



Impulse Response (IIR) filters. However, one can easily expand this process to Finite Impulse Response (FIR) filters too.

Most important implementation approaches are as follows: fully-combinational, word-serial, combinational-sequential and bit-serial. In this paper, the above implementations are applied to a sample digital filter. The bit-true models are extracted for each approach and described using Verilog HDL modelling and synthesized for both 0.35  $\mu\text{m}$  CMOS ASIC library and VirtexII Pro FPGA devices. The obtained results show that word-serial approach is the best in terms of gate count reduction (area). From the view point of delay (speed), a fully parallel (combinational) method is the fastest.

This paper is organized as follows: in Section II, finite arithmetic considerations in hardware implementation with emphasis on bit-true model extraction are presented. Different hardware implementation methods with their appropriate architectures are described in Section III. Simulation and synthesis results are explored in Section IV. The paper is concluded in Section V.

## 2 Finite Arithmetic Considerations

DSP algorithms are generally implemented in two major number representations: fixed and floating point. Typically, for computation-intensive applications where numerical precision and dynamic range are critical, designers choose the floating point numbers. In the real world signal processing applications, the additional precision and range provided by floating point is not required. In other word, because of more hardware complexity, floating point implementations tend to be more expensive and power consuming. Therefore, the need to lower design cost and power consumption move the designers to use the fixed point format instead of the floating [1], [7].

A sixth-order low-pass IIR filter is selected as a sample filter in this work. The filter parameters are as follows: attenuation ripple at pass-band ( $R_p$ ) = 1 dB, attenuation at stop-band ( $R_s$ ) = 60 dB, sampling frequency ( $F_s$ ) = 44 KHz, pass-band frequency ( $F_{\text{pass}}$ ) = 3.125 KHz and stop-band frequency ( $F_{\text{stop}}$ ) = 4 kHz.

### 2.1 Quantization of Coefficient

Basically, quantization of filter coefficients causes deviations of poles and zeros positions. Therefore, frequency response of the quantized filter is changed. Generally, the closer the poles are to each other, the greater is the deviation. For this reason,

we break down higher-ordered filters in the form of parallel or cascade combination of biquad filters. Since a biquad filter has a pair of complex conjugate poles and they are far from each other, the effects of coefficient quantization are reduced. Hence, we can achieve the desired frequency response by a smaller word length. In this work, we focus on the parallel form realization.

By comparing the frequency response of the quantized and not-quantized (ideal) filters, we can determine the best coefficient word length that satisfies the filter specification. Figure 2 compares the frequency response of the quantized and ideal models of the sample biquad filter.

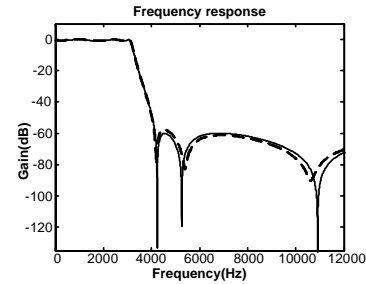


Figure 2: Frequency response comparison of the quantized and ideal filter models.

### 2.2 Intermediate Word-Length Determination

In the finite word-length implementation of digital filters, if the word-length of the intermediate values is not determined properly, output values might be overflowed. This may cause signal degradation and parasitic oscillations. By safe scaling for internal-value word length determination, the overflow can be avoided. In the safe scaled filter, impulse responses from input to all of the internal nodes are calculated and the additional word-lengths are determined as below:

$$\text{Additional word length} = \log_2 \sum_{n=0}^{\infty} |h(n)|$$

where  $h(n)$  is the impulse response of each node. The total word length is equal to the sum of the input and additional bits. However, from the practical point of view, the proper word-length for a filter can be obtained by observing the intermediate signal level swings. Table 1 compares the safe scaling with simulation results for various input data such as voice and multi-tone sine signals.

Table 1: Safe scaling versus practical method.

