# Theoretical Patchability Quantification for IP-level Hardware Patching Designs

**ASP-DAC 2024**

Presented by: Wei-Kai Liu

Wei-Kai Liu, Benjamin Tan, Jason Fung, and Krishnendu Chakrabarty

UNIVERSITY OF CALGARY

Duke UNIVERSITY

ASU Arizona State University

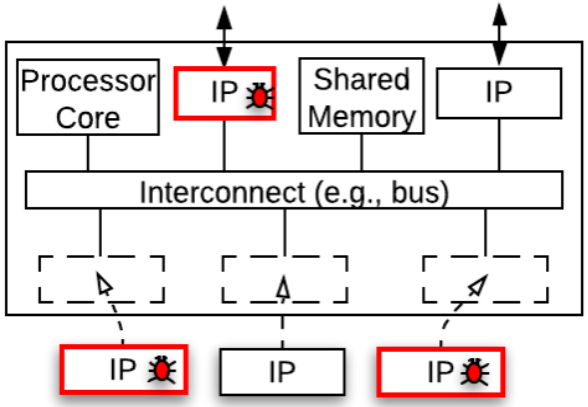
# Outline

- Background & Motivation
- Patchability Quantification Formulation
- Case Study Analysis
- Conclusions

# Patching Hardware -- Overview

- Survivability of System-on-Chip designs
  - Integrate many different IPs, processors,
  - Different vendors, use cases

**Issues may manifest after deployment!**

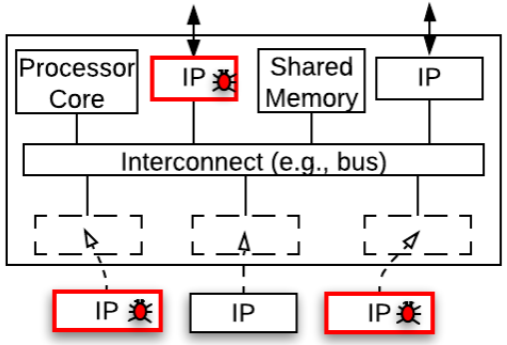


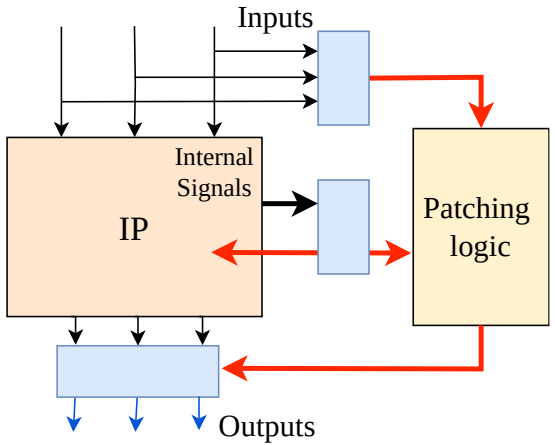

- Patches for security issues
  - Software, firmware, hardware

- Software and firmware patches are not enough
  - Not real-time fixes
  - Many data are non-reprogrammable

# Patching Hardware -- Overview

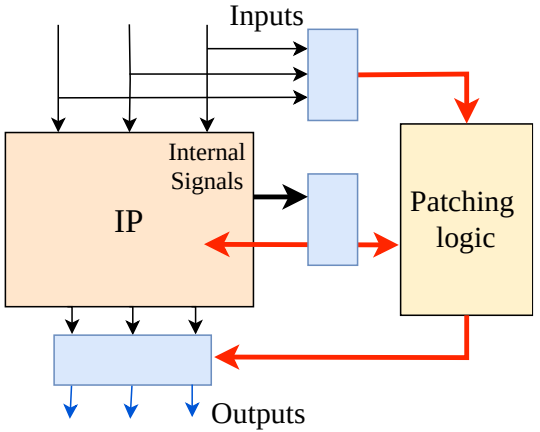
- Patching hardware
  - Architecture for post-deployment patching
- IP-level patching design
  - Observe IP's behavior
    - Monitoring inputs/outputs/internal signals
  - Control IP's behavior
    - Override IP's signals if misbehavior is identified

