Secure Information Flow Verification with Mutable Dependent Types

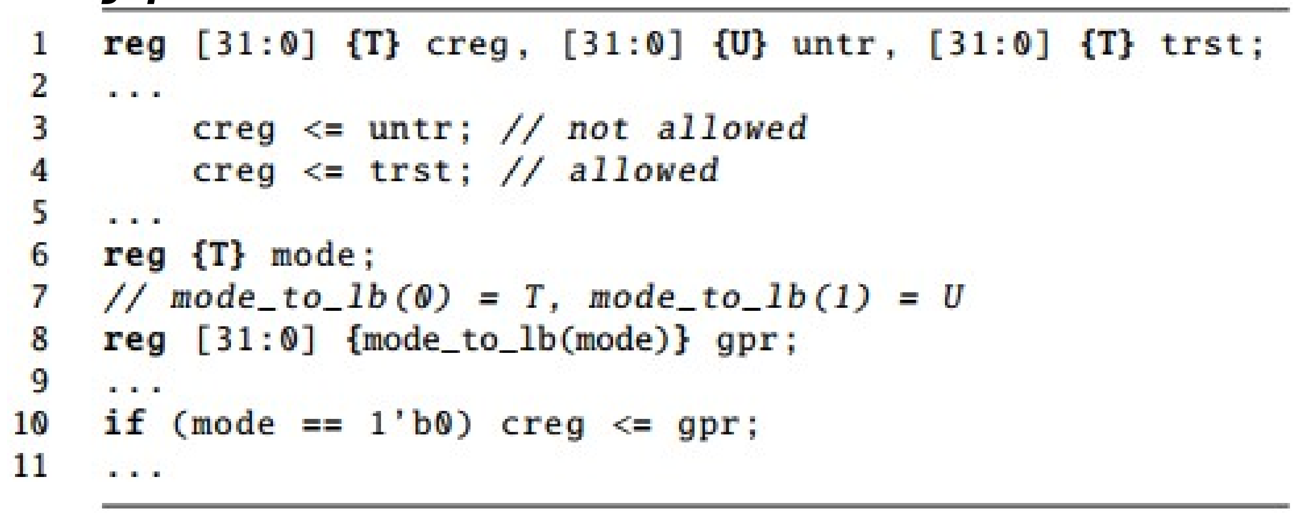
Andrew Ferraiuolo, Weizhe Hua, Andrew C. Myers, G.Edward Suh

Background

- HDL: hardware description languages
- IFC: information flow control
- Type of variables(signal) are annotated with security labels
  - T(trusted); U(untrusted)
- Security policy
  - Flow is allowed from trusted(T) signals to untrusted (U) ones, but flow is not allowed in the other direction
- Type system
  - Untrusted signals do not affect trusted ones

Multiple security levels

- To design efficient hardware, hardware resources con be shared among multiple security levels over time.
- SecVerilog: sharing is permitted through *dependent types*

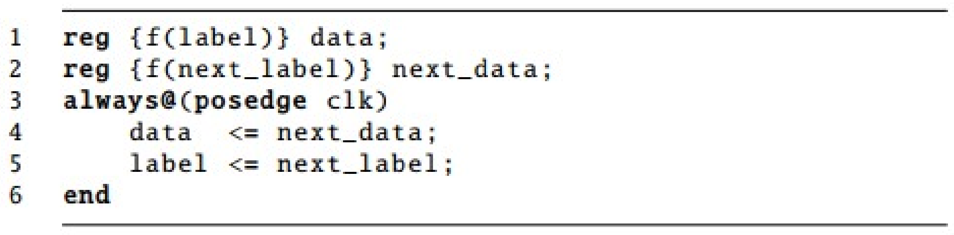


```
reg [31:0] {T} creg, [31:0] {U} untr, [31:0] {T} trst;
...
    creg <= untr; // not allowed
    creg <= trst; // allowed
...
reg {T} mode;
// mode_to_lb(0) = T, mode_to_lb(1) = U
reg [31:0] {mode_to_lb(mode)} gpr;
...
if (mode == 1'b0) creg <= gpr;
...
```

Figure 1: SecVerilog code example.

