Engineering and Technology Journal
Journal homepage: https://etj.uotechnology.edu.iq

Power Optimization of Binary Multiplier Based on FPGA

Fadi T. Nasser*, **Ivan A. Hashim**

Electrical Engineering Department., University of Technology-Iraq, Alsina'a street, 10066 Baghdad, Iraq.
*Corresponding author Email: eee.19.23@grad.uotechnology.edu.iq

HIGHLIGHTS

- Different multiplication algorithms are implemented using Xilinx System Generator.
- A new approach is called the VHDL approach used for dynamic power minimization.
- The reconstruction method is used to implement the multiplication algorithms.
- The multiplication algorithms are realized on FPGA using the Xilinx Spartan 3A kit.
- Dynamic power dissipation is varied proportionally with the operating frequency.

ARTICLE INFO

Handling editor: Jawad K. Ali
Keywords:
Low power multiplier
Array multiplier
Booth's multiplier
Sequential multiplier
Dynamic power optimization
Techniques for dynamic power

ABSTRACT

In the VLSI circuits, power dissipation is a critical design parameter and it plays a vital role in the performance of different digital systems. The decrease in chip size along with the increase in chip density and complexity will increase the difficulty in designing higher performance and low power digital systems. Therefore, achieving a fast and low power system is the major concern of VLSI designers. Most of the digital systems have different math operations in their architectures. This paper focuses on the multiplication operation. Multiplication requires more iterations, long time, large area, and consumes high power of the digital system compared with the other basic computation operations. Hence to improve the system's performance, it is required to design a high speed and low power multiplier. In this paper, a dynamic power dissipation is targeted; therefore, different designs of multiplier algorithms such as a sequential multiplier, array multiplier, Booth's multiplier (Radix-2), and modified Booth's multiplier (Radix-4) are proposed to investigate the design that consumes the lowest dynamic power. New techniques such as VHDL and Basic Logic Elements are presented and applied to the proposed designs. The VHDL approach satisfies the highest optimization criteria in dynamic power at 87% for the sequential multiplier than the traditional ones.

1. Introduction

In recent years, low power design of digital systems has become one of the vital concerns in the VLSI design. The primary concerns of VLSI designers are performance, area, reliability, and cost at the beginning. Achieving high-performance in digital systems leads to increase in the number of transistors over the integrated circuits (ICs). According to Moor's law, when the density increases, the size of these transistors needs to be shrunk in these ICs. Moor's law states that "over the time, the number of transistors that can be incorporated into a single die can increase exponentially" [1]. However, this exponential growth in the number of transistors results in high power dissipation. Power dissipation is considered the main motivation for VLSI designers to produce techniques for optimizing the power consumed inside the digital circuits and systems. The significant advantages of power optimization in digital systems are; battery life and battery efficiency, system reliability, noise immunity, the demand for portable systems, and system cooling and packaging cost. Power dissipation has two main types, which are static power and dynamic power. This work will focus on the dynamic power dissipation instead of static power because the dynamic power is dominant in digital circuits and the ability to apply the dynamic optimization techniques to the digital circuits due to ease of handling the logic elements and structure of the digital systems. Alternatively, it is technology-independent. In contrast, static power is technology-dependent which deals with intellectual property (IP) of manufacturing design such as transistor size, length, and width of gate oxide channel [2], which is out of the scope of this research.

Most of the digital system configurations include different math operations such as addition, subtraction, multiplication, and division in their designs. This paper focuses on multiplication operation, which is considered the heart of most computational digital systems such as digital signal processing (DSP) and their branches (IIR & FIR) filters, image processing, communication systems, etc. The repeated addition is the basic idea of a multiplication operation [3]. The multipliers have long latency, large

http://doi.org/10.30684/etj.v39i10.2156
Received 05 May 2021; Accepted 12 June 2021; Available online 25 October 2021

area overhead, and consume a large amount of power. Therefore, the reduction of power dissipation will make the performance of digital systems more reliable.

Many analyses and studies that have been made related to the most popular multiplication algorithms consider the power dissipation issue, particularly the dynamic power dissipation which is the core focus of this paper. Low power Radix-4 Booth pre-encoded was proposed in [4]. The basic idea of this technique is to reduce the switching activities of the encoder and decoder. This technique is applied to (8-bit and 16-bit) multiplier, for which the optimized dynamic power is (45 %) and 65% respectively. D. Nandan et al., obtained a total power reduction of about (39 %) by using their proposed iterative logarithmic multiplier (ILM) technique based on Mitchell's algorithm with leading one detector (LOD) and smooth pipelined technique[5]. In [6], the Gate Diffusion technique on various architecture of multipliers is proposed. The total power optimization was (35 %) of the proposed design relative to other designs. 32-bit array multiplier was suggested in [7], through which the researchers applied optimized Carry Select Adder (CSLA) to the structure of traditional multiplier to eliminate the redundant transitions; therefore, the total power saving was (41 %). As mentioned in [8], they reduced the area of Wallace multiplier by reducing the number of half adders. As the area is reduced, the power consumption is also reduced. In [9], the researchers proposed two comparative studies of two (8-bit by 8-bit) multiplier algorithms; the first one is the Booth's multiplier and the other is the Vedic multiplier in which both designs use the reversible logic gates method. The proposed design optimized (26 %) of the average power dissipation. K. R. Varma and S. Agrawal suggested a high speed, low power approximate multiplier with the use of (4-2) compressor technique. This work achieves power saving about (3.79 %) of the total power dissipation[10]. By using the operand decomposition technique, the authors made a (21 %) reduction in dynamic power dissipation for the proposed (Radix-8) modifier Booth's multiplier[11].

The aforementioned designs of multipliers were verified and implemented by using different platforms such as field-programmable gate array (FPGA) and application-specific integrated circuits (ASIC). In this paper, the proposed multipliers are implemented by using the FPGA platform. This is due to the advantages of FPGA relative to the traditional fixed logic designs (e.g. ASICs). These advantages are the possibility for performing parallel data computation [12], flexibility enabling the user to program the FPGA on the field, reusability since it can be reprogrammed many times, and faster performance where these advantages make the FPGA suitable for testing and prototyping for small and medium scale digital systems [13-15].

The main objective of this research is to reduce the dynamic power dissipation in arithmetic operations, particularly for multiplication. Consequently, many proposed designs of multipliers are introduced, and new techniques are applied to these designs to reduce the dynamic power dissipation. One of these techniques is the VHDL approach. In this approach, the dynamic power is reduced due to the reduction in switching activities and transforming the design to the basic elements that consume less power. This paper is organized as follows; the sources of power dissipation are described in the second section, power optimization techniques are explained in the third section, in the fourth section types of multiplication algorithms used in this research are given, and the proposed designs are explained in detail in section five. The simulation results and discussion are presented in the sixth section, while the conclusion is given in the seventh section.

2. Sources of Power Dissipation

Before studying how to reduce the power dissipation in VLSI circuits, it is necessary to investigate the sources of power dissipation in these circuits. Power dissipation can be classified into two categorizations; the first one is the static power dissipation, and the second is the dynamic power dissipation. The static power is taking place when the device is turned ON, but there is no task to perform. This means it is in the idle (stand by) mode, and there is no signal transition. Also, static power is known as inactive, leakage, and quiescent power. Ideally, VLSI circuits should not consume any power in this mode. Still, in practice, the transistors constantly pass some leakage current, indicating that a certain amount of power is consumed by the CMOS gates. The sources of the static power dissipation are due to seven leakage current mechanisms, which are depicted in **Error! Reference source not found.**.

