Implementation and Evaluation of FPGA Based Arbiter Physical Unclonable Function

Zezhou Zhang, Xunyu Li, Runyu Miao, and Hanwen Zhu
Department of Electrical and Computer Engineering, University of Florida, USA

***Abstract*—Physical unclonable function (PUF) is an emerging potential security block for generating volatile secret keys in cryptographic application. Although the Arbiter PUF can be considered as an ideal tool through ASIC implementation for a long time, we still do not clearly know the performance detail of different kinds of FPGA based Arbiter PUF. In this paper, we design the 64-bit Arbiter and XOR-Arbiter Architectures with the help of Xilinx Spartan-7 FPGA and evaluate the quality of our designed PUF in three dimensions: reliability, uniqueness and uniformity by using MATLAB. In our experiment, the data size for each of the PUF is 100. Our analysis showed that Arbiter PUF reliability and uniformity are 51.94% and 61% respectively while the performance on XOR PUF are 49.82% and 54% respectively. The uniqueness of both PUFs are the same, which is 53%. Hence, in our design, the XOR PUF has a better performance than the Arbiter PUF.**

***Keyword*— Arbiter PUF, FPGA, reliability, uniqueness, uniformity.**

I. INTRODUCTION

Due to manufacturing variations, some of small random delay differences will take place on symmetrical electrical paths on chips. The entropy of the delays is sufficient to ensure a unique PUF response for each individual device instance [1].

A. Arbiter PUF

Arbiter PUF is an example of delay-based PUFs [2]. Arbiter PUFs utilize the intrinsic timing differences of two symmetrically designed paths to a single bit of the response at the output of the circuit [3]. The structure of Arbiter PUF includes several connected stages and an arbiter, which is located at the end of the PUF chain. There are three inputs and two outputs on each stage. There are also challenges on the stage, which are used to create selection signals. On the first stage, the input is a common enable signal while the last stage linked to the unique Arbiter, which determines which signal arrived at first. Because of this structure, the arbiter generates bits as response of the Arbiter PUF. Although the crossed paths are the same, the small time difference between two paths can still generate different responses.

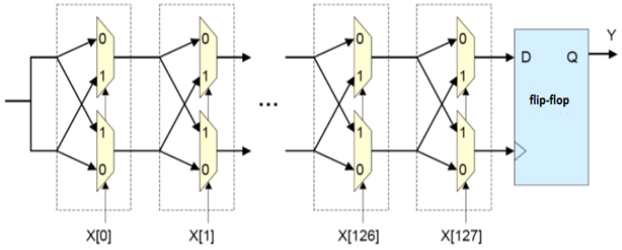

Fig.1. Multiple stage schematic of the Arbiter PUF

B. XOR PUF

Some of the machine learning methods are used to attack PUFs. In order to increase the resistance of Arbiter PUFs against machine learning attacks adding a non-linear element to the PUF design was proposed. One of the most common methods to add non-linearity to a PUF design is the XOR PUF [4]. In the structure of the XOR PUF, the responses of the PUFs are XORed to generate a final response bit. During the procession of XOR, the PUF increases its resistance to machine learning attack. Another feature is that the more PUFs are XORed, the less PUF reliability is.

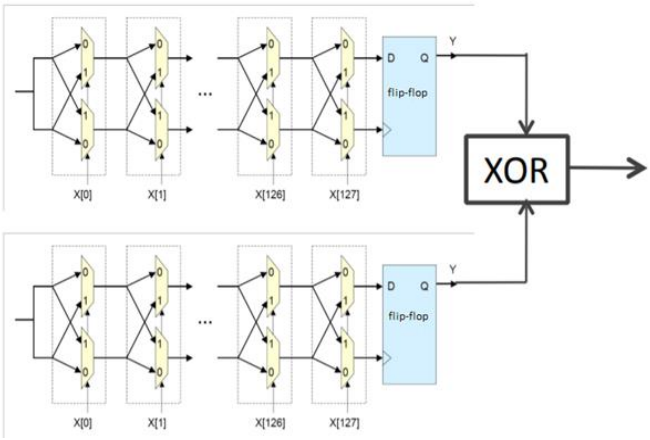

Fig.2. Schematic of XOR PUF

C. Uniqueness

Uniqueness represents the ability of a PUF to uniquely distinguish a particular chip among a set of chips of the same type [5]. It is defined as the Hamming distance between a pair of PUF identifiers. The reliability reflects inter-chip variation instead of the inter-chip variation. In general, the Uniqueness should be approximately 50%.

D. Reliability

PUF reliability is defined as how efficient a PUF is in reproducing the response bits [5]. The reliability is evaluated by detecting intra-chip Hamming Distance from several samples of PUF response bits.

Reliability=100%- INTRA Hamming Distance

E. Uniformity

Uniformity of a PUF estimates how uniform the proportion of '0's and '1's is in the response bits of a PUF [5]. If the response of PUF is considered random, the proportion should

be 50%. Therefore, the uniformity of a 64-bit PUF identifier is in proportion to its hamming weight.

II. MOTIVATION

We decided to focus on APUF and XOR PUF because of our interest in the relationship between its implementation and performance. In fact, there are plenty of papers that have already done the analysis on APUF and XOR PUF respectively. Some papers show the physical characteristics of Arbiter PUFs while the other demonstrate the insecurity of XOR arbiter PUFs. However, seldom of them combine them together and compare their differences. Nowadays, the designer should not only know about the PUFs characterization, but also consider more about their difference and suitable used place. Therefore, we hold a strong belief that it is necessary to do the research on this topic. Another motivation is improving knowledge on hardware security design. Before the beginning of this project, we believe PUF designing can be a fundamental skill for a hardware security engineer. Considering these motivations, we started this topic and tried to obtain some meaningful conclusion from it.

III. PROBLEM STATEMENT

There are several of problem we should solve in this topic:

1. How to convert the analog delay difference between paths to a digital value (how to design Arbiter)?
2. How to design symmetrical signal routing in delay-based PUFs?
3. On the BRAM design process, how to store the 64-bit challenges and output result?
4. In the implementation process, how to make sure each two clock cycles the challenges are inputted and responses are gathered, instead of missing some of input signals?
5. How to analyze the data from experiment and figure out uniqueness, reliability and uniformity?
6. What data size is suitable for both data analysis and experiment?
7. How to evaluate our result and give a conclusion?

IV. DESIGN AND IMPLEMENTATION

A. *The Principle of PUF design*

Our design target was to create a 64-bit Arbiter PUF and a XOR Arbiter PUF which both take 64-bit challenges. These two PUFs were both implemented in Arty-s7-25 board, using a Xilinx Spartan-7 FPGA. The design of PUF takes advantage of the unclonable chip signature generated by process variations. Both Arbiter PUF and XOR Arbiter PUF have the same idea that is to design a circuit that generates logic-0 or logic-1 based on process variations and obtain n-bit binary chip responses by running the circuit n times. For normal Arbiter PUF, we use one input to trigger both paths. At the end of two parallel (racing) paths, an arbiter (D flip-flop) is used to convert the analog delay difference between paths to a digital value. By inserting a path switch, you can divide the two paths into smaller sub-paths. Each set of inputs to the switch acts as a challenge set (represented by Ci), defining a new pair of race paths whose delays can be used to compare for generating a bit-response [6]. Similar to Arbiter PUF, XOR Arbiter PUF also need to observe which path reaches the arbiter with less delay, but XOR Arbiter PUF doubles the process and uses a XOR gate to finalize the output. The reason behind it is trying to create a more random result by locating the two final selectors before the arbiter. In this paper, we evaluate these two PUFs as for uniqueness, uniformity, and reliability.

B. *PUF Design Process*

In our normal Arbiter PUF design, we use 64 pairs of 2-to-1 MUXs to compose the two paths and 1 D flip-flop to be the arbiter. The two paths are connected to the clock and D input of the D flip-flop, so we can see from the output whether the input D has arrived when there is a rising edge of the clock. Similarly in our XOR Arbiter PUF, we double the process and connect the outputs of two D flip-flops to the XOR gate as a result. The major problem when designing PUFs and FPGAs, especially delay-based PUFs, is designing symmetrical signal routing. This problem can be effectively solved when using ASIC due to its hand-drawn layout; however, in FPGAs it is physically constrained by its interconnect structure [7]. In this paper, we use hard macro to implement the module that contains the necessary physical information of a pair of 2-to-1 MUXs. By duplicating the hard macros 64 times, we can create a relatively symmetrical routing. Because hard macro is pre-compiled modules, the whole design can be created within seconds when assembling those modules together. Although we can use hard macro to ease the asymmetry problem, it is still hard for designers to achieve complete symmetry between the top and bottom paths. This disadvantage can be shown from the evaluation result, which can be a potential bias problem because attackers can try to use machine learning to hack based on the CRPs.

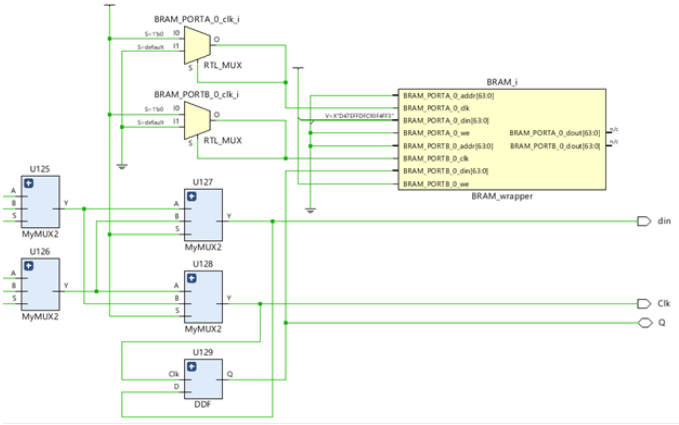

Fig.3. Output process of the Arbiter PUF

C. *BRAM Design Process*

This implementation is simple but insufficient and time-consuming because it only produces one-bit response every

time a challenge is applied. In order to produce 64-bit key effectively, it requires more work to do. Our plan was to create a BRAM IP core to store the 64-bit challenges and one output result. In addition, we use logic-1 signals to be counted as inputs for the synchronization of two paths, which means we need to make sure two paths start at the same time. In this way, the width will be 64 bits and depth will be 64 to produce one key. Also, we need to define True Dual Port RAM, which requires two cycles delay both reading the 64-bit challenge and writing 1-bit response. For PORTA, we enable the writing because it needs to store the data into the BRAM, and also enable the writing for PORTB because the response needs to be written into BRAM. When implementing BRAM, as long as the *WE* signal is working, the data is read at the positive edge of the clock cycle at the address specified by *Addr*. Value can be read from the *Rd* data, which is the data stored in the BRAM. Since only one *Rd* data value can be read per clock cycle, then if our block RAM has a depth of 64 values, there are at least 64 clock cycles to read the entire data. Then, for writing responses, there are additional 64 clock cycles to write into BRAM.

