Design and evaluation of on-line arithmetic for signal processing applications on FPGAs

Reto Galli and Alexandre F. Tenca

Electrical & Computer Engineering Dept., Oregon State University, Corvallis, OR, USA

ABSTRACT

This paper shows the design and the evaluation of on-line arithmetic modules for the most common operators used in DSP applications, using FPGAs as the target technology. The designs are highly optimized for the target technology and the common range of precision in DSP. The results are based on experimental data collected using CAD tools. All designs are synthesized for the same type of devices (Xilinx XC4000) for comparison, avoiding rough estimates of the system performance, and generating a more reliable and detailed comparison of on-line signal processing solutions with other 'state of the art' approaches, such as distributed arithmetic. We show that on-line designs have a hard stand for basic DSP applications that use only addition and multiplication. However, we also show that on-line designs are able to overtake other approaches as the applications become more sophisticated, e.g. when data dependencies exist, or when non constant multiplicands restrict the use of other approaches.

Keywords: On-line Arithmetic, Signal Processing, DSP, FPGA, Xilinx XC4000

1. INTRODUCTION

One of the most important design considerations for signal processing applications on *Field Programmable Gate Arrays* (FPGAs) is the type of arithmetic that is used for the implementation. The manufacturers of FPGAs document in their application notes signal processing solutions using different kinds of bit parallel, bit serial or mixed serial/parallel arithmetic.[1–3] Some researchers proposed the use of on-line arithmetic[4] for FPGA solutions for signal processing applications.[5–8]

Although it was shown that on-line arithmetic has interesting properties for DSP on FPGAs, nobody showed so far how on-line arithmetic competes with other 'state of the art' implementations on FPGAs. We will show in this work how on-line modules can be implemented efficiently on FPGAs, and we will compare realizations of algorithms built from these modules with DSP building blocks that are available as *Intellectual Property* (IP) from an FPGA vendor and with networks that are built with arithmetic IP components. The criteria for this comparison are computation latency, throughput and chip area.

On-line Arithmetic algorithms take the input operands in a digit-serial manner, *most significant digit* (MSD) first. The result, which is also produced in digit serial manner, MSD first, is obtained after a short delay. An overview of On-line Arithmetic can be found in Ref. 4. One advantage of these algorithms is that all arithmetic operations can be processed in the same direction, therefore it is possible to overlap (pipeline) consecutive operations that could not be pipelined using conventional digit serial algorithms. The second main advantage is that the computation can be stopped after the desired precision is attained. In conventional multiplication of two N-digit numbers, a $2N$-digit result is produced, and in usual signal processing calculations, the least significant half of the result is never used.

Conventional serial designs have a delay of 1 clock cycle. On-line arithmetic modules produce the MSD of the result after $\delta + 1$ clock cycles, where δ is called on-line delay and is usually a small constant integer. That means, to generate the j^{th} digit of the output, $j + \delta$ digits of the input operands are needed. It is possible in most arithmetic operations that less significant digits of the inputs can change all previous digits of the result. An easy example for this is the addition $100001 + 99999$ where the carry from the right most position propagates through the whole result and changes all previous digits including the MSD. To avoid this problem and produce the MSD of the result before the complete input operands are known, on-line arithmetic uses *redundant number systems* (RNS). One of the most used RNS is *signed digit*[9] (SD), a weighted radix-r number system, where a fixed point fractional number is represented by $\sum_{i=1}^{n} a_i r^{-i}$ with the digits a_i in the digit set $\mathcal{D} = \{-(r-1), \ldots, 0, \ldots, (r-1)\}$. It is common to work with fractional numbers in on-line arithmetic to simplify the alignment of the operands and to make them compatible with all operations.

Authors' E-mail: reto@orst.edu, tenca@ece.orst.edu

2. ON-LINE OPERATORS

To implement DSP algorithms using on-line arithmetic we need modules for all common operations in such algorithms. The modules we used for this work were on-line addition and on-line multiplication of two serial operands, multiplication and *multiply and accumulate* (MAC) of a serial and a parallel operand,* and division of two parallel operands (SRT-division[10]). We used SRT-division instead of on-line division to simplify the normalization of the operands and therefore to simplify the control of the network of modules. For low precision operands (as common in DSP) the use of SRT-division does not add significantly more delay compared to the use of on-line division. The parallel inputs of the multiplication and the MAC modules are either constants, such as filter coefficients, or inputs of the system that are available in parallel. All modules produce the result in a digit serial manner, MSD first.

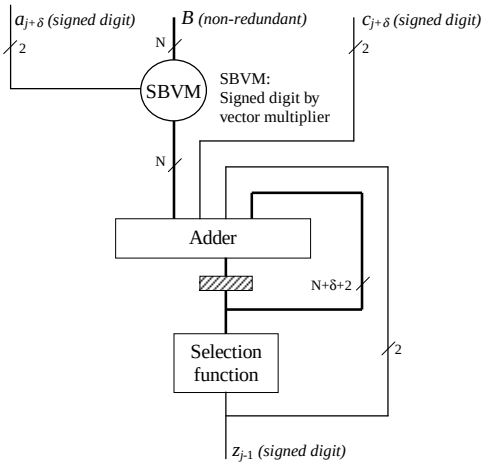
Besides addition, all operators were implemented based on a recurrence equation to calculate an internal residue $W[j]$, and on a selection function to select the result digit z_j, where j is the step in the algorithm. The recurrence equation is a function of the previous residue $W[j-1]$, the result produced up to this point ($Z[j-1] = \sum_{i=0}^{j-1} z_i r^{-i}$) and the received digits of the operands ($A[j] = \sum_{i=1}^{j+\delta} a_i r^{-i}$, $B[j], \dots$). It should contain only addition, subtraction, multiplication with one digit, and shift operations for the algorithm to be efficient. Table 1 shows the recurrence equations for the above mentioned operations. The methodology on how to derive these equations was presented in Ref. 11 and Ref. 12. Figure 1 shows the block diagram of the MAC module as an example of how such an on-line algorithm can be implemented together with a numerical example for radix $r = 2$ and on-line delay $\delta = 2$.

