# IMPLEMENTATION AN ASIC FOR AN EXTREME LEARNING MACHINE

A DISSERTATION SUBMITTED IN PARTIAL FULFILMENT
FOR THE COURSE OF

***DIGITAL VLSI DESIGN***

***In***

***Master of Technology***

IN THE
FACULTY OF ENGINEERING

BY

GANESH KUMAR SHAW

GUIDED BY

PROF. CHETAN SINGH THAKUR


DEPARTMENT OF ELECTRONIC SYSTEMS ENGINEERING
INDIAN INSTITUTE OF SCIENCE, BANGALORE

OCTOBER 2021

# Notations

n or P: number of neurons which is same as Perceptron(P).

$W_{10}$ : Weight of connection between input and hidden neurons.

$W_{21}$ : Weight of connection between hidden neurons and output. radd : read address(among 10 registers in the module2)

P_index : $10 \times n^{th}$

# Contents

# List of Figures

# Chapter 1

# Pre-study

This project is about design and implement an ASIC for an Extreme Learning Machine(ELM) inferrence engine. Which can classify the handwritten digits 0 - 9.

## 1.1 Background

Extreme learning machine (ELM) is a training algorithm for *single hidden layer feedforward neural network (SLFN)*, which converges much faster than traditionl methods and yield promising performance. Becasue of its outstanding performance, ELM has been successfully applied in many real-time learning tasks for classification, clustering and regression.

Neural network, which usually refers to artificial neural network, was firstly proposed in last century for a history of more than 70 years. Neural network was inspired by biological neural network and its basic structure mimics the nerve system in human brain. As a kind of machine learning method, a typical neural network consists of neurons, connections, and weights. In neural network, a neuron, also called perception , receives input signals from former neurons and sends out output signals to later neurons. Neurons within a neural network are often organized in different layers.The connection was defined as relation between neurons of different layers. A neuron may have lots of connections to neurons of its former layer and later layer. And each connection delivers an important parameter called *weight*. *It is weights that decide how a neuron processes input signals to give out output signals*. Extreme learning machine (ELM) was proposed by Guang-Bin and Qin-Yu, which was aim to train single-hidden layer feedforward networks (SLFNs).ELM assigns random values to the weights between input and hidden layer and the biases in the hidden layer, and these parameters are frozen during training.

The nonlinear activation functions in hidden layer provide nonlinearity for the system. Then, it can be regarded as a linear system. The only parameter needs to learn is the weight between hidden layer and output layer. ELM is widely applied in a variety of learning problems, such as classification, regression, clustering, and feature mapping. ELM evolved as many variants have been proposed to further improve its stability and generalization for specific applications.

## 1.2 Functional aspects

The following project is to meant for recognise the digit (0 - 9) which will be provide to the device in the form of binary formate by converting $16\times$ 16 image into binary values.

# Chapter 2

# Study

ELM was invented to train SLFNs, which is the most widely used artificial neural network structure. A conventional SLFN consists of three layers: input layer, hidden layer and output layer, shown in Fig 2.1

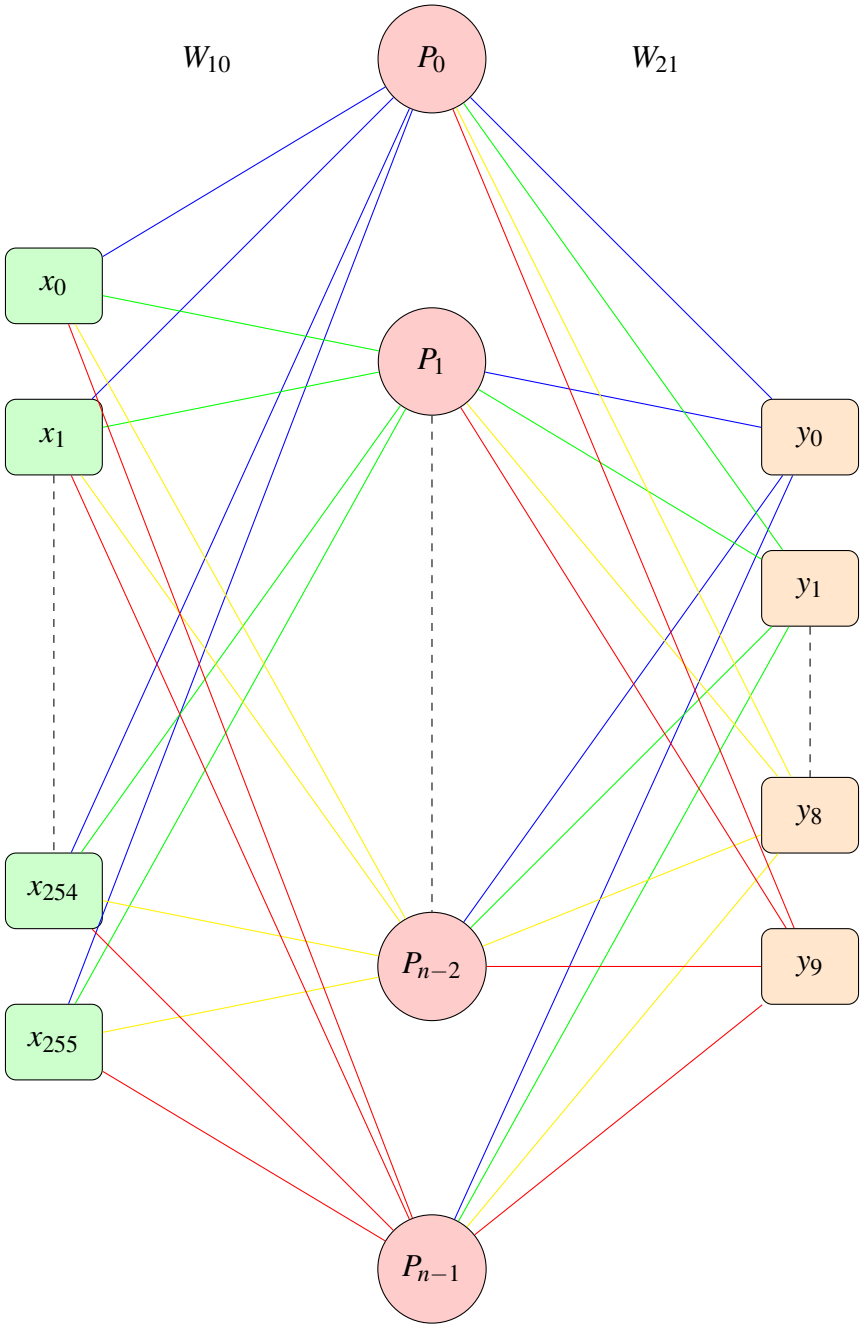


Figure 2.1: Structure of SLFN

## 2.1 SLFN Taining

Given a training set for single image ,

X= $\begin{bmatrix} x_0 & x_1 & ..... & x_{255} \end{bmatrix}$

Y= $\begin{bmatrix} y_0 & y_1 & ..... & y_9 \end{bmatrix}$

Random value of $W_{10}$ which will be generated by LFSR (Linear Feedback Shift Register)

$$W_{10} = \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots\cdots & w_{0,n-1} \\ w_{1,0} & w_{1,1} & \cdots\cdots & w_{1,n-1} \\ & & & \\ & & & \\ w_{255,0} & w_{255,1} & \cdots\cdots & w_{255,n-1} \end{bmatrix}$$

### 2.1.1 Calculation of Perceptron

$$\text{H=} \begin{bmatrix} P_0 & P_1 & \cdots\cdots & P_{n-1} \end{bmatrix} \text{=R(} \begin{bmatrix} x_0 & x_1 & \cdots\cdots & x_{255} \end{bmatrix} \times \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots\cdots & w_{0,n-1} \\ w_{1,0} & w_{1,1} & \cdots\cdots & w_{1,n-1} \\ & & & \\ & & & \\ w_{255,0} & w_{255,1} & \cdots\cdots & w_{255,n-1} \end{bmatrix} \text{)}$$

Where R is the ReLu activation function R= $\begin{Bmatrix} N & if N \geq 0 \\ 0 & else \end{Bmatrix}$

### 2.1.2 Calculation of $W_{21}$ using Perceptron

The values of $W_{21}$

$$W_{21} = \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots\cdots & w_{0,9} \\ w_{1,0} & w_{1,1} & \cdots\cdots & w_{1,9} \\ & & & \\ & & & \\ w_{n-1,0} & w_{n-1,1} & \cdots\cdots & w_{n-1,9} \end{bmatrix}$$

Therefore we can write,

$H \times W_{21} = Y$

Taking Moore-Penrose Pseudoinverse on both side of H

$\Rightarrow W_{21} = pinv(H) * Y$

Now we can apply test input value since we can generate $W_{10}$ using LFSR and we already have $W_{21}$. Using this value we can get output i,e **Y=**$P \times W_{21}$

Since we have calculated $W_{21}$ weights using 1 training set(i,e one image) hence the accuracy would be very very low. So to get good accuracy we have to use as much as images possible as a training set ( it must consists of 0-9 digits). Here I shall use 1110 images as training.

