Design of a Safety-First Hardware Design Language

Lennart Van Hirtum

Prof. Christian Plessl - Promotor

Paderborn University, Germany
Paderborn Center for Parallel Computing

NHR Summer School, Paderborn University, 12-16 June 2023

Paderborn Center for Parallel Computing

Why not existing RTL languages?

- (System)-Verilog & VHDL are industry standard
- Incredibly verbose
  - But not much is gained from that verbosity
- Bugs in Verilog/VHDL are common & difficult to find!

This is REAL Verilog code, written by REAL programmers!

```
wire[ ADDR_WIDTH-1:0] addrToComputeModule2x;

wire pipelineRST2x;
synchronizer pipelineRSTSynchronizer(clk, pipelineRST, clk2x, pipelineRST2x);
(* dont_merge *) reg inputFIFORST2x; always @(posedge clk2x) inputFIFORST2x <= pipelineRST2x;
(* dont_merge *) reg cccRST2x; always @(posedge clk2x) cccRST2x <= pipelineRST2x;

// request Pipe has 0 cycle, FIFO has 5 cycles read latency, dataOut pipe has 0 cycles
`define FIFO_READ_LATENCY (0+5+0)
wire inputFifoECC2x;
FastDualClockFIFO_M20K #(.WIDTH(128+`ADDR_WIDTH), .DEPTH_LOG2(6), .ALMOST_FULL_MARGIN(10)) input
    // input side
    .wrclk(clk),
    .wrrst(pipelineRST),
    .writeEnable(isBotValid),
```

?????????

```
// PIPELINE STEP 5
reg[31:0] reducedGraphIsZeroIntermediates_SUMMARIZE;
reg[7:0] reducedGraphIsZeroIntermediates_SUMMARIZE_D;
reg[63:0] extentionFinishedIntermediates_SUMMARIZE;
reg[15:0] extentionFinishedIntermediates_SUMMARIZE_D;
reg[3:0] extentionFinishedIntermediates_SUMMARIZE_DD;
```

????????????????

```
// The combinatorial pipeline that does all the work. Loopback is done outside of
// Pipeline stages are marked by wire_STAGE for clarity
// If graphInValid == 0, then graphIn must == 128'b0
module pipelinedCountConnectedCombinatorial #(parameter EXTRA_DATA_WIDTH = 10) (
    input clk,

    // input side
    input[127:0] graphIn,
    input graphInValid,
    input[EXTRA_DATA_WIDTH-1:0] extraDataIn,

    // state loop
    input runEndIn,
    input[127:0] extendedIn_HASBIT,
    input[127:0] reducedGraphIn,
    input shouldGrabNewSeedIn_HASBIT,
    input validIn_NSD,
    input[5:0] storedConnectionCountIn_NSD,
    input[EXTRA_DATA_WIDTH-1:0] storedExtraDataIn_NSD,

    output request_EXPL,
    output shouldGrabNewSeedOut_EXPL_D,
    output[127:0] extendedOut_DOWN,
    output[127:0] reducedGraphOut_DOWN,
    output reg validOut_NSD_D,
    output reg[5:0] connectionCountOut_NSD_D,
    output reg[EXTRA_DATA_WIDTH-1:0] storedExtraDataOut_NSD_D,

    // output side
    output reg done,
    output reg[5:0] connectCountOut,
    output reg[EXTRA_DATA_WIDTH-1:0] extraDataOut
);
```

????????????????????????

