Using HW/SW modeling to optimize embedded control systems

YVES MATHYS  
rxsb10@email.sps.mot.com

ANDRE CHATELAIN  
r39520@email.sps.mot.com
Advance Vehicle Systems Division, Motorola SPS, 207, rte de Ferney, CH-1228 Gd-Saconnex, Geneva

Key words:  
HW/SW codesign, control system, profiling, Esterel

Abstract:  
The today semiconductor technology allows the building of very complex embedded systems on a chip. Building such systems requires optimal HW/SW partitioning and early system validation. We have developed a simulation based HW/SW design framework for control applications. The framework assists the designer to quickly explore HW/SW partitions by providing accurate architectural profiling. We present a case study of engine ignition functions where we reduce CPU loading by 50% by introducing dedicated IO channels.

1. PROBLEM STATEMENT

Today’s semiconductor technology allows one to build embedded systems on a chip. The complexity of the systems tracks the rapid evolution of the process technology which will enable 50-100 millions transistors chips by early 2000.

For optimizing systems on chip, hardware/software (HW/SW) partitioning is of key importance. As more memory is becoming available on embedded chip, the complexity of the software is growing rapidly. An efficient partitioning between hardware and software leads to optimal systems by balancing criteria such as flexibility, performances, cost or power...
consumption. In addition, the hardware architecture will grow from basic microcontroller, memory and simple IO structure towards multi-processors and intelligent IO subsystem architectures.

Designing such complex systems on chip requires early validation and verification of the complete system by simulating hardware and software early in the development stage. Co-simulation allows one to explore architectures and quickly identifies bottlenecks such as bus loading or system latency.

In this paper, we present a system design framework to optimize system architectures for control applications and to accelerate the hardware design cycle. We use simulation based HW/SW partitioning techniques for early validation of the system requirements and for formally specifying new hardware modules. Defining and validating system requirements early on allow reducing design iterations and then accelerating the design cycle. In addition, the formal specifications of both hardware and software enables the generation of executable models and test benches which facilitate IC design by limiting ambiguities of textual specifications. We illustrate the advantages of the HW/SW codesign approach by analyzing dedicated peripherals for engine ignition timing functions. The framework helps us to specify a new hardware IO channel that is significantly reducing CPU loading.

The paper is organized as follows. In Section 2 we present the HW/SW systems on which we have developed our system design environment. The section 3 describes the main design steps from design capture to performance simulations. Before concluding, we present the results of a case study of HW/SW partitioning for ignition engine functions.

2. HW/SW CO-DESIGN SYSTEMS

Esterel [EST92] is a synchronous programming language for control-dominated reactive applications. Esterel was invented in the early 80s and is now a robust language for control applications. The early versions of the compiler produced automaton code. Automata compilation does not scale well with design space and is then limited to small or medium size functions. Recent versions [EST96] of the compiler solved the problem of state space explosion by compiling the design into sequential circuits. As a side benefit, the today compiler offers a natural path to hardware description languages (HDL) such as Verilog or VHDL.

The Esterel language allows the capture of both hardware and software control functions and simulation of them in the same environment. The discrete event simulation is very fast and permits the validation of the
complete design against realistic scenarios. The problem is partitioned into hardware and software modules and synthesized into respectively automaton and circuit code. Esterel provides constructs to write things once, supporting design reuse and library creation.

Esterel is the main capture language for POLIS [POL97], a hardware/software co-design tool. POLIS is a complete co-design environment, which allows partitioning, simulation, synthesis, modules interfaces and RTOS generation for embedded systems. POLIS follows a different synthesis path than Esterel that is based on CFSM, Co-design Finite State Machine. POLIS offers early performance estimations based on static code analysis for pre-characterized processors.

The main POLIS limitations are 1) the quality of its static estimation for modern cores, 2) the efficiency of the software synthesis compared to Esterel’s automata 3) the limited reliability of the heavy weighted framework and 4) the difficulty to handle large designs.

3. **HW/SW DESIGN FRAMEWORK**

This section presents the HW/SW design framework that we have developed for optimizing the partitioning between hardware and software modules. The framework allows us to quickly explore a variety of HW/SW partitioning driven by criteria such as data movement, interrupts rate and overall performance. In addition, both hardware and software design cycles are accelerated by providing comprehensive test benches and behavioral models.

The HW/SW design flow is represented in *Figure 1* and is composed of four main design steps: capture, verification, profiling and performance analysis.
3.1 Design capture

The first design step is to formally capture the behavior of the control algorithms and to build a test bench for verification purpose. In order to reuse existing hardware components, the designer builds an initial SW/HW partition with components of a HW library. Both HW and SW functions are described in Esterel. Esterel offers adequate semantics for capturing HW components, that is, concurrency, synchronization and exceptions handling.

