User Manual

for the

W65cSDS Monitor (WDCMON) V1.04

OBJECTIVE

This manual is designed to help in the porting of the customer's project/target in order to establish communication with the WDC Debugger (WDCDB) program. The Debugger is part of the W65cSDS and resides in the directory C:\WDC_SDS\bin.The customer can leave the monitor code in their final project or remove it after the debugging process is over. The source program is contained in the files wdcmon.asm and eval.equ. The individual boards and software are defined in the DOS "Batch" files used to assemble the target code. The customer can also elect to use our SDS Loader as an example of PC source code to write their customized interface between their target and a PC. The "Loader" example can be used as a "Data Acquisition" interface to upload or download data to the target 65cxx. The source code for the SDS Loader is included for the customer in "C:\WDC_SDS\DEBUG\SK_Loader.
This is not a Real-Time Emulator, but it is fast!

For new Projects:
The customer can design a VIA into their project 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 customer can also opt to put a VIA in an area that can be cut off the main board after testing and debugging is finished.

Our WDCMON source code is a versatile solution for designing an ASIC product. This code is especially useful for ASIC development using our WDCDB. (The W65C134 is an example of how to interface an ASIC part that has I/O interrupt vectors)

The WDCMON uses RAM memory from $00:7D00 - $00:7FFF for Registers, Flags, and Shadow Vectors. It also uses Zero Page RAM ($00:0000 - $00:0005).

For Old Projects:
The project must integrate the target memory map with our ICE board memory map, which could require the modification of our CPLD code to avoid memory mapping conflicts.

THE WDCMON PROGRAM

In the beginning, there was the TTY/UART debug monitor. Inexpensive but effective, it still serves handily alongside the most expensive debugging tools. Yet monitors have their drawbacks and weaknesses. For instance, they require ROM, RAM, and a communications channel from the target; they need to be ported to the target hardware; and they don't let you set breakpoints in programs running out of ROM. We have a simple UART debug monitor we offer for both the W65C134 and W65C265. They are located in C:\Wdc_sds\134_Internal_Monitor_software\V107 and C:\Wdc_sds\265_Internal_Monitor_software\V207 respectively. They can be modified to run on the W65C02 & W65C816 if an external UART is made available.

The WDCMON program is a step above the UART debug monitor. We use a faster interface, a parallel port, (65c22/VIA). We also pass only data and add the extra GUI/(data type) information on the PC side (WDCDB). It is designed to work with the Western Design Center's Developer Boards - W65C02DB (8 bit), W65C134DB (8 bit), W65C816DB (16 bit), and W65C265DB (16 bit).

We define a communications standard for interfacing to a debug core via a 26-pin connector on the target system. A host computer, running the WDCBD remote debugger, can communicate with and control the processor through this interface. This is a two-way communication to the PC using a command sequence. The command sequence is initiated by using the VIA to create the command sequence on the WDC Developer Board. The **NMI** is always engaged for the WDCMON Software. (The VIA IRQ is brought thru to the processor. It is not used for the WDCMON, so the user can use the other half of the VIA and its IRQ).

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.

Commands to the WDCMON:
The **NMI** is always engaged for the WDCMON Software.
This enables the WDCMON to respond to the PC. The WDCMON only responds to the NMI interrupt.
The WDCMON responds with waiting for a $55 and $AA from the PC.
The WDCMON then responds with a $CC to the PC.
The WDCMON then waits for a command from the PC to read or write Data or execute.

There are only 6 commands from the PC to the WDCDB 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), Data
write_data_to_PC	- (03) Write
registers	- (04) Setup pointer to internal Registers and send this to the host/PC
exec	- (05) Execute code
noop	- (06) Do nothing
noop	- (07) Do nothing – (Future implementation– Send version & date)

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.

Method of Debugging the Customer's Target Board:
The Software design group can use the WDCDB in simulation mode to partly debug code before the target hardware is available. The WDCDB can then be used in emulation mode to partly debug a portion of your code after you put it in EEPROM. Target hardware and software can also be debugged by porting the wdcmon.asm and eval.equ files into the target code. This will allow the target to interface to the WDCDB as a debug platform.

