20 GFLOPS QR processor on a Xilinx Virtex-E FPGA

Richard L. Walke[*a], Robert W. M. Smith[a], Gaye Lightbody[b]

[a]Defence Evaluation and Research Agency, Malvern, WR14 3PS, UK.

[b]Queen's University of Belfast, Ashby Building, Stranmillis Road, Belfast BT9 5AH, UK.

ABSTRACT

Adaptive beamforming can play an important role in sensor array systems in countering directional interference. In high-sample rate systems, such as radar and comms, the calculation of adaptive weights is a very computational task that requires highly parallel solutions. For systems where low power consumption and volume are important the only viable implementation is as an Application Specific Integrated Circuit (ASIC). However, the rapid advancement of Field Programmable Gate Array (FPGA) technology is enabling highly credible re-programmable solutions. In this paper we present the implementation of a scalable linear array processor for weight calculation using QR decomposition. We employ floating-point arithmetic with mantissa size optimised to the target application to minimise component size, and implement them as relationally placed macros (RPMs) on Xilinx Virtex FPGAs to achieve predictable dense layout and high-speed operation. We present results that show that 20GFLOPS of sustained computation on a single XCV3200E-8 Virtex-E FPGA is possible. We also describe the parameterised implementation of the floating-point operators and QR-processor, and the design methodology that enables us to rapidly generate complex FPGA implementations using the industry standard hardware description language VHDL.

Keywords: FPGA, DSP, QR, array processor, Radar, Smart Antenna, Adaptive Beamforming

1. INTRODUCTION

The beam pattern of an array of antennas can be shaped by adaptive beamforming to provide high gain in the direction of a wanted signal whilst rejecting toublesome interference from other directions[1]. The resulting performance benefits are often sufficient to justify multiple antennas and their receivers, and the technique is now commonly employed in radar and sonar to provide high degrees of resilence to interference and has application in communications systems for increasing capacity or cell size. However, in high sample-rate applications, such as radar where the base-band data-rate is of the order of MHz, the task of calculating the adaptive weights can be very computationally demanding. In many cases a parallel processor is necessary to achieve the necessary weight update rates.

Multiple programmable Digital Signal Processing (pDSP) processors, and increasingly General Purpose Processors (GPP), are commonly employed to realise parallel processors, and commercial-off-the-shelf (COTS) boards and systems help to avoid specialised solutions. However, for weight calculation the limited interprocessor bandwidth can severly restrict the scalability of a conventional programmable processor solution. Furthermore, the high computation combined with low power and low volume requirements can make an ASIC the only viable solution. Unfortunately, in low volumes the high set-up costs of ASICs make them unattractive. However, FPGA technology, with its rapid advancement over the last few years, now offers a viable alternative.

Current FPGA devices offer the equivalent of over a million programmable gates and can achieve system rates in excess of 150MHz with careful circuit design. Furthermore, many FPGAs use SRAM to store their circuit configuration and are reprogrammable, offering both the flexibility of conventional processors, but with a DSP capability closer to an ASIC.

The power consumption of an FPGA circuit is over an order of magnitude greater than ASIC in equivalent technology. However, there are two aspects of FPGAs that may help to redress this balance. Firstly, FPGAs are built with the best integrated circuit technology. This is currently 0.18µm and will soon be 0.15µm. The best ASIC technology which is *affordable* in volumes representative of our (military) equipment production runs is 0.35µm. Power consumption reduces with feature size, and so the power consumption difference between a modern FPGA an a 0.35µm *affordable* ASIC is much reduced. Secondly, the programmablility of FPGAs allows circuits to be optimised to specific system needs, whereas an ASIC implementation is often more generic to justify its high development costs so may be less efficient than a specialised one. Specialisation of both the circuit and algorithm to a particular application, or even mode of operation, can lead to large improvements in circuit size

* Correspondence: walke@signal.dera.gov.uk *

and speed. For example, the size of an FIR filter can be reduced using constant coefficients multipliers which are half the size of programmable ones[2], and by optimising coefficients or eliminating them completely. This can reduce power consumption, which combined with better fabrication technology can make the FPGA's power consumption more comparable to that offered by an *affordable* ASIC technology.

Furthermore, circuit specialisation also has the advantage of simplifying circuit design and making automatic design of complex parallel-DSP implementations more realistic. Circuit design time is not falling at the same rate as which gate count is increasing, and this is an increasing problem which the reprogrammability of FPGAs may help to address.

It is clear that FPGA is a very relevant technology for implementing high performance DSP, particularly in military systems. In this paper we present a scalable, parallel implementation of a weight calculation processor for implementation in FPGA. By way of introduction we start in section 2 with a brief overview of adaptive beamforming in radar systems and present QR-decomposition as a method for solving the weight calculation problem. Wordlength is critical to the size of our FPGA implementations, so we have included numerical simulation results that show we can use low wordlength when employing floating point arithmetic and QR-decomposition.

In section 3 we provide an overview of how we implement the QR operation using a highly scalable linear array of special purpose processors which is suitable for implementation over one or more FPGAs. In section 4 we present our design methodology for achieving high performance DSP implementations in FPGA using VHDL. As floating-point arithmetic arithmetic is rarely used in FPGA we present in section 5 an overview of our operator implementations. In section 6 we present results of the implementation of the processor on the largest of the current generation of FPGA and compare the predicted performance with ASIC and programmable DSP approaches. Our conclusions are presented in section 7.

2. ADAPTIVE BEAMFORMING IN RADAR SYSTEMS

2.1 System overview

Figure 1 provides an overview of an adaptive beamformer in a radar application in which the objective is to combine the outputs of an array of antennas in a way which offers low gain in the direction of interference whilst maintaining a high gain in the direction of interest[1]. The antenna outputs (p in number) are down-converted to base-band frequencies, digitised and filtered. Some of this data is used to calculate the weights, $\boldsymbol{w_i}$, for each beam and these are applied to the whole block of this data via an array of complex multipliers to give each beam output.