Then H becomes

$$\text{H=}\begin{bmatrix} Forimage0: & P_0 & P_1 & ..... & P_{n-1} \\ Forimage1 & P_0 & P_1 & ..... & P_{n-1} \\ & & & & \\ & & & & \\ For_image_1109: & P_0 & P_1 & ..... & P_{n-1} \end{bmatrix}$$

Similarly

$$\text{Y=}\begin{bmatrix} Forimage0: & y_0 & y_1 & ..... & y_9 \\ Forimage1: & y_0 & y_1 & ..... & y_9 \\ & & & & \\ & & & & \\ Forimage1109: & y_0 & y_1 & ..... & y_9 \end{bmatrix}$$

*But the procedure of calculating the weights of $W_{21}$ will be same i,e pinv(H)*Y*(Here Y representing the training sets output) and the following procedure of calculating $W_{21}$ weights can be done in tool(matlab,etc...).

## 2.2 LFSR design in matlab for $W_{10}$ weights

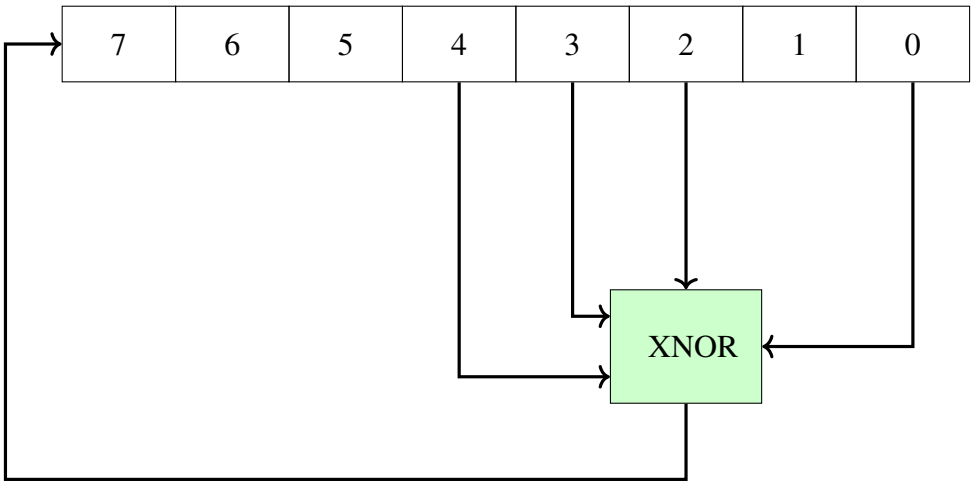


Figure 2.2: LFSR BLOCK DIAGRAM

A simple 8-bit linear feedback shift register built from D-flipflops. In the fig 2.2, the outputs of flipflops 4,3,2,0 are summed via XNOR gates (this is a linear operation, hence the name) and fed back into the first flipflop.

Depending on the logic used in the feedback path, the register follows a predefined sequence

of states, with a maximum sequence length of $(2^n) - 1$ in a n-bit register. Perhaps the most important use of LFSRs is as pseudorandom number generators, especially for automatic self-test, because the generators are very cheap and can be run at very high clock frequencies. With a register of n+1 bits, each n-bit input value can be generated. Naturally, subsequent output values are highly correlated, as only the first bit changes while all other bits are simply shifted.

# Chapter 3

# Design

**Module Designs**

## 3.1 Module1 for Perceptron

In this section I will implement the Perceptron generator circuit.

