COL215

HW Assignment 3 Report

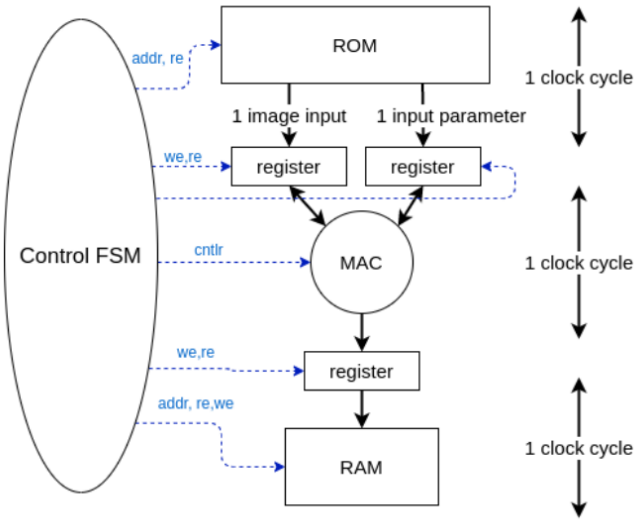
Matrix Multiplication

- Kushagra Gupta (2021CS50592)

- Pratham Agrawal (2021CS10891)

The motive of the assignment is design and implement a circuit that is used in a matrix multiplication on basys-3 board. This involves the design and integration of memories, registers, and multiplier-accumulator components (MAC) in our design. This design that we have implemented contains 4 main components:

1) MAC – The multiplier-accumulator component responsible for multiplying the matrix entries and computing partial sums for each entry of the product matrix.
2) Memories – 2 ROM units (ROM1 and ROM2) initialised to contain the 128 x 128 8-bit matrix entries responsible for performing the multiplication and 1 Single Port RAM_output unit responsible for storing the output matrix.
3) Registers – Register and Register_8bit: For storing values, functioning the same as auxiliary signals/
4) Main_fsm – Our main unit, containing an fsm to control the states numbered by 3 bits such as "000", " depending on the value of a counter signal ranging from 0 to 128^3, indicating us which addresses to take of the ROMs and where to store the Data in the RAM.
   Also, responsible to fetch out the Data stored in the RAM depending on the input addresses sent by 16 bit switches, to display on the 7 segment display, using the implementation of our first assignment.



Our approach-

We modelled our MODULAR circuit using 8 different parts (3 out of which are taken from assignment 1) as follows-

1) MATMUT – Main file taking address from switches and displaying our product matrix output (computed by sum of partial products) using index from Main.fsm and 7-segment display code
2) Main.fsm – Our implementation of the FSM involving MAC, Registers and Memories containing multiple 4-bit states, and counters.
3) MAC - MAC (Multiplier-Accumulator block) component
4) USED IN Main.fsm RAM and ROM Components (Memories) -> Registered
5) Registers – 8 and 16 bit used for storing intermediate values
6) Timing Circuit (timingcircuit.vhd)
7) 7 segment decoder (segment.vhd)
8) Mux (multiplexer.vhd)

