Centro de Investigación y de Estudios Avanzados

TAE 2023 – ASIC DESIGN

Convolution Project Report

SoC Design Methodology

Teacher: Rodrigo Jaramillo Ramírez

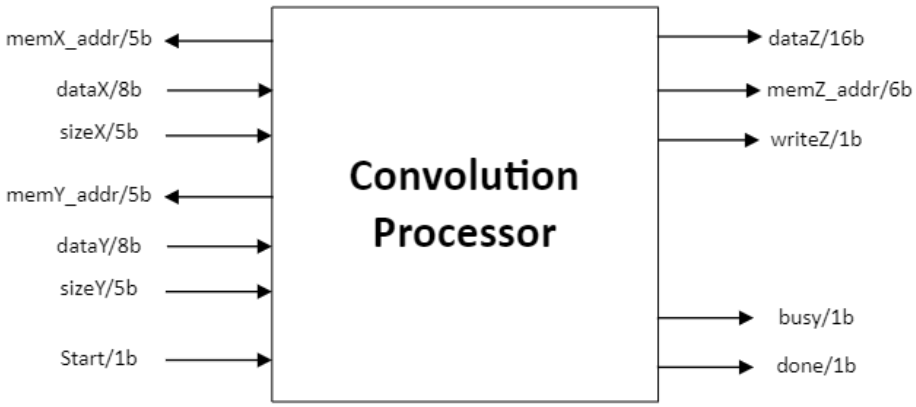
Student: Juan Antonio Serrano Gómez

Guadalajara, Jalisco, 30/04/2023

PROBLEM DESCRIPTION

We want to implement a convolution coprocessor from scratch, with two using a top-down aproach, following the steps seen in our class. This convolution processor will follow the below description, so we can test it in a simulation TB with RAM and ROM memory designs, simulate it's waveform and analyze it´s performance.

BLACK BOX



This top level block will require the following I/O ports:

Inputs:

- dataX & dataY = Data stored in memory X and memory Y.
- sizeX & sizeY = Size of the signals stored in each memory.
- memX_addr & memY_addr = Addresses from where to read in memories.
- Start = Start signal.

Outputs:

- Done = One-shot signal to indicate that the operation is complete.
- Busy = Signal indicating the processor is busy.
- WriteZ = Signal to store result in memory Z.
- dataZ = Data to be stored in output memory.
- memZ_addr = Adress where ouput data will be stored.

PSEUDOCODE GENERATION

To develop a pseudocode for our convolution process, we use the formula for convolution reviewed in class, evaluate the specific necessities for its implementation in a HDL design, and créate a process that can get, from two input memories, a convolutional output for an output memory:

Convolution Formula:

$$z(n) = \sum_{-\infty}^{\infty} x\,[k]y[n-k]$$

Given that we wanto to do a sweep for n and k, we have to avoid the conditions where the adresses fall outside the sizes of our memory vectors. I found the optimal evaluation conditions to be: (n < Zsize) and (n >= k). Finally, we have to consider the I/O requirements. This results in the following pseudocode:

Pseudocode
1.- While Start = 0; 2.- End While; 3.- Busy = 1; 4.- Zsize = SizeX+SizeY-1; 5.- for n = 0 to (n < Zsize) 6.- for k = 0 to (n >= k) 7.- memX_addr = k; 8.- memY_addr = n-k; 9.- reg_Z = reg_Z+dataX+dataY; 10.- End For 11.- memZ_addr = n; 12.- dataZ = reg_Z; 13.- WriteZ=1 14.- WriteZ=0 15.- reg_Z = 0; 16.- End For 17.- busy = 0; 18.- done = 1; 19.- done = 0; 19.- If Start = 1 21.- goto 1 22.- Else goto 19