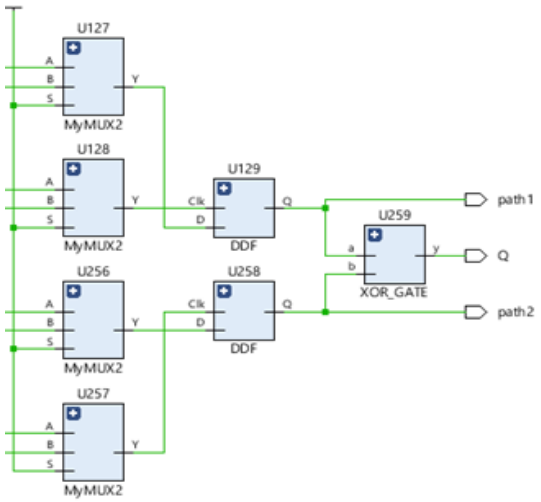

Fig.4. Output process of XOR Arbiter PUF

D. Implementation of PUF

After designing all necessary modules for Arbiter PUF, we need to specify the timing behavior of the circuit shown on Figure 1, which shows an example of producing 1-bit response in one cycle. To ensure each two clock cycles the challenges are inputted and responses are gathered, we have to specify a relatively concise constant clock time. Therefore, 3 LED lights on our Arty S7 FPGA board are used to display outputs of two paths and 1-bit response. When a LED light is on, it means logic-1; otherwise, it means logic-0. The reason switches are not used to be inputs of two paths is due to the asynchronous issues when trying to manually flip two switches at the same time. From the observation, eyes cannot distinguish the sequential order of lighting of two LED lights, which stand for two paths' output, but the result can be seen from the other LED light, which stands for the response bit. From the implementation report, we can see the maximum combinational path delay to decide the clock cycle time. When the first challenge from BRAM PORTA is inputted into selectors in one clock cycle from BRAM, the race begins. After the D flip-flop arbiters the output from top and bottom paths, the output Q can be stored into the BRAM PORTB. When implementing XOR Arbiter PUF, we use the same methods but different clock cycle time because the more complex design can cause more path delay during the process.

IV. IMPLEMENTATION RESULTS

To analyze the data from PUF, both Arbiter PUF and XOR PUF have 100 sets of data. For every dataset, the first 64 are inputs and the last one is output. We assess the data performance from its reliability, uniqueness and uniformity using MATLAB. To obtain the complete 100 datasets for analysis, we have had some reference data and chose some as dataset [8].

Reliability defines how reliable the PUF is, which can be evaluated by calculating its intra-chip hamming distance (HD), and hamming distance is the number of different bits out of 100. In MATLAB, this value is calculated every two out of 100 responses. The lower the intra-chip HD is, the higher the reliability of the PUF is, since the result responses processed by intra-chip HD are all from one single chip, and low value indicates low noise, so the chip is highly reliable.

Uniqueness stands for how to distinguish a PUF by comparing with other PUFs, which can be evaluated by calculating its inter-chip HD. In MATLAB, this value is resulted from putting 100 response data of different two PUFs in different rows. With the bigger value of inter-chip HD, the PUF shows better performance for the individual chip and will be identified easier.

Uniformity is the proportion of how many '0's and '1's in the responses of a PUF, the closer the ratio is to 1, the better the uniformity is of a chip. In MATLAB, we calculate the mean of all responses and draw a two-dimensional schematic to figure out its uniformity.

A. Arbiter PUF

For the result, the reliability of Arbiter PUF is evaluated by X one hundred percent minus the intra-chip HD, intra-chip HD is the mean of every two out of 100 responses (with the sum of 4950 values), and the result of reliability is 51.94%, which has some gaps with the ideal performance at a hundred percent.

The result of inter-chip HD comparing Arbiter PUF with XOR PUF is 53%, which is far from the ideal value at 100% but still good to meet the requirement of distinguishing different PUFs.

To evaluate the uniformity, first we calculate the mean of all 100 responses and the result is 0.61, which is not so ideal for the true uniformity to obtain the mean of 0.5 and has the same amounts of '0's and '1's.

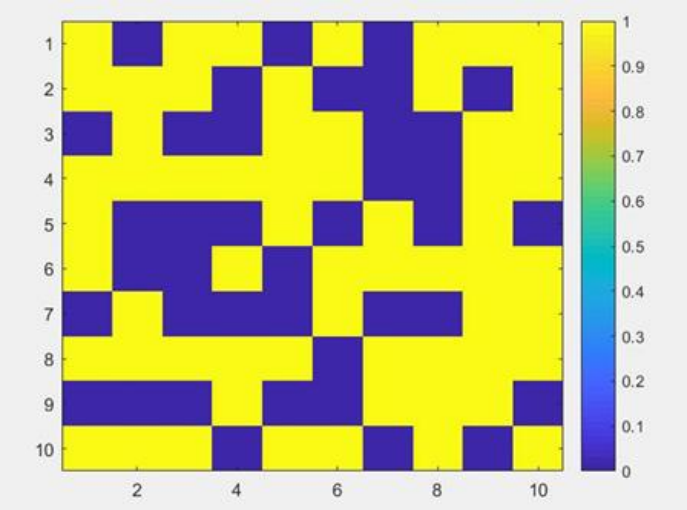

Fig.5. Two-dimensional schematic of 100 responses for Arbiter PUF

Then we draw the two-dimensional schematic in a 10x10 matrix and observe the distribution of '0's and '1's (Figure 5). In this figure, the yellow parts stand for '1' and the blue parts stand for '0', it can be clearly seen that there are more yellow parts than blue ones, and this result is consistent with the mean we calculated.

B. XOR APUF

Secondly, we analyze the performances of XOR Arbiter PUF from the data.

The first performance is uniformity. We use our FPGA to get 100 responses and put them into a 10x10 matrix, and then we achieve the distribution of 0 and 1 in Figure 6.

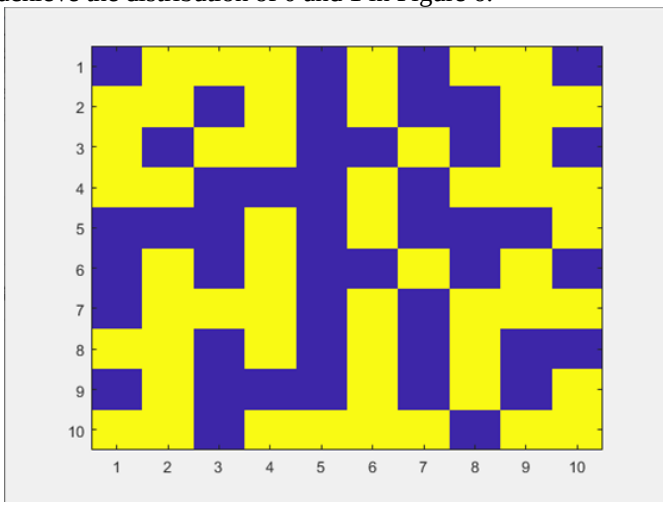

Fig.6. Two-dimensional schematic of 100 responses for XOR PUF

From the figure, we can find that the probability for '0' and '1' is random. Due to the XOR Arbiter PUFs performances of uniformity, the frequency of 0 and 1 for this FPGA is close to 50%, it means they are distributed equally likely between '0' and'1'. Hence, XOR Arbiter PUF shows good performance in uniformity.

Then, the second performance is uniqueness. This quality can be defined by the value of inter-HD distance. From the MATLAB code, we calculate the value of inter-HD distance is 53%, it means this value is close to the ideal value of 50%. Hence, the performance of uniqueness for XOR Arbiter PUF is good.

The last performance is reliability. This value is related to intra-HD distance. We can achieve this value of 50.18%, hence the value of reliability is approximately 49.82%. It is far from the ideal value of 100%. Therefore, the quality of reliability for XOR Arbiter PUF is not so good.

V. EVALUATION

To compare the result of Arbiter PUF with XOR PUF, table 1 shows the result data of Arbiter PUF and XOR PUF we processed.

	Reliability	Uniqueness	uniformity
Arbiter PUF	51.94%	53.00%	61%
XOR PUF	49.82%	53.00%	54%

Table 1. Comparison between Arbiter PUF and XOR PUF

From the table, since the uniqueness is compared from each other, they have the same value. XOR PUF shows better performance at uniformity, the proportion of '0's and '1's of XOR PUF is closer to 50% than that of Arbiter PUF. In my opinion, better performance in XOR PUF results from the adding XOR gate with two D-flip-flops (one more flip-flop than arbiter PUF). As for their reliability, both show results that's not ideal, even though arbiter PUF seems a little better, it is still difficult to achieve the ideal state. The reason might be that the output of a PUF is easily influenced by temperature, voltage, device aging and so on, so that the reliability of PUFs might be not so good as expected.

For further improvement, since many experiments show that the PUF response from same challenge input is proportional to the PUF delay difference of PUF wires, which means that the greater delay difference generates higher reliability of responses [9], so maybe increasing the delay difference can make the reliability of Arbiter PUFs perform better.

VI. CONCLUSION

In conclusion, we found that the XOR PUF has better performance on uniformity due to its unique XOR gate design. Furthermore, we discover that the greater delay difference generates higher reliability of responses. In further research, it will be good to change our design by increasing and decreasing delay difference. Then, we measure the performances by three

parameters again to verify the idea: whether increasing the delay difference can make the reliability of Arbiter PUFs perform better or not.

VII. Personal comment

Hanwen Zhu: We did a good job on evaluating the data set. However, we still do not detect the difference in the Uniqueness. In addition, I believe doing some comparison of Feed-Forward PUF may help us know more about PUF.

Zezhou Zhang: From the design and implementation process, the inner structure of FPGA we used in this paper made a restriction on the symmetry routing, which directly causes the problem of bias when producing the response key. In order to make this acceptable, using ASIC or implementing programmable delay paths can be a better choice.

Runyu Miao: The performance of reliability is far from my expectation since they are not big enough. Amounts of data set and reliable design are still needed to be improved. I think this experience will help us do better next time.

Xunyu Li: From the analysis of XOR APUF, the performances are not good enough as my expectation. Considering about not good understanding of PUFs structure, I can learn many useful knowledges during this project procedure.

References

[1] Maes, R., Verbauwhede, I.: Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. In: Towards Hardware-Intrinsic Security, pp. 3–37. Springer (2010)

[2] Lee, J.W., Lim, D., Gassend, B., Suh, G.E., Van Dijk, M., Devadas, S.: A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications. In: VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on. pp. 176–179. IEEE (2004)

[3] Tajik S. et al. (2014) Physical Characterization of Arbiter PUFs. In: Batina L., Robshaw M. (eds) Cryptographic Hardware and Embedded Systems – CHES 2014. CHES 2014. Lecture Notes in Computer Science, vol 8731. Springer, Berlin, Heidelberg

[4] Becker G.T. (2015) The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs. In: Güneysu T., Handschuh H. (eds) Cryptographic Hardware and Embedded Systems -- CHES 2015. CHES 2015. Lecture Notes in Computer Science, vol 9293. Springer, Berlin, Heidelberg

[5] Maiti A., Gunreddy V., Schaumont P. (2013) A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions. In: Athanas P., Pnevmatikatos D., Sklavos N. (eds) Embedded Systems Design with FPGAs. Springer, New York, NY

[6] N. H. N. S. Kiran and R. Bhakthavatchalu, "Implementing delay based physically unclonable functions on FPGA," *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, Ramanathapuram, 2016, pp. 137-140.

[7] M. Majzoobi, A. Kharaya, F. Koushanfar and S. Devadas , "Automated design, implementation, and evaluation of arbiter-based PUF on FPGA using programmable delay lines" , 2014.

[8] http://archive.ics.uci.edu/ml/datasets/Physical+Unclonable+Functions

[9] J. Wen, M. Huang, Z. Chen, L. Zhu, S. Chen and B. Li, "A Multi-line Arbiter PUF with Improved Reliability and Uniqueness," 2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP), Wuxi, China, 2019, pp. 641-648.

