ELEC40006: 1st Year Electronics Design Project 2020

Initial Specification

Esther Perea and Ed Stott
May 2020

Introduction

The purpose of this module is to bring together the knowledge and skills you have gained throughout this academic year by applying them to a practical problem.

You will also acquire new skills that will help you complete the project successfully.

The project is conducted in groups of three.

Timeline

Week no.	Date	Action item
2	5th May	Project Brief released
	10th May	Project selection and group formation deadline
3	11th May	Confirmation of group composition
9	14th June	Report, video, code and source files submission

Deliverables

Report (50%): The report is a formal documentation of all the technical and non-technical work you have done on the project. By this time all your design decisions will have been made and you will be able to document the performance of various aspects of your system / simulation. You should also have a clear plan for any work outstanding before you can complete the demonstration. One team member should act as overall editor to ensure that the report is consistent.

Video demonstration (50%): The team will record a video explaining their final design, alongside a demonstration.

Code and source files for plagiarism checking

Additional information for all three deliverables will be given at a later date.

Rules for sharing

1. You **may** discuss the project with students from other groups
2. You **may** share ideas and try out others' suggestions
3. You **may** share links to useful resources
4. You **may not** share your own designs or data (schematic files, code, screenshots, plots)
5. You **may not** share elements of your report or demo video

Project Options

Choose one of the following options by Thursday 5th May:

Option 1: Analogue Music Synthesiser

Analogue music production remains popular today, despite the availability of digital alternatives, thanks to its authenticity. In this option, you will design a circuit for an analogue music synthesiser and simulate it using LTSpice.

Essential Requirements

a. The synthesiser must generate audio frequency tones for the 7 notes in the C major scale in an octave of your choice
b. The input to the synthesiser should be 7 voltage sources, each representing one key of a keyboard. 5V represents a pressed key and 0V a released key.
c. All the components in your simulation must be 'real' devices (passive components, or semiconductors with part numbers), except for the voltage sources used as DC power supplies or to represent the keyboard inputs. No 'behavioural' components or any other sources are permitted in the final design, but they can be used to test sub-circuits. Ask if you are not sure about a component.
d. The synthesiser must drive a loudspeaker with 8Ω impedance

Optional Requirements

Some design choices are left open:

a. The synthesiser can be monophonic or polyphonic
b. The implementation can use discrete transistors or integrated circuits

You can also add any functionality that enhances your design. Possibilities include:

a. Choice of waveforms/instruments
b. Modulation, including tremolo (amplitude modulation) and vibrato (frequency modulation)
c. Enveloping (another form of amplitude modulation)
d. Filtering
e. Low frequency oscillator to drive the modulation and filter blocks
f. Stereo modulation effects
g. Arpeggiator

You'll need to conduct your own research to decide what functionality should be included and how it should be implemented. Existing circuits can be adapted if they are correctly referenced, and the report shows that their operation is understood, and their use is appropriate.

Use the PWL source type for the keyboard inputs because this will allow you to simulate a sequence of key presses over time.

Use the MATLAB script supplied on Blackboard to listen to your outputs and visualise them with a spectrogram plot. You should adopt a modular and systematic approach to break the system down into sub-circuits, and develop each sub-circuit independently before you integrate the overall system.

Evaluation

1. The quality of a music synthesiser is a subjective criterion, but try comparing the sound it makes with recorded samples of commercial synthesisers that you find on the internet
2. Calculate the BOM (bill of materials) cost for your design by summing up the costs of all the components you have used
3. Find the power consumption of your design and investigate if it varies as you use different functions. Find how the power consumption breaks down between the different circuit blocks.

Option 2: Central Processing Unit

An efficient CPU implementation is optimised to solve commonly occurring computing problems. Features must be chosen carefully to achieve the best performance in the greatest number of applications for the smallest number of transistors.

