System Level Partitioning, Synthesis and Interfacing

Hardware Software CoDesign

October 2011

Agenda

Analysis and Estimation

1. Basic Themes that came out through the Historical Background (repeat)
2. System Level Partitioning, Synthesis and Interfacing
3. TextBook example of a single-purpose processor implementing the GCD program
4. Continue on "Answering Machine" Assignment Discussions. Give clear milestones.

Analysis and Estimation

Basic Themes

1. **Modeling** :: Because parallelism is important, the choice of an appropriate modeling paradigm is also important. Different modeling formalisms need to be developed that capture the various aspects of system behavior.

2. **Analysis and Estimation** :: Different models of the same system behavior need to be analyzed and estimated for performance. Performance would include tradeoffs for Area, Speed, Power. Cost to build, Time to Market are other factors.

3. **System-level partitioning, synthesis and interfacing** :: The basic steps of co-synthesis. A range of methodologies are applied for this.

4. **Implementation generation** :: Once the architecture has been generated and trade-offs known, the designs for the hardware and software components must be created.

5. **Co-simulation and Emulation** :: This helps designers to evaluate architectures and validate assumptions on implementations. Emulation uses FPGA / other emulation techniques to further speed up execution of system models.

System Level Partitioning, Synthesis and Interfacing

Introduction

1. A hardware / software partitioning problem can be stated as finding those parts of the model best implemented in hardware and those best implemented in software.
   - Partitioning can be decided by the designer, with successive refinement of initial model
   - It can also be determined by a CAD tool.
2. In the case of embedded systems, a hardware/software partition represents a physical partition of system functionality into application-specific hardware and software on one or more processor(s).
3. There are multiple techniques... We will discuss the paper by Kalavade and Lee.

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

1. Take a global view of the partitioning problem.

2. Assume that a homogeneous procedural model is compiled into task graphs.

3. Then determine the implementation choice (hw/sw) for each task graph node.

4. Also, scheduling these nodes at the same time so that real-time constraints are met.

5. Note that there is an intimate relation between partitioning and scheduling. This is caused by wide variation in timing properties of the hw/sw implementations of a task which affects the overall latency significantly.

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

1. The basic premises is that typically the designer needs to explore the possible options, tools and architectures.

2. A **design methodology management** framework helps to manage the design space exploration process. A software environment called **Design Assistant** addresses :-

   (a) Specific tools for partitioning, synthesis and simulation that are configured for a particular hw/sw codesign flow.

   (b) An underlying design methodology management infrastructure for design space exploration.

3. Each node in the task-level specification can be implemented in several ways in both hw/sw mappings.

4. At hw implementation it can be :-

   (a) *Algorithm level* :: Several algorithms can be used to describe the same task. Example a FIR filter can be implemented either as an inner product or using the FFT in a shif-and-add algorithm.

   (b) *Transformation level* :: For a particular algorithm, several transforamtions can be applied on the original task description.

   (c) *Resource level* :: A task, for a specified algorithm and transformation set, can be implemented using varying numbers of resource units.