Table 1. Recurrence equations for different on-line operators and SRT-Division

Operator	*Recurrence Equation*
Multiplication $Z = A \cdot B$	$W[j] = r \cdot (W[j-1] - z_{j-1}) + r^{-\delta} \cdot (a_{j+\delta} \cdot B[j-1] + b_{j+\delta} \cdot A[j])$
Multiplication $Z = A \cdot B$ with B in parallel	$W[j] = r \cdot (W[j-1] - z_{j-1}) + r^{-\delta} \cdot (a_{j+\delta} \cdot B)$
MAC $Z = A \cdot B + C$ with B in parallel	$W[j] = r \cdot (W[j-1] - z_{j-1}) + r^{-\delta} \cdot (a_{j+\delta} \cdot B + c_{j+\delta})$
SRT-Division $Q = N/D$ (not on-line)	$W[j] = r \cdot W[j-1] - q_j \cdot D$

The on-line delay and the selection function are defined based on further design choices such as the type of the adder that is used to calculate the residue and the radix. We will discuss these parameters in Sect. 3.

As mentioned before, addition is the only module that is not implemented using a recursion. To perform on-line addition in a radix-2 SD RNS, a scheme using two full-adders, inverters and registers can be used.[13]



$W[j] = 2 \cdot (W[j-1] - z_{j-1}) + 2^{-2} \cdot (a_{j+2} \cdot B + c_{j+2})$

$z_j = -1$ if $W[j] < -1/2$, 0 if $-1/2 \le W[j] < 1/2$, 1 if $W[j] \ge 1/2$

$A = 0.101\bar{1}\bar{1}01 = 0.5390625$, $B = 0.1101010 = 0.8281250$

$C = 0.110\bar{1}0\bar{1}1 = 0.6796875$, $Z = 1.1\bar{1}\bar{1}0000011\bar{0}1\bar{1}0 = 1.1260986328125$

j	a_j	$A[j]$	B	c_j	$C[j]$	$W[j]$	z_{j-1}	$Z[j]$
–1	1	0.1	0.1101010	1	0.1	0.011101010		
0	0	0.10	0.1101010	1	0.11	1.001010100		
1	1	0.101	0.1101010	0	0.110	0.100010010	**1**	1.
2	–1	0.1001	0.1101010	–1	0.1011	–1.011000110	**1**	1.1
3	–1	0.10001	0.1101010	0	0.10110	–0.111110110	**–1**	1.01
4	0	0.100010	0.1101010	–1	0.101011	–0.001101100	**–1**	1.001
5	1	0.1000101	0.1101010	1	0.1010111	0.000010010	**0**	1.0010
6		0.1000101	0.1101010		0.1010111	0.000100100	**0**	1.00100
7		0.1000101	0.1101010		0.1010111	0.001001000	**0**	1.001000
8		0.1000101	0.1101010		0.1010111	0.010010000	**0**	1.0010000

Figure 1. Standard implementation of an on-line MAC (left) and numerical example for MAC algorithm (right)

*Where the parallel operand is known in the first clock cycle of the operation or at implementation time.

3. FPGA IMPLEMENTATION OF ON-LINE MODULES

When implementing on-line algorithms on FPGAs and comparing these implementations with other comparable (digit serial) designs it became obvious that we can not achieve the desired performance in terms of area and throughput using the straight forward designs presented in the literature. It is necessary to perform technology specific optimizations. To perform these optimizations, the developer is required to have detailed knowledge about the target technology and the synthesis tools that are used. All on-line modules were described using VHDL code and synthesized using FPGA Express, that comes with the Xilinx Foundation implementation tools. Our target hardware was Xilinx XC4000E and XC4000XL chips.[14]

All implementations presented in this paper work with radix-2 and the digit set $\mathcal{D} = \{-1, 0, 1\}$. As bit-level representation of the digits we used either *borrow-save* (BS) $[a^+a^-] = a^+ - a^-$ or *two's complement* (TC) $[a_1a_0] = -2 \cdot a_1 + a_0$.

3.1. Optimizations

The main goal of the optimizations is to modify the algorithms and the logic functions in a way that they better fit into the very regular FPGA structure. Xilinx FPGAs use 4-input *look-up tables* (LUTs) to realize logic functions. At the output of every LUT is one register available. This means that logic functions have to be partitioned into these tables. However, the synthesis tools do not always find the best partitioning,[15] in this case it is necessary to provide the partitioning or to bring the functions in a different form, so that they can be partitioned more efficiently. This can reduce the levels of logic in a critical path and therefore, it will allow a higher clock rate, or it can reduce the area of the design.

An example for such a partitioning problem is the Append-Register with integrated OFC, a component that is used in several on-line modules to accumulate incoming digits.[16,17,12] One of the functions that is used several times in this component can not be partitioned efficiently as long as digits are represented in BS form. However, using TC representation for the digits, this function slightly changes and therefore it can be better partitioned. The component for SD in TC representation needs 30% less area than the one for the BS case. If a BS representation is used in the overall design, it is better to convert the digits before sending them to the Append-Register.

