Modeling micro-controller peripherals for high-level co-simulation and synthesis

Harry Hsieh* Luciano Lavagno† Claudio Passerone‡ Claudio Sansoè§
Alberto Sangiovanni-Vincentelli¶

Abstract

Mapping a behavior on an embedded system involves hardware-software partitioning and assignment of software and hardware tasks to different components. In particular, software tasks in embedded controllers are mostly assigned to a micro-controller. However, some micro-controller peripherals are implemented with partly programmable components that can be regarded as very simple co-processors with limited instruction sets and capabilities. Embedded system designers are used to mapping some simple software tasks onto these simple co-processors, obtaining overall performances that can be orders of magnitude superior to the ones obtained mapping all software tasks to the micro-controller itself. In this paper, we propose a methodology to specify, simulate, and partition tasks that can be implemented on programmable micro-controller peripherals such as Timing Processing Units (TPUs). Following our general philosophy, we let the designer propose a partition, and we provide an environment

- to efficiently simulate and evaluate a particular implementation choice
- to automate downstream synthesis for software, hardware, as well as peripheral programming routines.

1 Introduction

Embedded systems are implemented as a mix of hardware and software components. Software is generally used to provide the flexibility required to re-use the system in a variety of situations. Hardware is used to provide the performance required to meet real-time constraints. Hardware components can be either Application Specific Integrated Circuits, or (more often in low-end products) peripheral devices integrated on a micro-controller. The functionality of these peripherals can be, for example, measuring time (e.g. counters, timer units, ...) or communication (e.g. serial and parallel I/O controllers, DMAs, ...). Even though those functions could theoretically be performed directly in software, often such an implementation would have unacceptable performance. Consider, for example, the simple task of generating a square wave with a given duty cycle on an output pin of the micro-controller (such Pulse-Width Modulation is often used in practice to control actuators). It could be implemented via software by scheduling a simple task, that checks the current time and raises or lowers the output via a port I/O operation, as frequently as required by the period and precision specification for the waveform. This implementation would be acceptable in general only if the context switching overhead and the task execution time are very small compared to the period and precision requirements. Quite often, this is not the case, and a small peripheral, consisting of a counter, a duty-cycle register and a comparator is used instead[1]. Recently, micro-controller peripherals have been implemented as very simple co-processors with limited programming capabilities to favor design re-use and flexibility[2]. Hence, a number of simple tasks can now be mapped onto these processors to reduce the load of the main micro-controller unit. This implies that the partitioning task is more complex: two or more processors can perform software tasks with very different cost-performance trade-off.

*University of California, Berkeley, CA. Supported by SRC Contract 96-DC-324
†Politecnico di Torino, Italy
‡Politecnico di Torino, Italy
§Politecnico di Torino, Italy
¶University of California, Berkeley, CA

[1] Such functionality is common to the timing units of almost all modern micro-controllers.

[2] The most recent micro-controller offering from Motorola includes a Timing Processing Unit that is effectively a simple fully programmable RISC processor.

A commonly used design methodology consists of the following steps:

1. Specify the functionality requirements of the embedded system in a uniform fashion[3],
2. Evaluate the performance and the cost of several implementation alternatives, trading off hardware and software, choosing the target micro-controller, defining the real-time scheduling strategy.
3. Validate the specification against functional requirements, e.g., by co-simulating hardware and software components together at various levels of abstraction.
4. Synthesize, under the designer's guidance, the hardware, the software, and the interfaces[4].

The literature on hardware-software co-design has not addressed thus far in its generality the problem of mapping software tasks to micro-controller peripherals. In this paper we propose a method to partially alleviate the problem, by specifying, simulating and partitioning using a *high-level model* also functions that can be implemented on micro-controller peripherals. The implementation of these functions is done with library functions that appropriately program the peripheral, but it can also be done by synthesizing from the high-level model, thus retaining (at least partially) the hardware/software trade-off capabilities. We also provide a "behavioral" model for simulating the peripherals when a more precise but more costly RTL model is not required for the simulation pattern at hand.

