Dynamic Power Reduction of FPGA-based Reconfigurable Computers using Precomputation *

Chi-Chiu Tsang, Hayden Kwok-Hay So
Department of Electrical and Electronic Engineering
The University of Hong Kong, Hong Kong
{cctsang, hso}@eee.hku.hk

ABSTRACT

This paper examines the effectiveness of employing precomputation techniques to reduce power consumption of field configurable computing systems. Multiplier is modified with precomputation techniques and are implemented using commercial off-the-shelf FPGAs. Precomputation techniques reduce dynamic power consumption of a module by eliminating unnecessary signal switching activities in inactive portions of the modules. Experiments have shown that up to 52% of logic and signal power consumption can be reduced in multiplier module. Furthermore, when compared to ASIC implementations, FPGA implementations of precomputation modules have the advantage of lower area overhead as most of them can be implemented using originally unoccupied related FPGA resources. Finally, it was found that the effectiveness of precomputation depends heavily on the input data statistics. It is expected that compilers for future reconfigurable computers may take full advantage of such power saving techniques by optimizing the architecture according to data input statistics.

Categories and Subject Descriptors

B.7 [**Integrated Circuits**]: Design Aids

General Terms

Design

Keywords

Field-programmable gate arrays, FPGAs, reconfigurable computer, precomputation, dynamic power reduction

1. INTRODUCTION

Over the past few decades, the demand for a new class of power-efficient high performance computer systems has been on a constant increase. On a microscopic scale, the performance of modern integrated circuits has already been shown to be heavily limited by their power dissipations. On a system level, the technical challenges and the associated cost of heat removal in large-scale data centers and server farms have also fueled the research for such a class of power-efficient high performance computing systems.

To address such problem, field configurable computer systems based on field programmable gate arrays (FPGAs) provides unique opportunities to implement optimized computer architecture for each application that may deliver optimal power-performance ratio. In such machine, a customized compute fabric, sometimes in the form of a hardware accelerator, can be generated on a per-application basis.

To that end, this paper examines the effectiveness of the use of precomputation as a way to lower power consumption in FPGA-based reconfigurable computers without incurring significant penalties in performance. Precomputation is a well-studied circuit technique that allows inactive portions of a module to be turned off dynamically during runtime. The reduced circuit switching activities reduces dynamic power consumption of the system at the expense of an increased power and area overhead for precomputation. One drawback of such technique is that the amount of precomputation that results in optimal power saving during run-time depends heavily on the input data statistics of an application. As a result, a circuit module with a fixed precomputation architecture is unlikely to provide significant power saving across a diverse set of applications. FPGA-based reconfigurable computers, on the other hand, provide a platform to customize such precomputation architecture depending on profiled data statistics for each application.

In particular, adders and multipliers enhanced with precomputation techniques are implemented on commercial off-the-shelf (COTS) FPGAs using standard vendor tools. The tradeoffs in power reduction, area overhead, and performance impact under different input load characteristics are studied. Based on the results, future directions in developing customized FPGA compute fabric on a per-application basis are proposed.

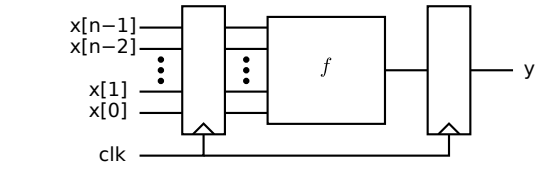
The rest of this paper is organized as follow: In Section 2, basic working principle and related work on precomputation will be explored. In Section 3 and Section 4, the design and performance of the pre-computation equipped adder and multiplier will be described. Future direction of this work and concluding remarks will be provided in Section 6.

