Hardware-Software Co-Design System Partitioning

COE718: Embedded Systems Design
http://www.ecb.torontomu.ca/~courses/coe718/

Dr. Gul N. Khan
http://www.ecb.torontomu.ca/~gnkhan
Electrical and Computer Engineering
Toronto Metropolitan University

Overview

- Embedded System Co-design
- Co-specification
- Hardware-Software Co-synthesis and Partitioning
- Co-simulation and Co-verification

Introductory Articles on Hardware-Software Co-design available at the course web-page

Real-time Embedded System

- Real-time Embedded System Design Involves:
  - Performance analysis
  - Scheduling and allocation
- Accelerated systems
  Use additional computational unit(s)
  dedicated to some functions ?
  - Hardwired Logic (e.g., FPGA)
  - Multiple processing elements (PEs) including extra CPUs

Hardware/software Co-design:

Joint design of hardware and software architectures.

Why use Microprocessors?

- Alternatives: field-programmable gate arrays (FPGAs), custom logic, etc.
- Microprocessors are often very efficient: can use same logic to perform many different functions.
- Microprocessors simplify the design of products.
- Microprocessors use much more logic to implement a function than does a custom logic.
- But microprocessors are often at least as fast:
  - heavily pipelined;
  - large design teams;
  - Agressive VLSI technology.

SoPC? FPGA with soft CPU cores

Challenges in Embedded System Design

- How much hardware do we need?
  - How big is the CPU? Memory?
- How do we meet our deadlines?
  - Faster hardware or cleverer software?
- How do we minimize power?
  - Turn off unnecessary logic? Reduce memory accesses?

Design methodologies

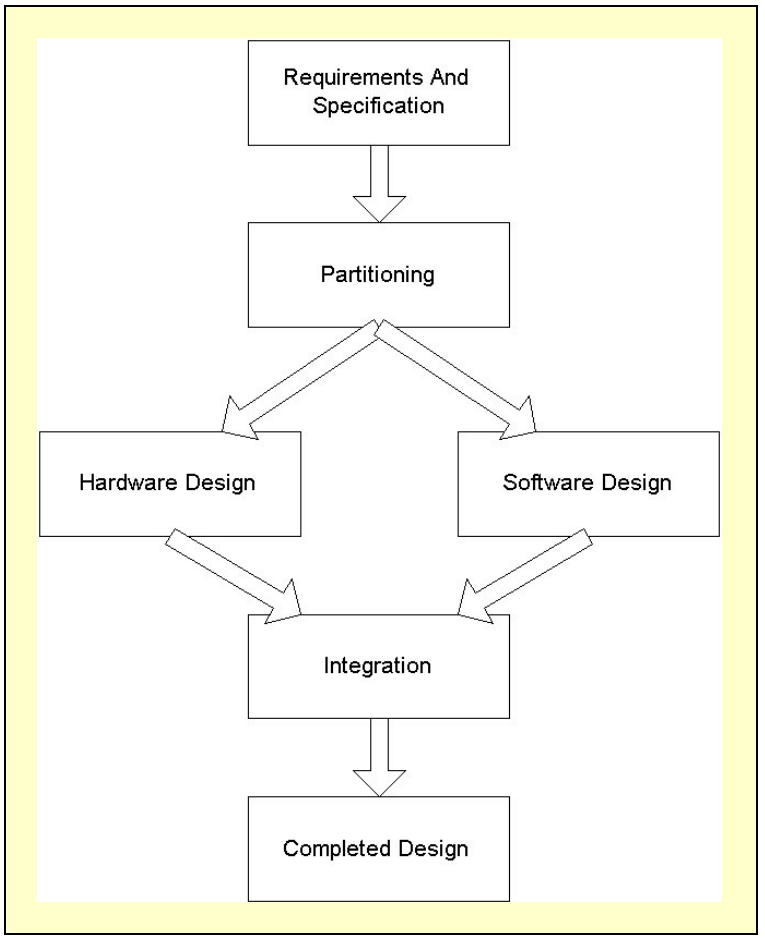
- A procedure for designing a system.
- Understanding your methodology helps you ensure you didn't skip anything.
- Compilers, software engineering tools, computer-aided design (CAD) tools, etc., can be used to:
  - Help automate methodology steps;
  - Keep track of the methodology itself.

Design goals

- Performance.
  Overall speed, deadlines.
- Functionality and user interface.
- Manufacturing cost.
- Power consumption.
- Other requirements
  (physical size, etc.)

