SecSi Product Development:

Techniques for ensuring Secure Silicon applied to open-source Verilog projects

Joe FitzPatrick
@securelyfitz
http://www.securinghardware.com

This document is a preliminary revision.
For the latest version, please see
http://securinghardware.com/secure-opencores

Joe FitzPatrick
@securelyfitz
http://www.securinghardware.com

whoami

- Electrical Engineering education with focus on CS and Infosec
- 8 years doing security research, speed debug, and tool development for CPUs
- Hardware Pen Testing of CPUs
- Security training for functional validators worldwide
- Independent research of low-cost side channel attacks and PCI Express attacks
- Software Exploitation via Hardware Exploits workshop, Black Hat 2014

Joe FitzPatrick
@securelyfitz
joefitz@securinghardware.com

black hat USA 2014

Overview

- What Hardware Security means to ME
- How Hardware is Coded
- OpenRISC
  - JTAG and other state machines
- OpenMSP430
  - Hiding signals and case statements
- Amber
  - Memory aliasing and other tricks
- Summary

black hat USA 2014

Physical Security
is **NOT**
Hardware Security

Supply Chain Security is **NOT** Hardware Security

Secure Boot
is **NOT**
Hardware Security

Secure Firmware
is **NOT**
Hardware Security

However, Hardware Security Depends on:

Physical Security

Supply Chain Security

Software Security

Firmware Security

black hat USA 2014

Coding Hardware
or: Verilog in 2 slides

Programming Languages:

- Make Programs when compiled
- Are broken down into functions/procedures and libraries
- Are a sequential list of instructions (usually)

Hardware Description Languages:

- Make hardware when synthesized
- Are broken down into modules and IP blocks
- Are a set of assignments that all calculate in parallel

Coding Hardware
or: Verilog in 2 slides

C example:

```
int sum(int a,b,c){
  int temp1, temp2;
  temp1=a+b;
  temp2=temp1+c;
  return temp2
}
```

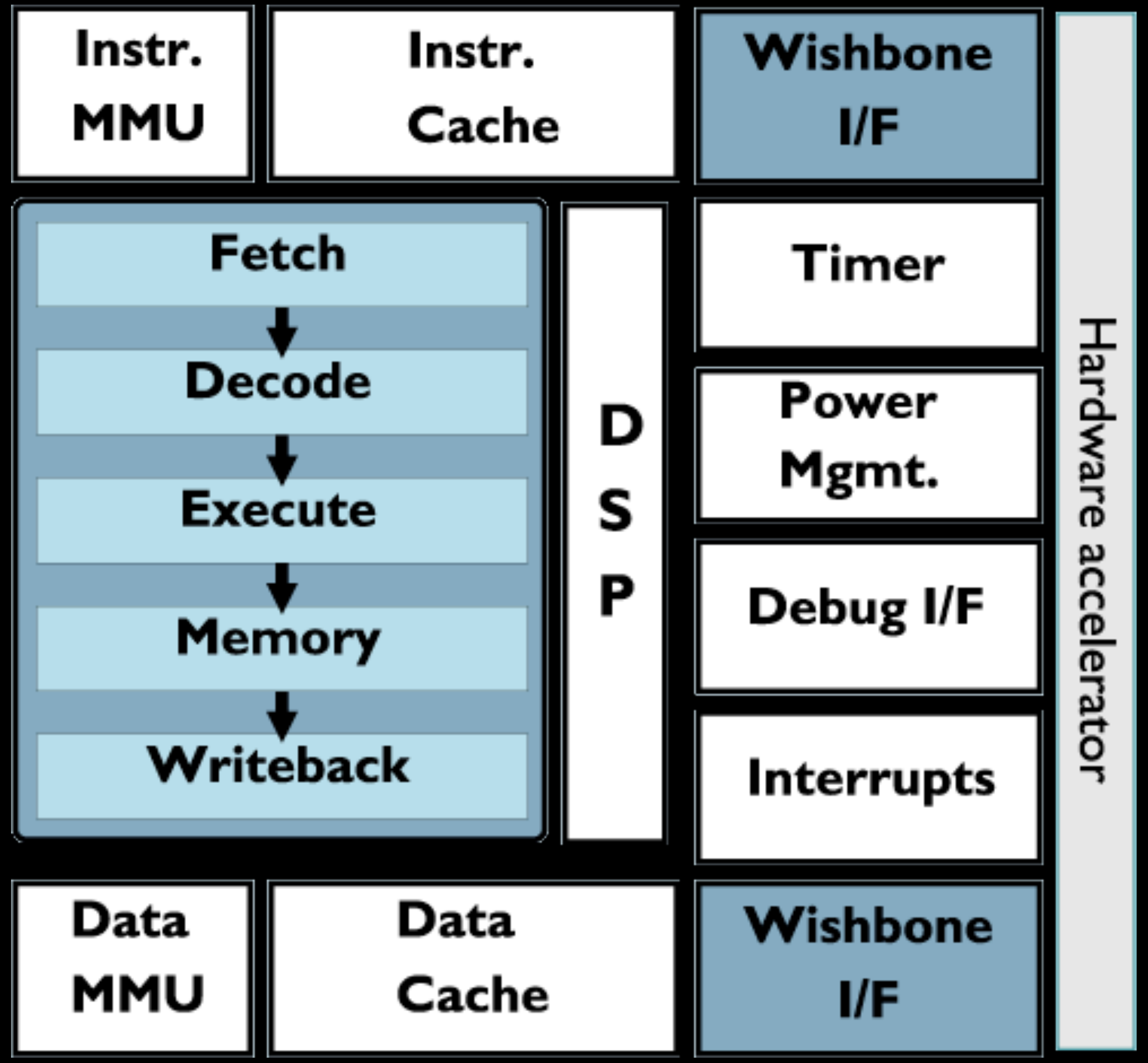
Verilog Example:

```
module sum(a,b,c,sum);
  input[15:0] a,b,c;
  output[15:0] sum
  reg[15:0] temp1,temp2;

  sum=temp2;
  temp2=c+temp1;
  temp1=a+b;
endmodule
```

