Lightweight On-Chip Decoder Design for Information Hiding in Compiled Programs

Malcolm Taylor
University of Maryland Baltimore County

Dr. Min Wu
Dr. Gang Qu
Faculty Advisors

Department of Electrical & Computer Engineering

University of Maryland-College Park

Communication and Signal Processing Laboratory

Abstract

The importance of trusted computing has become more prevalent due to the amount of increasing computer security threats. As this trend continues, there has become a push for modifying the underlying architectural base of a computer to directly aide in the creation of a secure computing environment. Early work has shown that it is possible to hide information at the instruction set level and use the extracted data for security and authentication purpose. The goal of this project is to conduct a proof-of-concept design to validate this approach. Specifically, we have designed and simulated a small footprint chip add-on that can extract the hidden information and remap the instructions back to the original form for seamless architectural integration. The FPGA based prototype shows that the design requires about 0.2% hardware of a modern single core processor and drains only 0.07% of its power. We are currently looking at further changes to improve the speed of the design.

1 Introduction

1.1 Background

The instruction Set Architecture (ISA) of a processor dictates the final form in which a compiled computer program takes for interpretation by the cpu. This allows high-level computer languages to be abstracted to a much smaller predetermined set of instructions that can be fed to the CPU. With this level of abstraction in place the ability to generate multiplatform code becomes much easier. An instruction set comprising of fixed width instructions is one of the key aspects in Reduced Instruction Set Computing (RISC) architecture principles. RISC is a commonly used architecture because it allows quicker processing due to its fixed width instructions. Along with these performance gains comes minimal ability to modify or encode data to add side information. Because many of today's high-performance processors are striving for security gains, this strict architecture limits the possible ways in which a software/hardware security implementation can be instantiated.

1.2 Prior Research

In the past it has been shown that there is slight flexibility to embed information in instructions to aide in security. Recent research by Swaminathan et al.[1] looked further into the possibility of data hiding in a fixed framework such as RISC. The emphasis of the research focused on evaluating the feasibility of encoding extra data into the 26-bit operand of an instruction on the RISC architecture. It was determined that by remapping bit positions to smaller bit spaces, hidden information could be stored in the unused bits. Remapped bit positions were found using search algorithms tailored to this type of data. This lossless encoding scheme allows for full recovery of the original instruction therefore no underlying change to the architecture is needed.