In this diagram, $\boldsymbol{I_1}$ is the reverse bias p–n junction diode leakage current, $\boldsymbol{I_2}$ is the reverse-biased p–n junction current due to tunneling of electrons from the valence bond of the p region to the conduction bond of the n region, $\boldsymbol{I_3}$ is the sub-threshold leakage current between the source and the drain when the gate voltage is less than the threshold voltage $\boldsymbol{V_t}$, $\boldsymbol{I_4}$ is the oxide-tunneling current due to a reduction in the oxide thickness, $\boldsymbol{I_5}$ is the gate current due to hot-carrier injection of elections, $\boldsymbol{I_6}$ is the gate induced drain leakage (GIDL) current due to a high field effect in the drain junction, and $\boldsymbol{I_7}$ is the channel punch-through current due to the proximity of the drain and the source in short-channel devices [17].

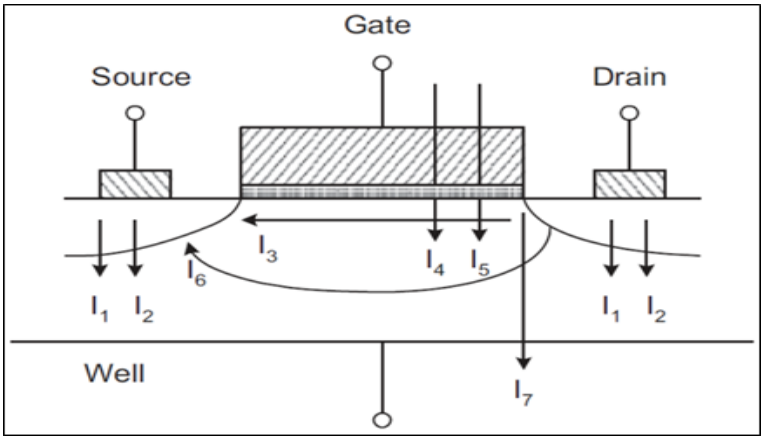

Figure 1: Sources of Static Power Dissipation [16]

The equation of the static power dissipation can be described as:

$$P_{static} = I_{leakage} \times V_{DD} \quad (1)$$

where $\boldsymbol{P_{satitc}}$ is the static power dissipation, $\boldsymbol{I_{leakage}}$ is the total current of all seven leakage current mechanisms, and $\boldsymbol{V_{DD}}$ is the supply voltage.

The second type of power dissipation is the dynamic power dissipation. The dynamic power is the power consumed while the device is operating. In other words, the CMOS device is in the active mode. This power can be classified into three types which are switching power, short circuit power, and glitch power.

The switching power is the power dissipation due to charging and discharging of the output loading capacitance. Equation (2) demonstrates the power switching activity[16-18]:

$$P_{switch} = \alpha\, C_L V_{DD}^2 f_p \quad (2)$$

where P_{switch} is the switching power, α is known as switching activity, C_L is the total load capacitance of all transistors, V_{DD} is the supply voltage and f_p is the frequency of input signal.

The short circuit power is caused by a crowbar flowing through the lapse of time when both $\boldsymbol{PMOS}$ and $\boldsymbol{NMOS}$ transistors are in the ON state. The expression of short circuit power can be given in (3) [17][19-20]:

$$P_{s.c} = \frac{\beta}{12} V_{DD}^3 \left(1 - 2\frac{V_t}{V_{DD}}\right)^3 t_{rf}\ f_p \quad (3)$$

where $\boldsymbol{\beta}$ is the current gain of the MOS transistor, and $\boldsymbol{t_{rf}}$ is the rising and falling time

The glitch power occurs when the input signals arrive at different times to a single logic block, allowing a number of intermediate transitions to occur before the logic block output stabilizes. It also occurs at the same point in the circuit when paths with unequal propagation delays converge. The glitch power is given in equation (4) below[18]:

$$P_{gltich} = \frac{1}{2} C_L V_{DD}(V_{DD} - V_t) \quad (4)$$

where, $\boldsymbol{P_{gltich}}$ is the glitch power and $V_{th}$is the threshold voltage.

Therefore, the total dynamic power dissipation is given in equation (5)

$$P_{dyn} = P_{switch} + P_{S.C} + P_{glitch} \quad (5)$$

The average power dissipation can be given in the following expression:

$$P_{avg} = P_{static} + P_{dyn} \quad (6)$$

By substituting the equations ((1, (2, (3, and (4) in equation (6), then the expression for the average power dissipation can be summarized in the following equation [20]:

$$P_{avg} = (I_{leakage} \times V_{DD}) + (\alpha\, C_L V_{DD}^2 f_p) + \left(\frac{\beta}{12} V_{DD}^3 \left(1 - 2\frac{V_t}{V_{DD}}\right)^3 t_{rf}\ f_p\right) + \left(\frac{1}{2} C_L V_{DD}(V_{DD} - V_t)\right) \quad (7)$$

3. Techniques For Dynamic Power Optimization

The classification of power optimization techniques can be categorized either by abstraction levels or depending on the type of power dissipation. In abstraction levels, the digital circuit passes through different design stages (levels). There are different abstraction levels such as system level, algorithmic level, register transfer logic (RTL) level, logic or gate level, and transistor level. High power saving can be obtained from the high levels. This means system level, algorithmic level, and RTL level in which the optimized power (either dynamic or static) is about (10-100 %), (10-90%), and (15-50 %) respectively [21]. Another

classification is based on the type of power dissipation. In this paper, the dynamic power is the targeted power to be optimized. Therefore, the spotlight will be focused only on the dynamic power techniques.

Many techniques can be used for dynamic power consumption but in this work, some of them will be studied according to the techniques applied to the proposed designs:

3.1 Operand Isolator Technique

Operand isolator is a technique based on operating transformations into equivalent computation implementations at the algorithmic level. This technique is a way of saving power for data-path operators or combinational circuits that are not completely used in each clock cycle by design. These operators execute inefficient and redundant operations, which are obviously wasting power. The fundamental concept of isolating an operator is based on eliminating unwanted operations done by the isolated operator. In other words, when non useful computation is done, the logic blocks are shut off. Shutting off is achieved when the block output is not used by disallowing the inputs to toggle in clock cycles [22].

3.2 Pre-computation technique

The pre-computation technique is a logic optimization method at logic level design [23], which tends to minimize logic transitions in combinational digital circuits by selectively pre-evaluating the output values of a combinational logic function only one-clock cycle before they are needed and then by using the pre-evaluated values to decrease internal switching activities in the next clock cycle[24].

3.3 Guarded Evaluation Design Techniques

Guarded evaluation is a technique of gate-level abstraction power optimization that is based on disabling the inputs of complex combinational circuits or data-path systems to reduce the transition when these inputs do not relate to the generation of output for a given input vector. In other terms, if an output is not detected under such situations, i.e. if it has, observable don't care (ODC) situations, then it is possible to insert transparent latches or floating gates at the required input [25].

3.4 Operation reduction technique

The reduction of the operation is a technique based on operating transformations into equivalent computation implementations at the algorithmic level or can be known as register transfer logic (RTL) level. In this technique, the aim is to optimize type, number, interconnection, and the sequencing of computational modules while retaining the input/output behavior. This is a convenient way to reduce the switching capacitance in a circuit [29].

3.5 Operation Substitution Technique

Switching capacitance is minimized in the operation substitution technique by substituting high power operations with low power operations in the data flow graph (DFG) at the algorithmic (RTL) level [26]. An addition operation that needs an adder, for example, is less power-consuming than a multiplication operation that needs a multiplier. When the number of multiplication operations in the data flow graph is reduced, this will lead to a reduction in the switching capacitances, and thus the dynamic power dissipation will decrease.

