FPGA Implementation of Residue Number System Structures

R.Venkatesan, D.Phoukas, G.A. Jullien
VLSI Research Group
University of Windsor
Windsor, Ontario, Canada N9B 3P4
(519) 253-4232 Ext. 2574
jullien@engn.uwindsor.ca

Abstract

In this paper we explore the feasibility of implementing Residue Number System (RNS) based DSP algorithms on Look-up table based Field Programmable Gate Arrays (FPGAs). In this process, Binary Decision Diagrams are used for logic representation, and a new minimizing algorithm for incompletely specified functions is introduced. A sample residue arithmetic based design is presented along with promising simulation results. The architectural model is based on a previously reported bit-level systolic array cell.

INTRODUCTION

FPGAs offer the designer the benefits of custom hardware, eliminating high development costs and manufacturing time. Proper design techniques and effective logic optimization are the key to efficient FPGA implementation.

The RNS has often been proposed for custom hardware implementation of high throughput DSP algorithms. The lack of a carry propagation is often cited as the main advantage of the number system, but it is probably the structural decomposition of regular DSP arrays that is the more attractive property. The ability to use 'algebraic tricks', such as those associated with index calculus and quadratic rings, provide an extra stimulus to the use of RNS [1], and there is evidence of the industrial use of this number system [2] to exploit such properties.

Since computations over finite rings are, in general, difficult to logically decompose, many references suggest the use of tables to store pre-computed results. Reference [3] presents a general methodology to realize residue arithmetic operations using small look-up tables, where the look-up tables store unminimzed truth tables. Since Xilinx FPGAs [6] use look-up tables as configurable logic blocks, they may be considered an ideal choice for RNS based designs. It is of interest to explore such a marriage of programmable hardware with designs based on arrays of look-up tables. There is some evidence that the memory based CLB structure has attracted interest for the implementation of traditional ROM based arithmetic architectures [7].

In this paper, we elect to represent and manipulate the logic functions, specified by the look-up tables, using Binary Decision Diagrams (BDDs). Due to a mismatch between the ring order and the power of two modulus ring in which the computation is embedded, the logic functions can have large numbers of *don't care* states. As an example, in order to represent a mod-17 computation, 5 bits are required, so out of 2^5=32 cases, 15 are don't cares. If these 15 don't cares are assigned appropriate values, then the overall logic function can be greatly minimized. In order to exploit this property we present an algorithm for minimizing the BDD size of incompletely specified functions and target the algorithm to an established bit-level systolic array architecture.

DON'T CARE ELIMINATION ALGORITHM

Since BDDs [4] are efficient, concise representation of logic functions, they are used to represent ROM contents of RNS table arrays. Minimizing the size of a BDD is a crucial factor to efficient FPGA resource utilization. This section presents an algorithm for minimizing the BDD size of functions with don't cares.

The algorithm adopts a divide and conquer strategy, and it is presented here as a preprocessing step prior to the application of Bryant's reduction rules [4]. Given a decision tree the basic idea is to eliminate as many redundant nodes as possible. This is done in a *greedy* manner, i.e. starting from the root of the BDD we minimize the number of nodes at each level by assigning constant values to as few don't cares as necessary and propagating constraints about the rest of them downwards. The objective is to make the two subgraphs

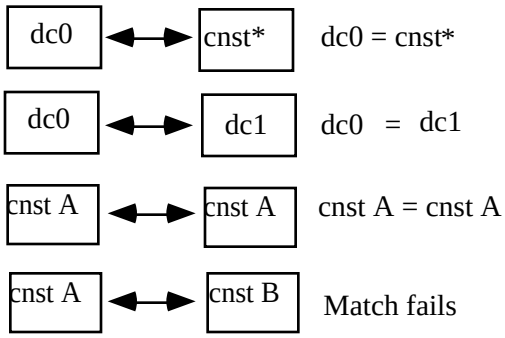
equal as close to the root as possible.

In our algorithm, each incompletely specified output (don't care value) is regarded as a distinct variable and the completely specified outputs are regarded as constants. First the decision tree is examined at each node starting from the root, and the two children of the node, the high child and low child are compared with each other until all the don't care variables are assigned constant values. For an *n*-variable function there can be a maximum of $2^n - 1$ comparisons. The operation of comparing two subtrees and assigning appropriate values to don't cares is performed by the function *match.*

Match

Match is a recursive function which takes two trees and returns, as a result, the assignments of constants to the don't care variables. Matching two sub-trees fails if any of the corresponding sub-trees of them are not matchable, otherwise it succeeds by returning the possible bindings of don't cares. In case of leaf nodes, a match can be defined by the four cases shown in Figure 1 below (boxes represent leaf cells).

