A Digit-Serial Silicon Compiler

Richard I. Hartley and Peter F. Corbett

General Electric R&D Center,
Schenectady, NY, 12309.

ABSTRACT

A new silicon compiler is described, called PARSIFAL (not an acronym). It constructs chips with a data flow architecture in which data is passed in a digit-wide pipeline from one computational element to the next. The size of a digit may be specified by the user to be any value between one and the full word size of the chip. A digit size of one gives bit-serial chips whereas a digit-size equal to the word-size gives fully parallel computation. It is shown that an intermediate value of digit-size usually gives the most efficient chips in terms of throughput per unit area.

1. Introduction

The compiler described in this paper grew out of earlier work on bit-serial computation reported in [5] and [8]. The GE Bit-Serial Compiler (BSSC) has been used for a number of chip designs and has given good results for applications with moderate throughput. For higher throughput applications, however, it was clear that a move towards higher width operations was necessary. On the other hand, fully parallel arithmetic operators did not seem to be the best solution, since the overhead of carrying full parallel buses was seen as prohibitive. The size of individual operators would become a further limiting factor.

The compromise position of carrying out operations digit by digit seemed to combine the advantages of bit-serial computation (limited interconnect and small operator size) with that of fully parallel computation (higher throughput). By making the digit size variable from design to design, the capability is provided of making a trade off between throughput and chip area.

2. Previous Work

In the past few years, bit-serial design has been much studied, especially as a vehicle for the automatic generation of chips (using silicon compilers). For instance, the FIRST compiler ([1]), Cathedral ([4]) and the BSSC ([8]) have been successful in producing good quality bit-serial designs.

To avoid the disadvantages of bit-serial and fully-parallel architectures, the compromise of using a digit-serial architecture has been mentioned occasionally in published work. Smith, McGregor and Denyer [6] discuss various methods for increasing throughput of bit serial architectures and suggest the use of radix-4 (2 bit digit size) computation to increase performance. Smith and Denyer ([7]) report on a set of radix-4 computational modules. Our point of view is that a given fixed digit size is not sufficiently flexible to allow the designer a full range of trade-off choices. In our system, the designer may choose to make the digit width any divisor of the word size. For instance for a 16 bit word size, digit sizes of 1, 2, 4, 8 or 16 are allowed.

3. HARDWARE DESCRIPTION

3.1. The PARSIFAL architecture

The basic architecture used in PARSIFAL is similar to the bit-serial architecture proposed in [3] and used in the FIRST [1] and the BSSC [5] compilers. Figure 0 shows an example of a simple circuit conforming to our architecture. The circuit of figure 0 will calculate $|a+b|$ for 16-bit inputs a and b. (The Δ symbols represent clocked delays of the number of cycles indicated.)

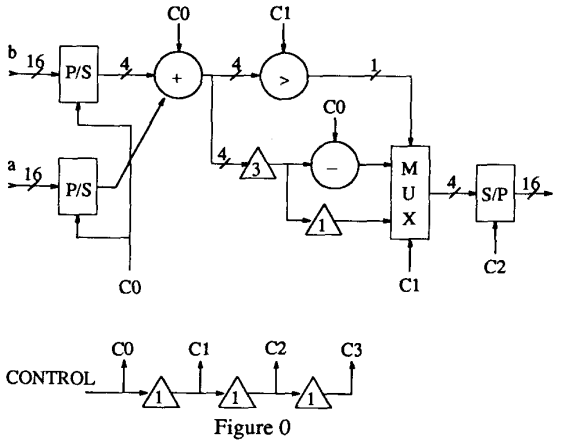

Figure 0

The chip architecture is a maximally pipelined one in which data flows from one computational element to the next. Synchronization of the pipeline is done by insertion of shift register delays between operators.

The main difference between the bitserial architecture and that of PARSIFAL is that signals are not all restricted to width 1 (as in the bit-serial architecture) but may have any width. The width of a signal is the number of wires required to carry it.

