A Compact 4-Stage Pipelined 0.35µm Programmable Digital Filter

Hemant Savla, Hsinwei Chou, Harsha Mokkarala, and Anand Lakshmanan

***Abstract*—A Programmable Digital Filter modeling the equation $u(kT) = a_0.e(kT) + a_1.e((k-1)T) + a_2.e((k-2)T)$ is implemented at the transistor level with three 12x12-bit array multipliers and two 24-bit combinational adders. Transmission gate full adders were sized to achieve minimum area and respectable performance. A 4 stage pipeline utilizing two different clocks yields a high throughput design.**

Index Terms—**programmable digital filter, FIR, equal-delay full adder, array multiplier**.

I. Introduction

A Programmable digital filter (PDF) is a hardware implementation of a digital filter whose properties are programmable by external control signals. Filtering is the process by which the signal spectrum is convolved with the frequency domain impulse response of the filter in order to pass only the desired frequencies. There are two basic types of digital filters, Finite Impulse Response (FIR) and Infinite Impulse Response (IIR). IIR filters have one or more non-zero feedback coefficients, whereas FIR filters have none. This paper focuses on the design of a compact and high performance 0.35µ FIR filter. The general form of the FIR difference equation is:

$$y(n) = \sum_{i=0}^{N} a_i x(n-i)$$

This paper limits N to 2 in order to achieve a compact design. The cost to accuracy resulting from this restriction is insignificant for the inputs tested.

The a_i, x(n-i), and y(n) are 12-bit fixed point numbers in the range:

$$0 \text{ to } \sum_{i=1}^{12} 2^{-i}$$

The filter operates in 2 modes, configure and run. In configure mode, the a_i and x(n-i) are set to a starting condition. When the filter switches to run mode, the current input x(n) is combined with the starting condition to compute the new output.

The paper is organized as follows. Section 2 outlines the design methodology of the arithmetic modules in the filter. Section 3 presents an architectural description of the PDF, followed by a summarization of the clocking scheme in section 4. Circuit operation and a list of optimization steps taken are given in sections 5 and 6, respectively. Sections 7 analyzes the critical path, while section 8 lists all the components used in the design. Section 9 is a high level description of the chip floorplan, while section 10 summarizes the delay and area of the arithmetic modules. Section 11 illustrates the schematics and layouts for some of the key components, while closing remarks are given in section 12.

II. Design Methodology

After surveying a variety of paper on multiplier implementations, it became apparent that for a 12x12 multiplier, the low computation effort does not warrant a higher hardware complexity. For instance, multiplying two 12-bit numbers using a Booth Encoded Wallace Tree multiplier will not be significantly faster than using a straightforward array multiplier. The speed advantage derived from the more complex implementations is only significant in larger multiplications, such as 32x32 or 64x64 bits. In fact, after layout, a complicated design has a high probability of being slower than the standard array multiplier. This is because the routing capacitance resulting from irregular wiring can dominate the total delay. [1] points out this critical point. In addition a complicated multiplier will have a substantial area overhead. Therefore, an array multiplier implementation is chosen for its low area, and routing regularity. Since each component in the array needs to connect only to an adjacent component, the routing capacitance is minimized.

For reasons explained later, the choice of 24-bit adder implementation is not significant because it is not the bottleneck in the pipeline. Therefore, a ripple-carry static-logic adder is selected for area savings. Both the multiplier and the adder are constrained to static-logic for compatibility.

III. Architectural Description

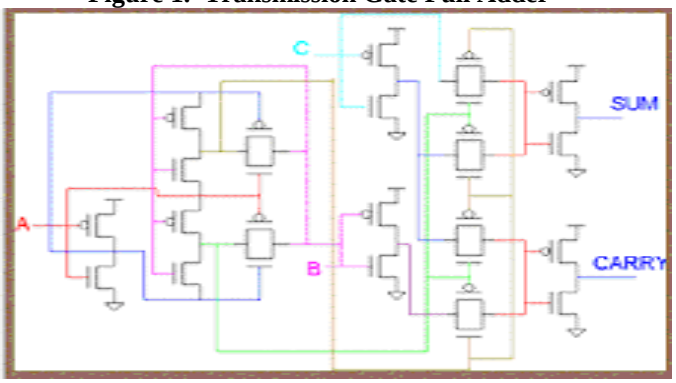
The filter is comprised of three 12-bit multipliers, two 24-bit adders, a control unit, and several latches and flipflops for

pipelining purposes. The full adders used in the multipliers and the adders are the same.

