Journal of Engineering Sciences, Assiut University, Vol. 34, No. 4, pp. 1283-1292, July 2006

HIGH-SPEED, AREA-EFFICIENT FPGA-BASED FLOATING-POINT ARITHMETIC MODULES

M. Taher
Electronic Engineer in Tibben Institute for Metallurgical Studies
Address: El_hadeed wa El_soulb st., Tibben, Cairo, Egypt
E. mail: marwa_t_1979@yahoo.com

M. Aboulwafa, A. Abdelwahab and E. M. Saad
Faculty of Engineering, Helwan University, Cairo, Egypt

(Received April 8, 2006. Accepted May 20, 2006)

ABSTRACT– *In this paper, single-precision floating-point IEEE-754 standard Adder/Subtractor and Multiplier modules with high speed and area efficient are presented. These modules are designed, simulated, synthesized, optimized, and implemented on an FPGA based system. A comparison between the results of the proposed design and a previously reported one is provided. The effect of normalization unit at the single-precision floating-point multiplier and adder/Subtractor modules on the area, and speed is explained.*

1. INTRODUCTION

The floating-point arithmetic modules were virtually impossible to be implemented on the older generations of FPGAs due to its limited density and speed.

Recently, the density and speed of FPGA are increased, so it becomes easy to implement floating-point arithmetic modules on it. With the appearance of high-level languages such as VHDL, rapid prototyping of floating point units has become feasible. Simulation and synthesis tools at a higher-level design aid the designer for a more controllable and maintainable product. Although low-level design specifications were alternately possible, the strategy used in the design that is presented here is to specify every aspect of the design in VHDL and rely on automated synthesis to generate the FPGA mapping.

The usage of floating point helps to manipulate the underflow and overflow problems often seen in fixed-point formats. This paper examines the implementations of floating-point arithmetic modules using single precision floating-point IEEE-754 standard format [1]. These modules have been synthesized on Xilinx Virtex-II XC2V6000bf957 FPGAs [2].

The general computing world has settled on floating-point formats, which conform to IEEE-754 standard [3]. These standards play a crucial role in ensuring numerical robustness and code compatibility among machines of vastly different architectures. However, the choice of floating-point format has such a dominant impact on FPGA implementation cost that the standards are often bent, giving the designer

freedom to choose a custom floating-point format in order to spend FPGA resources as efficiently as possible. For example, work has been done to automatically determine custom floating-point bit widths for each node of a computation [4]; others have demonstrated the suitability of very tiny floating-point formats with much less precision and range than IEEE single-precision [5].

2. FLOATING-POINT FORMAT REPRESENTATION

The floating-point format, which is used in this design, is the single-precision floating-point of IEEE-754 standard format [1] as shown in **Figure 1**.

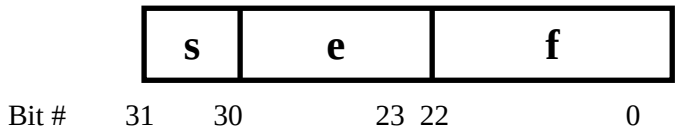


Figure 1: 32 Bit Floating Point Format.

The floating-point value (V) is computed by:

$$V = (-1)^{s} \times 2^{(e-bias)} \times (1.f) \quad (1)$$

As illustrated in **Figure 1**, the sign field, s, is bit number 31 and is used to specify the sign of the number, if s equals one the value will be negative, but if s equals zero the value will be positive. Bits 30 down to 23 are the exponent field. This 8-bit quantity is a signed number represented by using a bias of 127. Bits 22 down to 0 are used to store the binary representation of the fraction for the floating-point number. The leading one in the mantissa, 1.f, does not appear in the representation; therefore the leading one is implicit. For example, -3.625 (decimal) or -11.101 (binary) will be normalized as illustrated in equation (2) and the number is stored as in **Figure 2**.

$$V= (-1)^{1} 2^{(128-127)} (1.1101) \quad (2)$$

S	e								f																			
1	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C			0			6			8			0			0			0			0							

Figure 2: Representation of –3.625 in a single precision floating-point format.