Main.fsm

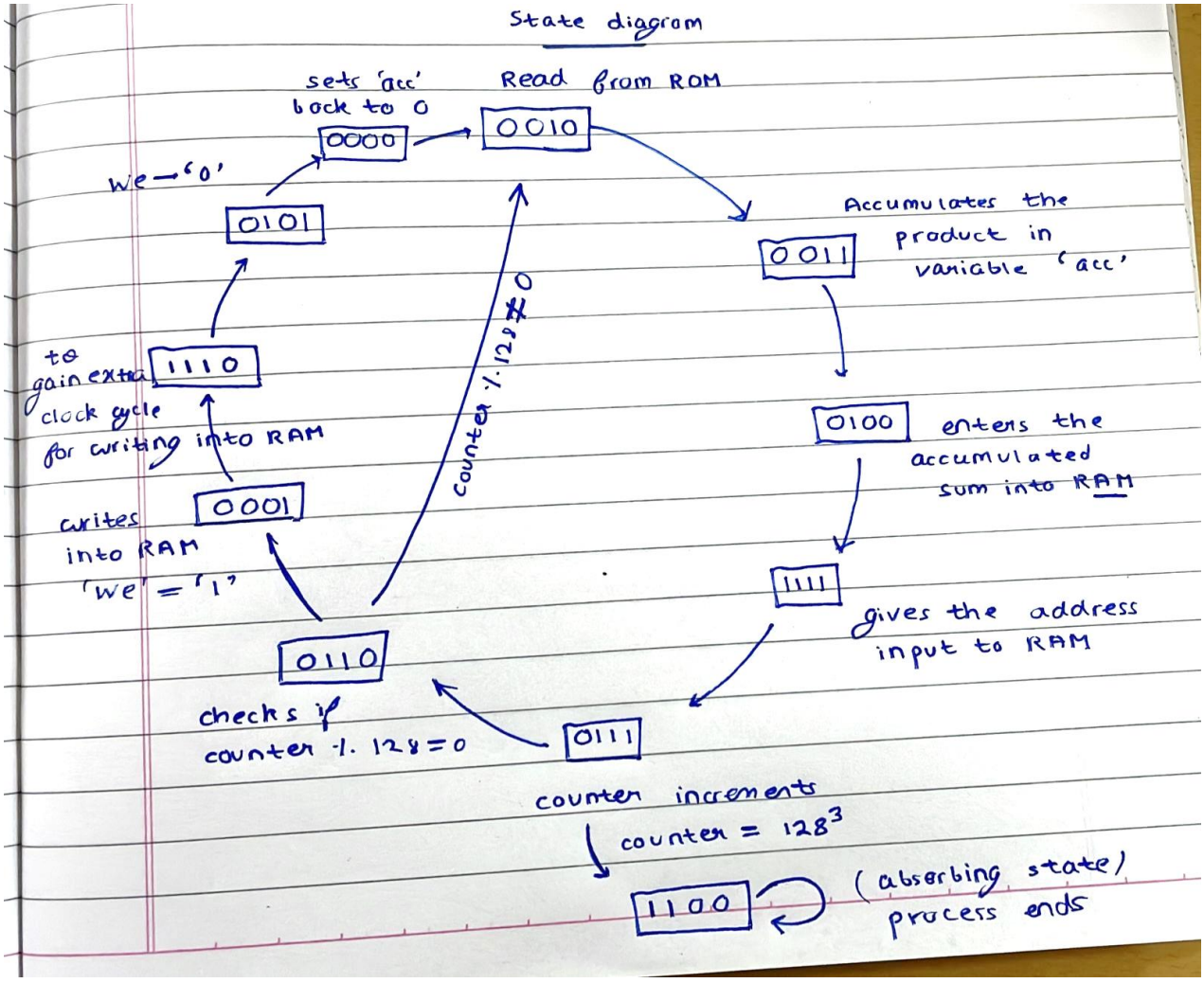


Figure : Showing our implementation of Main_Fsm involving mutliples states and conditions to move from one state to another depending on value of counter signal

We designed our fsm in such a manner that initially we are in the beginning state in which we take the 2 inputs from the given matrices that is the given .coe file and send these values to the next state where the variable "acc" accumulates the product (this code is same as what a MAC does). Initially, our counter is 0 and we calculate i(row number), j(column number) and k(temporary) in the following manner-

i = (counter/128)/128)

j = (counter/128) mod 128

k = counter mod 128

using these values of i,j and k, we calculate the address at which the accumulated sum "acc" would be written down in the RAM and give this address in the addr port of the RAM in the next state. We then increment the counter so that the value of i,j, and k changes changing the address.

Input1 has the address (i,k) in ROM1 and Input 2 has the address (k,j) in ROM2, as k varies from 0 to 127, we compute the partial sums and add them to obtain our RAM entry at address (i,j) and so the addresses sent addr1 and addr2 are 128*k+i and 128*k+j.

(GIVEN- ROM1 and ROM2 are in COLUMN MAJOR ORDER)

If the counter becomes a multiple of 128, that means that one complete row has been multiplied with one entire row and its partial product has been accumulated. So, if counter%128 becomes 0 we make the write enable "we" equals to '1', which means that we write the accumulated sum into the RAM. Otherwise, we increment the counter. We wait for 2 clock cycles for the writing process into the RAM. Once data has been written into the RAM, we move to the next state where we make the accumulated sum "acc" equals to 0, and disable the "we" that is, we set we='0'.

Once the counter reaches 128^3 that is 2097152, the matrix multiplication is complete and we make the system go into an absorbing state, i.e the process ends.

MAC –

MAC (Multiplier-Accumulator block) component-

The MAC component takes 2 inputs, call them input1 and input 2, and at the rising edge of the clock input, it multiplies the 2 given inputs and accumulate the so-obtained product into a variable say "acc" untill the multiplication of 1 entire row with 1 entire column is done. The design of MAC consists of a signal variable(register here) "temp" (initialised to 0) which contains the already accumulated sum and once it receives the product, it adds this product to the already stored value in the signal.

Temp <= input1*input2 + Temp

The incoming inputs are 8 bits each and the signal variable "temp" is of 16 bits and the final output that we get from the MAC unit is also 16bits. This final output goes to the RAM unit of memory

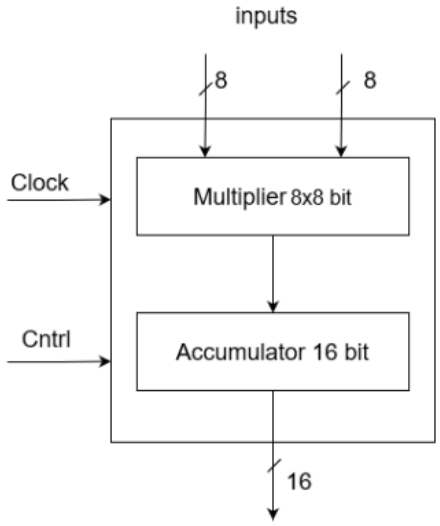


Figure 5: Multiplier-Accumulate Unit

.

high using 4 helper signals, which stores the already existing states of these switches.

And our final signal which represents the event of switching the switch *high* is defined to be *high* when new state is *high* and the helper signal state is *low* (that is already existing state is low), where we start to increment the counter q and our signal "b" if q mod 10^7=0.

ROM - We created 2 registered ROM's using the IP catalog and initialised these ROM's with the given input matrices (.coe file). The purpose of these RAM is to enable us to read the matrices as inputs and send them to the MAC unit where the multiplication and accumulation is performed

RAM (Random Access Memory) - We created 1 registered single port RAM using the IP catalog. The purpose of this RAM is to enable us to store the final output matrix in the memory of the computer. Once the multiplication is complete, we can read the data stored in the RAM by providing the address at which the so-called data to be read is stored. This is used to display the elements of the output matrix given the address of that element.

Registers – We created 2 registers – 8 bit and 16 bit and they have write and read as input ports, in addition to din and clk where write represents we = write enable, i.e when they have to write data into the internal signal REGIS from din and re=read enable represents when they have to read data from the REGIS signal to dout, which is our OUTPUT PORT. We have also used auxiliary signals in our implementation of main_fsm for Registers.