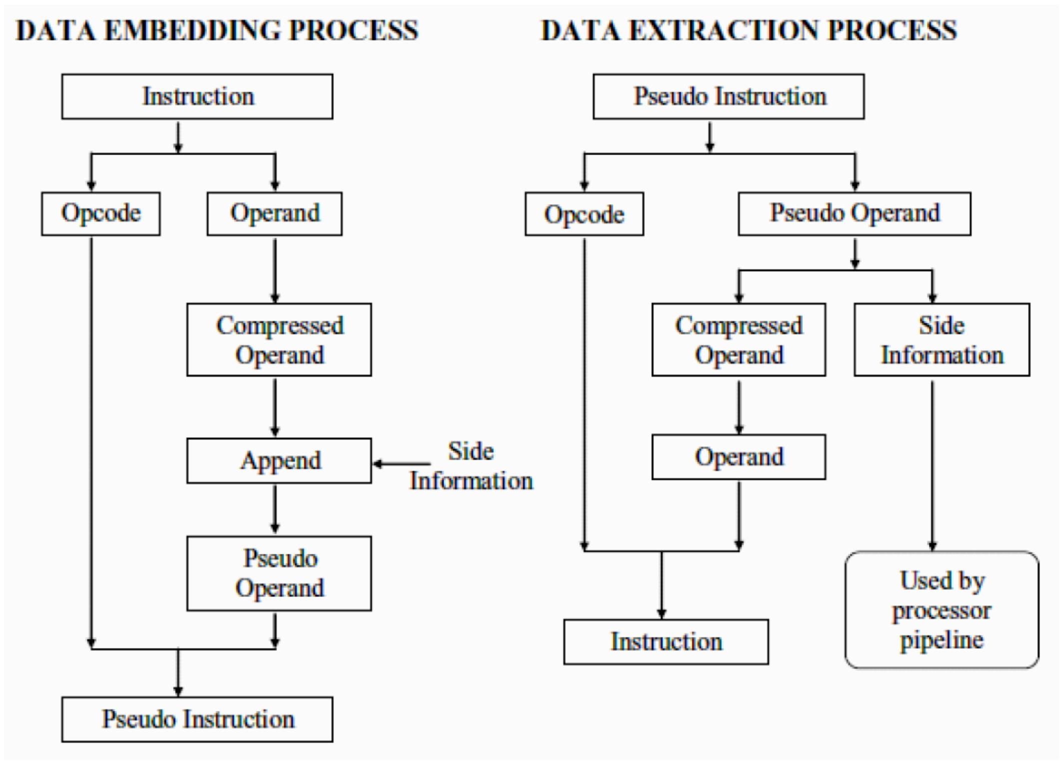
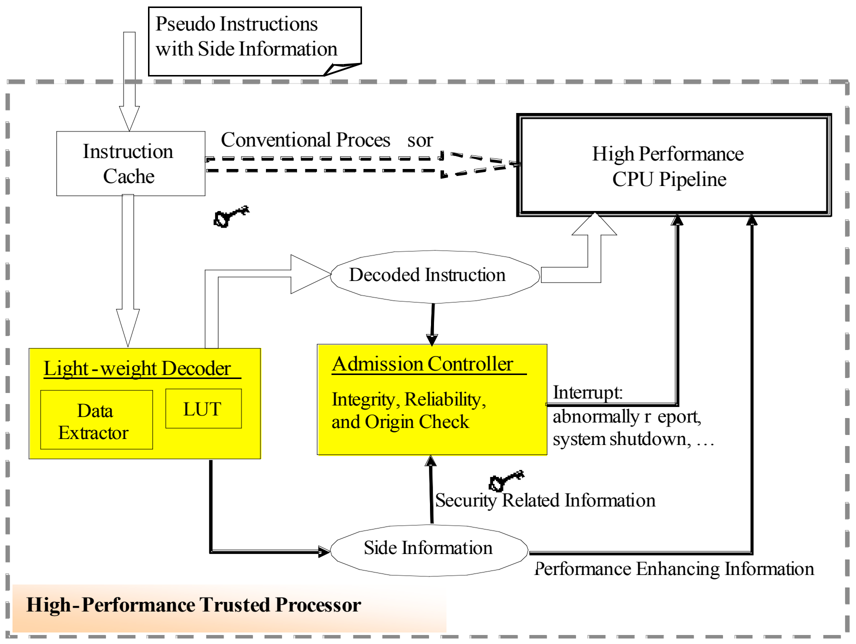


Figure 1: Encoding/Decoding Architecture

The diagram in figure 1 shows the framework proposed by Swaminathan et al. (2005). In software, the instructions for a given compiled binary are first broken into opcode and operand. The operand field of the instruction is much larger, and therefore is the target of compression. The process of compressing the data involves three different search algorithms that evaluate all of the bits contained in each of the instruction's operands and determine which positions are the most optimal for remapping. By finding the individual combinations for these positions a lookup table can be created that stores the original bit combinations and the new streamlined mapping.



Pseudo Instructions with Side Information
Instruction Cache
Conventional Proces sor
High Performance CPU Pipeline
Decoded Instruction
Light-weight Decoder
Data Extractor
LUT
Admission Controller
Integrity, Reliability, and Origin Check
Interrupt: abnormally r eport, system shutdown, …
Security Related Information
Side Information
Performance Enhancing Information
High-Performance Trusted Processor

Figure 2:High Performance Trusted Processor

A high-performance trusted processor architecture has also been proposed based on previously stated information hiding technique (see figure 2). In this approach, the hidden information, after being extracted by the on-chip decoder, can be used to facilitate trustworthy computing by, for example, checking the integrity, reliability, and authenticating the source of the code before execution. Under such framework, it becomes critical to implement both the decoder and the admission controller in hardware.

