Introduction

A VHDL model of the 8051 microcontroller has been developed at the University of Missouri – Rolla. The model supports the entire 8051 instruction set and is timing compatible with the 8051. This 8051 model combined with other hardware simulation models may be used to simulate a complete hardware system. 8051 software may then be simulated simultaneously with the hardware to allow cosimulation of a hardware/software system.

This report gives an example of a complete simulation-ready system incorporating the 8051 model. This is followed by a description of the full capabilities of the VHDL model. The external interface of the 8051 model is described in detail.

The internal implementation of the 8051 model is also described. The part of the implementation necessary for basic code fetches and code execution is described. This is followed by a description of the peripheral timers and serial UART.

Testing a piece of software on a model of the targeted hardware is not useful unless the model itself is known to be correct. The VHDL model was extensively tested in the spring of 2000, and bugs were fixed in the fall of 2000. This report describes the testing methodology used to verify correct operation of the 8051 model.

VHDL 8051 Model Technical Description

System Overview

The VHDL 8051 model was based on the 80C51 Technical Description found in the handbook 80C51-Based 8-Bit Microcontrollers from Philips Semiconductors. The model is contained in three source files: pack8051.vhd, containing definitions, constants, and simple functions; mc8051.vhd, containing the processor entity and architecture description; and uart.vhd, containing the serial UART description.

To create a complete system, 64 kbytes of RAM and an eight-bit address latch are interfaced with the model. The RAM is described in nvsram.vhd and the address latch is described in latch373.vhd. These components are tied together in the file system.vhd. See Figure 1 for the block diagram of this system level model.

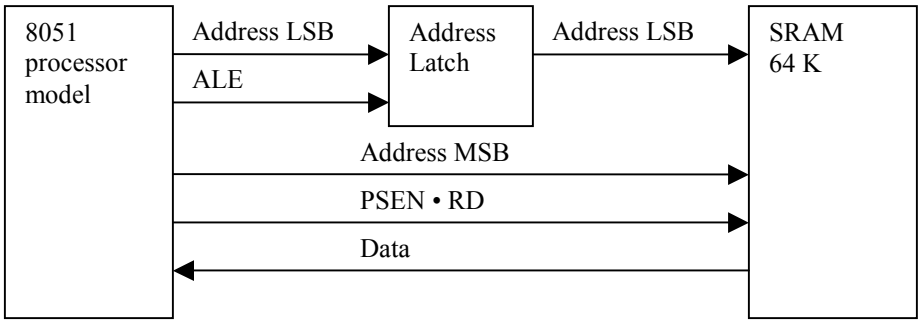

Figure 1: Example system using the VHDL 8051 model

Memory

The memory available to the 8051 model consists of four kbytes of internal ROM, 256 bytes of directly addressable internal memory, and 128 bytes of indirectly addressable internal memory. Additionally, ports P0 and P2 may be used to address up to 64 kbytes of external data memory and 64 kbytes of external program memory. The internal ROM is loaded from an Intel HEX file called code.hex. Most 8051 compilers can produce output in Intel HEX format, so this format is a convenient choice for code memory.

In the example system shown in Figure 1, data and program memory are both mapped to the same memory space. The RAM model nvsram.vhd loads the initial contents of memory from a file called sram.hex, which must also be in Intel HEX format.

If the processor is executing instructions from internal program memory, and the end of program memory is reached at the address 0FFFH, execution continues by reading from external program memory starting at 1000H. This is consistent with the description in the Philips handbook.

If the processor reaches the address FFFFH in program memory, the address rolls over to 0000H and execution continues. When this happens the 8051 model issues a warning that the end of program memory has been reached.

Instruction Set

The VHDL 8051 model supports the entire 8051 instruction set. One instruction cycle lasts for twelve clock cycles. Each instruction takes 1, 2, or 4 instruction cycles to complete. The timing of the instructions is consistent with the 80C51 Technical Description from Philips Semiconductors. All special function registers (SFR) described in that handbook are supported, with the exception of the PD and IDL bits of PCON which control power-down and idle modes, respectively.

The instruction MOV A, ACC is an illegal instruction according to the handbook. When encountered by an 8051 variant, the results are unpredictable. The instruction may act as a NOP on some processors but is not guaranteed to work for all processors. For this reason, the 8051 model issues a warning when this instruction is encountered. The instruction is considered as a NOP and execution continues as normal.

The instruction DIV AB also has undefined behavior when the B register contains 00H. If the B register contains 00H when the instruction DIV AB is executed, the A and B registers remain unchanged and a warning is issued. Divide by zero errors are usually programming bugs, so a warning should help catch this type of bug.

