RAMAGT – A Tool for Automatic Generation of Radix-2^m Array Multipliers

Jaccottet, Diego P.; Pieper, Leandro Z.; Costa, Eduardo A.; Almeida, Sérgio J.
Universidade Católica de Pelotas
diego_porto_j@yahoo.com , leandrozaf@pop.com.br, {ecosta,smelo}@ucpel.tche.br

Abstract— This paper presents a new Automatic Generation Tool (RAMAGT) that generates Radix-2^m Array Multipliers. With the increase of digital systems complexity, textual descriptions of the circuits became very hard to be made by hand. This problem can be solved by using tools that can automatically generate textual description of digital circuit. As multiplication is a common operation in many digital circuits such as Digital Signal Processors for example, it has motivated us to create a tool that generates multiplier circuit descriptions for VHDL (VHSIC Hardware Description Language) and BLIF (Berkeley Logic Interchange Format) formats automatically. This tool generates n bit wide multiplier circuits operating on group of m bits of the Radix-2^m Array Architecture. To validate the multipliers generated by the RAMAGT tool we have used SIS (A System for Sequential Circuit Synthesis) and SLS (Switch Level Simulator) environments.

I. INTRODUCTION

Multiplier modules are common to many DSP applications and general purpose processors. The state of the art has pointed to multipliers that can operate on 64-bit wide, which is very hard to describe by hand due to the complexity of this type of circuit. As circuit designers do not have any time to spend making the circuit textual description by hand, mainly due to the time-to-market aspect, thus it is very important to develop tools that can automatically generate state of the art of digital circuits. In this work we propose a RAMAGT tool that can generate descriptions of radix-2^m array multiplier circuits automatically.

The multipliers generated by the RAMAGT tool were proposed by [1] and consists of pure array multipliers that can operate on 2's complement and use a radix-2^m encoding to reduce the partial product lines. The RAMAGT tool can generate BLIF (Berkeley Logic Interchange Format) [2] and VHDL descriptions of *n*-bit wide multipliers automatically. Besides, binary and hybrid encoding can be used on group of *m* bits. The hybrid encoding can reduce the glitching activity along the array multiplier and this encoding can be set in the RAMAGT tool when low power multipliers design are taken into account. Another important aspect of the tool is the possibility of enabling different group of *m*-bit operation in the array multipliers easily. This aspect enables the array multiplier to reduce significantly the number of partial product lines. In this work, we have limited the generation of multipliers to group of m=6 and thus, the multipliers can operate on the maximum of radix-64.

While the BLIF description enables the design of the multipliers at the gate level, the VHDL description enables the multipliers to be synthesized onto FPGAs. With this flexibility proposed by the RAMAGT tool, the designer can choose the most adequate design environment depending on the metrics to be considered (area, performance or power). Besides, another important issue is that commercial tools such as Mentor Graphics [3] accept BLIF and VHDL as input descriptions in its flow [4].

II. RELATED WORKS

A lot of work has focused in the development of automatic generators of multiplier circuits. In [5] it is proposed a M-bit by N-bit Booth encoded parallel multiplier generator. In [6] a new low-power constant multiplier with a generator written in C++ is used to generate technology-independent VHDL code of the constant multiplier for different input specifications. In [7] an automatic hardware generator is developed for computing multiplier-based arithmetic functions where verilog codes are generated. In [8] a regularized multiplier generator is proposed creating VHDL descriptions for signed and regular multiplier. One of the most significant publications is presented in [9], where a module generator called Mullet is presented to produce a near-optimal parallel multipliers, creating multipliers architectures that broken down into a partial product generator (PPG) and a partial product summer (PPS). The tool can also be used to explore tradeoffs between architectures. Although the great amount of tools to generate automatic descriptions of multipliers present in literature, none of them has taken into account the flexibility of generating another textual description rather than VHDL or Verilog. Besides, neither of them considers another operand encoding rather than binary. In our tool, radix 2^m hybrid array multipliers are generated automatically as a way of reducing the switching activity along the array, and thus enabling low power design.

III. OVERVIEW OF THE RADIX-2^m ARRAY MULIPLIER

In this section we summarize the methodology of [1] for the generation of regular structures for arithmetic operators using signed radix-2^m representation.

For the operation of a radix-2^m multiplication, the operands are split into groups of *m* bits. Each of these groups can be seen as representing a digit in a radix-2^m. Hence, the radix-2^m multiplier architecture follows the basic multiplication operation of numbers represented in radix-2^m in 2's complement.

