RESEARCH ARTICLE

INTERNATIONAL RESEARCH JOURNAL OF MULTIDISCIPLINARY TECHNOVATION

Asian Research Association

DOI: 10.54392/irjmt2556

Resource efficient design of 16 x 16 multiplier by Block RAM generalized to n x n multiplier realized in FPGA

G. Dhanabalan [a,*], H.B. Michael Rajan [b], R. Ashok [c]

[a] Department of Electronics and Communication Engineering, Vel Tech Rangarajan Dr. Sagunthala R & D Institute of Science and Technology, Chennai, Tamil Nadu, India.

[b] Department of Mechanical Engineering, Kings College of Engineering, Punalkulam, Pudukkottai, Tamil Nadu, India

[c] Department of Electronics and Communication Engineering, Kamaraj College of Engineering and Technology, K. Vellakulam, Tamil Nadu, India

* Corresponding Author Email: dhanabalan1972@gmail.com

DOI: https://doi.org/10.54392/irjmt2556

Received: 11-04-2024; Revised: 12-08-2025; Accepted: 07-09-2025; Published: 24-09-2025

Abstract: Signal processing and image processing applications demand an efficient and high speed multiplier. It aids in achieving the overall performance of a particular application. Conventionally, multiplication is achieved with the support of an addition operation. Field Programmable Gate Array (FPGA) based multiplier also utilizes adder elements in its design. This work has proposed a resource optimized solution for multiplication using Block RAM (BRAM), an FPGA hardware resource. It generates partial product by shifting. Shifting a number by '1' bit position to the left is equivalent to its multiplication by 2. FPGA logic identifies binary '1' and its bit position in the 16-bit 'multiplier', left shifts the 16-bit 'multiplicand' according to the bit position. 16-bit 'multiplicand' is converted into 32-bit by adding suitable number of '0's both on least significant bit and most significant bit side. This results in 16 partial products of 32-bit. The design develops a 4-bit adder using BRAM. Suitable port mapping of 4-bit adder produces 8-bit / 16-bit / 32-bit adders. Thus designed 32-bit BRAM adder performs concurrent addition in four stages and releases the output of multiplication of two 16-bit numbers. The delay time of 16-bit multiplier, synthesized for xc7v2000t-2fhg1761 chip, has been identified as 3.070*ns*. The design utilized only 78 and 203 number of lookup tables (LUT) and BRAMs respectively. The proposed idea is valid for the design of n x n multiplier when the HDL logic is modified for 'n' times shifting and 2n-bit adder circuit.

Keywords: Block RAM, Multiplier, FPGA, Logical shifting, LUT

1. Introduction

Many applications like modulation / demodulation of telecommunication signals, image processing [1-3], signal processing [4, 5], spectrum analysis and cryptography [6] have multiplication as a part of it. Apart from these conventional applications, adders have started to intrude in latest technology like deep learning [7]. Field Programmable Gate Array (FPGA) based multiplier also supports Internet of Things based applications as a System on Chip [8]. It has been identified to be the replacement for conventional embedded based multiplier [9]. Embedded based multiplier does not support different applications written in a single program. FPGA allocates hardware resources for other irrelevant applications in addition to the multiplier. In order to achieve the multiplication, different multipliers like Vedic multiplier, Baugh Wooley Multiplier, Wallace tree multiplier and Booth multiplier are used. Many multipliers perform multiplication using adder circuit [10]. Dedicated signed multipliers are also developed using FPGA [11]. This motivates the development of an optimized adder circuit. Optimization shall focus on reduction in the power consumption, execution time, amount of hardware and the area of the entire hardware circuit. Adders, designed using transmission gates, help to produce fast adder with lower power consumption and less area [12]. A typical multiplication algorithm involves at least three stages to complete multiplication. A partial product is generated in the first stage. Second and third stages are used to complete the addition of these partial products. The efficiency of multiplication is achieved when this addition process is optimized.

Several adder circuits like carry save adder, carry look-ahead adder and ripple carry adder are used to do addition. Researchers also concentrate on this part to get the fruitful result of high speed multiplication without compromising on hardware requirement and power consumption [13]. DSP blocks in FPGA shall be used as a resource in the realization of multiplier. It

supports multiplications of different bit sizes like 18, 25 and 36. It also uses BRAM for storing input or output data [14]. Multiplier in [15] is designed by combining grouping algorithm with Look-Up Table (LUT) which is used to generate partial products. Design of 8-bit multiplier using 4:2 compressors is discussed in [16]. It involves full adder for the calculation of partial products. The design is synthesized using Cadence design software. Finite Impulse response (FIR) filter design using 8X8 multiplier has been experimented in [17]. Adders and multipliers are the necessary components in the realization of FIR filter. Therefore the execution speed of FIR filter is governed by these components. Since the multiplier cannot be designed without the adder, speed of the filter is ultimately governed by the adder. Adder is used in the generation of partial products. Multipliers and adders occupy considerable area in designing applications like signal processing and audio processing [18]. Researchers attempt to reduce the requirements of hardware logic for addition. Exclusive review on the performance of FPGA indicates the fact that its reliability depends on the aging of the digital circuits [10]. It actually degrades the performance of digital circuit at transistor level. Therefore, it is necessary to keep track on its performance periodically. A testing logic in addition to the actual application may confirm the deviation of FPGA from its normal functions.

Multiplication using BRAM adder techniques is used in this proposed work. Section 2 elaborates the proposed design of multiplier. It starts with an overview of BRAM, explains the concept of shifting used in multiplication algorithm. It also narrates the idea of invoking the 4-bit BRAM adder to realize a 8-bit / 16-bit / 32-bit adder. This validates the design of n-bit adder which invokes 4-bit adder ultimately.

The simulation results of the proposed design have been discussed in the section 3. It also indicates the FPGA implementation of proposed multiplier and compares the result with the existing multiplier. The conclusion section briefs about the future work that incorporates the proposed design as its basic element.

2. Literature Review

Multipliers play a crucial role in a wide range of applications, particularly in digital signal processing (DSP), digital image processing and other data-intensive operations. They significantly influence both the execution time and power consumption of a system. Therefore, designing energy efficient and high performance multipliers is essential. Resistive RAM (RRAM) is one of the emerging technologies employed in developing efficient multipliers [19]. The resistance state of RRAM—either high or low—affects the logical output, making it suitable for logical circuit designs. In this context, carry save adders (CSA) implemented using serial architecture are commonly integrated into array multiplier designs. These adders are further optimized by employing memristor based implementations. In another approach, lookup table (LUT) based multipliers [20] have been developed to enhance area and delay efficiency. A 4-bit multiplier constructed using four 2-bit LUT based multipliers has been proposed, which has been effectively utilized in the implementation of finite impulse response (FIR) filters. This architecture, based on the Residue Number System (RNS), combines both adders and multipliers to realize an efficient RNS based FIR filter. In the domain of cryptography, modular multipliers with high performance are essential. Two-level CSA architecture [21] has been used to develop a high-speed modular multiplier with low hardware cost.

Table 1. Comparison of multiplier designed in different technologies

Sl. No	Specifications	Results	Remarks
1.	4-bit serial unsigned multipliers [19]	Energy consumption – improvement in % is 30.58%	Memristors used – 36 Nos. The work was compared with Dadda, add & shift and array multiplier.
2.	4-bit serial signed multipliers [19]	Energy consumption – improvement in % is 57.41%	Memristors used – 36 Nos. The work was compared with add & shift, Radix-2 Booth, Baugh-Wooley and array multiplier.
3.	LUT multiplier [20]	Logic elements used – 978. F_{max} for 8 tap with 8-bit word length – 1095.29 MHz	The design was compared with Vedic, Dadda, Wallace multipliers.
4.	CS- R4 multiplier [21]	LUTs – 6804 FFs – 4176 Time – 1.05µs	Proposed multiplier was compared with the existing works cited as references.
5.	4-bit multiplier [22]	LUT – 13 Frequency – 775MHz Critical path delay – 2 X LUT Power – 0.97 µW/MHz	Compared with logic multiplier, LMUL and other multiplers cited in the references.

Radix-4 Montgomery multipliers, which typically suffer from longer critical paths, have been improved by introducing the CS-R4MMM algorithm. This algorithm eliminates carry propagation, significantly reducing computation delay. For convolutional neural network (CNN) applications, a multiplier design that avoids using the digital signal processing (DSP) block of an FPGA core has been proposed in [22]. Instead, multiplication is performed using pre-computed results stored in LUTs. Techniques aimed at minimizing the number of required LUTs further enhance the efficiency of the multiplier by reducing storage overhead.

Table 1 compares various multiplier architectures that utilize techniques such as shift-and-add, memristors, CSAs and LUTs. A critical observation is that most existing designs do not consider BRAM as a resource for multiplication operations. The use of BRAM is gaining traction in different computing applications. Notably, in Very Long Instruction Word (VLIW) processor architectures, BRAM is employed to store frequently used inputs and outputs [23]. This strategic reuse enables certain functions to execute more efficiently. Thus leveraging BRAM in this manner results in 1.52× speed improvement.

3. Proposed Methodology

Nowadays, FPGA is becoming an alternative for microprocessor in many of the commercial as well as industrial applications [24]. It is able to perform the functionalities of a programmable logic controller (PLC) in an efficient manner. It is most widely used in PID controller [25]. The basic hardware resources of FPGA are the slice registers and LUTs. It also facilitates reusable logic unit in the name of IP logic core. Typical examples of it include shift registers, counters, digital signal processor, data transfer logic, direct memory access, embedded processor etc., This work has focused to develop a multiplier using an IP logic core Block RAM (BRAM), hardware resource of FPGA. It is available as a single port or dual port memory. Dual port memory shall be configured as a two independent controllable single port memory. Both ports can be accessed simultaneously for ‘read’ / ‘write’ operation. When one port is accessed for ‘Read’ operation, the other port shall be accessed to perform ‘Write’ operation. Latest FPGA accommodates BRAMs in terms of thousands. A 7 Series Xilinx FPGA has 1,410 BRAM blocks in which each has the memory capacity of 36Kb [26]. All these blocks shall be used independently either for a specific application or different applications. This powerful feature is the motivation for the development of multiplier using BRAM.

