Flexible Modeling Environment for Embedded Systems Design

Shailesh Sutarwala, Pierre Paulin

Bell-Northern Research Inc.
Ottawa, ONT, Canada

Abstract

The integration of hardware and software is perhaps the most important issue of embedded systems design. The software content of these systems is increasing in complexity which makes the code verification an important issue. Moreover, if the software development is to proceed in parallel with the hardware design, a simulation model representing the hardware behavior is needed. In this paper we describe a strategy for modeling the behavior of instruction set processors. The proposed strategy allows for retargetability and cycle true simulation with practically acceptable speed. The approach allows for very flexible timing annotations so that complex instruction sets with data dependent or addressing mode dependant timing can be modeled. The cycle true behavior allows the model to be embedded into its environment so that the system operation can be verified. The approach is seen as an improvement over a stand alone, hard-coded models since being retargetable, it requires a fraction of the development time.

1 Introduction

A real-time embedded system is a digital system which uses instruction set processor running real-time software to perform some or all of its functions. Wide variety of products ranging from automobiles, to laser printers to sophisticated telecommunication components use embedded systems.

There are several motivations behind rapid emergence of the embedded systems. They provide design flexibility to accommodate late product specification changes. The programmable aspect also allows companies to roll out new and improved versions of products by taking advantage of the existing software base. In competitive industries, such as telecommunications, time to market is a key factor. Embedded systems can potentially enable companies to reduce the design intervals provided proper methodology is used to design the hardware and software components in parallel (Hw/Sw co-design).

Embedded systems are not above real-time constraints. In many cases, the manufacturing costs and time-to-market considerations make the use of the off-the-shelf microprocessors or DSPs (depending on the application) more attractive. In other cases, high speed, low power, or high volume (hence cost sensitive) requirements favor the use of application specific processors.

The integration of hardware and software is perhaps the most important issue of embedded systems design. The software content of these systems is increasing in complexity. The hardware design has benefitted from design automation but the integration of software into embedded systems design still leaves a lot to be desired. The lack of proper design methodology and the tools prevent embedded products to get to market quickly and efficiently.

Considering the task at hand, two important aspects of the problem come to mind. In order to boost the productivity, compilers are needed. There is a significant amount of assembly code being written in embedded systems today [1]. Compilers increase software productivity analogous to that provided by logic synthesis in hardware
design. However, these compilers must produce code which meet the real-time constraints. If a compiler is retargetable, it is an added bonus considering the high development time of the compilers. Another important aspect is the code verification. If the software development is to proceed in parallel with the hardware design, a simulation model representing the hardware behavior is needed. It is this simulation aspect of the embedded systems design, that is considered in more detail in this paper. Section 2 describes the proposed modeling approach. Section 3 presents some experiences of developing models for various processors. There we also discuss how this exercise lead us to some of the improvements in our approach. Section 4 provides some conclusion and future work.

2 Proposed modeling strategy

2.1 Basic philosophy

A variety of commercial microprocessors and DSP processors, as well as custom made application specific processors are used in BNR [1]. Each time a new processor is designed, a simulation model needs to be developed from scratch. This requires significant amount of time and in many cases a learning curve for the Hardware Description Language. This prompted us to conclude that ideally, the simulation environment should be easily retargetable.

One of the main purpose of the processor model is to verify the software in its system environment. That is, the model needs to interact with other system components which may be hardware. Thus a synchronization mechanism is needed which is provided by a cycle-true model. However, the model does not need to have the details of the RTL or a gate-level which slows down the simulation. Therefore, we propose a cycle true, behavioral model.

In lieu of above considerations, Insulin, an instruction set simulation environment was developed. In the proposed approach, we define a generic set of simple, atomic instructions. This set is referred to as the generic instructions. An efficient VHDL model is written to execute this instruction set. The behavior of a target instruction set can then be described using these generic instructions. From this specification, a cross assembler is automatically generated to convert the target assembly code into the generic assembly code which can be simulated. Thus the underlying model does not change, accept for the parameters such as registers, memory size, stack size etc. - which are automatically communicated to the model using “generics” feature of VHDL [2]. An automatically reconfigured graphical interface allows the user to interact with the model. Thus, the approach allows full retargetability to adapt to a target processor (instruction set), while keeping the underlying model and the interfaces intact. Some details of the Insulin approach are described in [3]. It is also a part of the FlexWare firmware development environment for embedded processors [1].

