Co-Simulation and Emulation

Hardware Software CoDesign

October 2011

Agenda

Co-Simulation and Emulation

1. Basic Themes that came out through the Historical Background (repeat)

2. Co-Simulation and Emulation

3. Continue on "Answering Machine" Assignment Discussions. Give clear milestones.

4. Taking Stock

Co-Simulation and Emulation

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.

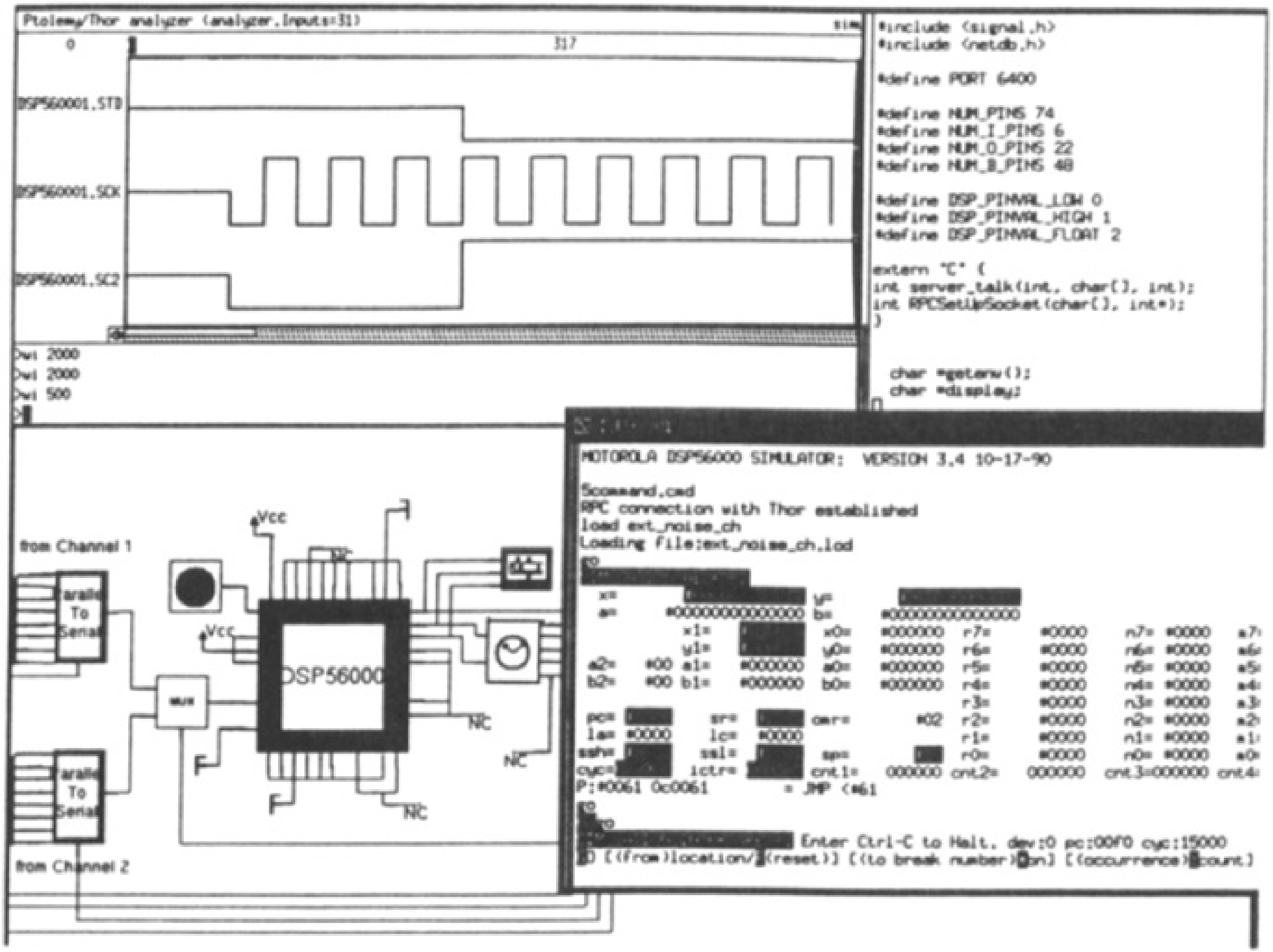
3. **System-level partitioning, synthesis and interfacing** :: The basic steps of co-synthesis. A range of methodologies are applied for this.

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.

Co-Simulation and Emulation

Screen Shot of CoSimulation System



```
Dwi 2000
Dwi 2000
Dwi 500
```

```
#include <signal.h>
#include <netdb.h>

#define PORT 6400

#define NUM_PINS 74
#define NUM_I_PINS 6
#define NUM_O_PINS 22
#define NUM_B_PINS 48

#define DSP_PINVAL_LOW 0
#define DSP_PINVAL_HIGH 1
#define DSP_PINVAL_FLOAT 2

extern "C" {
int server_talk(int, char[], int);
int RPCSetUpSocket(char[], int*);
}

  char *getenv();
  char *display;
```

```
MOTOROLA DSP56000 SIMULATOR:  VERSION 3.4 10-17-90

Scommand.cmd
RPC connection with Thor established
load ext_noise_ch
Loading file:ext_noise_ch.lod

   x=                          y=
   a=      #00000000000000 b=      #00000000000000
           x1=             x0=     #000000  r7=   #0000   n7= #0000   a7
           y1=             y0=     #000000  r6=   #0000   n6= #0000   a6
  a2=  #00 a1=   #000000   a0=     #000000  r5=   #0000   n5= #0000   a5
  b2=  #00 b1=   #000000   b0=     #000000  r4=   #0000   n4= #0000   a4
                                            r3=   #0000   n3= #0000   a3
  pc=          sr=         omr=        #02  r2=   #0000   n2= #0000   a2
  la=  #0000   lc=  #0000                   r1=   #0000   n1= #0000   a1
 ssh=          ssl=        sp=              r0=   #0000   n0= #0000   a0
 cyc=          ictr=       cnt1=    000000 cnt2=  000000  cnt3=000000 cnt4
P:#0061 0c0061          = JMP <#61

                               Enter Ctrl-C to Halt. dev:0 pc:00f0 cyc:15000
D [(from)location/(reset)] [(to break number)on] [(occurrence)count]
```

Co-Simulation and Emulation

Introduction

1. *Hardware-software co-simulation* is the combination of simulation of software running on a processor hardware with the fixed-function hardware components / sub-systems.