The proposed logic in the design of multiplier shall be better understood by this example. Consider the multiplication of a number 257 by another number 59. Mathematical expression for 257 x 59 can be written as,

$257 \times 59 = 257 \times (32+16+8+2+1) = (257 \times 32) + (257 \times 16) + (257 \times 8) + (257 \times 2) + (257 \times 1) = (257 \times 2^5) + (257 \times 2^4) + (257 \times 2^3) + (257 \times 2^1) + (257 \times 2^0)$

Now, the binary equivalent of the number 59 is 00111011. It indicates the value of ‘1’ in the bit positions, 0, 1, 3, 4, 5. Therefore, refer the bit position of ‘1’ in the number 59 and shifts the binary equivalent of 257 to the left accordingly. This leads to the multiplication of the above expression. Thus, partial products are generated with the help of shifting. Adder, designed using BRAM, formulates the partial products suitably and executes faster addition. Proposed multiplication algorithm, solely focusing on BRAM, is explained below. Result of any n-bit multiplication produces the product of 2n-bits. Hence, ‘0’s will be added in the least significant bit and most significant bit side.

3.1 BRAM – an Overview

FPGA has embedded BRAM resources which shall be created using Block Memory Generator (BMG). It has two independent ports, both of which has ‘read’ and ‘write’ interface. One can configure the BMG either as a single port or dual port memory. BMG in a design will be automatically optimized for low power and efficient resource utilization during synthesis.

Table 2. Signal details of single port BRAM

Signal name	Input / Output	Signal description
ADDRA[7:0]	Input	Memory address of Port A to access the data. It is configured as 8-bit address. Therefore, 256 data can be stored.
DINA[4:0]	Input	Data input which will be written into the BRAM through Port A. It accepts 5-bit data.
WEA	Input	Write operation is enabled through Port A.
CLKA	Input	Clock signal applied to the Port A is synchronous.
DOUTA[4:0]	Output	Data stored in the BRAM shall be read and released through Port A. It releases 5-bit data.

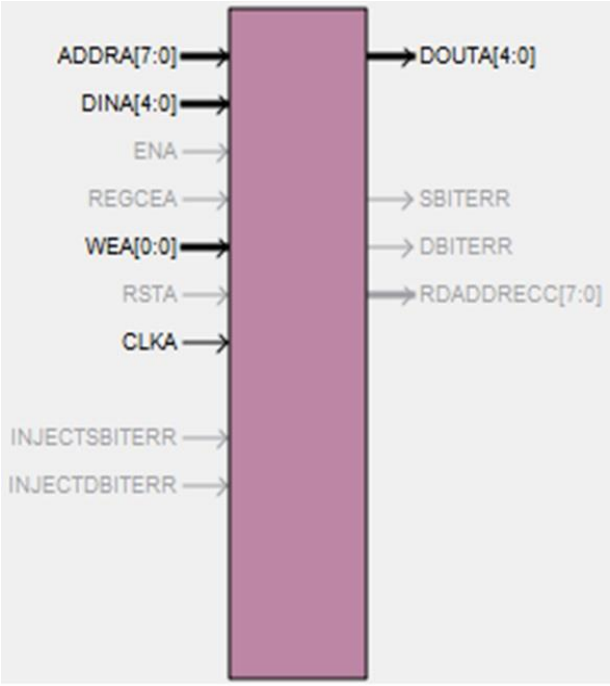

Figure 1. Single port BRAM

It does not demand any additional logic when the BRAM is implemented as a single port BRAM. A typical FPGA supports the data width of 1 – 4608-bits, memory depths of 2 – 9M words [26]. One of the interesting features of the BRAM is that 'read' and 'write' channels are independent. Figure 1 indicates the details of a BRAM configured as a single port RAM.

Description of signals, assigned for a single port BRAM, is listed in Table 2. Inference of Table 2 indicates that BRAM has been designed for the memory size of 5 X 256.

3.2 Multiplication Algorithm

Step 1: Get two 16-bit numbers (multiplicand, multiplier) which are to be multiplied.

Step 2: Refer the value of LSB bit of multiplicand. If it is '0', then store the result as '0' of 32-bits. If it is '1', then retain the multiplier value, insert a '0' in the LSB side, and insert sufficient number of '0' in the MSB side so that the total number of bit to be stored is 32-bits.

Step 3: Refer the first bit position of the multiplicand and follow the rule mentioned in the Step 2.

Step 4: Repeat the Step 3 until all the bit position of the multiplicand is referred and processed. Now, there will be 16 data arrived for 16-bit positions of the multiplicand.

Step 5: In stage 1, group the 16 data into 8 pairs and perform the addition using BRAM. After the addition, there will be 8 data.

Step 6: In stage 2, group the 8 data into 4 pairs and perform the addition using BRAM. After the addition, there will be 4 data.

Step 7: In stage 3, group the 4 data into 2 pairs and perform the addition using BRAM. After the addition, there will be 2 data.

Step 8: In stage 4, group the 2 data into a single and perform the addition using BRAM. This leads to the multiplication of two 16-bit numbers.

Figure 2 shows the process of shifting the multiplier and adding '0's in suitable bit positions with reference to the bit value of multiplicand by assuming its data size to be of 8-bit.

The block diagram representation of proposed multiplier designed using BRAM adder is portrayed in the Figure 3. Partial products are generated through the process of shifting. They are grouped as pairs and concurrent additions are performed by 4-bit BRAM adders. 4-bit adders are invoked for required number of times to realize a 8 / 16 / 32-bit adder.

3.3 4-bit BRAM adder – Realization

Efficient adder circuit is the backbone of any application involving complicated arithmetic operations. There are different adder circuits like carry save adder, carry look-ahead adder and ripple carry adder to carry out efficient addition [27]. In general, speed of addition is solely based on how effectively the adder circuit handles the carry. This work has attempted to develop a multiplier circuit using BRAM addition. Two numbers which are to be added are concatenated to form the address of BRAM and write the anticipated addition as data in it. The process of address and data generation using 2-bit addition is shown in Table 3.

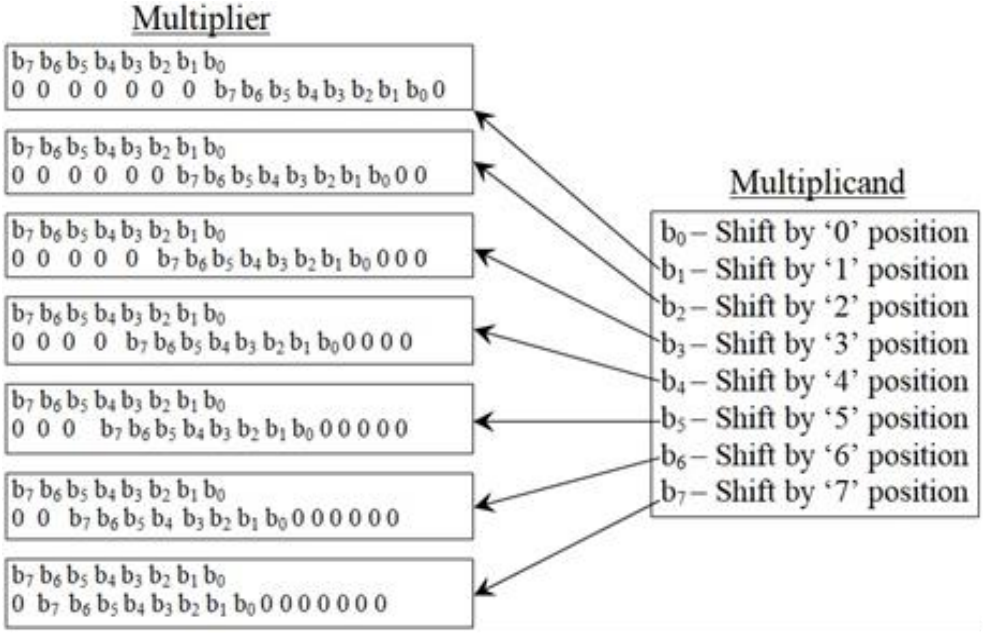

Figure 2. 8-bit 'Multiplier' shifting in reference to Multiplicand bit status

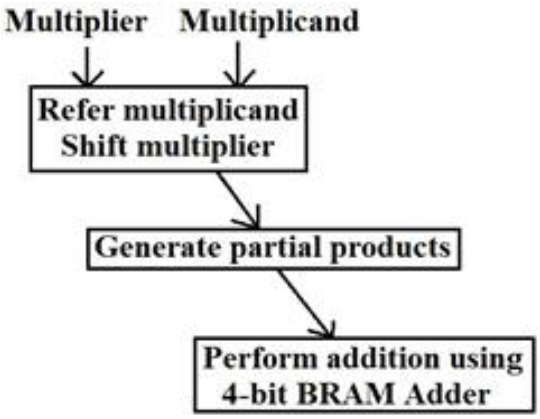

Figure 3. Proposed BRAM multiplier

Table 3. Conversion of addition in the form of LUT

Sl. No	Numbers to add (in Decimal)	Address (in Binary)				Addition as Data (in Binary)			Addition (in decimal)
		ad_3	ad_2	ad_1	ad_0	d_2	d_1	d_0	
1.	0 + 0	0	0	0	0	0	0	0	0
2.	0 + 1	0	0	0	1	0	0	1	1
3.	0 + 2	0	0	1	0	0	1	0	2
4.	0 + 3	0	0	1	1	0	1	1	3
5.	1 + 0	0	1	0	0	0	0	1	1
6.	1 + 1	0	1	0	1	0	1	0	2
7.	1 + 2	0	1	1	0	0	1	1	3
8.	1 + 3	0	1	1	1	1	0	0	4
9.	2 + 0	1	0	0	0	0	1	0	2
10.	2 + 1	1	0	0	1	0	1	1	3
11.	2 + 2	1	0	1	0	1	0	0	4
12.	2 + 3	1	0	1	1	1	0	1	5
13.	3 + 0	1	1	0	0	0	1	1	3
14.	3 + 1	1	1	0	1	1	0	0	4
15.	3 + 2	1	1	1	0	1	0	1	5
16.	3 + 3	1	1	1	1	1	1	0	6

