SYSTOLIC & SEMI-SYSTOLIC DIGIT SERIAL MULTIPLIERS

Poras T. Balsara & Robert M. Owens

Department of Computer Science
Pennsylvania State University
University Park, Pa 16802.

Abstract

Digit serial data transmission can be used to an advantage in the design of special purpose processors where communication issues dominate and where digit pipelining can be used to maintain high data rates. VLSI signal processing is one such problem domain. We propose designs of systolic and semi-systolic digit serial multipliers. These multipliers are programmable i.e. one operand is pre-stored in the multiplier and the other operand is fed in a digit serial fashion. The VLSI implementation of the systolic multiplier is also given. This systolic multiplier is used in our VLSI signal processing system.

Introduction

Data which is transmitted bit or digit serial results in efficient communication both within and between VLSI chips. Bit serial transmission requires only one communication wire per data item, but requires $2p$ cycles to transmit the data across that wire (where $2p$ is the precision in bits). Digit serial transmission requires only $O(\log b)$ communication wires per data item and $2p/(\log b)$ cycles to transmit the data (where b is the base). Limiting transmission between chips is desirable due to pin limitations in VLSI. Limiting transmission within chips is important due to the desire to reduce the chip area used by the interconnect. Although the bit or digit serial transmission slows down the processing speed, the slow down can be overcome by using pipelining to its fullest extent (pipelining at the digit level). As soon as a result digit is generated by a processing element, it is passed along to the next element as an input. This achieves very high degree of concurrency, and a high rate of data flow can be maintained [DeR, KWK].

In this paper we give the designs of digit pipelined multiplier (also referred to as digit online in some of the literature [TrE, Erc, IrO]). First, some of the system conventions are established. Then we discuss the design of the primitive component called *Scaler*, which is used in these multipliers. Finally, we give the design of the above multipliers for *base 4* digit serial multiplication, along with suitable examples. The constructions meet various requirements for a VLSI design to achieve efficient use of silicon; *viz*, simple and regular design with replication of simple processing components, regular and limited interconnect, regular and limited control, and a high degree of concurrency [Kun].

System Conventions

Data Communication Format -- In our system all data communication is digit serial, where a digit is represented in *base 4* signed-digit format [Atk]. Thus the allowable digit sets for the operands are :

Digit Sets :
Conventional digit set = {0, 1, 2, 3}
Maximally Redundant Symmetric Signed-digit set = {-3, -2, -1, 0, 1, 2, 3}
Minimally Redundant Symmetric Signed-digit set = {-2, -1, 0, 1, 2}

Two's complement encoding is used to encode the digits into a binary form.

Data Communication Flow -- The data flow in our system is *right directed* (i.e. from the most significant digit (*msd*) to the least significant digit (*lsd*)). The reason for this choice is explained later.

Numerical Format & Word length -- The data is stored in fixed point, two's complement format. All data has fixed and constant word length of p, *base* 4 digits. Overflow is not permitted in our scheme.

Clocking Scheme -- The standard convention of two phase, non-overlapping clocks [MeC] is used.

Latency -- The first result digit is generated and output some number of clock cycles after the first set of operand digits are input. This latency (or digit online delay [TrE]) is measured as an integer number of digits. In our system the primitive component (the *Scaler*) has a fixed latency of *one*.

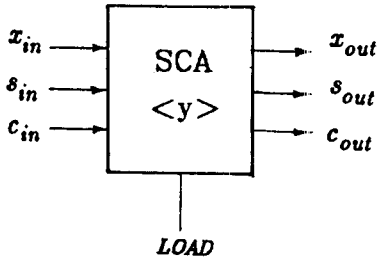
Primitive Component Size -- The *Scaler* component can be classified as *Fine Grain* [Sei]; i.e., the component is limited in size so that many components/processors can fit on a single (or a very few) chip(s).

The *Multiplication* operation, if done *lsd* first, requires latency equal to the precision due to the convention that all data in the system have fixed word lengths. *Lsd* first multiplication of two fixed point, single precision (fractional) values gives a double precision result with the least significant word of the the result produced first, only to be discarded. The most significant word is the one that must be output.