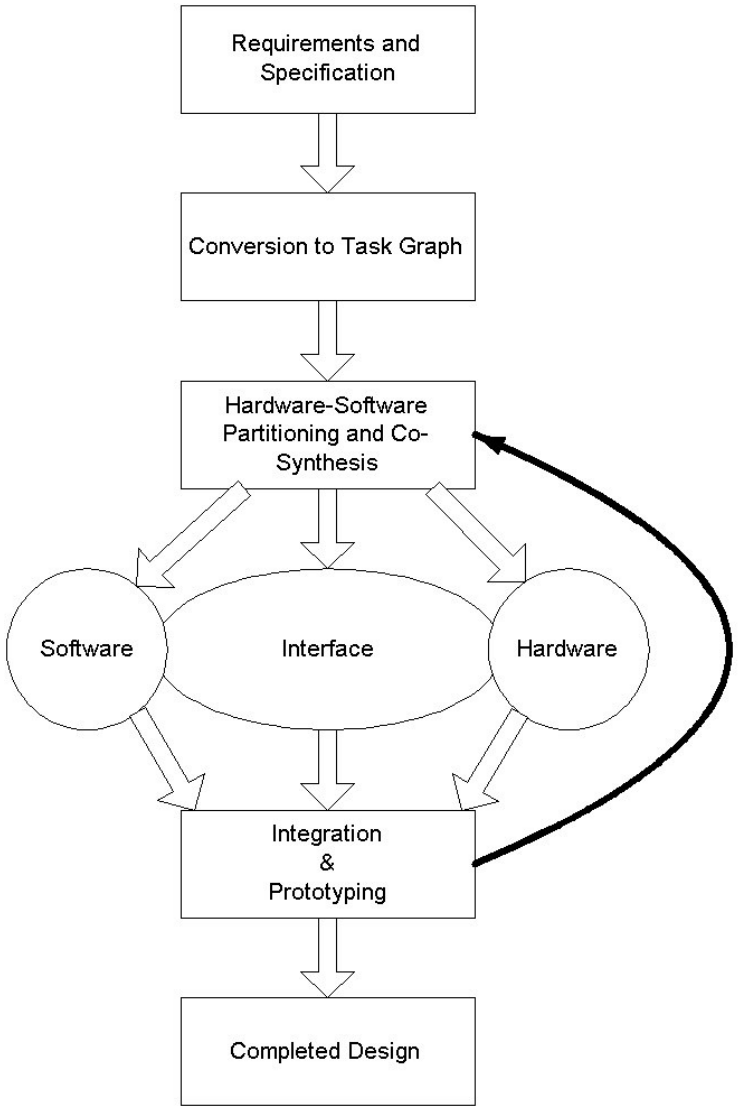
Traditional Embedded System Design

- HW/SW Partitioning performed at an early stage.
- Design mistakes have huge negative effect
- Inability to correct mistakes performed at the partitioning phase

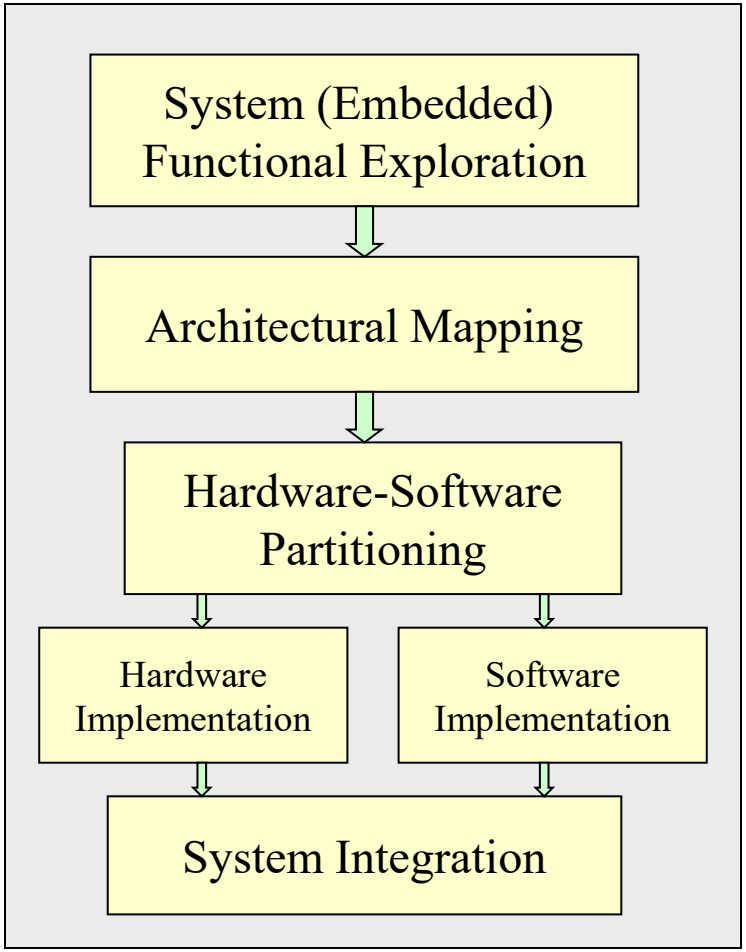


HW/SW Co-Design

- Codesign is divided into:
  - Co-specification
  - Co-synthesis
  - Co-simulation/Co-verification
- Key is the design refinement at hardware-software integration stage



Hardware-Software Codesign



- Functional exploration: Define a desired product's requirements and produce a specification of the system behavior.
- Map this specification
- Partition the functions between silicon and code, and map them
- Integrate system

Hardware-Software Codesign

Co-design of (embedded) computer systems encompassing the following parts:

- **Co-Specification**

  Developing system specification that describes hardware, software modules and relationship between the hardware and software

- **Co-Synthesis**

  Automatic and semi-automatic design of hardware and software modules to meet the specification

- **Co-Simulation and Co-verification**

  Simultaneous simulation of hardware and software

HW/SW Co-Specification

- Model the (embedded) system functionality from an abstract level.
- No concept of hardware or software
- Common environment

  SystemC: based on C++.
- Specification is analyzed to generate a task graph representation of the system functionality.

Co-Specification

- A system design language is needed to describe the functionality of both software & hardware.
- The system is first defined without making any assumptions about the implementation
- A number of ways to define new specification standards grouped in three categories:

➢ SystemC An open-source library in C++ that provides a modeling platform for systems with hardware and software components

SystemC for Co-specification

Open SystemC Initiative (OSCI) 1999 by EDA venders including Synopsys, ARM, CoWare, Fujitsu, etc.

- A C++ based modeling environment containing a class library and a standard ANSI C++ compiler.
- SystemC provides a C++ based modeling platform for exchange and codesign of system-level intellectual property (IP) models.