Where s = 1, e = the times of right shifting for the number + the bias, the bias for single precision is 127, and the times of right shifting for this value " 11.101" is one, so e = 1+127 = 128(decimal) = 80 (hex), and f=680000 (hex). Therefore, -3.625 is stored as: C0680000 (hex) as illustrated in **Figure 2**.

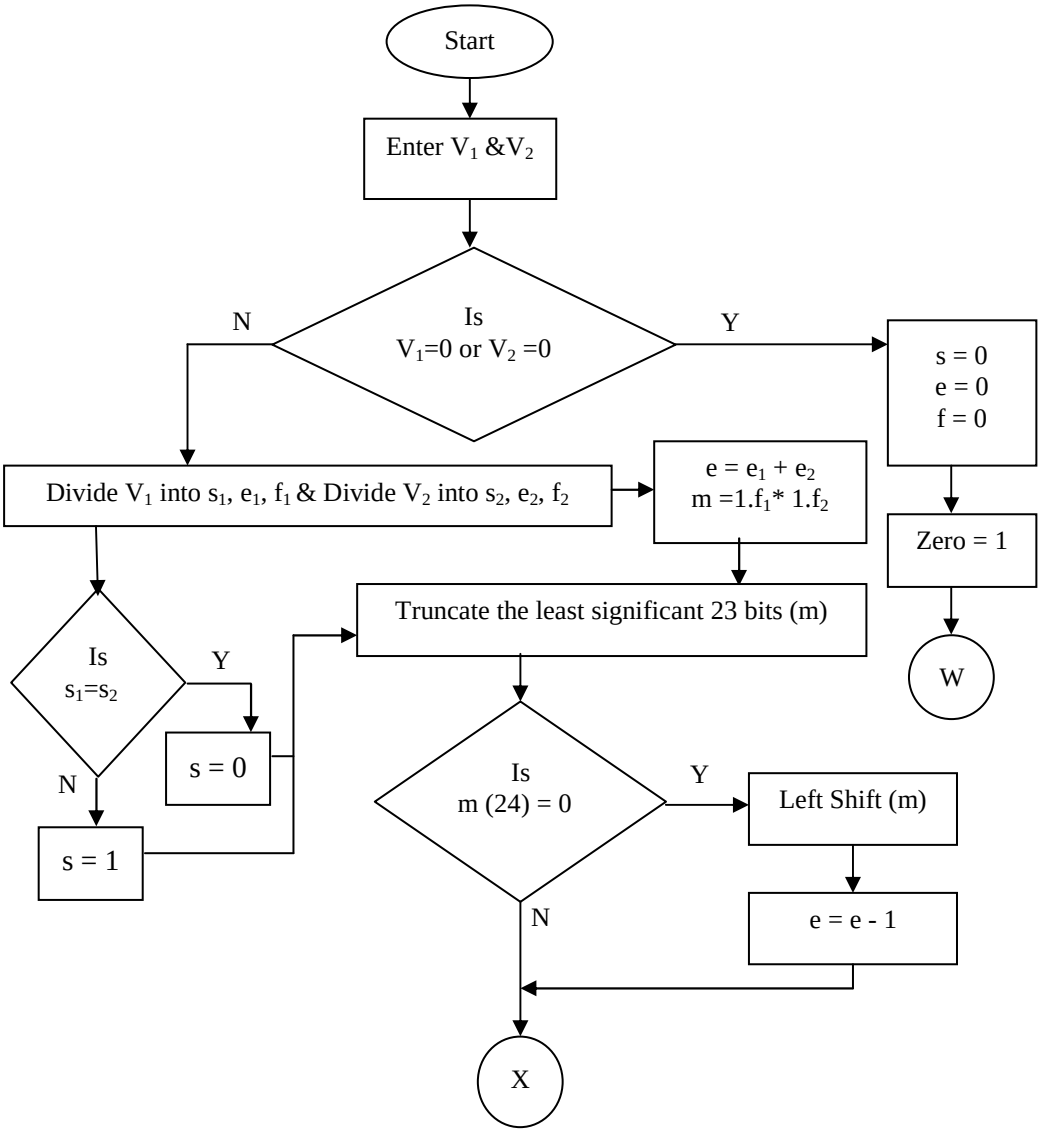
3. FLOATING POINT MULTIPLIER

Floating-point multiplication is similar to integer multiplication, because floating-point numbers are stored in sign-magnitude form; the multiplier needs only to deal with unsigned integer numbers and normalization. The optimized design of the single-precision floating-point multiplier has a latency of one clock cycle. The