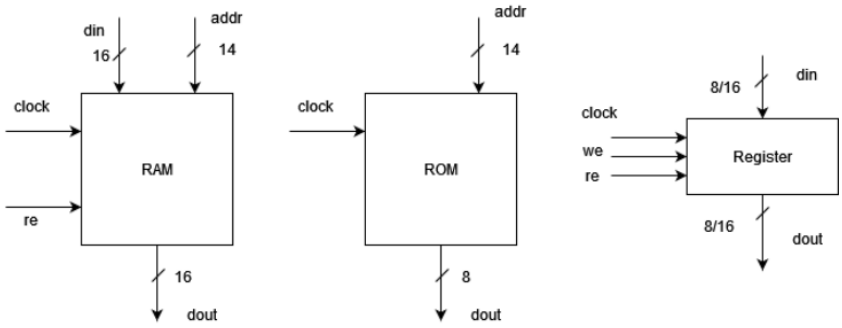


Figure 6: RAM, ROM and Registers block diagram

Segment.vhd file-

Now, after we obtain outputs from the counter.vhd file in form of the numbers to be displayed on each 7-segment (in form of vectors b0,b1,b2,b3) and providing this output as input to the segment component in our stopwatch.vhd file which performs the above task (assignment-1).

This is used to display the 4 digits on the 4 7-segment displays on the basys board at the same time. It uses 2 components, the multiplexer component and the timing circuit component as explained in the report for previous assignment.

MATMUT.vhd file-

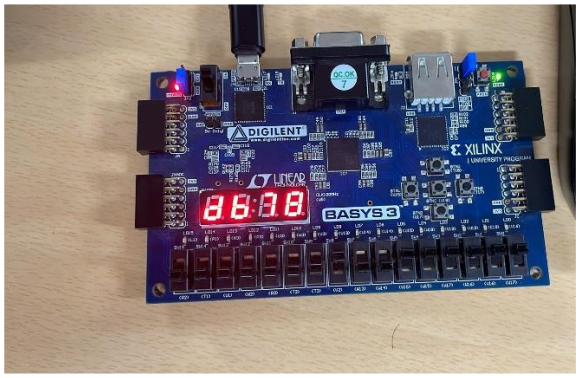
This file combines the main_fsm and segment file by taking in 14 input switches, denoting the address of our RAM output, and converting them to addri for main_fsm, reading out the element of the output matrix in the signal display or g and sending it the 7-segment display file for output on the board.

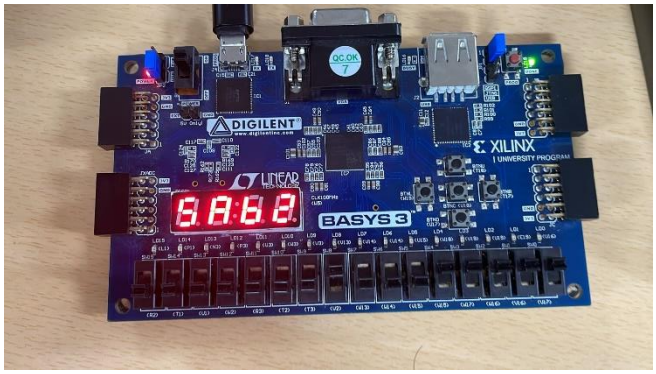
Constraint File-

The basys3.xdc file is the constraint file for our Basys-3 FPGA board and

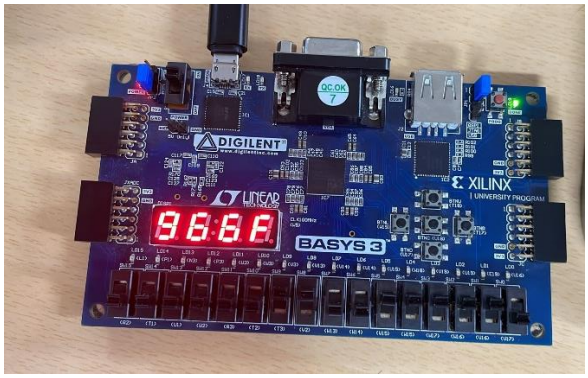
a) The input is our inbuilt clock (period 10ns - freq 100Mhz, tp =10 ns) and the 14 switches that denote the 7+7 address indices.
b) The output is our 7 segment cathode bits and anode vector called as s0, s1 s2, s3, s4, s5, s6, s7 for the cathodes G,F,E,D,C,B,A and the vector with 0 for the digit lighted up, and dp cathode, which we use to light us the decimal points as shown in the stopwatch display.

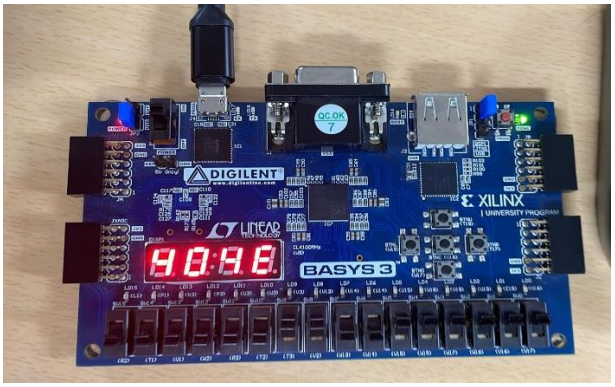
Here are a few testing inputs we used to check our strategy:





Different input switch bits

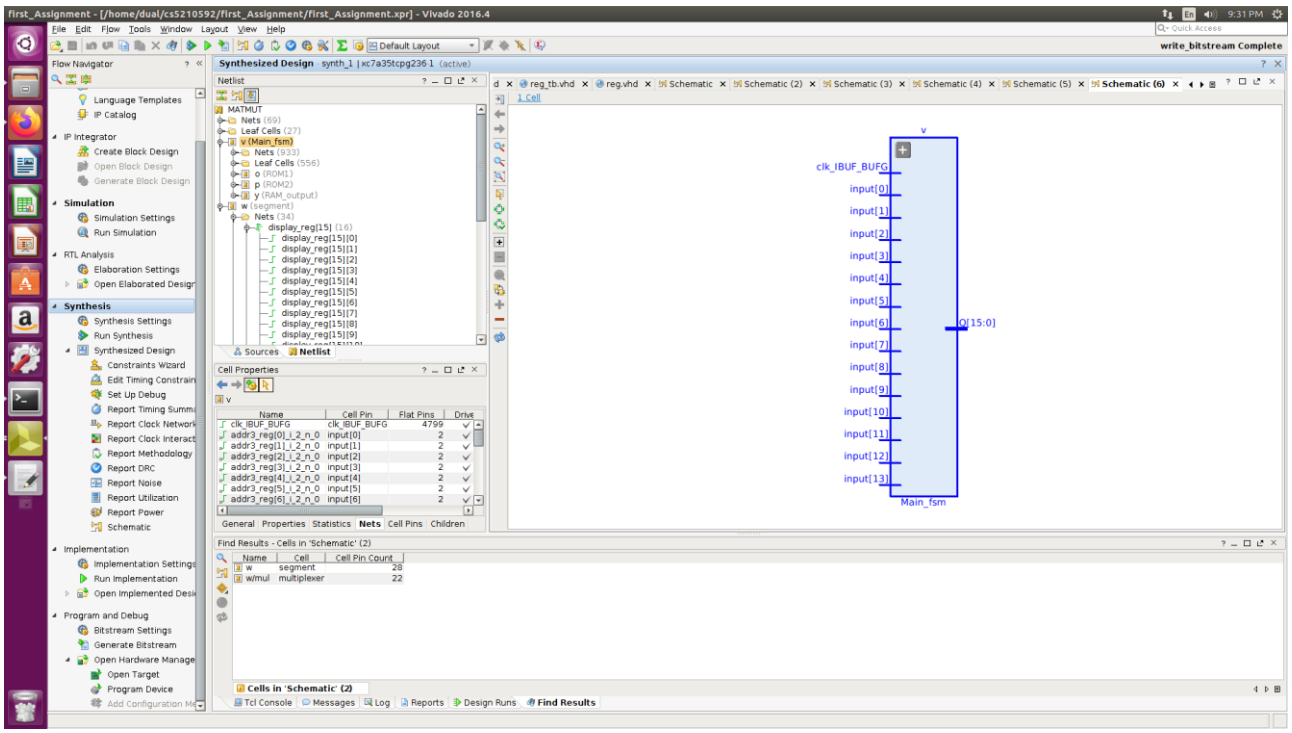




Observe the flipflops and their respective 7 segment display output

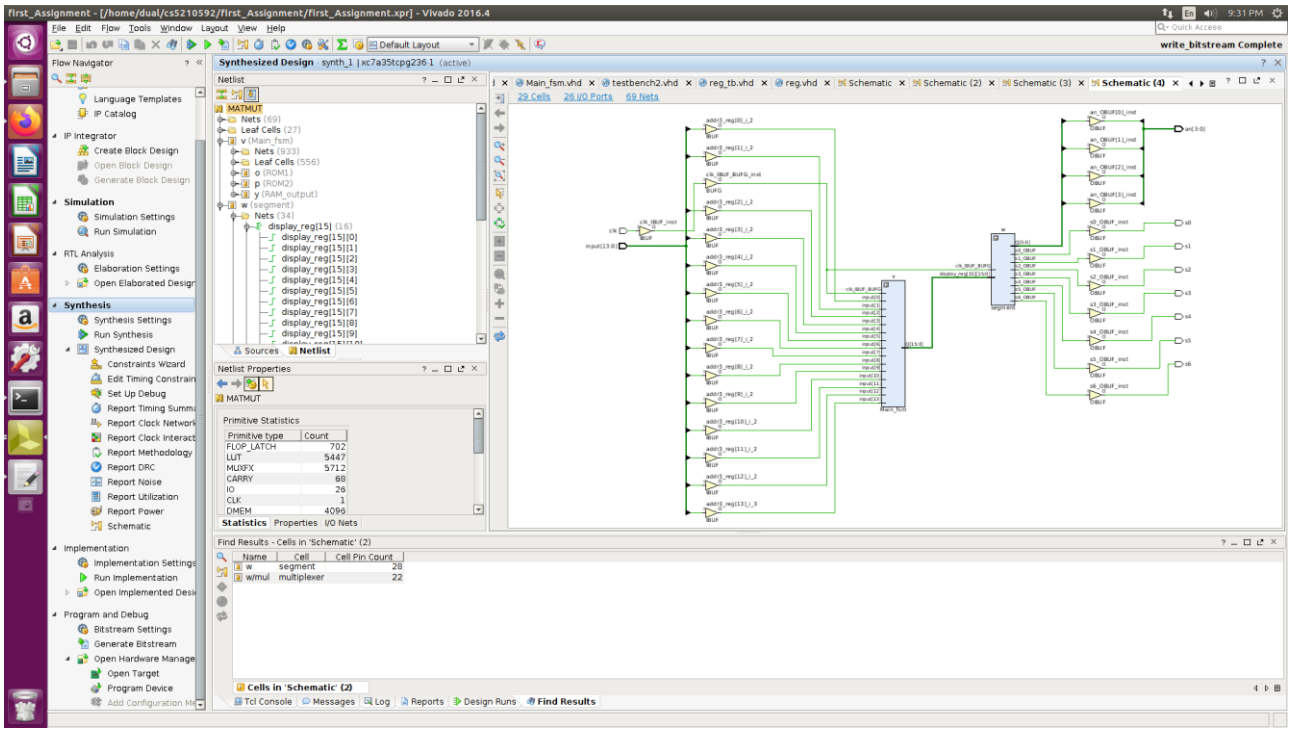
Here is a drive link containing a video of the various combinations of input switches we tested on our board and their desired outputs for our matrix multiplier:
Matrix_Mult_Test_1 Matrix_Mult_Test_2