2.2 Flexible timing

As mentioned earlier, it is very important that the model of the instruction set can be embedded in its environment to verify the operation of the system. For this reason, a mere functional model is not sufficient a cycle true, behavioral model is needed. For instance, in most microprocessors, instructions are executed in sequence (no instruction level parallelism), however, each instruction may take different amount of time. Some instructions may take one cycle, some may take multiple cycles, and some may even have varying timing requirements. For instance, a move instruction may take different number of cycles depending on the addressing mode used. A division instruction may take a number of cycles which is function of the bit-width of the divisor. On the other hand, in
many DSP processors, several instructions may be executed in parallel in one cycle.

In order to accommodate the flexible timing requirements, the Insulin approach allows each instruction behavior to be annotated with delays specified by timing expressions. These expressions are evaluated at the simulation time by a stack machine built-in to the generic VHDL model. For VLIW processors such as DSPs, the parallelism is handled by specifying the number of fields in the micro-instructions. Based on this number, the generic model is automatically configured at the simulation-time to include the appropriate number of execution engines. Each of the engine is capable of executing any generic instruction. Some examples are provided in Section 3 to illustrate how timing information can be specified.

2.3 The generic model

The basic organization of the generic model is shown in Figure 1. The number of registers, the size of data memory and number of parallel execution units are configured at the simulation time. The model then proceeds with the fetching of each instruction. The built-in stack machine is used to evaluate the operands. Then, the decoded instruction and the operands are passed to one of the possibly many execution engines. At this stage, if there are any timing annotation (delay) associated with the instruction, it is evaluated using the stack machine. Based on the value of this evaluation, the main unit waits for the appropriate number of cycles before updating the results of the current instruction to the data memory and/or registers. Then the next instruction is submitted to the execution unit. All the operations are synchronized using one clock. The communication with the external

---

**Figure 1. The Generic Model Organization**
models is done using memory mapped I/O.

The generic model also allows the usual user interactions such as observing or downloading the register or memory values, setting breakpoints etc. Note however, that the generic nature of the model is completely transparent to the user [3].

### 3.1 Instruction behavior and timing

As mentioned earlier, the behavior of each instruction in the target instruction set is specified in terms of the generic instructions (there are about 35 generic instructions). A simplest form of an instruction is one which executes in one cycle and has a one-to-one mapping to a generic instruction. Figure 2 depicts some representative instructions from different processors.

The first instruction is a multiply-accumulate instruction from an internal BNR DSP processor. The instruction multiplies its two operands and adds the result to the accumulator ACC. Since the processor has a 16x4 bit multiplier, the full 16x16 bit MACC instruction takes 4 cycles. This behavior is specified by using two generic instructions "g_mult" and "g_add", and a delay of 4 cycles (rather than the default 1 cycle) before the accumulator is updated.

### Figure 2. Instruction Descriptions with Timing

<table>
<thead>
<tr>
<th>Multiply Accumulate</th>
<th>Shift Left</th>
<th>Jump if Z = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACC &lt;x&gt;, &lt;y&gt; cycles: 4 cycles</td>
<td>SHL &lt;x&gt;, &lt;nshifts&gt; cycles: 7+1 per shift</td>
<td>JE &lt;label&gt; cycles: 4 if jump not taken 8 if jump taken</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>g_mult</th>
<th>g shl</th>
<th>g_eq</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>[7 + &lt;nshifts&gt;]</td>
<td>T F</td>
</tr>
<tr>
<td>g_add</td>
<td>[4]</td>
<td>[8]</td>
</tr>
</tbody>
</table>

3 Modeling experience

Several models have been developed using the Insulin approach. These include both the BNR in-house and commercial DSP processors and micro controllers. In this section we discuss the experience of modelling such variety of processors. First we provide some examples of instructions with the varying timing requirements, and demonstrate how easy it is to specify them using the proposed method. Finally we discuss some of the difficulties we faced in modelling - which lead us to include other features.
The second instruction is a logical shift left from Intel 80C196 microprocessor [4]. The behavior is simple and has a one to one mapping to the generic instruction “\texttt{g.shl}”. However, the timing is data dependent. It takes 7 cycles minimum, plus 1 cycle per shift. Therefore, a shift by 4 will require 11 cycles whereas a shift by 8 will take 15 cycles. This requirement can be easily annotated using the delay expression “7 + \texttt{<nshifts>}” as shown in Figure 2.

