Architectural and Bus Issues for HW/SW Design

Joseph Rothman
CARDtools Systems Corporation
101 Metro Dr. Suite 250
San Jose, CA 95110
408-501-0880
joseph@cardtools.com

Abstract

HW/SW tradeoff decisions made early in the design process improve performance. The complexity of SoC design with its processors, ASICs, operating systems, applications software and other HW or SW components makes it necessary to analyze bus utilization, arbitration and transactions before committing to specific hardware or software.

This paper provides an overview of the issues involved with measuring bus utilization early in the SoC design cycle with several examples of how this is measured and how performance can be improved.

1. Overview of SoC design, Importance of Architectural and Bus Utilization
2. How Architectural and Bus-related issues can be addressed early in the design cycle
3. Examples of how HW/SW design tradeoffs can be made based on busload.

Introduction – Today’s Embedded Market

Welcome to the new world of designing embedded applications. The embedded systems market is one of today’s most exciting and innovative markets. New embedded products and even new applications are being generated and consumed in an ever-accelerating pace. Some of the most recent and promising embedded application additions include:

- Personal Digital Recorders,
- MP3 players or digital jukeboxes
- Cable/Satellite Set-top Boxes
- Combined DSL/Hub/Firewall products
- Cell Phones
- Multifunction printers
- Digital cameras
- HDTV and DVDs.

Along with further developments in technologies, who can predict the success of new killer applications, such as?

- 3G phones
- Bluetooth and other wireless networks
- Terabit Networks
Automobile Information Systems

Architectural Design Challenges
It’s a very bright, and more importantly for the designers of these new embedded systems, challenging future. Even though the form and function of these systems may differ, they are all based on the same building blocks. All these systems consist of ASICs, processors, memory, buses, operating systems (usually real-time), software, drivers, middleware etc.

But today’s embedded systems are far more sophisticated than yesterday’s predecessors. The firmware is more elaborate and layered. The hardware more capable, with increased clock speed, improved instruction technology, pipelining, and greater bus widths. And even the process technology has improved in terms of reduced silicon size, integration of multiple cores, integrated memories, and even the possible mix of analog and digital. All resulting in our ability as an industry to produce SoC (System on Chip) embedded systems and solutions.

Along with all this new complexity and SoC capability, comes the need to understand the architectural design, both hardware and software, and bus issues of the entire system or risk the likelihood of a failed design at the integration test floor or in the verification phase.

It is no longer possible, or even desired, to allow the division of hardware and software in the development phases. The embedded system behavior and hardware/software partitioning must be understood as early as possible in the design process to insure specifications are satisfied and the proper technology tradeoffs are made. For example: what is the correct mix of software versus hardware? Which is the best processor for the job? How much software can be truly loaded? Will the system work? Where are the bottlenecks? Will the bus choke? As embedded designers we are always making trade-off decisions, but what’s the impact and at what cost?

An Embedded System consists of following components and considerations:
- Processor type – Which one, at what speed, with which on-chip peripherals,
- Memory- Needed memory size and speed,
- RTOS – Which Real Time Operating System, or simple home grown scheduler.
- Software Architecture – Tasking structure, Priority Schemes,
- Multiprocessing – Partitioning, loading,
- ASIC/Devices – Partitioning, signals,
- Power – Consumption, States, battery saving techniques
- Bus – transaction throughput, channel and width, partitioning.
All these building blocks affect one another. For example, if we are designing a new generation of a cellular phone with the goal of reducing its power consumption, we may decrease the processor speed, but the number of samples to be processed doesn’t decrease, so our buffer memory requirements may have increased. In addition, the RTOS overhead percentage has also increased and the algorithm’s performance runs slower. As a result, we may be forced to offload housekeeping tasks to a second processor or move critical algorithms to a dedicated ASIC or device. The result is that our bus traffic has increased which may or may not be our new system bottleneck. In addition, the new devices and architecture may now void all our previous power saving schemes.

In the final analysis, all of our embedded/SoC building block are all non-linearly coupled and dependent on each other’s behavior.

HW/SW tradeoff decisions made early in the design process improve performance. The complexity of SoC design with its processors, ASICs, operating systems, applications software and other HW or SW components makes it necessary to analyze bus utilization, arbitration and transactions before committing to specific hardware or software. The Bus is the critical artery of the entire embedded/SoC system.

