High Performance Branch Predictors For Soft Processors

by

Di Wu

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

High Performance Branch Predictors For Soft Processors

Di Wu
Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2014

Branch prediction has been extensively studied in the context of application specific custom logic (ASIC) implementations. Since the tradeoffs are different for reconfigurable logic, naïvely porting ASIC-based branch predictors to FPGAs may prove slow and/or resource-inefficient. Accordingly, this work studies the FPGA implementation of several commonly used branch predictor designs and does so in the context of simple pipelined processors, the most commonly used general purpose soft processor architecture due to its excellent balance of performance and resource cost. For this purpose, it assumes a pipelined processor implementation representative of Altera's Nios II-f and investigates the performance and resource cost of various branch predictors. The analysis confirms that existing designs are not efficient nor high-performing on reconfigurable logic. Accordingly, this work proposes FPGA-specific modifications that improve accuracy, resource cost, or both.

Acknowledgements

I have been fortunate enough to receive enormous help and support while completing this thesis. Firstly, I would like to express my deepest gratitude to my supervisor, Prof. Andreas Moshovos, for his unending guidance and support. Secondly, it has been my great pleasure to work with all the members of the AENAO research group for their feedbacks throughout this work. Special thanks to Kaveh Aasaraai and Ian Katsuno for their unreserved help on FPGA technologies.

Finally, I would like to thank my parents for all the unconditioned love and support during my undergraduate and graduate studies. Moreover, a heartful thank you to my beloved girlfriend Mengyun Shen for always being there cheering me up, I cannot imagine finishing this work without your constant understanding and faith in me.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

FPGA-based designs often incorporate one or more general purpose soft processors. As the range of FPGA applications broadens and evolves, it is likely that the performance demand from soft processors will increase. A key performance enhancing technique that even simple general purpose processors use is branch prediction. Without branch prediction, a branch has to execute completely before the processor can fetch the instructions that follow. Branch prediction eliminates these stalls by guessing the target address of branches. Current state-of-the-art branch prediction techniques, e.g., TAGE [1], rely on dynamically collected information about past branch behaviour. Such techniques have been proven to be very effective even in deeply pipelined, highly speculative, high-performance custom processor designs.

Branch prediction has been extensively studied in the context of application specific custom logic (ASIC) implementations. Since the tradeoffs are different for reconfigurable logic, naïvely porting ASIC-based branch predictors to FPGAs may prove slow and/or resource-inefficient. Accordingly, this work studies the FPGA implementation of several commonly used branch predictor designs and does so in the context of simple pipelined processors, the most commonly used general purpose soft processor architecture due to its excellent balance of performance and resource cost. For this purpose, it assumes a pipelined processor implementation representative of Altera's Nios II-f and investigates the performance and resource cost of various branch predictors. The analysis confirms that existing designs are not efficient nor high-performing on reconfigurable logic. Accordingly, this work proposes FPGA-specific modifications that improve accuracy, resource cost, or both.

This work proposes a minimalistic single-cycle branch predictor for Nios II-f. In more detail, this work makes the following contributions: (1) It studies the FPGA-implementation of Branch Target Buffers (BTB), including designs that fuse the BTB and the direction predictor and shows that, contrary to ASIC implementations, it is best to avoid a BTB and instead to calculate branch target addresses on-the-fly. (2) It studies the FPGA implementation of the three most commonly used branch direction predictors (DIR): bimodal [5], gshare, and gselect [6]. The analysis corroborates the results of past studies showing that gshare achieves the best accuracy among the three for practical table sizes, but also shows that unlike an ASIC implementation, frequency suffers with gshare on FPGAs. It proposes *gRselect*, an FPGA-friendly gselect implementation that uses a simple indexing scheme to outperform gshare by 11.4%. (3) It demonstrates that a conventional Return Address Stack (RAS) maps well onto the MLABs of FPGAs improving performance with little additional cost.

Other more accurate, albeit more elaborate predictors exist, such as perceptron [4] and TAGE [1].

These predictors often use multiple tables and tagged entries, which require comparator-driven multiplexers and thus may not map well onto FPGAs. As a result, single-cycle accesses may not be possible for these predictors. This work also considers to increase the number of pipeline stages in Nios II-f, use a simple bimodal predictor to provide a baseline prediction in a single cycle, and then the more elaborate branch predictors as overriding predictors. It shows that XXXXXXXXXXXXX????????????????????????????????????

