NOVEL MAPPING OF A LINEAR QR ARCHITECTURE

G. Lightbody[☆], *R. Walke*[•], *R. Woods*[☆] *and J. McCanny*[☆]

DSiP™ Laboratories, Queen's University of Belfast, Belfast, N. Ireland[☆]
DERA, St. Andrew's Road, Malvern, England[•]

ABSTRACT

This paper presents a novel architecture mapping technique which was essential in the design of a QR array which forms the core processor of a single chip adaptive beamforming system. The mapping technique assigns a QR triangular array of $2m^2+3m+1$ cells down onto a linear architecture of $m+1$ processors. The mapping results in a linear systolic architecture with one hundred percent hardware utilisation, local interconnects and individual processors for boundary and internal cell operations. In addition, this paper highlights the effect latency has on the validity of the linear architecture.

1. INTRODUCTION

Adaptive filters play a key role in applications where the statistics of the incoming signals are unknown or changing. One such application is in adaptive beamforming in which the system aims to suppress signals from every direction other than the desired "look direction" by forming null steering beams [1]. The weights used to generate these beams are calculated employing one of a range of adaptive algorithms, such as the Recursive Least Squares (RLS) or Least Mean Square (LMS) algorithms. There has been a considerable body of work devoted to algorithms and VLSI architectures for Recursive least Squares (RLS) filtering [2,3]. In particular, methods based on QR-decomposition (e.g. using Given's rotations) have been popular as they remove the computational bottleneck caused by matrix inversion. [4-7]. In addition, the basic structure of the classical QR algorithm lends itself for implementation on a highly parallel triangular array processor [4], offering all the characteristics of a systolic array.

The main focus of this paper is on the use of a novel architecture mapping technique which was applied in the development of a QR array processor. This design forms the core processor of a single chip implementation of an adaptive beamformer system capable of performing 50 GigaFlops and consisting of over 5 million transistors [8]. To suit a wide range of adaptive beamforming applications we have designed a 41 input QR array processor implementing the Enhanced-Squared Givens rotation algorithm [9]-a derivative of the conventional square root Givens rotation algorithm. The QR decomposition is performed on a set of complex input signals necessary to model the amplitude and phase of the incoming beams. The implementation of complex arithmetic has a dramatic effect on the level of computation as complex operations require multiple instances of real operations. The level of computation is depicted in the signal flow graphs for the QR cells shown in figure 8. With current IC technology, a direct implementation of the full 41-input triangular array (i.e. 861 cells) would not be practical. Results presented in [8] show that only 35 complex QR cell processors using a 16 bit floating point wordlength and 0.35 μm technology can fit onto a silicon area of 225mm^2.

The purpose of this paper is to present the architecture mapping technique used to reduce the number of required QR processors down to a feasible level. It is a novel method developed by Walke [9], which assigns a triangular array[1] of $(2m^2+3m+1)$ cells onto a linear architecture of $(m+1)$ processors. The features of the resulting architecture are that it requires dedicated processors for both the QR array operations (unlike [10]), which are locally interconnected and are used with 100 % efficiency, thus displaying the characteristics of a systolic array. The following section demonstrates the architecture mapping technique applied to a 7 input triangular array example. Section 3 analyses the affects of latency on the validity of the mapping technique; with the main conclusions presented in section 4.

2. MAPPING THE QR ARRAY

Conventional square-root Givens Rotations is one of a number of rotation methods which can be used to solve RLS by QR-decomposition, and can be implemented on a systolic triangular array processor such as that shown in Figure 1. The QR decomposition transforms a matrix **X** and vector **y** into an upper triangular matrix **R** and a vector **u** by a series of 2-dimensional Givens rotations, **Q**. In adaptive beamforming the inputs to the array (i.e. x_1 to x_6, and y_1) come from snapshots of the signals received by the system. These enter the array at the top and are processed down through the cells on each clock cycle. The rotation is achieved by computing two rotation parameters, a and b, within a boundary cell such that they eliminate the input x_{BC} (shown in figure 1). In the process, the **R** and **u** values are updated to account for the rotation. The a and b parameters are then passed unchanged along the row of internal cells continuing the rotation. The output values of the internal cells, x_{OUT}, become the input values for the next row. The elements of the **R** matrix and **u** vector are stored within each of the processors until updated by the next rotation process. Meanwhile, new inputs are fed into the top of the array and the process repeats.

