International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 18
December2016

A Power-Efficient Floating-point Co-processor design

T. Sowmya
TURAKASOWMYA@GMAIL.COM
Department Of Ece Malinani Lakshmiah Engineering College
G.Manga Rao
Assistant Professor Department Of Ece Malinani Lakshmiah Engineering College
MANGARAO.GUNJI@GMAIL.COM

ABSTRACT

In recent years computer applications have increased in their computational complexity. The processor designers to pay particular attention to implementation of the floating-point unit. And also due to drastically growing interests in low power and area efficient embedded processor, designers must establish the proper power and area strategies in their architecture while design new embedded processor core. This paper proposed efficient architecture to design a SPARC compatible floating-point co-processor, which is part of a SPARC compatible embedded processor, which implement the SPARC V8 floating-point instruction sat except for square root. In the proposed architecture, decoder stage of the integer unit pipeline generates the clock gating signals so that the unused floating-point co-processor execution pipeline can be clock-gated, which leads to lower the power dissipation and of floating-point co-processor.

INTRODUCTION

This chapter discusses about the introduction, types of co-processors, basic formats of floating-point numbers, normalization, rounding methods with floating-point numbers

Co-processor

A Co-processor is a computer processor used to supplement the functions of the primary processor (the CPU). Operations performed by the co-processor may be floating- point arithmetic, graphics, signal processing, string processing, or encryption. By offloading processor-intensive tasks from the main processor, co-processors can accelerate system performance. Co-processors allow a line of computers to be customized, so that customers who do not need the extra performance need not pay for it. The 8087 was tightly integrated with the 8086/8088 and responded to floating-point machine code operation codes inserted in the 8088 instruction stream. An 8088 processor without an 8087 could not interpret these instructions, requiring separate versions of programs for FPU and non-FPU systems, or at least a test at run time to detect the FPU and select appropriate mathematical library functions. Fig.1.1 shows Intel 80386 CPU with 80387 math co-processor

Normalization

Most often M is a normalized fraction in the sign-magnitude form. The true magnitude of it is $|M| = (0.m_{-1}m_{-2}\dots m_{-k})$. That is, $1 \le |M| \le 2$. By normalizing, the MSD of the mantissa equals non-zero, that is, $m_{-1} \neq 0$. If it is zero, the floating-point number is un-normalized. F = (-0.0025, +3) = (-0.25, +1) indicates how the normalization procedure can be conducted. Note that a normalized mantissa has its absolute value limited by $\underline{1} \leq |M| < 1$,

where $\underline{1}$ is the weight carried by the MSD in the mantissa. In other words, if $\underline{1} \leq |M| < 1$ the floating-point number is normalized. For the binary case with k bits in the mantissa, $\underline{1} \leq |M| \leq 1 \_ 2^{-k}$. The reason for normalization is to fully utilize the available bits which are limited in the computer system and are hence precious. Allowing too many leading 0s in the fraction may cause unnecessary truncation of the lower order bits in the mantissa. The bit positions occupied by those leading 0s are wasted, and the accuracy of the number representation is degraded.

PIPELINING

This chapter describes about pipeline processing and general considerations in pipeline process.

Pipelining is a technique of decomposing a sequential process into sub-operations, with each sub-process being executed in a special dedicated segment that operates concurrently with all other segments. A pipeline can be visualized as a collection of processing segments through which binary information flows. Each segment performs partial processing dictated by the way the task is partitioned. The result obtained from the computation in each segment is transferred to next segment in the pipeline. The final result is obtained after the data have passed through all segments. The name ―pipeline‖ implies a flow of information analogous to an industrial assembly line. It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time. The overlapping of computation is made possible by associating a register with each segment in the pipeline. The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

General considerations

Any operation that can be decomposed into a sequence of sub-operations of about the same complexity can be implemented by a pipeline processor. The technique is efficient for those applications that need to repeat the same task many times with different sets of data. The general structure of a four-segment pipeline is illustrated in Fig.2.2. The operands pass through all four segments in a fixed sequence. Each segment consists of a combinational circuit Si that performs a sub-operation over the data stream flowing through the pipe. The segments are separated by registers Ri that hold the intermediate results between the stages. Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously. It is defined that a task as the total operation performed going through all the segments in the pipeline.

BOOTH ALGORITHM

This chapter discusses about booth algorithm and multiplication methods.

