Simopt

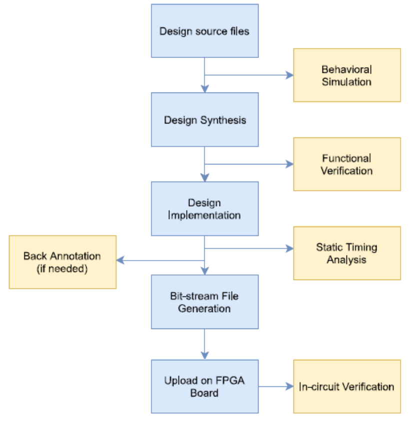
Simulation pass for Speculative Optimisation of FPGA-CAD flow

Eashan Wadhwa, Shanker Shreejith

July 30, 2024

Trinity College Dublin

Simulator : not used in the CAD flow

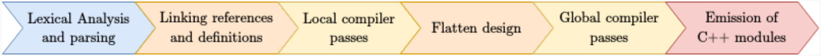


- Simulation is **only** used to verify your HDL circuit
- Not used at all in the CAD flow
- Compared to other Processing elements (CPUs/GPUs) where the `main()` function dictates importance

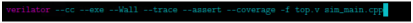
An example conditional Verilog model

```verilog
// ports:
// input [31:0] in, reset, clk;
// output [31:0] out;
//
reg [31:0] count_c;    // assume = 0
reg [31:0] pointless;  // assume = 0
always_ff @ (posedge clk) begin
    count_c <= count_c + 1;
    if (~reset) begin
        assign out = in;
    end
    else begin
        assign out = count_c;
        if (pointless == 31'hF00BA) begin
            assign out = 31'hF00BA;
        end
    end
end
always_ff @ (posedge clk) begin
    pointless <= pointless + 1;
end
```

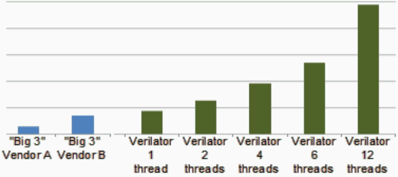
Simulator : Verilator[1]



- Open-source simulator which converts Verilog to C++
- Fast compared to some of the other simulators
- Follows object oriented principles making any code integration much easier (and reliable)

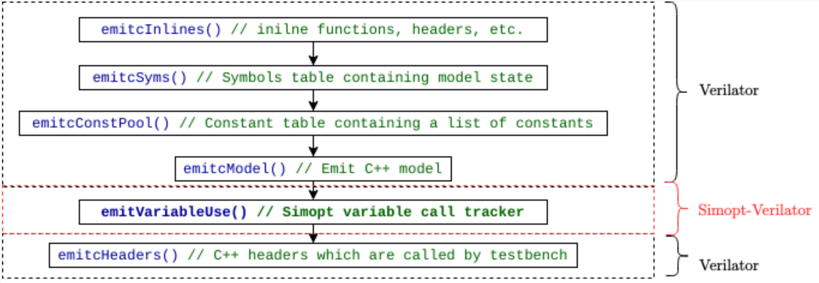


```
verilator --cc --exe --Wall --trace --assert --coverage -f top.v sim_main.cpp
```



[1] https://github.com/verilator/verilator

Simopt Verilator



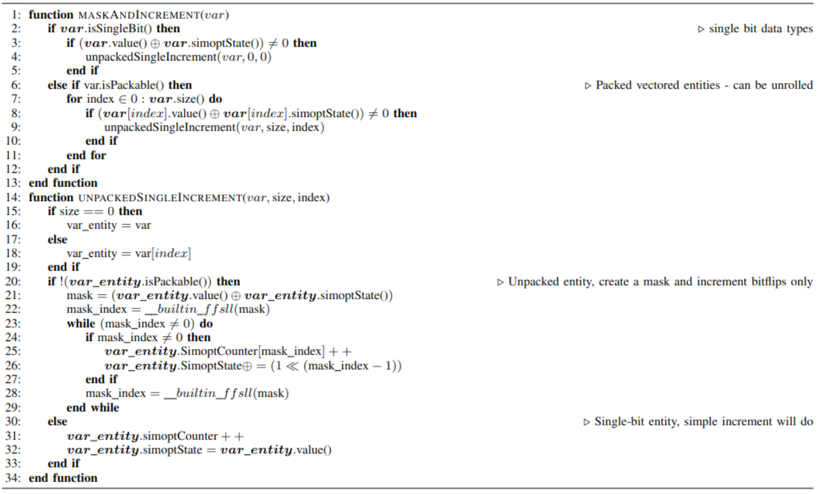
```
emitcInlines() // inilne functions, headers, etc.
emitcSyms() // Symbols table containing model state
emitcConstPool() // Constant table containing a list of constants
emitcModel() // Emit C++ model
emitVariableUse() // Simopt variable call tracker
emitcHeaders() // C++ headers which are called by testbench
```

