High-Throughput Digital IIR Filter Design
Zhao Wang*, Chiu-Wei Pan, Yuanchen Song, Carl Sechen*
The University of Texas at Dallas, TX, 75080, U.S.A
*wangzhao1984@hotmail.com; *4carl.sechen@utdallas.edu

Abstract—We have implemented a new, very high throughput digital IIR filter architecture that operates at 600 MHz using the IBM 130 nm process at 1.2 V. The inverter fanout-of-four (FO4) delay for the IBM process is 64 ps. The IIR filter’s worst-case stage delay is therefore 26.2 FO4 delays. This is a substantially higher throughput compared to the fastest previously reported IIR filter implementations. The operating frequency was measured using the industry-standard static timing analysis tool (PrimeTime) on a 3D extraction of the complete IIR filter layout. Critical to this achievement is the use of Generalized Carry-Save (GCS) arithmetic, a pipelined IIR filter implementation and a novel implementation of the quantizer. This is the first design that moves the quantizer (limiter) out of the usual final pipeline stage and into its own separate pipeline stage. In previous digital IIR filter designs, the quantizer had to be included in one of the computational stages and hence was a major factor that limited the throughput of the filter. Maximizing the throughput of digital filters is critical today due to the need for low power circuit designs. Even if a lower throughput is desired, first maximizing the throughput and then lowering the power supply voltage to just meet the throughput requirement yields the lowest dynamic energy implementation as the dynamic energy is quadratically related to the supply voltage.

Keywords—High Throughput; Digital IIR Filter; GCS Arithmetic; Compressor; Quantizer

I. INTRODUCTION

Digital filters can be partitioned into two subclasses: finite impulse filters (FIR) and infinite impulse response (IIR) filters [1]. The impulse response of an FIR filter is finite in duration, and involves no feedback. The response of an Nth-order FIR filter is given by:

\[ y(n) = \sum_{i=0}^{N} c_i x(n-i) \]  

(1)

The input signal is \( x(n) \), the output signal is \( y(n) \) and \( c_i \) are the filter coefficients. \( N \) is known as the filter order.

IIR digital filters have feedback paths that can sustain an impulse response indefinitely. The basic equation of an IIR digital filter can be written as:

\[ y(n) = \sum_{i=0}^{M} b_i x(n-i) - \sum_{j=0}^{N} a_j y(n-j) \]  

(2)

The input to output transfer function for (2) in the \( z \)-domain is given by:

\[ H(z) = \frac{\sum_{i=0}^{M} b_i z^{-i}}{1 + \sum_{j=0}^{N} a_j z^{-j}} \]  

(3)

Compared with FIR filters, IIR filters generally require smaller circuit area and provide higher computational efficiency for specific filtering functions [2]. However, this comes at a price in that the IIR filter has feedback whereas the FIR filter does not. In the absence of feedback, it is generally possible to include additional pipelining to further increase the throughput. However, with feedback, adding pipeline stages does not necessarily increase the throughput since new input operands cannot be issued until the feed-back output values (depending on the previous input operands) reach the input.

Very high throughput digital filters are of strong interest today due to the need for low power designs. Even if a lower throughput is all that is needed, first maximizing the throughput and then lowering the power supply voltage to just meet the throughput requirement yields the lowest dynamic energy implementation since the dynamic energy is quadratically related to the supply voltage whereas the delay is linearly related to the supply voltage (when operating at supply voltages sufficiently above the threshold voltage).
Prior work has addressed the design of fast IIR digital filters [2-4]. This work was primarily directed at modifying the transfer function $H(z)$ in (3) to obtain higher levels of pipelining. The main transformations that improved the IIR filter efficiency are the clustered look-ahead (CLA) transformation [3], the scattered look-ahead (SLA) transformation [4] and the generalized CLA (GCLA) transformation [2]. In [3], the CLA transformation allows a higher degree of pipelining (smaller pipeline stage delays) and hence a higher throughput, but at the cost of introducing stability issues. The transformation in [4] was proposed to avoid these stability issues but it came with a price [2]. First, the number of pipeline stages must be a power of two. Second, it requires one more multiplier/accumulator when the number of pipeline stages is less than eight. An improved transformation (GCLA), proposed in [4], addresses the drawbacks in both [3] and [4], while providing an enhanced throughput. In [2], implementation concepts for both the original IIR structure and the transformed GCLA structure are discussed.

