Flexible Block–Multiplier Generation

H.M.A.M. Arts, J.T.J. van Eijndhoven and L. Stok*
Design Automation Section, Eindhoven University of Tech, Eindhoven, The Netherlands
*Computer Science Department, IBM TJ Watson Research Center, Yorktown Heights, NY 10598

Abstract

In a high level synthesis environment there is a strong need for flexible module generators. For the generation of regular structures efficient dedicated module generators can be built. This paper describes the structure of a 'block–multiplier', which features a wide range of area–time trade-offs maintaining efficiency. The structure makes it possible to implement a fully serial or a fully parallel multiplier and many combinations in between. A new concept for the Carry–Hold circuitry plays a key role. The theoretically derived formulas which describe the relations between the area, timing and bitwidth of this multiplier structure are verified by a large number of experiments.

1 Introduction

High level synthesis tools are concerned with the task of transforming an algorithmic behavioral specification of the system to the primitives of a structural representation. These primitives are often called modules. To meet the constraints concerning functionality, area, timing and bitwidth as dictated by the high level synthesis, a large variety of modules has to be available. These modules can be retrieved from a library but the great number of combinations makes it impractical to store all possible combinations. A more promising approach is to use module generators generating modules with the required functionality meeting the constraints as closely as possible.

There are several possibilities to generate modules on the fly. The first is design by hand. However, this will usually take a lot of time and does not fit in the design philosophy of the synthesis systems to shorten the design cycle. A more automated approach is described in [1]. The module generator in the Cathedral environment (MGE) is mainly directed by the designer. The designer specifies both structural and topological information at the same time. Construction rules specify how the real layout of the module must be made starting from the topology. However, this still requires considerable design time.

Several other approaches have used a logic synthesis system [2][3][4] to generate the modules. This works fine for irregular structures, but logic optimization did not yet succeed to take advantage of the regular structure of specific modules. An example of such a module is a multiplier. In the past a large number of multipliers have been developed. Examples are the shift/add multiplier [5][6], the serial–parallel multiplier [7], the carry–save multiplier [8] and other parallel architectures [9]. This paper describes a special structure for a multiplier, which is very suitable for automatic generation. The structure of the 'block–multiplier' allows a large number of implementations with a wide spread in both the area and the time dimension.

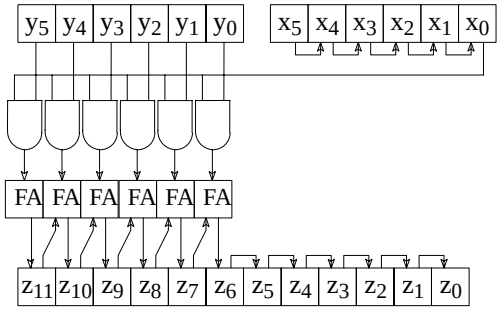

Figure 1: Shift and Add Multiplier

The next section describes some backgrounds in multiplication. Section 3 describes the concepts and the structure of the block–multiplier. Section 4 gives some experiments and results.

2 Conventional Multipliers

The most common multiplier is the Shift and Add Multiplier [5] [6]. Consider two binary unsigned integer words X and Y and their binary representation :

$$X = \sum_{i=0}^{N_x-1} x_i 2^i \qquad Y = \sum_{j=0}^{N_y-1} y_j 2^j \tag{1}$$

The product $Z = X \cdot Y$ can be written as:

$$Z = \sum_{i=0}^{N_x-1} x_i Y 2^i = (...((x_{N_x-1}Y)2 + x_{N_x-2}Y)2 + ...)2 + x_0 Y \tag{2}$$

We may now compute Z by the recurrence:

$$D_0 = 0 \qquad D_{i+1} = D_i 2^{-1} + x_i Y \qquad Z = D_{N_x} 2^{N_x} \tag{3}$$