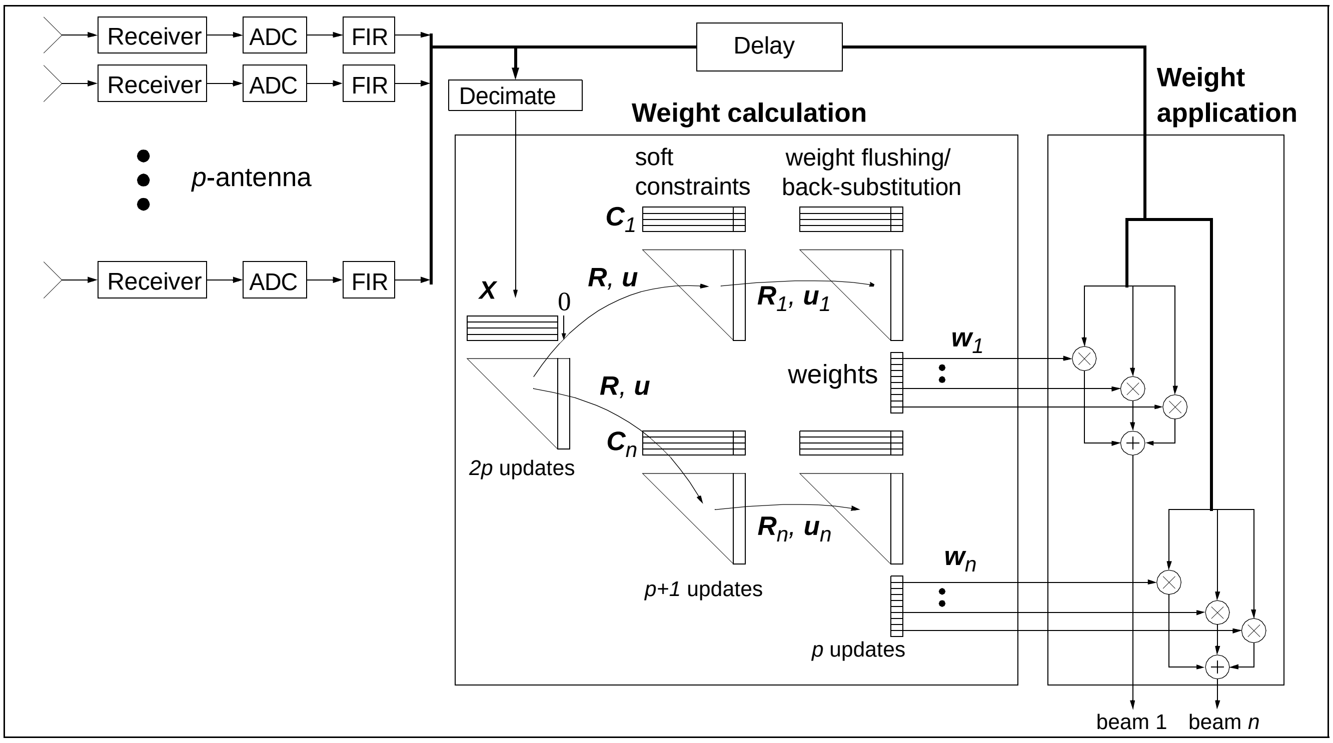

Figure 1: Adaptive beamforming system architecture

At the heart of the weight calculation process is the need to solve a least mean squares problem. By employing QR-decomposition[3] we can obtain good numerical performance[4] and so reduce the required wordlength for an efficient ASIC or FPGA implementation. In the proposed system we employ QR-decomposition in a three-stage process. In the first step we decompose at least *2p* samples from each block of input data, referred to collectively as $\boldsymbol{X}$, into an upper-triangular matrix $\boldsymbol{R}$ and vector $\boldsymbol{u}$. This captures the interference environment. To this are applied constraints for each beam, which stabilise the beam pattern and set a particular look-direction. The resulting $\boldsymbol{R_i}$ and $\boldsymbol{u_i}$ are related to the weights by $\boldsymbol{R_i w_i}=\boldsymbol{u_i}$, and the weights are obtained in the final step by either back-substitution or weight-flushing[5] (the latter shown in Figure 1). Back-substitution requires a separate processor but is computationally efficient, whereas weight flushing reuses the same hardware, but requires many more operations.

If QR-decomposition is implemented using Givens Rotations[6] then a highly parallel triangular array processor architecture can be used, known as the *QR-array*. This allows a very-high throughput implementation to be achieved by employing a large number of processors operating in parallel[7] and is described further in section 3.1.

The QR-array requires two types of operation, often referred to as *boundary* and *internal cell* operations, which are simple 2D rotations. Algorithm variants have been developed which avoid the square-root and division normally associated with a rotation operation, offer reduced operation count and allow fixed-point arithmetic to be employed. The Squared Givens Rotation (SGR) variant[8] that we employ here offers low operation count and is square-root free. Floating-point arithmetic is required but as we show next with relatively short mantissa wordlength.

2.2 Wordlength requirements

The size of an FPGA or ASIC implementation employing fully parallel arithmetic operators is dependent upon the square of the wordlength (at least to a first approximation). Therefore, it is important to determine the minimum wordlengths from system simulations. In the adaptive beamformer application, the wordlength has to be sufficient to calculate the weights to an accuracy that maximises the cancellation of the interference. To put this in perspective, the input contains 3 components: signal, interference and thermal noise (the latter from receiver components). The interference and signal are usually measured relative to thermal noise, where the signal is generally smaller and the interference much larger than thermal noise. The scaling of the ADC input is usually arranged so that thermal noise toggles the least significant bit or so, and has sufficient range to digitise interference without significant probability of saturation.

The task of the adaptive beamformer is to suppress interference to thermal noise levels. Therefore, the weights must be calculated with sufficient accuracy to do this. For 60dB interference (i.e. a voltage 1000 times greater than thermal noise), approximately 10-bits of accuracy are required.

The accuracy of the weights depends upon the nature of the arithmetic error and how it accumulates. Figure 2 shows the signal to interference+noise ratio (SNIR) for a range of wordlengths, for the SGR algorithm. The number of antennas is 32 (i.e. p=32). At low wordlengths the SNIR is dominated by arithmetic errors, but as wordlength is increased the SNIR improves until it becomes dominated by the thermal noise (as all interference has been removed). Here, the maximum SNIR after post-processing, is 17dB and there is no benefit in increasing the wordlength further.

