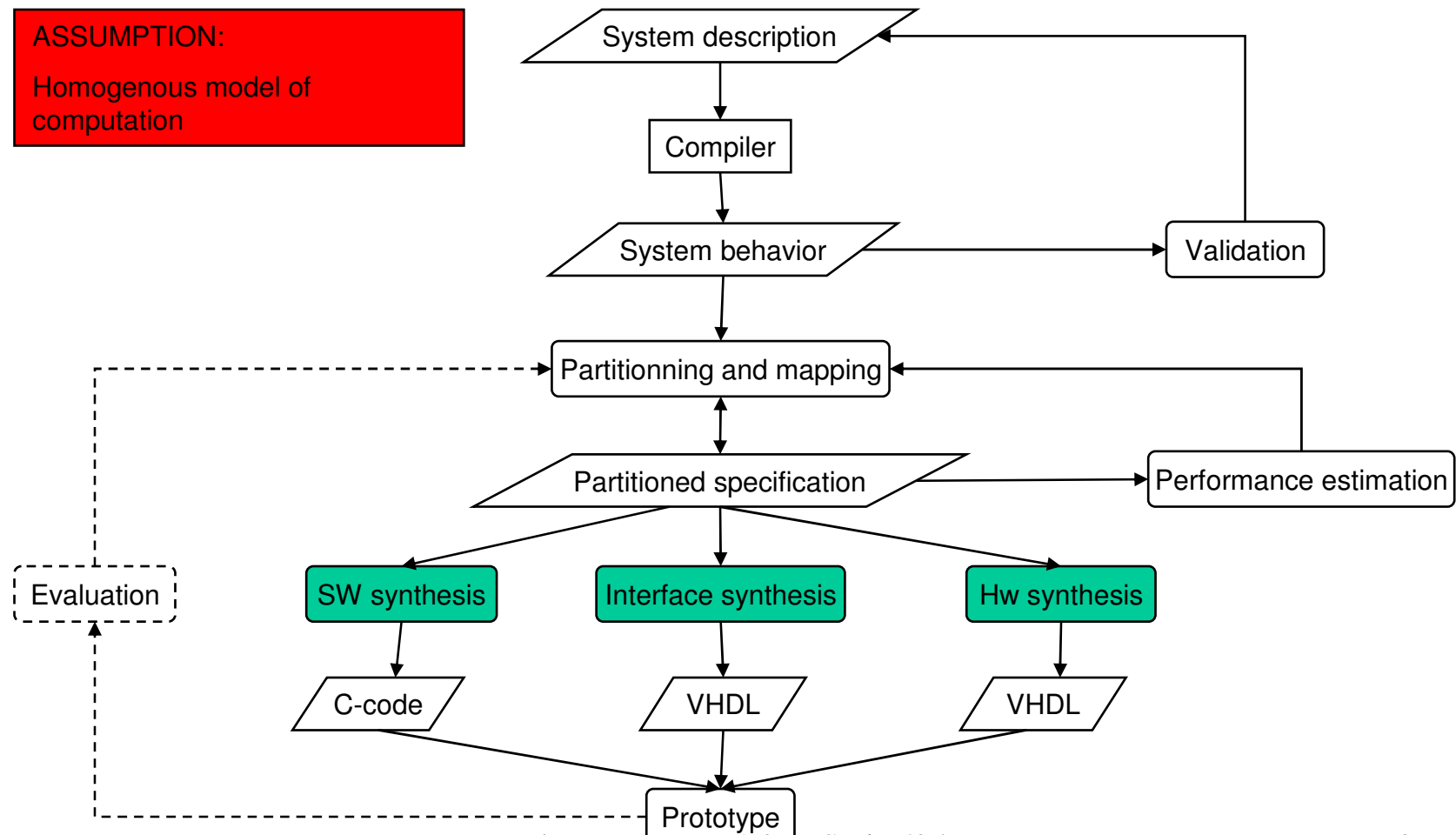


# Introduction to Cosynthesis

Rabi Mahapatra

# A Co-design framework

**ASSUMPTION:**  
Homogenous model of computation

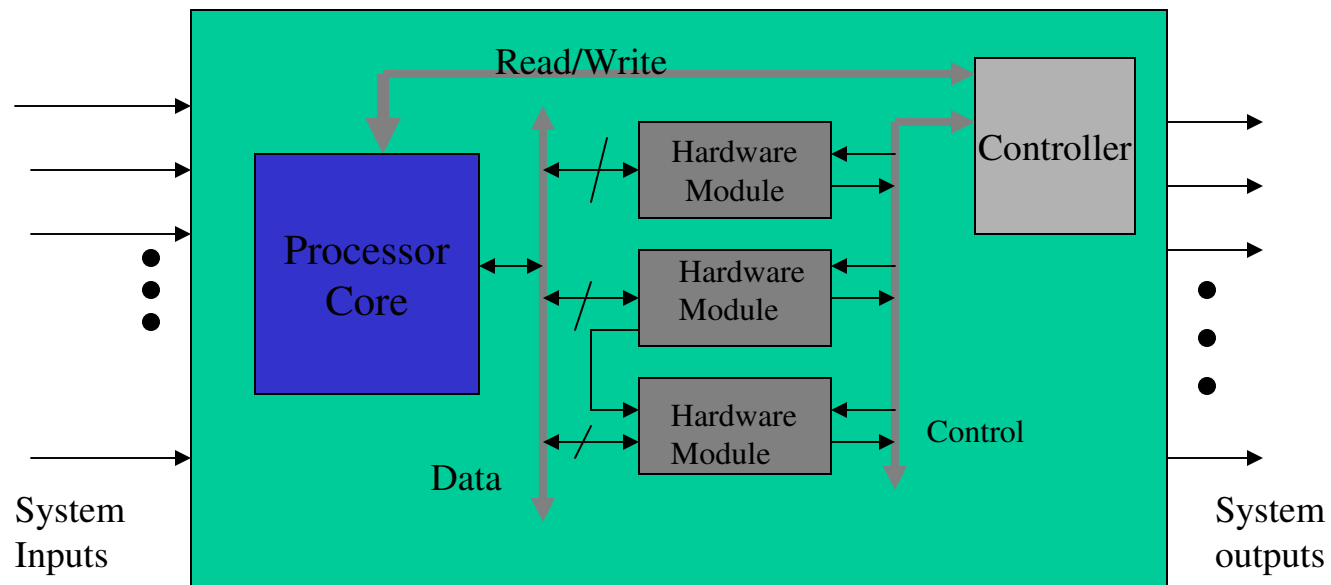


# Introduction

- What is cosynthesis?
  - It is a problem to synthesize the hardware, software and interface for final implementation.
- Based on a target architecture, we consider the development of architecture and present here.
- This step comes after partitioning the nodes and selecting the application beans and obtaining a suitable schedule.

