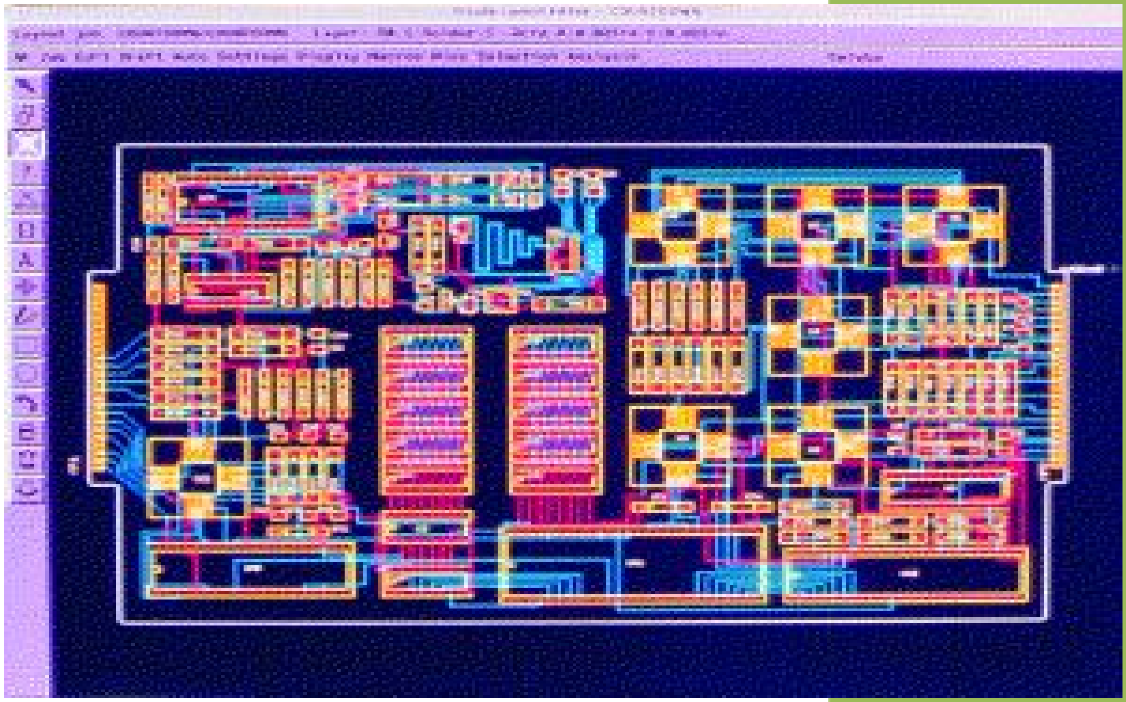
Computer Engineering Department
Faculty of Engineering
Cairo University

Spring 2020
CMP 305
VLSI

ODE Solver

System Description

2020



VLSI Semester Term Project

VLSI Project

Computer Engineering Department

Objectives

- To better understand Digital Design Flow
- To be proficient at VHDL/verilog
- To create real world hardware applications
- To understand the mapping between Algorithm Specifications & Hardware Implementation
- To understand Design Trade-offs
- To learn how to optimize Hardware Designs
- To practice scientific writing

Introduction

Solving Ordinary Differential Equations is a crucial part of modeling and simulating systems. It is used in order to model how a system state changes over time. Simulating models help us with designing systems and controllers. Solving an ODE analytically can be a very hard and sometimes impossible approach. Therefore, it is a common practice to solve ODE using approximation numerical methods.

In this project, you will design a detailed low-level design of a chip that provides multiple solvers for an ODE. The chip should be a stand-alone chip that reads the system's data from the user, applies the chosen solver, and generates the output. Solving complicated systems ODEs can be very costly. It is required to design the chip to assist the cpu in solving the ODEs in a faster time. Such dedicated chips are called **accelerators**. You will also experience the whole cycle of Design & Fabricating a system on chip, best described as in Figure 1.

