An Area Efficient and High Speed Programmable Digital Biquad Filter

by

Siu Kuen Silas Li

A thesis submitted in conformity with the requirements
for the Degree of Master of Applied Science in the
Graduate Department of Electrical and Computer Engineering,
University of Toronto

Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

0-612-29413-7

Canada

Biquad Filter

Siu Kuen Silas Li
M. A. Sc., 1997
Department of Electrical and Computer Engineering
University of Toronto

Abstract

This thesis presents an area efficient and high speed implementation of a Δ–Σ based biquad filter. This filter processes a 1-bit Δ–Σ modulated signal at an oversampled rate directly without downsampling and upsampling. To save area, the filter architecture uses only one simple ALU to perform all arithmetic for the filter. The implementation of the filter chip uses true-single-phase-clocking dynamic logic. TSPC logic is used to pipeline the filter at the complex-gate level, and improves the clock rate and throughput of the filter significantly.

The filter chip has been sent to CMC to be fabricated in a 3-layer metal 0.8μm BiCMOS process. It occupies an area of 3140×3440 μm^2. HSPICE simulations show that the filter should work at the clock rate of 660 MHz. At this clock rate, the filter can handle input at a maximum sampling rate of 18.3 MHz, corresponding to a bandwidth of 36.6 kHz with an OSR of 256.

Acknowledgments

I would like to thank my supervisor Professor David M. Lewis for his supervision, patience, advice, and guidance throughout my thesis research and write-up. Without his help, this thesis would not be possible.

I want to also thank my parents and my fiancee Patkie P. S. Cheung for their constant encouragement and support. In addition, I would like to thank everyone in this department for their friendship and help. In particular, I want to thank Jamil Ahmed, Aris Balatsos and Gary Liu for their advice, support, encouragement and help.

Finally, NSERC's financial support and CMC's support for fabrication are gratefully acknowledged.

CHAPTER 1 *Introduction*

1.1 Motivation

The use of delta-sigma (Δ–Σ) modulation in the implementation of the analog-to-digital (A/D) converters and the digital-to-analog (D/A) converters is a popular method for high resolution signal conversion [2],[5],[6]. In traditional digital filter processing [3],[18], the modulated signal at the oversampled rate is first downsampled to the Nyquist rate by a decimation filter. The decimated signal is then processed by the digital filter operating at the Nyquist rate, which is twice the signal bandwidth. The filtered signal is finally upsampled to the oversampled rate by an interpolation filter before being used by the D/A converter. These filters not only introduce long latency in the filter processing but also consume costly hardware [11],[12],[14].

References [11],[12] and [21] proposed several filter structures to process the modulated signals at the oversampled rate directly. With these filter structures, digital filtering can be performed at the oversampled rate. Therefore, they eliminate the need for the decimation and interpolation processes inherent in the traditional digital filter processing performed at the Nyquist rate.

modulate the multi-bit internal signals to 1-bit signals. This multi-bit to 1-bit conversion greatly simplifies the arithmetic involved in filtering. Fast arithmetic operations are possible in these Δ–Σ based filters. Several filter architectures proposed by [11] were implemented. In the filter designs of [1], [12], [14] and [17], carry-save adders were often used to reduce the long propagation delay. Partial carry-propagation was also used in the filter design of [14] to further improve speed. However, all of these filters directly implement each of the adders, multipliers, and quantizers. Although deeply pipelined versions [14] can attain high speed, all of these filters require significant area. There is a strong motivation to extend their work to minimize the filter area.

1.2 Objectives

This thesis presents a new filter architecture for the Δ–Σ based biquad filter described in [11]. The new filter architecture is both area efficient and high speed. The biquad filter uses Δ–Σ modulators to modulate internal multi-bit signals to single bit signals, which simplifies the filter arithmetic. It also uses carry-save addition and the 2-level quantization scheme with partial carry-propagation described in [14] to reduce delay. In the new architecture, a simple ALU which is able to perform carry-save addition and the XOR function is used to perform all the arithmetic required for the biquad filter. Since only one ALU is used, the design approach saves significant silicon area. The use of simple logic functions in the implementation of the biquad filter allows fast logic operations. True-single-phase-clocking (TSPC) dynamic logic is used in the VLSI implementation of the biquad filter chip, and enables the filter to operate at a high clock rate [7], [14], [16],[22]. Hence, the filter attains high throughput and high speed.

This thesis contains five chapters. Chapter 1 gives the motivation and objectives of this thesis. Chapter 2 provides the background information related to this thesis. The detailed description and principles of the Δ–Σ modulation are presented first. Chapter 2 then discusses the application of Δ–Σ modulation in the biquad filter and TSPC dynamic logic. Chapter 3 describes an area efficient architecture of the biquad filter using a single ALU approach. Moreover, the register-transfer-level design of the filter using the new architecture is presented at the end. Chapter 4 discusses the VLSI implementation of the biquad filter and other VLSI design issues involved such as I/O interface, clock generation, clock distribution, and power distribution. Chapter 5 concludes this thesis.

CHAPTER 2 *Background Information*

This chapter discusses the background information related to this thesis. The principles of Δ–Σ modulation and the biquad filter are discussed in detail. The goal of this thesis is to design an area efficient high speed biquad filter by exploring a filter architecture using Δ–Σ modulation, and circuit implementation using true-single-phase-clocking (TSPC) dynamic logic.

This chapter has three sections. Section 2.1 describes the principles of Δ–Σ modulation. The applications of Δ–Σ modulation on the biquad filter is discussed in section 2.2. Finally, the principles of TSPC dynamic logic is described in section 2.3.

2.1 Delta-Sigma Modulation

If Δ–Σ modulated signals are used in A/D and D/A conversion, processing these signals at the Nyquist rate introduces two additional filters, the decimation filter and the interpolation filter [11]. Filtering these oversampled Δ–Σ modulated signals at the oversampled rate eliminates the need for the decimation filter and the interpolation filter. Thus, the circuit complexity and signal delays are reduced significantly, and the filter

efficiently implemented [11]. Figure 2.1 shows the comparison of the conventional filter pipeline and the Δ–Σ based filter pipeline.

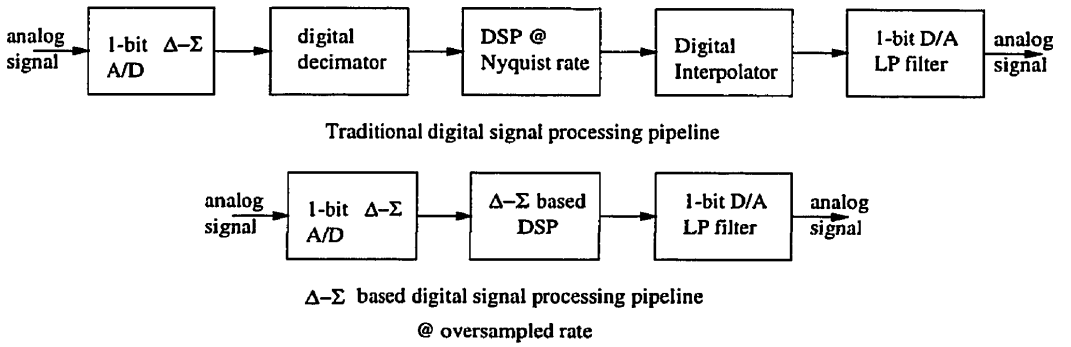


FIGURE 2.1 Conventional filter pipeline vs. Δ–Σ based filter pipeline [11]

Before the discussion of the fundamentals and the application of Δ–Σ modulation in this filter, the definition of the oversampling ratio (OSR) is given in EQ 2.1:

$$OSR = \frac{f_o}{2f_s} \qquad \text{(EQ 2.1)}$$

where the f_o is the oversampled rate and the term $2f_s$ is the Nyquist rate.

2.1.1 First-order Δ–Σ modulator

Figure 2.2 shows a first-order Δ–Σ modulator. The input signal X is at the oversampled rate f_o which is much higher than the Nyquist rate $2f_s$ of the signal. The difference D between X and the analog output of the feedback D/A converter is integrated, and the integrated result is then converted to the output digital signal Y. The feedback ensures that the value of the output Y is tracked by the input value. Thus the average value of the output Y is equal to that of the input X [18]. Figure 2.3 shows the discrete time counterpart of the first-order Δ–Σ modulator.

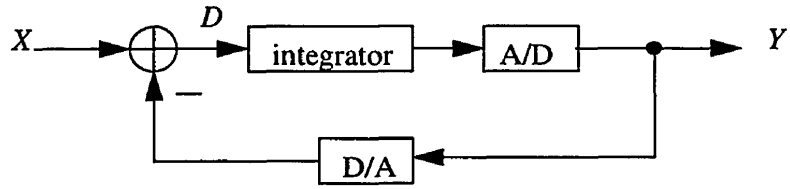


FIGURE 2.2 First-order Δ–Σ modulator

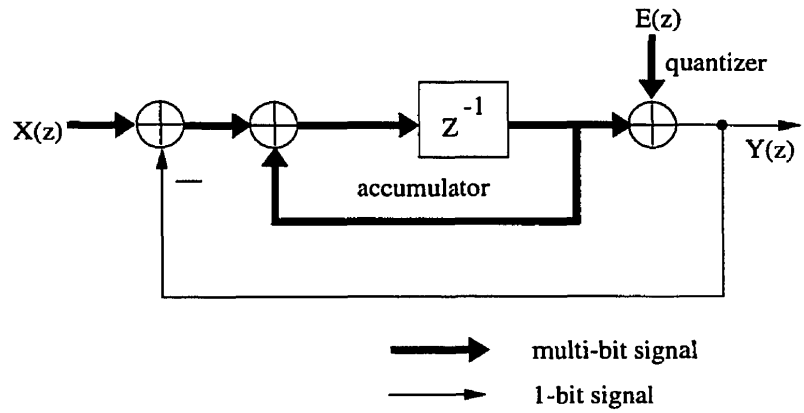


FIGURE 2.3 Discrete time model of the first-order Δ–Σ modulator

The discrete time model uses an accumulator instead of the integrator in the analog model. The A/D converter in the analog model is represented by an additive noise source *E(z)*. EQ 2.2 describes the first-order Δ–Σ modulator:

$$Y(z) = X(z)\, z^{-1} + (\, 1 - z^{-1}\,)\, E(z) \qquad \text{(EQ 2.2)}$$

The output *Y(z)* is the sum of the delayed value of the input *X(z)* and the shaped value of *E(z)*. The term $(1 - z^{-1})$ does not distort the signal but shapes the noise *E(z)*. It is called the noise transfer function, $H_n(z)$, of the first-order Δ–Σ modulator. $H_n(z)$ has high-pass filter characteristics which shape the noise power spectral density so that most of the noise power spectral density concentrates in the high out-of-band region (high frequency region) leaving the in-band region (low frequency region) for the high resolution signal *X(z)*. The additive noise source is due to the quantization error of the signal. It is usually assumed to be white-

power of P_n with step size Δ is given in EQ 2.3 [18].

$$P_n = \sigma_e^2 = \frac{\Delta^2}{12} \qquad \text{(EQ 2.3)}$$

For the first-order Δ–Σ modulator, with a sampling frequency, $f_o >> 2\ f_s$, the in-band quantization noise power, σ_n^2, of the first-order Δ–Σ modulator is given in EQ 2.4 [18].

$$\sigma_n^2 = \frac{\pi^2 \sigma_e^2}{3 \cdot OSR^3} \qquad \text{(EQ 2.4)}$$

It can be shown that doubling the sampling frequency and hence the OSR reduces the noise power by 9 dB because of the term OSR^{-3} [18].

2.1.2 Second-order Δ–Σ modulator

The noise power in Δ–Σ modulation can be reduced by increasing the sampling rate and using a higher order modulator [11],[18]. Since the quantization noise is not perfectly white over the entire frequency band, higher order modulators are considered in the biquad filter design because of their better noise-shaping ability than that of a first order modulator. In the design of the biquad filter, a second-order Δ–Σ modulator is used because of its simple structure and good noise-shaping ability. The simple structure allows possible high speed realization.

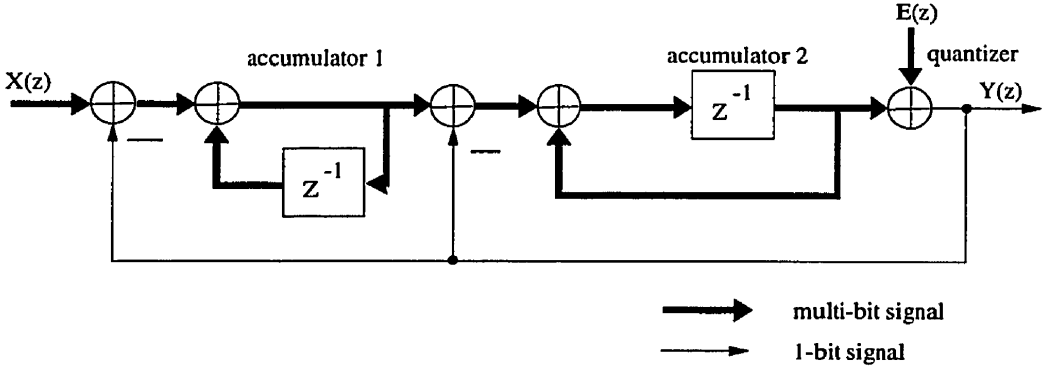


FIGURE 2.4 Second-order Δ–Σ modulator

EQ 2.5 describes the second-order Δ–Σ modulator shown in Figure 2.4 [18].

$$Y(z) = X(z)\, z^{-1} + (\, 1 - z^{-1}\,)^2\, E(z) \qquad \text{(EQ 2.5)}$$

Similar to the first-order modulator, the output $Y(z)$ is the sum of the delayed value of input $X(z)$ and the shaped value of $E(z)$. The term $(1 - z^{-1})^2$ is the noise transfer function, $H_n(z)$, of the modulator. This second-order noise transfer function, $H_n(z)$, has higher noise suppression ability than that of the first order modulator. The in-band quantization noise power, σ_n^2, of the second-order Δ–Σ modulator is given in EQ 2.6 [18].

$$\sigma_n^2 = \frac{\pi^4 \sigma_e^2}{5 \cdot OSR^5} \qquad \text{(EQ 2.6)}$$

EQ 2.6 implies a 15-dB reduction in the quantization noise power for every doubling of the sampling frequency, f_o, because of the term OSR^{-5} [18]. For the first-order modulator, only a 9-dB reduction in the noise power is obtained for every doubling of f_o. Hence, the noise-shaping ability of the second-order modulator is almost twice that of the first-order one in the in-band region.

In addition to using a higher order modulator, the in-band noise power can be reduced using a multi-level quantization scheme [1]. However, a multi-level quantizer increases the complexity of the multipliers. Therefore, it is not desirable in the design of the biquad filter because a complex circuit will slow down the clock rate. Thus, the modulator of this thesis uses a two-level quantization scheme because the two-level quantization scheme requires a small amount of hardware to implement. The output of the modulator using a two-level quantizer is either -1 or 1. If it is multiplied by a k-bit coefficient, a, the result of multiplication is always either $-a$ or $+a$. Therefore, this $1 \times k$-bit multiplication can be efficiently realized. The design of this type of multiplier will be described in Chapter 3. Since multiplications are common in digital filters, the multipliers may limit the filter to run at low speed. Simplification of multiplication to multipliers of ± 1 in the filters helps to attain high speed.

2.2 *Application of Δ–Σ Modulator in the Biquad Filter*

Several finite-impulse-response (FIR) filters using the Δ–Σ modulation encoding were studied in [21]. However, their specifications can be often met using lower-order infinite-impulse-response (IIR) filters. Reference [11] described a design approach where internal filter signals are remodulated by Δ–Σ modulators to simplify the filter arithmetic operation such as multiplication. Several Δ–Σ based IIR filters were discussed in [11]. One of these IIR filters is the Δ–Σ based biquad filter. The following sections discuss the application of the Δ–Σ modulator in the biquad filter in detail.

2.2.1 Biquad filter

A general biquadratic transfer-function is given in EQ 2.7 [11].

$$T(z) = \frac{n_2 z^2 + n_1 z + n_0}{z^2 + p_1 z + p_0} \qquad \text{(EQ 2.7)}$$

The structure of the biquad filter allows a simple implementation in hardware [11], [17]. In addition, higher order filters can be realized by cascading several biquad filters with carefully chosen poles and zeroes. The biquad's simple structure allows a high speed and area-efficient hardware implementation. If second-order Δ–Σ modulators are used in the biquad filter to convert internal multi-bit signals to single bit signals, the modulator will introduce delay. If the biquad filter uses Δ–Σ modulators internally, the biquad structure should be based on two delaying integrators with input summing to obtain the correct transfer-function zeroes. The input summing approach eliminates the need for an extra modulator to convert a multi-bit output signal to a single-bit signal like the output summing approach [11]. The single-bit output can be delivered to the Δ–Σ based D/A converter directly. Figure 2.5 shows the block diagram of a biquad filter using two delaying integrators. The coefficients, a_0, a_1 and b_{0-2}, describes the characteristics of the filter (see EQ 2.8). *U(z)* is the 1-bit input signal and *Y(z)* is the multi-bit output signal.

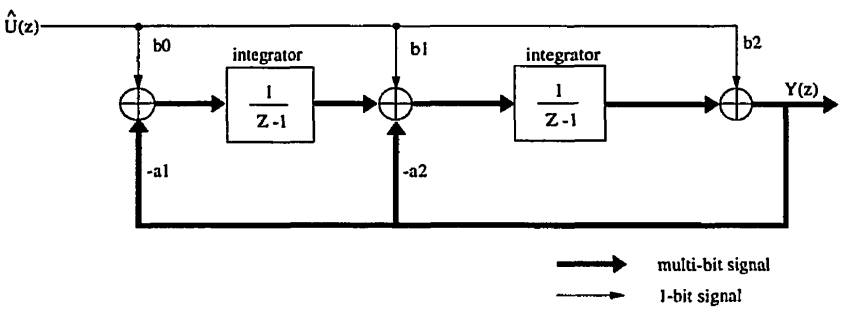


FIGURE 2.5 Biquad filter using two delaying integrators

2.2.1.1 Δ–Σ based biquad filter

Figure 2.6 shows a Δ–Σ based biquad filter with input summing [11], which is obtained from the biquad filter using two delaying integrators shown in Figure 2.5.

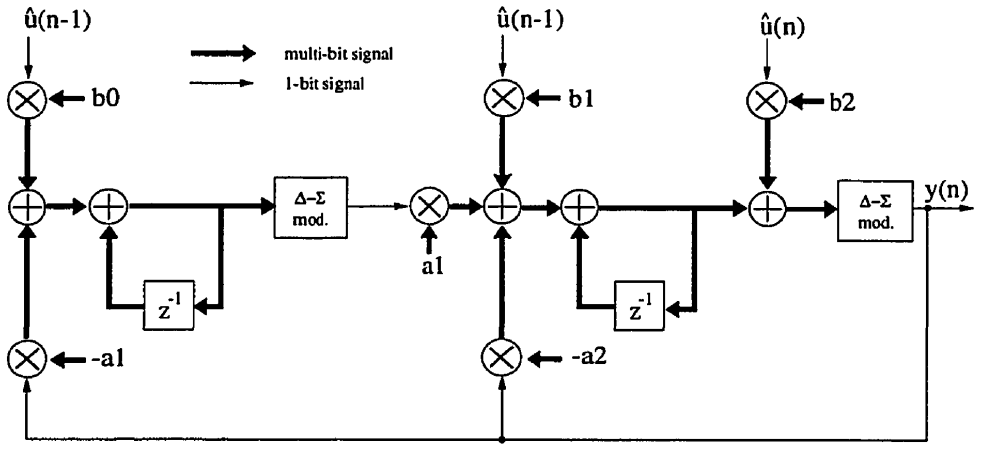


FIGURE 2.6 Δ–Σ based biquad filter with input summing

The transfer function of the Δ–Σ based biquad filter shown in Figure 2.6 is given in EQ 2.8 [11].

$$T(z) = \frac{b_2 z^2 + (b_1 - 2b_2)z + (b_0 a_1 - b_1 + b_2)}{z^2 - (2 - a_2)z + (1 + a_1^2 - a_2)} \qquad \text{(EQ 2.8)}$$

Equating EQ 2.7 and EQ 2.8, the relationships between two sets of coefficients are found [11].

$$a_2 = p_1 + 2 \qquad \text{(EQ 2.9)}$$

$$a_1 = (p_0 + p_1 + 1)^{1/2} \qquad \text{(EQ 2.10)}$$

$$b_2 = n_2 \qquad \text{(EQ 2.11)}$$

$$b_1 = n_1 + 2n_2 \qquad \text{(EQ 2.12)}$$

$$b_0 = (n_0 + n_1 + n_2)/a_1 \qquad \text{(EQ 2.13)}$$

The filter characteristics of the biquad filter are defined by the coefficients (a_0, a_1 and b_{0-2}). If registers are used to store these filter coefficients, the filter characteristics can be programmable. As seen in Figure 2.6, Δ–Σ modulators are used internally to convert the

design of the biquad filter will be addressed in detail.

2.2.2 Accuracy of the filter

The filter uses the fixed-point two's complement number system, with 28 bit numbers containing 4 integer bits and 24 bits in fraction.

Since the fixed-point number system is used, the accumulators in the modulators may suffer from overflow if the number of bits to store the state values are not sufficient. Overflow will significantly degrade the performance of the modulator and the filter. Simulation results given by [14] show that four bits in the integer part are enough to present overflow for input signal magnitude less than 0.6 and is suitable for the 4-bit partial carry-propagation used in the modulator. The biquad filter of this thesis uses four bits in the integer part. The details of the 4-bit partial carry-propagation in the modulator will be discussed in Chapter 3.

The requirement of the number of fraction bits in the filter system depends on the degree of accuracy needed for the filter. Usually, if the oversampling ratio increases, the values of the filter coefficients will tend to decrease [1], [11], [17]. Thus, a high speed filter which can handle high oversampling rate requires a large number of bits in the fractions to represent small coefficients accurately. The effect on lack of fraction bits was shown by the simulations done by Reference [1]. For a specific oversampling rate, the characteristics of the filter were simulated by the logic simulator Tortle [15] with different accuracy in the fraction. This implementation of the biquad filter uses twenty four bits in the fraction to ensure high precision in the filter system.

Logic

TSPC logic [22] was selected for the implementation of the biquad filter because of the following reasons:

- the dynamic logic improves clock rate so high speed can be attained.
- high throughput can be obtained because pipelining can be done at the complex logic gate level.
- only one phase of the system clock is used throughout the circuit so clock skew problems are reduced.

In a TSPC pipelined system shown in Figure 2.7, the N-block and the P-block are placed alternately because the two types of blocks evaluate their logic functions and hold their output values in opposite logic values of the clock signal. This alternating placement of N-block and P-block keeps the pipelined system race-free.

The TSPC logic works in the following way. When $clk = 1$, the N-block is in the evaluation state. Transistor B is *on*, transistor A is *off* and the logic function f_n is evaluated. Since the inputs to f_n are from P-blocks, their logic values will not change when $clk = 1$. In addition, transistor D is *on* so the output stage acts as an inverter made up of PMOS transistor C and NMOS transistor E. Hence, the output, out_n, will be the negation of x_n. Therefore, if the logic function is *true*, the node x_n will be pulled down to 0 and out_n will be 1. Otherwise, x_n will stay its precharged value 1 and out_n will be 0.

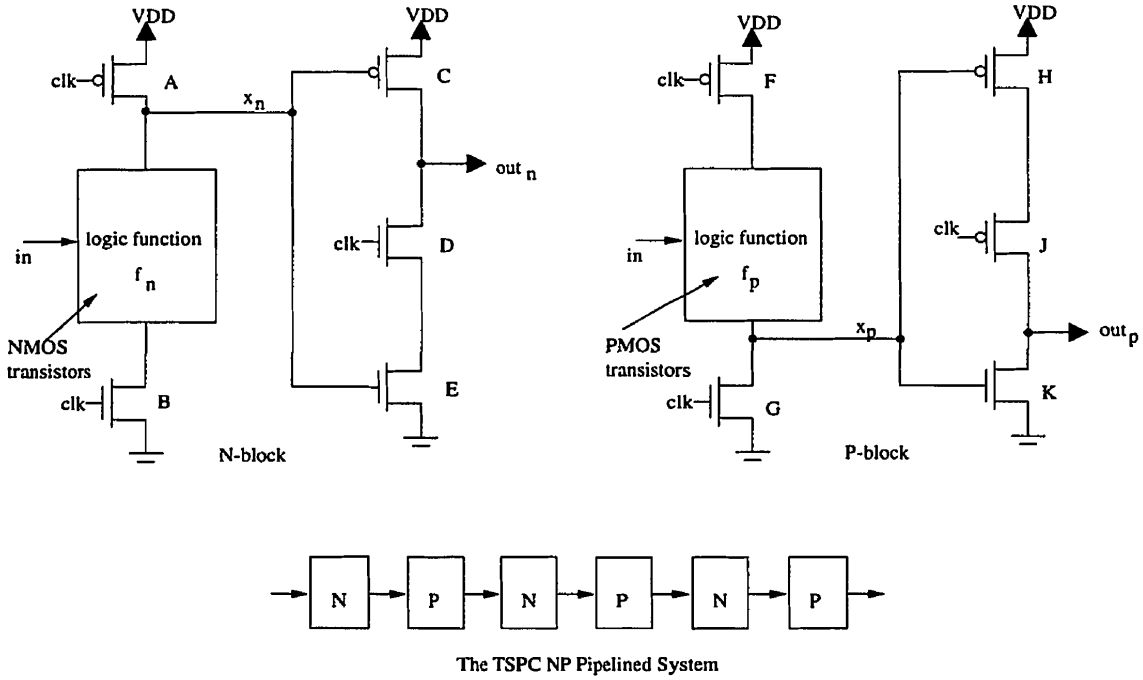


FIGURE 2.7 Basic building blocks of TSPC logic.

However, when $clk = 0$, the N-block is in the pre-charge state. Transistor B is *off* and transistor A is *on* so the node x_n is pre-charged to 1. Since transistor C and transistor D are *off*, the output node out_n is isolated and keeps its original logic value.

Similarly, the P-block is the dual of the N-block. In summary, when $clk = 1$, the N-block evaluates, and the P-block holds its output and pre-charges the internal node. When $clk = 0$, the N-block holds its output and precharges its internal node, and the P-block evaluates.

Even though two logic functions can be evaluated in one clock cycle, usually only the N-block is used to implement complex logic functions. This is because the PMOS transistors in the P-block have slow charge-up time. Thus, complex logic functions are usually realized in the N-blocks only, because of their fast pull-down ability. The implementation of the biquad filter uses the P-blocks as simple logic functions, latches or buffers.

2.3.1 All-N TSPC dynamic logic

Improving the speed of the P-block of the TSPC logic allows more complex logic function to be built into the P-block than that of a traditional TSPC P-block. In reference [7], an all-N TSPC dynamic logic was introduced. This type of TSPC logic is compatible with the traditional TSPC logic. The all-N TSPC logic also has two types of basic blocks such as N-block and P-block.. Both basic blocks use NMOS transistors only to realize the logic function. Therefore, the P-block of all-N TSPC logic will be faster than that of the traditional TSPC logic.