A. *Transmission Gate Full Adder*

Figure 1 shows a transmission gate full adder implementation. This adder has 26 transistors, the same as the combinational static adder, but has the advantage of having equal SUM and CARRY delay times. In addition, the SUM and CARRY are not inverted. Due to the equal arrival times of SUM and CARRY, the transmission gate adder glitches less often than a fully combinational adder, making it a favorable circuit style for high performance arithmetic blocks.

Figure 1. Transmission Gate Full Adder



B. *12x12 Square Array Multiplier*

A straightforward array multiplier is designed using the transmission gate full adder as the basic building block. A square realization of the array multiplier is implemented for compactness as well as layout regularity, which reduces the routing capacitance. Figure 2 illustrates a sample 4x4 square rearrangement of an array multiplier. Each full adder (FA) cell is connected to an adjacent FA cell as shown in figure 3.

Figure 2. Sample 4x4 Square Array Multiplier

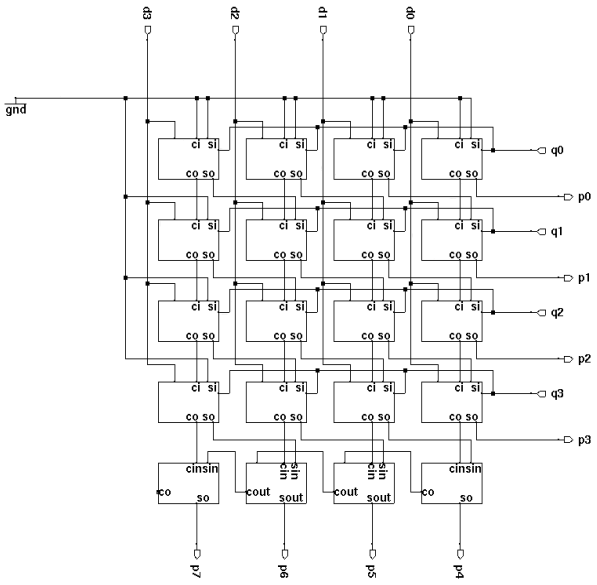
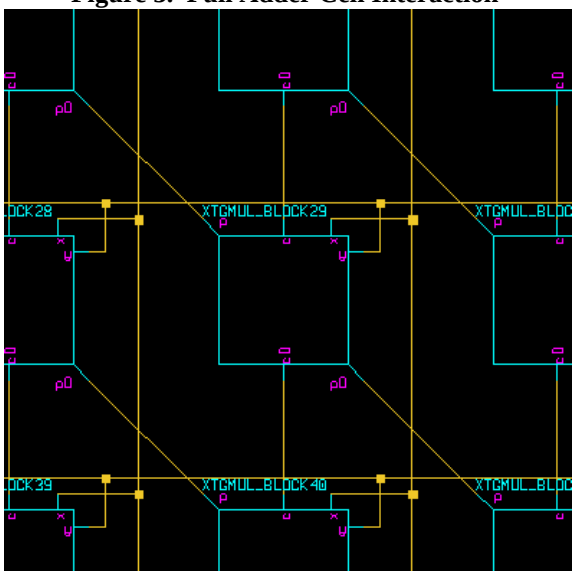


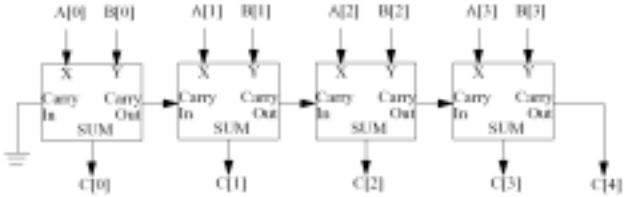
Figure 3. Full Adder Cell Interaction



C. *24-bit ripple-carry adder*

The adder is composed of 24 transmission gate full adders connected in a traditional ripple-carry fashion, as seen in figure 4. In the pipeline, the bottleneck is the multiplier, not the adder. Therefore, a ripple-carry implementation suffers no loss in performance compared to a carry-look ahead adder, but offers significant savings in area.

