CONFIGURABLE HARDWARE: TWO CASE STUDIES OF MICRO-GRAIN COMPUTATION.

Tom Kean
University of Edinburgh.
John Gray
European Silicon Structures Ltd.

Abstract
This paper describes a new VLSI architecture - Configurable Array Logic (CAL) which, at its lowest level, can be programmed electrically to implement any circuit composed of gates. At higher levels the technology provides a medium for the direct implementation of algorithms. It particularly addresses systolic and cellular automaton algorithms where the basic computational elements perform computations unsuited to conventional processors.

1 Introduction.

In their seminal paper [1] Foster and Kung argued for a computer architecture based around the classic Von Neumann processor and memory but with a number of special purpose VLSI chips added to the bus. These chips would implement systolic algorithms to provide high performance computation of important functions such as pattern matching, fast fourier transform and sorting. Foster and Kung describe a methodology for the implementation of such chips based on careful algorithm design and simplified and formalised layout techniques. Despite the considerable potential performance advantages of this architecture it has not been succesfully adopted in any common computer design to date. Commercial designers have not been able to justify the large design cost of many special purpose chips, all of which would require different support from applications programs in order to function effectively.

In this paper we will present an alternative approach to the implementation of systolic algorithms within a conventional computer system: instead of a number of special purpose systolic chips a single configurable computing surface is provided. This architecture is termed Configurable Array Logic (CAL) because it is best viewed as a reconfigurable hardware implementation style: that is, algorithms are programmed by specifying connections between active logic elements, via a programmable switching structure, rather than as instruction data to be interpreted by processing units. The advantage of this is that the programmable structure can implement bit level systolic algorithms unsuited to arrays of conventional processors.

This approach is illustrated by direct conversion of two well known algorithms previously implemented in special purpose silicon ([1], [2]) to the

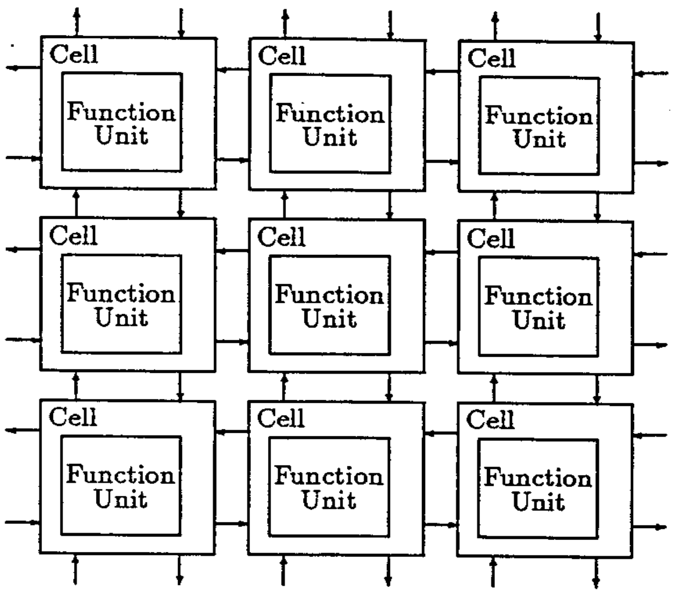

Figure 1: Basic Structure.

programmable structure. The ease with which this transformation can be done suggests that the programmable structure is suitable for use as a general purpose systolic coprocessor.

2 Architecture.

The basic configurable logic structure is a rectangular array of identical cells with the same orientation and nearest neighbour connections (figure 1). Each cell can compute any logic function of two variables or a simple D type latch and route the result to any neighbouring cell. It can also support all 'useful' permutations of input and output neighbour connections. As well as neighbour connections each cell has 2 global input signals; G1 and G2 which can be used as a two phase non-overlapping clock and can be programmed to drive a single global output signal, FTEST. FTEST is intended to support debugging of user designs by allowing internal signals to be monitored. Arbitrary sized arrays of cells can be built up using multiple chips at the board level: the chip boundaries are transparent to user designs.

More detailed description of the CAL architecture is presented elsewhere [3, 4, 5]. Alternative configurable architectures are described in [6] [7].