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

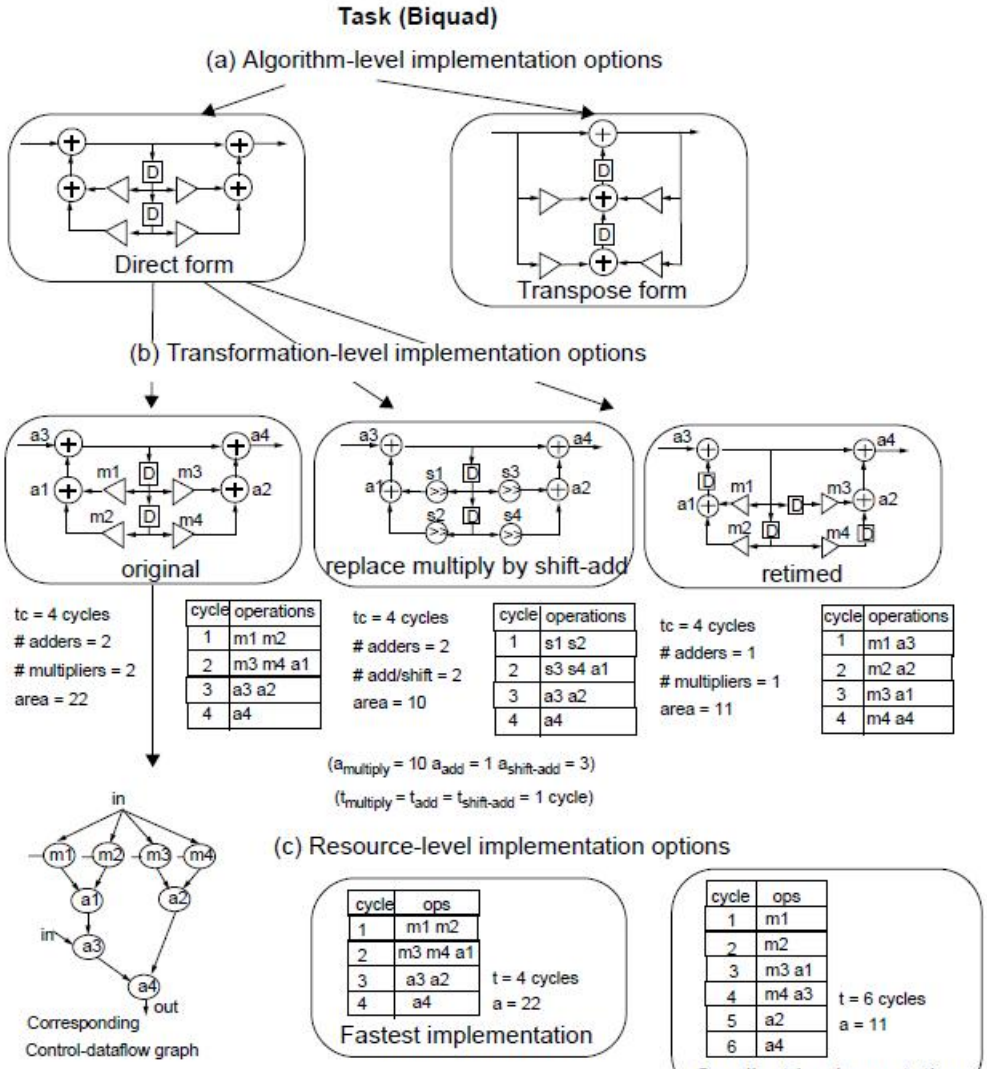


Figure 1. **Hardware design options for a "node".**

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

1. The goal of partitioning is to determine three parameters :: mapping (hw/sw), implementation (type wrt algorithm, transformation and area-time value) and schedule (when it executes wrt other tasks).

2. Partitioning is non-trivial problem :: consider a task-level specification, typically in the order of 50 to 100 tasks. Each task can be mapped into hw/sw. Given a mapping say there are 5 design options for each node, then there will be $(2 \cdot 5)^{100}$ design options. Say there are p preferred implementations, there are still a large number of design alternatives wrt the remaining nodes $(2 \cdot 5)^{100-p}$.

3. Partitioning again can be divided into two stages ::

   (a) *Binary partitioning* :: problem of determining hw/sw mapping and schedule.

   (b) *Extended partitioning* :: problem of selecting an appropriate implementation over and above binary partitioning.

4. We will discuss Binary partitioning in detail and Extended partitioning briefly.

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

The Partitioning Method

1. Serially traverse a node list (usually from the source node to the sink node in the DAG)

2. For each node select a mapping that minimizes an objective function. The objective function can be to minimize the finish time, or minimize the area (i.e, hw area or software size).

1. There are limitations :: The CAD algorithms tend to be greedy and hence can be suboptimal because they can either go for only finish time or only area.

2. To overcome the limitations :: adaptively select an appropriate mapping objective at each step to determine the mapping and schedule.

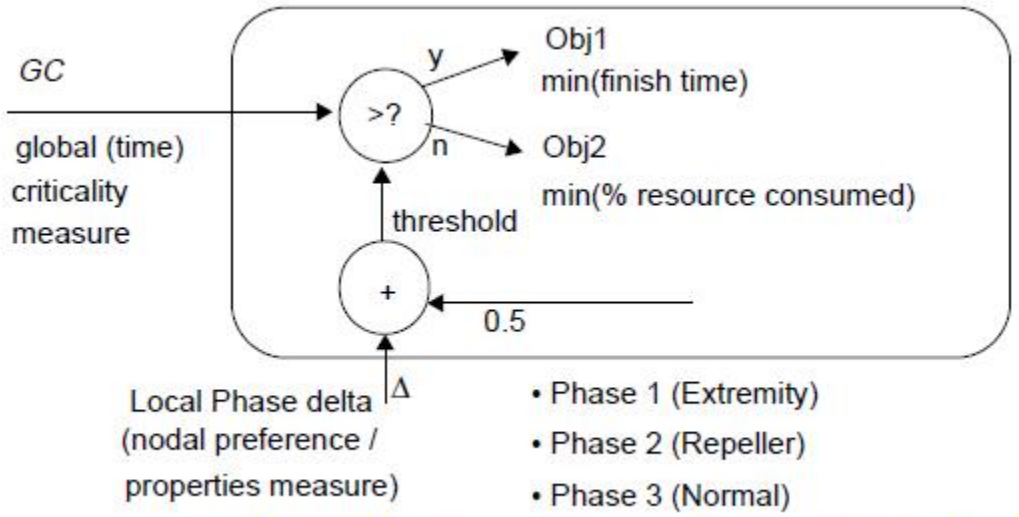
3. *Global Criticality GC* :: GC is a global look-ahead measure that estimates the time criticality at each step of the algorithm. GC is compared to a threshold to determine if time is critical. If time is critical, an objective function that minimizes finish time is selected, otherwise one that minimizes area is selected. GC can change at every step (node) of the algorithm.

4. *Local Phase LP* :: LP is a classification of nodes based on their heterogeneity and intrinsic properties. Each node is classified as an extermity (local phase 1), repeller (local phase 2) or normal (local phase 3) node. A measure called "local phase delta" quantifies the local mapping preferences of the node under consideration and accordingly modifies the threshold used in the GC comparison.

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

The Partitioning Method



Selection of the mapping objective at each step of GCLP.

