GENERIC SCHEDULING METHODS FOR A LINEAR QR ARRAY SoC PROCESSOR

Zhaohui Liu G. LightBody[*] R. Walke[+] Y.Hu[*] J.McCanny

DSiP Laboratory, School of Electrical and Electronics Engineering,
The Queen's University of Belfast, Belfast BT 9 5AH, N. Ireland
ISS Limited, 50 Malone Road, Belfast BT9 5BS, N Ireland[*]
DERA, St. Andrew's Road, Malvern, England[+]

ABSTRACT

A scheduling method for implementing a generic linear QR array processor architecture is presented. This improves on previous work. It also considerably simplifies the derivation of schedules for a folded linear system, where detailed account has to be taken of processor cell latency. The architecture and scheduling derived provide the basis of a generator for the rapid design of System-on-a-Chip (SoC) cores for QR decomposition.

1. INTRODUCTION

Recursive Least Squares (RLS) filtering is a key signal processing technique, which has a very widespread potential usage if this can be implemented cost effectively in silicon [1,2]. One such application is adaptive beamforming in which the system aims to suppress signals from every direction other than the desired "look direction" by forming null steering beams. For RLS filtering, methods based on QR-decomposition have been popular and the basic structure of the QR algorithm can be implemented using a triangular systolic array architecture [3,4]. In recent papers, we have considered the practical issue of implementing QR systolic systems on a single chip and show how a novel mapping and folding technique can be employed to produce a linear architecture with local communication and 100% efficiency [5,6]. The scheme presented involves quite complex scheduling and re-timing, the details of which can be dependent on specific implementation details. The purpose of this paper is to extend this work by presenting a simpler and much more generic scheduling scheme that automatically accounts for parameters such the number of input data values and processing cell latency. A key motivation has been the creation of a generic and re-usable architecture for the rapid design and synthesis of System-on-Chip (SoC) IP cores.

2. SCHEDULING AND TIMING OF LINEAR ARRAY

2.1 Linear array

The folding and projection used for the derivation of the linear architecture used is depicted in figures 1 and 2. This is based on the scheme described in reference [6]. This considers the example of a 7 input triangular array. Here correct operation is ensured by data flowing down the schedule lines. Obviously, the scheduling required is complex. In this example the architecture is constructed using one boundary cell and 3 internal cells, depicted by the circle and squares, respectively.

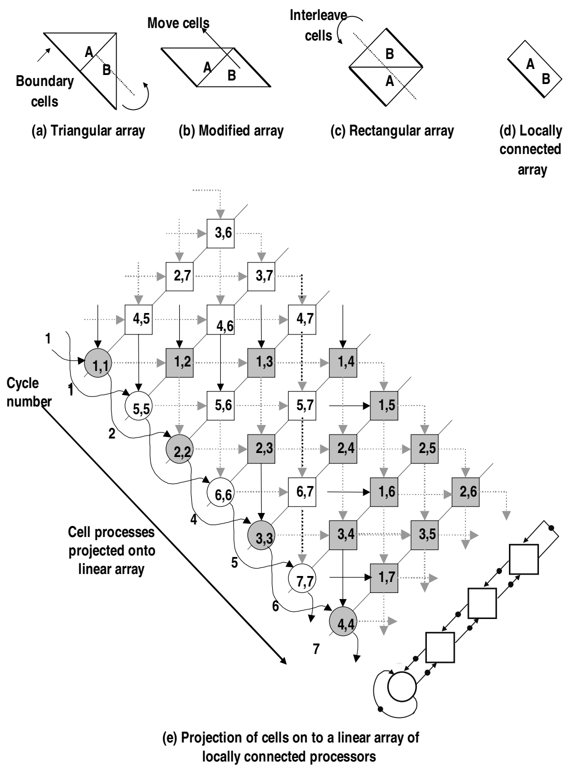
Move cells
Interleave cells
Boundary cells
(a) Triangular array
(b) Modified array
(c) Rectangular array
(d) Locally connected array
Cycle number
Cell processes projected onto linear array
(e) Projection of cells on to a linear array of locally connected processors

Figure 1: QR mapping to a linear architecture

The rotation parameters, *a* and *b*, are calculated in the boundary cell and these are passed unchanged along internal cells continuing the rotation. The output values of the internal cells, *x*, become the input values for the adjacent cells; meanwhile, new inputs are fed into the array. Multiplexers at the top of the array ensure the correct scheduling of data inputs. Those at the bottom cater for the different directions of data flow that occur between rows in Figure 1.