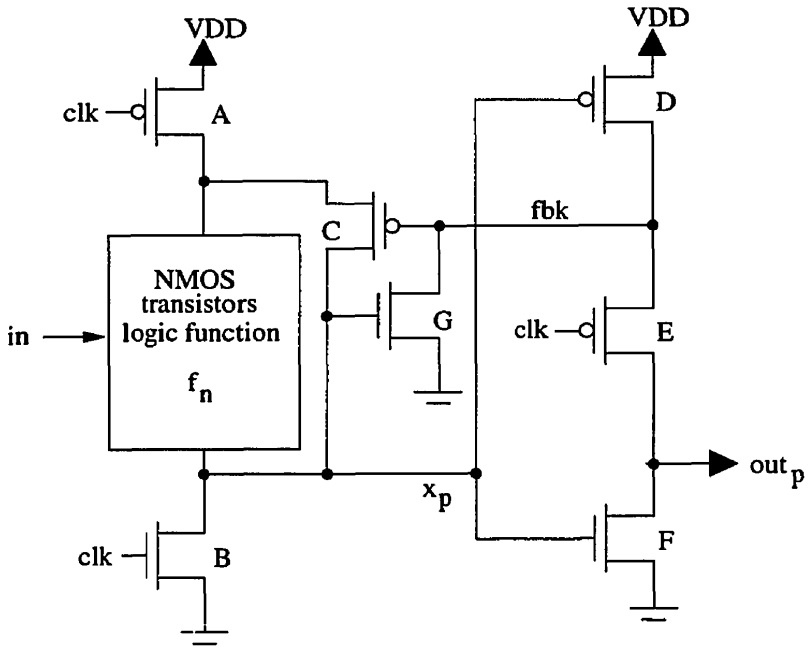


FIGURE 2.8 P-block of all-N TSPC dynamic logic

The P-block of all-N TSPC logic shown in Figure 2.8 works in the following way. When $clk = 1$, the P-block is in the pre-charge stage. Transistors A and E are *off*, but transistor B is *on*. Therefore, transistor B precharges the node x_p to 0. In addition, transistors F and E are *off* so the output out_p is isolated and keeps its old value. Since transistor D is *on* due to the pre-charged x_p, the node *fbk* is then pulled up to 1 so transistor C is *off*. Transistor G is also *off* due to the pre-charged x_p. As a result, the internal node x_p is pre-charged to 0 and the output value out_p remains the same.

and transistors A and E are *on*. PMOS transistor D and NMOS transistor F act as an inverter as E is *on*. The output out_p is the negation of x_p. Since transistor A is *on*, if the logic function f_n is *true*, the node x_p will be pulled up. This slight pull-up triggers the positive feedback from transistors C and G. The feedback works as follows. When x_p is being pulled up, transistor G is *on*. This turns on transistor C, resulting in pulling the node x_p to 1, if necessary. If f_n is *false*, x_p will remain 0 so the positive feedback is *off* and the out_p becomes the negation of x_p. i.e. 1. Since the output of the P-block all-N TSPC logic is inverting, a static inverter can be placed at the output to correct this.

For traditional TSPC logic, the clock rate of the system depends on the slow P-block. Therefore, the slow clock rate degrades the performance of the fast N-block. However, if the all-N TSPC P-block is used in the system instead of the traditional TSPC P-block, the system clock rate will not be affected. The speed gain in the all-N TSPC P-block is due to the fast NMOS transistors used in realizing the logic function.

However, the all-N TSPC P-block requires more transistors than the traditional TSPC P-block to implement a logic function. Therefore, the traditional TSPC P-block is used to implement simple logic functions and buffers. The all-N TSPC P-block is only used to help the N-block to implement complex logic functions where the N-block alone cannot give a high speed realization. The speed difference between the N-block of traditional TSPC logic and that of all-N TSPC logic is minimal. To implement the same logic function, the N-block of all-N TSPC logic requires more transistors than the traditional TSPC one. Therefore, the biquad filter does not use the N-block of all-N TSPC logic. The implementation of the filter uses the N-block of traditional TSPC logic and P-blocks of both types of TSPC logic.

CHAPTER 3 *Delta-sigma Based Biquad Filter Design*

This chapter presents the architecture and the logic-gate-level design of an area-efficient high-speed biquad filter which uses a single pipelined ALU. Before the discussion of the filter, the architecture and design of a second order Δ–Σ modulator using carry-save-adders (CSAs) is described in section 3.1. The architecture of the biquad filter is described in section 3.2. Finally, the logic-gate-level design of the biquad filter is presented in section 3.3.

3.1 Second Order Δ–Σ Modulator Realization Using CSAs

A second order Δ–Σ modulator is used in the biquad filter to convert 28-bit internal signals to 1-bit quantized signals. To understand how to realize the biquad filter using CSAs, the realization of the second order Δ–Σ modulator using CSAs is described first. This is followed by a discussion of the quantizer using 4-bit partial carry-propagation. At the end, a complete realization of the second order Δ–Σ modulator using CSAs is presented.

Figure 2.4.

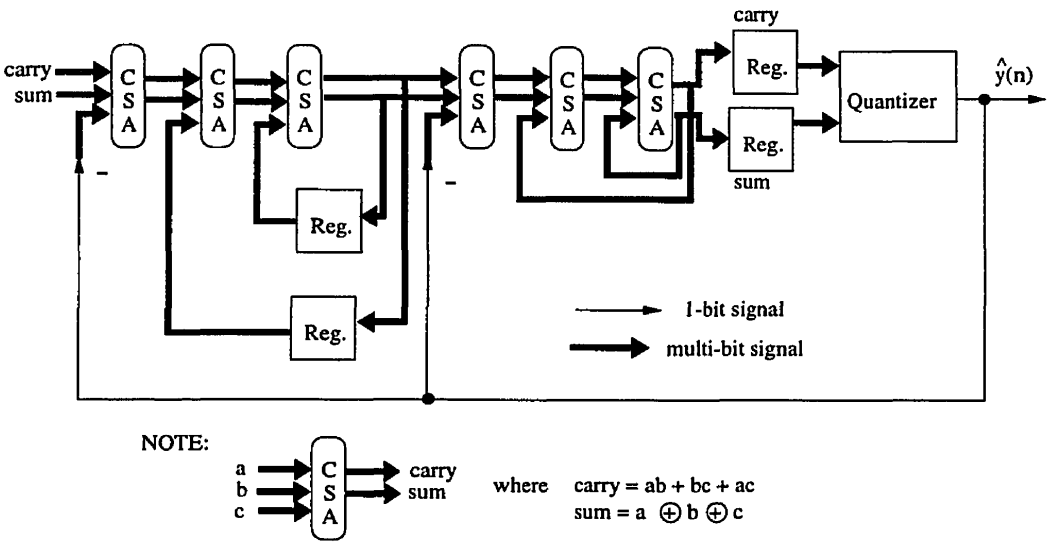


FIGURE 3.1. Second order Δ–Σ modulator realization using CSAs

Since the CSAs produce *sum* and *carry*, four registers are needed to store the states of the modulator. Since using CSAs eliminates the delay due to carry-propagation, this modulator is faster than the modulator using traditional carry-propagation adders (CPAs). However, the problem now appears in the realization of the quantizer because the quantizer needs carry-propagation. The following section presents a solution to this problem.

3.1.1 Quantization and carry-propagation

As mentioned in Chapter 2, a two-level quantizer is used in the modulator. The output of the quantizer is 1 for $Y \geq 0$, or 0 for $Y < 0$. Hence the output of the quantizer is actually the inversion of the most significant bit (MSB) or the sign bit of the output Y. However, the output of the second accumulator of the modulator contains two parts - the carry and the intermediate sum due to the CSA operation. If a full carry-propagation adder is used to do the final summing, the modulator will suffer from a long carry-propagation delay. The same design problem was faced by [14], and a 4-bit partial carry-propagation quantization scheme was used in the filter designs. Figure 3.2 shows a 4-bit section of a 2-input pipelined accumulator

inputs and propagates the carry across a 4-bit section. This 4-bit section can be stacked to arbitrary width. As seen in Figure 3.2, there are five pipeline stages which include one for the CSA and four for the carry-propagation [14]. The accuracy of the accumulator is not affected because the sum and the carry are fed back for accumulation. However, the output is only an approximate representation of the accumulator's state because of the hidden carries.

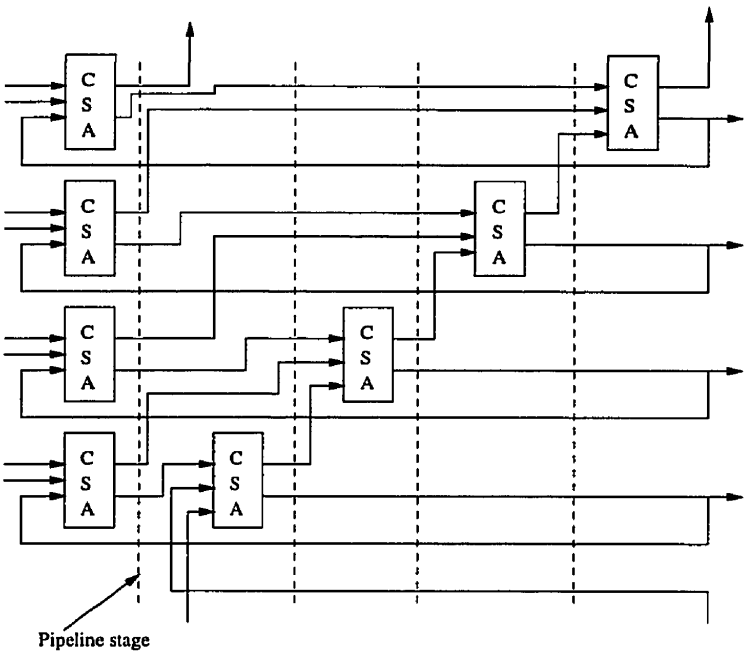


FIGURE 3.2. 2-input accumulator with 4-bit carry propagate [14]

According to [14], the 4-bit partial carry-propagation introduces additional noise into the modulator due to the hidden carry. Simulations in [14] show that for a modulator using 4-bit partial carry-propagation, the signal-to-noise ratio (SNR) drops by 2 dB compared to a modulator using full carry propagation. The modulator using full carry propagation has a maximum SNR of 85.9 dB [14]. The 4-bit partial carry-propagation method is desirable in the filter implementation because the delay due to partial carry-propagation is less than that of full carry propagation. Therefore, the quantizer of the modulator in this thesis uses 4-bit partial carry-propagation.

Figure 3.3 shows the use of CSAs to implement a second order Δ–Σ modulator with the quantizer using 4-bit partial carry propagation. Even though 4-bit partial propagation is used, the accuracy of the modulator is not affected because both 28-bit *sum* and *carry* are fed back for accumulation. Note that the output of the quantizer is the sign bit (or MSB) of the output sum of the CSA. The modulator in Figure 3.3 was not actually implemented as a single entity but it illustrates the idea of using CSAs as building block. This idea is important when the architecture of the biquad filter is discussed in section 3.2.

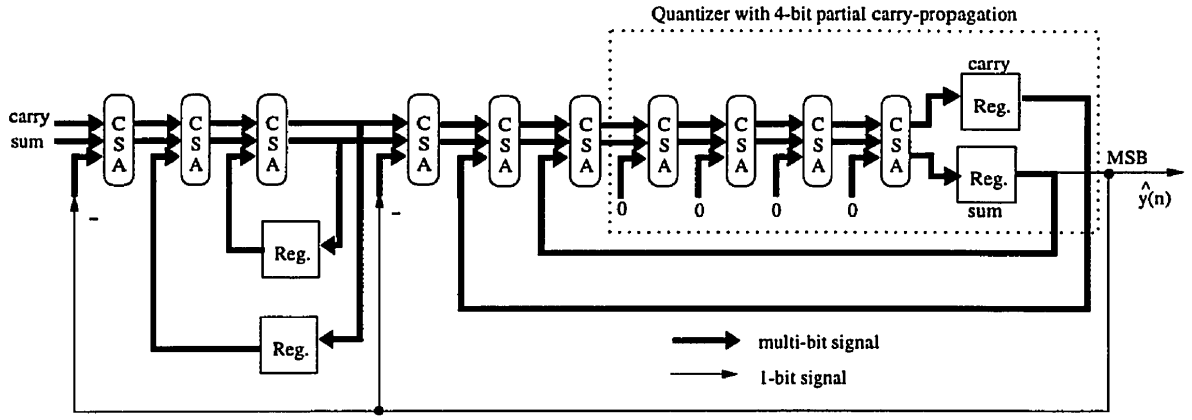


FIGURE 3.3. Method showing CSA realization of the second order Δ–Σ modulator with 4-bit partial carry-propagation quantizer

3.2 Δ–Σ Based Biquad Filter Realization Using CSAs

The architecture of the filter contains an ALU, registers and a control unit. The control unit issues a series of instructions to the ALU and the registers to perform the filter arithmetic. Figure 3.4 shows the block diagram of the filter architecture.

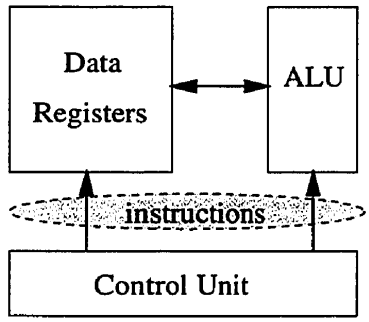
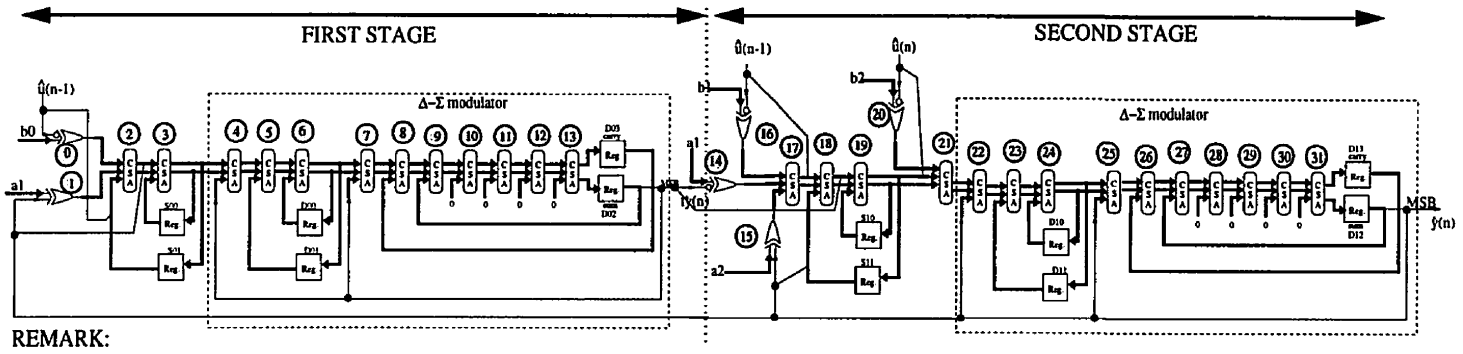


FIGURE 3.4. Architecture of the biquad filter using the single ALU approach

A Δ–Σ based biquad filter can be realized in a similar manner as the second order Δ–Σ modulator described in section 3.1. Figure 3.5 shows a complete realization of the Δ–Σ based biquad filter of Figure 2.6 using CSAs and XOR gates. To implement the single ALU filter architecture, the data-flow graph and the data dependencies of the biquad filter shown in Figure 3.5 have to be known. The minimum number of data registers required in the new architecture, and the filter instructions can be obtained from the data-flow information. Then, the biquad filter can be realized by the new architecture with the filter instructions.

Before the description of the data dependencies, the 1×28-bit multiplication in the filter is first discussed in 3.2.1. Possible pipeline hazards are discussed in section 3.2.2. Section 3.2.3 presents the data-flow graph of the biquad filter. The filter instructions are described in section 3.2.4. The detailed filter architecture using the new architecture is described in section 3.2.5.



REMARK:

a1, a2, b0, b1 and b2 are filter coefficients.

ty(n) is the quantizer output of the first modulator

$\hat{y}(n)$ is the quantizer output of the second modulator. It is also the filter output.

$\hat{u}(n)$ is the 1-bit filter data input and $\hat{u}(n-1)$ is the delayed version of $\hat{u}(n)$.

ty(n), y(n), u(n), and u(n-1) are 1-bit signals.

S00, S01, D00, D01, D02, D03, S10, S11, D10, D11, D12, D13 are registers storing the state of the filter

1-bit value "u" is fed into the carry-in of the 28-bit "carry" where the LSB is always 0.

→ multi-bit signal

— 1-bit signal

FIGURE 3.5. Δ–Σ based biquad filter realization using XOR and CSA

The 28-bit coefficients of the Δ–Σ based biquad filter are multiplied by the 1-bit signals. Since the 1-bit signals have the values of either +1 or -1, the results of multiplication by x are either x or -x. This is the conditional two's complementation of the coefficients by a 1-bit signal. Therefore, in the CSA design of the filter shown in Figure 3.5, a conditional two's complementation replaces the 1 × 28-bit multiplication. The conditional two's complementation can be implemented by XORing the coefficient with the 1-bit signal, and using the 1-bit signal as the LSB of related CSA operations. EQ 3.1 shows an example of the 1×28 bit multiplication replaced by the conditional two's complementation. Note that $y(n)$ is the quantizer output of the second stage and a_1 is the coefficient. The 1-bit signal $y(n)$ is replicated 28 times before XORing with a_1 bitwise. Then, the 1-bit value of $y(n)$ is added to the XORed result.

$$y(n) \times a_1 = (a_1 \oplus (28 * y(n)[0])) + y(n)[0] \qquad \text{(EQ 3.1)}$$

where $28 * y(n)[0]$ means that the 1-bit signal $y(n)$ is replicated 28 times.

3.2.2 Pipeline hazard

Figure 3.5 shows a method for realizing the Δ–Σ based biquad filter using CSAs and XOR gates only. From this realization method, an ALU which is able to handle the CSA function and the XOR function only is sufficient to realize the filter. The detailed design of the ALU is described in section 3.3.1. For now, assume that the high speed pipelined ALU takes two clock cycles to process the inputs, and has four output ports. The four ALU outputs are responsible for storing two pairs of sum and carry results of the CSA operations. One pair of sum and carry is produced by the ALU processing the even number clock cycle input, and the other pair is produced by the ALU processing the odd number clock cycle input.

implies that two logically consecutive CSAs cannot be handled by the pipelined ALU in consecutive clock cycles because there is a read-after-write pipeline hazard [8]. For example, in Figure 3.6, the CSAs 4 and 5 of the filter in Figure 3.5 cannot be handled by the ALU in consecutive clock cycles. Figure 3.6 illustrates the pipeline hazard encountered if the pipelined ALU is used to implement the filter directly.

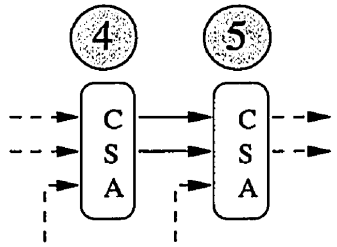


FIGURE 3.6. Pipeline hazards in the ALU

From Figure 3.6, the output of CSA 4 will be ready in two clock cycles. However, the CSA 5 needs CSA 4's output in the next clock cycle. Therefore, a read-after-write pipeline hazard occurs in the ALU.

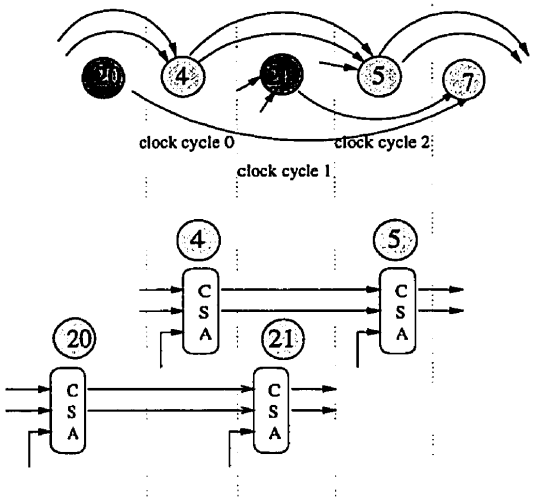


FIGURE 3.7. Solution to the pipeline hazard

observation of Figure 3.5, the data dependencies between the CSA and XOR operations of the first stage and those of the second stage are longer than two clock cycles. The easiest method to interleave the CSA and XOR operations of the filter is to schedule the ALU to handle these operations, alternating between two stages in alternate cycles. For example, the data dependency between the CSA pairs, CSA 4 and 5, and, CSA 21 and 22, is longer than two clock cycles. Therefore, the ALU can be sequenced to handle these operations in the following sequence: CSA 21, 4, 22 and then 5.

3.2.3 Data-flow graph of the Δ–Σ based biquad filter

With the interleaving solution to solve the pipeline hazard, the sequence of execution of the ALU instructions alternates between those instructions in the first stage and those in the second stage. As seen in Figure 3.5, there are fourteen operations in the first stage, and eighteen operations in the second stage. To keep the operation of the ALU hazard-free, four NO-OP instructions are inserted into the filter instructions of the first stage. Therefore, a total of thirty-six ALU instructions are needed including four NO-OPs to realize the filter. The presence of NO-OP instructions in the filter realization is due to the unbalanced number of ALU instructions in two stages. From Figure 3.5, the data-flow graph of the Δ–Σ based biquad filter can be constructed. With this data-flow graph, the ALU instructions can be generated.

Figure 3.8 shows the data-flow graph and the data dependencies of the Δ–Σ based biquad filter of Figure 3.5. Note that the ALU has four output ports - ALUS0, ALUC0, ALUS1 and ALUC1. Ports ALUS0 and ALUC0 store the valid ALU outputs for the even numbered ALU instructions. Ports ALUS1 and ALUC1 store the ALU outputs for the odd numbered ALU instructions. During the XOR operations, the results are saved into either ALUS0 or ALUS0, and the values in ALUC0 and ALUC1 are ignored. The data-flow graph

instructions and the odd numbered filter instructions.

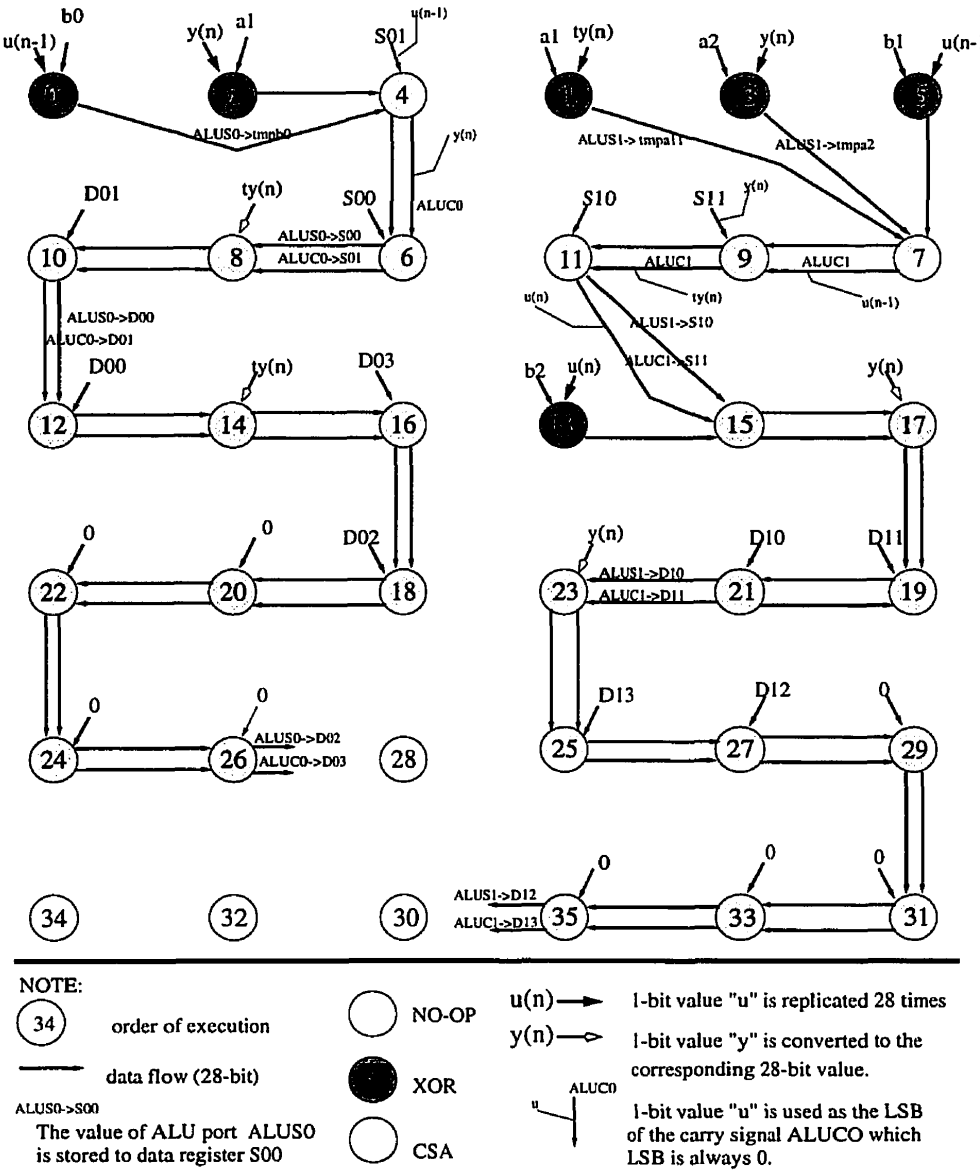


FIGURE 3.8. Data flow graph and data dependences of the Δ–Σ based biquad filter

The data-flow graph is somewhat hard to read without explanation. The following two examples will clear the confusion.

previous 1-bit filter input *u(n-1)* to produce an XORed result. The result is stored in *ALUS0*. Since this result will be used by instruction 4, it has to be stored into a temporary data register *tmpb0* first.

Consider instruction 25 and instruction 27. They have two clock cycles of data dependency. In instruction 25, the ALU reads the sum and carry results (*ALUS1* and *ALUC1*) of instruction 23 and the value of the data register *D13*. Then, the ALU results (the new *ALUS1* and *ALUC1*) of instruction 25 can be used directly by instruction 27 because the pipeline latency of the ALU is two clock cycles.

3.2.4 Information extracted from the data-flow graph

The data-flow graph in Figure 3.8 gives valuable information regarding the design of the biquad filter. The information includes the number of ALU instructions, data flow, the number of data registers required, handling of quantization, handling of XOR instructions, and the LSB handling.

The ALU instructions will be discussed in 3.2.5. The following sub-sections address the issues of the requirement of the number of data registers, the handling of the quantization outputs, XOR operations and the LSB handling.

3.2.4.1 Requirement of the number of data registers

The data-flow diagram shows that in addition to the twelve filter state registers (*S00, S01, D00, D01, D02, D03, S10, S11, D10, D11, D12*, and *D13*), three more temporary data registers (*tmpb0, tmpa11*, and *tmpa2*) are needed to store the temporary results of the XOR instructions 0, 1 and 3. Moreover, five data registers are needed to store five filter coefficients. Therefore, the filter architecture requires a total of twenty data registers.

In CSA instructions 8, 14, 17 and 23, the outputs *ty(n)* and *y(n)* of both quantizers are fed back to the ALU. Since the outputs are 1-bit signals, they have to be converted to their 28-bit fixed-point 2's complement representation before being used in the ALU.

3.2.4.3 XOR operations

In all XOR instructions, the 1-bit signals are replicated 28 times before being XORed with the 28-bit coefficients bitwise. The 1-bit signals are the filter input, *u(n)*, the previous filter input, *u(n-1)*, and the quantizers' outputs, *ty(n)* and *y(n)*.

