SOFTWARE

Quite frankly, ShBoom was designed by a Forth programmer who intends to program it in Forth. Forth provides maximum control of the hardware to the programmer, which results in efficient programs. ShBoom supports this.

To reach a wider range of potential users, every effort has been made to assure ShBoom suitability for C. This includes byte addressing, on-chip registers, stack frames and floating-point arithmetic. Consideration is being given to signed byte and halfword fetch/store.

Russell Fish has persuaded Richard Stahlman to provide a GNU C compiler for ShBoom. Any C benchmarks must await the completion of this. Forth benchmarks will become available as development cards get used.

The ability of a compiler to utilize stack (EMPTY , FILL) and queue (next , skip) optimize instructions remains to be seen. They have been included to permit the programmer to reduce the impact of DRAM memory delays. Task switching and stack overflow are being explored with the prototype chips to assure the instructions to minimize their cost are available.

The ability of Forth to facilitate hardware debugging is widely recognized. This was essential in validating the prototype chips. The high speed of ShBoom will be invaluable in testing the various hardware that it will control.

TIMING

Central to the custom implementation is a high-speed on-chip clock. This is a simple ring-oscillator suggested by Russell Fish. It has several advantages over an external clock.

First, it can run extremely fast. A 7-inverter ring with .3ns gate delay has a 4.2ns period (250 Mhz). Running such a clock off-chip is not practical.

The purpose of this clock is to synchronize the latching of registers. This controls both the CPU and IO processors and their interaction through the memory interface. It is expected that 14 (or 18) gate delays are adequate for instruction decode and execution. The exact number will be conservatively chosen when layout and simulation are complete.

Second, an on-chip clock tracks process variations. So long as process (and temperature and voltage) varies consistently across the chip, the clock will run fast (or slow) together with the delays it is timing. Thus each chip will run at its personal maximum performance, without derating to conform to an external, absolute time reference.

Third, an on-chip clock reduces parts count. It might even save a pin. However, it introduces the need to synchronize with external events. ShBoom in unique in having an IO processor to deal with this. It has a WAIT instruction that will delay until an event (crystal clock tick) occurs. Thus IO can be synchronized while CPU runs at full speed.

The advantage of a fast clock is most apparent in the CPU. Multiply (also divide and floating-point) operations require multiple clock cycles, typically one per bit. Their speed is directly proportional to clock frequency.

Memory access requires a cycle to determine whether a RAS cycle is necessary. A shorter cycle means less overhead on every access. The duration of each DRAM control signal is programmed at power-up, according to manufacturer's specifications and processor speed. Since ShBoom and its memory share the same temperature and voltage environment, their speed will tend to track as these vary, and timing margins can be smaller.

However, this fast clock does require a careful test strategy. Both wafer and packaged test equipment run far more slowly. The OKI prototypes were tested at 1 Mhz (and functioned at 50 Mhz). At power-up control signals will default to their slowest timing, but this may be 50 Mhz and variable (by a factor of 2).

The best strategy is to connect the wafer probes to a socket on a test card, and allow ShBoom to boot up and run a self-test. This can provide not only a go/no go decision, but measure the speed against a time reference. If this is not feasible, a test mode with external clock input can be provided. Possibly on a pad that is not brought out to a pin. A couple thousand test vectors provide good error coverage. But not as good as can be done with self-test.

TIMING DIAGRAMS

One of the features of ShBoom is the ability to tailor the memory interface. The prototype has 2 access modes: 8 and 32-bit. 8-bit is used to access ROM, latches and buffers. 32-bit is used for DRAM, bus interface and 16-bit data.

With 32-bit data transfers, the address and data are multiplexed on a 32-bit bus. Bits 0-10 are used for input during address time; bits 11-19 for multiplexed RAS/CAS address; bits 20-30 for device and memory select; bit 31 is 0 to indicate 32-bit mode. The DRAM control signals RAS, CAS, OE and WE are generated by ShBoom with the timing suitable for the particular chips used. The prototype was designed for OKI 100ns parts; others have been used successfully. The custom chip will have programmable timing with internal clock resolution (5ns). These specify RAS high, CAS high, CAS to OE, CAS to WE, and WE low. Timing diagrams are available on manufacturer's DRAM data sheets. ShBoom uses RAS read and late write, fast page mode read and late write, and CAS before RAS refresh.

With 8-bit data transfers, address and data are not multiplexed. Bits 0-7 are used for data; bit 8 ignored; bits 9-10 are a byte address; bits 11-19 are the CAS address; bits 20-30 for select; bit 31 is 1. Timing on the prototype is a full RAS cycle; this is suitable for 120ns PROM. Custom timing is to be determined.

INTERRUPTS

In considering the problem of real-time control, ShBoom's concept of an IO processor evolved as an alternative to conventional interrupts. The main problem with interrupts is the disruption of timing they introduce at unpredictable times. This leads to their frequently being disabled, an in effect polled.

The custom ShBoom will include conditional IO instructions to transfer data if a signal is set. This will monitor a number of channels at convenient times. It has an interrupt between the IO and CPU, and it has an external interrupt to the CPU. This latter is to handle situations we overlooked and to reassure users about a capability they value. Various forms of soft and hard-vector can be added; in fact, the prototype can be given vectored interrupts with external hardware.

The purpose of interrupts is to handle asynchronous events. These are low frequency but it's not feasible for a busy processor to poll them. It is feasible for a dedicated IO processor to poll them. For example, 960Hz serial input can be polled during horizontal retrace of video output.