Appendix
APUF.VHD

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY APUF IS
  PORT ( Q : INOUT STD_LOGIC;
      CLK: OUT STD_LOGIC;
      DIN : OUT STD_LOGIC);
END APUF;

ARCHITECTURE BEHAVIORAL OF APUF IS
COMPONENT MYMUX2
PORT (  A : IN STD_LOGIC;
     B : IN STD_LOGIC;
     S : IN STD_LOGIC;
     Y : OUT STD_LOGIC);
END COMPONENT;

COMPONENT DDF
PORT(
     Q : OUT STD_LOGIC;
     CLK :IN STD_LOGIC;
     D :IN  STD_LOGIC);
END COMPONENT;

COMPONENT BRAM_WRAPPER
PORT (
  BRAM_PORTA_0_ADDR : IN STD_LOGIC_VECTOR ( 63 DOWNTO 0 );
  BRAM_PORTA_0_CLK : IN STD_LOGIC;
  BRAM_PORTA_0_DIN : IN STD_LOGIC_VECTOR ( 63 DOWNTO 0 );
  BRAM_PORTA_0_DOUT : OUT STD_LOGIC_VECTOR ( 63 DOWNTO 0 );
  BRAM_PORTA_0_WE : IN STD_LOGIC;
  BRAM_PORTB_0_ADDR : IN STD_LOGIC_VECTOR ( 63 DOWNTO 0 );
  BRAM_PORTB_0_CLK : IN STD_LOGIC;
  BRAM_PORTB_0_DIN : IN STD_LOGIC_VECTOR ( 63 DOWNTO 0 );
  BRAM_PORTB_0_DOUT : OUT STD_LOGIC_VECTOR ( 63 DOWNTO 0 );
  BRAM_PORTB_0_WE : IN STD_LOGIC);
END COMPONENT;
SIGNAL A: STD_LOGIC := '1';
SIGNAL B: STD_LOGIC := '1';
SIGNAL C: STD_LOGIC_VECTOR(63 DOWNTO 0) :=
B"1101010001111110111111111101111111001001001111110100111111110011";
SIGNAL ABOVE_VECTOR:  STD_LOGIC_VECTOR(63 DOWNTO 0) := (OTHERS =>
'0');
SIGNAL BELOW_VECTOR:  STD_LOGIC_VECTOR(63 DOWNTO 0) := (OTHERS =>
'0');
SIGNAL BRAM_PORTA_0_ADDR : STD_LOGIC_VECTOR ( 5 DOWNTO
0 ) := B"000000";
SIGNAL BRAM_PORTA_0_CLK : STD_LOGIC := '0';
SIGNAL BRAM_PORTA_0_DIN : STD_LOGIC_VECTOR ( 63 DOWNTO 0 ) :=
B"1101010001111110111111111101111111001001001111110100111111110011";
SIGNAL BRAM_PORTA_0_DOUT : STD_LOGIC_VECTOR ( 63 DOWNTO 0 );
SIGNAL BRAM_PORTA_0_WE : STD_LOGIC := '0';
SIGNAL BRAM_PORTB_0_ADDR : STD_LOGIC_VECTOR ( 5 DOWNTO
0 ) := B"000000";
SIGNAL BRAM_PORTB_0_CLK : STD_LOGIC := '0';
SIGNAL BRAM_PORTB_0_DIN : STD_LOGIC_VECTOR ( 63 DOWNTO 0 );
SIGNAL BRAM_PORTB_0_DOUT : STD_LOGIC_VECTOR ( 63 DOWNTO 0 );
SIGNAL BRAM_PORTB_0_WE : STD_LOGIC := '1';

BEGIN
  BRAM_I: COMPONENT BRAM_WRAPPER
   PORT MAP (
    BRAM_PORTA_0_ADDR(63 DOWNTO 0) => BRAM_PORTA_0_ADDR(5
DOWNTO 0),
    BRAM_PORTA_0_CLK => BRAM_PORTA_0_CLK,
    BRAM_PORTA_0_DIN(63 DOWNTO 0) => BRAM_PORTA_0_DIN(63
DOWNTO 0),
    BRAM_PORTA_0_DOUT(63 DOWNTO 0) => BRAM_PORTA_0_DOUT(63
DOWNTO 0),
    BRAM_PORTA_0_WE => BRAM_PORTA_0_WE,
```

```
    BRAM_PORTB_0_ADDR(63 DOWNTO 0) => BRAM_PORTB_0_ADDR(5 DOWNTO 0),
    BRAM_PORTB_0_CLK => BRAM_PORTB_0_CLK,
    BRAM_PORTB_0_DIN(63 DOWNTO 0) => BRAM_PORTB_0_DIN(63 DOWNTO 0),
    BRAM_PORTB_0_DOUT(63 DOWNTO 0) => BRAM_PORTB_0_DOUT(63 DOWNTO 0),
    BRAM_PORTB_0_WE => BRAM_PORTB_0_WE
  );
  U1: MYMUX2 PORT MAP (A, B, C(0), ABOVE_VECTOR(0));
  U2: MYMUX2 PORT MAP (B, A, C(0), BELOW_VECTOR(0));
  U3: MYMUX2 PORT MAP (ABOVE_VECTOR(0), BELOW_VECTOR(0), C(1), ABOVE_VECTOR(1));
  U4: MYMUX2 PORT MAP (BELOW_VECTOR(0), ABOVE_VECTOR(0), C(1), BELOW_VECTOR(1));
  U5: MYMUX2 PORT MAP (ABOVE_VECTOR(1), BELOW_VECTOR(1), C(2), ABOVE_VECTOR(2));
  U6: MYMUX2 PORT MAP (BELOW_VECTOR(1), ABOVE_VECTOR(1), C(2), BELOW_VECTOR(2));
  U7: MYMUX2 PORT MAP (ABOVE_VECTOR(2), BELOW_VECTOR(2), C(3), ABOVE_VECTOR(3));
  U8: MYMUX2 PORT MAP (BELOW_VECTOR(2), ABOVE_VECTOR(2), C(3), BELOW_VECTOR(3));
  U9: MYMUX2 PORT MAP (ABOVE_VECTOR(3), BELOW_VECTOR(3), C(4), ABOVE_VECTOR(4));
  U10: MYMUX2 PORT MAP (BELOW_VECTOR(3), ABOVE_VECTOR(3), C(4), BELOW_VECTOR(4));
  U11: MYMUX2 PORT MAP (ABOVE_VECTOR(4), BELOW_VECTOR(4), C(5), ABOVE_VECTOR(5));
  U12: MYMUX2 PORT MAP (BELOW_VECTOR(4), ABOVE_VECTOR(4), C(5), BELOW_VECTOR(5));
  U13: MYMUX2 PORT MAP (ABOVE_VECTOR(5), BELOW_VECTOR(5), C(6), ABOVE_VECTOR(6));
  U14: MYMUX2 PORT MAP (BELOW_VECTOR(5), ABOVE_VECTOR(5), C(6), BELOW_VECTOR(6));
  U15: MYMUX2 PORT MAP (ABOVE_VECTOR(6), BELOW_VECTOR(6), C(7), ABOVE_VECTOR(7));
  U16: MYMUX2 PORT MAP (BELOW_VECTOR(6), ABOVE_VECTOR(6), C(7), BELOW_VECTOR(7));
  U17: MYMUX2 PORT MAP (ABOVE_VECTOR(7), BELOW_VECTOR(7), C(8), ABOVE_VECTOR(8));
  U18: MYMUX2 PORT MAP (BELOW_VECTOR(7), ABOVE_VECTOR(7), C(8), BELOW_VECTOR(8));
  U19: MYMUX2 PORT MAP (ABOVE_VECTOR(8), BELOW_VECTOR(8), C(9), ABOVE_VECTOR(9));
  U20: MYMUX2 PORT MAP (BELOW_VECTOR(8), ABOVE_VECTOR(8), C(9), BELOW_VECTOR(9));
  U21: MYMUX2 PORT MAP (ABOVE_VECTOR(9), BELOW_VECTOR(9), C(10), ABOVE_VECTOR(10));
  U22: MYMUX2 PORT MAP (BELOW_VECTOR(9), ABOVE_VECTOR(9), C(10), BELOW_VECTOR(10));
  U23: MYMUX2 PORT MAP (ABOVE_VECTOR(10), BELOW_VECTOR(10), C(11), ABOVE_VECTOR(11));
  U24: MYMUX2 PORT MAP (BELOW_VECTOR(10), ABOVE_VECTOR(10), C(11), BELOW_VECTOR(11));
  U25: MYMUX2 PORT MAP (ABOVE_VECTOR(11), BELOW_VECTOR(11), C(12), ABOVE_VECTOR(12));
  U26: MYMUX2 PORT MAP (BELOW_VECTOR(11), ABOVE_VECTOR(11), C(12), BELOW_VECTOR(12));
  U27: MYMUX2 PORT MAP (ABOVE_VECTOR(12), BELOW_VECTOR(12), C(13), ABOVE_VECTOR(13));
  U28: MYMUX2 PORT MAP (BELOW_VECTOR(12), ABOVE_VECTOR(12), C(13), BELOW_VECTOR(13));
  U29: MYMUX2 PORT MAP (ABOVE_VECTOR(13), BELOW_VECTOR(13), C(14), ABOVE_VECTOR(14));
  U30: MYMUX2 PORT MAP (BELOW_VECTOR(13), ABOVE_VECTOR(13), C(14), BELOW_VECTOR(14));
  U31: MYMUX2 PORT MAP (ABOVE_VECTOR(14), BELOW_VECTOR(14), C(15), ABOVE_VECTOR(15));
  U32: MYMUX2 PORT MAP (BELOW_VECTOR(14), ABOVE_VECTOR(14), C(15), BELOW_VECTOR(15));
  U33: MYMUX2 PORT MAP (ABOVE_VECTOR(15), BELOW_VECTOR(15), C(16), ABOVE_VECTOR(16));
  U34: MYMUX2 PORT MAP (BELOW_VECTOR(15), ABOVE_VECTOR(15), C(16), BELOW_VECTOR(16));
  U35: MYMUX2 PORT MAP (ABOVE_VECTOR(16), BELOW_VECTOR(16), C(17), ABOVE_VECTOR(17));
  U36: MYMUX2 PORT MAP (BELOW_VECTOR(16), ABOVE_VECTOR(16), C(17), BELOW_VECTOR(17));
  U37: MYMUX2 PORT MAP (ABOVE_VECTOR(17), BELOW_VECTOR(17), C(18), ABOVE_VECTOR(18));
  U38: MYMUX2 PORT MAP (BELOW_VECTOR(17), ABOVE_VECTOR(17), C(18), BELOW_VECTOR(18));
  U39: MYMUX2 PORT MAP (ABOVE_VECTOR(18), BELOW_VECTOR(18), C(19), ABOVE_VECTOR(19));
  U40: MYMUX2 PORT MAP (BELOW_VECTOR(18), ABOVE_VECTOR(18), C(19), BELOW_VECTOR(19));
  U41: MYMUX2 PORT MAP (ABOVE_VECTOR(19), BELOW_VECTOR(19), C(20), ABOVE_VECTOR(20));
  U42: MYMUX2 PORT MAP (BELOW_VECTOR(19), ABOVE_VECTOR(19), C(20), BELOW_VECTOR(20));
  U43: MYMUX2 PORT MAP (ABOVE_VECTOR(20), BELOW_VECTOR(20), C(21), ABOVE_VECTOR(21));
  U44: MYMUX2 PORT MAP (BELOW_VECTOR(20), ABOVE_VECTOR(20), C(21), BELOW_VECTOR(21));
  U45: MYMUX2 PORT MAP (ABOVE_VECTOR(21), BELOW_VECTOR(21), C(22), ABOVE_VECTOR(22));
  U46: MYMUX2 PORT MAP (BELOW_VECTOR(21), ABOVE_VECTOR(21), C(22), BELOW_VECTOR(22));
  U47: MYMUX2 PORT MAP (ABOVE_VECTOR(22), BELOW_VECTOR(22), C(23), ABOVE_VECTOR(23));
  U48: MYMUX2 PORT MAP (BELOW_VECTOR(22), ABOVE_VECTOR(22), C(23), BELOW_VECTOR(23));
  U49: MYMUX2 PORT MAP (ABOVE_VECTOR(23), BELOW_VECTOR(23), C(24), ABOVE_VECTOR(24));
  U50: MYMUX2 PORT MAP (BELOW_VECTOR(23), ABOVE_VECTOR(23), C(24), BELOW_VECTOR(24));
  U51: MYMUX2 PORT MAP (ABOVE_VECTOR(24), BELOW_VECTOR(24), C(25), ABOVE_VECTOR(25));
  U52: MYMUX2 PORT MAP (BELOW_VECTOR(24), ABOVE_VECTOR(24), C(25), BELOW_VECTOR(25));
  U53: MYMUX2 PORT MAP (ABOVE_VECTOR(25), BELOW_VECTOR(25), C(26), ABOVE_VECTOR(26));
  U54: MYMUX2 PORT MAP (BELOW_VECTOR(25), ABOVE_VECTOR(25), C(26), BELOW_VECTOR(26));
  U55: MYMUX2 PORT MAP (ABOVE_VECTOR(26), BELOW_VECTOR(26), C(27), ABOVE_VECTOR(27));
  U56: MYMUX2 PORT MAP (BELOW_VECTOR(26), ABOVE_VECTOR(26), C(27), BELOW_VECTOR(27));
  U57: MYMUX2 PORT MAP (ABOVE_VECTOR(27), BELOW_VECTOR(27), C(28), ABOVE_VECTOR(28));
  U58: MYMUX2 PORT MAP (BELOW_VECTOR(27), ABOVE_VECTOR(27), C(28), BELOW_VECTOR(28));
  U59: MYMUX2 PORT MAP (ABOVE_VECTOR(28), BELOW_VECTOR(28), C(29), ABOVE_VECTOR(29));
  U60: MYMUX2 PORT MAP (BELOW_VECTOR(28), ABOVE_VECTOR(28), C(29), BELOW_VECTOR(29));
  U61: MYMUX2 PORT MAP (ABOVE_VECTOR(29), BELOW_VECTOR(29), C(30), ABOVE_VECTOR(30));
  U62: MYMUX2 PORT MAP (BELOW_VECTOR(29), ABOVE_VECTOR(29), C(30), BELOW_VECTOR(30));
  U63: MYMUX2 PORT MAP (ABOVE_VECTOR(30), BELOW_VECTOR(30), C(31), ABOVE_VECTOR(31));
  U64: MYMUX2 PORT MAP (BELOW_VECTOR(30), ABOVE_VECTOR(30), C(31), BELOW_VECTOR(31));
  U65: MYMUX2 PORT MAP (ABOVE_VECTOR(31), BELOW_VECTOR(31), C(32), ABOVE_VECTOR(32));
  U66: MYMUX2 PORT MAP (BELOW_VECTOR(31), ABOVE_VECTOR(31), C(32), BELOW_VECTOR(32));
  U67: MYMUX2 PORT MAP (ABOVE_VECTOR(32), BELOW_VECTOR(32), C(33), ABOVE_VECTOR(33));
  U68: MYMUX2 PORT MAP (BELOW_VECTOR(32), ABOVE_VECTOR(32), C(33), BELOW_VECTOR(33));
  U69: MYMUX2 PORT MAP (ABOVE_VECTOR(33), BELOW_VECTOR(33), C(34), ABOVE_VECTOR(34));
  U70: MYMUX2 PORT MAP (BELOW_VECTOR(33), ABOVE_VECTOR(33), C(34), BELOW_VECTOR(34));
```

