Design of high-radix digit-slices for on-line computations

Alexandre F. Tenca Miloš D. Ercegovac

Computer Science Department
University of California, Los Angeles

ABSTRACT

We present a design of high-radix digit-slices for the implementation of on-line multiply-add operator (OMA). Our evaluation of performance and cost shows that speedups above 1.5 can be obtained with respect to radix 2 at reasonable increase in cost. The design and evaluation are based on the Xilinx FPGAs. We also discuss the use of OMAs modules in solving linear recurrences.

Keywords: computer arithmetic, high radix on-line arithmetic, multiply-add, field-programmable gate arrays (FPGAs), linear recurrences, redundant number systems.

1 Introduction

An on-line operator accepts the inputs serially, most-significant digit first, and produces the result in the same manner. There is an on-line delay δ between the j^{th} input digit and the corresponding output digit – usually 2 to 5 digit cycles. On-line modules are attractive for specific applications, in particular those requiring long sequences of dependent operations as well as long precision. On-line arithmetic reduces the bandwidth of inter-module connections to digit-serial links while allowing the overlap among data-dependent operations thus reducing the total computation time.[8,4,1]

The paper presents preliminary results obtained in our investigation of high-radix designs for on-line arithmetic. Most of the published designs use radix 2 or radix 4.[14,1,13,9] The on-line delay δ, the radix r, the digit cycle time t_r, the number of pipeline stages p, and the number of cycles m are all interdependent and determine the latency (total time) and throughput of the networks of on-line modules. Radix-2 modules are simplest to design, have shortest digit cycle time but require the largest number of cycles. The choice of these parameters is not obvious. For example, a higher radix reduces the number of cycles and on-line delay but increases the digit cycle time. Increased pipelining decreases the cycle time while increasing the on-line delay. The problem addressed here deals with a possibility of using designs with higher radices while managing the drawbacks so that a speedup obtained and the cost are advantageous with respect to radix 2. We consider here an on-line multiply-add operator (OMA). Multiply-add operators are frequently used in signal processing applications. On-line alternative is useful when a sequence of multiply-add operations is needed to minimize the effect of data dependencies and reduce the interconnections between the operators. As an example of its use we discuss an on-line scheme for solving linear recurrences.[11,10] We assume the reader is familiar with the structure and features of Xilinx FPGAs.[16]

The paper is organized as follows: first we present the basic on-line recurrence equation, followed by a discussion on possible implementations for high-radix designs, a comparison of radix-2 and higher radices circuits, solving linear recurrences on the VCC EVC board,[15] and conclusion.

2 On-line algorithms for $ax + b$ (OMA) module

We now discuss an on-line algorithm for the multiply-add operation $y = ax + b$ where a is in parallel and x and b in digit-serial form. All inputs are fractions, i.e., $1/2 \leq a, x, b < 1$. The recurrence equation is derived following [4]. At step j, the error between the actual and computed function values is bounded by

$$|ax[j] + b[j] - y[j]| < r^{-j}$$

where

$$x[j] = \sum_{i=0}^{j+\delta} x_i r^{-i}, b[j] = \sum_{i=0}^{j+\delta} b_i r^{-i} \text{ and } y[j] = \sum_{i=0}^{j} y_i^* r^{-i}$$

and $x_j, b_j \in \{-\rho, ..., \rho\}, \rho < r$ and $y_j^* \in \{-\gamma, ..., \gamma\}, \gamma \geq r$. To simplify the selection of y_j^*, we allow the output digit set to be over-redundant. From the scaled residual:

$$W[j] = r^j(ax[j] + b[j] - y[j]) \tag{1}$$

with the initial condition: $W[0] = ax[0] + b[0]$, we obtain the residual recurrence equation:

$$W[j] = \underbrace{rW[j-1] + \overbrace{r^{-\delta}(ax_{j+\delta} + b_{j+\delta})}^{H[j-1]}}_{Z[j-1]} - y_j^* \tag{2}$$

From the residual bound $W[j]| \leq \omega < 1$ and the containment condition

$$max(z[j-1]) \leq \omega + \gamma \Rightarrow r + r^{-\delta} 2\rho r^{-\delta} \leq 1 + \gamma \Rightarrow r^{-\delta} \leq \frac{r}{2\rho}$$

we obtain the on-line delay $\delta \geq 1$ since $\rho \leq (r-1)$. Consequently, the residual recurrence is

$$W[j] = rW[j-1] + r^{-1}(ax_{j+\delta} + b_{j+\delta}) - y_j^*$$

The selection of the ouput digit y_j^* is performed by taking the integer part of $Z[j-1]$, i.e., $y_j^* = \lfloor z[j-1] \rfloor$. The computation of the residual and the selection by truncation are illustrated bellow:

$$\begin{array}{lccccccccc}
rW[j-1] & & x. & x & x & x & x & x & x & x \\
b_{j+1}.r^{-1} & & & x & & & & & & \\
a.x_{j+1}.r^{-1} & & & & x & x & x & x & x & x \\
\hline
 & t & z. & z & z & z & z & z & z & z \\
 & \underbrace{}_{y_j^*} & & \multicolumn{7}{c}{\underbrace{}_{W[j]}}
\end{array} \tag{3}$$

where the most-significant transfer digit $t \in \{-1, 0, 1\}$ and z is a signed digit in the set $\{-\rho, ..., \rho\}$.

There are two possibilities in dealing with the over-redundant output digits. One solution is to use a simplified radix-r on-line adder to recode the output digit into a redundant set as discussed later. The overall timing relations during on-line operation is shown bellow where $y_j = t_j + s_{j+1}$:

x_{j+1}	x_0	x_1	x_2	x_3	x_4	x_5	x_6	
b_{j+1}	b_0	b_1	b_2	b_3	b_4	b_5	b_6	
$W[j]$			$W[1]$	$W[2]$	$W[3]$	$W[4]$	$W[5]$	
y_j^*			t_0	t_1	t_2	t_3	t_4	t_5
			s_1	s_2	s_3	s_4	s_5	s_6
y_j			y_0	y_1	y_2	y_3	y_4	y_5

Another way of dealing with the over-redundant output digits is to use it directly (without recoding) in forming the product $a \times x_j$. This may require an increase in reduction stages (increasing the delay), but in the other hand, it will make possible to remove the output digit recoding from the circuit (reducing the delay). The step time can be kept the same by the use of pipelining. Because of the large increase in the cost of multiple generation, we do not consider this approach further.

3 Implementation of OMA module

The general diagram of the OMA module is shown in Figure 1.

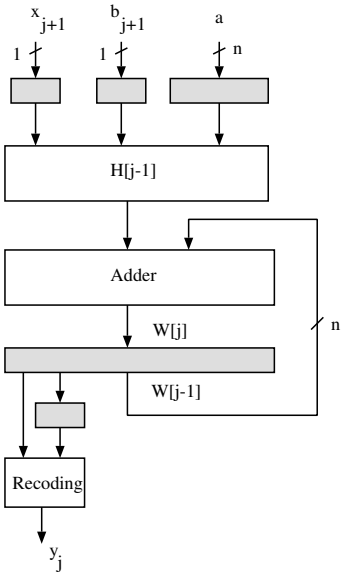

Figure 1: General Organization of OMA Module

The main components to implement radix-r digit-slices for the OMA module are: the radix-r signed-digit adder SDA_r which produces the residual W, the H network with the ax_j multiplier which produces the term $ax_{j+1} + b_{j+1}$, and the output digit recoder ODR which produces y_j. The on-line delay of implementation δ_{impl} depends on the theoretical on-line delay δ and the number of pipelined stages p in the network H, the redundant adder SDA, and the recoder ODR. The cycle time t is determined by the slowest stage in the pipeline.

Residual Adder. This adder is composed of basic radix-2 signed-digit adders (SDAs) organized in groups of size k to implement a radix-2^k adder. The structure of radix-8 SDAs and the basic radix-2 SDA are shown in Figure 2. A radix-r SDA has three inputs u, v and t_{in}, and two outputs t_{out} and s, where u, v and $s \in D_\rho$ and

the transfer digits t_{in} and $t_{out} \in S_{bit} = \{-1, 0, 1\}$. We consider the digits in radix r represented in *Borrow-Save (BS) code*.[3] A digit d is in the set $D_\rho = \{-(r-1), ..., -1, 0, 1, ..., (r-1)\}$, and represented as a signed-bit vector $(d_{k-1}, ..., d_1, d_0)$, with $d_i = (d_i^+ - d_i^-)$, $d_i^+, d_i^- \in \{0, 1\}$, and $k = log_2 r$. The value of the digit is $d = \sum_{i=0}^{k-1} d_i 2^i$. The function performed by a SDA is $v + u + t_{in} = rt_{out} + s$. The SDA we assume uses two levels of PPM adders. A PPM module corresponds to a full-adder as described in [3]. The PPM module produces 2 binary outputs, t and u, from 3 binary variables x, y, and z, such that $x + y - z = 2t - u$. The switching expressions are $t = xy + xz' + yz'$, and $u = x \oplus y \oplus z$. A radix-r SDA is composed of k SDAs in radix 2. The implementation of this module with FPGAs XC4000 uses $2k$ CLBs for a SDA_r ($k = log_2(r)$). The delay is equivalent to 2 F-function generators (10ns) plus interconnection delay.