Most FPGA architectures have, beside programmable resources to realize logic functions, also other, more specialized, programmable resources. Xilinx as well as Altera FPGAs provide a *Fast Carry Logic* (FCL) to realize *Ripple Carry Adders* (RCAs). ASIC designs of on-line algorithms always use redundant adders, such as *Carry Save Adders* (CSA) or *Signed Digit Adders* (SDA) to calculate the residue. That means, the residue is also stored in a redundant form. To realize a one-level adder[†] of these types we need one or two *Full Adders* (FA) per bit slice, depending on the representation of the vector that is added to the residue. The implementation of a FA needs one CLB (1 LUT for the sum and 1 LUT for the carry) in a Xilinx XC4000 chip. To realize a one-level RCA using the FCL on the same chip needs only half a CLB per bit slice and therefore it brings an area reduction of 50% for the residue adder. However, this reduction comes for the price of some additional delay from the carry chain of the RCA.

3.2. Final Designs

Based on the optimization methods described above we modified all on-line modules to use fast and area efficient FPGA structures. These modules were then used to implement DSP algorithms on FPGAs. While in many computing applications a precision of 32 or 64 bits is nowadays standard, precisions of 8, 12 or 16 bits are widely used in DSP. We optimized our designs especially for these low precisions, for higher precision different designs should be used.

3.2.1. On-line Multiplier with Parallel Input

This module realizes the function $Z = A \cdot B$, where B is available in parallel. As can be seen in the recurrence equation (Tab. 1) this module needs a one-level adder to calculate the residue. Besides of reducing the area, the use of an RCA to compute the residue also means that the residue will be stored in a non-redundant form and therefore the selection function becomes simpler. In the case of this module, the selection function for a non-redundant residue can be realized using only one level of logic compared to two levels in the redundant case. Thus, there is less delay in the critical path of the module. Furthermore, the use of a a non-redundant residue reduces the on-line delay δ by 1, which reduces the necessary precision of the adder and therefore the carry chain. The logic level that can be saved in the selection function compensates the additional delay of the carry chain up to 12 bits of precision of operand B.

[†]One level adder refers to an adder that adds one more input vector to the previous residue.

The critical path of this design for precisions between 12 and 16 bits is not in the recursion feedback path but from the input $a_{j+\delta}$ over the bit-multiplier into the adder. If the parallel multiplicand B is available at implementation time, this delay can be reduced using a different type of bit-multiplier. The conventional bit multiplier produces the -1 multiple of B by inverting all bits of B and sending a 1 to the least significant position of the residue adder. If B is known at implementation time the -1 multiple of B can be pre-computed and a multiplexer can be used to chose the multiple. This reduces the precision of the adder again by one bit position and the bit-multiplier becomes slightly faster because the synthesis tools can use the optimizations for multiplexers.

This final design of the multiplier using an RCA is faster than a version using a CSA for precisions of B up to 14 bits and only slightly slower for 16 bits. If B is known at implementation time the design can run at a clock rate that is about 10% faster compared to the design where B is not known previously. The overall area of the design could be reduced by 45% compared to designs using a redundant adder. The on-line delay turns out to be $\delta = 1$ for this configuration.

3.2.2. On-line MAC with one Parallel Input

This module computes the function $Z = A \cdot B + C$, where B is available in parallel. If the module would be implemented as shown in Fig. 1 we would need a two-level adder to calculate the residue. However, it is possible to include the addition of $c_{j+\delta}$ into the bit-multiplier. This increases the size of the multiplier by only one CLB and does not add another level of logic. Once we have an one-level adder, we can perform the same optimizations as for the multiplier with B in parallel and get similar results. The final version of the MAC module has an on-line delay $\delta = 2$. That is one clock cycle less than a network consisting of the previously shown multiplier and a consecutive on-line adder. The area of the MAC is also less than a multiplier plus an adder.

3.2.3. On-line Multiplier

The block diagram of an on-line multiplier is shown in Fig. 2. The additional input *token* has to be activated when the module receives the first digits. It is used to enable the selection function and to synchronize the Append-Register components. They accumulate the incoming digits and if a CSA or an RCA is used to calculate the residue, they convert them into a non-redundant TC number[‡]. The implementation of the Append-Register component is discussed in Ref. 12. As mentioned before a version working with SD in TC representation is preferable.

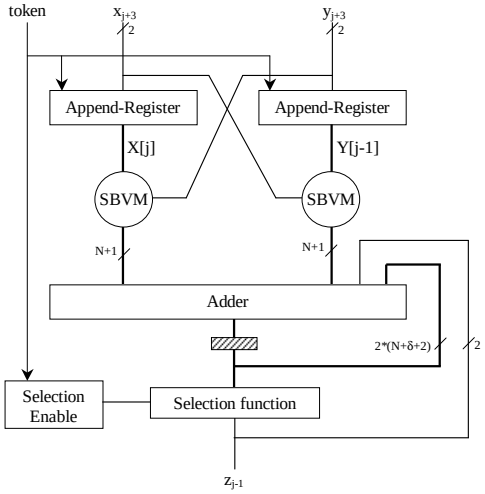


Figure 2. Block diagram for on-line multiplier

It was shown in Ref. 12 that a version using a SDA to calculate the residue is slower and uses more area than a version using a CSA. Therefore we do not consider the SDA case. The use of an RCA would also be preferable to reduce the area. However, a complete reduction of all inputs of the recurrence equation into a non-redundant residue would require a two-level adder, which is not acceptable for its critical path. A possible solution for this problem

[‡]Not to confuse with TC representation of a signed-digit

is to store the residue in a redundant *Carry Save* (CS) form, but using one RCA to produce the W_S vector and another one to produce the W_C vector instead of a CSA. Careful considerations of the adder designs make it possible to realize this residue adder using a $N+5$ and a $N+2$-bit RCA. The use of this special RCA configuration brings an area reduction of about 20% compared to a design using a CSA. However, we do not have the simplification of the selection function, because the residue is still stored in CS form. Furthermore, the selection function is not in the critical path of the multiplier, which means that it would not compensate the loss in speed even if it would be simplified.

