Improving Functional Density Through Run-Time Constant Propagation[*]

Michael J. Wirthlin and Brad L. Hutchings
Department of Electrical and Computer Engineering
Brigham Young University
Provo, UT 84602

Abstract

Circuit specialization techniques such as constant propagation are commonly used to reduce both the hardware resources and cycle time of digital circuits. When reconfigurable FPGAs are used, these advantages can be extended by dynamically specializing circuits using run-time reconfiguration (RTR). For systems exploiting constant propagation, hardware resources can be reduced by folding constants within the circuit and dynamically changing the constants using circuit reconfiguration.

To measure the benefits of circuit specialization, a functional density metric is presented. This metric allows the analysis of both static and run-time reconfigured circuits by including the cost of circuit reconfiguration. This metric will be used to justify run-time constant propagation as well as analyze the effects of reconfiguration time on run-time reconfigured systems.

1 Introduction

Configurable computing systems based on Field Programmable Gate Arrays (FPGAs) are becoming viable alternatives to conventional computing approaches. These configurable computing machines (CCM) have been shown to outperform conventional computing systems such as standard microprocessors, high-end workstations, and even supercomputers. One of the methods used to achieve such high-levels of performance with limited hardware resources is circuit *specialization.* CCM circuits are specialized to the algorithm, interface, and user data-set to allow limited circuit resources achieve maximum levels of performance.

1.1 Constant Propagation

One of the most common methods of circuit specialization among CCM systems is the propagation of constants within arithmetic or other general-purpose operators. Many digital systems are composed of such operators that use a single fixed value for one of its inputs. Digital filters, for example, commonly use constant coefficients for the multiplier of each filter tap.

The circuit resources required to build such constant operators can be reduced by 'folding' the fixed operand value into the circuit [1]. Registers and decoding circuitry used to hold the operand value can be removed. In addition, standard logic optimization and minimization techniques can be used to further reduce the hardware [2].

Several CCM applications have used this technique to reduce hardware and improve circuit speed. Digital filters designed within FPGAs propagate filter coefficients into multipliers to free hardware for additional functionality [3, 4, 5]. One such "dedicated" IIR filter has been shown to fit in half the space of its general-purpose counterpart [6]. Other application areas demonstrating this technique on FPGAs include neural-networks [7, 8] and text-searching [1, 9, 10].

1.2 Run-Time Constant Propagation

If an operator within a system must support any arbitrary input value, special-purpose constant propagated operators cannot be used and the advantages of constant propagation are not available. With dynamically configurable hardware, however, arbitrary input values can be supported by reconfiguring constant-propagated operators at *run-time.* A form of run-time reconfiguration (RTR) [11], this technique allows circuits to achieve the advantages of constant propaga-

[*] This work was supported by ARPA/CSTO under contract number DABT63-94-C-0085 under a sub-contract to National Semiconductor.

tion while preserving the ability to support any arbitrary input value.

Run-time reconfiguration of constants is not a new technique — several working systems have been designed to demonstrate this concept. Such systems include the run-time reconfiguration of synaptic weights within a neural network [12] and the reconfiguration of templates within a pattern-matching system [13].

Reconfiguring constants within a circuit at run-time is not available without cost. The time required to reconfigure constant operators increases the total execution time of a system. This reconfiguration time can be quite large and may eliminate the benefits gained by constant propagation. The advantages associated with constant propagation are achieved only when carefully balanced against the extra time of reconfiguration.

Although working systems have demonstrated the advantages of run-time specialization techniques such as constant propagation, no metrics have been proposed or used to quantify these advantages. Further, it is unclear when the advantages of run-time constant propagation override the disadvantages of reconfiguration time. This paper will address these issues by introducing and using a simple functional density metric. This metric will be used to quantify the advantages of constant propagation in both statically and run-time reconfigured circuits. It will also be used to evaluate the effect of reconfiguration time in RTR systems.

Section 2 will introduce the functional density metric for both static and dynamic systems. Section 3 will describe a template matching circuit that will be used to demonstrate the functional density metric. Sections 4 and 5 will analyze the advantages of static and run-time constant propagation by analyzing the various template matching circuits. Section 6 will summarize the results of the analysis.