	Simulation		Safe scaling
	Voice	Multi tone	
Additional Word-length	4 bit	6 bit	8 bit

### 2.3 Precision Adjustment

Based on the obtained results, there is a solution for calculating the signal to noise ratio (SNR) of the quantized filter output. At first, sample input signal is injected to the both of ideal (float) and quantized models. Then, the difference between two outputs is calculated. The ratio of the ideal filter output signal and the difference result is considered as output SNR value for the corresponding input signal. Figure 3 shows a model for this SNR calculation.

Because of the rounding and truncation operations in the quantized filter, precision is reduced in comparison to the floating-point model. This will cause a reduction in SNR of the output signal. For increasing the precision of the intermediate number values, a gain factor of  $G = 2^n$  (as an input gain) is multiplied in the input signal where  $n$  is the additional input and intermediate word-length. This is performed after input quantization step. Therefore, the input quantization noise is maintained while room is provided for further internal precision with the added swing provided from the right (the least-significant bits side). This can lead to equality of output and input SNRs. After a certain limit, extra bit-widening enhancement (added input gain) has no effect on output SNR. Figure 4 plots the output SNR versus number of extra bits for three different input signals.

## 3 Hardware Implementation

Top-down design process starts with system specification analysis, followed by a number of design steps that ultimately lead to logical and structure description of the system components. At the end, design process is completed with integrated circuit design, manufacturing and test [6].

The above process is also applicable to all other DSP systems such as digital filters. In this way, a higher-order filter is partitioned into lower-order parts (filter building-block or biquad) in the cascade or parallel forms. For making a scalable and reusable filter structure, one can implement a biquad building block, instantiate more blocks

based on the first one, configure these blocks using one of the cascade or parallel forms and finally adjust the main filter structure. Therefore, if all of the hardware implementation issues are considered for the biquad filter then the system characteristics are satisfied correctly [6].

When the system is hierarchically partitioned into a set of cooperating processes, these processes are scheduled in such a way that they are mapped onto suitable hardware resources. The purpose of the scheduling is to distribute processes in time so that the required hardware is minimized and the system resource utilization is improved. The most commonly used cost function is the number of computational resources. Several optimization algorithms are deployed for scheduling such as ASAP (As Soon As Possible) and ALAP (As Late As Possible) [6]. The former is used in this work. The starting point for scheduling is driving the computational graph from fully-specified signal flow graph (FS-SFG). In scheduling, because of resource limitations, some intermediate values must be stored until proper operands become available for them. Hence, some delay elements are inserted in scheduling diagram to fit the sequence of operations in time (also called shimming delay elements) [6].

### 3.1 Fully Combinational Implementation

In this approach, computational modules such as full-adders and multipliers are all implemented in parallel. There is no resource limitation in this method. Functional elements are fully combinational and have signed attributes. Because of the hardware design constraints (area and delay), in this work, array multipliers and carry look-ahead adders are selected for multiplication and addition, respectively. Two's complement number representation is used to better process the signed digits. For signed multiplication, array multiplier is optimized with the *Modified Baugh-Wooley algorithm*.

Based on the extracted data bus and coefficient width, the array multipliers are selected. Each of them has two input operands in 2's complement format: data (20 bits) and coefficient (14 bits). The multiplier outputs are 34-bit signed values.

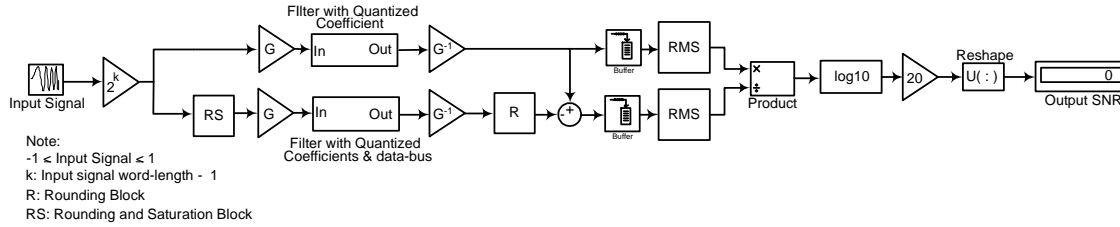


Figure 3: SNR calculation model.