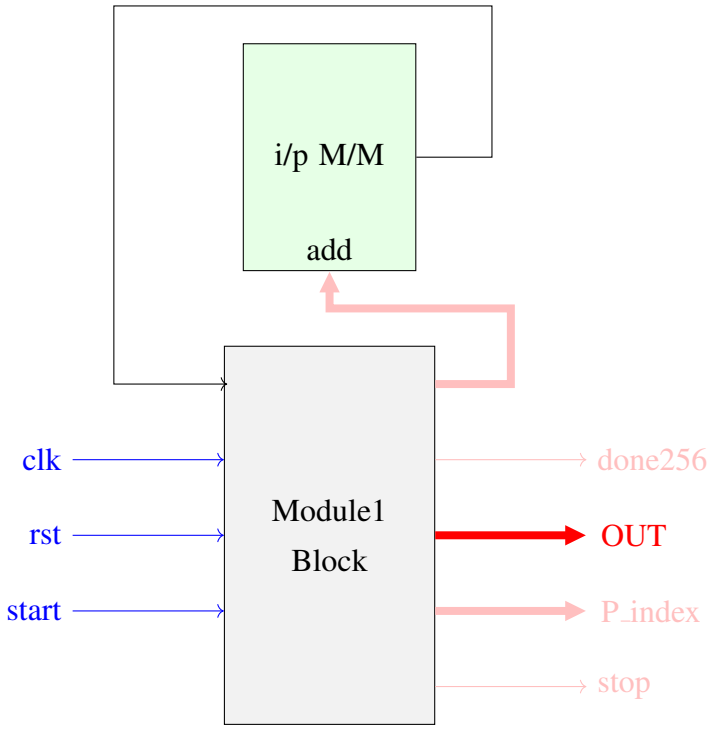


Figure 3.1: Module1 Block Diagram

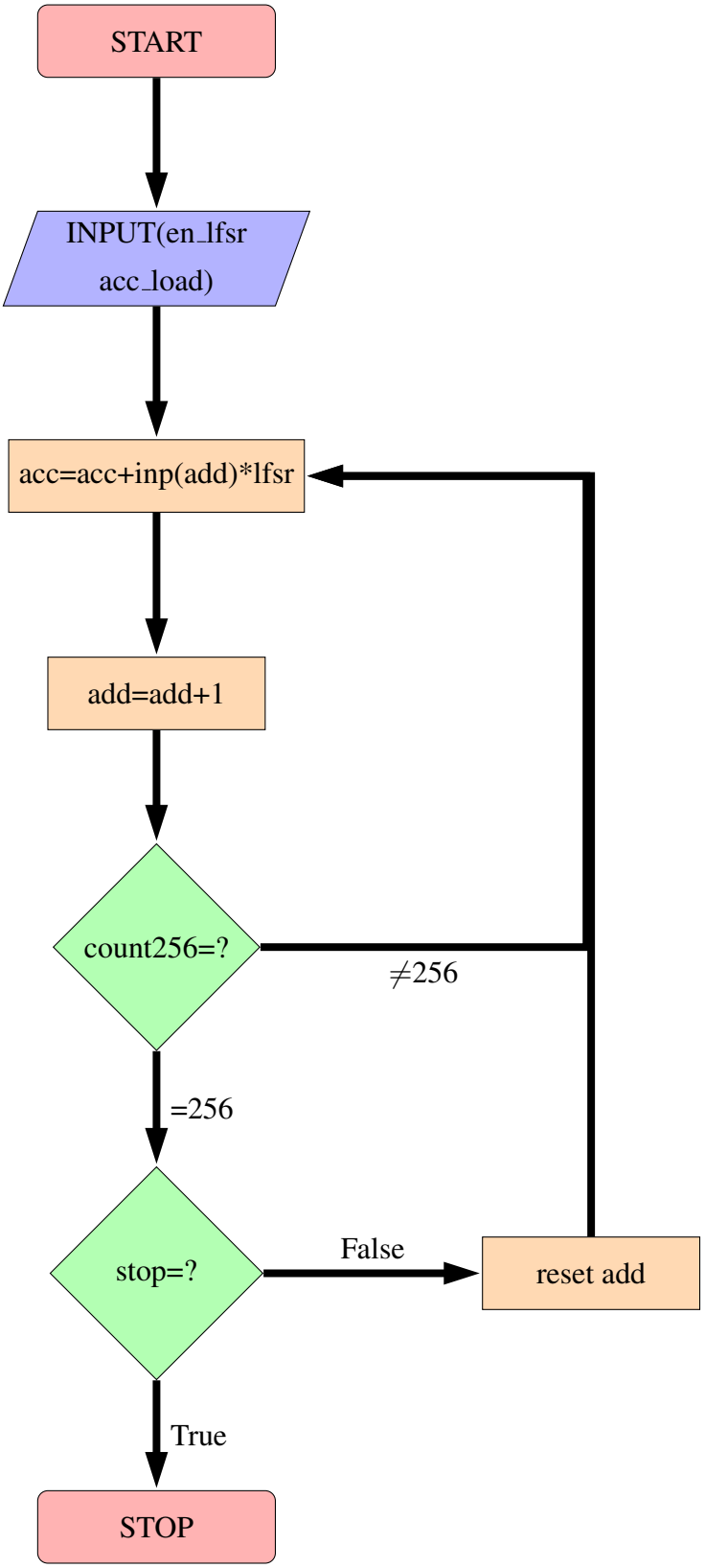


Figure 3.2: Module1 Flow Chart

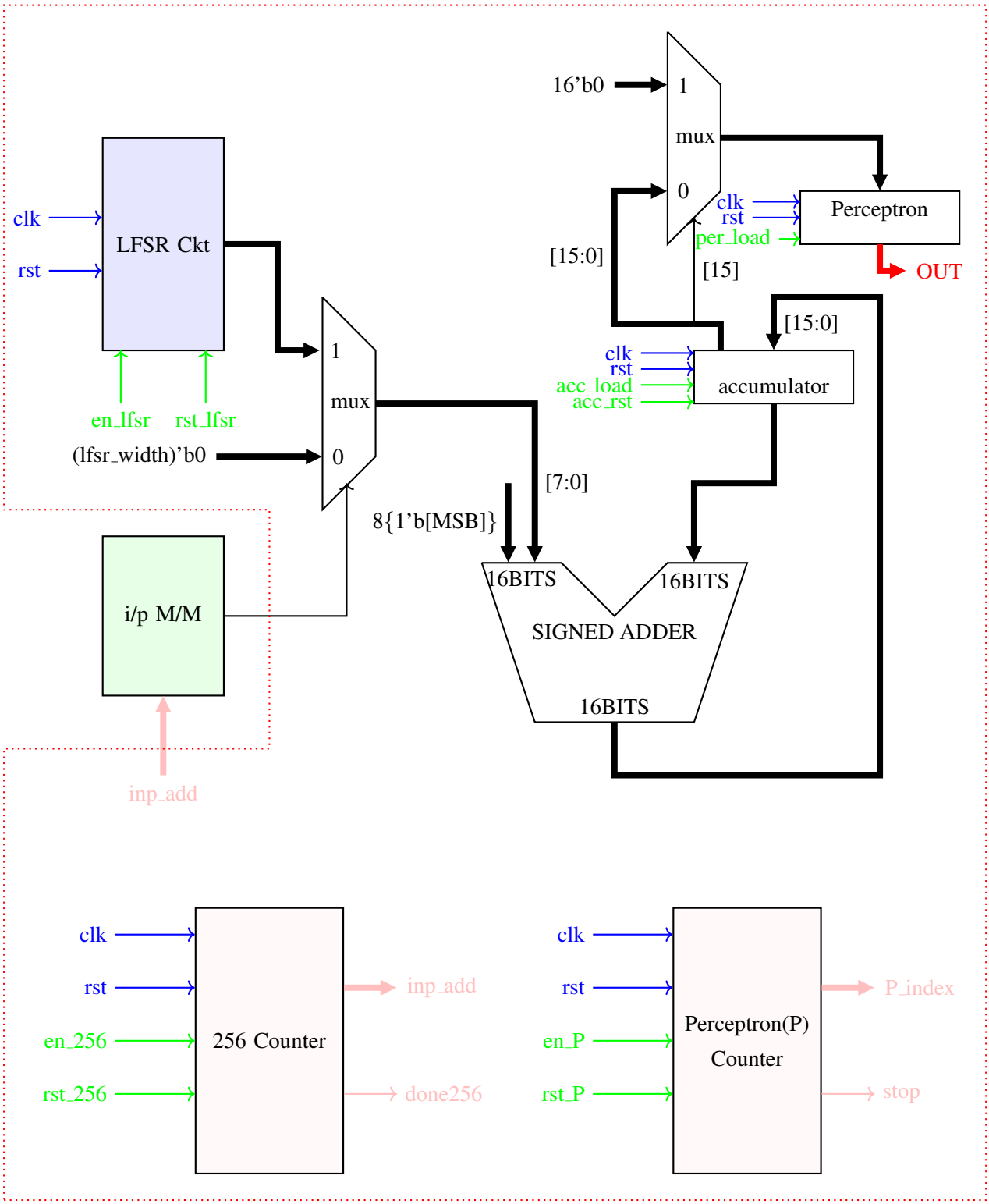


Figure 3.3: Module1 Data Path

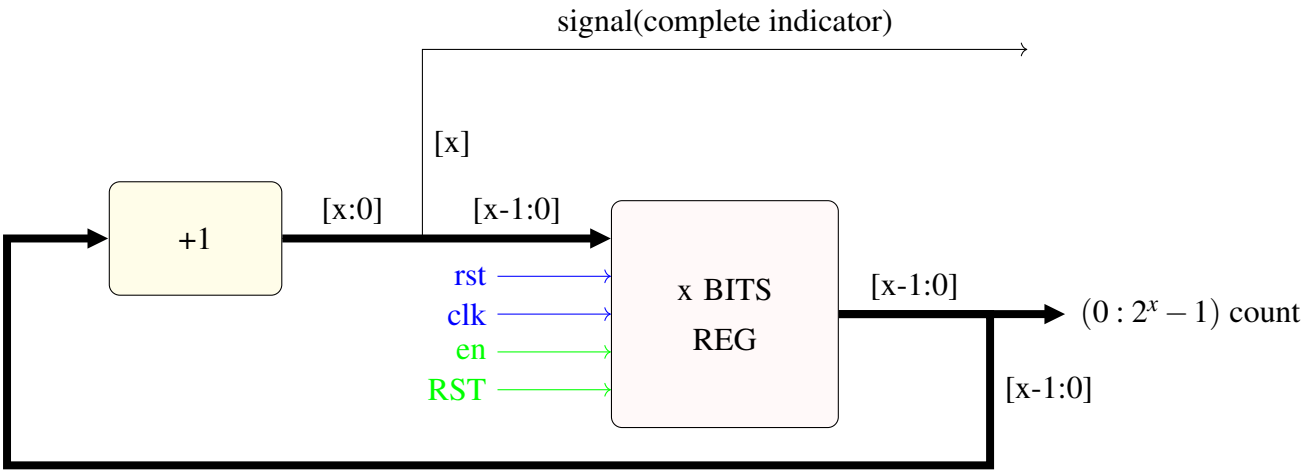


Figure 3.4: x BITS COUNTER

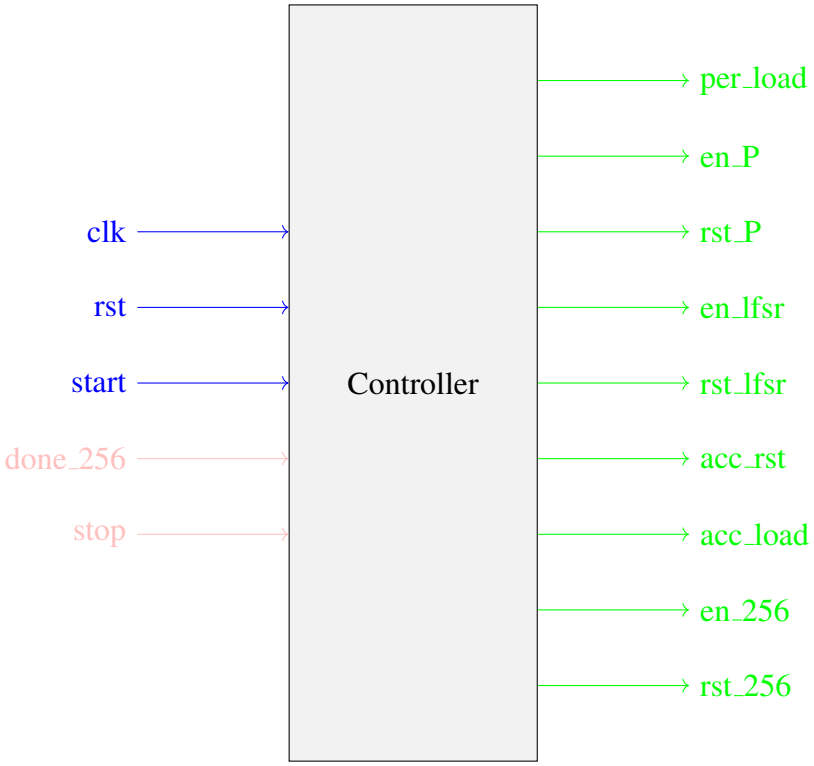


Figure 3.5: Module1 Controller

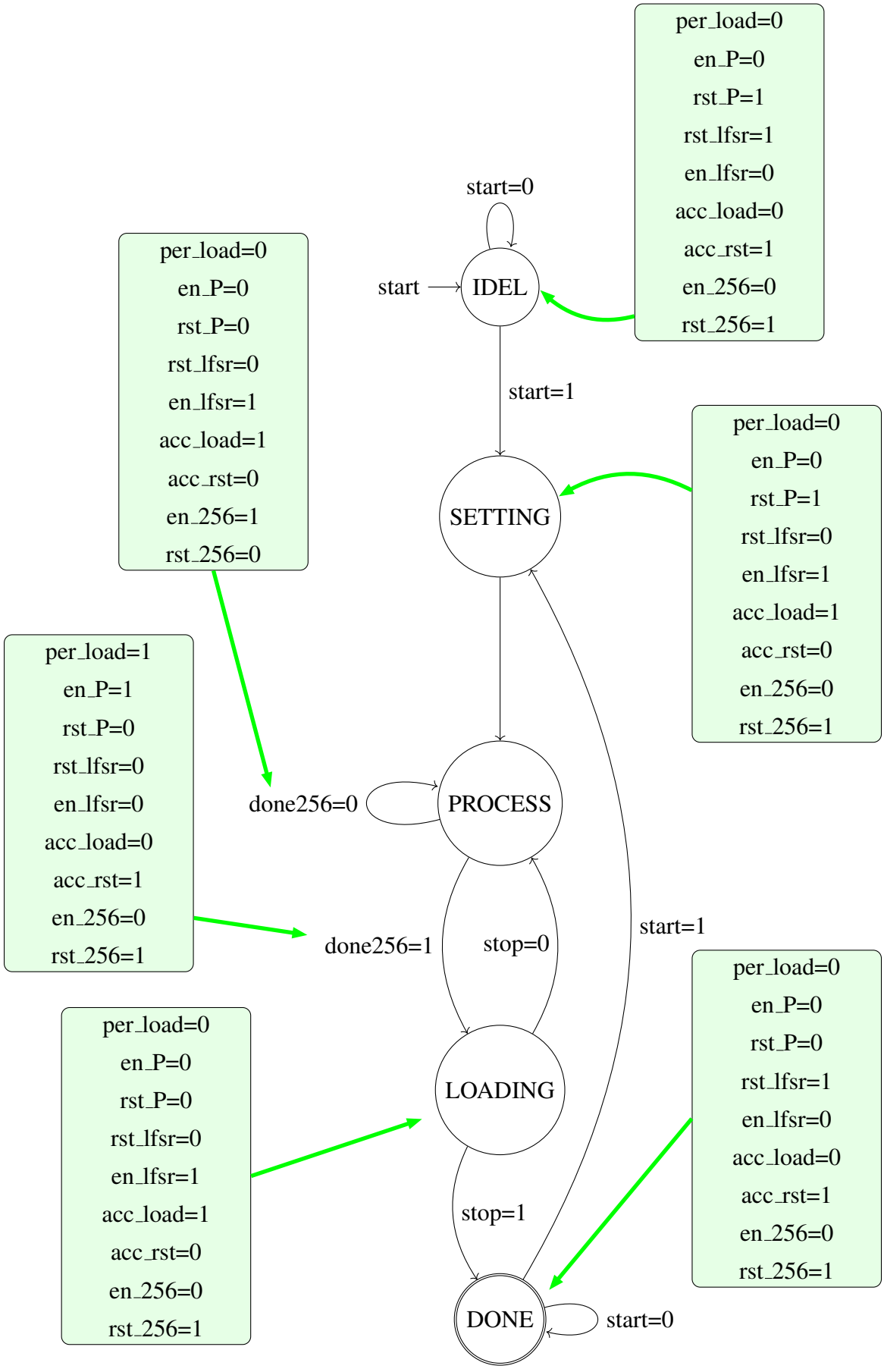


Figure 3.6: Module1 State Diagram

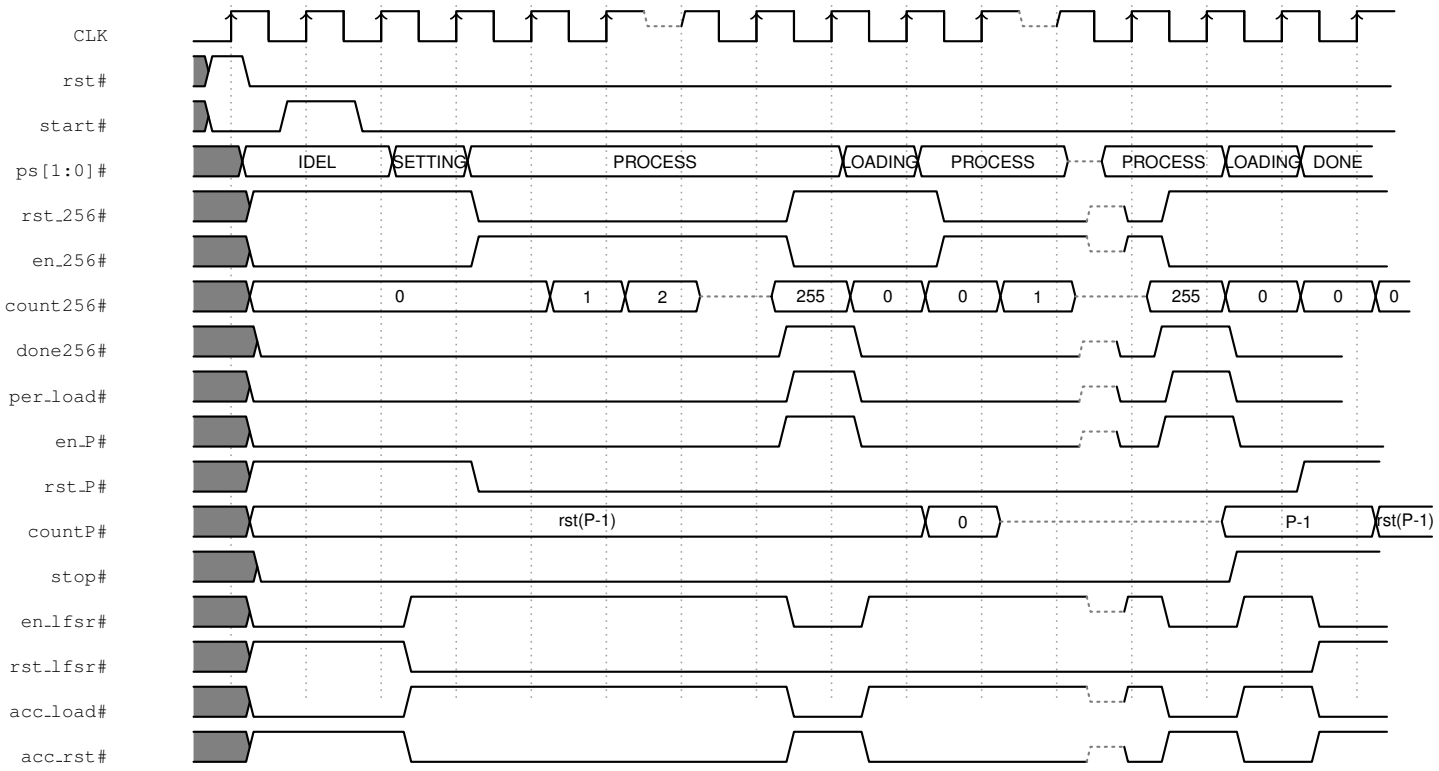


Figure 3.7: Module1 Timing Diagram

## 3.2 Module2 for Output Reg Files

In this module we are going to design circuit for Output Reg Files (equal to number of output nodes). In the $3^{rd}$ module we will use this reg file to evaluate output. 'radd' input, generated by the external controller in the module3, will be used after all the writing operation over in the reg files before that 'radd' will be generated by internal controller in the module2.

'done256' input signal will work as an initializer of the module2 FSM and 'start' signal will reset the the module2 FSM.