2 Functional Density

As discussed above, specialization techniques such as constant propagation reduce the area and improve the speed of FPGA circuits. A functional density metric, D, will be used to quantify these benefits of specialization. Functional density is a composite area-time metric used to identify the computational throughput (operations per second) of unit hardware resources. Functional density is defined as follows:

$$D = \frac{1}{AT}. \qquad (1)$$

Circuit specialization techniques that reduce the circuit area (A) or operating time (T) will improve functional density.

For this analysis, functional density will be used to evaluate the benefits of both static and run-time constant propagation. To compare these techniques, technology dependent measures will be used. The area, A, will be measured in the FPGA "cell-count" of the circuit. The operating time, T, will be measured as the execution time (t_{exec}) of the system based on the optimal "cycle-time" of the mapped FPGA circuits.

As suggested earlier, run-time reconfigured circuits have the added costs of reconfiguration time. In order to justify run-time reconfiguration, the improvements in area and time of specialized circuits must be balanced against the cost of reconfiguration. This trade-off can be made by adding reconfiguration time, t_{config}, to the operating time, T, of Equation 1 as follows:

$$T = t_{exec} + t_{config}. \qquad (2)$$

The run-time functional density can be written as:

$$D_{rtr} = \frac{1}{A(t_{exec} + t_{config})}. \qquad (3)$$

The cost of reconfiguration associated with each operation can be reduced by executing several operations before reconfiguring the hardware. For the execution of n operations between configuration, the functional density of Equation 3 can be modified as follows:

$$D_{rtr_n} = \frac{1}{A\left(t_{exec} + \frac{t_{config}}{n}\right)}. \qquad (4)$$

As more operations are completed between configuration steps (increasing n), the cost of configuration on a per-operation basis is reduced. The upper bound of D_{rtr_n} occurs when the execution time is much longer than reconfiguration time ($t_{exec} \gg \frac{t_{config}}{n}$). Under these conditions, functional density reduces to Equation 1 as follows:

$$\begin{aligned} D_{rtr_n} &= \frac{1}{A\left(t_{exec} + \frac{t_{config}}{n}\right)} \\ &\approx \frac{1}{At_{exec}}. \end{aligned}$$

The functional density of circuits employing run-time constant propagation is bound by the functional density of the same circuit that does not reconfigure ($t_{config} = 0$).

Equations 1 and 3 will be used to compare the functional density of run-time reconfigured circuits against their statically configured counterparts. Such a comparison may justify the use of run-time reconfiguration by showing that a dynamically specialized circuit provides greater computation per unit resource than its more general-purpose static alternative.

3 Template Matching

A simple template matching circuit will be used to demonstrate the benefits of both static and dynamic constant propagation. Several variations of this circuit will be discussed to demonstrate the advantages and limitations of run-time constant propagation. These circuit variations include a general-purpose circuit that does not exploit constant propagation, a staticly configured constant-propagated circuit, and several run-time reconfigured constant-propagated circuits.

Template matching is a common operation used in many image understanding systems. This operation, however, requires significant computation. Template matching computes the cross-correlation, C, between an image, $f[i,j]$, and a template $g[i,j]$. For a template of size $M \times N$, the cross-correlation is computed as follows:

$$C[i,j] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} g[k,l] f[i+k, j+l]. \qquad (5)$$

To simplify the computation, the template image, g, can be reduced to binary precision. This replaces the multiplications in Equation 5 with `AND` operations. No multiplications are needed and the entire operation can be performed with addition.

For this analysis, the image and template parameters of the Sandia Automatic Target Recognition will be used [14]. Input images are 8-bit precision with 128 × 128 pixels. Template images are 16 × 16 and have binary precision.

Each correlation result, $C[i,j]$, will be obtained by computing each of the 16 × 16 additions of Equation 5 in parallel using bit-serial arithmetic. Conditional bit-serial adders are used to perform the conditional addition at each template pixel $g[k,l]$. As seen in Figure 1, each operator contains a template register to hold the value of a template image pixel and a bit-serial adder. A logical '1' within the template register enables the bit-serial adder while a '0' disables the addition.