2. PRECOMPUTATION TECHNIQUES

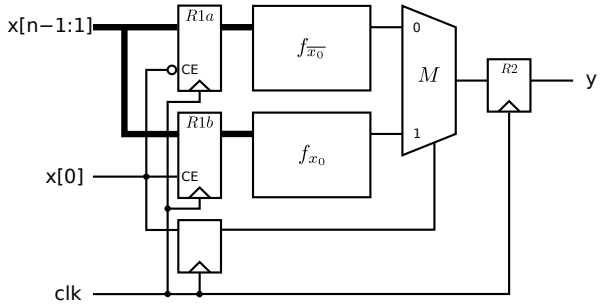
Precomputation is a well-known technique for reducing switching activity of a circuit that leads to reduced power consumption. Sometimes called guarded evaluation [2, 5], the basic idea of precomputation is to examine the input

*This work was supported in part by the Research Grant Council of Hong Kong, project HKU 716408E.

This work was presented in part at the first international workshop on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2010), Tsukuba, Ibaraki, Japan, June 1, 2010.



(a) Original circuit.



(b) Transformed circuit.

Figure 1: Transforming a simple sequential circuit using Shannon's expansion.

of a circuit module on each clock cycle to detect for conditions under which part of the complex circuit module may be disabled. In these cases, the output of the circuit module will be computed using a smaller sub-module or only part of the original module instead of the complete circuit. The reduced circuit activities consequently leads to reduced dynamic power consumption. For an in-depth discussion, please refer to [1]. Here, we present a simple case for illustration purpose.

Consider a circuit module that implements an n-input logic function $f(x_{n-1}, x_{n-2}, \ldots, x_0)$. Both the input and output are registered as shown in Figure 1(a). Now, consider the Shannon's expansion of f:

$$f(x_{n-1}, x_{n-2}, \ldots, x_0) = x_0 \cdot f_{x_0} + \overline{x_0} \cdot f_{\overline{x_0}} \quad (1)$$

where

$$f_{x_0} = f(x_{n-1}, x_{n-2}, \ldots, 0) \quad \text{and}$$
$$f_{\overline{x_0}} = f(x_{n-1}, x_{n-2}, \ldots, 1)$$

are cofactors of f with respect to x_0. Using this expansion, the original logic function is essentially split into two halves. Furthermore, depending on the value of x_0, only half of the logic is active.

Equation (1) can be implemented as shown in Figure 1(b). If $x_0 = 0$ in cycle t, then the input register $R1b$ is disabled because its clock enable (CE) port is deasserted. As a result, all signal switching activities within f_{x_0} are eliminated in cycle $t + 1$, saving significant dynamic power. On the other hand, the values $[x_{n-1}, \ldots, x_1]$ are loaded into input register $R1a$ as usual. These values will be used for the computation of $f_{\overline{x_0}}$ in cycle $t + 1$. Finally, the multiplexor (M) selects the correct result to load into output register $R2$.

Comparing the circuit in Figure 1(b) with the original implementation in Figure 1(a), we note a number of changes. First, the original function f is split into two, f_{x_0} and $f_{\overline{x_0}}$. Each of these cofactors depends on one less input than f. As a result, it is likely that the circuit implementations of f_{x_0} and $f_{\overline{x_0}}$ are both smaller than f. Furthermore, only one of the two will be active in each cycle. Therefore, the overall power consumption is expected to be lower than the original circuit.

However, the original n-bit input register $R1$ is almost duplicated into two $(n-1)$-bit registers, $R1a$ and $R1b$. The duplication of register has increased not only the load capacitance on the input x, but also the load on the clock signal, which has a 100% switching probability. This overhead must be taken into consideration when determining if such an implementation may result in lowered overall power consumption. That is, for such precomputation scheme based on Shannon's expansion to be able to reduce dynamic power consumption, the following must hold:

$$\alpha C > P(x_0)\alpha_{x_0} C_{x_0} + P(\overline{x_0})\alpha_{\overline{x_0}} C_{\overline{x_0}} + overhead \quad (2)$$

where $P(x_0)$ and $P(\overline{x_0})$ are probabilities of x_0 being 1 or 0 respectively, and subscripted α and C are the associated activity factors and load capacitance of f_{x_0} and $f_{\overline{x_0}}$ respectively.

