Introduction

Embedded systems play a crucial role in many applications including Image processing, Digital Signal Processing, Cryptography as well as Deep Neural Networks. Global demand for embedded systems is increasing and power and computational inefficiency are the main obstacles in boosting the performance of embedded systems to meet customer needs. In this context, computationally intensive blocks within the embedded systems pose a great challenge in power consumption. One well-known solution to optimize power consumption and computational efficiency is to use a proper design of arithmetic unit. Multipliers and adders constitute main components of an arithmetic unit, therefore designing of efficient multipliers and adders improve computational efficiency an embedded system.

Add and shift algorithm is one of the most well-known methods to design a multiplier. On the other hand, the fundamental component to perform a multiplication is adder and designing an efficient adder results in a more efficient multiplier. In this project, we introduce an architecture of arithmetic unit to compute $Z = \frac{1}{4}[A * B] + 1$ based on "Add and shift" multiplier by using Ripple-Carry Adder (RCA) and Carry-Lookahead Adder (CLA). Moreover, we compare the delay, area and power consumption of the arithmetic unit by using different types of adders.

This report is organized as follows: In section 1, Project requirements to design the arithmetic unit of $Z = \frac{1}{4}[A * B] + 1$ is described. In section 2, different types of adders and *Shift and Add* multiplier are introduced. In the section 3, we design an efficient finite-state machine for this arithmetic unit based on *Add and Shift* algorithm and fundamental components to calculate $Z = \frac{1}{4}[A * B] + 1$ are proposed. Moreover, the simulation results of the arithmetic unit are presented at the end of this section.

Chapter 3

Design Methodology

In this section we proposed an arithmetic unit to calculate $Z = \frac{1}{4}[A * B] + 1$ where A and B are n-bit unsigned numbers. The operands are latched into registers RA and RB when $LOAD$ signal transits from high to low. A $2n$-bit number is shown in the output port when END_FLAG signal becomes high, we have the result of equation Z. The $CLEAR$ signal will clear all registers to 0.

3.1 Proposed Multiplication algorithm

We rewrite the equation for Z as follows:

$$Z = 2^{-2}W \tag{10}$$

$$W = (A * B) + 4 \tag{11}$$

where Z is calculated by shifting W, a 2n-bit binary number, to the right by two bits. As a result, the multiplication $A * B + 4$ is performed as follows:
$A = A_{n-1}A_{n-2}\ldots A_1A_0$ Multiplier
$B = B_{m-1}B_{m-2}\ldots B_1B_0$ Multiplicand
$4_10 = 000...0100_2$

				A_{n-1}	A_{n-2}	...	A_1	A_0
				B_{n-1}	B_{n-2}	...	B_1	B_0
				$B_0.A_{n-1}$	$B_0.A_{n-2}$	...	$B_0.A_1$	$B_0.A_0$
			$B_1.A_{n-1}$	$B_1.A_{n-2}$	...	$B_1.A_1$	$B_1.A_0$	0
		$B_2.A_{n-1}$	$B_2.A_{n-2}$	...	$B_2.A_1$	$B_2.A_0$	0	0
	$B_3.A_{n-1}$	$B_3.A_{n-2}$	...	$B_3.A_1$	$B_3.A_0$	1	0	0
	...	...	...	...	...	...	...	...
$B_{m-1}.B_{m-1}$	$B_{m-1}.B_{n-2}$	...	$B_{m-1}.A_1$	$B_{m-1}.A_0$				
P_{2n-1}	P_{2n-2}	...	...	...	...	P_2	P_1	P_0

As you can see in third stage, we need to perform addition, $B_2.A_{n-1}B_2.A_{n-2}...B_2.A_1B_2.A_0+1$ which requires to design a multiplier algorithm based on multiplier. This algorithm depends on LSB multiplier and $counter = 2$.

As demonstrated in Figure 13, this new *shift and add* multiplier has four main components including an n-bit adder, n-bit Mux 2 to 1, n-bit register of A, $2n$-bit register W and synchronous control unit.

