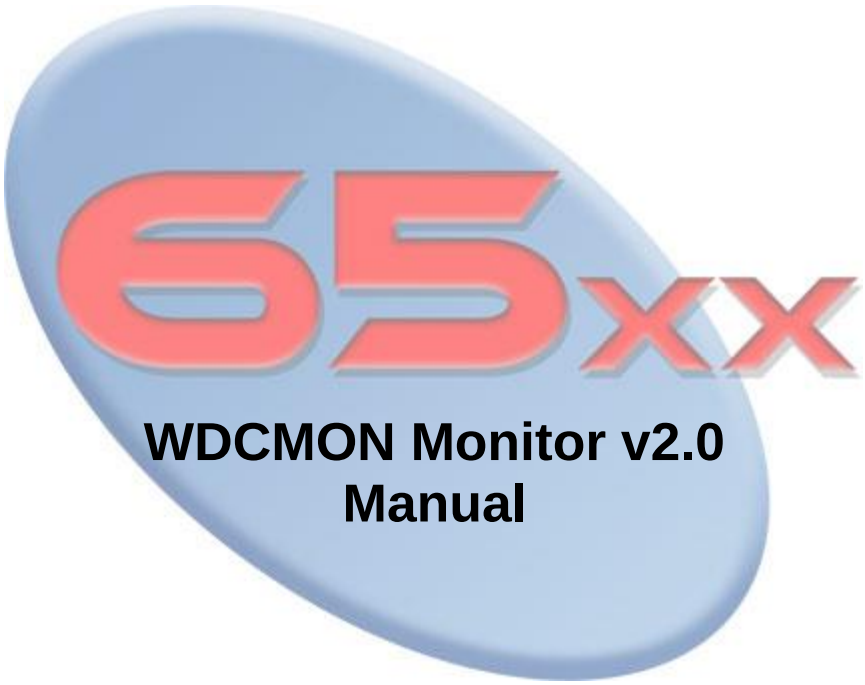


WDCMON Monitor v2.0 Manual

Intentionally left blank

DOCUMENT REVISION HISTORY

Version	Date	Author	Description
1.0	23-Aug-25	David Gray, Bill Mensch	Initial Document Entry
2.0	23-Aug-30	Bill Mensch, David Gray	Some text and formatting improvement

Table of Contents

Intentionally left blank

Introduction and Description

An electronic system can be described as having both system hardware and system software features. System software sometimes called a Monitor or basic IO system (BIOS) can be changed as in FLASH or EEPROM is not volatile, called firmware. System software in RAM is volatile software, when the power to the system is removed the software vanishes. System software that can not be modified as in a mask ROM can be thought of as software that is hardened. System software can include a monitor, operating system (OS) and application software.

WDC's monitor, WDCMON, is in FLASH on the W65C02SXB, W65C816SXB and MyMENSCH™ provides the functions to start up the system, load and store code and data to and from the system. WDCMON can be updated to include other features as determined from time to time.

The W65C134S and W65C265S System-on-Chip (SoC) monitors are explained in their Monitor Manuals, are not the WDCMON, are in the SoC Mask ROM (MROM) with libraries of useful routines that can be expanded with routines loaded in the SXB FLASH to handle added hardware modules. The W65C134S and W65C265S SoCs have been designed so the user can actually copy the contents of the MROM into FLASH and run externally in FLASH.

When one adds on the W65C02EDU and W65C816EDU to the SXBs and the W65Cx65PRO to the MyMENSCH™ one can add system software to handle added modules.

This manual was created to describe the details of the WDCMON and how it communicates with the host PC USB User Port for use of the Debugger or Uploader.

Both the Debugger and Uploader are part of the WDCTools and resides in the directory C:\WDC\Tools\bin. The customer can leave WDCMON in their final project or remove it after the development process is over.

Future Enhancements

WDCMON is a living monitor that has evolved since WDC's original DB boards that used an LPT Printer port. Features and communication interfaces will continue to evolve overtime and should be expected. WDC also encourages users to adopt WDCMON for their own system and adapt new features for all to enjoy. Below is a description of identified enhancements to WDCMON.