In each step of the recurrence one bit of X is multiplied (a simple and–operation) with Y and added to the intermediate result D_i which is shifted one bit. Figure 1 shows an implementation of the Shift and Add multiplier for $N_x = N_y = 6$. For a (N_x, N_y)–bits multiplier it takes N_x clock–cycles to complete the multiplication, the delay of the combinatorial circuit (which determines the maximum clock frequency) is approximately: $N_y \delta_{FA}$ (δ_{FA} is the delay of a fulladder, the delay of a register is negligible). So the multiplication time

```
      11 11 11
x     11 11 11
         1001 (0)
        1001  (1)
        101101
       1001   (2)
       10111101
        1001  (3)
       11100001
       1001   (4)
      101110001
      1001    (5)
      1110110001
        1001  (6)
     10001000001
      1001    (7)
     11010000001
     1001     (8)
     111110000001
```

Figure 2: Normal Block multiplication

```
CH     11 11 11
     x 11 11 11
00         1001 (0)
          1001  (1)
00        101101
         1001   (2)
00       10111101
          1001  (3)
01       10100001
         1001   (4)
10      100110001
        1001    (5)
00      1110110001
          1001  (6)
01      1101000001
        1001    (7)
10     10110000001
       1001     (8)
00     111110000001
```

Figure 3: Block Multiplication using Carry Hold

is $N_x T$ ($1/T$ is the clock frequency) with $T \geq N_y \delta_{FA}$. The cost of a Shift and Add Multiplier is: $(3N_y + 2N_x)\gamma_{FA}$ (the cost of a fulladder, γ_{FA}, is assumed to be equal to the cost of a register).

Another implementation leads to the fast but large Carry Save Multiplier [8]. The multiplication of two 4–bit binary numbers can be written as:

$$\begin{array}{cccccccc}
 & & & & X_3 & X_2 & X_1 & X_0 \\
 & & & & Y_3 & Y_2 & Y_1 & Y_0 \\
\hline
 & & & & P_{30} & P_{20} & P_{10} & P_{00} \\
 & & & P_{31} & P_{21} & P_{11} & P_{01} & \\
 & & P_{32} & P_{22} & P_{12} & P_{02} & & \\
 & P_{33} & P_{23} & P_{13} & P_{03} & & & \\
\hline
Z_7 & Z_6 & Z_5 & Z_4 & Z_3 & Z_2 & Z_1 & Z_0
\end{array}$$

where $P_{ij} = X_i \wedge Y_j$. The addition of all P_{ij} terms can be done in an array of fulladders. The delay of this type of multipliers is: $(N_x + N_y - 1)\delta_{FA}$. But since the circuit is combinational the multiplication can be done in one cycle and thus: $T \geq (N_x + N_y - 1)\delta_{FA}$. However, the cost is: $(N_x - 1)N_y\ \gamma_{FA}$ plus $(2N_y + 2N_x)\gamma_{FA}$ if X, Y and Z–registers are accounted as in the shift and add case.

3 The Block Multiplier

A combination of both methods leads to a flexible architecture in which a suitable selection can be made among different implementations with a wide variety in the number of clock cycles, delay and area.

3.1 Basic Circuit

The basic idea is to divide the input words in blocks, and to multiply each block of the first word with each block of the second word in a fast Carry Save Multiplier and add every result, with a proper offset, to the result register. Figure 2 shows an example of the multiplication of two 6–bits words each divided in 3 blocks of 2 bits.