black hat USA 2014

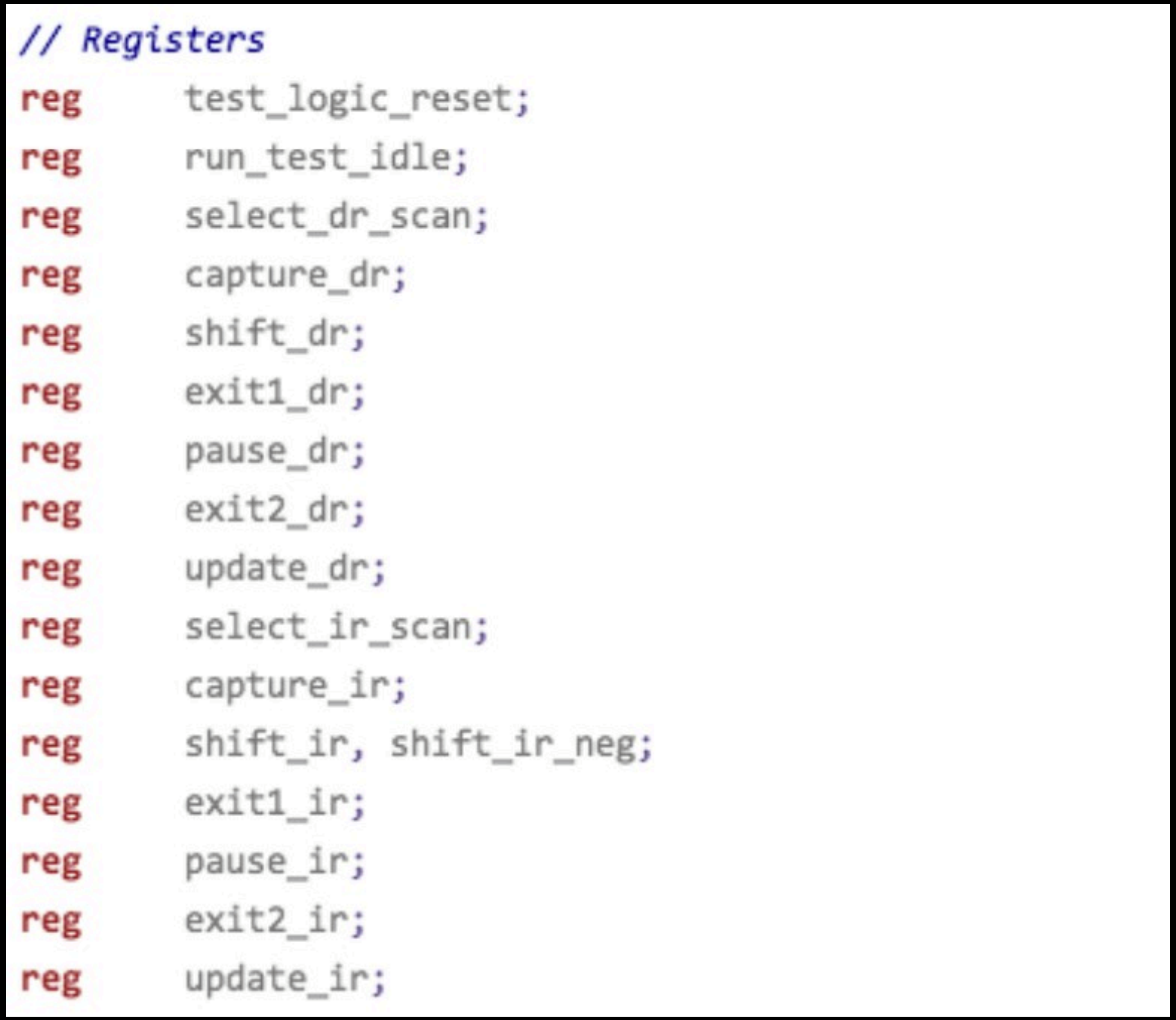
OpenRISC

- Open-Source Verilog CPU
- Linux Kernel Support
- Used in academia and in commercial products
- Targets both FPGAs and ASICs
- Used in the OpenRISC Reference Platform System on Chip (ORPSoC)