  - SystemC is not an extension to C++

    SystemC 2 has various versions

  It has a new C++ class library

  User needs to learn the use of new classes to model hardware design

SystemC Library Classes

SystemC classes enable the user to

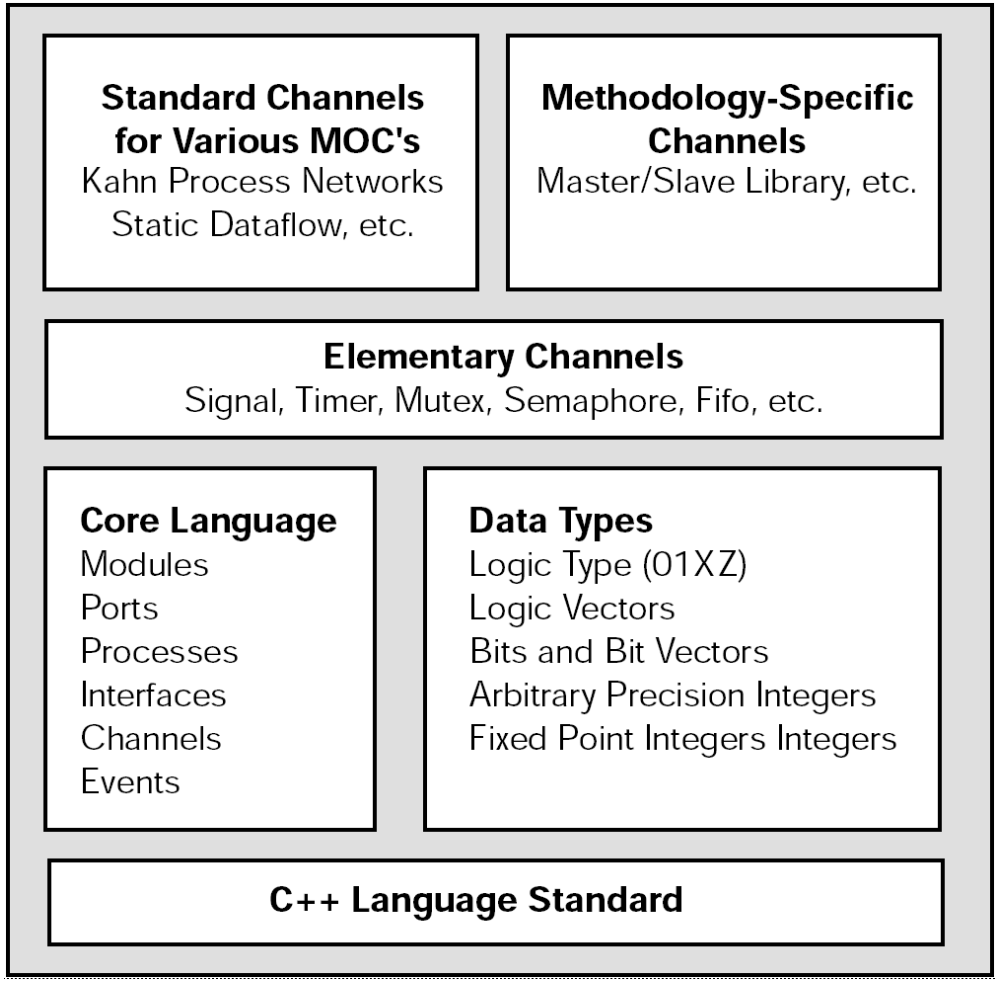
- Define modules and processes
- Add inter-process/module communication

Modules and processes can handle a multitude of data types: Ranging from bits to bit-vectors, standard C++ types to user define types like structures

Modules also introduce timing, concurrency and reactive behavior.

- Using SystemC requires knowledge of C and very little of C++ and the approach is similar to VHDL/Verilog.

SystemC 2.0 Language Architecture



SystemC 2.0 Language Architecture

- All of SystemC builds on C++
- Upper layers are cleanly built on top of the lower layers
- The SystemC core language provides a minimal set of modeling constructs for structural description, concurrency, communication, and synchronization.
- Data types are separate from the core language and user-defined data types are fully supported.
- Commonly used communication mechanisms such as signals and FIFOs can be built on top of the core language.

  The MOCs can also be built on top of the core language.

- If desired, lower layers can be used without needing the upper layers.

SystemC Benefits

SystemC 2.0 allows the following tasks to be performed within a single language:

- Complex system specifications can be developed and simulated
- System specifications can be refined to mixed software and hardware implementations
- Hardware implementations can be accurately modeled at all the levels.
- Complex data types can be easily modeled, and a flexible fixed-point numeric type is supported
- The extensive knowledge, infrastructure and code base built around C and C++ can be leveraged

Other Co-Specification Languages

UML

- Commonly used for modeling software systems.
- Supports many modeling specifications and profiles.

  e.g. UML Profile for real-time systems

ESTEREL

- To model reactive real-time systems where events trigger actions and vice versa.
- **http://www-sop.inria.fr/meije/esterel/esterel-eng.html**

SDL (systems, blocks, channels etc.)

- Formal Specification and Description Language.
- Often used to model real-time embedded systems.
- http://www.sdl-forum.org/

Hardware-Software Co-Synthesis

Four Principal Phases of Co-synthesis:

- **Partitioning**

  Dividing the functionality of an embedded system into units of computation.

- **Scheduling**

  Choosing time at which various computation units will occur.

- **Allocation**

  Determining the processing elements (PEs) on which computations will occur.

- **Mapping**

  Choosing component types for the allocated units (of computations).

HW/SW Co-Synthesis

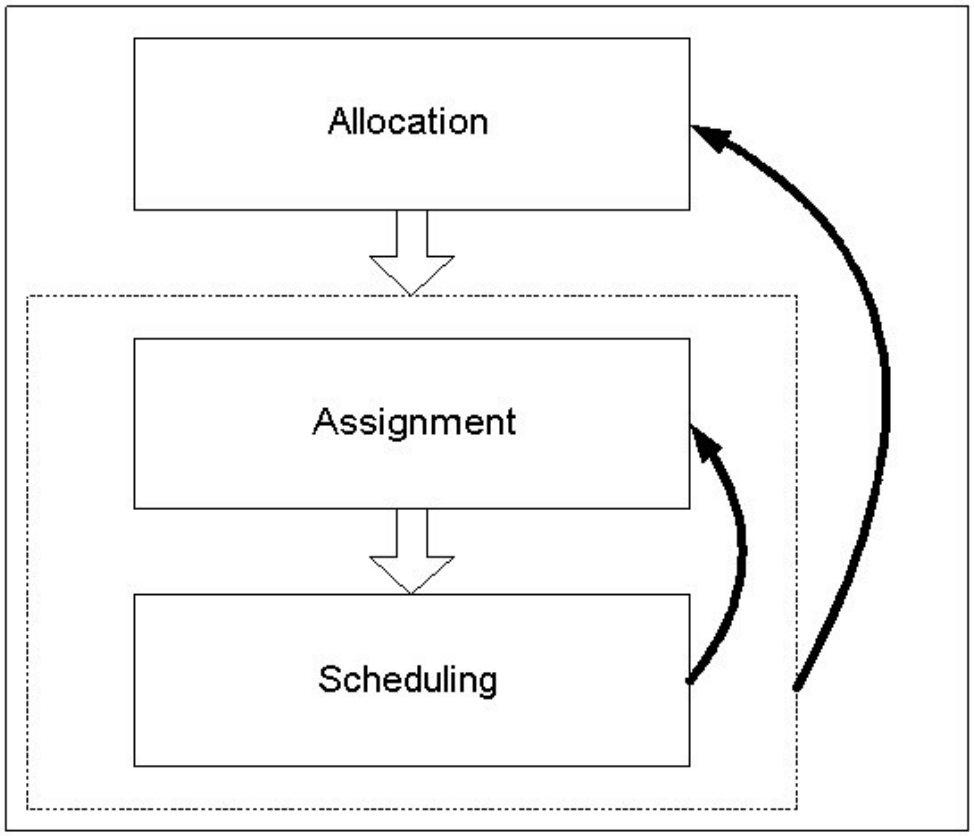
Automatically derive the system architecture