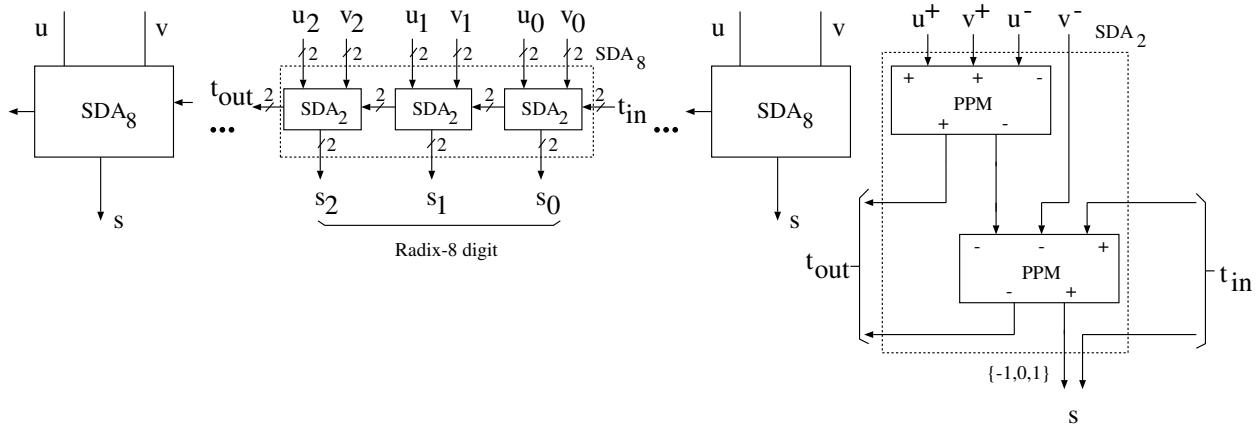

Figure 2: Structure of a radix-8 Signed-Digit Adder

H Network ($ax_j + b_j$ generation). A basic operation in the generation of $ax + b$ is the word-by-digit multiplication (i.e., multiple generation). The multiple generation increases in complexity as the radix increases. Let's assume that the operand a is a vector of radix-r signed digits and x_j is a single digit, all of them in BS form. This assumption comes from the fact that x_j is produced in the BS form and the insertion of b_j is simplified if a is in SD form as discussed later. A parallel multiplier to compute ax_j uses digit-by-digit primitive multiplications of a by x_j, $ax_j = (a_{n-1}x_j, a_{n-2}x_j, ..., a_1 x_j, a_0 x_j)$, with $a_i \in D_\rho$. Each individual digit-by-digit multiplication generates two-digit product (p, q) such that $a_i x_j = rp_i + q_i$, $r > 2$.

The digit-by-digit products are implemented at the binary level using signed bits (sbits). The matrix of partial products in the sbit form for $a_i x_j$ is:

$$\begin{array}{cccccc} & & (a_i^{k-1}x_j^0) & \cdots \quad \cdots & (a_i^1 x_j^0) & (a_i^0 x_j^0) \\ & (a_i^{k-1}x_j^1) & (a_i^{k-2}x_j^1) & \cdots \quad \cdots & (a_i^0 x_j^1) & \\ & \vdots & \vdots & & & \\ (a_i^{k-1}x_j^0) & \cdots \quad \cdots & (a_i^0 x_j^{k-1}) & & & \end{array}$$

The total number of sbits obtained with this product generation is k^2. As each sbit multiplier requires one CLB in the XC4000 FPGAs, the total number of CLBs used is k^2. The total number of sbits, for a precision m bits of the operand a and n radix-r digits is km. The multiple generator is the combination of all digit-by-digit multiplications as shown in Figure 3.

We considered two possibilities to reduce the partial products: with SDAs and with FAs and HAs. The

second approach reduces the positive and negative parts of the sbits independently, and the individual results are combined at the end to obtain the final multiplication result. Comparing both cases we concluded that the SDA reduction is faster, requiring the same area as the FA reduction. The use of SDAs to reduce the partial products is better because a SDA is a (2,1) counter with a reduction factor of 50%, while FAs are (3,2) counters with a reduction factor of 33%. The number of stages in the reduction tree using SDAs is $\lceil log_2(N) \rceil$ where N is the number of sbit rows. The delay and area used by the SDA reduction structure is:

$$delay = \lceil log_2(N) \rceil \times t_{SDA}$$

$$\#CLBs = 2 \times (\sum_{i=0}^{log_2 N} N2^i) = 2N(N-1)$$

The reduction result of the digit-by-digit multiplication generates two radix-r digits. That forces the utilization of another level of SDAs to obtain only one digit. To avoid this extra reduction level and make a structure more suitable for digit-slice implementation we propose the organization shown below (and in dashed line in Figure 3). This organization overcomes the problem of irregular structure in the reduction structure of the digit multiplier, making it more suitable for VLSI and FPGA implementation. The slices shown in dashed lines are organized in such a way that the digit multiplier receives two digits from operand a and transfer digits from its neighbor. With these inputs it calculates the digit of $ax_j + b_j$ used in the slice. The following matrix of sbits is considered:

$$\left\{ \begin{array}{ccccc} (a_i^{k-1}x_j^0) & \cdots & \cdots & (a_i^1 x_j^0) & (a_i^0 x_j^0) \\ (a_i^{k-2}x_j^1) & \cdots & \cdots & (a_i^0 x_j^1) & (a_{i+1}^{k-1} x_j^1) \\ & \cdots & \cdots & & \\ (a_i^0 x_j^{k-1}) & \cdots & \cdots & (a_{i+1}^2 x_j^{k-1}) & (a_{i+1}^1 x_j^{k-1}) \end{array} \right\}$$

A block diagram of the digit-slice is also shown in the Figure 3. A high degree of pipelining can be used. This would increase the on-line delay while reducing the cycle time to a value close to the delay of a SDA. An evaluation of this effects is presented in section 4.

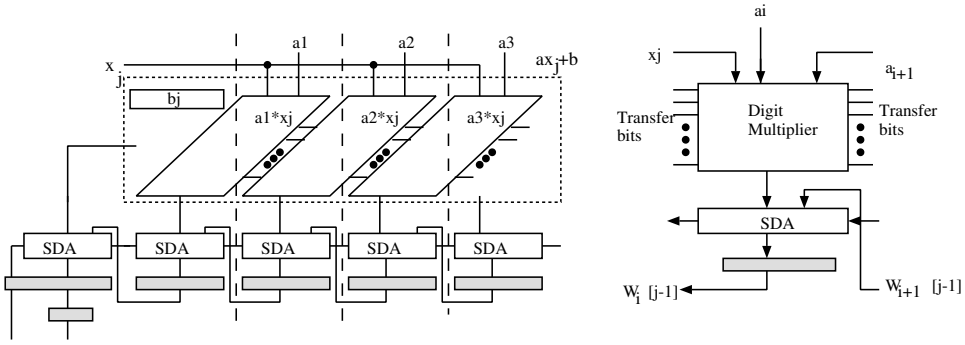

Figure 3: Organization with Digit Slices

The insertion of b_j is done in the leftmost digit-slice. This slice is different from the other ones since the reduction structure is specialized for reduction of transfer digits. Now we can explain that if a is in conventional form, the bit a_0 will be the sign bit and can be different than zero, using the space where b_j would be inserted. This would cause an extra level of SDAs to combine b_j to form the residual $W[j]$. The use of BS code instead of two's complement will not make the circuit slower or more area consuming. The only disadvantage is that it increases the amount of information that needs to be transferred during computation.

ODR recoder. This digit recoder is implemented as a simplified radix-r on-line adder with one operand restricted to -1, 0, and 1 values. The most significant sbits of the digit t_j are zeroes in the radix r representation. In this case, the PPM modules at the upper level can be removed with proper interconnection. From the PPM equations, assuming $x = z = 0$, we have that $t = u = y$. A radix-8 on-line recoder is shown in Figure 4. The two least significant sbits are latched to synchronize the values between different cycles. Another latch is used for z_2 to align the sbits in the output digit y_j.

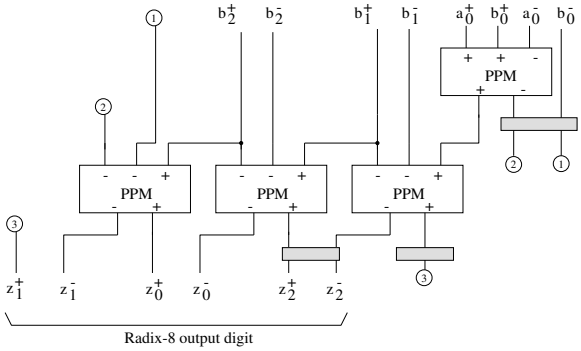

Figure 4: Radix-8 On-Line Recoder.

The implementation of this module with FPGAs XC4000 uses $k + 1$ CLBs for a radix $r = 2^k$ recoder. The delay is equivalent to 2 F-function generators plus latch delay (13ns) plus interconnection delay.