Multiplication of Positive Numbers

The usual algorithm for multiplying integers by hand is illustrated in Fig.3.1a for the binary system. This algorithm applies to unsigned numbers and to positive signed numbers. The product of two n-digit numbers can be accommodated in 2n digits, so the product of the two 4-bit numbers in this example fits into 8-bits, as shown. In the binary system, multiplication of the multiplicand by one bit of the multiplier is easy. If the multiplier bit is 1, the multiplicand is entered in the appropriate position to be added to the partial product. If the multiplier bit is 0, then 0s are entered, as in the third row of the example.

```
    1 1 0 1 ( 13 ) Multiplicand M
  x1 0 1 1 ( 11 ) Multiplier Q
  ----------------------------
      1 1 0 1
    1 1 0 1
  0 0 0 0
1 1 0 1
------------
1 0 0 0 1 1 1 1
```

Implementation of Booth Algorithm

The booth algorithm generates a 2n-bit product and treats both positive and negative

2's-complement n-bit operands uniformly. To understand the essence of this algorithm, consider a multiplication operation in which the multiplier is positive and has a single block of 1s, for example, 0011110. To derive the product, add four appropriately shifted versions of the multiplicand, as in the standard procedure. However, it then reduces the number of required operations by regarding this multiplier as the difference between two numbers:

```
 0 1 0 0 0 0 0 (32)
-0 0 0 0 0 1 0 ( 2 )
--------------------
 0 0 1 1 1 1 0 ( 30 )
```

This suggests that the product can be generated by adding 2^5 times the multiplicand to the 2's-complement of 2^1 times the multiplicand. For convenience, describe the sequence of required operations by recoding the preceding multiplier as 0 +1000 -10.

Fast Multiplication

This describe two techniques for speeding up the multiplication operation. The first technique quarantines that the maximum number of summands (versions of the multiplicand) that must be added is n/2 for n-bit operands. The second technique reduces the time needed to add the summands.

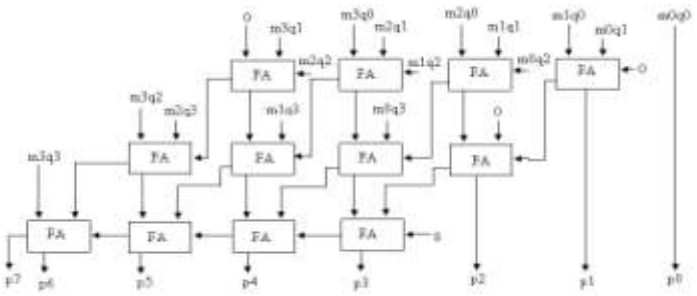
WALLACE TREE

This chapter discusses about Wallace tree and carry-save addition methods.

Carry-Save Addition Of Summands

Multiplication requires the addition of several summands. A technique called carry-save addition (CSA) speeds up the addition process. Consider the array for 4x4 multiplication shown in Fig.4.1a this structure is the general array shown in Fig.3.1, with the first row consisting of just the AND gates that implement the bit products m3q0,m2q0,m1q0, and m0q0.

a. Ripple-Carry Array



b. Carry-Save Arra

Implementation of Wallace Tree

A Wallace tree is a combinatorial circuit used to multiply two numbers. Although it requires more hardware than shift-add multipliers, it produces a product in far less time. Instead of performing additions using standard parallel adders, Wallace trees use carry-save adders and only one parallel adder.

A carry-save adder can add three values simultaneously, instead of just two. However, it does not output a single result. Instead, it outputs both a sum and a set of carry bits. To illustrate this, consider the carry-save adder

CARRY-PROPAGATE ADDER

This chapter discusses about Carry-propagate adder.
With the carry input, full adders can be cascaded to produce

an n-bit adder by connecting output C of an adder to Cin of the next adder. This configuration is called a ripple adder.A 4-bit ripple adder is shown in Fig.5.1, along with its schematic symbol. Note that the least significant adder is a full adder with an externally generated carry input. Most CPUs require this carry input for some operations. If it was not needed, this full adder either could have its carry input set to 0, it could be replaced by a half adder.

SRT DIVISION

This chapter discusses about SRT division methods.