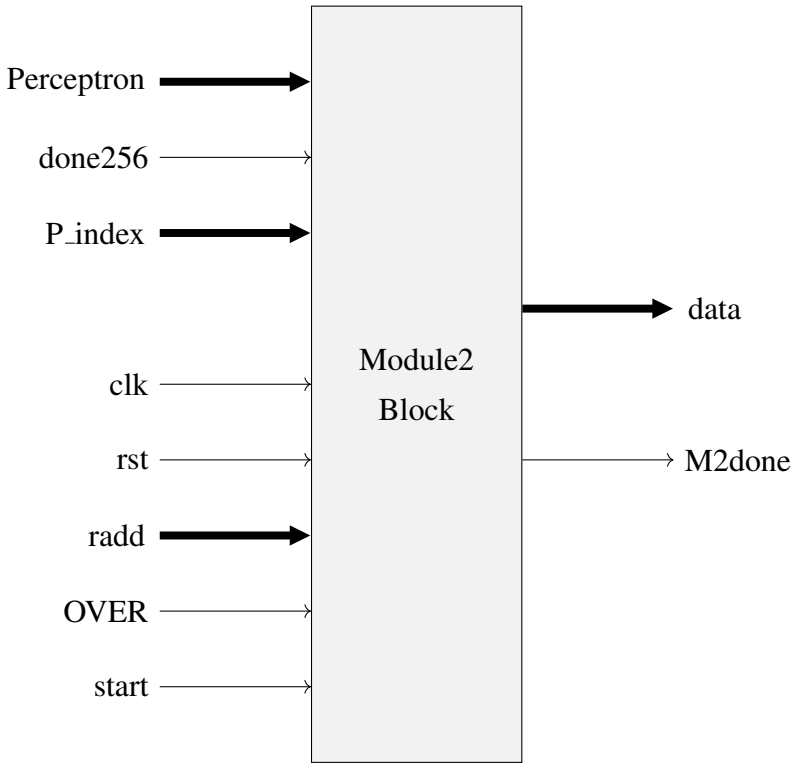


Figure 3.8: Module Block Diagram

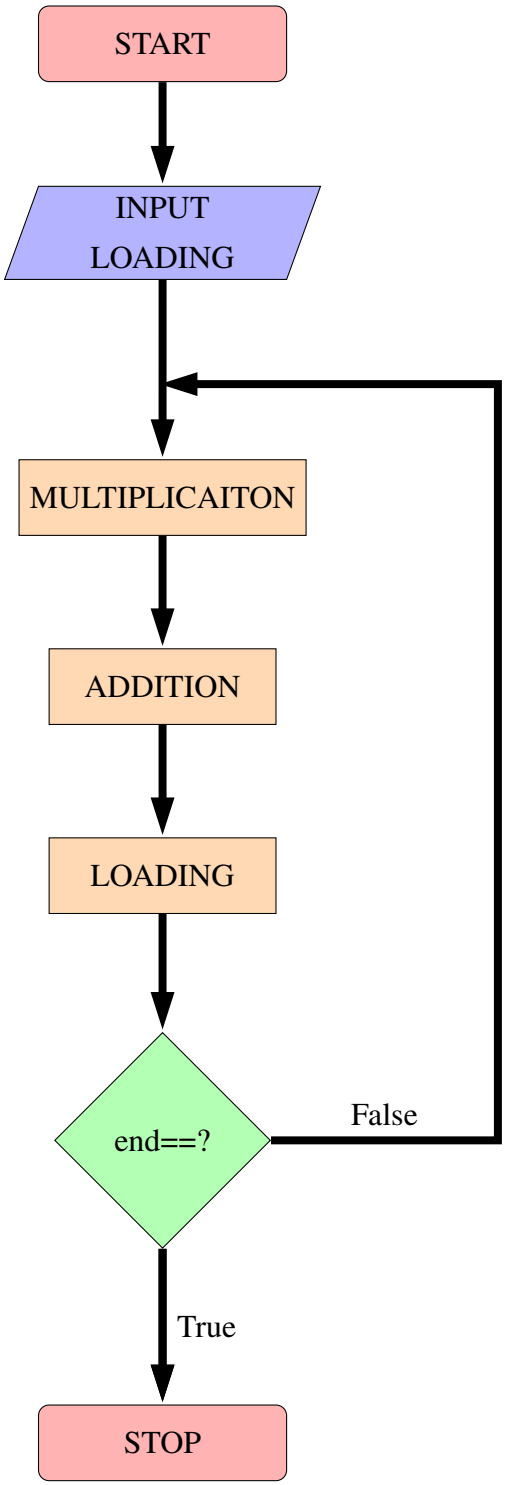


Figure 3.9: Module2 Flow Chart

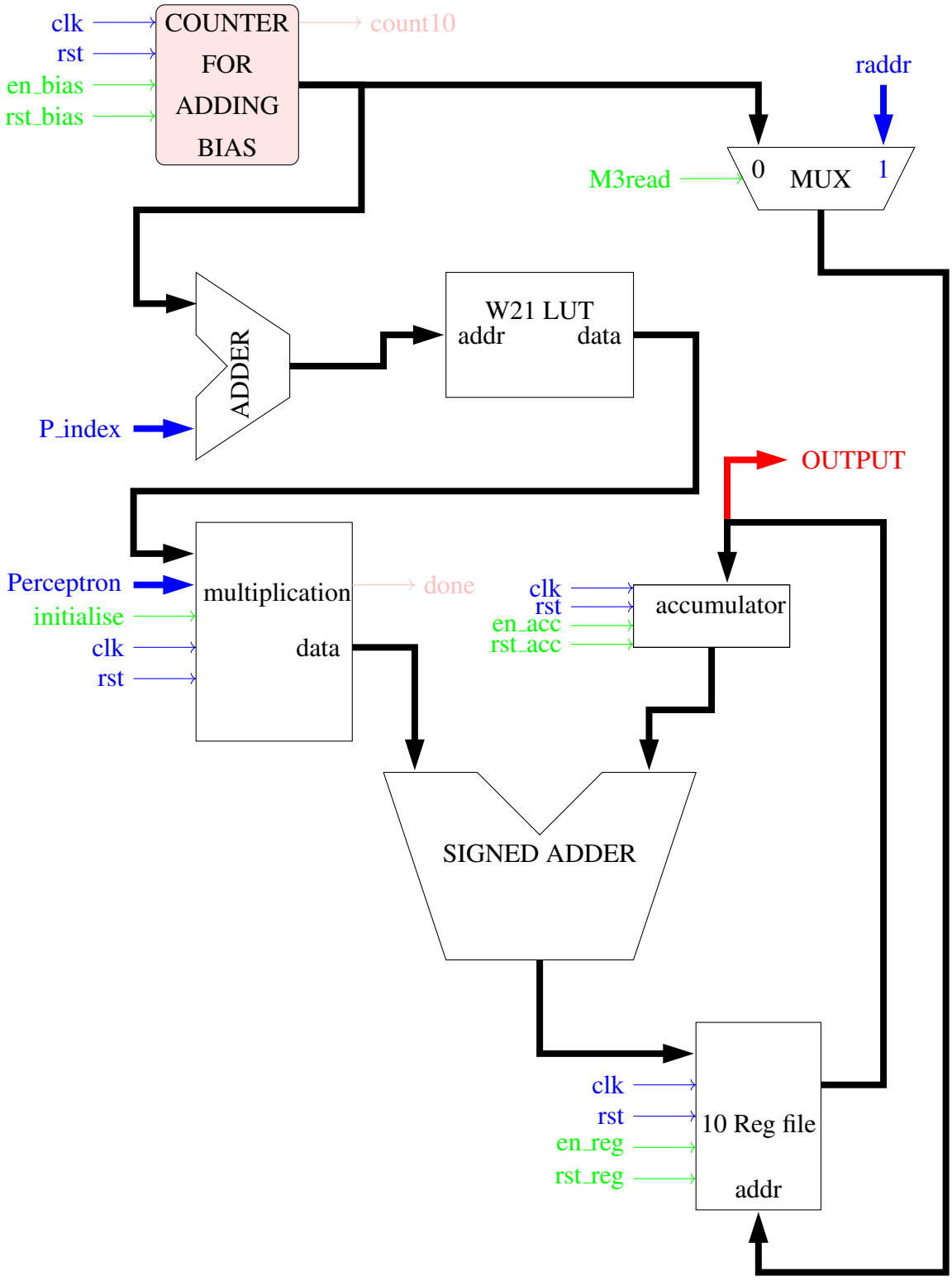


Figure 3.10: Module2 Data Path

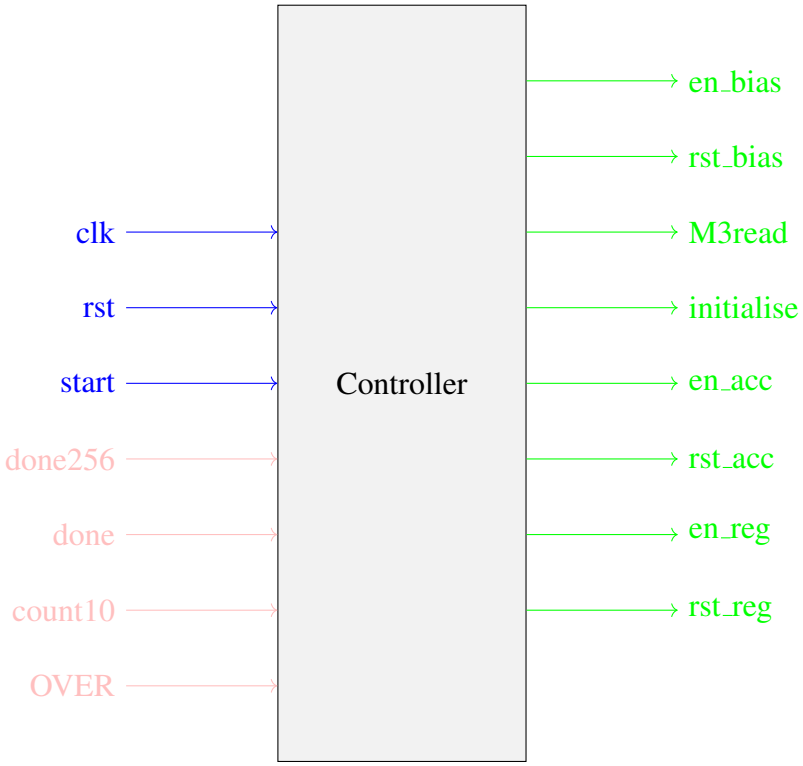


Figure 3.11: Module2 Controller

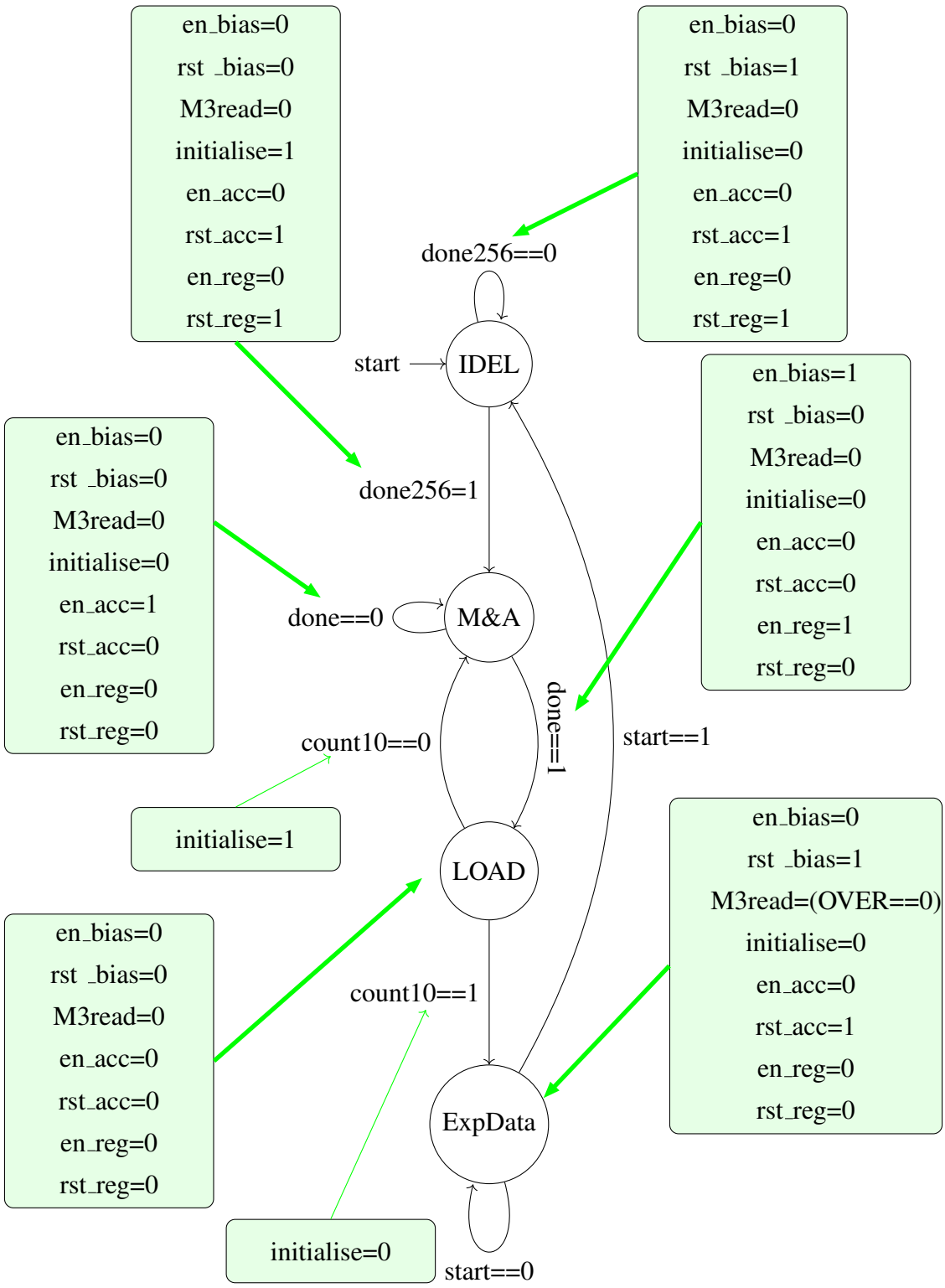


Figure 3.12: Module3 State Diagram

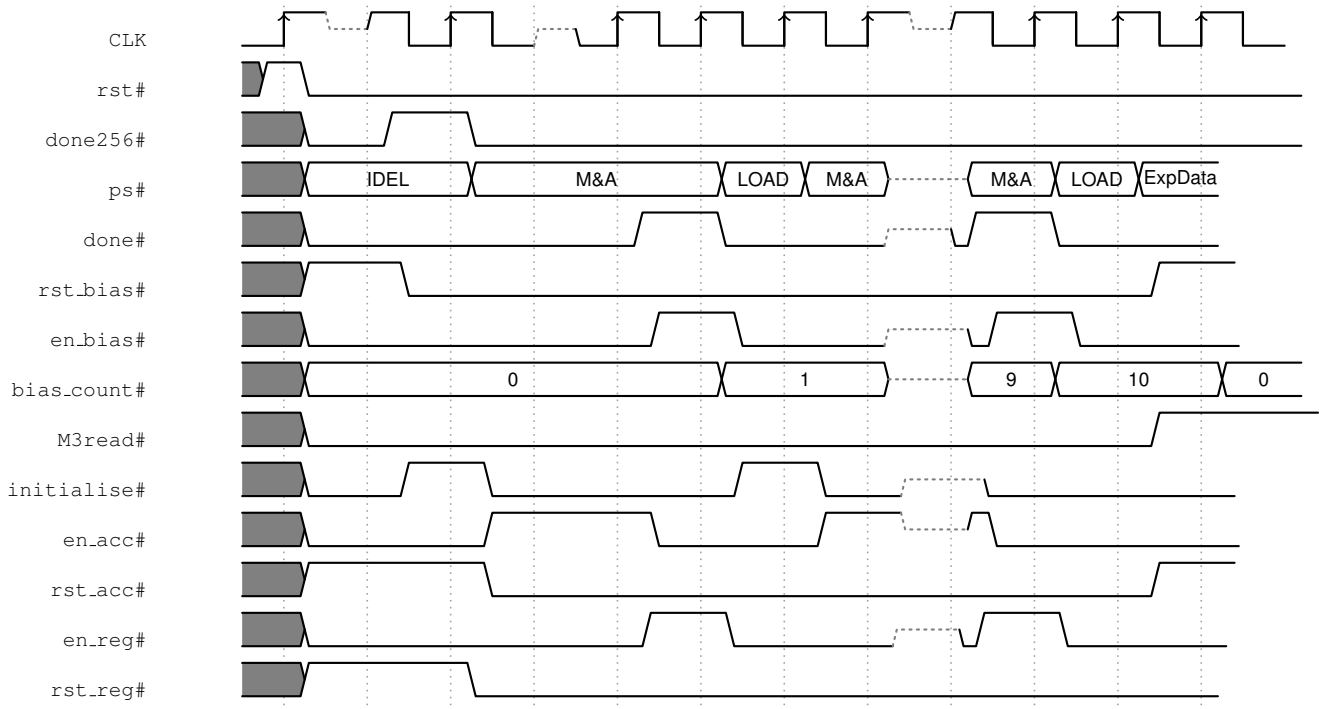


Figure 3.13: Module2 Timing Diagram