Figure 4. Ripple-Carry Adder



D. *Control Block*

The control signals are A_0LOAD, A_1LOAD, A_2LOAD, U_0LOAD, E_0LOAD, and E_{-1}LOAD. Each signal is generated through a 4-input AND gate, with the inputs being the corresponding Const[2:0] values and mode. In run mode, the registers storing the A's retain their values. E_0LOAD and E_1LOAD are also low, but there is an external circuitry to allow the E(kT) shift registers to remain on.

E. *Latches and FlipFlops*

Three 24-bit positive-edge flipflops form a shift register to store the E(kT) values. Separately, 24-bit single-phase positive level-sensitive latches save A_0, A_1, and A_2 during configure mode. The outputs from the multipliers and the adders also utilize the same type of latch for storage in run mode.

Single-clock latches were chosen over two-phase implementations in order to utilize a two-phase clocking

scheme. Other motivations include reduction of timing complexity and readily available layouts.

IV. Clocking scheme

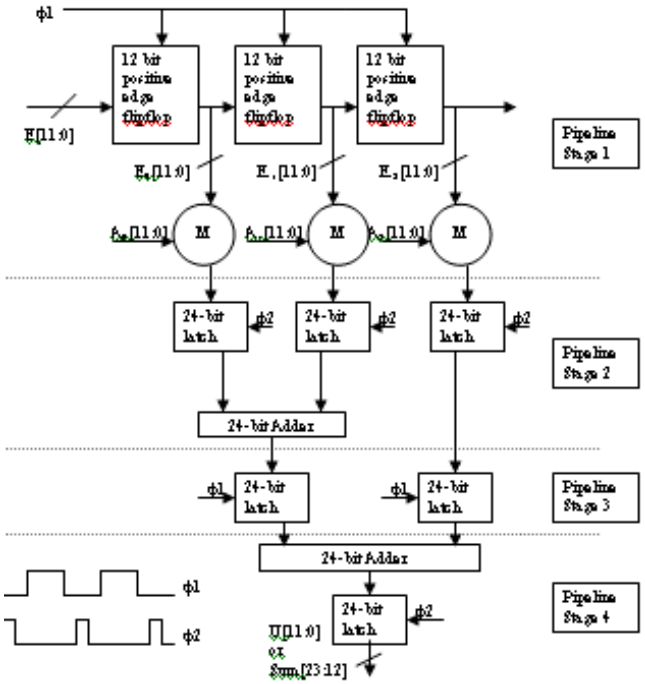
$\phi1$ and $\phi2$ are non-overlapping clocks. $\phi2$ is not the complement of $\phi1$. On the rising edge of $\phi1$, three 12-bit shift registers latch onto the previous value of E(kT). After $\phi1$ transitions low, a short period of time will pass before $\phi2$ transitions high. During the high cycle of $\phi2$, three 24-bit latches will store the output of the three multipliers corresponding to A_0E_0, A_1E_{-1}, and A_2E_{-2}. On the next cycle, when $\phi1$ is high, two 24-bit registers will latch onto the result of an adder and the output of the third multiplier, again. Finally, during the high cycle of $\phi2$, a 24-bit latch will store the result of the second adder and output it on U[11:0].

SH goes high a short time before $\phi1$ rises. On the rising edge of $\phi1$, the inputs must be present on the E[11:0] in order to be latched in. The time duration between the rising edge of SH and the rising edge of $\phi1$ is the setup time for the inputs E[11:0]. The hold time for E[11:0] is the delay of a positive-edge triggered flip-flop. The output U[11:0] is guaranteed to be stable at one positive level-sensitive latch delay after the rising edge of $\phi2$.

The multiplier delay is the bottleneck in the entire filter, which is also the pipeline period.

The clocking scheme and the 4-stage pipelined structure is explained in figure 5.