There is a substantial drawback with applying transformations to the original filter equation. In particular, transformations such as those in [2-4] increase the number of terms in the equation, resulting in substantial additional circuit implementation complexity, and correspondingly higher power. Moreover, these transformations change the coefficients and thus it is likely that the coefficients will become more complicated (more “1” bits, resulting in more partial products) compared to those provided in the original design. Therefore we did not employ these transformations in our new IIR filter design. Instead, we sought to significantly improve the throughput while utilizing the original (provided) coefficients and minimizing the circuit implementation complexity. We should note, however, that [2] described an improved high throughput IIR filter structure that avoids transformation of the coefficients and in fact we utilized this architecture as our starting point (details are in Section 3).

The design and implementation of the quantization component is also very important in the IIR digital filter design. For digital filters with feedback, fixed precision (and usually 2’s complement arithmetic) is used to store the various intermediate and final results in the network since this greatly saves area and power versus a mantissa and exponent approach. Given the fixed number of bits used to store the fed-back output value, should output overflow (underflow) occur, the value will have both a changed sign and a significantly altered magnitude relative to the correct output and hence will cause the filter to become unstable. This is typically solved with a limiter or quantizer at the output. Should the output value exceed the upper limit, the quantizer effectively sets the output to the upper limit (similarly in the case of an underflow, the quantizer sets the output to the lower limit). However, comparing the computed output value to the upper limit (and lower limit) requires appreciable delay. Simply adding the quantizer to the final pipeline stage of the IIR filter has been the best solution reported up to now [5]; however, this causes the final stage delay to be substantially dominant and hence limits the operating frequency of the filter.

Another challenging problem with the quantizer occurs when the output is retained in redundant form (i.e., as two vectors, so-called carry and save, which when summed results in the true binary output). A fast method of determining whether the sum of the redundant vectors in fact exceeds the upper limit of the quantizer (or falls below the lower limit) has proved elusive until now.

In order to overcome these difficulties, we developed a novel IIR filter architecture that includes a quantizer. The new IIR filter was implemented using Generalized Carry Save (GCS) arithmetic [6] which systematically seeks to avoid any rippling of carries in a signal-processing network of additions and multiplications. The carry operation is serial and thus certain mathematical operations cannot be done in parallel. GCS arithmetic enables nominally bit-wise serial mathematical operations, such as addition and multiplication, to be computed bit-wise parallel. Power efficiency was enhanced through the use of a typical industrial discrete standard cell library along with a near optimal cell-sizing tool.

The new digital IIR filter operates at 600 MHz using the IBM 130 nm process at 1.2 V. The clock period of 1674 ps, relative to the inverter fanout-of-four (FO4) delay of 64 ps for the IBM process, means that the worst-case stage delay for the IIR filter is 26.2 FO4 delays. This is a substantially higher throughput compared to the fastest previously reported IIR filter implementations. The operating frequency was measured using the industry-standard static timing analysis tool (PrimeTime) on a 3D extraction of the complete IIR filter layout.

II. GENERALIZED CARRY-SAVE (GCS) ARITHMETIC

A. Introduction to GCS

The carry operation in addition and multiplication is bitwise serial in nature, greatly limiting speed and increasing power. We have therefore developed Generalized Carry Save (GCS), which seeks to convert arbitrary networks of additions and multiplications, such as those found in DSP blocks (e.g., IIR filters) [6], into bitwise parallel computations, greatly increasing speed. In other words, we seek to convert an entire network ofadders and multipliers into a single Carry Save Adder array (CSA), whenever that is possible and when it is efficient to do so.

