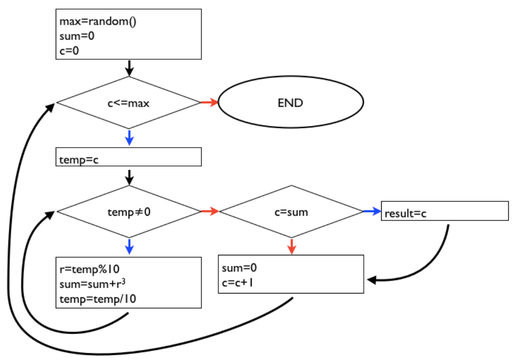
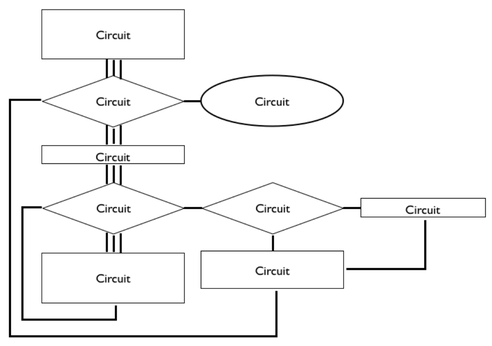
프로그램의 정적 분석을 디지털 회로로 표현하기

윤용호
서울대학교 프로그래밍연구실

1. 동기

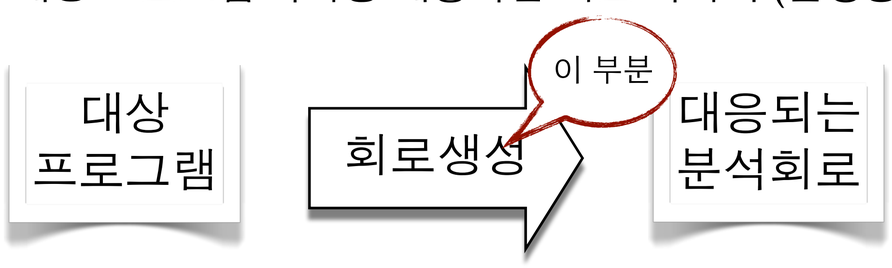




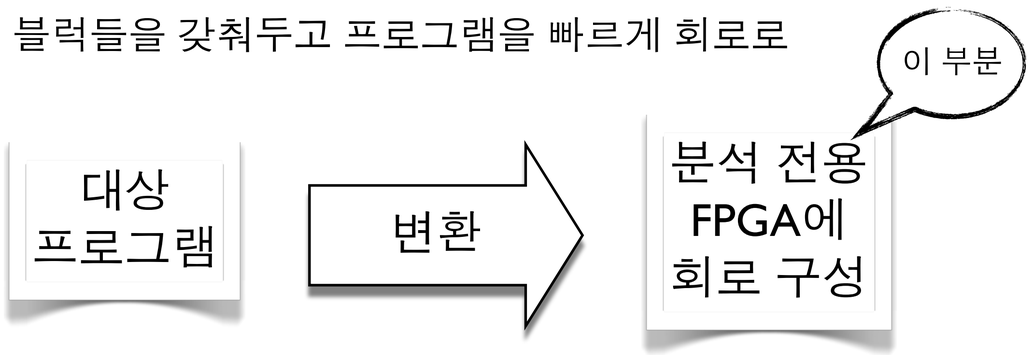
- 프로그램의 CFG는 회로와 닮았다
- 분석에서 각 블럭이 각자 자기 일만 하는 것도 닮았다
- 모든 계산을 동시에 하는 회로의 능력을 활용할 수 없을까?
- 새로운 시도, 재미있으니까!