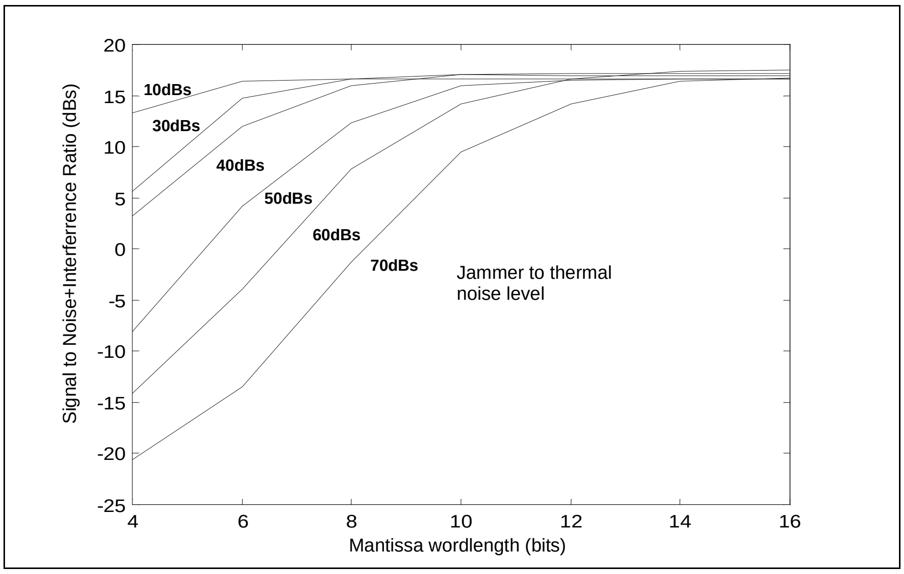

Figure 2: Numerical simulation results

It is clear from these results that relatively high system performance of 70dB interferrence suppression can be achieved with only 14-bit mantissa wordlength. Note that if fixed-point algorithm and arithmetic were used then the wordlength requirements would be significantly higher for two reasons. Firstly, fixed-point errors are greater than the floating-point errors due to the relative nature of the latter. Secondly, additional bits would be required to realised the necessary dynamic range (which in floating point is achieved via the exponent). For example, our fixed-point CORDIC (COordinate Rotation by DIgital Computer)[9] implementation required almost twice the wordlength to achieve the same numerical performance[10].

3. QR-ARRAY MAPPING

3.1 QR-array

The QR-array, as shown in Figure 3, is a highly parallel processing architecture for QR-decomposition. The operation performed by the two types of processor are shown in the insets. The antenna data enters from the top and progresses down the array. On each row the leading term is eliminated by a 2-D rotation between cell input and elements of ***R*** and ***u*** stored within each cell.

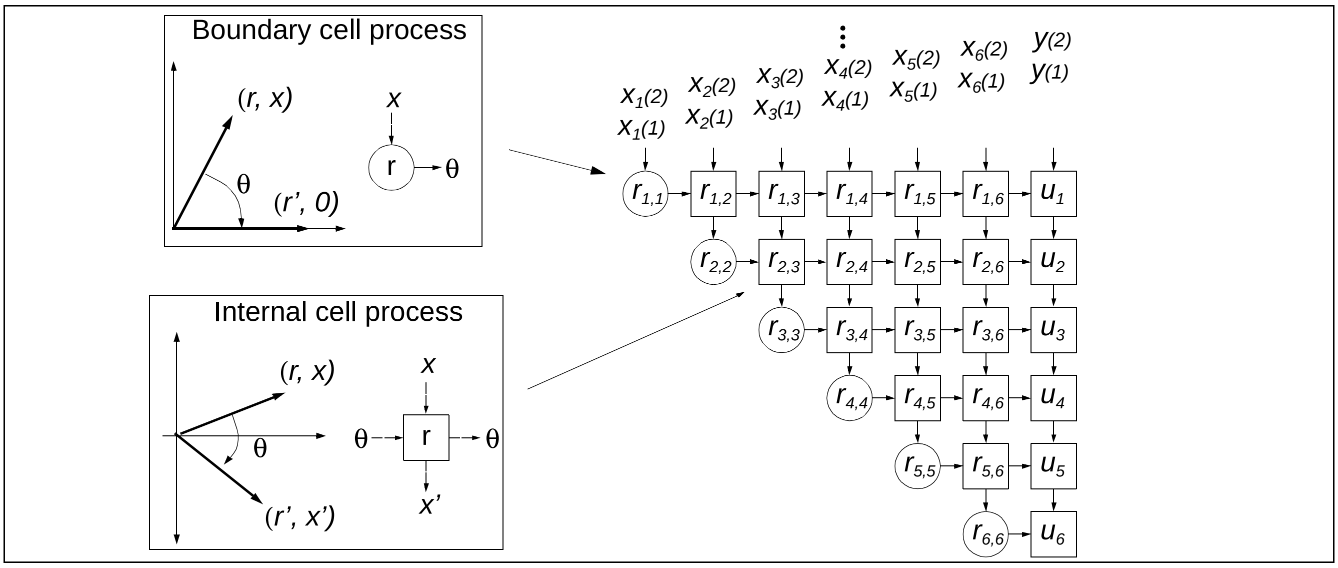

Figure 3: Systolic QR-Array Processor

In this case the array has 7 inputs and utilises $p(p+1)/2-1=27$ processors. In reality, the number of processors used by the QR-array solution is too high and if implemented in a parallel fashion the throughput would be far in excess of most system requirements.

To avoid this the array is either mapped to a reduced number of time-shared processors, or alternatively, the processors themselves are reduced in size by constructing them from time multiplexed arithmetic units (e.g. digit serial arithmetic). In our work we have adopted the former approach to provide a greater range of throughput options and avoid the inefficiency that may arise from time-multiplexing at the arithmetic operator level within the processor.

Techniques for mapping large arrays of operations onto a reduced number of processors are well established[11,12]. We present a mapping that we have derived for the QR-array which results in a highly scalable locally interconnected linear array.

3.2 Mapping to a linear QR array

In Figure 4 we show how the operations of the triangular QR-array of Figure 3 can be mapped onto a smaller number of processors. The resulting array is linear, locally interconnected and employs processors dedicated to either boundary or internal cell operations. The latter feature allows optimisation of the processors to either boundary cell or internal cell operations which is highly beneficial in the parallel implementations presented in section 6.