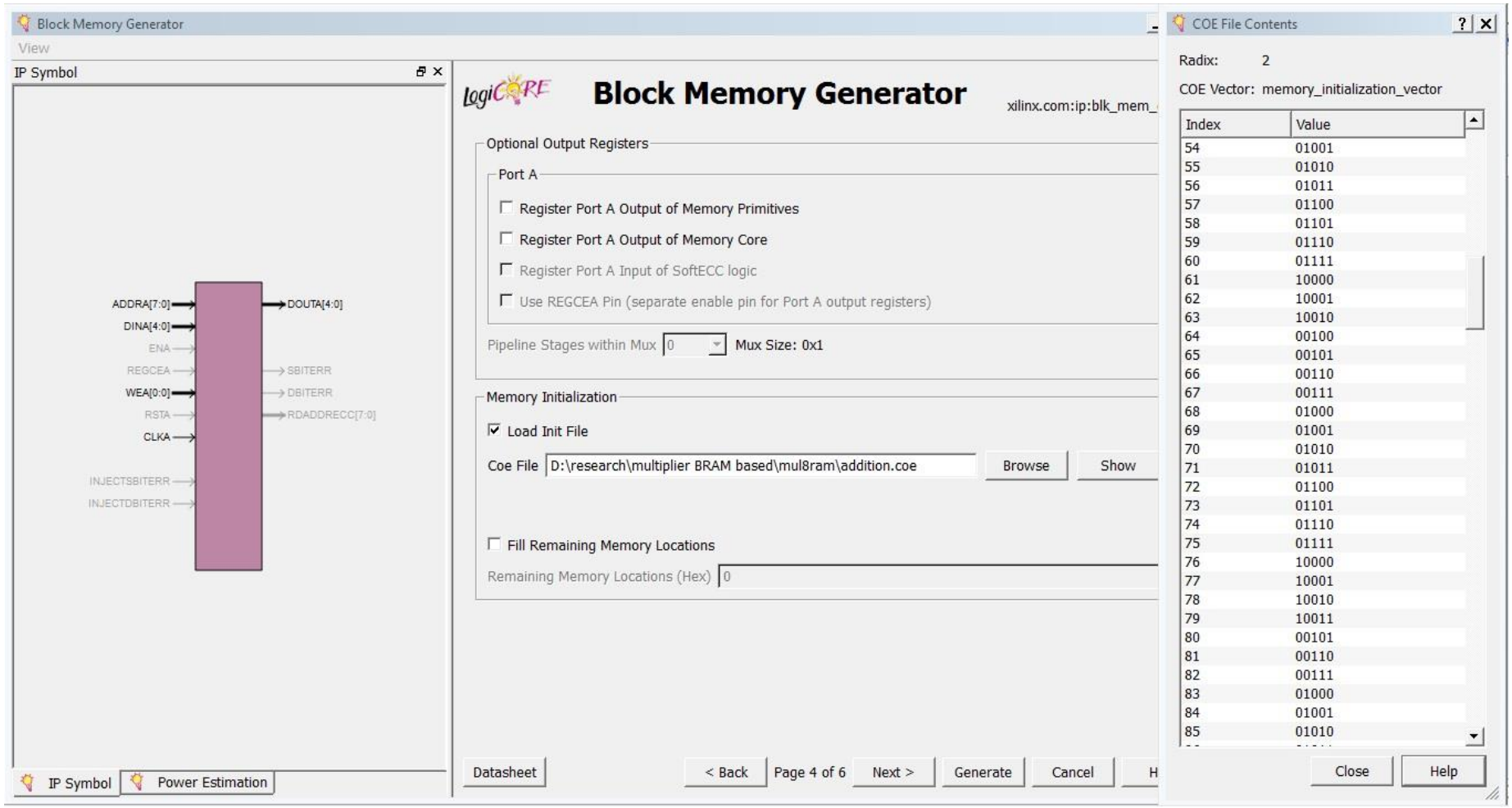

Figure 4. Overview of LUT content during BRAM creation in BMG

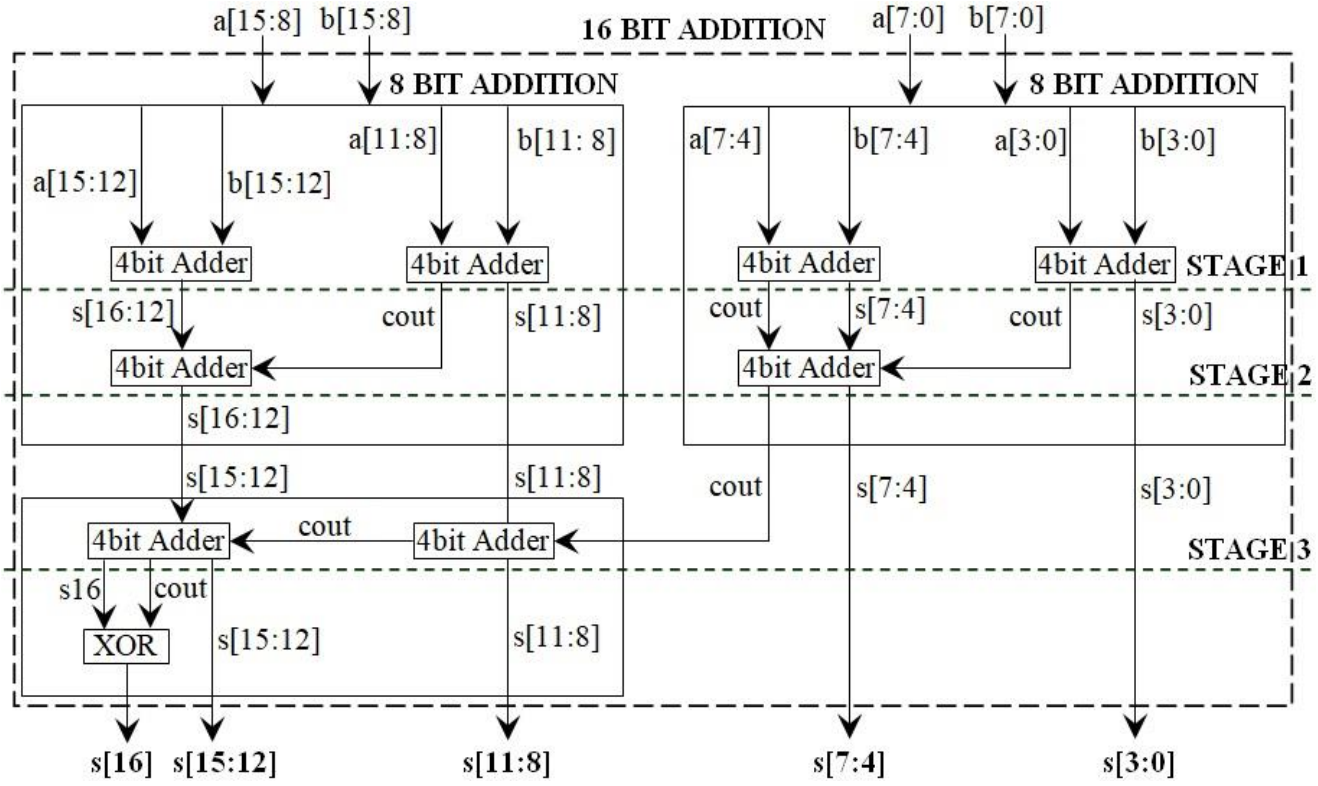

Figure 5. Three stages of 16-bit addition

Address Generation:

$(a_1\, a_0 + b_1\, b_0)$ $= \{b_1\, b_0\, a_1\, a_0\}$ (1)

$= ad_3\, ad_2\, ad_1\, ad_0$ (2)

Data generation:

$(a_1\, a_0 + b_1\, b_0)$ $= (b_1+a_1) + (b_0+a_0)$ (3)

$= c_0\, s_1\, s_0$ (4)

$= d_2\, d_1\, d_0$ (5)

Since the addition of 2-bit generates a 3-bit data, BRAM stores 3-bit data using 4-bit address. 2-bit addition will have 16 possible combinations of addition. Therefore, the address size must be 4-bit as listed in the Table 3. LUT for 4-bit addition shall be derived by following the footsteps of 2-bit addition. The concatenation of two 4-bit data results in 8-bit. Thus it can have 256 possible combinations of additions. Result of 4-bit addition is a 5-bit data. Therefore, it is necessary to design a BRAM of storing 5-bit data in 256 locations. LUT designed for 4-bit addition must be stored as a file with the extension of '.coe'.

Thus, formulated LUT file needs to be uploaded when creating a BRAM using BMG. Figure 4 indicates the creation of a BRAM with the specification of 5 X 256. COE Vector Table (LUT) can be seen in the right side of the Figure 4. It shall be understood by analyzing the following example. The value of 'index' 67 is denoted as 00111.

In the context of BRAM, 'index' and 'value' are equivalent to 'address' and 'data' respectively. Binary equivalent of '67' is '01000011'. This value is arrived while concatenating '0100' ('4' to be added with) with '0011' ('3'). Addition of these two numbers (4 + 3) results into 7 whose equivalent binary number is 0111. The 'value' is represented in 5-bit size considering the generation of carry in a 4-bit addition. BGM allocates one 18K BRAM. Its memory type is configured as a single port BRAM memory. This design has not been opted for any power optimization techniques. The data will be read in a single clock cycle.

Figure 5 elaborates the different stages of 16-bit addition. It separates the 16-bit numbers into two 8-bit numbers and performs 8-bit addition in the fist stage. 8-bit numbers are further grouped into two 4-bit numbers and the 4-bit BRAM adders generate the result of 4-bit addition. Therefore, 4 BRAMs are used to complete the addition in stage1. In the similar way, two BRAMs are used in the remaining stages to complete the addition. Finally, the value of s [16] is XORed with the carry generated by the 4-bit adder in the third stage. This is the carry generated out of a 16-bit addition.

4. Results and Discussions

This work develops a 16-bit multiplier targeting BRAM as its fundamental resource. Multiplication has been established by following the process of shifting and addition. The complete synthesis process has been carried out using Xilinx ISE 14.7 software. The program has been developed using VerilogHDL language. Synthesis result of shifting the data is shown in the Figure 6.