Thus signals may enter from the chip as full word-size signals, pass through parallel/digit-serial converters for internal calculations and then be converted to parallel signals for output again. Most arithmetic values have width equal to the prevailing digit-width, whereas logic operands (such as results of comparisons) have width one. However, double precision and complex signals are also supported and truncation and sign extension of parallel signals to give signals of different width is also possible.

The standard library of cells for performing arithmetical operations is designed to support operations on digit-width values. Suppose that the word size of arithmetic values is W bits and that the digit width is N. Arithmetic values are made up of W/N digits that are transmitted over N wires serially, least significant digit first. Values are represented in twos' complement format with the conceptional binary point in any desired position. The number of clock cycles required to transmit one complete word, W/N cycles, is known as a *sample period*. Similarly, logic values have width 1, but each value has a duration of one sample period. Thus, a **true** value is represented by W/N consecutive 1 values and a **false** is made up of W/N zeros transmitted serially. Thus, all computational data is divided up into words of *length* W/N. The throughput of the circuit is one complete calculation every sample period.

3.2. Control

A circuit produced from the basic cell library using PARSIFAL is largely self controlled. All that is necessary is that each cell knows when each new data word begins. Since, because of latency, the beginning of the data word will vary in time from operator to operator in the circuit, each operator must be notified at a different time of the beginning of the data word.

Control is centralized in a CONTROLLER cell that accepts a single input

called CONTROL that may be an input of the chip, or else may be generated internally. The CONTROL signal is high for one clock cycle of each sample period (in the MSD), and otherwise low. The CONTROLLER cell produces delayed versions of this signal. The delayed versions are the same as the input, except that the high cycle is shifted in time. The properly delayed CONTROL signal is now routed to each operator in the chip. If the sample size is W and the nibble-size is N, then there are only W/N distinct CONTROL signals because of periodicity of CONTROL. Thus the overhead in the circuit given over to centralized control is minimal.

3.3. Digit-serial Operators

The basic design of the individual computational elements is described next.

Consider a digit-serial adder, for illustration purposes 4 bits wide. This circuit accepts 4 bits of data from each of two inputs, A and B in each clock cycle. There is also a carry in at the least significant bit. It produces 4 bits of result and a carry out of the most significant bit. The carry in is reset to zero at the start of each data word. This reset operation is controlled by a special CONTROL signal that is high only in the last digit of each word. The 4-bit serial adder is shown in figure 1.

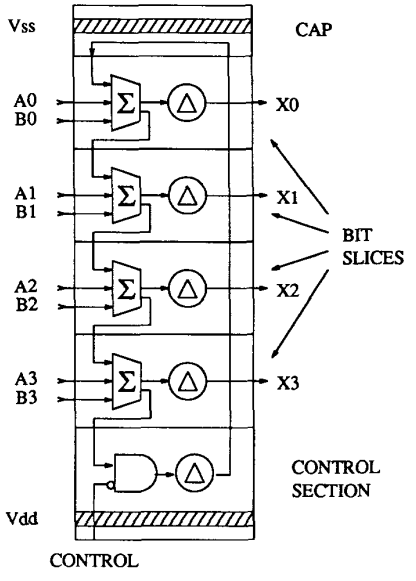

Figure 1. Digit-serial adder

As is seen, the serial adder consists of a control section that fields the carry out signal from the most significant bit of the digit, delays it one cycle and returns it to the least significant digit bit in the next clock cycle. It resets it to zero at the start of each sample. In addition, there are 4 identical bit slices each containing a full adder, and a cap cell that has no circuitry but is used to carry a power bus and to return the carry signal to the top of the stack. Most of the digit-serial operators can be constructed in a similar way, though some, for instance multiplication, are more complicated.

3.4. The Basic Template

Figure 2 shows the basic template for all cells in the digit-serial library (though some cells show slight variations on this theme). It consists of a control cell, n bit slices (n is the digit width) and a cap cell.