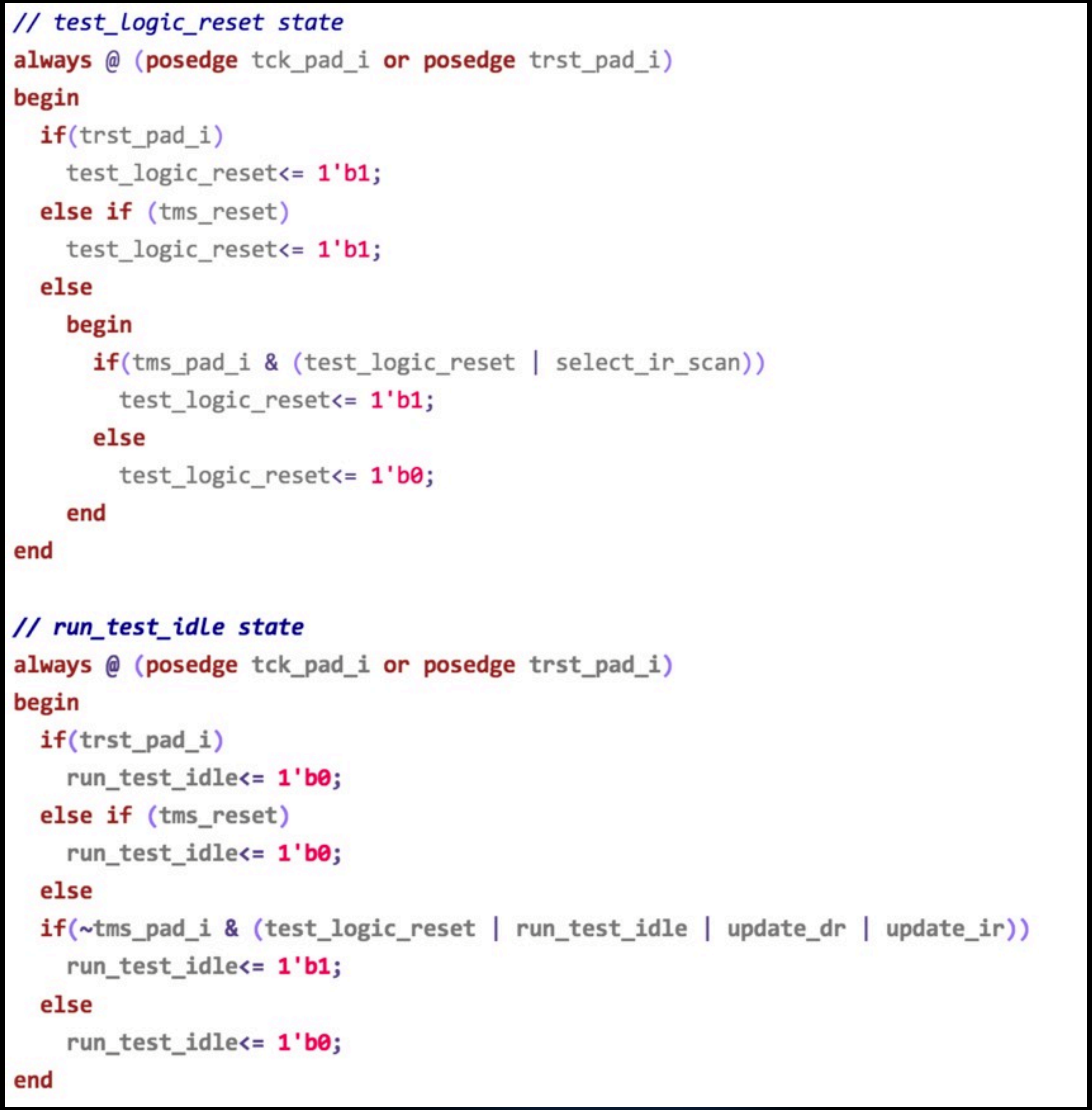
black hat USA 2014

One-Hot and JTAG



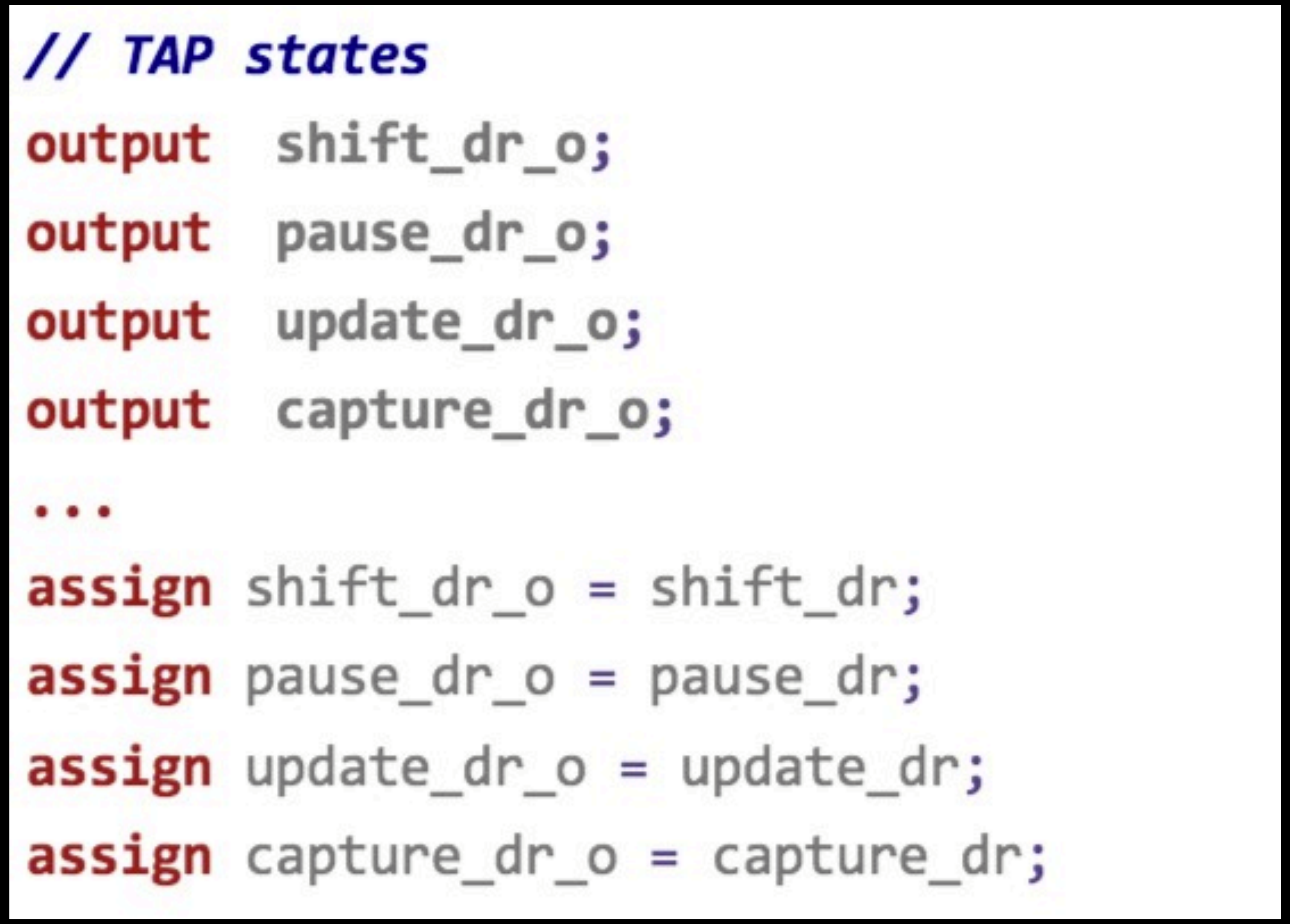
```
// Registers
reg     test_logic_reset;
reg     run_test_idle;
reg     select_dr_scan;
reg     capture_dr;
reg     shift_dr;
reg     exit1_dr;
reg     pause_dr;
reg     exit2_dr;
reg     update_dr;
reg     select_ir_scan;
reg     capture_ir;
reg     shift_ir, shift_ir_neg;
reg     exit1_ir;
reg     pause_ir;
reg     exit2_ir;
reg     update_ir;
```

One-Hot and JTAG

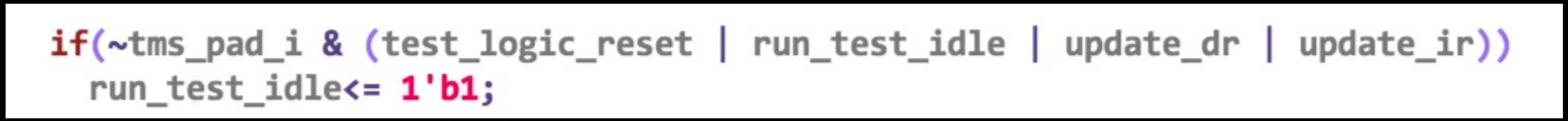


```
// test_logic_reset state
always @ (posedge tck_pad_i or posedge trst_pad_i)
begin
  if(trst_pad_i)
    test_logic_reset<= 1'b1;
  else if (tms_reset)
    test_logic_reset<= 1'b1;
  else
    begin
      if(tms_pad_i & (test_logic_reset | select_ir_scan))
        test_logic_reset<= 1'b1;
      else
        test_logic_reset<= 1'b0;
    end
end

// run_test_idle state
always @ (posedge tck_pad_i or posedge trst_pad_i)
begin
  if(trst_pad_i)
    run_test_idle<= 1'b0;
  else if (tms_reset)
    run_test_idle<= 1'b0;
  else
  if(~tms_pad_i & (test_logic_reset | run_test_idle | update_dr | update_ir))
    run_test_idle<= 1'b1;
  else
    run_test_idle<= 1'b0;
end
```

One-Hot and JTAG

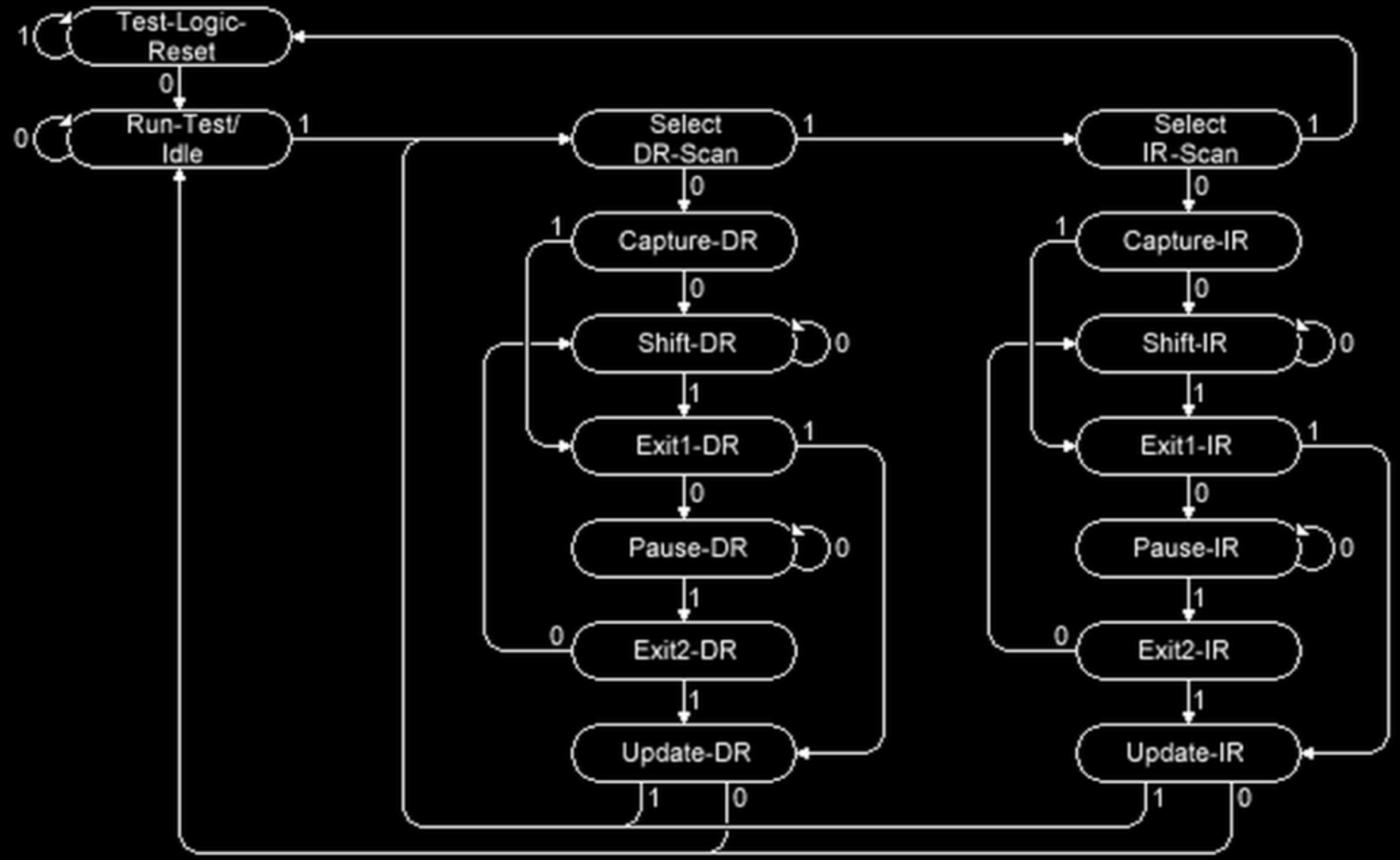


```
// TAP states
output  shift_dr_o;
output  pause_dr_o;
output  update_dr_o;
output  capture_dr_o;
...
assign shift_dr_o = shift_dr;
assign pause_dr_o = pause_dr;
assign update_dr_o = update_dr;
assign capture_dr_o = capture_dr;
```