The QR array in figure 1, employs a processor for each rotation operation. Implementing this array would be difficult enough without even considering the 41 input triangular array desired for our beamforming applications, (which would result in 861 cells). Therefore, a means is required to reduce the number of processors by incorporating a method of hardware sharing. The method we used is a novel assignment technique proposed by Walke [9] in which a triangular array with $2m+1$ inputs (i.e. $2m^2+3m+1$ cells) is assigned to a linear architecture of $m+1$

[1] The mapping depends on there being an odd number of inputs into the triangular array, i.e. $2m+1$.

processors. This procedure is depicted in the following diagrams in an example mapping of a 7 input triangular array shown in figure 1. The resulting linear architecture is shown in figure 5.

2.1 Derivation of linear architecture

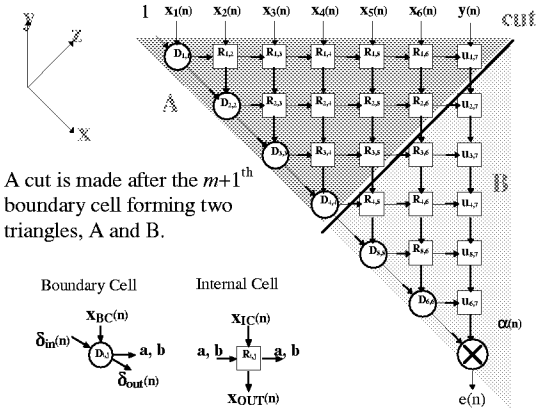

Figure 1: Complex SGR QR Array Signal Flow Graph[2]

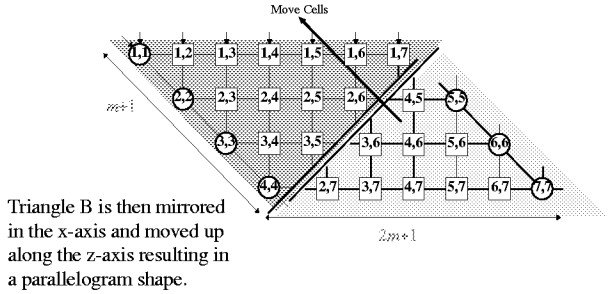

Figure 2: Modified array

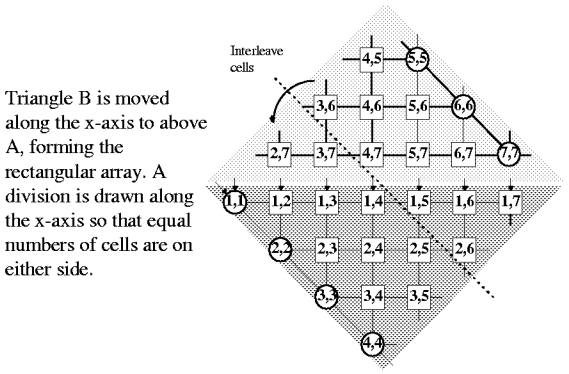

Figure 3: Folded array

The folded array in Figure 3 contains global connections. These are removed by projection onto a linear array. However, the connections are also transposed which must be removed first. This can be achieved by folding the array to interleave the processors as shown in Figure 4. This fold also places the boundary cell operations down back on one diagonal. By projecting down the diagonal it is possible to assign all the boundary cell operations to one boundary cell process and all the internal cell operations to a row of internal cell processors. The resulting linear architecture is shown in figure 5.

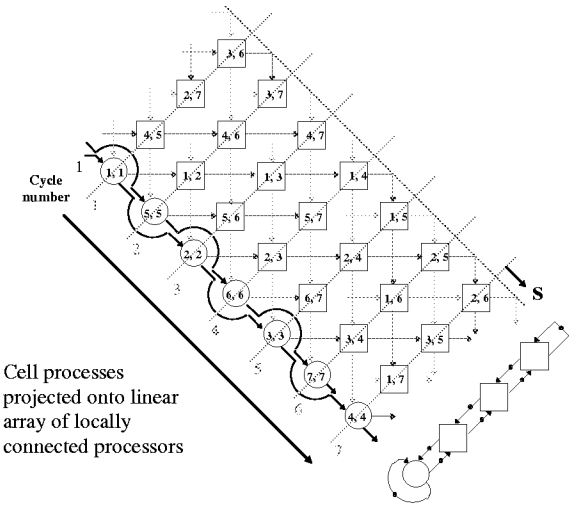

Figure 4: Locally connected linear array

The latches are present on all processor outputs to maintain the data between operations performed on one diagonal to the next. The latches for maintaining the parameters within the cells are also shown. In this case, a total of 7 latches are required; one for each diagonal of the 2D-array. Multiplexers are present at the top of the array so that inputs to the QR-array can be supplied to the cells at the right instances of time. The multiplexers at the bottom of the figure cater for the different directions of data flow that occur between rows of the original array (due to the second fold).

