SOFT IP CORE IMPLEMENTATION OF RECURSIVE LEAST SQUARES FILTER USING ONLY MULTPLICATIVE AND ADDITIVE OPERATORS

Gaye Lightbody

School of Computing and Mathematics,
Faculty of Engineering, University of Ulster,
Jordanstown, BT370QB, N. Ireland
email: g.lightbody@ulster.ac.uk

Roger Woods, Jonathan Francey

School of Electronics, Electrical Engineering
and Computer Science, ECIT, Queen's Island,
Queen's Road, Belfast, BT3 9DT, N. Ireland.
email: r.woods, jfrancey03@qub.ac.uk.

ABSTRACT

Soft IP cores can be realized as parameterisable HDL descriptions of circuit architecture where the performance comes from efficiently mapping system functionality. However, special arithmetic operations e.g. division, reciprocal, can restrict this mapping. An approach is presented that maps the system onto foundation operations, multiplication and addition, thereby giving a freer mapping of the full system. The methodology and results are given for a QR-based recursive least squares filter design on a Xilinx Virtex 4 FPGA giving a 5 GFLOPS performance.

1. INTRODUCTION

One design approach for FPGA-based DSP system design uses pre-designed functional blocks known as intellectual property (IP) cores [1]. FPGA vendors provide their own IP core libraries as well as a brokering service e.g. the AllianceCORE (Xilinx) and Megafunction partner program (Altera). A key issue with IP cores is the development of the underlying architecture such that factors e.g. throughput and latency, can be used to drive the resulting parameter selection and therefore core functionality. For example, throughput can be used to scale the parallelism as well as pipelining levels of the circuit architecture.

Implementations of matrix based algorithms based on systolic arrays [2] are powerful representations as they allow processing power to be gained through the concurrent use of an array of simple repetitive units. This offers high levels of parallelism, pipelining and local interconnections and lends itself to scalability. This process of generating a QR systolic array version of the recursive least squares (RLS) filter [2] from a mathematical description is summarized in Fig. 1. The RLS architecture uses boundary cells (BC) and internal cells (ICs) defined in Fig. 2. The algorithm is firstly mapped onto a dependency graph (DG) of the triangular matrix representation then onto signal flow graph (SFG) and circuit architecture realizations [3]. By capturing the mappings of the final stage, it is then possible to parameterize functionality where performance translates to array size and pipelining levels in the cells.

A major design challenge for IP core providers has been the mapping of DGs onto efficient and suitable cell architectures. For Fig. 1, a simple mapping of the triangular DG onto a linear array can result in an architecture where the cell efficiency is not maintained (Fig. 1b). This has been overcome in two ways. By using an intricate mapping that overlaps computations in time (Fig. 1c), it is possible to map QR functionality efficiently onto a linear array [3] and even a wider range of structures [4], while maintaining individual QR cell functionality. Radar [5] applied a simpler mapping (Fig. 1d) that achieved 100% efficiency but needed a generic cell based on CORDIC that performed the BC and IC cell functionality. The work in [3] involved a mapping of the triangular array of cells down onto a reduced architecture. An alternative approach is considered here where multipliers and adders are used as the key *foundation* blocks allowing all the cell functionality to be broken into series of these operations.

The paper is organized as follows. Section 2 presents the Squared Givens Rotation (SGR) algorithm for QR-based RLS filtering. The implementation of the reciprocal function is given in section 3. Section 4 presents the revised design and performance of a generic cell using a Xilinx Virtex 4 family. Conclusions are given in the final section.

2. QR-RECURSIVE LEAST SQUARES

Computationally compact solutions for RLS filtering have evolved that are based on QR decomposition solved by Givens rotations performed on a triangular array of processors similar to that given in Fig. 1 [2]. The rotation parameters to eliminate one sub diagonal element of the data matrix are calculated within the BC and then passed to the ICs in the same row. The process in its raw form contains a number of highly complex arithmetic operations such as square root and division but approaches such as the Squared Givens Rotations (SGR) algorithm [3] remove the need for the square root operation and halve the number of multiplications, whilst still maintaining the SGR algorithm advantages of low word length characteristics and performance [3]. However, the reciprocal function is still needed. This can be directly implemented but the intention here is to investigate how this operation can be realized

1-4244-1060-6/07/$25.00 ©2007 IEEE.