The critical path of the on-line multiplier is not the recursion as often assumed. It is the path from the $x_{j+\delta}$ input over the Append-Register, the SBVM and the residue adder. The path from the $y_{j+\delta}$ input is not a problem because only $Y[j-1]$ is used to calculate the residue. That means we can use the registered output of the Append-Register instead of the immediate output as we have to use for $X[j]$. This path is not only critical because of the levels of logic, but also because of the high fan-out of the net into the Append-Register and the SBVM, which causes a high routing delay in FPGAs. Because the worst delay is not in the residue feedback path, but on the input path, it can be reduced by the use of two pipeline stages.

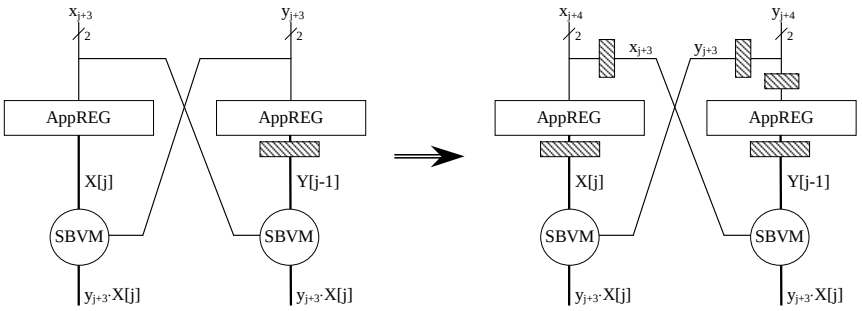


Figure 3. Input stage of ol-MULT without (left) and with pipelining (right)

Figure 3 shows the pipelined input stage of the on-line multiplier. The registers at the output of the Append-Register shown in this Figure are the ones inside the Append-Register. They are just shown to illustrate the pipelining. All additional FFs are at the 2-bit wide digit inputs and therefore do not add a significant amount of area. The high fan-out of the $y_{j+\delta}$ net can be reduced when using two registers, one for the Append-Register and one for the SBVM.

This modification increases the overall delay of the on-line multiplier by one clock cycle. The on-line delay of this configuration turns out to be $\delta = 4$.

3.2.4. SRT-Divider

The divider module has by far the most levels of logic in the critical path. Therefore, it is likely to be the limiting module for clock rate in a network of arithmetic modules. Area optimizations that would add more delay into the critical path, such as the use of an RCA, can not be used for this module. The required precision for the selection would not allow to reduce the levels of logic in the selection function if the residue is in a non-redundant form. Thus, the best choice for this module is to use a one-level CSA to calculate the residue.

The necessary normalization of the inputs can be performed in the component that parallelizes the inputs. This is one of the main advantages of SRT-division compared to on-line division. For on-line division we would need a quasi-normalizer for the inputs, which has a variable processing time. This would complicate the control of networks of on-line modules significantly. For short precision the use of SRT-Division does not add a significant amount of clock cycles. Furthermore, SRT-division runs on a 30% higher clock rate than on-line division. Considering the fact that the division is the slowest component, this means that the whole network of modules can run on a 30% higher clock rate. This significant speed up compensates the additional clock cycles in the overall latency of a network.

3.3. Implementation Results

Table 2 shows the implementation results of the modules for Xilinx XC4000E and XC4000XL chips. The given precision refers to the precision of a non-redundant representation of the operands. A signed-digit number uses

Table 2. Implementation results for arithmetic modules

		8 bit	12 bit	16 bit
MAC with B in parallel	area	12 CLBs	14 CLBs	16 CLBs
	clock XC4000E-1	94 MHz	84 MHz	76 MHz
	clock XC4000XL-09	104 MHz	95 MHz	86 MHz
Multiplier with B in parallel	area	10 CLBs	12 CLBs	14 CLBs
	clock XC4000E-1	91 MHz	80 MHz	72 MHz
	clock XC4000XL-09	103 MHz	96 MHz	89 MHz
Multiplier	area	48 CLBs	68 CLBs	88 CLBs
	clock XC4000E-1	77 MHz	69 MHz	63 MHz
	clock XC4000XL-09	89 MHz	85 MHz	79 MHz
SRT-Division	area	19 CLBs	27 CLBs	36 CLBs
	clock XC4000E-1	72 MHz	72 MHz	70 MHz
	clock XC4000XL-09	82 MHz	82 MHz	78 MHz

$N-1$ digits to represent the same range of numbers as a signed N-bit TC number. The size of the modules with one parallel input depends on the precision of this particular input. Only the size of the on-line multiplier depends on the precision of the serial inputs that are processed.

3.4. Other modules

To implement networks of on-line modules, delay components to align operands and interface components to connect to other circuits are necessary.

3.4.1. Serial-in/Serial-out Shift-Register

The implementation of conventional serial-in/serial-out shift registers using flip-flops in FPGAs is very area consuming. Xilinx gives in one of their application notes[18] a solution to this problem with the use of the RAM available on their chips.

3.4.2. Interface components

In most cases, circuits will provide inputs and expect outputs in a non-redundant TC form. The conversion of a TC number to a SD number is straight forward. The MSD has a negative weight and all following digits are positive. It is also possible to eliminate the MSD and combine it with the following ones using a simple state machine.[19]

To convert the outputs from signed-digit to TC representation an *On-The-Fly Converter* (OFC)[20] can be used. However, it is much more area efficient to use a simpler converter designed as an RCA to add or subtract the incoming digits.[19] Although this converter is not as fast as the OFC, it is still faster than all the on-line modules presented previously.