The third instruction poses yet another challenge. This is a jump-if-zero instruction from Intel 80C196 microprocessor. This can also be specified with ease. As shown in Figure 2, the Z bit of the CC register (bit 0 of the condition code register) is accessed using the read-bit-range instruction “\texttt{g.rbit}”. This bit is compared to the number 1 (#1) using the generic instruction “\texttt{g.eq}”. If the comparison yields false, the program counter PC is not modified and the instruction is completed after the delay of 4 cycles. However, if the comparison yields true, the jump address \texttt{<label>} is moved to the program counter after the delay of 8 cycles.

In some processors including the SGS Thomson’s ST7291 Micro Control Unit [5], instruction timing is dependant on the addressing mode used. We have been able to model this successfully. The above examples should give a good indication of how the complex behaviors and timing is specified using simple generic \textit{“behaviors”} and the powerful delay annotations.

3.2 Other practical issues

One of the caveats of the generic approaches is that the real-life problems never quite fit the approach exactly. We encountered a few such glitches. Here we discuss what problems we faced and how we dealt with them.

While modeling, we quickly discovered that some snippet of behavior was repeated in many instructions. For example, in most processors, after any arithmetic instruction, the condition codes such as Zero, Carry, Overflow etc. are updated. Hence, the behavior describing how to set the condition codes is repeated over all the arithmetic instructions. To alleviate this inconvenience, the obvious solution we found was to allow encapsulation of behavior snippets. We call them Macros.

Consider for instance, that in the MACC instruction described in Figure 2, the condition code register (CC) bit 5 is set if the result is zero and the bit 8 is set if the result is negative. We can write a common behavior snippet (macro) to set the two condition codes and rewrite the behavior for MACC instruction as shown in Figure 3. The same behavior may be referenced by another instruction using the macro “\texttt{update.cc}”. The usual input/output parameter substitution is allowed. In Figure 3, the value of the accumulator ACC, and the value of the input parameter “\texttt{inp_1}” is the same. There are no restrictions on the number and size of macros. However, no delays are allowed within macros. This is deemed as a desired restriction.

Another difficulty of similar nature was encountered. For instance, in a SGS-Thomson DSP processor [6], the termination of a loop is determined by a program label. This is achieved by the management of three loop registers - the loop start address register LS, the loop end address register LE, and the loop count register LC. While the value of LC is not equal to one, the loop controller compares the program counter (PC) content with LE value: when the last instruction of the loop is fetched, the PC is loaded from LS. When LC is 1, the last iteration of the loop proceeds. It is completed when PC equals to LE. At this stage, LE is cleared and the loop terminates. This behavior of the loop controller must proceed independent of any instruction.

In order to accommodate the above, the
3.3 Overall impressions

The proposed approach is being used at BNR with some positive results. We have so far modeled various processors (instruction sets) including DSPs and microcontrollers - both internal and commercial. Here we describe some of the findings.

First of all, being strictly behavioral, our approach differs from some of the structural approaches [7], [8], [9]. However, being a
cycle true model it allows the processor to be embedded in its environment. This is important in order to verify the correct operation of the system. Another important requirement is the simulation speed. The Insulin environment can simulate about 2000 generic instructions per second. This is on SPARC 2 using Synopsys interpreted VHDL simulator. The compiled VHDL simulator is expected to enhance the speed further. The speed is mainly attributed to the behavioral nature of the model and the fact that the generic model is written once, efficiently by a person with prior experience.

Perhaps the most important advantage of our approach is the ability to very quickly develop the models of a variety of instruction set processors. This is mainly attributed to the very simple input specification which cuts down the learning curve tremendously. For instance, an in-house processor which took a novice VHDL user more than a month to develop. However, the Insulin specification of the same took a couple of days. The ST7291 commercial microcontroller was modeled in three weeks by a co-op student with no prior background. Typically the commercial processors take longer to model than the internal ones - simply due to the much larger instruction set and the addressing modes available in the commercial ones. It was also found that microcontrollers are in general easier to model using this approach than the DSPs. In the DSPs, it is the behavior which is more complex, whereas in microcontrollers it is the timing which is more complex. In any case the typical model development time savings range from a few person weeks to even a couple of person months! This is extremely valuable considering the push for the shorter time-to-market.

4 Conclusion and future work

A strategy has been described for modeling the behavior of instruction set processors. The proposed strategy allows for retargetability and cycle true simulation with practically acceptable speed. The cycle true behavior allows the model to be embedded into its environment so that the system operation can be verified. The Insulin models are used to verify the retargetable compilers being developed here at BNR. The approach is seen as an improvement over a stand alone, hard-coded VHDL or Verilog model since it requires a fraction of the development time.

The future efforts will likely include allowing interrupt capability, management of multiple data memories, and providing bit-true simulation.

5 References