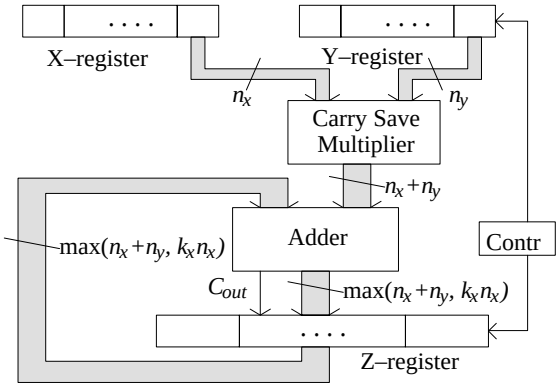

Figure 4: Architecture of the block multiplier

We can write this principle again in a recurrence relation. Consider two binary integer words X (with N_x bits) and Y (with N_y bits), divided in resp. k_x and k_y blocks of resp. n_x and n_y bits (so $N_x = n_x k_x$ and $N_y = n_y k_y$):

$$X = \sum_{i=0}^{k_x-1} X_i 2^{in_x} \qquad Y = \sum_{j=0}^{k_y-1} Y_j 2^{jn_y} \tag{4}$$

The product $Z = X \cdot Y$ can be computed by the relations:

$$D_{i+1} = \begin{cases} 2^{-n_x}(D_i + X_{i\%k_x} Y_{i/k_x}) & \text{if } (i+1)\%k_x \neq \\ 2^{(k_x-1)n_x - n_y}(D_i + X_{i\%k_x} Y_{i/k_x}) & \text{if } (i+1)\%k_x \end{cases}$$

$$Z = 2^{n_y k_y} D_{k_x k_y} \qquad (D_0 = 0) \tag{5}$$

During every step of the recurrence the result from the Carry Save Multiplier is added to the intermediate result D_i and shifted. The result from the Carry Save Multiplier has bitsize $n_x + n_y$, but the result D_i, to which it has to be added, can be larger. We can see in the example of figure 2 that during step 3 and 6 a 4–bit adder would have been too small. Using (5) we calculate the required size of the adder: For every step $i = ak_x - 1$ the following add and shift operation is done:

$$D_{ak_x} = 2^{(k_x-1)n_x - n_y}(D_{ak_x-1} + X_{k_x-1} Y_{a-1}) \tag{6}$$

And in the next step

$$D_{ak_x+1} = 2^{-n_x}(D_{ak_x} + X_0 Y_a) \tag{7}$$

Any product $X_i Y_j$ has $n_x + n_y$ bits, so the integer part of D_{ak_x}, see (6), has:

$$(k_x - 1)n_x - n_y + (n_x + n_y) = N_x \ \ bits. \tag{8}$$

The adder therefore has to add over, see (7), $\max(n_x + n_y, N_x)$ bits in the D_i result. Figure 4 shows the architecture. The word in the X–register has to be shifted every step over n_x bits. The word in the Y–register has to be shifted over n_y bits after every k_x steps. The D_i result in the Z–register, which has to be cyclic, has to be shifted over n_x bits every step, but after every k_x steps it has to be shifted back over $(k_x - 1)n_x - n_y$ bits. When the multiplication is done the result D_i has to be at the proper position in the Z–register. From the last equation of (5) ,we see that the add operation has to be done on the Z–register with an offset of $n_y k_y$ bits.

The area and delay of all used modules are shown in table

1. The controller is just a modulo k_x counter, with its internal reset signal as an output, and therefor needs a logk_x–bits register plus a logk_x–bits incrementor. The delay of the multiplier is only determined by the delay of the adder, because the signals from the CSM appear with the same speed at the inputs of the adder as the carry ripples through the adder.

3.2 Carry Hold Circuit

In most cases ($N_x > n_x + n_y$) the performance of this multiplier is affected by the extra size and delay of the adder. To prevent this a Carry Hold principle is introduced. Figure 3 shows an example of the Carry Hold principle. A 2–bit shift register is used for the Carry–Hold. The carry out of the 4–bit adder is shifted in the Carry Hold Register (CHR) on every step in which it cannot be written to the appropriate bit in the Z–register, because a result may already have been written to that place, i.e. on step (0,) 3 and 6.

