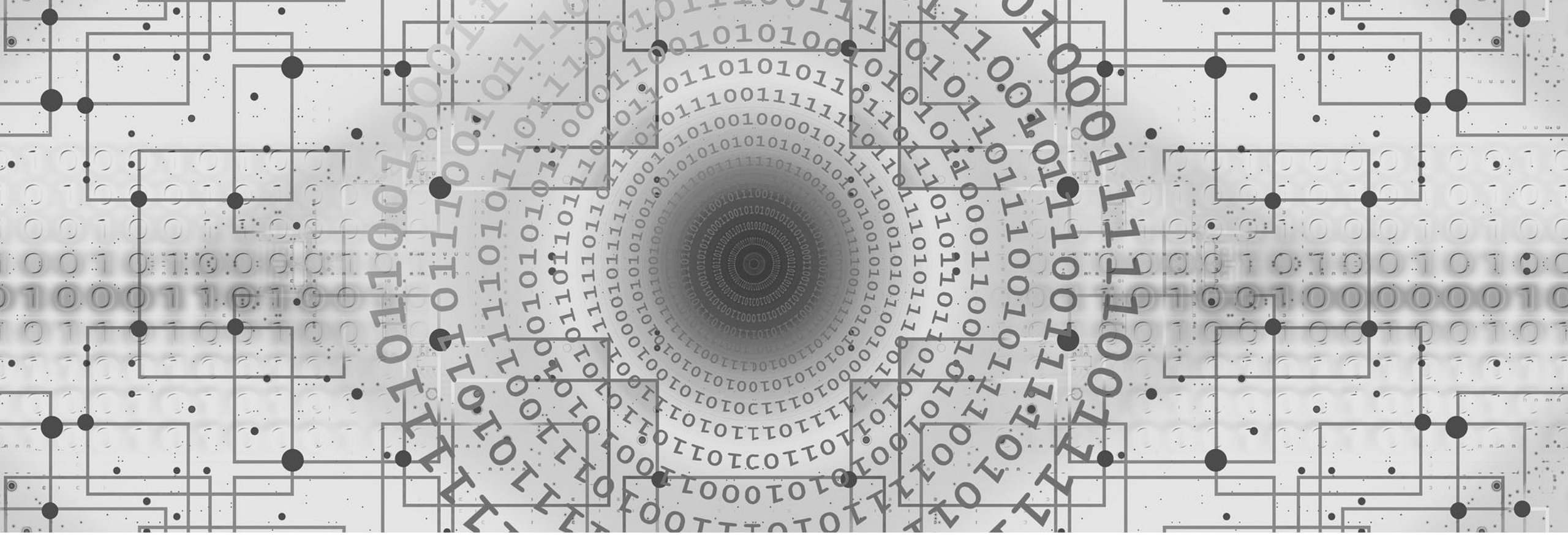


Lecture-1 : Introduction

ECE-111 Advanced Digital Design Project

Vishal Karna

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Winter 2022

Teaching Staff

- **Lecturer :** Vishal Karna
  - Online office hours: Wednesday's 6:20-7:00pm (after lecture hours)
  - Students can request for additional 1-1 or group meeting
  - For project discussion, additional office hours will be published

- **Lecture Schedule :**
  - Each week Monday and Wednesday 5:00pm to 6:20pm

- **Weekly Discussion Session :**
  - Each week Monday from 4:00pm to 4:50pm, TA and/or Instructor to conduct the discussion sessions
  - **Note :** Instructor will be conducting some of the discussion session on final project which will be announced on piazza

- **Teaching and Support Staff :**
  - Zixiang Zhou, Teaching Assistant, email : ziz358@ucsd.edu
  - Naman Sehgal, Teaching Assistant, email : nsehgal@ucsd.edu
  - Brandon Saldanha, Teaching Assistant, email : bsaldanha@ucsd.edu
  - Vijayalakshmi Swaminathan, Teaching Assistant, email : vswaminathan@ucsd.edu
  - Shengfan Hu, Tutor, email : shh042@ucsd.edu
  - TA online support and office hours and zoom links are posted on piazza :
    - https://piazza.com/ucsd/winter2022/ece111/staff
    - Students can request to TA for additional sessions to get support on homework and projects

Teaching Platforms And Resources

- **Live streaming of lectures using Zoom Platform :**
  - Lectures will also be recorded, and links will be published to students on canvas
  - Each week lecture online meetings are scheduled and published on class canvas :
    - https://canvas.ucsd.edu/courses/33921/external_tools/628 (**meeting password** : galileo111)

- **Canvas will be used to publish course material and resources :**
  - Lecture slides, Zoom meetings, homework, project, quiz, tools instructions, learning resources
  - Canvas course webpage : https://canvas.ucsd.edu/courses/33921

- **Piazza for Q&A and Announcements :**
  - All announcements on piazza such as quiz date, polls, project discussion sessions and more
  - Using piazza students can ask any questions on lectures, homework, projects, quiz and more
  - Piazza course webpage : https://piazza.com/ucsd/winter2022/ece111/info
  - Piazza Q&A : https://piazza.com/class/kxt74scerj075n
  - Piazza access code : **ECE111WI22**

- **Gradescope to publish and upload homework and final project assignments :**
  - Homework's and final project will be published on gradescope which is linked inside canvas
  - Students to upload their homework and final project report on gradescope through canvas

Course Objectives

- **Design synthesizable hardware models using SystemVerilog Language**
  - Design combinational and sequential logic circuits using SystemVeriog
  - Hardware modeling styles such as gate level, behavioral, RTL, switch level
  - Develop synchronous finite state machines, state diagrams and state tables
  - Synthesis coding guidelines through examples
  - Differences between Verilog and SystemVerilog language and its advantages
  - Develop synthesizable SHA256 cryptographic and bitcoin hashing model as part of final project
- **Learn how to use CAD Tools for hardware design and verification :**
  - Modelsim simulator to perform SystemVerilog design code simulation and debug waveform
  - Altera Quartus Prime Lite to synthesis SystemVerilog design code and convert to gate level circuit
  - Understand FPGA resource allocation, RTL and post mapping netlist and review timing reports
- **FPGA and ASIC Concepts**
  - Understand FPGA architecture, application and how digital logic function is implemented inside FPGA
  - ASIC and FPGA frontend and backend design flow
  - Verification concepts including testbench and event driven and cycle accurate simulation
- **Digital Design Timing Concepts**
  - Understand the fundamentals of digital design timing and synchronization techniques

Course Grading Policy (Tentative)

- **Final Quiz : 20% of final grade (Note : There will be no other mid-term or final exam apart from 1 final Quiz)**
  - Final quiz will be conducted using canvas and zoom platform during one of the lecture hours
    - Students will take the quiz online and it will be proctored by Instructor and TA's.
    - Students will be allowed to refer to lecture slides and it will be a 25 to 30 multiple choice and textbox-based questions
  - Quiz date will be **reconfirmed,** and Quiz outline will be published to all students on a later date
    - **Tentative date** of Final quiz for now is **February 28th**, 2022. Around **2.30 hours** will be provided to students to finish the quiz. Students will be allowed to refer lecture slides during the quiz.