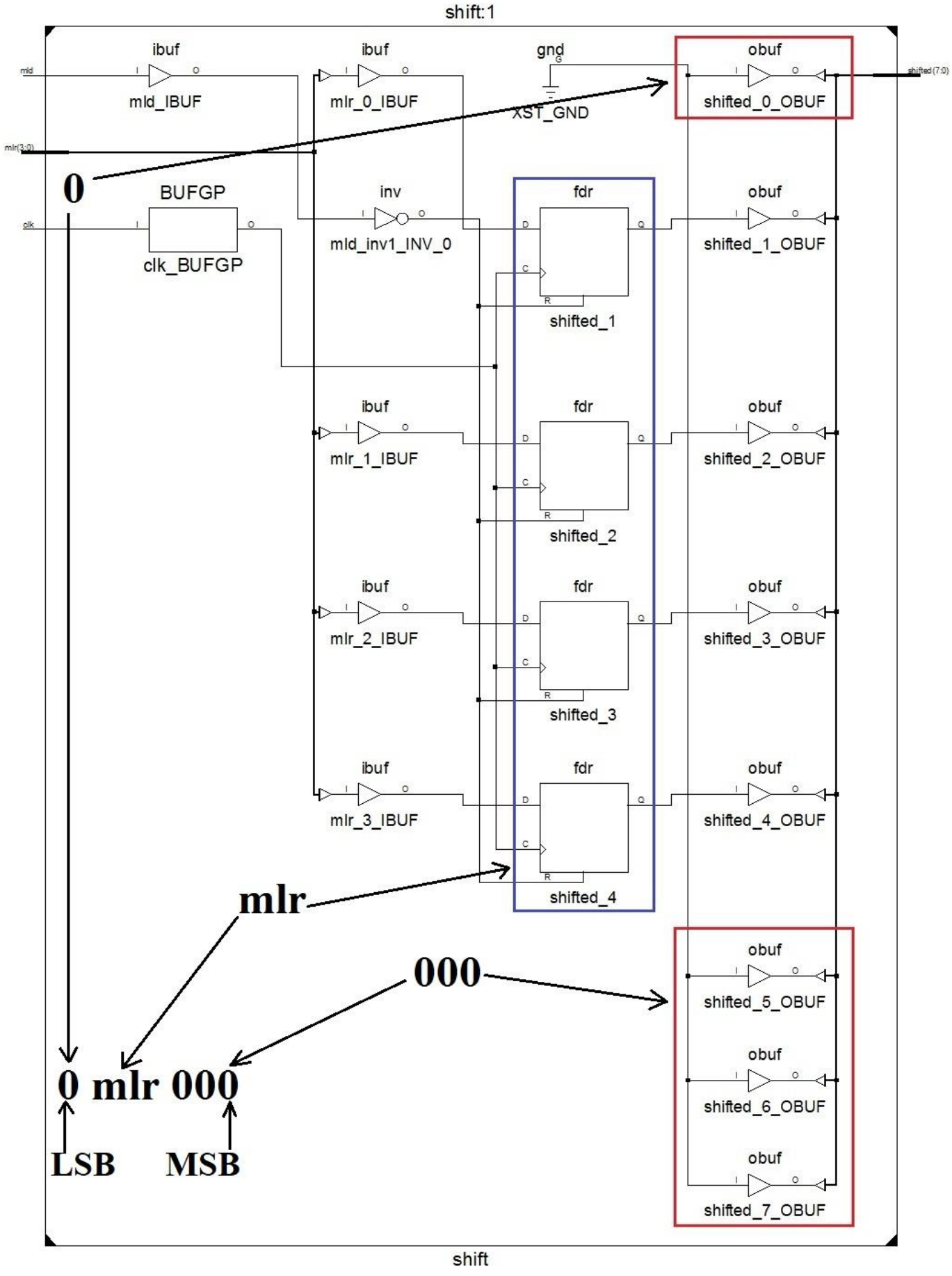

Figure 6. RTL view of data shifting for 8-bit generation

It performs shift left by '1' bit position. It accepts the 4-bit 'mlr' and 1-bit of 'mld' as its inputs and generates the output 'shifted' as an 8-bit data. The 'mlr' is connected to the reset pin of the D flip flop. It forwards the 'mld' to the output when the input of 'mlr' is '1'. A '0' input to the 'mlr' resets the flip flop which leads to the output value 'shifted' as '0'. The process of inserting '0's in suitable bit positions is evident from the Figure 6. Grounding the buffer generates Logic '0'.

RTL schematic of 4-bit adder designed using BRAM is shown in the Figure 7. The design does not require any other hardware component, except a BRAM.

The designs of 8-bit, 16-bit and 32-bit adders are shown in the Figure 8, 9, 10 respectively. In all the adder design, only BRAM adders have been used. This helps to achieve higher execution speed. BRAM is designed to have the feature of concurrent execution of all the BRAMs involved in a design. XOR logic is used in every design to finalize the carry generated out of addition.

RTL schematic of 16-bit adder shown in the Figure 9 performs the addition in two steps. 8-bit modules fa and fb are independent. While, fc is able to perform addition after receiving the carry result from the module fb. 8-bit module is actually derived from three 4-bit modules. This is also executed in two stages as evident from the RTL schematic.

Figure 11 indicates the RTL schematic of 16-bit multiplier. 32-bit adders in the first group receive the input from the shifter. This confirms that the addition and shifting are done in parallel at the maximum. Remaining additions are performed in three stages. Synthesis report of the 16-bit multiplier has indicated better execution time. Logic involved in the design is minimal. Data transfer from BRAM to other contributes the total delay time.

The Proposed multiplier has also attempted to analyze the impact of 8-bit multiplier. Figure 12 shows its RTL schematic. Here, the addition is performed in three stages. It was observed that the execution time of this multiplication is equivalent to the execution time of 32-bit adder.

4.1 n x n Multiplier

RTL schematic of multiplier indicated in Figure 11 and Figure 12 clarified the fact the 4-bit BRAM adder has been the fundamental resource for the adder. Shifting and addition are the components needed for multiplication. 8-bit multiplier has been realized using 16-bit adder. 16-bit multiplier has been realized using 32-bit adder. Similarly, shifting was done for 8, 16 times in the respective 8, 16-bit multiplier. Therefore n x n multiplier shall be developed when the logic for 'n' times shifting and '2n-bit' adder circuit are developed. The logic is also equivalent to the FPGA hardware resources assigned to it.

The simulation of a 32-bit multiplier is shown in the Figure 13. It indicates the result of multiplication of 9875 * 55 and 9875 * 312. The design has been also implemented in the chip xc6slx9-2-tqg144. It is a SPARTAN6 FPGA educational board.