# Architecture Model

- Target Architecture



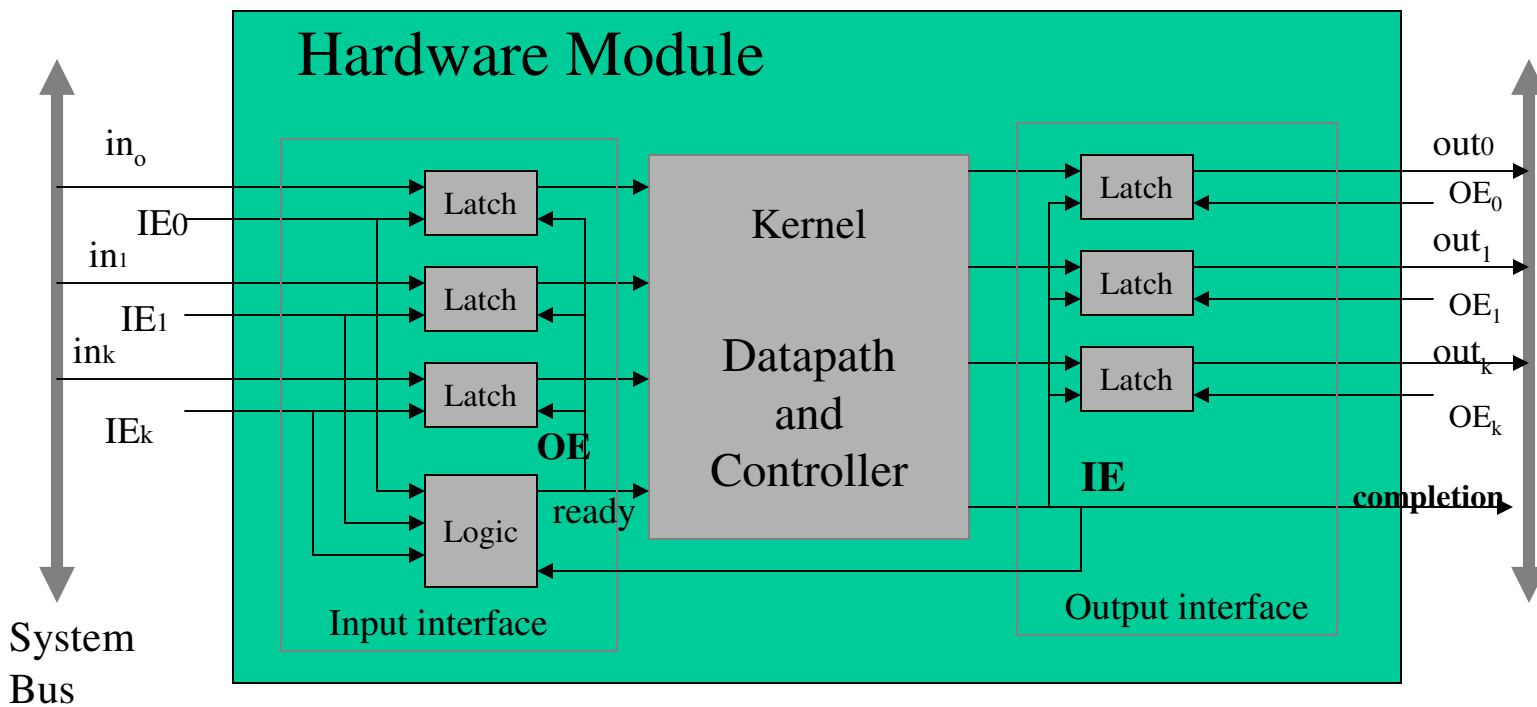
# Architecture Model

- Single programmable processor and multiple hardware modules connected through a single system bus.
- Hardware Module:
  - datapath and controller, input and output interface. I/O interfaces between hardware modules and processor.
  - Each node mapped to hardware is synthesized as hardware module (no hardware reuse between module).
- Processor:
  - software component of the architecture, the program that runs here. Includes device drivers that communicate between hw & sw

# Architecture Model

- Communication between hardware and software:
  - Memory mapped, asynchronous and blocked.
  - Different type of communication are possible between the components.
- Component Configurations: Globally asynchronous and locally synchronous
- Reasoning: *schedule generated by partitioning is based on execution-time estimates and is not guaranteed to be cycle-accurate. One can not determine when processor or hardware module would clock!*
- Controller: responsible to activate the hardware modules based on the schedule due to partition.

# Hardware Module Architecture



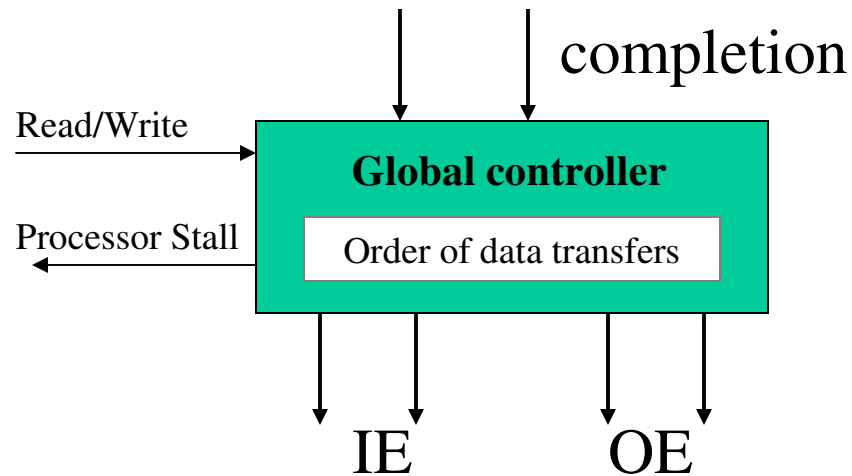
# Hardware Module Architecture

- Handshaking Signals: *ready, completion*
- Latch: controlled by input enable (IE) and output enable (OE)
- Working: *kernel begins computation at ready signal indicating the availability of data at its input. Then raises completion flag after computation. The ready signal is a function of input enable signals and the completion signal (generated by logic).*

# Hardware-software interface

- Each input and output of hardware module has unique address in the shared address space of the processor (memory mapped).
- How to communicate between processor and HW module now?
  - Can we use decoders to select appropriate input in the module?
  - It is possible but for large number of modules, decoder area and delay becomes significant.
- The *ordered transaction principle* is applied.
  - Based on the schedule, the order of data transfer is examined across hardware-software interface. Each data transfer is assigned a unique memory address. This way, the sequence of addresses to which read/write takes place is known a priori. This address corresponds to unique latch input/output address. A global controller uses the order information to activate the input and output latches.

# Hardware software interface



1. Processor issues Write request: Global controller issues appropriate IE signal for the latch. The read cycles can not overlap. When all input signals available, ready signal is issued.
2. Read action: controller checks if corresponding module has completion signals set. After completion is set, OE signals for the output latch is set.

