Dynamic Circuit Specialization of a CORDIC Processor

Eric Keller

Xilinx Inc.
2300 55th Street
Boulder, CO 80301
Eric.Keller@xilinx.com

Abstract. A significant advantage of run-time reconfiguration (RTR) is that circuitry can be minimized or the delay path can be reduced through customization based on the current problem. Two reconfiguration techniques are common in todays applications. These include context switching and constant folding. In this paper I describe an implementation of a CORDIC processor and show how other reconfiguration techniques, including routing, can be applied and require low overhead if care is taken. CORDIC is an algorithm to efficiently calculate several different functions in hardware, such as sine and cosine. Through the use of JBits [1] and JRoute [5] the CORDIC processor can be customized to the specific problem, and can be reconfigured at run-time, with minimal changes, to fit another problem. In doing so, the complexity of the circuit is only as complex as it needs to be, and an optimal circuit is maintained. In this paper I describe the CORDIC algorithm and its implementation in an FPGA. The implementation is specific to the mode of operation of the CORDIC processor, it is not general purpose. That reduces the complexity of the circuit and RTR can be used to fit the given problem. I also present what has been done to reduce the run-time overhead to change between different modes.

1 Introduction

Much research has been done to explore run-time reconfiguration (RTR) [11][12][13]. Two common methods of utilizing run-time reconfiguration are constant folding and context switching. Constant folding is where a parameter that is normally an input to the circuit is worked into the circuit as a constant. Constants can be used as one of the inputs in a multiplier [7] or a key in a DES encryptor [10]. Context Switching is where the entire configuration is swapped out with a new, precompiled configuration. It has been used to reduce hardware for a Neural Network [8]. The CSRC (Context-Switching Reconfigurable Computing) chip by Sanders [4] is a specialized FPGA that swaps configuration in one clock cycle.

This paper presents a CORDIC processor that utilizes RTR. As presented here, RTR is used in more ways than just constant folding and doesn't use context switching. The CORDIC processor is set up according to the specified problem, and through RTR it can change to work for another specific problem. The circuit only gets as complex as it needs to and when it needs to.

While it can't change configuration in one cycle and the reconfiguration time is slightly greater than that of changing constants, this method of reconfiguration is more flexible than either. It involves finding similarities and designing circuits so that they can change into different circuits, while maintaining a similar structure.

2 CORDIC Background

CORDIC is an iterative algorithm that can be used to perform a wide variety of functions, including, but not limited to, sine, cosine, arc tangent, and square root. CORDIC, which stands for **CO**ordinate **R**otation **DI**gital **C**omputer, was first introduced by Volder [3] and can be used to efficiently find solutions to functions in hardware. Calculators such as the HP-35 have a CORDIC processor built in [6]. Walther [9] and others provide extensions that provide solutions to several functions not cited by Volder.

2.1 Variations

There are several types of operation that CORDIC can work in. The configuration determines what function will be calculated. Typically there are three inputs (x, y, and z) and three outputs (xout, yout, zout). However for certain configurations, a fourth input (c) is used. I will distinguish the configurations into modes and sets. Each set can be configured in either mode.

Modes and Sets There are normally two modes of operation. The first, called rotation mode, works by driving z to zero. The second, caled vector mode, works by driving y to zero. Each mode can be used with either circular set, linear set, or hyperbolic set. The following equations is the set of unified equations as described by Walther [9]. These unified equations are a combination of each set and mode with the variables and comparison conditions determining the set and mode being used.

$$x_{i+1} = x_i - y_i \cdot m \cdot d_i \cdot 2^{-i}$$

$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$

$$z_{i+1} = z_i - d_i \cdot e_i$$

where:

$$d_i = \begin{cases} -1 \text{ if } z_i < 0, +1 \text{ otherwise for Rotation Mode} \\ +1 \text{ if } y_i < 0, -1 \text{ otherwise for Vector Mode} \end{cases}$$

$$e_i = \begin{cases} \tan^{-1}(2^{-i}) & \text{for the Circular Set of equations} \\ \tanh^{-1}(2^{-i}) & \text{for the Hyperbolic Set of equations} \\ 2^{-i} & \text{for the Linear Set of equations} \end{cases}$$

$$m = \begin{cases} 1 & \text{for the Circular Set of equations} \\ -1 & \text{for the Hyperbolic Set of equations} \\ 0 & \text{for the Linear Set of equations} \end{cases}$$

The equations above can be used to calculate different functions. Listed in table 1 are the results you get for each configuration.