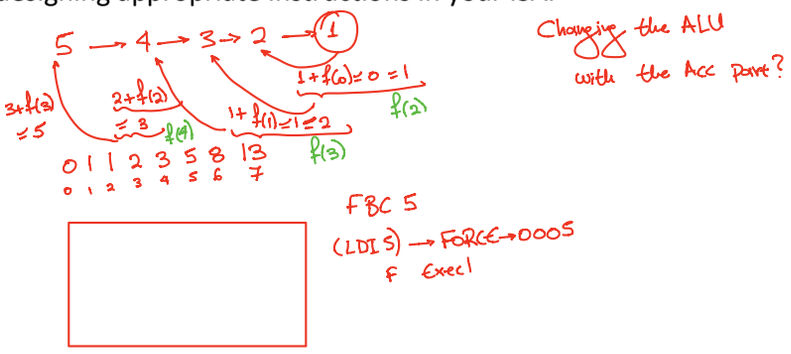
Design a CPU with an ISA that is optimised to execute the following algorithms efficiently. Describe the CPU using block schematics and Verilog modules, and simulate using Quartus Prime. You will need to implement the benchmark algorithms in the machine code for your ISA.

You may assume that the CPU word length (and the C++ `int` variables used below) are all 16 bits. Your CPU should implement enough memory for at least 2K words of instructions and 2K words of data. Your CPU may use separate instruction and data memory units, or one combined memory unit.

Calculate Fibonacci numbers using recursion

This benchmark requires a *stack* to keep track of all the nested intermediate data values in a (very inefficient) recursive implementation of the `fib()` function. Alternative implementations that don't require a stack are not permitted. The stack is a history of all the functions the CPU is currently executing, so that when it finishes a function it can return to where it left off, including all the local variables that were being used. You could implement the stack with custom hardware or using normal data memory, designing appropriate instructions in your ISA.

```
int fib(const int n){
     int y;
     if (n <= 1) y = 1;
     else {
         y = fib(n-1)
         y = y + fib(n-2);
     return y;
}
```



A typical use of the benchmark would be `fib(5)`.

Calculate pseudo-random integers with a linear congruential generator (LCG)

One commonly used way to implement random numbers on a computer is the linear congruential generator which computes the sequence: $x_{n+1} = (ax_n + b) \bmod 2^N$. For suitably chosen a, b and N this sequence approximates random numbers in the range [0..2^N-1]. Typically, N is chosen to be the computer word length, so the modulo operation is truncation that happens at no cost.

This example requires multiplication: there are many ways to implement this in hardware or software and you may choose whatever works, however you are not allowed to use a Verilog IP blocks, nor to use a Verilog multiply operator, since they use fixed hardware constructs that restrict implementation choices.

```
int lcong(
      const unsigned int a,
      const unsigned int b,
      const int n,
      const unsigned int s)
{
      unsigned int y = s;
```

```
        unsigned int sum = 0;
        for (int i = n ; i > 0; i--){
           y = y*a + b // calculate the new pseudo-random number
           sum = sum + y // add it to the total
        }
        return sum;
}
```

The benchmark code finds N numbers in the sequence and adds them together. A typical problem here would have a = 25385, b = 3, n = 8. You might try to investigate which values of a, b lead to optimal generators, achieving the longest possible sequence before it repeats. Parameter s is a seed – it defines a starting point for the sequence.

Traverse a linked list to find an item

This example steps through items in a linked list and search for a given value. Since each item in the list is reached by a *pointer* in the previous item, the CPU must use an efficient form of *indirect addressing* to traverse the data.

```
typedef struct item{
     int value;
     struct item *next;
} item_t;

item_t* find(const int x, item_t* head){
     while (head->value != x){
           head = head->next;
           if (head == NULL) break;
     return head;
}
```

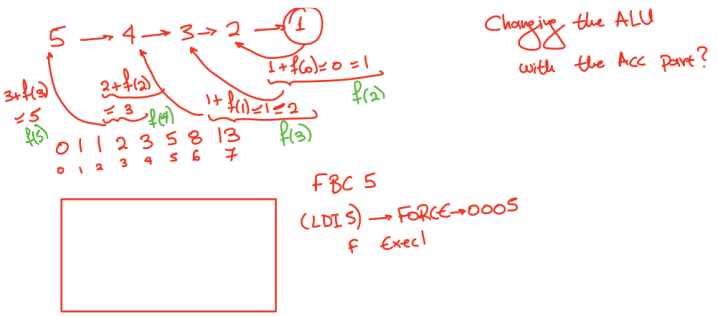
You will need to create a linked list in your RAM initialisation data to test this algorithm. A typical problem here would consist of a list of length 10 nodes.