A carry–out will be written to the CHR if it cannot be written directly to the result D_i. Every ak_x steps the integer part of D_{ak_x} has N_x bits, on step $ak_x + j$ $(0 \le j < k_x)$ this integer part still has $N_x - jn_x$ bits, while the result of the addition has $n_x + n_y$ bits. A carry–out of this addition may be written directly into the intermediate result if $n_x + n_y \ge N_x - jn_x$ which is true if $j \ge k_x - n_y/n_x - 1$. A carry–out produced on $0 \le j_0 < k_x - n_y/n_x - 1$ has to be shifted in the CHR and shifted out to the carry–input of the adder on $j_0 < j_1 < k_x$, when the lowest bit of the addersum will be written to the same bit in the result D_i as to which the carry–out should have been written, i.e.: $(j_1 - j_0)n_x = n_x + n_y \Rightarrow (j_1 - j_0) = 1 + n_y/n_x$. The CHR has to be a $(1 + n_y/n_x)$–bit shift register (thus n_y should be a multiple of, or equal to, n_x).

Figure 5 shows the architecture of the block multiplier using the Carry Hold principle. A multiplexer for the carry–out and the CHR have been added, and the controller has one output more. Table 2 gives the area and delay of CHR–block multiplier. The controller has become slightly larger. It has to give a "carry to Z–register" signal for every count $j \ge k_x - n_y/n_x - 1$ (on $j = k_x - 1$ there cannot be a carry–out so it does not matter where it is written to), the cost to realize this extra signal will not exceed logk_x γ_{FA}.

The advantage of the Carry Hold principle is an improvement of $(N_x - n_x - n_y)\delta_{FA}$ in the delay and an area decrease of $(N_x - n_y - n_x - n_y/n_x - 1 - \log k_x)\gamma_{FA}$. For example a 32–bits

Table 1: Area and delay of a block–multiplier

module	area (in γ_{FA})	delay (in δ_{FA})	cycles
X–reg	N_x	–	
Y–reg	N_y	–	
CSM	$(n_x - 1)n_y$	$n_x + n_y$	
Adder	$\max(n_x + n_y, N_x)$	$\max(n_x + n_y, N_x)$	
Z–register	$N_x + N_y$	–	
Controller	$2\log k_x$	–	
Total	$2N_x+2N_y+n_xn_y-n_y+\max(n_x+n_y,N_x)+2\log k_x$	$\max(n_x+n_y, N_x)$	k_xk_y

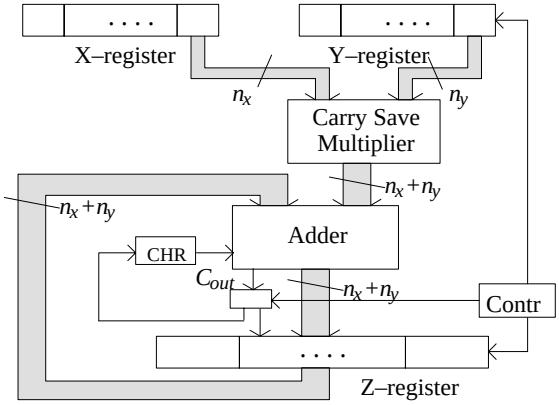

Figure 5: Architecture of the block multiplier with Carry Hold Register

block multiplier with both inputs divided in 4 blocks will have a delay of 32 δ_{FA} if no CHR is used and 16 δ_{FA} if a CHR is used. The area will reduce from 220 γ_{FA} to 208 γ_{FA}.

Figure 6 shows the estimations based on table 1 and table 2 for a (12,12)–bit multiplier divided in all possible blocks. It shows the area (in γ_{FA}) versus the multiplication time ($k_xk_y\delta_{FA}$). For all configurations with $n_x + n_y \ge 12$ or $n_x = n_y = 1$ no CHR is needed so table 1 is used. The multipliers with $n_y = 12$ do not need a controller either, which reduces their area extra by $2\log k_x \gamma_{FA}$. For the multipliers with $n_x = 1$ the result of the CSM has n_y bits instead of $n_x + n_y$, so the adder has one stage less. A compensation for these configurations is used: $-\delta_{FA}$ for the combinational delay ($-12k_y\delta_{FA}$ for the multiplication time) and $-\gamma_{FA}$ for the area. The configuration with $n_x = n_y = 12$ is pure combinational and therefor is represented without registers. No adder is needed either.

