Hardware/Software Selected Cycle Solution

John Wilson

Viewlogic/Vantage

1. Introduction

For many system designs where hardware and software are co-dependent , there often exists a design bottleneck where most of the hardware design has to be completed and fabricated before meaningful software verification can begin.

In some sectors of electronic design where competitive markets require that the design cycle has to be completed quickly - mobile phone handsets and image decoders, for instance - having the software ready and substantially debugged as soon as the hardware is ready can give a significant competitive advantage.

This paper describes a methodology that allows practical hardware/software co-simulation to achieve meaningful software verification early in the design cycle.

The technique allows software development and testing to be more concurrent with the hardware development by using simulated models of the hardware design to interact with the software programs.

A high level of performance is achieved by using the simulated description on selected target processor cycles only. Only the important bus cycles are simulated in the hardware simulator.

Most of the software program is executed directly on the host computer.

The techniques described in this paper are the subject of a patent application.

2. Previous Work

Software and hardware designers currently use some aspects of the work described in this paper.

2.1 The Software Developers View of Hardware

Software developers are used to dealing with hardware in terms of abstract function calls. All interaction with the hardware is via these function calls instead of directly reading and writing memory mapped addresses.

The functions take care of the sequences of low-level reads, writes, and interrupts require for each action.

This methodology allows software designers to test their code without the hardware being present. The functions that interact with the hardware are often replaced with alternative functions that emulate possible responses from the hardware.

0-8186-6315-4/94 $04.00

The main problem is that the functions that emulate the hardware seldom provide completely accurate or verified responses. The modelling overhead can be significant.

3. The Hardware Developers View of Software

Most hardware designers who create simulation models of a circuit that incorporate processors can conceive of running programs on the simulated hardware circuit.

There are a number of practical considerations which prevent a reasonable level of verification of the software program.

For commercial processors, an accurate simulation model may not be available for the simulator. Processor design houses carefully guard detailed architecture models to make reverse engineering by competitors as difficult as possible.
Some fully functional 'binary" models of processors are available for simulators[4]. To execute the software program on the simulator requires that the software program be compiled into a binary format and loaded into the simulated circuit memory. Executing the program at this binary level requires the simulator to do many calculations during each processor cycle. The result is that only very limited amounts of code (usually substantially less than a complete program) can be verified.

Another alternative is to replace the full processor model with a bus-functional model. This can emulate each of the processor cycles but has no internal representation of the processor. Some bus-functional models have a meta-language[4] which allows sequences of bus cycles to be executed under the control of a program - but this program has no connections with the software developers program.

Some developers have started to use hardware modellers to interface real-life processors to the simulation model. The object code can execute on the target processor. However, here the speed bottleneck is not in executing the program, but in the communications overhead require to interface the hardware modeller to the simulator program.
This can involve significant communication across a computer network, and can require the hardware modeller to rerun all the previous simulation patterns for each new processor cycle.

4. Method Description

From outside the simulation, bus cycles messages are passed down the pipe into the processor model. The model would then perform the appropriate bus cycles. A 'write' cycle message would cause the pins on the processor simulator model to perform the sequence of a write cycle. A 'read' cycle message would cause the processor simulation model to perform the sequence of a ready cycle, and report back the results that appear on the data bus of the model (Figure 1).

Selected Cycle Simulation (SCS) recognises that processor based circuits can be divided into two distinct functional parts.

The essential circuit comprises the processor and the other parts of the circuit without which the processor could not function. This typically includes memory such as RAM and ROM, and other related logic functions.

The peripheral circuit comprises the rest of the circuitry which makes this design unique from other designs based on the same processor. This may include ASICs and FPGAs which assist the processor in calculations. Or, the processor circuit may act as a 'housekeeper' for a specialised peripheral circuit - taking care of initialising registers and general maintenance tasks while the peripheral circuit performs it's function.