In other word, the effectiveness of the circuit in Figure 1(b) in saving power depends critically on the arbitrary choice of input x_0. For example, it is preferable to pick x_0 such that the implementations of f_{x_0} and $f_{\overline{x_0}}$ are significantly smaller than f. Alternatively, it is desirable to pick an input x_0 such that the implementation of one of its cofactor functions is significantly simpler than the other and x_0 has a biased probability distribution. This way, the circuit will have a high probability of having the much simpler cofactor function active during run-time.

To the best of our knowledge, [3] was the first attempt to bring the guarded evaluation from the ASICs world to the FPGAs world. Howland et al. try to make use of the select signal of the multiplexors to trace back which logic blocks will be idle during computation. If there exists, they add some guarding logics to block the input signals of the logic blocks from transition in order to reduce the dynamic power. Their work is encourage, but is limited to use the select signal of the multiplexors to perform signal guarding. However, [2] further improves their method to make use of AND-inverter graph (AIG) to discover the non-inverting paths, so that the CAD tool can make use of this path to locate the signal, which can act as the enable signal of the logic block, to guard the signal transition. Hence, it can reduce the dynamic power.

For the two above discussed methods just use one single input to be the guarding decision signal. In general, instead of using only one single input as the determining factor for turning off part of the circuit module, similar precomputation techniques can be applied to any larger groups of input.

In the following sections, the design of adders and multipliers will be extended using similar precomputation technique described above. In most cases, the most significant m bits of the inputs are used to determine if the rest of the module can be turned off during runtime.

3. ADDERS

The action of the precomputation logic is best illustrated by the circuit diagram of the modified adder as shown in Figure 2.

The two inputs are divided into m-bit most significant parts and $(N-m)$-bit least significant parts. Denote the most and least significant parts of the two inputs a and b as a_H, a_L, b_H, and b_L respectively, then the original addition operation can be performed using two smaller additions as follows:

$$\begin{aligned} a+b &= \left(a_H \cdot 2^{N-m} + a_L\right) + \left(b_H \cdot 2^{N-m} + b_L\right) \\ &= (a_H + b_H) \cdot 2^{N-m} + (a_L + b_L) \quad (3) \\ &= (a_H + b_H + cout_L) \cdot 2^{N-m} + (s_L) \quad (4) \end{aligned}$$

where $cout_L$ is the carry out from the addition $a_L + b_L$.

For ease of discussion, denote the adder performing $a_H + b_H$ as $+_H$ and the adder performing $a_L + b_L$ as $+_L$, i.e., $a_H +_H b_H = s_H$ and $a_L +_L b_L = s_L$. From Figure 2, the main precomputation operation takes place in the first sign-extension check highlighted in the dotted box. This sub-module checks if the bits in a_H and b_H are simple sign extensions of the top bit of $a_L[N-m-1]$ and $b_L[N-m-1]$ respectively. If this condition is satisfied in both inputs, then s_H will be a simple sign extension of s_L, thereby eliminating the need of action from $+_H$. Finally, to reconstruct s_H, an additional check must be performed on $a_L[N-m-1]$ and $b_L[N-m-1]$ to determine the value of sign extension.

On the other hand, if either one of a_H or b_H is not a simple sign-extension of its respective least significant part, then s_H will be calculated as usual by adding a_H, b_H and $cout_L$.

3.1 Power Reduction

The adders with various degrees of precomputation, m, were implemented under the same software environment as in the previous section. Both $+_H$ and $+_L$ have been implemented first using standard ripple-carry adders native to Xilinx FPGAs, and then using custom-made carry lookahead adders. Furthermore, data with different statistics were used as inputs. Both uniformly distributed random numbers and zero-mean Gaussian random numbers with different standard deviations (σ) have been used as input data to evaluate the performance of precomputation under different data characteristics. The resulting power consumption are shown in Figure 3(a) and Figure 3(b). In both figures, only "logic + signal" power consumptions are shown for brevity, and their values are normalized to the case without precomputation ($m = 0$).

From the results in Figure 3(a) and Figure 3(b), a number of observations can be made.

