COL215 Hardware Assignment 3

Mihir Kaskhedikar, 2021CS10551
Dhruv Ahlawat, 2021CS10556

Novembr 2022

1 Introduction

In this assignment we perform 128*128 matrix multiplication on Basys3 board by using components like ROM, 8-bit registers, 16-bit registers, Multiplier and Accumulator(MAC) and RAM and construct a Finite State Machine (FSM) that controls the functioning of each components and helps co-ordinate between different components. We describe the components first.

2 Components

ROM: It stands for read only memory. The two matrices that need to be multiplied are stored in ROM. We have used a registered in-built ROM that consists of 2 input signals and 1 output signal. The input signals are clock and address while the output is value stored at the address. Using the clock we set the frequency of changing of address and thus we can read values stored in the memory which change at a definite frequency. For this assignment we have set the address to be 14 bit and value to be 8-bit.
RAM: It stands for random access memory. Output matrix is stored in RAM. It consists of 4 input signals and 1 output signal. The input signals are address, input data, clock and read-enable. The output signal is output data. When the read enable of RAM is set, it means that we can access the value at the address send as input. RAM clock is for controlling the frequency of changing of address so that the new input data is stored in a new location. For this assignment we have used a RAM which has a 14-bit address and 16-bit data.
Registers: Registers are sequential elements (set of flip-flops) that can be used to store data at clock edges. 3 signals are used to control the register. Clock(clk) signal is used to provide a rising edge, upon which the code of the register gets executed. Write enable(we) signal controls when the value stored in the register changes and Read enable(re) signal controls when the value stored in the register can be read. In this assignment we have kept the read enable set for all the registers and just control the write enable signal.
MAC: MAC has 4 inputs: Clock(clk), first number to be multiplied (num1), second number to be multiplied (num2) and cntrl signal. The MAC process is executed whenever clk gets a rising edge. num1 and num2 are the two numbers to be multiplied which we multiply by using the inbuilt multiplier of Vivado. The cntrl signal is used to know if the multiplication result needs to get accumulated further or not (i.e. we are checking if the numbers to be multiplied are first in the row and column respectively or not). If it is not set, we add the multiplication value to the accumulated value using inbuilt adder of Vivado else we reset the accumulated value to the multiplication value. MAC also has one output which just stores the accumulated value.
FSM: In this we create states the matrix-multiplication process goes into and define the state transitions with the help of signals. Our FSM consists of 6 states: **transition state**, **read**, **processing**, **writing**, **endstate** and **secondendstate**. We wrote the functions of state transition inside a process which gets called whenever the current state changes. To enable changing between states we declared 2 variables CurState and NextState which store the current state and the expected next states resepctively and at every rising edge of the clock passed in FSM, NextState is assigned to CurState and the process of state transitions is triggered. We desribe the detailed working of the FSM and each state of it in the next section.

3 FSM Variables

We first state all those variables that we use in the FSM for port mapping to components.
signal i: It stores the address of the number in the first matrix on which operation is performed in decimal form. It is initialized to 0.
signal j: It stores the address of the number in the second matrix on which operation is performed in decimal form. It is initialized to 0.
signal AddrRam: It stores the address in the RAM where the result needs to be stored. It is initialized to -1
signal AddrJVec: It is signal i in binary.
signal AddrIVec: It is signal j in binary.
signal AddrRamVec: It is signal AddrRam in binary.
signal ValueAtAddressOne: It is the value that is stored in AddrIVec and the input to the 8-bit register corresponding to the first matrix.
signal ValueAtAddressTwo: It is the value that is stored in AddrJvec and the input to the 8-bit register corresponding to the second matrix.
signal firstnum: It is the fist operand of the multiplication in MAC and the output of the 8-bit register corresponding to the first matrix.
signal secondnum: It is the second operand of the multiplication in MAC and the output of the 8-bit register corresponding to the second matrix.
signal writeNum1: It is the write enable signal of the 8-bit register corresponding to the first matrix.
signal writeNum2: It is the write enable signal of the 8-bit register corresponding to the second matrix.
signal storingOutPut : It is the input that is passed to the 16 bit register.
signal RegisterOutPut ; It is the output that is passed by the 16 bit register.
signal writeRAM: It is the write enable signal of the RAM.
signal writeRegisterRam: It is the write enable signal of the 16-bit register corresponding to RAM.
signal ValueAtRAM: It is the value outputted by RAM. (which would be needed while displaying the output matrix entries).
signal cntrl: It is mapped to the cntrl signal of the MAC.
signal MACClk: This would be passed as a clock to the MAC. So when it would be set, it means the MAC operations must be performed otherwise they shouldn't be. Also note that for components other than MAC, the clock passed is same as the clock of FSM.
signal done: It would be set when matrix multiplication process is complete. So initially it is not set.