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

The Partitioning Method

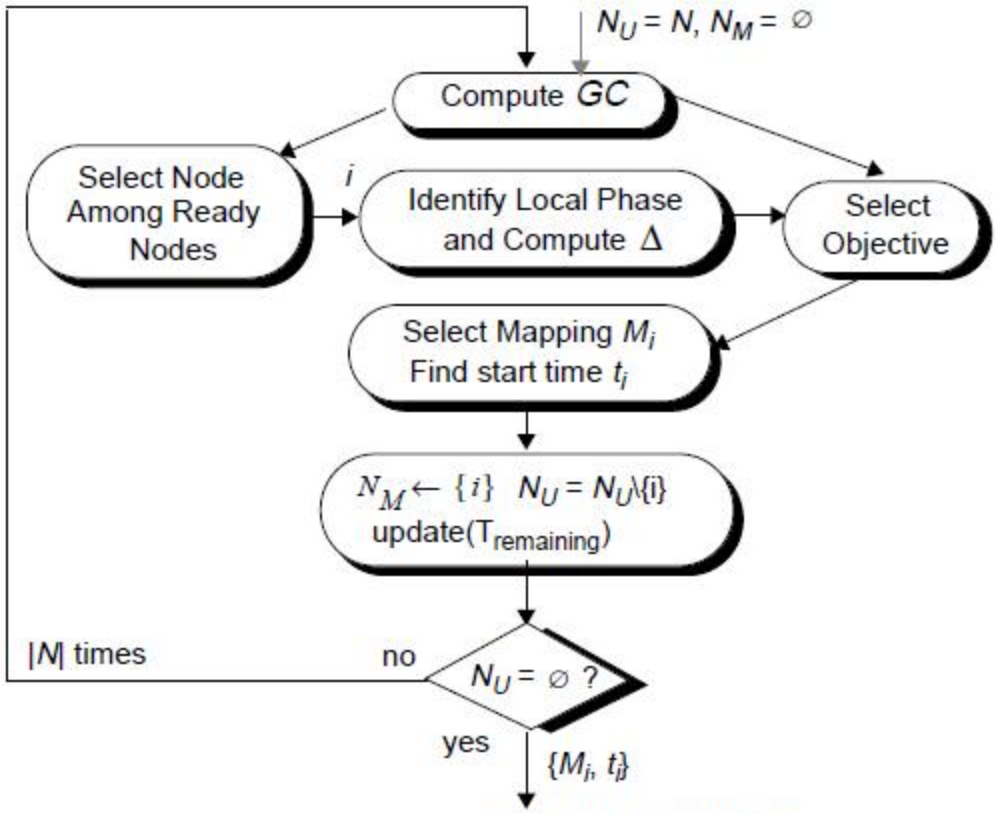
1. N_U = set of unmapped nodes at the current step. It is initialized to N the total nodes in the DAG.

2. N_M = set of mapped nodes at the current step.

3. Unmapped nodes whose predecessors have already been mapped and scheduled are called "ready" nodes.

4. A node is selected for mapping from the set of ready nodes using an urgency criteria. ie., a ready node that lies on the critical path is selected for mapping.

5. The local phase of the selected node is identified and the corresponding local phase delta is computed.

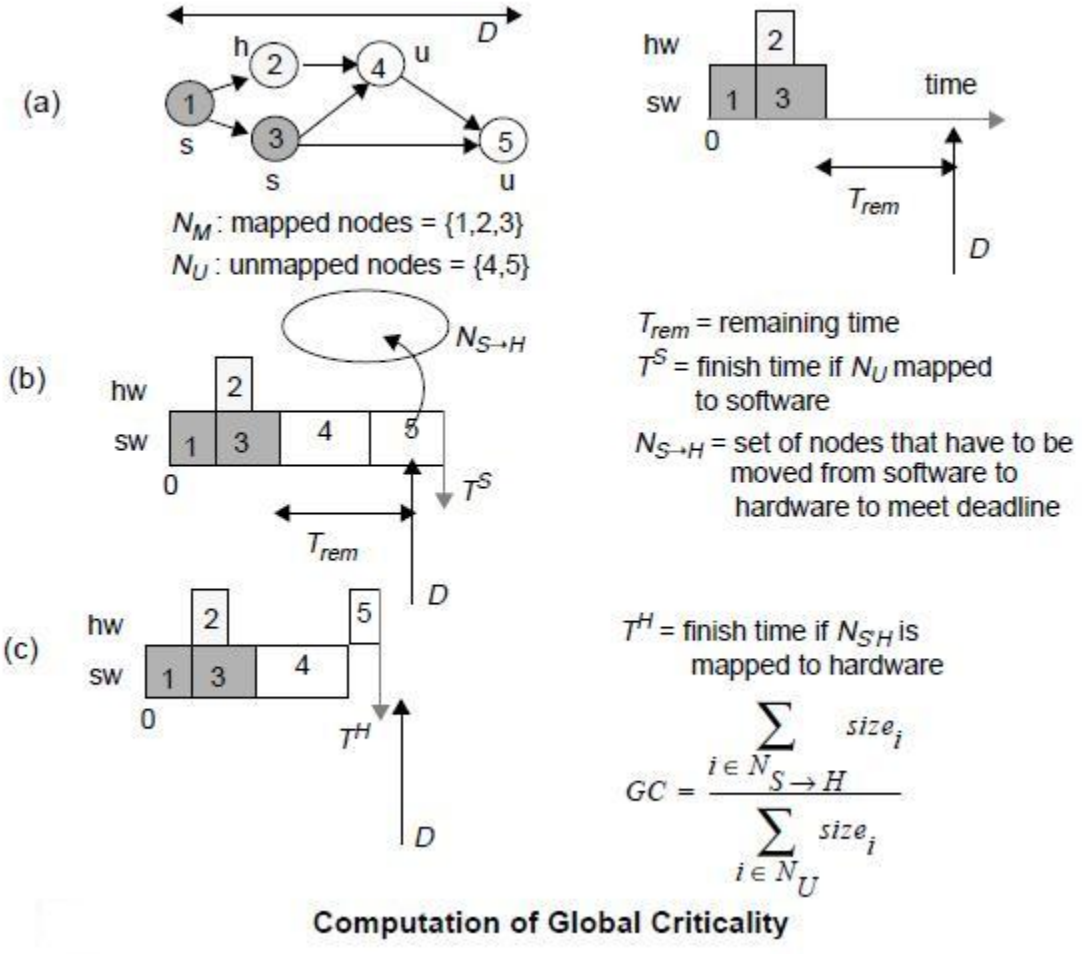


The GCLP Algorithm.

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

The Partitioning Method



Computation of Global Criticality

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

The Partitioning Method – Assumptions

1. Requirement of a DAG. The throughput constraint on the graph translates to a deadline D ie., the execution time of the DAG should not exceed D clock cycles. The DAG itself is a SDF (Synchronous Data Flow) graph.

