code at address
[HHHHHH]{.HHHH}{L}{Return}	Disassemble a block of code at address Immediate values are assumed as 8 bit
{HHHH.HHHH>HHHH{M}{Return}	Move range from 1st HHHH to 2nd HHHH to 3rd HHHH, start at 1st HHHH and move up
{HH}{O}{Return}	Set serial port parameters
{HHHH.HHHH}{P}{Return}	Put memory->PC via RS-232 XMODEM transfer
{R}{Return}	Dump Register contents
{HH}{S}{Return}	Set source block number
{HH}{T}{Return}	Set destination block number
[HH]{U}{Return}	Upload a file using XMODEM from PC->SBC-3
{V}{Return}	Display System Monitor Version
[HHHHHH]{X}{Return}	Disassemble 20 lines of code at address
[HHHHHH]{.HHHH}{X}{Return}	Disassemble a block of code at address Immediate values are assumed as 16-bit
{?}{Return}	Display command help screen

XMODEM File Upload

To perform an XMODEM file transfer from a PC connected to the RS-232 port, you do the following:

An example would be a picture file being sent to address $20000

1. Make a null-modem connection between the PC and SBC-3.
2. Open a communications program on the PC (hyperterminal)
3. Set communications parameters to match SBC-3's Example (9600, N81)
4. On SBC-3 keyboard, type 02U{return}
5. In the terminal window, you will see "C"'s appear
6. Start the XMODEM/CRC transfer on the PC (Send File)

When the file has finished transferring, the SBC-3 will return to a command prompt. If you need to abort the transfer, from the terminal program on the PC, press "Esc" until the SBC-3 returns to a Command Prompt.

XMODEM File Download

To perform an XMODEM file transfer to a PC connected to the RS-232 port, you do the following:

An example would be a data file beginning at $10000 and ending at $28000. This crosses a block boundary and is $18000 bytes long.

1. Make a null-modem connection between the PC and SBC-3.
2. Open a communications program on the PC (hyperterminal)
3. Set communications parameters to match SBC-3's Example (9600, N81)
4. Start an XMODEM/CRC file transfer on the PC (Receive File) Give it a file name and begin transfer
5. On SBC-3 keyboard, type 01S{return}
6. On SBC-3 keyboard, type 02T{return}
7. On SBC-3 keyboard, type 0000.8000P{return}

When the file has finished transferring, the SBC-3 will return to a command prompt. If you need to abort the transfer, from the terminal program on the PC, press "Esc" until the SBC-3 returns to a Command Prompt.

System Monitor Routines

Here are the addresses for some useful routines in the System Monitor:

$EFDC	Clearsc	Clear the screen
$F4D7	delay	Delay loop based on value of A reg
$f101	drawscrn	Redraw the text screen
$EEA2	dispinit	Initialize the text Display
$F37A	get_ser	Wait for a byte from RS-232 port
$F26D	Get_Status	Read status of ATMmega8, ret in A reg
$F473	input	Print a prompt and get a line of characters from keyboard Buffer=$00200
$EEC4	output	Print character in A reg to screen
$F448	print2byte	Print 16 bit Hex value from X & A reg
$F44C	print1byte	Print 8 bit Hex value from A reg
$F455	printdig	print 4 bit Hex value from A reg
$F466	printxsp	Print # of spaces stored in X reg
$F46B	print2sp	Print 2 spaces
$F46E	print1sp	Print 1 space ($20)
$F43B	print_cr	Print CR and LF characters ($0D, $0A)
$FF00	reset	Run system reset code
$F30F	scan_serin	If data ready on RS-232, Set Carry flag
$F350	ser_out	Send character in A reg to RS-232 port
$F323	ser_in	Wait for character from RS-232
$F29A	set_kb	Send command to KB from A reg
$F394	set_ser	Ser Serial port parameters from A reg
$F2E8	spi_kbscan	If data ready on Keybrd, set Carry flag
$F2BD	spi_kbin	Wait for byte from keyboard, ret A reg

SUPPORT PACKAGE

I am including a support package that contains all of the IC datasheets, the system schematics, the board layout and trace details, a parts list, the source and JEDEC program files for both CPLD’s and the ATMega8, and the source and assembled copies of the System Monitor. You are free to modify anything you wish.

I encourage all who build the SBC-3 or use this information to build a modified version to share their experiences and designs with others. You are welcome to send me any details of your projects and I’d be happy to include them on my web site. This includes SPI devices, other hardware additions, Operating System software, and software applications, cool graphics; anything others might find useful.

For questions, email Daryl Rictor at sbc2@surewest.net

Website: http://sbc.rictor.org/

V0.1 – First draft
V1.0 – First release: 11-09-2008