Problem

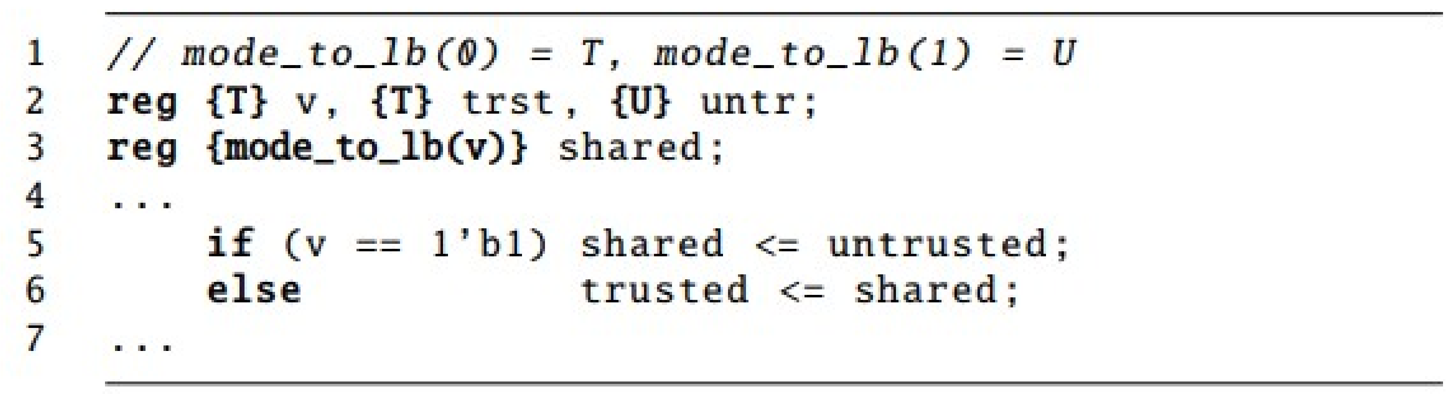
- Both value and label change together
- SecVerilog fails to typecheck this secure code
    - It doesn’t distinguish between updates that occur in the current clock cycle and the next cycle
    - Line 4 is rejected
- SecVerilogLC can detect label is updated and conclude



```
1  reg {f(label)} data;
2  reg {f(next_label)} next_data;
3  always@(posedge clk)
4      data  <= next_data;
5      label <= next_label;
6  end
```

Figure 2: A label propogation example.

Implicit downgrading



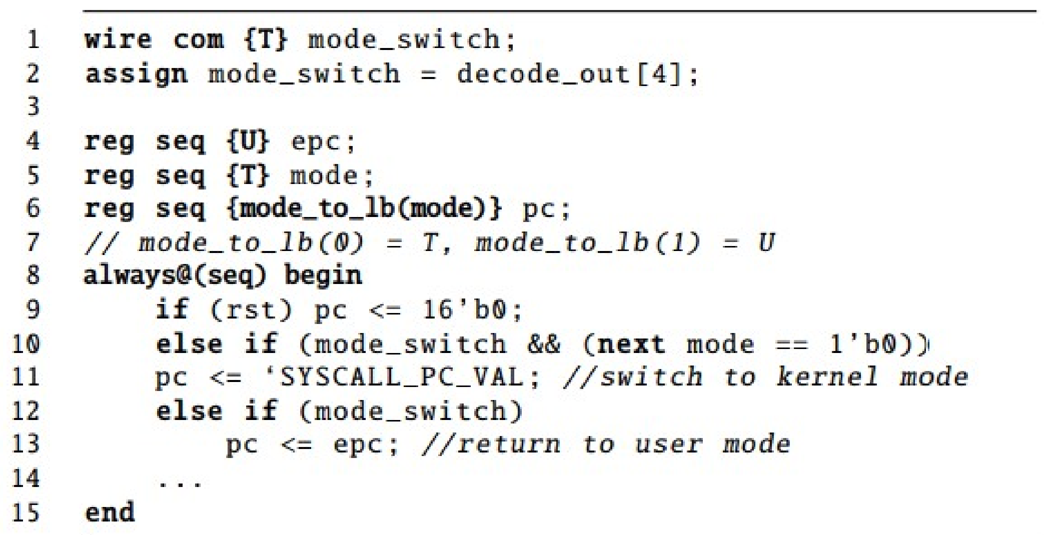
```
// mode_to_lb(0) = T, mode_to_lb(1) = U
reg {T} v, {T} trst, {U} untr;
reg {mode_to_lb(v)} shared;
...
    if (v == 1'b1) shared <= untrusted;
    else           trusted <= shared;
...
```

Figure 3: Implicit downgrading example.

- Solution: dynamic clearing
  - Compiler automatically inserts logic to clear dependently labeled registers whenever the labels of these registers are changed

Approach

- Making the propagation of signals on clock edges
- Introduce a syntax for testing labels for the next clock cycle
- Use the type system to statically establish that registers are securely updated alo



```
wire com {T} mode_switch;
assign mode_switch = decode_out[4];

reg seq {U} epc;
reg seq {T} mode;
reg seq {mode_to_lb(mode)} pc;
// mode_to_lb(0) = T, mode_to_lb(1) = U
always@(seq) begin
    if (rst) pc <= 16'b0;
    else if (mode_switch && (next mode == 1'b0))
    pc <= ‘SYSCALL_PC_VAL; //switch to kernel mode
    else if (mode_switch)
        pc <= epc; //return to user mode
    ...
end
```

Figure 4: PC during mode switches.

Evaluation

- A self designed processor
- Pc <- (F) stage <- untrusted value

Overhead

- Baseline processor :1487 lines
- SecVerilogLC : 257 lines were changed
- 14 kubes were added ti gabdke exokucut diwbgrades
- TSMC 65nm process and target clock :2ns
  - Baseline:29638um^2
  - Labeled design: 29843 um^2