Figure 5. General PDF Clocking Scheme



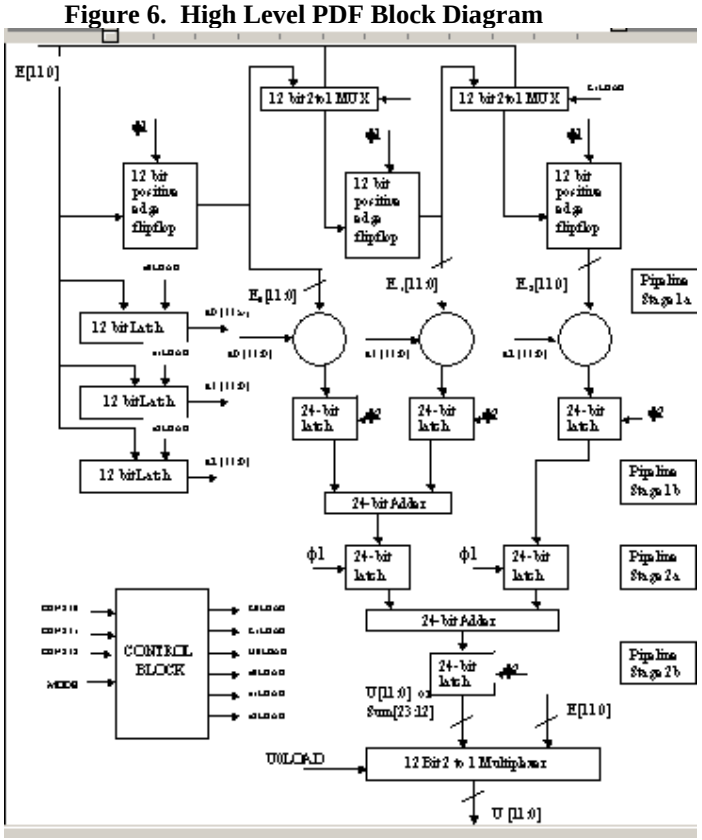
V. Circuit operation

The PDF operates as follows. Starting in configure mode, the inputs const[2:0] are varied to load a_0, a_1, a_2, e_0, and e_{-T} signals into the approriate registers. Unlike the a_i latches, the e_i registers are connected to each other as a shift register unit. Therefore, multiplexers are used to directly set the e_0, and e_{-T} values during configure mode. Similarly, a user may externally set the output u_o through a multiplexer that links the E[11:0] pins to the register storing U[11:0]. Once all these initial conditions are set, run mode initiates.

In run mode, the e_i shift registers are continuously on. For every clock cycle, a new input e_0 is shifted in, while the previous two e_i values slide to the next register. The e_{-2} value . is erased permanently. Once all e_i's get latched, the multiplier calculates the product a_i*e_i and stores it in another latch. Then, 2 adders sequentially add all three products and output the result on U[11:0].

Figure 6 shows a high level block diagram of the PDF design.

Figure 6. High Level PDF Block Diagram



VI. Optimizations

The design was optimized with the following ideas strictly adhered:

a. For a digital filter, the throughput of the design outweighs the overall latency.
b. The bottleneck of the design is the array multiplier, not the ripple-carry adder.
c. The area and regularity of layout must be carefully monitored during optimization. Routing delay can easily overshadow the performance gain from sizing and architectural tuning. Irregular layout complicates the area overhead and time-to-market completion time.

A. Architectural Improvements

- 4-stage pipeline – the datapath is executed in 4 stages, input, multiply, add1, and add2:
  - Input – previous E(kT) values shift and the new input E(0) latches in.
  - Multiply – 3 multipliers execute in parallel, computing A_iE_i, then the outputs are saved in three latches.
  - Add1 – the first 2 multiplier products are added and stored, and the third multiplier's result is relatched to a different register.
  - Add2 – the third multiplier's result is added to the sum of the other 2 products and stored into the output pins, throwing away the carry-out and the last 12 LSBs.

  With the pipelining, the very first computation consumes 2 clock cycles. Afterwards, the result is available in every cycle.

