AVR 65C02 Emulator

Written By Daryl Rictor

I created this project to provide a simple, inexpensive computer system that could be used to learn computer programming and microprocessor controls. I chose to emulate a 65C02 microprocessor as this is well-known general purpose device that has a wealth of information, code examples, and support available. Placing the 65C02 into an AVR microcontroller allowed me to keep all features onboard one chip while provide a wide range of IO device support, including two USART's, I2C, SPI, four 8-bit data ports. There are also 3 timers and an 8 channel ADC.

The system features include:

- 65C02 instruction set
- 16K Bytes of RAM (15.75 actual)
- 48K bytes of ROM
- Full access to the AVR's I/O registers
- IRQ input is supported from all AVR interrupt sources
- NMI, RDY, SO inputs are not supported

The Instruction set includes all of the WDC 65C02 instructions. Invalid (unused) opcodes are treated as NOP instructions. Decimal mode is also supported.

The Emulator's 65c02 engine runs at about 2.1 MHz. Some minor fluctuations occur depending upon the memory accesses used but worst case is 1.9 MHz and best case can be up to 2.3 MHz.

The default OS is based on my SBC-2 OS but users can download their own 48k byte ROM image and run their own custom applications. The AVR's flash has a 10,000 write-cycle rating so making periodic changes should not be a problem.

Included in the SBC-2OS are Lee Davison's EhBASIC, FigForth, Peter Jennings' MicroChess, and my own mini-assembler and disassembler. There is still 24k of ROM space available to add more features.

To exit from EHBASIC, type "SYS"
To exit FigForth, type "mon"
To exit from MicroChess, type "q"

Upon power up, the Boot loader will prompt you to either start the emulator or download a new ROM image. Press [Enter] to start the emulator or press [u] or [U] to begin the download. There is a simple delay built-in that will auto-start the Emulator after about 15 seconds if no key is pressed. There will be eight periods (. . . . .) printed during this delay so you can see the bootloader is working.

The download protocol is XMODEM/CRC only. The file must be a raw binary file that is exactly 49152 bytes long, corresponding to addresses $4000-$FFFF in ROM. Any other file type or size will result in improper operation of the emulator. I have included a copy of the SBC-2OS image as a back-up copy.

In the unlikely event the 6502 Engine gets corrupted, you can download a clean copy from the bootloader. To re-load the 65C02 Engine, you press the | key at the bootloader prompt. This will start an XMODEM file transfer from the terminal. You can use the “avr6502.bin” included in the distribution to restore the Engine to default. This will reload the Emulator engine and the SBC-2OS and get you back to “factory default.”

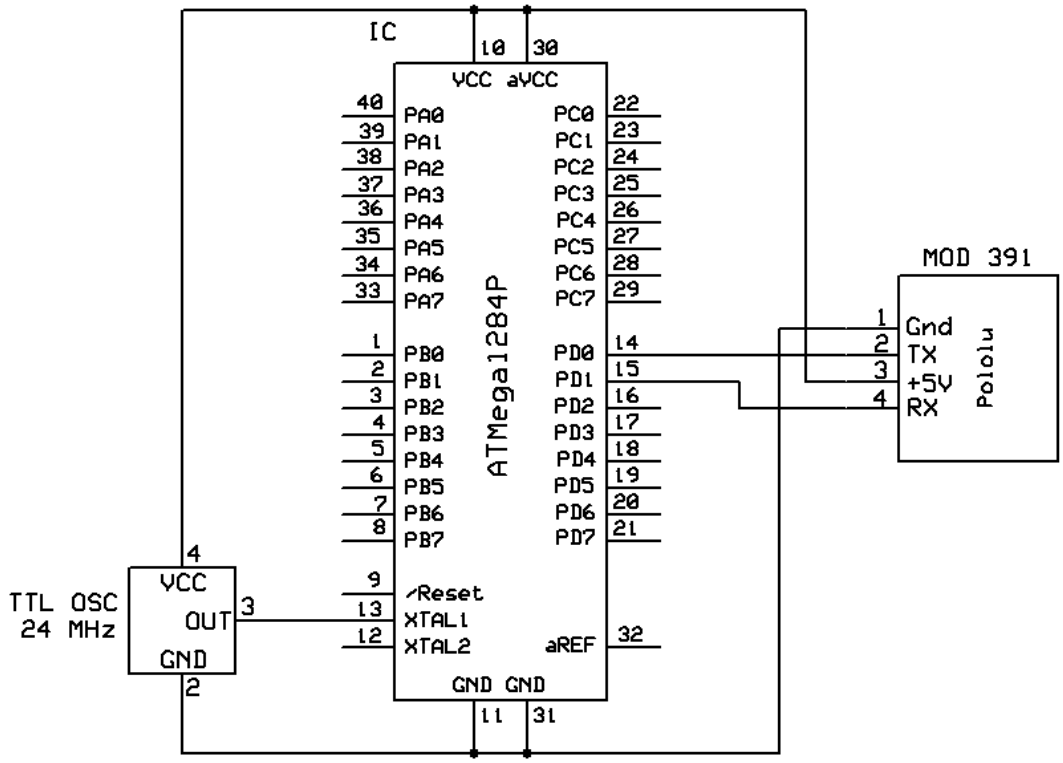
Component List (Digikey Part # and cost included):

1 – ATMega1284P Microcontroller (ATMEGA1284-PU-ND $7.33)

1 – 24MHz TTL oscillator (CTX757-ND $2.49)

1 – 5v TTL RS-232 to USB adapter (Pololu.com Item #391 $14.95)

Schematic:



Construction:

Simple point-to-point wiring can be used for this project. The Pololu Module can provide up to 25mA of 5 volt power to the system. If you connect more IO devices and exceed that total, disconnect the 5v output from the module and use an external 5v power supply connected to the VCC and GND pins.