This operation is illustrated in Figure 1, for a radix-16 multiplication example. For the *n*-*m* least significant bits of the operands unsigned multiplication can be used. The partial product modules at the left and bottom of the array need to be different to handle the sign of the operands.

For this architecture, three types of modules are needed. Type I are the unsigned modules. Type II modules handle the *m*-bit partial product of an unsigned value with a 2's complement value. Finally, Type III modules that operate on two signed values. Only one Type III module is required for any type of multiplier, whereas $\frac{2n}{m}-2$ Type II modules and $\left(\frac{n}{m}-1\right)^2$ Type I modules are needed.

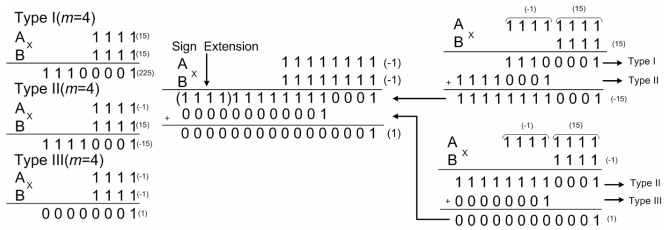

Figure. 1 - Example of a 2's complement 8-bit wide radix-16 multiplication

IV. RADIX-2^m ARRAY MULTIPLIER GENERATION

The RAMAGT tool can generate textual descriptions to *n* bit wide multiplier circuits. The wide of the operands is selected by the user as will be after presented in Section VI. Another variable, named *m*, is also available for the user and it represents the number of group of bits that is multiplied at a time. The range of *m* values that we are presenting in this work can varies from *m*=2 (radix-4) until *m*=6 (radix-64). Although we have presented results until *m*=6, the tool can naturally generate higher radices multipliers. In the array multiplier architecture, the operands are split into group of *m* bits. Thus, the number of operands (*x*) depends on the values of *n* and *m* and its calculation is given according to the Equation 1.

$$x = \frac{n}{m} \tag{1}$$

As should be observed, each operand represents an operation of *m* x *n*. Thus, if the value of *x* is an integer number, the structure of the array multiplier will be composed by *x* operands. As should be also observed, the Equation 1 can also be used in order to calculate the number of dedicated multiplication blocks into each operand. As an example, if a 16-bit multiplier is generated to operates on radix-16 (*m*=4), thus the number of operands is equal to 4. Thus, the general architecture will be composed by 16 *m*=4 dedicated multiplication blocks.

When the value of *x* in Equation (1) does not represents an integer number, the RAMAGT tool calculates a temporary value for *n* named *n_temp* that is presented showed in Equation (2). This temporary value is needed to calculate the number of asymmetrical of dedicated multiplication blocks. As should be observed in Equation (2), the term *int* represents the integer part of *x* calculated by Equation 1.

$$n\_temp = m \times \left(\text{int}\left(x\right)+1\right) \tag{2}$$

This strategy is needed to calculate the size of the asymmetrical block (*fb*). This is made according to the Equation (3).

$$fb = m - \left(n\_temp - n\right) \tag{3}$$

The number of *m* x *fb* blocks is defined as *fc* and calculated according to Equation (4). The term int(*x*) is multiplied by 2 in (4), because the two operands will be multiplied by the *fb* terms.

$$fc = 2 \times \text{int}(x) \tag{4}$$

As an example, if we need to generate a *n*=16 and *m*=3 multiplier architecture the values of *n_temp*= 18 and *fb*=1, according to the Equation (2) and (3). So, the description of the multiplier will need ten 3x1 (*fc*) and one 1x1 (*fb* x *fb*) blocks in this architecture. As should be observed, independent of the size of the operands, we will ever need only one *fb* x *fb* block, because this represents the multiplication of the last most significant bits. Those blocks are previously designed and they are available in the library of the RAMAGT tool. In Figure 2 it is shown an example of the structure of an array multiplier with *m*=3 and *n*=4 bit wide.