  Delay values for worst case multiplier and adder computations are 5.67ns and 3.72ns respectively. Therefore, a 3-stage pipeline with both adders finishing before the multiplier is impossible. With 4 stages, a high throughput is achieved at the sacrifice of overall latency. Considering the recursive nature of a digital filter, the benefit more than outweighs the cost.
- Noice's 2-phase clocking – Noice's rules are followed in order to prevent clock skew. A $\phi 1$ latch outputs a stable $\phi 2$ signal, which is then passed to a $\phi 2$ latch after arithmetic processing. The same is true vice versa.
- Square array multiplier – since the area of the multiplier is the major concern, the array multiplier can be structured as a regular square. This minimizes both the chip area as well as the routing delay.

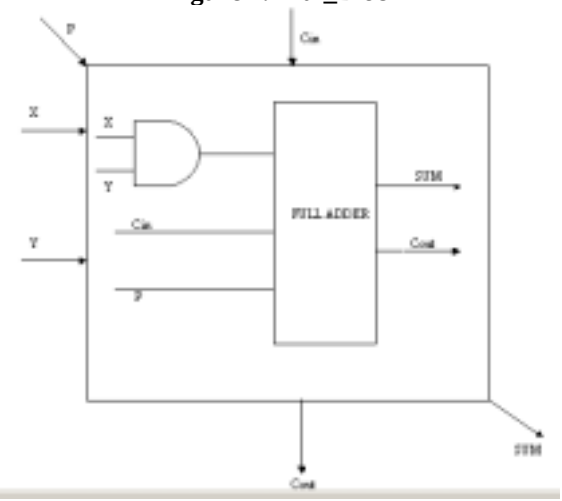
B. Circuit Optimizations

- Glitch minimization – glitches impact the performance because erroneous transitions incur a correction time overhead. Reducing the glitching probability will lead to performance boosts. Since glitches in full adders originate from unequal arrival time of SUM and CARRY, the full adder used in the multiplier as well as the 24-bit adder was implemented using transmission gate logic, which inherently has equal SUM and CARRY delay. An alternative 10-transistor SERF full adder [5] was implemented and tried as the building block of the multiplier and the adder. The result was very glitchy and impacted the delay significantly. Therefore, a conclusion drawn is that glitchless full adder operation is crucial to optimizing the delay of the entire PDF design.
- Transistor sizing – the transmission gate full adder was sized to achieve a good delay and yet not exceed 30λ for area considerations. Other components in the multiplier, such as the AND gates, were sized up to a moderate width of 30λ to achieve an optimal tradeoff between performance and area. Because the multiplier is the bottleneck, the adder's delay is not a major concern in the pipelined system. Therefore, transistors in the adders were scaled down to achieve a low area.
- Latch selection – latches were chosen over D flipflops as the storage unit for area savings. The e_i shift registers, however, require a D flipflop implementation for proper operation because a latch would lead to continous shifting from the 1st register all the way down to the last shift register. Variants of the level sensitive latch are employed, such as the 2-phase latch and the traditional single phase latch. The 2-phase latch is only used in storing the A_i's, since these values remain stable during run time regardless of $\phi 1$ and $\phi 2$. 2-phase latches are more area efficient than single-phase ones, so some area reduction is achieved.

C. Layout Enhancements

- Mul_blocks – since each partial product is an input to a full adder, routing area can be reduced through the formation of mul_blocks, a building cell for the array multiplier. Instead of generating the partial products elsewhere and routing them to their destination full adders, the partial product generator, a 2-input AND gate, can be combined with the full adder to form a mul_block cell. This way, the AND gate partial product output is directly routed to a neighboring full adder, saving routing area and delay. Figure 7 illustrates a mul_blocks.