Support for Banks 0, 1, and 2 of the SST FLASH – The SST FLASH has a capacity of 128Kbytes wired to have 4 banks of that 32KB of FLASH. Currently the W65C02SXB and W65C816SXB have the monitor in Bank 3. Future addition would allow for reading, writing, and executing to/from Banks 0,1, and 2.

Support for W65C134SXB and W65C265SXB – The W65C134SXB and W65C265SXB both have a socket for the SST FLASH, but currently use the internal ROM Monitor of the W65C134S and W65C265S chips.

Hardware Interface

The WDCMON can use either the VIA parallel interface, GPIO parallel interface or ACIA/UART serial interface. WDC used the faster interface, the W65C22S VIA parallel port on the W65C02SXB and W65C816SXB using the Debugger for connecting to a USB port of a Windows PC. For a Mac/Linux machine we recommend using the serial UART Python Uploader. MyMENSCH™ Rev-A and Rev-B, use the GPIO pins similar to the VIA for interfacing to a Windows PC. MyMENSCH™ Rev-C uses a UART Python Uploader for PC/Mac/Linux development hosts.

I/O Pin description:

W65C02SXB/W65C816SXB Using U5 W65C22 with FT245R

- Port “A”, VIA ORA are the data bits
- Port “B”, VIA ORB are the control lines for working with the FT245R

VIA Signal	FTDI Signal / Function
PB0	TXE – Transmitter Empty
PB1	RXF – Receiver Full
PB2	WR – Write Strobe
PB3	RD – Read Strobe
PB5	PWREN – Power Enable

MyMENSCH™ Rev-A/Rev-B Using GPIO with the FT245R/FT240X

- GPIO C (0x7F10) is used for the data
- GPIO D (0x7F18) is used for the control lines for working with the FT245R/FT240X

VIA Signal	FTDI Signal / Function
GPIOD0	TXE – Transmitter Empty
GPIOD1	RXF – Receiver Full
GPIOD2	WR – Write Strobe
GPIOD3	RD – Read Strobe
GPIOD5	PWREN/SIWU– Power Enable/Send Immediate/Wake-UP

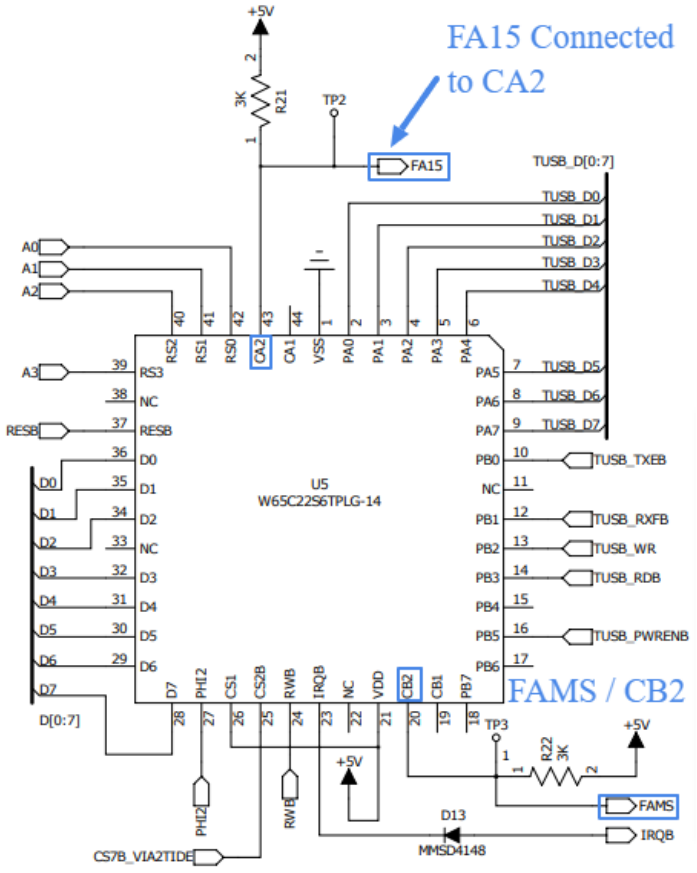
MyMENSCH™ Rev-C Using ACIA A with FT232/CH340

- ACIA A (0x7F60) is used for the data
- GPIO E (0x7FA0) is used for the control lines for working with the FT232/CH340

FLASH Memory Control for SXB Boards.