Circular - Rotation	Circular - Vector
$x_n = A_n[x_0 \cos z_0 - y_0 \sin z_0]$	$x_n = A_n\sqrt{x_0^2 + y_0^2}$
$y_n = A_n[y_0 \cos z_0 + x_0 \sin z_0]$	$y_n = 0$
$z_n = 0$	$z_n = z_0 + \tan^{-1}(y_0/x_0)$
$A_n = \prod_n \sqrt{1 + 2^{-2i}}$	$A_n = \prod_n \sqrt{1 + 2^{-2i}}$
Linear - Rotation	Linear - Vector
$x_n = x_0$	$x_n = x_0$
$y_n = y_0 + x_0 z_0$	$y_n = 0$
$z_n = 0$	$z_n = z_0 - (y_0/x_0)$
Hyperbolic - Rotation	Hyperbolic - Vector
$x_n = A_n[x_0 \cosh z_0 + y_0 \sinh z_0]$	$x_n = A_n\sqrt{x_0^2 - y_0^2}$
$y_n = A_n[y_0 \cosh z_0 + x_0 \sinh z_0]$	$y_n = 0$
$z_n = 0$	$z_n = z_0 + \tanh^{-1}(y_0/x_0)$
$A_n = \prod_n \sqrt{1 - 2^{-2i}}$	$A_n = \prod_n \sqrt{1 - 2^{-2i}}$

Table 1. The table shows the results of performing the CORDIC operation for each mode and set of operation.

3 JBits

Until recently RTR research was limited because of the lack of software to perform circuit changes. With the development of JBits [1], the possibilities are now open to include more techniques. JBits is a Java API that provides access to the Xilinx bitstreams. JBits main method is *set(row, col, resource, value)* which sets the resource (ie a PIP) to a certain value (ie ON).

4 JRoute

JRoute [5] is a run-time routing API that is built on JBits. Because of it's underlying JBits foundation, JRoute retains the same features of RTR that JBits has. JRoute gives the user various levels of control. It also provides support for cores by allowing the user to define ports. The ports are logical connections and because of the implementation, they allow for incremental design. That is to say, if a port is not to be connected to, the connection can be ommited. Then if later, at run-time, the port is needed, it can be connected to with no problems. JRoute also keeps track of resource usage, including the connections for each wire. This means that part or all of a net can be removed as needed.

5 Hardware Implementation

The CORDIC algorithm is made up of several iterations of add and shift operations, leading to an efficient hardware implementation. There are many ways to implement it in a FPGA, as pointed out in [6]. I chose to concentrate on a fully unrolled pipelined version implemented in a Xilinx Virtex™. Pipelining the operations can lead to very efficient implementation in hardware. Also, the shifts can be done in the routing to further reduce the logic.

Figure 1 shows the general form of a single stage. Included are three Add-Subtract units and a buffer. The Z add-subtract unit has a constant as one of the inputs, as it can be calculated before hand. The buffer is used to reduce the routing changes. It uses the Virtex carry chain to take a single input and outputs it for each clb in the buffer. Since the line that controls whether the add-subtract unit adds or subtracts, depending on mode of the CORDIC processor, the routing will need to change. The add_subtract line, $sign_i$ in figure 1, comes from either the sign bit of y_i or z_i. To limit the run-time routing and reduce the fanout, a buffer is inserted which connects to all of the add-subtract line sinks for each unit. This will decrease performance, however, it is an unneccessary step. Other forms of buffering to reduce fanout can be used and stubs can be used to reduce the run-time routing. Figure 2 shows the configuration of a clb to set the add-subtract unit up as add controlled. This means that when the add_sub line is asserted, the unit adds, and when it is deasserted, the unit subtracts. The unit can be set up as subtract controlled as well. This implies that the add-subtract line connects to every lut in the add-subtract unit. With a buffer, only a single route must be unrouted and rerouted to account for the change between rotation and vector mode.

