An FPGA Based Architecture for Moving Target Indication (MTI) Processing Using IIR Filters

Ali Arshad, Fakhar Ahsan, Zulfiqar Ali, Umair Razzaq, and Sohaib Sajid

***Abstract*—Design and implementation of an IIR filter based Moving Target Indication (MTI) processor is presented. IIR filters offer an advantage compared to FIR filters in terms of hardware resources but are prone to problems in implementation. These are variable overflows and accumulating error. This results in a need for calculation of range and error bounds for the filter variables. This paper presents a method for dealing with these issues and proposes a time shared architecture for the processor. The design is implemented on a Virtex-4SX35 device on the Xilinx Xtreme DSP kit. The processor is tested using unprocessed baseband data from a working TA-10K air traffic control radar.**

***Index Terms*—Fixed point issues, FPGA; IIR filter; Moving Target Indication (MTI).**

I. Introduction

Moving target indication (MTI) algorithms in radars are used to distinguish the wanted target returns from the unwanted clutter returns. The clutter suppression capability of a radar signal processor is a pivotal performance parameter. Delay line canceller (DLC) processing is the simplest form of MTI. DLCs, though simple in implementation, have a poor response against slowly moving clutter. Higher order filters are needed to shape the filter response appropriately and get higher attenuation. Thus the requirement is to have a filter with required attenuation and minimum computational requirements.

Architecture for achieving the desired objective using high order FIR filters has been proposed in [1]. Use of FIR filters in this case gives the desired characteristics but requires high computational power along with greater memory and memory bandwidth requirements. An alternate option could be the use of IIR filters with much lower order and matching characteristics. This would drastically reduce the computational, memory and memory bandwidth requirements. But there is a downside to IIR filters in that they don't guarantee a linear phase response. Moreover, stability issues have to be looked after. Another very important issue that arises in case of fixed point implementations of IIR filters is the selection of bit widths for both signals and coefficients. Coefficient bit widths are chosen to ensure that the location of quantized poles and zeros is within acceptable limits. Too much deviation of poles, apart from altering the required filter characteristics can render the system unstable. Appropriate signal widths are chosen for each signal to ensure that maximum and minimum values are accommodated and the required precision obtained. This guarantees the required filter characteristics with an optimal utilization of FPGA resources.

We have not come across any literature discussing FPGA implementation of MTI. Xu et al. [2] have proposed a scheme for MTI using a DSP processor-FPGA combination with the DSP processor performing all the arithmetic. However, the arrival of new FPGAs having rich DSP resources has made it feasible to implement the computational part of the processor on the FPGA. Stapleton et al. [3] has discussed possibilities of improving the existing capabilities of radar signal processors using FPGAs. However, no dedicated architecture for IIR or FIR anti-clutter processing has been provided. Various discussions of issues related to FPGA implementations of IIR filters have been discussed in [4-5].

In this paper, we present a scheme for the implementation of an area optimized architecture for an IIR filter based MTI engine. The architecture is built around two 3rd order IIR filters that operate on data streams from different range bins of the radar echo return in a time shared manner. The architecture uses on chip block RAM (BRAM) to store the raw data, intermediate and final results. The filters are built around the Virtex-4 DSP-48 slices.

The rest of the paper is organized as follows. Section 2 outlines various design issues such as the speed requirements and fixed point issues. Section 3 describes the design flow along with the complete architecture details. Results are shown in section 4 and the paper is concluded in section 5.

II. Design Issues

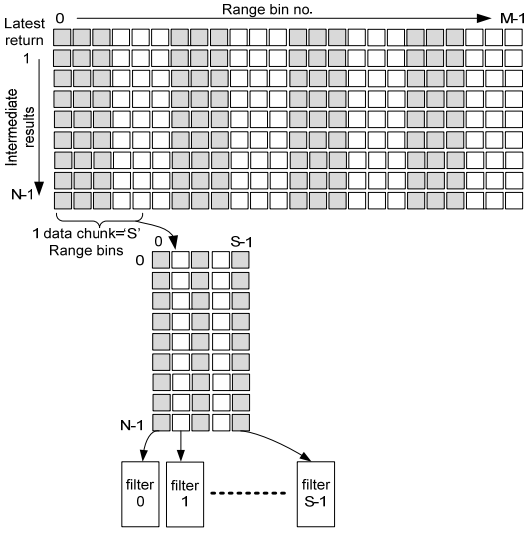

Fig. 1. Data visualization for MTI processing.