Evaluation

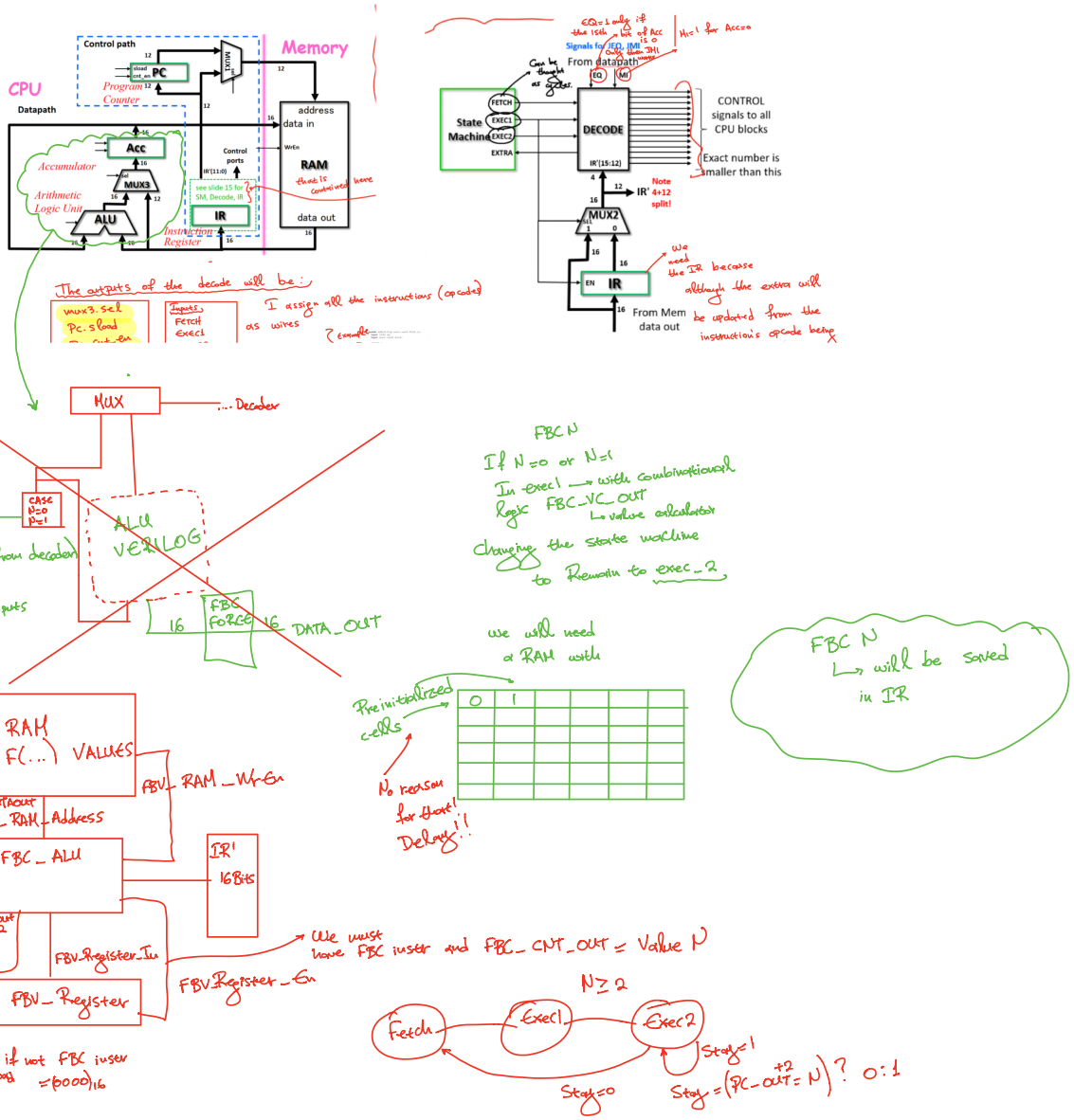
Your solution should be evaluated against the following criteria:

1. Test your CPU for correctness by writing the benchmark algorithms in assembler and running them on trial data. Compare to results calculated with a paper analysis
2. Find the speed of the COU by counting the number of CPU cycles required to run the benchmarks and see how this figure changes with the size of the problem (e.g. length of the list) and any implementation options you have tried. Use Quartus to find the maximum clock speed of your design (see detailed instructions), and hence calculate the minimum execution time in microseconds. When trading off performance of each circuit you should minimise the geometric mean time: $(T_1T_2T_3)^{1/3}$ where T_1, T_2, T_3 are the times of each algorithm. This gives equal weight to each algorithm.
3. The power consumed by a digital circuit relates approximately to the number of logic gates and the clock speed. Find the number of logic gates (see detailed instructions) to estimate the overall power consumption.