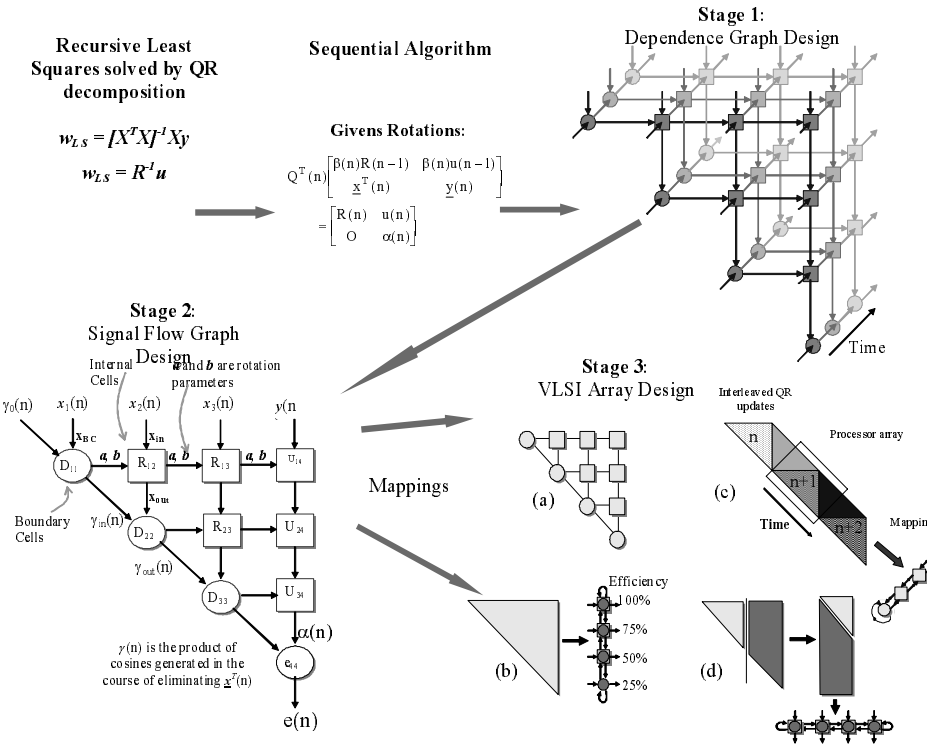


Fig. 1 Systolic array architecture design flow illustration for Recursive Least Squares filtering

using convergent division methods [6]. By breaking this into a series of multiplications and additions, we eliminate the key difference between the BC and IC cell and allow the possibility of a generic QR cell that can give the full functionality of RLS solved by QR decomposition.

3. SQUARED GIVENS ROTATIONS

3.1. QR cells with complex arithmetic

The starting point for the generic QR cell is analysis of the BC and IC cells (Fig. 2). For applications such as adaptive beamforming [7], complex arithmetic is required to represent the spatial element to the filtering application. In addition, the performance requirements are such that floating-point arithmetic is desirable [3]. The SFGs for the complex SGR cells can be broken down into their real arithmetic components (as depicted in Fig. 4 for the IC).

3.2. Reciprocal function

The key difference between the cells in Fig. 2 is the real reciprocal function in the BC which can be implemented by digit recurrence algorithms where digits are computed in a most significant bit first fashion. Typically, $2n+1$ additions are needed (n is the word length) but this can be reduced to n by employing the non-restoring recurrence algorithm of the SRT division algorithm [8],[9], but these suffer from high latency and converge linearly to the quotient.