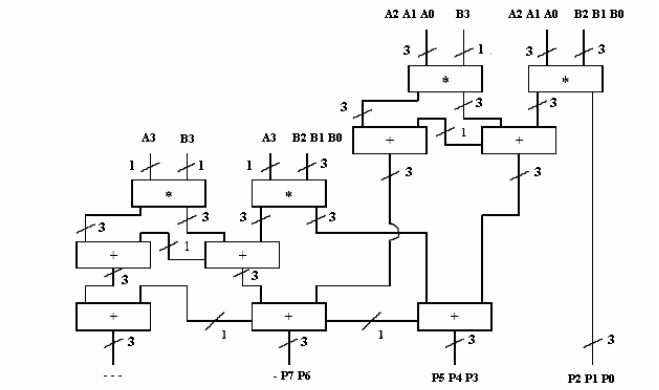

Figure. 2 - An array multiplier for 4-bit wide and *m*=3

As can be observed in Figure 2 the structure of an array multiplier for 4-bit wide and *m*=3 need one block 3x3 (calculated by int(*x*)), two blocks (calculated by *fc*) 3x1 (*m* x *fb*) and one block 1x1 (*fb* x *fb*).

V. METHODOLOGY

In this section we will briefly show how the RAMAGT tool was developed. The tool was implemented by using two programming languages. One of them is C++ that generates executable files in a faster way. The other is the CSL (C Scripting Language) [11] that is a powerful and easy programming language that follows the C syntax very closely and it is used like an interpreter. A language that uses scripts has the advantage that the code does not need to be compiled [12], so that the projects can be easily generated. The C++ language is used to link the interface of the projects

requirements with the user. It allows that many different projects can be done at the same time.

In Figure 3 we show the methodology used to generate a multiplier description. The main program executes the secondary one with the language format as parameter (BLIF or VHDL), allowing multiple secondary programs with different parameters and languages to be opened at the same time. The secondary program lists the tree of projects for the given language and it executes the selected project from the tree with the given parameters set. It will execute according to the configuration of the language (a configuration file on the language folder includes the way it will execute, its extension and the description for the language). It will also send the locations for the project to execute appropriately. The project folders contain the files to be executed and any extra files (the main CSL executable and its companions may require to execute appropriately). It also contains the notes on the project and a file describing the parameters and initial values for them.

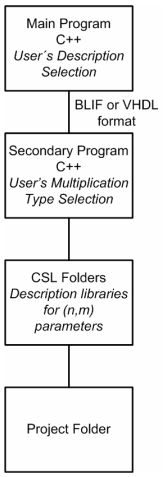

Figure 3. Methodology used to create the RAMAGT tool

VI. THE RAMAGT TOOL

In this section we present the steps in the RAMAGT tool to generate multiplier descriptions of radix-2^m array architectures. The main screen of the RAMAGT tool is presented in fig. 4. In this screen we can select the format of the multiplier circuit description (BLIF or VHDL). It should be observed that a good characteristic of this tool is that we can make many projects at the same time.

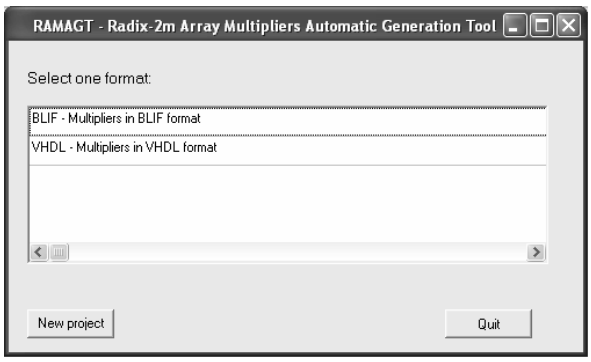

Figure 4. RAMAGT main screen

Once we have selected the textual description format for the multipliers, another window is opened and then we will able to select the number of groups of bits (*m*) and the length of the operands, as it is shown in Figure 5. After that we just click on "Generate" button, and it will be created a folder in the current path called "Projects" where will be contained the files with the textual description of the multiplier. To validate the descriptions created by the tool we introduce the VHDL files in Quartus II from Altera or the BLIF format in SIS environment.