2. The target architecture consists of a single programmable processor, which executes the software component and a custom datapath (the hardware component). The software and hardware components have capacity constraints – the software (program and data) size and should not exceed AS (memory requirements) and the hardware size should not exceed AH.

3. The area and time estimates for the hw/sw implementation bins are assumed to be known.

4. There is no reuse between nodes mapped to hw.

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

Extended Partitioning

1. The GCLP algorithm solves the binary partititon problem.

2. The Extended Partitioning problem is to jointly optimize the mapping as well as implementation bin for each node.

3. Consider an implementation bin set (usually a graph with implementation points) with L = fastest implementation and H = slowest implementation.

4. This is far more complex than the binary partition problem. The binary partition problem has $2^{|N|}$ mapping possibilities. Given B implementation bins within a mapping, the extended partitioning problem has $(2B)^{|N|}$ possibilities in the worst case.

5. It is not enough to decompose the extended partitioning problem into two isolated steps viz, mapping followed by implementation bin selection. WHY?
   - The serial traversal of nodes in the DAG means that the implementation bin of a particular node affects the mapping of as-yet-unmapped nodes.
   - There is a correlation between mapping and implementation-bin selection, they cannot be optimized in isolation.

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

Extended Partitioning Method (MIBS)

1. Let each node have attributes :: mapping, implementation bin, schedule.

2. As the algorithm progresses, depending on the extent of information that has been generated, each node goes through a sequence of three states :: (1) free, (2) tagged, (3) fixed.

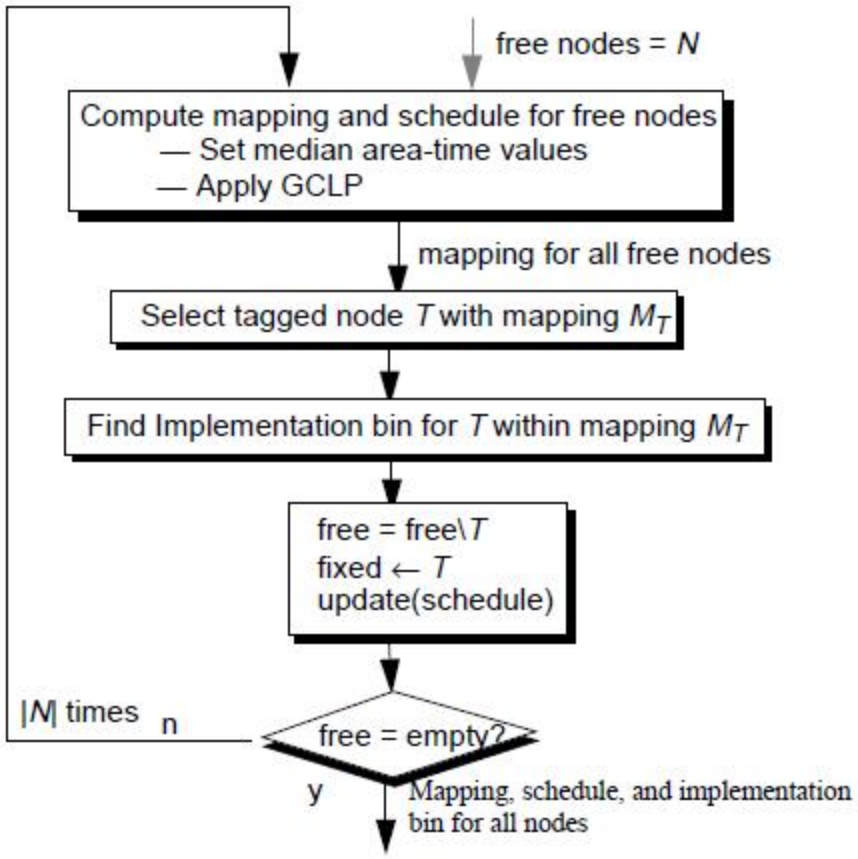
3. GCLP is first applied to get a mapping and schedule for all the free nodes in the DAG.

4. A particular free node (called a "tagged" node) is then selected.

5. Assume its mapping to be that determined by GCLP, an appropriate implementation bin is then chosen for the tagged node. HOW ??

6. Once the mapping and implementation bin are known, the tagged node becomes a "fixed" node.

7. GCLP is applied on the remaining nodes and this process is repeated until all nodes in the DAG become "fixed".



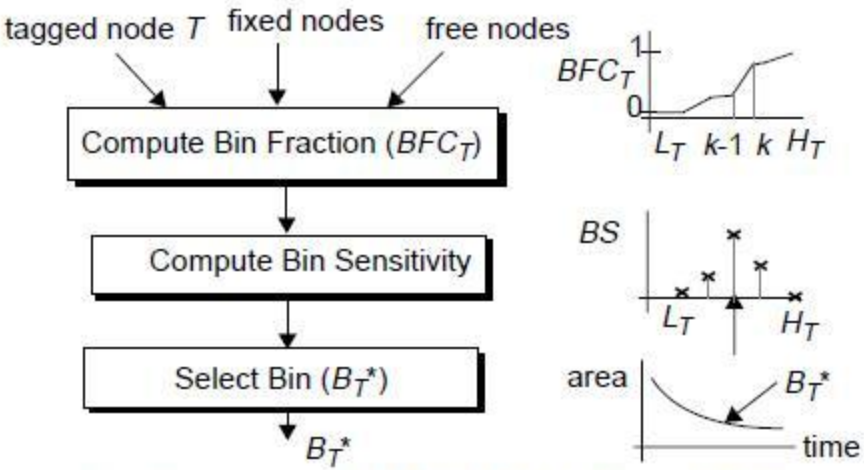
MIBS approach to solving extended partitioning

