Digital Techniques 2

L7: Sequential Logic 2

UNIVERSITY OF OULU
CIRCUITS and SYSTEMS

Recap from Previous Lecture

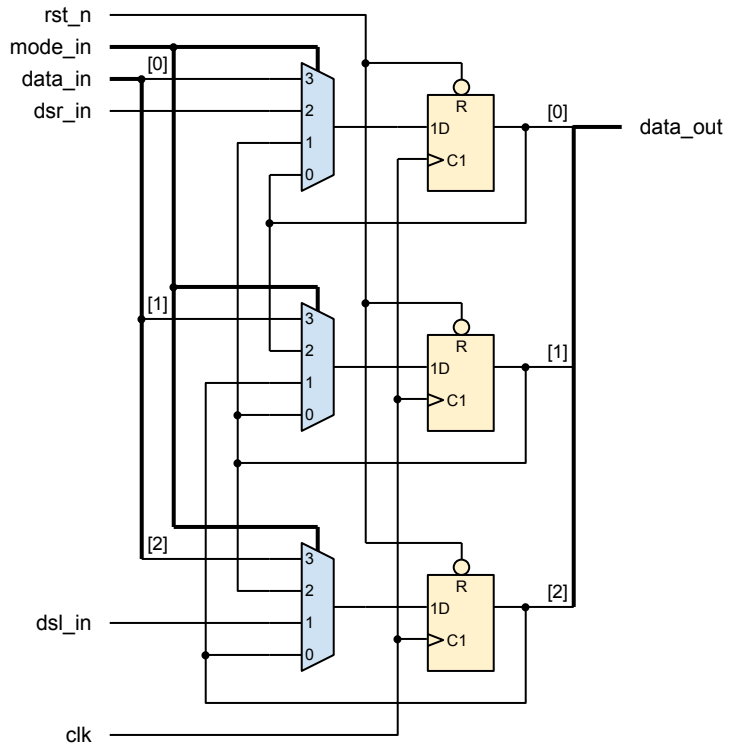
- Sequential processes must be modeled using "industry-standard" **code templates** so that all synthesis tools can infer registers in the same way
- Learn the basic templates for different cases (with or without active-high or active-low asynchronous or synchronous reset)
- Start coding by first writing the complete template and then fill in the blanks

```
always_ff @ (posedge clk or negedge rst_n)
  begin
    if (rst_n == '0)
      begin
        // Reset the variables here
      end
    else
      begin
        // Compute next state
        // values here
      end
  end
```

Example: Universal Shift-Register

	mode_in	dsl_in	dsr_in	data_in	srg_r	next srg_r
NO OP	00	0	0	111	000	000
LOAD	11	0	0	111	000	111
SHIFT LEFT	01	0	0	111	111	110
SHIFT LEFT	01	0	0	111	110	100
SHIFT RIGHT	10	0	0	111	100	010
SHIFT RIGHT	10	1	0	111	010	101