presented design has some ideas of that of Shirazi's 18-bits floating-point format multiplier [8]. The bottleneck of this design is the normalization unit. The optimization of the normalization unit is presented in this paper which allows the multiplier to run at slightly faster clock speed. It also helps in reducing the usage area.

3.1. Algorithm

The flowchart for a single-precision semi-parallel floating-point multiplier is shown in **Figure 3**.



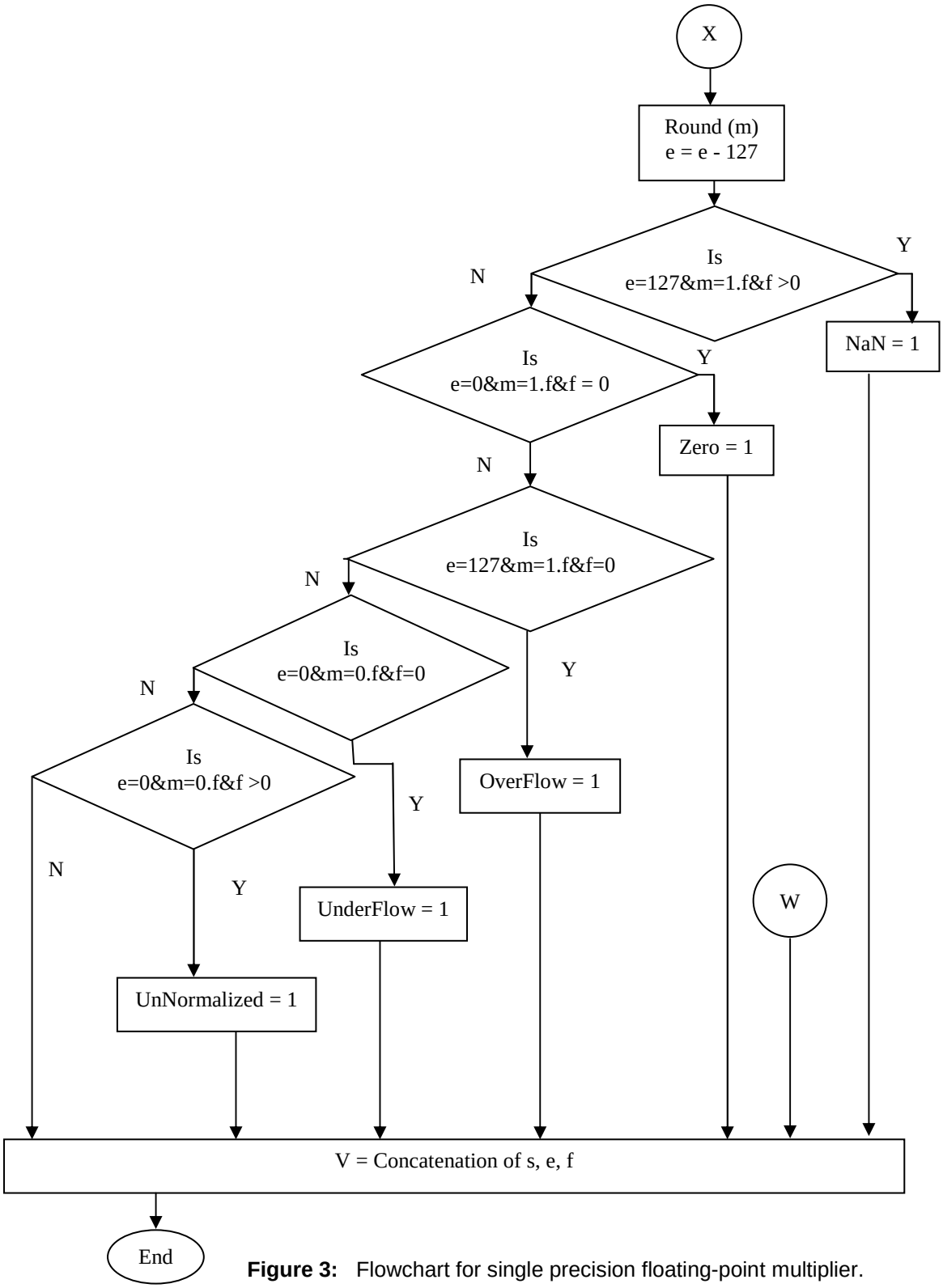


Figure 3: Flowchart for single precision floating-point multiplier.

Where:

V: value represented in a single precision format
s: sign bit.
e: exponent.
f: fraction.
m: mantissa.
NaN: Not a Number.

3.2. Results

The proposed single-precision semi-parallel floating-point multiplier module is implemented using VHDL language. It is mapped on the same FPGA chip that is used in [7] (Xilinx Virtex-II XC2V6000bf957). The synthesis results of the proposed configuration are compared with previous published results in [7] as shown in **Table 1**.

Table 1: The comparison of the synthesis results.

The results of the proposed configuration		The results in [7]	
Function generator (F.G.)	Speed	Function generator (F.G.)	Speed
202	11.24 ns = 89 MHZ	452	49 ns = 20.4 MHZ

By comparing the results of the proposed technique that are given in **Table 1** with those results in [7] also shown in the same table, it can be seen that the used area in our design is reduced by 55% while the speed is increased by 336.3%.