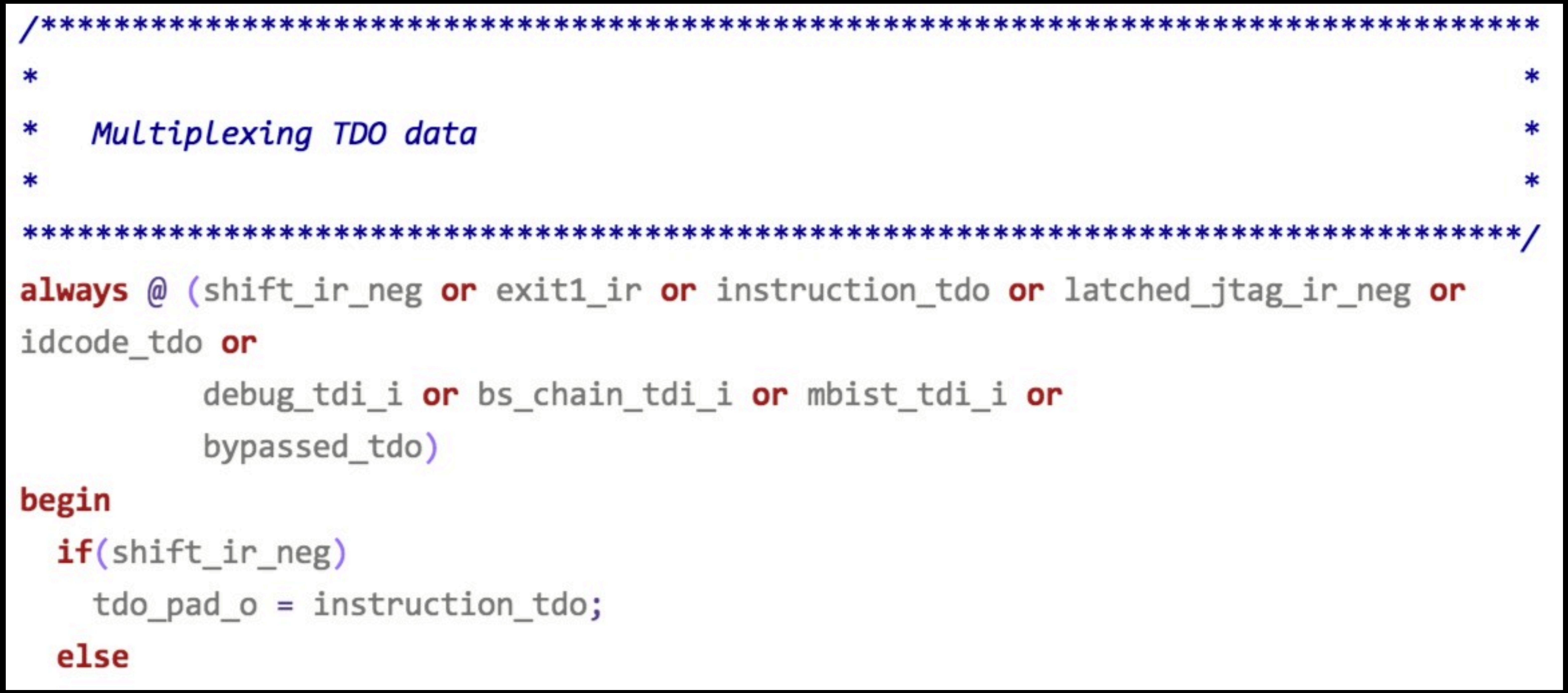
```
if(~tms_pad_i & (test_logic_reset | run_test_idle | update_dr | update_ir))
  run_test_idle<= 1'b1;
```

One-Hot and JTAG



16

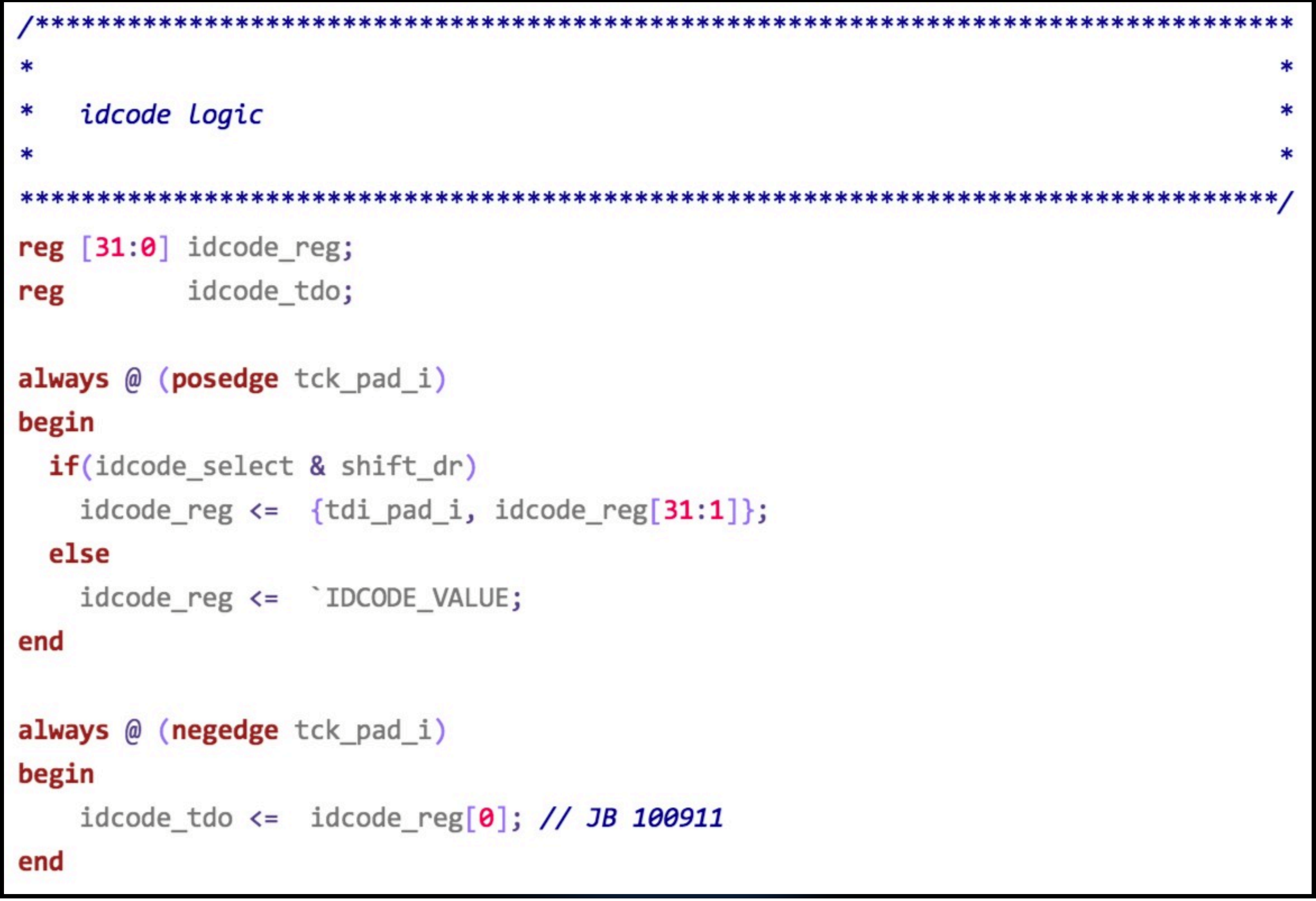
One-Hot and JTAG



```
/************************************************************************
*                                                                      *
*   Multiplexing TDO data                                              *
*                                                                      *
************************************************************************/
always @ (shift_ir_neg or exit1_ir or instruction_tdo or latched_jtag_ir_neg or
idcode_tdo or
          debug_tdi_i or bs_chain_tdi_i or mbist_tdi_i or
          bypassed_tdo)
begin
  if(shift_ir_neg)
    tdo_pad_o = instruction_tdo;
  else
```