SRT is named after its inventors Sweeney, Robertson and Tocher. Independently and

at about the same time, D.W. Sweeney of IBM, J.E Robertson of the university of Illinois and K.D Tocher of imperial college, London, discovered a new method of binary division. In
partial dividends are normalized fractions. Recall that the partial dividend is r times of the partial remainder in general. Unlike in non-restoring division algorithm q_{j+1} Є {-1, 1}, q_{j+1} Є

{-1, 0, 1} now. Here the divisor is shifted, or added to or subtracted from the partial dividend since {-1, 0, 1} is to be selected. That is,

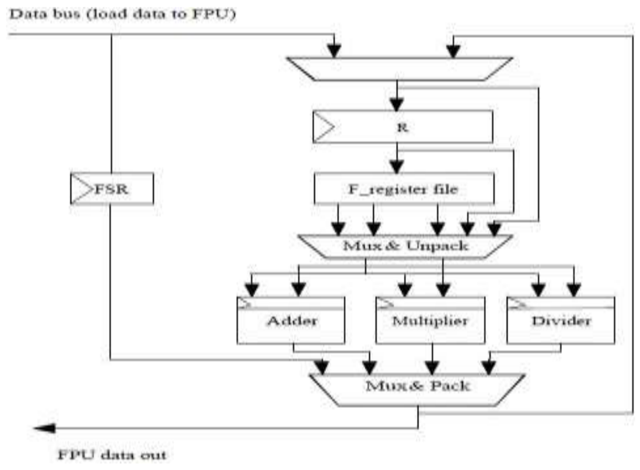
Modified SRT Division

The design of fast dividers is an important issue in high-speed computing because division accounts for a significant fraction of the total arithmetic operation. Most implementations for the division are based on the SRT algorithm that uses a recurrence producing one quotient digit for each step. The speed of such SRT-based dividers is mainly determined by the complexity of the quotient-digit selection. Fig.6.1 illustrates architecture of a radix-4 SRT divider which employs a quotient-digit selection table (QST). The use of QST significantly reduces the complexity of quotient-digit selection. However, the table size increases drastically with high radices. The table size can be reduced significantly by estimating the quotient digit instead of finding the exact one. The estimated quotient digit is calibrated in parallel with updating the new partial remainder. Since the two-step process does not affect the division speed, the approach has fast speed performance due to significant reduction in table size.

ARCHITECTURE OF FLOATING-POINT UNIT

This chapter discusses abouArchitecture of floating-point unit operation.

The power dissipation of pipeline register is an important part of the whole processor. In addition, the low power strategy design in this project is a fine-grained clock gating technique, which can reduce the power dissipation of pipeline register by gating the register's input clock when the corresponding execution pipeline is unused. Because of these facts, in this project emphasize the execution pipeline organization and eliminate the irrespective design details. The data path shown in Fig.7.1, implements an always clock on block. Considering the performance and design complexity, the floating-point co-processor includes three independent execution pipelines: floating-point adder which completes addition, subtract, compare and conversion instructions, floating-point multiplier which completes multiply instructions and floating-point divider which executes divide instructions.



Floating-Point Division

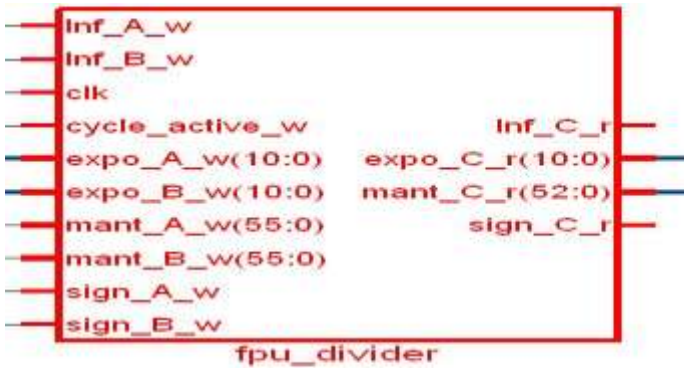
Floating-point division requires that the exponents be subtracted and the mantissas divided. The mantissa division is done as in fixed-point except that the dividend has a single-precision mantissa that is placed in the mant_C_r. remember that the mantissa dividend is a fraction and not an integer. For integer representation, a single-precision dividend must be placed in mant_A_w.