Programming the ATMega1284P:

Use the AVR6502.hex file from the AVRSource folder to load the AVR. Also, you will need to modify the fuses to enable the external oscillator and set the boot location to $F000. Here are the fuse settings:

LOW: 0xC0
HIGH: 0xD8
EXT: 0xFF
LOCK: 0xFF

The memory map for the emulated processor looks like this:

$0000-$3EFF – System RAM (just under 16k)
$3F00-$3FFF – IO page (see the IO address assignments below)
$7F00-$7FFF – $8000-$FFFF – 48k ROM

The IO page is broken down into two sections.

The main section is located from $3F20 - $3FFF. This is the AVR's memory-mapped IO registers. I will not list them all here as you will need to become familiar with their functions from the AVR's datasheet anyway. Just go the Register Summary section. The address given in parentheses is the memory-mapped address. For instance, the last entry looks like this:

0x00 (0x20) PINA …

Add $3F00 to the address shown, i.e., the PINA register is located at $3F20

The registers here can be read from or written to just like any other memory address in the emulator.

*** NOTE ***

The emulator uses GPIOR0 for writing to EEPROM, so please do not use that register for user functions.

The other IO Page section is located from $3F00 - $3F1F. This is a special use area as the AVR's IO space is mapped from $20 to $FF. I take advantage of this with two added features.

1) Reading any address from $3F00 - $3F1F will return the status of the Emulator's interrupt vector. This can be used to determine which IO device caused an interrupt. The vector value is from $02 - $44 using even numbers only, so a fast jump table can be used. $00 will mean no native interrupt occurred.

The jump table code would look like this:

```
        ldx     $3F00
        jmp     (irqjumptbl,x)    ; irqjumptbl would be a table of jmp command to each handler.
```

Here's the table of the Interrupt Sources:

02 - INT0V	External Interrupt Request 0
04 - INT1V	External Interrupt Request 1
06 - INT2V	External Interrupt Request 2
08 - PCINT0V	Pin Change Interrupt Request 0
0a - PCINT1V	Pin Change Interrupt Request 1
0c - PCINT2V	Pin Change Interrupt Request 2
0e - PCINT3V	Pin Change Interrupt Request 3
10 - WDT	Watchdog Time-out Interrupt
12 - TIMER2_COMPA	Timer/Counter2 Compare Match A
14 - TIMER2_COMPB	Timer/Counter2 Compare Match B
16 - TIMER2_OVF	Timer/Counter2 Overflow
18 - TIMER1_CAPT	Timer/Counter1 Capture Event
1a - TIMER1_COMPA	Timer/Counter1 Compare Match A
1c - TIMER1_COMPB	Timer/Counter1 Compare Match B
1e - TIMER1_OVF	Timer/Counter1 Overflow
20 - TIMER0_COMPA	Timer/Counter0 Compare Match A
22 - TIMER0_COMPB	Timer/Counter0 Compare match B
24 - TIMER0_OVF	Timer/Counter0 Overflow
26 - SPI_STC	SPI Serial Transfer Complete
28 - USART0_RX	USART0 Rx Complete
2a - USART0_UDRE	USART0 Data Register Empty
2c - USART0_TX	USART0 Tx Complete
2e - ANALOG_COMP	Analog Comparator
30 - ADCC	ADC Conversion Complete
32 - EE_READY	EEPROM Ready
34 - TWI	2-wire Serial Interface
36 - SPM_READY	Store Program Memory Ready
38 - USART1_RX	USART1 Rx Complete
3a - USART1_UDRE	USART1 Data Register Empty
3c - USART1_TX	USART1 Tx Complete
3e - TIMER3_CAPT	Timer/Counter3 Capture Event
40 - TIMER3_COMPA	Timer/Counter3 Compare Match A
42 - TIMER3_COMPB	Timer/Counter3 Compare Match B
44 - TIMER3_OVF	Timer/Counter3 Overflow

2) Writing to any address from $3F00 - $3F1F will trigger an EEPROM Write cycle. The AVR's 4k of EEPROM is available for user storage. It can be read using the EEPROM access register in the IO register set. You will need to refer to the AVR's datasheet for more details. Writing to EEPROM on an AVR requires two successive accesses to an IO register with 4 clock cycles. You cannot achieve that using the emulator so I coded a write access to trigger an AVR routine to perform the write. And, just to ensure accidental writes to $3F00 - $3F1F do not corrupt the EEPROM data, I added a level of safety. You must first write the value $5A to the GPIOR0 register, located at $3F3E. The AVR will test this register and if it does not match $5A, then it will not perform the write sequence. The value of GPIOR0 is set to $00 during system reset. Be sure to change the GPIOR0 value after you have completed writing data to EEPROM.

Summary

Source code for the project is provided. You are free to use it for personal use without cost. Any commercial use of this software is subject to a license agreement and possible fees. Please respect my hard work and feel free to contact me if you have any questions.

I have included a method of installing updates without using an AVR programmer so any updates that I make can be easily installed by the end user.

This project, including source code and documentation are provided as-is and without any warranty. Use it at your own risk.

Thank you very much.

Daryl Rictor