- **Weekly Assignments : 45% of final grade**
  - Design hardware circuits using synthesizable RTL code using System-Verilog
  - Synthesize design using Altera Quartus prime software and view circuit generated
  - Simulate design using tesbench code and debug waveforms to ensure design code behavior
  - Testbench will be provided to students for most assignments. Students can modify.
  - For details on each assignment and deadline will be published canvas/gradescope each week

- **Final project : 35% of final grade (It is a group project. Per group 1 or 2 or at max 3 students allowed**)
  - SHA-256 cryptographic and Bitcoin hashing RTL model development and optimization
  - More information on the final project and separate project discussion sessions will be announced
  - You can register yourself to a group on Canvas : https://canvas.ucsd.edu/courses/33921/groups#tab-10650

ECE-111 Winter'22 Course Format and Schedule

Note : ECE-111 Course is Designed with Top Down Approach

Date	Time	Lecture and Discussion Agenda		Assignment Description	Assignment Due Date
January 3 (Mon)	5 PM to 6.20 PM	Lecture-1	Introduction to ECE-111		
January 5 (Wed)	5 PM to 6.20 PM	Lecture-2	ASIC, FPGA, Logic Synthesis Fundamentals		
January 5 (Wed)**	4 PM to 5 PM	Weekly Discussion	Software Installation, Usage and Project Creation Overview		
January 10 (Mon)	4 PM to 5 PM	Weekly Discussion	Applications of FPGA		
January 10 (Mon)	5 PM to 6.20 PM	Lecture-3	SystemVerilog Modeling Abstraction	Homework-1 : Synthesize MUX, FPGA Resource Usage Analysis, 4-bit ALU	1/17/2022, 11.59 PM
January 12 (Wed)	5 PM to 6.20 PM	Lecture-4	Anatomy of SystemVerilog		
January 17 (Mon)	Martin Luther King, Jr. Holiday (No Lecture)				
January 19 (Wed)	5 PM to 6.20 PM	Lecture-5	SystemVerilog Data Types, Continuous Assignment Statement and Conditional Operator, Case Statements	Homework-2 : Behavioral, DataFlow and Gatelevel Model of a decoder	1/24/2022, 11.59 PM
January 24 (Mon)	4 PM to 5 PM	Weekly Discussion	Overview of SystemVerilog Operators		
January 24 and 26 (Mon,Wed)	5 PM to 6.20 PM	Lecture-6, 7	Blocking and Non-Blocking Assignment statements	Homework-3 : Johnson Counter, Universal Shift Register, Barrel Shifter	1/31/2022, 11.59 PM
January 31 and February 2 (Mon,Wed)	5 PM to 6.20 PM	Lecture-8, 9	Procedural Blocks	Homework-4 : LFSR, SECDED Error Correction and Detection	2/7/2022, 11.59 PM
January 31 (Mon)	4 PM to 5 PM	Weekly Discussion	Testbench Fundamentals		

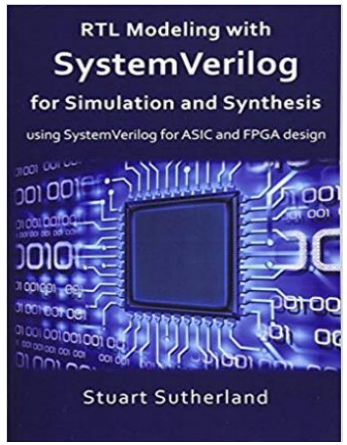
** January 5 and March 2 are the two exceptions for weekly discussions which will falls on Wednesday instead of official weekly discussion day for ECE-111 which is Monday

ECE-111 Winter'22 Course Format and Schedule

Date	Time	Lecture and Discussion Agenda		Assignment Description	Assignment Due Date
February 7 and 9 (Mon,Wed)	5 PM to 6.20 PM	Lecture-9, 10	RTL Programming Statements	Homework-5 : Carry Look Ahead Adder, Clock Divider, gray to binary converter	2/14/2022, 11.59 PM
February 7 (Mon)	4 PM to 5 PM	Final Project Discussion	SHA256 Concepts		
February 14 and 16 (Mon,Wed)	5 PM to 6.20 PM	Lecture-11, 12	Finite State Machines	Homework-6 : Vending State Machine, Integer Multiplier (or Booth Multiplier)	2/22/2022, 11.59 PM
				Homework-7 : UART Transmitter-Receiver System with Parity Checker	3/1/2022, 11.50 PM
				Homework-8 : Handshake Synchronizer with Memory Model	3/6/2022, 11.59 PM
February 14 (Mon)	5 PM to 6.20 PM	Final Project Discussion	Part-1 SHA256 Algorithm	Final Project Part-1 : SHA256	3/18/2022, 11.59 PM
February 21 (Mon)	President's Day, Holiday (No Lecture)				
February 23 (Wed)	5 PM to 6.20 PM	Final Project Discussion	Part-2 Bitcoin Model	Final Project Part-2 : Bitcoin	3/18/2022, 11.59 PM
February 28 (Mon)	5 PM to 7.30 PM	Final Quiz	Final Quiz		Quiz Results will be published by 3/7/2022
March 2 (Wed)**	4 PM to 5 PM	Final Project Discussion	SHA and Bitcoin Optimization		
March 2 (Wed)	5 PM to 6.20 PM	Lecture-13	Memory Modeling, Unpacked and Packed Arrays		
March 7 and 9 (Mon, Wed)	5 PM to 6.20 PM	Lecture-14, 15	Timing and Synchronization	Homework-9 : Asynchronous FIFO (optional)	Not Applicable since it is an optional assignment for learning purpose

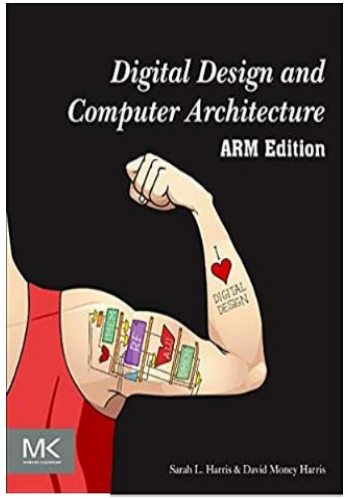
** January 5 and March 2 are the two exceptions for weekly discussions which will falls on Wednesday instead of official weekly discussion day for ECE-111 which is Monday

References



RTL Modeling with SystemVerilog for Simulation and Synthesis: Using SystemVerilog for ASIC and FPGA Design, 1st Edition
Author : Stuart Sutherland
ISBN-13: 978-1546776345
ISBN-10: 1546776346
Note : This book is not available in UCSD Library.
https://www.amazon.com/RTL-Modeling-SystemVerilog-Simulation-Synthesis-ebook/dp/B071GY6MND

Primary Textbook



Digital Design and Computer Architecture: ARM Edition 1st Edition
Author : Sarah Harris and David Money Harris
ISBN-13: 978-0128000564
ISBN-10: 9780128000564

Note : Both these books are great learning resources however it is not mandatory to purchase the book for passing ECE-111 class. Lecture slides will have enough material which is required to complete homework assignments, final project and prepare for the final quiz !

IEEE SystemVerilog, LRM https://standards.ieee.org/standard/1800-2012.html

Accellera https://www.accellera.org/downloads/standards

Software

- **Intel Altera Quartus Prime Lite 18.1 Edition (19.1, 20.1 Editions can also be used)**
  - Includes both RTL code synthesis, implementation and simulator tools
  - ModelSim-Altera FPGA edition simulator is part of the installation
  - FPGA Target Device for Synthesis : Arria-II

- **Quartus Prime Lite download and installation instructions published on Canvas in files→tool_docs**: https://canvas.ucsd.edu/courses/33921/files/folder/tools_docs
  - For Windows OS machines refer to document name mentioned below :
    - quartus_modelsim_instructions_windows.pdf
  - For Mac OS machines refer to document mentioned below :
    - guide_for_accessing_quartus_on_mac.pdf
  - Alternatively, refer to Amazon Web Services (AWS) usage for Mac OS and Windows machines refer to document mentioned below :
    - access_quartus_using_AWS.pdf

- **Refer to these videos listed on Canvas under Media Library :** https://canvas.ucsd.edu/courses/33921/external_tools/82
  - How to create project and synthesize SystemVerilog code using use Quartus Prime Lite
    - https://youtu.be/iLbmSTG7bpA
  - How to invoke Modelsim-Altera from Quartus Primte-Lite and simulate SystemVerilog code
    - https://youtu.be/BcvclrqZ2fc

Important Note : Only for MAC OS based Machine Users

- **Intel Altera Quartus Prime Lite only supports Windows and Linux OS**
  - **MAC OS is not supported by Quartus Prime Lite!**
  - EBU1-4309 Lab has windows machine with Quartus prime-lite installed
  - Due to COVID-19 situation and advisory all lab rooms will be closed
  - UCSD IT, is working on different solutions to make this software available remotely for MAC machine users.