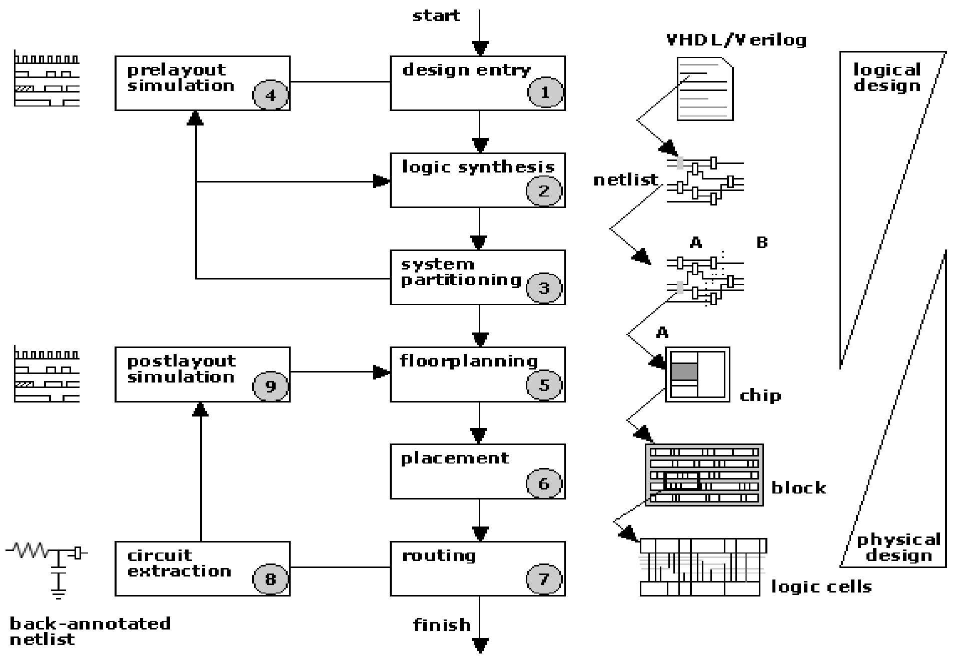
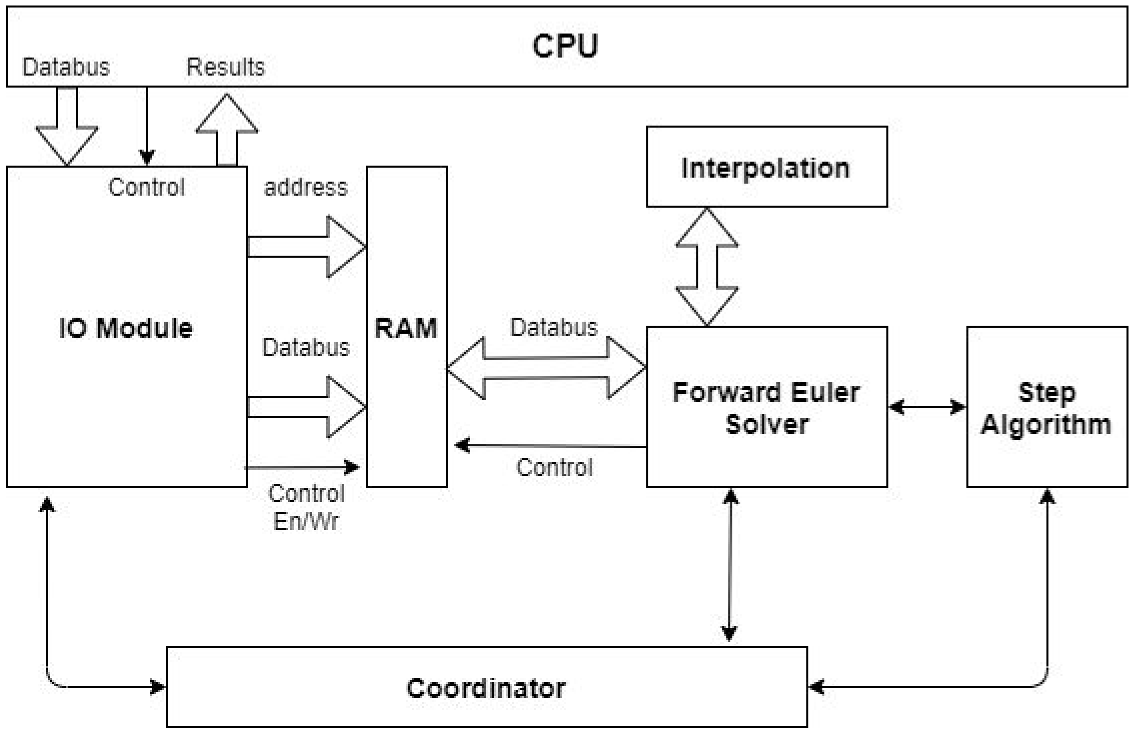


Figure 1: ASIC Chip Design Flow

Design Requirements



The chip is required to solve the equation $X' = AX + BU$ where:

- X is a vector of size N*1 that represents the state of the system
- A is a matrix of size N*N
- U is a vector of size M*1 that represents external input to the system
- B is a matrix of size N*M

The chip consists of four main modules, IO module, forward euler solver, interpolation module and step algorithm module.