3.2.4.4 LSB handling

For instructions 4, 6, 9, 11, 15, the LSBs of the carry port of the ALU are appended with the 1-bit signals. The LSB handling is needed because the biquad filter has 1×28 bit multiplications which have been replaced by conditional two's complementation. The XOR instructions together with the LSB handling perform the conditional two's complementation. The detailed design will be addressed in section 3.3.

3.2.5 Biquad filter instructions

The CSA and XOR realization of the Δ–Σ based biquad filter in Figure 3.5 and its data-flow graph in Figure 3.8 shows that thirty-six filter instructions are required to process each filter input. Table 3.1 summarizes and explains the functions of these thirty-six ALU instructions. Of these thirty-six ALU instructions, some of them are identical. Therefore, instruction cycles that use the same filter instruction will have the same instruction number but different cycle numbers. There are only twenty-seven distinct filter instructions in the thirty-six instruction cycles.

Cycle #	Instruction #	Functions
0	0	tmpb0 ← b0 ⊕ !u(n-1)[0] * 28
1	1	tmpa11 ← a1 ⊕ y(n)[0] * 28
2	2	ALUS0 ← a1 ⊕ y(n)[0] * 28
3	3	tmpa2 ← a2 ⊕ ty(n)[0] * 28
4	4	ALUC0 ← ALUS0 + tmpb0 + { S01[27:1], !u(n-1)[0]}[a] ALUS0 ← ALUS0 ⊕ tmpb0 ⊕ { S01[27:1], !u(n-1)[0]}
5	5	ALUS1 ← b1 ⊕ !u(n-1)[0] * 28
6	6	*S01 ← ALUS0 + S00 + { ALUC0[27:1], y(n)[0] }* S00 ← ALUS0 ⊕ S00 ⊕ { ALUC0[27:1], y(n)[0] }
7	7	ALUC1 ← ALUS1 + tmpa11 + tmpa2; ALUS1 ← ALUS1 ⊕ tmpa11 ⊕ tmpa2
8	8	ALUC0 ← ALUC0 + ALUS0 + [ty(n)][b] ; ALUS0 ← ALUC0 ⊕ ALUS0 ⊕ [ty(n)]
9	9	ALUC1 ← ALUS1 + { ALUC1[27:1], !u(n-1)[0]} + { S11[27:1], !ty(n)[0] } ALUS1 ← ALUS1 ⊕ { ALUC1[27:1], !u(n-1)[0]} ⊕ { S11[27:1], !ty(n)[0] }
10	10	ALUC0 ← ALUC0 + ALUS0 + D01; ALUS0 ← ALUC0 ⊕ ALUS0 ⊕ D01
11	11	S11 ← ALUS1 + { ALUC1[27:1], !y(n)[0]} + S10 S10 ← ALUS1 ⊕ { ALUC1[27:1], !y(n)[0]} ⊕ S10
12	12	D01 ← ALUC0 + ALUS0 + D00; D00 ← ALUC0 ⊕ ALUS0 ⊕ D00
13	13	ALUS1 ← b2 ⊕ u(n)[0] * 28
14	8	ALUC0 ← ALUC0 + ALUS0 + [ty(n)]; ALUS0 ← ALUC0 ⊕ ALUS0 ⊕ [ty(n)]
15	14	ALUC1 ← ALUS1 + S10 + { S11[27:1], !u(n)[0] } ALUS1 ← ALUS1 ⊕ S10 ⊕ { S11[27:1], !u(n)[0] }
16	15	ALUC0 ← ALUC0 + ALUS0 + D03; ALUS0 ← ALUC0 ⊕ ALUS0 ⊕ D03
17	16	ALUC1 ← ALUC1 + ALUS1 + [y(n)]; ALUS1 ← ALUC1 ⊕ ALUS1 ⊕ [y(n)]
18	17	ALUC0 ← ALUC0 + ALUS0 + D02; ALUS0 ← ALUC0 ⊕ ALUS0 ⊕ D02
19	18	ALUC1 ← ALUC1 + ALUS1 + D11; ALUS1 ← ALUC1 ⊕ ALUS1 ⊕ D11
20	19	ALUC0 ← ALUC0 + ALUS0 + #0; ALUS0 ← ALUC0 ⊕ ALUS0 ⊕ #0
21	20	D11 ← ALUC1 + ALUS1 + D10; D10 ← ALUC1 ⊕ ALUS1 ⊕ D10
22	19	ALUC0 ← ALUC0 + ALUS0 + #0; ALUS0 ← ALUC0 ⊕ ALUS0 ⊕ #0
23	16	ALUC1 ← ALUC1 + ALUS1 + [y(n)]; ALUS1 ← ALUC1 ⊕ ALUS1 ⊕ [y(n)]
24	19	ALUC0 ← ALUC0 + ALUS0 + #0; ALUS0 ← ALUC0 ⊕ ALUS0 ⊕ #0
25	21	ALUC1 ← ALUC1 + ALUS1 + D13; ALUS1 ← ALUC1 ⊕ ALUS1 ⊕ D13
26	22	D03 ← ALUC0 + ALUS0 + #0; D02 ← ALUC0 ⊕ ALUS0 ⊕ #0 ty(n) ← MSB(D02)
27	23	ALUC1 ← ALUC1 + ALUS1 + D12; ALUS1 ← ALUC1 ⊕ ALUS1 ⊕ D12
28	24	NO-OP
29	25	ALUC1 ← ALUC1 + ALUS1 + #0; ALUS1 ← ALUC1 ⊕ ALUS1 ⊕ #0
30	24	NO-OP

Cycle #	Instruction #	Functions
31	25	ALUC1 ← ALUC1 + ALUS1 + #0; ALUS1 ← ALUC1 ⊕ ALUS1 ⊕ #0
32	24	NO-OP
33	25	ALUC1 ← ALUC1 + ALUS1 + #0; ALUS1 ← ALUC1 ⊕ ALUS1 ⊕ #0
33	24	NO-OP
35	26	D13 ← ALUC1 + ALUS1 + #0; D12 ← ALUC1 ⊕ ALUS1 ⊕ #0 y(n) ← MSB(D12)

a. { S01[27:1], !u(n-1)[0]} means that the 1-bit value of *!u(n-1)* is used as the LSB of the *S01*. Since LSB of *S01* and other carry values are always 0, their LSB can be used in the conditional 2's complementation.

b. [y(n)] means 1-bit value of y(n) is converted to its corresponding 28-bit fixed point 2's complement representation.

TABLE 3.1. Filter instruction cycles and instruction information

3.2.6 Single ALU architecture of the Δ–Σ based biquad filter

Figure 3.9 shows the detailed architecture of the filter. The filter core has twenty data registers and an ALU which can handle XOR and CSA functions. The control unit containing all thirty-six instructions sends the control signals to the ALU and the data registers. Moreover, the 1-bit signal handler block is needed to handle all 1-bit signals involved in the quantization and the XOR instructions.

Note that three rows of registers are used because the ALU has three input ports. The registers of the top two rows store the information of the filter states, and the temporary XOR results. The filter coefficients are arranged into the lowest row to faciliate the loading of the coefficients from off-chip. The following sub-sections discuss the details of the architecture shown in Figure 3.9. The detailed logic-gate-level design of the architecture is described in the Section 3.3.

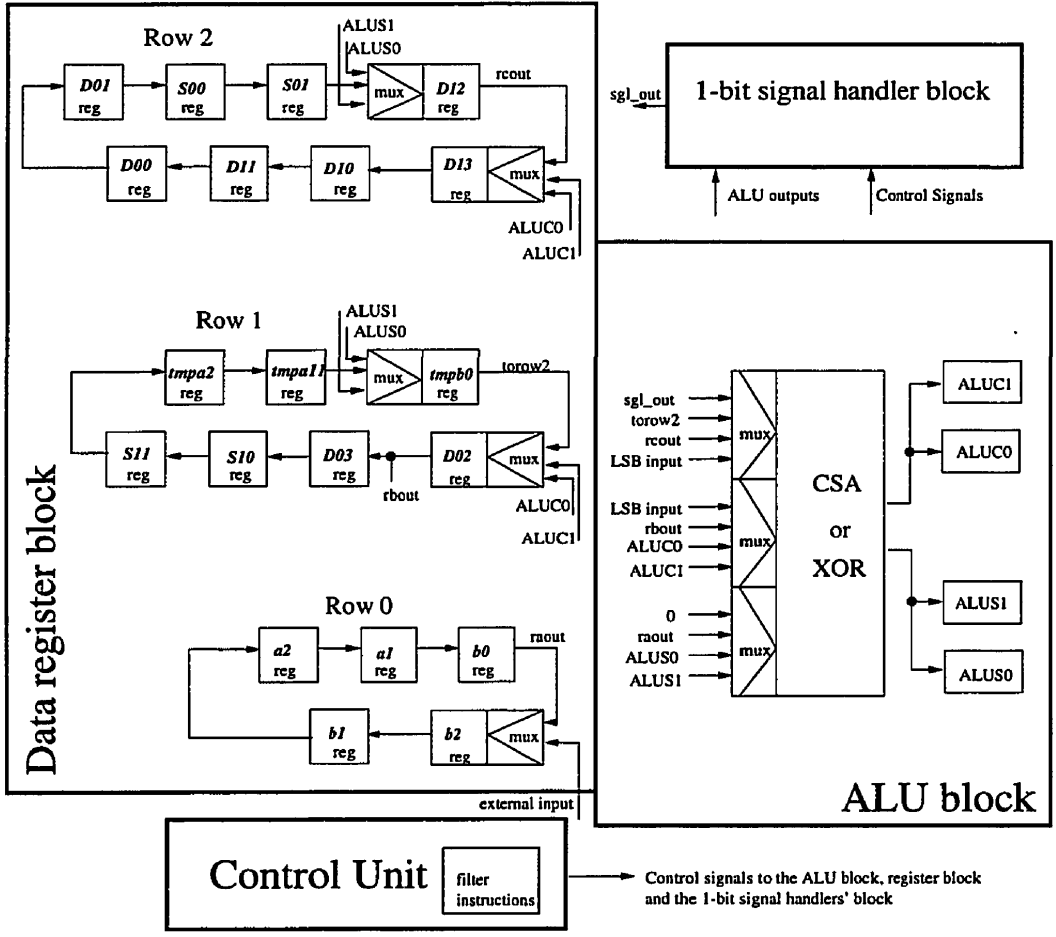


FIGURE 3.9. Architectural block diagram of the biquad filter using a single ALU approach

3.2.6.1 Arrangement of data registers

Arranging the data registers in their order of appearance to the ALU ports will minimize the number of temporary registers and the circuit complexity. The data-flow graph and the filter instructions indicate the order of appearance of all registers. For example, coefficient *b0* is used in instruction 0 while *S01* is used in instructions 4 and 6. With the order of appearance information, the data registers can be arranged into three rows (in loops) as shown in Figure 3.9 to allow efficient loading and storing of data to and from the ALU. Registers are arranged in loops so that the information in them will be stored within the loops

the data registers is shown in Figure 3.9.

Only one multiplexer, attached to data register *b2*, is needed for external loading of coefficients because all other coefficients can be loaded into the data registers serially through the external input port of *b2*. This is an advantage of arranging the row of registers in a ring. The coefficient values are preserved in the data registers until reset.

3.2.6.2 Multiplexers in the ALU block and the data registers block

Figure 3.9 addresses the use of multiplexers between the ALU and three rows of registers. The input ports of the ALU and the output ports of data registers of the top two rows use multiplexers because they enable these input/output interfaces to select the correct data to load or store. For example, the output data register labelled *D12* may store the contents of *ALUS0* or *ALUS1* depending on the instructions. The top multiplexer of the ALU has the ability to select one of the following signals:

- the converted 28-bit values of 1-bit quantizer outputs for instructions 8, 14, 17 and 23,
- a value of zero for instructions doing 4-bit partial carry propagation, and,
- the outputs of the data registers of the top two rows depending on the filter instructions.

3.3 Design of the Δ–Σ Based Biquad Filter Using the Single ALU Approach

This section presents the detailed logic-gate-level implementation of the biquad filter architecture shown in Figure 3.9 of section 3.2.6. The ALU block is discussed in section 3.3.1. The data register block is described in section 3.3.2. The handling of the 1-bit signals is

Finally the simulation results of the design of the biquad filter using the single ALU approach are presented at the end.

3.3.1 ALU block

The ALU performs CSA and XOR functions required by the biquad filter. Therefore, the ALU has two modes of operations. Figure 3.10 shows the design of a 1-bit ALU block. A selector signal, *mode*, is used to set the ALU in the proper mode of operation. In general, for each ALU bit, the *sum* and *carry* are generated by EQ 3.2 and EQ 3.3.

$$carry = AB + BC + AC \qquad \text{(EQ 3.2)}$$

$$sum = A \oplus [\ mode \bullet v\ + !mode\ (B \oplus C)] \qquad \text{(EQ 3.3)}$$

If the CSA mode is on (*mode* = 0), the ALU accepts three selected 28-bit inputs (*A*, *B*, and *C*) through the input multiplexers from other blocks. Then the ALU performs the CSA function on them. The multiplexers are controlled by *as*, *bs* and *cs* which are provided from the control unit. The resulting *sum* and *carry* values are sent to the output ports *ALUS0, ALUC0, ALUS1* and *ALUC1* depending the value of *odd* which originates from the control unit. When *odd* is 0, the ALU processes the even numbered instructions. When *odd* is 1, the ALU handles the odd numbered instructions.

If the ALU performs an XOR function (*mode* = 1), only the value of output *A* and the value of *v* are used. The value *v* is a 1-bit signal sent from the 1-bit signal handler block. It is used by twenty-eight 1-bit ALU blocks. Therefore *A* is XORed with *v* bitwise. Note that *A* will have the value of one of the coefficients stored in row 0 of the data register block. The value *v* is selected by the 1-bit signal handler block from the signals, *u(n-1), u(n), ty(n)* and *y(n)*.

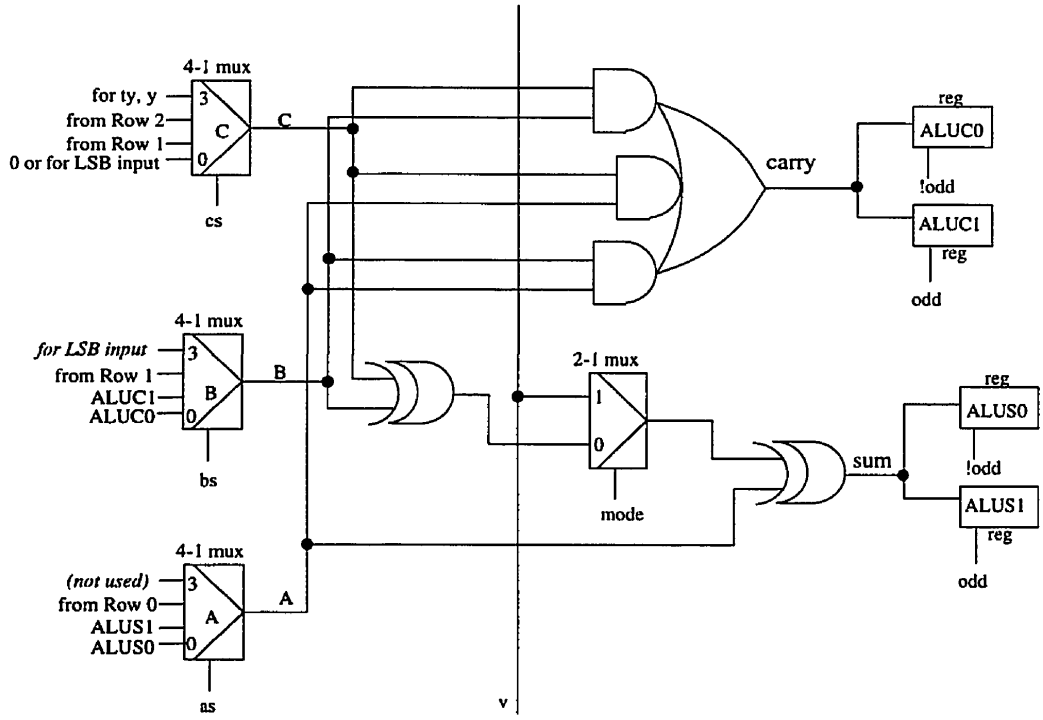


FIGURE 3.10. 1-bit ALU block

Note that port 3 of Mux A is not used. Port 3 of Mux B is used only in the LSB of the ALU for LSB appending. Port 3 of Mux C is reserved for accepting 28-bit versions of *ty(n)* and *y(n)*. Port 0 of Mux C is always zero for ALU bit 1 to ALU bit 27. For ALU bit 0, port 0 of Mux C accepts input from the LSB handlers. Therefore, during the 4-bit carry-propagation, the output *C* of Mux C is set to 0 so that the ALU performs carry-propagation with ALUS0, ALUS1, ALUC0 and ALUC0 being fed back from Mux A and Mux B.

3.3.1.1 Pipelining the ALU

One of the longest critical paths is highlighted in Figure 3.11. The time delay for this path is long, limiting the performance of the ALU. Other logic blocks such as the data register block, 1-bit signal handler block and the control unit have shorter critical paths, and will suffer from the slow clock as well.

and throughput can be obtained. To pipeline the ALU, latches are inserted into the ALU so that the ALU will take two clock cycles to process each set of inputs. An effort was made to determine the best position for inserting latches so that the ALU logic is divided into two parts with nearly equal delay. Figure 3.11 shows the critical path and the insertion point of the pipeline latches. Since using this pipeline approach the clock rate is higher, the system will have better performance.

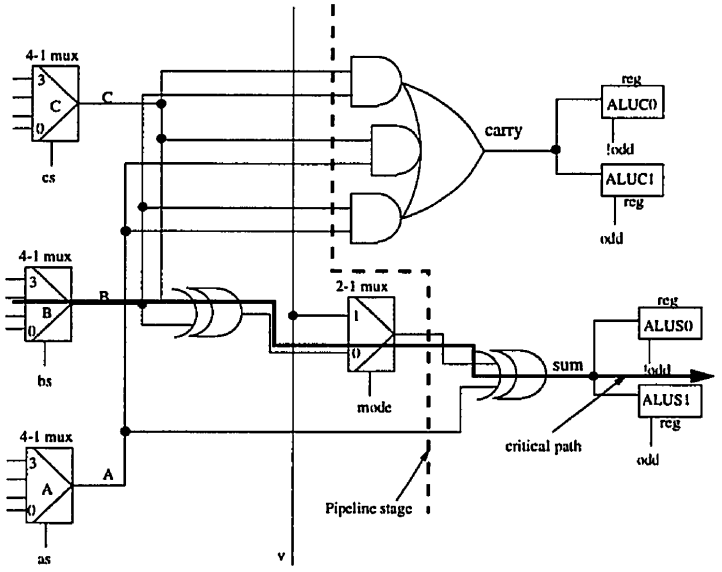


FIGURE 3.11. Critical path and insertion point of pipeline latches of the ALU

3.3.2 Data register block

Figure 3.12 shows the register-transfer-level block diagram of the data register block. The initial arrangement of the contents of the data registers is also shown in Figure 3.12. The registers labelled *b2*, *D02*, *tmpb0*, *D13*, and *D12* are muxregs which are formed by multiplexers and registers so that they can make a selection from multiple inputs.

In row 0, control signal, *lda*, controls the loading of all registers. When the loading of the coefficients is performed, the signal *ras* selects the external input so that the contents of register *b2* are overwritten by the external input. The original contents of *b2* are shifted

completed, the signal *ras* selects the output of *b0*. Then, when loading occurs, the contents of the registers are shifted within the register ring. The signal *ras* is an externally controlled signal.

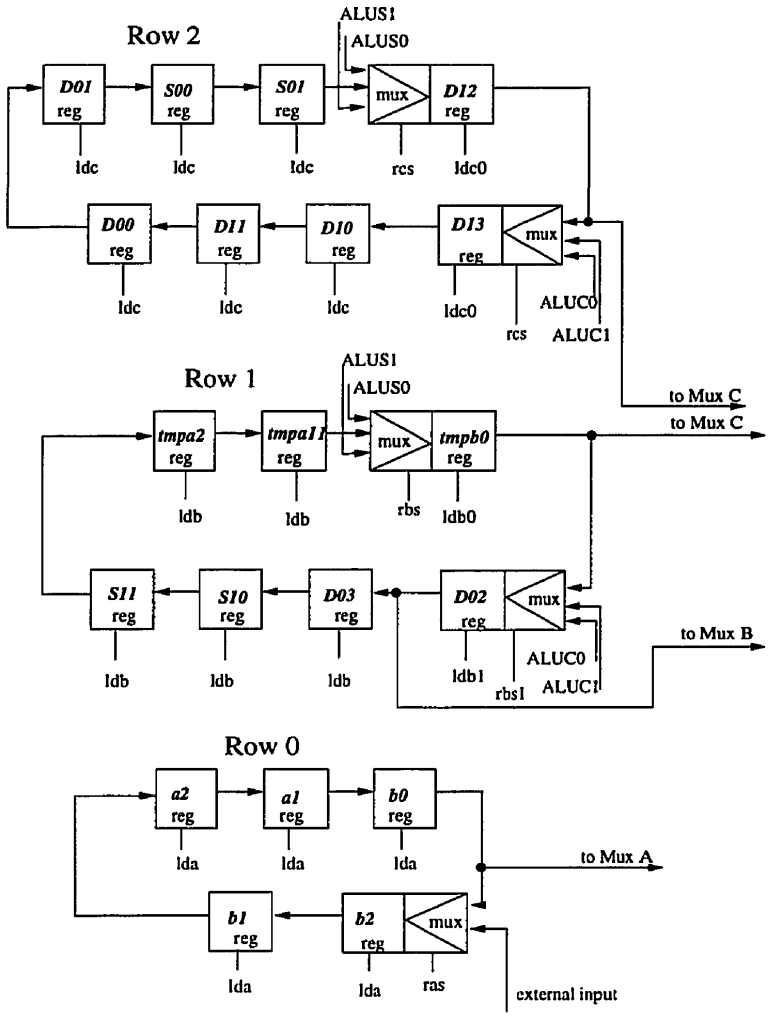


FIGURE 3.12. Data register block with the initial arrangement of the content of data in registers

In row 1, the control signals, *ldb*, control the shifting of all registers except the muxregs, *tmpb0* and *D02*. These two muxregs have their own control signals so that their contents can be updated independently without shifting the entire row. The muxreg controlled by *rbs0* and *ldb0* is responsible for updating the contents of *tmpb0*, *tmpa11*, *tmpa2*, *S10* and *D02*. The muxreg controlled by *rbs1* and *ldb1* is responsible for updating *S11* and *D03*.

muxregs *D12* and *D13*. Both muxregs are controlled by *rcs* and *ldc0*. The muxreg labelled *D12* is responsible for updating the contents of *S00, D00, D10,* and *D12*. The muxreg labelled *D13* updates the contents of *D13, D11, D01,* and *S01*.

3.3.3 1-bit signal handler block

The 1-bit signal handler block contains the following logic blocks:

- block_v which stores all 1-bit signals and generates the 1-bit XOR flag input, *v*, to the ALU.
- genNegVal which converts the 1-bit quantizers' outputs, *ty(n)* and *y(n)*, to their corresponding 28-bit fixed point values.
- Mx12lsb and Mx23lsb which are responsible for appending the 1-bit values (*u(n-1), u(n), ty(n),* and *y(n)*) to the corresponding input ports of Mux B and Mux C of the 1-bit LSB ALU block.

Figure 3.13 shows the block diagram of the 1-bit signal handler block.

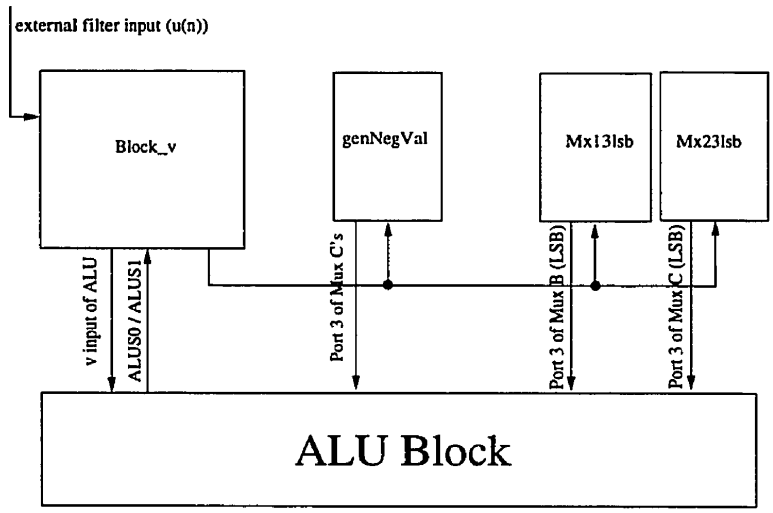


FIGURE 3.13. Block diagram of the 1-bit signal handler block

In the biquad filter, there are six XOR operations. Only the 28-bit ALU input port, *A*, (output of Mux A) and the 1-bit input, *v*, are used to perform these XOR operations - $(a \oplus \underline{v})$, where $\underline{v}$ is the 28-bit version of *v* and is obtained by replicating the value of *v* 28 times. The value of *v* comes from the 1-bit filter inputs, *u(n-1)* and *u(n)*, and the 1-bit quantization outputs, *ty(n)* and *y(n)*. The functions of the block_v are to store these 1-bit values, and to generate the corresponding *v* value for the ALU. The design of the block_v is quite easy because all it needs are four 1-bit registers to store these four 1-bit values, and a 4-to-1 multiplexer to select the corresponding *v* for the XOR instructions. Figure 3.14 shows the register-transfer-level design of the block_v. The load control signals, *ldu, ldpdout* and *ldpout*, and the select control signal, *ctlv*, originate from the control unit, and manipulate the generation of signal *v*.

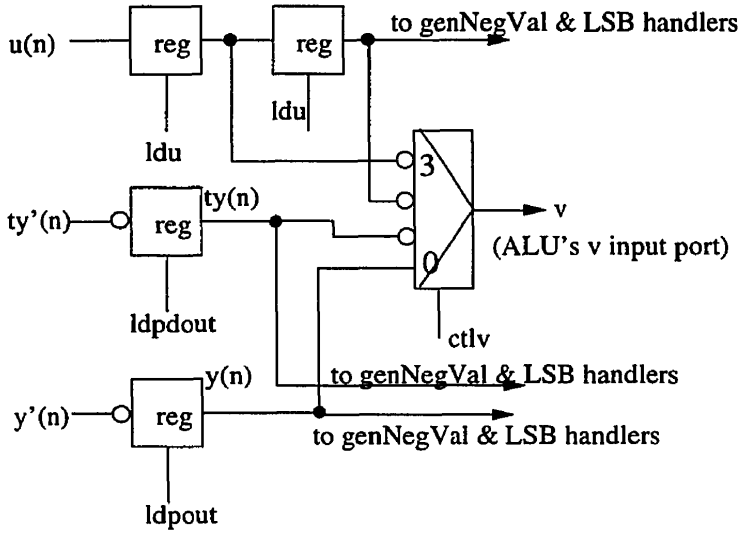


FIGURE 3.14. Design of block_v

Note that *ty(n)* and *y(n)* are the inversions of the MSB of *ALUS0* and the MSB of *ALUS1* respectively because if *ALUS0* and *ALUS1* are negative, their MSBs are 1; otherwise 0. However, for all 1-bit signals, the value '1' represents positive value (≥ 0) and the value '0' negative value (< 0). Therefore, the MSB outputs of *ALUS0* and *ALUS1* are inverted before being stored as *ty(n)* and *y(n)*.