Monitoring data movement, interrupts rate and tasks switching during simulation drives the partitioning. The partition is manually defined based on the simulations results.

Since both HW and SW modules are captured into a common language, a variety of partitioning scenarios can be explored quickly by redrawing the line between the two domains. Furthermore having a common co-design environment greatly eases code debugging.

The co-simulation is very fast: between 10-15 millions equivalent instructions per second on a Sun-Ultra-1 workstation. The speed of the co-simulation allows simulating stressful engine conditions over multiple engine cycles.

Figure 1. HW/SW Design Flow
3.2 Verification

The verification of the design consists of functional simulations driven by application data of the complete system and system properties checking through Formal Verification (FV) techniques.

The test bench creation includes a comprehensive set of applications data covering worst case conditions. The simulation speed offered by Esterel allows checking of the complete design against real application conditions. This results in a robust IC design and reduces design iterations.

As we formally captured the complete system, we can use FV tools for properties checking. FV complements simulations and guarantees that specific system properties are met by the design. Esterel offers a path to BDD, Binary Decision Diagram, solvers tools [VER96]. For example, we can ensure that one and only one spark pulse is generated per cylinder during an engine cycle.

3.3 Profiling

During the first two design steps, the design is optimized for data movement and inter tasks communications (messages passing). The next design step is to validate the partition by profiling the execution of the modules assigned as SW.

The execution time can be estimated by static analysis tools, such as POLIS [POL97] or Cinderella [Cin96]. POLIS gives static estimations of execution time and code size for a given target processor. The execution estimation gives the minimum and maximum cycles, which are good indications but are not accurate for today complex processors. The maximum given by POLIS does not take in account that branches could be mutually exclusive. The static estimation is difficult due to the complexity of processors that have pipelined instruction execution units and cached memory system. As cycle accurate simulators are getting available [SDS96], we choose to validate the execution estimations by running the complete design into a cycle accurate simulator.

The SW tasks are estimated by generating C code for both HW and SW modules and running the complete design on a cycle accurate simulator. The HW modules are compiled into C as sequential circuits while the SW modules are compiled into automata. The automata synthesis gives faster execution over circuits’ synthesis but does not scale very well with modules size. We developed a tool, SchedGen, to automatically generate a tasks scheduler as a Esterel function to separate HW/SW modules. SchedGen takes the top level wiring information between modules, that is, the signals between modules, to generate a tasks scheduler. Every module is compiled
and optimized separately. The tasks scheduler monitors modules output events and fires the tasks consuming the incoming events.

The execution of the SW tasks is monitored during simulation, which gives a complete execution profile over time. The worst case execution time (WCET) is determined by running worst case application stimuli. The WCET found by simulation is not the absolute longest sequence of instructions but a realistic longest path based on application scenarios. We believe that using static WCET values leads to over design.

Simulations on the cycle accurate simulator are two orders of magnitude slower than in Esterel environment. Most design iterations are done during the design capture phase. In practice, only a small number of complete cycle accurate simulations are required to validate the design.

3.4 Performance simulation

The final design validation step is to look at overall system performance. At this stage, we need fast simulation to handle the entire system.

The system observables for performance simulations are CPU loading, bus loading, worst case latency and hardware utilization. The architectural model is a delay model where the data traffic is simulated and probed. A delay model is a very abstract model of the HW components where the behavior is captured at the interface level only. For example, a CPU core delay model does not execute the logical instructions but hold the simulation for a number of cycles and interacts with the others HW components of the architecture (memories, interrupt controller, DMA, peripherals).

The input data for performance simulation is a list of SW tasks that have been profiled in the previous design step. Each SW task is defined by a set of attributes: the rate of execution, the execution cycles on a target processor, the registers access or the task’s priority.

The performance model, developed in BONes [BON96], reads the task table into a scheduler model that selects the active task based on its priority. The selected task is fed to the CPU that interprets the execution cycle count and peripherals access. Probes are attached to the different architectural elements to monitor the overall system performance.

4. SPARK TIMING FUNCTION

In this section, we illustrate the benefits of the co-design methodology by presenting an example of HW/SW partitioning for an engine timing function. The engine ignition is a critical timing function of the overall engine strategy.
4.1 Background

In most engine today the position of the pistons is measured via a tooth wheel installed on the crankshaft. The number of teeth of the wheel is nominally 60, whereby two teeth are missing for synchronization purpose. The ECU (Engine Control Unit) monitors the 6-degree tooth pattern, via an inductive sensor, in order to generate the ignition signals at a precise angle.