- Emits variable trackers called *simopt-counters*
- Packed verilog types need *simopt-states* which save the encountered simopt-counter
- Once the implementation is defined, needs to be called in runtime

How the emitted conditional Verilog model looks

```verilog
// ports:
// input[31:0] in, reset, clk;
// output[31:0] out;
//
reg [31:0] count_c, pointless;
always_ff @ (posedge clk) begin
    count_c <= count_c + 1;
    if (~reset) begin
        assign out = in;
        maskAndIncrement(in, out);
    end
    else begin
        assign out = count_c;
        if (pointless == 31'hFOOBA) begin
            assign out = 31'hFOOBA;
            maskAndIncrement(out);
        end
        maskAndIncrement(count_c, out, pointless);
    end
    maskAndIncrement(reset, count_c);
end
```

maskAndIncrement() algorithm



```
function MASKANDINCREMENT($var$)
    if $\boldsymbol{var}$.isSingleBit() then                                        ▷ single bit data types
        if $(\boldsymbol{var}.\text{value}() \oplus \boldsymbol{var}.\text{simoptState}()) \neq 0$ then
            unpackedSingleIncrement($var$, 0, 0)
        end if
    else if var.isPackable() then                                   ▷ Packed vectored entities - can be unrolled
        for index $\in 0 : \boldsymbol{var}$.size() do
            if $(\boldsymbol{var}[index].\text{value}() \oplus \boldsymbol{var}[index].\text{simoptState}()) \neq 0$ then
                unpackedSingleIncrement($var$, size, index)
            end if
        end for
    end if
end function
function UNPACKEDSINGLEINCREMENT($var$, size, index)
    if size $==$ 0 then
        var_entity $=$ var
    else
        var_entity $=$ var$[index]$
    end if
    if !($\boldsymbol{var\_entity}$.isPackable()) then     ▷ Unpacked entity, create a mask and increment bitflips only
        mask $= (\boldsymbol{var\_entity}.\text{value}() \oplus \boldsymbol{var\_entity}.\text{simoptState}())$
        mask_index $= \_\_builtin\_ffsll$(mask)
        while (mask_index $\neq$ 0) do
            if mask_index $\neq$ 0 then
                $\boldsymbol{var\_entity}$.SimoptCounter[mask_index] $++$
                $\boldsymbol{var\_entity}$.SimoptState$\oplus = (1 \ll (\text{mask\_index} - 1))$
            end if
            mask_index $= \_\_builtin\_ffsll$(mask)
        end while
    else                                                            ▷ Single-bit entity, simple increment will do
        $\boldsymbol{var\_entity}$.simoptCounter $++$
        $\boldsymbol{var\_entity}$.simoptState $= \boldsymbol{var\_entity}$.value()
    end if
end function
```

maskAndIncrement() algorithm (cntd.)

Single types

```
logic single

// 1-bit logic
```

Vectored types

```
logic [31:0] vectored

// 32-bit wide logic
```

Packed arrays

```
logic [3:0][7:0] packed_array

// 4 elements
// 8 bit wide
```

Find First Set Bit Long (FFSL)

Returns Most significant **set** bit index of an input value

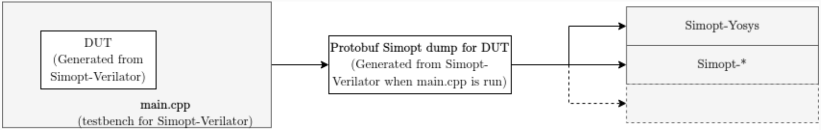
```
wire [31:0] array;

initial begin
  // Assign values to each 8-bit slice
  array[7:0] = 8'h01; // First element
  array[15:8] = 8'h02; // Second element
  array[23:16] = 8'h03; // Third element
  array[31:24] = 8'h04; // Fourth element
end
```

```
logic [3:0][7:0] array;

initial begin
  // Assign values to each element
  array[0] = 8'h01; // First element
  array[1] = 8'h02; // Second element
  array[2] = 8'h03; // Third element
  array[3] = 8'h04; // Fourth element
end
```

Simopt dump


DUT
(Generated from
Simopt-Verilator)
main.cpp
(testbench for Simopt-Verilator)
Protobuf Simopt dump for DUT
(Generated from Simopt-
Verilator when main.cpp is run)
Simopt-Yosys
Simopt-*

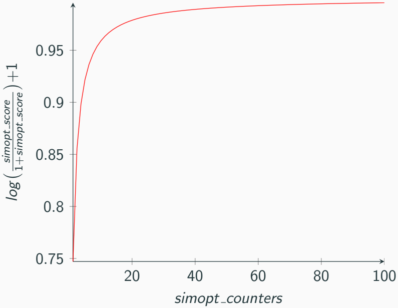
- Protobuf format [2] is used to encode the Simopt dump
- A Protobuf parser is needed on the client side (here Simopt-Yosys)
- Easier integration