Chapter 2

Background and Goals

Fig. 2.1 shows the organization of a typical branch predictor comprising: (1) a DIR, (2) a BTB, and (3) a RAS. The predictor operates in the fetch stage where it aims to predict the program counter (PC), that is the address in memory, of the instruction to fetch in the next cycle using the current instruction's PC and other dynamically collected information. The DIR guesses whether the branch will be taken or not. The BTB and the RAS guess the address for predicted as taken branches and function returns respectively. The multiplexer at the end selects based on the branch type and the direction prediction whether the target is the fall through address (PC+4 in Nios II), the target predicted by the BTB, or the target provided by the RAS. Since, at this point in time, the actual instruction is not available in a typical ASIC implementation, it is not directly possible to determine whether the instruction is a return, a branch, or some other instruction. Accordingly, a Selection Logic block uses either pre-decode information or a PC-based, dynamically populated lookup table to guess which target is best to use. With the latter scheme, when no entry exists in the lookup table, some default action is taken until the first time a branch is encountered. Once the branch executes, its type is stored in the lookup table where it serves to identify the branch type on subsequent encounters. This scheme is not perfectly accurate due to aliasing.

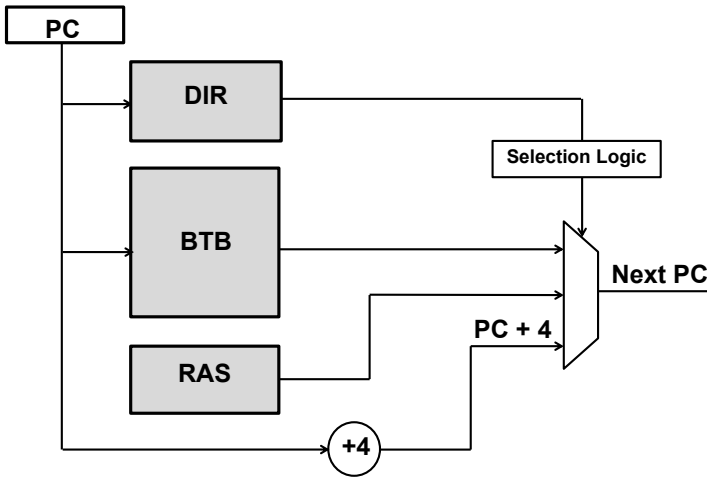

Figure 2.1: Canonical Branch Predictor.

2.1 Design Goals

This work aims to design a branch predictor that (1) balances operating frequency and accuracy to maximize execution performance, and (2) uses as few on-chip resources as possible.

This work proposes a minimalistic branch predictor for Altera's highest performing soft-processor Nios II-f. Since Nios II-f uses three BRAMs in total and only one BRAM is used for branch prediction [3], the proposed minimalistic branch predictor has a limited resource budget of one BRAM; each additional BRAM would represent a more than 1/3 overhead in terms of BRAM resources. The design of the minimalistic predictor considers the most commonly used direction predictors, bimodal, gshare and gselect. These predictors use a single lookup table and map relatively well onto a single BRAM.

This work also includes an investigation of perceptron and TAGE predictors. The study implements these predictors in a modified Nios II-f that has a deeper pipeline because these predictors often cannot be accessed within a single cycle. A simple bimodal predictor is used to provide base prediction, and the more elaborate predictor overrides the base prediction if it disagrees with the bimodal predictor.

Chapter 3

The Minimalistic FPGA-Friendly Branch predictor

This section discusses the architecture of the various FPGA-friendly branch predictors considered. Section 3.1 discusses target prediction, while Section 3.2 discusses direction prediction.

3.1 Target Prediction

A conventional method to predict branch targets is to use Branch Target Buffers (BTB). A BTB is a table that caches branch target addresses. When a branch executes for the first time, the BTB stores the target address so that it can be used on subsequent encounters of the branch. Ideally, the BTB would be large enough so that each branch can use a separate entry. In a practical implementation, however, aliasing will occur reducing prediction accuracy.

The simplest BTB design does not use an address tag per entry and directly predicts the target address for all instructions. Not using a tag results in a fast design that uses one direct SRAM lookup. In addition, not filtering non-branch instructions is desirable since at the time of access the instruction opcode is not available. Unfortunately, as Section **??** shows, when all instructions use the BTB, high destructive aliasing results in poor accuracy. To reduce aliasing, a small decode logic can prevent non-branches from updating the BTB. However, as Section **??** shows, while this solution increases target prediction accuracy by 30%, the additional logic reduces the maximum frequency by 35%. An alternative is to calculate the target address during the fetch cycle.