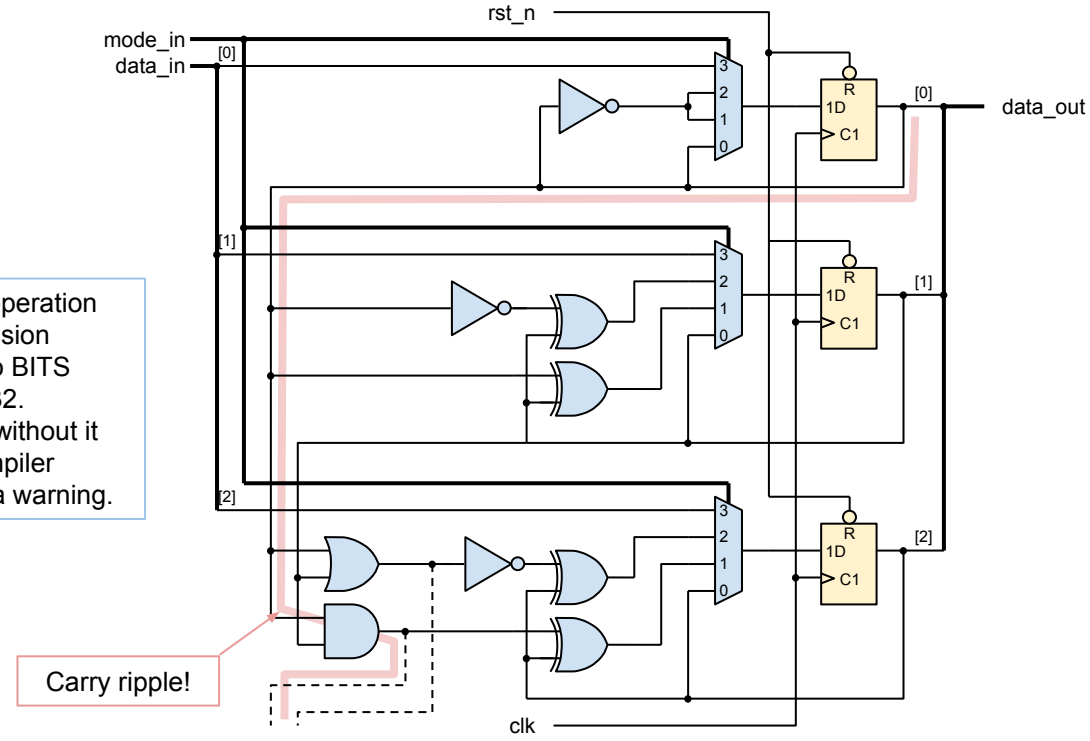
```
module srg #( parameter int BITS = 4 )
  (input logic clk, rst_n, dsl_in, dsr_in
   input logic [1:0]       mode_in,
   input logic [BITS-1:0]  data_in,
   output logic [BITS-1:0] data_out);

   logic [BITS-1:0]      srg_r;

   always_ff @(posedge clk or negedge rst_n)
     begin : srg_proc
       if (rst_n == '0)
         srg_r <= '0;
       else
         case (mode_in)
           2'b00:  // NO OP
             srg_r <= srg_r;
           2'b01: // SHIFT LEFT
             srg_r <= { srg_r[BITS-2:0], dsl_in }; // { } = concatenation
           2'b10:  // SHIFT RIGHT
             srg_r <= { dsr_in, srg_r[BITS-1:1] };
           2'b11:  // LOAD
             srg_r <= data_in;
         endcase
     end : srg_proc

   assign data_out = srg_r;
endmodule
```



Example: Up-Down Counter with Parallel Load

```
module ctr #( parameter int BITS = 3 )
  (input logic clk, rst_n,
   input logic [1:0]       mode_in,
   input logic [BITS-1:0]  data_in,
   output logic [BITS-1:0] data_out);

   logic [BITS-1:0]      ctr_r;

   always_ff @(posedge clk or negedge rst_n)
     begin : counter
       if (rst_n == '0)
         ctr_r <= '0;
       else
          case (mode_in)
            2'b00:  // NO OP
              ctr_r <= ctr_r;
            2'b01: // COUNT UP
              ctr_r <= BITS'(ctr_r + 1);
            2'b10:  // COUNT DOWN
              ctr_r <= BITS'(ctr_r - 1);
            2'b11:  // LOAD
              ctr_r <= data_in;
          endcase
     end : counter

   assign data_out = ctr_r;
endmodule
```

Type cast operation sets expression bit-length to BITS instead of 32. Works OK without it but the compiler generates a warning.