The approach of [5] and [8] to synthesize interfaces between hardware and software is complementary to our work: it proposes a scheduling-based algorithm for interface partitioning, based on a graph model derived from a formalized timing diagram that describes the peripheral interface protocol. The problem of automatically selecting from a library the best device to perform a given function has been tackled in [**?**], with a specialized solution based on syntactic matching, and also complements nicely our solution. In fact, our method helps the designer in selecting which function should be implemented on a peripheral, based on performance and cost analysis, while the algorithm of [**?**] can be used to find the peripheral in the library.

2 Polis Co-design methodology

In our co-design methodology ([4]) the designer specifies the system functionality using extended asynchronous Finite State Machines called Co-design Finite State Machines (CFSMs). CFSMs can be derived automatically from a range of high level languages such as ESTEREL [2]. A CFSM is a Finite State Machine operating on a set of integer variables with arithmetic, relational and logical operators. CFSMs communicate asynchronously via events. An event is a uni-directional buffered communication primitive. The sender is allowed to continue its activity after emitting an event (that can carry an integer value), and the receiver can detect it and use it later. A single buffer location is used in the current implementation of the co-design environment, which means that events can be "lost" if they are produced faster than they are consumed. Some other mechanisms, like a handshaking protocol, or an appropriate real-time scheduling policy must be used in order to ensure that such overwriting cannot occur for *critical* events.

The designer must manually assign CFSMs onto computational elements (hardware, software, peripherals). We provide an integrated environment for rapid-prototyping, fast co-simulation, as well as formal verification to help the designer in that task. Once a particular implementation choice is deemed acceptable both from the functional and timing standpoints, automatic synthesis takes over to produce C code implementing the CFSMs mapped to software and interfacing to the peripherals, as well as an HDL specification of the CFSMs mapped to hardware.

One key point of our methodology is the ability to perform fast hardware-software co-simulation. Co-simulation is based on software timing estimates for high level constructs in the synthesized C-code. Clock cycles are accumulated during simulation to synchronize the software with the hardware and the environment. We are able to simulate millions of cycles per second for SW blocks and thousands of cycles per second for HW blocks.

[3] Even though a single specification *language* may not be usable in all cases, nor for all components of a system, at least a common *semantic model* underlying the various languages used for a specification is sufficient to meet this objective.

[4] Often this requires using the services provided by some customized or standard Real-Time Operating System to coordinate the software tasks, allowing them to communicate with each other, with the hardware, and with the environment.

3 Modeling Micro-controller Peripherals

In this section we propose a methodology to model functions that can be implemented on a peripheral, by modeling them with *library CFSMs*. Library CFSMs are usually specific to a family of micro-controllers, and they have a certain degree of programmability to make them suitable for different functions. They are described using the *same* specification model as other parts of the design, by using languages with underlying CFSM semantics. Note that micro-controller peripherals are subject to even more standardization than micro-controllers, because often a manufacturer uses, for example, the same timing unit or serial I/O controller for several different micro-controllers, even with different instruction sets.

During design and evaluation time, the designer picks one item from this library, customizes it and then uses it as if it were hand-designed. In this way the performance evaluation, co-simulation ([7]) and formal verification ([1]) tools available in our co-design methodology are also available in this case to assist in partitioning and validating the design with peripherals.

In addition, we provide a behavioral model for some library CFSMs, in order to speed up co-simulation at the cost of simulation accuracy. A saving of about two orders of magnitude can be achieved at the cost of cycle-by-cycle accuracy of behavior (which often is not required for a first cut performance evaluation). The choice between behavioral, hardware (custom or peripheral) or software implementation is done at simulation time, by simply changing the value of a parameter associated with the library CFSM.

Peripheral function customization is typically done by assigning a value to a parameter of the library CFSM. For example, a constant may define the period of a watchdog timer, or which one of several identical functional units (sharing the same RTL model) is used to implement a specific function.

At synthesis time, the co-design environment extracts the appropriate drivers from the library, that together with the customization parameters implements the required function on the chosen peripheral. For example, the value of one of the parameters (possibly scaled by an appropriate factor) is written into a peripheral programming register. The RTOS synthesis task is informed that this specific CFSM will be implemented using semi-customizable hardware, and modifies the I/O driver generation task accordingly.

This methodology offers the following advantages, compared with traditional design methods:

1. The uniform specification model enables efficient validation and performance evaluation, with a trade-off between simulation speed and accuracy.