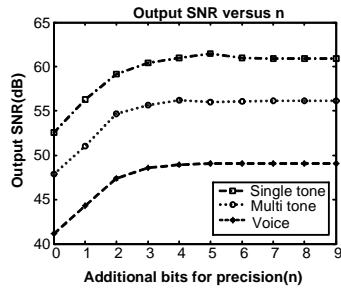


Figure 4: Output SNR versus number of extra bits for various input signals.

However, based on the filter structure, only 20-bit valid data can be passed to the next step (through the 20-bit data bus). To do this, one rounding method must be applied to build the 20-bit proper intermediate data (fixed-point arithmetic must be considered). Finally, our hardware module will be compared with the bit-true model extracted from MATLAB; one compatible rounding strategy is suitable here: rounding to the nearest integer number. So, we look at the lower order bits of the 34-bit result and throw them away if they are less than 0.5 (in fixed-point, if only one bit is 1, the whole lower ordered part must be thrown away). To have a correct addition and because of the filter's iterative structure, adder outputs need to be corrected when they are increased alternatively (to avoid from wrap around phenomena in 2's-complement systems). For this, saturation is applied to the adder's outputs when it is reached to the maximum positive or negative values.

### 3.2 Word-Serial Implementation

In this method, an assumption for hardware resources must be considered: there is only one adder with an extra controller that controls all the additions and multiplications in a sequential manner. For this purpose, sequential multiplication (shift-and-add) methodology is deployed. One extra controller is combined with serial adder to construct the sequential multiplication. The multiplication of two 2's-complement numbers,  $y = a * x$ , requires the formation of the partial bit-products by multiplying the coefficient,  $a$ , by the bits of  $x$ . The partial products then are added

together with the proper shifted (weighted) values. One can reduce the multiplication cycles using higher radix multiplication, i.e. radix-4. For signed multiplication, the Booth recoding algorithm is used to construct signed partial products [6], [7]. In this structure, because of a single adder, only one operation (addition or multiplication) can be performed in a single time slot, so total sample processing time is increased according to the input bit-number ( $n$ ), number of operations, and their dependencies. By this multiplication scheme, 20-bit multiplication, with radix-4 Booth recoding, consumes 13-clock cycles, including one for resetting, ten for partial products additions, one for final addition and one for reading the output result. The addition only takes one clock cycle. Based on extracted clock counts, the final scheduling for this method can be performed. Number of shimming delays is reduced because of the computational resource limitations. Because final result is in parallel form, correction phase (rounding and saturation) is the same as fully-combinational, i.e. rounding to the nearest integer plus saturation.

### 3.3 Combinational-Sequential Implementation

Considering the combinational and sequential functionalities, one can use third approach that partially supports both: the combinational-sequential approach. In this approach, we combine characteristics of the fully-combinational and word-serial methods together. In the other words, the basic functional blocks (adder and multiplier) are implemented in a fully-combinational way (i.e., *Modified Baugh-Wooly* array multiplier and Carry Look-Ahead adder); while one multiplier, one rounding module and one adder are utilized for the biquad blocks (as shown in Figure 5). In the combinational logic, rounding is applied to the output of the multiplier. The rounded result constitutes an input to the adder. The adder output is saturated structurally. Each clock cycle, one multiplication with rounding and one addition with saturation can be performed.

