REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
14-09-2011	Final	13-09-2011 to 14-09-2011

4. TITLE AND SUBTITLE

(DURIP-10) HELIX PROJECT TESTBED- TOWARDS THE SELF-REGENERATIVE INCORRUPTIBLE ENTERPRISE

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-10-1-0292

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Frederic Chong

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

UCSB
Santa Barbara CA 93111

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Robert Herklotz
robert.herklotz@afosr.af.mil
Air Force Office of Scientific Research
875 North Randolph St.
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-OSR-VA-TR-2012-0786

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution A - Approved for Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Information flow is an important security property that must be incorporated from the ground up, including at hardware design time, to provide a formal basis for a system's root of trust. We incorporate insights and techniques from designing information-flow secure programming languages to provide a new perspective on designing secure hardware. An important result of our DURIP is a new hardware description language, Caisson, that combines domain-specific abstractions common to hardware design with insights from type-based techniques used in secure programming languages. The proper combination of these elements allows for an expressive, provably-secure HDL that operates at a familiar level of abstraction to the target audience of the language, hardware architects.
Additional work in this DURIP was to construct a minimal but configurable secure architectural skeleton. This skeleton couples a critical slice of the low level hardware implementation with a microkernel in a way that allows information flow properties of the entire construction to be statically verified all the way down to its gate-level implementation.

15. SUBJECT TERMS

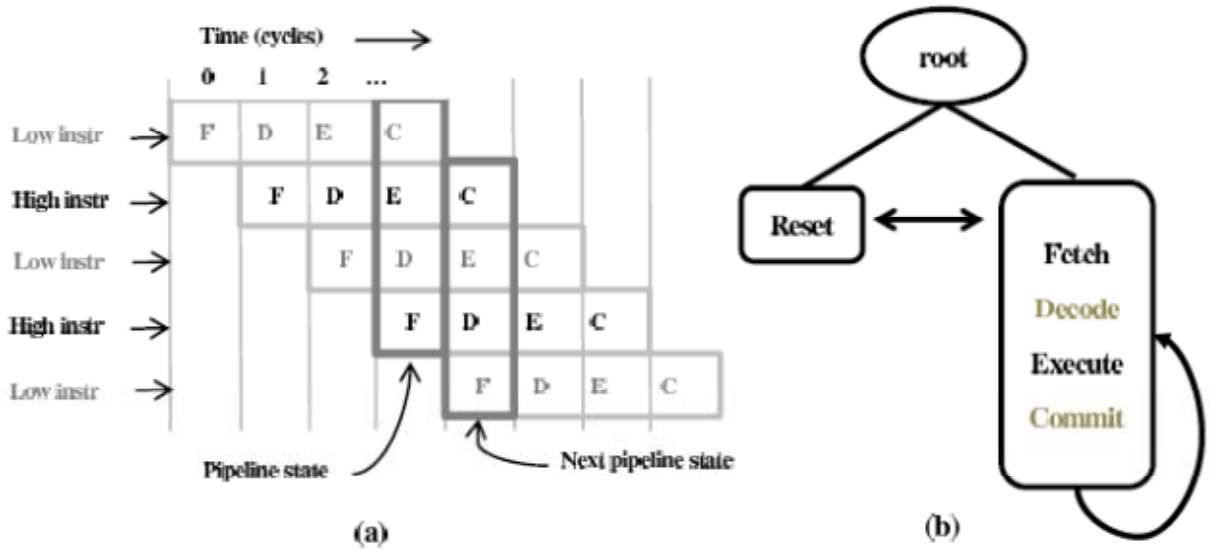
Secure hardware, hardware description language, information flow tracking, security policies

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Frederic Chong
U	U	U	UU	6	**19b. TELEPHONE NUMBER** *(include area code)* 805-310-7931

Information flow is an important security property that must be incorporated from the ground up, including at hardware design time, to provide a formal basis for a system's root of trust. We incorporate insights and techniques from designing information-flow secure programming languages to provide a new perspective on designing secure hardware. An important result of our DURIP is a new hardware description language, Caisson, that combines domain-specific abstractions common to hardware design with insights from type-based techniques used in secure programming languages. The proper combination of these elements allows for an expressive, provably-secure HDL that operates at a familiar level of abstraction to the target audience of the language, hardware architects.

We have implemented a compiler for Caisson that translates designs into Verilog and then synthesizes the designs using existing tools. As an example of Caisson's usefulness we have addressed an open problem in secure hardware by creating the first-ever provably information-flow secure processor with micro-architectural features including pipelining and cache. We synthesize the secure processor and empirically compare it in terms of chip area, power consumption, and clock frequency with both a standard (insecure) commercial processor and also a processor augmented at the gate level to dynamically track information flow. Our processor is competitive with the insecure processor and significantly better than dynamic tracking.