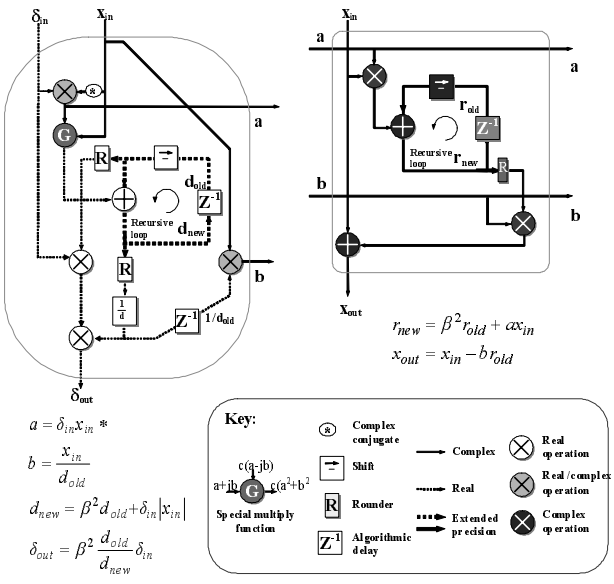


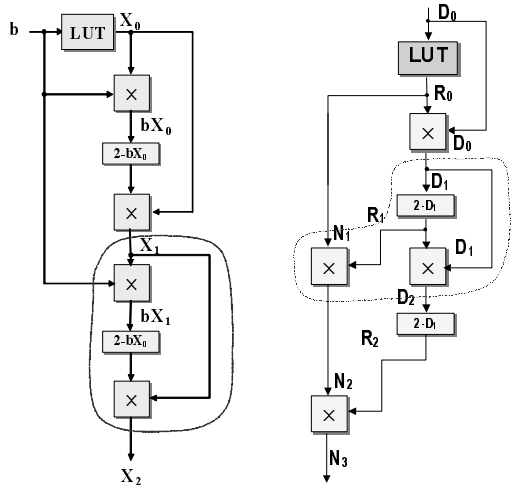
Fig. 2 Cell definitions for SGR QR algorithm

On the other hand, functional iterative algorithms [6] employ multiplication as the fundamental operation and produce at least double the number of correct bits at each iteration. As the initial stages only retire a few bits, there has been a significant interest in approximation methods to determine the first few correct bits [10]. For example, if the first 7 bits of the quotient were found from a look up

table (LUT) then the number of iterations required to produce a result of 53 bit accuracy is reduced to 3. However, as the complexity of LUTs grows exponentially, i.e. 2^n, the LUT is only used to create the initial result and other iterations are computed using suitable hardware.

Two techniques for functional iteration of division were considered, Newton Raphson and series convergence expansions. The series convergent reciprocal approach gives an algorithm with two separate series multiplications, one at the start and one at the end (Fig. 3(b)) and with the stages in between, performed by 2 parallel multiplications. The Newton Raphson iteration converges to the reciprocal and then for full division, it multiplies this by the dividend to calculate the quotient (Fig. 3(a)).

In both methods, the iteration has 2 multiplications and a two's complement operation. However, the Newton Raphson multiplications are dependent whereas the series expansion method are not and can be carried out in parallel. Secondly, the Newton Raphson iteration is self checking in that any error in computing x_i can be corrected by the subsequent iteration, since all operations are dependent. For the series expansion, the result is computed as the product of independent terms and an error in one of them will not be corrected. To account for this, a few extra bits of precision [6], [10] are used. The circuits for the reciprocal and the complex multiplication and addition are applied to the QR filter to giving the revised IC (Fig. 4) and BC cells (Fig. 5).



(a) Newton Raphson (b) Series expansion

Fig. 3 Two stage reciprocal circuits

For the QR, a range of mantissa word lengths typically 12 to 24 bits suffices, bringing the bit width up to the IEEE single precision floating point standard. With 3 to 6 bit LUT, a 12 to 24 bit result (respectively) can be obtained after only 2 iterations with the same being achieved in a single iteration for a 6 to 12 bit LUT. The arithmetic cores available from Northeastern University [11] are used.