As seen in Figure 2, the 256 conditional adders are distributed spatially to represent each of the 256 template values, $g[k,l]$. The circuit is pipelined to produce one cross-correlation result, $C[i,j]$, every 16 cycles with an initial latency of 240 cycles.

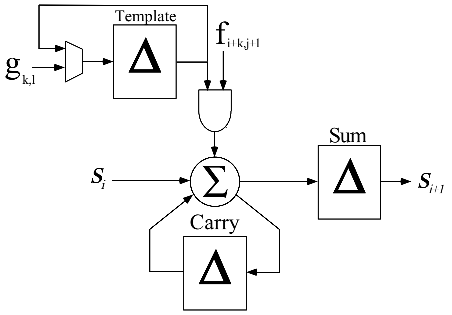

Figure 1: General-Purpose Conditional Bit-Serial Adder.

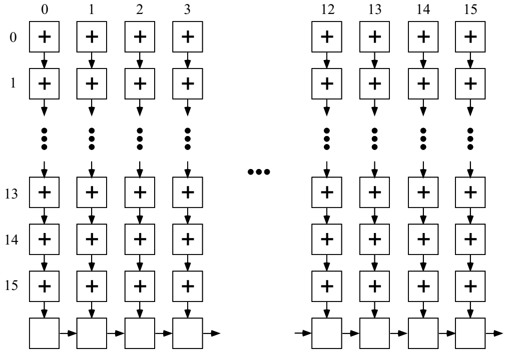

Figure 2: Parallel Computation of Correlation.

The conditional bit-serial adder circuit of Figure 1 was mapped to the fine-grain National Semiconductor CLAy FPGA [15]. This circuit consumed nine cells and operates with an internal delay of 9.4 ns. A complete correlation circuit is created by physically tiling 256 of these conditional bit-serial adders into a 16 × 16 array as suggested by Figure 2. The 256 operators consume 2304 cells and produce a correlation result every 150 ns. Using Equation 1, the functional density of this general-purpose circuit is calculated at 2890 $\frac{correlations}{cell-second}$.

4 Static Constant Propagation

If the template image for this correlation calculation is held constant, each template value can be propagated into the conditional bit-serial adder of Figure 1. Propagating each template value, $g[k,l]$, into the conditional bit-serial adder results in two unique circuits - one for the constant $g[k,l] = 0$ and another for the constant $g[k,l] = 1$. For the constant '1', the multiplexor, flip-flop, and AND gate associated with the conditional bit-serial adder are removed as seen in

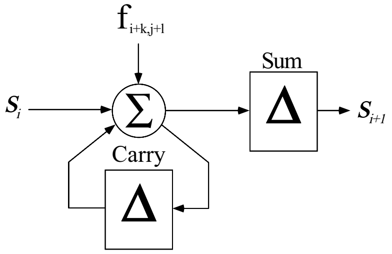

Figure 3: Special-Purpose Conditional Adder ($g[k,l] = 1$).

Figure 3. Propagating the constant '1' into the circuit reduced the FPGA resources from nine cells to six cells. In addition, removing the extra circuitry reduced the propagation delay down to 8.9 ns.

For the constant $g[k,l] = 0$, most of the hardware associated with the bit-serial adder is removed. Only a flip-flop remains to propagate the sum of the previous adder in the pipeline. This circuit requires only a single cell (flip-flop) and operates at a speed of 2.1 ns.

The two constant values, '0' and '1', produce circuits that vary with respect to area and time. To measure the functional density of a complete system, the worst-case '1' circuit will be used for area and timing parameters. The full array of six-cell bit-serial adders consumes 1536 cells and require 142 ns for a single correlation computation. Using Equation 1, functional density of this special purpose circuit is 4580 $\frac{correlations}{cell-second}$, a 58% improvement over its general-purpose counterpart. The functional density of both the general-purpose and constant propagated circuit are contrasted in Table 1.