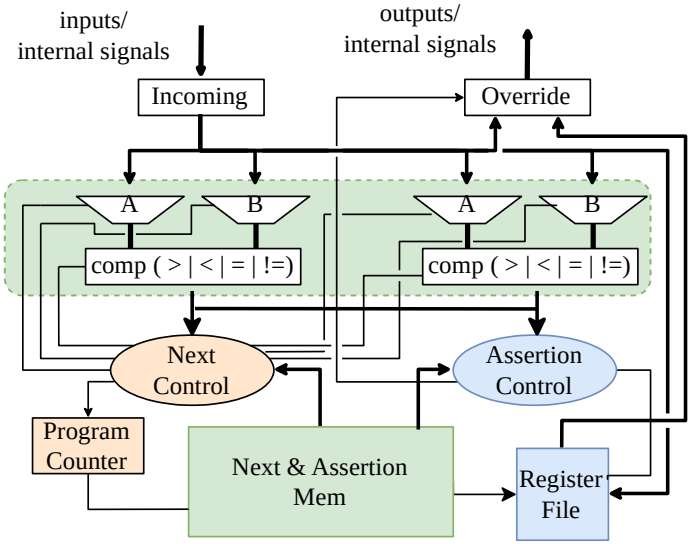


# Research Motivation

- Patching hardware is generated based on heuristics
  - What signals to monitor/control?
  - How many signals is enough?
- Tradeoff: resource investment and what patching hardware can do

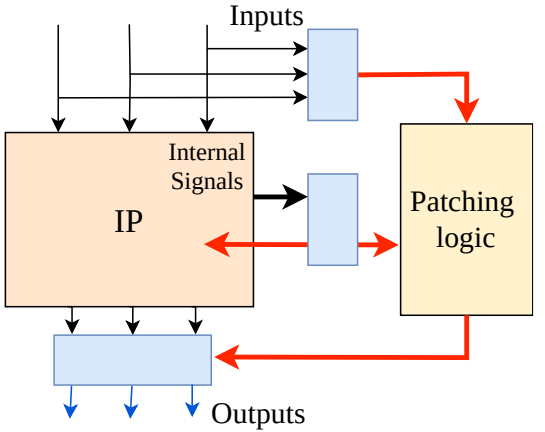


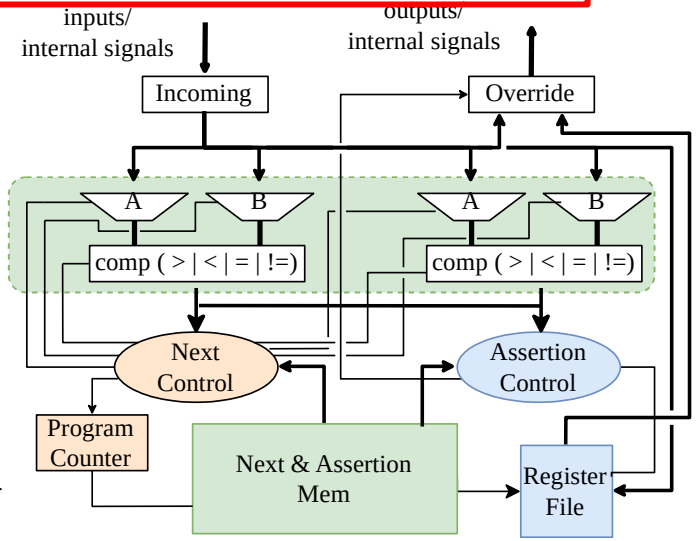



# Research Motivation

- Patching hardware is generated based on heuristics
  - What signals to monitor/control?
  - How many signals is enough?
- Tradeoff: resource investment and what patching hardware can do