3 CAL Implementation.

At present a prototype CAL chip implemented in $2\mu m$ double metal CMOS is available. This chip contains a 16x16 array of configurable cells and is packaged in a 144 pin PGA package. A commercial version of the chip is under development in $1.5\mu m$ CMOS which will contain a 32x32 array of cells [8]. This chip will also come in a 144 pin package: a novel pad sharing scheme is used to allow all 32 inputs and outputs on each side of the array to communicate with neighbouring chips. Communication to non-CAL chips takes place at normal logic levels and has no special timing requirements.

CAL chips are programmed by writing into a static RAM which controls the configuration circuits. In the initial version of the chip this RAM appears to the outside world as a long shift-register and programming is via a serial data stream. Future versions of the chip will provide normal address decoding allowing the control RAM to be mapped into the memory of a host computer.

4 Systolic String Matching.

The systolic string matching algorithm of Foster and Kung [1] is a classic example of a linear systolic array. In this section we will convert the VLSI layouts of the original paper into an equivalent configurable logic design.

In order to implement the string matching algorithm two functional cells must be designed: the one bit comparator and the accumulator. These cells are then arrayed as shown in figure 2 (this figure is based on one in [1]).

The comparator computes the following function: $p_{out} \leftarrow p_{in}$, $s_{out} \leftarrow s_{in}$ $\omega_{out} \leftarrow \omega_{in}.(p_{in} = s_{in})$. The variables p and s represent single bits of the pattern we are matching against and the string respectively, ω indicates that the match has been sucessful up to this bit position. The cellular layout is shown in figure 3: on this diagram the grey boxes represent individual cells within the array, the black box in the centre of some cells represents the function unit, its two inputs are on the left and right hand side and its output comes from the centre. The function being computed is printed at the top left hand corner of the grey box. Black lines represent connections through the routing structure, in some cases the label G1 or G2 is printed beside the left function unit input (the clock line when the function unit implements a latch) to indicate it is being driven by a global signal. Note that this unit is designed to allow a one cell vertical overlap with the adjacent unit in the array.

The accumulator computes the following functions: $\lambda_{in} \rightarrow \lambda_{out}$, $\Omega_{in} \rightarrow \Omega_{out}$, $TEMP.\lambda_{in} + m_{in}.\overline{\lambda_{in}} \rightarrow m_{out}$, $\lambda_{in} + TEMP.(\Omega_{in} + \omega_{in}) \rightarrow TEMP$. The state variable $TEMP$ indicates that so far there has been a match up to this character position, m is the result, λ is used to mark the end of the pattern to initialise $TEMP$, Ω allows wild card matching at this position, ω indicates a succesful character match at this position.

The overall string matcher is composed of an array of these two cell types,

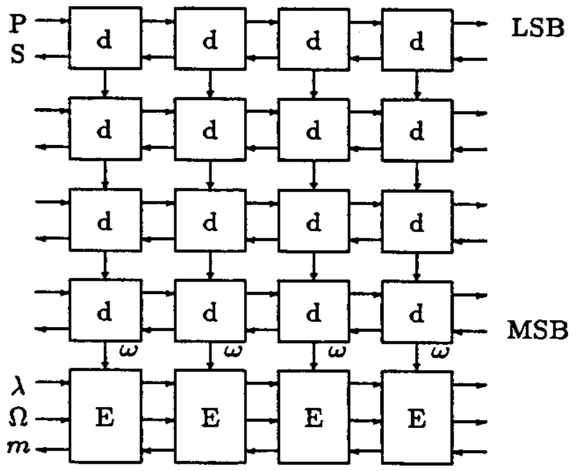

Figure 2: Systolic String Pattern Matcher.

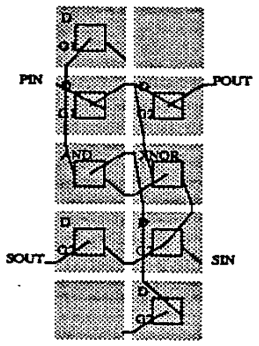

Figure 3: One Bit Comparator Layout.

