KU LEUVEN

Design of Digital Platforms

Exercise Session #2

Introduction to Gezel

Outline

- Register Transfer Level (RTL) hardware modeling

- GEZEL
  - Datapath
  - Controller
  - Testbench

Hardware Description

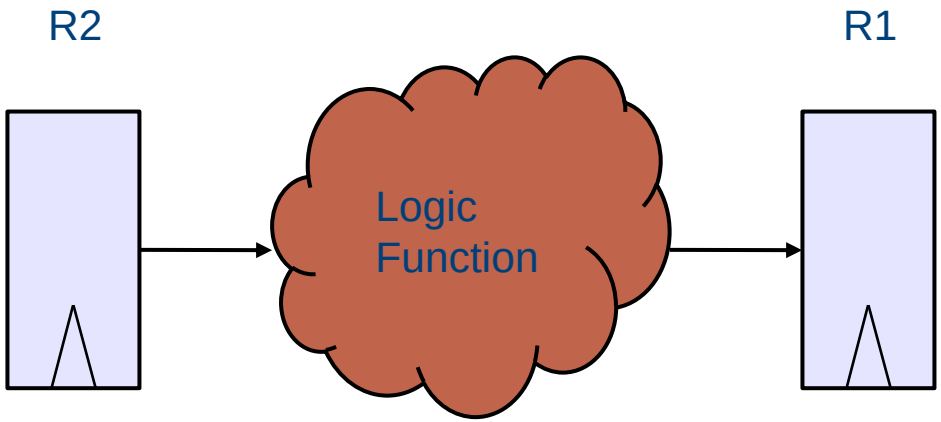
- Specific to implementation
  - Speed optimized
  - Area optimized

- Speed vs area trade off

- *Project Goal*: RSA coprocessor
  - Architectural possibilities:
    - Speed
    - Area
    - Flexibility

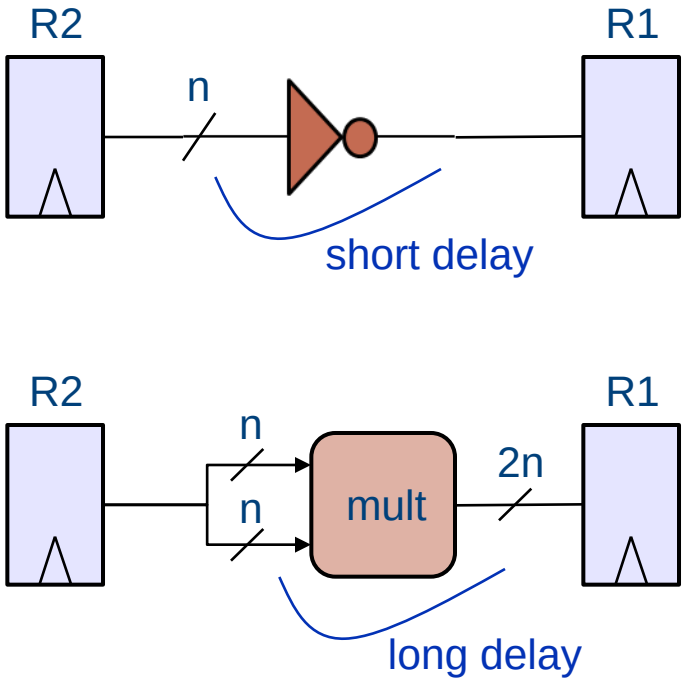
Register Transfer Level (RTL)

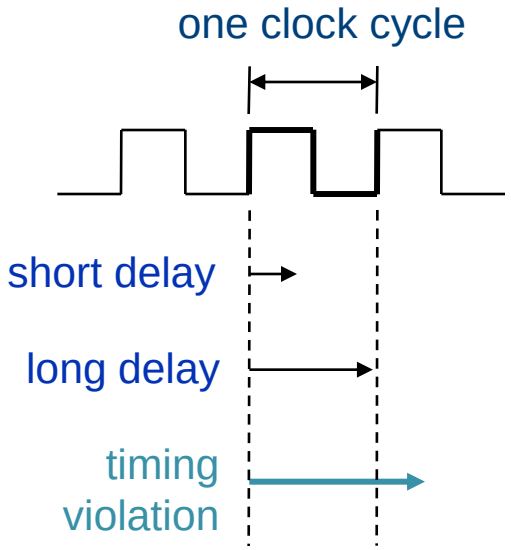
- Above gate level
- Time abstracted to clock cycles



Register Transfer Level (RTL)