WDC's SXB's all have a socket for the FLASH Memory chip. The W65C02SXB and W65C816SXB have a 128K Flash ROM that can be updated under from the monitor as of v2.0. Both boards divide the ROM into four 32Kbyte banks, each mapped into the memory area between $00:8000 and $00:FFFF. The WDCMON currently resides in Bank 3 from $00:F000 - $00:FFFF but in the future may reside in each bank for flexible use. The control lines for the banks are controlled by the same W65C22 device that controls the hardware interface to the PC/host. FA15 is wired to CA2, and FAMS is wired to CB2. A different bank can be selected by changing FAMS and FA15 according to the table below. These changes should occur in RAM to avoid switching banks during the process. Bank 3 is the default bank on startup due to the pull-up resistor on the FAMS/CB2 and FA15/CA2 signal and the default state of CB2 and CA2.

Bank	FAMS/CB2	FA15/CA2
0	0	0
1	0	1
2	1	0
3	1	1



WDCMON Commands:

The command set is generally the same across processor families, but differences exist due to the inherent architectural differences. Commands invoked while the processor is running and involving memory will "steal" bus cycles from the CPU, much as a DMA controller would. As the debugger reads and writes memory and registers, and halts the CPU and restarts it, the processor proper doesn't have any knowledge that these activities are taking place. When it's not halted, it simply hums along fetching instructions as specified by the program.

WDCMON responds by waiting for $55 and $AA from the PC.

WDCMON then responds with a $CC to the PC.

WDCMON then waits for a command from the PC to read or write Data or execute.

For Versions v1.x, there were only 6 commands from the PC to the board:

```
sync              - (00) resync data port, send Sync byte ($00) to PC
echo              - (01) Echo response
read_data_from_PC - (02) Read an Address (3 bytes), data block size (2 bytes)
write_data_to_PC  - (03) Write
registers         - (04) Setup pointer to internal Registers and send this to
the PC
exec              - (05) Execute code
```

For v2.x and beyond, there were several functions added to support writing of the FLASH, updating the monitor, etc. Versions v2.x and beyond have the following functions:

```
sync                    - (00) resync data port, send Sync byte ($00) to PC
sync                    - (01) Echo response
Write_Data_To_Memory    - (02) Write an Address(3 bytes), data blk sz(2 bytes)
Read_Data_From_Memory   - (03) Read an Address
sync                    - (04) No longer used, Internal Registers
sync                    - (05) Execute code For Debug
Execute                 - (06) Execute code Directly
Write_Data_To_Flash     - (07) Write Data to Flash
Read_Data_From_Flash    - (08) Read Data from Flash
Clear_Flash             - (09) Clear Flash
Check_Flash             - (0A) Check if Flash is Clear
Execute_From_Flash      - (0B) Execute Code Flash
Board_Info              - (0C) Board and Monitor Version
Update_Monitor          - (0D) Update Monitor
```

These commands are similar to those of a typical debug monitor. The remote debugger builds on simple capabilities like reading and writing registers and individual memory locations.

Command Example – Command $02

PSEUDO CODE FOR READING DATA FROM A PC AND STORING IT IN RAM ($02) ON W65C02SXB/W65C816SXB/MYMENSCH REV-A/B

- Get 3 bytes from the PC to be used as the destination Address (Lowest byte (RAM0) of Address first), (RAM0-RAM2).
- Check to see if there is a conflict with WDCMON internal RAM ($00:0000 - $00:0005) for the W65C02.
- Get 2 more bytes from the PC – Number of bytes to download (Lowest byte (RAM3) of size first) (RAM3-RAM4).
- Get a Data byte.
- Check to see if there are any more bytes. If so, get them.