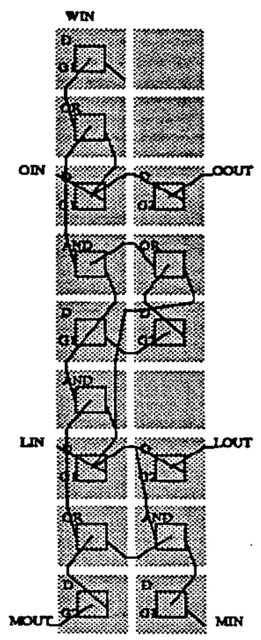

Figure 4: Accumulator Layout.

the comparator cells on the top are slightly different because there is no ω_{in} from previous comparators. Figure 5 shows a 3 bit 4 stage comparator implemented in a 20x8 array of cells: it is obvious that reasonable sized comparators could easily be implemented in a multiple chip array.

5 Bit Serial Multiplication.

Implementation of multiplication is arguably the most critical component of most hardware Digital Signal Processing systems. In this section we will illustrate the implementation of a pipelined bit-serial multiplier using configurable logic. The multiplier design is based on that used in the FIRST silicon compiler [2] allowing comparison of the two implementation styles. The implementation of the multiplier also illustrates the potential to build a complete library of FIRST primitives in configurable logic allowing implementation of FIRST DSP designs using the configurable technology.

In many applications (e.g. fixed response FIR filters) the multiplier coefficients are parameters which are fixed or change much less frequently than the multiplicand data. In the case of a VLSI multiplier it is still necessary to support variable coefficients in order to increase the range of usefulness of the chip but in a configurable logic implementation it makes much more sense to write a fixed coefficient multiplier generator (i.e. a program which can generate

