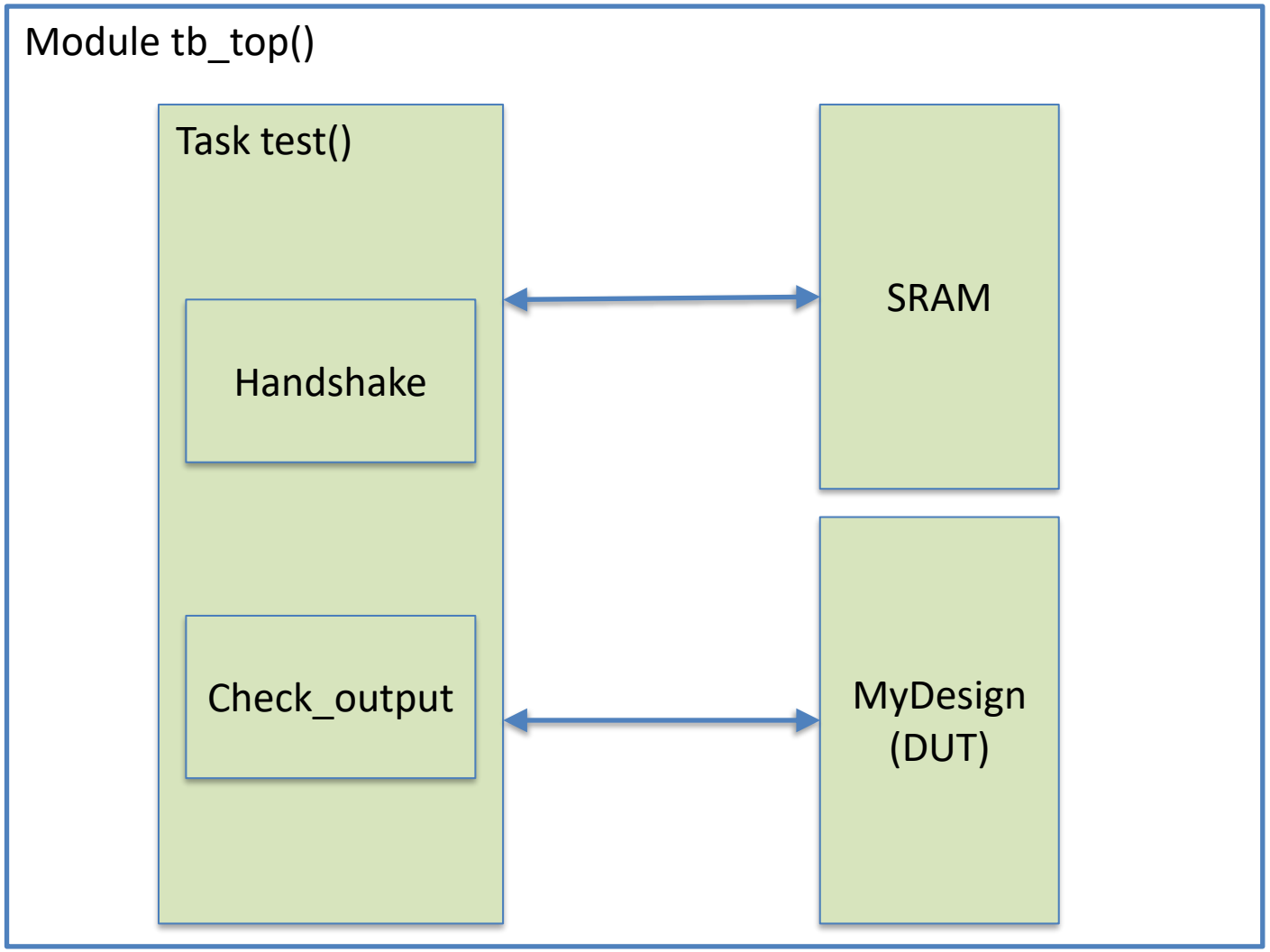
HW6
ECE 464/564: Fall 2023

Prasanth Prabu Ravichandiran

HW6 - Overview

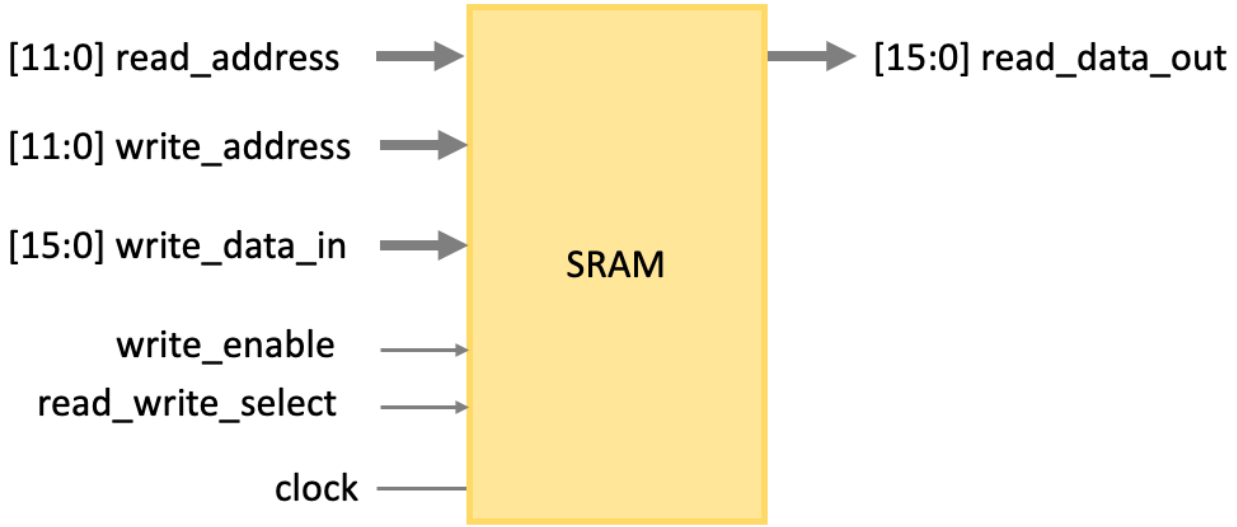
- Design a hardware module that accumulates the floating-point numbers of an array stored in a SRAM.
- Interact with testbench and SRAM module to retrieve the SRAM data for the arithmetic operation and store the results back in the SRAM. (detailed later)