System Level Partitioning, Synthesis and Interfacing

Kalavade and Lee (Kal97)

Extended Partitioning Method (MIBS) – Implementation Bin Selection

1. Assume its mapping to be that determined by GCLP, an appropriate implementation bin is then chosen for the tagged node. HOW ??
   (a) Let the free nodes mapped to hw at the current step be called $free^h$ nodes. (Same can be extended to software implementation bin also).
   (b) Select the most responsive bin in this respect as the implementation bin for the tagged node.
   (c) The key idea is to use a look-ahead measure to correlate the implementation bin of the tagged node with the hardware area required for the $free^h$ nodes.
   (d) The look-ahead measure (called bin fraction BT_T^j) computes, for each bin j of the tagged node T, the fraction of $free^h$ nodes that need to be moved from H bins to the L bins in order to meet timing constraints.
   (e) A high value of BT_T^j indicates that if the tagged node T were to be implemented in bin j, a large fraction of $free^h$ nodes would likely get mapped to their fast implementations (L bins), hence increasing the overall area (or memory).
   (f) The bin fraction curve BFC_T is the collection of all bin fraction values of the tagged node T
   (g) The bin sensitivity is the gradient of BFC_T, which reflects the responsiveness of the bin fraction to the bin motion of node T.



The bin selection procedure

System Level Partitioning, Synthesis and Interfacing

Summary of other Methodologies / Algorithms

Polis System :: Hardware-Software Codesign of Embedded Systems

1. Chiodo et. all (Chi94) describe the Polis System where designs are described as networks of co-design finite state machines (CFSMs).

2. A CFSM designś components are assigned to implementation in either hw or sw.

3. Hardware units are fully synchronous.

4. Each software component is implemented as a standalone C program.

System Level Partitioning, Synthesis and Interfacing

Summary of other Methodologies / Algorithms

Heterogeneous MPU systems :: SOS Synthesis of Application-Specific Heterogeneous Multiprocessor Systems

1. Prakash and Parker (Pra92) describe a formal method of design, based on mixed-ILP programming model. The basics include :-

2. Synthesis of hardware units involves several subtasks.

3. First and foremost is operation scheduling, which may or maynot be combined with system level partitioning.

4. Different scheduling approaches are used, often borrowed from the OperatingSystems and Real-Time Systems concepts.

5. The method provides a static schedule, as well as an assignment of tasks to processors.

System Level Partitioning, Synthesis and Interfacing

Summary of other Methodologies / Algorithms

Wolf (Wol97) :: An Architectural co-Synthesis Algorithm for Distributed, Embedded Computing Systems

1. This paper addresses hw/sw co-synthesis in distributed systems.

2. Describes a method to simultaneously synthesize the hardware and software architectures of a distributed system to satisfy performance requirements and minimize cost.

3. The hardware consists of a network of processors with an arbitary communication topology.

4. The software consists of an allocation of processes to processors and a schedule for the processes.

Example of single-purpose processor implementing the GCD program

Problem Statement

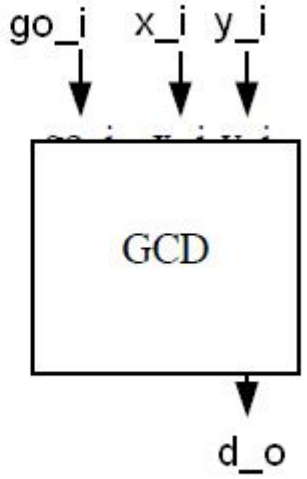
1. Take a break for 10mins.

Example of single-purpose processor implementing the GCD program

Problem Statement

1. The system's functionality :: the output should represent the GCD of the inputs
2. i.e, if 12, 8 are inputs, output should be 4. If 13 and 5 are inputs output should be ??
3. GROUP ACTIVITY :: Is the below algorithm actually for GCD. Are there better ways ??
4. GCD algorithm and entity description of the implementation

```
0: vectorN  x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while  (x != y)  {
6:      if  (x < y)
7:         y = y - x;
       else
8:         x = x - y;
     }
9:   d_o = x;
   }
```



Example of single-purpose processor implementing the GCD program

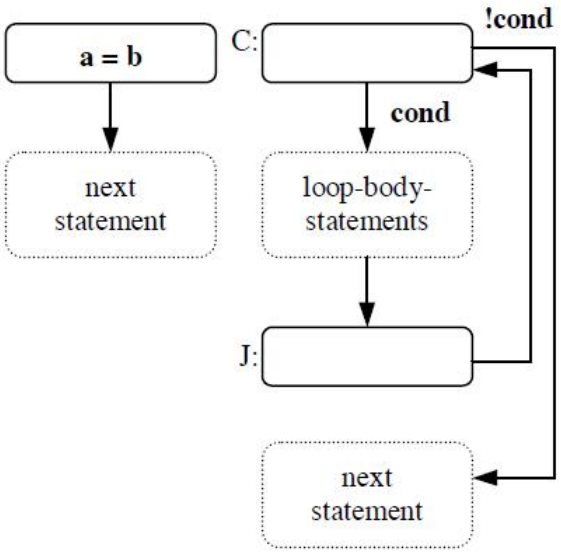
Building Blocks

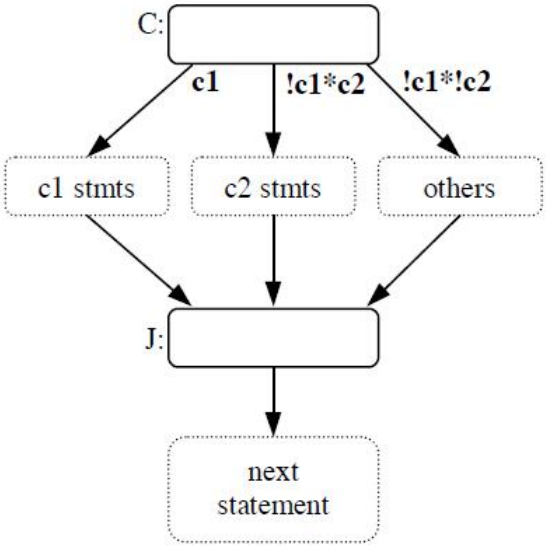
1. First convert the program into a (complex) state diagram, in which states and arcs may include arithmetic expressions, and these expressions may use external inputs, outputs, or variables.