- The control cell will typically carry out tasks such as delaying and resetting carries, buffering and inverting clock signals and any other control logic that is necessary. It also carries the VDD power bus. The precise function of each control cell will vary from operator to operator.
- The operator bit-slice is to be repeated n times in the stack. It contains the main computational part of each operator.
- The cap cell is usually used only to carry the VSS power bus and to make minor routing connections.

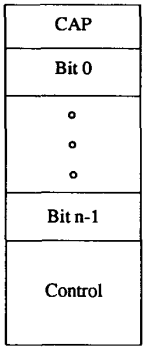

Figure 2 - Basic Template

For all operators that conform to the basic template, the height of each of these cell types is constant. Thus if A and B are two such operators, the control cell for cells A and B will be equal in height. Similarly, the bit-slices of A and B will be equal in height as will the cap cells. The width of the cells may be different for operators A and B however. As a result of this height rule, the total height of n-bit cells for operators A and B will be the same. The height of a stacked cell will be

$$total\_height = control\_height + cap\_height + n \times slice\_height$$

Furthermore, the power and clock lines, being at standard locations in the control and cap cells will match for the two cells. Because of this, cells may be placed side by side (perhaps with a small routing area between them) in rows of cells of equal height. This is important so that cells may be placed and routed with efficient standard-cell place and route methods.

As a general principle (though it is not strictly enforced) the inputs and outputs are respectively on the left and right of each bit-slice at the same level. This allows connections between cells to be made by abutment, thereby further reducing routing cost.

When the user specifies a digit size for a design, a set of digit-serial operators is constructed automatically by stacking basic library cells according to the template. These set of operators are then used to implement the required logical and arithmetic functions. All the cells will be of the same height, although their widths will vary.

In some cases, it is not possible to conform exactly to the standard template and some operators are constructed to slightly different templates. This is nevertheless done in such a way as to maintain the same cell height.

4. Speed of digit-serial computation

We illustrate the speed performance of digit-serial computation by considering as an example a 16 bit adder. Assume that our basic functional unit for performing addition is a one bit full adder. A 16-bit ripple-carry adder may be constructed by chaining 16 of these modules together. A bit-serial adder on the other hand consists of one full adder plus circuitry to latch and reset the carry bit. The reset of the carry bit is done using a control signal which must be provided to mark the end of a word.

The circuitry to latch and reset the carry represents an overhead in area for the bit-serial operator. This overhead circuitry is not present in the full width version. The idea behind digit-serial computation is to make better use of this overhead circuitry and increase the throughput of the operator without going to a full width solution. Figure 1 shows how a digit-serial adder works.

The advantage of digit-serial computation is that clock speeds may be higher than for full parallel operation, since the length of the ripple is reduced. Let us consider the area, A, and time constant, T, of parallel and digit-serial computational elements. (Define the time constant, T to be the time required to complete one complete word addition.) For the full width 16 bit adder, we have $A = 16a$ where a is the area of a full adder, and $T = 16t + t_0$ where t is the time for one bit of addition to complete and t_0 is a minimum overhead time requirement.

The overhead requirement includes time lost because of non-overlap of clock phases, allowances due to clock skew and propagation time for the output to reach the next operand.

For an adder of half width (that is, digit-size 8), we have area $A = 8a + a_0$ where a_0 is the overhead area, and $T = 2 \times (8t + t_0)$. If t_0 and a_0 are relatively small, then the half width adder gives almost equivalent throughput to the full width adder while using just over half the area.

In a general case where word size is W and digit-size is N, we have $A = Na + a_0$ and $T = W/N \times (Nt + t_0)$

Figures 3 and 4 show the way that area and throughput increase as the digit-size of a circuit is increased, the word-size being held fixed. Values of a, t and t_0 have been chosen for these curves from realistic simulation and layout area estimates, namely t = 2.5ns, t_0 = 20ns, a_0=2×a. As can be seen, both throughput and area increase as the digit-size increases.