For example, consider Eqs. (4)-(6) below, where the inputs a-f are 6-bit vectors in 2’s compliment form. It is common for a high
level description of a DSP computation to be specified in such a manner, with various intermediate results. A commercial automatic
synthesis tool will not consider the possibility of merging a set of equations, thus failing to identify an important optimization pos-
sibility. Instead, in our GCS approach, we merge equations whenever possible [6]. For this example, we obtain Eq. (7).

\[ X = 4 \ast (a \ast b + 2 \ast b) + 10 \]  
\[ Y = c \ast d + e - 4 \]  
\[ F = 2 \ast X + 4 \ast Y + f \]  
\[ F_{opt} = 8 \ast a \ast b + 16 \ast b + 4 \ast c \ast d + 4 \ast e + f + 4 \]  

The next step is to generate all of the partial products and align all vectors in a large compression matrix (CM), as shown in Fig.
1. In GCS, sign extension is handled by first assuming all partial product rows are negative and then adding the corresponding sign
extension “1” bits, creating a separate so-called partial product row for these summed sign extension bits. A correction literal (the
complement of the actual sign bit for that partial product) is added in case the partial product was in fact positive [6, 8].

The partial product rows, such as those in Fig. 1, are then compressed down to two rows in a bit-wise parallel manner. As a basic
example, consider 4-row to 2-row compression as shown in Fig. 2. As Fig. 3 shows, in 4:2 compression [7] \( C_{out} \) does not depend on
\( C_{in} \) (the latter only impacts the so-called dropped carry \( C \) in blue) and hence no rippling of carries occurs. Here the four input vectors
are transformed via a set of full adders to two vector outputs, often called carry and sum. Compared to the conventional 4-bit ripple
adder, which requires four Full Adder (FA) gate delays for the computation, the compressor structure in Fig. 2 only requires two FA
delays.
In general, compressors that transform \( n \) vectors down to 2 vectors are straightforward to develop, and the worst-case delay path through the compressor grows as the logarithm of \( n \). Such structures are known as Wallace trees [8].

If a final binary result is required, then a pipelined carry propagate adder can be employed.

**B. GCS Application to an FIR Filter**

Fig. 4 gives the schematic of a basic second-order FIR digital filter circuit [1]. It consists of coefficients \( c_1 \) and \( c_2 \), two multipliers, two adders and two unit-delay elements. The expression for the output \( y(n) \) of a second order FIR can be written as

\[
y(n) = \sum_{i=0}^{N} c_i x(n-i)
\]

Fig. 5 shows a GCS implementation of the FIR filter. Here we use binary vectors to represent all numbers. For simplicity, here we assume the input \( x(n) \) and the coefficients \( c_1 \) and \( c_2 \) are all positive numbers and represented with four bits. Hence multiplication produces four partial products in this simplified scenario. Each small square in the compressors of Fig. 5 represents one binary bit and each row is one vector. A pair of 4:2 compressors separately generate \( c_1 x(n-1) \) and \( c_2 x(n-2) \). A 6:2 compressor array compresses the two rows from \( c_1 x(n-1) \), the two rows from \( c_2 x(n-2) \) with the input \( x(n) \), which is assumed to be in redundant (carry-save) form. If \( x(n) \) is not in redundant form, the 6:2 compressor array would be converted to a 5:2 compressor array. The final sum and carry rows that comprise the output \( y(n) \) are the outputs from the 6:2 compressor array.
III. CONVENTIONAL TWO-STAGE SECOND-ORDER FILTER SECTION WITH QUANTIZER

A conventional second-order section (SOS) of a digital IIR filter is shown in Fig. 6 [4]. It contains coefficients $c_1$ and $c_2$, two multipliers, one adder and two unit-delay elements. To maintain stability with the use of fixed precision arithmetic, the filter employs a quantizer ($Q$). The quantizer is also known as limiter which is used to determine if there is an overflow (or underflow) of vectors and truncate the vectors to a desired length.

The expression for the output $y(n)$ can be written as

$$y(n) = Q[x(n) + c_1 y(n-1) + c_2 y(n-2)]$$  \hspace{1cm} (9)

The critical stage delay in Fig. 6 consists of a multiplication (e.g., $c_2$ times $y(n-2)$), an addition (adding $c_2 y(n-2)$ and $c_1 y(n-1)$ to $x(n)$), and one quantization. The adder must actually sum three terms together (one input vector $x(n)$ and the two multiplication outputs). This results in a rather long minimum achievable clock period and hence limits the throughput.

