Scalable Pipeline Insertion in Floating Point Units for FPGA Synthesis

Irvin Ortiz Flores
Advisor: Manuel Jiménez

Electrical and Computer Engineering Department
University of Puerto Rico, Mayagüez Campus
Mayagüez, Puerto Rico 00681-5000
irvin.ortiz@ece.uprm.edu

Abstract

Most modern processors rely on pipeline techniques to achieve high throughput. This work reports the development of scalable, floating-point (FP) arithmetic operators with variable number of pipeline stages. A new algorithm for pipeline insertion was developed and used for FP Multiplication and FP Addition. The use of this algorithm enables operating frequencies up to 175MHz when implemented on a Xilinx Virtex II FPGA. Future work includes the automation of the process and the inclusion of the algorithm into FP square root and division units.

1.Introduction

Pipeline techniques allow operating a circuit at high clock rates by dividing a large task into smaller non-overlapping sub-tasks. This allows for parallel processing without the need of extra computing units. Final results are obtained after completing all stages. Careful selection of the latch insertion points is an important factor for obtaining optimal throughput.

In special purpose computing, dedicated adders are required to have high throughput while latency constraints are not severe. In such cases, pipelined architectures are used widely. Traditional pipelined adders for parallel addition of two operands are based on carry save addition [3] or ripple adders.

Pipelined multipliers are desirable for high-performance arithmetic applications such as digital signal processing. The most common type of multiplier used for pipelined applications is the array multiplier. This is due to its regular and modular design. Asato et al. developed a compiler to produce customized, pipelined array multipliers optimized to operate at a given clock rate [2]. Their method for pipeline insertion consisted in the introduction of rows of latches through the multiplier structure, which divides the array into rows of cells that operate independently from each other. The results of this approach for a 32×32 multiplier were a 33% area increase and three times the clock rate of an unpipelined design.

2.Pipeline algorithm

An algorithm for pipeline insertion has been developed. It works on regular structures like adders, multipliers, and multi-stage operators with similar delays. It uses two main parameters, which include the number of circuit stages (s) and the number of pipeline stages (p). The algorithm generates (x) cells of granularity $(g1)=\left\lceil s/p \right\rceil$ where $x=\mathrm{mod}\left(s/p\right)$ and $(p-x)$ cells of granularity $(g2)=\left\lfloor s/p \right\rfloor$. Under this scheme, the granularity of each part of the FP unit can be independently adjusted. It is necessary to give the optimal pipeline parameter to each component to achieve optimal throughput of the FP unit. All the FP unit and subcomponents were designed to provide scalable mantissa and exponent fields as well as a variable number of pipeline stages.

3.FP Adder

Figure 1 shows the basic structure of the FP adder. Exponent and mantissa field widths are specified through parameters ***ebit*** and ***mbit***, respectively. The number of pipeline stages is specified through 5 parameters (***pip1, pip2a, pip2, pip3a and pip3b***). In the following sections we provide descriptions of the main operators, components and processes used by the FP Adder, along with their associated pipeline parameters.

Shifter: A right shifter is used to denormalize the smaller mantissa as required by the exponent equalization step. The shifter in Figure 2 uses a log-2 right scheme based on multiplexers [4]. Each bit in the operand specifying the number of shift positions works as a multiplexer's selector signal. Each multiplexer selects either the input vector or a shifted version of the input vector.

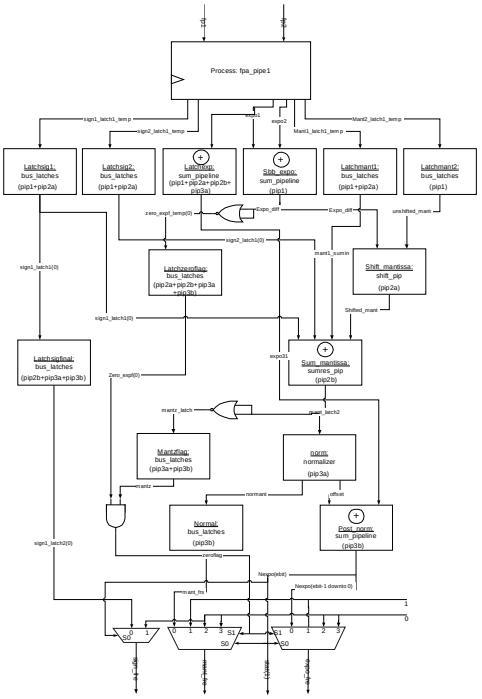

Figure 1: Pipelined FP Adder.