3.6 Parallelism Technique

The basic principle of this technique is to use several hardware resource copies, like processors and arithmetic and logic units (ALUs), to work in parallel in order to provide maximum performance. Usually, the parallelism technique is used to improve the performance at the expense of increasing chip area and higher power consumption. Instead of improving the performance, parallel processing is also used to reduce the power. As it is known, the scaling of the supply voltage is the most effective way to decrease power consumption. The power saving will lead to reduction in performance, or more accurately, maximizing the operating frequency. At the Architecture level, this technique can be used [31].

3.7 Pipelining Technique

Pipeline technique can also be implemented on the architecture level. It is an execution process in which various tasks are carried out in a covered manner. This technology is used for an advanced digital system that uses microprocessors, where the microprocessor starts executing the second instruction when the first instruction is still in the processing stage. In the pipeline technique, the delay through the critical path of the digital circuit is minimized instead of decreasing the clock frequency so that the supply voltage can be decreased to reduce the power. This technique has the advantage of area-efficient over the parallelism technique [27].

4. Multiplication Algorithms

The basic idea of a multiplication operation is based on repeated addition. Multiplication plays a vital role in most high-performance systems. At the same time, the multipliers are very complicated circuits with high cost. There are two ways to implement the hardware of multiplication. The first one is by using huge resources to achieve high performance, which consumes high power (leading to fast execution). On the other hand, the second way uses fewer hardware resources, which consumes less power with high delay (which means slow execution). The multiplication operation is more complicated than the addition operation due to the fact that multiplication includes two operations, addition and shifting. Multiplicand and multiplier are the

two components of multiplication. There are many algorithms to implement the multiplication, such as classical (paper and pencil calculation) algorithm, array multiplier algorithm, Booth's multiplication algorithm, etc. [3].

4.1 Sequential Multiplier

The internal construction of this multiplier is a sequential circuit that uses a single *n*-bit adder to calculate the product of any two binary numbers as shown in Figure 2. In this sketch, *A* and *B* represent binary numbers with length of *n* and *m* bits, respectively. The operation of this sequential circuit is to compute the partial products one time at each cycle and repeats the operation *m* times. At each cycle, the partial products will be produced, then summed to an accumulated partial sum and the resulting partial sum is shifted to the right one-bit position to align the accumulated sum with a partial product of the next cycles [28]. Therefore, each cycle of a sequential multiplication operation consists of three arithmetic operations:

A. Producing partial product.
B. Summing the produced partial products to the accumulated partial sum
C. Shifting the partial sum.

The drawback of this method is that it consumes a lot of resources. Moreover, when the value of multiplication is very high (i.e., there are many partial products or many bits to be summed), the performance will slow down.

4.2 Array Multiplier

Error! Reference source not found. 3 shows the array multiplier of a 4-bit × 4-bit algorithm. The construction of array multiplier is based on an array of full-adders (FAs), half-adders (HAs), and AND gate blocks to compute the result of the multiplication. The basic idea of this algorithm is based on Add-Shift operations that can be applied to any number system, including the binary number system. The partial product at each cycle is produced by multiplying each bit of the multiplier with the entire bits of the multiplicand. The number of AND gates represents the partial products (i.e. No. of AND gates = 4×4=16), $the\ number\ of\ (n-bit)\ adder\ =\ the\ number\ of\ (n\text{-}bit)\ -1$ (i.e.$No.of\ 4bit\ adder\ =\ 4-1=\ 3$), and the number of partial products (final product bits) equals to the sum of multiplier bits with multiplicand bits (i.e. $No.of\ partial\ products\ =\ 4\ +\ 4\ =\ 8\ bits$), as shown in **Error! Reference source not found.**. Multiple partial products (B0A0, B0A1=X0, B0A2=X1, B0A3=X2, etc.) are resulted from each individual multiplication and these partial products are obtained by shifting one-bit position to the right and finally summing all partial products to obtain the final result of the multiplication (i.e. P0 to P7). This type of algorithm is used only for unsigned binary multiplication [29].

4.3 Booth's Multiplier Algorithm

Booth's algorithm is the most common algorithm used to implement the multiplication operation only by using shift and addition or subtraction operations. Andrew Donald Booth proposed this algorithm in 1951 when he was researching crystallography at Birkbeck College in Bloomsburg. Booth's algorithm is suggested to solve the waiting problem of the partial products by cutting the number of partial products into half. Booth's algorithm has the advantage of accelerating the multiplication operation in comparison to the classical method in which it can be applied in the multiplication of signed numbers without using any transformation. The hardware components of this algorithm are A, M, Q, and Count, which represent accumulator, multiplicand, multiplier, and counter registers, respectively. And one-bit register placed to the right of multiplier Q donated by Q_{-1}. The procedure of Booth's algorithm is described in details as follows [3]:

1 Register initialization: where A and Q_{-1} registers are initialized with zeros, M is initialized with multiplicand value, and count register is the number of bits in multiplicand (M) or multiplier (Q) registers.
2 By comparing the least significant bit (Q_0) of Q register with the Q_{-1}:
3 If $Q_0\ =\ Q_{-1}$ either for (0-0) or (1-1), then do nothing.
4 If $Q_0\ \neq\ Q_{-1}$ then there are two cases:

a. Case1: if $Q_0\ =\ 0$ and $Q_{-1}\ =\ 1$then add multiplicand (M) to the accumulator (A).
b. Case2: if $Q_0\ =\ 1$ and $Q_{-1}\ =\ 0$, then subtract the multiplicand (M) from accumulator (A).

5 a. Applying the arithmetic shift right to the accumulator, multiplier, and Q_{-1} for step (2.b).
b. $Count\ =\ Count\ -\ 1$, if $Count\ \neq\ 0$, then repeat steps from 2 to 3. Otherwise, end the algorithm and the result placed in the combined register AQ register.

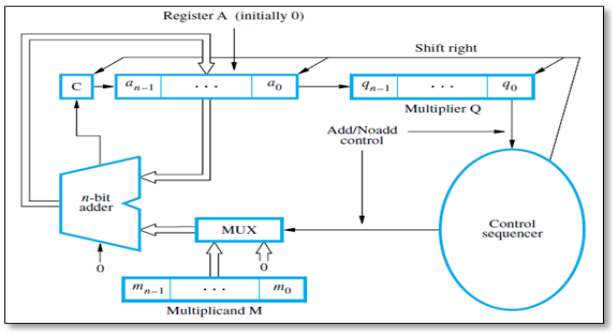

Figure 2: Hardware implementation of the sequential multiplier [28]

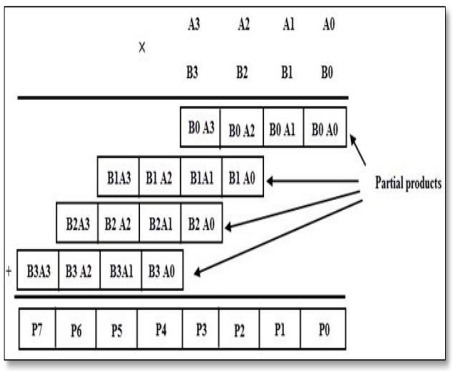

Figure 3: Array Multiplier 4-bit x 4- bit algorithm [30]