The quantizer outputs, *ty(n)* and *y(n)*, are 1-bit values. When they are fed back to the biquad filter, these signals have to be converted to the 28-bit values before being used in the CSA instructions. Therefore, a special function block, genNegVal, is required to convert these 1-bit signals to their corresponding 28-bit fixed point 2's complement numbers. Since the CSA instructions use the negative feedback values of the modulators, conversion of these 1-bit signals becomes:

0 → **000**1.0000 0000 0000 0000 0000 0000
1 → **111**1.0000 0000 0000 0000 0000 0000

These indicate the following conversion:

x → xxx1.0000 0000 0000 0000 0000 0000

Since the fraction parts of both +1 and -1 are zero in the fixed point number system, the 1-bit feedback signal can be easily converted to a 28-bit value. Since a bit slice approach is used in the VLSI implementation of the filter, the input port 3 of Mux C of the ALU slice is via programmed to select a 0, 1, or the output of the genNegVal depending on the position of the slice. Figure 3.15 shows the implementation of the genNegVal. Figure 3.15 also shows the wiring of all bit signals to the port 3 of the ALU input multiplexer, Mux C.

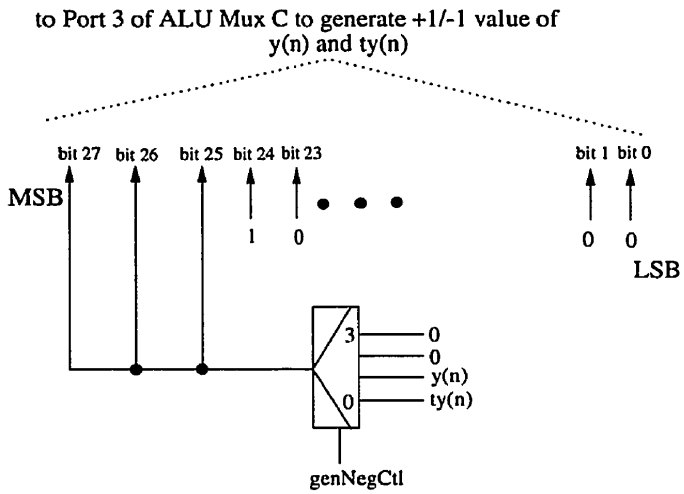


FIGURE 3.15. Block diagram of genNegVal

The LSB handlers, Mx12lsb and Mx23lsb, are responsible for providing the ALU with the corresponding saved 1-bit LSB when the related XORed results are used in the CSA instructions. Since the block_v stores all 1-bit signals needed by the Mx13lsb and the Mx23lsb, two 4-to-1 multiplexers can realize both LSB handlers. The Mx13lsb produces carry-ins for instructions 4, 6, 9 and 15, and the Mx23lsb for instructions 9 and 11 (see Figure 3.8). The wiring information and the design of both LSB handlers are shown in Figure 3.16.

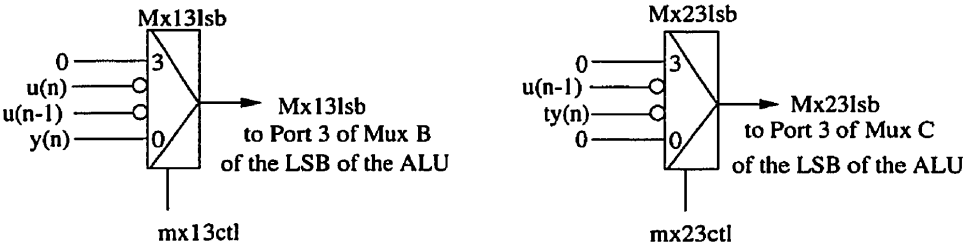


FIGURE 3.16. Block diagram of Mx13lsb and Mx23lsb.

3.3.4 Control unit

The control unit controls the data flow in the datapath and the modes of operations of the ALU. Since each filter input requires thirty-six instructions to process, a finite state machine (FSM) can be used in the control unit. The instructions are stored in a ROM (read only memory). Figure 3.17 shows the block diagram of the control unit.

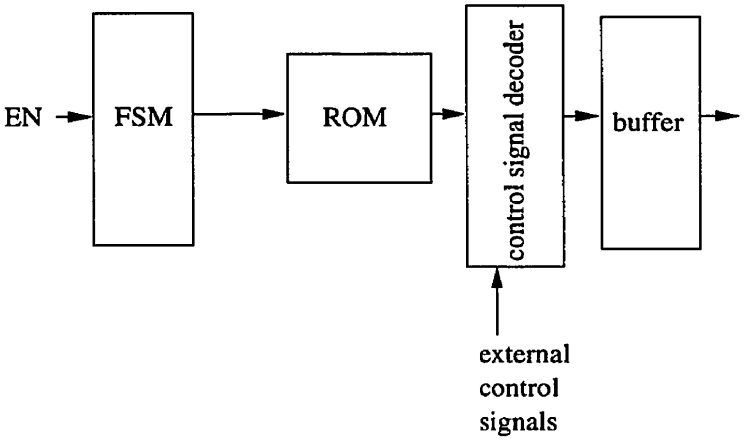


FIGURE 3.17. Block diagram of the control unit.

decoder and the control signal buffer. The overall structure of the control unit is that a FSM block generates address signals to the ROM and selects the corresponding filter instruction; then the control signal decoder decodes the instruction from the ROM and the control signal buffer sends the decoded control signals to the rest of the chip. Some external control signals are also sent to the control signal decoder so that the operations of the filter can be altered by the off-chip signals. These signals are:

- *extld* which enables the loading of the coefficients in row 0 of the data register block,
- *ras* which controls the muxreg of the row 0 of the data register block and selects the corresponding input port for the muxreg when loading coefficients is performed, and,
- *nclr* which is the active-low clear / reset signal to the filter chip.

In addition to these signals, the *EN* signal sent to the FSM enables the entire filter. Upon receiving a new filter input, the *EN* signal goes from 0 to 1 which causes the filter to process the new input. The following sections discuss the implementation of the function blocks of the control unit.

3.3.4.1 Finite state machine (FSM)

The data-flow graph in Figure 3.8 shows that thirty-six clock cycles are required to process each filter input. This implies the FSM should have thirty-six states. Since a ROM stores all filter instructions, the FSM must generate suitable output signals to be used by the ROM. The output of the FSM is a 36-bit wide one-hot encoded value and each signal generates one clock cycle wide pulse for every instruction. However, as seen in Table 3.1, some filter instructions are identical. Therefore, to save area, only unique instructions are stored in the ROM. OR-gates are needed to map the multiple state outputs of the FSM to specific instruction words in the ROM. Figure 3.18 shows the block diagram of the FSM block which consists of a pulse generator, the FSM and the output OR-gates.

processed, the external control signal *EN* will go from 0 to 1. Then, the pulse generator generates an one clock cycle wide pulse to the FSM. The pulse travels through the thirty-six register delay line. The outputs of the register delays provide state information to the OR-gates. These OR-gates map the state outputs to the corresponding filter instructions. The outputs of the OR-gates can be used directly as the word lines of the ROM.

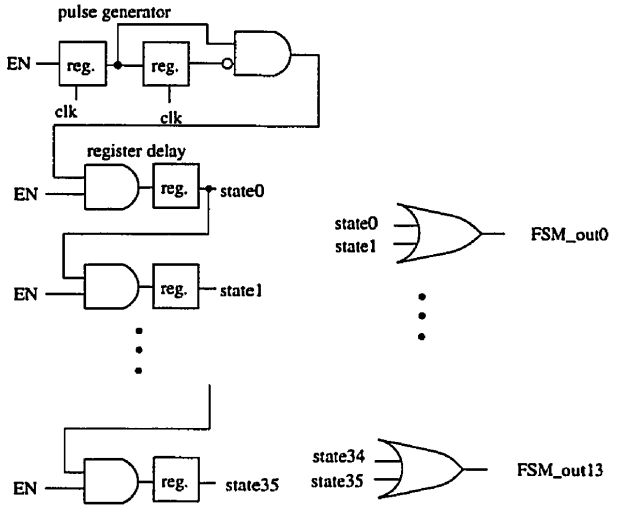


FIGURE 3.18. Block diagram of the finite state machine

3.3.4.2 ROM

From Table 3.1, there are only twenty-seven unique 32-bit instructions because some of the instructions are identical. The ROM stores these twenty-seven distinct filter instructions only to save area.

In the actual implementation of the ROM, a 14 × 64 bit ROM was laid out. The detailed layout and the design of the ROM will be discussed in Chapter 4. Since twenty-seven unique 32-bit words are stored in the 14 × 64 bit ROM, two 32-bit words are stored in the same address location. Then a 32-bit 2-to-1 multiplexer is used to select the corresponding 32-

selecting the 32-bit instruction for a given FSM state output. If *sel* = 0, the even numbered filter instruction is selected. If *sel* = 1, the odd numbered filter instruction is selected. The design of the *sel* signal circuit is shown in Figure 3.19.

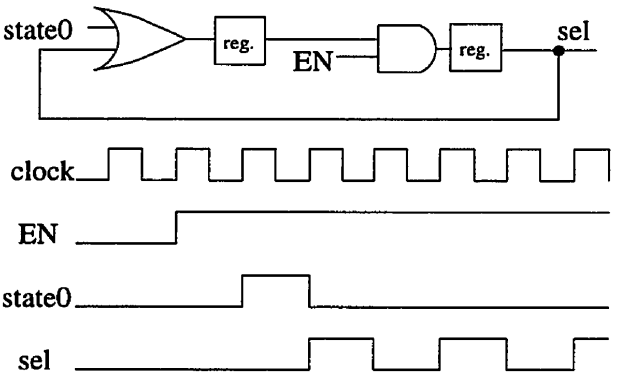


FIGURE 3.19. Design of the sel signal circuit

3.3.4.3 Control signal decoder and buffer circuit

Before the control signals are delivered to the registers and the ALU, they need to be decoded and buffered. There are four functions of the control signal decoder and the buffer circuit. They are:

- decoding 2-bit select signals of all multiplexers and muxregs to one-hot 4-bit select signals so that multiplexers and muxregs can be realized by simple TSPC 4-input AND-OR gates to save area,
- allowing external signals to control the operations of the filter,
- keeping the states of the filter unchanged when the filter is idle, and,
- buffering the control signals to the filter core

The first function of the control signal decoder is to decode the 2-bit select signals to 4-bit signals. The 2-to-4 decoders are used in decoding. The reason for decoding these 2-bit

realized by simple 4-input AND-OR gates directly. Since the 4-input AND-OR gate is simpler than a full 4-to-1 multiplexer, using it to realize 4-to-1 multiplexers with four decoded select signals will save silicon area and improve the speed of the filter. For example, for the 2-bit select signal, *as*, of Mux A of the ALU (see Figure 3.10), a 2-to-4 decoder is used to decode the 2-bit *as* to four unbuffered *as* signals *(as0, as1, as2 & as3).* These pre-decoded select signals can be used by Mux A which is realized by a 4-input AND-OR gate. Figure 3.20 shows the logic gate level design of the 2-to-4 decoder.

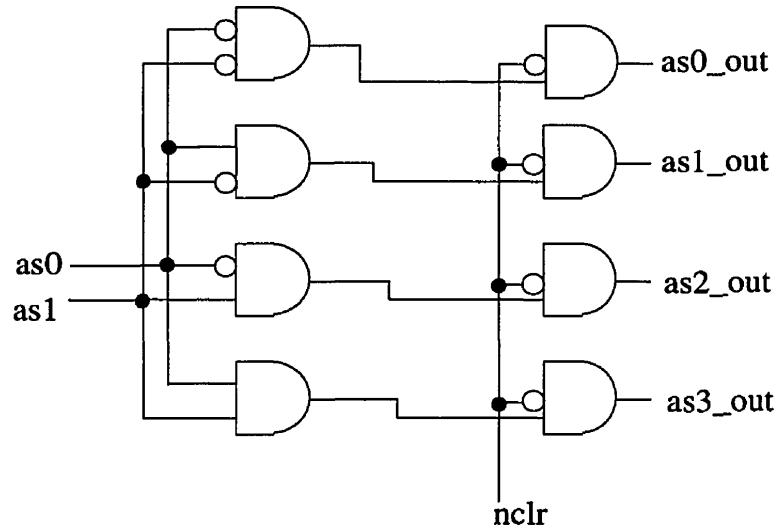


FIGURE 3.20. 2-to-4 decoder with clear signal (nclr)

In addition, with one AND-OR input set to 0, the 4-input AND-OR gate can also be used in the calculation of *carry*. The carry calculation block, the muxregs and the input multiplexers in the ALU can be realized by the same logic circuit, the 4-input AND-OR gate. This saves development time by reusing the same logic circuit for multiple blocks.

The second function of the control signal decoder is to allow external signals to control the operations of the filter core. Since resetting the filter with *nclr* and loading the filter coefficients with *extld & ras* are controlled externally, these external control signals are introduced to the control signal decoding stage. The active-low reset signal, *nclr*, for example,

signals become 0. Then, the contents of all TSPC registers and the ALU will be cleared to 0 in the next clock cycle. More importantly, the fanout of the *nclr* signal (which AND's all control signals in the decoding stage) is less than that of clearing twenty 28-bit registers and the ALU. Figure 3.21 shows the use of the *nclr* and *extld* signals.

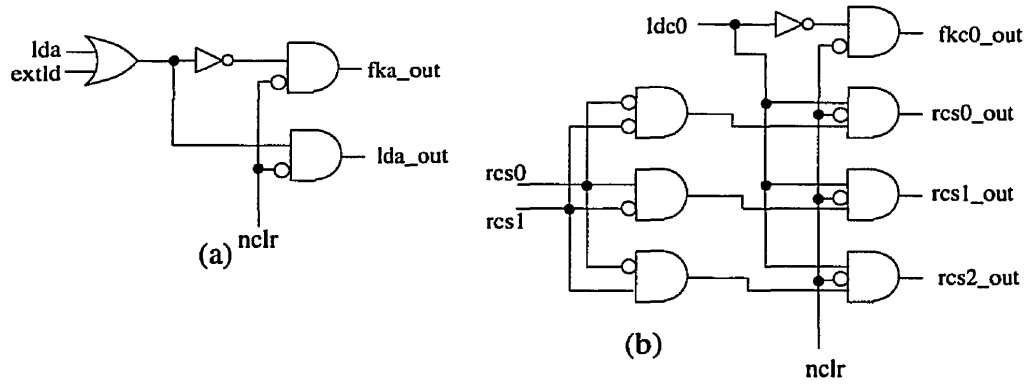


FIGURE 3.21. External control signal circuits for extld and nclr

The third function of the control signal decoder is to keep the states of the filter unchanged when the filter is idle or is waiting for input. Using the control signals from the ROM, the control signal decoder can generate feedback signals to make the registers and the ALU hold their values when the filter is idle. In general, the feedback signals of the registers are the inversions of the registers' load signals. Figure 3.21 (a) shows the generation of the feedback signal of *fka_out* from *lda* and *extld*, where *lda* is the load signal of the registers in row 0. Figure 3.21 (b) shows the generation of *fkc0_out* feedback signal for the muxregs of the row 2. Note that there is no load signal for the muxregs of row 2 because the load signal is ANDed with the select signals, as the muxregs are implemented by simple 4-input AND-OR gates.

The fourth function of the control signal decoding and buffer stage is to buffer all control signals to the rest of the chip. The detailed design of the signal buffer is presented in the Chapter 4.

Several C programs were used to verify the filter architectures shown in Figure 2.6, Figure 3.5 and Figure 3.9. A register-transfer-level model of the biquad filter using the architecture discussed in section 3.3 was written in Tortle. Simulations were done using 64K input samples per frequency with the filter coefficients in Table 3.2 and OSR of 128. For all simulations, the first 48K outputs were ignored in the calculation of the power spectrum of the filter output to remove the transient. Then, the 16K outputs left were used to calculate the power spectrum of the output. Figure 3.22 shows the transfer characteristic of the Tortle [15] model of the Δ–Σ based biquad filter using the single ALU architecture. The signal bandwidth f_s is equal to 64 because the OSR = 128 and 16K samples were used in calculation of power spectrum. All biquad filter models gave the same filter transfer characteristics for this set of filter coefficients, and the outputs of the ALU for all models matched bit for bit.

coefficient	value
a1	0.007832105606854
a2	0.001105716672898
b0	0.001111633439626
b1	8.706430495261652e-06
b2	0.007874015748031

TABLE 3.2. Filter coefficient table

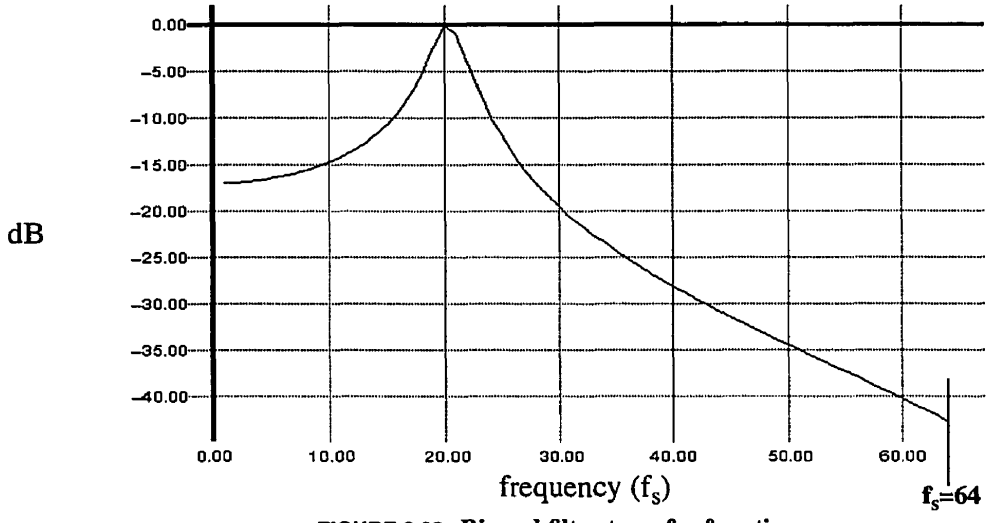


FIGURE 3.22. Biquad filter transfer function

CHAPTER 4 *VLSI Implementation of the Delta-sigma Based Biquad Filter*

This chapter presents the design and implementation of the Δ–Σ based biquad filter chip. This filter chip uses the single ALU architecture described in chapter 3. Section 4.1 describes the floor plan of the biquad filter chip. Section 4.2 describes the design of the filter core and its basic cells. The implementation of the control unit is discussed in section 4.3. The I/0 interfaces such as the input block and the output block are presented in section 4.4 and section 4.5 respectively. The clock generation and the clock distribution are described in section 4.6. The power estimation and the power distribution are discussed in section 4.7. The summary of the chip is presented in section 4.8. Finally, the simulation and test results are given in section 4.9.

4.1 Floor Plan of the Biquad Filter Chip

As discussed in Chapter 3, the single ALU version of the biquad filter contains an ALU block, a register block, a 1-bit signal handler block and a control unit. In addition, several other blocks are required in the chip because the clock signal used by the filter is generated internally for efficiency, and an I/O interface is used to synchronize the slow off-

component blocks and their functions:

- The biquad filter core contains a 28-bit ALU, twenty 28-bit registers and four 1-bit signal handlers. The core performs all arithmetic for the biquad filter.
- The control unit contains a finite state machine (FSM), a ROM, a control signal decoder and a signal buffer. The control unit generates control signals to the filter core.
- The input block handles the filter input, the loading of the filter coefficients, and the external control signals.
- The output block handles the buffering of the output signals to the pads, and provides the filter busy signal off-chip.
- The clock generation unit contains a voltage controlled oscillator (VCO) to generate the clock signal used in the filter chip. A clock buffer tree distributes the clock signal to the entire chip.

Figure 4.1 shows the floor plan of the filter chip. In this floor plan, the internal signals of the 28-bit filter core flow horizontally. The control signals are delivered vertically upwards from the control unit in the bottom. Local clock buffers are located on the left side of the filter core, and help reduce clock skew. The clock signal moves from the left to the right. The details of the clock generation and distribution will be discussed in section 4.6. DC power is delivered from both sides (left and right) of the chip so that the IR drop between the pads and the filter logic can be minimized.

In the biquad filter, all basic cells have the same pitch to make the power distribution and the clock distribution efficient. The clock distribution scheme and the power distribution scheme will be addressed in section 4.6 and section 4.7 respectively. The following sections discuss the design and implementation of all major component blocks of the filter chip shown in Figure 4.1.

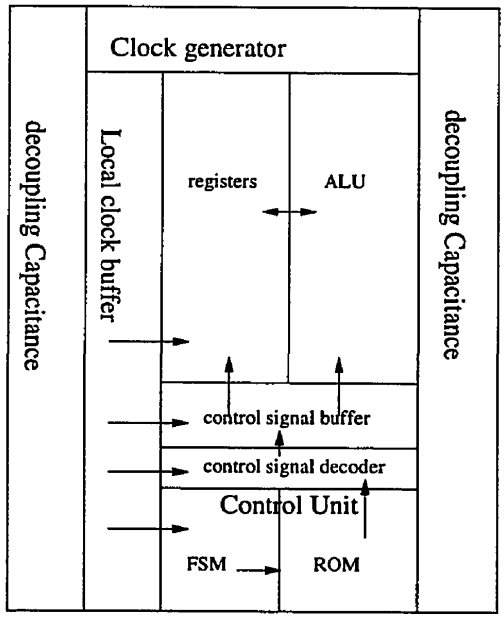


FIGURE 4.1. Floor plan of the biquad filter chip

4.2 Biquad Filter Core

The biquad filter core contains the 28-bit ALU, twenty 28-bit registers and four 1-bit signal handlers. A regular design approach can be applied to the design of the filter core, because each bit of the ALU is the same and each bit of the register block is identical. Therefore, a bit slice approach can be used in the design of the core except for the 1-bit signal handlers. As discussed in Chapter 3, there are four 1-bit signal handlers in the core, which are not difficult to lay out. Using the bit slice approach, only a 1-bit ALU and twenty 1-bit registers need to be laid out. The 28-bit core is obtained by replicating the bit slice 28 times in addition to the 1-bit signal handlers.

presented first. Then, the design of the basic components of the bit slice is discussed. Finally, the design of the 1-bit signal handlers is discussed at the end.

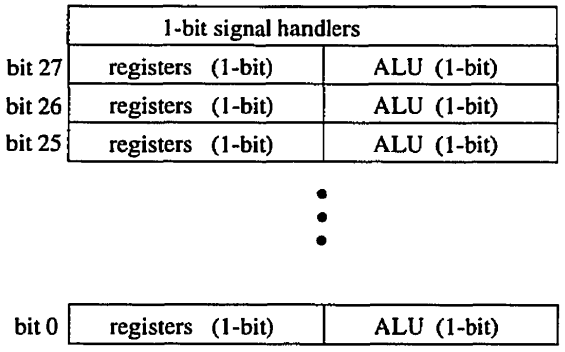


FIGURE 4.2. Floor plan of the biquad filter core

4.2.1 Bit slice approach

The bit slice of the filter core is made from a 1-bit ALU and a 1-bit wide register file. However, the ALU and the register file have a different number of basic cells. Pitch matching has to be done to make power distribution and clock distribution efficient. This also allows easy replication of the slice and saves silicon area. Figure 4.3 illustrates pitch matching of the ALU and the register file.

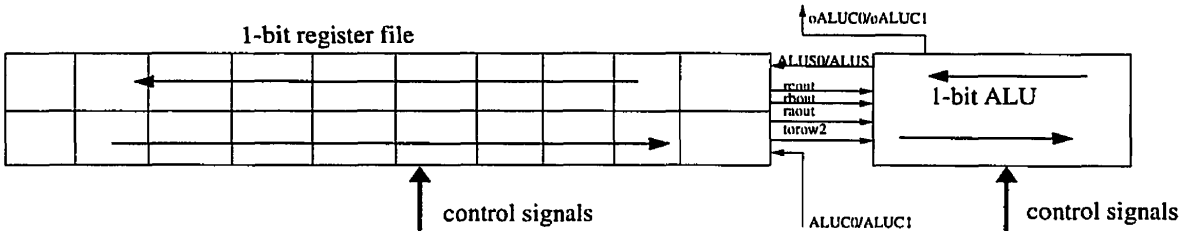


FIGURE 4.3. Floor plan of the 1-bit slice of the biquad filter core

The routing of the local interconnect is done on top of the basic cells so that no space is wasted. The control signals flow across the bit slice vertically from the bottom control unit. This bit slice is replicated 28 times in the filter core.

All basic cells of the core were laid out with the same height so pitch matching can be done easily. Figure 4.3 shows that there are two rows of registers in the 1-bit wide register file

basic cells. If one row of registers were used in the pitch matching, it would be hard to lay out and the local routing length would be long. Since the basic cells do not have large fanout, if long metal wires are to do the local routing, the load and the signal delay will be increased. If three rows of registers were used, the height of the bit slice would be tall. A tall bit slice is not desirable because after replicating it 28 times, the biquad core would be too tall. This would make the filter chip a thin rectangle instead of square. The taller the core, the longer the control signal wires will be. This increases delay and load of the control signals. The following sub-sections discuss the detailed floor plan and the design of the 1-bit ALU and the 1-bit wide register file.

4.2.1.1 Floor plan of the 1-bit ALU

Figure 4.4 shows the arrangement of the basic cell layouts of the 1-bit ALU. The detailed design and implementation of the 1-bit ALU and its basic cells using TSPC logic will be discussed later. As shown in Figure 4.4, the basic cells of the ALU are arranged into two rows like the register file. This makes the interconnection between the ALU and the register file easy.

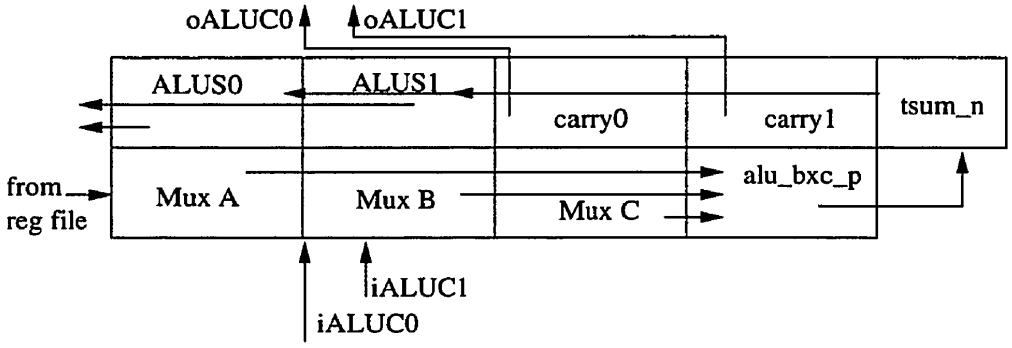


FIGURE 4.4. Arrangement of the basic cell layouts of the 1-bit ALU

4.2.1.2 Floor plan of the 1-bit wide register file

Figure 4.5 shows the arrangement of the basic cell layouts and the rough interconnect of the 1-bit wide register file. As seen in Figure 4.5, there are two types of registers in the

register file. They are register and muxreg. The detailed design of the muxreg and the register will be addressed later. For now, assume that the width of the muxreg is about twice of that of a register.