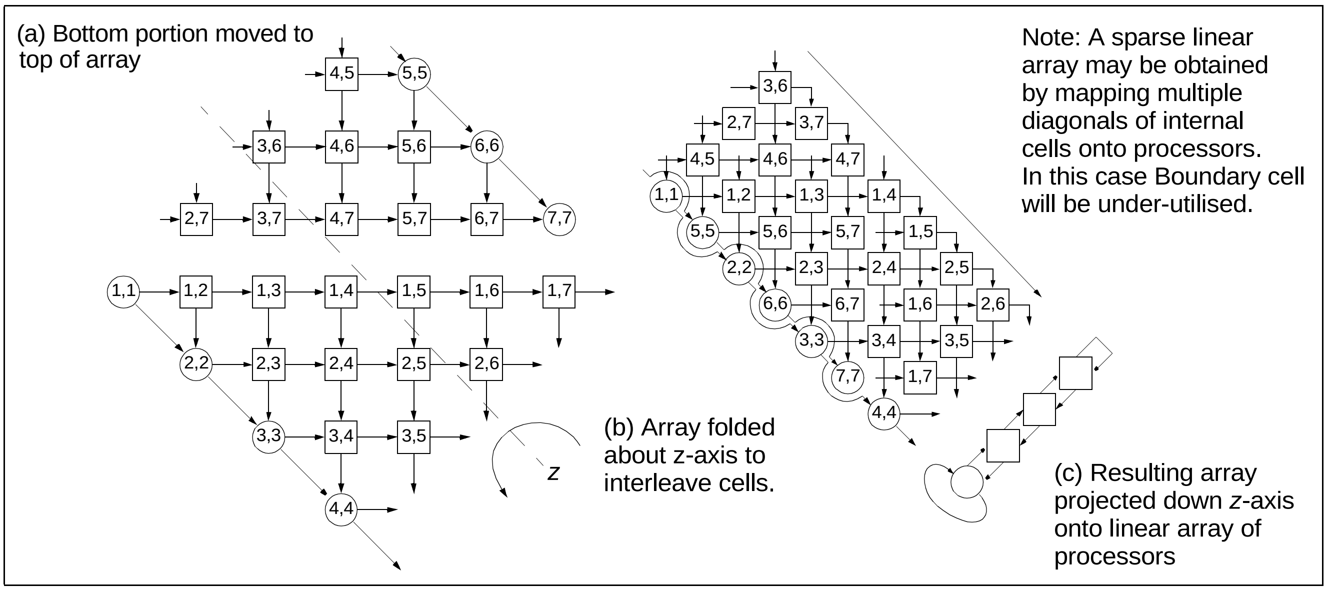
(a) Bottom portion moved to top of array
Note: A sparse linear array may be obtained by mapping multiple diagonals of internal cells onto processors. In this case Boundary cell will be under-utilised.
(b) Array folded about z-axis to interleave cells.
z
(c) Resulting array projected down z-axis onto linear array of processors

Figure 4: Mapping used to obtain locally-interconnected linear array from triangular array

The linear array is obtained by (a) first moving the lower portion of the triangular array to the top, and then (b) folding it to interleave operations. This places the same number of operations in each diagonal and ensures that local processor inter-connections are obtained when the operations are projecting down the diagonal onto a linear array of processors, as shown in Figure 4 (c). A more detailed description of the mapping and processor schedule is given by Lightbody *et al*[13].

Figure 5 shows a signal flow graph for the boundary and internal cell operations when employing the SGR algorithm.

Note that the loop to update the $\boldsymbol{R}$ and $\boldsymbol{u}$ quantities consists of a simple adder. This has two advantages. Firstly, the word-length of the adder can be increased to improve the accuracy to which $\boldsymbol{R}$ and $\boldsymbol{u}$ are accumulated over long runs of data. Secondly, with appropriate input scaling the adder can be made fixed-point and $\boldsymbol{R}$ and $\boldsymbol{u}$ terms updated on every clock cycle (assuming a processor for each cell in the array i.e. the full QR-array is used). Therefore, very high sample-rate operation in excess of 100MHz is possible.