First, both figures show a clear trend in power consumptions with respect to m. A jump in power consumption occurs between $m = 0$ and $m = 1$ when additional precomputation circuits are implemented on top of the base case. However, as m increases, the effect of precomputation also increases, lowering the overall power consumption until an optimal m is achieved. This optimal m, denoted m^*, is a function of the data input. In general, m^* increases as σ decreases. It is because as σ decreases, the absolute values of data inputs decreases, making it more likely that $+_H$ can be turned off. In fact, on the opposite side of the spectrum when the inputs are uniformly distributed random numbers, very little power saving is observed as both $+_H$ and $+_L$ are activated in most of the cases.

Secondly, in the cases where $+_H$ and $+_L$ are implemented as custom-made carry lookahead adders, up to 30% reduction of dynamic power consumption have been observed when compared to the base case when $\sigma = 1.0$. However, as σ increases, the optimal power saving decreases to close to the base case.

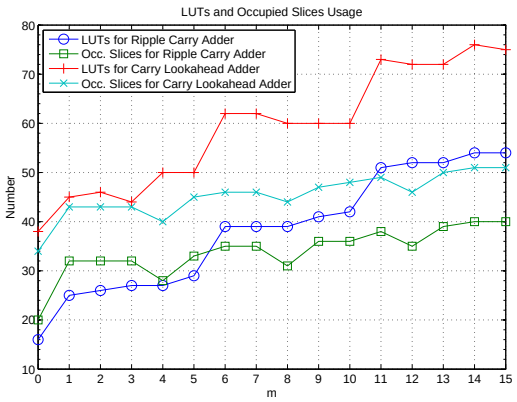


Figure 4: Occupied slices and LUTs vs. m for adders with precomputation.

Finally, when $+_H$ and $+_L$ are both implemented using native ripple adders, we see that the overhead of precomputation clearly outweigh the benefit of the power saving it may provide. Although the general trend of reduced power consumption can still be observed, none of them is capable of resulting in an overall power consumption lower than the plain native adder with $m = 0$. This is partly due to the significant overhead of precomputation in adder designs, but mostly an indication that the native FPGA adders are already very power-efficient, especially for small input sizes.

3.2 Resource consumptions

Figure 4 shows resource consumptions of adders with precomputation. In both cases with ripple adders and carry lookahead adders, the LUT usage increases almost linearly as m increases. This is due to the linearly increases in the size of sign extension units as well as large multiplexors that are used to construct the outputs. However, when compared to the increase in occupied slices, we see that the amount of occupied slices, usually a metric for FPGA design resource comparison, do not increase as fast as the increase in LUT usages. It is due to the fact that most of the increased LUT can be efficiently packed with the flip-flops used for pipeline registers in most of the occupied slices. As observed in [6], it is an property unique to FPGA platforms not available in ASIC designs.

4. MULTIPLIERS

Based on the design idea of the adders with precomputation logic, we have also extended a standard multiplier with similar techniques. Nevertheless, because of the complex inner operation of a multiplier, the working of the precomputation-equipped multiplier is considerably more complex than the two modules discussed so far. Our first attempt in designing such multiplier is depicted in Figure 5.

The basic working principle of the multiplier is as follows. A multiplier multiplies two N-bit signed inputs, a and b, and produce a $2N$-bit signed product, $p = ab$. Now, denote the most significant m_a bits of a as a_H and the rest of a as

Table 1: Condition under which each of the 4 sub-multipliers of the multiplier with precomputation should be activated.

a_u	b_u	$a_H b_H$	$a_H b_L$	$a_L b_H$	$a_L b_L$
0	0	off	off	off	signed
0	1	off	off	on	signed
1	0	off	on	off	signed
1	1	on	on	on	unsigned

a_L. Similarly the top m_b bits of b are denoted b_H and the rest are denoted b_L. Then, the product p can be calculated as:

$$\begin{aligned} a \times b &= \left(a_H \cdot 2^{N-m_a} + a_L\right) \times \left(b_H \cdot 2^{N-m_b} + b_L\right) \\ &= a_H b_H \cdot 2^{2N-m_a-m_b} + a_H b_L \cdot 2^{N-m_a} \\ &\quad + a_L b_H \cdot 2^{N-m_b} + a_L b_L \qquad (5) \\ &= \left(a_H b_H \cdot 2^{2N-m_a-m_b} + a_L b_L\right) \\ &\quad + a_H b_L \cdot 2^{N-m_a} + a_L b_H \cdot 2^{N-m_b} \qquad (6) \end{aligned}$$