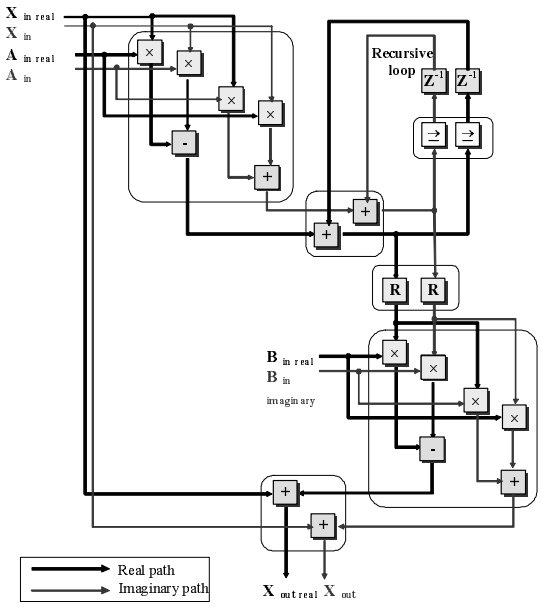


Fig. 4 IC for the SGR QR algorithm

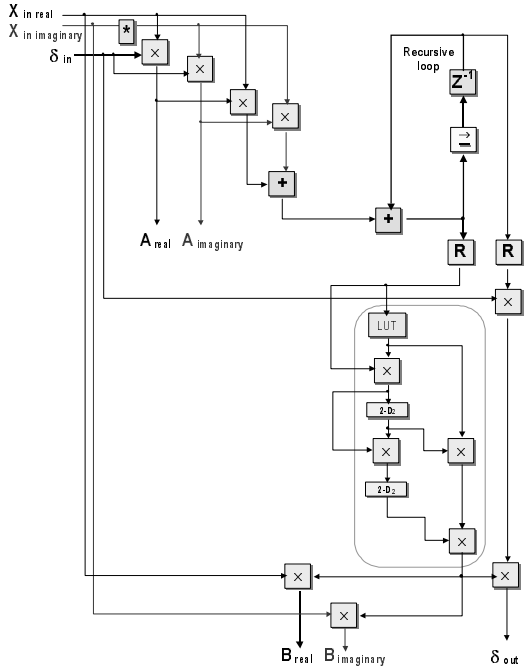


Fig. 5 BC for the SGR QR algorithm

4. FPGA IMPLEMENTATION

Table 1 gives Xilinx Virtex XC4VLX15 FPGA details for reciprocal options: a full 23-bit LUT giving 16-bit accuracy; a 10-bit and 6-bit LUT with the 1 stage. In each case, Synplify Pro v8.6.2 was used and optimized for speed with all available settings applied, such as retiming.

Floating point implementation on FPGA has considerably clouded the classical method of using smaller LUT sizes and additional computation stages. This is

highlighted in Table 2 which shows the respective performance, throughput and area, of an individual floating point adder, a floating point multiplier and a reciprocal based on only 1 multiplication. Note that the multiplier was optimized for speed increasing the area beyond that of a reciprocal block which contains a multiplication.

	LUT only	Full reciprocal unit			Speed (MHz)
	LUTs	DSP48	FD	LUT	
6-bit	22	4	266	745	116.5
10-bit	619	4	266	1342	116.5
23-bit	1457				

Table 1. FPGA implementation using Xilinx Virtex4

The approach adopted here was to use a *generic cell* (Fig. 6) that performs both computations and clearly shows the overlap. A schedule based on 2 multipliers and 2 adders, is used to compute the early part of the computation. A dedicated multiplier and LUT were used for the reciprocal but options exist for using an existing multiplier and one LUT shared amongst a number of cells to improve the utilization and increase scheduling complexity. A fixed-point multiplier could have been used for the reciprocal.

