Reference: "Embedded System Design, Modeling, Synthesis and Verification by Daniel D. Gajski and Samar Abdi.

CHAPTER 1 INTRODUCTION

In a MPSoC (multiple processor system on chip) the system design of a many-processor platform consists of performing co-simulation of software (SW) and hardware (HW) components. The custom HW components or Intellectual property are modeled with a timed functional model and integrated with the processor models into a transaction level model (TLM) hence representing the platform communication between components.

Basic concepts include well defined models, design decisions and model transformations.

System abstraction levels need to be defined in order to apply design automation techniques. Virtual systems can be prototyped in C based languages such as System Level Design Languages (SLDL) like SystemC and SpecC.

System Level Models include

Specification Model (SM): Used by application designers to prove that the algorithms work on a given system platform.

Transaction Level Model (TLM): Used by system designers to approximate design metrics such as performance, cost, traffic, communication, power-consumption and reliability

Cycle Accurate Model (CAM): Used by HW designers to verify the correctness of the generated system HW components.

Platform Design Include: The platforms usually have one or more standard processors, multi-media processors, specialty IPs and a multimedia of special interface components (IFs). Essential components consist of

Processor Components (PE): Standard or custom processors for computation tasks.

Storage Components: Local and global memories for data storage.

***Communication Components*:** Examples include transducers, bridges, data buffering and translation of one communication protocol to next.

***Interface Components**:* Examples of interface components include arbiters, DMA controllers, memory traffic, interrupt controllers which would be utilized for synchronizations and UARTs.

System Design Tools

Front End Tools: Examples for system design tools include C, C++, SystemC, Matlab, UML and similar graphic representations. It consists of System capture and platform development. The System capture could be a GUI which defines the platform architecture and product application code. The platform development tool generates timed TLMs of the platform architecture which executes the product application that is designed by the schematic capture tool. The Back End Tools: Utilized by designers to implement HW/SW systems that can be defined/divided to ASIC and FPGA manufacturing. The validation user interface (VUI) is used to debug and validate the developed SW and HW tools. The decision user interface (DUI) can be utilized to evaluate the quality of the metrics generated by estimators. Metrics can be generated to focus on platform creation, component partitioning, model creation and other system design decisions.

The current simulation based design flow is perceived to be extremely inefficient with limited scope and scale. The system design methodology is designed to improve the productivity for design engineers.

CHAPTER 2 SYSTEM DESIGN METHODOLOGIES

These are custom to products and are very product oriented.

Bottom up methodology: The abstraction levels are separated within it's own library. Each group supplies a component library for the next level of abstraction. The disadvantage of this approach is that the libraries must include all possible variations for metrics for future applications. This becomes extremely difficult to predict and can become recursive very quickly.

Top down methodology: Completes the overview of the design first. Hence it does not initiate component or system layout until the entire design is completed. The disadvantage of the top-down methodology is that it is very difficult to optimize the entire design. The design

decomposition/synthesis is repeated and causes further inefficiencies for designers. Hence it becomes very difficult to achieve metric disclosure since metric estimations can be difficult.

Meet-in-the middle methodology: Most designers utilize the meet-in-the middle methodology to take advantage of both methodologies and minimize disadvantages displayed by both methodologies. The methodology that is described in the textbook appears to be iterative where the physical design or layout is conducted 2-3 times. Hence this approach has the drawback of requiring designers to do layout more than once, hence system optimizations can be very difficult for the RTL components.

Platform methodology: The system design is usually initiated with an already defined platform. Such platforms have standard components such as memories and standard processors with well defined layouts. To optimize the design one can utilize standard templates for standard processor components and then custom components can be inserted for application optimization. The platform methodology can be upgraded to a system level methodology by the introduction of standard architecture cells and re-targetable compilers. Standard architecture cells can be pre-synthesized with standard cells and inserted into the library of processor components. The platform methodology with the two highest abstraction layers are well-suited for application experts with minimal knowledge of system and processor design. It also utilizes a re-targetable compiler to cover different architecture cells.

FPGA methodology: This uses a top-down methodology on both system and processor levels. It is difficult to optimize the entire design once completed.

System level synthesis: This starts with an application written in some model of computation (MoC) that must be executed on a platform of multiple standard and custom processors connected to an arbitrary network. Hence this type of platform is defined partially or completely after estimating some characteristics of application in terms of performance, cost, power utilization, configurability and other considerations.