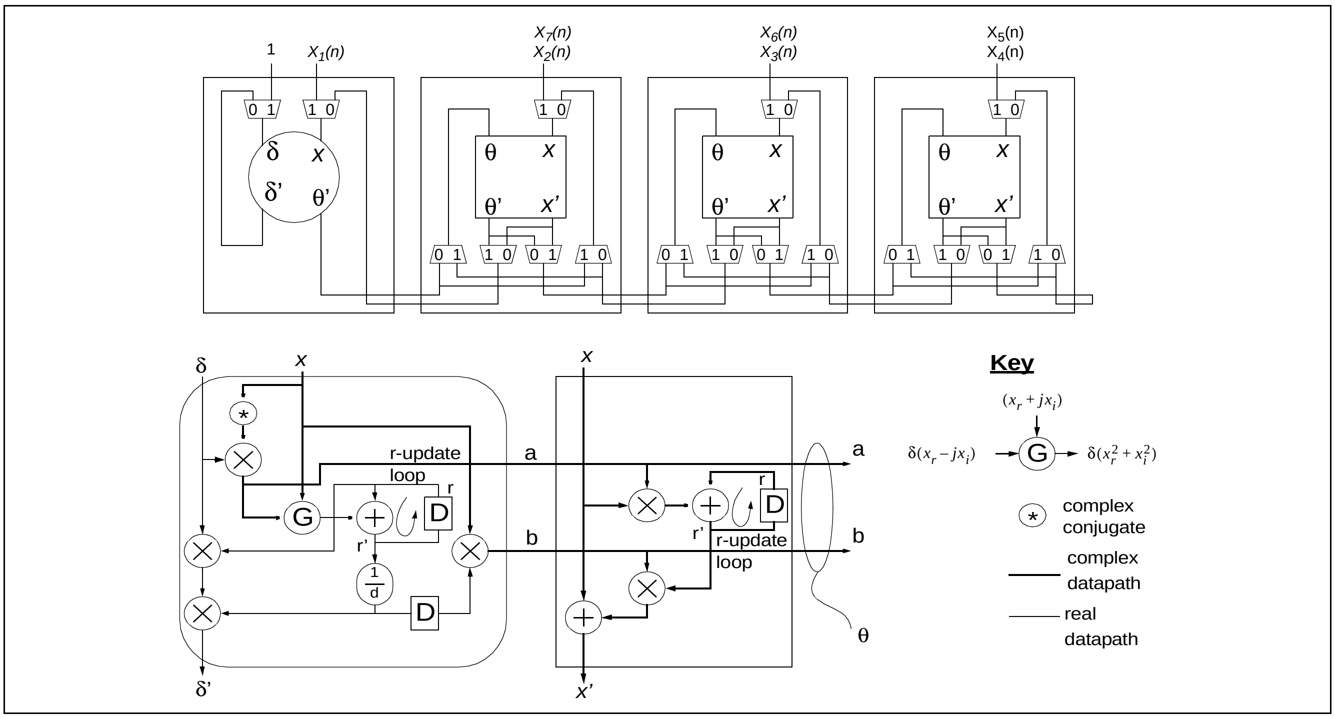

Figure 5: Squared Givens Rotation Algorithm

4. FPGA DESIGN METHODOLOGY

4.1 Hierarchical library of paramaterised circuits

We have built the FPGA implementation of the linear array processor using a hierarchal library of structural VHDL circuit descriptions as shown in Figure 6. The libraries are currently targetted at Virtex and Virtex-E. At the lowest level are the basic Virtex primitives, such as look-up-tables (LUTs) and registers. Upon this we build basic macros, such as adders and logic gates. These are parameterised for wordlength, amongst other things, and the description includes RLOC placement constraints[14] as VHDL attributes. The latter ensures that interconnected components are placed close together to achieve dense layout with high-speed, predictable timing. There is good support within Xilinx tools for specifying relative component placement and such macros are refered to as RPMs (Relationally Placed Macros). Generating placement for parameterised components can be a time-consuming process for complex components, however, it only needs to be done once and we have implemented a library of VHDL function calls providing such information as component size and latency to assist in the process

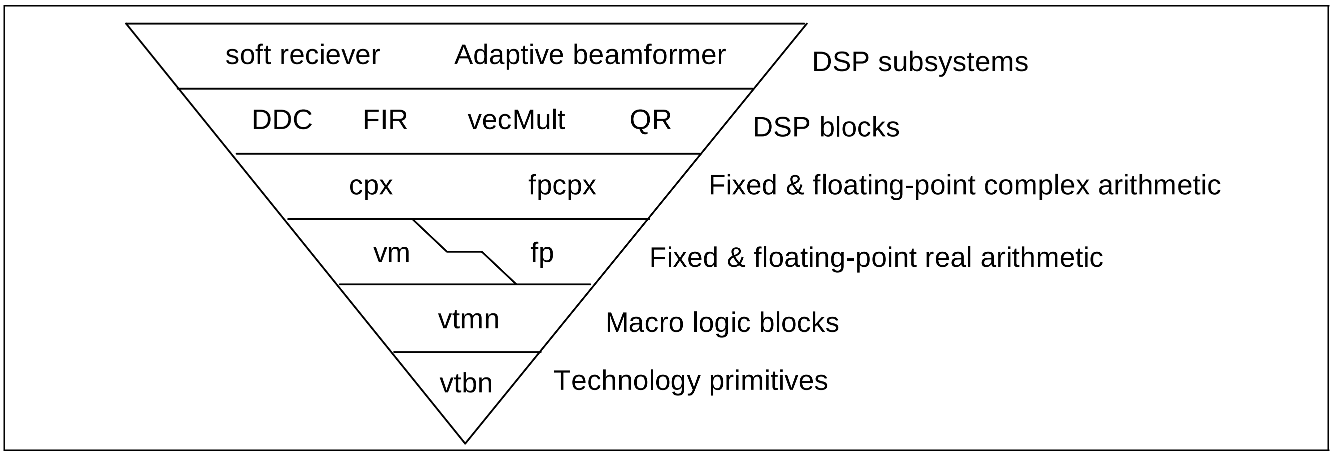

Figure 6: Heirarchical library structure

Using these basic macros we have created parameterised fixed-point and floating-point operators and more complex DSP operations, such as FIR filters, simple digital down conversion and vector multiply. These have been used to create radar receivers and beamformers, and we are currently in the process of implementing an adaptive beamformer.

Xilinx have a core generator which now provides many basic cores across their whole family for free. However, floating-point and complex arithmetic are not yet amongst their suite. Also, the core source code is not avialable, so creating new parameterised cores or modifying old ones for new applications is not possible. Furthermore, we have developed a fixed-point multiplier employing a novel algorithm to give reduced size and delay. As multiplication is a key operation in DSP, we are able to benefit from this in nearly all of our DSP components.

4.2 Design flow

Figure 7 summarises the design methodology that we currently use to implement DSP in FPGA. The first step is to design the system architecture, taking into consideration both the DSP datapath and control. It is important to get it right at this level if high-speed, scalable designs are to be possible. This high-level design is based on a knowledge of component size and performance, and the predictable speed and area of our cores is important in this process. A Matlab simulation of the system using bit-true models of our cores is then used to establish key parameters, such as wordlength. These parameters are employed in a VHDL description of the system employing established cores, and special purpose circuits (created from lower level macros). Simulation of the VHDL confirms its functionality before synthesis to a circuit netlist. We use Synplify because of its ability to specify component attribute values using complicated function calls.

After placement and routing of the circuit using Xilinx's M3.1i tools, we can establish whether the timing requirement have been met and modify the circuit if not. Resimulation of a VHDL version of the actual FPGA design, including timing, also ensures that the functionality has been maintained through the tool flow.