4. FLOATING POINT ADDER/SUBTRACTOR

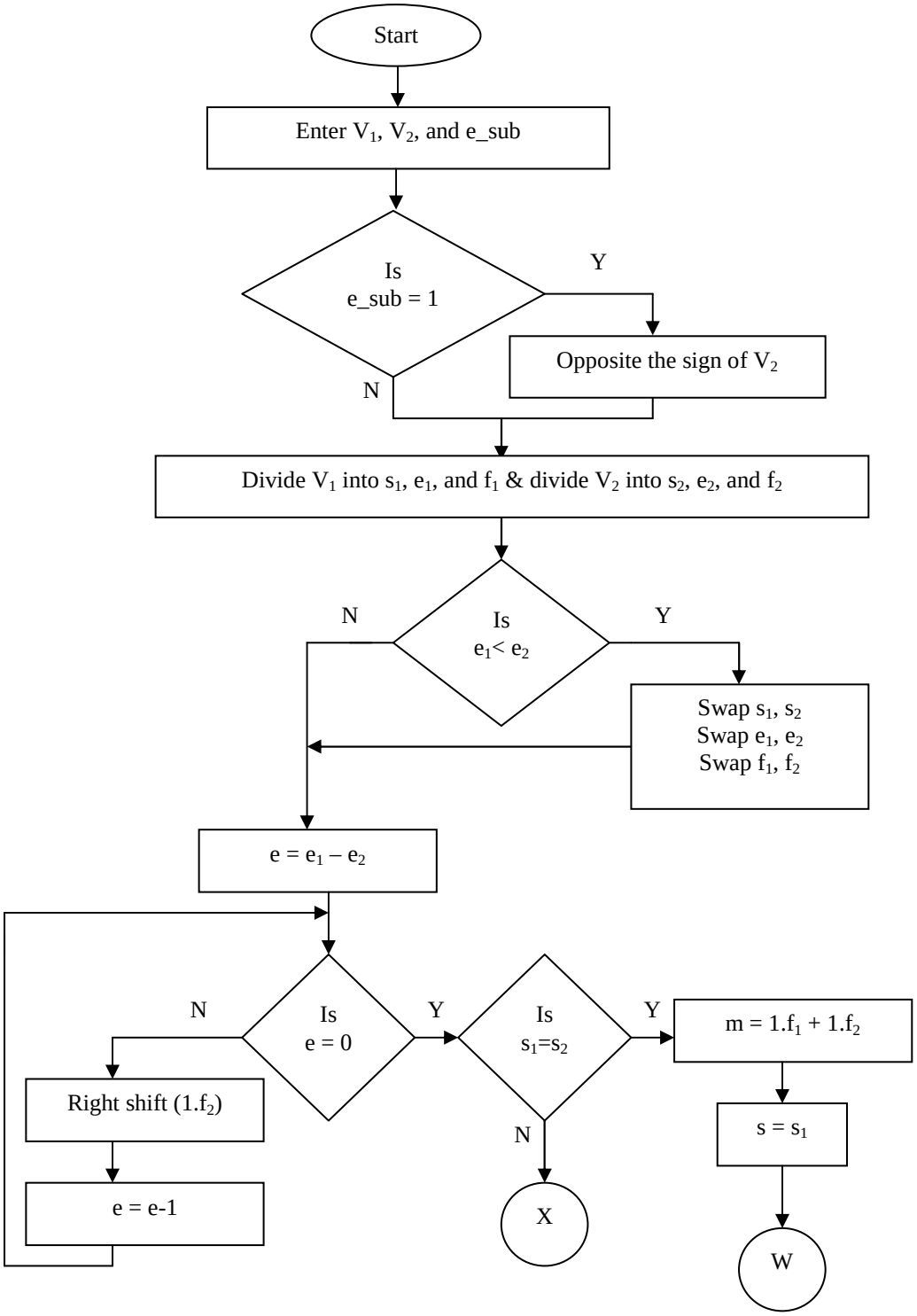
An optimized design of the 32-bit floating-point Adder/Subtractor has a latency of one clock cycle is proposed. The presented design has some idea as that of Shirazi's 18-bit floating-point format Adder/Subtraction in [8]. But, the configuration of the normalization module allows the Adder/Subtraction to run at a slightly faster clock speed and also helps to reduce the used area. The bottleneck of this design was the normalization unit.