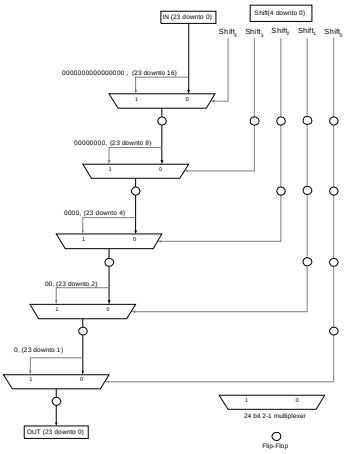

Figure 2: Scalable pipelined shifter

Adder: Figure 3 shows the scheme selected for pipelined fixed-point addition, based on the approach proposed by Dadda et al. [3]. Latches are used to propagate the signals through each pipeline stage to its subsequent stage.

Normalizer and leading zero detector: A topology shown in (Figure 4), has been developed, which follows the structure of the shifter. The last multiplexer's output is the normalized version of the unit's input. The result of each multi-input nor-gate is combined to form the total leading-zero amount. This topology improves over previous approaches by performing the zero leading detection and mantissa normalization in a single step, without the requirement of independent operations.

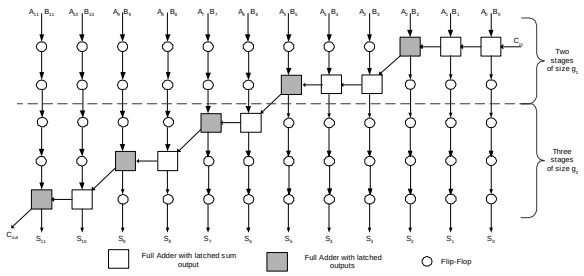

Figure 3: 12-bit pipelined adder (*s=12, p=5*)

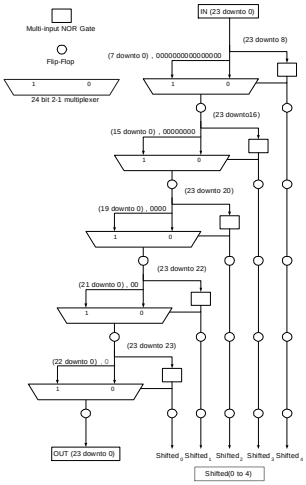

Figure 4: Normalizing and zero detection unit

3.1.FP adder components

Sbb_expo (*pip1*)**:** Subtracts both exponents to determine the number of shifting positions when denormalizing the smaller mantissa

Latchexp (*pip1, pip2a, pip2, pip3a*)**:** Adds one to the greater exponent.

Shift_mantissa (*pip2a*)**:** Right shifts the smaller number's mantissa.

Sum_mantissa (*pip2b*): Mantissa addition or subtraction depending on the sign of the input operands.

Norm (*pip3a*): Detects leading zeros on the mantissa and normalizes it.

Post_norm (*pip3b*): Adjusts the exponent result by subtracting the number of leading zeros provided by the normalizer.

Bus_latches: Maintain the data integrity through the pipeline.

3.2.FP Adder Processes

fpa_pipe1: Compares the input operands and swap them if necessary.

4.Floating Point Multiplier

Figure 5 shows the general organization of the FP multiplier. Exponent and mantissa widths are specified through parameters *ebit* and *mbit*, respectively. The number of pipeline stages is specified through 3 parameters (*pip1, pip1b, pip2*). The following sections provide descriptions of the main operators, components and processes used by the FP Multiplier, indicating their associated pipeline parameters.