2. Some degree of flexibility in the final implementation choice is retained, because one can easily map the library item to custom hardware or software.

3. This approach permits to design customized micro-controllers with application-specific peripherals. The application is simulated and prototyped using hardware synthesis and FPGA. When the ASIC including the CPU core and the peripheral itself becomes available, the co-synthesis system is then instructed to use the peripheral routine instead of the synthesized hardware. The new interfacing mechanism is also synthesized automatically.

4 Implementation and Case Study

We have implemented such a library for the 68hc11 family of micro-controllers from Motorola ([6]). It includes:

- The timer unit, implementing input capture and output compare functions, that measure time between input and output events using a 16 bit free running counter.

- The A/D converter[5].

- The PWM generators present on a specific family member targeted for automotive applications.

Each function of the timer unit and of the PWM generator is described both at the *behavioral* level, using the simulator timing functions to implement time, and at the *Register Transfer* level, using a cycle-accurate model of the hardware. The latter is also used for hardware and software synthesis, if the peripheral is not used. C routines are

[5] Of course, no custom hardware or software synthesis is possible for this peripheral.

used to interface to the peripheral. The precision of the RTL model can be scaled, by dividing the clock. A single simulation parameter controls this scaling without affecting the overall behavior (apart from the loss of precision).

We have specified the functionality of a complete dashboard controller, that uses the 68hc11 peripherals. Note that with the standard co-design methodologies, using only fully programmable processors or hardware, all the tasks implemented by the peripherals could be implemented only as software tasks, thus yielding a less performing solution, or as hardware tasks, thus yielding a higher cost and less flexible solution. Hence the method presented here is required in order to obtain a solution quality comparable with manual design.

The speed of the behavioral simulation was about 260,000 clock cycles per second. The speed of the RTL simulation ranged almost linearly from 2,000 clock cycles per second to 50,000 clock cycles depending on the clock scaling factor, from 1 to 32. The experiments were performed on a 60MHz ULTRAsparc.

At synthesis time, the appropriate I/O drivers are extracted from the library and customized by the co-design tools. We also synthesized a hardware implementation for the PWM generators of the dashboard controller, because they are not available on all members of the 68hc11 family. We analyzed the cost trade-offs of using some small ASIC to implement that function. A hardware implementation, using XILINX FPGAs for rapid prototyping purposes ([3]) required 374 CLBs (with 203 flip-flops) and 60 I/O pads, that would fit on a XILINX 4010 chip.

5 Conclusion

The proposed solution for high-level specification of micro-controller peripherals retains most of the advantages and flexibility of hardware/software co-design (uniform modeling, fast co-simulation, formal verification, flexibility in target implementation, . . .). The limit is that the designer has to decide on whether or not a function is implementable using a particular peripheral, and sometimes such a decision must be made about peripherals that may be only slightly different between different micro-controllers. Further research is still needed to develop mapping techniques from an unbiased specification to *partially programmable* devices.

References

[1] F. Balarin, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli. Formal verification of embedded systems based on CFSM networks. In *Proceedings of the Design Automation Conference*, 1996.

[2] G. Berry, P. Couronné, and G. Gonthier. The synchronous approach to reactive and real-time systems. *IEEE Proceedings*, 79, September 1991.

[3] S. Cardelli, M. Chiodo, P. Giusto, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli. Rapid-prototyping of embedded systems via reprogrammable devices. In *7th IEEE International Workshop on Rapid System Prototyping*, 1996.

[4] M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli. Hardware/software codesign of embedded systems. *IEEE Micro*, 14(4):26–36, August 1994.

[5] P. Chou, E.A. Walkup, and G. Borriello. Scheduling for reactive real-time systems. *IEEE Micro*, 14(4):37–47, August 1994.

[6] Motorola Inc. *M68HC11, Reference Manual*, 1991.

[7] C. Passerone, M. Chiodo, W. Gosti, L. Lavagno, and A. Sangiovanni-Vincentelli. Evaluation of trade-offs in the design of embedded systems via co-simulation. Technical Report UCB/ERL M96/12, U.C. Berkeley, 1996.

[8] E. Walkup and G. Borriello. Automatic synthesis of device drivers for hardware-software codesign. In *Proceedings of the International Workshop on Hardware-Software Codesign*, October 1993.