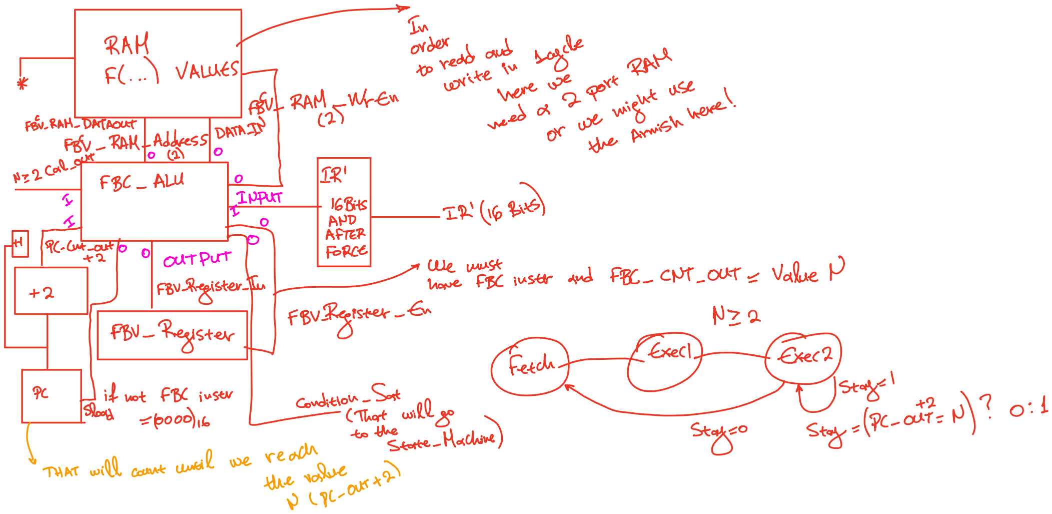
```
int fib(const int n){
    int y;
    if (n <= 1) y = 1;
    else {
        y = fib(n-1)
        y = y + fib(n-2);
    return y;
}
```



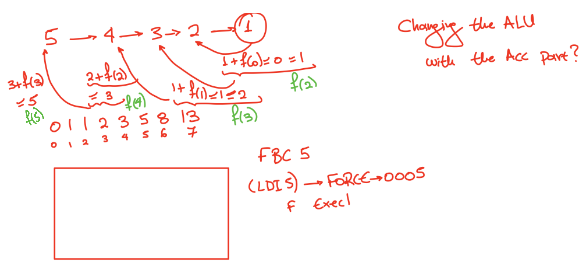
A typical use of the benchmark would be `fib(5)`.