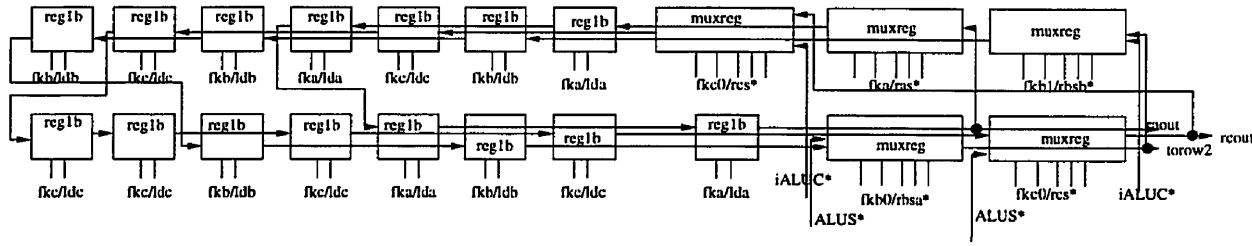


FIGURE 4.5. Arrangement of the basic cells and the rough interconnect of registers in the register file

4.2.2 ALU

Section 3.3.1 discussed the register-transfer-level design of the 1-bit ALU. This section describes the logic block partitioning of the 1-bit ALU so that it can be implemented by TSPC logic basic cells efficiently. Figure 4.6 shows the block diagram of the 1-bit ALU for efficient TSPC logic implementation. As seen in Figure 4.6, there are many input, output and control signals. Table 4.1 summarizes all signals of the 1-bit ALU shown in Figure 4.6.

Signal Name	Descriptions
m	the mode signal of the ALU sent from the control unit
iALUC0, iALUC1	the carry-in signals from the lower core bit slice
oALUC0, ALUC1	the carry-out signals of the current core bit slice
ALUS0, ALUS1	the summation output of the current core bit slice
raout, rbout, rcout, torow2	the outputs of the register file of the current core bit slice
as, bs, cs	the 4-bit pre-decoded select signals of the input multiplexers
ld, ld0, ld1, pld0, pld1	the load control signals to enable loading of all muxregs and the output registers with selectable load inputs
fbk, fbk0, fbk1, fpld1, fpld0	the feedback control signals to keep the state of the ALU when the ALU is idle.

TABLE 4.1 Input, output and control signals of the 1-bit ALU

The general design of the 1-bit ALU in Figure 4.6 is discussed in the following subsections. The detailed TSPC basic cells of the ALU will be presented in section 4.2.3.

As mentioned in Chapter 3, all multiplexers are implemented by 4-input AND-OR gates with pre-decoded feedback and select signals. As seen in Figure 4.6, the input multiplexers and the carry calculation block are implemented by the same basic cell, muxreg4x1b. The muxreg4x1b is made by a 4-input AND-OR gate with feedback. In the carry calculation, one pair of the AND-OR inputs of the muxreg4x1b are grounded. Note that the muxreg4x1b feeds its output back from the P-latch so that when the filter is idle, the muxreg4x1b holds its contents. The detailed design of the muxreg4x1b is discussed in section 4.2.3.2.

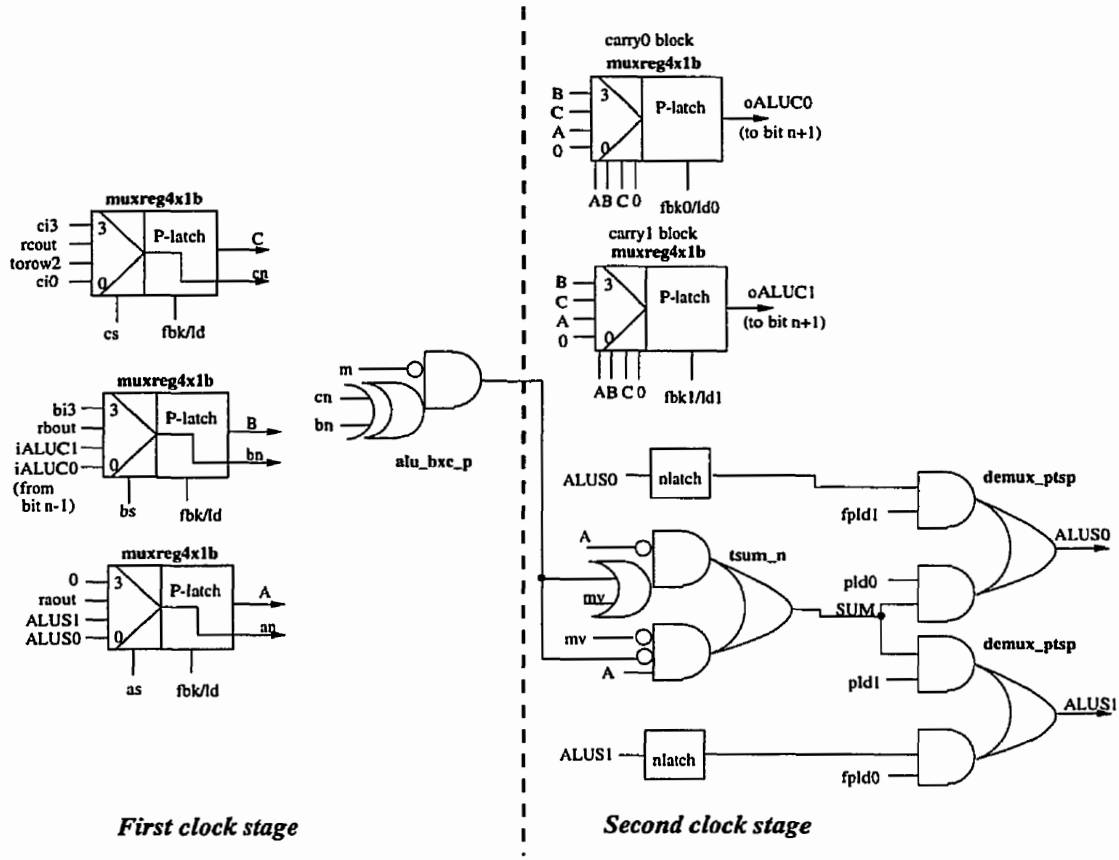


FIGURE 4.6. Block diagram of the partition of the ALU for TSPC logic implementation

The summation part is in the critical path of the ALU. From the information in section 3.3.1, the following logic manipulation is done to EQ 4.1 to avoid a complicated logic realization of the summation part with TSPC.

$$SUM = A \oplus (m \bullet v + !m \ (B \oplus C)) \quad \text{(EQ 4.1)}$$

$$A \oplus (m \bullet v + !m (B \oplus C)) = !A (m \bullet v \ + !m (B \oplus C)) + A (!(m \bullet v) !(!m (B \oplus C))) \quad \text{(EQ 4.2)}$$

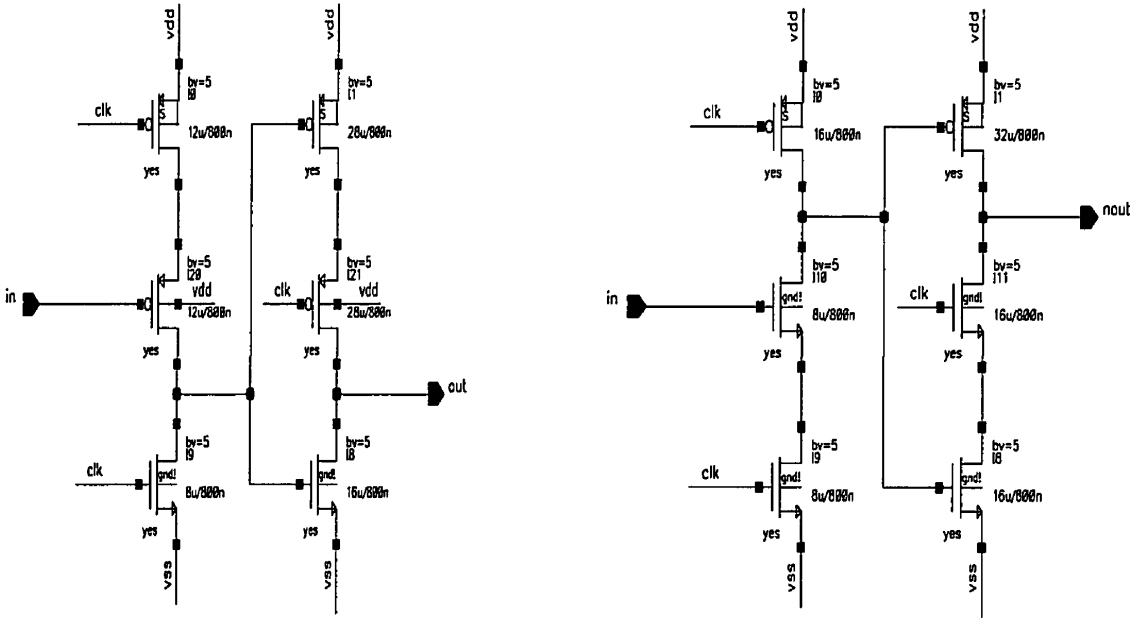
The *m* signal is the *mode* signal of the ALU. With this expansion of EQ 4.1, the expressions $m \bullet v$ and $!m \ (B \oplus C)$ can be evaluated in the first ALU clock cycle. The final *SUM* can be calculated in the second ALU clock cycle. This ensures that the summation TSPC logic will not be complex. As seen in Figure 4.6, the basic cells, alu_bxc_p, evaluates the expression, $!m \ (B \oplus C)$. The basic cell, tsum_n, are used to calculate the final *SUM*. The basic cell, demux_ptsp, is used to store the value of *SUM*.

4.2.3 Basic cells of the ALU

This section presents the TSPC logic implementation of all basic cells of the ALU. They are P-latch, N-latch, muxreg4x1b, alu_bxc_p, tsum_n, and demux_ptsp.

4.2.3.1 P-latch and N-latch

The P-latch and the N-latch are used to delay the signals by half of a clock cycle or to buffer weak signals to have stronger drive. Figure 4.7 shows the transistor view of the P-latch and the N-latch using TSPC logic. As seen in Figure 4.7, the transistors in the output stage of both latches are twice the size of those in the input stage to allow at least twice the drive capability.



a) P-latch b) N-latch

FIGURE 4.7. a) P-Latch, b) N-Latch

4.2.3.2 Muxreg4x1b cell

The muxreg4x1b implements the input multiplexers and the carry calculation blocks of the ALU. Figure 4.8 shows the circuit of the muxreg4x1b cell. As seen in Figure 4.8, the muxreg cell evaluates EQ 4.3.

$$out = fbk \bullet out + ld\,(sel0 \bullet in0 + sel1 \bullet in1 + sel2 \bullet in2 + sel3 \bullet in3) \qquad \text{(EQ 4.3)}$$

During the multiplexing operation, only one of the pre-decoded select signals is 1. However, when the muxreg4x1b cell is used in the carry calculation, it performs the function of a normal 4-input AND-OR-gate with *sel3* and *in3* set to 0. Note that the *fbk* and *ld* signals are ANDed with the reset signal, *nclr*, in the control unit. When *nclr* is 0, the muxreg4x1b is reset.

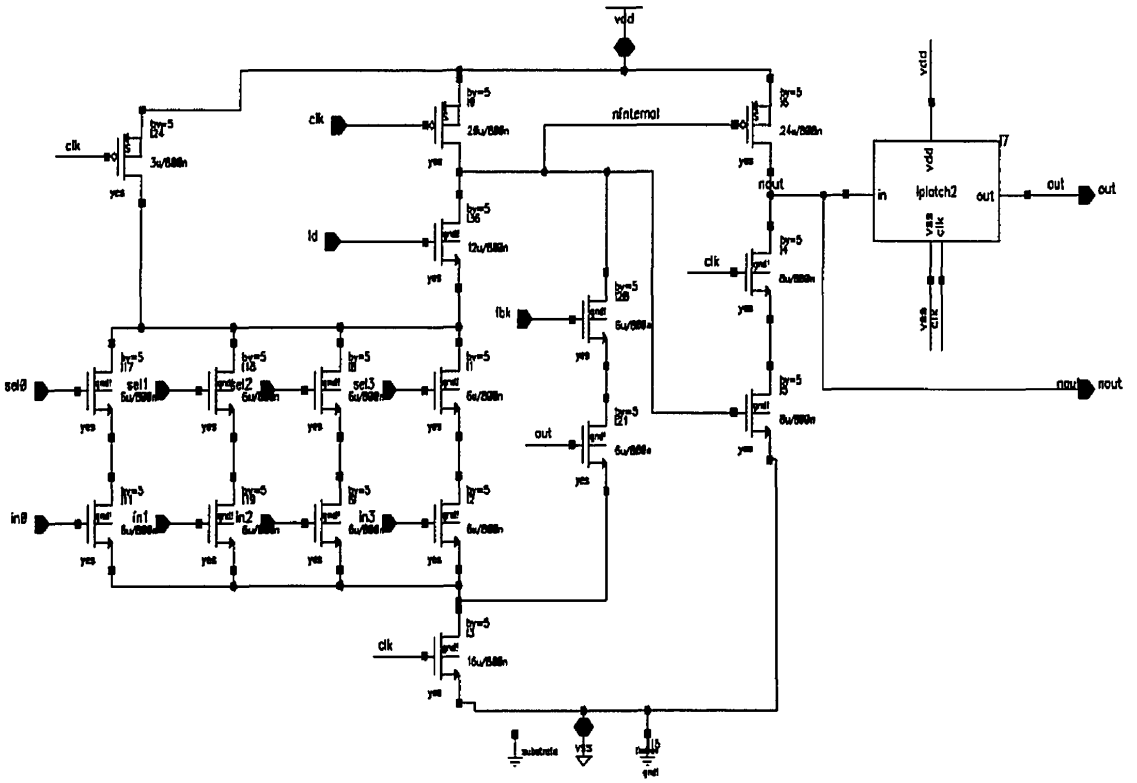


FIGURE 4.8. Circuit of the muxreg4x1b cell of the ALU

4.2.3.3 Alu_bxc_p cell

The alu_bxc_p cell evaluates the expression *!m (B ⊕ C)*. Since this expression is fairly complicated to use in a P-block stage, its implementation is based on the P-block of all-N TSPC logic to meet the speed requirement. Device sizing was done on this function block to ensure that this alu_bxc_p cell works at 660 MHz. (The device sizing was done by a lot of trials.) The NMOS transistors in the function block are larger than those in other basic cells' function blocks to allow strong pull-up. Figure 4.9 shows the circuit of the alu_bxc_p cell. Figure 4.9The active-low, *notm*, signal is the inversion of the *mode* signal.

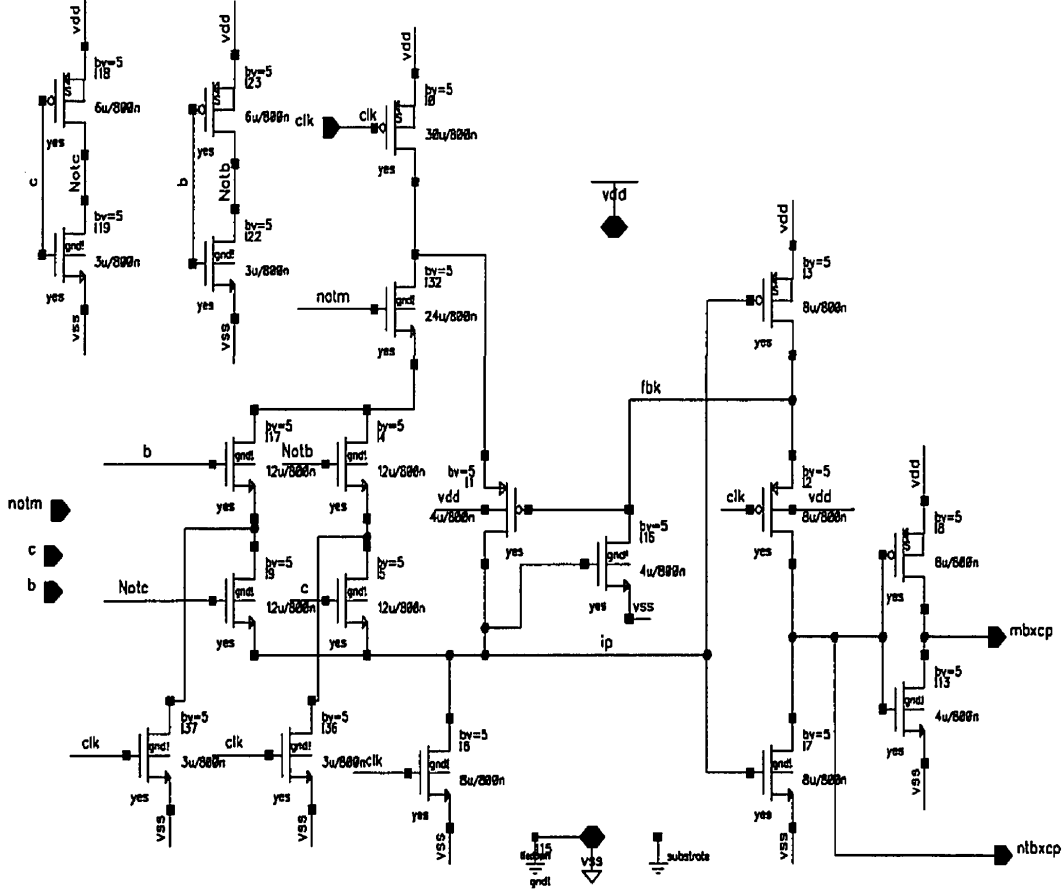


FIGURE 4.9. Circuit of the alu_bxc_p cell of the ALU

4.2.3.4 Tsum_n cell

The tsum_n cell evaluates the expression $A \oplus (m \bullet v + !m \ (B \oplus C))$ and produces the final SUM. Note that the value of $m \bullet v$ and the value of the expression $!m(B \oplus C)$ are calculated in the previous ALU clock cycle. All internal nodes in the logic function are pre-charged to 1 to eliminate the charge sharing problem. Figure 4.10 shows the circuit of the tsum_n cell.

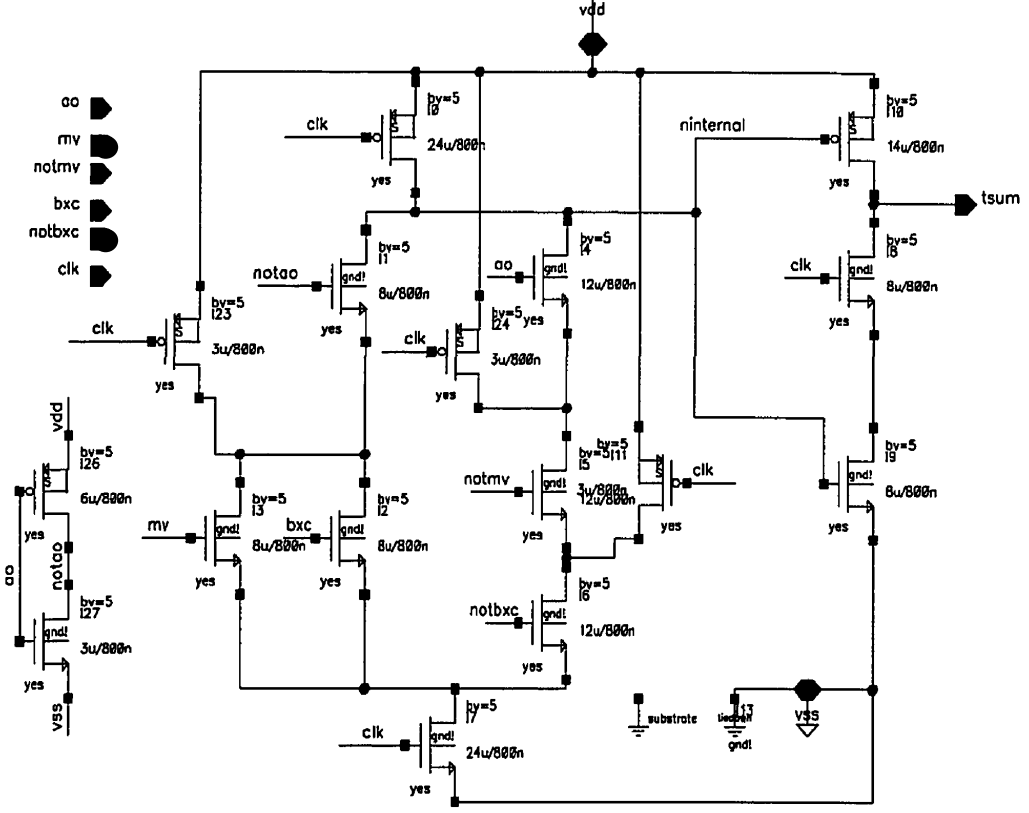


FIGURE 4.10. Circuit of the tsum_n cell of the ALU

4.2.3.5 Demux_ptsp cell

Figure 4.11 shows the circuit of the demux_ptsp cell. The demux_ptsp cell evaluates the expression, *sel0* •*in0* + *sel1* •*in1*. In the ALU, the demux_ptsp cell either loads the SUM output of the tsum_n cell, or keeps its output value through an N-latch. To do this, *sel0* is connected to the feedback signal, *fpld0* or *fpld1*. *in0* is connected to the feedback output of the N-latch. *sel1* is connected to the load signal, *pld0* or *pld1*. *in1* is connected to the output of the tsum_n cell. Note that the load signals are pre-decoded in the control unit so that only one of the demux_ptsp cells in the ALU loads the output of the tsum_n cell at any time. All control signals are ANDed with the reset signal, *nclr*, in the control unit. Since the logic function of the demux_ptsp is complex, its implementation is based on the structure of the P-block of all-N TSPC logic to meet the speed requirement.

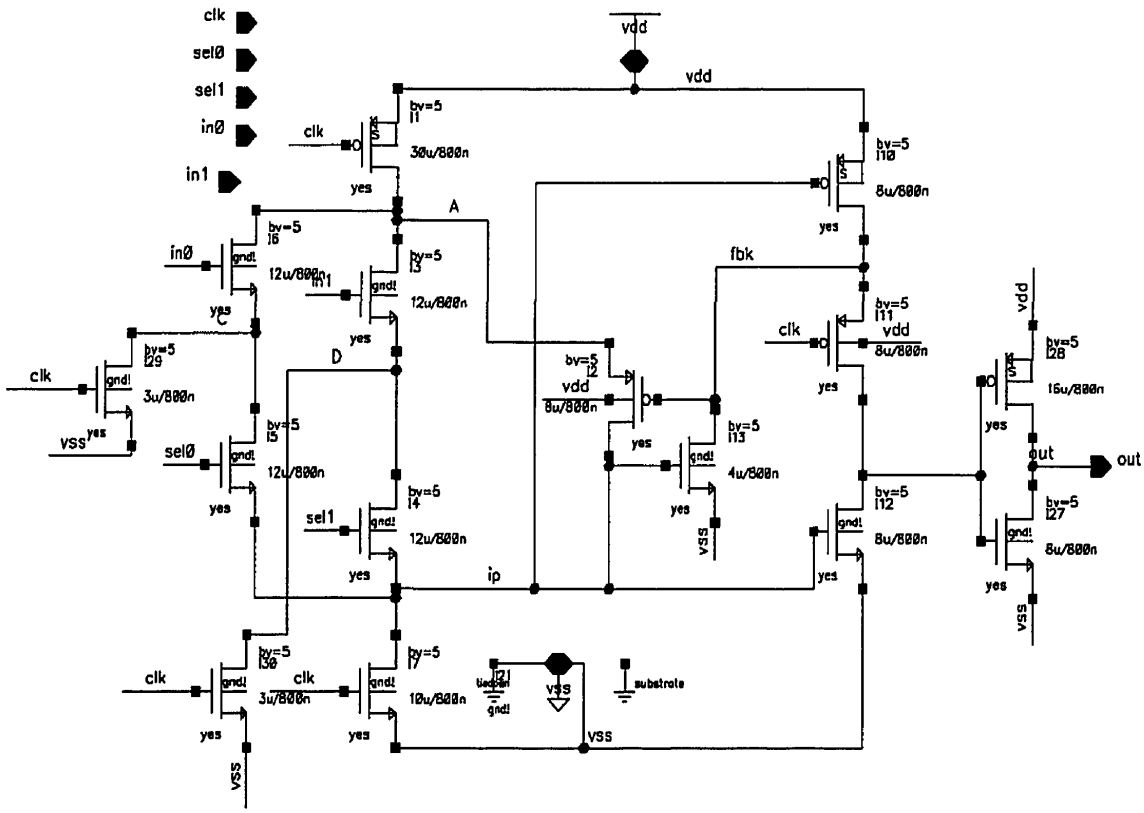


FIGURE 4.11. Circuit of the demux_ptsp cell of the ALU

4.2.4 Basic cells of the 1-bit register file

There are two types of registers in the register file:

- reg1b, which is a register, is able to load input or to keep its value by feedback, and
- muxreg, which is realized by a 4-input AND-OR gate, is able to select multiple inputs or to hold its contents by feedback.

4.2.4.1 Reg1b cell

The reg1b cell performs the logic function of $ld \bullet in + fbk \bullet out$. Both *ld* and *fbk* signals are pre-decoded in the control unit and also ANDed with the reset signal, *nclr*. Figure 4.12 shows the circuit of the reg1b.

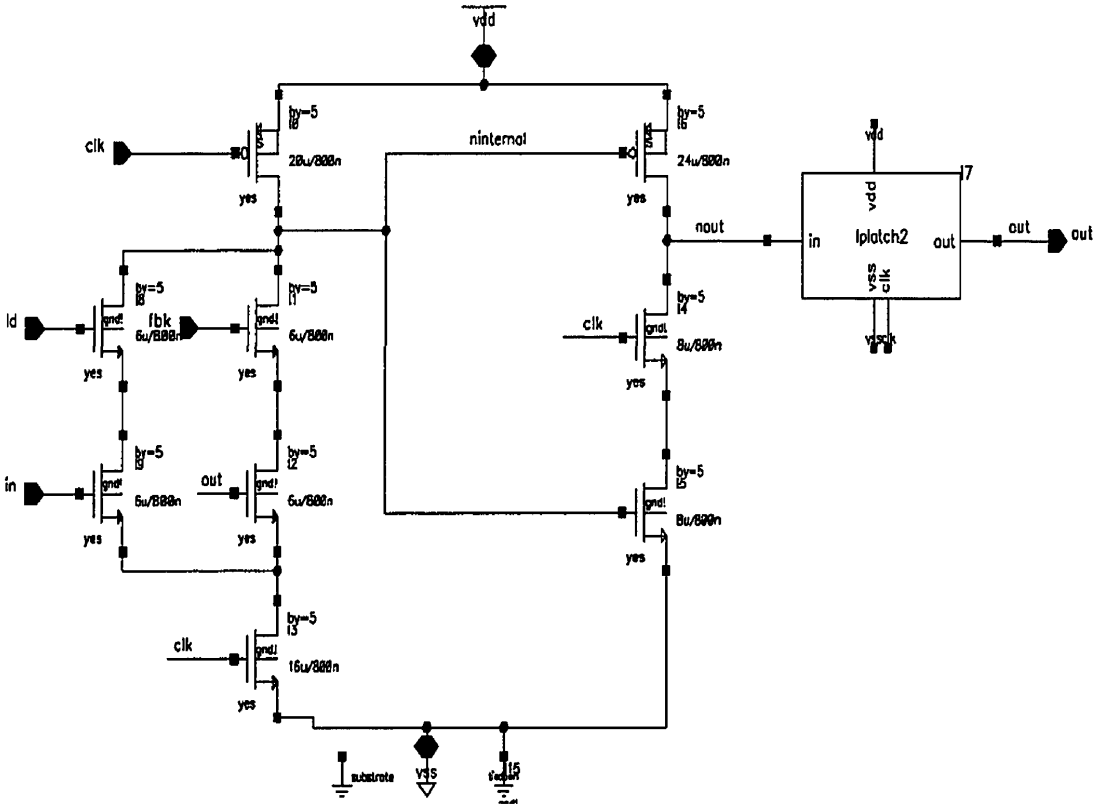


FIGURE 4.12. Circuit of the reg1b of the 1-bit wide register file

4.2.4.2 Muxreg cell

The muxreg cell is actually made from a 4-input AND-OR gate. The select control signals and the feedback control signal of the muxreg in the register file are pre-decoded and also ANDed with the reset signal, *nclr*. Since all muxregs of the register file have only three pairs of AND-OR inputs, one of the AND-OR inputs is used as the feedback mechanism. Figure 4.13 shows the circuit of the muxreg and its feedback path.