Equation (5) indicates that the original N-bit multiplier can indeed be decomposed into 4 smaller multiplier modules as shown in Figure 5. In our first attempt to incorporate precomputation logic into a standard multiplier, these 4 smaller multiplier sub-modules form the basis that are turned off opportunistically.

It can be seen that if the most significant part of an input is a simple sign extension of the least significant part, then most of the computation involving the most significant part are redundant. In the case of the multiplier design, there are 4 combinations depending on whether such redundant computation is presented in the two inputs. Define two signals a_u and b_u such that $a_u = 1$ if and only if $a_H \neq$ sign extension of a_L and similarly for b. Using these notations, Table 1 shows when each of the four sub-multipliers should be turned on or off.

Note that in Table 1 the multiplier for $a_L b_L$, denoted $\times_{LL}$, is always on. Furthermore, it must perform either a signed or unsigned multiplication depending on the values of a_u and b_u. To eliminate such extra logic, $\times_{LL}$ is designed to be slightly wider than it needs to be in our current implementation with the top bits correctly inserted.

Finally, a special optimization is performed by regrouping (5) as (6). With such regrouping, note that within the first parenthesis, the values $a_L b_L$ never overlaps with that from $a_H b_H$. As a result, a simple concatenation of bits is used instead of an actual adder, which provides additional area and power savings.

4.1 Power Reduction

The multiplier with precomputation designs are implemented under the same software environment as before. In this work, N is fixed at 16.

Figure 6 shows the dynamic power consumption of our multipliers under different input data statistics and different degree of precomputation. In this figure, m_a is fixed at 9 except in the case where $m_b = -1$ and $m_b = 0$. For $m_b = 0$, a pure LUT-based 16×16 multiplier is used as a the base case. For $m_b = -1$, a DSP multiplier block in Xilinx Virtex-5 FPGA is used for reference.

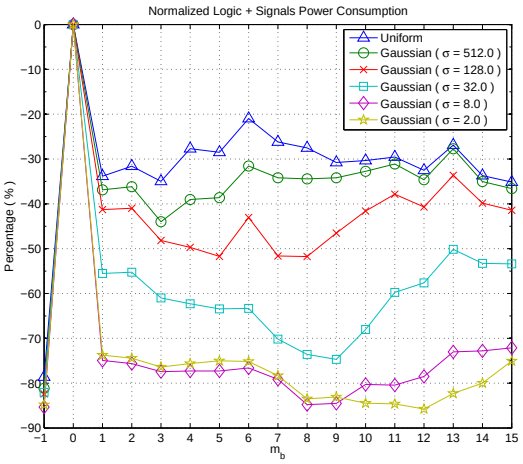


Figure 6: "Logic+signals" power consumptions of multipliers with precomputation normalized ($m_a = 9$) against standard unoptimized multiplier ($m_b = 0$) and hard block DSP multiplier ($m_b = -1$).

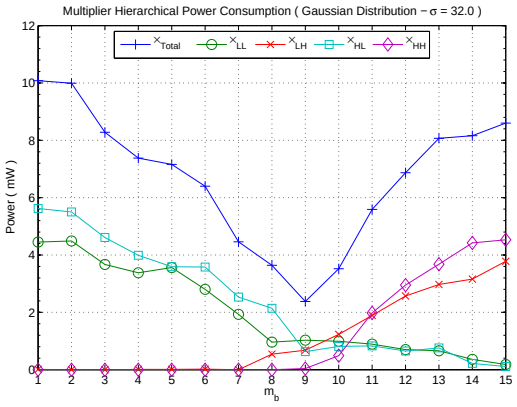


Figure 7: Contribution to overall power consumption from each sub-multiplier. Inputs are Gaussian random numbers with $mean = 0$ and $\sigma = 32$

There is a trend in different optimal m_b, i.e. m_b^*, can still be observed in Figure 6. In general, as σ decreases, making the input values closer to zero, overall power consumption decreases and m_b^* increases. This reiterates the benefit for FPGA-based reconfigurable computers in which customized arithmetic blocks can be used for different applications with different input statistics.