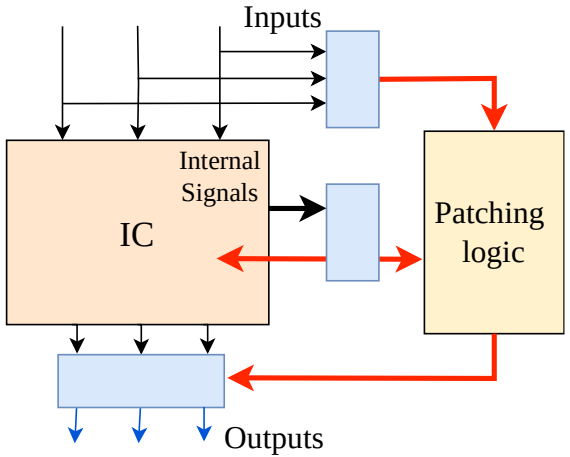
**How to quantify?**



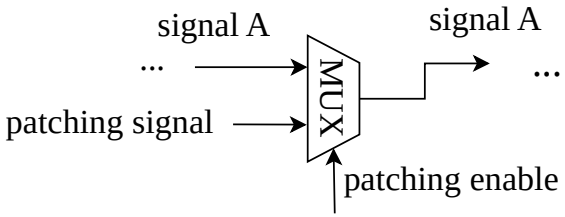
# Definition of Patchability

- Patchability
  - The ability of a patching hardware to patch
  - Observability
    - The ability of a patching design to observe an IP
    - Bugs identification
  - Controllability
    - The ability of a patching design to control an IP
    - Bugs correction
- Quantification
  - How many bits of a design can be observed/controlled by the patching design

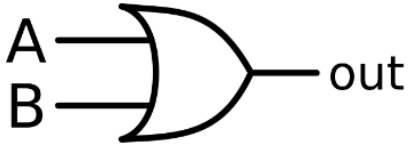


# Patchability -- Controllability

- Patching cell
  - Replace the original signal with the patching signal
- The patched signal then becomes ”fully controllable”
  - Controllability is $n$ for an $n$-bit signal
- Controllability measurement
  - Need to consider the relationships between signals
  - E.g., Out is controllable if A and B are controlled

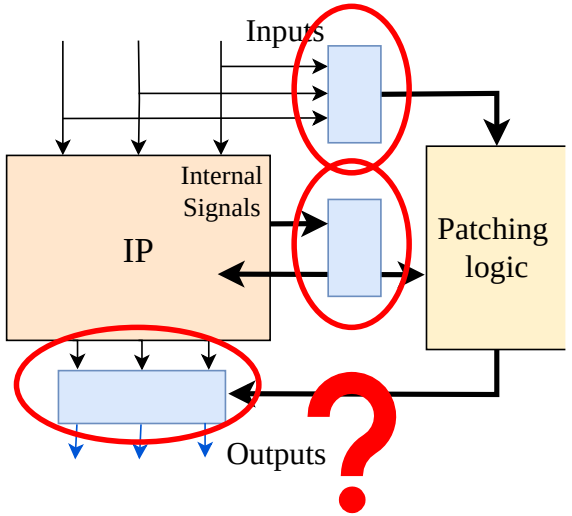



Patching cell



# Research Questions

- Our goal
  - Assist system integrators in designing the patching architecture
- How to achieve the highest patchability under the resource limit?
  - What signals should be controlled/monitored in different scenarios?



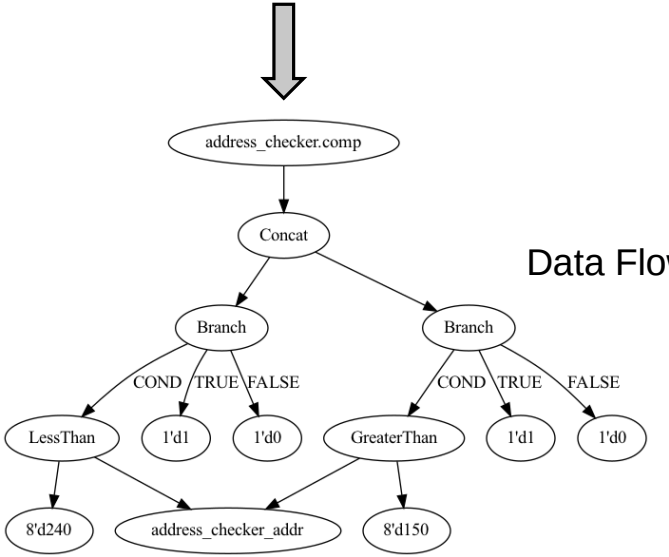
# Outline

- Background & Motivation
- Patchability Quantification Formulation
- Case Study Analysis
- Conclusions