4 FSM States

Initially the curState is kept as **transition state** and when in transition state we go to **read** state. Basically **transition state** is just to ensure that the **read** state is not skipped when the code is run.

Read

In this state, we are extracting the values of num1 and num2 to be multiplied. To do that we first reset the **MACClk** and set signals **writeNum1** and **writeNum2**. This ensures that all MAC processes get stopped and **firstnum** and **secondnum** get updated to the values stored at adresses **i** and **j** respectively. Now if i and j were such that they are the addresses of the first entries of a new pair of row and column and not both of them are 0, (i.e. the value accumulated in the MAC is the desired entry at the current address of the RAM), then we need to store the accumulated value in the RAM first before we begin the next computation. Hence we set the nextState to **Writing** and **cntrl** to 1 as the next computation is going to start afresh. Also we ensure here that if i becomes 128^2, then process is completed and **Done** is set. If i and j aren't the addresses of the first entries, we set the nextState to **Processing**, reset **cntrl** and also reset **WriteRegisterRam** so that accumulation doesn't stop.

Processing

In this state, we are carrying out the multiplication of **firstnum** and **secondnum** and also changing **i** and **j**. To do so we first reset all the write enable signals of all components, namely **writeNum1**,

writeNum2, **writeRAM** and **writeRegisterRam** and **MACClk** is set to perform MAC operations. Also to change i and j we consider 3 cases that can be seen from the following code (here MD is 128):

```
if(i mod MD = MD-1 and not(j= (MD*MD)-1)) then
            i<=i-MD+1;
            j<=j+MD-(MD*MD)+1;

            elsif(i mod MD=MD-1 and j=MD*MD-1) then
            i<=i+1;
            j<=0;

            else
            i<=i+1;
            j<=j+MD;
end if;
```

Finally we set nextState as **Read**.

Writing

Here we need to store the accumulated value in the RAM. So we set **writeRAM** and **writeRegisterRam** and increment **AddrRam** by 1. If **done** is set we set nextState to **EndState** else we set nextState to **Processing** (note that we need not go to Read state as for reading the next values as we had already done that before whenever we came to Writing state).

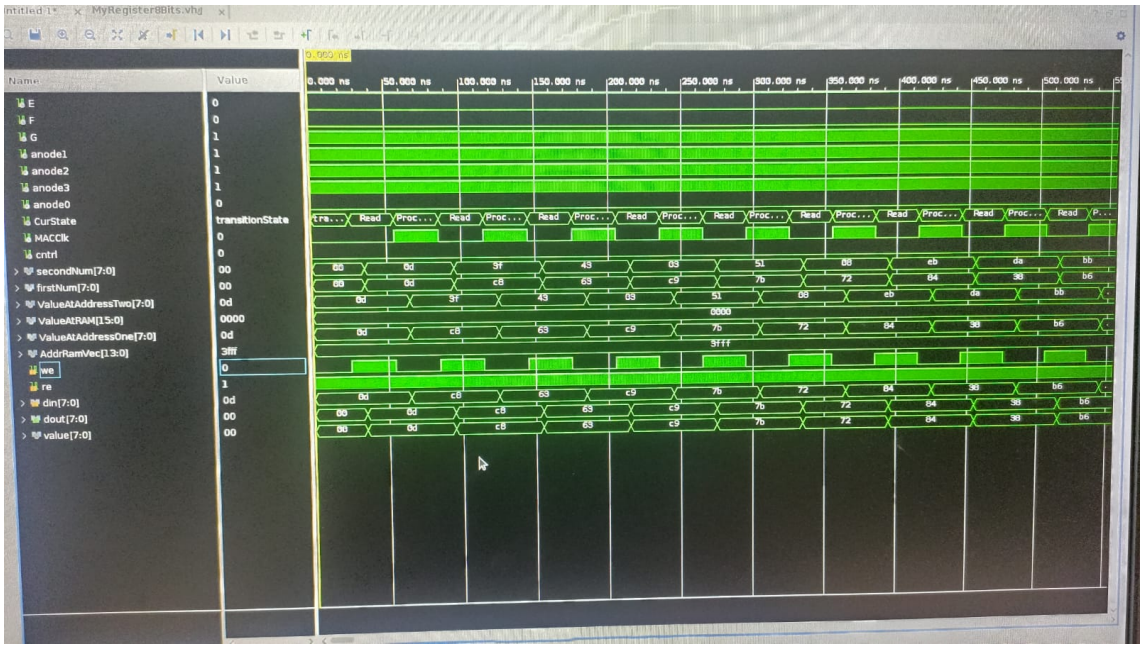
EndState and SecondEndState

Here we would oscillate between these 2 states to ensure that states keep on changing continously which we require for displaying the matrix entries on the FPGA board.