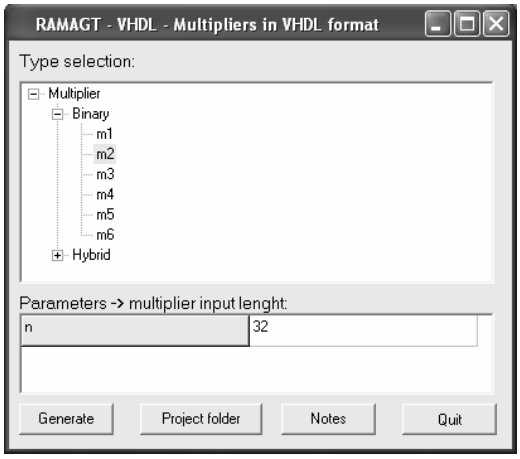

Figure 5. Selection of the type of group of *m* bits

Examples of VHDL and BLIF descriptions from 4-bit, *m*=2 array multiplier generated by RAMAGT tool are presented in Figures 6 and 7. For simplicity of the Figures we have not shown the operand1 and operand2 that are subcircuits of the array architecture.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY Multiplicador_Array_ComSinal_ComExtensao_binario_m2_n4 IS
       PORT (a, b: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
             s: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
            );
END Multiplicador_Array_ComSinal_ComExtensao_binario_m2_n4;
ARCHITECTURE comportamento OF Multiplicador_Array_ComSinal_ComEx
SIGNAL so0e1,so0e0: STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL sole2,sole1,sole0: STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL cout0e : STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL temp0: STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
       estagio0: operandol PORT MAP (  a(1 DOWNTO 0), b(3 DOWNTO
DOWNTO 0));
       estagio1: operando2 PORT MAP ( a(3 DOWNTO 2), b(3 DOWNTO
sole0 );
       estagio2: halfadder_2Bits PORT MAP ( so0e0, sole0, s(3 DO
       estagio3: fulladder_2Bits PORT MAP ( sole1, so0e1, cout0e
       temp0(0) <= so0e1(1);
       temp0(1) <= so0e1(1);
       estagio4: fulladder_2Bits PORT MAP ( sole2, temp0, cout0e
END comportamento;
```

Figure 6 - VHDL format example