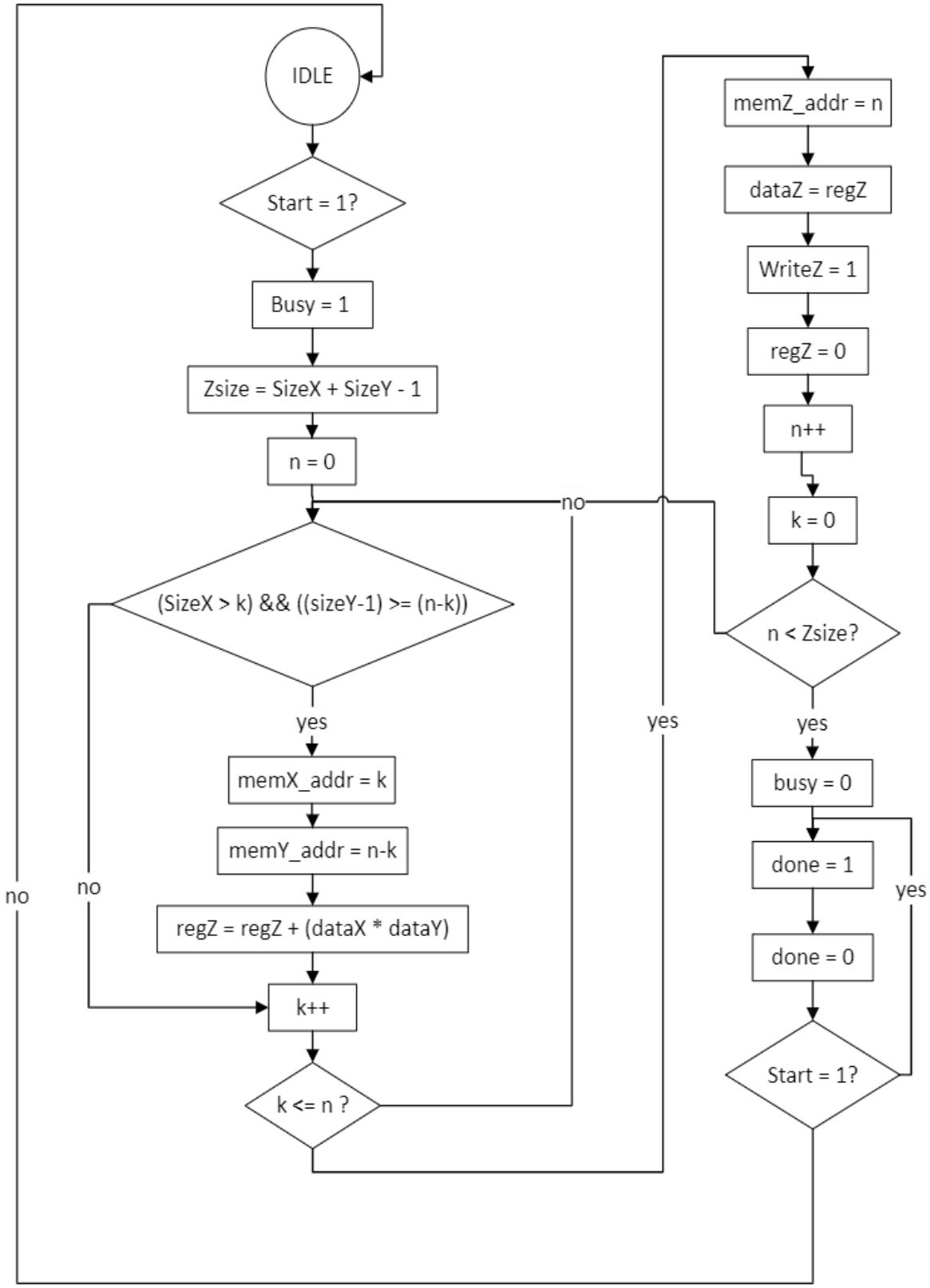
PSEUDOCODE VALIDATION

In order to validate the functionality of our pseudocode, we implent it in a C program, using 3 arrays as the memories (two input memories and one ouput memory). We also ad dan algorithm to calculate the input memory (array) sizes, so as to improve the Flow of the validation:

Validación del pseudocódigo en C

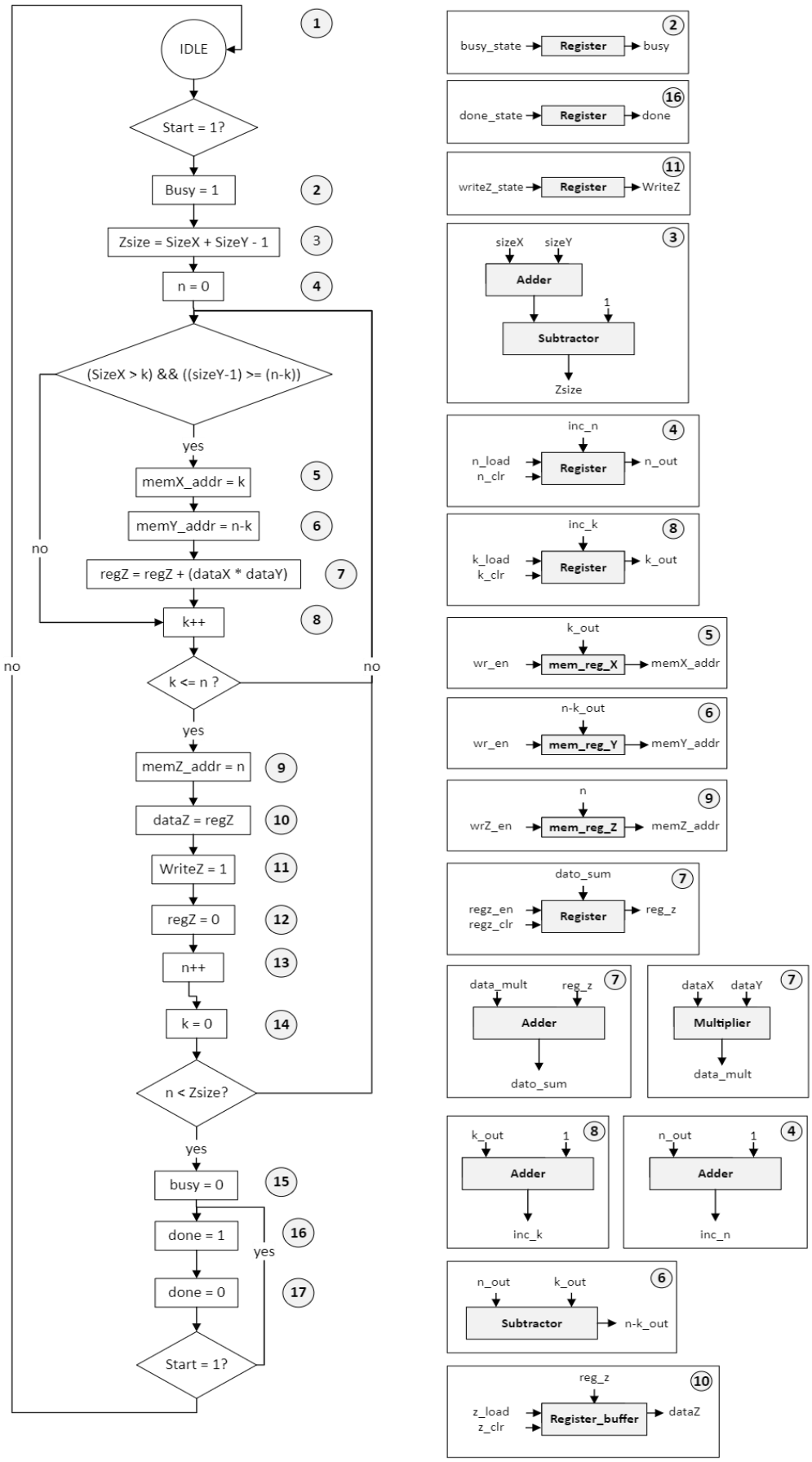
```
// Variable Declaration
int data_X, data_Y;
int size_X, size_Y;
int memY__addr, memX_addr, memZ_addr;
int data_Z;
int Z_size, n, k, i,;
int reg_Z = 0;

//Memory declaration
int memY[3] = {2, -1, 0};
int memX[10] = {-1, 0, 1, 4, 6, 7, -8, 4, 5, -2};
int memZ[64];

int main(void) {
//Wait for start input
   while(!start){
      printf("Waiting for Start 1:" );
      scanf("%c", &start);
  }
//Vectors X, Y & Z Size Calculation
   int size_X = sizeof(memX) / sizeof(memX[0]);
   int size_Y = sizeof(memY) / sizeof(memY[0]);

   Z_size = size_X + size_Y - 1;

   printf("Size Z: %d \n",  size_X);
// Convolutional Process
   for(n=0; n < Z_size; n++) {
      for(k=0; k <= n; k++){
         if((k < size_X) && ((size_Y-1) >= (n-k)))
         reg_Z = reg_Z + memX[k]*memY[n-k];
   }
      memZ[n] = reg_Z;
      reg_Z = 0;
   }
// Print Z Memory Locations
   for(i=0; i < Z_size; i++){
         printf("z[%i]: %i \n",  i, memZ[i]);
        sleep(1);
   }
 }
```