# Patchability Quantification Algorithm

```
input [7:0] addr;
assign comp[0] = (addr > 8'h96);
assign comp[1] = (addr < 8'hF0);
```

RTL Design




Data Flow

Quantification Functions

$$S_{comp} = \frac{\frac{\frac{S_{addr}}{8} + 0}{2} + \frac{\frac{S_{addr}}{8} + 0}{2}}{2}$$

Patchability Score Function

Patching Configurations

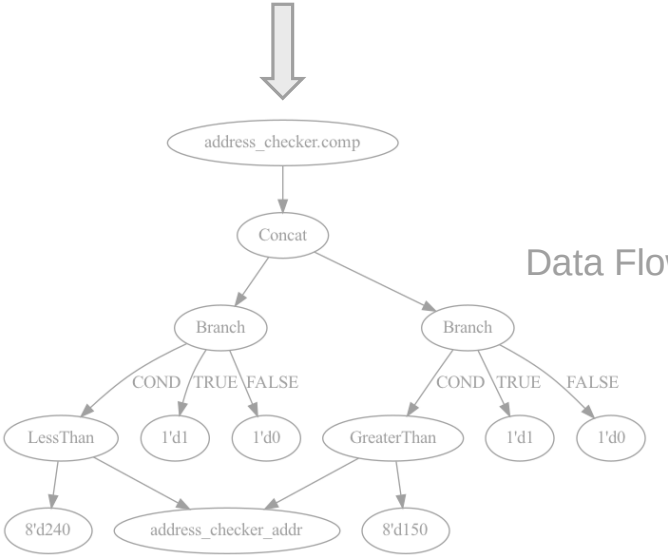
$S_{comp} = 2$

Result

# Patchability Quantification Algorithm

```
input [7:0] addr;
assign comp[0] = (addr > 8'h96);
assign comp[1] = (addr < 8'hF0);
```

RTL Design




Data Flow

Quantification Functions

$$S_{comp} = \frac{\frac{\frac{S_{addr}}{8} + 0}{2} + \frac{\frac{S_{addr}}{8} + 0}{2}}{2}$$

Patchability Score Function

Patching Configurations

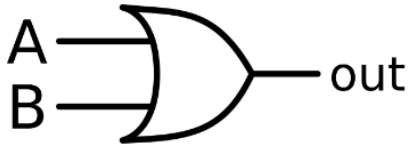
$$S_{comp} = 2$$

Result

# Patchability Quantification for RTL Operations

- Quantifying controllability for OR operation
  - $S_A$ and $S_B$: controllability score of A and B
    - Probability of A and B are controllable (1-bit signals)
  - Out is fully controllable if
    - A and B are both fully controllable → $S_A \times S_B$
    - A or B is fully controllable, and the other is a **non-controlling** value

      $\rightarrow (1 - S_A) \times S_B \times \frac{1}{2} + S_A \times (1 - S_B) \times \frac{1}{2}$
      - Assumption: each bit has equal prob. to be 0 and 1
  - $S_{out} = S_A \times S_B + (1 - S_A) \times S_B \times \frac{1}{2} + S_A \times (1 - S_B) \times \frac{1}{2} = \frac{S_A + S_B}{2}$



# Patchability Quantification for RTL Operations

- Conditional statements
  - A = (B) ? C : D
  - Assumption: assign the signal with higher score to A
  - $S_A = S_B \times \max(S_C, S_D) + (1 - S_B) \times \frac{(S_C + S_D)}{2}$