```
.model multiplicador_array_c/sinal_4Bits_2agrupamento
.inputs A3 A2 A1 A0 B3 B2 B1 B0
.outputs S7 S6 S5 S4 S3 S2 S1 S0
.subckt operandol A1=A1 A0=A0 B3=B3 B2=B2 B1=B1 B0=B0 S5=S0E3
.subckt operando2 A1=A3 A0=A2 B3=B3 B2=B2 B1=B1 B0=B0 S5=S1E5
S0=S1E0
.subckt halfadder_2Bits A1=S0E1 A0=S0E0 B1=S1E1 B0=S1E0 COUT=
.subckt fulladder_2Bits CIN=COUT0E0 A1=S0E3 A0=S0E2 B1=S1E3 B
.subckt fulladder_n++_2Bits CIN=COUT0E1 A1=S0E3 A0=S0E3 B1=S1
.end
```

Figure 7 - BLIF format example

VII. PERFORMANCE RESULTS

In this section will be presented results for *n*=16 and 32-bit multiplier architectures on radix 4,8,16, 32 and 64 of the radix-2^m array multipliers that were generated by RAMAGT tool. The results are presented by considering BLIF descriptions format. Area, delay and power consumption results are presented for the 16 and 32-bit multiplier architectures. Area and delay were obtained by using SIS tool and power results were obtained with SLS tool, after converting BLIF to SLS format. This is made by a blif2sls converter. Area results are presented in terms of number of literals. Delay results were obtained using the worst delay propagation between the input and output signals. Power results were obtained by using the average power value of the SLS tool [8], by using the general delay model. For the power simulation we have applied a random pattern signal with 10000 input vectors.

Table 2. Performance results for 16-bit radix-2^m array multipliers

Parameters	*m*=2	*m*=3	*m*=4	*m*=5	*m*=6
Area (Literals)	4602	8382	14983	33452	61758
Delay (ns)	227,80	218,50	206,20	234,50	322,70
Power (W)	0,2414	0,2315	0,2292	0,3748	0,5144

Table 3. Performance results for 32-bit radix 2^m array multipliers

Parameters	*m*=2	*m*=3	*m*=4	*m*=5	*m*=6
Area (Literals)	18586	31849	61935	131061	325885
Delay (ns)	478,20	448,50	431,40	458,90	536,10
Power (W)	2,5231	1,6609	1,4635	1,8387	2,9470

As can be observed in Table 2 and Table 3, the 16 and 32-bit multipliers present higher area values according to the growth of the radix operation. This occurs because as the higher radix operation of the multipliers, higher is the complexity presented by the dedicated multiplication blocks. While the radix 4 dedicated multiplication block is implemented by using only 8 logic gates, for the radix-64 more than 5.000 logic gates are needed to implement the dedicated multiplication block. This justifies the large difference of area presented by the 16 and 32-bit multipliers according to the *m* value used.

In terms of delay it should be observed that the multipliers on radix-16 are that present the best result among the architectures. In fact, from radix-4 to radix-16 the critical path decrease because there is a reduction of partial product lines and the dedicated modules are more simple to be implemented. Although the radix-32 and radix-64 multiplication blocks enable the reduction of partial product lines of the multipliers, these blocks are more complex to be implemented and thus, the higher complexity of these modules do not enable reduction in the critical path and consequently in the delay value. As can be observed in Table 2 and Table 3, the radix-16 multiplier is that present the less power consumption compared against the other architectures. This indicates that the radix-16 represents a near optimum point between the studied multipliers, where these circuits can operate with higher performance and power reduction. In terms of power reduction we have observed that the reduction of glitching activity is the main aspect of power reduction in the 16 and 32-bit radix-16 array multipliers.

VIII. CONCLUSIONS AND FUTURE WORK

We have presented a new RAMAGT tool for the automatic generation of radix-2^m array multiplier. The tool generates automatically BLIF or VHDL descriptions of *n* bit and *m* group of bits of array multipliers. The circuit descriptions enable that the multipliers can be synthesized to FPGAs (VHDL description) or at gate level by using SIS environment (BLIF description). We presented area, delay and power results from 16 and 32-bit array multipliers operating on different radices. These multipliers were generated by RAMAGT in BLIF description format and the results showed that the radix-16 multipliers presented the higher performance and the less power consumption. As future work we intend to present results of the multipliers with Hybrid encoding. We also intend to introduce in the RAMAGT new architectures of multipliers that use the low power techniques proposed by [13].

REFERENCES

[1] COSTA, E.; MONTEIRO, J.; BAMPI, S. "A New Architecture for Signed Radix 2^m Pure Array Multipliers" . In *IEEE International Conference on Computer Design*, pages 112-117, 2002.

[2] BERKELEY, U. BLIF – "Berkeley Logic Interchange Format". University of California. At http://embedded.eecs.berkeley.edu/ . 2007.

[3] GRAPHICS, M.; "QuickPath User's and Ref. Manual." *Mentor Graphics.* At http://www.mentor.com . 2008.

[4] OLIVEIRA, L. L.; "Prototipação e Análise de Circuitos Multiplicadores Array de Baixo Consumo". Dissertação de Mestrado. PPGEE. UFSM, RS. 2005.

[5] FADAVI-ARDEKANI, J. "M×N Booth encoded multiplier generator using optimized Wallace trees". *Very Large Scale Integration (VLSI) Systems*, IEEE Trans. on Volume 1, Issue 2, June 1993 Page(s):120 – 125.

[6] CHENG-YU, P.; AL-KHALILI, A.J.; LYNCH, W.E; "Low-Power Constant-coefficient multiplier generator" . ASIC/SOC Conference, 2001. In *14th Annual IEEE Int. 12-15* Sept. 2001 Page(s):185 – 189.

[7] TSO-BING, J.; JENG-HSIN, J.; MING-YU, T.; SHEN-FU H.. "A high performance function generator for multiplier-based arithmetic operations". ASIC, 2002. Proceedings. 2002 IEEE Asia-Pacific Conference on 6-8 Aug. 2002 Page(s):331 – 334

[8] QIAN, Yu.; WANG, Dong-Hui;. "A design of regularized multiplier generator". ASIC, 2003. Proceedings. In *5th International Conference* on Volume 2, 21-24 Oct. 2003 Page(s):1269 - 1272 Vol.2

[9] TSOI, K. H.; LEONG, P.H. GENDEREN, "A. Mullet - a parallel multiplier generator". In *Field Programmable Logic and Applications*, 2005. Int. Conference on 24-26 Aug. 2005 Page(s):691 - 694.

[10] COSTA, E.; BAMPI, S.; MONTEIRO, J.; "A New Pipelined Array Architecture for Signed Multiplication". In *Symposium on Integrated Circuits and Systems Design*. 2003

[11] KOCH, P. "C Scripting Language –Reference Manual V. 4.4.0". At http://csl.sourceforge.net/csldoc/index.htm Last Revision: 2002

[12] OUSTERHOUT, J. K. "Scripting: Higher Level Programming for the 21st Century" . IEEE Computer Magazine. March 1998

[13] PIEPER, L.; COSTA, E.; ALMEIDA, S.; BAMPI, S.; MONTEIRO, J. Efficient Dedicated Structures for Radix-16 Multiplication. In XIV Iberchip – IWS, 2008