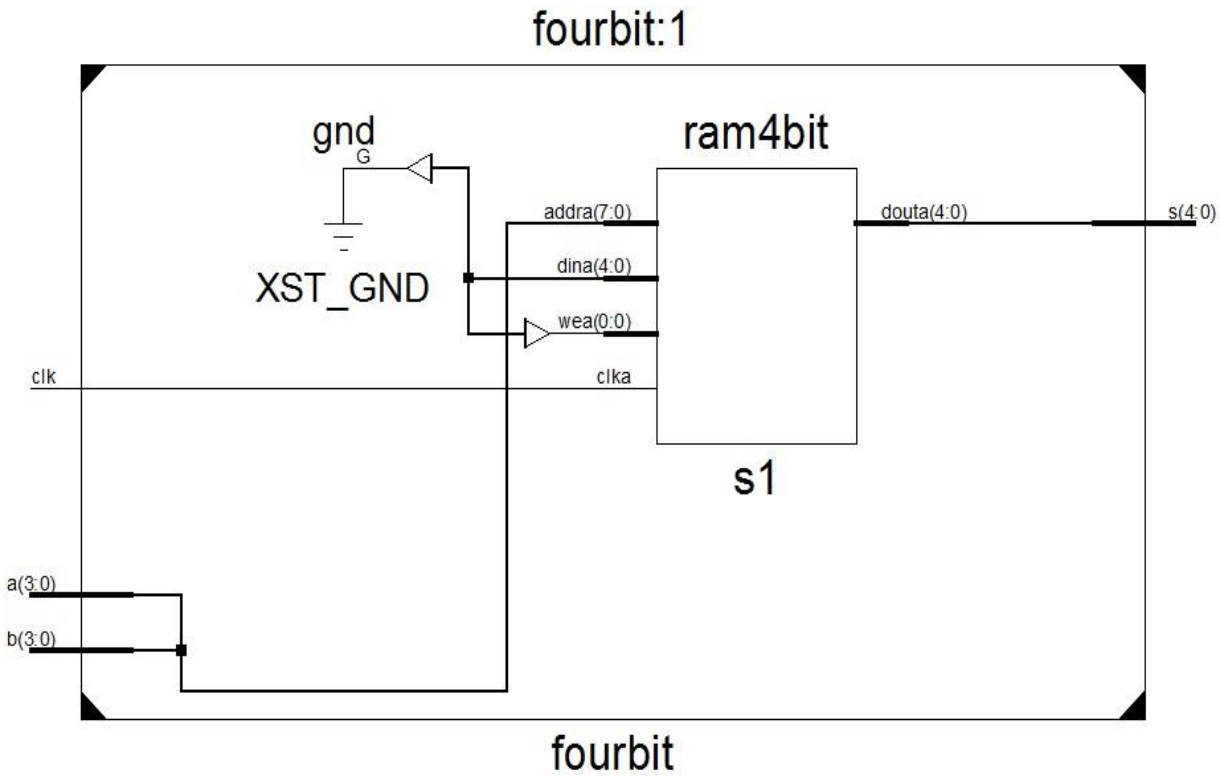

Figure 7. RTL schematic of 4-bit adder using BRAM

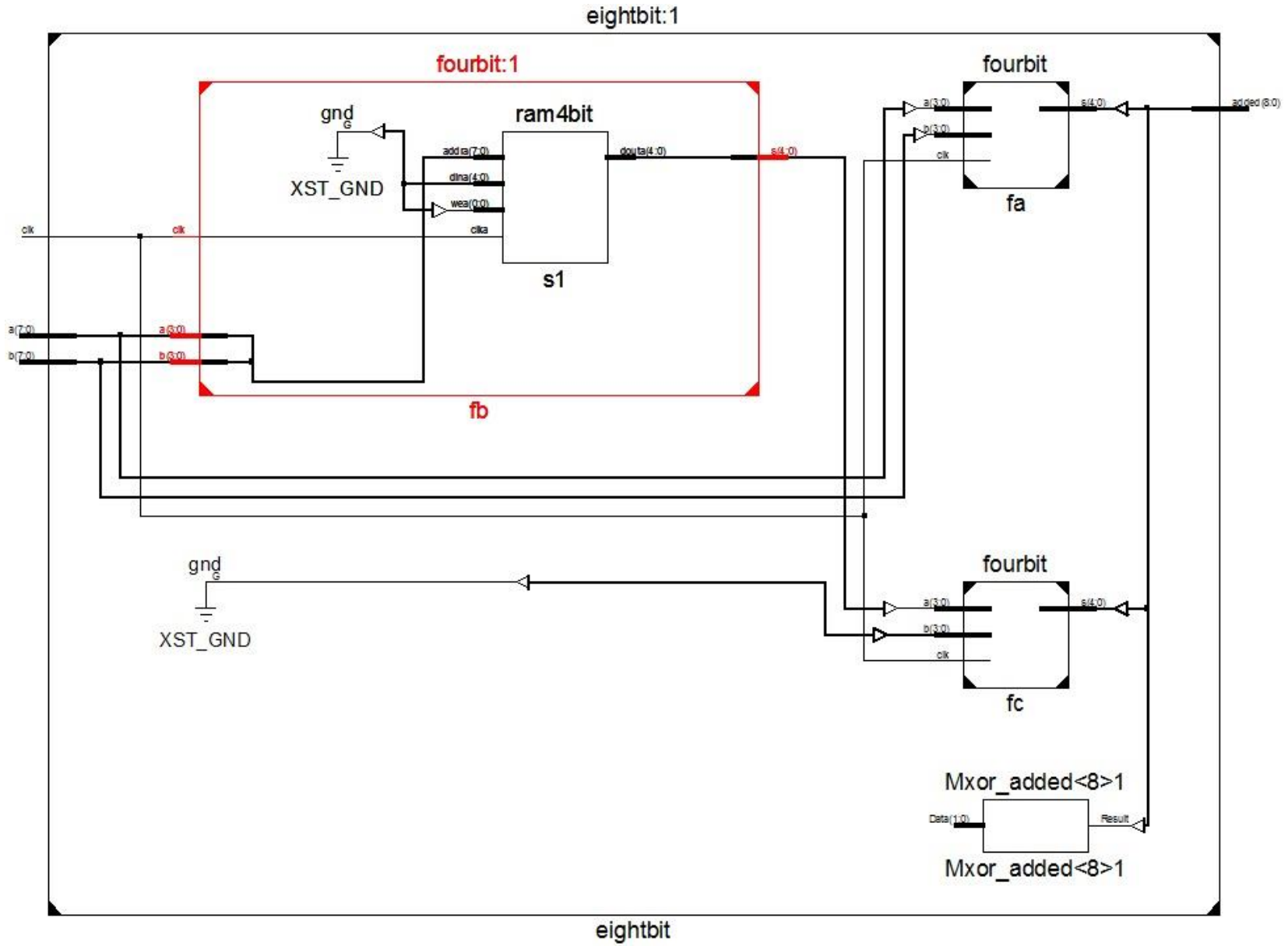

Figure 8. RTL schematic of 8-bit adder

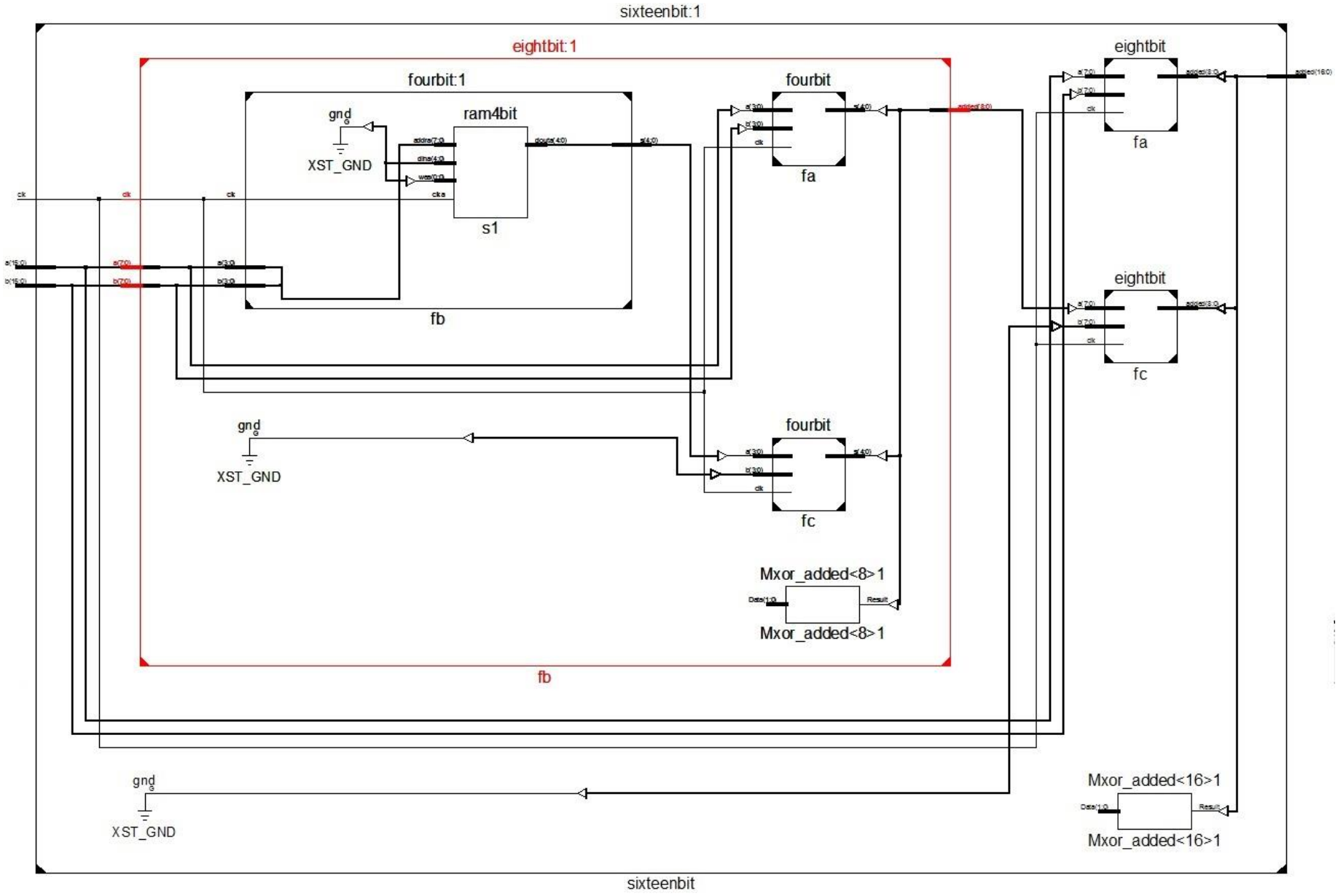

Figure 9. RTL schematic of 16-bit adder

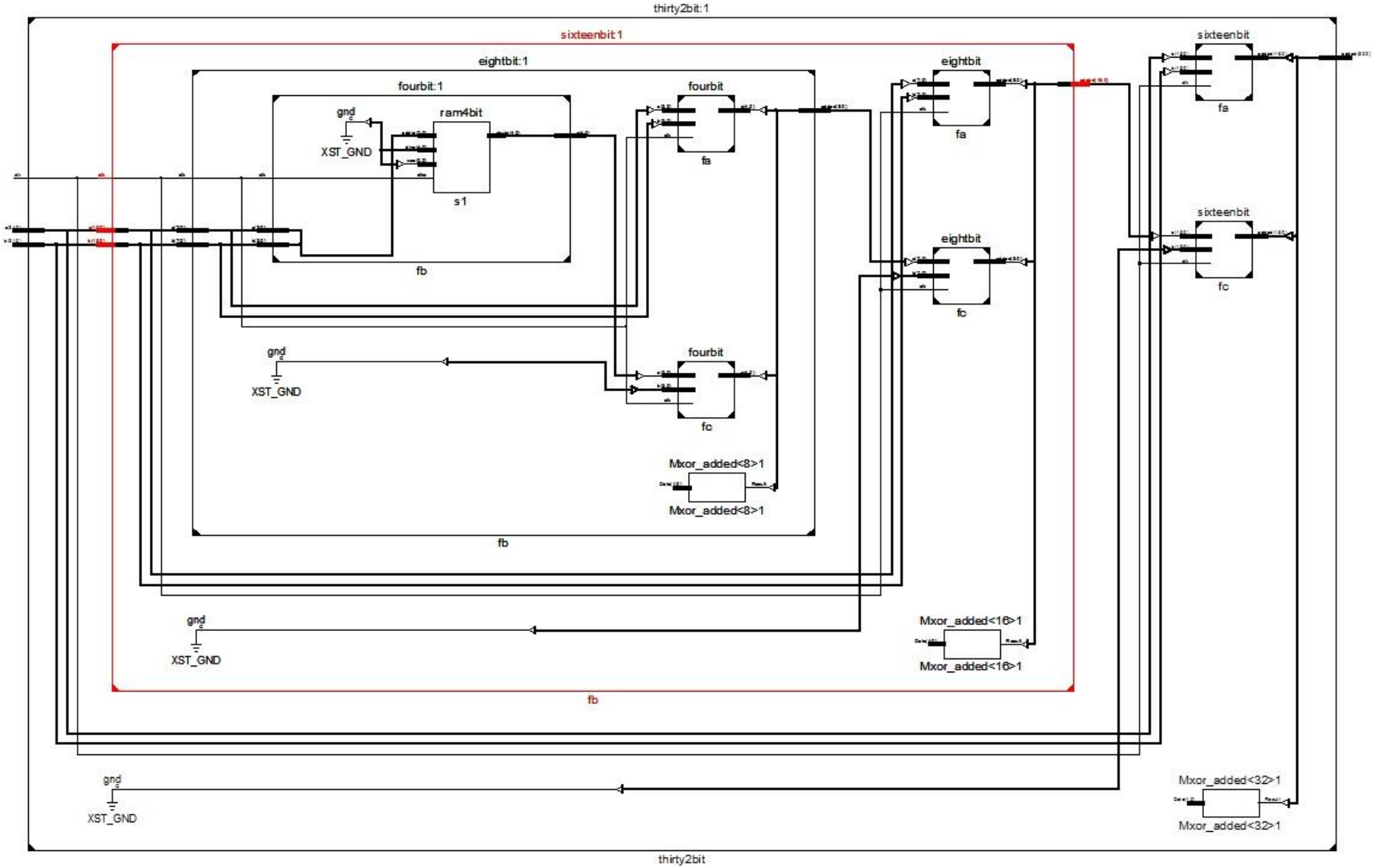

Figure 10. RTL schematic of 32-bit adder

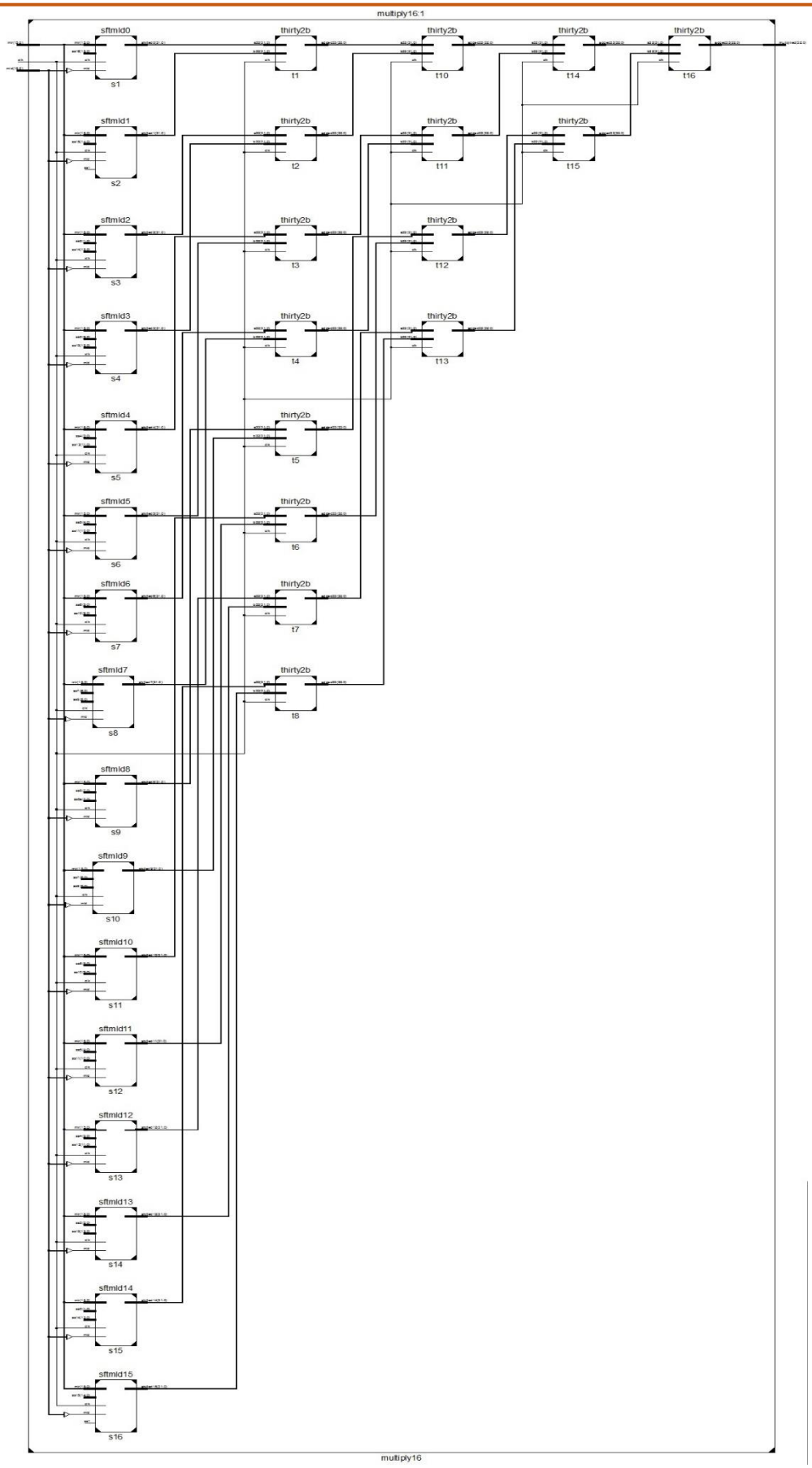

Figure 11. RTL schematic of 16-bit multiplier

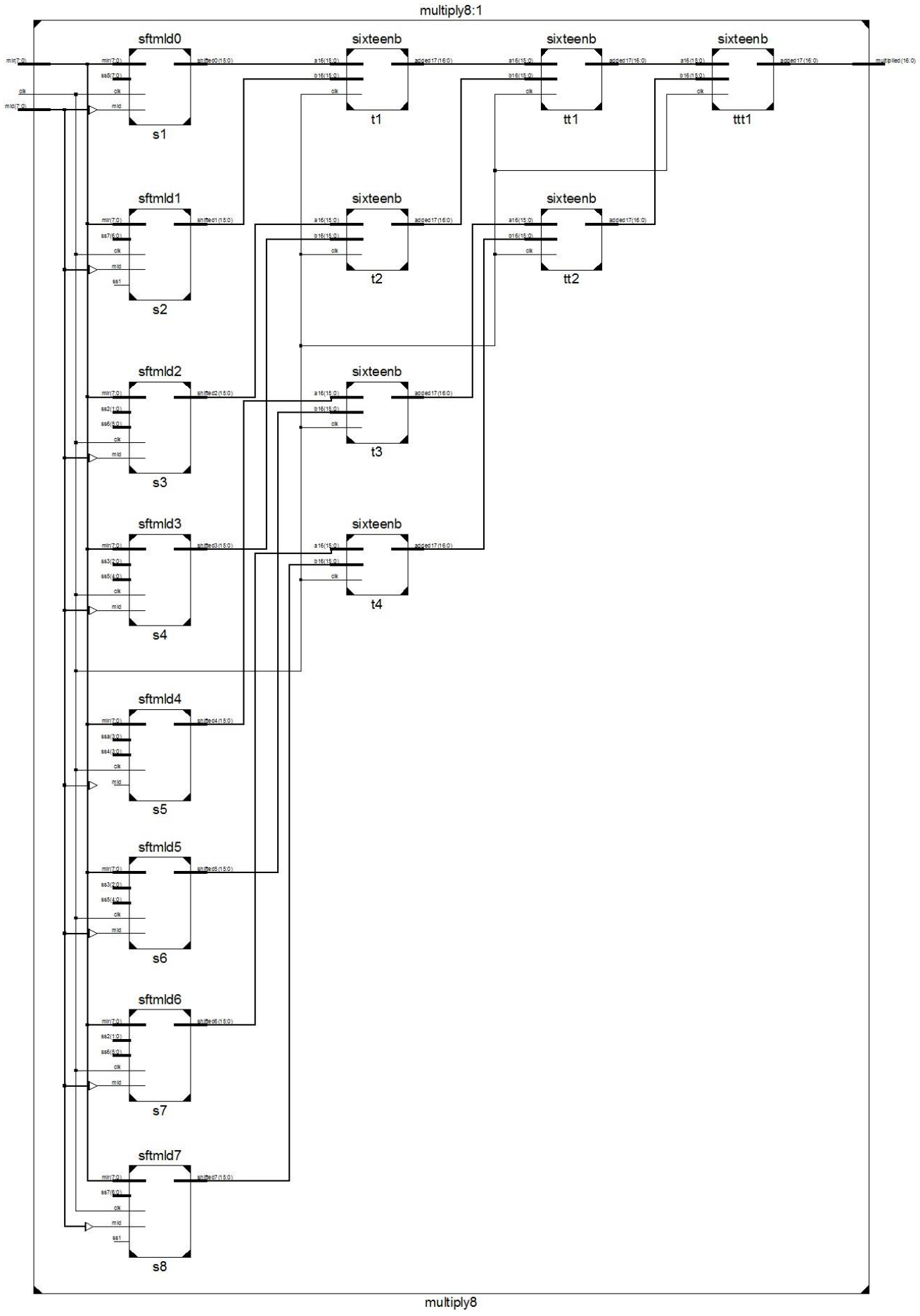

Figure 12. RTL schematic of 8-bit multiplier

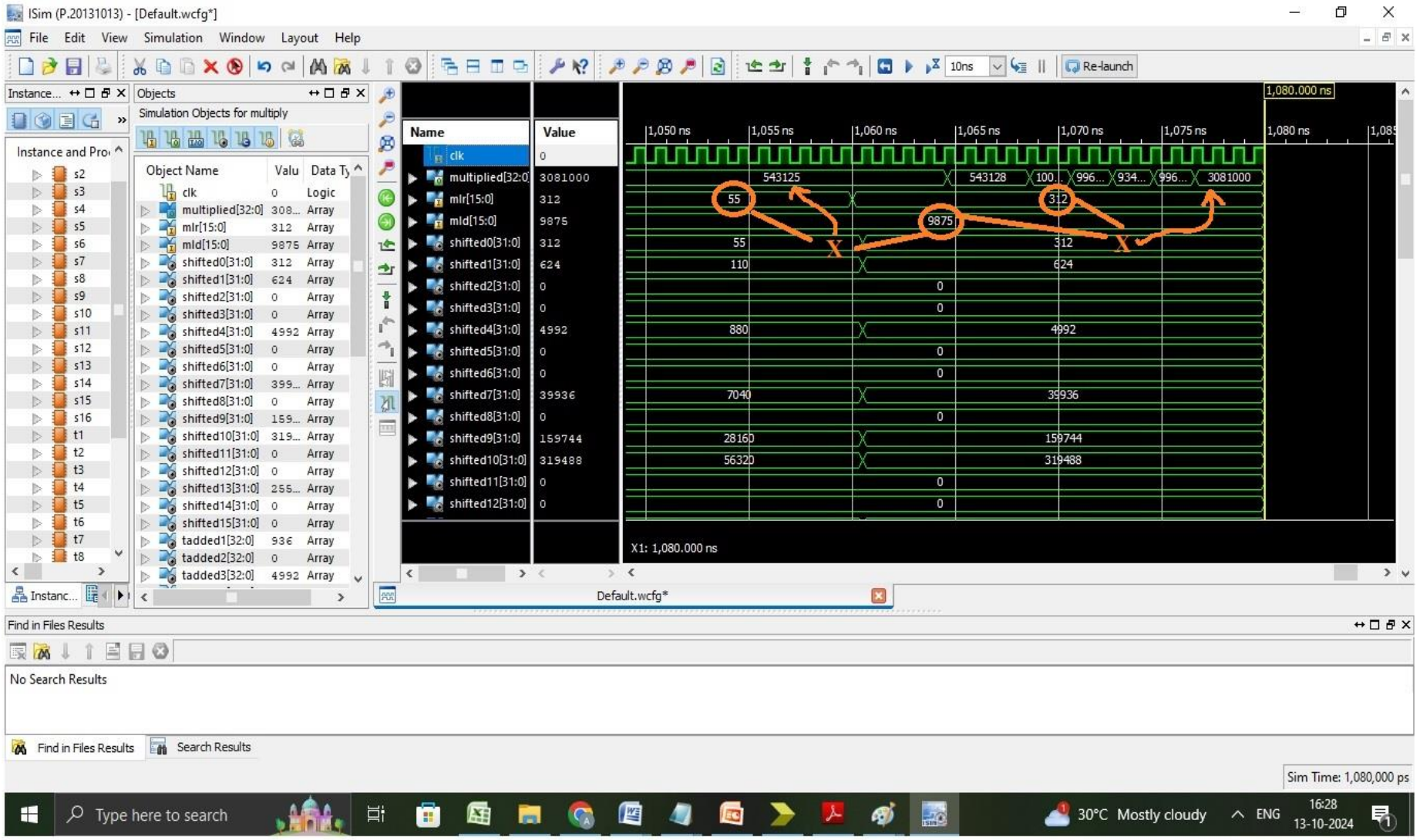

Figure 13. Simulation of 32-bit multiplier

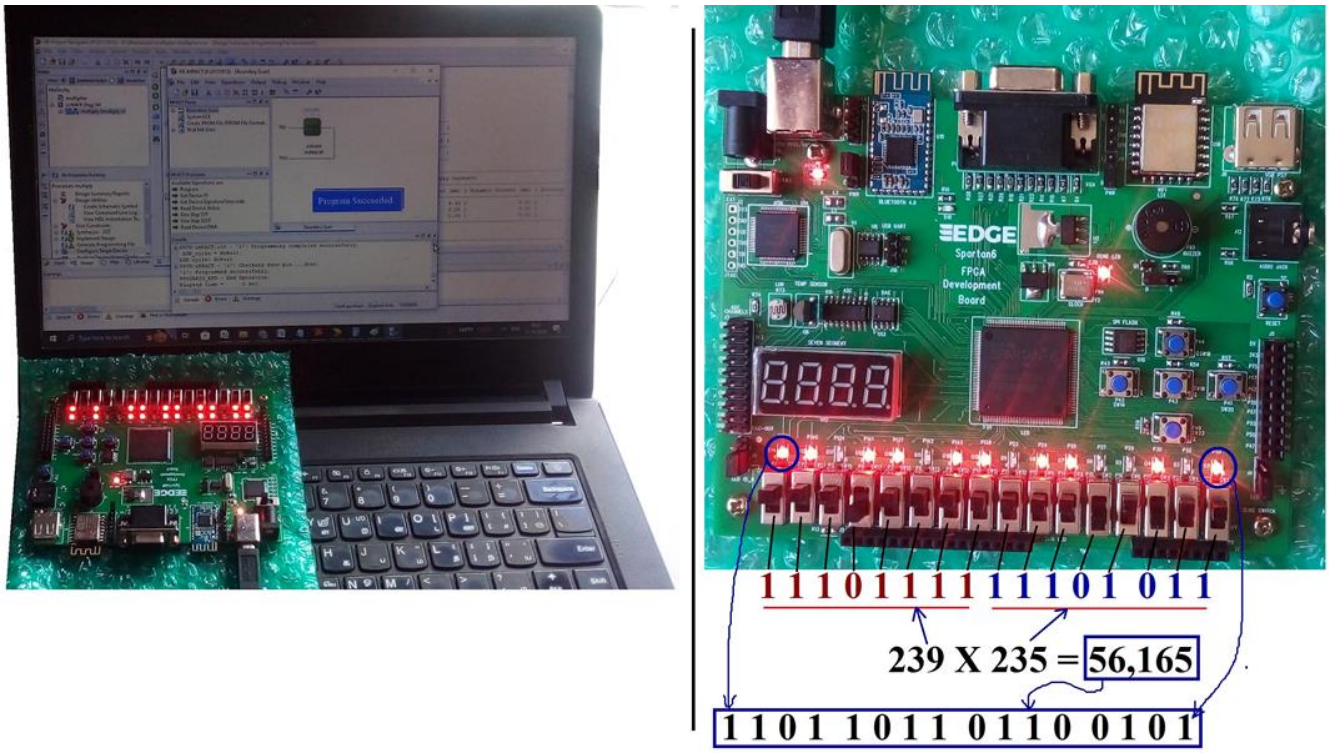

Figure 14. FPGA implementation of 8-bit multiplier in the chip, xc6slx9-2-tqg144

It has 16 switches and 16 LEDs as shown in the Figure 14. Hence a 8-bit multiplication has been implemented to validate the design. Xilinx ISE 14.7 software has been used for FPGA implementation. The bit file generated after the synthesis of the design is downloaded into the chip.

4.2 FPGA hardware resource utilization

Virtex-7 family FPGA was used to synthesize the proposed multiplier. The exact specification of the FPGA chip used is xc7v2000t-2fhg1761. Table 4 lists the hardware utilization of FPGA chip from the available resources. The analysis was done for different types of adders (4/8/16/32-bit) and multipliers (4/8/16-bit). This particular chip has a total of 1292 BRAMs. Xilinx synthesis software optimizes the total number of BRAMs required in the design. Thus the maximum number of BRAMs used in the design is 203 only which is 15.7%.