With this prior research in mind, the next step is to evaluate possible implementation strategies. The diagram in figure 1 shows a proposed High-Performance Trusted Processor framework. For this type of architecture to be feasible the implementation must be transparent to the Central Processing Unit (CPU). Even with this lossless encoding there must be a way to recover the encoded data. The encoder can be implemented in a modified compiler that does the remapping after creating the program. However, the decoder cannot be implemented in software because there is no advantage to this and the framework becomes a mirror of other common software solutions. To circumvent this problem the decoder can be implemented in hardware and perform the decoding inline with the CPU. Not only does this create a much more secure architectural base, but it is also faster than a possible software solution.

To complement the software side of the design, a hardware-based decoder is created to reverse the encoding process. The software process generates a lookup table and the positions of the remapped bits. These two sources of information alone are enough to attempt the reverse mapping process and can be stored on the chip. Along with storing this information, the hardware device must take in an already remapped operand, recover the secret information, and remap the operand back to its original form for processing by the CPU.

1.3 Goal

The goal of this project is to validate the concept of implementing decoder in hardware. In addition to showing that the design is capable of extracting the hidden information, we need to answer the following fundamental questions: how much hardware will be required, how much power it will consume, how flexible is the design, and can it keep pace with the modern high-performance processors? We provide a design that meets these requirements and simulate the design through FPGA based prototyping.

2 Challenges, Feasibility and Methods

In this section the challenges and possible design solutions for this type of hardware prototyping are investigated. Also, addressed is the feasibility of implementing such a design inline with already established architecture.

2.1 Challenges and Feasibility

Because of the complexity involved in abstraction of logic to a physical gate based representation, many complications arise when implementing algorithms in hardware. The goal is to determine the simplest design model that can easily be transformed into purely sequential and combinational logic. However, when implementing hardware-based solutions a large number of constraints and parameters have to be evaluated. This is especially true when designing a device that has to conform to an existing architecture, because many parameters are dictated by previously created standards. Ideally, the optimal design for this project will build off of existing high performance computing architecture and not hinder performance in any way; therefore some constraints must be addressed before proposing the design

2.1.1 Power

Power consumption is an important parameter due to the limited power availability on a modern high performance computer. With clock speeds increasing, common single core processors can use up to 100W[2] of power. Too much power consumption would make the design of the chip less feasible to implement and force the use of an external power supply. Additionally, the inclusion of a power supply forces a much larger chip footprint.

2.1.2 Gate Count

Another important parameter to minimize is gate count. Removal of complex logic and the reuse of existing memory structures aids in the transistor gate reduction. Within the architectural footprint, where a high-performance processor resides, there is limited space and a larger chip size will make implementation less feasible. Today's current single core processors have a transistor count of nearly 40 million[3]. The goal is to create a design that dwarfs the CPU in gate count, therefore minimizing implementation issues when moved to actual application in the given architecture

2.1.3 Gate Delay

Minimized gate delay is important when working in the high-performance processor domain. The processor is the fastest component in the architecture and it is frequently waiting on data to be sent therefore, the goal is to create a design that does not cause the CPU to wait longer. Performance loss will be noticeable if the chip delays incoming operands by a large amount. As with the other challenges, the best way to address this issue is by minimizing the design complexity.

Minimizing power, gate count and gate delay guarantees a proposed design that is tailored for the target platform. It is important to address these issues before the design process to frame the architectural limitations.