```
   U71: MYMUX2 PORT MAP (ABOVE_VECTOR(34), BELOW_VECTOR(34),
C(35), ABOVE_VECTOR(35));
   U72: MYMUX2 PORT MAP (BELOW_VECTOR(34), ABOVE_VECTOR(34),
C(35), BELOW_VECTOR(35));
   U73: MYMUX2 PORT MAP (ABOVE_VECTOR(35), BELOW_VECTOR(35),
C(36), ABOVE_VECTOR(36));
   U74: MYMUX2 PORT MAP (BELOW_VECTOR(35), ABOVE_VECTOR(35),
C(36), BELOW_VECTOR(36));
   U75: MYMUX2 PORT MAP (ABOVE_VECTOR(36), BELOW_VECTOR(36),
C(37), ABOVE_VECTOR(37));
   U76: MYMUX2 PORT MAP (BELOW_VECTOR(36), ABOVE_VECTOR(36),
C(37), BELOW_VECTOR(37));
   U77: MYMUX2 PORT MAP (ABOVE_VECTOR(37), BELOW_VECTOR(37),
C(38), ABOVE_VECTOR(38));
   U78: MYMUX2 PORT MAP (BELOW_VECTOR(37), ABOVE_VECTOR(37),
C(38), BELOW_VECTOR(38));
   U79: MYMUX2 PORT MAP (ABOVE_VECTOR(38), BELOW_VECTOR(38),
C(39), ABOVE_VECTOR(39));
   U80: MYMUX2 PORT MAP (BELOW_VECTOR(38), ABOVE_VECTOR(38),
C(39), BELOW_VECTOR(39));
   U81: MYMUX2 PORT MAP (ABOVE_VECTOR(39), BELOW_VECTOR(39),
C(40), ABOVE_VECTOR(40));
   U82: MYMUX2 PORT MAP (BELOW_VECTOR(39), ABOVE_VECTOR(39),
C(40), BELOW_VECTOR(40));
   U83: MYMUX2 PORT MAP (ABOVE_VECTOR(40), BELOW_VECTOR(40),
C(41), ABOVE_VECTOR(41));
   U84: MYMUX2 PORT MAP (BELOW_VECTOR(40), ABOVE_VECTOR(40),
C(41), BELOW_VECTOR(41));
   U85: MYMUX2 PORT MAP (ABOVE_VECTOR(41), BELOW_VECTOR(41),
C(42), ABOVE_VECTOR(42));
   U86: MYMUX2 PORT MAP (BELOW_VECTOR(41), ABOVE_VECTOR(41),
C(42), BELOW_VECTOR(42));
   U87: MYMUX2 PORT MAP (ABOVE_VECTOR(42), BELOW_VECTOR(42),
C(43), ABOVE_VECTOR(43));
   U88: MYMUX2 PORT MAP (BELOW_VECTOR(42), ABOVE_VECTOR(42),
C(43), BELOW_VECTOR(43));
   U89: MYMUX2 PORT MAP (ABOVE_VECTOR(43), BELOW_VECTOR(43),
C(44), ABOVE_VECTOR(44));
   U90: MYMUX2 PORT MAP (BELOW_VECTOR(43), ABOVE_VECTOR(43),
C(44), BELOW_VECTOR(44));
   U91: MYMUX2 PORT MAP (ABOVE_VECTOR(44), BELOW_VECTOR(44),
C(45), ABOVE_VECTOR(45));
   U92: MYMUX2 PORT MAP (BELOW_VECTOR(44), ABOVE_VECTOR(44),
C(45), BELOW_VECTOR(45));
   U93: MYMUX2 PORT MAP (ABOVE_VECTOR(45), BELOW_VECTOR(45),
C(46), ABOVE_VECTOR(46));
   U94: MYMUX2 PORT MAP (BELOW_VECTOR(45), ABOVE_VECTOR(45),
C(46), BELOW_VECTOR(46));
   U95: MYMUX2 PORT MAP (ABOVE_VECTOR(46), BELOW_VECTOR(46),
C(47), ABOVE_VECTOR(47));
   U96: MYMUX2 PORT MAP (BELOW_VECTOR(46), ABOVE_VECTOR(46),
C(47), BELOW_VECTOR(47));
   U97: MYMUX2 PORT MAP (ABOVE_VECTOR(47), BELOW_VECTOR(47),
C(48), ABOVE_VECTOR(48));
   U98: MYMUX2 PORT MAP (BELOW_VECTOR(47), ABOVE_VECTOR(47),
C(48), BELOW_VECTOR(48));
   U99: MYMUX2 PORT MAP (ABOVE_VECTOR(48), BELOW_VECTOR(48),
C(49), ABOVE_VECTOR(49));
   U100: MYMUX2 PORT MAP (BELOW_VECTOR(48), ABOVE_VECTOR(48),
C(49), BELOW_VECTOR(49));
   U101: MYMUX2 PORT MAP (ABOVE_VECTOR(49), BELOW_VECTOR(49),
C(50), ABOVE_VECTOR(50));
   U102: MYMUX2 PORT MAP (BELOW_VECTOR(49), ABOVE_VECTOR(49),
C(50), BELOW_VECTOR(50));
   U103: MYMUX2 PORT MAP (ABOVE_VECTOR(50), BELOW_VECTOR(50),
C(51), ABOVE_VECTOR(51));
   U104: MYMUX2 PORT MAP (BELOW_VECTOR(50), ABOVE_VECTOR(50),
C(51), BELOW_VECTOR(51));
   U105: MYMUX2 PORT MAP (ABOVE_VECTOR(51), BELOW_VECTOR(51),
C(52), ABOVE_VECTOR(52));
   U106: MYMUX2 PORT MAP (BELOW_VECTOR(51), ABOVE_VECTOR(51),
C(52), BELOW_VECTOR(52));
   U107: MYMUX2 PORT MAP (ABOVE_VECTOR(52), BELOW_VECTOR(52),
C(53), ABOVE_VECTOR(53));
   U108: MYMUX2 PORT MAP (BELOW_VECTOR(52), ABOVE_VECTOR(52),
C(53), BELOW_VECTOR(53));
   U109: MYMUX2 PORT MAP (ABOVE_VECTOR(53), BELOW_VECTOR(53),
C(54), ABOVE_VECTOR(54));
   U110: MYMUX2 PORT MAP (BELOW_VECTOR(53), ABOVE_VECTOR(53),
C(54), BELOW_VECTOR(54));
   U111: MYMUX2 PORT MAP (ABOVE_VECTOR(54), BELOW_VECTOR(54),
C(55), ABOVE_VECTOR(55));
   U112: MYMUX2 PORT MAP (BELOW_VECTOR(54), ABOVE_VECTOR(54),
C(55), BELOW_VECTOR(55));
   U113: MYMUX2 PORT MAP (ABOVE_VECTOR(55), BELOW_VECTOR(55),
C(56), ABOVE_VECTOR(56));
   U114: MYMUX2 PORT MAP (BELOW_VECTOR(55), ABOVE_VECTOR(55),
C(56), BELOW_VECTOR(56));
   U115: MYMUX2 PORT MAP (ABOVE_VECTOR(56), BELOW_VECTOR(56),
C(57), ABOVE_VECTOR(57));
   U116: MYMUX2 PORT MAP (BELOW_VECTOR(56), ABOVE_VECTOR(56),
C(57), BELOW_VECTOR(57));
   U117: MYMUX2 PORT MAP (ABOVE_VECTOR(57), BELOW_VECTOR(57),
C(58), ABOVE_VECTOR(58));
   U118: MYMUX2 PORT MAP (BELOW_VECTOR(57), ABOVE_VECTOR(57),
C(58), BELOW_VECTOR(58));
   U119: MYMUX2 PORT MAP (ABOVE_VECTOR(58), BELOW_VECTOR(58),
C(59), ABOVE_VECTOR(59));
   U120: MYMUX2 PORT MAP (BELOW_VECTOR(58), ABOVE_VECTOR(58),
C(59), BELOW_VECTOR(59));
   U121: MYMUX2 PORT MAP (ABOVE_VECTOR(59), BELOW_VECTOR(59),
C(60), ABOVE_VECTOR(60));
   U122: MYMUX2 PORT MAP (BELOW_VECTOR(59), ABOVE_VECTOR(59),
C(60), BELOW_VECTOR(60));
   U123: MYMUX2 PORT MAP (ABOVE_VECTOR(60), BELOW_VECTOR(60),
C(61), ABOVE_VECTOR(61));
   U124: MYMUX2 PORT MAP (BELOW_VECTOR(60), ABOVE_VECTOR(60),
C(61), BELOW_VECTOR(61));
   U125: MYMUX2 PORT MAP (ABOVE_VECTOR(61), BELOW_VECTOR(61),
C(62), ABOVE_VECTOR(62));
   U126: MYMUX2 PORT MAP (BELOW_VECTOR(61), ABOVE_VECTOR(61),
C(62), BELOW_VECTOR(62));
   U127: MYMUX2 PORT MAP (ABOVE_VECTOR(62), BELOW_VECTOR(62),
C(63), ABOVE_VECTOR(63));
   U128: MYMUX2 PORT MAP (BELOW_VECTOR(62), ABOVE_VECTOR(62),
C(63), BELOW_VECTOR(63));
   U129: DDF   PORT MAP (Q, BELOW_VECTOR(63), ABOVE_VECTOR(63));
   BRAM_PORTA_0_CLK <=  '1' AFTER 100 NS WHEN
BRAM_PORTA_0_CLK = '0' ELSE
               '0' AFTER 100 NS WHEN BRAM_PORTA_0_CLK = '1';
   BRAM_PORTB_0_CLK <=  '1' AFTER 100 NS WHEN
BRAM_PORTB_0_CLK = '0' ELSE
               '0' AFTER 100 NS WHEN BRAM_PORTB_0_CLK = '1';
   CLK <= BELOW_VECTOR(63);
   DIN <= ABOVE_VECTOR(63);
   BRAM_PORTB_0_DIN(0) <= Q;
END BEHAVIORAL;
```

MYMUX2.VHD

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MYMUX2 IS
   PORT ( A : IN STD_LOGIC;
        B : IN STD_LOGIC;
        S : IN STD_LOGIC;
        Y : OUT STD_LOGIC);
END MYMUX2;

ARCHITECTURE BEHAVIORAL OF MYMUX2 IS

BEGIN
   Y <= ((NOT S) AND A) OR (S AND B);

END BEHAVIORAL;
```