2. From here on.. there can be various methods for co-simulation.

   (a) Do a detailed processor simulation (full RTL) $\rightarrow$ too time consuming when simulating large software programs, and too slow to bring existing components to work with simulated components. Thus there is a requirement for a set of models for existing components (called test-bench). The creation, maintenance and usage of testbench adds to the cost of the device which is being designed.

   (b) For simulating larger software programs, abstract processor models are used. (usage of behavioral models for processors).

   - *Bus Functional Model (BFM)* :: abstracts from program execution and describes the processor bus interface function and timing only (think of a donut).
   - *Cycle-accurate Model* :: executes the program instructions with the accurate number of processor clock cycles, but without detailed interface timing. Such a model allows the designer to analyze the system timing and to validate the cooperation of hardware components and processors.
   - *Instruction Set Simulator Model* :: executes the program instructions preserving the program function but completly abstracts from timing. Main applications are program validation and debugging the software program.

3. Cycle-accurate and Instruction Set simulator are well suited to compiled mode (say nc)simulation.

4. Abstract Models are simpler, faster to execute, and can be used in early phases of a design where implementation details are still open.

5. Co-simulation uses abstract models to form a virtual prototype, *co-emulation* provides a real prototype by function implementation in hardware. This prototype can be used to accelerate co-simulation, but it can also be installed in the real environment *like a hardware in the loop* to investigate the system function under real conditions.

6. The holy grail for co-simulation and emulation :: Investigation of various possibilities with different tradeoffs in hw/sw is only sensible if different hw partitions can be implemented as fast as software can be compiled into machine code. High Level of Synthesis, different abstraction models are therefore enablers for hw/sw co-design.

Co-Simulation and Emulation

Emulation System

1. An Hardware Emulator would allow the full design to be imported onto the Emulator (through a said flow) onto a board which has real components.

2. FPGAs, QuickTurn (Palladium) are different emulation system (system components) which are used for prototyping.

3. Advantage of emulation are ::

    - Higher speed, in cases it is just 100x slower than real time components.
    - Allows for Higher Level of Hardware Verification or Application Level Verification.
    - Allows for Development of Firmware.
    - Allows for testing the design and the software before silicon. (With a speed of 500,000 cycles per second, it was possible to test the boot sequence of Solaris on a UltraSparc Architecture)
    - Simulation aims at the general reduction of the number of functional and timing errors, whereas emulation in general serves realistic loads in demonstrating functional correctness.

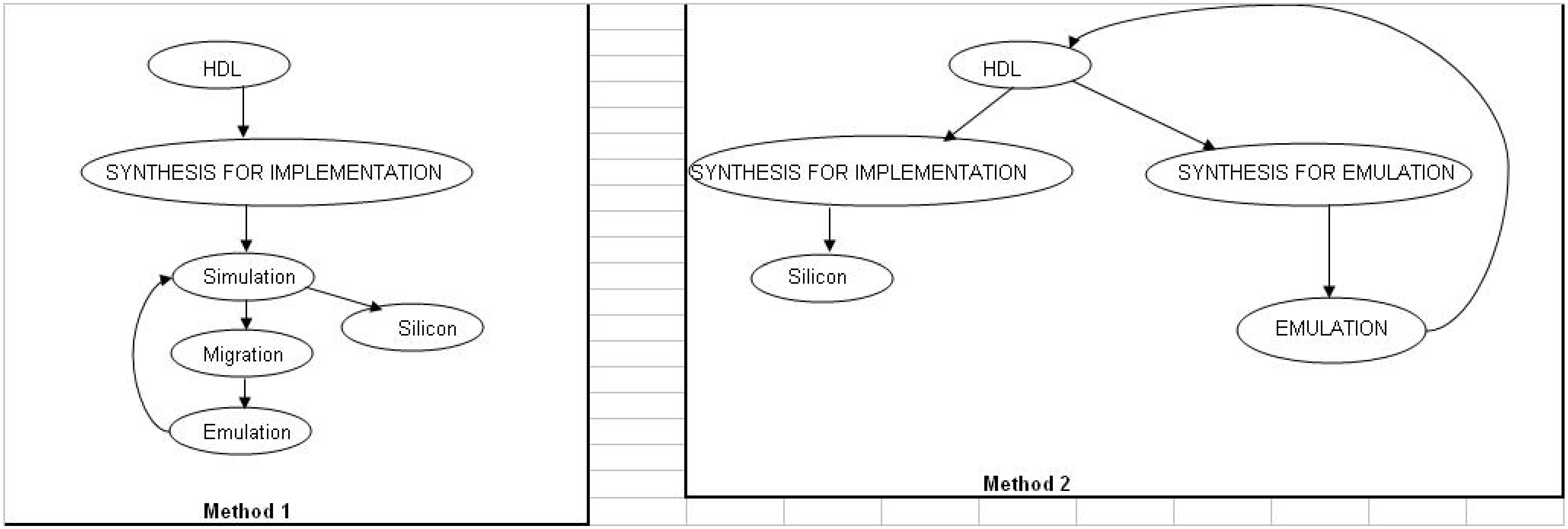
Co-Simulation and Emulation

Emulation System

1. Dis-Advantage of emulation are ::
   - Timing errors in the actual device are hard to detect on emulation platforms (this is possible in simulation system using sdf timing annotations).
   - In case of FPGAs, a separate effort has to be done to meet timing requirements for the FPGA itself.
   - Cost is higher for emulation.
   - Emulation is generally only used after thorough simulations so that there is low expectation of design errors.