5 Displaying matrix entries

We take the 14-bit number given on the FPGA as input and pass it as the address location to RAM which in return gives the value stored at this address which is shown on the displays by importing the code of assignment 1 as a component.

6 Simulation Snapshots



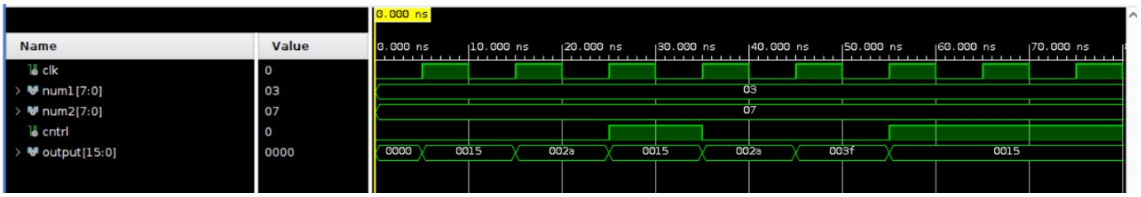


Figure 1: MAC simulation

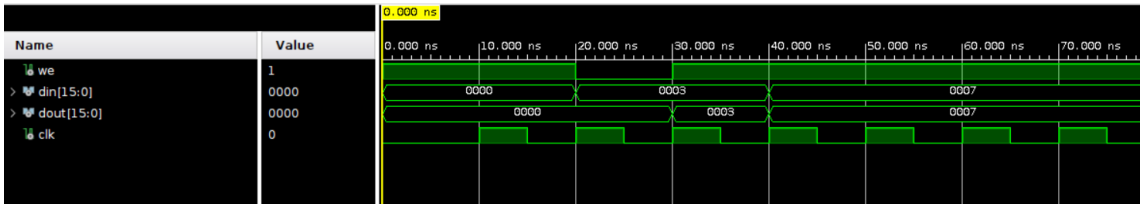


Figure 2: Register simulation

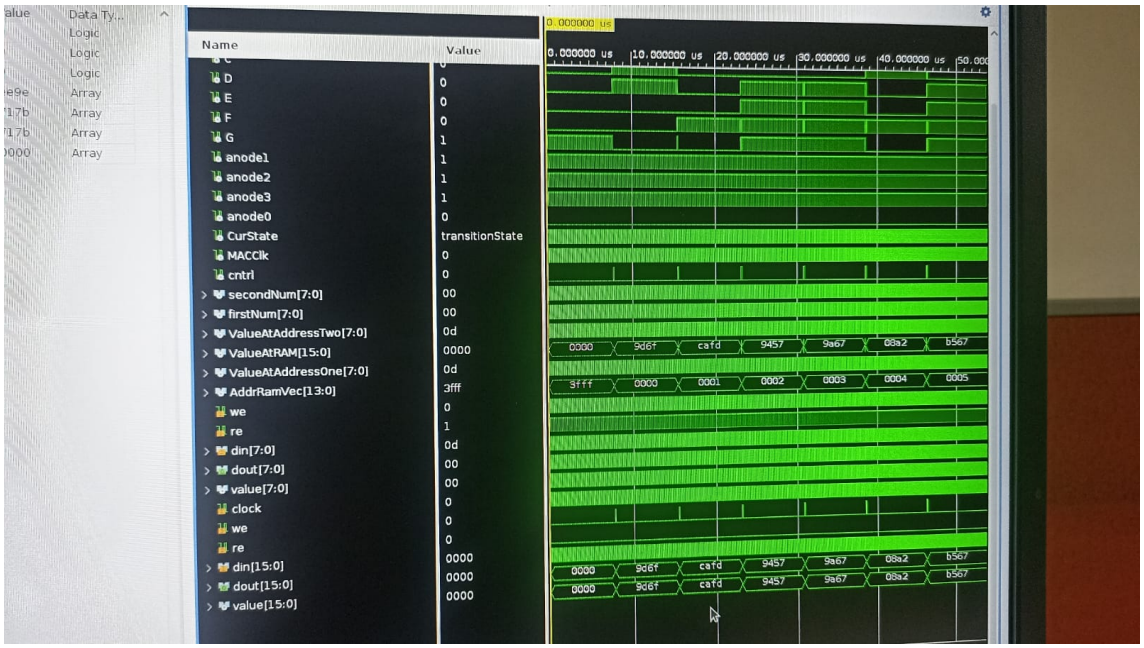


Figure 3: entire FSM

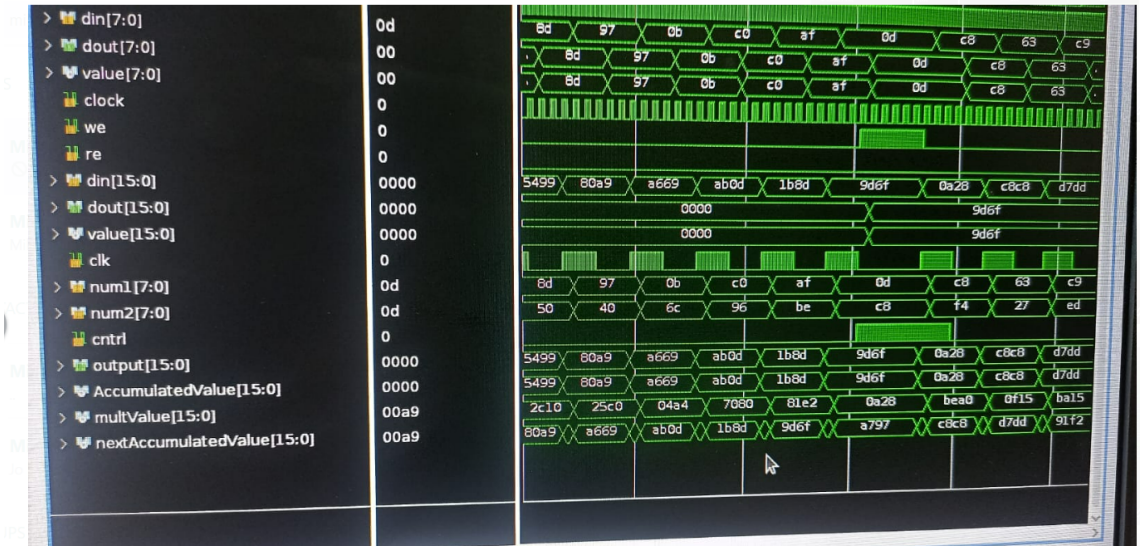
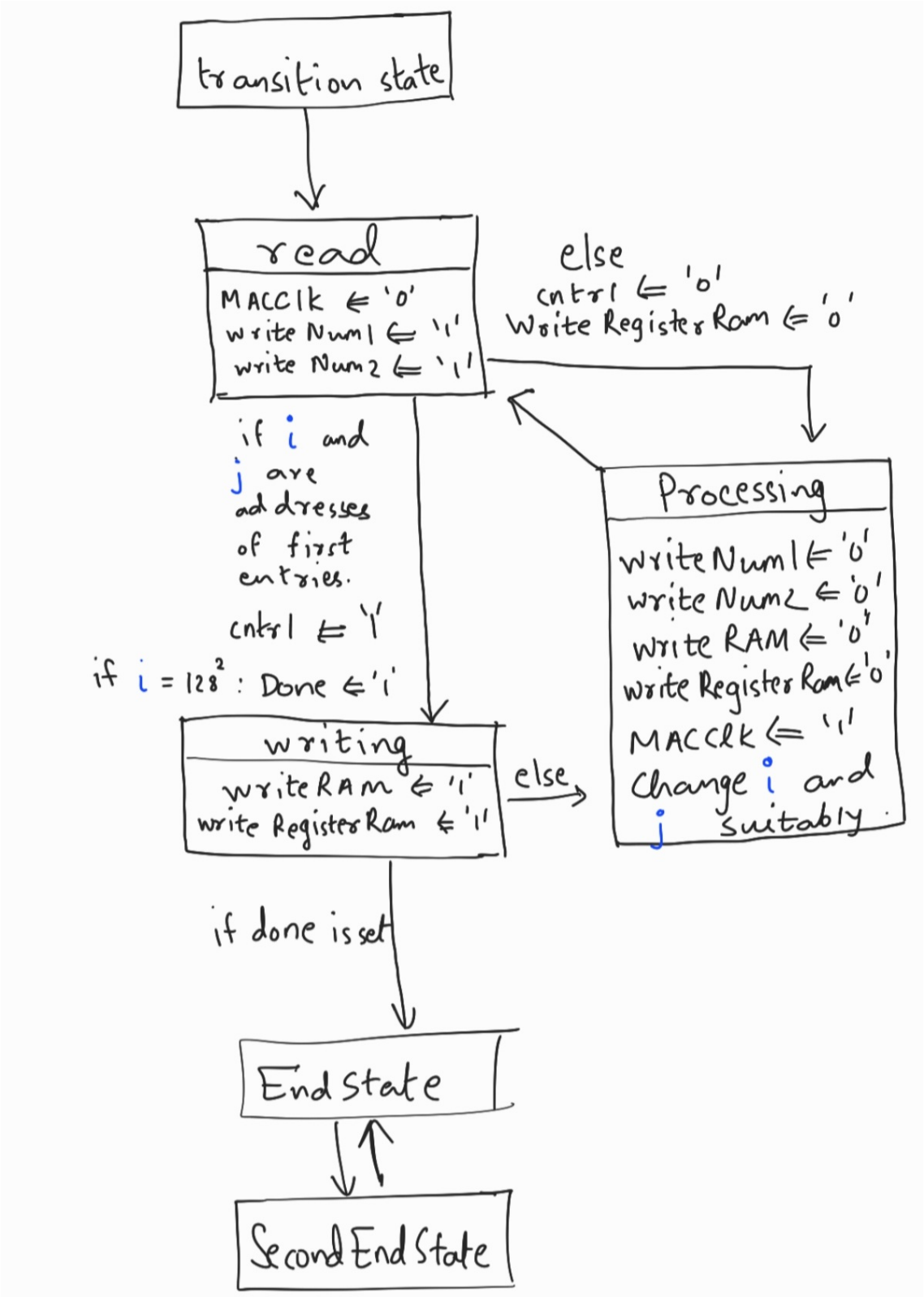