From a conceptual point of view, the data storage scheme

Manuscript received August 28, 2012; September 21, 2012.

Ali Arshad, Fakhar Ahsan, Zulfiqar Ali, and Umair Razzaq are with Irtiqa Technologies Islamabad, Pakistan (e-mail: ali@irtiqa.com, fakhar59@hotmail.com, zulfiqar@irtiqa.com, umair@irtiqa.com).

Sohaib Sajid is a student at Nanyang Technological University, Singapore (e-mail: sohaib1@e.ntu.edu.sg).

for MTI filtering in radars may be viewed as in Fig. 1. 'M' represents the number of data samples per radar echo return while 'N' is the filter order. The array of samples in one echo return may be considered to be stored as a single row. Similarly the array of samples from second return is stored in the second row and so on. In case of FIR filters, this forms a data matrix from which column wise chunks of data are picked up for filtering. However, the difference in IIR filters lies in that the data matrix is composed of one row of raw unprocessed data from the latest radar return and 'N-1' rows of intermediate filter results for each range bin. MTI filters run 'vertically' on a range bin to range bin basis. This means that either there are 'M' filters operating in parallel or a lesser number of time shared filters ('S' in Fig. 1) are running.

The latest sample for each filter computation must therefore come from the latest radar return's corresponding bin. This sample is used, along with the previously computed and stored intermediate results to produce a new result. For a filter order of 'N', the processor needs to store one latest radar echo return and N-1 previously computed intermediate results each comprising 'M' samples. Thus there is need to store 'MxN' number of samples at a time in the processor memory buffers. In case of IIR filters, the value of 'N' is generally much lower than that in case of FIR filters and thus it is possible to build this memory using the on-chip block RAM memory resources. The filter used, which is a 3rd order IIR filter with direct form II realization, is shown in Fig. 2. The filter transfer function is given by the following general form:

$$G(z) = \frac{b_0 z^3 + b_1 z^2 + b_2 z + b_3}{z^3 + a_1 z^2 + a_2 z + a_3} \quad (1)$$

A. Speed and Area Requirements

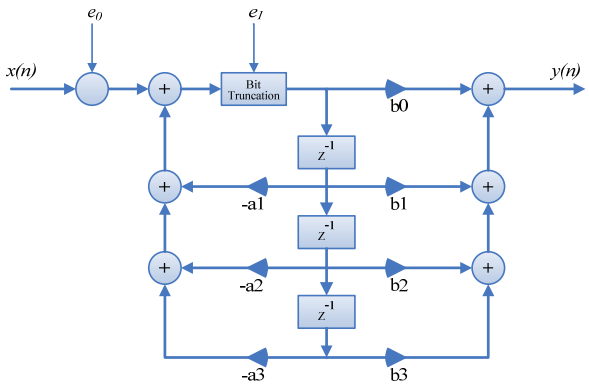

Fig. 2. Structure of 3-tap IIR Filter showing truncation errors e0 and e1.

The processor runs on a total of 'M' range bin samples. For an 'Nth' order filter with direct from II realization, this implies 2N+1 multiplications, 2N additions and N delays peroutput sample. For a total of 'M' range samples per echo return, this implies a total of 'M (2N+1)' multiplications and '2MN' additions. These operations have to be performed inside a time bound which is the time between two successive transmission pulses, known as the pulse repetition interval (PRI). Thus the speed requirements of the system are dependent on the number of range bin samples 'M', filter order 'N' and the PRI of the radar. In our case the value of 'M' and 'N' were 1650 and 3 respectively while the PRI was 1msec. With these values, it might have been possible to go for a DSP processor based implementation but it would not have been possible to simultaneously run the preceding and subsequent radar signal algorithms on the same processor. Aiming for a single chip radar signal processor implementation, we needed a platform that could manage all the algorithms simultaneously. Only FPGAs provided us with this level of flexibility.