It is clear that the design has used only minimal amount of the FPGA hardware. To prove the better performance of the proposed design, it is compared with the vedic multiplier in [28]. Details of LUT utilization mentioned in Table 4 and Table 6 are reorganized in Table 5 for easy understanding.

Table 5 clearly indicates that the proposed 4-bit, 8-bit and 16-bit multiplier uses only about 25%, 15% and 18% of LUT used by respective vedic multiplier. Moreover, proposed multiplier has used BRAM while vedic multiplier did not use it. 16% (203/1292) of the FPGA BRAM is used in the proposed multiplier. As a whole, hardware resources utilized to realize a 16-bit multiplier in the proposed design is very minimal. It shall be noted that hardware details mentioned in [1] and [29] are available only for 16-bit and 8-bit multiplier respectively.

4.3 Delay analysis

Table 6 discusses about the delay time created by the FPGA to complete the multiplication. The performance analysis of the proposed multiplier has been compared with [1, 2, 28, 29]. It indicates that the proposed multiplier is able to execute the multiplication at higher speed when compared with the existing multipliers.

This work also analyzed the behaviour of multiplier in comparison with the BRAM adder. BRAM adder has been used in the design of multiplier. Since the multiplication is realized by shifting and addition process, there is not too much impact on the delay time. However, as seen from the Table 7, total delay time of 8-bit and 16-bit multiplier is equivalent to the total delay time of 16-bit and 32-bit adder respectively. It is because, the n-bit multiplier demands 2n-bit addition. This proves that a better design in adder circuit will be able to develop a better multiplier with lesser delay time.