Tightly coupled with HW/SW Partitioning

Consists of three other stages:

- **Allocation:** select the number and type of communication links and processing elements for the target system.
- **Assignment (Mapping):** Mapping tasks to processing elements.
- **Scheduling:** Timing of task execution and communications.

Common Co-Synthesis Structure



System Partitioning

Introduces a design methodology that uses several techniques:

- **Partition the system specification into processes/tasks**

  The best way to partition a specification depends on the characteristics of the underlying hardware platform

- **Determine the performance of the function when executed on the hardware platform**

  We usually rely on approximating

- **Allocate processes onto various processing elements**

HW/SW Partitioning

- An area of significant research
- Analyzes task graph to determine each task's placement (HW or SW)
- Many algorithms have been developed.
- Major problem involves the computation time of partitioning algorithm

Hardware-Software Partitioning

Hardware/Software System Design involve: Modeling, Validation and Implementation

- **System implementation involves:**
  Hardware-Software Partitioning

  Finding those parts of the model best implemented in hardware & those best implemented in software.
- **Such partitions can be decided by the designer**
  with successive refinements
  or determined by the CAD tools

Hardware-Software Partitioning

For embedded systems, such partitioning represents a physical partition of the system functionality into:

- **Hardware**
- **Software executing on one or more CPUs**

Various formation of the Partitioning Problem that are based on:

- **Architectural Assumptions**
- **Partitioning Goals**
- **Solution Strategies**

COWARE: A design environment for application specific architectures targets telecom applications

Partitioning Techniques

Hardware-Software Homogeneous System Model => Task Graph

For each node of the task graph, determine implementation choices (HW or SW):

- Keep the scheduling of nodes at the same time
- Meet real-time constraints

- There is intimate relationship between partitioning and scheduling.
- Wide variation in timing properties of the hardware and software implementation of a task.

That effects the overall latency significantly

HW/SW Partitioning

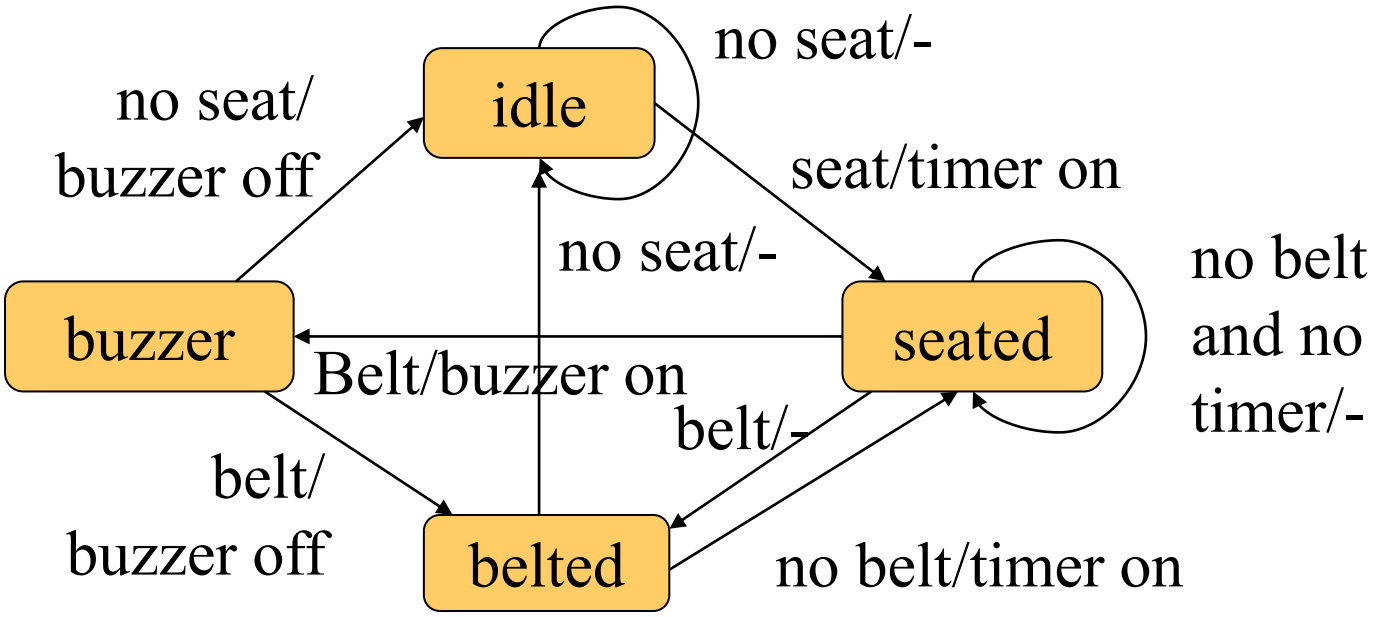
- Both textual and graphical representation like DAG are used to describe system.
- Analyzes task graph to determine each task's placement
- Many partitioning algorithms being developed
- Major problem involves the computation time of the algorithm.