It is not obvious that we can perform multiplication in an *msd* first, right directed fashion, since, with right directed processing it appears that there is no way to wait for a *carry* which starts at the least significant end ! The use of Signed-digit (or redundant) [Atk] arithmetic allows us to achieve right directed processing. The choice of the *base 4* system was governed by the fact that it uses fewer interconnects and has simpler logic, and it is consistent with the other components of our VLSI signal processing system.

Digit Pipelined Scaler

Digit Pipelined Scaler is a right directed primitive component of the Digit Serial Multiplier, which conforms to the conventions given in the previous section. A digit pipelined scaler is shown in Figure 1. This *scaler* scales the input operand X_{in} by the digit value y. This digit value is input on the s_{in} line while the *LOAD* signal is active to load the internal y register (i.e. this scaler is programmable).



Load Cycle *(LOAD = 1)* :

$$s_{out} = s_{in}, \quad y = s_{in}$$

Multiply cycle *(LOAD = 0)* :

$$(i)\ x_{out} = x_{in}$$

$$(ii)\ s_{out} = yx_{in} + s_{in} + c_{in} - bc_{out}$$

where, for base $b = 4$,

$$x_{in}, x_{out} \in \{\bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3\}$$

$$s_{in}, s_{out} \in \{\bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3\}$$

$$y, c_{in}, c_{out} \in \{\bar{2}, \bar{1}, 0, 1, 2\}$$

Figure 1 *MSD Pipelined Scaler.*

During the scaling operation, the *LOAD* signal is inactive and the s_{in} line is set to zero. Since the scale digit $2 \leq y \leq 2$, a slightly modified version of this component can be used for Arithmetic Shift, Clear and Complement operations. The scaler, a primitive component for digit pipelined multiplier, has a latency of one digit.

The SCA component comprises of a full digit product cell (MULT), two digit adder cells (ADD1, ADD2 & ADD3) and some delay cells. Figure 2 shows the block diagram of the SCA component.

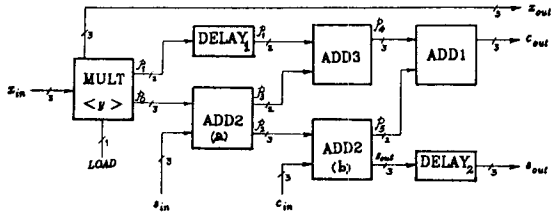


Figure 2 *Block Diagram of the SCA Component*

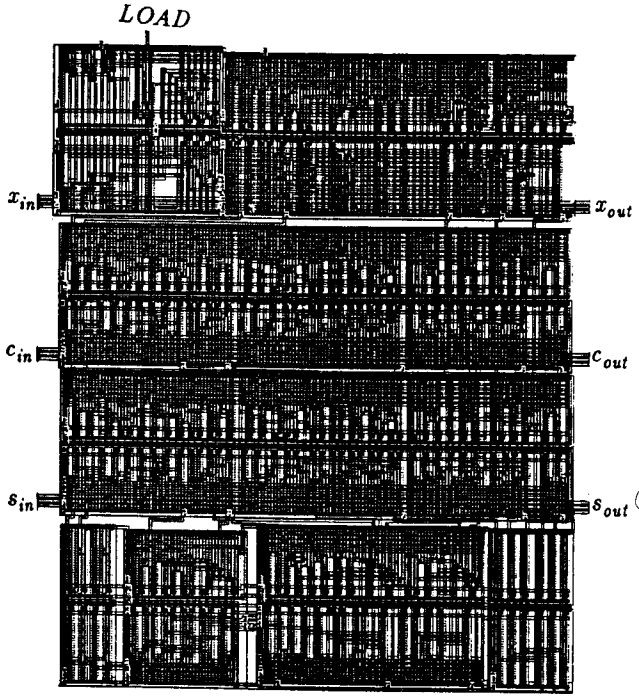


Figure 3 *Layout of the SCA Component*

The SCA component measures 967λ by 862λ in custom logic, double layer metal CMOS (using the *MOSIS* scalable CMOS rules). From Figure 2, one can see that the SCA component is made up of different pipelined stages through which the data is clocked by the system clock. Hence, the clocking speed of the SCA component will depend on maximum time delay through a stage. This delay, in our design is approximately 50 *nsec*. This time delay was obtained by electrical simulation of the circuit using *SPICE*. Table 1 gives the inputs-outputs,

area and time delays of each SCA component. The SCA component layout is done using the Mesh Array technique for CMOS circuit design [Bee].