Figure 1 Four match cases

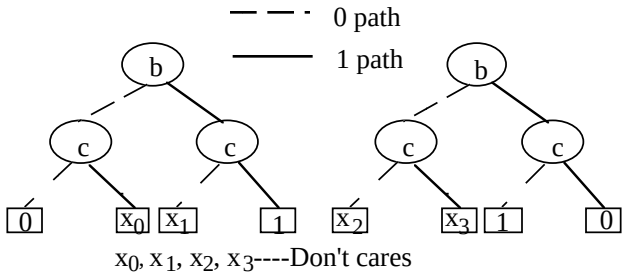


As an example, in Figure 2, for a match between sub-trees 1 and 2, *match*(1,2), we observe that although the first three leaf cells are matchable with the result $x_2 = 0, x_0 = x_3, x_1 = 1$ the final leaf cells cannot be matched, therefore no bindings of don't cares can be made, i.e. the match fails.

The main features of the algorithm are:

1. It eliminates as many redundant nodes as possible.
2. It never considers subtrees of different parents for a match.
3. It will perform a maximum of $2^n - 1$ matches for an *n* variable tree.
4. It is deterministic, i.e for a given function it gives the same result every time.

Figure 2 Match fail example



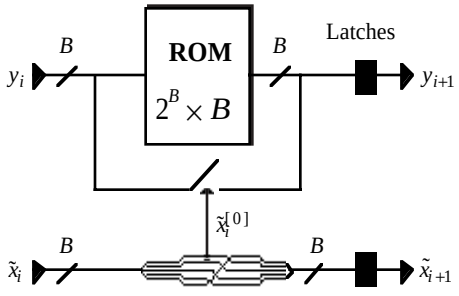
THE RESIDUE PROCESSING CELL

Although most DSP applications are computationally intensive, often those computations are simple cascades of additions and multiplications in a well defined structure. That operation of repeated multiplication and addition is performed by a generic processing cell, called the Inner Product Step Processor (IPSP). Over a finite ring this becomes an $IPSP_m$. For better throughput, the $IPSP_m$ can be sliced to the bit level; this is referred to as a $BIPSP_m$, [5] defined by eqn. (1).

$$y_{i+1} = y_i \oplus_m [a \otimes_m x^{[i]} \otimes_m 2^i] \tag{1}$$

where i is the spatial array index, $y_{i+1}, y_i, a \in R(m)$, and $x^{[i]}$ is the ith bit of $x \in R(m)$ [5]. The cell is shown, for completeness, in Figure 3.

Figure 3. The $BIPSP_m$ cell



The inputs to the cell are y_i and x_i, the outputs are y_{i+1} and x_{i+1}, each output line is latched. The cell computes the relationship:

$$y_{i+1} = \begin{cases} y_i \oplus_m [2^i \otimes_m a] & \text{for} \quad \tilde{x}_i^{[0]} = 1 \\ y_i & \text{for} \quad \tilde{x}_i^{[0]} = 0 \end{cases} \tag{2}$$

B such cells are connected in cascade to implement a $IPSP_m$ cell (*B* is the number of bits required to represent a ring ele-

ment). The x bits are circularly shifted by one position for each cell to present the correct steering bit to the ROM.

XILINX IMPLEMENTATION

We now consider the use of Xilinx FPGAs to implement arrays of $BIPSP_m$ cells.

The Xilinx 4000 series of FPGAs [6] consist of a two dimensional array of look-up table based Configurable Logic Blocks (CLBs) with routing resources to provide interconnections between them. Each CLB also has 2 flip-flops to aid in the design of sequential logic.

The $BIPSP_m$ cell

Consider the implementation of the $BIPSP_m$ systolic building block. Each CLB in the 4000 series can be configured as a single 32x1 ROM. So a single $BIPSP_m$ cell for a 5 bit wordlength is expected to occupy 7.5 CLBs, 5 CLBs for the 32x5 ROM (one CLB for each output bit of the ROM) and 2.5 CLBs for the steering operation. A 5 bit $IPSP_m$ cell requires 37.5 CLBs for implementation, with roughly 2.5 $IPSP_m$s being implemented in a 100 CLB XC4003 device.

We can use the BDD minimization procedure to reduce the CLB count. First the ROM contents are represented in BDDs and the don't cares in the function are optimized using our algorithm. We then apply Bryant's reduction rules [5] to obtain a Reduced Ordered BDD. Finally the ROBDD is mapped to Xilinx CLBs.