DFF.VHD

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY DDF IS
  PORT(
    Q : OUT STD_LOGIC;
    CLK :IN STD_LOGIC;
    D :IN  STD_LOGIC
  );
END DDF;
ARCHITECTURE BEHAVIORAL OF DDF IS
BEGIN
 PROCESS(CLK)
 BEGIN
 IF (CLK'EVENT AND CLK = '1') THEN
   IF(D='1' OR D='0') THEN
     Q <= D;
   ELSE
     Q <= '0';
   END IF;
 END IF;
 END PROCESS;
END BEHAVIORAL;
```

XOR_APUF.VHD

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY XOR_APUF IS
   PORT ( --A :     IN STD_LOGIC;
        --B :     IN STD_LOGIC;
        --C : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
        Q :     OUT STD_LOGIC;
        PATH1:  OUT STD_LOGIC;
        PATH2:  OUT STD_LOGIC);
END XOR_APUF;

ARCHITECTURE BEHAVIORAL OF XOR_APUF IS
COMPONENT MYMUX2
PORT (   A : IN STD_LOGIC;
       B : IN STD_LOGIC;
       S : IN STD_LOGIC;
       Y : OUT STD_LOGIC);
END COMPONENT;

COMPONENT XOR_GATE
PORT (   A : IN STD_LOGIC;
       B : IN STD_LOGIC;
       Y : OUT STD_LOGIC);
END COMPONENT;

COMPONENT DDF
PORT(
       Q : OUT STD_LOGIC;
       CLK :IN STD_LOGIC;
       D :IN  STD_LOGIC);
END COMPONENT;

SIGNAL A: STD_LOGIC := '1';
SIGNAL B: STD_LOGIC := '1';
SIGNAL C: STD_LOGIC_VECTOR(63 DOWNTO 0) :=
B"1101010001111110111111111110111111100100100111111010011111111
0011";
SIGNAL ABOVE_VECTOR:  STD_LOGIC_VECTOR(63 DOWNTO 0) := (OTHERS =>
'0');
SIGNAL BELOW_VECTOR:  STD_LOGIC_VECTOR(63 DOWNTO 0) := (OTHERS =>
'0');
SIGNAL ABOVE_VECTOR_2:  STD_LOGIC_VECTOR(63 DOWNTO 0) := (OTHERS
=> '0');
SIGNAL BELOW_VECTOR_2:  STD_LOGIC_VECTOR(63 DOWNTO 0) := (OTHERS
=> '0');
SIGNAL DFF_OUTPUT:    STD_LOGIC_VECTOR(1 DOWNTO 0) := (OTHERS => '0');
BEGIN
   U1: MYMUX2 PORT MAP (A, A, C(0), ABOVE_VECTOR(0));
   U2: MYMUX2 PORT MAP (A, A, C(0), BELOW_VECTOR(0));
   U3: MYMUX2 PORT MAP (ABOVE_VECTOR(0), BELOW_VECTOR(0), C(1),
ABOVE_VECTOR(1));
   U4: MYMUX2 PORT MAP (BELOW_VECTOR(0), ABOVE_VECTOR(0), C(1),
BELOW_VECTOR(1));
   U5: MYMUX2 PORT MAP (ABOVE_VECTOR(1), BELOW_VECTOR(1), C(2),
ABOVE_VECTOR(2));
   U6: MYMUX2 PORT MAP (BELOW_VECTOR(1), ABOVE_VECTOR(1), C(2),
BELOW_VECTOR(2));
   U7: MYMUX2 PORT MAP (ABOVE_VECTOR(2), BELOW_VECTOR(2), C(3),
ABOVE_VECTOR(3));
   U8: MYMUX2 PORT MAP (BELOW_VECTOR(2), ABOVE_VECTOR(2), C(3),
BELOW_VECTOR(3));
   U9: MYMUX2 PORT MAP (ABOVE_VECTOR(3), BELOW_VECTOR(3), C(4),
ABOVE_VECTOR(4));
   U10: MYMUX2 PORT MAP (BELOW_VECTOR(3), ABOVE_VECTOR(3), C(4),
BELOW_VECTOR(4));
   U11: MYMUX2 PORT MAP (ABOVE_VECTOR(4), BELOW_VECTOR(4), C(5),
ABOVE_VECTOR(5));
   U12: MYMUX2 PORT MAP (BELOW_VECTOR(4), ABOVE_VECTOR(4), C(5),
BELOW_VECTOR(5));
   U13: MYMUX2 PORT MAP (ABOVE_VECTOR(5), BELOW_VECTOR(5), C(6),
ABOVE_VECTOR(6));
   U14: MYMUX2 PORT MAP (BELOW_VECTOR(5), ABOVE_VECTOR(5), C(6),
BELOW_VECTOR(6));
   U15: MYMUX2 PORT MAP (ABOVE_VECTOR(6), BELOW_VECTOR(6), C(7),
ABOVE_VECTOR(7));
   U16: MYMUX2 PORT MAP (BELOW_VECTOR(6), ABOVE_VECTOR(6), C(7),
BELOW_VECTOR(7));
   U17: MYMUX2 PORT MAP (ABOVE_VECTOR(7), BELOW_VECTOR(7), C(8),
ABOVE_VECTOR(8));
   U18: MYMUX2 PORT MAP (BELOW_VECTOR(7), ABOVE_VECTOR(7), C(8),
BELOW_VECTOR(8));
   U19: MYMUX2 PORT MAP (ABOVE_VECTOR(8), BELOW_VECTOR(8), C(9),
ABOVE_VECTOR(9));
   U20: MYMUX2 PORT MAP (BELOW_VECTOR(8), ABOVE_VECTOR(8), C(9),
BELOW_VECTOR(9));
   U21: MYMUX2 PORT MAP (ABOVE_VECTOR(9), BELOW_VECTOR(9), C(10),
ABOVE_VECTOR(10));
   U22: MYMUX2 PORT MAP (BELOW_VECTOR(9), ABOVE_VECTOR(9), C(10),
BELOW_VECTOR(10));
   U23: MYMUX2 PORT MAP (ABOVE_VECTOR(10), BELOW_VECTOR(10),
C(11), ABOVE_VECTOR(11));
   U24: MYMUX2 PORT MAP (BELOW_VECTOR(10), ABOVE_VECTOR(10),
C(11), BELOW_VECTOR(11));
   U25: MYMUX2 PORT MAP (ABOVE_VECTOR(11), BELOW_VECTOR(11),
C(12), ABOVE_VECTOR(12));
   U26: MYMUX2 PORT MAP (BELOW_VECTOR(11), ABOVE_VECTOR(11),
C(12), BELOW_VECTOR(12));
   U27: MYMUX2 PORT MAP (ABOVE_VECTOR(12), BELOW_VECTOR(12),
C(13), ABOVE_VECTOR(13));
   U28: MYMUX2 PORT MAP (BELOW_VECTOR(12), ABOVE_VECTOR(12),
C(13), BELOW_VECTOR(13));
   U29: MYMUX2 PORT MAP (ABOVE_VECTOR(13), BELOW_VECTOR(13),
C(14), ABOVE_VECTOR(14));
   U30: MYMUX2 PORT MAP (BELOW_VECTOR(13), ABOVE_VECTOR(13),
C(14), BELOW_VECTOR(14));
   U31: MYMUX2 PORT MAP (ABOVE_VECTOR(14), BELOW_VECTOR(14),
C(15), ABOVE_VECTOR(15));
   U32: MYMUX2 PORT MAP (BELOW_VECTOR(14), ABOVE_VECTOR(14),
C(15), BELOW_VECTOR(15));
   U33: MYMUX2 PORT MAP (ABOVE_VECTOR(15), BELOW_VECTOR(15),
C(16), ABOVE_VECTOR(16));
   U34: MYMUX2 PORT MAP (BELOW_VECTOR(15), ABOVE_VECTOR(15),
C(16), BELOW_VECTOR(16));
   U35: MYMUX2 PORT MAP (ABOVE_VECTOR(16), BELOW_VECTOR(16),
C(17), ABOVE_VECTOR(17));
   U36: MYMUX2 PORT MAP (BELOW_VECTOR(16), ABOVE_VECTOR(16),
C(17), BELOW_VECTOR(17));
   U37: MYMUX2 PORT MAP (ABOVE_VECTOR(17), BELOW_VECTOR(17),
C(18), ABOVE_VECTOR(18));
   U38: MYMUX2 PORT MAP (BELOW_VECTOR(17), ABOVE_VECTOR(17),
C(18), BELOW_VECTOR(18));
```