2. 여러 방향 구상

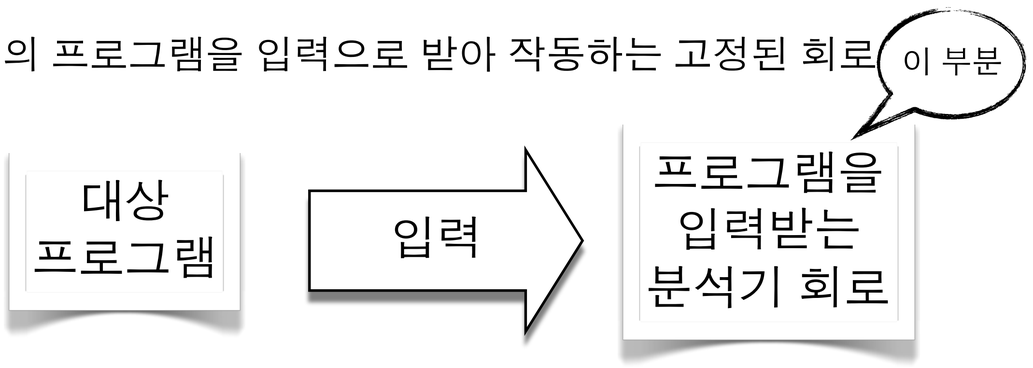
1. 분석 대상 프로그램 하나당 대응하는 회로 하나씩 (진행중)



2. 미리 블럭들을 갖춰두고 프로그램을 빠르게 회로로



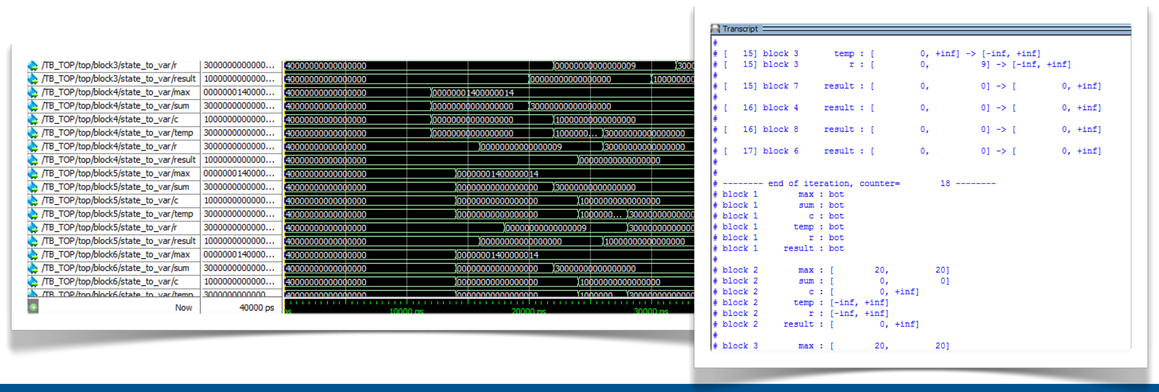
3. 임의의 프로그램을 입력으로 받아 작동하는 고정된 회로



3. 환경 (Verilog & Modelsim)

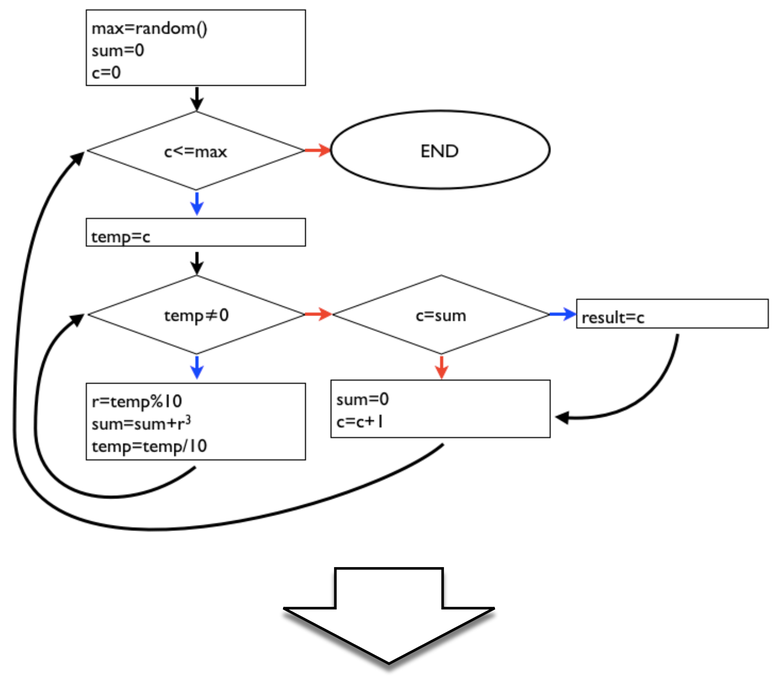
- Verilog HDL
  - 가장 널리 쓰이는 HDL(Hardware Description Language)
  - 구조가 단순해서 생성될 회로 예측이 쉬움