- Use at least one coding feature unique to SystemVerilog.

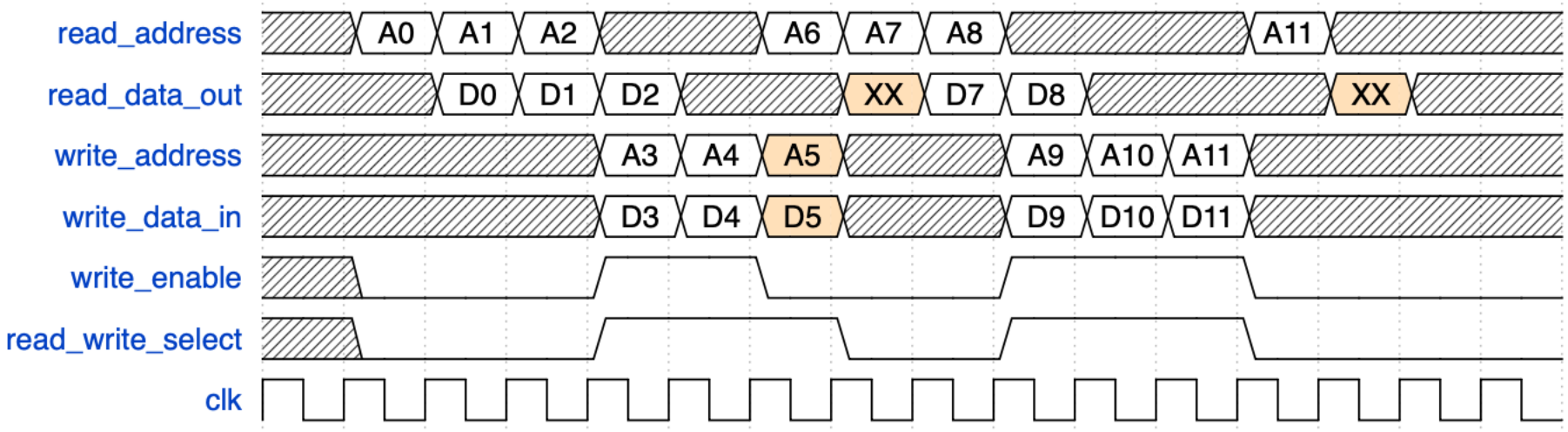
Overall architecture



SRAM

- The SRAM is word addressable and has a one cycle delay from providing the "*read_address*" to "*read_data_out*".
- When writing to the SRAM, you would have to set the "*write_enable*" to high.