- **There are multiple options for MAC OS users to access Quartus Prime-Lite**
  - Option 1 : Using remote desktop login to UCSD server and access linux version of Quartus
  - Option 2 : Windows10 dual boot setup on MAC machine using dual boot camp free software and install Quartus windows version. **Windows10 student edition is free for UCSD students !**
  - Option 3 : Using VirtualBox free software on MAC machine and iWindow10
  - Option 4 : Using Amazon AWS and run Quartus Prime-Lite directly on cloud

- **Recommendations :**
  - **Option 1** : Second best in terms of performance, however linux version currently is available for 17.1 Quartus lite version. Not widely used and tested.
  - **Option 2** : Preferred and suggested for best software performance. Requires at least 60GB of free space on machine and backuping of current data onto an external drive.
  - **Option 3** : Slowest in terms of performance and this should be the very last resort.
  - **Option 4** : Login to Amazon AWS webservice and run Quartus prime on Cloud. UCSD IT is working on this and steps are not available yet. And software performance not known yet.

Software Usage And Support

- **To demonstrate Quartus Altera software usage (both for Windows & MAC Users)**
  - Teaching staff will conduct software setup and usage session on January 5th from 4 PM to 5 PM PST.
  - Zoom links for these software setup have been announced on Piazza and scheduled in Canvas
  - Both these discussion sessions will be recorded and uploaded to Canvas

- **MAC machine users should contact Teaching Assistant Zixiang Zhou and Brandon Saldhana for questions and any support required during installation**

Acronyms

- **HDL** Hardware Description Language
- **HVL** Hardware Verification Language
- **ASIC** Application Specific Integration Circuit
- **FPGA** Field Programmable Gate Array
- **SoC** System-On-Chip
- **IP** Intellectual Property
- **AMS** Analog Mixed Signal
- **PRD** Product Requirement Document
- **DUT** Design Under Test
- **DUV** Design Under Verification
- **FSM** Finite State Machine
- **LRM** Language Reference Manual
- **EDA** Electronic Design Automation
- **CAD** Compute Aided Design
- **VHSIC** Very High Speed Integrated Circuit
- **PCB** Printed Circuit Board
- **DFT** Design For Testability
- **OOP** Object Oriented Programming
- **HW** Hardware
- **SW** Software
- **NBA** Non Blocking Assignment
- **LHS** Left Hand Side
- **RHS** Right Hand Side
- **GPU** Graphics Processing Unit
- **DPU** Data Processing Unit
- **CPU** Central Processing Unit

Lecture-1 Outline

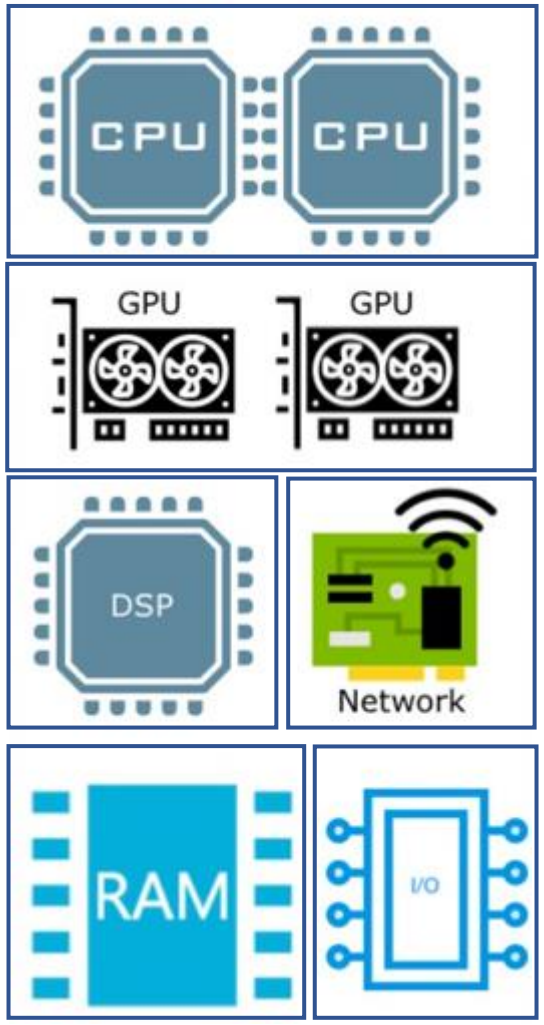
- **Introduction to below mentioned terminology :**
  - SoC, FPGA, ASIC, HDL, HVL
- **Full custom chip designing process and challenges, Problem Statement**
- **Role of hardware description language and why it is required ?**
- **Why learn SystemVerilog over other hardware description languages,**
- **Evolution of SystemVerilog and differences compared to VHDL**
- **SystemVerilog Language Capabilities and Syntax Summary**

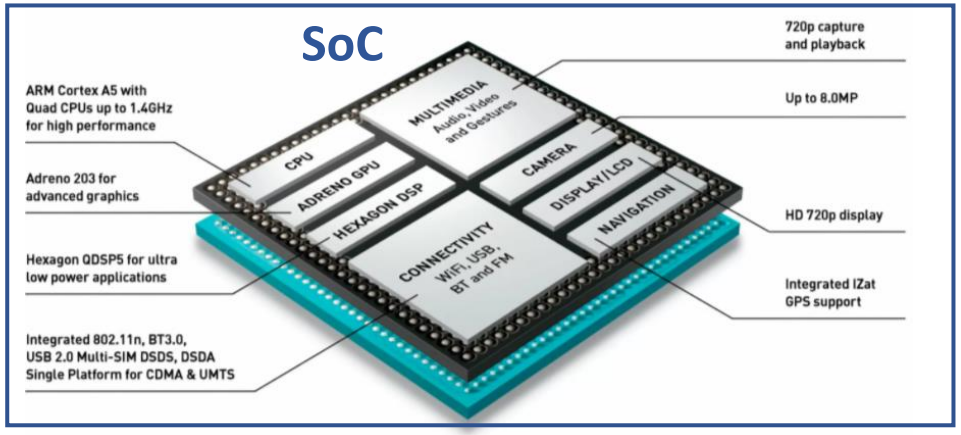
Note : Applications of FPGA will be covered during Weekly Discussion Session On :

- **January 10th from 4 PM to 4.50 PM**

What is an SoC ?