2.2 Design Methods

2.2.1 High Level Design

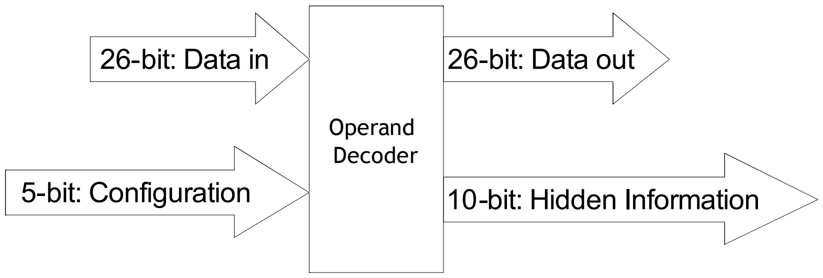


Figure 3: High level Design

After evaluating the decoding process specifics, a design consisting of 2 input buses and 2 output buses was formulated (see figure 3). These four buses will allow full functionality yet minimize the amount of connections that need to be made to the existing architecture. To accomplish this minimalist port design, the data-in bus must be used for receiving information from the encoding process and pass-through data. To accommodate this design, multi-state logic must be used.

The first operation state of the design is the configuration state where the chip takes configuration information through the data-in bus. Enabling this state is achieved by setting the input value of the configuration bus to a nonzero number. The data-in bus takes three different types of configuration settings while in this mode. The first is the *Bit Remapping* bit mask, in which 1s are used to represent the bits that have been found to be the most optimal for remapping. The second is the *Secret Information* bit mask. This bit mask represents the bits that will be used for the information hiding. Rather than determining these bits by evaluating the amount of mappings, to reduce complexity, this information is retrieved from the encoding side of the design. Finally, the data in bus takes information about the individual lookup table values. These mappings are sent in sequentially according to the value that the actual index maps back to. For example if 8 maps to the value 0 then it is the first bit combination to be sent in.

Using these configuration settings, the individual components of the design are setup. After configuration, is the hardware device acts as a pass-through for the input operand. The goal is to have seamless transition from the data-in bus to the data-out bus with minimal delay.

2.2.2 Proposed Hardware Implementation

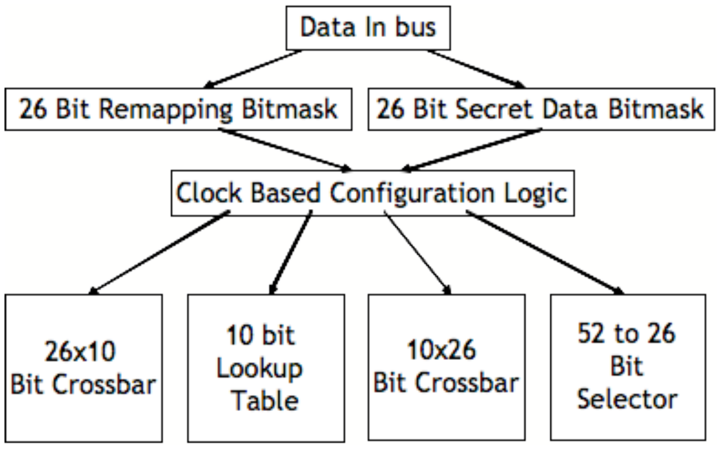


Figure 4-1: Configuration Mode

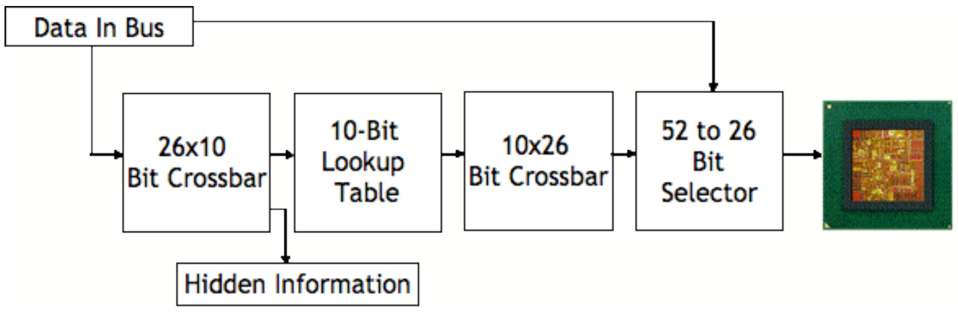


Figure 4-2: Pass-through mode

After evaluation of the proposed design goal a formal internal architectural layout was constructed (see Figure 4-1 and 4-2). The architecture is formed using the specified minimization parameters and the proposed high-level design. Since this is a two state design there must be an architectural setup for both states. Additionally, components that can work without clock dependency in pass-through mode are needed because of the stringent timing goals of the overarching architecture.