## 3.3 Module3 for Output

In this section we will compare the regfile values and based on that appropriate signal(the index with highest value signal will be set). The output 'raddr' is for fetching data from regfiles shown

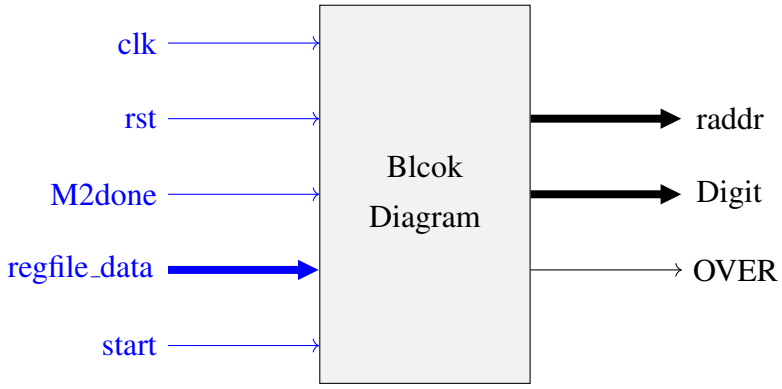


Figure 3.14: Module3 Block Diagram

in module 2 in the fig 3.10 by providing read address. Digit is the 10 bit final output which will indicate the input digit value by set the corresponding index value.