- **SoC** (**S**ystem-**O**n-**C**hip) is a collection of heterogenous components connected appropriately to perform specific function for end user application
  - **Example :** SoC inside Mobile Phones, usually holds many computer components such as the CPU, GPU, DSP, WIFI, Memory, input/output (I/O) ports, SSD storage on a **single die (silicon substrate) !**





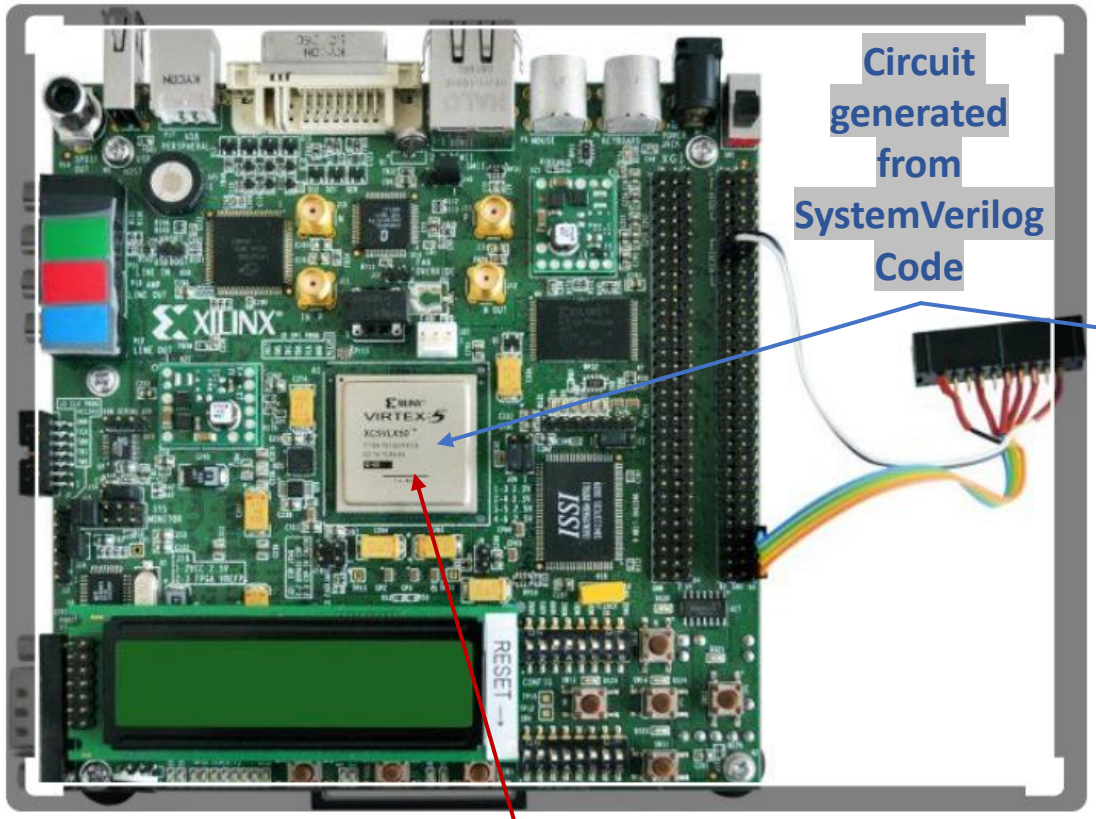
- **Benefits of SoC :**
  - Increase Performance
  - Low Power Consumption
  - Reduced Size
  - Reduced Overall System Cost

- **SoC's are Everywhere !**
  - **Inside Smartphones :**
    - Qualcomm Snapdragon 845
    - Apple A12 Bionic
  - **Inside Gaming Box**
    - Nvidia Tegra X1
  - **Inside Tablets**
    - Samsung Exynos 9611
  - **Inside Mesh WIFI Routers**
    - Qualcomm IPQ4019

FPGA and ASIC

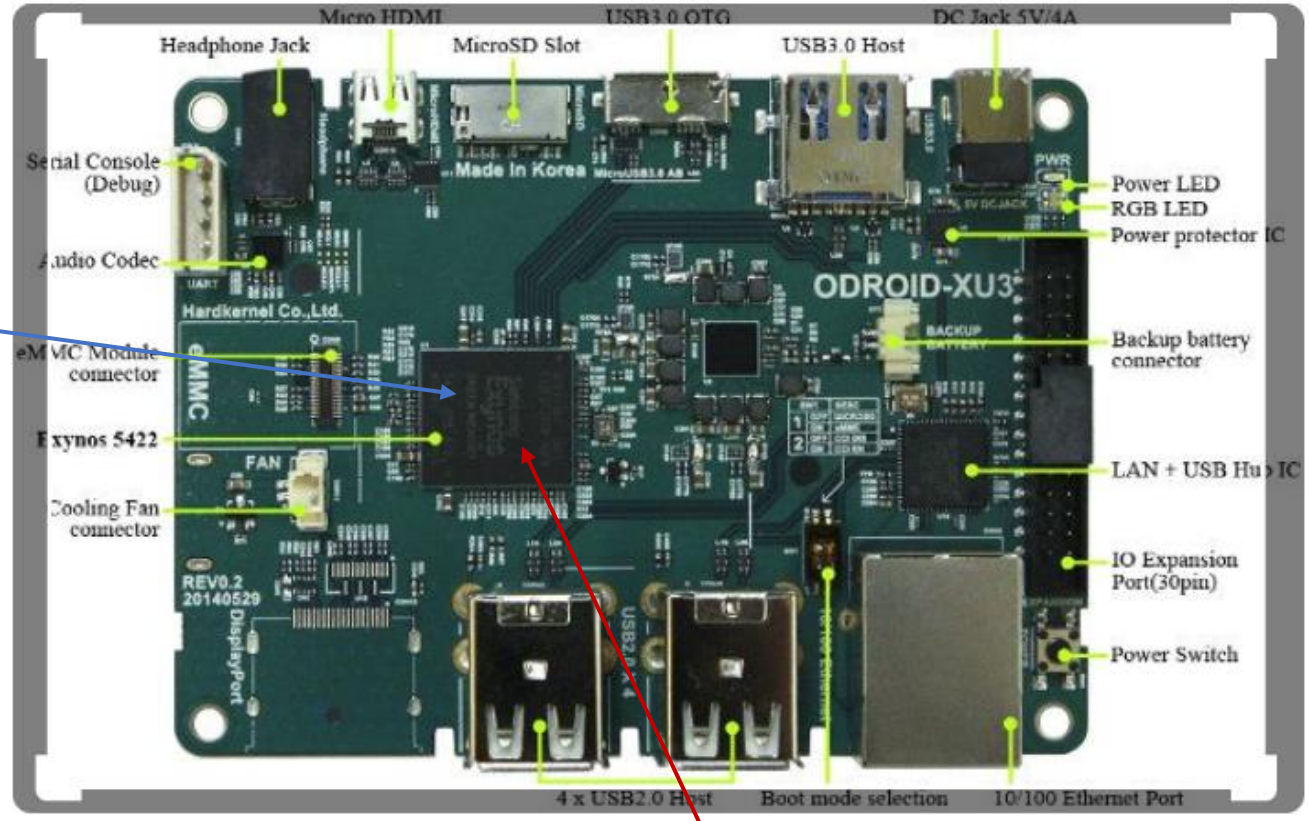
- FPGA stands for Field Programmable Gate Arrays – Re-configurable and programmable hardware
- ASIC stands for Application Specific Integrated Circuits – Fixed function hardware

FPGA based Hardware-Software Prototyping Board



FPGA can be programmed multiple times to implement different hardware designs such as GPU, CPU, Mobile Processor, etc. Runs slower than ASIC based design !