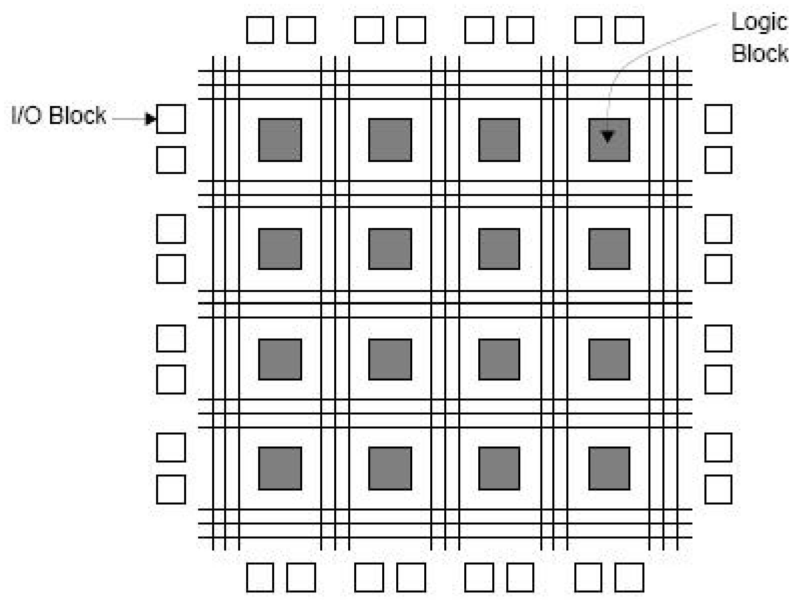
Co-Simulation and Emulation

Methods in Deploying Emulation System



Co-Simulation and Emulation

FPGA Based System



1. The Static RAMs based implementation provides unlimited programmability to FPGAs.

2. FPGAs use building blocks called CLB (configurable logic block),

3. Add Dedicated Memory Blocks,

4. Clocking Blocks,

5. With Interconnects to connect one CLB, Memories, Clocks, IO to each another, and

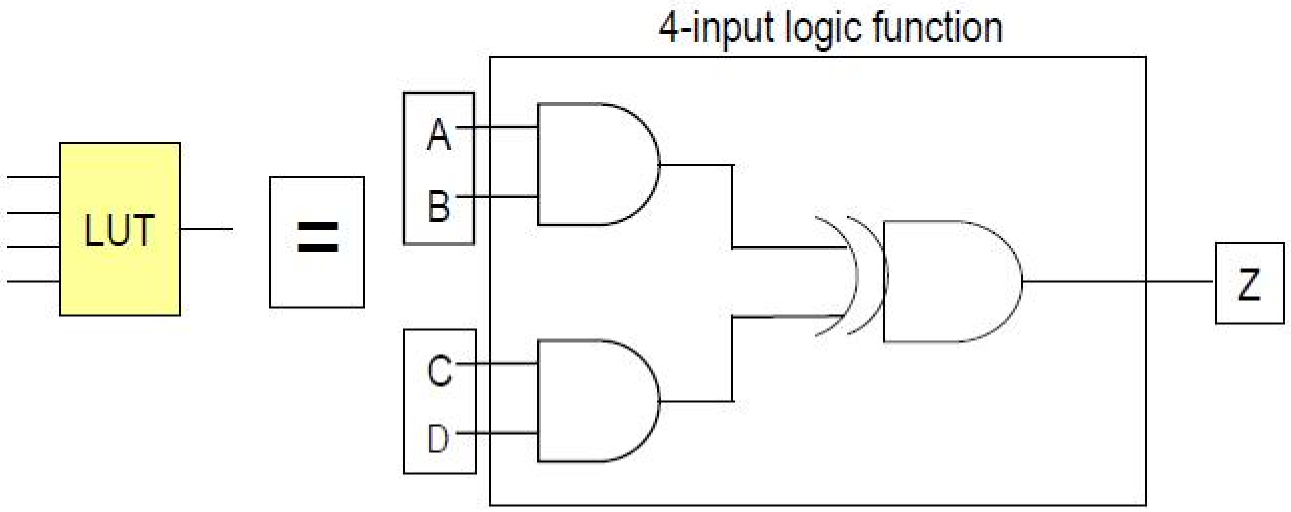
6. An IO interface to talk to external (real) components.

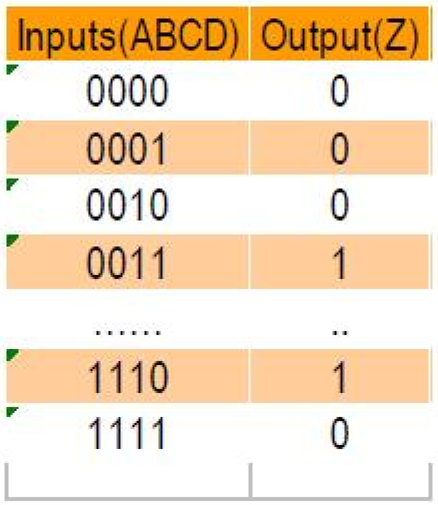
7. GROUP DISCUSSION :: Discuss that a 4 input LUT can serve as a universal gate.