Array multiplier: Figure 6 shows the method used to add pipeline to the array multiplier. Based on the scheme developed by [2].

Adder: Uses the same structure as those in the FP Adder.

4.1.FP Multiplier Components

Mult_mant (*pip1a, pip1b*): Performs mantissa multiplication. The top-portion of the array has *pip1a* stages. The bottom-portion has *pip1b* stages.

Add_expo (*pip1a, pip1b*): Performs exponent addition.

Add_bias (*pip2*): Performs subtraction of the exponent bias and exponent adjustments due to mantissa normalization.

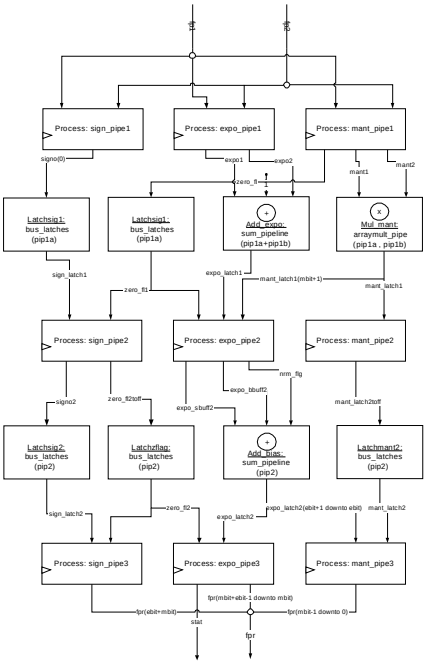

Figure 5: Pipelined FP Multiplier

4.2.FP Multiplier Processes

Sign_pipe1: Performs XOR of the signs.

expo_pipe1: Pass the exponents

mant_pipe1: Zero detection and adds the implicit hidden one to the mantissa.

sign_pipe2: Pass the zero flag and the sign.

expo_pipe2: Prepares the operands for exponent bias subtraction.

mant_pipe2: Normalizes the mantissa.

sign_pipe3: Modifies the sign in case of zero result.

expo_pipe3: Modifies the exponent in case of overflow or underflow or zero result. Also set the status flags.

mant_pipe3: Set underflow and underflow conditions.

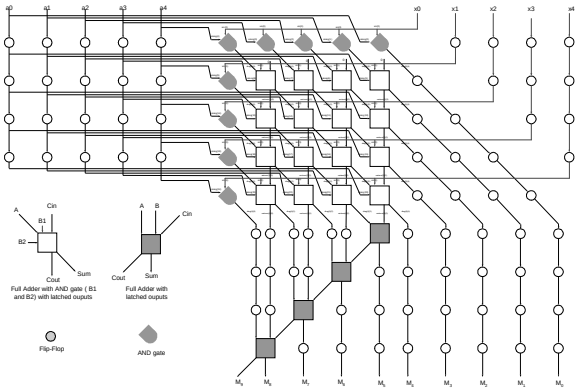

Figure 6: 5x5 pipelined array multiplier

5.Results

Several units were synthesized of both FP adder and multiplier to quantify the performance and space requirements under the reported approach. The synthesis was carried from a VHDL source and the target device was a Xilinx Virtex-II FPGA (2V1000FG456–6).

The effect of varying the number of pipeline stages on the speed of the FP units is illustrated in Figure 7. This graph shows that increasing the number of stages does effectively increase the operating frequency. This increase, however, has a variable rate mainly due to the routing delay, which sometimes achieves values over 50% of the worst delay path. Note that FP operands work at a lower frequency than its components because of the extra logic needed for FP arithmetic. The slowest component in the FP Multiplier is the array multiplier, while in the FP adder the bottleneck is created by the normalizer. This means that these components have priority in the assignment of pipeline parameters. Note also that increasing the number of pipeline stages

increases the consumption of FPGA resources as seen in. Figure 8 through the slice occupation. This increase seems to have a linear behavior and is mainly due to the increased usage of latches.