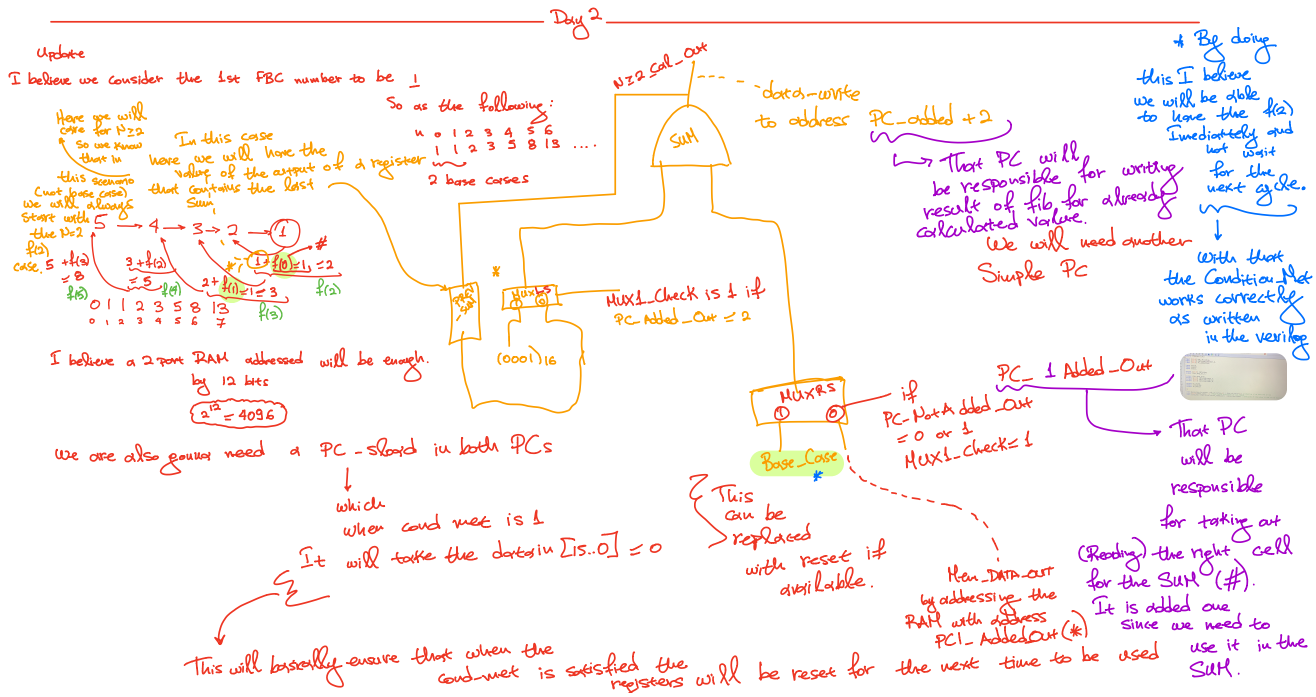
Πάω σε ΜΜΟ (μπορεί να είναι και ΜΜΟ ppl) δεν μας νοιάζει για την ώρα.




```
int fib(const int n){
    int y;
    if (n <= 1) y = 1;
    else {
        y = fib(n-1)
        y = y + fib(n-2);
    return y;
}
```



A typical use of the benchmark would be fib(5).



Therefore I believe we could use just 1 PC and then use verilog for the +1/+2 fixed values.

So basically the calculation for values $N \geq 2$ will be taken from the block above → taken into the ALU (which according to whether the condition is met and we dont have the base case → It will output that. Otherwise If we have the condition met and the base case it will just output the base case

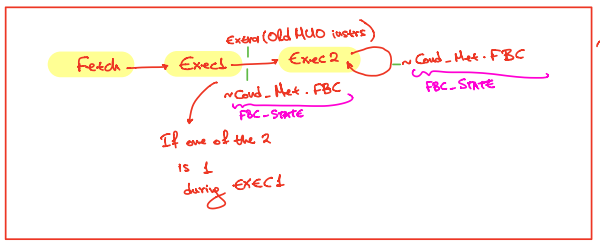
Day 3

For N=0 or N=1 or N=2 we will have instantly during Exec1 the fib(N) result (I believe we can do this since all these instructions are based on the base case N=0 or N=1)

- Taken from the verilog instantly. The PC + 2 → Shows instantly what value is calculated so for N=2 the Cond_met will be satisfied from that.
- Taken from the block above instantly
- whereas for N>2 when PC+2 → with value from the instruction we will know that the

So for N=0 or N=1 or N=2 the condition will be satisfied instantly and we wont have to go to EXEC2 (Cond_Met going into the State_Machine)

Otherwise, we will go into EXEC2 and if we still dont have the Cond_Met satisfied, it will remain there until it is satisfied



→ State Machine Changes.