Power analysis of the design for 8-bit multiplier using SPARTAN6 FPGA chip, xc6slx9-2-tqg144, indicates its total on power chip has been to be 14mW. 8-bit multiplier designed using hybrid compressor technique indicates the power requirement of 58.25nW [29].

Table 4. Hardware utilization of xc7v2000t-2fhg1761 FPGA chip for multiplier design

FPGA chip : xc7v2000t-2fhg1761	**Resources available**	**Adder**				**Multiplier**		
		4bit	**8bit**	**16bit**	**32bit**	**4bit**	**8bit**	**16bit**
Number of Slice Registers	24,43,200	0	0	0	0	16	64	256
Number of Slice LUTs	12,21,600	0	1	2	6	4	16	78
Number of bonded IOBs	850	14	26	50	98	17	34	66
Number of Block RAM/FIFO	1,292	1	2	5	14	5	32	203
Number of BUFG/BUFGCTRLs	128	1	1	1	1	1	1	1

Table 5. Hardware comparison – proposed multiplier with Vedic multiplier

Hardware description	[28]			[29]	[2]			[1]	**Proposed multiplier**		
	4-bit	**8-bit**	**16-bit**	**8-bit**	**4-bit**	**8-bit**	**16-bit**	**16-bit**	**4-bit**	**8-bit**	**16-bit**
LUT	16	104	436	83	12	57	206	194	4	16	78
Block RAM	Not used			Not used	Not used	Not used	Not used	Not used	5	32	203

Table 6. Delay analysis of proposed design with existing multipliers

Vedic / proposed multiplier	**Bit size**	**LUTs**	**Block RAM**	**Delay (*ns*) = Logic delay + Route delay**		
				Delay (*ns*)	**Logic delay (*ns*)**	**Route delay (*ns*)**
Proposed multiplier (xc7v2000t-2fhg1761)	4	4	5	2.139	1.800	0.339
	8	16	32	2.704	1.843	0.861
	16	78	203	3.070	1.886	1.184
[1]	16	194	Not used	3.68	Not Available	Not Available
[2]	4	12	Not used	2.016	Not Available	Not Available
	8	57	Not used	3.508		
	16	206	Not used	4.721		
[28]	4	16	Not used	4.1	Not Available	Not Available
	8	104	Not used	4.7		
	16	436	Not used	6.4	Not Available	Not Available
[29]	8	83	Not used	13.75	Not Available	Not Available