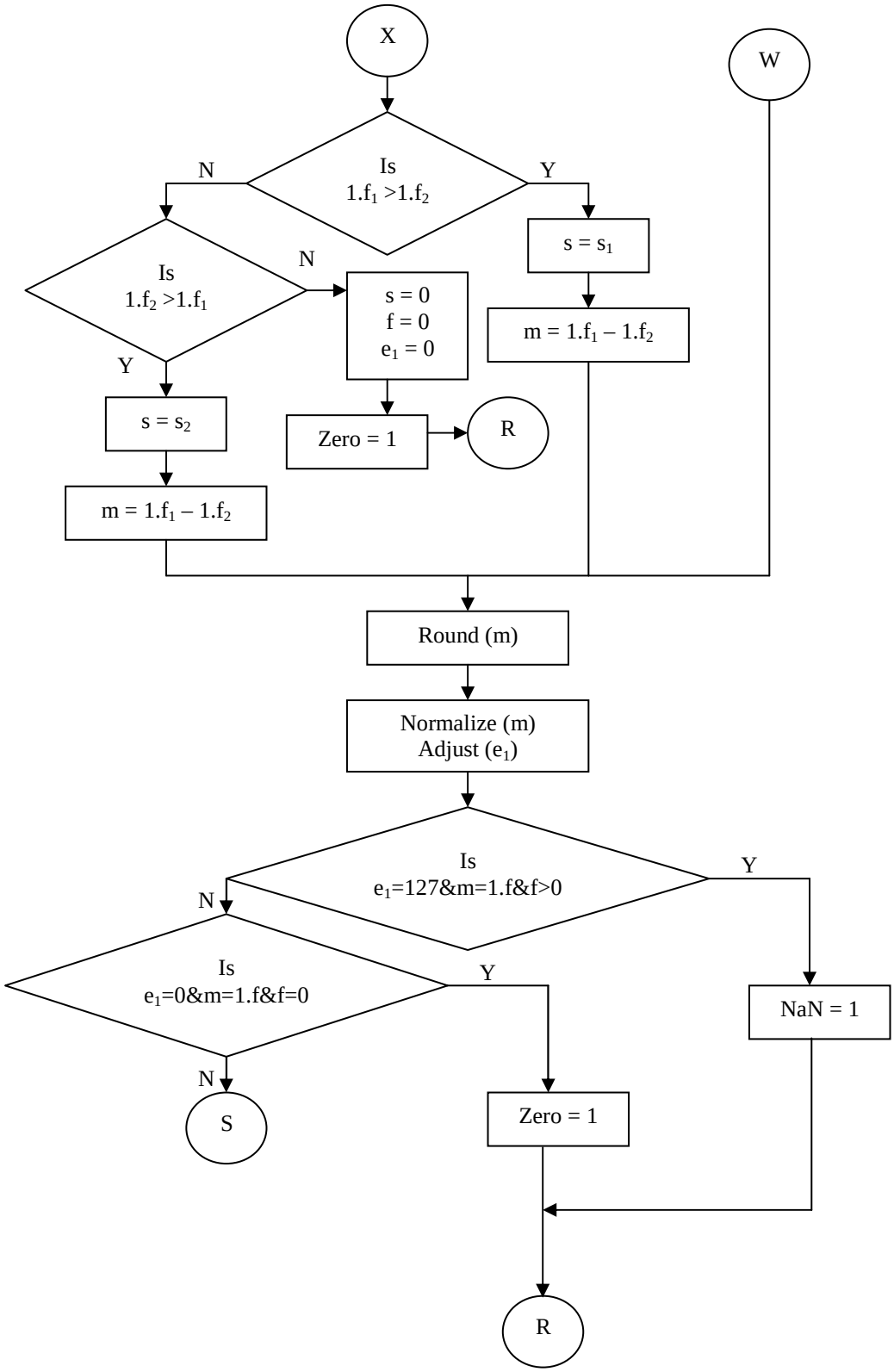
4.1. Algorithm

The flowchart of a single-precision cascaded floating-point Adder/Subtraction is shown in **Figure 4**.

Where:

V: value represented in a single precision format
s: sign bit.
e: exponent.
f: fraction.
m: mantissa.
e_sub: selection line to perform the addition or subtraction processes.
NaN : Not a Number.


Start
Enter V_1, V_2, and e_sub
Is e_sub = 1
Y
Opposite the sign of V_2
N
Divide V_1 into s_1, e_1, and f_1 & divide V_2 into s_2, e_2, and f_2
N
Is $e_1 < e_2$
Y
Swap s_1, s_2
Swap e_1, e_2
Swap f_1, f_2
$e = e_1 - e_2$
N
Is e = 0
Y
Is $s_1 = s_2$
Y
$m = 1.f_1 + 1.f_2$
Right shift ($1.f_2$)
e = e-1
N
X
$s = s_1$
W


X
W
Is
1.f1 >1.f2
N
Y
s = s1
m = 1.f1 – 1.f2
Is
1.f2 >1.f1
N
Y
s = 0
f = 0
e1 = 0
Zero = 1
R
s = s2
m = 1.f1 – 1.f2
Round (m)
Normalize (m)
Adjust (e1)
Is
e1=127&m=1.f&f>0
N
Y
Is
e1=0&m=1.f&f=0
Y
N
NaN = 1
S
Zero = 1
R