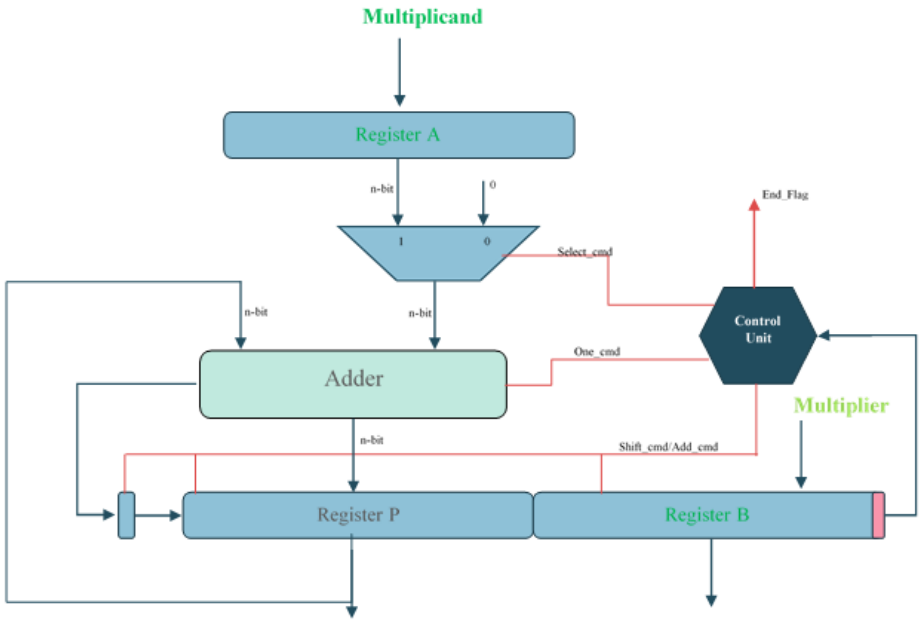

Figure 13: Proposed architecture of the *shift and add* multiplier

3.2 Designing Control Unit

1. When control unit receives a $start - cmd$ signal, the multiplicand is loaded into register A, while the multiplier is loaded to n least significant bits of register W.

2. LSB register W is tested, the counter is two:

   If the LSB is 1: a $add - cmd$ signal is issued, n most significant bits of register W are added to register A, then the result is put at n most significant bits of register W. Next, a shift signal is issued by control unit to shift the register W one bit to the right.

   If the LSB is 0: the control unit issues a shift signal into register W to shift the register W one bit to the right.

3. LSB register W is tested, the counter is one:

   If the LSB is 1: 1- $one - cmd$ and $add - cmd$ signals are issued, where $one - cmd$ signal is considered as C_{in} of the adder, in order to calculate sum of n most significant bits of register W, register A and one. Then , the result is put at n most significant bits of register W. Next, a shift signal is issued by the control unit to shift register W one bit to the right.

   f the LSB is 0: we need to increment n most significant bits of register W. Therefore, control unit issues three signals, $one - cmd$, $add - cmd$ and $select\_multiplicand\_0$.

   $select\_multiplicand\_0$ is considered as the select signal $mux2to1$ to insert 0 to adder.

   $one - cmd$ signal is considered as C_{in} adder, in order to increment n most significant bits of register W. The result is put at n most significant bits of register W. Then, a shift signal is issued by the control unit to shift the register W one bit to the right fig??.

4. When all multiplier bits have been tested (counter is $n - 1$) control unit shows END_FLAG, and value of the register W is product of $A * B + 4$.

Hence, W_1W_0 is a representation the fraction part of Z and $W_{2n-1}...W_3W_2$ is a representation of integer part of Z.

3.2.1 Finite State Machine

Therefore, we can design th state diagram based on the proposed algorithm as represented in Figure 14 .