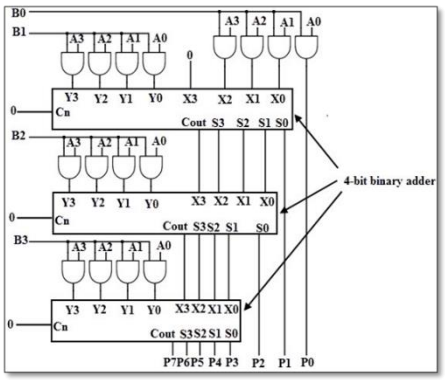

Figure 4: Array Multiplier 4-bit x 4-bit implementation [31]

4.4 Modified (Radix-4) Booth's Multiplication Algorithm

Modified Booth's algorithm, which is also known as Radix-4 Booth's algorithm, has three main steps: the first step is to produce the partial product (recoding). The second step reduces the number of partial products, and the final step is the addition operation. The Radix-4 algorithm is an efficient way to increase the multiplication efficiency because it divides the multiplier into overlapping groups. Each group with three adjacent bits, which means there are eight states and according to the adjacent three bits of multiplier, the modified Booth's algorithm can produce the appropriate coefficient (M_i), in which M_i has five possible values ($\pm 1, \pm 2 \; or \; 0$); these coefficients have recoded the multiplicand, as depicted in TABLE 1[6]. The advantage of this algorithm is that the partial product can be reduced by half. This will reduce the circuit complexity. As the complexity is reduced, the power dissipation is also reduced. This algorithm can be used for signed and unsigned numbers.

The steps of the modified Booth's algorithm are described as follows[4]:

1 Check the number of bits for multiplicand and multiplier if they are equal;

a. If they are even, there is no need to add zeros.

b. If one of them is odd, (n-bits) either 0s or 1s should be added to the left (for multiplier or multiplicand) according to the number of highest bits.

2 Initialize the LSB bit of multiplier (Q_{i-1}) with zero.

3 Overlapped grouping should take place for three adjacent bits (Q_{i+1}, Q_0, Q_{i-1}) of the multiplier,

4 The multiplicand is recoded by using the Radix-4 encoding table (see TABLE 1), which can take five possible values ($\pm 1, \pm 2 \; or \; 0$) to find the partial products.

5 To get the final result of multiplication, an addition operation should take place for all the partial products.

5. The Proposed Design of Low Power Multipliers

The multipliers are considered as the core of many digital systems. Due to the long delay, large area, and high power consumption of these multipliers, the designers try to use reduction techniques to optimize the power dissipation. In this section, seven proposed designs are implemented in different algorithms to investigate which one of the proposed designs has the lower dynamic power dissipation. Besides, new techniques are applied to each algorithm to achieve the best power optimization of the multiplier. Alternatively, to reach the optimal power multiplier, two procedures are involved; the first is implementing different structure designs, and the second is applying the power optimization techniques. In this paper, new techniques such as the Basic logic Elements technique and VHDL approach are applied to the proposed designs to reduce the dynamic power dissipation. To evaluate the power optimization percentage, the sequential multiplier using Hard FPGA blocks is considered as the reference design with the highest power dissipation to compare it with the powers of the other proposed designs. All the proposed multipliers are implemented by using Xilinx System Generator (XSG) software which is resulted in the configuration of MATLAB 2012a and ISE 14.7 software. These multipliers have the same width of multiplicand and multiplier, which is 32 bits for each. The designs are verified and simulated by using Xilinx Spartan 3A-3N/XC3S700a/-4/fg484 FPGA platform.

5.1 Sequential Multiplier Using Hard FPGA Blocks

The proposed sequential multiplier is implemented by using Hard FPGA blocks. In contrast to the original sequential multiplier, which is restricted to unsigned numbers, this design can be used for signed and unsigned numbers as well.

Error! Reference source not found. shows the construction of this design which is consist of A register represents the accumulator register and CAQ register, which is combined the accumulator register (A register), quotient register (Q register), and C bit which is the most significant bit of A register. The multiplier (Q) and multiplicand (M) represent the inputs that the user can determine. These inputs are driven to Xilinx Absolute and Absolute1 blocks t take the absolute values. The Xilinx Absolute block output is connected to the first input ($d0$) of the Mux1 block. When the design is running at the first cycle, the state of the counter is equal to zero, and the output of the Relational block is also zero. Then, the multiplier value will be passed through the Mux1 block and stored in the Register block. The least significant bit (Q_0) will be extracted from the Register block using Slice block. Q_0 represents the condition of the sequential multiplier algorithm, and there are two cases for that: the first

one when $Q_0 = Sel = 0$, then the CAQ register content is one-bit right-shifted using BitBasher1 block. The second when $Q_0 = Sel = 1$, then the multiplicand will be added to the high content of the CAQ register (CAQ High), and the partial product is shifted one bit to the right. The output of the Mux2 block will be shifted using the BitBasher block and then stored in the CAQ register. To enable this design to operate with signed and unsigned numbers, the most significant bits (MSBs) of each input XORed using Logical xor block. The output of the Logical block will operate as a select line to choose the positive or negative value of the final result outcome from the CAQ register. The result of this design can be obtained after 33 cycles.

5.2 Sequential Multiplier Using VHDL Code

The exact sequential algorithm of the previous design is applied to this design. Alternatively, this design is implemented based on the VHDL approach. The VHDL code is written using ISE 14.7 package and imported to XSG using the Black Box block, as shown in Figure 6. The flowchart of the proposed sequential multiplier is demonstrated in Figure 7. From this figure, the multiplier (A), the multiplicand (B), the AQ register, AQ [0], and the counter are initialized with zero. Each one with its corresponding bit(s). after that, the AQ [0] bit must be checked, representing the least significant bit of the AQ register, and there are two cases. If AQ [0] =0, then no operation is performed, and if AQ [0] =1, then add the absolute value of multiplicand (|B|) to the high content of AQ register (i.e., AQ [63:32] +B2). After that, the entire content of the AQ register will be shifted one bit to the right, and the counter is incremented by one. The counter is checked; if the counter ≠ 32, repeat the steps from checking the AQ [0] bit, and if the counter = 32, then proceed to the next step. The next step is to check the sign bit, which is equal to the XORing of the MSBs of the multiplier and multiplicand (i.e., S=B [31] XOR A [31]). If S=1, then 2's complement is taken for the content of AQ register, and if S=0, then the algorithm is ended, and the result will be obtained on the Display block, as shown in Figure 6.

Table 1: Radix-4 Booth's Algorithm Recoding [4]

Q_{i+1}	Q_i	Q_{i-1}	Partial products
0	0	0	$+0 * Multiplicand$
0	0	1	$+1 * Multiplicand$
0	1	0	$+1 * Multiplicand$
0	1	1	$+2 * Multiplicand$
1	0	0	$-2 * Multiplicand$
1	0	1	$-1 * Multiplicand$
1	1	0	$-1 * Multiplicand$
1	1	1	$-0 * Multiplicand$