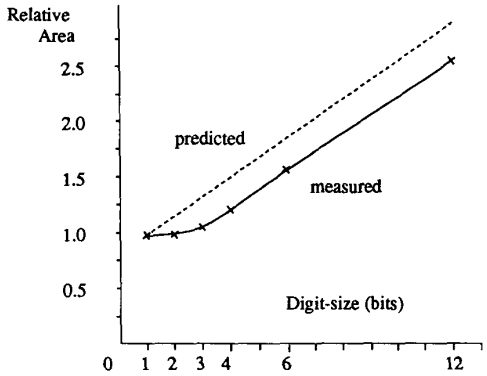

Figure 3. Area Penalty Curve

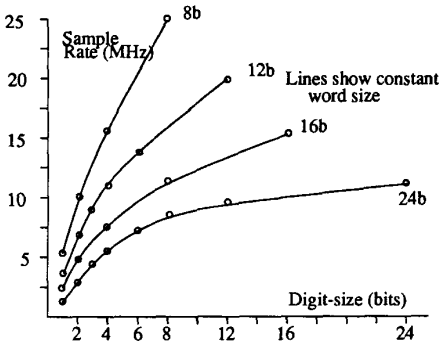

Figure 4. Throughput curves

A standard metric for evaluating the efficiency of an operator is its area time product. Figure 5 shows *throughput per unit area* = $1/AT$ for various different word sizes. Throughput per unit area will be referred to as *throughput efficiency*. As can be seen, the maximum throughput efficiency is achieved when the word is broken into digits of from four to eight bits, depending on the process parameters. Simple mathematical analysis shows that the maximum throughput efficiency is achieved when $N = \sqrt{\frac{a_0 t_0}{at}}$. For higher digit size, the efficiency decreases as a result of slightly higher throughput but much higher area.

Of course, there are faster ways of doing addition than simple ripple carry addition. The analysis, however, is still essentially valid, except that operations may be faster. The same analysis applies to many different operators, notable multiplication. Not all operators are limited in speed by propagation of partial results along their length, such as the ripple carry in the adder. However, the slowest operator in a circuit will limit the clock speed, and this will typically be an operator with propagating partial results.

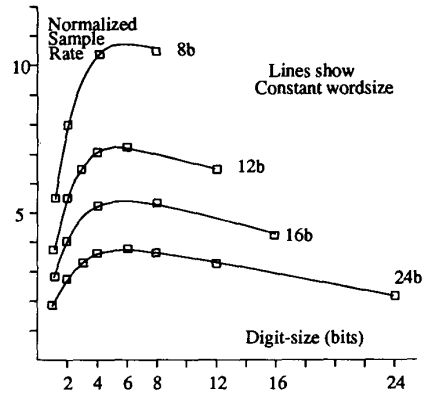

Figure 5. Throughput efficiency

5. The Design Framework - PARSIFAL

In order to provide the capability of rapid feedback of design information dependent on a chosen digit size, a "silicon compiler", referred to here as PARSIFAL has been developed. This compiler allows the user to specify his circuit in an algorithmic input language. By changing the NIBBLESIZE specification in the input file, the designer may generate chips with any given digit-size. Layout information (including final chip size) is given. There are also tools for estimating the achievable clock rate, thereby giving rapid feedback of the throughput of the circuit. (The figures 3 and 4 above may be used to get a preliminary rough estimate.)

The circuits are generated by assembling a standard-cell style layout from a library of pre-designed "leaf cells".

The digit-serial compiler is part of a design environment described in [5] and [8]. The environment provides for behavioural and logic simulation, fault simulation, and design verification software for checking the correctness of the design at various levels. The user describes his chip in a high level algorithmic language which is translated by the compiler into an intermediate structural description from which layout is generated. Examples of chip descriptions are contained in [8].

A most important aspect of the high-level language description is that it relieves the user of the responsibility of worrying about control or synchronization delays (see figure 0). For instance the circuit in figure 0 can be described in two lines as

```
sum := ps(a) + ps(b);
xout := sp (sum < 0.0 ? -sum : sum);
```

The compiler will insert delays in a (guaranteed) optimal manner to minimize the number of bits of delay, taking into account the widths of the signals. It will also provide for control circuitry. Details of the high-level to structural translation process is described in [2].