Read (Get) Byte Function:
- Wait until Receiver Full signal is set
- Set RD read strobe low
- Get Data byte on Port A.
- Set RD read strobe high

Monitor Library Subroutines:

In addition to the Commands several basic functions have been added to support simple use of the hardware interfaces that are present on the SXBs and MyMENSCH™. The following functions are present in v2.x and later versions of WDCMON. Note: In the case of the Code port functions, these are the same routines that the WDCMON uses for all code port communication. The locations have just been locked and set up for use by the user starting at address $00:F800.

Routine	*Vector*	*Description*
Init_Serial	$00:F800	Code port – Initialize serial
Write_Byte	$00:F803	Code port – write byte to host
Read_Byte	$00:F806	Code port – read byte from host
Check_Byte	$00:F809	Code port – check if byte received from host
i2c_Init	$00:F80C	I2C - Initialize
i2c_Wr_Control	$00:F80F	I2C – Write control (direction and start)
i2c_Wr_Data	$00:F812	I2C – Write Data
i2c_Wr_Data_Stop	$00:F815	I2C – Write Data and send Stop command
i2c_Rd_Data	$00:F818	I2C – Read Data and send ACK
i2c_Rd_Data_NACK	$00:F81B	I2C – Read Data and send NACK
i2c_Wr_Stop	$00:F81E	I2C – Send Stop command
spi_Init	$00:F821	SPI - Initialize
spi_Write_Read	$00:F824	SPI – Write and Read
ACIA_Init_Serial	$00:F827	ACIA – Initialize port
ACIA_Write_Byte	$00:F82A	ACIA – Write a byte of data to host
ACIA_Read_Byte	$00:F82D	ACIA – Read a byte of data from host
ACIA_Check_Byte	$00:F830	ACIA – Check to see if a byte has been received

Retargeting WDCMON for a User Designed System

The WDCMON is used on MyMENSCH™ for development of educational and commercial FPGA SoC designs that can be retargeted to ASIC SoC designs.

A hardware design can have a VIA/ACIA/GPIO as a daughter board through a dip header that has address lines, data lines, and control pins. These daughter board pins are useful for manufacturing production board testing. The Developer can also opt to put a VIA in an area that can be cut off the main board after testing and debugging is finished. The WDCDB Debugger in simulation mode can be used to partly debug code before the target hardware is available.

The source program is contained in the files *wdcmon.asm* and uses several files from the include folder. The files used will depend on the microprocessor, code interface chip, external FLASH, I2C/SPI/External ACIA used. For adding a new board to the WCDMON, create a unique variable name (i.e. USING_SXB_02) for the IF statement. Below is a sample of the W65C02SXB file setup from the *wdcmon.asm* file.

```
      IF    USING_SXB_02
            include include/main02.inc
            include include/ftdi245_VIA.inc
            include include/Flash-SST.inc
            include include/i2c_BitBang.inc
            include include/spi_BitBang.inc
            include include/ACIA_SXB.inc
```

Within the wdcmon.asm file, the board information section would need to be filled out with specifics about your design. The WDCMON for the W65C02S and W65C816S use 23 bytes in Zero Page RAM ($00:0000 - $00:00022), so make sure RAM is available, although most systems do this.
Please contact WDC for any technical assistance needed. Here is the board information using the W65C02SXB example again:

```
      IF    USING_SXB_02
            title  "WDCMON Monitor Program V 2.00 for SXB 65c02 Board"
            sttl
BRD_TYPE    gequ 'X'

Model_Start
Board_No          db "SXB2"
H_Version_No      dl 101      ;$00,$01,$00,$02
S_Version_No      dl 2005     ;$00,$00,$00,$30
Model_End

I_O_BASE          gequ $7F00
I_O_END           gequ $7FFF
HARD_BRK_BASE     gequ $FFFFFF
WDCMON_RomBeg     gequ $F800
CPU_TYPE          gequ $02
WPR               gequ $FFFF
```

In addition to changing the wdcmon.asm file and creating or modifying any of the include files, the user will have to make changes to the uploader python script so that the board is properly supported.