4 Comparison between OMAs designed for radix-2 and radix-r

This section compares the area and time used for radix-r and radix-2 organizations in computing a single multiply-add result. The precision of x and b is m bits, and of the operand a is m_a bits. The level of pipelining in the multiplier of the digit-slice is p. For example, $p = 1$ if one level of latches is inserted in the H network. Besides the on-line delay of the multiply-add algorithm ($\delta = 1$), the recoder adds a delay $\delta_{recoder} = 2$ for $r > 2$ and $\delta_{recoder} = 3$ for $r = 2$. For simplicity, in our estimates we do not consider the interconnection delay.

The area required by the implementations depends mainly on the precision of the operand a (m_a), the coefficient of the equation. It determines the number of digit-slices used. The precision of the result y can be variable, depending only on the precision of x and b operands and the number of iterations executed in the OMA module.

Using radix-2 digit-slices. The time to compute $ax + b$ using radix-2 OMA module is obtained as:

$$T_2 = (m + \delta_2 - 1) \text{ (cycles)}$$

$$\delta_2 = \delta + p + \delta_{recoder} = 1 + p + 3 = p + 4$$

The cycle time is dependent on the parameter p. Higher values of p results in a shorter cycle time. If $p = 1$, the on-line delay $\delta_2 = 5$ and the radix-2 digit-slice cycle time is equivalent to the propagation time of a SDA (t_{SDA}) plus setup and propagation time of the latch. For $p = 0$, $\delta_2 = 4$ and the cycle time increases by the sbit multiplication time (equivalent to a function generator delay – 4.5ns).

The area used consists of the digit slices' area and the on-line adder used for output generation. The total number of slices is $m_a + 1$. Using XC4000 FPGA, a radix-2 digit slice takes 4 CLBs (2 CLBs for the radix-2 SDA, 1 CLB for the digit multiplier and 1 CLB to store the input a_i). See Figure 5. The on-line adder uses 3 CLBs (same area as a SDA_2 plus latches). The expression is (in CLBs):

$$Area_2 = (m_a + 1) \times 4 + 3$$

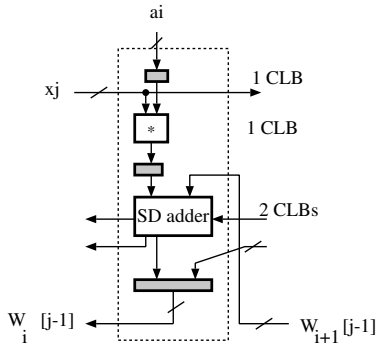

Figure 5: Radix-2 digit-slice for FPGAs

Using radix-r digit-slices The time to compute the result of m bits of precision, using radix-r digit slices is:

$$T_r = (n + \delta_r - 1) = (\lceil \frac{m}{k} \rceil + \delta_r - 1)$$

where $k = log_2(r)$. The on-line delay is computed as:

$$\delta_r = \delta + p + \delta_{recoder} = 1 + p + 2 = p + 3$$

The value of p can be at most $(\lceil log_2(k) \rceil + 1) = S + 1$ (number of stages in the multiplier reduction plus the partial product generation stage). With this maximum p, the cycle time is equivalent to a SDA propagation time (t_{SDA}) plus setup and propagation time of a latch. Lets assume that the worst cycle time is $(S + 2)t_{SDA} + t_{latch}$ (the delay of the product generator is the same of an SDA, pessimistic assumption). If the pipeline stages are correctly placed in order to minimize the resulting critical path, the cycle time can be calculated as:

$$t_r = \lceil \frac{S + 2}{p + 1} \rceil t_{SDA} + t_{latch}$$

The total area used by this scheme is (in CLBs):

$$Area_r = (\lceil \frac{m_a}{k} \rceil + 2) \times 2k + \lceil \frac{m_a}{k} \rceil (3k^2 - 2k) + (k + 1)$$

where $(3k^2 - 2k)$, $2k$ and $(k + 1)$ are the number of CLBs used by a radix-r digit multiplier, a radix-r SDA and a recoder, respectively.