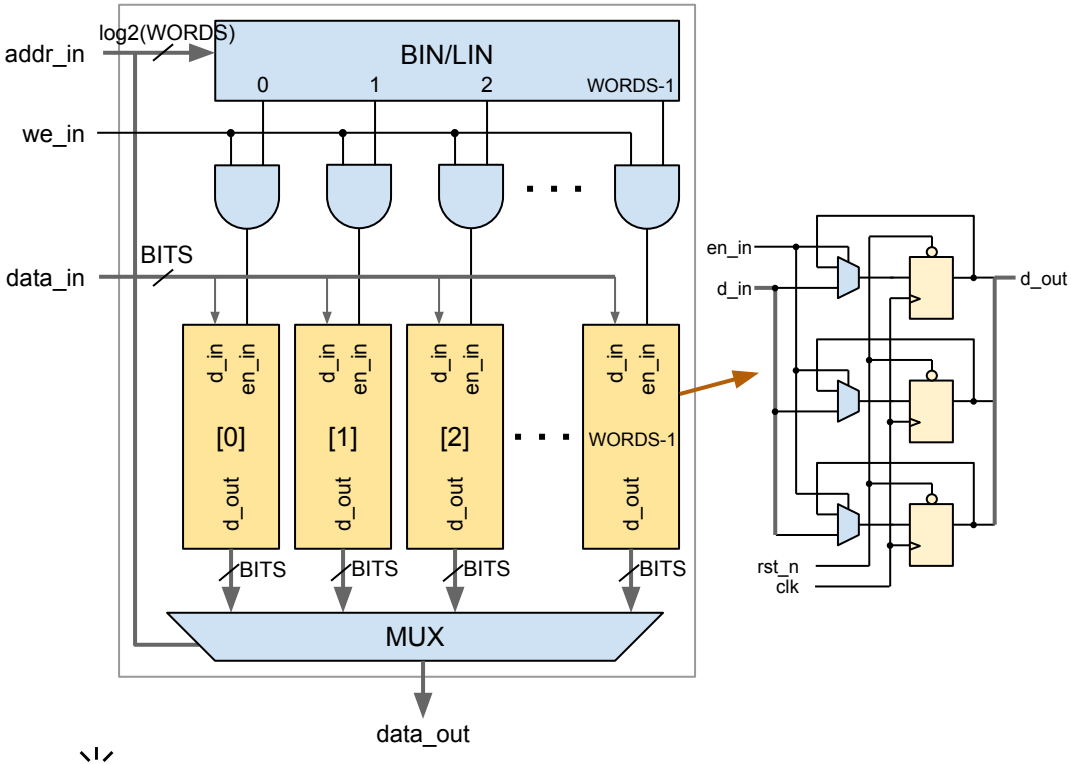


For a faster solution, google "ring counter"

Example: Register Bank

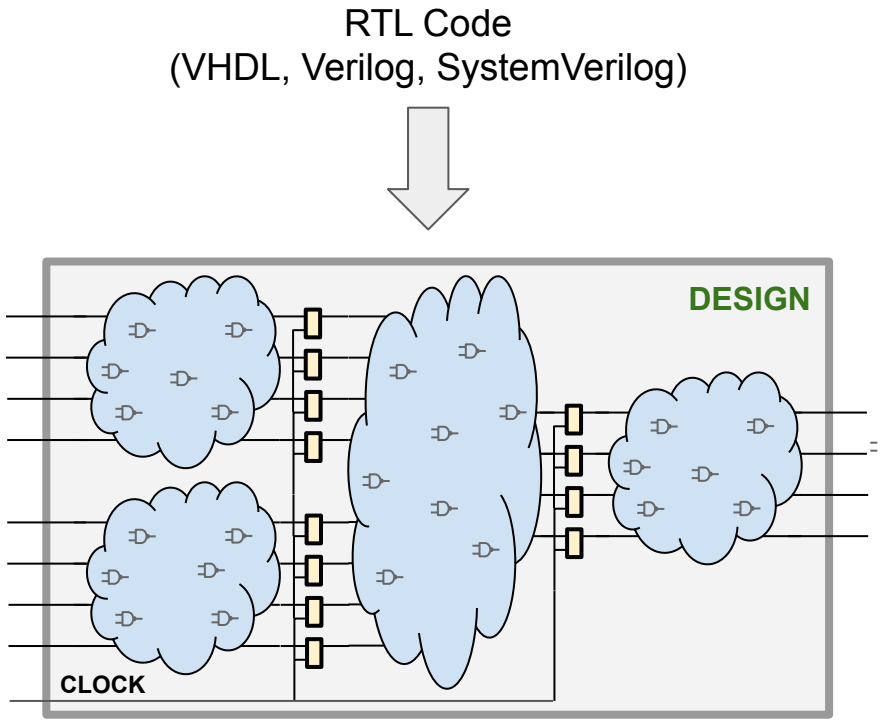
```
module regbank #( parameter int WORDS = 8, BITS = 4 )
  (input logic clk, rst_n, we_in,
   input logic [$clog2(WORDS)-1:0] addr_in,
   input logic          [BITS-1:0] data_in,
   output logic         [BITS-1:0] data_out);

   logic [WORDS-1:0][BITS-1:0] regbank_r;

   always_ff @(posedge clk or negedge rst_n)
     begin : regbank_proc
       if (rst_n == '0)
         regbank_r <= '0;
       else
         if (we_in == '1 && addr_in < WORDS)
            regbank_r[addr_in] <= data_in;
     end : regbank_proc

   always_comb
     begin
       if (addr_in < WORDS)
         data_out = regbank_r[addr_in];
       else
         data_out = '0;
     end

endmodule
```

* If WORDS is a power of 2, range checking is not required.