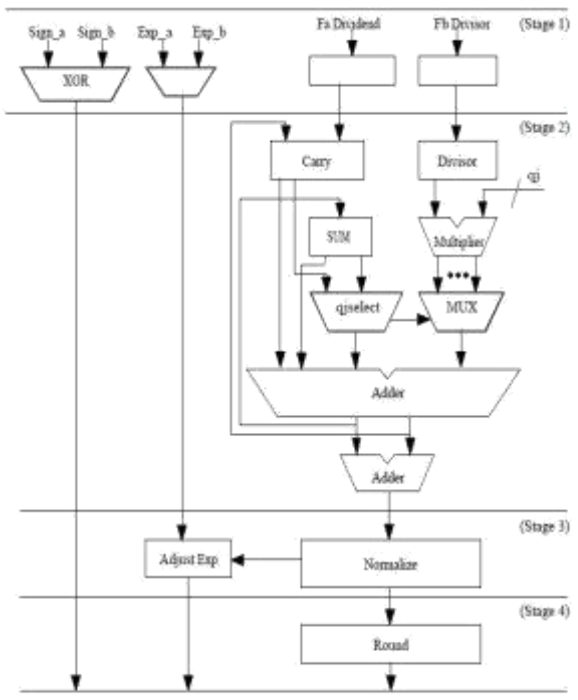
The check for divide-overflow is the same as in fixed-point representation. However, with floating-point numbers the divide-overflow imposes no problems. If the dividend is greater than or equal to the divisor, the dividend fraction is shifted to the right and its exponent incremented by 1. For normalized operands this is a sufficient operation to ensure that no mantissa divide overflow will occur. The operation above is referred to as a dividend alignment.

The division algorithm can be subdivided into four parts or stages.

1. Initialize the values and evaluate the sign and exponent.
2. SRT algorithm is implemented.
3. Adjust the exponents and mantissa.
4. Round the result.

The RTL schematic of divider block and pipeline flow for floating-point division





RESULTS

This chapter gives the simulation results occurred with different combination of inputs. Mant_c_r is used as mantissa output, expo_c_r is used as exponent output and sign_C_r is used as sign output. Sign, exponent and mantissa inputs are given through sign_A_w, sign_B_w, expo_A_w, expo_B_w, mant_A_w, mant_B_w. when a valid inputs are given the outputs are generated or else the output will be not-a-number, infinity and so on signals will appear.

Inputs:

sign_A_w=0, sign_B_w= 0, expo_A_w=10000000110, expo_B_w=10000000110,

mant_A_w=1111001100000000000000000000000000000000 00000000000000 0=243,

mant_B_w=1110101000000000000000000000000000000000

00000000000000 0=234.

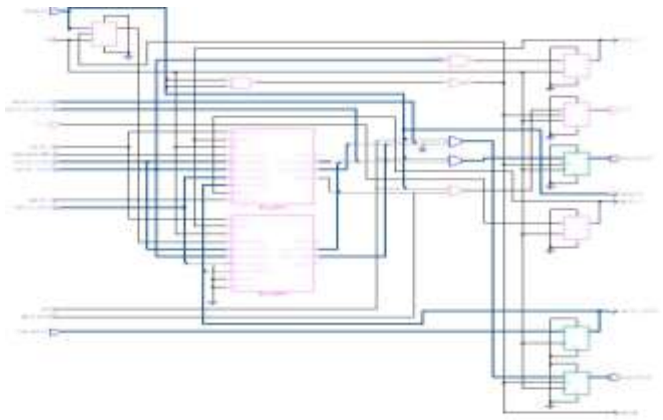
Outputs:

sign_C_r=0, expo_C_r=10000000111,

mant_C_r=1110101000000000000000000000000000000 0000000000000

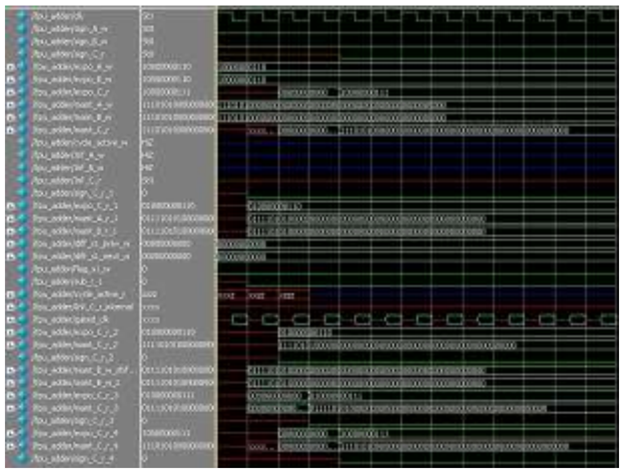
RTL Schematic for FPU (Floating-Point Unit) – Adder, Divider