Comparison of OMA designs The impact of p in the cycle time and the total time is presented in Table 1. The increase in the pipeline level reduces the cycle time (shown in the table normalized to the propagation delay of a SDA – t_{SDA}), and increase the on-line delay of the implementation. The values of on-line delay and cycle time were used to estimate the total time to obtain a result with $m = 32$ bits of precision. For this precision all

Cycle time (number of t_{SDA})						
Radix	p					S
	0	1	2	3	4	
4	3	2	1			1
8	4	2	2	1		2
16	4	2	2	1		2
32	5	3	2	2	1	3
OL delay	3	4	5	6	7	
Total time (t_{SDA}) m=32						
Radix	p					
	0	1	2	3	4	
4	54	38	20			
8	52	28	30	16		
16	40	22	24	13		
32	45	30	22	24	13	

Table 1: Impact of Pipelining On the Total Time and On-line Delay

Total Area (#CLBs)						
Radix	Precision (m_a)					k
	8	16	32	64	128	
2	39	71	135	263	519	
4	59	107	203	395	779	2
8	97	178	313	610	1177	3
16	117	213	405	789	1557	4
32	176	326	551	1001	1976	5
Speedup over radix 2						
Radix	Precision (m)					
	8	16	32	64	128	
4	1.50	1.67	1.80	1.89	1.94	
8	1.50	1.82	2.25	2.52	2.75	
16	1.71	2.22	2.77	3.24	3.57	
32	1.50	2.00	2.77	3.58	4.12	
64	1.50	2.22	3.00	4.00	4.71	

Table 2: Area, Time and Speedup for OMA modules in Different Radices

the radix r designs considered will compute the result in less time if a maximum degree of pipelining is used. For small precision the impact of a large on-line delay (large p) is worse than for large precision.

Table 2 presents the area (number of CLBs in the XC4000) and speedup obtained using radix r OMAs. Large speedups can be obtained for some radices, with corresponding increase in the area. Since the area used is a function of the precision of operand a, one can limit the precision of the operand a, and continue to have the possibility to compute x and b in variable precision. The designer need be aware of the relation area and performance to select the best radix r implementation. A Figure of merit, commonly used in VLSI implementations, was applied in our study (AT^2 – product of area and the square of the time) and the values obtained show that for $m_a = 32$ the best design is the radix-16 OMA. For $m_a = 8$, the best design is radix-4 OMA. When the precision increases, the use of higher radix designs results in a more cost-effective solution.

5 OMA Module with precomputed multiples

Since the input a is a known coefficient, we can precompute the multiples ax_j for all possible values of x_j. The precomputed multiples are stored in a memory, addressed by x_j. The scheme is presented in Figure 6. The designer would have to prepare appropriate OMA configurations for different values of a and download into the FPGA the configuration that is needed.

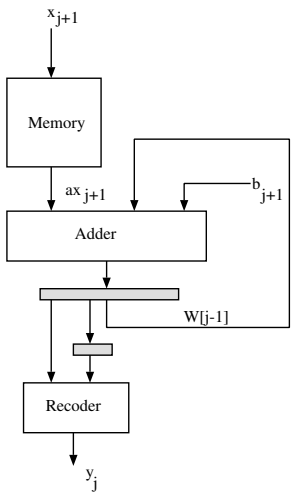

Figure 6: General Organization for OMA Module with Precomputed Values

Using XC4000, each CLB can be configured as a 16×2 bit array of Read/Write Memory. With this capability, the OMA may be designed with a RAM memory that is loaded as needed with the proper multiples of a for a particular radix from the Local Memory. This is probably the best solution since the FPGA configuration time is larger than the memory access time.

We illustrate this approach by a high-level design of a radix-16 OMA module. We assume that x_j is in BS code, with 8 bits representing each radix-16 digit x_j. In order to take advantage of the RAM configuration of CLBs in the XC4000 we propose a decomposition of ax_j as follows:

$$a \cdot x_j = a(x_j^+ - x_j^-) = a \cdot x_j^+ + (-a)x_j^-$$

where a is a constant. The values of ax_j^+ in BS form are stored in the memory bank addressed by x_j^+, and the values of $-ax_j^-$ are stored in the memory bank addressed by x_j^- as shown in Figure 7. Each of these multiples (ax_j^+ or ax_j^-) has $(\lceil \frac{m_a}{4} \rceil + 1)$ digits. Each digit has 4 sbits and the total number of sbits for the multiple is $8(\lceil \frac{m_a}{4} \rceil + 1)$. Each sbit can be stored in one CLB. The two selected multiples are combined into ax_j using one level of SDAs. Only 3 CLB levels are traversed to get the result. The number of CLBs require by the memory and the SDA level is: $16(\lceil \frac{m_a}{4} \rceil + 1)$ CLBs. The total number of CLBs is estimated as $16(\lceil \frac{m_a}{4} \rceil + 1) + 8(\lceil \frac{m_a}{4} \rceil + 2) + 5$. This includes multiple generation, the residual adder and the output digit recoder. When compared to the radix-16 OMA described before, estimated to use $48\lceil \frac{m_a}{4} \rceil + 21$ CLBs, the precomputation of multiples reduces the area by 45% for $m_a = 32$. The on-line delay and the cycle time, assuming a pipelined design, remain the same as in the previous radix-16 OMA design. with equivalent cycle time. This scheme has the disadvantage of needing to be reconfigured for different values of coefficient a. The reconfiguration time is considerable and we don't have an estimate of its effect on the performance. This approach becomes preferable if FPGAs with fast and partial reconfiguration are available.