Our Synthesized design (and block diagram obtained from the synthesis)



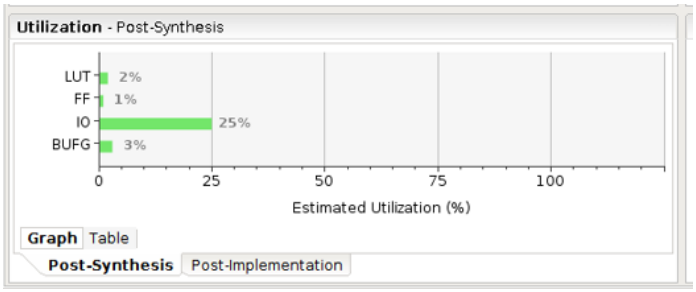
Block Diagram showing our schematic for the Main_fsm our FSM file used to generate the partial sums and control signals sending them to store our computed sum in RAM, by calculating them using input signals from ROM.

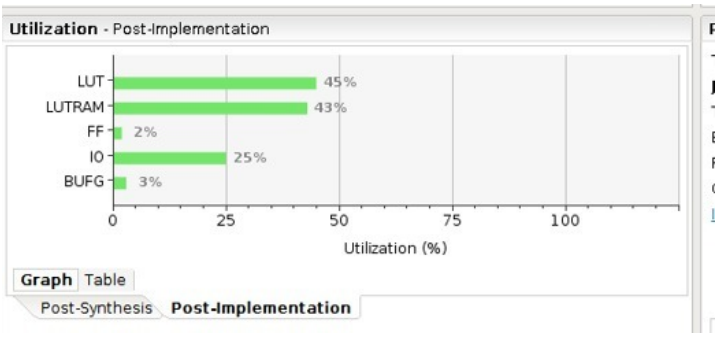
Netlist



Once, we have successfully synthesized our circuit, we run implementation on it and generate the bitstream file (.bit) uploaded used to program the device.

After our synthesis and implementation note the resource counts, and how they vary based on the implemented design.