Processor synthesis: On the processor level, the components are synthesized as standard processors, custom processors and custom HW units called Intellectual Property (IPs). The standard and custom processors are defined by their instruction sets. Custom processors can also be defined by the algorithm or the programming language code that they execute. The custom processors are programmable so that new algorithms and code can be added. Custom HW units or IP are usually not

programmable, they are used as accelerators to execute special functions for a specific application such as multimedia applications. The process of synthesis starts with a given specification in a programming language which is compiled into some tool model such as CDFG or FSMD or a three-address code. This formal model can be used for estimation of future processor architecture and its metrics along with partial or complete allocation, binding and scheduling. Processor synthesis is called High Level Synthesis (HLS).

CHAPTER 3 MODELING

Models of Computation (MoCs) and design languages provide the foundation for defining system behavior and models throughout the design flow. The most common model of computation is an imperative model written in C and C++. The program statements are composed into concurrent blocks that communicate through signals to exchange sequences of values and events. The so called synchronous languages follows an approach where concurrency and ordering is specified in the code instead of relying on extracted or underlying scheduling of operations.

Process-based model: Process based MoCs represents computation as a set of concurrent processes. The overall system is modeled as a set of blocks of code that execute in parallel and are generally independent of each other. The various processed based MoCs vary in semantics of communication that is supported to exchange data and establish dependencies. Low-level and implementation-oriented thread-based models are built on shared memory and shared variable semantics to explicitly synchronize access to shared resources.

Definitions of concurrency and communication in process based models translate into properties such as deadlocks and determinism. Deadlocks can arise if there is a circular dependency between two or more processes where each process holds an exclusive resource that the sequential one in the chain is waiting for. Deadlocks can be prevented or avoided by statically ensuring that chains can never occur or by dynamically breaking them at runtime.

Determinism is related to the outputs of a model for a given set of inputs. For a given deterministic model the same inputs will produce the exact same results. If it is non-deterministic its behavior at least for some inputs will be un-known.

*Process Networks: S*pecialized process-based MoCs have been proposed that provide deterministic properties on a global scale and allow the non-deterministic execution of individual processes. This can only be achieved if the order of process execution does not affect overall behavior of the system. The Kahn Process Network (KPN) processes are allowed to communicate via unidirectional and point-to-point asynchronous message passing channels where messages (also called tokens) are of arbitrary type. Channels are unbounded where sender do not block and receivers always block until a message is available. Within the processor the sequence of channel accesses is predetermined and processes cannot change their behavior depending on the order in which data arrives on their inputs. The behavior of the overall system is deterministic and does not depend on the order in which processes are scheduled.
In a KPN process, the chosen scheduling strategy will have an impact on properties such as completeness or memory requirements.

Dataflow: The KPNs require both dynamic scheduling with runtime context switching and dynamic memory allocation. Demand driven scheduling prioritizes boundedness over completeness. The dataflow models map into concepts of block diagrams with continuous streaming of data from inputs to outputs. As a result they are used in the signal processing domain and as the basis for many commercial tools such as Lab View and Simulink. The synchronous data flow (SDF) models have been widely adopted.

System Design Languages: The MoC can be represented in a variety of languages ranging from C/ C++ to SystemC.

Netlist and Schematics: Netlist models are the basis for describing block diagrams in early tools for computer aided schematic entry and editing.

Hardware Description Languages: After capturing netlists and schematics, interest rose in representing not only the structure of designs but also their design behavior. In the early states, the

ideas were applied to the description of hardware blocks at the gate level and later were transferred to the register transfer level (RTL). The design is described in the micro-architecture which consists of functional and storage units connected by wires at the RT level. Each RT component such as a register or an ALU will consist of logic gates while its behavior is inherently modeled in the form of Boolean expressions.

For simulation of HDL models, a so called discrete event simulator maintains a logical simulated time and a queue of events ordered by their time stamps, the execution of these events is triggered.

System Modeling

The system design is the process of going from a high level system specification of the desired functionality to a system implementation at the RT or instruction-set level.

Design Process

In each design step, a refinement tool takes the input model and implements a set of design decisions in order to generate an output model at the next lower level of abstraction. In the process tools, insert a new layer of computation that reflects and represents the given decisions.

Processor Modeling

The computation processes of the system behavioral model are mapped onto processors. Each processor runs a piece of application code, these can be general purpose processors or customizable Application Specific Integrated Processors (ASIPs) down to the fully custom hardware units. At the specification level, the application is modeled as a network of communicating processes. In the case of a SW processor the application processes runs on top of the operating system and provides dynamic scheduling and multi-tasking services. This application and OS software has to run on top of the actual processor hardware which realizes physical bus interfaces and interrupts for communication with the external world. The hardware abstraction layer (HAL) provides interfaces such bus drivers and interrupt service routines (ISRs) for accessing the processor hardware from the software side.