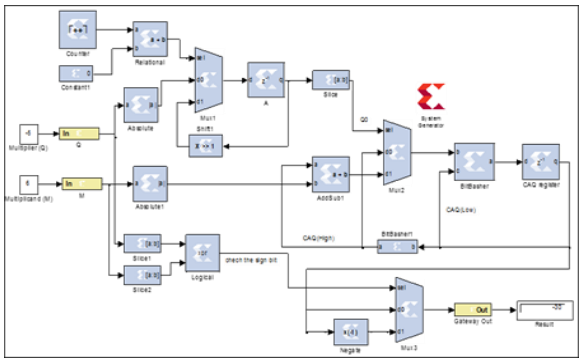

Figure 5: The Proposed Sequential Multiplier Using XSG Blocks

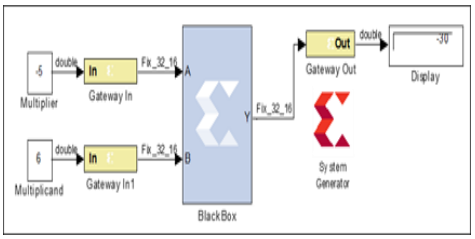

Figure 6: Proposed design of Sequential Multiplier using VHDL code

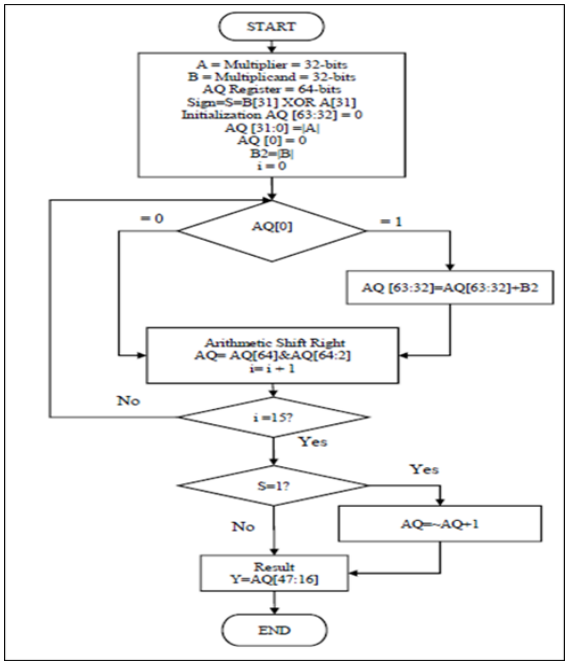

Figure 7: Flow chart of the proposed sequential multiplier

5.3 *Xilinx LogiCORE IP* Multiplier

This design is implemented by using ***Xilinx LogiCORE^{TM} IP Multiplier***. This multiplier can be found in the Math and Index Xilinx Block-set libraries. Xilinx Mult Block represents a multiplier in which the calculations of the data can be done on its two inputs and shows the results on its output, as shown in **Error! Reference source not found.**. High-performance and optimized multipliers are implemented by **Xilinx LogiCORE IP** multiplier. There are several options to compromise between resources and performance to adapt the core to a given application. The feature of this multiplier can be configured in either parallel architecture or constant-coefficient architecture. In parallel architecture, the multiplier takes two inputs and produces the product of these two multiplied numbers. In constant-coefficient architecture, the multiplier takes one input (a) and multiplies it by a constant value determined by the user. The inputs ranging from (1) to (64) bits wide and the outputs ranging from (1) to (128) bits wide. The latency for all multiplier inputs is configurable [32]. This multiplier is used for signed and unsigned numbers.

5.4 Array Multiplier Using Basic Elements

The proposed array multiplier is implemented by using XSG blocks. In this multiplier, the full adders (FAs) and half adders (HAs) are built by using the basic logic elements such as AND, XOR, and OR gates. These basic logic elements have low power consumption due to lower switching activity and the load capacitance is minimum relative to the traditional design. The proposed design is characterized by the use of signed and unsigned numbers, as illustrated in Figure 9, opposite to the traditional array multiplier therefore to enable this design to operate with signed and unsigned numbers, the most significant bits (MSBs) of each input (Q and M) XORed using Logical xor block. The output of the Logical block will operate as a select line to choose the positive or negative value of the final result outcome from the 32 bit x 32 bit Sub-System block. The proposed signed array multiplier according to the design rules of the traditional array multiplier consists of 1024 AND gates which represent the partial products, (31) of a 32-bit adder, and 64-bits representing the bits of the final product as shown in 0.

0 shows the construction of the 32 bit x 32 bit Sub-System block, where each bit of the M is ANDed with each bit of the Q in as many levels as there are bits in the M. The binary output in each level of AND gates is added with the partial product of the previous level using the 32 bit Adder block to form a new partial product. The last level produces the product that represented by 32bit Adder 31 block.

5.5 Booth's Multiplier by Hard FPGA Blocks

According to Booth's multiplication algorithm steps, the proposed Booth's multiplier is implemented using XSG blocks. This is a unique implementation with the use of Simulink software. As depicted in Figure 11, the construction of this is consist of the accumulator (A) register and the combined accumulator and quotient (AQ) register that produces the final value. The multiplier value is directly driven to the BitBasher1 blocks instead of using the Absolute block. This value will be concatenated with the Constant block to initialize the A register. At the running of the first cycle, the Counter block starts counting from (0) to (31). Therefore, the output of the Relational block, at the first cycle, is logic 0. The Mux1 block passes the concatenated value of the multiplier and stores it in the A register. For the remaining counts (i.e., 1 to 31), the Mux block will choose the A register's right-shifted value. After that, the content of the A register will be sliced using the Slice block to extract the least two significant bits (i.e., Q_1 and Q_0). The Q_1 and Q_0 bits represent the three conditions of Booth's algorithm. The arithmetic operation is done by comparing the Q_1 bit and Q_0 bit. The three conditions are: If Q_1, Q_0= 00 or 11, then no operation is done, if Q_1, Q_0=01, then the multiplicand is added to the high content of the AQ register (i.e., AQ High) using the AddSub1, or if Q_1, Q_0=10, then the

multiplicand is subtracted from the AQ. High using the AddSub block. The output of the Mux2 block will be one bit shifted to the right using the BitBasher2 block, and the result will be stored in the AQ register. The obtained result can be shown on the Display block. The performance of this design takes 33 cycles to obtain the final result.