Table 1. *Details of The Cells in an SCA*

Component Name	Input	Output	Area (λ^2)	Speed (*nsec*)
MULT	3-bit stored(y) 3-bit input(x_{in})	3-bit product($p_{2:0}$) 2-bit carry($p_{4:3}$)	200175	48
ADD1	3-bit & 2-bit inputs($p_{4:0}$)	3-bit sum(c_{out})	86132	35
ADD2	two 3-bit inputs (a) (p_{0},s_{in}) (b) (p_{2},c_{in})	3-bit sum 2-bit carry (a) (p_{2},p_{3}) (b) (s_{out} ,p_{5})	173166	40
ADD3	two 2-bit inputs($p_{1:3}$)	3-bit sum(p_{4})	44969	38
DELAY1	2-bit input(p_{1})	2-bit output(p_{1}) (1 clock delay)	20160	20
DELAY2	3-bit input(s_{out})	3-bit output(s_{out}) (2 clocks delay)	30500	20

$$\overline{3} \leq x_{in} \leq 3 \qquad \overline{3} \leq c_{in}, c_{out} \leq 3$$

$$\overline{3} \leq s_{in}, s_{out} \leq 3 \qquad \overline{3} \leq p_0, p_2 \leq 3$$

$$\overline{2} \leq y \leq 2 \qquad \overline{1} \leq p_1, p_3, p_5 \leq 1$$

$$\overline{2} \leq p_4 \leq 2$$

Digit Pipelined Multiplier

This multiplier is constructed from the SCA primitive component mentioned above. Two types of multipliers could be considered : *(i)* programmable multipliers -- which are preloaded with the multiplier while the multiplicand is supplied to the unit at processing time, and *(ii)* multipliers where neither operand is preloaded but both are supplied at processing time. In this paper we consider only the programmable kind, so that we can design a multiplier which has both low latency and limited intercomponent interconnect. We can either design a pure systolic or a semi-systolic multiplier [Kun].

To multiply two p digit values, $(p+3)$ SCA components are interconnected as shown in Figures 4 and 6 (where $p=4$). The three extra SCA components are necessary to combine appropriate carries and sums.

The three rightmost SCA components are loaded with $y=0$. Thus, the function of these three components is to allow for the carry free formation of s_{out} at the output of the last stage. The SCA components are programmed with the multiplier $Y=(.y_1y_2y_3y_4)_4$, which has been recoded so that $y_i \in \{2,1,0,1,2\}$. The digits of Y are loaded using the s_{in} line while the *LOAD* line is active. The least significant digit of Y is stored in the leftmost SCA component. The digits of the multiplicand are assumed to belong to the maximally redundant ***base*** 4 digit set.