ASM DIAGRAM

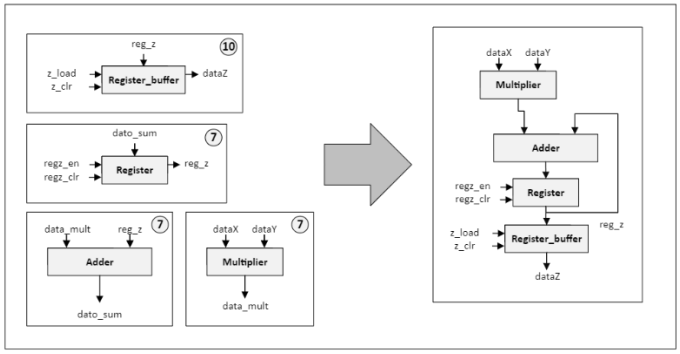


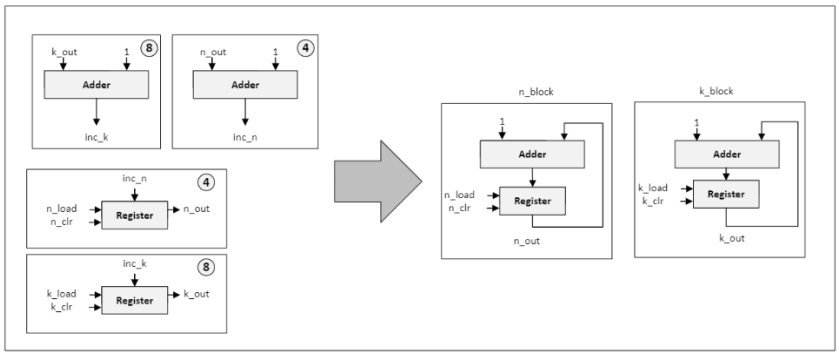
From this ASM diagram, we can start generating the “optimum” Building Blocks for the datapath:

BUILDING BLOCKS GENERATION

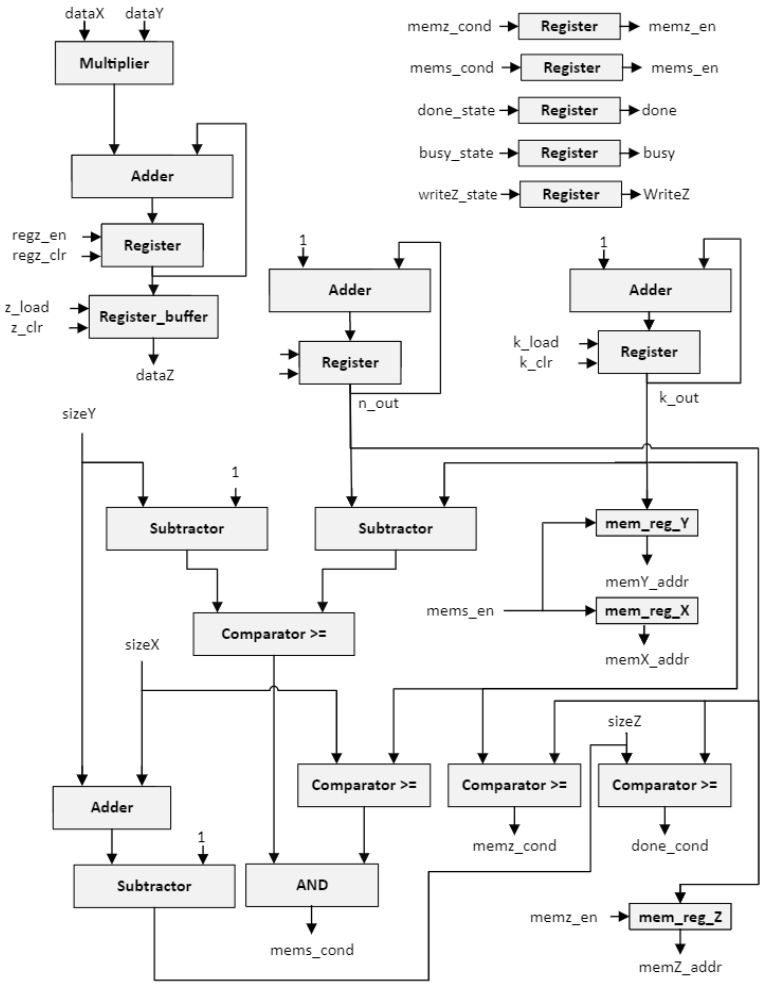


BUILDING BLOCKS OPTIMIZATION

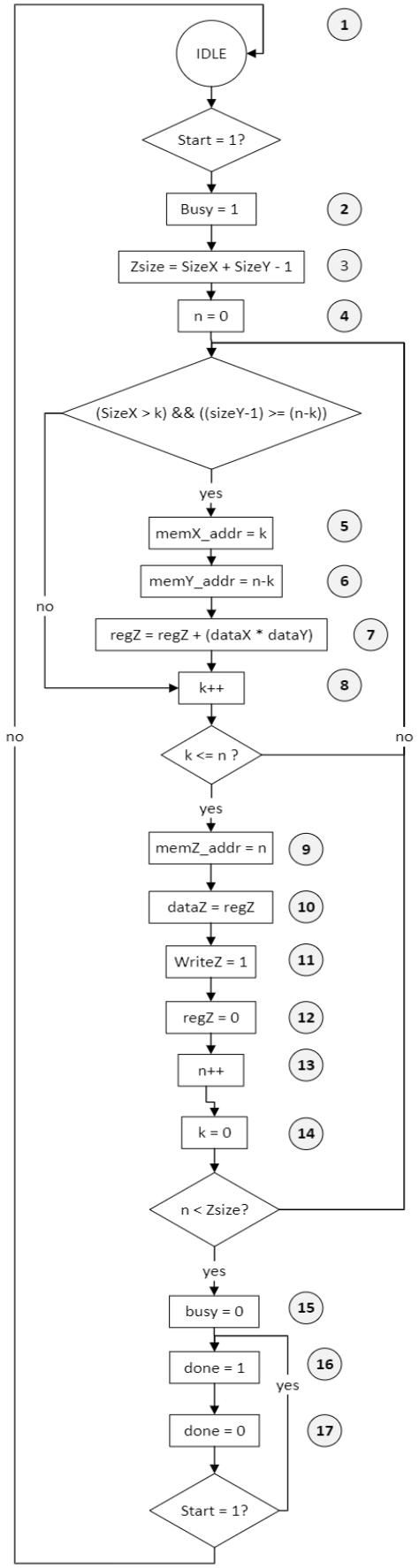




DATAPATH CONSTRUCTION

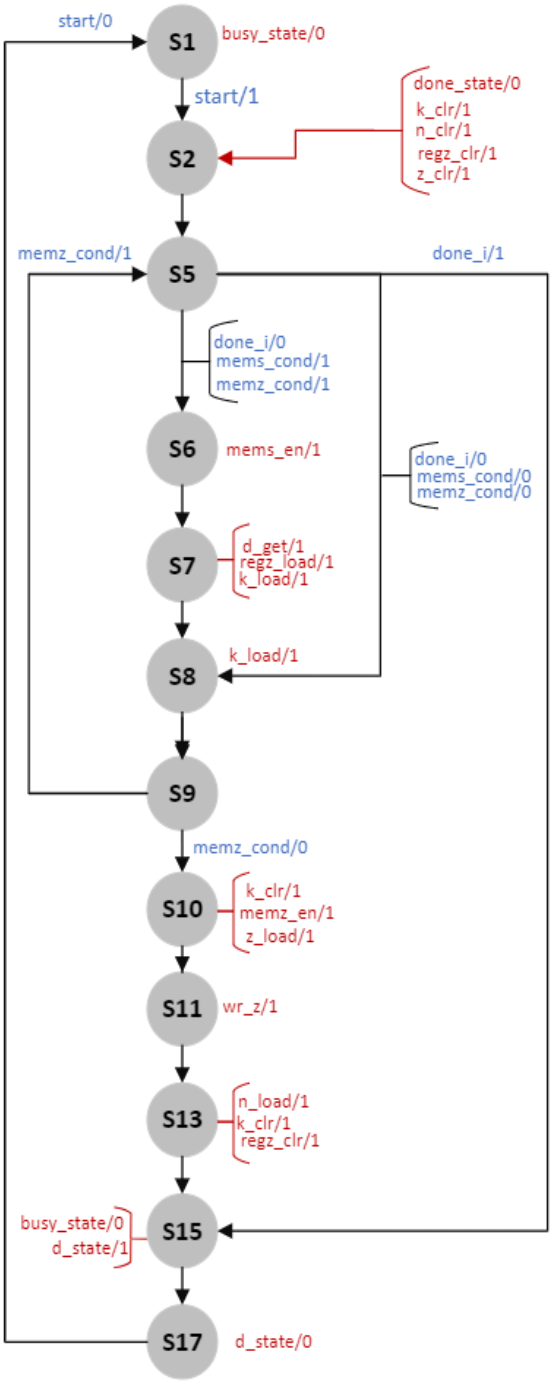


CONTROL SIGNALS FOR DATAPATH



Control FSM		
State	Input/Value	Output/Value
1	Start/0	
2	Start/1	busy_state/1
3		ini_State/1
4		n_clear/1
5	done_i/0 mems_cond/1 memz_cond/1	
6		mems_en/1
7		d_get/1 regz_load/1
8		k_load/1
9	memZ_cond	
10		memz_en/1
11		wr_z/1 z_load/1
12		regz_clr/1
13		n_load/1
14		k_clr/1
15		busy_state/0
16		done_state/1
17	Start/0	done_state/0