As inputs we will need a Fib_Check → coming from the old Decoder Indicating that we have a Fib opcode

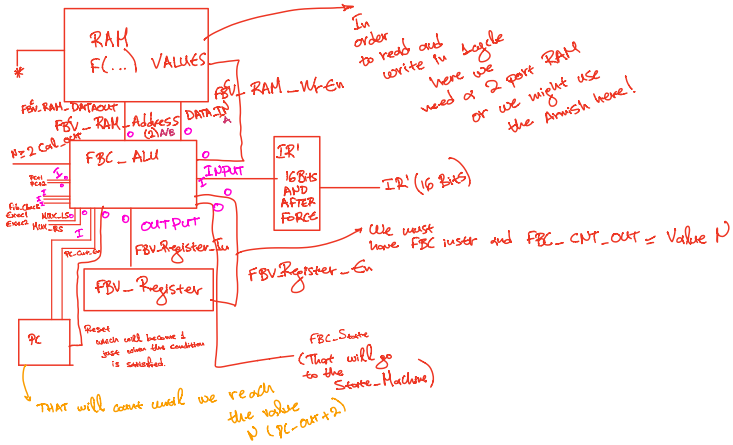
And inputs Exec1/Exec2 → Indicating that we are in EXEC1/EXEC2 to avoid any issues for examples with values during Fetch!

Pay Attention

In the old Decoder we must also create an output which will take out Fib_Check when the opcode is the same as the one of the Fibonacci ALU.

This will be taken as an input to the Fibonacci_ALU and used for the Cond_Met!

Of course in the Fibonacci_ALU we will also take as inputs Exec1 and Exec2 from the old Decoder



1.2. Implementing the State Machine

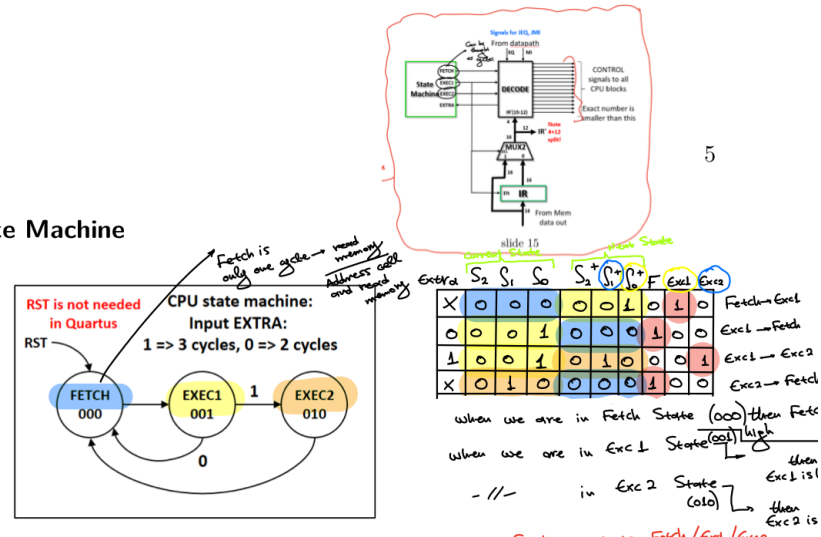
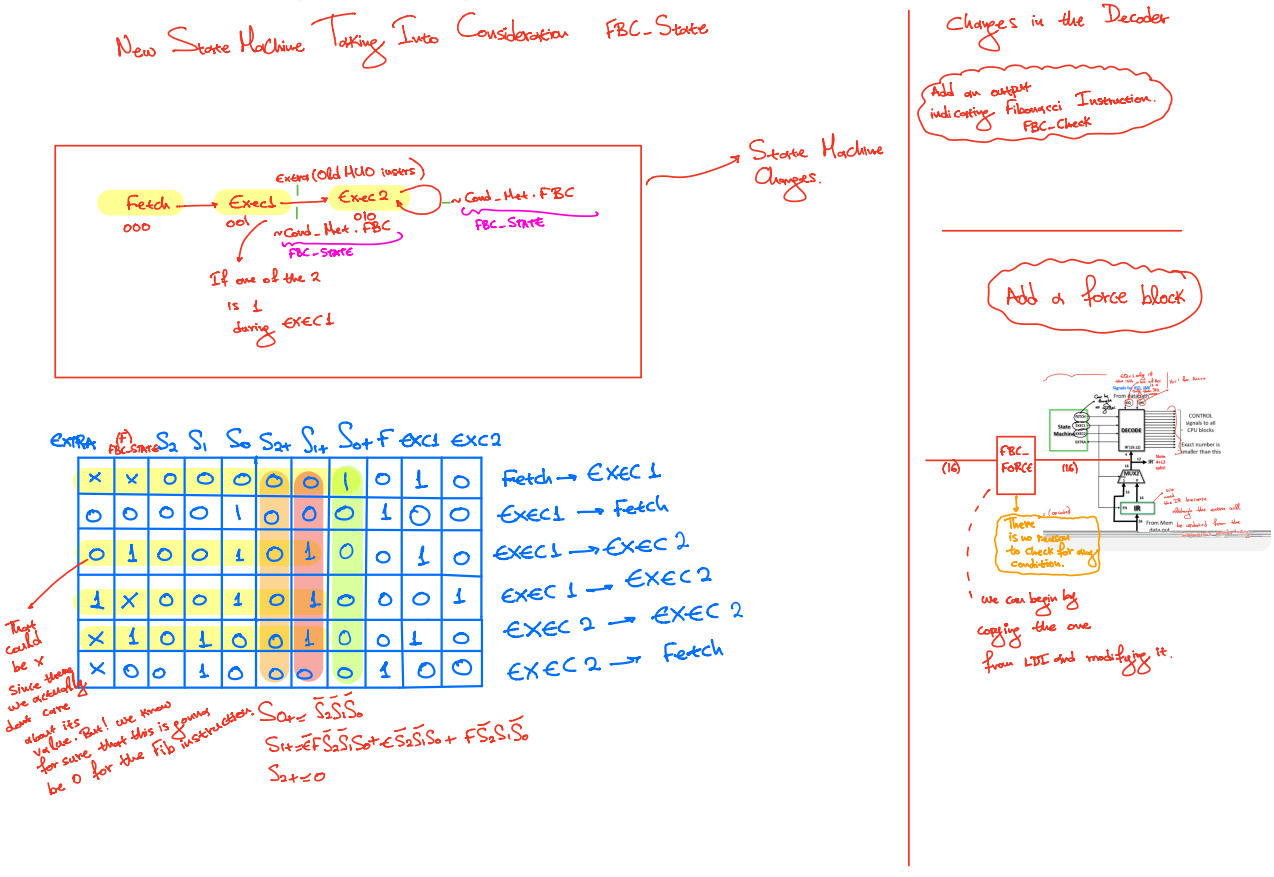