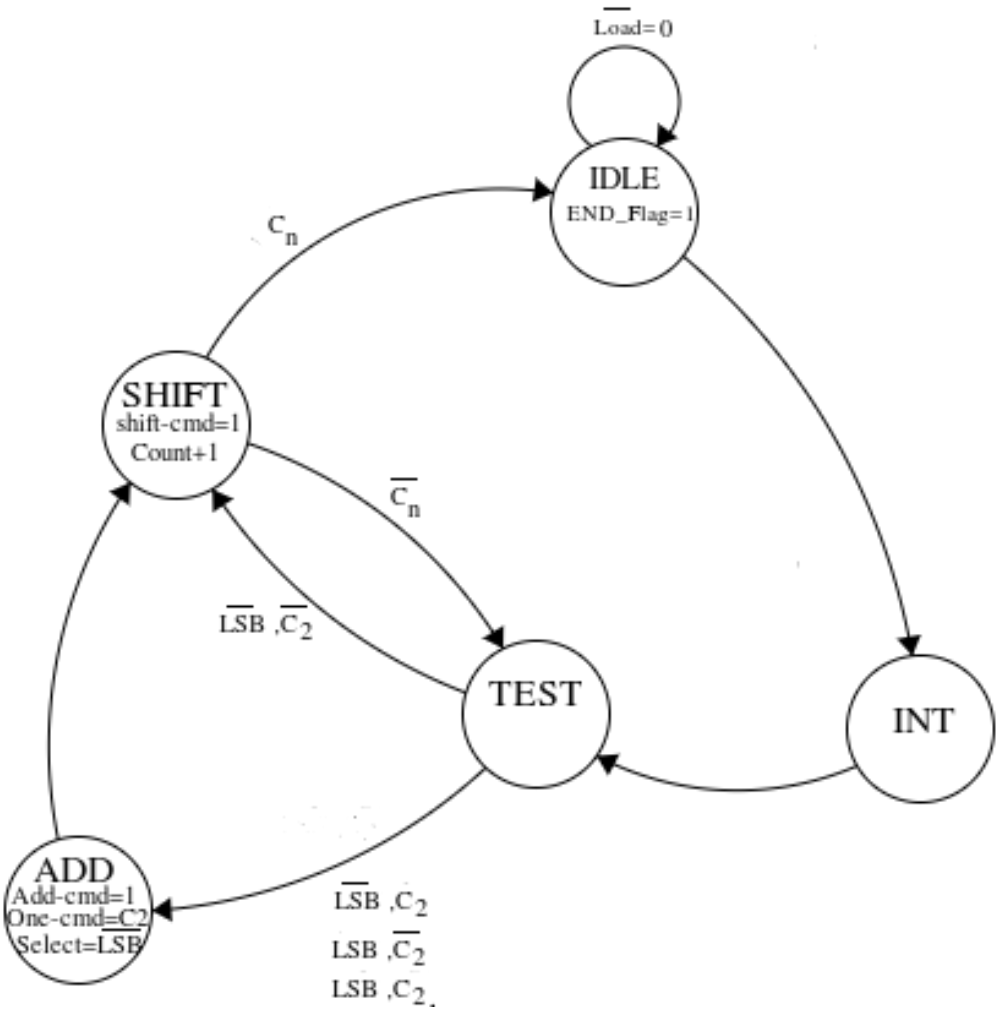

Figure 14: Proposed Controller FSM Diagram

As shown in Table 3, there are K-maps corresponding to each of the next states. The next state circuit of the proposed controller can be drawn using the next state

table and the derived output equations as shown in Figure 15and D= y^+ is the standard D-Flip Flop next state equation.

Table 3: Next state transition table

Present State	$y_2y_1y_0$	Next State $y_2^+y_1^+y_0^+$ C_nC_2LSB							
		000	001	011	010	100	101	111	110
IDLE	000	001	001	001	001	001	001	001	001
INIT	001	010	010	010	010	010	010	010	010
TEST	010	100	011	011	011	100	011	011	011
ADD	011	100	100	100	100	100	100	100	100
SHIFT	100	010	010	010	010	000	000	000	000

Table 4: Truth table y_2^+

Present State	$y_2y_1y_0$	Next State y_2^+ C_nC_2LSB							
		000	001	011	010	100	101	111	110
IDLE	000								
INIT	001								
TEST	010	1				1			
ADD	011	1	1	1	1	1	1	1	1
SHIFT	100								

$$y_2^+ = \overline{y_2}y_1y_0 + \overline{\mathrm{LSB}}.\overline{\mathrm{C_2}} \tag{12}$$

Table 5: Truth table y_1^+

Present State	$y_2y_1y_0$	Next State y_1^+ C_nC_2LSB							
		000	001	011	010	100	101	111	110
IDLE	000	0	0	0	0	0	0	0	0
INIT	001	1	1	1	1	1	1	1	1
TEST	010	0	1	1	1	0	1	1	1
ADD	011	0	0	0	0	0	0	0	0
SHIFT	100	1	1	1	1	0	0	0	0

$$y_1^+ = \overline{y_2}.\overline{y_1}.y_0 + \overline{y_2}.y_1.\overline{y_0}.LSB + \overline{\mathrm{LSB}}.C_2 + y_2.y_1.\overline{y_0}.\overline{\mathrm{C_n}} \tag{13}$$

Table 6: Truth table y_0^+

Present State	$y_2y_1y_0$	Next State y_0^+ C_nC_2LSB							
		000	001	011	010	100	101	111	110
IDLE	000	1	1	1	1	1	1	1	1
INIT	001								
TEST	010		1	1	1		1	1	1
ADD	011								
SHIFT	100								

$$y_0^+ = \overline{y_2}.\overline{y_1}.\overline{y_0} + y_1.LSB + \overline{\mathrm{LSB}}.C_2 \tag{14}$$