8. There can be architectures where the CLB itself can serve as a Memory Block.

Co-Simulation and Emulation

FPGA Based System



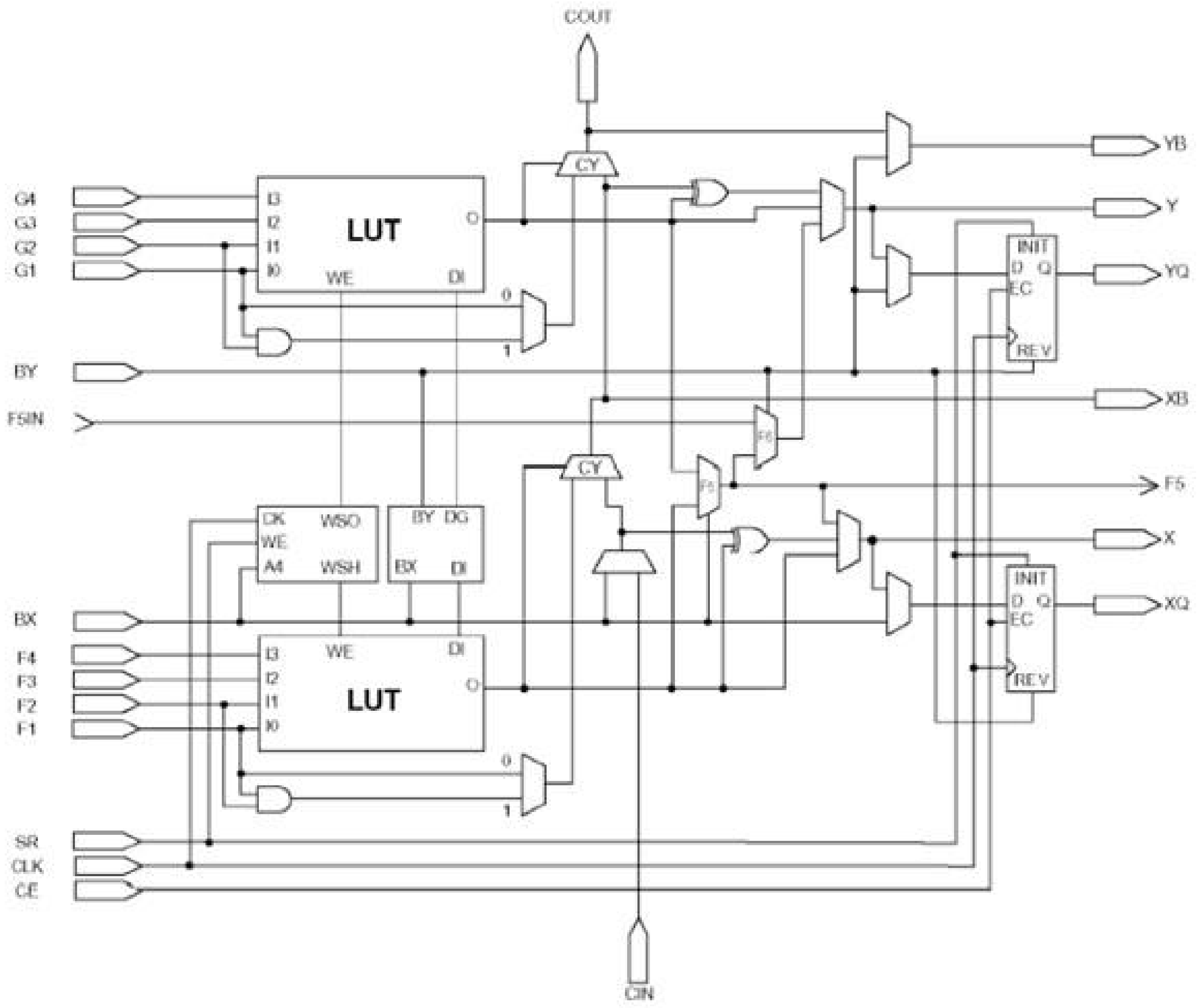


Inputs(ABCD)	Output(Z)
0000	0
0001	0
0010	0
0011	1
......	..
1110	1
1111	0

1. GROUP DISCUSSION :: Discuss that a 4 input LUT can serve as a universal combo gate.

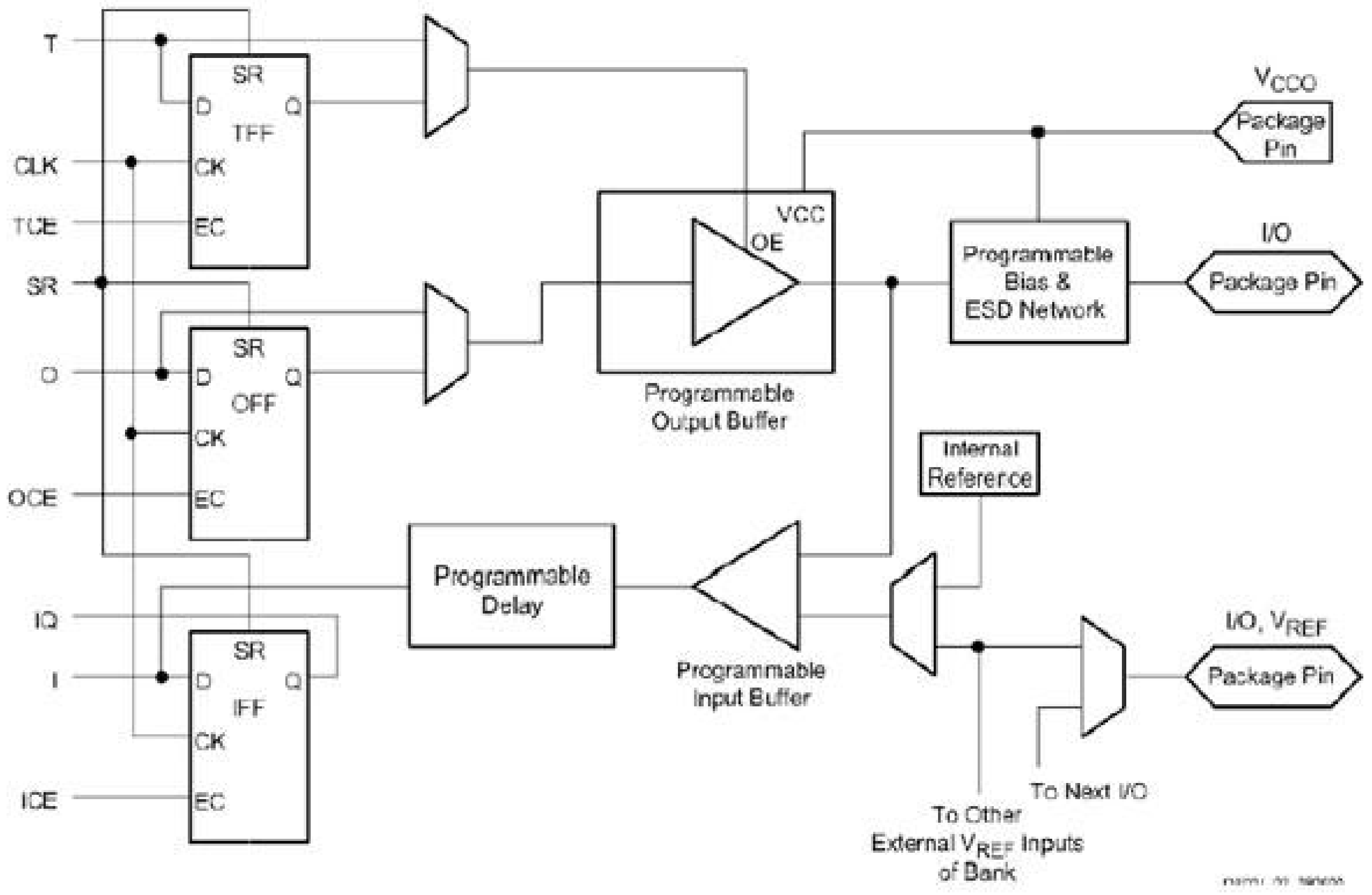
Co-Simulation and Emulation

FPGA Based System – CLB of a Xilinx FPGA



Co-Simulation and Emulation

FPGA Based System – IO of a Xilinx FPGA



Co-Simulation and Emulation

FPGA Based System

1. FPGAs also come with various capacity tags, which are suitable for different ranges of devices being built till date.

2. There exists systems and software which use ”arrays” of FPGAs for larger SoC’s or ”chip-sets”

3. As FPGAs evolve through the technology nodes, capacity naturally increases and increasingly FPGA vendors are able to integrate common CPUs (like ARM), common blocks like DSP MAC units, Communication physical layer (Phy) like Serdes, Ethernet Phy, USB Phy, differential IO’s.