Table 7. Delay analysis of proposed adder and multiplier

Bit size	**Adder**			**Multiplier**		
	Logic delay (ns)	**Route delay (ns)**	**Total delay (ns)**	**Logic delay (ns)**	**Route delay (ns)**	**Total delay (ns)**
4-bit	1.800	0.339	**2.139**	1.800	0.339	**2.139**
8-bit	1.843	0.744	**2.587**	1.843	0.861	**2.704**
16-bit	1.843	0.861	**2.704**	1.886	1.184	**3.070**
32-bit	1.886	1.184	**3.070**	----------		

5. Conclusion

In this work, an exclusive and efficient 4-bit BRAM adder is designed and implemented. Applications which are using multiplication shall incorporate this logic as this will reduce the usage of LUTs. On the contrary, those LUTs shall be used for other applications. Synthesis result of shifter is analysed as the multiplication is achieved by shift and add method. Therefore, the design consumed lesser amount of hardware and resulted in high speed multiplier. However, it demanded more number of BRAMs. This ensures the effective utilization of the hardware resource with minimal logic. An intricate analysis of the design of BRAM adder and multiplier on different FPGA chips demonstrated different execution speed. Comparison between the delay time of BRAM adder and multiplier disclosed a fact that delay time of 'n-bit' multiplier is equivalent to the delay time of '2n-bit' BRAM adder. This design is applicable for integer multiplication only. Many applications involve Floating point multiplication. Design of floating point multiplication using BRAM adder will be developed in the feature work.

References

[1] H. Waris, C. Wang, W. Liu, F. Lombardi, AxBMs: Approximate Radix-8 Booth Multipliers for High-Performance FPGA-Based Accelerators. IEEE Transactions on Circuits and Systems II: Express Briefs, 68(5), (2021) 1566-1570. https://doi.org/10.1109/TCSII.2021.3065333

[2] S. Ullah, S. Rehman, M. Shafique, A. Kumar, High-Performance Accurate and Approximate Multipliers for FPGA-Based Hardware Accelerators. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(2), (2022) 211-224. https://doi.org/10.1109/TCAD.2021.3056337

[3] S. Cekli, A. Akman, A high speed pipelined radix-16 Booth multiplier architecture for FPGA implementation. AEU - International Journal of Electronics and Communications, 185, (2024) 155435. https://doi.org/10.1016/j.aeue.2024.155435

[4] A. Bottcher, M. Kumm, Towards Globally Optimal Design of Multipliers for FPGAs. IEEE Transactions on Computers, 72(5), (2023) 1261-1273. https://doi.org/10.1109/TC.2023.3238128

[5] L. Malathi, A. Bharathi, A.N. Jayanthi, FPGA design of FFT based intelligent accelerator with optimized Wallace tree multiplier for image super resolution and quality enhancement. Biomedical Signal Processing and Control, 88(B), (2024) 105599. https://doi.org/10.1016/j.bspc.2023.105599

[6] A. Parihar, S. Nakhate, Low latency high throughput Montgomery modular multiplier for RSA cryptosystem. Engineering Science and Technology, an International Journal, 30, (2022) 101045. https://doi.org/10.1016/j.jestch.2021.08.002

[7] C.M. Kalaiselvi, R.S. Sabeenian, Design of area-speed efficient Anurupyena Vedic multiplier for deep learning applications. Analog Integrated Circuits and Signal Processing, 119 (2024) 521–533. https://doi.org/10.1007/s10470-024-02255-2

[8] A. A. H. Abd-Elkader, M. Rashdan, E. -S. A. M. Hasaneen, H. F. A. Hamed, FPGA-Based Optimized Design of Montgomery Modular Multiplier. IEEE Transactions on Circuits and Systems II: Express Briefs, 68 (6), (2021) 2137-2141. https://doi.org/10.1109/TCSII.2020.3040665

[9] C. C. Yen, M. Y. Yeh, M.S. Chen, Unified Multiple Constant Multipliers for Dynamic Exchange of Low-Precision Kernels on FPGAs. IEEE Transactions on Circuits and Systems II: Express Briefs, 69(3), (2022) 1617-1621. https://doi.org/10.1109/TCSII.2021.3130428

[10] S. Justin, F. Marc, Degradation Measurement and Modelling under Ageing in a 16 nm FinFET FPGA. Micromachines, 15(1), (2024) 19. https://doi.org/10.3390/mi15010019

[11] F. Spagnolo, P. Corsonello, F. Frustaci, S. Perri, Efficient implementation of signed multipliers on FPGAs. Computers and Electrical Engineering, 116 (2024) 109217. https://doi.org/10.1016/j.compeleceng.2024.109217

[12] S. Umadevi, P. Penumaka, C.K. Ram, T. Kalavathi Devi, High Performance MAC Unit Design with Grouping and Decomposition Multiplier and 18 T Gate Diffusion Input-Transmission Gate Adder. Circuits, Systems, and Signal Processing, 44(4), (2025) 2830–2854. https://doi.org/10.1007/s00034-024-02949-y

[13] R. Karthi Kumar, S.P. Vimal, Comparative analysis of Vedic multiplier using Vedic sutras with existing multipliers in biomedical application. Measurement: Sensors, 36, (2024) 101302. https://doi.org/10.1016/j.measen.2024.101302

[14] P. Paz, M. Garrido, Efficient Implementation of Complex Multipliers on FPGAs Using DSP Slices. Journal of Signal Processing Systems, 95, (2023) 543–550. https://doi.org/10.1007/s11265-023-01867-7

[15] R.K. Sakali, S. Veeramachaneni, S.K. Noor Mahammad, Preferential fault-tolerance multiplier design to mitigate soft errors in FPGAs. Integration, 93, (2023) 102068. https://doi.org/10.1016/j.vlsi.2023.102068

[16] S. Mondal, R. Marimuthu, S. Ravi, Approximate 8-bit multipliers and their physical design implementation. E-Prime-Advances in Electrical Engineering, Electronics and Energy, 6, (2023) 100300.

https://doi.org/10.1016/j.prime.2023.100300

[17] J. Charles Rajesh Kumar, D. Vinod Kumar, M.A. Majid, High-performance, energy-efficient, and memory-efficient FIR filter architecture utilizing 8x8 approximate multipliers for wireless sensor network in the Internet of Things, Memories - Materials, Devices. Circuits and Systems, 3, (2022) 100017. https://doi.org/10.1016/j.memori.2022.100017

[18] U. Penchalaiah, V.S. Kumar, A facile approach to design truncated multiplier based on HSCG-SCG CSLA adder. Materials Today: Proceedings, 46(9), (2021) 4102-4109. https://doi.org/10.1016/j.matpr.2021.02.629

[19] L. Denisov, A. Galimberti, D. Cattaneo, G. Agosta, D. Zoni, Design-time methodology for optimizing mixed-precision CPU architectures on FPGA. Journal of Systems Architecture, 155, (2024) 103257. https://doi.org/10.1016/j.sysarc.2024.103257

[20] S.E. Fatemieh, B. Bagheralmoosavi, M.R. Reshadinezhad, Energy-efficient and fast memristor-based serial multipliers applicable in image processing. Results in Engineering, 25, (2025) 104013. https://doi.org/10.1016/j.rineng.2025.104013

[21] M. Balaji, N. Padmaja, Area and delay efficient RNS-based FIR filter design using fast multipliers. Measurement: Sensors, 31, (2024) 101014. https://doi.org/10.1016/j.measen.2023.101014

[22] S.R. Kuang, C.Y. Wang, Y.J. Chen, A low-cost high-speed radix-4 Montgomery modular multiplier without carry-propagate format conversion. Engineering Science and Technology, an International Journal, 54, (2024) 101703. https://doi.org/10.1016/j.jestch.2024.101703

[23] B. Zhao, Y. Wang, H. Zhang, J. Zhang, Y. Chen and Y. Yang, 4-bit CNN Quantization Method With Compact LUT-Based Multiplier Implementation on FPGA. IEEE Transactions on Instrumentation and Measurement, 72, (2023) 1-10. https://doi.org/10.1109/TIM.2023.3324357

[24] P.H.E. Becker, A.L. Sartor, M. Brandalero, A.C.S. Beck, BRAM-based function reuse for multi-core architectures in FPGAs. Microprocessors and Microsystems, 63, (2018) 237-248. https://doi.org/10.1016/j.micpro.2018.09.007

[25] A. Ali, K. Bingi, R. Ibrahim, P.A.M. Devan, K.B. Devika, A review on FPGA implementation of fractional-order systems and PID controllers. AEU - International Journal of Electronics and Communications, 177, (2024) 155218. https://doi.org/10.1016/j.aeue.2024.155218

[26] User Guide, (2019) 7 Series FPGAs Memory Resources, UG473 (1.14).

[27] A. Ibrahim, F. Gebali, Compact modular multiplier design for strong security capabilities in resource-limited Telehealth IoT devices. Journal of King Saud University - Computer and Information Sciences, 34(9) (2022) 6847-6854. https://doi.org/10.1016/j.jksuci.2022.06.009

[28] V. Bianchi, I. De Munari, A modular Vedic multiplier architecture for model-based design and deployment on FPGA platforms. Microprocessors and Microsystems, 76, (2020) 103106. https://doi.org/10.1016/j.micpro.2020.103106

[29] V. Thamizharasan, V. Parthipan, Design of efficient binary multiplier architecture using hybrid compressor with FPGA implementation. Scientific Reports, 14, (2024) 8492. https://doi.org/10.1038/s41598-024-58482-0

Authors Contribution Statement

G. Dhanabalan: conceptualization, Methodology, Software, Writing - Original Draft. H.B. Michael Rajan: Investigation, Software, Validation, Writing - Review & Editing. R. Ashok: Visualization, Supervision. All authors have read and agreed to the published version of the manuscript.

Funding

The authors declare that no funds, grants or any other support were received during the preparation of this manuscript.

Competing Interests

The authors declare that there are no conflicts of interest regarding the publication of this manuscript.

Data Availability

The data supporting the findings of this study can be obtained from the corresponding author upon reasonable request.

Has this article screened for similarity?

Yes

About the License