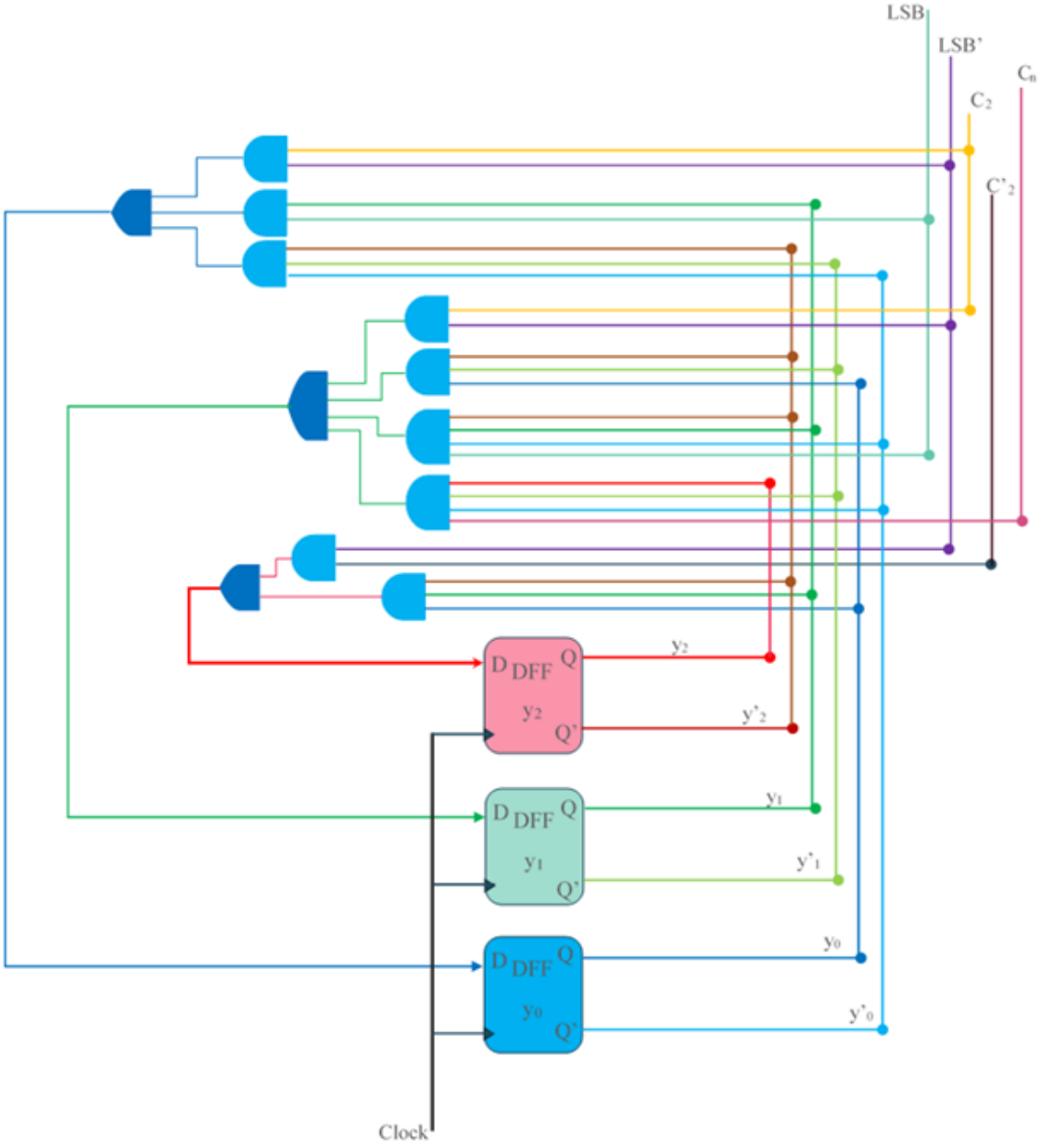

Figure 15: Next state circuit

Table 7: Output table

Present State	$y_2y_1y_0$	Output						
		$load-cmd$	$add-cmd$	$shift-cmd$	$count-cmd$	$one-cmd$	$select-cmd$	$End-Flag$
IDLE	000							1
INIT	001	1						
TEST	010							
ADD	011		1			C_2	$\overline{\text{LSB}}$	
SHIFT	100			1	1			

The control signals can be obtained immediately from the output table as shown in Table 7.