- Modelsim
  - 디지털 회로 컴파일러 & 시뮬레이터
  - 비트, 클럭 단위 시뮬레이션
  - 결과를 파형과 콘솔 텍스트 출력으로 표현



4. 간단한 실험 결과

대상 프로그램




```
module TOP(reset, clk);
  `include "parameters.v"

  input reset, clk;

  wire [state_size-1:0] b1, b2_1, b2_2, b3, b4, b5, b6, b7_1, b7_2, b8_1, b8_2, joined_b2, joined_b5, joined_b7;

  wire [state_size-1:0] bot_state;
  wire [value_size-1:0] bot_value;

  wire zero;
  wire widen;

  assign zero=0;
  assign widen=1;

  assign bot_value={3'b100, 32'd0, 32'd0};

  assign bot_state={bot_value[value_size-1:0], bot_value[value_size-1:0], bot_value[value_size-1:0], bot_value[value_size-1:0],

  BLOCK1 block1(.state_in(bot_state), .state_out(b1), .widen(zero), .reset(reset), .clk(clk));
  JOIN_STATE join2(.state_a(b5), .state_b(b1), .state_out(joined_b2), .widen(zero));
  BLOCK2 block2(.state_in(joined_b2), .state_true(b2_1), .state_false(b2_2), .widen(widen), .reset(reset), .clk(clk));
  BLOCK3 block3(.state_in(b2_1), .state_out(b3), .widen(zero), .reset(reset), .clk(clk));
  BLOCK4 block4(.state_in(b7_1), .state_out(b4), .widen(zero), .reset(reset), .clk(clk));
  JOIN_STATE join5(.state_a(b6), .state_b(b8_2), .state_out(joined_b5), .widen(zero));
  BLOCK5 block5(.state_in(joined_b5), .state_out(b5), .widen(widen), .reset(reset), .clk(clk));
  BLOCK6 block6(.state_in(b8_1), .state_out(b6), .widen(zero), .reset(reset), .clk(clk));
  JOIN_STATE join7(.state_a(b3), .state_b(b4), .state_out(joined_b7), .widen(zero));
  BLOCK_BRANCH block7(.state_in(joined_b7), .state_true(b7_1), .state_false(b7_2), .widen(widen), .reset(reset), .clk(clk));
  BLOCK_BRANCH block8(.state_in(b7_2), .state_true(b8_1), .state_false(b8_2), .widen(zero), .reset(reset), .clk(clk));

endmodule
```

- 수작업으로 Verilog 코드로 변환 (총 크기 30KB)

- 정수 구간 분석(Interval)을 하는 회로
  - 축지법(Widening), 좁히기(Narrowing)도 적용

- 18 사이클에 수렴
  - 학부 강의에서 쓰는 300MHz FPGA 보드라면? 60ns

5. 앞으로

- Sparse 분석을 하는 회로
  - 레지스터 수는 메모리 크기보다도 큰 제약
  - 회로가 스스로 Sparse 분석을 구성하는 것이 가능한가?

- C → Verilog 코드 자동 생성기
  - 실제 회로로 합성(Synthesis) 가능한 코드

- Verilog 코드 → 회로 합성 속도 고민
  - 일반적으로 회로 합성엔 매우 시간이 오래 걸림
  - 속도를 빠르게 할 수 있는 방법 고찰
    - 미리 정해진 블럭을 최대한 활용