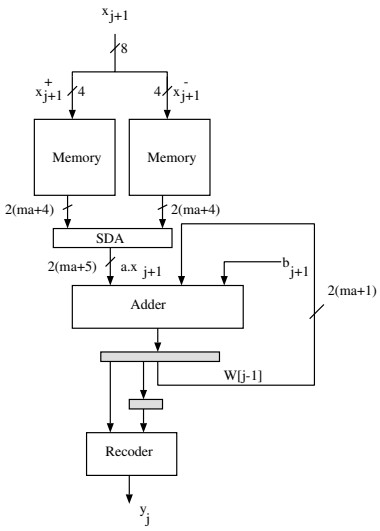

Figure 7: Radix-16 OMA Module with Precomputed Multiples

6 Using the on-line module to compute linear recurrence equations

On-line modules can be used effectively to solve a system of linear recurrences [10]. An M^{th} *order linear recurrence system* of N equations, $R < N, M >$, with $M \leq N-1$ is defined in [11] as:

$$x_i = 0 \text{ for } i \leq 0$$

$$x_i = c_i + \sum_{j=i-M}^{i-1} a_{ij} x_j$$

For example, consider the situation $R < 4, 2 >$:

$$\begin{array}{lcllll} x_1 & = & c_1 & & & \\ x_2 & = & c_2 & +a_{21}x_1 & & \\ x_3 & = & c_3 & +a_{31}x_1 & +a_{32}x_2 & \\ x_4 & = & c_4 & & +a_{42}x_2 & +a_{43}x_3 \end{array} \tag{4}$$

The computation graph for this system of equations and an example of the system organization for $R < N, 2 >$ is shown in Figure 8.

With $M \times N$ OMA modules, a linear array solves $R < N, M >$ in $T_{(N,M)} = (N-1)\delta_2 + n - 1$ radix-r digit cycles. A linear array scheme using conventional radix-r multiply-add modules would require $O(N \times n)$ cycles. If fewer modules are implemented, the performance degrades correspondingly. If, for example, we can use concurrently only 2 OMA modules, the performance of the on-line scheme matches that of a corresponding conventional implementation. The key advantage of the on-line approach is that it provides an easy way to implement user-directed variable-precision computations by including the corresponding number of digit slices. For a precision of $m' < m$ digits, only about m' digit-slices are needed. As an illustration of the flexibility when on-line modules are used, we consider the EVC FPGA board[15] with one Xilinx 4010 FPGA chip with 400 CLBs,