- Technology independent
- Number of logic operations per cycle is technology dependent
- **Critical path** depends on largest delay between two registers
  - Determines max clock frequency



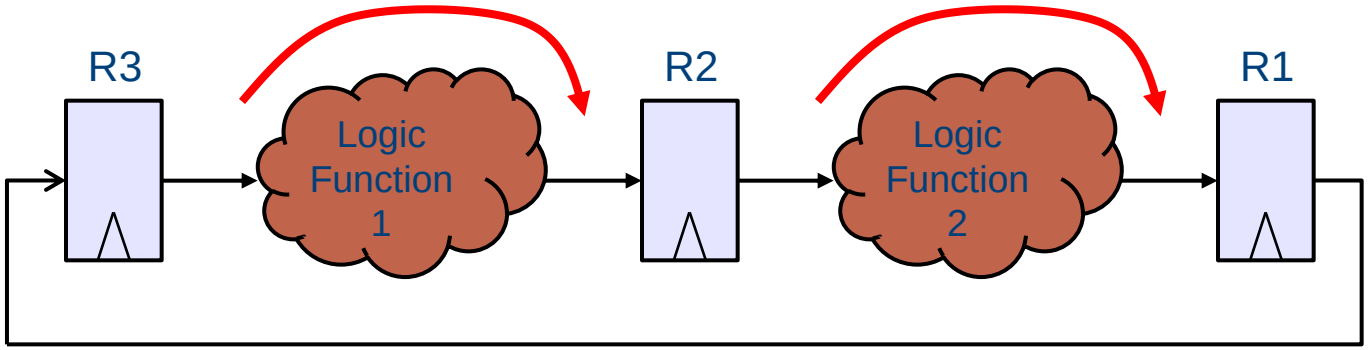


KU LEUVEN

RTL Computation

Computation steps happen in two phases:

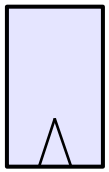
1. Logic Computation (Logic Function)
2. Update (Registers)



Language: GEZEL

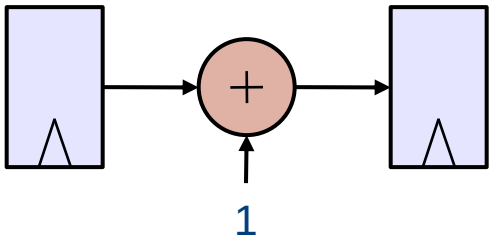
- Many languages available for RTL Modeling: VHDL, Verilog, SystemC are mainstream.
- We use GEZEL
  - http://rijndael.ece.vt.edu/gezel
- Anything you write in GEZEL, you can also write in VHDL, Verilog, SystemC
- GEZEL can also be translated (automatically) into VHDL

Datapaths

Registers  ***reg*** *r* : ns(n)

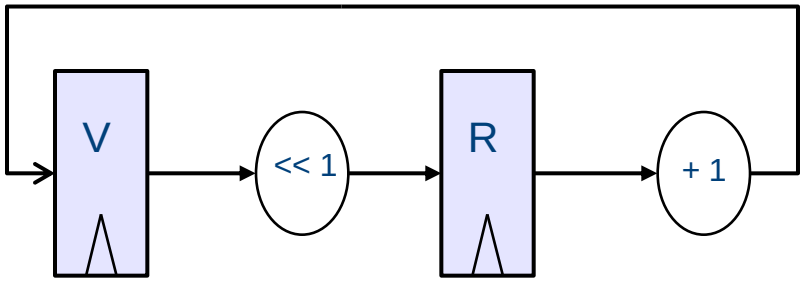
Wires ——————— **sig** *r :* ns(n)

Circuits



Expressions

Datapaths

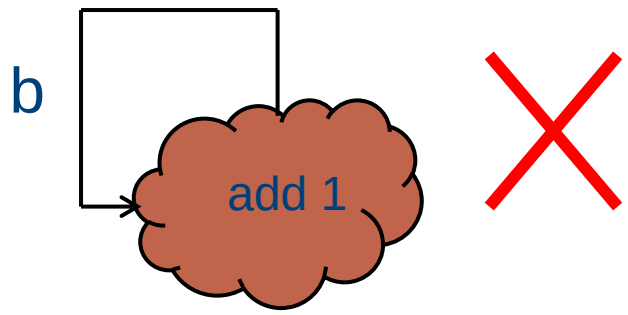



```
reg r : ns(3);
reg v : ns(3);

sfg run {
        v = r + 1;
        r = v << 1;
}
```