6. An Example : Four tap FIR filter.

As an example of the trade-off of speed for chip size that can be made by varying the nibble-size, and to illustrate some of the features of the compiler, we consider a simple example of a 16 bit 4-tap FIR filter.

Input and output to the chip is in parallel format with parallel/serial conversion being done on the chip.

The coefficients of the filter are loadable from off the chip through a parallel data bus. For simplicity, the data input bus is used to carry the coefficient data. In addition there is a 2-bit address bus to address the four coefficient registers and a wr (write) signal to indicate that a coefficient should be loaded. Loading a coefficient register is asynchronous and similar to a RAM write operation.

The following chip description shown gives a digit (nibble) size of 1. To make chips of other digit sizes, simply change the value of **nsize**. With a digit-size of n, the chip will accept a new input every $16/n$ clock cycles and give outputs at the same rate. For a detailed explanation of the syntax of this description, the reader is referred to [2] and [8].

```
define(wsize, 16) define(nsize, 1)
include(m4operators.lib)

SYMBOLIC PROCESSOR fir (IN : Xin[wsize], A[2], wr; OUT : Yout[wsize]);

  WORDSIZE wsize; NIBBLESIZE nsize;

#include "operators.h"
FOR latch USE lat CALL "latchgen";
FOR decode USE DECODER;

  SIGNAL
       x[nsize], y[nsize],      /* Serial input and output */
     a0[nsize], a1[nsize], a2[nsize], a3[nsize], /* Coefficients */
       w0, w1, w2, w3;                  /* Select lines for the registers */

  SPECIFICATION
       /* Latch coefficients */
       (w0, w1, w2, w3) := decode (A, wr);
       a0 := ps (latch (Xin, w0));
       a1 := ps (latch (Xin, w1));
       a2 := ps (latch (Xin, w2));
       a3 := ps (latch (Xin, w3));

       /* Change Xin to serial */
     x := ps (Xin);

       /* Calculate the filter output */
       y := a3*x[-3] + a2*x[-2] + a1*x[-1] + a0*x;

       /* Change to parallel again */
       Yout := sp (y);
END;
```

The following table gives the result of compiling the chip using different digit sizes.

digit size	Area (sq. mils)	Time per sample(ns)	Sample rate(MHz)	1/AT
1	16286	22.5 × 16	2.7	170.6
2	16788	25.0 × 8	5.0	297.8
4	19720	30.0 × 4	8.3	422.6
8	26972	40.0 × 2	12.5	463.5
16	42430	60.0 × 1	16.7	392.8
2 × 8	47391	40.0 × 1	25.0	527.5
4 × 4	59843	30.0 × 1	33.3	557.0

Table 1.

The areas are actual areas of compiled chips (including pads). The speed rates are estimates. The last two lines of this table will be explained later.

Note the way that efficiency (1/AT) peaks for an 8 bit digit size and then decreases for the 16 bit digit-size.[1] This is despite the fact that for the 16 bit or fully parallel version of the chip, the parallel-serial converters were not included, being unnecessary. In fact, for high word size, we have met a situation of diminishing returns where chip area increases substantially, but throughput does not.

Note on the other hand, the great improvement of digit-size 2 over bit-serial (digit size 1). The size of the chip is increased by only 3%, but the throughput almost doubles. This is a fair comparison with bit-serial designs. Comparison has shown that the size of bit-serial (digit-size 1) chips compiled using PARSIFAL is almost the same as those produced using BSSC which uses a dedicated bit-serial cell library.

6.1. Increased efficiency using parallel streams

With a digit size of 16, efficiency decreases, as is shown by the table and figure 5. Using digit-serial computation, we can increase the speed of our design by splitting the computation into parallel streams. In order to maintain a throughput of one sample per clock cycle, but maintain a high clock rate, the input stream of 16 parallel bits can be split into two 8-bit wide streams, whereby the even numbered samples are assigned to one stream and the odd numbered ones to the other stream. Two parallel computations are now carried out at the same time and the results merged at output.