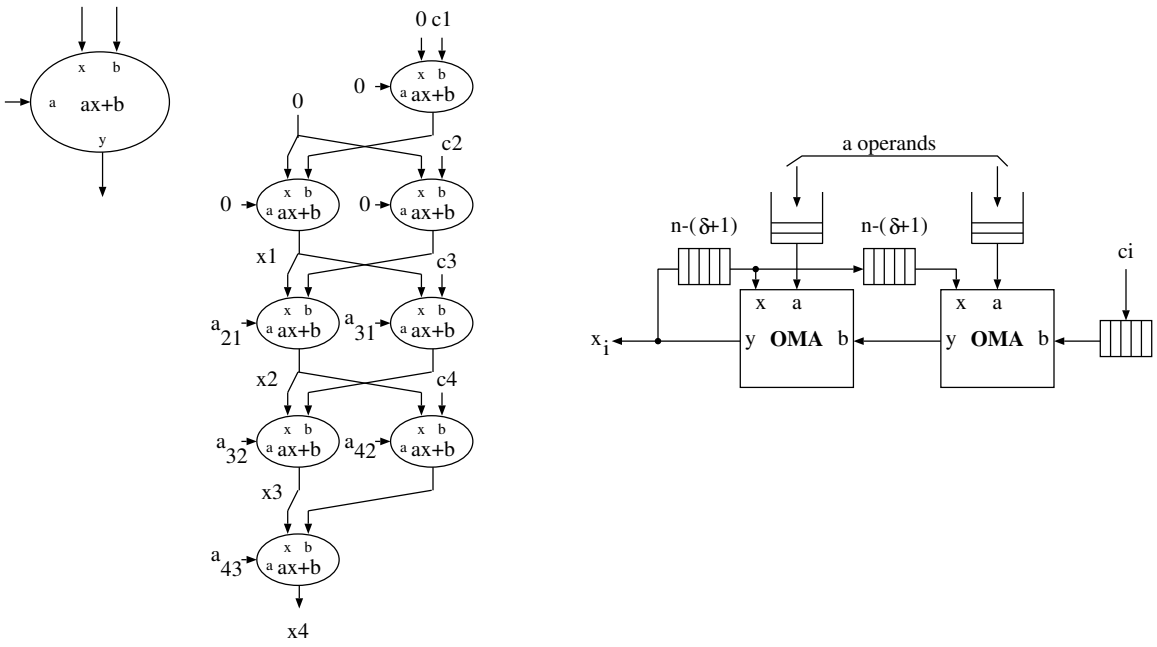

Figure 8: Computational graph and linear recurrence structure for $R < N, 2 >$

	r			
	2	8	16	32
#OMAs/chip	10	4	3	2
Total Time (cycles)	51	34	31	52

Table 3: Number of OMAs and cycles to compute $R < 5, 2 >$ in the EVC1

a local memory (LM) and a system bus interface to the host computer (HC). Table 3 indicates the approximate mapping and number of cycles to solve a $R < 5, 2 >$ system with $m = 32$ and $m_a = 8$. The radix-16 OMA design is the one that provides the best performance.

7 Conclusion

The paper presents the tradeoffs and issues in the design of on-line multiply-add modules using higher radices. We discuss when radix-2 designs on-line modules can be replaced by high-radix design with advantages. We show that speedups larger than 1.5 can be obtained at a reasonable cost. As expected, the advantage of using higher radix on-line designs becomes more significant for large precision computations. In this case the cost of the implementation is much more attractive than that of a radix-2 implementation. We also discuss the use of on-line multiply-add modules in solving linear recurrences.

Acknowledgements. We thank S. Casselman and J. Schewel of Virtual Computer Corporation, and S. Kelem of Xilinx for interest and support.

8 REFERENCES

[1] Bajard, J-C., Duprat, J., Kla, S., and Muller, J-M.; Some Operators for On-Line Radix-2 Computations;*J. of Parallel and Distributed Computing*, Vol. 22, 1994, pp. 336-345.

[2] Bickerstaff, K'A. C., Schulte M., and Swartzlander, E. E.; Reduced Area Multiplier; *Proceedings IEEE Application Specific Array Processors*; 1993, pp. 478-489

[3] Daumas, M.; Muller, J-M. and Vuillemin, J.; Implementing On Line Arithmetic on PAM; *Field-Programmable Logic: Architectures, Synthesis and Applications; Proceedings of the 4th International Workshop on Field-Programmable Logic and Applications, FPL'94*, pp. 196-207.

[4] Ercegovac, M. D. and Lang T.; On-line Arithmetic: A design Methodology and Applications; *IEEE Workshop on VLSI*, 1988.

[5] Ercegovac,M.D., Muller,J.-M., and Tisserand,A.; FPGA implementation of polynomial evaluation algorithms. *Proc. SPIE on Filed Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing*, volume 2607, pages 177–188, 1995.

[6] Ercegovac,M.D. and Lang, T.; Fast arithmetic for recursive computations. *Proc. IEEE Workshop on VLSI Signal Processing*, pages 14–28, 1992.

[7] Ercegovac, M. D. and Lang, T.; *Division and Square Root: Digit-Recurrence Algorithms and Implementations*; Kluwer Academic Publishers, 1994.

[8] Ercegovac, M. D.; On-Line Arithmetic: An Overview;*SPIE*, Vol. 495 Real Time Processing VII, 1984.

[9] Fernando, J.S. and M.D. Ercegovac.; Conventional and on-line arithmetic designs for high-speed recursive digital filters; *J. of VLSI Signal Processing*, 7:189–197, 1994.

[10] Grnarov, A. L. and Ercegovac, M. D.; VLSI-Oriented Iterative Networks for Array Computations; *Proceedings of the 1980 IEEE International Conference on Circuits and Computers*; New York.

[11] Kuck, D. J.; *The structure of Computer and Computations*; John Wiley & Sons, 1978

[12] Skaf, A. and Guyot, A.; VLSI Design of On-Line Add/Multiply Algorithms; *IEEE International Conference on Computer Design*; October 1993, pp. 264-267.

[13] Tu, P.K.-G. Tu and Ercegovac,M.D.; Gate array implementation of on-line algorithms for floating-point operations. *J. of VLSI Signal Processing*, (3):307–317, 1991.

[14] Tullsen,D. and Ercegovac,M.D.; Design and implementation of an on-line algorithm. *Proc. SPIE Conference on Real-Time Signal Processing*, San Diego, August 1986.

[15] *Engineers' Virtual Computer, EVC1s Users Guide*; Virtual Computer Corporation

[16] Xilinx Manuals