System Design Patterns

Design Pattern: A generalized description of the design of a certain type of program that can also be used for system representation and hardware-software partitioning.

- State Diagram
- Data Flow Graph
- Control Data Flow Graph (CDFG)
- Directed Acyclic Graph (DAG) similar to DFG

State Machine: Seat-belt System




```
switch (state) {
    case IDLE: if (seat) { state = SEATED; timer_on = TRUE; } break;
    case SEATED: if (belt) state = BELTED;
                 else if (timer) state = BUZZER; break;
    ……
}
```

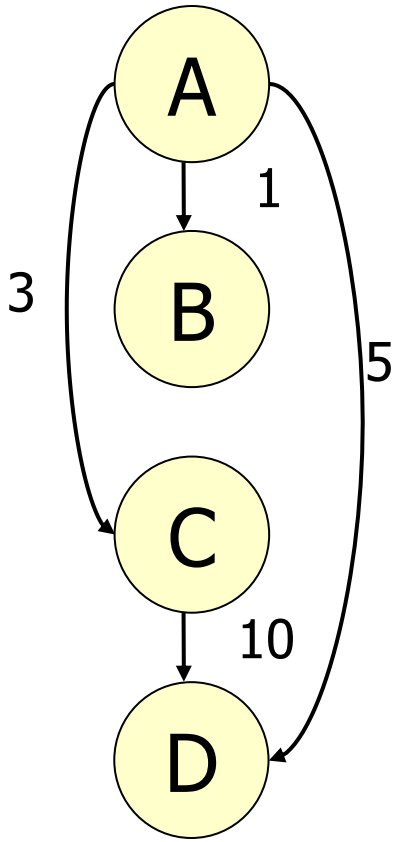
Data Flow Graph

DFG: Data Flow Graph

- DFG does not represent control
- It models the Basic Block: code or a system block with one entry and exit
- Describes the minimal ordering requirements on operations

DAG: Directed Acyclic Graph

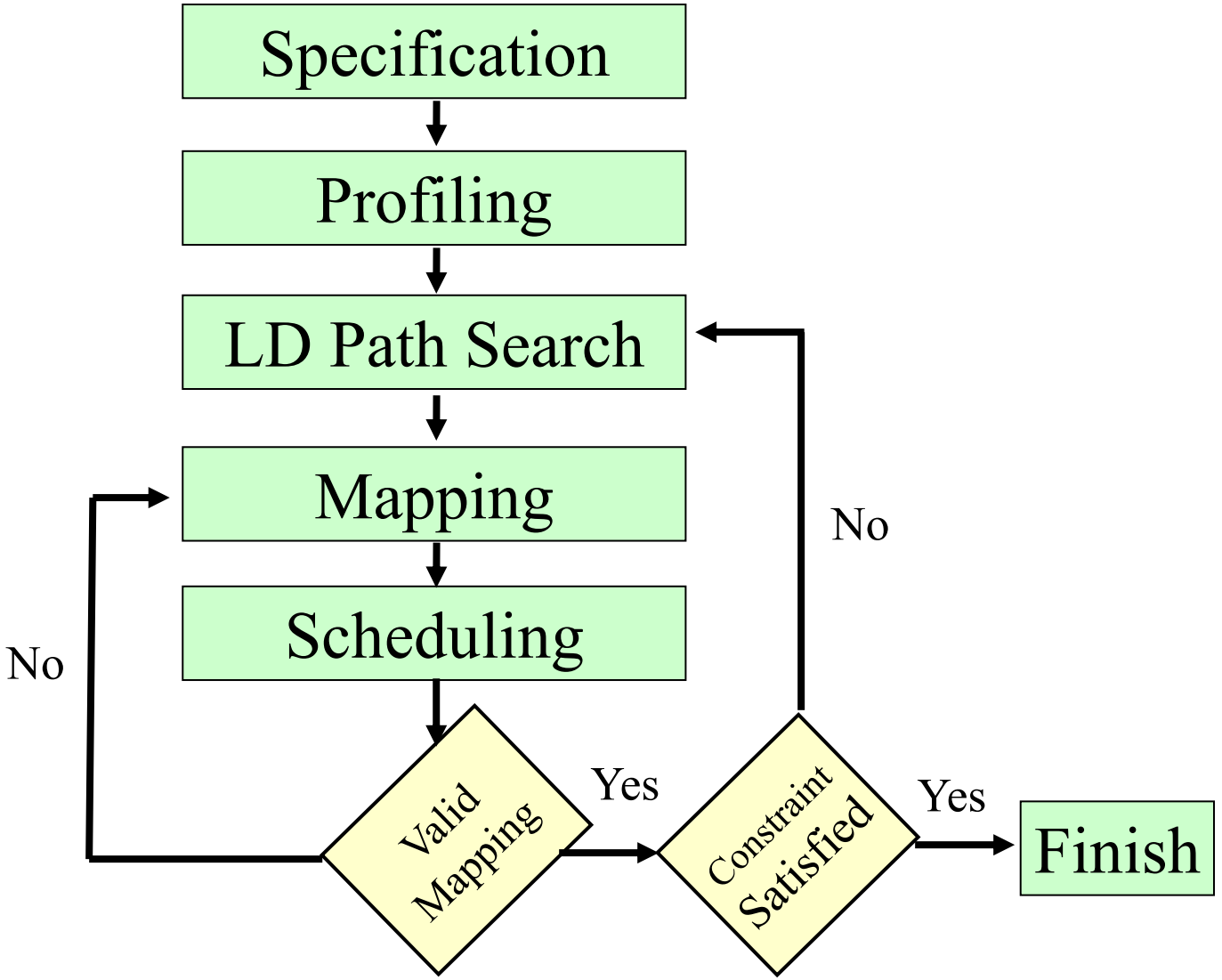
- Arrow represents dependence relationship
- Arc values represent communication time.
- Precedence dependency captures the order of execution between nodes and such nodes can be executed in parallel.
- Only necessary parallelism is exposed



Relevant Partitioning Research

- HW-SW Partitioning is a difficult.
- To find optimal partitioning set, it is very difficult due to many factors affecting the partitioning decision.
- A new partitioning Heuristics are being researched.
- HW/SW Partitioning based on DADGP, Directed Acyclic Data Dependency Graph with Precedence.
- Specified a new task-graph format with less restrictive types of communication links.