4. COMPARISON OF ON-LINE SOLUTIONS WITH OTHER APPROACHES

We implemented two typical DSP algorithms using the previously presented modules and compared these implementations with IP building blocks provided by an FPGA vendor or with networks built with IP arithmetic modules. We designed for both algorithms the complete network of modules including the control structure, using the design methodology for networks of on-line modules presented in Ref. 19 and described in more details in Ref. 21. They were synthesized, simulated, placed and routed for the target hardware, to obtain accurate figures rather than estimations of the system performance.

The first algorithm is a transversal structure of an *Finite Impulse Response* (FIR) Filter. We chose this type of filter, because it is the only one where a digit serial IP version, using *Serial Distributed Arithmetic* (SDArith), is

available for comparison. Other FIR structures, such as Lattice Filters can not be implemented using SDArith. We would have to compare a Lattice structure with a *Parallel Distributed Arithmetic* (PDA) version, which would not be meaningful.

For the second algorithm we chose a DSP problem that shows less parallelism, more data dependencies and contains division. It is the Levinson-Durbin Recursion in application of Yule-Walker Power Spectrum Estimation.[22] There are no complete building blocks for this algorithm available. Therefore, we compared it to a network that was built with IP arithmetic modules.

4.1. Criteria for Efficiency of the Design

To compare on-line designs with others, we have to define the criteria for efficiency of a design. The efficiency of an arithmetic implementation can be measured by three numbers: latency, throughput, and area. We will use the following definitions for these three figures in all design comparisons:

- *Latency*: The latency is the number of time units between applying the inputs and getting the output. In the literature about on-line arithmetic, latency is often expressed as the on-line delay δ. But this definition is not compatible with other kinds of arithmetic. Thus, we will use the time between a parallel input and a parallel output if it is applicable, or, if the inputs and outputs are in serial form, between the first digit of the input and the last digit of the output.
- *Throughput*: The throughput of a design is defined as the number of input sets the design can process per time unit. In on-line designs, the throughput is limited by the component that is busy for the most clock cycles to process the inputs and produce the output.
- *Area*: The area gives the number of area units in the chip that are occupied by the design. It is a measure for costs and for power consumption of a design. For FPGA designs, area is mostly expressed as the number of CLBs.

4.2. FIR Filters

A transversal structure of an FIR-filter follows directly the difference equation $y(n) = \sum_{k=0}^{M-1} b_k \cdot x(n-k)$.

We implemented two different versions of FIR Filters. The first one uses MAC modules. It has a constant on-line delay of $\delta = 2$ that is independent of the filter length. The second version uses multiplier modules and an adder tree to accumulate the results. The on-line delay of this version is dependent on the filter length, but it shows a simpler control structure. Figure 4 shows a 4-tap version of the MAC structure and a 8-tap, symmetric version of the multiplier and adder structure.

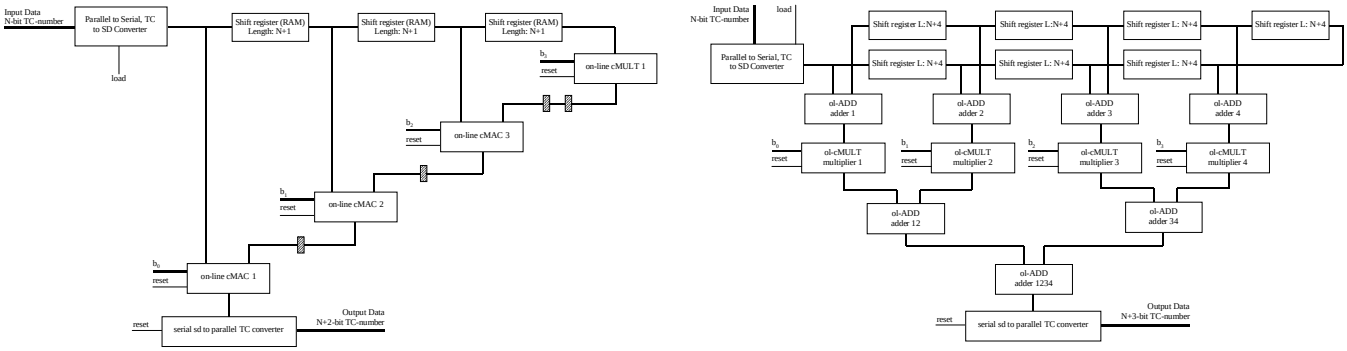


Figure 4. FIR Filter: MAC structure (left) and multiplier and adder structure (right)

The version using MAC modules can process one input every $N+5$ clock cycles and the version using multipliers and adders can process one every $N+4$ clock cycles, where N is the precision of the data samples. Table 3 shows the implementation results for different precisions and filter lengths of these two approaches. Note that these designs

Table 3. Implementation results for on-line FIR Filters

		Design using MAC modules		Design using MULT and ADD modules		
taps coefficient length		4, non-symm 8 bits	8, non-symm 8 bits	8, symm 8 bits	16, symm 8 bits	8, symm 12 bits
area [CLBs]		60	110	84	149	98
clock [MHz], data rate [MS/s]	XC4000E-1	87, 6.7	76, 5.8	85, 7.0	85, 7.0	79, 4.9
	XC4000XL-09	94, 7.2	88, 6.7	96, 8.0	93, 7.7	90, 5.6

include the parallel to serial and serial to parallel converters. An 8-bit design without these converters would use 19 CLBs less and a 12-bit design without them would use 25 CLBs less.