The spark pulse definition, Figure 2, is entirely defined by five parameters produced by the high level strategy.

The dwell time corresponds to the charging time of the ignition coil. The end angle gives the spark position. The start time is derived from the end angle time by subtracting the dwell time. Depending on the engine speed variation, the angle based falling edge can be forced when the dwell time limitations min/max are not respected. During the burn time no new pulses can be initiated.

![Figure 2. Pulse Spark Definition](image)

4.2 Constraints

The parameters produced by the high level strategy change dynamically due to other parameters such as battery voltage, load, etc. Hence, low level strategy receives these figures and must ensure the timing of the spark pulse in worst case acceleration/deceleration conditions with a resolution of 0.1 degree. Instant speed variations due to road bumps or engine cranking can reach 50’000 RPM per second.

This application assumes that a time and an angle based counters are available for input or output compare.

Due to new engine regulations, additional engine control strategies are implemented for improving engine emissions and performances. The objective is to reduce today embedded CPU loading which is already close to 100% in critical conditions.

The study presented here aimed at reducing CPU loading for the ignition driver by optimizing the HW/SW partitioning. As a result, we introduced a
Another important benefit of this approach is to speed up the process of creating requirements for new hardware peripherals.

4.3 Design capture

As you can see in figure 4, the hardware consists of a MCU, 32-bits RISC machine, an angle counter, a time bus and action sub-modules. The software is state machine based and does intensive calculations to compensate for engine acceleration and deceleration.

The initial mapping uses a Single Action Sub Module, SASM, a IO peripheral of the MIOS [MIO98]. The design is then optimized by mapping some SW functions to HW and tuning the actions sub-modules. This results in the definition of a new multi-actions sub-module.

We validated both designs by functional simulations and analyzed the benefits through profiling.

As we refined the partitioning, we implemented the HW/SW interface so the SW code has only one embedded await statement. Having multiple await statements will over fire the execution of the module and produce incorrect profiling results. We primarily used a memory map register interface between HW and SW.

4.4 Verification

For the two models, a functional simulation is done using the Esterel tools (csimul and xes). As stimulus, a sampled tooth pattern file of ten engine cycles is used. This functional simulation verifies if the pulses correspond to the specifications and constraints.

Then, a formal verification ensures that, depending of the variation of the five input parameters, only one pulse can be generated per cylinder during an engine cycle.
4.5 Profiling and results

Once the designs are validated, we profile the execution of software modules. From the profiling results, we build the tasks profile table for the performance simulation.

<table>
<thead>
<tr>
<th>Name</th>
<th>domain</th>
<th>Rate</th>
<th>Window</th>
<th>cycle</th>
<th>IO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sched_rise</td>
<td>angle</td>
<td>60</td>
<td>350-230</td>
<td>180</td>
<td>sasm</td>
</tr>
<tr>
<td>Sched_fall</td>
<td>angle</td>
<td>60</td>
<td>350-230</td>
<td>180</td>
<td>sasm</td>
</tr>
<tr>
<td>Control</td>
<td>angle</td>
<td>10</td>
<td>0-720</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Period</td>
<td>angle</td>
<td>10</td>
<td>0-720</td>
<td>80</td>
<td>TPU1</td>
</tr>
</tbody>
</table>

Table 1 shows a part of the table for the ignition function where the domain is either time or angle, rate is the rate of execution, window is the angle limits when the task is active and cycle is the execution speed for that window.

During performance simulations, various results are obtained like worst case task latency, CPU load, bus load and dynamic task distribution. The main result of the comparison of two architectures is the relative CPU load of the SASM version versus the new submodule. The analysis shows that CPU loading for 8 cylinders at 8,000 RPM on a M.CORE at 40Mz [MCO97] is reduced by 50% from 14% down to 7.3%.

5. CONCLUSION

We presented a HW/SW codesign framework that assists the designer in the exploration of the design space to optimize control systems on embedded chips. System design cycle is accelerated by 1) early validation of the complete system, 2) accurate performance analysis by using cycle accurate simulators, and 3) behavioral models synthesis for both SW and HW components.

We discussed a case study for an engine ignition timing function. The HW/SW analysis resulted in a new HW/SW partitioning with a saving of 50% CPU loading in worst case conditions over an existing HW/SW mapping.

Timer Processor Unit [TPU96]
REFERENCE