1. Semi-Systolic Digit Pipelined Multiplier :

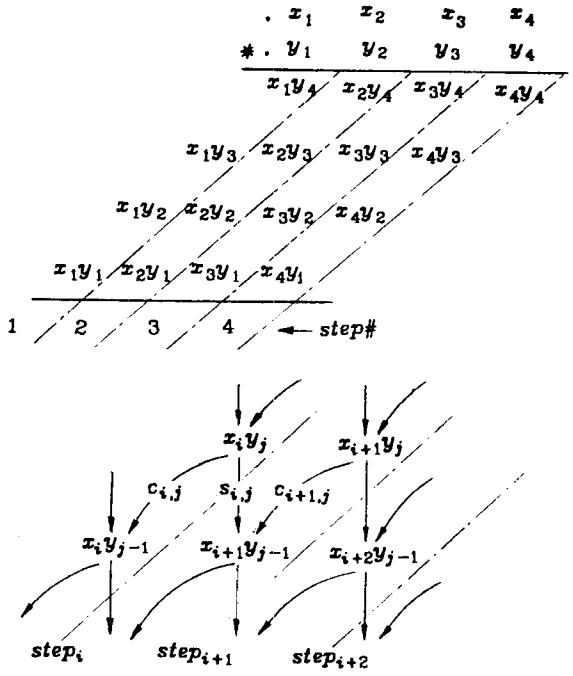


Figure 5 *Flow of Computation in the Semi-Systolic Multiplier.*

The multiplicand digits in the semi-systolic version are broadcast to all the SCA components, while the intermediate data is passed from one component to the next. The msd of the multiplicand X_{in} is broadcast first.

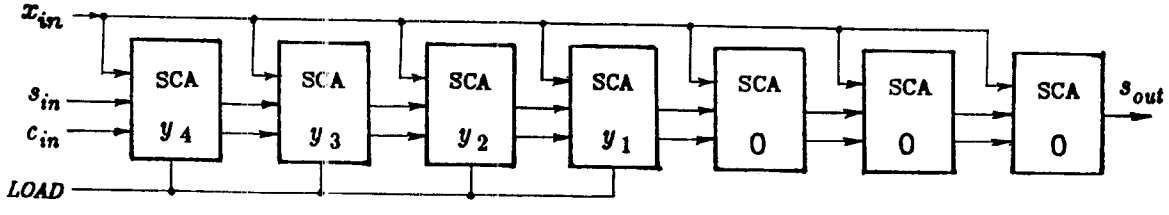


Figure 4 *Block Diagram of the Semi-Systolic Digit Pipelined Multiplier.*

The inputs s_{in} and c_{in} flow through the multiplier in a systolic fashion. Figure 4 shows the block diagram of a semi-systolic digit pipelined multiplier. Figure 5 shows the flow of computation in this multiplier. It also shows how different carries c_{out} and sums s_{out} are combined. One can see that for every broadcast the carries that are generated are combined with the relevant sums in the same step i.e, there is an inherent time delay due to the carry ripple. Because of this there has to be an interval of $O(p)$ between the subsequent broadcasts of the digits of the multiplicand X_{in}.

Table 2. *Semi-Systolic Multiplication Example*

Y		1	1	2	2	0	0	0
$x_{in}=1$	yx_{in}	1	1	2	2	0	0	0
	c_{out}	0	0	0	0	0	0	0
	s_{out}	1	1	2	2	0	0	0
$x_{in}=2$	yx_{in}	2	2	4	4	0	0	0
	c_{out}	0	0	1	1	0	0	0
	s_{out}	2	3	1	3	3	0	0
$x_{in}=2$	yx_{in}	2	2	4	4	0	0	0
	c_{out}	0	1	2	2	1	1	0
	s_{out}	2	0	0	3	0	0	1
$x_{in}=3$	yx_{in}	3	3	6	6	0	0	0
	c_{out}	0	1	1	1	1	0	0
	s_{out}	3	1	3	3	0	1	0
$x_{in}=0$	yx_{in}	0	0	0	0	0	0	0
	c_{out}	0	0	0	0	0	0	0
	s_{out}	0	3	1	3	3	0	1
$x_{in}=0$	yx_{in}	0	0	0	0	0	0	0
	c_{out}	0	0	0	0	0	0	0
	s_{out}	0	0	3	1	3	3	0

An example of msd, base 4, digit pipelined multiplication is given in the Table 2. In this example, $X_{in}=(.1223)_4$ and $Y=(.2211)_4$; therefore, $S_{out}=(.1011(0121))_4$ [$=(.1010(3313))_4$] From the example given in the Table 2, it is clear that the first digit of the result is available after the two multiplicand digits are supplied in, hence the latency of this multiplier is two. This latency is independent of the precision. However, in terms of actual clock delay, this scheme is quite slow because of the $O(p)$ clock delay between the two broadcasts.