Peripherals and Interrupts

The 8051 model supports both external interrupts, two timer interrupts, and a serial interrupt. Two levels of interrupt priority are supported. Interrupts of the same priority level also have an implied priority in the descending order IE0 (external interrupt 0), TF0 (timer 0 overflow), IE1 (external interrupt 1), TF1 (timer 1 overflow), and RI or TI (serial receive or transmit interrupt). This priority structure complies with the 8051 Technical Description.

Timer 0 and timer 1 are supported in all four timer modes. The serial port also supports all four serial modes. The timers can be controlled through the TMOD and TCON SFRs, while the serial UART is controlled through SMOD, PCON, and possibly the timer 1 overflow rate.

External Interface

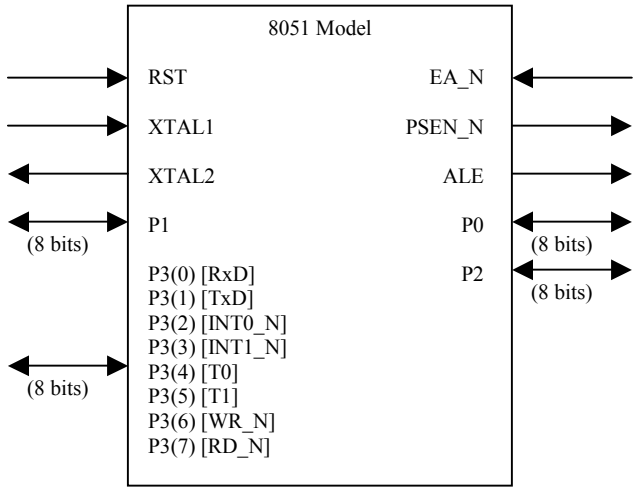

Figure 2: VHDL 8051 model external interface

The external interface to the VHDL 8051 model, shown in Figure 2, looks much like an actual 8051 microcontroller without the power and ground inputs. The following paragraphs briefly describe how each signal on the interface should be used.

RST should normally be driven low. To reset the processor, RST should be held high until the current instruction completes. In the worst case, the setup time for RST needs to be 48 clock cycles. There is no minimum hold time. After reset goes low, the processor will resume execution of code at address 0000H between 8 and 19 clock cycles later.

An external clock should be applied to XTAL1. XTAL2 is not functional; the model contains no on-chip oscillator. The XTAL2 line is present on the interface for completeness, in case the oscillator is added to the model at a later time.

The EA_N (external access enable) input is active low. If this pin is driven low, code will be read from external memory by using P0 and P2. If this pin is driven high, code will be read from the internal ROM contained in the file code.hex. The value of EA_N is not latched when the processor is reset, so the input to EA_N should not be changed while the processor is running.

ALE (address latch enable) is active high and should be connected to the clock input of the external P0 address latch if external program or data memory is to be used. This signal is automatically generated by the 8051 when appropriate, so the programmer need not (and cannot) toggle this line manually.

PSEN_N (program memory strobe enable) is active low and signals that the processor is ready to read a byte from external program memory through P0. This signal is also automatically generated by the 8051.

Four eight-bit input/output ports are also available. P0, P2, and P3 all have alternate functions. P0 is used for the low address byte and P2 is used for the high address byte when external program or data memory is in use. P0 is also used to input data from external program memory, or to input and output data from external data memory. Thus P0 must be connected to an external eight-bit address latch, with ALE used to clock the latch.

Each pin of P3 has its own special function as labeled in Figure 2. RxD is used for receiving serial data, and TxD is used for transmitting serial data. INT0_N and INT1_N are the two active low external interrupts. T0 and T1 are the counter inputs to timer/counter 0 and timer/counter 1, respectively. WR_N is an active low signal that signifies that the processor is writing a byte to external data memory through P0. RD_N is an active low signal asserted when the processor is ready to read a byte from external data memory through P0.

VHDL 8051 Model Internal Implementation

Core of Processor

Several processes control normal code fetches and execution in the 8051 model. The process get_pmem handles the fetches from internal or external code memory, as appropriate. The process main handles decoding of opcodes, execution of instructions, and interrupt processing. A third process called oscillator divides the clock input from XTAL1 into the six internal states of the 8051. In addition, several minor processes handles the resolution of signals driven from multiple processes, such as P0 and P2. Figure 3 shows the portion of the 8051 model necessary for code fetches and program execution.