Figure 7. Mul_Block



- Power supply rail placement – both VDD and GND lines are placed on the top and bottom of each row of mul_block so that they can be shared by all blocks above and below it. This results in a significant global routing area reduction.
- L-shaped registers – the output latches of the multiplier are 24 bits wide, while the multiplier itself is 12x12. In order to reduce the latch area, the latches are laid out into a L-shape. This way, they can bend and wrap around the multiplier to save area.
- Decomposed control block – The entire control block, which generates all the load signals, is decomposed into separate, small blocks that generate the control signals individually. This allowed the smaller control units to fit into gaps in the PDF design, where the original, entire control block could not fit. This efficient usage of space leads to a reduction in total chip area as well as routing delay minimization.
- Over-the-cell routing – control lines and the clock signal are routed over the cells using metal4, which reduces the total routing area.
- Compact floorplan – the floorplan for the entire design was carefully done to achieve the most compact area. More details can be found in section 9.
- Exact width allocation – in some locations, the exact width of blocks to be placed is pre-calculated so that there is no approximate allocation that leads to potential waste of space. A formula used to compute the required allocation is:

$$A = N*W + (N-1)*D$$

Where A = the width to be allocated
N = the number of lines to be routed
W = minimum width of each line
D = minimum spacing between lines

Below is a summary list of all the optimizations discussed:

Figure 8. Summary of Optimizations

Architecture improvement	Circuit improvement	Layout improvement
4-Stage Pipeline	Glitch Minimization	Mul_blocks
Noice's Rule	Transistor Sizing	Power Supply Rail Placement
Square Array Multiplier	Latch Selection	L-Shaped Registers
		Decomposed Control Block
		Over-The-Cell Routing
		Compact Floorplan
		Exact Width Allocation

VII. CRITICAL PATH

The critical path for the design is defined to be the worst case delay of the multiplier. For an array multiplier, this corresponds to an input pattern that propagates a carry through as many bits as possible. Since a carry can only propagate when there is an odd number of 1's in a column of partial products, the critical path was derived to be the delay of the input pattern 100000000001 * 111111111111, shown in figure 9.

Figure 9. Array Multiplier Critical Path

100000000001
100000000001
100000000001
100000000001
100000000001
100000000001
100000000001
100000000001
100000000001
100000000001
100000000**1**

For the 24-bit ripple adder, the critical path corresponds to the input pattern 111111111111 + 000000000001. For this input set, a carry will be rippled starting from the 0^{th} bit all the way out of the 23^{rd} bit.

VIII. COMPONENT TABLE

Figure 10 lists all the components in the design.

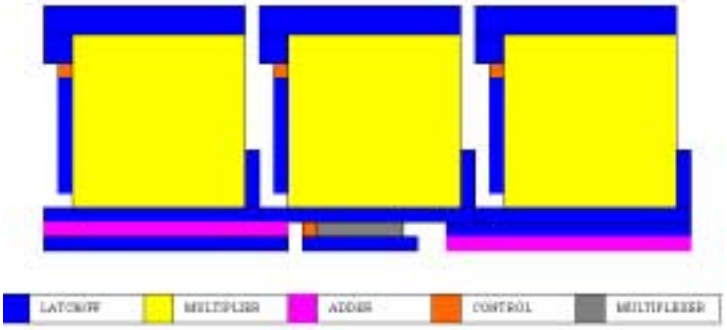
Figure 10. PDF Components

	Number of Instances in the PDF Design	Usage
12x12 Array Multiplier	3	$A_i * E_i$
24-bit Ripple Transmission Gate Full Adder	2	$(A_0 * E_0) + (A_1 * E_{-T}) + (A_2 * E_{-2T})$
24-bit Positive Edge FlipFlops	3	Shift Registers storing the E_i's
24-bit latches	3(configure) + 6(run) = 9	Saves A_0, A_1, and A_2 in configure mode. In run mode, they store the adder and multiplier outputs, $(A_i * E_i)$ and $(A_i * E_i) + (A_i * E_i)$
Multiplexer	3	Enables u_0, e_0, and e_{-T} to be externally set during configure mode
Control block	1	Outputs control signals for the appropriate latches
Transmission Gate Full Adder	3 * (number of instances in the array multiplier) + 2 * (24)	Building block of the multiplier as well as the ripple-adder

IX. CHIP FLOOR PLAN

Through efficient floorplanning and routing, the final PDF area is only slightly bigger than the combined area of the 3 multipliers. The overall chip floorplan is shown in figure 11.