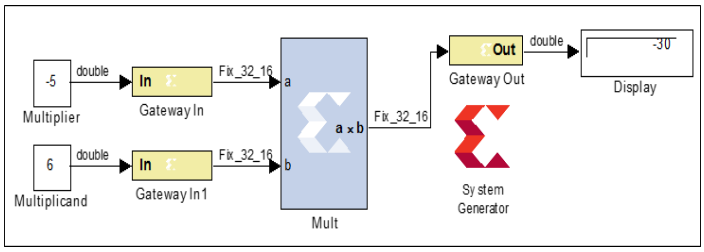

Figure 8: Proposed design using Xilinx LogiCORE^TM IP Multiplier

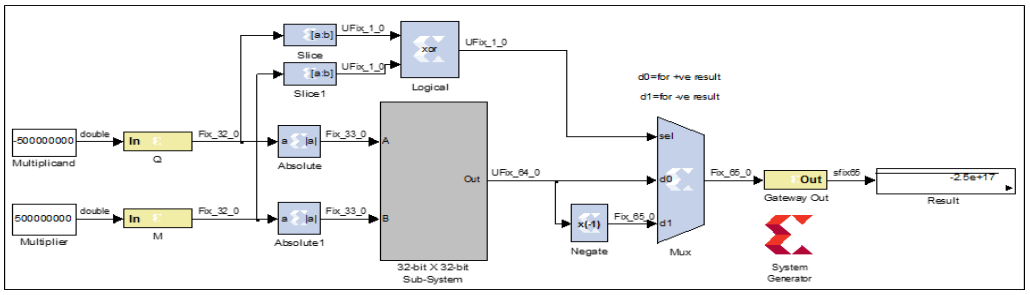

Figure 9: Proposed Design of Array Multiplier Using Basic Elements XSG Blocks

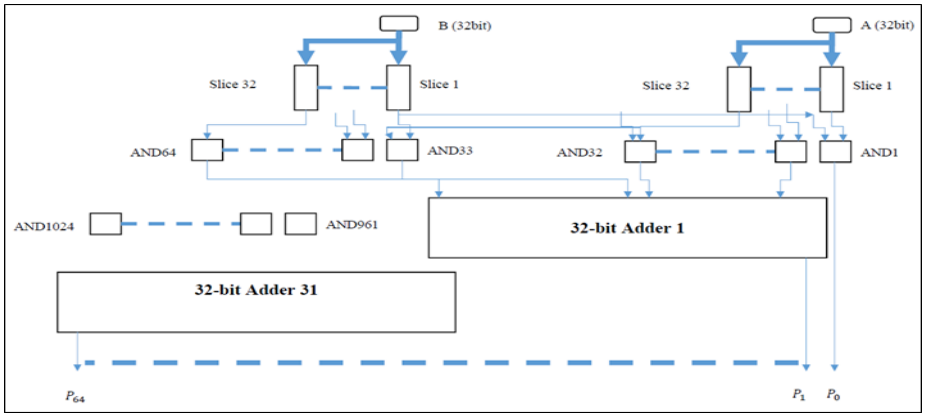

Figure 10: Subsystem of 32x32 bit array multiplier

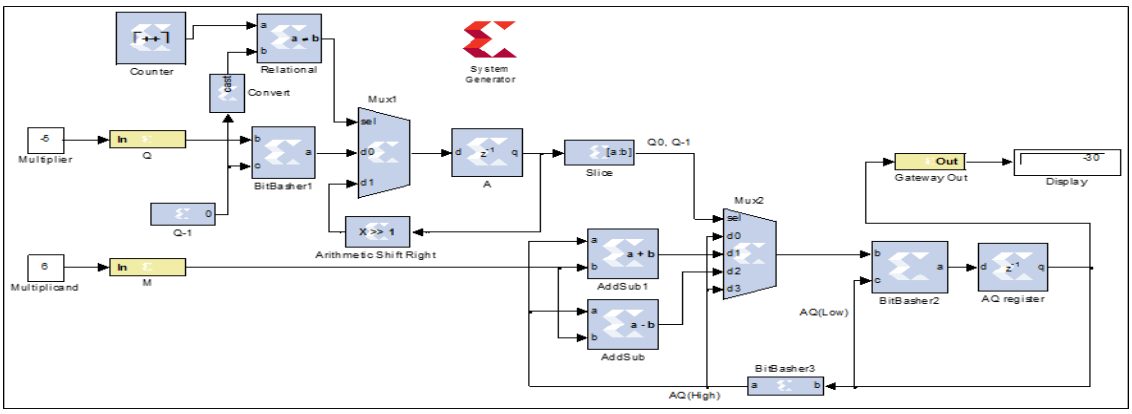

Figure 11: Proposed Booth's multiplier using XSG blocks

5.6 Booth's Multiplier with VHDL Code

The proposed design of Booth's multiplier is implemented by using VHDL code instead of using XSG blocks as in the previous design. The VHDL source code has been written by using ISE 14.7 and exported to the Xilinx Black Box block as illustrated in Figure 12. According to Booth's algorithm procedure that depicted in 0, the first step is initializing the multiplier (A), the multiplicand (B), the high content of the AQ register (AQ High), and AQ [0] with zeros, each one corresponding to its bit(s). In addition, initializing the low content of the AQ register with the value of multiplier (i.e., AQ [32:1] =A). The next step is to compare the AQ [1] and AQ [0] bits. Three cases resulted from this comparison. The cases are AQ [1], AQ [0] = 00 or 11,

01, and 10, representing no operation performed, adding the multiplicand to the AQ. High subtracting the multiplicand from the AQ. High, respectively. After that, the result will be right-shifted one bit (i.e., AQ=AQ [64] & AQ [64] & AQ [64:1]), and the counter is incremented by one. Checking the counter, if counter ≠ 32, then repeat the process of reviewing the AQ [1] and AQ [0], and when the counter =32, then end the algorithm, and the result will be presented on the Display block.

5.7 Modified Booth's Algorithm Using VHDL Code

A VHDL code is used to implement the proposed design of modified Booth's multiplier. Black Box block is used to import the VHDL code written in ISE 14.7 of the proposed modified Booth's multiplier, as shown in Figure 14. In this design, the number of counts is reduced to half (i.e. 15 counts) according to the Radix-4 algorithm. This reduction in the number of counts will reduce the number of switching activities and redundant transitions. After one cycle, the result can be obtained.

The flowchart of modified Booth's multiplier is illustrated in **Error! Reference source not found.**. Initialization the multiplier, the multiplicand, the least significant bit of the Q register (AQ [0]), and the counter with zeros, each one with its corresponding bit(s); initializing the high content of the AQ register with the value of multiplier (i.e., AQ [32:1]=A); and initialize the additional register (B2) with the right-shifted value of the multiplicand (i.e., B2=B[30:0]&0), all these represent the first step of the algorithm. This algorithm is scanning the last three bits of the AQ register instead of two bits as in the original Booth's algorithm mentioned in the previous subsection. This means the comparison will be made between the least significant bit (AQ [0]) and the other two least significant bits (AQ [2] and AQ [2]), and this comparison leads to reduce the number of iterations to half, which means from 32 to 15. According to these three bits comparison, five cases will be there: the first three cases when AQ [2], AQ [1], and AQ [0]= (000 or 111), (001 or 010), and (101 or 110), representing no operation is performed, the multiplicand (B) is added to the AQ. High register (i.e., AQ [64:33]= AQ [64:33]+B), and the multiplicand is subtracted from the AQ. High register (i.e., A.Q. [64:33]= A.Q. [64:33]+B), respectively. In the fourth case, when AQ [2], AQ [1], and AQ [0] = 011, then the shifted multiplicand (B2) is added to the AQ High register (i.e., AQ [64:33] = AQ [64:33] +B2). Where the fifth case when AQ [2], AQ [1], and AQ [0] = 100, then the shifted multiplicand is subtracted from the AQ High register (i.e., AQ [64:33] = AQ [64:33]-B2). After that, the arithmetic right shift is performed, and the counter is incremented by one. The next step is checking the counter. If the counter ≠15, then repeat the scanning process to the three bits, and if the counter = 15 (it's the half number of the 32 counts), then end the algorithm, and the result will be shown at the Display block.

6. Simulation Results and Discussion

In this paper, seven proposed designs of multiplier algorithms were implemented by using different approaches. Each multiplier has multiplicand and multiplier widths of 32-bits. All the proposed designs are verified by using Xilinx Spartan 3A-3N/XC3S700a/-4/fg484 FPGA platform. Since some of the proposed designs were implemented by Xilinx System Generator (XSG) block sets that can be obtained from the configuration of MATLAB R2012a and ISE 14.7, the simulator and the other designs can be implemented by means of the VHDL method. The VHDL code can be considered as a new approach for power optimization and particularly dynamic power. The dynamic power is targeted in this research. X-power Analyzer software was used for estimating the performances of the proposed designs in terms of total power, dynamic power, and area overhead. In this work, two comparisons were made in terms of power analysis. The first one is the comparison of the highest power for the proposed sequential multiplier using Hard FPGA blocks with the other six proposed designs as shown in TABLE 2. The second comparison is between the related works and the best optimal power multiplier as listed in TABLE 3. From TABLE 2, the comparison is made between the proposed design of the sequential multiplier by using hard FPGA blocks since it is considered as a reference design and the other six proposed designs in terms of dynamic power and total power. From this comparison, it can be found that the optimization percentages of dynamic power and total power respectively are (19.62, 11.65) % of Booth's multiplier using hard FPGA blocks, (49.73, 29.52) % of array multiplier by using basic elements hard FPGA blocks, (77.8, 46.18) % of Xilinx LogiCore multiplier, (80.31, 47.67) % of Booth's multiplier using VHDL approach, (82.24, 48.81) % of modified Booth's multiplier using VHDL approach, and (87.35, 51.86) % of the sequential multiplier using VHDL approach.