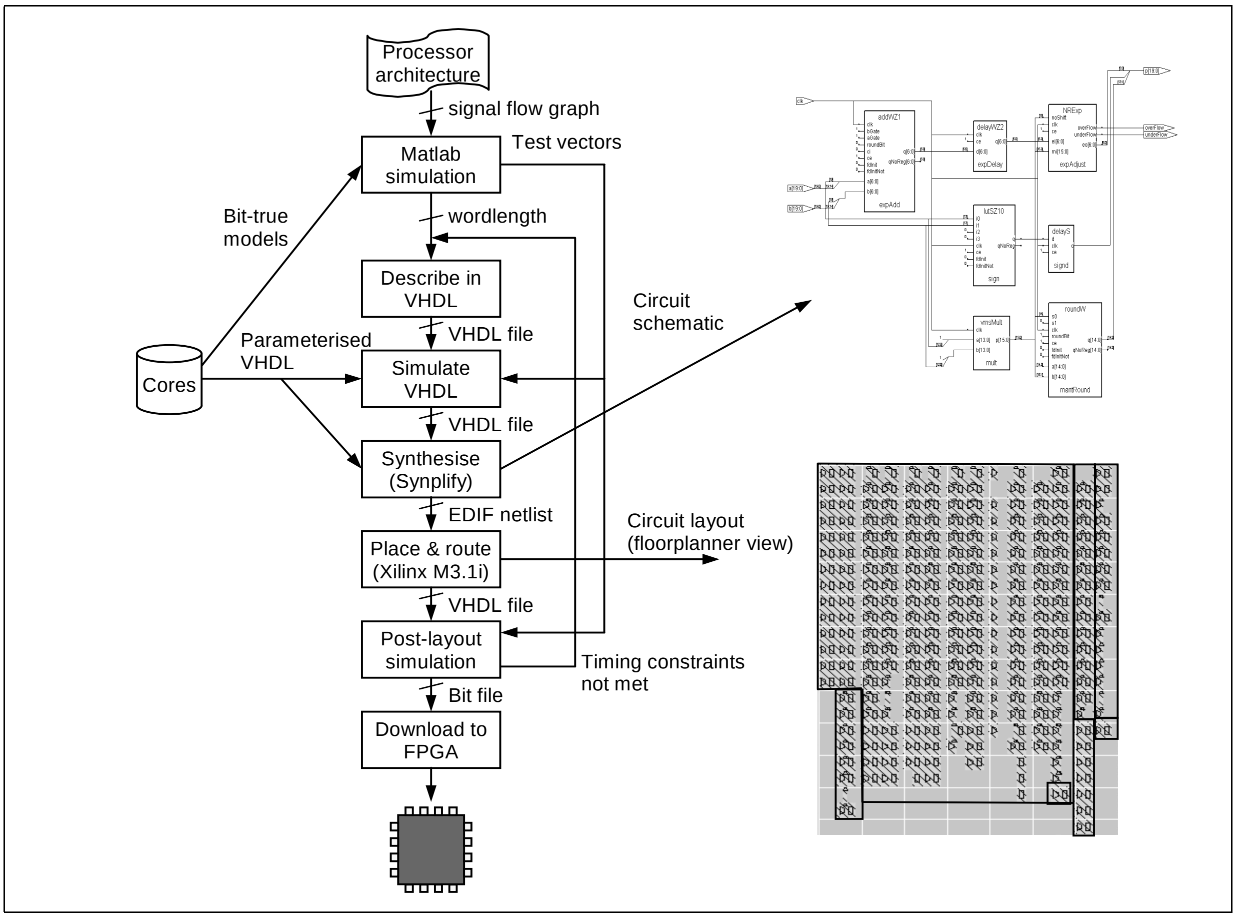

Figure 7: Design methodology

5. FPGA FLOATING-POINT OPERATORS

5.1 Floating-point representation

Our floating-point numbers are represented by 3 parts: an exponent, *e*, of *ew*-bits, a mantissa sign bit, *s*, and a mantissa magnitude word, *m*, of *mw* bits. The exponent is in 2's complement representation and its value chosen so that $0.5 \le m < 1.0$. Consequently, the most-significant bit of the mantissa is always 1 and so is not stored. Hence the overall number of bits required to represent a floating-point number is *mw+ew*. The value is given by $v = (-1)^s m 2^e$. Note that conversion to and from IEEE standard floating-point format[16] is trivial.

We have implemented the floating-point operators in a fully parallel and pipelined manner, so that they are capable of accepting new operands on each clock cycle.

5.2 Floating-point multiplier

A block diagram of our floating-point multiplier is shown in Figure 8(a). Floating-point multiplication is obtained by multiplying mantissas, adding exponents and EXORing the sign bit. The mantissa product is in the range $0.25 \le p < 1.0$ so a possible 1-bit left-shift is required to normalise it. This is combined with the rounding operation for efficiency. Rounding to nearest is implemented, and when equally near the result is rounded-up*. A product which has been normalised and then rounded may overflow (i.e. p=0.1...111 will become 1.0...000) and mantissas which cause this are detected by the exponent adjust unit, and renormalised. We make use of most of the features of the Virtex configurable logic blocks to implement the mantissa and exponent circuitry in a fast and efficient manner. As a result the floating-point multiplier is approximately only 20% larger than a fixed-point one.

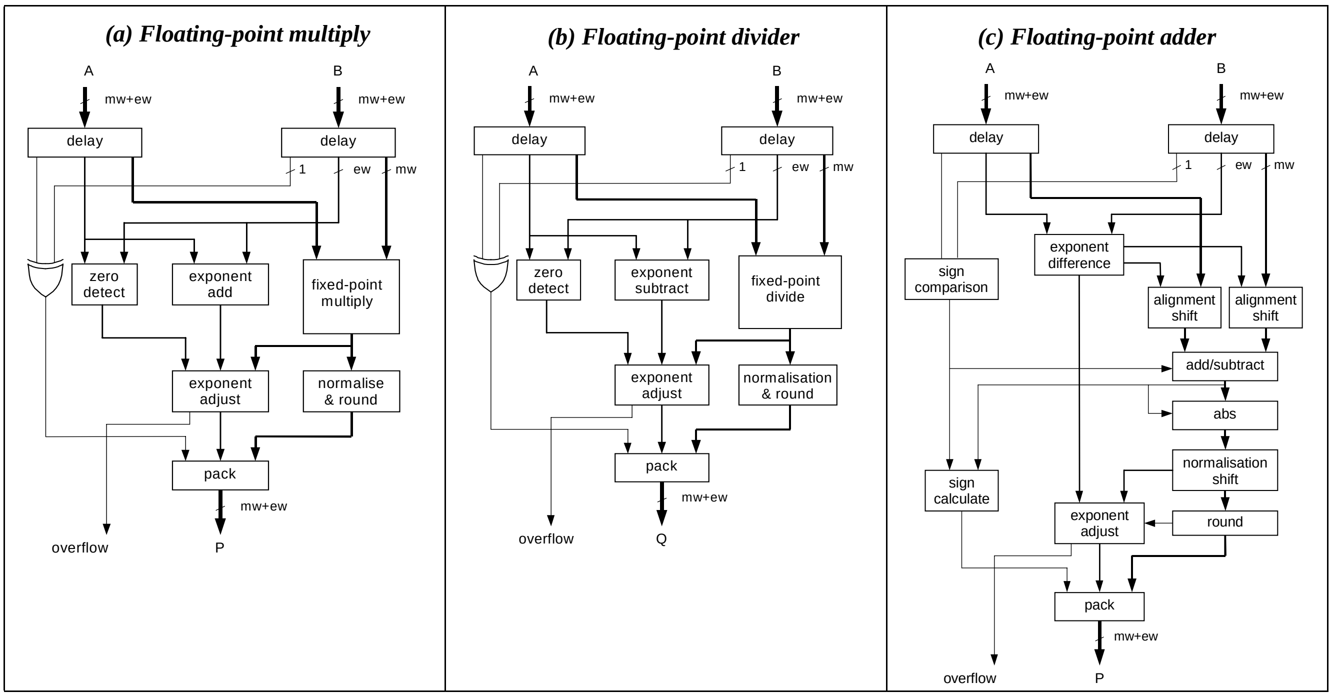

Figure 8: Block diagrams of floating-point operators

5.3 Floating-point divider

A block diagram of the floating-point divider is shown in Figure 8(b). Division is very similar to multiplication, except that