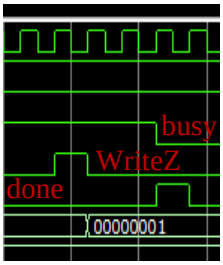
FSM: (inputs and outputs)



Control FSM		
State	Input/Value	Output/Value
1	Start/0	
2	Start/1	busy_state/1
3		ini_State/1
4		n_clear/1
5	done_i/0 mems_cond/1 memz_cond/1	
6		mems_en/1
7		d_get/1 regz_load/1
8		k_load/1
9	memZ_cond	
10		memz_en/1
11		wr_z/1 z_load/1
12		regz_clr/1
13		n_load/1
14		k_clr/1
15		busy_state/0
16		done_state/1
17	Start/0	done_state/0

DESIGN CORRECTIONS AND IMPROVEMENTS:

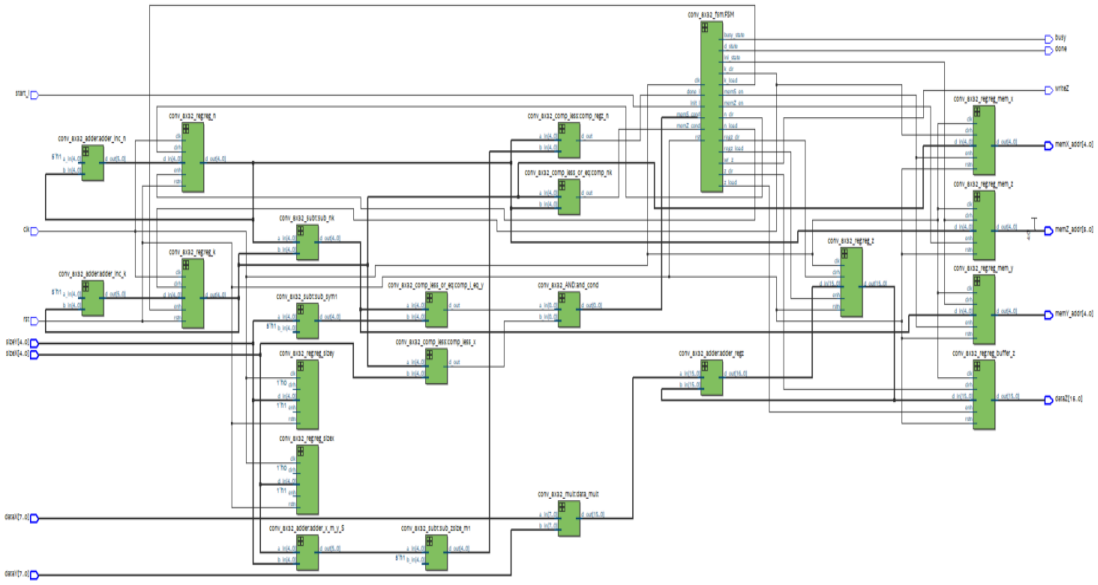
FSM outputs were connected directly to the top level ports.



Done state is set to just one clock cycle.