Our DURIP infrastructure has allowed us to simulate and emulate our designs. We evaluate a processor design with the following structure:



To quantify the hardware design overhead introduced by our approach we compare our processor design (Caisson) with a non-secured, simplified version of the commercial Nios Processor (Base) and the same Nios processor augmented to dynamically track information flow using GLIFT (GLIFT), previous work under our associated MURI.

GLIFT implements full system information flow tracking at the logic gate level: it associates each bit in the system with a taint bit indicating its security level, and augments each gate in the hardware design with additional gates that compute taint propagation.
All CPUs have identical functionality and configuration. However both Caisson and GLIFT can only utilize half of the cache and memory capacity effectively although they have identical configuration as the Base processor. The reason is that in our Caisson design the memory and cache have to be partitioned into two parts with different security levels, while GLIFT needs to associate a one-bit tag for each bit in the memory and cache. We implemented the Base processor (from the Nios design) in Verilog with no additional security features. To get the Caisson implementation we remodeled the Base implementation using security widgets provided by the Caisson language and statically partitioned all registers, caches, and memories into Trusted and Untrusted. To get the GLIFT implementation, we first synthesized the Base design into a gate level netlist and then augmented the netlist with shadow logic to track information flow. We passed the Base and Caisson designs through Altera's QuartusII v8.0 tool
to synthesize the designs onto a Stratix II FPGA for functional testing and verification. We then obtain the area, timing and power results using the Synopsis Design Compiler and the SAED 90nm technology library assuming a switching activity factor of 50% for the circuit.

Almost as important as the quantitative performance results are the *qualitative* results of how easy each design was to implement— this is an important test for the usability of a language. We find anecdotally that Caisson is easily usable by a programmer trained in Verilog. The original Base design required 709 lines of Verilog—the corresponding Caisson design required only 724 lines and took little additional time to implement. By contrast, GLIFT required
us to make a hard choice: we could either (1) manually design the gate-level netlist at a structural level (i.e., manually place the logic gates to create the design), which in our experience is infeasible for such a complex design; or (2) generate a gate-level netlist from the behavioral Verilog design using an existing tool, then automatically generate the GLIFT shadow logic using the resulting netlist. We used the latter option, and while it simplifies the process for the programmer the resulting design is intractably difficult to debug and optimize.

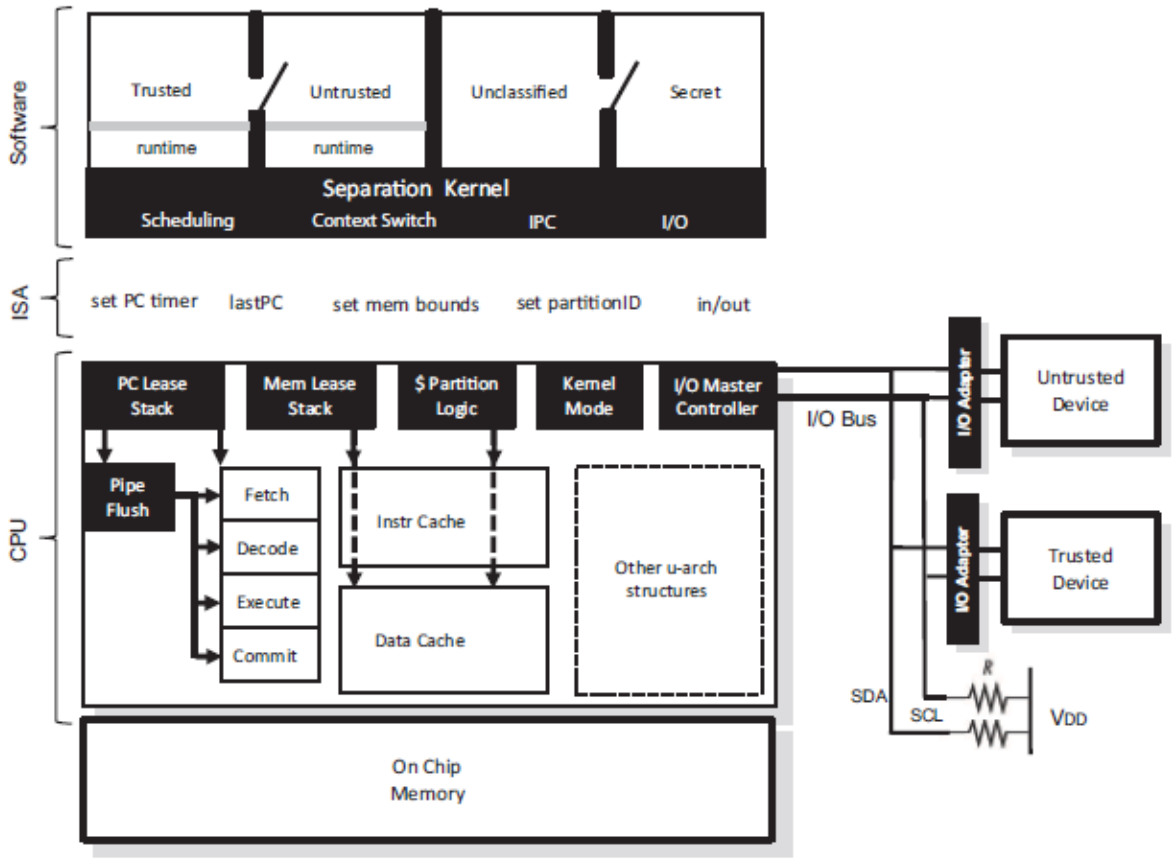
	Base	GLIFT		Caisson	
Synthesis Time (min)	1:50	153:56	**83.96X**	4:04	**2.22X**
Area (μm^2)	38462.86	128506.89	**3.34X**	52088.78	**1.35X**
CPU delay (ns)	2.96	7.79	**2.63X**	4.32	**1.46X**
Power (μW)	614.148	1730	**2.82X**	666.912	**1.09X**