U39: MYMUX2 PORT MAP (ABOVE_VECTOR(18), BELOW_VECTOR(18), C(19), ABOVE_VECTOR(19));
U40: MYMUX2 PORT MAP (BELOW_VECTOR(18), ABOVE_VECTOR(18), C(19), BELOW_VECTOR(19));
U41: MYMUX2 PORT MAP (ABOVE_VECTOR(19), BELOW_VECTOR(19), C(20), ABOVE_VECTOR(20));
U42: MYMUX2 PORT MAP (BELOW_VECTOR(19), ABOVE_VECTOR(19), C(20), BELOW_VECTOR(20));
U43: MYMUX2 PORT MAP (ABOVE_VECTOR(20), BELOW_VECTOR(20), C(21), ABOVE_VECTOR(21));
U44: MYMUX2 PORT MAP (BELOW_VECTOR(20), ABOVE_VECTOR(20), C(21), BELOW_VECTOR(21));
U45: MYMUX2 PORT MAP (ABOVE_VECTOR(21), BELOW_VECTOR(21), C(22), ABOVE_VECTOR(22));
U46: MYMUX2 PORT MAP (BELOW_VECTOR(21), ABOVE_VECTOR(21), C(22), BELOW_VECTOR(22));
U47: MYMUX2 PORT MAP (ABOVE_VECTOR(22), BELOW_VECTOR(22), C(23), ABOVE_VECTOR(23));
U48: MYMUX2 PORT MAP (BELOW_VECTOR(22), ABOVE_VECTOR(22), C(23), BELOW_VECTOR(23));
U49: MYMUX2 PORT MAP (ABOVE_VECTOR(23), BELOW_VECTOR(23), C(24), ABOVE_VECTOR(24));
U50: MYMUX2 PORT MAP (BELOW_VECTOR(23), ABOVE_VECTOR(23), C(24), BELOW_VECTOR(24));
U51: MYMUX2 PORT MAP (ABOVE_VECTOR(24), BELOW_VECTOR(24), C(25), ABOVE_VECTOR(25));
U52: MYMUX2 PORT MAP (BELOW_VECTOR(24), ABOVE_VECTOR(24), C(25), BELOW_VECTOR(25));
U53: MYMUX2 PORT MAP (ABOVE_VECTOR(25), BELOW_VECTOR(25), C(26), ABOVE_VECTOR(26));
U54: MYMUX2 PORT MAP (BELOW_VECTOR(25), ABOVE_VECTOR(25), C(26), BELOW_VECTOR(26));
U55: MYMUX2 PORT MAP (ABOVE_VECTOR(26), BELOW_VECTOR(26), C(27), ABOVE_VECTOR(27));
U56: MYMUX2 PORT MAP (BELOW_VECTOR(26), ABOVE_VECTOR(26), C(27), BELOW_VECTOR(27));
U57: MYMUX2 PORT MAP (ABOVE_VECTOR(27), BELOW_VECTOR(27), C(28), ABOVE_VECTOR(28));
U58: MYMUX2 PORT MAP (BELOW_VECTOR(27), ABOVE_VECTOR(27), C(28), BELOW_VECTOR(28));
U59: MYMUX2 PORT MAP (ABOVE_VECTOR(28), BELOW_VECTOR(28), C(29), ABOVE_VECTOR(29));
U60: MYMUX2 PORT MAP (BELOW_VECTOR(28), ABOVE_VECTOR(28), C(29), BELOW_VECTOR(29));
U61: MYMUX2 PORT MAP (ABOVE_VECTOR(29), BELOW_VECTOR(29), C(30), ABOVE_VECTOR(30));
U62: MYMUX2 PORT MAP (BELOW_VECTOR(29), ABOVE_VECTOR(29), C(30), BELOW_VECTOR(30));
U63: MYMUX2 PORT MAP (ABOVE_VECTOR(30), BELOW_VECTOR(30), C(31), ABOVE_VECTOR(31));
U64: MYMUX2 PORT MAP (BELOW_VECTOR(30), ABOVE_VECTOR(30), C(31), BELOW_VECTOR(31));
U65: MYMUX2 PORT MAP (ABOVE_VECTOR(31), BELOW_VECTOR(31), C(32), ABOVE_VECTOR(32));
U66: MYMUX2 PORT MAP (BELOW_VECTOR(31), ABOVE_VECTOR(31), C(32), BELOW_VECTOR(32));
U67: MYMUX2 PORT MAP (ABOVE_VECTOR(32), BELOW_VECTOR(32), C(33), ABOVE_VECTOR(33));
U68: MYMUX2 PORT MAP (BELOW_VECTOR(32), ABOVE_VECTOR(32), C(33), BELOW_VECTOR(33));
U69: MYMUX2 PORT MAP (ABOVE_VECTOR(33), BELOW_VECTOR(33), C(34), ABOVE_VECTOR(34));
U70: MYMUX2 PORT MAP (BELOW_VECTOR(33), ABOVE_VECTOR(33), C(34), BELOW_VECTOR(34));
U71: MYMUX2 PORT MAP (ABOVE_VECTOR(34), BELOW_VECTOR(34), C(35), ABOVE_VECTOR(35));
U72: MYMUX2 PORT MAP (BELOW_VECTOR(34), ABOVE_VECTOR(34), C(35), BELOW_VECTOR(35));
U73: MYMUX2 PORT MAP (ABOVE_VECTOR(35), BELOW_VECTOR(35), C(36), ABOVE_VECTOR(36));
U74: MYMUX2 PORT MAP (BELOW_VECTOR(35), ABOVE_VECTOR(35), C(36), BELOW_VECTOR(36));
U75: MYMUX2 PORT MAP (ABOVE_VECTOR(36), BELOW_VECTOR(36), C(37), ABOVE_VECTOR(37));
U76: MYMUX2 PORT MAP (BELOW_VECTOR(36), ABOVE_VECTOR(36), C(37), BELOW_VECTOR(37));
U77: MYMUX2 PORT MAP (ABOVE_VECTOR(37), BELOW_VECTOR(37), C(38), ABOVE_VECTOR(38));
U78: MYMUX2 PORT MAP (BELOW_VECTOR(37), ABOVE_VECTOR(37), C(38), BELOW_VECTOR(38));
U79: MYMUX2 PORT MAP (ABOVE_VECTOR(38), BELOW_VECTOR(38), C(39), ABOVE_VECTOR(39));
U80: MYMUX2 PORT MAP (BELOW_VECTOR(38), ABOVE_VECTOR(38), C(39), BELOW_VECTOR(39));
U81: MYMUX2 PORT MAP (ABOVE_VECTOR(39), BELOW_VECTOR(39), C(40), ABOVE_VECTOR(40));
U82: MYMUX2 PORT MAP (BELOW_VECTOR(39), ABOVE_VECTOR(39), C(40), BELOW_VECTOR(40));
U83: MYMUX2 PORT MAP (ABOVE_VECTOR(40), BELOW_VECTOR(40), C(41), ABOVE_VECTOR(41));
U84: MYMUX2 PORT MAP (BELOW_VECTOR(40), ABOVE_VECTOR(40), C(41), BELOW_VECTOR(41));
U85: MYMUX2 PORT MAP (ABOVE_VECTOR(41), BELOW_VECTOR(41), C(42), ABOVE_VECTOR(42));
U86: MYMUX2 PORT MAP (BELOW_VECTOR(41), ABOVE_VECTOR(41), C(42), BELOW_VECTOR(42));
U87: MYMUX2 PORT MAP (ABOVE_VECTOR(42), BELOW_VECTOR(42), C(43), ABOVE_VECTOR(43));
U88: MYMUX2 PORT MAP (BELOW_VECTOR(42), ABOVE_VECTOR(42), C(43), BELOW_VECTOR(43));
U89: MYMUX2 PORT MAP (ABOVE_VECTOR(43), BELOW_VECTOR(43), C(44), ABOVE_VECTOR(44));
U90: MYMUX2 PORT MAP (BELOW_VECTOR(43), ABOVE_VECTOR(43), C(44), BELOW_VECTOR(44));
U91: MYMUX2 PORT MAP (ABOVE_VECTOR(44), BELOW_VECTOR(44), C(45), ABOVE_VECTOR(45));
U92: MYMUX2 PORT MAP (BELOW_VECTOR(44), ABOVE_VECTOR(44), C(45), BELOW_VECTOR(45));
U93: MYMUX2 PORT MAP (ABOVE_VECTOR(45), BELOW_VECTOR(45), C(46), ABOVE_VECTOR(46));
U94: MYMUX2 PORT MAP (BELOW_VECTOR(45), ABOVE_VECTOR(45), C(46), BELOW_VECTOR(46));
U95: MYMUX2 PORT MAP (ABOVE_VECTOR(46), BELOW_VECTOR(46), C(47), ABOVE_VECTOR(47));
U96: MYMUX2 PORT MAP (BELOW_VECTOR(46), ABOVE_VECTOR(46), C(47), BELOW_VECTOR(47));
U97: MYMUX2 PORT MAP (ABOVE_VECTOR(47), BELOW_VECTOR(47), C(48), ABOVE_VECTOR(48));
U98: MYMUX2 PORT MAP (BELOW_VECTOR(47), ABOVE_VECTOR(47), C(48), BELOW_VECTOR(48));
U99: MYMUX2 PORT MAP (ABOVE_VECTOR(48), BELOW_VECTOR(48), C(49), ABOVE_VECTOR(49));
U100: MYMUX2 PORT MAP (BELOW_VECTOR(48), ABOVE_VECTOR(48), C(49), BELOW_VECTOR(49));
U101: MYMUX2 PORT MAP (ABOVE_VECTOR(49), BELOW_VECTOR(49), C(50), ABOVE_VECTOR(50));
U102: MYMUX2 PORT MAP (BELOW_VECTOR(49), ABOVE_VECTOR(49), C(50), BELOW_VECTOR(50));
U103: MYMUX2 PORT MAP (ABOVE_VECTOR(50), BELOW_VECTOR(50), C(51), ABOVE_VECTOR(51));
U104: MYMUX2 PORT MAP (BELOW_VECTOR(50), ABOVE_VECTOR(50), C(51), BELOW_VECTOR(51));
U105: MYMUX2 PORT MAP (ABOVE_VECTOR(51), BELOW_VECTOR(51), C(52), ABOVE_VECTOR(52));
U106: MYMUX2 PORT MAP (BELOW_VECTOR(51), ABOVE_VECTOR(51), C(52), BELOW_VECTOR(52));
U107: MYMUX2 PORT MAP (ABOVE_VECTOR(52), BELOW_VECTOR(52), C(53), ABOVE_VECTOR(53));
U108: MYMUX2 PORT MAP (BELOW_VECTOR(52), ABOVE_VECTOR(52), C(53), BELOW_VECTOR(53));
U109: MYMUX2 PORT MAP (ABOVE_VECTOR(53), BELOW_VECTOR(53), C(54), ABOVE_VECTOR(54));
U110: MYMUX2 PORT MAP (BELOW_VECTOR(53), ABOVE_VECTOR(53), C(54), BELOW_VECTOR(54));
U111: MYMUX2 PORT MAP (ABOVE_VECTOR(54), BELOW_VECTOR(54), C(55), ABOVE_VECTOR(55));
U112: MYMUX2 PORT MAP (BELOW_VECTOR(54), ABOVE_VECTOR(54), C(55), BELOW_VECTOR(55));