To examine the inner working of our multiplier, Figure 7 shows the power consumption of each individual sub-multipliers. Only one case where $\sigma = 32, m_a = 9$ is shown for brevity.

From Figure 7, it is clear that for small values of m_b, the two sub-multipliers that takes b_H as input, $\times_{LH}$ and $\times_{HH}$ are turned off most of the time. They are only activated more often when m_b increases because it is becoming less likely that b_H is a simple sign extension of b_L. On the other hand, the power consumption from $\times_{LL}$ and $\times_{HL}$ decreases as m_b increases because of their gradually decreasing sizes.

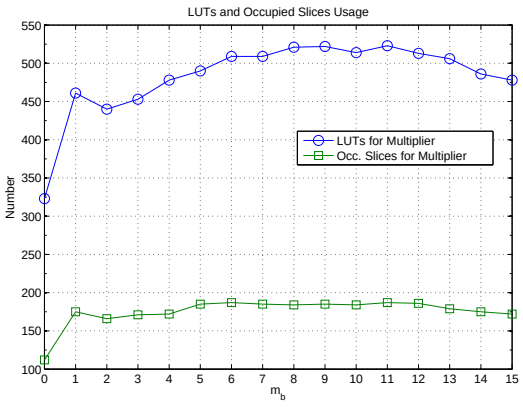


Figure 8: Occupied slices and LUTs usage vs. m_b for multipliers with precomputation, $m_a = 9$.

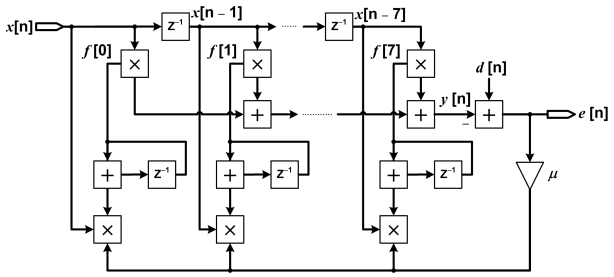


Figure 9: 8-tap LMS-based adaptive filter

4.2 Resource Consumptions

The resource consumption of the multipliers with precomputation shows an interesting concave trend as shown in Figure 8. It is due to the fact that as m_b varies, the sizes for all 4 sub-multipliers change. It results in the concave shape with maximum resource usages near the center when all 4 sub-multipliers are of similar sizes.

Note that in Figure 8 a standard multiplier without any precomputation logic ($m_a = m_b = 0$) consumes at least 30% less logic resources than even the smallest multiplier with precomputation, yet when compared to Figure 6, it can be seen that it consumes almost 50% more power. Further investigations revealed that the extra power consumptions are due to longer routes presented in the flat standard multiplier when compared to the hierarchical precomputation multipliers.

5. APPLICATION EXAMPLE

In order to illustrate the effectiveness of the precomputation-based dynamic power reduction, we have implemented an LMS-based adaptive filter [4], which makes use of the multiplier enhanced with precomputation to do the power analysis.

In our example filter, an 8-tap LMS-based adaptive filter is implemented. In the communication system, the transmission input always be interfered with different frequency sinusoidal wave and Gaussian white noise. We have x to be the system input of the adaptive filter, which is a sinusoidal wave, to predict the interference added to the original transmission signal in the system output of the filter, y. So that, finally we have d, the desired response of the adaptive filter, which is the noisy signal input to the communication system, so that we can filter out the interference and restore the original transmission code in the channel from the estimation difference from the adaptive filter, which is e in the Figure 9. $f[0]$ to $f[7]$ are the adaptive filter coefficients which are initialized zero and updated with the estimation e throughout the filtering process automatically. Last but not least, μ is the adaptation coefficient to control the adaptation rate of the filter.