- $load - cmd = \overline{y_2}.\overline{y_1}.y_0$,
- $add - cmd = \overline{y_2}.y_1.y_0$,
- $shift - cmd = y_2.\overline{y_1}.\overline{y_0}$
- $one - cmd = \overline{y_2}.y_1.y_0.C_2$
- $select - cmd = \overline{y_2}.y_1.y_0.\overline{\text{LSB}}$
- $count - cmd = y_2.\overline{y1}.\overline{y_0}$

As shown in Figure 16, the circuit can be drawn D-flip flops by following the next state and the output equations derived.

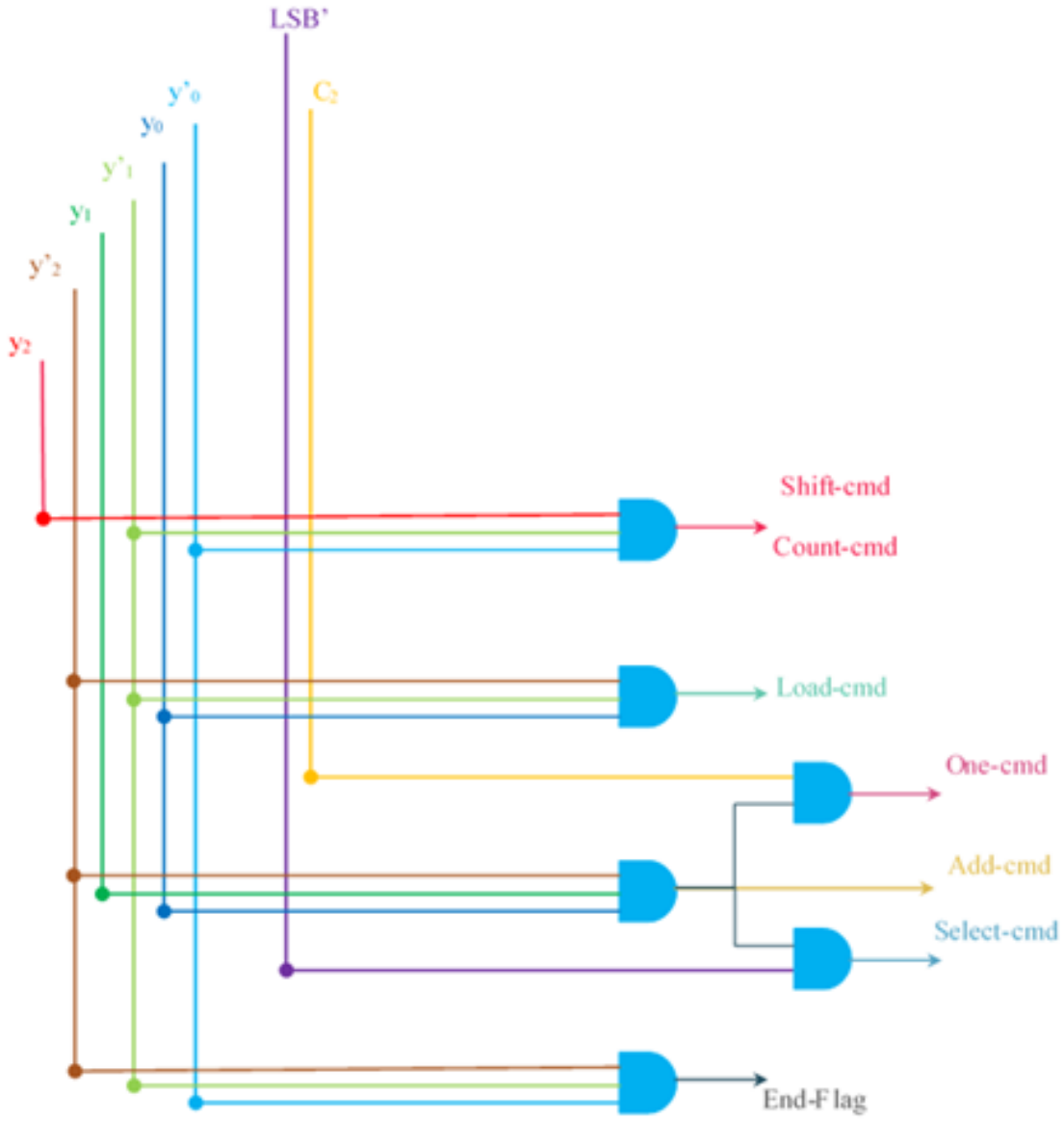

Figure 16: The circuit control signals of Controller Unit

3.2.2 Designing Counter

In order to design a general arithmetic unit, we need a general counter unit to be used in the control unit. Finite-state diagram of this counter is shown in Figure ??. Moreover, the output truth table and n-bit block diagram of the counter is demonstrated in Table 8 and Figure 18 , respectively.