[2] https://protobuf.dev/

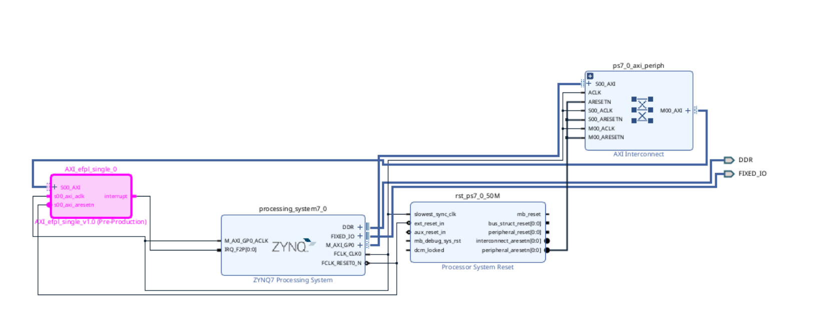
Integration of Simopt-counters into placing

- Used ABC's placing strategy as our target
- Use the existing ABC commands hardcoded for *synth_xilinx* in Yosys.
- Uses a "priority cut" method which works on a basis of LUT and cost function
- Integrate into the LUT mapping cost function -

$$A_{simopt} = \sum_{i}^{N} \left[\frac{L_{inputs}}{L_{max\_inputs}} \times L_{outputs} \times \left(log \left(\frac{simopt\_score}{1 + simopt\_score} \right) + 1 \right) \right]$$

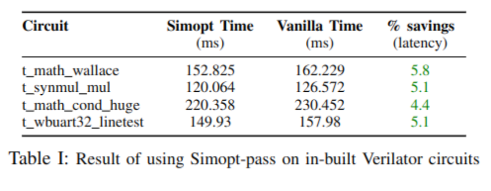


How to generate results



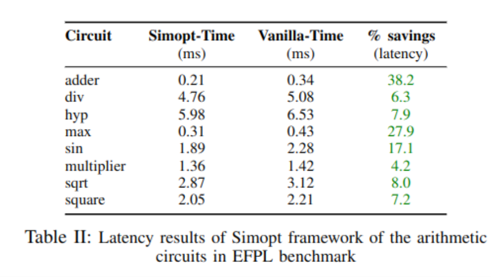
- Wrapped the placed design as a black-box design in Vivado IP integrator
- Is then stitched together with the rest of the flow
- A FreeRTOS driver was written which timed the interrupt difference

Results



Circuit	Simopt Time (ms)	Vanilla Time (ms)	% savings (latency)
t_math_wallace	152.825	162.229	5.8
t_synmul_mul	120.064	126.572	5.1
t_math_cond_huge	220.358	230.452	4.4
t_wbuart32_linetest	149.93	157.98	5.1

Table I: Result of using Simopt-pass on in-built Verilator circuits

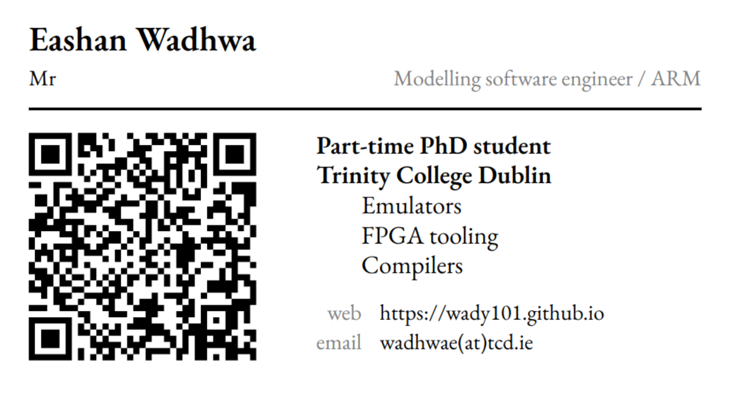


Circuit	Simopt-Time (ms)	Vanilla-Time (ms)	% savings (latency)
adder	0.21	0.34	38.2
div	4.76	5.08	6.3
hyp	5.98	6.53	7.9
max	0.31	0.43	27.9
sin	1.89	2.28	17.1
multiplier	1.36	1.42	4.2
sqrt	2.87	3.12	8.0
square	2.05	2.21	7.2

Table II: Latency results of Simopt framework of the arithmetic circuits in EFPL benchmark

Conclusion

- Simulation metadata can actually be used to speed up runtime
- FPGAs can a learn a lot from CPUs/GPUs and vice-versa :)
- Introduces speculation into CAD flow



Eashan Wadhwa

Mr

Modelling software engineer / ARM

Part-time PhD student
Trinity College Dublin

- Emulators
- FPGA tooling
- Compilers

web	https://wady101.github.io
email	wadhwae(at)tcd.ie