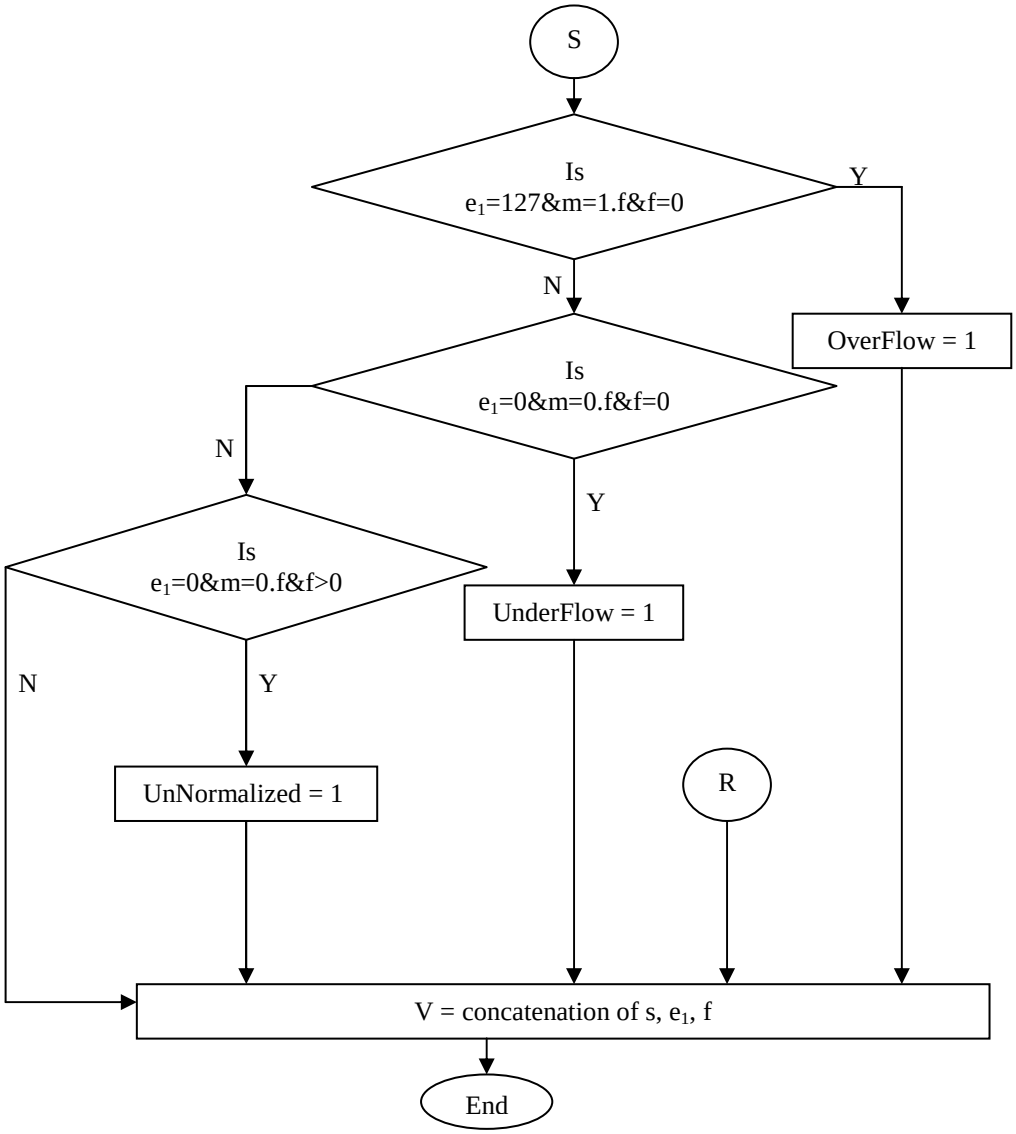


Figure 4: Flowchart for single precision floating-point adder/subtraction.

4.1.1. Normalization

Every four bits of the mantissa will be the inputs to an OR-gate. Then, “which of the six outputs of the OR-gates that is the leading one” can be detected rapidly, and the leading-one detection logic decides which of the six nibbles of the mantissa value contains the leading-one.