Figure 11. PDF Floorplan



X. AREA, DELAY, AND MAXIMUM CLOCK SPEED

	Worst Case Delay(ns)	Area (um^2)
Array Multiplier	5.67	1011x1412
Ripple-Adder	3.72	1975x102
PDF	Does Not Apply, output is latched	4821x2214

T = 5.67 ns (multiplier delay, pipeline)
Maximum Clock Frequency = 176.37 MHertz

XI. KEY COMPONENT SCHEMATICS AND LAYOUTS

The following figures show some key PDF components with appropriate sizing. The full adder and the AND gate comprise the multiplier and the ripple-adder entirely.

Figure 12. Transmission Gate Full Adder

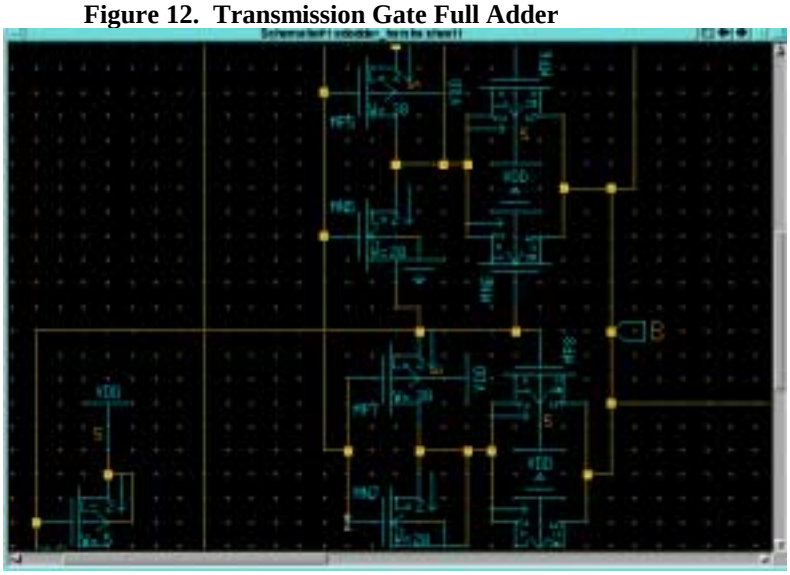


Figure 13. AND gate in mul_block

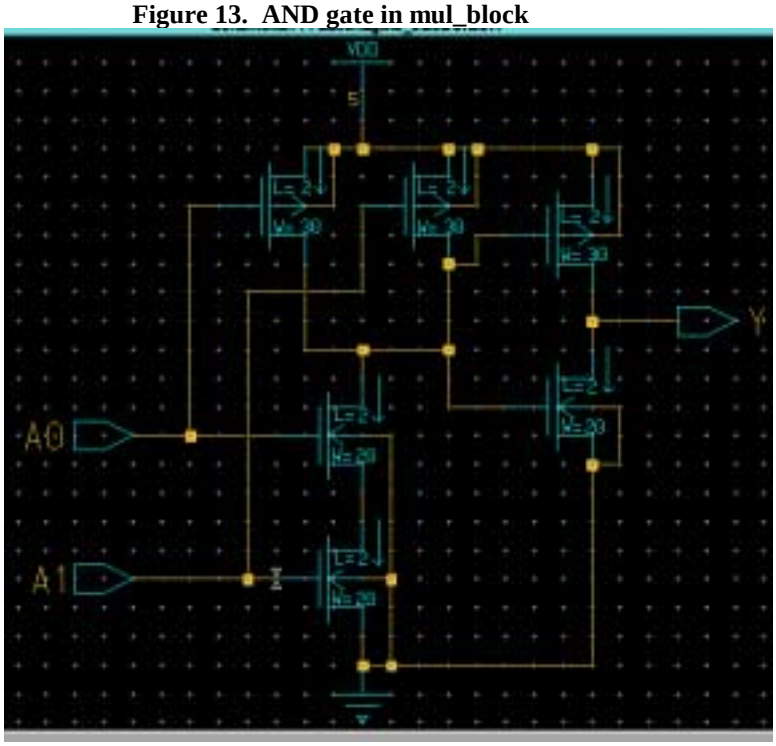


Figure 14. PDF Layout

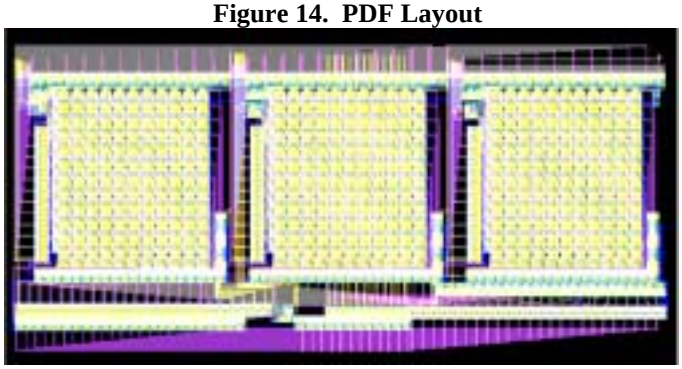


Figure 15. 12x12 Array Multiplier Layout

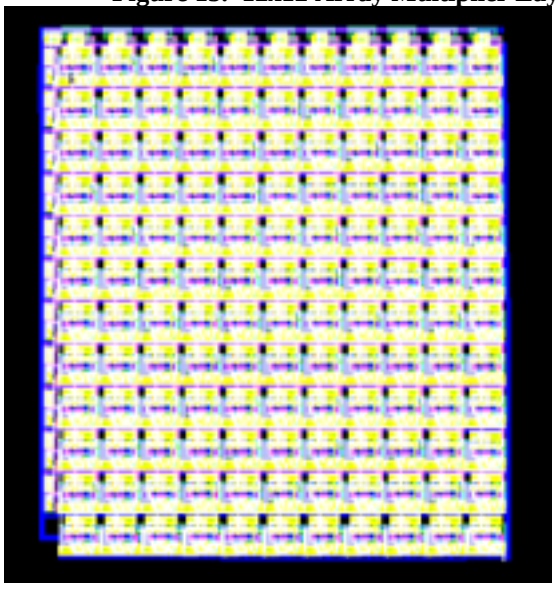
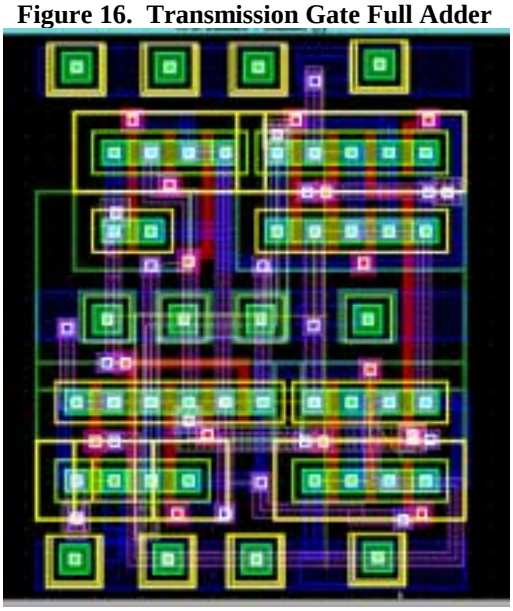


Figure 16. Transmission Gate Full Adder



XII. Conclusion

A digital filter is designed to operate continously in run mode, sampling an input and computing an output for every clock cycle. The recurrent nature of the filter mandates a high throughput, even if it impacts the overall latency. Through a 4-stage pipeline, the design given in this paper is able to satisfy the throughput demand with a low latency. The chip area is significantly compact due to the regularity of the square array multiplier and efficient bending of transistors during the layout process. A full programmable digital filter with high throughput, compact area, and decent latency is achieved in this design.

References

[1] http://www.andraka.com/multipli.htm
[2] Weste, Eshraghian, “Principles of CMOS VLSI Design”. Equal delay transmission gate full adder: pp. 524-526. Square array multiplier: pp. 547-548.
[3] Shailesh Shah, “Design and Comparison of 32-bit Multipliers for Various Performance Measures”, Meng. Project Report, Concordia University, January 2000.
[4] C.S. Wallace, “A Suggestion for a Fast Multiplier”, IEEE *Trans. Elecron. Comp.*, vol. EC-13, pp. 14-17, Feb. 1964.
[5] Bui, Wang, and Jiang, “Design and Analysis of Low Power 10-Transistor Full Adders Using Novel XOR-XNOR Gates.