- Comparisons
  - E.g., Greater than (for 1-bit signals)
    - $sig_1 > sig_2 \equiv sig_1 \&\&! sig_2$
  - For multi-bit comparisons
    - Compare each bit and combine the results

| Operator | Resulting PC score |
|---|---|
| ASSIGN/NOT | $S_A$ |
| SHIFT | $S_A \times \frac{n-1}{n}$ |
| OR/NOR | $\frac{S_A+S_B}{2}$ |
| AND/NAND | $\frac{S_A+S_B}{2}$ |
| XOR/XNOR | $\frac{S_A+S_B}{2}$ |
| CONCATENATE | $S_A + S_B$ |

| Operator | Resulting PC score |
|---|---|
| == | $\frac{S_1+S_2}{2}$ |
| != | $\frac{S_1+S_2}{2}$ |
| > | $\frac{S_1+S_2}{2}$ |
| < | $\frac{S_1+S_2}{2}$ |
| ≥ | $\frac{S_1+S_2}{2}$ |
| ≤ | $\frac{S_1+S_2}{2}$ |

# Outline

- Background & Motivation
- Patchability Quantification Formulation
- Case Study Analysis
- Conclusions

# Case Study

- In: patching configurations
- Out: the resulting controllability scores
- Investment: sum of the In column
- Output score: sum of the Out column
- Normalized score: normalize the score of each signal by their widths and average them

| Signal Name | All Fully Patchable | Greedy In | Greedy Out | Proposed In | Proposed Out |
|---|---|---|---|---|---|
| rst_ni | 1 | 1 | 1 | 1 | 1 |
| jtag_unlock | 1 | 1 | 1 | 1 | 1 |
| rst_9 | 1 | 1 | 1 | 1 | 1 |
| we | 1 | 1 | 1 | 1 | 1 |
| address | 64 | 64 | 64 | 64 | 64 |
| wdata | 32 | 32 | 32 | 32 | 32 |
| reglk_ctrl_i | 8 | 8 | 8 | 8 | 8 |
| en_acct | 1 | 1 | 1 | 1 | 1 |
| acct_ctrl_i | 1 | 1 | 1 | 1 | 1 |
| reglk_mem[0] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[1] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[2] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[3] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[4] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[5] | 32 | 32 | 32 | 0 | 24 |
| en | 1 | 1 | 1 | 0 | 1 |
| reglk_ctrl | 16 | 16 | 16 | 0 | 8 |
| rdata | 32 | 0 | 32 | 0 | 18 |
| reglk_ctrl_o | 112 | 0 | 112 | 0 | 112 |
| Investment (bits) | 463 | 319 | | 110 | |
| Output Score (bits) | 463 | 463 | | 393 | |
| Normalized Score | 1 | 1 | | 0.9 | |

Greedy option selects every input/internal signals for better controllability

# Case Study

- In: patching configurations
- Out: the resulting controllability scores
- Investment: sum of the In column
- Output score: sum of the Out column
- Normalized score: normalize the score of each signal by their widths and average them

| Signal Name | All Fully Patchable | Greedy In | Greedy Out | Proposed In | Proposed Out |
|---|---|---|---|---|---|
| rst_ni | 1 | 1 | 1 | 1 | 1 |
| jtag_unlock | 1 | 1 | 1 | 1 | 1 |
| rst_9 | 1 | 1 | 1 | 1 | 1 |
| we | 1 | 1 | 1 | 1 | 1 |
| address | 64 | 64 | 64 | 64 | 64 |
| wdata | 32 | 32 | 32 | 32 | 32 |
| reglk_ctrl_i | 8 | 8 | 8 | 8 | 8 |
| en_acct | 1 | 1 | 1 | 1 | 1 |
| acct_ctrl_i | 1 | 1 | 1 | 1 | 1 |
| reglk_mem[0] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[1] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[2] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[3] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[4] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[5] | 32 | 32 | 32 | 0 | 24 |
| en | 1 | 1 | 1 | 0 | 1 |
| reglk_ctrl | 16 | 16 | 16 | 0 | 8 |
| rdata | 32 | 0 | 32 | 0 | 18 |
| reglk_ctrl_o | 112 | 0 | 112 | 0 | 112 |
| Investment (bits) | 463 | 319 | | 110 | |
| Output Score (bits) | 463 | 463 | | 393 | |
| Normalized Score | 1 | 1 | | 0.9 | |