4. Unlike Emulation systems, FPGAs are also being used for doing field trials i.e, in an environment where the actual ASIC would be. Thus portability is also become a requirement for prototyping.

Co-Simulation and Emulation

Emulation Based System – Example of Palladium XP – From Cadence Website

Co-Simulation and Emulation

Emulation Based System – Example of Palladium XP – From Cadence Website

1. Unparalleled operational efficiency and user flexibility with highly scalable systems
2. Delivers high performance of up to 4MHz
3. Allows flexible configurations from 4 million gates to 2 billion gates
4. Supports up to 512 users simultaneously
5. Advanced compiler and runtime capabilities
6. Integrates seamlessly with Incisive products
7. Offers a unified, advanced debug environment for HW/SW co-verification
8. Supports SCE-MI and SystemVerilog DPI for third-party models/tools integration
9. Unique platform extensions
10. Supports metric-driven verification acceleration
11. Supports industry-standard hardware design and verification languages and the Open Verification Methodology

12. Enables Dynamic Power Analysis and verification by integrating with the Encounter RTL Compiler power estimation engine

13. SystemC-to-emulation flow allows users to integrate high-level abstraction models into the system verification environment

14. Integrates with the comprehensive SpeedBridge family of rate adapters and the Cadence Verification IP portfolio

Co-Simulation and Emulation

Emulation Based System – Example of Palladium XP – From Cadence Website

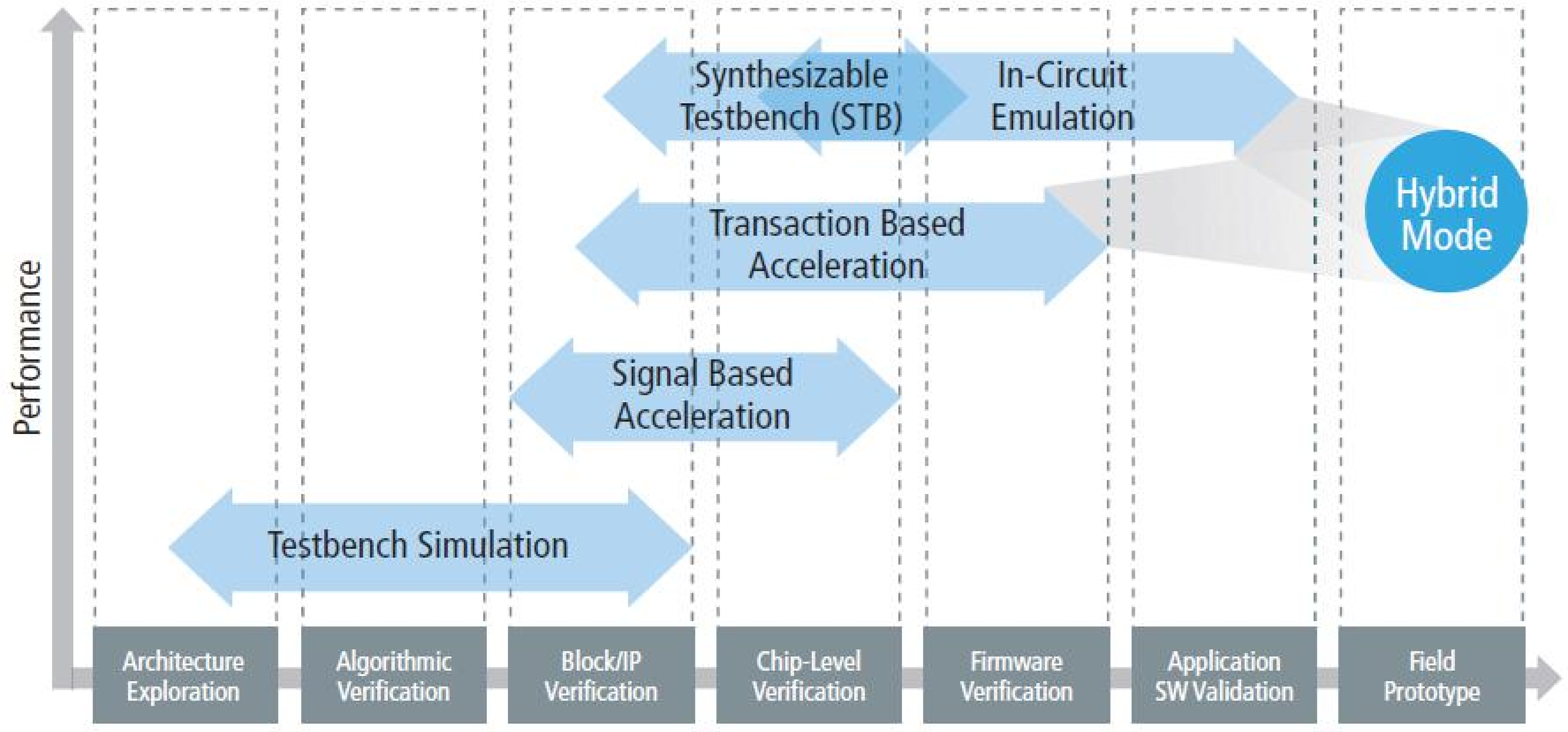
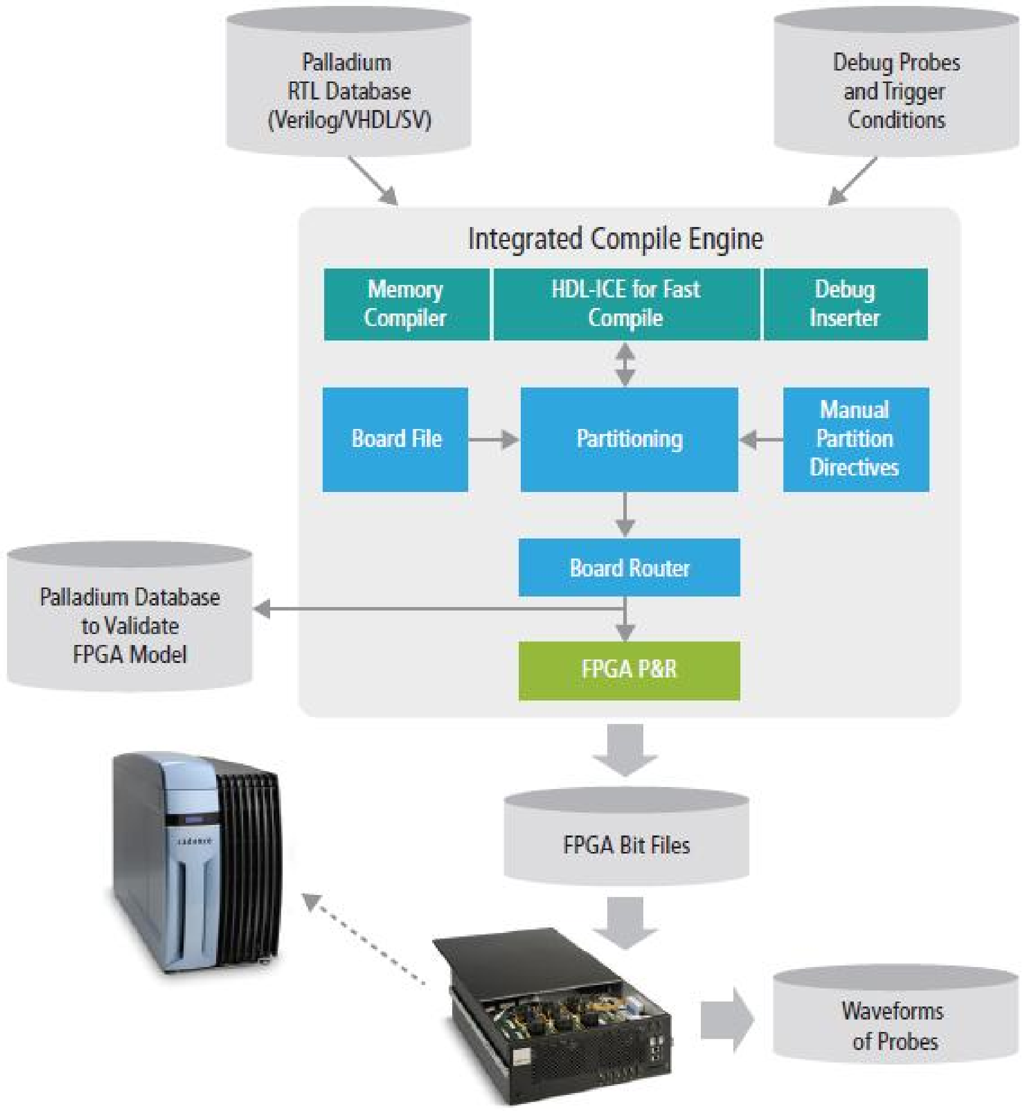