In order to evaluate the power reduction efficiency from the precomputation enhanced multiplier, we first simulate the power consumption of the adaptive filter with the simple multipliers generated from Xilinx Coregen. Then, we replace them with our modified multipliers, which both the parameters m_a and m_b are equal to 6, to compare the power consumption change. After running the simulation, we can obtain roughly 46% reduction in dynamic power consumption with 40% increment in the usage of slice LUT-Flip Flop pairs.

6. CONCLUSIONS

This paper has demonstrated the designs and implementation tradeoffs of adders and multipliers enhanced with precomputation techniques on FPGAs. The use of precomputation techniques allows part of the circuit module be turned off dynamically during runtime that results in more than 50% of total dynamic power reduction. The power saving, however, depends highly on the input data statistics. In the design of the multiplier, for example, depending on the input data, logic power saving from the same multiplier may drop from 80% down to less than 20%. In the future, a smart compiler may perform profile-based optimizations that generate the optimal arithmetic units depending on the data statistics. Furthermore, dynamic runtime reconfiguration provides an extra degree of adaptation that the runtime operating system may take advantage of.

7. REFERENCES

[1] M. Alidina, J. Monteiro, S. Devadas, A. Ghosh, and M. Papaefthymiou. Precomputation-based sequential logic optimization for low power. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(4):426–436, Dec 1994.

[2] J. H. Anderson and C. Ravishankar. FPGA power reduction by guarded evaluation. In *Proc. ACM/SIGDA International Conference on Field Programmable Gate Arrays (FPGA'10)*, pages 157–166. ACM, 2010.

[3] D. Howland and R. Tessier. RTL dynamic power optimization for fpgas. In *Circuits and Systems, 2008. MWSCAS 2008. 51st Midwest Symposium on*, pages 714 –717, aug. 2008.

[4] U. Meyer-Baese. *Digital signal processing with field programmable gate arrays*. Springer, 3rd edition, 2007.

[5] V. Tiwari, S. Malik, and P. Ashar. Guarded evaluation: pushing power management to logic synthesis/design. *IEEE Trans. on CAD*, 17(10):1051–1060, October 1998.

[6] C. C. Tsang and H. K. H. So. Reducing dynamic power consumption in FPGAs using precomputation. In *Proc. IEEE International Conference on Field Programmable Technology (FPT'09)*, 2009.