The SDArith FIR Filter IPs Xilinx offers[23] uses about 60% less area than our on-line designs. They run at about the same clock rate. Nevertheless, due to the fact that it is possible in the SDArith approach to produce the result in parallel, these filters can process one input sample every $N+1$ clock cycles. Therefore, the SDArith FIR Filter has a higher throughput than the on-line FIR Filter of about 50% for 8-bit designs and 30% for 12-bit designs. The IP specification does not give the latency. Tests showed that an 8-bit, 8-tap symmetrical filter takes 21 clock cycles to produce the output $y(n)$ after $x(n)$ was applied. The on-line FIR filters need 11 clock cycles (MAC structure) and 19 clock cycles (multiplier structure). Thus, the latency of this SDArith FIR is about 100% longer than the one of the on-line MAC version and it would increase with the filter length.

4.3. Levinson-Durbin Recursion

The Levinson-Durbin Recursion[22] solves a p^{th}-order linear system in Toeplitz form in p iterations. Each iteration is defined by the equations

$$a_m(m) = -\frac{r_x(m) + \mathbf{r}_{m-1}^{bt} \cdot \mathbf{a}_{m-1}}{r_x(0) + \mathbf{r}_{m-1}^{bt} \cdot \mathbf{a}_{m-1}^{b}} \tag{1}$$

$$a_m(k) = a_{m-1}(k) + a_m(m) \cdot a_{m-1}(m-k) \tag{2}$$

with $k = 1, 2, \ldots, m-1$ and where the superscript t denotes the transpose of a vector and the superscript b denotes the vector with elements in reverse order. The auto-correlation vector is defined as $\mathbf{r}_{m-1}^{t} = [r_x(1), r_x(2), \ldots, r_x(m-1)]$.

The recursion shows a data dependency between $a_m(m)$ and $a_m(k)$ and between $a_m(k)$ and $a_{m+1}(m+1)$. We implemented the first three unfolded iterations of the recursion. Figure 5 shows the block diagram of the on-line design for the Levinson-Durbin Recursion. To obtain a modular solution in term of the order, the design has been split up into three components, one for each iteration. Figure 6 shows the timing and the overlap of the computations for a 12-bit design.

The design receives all inputs $r_x(m)$ over a bus in parallel and uses them in parallel as inputs for multiplier or SRT-Division modules. As mentioned earlier, the use of SRT-division simplifies the control because no quasi-normalizer with variable processing delay has to be used. However, as can be seen in Fig. 6 it adds several clock cycles of delay because there is no overlapping of the division with the previous computations. However, because it runs on a higher clock frequency than on-line division, the whole design can run on a higher clock frequency and the extra clock cycles are compensated. There is less overall latency when using SRT-Division in this case.

Table 4 shows the implementation results for the 12-bit design on a Xilinx XC4036XL-09 chip. The area results can be extrapolated and we can conclude that it should be possible to implement the unfolded Levinson-Durbin Recursion up to 6^{th} order on the largest XC4000 chip.

We realized the same first 3 unfolded iterations of the algorithm using arithmetic components provided by Xilinx as IP.[24,25] These components are all bit-parallel components. There exist no bit-serial arithmetic IP components for XC4000 chips. Anyway, bit-serial arithmetic LSD first could not compete with the presented on-line version for the following reason: (1) The division could not be overlapped with following computations and the first N digits produced by the multiplier would be discarded. Therefore, N cycles of computation would be wasted, what would approximately triple the latency. (2) The area gain that could be achieved by using bit serial LSD first multipliers