B. Fixed Point Issues

Although floating point arithmetic is possible on FPGAs, it is not feasible. The reason for this stems from the fact that floating point units need to be custom built in the FPGA. These are extremely expensive units in terms of FPGA resources which are limited and very precious in our case as we want to implement the complete signal processor on the same chip. All calculations have to be done in fixed point, but this can cause errors. These are coefficient quantization error (caused by representation of coefficients in a finite number of bits), overflow errors (caused by accumulation of numbers of very large magnitude) and truncation errors caused by the truncation of a result to the desired number of bits i.e. the word length of the register holding the result. Thus the choice of bit widths for representation of each signal in the filter is of critical importance. It is essential to choose bit widths that not only prevent the above stated errors but also ensure an optimal utilization of the FPGA resources. The coefficient bit widths are chosen to keep the pole zero perturbations within acceptable limits. Similarly, signal widths are chosen to accommodate the maximum possible values of different signals and ensure that the errors at output are within specified bounds.

1) Coefficient bit width selection

Each filter coefficient has to be represented in a finite number of bits. We have to choose bit widths for both the integer and fraction part of the coefficient. The calculation for the integer bits required for a coefficient 'c' is given by:

$$\lceil \log_2 c \rceil + 1 \quad (2)$$

The calculation for fractional part requires more consideration. Coefficient quantization moves the filter poles and zeros from their ideal locations. This disturbs the filter frequency response. In an extreme case, the coefficients may move outside the z-plane unit circle leading to instability.

As the denominator coefficients are disturbed by Δa_3, Δa_2 and Δa_1, the resulting shift in the location of the poles is given by Eq. 3 [6]:

$$\Delta p_i = -\frac{\sum_{k=1}^{3} p_i^{3-k} a_k}{\prod_{\substack{l=1 \\ l \neq i}}^{3} (p_i - p_l)}, \qquad i=1, 2, 3 \quad (3)$$

If p1, p2 and p3 are the filter poles, the disturbances $\Delta p_1, \Delta p_2,$ and Δp_3 are given by:

$$\Delta p_1 = \frac{p_1^2 \Delta a_1 + p_1 \Delta a_2 + \Delta a_3}{(p_1 - p_2)(p_3 - p_1)} \quad (4a)$$