Figure 7: The Palladium XP Verification Computing Platform stretches through much of the hardware/software development cycle

1. **GROUP DISCUSSION** :: Compare an Emulator Vs FPGA

Co-Simulation and Emulation

Emulation Based System – From Palladium XP – Rapid FPGA Based System – From Cadence Website


Palladium RTL Database (Verilog/VHDL/SV)
Debug Probes and Trigger Conditions
Integrated Compile Engine
Memory Compiler
HDL-ICE for Fast Compile
Debug Inserter
Board File
Partitioning
Manual Partition Directives
Board Router
Palladium Database to Validate FPGA Model
FPGA P&R
FPGA Bit Files
Waveforms of Probes

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 → November 13 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

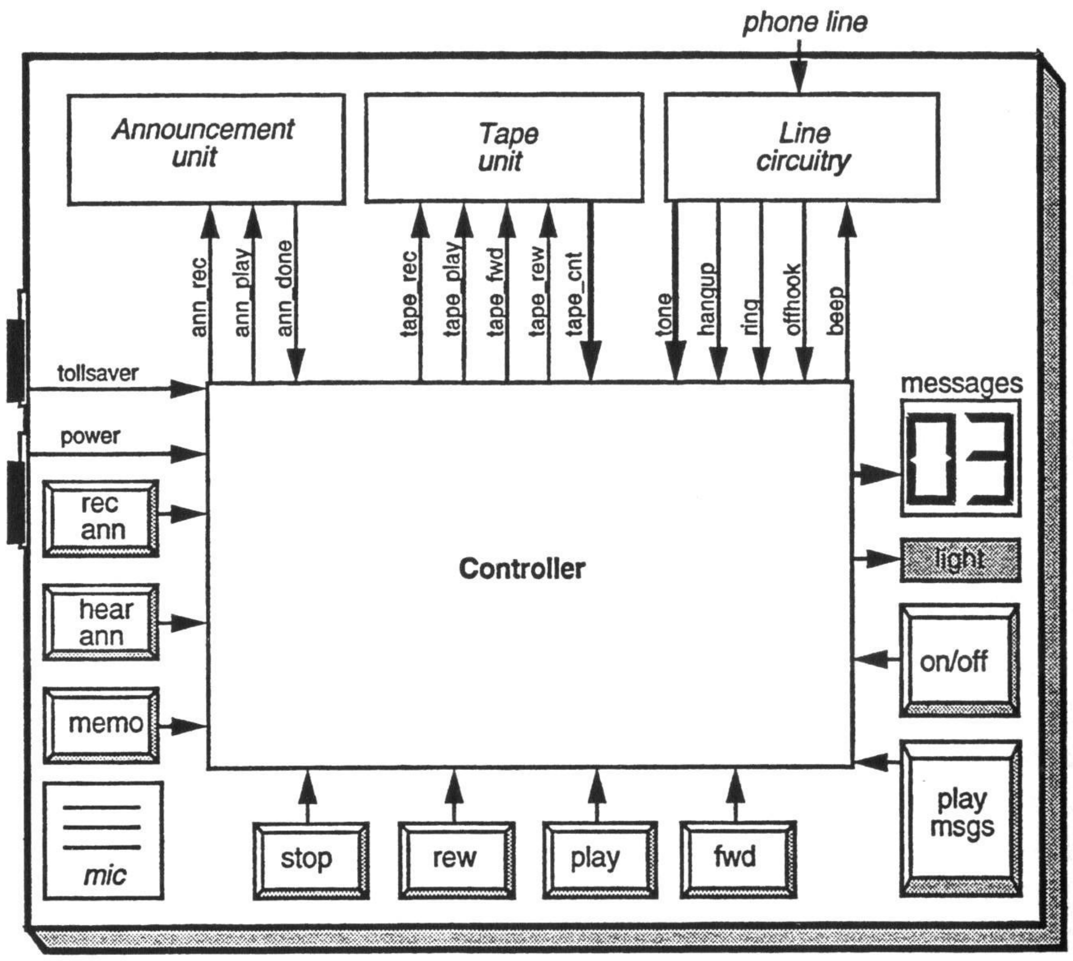
1. System C model of the system.
2. Use models for analog components.
3. Should have System C testbench as a part of SoC verification.
4. ReUse preferred to creation of already existing components.