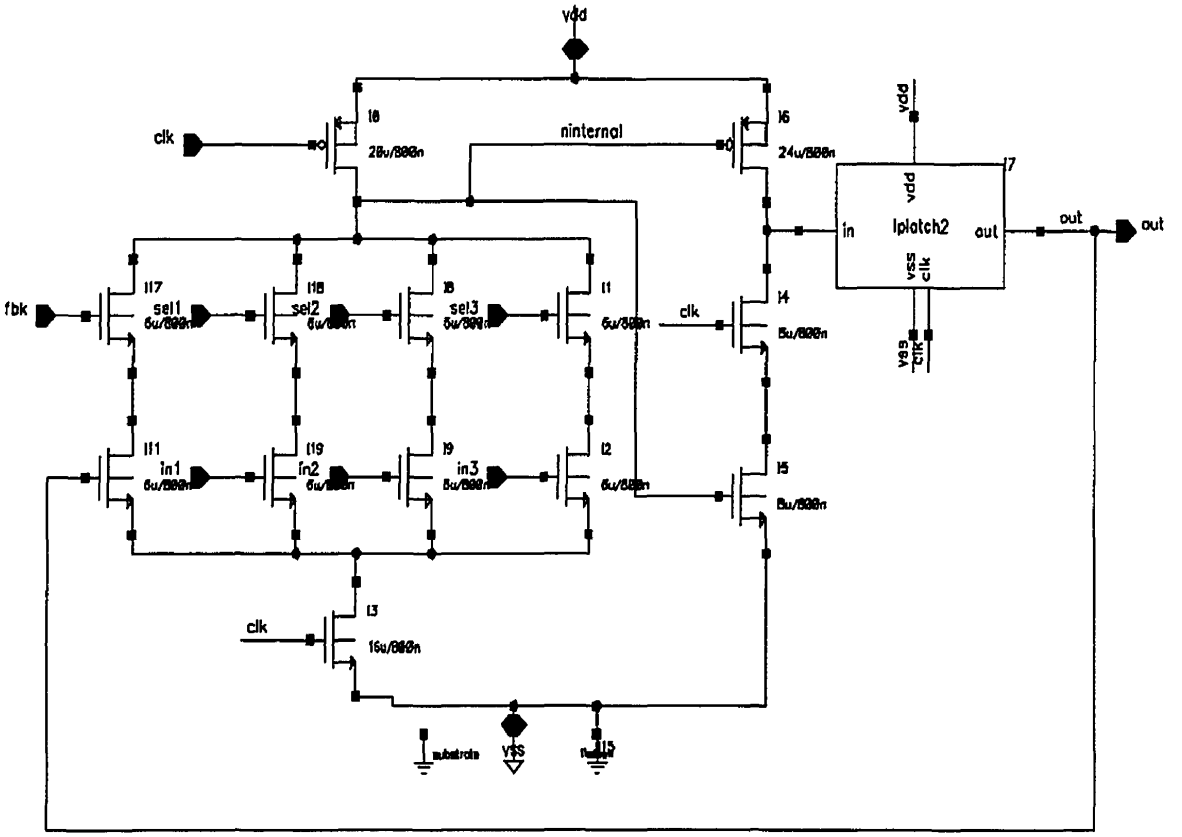


FIGURE 4.13. Circuit of the muxreg of the 1-bit wide register file

4.2.5 1-bit signal handler block

As discussed in section 3.3, the design of all 1-bit signal handlers such as block_v, genNegVal, Mx13lsb and Mx23lsb are based on 4-to-1 multiplexers and 1-bit registers. Thus, the muxreg cell used by the register file can realize the genNegVal, the Mx13lsb and the Mx23lsb cells directly except the block_v cell.

4.2.5.1 Design of block_v

The output of the block_v cell is *mv*, which is the ANDed result of signals, *mode* and *v*. This output is used directly by the tsum_n cell. Since the fanout of *mv* is 28, the output of

design of the block_v cell.

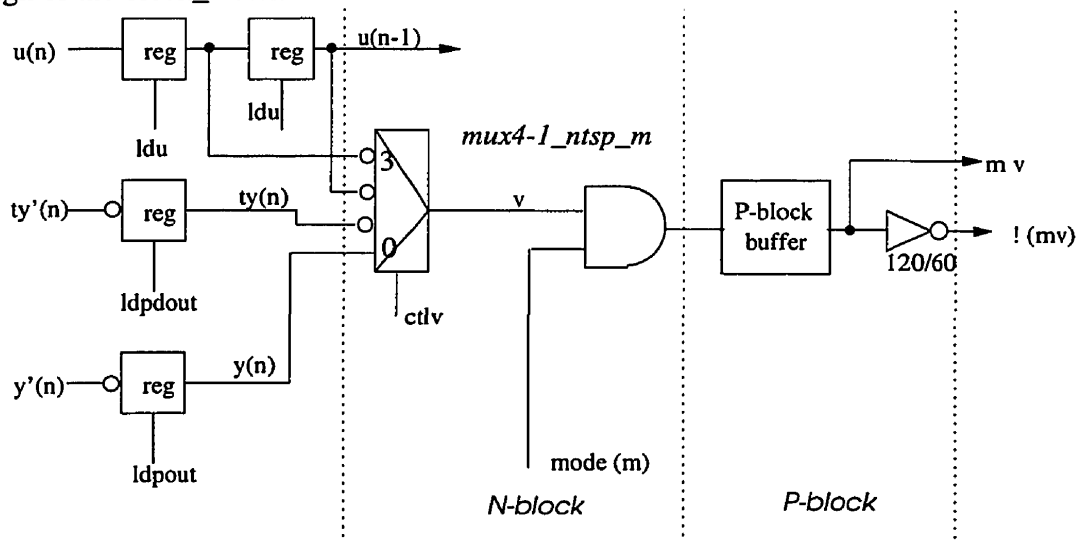


FIGURE 4.14. Logic design of the block_v

As shown in Figure 4.14, the block_v cell uses three kinds of basic cells such as the 1-bit register, mux4-1_ntsp_m and pbuf27. Four 1-bit registers store all 1-bit signals. The mux4-1_ntsp_m selects the corresponding 1-bit signal to be the *v* signal, and produces *mv*. The pbuf27 and the inverter buffer the *mv* and *!mv* signals to the 28-bit ALU. The reg1b described in the register file can be used to implement the 1-bit register. The mux4-1_ntsp_m can be obtained by modifying the muxreg.

Figure 4.15 shows the circuit of the mux4-1_ntsp_m. The mux4-1_ntsp_m cell evaluates the expression, $m\ (sel0 \bullet in0 + sel1 \bullet in1 + sel2 \bullet in2 + sel3 \bullet in3)$. By comparing Figure 4.14 and Figure 4.15, the *sel0-sel3* are the pre-decoded 4-bit *ctv* signal generated from the control unit.

Figure 4.16 shows the circuit of the pbuf27. Since it is used to provide a fanout of 28, the transistors in the output stage are four times the size those of the regular P-latch.

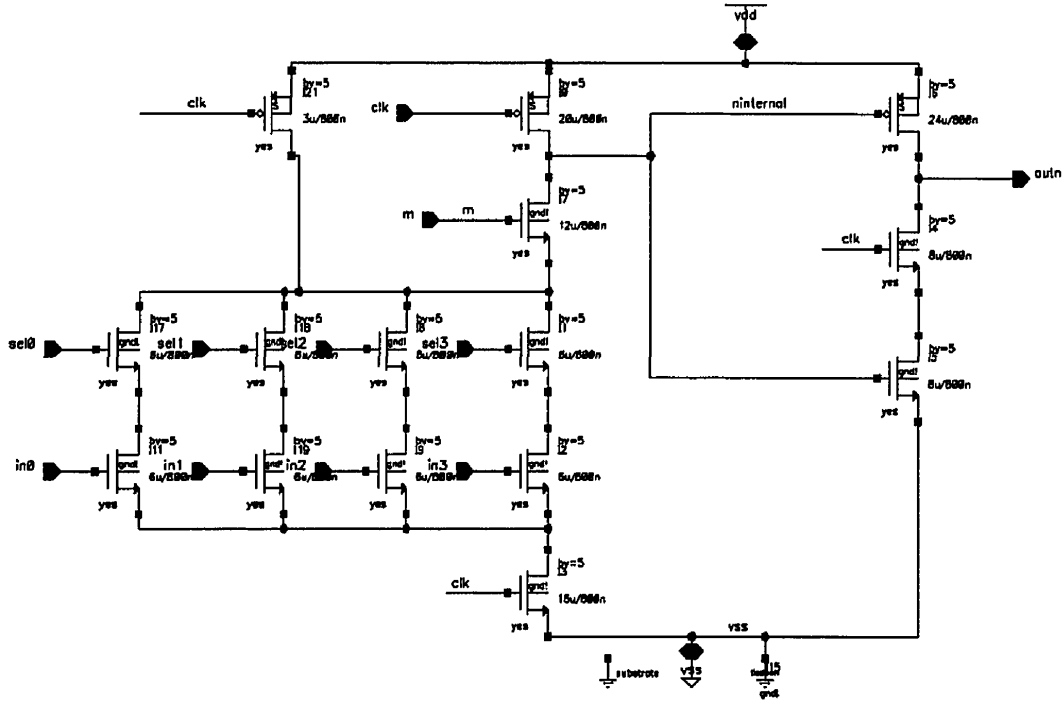


FIGURE 4.15. Circuit of the mux4-1_ntsp_m cell of the block_v

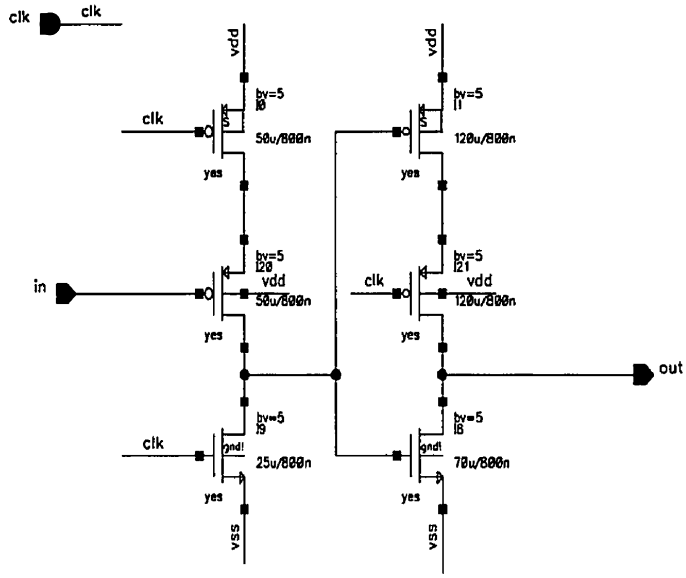


FIGURE 4.16. Circuit of the pbuf27_not of the block_v

The TSPC implementation of all internal blocks except the ROM is straight forward because the logic functions involved in these blocks are simple (e.g. AND and OR logic). Therefore direct translation from a logic gate level circuit to a TSPC transistor level circuit can be done without any complications.

However, the final layout of the control unit was more complex. Even though there were only a few basic cell layouts, the wiring of the control unit was difficult. Figure 4.17 shows the floor plan of the control unit. Since the control unit does not have a regular structure like the filter core, all local and global wiring was done manually. To make the power distribution and the clock distribution easy, all basic cells of the control unit except the ROM were laid out using the same pitch as the biquad filter core bit slice. All basic cells which build up the entire control unit then form multiple distinct slices of basic cells. To save silicon area, the basic cells for every block were placed in such a way that every block fits into its assigned rectangular region.

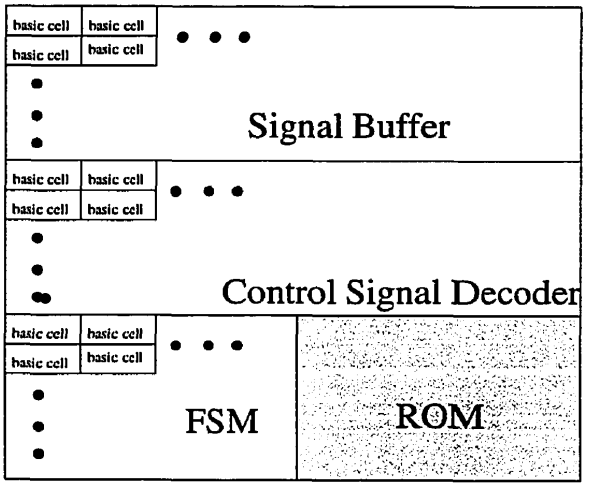


FIGURE 4.17. Floor plan of the control unit

The following sections discuss in detail the TSPC implementation of the FSM, the ROM, the control signal decoder and the control signal buffer.

The logic-gate-level design of the FSM discussed in section 3.3 uses latches, 2-input AND gates, and OR gates. Figure 4.18 shows the floor plan of the FSM and the rough signal interconnect. The AND basic cells which implement the pulse delay lines are placed on the top. The OR basic cells with inverter buffers which drive the word lines of the ROM are placed at the bottom. Therefore, the output signals of the AND cells go down to the OR cells. The buffered output signals of the OR cells are delivered to the ROM on their right horizontally. The following sub-section discusses the design of the OR-gate with inverter buffering.

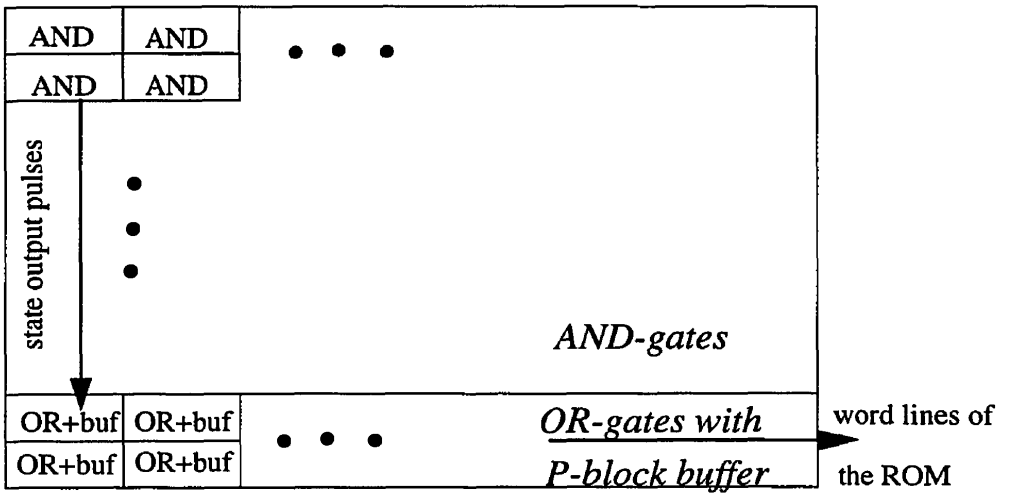


FIGURE 4.18. Floor plan of the FSM

4.3.1.1 OR cell with inverter buffer

The outputs of all OR-gates of the FSM need to drive the word lines of the ROM which have a maximum fanout of 14. Static inverters were inserted into the OR cell to act as buffers. Static inverters were used instead of sizing the P-latches because for the same drive strength, static inverters occupy less area than P-latches using big transistors. Figure 4.19 illustrates the design of the OR cell buffer. Two inverters are inserted into the OR-cell to ensure the logic value of the OR cell is unchanged.

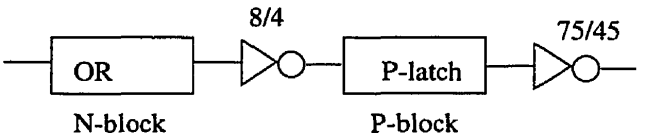


FIGURE 4.19. Design of the OR cell buffer of the FSM

4.3.2 ROM

Figure 4.20 illustrates the circuit of the ROM. As seen in Figure 4.20, the ROM contains two parts: the precharge word slices and the ROM output slices. Since a 14×64 bit ROM is implemented instead of a 27×32 bit ROM, the fourteen 64-bit words are stored in the precharge slices such as *S0, S1,.., S63*. They are accessed by the fourteen word lines such as *A0, A1,.., A13*. The address information is sent from the FSM. If one of the word lines is 1, a 64-bit word is sent to the ROM output slices. The ROM output slices select the corresponding 32-bit instruction word by using the control signals, *sel0* and *sel1*. If an even number instruction is to be selected, *sel0* is 1, otherwise, *sel1* is 1. The *sel0* and *sel1* signals are generated by the FSM shown in section 3.3.

From Figure 4.20, the ROM has a total pipeline delay of two clock cycles. In the first clock cycle, the 64-bit word is produced. In the second clock cycle, the ROM output slices produce the corresponding 32-bit filter instruction. Only a few basic units of the ROM were laid out. They are the ROM output slice and the basic cells which build the precharge word slices such as the clock transistor cells, the "1" cell and the "0" cell. The ROM was compiled with these basic units by the Cadence Structure Compiler [4]. Figure 4.21 shows a layout slice of the compiled ROM.

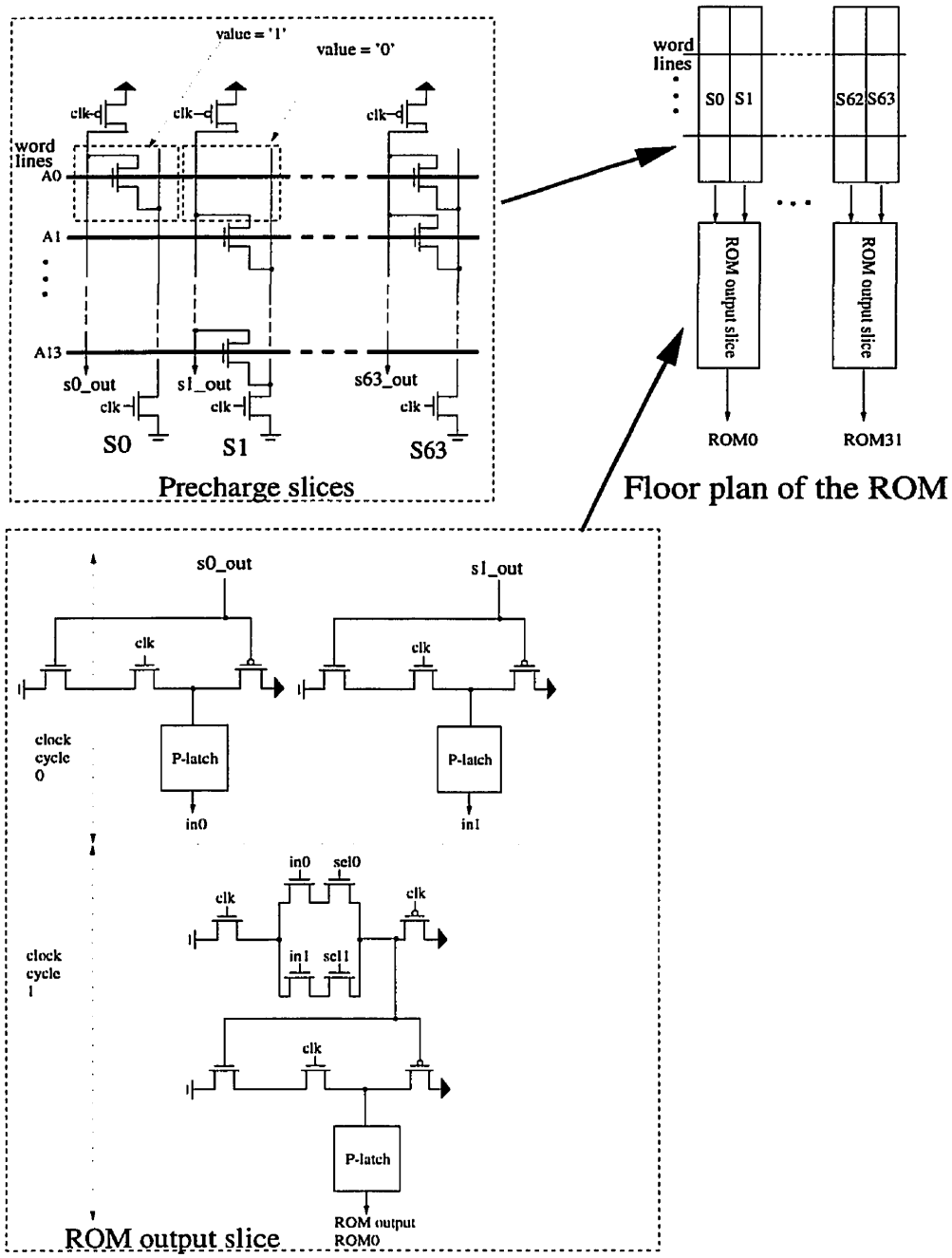


FIGURE 4.20. Circuit and floor plan of the ROM

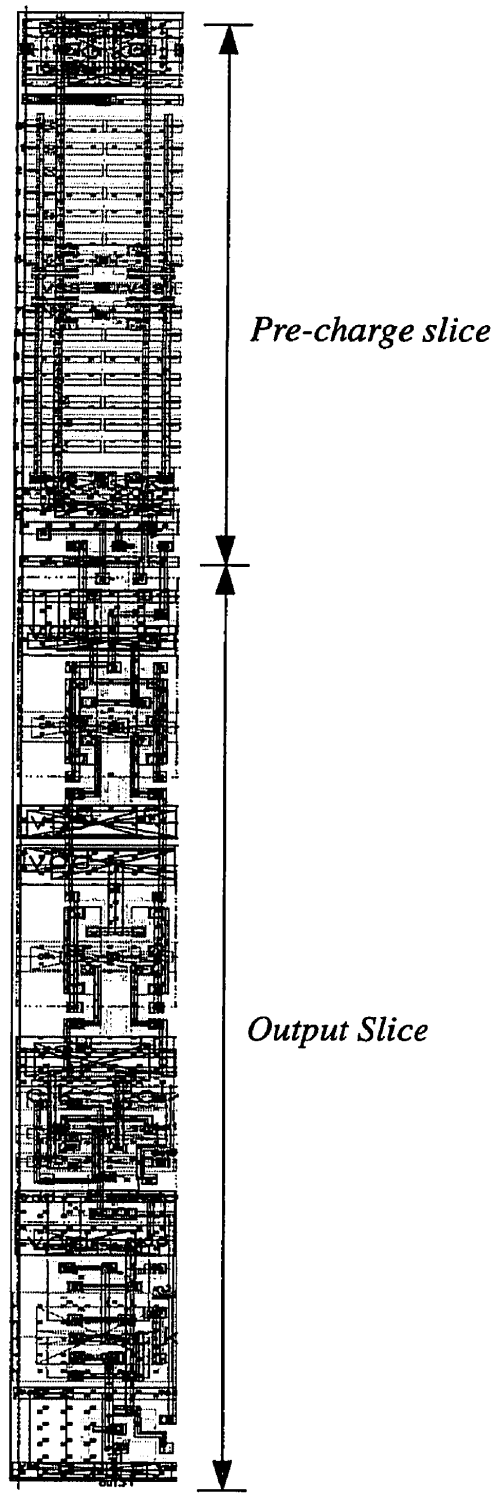


FIGURE 4.21. Slice of the layout of the ROM

4.3.3 Control signal decoder

Figure 4.22 shows the floor plan of the control signal decoder. This floor plan is similar to that of the FSM. The input control signals of the decoder come from the top of the ROM located to the bottom right of the decoder. The decoded signals are sent to the control signal buffer above.

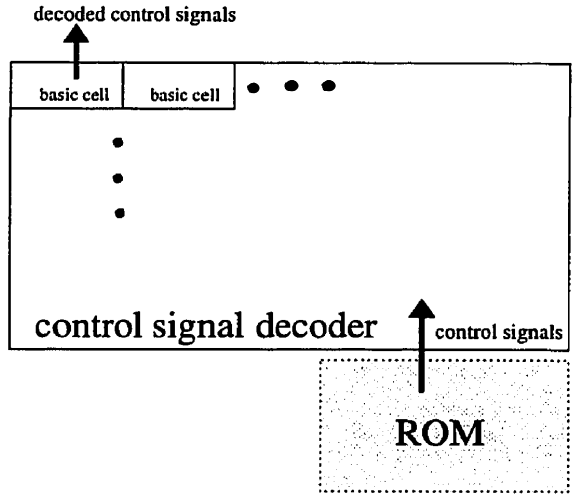


FIGURE 4.22. Floor plan of the control signal decoder

4.3.3.1 Design of the basic cells

Section 3.3 illustrated three types of decoding cells in Figure 3.20 and Figure 3.21. Since the control signal decoder uses only these three types of decoding cells, they can be categorized into:

- Type 1 which decodes the 2-bit select signals used by the ALU input muxes.
- Type 2 which decodes the 2-bit select signals used by the muxregs of the register file.
- Type 3 which produces the feedback signals of registers in the register file.

Figure 4.23 shows the N-P block partitioning of all types of basic cells. As seen in Figure 4.23, the total pipeline latency of the decoder is one clock cycle.