Methods of Determining Architectural Impact Early

So how do embedded/SOC designers determine bus utilization early in SoC design cycle? Unfortunately, today's design decisions are based only on previous experiences, which may have limited validity in new SoC designs; therefore, their impact may have unpleasant consequences. In essence, designers will not truly know how the SoC design will perform until the first reference platform prototype is complete on the integration test floor. Obviously, any undesirable circumstances will be very expensive to fix and will result in missing the market window.

One option is to allow designers to build numerous prototypes to aid in their trade-off analysis. However, in today's market conditions designers simply do not have the luxury in terms of both time and money for such an approach.

Even co-simulation technologies are ineffective and save only minimal time. At the co-simulation stage all the major design investment has been done. In order to use these technologies all the hardware must be completely designed (e.g. all the HDL written) and all the software, including the device drivers, must be complete. Co-simulation technology is too late in the design cycle, too ASIC (EDA) centric and too slow to do a complete co-verification of the system.
In fact, the traditional EDA design flow of functional verification first and timing verification second may be adequate to ASIC design, but is dangerously inadequate for complete SOC systems. Today’s embedded systems are asynchronous event driven systems. The timing of the system will directly influence its architecture, therefore, its design. As a result, the buses of the system are the critical highways within a SOC design and will directly affect the partitioning decisions between the hardware and software.

Designers need an approach that can answer performance and timing oriented questions at the design phase, not at or near the final integration test phase.

One valid and new approach is for designers to model and simulate a virtual prototype of the embedded application or SOC to show that the hardware/software partitioning and bus issues can be addressed. Early application modeling and simulation can address design challenges, while at the same time show how inherent problems such as bus utilization and system delays can be identified and avoided prior to making any commitments on the final hardware and software configuration. This approach used is a Co-Design or a concurrent design modeling technique.

**The Co-Design Modeling Approach**

A Co-Design approach allows for high-level modeling concurrent simulation. In order to properly address these high-level architectural trade-off issues one begins by partitioning the design into its hardware and software components. The level of detail required starts with describing the logic and timing behavior as well as the interaction of both types of components, and refining these components as the design progresses. A Co-Design approach requires the designer to specify the probable devices, including the microprocessor and bus, and on the firmware/software side, the operating system and tasking architecture. The approach must be application-centric (bus-centric), not hardware or software centric.

The best co-design modeling approach would be modeling the system in a natural manner, not requiring new paradigms or methodologies. 1) The approach should allow modeling general functions, or hardware components and software components in a natural hierarchy approach. 2) Once the components are defined, we need to define the method of communication between the components. This should be a very general method or if interfaces are well known at a natural register, port or signal level for hardware communication and events, RTOS system calls or device drivers for software communication. 3) Finally each component/block needs to represent the behavior naturally. For hardware, concurrency must be modeled. For software, multi-tasking
must be modeled. One of the most difficult tasks will be to determine the detail of model needed. We recommend modeling to the level of the databook for the given component. This will allow users to “program” the device. In addition, timing should be coupled with the device logic.

Modeling the Bus will be one of the most critical components to model correctly, since the processors (along with the software), memory, devices and peripherals will use this channel to communicate throughout the system. Again it is recommended that the bus modeling consider the issues below, but that modeling is maintained at a “bus transaction”, not an HDL, level. The transaction level is the correct level of abstraction to understand system interaction efficiently.

When modeling the bus, features to model at a transaction may include:

- Available bus widths
- Number and type of bus channels (read, write)
- Speed of the bus
- Burst capability and methods
- Master/slave support
- DMA transfers
- Arbitration schemes
- Pipeline, cycle, delay schemes

An Example - Addressing Architectural and Bus-related Issues Early in the Design Cycle:

The following example provides an overview of the issues involved with measuring bus utilization early in the SoC design cycle and how system performance can be improved.

This Bus oriented system example will be based on a high level C model. It highlights multiprocessing support, Bus and On-chip interaction, Hardware to Software, Software to Software, Software to Hardware, and Hardware-to-Hardware modeling and communication examples and some “What-if” Analysis.
The example starts off conceptually straightforward. The system consists of two processors hooked into the same SOC on-chip bus. A simple C model of DRAM is also attached to the bus.