Our proposed option selects signals based on the patchability quantification algorithm

```
assign reglk_ctrl_o = {reglk_mem[5], reglk_mem[4],
reglk_mem[3], reglk_mem[2], reglk_mem[1], reglk_mem[0]};
assign reglk_ctrl = reglk_ctrl_i;
assign en = en_acct ? acct_ctrl_i: 0;
always @(posedge clk_i) begin
  if(rst_ni && ~jtag_unlock && ~rst_9) begin
    for (j=0; j < 6; j=j+1) begin
      reglk_mem[j] <= 'h0;
    end
  end
  else if(en && we) begin
    case(address[7:3])
 0: reglk_mem[0] <= reglk_ctrl[1] ? reglk_mem[0] : wdata;
 1: reglk_mem[1] <= reglk_ctrl[1] ? reglk_mem[1] : wdata;
 2: reglk_mem[2] <= reglk_ctrl[1] ? reglk_mem[3] : wdata;
 3: reglk_mem[3] <= reglk_ctrl[1] ? reglk_mem[3] : wdata;
 4: reglk_mem[4] <= reglk_ctrl[1] ? reglk_mem[4] : wdata;
 5: reglk_mem[5] <= reglk_ctrl[1] ? reglk_mem[5] : wdata;
    endcase
  end
end
always @(*) begin
  rdata = 64'b0;
  if (en) begin
    case(address[7:3])
      0: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[0];
      1: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[1];
      2: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[2];
      3: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[3];
      4: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[4];
      5: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[5];
    endcase
  end
end
```

Width

| Signal Name | All Fully Patchable | Greedy In | Greedy Out | Proposed In | Proposed Out |
|---|---|---|---|---|---|
| rst_ni | 1 | 1 | 1 | 1 | 1 |
| jtag_unlock | 1 | 1 | 1 | 1 | 1 |
| rst_9 | 1 | 1 | 1 | 1 | 1 |
| we | 1 | 1 | 1 | 1 | 1 |
| address | 64 | 64 | 64 | 64 | 64 |
| wdata | 32 | 32 | 32 | 32 | 32 |
| reglk_ctrl_i | 8 | 8 | 8 | 8 | 8 |
| en_acct | 1 | 1 | 1 | 1 | 1 |
| acct_ctrl_i | 1 | 1 | 1 | 1 | 1 |
| reglk_mem[0] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[1] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[2] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[3] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[4] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[5] | 32 | 32 | 32 | 0 | 24 |
| en | 1 | 1 | 1 | 0 | 1 |
| reglk_ctrl | 16 | 16 | 16 | 0 | 8 |
| rdata | 32 | 0 | 32 | 0 | 18 |
| reglk_ctrl_o | 112 | 0 | 112 | 0 | 112 |
| Investment (bits) | 463 | 319 | | 110 | |
| Output Score (bits) | 463 | 463 | | 393 | |
| Normalized Score | 1 | 1 | | 0.9 | |

```
assign reglk_ctrl_o = {reglk_mem[5], reglk_mem[4],
reglk_mem[3], reglk_mem[2], reglk_mem[1], reglk_mem[0]};
assign reglk_ctrl = reglk_ctrl_i;
assign en = en_acct ? acct_ctrl_i: 0;
always @(posedge clk_i) begin
  if(rst_ni && ~jtag_unlock && ~rst_9) begin
    for (j=0; j < 6; j=j+1) begin
      reglk_mem[j] <= 'h0;
    end
  end
  else if(en && we) begin
    case(address[7:3])
 0: reglk_mem[0] <= reglk_ctrl[1] ? reglk_mem[0] : wdata;
 1: reglk_mem[1] <= reglk_ctrl[1] ? reglk_mem[1] : wdata;
 2: reglk_mem[2] <= reglk_ctrl[1] ? reglk_mem[3] : wdata;
 3: reglk_mem[3] <= reglk_ctrl[1] ? reglk_mem[3] : wdata;
 4: reglk_mem[4] <= reglk_ctrl[1] ? reglk_mem[4] : wdata;
 5: reglk_mem[5] <= reglk_ctrl[1] ? reglk_mem[5] : wdata;
    endcase
  end
end
always @(*) begin
  rdata = 64'b0;
  if (en) begin
    case(address[7:3])
      0: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[0];
      1: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[1];
      2: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[2];
      3: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[3];
      4: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[4];
      5: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[5];
    endcase
  end
end
```