[1] Units of efficiency are samples per sq.mil per second.

Since computations are on 8-bit wide data (digit-size 8), a high clock rate may be maintained. We are also located at the peak of the efficiency curve in figure 5. The code section that accomplishes the filter computation is shown next :

```
/* Serialize the inputs */
(s0, s1) := ps(Xin; 0, 1);

/* Calculate the product filter outputs */
y0 := s1[-2]*a3 + s0[-1]*a2 + s1[-1]*a1 + s0*a0;
y1 := s0[-1]*a3 + s1[-1]*a2 + s0*a1 + s1*a0 ;

/* Merge the outputs */
Yout := sp (y0, y1; 0, 1);
```

A precise explanation of this code is omitted. Note only that the split and merge operations on data streams are considered so basic that they are provided as standard options of the sp (serial/parallel) and ps (parallel/serial) functions.

The second last line of table 1 refers to this design. As can be seen, this gives a 25MHz FIR filter. Using four parallel streams 4 bits wide, a throughput of 33.3MHz can be achieved. The last line of table 1 refers to this design and shows that this achieves an even higher efficency. Since designs with more than 4 streams are probably not feasible for reasonable sized chips, the designs represented by the last two lines in table 1 give the best achievable throughput using this method.

We summarize this result in a general rule which may be taken as one of the main conclusions of our paper.

> Greater speed and efficiency can be achieved in DSP designs by using digit-serial rather than word-parallel computation. The highest sample rates can be achieved by splitting the computation into parallel computational streams in order to use a digit-size in the 4-8 bit range.

7. Status

The compiler as described is fully functional. A number of test chips have been designed and fabricated, including the 25MHz FIR filter described here. The chips run correctly at the expected clock speeds.

REFERENCES

[1] Denyer, Peter and David Renshaw, *VLSI Signal Processing: A Bit-Serial Approach*, Addison-Wesley Publishing Company, Inc., Reading Massachusetts, 1985.

[2] Hartley, Richard and Jasica J. *Behavioral to Structural Translation in a Bit-serial Silicon Compiler*. To appear in IEEE Trans. on CAD.

[3] Jackson, Leland B., James F. Kaiser, Henry S. McDonald, *An Approach to the Implementation of Digital Filters*, IEEE Transactions on Audio and Electroacoustics, Vol. AU-16, No. 3, September 1968, pp. 413-421.

[4] Jain, Rajeev, Francky Catthoor, Jan Vanhoof, Bart J.S. De Loore, Gert Goossens, Nelson F. Goncalvez, Luc J. M. Claesen, Johan K. J. Van Ginderdeuren, Joos VanDewalle, Hugo J. De Man, *Custom Design of a VLSI PCM-FDM Transmultiplexor from System Specification to Circuit Layout Using a Computer-Aided Design System*, IEEE Journal of Solid-State Circuits, Volume SC-21, Number 1, February 1986, pp 73-85.

[5] Jasica, Jeffrey R., Richard Hartley, Sharbel Noujaim, and Michael Hartman, *A Bit-Serial Silicon Compiler*, Proceedings of the International Conference on Computer-Aided Design (ICCAD-85), Santa Clara, CA, 1985, pp. 91-93.

[6] S.G. Smith, M.S.McGregor and P.B.Denyer, *Techniques to Increase the Computational Throughput of Bit-Serial Architectures*, Proceedings of ICASSP 87, p 543 (April, 1987).

[7] S.G. Smith and P.B.Denyer, *Radix-4 Modules for High Performance Bit-Serial Computation*, submitted to Proc. IEEE, Part E. October, 1986.

[8] Yassa, Fathy, Jeffrey Jasica, Richard Hartley, and Sharbel Noujaim, *A Silicon Compiler for Digital Signal Processing: Methodology, Implementation and Applications*, IEEE Proceedings Special Issue on Hardware and Software for Digital Signal Processing, Volume 75, No. 9, September 1987.