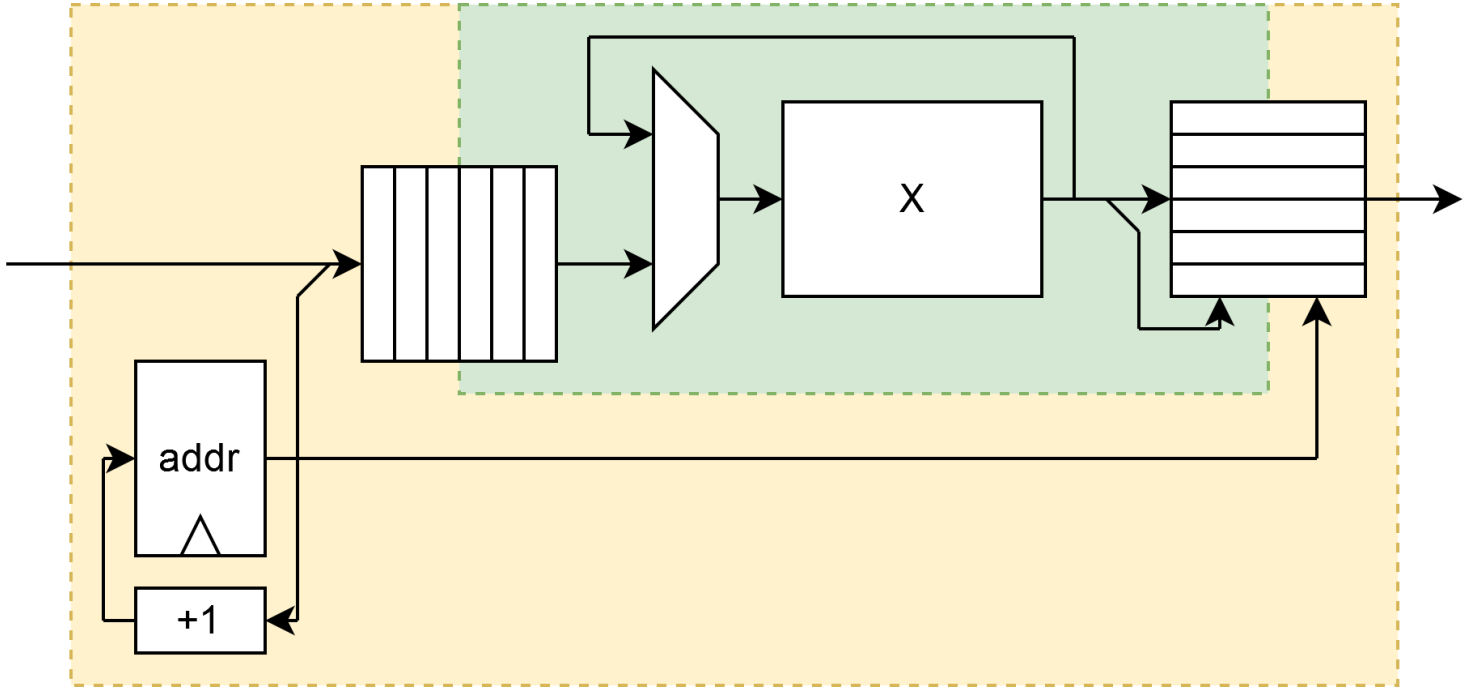
Why not HLS?

- HLS delivers good results on datapath-centric designs
- Imperative nature of HLS vs
  Simultaneous nature of hardware
- Clever compilers
  - But much fiddling
- Little control
- Proprietary

```
__kernel
__attribute__((uses_global_work_offset(0)))
void add_/*PY_CODE_GEN i*/(__global const DEVICE_ARRAY_DATA_TYPE * restrict in1,
          __global const DEVICE_ARRAY_DATA_TYPE * restrict in2,
          __global DEVICE_ARRAY_DATA_TYPE * restrict out,
          const uint array_size) {
    uint number_elements = array_size / VECTOR_COUNT;
    __attribute__((opencl_unroll_hint(UNROLL_COUNT)))
    for (uint i=0; i<number_elements; i++){
        out[i] = in1[i] + in2[i];
    }
}
```

My Goal

- New RTL Language
- As easy to use as HLS
- Stay low-level
- More Type-safety!
- Focus on tools!



Inspirations

- TL-Verilog
- Filament
- Chisel
- Rust 

github.com/pc2/sus-compiler