After that, the 5-bit shift value using the data word from the leading-one detection logic that determines which of the six nibbles in the resulting mantissa the one resides in. The data word can be used to determine what the upper three bits of the

shift value are to be while the lower two bits are determined by the bit values in the nibble containing the leading one. The combinational logic to determine the lower two bits can be constructed from two, 4-variable logic equations:

$$S_0 = (\text{not } n_3) \text{ and } (n_2 \text{ or } ((\text{not } n_1) \text{ and } n_0)) \quad (3)$$

$$S_1 = (\text{not } n_3) \text{ and } (\text{not } n_2) \text{ and } (n_1 \text{ or } n_0) \quad (4)$$

Where: s_0 and s_1 are bits 0 and 1 of the constructed shift value, respectively. The n_3, n_2, n_1, and n_0 values represent bits 3 to 0, respectively, of the nibble containing the leading-one.

4.2. Results

The proposed single-precision cascaded floating-point adder/subtractor module is implemented using VHDL language. It is mapped on the same FPGA chip that is used in [7] (Xilinx Virtex-II XC2V6000bf957). The synthesis results of the proposed configuration are compared with previous published results in [7] as shown in **Table 2**.

Table 2: The comparison of the synthesis results.

The results of the proposed configuration		The results in [7]	
Function generator (F.G.)	Speed	Function generator (F.G.)	Speed
490	30.67 ns = 32.6 MHZ	521	51.5 ns = 19.4 MHZ

By comparing the results of the proposed technique given in **Table 2** with corresponding results in [7] that are shown in the same table, it can be seen that the used area in our design is reduced by 6% while the speed is increased by 68%.

5. CONCLUSION

A design of the single-precision floating-point arithmetic modules with an optimized area and speed is presented. The effect of normalization on the area and speed has been examined experimentally. The design has been mapped on Xilinx vertex-II XC2V6000bf957. Comparisons of results between the proposed systems and previously published results have been demonstrated. The presented single-precision floating-point multiplier, adder, and subtractor modules run at slightly faster clock speed with used area less than that used previously.

REFERENCES

[1] IEEE Task P754, “A Proposed Standard for Binary Floating-Point Arithmetic,” IEEE Computer, Vol.14, No.12, pp.51-62, Mar.1981.
[2] Xilinx, Inc., the Programmable Logic Data Book, San Jose, California, 1993.
[3] IEEE Standards Board. IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE STD 754-1985 edition, 1985.

[4] A. A. Gaffar, O.Mencer, W.Luk, P.Y.Cheung, and N.Shirazi. "Floating Point Bit width Analysis via Automatic Differentiation". Proceedings of the International Conference on Field Programmable Technology, 2002.
[5] J. Dido et al. "A Flexible Floating-Point Format for Optimizing Data-Paths and Operators in FPGA Based DSPs". ACM/SIGDA Tenth ACM International Symposium on Field-Programmable Gate Arrays (FPGA'02), 2002.
[6] GH. A . At y, A. Hussein, I. Ashour, and M. Mones, "High-speed area-efficient FPGA-based floating-point multiplier", Proceedings of ICM 2003 Conference, Dec.2003, Cairo, EGYPT, pp.274-277.
[7] Bryan Cantanzaro, Brent Nelson, "Higher Radix Floating-Point Representations for FPGA-Based Arithmetic" in Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machiens, 2005.
[8] N. Shirazi, A. Walters, and P. Athanas, "Quantitative analysis of floating point arithmetic on FPGA based custom computing machines", in Proceedings of IEEE Symposium on FPGAs for custom Computing machines, pp.155-162, 1995.

تصميم وتنفيذ الوحدات الحسابية بصيغة النقطة العائمة مستخدماً مصفوفة البوبات القابلة للبرمجة "FPGA"

هذه المقالة تناولت تصميم وحدة الضرب و وحدة الجمع والطرح بصيغة النقطه العائمة ذات الدقة الاحادية بمعيار "IEEE-754" . تم تحسين السرعة والمساحة المستخدمة للوحدات السابقه كما تم تنفيذها باستخدام مصفوفة البوبات القابلة للبرمجة "FPGA" ومقارنة النتائج من حيث السرعة والمساحة بالنتائج التى تم نشرها سابقا. كما تم توضيح تأثير وحدة التطبيع على المساحة والسرعة لتلك الوحدات.