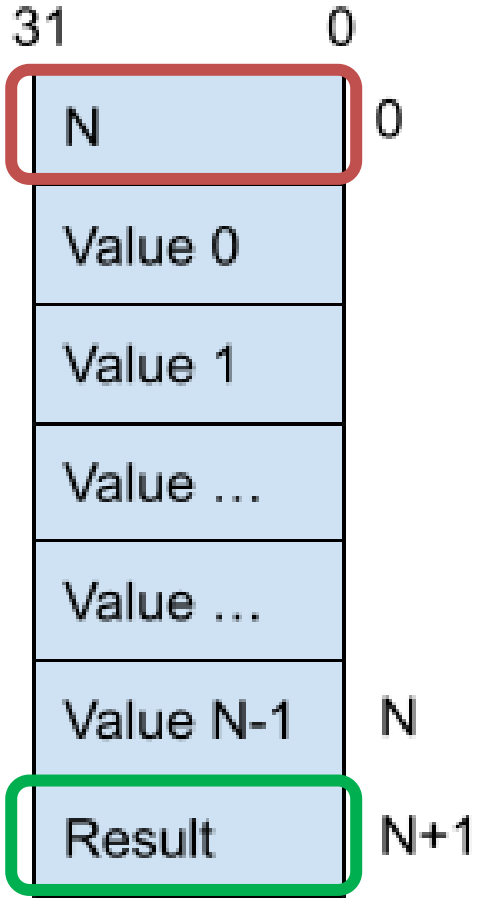




SRAM contents

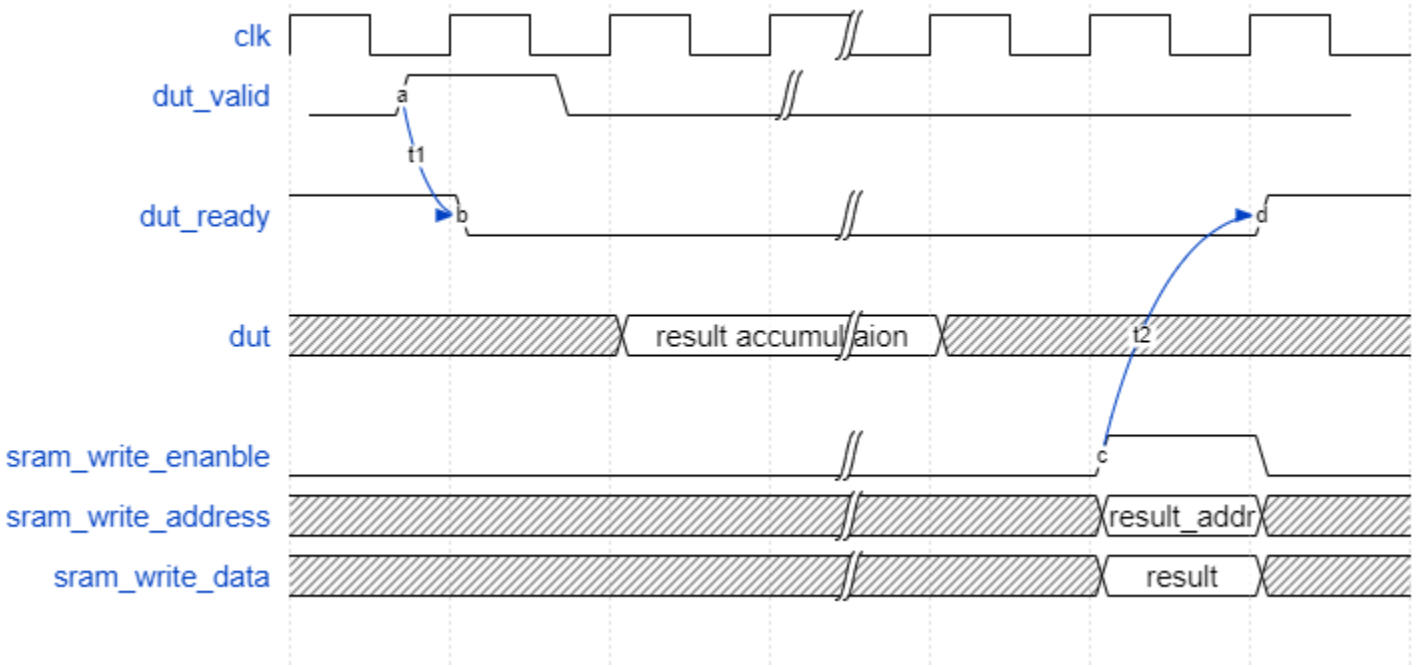
- SRAM contents are loaded from by the testbench from the corresponding /input/test*.dat file.
- The “array_size” is stored in the address = 0, as “N”.
- The array values are stored from address 1 to “N”.
- Result of floating-point accumulation needs to stored in address = N+1 for that iteration.

- **Note:** Floating-pointing numbers are not associative, so the order of accumulation should be from 0 to N-1. [i.e: (a+b)+c ≠ a+(b+c)].