After a thorough evaluation of the decoding scheme, it is apparent that processing such a dynamic input creates a complex problem. When retrieving data in the lookup table, the actual table index can be pulled from anywhere in the operand due to the 2^{26} possible combinations of the Bit Remapping Bitmask. This cannot be avoided because the bit positions of the remapped bits are different depending on the combinations of instructions of the current program.

To address this issue we determined that the use of a crossbar device was necessary. A crossbar is a device that can map any input to one or multiple outputs and can be configured multiple times in hardware. The crossbar will interconnect the data in bus and the lookup table to create the address for data retrieval. Additionally, another crossbar must be used to remap the outgoing bits of the lookup table back to their original bit positions.

Finally, some of the inputs must pass directly through the hardware device. To minimize the power dissipation these inputs need to be dynamically mapped to the outputs. Facilitation of this is possible through a simple bit selector component. This device is comprised of 26 2 to 1 bit multiplexers that can be programmed individually, therefore if a bit is not remapped the input for the given bit becomes the output. However, if the bit is remapped the output bit is selected from the second crossbar device.

All of these components are configurable but also allow for pass-through functionality. This is the type of design needed for the two-state architecture that was initially proposed. Additionally none of the devices have clock dependencies in pass through mode therefore this piece of hardware should not create problems when integrated into existing high-speed processor architecture.

2.2.3 Prototyping Platform

To test the proposed design a prototyping platform had to be selected. For this a Field Programmable Gate Array (FPGA) would offer the most design flexibility. An FPGA allows for a hardware design process that is similar to that of a software design as it allows for multiple configurations and simple testing.

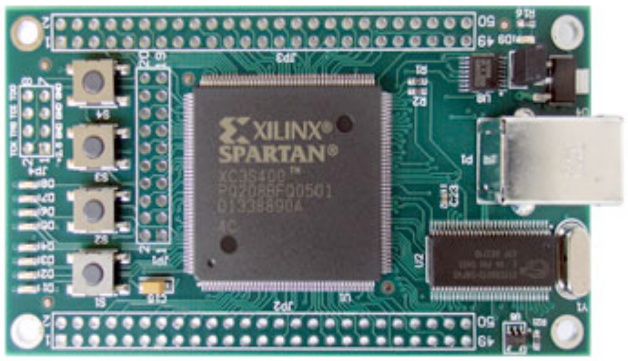


Figure 5: Opal Kelly™ XEM3001v2

The specific FPGA chosen is the Opal Kelly™ XEM3001v2, shown in figure 5 above. The XEM3001v2 is based around the XILINX™ Spartan 3 FPGA architecture however; Opal Kelly™ has added easy an easy to interface USB connection. The ability to send data to the design through the USB protocol allows for quick testing. Along with data input and output capabilities the USB connection is also used to for FPGA configuration. To configure the FPGA, Verilog HDL was used and synthesized to bit code in the XILINX™ Integrated Software Environment (ISE).

3 Results

After implementing the design on the XEM3001v2 the next step involves testing the hardware device with actual configuration and operand data. Additionally, measurement of the design minimization parameters occurred to determine if any design changes need to be made.

3.1 Operational Test

The original test was done within the XILINX™ ISE software through the use of a logic simulator. These tests did not implement the USB functionality therefore allowing uninhibited measurements of specific design parameters. Also, simple configuration and pass-through data was used to determine if the design worked as planned. These tests determined that the design successfully could take in configuration input and remap incoming operands.

The next step involved interfacing the design with the USB port on the Opal Kelly XEM3001v2. To test this actual implementation, a simple python program was created utilizing functions from the provided Opal Kelly API. Functions were created to reset, configure, and send data through the data-in bus. The data out bus was monitored to determine if the design was correctly processing the operands. With 100 percent success the hardware device was able to store configuration information and decode the passed in operands.