The table above gives the performance figures for each design. We give the concrete numbers for all three designs as well as normalized numbers for Caisson and GLIFT (using Base as a baseline). The area numbers do not include the memory hierarchy

since all three designs use an identical memory configuration. The power numbers include both dynamic and leakage power. The GLIFT design comes with a large overhead in all four performance categories due to the shadow logic that GLIFT introduces to the processor. This shadow logic takes a long time to synthesize, requires a large chip area, consumes a great deal of power, and drastically slows the processor cycle frequency.

Caisson, in contrast, has a much lower overhead, though it is certainly not free. This overhead mainly comes from two sources: the duplicated state (i.e., registers) and the additional encoders and decoders used to multiplex the partitioned state onto the same logic circuits. We note that the overhead generated by the Caisson de- sign does not grow with CPU complexity (e.g., number of functional units)—a more powerful and complex CPU would not require any additional overhead, while the GLIFT design's overhead *would* proportionately with the CPU complexity. For perhaps the most important performance metric, power, Caisson's overhead is almost negligible. The synthesis time for the Caisson design includes type-checking, which is sufficient to verify the design's security. The GLIFT synthesis time does *not* include verification GLIFT only detect security violations at runtime.

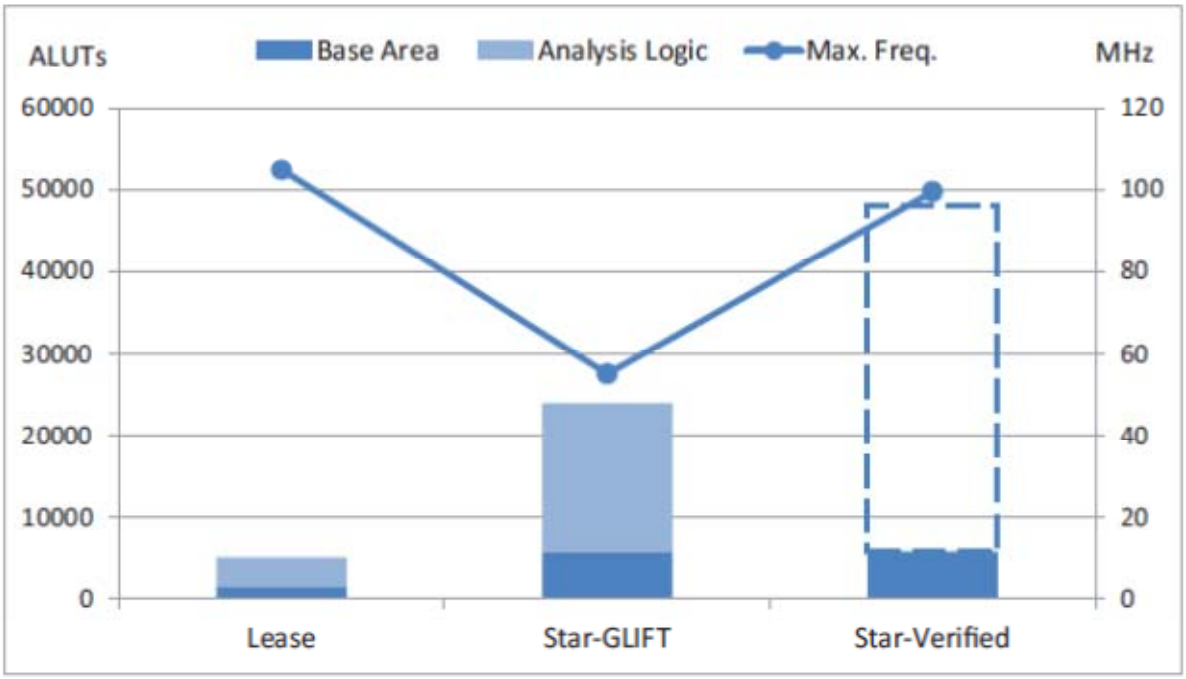
Additional work in this DURIP was to construct a minimal but configurable secure architectural skeleton. This skeleton couples a critical slice of the low level hardware implementation with a microkernel in a way that allows information flow properties of the entire construction to be statically verified all the way down to its gate-level implementation. This strict structure is then made usable by a runtime system that delivers more traditional services (e.g. communication interfaces and long-living contexts) in a way that is decoupled from the information flow properties of the skeleton. To test the viability of this approach we design, test, and statically verify the information-flow security of a hardware/software system complete with support for unbounded operation, inter-process communication, pipelined operation, and I/O with traditional devices. The resulting system is provably sound even when adversaries are allowed to execute arbitrary code on the machine, yet is flexible enough to allow caching, pipelining, and other common case optimizations.