Figure 5: MU0 State Machine: RST is not needed because Quartus initialises all registers to 0

The required boolean logic for NS(2:0) is very simple and can be determined from Figure 5 - the state machine shown in lecture 4 - slide 12. It is not not necessary to have additional logic to generate EXEC1 and EXEC2 outputs because they are available as NS(0) and NS(1), however you can output these signals from the combinational block with the correct name, using a Verilog assignment. That also gives you some flexibility later on if you wish to change the state machine.

□ **Task 3.** *Add your state machine to the schematic as a 3 bit register (for state) and a block of combinational logic implementing the NS(2:0) bus and the FETCH, EXEC1 and EXEC2 outputs.*



V. Notes ofcourse the LPM ADD that we used will keep working additions all the time

But the Result will take the value only if the Cond_Met is satisfied which basically takes into consideration Fib_Check, EXEC1, EXEC2.

→ Only then it will show the correct value, all the other times it will show 0's

The PC will only count if the condition has not been satisfied, (Cond_Met)

↳ which could also for example mean Fib_Check is not satisfied is not satisfied and this starts counting

therefore for Cnt_En which uses ~Cond_Met → By adding & (Exec1 | Exec2) & Fib_Check

{ => We eliminate any possible malfunction

Option 3: Circuit Simulator

Write a software package that performs a transient simulation of a circuit, like LTspice. The main elements of such a system are described below

Parse the netlist file

The netlist should be described in a file using a reduced SPICE format, which will be provided. You will need to read the file and store the circuit in a suitable data structure.

Set up the simulation

A transient simulation takes place by calculating the node voltages at each successive instant in time. The temporal resolution is known as the timestep and the duration is the stop time. A simulation with a stop time of 100ms and a timestep of 1µs would need to calculate 100,000 time instances.