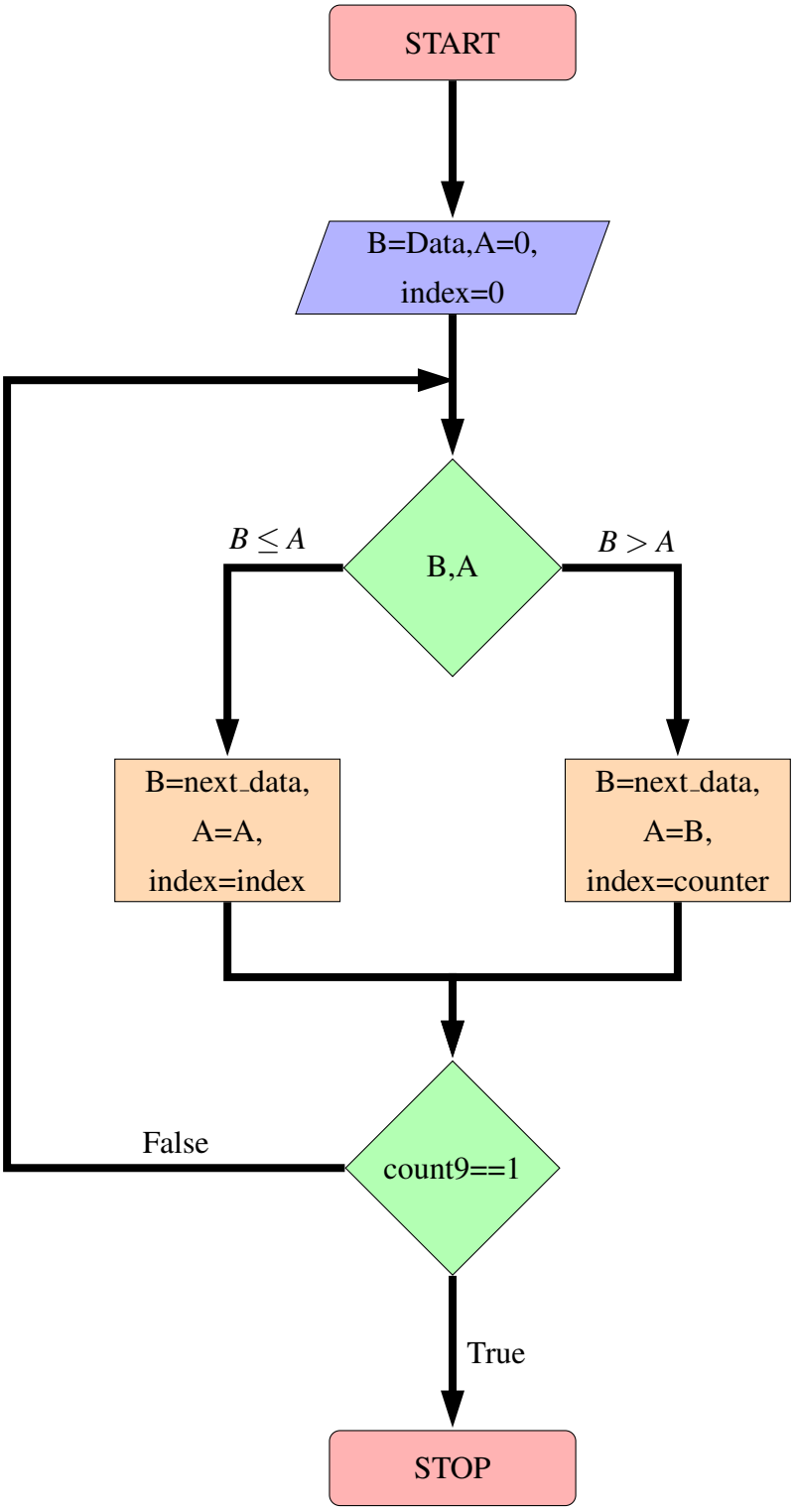


Figure 3.15: Module3 Flow Chart

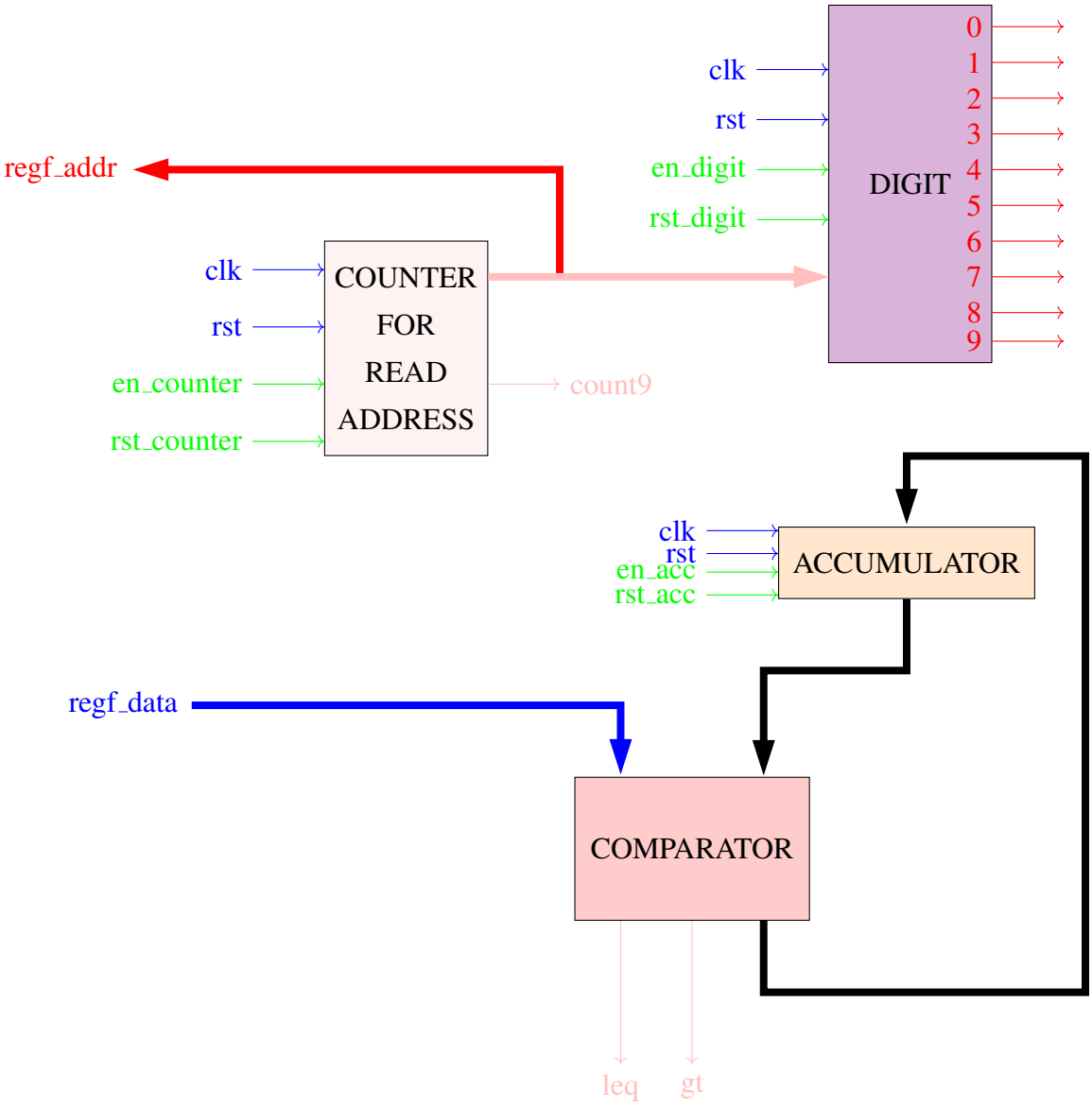


Figure 3.16: Module3 Data Path

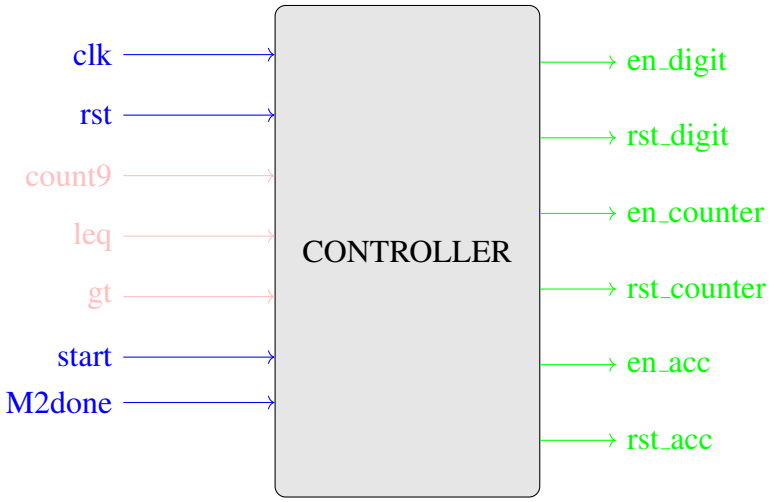


Figure 3.17: Module3 Controller