Rules

```
sig b : ns(3);
sfg do_always {
  b = b + 1;
}
```



RULES:

#1: No combinatorial loops between signals in a clock cycle

#2 Every output has to be assigned in each clock cycle

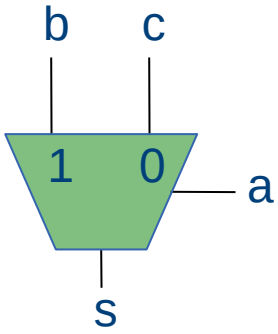
#3: A signal has to be assigned if used during a clock cycle

#4: Only one assignment during a clock cycle

GEZEL operators

Bitwise: **a | b, a & b, a ^ b, ~a**

Selection: **s = a ? b : c**



Arithmetic: **a + b, a - b, a * b, -a**

Arithmetic: **a << n, a >> n**

Comparison: **a < b, a > b, a <= b, a >= b, a != b, a == b**

Controllers

- Generates *control signals*
  - Multiplexing
  - Register update enable

- Datapath is useless without the controller

- Two types:
  - Combinatorial (Hardwired)
  - Finite State Machine with Datapath (FSMD)

Controllers: Hardwired

```
dp example1( in a_in, b_in, c_in : ns(1);
                    out   result  : ns(1))
{

   sfg run {
              result = (a & b) ^ c;
              }
}

hardwired hw_control(example1){run;}
```

Controllers - Finite State Machine with Datapath (1)

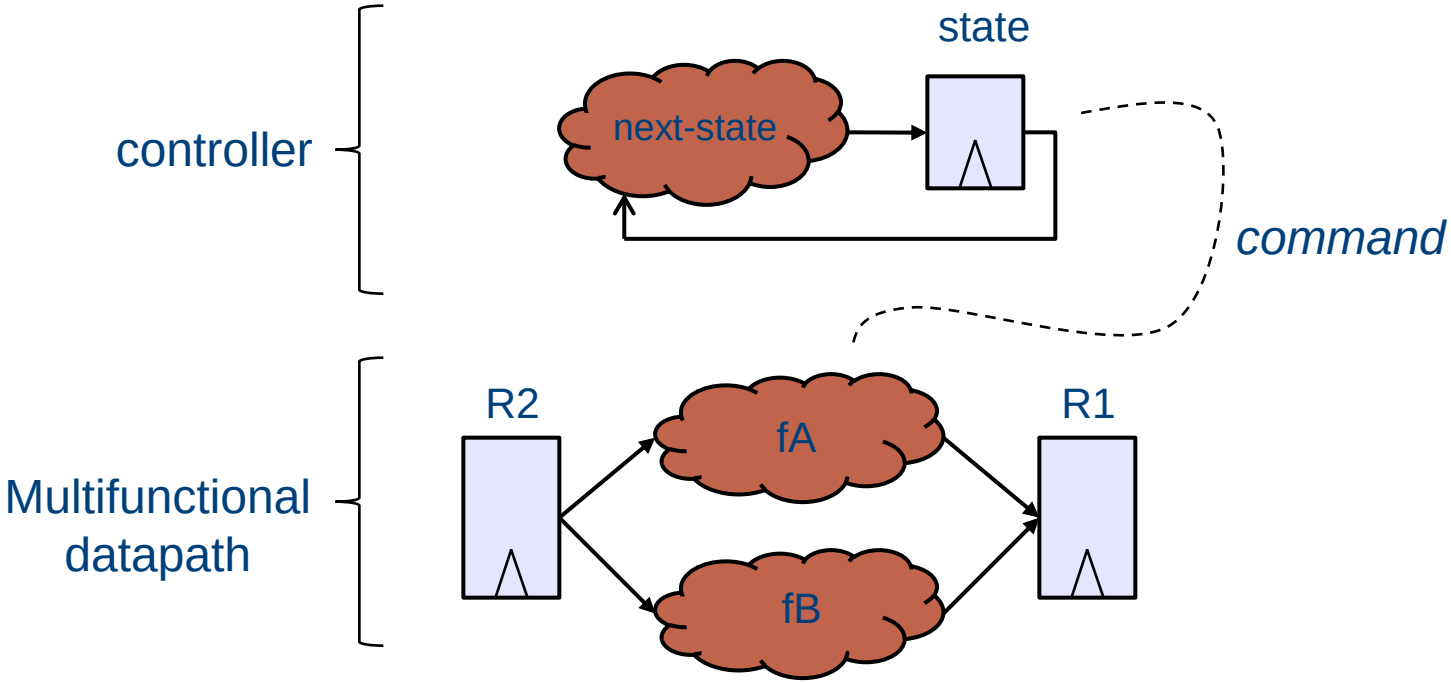
Basic idea: Controller and datapath exist in two separate modules.