DUT – Test fixture handshake

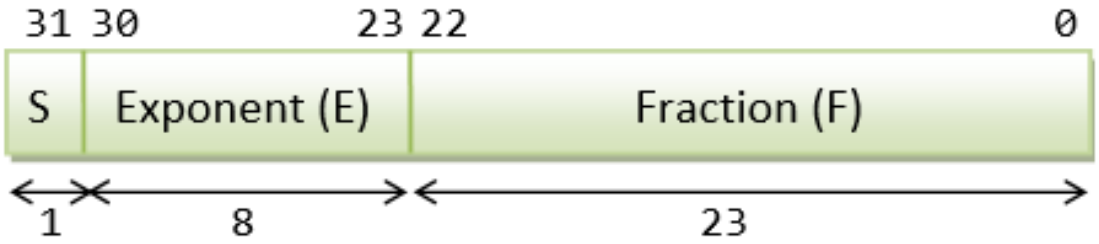
- **dut_valid:** (Test fixture -> DUT)
  - Signals that a valid input can be computed from the SRAM.
- **dut_ready:** (DUT -> Test fixture)
  - Signals that the DUT is ready to receive new input from the SRAM.
  - DUT should hold the dut_ready low until it has populated the result value back in the SRAM.



Floating point number

- 32-bit Single-Precision Floating-point number as per IEEE-754 standard is used.
- Use the syntax to make it readable:

```
155  // synopsys translate_off
156    shortreal test_val;
157    assign test_val = $bitstoshortreal(sum_r);
158  // synopsys translate_on
```



32-bit Single-Precision Floating-point Number

- Reference:
  https://en.wikipedia.org/wiki/Floating-point_arithmetic

DW modules

- DW_fp_add is a floating-point component that adds two floating-point values, a and b, to produce a floating-point sum, z.
- Follows IEEE-754 standard for single precision (32-bit).

```
50    DW_fp_add  #(
51      .sig_width        (23),
52      .exp_width        (8),
53      .ieee_compliance  (3)
54    ) fp_add_mod (
55      .a                (sum_r),
56      .b                (in),
57      .rnd              (3'd0),
58      .z                (sum_w),
59      .status           (status));
60
```

Pin Name	Width	Direction	Function
a	*sig_width* + *exp_width* +1 bits	Input	Input data
b	*sig_width* + *exp_width* +1 bits	Input	Input data
z	*sig_width* + *exp_width* +1 bits	Output	a + b

Provided in assignment

You will find the following directories in HW6/

- **inputs/** - Contains the .dat files for the input SRAMs used in HW.
- **rtl/** - A template dut.v that interfaces with the test fixture is provided.
- **run/** -Contains the Makefile to compile and simulate the design.
- **synthesis/** - To synthesize your design and generate reports.
- **testbench/ -** Contains the test fixture of the HW.
- README – Contains instructions for the HW and other details.
- Designware documents are provided in the Moodle under HW6 description.

Submission format

- **Project Verilog and synthesis files.** Submitted electronically on the date indicated in the class schedule. Please turn in the following:
  - **dut.sv** (All Verilog files AS ONE FILE)
  - Modelsim simulation results file showing correct functionality. Logs from 'HW6/run/logs/*.log'
  - Synopsys view_command.log file from complete synthesis run. Logs from 'HW6/synthesis/logs/*.log' and 'HW6/synthesis/reports/*.log'

- **Project Report. <unity_id_hw6_report.pdf>** (eg: pravich2_hw6_report.pdf)
  - Complete report to be turned in electronically with project files. It must follow the format provided. There is a 10% penalty for not following the format.

DW_fp_mac

- DW_fp_mac is a floating-point component that performs the multiply and add operation. It sums up a floating-point product of input a and b to input c (ab + c) to produce a floating-point multiply and add result, z.

Pin Name	Width	Direction	Function
a	*exp_width* + *sig_width* + 1 bits	Input	Multiplier
b	*exp_width* + *sig_width* + 1 bits	Input	Multiplicand
c	*exp_width* + *sig_width* + 1 bits	Input	Addend
rnd	3 bits	Input	Rounding mode; suppo in the *Datapath Floatin*
z	*exp_width* + *sig_width* + 1 bits	Output	MAC result (a x b + c)

DW_fp_mac

Instantiation:

```
88    // Instance of DW_fp_mac
89    DW_fp_mac #(inst_sig_width, inst_exp_width, inst_ieee_compliance) U1 (
90      .a(inst_a),
91      .b(inst_b),
92      .c(inst_c),
93      .rnd(inst_rnd),
94      .z(z_inst),
95      .status(status_inst)
96    );
97
```

Common suggestion

- Make sure you are resetting all the registers to a known state.
- Separate your data-path and control-path.
- Do not interchange non-blocking and blocking statements for a register.
- If your design is timing out, then check your dut_ready and dut_valid signals.