**Mismatched!**

Width

| Signal Name | All Fully Patchable | Greedy In | Greedy Out | Proposed In | Proposed Out |
|---|---|---|---|---|---|
| rst_ni | 1 | 1 | 1 | 1 | 1 |
| jtag_unlock | 1 | 1 | 1 | 1 | 1 |
| rst_9 | 1 | 1 | 1 | 1 | 1 |
| we | 1 | 1 | 1 | 1 | 1 |
| address | 64 | 64 | 64 | 64 | 64 |
| wdata | 32 | 32 | 32 | 32 | 32 |
| reglk_ctrl_i | 8 | 8 | 8 | 8 | 8 |
| en_acct | 1 | 1 | 1 | 1 | 1 |
| acct_ctrl_i | 1 | 1 | 1 | 1 | 1 |
| reglk_mem[0] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[1] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[2] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[3] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[4] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[5] | 32 | 32 | 32 | 0 | 24 |
| en | 1 | 1 | 1 | 0 | 1 |
| reglk_ctrl | 16 | 16 | 16 | 0 | 8 |
| rdata | 32 | 0 | 32 | 0 | 18 |
| reglk_ctrl_o | 112 | 0 | 112 | 0 | 112 |
| Investment (bits) | 463 | 319 | | 110 | |
| Output Score (bits) | 463 | 463 | | 393 | |
| Normalized Score | 1 | 1 | | 0.9 | |

```
assign reglk_ctrl_o = {reglk_mem[5], reglk_mem[4],
reglk_mem[3], reglk_mem[2], reglk_mem[1], reglk_mem[0]};
assign reglk_ctrl = reglk_ctrl_i;
assign en = en_acct ? acct_ctrl_i: 0;
always @(posedge clk_i) begin
  if(rst_ni && ~jtag_unlock && ~rst_9) begin
    for (j=0; j < 6; j=j+1) begin
      reglk_mem[j] <= 'h0;
    end
  end
  else if(en && we) begin
    case(address[7:3])
 0: reglk_mem[0] <= reglk_ctrl[1] ? reglk_mem[0] : wdata;
 1: reglk_mem[1] <= reglk_ctrl[1] ? reglk_mem[1] : wdata;
 2: reglk_mem[2] <= reglk_ctrl[1] ? reglk_mem[3] : wdata;
 3: reglk_mem[3] <= reglk_ctrl[1] ? reglk_mem[3] : wdata;
 4: reglk_mem[4] <= reglk_ctrl[1] ? reglk_mem[4] : wdata;
 5: reglk_mem[5] <= reglk_ctrl[1] ? reglk_mem[5] : wdata;
    endcase
  end
end
always @(*) begin
  rdata = 64'b0;
  if (en) begin
    case(address[7:3])
      0: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[0];
      1: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[1];
      2: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[2];
      3: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[3];
      4: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[4];
      5: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[5];
    endcase
  end
end
```