In this example, the first processor will write a block of words to memory, by requesting access to the bus channel and write to the defined memory location. Once the block is complete, a signal will be sent to the second processor to request the bus and read from memory. This logic then repeats (See below).
1. ITmailbox starts by sending a interrupt CPU1_1
2. Processor_1 starts a task to write a block of 16 consecutive words to memory starting at address 64, initial value of 512 incrementing by 1
3. Processor_1 task will send the write request to the BusInterface
4. BusInterface sends the request to the bus,
5. Bus take the request and passes it to the RAM and waits
6. RAM processes the request and sends the write complete signal to the bus
7. Bus sends the write complete back to the BusInterface
8. BusInterface interrupts the Processor _1 task write is complete and it can continue
9. ITmailbox is monitoring the cycle complete messages passed from memory to the bus, looking for the last address in the block
10. When the ITmailbox detects the last address acknowledge message, it sends an interrupt to the Processor _2 CPU to start the read cycle
12. Processor _2 receives the interrupt and starts the ReadTask.
13. ReadTask, reads the same data written by the WriteTask. It issues a read command.
14. The read request is sent to the uP2 BusInterface
15. uP2 BusInterface requests the bus for a read.
16. The bus processes the read and sends it the RAM
17. RAM processes the read request, and sends the data and the acknowledgement back to the bus.
18. Bus forwards the data to the interface, interrupts the ReadTask
19. ReadTask retrieves the data from a register in the BusInterface
20. ITmailbox is looking for the last address in the block and when detected it sends an interrupt to the write task
21. Cycle starts over till simulation time runs out

In this example the bus is modeled as a series of potentially concurrent ‘C’ functions or C-threads. The bus is requested/released by setting the associated bus “port.” The name of the port is the name of the C-function called. The port call passes a C-structure:

- Identifying the “id” of the device requesting the bus,
- The priority of the requesting device
- The address of the requesting device
- Type of operation (read/write)
- Size of data being passed
- Actual data

The OnChip Bus consists of a number of C functions including the:
- Definition of input and output ports
- Queue handler
- Queue request and transfer
- Signaling the completion of the bus cycle
- Bus protection – bus locking
- Bus arbitration
- Transferring request to target device

**Reviewing the results**
The following example shows a timeline of the system, and its components as a function of time. The two processors are shown in blue and the bus and RAM (the critical devices) are shown in green. Within the processor timeline multi-tasking is visible as individual tasks jockey for the CPU resource. Under this model the processor is done writing its block to memory by 5300 microseconds.

Bus utilization (meaning the bus is locked and in use transferring the request) is shown in green. It appears the bus is in use approximately 30% of the time. We can also inspect the bus queue and we can see that the queue never exceeds 1, meaning the bus resource is always granted when requested and no bus arbitration occurs.
If we look closer at the timeline, we can review the id of the device requesting and using the bus. In this case 0 stands for processor_1 and “1” stands for processor_2.

To truly understand the value of modeling at this level, let’s add two additional devices to the bus. We will add two DMA devices to interact with the same system we just profiled. The DMA devices will now compete for the bus along with the processors. If we re-run the model we can see some very interesting effects.
First looking at the timeline we can see the same general pattern as before, of processor 1 requesting the bus and writing to memory until complete; however, the amount of time it took to write the data is much longer. Instead of 5300 microseconds, as in our baseline it now takes 6600 microseconds. In addition, you can see that bus utilization has grown from 30% to nearly 100% utilization. If we review the bus queue, we now find that bus arbitration logic is being activated. On closer inspection, we can see the DMA ids, 3 and 4 are using the bus extensively.
As a result we can easily see that this system is "Not" processor bound, but bus bound. Having a faster processor will not necessarily improve system level performance. Instead, the designer should consider options such as a faster bus, bus splitting and other architecture strategies to off load the bus.

Conclusion

In summary, exciting new SOC technologies offer numerous design opportunities. The challenge is allowing designers to internalize and take advantage of these design opportunities early in the process. Today's embedded market is highly competitive and the applications complex. Waiting for the integration test floor to prove out ideas and understand bus traffic is no longer a prudent option, designers are far better off using co-design modeling techniques. For best results modeling should consider bus timing, data transfer, along with hardware and software models as a single system in order to allow proper trade-off analysis of various design approaches.
System Exploration Result: Bus Limited

CPU Done
Writing to
Memory

Add DMA
Load

Bus
Utilization

CPU Done
Writing to
Memory

Copyright © 2001 CARDtools Systems, All Rights Reserved.