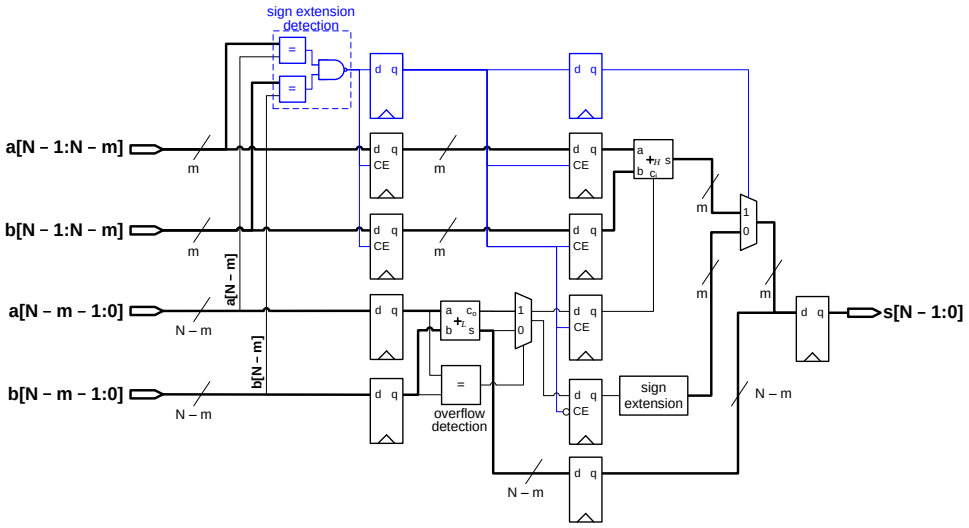
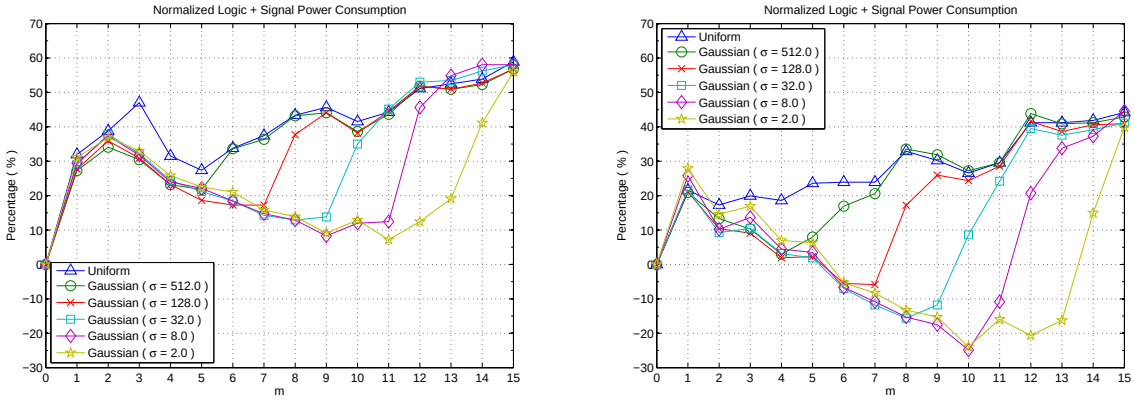


Figure 2: Adder with precomputation.



(a) Ripple Carry Adders

(b) Carry Lookahead Adders

Figure 3: Normalized "logic + signals" power consumption.

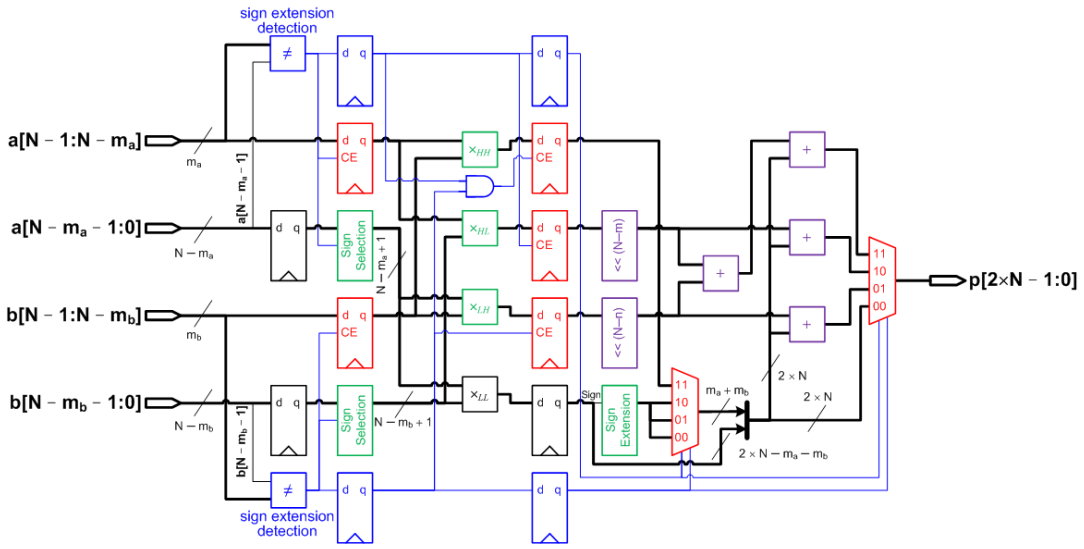


Figure 5: Multiplier with precomputation.