Some components have a value that changes over time with a predefined function. For example, a sinusoidal voltage source has a time-varying voltage: $v_{\mathrm{SIN}}(t) = A\sin(\omega t)$

Some components have a value that depends on the previous history of the circuit. For example, a capacitor has a voltage proportional to the integral of the current through the capacitor: $v_{CAP}(t) = \int \frac{i(t)}{C} dt$

Construct and solve the conductance matrix

You have seen how nodal analysis can be performed by writing an equation for each node that satisfies Kirchoff's current law, then solving these equations simultaneously to find the unknown node voltages. Such systems of linear equations can be solved systematically by writing them in matrix form and solving for the vector of unknowns.

$$\begin{bmatrix} G_{11} & -G_{12} & \dots & -G_{1n} \\ -G_{21} & G_{22} & \cdots & -G_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -G_{n1} & -G_{n2} & \dots & G_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix}$$

Equation for finding a vector of unknown node voltages from a conductance matrix

Here, G_{12} is the conductance directly connecting nodes 1 and 2, G_{11} is the total conductance connected to node 1, i_1 is the total current from current sources entering the node 1, and v_1 is the unknown voltage of node 1. The reference node is not included in the conductance matrix and instead its voltage is defined as zero.

To construct the conductance matrix, inductors are treated as current sources since their current cannot change instantaneously and it is effectively fixed during each simulation iteration. Similarly, capacitors are treated as voltage sources (discussed later) since their voltage cannot change instantaneously.

The solution is found by calculating the inverse of conductance matrix. This is complex but common operation in computing, so it makes sense to use a library, but you must justify your selection of a library and how you have used it in your report.

Process voltage sources

Voltage sources must be treated specially since the conductance of an ideal voltage source is infinite and so it cannot appear in the conductance matrix. If one terminal of the voltage source is connected to the reference node then the node connected to the other terminal can be expressed

simply as something like $v_1 = v_{src}$, e.g. for a source v_{src} volts connected to node 1. The conductances connected to that node are ignored and in matrix form that would look like this:

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ -G_{21} & G_{22} & \cdots & -G_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -G_{n1} & -G_{n2} & \dots & G_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_{src} \\ i_2 \\ \vdots \\ i_n \end{bmatrix}$$

Circuit analysis equation containing a voltage source between node 1 and reference

If the voltage source is connected between two non-reference nodes then two rows are needed in the matrix, since there are two unknown voltages. The first row represents the voltage source, but it now shows the difference between two nodes rather than an absolute, e.g. $v_1 - v_2 = v_{src}$

The second row is KCL for the supernode enveloping the voltage source. It is equal to the sum of the KCL equations for both the nodes in the supernode, which cancels out the conductance of the voltage source. For example, here a voltage source between nodes 1 and 2 creates a supernode S:

$$\begin{bmatrix} 1 & -1 & \dots & 0 \\ 0 & G_{SS} & \cdots & -G_{Sn} \\ \vdots & \vdots & \ddots & \vdots \\ -G_{n1} & -G_{n2} & \dots & G_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_{src} \\ i_S \\ \vdots \\ i_n \end{bmatrix}$$

Circuit analysis equation containing a voltage source between node 1 and node 2

The principle is extended if more than one voltage source is connected to a node. Each voltage source creates one matrix row that defines its potential difference, and if the supernode enveloping all the connected voltage sources does not include the reference node, then there is also a row that describes KCL for the supernode.

Write the output

The results of the simulation are reported by creating a file describing the voltage at each node and the current through each component at every instant in the simulation. Write the output in CSV format where the columns are the nodes and components and the rows are the instances in time. Use the MATLAB script provided on blackboard to plot the results.

Add support for non-linear components (advanced)

When the circuit contains non-linear components, like diodes, there is no analytic solution. Instead a numerical solution is calculated using the iterative Newton Raphson method.

The component is converted to a linear approximation (a Thevenin equivalent) by guessing the potential difference and differentiating the I-V characteristic to obtain its gradient. Then, nodal analysis is performed as usual. The results of the nodal analysis are used to find a new potential difference, and from that a new linear approximation. The process is repeated until the node voltages stop changing between each iteration and converge on the solution.

Evaluation

Your solution should be evaluated against the following criteria:

1. Accuracy: compare the outputs to pen and paper solutions. For simple circuits you should be able to calculate an exact solution as a reference. You can also compare with LTspice.
2. Efficiency: find how long the simulation takes and, by estimating the power consumption of your computer, the amount of energy needed. How does it scale with the number of nodes in the circuit? Are there any implementation choices that affect the efficiency?