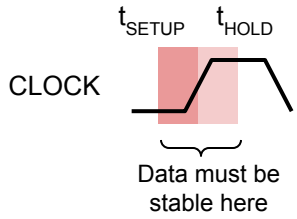


Synthesis of Sequential Logic Circuits

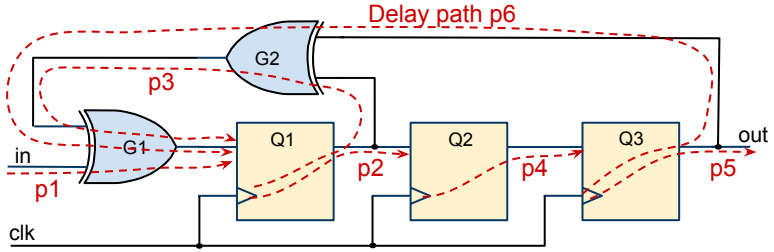
- After RTL code translation, the design consists of **combinational logic functions** and **flip-flops**
- The number of flip-flops is determined by the RTL architecture defined in the code
- Combinational logic, on the other hand, can be optimized for timing and area, and mapped to gates in many ways
- In logic synthesis, the synthesis tool tries to optimize all combinational logic "clouds" between the registers so that their **timing constraints** are met

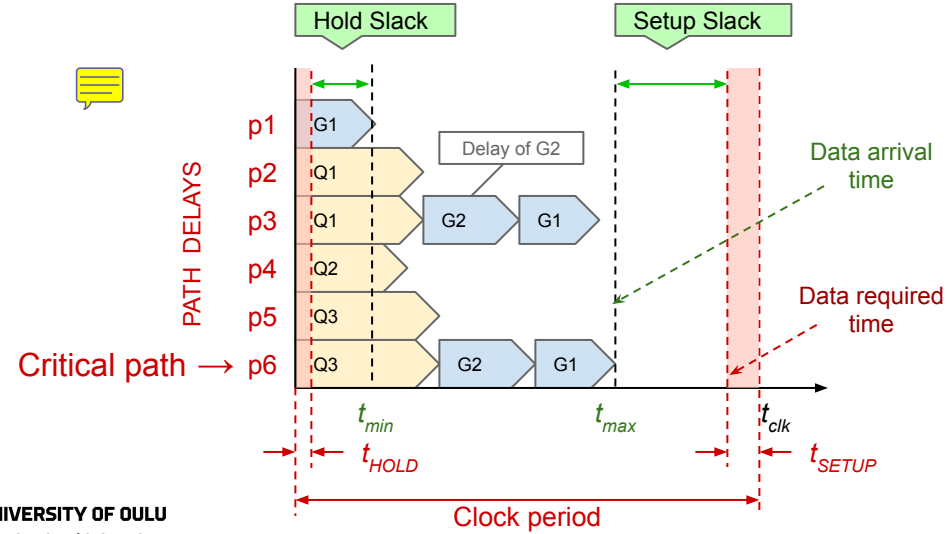


Static Timing Analysis (STA)