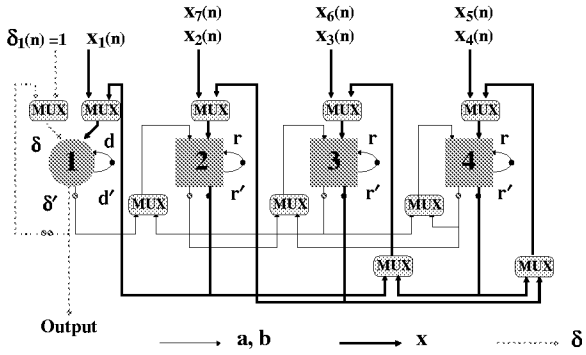

Figure 5: 4 cell linear architecture

2.2 Scheduling the QR operations

The order in which operations are performed on the linear array is identified by the schedule, shown in Figure 4 as the diagonal lines (referred to as hyperplanes). These cut across the operations in the array which should be performed at the same time. The schedule is denoted more compactly in Figure 4 by a schedule vector, S, normal to the hyperplanes. A valid schedule is obtained by ensuring that the data required by each set of scheduled operations is available at the time of execution. This

[2] The D term within the boundary cell processor is R^2, this removes the need for evaluating a square root. Also, in the linear array the last multiplication (cell 7,7) is performed on the boundary cell.

implies that the data must flow across the schedule lines in the direction of the schedule vector. This is true in Figure 4 except for the connections that pass from the bottom of the array to the top due to the first fold. These dependencies can be removed by delaying the dependent operations until the data is available. This results in a delay of one cycle of the array, which can be easily justified by considering the schedule of the unfolded array shown in Figure 6(a). Figure 6(b) shows the array and the schedule with the first fold.

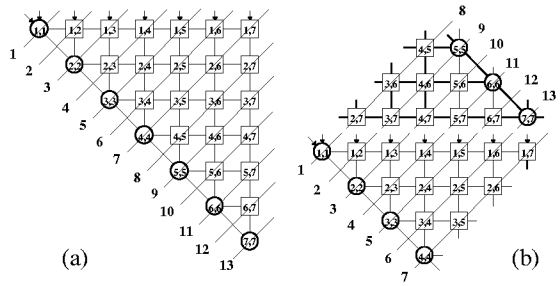

Figure 6: Scheduling the array

By continuing the schedule shown in figure 4 for a further 6 cycles we obtain the version shown in figure 7(a). The highlighted cells show the operations that are performed on each clock cycle for a specific QR operation. Figure 7(b) shows the previous and following QR operations interleaved before and after iteration n=1. To summarise the operation, the first QR operation begins at cycle =1 then after 7 cycles of the linear architecture the next QR operation begins. Likewise, after a further 7 cycles the third QR operation is started.