Figure 16 shows the area overhead of the seven proposed designs in which the Booth's multiplier using the VHDL approach has the highest number of look-up tables with (3489) LUTs. The proposed array multiplier has the highest number of occupied slices with (1985) slices. The highest number of flip-flops (D-FF) has been found in the sequential multiplier and Booth's multiplier of hard FPGA blocks with (102) flip-flops for each and zero flip-flops for the other designs.

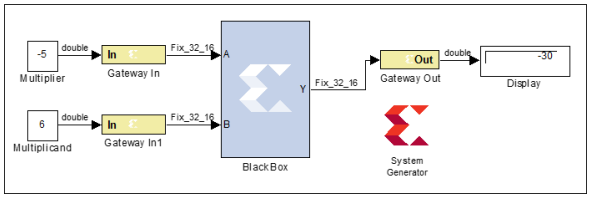

Figure 12: The proposed design of Booth's algorithm using VHDL code

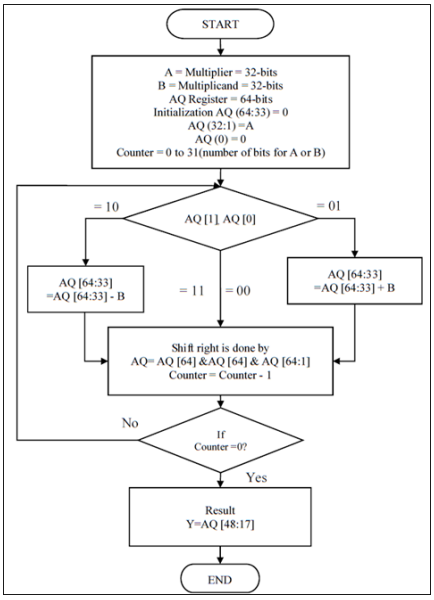

Figure 13: Flowchart of the Booth's multiplier using VHDL

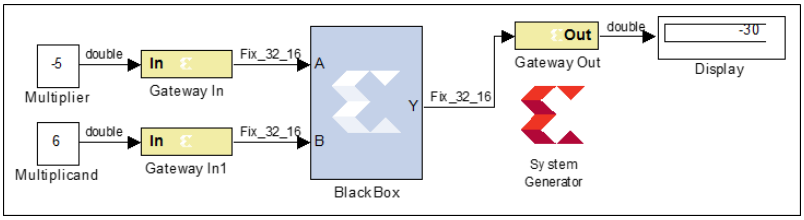

Figure 14: The proposed design of the Modified Booth's algorithm using VHDL code

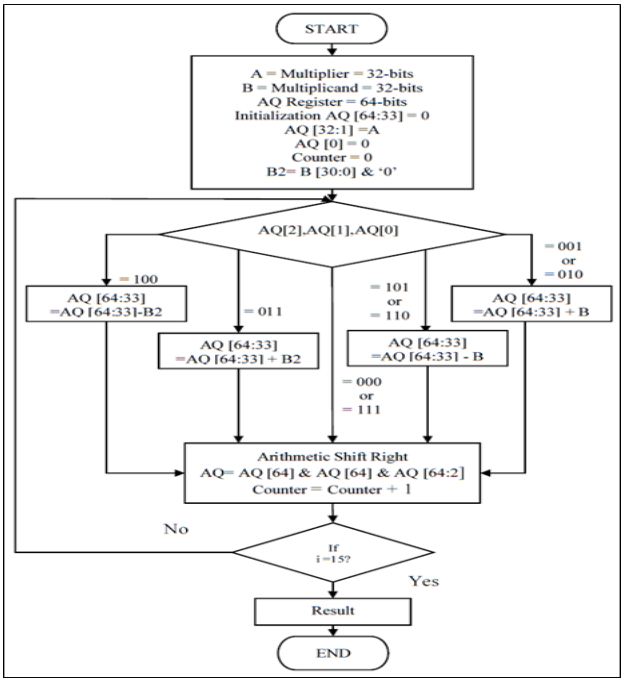

Figure 15: Flowchart of the modified Booth's algorithm

Table 2: Power dissipation comparison of the proposed designs

Multiplier Type	Dynamic Power Dissipation (mW)	Static Power Dissipation (mW)	Total Power Dissipation (mW)	Percentage of dynamic power optimization	Percentage of total power optimization
Sequential Multiplier Using Hard FPGA Blocks	45.77	31.77	77.54	-	-
Booth's Multiplier Using Hard FPGA Blocks	36.79	31.72	68.51	19.62 %	11.65 %
Array Multiplier Using Basic Elements of Hard FPGA Blocks.	23.01	31.64	54.65	49.73 %	29.52 %
Xilinx LogiCORE^{TM} IP Multiplier	10.16	31.57	41.73	77.80 %	46.18 %
Radix-4 Booth's Multiplier Using VHDL Code	9.01	31.57	40.58	80.31 %	47.67 %
Modified Booth's Multiplier Using VHDL Code	8.13	31.56	39.69	82.24 %	48.81 %
Sequential Multiplier Using VHDL Code	5.79	31.55	37.33	87.35 %	51.86 %

Table 3: Comparison of the proposed Sequential Multiplier Using VHDL with the related works

Multiplier Type	No. of bits	Dynamic Power Optimization percentage (%)	Total Power Optimization percentage (%)
[4]	8	45%	-
	16	65%	-
[11]	8	21%	-
[5]	8	-	17.8%
	16	-	26%
	32	-	39%
[6]	32	-	35%
[7]	32	-	41%
[9]	8	-	26%
[10]	4	-	3.79%
Proposed Sequential Multiplier Using VHDL Approach	32	87.35%	51.86%

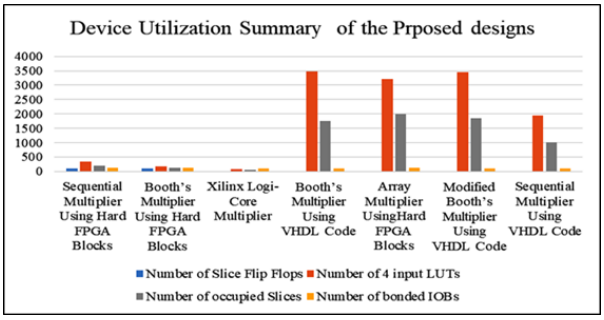

Figure 16: Device Utilization Summary of the proposed designs

The proposed design of sequential multiplier with the VHDL approach has the lowest dynamic and total power consumptions than the others. This decrease in power is due to the VHDL approach to implement the proposed design using the basic element components. Moreover, the steps of the multiplier algorithm for the proposed design take only one cycle for execution. This means there is no latency (delay) and this will lead to the minimization of the critical paths. In addition, there is also internal optimization handling by the placing and routing phases. Besides, all the above-mentioned zero flip-flops lead to increasing the switching activities. On the other hand, the proposed design of sequential multiplier using hard FPGA blocks takes less area overhead in terms of the number of LUTs and number of I/O blocks, but at the same time, the design consumes many flip-flops as illustrated in Figure 16. These flip-flops will increase the switching activities of the design. The result of multiplication for this proposed design will be obtained after (33) cycles, which means it is slower than the proposed design by the VHDL approach. The delay means there are a lot of critical paths which will result in high power consumption.