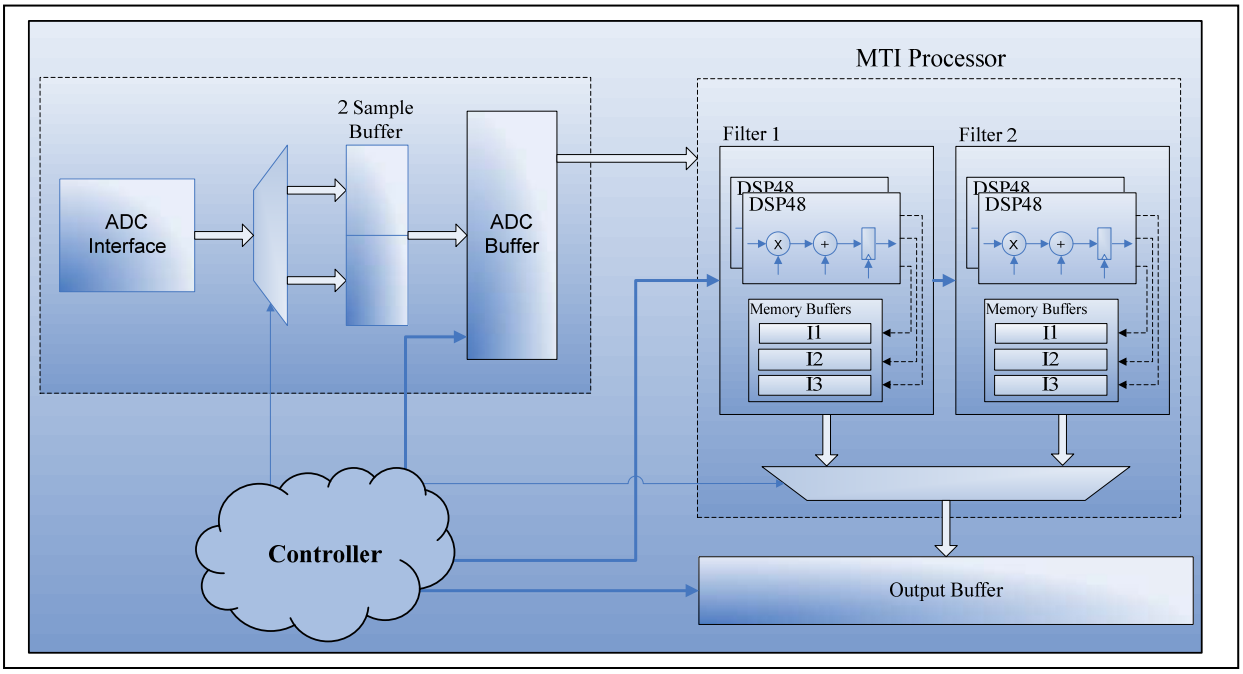

Fig. 3. System block diagram.

$$\Delta p_2 = \frac{p_2^2 \Delta a_1 + p_2 \Delta a_2 + \Delta a_3}{(p_1 - p_2)(p_2 - p_3)} \quad (4b)$$

$$\Delta p_3 = \frac{p_3^2 \Delta a_1 + p_3 \Delta a_2 + \Delta a_3}{(p_1 - p_3)(p_3 - p_2)} \quad (4c)$$

We can see that closely spaced poles have a greater susceptibility to coefficient quantization error. A similar relation also holds for zeros of the system. Having calculated the pole disturbances, we need to quantify the permissible pole perturbations by putting a condition on the amount of pole movement we can afford. This condition given by [7] is expressed relative to the distance from the unit circle, a factor that is absolutely critical to the stability and performance of the system. The condition is given by:

$$\left|\frac{\Delta p_i}{1-|p_i|}\right| < \varepsilon \quad (5)$$

where 'ε' is maximum allowable percent change in pole location with respect to its distance from the unit circle. [7] further gives a sufficient condition for holding Eq. 5 true given here for a 3rd order filter:

$$\Delta a_1 + \Delta a_2 + \Delta a_3 < \epsilon \left|1 - |p_i|\right||p_1 - p_2||p_1 - p_3| \quad (6)$$

If a_1, a_2 and a_3 have the same number of fractional bits, $\Delta a_1 = \Delta a_2 = \Delta a_3$ and the number of bits 'f' required for each of the three coefficients is given by:

$$f \geq \left\lceil -log_2\left(\epsilon \left|1 - |p_i|\right||p_1 - p_2||p_1 - p_3|\right)\right\rceil + 1 \quad (7)$$

Taking the value of ε = 5% for our filter, we obtain a value of 'f' = 11. Thus 11 bits are used to represent the fraction part of the filter coefficients.

2) Range bounds

Range bounds are calculated to ensure that there are no register overflows. We need to know the range of possible values of each variable. For our output variable 'v', the maximum possible absolute value is given by [7].

$$\|v\|_\infty \leq \left\|g_{v,u}\right\|_1 . \|u\|_\infty \quad (8)$$

where 'u' is the system input, '$g_{v,u}$' is the transfer function's impulse response, $\left\|g_{v,u}\right\|_1 \equiv \sum_{k=0}^{\infty}\{|g_{v,u}(k)|\}$ and $\|u\|_\infty \equiv$ max (|u|). Our system transfer function is given by (9):

$$G(z) = \frac{0.8911z^3 - 2.6734z^2 + 2.6734z - 0.8911}{z^3 - 2.7697z^2 + 2.5652z - 0.7941} \quad (9)$$

The value of $\left\|g_{v,u}\right\|_1$ for G(z) is computed to be 2.6305. Our A/D converter is 14-bit with an input voltage range of [-1,+1). Thus the value of $\|u\|_\infty = 1$. From (8), the maximum possible value at the filter output is 2.6305. Therefore, the required number of bits for the integer part of the output is 3, sign bit included.