Above is the proposed architectural skeleton (shaded black in the CPU) that allows explicit software control over the entire processor state. The processor includes dynamic micro-architectural features such as caches and pipelining. This hardware skeleton is used by a separation kernel to manage execution time, memory, and I/O devices among multiple security partitions. We also introduce trusted adapters for secure I/O. Here, an I2C master controller on the CPU manages a shared bus among off-the-shelf I2C devices with different trust levels. In the end, we verify that the hardware and kernel together enforce a desired information flow policy such as non-interference.

Our DURIP allowed us to evaluate our CPU design (Star-CPU) in detail and compare its functionality and area-delay with our prior work MURI work. The Star-CPU
pipeline is single-issue, executes in-order, and has 4 stages (fetch, decode, execute, and commit/write-back). It has 8 general purpose registers, a mode bit to indicate kernel/user mode, and a partition ID register to record the current security context. The memory hierarchy includes a 2kB direct-mapped data cache, and 64kB each of instruction and data memory. The data cache is implemented on the FPGA using comparator logic and registers and requires one cycle if a memory access is a hit, while the memory is implemented using on-chip block RAMs that take two cycles to service

a memory request. To emulate memory access latency in an ASIC implementation of the system, the memory controller is implemented to introduce an additional delay of 100 cycles. Without micro-architectural features such as branch predictors, TLBs, and Out-of-Order execution, the Star-CPU pipeline stalls on each cache miss and requires the compiler to ensure that a register used in a conditional jump instruction has the desired value at least 4 instructions before it is used.



The above graph quantifies the size and performance advantages of the Star-CPU against the Execution Lease CPU and against the Star-CPU with dynamic GLIFT logic (Star-GLIFT). The Star-CPU provides caches, pipelining, and kernel support beyond the Lease CPU in equivalent area and clock-frequency, and provides static security guarantees compared to Star-GLIFT in almost 1/4 the logic, 1/2 the memory, and 2X the clock-frequency.

The Star-CPU's base functionality is implemented in 5756 ALUTs (Adaptive Look-Up Tables in an Altera FPGA, where 1 ALUT corresponds very approximately to 9-12 gates), and while the base functionality in the Lease CPU requires only 1511 ALUTs, it requires 5040 ALUTs when the dynamic analysis logic is factored in. Thus the Star-CPU replaces analysis logic overhead with a cache and pipeline logic. In terms of performance, the Star-CPU and Lease CPU have similar frequencies (99 MHz vs. 104MHz), but the unpipelined Lease CPU only commits one instruction every 5 cycles. Further, without a cache, every memory access in the Lease CPU goes to main memory.

Comparing the verified Star-CPU to Star-GLIFT, we observe that the Star-GLIFT CPU requires 23,956 ALUTs for logic and 2×133kB for state and state labels, whereas the

Star-CPU only requires 5756 ALUTs and 133kB for state. Adding dynamic tracking logic for the complex control logic of the CPU introduces substantial delays and reduces the maximum operating frequency of the Star-GLIFT CPU to 55MHz (from 99MHz for the verified Star-CPU). In summary, the verified Star-CPU provides better functionality than the Lease CPU, and static verification in comparison to Star-GLIFT CPU with much lesser area and delay.