	General-Purpose	Constant-Propagated
A	2304	1536
T	150 ns	142 ns
D	2890	4580

Table 1: Functional Density of Template Matching Circuit.

5 Run-Time Constant Propagation

For systems that must support any arbitrary template image, the template pixel values may be propagated into the circuit at run-time through reconfiguration. A special-purpose correlation circuit for each template image is created and configured as needed on the FPGA.

For this analysis, the template matching system will compute each correlation value of the output image against a single template before configuring a new constant template image. With an output image of size 113 × 113, 12769 correlation operations will be computed between configuration steps. This value is the iteration count, n, used to determine the run-time functional density in Equation 4.

Before Equation 4 can be applied to the template matching circuit, the configuration time, t_{config}, of the complete circuit must be determined. According to device specifications, the minimum configuration time of the CLAy31 FPGA is 810 us. Using this value for t_{config} and the circuit values found in Table 1, the functional density of this run-time reconfigured circuit is calculated at 3170 $\frac{correlations}{cell-second}$. This system provides 10% more functional density than its general-purpose counterpart.

As seen in Equation 4, the iteration count, n, has a significant effect on the functional density. Using the circuit parameters above, the functional density, D_{rtr_n}, of run-time constant-propagated circuits can be found as a function of the iteration count,

n. Figure 4 plots the relationship between the functional density and the iteration count of this constant propagated template matching system.

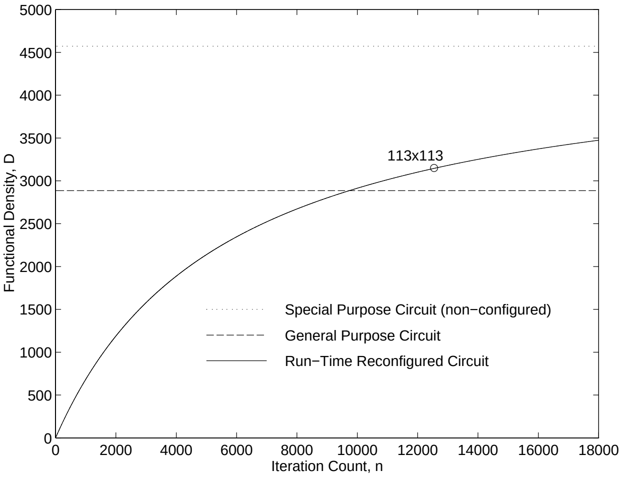

Figure 4: Functional Density as a Function of Iteration Count.

An important observation to make from Figure 4 is the location in the graph where the functional density of the run-time reconfigured (D_{rtr_n}) circuit matches the functional density of the general-purpose non-configured circuit (D_{gp}). At some iteration count, n_0, the functional density of both the run-time reconfigured circuit and the general-purpose non-configured circuit will be equal. This "break-even" point indicates the iteration count at which run-time reconfiguration is justified. This value can be derived by solving for n_0 as follows:

$$\begin{aligned} D_{gp} &= D_{rtr_n} \\ \frac{1}{A_{gp} t_{gp\_exec}} &= \frac{1}{A_{rtr}\left(t_{rtr\_exec} + \frac{t_{config}}{n_0}\right)} \\ n_0 &= \frac{A_{rtr} t_{config}}{A_{gp} t_{gp\_exec} - A_{rtr} t_{rtr\_exec}} \qquad (6) \end{aligned}$$

For the template matching circuit, the break even point is calculated as follows:

$$\begin{aligned} n_0 &= \frac{(1536 cells)(810 us)}{(2304 cells)(150 ns) - (1536 cells)(142 ns)} \\ &= 9760 \end{aligned}$$

This result indicates that at least 9760 correlation calculations must be performed before configuring the circuit with different constants. At this point, the functional density of the general purpose circuit matches the functional density of the constant propagated circuit. This would allow images as small as 99 × 99 to benefit from run-time constant propagation. Performing additional iterations between configurations (i.e. larger input images) will only improve the functional density of the circuit as indicated in Figure 4.