Application Layer

To achieve the required simulation speeds, the application model is executed on the event driven SLDL simulation kernel which is running natively on the simulation host.

Operating System Layer

In the application layer, computation processes are modeled as running concurrently. However the processors can execute a single thread of control or a limited number of threads at any given time. Hence the operating system layer introduces scheduling of parallel processes on the inherently sequential processors.

In a second step, the remaining tasks are then considered to be dynamically scheduled during runtime by a (RTOS) Real –Time Operating System. An abstracted model of the RTOS is inserted into the processor's operating system layer. In the process, tasks are refined to run on top of the OS model by inserting the necessary OS calls for task management (creation and deletion), synchronization (event handling) and timing (delay modeling).

The goal of high level RTOS modeling is to provide a solution that combines the speed of native application. Key modeling concepts are related to multi-tasking, preemption, interrupt handling and inter-process communication and synchronization. Functionality of the OS model is as follows: tasks and services are not allowed to access the simulation kernel directly. The OS model's functionality includes selectively relaying calls to the kernel and ensuring that at any given time only one task is active and all other tasks are blocked at the SLDL level.

Hardware Abstraction Layer (HAL)

The hardware abstraction layer provides the lowest level of functionality that can be implemented in software. This will include hardware specific driver code to transfer arbitrary blocks of bytes over the processor bus interface. The HAL contains templates of low-level interrupt handlers that are associated with corresponding HW interrupt sources.

Hardware Layer

With the final hardware layer an accurate model of the actual processor hardware is included. In relation to its external bus interfaces, the HW layer provides two different levels of communication detail. The bus functional model (BFM) extends the processor hardware layer by including a cycle accurate model of external bus and interrupt protocols. The bus functional processor models are accurate for system integration and hardware synthesis.

Communication Modeling

The OSI/ISO model divides the functionality required to implement any network layer into seven different layers. Other modes of communication include buses for serial, point to point or network oriented busses such as RS232, CAN, Ethernet or wireless protocols. In the OSI model the application layer provides the designer with high level programming model for describing the system behavior. Commonly used communication methodologies include (a) one-way synchronization (control flow) (b) Shared variables which holds data and does not include synchronization (c) Synchronous and asynchronous message passing channels (d) Queues (e) Complex and user –defined channels with extended semantics. The presentation layer is responsible for formatting of data and for conversion of abstract data types found in the application into un-typed blocks of bytes to be transferred over the network. The session layer is responsible for selecting the end-to-end channel used to transport messages of each application stream and to implement all the necessary means to separate messages of different streams over a single transport. The network layer spits the overall network into smaller subnets where different subnets can be implemented using different underlying media. Transducers synchronize and exchange address and data items with each side. Part of the transport layer implementation is the selection of the maximum packet size. Link layer provides logical links for point-to-point packet transfers between adjacent stations of the network. The stream layer multiplexes different data stream over an underlying shared medium. The media access (MAC) layer is the first layer to provide an immediate abstraction of the shared underlying medium. The protocol layer provides services to transfer groups of bits over the actual bus.

System Models

Various system models include Network TLM, protocol TLM, Bus cycle accurate model and cycle accurate model.

CHAPTER 4 SYSTEM SYNTHESIS

In hardware system design, physical synthesis automates the placement of transistors and the routing of interconnects from a gate level description. The gate level description has been created by logic synthesis from a register transfer level (RTL) model. The three techniques to enable system synthesis are model generation, mapping generation and platform generation.

System design trends: The platform architects define the platform definition which includes the type of processors and their communication architecture. The number of processors will depend on the required number of independent tasks or available parallelism in the application. The hardware engineers develop the HDL models and configure the processors and other IPs along with the board design. The SW engineers develop the system software. A virtual platform based design methodology begins with the idea that a model of the HW platform may be used for SW development rather than prototyping a board. This is called a virtual platform. The VP provides a programmable model of all SW processors and functional models of all custom HW components. Virtual platforms are implemented in C/C++. With SystemC virtual platforms for multi-core architectures can be modeled.

TLM based design: The input to the design process is the system specification model where the specification consists of an application model mapped to the SW/HW platform. *TLM based performance estimation:* Various metrics can be evaluated by simulating the TLM. These metrics help the designer evaluate design decisions such as platform component selection and application to platform mapping along with timing estimation and communication delays. Synchronization in the bus channel may be modeled with a set of flags and events. The sender process must wait until the receiver process is ready to receive and the receiver must not read the data until the sender has provided it. Post synchronization, bus arbitration is modeled. The simplest model is the mutual exclusive object called the mutex. An arbitration request corresponds to a mutex lock operation and once the transaction is complete the process releases the bus with a mutex unlock operation. Data transfer modeling: After synchronization and arbitration the sender process is ready to write the data on the bus TLM semantics.