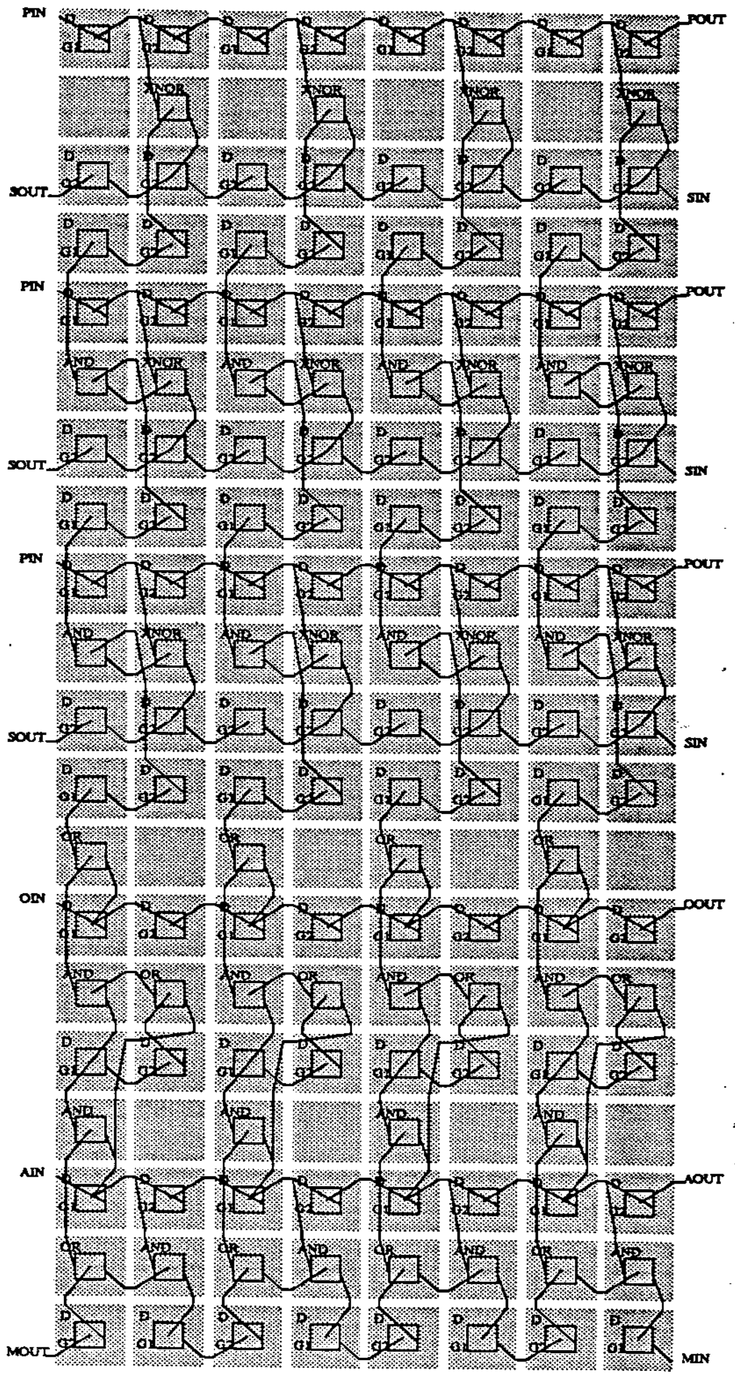

Figure 5: String Matcher Layout.

fixed coefficient multiplier designs for particular coefficients) and reconfigure the chip as necessary.

Note that this is an example of a general technique: analogous techniques are often used for computation expensive processes on conventional computers. For example, the GOALIE program for artwork analysis of integrated circuits generates a program for a particular set of design rules which is then run on input data rather than using a parameterised program capable of handling general design rules. This allows conditionals to be eliminated within key inner loop code which, combined with loop unrolling and other transforms provides a factor of two speedup [10]. This technique is central to the efficient use of configurable logic in many applications and it is conjectured that in many cases it can recover much of the area/time penalties over conventional hardware incurred by the programming circuitry.

The FIRST multiplier uses a modified Booth algorithm and the block diagram of a single stage is shown in figure 6. Data and control signals are fed from right to left through the stage with 3 bits of delay, requiring 6 latches per signal. The block LATCH_AND_COEFF is concerned with latching the coefficient which is fed in bit serially like the data. A latch signal is shifted in in parallel with the coefficient which when high causes the coefficient bits to be latched into the recoder unit prior to multiplication. The block labelled LATCH_AND_RECODE decodes three bits of the coefficient according to to table 1 and generates control signals for the selector and the programmable adder/subtractor (ADDSUB). The block labelled SELECTOR outputs 0,a or $2a$ to ADDSUB depending on the control inputs (x_0, x_1, x_2), a and $2a$ are available on the three bit delay line DATA_LINE. The block C1_LINE is concerned with an initialisation signal. ADDSUB is a programmable carry save adder/subtracter with sign extension and settable carry, it adds the appropriate value to the current partial product sum *pps* and passes the result to the next stage.

Fixed coefficient multipliers can be much smaller and faster than variable coefficient multipliers, the savings are greatest when a recoding scheme is used since the recoding can be chosen to minimise the number of stages where addition and subtraction are required. In many cases fewer than half the stages will require adders or subtractors [9]. In a fixed coefficient design we can eliminate the LATCH_AND_COEFFICIENT, LATCH_AND_RECODE and SELECT blocks. We can also replace the complex programmable block marked ADDSUB with a simple carry-save adder or subtractor or eliminate it entirely depending on the coefficient bits for a given stage. Thus at each stage of the multiplier we replace a single complex design with one of a number of much simpler designs selected automatically by a multiplier-generator program according to the coefficient. The fixed coefficient stages have approximately 1/4 the hardware of the programmable stages and are considerably faster because the critical path is much smaller.

A layout of a 'typical' stage in the multiplier is shown in figure 7: this consists of delay elements and a carry save adder with presettable carry. This

b_{i+1}	b_i	b_{i-1}	*Operation*
0	0	0	$PP \leftarrow (1/4)PP$
0	0	1	$PP \leftarrow (1/4)PP + a$
0	1	0	$PP \leftarrow (1/4)PP + a$
0	1	1	$PP \leftarrow (1/4)PP + 2a$
1	0	0	$PP \leftarrow (1/4)PP - 2a$
1	0	1	$PP \leftarrow (1/4)PP - a$
1	1	0	$PP \leftarrow (1/4)PP - a$
1	1	1	$PP \leftarrow (1/4)PP$

Table 1: Multiplier Recoding.

stage design corresponds to the operation $PP \leftarrow (1/4)PP + a$ in the recoding, table 1. The other operations in the table are computed similarly by substituting a subtracter and taking the $2a$ signal from after the third delay element on the data input as required. Since two bits of coefficient are dealt with in each stage the number of stages required is half the number of coefficient bits. The first and last stage designs in the array are slightly simpler than the general stage.