# Software -Software interface

- Data transfer between two software nodes are assigned memory addresses in the internal data memory of the processor.
- Communication between the two takes place by writing the results to corresponding locations in the internal data memory.
- There is no need to check semaphore as the software executes sequentially.
- **Hardware-Hardware interface:**
  - direct connection, and use of ready and completion signals.

# Cosynthesis Approaches

- What we Need to synthesize?
  - datapath & controller for each hardware module,
  - program running on processor including codes for memory access and communication with HW modules.
  - Global controller,
  - I/O interface of hardware modules and
  - netlist connecting the controller to various modules and processor.

# Cosynthesis Approach

- The input to the cosynthesis stage is annotated DAG after partitioning. It includes the possible mapping selections, implementation bins and a schedule.
- The first step in cosynthesis is *Retarget the DAG*. It is a technology dependent presentation tool.
  - Each incoming node in DAG is converted to appropriate state of presentation. Example: A software node takes either C code or assembly code and hardware node as VHDL/Verilog.
  - If different implementations are available for a node the target tool selects the necessary bin here.
  - In case of hardware-mapped nodes, transformation and resource-level implementation is possible.

# Retargeting approaches

- **Library Approach:** Assumes the existence of library for each technology and implementations (Ex: if FIR filter is the node, one should maintain FIR filter in C, verilog, assembly codes. Also different algorithms of FIR filter with multiple representations)
- Advantages:
  - computationally efficient and retains modularity. If a node or its implementation changes, one can select the new library element and re-synthesize.
  - Each module can be individually optimized to improve the quality of implementation.
- Disadvantage:
  - It depends on richness of the library. If an element is not there, one has to develop it.

# Retargeting Approach

- **Compilation Approach:** Compile the technology-independent representation of every node into its desired representation. The node is described in high level description such as C or Java. Then, this high level description is translated to intermediate representation (like control-dataflow graph). The intermediate representation is compiled to either Verilog or assembly, depending on desired synthesis technology.
- Advantage: Simplistic and traditional approach. Can use available compilers. Not to worry about the special library components.
- Disadvantages: optimization depends on compiler. Recompile complete design that takes more time.

# HW, SW and Interface Graph

- Graphs are fed to the hardware, software and interface synthesis tools.
- Hardware Graph:
  - A separate hardware graph is generated for each node mapped to hardware.
  - Hardware graph contains several sub-nodes. These are annotated with the transformation and sample period corresponding to implementation bins.
  - The hardware synthesis tool generates a datapath and controller for each hardware graph.

# Software graph

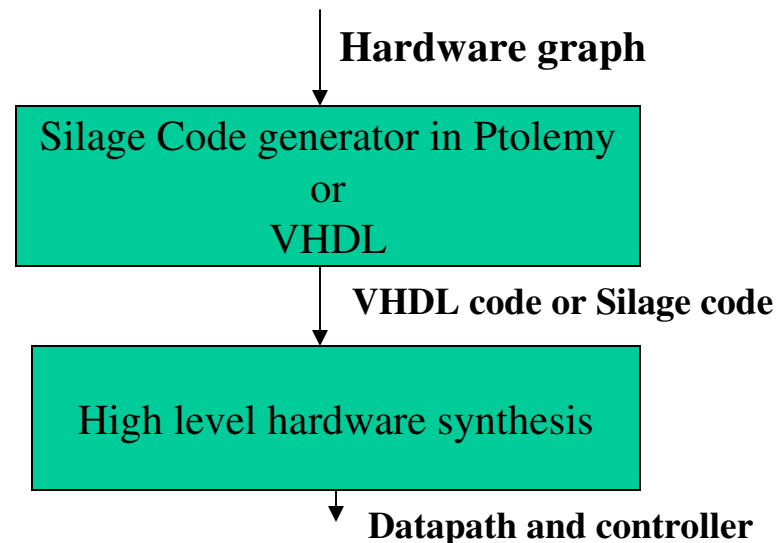
- All nodes mapped to software are combined into a single software graph.
- *Send* and *receive* nodes are added wherever a hardware-software communication occurs.
- Note: partitioning algorithm generated a global schedule to execution order of all nodes. An ordering between the nodes of the software graph is derived from the above schedule.
- The software synthesis tool generates a single program from the software graph. The program contains codes for all nodes. The code is concatenated in the predetermined ordering.