- The IO module is responsible for loading the initial state of the system, the system parameters, and the solver options.
- The forward euler solver module is responsible for calculating X of the next time step X_{n+1}.
- The time step algorithm module is responsible for calculating the timer step depending on the chosen mode.
- The interpolation module is responsible for calculating U at unknown timesteps.
- The coordinator orchestrates the execution of the chip through its different execution steps.

Numerical Method Explanation

Forward Euler

The chip will follow a simple method to calculate X vector at each time step. We use the forward euler method to approximate X at tn+1, with the following equation.

$X_{n+1} = X_n + hX'_n$ Where $X'_n = AX_n + BU_n$

Updated 12_March

Assume
-- Time Step (h) = 0.5
-- X0 = [1 , 2]T (T stands for Transpose)
-- A = [[1,1],[2,2]]
-- B = [[1,2,3],[4,5,6]]
--U0 = [1,2,3]T
-- U1= [4,5,6]T , U at time step 1 (time = 0.5)

It is required to calculate X at time 1
X1 at 0.5 =X0 + h(A*X0 + B*U0) = [3,6]T + [14,32]T = [1,2] + 0.5*[17,38] = [9.5,21]
X2 at 1 = X1 + h(A*X1 + B*U1) = [9.5,21] + 0.5([30.5,61]T + [32,77]T) = [40.75, 55]T $\leftarrow$ the required result

In Phase 2 : You are required to implement the above algorithm using HDL code of your choice. Take care that all numbers are NOT integers (Fixed Point or Floating Point).

Variable-step Algorithm

A variable step solver can be used to solve stiff ODEs. Given an error tolerance the solver tries to vary the time-step until the error is smaller than the tolerance. The algorithm does the following:

Starting with an initial timestep h and tolerance L

1. Calculate $X_{n+1}^{(0)}$ using one step of forward euler with timestep h
2. Calculate $X_{n+1}^{(1)}$ using two steps of forward euler with timestep h/2
3. Calculate error (e) as $X_{n+1}^{(1)} - X_{n+1}^{(0)}$
4. Repeat until e <= L
   - Calculate hnew = $(0.9*h^2*L)/(e)$
   - Recalculate $X_{n+1}^{(1)}$, $X_{n+1}^{(0)}$, and e
5. Proceed to next time step

In Phase 2 : You are required to implement the above algorithm using HDL code of your choice. Take care that all numbers are NOT integers (Fixed Point or Floating Point).

Interpolation Module

We have mentioned in the previous step that we will use U_n, however in the input stage we are only given U at a few time steps. We will use linear interpolation to calculate U at the unknown times.

Assume that we have U at time n (T_n) and at time z (T_z), to calculate U at time k where Tn<Tk<Tz

$$U_k = U_n + (T_k - T_n)\frac{U_z - U_n}{T_z - T_n}$$

```
-- Un = [ 1,2,3]T
-- Uz= [4,5,6]T  ⟵ vector
-- Tn = 0,  Tz = 1 , Tk = 0.5   ⟵ scaler, always one digit
Uk = [1,2,3]T + (0.5-0) x ([4,5,6]T-[1,2,3]T)/(1-0) =  [ 2.5,3.5,4.5]T
```

In Phase 2 : You are required to implement the above algorithm using HDL code of your choice. Take care that all numbers are NOT integers (Fixed Point or Floating Point).

System Specifications

The system should have the following I/O ports:

Port	Direction	Size
Clk	IN	1 bit
Rst	IN	1 bit
Interrupt	IN	1 bit
Load/Process	IN	1 bit
Done	OUT	1 bit
Databus	INOUT	32 bits

Module Details

IO and Coordinator Modules

The IO module is responsible for communicating with the CPU. The cpu reads a JSON file with the data and configuration - listed below -, compresses it and sends it over to the chip through a 32 bit parallel port. Moreover, the CPU uses additional control signals to control the chip (for example: an interrupt signal to notify the chip that a new command will be sent). The CPU sends control signals & data, and receives status signals & result. The I/O module receives the CPU commands & data, and passes the commands to the rest of the chip.

The CPU can send the following control signals to the chip:

- System clock: Controls the chip frequency
- Reset signal: Restart signal that initialize the chip for processing
- Interrupt signal: A notification for the chip that a command is sent
- Load/Process signal: A command signal that chooses between loading a new data or processing the loaded data.

You can add any additional signal you need to allow the CPU control the chip (justify the additional signals).

The IO module will decompress the sent data and writes them to the local RAM. The data read from the CPU can be listed as follows.