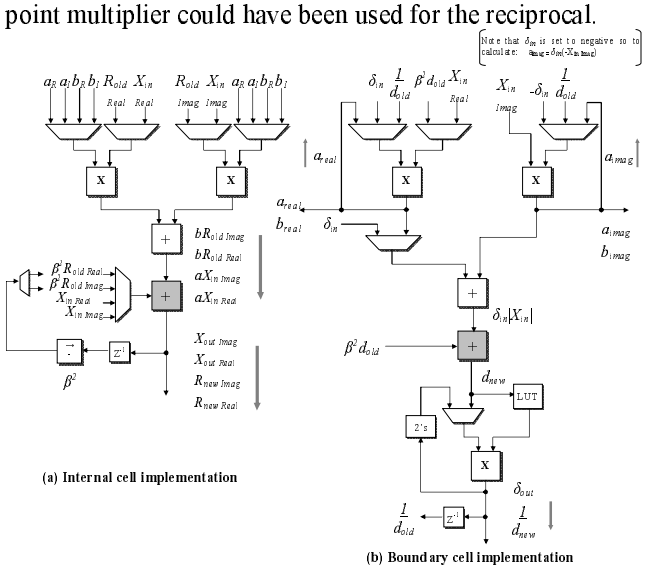


Fig. 6 QR cell implementations on reduced architectures

The *generic cell* was synthesized using the floating point libraries only giving an area of 3229 LUTs, 12 DSP48s and 1411 D flip-flops. Thus, the processor is capable of performing 547MFLOPs (5 FLOPS per cell at 109.4 MHz) indicating a performance of 5GFLOPs for the FPGA without using the optimizations suggested. It does highlight the poor return of floating point on FPGA but Walke [3] has indicated that the same performance may be possible with a 24-bit fixed-point wordlength.

5. CONCLUSIONS

The paper presents an approach for mapping specific arithmetic functions into a series of multiplications and additions thereby easing the final mapping of the design into FPGA hardware. Whilst the approach has only been demonstrated for the reciprocal function, the same approach can be applied to division and square root.

	DSP48	LUT	FD	Speed (MHz)
Multiplier	4	799	347	141.4
Adder	-	620	343	208.2
Reciprocal	4	745	266	116.5

Table 2. Processor implementations

6. REFERENCES

[1] P. S. Zuchowski et el., "A hybrid ASIC and FPGA architecture" Proc. IEEE/ACM Int'l Conf. on Computer-Aided Design, pp. 187--194, 2002.

[2] J. G. McWhirter, "Recursive least squares minimisation using systolic array", Proc. SPIE (Real-Time Signal Processing IV), vol. 431, pp. 105-112, 1983.

[3] R. L. Walke, High Sample Rate Givens Rotations for Recursive Least Squares, PhD Thesis, University of Warwick, 1997.

[4] G. Lightbody, R. Woods and R. Walke, "Design of a parameterisable silicon intellectual property core for QR-based RLS filtering", IEEE Trans. on VLSI Systems, vol. 11, No. 4, pp.659-678, August 2003.

[5] C. M. Rader "VLSI systolic arrays for adaptive nulling", IEEE Signal Proc. Mag., vol. 13, no. 4, pp. 29-49, Jul 1996.

[6] M. J. Flynn, "On Division by Functional Iteration", IEEE Trans. on Comp., vol. C-19, No. 8, pp. 702-706, 1970.

[7] T. J. Shephard and J. G. McWhirter, "Systolic Adaptive Beamforming", chapter 5, *Array Processing*, Eds. S. Haykin, J. Litva and T. J. Shephard, Springer-Verlag, ISBN 3-540-55224, pp. 153-243, 1993.

[8] K. D. Tocher, "Techniques of Multiplication and Division for Automatic Binary Computers", *Quart. J. Mech. Appl. Math.*, vol. XI, Pt. 3, pp364-384, 1958.

[9] J. E. Robertson, "A new Class of Division Methods", *IRE Trans. on Electronic Comp.*, vol. EC-7, pp. 218-222, 1958.

[10] M. J. Schulte, J. E. Stine and K. E. Wires, "High-Speed reciprocal Approximations," *Proceedings of the 14th Symposium on Computer Arithmetic*, pp. 1183-1187, 1999.

[11] "Variable Precision Floating Point Modules" available from NorthEastern University, http://www.ece.neu.edu/groups/rpl/projects/ floatingpoint/index.html.