Samsung Exynos Mobile Processor (ASIC)



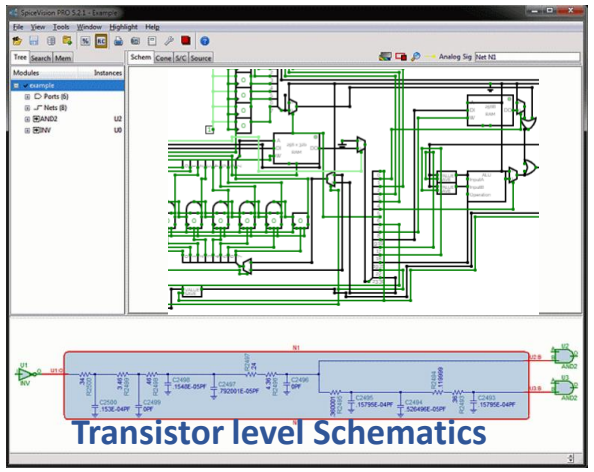
ASIC design which acts as a Mobile Processor SOC and it cannot be re-programed to change its function to something other than mobile processor. Runs faster than FPGA !

Full Custom and Manual Chip Designing Process

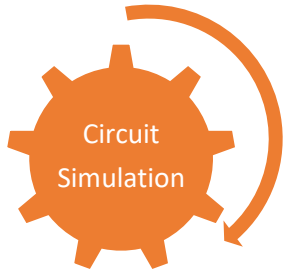
Step-1 : Draw Logic Circuit Schematics for Multiplier

Hardware Design Requirement & Specification (Design Multiplier)

Gate level Schematics (Concept Design)



Step-2 : Pre-Layout Circuit Simulation



Circuit Simulation Results (Waveform)

Update Schematics

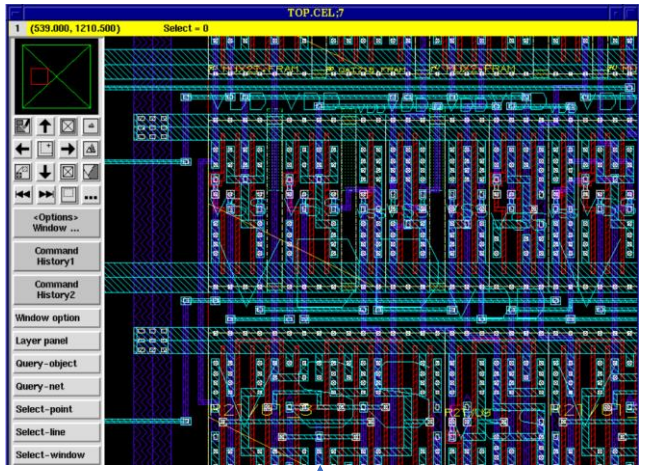
No

Does this work as a Multiplier ?

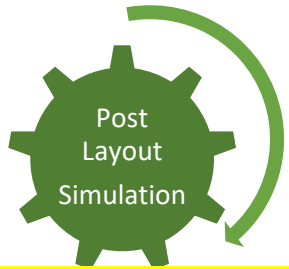
Yes

Step-3: Draw Transistor Level Layout

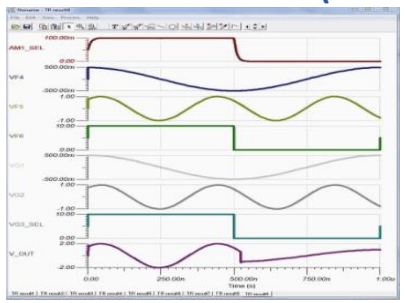
Final Transistor Level Custom Layout (Actual Design)



Step-4: Post-Layout Simulation



Characterization Results (Waveforms)



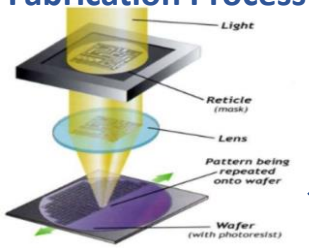
Does this work as a Multiplier and Meet requirements ?

No

Update Layout

Yes

Chip Fabrication Process



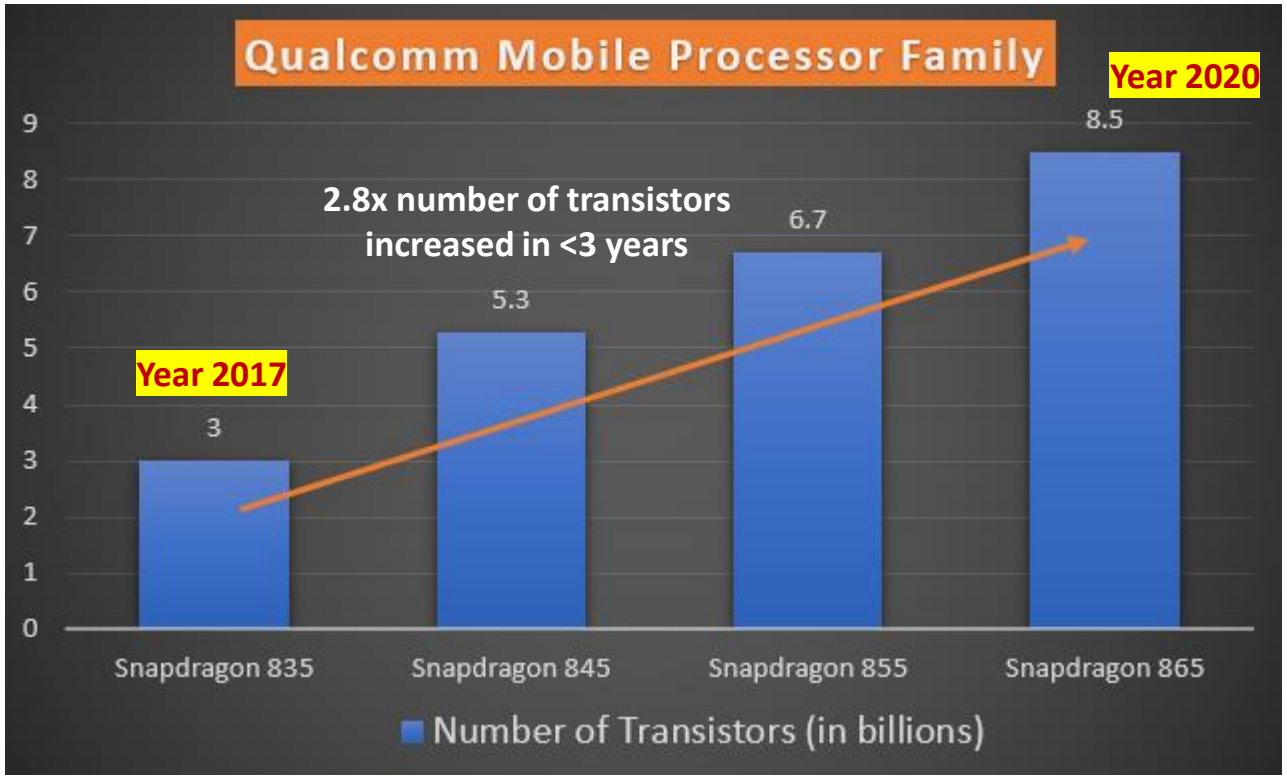
Step-5 : Send Layout to Foundry for Chip Fabrication

Multiplier Chip

Problem Statement

- **Modern digital systems and designs are highly complex and made of billions of transistors !**
  - Not feasible for human's to manually design such multi-million gate level circuits and its is transistor level layouts **– Complexity and Size**
  - There is a need for an efficient digital design methodology to ***automatically*** generate gate level circuits and transistor level layouts for developing such large systems **– Efficiency and Scalability**

For example, Qualcomm Snapdragon 845 SoC has around 5.3 billion transistors !