6 Conclusions.

A novel approach to implementing systolic algorithms for important applications within a conventional computer has been described. Instead of providing a variety of special purpose chips a single fine grain computing surface is used which can be configured to emulate arbitrary gate level circuits. Many systolic algorithms typically implemented in silicon can easily be mapped onto this surface. This mapping is illustrated for two well known algorithms.

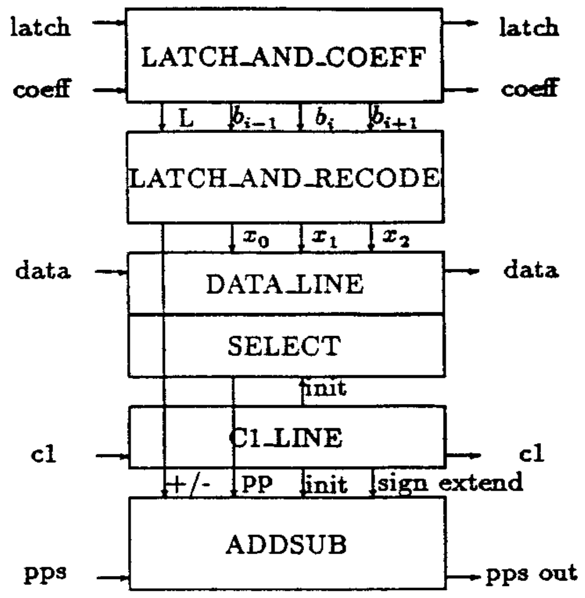

Figure 6: Serial Multiplier Stage - Block Diagram.

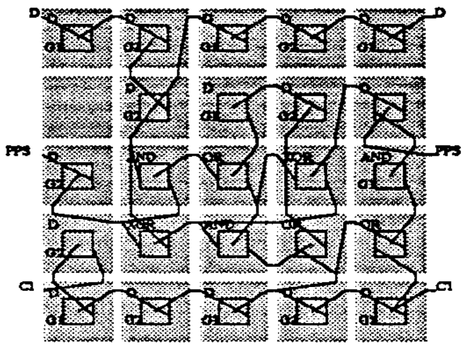

Figure 7: Multiplier Stage Layout.

References

[1] M.J. Foster and H.T. Kung. *The Design of Special-Purpose VLSI Chips.* IEEE Computer, January 1980, pp 26–40.

[2] Peter Denyer and David Renshaw. *VLSI Signal Processing: A Bit-Serial Approach.* Addison Wesley, 1985.

[3] Tom Kean. *Configurable Logic: A Dynamically Programmable Cellular Architecture and its VLSI Implementation.* PhD Thesis. University of Edinburgh, Dept. of Computer Science, 1989.

[4] J.P. Gray and T.A. Kean. *Configurable Hardware: A New Paradigm for Computation.* To Appear in Proc. Decennial Caltech Conference on VLSI, Pasadena CA, March 1989.

[5] Jouko Viitanen and Tom Kean. *Image Pattern Recognition using Configurable Logic Cell Arrays.* To Appear in Proc. Computer Graphics International, Leeds UK, June 1989.

[6] R.C. Minnick. *A Survey of Microcellular Research.* J. ACM, 14(2):203–241, April 1967.

[7] Xilinx Inc. *The Programmable Gate Array Design Handbook,* San Jose, CA., 1986.

[8] Algotronix Ltd. *CAL1024 Preliminary Data Sheet.* Edinburgh UK, 1989.

[9] R.F. Lyon. *Two's Complement Pipeline Multipliers.* IEEE Transactions on Communications, April 1976

[10] Thomas G. Szymanski and Cristopher J. Van Wyk. *GOALIE: A Space Efficient System for VLSI Artwork Analysis.* Proc. International Conference on Computer Aided Design, 1984.