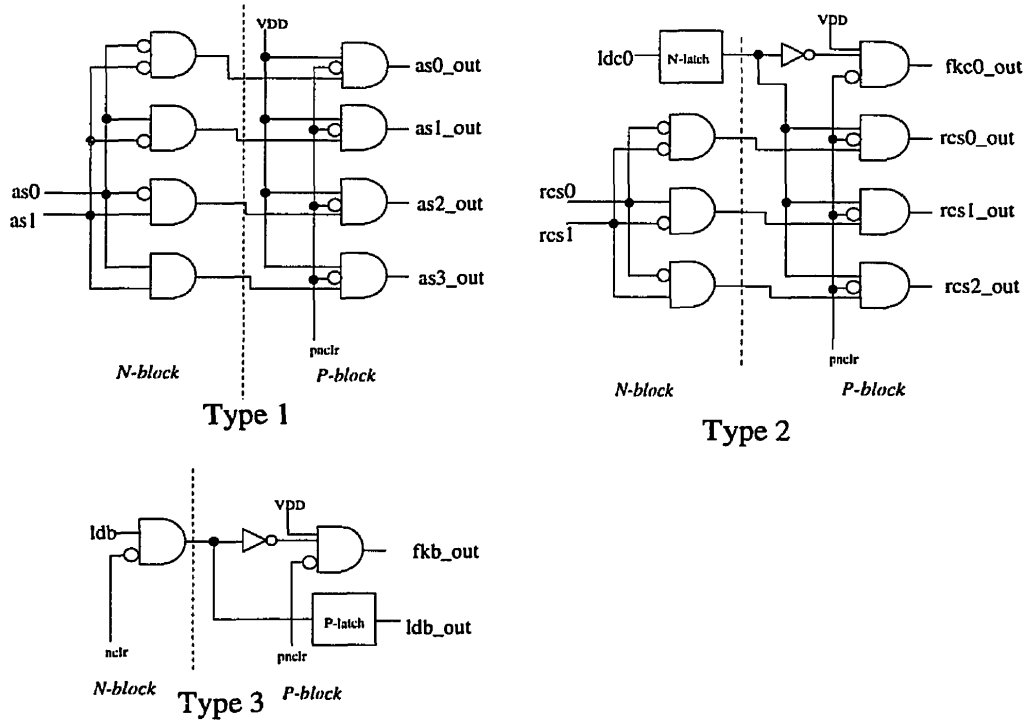


FIGURE 4.23. N-P block partitioning of the basic cells of the control signal decoder

As seen in Figure 4.23, the decoding cells use 2-input N-block AND-gates, 3-input P-block AND-gates, P-latches and N-latches. In order to reuse as many TSPC basic cells as possible, type 1 and type 3 decoding cells use 3-input P-block AND-gates with one input connected to VDD. Since the performance and the area of 2-input P-block AND-gate and 3-input P-block AND-gate are almost the same, only 3-input P-block AND-gate is implemented to save development time.

Except for the 3-input AND-gates, all other TSPC cells have been discussed. Figure 4.24 shows the circuit of the 3-input P-block AND-gate. Since in the P-block of traditional TSPC logic, the 3-input AND-gate is realized by three PMOS transistors connected in parallel. The speed of this 3-input AND-gate is fast enough that there is no need to realize it based on the complicated P-block of all-N TSPC logic. Note that the transistors in the output

stage of the 3-input AND gate are the same as those in the P-latch to ensure that the 3-input AND gate has the same drive capability as the P-latch.

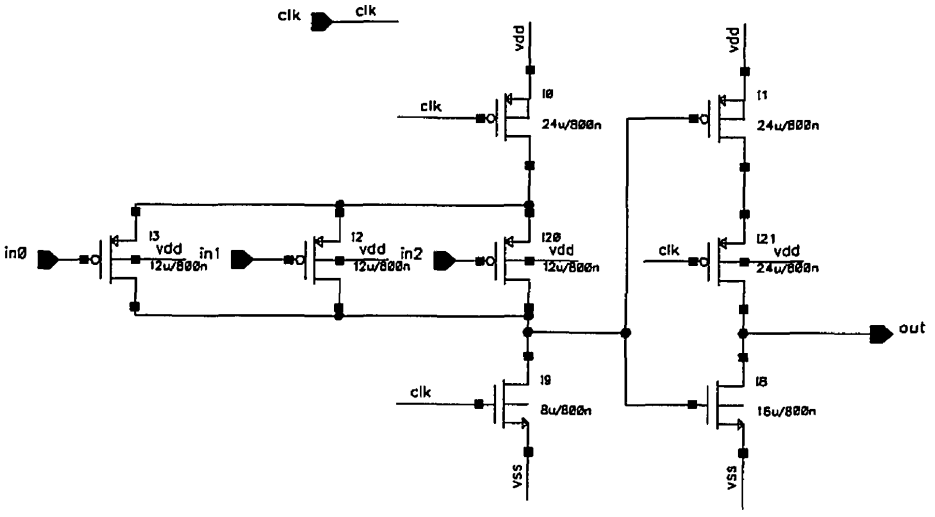


FIGURE 4.24. Circuit of the 3-input P-block AND-gate of the control signal decoder

4.3.4 Control Signal Buffer

The control signal buffer performs the buffering for all control signals decoded by the control signal decoder, and then delivers the buffered signals to the biquad filter core. All control signals can be categorized into four types in terms of their fanout requirements and the number of clock cycle delays. Figure 4.25 shows the design of all buffers.

- Type 1: The load and feedback signals of the register file need buffers with a fanout of 190. For example, the *ldc* signal, which controls the loading of six registers, needs to have a fanout of $28 \times 6 = 168$. A higher fanout buffer is used to buffer *ldc* because of additional load due to wire capacitance.

- Type 2: The load and feedback signals of the basic cells of the ALU need buffers with a fanout of 190 and buffers with different delays. For example, the carry blocks in the ALU are N-block, and they use the load signals directly from the P-block buffer with no delay. However, each P-block demux_ptsp needs the load signals originating from P-latches and going through N-block buffers. Thus, the load signals have to be delayed by 1/2 a clock cycle. Therefore, two types of load signals with different delay requirement are buffered.

For example, the *ldc0* signal needs to drive fifty six 1-bit muxregs ($28 \times 2 = 56$) and the additional load due to wire capacitance.

- Type 4: The load signals and select signals of the 1-bit signal handlers need to have a fanout of one. Therefore, standard N-latches and P-latches are used to act as delays to synchronize them with other control signals.

Figure 4.26 and Figure 4.27 show the circuits of the basic cells of the control signal buffer.

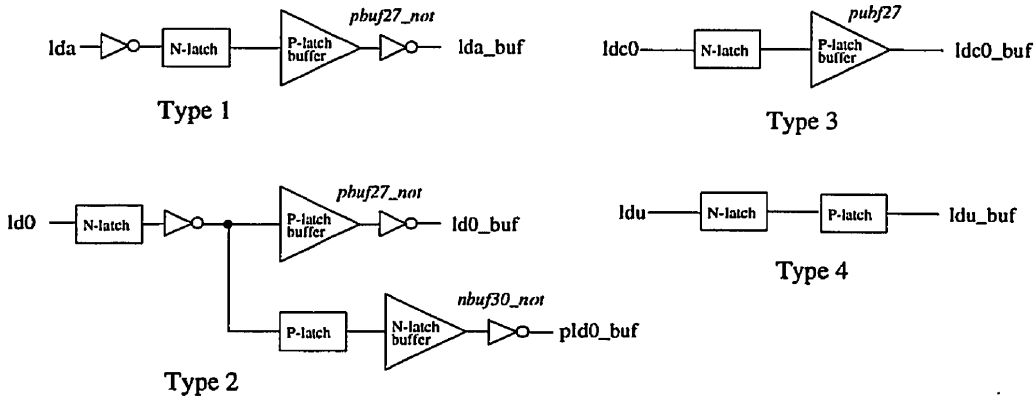


FIGURE 4.25. N-P block partitioning of the basic cells of the control signal buffer

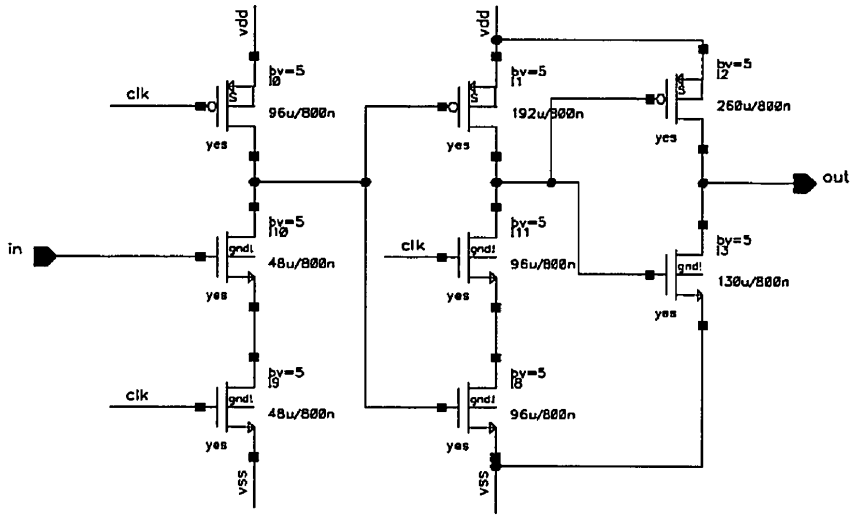


FIGURE 4.26. Circuit of the N-latch buffer with inverter buffering of the control signal buffer

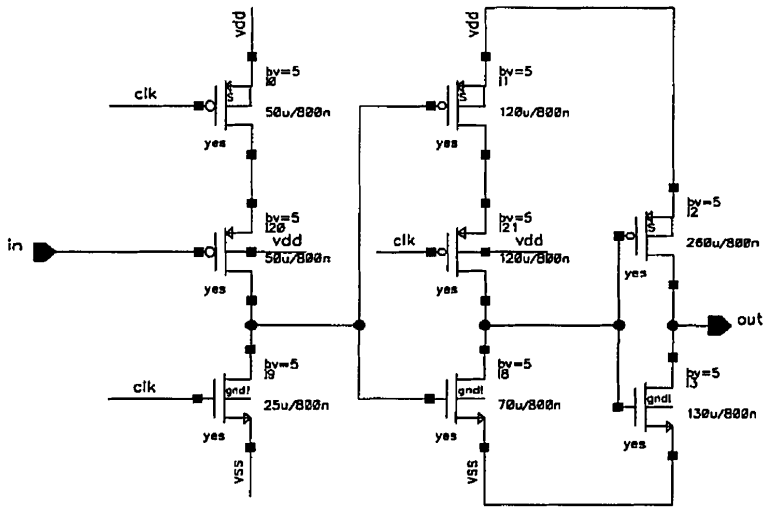


FIGURE 4.27. Circuit of the P-latch buffer with inverter buffering of the control signal buffer

4.4 *Input Block*

The input block is responsible for :

- loading the filter coefficients,
- loading the filter input, *u(n)*, and,
- synchronizing the slow external control signals to the high speed internal TSPC logic.

There are special off-chip control signals which control the operation of the chip such as enabling the input/output, enabling/resetting the filter chip and loading the filter input and the coefficients. The design for the input interface to do all the above functions is discussed in the following sections.

Since the filter coefficients are 28 bits wide, it is impractical to load each of the coefficients in parallel because this will require twenty eight pads. Many input or output pads will increase the total die area. To save more pads, each coefficient is shifted serially into a 28-bit static D flip-flop register (SDFFR) first. The coefficient is then loaded to the first register (muxreg) of the register file from the SDFFR. Similarly, the second coefficient is first loaded into the 28-bit SDFFR serially. Then, it is loaded into the muxreg while the first coefficient is shifted to the second register. The other three coefficients are loaded to the register file in the same way. This method of loading coefficients uses four pads only.

Table 4.2 summarizes the functions of these four pads. Figure 4.28 shows the rough timing diagram of loading coefficients. Figure 4.29 shows the block diagram of the loading mechanism. Figure 4.30 shows the transistor schematic of the SDFFR. The 1-bit DFF was laid out and put into the 1-bit filter core slice. Therefore, there is no need to lay out the entire 28-bit DFF manually as a bit slice approach is used.

Pad name	**Function**
coef	the serial input of the coefficient of the DFF
exsh	the external shift signal (external clock) of the DFF
extld	the external shift (or load) signal of the internal TSPC registers storing the coefficients
iras1 (ras)	the selection signal of the muxreg so that the output of DFF is selected when loading coefficients

TABLE 4.2 Pads responsible for loading filter coefficients

As seen in Figure 4.28 (a), on every rising edge of *exsh*, one bit of the coefficient is loaded into the DFF from the *coef* pad. In Figure 4.28 (b), after one coefficient is loaded into the 28-bit DFF, the *iras1* is set to 1 to make the muxreg of the register file select the DFF output. Then, the rising edge of the *extld* will generate a load pulse *ld_in* (*lda* + *extld*) which triggers the TSPC registers to load the 28-bit coefficient from the DFF.

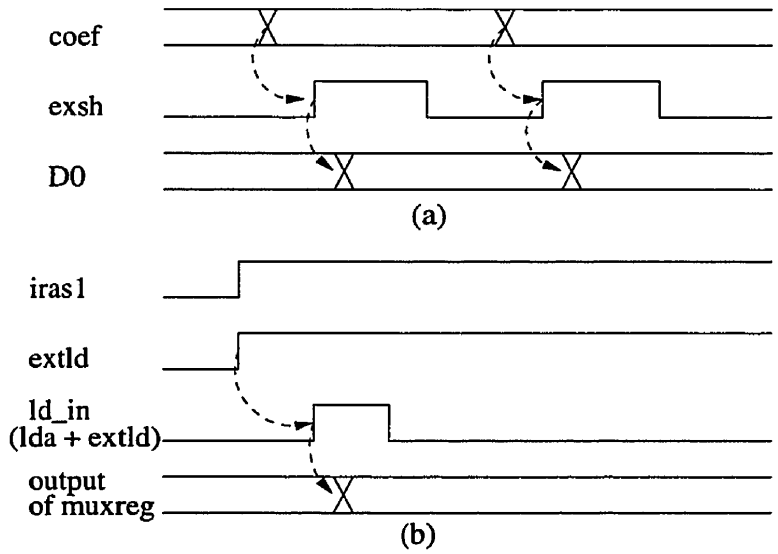


FIGURE 4.28. Rough timing diagram of loading filter coefficients, a) loading into the DFF, b) loading into the TSPC registers from the DFF

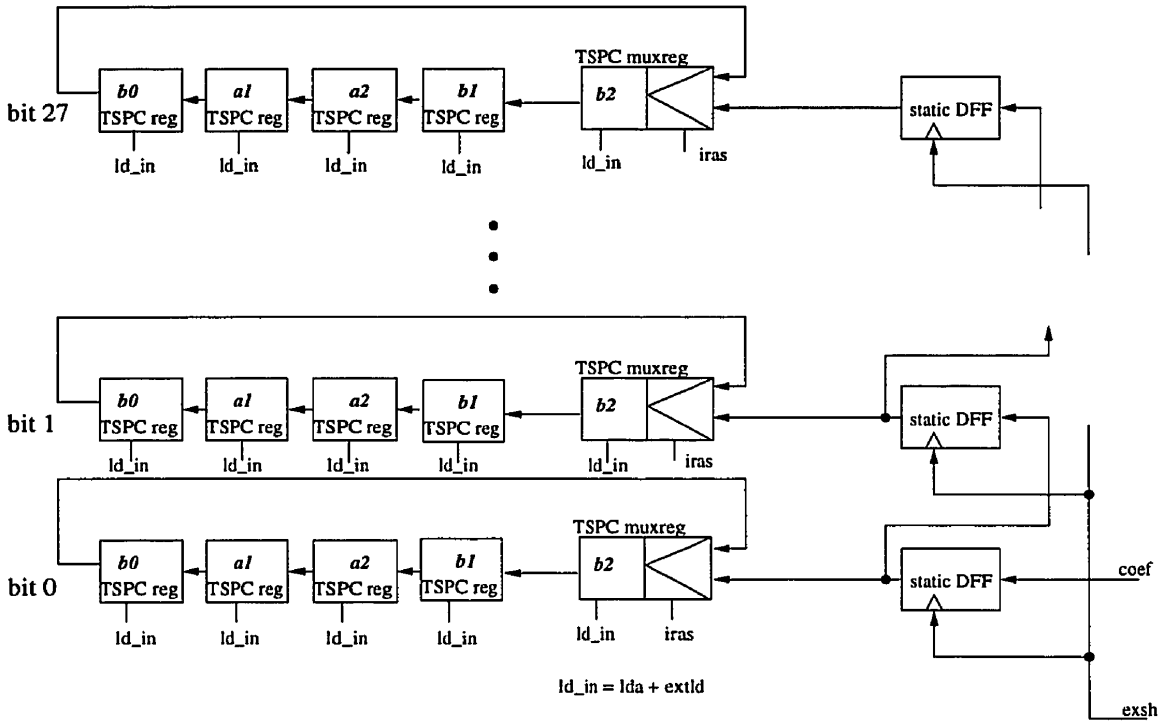


FIGURE 4.29. Design of the loading coefficient logic

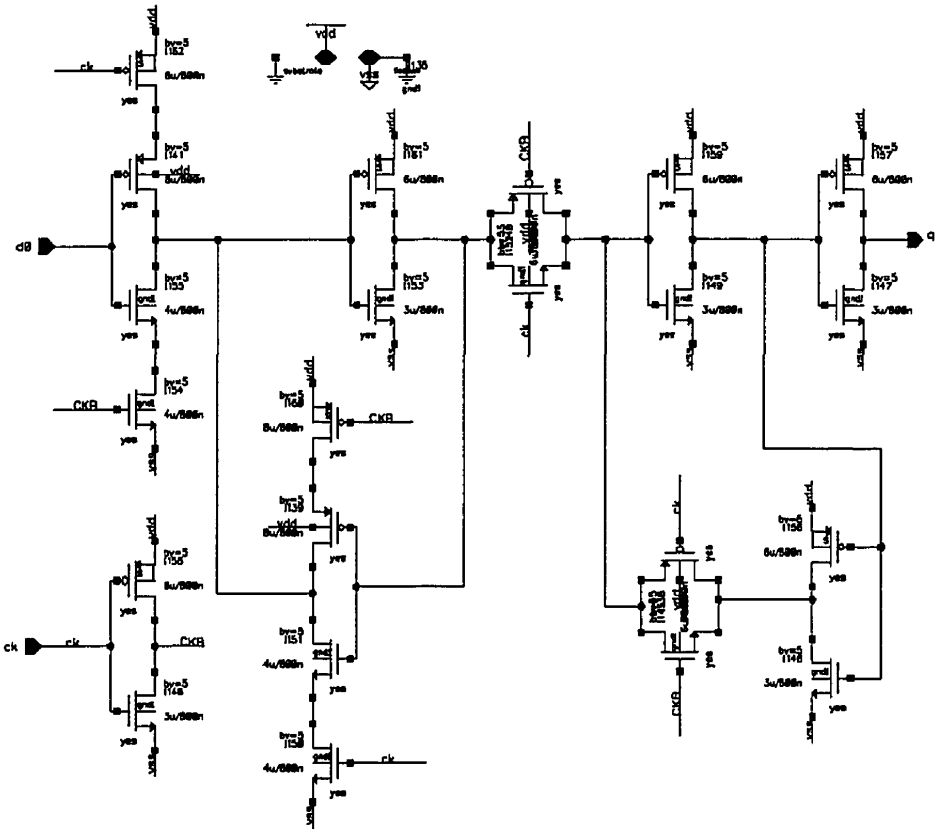


FIGURE 4.30. Schematic of the D-Flip-Flop

4.4.2 Synchronization of the external input signals to the internal TSPC dynamic logic

The biquad filter chip uses a high speed internal clock. Therefore, all external input signals have to be synchronized with the internal clock edge. Since all external input signals are slow compared to the internal clock rate, the slow rising / falling slope signals may introduce synchronization problems when being used by the TSPC logic. Therefore, a series of TSPC latches are used to synchronize slow off-chip signals to the on-chip clock. External control signals like *extld*, *iras1*, *nclr* and *en* are sampled by four pairs of TSPC N-latches and P-latches before being used by the internal logic. The filter input, *iu*, is also sampled by this type of synchronization latch.

The output block is responsible for buffering the outputs to the output pads and providing the busy signal of the filter off-chip. There are only two output signals in the filter chip. They are the 1-bit output and the busy signal. Both output signals are slow compared to the clock signal because both signals change at most at 1/36 of the clock rate. Static inverters of different sizes are used to buffer these signals to the output pads. The input of the output pad has a fanin of three. The last inverter of the output buffer has a fanout of nine which is strong enough to drive the output pad and the wire capacitance.

4.5.1 Busy signal generation

The busy signal has two functions. It indicates when the filter is processing the input. Since the number of clock cycles needed to process a filter input are constant, the busy signal also provides a reference for the clock rate. To implement the busy signal generator, the state 0 output (the initial state) and the state 35 output (the last state) of the FSM are used. It is because these two pulses indicate the beginning and the end of filtering on the input. A fast static SR latch shown in Figure 4.31(a) is used to produce the busy signal by using the state 0 pulse as the *set* signal and the state 35 pulse as the *reset* signal. With these *set* and *reset* signals, the output of the SR latch provides a time reference for 35 clock cycles as well as the time when the filter processes the input. Figure 4.31 shows the schematic of the fast static SR latch and its rough timing information.

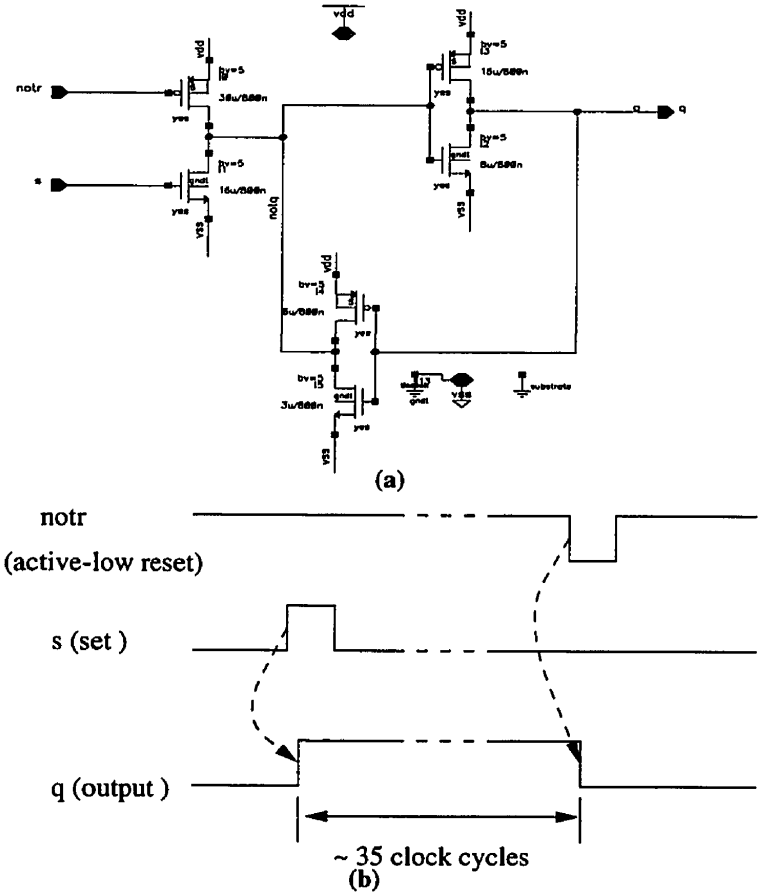


FIGURE 4.31. a) Fast static SR latch, and, b) its timing information

4.6 *Clock Generation and Clock Distribution*

The clock used by the biquad filter chip is generated internally. Using an internal clock has the following advantages over using an external clock:

- an internal clock is less sensitive to the off-chip noise and the noise picked up by the bonding wires and pins than an external clock;

pins and input pads have limited bandwidth, unless specially designed input pins and pads are used. However, using special packaging, or specially designed pins and pads will increase the cost for the chip. Using an internal clock avoids this problem.

In this section, the basic idea of ring oscillators is presented first. The design of the clock generation using a voltage controlled oscillator (VCO) is described next. The design of the delay element used in the VCO follows. Finally the clock distribution scheme and the clock buffer slice are presented at the end.

4.6.1 Ring oscillator

Usually a clock generator is made from a ring oscillator. The design of a ring oscillator is shown in Figure 4.32. The ring oscillator is implemented by an odd number chain of inverters. Since this inverter chain is unstable, the outputs of inverters toggle between 1 and 0 when the power is turned on. The frequency of oscillation in the ring depends on the total inverter delays in the ring.

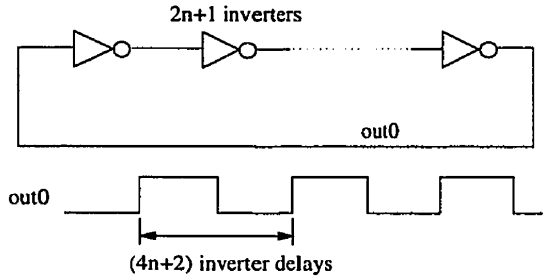


FIGURE 4.32. Ring oscillator

4.6.2 Voltage controlled oscillator (VCO)

The most common variation of the ring oscillator used in clock generation is the voltage controlled oscillator (VCO). The design of the VCO is shown in Figure 4.33. To make the oscillator cover a wide interested frequency range, several taps are added to the ring. This

VCO to have continuously tunable clock frequencies [14].

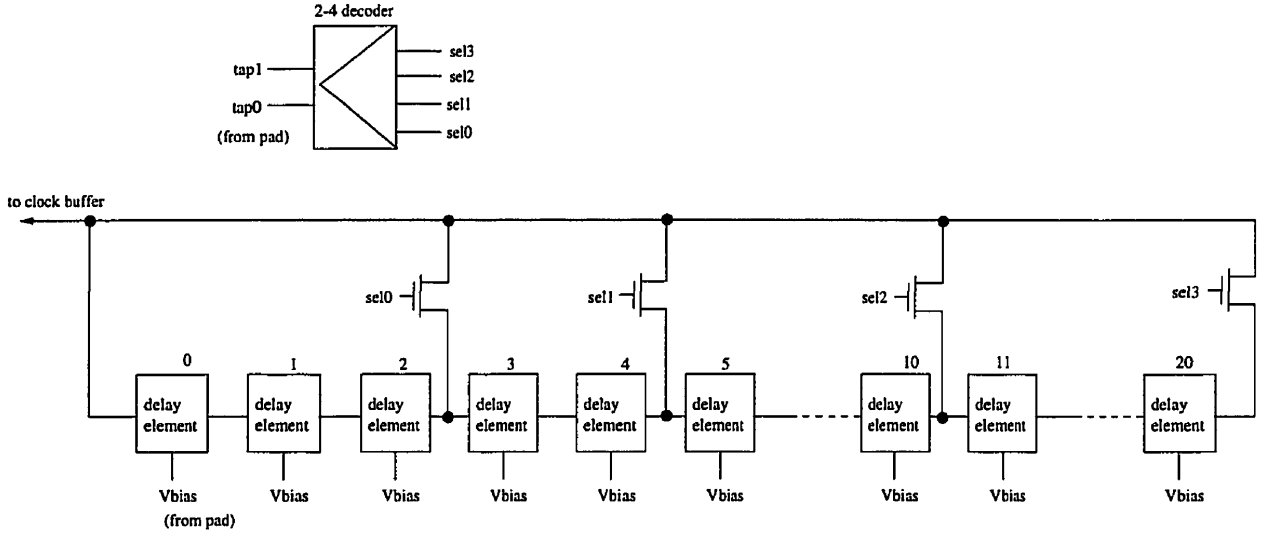


FIGURE 4.33. Voltage controlled oscillator (VCO)

As seen in Figure 4.33, *Vbias* is the external bias voltage to the delay element. *Tap0* and *tap1* are external select signals to configure the number of delay elements in the ring. Simulation results indicate that the VCO with selectable 3, 5, 11, and 21 delay elements covers the frequency range between 79 MHz and 687 MHz.

4.6.2.1 Design of the delay element

The delay element of the VCO contains an inverter, a pass transistor with an external voltage control signal *Vbias* and a MOS capacitor, C_{load}. The delay in the delay element increases as the capacitance of C_{load} increases. Figure 4.34 shows the schematic view of the delay element. Figure 4.35 shows the layout of the delay element.

As seen in Figure 4.34, the output of the inverter is connected to a pass transistor which is attached with a capacitor, C_{load}. The gate of the pass transistor is connected to the *Vbias* signal. The pass transistor acts as a variable resistance controlled by *Vbias*. With this setup, the amount of capacitance seen by the output node of the inverter in the ring is

capacitance seen by the output node of the inverter. Thus, the total delay along the ring can be varied by the voltage of the *Vbias* signal line.