U113: MYMUX2 PORT MAP (ABOVE_VECTOR(55), BELOW_VECTOR(55), C(56), ABOVE_VECTOR(56));
U114: MYMUX2 PORT MAP (BELOW_VECTOR(55), ABOVE_VECTOR(55), C(56), BELOW_VECTOR(56));
U115: MYMUX2 PORT MAP (ABOVE_VECTOR(56), BELOW_VECTOR(56), C(57), ABOVE_VECTOR(57));
U116: MYMUX2 PORT MAP (BELOW_VECTOR(56), ABOVE_VECTOR(56), C(57), BELOW_VECTOR(57));
U117: MYMUX2 PORT MAP (ABOVE_VECTOR(57), BELOW_VECTOR(57), C(58), ABOVE_VECTOR(58));
U118: MYMUX2 PORT MAP (BELOW_VECTOR(57), ABOVE_VECTOR(57), C(58), BELOW_VECTOR(58));
U119: MYMUX2 PORT MAP (ABOVE_VECTOR(58), BELOW_VECTOR(58), C(59), ABOVE_VECTOR(59));
U120: MYMUX2 PORT MAP (BELOW_VECTOR(58), ABOVE_VECTOR(58), C(59), BELOW_VECTOR(59));
U121: MYMUX2 PORT MAP (ABOVE_VECTOR(59), BELOW_VECTOR(59), C(60), ABOVE_VECTOR(60));
U122: MYMUX2 PORT MAP (BELOW_VECTOR(59), ABOVE_VECTOR(59), C(60), BELOW_VECTOR(60));
U123: MYMUX2 PORT MAP (ABOVE_VECTOR(60), BELOW_VECTOR(60), C(61), ABOVE_VECTOR(61));
U124: MYMUX2 PORT MAP (BELOW_VECTOR(60), ABOVE_VECTOR(60), C(61), BELOW_VECTOR(61));
U125: MYMUX2 PORT MAP (ABOVE_VECTOR(61), BELOW_VECTOR(61), C(62), ABOVE_VECTOR(62));
U126: MYMUX2 PORT MAP (BELOW_VECTOR(61), ABOVE_VECTOR(61), C(62), BELOW_VECTOR(62));
U127: MYMUX2 PORT MAP (ABOVE_VECTOR(62), BELOW_VECTOR(62), C(63), ABOVE_VECTOR(63));
U128: MYMUX2 PORT MAP (BELOW_VECTOR(62), ABOVE_VECTOR(62), C(63), BELOW_VECTOR(63));
U129: DDF PORT MAP(DFF_OUTPUT(0), BELOW_VECTOR(63), ABOVE_VECTOR(63));