Figure 4: MAC inside FSM

7 Basic Block Diagram



8 Synthesis Report


```
------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2022.1 (lin64) Build 3526262 Mon Apr 18 15:47:01 MDT 2022
| Date         : Mon Nov 14 04:51:40 2022
| Host         : dhd running 64-bit Ubuntu 20.04.3 LTS
```

```
| Command      : report_utilization -file FSMWorking_utilization_synth.rpt -pb FSMWorking_utilizat
| Design       : FSMWorking
| Device       : xc7a35tcpg236-1
| Speed File   : -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 | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |  245 |     0 |          0 |     20800 |  1.18 |
|   LUT as Logic          |  245 |     0 |          0 |     20800 |  1.18 |
|   LUT as Memory         |    0 |     0 |          0 |      9600 |  0.00 |
| Slice Registers         |  204 |     0 |          0 |     41600 |  0.49 |
|   Register as Flip Flop |  120 |     0 |          0 |     41600 |  0.29 |
|   Register as Latch     |   84 |     0 |          0 |     41600 |  0.20 |
| F7 Muxes                |    0 |     0 |          0 |     16300 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |      8150 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full implementation, is typically


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 |
| 84    |          Yes |           - |        Reset |
| 0     |          Yes |         Set |            - |
| 120   |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+


2. Memory
---------
```

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

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only o

3. DSP

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

4. IO and GT Specific

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

5. Clocking

Site Type	Used	Fixed	Prohibited	Available	Util%
BUFGCTRL	2	0	0	32	6.25
BUFIO	0	0	0	20	0.00
MMCME2_ADV	0	0	0	5	0.00
PLLE2_ADV	0	0	0	5	0.00
BUFMRCE	0	0	0	10	0.00
BUFHCE	0	0	0	72	0.00
BUFR	0	0	0	20	0.00

6. Specific Feature

Site Type	Used	Fixed	Prohibited	Available	Util%
BSCANE2	0	0	0	4	0.00
CAPTUREE2	0	0	0	1	0.00
DNA_PORT	0	0	0	1	0.00
EFUSE_USR	0	0	0	1	0.00
FRAME_ECCE2	0	0	0	1	0.00
ICAPE2	0	0	0	2	0.00
PCIE_2_1	0	0	0	1	0.00
STARTUPE2	0	0	0	1	0.00
XADC	0	0	0	1	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	120	Flop & Latch
LDCE	84	Flop & Latch
LUT2	68	LUT
LUT6	65	LUT
CARRY4	65	CarryLogic
LUT4	55	LUT
LUT5	49	LUT
LUT3	27	LUT
LUT1	23	LUT
IBUF	15	IO
OBUF	11	IO
BUFG	2	Clock

8. Black Boxes

Ref Name	Used
OutputMatrix	1
DistMatrixTwo	1
DistMatrixOne	1

9. Instantiated Netlists

Ref Name	Used