A software program written in a high level language, and with the appropriate compiler available, could conceivably execute on any processor as long as its essential circuitry was in place, and as long as it did not try to access or utilise any peripheral circuits. In fact, such a self-contained program would be useless because there would be no way in which the results of the program

could be communicated to the outside world. However, it does serve to illustrate that the bus cycles that the processor performs in conjunctions with the essential circuits may be irrelevant to the verification of the program.

The software program would also contain 'drivers' to interact with the peripheral hardware. There drivers may contain a sequence of read, write and interrupt cycles which the processor has to perform in conjunction with the peripheral hardware to allow interaction between the program and the peripheral hardware functions.

This method proposes that, within the simulated model of the processor circuit, a detailed model of the processor is not required. Instead, the processor can be modelled as a bus-functional unit, having knowledge of the different processor cycles, but having no internal representation of the processor. The processor model should also implement a bi-directional communications channel outside of the simulation. This channel might be implemented as 'pipes' in the UNIX operating system, for instance.

Controlling the messages into the pipe can be the software program. However, the software program has been prepared in a different way to work with the pipes. Firstly, each 'driver' has been replaced with an equivalent function which passes messages to the pipe and waits for an answer (if appropriate). Secondly, the software program is cross-compiled to run on a host computer with the simulation 'drivers' linked in place of the ultimate 'drivers' (Figure 2).

The net effect is that most of the software program is executed within the host computer. Whenever the hardware driver routines are executed, some interaction with the hardware simulator is initiated. The hardware simulator executes only the bus cycles required by the driver. As soon as the require response is available, the hardware simulator can suspend processing and the software program resumes.

5. Limitations of this Methodology

This technique has some limitations which may restrict it's application or invalidate results obtained. However, in many cases, these limitations can be overcome quite effectively.

5.1 Modelling Limitations

This technique relies on separating the essential circuit from the peripheral circuit. In classical processor circuits it is very easy to obtain the information required to model the bus cycles. The internal structure is not important.

However, processors with integrated peripherals are widely used. This means that part of the peripheral circuit may be contained with the processor model and may have to be modelled. This is far from a trivial task. If conventional models exist for the processor, it is possible to replace the CPU core whilst retaining the peripheral circuitry description.

5.2 Time Dependent Software Functions

Sometimes programmers will use software timing loops to pause the program for expected responses from the peripherals. However, the time sense of the executing software program and the simulator 'slip' with respect to each other. The response may not be available when expected.

One way around this problem is to use some handshaking protocols, where the program keeps checking a status flag to find out if a valid response has arrived yet. Another technique is to expand the bus cycle commands to ensure that the cycles are executed within the simulator - and a number of extra cycles corresponding to the timing loop.

5.3 Different Technique for Handling Interrupts

The best method of handling interrupts seems to differ for different program styles.

One method is to pass an interrupt message, via the pipe, to the software driver function, and let the driver function handle the interrupt. If the interrupt is part of the handshaking protocol of the driver, this may be acceptable (for instance if a register is written, and an interrupt signals that the response is ready to be read).

A second method is to utilise the software facilities in UNIX. An extra pipe can be created to allow interrupt messages to be passed back to the OS. This can cause a UNIX interrupt to be created. The software program can respond to a UNIX interrupt. In many cases, the structure of interrupt driven software make this second method more attractive.

5.4 Performance Management

The time slippage in the hardware simulator make accurate timing measurements of programs executing on the simulated hardware impossible. However, it is quite easy to measure accurately the amount of peripheral read and write cycles.
Combining the with 'code-coverage' statistics which can be generated from the software program can result in a quite accurate 'guesstimate' of program/circuit performance. Complete accuracy, however, can only be obtained from a complete system model, with the commensurate performance penalty in speed of program execution.

5.5 Practical Experiences

A number of test designs, including a full industrial test installation at Siemens in Munchen have been completed.