5.1 Improving Configuration Time

As seen from Equation 4, functional density is sensitive to reconfiguration time. Reconfiguration time is a parasitic effect that reduces functional density. Reducing the reconfiguration time has several positive effects on the system. First, the functional density improves. Second, the iteration count, n_0, required to break even decreases. Such improvements extend the technique to other systems where the technique would otherwise be unjustifiable.

Partial configuration is one technique that can significantly reduce reconfiguration time. This technique, available in several FPGA families [15, 16, 17], allows sub-circuits to be configured within the FPGA without configuring the entire device. For some systems, only minor circuit changes are required from one configuration to the next [18]. Partial configuration can significantly lower configuration time by configuring only *changes* within a circuit.

For the template matching circuit using constant bit-serial adders, partial configuration can have a significant effect on configuration time. Because changes to the circuit involve only local changes to the constant bit-serial adders, the configuration overhead for I/O and global routing is eliminated. In addition, only two of the six cells associated with the bit-serial adder are needed to convert a '1' cell to a '0' cell or vice-versa.

The two cells of logic required to modify a bit-serial adder require four bytes of configuration data. If every constant bit-serial adder within the circuit is modified between configuration steps, 1024 bytes are required to reconfigure the circuit. In practice, however, few template cells are actually modified. With an average of 128 bit-serial adders changing between configuration, only 512 bytes are required for configuration. Configuring at 10 MByte/s, this configuration processes consumes 51 us, or 15 times less time than the statically configured circuit.

As expected, reducing configuration time from 810 us to 51 us improves the functional density of the template matching circuit. Using Equation 4, functional density improves to 4460 $\frac{correlations}{cell-second}$ (54% improvement over the general-purpose circuit). Using Equation 6, only 614 iterations are required to justify run-time constant propagation. This allows images as small as 25 × 25 to take advantage of this technique.

Although improving configuration time by over a factor of fifteen increases functional density, further improvements in configuration time for this template matching circuit provide diminishing returns. Assuming device configuration speeds were improved by a factor of 100, configuration of the device would reduce from 810 us to 8.1 us. The functional density of this system is 4560 $\frac{correlations}{cell-second}$ (only a 2% improvement over the partially configured circuit). In this example, the faster configuration speed provided limited improvements over the partially configured circuit. For circuits that execute for a long time relative to the configuration time, improving configuration speed has a limited effect.

Although further improvements in configuration time does not significantly improve functional density for this example, it has a significant impact on systems that execute with less time than this example. Figure 5 plots the effect of iteration count on functional density for the three run-time reconfigured circuits (complete configuration, partial configuration, and fast configuration). For shorter execution profiles, this faster configuration provides significant improvements to functional density and the break-even point.

Table 2 summarizes the functional density and break-even points of the various systems.

Circuit	t_{config}	D	n_0
General Purpose	N/A	2890	N/A
Run-time Configured	810 us	3170	9760
Partially Configured	51 us	4460	614
Fast Configuration (100x)	8.1 us	4560	98
Statically Configured	0	4580	N/A

Table 2: Summary of Functional Density.

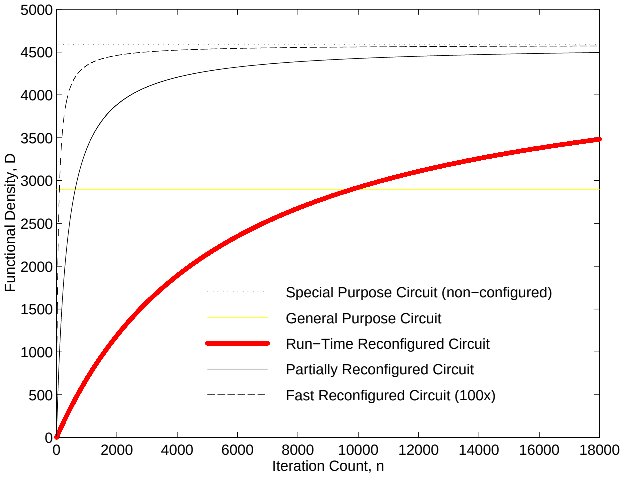

Figure 5: Improved Configuration Time.