3.1.1 Target Address Pre-calculation

ASIC processor implementations use a BTB since the cache latency dominates the clock cycle leaving no room for further action. This is not true in an FPGA implementation where memory is generally faster than logic. This creates an opportunity to pre-calculate the target address for branches and thus eliminate the BTB. In this scheme the processor fetches the instruction from the cache and then, during the same cycle, calculates the instruction's taken address. As an added benefit, address pre-calculation may improve accuracy since, if possible, it is always correct. Unfortunately, it is not possible to pre-calculate the target address for all branches. The Nios II ISA includes two types of branches: *direct*

and *indirect*. The target of a direct branch can be calculated using the current PC and an offset that is embedded in the instruction. Indirect branch targets are read from the register file.

This work proposes enhancing target prediction with *Full Address Calculation* (FAC), which as Fig. 3.1 shows, calculates the target address for all direct branches and uses a BTB or some other storage for indirect branches. A selection logic identifies direct branches which can benefit from FAC. FAC selects among four possible addresses depending on the branch type. The Nios II ISA supports two schemes for direct branch target addresses, one uses a 16-bit offset (IMM16) and the other a 26-bit range (IMM26). Combined with the fall-through address (i.e., PC + 4) and the predicted address coming from the BTB, BTB+FAC uses a four-way multiplexer to select among these four possible addresses. Unfortunately, this multiplexer falls into the critical path.

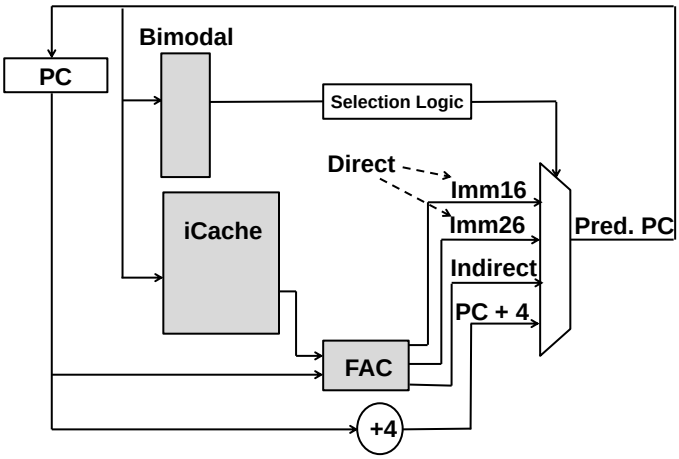

Figure 3.1: BTB with Full Address Calculation.

A lower cost and faster alternative to FAC is *Partial Address Calculation* (PAC) which relies on typical program behavior to reduce the number of choices for the final address multiplexer. Fig. 3.2 reports the relative frequency of the various branch types (see Section **??** for the methodology). Since IMM26 branches are far less frequent than IMM16 branches, PAC precalculates IMM16 branches and uses the BTB for IMM26 and indirect branches.

3.1.2 Return Address Stack

The RAS is a hardware, stack-like structure that accurately predicts the target address of function returns. When a call instruction executes, it also pushes its return address onto the RAS. Upon fetching a return instruction, the branch predictor can pop the top value from the RAS accurately predicting the return address. As long as the RAS has enough entries, it will accurately predict all return instructions. Since the call depth of typical workloads is not deep, virtually all high performance processors incorporate a shallow RAS. For the workloads studied a 16-entry RAS proves sufficient. The RAS for a simple pipeline is simple to implement on an FPGA. Deeper pipelines may require support for speculative RAS insertions and deletions complicating its design.