CHAPTER 5 SOFTWARE SYNTHESIS

The synthesis overview consists of the Model of Computation -> System Synthesis -> System Model -> Hardware Synthesis -> Interface Synthesis -> Software Synthesis -> Binary Code and HDL RTL. Embedded software drives and control customized hardware accelerators that are integrated into the platform. The embedded system is part of a physical control process such as anti-lock brake system. Embedded systems can be programmed in C, C++ and Java. Java omits the complicated C++ constructs including templates, namespaces, multiple inheritance and operator overloading. It also removes visible complexities in memory management and pointer arithmetic from the user. Java was implemented with an interpreter to make it's object code, the Java byte code portable. The RTOS which is a real time operating system is a software layer above the bare processor that controls the concurrent execution of applications and hence provides services for communication and synchronization. A real time operating system RTOS provides the following services: Task management and Inter Task Communication (IPC). Task management creates, terminates and controls tasks while IPC enables tasks to exchange information and synchronize with each other. An RTOS's ability to manage its various responsibilities is dependent on supported scheduling algorithms. A scheduling algorithm determines the order in which different flows of execution are given access to a resource.

Classification properties for scheduling algorithms are: Preemptive/ Non-preemptive (with this algorithm a running task may be interrupted in its execution at any point based on the scheduling policy). Static/Dynamic (whether the task scheduling parameters can be updated during runtime).Offline/On-line (characterizes when scheduling decisions are made). Priority based scheduling (The task order is determined by task independence and importance). Earliest deadline first (EDF), Rate Monotonic (RM) –tasks are assigned, Round Robin (RR) – assigns tasks to the CPU based on time slices and tasks take turns. Code generation is the first step of software synthesis, it generates sequential code in the target language for each task and programmable component, it also translates the application module hierarchy into the target programming language.

The generic software stack consists of the RTOS based multi-tasking application which consists of SW application -> Drivers -> RTOS abstraction layer -> RTOS -> interrupts -> hardware application layer. Interrupt based multi-tasking: It is an alternative option for dynamic scheduling. Synchronization is

required to signal that a communication partner on the same link is ready for a data transfer. Polling synchronization: The hardware unit exposes a memory location to the processor bus.

Communication: IPC takes between tasks and the same processor. External communication is the communication between software processor and an external hardware accelerator, external memory and a communication element. This type of communication consists of data formatting, packetization, synchronization, interrupt synchronization, polling synchronization and media access control.

Startup code generally consists of both platform specific and application specific code. It initializes the underlying hardware, registers, interrupts handlers and prepares them for multi-tasking. *Binary image generation*: The final aspect of SW synthesis is the generation of a complete target binary. A cross compiler tool chain compiles the generated code, the cross compiler is specific to the target processor and binary format. *Execution:* After successful binary generation the target platform is implemented

CHAPTER 6 HARDWARE SYNTHESIS

HW components are synthesized as standard or custom processors or as special HW units also called Intellectual property components (IPs). HW component synthesis which is called High Level Synthesis (HLS) uses the tool model to estimate metrics and performs allocation, binding and scheduling tasks. HLS is initiated with RTL Components such as storage components, functional units and buses. Estimation and optimization focuses on data-path connectivity, reducing the number of functional units in the data-path and focusing on storage components. Various optimization strategies include reducing data path cost by reducing the number of registers, functional units and connections. Pipelining is also applied to functional units, data paths and controllers. Interface synthesis: These tasks included RTL component allocation, variable, operation and connectivity binding, pipelining, scheduling and RTL generation.

CHAPTER 7 VERIFICATION

The techniques for verifying design models can be classified into two groups: simulation based models and formal methods. In formal verification methods a specified property is statically checked instead of an instance of a property. Hence this means that the property will be confirmed under all specified inputs. Types of formal verification include equivalence checking, model checking and theorem proving. In equivalence checking a mathematical transformation is applied, in model checking the implementation is a state transition and the specification is a set of properties. Each property in the specification is checked by traversing all the states in the transition system. Theorem proving methods deduces the equivalence of formulas of the specification model and the implementation model to a given mathematical logic. Techniques like assertion based verification are used to complement the traditional simulation and debugging of design models. In contrast the second form of verification which is simulation allows designers a high degree of independence in writing models.