A. Specific Industrial IIR Filter

We received a specific IIR filter from industry. The input $x(n)$, each coefficient and the output $y(n)$ were each given in fixed-point arithmetic, in 2’s complement, specified in the form of $[t\ w]$. Here $t$ is the number of integer bits and $w$ is the number of fractional bits. In particular, the input $x(n)$ is in the format $[1, 21]$. The output $y(n)$ is in the format of $[13, 15]$. Meanwhile, the coefficients $c_1$, $c_2$ are in the format of $[3, 16]$, where their specific values were: 001.1111101111110010 and 111.0000001000111100, respectively. The Most Significant Bit (MSB) is a sign bit. By using Booth encoding these two coefficients can be rewritten as 010.00000010010001100 and 001.00000010010001100, respectively.

Fig. 7 shows the arrangement of partial products, for the mentioned coefficient values, after applying GCS arithmetic to the basic second-order IIR digital filter in Fig. 6.
In Fig. 7, we use one 8:2 compressor array to compress the partial products for the multiplication of \( c_1 y(n-1) \) down to two rows, and one 9:2 compressor array to compress the partial products of \( c_2 y(n-2) + x(n) \) down to two rows. A 4:2 compressor array then produces the output \( y(n) \) in redundant format, in terms of the two vectors named \textit{sum} and \textit{carry}.

### B. Pipelined Second-Order IIR Filter

In order to reduce the clock period and thus increase the operating frequency, we used the higher throughput digital IIR filter structure shown in Fig. 8 [4], which still uses the same coefficients. The expression for \( y(n) \) for this structure is also given by (5).

With this improved design, the worst stage delay (producing output \( y(n-2) \) in redundant form) consists of one multiplication delay, one addition delay and the quantization delay. While this is similar to the previous conventional structure in Fig. 7, the addition is considerably faster since only two terms need to be summed (the output of the first delay element and the multiplication \( c_1 y(n-2) \)) and this can be accomplished by a much smaller, and faster, compressor array. Hence this pipelined structure is the design of choice for very high throughput while maintaining low power. However, the delay of the quantizer \( Q \) severely impacts the delay of this stage, making it by far the most critical stage in the design and thus limits the throughput. In the next section we will describe our redundant quantizer design and it will be shown how the quantization delay is removed from the critical path.

### IV. REDUNDANT QUANTIZER DESIGN

If the output data is a single vector, the sign bit and all overflow bits can be examined to determine if there is an overflow (or underflow). However, due to our use of GCS arithmetic, the output data is a redundant pair of vectors, \textit{carry} and \textit{sum}, in the format [13, 17] (13 integer bits and 17 fractional bits) in 2’s complement representation. In principle, we would have to add this pair of vectors to determine whether overflow or underflow has occurred. Furthermore, although the data precision requirement for the integer part of the output is only 13 bits, all intermediate results use 15 integer bits; that’s due to the fact that if the previous output \( y(n) \) is on the verge of overflow or underflow, the next calculation of \( y(n) \) may require as many as 15 bits. Overflow (or underflow) is avoided if the most significant three bits of the weight are all 0 or all 1. The situation is shown in Fig. 9, where bits \( S_{14}, S_{13}, \) and \( S_{12} \) must be checked to see if they are all 0 or all 1. If that is not the case, then overflow has occurred if bit \( S_{14} \) is 0, or underflow has occurred if bit \( S_{14} \) is 1.
One approach to ascertain bits $S_{14}, S_{13}$ and $S_{12}$ would be to use a carry-propagate adder to add the redundant vectors $sum$ and $carry$. However, this would severely increase the delay of the quantizer. Instead, we use a carry-select addition approach, in which two 3-bit carry select adders and one 29-bit carry look-ahead tree are used (the latter operates on bit weights 11 through -17). These two carry select adders, $adder1$ and $adder0$, compute the three-bit addition result $S_{14}S_{13}S_{12}$ with the assumed carries of $C_{12}=1$ and $C_{12}=0$, respectively, and the carry look-ahead tree computes the real $C_{12}$ in parallel with the operation of these two adders. The correct values of $S_{14}S_{13}S_{12}$ are then selected by a multiplexer driven by $C_{12}$.