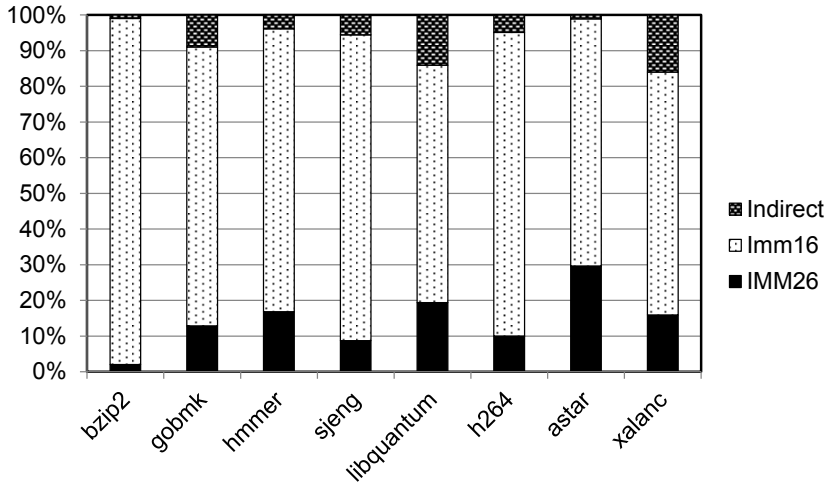

Figure 3.2: Branch Target Type Distribution.

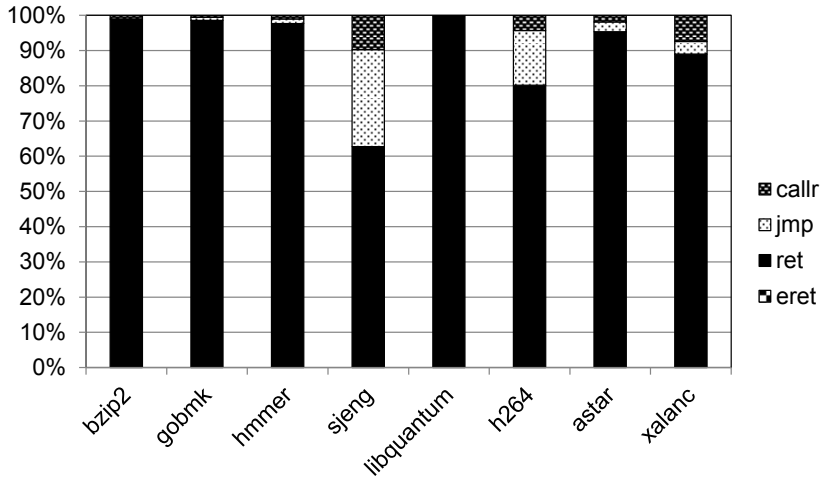

Figure 3.3: Indirect Branch Instruction Type Distribution.

3.1.3 Eliminating the BTB

Fig. 3.3 shows that for the workloads studied in this work, 97% of the indirect branches are returns (other workloads, e.g., those using virtual functions, may behave differently). Once a RAS is included along with FAC, the BTB ends up being used for only less than 1% of all branches. Accordingly, the BTB can be eliminated, and instead use a static, not-taken predictor for all indirect branches other than returns. Section **??** shows that removing the BTB in the presence of RAS reduces accuracy negligibly. Section 3.3.1 explains that lower-level FPGA related considerations also favor eliminating the BTB.

3.2 Direction Prediction

A bimodal branch direction predictor is a table of two-bit saturating counters that is indexed with a portion of the PC [5]. The counters are updated up or down depending on whether the branch is

taken or not respectively. The lower bit provides hysteresis to changes while the upper bit provides the prediction. As Section **??** corroborates, increasing the number of bimodal entries does not proportionally improve accuracy. Eventually, using a larger bimodal ceases to provide any improvement since bimodel is fundamentally limited on the branch prediction patterns it can predict.

Gshare is a pattern-based predictor wihch uses a combination of the PC and a global direction history register (GHR) to index the counter table [6]. GHR stores the direction of the last few branches in a bit vector. Each GHR bit stores the direction (taken or not) of a previous branch. Section **??** shows that gshare is far more accurate than bimodal. However, as Section 3.3 explains, latency suffers with gshare due to its more complex indexing scheme. Gselect, an alternative to gshare, indexes the counter table using a simple concatenation of the GHR and the PC [6]. For practical table sizes, gshare proves more accurate than gselect. Section 3.3.2 explains that with proper modification, gselect proves faster than gshare on an FPGA while sacrificing little in accuracy.

3.3 FPGA implementation optimizations

This section discusses additional FPGA-specific implementation optimizations. While this section assumes a modern, Altera FPGA, the optimizations presented should be broadly applicable.

3.3.1 Eliminating the BTB

As Section **??** explained, this work aims to use one M9K BRAM. An M9K BRAM can be configured as wide as 36 bits with 256 rows [2], and it can be used to implement a fused BTB and direction predictor. Specifically, each BRAM row can store one BTB entry along with up to three direction prediction entries for a total of 768 direction entries and 256 target entries. This fused BTB+DIR predictor works well with a bimodal DIR. The PC indexes a row which contains a single target prediction entry and up to three direction prediction entries. Another portion of the PC selects one of these direction prediction entries. The target is used only from taken branches.