Width

| Signal Name | All Fully Patchable | Greedy In | Greedy Out | Proposed In | Proposed Out |
|---|---|---|---|---|---|
| rst_ni | 1 | 1 | 1 | 1 | 1 |
| jtag_unlock | 1 | 1 | 1 | 1 | 1 |
| rst_9 | 1 | 1 | 1 | 1 | 1 |
| we | 1 | 1 | 1 | 1 | 1 |
| address | 64 | 64 | 64 | 64 | 64 |
| wdata | 32 | 32 | 32 | 32 | 32 |
| reglk_ctrl_i | 8 | 8 | 8 | 8 | 8 |
| en_acct | 1 | 1 | 1 | 1 | 1 |
| acct_ctrl_i | 1 | 1 | 1 | 1 | 1 |
| reglk_mem[0] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[1] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[2] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[3] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[4] | 32 | 32 | 32 | 0 | 24 |
| reglk_mem[5] | 32 | 32 | 32 | 0 | 24 |
| en | 1 | 1 | 1 | 0 | 1 |
| reglk_ctrl | 16 | 16 | 16 | 0 | 8 |
| rdata | 32 | 0 | 32 | 0 | 18 |
| reglk_ctrl_o | 112 | 0 | 112 | 0 | 112 |
| Investment (bits) | 463 | 319 | | 110 | |
| Output Score (bits) | 463 | 463 | | 393 | |
| Normalized Score | 1 | 1 | | 0.9 | |

```
assign reglk_ctrl_o = {reglk_mem[5], reglk_mem[4],
reglk_mem[3], reglk_mem[2], reglk_mem[1], reglk_mem[0]};
assign reglk_ctrl = reglk_ctrl_i;
assign en = en_acct ? acct_ctrl_i: 0;
always @(posedge clk_i) begin
  if(rst_ni && ~jtag_unlock && ~rst_9) begin
    for (j=0; j < 6; j=j+1) begin
      reglk_mem[j] <= 'h0;
    end
  end
  else if(en && we) begin
    case(address[7:3])
 0: reglk_mem[0] <= reglk_ctrl[1] ? reglk_mem[0] : wdata;
 1: reglk_mem[1] <= reglk_ctrl[1] ? reglk_mem[1] : wdata;
 2: reglk_mem[2] <= reglk_ctrl[1] ? reglk_mem[3] : wdata;
 3: reglk_mem[3] <= reglk_ctrl[1] ? reglk_mem[3] : wdata;
 4: reglk_mem[4] <= reglk_ctrl[1] ? reglk_mem[4] : wdata;
 5: reglk_mem[5] <= reglk_ctrl[1] ? reglk_mem[5] : wdata;
    endcase
  end
end
always @(*) begin
  rdata = 64'b0;
  if (en) begin
    case(address[7:3])
      0: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[0];
      1: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[1];
      2: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[2];
      3: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[3];
      4: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[4];
      5: rdata = reglk_ctrl[0] ? 1'b0 : reglk_mem[5];
    endcase
  end
end
```

| Signal Name | V1 In | V1 Out | V2 In | V2 Out | V3 In | V3 Out |
|---|---|---|---|---|---|---|
| rst_ni | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| jtag_unlock | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| rst_9 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| we | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| address | 0.0 | 0.0 | 64.0 | 64.0 | 64.0 | 64.0 |
| wdata | 32.0 | 32.0 | 32.0 | 32.0 | 32.0 | 32.0 |
| reglk_ctrl_i | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 |
| en_acct | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| acct_ctrl_i | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| reglk_mem[0] | 0.0 | 15.8 | 0.0 | 24.0 | 0.0 | 24.0 |
| reglk_mem[1] | 0.0 | 15.8 | 0.0 | 24.0 | 0.0 | 24.0 |
| reglk_mem[2] | 0.0 | 15.8 | 0.0 | 24.0 | 0.0 | 24.0 |
| reglk_mem[3] | 0.0 | 15.8 | 0.0 | 24.0 | 0.0 | 24.0 |
| reglk_mem[4] | 0.0 | 15.8 | 0.0 | 24.0 | 0.0 | 24.0 |
| reglk_mem[5] | 0.0 | 15.8 | 0.0 | 24.0 | 0.0 | 24.0 |
| en | 1.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 |
| reglk_ctrl | 0.0 | 8.0 | 0.0 | 8.0 | 0.0 | 8.0 |
| rdata | 0.0 | 11.8 | 0.0 | 18.0 | 32.0 | 32.0 |
| reglk_ctrl_o | 0.0 | 94.5 | 0.0 | 112.0 | 112.0 | 112.0 |
| Investment (bits) | 45 | | 110 | | 254 | |
| Output Score (bits) | 253.8 | | 393 | | 407 | |
| Normalized Score | 0.6 | | 0.9 | | 0.9 | |

Different patching strategies result in different patchability scores

# Outline

- Background & Motivation
- Patchability Quantification Formulation
- Case Study Analysis
- Conclusions

# Conclusions

- Propose a novel definition of patchability
- We present a new approach to measure the patchability of patching logic.
- Our method assists designers in exploring the effect of different patching options.
- Our proposed approach achieves a normalized score of 0.9 while using 65% fewer resources compared to a greedy approach