TECHNICAL NOTES FOR THE PARALLEL PORT INTERFACE

Warning:
Please define ALL IRQ vectors in Vector tables.
Make sure the NMI pin is connected from the Display board to the Main Starter Kit board

Warning: Some Zero page ram is needed for indexed indirect opcodes.
- The WDCMON02, 816, & 265 use 5 bytes in Zero Page RAM ($00:0000 - $00:0005).
- The WDCMON134 uses 5 bytes in Zero Page RAM ($00:00EA - $00:00EF), because of a conflict with the W65c134 I/O ports at $00:0000 - $00:0005.
- The $00D0-00DF addresses are reserved for the hardware breakpoint registers.
- Non-Zero Page Ram is $00:7D00 - $00:7FFF.
- Download is limited to 65K bytes in one block.
- Data read from I/O ports may be unreliable because some I/O devices read different internal buffers when they write.

I/O Pin description:
- Port "A", VIA ORA are the data bits (Pins 7, 8, 9,10,11,12 on J1, Pins 2-9 on the DB25)
- VIA PCR is the data ready signal to the PC via CA2 pin.
  CA1 - Pin 5 on dip header J1, Pin 1/Strobe on the DB25.
  CA2 - Pin 6 on dip header J1, Pin 10/Acknowledge on the DB25.
- VIA IFR is polled to see if the handshake was received by way of a VIA IRQ.
- Port "B", VIA ORB are the LED segments and two push buttons (CB1 and CB2).
- PC Parallel I/O port - Typical
  LPT1 - $3BC - $3BF
  LPT2 - $378 - $37F
  LPT3 - $278 - $27F
- PC Parallel Port Status - $279 (Inputs in PC Software).
  Acknowledge – bit $40 (Ack is a BAR, i.e. negative logic).
- PC Parallel Port Control - $27A (Outputs in PC Software).
  Strobe - bit $01 (Strobe is a BAR, i.e. negative logic).

Other lines are:
- PC Parallel Port Status - $279
  Busy - $80
  Ack - $40
  PE - $20

Select - $10
Error - $08
IRQ - $04
Reserved-$02
Reserved-$01
- PC Parallel Port Control - $27A
Reserved-$80
Reserved-$40 EPP Enable
Direction-$20
IRQ - $10
Select in - $08
Init - $04
Autofeed-$02
Strobe - $01

For the average program or function testing, use the RAM between $00:200 and $00:7CFF.
The target software can use the extra half of the VIA (Port B) and CB1 & CB2 for application I/O.

PSEUDO CODE FOR READING DATA FROM A PC AND STORING IT IN RAM ($02)

- The Init pin sets CA2 high and enables the CA1 Independent Negative Edge IRQ.
- 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.

Get Byte Function:
- Set CA2 low (Pin 6 on dip header J1, Pin 10/Acknowledge on the DB25) and Enable CA1 Negative Edge IRQ.
- Wait until IRQ Flag is set. I.e., we got a negative edge on CA1 (Pin 5 on dip header J1, Pin 1/Strobe on the DB25).
- Get Data byte on Port A.
- Set CA2 High (Pin 6 on dip header J1) and Enable CA1 Positive Edge IRQ.
- Wait until IRQ Flag is set (High). I.e., we got a positive edge on CA1 (Pin 5 on dip header J1).
- Leave CA2 High (Pin 6 on dip header J1) and Enable CA1 Negative Edge IRQ.

RETARGETING FOR USER IP AND USER APPLICATION BOARDS

Refer to WDCMON.ASM and EVAL.EQU
To retarget the WDCMON source code, look for the strings “IF” “ELSE” for a given processor similar to one in the code.

Note:
Starting in Version 1.04, Version & Assembly Date have been added to the listings and S28/EPROM code for each WDCMON.

FUTURE ENHANCEMENTS TO THE WDCMON:

Serial Interface for WDCMON

The **NMI** is always engaged for the WDCMON Software. The DSR pin of the UART will need to toggle the Starter Kit NMI.
A jumper will be needed to switch between sources of NMI, so make sure that the jumper is installed between the SK & parallel or serial port.

Other Items to be Implemented:
The W65C134 and W65C265 will have the bus going out all the time so that the developer can hang a logic analyzer on the address and data lines.

REVISIONS

To be added at a later date.