Table 1 shows all of the no overflow/underflow conditions for $adder0$, assuming that the $C_{12}$ bit is “0”. The first two cases occur when the assumed carry is equal to the actual carry. In this case, the actual sum bits $S_{14}S_{13}S_{12}$ must all be 0 or all be 1 to avoid overflow and underflow. For case 3, the carry-select adder $adder0$ assumed a carry of 0, but the actual carry value $C_{12}$ was one. Hence we need to add a 1 bit at the weight of 12 in order to get the correct sum bits $S_{14}S_{13}S_{12}$. Adding 001 to 111 yields 000 which is consistent with the absence of overflow/underflow. Similarly, for case 4, adding 001 to 110 gives 111, again a condition for the absence of overflow/underflow.

**TABLE 1 NO_OF/UF CONDITIONS FOR $adder0$**

<table>
<thead>
<tr>
<th></th>
<th>$S_{14}$</th>
<th>$S_{13}$</th>
<th>$S_{12}$</th>
<th>$C_{12}$ assumed</th>
<th>$C_{12}$ real</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Case 2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Case 3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Case 4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2 shows the no overflow/underflow conditions for $adder1$, which assumes that the $C_{12}$ bit is ‘1’. For case 1’, where the assumed carry was 1 but the real carry was 0, we must subtract 001 from 000, which yields the no overflow/underflow condition of 111. Case 2’ is similar, where the subtraction of 001 yields 000, again a condition for no overflow/underflow. For cases 3’ and 4’, the assumed carry matches the actual carry, and the conditions for no overflow/underflow are, respectively, 111 and 000.

**TABLE 2 NO_OF/UF CONDITIONS FOR $adder1$**

<table>
<thead>
<tr>
<th></th>
<th>$S_{14}$</th>
<th>$S_{13}$</th>
<th>$S_{12}$</th>
<th>$C_{12}$ assumed</th>
<th>$C_{12}$ real</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1’</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Case 2’</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Case 3’</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Case 4’</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Of course, the 8 cases in Tables 1 and 2 are not independent. Case 2’ in Table 2 is equivalent to case 1 in Table 1, since in the former case the assumed carry is 1 whereas the real carry is 0; thus, it is necessary to subtract 001 from $S_{14}S_{13}S_{12}$ and thereby obtain 000. Similarly, case 2 is equivalent to case 1’, case 3 is equivalent to case 4’, and case 4 is equivalent to case 3’. Therefore Tables 1 and 2 can be merged into the single Table 3.
TABLE 3 NO_OF/UF CONDITIONS FOR BOTH ADDERS

<table>
<thead>
<tr>
<th>Case</th>
<th>S_{14}</th>
<th>S_{13}</th>
<th>S_{12}</th>
<th>C_{12} assumed</th>
<th>C_{12} real</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (Adder1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2 (Adder1)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3 (Adder0)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4 (Adder0)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

We chose the particular four conditions in Table 3 among the eight redundant conditions since the logical expression for the no overflow/underflow condition is the simplest:

\[
\text{No}_\text{OF}/\text{UF} = (S_{14} S_{13} C_{12})_{\text{adder2}} + (S_{14} S_{13} C_{12})_{\text{adder1}}
\]  

Fig. 10 shows the circuit implementation for (10). When the output No_OF/UF is high there is no overflow or underflow.

![Fig. 10 No_OF/UF signal generation](image)

We also need another signal to specify whether a violation is an “overflow” or an “underflow”. This signal is given by the sign bit, S_{14}. The upper part of Fig. 11 shows the implementation for Sign_bit generation.