4 Results

The Carry Hold–block multiplier is implemented in a module generator program. Figure 7 shows the results for all (12,12)–bit multipliers. The area and delay values are derived by processing the module generator output (network lists using logic gates like nand, nors, invertors and flip–flops) through the MentorGraphics™ software. The solid line connects those configurations which have increasing multiplication time with decreasing area. The dotted line gives the curve for "area x multiplication time = Constant"

Table 2: Area and delay of a CHR–block–multiplier

module	area (in γ_{FA})	delay (in δ_{FA})	cycles
X–reg	N_x	–	
Y–reg	N_y	–	
CSM	$(n_x - 1)n_y$	$n_x + n_y$	
Adder	$n_x + n_y$	$n_x + n_y$	
CHR	$1 + n_y/n_x$	–	
Z–register	$N_x + N_y$	–	
Controller	$3\log k_x$	–	
Total	$2N_x+2N_y+n_xn_y+n_x+n_y/n_x+2+3\log k_x$	n_x+n_y	k_xk_y

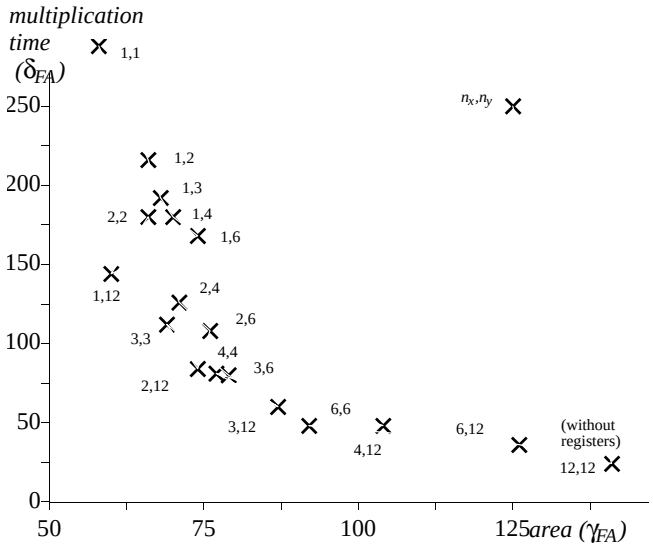

Figure 6: Area/time estimation for a (12,12)–bit multiplier

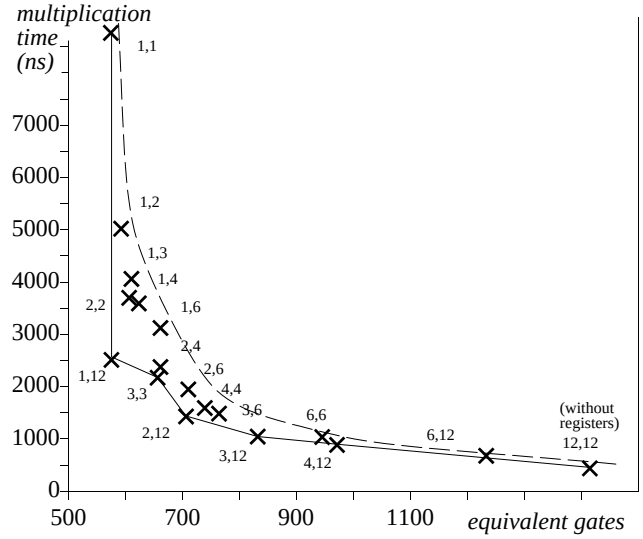

Figure 7: Area/time for a (12,12)–bit multiplier

i.e. A = 500 (1/T + 1). An offset of 500 for the area is used because that is the number of equivalent gates required for the X, Y and Z–registers.