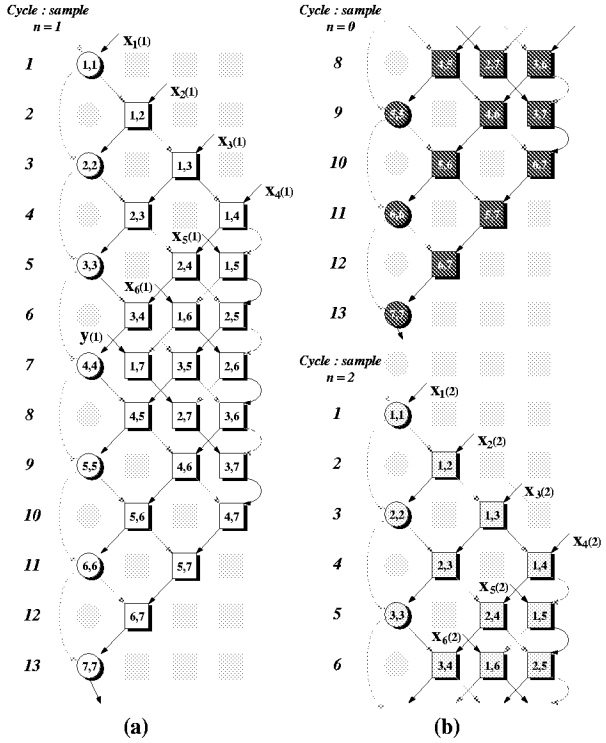

Figure 7: Interleaved QR iterations

From Figures 7(a) and 7(b) we can see that there are two types of **x** inputs into the QR cells. the first type, referred to as external **x** inputs, come from the snap shots of data forming the input **X** matrix and **y** vector. These inputs are fed into the linear architecture every 7, (i.e. $2m$+1), clock cycles. The second type of **x** input, referred to as internal **x** input, result from the transfer of x values from one internal cell to another. For the analysis of the schedule the **x** inputs will only be considered as the rotation parameters are set to input the QR cells at the same time instances as the internal x inputs.

3 RETIMING THE LINEAR ARCHITECTURE

The linear architecture discussed so far, (Figure 5), has single latches present on all processor outputs to maintain data between operations performed on one cycle to the next. In other words, the QR cells have a latency of one cycle. The mapping of the linear architecture is based on this factor. Hence, the scheduling of the external sample input data will not conflict with the internal cycles of **x** inputs. However, the inclusion of real timing issues may affect the validity of the linear array mapping. The arithmetic processors used to build the QR cells, such as multiplication and division, need to be pipelined to meet the timing performance demands of the circuit. These pipe stages incur a latency in the QR cells, as shown in the figure below.

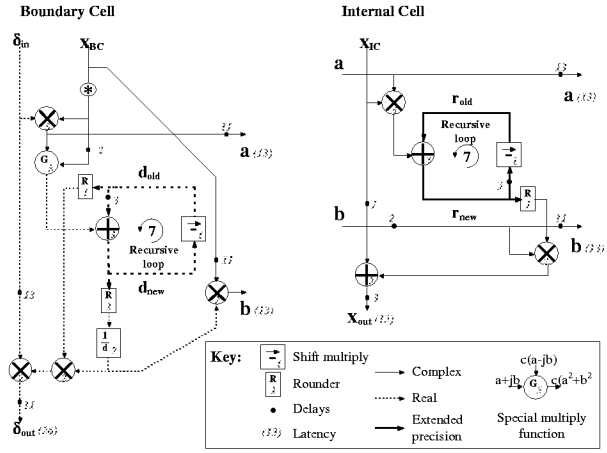

Figure 8: Retimed QR cells

The latency of the internal cell in producing the **x** output can be expressed generically as the term l. The latencies of the boundary cell in producing the rotation parameters and delta output are therefore l and $2l$ respectively. Figure 9 shows how these latencies will affect the schedule of the linear architecture.

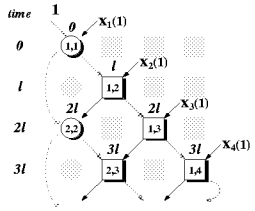

Figure 9: Effect latency has on the schedule

The latency of the QR cells has caused a complete shake up of the scheduling of the linear architecture. Looking at figure 7 we can see that iteration n=2 begins 7 clock cycles after the start of iteration n=1. However, the introduction of processor latency stretches out the scheduling diagram such that the n=2 iteration begins after 7*l* clock cycles. In this example the latency term *l* is 13. Therefore the second iteration would be beginning after 91 clock cycles. This is obviously not an optimum use of the linear architecture as it would only be in use every 13th clock cycle. Therefore, we need to introduce the new samples at a much faster rate. This factor is referred to as the number of clock cycles between successive external **x** inputs, T_{ext}. The latency within the recursive loop of a QR cell determines the minimum value of T_{ext}. In this example, (figure 8), the latency within the recursive loops for both types of QR cell is 4 clock cycles. Thus new samples could theoretically be fed into the linear architecture every 4th clock cycle. However, we need to ensure that increasing the rate of inputting external **x** inputs will not produce an invalid schedule. For example, if T_{ext} was set to 7 and *l* to14 then there would be a collision of input data into the boundary cell after 28 clock cycles, as depicted in figure 10.