- Dimension of X vector (N). N ranges from 1-50
- Dimension of U vector (M). M ranges from 1-50
- Solver mode (Fixed-step or Variable-step) (0→ Fixed Step, 1→ variable step)
- Timestep (h). (initial timestep if variable-step-size mode)
- Error tolerance. (for variable-step-size mode)
- Fixed-point or floating-point precision (1→ fixedpoint, 2 → fp64, 3→ fp32)
- Count of Timesteps needed (T:count of the number of outputs)
- Matrix A (N*N)
- Matrix B (N*M)
- Initial value of X. (X0)
- Time points where solutions are required
- Initial U vector
- U vector at time points where solution is required a matrix of TxM (where each row represents a U vector, you will need to calculate Us at intermediate Steps Using Interpolation)

Sample Input in JSON Format

```
{
  "N" : 3,
  "M" : 2,
  "Mode":1,
  "H":0.1,
  "Err":0.02,
  "Fixedpoint":1,
  "Count":2,
  "A": [[1.3,2,3.4],[4,5,6],[7,8,9]],
  "B": [[0.1,2.5],[3,4],[5,6]],
  "X0": [0.1,2,4.5],
  "T": [1,2],
  "U0": [3.3 , 2.1],
  "Us": [[2.3,6.4],[3.3,10]]
}
```

The working scenario should be as follows:

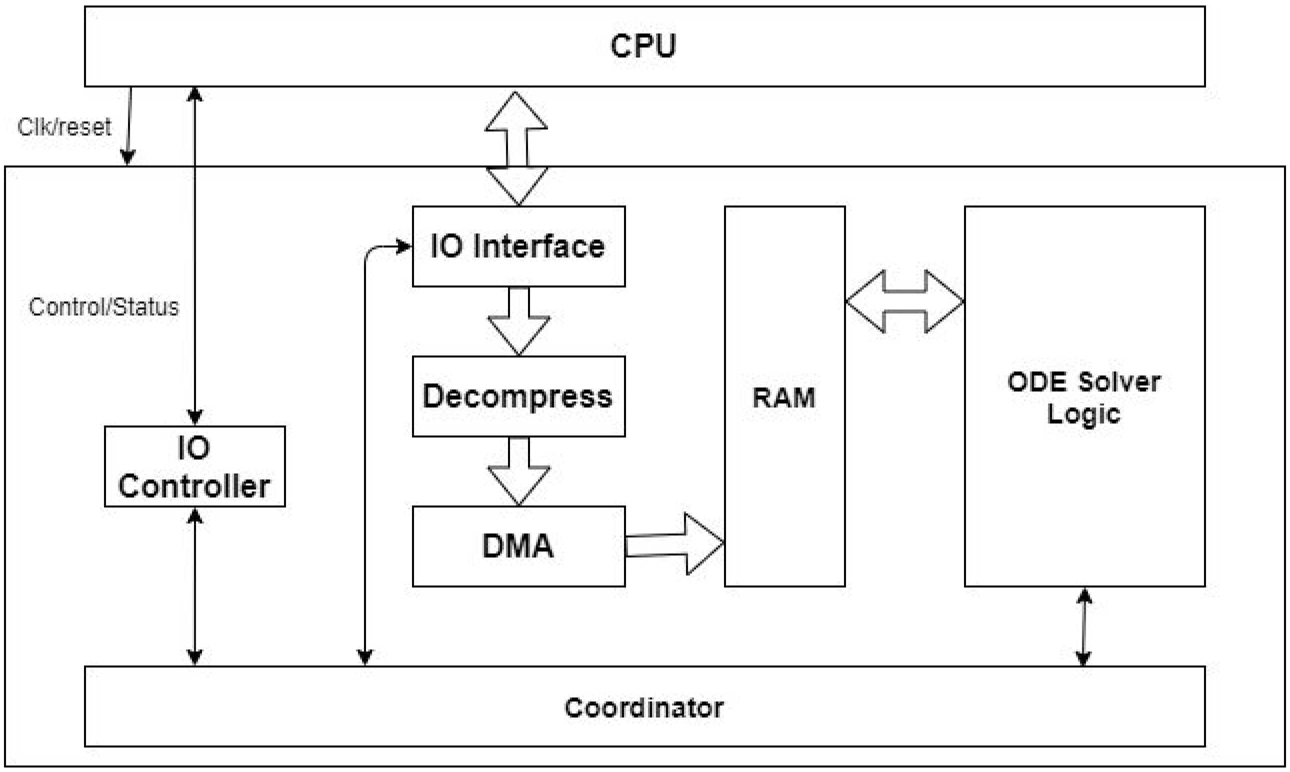
1. The CPU generates clock signal for the chip (constant frequency)
2. The CPU sends a reset signal to restart the chip (pulse signal for 1 clock cycle)
3. The CPU sends Load command + interrupt signal to notify the chip of the command
4. The CPU sends the **compressed** data (compression will be explained below)
5. The I/O sends done loading signal to the CPU
6. The CPU sends process command + interrupt signal to notify the chip of the command
7. The I/O passes the command to the chip
8. The I/O passes the done processing signal + result from the chip
9. The Result is the time followed by the output vector. (Take care the path is still 32-bit) .