# Interface graph

- Interface Graph:
  - The *order generator* determines the order of transfers between nodes mapped to software and hardware
  - The interface synthesis tool generates the global controller using this order of transfers.
  - It also interface glue logic (latches, logic to generate ready signal) for the hardware modules.
- A netlist that describes the connectivity between hardware modules, the processor, and the global controller is next generated. A standard placement and routing tool can be used to synthesize the layout for the complete system.

# Hardware Synthesis

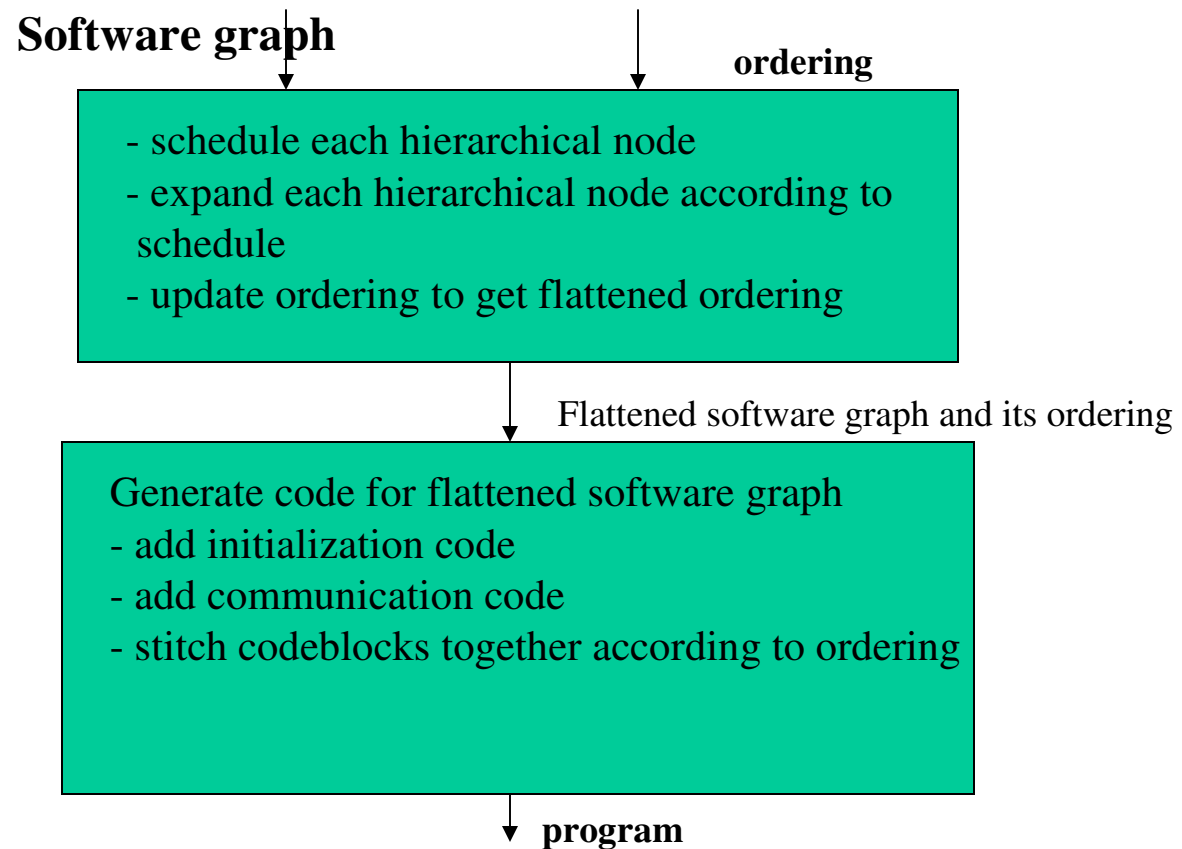
- Approach:
  - Use VHDL or Verilog (may be Silage for Ptolemy) to describe the hardware graph to synthesize its datapath and controller.
  - Feed this description to hardware synthesis tool for implementation.



# Software Synthesis

- All the software nodes are to be together and augmented by send-receive.
- Each software node is called a *codeblock* which is technology dependent and represents the node functionality.
- Software synthesis process needs to stitch together these codeblocks and form a single program to be executed by the processor.
- Use *flattened ordering* that consider the schedule from partitioning and includes send-receive nodes.

# Software synthesis



# Interface synthesis

- Involves synthesis of global controller and other glue logic
- FSM can be used to describe the global controller
  - Use standard logic synthesis tool.
- A template-based approach may be used for glue logic.

# Other synthesis approaches

- VULCAN system: hardwareC and C
- COSYMA system: C like description, hardwareC
- SIERRA system: for multi board application.