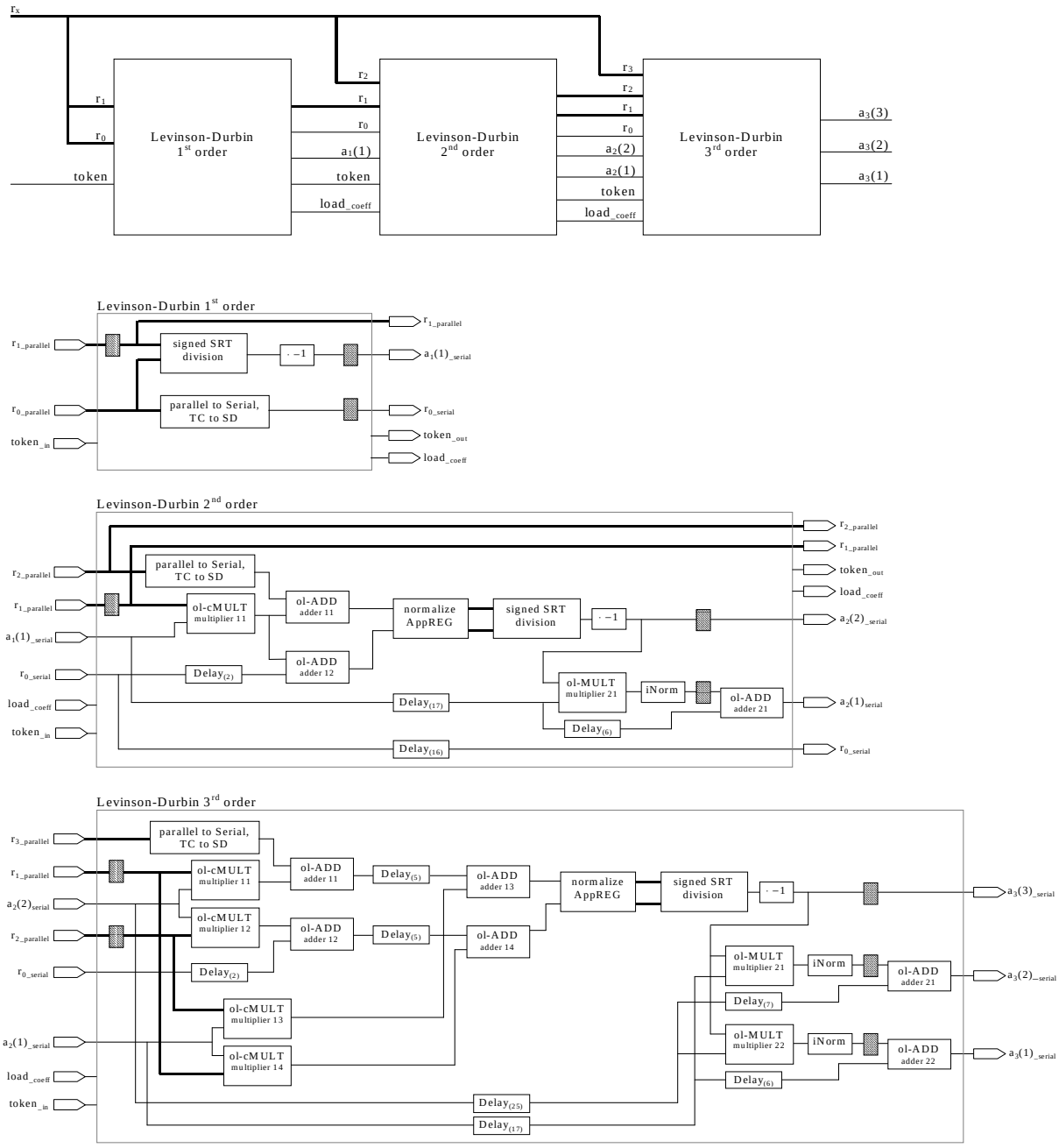


Figure 5. Block diagram for on-line 3rd order Levinson-Durbin Recursion

instead of the area intense on-line multiplier in the calculation of $a_m(k)$, would partially be consumed by the structure that parallelizes and re-serializes the division result, making it as area consuming as the on-line solution.

The bit parallel IP components exist in both area and performance optimized versions. Because of the limited area available on XC4000 chips, it is not possible to use the performance optimized IP components for this design. However, the recursion can be implemented using the area optimized IP components. The area for this design sums up to 1848 CLBs, which is nearly 3 times as much as the on-line design. The total latency is 101 clock cycles, which is mainly because of the latency of $N+12$ clock cycles of the division module. The maximum clock frequency of this design is also about 70 MHz. Therefore, the latency is 101/70 MHz = 1.443μs or 50% more than the latency of the on-line design. The pipelined divider in this configuration takes 1 set of inputs every 8 clock cycles. Thus, it is the

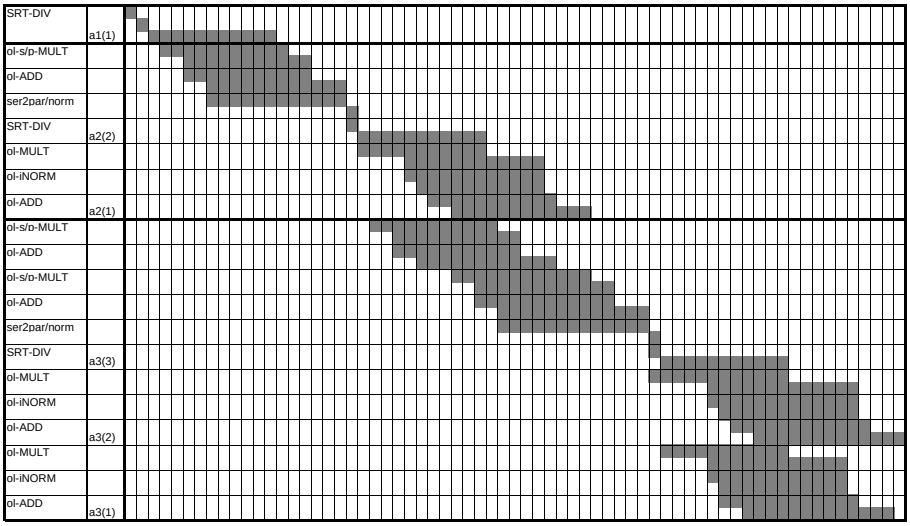


Figure 6. Timing for 12-bit on-line Levinson-Durbin Recursion (3rd order)

Table 4. Implementation results for 12-bit on-line Levinson-Durbin Recursion

	1^{st} iteration	2^{nd} iteration	3^{rd} iteration	total design (3 iterations)
area [CLBs]	51	237	367	678
clock rate [MHz]	76	74	78	71
throughput [10^6 datasets/s]	5.4	4.3	4.1	3.7
latency [clock cycles], [μs]	14, 0.182	38, 0.513	47, 0.602	67, 0.937

limiting component for throughput, which turns out to be $8.75 \cdot 10^6$ datasets/s. Therefore, the throughput of the bit parallel design is a little more than twice the one of the on-line design. It would be possible to implement up to 4^{th} order on the largest XC4000 chip compared to 6^{th} order using the on-line design.

5. CONCLUSIONS

We discussed how on-line arithmetic modules can be implemented efficiently on Xilinx FPGAs. The implementations targeting ASIC designs that are shown in the literature do not necessarily lead to the best FPGA implementation. For precision values up to 16 bits, which are very common in signal processing, implementations using ripple carry adders are often more efficient, mainly because of a significant area reduction. The simplification in the selection function when a non-redundant residue is used compensates the delay that was added with the ripple carry chain.