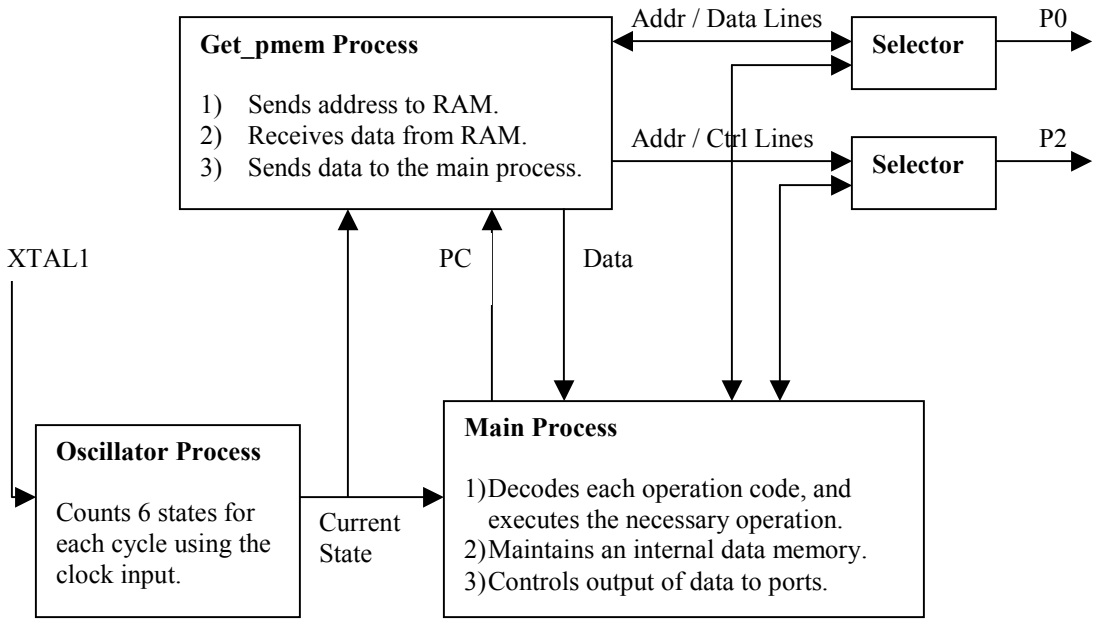

Figure 3: Behavior of 8051 model core

The process get_pmem accesses external memory when the EA_N input is low, or when EA_N is high and the program counter (PC) is greater than 0FFFH. External memory is accessed by putting the high byte of the address on P2, and the low bytes of the address on P0. Then after address latch enable (ALE) goes low, P0 is no longer driven by the 8051. PSEN_N is driven low and the next byte is read from external program memory through P0.

If EA_N is high, the internal ROM of the 8051 is used. When get_pmem is reading from internal code memory, the HEX file specified by the generic constant program_filename (by default set to "code.hex") is used for program memory.

Process main controls the majority of the functions of the 8051 model. If the processor has just been reset, main assigns the reset values to the special function registers. Main handles the calls to interrupt vectors when an interrupt condition occurs, and recognizes two levels of priority among interrupts. Process main also decodes and executes the current opcode.

Timers and UART

The 8051 microcontroller also has two built-in timer/counters and a serial port. The serial port was implemented in a separate UART component, contained in the file uart.vhd. Depending on the operating mode of the UART, the clock for the UART is either the timer 1 overflow, or a signal p2clk, which essentially is XTAL1 divided in half.

Figure 4 shows the interaction of the peripherals with process main. The two timers were implemented in three separate processes. Process timer1 handles timer 1 in all four timer modes. Process timer 0 handles timer 0 in modes 0, 1, and 2, and also handles TL0 in mode 3. Mode 3 for timer 0 is a special case, since TL0 is used as an ordinary 8-bit timer/counter, and TH0 is an independently running 8-bit counter. TH0 hijacks the timer 1 overflow flag TF1 for its overflow when timer 0 is in mode 3. To handle this special case, process timer0_mode3 controls TH0 and TF1 only when timer 0 is in mode 3.