black hat USA 2014

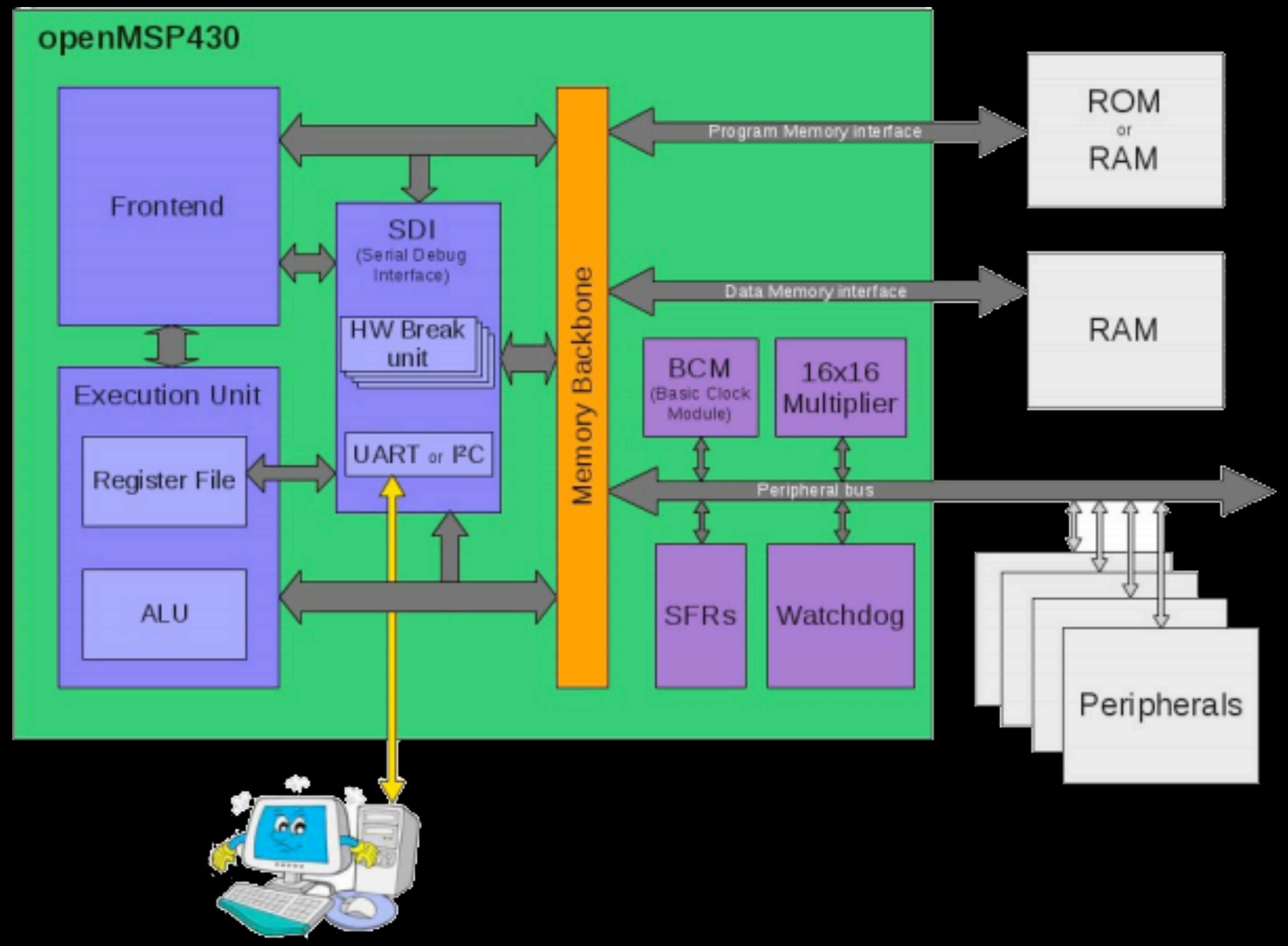
Bonus JTAG Bug!



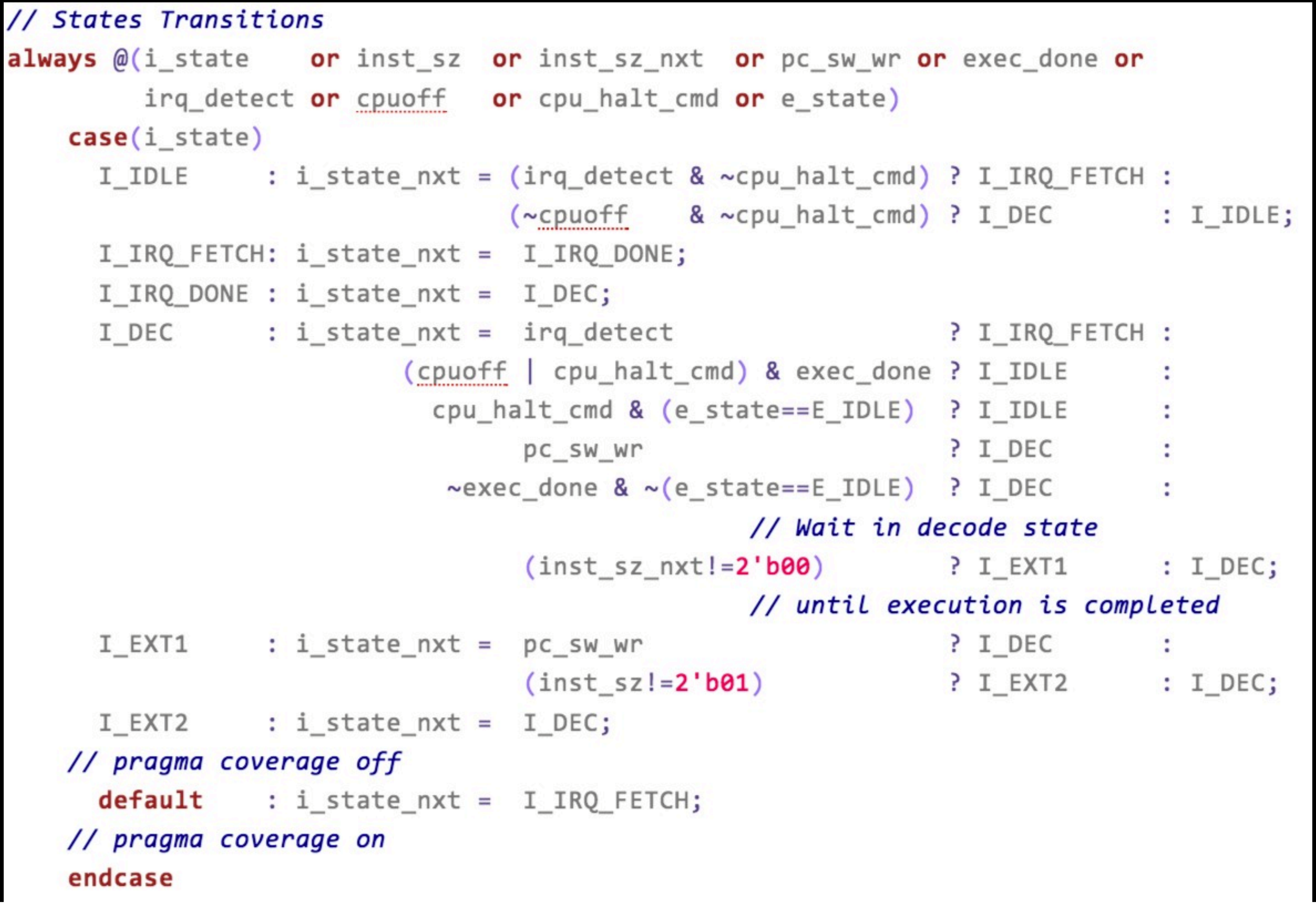
```
/****************************************************************************
*                                                                           *
*   idcode logic                                                            *
*                                                                           *
****************************************************************************/
reg [31:0] idcode_reg;
reg        idcode_tdo;

always @ (posedge tck_pad_i)
begin
  if(idcode_select & shift_dr)
    idcode_reg <=  {tdi_pad_i, idcode_reg[31:1]};
  else
    idcode_reg <=  `IDCODE_VALUE;
end

always @ (negedge tck_pad_i)
begin
    idcode_tdo <=  idcode_reg[0]; // JB 100911
end
```