3) Output error bounds:

The output error is dictated by two types of quantization errors, one due to the finite precision of the ADC (Δy_0) and the other due to truncation of the calculated sum (Δy_{trunc}). In this section we will derive the maximum permissible error at the filter output as a linear combination of the two types of errors. We will use the ADC quantization error, which is beyond our control for a given ADC chip to derive acceptable quantization error bounds for our calculated output. For a particular error 'e_i', the error induced at the output is bounded by [7]:

$$\|\Delta y_i\|_\infty \leq \left\|g_{y,e_i}\right\|_1 \cdot \|e_i\|_\infty \quad (10)$$

where g_{y,e_i} is the impulse response of the system from the error source to the output and $\|e_i\|_\infty$ is the maximum error possible. The total error at the output is given by:

$$\|\Delta y\|_\infty = \|\Delta y_0 + \Delta y_{trunc}\|_\infty \leq \sum_{i=0}^{1}\left\|g_{y,e_i}\right\|_1 \cdot \|e_i\|_\infty \quad (11)$$

In order to calculate Δy_0, we need the transfer function from the error source 'e_0' to the system output, which is the system transfer function given by (9). The value $\left\|g_{y,e_0}\right\|_1 = 2.6305$ while for our 14-bit ADC, $\|e_0\|_\infty = 2^{-14}$V as we are interpreting the ADC input in Q1.13 fixed point format.

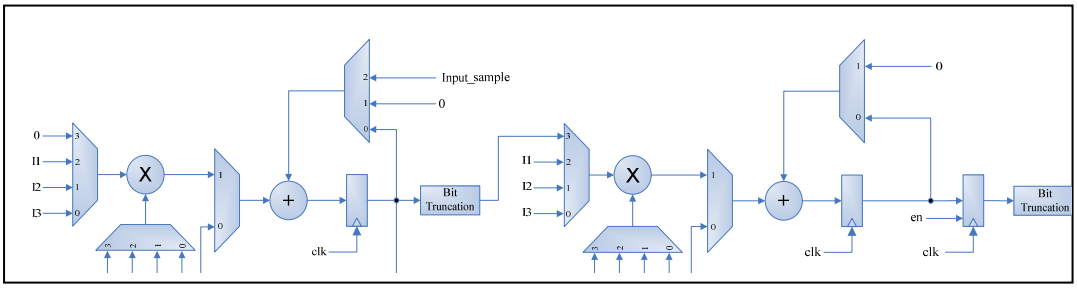

Fig. 4. Filter architecture.

Therefore, the maximum error bound due to ADC's quantization noise is calculated as:

$$\|\Delta y_0\|_\infty = \|g_{y,e_0}\|_1 . \|e_0\|_\infty = 2.6305 \cdot 2^{-14} = 0.16055\, mV$$

To calculate Δy_{trunc}, we need the transfer function from the error source 'e_1' shown in Fig. 2 to the system output given by:

$$G_{y,e_1}(z) = 0.8911z^3 - 2.6734z^2 + 2.6734z - 0.8911 \quad (12)$$

Therefore,

$$\|\Delta y_{trunc}\|_\infty \le \|g_{y,e_1}\|_1 . \|e_1\|_\infty = 7.129 \cdot 2^{-(f+1)}\ V \quad (13)$$

where g_{y,e_1} is the impulse response of the transfer function given in (9). Choosing Δy_{trunc} such that it does not contribute significantly to Δy, we select a value that is 10% of Δy_0 i.e. $7.129 \cdot 2^{-(f+1)} \le (10\%)\ 0.16055$ mV. This gives a value of $f \ge 18$. Thus a minimum of 18 bits are required to be preserved during the calculation of the sum. Together with the 3 bits of the integer part, the output of the filter requires a total of 21 bits.

III. Architecture

Fig. 3 shows the processor block diagram. The raw data is buffered in a ping-pong style ADC buffer with each location storing 2 ADC samples. This allows processing of two range bins at a time using two filters running in parallel. The two results that are computed at the filter output are stored one by one inside an output buffer. Two new range bin samples are subsequently read from the ADC buffer for similar processing and storage. All this is taking place while a new radar return is being written inside the second part of the ADC buffer.