FPGA SIMULATION AND SYNTHESIS

TOP LEVEL SCHEMATIC:

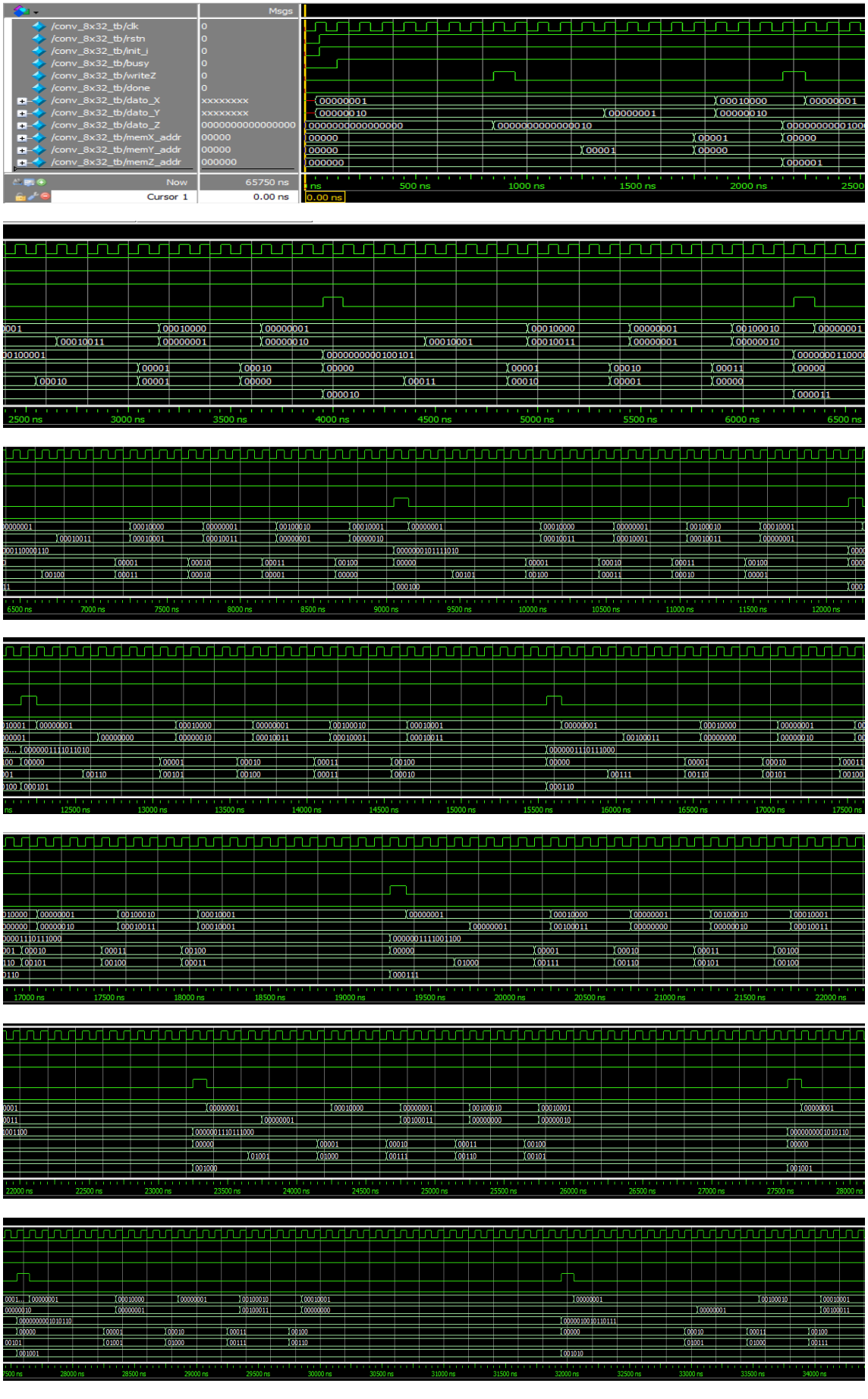


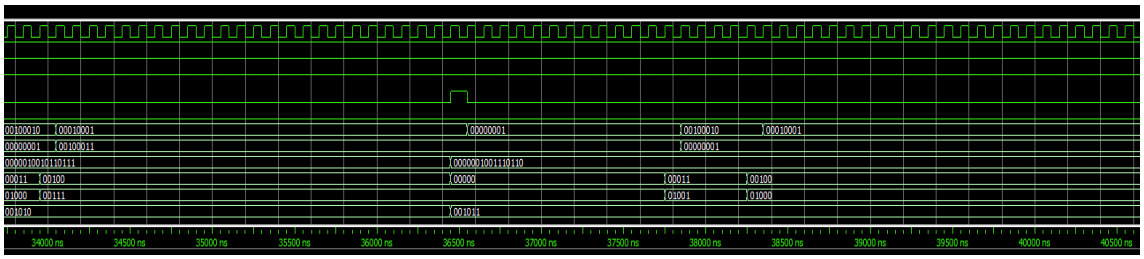
MEMORY FILE GENERATOR:

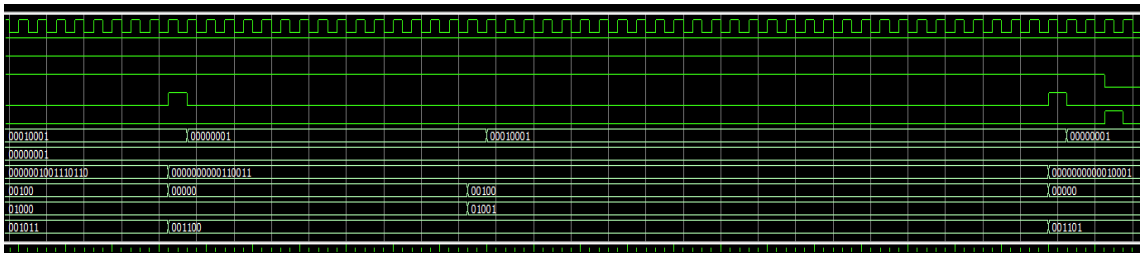
Memory File Generator (Maltab .m code)

```
%Write X size
Xsize = 5;

%Write Y size
Ysize = 10;

x = randi([0 17],1,Xsize);
y = randi([0 17],1,Ysize);

%Write file location
fileID = fopen('C:/juanquart/conv_proy/mem_X.txt','w');
fprintf(fileID,'%02X\r',x);
fclose(fileID);

%Write file location
fileID = fopen('C:/juanquart/conv_proy/mem_Y.txt','w');
fprintf(fileID,'%02X\r',y);
fclose(fileID);
```

SIMULATION WAVEFORM FOR: Xsize = 5 and Ysize = 10







AREA

Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	50 / 32,070 (< 1 %)
Total registers	85
Total pins	64 / 457 (14 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	1 / 87 (1 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

MAXIMUM FREQUENCY

Slow 1100 mV 85C model

Slow 1100mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	326.37 MHz	326.37 MHz	clk	

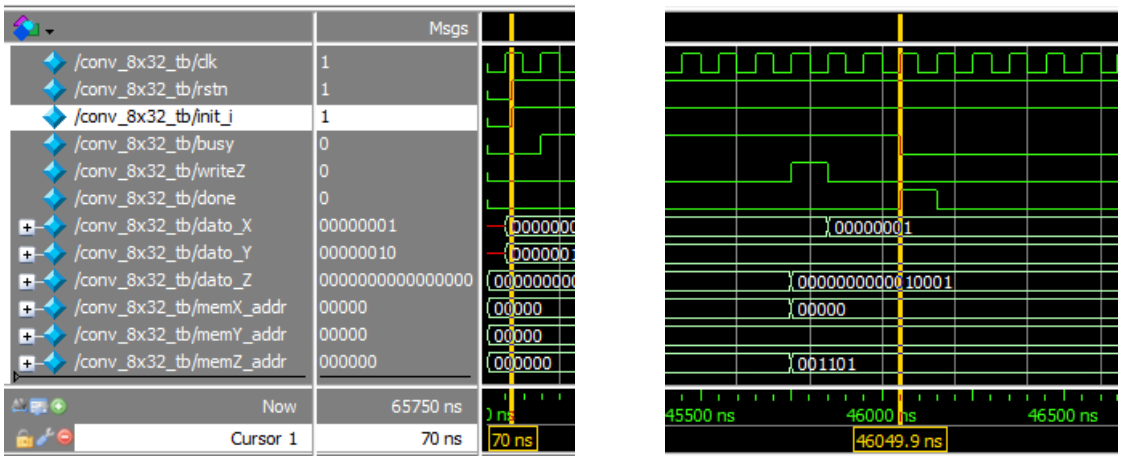
Slow 1100 mV 0C Model

Slow 1100mV 0C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	313.38 MHz	313.38 MHz	clk	

NUMBER OF CLOCK CYCLES FOR: Xsize = 5 and Ysize = 10



R.- 459 - 460 Clock Cycles, depending on wether you start counting from the first edge before the start condition, or after (meaning, if we start from the IDLE state or the INICIO state).