You are required to Make a Software to mimic the CPU Behaviour,

1. **Read input json**
2. **Compress data**
3. **Schedule and send signals as the scenario above**

Your choice of the frequency should be relative to the Accelerator Chip You will design.
More Details Below

The IO and coordinator module block diagram should look like the following:



Note that the RAM/Coordinator are common modules that are used/controlled by other main modules in the chip, so you need to coordinate with the rest of the team.

Compression and Decompression

For compression & decompression you will use bit-level Run-Length Encoding (RLE) algorithm. The compression step is implemented as a software script, while the decompression step is implemented in your decompress module. Additionally, you will need to write a Tcl file (or a test-bench) that reads the compressed file & send it to the chip.

Compression step is done by the following:

1. Read the json file with the data.
2. Convert the data into binary digits in row format. Rows should be as follows
    a. One row containing N, M, Solver Mode, TimeStep, Error tolerance (if any), Precision mode, and Count of timesteps Needed
    b. Each row of matrix A separately
    c. Each row of matrix B separately
    d. X0
    e. Time points required for solution
    f. Initial U
    g. Each additional U vector in a separate row
    h. Take Care, in case of Fixed point, you need to consider how will you handle scale factors, whether it will initially all start with the same scale factor or different factors.
    i. add any additional flags you need to decompress the data.
3. Loop on the data row by row & compute the RLE compressed row (bit-level RLE)
4. Save the compressed file to be submitted to the chip in runtime

Submitting the data to the chip is done by the following:

1. Open the compressed file
2. Read Line by line
3. Split the line into the input databus size (32-bit)
4. Generate the control signals required to load the data to the chip

Decompression step is done by the following:

1. Wait for the load control signal
2. Receive the compressed data
3. For each line, compute the original row using RLE decompression
4. Save the original data in the chip RAM in a specific place

Rules & Regulations

- Phases Deliveries are scheduled on your Calendar
- You are free to design your system as you want as long as you perform the required functionality.
- You could add any additional I/O ports and modules according to your need, but you have to justify your design choices.
- Your design should be modifiable to meet the design constraints in the next phases.

- Your design should be integratable with the rest of the bigger team.
- Take care that your design is logically mappable to hardware or you will have to repeat it all again.
- Open your mind and don't limit yourself .
- **You are not allowed to copy from any external resources in your implementation and you may suffer from a big penalty.**
- You are allowed to consult external resources for Design but Do your OWN & you have to fully understand it.
- **Grades are based Mainly on Individual work + your team work . if you didn't work and the project was complete you will still get a zero grade. And we really mean it.**
- The document is variable to change with a previous notification.

References for more Information

1. https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm
2. https://en.wikipedia.org/wiki/Binary_multiplier
3. http://www.geoffknagge.com/fyp/booth.shtml
4. http://www.ecs.umass.edu/ece/koren/arith/simulator/Booth/
5. https://guest.iis.ee.ethz.ch/~zimmi/publications/adder_arch.pdf
6. http://iosrjournals.org/iosr-jvlsi/papers/vol9-issue2/Series-1/B0902010613.pdf
7. https://en.wikipedia.org/wiki/Fixed-point_arithmetic
8. https://spin.atomicobject.com/2012/03/15/simple-fixed-point-math/
9. http://www.iosrjournals.org/iosr-jvlsi/papers/vol4-issue2/Version-1/E04212935.pdf
10. https://en.wikipedia.org/wiki/Run-length_encoding
11. https://www.dcode.fr/rle-compression
12. https://www.fileformat.info/mirror/egff/ch09_03.htm
13. https://www.allaboutcircuits.com/technical-articles/fixed-point-representation-the-q-format-and-addition-examples/
14. https://www.allaboutcircuits.com/technical-articles/multiplication-examples-using-the-fixed-point-representation/
15. http://apachepersonal.miun.se/~amiyou/micro/Lecture3.pdf