TABLE 3 represents the second comparison of the proposed sequential multiplier by using the VHDL approach with the related works. It can be shown from this table that the proposed design has the highest percentage of dynamic power consumption

optimization relative to [4], [11] and also has the highest percentage in terms of power consumption optimization compared with [3] [6], [7], [9], and [10].

7. Conclusion

In this work, several 32-bit by 32-bit multipliers have been designed and implemented. Four of these designs were implemented using Hard FPGA blocks, while three were implemented using the VHDL code approach.

From the performance analysis in terms of dynamic and total power dissipation, it can be concluded that the proposed designs with the use of Hard FPGA blocks consume high power dissipation in regard to the dynamic power and total power. On the other hand, the proposed designs with the use of the VHDL approach have minimum dynamic and total power dissipation. This reduction is due to decreasing or eliminating the switching activities and low critical path delays in these designs.

Therefore, as a conclusion, the most efficient technique to obtain higher power savings can be achieved by using the VHDL approach, where the power savings are about (87 %) and (52%) for dynamic and total power optimization, respectively in the case of the proposed design of sequential multiplier using VHDL code. The latter design achieves the highest power optimization compared to other proposed designs and the existing designs in the literature. The VHDL approach has two main advantages; the first is transforming the design to its basic logic elements that consume less power. the second, it has a slight delay. whereas the previous works either consume significant resource utilization or have a long execution time.

Author contribution

All authors contributed equally to this work.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data availability statement

The data that support the findings of this study are available on request from the corresponding author.

Conflicts of interest

The authors declare that there is no conflict of interest.

References

[1] G. E. Moore, Cramming more components onto integrated circuits. McGraw-Hill New York, NY, USA, (1965).

[2] T. L. Floyd, Digital Fundamentals. Pearson Education India, (2010).

[3] F. W. Wibowo, Comparison of multiplication algorithms based on FPGA, in 2018 2nd Borneo International Conference on Applied Mathematics and Engineering (BICAME), (2018), 326–331.

[4] Y.-J. Chang, Y.-C. Cheng, S.-C. Liao, and C.-H. Hsiao, A Low Power Radix-4 Booth Multiplier With Pre-Encoded Mechanism, IEEE Access, 8, (2020), 114842–114853.

[5] D. Nandan, J. Kanungo, and A. Mahajan, An efficient architecture of iterative logarithm multiplier, Int. J. Eng. Technol.,7 (2018), 24–28, doi: 10.14419/ijet.v7i2.16.11410.

[6] B. Mukherjee and A. Ghosal, Design and Analysis of a Low Power High-Performance GDI based Radix 4 Multiplier Using Modified Booth Wallace Algorithm, in 2018 IEEE Electron Devices Kolkata Conference (EDKCON), (2018), 247–251.

[7] N. F. Afreen, M. M. Basha, and S. M. Das, Design and implementation of area-delay-power efficient CSLA based 32-bit array multiplier, in 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), (2017), 1578–1582.

[8] S. Kakde, S. Khan, P. Dakhole, and S. Badwaik, Design of area and power aware reduced Complexity Wallace Tree multiplier, in 2015 International Conference on Pervasive Computing (ICPC), (2015), 1–6.

[9] H. P. Patil and S. D. Sawant, FPGA Implementation of conventional and vedic algorithm for energy efficient multiplier, in 2015 International conference on energy systems and applications, (2015), 583–587.

[10] K. R. Varma and S. Agrawal, High speed, Low power Approximate Multipliers, in 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), (2018), 785–790.

[11] R. Mudassir, M. Anis, and J. Jaffari, Switching activity reduction in low power Booth multiplier, in 2008 IEEE International Symposium on Circuits and Systems, (2008), 3306–3309.

[12] K. Paulsson, M. Hübner, and J. Becker, Dynamic power optimization by exploiting self-reconfiguration in Xilinx Spartan 3-based systems, Microprocess. Microsyst., 33, 46–52, (2009).

[13] I. A. Hashim, An FPGA Based a Digital Circuit Design for Route Optimization, Eng. Tech. J., 30, (2012).

[14] I. A. Hashim, Design and Implementation of a Two Stage Controller for Ball and Beam System Using FPGA, Eng. Technol. J., 36, (2018).

[15] I. Kuon and J. Rose, Measuring the gap between FPGAs and ASICs, IEEE Trans. Comput. Des. Integr. circuits Syst., 26, (2007), 203–215.

[16] N. P. Kumar, B. S. Charles, and V. Sumalatha, A Review on Leakage Power Reduction Techniques at 45nm Technology, Mater. Today Proc., 2, (2015), 4569–4574.

[17] A. Pal, Sources of Power Dissipation, in Low-Power VLSI Circuits and Systems, Springer, (2015), 141–173.

[18] R. Y. Reetu, Dynamic power reduction of VLSI circuits: a review, Int. J. Adv. Res. Electron. Commun. Eng, 7 (2018) 245–259.

[19] M.-B. Lin, Introduction to VLSI systems: a logic, circuit, and system perspective. CRC press, (2011).

[20] A. Ben Abdallah, Power Optimization Techniques for Multicore SoCs, in Advanced Multicore Systems-On-Chip, Springer, (2017), 225–244.

[21] M. Arora, The art of hardware architecture: Design methods and techniques for digital circuits. Springer Science & Business Media, (2011).

[22] A. Pal, Low-Power VLSI circuits and systems. Springer, (2014).

[23] S. Devadas and S. Malik, A Survey of Optimization Techniques Targeting Low Power VLSI Circuits Srinivas, 32nd ACM/IEEE Des. Autom. Conf., 1995.

[24] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou, Precomputation-based sequential logic optimization for low power, IEEE Trans. Very Large Scale Integr. Syst., 2, (1994), 426–436.

[25] E. Macii, M. Pedram, and F. Somenzi, High-level power modeling, estimation, and optimization, IEEE Trans. Comput. Des. Integr. circuits Syst., 17, (1998), 1061–1079.

[26] S. P. Mohanty, N. Ranganathan, E. Kougianos, and P. Patra, Low-power high-level synthesis for nanoscale CMOS circuits. Springer Science & Business Media, (2008).

[27] G. Verma, M. Kumar, V. Khare, and B. Pandey, Analysis of low power consumption techniques on FPGA for wireless devices, Wirel. Pers. Commun., 95, (2017), 353–364.

[28] W. Stallings, Computer organization and architecture: designing for performance. Pearson Education India, (2003).

[29] S. Srikanth, I. T. Banu, G. V. Priya, and G. Usha, Low power array multiplier using modified full adder, in 2016 IEEE International Conference on Engineering and Technology (ICETECH), (2016), 1041–1044.

[30] C.-C. Wang and G.-N. Sung, Low-power multiplier design using a bypassing technique, J. Signal Process. Syst., 57, (2009), 331–338.

[31] J. Qian and J. Wang, A 4-bit array multiplier design by reversible logic, in Information Technology: Proceedings of the 2014 International Symposium on Information Technology (ISIT 2014), Dalian, China, 14-16 October 2014, (2015), 5.

[32] Xilinx and Inc, Xilinx DS255 Multiplier v11.2, Data Sheet, 2011. Accessed: Mar. 09, (2021). [Online]. Available: www.xilinx.com1.