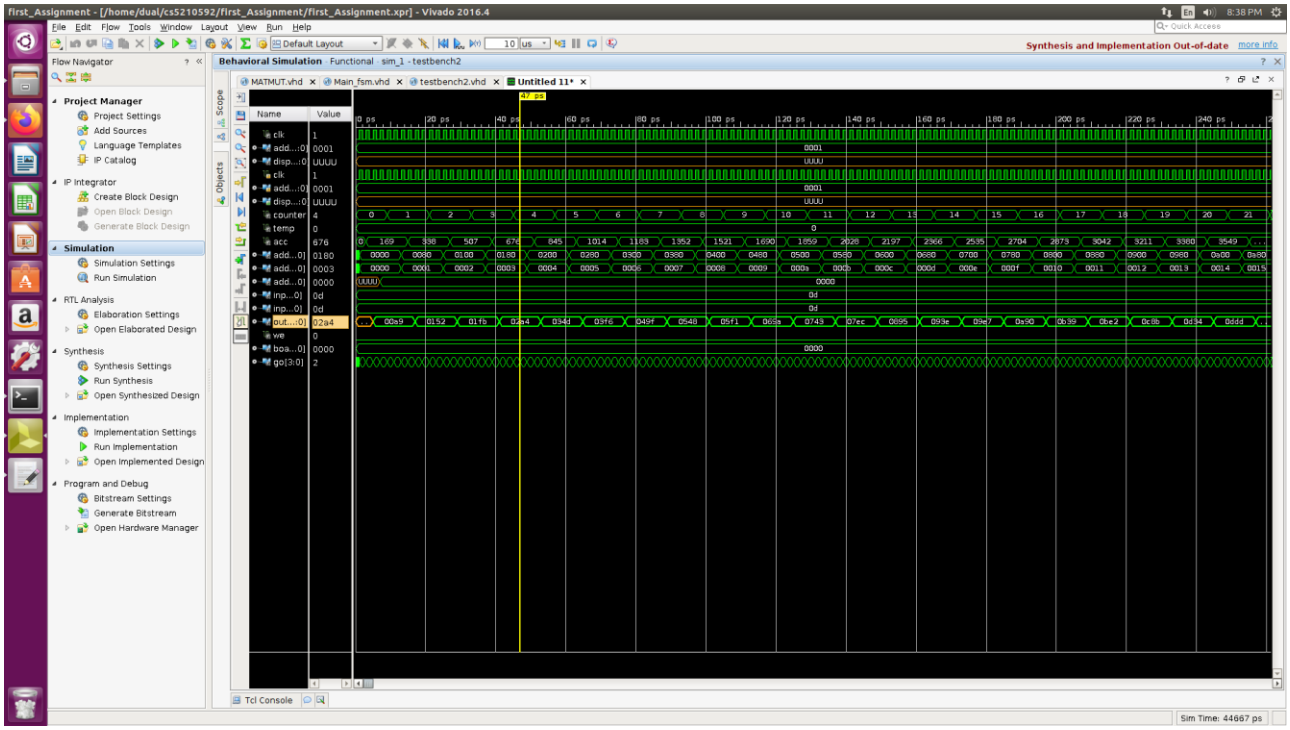


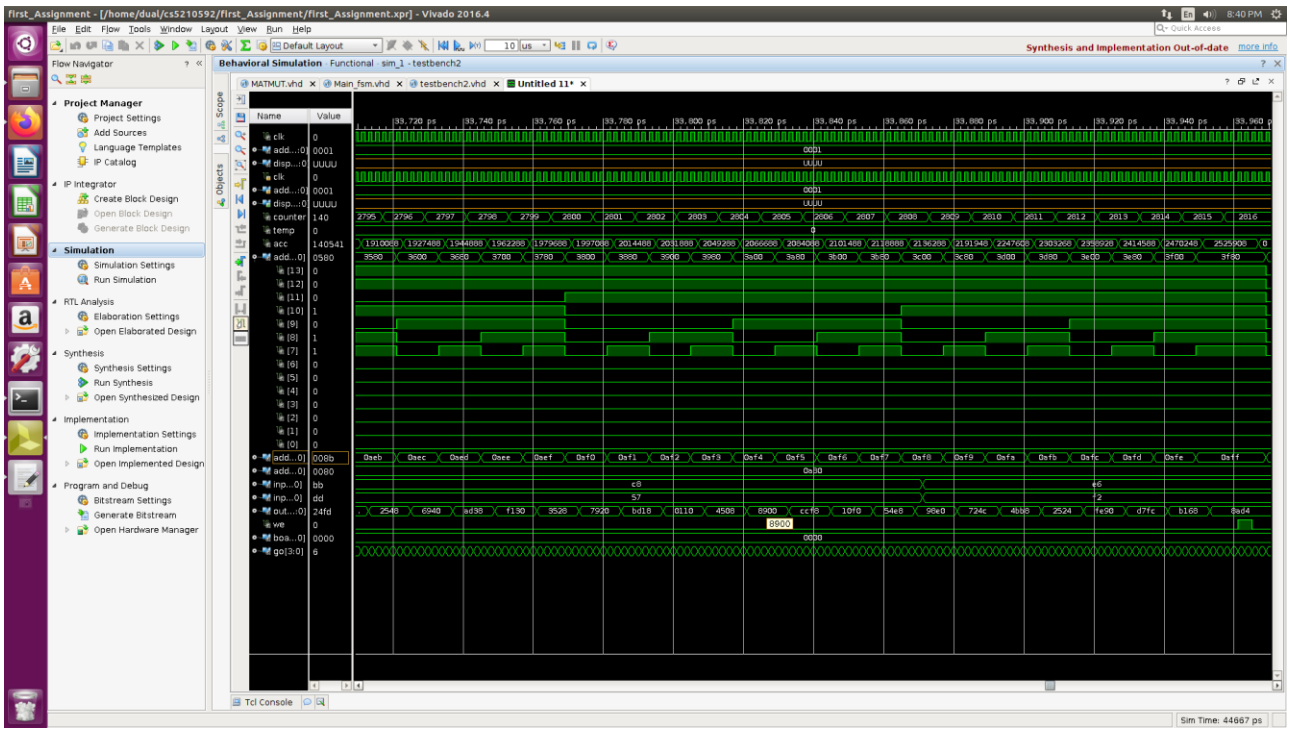


Testbench input and output (testbench2.vhd – for main_fsm, MAC_tb, reg_tb) -

We used a testbench on our main function where we use an oscillating clock with 10ns timeperiod so it changes parity for every 10 ns so it is 1 half.