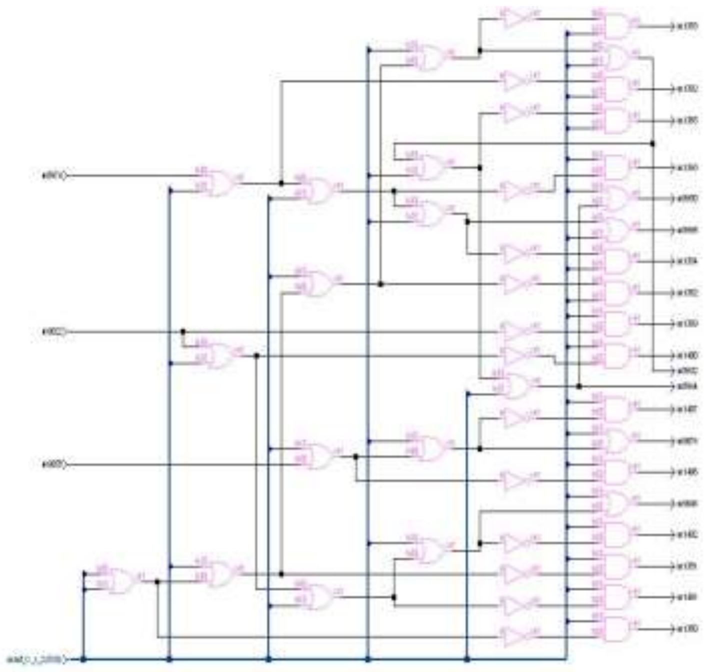
RTL schematic for FPU (Floating-Point Unit) – Adder, Divider



RTL Schematic for Divider

RTL schematic for divider, a block in the schematic.





CONCLUSION

This chapter gives the final conclusion of the overall project.

Although some commercial processors may use some form of clock gating, most of them are focused on the circuit level. Aimed at our space application, this develops a SPARC compatible floating-point co-processor and corresponding micro-architecture clock gating method with tiny cost, based on the intuition of DCG (Deterministic Clock Gating). The floating-point co-processor contains roughly 23150 gates. As per VIRTEX IV family and 4VSX35FF668-12 device CLB slices used are 10013, function generators used are 20026, latches used are 5748. Individual blocks in the floating-point co-processor like adder uses 1215 gates, multiplier uses 11771, divider uses 10064 gates. And at 330 MHz frequency 675 mW.

REFERENCES

[1] D.F.M keating, *Low Power Methodology Manual for System-on-Chip Design*. New York, USA: Springer, 2007.

[2] *micro*SPARC-*IIep User Manual*. CA, USA: SUN Microsystems, 1999.

[3] J. Gaisler, The LEON-2 Processor User's Manual Version 1.0.13. Goteberg, Sweden: Gaisler Research, 2003.

[4] *The SPARC Architecture Manual Version 8*. Menlo Park, California, USA: SPARC International, Inc., 1991.

[5] *IEEE Standard for Binary Floating-Point Arithmetic*. New York, USA: IEEE, 1985.

[6] M. Lu, *Arithmetic and Logic in Computer Systems*. New Jersey, USA: John Wiley Sons, 2004.

[7] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York, USA: Oxford, 2000.

[8] W. J. P. Silvia M. Mueller, *Computer Architecture: Complexity and Correctness*. New York, USA: Springer, 2000.

[9] J. E. Stine, *Digital Computer Arithmetic Data Path Design Using Verilog HDL*. New York, USA: Kluwer, 2003.

[10] S. M. R. I. Bahar, ―Power and energy reduction via pipeline balancing,‖ in Proc. *28th International Symposium on Computer Architecture (ISCA'01)*, July 1999, pp. 218-229.

[11] R. Gonzalez and M. Horowitz, ―Energy dissipation in general purpose processors,‖ *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp.1277-1284, September 19

BOOK'S

[1] *Computer system organization* by Morris Mano.

[2] *Computer organization* by Carl Hamacher.

[3] *Computer systems organization & architecture* by John D. Carpinelli.