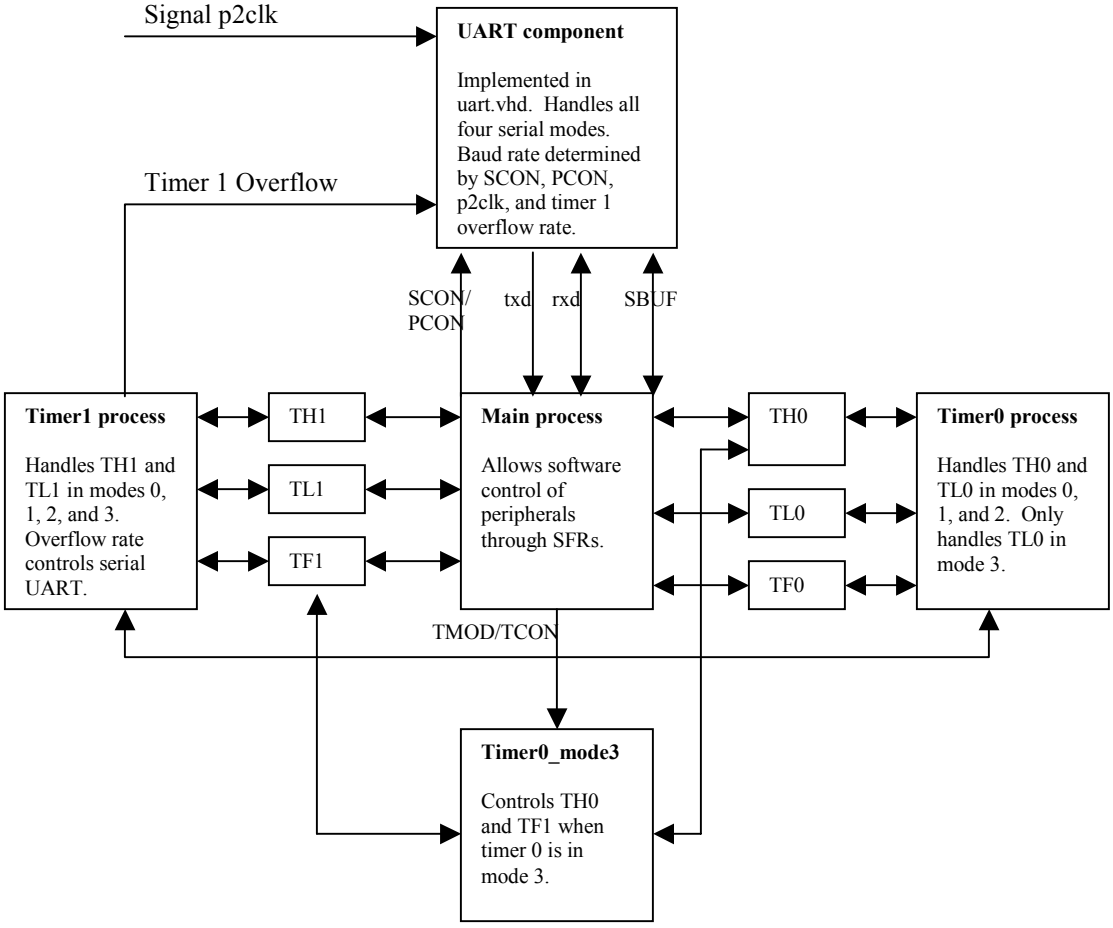

Figure 4: Behavior of 8051 model peripherals

The remaining processes are very simple. They resolve signals driven by multiple processes. For example, TH0 is controlled by main, timer0, and timer0_mode3. One resolution process resolves the inputs from these three processes to assign to TH0.

Testing of the VHDL 8051 Model

Testing Overview

Testing of the 8051 model was done using a set of assembly programs that tested the entire 8051 instruction set. Each assembly program had an associated VHDL testbench that verified the correct operation of the program. The µVision 8051 assembler from Keil Software and the Asm51 assembler from Tasking Software were used to assemble the test programs. Modelsim from Mentor Graphics was used to simulate the 8051 model and the testbenches.

White Box Versus Black Box Testing

Black box and white box testing are two methods of testing a piece of software or hardware. In black box testing, the internal details of the component to be tested are ignored. Inputs are given, and the output is examined to verify that the component is functionally correct. This type of testing may be useful when testing different implementations of the same component, since the same set of tests will be valid for all implementations.

White box testing tests the internal portion of the component. The internal architecture of the model must be well understood for white box testing to be effective. Access to the internal details of a component can make it easier to find errors, but may tie the testing methodology to one particular implementation.

White box testing was chosen as the method for testing the VHDL 8051 model. Specifically, the values in internal memory were examined during the execution of the test programs. But a VHDL testbench can only use black box testing. There is no way to access internal signals from outside of the model. To get around this limitation of VHDL, the model was modified to add extra output ports. All of directly and indirectly addressable internal memory was written out to these extra ports. It is important to note that only outputs were added. No extra inputs were added, and no internal signals were modified in this debugging version of the 8051 model. This ensured that the internal behavior of the model would not be affected by these modifications.