Simulation based methods: Simulation is the most widely used method to verify system models, the stimuli is applied to the DUT's inputs. These inputs then trigger a series of events and computations to the DUT model. A typical simulation environment consists of Specifications, Simulators, Stimulus, Device-Under-Test and Monitors.

Test bench: The stimuli and monitor for a verification effort can be created from a high level specification of the DUT. The paired stimulus and corresponding monitor are also called a test-case. The part of the device under test (DUT) is called its simulation coverage.

Coverage: We can use statement coverage to see how many lines of code were visited during a verification run. If during simulation with a given test-case, 100 statements out of 1000 statements in the design were executed the statement coverage for the test case is 10%. However not all possible scenarios of the statement is executed.
Coverage is limited to the given property and model representation.

Performance improvements: The simulation performance can be improved by selecting test cases that maximize coverage with minimal simulation runs. Another strategy is to improve simulation performance

Stimulus optimization: Generate only valid test vectors that focus on the scenarios that the design is constrained to work for. The constraints specified in the property lead to a set of legal inputs that form the test pattern. The test generation tool can identify these pragmas and produce tests based on them. Pragmas are properties that are embedded in the design model with special comments.
Monitor optimization: Monitoring only primary outputs of a design during simulation lets us know if a bug exists however tracing the bug to its source can be difficult for complex designs. If the source code of the model is available then assertions can be placed on internal variables or signals in the model.

Speed up Techniques: Overall simulation time can be reduced by simply increasing the simulation speed. The two common speedup techniques are cycle simulation and emulation. Cycle simulation is used when we are concerned about the signal and variable values only at the clock boundaries, hence the signal values are updated at the clock boundaries only. In contrast emulation or event driven simulation keeps track of all events even those between clock edges and is thus much slower. A third speedup technique is the use of reconfigurable hardware to implement the DUT. The FPGA can be hardwired in the system for rapid prototyping this is called in-circuit simulation.

Modeling Techniques: Abstract level models can reduce functional verification time. The abstract level model removes unnecessary implementation details and hence simulates quicker. There are numerous abstract models including functional specification model, platform model and transaction level model.

Formal Verification Methods: These techniques use mathematical formulations to verify designs. The key difference from simulation based verification is the absence of a test pattern.

Comparative Analysis of Verification Methods: The three most common metrics for determining a verification method is coverage, cost and scalability. Coverage of a verification method determines how much of the design functionality as been tested. Scalability refers to the limitations on the size or type of design that is being verified.
Coverage provided by formal methods is higher than simulation methods over the same run time. However model checking has the disadvantage of poor scalability and theorem proving is very expensive which makes equivalence checking the most commonly used techniques in the industry. Assertion based techniques replace pseudo random simulation since they offer better coverage. A

system level design methodology starts with a well defined executable specification model that serves as a golden reference.

CHAPTER 8 EMBEDDED DESIGN PRACTICE

In recent years, we have seen a push towards development of so called Electronic System Level (ESL) tools. In the past the EDA tools were utilized however with the need of higher levels of abstraction, a new class of tools called the ESL tools have emerged.

Academic System Design tools include Metropolis, SystemCoDesigner, Daedalus, Peace and System on Chip environment SCE. SystemCoDesigner is a system level design space exploration environment. The SystemCoDesigner supports applications written in a dynamic dataflow oriented MoC towards streaming applications. Such input models are captured by using a well-defined subset of SystemC called SysteMoC. The SysteMoC model is similar to KPN. The SystemCoDesigner will automatically generate a library of SW and HW implementations. SW implementation include transforming SysteMoC into C code, on the HW side , Forte Csynthesizer tool to RTL . Commercial System Design tools include CoFluent, Space Codesign, CoWare, SoC designer, Vast and Virtutech.

Academic Embedded SW Design tools include Eclipse, POLIS, DESCARTES. Commercial Embedded SW Design tools include Mathworks real time workshop, DSpace: TargetLink, Esterel Technologies: Scade and UML/SYSML Products.

Academic HW design tools (C to RTL) include GUAT, NISC, SPARK HLS and XPILOT Synthesis system. Commercial HW design tools include Catapult Synthesis, Csynthesizer, PICO, CyberWorkBench and Bluespec.

Bibliography

Daniel D. Gajski, S. A. *Embedded System Design, Modeling Synthesis and Verification.*

https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/edh_ed51006.pdf. (n.d.).

https://github.com/alpaddesai/SystemOnChip

https://insights.sei.cmu.edu/sei_blog/2017/01/the-challenges-of-testing-in-a-non-deterministic-world.html

https://www.capsl.udel.edu//courses/cpeg852/2015f/slides/CPEG852_2015_SDF_INTRO.pdf