2.2 Scheduling

For N=2m+1 input channels, m+1 processors are required. In our previous work [6] it was shown that (a) the required data schedule is periodic repeating every $2m+1$ cycles, (b) external *x* inputs are also fed into the linear array, also every $2m+1$ cycles and (c) the input to internal cells come either from an adjacent

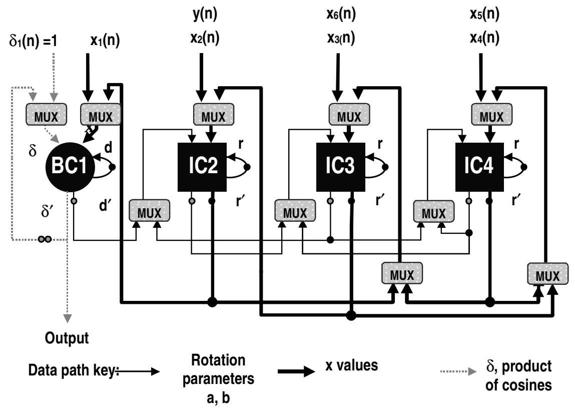

Figure 2. 7-input linear array

internal cell or are input externally.

From this, the detailed scheduling of the parameterized QR core is implemented in two steps. Firstly, we deduce the data scheduling principles related to the input channels when processor latency is unity. We then consider the influence of latency and scale the schedule sequence to derive the control data required for each processing cell.

Consider processing elements, labeled as PE_0, PE_1 $\dots PE_m$, where PE_0 is the boundary cell, and the others are internal cells. In our previous work [6], control was determined by whether the x input is derived from within the array or provided externally and by whether the (a, b) input direction is folded or unfolded. Because there is not a clear relationship between data flow and whether these input directions are folded, an alternative and more structured approach is the following. For each internal cell, PE_i, in Figure 2, four different possibilities for (a,b) and x exist (if PE_i is the last internal cell, let $PE_{i+1} = PE_i$):

- (a,b) comes from PE_{i-1}, and x comes from the output of PE_{i+1} - denoted as 0;
- (a,b) comes from PE_{i+1}, and x comes from the output of PE_{i-1} - denoted as 1;
- (a,b) comes from PE_{i-1}, and x comes from the external input - denoted as 2;

Table 1. Control data for schedule

a) Input channel number is 7

Index	PE_1	PE_2	PE_3
$j=0$	0	1	0
$j=1$	2	0	1
$j=2$	0	2	0
$j=3$	0	0	2
$j=4$	0	0	3
$j=5$	0	3	0
$j=6$	3	0	1

b) Input channel number is 9

Index	PE_1	PE_2	PE_3	PE_4
$j=0$	0	1	0	1
$j=1$	2	0	1	0
$j=2$	0	2	0	1
$j=3$	0	0	2	0
$j=4$	0	0	0	2
$j=5$	0	0	0	3
$j=6$	0	0	3	0
$j=7$	0	3	0	1
$j=8$	3	0	1	0

- (a,b) comes from PE_{i+1}, and x comes from the external input - denoted as 3;

At the same time, we express the control data index as

$$j = T \bmod (2m+1) \qquad (1)$$

Where (2m+1) stands for the scheduling period, and *T* is clock cycle. Using these labels and referring to the data flow in Figure 1, we can derive the following control signals for a schedule - based on unit latency - for the cases of 7 and 9 input channels respectively. This is presented in table 1. From this table, it is clear that the control signal changes regularly according to the input channel numbers. This can be represented mathematically. Letting $C_{i,j}$ be the control data of PE_i at the *jth* clock cycle then the control signal can be generated using a very simple procedure:

```
1)  for  $PE_m$ :
      $C_{m,m} = 2, \quad C_{m,m+1} = 3$
      for  $j = m-1, \cdots, 0$
          $\begin{cases} C_{m,j} = 0, & if \quad m-j \quad is\ odd, \\ C_{m,j} = 1, & if \quad m-j \quad is\ even; \end{cases}$
      for  $j = m+2, \cdots, 2m$
          $\begin{cases} C_{m,j} = 0, & if \quad j-m \quad is\ even, \\ C_{m,j} = 1, & if \quad j-m \quad is\ odd; \end{cases}$
2)  for  $i = 1, 2, \cdots, m-1,$
      $\begin{cases} C_{i,m} = 0 \\ C_{i,m+1} = 0 \end{cases}$
3)  for  $i = m-1, \cdots, 1$
        for  $j = m-1, \cdots, 0$
            $C_{i,j} = C_{i+1,\, j+1}$
        for  $j = 0, \cdots, m-1$
            $C_{i,j} = C_{i+1,\, j-1}$
```