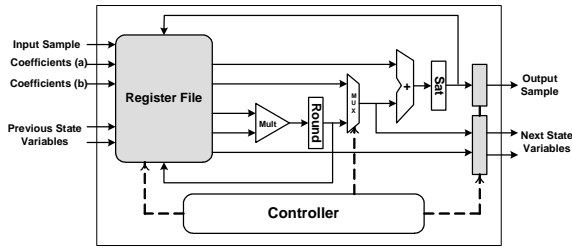


Figure 5: Combinational-Sequential hardware block diagram.

For the sequential part, the system controller is designed as a Finite State Machine (FSM). The state of the controller is changed at each clock cycle. At each state transition, one operation is performed. The operations of each biquad are scheduled in six states (as shown in Figure 6).

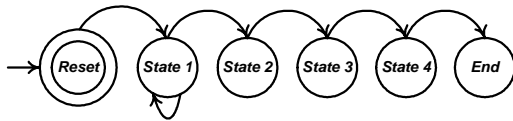


Figure 6: FSM diagram of the combinational-sequential approach.

After resource allocation and mapping, a fully-specified signal flow graph (FS-SFG) is obtained. Then, scheduling on the graph is done such that only one multiplication and one addition can be performed at each clock cycle (state transition). As shown in Figure 7, initial values are  $x(n)$ ,  $v1(n-1)$  and  $v2(n-1)$  where they indicate input data and register outputs respectively. We have also called  $v1$  and  $v2$  the state variables.

As shown, the values of one stage are passed to the multiplier or adder of the next stage with the proper coefficient values. Weighted lines in the Figure 7 represent the amounts of the required shimming delays (15 delay elements in this case). Multiplication by  $a(1)$  is done by left shift operation. After six state transitions, final output result plus the state variables for the next input samples are generated.

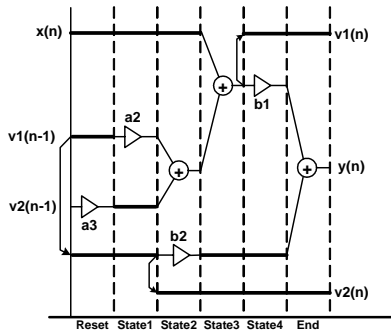


Figure 7: Scheduling diagram of the combinational-sequential approach.

### 3.4 Bit-Serial Implementation

A major advantage of bit-serial implementation over bit-parallel is that it significantly reduces the chip area. Two's complement number representation system is suitable for this type of implementation [6]. In this approach, each processing element is broken down into the bit-sliced functional blocks (e.g. 1-bit serial adder). The input data arrive in parallel form, serialized with a simple parallel-to-serial shift register and injected to the filter at every clock cycle bit by bit. This implementation requires much more clock cycles for data propagation and processing. For each  $n$ -input serial multiplier,  $2*n$  clock ticks are required:  $n$  for LSB and additional  $n$  cycles for MSB part. In unsigned multiplication, for additional  $n$ -clock cycles, 0 is injected to push the remaining results out. In signed multiplication, the sign value is injected instead of 0. This will cause input to be sign-extended. In addition, 1 clock cycle for 1-bit operation is consumed.

Because after each multiplication, all of the 34-bit results are prepared serially, output must be rounded correctly. Each operation must be performed to the correctly-ordered bits. One solution is disabling the next stage storage elements to discard the inappropriate bits and accept the others. A controller is designed to keep the data bit-order properly. The load signals for each storage element (shift register) are controlled by the controller. Figure 8 shows the block diagram of the scheduled bit-serial implementation of a sample biquad filter.

As shown in Figure 8, five various-length shift registers (SR) with controllable load, shift, and clock signals are used.

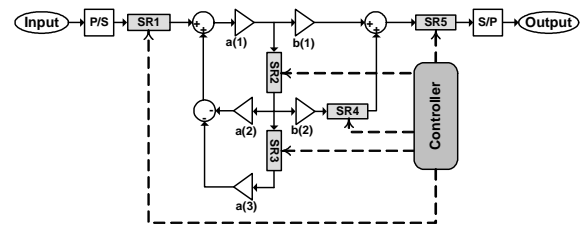


Figure 8: Bit-serial implementation block diagram.