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

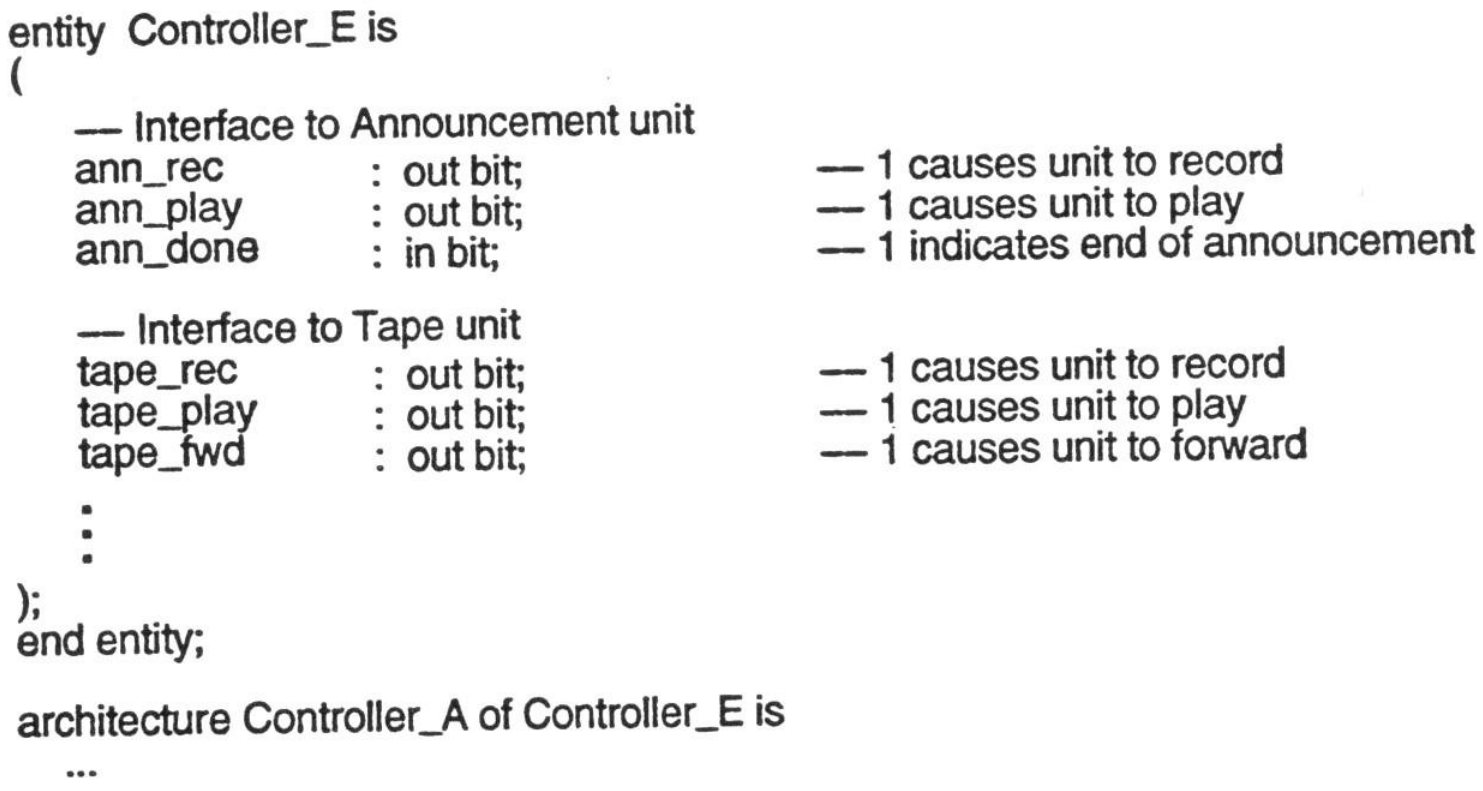
2. Processor / Dedicated Hardware decision

3. System State Machine or StateChart diagram (gross functionality)

4. Individual functionality State Macine or StateChart diagram

"Answering Machine" Assignment

Next Steps for Implementation document – Entity Definition

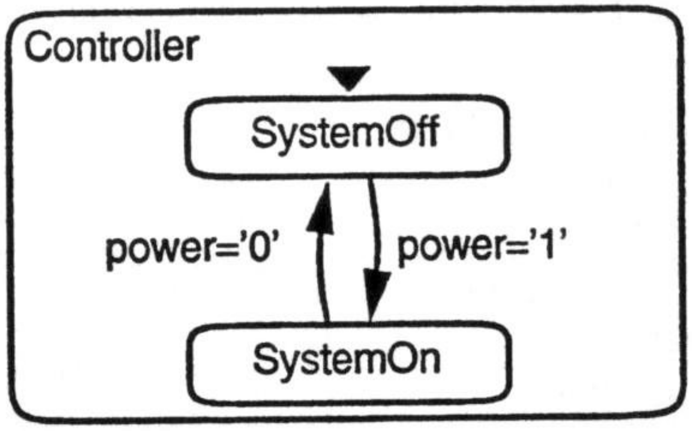


```
entity  Controller_E is
(
    -- Interface to Announcement unit
    ann_rec         : out bit;              -- 1 causes unit to record
    ann_play        : out bit;              -- 1 causes unit to play
    ann_done        : in bit;               -- 1 indicates end of announcement

    -- Interface to Tape unit
    tape_rec        : out bit;              -- 1 causes unit to record
    tape_play       : out bit;              -- 1 causes unit to play
    tape_fwd        : out bit;              -- 1 causes unit to forward
    :
    :
);
end entity;

architecture Controller_A of Controller_E is
   ...
```

The answering machine controller's interface.