An example: Mod-17 Encoder

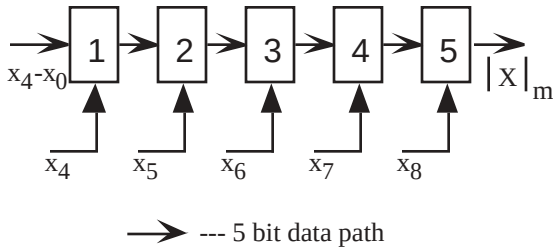
In this section we will illustrate the use of our BDD minimization algorithm, by constructing an encoder from $BIPSP_m$ cells. We have chosen a Mod 17 ring to illustrate the savings possible when there are many don't care states (17 is also a Fermat prime with some rather useful properties). An element of a mod-17 ring has to be represented in 5 bits, where there are 17 completely specified output states and 15 don't care states. The general 32x5 ROM, contained in the $BIPSP_m$, cell, will simply store all the 32 possible states, with a **47%** redundancy.

The encoder maps a binary weighted number into a finite ring. An *S*-bit binary number, *x*, can be mapped to to a modulo *m* ring as shown in eqn. (3).

$$|X|_m = \sum_{b=0}^{S-1} {}_m 2^b \otimes_m x_b \qquad (3)$$

The systolic structure in Figure 4 [5], where a 9-bit encoder is realized using a single $IPSP_m$ cell, is considered for implementation.

Figure 4. 9-bit Encoder using an array of $BIPSP_m$ cells.



In Figure 4, x_0-x_8 represents the nine bit binary number to be encoded, x_4, x_5, x_6, x_7, x_8 are the steering bits for the respective 5 $BIPSP_m$ cells. The least significant 4 bits, x_0-x_4 of the binary number are given as inputs to the first $BIPSP_m$ cell, the fifth input of the first cell is tied to zero. The data path throughout the design is 5 bits wide.

All the 5 ROMs in the $IPSP_m$ cell are minimized and the mod-17 encoder is realized in a one dimensional, unidirectional systolic array. For comparison purposes, the unminimized generic ROM based mod-17 encoder is also implemented in the same architecture. The results are tabulated in Table 1. The results given are for the part XC4003PC84, speed grade 6.

Table 1: Mod-17 Encoder Results.

Design Method	No of Occupied CLBs	Longest path between any two flipflops	Operating Speed
Generic ROM based	38	27.3ns	30 MHz
Minimized ROM based	22	27.2ns	30 MHz

As can be seen from the table, the number of CLBs is reduced from 38 to 22, providing a **42%** reduction in area. This is very close to the actual redundancy in the problem; certainly within the granularity of the CLB building blocks. This area reduction is achieved by minimizing the size of the BDD representing the logic function of the ROM. In the minimization process, if the height of the BDD is reduced from 5 variables to 4 or less, each output bit of the ROM (32x1 locations) does not require a complete CLB. Since a CLB in the Xilinx 4000 series can be configured as a single

32x1 ROM or as two 16X1 ROMs, if the tree height is reduced to less than 5, two output bits of the ROM can be accommodated in a single CLB. When compared with the case of the generic ROM, where only one output bit of the ROM can be obtained from a single CLB, a 50% reduction is immediately achieved.

The operating speed is obtained from a Verilog-XL^{TM} simulation. The simulation is performed after extracting timing information for routing delays, from the fully routed XC4003 chip.

CONCLUSIONS

A full custom implementation of RNS building blocks has been retargeted to a look-up table based FPGA architecture. We have discussed an effective logic minimization technique based on ordered binary decision diagrams, that allows us to recover the inherent redundancy in representing finite rings operations using binary variables. Although, in this work, we are unable to take advantage of the special binary arithmetic features of modern FPGAs, such as Fast Carry Logic, the initial results with this approach show some promise.

REFERENCES

[1] G.A. Jullien, "Number Theoretic Techniques in Digital Signal Processing", *Advances in Electronics & Electron Physics*, Vol. 80. 1991.

[2] Barraclough, S.R., Sotheran, M., Burgin, K., Wise, A.P., Vadher, A., Robbins, W.P. and Forsythe, R.M., "The Design and Implementation of the IMS A110 Image and Signal Processor", *IEEE Custom Integrated Circuits Conf.*, pp. 24.5.1-24.5.4, 1989.

[3] M.A. Bayoumi, G.A. Jullien, W.C. Miller, "A Look-up Table VLSI Design Methodology for RNS Structures Used in DSP Applications, "*IEEE Trans. on Circuits and Systems*, CAS-34 No. 6, June 1987.

[4] R.E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computers*, vol C-35,N0.8,pp. 677-691, August 1986.

[5] M. Taheri, G. A. Jullien and W. C. Miller, "High Speed Signal Processing Using Systolic Arrays Over Finite Rings", *IEEE Trans. Selected Areas in Comm*., 6,3, 1988

[6] XC4000, Xilinx Data book, August 1992.

[7] Les Mintzer, "FIR Filters with Field-Programmable Gate Arrays", *Journal of VLSI Signal Processing*, Vol. 6, No. 6, 119-127, August 1993.