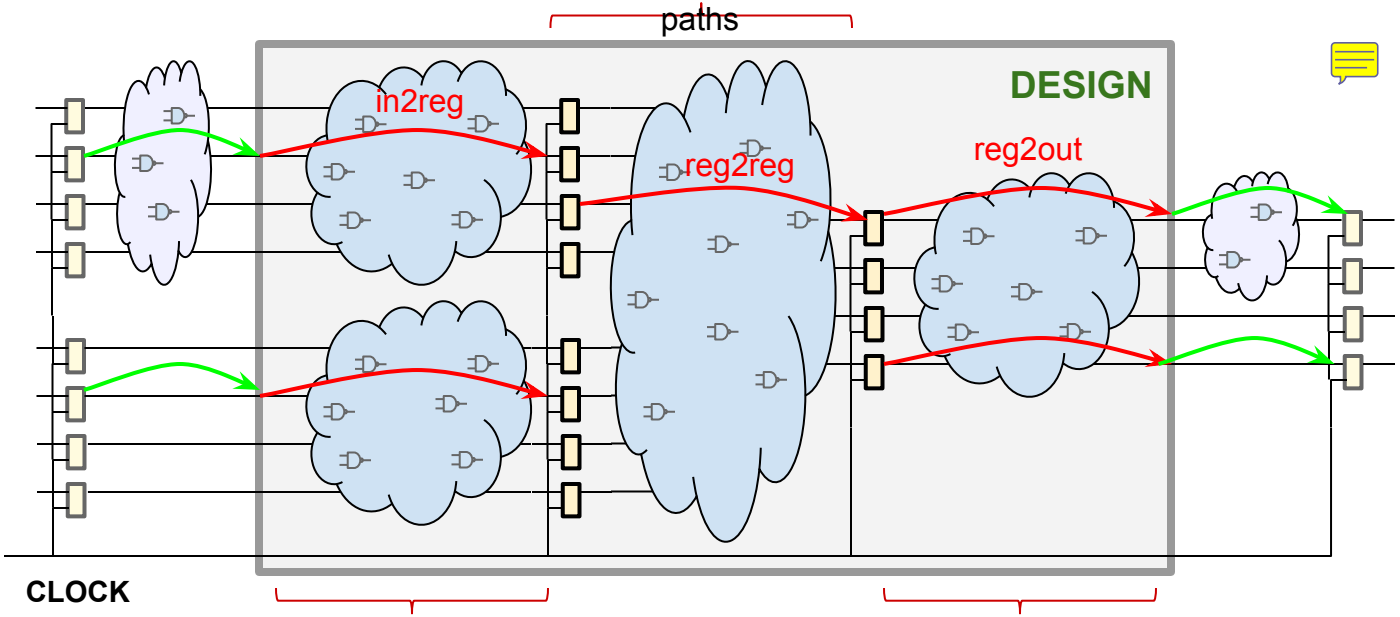
- **Setup** and **hold** time constraints of flip-flops must be respected on all delay paths
- In sequential logic synthesis, **static timing analysis** of all delay paths is performed to make sure that the design will work
- Typically, only setup time is considered in logic synthesis (hold fixing is done in routing phase)
- STA procedure for setup analysis:
  1) List all delay paths and their components
  2) Add up delays of all components on every path to form **data arrival times** for all path endpoints
  3) Compare most critical arrival time to **data-required time** (T_{clk} - T_{SETUP} for setup paths)