Templates for creating a state diagram from a program.

Assignment statement	Loop statement	Branch statement

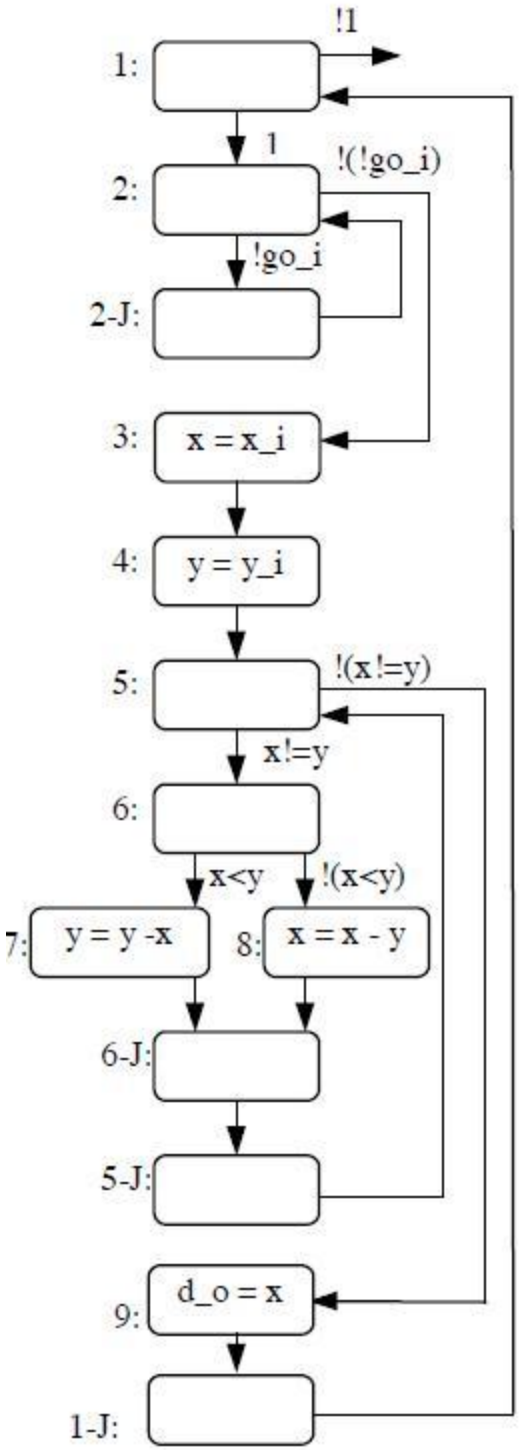
a = b
next statment
	while (cond) {
	loop-body-statements
	}
	next statement
		if (c1)
		c1 stmts
		else if c2
		c2 stmts
		else
		other stmts
		next statement





Example of single-purpose processor implementing the GCD program

Use Building Blocks to convert to a State Diagram




```
0: vectorN  x, y;
1: while (1) {
2:   while (!go_i);
3:   x = x_i;
4:   y = y_i;
5:   while  (x != y)  {
6:      if  (x < y)
7:         y = y - x;
       else
8:         x = x - y;
     }
9:   d_o = x;
   }
```

Example of single-purpose processor implementing the GCD program

From State Diagram – BreakUp the functionality into datapath and a controlpath

1. Create a register for any declared variable. (x, y). For registered outputs create d as register

2. Create a functional unit for each arithmetic operation in the state diagram. ie., 2 subtractors, 2 comparators one for lessThan (lt), one for nonEq (neq).