Several stages are put together to form the CORDIC processor. The shifts shown in figure 1 are implemented by routing an output wire to the input at the shifted location. For example, input x shifted 2 bits would route input x[n] to the y adder input in[n-2]. The shifts also account for sign extension, routing x[n] to in[n] and in[n-1] as well.

Constant folding can be used to hardcode the precomputed angle rotations, ie $\tan^{-1}(2^{-i})$, to make the Z add-subtract unit a constant add-subtract unit. While this does not reduce the hardware for the add-subtract unit, it removes the need to have an additional register to hold the constant and the routing needed for an additional input.

Since some sets of operation, specifically hyperbolic and inverse, require double iterations that needed to be handled. Figure 3 shows that the double iterations were placed above the iteration it proceeds. In figure 3 the CORDIC stages have inputs called x_alt, y_alt, and z_alt. These allow the user to switch

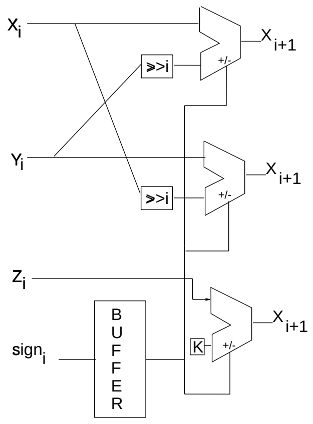

Fig. 1. Single stage in a CORDIC processor.

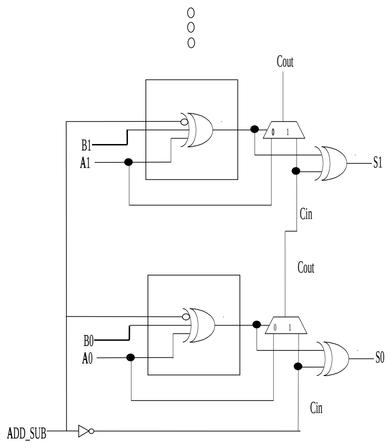

Fig. 2. Add-Subtract unit set up as add controlled.

between two different sets of operation, one requiring a double iteration and one not, while reducing the amount of run-time routing. Figure 4 shows a part of an add-subtract unit and how it can be configured to switch between two different groups of inputs, ie A0 and ALT_A. The B inputs are both connected to the LUT inputs at the same time. The LUT configuration will determine which input to choose from. The A inputs, however, cannot both be connected to the LUT inputs at the same time because there are only 4 inputs in a Virtex LUT. What is done in this case is to route the input up depending on the initial mode. Then when the mode is to be changed, the Single Line to LUT input connection is turned off and remembered. Then the alternate source is routed to the LUT input. Then when a change is required again, two low-level JRoute set calls are made, one to turn off the current connection, and one to turn on the other connection.

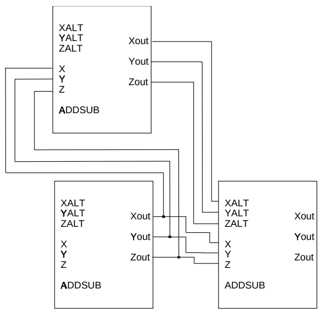

Fig. 3. The Double Iterations in a Cordic Processor can be connected to the X, Y, and Z alternate inputs.

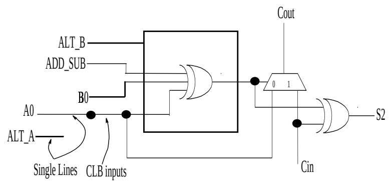

Fig. 4. Add-Subtract unit set up as add controlled.

To implement the inverse operations another input c is required. The value of y_i or z_i are no longer compared against zero so simply taking the sign bit is not acceptable. In this case a comparator is required. To keep from increasing the size of the CORDIC processor, the buffer and comparator need to be overlapped. The comparator is implemented as a subtractor with the sign bit being the result: 1 implies $a < b$ while 0 implies $a \geq b$. The buffer in this case cannot use the carry chain because the comparator uses it. So the sign bit of the comparator is routed to a bypass input for each clb. The CLB is then configured to pass the input through to the same output that the carry chain implemented buffer would.