![Fig. 11 Implementation of sign bit generation and data selection](image)

Using the two signals No_OF/UF and Sign_bit, the proper output can be selected, as shown in the lower part in Fig. 11. Two MUX blocks are used. When No_OF/UF is high there is no overflow (underflow) and the real data will be selected by the second MUX. When No_OF/UF is low, the second MUX will select the upper limit or lower limit generated by the first MUX according to
the selection signal Sign_bit.

Although we use an optimized method to implement the quantizer block, it still requires the computation time associated with the 29-bit carry look-ahead tree. So it is important to try to move the quantization block outside of any critical stage delay. A new second-order IIR filter design that accomplishes this is put forth in the next section.

V. HIGHER THROUGHPUT SECOND-ORDER IIR FILTER WITH OPTIMIZED QUANTIZER

Our new IIR filter starts with the pipelined structure shown in Fig. 8. However, we were able to move the quantizer out of any critical stage delay. The schematic of this new design is shown in Fig. 12.

![Fig. 12 New second-order IIR filter with quantizer](image)

Note that while the non-quantized output (the output of the second delay unit or flip-flop) is separately multiplied by both $c_1$ and $c_2$, the quantizer is in parallel determining whether or not overflow/underflow has occurred. By the time the multiplications and additions (actually, compressions) have been completed, and the three inputs to MUXI and MUXII are ready, the select signals for MUXI and MUXII will have arrived by means of the quantizer. If overflow (underflow) occurred, instead of passing the result of the multiplication (by $c_2$) and addition of $x(n)$, MUXI will pass the redundant vector pair consisting of $x(n)$ and the pre-multiplied value of $c_2$ times the upper (lower) limit of the quantizer. Furthermore, if overflow (underflow) occurred, instead of passing the (redundant) result of the multiplication (by $c_1$) and addition of the intermediate redundant vectors from the first delay element (or flip-flop), MUXII will pass the redundant vector pair resulting from 3:2 compression of the pre-multiplied value of $c_1$ times the upper (lower) limit of the quantizer and the redundant vector pair output by the first delay element (or flip-flop).

Since the multiplication and addition computations (actually, by means of bit-wise parallel compression) take more time than to determine the quantizer output, when the addition result arrives at the multiplexers, the quantizer selection signals (No_OF/UF and Sign_bit) are ready at the selection ports of MUXI and MUXII. A third multiplexer (MUXIII) outputs either the computed non-quantized output or the upper or lower limits of the quantization process, again according to the selection signals (No_OF/UF and Sign_bit).

As we have moved the quantizer out of the critical stage delay now, the multiplication components do not have to wait for the quantized data; instead they receive data before the quantization step. Therefore, the achievable minimum clock period is reduced to the bit-wise compression delay corresponding to one multiplication and one addition, along with one MUX delay now, effectively removing the quantization delay from the structure in Fig. 8.

A. Circuit Implementation

To further improve the efficiency of the novel structure described in Fig. 13, we used GCS arithmetic for the multiplications and additions in our IIR filter hardware implementation. Fig. 14 shows the circuit implementation for the second-order IIR filter in Fig. 12.
Booth’s algorithm is used in the multiplication process to reduce the number of partial products that are generated [8, 10]. A string of 1’s in the multiplier vector is replaced with a 1 at the next highest bit weight, and with a -1 at the lowest bit weight (for the string of 1’s). Given that our design has fixed coefficients, the Booth recoding process is done at the design stage, rather than being implemented in hardware.

Figs. 14 and 15 show how GCS arithmetic was applied to the whole structure of the new second-order IIR filter in Fig. 12. Fig. 13 represents the multiplication \( c_1y(n-1) \) in (5). Fig. 15 is the entire GCS arithmetic implementation for the design. One 8:2 compressor array and one 9:2 compressor array are used for partial product reduction (for the two required multiplications).

![Fig. 13 Circuit implementation of the novel second-order IIR filter. Here sel consists of two bits, No_OF/UF and Sign_bit](image-url)

![Fig. 14 GCS for c1y(n-1)](image-url)
The redundant carry and sum vectors from the MUXIII in Fig. 12 represent the final output of the IIR filter. A final carry propagate adder is necessary if the subsequent block (using the IIR filter output) demands a binary vector. In this case, it is straightforward to deploy a pipelined adder so as not to impact the throughput of the IIR filter.