** IEEE round to nearest operation will deliver the nearest value with least significant bit equal to zero when exactly half way between representable values. This avoids any bias in the rounding error, but is costly to implement. Our QR simulations show no benefit from true round to nearest, so we do not use it in our implementations.*

we replace the fixed-point multiplier with a divider, subtract exponents rather than add them and perform a 1-bit right shift rather than left-shift to normalise (as $0.5 \le \frac{N}{D} < 2$). A non-restoring division algorithm has been employed which consists of a series of add/subtract operations.

5.4 Floating-point adder

A block diagram of the floating-point multiplier is shown in Figure 8(c). This is the most complex floating-point operator, as both mantissa alignment before addition and normalisation after addition are required. These operations may shift the mantissa over its full wordlength, requiring barrel-shifters which are expensive in FPGA technology.

The alignment operation equalises the exponents so that mantissas can be added, or subtracted (dependent upon their sign). The absolute value of the result is then obtained and left or right shifted to normalise the result. The result is then rounded, and renormalised if overflow occurs.

6. IMPLEMENTATION RESULTS

6.1 FPGA layout

Figure 9 shows an implementation of the linear QR processor on the XCV3200E employing 1 boundary and 2 internal cell processors. With an FPGA speed grade of -8 a clock rate of 150MHz is achieved. A mantissa size of 14-bits and exponent of 6-bits has been employed, so the array is capable of meeting relatively demanding radar applications (as shown in Figure 2). The architecture employed allows more than one number of columns of operations mapped to each internal cell processor, so the design shown is capable of performing QR for a wide range of problem sizes.

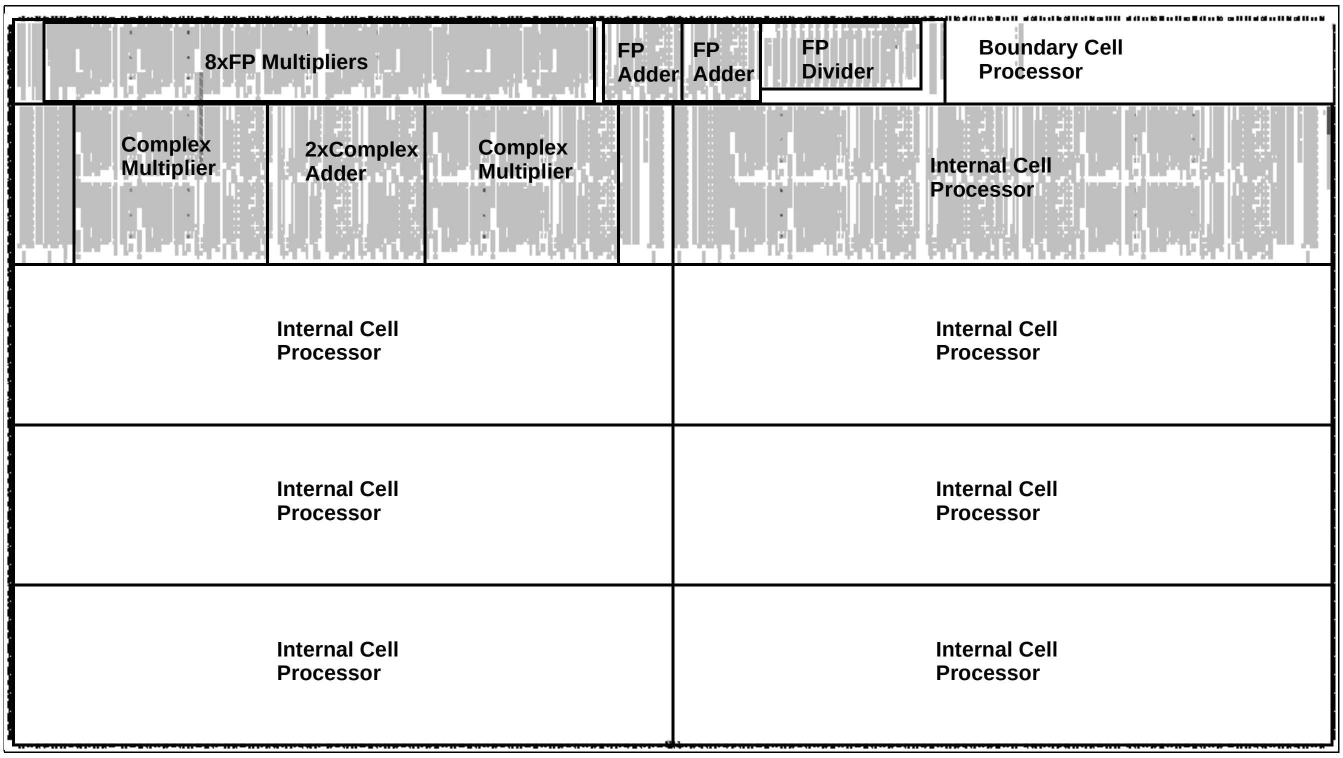

Figure 9: QR processor implementation on an XCV3200E-8

6.2 A 20 GFLOPS processor

Placement of the boundary and internal cell processors has been determined to allow expansion to 9 processors on a single XCV3200E FPGA. As indicated in the Figure 9, there is space for an additional 6 internal cell processors making a total of 9. Experience has shown that the use of locally interconnected processors whose placement has been completely predefined (i.e. realised as RPMs) allows designs to be scaled with little impact on timing. Therefore, a 9 processor implementation operating close to 150MHz is realistic*.

6.3 Performance

The throughput of the processor can be expressed in terms of number of floating-point operations performed per second (FLOPS). In Table 1 we compare the estimated throughput of the 9-processor FPGA implementation with programmable DSP and ASIC implementations. The number of FLOPS is calculated on the basis that boundary and internal cell processors employ 11 & 16 floating-point operations respectively.