6 Variable Accuracy

A benefit of being able to modify a circuit at run-time is the ability to increase the resolution, or number of bits of accuracy, of the result as some space on the chip frees up. Conversely, it also allows resolution to be sacraficed if space is needed. For example, in the CORDIC processor the number of bits of accuracy is related to the size of the add-subtract units for each stage and the number of stages. The way JRoute defines ports allows the user to create an adder that at run time can increase, or decrease, in size. Only the extra bits in the adder would need to get routed up instead of having to reroute the entire adder.

7 Conclusions

While it has been thought that run-time reconfiguration that includes routing has too much overhead, I have shown that this might not necessarily be the case. While using software based reconfiguration such as JBits and JRoute have overhead, certain applications might be able to accept a short reconfiguration time. This is better than the main stream tools that have far greater execution times. Also, while context switching works well with certain applications, the application is limited to what was defined at design time. A context switching approach wouldn't work in situations such as where the circuit is optimized based on a constant, for example as a key in a cryptographic system, because there are too many contexts to compile a bitstream for each. And while the CORDIC processor could be compiled for each operation, the circuit that it is a part of may not, making JBits and JRoute necessary.

8 Acknowledgements

Support for this work was funded in part by DARPA in the Adaptive Computing Systems (ACS) program under contract DABT63-99-3-0004.

References

1. S. A. Guccione and D. Levi, "XBI: A Java-based interface to FPGA hardware," *Configurable Computing Technology and its uses in High Performance Computing, DSP and Systems Engineering*, Proc. SPIE Photonics East, J. Schewel (Ed.), SPIE - The International Society for Optical Engineering, Bellingham, WA, November 1998.

2. D. Levi and S. A. Guccione, "BoardScope: A Debug Tool for Reconfigurable Systems," *Configurable Computing Technology and its uses in High Performance Computing, DSP and Systems Engineering*, Proc. SPIE Photonics East, J. Schewel (Ed.), SPIE - The International Society for Optical Engineering, Bellingham, WA, November 1998.
3. J. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. on Electronic Computing*, Vol EC-8, Num. 3, pp 330-334, Sept 1959.
4. S.M. Scalera and J.R. Vzquez, "The Design and Implementation of a Context Switching FPGA," *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA, April, 1998.
5. E. Keller, "JRoute: A Run-Time Routing API for FPGA Hardware," *7th Reconfigurable Architectures Workshop*, Lecture Notes in Computer Science 1800, pp 874-881, Cancun, Mexico, May, 2000.
6. R. Andraka, "A Survey of CORDIC Algorithms for FPGAs," *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, Monterey, CA, Feb., 1998
7. K. Chapman, "Fast Integer Multipliers fit in FPGAs," *Electronic Design News*, May 12, 1994.
8. J.G. Eldredge and B.L. Hutchings, "RRANN: The Run-Time Reconfiguration Artificial Neural Network," *Custom Integrated Circuits Conference*, pp 77-80, San Diego, CA, May 1994.
9. J. S. Walther, "A Unified Algorithm for Elementary Functions," Joint Computer Conference, pp 379-385, Spring 1971.
10. C. Patterson, "High Performance DES Encryption in Virtex FPGAs using JBits," *IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, CA, April, 1996.
11. W. Luk, N. Shirazi and P. Cheung, "Compilation tools for run-time reconfigurable designs," *IEEE Symposium on FPGAs for Custom Conputing Machines*, pp 56-65, April, 1997.
12. H. Schmit, "Incremental Reconfiguration for Pipelined Applications," *IEEE Symposium on FPGAs for Custom Computing Machines*, pp 47-55, April, 1997.
13. J. Burns, A. Donlin, J. Hogg, S. Singh, and M. deWit, "A Dynamic Reconfiguration Run-Time System," *IEEE Symposium on FPGAs for Custom Computing Machines*, pp 66-75, April, 1997.