Testbench2.vhd – Simulations for our Main.fsm Testbench

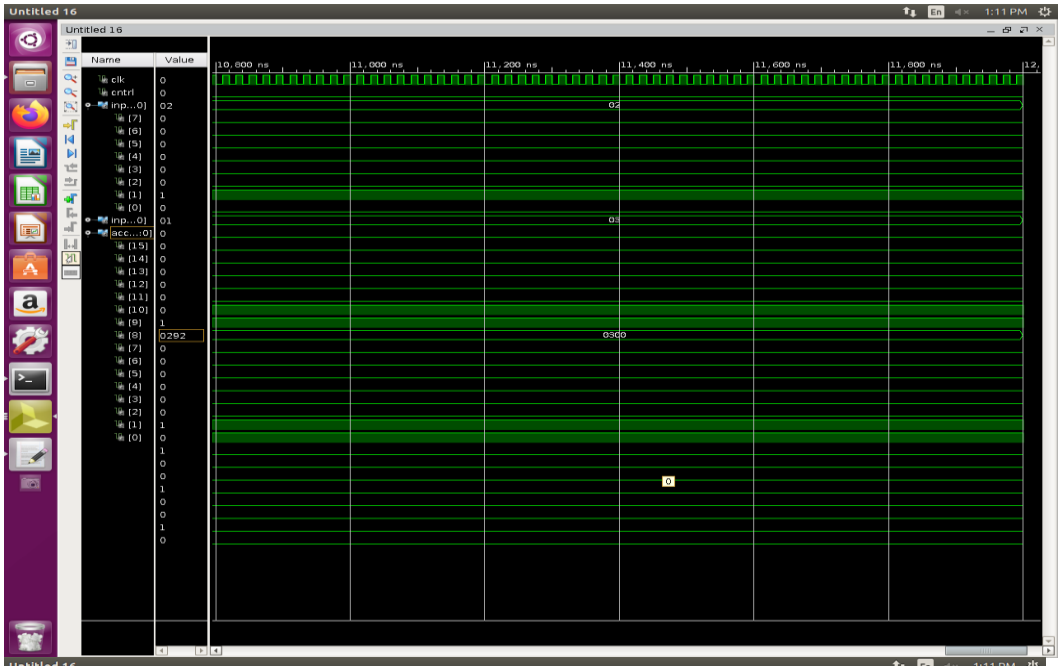


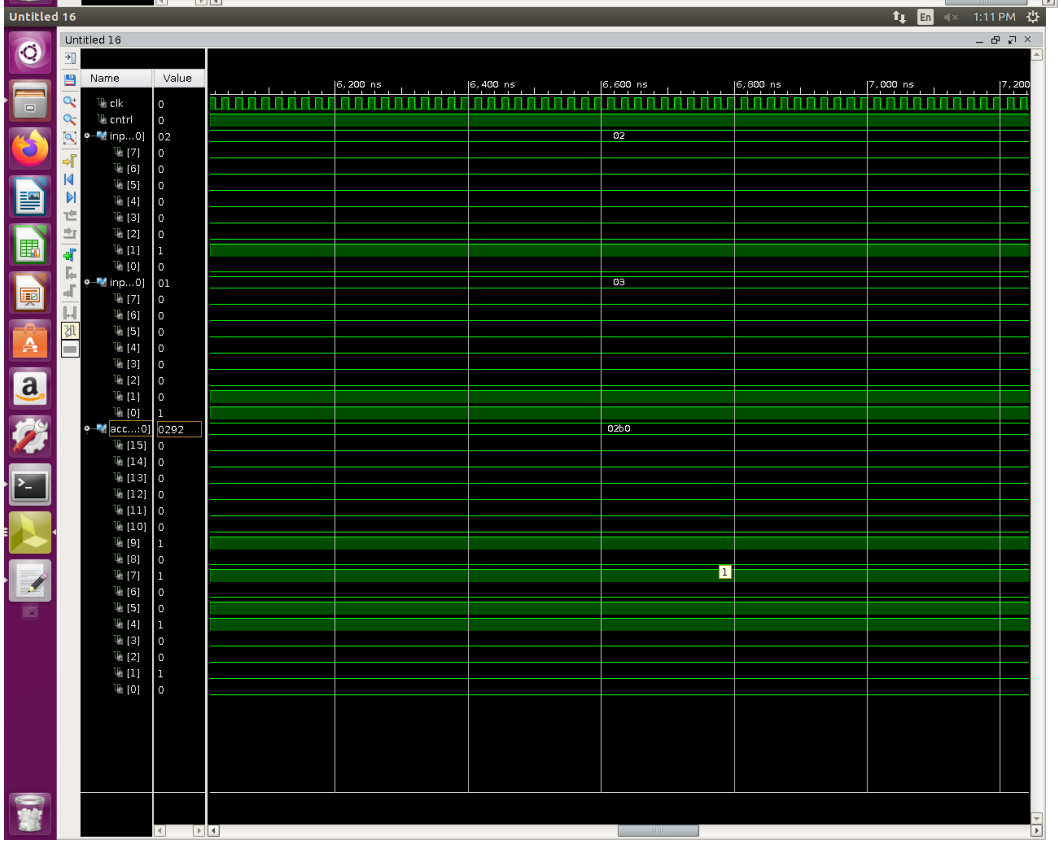



```
Input for the above waveform
begin
UUT: Main_fsm PORT MAP (clk,addrin,display);
addrin <="00000000000001";
clk<= not clk after 0.001 ns;
```

Here clock cycle changed from 10ns to 1ps to observe accumulation over matrix entries, here accumulate and addr3 signal changes while addr1 is same.

MAC_tb.vhd – Simulations for our MAC Testbench

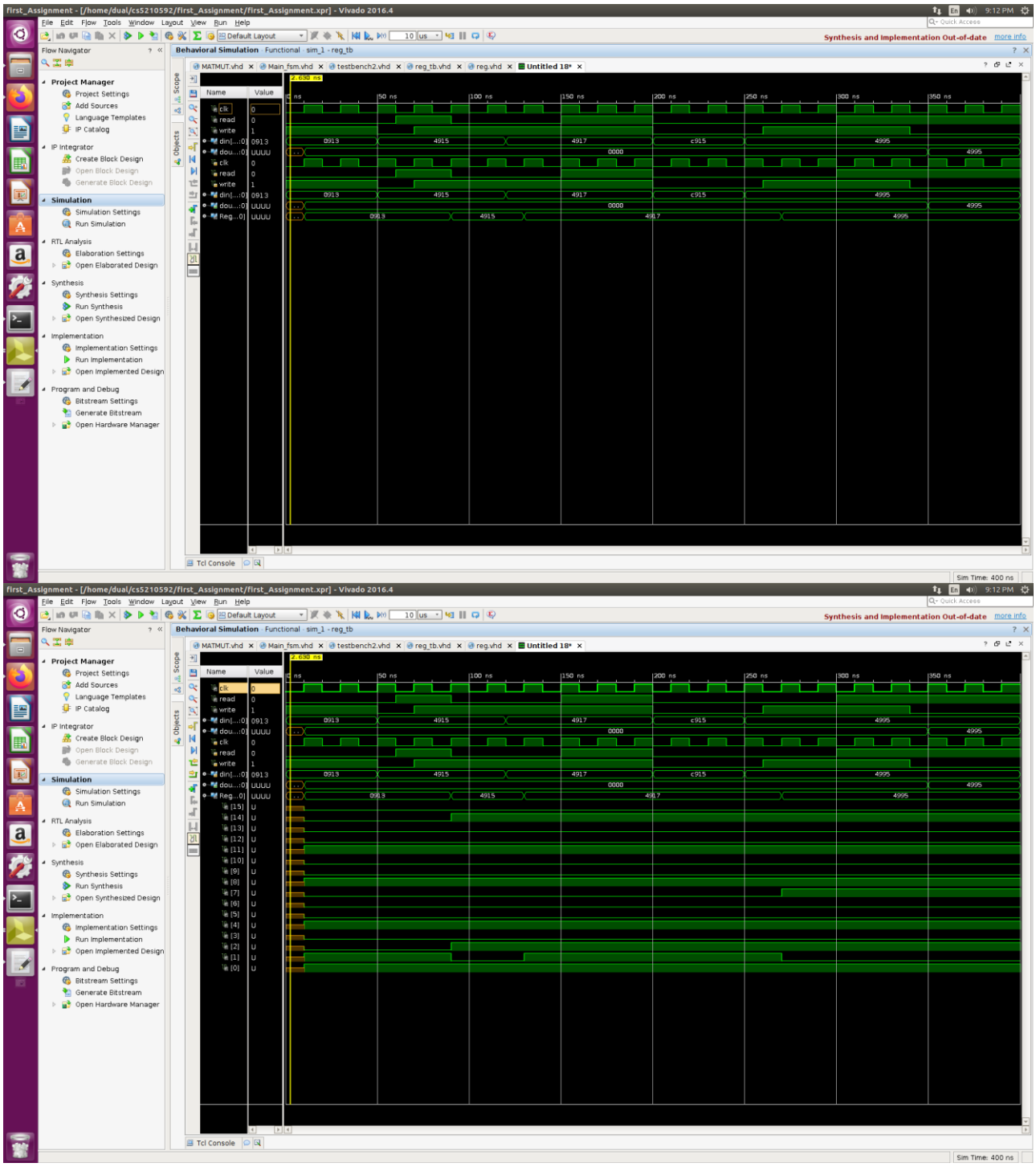





```
Input for the above waveform
UUT:MAC port map (input1, input2, clk,cntrl,accumulate);
clk<= not clk after 10 ns;
   cntrl <= '0' after 2560 ns, '1' after 2600 ns, '0' after 5160 ns, '1' after
5200 ns, '0' after 7760 ns, '1' after 7800 ns, '0' after 10360ns ;
   input1 <= "00000111", "00000001" after 640 ns, "00001011" after 1200ns,
"00000010" after 1800ns ;
   input2 <= "00000001", "00000001" after 520 ns, "00000011" after 3000ns;
end tb;
```