DAG-based Partitioning Structure



DAG-based Partitioning

i. Profiling and building an initial DAG
ii. Find the LD_path in DAG
iii. Mapping of LD-path nodes to hardware
iv. Schedule and if invalid mapping then goto Step iii
v. Update DAG and calculate the total execution time of target system.
vi. If system constraints are not met then goto Step ii, otherwise quit.

Profiling

Profiler collects the following data for each task node (module)

- Hardware/Software execution time
- Hardware Area
- Amount of data transfer
- Execution order
- Data dependencies between nodes

Longest Delay Path Search

Longest Delay path means, longest execution-time path

- Finding the longest delay path (LD-path) in DAG is equivalent to finding a bottleneck of the system.

- Minimizes search space for mapping

Mapping

- Maps a node/task to be implemented as a dedicated hardware unit
- Mapping can change the Longest Delay path, as well as DAG
- Mapping of a node/task is valid if implementing the node/task to Hardware gives the shortest LD-path in the modified DAG

Scheduling

- Very simple List-based scheduling approach.
- Schedules the earliest node (task) first without violating the resource limit.
- Exposes parallelism and changes the DAG accordingly.

DAG-based Scheduling

- Start scheduling from the root node of DAG.
- Traverse down the LD-path tree and schedule the earliest starting time node.
- If the node/task is not connected, check whether exposing parallelism is possible. Roots of the two DAGs are combined to form a single DAG with a dummy root node.
- In case of multiple descendants, schedule them forcibly by adding PEs.
- Update the PE resource (HW-SW) library.

Constraints

- Constraints of deadline and cost is given by the system designer.
- Hardware cost is calculated by the gate or transistor count.
- Different **granularity** level should be explored if no solution is found.