C. Intermediate Data Truncation

The fixed coefficients $c_1$ and $c_2$ are in the format $[2, 16]$ and the output data is $[13, 15]$. Thus, the full precision of one intermediate result, such as $c_1y(n-1)$ or $c_2y(n-2)$, is $[15, 31]$. Reducing this precision to save circuit area is a desired optimization as long as the maximum error is below the system requirement. In our design, the output (sum of final carry and sum vectors) is specified as $[13, 15]$, and accordingly the maximum error for the fractional part is approximately $2^{-15}$ (occurring when all truncated bits are 1).

Fig. 16 shows an example of the maximum error generated by a truncation. If we truncate all bits after bit weight -21 for 31 fractional bits, when all bits from -22 to -31 are ‘1’ we will obtain a maximum error of $2^{-21} – \text{ulp}$, where the $\text{ulp}$ (unit in the least position) is equal to $2^{-31}$.

In our case, however, the output is the redundant pair of vectors sum and carry. Since both will be truncated, each of these two vectors must be retained with 17 fractional bits, meaning that the error will be approximately $2^{-16}$ (or two times $2^{-17}$). We need to maintain the more stringent bound of $2^{-16}$ since we also perform truncation during compression of the 17 partial products, which saves a great deal of compressor array circuit area. If we restrict the error of the latter source (i.e., the compression) to also be $2^{-16}$, our overall truncation error will be $2^{-15}$ (the sum of $2^{-16}$ and $2^{-16}$).

From Figs. 14 and 15, we have 17 partial products that will be truncated while retaining $p$ fractional bits. Thus, this truncation error of 17 times $2^p$ must be less than $2^{-16}$. The value of $p$ is the number of fractional bits we must retain.

$$17 \times 2^{-p} < 2^{-16}$$

In (11) the minimum integer $p$ is 21, which means for intermediate vectors 21 fractional bits must be retained. The overall
truncation error is thus given by (12), where the $2^{-16}$ term is from (12) with $p = 21$, and where the term $2$ times $2^{-17}$ is due to retaining 17 bits for each of the final redundant outputs sum and carry. This shows that the original truncation error bound of $2^{-15}$ is satisfied.

$$2^{-16} + 2^{-17} \times 2 = 2^{-15}$$  \hspace{1cm} (12)

VI. RESULTS

The new IIR filter design was implemented using the IBM 130 nm 1.2 V technology. We used a cell library that included the following gates: INV, NOR2, NOR3, NAND2, NAND3, NAND4, AOI22, AOI12, OA112, CARRY, SUM and FLIP-FLOP. Previously it was shown that a library with these functional elements yielded the best power efficiency [9]. For each type of gate, there were typically nine drive strengths: 0.5X, 1X, 2X, 3X, 4X, 5X, 6X, 7X, and 8X. In addition, several beta ratios (defined as the pull-up pMOS widths divided by the pull-down nMOS widths) were provided.

To show the benefit of the new IIR design that removes the quantizer from the critical second stage delay (which includes $c1*y(n-2)$), the prior art IIR filter structure shown in Fig. 8 was also implemented. The same coefficients and bit widths were used for both. In addition, the same design flow was used for both, including the application of the GCS algorithm.

We used Synopsys Design Compiler to synthesize the two designs for a wide range of delay targets, from the fastest it could produce to nearly the slowest (minimum power and area). Fig. 17 shows the resulting energy (per operation) versus delay plots.

![Energy vs. Delay](image)

Fig. 17 Energy versus delay plots

The minimum delay possible using the design from Fig. 8 was 1603 ps, whereas using our new design enabled a minimum delay of 1163 ps, which represents a 28% reduction in delay. However the prior art design in Fig. 8 does achieve lower energy for long delay targets. When the delay target is greater than 1750 ps, the design in Fig. 8 is advantageous with respect to energy efficiency. However, when the delay target is less than 1750 ps, our new IIR design should be used since it is more energy efficient for faster delays, and also yields faster design points than what the prior art design can achieve.