Fig. 5 shows the flow of data inside the processor. When the ADC buffer has stored a new radar return, a signal is asserted that tells the controller to start processing the new return. The return, picked up two samples at a time is supplied to the two IIR filters. Since there are 'M' samples in a return and 'S' filters operating in parallel, where M>>S, filters have to be re-used for different range bins in a time shared manner. IIR filters require intermediate results of a previous calculation to produce a new result, we need to store the intermediate variables calculated for each range bin to be used the next time around when another result of the same range bin is computed. For Nth order filter 'N', we require 'N-1' intermediate variable buffers each with depth 'M'.

Fig. 6 shows the internal architecture of the filter used. The pole and zero sections are each built around a single multiplier-adder combination. The filter coefficients are stored in registers and supplied to the respective multipliers via multiplexers.

IV. Results

The design was implemented using a Virtex-4SX35 FPGA on Xilinx Xtreme DSP Kit-IV. The testing of the processor was carried out using baseband 'I' channel data from an operational TA-10K radar. During testing, data was fed from the host processor to the FPGA using the PCI interface once an interrupt was generated by the FPGA. The result was also read back from the FPGA. Results shown in Fig. 6 show a single radar echo return from the TA-10K radar. The upper part of the figure shows raw unprocessed baseband I-channel data. The lower part of the figure shows the processed output data in which clutter has been removed and the target preserved. This data is then fed to the next part of the processor for target thresholding. The coefficient quantization, maximum variable bounds and maximum output error bounds used in the IIR filters ensure that the filter response is acceptable, there are no instability issues, no variable overflows and the output error is within reasonable limits.

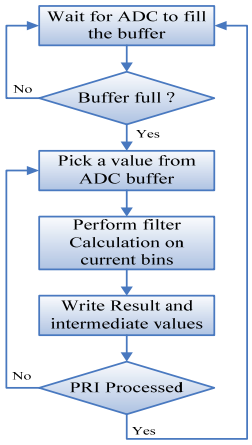

Fig. 5. Data flow.

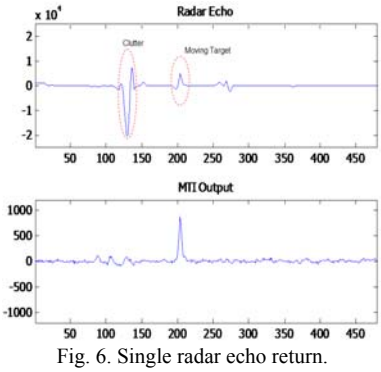

Fig. 6. Single radar echo return.

V. Conclusion

Architecture for IIR filter based implementation of the Moving Target Indicator algorithm is presented. The approach is computationally less intensive than the one employing FIR filters and gives matching performance. However, care has to be taken to avoid variable overflows, which are fed back for the next calculation. This will cause the system to produce wrong results from that point onwards, unlike FIR filters where feedback is not required. The techniques presented in this paper allow the developer a method for achieving a balance between resource utilization and maximum allowable output error.

References

[1] Z. Ali, A. Arshad, and U. Razzaq, "An FPGA Based Semi Parallel Architecture for Higher Order MTI Processing," *IEEE symposium on Rapid System Prototyping (RSP)*, June 2010

[2] S. Xu, M. Li, and J. Suo. "Clutter Processing for Digital Radars Based on FPGA and DSP," *In IET Intl. RADAR conference*, 2009

[3] R. Stapleton, K. Merranko, C. Parris, and J. Alter. "The use of Field Programmable Field Arrays in High Performance Radar Signal Processing Applications," pp. 850-855, *IEEE International Radar Conference*, 2000.

[4] B. D Green and L. E Turner, "New Limit Cycle Bounds for Digital Filters," *IEEE Transactions on Circuits and Systems*, vol. 35, no. 4, 1988.

[5] K. K. Johnson and I. W. Sandberg, "A separation theorem for finite precision digital filters," *IEEE Transactions on Circuits and Systems-I*: Fundamental Theory and Applications, vol. 42, no. 9, Sep 1995

[6] J. G. Proakis and D. G. Manolakis, "Digital Signal Processing: principles," *algorithms and applications*, Prentice-Hall, New Jersey, 1996.

[7] J Carletta, R. Veillette, F. Krach, and Z. Fang, "Determining appropriate precisions for signals in fixed point IIR filters," pp. 656-661, DAC2003.