Timing Constraints for Sequential Logic Synthesis

Clock period completely constrains all register-to-register paths

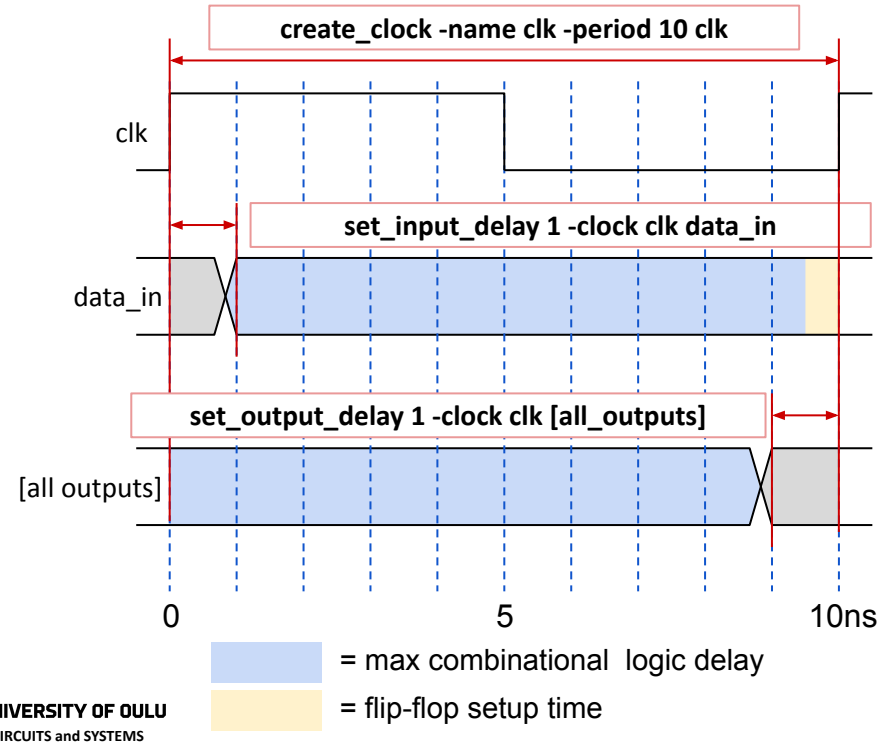


On paths that begin from inputs (in2reg) or end at outputs (reg2out) the maximum allowed delay is determined by clock period and the delay of **external paths**.

Creation of Timing Constraints

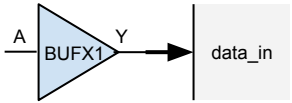
- **Clock period** sets a constraint on all register-to-register paths
- For input-to-register paths, we must define the **input delay** that tells when the data arrives at the input pins from a register outside our design
- For register-to-output paths, we must define the **output delay** that tells how much time must be reserved for the data to travel form our design's outputs to the next register outside our design
- SDC (Synopsys Design Constraints) is a widely used format for timing constraints

SDC Commands for Timing Constraints:

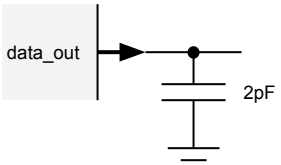


Environment Modeling for Synthesis

- For realistic timing analysis, the physical environment must be modeled
- For each input, the **drive capability** of the external driver must be modeled: Often a library component is used as a model for the driving component ("cell")
- For each output, a **load capacitance** must be defined
- Manufacturing **process**, supply **voltage** level, and operating **temperature** are usually modeled by using a component library that has been characterized for specific **operating conditions** ("PVT corner")




```
set_driving_cell -lib_cell BUFX1 -pin Y data_in
```




```
set_load 2 data_out
```

```
set target_library "saed32hvt_tt1p05v25c.db"

Process:     Typical NMOS and PMOS transistors
Voltage:     1.05V
Temperature: 25℃
```

Timing Analysis Report

- Typically, a report for the most **critical path** is generated
- The report presents two paths:
  - **Data arrival time**: Delay of a data path from a launch clock edge through a register or input and combinational logic to a register or output
  - **Data-required** time computed from the capture clock edge and setup time requirement
- The difference between data required and data arrival time is called **slack**
- Slack should be positive or zero
- The optimizer tries to bring negative slack to zero on all paths by changing combinational logic structure

```
Point                                          Incr       Path
-----------------------------------------------------------------
clock clk (rise edge)                          0.00       0.00
clock network delay (ideal)                    0.00       0.00
state_r_reg_0_/CLK (DFFX1_HVT)                 0.00       0.00 r
state_r_reg_0_/Q (DFFX1_HVT)                   0.17       0.17 f
U19/Y (AND4X1_HVT)                             0.11       0.28 f
U23/Y (NOR3X0_HVT)                             0.12       0.40 r
U26/Y (AND3X1_HVT)                             0.11       0.51 r
U29/Y (OA222X1_HVT)                            0.13       0.64 r
state_r_reg_3_/D (DFFX1_HVT)                   0.01       0.65 r
data arrival time                                         0.65

clock clk (rise edge)                         10.00      10.00
clock network delay (ideal)                    0.00      10.00
state_r_reg_3_/CLK (DFFX1_HVT)                 0.00      10.00 r
library setup time                            -0.07       9.93
data required time                                        9.93
-----------------------------------------------------------------
data required time                                        9.93
data arrival time                                        -0.65
-----------------------------------------------------------------
slack (MET)                                               9.27
```

actual time jaidar modde data obossoi pouchaite hbe and aida nirdaron kore jee rule dewa ase ta,,like aii ckt aii poriman max time khabe and aii babe require max nirdaron hoi

References

1. Synopsys Timing Constraints User Guide