The programmable DSP (pDSP) is represented by the TI TMS320C6701 processor. Programmable processors are poorly suited to the weight calculation problem due to the two cell types, their irregularity, and the division in the bourdary cell. Consequently, less than 40% utilisation of the processor execution units is achieved. The throughput is almost a factor of 100 lower than the FPGA implementation. Higher computation rates may be possible with a GPP such as the PowerPC 7400 (AltiVec processor).

The ASIC figures are based on a design using 0.35μm technology[15]. This is not state-of-the art technology, and we have scaled the thoughput by a factor of 6 to obtain estimates for more current 0.18μm technology. However, the comparison of FPGA with 0.35μm ASIC is useful, as this is more representative of what ASIC technology we would likely use given development timescales, budgets and volumes. When put in this light, the FPGA performance looks very attractive.

Table 1: Maximum throughput in FLOPS

Implementation approach	Clock (MHz)	Number of processors	MFLOPS
pDSP, TMS320C6701	*167*	*1*	*250*
FPGA, XCV3200E-8 (0.18μm)	*150*	*9*	*20,850*
ASIC, 0.18μm	*190*	*74*	*225,000*
ASIC, 0.35μm	*100*	*21*	*32,900*

b) **Weight update period:** A more useful interpretation of throughput to the radar system designer is the weight update period. This is summarised in Table 2 for a range of problem size *p*, and is based on *2p* data samples, followed by *p+1* constraint inputs for each beam.

Table 2: Weight-update period estimates

Number of beams	Period (ms) for XCV3200E-8			
	Number of antennas (*p*)			
	16	32	64	128
1	*5.55*	*42.7*	*335*	*2,649*
3	*9.18*	*71.7*	*560*	*4,424*

6.4 Scalability

Larger arrays can be obtained by implementing further internal cell processors on additional FPGAs interconnected in a linear array fashion (without memories or additional components). The input and output capabilities of current FPGAs readily support the required interconnect in this configuration. Furthermore, additional pipelining delays can be accomodated on the paths between FPGAs to maintain full clock speed. This represents a level of scalability difficult to achieve with pDSPs.

7. CONCLUSIONS

7.1 Floating-point on FPGA

By employing a highly parallel systollic array architecture we have shown that very high performance implementations of adaptive weight calculation are possible using FPGA technology. We have presented results which indicate that a single FPGA may perform in excess of 20 billion useful floating-point operations per second. This is over an order of magnitude

** A 9 processor design has not yet been implemented as the tools require large amounts of physical RAM.*

better than current programmable DSP technology, and clearly demonstrates that FPGA technology offers a viable means for implementing floating-point arithmetic when the wordlength requirements are not too great.

It should be emphasised that this performance has been achieved using a technology with little optimisation for DSP. With the embodiment of dedicated multipliers in Xilinx's next family of FPGA, Virtex 2, the level of DSP performance is will increase significantly.

7.2 Advantages of floating-point over other approaches

We are building an adaptive beamforming system based upon our floating-point FPGA core. One advantage of using floating-point arithmetic over special operations, such as CORDIC, is that they are well established and understood. As a consequence there is more acceptance of designs by system designers when based upon conventional operations.

At present, the floating-point implementation consumes a little more chip area than one based on a fixed-point CORDIC operator[10], but looking to the future, has more scope for optimisation both at the algorithm and implementation levels. Floating-point cores are likely to become standard, freely available and updated as FPGA technology progresses. Therefore, our array designs should be more portable than one based on less common operations. Furthermore, the use of floating-point arithmetic simplifies algorithm implementation. Enhancing the ability of FPGAs to realise floating-point more effectively will widen their application into more mainstream DSP.

8. ACKNOWLEDGEMENTS

This work was carried out as part of Technical Group 10 of the MoD Corporate Research Programme. I would like to acknowledge the work by Chris Booth on the implementation of the floating-point divider.

9. REFERENCES

1. A. Farina, *Antenna-Based Signal Processing Techniques for Radar Systems*, Artech House, 1991.
2. Xilinx Core Generator, http://support.xilinx.com/.
3. S. Haykin, Adaptive Filter Theory, 2nd Edition, Prentice Hall, ISBN 0-13-013236-5, 1991.
4. C. R. Ward, P. J. Hargrave, and J. G. McWhirter, "A Novel Algorithm and Architecture for Adaptive Digital Beamforming", *IEEE Trans. on Antennas and Propagation*, Vol. AP-34, No. 3, pp. 338-346, 1986.
5. J . G. McWhirter, "Recursive least-squares minimization using a systolic array", *Proc. SPIE 431, Real-Time Signal Processing VI*, pp. 105-112, 1983.
6. W. Givens, "Computation of Plane Unitary Rotations Transforming a General Matrix to Triangular Form", *J. Soc. Indust. Appl. Math.*, Vol. 6, No. 1, pp. 26-50, March 1958.
7. W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays", *Proc. SPIE 298, Real-Time Signal Processing IV*, pp. 19-26, 1981.
8. R. Döhler, "Squared Givens Rotations", *IMA J. of Numerical Analysis*, Vol. 11, pp. 1-5, 1991.
9. J. Volder, "The CORDIC Trigonometric Computing Technique", *IRE Trans. Electron. Comput.*, Vol. EC-8, pp. 330-334, 1959.
10. R. Walke, R. W. M. Smith and G. Lightbody, "Architectures for Adaptive Weight Calculation on ASIC and FPGA", *Proc. 33rd Asilomar Conference on Signals, Systems and Computers*, 1999.
11. S. Y. Kung, *VLSI Array Processors*, Prentice Hall, ISBN 0-13-942749-X, 1988.
12. G. M. Megson, *An Introduction to Systolic Algorithm Design*, Clarendon Press, ISBN 0-19-853813-8, 1992.
13. G. Lightbody, R. L. Walke, R. Woods, J. McCanny, "Linear QR Architecture for a Single Chip Adaptive Beamformer", Journal of VLSI Signal Processing, Vol. 24, pp. 67-81, 2000.
14. Alliance Series 3.1i Software Documentation: Libraries Guide, http://support.xilinx.com/
15. G. Lightbody, R. L. Walke, R. Woods, J. McCanny, "Novel Mapping of a Linear QR Architecture", *Proc. ICASSP*, vol. IV, pp. 1933-6, 1999.
16. *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985.