Full Custom and Manual Chip Designing Process Limitations

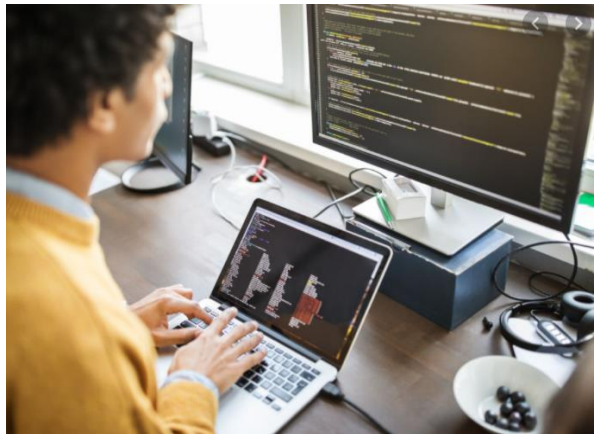
- **Can full custom and manual chip designing process address these requirements ?**
  - **Shorter time to Market (Productivity and Efficiency) :**
    - Is it feasible to manually design large size chips which have multi-billion transistors with very high complexity within reasonable amount of time ?
    - Is it feasible to verify large size designs using circuit simulation in reasonable amount of time ?
  - **Quality (Verification Completeness and Efficiency) :**
    - Can verification using circuit simulation handle extremely large set of usecases and millions of input stimulus combinations to identify bugs prior to chip fabrication ?
  - **Design Re-use (Scalability) :**
    - Can design blocks within a Chip easily be re-used with minor changes in next generation Chip ?

- **Solutions**
  - **Higher Level of Abstraction to describe complex logic circuit behavior**
    - Invent of Hardware Description Languages (Verilog, VHDL, SystemVerilog)
  - **Re-use Design Methodology**
    - Invent of modern ASIC block-to-top design methodology
  - **Efficient and Effective Verification Methodology**
    - Invent of Constraint Random, Coverage driven and Object oriented base modern verification methodology
  - **CAD Automation**
    - Invent of Logic Synthesis and Physical Synthesis tools :
      - to convert hardware description language-based design to logic gates and then to transistor level design
    - Invent of Simulator for Verification :
      - Verify designs that are developed using Hardware Description Languages prior to fabrication process

Why Hardware Description Language ?

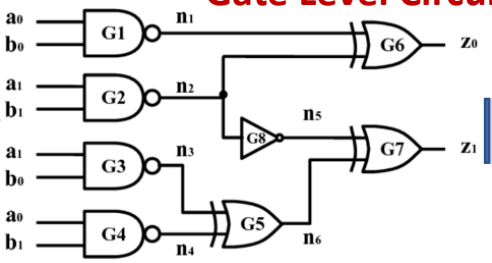
- **Hardware description languages such as SystemVerilog, Verilog, VHDL allows digital design engineers :**
  - To concisely express complex ***digital logic*** circuit behavior and hardware algorithms using constructs which are very close to natural programming languages such as '**C**'
  - To design large scale digital systems without dealing with the complexity of gate and transistor level implementation
- **Computer Aided Design (CAD) tools provides a software platform to virtualize and mimic real hardware before the actual chip is fabricated**
  - Using logic simulator, engineers can perform simulation on SystemVerilog code to test circuit behavior on computer before actual hardware (chip) is manufactured
  - Using synthesis tool, a SystemVerilog program can be automatically converted to gate-level circuit and then to transistor level layout
  - **Synthesis tools takes away the burden from the design engineer the complexity of hardware implementation using logic gates and transistors**

SystemVerilog HDL Code to Model Digital Hardware Design



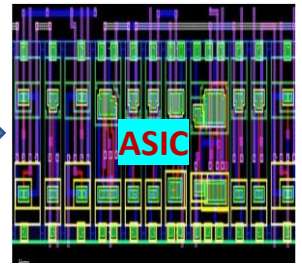
Logic Synthesis

Gate Level Circuit

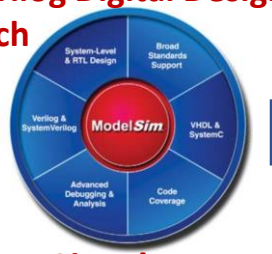


Physical Synthesis

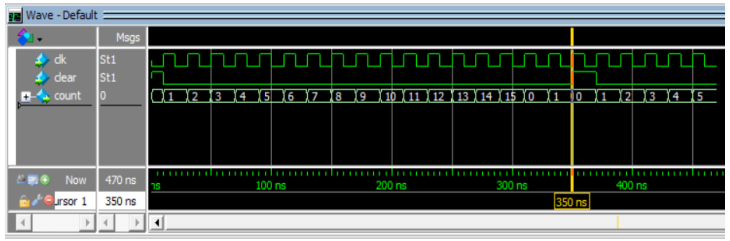
Transistor Level Layout



System Verilog Digital Design & Testbench Code



Simulator



Waveform to Virtually Confirm Hardware Design behavior before manufacturing

Modern Hardware Design Languages

- **There are two major Hardware Design Languages :**
  - **VHDL** (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit **H**ardware **D**escription **L**anguage)
  - **SystemVerilog** (extension of **Verilog**)

VHDL

- **VHDL** is more verbose language and it is also has a non-C like syntax (**More lines of code**)
- It is a strongly typed language
  - Each data type (integer, character, or etc.) has been predefined by the language itself.
  - All values or variables defined in the code must be described by one of the data types
- **Advance OOP based verification constructs are not supported !**

SystemVerilog

- **SystemVerilog** is more compact language and it is more like C language (**lesser lines of code**) with lower level of implementation
- It is not a strongly typed language
  - Supports user defined data types
- It is a unified language for design and verification
  - **Advance OOP based verification constructs are supported ! – Enables scalability and re-used of test code across designs**
- SystemVerilog covers best features of VHDL and Verilog hardware description languages

VHDL and SystemVerilog Example Code

VHDL code for a 4-bit unsigned down counter with synchronous set

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
  port( clock, S : in  std_logic;
        Q : out std_logic_vector(3 downto 0));
end counter;
architecture behavioral of counter is
  signal tmp: std_logic_vector(3 downto 0);
  begin
    process (clock)
      begin
        if (clock'event and clock='1') then
          if (S='1') then
            tmp <= "1111";
          else
            tmp <= tmp - 1;
          end if;
        end if;
    end process;
    Q <= tmp;
end behavioral;
```

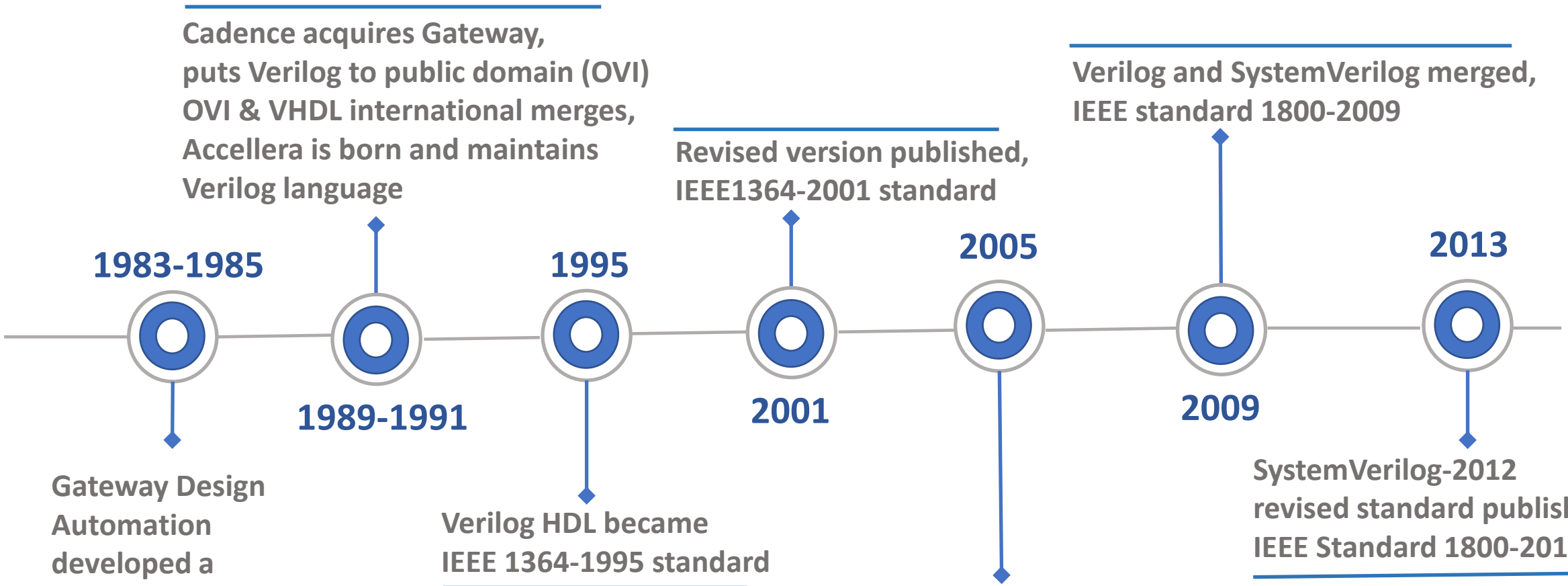
Verbose Representation of Hardware Design

SystemVerilog code for a 4-bit unsigned down counter with synchronous set