## 4 Experimental Results

After golden-model extraction for the sample biquad filter, essential hardware implementation parameters are obtained for design optimization. After word-length considerations, sample biquad filter is modelled for the four different methods using Verilog HDL. Then, we have synthesized these models for 0.35 $\mu$ m CMOS ASIC library and

VirtexII Pro FPGA devices. The obtained results are shown in Tables 2 and 3. The throughput of each method is calculated as Maximum Clock Frequency divided by Sample Clock Count. This parameter presents maximum sample processing rate of the filter.

A self-checking test-bench is written which reads data vectors from the bit-true model, applies it to the biquad hardware model and compares the output data with the output data of the MATLAB bit-true model.

## 5 Conclusion

Different hardware implementation methods for digital filters plus a survey on the design and implementation process are explored in this paper. For the first step, bit-true model of a sample second-order biquad filter is extracted according to the finite arithmetic effects. Based on this model and extracted parameters, biquad filter is implemented for four different methods: fully-combinational, word-serial, combinational-sequential and bit-serial. All of these methods are described in Verilog HDL and synthesized for both 0.35 CMOS ASIC library and VirtexII Pro FPGA devices. Experimental results show that the word-serial method is the best in terms of gate area complexity. In order to reduce the delay, fully-combinational approach is in the best place. The other two methods are tending to balance the trade-off between hardware area and delay complexities.

## References

[1] A. V. Openheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, Prentice Hall, 1999.

[2] N. Benvenuto, M. Marchesi, and A. Uncini, "Results on the application of simulated annealing algorithms for the design of digital filters with powers-of-two coefficients," *Proc. ICASSP '90* vol. 3, pp. 1301-1304, April 1990.

[3] Tracy C. Denk, Keshab and K. Parhi, "Exhaustive scheduling and retiming of digital signal processing systems," *IEEE Transactions on Circuits and Systems -II: Analog and Digital Signal Processing*, vol. 45, no. 7, Jul. 1998.

[4] R. W. Mehler, Dian Zhou, "Architectural synthesis of finite impulse response digital filters," *15th Symposium on Integrated Circuits and Systems Design*, pp. 20-25, Sep. 2002.

[5] J.X. Hao and G. Li, "An improved stability measure for digital filter implementation," *Proc. ICASSP*, 2003.

[6] L. Wanhammer, *DSP Integrated Circuits*, New York: Academic Press, 1999.

[7] B. Parhami, *Computer Arithmetic Algorithms and Hardware Designs*, Oxford University Press, 1999.

[8] C. Pradabpet, and S. Yimman, "Design and implementation of biquad digital filter," *Proc. 9th Asia-Pacific Conference on Communications (APCC)*, vol. 3, Sep. 2003.

[9] C. Ditzgen, and J. Haight, "A parameterizable biquad block for IIR filters in ASICs: implementations and motivations," *Proc. ASIC Seminar and Exhibit*, pp. P7/4.1-P7/4.4, Sep.1990.

[10] H. Hollisaz, N. Moezzi-Madani, and S. M. Fakhraie, "A quantitative approach to digital filter implementation", *Proc. International Conference on Microelectronics*, Pakistan, 2005.

Table 2: FPGA synthesis results.

Implementation method	Function Generators	CLB Slices	Dffs or Latches
Fully-Combinational	4838	2419	38
Word-Serial	1228	614	151
Combinational-Sequential	64	32	246
Bit-Serial	336	264	368

Table 3: ASIC synthesis results.

Implementation method	Gate Count (Area)	Maximum Clock Frequency (MHz)	Sample Clock Count	Throughput(MSample/s)
Fully-Combinational	5465	25.1	1	25.1
Word-Serial	1141	54.1	13	4.16
Combinational-Sequential	3824	90.5	7	12.92
Bit-Serial	4270	74.5	90	0.828