Note the values of acc change according to our cntrl, input1 and input 2 signals.

reg_tb.vhd – Simulations for our Register Testbench



Note that the regis storing signal changes depending upon our write and read signals and the data provided to our testbench, din.

```
Input for the above waveform
clk<= not clk after 10 ns;
din <="0000100100010011", "0100100100010101" after 50 ns, "0100100100010111"
after 120 ns, "1100100100010101" after 200 ns,"0100100110010101" after 250 ns;
read <= '0', '1' after 60 ns, '0' after 90 ns, '1' after 150 ns, '0' after 200
ns, '0' after 250 ns, '1' after 300 ns;
write <= '1', '0' after 50 ns, '1' after 70 ns, '0' after 200 ns , '1' after 260
ns, '0' after 340 ns ; end tb;
```

Now, in order to see our oscillating clock and cyclic output more clearly, we can change the period of output from 4 ms to 40 ns (in timing circuit,just for observing the waveform clearly) and we see the following waveforms clearly for the following input:

Synthesis report resource counts: Flip-flops, LUTs, BRAMs, and DSPs (.rpt also uploaded)

```
Copyright 1986-2016 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------------------------------
---
| Tool Version : Vivado v.2016.4 (lin64) Build 1756540 Mon Jan 23 19:11:19 MST 2017
| Date         : Sun Nov 13 21:19:39 2022
| Host         : yamuna running 64-bit Ubuntu 16.04.7 LTS
| Command      : report_utilization -file MATMUT_utilization_synth.rpt -pb
MATMUT_utilization_synth.pb
| Design       : MATMUT
| Device       : 7a35tcpg236-1
| Design State : Synthesized
------------------------------------------------------------------------------------------------------------
---

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |  392 |     0 |     20800 |  1.88 |
|   LUT as Logic          |  392 |     0 |     20800 |  1.88 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         |  165 |     0 |     41600 |  0.40 |
|   Register as Flip Flop |  165 |     0 |     41600 |  0.40 |
|   Register as Latch     |    0 |     0 |     41600 |  0.00 |
| F7 Muxes                |    0 |     0 |     16300 |  0.00 |
| F8 Muxes                |    0 |     0 |      8150 |  0.00 |
+-------------------------+------+-------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full implementation, is typically
lower. Run opt_design after synthesis, if not already completed, for a more realistic count.


1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
```

0	Yes	-	-
0	Yes	-	Set
0	Yes	-	Reset
0	Yes	Set	-
165	Yes	Reset	-

2. Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	0	0	50	0.00
RAMB36/FIFO*	0	0	50	0.00
RAMB18	0	0	100	0.00

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

3. DSP

Site Type	Used	Fixed	Available	Util%
DSPs	0	0	90	0.00

4. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	26	0	106	24.53
Bonded IPADs	0	0	10	0.00
Bonded OPADs	0	0	4	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	104	0.00
GTPE2_CHANNEL	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	106	0.00
OLOGIC	0	0	106	0.00

5. Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	1	0	32	3.13
BUFIO	0	0	20	0.00
MMCME2_ADV	0	0	5	0.00
PLLE2_ADV	0	0	5	0.00
BUFMRCE	0	0	10	0.00
BUFHCE	0	0	72	0.00
BUFR	0	0	20	0.00

6. Specific Feature

Site Type	Used	Fixed	Available	Util%

BSCANE2	0	0	4	0.00
CAPTUREE2	0	0	1	0.00
DNA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE2	0	0	1	0.00
ICAPE2	0	0	2	0.00
PCIE_2_1	0	0	1	0.00
STARTUPE2	0	0	1	0.00
XADC	0	0	1	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	165	Flop & Latch
LUT1	99	LUT
LUT2	95	LUT
LUT6	88	LUT
LUT5	75	LUT
CARRY4	68	CarryLogic
LUT3	45	LUT
LUT4	37	LUT
IBUF	15	IO
OBUF	11	IO
BUFG	1	Clock

8. Black Boxes

Ref Name	Used
ROM2	1
ROM1	1
RAM_output	1

9. Instantiated Netlists

Ref Name	Used