All the problems found with the 8051 model are shown in Appendix A. These bugs have all been corrected in the latest version of the 8051. See Appendix B for an example test program and VHDL testbench. The only remaining known problem is related to the interrupt system. Certain conditions should delay an interrupt being serviced. If the current instruction is a RETI or a write to the IE or IP registers when the interrupt occurs, then one more instruction must be executed before an interrupt is serviced. In the VHDL 8051 model, this is not the case. Any time an interrupt occurs, if an interrupt of equal or higher priority is not already in progress during the polling cycle, the interrupt will be serviced.

Appendix A: Problems Discovered Through Testing the 8051 Model

The following bugs, missing features, and wishlist items have all been corrected in the mc8051.vhd model.

Bugs
- Incorrect interrupt priorities
- Pins for external interrupt 0 and 1 reversed
- DIV AB instruction causes run-time error
- Assignment to PSW can change parity bit (parity bit not hard-wired to accumulator parity)
- POP SP does not decrement SP before loading SP with a value
- MOVC A, @A+PC instruction is off by one on source address
- MUL AB does not clear B register when result less than 256
- SUBB instruction does not correctly set bits of PSW
- Several SFR's not initialized properly upon reset

Unimplemented features
- PCON unimplemented
- Timer-related SFR's unimplemented
- Timer 0 unimplemented
- Timer 1 modes 0, 1, and 3 unimplemented

Additional wishlist items
- MOV A, ACC illegal instruction gives no warning
- DIV AB gives no warning when B is 0
- No warning is issued when memory rolls over from FFFFH to 0000H

Appendix B: Sample Test Program

In the following example code excerpts, the SUBB instruction is tested. The assembly program in Figure 5 performs several subtractions. Each time SUBB is executed the overflow (OV), carry (C), and auxiliary carry (AC) flags should be set appropriately in the program status word (PSW). The VHDL testbench in Figure 6 checks that the result of the subtraction is correct and the PSW is set correctly. If not, a warning is issued and testing continues until the end of the test program. These excerpts only test the SUBB instruction in its immediate form. In the actual test programs, the SUBB instruction was also tested using direct and indirect addressing and all eight register forms of the instruction using all four register banks.

Figure 5: subb.a51 excerpt

```
; SUBB A, #immediate
      CLR C
      MOV A, #00H
      SUBB A, #00H ; 0

      CLR C
      MOV A, #15H
      SUBB A, #03H ; no C no AC no OV

      SETB C
      MOV A, #1FH
      SUBB A, #0FH ; AC

      CLR C
      MOV A, #25H
      SUBB A, #1FH ; AC

      SETB C
      MOV A, #42H
      SUBB A, #0E1H ; C

      CLR C
      MOV A, #00H
      SUBB A, #0FFH ; AC + C

      SETB C
      MOV A, #00H
      SUBB A, #0FFH ; AC + C

      SETB C
      MOV A, #40H
      SUBB A, #80H ; OV

      CLR C
      MOV A, #80H
      SUBB A, #40H ; OV + C
```

Figure 6: subb.vhdl excerpt

```
    -- Verify instructions
    VERIFY: process is
        variable t1, t2: unsigned(7 downto 0);

        -- Check the test set
        procedure subbchk (delay: in integer) is
            procedure chk (val: in integer; c, ac, ov: in std_logic :=
'0') is
            begin
                iwait(delay);
                t1 := to_unsigned(val, 8);
                t2 := unsigned(acc);
                assert to_01(t1) = to_01(t2)
                    report "Acc should be "
                           & integer'image(val)
                           & " but is "
                           & integer'image(to_integer(t2));
                assert to_X01(psw(7)) = c
                    report "C incorrect!";
                assert to_X01(psw(6)) = ac
                    report "AC incorrect!";
                assert to_X01(psw(2)) = ov
                    report "OV incorrect!";
            end procedure chk;
        begin
            chk(0);
            chk(16#12#);
            chk(val => 16#0F#, ac => '1');
            chk(val => 16#06#, ac => '1');
            chk(val => 16#60#, c => '1');
            chk(val => 16#01#, c => '1', ac => '1');
            chk(val => 16#00#, c => '1', ac => '1');
            chk(val => 16#BF#, ov => '1');
            chk(val => 16#40#, ov => '1', c => '1');
        end procedure subbchk;

    begin
        -- Wait for initial instruction cycle
        iwait(1);

        -- SUBB A, #immediate
        subbchk(3);

        wait;
    end process VERIFY;
```

References

Philips Semiconductors. 80C51-Based 8-Bit Microcontrollers.

Ashenden, Peter J. The Designer's Guide to VHDL. San Francisco: Morgan Kaufmann, 1996.

Mayer, Michael R. "Hardware and Software Cosimulation of an Embedded Microcontroller System". January 1998.