OpenMSP430

- Open-Source Verilog CPU
- Clone of TI's MSP430
- Binary compatible with all MSP430 toolchains
- Simpler microcontroller-style design than OpenRISC



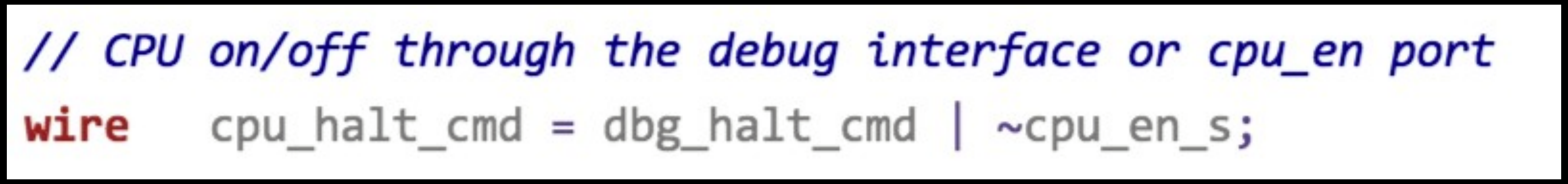
Frontend State Machine Clocking



```
// States Transitions
always @(i_state    or inst_sz  or inst_sz_nxt  or pc_sw_wr or exec_done or
         irq_detect or cpuoff   or cpu_halt_cmd or e_state)
    case(i_state)
      I_IDLE     : i_state_nxt = (irq_detect & ~cpu_halt_cmd) ? I_IRQ_FETCH :
                                 (~cpuoff    & ~cpu_halt_cmd) ? I_DEC       : I_IDLE;
      I_IRQ_FETCH: i_state_nxt =  I_IRQ_DONE;
      I_IRQ_DONE : i_state_nxt =  I_DEC;
      I_DEC      : i_state_nxt =  irq_detect                  ? I_IRQ_FETCH :
                          (cpuoff | cpu_halt_cmd) & exec_done ? I_IDLE      :
                            cpu_halt_cmd & (e_state==E_IDLE)  ? I_IDLE      :
                                  pc_sw_wr                    ? I_DEC       :
                             ~exec_done & ~(e_state==E_IDLE)  ? I_DEC       :
                                                  // Wait in decode state
                                  (inst_sz_nxt!=2'b00)        ? I_EXT1      : I_DEC;
                                                  // until execution is completed
      I_EXT1     : i_state_nxt =  pc_sw_wr                    ? I_DEC       :
                                  (inst_sz!=2'b01)            ? I_EXT2      : I_DEC;
      I_EXT2     : i_state_nxt =  I_DEC;
    // pragma coverage off
      default    : i_state_nxt =  I_IRQ_FETCH;
    // pragma coverage on
    endcase
```

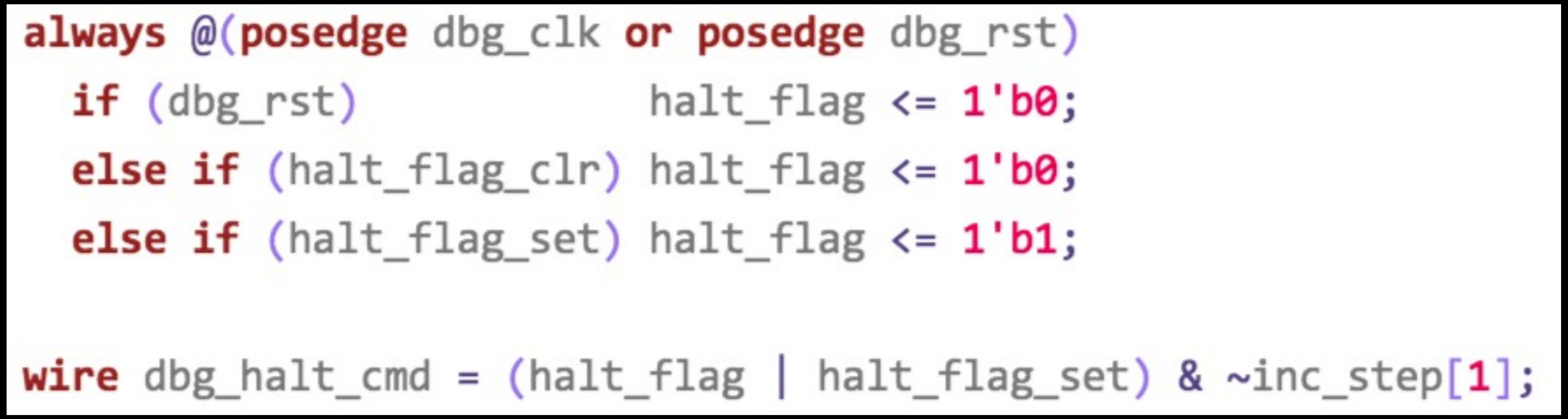
20

Frontend State Machine Clocking



```
// CPU on/off through the debug interface or cpu_en port
wire    cpu_halt_cmd = dbg_halt_cmd | ~cpu_en_s;
```