```
module counter (
  input logic clock, S;
  output logic [3:0] Q
);
logic [3:0] tmp;
always@(posedge clock)
 begin
   if (S)
     tmp = 4'b1111;
   else (S)
     tmp = tmp – 1;
   end
   assign Q = tmp;
 end
endmodule
```

Concise Representation of Hardware Design

SystemVerilog Evolution



** Note :

- Verilog-AMS was developed to describe Analog logic of a System and verify it along with digital logic prior to fabrication of a chip
- Circuit described in Verilog-AMS cannot be converted to transistor layout using automation such as physical synthesis

Why learn SystemVerilog Hardware Description Language ?

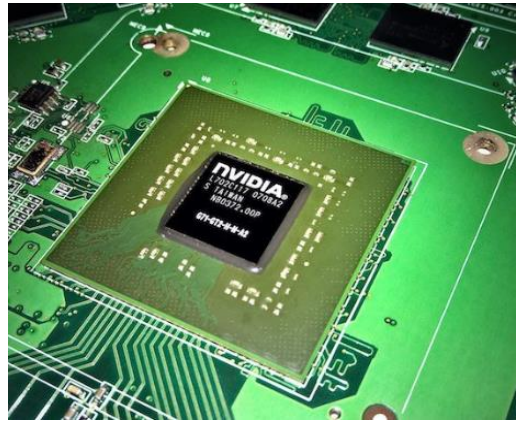
- ❑ Both FPGA and ASIC based Hardware Solutions are designed using SystemVerilog
- ❑ Most of the modern System-On-Chip and hardware designs are built using SystemVerilog
- ❑ Most hardware engineering jobs requires SystemVerilog based design knowledge

Mobile Phone SOC

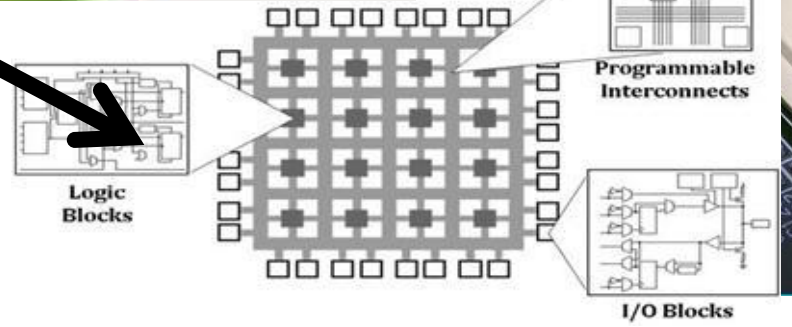
CPU

GPU

ASIC Design

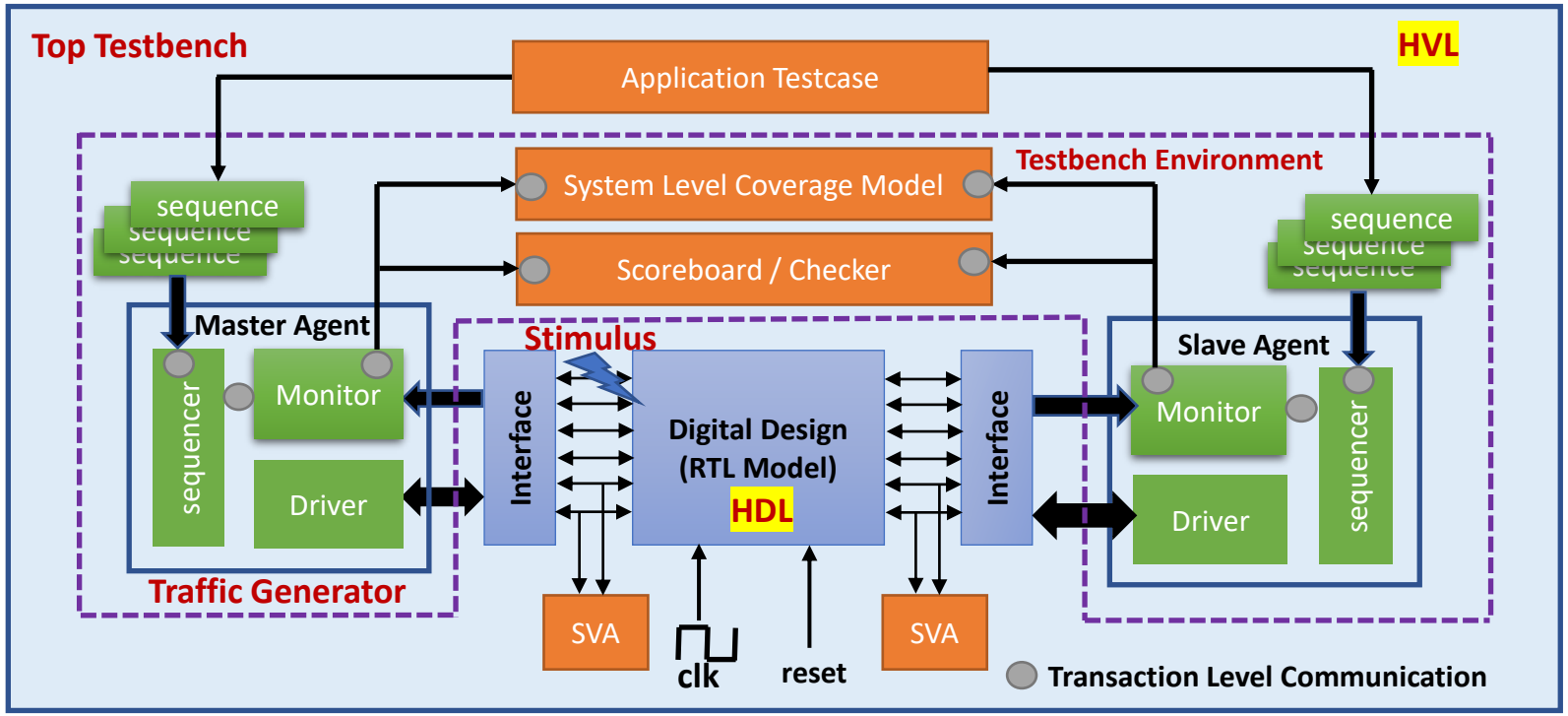


FPGA Design



Why SystemVerilog over other Hardware Programming Languages ?

- **SystemVerilog is a unified language for both design and verification !**
  - SystemVerilog is both H**D**L and H**V**L. Seamless data flow from testbench world to design world !
  - Design described using synthesizable constructs. More concise representation of design **compared to VHDL**
  - **Re-usable, scalable, constraint and coverage driven verification environment** using C++ like constructs (**not supported in VHDL**)
  - Advance methodologies such as UVM, VMM for verification and enables hardware acceleration
- **Can interface with C and System-C languages.**
- **Provides extensions to create Analog Mixed Signal Models (Verilog AMS)**

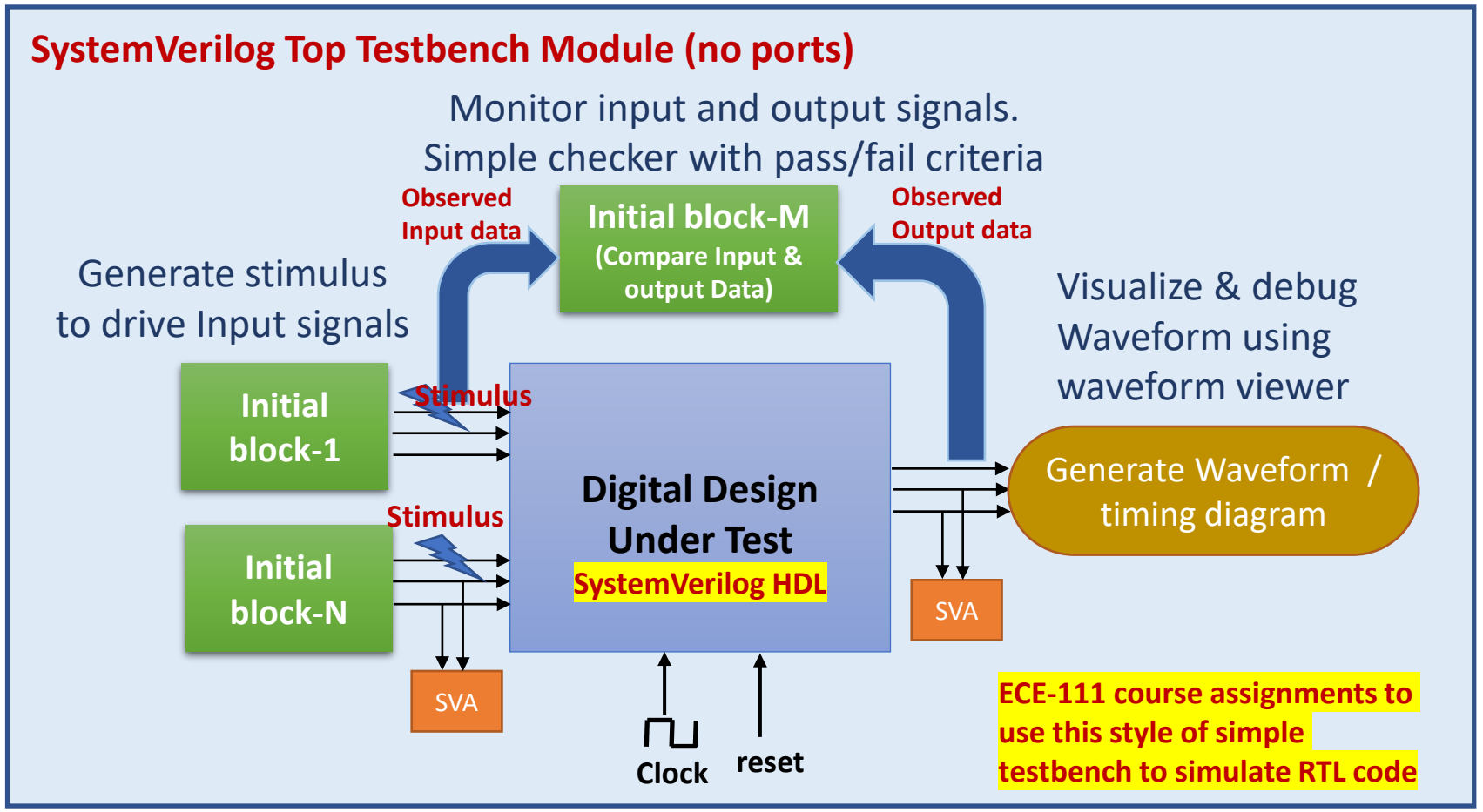


- **HDL** is used for developing design code (RTL)
  - HDL code is synthesizable and is part of final product
  - Non-OOP based constructs
- **HVL** is used for developing testbench to stimulate and verify design
  - HVL code is non-synthesizable and is not part of final product
  - OOP based constructs

Traditional SystemVerilog Testbench

❑ Traditional SystemVerilog Testbench

- No usage of SystemVerilog advance OOP constructs. No UVM or VMM based environment
- No usage of coverage and constraints.
- One or more initial blocks running concurrently to apply stimulus to input signals
- Monitor input and output signals in initial block with self directed checking mechanism



SystemVerilog Unified Language !