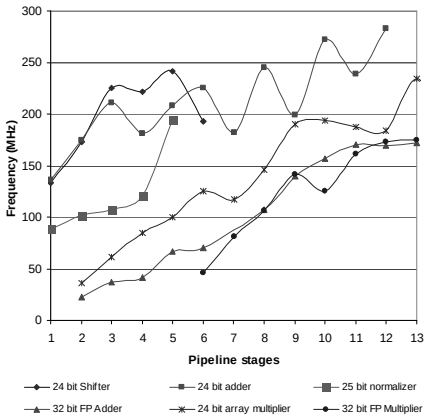

Figure 7:Operating frequency Vs Pipeline stages

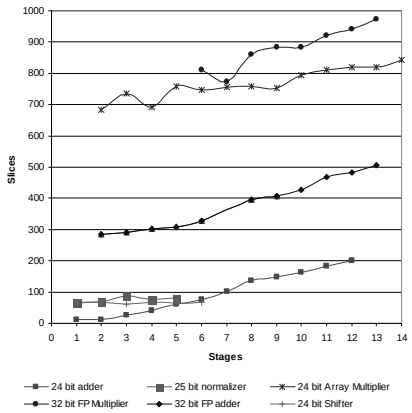

Figure 8: Slices consumption VS Pipeline stages

An implementation of single-precision, IEEE-754 compliant adder and multiplier units were found to operate at 170MFLPOS and 175MFLOPS, respectively. These speeds are competitive with those of highly refined, pre-routed core components commercially available units from several vendors. In terms of area, it results difficult to establish meaningful comparisons since the reference implementations use dedicated Virtex-II resources other than slices. Our approach tries to avoid the usage of such resources in order to keep the units portable to targets other than Virtex-II and to maintain the flexibility of adjustable range, precision, and pipeline granularity. Table 1 summarizes the obtained results along with typical speeds and resource utilization on some commercial implementations.

6.Conclusion and Future Work

An algorithm for pipeline insertion was developed and used to build several fixed-point, achieving operating frequencies well above 200MHz. Also a new topology for a mantissa normalizer was developed, which performs leading-zero detection and mantissa normalization in a single step without requiring an extra unit.

The FP operators achieved frequencies up to 175MHz. Our FP units compete well in terms of operating frequency, although there is room for improvement in terms resources consumption. Their advantage is the flexibility of scalable pipeline, mantissa and exponent fields as well as portability to a wide range of FPGA targets. This kind of flexibility is helpful for rapid prototyping and reconfigurable computing. Pipelining techniques developed here will be extended to the FP Square Root and FP Division operators. Further work includes automation of the pipeline optimization and insertion process.

Table 1: FP Adder comparations

FP unit	Source	Frequency	Slices	Latency	*mbit*	*ebit*
Adder	**Nallatech** [5]	184	290	14	24	8
	Quixilica [6]	147	121	11	20	6
	Ours	170	467	11	24	8
Multiplier	**Nallatech** [5]	188	126	6	24	8
	Quixilica [6]	122	326	6	24	8
	Ours	175	973	13	24	8

References

[1] P.A .A Walters. A scaleable FIR filter using 32-bit floating-point complex arithmetic on Reconfigurable computing machine. IEEE Symposium on FPGAs for Custom Computing Machines 1998 pp. 333–334.

[2] C.D.C.D. Asato. A data-path multiplier with automatic insertion of pipeline stages. IEEE Journal of Solid-State Circuits 1990. pp. 383 – 387.

[3] V. Dadda, L. Piuri. Pipelined adders. IEEE Transactions on Computers 1996. pp. 348 – 356

[4] S. Heo. A low-power32-bit datapath design. Master's thesis, Massachusetts Institute of Technology, 2000. pp. 66–76

[5] Nallatech Limited, "IEEE 754 Floating Point Core", 2001, Available HTTP: http://www.nallatech.com/

[6] QinetiQ, "Quixilica Floating Point Cores", 2002, Available HTTP: http://www.quixilica.com/