Figure 6 and 7 resemble very well. In a module generator environment, where the area and delay of all gates are known and where the several multiplexers can be taken into account, even better estimations can be made.

Giving a clock cycle *T*, a selection of a multiplier with suitable area and number of cycles can be made using figure 8. It shows the area versus the number of cycles. Each (n_x,n_y) multiplier is connected with a line to a scale which represents the combinational delay. For example, if a clock cycle of 200 ns is used, the (3,6) configuration is the fastest configuration which can be used: it uses 8 clock cycles. But if the clock cycle is 250 ns, the (2,12) configuration can be used, which needs only 6 clock cycles and has less cost.

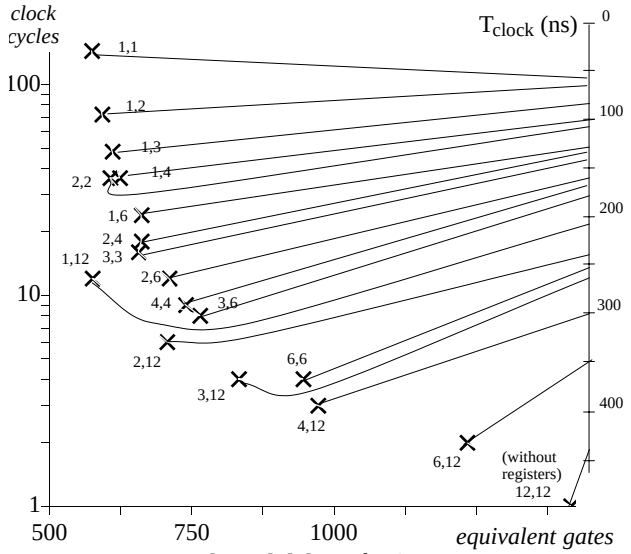

Figure 8: Area/cycle and delay of a (12,12)–bit multiplier

5 Conclusions

A multiplier structure has been described which leads to a very efficient implementation in a module generator environment. The structure provides the flexibility and predictability in both area and delay required to be useful in an automated synthesis environment. It allows configurability from bit–serial to parallel array multiplication. For large examples the block multiplier will be substantially faster then a shift and add implementation, using only slightly more area.

6 References

[1] De Man H., J. Rabaey, and L. Claesen, ”Cathedral–II A silicon-compiler for digital signal processing”, *Computer*, December 1986. pp. 13–25.

[2] Camposano, R. and R.K. Brayton, ”Partitioning before logic synthesis”, *Proc. of the IEEE International Conference on Computer Aided Design 1987*. pp. 324–326.

[3] Brayton, R.K., et al.”MIS: A multiple level logic optimization system”, *IEEE Trans. on comp. aided design*, Vol. 6, 1987.

[4] DeMicheli, G. and D.C. Ku,”HERCULES: a system for high level synthesis”, *Proceedings of the 25th Design Automation Conference*, June 1988. pp. 483–488.

[5] Davio, M. and J.–P. Deschamps, A. Thayse, ”Digital Systems with algorithm implementation”, *John Wiley, New York*, 1983.

[6] Hill, F. and G. Peterson, ”Digital Systems hardware organization and design”, *John Wliley, New York*, 1987. pp. 542–550

[7] Chu, Y, ”Digital Computer Design Fundamentals”, *McGraw–Hill*, New York, 1962

[8] Braun, E. L., ”Digital Computer Design”, *Academic Press*, New York, 1963.

[9] Maden, B. and C.G. Guy, ”Parallel Architectures for High Speed Multiplication”, *IEEE International Symposium on Circuits and Systems,*, 1989. pp.142–145.