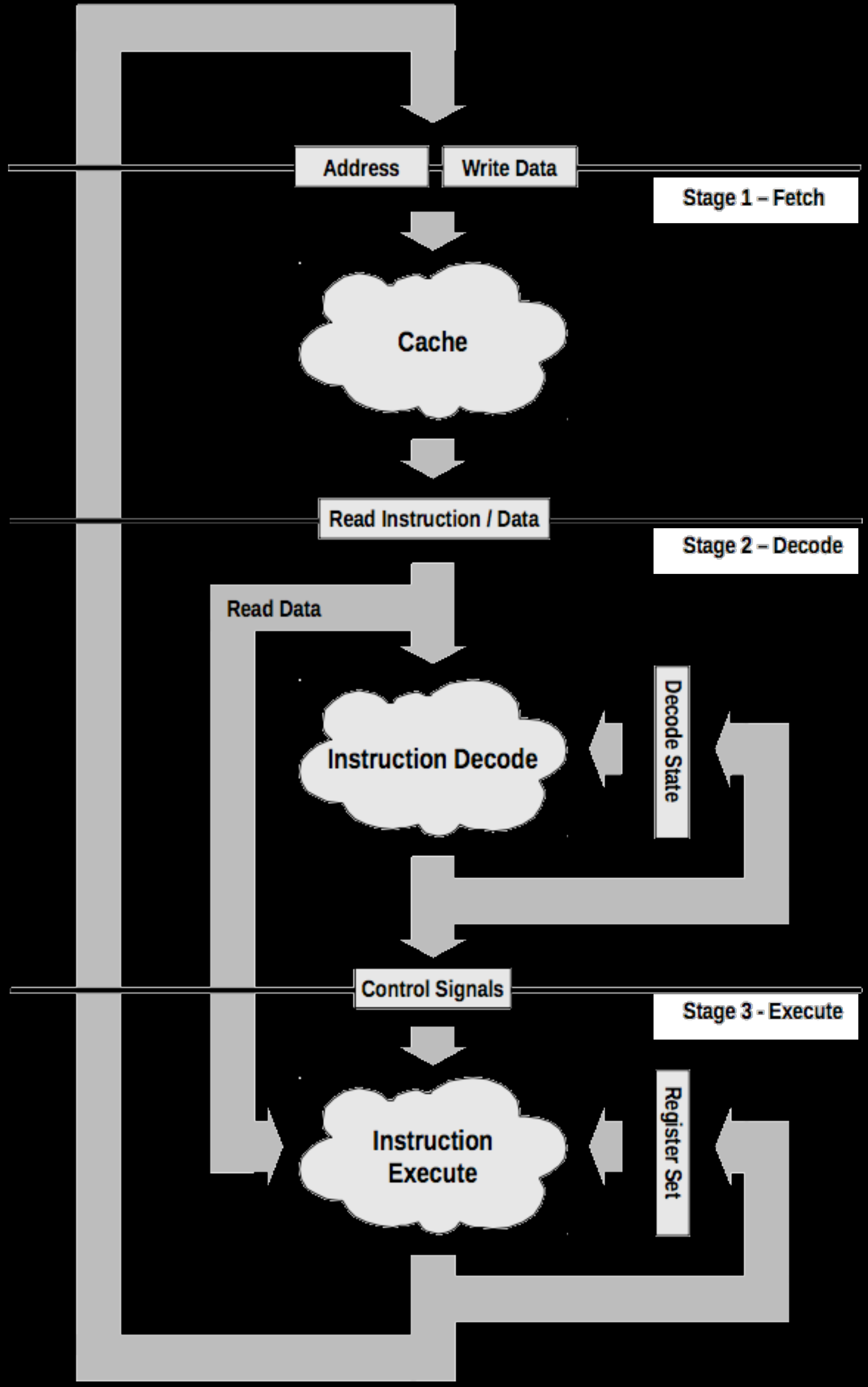
Frontend State Machine Clocking



```
always @(posedge dbg_clk or posedge dbg_rst)
  if (dbg_rst)              halt_flag <= 1'b0;
  else if (halt_flag_clr) halt_flag <= 1'b0;
  else if (halt_flag_set) halt_flag <= 1'b1;

wire dbg_halt_cmd = (halt_flag | halt_flag_set) & ~inc_step[1];
```

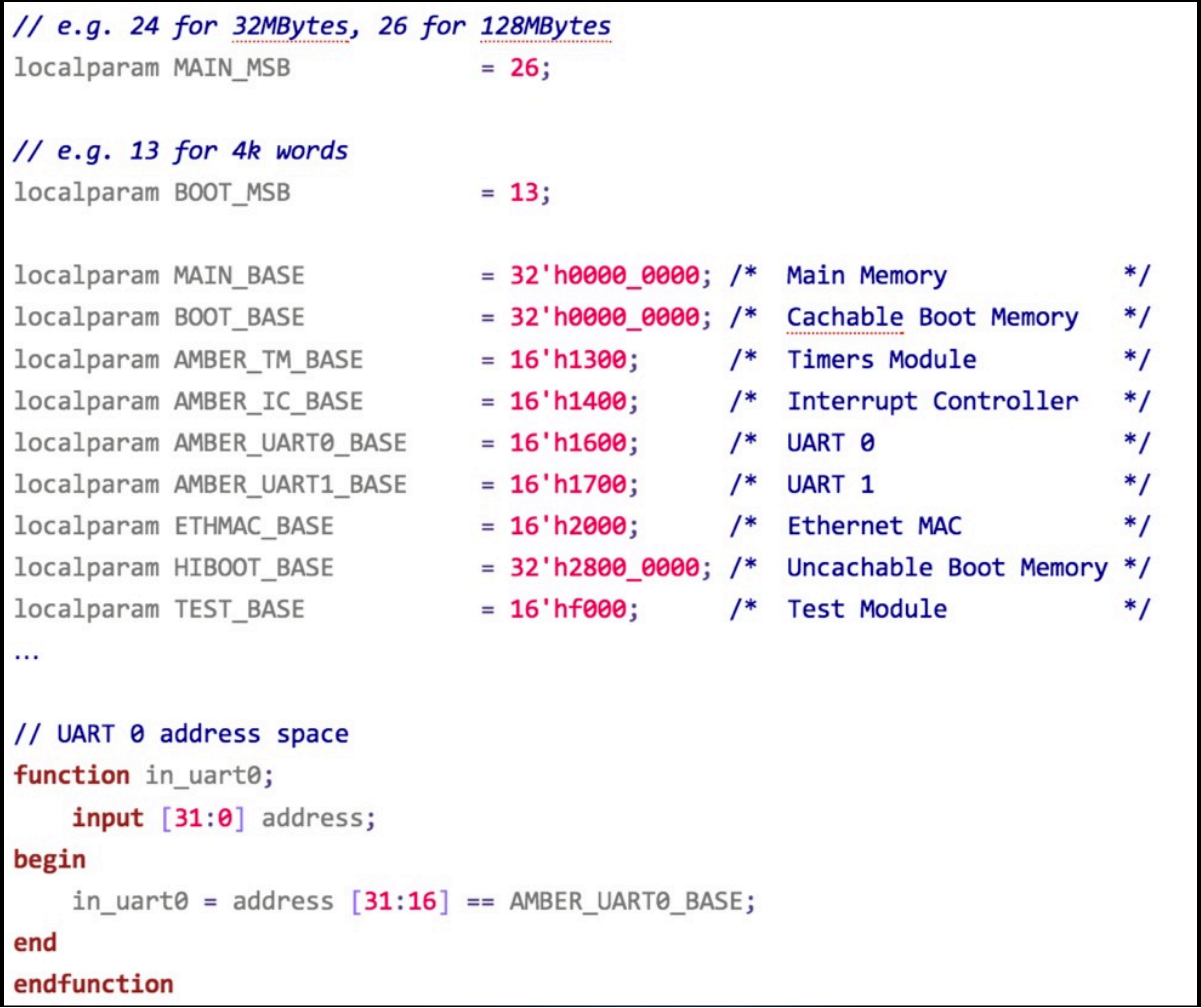
Amber

- Open-Source Verilog CPU
- Implementation of ARMv2
- Supported up to Linux 2.4
- More complex than MSP430, but still no native memory controller like OpenRISC