2. Systolic Digit Pipelined Multiplier :

In the pure systolic multiplier, the multiplicand digits as well as all intermediate data are passed from one SCA component to the next. The advantage of the pure systolic approach is it maintains nearest neighbor interconnect. The multiplicand X_{in} is fed serially into this multiplier (*msd* first). Figure 6 shows the block diagram of a systolic digit pipelined multiplier. Figure 7 describes the flow of computation in this multiplier. In this multiplier the carries of the previous step are combined with the relevant sums in the next clock step.

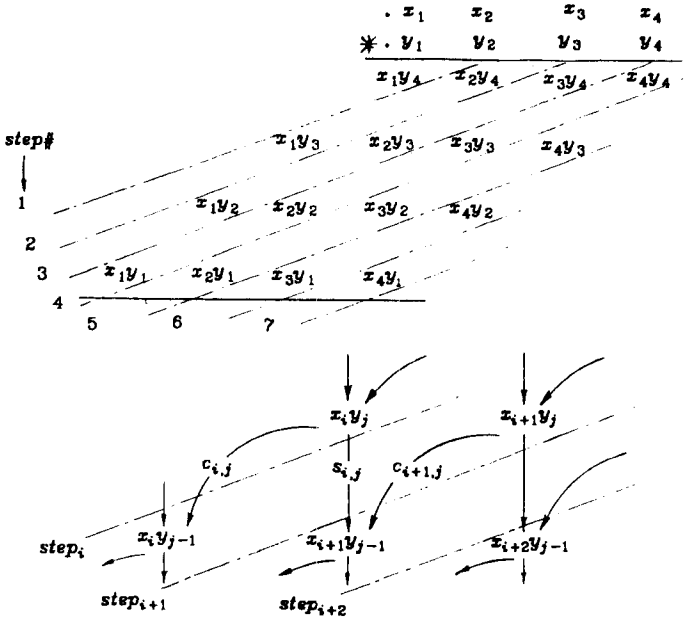


Figure 7 *Flow of Computation in the Systolic Multiplier.*

Thus, one digit of the multiplicand X_{in} is consumed every clock cycle. However, the latency of this multplier depends on the precision, i.e., it has an $O(p)$ digit latency. An example of msd, base 4, digit pipelined multiplication is given in the Table 3. In this example, $X_{in}=(.1222)_4$ and $Y=(.2112)_4$; therefore, $S_{out}=(.0332(0130))_4$ [$=(.1110(3313))_4$]