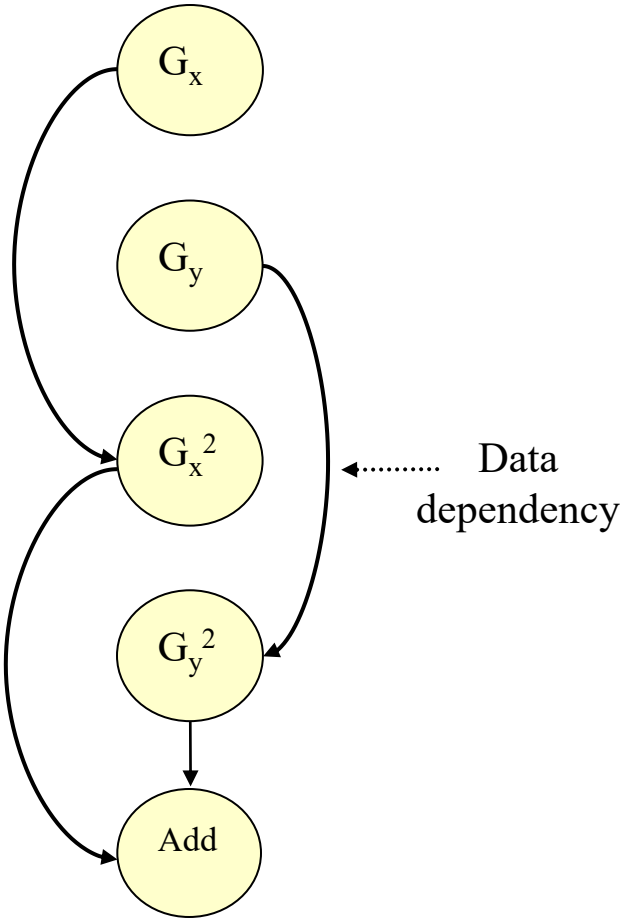
Edge Detection Example

Pair of masks are convolved to estimate gradients, G_x and G_y

Overall $\mathbf{G^2 = (G_x^2 + G_y^2)}$

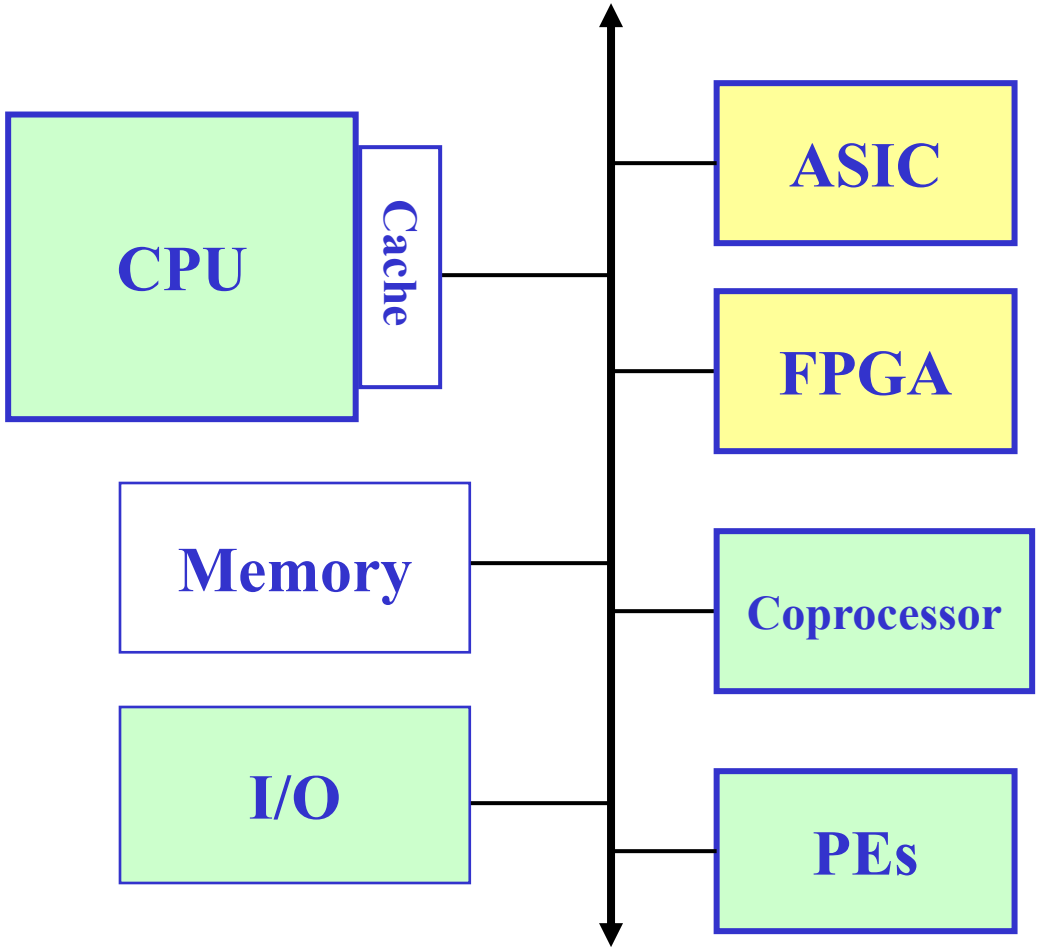
HW-SW Library

Operation	SW EXE (ms)	HW EXE (ms)	HW Area (gates)
Gradient (Gx or Gy)	9.4	1.4	1200
Square	5.2	0.9	500
Add	3.88	0.3	100



Embedded System Hardware Structure

Various Hardware Options



SOBEL Edge Detection

SOBEL masks

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Input Image

a_{11}	a_{12}	a_{13}		
a_{21}	a_{22}	a_{23}		
a_{31}	a_{32}	a_{33}		

Mask

m_{11}	m_{12}	m_{13}
m_{21}	m_{22}	m_{23}
m_{31}	m_{32}	m_{33}

Output Image

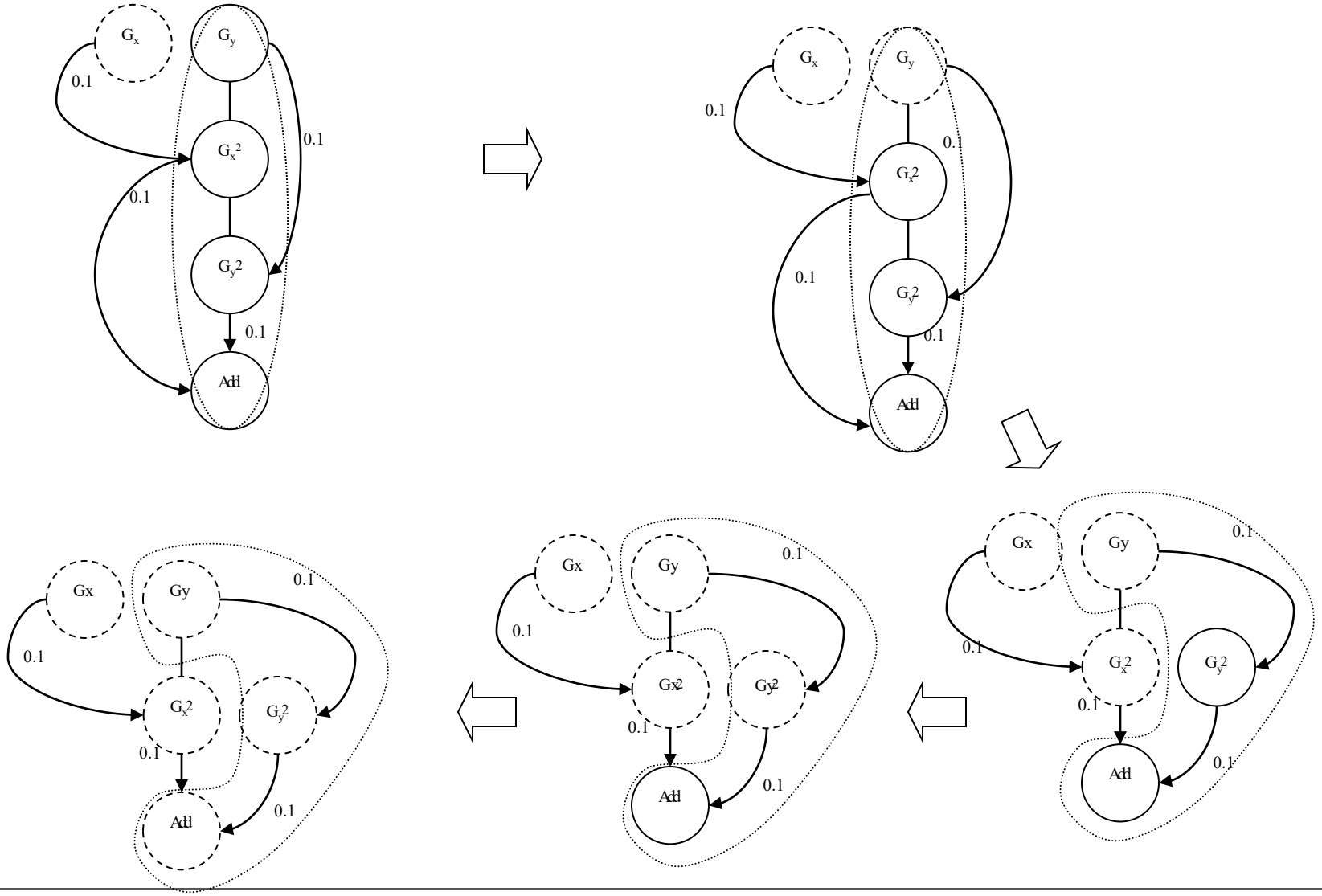
b_{11}	b_{12}	b_{13}		
b_{21}	b_{22}	b_{23}		
b_{31}	b_{32}	b_{33}		

$$b_{22}=(a_{11}*m_{11})+(a_{12}*m_{12})+(a_{13}*m_{13})+(a_{21}*m_{21})+(a_{22}*m_{22})+(a_{23}*m_{23})+(a_{31}*m_{31})+(a_{32}*m_{32})+(a_{33}*m_{33})$$