2.3 Retiming

The above control data is only suitable in systems with unit latency. When latency is greater than unity, retiming is necessary to ensure a valid schedule and to maintain 100% hardware utilization. Because of the existence of latency, the interval between two continuous control data signals is therefore scaled accordingly i.e. by the latency. Moreover, in the case of the periodic control data signals, their sequence is shuffled. The relationship between the control signals and the clock cycle can thus be expressed as

$$(j \cdot L_{IC}) \bmod (2m+1) = T \bmod (2m+1) \qquad j \in [0, 2m] \qquad (2)$$

where L_{IC} is processing cell latency, $2m+1$ is the control data cycle, *j* is the index of control

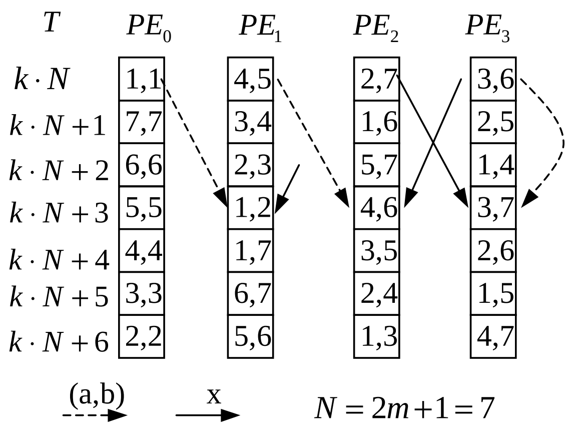

***Figure 3. Shuffle schedule sequence* ($L_{IC} = 3,\ 2m+1 = 7$)**

data and T is clock period. For example, if $T \bmod (2m+1) = 3$ then, from equation (2), $j = 1$ and thus from table 1(a), the control data is {2, 0, 1}. This defines how processors fetch (a, b) and x signals. Figure 3 shows the retimed schedule and depicts the fetching route for this example.

These control signals can be applied directly to the multiplexers shown in Figure 2 and thus it is very easy to change these signals to suit different array dimensions and cell latencies. When coupled with processor cells implemented using parameterized IP cores [6,7] (e.g. parameterized arithmetic processors) this provides a very elegant and direct approach to the creation of a generic core for implementing QR array processor systems in silicon. Typical parameters include the number of input values x, data wordlengths and levels of pipelining used within specific arithmetic building blocks.

3. DISCUSSION

Using the generic scheduling methods described, a parameterized QR core for adaptive beamforming has been captured in a hierarchical fashion using VHDL [7]. The operation of this core has been verified over a wide range of parameters and shown to be suitable for creating and synthesizing silicon QR systems covering all practical array dimensions and wordlengths and processing cell latencies. For example, based on 0.35 CMOS triple-layer technology, studies show that 60 complex QR cells, using 12bit wordlengths, can fit onto a single chip. (The corresponding number is 35 cells for the 16-bit case). This allows such systems to be rapidly created and the system to be rapidly redesigned for different application specifications, such as adaptive beamforming / noise cancellation for radar, sonar and mobile communications.

REFERENCES

1. S. Haykin, “Adaptive Filter Theory”, Prentice Hall: Englewood Cliffs, NJ, 1986.
2. 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.
3. J. G. McWhirter, “Recursive least squares minimization using systolic array”, Proc. SPIE IV, pp. 105-112, 1983
4. T. J. Shepherd and J. G. McWhirter, “Systolic adaptive beamforming”, chap.5, Array signal processing, S. Haykin, J. Litva, and T. J. Shepherd (Eds.), Springer-Verlag, ISBN 3-440-55224, 1993, pp. 153-243.
5. R.L. Walke, “High sample rate givens rotations for recursive least squares”, Ph.D. Thesis, University of Warwick, 1997.
6. G. lightbody, R.Walke, R. Woods and J.McCanny. “Linear QR architecture for a single chip adaptive beamformer”, Journal of VLSI signal processing system, vol. 24, pp. 67-81, 2000.
7. J. McCanny, D. Ridge, Y. Hu and J. Hunter. “Hierarchical VHDL libraries for DSP ASIC design”, IEEE International conference on Acoustics, Speech and Signal Processing ICASSP ’97, Munich, pp 675-678