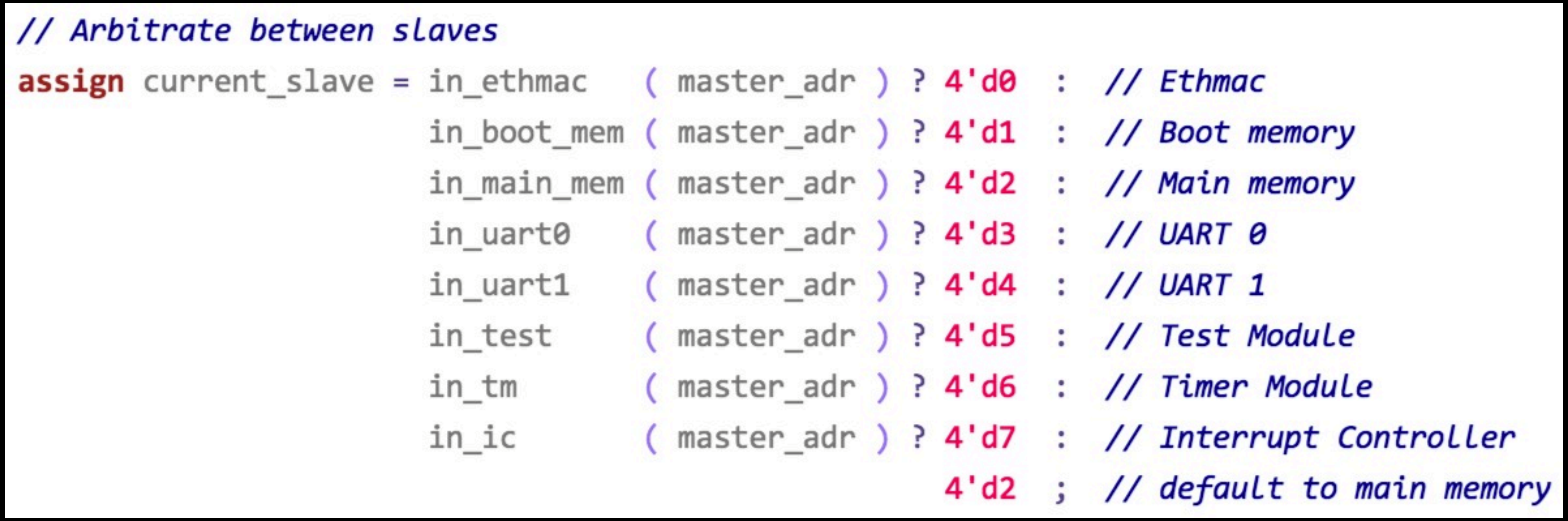
black hat USA 2014

Memory, Addressing, and Caching



```
// e.g. 24 for 32MBytes, 26 for 128MBytes
localparam MAIN_MSB              = 26;

// e.g. 13 for 4k words
localparam BOOT_MSB              = 13;

localparam MAIN_BASE             = 32'h0000_0000; /*  Main Memory            */
localparam BOOT_BASE             = 32'h0000_0000; /*  Cachable Boot Memory   */
localparam AMBER_TM_BASE         = 16'h1300;      /*  Timers Module          */
localparam AMBER_IC_BASE         = 16'h1400;      /*  Interrupt Controller   */
localparam AMBER_UART0_BASE      = 16'h1600;      /*  UART 0                 */
localparam AMBER_UART1_BASE      = 16'h1700;      /*  UART 1                 */
localparam ETHMAC_BASE           = 16'h2000;      /*  Ethernet MAC           */
localparam HIBOOT_BASE           = 32'h2800_0000; /*  Uncachable Boot Memory */
localparam TEST_BASE             = 16'hf000;      /*  Test Module            */
...

// UART 0 address space
function in_uart0;
    input [31:0] address;
begin
    in_uart0 = address [31:16] == AMBER_UART0_BASE;
end
endfunction
```

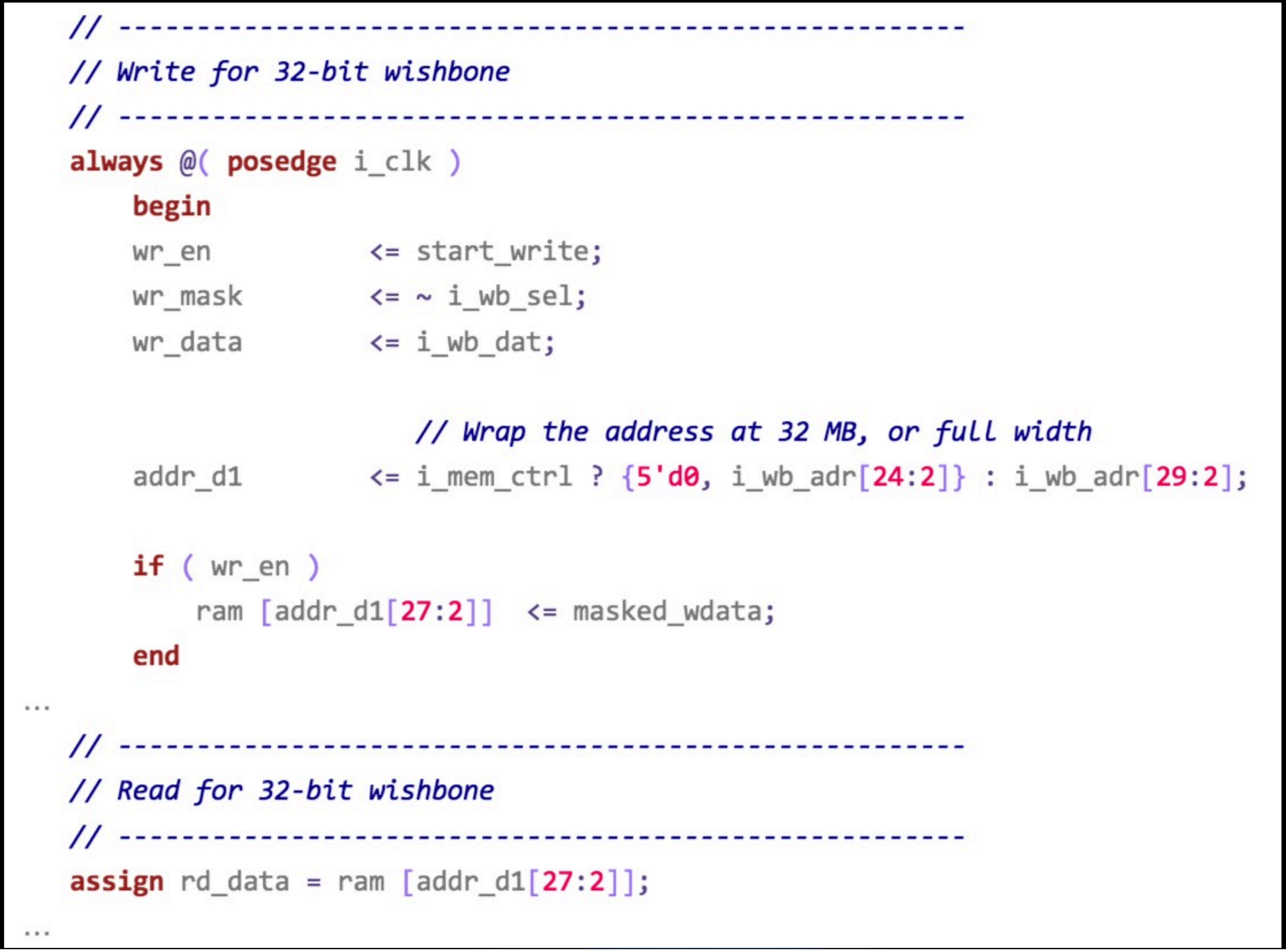
Memory, Addressing, and Caching



```
// Arbitrate between slaves
assign current_slave = in_ethmac   ( master_adr ) ? 4'd0  :  // Ethmac
                       in_boot_mem ( master_adr ) ? 4'd1  :  // Boot memory
                       in_main_mem ( master_adr ) ? 4'd2  :  // Main memory
                       in_uart0    ( master_adr ) ? 4'd3  :  // UART 0
                       in_uart1    ( master_adr ) ? 4'd4  :  // UART 1
                       in_test     ( master_adr ) ? 4'd5  :  // Test Module
                       in_tm       ( master_adr ) ? 4'd6  :  // Timer Module
                       in_ic       ( master_adr ) ? 4'd7  :  // Interrupt Controller
                                                    4'd2  ;  // default to main memory
```

black hat USA 2014

Memory, Addressing, and Caching



```
// ---------------------------------------------------
// Write for 32-bit wishbone
// ---------------------------------------------------
always @( posedge i_clk )
    begin
    wr_en          <= start_write;
    wr_mask        <= ~ i_wb_sel;
    wr_data        <= i_wb_dat;

                      // Wrap the address at 32 MB, or full width
    addr_d1        <= i_mem_ctrl ? {5'd0, i_wb_adr[24:2]} : i_wb_adr[29:2];

    if ( wr_en )
        ram [addr_d1[27:2]]  <= masked_wdata;
    end
...
// ---------------------------------------------------
// Read for 32-bit wishbone
// ---------------------------------------------------
assign rd_data = ram [addr_d1[27:2]];
...
```

26

Summary

- Unchecked State Encoding
- Untested State Transitions
- Permissive Sensitivity Lists
- Obscured Signal Combinations
- Aliased, Overlapped, or Mis-prioritized Memory Maps

black hat USA 2014

black hat USA 2014

SecSi Product Development:

Techniques for ensuring Secure Silicon applied to open-source Verilog projects

Questions?

Joe FitzPatrick
@securelyfitz
http://www.securinghardware.com