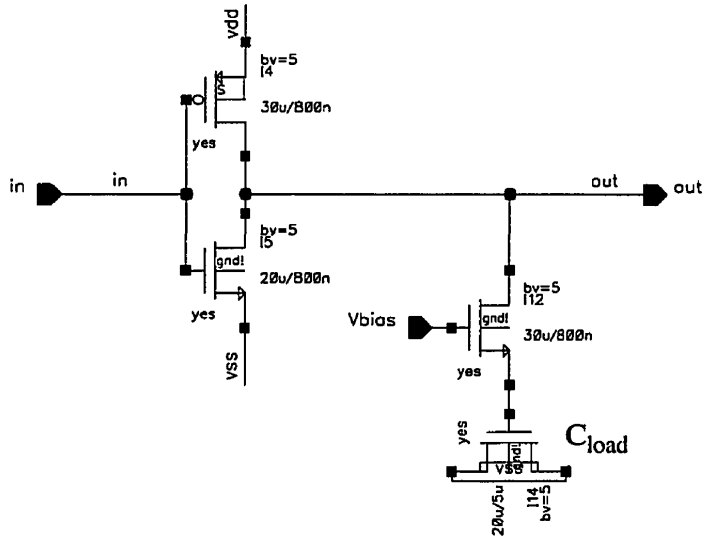


FIGURE 4.34. Schematic of the delay element of the VCO

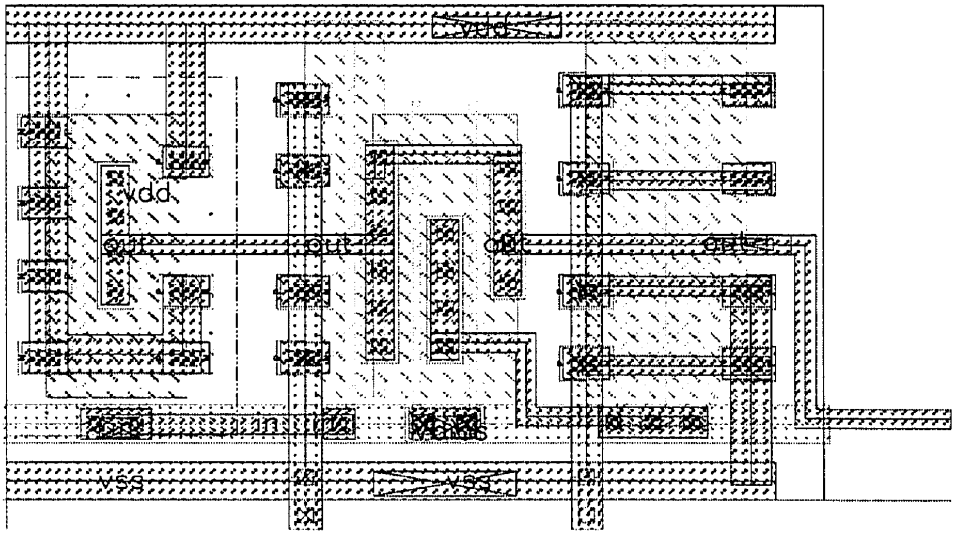


FIGURE 4.35. Layout of the delay element of the VCO

The following sections describe the design of the clock distribution scheme and the local clock buffer.

4.6.3.1 Clock Distribution

Clock distribution is important in the biquad filter because large clock skew will cause errors in the chip. Even though TSPC logic is less sensitive to clock skew [7], [22], minimizing clock skew improves the performance of the filter. Several clock distribution techniques were discussed in [20]. A local clock buffering distribution scheme was used to reduce the skew [20]. A clock tree type clock distribution technique is used to equalize the delay from the master clock to each block of the filter. Shorting bars were used to further reduce the skew which may be caused by device mismatch and differences in the clock loads of the neighboring blocks. Figure 4.36 shows the clock distribution scheme. A regular design approach can be used in the design of the clock buffer to save development time. The design of the clock buffer slice is discussed in the next section.

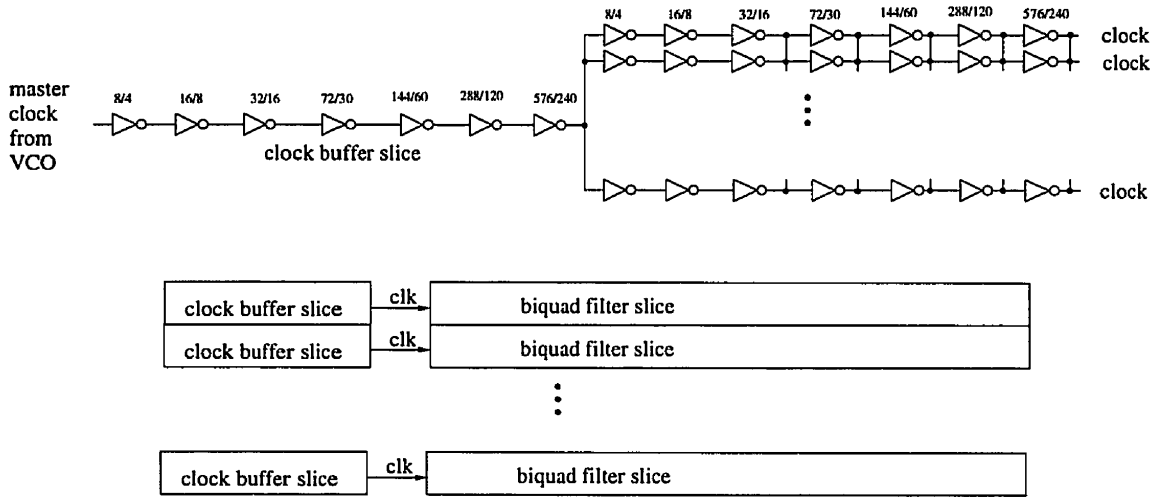


FIGURE 4.36. a) Clock tree, and, b) block diagram of the clock distribution scheme

Figure 4.36 illustrates a regular design approach for the clock buffer slice. The clock buffer slice is designed to handle the worst clock load requirement for all filter slices and the input load of all clock tree branches. The function of the clock buffer slice is to buffer the master clock generated by the VCO with a reasonably sharp output clock slope. Since the entire filter except the ROM were built by slices of basic cells, layout extraction was done on these slices individually. Then, the total capacitances of their clock nodes were found using IRSIM[10]. With these data, the clock buffer is designed to handle the worst clock load among these slices. The bit slice of the biquad filter core has the worst clock load of 3.3 pF.

The performance of TSPC logic degrades if the clock slope is slow. HSPICE simulations [9], [19] were used to size the devices in the clock buffer to improve the output clock slope. The clock buffer gives an output clock slope of 300 ps. Figure 4.37 shows the schematic and the layout of the clock buffer slice. The height of the layout of the clock buffer is the same as that of the filter slices to make the power distribution easy.

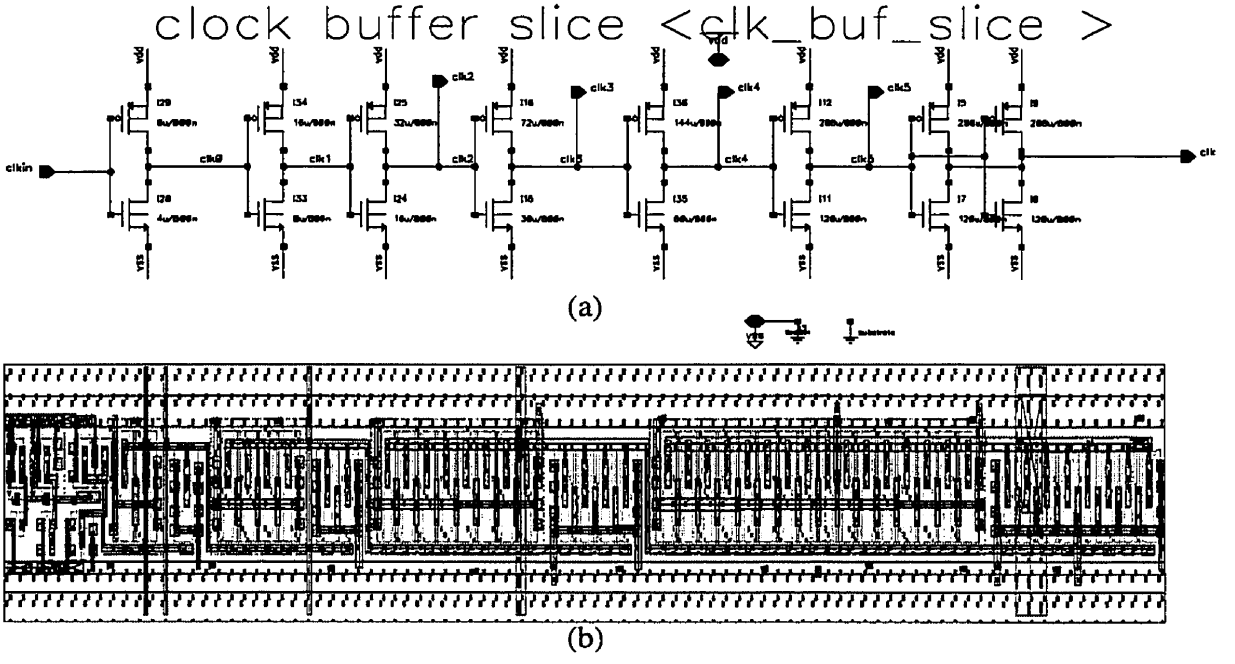


FIGURE 4.37. a) Transistor schematic, and, b) layout of the clock buffer

The power estimation method of the biquad filter is similar to the method of estimating the clock load in section 4.6.3.2. Every extracted slice was simulated using HSPICE to find out its total capacitance and its average current drawn from the power lines (VDD and VSS). IRSIM was used to find out the total capacitance of the entire filter logic [10]. Since different slices have different average current values, the worst case approach was used to size the metal wires for power distribution. Using HSPICE simulations, the slices of the control signal buffer were found to consume the highest average current of 36 mA per slice. Then, the worst case average current per slice is equal to (36 mA + 16 mA) = 52 mA, where 16 mA is the average current drawn for the clock buffer. The average current drawn by the 28 bit biquad filter core is founded to be 0.94 A. The control unit draws an average current of 0.40 A. The clock buffer draws an average current of 1.10 A. The total average current drawn by the filter chip is 0.94 A + 0.40 A + 1.10 A = 2.44 A. The total power of the biquad filter chip is estimated to be 2.44×5 V = 12.2 watts.

The distribution of power is easy because all slices have the same pitch and power comes from the pads of the left and right sides. Figure 4.38 illustrates the power distribution. Twelve pairs of VDD and VSS pads were used to deliver enough power to the biquad filter chip. These twelve pairs of the power supply pads can deliver a maximum average current of 3.2 A to the biquad filter chip, based on the metal width of the power pads. Metal 3 and metal 1 were used to deliver the power to the filter logic with IR drop less than 0.27 V from the VDD/VSS pads.

For the peak current, decoupling capacitance is placed near the power lines to ensure the peak current requirement can be met. The design of the decoupling capacitor is discussed in the next section.

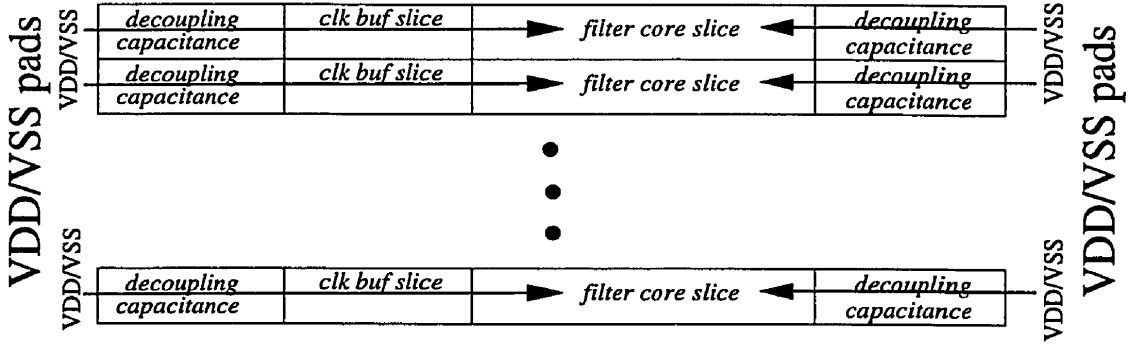


FIGURE 4.38. Power distribution scheme

4.7.1 Decoupling capacitances

Using IRSIM, the total capacitance of the biquad filter was found to be 1014 pF. Generally, decoupling capacitance should be ten times the total capacitance of the chip to ensure good decoupling results. However, the amount of extra area for ten times the total capacitance is bigger than the area occupied by the filter logic. After filling up the empty spaces between the filter logic and pads, the total decoupling capacitance is approximately three times the total capacitance of the filter logic. Since extra VDD/VSS pads were put in the chip to deliver power, these extra power pads together with the decoupling capacitance should be enough to meet the peak current requirement of the chip. Figure 4.39 shows the basic cell of the decoupling capacitance. A regular design approach was used in designing the basic cell of the decoupling capacitance. The basic cell can be replicated horizontally and mirrored vertically with self-connecting contacts.

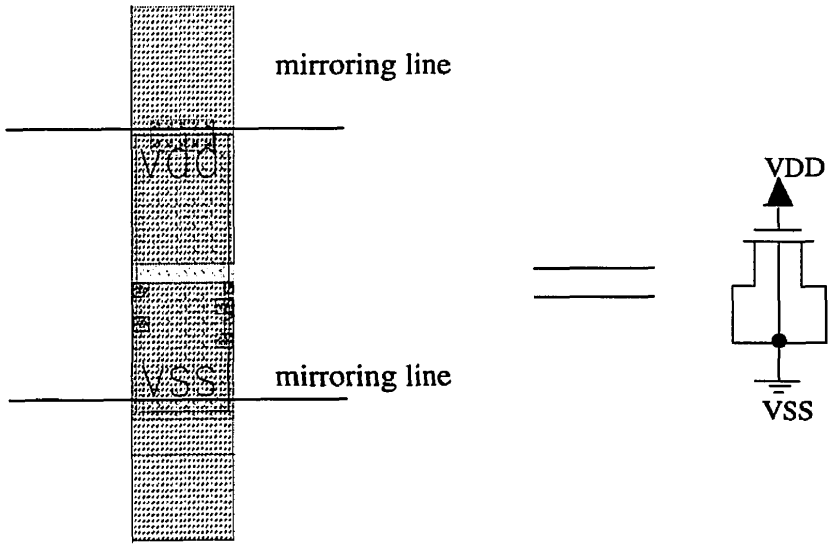


FIGURE 4.39. Layout of the decoupling capacitance basic unit

4.8 *Summary of the Biquad Filter*

The layout of the biquad filter chip occupies an area of $3140 \times 3440\ \mu m^2$. The final layout is very close to the floor plan of the biquad filter chip shown in Figure 4.1. Table 4.3 gives the summary of the chip.

Number of input pins	11
Number of output pins	2
Number of VDD pins	14
Number of VSS pins	14
Number of transistors	39,666
Dimensions	$3140 \times 3440\ \mu m^2$
Average current drawn from VDD (@ 660MHz)	2.44 A
Power supply voltage (DC)	5 V
Maximum internal clock rate	660 MHz

TABLE 4.3 Chip summary

input pins discussed in the previous sections:

- *Vbias* supplies the bias voltage for the VCO, and *tap0* and *tap1* are responsible for selecting the VCO taps.
- *coef*, *exsh*, *extld* and *iras1* control the loading of the filter coefficients.
- *iu* is the 1-bit filter input.

The output pins are:

- *pout* which is the filter output, and,
- *intclk* which is the busy signal.

There are fourteen pairs of VDD and VSS pins. One pair is placed on the top and one pair on the bottom of the filter chip to supply enough power to the input and output pads on the top and bottom. A total of twelve pairs of VDD/VSS pins on the left and right sides distribute power to the chip evenly.

There are only 24,670 transistors in the schematic view of the biquad filter chip but the extracted layout view of the chip has 39,666 transistors. The difference between the two is due to the fact that all large transistors were interdigitated as many small transistors connected in parallel. All gates of the interdigitated transistors were connected using metal to reduce the signal delay in the gate-poly.

4.8.1 Comparison to other Δ–Σ based filter chips

Table 4.4 summarizes the throughput and area of other Δ–Σ based filter chips and the Δ–Σ based biquad filter (TRHBQ) of this thesis. All throughput values are based on simulation results. The throughput here means that the maximum filter input rate can be processed by the filter. Since TRHBQ takes thirty six clock cycles to process each filter input, it has a maximum throughput of 18.3 MHz if it runs at 660 MHz. As seen in Table 4.4, the

mostly on the area. The area of TRHBQ is smaller than the others because only one ALU is used. The difference in area is more obvious when the core area of TRHBQ is compared to the others, because the other chips did not use as many decoupling capacitors. Since the feature sizes of the filters in the references [1] and [14] are 1.2 μm, their areas are normalized to the feature size of 0.8 μm for comparison. Note that [1] is 5th order.

Even though TRHBQ has less throughput than the other Δ–Σ based filter chips, the amount of silicon area occupied by TRHBQ is less than that of others.

Ref.	**Chip description**	**F (μm)**	**Throughput**	**Area (mm^2)**
This work (TRHBQ)	Δ–Σ based biquad filter using one ALU	0.8	18.3 MHz (660 MHz ÷ 36)	Chip: 3.14 × 3.44 = 10.8 Core : 1.640 × 2.618 = 4.29
[1]	Fifth order Δ–Σ based IIR filter (quasi-orthonormal structure)	1.2	45 MHz	Chip: 4.355 × 5.962 = 26.0 Normalized area (to 0.8 μm) = 11.5 Core: 3.396 × 5.133 = 17.4 Normalized area = 7.75
[14]	Δ–Σ based biquad filter with interleaved 12 independent sets of coefficients and data on one datapath	1.2	12 × 27.5 MHz (330 MHz in total)	Chip: 4.300 × 6.100 = 26.2 Normalized area = 11.6 Core : 3.200 × 6.100 = 19.52 Normalized area = 8.68

TABLE 4.4 Comparison of the Δ–Σ based biquad filter chip (TRHBQ) of this thesis to other Δ–Σ based filter chips

4.9 Simulation and Results

Several C programs were written to verify the architecture and the algorithm of the biquad filter. The register transfer level and logic gate level models of the biquad filter were verified using Tortle [15]. HSPICE [9] was used to simulate all transistor level basic cells and slices. The extracted layout views with parasitics of all basic cells and slices were verified using HSPICE..

The core of the biquad filter, excluding the clock generation and the clock buffer, were verified using AWSIM [13]. The AWSIM simulations of the biquad filter were slow. Therefore, the frequency response of the schematic and the extracted layout views of the biquad filter was almost impossible to obtain within a reasonable amount of time. To verify the functionality of the filter core, more than 4000 clock cycles of AWSIM simulations were done on these views. Then, the four ALU output values of the filter core were compared with those of the Tortle and C program models. The ALU outputs of the filter core obtained by AWSIM simulations matched with those of C and Tortle models bit-for-bit.

The biquad filter chip including the clock generation and the clock buffer were simulated using AWSIM. Simulations were done both on the schematic and the extracted layout views. The simulated ALU outputs matched with the ALU outputs of the Tortle and C models bit-for-bit. Since the filter has 36 states, more than 100 sets of 36 states were verified using AWSIM. Therefore, the biquad filter chip is functionally correct. In addition, the Layout-vs-Schematic (LVS) verification tool of Cadence [19] was used to further verify that the layout view matched the schematic view.

HSPICE simulations indicate that all basic cells and slices of the filter work at 660 MHz. Efforts have been made to distribute the clock to minimize the clock skew. The power distribution were done carefully to minimize the IR drop in the power lines. Therefore, the biquad filter chip should work at 660 MHz.

CHAPTER 5 *Conclusions*

This thesis has presented an area efficient and high speed architectural design of a Δ–Σ based biquad filter. The new design approach uses only one simple ALU to perform all arithmetic for the Δ–Σ based biquad filter. The use of a single ALU saves silicon area. TSPC dynamic logic is used in the implementation and enables a high speed operation of 660 MHz. In the implementation, the biquad filter is programmed in thirty-six instructions which are stored in a ROM. An on-chip VCO produces a high speed internal clock, and simple I/O interface circuits synchronize external low speed signals to the high speed on-chip operation.

5.1 Conclusions

The biquad filter described in this thesis uses second order Δ–Σ modulators to modulate multi-bit internal signals to single bit signals. The use of Δ–Σ modulators greatly simplifies the arithmetic involved in filtering. The Δ–Σ based biquad filter processes the input signals at the oversampled rate and thus eliminates the need for decimation filter and interpolation filter when filtering is performed at the Nyquist rate. The principle of Δ–Σ modulation and its application on the biquad filter are presented in Chapter 2.

CSAs instead of traditional adders, and eliminates the delays due to long carry-propagation. A two-level quantization scheme with 4-bit partial carry-propagation is used in the biquad filter which simplifies the filter arithmetic and reduces delay. At the end of Chapter 3, a register transfer level design of the biquad filter using the single ALU architecture is presented.

TSPC dynamic logic is used to implement the biquad filter. With TSPC logic, pipelining of the filter is done at the complex gate level. Therefore, the throughput and the speed of the filter are greatly improved. Chapter 4 presents the TSPC logic implementation of the biquad filter. In addition, Chapter 4 discussed other VLSI design issues of the biquad filter chip. The design issues are the clock generation using VCO, the clock distribution using a local clock buffering scheme, the power distribution and the design of the decoupling capacitances.

The biquad filter chip has been sent to CMC to be fabricated in a 3-layer Metal 0.8 μm BiCMOS process. The chip occupies an area of 3140×3440 μm^2. HSPICE simulations show that all basic cells and slices of the biquad filter chip work at 660 MHz. Tortle simulations and AWSIM simulations show that the biquad filter chip works functionally. Efforts have been made to distribute the clock to minimize the clock skew. The power distribution were done carefully to minimize the IR drop in the power lines. Therefore, the biquad filter chip should work at 660 MHz. At this clock rate, the filter can process input at the maximum sampling rate of 18.3 MHz.

In this thesis, the biquad filter was programmed with thirty-six instructions which are stored in ROM. However, the filter instructions can be stored in RAM or registers so that other types of filters can be programmed. In addition, more RAM or registers can be used in the filter to store more instructions. This allows the realization of more complex filter structures.

Even though the biquad filter uses the two-level quantization scheme with 4-bit partial carry-propagation, other quantization schemes such as a multi-level quantization scheme can be explored to further reduce the quantization noise in the in-band region. Other quantization schemes may be costly in hardware, but hardware efficient algorithms may still exist.

In this thesis, the biquad filter uses the second order Δ–Σ modulators internally. Higher order modulators can be used to get better noise suppression ability. However, the hardware implementation of higher order modulators may be costly.

[1] Dennis Kin-Wah Au, "An Integrated Delta-Sigma Based IIR Filter", M.A.Sc. Thesis, University of Toronto, 1993.

[2] B.P. Brandt and B.A. Wooley, "A 50-MHz multibit sigma-delta modulator for 12-b 2-MHz A/D conversion," *IEEE J. Solid State Circuits*, vol. 26, Dec. 1991, pp 1746-1756.

[3] B.P. Brandt and B.A. Wooley, "A Low-Power, Area-Efficient Digital Filter for Decimation and Interpolation", *IEEE JSSC*, vol. 29, June 1994, pp 679-687.

[4] *Cadence Structural Compiler,* Cadence 1994.

[5] J. C. Candy and G. C. Temes, "Oversampling methods for A/D and D/A conversion," *Oversampling Delta-Sigma Converters*, J. C. Candy and G. C. Temes, Eds, IEEE Press, 1992.

[6] P. Ferguson, Jr. *et al.*, "An 18b 20kHz dual sigma-delta A/D converter," *ISSCC91 Digest of Tech. Papers*, vol. 34, Feb. 1991.

[7] R.X. Gu and M.I. Elmasry, "All-N-Logic High-Speed True-Single-Phase Dynamic Logic", *IEEE JSSC*, February 1996, pp 221-227.

[8] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd ed., Morgan Kaufmann Pub. Inc., 1992.

[9] *HSPICE Manual*, Metasoft, 1989.

[10] *IRSIM User's Manual*, 1995.

[11] D.A. Johns and D.M. Lewis, "Design and Analysis of Delta-Sigma Based IIR Filter", *IEEE Trans. on CAS*, April 1993, pp 233-240.

[12] D.A. Johns and D.M. Lewis, "IIR Filtering on Sigma-Delta Modulated Signals", *IEEE Electronic Letters*, vol. 27, February 14, 1991, pp 307-308.

[13] D.M. Lewis, *Awsim User's Manual,* University of Toronto, 1991.

[14] D.M. Lewis, "Interleaved IIR Filter on Δ–Σ Modulated Signals", *Advanced Signal Processing Algorithm, Architectures & Implementation VI*, SPIE 1996, pp 55-62.

[15] D.M. Lewis, *Tortle User's Manual*, University of Toronto, 1990.

Accumulator in 1.2-um CMOS for Application as a Numerically Controlled Oscillator", *IEEE JSSC*, vol. 28, August 1993, pp 878-886.

[17] Tony Poon, "Implementation of a Pipelined Delta-Sigma Biquad Filter", M.A.Sc. Thesis, University of Toronto, 1994.

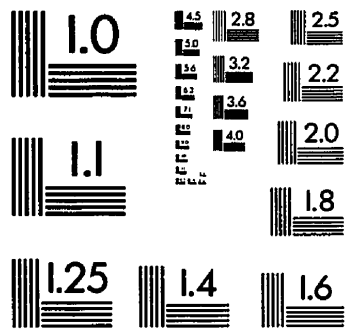
[18] J.G. Proakis and D.G. Manolakis, *Digital Signal Processing : Principles, Algorithms, and Applications*, 2nd ed., Macmillan Publishing Co., 1992.

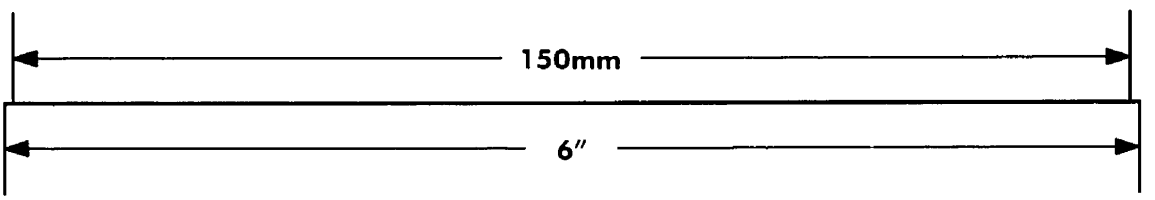
[19] *VLSI User's Manual*, vol. I, & vol. II, University of Toronto, 1995.

[20] N.H. Weste and K. Eshraghian, *Principles of CMOS VLSI design, A Systems Perspective*, 2nd Ed., Addison-Wesley Pub. Co., 1993.

[21] P.W. Wong and R.M. Gray, "FIR Filters with delta-sigma modulation encoding," *IEEE Trans. Acoust., Speech. Signal Processing,* vol. 38, June 1990, pp 979-990.

[22] J. Yuan and C. Svensson, "High-Speed CMOS Circuit Technique", *IEEE JSSC*, February 1989, pp 62-70.





APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989