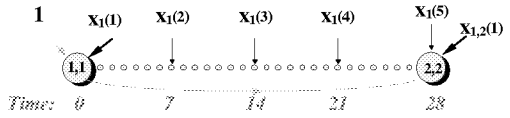

Figure 10: Invalid schedule

Another point to note is that the schedule of inputs has been stretched leaving gaps. The main objective was to ascertain the optimal combination of *l* and T_{ext} which ensured a valid schedule and 100% hardware utilisation. A MATLAB program was written to mathematically model the times for each input into each cell of the linear array. Tests were then carried out for input collisions and processor usage over a range of values of *l* and T_{ext}. The optimal values were used to finalise the timing of the linear architecture. The array was then modelled using Cadence's SPW™ tool. This allowed a quick validation of the scheduled inputs and control signals. The model was then simulated to confirm that its operation conformed identically with that of the original triangular array. This was the case.

4 CONCLUSIONS

The programs analysing the scheduling of the linear architecture have proven to be key tools in the design process. Retiming mapped architectures is an iterative process of retiming the QR cells, the linear architecture, then back again to the cells. The process is then repeated until an optimum solution is obtained. Then, with the addition of a single multiplexer the scheduling may be invalidated and the whole retiming process needs to begin again.

The choice of processor latency and scheduling are essential to the performance of the linear architecture. In the example shown the optimum value for T_{ext} is 7, and for *l* is 13. These values produce a fully 100% utilised linear architecture. However, if the latency term *l* was reduced to, for example, 3 and the T_{ext} was set to 21 then there would be gaps left within the schedule. These spaces could be filled by performing 2 additional independent QR operations almost in parallel with the initial one all on the same linear processor. So just by altering the schedule of the linear architecture a limited range of implementations are available.

The linear architecture offers characteristics of a systolic array. Firstly, there are two distinct processors for boundary and internal cell operations which enables the circuits to be optimised for a specific function. However, the disadvantage is the work needed to design and interface both VLSI circuits. Secondly, the linear architecture cells have only local interconnections with local storage within the QR cells.

The research within in this paper is part of a project to design cores to build a single chip for a rapid implementation of adaptive beamformer systems on silicon. To demonstrate their functionality a single chip beamformer is being implemented. The core of the design is using a linear architecture incorporating 21 QR processors, and processing up to 41 inputs. It consists of approximately 5 million transistors and has an area less than 200 mm^2 with a performance of 50 GigaFlops. The implementation is readily scaleable to meet the specific requirements of the application. The full beamformer system is currently being implemented and further details will be presented in due course.

5 REFERENCES

[1] L. C. Godara, "Application of antenna arrays to mobile communications, part II: Beam-forming and direction-of-arrival considerations", *Proc. of the IEEE,* Vol. 85, No. 8, pp. 1195-1245, Aug. 1997.

[2] Kalouptsidis and Theodoridis, Adaptive System Identification and Signal Processing Algorithms, Prentice Hall, Englewood Cliffs, NJ, 1993

[3] S. Haykin, *Adaptive Filter Theory,* Prentice Hall: Englewood Cliffs, NJ, 1986.

[4] W. M. Gentleman and H. T. Kung, "Matrix triangularisation by systolic array", *Proc. SPIE (Real-Time Signal Processing IV),* pp.329-369, 1973.

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

[6] J. M. Cioffi and T. Kailath, "Fast recursive-least-square, transversal filters for adaptive filtering," *IEEE Trans. Acoustics, Speech, Signal Processing,* vol. ASSP-32, No. 2, pp. 998-1005, 1984.

[7] S. F. Hsieh, K. J. R. Liu and K. Yao, "A Unified Approach for QRD-Based Recursive Least-Squares Estimation without Square Roots", *IEEE Trans. On Signal Processing,* Vol. 41, No. 3, pp. 1405-1409, March 1993.

[8] G. Lightbody, R. Walke, R. Woods, J. McCanny. "Rapid Design of a Single Chip Adaptive Beamformer", *IEEE Proc. on Signal Processing Systems,* to be published October 1998.

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

[10] C. M. Rader, "MUSE: A systolic array for adaptive nulling with 64 degrees of freedom using Givens transformations and wafer scale integration", *Proc. of the Int. Conf. Of Application Specific Array Processors,* pp. 277-291, 1992.