6 Conclusions

Propagating constants within FPGA circuits is a specialization technique that often improves the computation of unit-cell FPGA resources (functional density). This technique, as well as other specialization techniques, can be used in dynamic systems by reconfiguring circuit elements at run-time. The advantages of run-time specialization, however, must be balanced against the added cost of reconfiguration time.

Systems that execute for many cycles between configuration steps can easily justify the added cost of reconfiguration. For systems that do not execute for extended periods of time, justification is not as clear. A functional density metric was presented to clarify the execution time (iteration count, n_0, in this case) required to justify run-time specialization.

Reconfiguration time can have a significant effect on both the functional density (D) and iteration break-even point (n_0) of RTR systems. Decreasing reconfiguration time will increase the functional density of the system. In addition, systems with smaller execution profiles become justified.

References

[1] P. W. Foulk. Data-folding in SRAM configurable FPGAs. In D. A. Buell and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 163–171, Napa, CA, April 1993.

[2] André DeHon. *Reconfigurable Architectures for General-Purpose Computing.* PhD thesis, Massachusetts Institute of Technology, 1996.

[3] A. Giri, V. Visvanathan, S.K. Nandy, and S.K. Ghoshal. High speed digital filtering on SRAM-based FPGAs. *7th International Conference on VLSI Design*, pages 229–232, January 1994.

[4] Les Mintzer. FIR filters with field-programmable gate arrays. *Journal of VLSI Signal Processing*, 6(2):119–127, 1993.

[5] G. R. Goslin. Using Xilinx FPGAs to design custom digital signal processing devices. In *1995 Proceedings of DSPX*, pages 565–604, January 1995.

[6] C. Chou, S. Mohanakrishnan, and J. B. Evans. FPGA implementation of digital filters. In *Proceedings of the Fourth International Conference on Signal Processing Applications and Technology*, pages 80–88, Santa Clara, CA, 1993.

[7] M. van Daalen, P. Jeavons, and J. Shawe-Taylor. A stochastic neural architecture that exploits dynamically reconfigurable FPGAs. In D. A. Buell and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 202–211, Napa, CA, April 1993.

[8] C. E. Cox and W. E. Blanz. GANGLION - a fast field-programmable gate array implementation of a connectionist classifier. *IEEE Journal of Solid-State Circuits*, 27(3):288–299, March 1992.

[9] B. Gunther, G. Milne, and L. Narasimhan. Assessing document relevance with run-time reconfigurable machines. In J. Arnold and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, CA, April 1996.

[10] E. Lemoine and D. Merceron. Run time reconfiguration of FPGA for scanning genomic databases. In D. A. Buell and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 90–98, Napa, CA, April 1995.

[11] B. L. Hutchings and M. J. Wirthlin. Implementation approaches for reconfigurable logic applications. In W. Moore and W. Luk, editors, *Field-Programmable Logic and Applications*, pages 419–428, Oxford, England, August 1995. Springer.

[12] P. Lysaght, J. Stockwood, J. Law, and D. Girma. Artificial neural network implementation on a fine-grained FPGA. In R. Hartenstein and M. Z. Servit, editors, *Field-Programmable Logic: Architectures, Synthesis and Applications. 4th International Workshop on Field-Programmable Logic and Applications*, pages 421–431, Prague, Czech Republic, September 1994. Springer-Verlag.

[13] J. Villasenor, B. Schoner, K. Chia, and C. Zapata. Configurable computing solutions for automatic target recognition. In J. Arnold and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, CA, April 1996.

[14] Michael A. Rencher. A comparison of FPGA platforms through SAR/ATR algorithm implementation. Master's thesis, Brigham Young University, 1996.

[15] National Semiconductor. *Configurable Logic Array (CLAy) Data Sheet*, December 1993.

[16] Atmel, San Jose, CA. *Configurable Logic: Design & Application Book*, 1993-1994.

[17] Xilinx. *XC6200 Field Programmable Gate Arrays*, 1996.

[18] J. D. Hadley and B. L. Hutchings. Design methodologies for partially reconfigured systems. In P. Athanas and K. L. Pocek, editors, *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 78–84, Napa, CA, April 1995.