3. Connect the ports, registers and functional units.

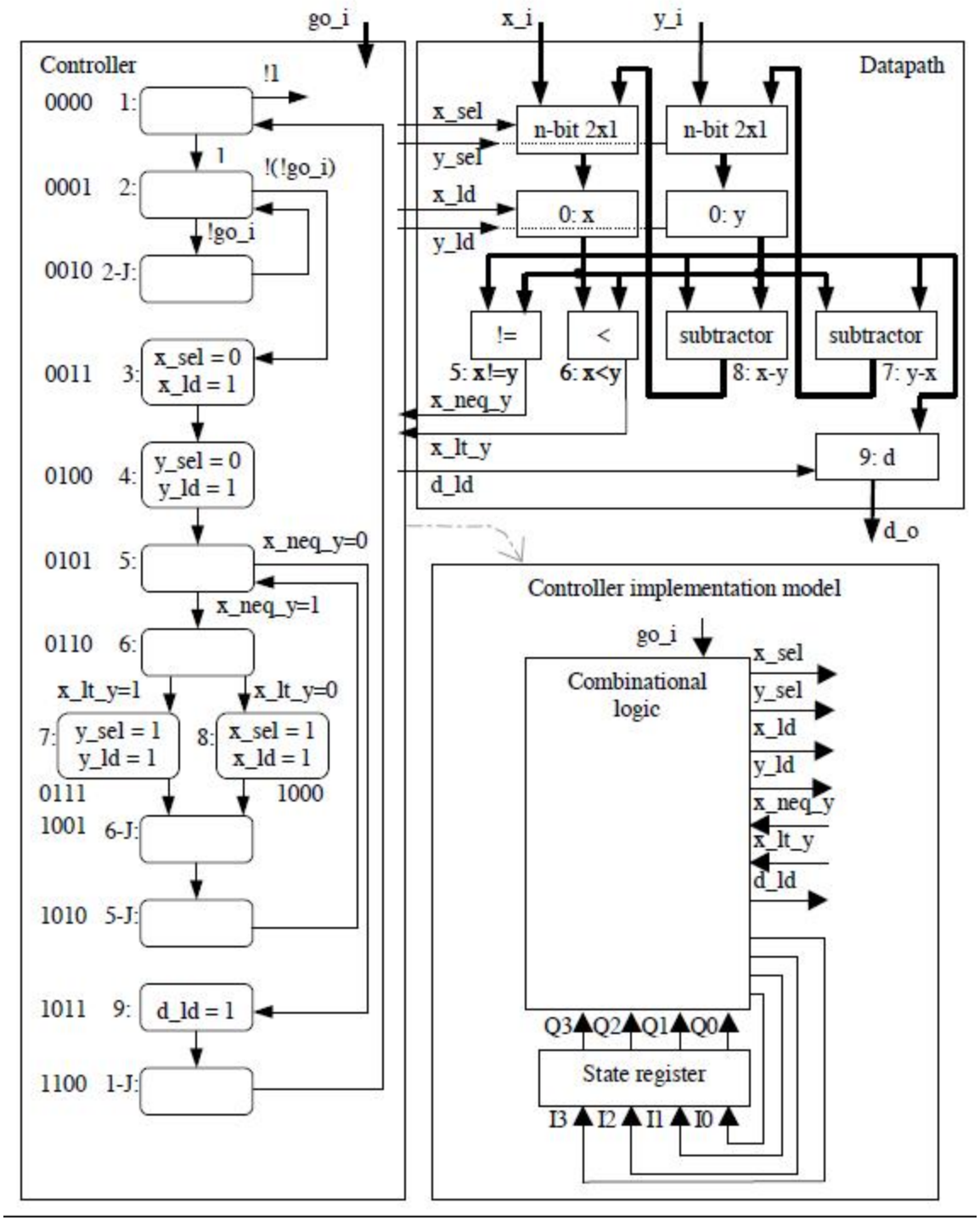
   (a) For each write to a variable in the state diagram, draw a connection from the writeś source (input port, functional unit, or another register) to the variableś register.

   (b) For each arithmetic and logical operation, connect sources to an input of the operationś corresponding functional unit.

   (c) When more than one source is connected to a register, add an appropriate mux.

4. Create a unique identifier for each control input and output of the data components.

Example after splitting into a controller and a datapath.



Example of single-purpose processor implementing the GCD program

From State Diagram – Build the StateTable for registers and IO's

State table for the GCD example.

State table

Inputs							Outputs								
Q3	Q2	Q1	Q0	x_neq_y	x_lt_y	go_i	I3	I2	I1	I0	x_sel	y_sel	p_ld_x	p_ld_y	p_ld_p
0	0	0	0	*	*	*	0	0	0	1	X	X	0	0	0
0	0	0	1	*	*	0	0	0	1	0	X	X	0	0	0
0	0	0	1	*	*	1	0	0	1	1	X	X	0	0	0
0	0	1	0	*	*	*	0	0	0	1	X	X	0	0	0
0	0	1	1	*	*	*	0	1	0	0	0	X	1	0	0
0	1	0	0	*	*	*	0	1	0	1	X	0	0	1	0
0	1	0	1	0	*	*	1	0	1	1	X	X	0	0	0
0	1	0	1	1	*	*	0	1	1	0	X	X	0	0	0
0	1	1	0	*	0	*	1	0	0	0	X	X	0	0	0
0	1	1	0	*	1	*	0	1	1	1	X	X	0	0	0
0	1	1	1	*	*	*	1	0	0	1	X	1	0	1	0
1	0	0	0	*	*	*	1	0	0	1	1	X	1	0	0
1	0	0	1	*	*	*	1	0	1	0	X	X	0	0	0
1	0	1	0	*	*	*	0	1	0	1	X	X	0	0	0
1	0	1	1	*	*	*	1	1	0	0	X	X	0	0	1
1	1	0	0	*	*	*	0	0	0	0	X	X	0	0	0
1	1	0	1	*	*	*	0	0	0	0	X	X	0	0	0
1	1	1	0	*	*	*	0	0	0	0	X	X	0	0	0
1	1	1	1	*	*	*	0	0	0	0	X	X	0	0	0

* - indicates all possible combinations of 0's and 1's

X - indicates don't cares

Example of single-purpose processor implementing the GCD program Other Optimizations ??

1. Could the algorithm be modified to yield lower / predictable latency (better algorithm)
2. Can you re-use subtractors by adding muxes and scheduling them (resource sharing)
3. Optimizing the FSM (state minimization)

Example of Dedicated H/W for matrix multiplication

Discuss the design of a Matrix Multiplication H/W $A(3x2) \cdot B(2x3) = C(3x3)$

```
main () {
  int A[3][2] = { {1,2}, {3,4}, {5,6}};
  int A[2][3] = { {7,8,9}, {10,11,12}};
  int C[3][3];
  int i, j, k;
  for (i=0; i < 3; i++) {
    for (j=0; i < 3; j++) {
      C[i][j] = 0;
      for (k=0; i < 2; k++) {
        C[i][j] += A[i][k] * B[k][j];
      }
    }
  }
}
```

Continue on "Answering Machine" Assignment Discussions

Marking Milestones

1. Take a break for 5mins.

2. Implementation Documentation – Hardware and Software components, with all IPś in place, assumptions et. all → October 30 Deadline (10 marks)

3. Actual Implementation, documentation updates and enabling me to replay with my own set of tests → December 30 Deadline (10 marks)