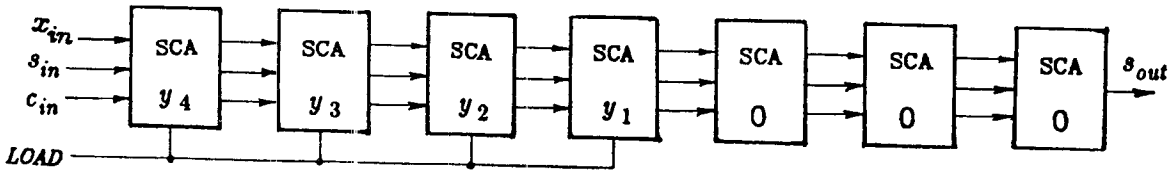


Figure 6 *Block Diagram of the Systolic Digit Pipelined Multiplier.*

Table 3. *Systolic Multiplication Example*

Y		2	1	1	2	0	0	0
$x_{in}=1$	yx_{in}	2	0	0	0	0	0	0
	c_{out}	0	0	0	0	0	0	0
	s_{out}	2	0	0	0	0	0	0
$x_{in}=2$	yx_{in}	4	1	0	0	0	0	0
	c_{out}	1	0	0	0	0	0	0
	s_{out}	0	1	0	0	0	0	0
$x_{in}=2$	yx_{in}	4	2	1	0	0	0	0
	c_{out}	1	1	0	0	0	0	0
	s_{out}	0	1	1	0	0	0	0
$x_{in}=3$	yx_{in}	4	2	2	2	0	0	0
	c_{out}	1	0	1	0	0	0	0
	s_{out}	0	3	0	2	0	0	0
$x_{in}=0$	yx_{in}	0	2	2	4	0	0	0
	c_{out}	0	0	0	1	0	0	0
	s_{out}	0	3	3	2	0	0	0
$x_{in}=0$	yx_{in}	0	0	2	4	0	0	0
	c_{out}	0	0	1	1	0	0	0
	s_{out}	0	0	1	0	3	0	0
$x_{in}=2$	yx_{in}	0	0	0	4	0	0	0
	c_{out}	0	0	0	2	0	0	0
	s_{out}	0	0	3	0	3	0	0
$x_{in}=2$	yx_{in}	0	0	0	0	0	0	0
	c_{out}	0	0	0	0	0	0	0
	s_{out}	0	0	0	1	2	3	0
$x_{in}=3$	yx_{in}	0	0	0	0	0	0	0
	c_{out}	0	0	0	0	0	0	0
	s_{out}	0	0	0	3	0	3	0
$x_{in}=0$	yx_{in}	0	0	0	0	0	0	0
	c_{out}	0	0	0	0	0	0	0
	s_{out}	0	0	0	0	1	2	3
$x_{in}=0$	yx_{in}	0	0	0	0	0	0	0
	c_{out}	0	0	0	0	0	0	0
	s_{out}	0	0	0	0	3	0	3
$x_{in}=0$	yx_{in}	0	0	0	0	0	0	0
	c_{out}	0	0	0	0	0	0	0
	s_{out}	0	0	0	0	0	1	2

Conclusions

We have proposed the designs of systolic and semi-systolic digit pipelined, most significant digit first multipliers. These constructions meet various requirements for a VLSI design to achieve efficient use of silicon, i.e., simple and regular design with replication of primitive components, regular and limited interconnect, regular and limited control, and a high degree of concurrency.

The systolic multiplier is designed in 3 micron, double metal CMOS using the *MOSIS* CMOS design rules. The electrical and logical simulations of the circuit were carried out using *SPICE* and *MOSSIM*. It is currently being used in the *Arithmetic Cube*, a programmable structure to solve linear transformations such as, Convolutions and Discrete Fourier Transforms [OwI].

References

[Atk] Atkins, D., *"An Introduction to the Role of Redundancy in Computer Arithmetic"*, Computer, Vol. 8, No. 6, pp. 74-76, June 1975.

[Bee] Beekman, J., *"Mesh Arrays for CMOS Circuit Design"*, M.S. Thesis, Dept. of Computer Science, Pennsylvania State University, August 1986.

[DeR] Denyer, P. and Renshaw, D., *VLSI Signal Processing : A Bit-Serial Approach*, Addison-Wesley, 1985.

[Erc] Ercegovac, M. D., *"On-Line Arithmetic . An Overview"*, Proceedings of SPIE, Vol. 495, 1984 -- Real Time Signal Processing VII, pp. 86-93, 1984.

[IrO] Irwin, M. J. and Owens, R. M., *"Fully Digit On-Line Newtworks"*, IEEE Transactions on Computers, Vol. C-32, No. 4, pp. 402-406, April 1983.

[Kun] Kung, H. T., *"Let's Design Algorithms for VLSI Systems"*, Proceedings of the First Caltech Conference on VLSI, pp. 65-90, January 1979.

[KWK] Kung, H.T., Whitehouse, H. J. and Kailath, T., Editors, *VLSI and Modern Signal Processing*, Prentice-Hall, 1985.

[MeC] Mead, C. and Conway, L., *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[OwI] Owens, R. and Irwin, M. J., *"The Arithmetic Cube"*, Department of Computer Science Technical Report CS-85-20, Pennsylvania State University, September 1985.

[Sei] Seitz, C., *"Concurrent VLSI Architectures"*, IEEE Transactions on Computers, Vol. C-33, No. 12, December 1984.

[TrE] Trivedi, K. S. and Ercegovac, M. D., *"On-Line Algorithms for Division and Multiplication"*, IEEE Transactions on Computers, Vol. C-26, No. 7, July 1977.