Models
(Concept Design)

- ❑ **Transaction and Register accurate models** of design and SOC for architecture exploration and software validation. (Non-synthesizable code)
- ❑ **Cycle accurate reference models** which can be used during architecture exploration and during verification to speed up simulations and also as a checker

Design
(Actual Design)

- ❑ **Synthesizable RTL and Gate level model** of a design

Verification
(Testing of Actual Design)

- ❑ **Testbench code for verification** using constraints, classes and other object oriented programming constructs provided in SystemVerilog LRM. (Non-synthesizable)
- ❑ Functional coverage models (Non-synthesizable)
- ❑ Assertion checkers (Non-synthesizable)

System Verilog and Verilog Constructs

Credit : Stuart Sutherland

HVL

HDL

SystemVerilog-2005/2009/2012

verification

assertions	mailboxes	classes	dynamic arrays	2-state types
test program blocks	semaphores	inheritance	associative arrays	shortreal type
clocking domains	constrained random values	strings	queues	globals
process control	direct C function calls	references	checkers	let macros

Non-Synthesizable Constructs

Synthesizable Constructs

design

interfaces	packed arrays	break	enum	++ -- += -= *= /=
nested hierarchy	array assignments	continue	typedef	>>= <<= >>>= <<<=
unrestricted ports	unique/priority case/if	return	structures	&= |= ^= %=
automatic port connect	void functions	do–while	unions	==? !=?
enhanced literals	function input defaults	case inside	2-state types	inside
time values and units	function array args	aliasing	packages	streaming
specialized procedures	parameterized types	const	$unit	casting

Verilog-2005

uwire `begin_keywords `pragma $clog2

Verilog-2001

ANSI C style ports	standard file I/O	(* attributes *)	multi dimensional arrays
generate	$value$plusargs	configurations	signed types
localparam	`ifndef `elsif `line	memory part selects	automatic
constant functions	@*	variable part select	** (power operator)

Verilog-1995 (created in 1984)

modules	$finish $fopen $fclose	initial	wire reg	begin–end	+ = * /
parameters	$display $write	disable	integer real	while	%
function/tasks	$monitor	events	time	for forever	>> <<
always @	`define `ifdef `else	wait # @	packed arrays	if–else	
assign	`include `timescale	fork–join	2D memory	repeat	

ECE-111 Course Emphasis

Honor Code

- **The UCSD Student Conduct Code**
  https://students.ucsd.edu/sponsor/student-conduct/regulations/22.00.html

- **Violations will be reported to the Student Conduct Office (as well as failing the class)**