All the models have been built to run in the Vantage Optium VHDL simulator using the STYX 'C' interface[3]. The STYX interface allows C models to create and respond to events within the simulation. STYX allows X library functions to be linked into the models.

Communications between the model and the software program have used UNIX pipes (fifos). This allows a very simple flow control mechanism between the software and hardware programs.

A 'software developers' kernel has been written. This contains a set of functions to interact with the pipes. It allows the software developer to interact with the hardware simulator at a cycle level (e.g. scs_read() and scs_write().

One benefit of the software developers kernel is to make the code processor-independent; the choice of target processor is irrelevant to the software programmer as long as the bus cycle instructions and the peripheral hardware interface remain constant. Functions for some bus cycles, such as read-modify-write, are relevant to some processors but not to others.

For VHDL simulators, there are specific initialisation requirements which can make it awkward to sequence the initialisation of the UNIX pipes[1]. For this reason a start-up script to create the pipes, initialise the simulator, and initiate pipe communications was implemented. This prevented occasional dead-lock where the simulator was waiting for pipe initialisation, and the software program was waiting for pipe initialisation, and the software program was waiting for the simulator to initialise the pipes.

All the information for the simulator models was taken from freely available data books. The same basic structure has been adopted for each processor's program.

All processor cycles are coded into the number of states required for that operation. A counter is used to track the current state of the cycle. A test is done at each state to establish it is valid t move to the next state. At the end of the cycle, the processor model moves back into an idle mode.

On each clock cycle (or appropriate trigger), a check is done for any high priority system actions such as reset signals or halt signals becoming active. If none are detected, then any current cycles are evaluated and processed. Only when the processor is an idle mode are interrupt signals processed.

When all sources of actions have been exhausted, the processor will check the pipe for the next instruction from the software program.

The only customisation require for each program is to map the high level actions onto the individual ports values at each place in the sequence.

5.6 System Performance

Initial test programs consisted of nested loops of read and write cycles. Subsequent test programs have had a higher proportion of software instructions. In all cases, the dominant performance factor is the speed of the simulator.

The overhead in the communications channel is relatively low. The part of the program executing on the host computer may be executing faster than on the ultimate target system.

Initial test programs consisted of nested loops of read and write cycles. Subsequent test programs have had a higher proportion of software instructions. In all cases, the dominant performance factor is the speed of the simulator.

The overhead in the communications channel is relatively low. The part of the program executing on the host computer may be executing faster than on the ultimate target system.
Initial tests on an HP715 computer, running Vantage vss_4.200_hp, showed speeds of over 107400 read/write cycles per hour. This figure is the elapsed time to run the software program including all interaction with the VHDL simulator (which was running concurrently on the same machine). The simulation model used was a SCS model of an Motorola MC68302[2], some read and write registers, and a gate-level 64x64 multiplier where data was written to the input registers, passed through the multiplier, and the answer was latched into the output register, and read back into the test program via the SCS model and interface..

As the VHDL models of the peripheral hardware became more complex, this speed of the read/write cycles decreased. This was due to the VHDL simulator taking more processor resource to execute the more complex VHDL models. Subsequent tests on application programs have shown that few applications seem to access the peripheral circuitry for more than 5% of the program.

6. Summary

This paper has shown a technique that allows significant software verification to happen concurrently with the hardware design phase of a project. This technique is practical because it builds on established techniques in both software and hardware design.

The software developer is used to testing programs by replacing missing or incomplete parts with emulations. In this case the emulation comes from the interaction with the hardware model simulator.

The hardware developer can test his models under more rigorous conditions by using the software program to apply test patterns in a realistic way. Through VHDL and other hardware description languages, hardware designers are becoming accustomed to the idea of more abstract simulations.

This technique allows for a larger amounts of code to be tested and verified than before - because only essential cycles are executed in the hardware simulator. Most of the program execution is taking place in the host computer processor at a rate comparable (sometimes faster!) than the final target system. Model implementation is relatively simple, and require information in the public domain only.