We implemented two types of FIR Filters using on-line modules. They can not compete with SDArith FPGA FIR filter implementations in terms of area and throughput, but they show better results in terms of latency. For most applications SDArith filters will be the best choice to implement a transversal FIR Filter. However for other types of FIR Filters, such as Lattice Filters, where SDArith can not be used, on-line could be preferable.

The use of on-line arithmetic is clearly a good choice to implement the Levinson-Durbin Recursion on FPGAs. The presented design is a good compromise between area and throughput. 1/3 of the area brings 1/2 of the throughput compared to a parallel design. Other bit-serial designs fail to be efficient because of the least significant part of the multiplication result that has to be calculated and because the division can not be overlapped with other computations. The overlapping of the operations makes the on-line design the one with the least latency.

We could identify certain properties of algorithms which can give a hint to the designer of DSP algorithms on FPGAs if on-line arithmetic could lead to a better design. An on-line design should be considered if the latency of the

result is an important design factor. The on-line designs show the least latency for all our benchmarks. The latency advantage of on-line designs becomes even bigger if the algorithm shows several data dependencies. On-line designs are a good choice if other digit serial arithmetic approaches can not be used efficiently. This is the case if division is used, or if non-constant multiplicands or data dependencies (e.g. Lattice FIR Filter) restrict the use of serial distributed arithmetic. For algorithms that tend to use a large amount of arithmetic modules, e.g. Levinson-Durbin Recursion, on-line arithmetic can bring the necessary area reduction compared to bit parallel designs to make an implementation feasible in terms of area and still maintaining a reasonably high throughput.

We also showed that it can be advantageous to mix on-line with other approaches, e.g. SRT-division in an on-line design. This could also be true for other approaches such as SDArith. On-line Arithmetic is not the answer for every DSP problem on FPGAs but under certain circumstances it shows clear advantages over other approaches.

REFERENCES

1. S. Knapp, "Using programmable logic to accelerate DSP functions," *Xilinx Technical Report*, 1995.
2. G. Goslin and B. Newgard, "16-tap, 8-bit FIR filter applications guide," *Xilinx Application Note*, Nov 1994.
3. K. Chapman, "Constant coefficient multipliers for XC4000E," *Xilinx Application Note*, Dec 1996.
4. M. Ercegovac, "On-line arithmetic: an overview," *SPIE Real Time Signal Processing VII* **495**, pp. 86–93, 1984.
5. M. Daumas, J.-M. Muller, and J. Vuillemin, "Implementig on line arithmetic on PAM," *Proceedings 4th International Workshop on Field Programmable Logic and Application FPL94*, pp. 112–118, Sep 1994.
6. A. Tenca and S. Hussaini, "A design of radix-2 on-line division using LSA organization," *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pp. 266–273, June 2001.
7. A. Tisserand and M. Dimmler, "FPGA implementation of real-time digital controllers using on-line arithmetic," *Proceedings 7th International Workshop on Field Programmable Logic and Application FPL97*, September 1997.
8. D. Lau, A. Schneider, M. Ercegovac, and J. Villasenor, "A FPGA-based library for on-line signal processing," *The Journal of VLSI Signal Processing-Systems for Signal Image and Video Technology* **28**, pp. 129–143, May 2001.
9. A. Avizienis, "Signed-digit number representation for fast parallel arithmetic," *IEEE Transactions on Electronic Computers* **EC10**, pp. 389–400, 1961.
10. J. Robertson, "A new class of digital division methods," *IRE Transactions on Electronic Computers*, pp. 88–92, Sep 1958.
11. M. Ercegovac and T. Lang, "On-line arithmetic: A design methodology and applications in digital signal processing," *VLSI Signal Processing III*, pp. 252–263, 1988.
12. A. Tenca, "Theory and implementation of radix-2 on-line multipliers," *Oregon State University, in prep.*, 2001.
13. A. Guyot, Y. Herreros, and J.-M. Muller, "JANUS, an on-line multiplier/divider for manipulating large numbers," *Proceedings 9th IEEE symposium on Computer Arithmetic*, pp. 106–111, Sep 1989.
14. Xilinx, "XC4000 FPGA products," *Xilinx Databook (http://www.xilinx.com/partinfo/databook.htm)*, May 1999.
15. N. Camilleri and C. Lockhard, "Improving XC4000 design performance," *Xilinx Application Note XAPP043*.
16. D. Tullsen and M. Ercegovac, "Design and VLSI implementation of an on-line algorithm," *Proceedings SPIE Conference on Real Time Signal Processing IX* **698**, pp. 92–99, 1986.
17. P.-G. Tu, *On-line Arithmetic Algorithms for Efficient Implementation*. PhD thesis, UCLA, Sep 1990.
18. P. Alfke, "Efficent shift registers, LFSR counters, and long pseudo-random sequence generators," *Xilinx Application Note XAPP052* , July 1996.
19. R. Galli, "Design and evaluation of on-line arithmetic modules and networks for singal processing applications on FPGAs," Master's thesis, Oregon State University, June 2001.
20. M. Ercegovac and T.Lang, "On-the-fly conversion of redundant into conventional representations," *IEEE Transactions on Computers* **C-36**, pp. 895–897, July 1987.
21. R. Galli and A. Tenca, "A design methodology for networks of on-line modules to implement DSP algorithms," *Oregon State University, in prep.*, 2001.
22. M. H. Hayes, *Statistical Digital Signal Processing and Modeling*, John Wiley & Sons, Inc., 1996.
23. Xilinx, "Serial distributed arithmetic FIR filter," *LogiCore Product Specification*, July 1998.
24. Xilinx, "Pipelined divider v2.0," *LogiCore Product Specification*, June 2000.
25. Xilinx, "Parallel multipliers - area optimized," *LogiCore Product Specification*, July 1998.