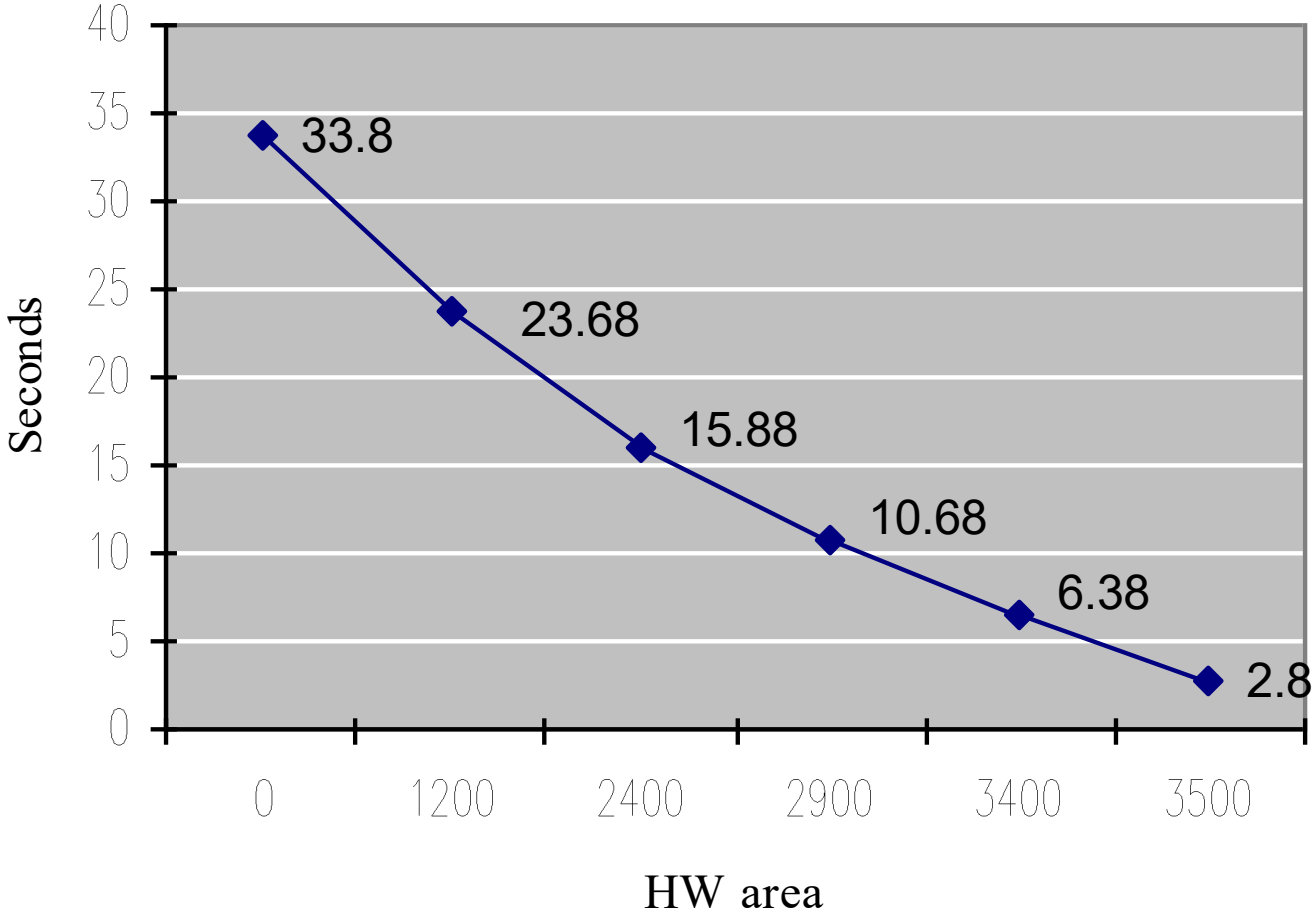
Sobel Edge Detection

```
main() {
unsigned char image_in[ROWS][COLS];
unsigned char image_out[ROWS][COLS];
int r, c; /* row and column array counters */
int pixel; /* temporary value of pixel */
         /*filter the image and store result in output array */
for (r=1; r<ROWS-1; r++)
 for (c=1; c<COLS-1; c++) {   /* Apply Sobel operator. */
   pixel = image_in[r-1][c+1]-image_in[r-1][c-1]
         + 2*image_in[r][c+1] - 2*image_in[r][c-1]
         + image_in[r+1][c+1] - image_in[r+1][c-1];
   /* Normalize and take absolute value */
   pixel = abs(pixel/4);
   /* Check magnitude */
   if (pixel > Threshold)
   pixel= 255; /*EDGE_VALUE;*/
   /* Store in output array */
   image_out[r][c] = (unsigned char) pixel;
 }
}
```

Edge Detection Solutions



Performance Improvement *vs.* HW area



HW/SW Co-Simulation

- Arguably the most important stage of Co-Design (prevent building faulty and expensive prototype)
- Can simulate entire system (Hardware and Software modules) before building the target system.
- Common co-simulation tool: Seamless, Coware, Eaglei, etc.
- Seamless allows various HW and SW debuggers to interact (respond to each other's events)

HW/SW Co-Verification

- Embedded system complexity prevents from relying on traditional validation techniques.
- Common approach involves Petri-net representation of system (PRES).
- PRES proves system correctness by determining truth of the computation tree logic.
- Other approaches exist – Seamless and other codesign tools are used for co-verification.

Conclusions

- Co-Design is getting mature
- Lot of research has been conducted recently and much more is to come in future.
- Research is being performed in all areas – specific attention is being addressed to system co-specification and co-synthesis.
- Used in SoC design and verification.