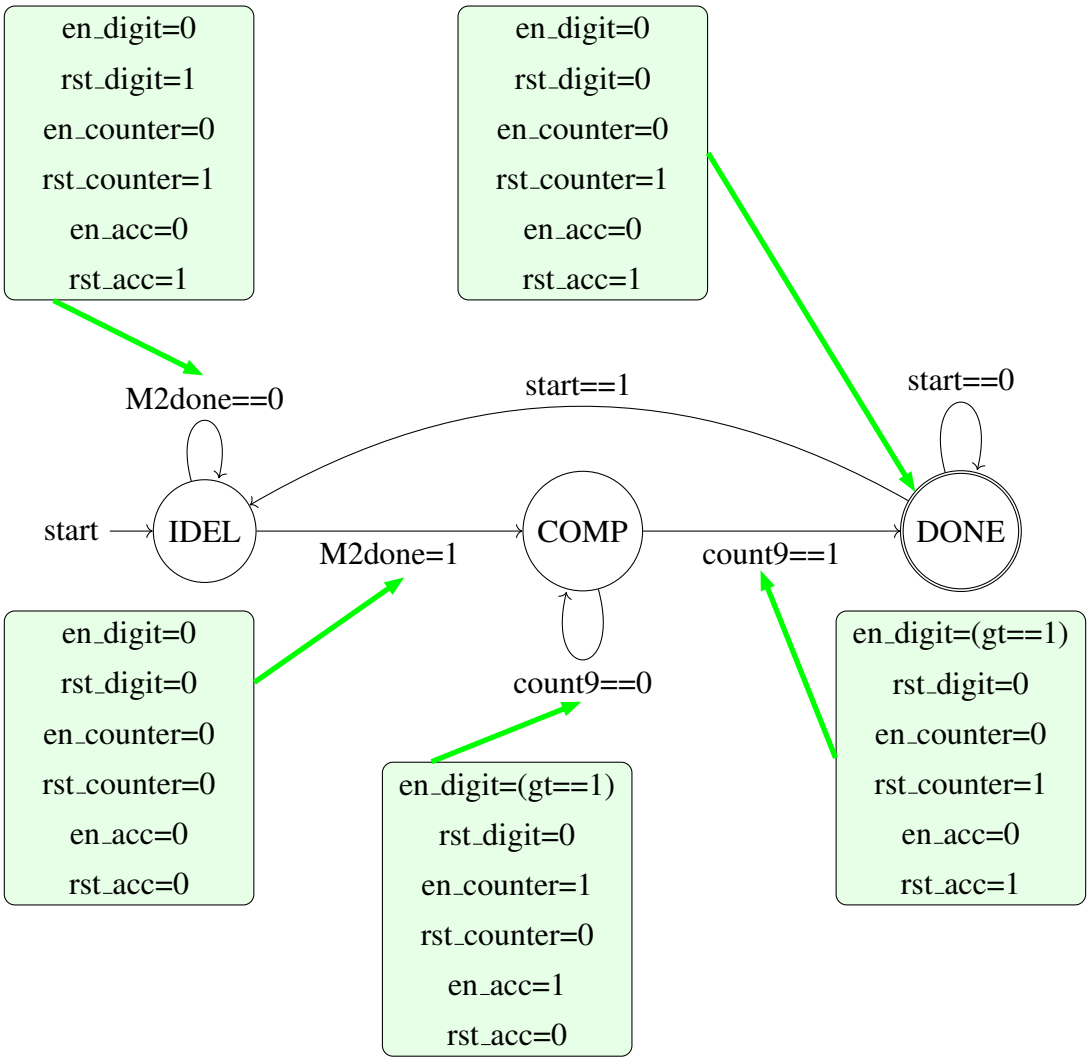


Figure 3.18: Module3 State Diagram

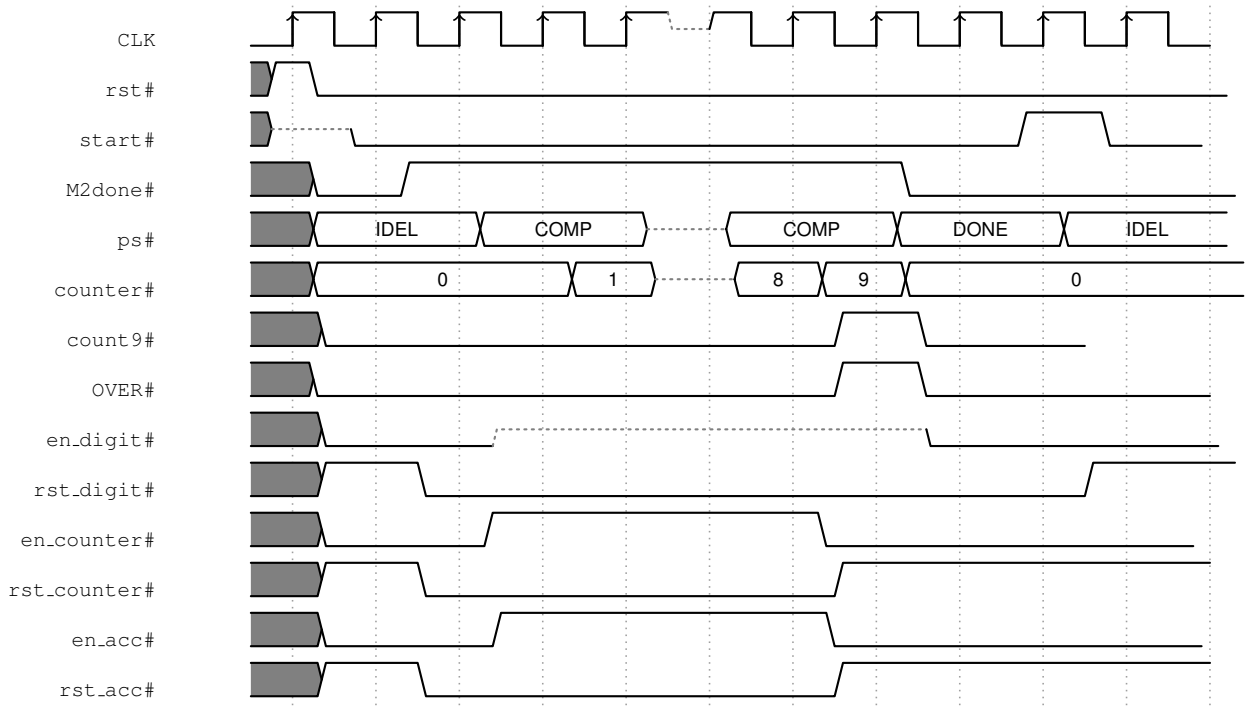


Figure 3.19: Module3 Timing Diagram

## 3.4 Module Assemble

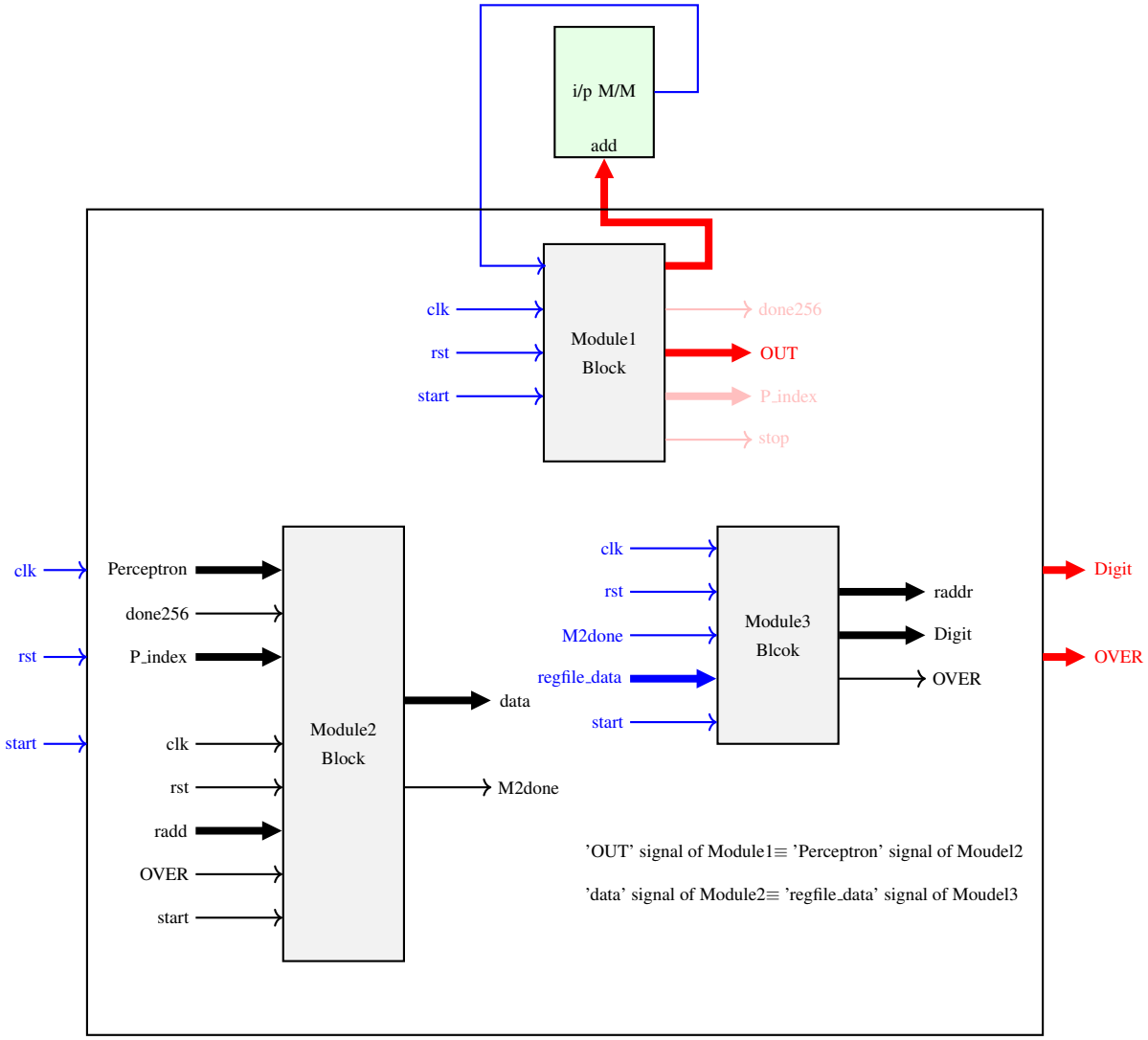


Figure 3.20: TOP MODULE

# Bibliography