"Answering Machine" Assignment

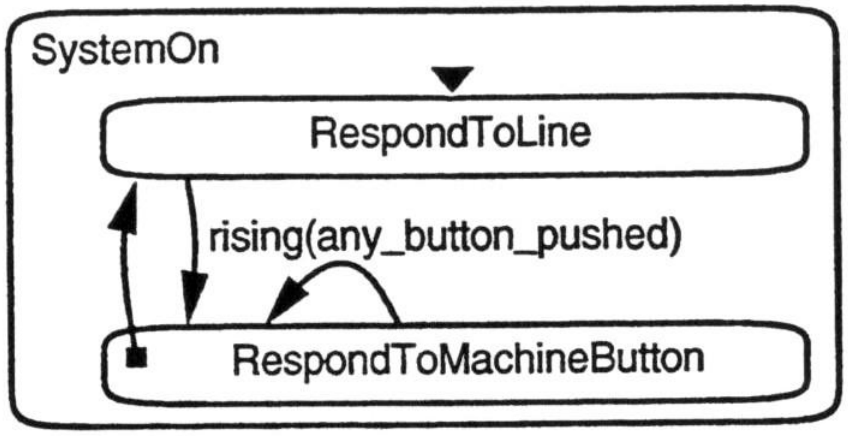
Next Steps for Implementation document – Function Level State Charts – ON/OFF



Highest-level view of the controller

"Answering Machine" Assignment

Next Steps for Implementation document – Function Level State Charts



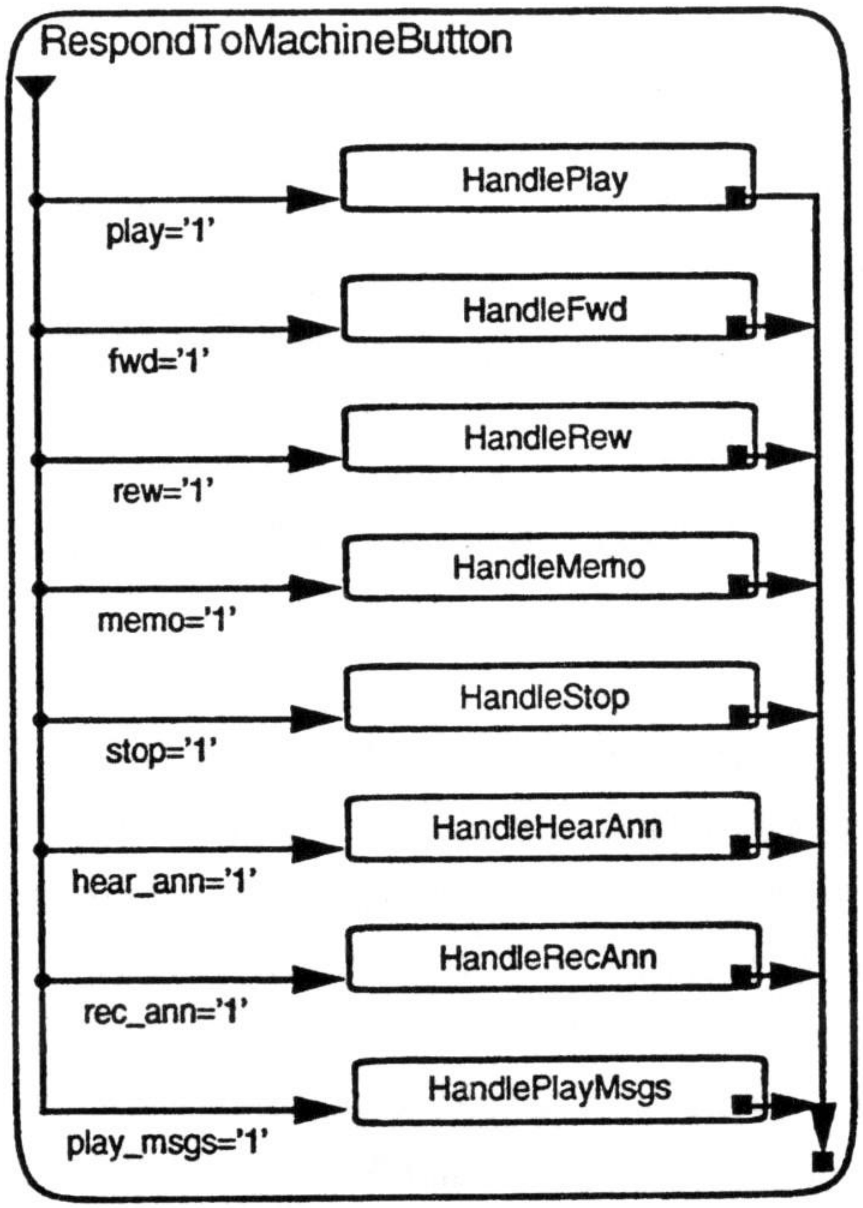
The SystemOn behavior

”Answering Machine” Assignment

Next Steps for Implementation document – Function Level State Charts

```
behavior RespondToMachineButton
  type code is
begin
   if (play='1') then
      HandlePlay;
   elsif (fwd='1') then
      HandleFwd;
   elsif (rew='1') then
      HandleRew;
   elsif (memo='1') then
      HandleMemo;
   elsif (stop='1') then
      HandleStop;
   elsif (hear_ann='1')  then
      HandleHearAnn;
   elsif (rec_ann='1') then
      HandleRecAnn;
   elsif (play_msgs='1') then
      HandlePlayMsgs;
   end if;
end;
```

(a)



(b)

The RespondToMachineButton behavior: (a) sequential statements, (b) equivalent subbehavior decomposition.

Taking Stock

1. Historical perspective – from microprocessor based designs to current SoCs.
2. A Multiplier example to illustrate the tradeoffs.
3. Typical HW/SW co-design flow.
4. 5-Basic Themes for hw/sw co-design.
5. Example of a TouchScreen System. Defining different parts of the system.
6. Modeling – Basic Modeling Requirements for co-design.
7. Modeling – Modeling using SystemC.
8. Analysis and Estimation – Path based enumeration technique.
9. Analysis and Estimation – Rate Monotonic Analysis for Single CPU, Deadline Driven analysis, Mixed Scheduling Algorithm.
10. Partitioning, Synthesis and Interfacing – GCLP, MIBS algorithms (Kalavade & Lee)
11. Example of converting a C based algorithm to HW (GCD, Matrix Multiplication)
12. CoSimulation and Emulation – FPGA based, Emulator based

Co-Simulation and Emulation

Acknowledgements

1. Readings in Hardware / Software Co-design ::G. D. Micheli, Rolf Ernst, Wayne Wolf :: Ch 6

2. Hardware / Software Co-Design - Principles and Practice :: J. Staunstrup & W. Wolf :: Ch 3

3. Fpga Prototyping Methodology Manual :: Ch 3

4. http://www.cadence.com/products/sd/palladium_xp/Pages/default.aspx