Controller:

- State-machine
- Controls the program flow

Datapath:

- Performs data processing operations



Controllers - Finite State Machine with Datapath (2)

```
dp count(in a_in : ns(10);   out result : ns(10)) {
reg a_reg : ns(10);
sfg load {
        a_reg = a_in;
        }
sfg work {
        a_reg = (a_reg < 999) ? (a_reg + 1) : 0;
        }
                ...
}

fsm ctl_count(count) {
  initial s0;
  state s1;
  @s0 (load) -> s1;
  @s1 (work) -> s1;
}
```

See: fsmd.fdl

Controllers - Finite State Machine with Datapath (3)

Tips for writing FSMDs:

- Avoid writing multiplexers (a ? b : c) inside the state machine
- Data processing and control sequencing should be separated

Simulation Directives

$display(arg, ..)	Used inside of an sfg. Prints strings, expressions and meta-variables.
$cycle	Used as a $display argument. Returns current clock cycle (first cycle = 1).
$sfg	Used as a $display argument. Returns the name of the current sfg.
$dp	Used as a $display argument. Returns the name of the current datapath.
$finish	Used inside of an sfg . Aborts the simulation.

Simulation

- **fdlsim**

fdlsim [d] [<design.fdl>] <cyclecount>

termination :

a. target cycle count is reached

b. runtime error

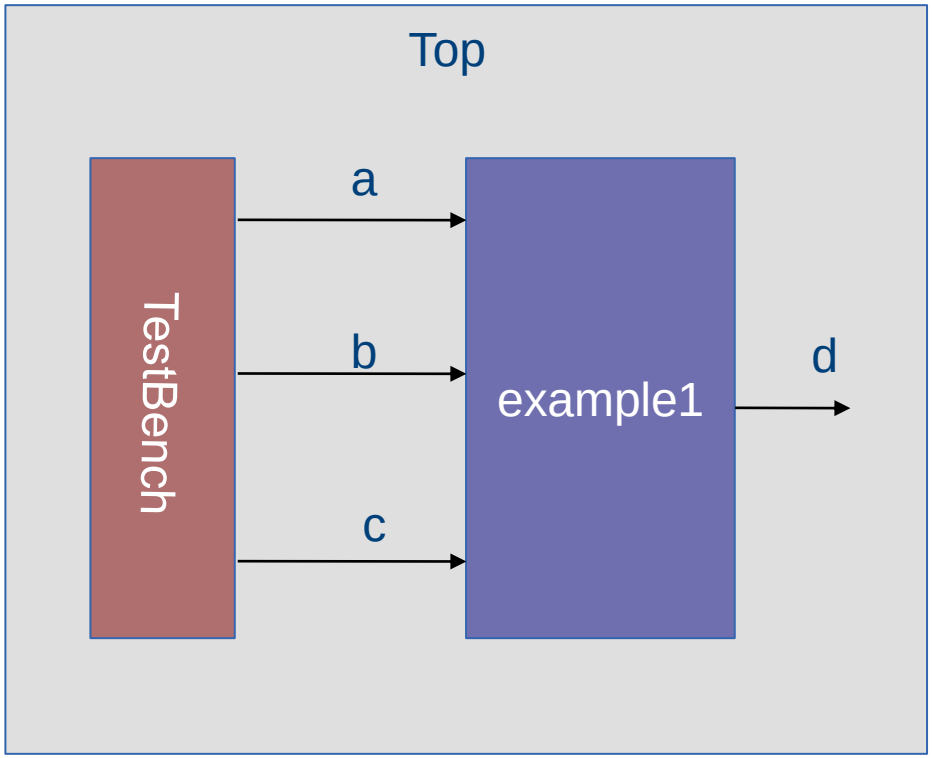
c. $finish directive is executed

- **-d: debug mode**

Example: fdlsim counter.fdl 20

Test Bench

Only used for simulation



KU LEUVEN

Test bench

```
dp TestBench( out a_out,
                    b_out,
                    c_out :ns(1))
{
  sfg run{
          a_out = 0b1;
          b_out = 0b1;
          c_out = 0b0;
  }
}

system s{
  top;
}

hardwired hw_control(TestBench){run;}
```

```
dp top(){
        sig a,b,c,d :ns(1);

        use TestBench(a,b,c);
        use example1(a,b,c,d);
}
```

See: testbench.fdl