Unfortunately, it is not possible to use one BRAM for both a BTB and the most accurate of the direction predictors considered, gshare. There are two reasons why: (1) gshare uses a different indexing scheme than the BTB, and (2) there is a limited number of ports per BRAM [2]. As Section **??** discussed, the BTB can be eliminated when address precalculation and a RAS are used. Eliminating the BTB frees up the entire BRAM for direction prediction.

3.3.2 FPGA-Friendly Direction Predictor Indexing

This section investigates which of the three branch direction predictors is best to use on an FPGA. Focusing just on accuracy gshare would be the best. However, performance is not the highest with gshare since its indexing scheme results in low clock frequency. Fig. 3.1 depicts why the predictor's indexing scheme, when implemented on an FPGA, falls into the critical path. At every clock cycle the predicted PC is used to index the direction predictor table for the next instruction. Since BRAMs are synchronous, their index must arrive before the clock edge and thus it cannot be a registered signal of the Predicted PC [2]. Moreover, the setup time for the BRAM is longer than that of simple registers. Therefore, the entire path starting from the BRAM data output, through the prediction logic and back into the lookup address of the BRAM forms the critical path. This is especially a concern with gshare

that uses the exclusive-or of the global history register (GHR) with Predicted PC to index the BRAM. This extra XOR logic prolongs the critical path, reducing the operating frequency.

Contrary to gshare, gselect has a simpler indexing scheme. Specifically, gselect uses a simple *concatenation* of the GHR with Predicted PC as index. Not only is gselect's indexing fast, but it can also be tailored to map well onto FPGAs. This work proposes *gRselect* which breaks the BRAM-to-BRAM critical path by breaking the BRAM access into two parts the first of which does not need to be "predicted PC". It uses GHR, which is a registered signal, to index the BRAM to retrieve one wide row of counters. Fig. 3.4 shows the gRselect scheme in more detail.

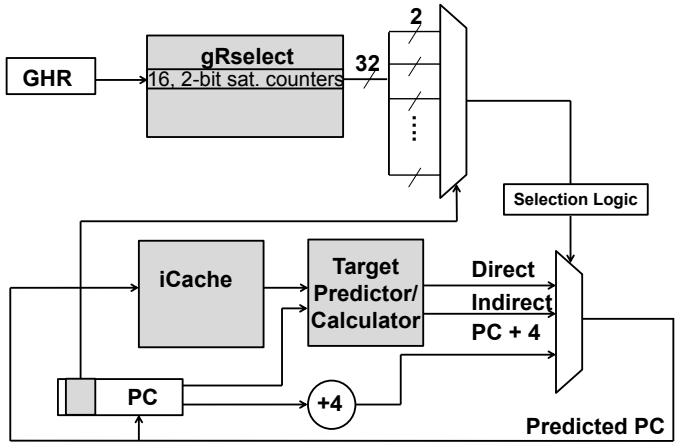

Figure 3.4: FAC with gRselect.

3.3.3 Instruction Decoding

To be able to select the appropriate target, the predictor needs to determine whether an instruction is a branch and if so, what kind of branch it is. This information is needed to select the corresponding predicted target through the output multiplexer. However, the decode logic lies in the critical path. To eliminate this delay, the predictor pre-decodes the instructions prior to installing them in the instruction cache. The pre-decode information is stored along with the instruction. This is similar to typical ASIC implementations.

Bibliography

[1] A. Seznec and P. Michaud. A case for (partially) tagged geometric history length predictors. In *Journal of Instruction Level Parallelism (http://www.jilp.org/vol7)*, April 2006.

[2] Altera Corp. *TriMatrix Embedded Memory Blocks in Stratix IV Devices*, Dec. 2011.

[3] Altera Corporation. *Nios II Processor Reference*, May 2011.

[4] D. A. Jimenez and C. Lin. Dynamic Branch Prediction with Perceptrons. In *Intl' Symposium on High-Performance Computer Architecture*, January 2001.

[5] J. E. Smith. A study of branch prediction strategies. In *Annual Symposium on Computer Architecture*, June 1981.

[6] S. McFarling. Combining Branch Predictors. TN-36, Digital Western Research Laboratory, June 1993.