4. Selection of key persons from the team (groups)
   (a) IP creation / reuse
   (b) System Hardware Integration
   (c) System Software Integration
   (d) TestBench Creation and System Checkout → Hardware & Software
   (e) Total Hardware / software → bringing it all together

Continue on "Answering Machine" Assignment Discussions

Requirements Till date

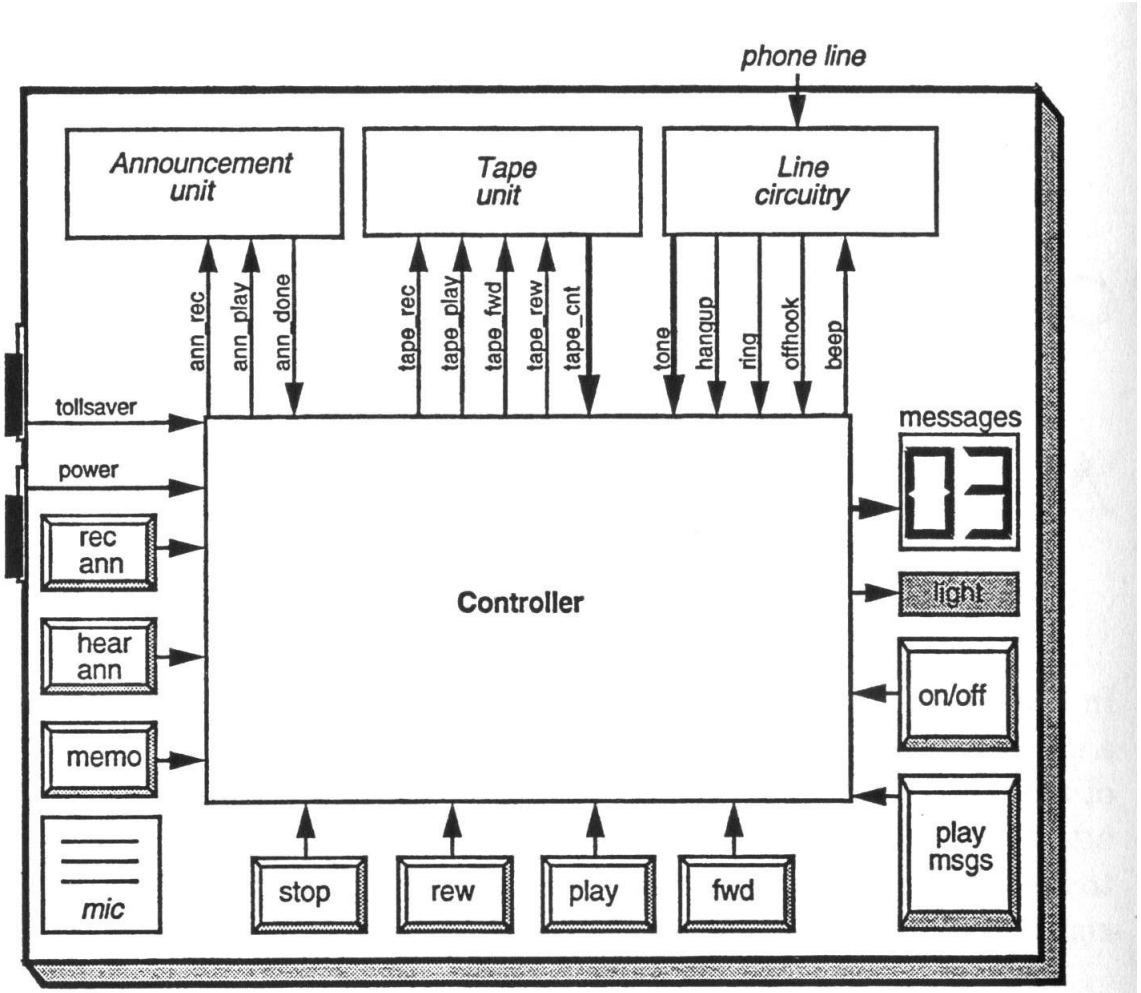
1. System C model of the system.
2. Use models for analog components.
3. Should have System C testbench as a part of SoC verification.
4. ReUse preferred to creation of already existing components.

Solution components / discussion

1. Decide Microprocessor.
2. How to write software program.
3. Memory requirements.
4. Model of ADC required (file input and digital output).
5. Input Buttons.
6. On/Off Button - special case.
7. Model of LEC (7 segment display).
8. Some real time clock – keep date and time (simplification is ok)
9. Each message has upper bound (10sec) – min 16 messages.
10. Announcement = single 10 seconds.
11. Announcement hear out through microphone (DAC)
12. Light keeps blinking if there is a new message.

"Answering Machine" Assignment

Diagram of Use Case



The answering machine controller's environment.

”Answering Machine” Assignment

Next Steps for Implementation document

1. Entity Definition
2. Processor / Dedicated Hardware decision
3. System State Machine or StateChart diagram (gross functionality)
4. Individual functionality State Macine or StateChart diagram

System Level Partitioning, Synthesis and Interfacing

Acknowledgements

1. Readings in Hardware / Software Co-design ::G. D. Micheli, Rolf Ernst, Wayne Wolf :: Ch 4

2. Embedded System Design - A Unified Hardware / Software Introduction :: Ch 2.4

3. The Extended Partitioning Problem : Hw/Sw Mapping, Scheduling, and Implementation bin selection :: Asawaree Kalavade, Edward Lee :: Proceedings of the 6th International Workshop on Rapid Systems Prototyping (1997).

4. Reference :: An efficient Hardware Design Tool for Scalable Matrix Multiplication :: Semih Aslan, Christophe Desmouliers, Erdal Oruklu and Jafar Saniie :: Electrical & Computer Engineering Dept (Illinois Institute of Technology).