The Cadence Placement and Routing (P&R) tool EDI was used to achieve a complete layout for our new design for the delay point of 1313 ps in Fig. 17. Also it is used to achieve a complete layout for prior second order IIR for the delay point of 1921 ps. These points are high speed, but somewhat closer to the “knee” or bend in the energy-delay curve where the energy-delay product is minimized.

Static timing analysis (STA) was done using Synopsys’ PrimeTime, first for the entire circuit before layout, and then also using the full 3D-extracted netlist. Table 4 gives the STA results, where the worst-case delay from register to register before layout was 1313 ps, compared to 1674 ps with a full RC extraction for the 130 nm IBM process at 1.2 V under nominal conditions. The new, very high throughput digital IIR filter architecture thus operates at 600 MHz. We also include the STA results of both pre-layout and post-layout for implementation from Fig. 8 in Table 4.
TABLE 4 TIMING ANALYSIS FOR OUR NEW IIR DESIGN

<table>
<thead>
<tr>
<th>Design</th>
<th>Conditions</th>
<th>Delay before P&amp;R (ps)</th>
<th>Delay with full RC extraction (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our design</td>
<td>25 C, 1.2V</td>
<td>1313</td>
<td>1674</td>
</tr>
<tr>
<td>Design in Fig. 8</td>
<td>25C, 1.2V</td>
<td>1921</td>
<td>2331</td>
</tr>
</tbody>
</table>

VII. SUMMARY

We have developed and implemented a new, very high throughput digital IIR filter architecture that operates at 600 MHz using the IBM 130 nm process at 1.2 V. The inverter fanout-of-four (FO4) delay for the IBM process is 64 ps. The IIR filter’s worst-case stage delay is therefore 26.2 FO4 delays. This is a substantially higher throughput than the fastest previously reported method to implement IIR filters [4].

Critical to this achievement is the use of GCS arithmetic, a pipelined IIR filter implementation and a novel implementation of the quantizer. GCS arithmetic systematically avoids rippling of carries in a signal-processing network of additions and multiplications. GCS arithmetic enables nominally bit-wise serial mathematical operations, such as addition and multiplication, to be computed bit-wise parallel. Another important aspect to the filter architecture is a novel implementation of the quantizer, which allows it to be placed in its own separate pipeline stage. In previous digital IIR filter designs, the quantizer and its associated delay had to be included in one of the computational stages and hence was a major factor that limited the throughput of the filter. In addition, we compared our new approach with the prior art second-order pipelined filter design. Our new IIR filter design was 28% faster than the prior art design, and had superior energy efficiency for a wide range of delays. For delays much longer than the fastest possible, the new design was not advantageous from an energy perspective.

Maximizing the throughput of digital filters is critical today due to the need for low power circuit designs. Even if a lower throughput is desired, first maximizing the throughput and then lowering the power supply voltage to just meet the throughput requirement yields the lowest dynamic energy implementation as the dynamic energy is quadratically related to the supply voltage.

REFERENCES


Zhao Wang acquired his Ph.D. in Electrical Engineering from The University of Texas at Dallas, TX in 2012. Before that he obtained his master degree in Electrical Engineering from The University of Texas at Dallas, TX in 2009. His main research areas when pursuing Ph.D. degree were circuit optimization and CAD algorithm development. He is currently working at Apple Inc., Cupertino, CA as a circuit designer. His current research interests are VLSI design, CAD algorithm development and new emerging technology deployment.

Carl M. Sechen acquired his Ph.D. in Electrical Engineering from UC Berkeley, CA in 1987. Before that He obtained his master degree in Electrical Engineering from MIT, MA in 1977 and bachelor degree in Electrical Engineering from University of Minnesota in 1975. He is currently a professor at The University of Texas at Dallas, TX. His research areas include design and computer-aided design of integrated circuits. Before joining UT Dallas he had been with University of Washington and Yale University. Dr. Sechen was rewarded as IEEE fellow in 2002.