U130: MYMUX2 PORT MAP (B, B, C(0), ABOVE_VECTOR_2(0));
U131: MYMUX2 PORT MAP (B, B, C(0), BELOW_VECTOR_2(0));
U132: MYMUX2 PORT MAP (ABOVE_VECTOR_2(0), BELOW_VECTOR_2(0), C(1), ABOVE_VECTOR_2(1));
U133: MYMUX2 PORT MAP (BELOW_VECTOR_2(0), ABOVE_VECTOR_2(0), C(1), BELOW_VECTOR_2(1));
U134: MYMUX2 PORT MAP (ABOVE_VECTOR_2(1), BELOW_VECTOR_2(1), C(2), ABOVE_VECTOR_2(2));
U135: MYMUX2 PORT MAP (BELOW_VECTOR_2(1), ABOVE_VECTOR_2(1), C(2), BELOW_VECTOR_2(2));
U136: MYMUX2 PORT MAP (ABOVE_VECTOR_2(2), BELOW_VECTOR_2(2), C(3), ABOVE_VECTOR_2(3));
U137: MYMUX2 PORT MAP (BELOW_VECTOR_2(2), ABOVE_VECTOR_2(2), C(3), BELOW_VECTOR_2(3));
U138: MYMUX2 PORT MAP (ABOVE_VECTOR_2(3), BELOW_VECTOR_2(3), C(4), ABOVE_VECTOR_2(4));
U139: MYMUX2 PORT MAP (BELOW_VECTOR_2(3), ABOVE_VECTOR_2(3), C(4), BELOW_VECTOR_2(4));
U140: MYMUX2 PORT MAP (ABOVE_VECTOR_2(4), BELOW_VECTOR_2(4), C(5), ABOVE_VECTOR_2(5));
U141: MYMUX2 PORT MAP (BELOW_VECTOR_2(4), ABOVE_VECTOR_2(4), C(5), BELOW_VECTOR_2(5));
U142: MYMUX2 PORT MAP (ABOVE_VECTOR_2(5), BELOW_VECTOR_2(5), C(6), ABOVE_VECTOR_2(6));
U143: MYMUX2 PORT MAP (BELOW_VECTOR_2(5), ABOVE_VECTOR_2(5), C(6), BELOW_VECTOR_2(6));
U144: MYMUX2 PORT MAP (ABOVE_VECTOR_2(6), BELOW_VECTOR_2(6), C(7), ABOVE_VECTOR_2(7));
U145: MYMUX2 PORT MAP (BELOW_VECTOR_2(6), ABOVE_VECTOR_2(6), C(7), BELOW_VECTOR_2(7));
U146: MYMUX2 PORT MAP (ABOVE_VECTOR_2(7), BELOW_VECTOR_2(7), C(8), ABOVE_VECTOR_2(8));
U147: MYMUX2 PORT MAP (BELOW_VECTOR_2(7), ABOVE_VECTOR_2(7), C(8), BELOW_VECTOR_2(8));
U148: MYMUX2 PORT MAP (ABOVE_VECTOR_2(8), BELOW_VECTOR_2(8), C(9), ABOVE_VECTOR_2(9));
U149: MYMUX2 PORT MAP (BELOW_VECTOR_2(8), ABOVE_VECTOR_2(8), C(9), BELOW_VECTOR_2(9));
U150: MYMUX2 PORT MAP (ABOVE_VECTOR_2(9), BELOW_VECTOR_2(9), C(10), ABOVE_VECTOR_2(10));
U151: MYMUX2 PORT MAP (BELOW_VECTOR_2(9), ABOVE_VECTOR_2(9), C(10), BELOW_VECTOR_2(10));
U152: MYMUX2 PORT MAP (ABOVE_VECTOR_2(10), BELOW_VECTOR_2(10), C(11), ABOVE_VECTOR_2(11));
U153: MYMUX2 PORT MAP (BELOW_VECTOR_2(10), ABOVE_VECTOR_2(10), C(11), BELOW_VECTOR_2(11));
U154: MYMUX2 PORT MAP (ABOVE_VECTOR_2(11), BELOW_VECTOR_2(11), C(12), ABOVE_VECTOR_2(12));
U155: MYMUX2 PORT MAP (BELOW_VECTOR_2(11), ABOVE_VECTOR_2(11), C(12), BELOW_VECTOR_2(12));
U156: MYMUX2 PORT MAP (ABOVE_VECTOR_2(12), BELOW_VECTOR_2(12), C(13), ABOVE_VECTOR_2(13));
U157: MYMUX2 PORT MAP (BELOW_VECTOR_2(12), ABOVE_VECTOR_2(12), C(13), BELOW_VECTOR_2(13));
U158: MYMUX2 PORT MAP (ABOVE_VECTOR_2(13), BELOW_VECTOR_2(13), C(14), ABOVE_VECTOR_2(14));
U159: MYMUX2 PORT MAP (BELOW_VECTOR_2(13), ABOVE_VECTOR_2(13), C(14), BELOW_VECTOR_2(14));
U160: MYMUX2 PORT MAP (ABOVE_VECTOR_2(14), BELOW_VECTOR_2(14), C(15), ABOVE_VECTOR_2(15));
U161: MYMUX2 PORT MAP (BELOW_VECTOR_2(14), ABOVE_VECTOR_2(14), C(15), BELOW_VECTOR_2(15));
U162: MYMUX2 PORT MAP (ABOVE_VECTOR_2(15), BELOW_VECTOR_2(15), C(16), ABOVE_VECTOR_2(16));
U163: MYMUX2 PORT MAP (BELOW_VECTOR_2(15), ABOVE_VECTOR_2(15), C(16), BELOW_VECTOR_2(16));
U164: MYMUX2 PORT MAP (ABOVE_VECTOR_2(16), BELOW_VECTOR_2(16), C(17), ABOVE_VECTOR_2(17));
U165: MYMUX2 PORT MAP (BELOW_VECTOR_2(16), ABOVE_VECTOR_2(16), C(17), BELOW_VECTOR_2(17));
U166: MYMUX2 PORT MAP (ABOVE_VECTOR_2(17), BELOW_VECTOR_2(17), C(18), ABOVE_VECTOR_2(18));
U167: MYMUX2 PORT MAP (BELOW_VECTOR_2(17), ABOVE_VECTOR_2(17), C(18), BELOW_VECTOR_2(18));
U168: MYMUX2 PORT MAP (ABOVE_VECTOR_2(18), BELOW_VECTOR_2(18), C(19), ABOVE_VECTOR_2(19));
U169: MYMUX2 PORT MAP (BELOW_VECTOR_2(18), ABOVE_VECTOR_2(18), C(19), BELOW_VECTOR_2(19));
U170: MYMUX2 PORT MAP (ABOVE_VECTOR_2(19), BELOW_VECTOR_2(19), C(20), ABOVE_VECTOR_2(20));
U171: MYMUX2 PORT MAP (BELOW_VECTOR_2(19), ABOVE_VECTOR_2(19), C(20), BELOW_VECTOR_2(20));
U172: MYMUX2 PORT MAP (ABOVE_VECTOR_2(20), BELOW_VECTOR_2(20), C(21), ABOVE_VECTOR_2(21));
U173: MYMUX2 PORT MAP (BELOW_VECTOR_2(20), ABOVE_VECTOR_2(20), C(21), BELOW_VECTOR_2(21));
U174: MYMUX2 PORT MAP (ABOVE_VECTOR_2(21), BELOW_VECTOR_2(21), C(22), ABOVE_VECTOR_2(22));
U175: MYMUX2 PORT MAP (BELOW_VECTOR_2(21), ABOVE_VECTOR_2(21), C(22), BELOW_VECTOR_2(22));
U176: MYMUX2 PORT MAP (ABOVE_VECTOR_2(22), BELOW_VECTOR_2(22), C(23), ABOVE_VECTOR_2(23));
U177: MYMUX2 PORT MAP (BELOW_VECTOR_2(22), ABOVE_VECTOR_2(22), C(23), BELOW_VECTOR_2(23));
U178: MYMUX2 PORT MAP (ABOVE_VECTOR_2(23), BELOW_VECTOR_2(23), C(24), ABOVE_VECTOR_2(24));
U179: MYMUX2 PORT MAP (BELOW_VECTOR_2(23), ABOVE_VECTOR_2(23), C(24), BELOW_VECTOR_2(24));
U180: MYMUX2 PORT MAP (ABOVE_VECTOR_2(24), BELOW_VECTOR_2(24), C(25), ABOVE_VECTOR_2(25));
U181: MYMUX2 PORT MAP (BELOW_VECTOR_2(24), ABOVE_VECTOR_2(24), C(25), BELOW_VECTOR_2(25));
U182: MYMUX2 PORT MAP (ABOVE_VECTOR_2(25), BELOW_VECTOR_2(25), C(26), ABOVE_VECTOR_2(26));
U183: MYMUX2 PORT MAP (BELOW_VECTOR_2(25), ABOVE_VECTOR_2(25), C(26), BELOW_VECTOR_2(26));
U184: MYMUX2 PORT MAP (ABOVE_VECTOR_2(26), BELOW_VECTOR_2(26), C(27), ABOVE_VECTOR_2(27));
U185: MYMUX2 PORT MAP (BELOW_VECTOR_2(26), ABOVE_VECTOR_2(26), C(27), BELOW_VECTOR_2(27));
U186: MYMUX2 PORT MAP (ABOVE_VECTOR_2(27), BELOW_VECTOR_2(27), C(28), ABOVE_VECTOR_2(28));

```
U187: MYMUX2 PORT MAP (BELOW_VECTOR_2(27),
ABOVE_VECTOR_2(27), C(28), BELOW_VECTOR_2(28));
U188: MYMUX2 PORT MAP (ABOVE_VECTOR_2(28),
BELOW_VECTOR_2(28), C(29), ABOVE_VECTOR_2(29));
U189: MYMUX2 PORT MAP (BELOW_VECTOR_2(28),
ABOVE_VECTOR_2(28), C(29), BELOW_VECTOR_2(29));
U190: MYMUX2 PORT MAP (ABOVE_VECTOR_2(29),
BELOW_VECTOR_2(29), C(30), ABOVE_VECTOR_2(30));
U191: MYMUX2 PORT MAP (BELOW_VECTOR_2(29),
ABOVE_VECTOR_2(29), C(30), BELOW_VECTOR_2(30));
U192: MYMUX2 PORT MAP (ABOVE_VECTOR_2(30),
BELOW_VECTOR_2(30), C(31), ABOVE_VECTOR_2(31));
U193: MYMUX2 PORT MAP (BELOW_VECTOR_2(30),
ABOVE_VECTOR_2(30), C(31), BELOW_VECTOR_2(31));
U194: MYMUX2 PORT MAP (ABOVE_VECTOR_2(31),
BELOW_VECTOR_2(31), C(32), ABOVE_VECTOR_2(32));
U195: MYMUX2 PORT MAP (BELOW_VECTOR_2(31),
ABOVE_VECTOR_2(31), C(32), BELOW_VECTOR_2(32));
U196: MYMUX2 PORT MAP (ABOVE_VECTOR_2(32),
BELOW_VECTOR_2(32), C(33), ABOVE_VECTOR_2(33));
U197: MYMUX2 PORT MAP (BELOW_VECTOR_2(32),
ABOVE_VECTOR_2(32), C(33), BELOW_VECTOR_2(33));
U198: MYMUX2 PORT MAP (ABOVE_VECTOR_2(33),
BELOW_VECTOR_2(33), C(34), ABOVE_VECTOR_2(34));
U199: MYMUX2 PORT MAP (BELOW_VECTOR_2(33),
ABOVE_VECTOR_2(33), C(34), BELOW_VECTOR_2(34));
U200: MYMUX2 PORT MAP (ABOVE_VECTOR_2(34),
BELOW_VECTOR_2(34), C(35), ABOVE_VECTOR_2(35));
U201: MYMUX2 PORT MAP (BELOW_VECTOR_2(34),
ABOVE_VECTOR_2(34), C(35), BELOW_VECTOR_2(35));
U202: MYMUX2 PORT MAP (ABOVE_VECTOR_2(35),
BELOW_VECTOR_2(35), C(36), ABOVE_VECTOR_2(36));
U203: MYMUX2 PORT MAP (BELOW_VECTOR_2(35),
ABOVE_VECTOR_2(35), C(36), BELOW_VECTOR_2(36));
U204: MYMUX2 PORT MAP (ABOVE_VECTOR_2(36),
BELOW_VECTOR_2(36), C(37), ABOVE_VECTOR_2(37));
U205: MYMUX2 PORT MAP (BELOW_VECTOR_2(36),
ABOVE_VECTOR_2(36), C(37), BELOW_VECTOR_2(37));
U206: MYMUX2 PORT MAP (ABOVE_VECTOR_2(37),
BELOW_VECTOR_2(37), C(38), ABOVE_VECTOR_2(38));
U207: MYMUX2 PORT MAP (BELOW_VECTOR_2(37),
ABOVE_VECTOR_2(37), C(38), BELOW_VECTOR_2(38));
U208: MYMUX2 PORT MAP (ABOVE_VECTOR_2(38),
BELOW_VECTOR_2(38), C(39), ABOVE_VECTOR_2(39));
U209: MYMUX2 PORT MAP (BELOW_VECTOR_2(38),
ABOVE_VECTOR_2(38), C(39), BELOW_VECTOR_2(39));
U210: MYMUX2 PORT MAP (ABOVE_VECTOR_2(39),
BELOW_VECTOR_2(39), C(40), ABOVE_VECTOR_2(40));
U211: MYMUX2 PORT MAP (BELOW_VECTOR_2(39),
ABOVE_VECTOR_2(39), C(40), BELOW_VECTOR_2(40));
U212: MYMUX2 PORT MAP (ABOVE_VECTOR_2(40),
BELOW_VECTOR_2(40), C(41), ABOVE_VECTOR_2(41));
U213: MYMUX2 PORT MAP (BELOW_VECTOR_2(40),
ABOVE_VECTOR_2(40), C(41), BELOW_VECTOR_2(41));
U214: MYMUX2 PORT MAP (ABOVE_VECTOR_2(41),
BELOW_VECTOR_2(41), C(42), ABOVE_VECTOR_2(42));
U215: MYMUX2 PORT MAP (BELOW_VECTOR_2(41),
ABOVE_VECTOR_2(41), C(42), BELOW_VECTOR_2(42));
U216: MYMUX2 PORT MAP (ABOVE_VECTOR_2(42),
BELOW_VECTOR_2(42), C(43), ABOVE_VECTOR_2(43));
U217: MYMUX2 PORT MAP (BELOW_VECTOR_2(42),
ABOVE_VECTOR_2(42), C(43), BELOW_VECTOR_2(43));
U218: MYMUX2 PORT MAP (ABOVE_VECTOR_2(43),
BELOW_VECTOR_2(43), C(44), ABOVE_VECTOR_2(44));
U219: MYMUX2 PORT MAP (BELOW_VECTOR_2(43),
ABOVE_VECTOR_2(43), C(44), BELOW_VECTOR_2(44));
U220: MYMUX2 PORT MAP (ABOVE_VECTOR_2(44),
BELOW_VECTOR_2(44), C(45), ABOVE_VECTOR_2(45));
U221: MYMUX2 PORT MAP (BELOW_VECTOR_2(44),
ABOVE_VECTOR_2(44), C(45), BELOW_VECTOR_2(45));
U222: MYMUX2 PORT MAP (ABOVE_VECTOR_2(45),
BELOW_VECTOR_2(45), C(46), ABOVE_VECTOR_2(46));
U223: MYMUX2 PORT MAP (BELOW_VECTOR_2(45),
ABOVE_VECTOR_2(45), C(46), BELOW_VECTOR_2(46));
U224: MYMUX2 PORT MAP (ABOVE_VECTOR_2(46),
BELOW_VECTOR_2(46), C(47), ABOVE_VECTOR_2(47));
U225: MYMUX2 PORT MAP (BELOW_VECTOR_2(46),
ABOVE_VECTOR_2(46), C(47), BELOW_VECTOR_2(47));
U226: MYMUX2 PORT MAP (ABOVE_VECTOR_2(47),
BELOW_VECTOR_2(47), C(48), ABOVE_VECTOR_2(48));
U227: MYMUX2 PORT MAP (BELOW_VECTOR_2(47),
ABOVE_VECTOR_2(47), C(48), BELOW_VECTOR_2(48));
U228: MYMUX2 PORT MAP (ABOVE_VECTOR_2(48),
BELOW_VECTOR_2(48), C(49), ABOVE_VECTOR_2(49));
U229: MYMUX2 PORT MAP (BELOW_VECTOR_2(48),
ABOVE_VECTOR_2(48), C(49), BELOW_VECTOR_2(49));
U230: MYMUX2 PORT MAP (ABOVE_VECTOR_2(49),
BELOW_VECTOR_2(49), C(50), ABOVE_VECTOR_2(50));
U231: MYMUX2 PORT MAP (BELOW_VECTOR_2(49),
ABOVE_VECTOR_2(49), C(50), BELOW_VECTOR_2(50));
U232: MYMUX2 PORT MAP (ABOVE_VECTOR_2(50),
BELOW_VECTOR_2(50), C(51), ABOVE_VECTOR_2(51));
U233: MYMUX2 PORT MAP (BELOW_VECTOR_2(50),
ABOVE_VECTOR_2(50), C(51), BELOW_VECTOR_2(51));
U234: MYMUX2 PORT MAP (ABOVE_VECTOR_2(51),
BELOW_VECTOR_2(51), C(52), ABOVE_VECTOR_2(52));
U235: MYMUX2 PORT MAP (BELOW_VECTOR_2(51),
ABOVE_VECTOR_2(51), C(52), BELOW_VECTOR_2(52));
U236: MYMUX2 PORT MAP (ABOVE_VECTOR_2(52),
BELOW_VECTOR_2(52), C(53), ABOVE_VECTOR_2(53));
U237: MYMUX2 PORT MAP (BELOW_VECTOR_2(52),
ABOVE_VECTOR_2(52), C(53), BELOW_VECTOR_2(53));
U238: MYMUX2 PORT MAP (ABOVE_VECTOR_2(53),
BELOW_VECTOR_2(53), C(54), ABOVE_VECTOR_2(54));
U239: MYMUX2 PORT MAP (BELOW_VECTOR_2(53),
ABOVE_VECTOR_2(53), C(54), BELOW_VECTOR_2(54));
U240: MYMUX2 PORT MAP (ABOVE_VECTOR_2(54),
BELOW_VECTOR_2(54), C(55), ABOVE_VECTOR_2(55));
U241: MYMUX2 PORT MAP (BELOW_VECTOR_2(54),
ABOVE_VECTOR_2(54), C(55), BELOW_VECTOR_2(55));
U242: MYMUX2 PORT MAP (ABOVE_VECTOR_2(55),
BELOW_VECTOR_2(55), C(56), ABOVE_VECTOR_2(56));
U243: MYMUX2 PORT MAP (BELOW_VECTOR_2(55),
ABOVE_VECTOR_2(55), C(56), BELOW_VECTOR_2(56));
U244: MYMUX2 PORT MAP (ABOVE_VECTOR_2(56),
BELOW_VECTOR_2(56), C(57), ABOVE_VECTOR_2(57));
U245: MYMUX2 PORT MAP (BELOW_VECTOR_2(56),
ABOVE_VECTOR_2(56), C(57), BELOW_VECTOR_2(57));
U246: MYMUX2 PORT MAP (ABOVE_VECTOR_2(57),
BELOW_VECTOR_2(57), C(58), ABOVE_VECTOR_2(58));
U247: MYMUX2 PORT MAP (BELOW_VECTOR_2(57),
ABOVE_VECTOR_2(57), C(58), BELOW_VECTOR_2(58));
U248: MYMUX2 PORT MAP (ABOVE_VECTOR_2(58),
BELOW_VECTOR_2(58), C(59), ABOVE_VECTOR_2(59));
U249: MYMUX2 PORT MAP (BELOW_VECTOR_2(58),
ABOVE_VECTOR_2(58), C(59), BELOW_VECTOR_2(59));
U250: MYMUX2 PORT MAP (ABOVE_VECTOR_2(59),
BELOW_VECTOR_2(59), C(60), ABOVE_VECTOR_2(60));
U251: MYMUX2 PORT MAP (BELOW_VECTOR_2(59),
ABOVE_VECTOR_2(59), C(60), BELOW_VECTOR_2(60));
U252: MYMUX2 PORT MAP (ABOVE_VECTOR_2(60),
BELOW_VECTOR_2(60), C(61), ABOVE_VECTOR_2(61));
U253: MYMUX2 PORT MAP (BELOW_VECTOR_2(60),
ABOVE_VECTOR_2(60), C(61), BELOW_VECTOR_2(61));
U254: MYMUX2 PORT MAP (ABOVE_VECTOR_2(61),
BELOW_VECTOR_2(61), C(62), ABOVE_VECTOR_2(62));
U255: MYMUX2 PORT MAP (BELOW_VECTOR_2(61),
ABOVE_VECTOR_2(61), C(62), BELOW_VECTOR_2(62));
U256: MYMUX2 PORT MAP (ABOVE_VECTOR_2(62),
BELOW_VECTOR_2(62), C(63), ABOVE_VECTOR_2(63));
U257: MYMUX2 PORT MAP (BELOW_VECTOR_2(62),
ABOVE_VECTOR_2(62), C(63), BELOW_VECTOR_2(63));
U258: DDF PORT MAP(DFF_OUTPUT(1), BELOW_VECTOR_2(63),
ABOVE_VECTOR_2(63));

U259: XOR_GATE PORT MAP(DFF_OUTPUT(0),DFF_OUTPUT(1),Q);
PATH1 <= DFF_OUTPUT(0);
PATH2 <= DFF_OUTPUT(1);
```

END BEHAVIORAL;