3.2 Gate Count/Size

The gate count of the implemented design totaled 95,456, which is 56% utilization of the Spartan 3 FPGA. This meets gate count minimization goals because 95,000 gates is ~0.2% of a modern single core processor. Adjusting the size of the lookup table, the amount of outputs on the first crossbar, and the amount of inputs on the second crossbar will minimize the gate count further however; these changes will adversely affect the ability to store hidden information.

The amount of gates is also directly related to component size. With this knowledge, an estimation of the components dimensions can be determined. The Spartan 3 FPGA used on the XEM3001v2 is packaged in a 456-ball fine-pitch ball grid array of dimension 17x17 mm or 289 mm^2. Assuming there is 56% utilization, a device of the size ~13x13 mm or ~162 mm^2 with the same underlying packaging could be created from the design. This is a very small add-on and will fit into existing high performance architectures.

3.3 Gate Utilization

With this working design 50% of the gates are used for logic while the other 50% is used for storage. These numbers are promising and illustrate that the lookup table is sized correctly for the proposed device implementation. Even with modifications to this design, this ratio will not change much due to the interdependency between logic and storage size. If fewer outputs are used on the first stage crossbar then the lookup table size will contain less values and vice versa.

3.4 Power Consumption

To determine the power consumption of the design, the XILINX^{TM} Xpower program was used. Xpower can generate power statistics on a HDL described hardware device using real world statistics and input specifications from the user. The design sent to Xpower did not contain the USB interface because in actual implementation this would not exist. After sending the data to XPower statistics were generated.

With the setup and clock speeds stated earlier, Xpower reported an average power consumption of 65 mW. This varies slightly depending on the operand input at a given time. 65 mW is well within the target power consumption goal and is only ~0.07% of a current single core processor.

3.5 Gate Delay

To determine the gate delay, the statistics generated by the logic synthesizer was used. After synthesizing, a list of overall gate delay is generated containing the worst case for the design. By evaluating this data, there shows a possible worst case 30 ns delay. For the current tests this is fine but for actual implementation and interface with a high-speed processor this may be too large of a delay.

One of the main factors contributing to the delay is the lookup table. Currently the table is implemented as an asynchronous read, synchronous write block RAM. This is not the most efficient memory structure and in actual implementation it would most likely be instantiated as a high-speed cache. Actual fabrication of the design with this component would most likely remove a large amount of the delay. Another option is attempting this design on a speed optimized FPGA. XILINX™ produces a version of the Spartan 3 FPGA with a higher speed grade and this would possibly minimize a large majority of delay.

4 Conclusion

In this work we determined the feasibility of implementing lossless encoding based decoding module in hardware while conforming to the standards of the already existing architecture. Our design is able to easily decode operands that have been encoded with the previously proposed lossless scheme and should have minimal problems interfacing with a high-performance processor. The design was also able to stay well within the minimization parameter goals while still being fast, efficient and lightweight. This research shows that a software/hardware co-design based security solution can be implemented with minimal hardware while still fully supporting the dynamic input generated by software encoding side of the framework.

---

[1] A. Swaminathan, Y. Mao, M. Wu, and Krishnan Kailas: “Data Hiding in Compiled Program Binaries for Enhancing Computer System Performance,” 7th International Information Hiding Workshop (IHW), Barcelona, Spain, in Lecture Notes on Computer Science (LNCS), June 2005.

[2] Intel Corporation, Intel Microprocessor Quick Referenc. Retrieved August 1, 2007, from Intel Microprocessor Quick Referenc Web site: http://www.intel.com/pressroom/kits/quickreffam.htm

[3] Intel Corporation, Press Kit -- Moore's Law 40th Anniversary. Retrieved August 1, 2007, from Moore's Law 40th Anniversary Web site: http://www.intel.com/pressroom/kits/events/moores_law_40th/