RTL Coding Guidelines

Michael Keating & Pierre Bricaud

Reuse Methodology Manual, for SoC designs, 2nd

Tzung-Shian Yang

VLSI Signal Processing Group

Department of Electronics Engineering

National Chiao Tung University

Outline

- Basic coding practices
- Coding for portability
- Guidelines for clocks and resets
- Coding for synthesis
- Partitioning for synthesis
- Conclusion

Basic Coding Practices

Basic Goal

- Develop RTL code simple and regular
  - Easier to design, code, verify and synthesize
- Consistent coding style, naming conventions and structure
- Easy to understand
  - Comments, meaningful names and constants or prarameters instead of hard-coded numbers

Naming Convention

- Lowercase letters for all signal, variable and port names

  e.g. wire *clk*, *rst*;
- Uppercase letters for names of constants and user-defined types

  e.g. `define MY_BUS_LENGTH 32
- Meaningful names
  - For a RAM address bus, *ram_addr* instead of *ra*

Naming Convention (2)

- Short but descriptive
  - During elaboration, the synthesis tool concatenates the names
- *clk* for clock signal,
  - More than one clock? → *clk1*, *clk2* or *clk_interface* ...
- Active low signals: postfix with '*_n*'
- *rst* for reset signals, if active low→*rst_n*

Naming Convention (3)

- When describing multibit buses, use a consistent ordering of bits
  - For Verilog: use <u>(x:0)</u> or (0:x)
- The same or similar names for ports and signals that are connected (e.g. *a=>a* or *a=>a_int*)

Naming Convention (4)

- Other naming conventions
  - **_r* : output of a register
  - **_a* : asynchronous signal
  - **_z* : tristate internal signal
  - **_pn* : signal used in the *n*th phase
  - **_nxt* : data before being registered into a register with the same name

Include Header

- e.g.

```
/ *This confidential and proprietary
  *software ...
  * © copyright 1996 Synopsys inc.
  *File          : Dwpci_core.v
  *Author    : Jeff Hackett
  *Date       : mm/dd/yy
  *Version   : 0.1
  *Abstract : ...
  *Modification History : date, by who,
                    version,change description ... */
```

Comment

- Placed logically, near the code that describe
- Brief, concise and explanatory
- Separate line for each HDL statements
- Keep the line length to 72 characters or less

always @(*a* or *b* or *c* or *d* or *e* or *f* or *g* or
  h or *i*)

Indentation

- Improve the readability of continued code lines and nested loops, tab of 4 is recommended

```
if(a)
    if(b)
        if(c)
          …
```

- Port ordering

```
module my_module(clk, rst, ...);
{        // Inputs:
    clk, // comment for each
    rst, // ...
    ...
         // Outputs:
    ...
}
```

Port Map

- Use named association rather than positional association

```
my_module U_my_module(
   .clk(clk),
  .rst(rst),
  …
);
```

Use Function

```
function [`BUS_WIDTH-1:0] convert_address;
    input input_address, offset;
    begin
        // ... function body
    end
    endfunction //convert_address
```

Use Loop and Array

```
module my_module( … );
…
reg [31:0] reg_file[15:0];
integer tmp;

always @(posedge clk or posedge rst)
if(rst)
    for(tmp=0; tmp<16; tmp++)
        reg_file[tmp] <= 32'd0;
…
```

Meaningful Label

- Helpful for debug
  - Label each process block <*name*>_PROC
  - Label each instance U_<*name*>

Coding for Portability

Portability

- Create code technology-independent, compatible with various simulation tools and easily translatable between Verilog and VHDL
- Constants instead of hard-coded value

```
`define MY_BUS_SIZE 8
reg [`MY_BUS_SIZE-1:0] my_out_bus ;
```

- Keep the `define statements for a design in a single separate file and name the file *DesignName*_params.v

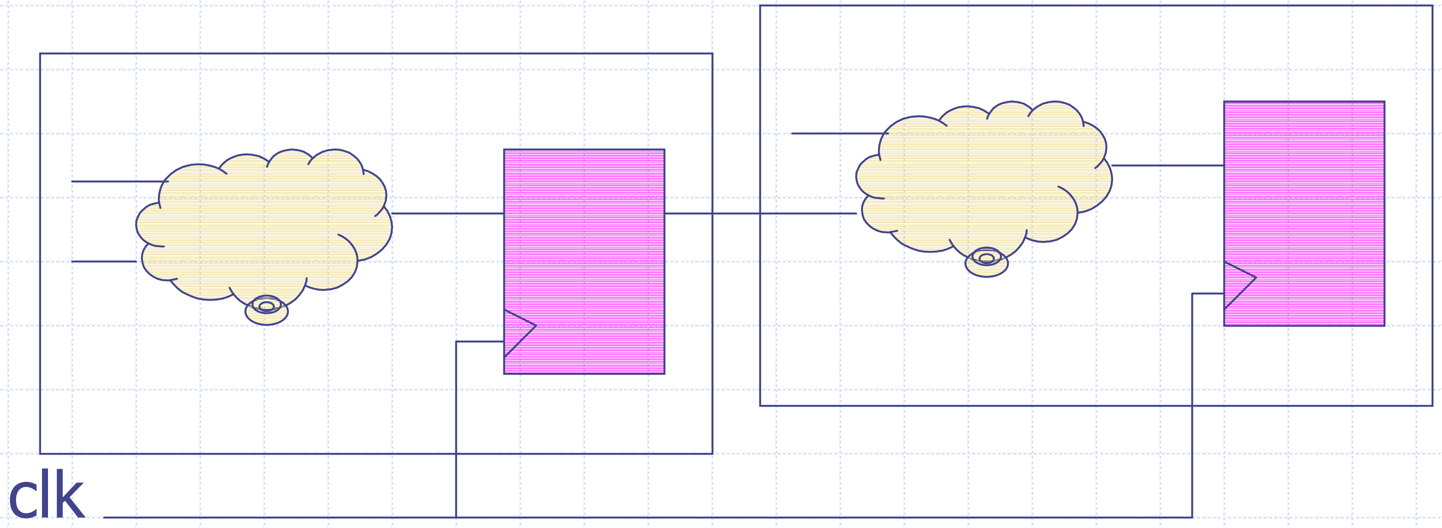
Technology Independent

- Avoid embedding dc_shell scripts
  exception: the synthesis directives to turn synthesis on and off must be embedded in the code in the appropriate places
- Use *DesignWare* Foundation Libraries
- Avoid instantiating gates
- If you must use technology-specific gates, then isolate these gates in a separate module

Guidelines for Clocks and Resets

Clock and Reset

- Simple clocking structure: a single global clock and positive edge-triggered flops as the only sequential devices



Mixed Clock Edges

- Avoid!
  - Duty cycle of the clock become a critical issue in timing analysis
  - Most scan-based testing methodologies require separate handling of positive and negative-edge triggered flops
- If you must use both,
  - Model the worst case duty cycle
  - Document the assumed duty cycle
  - If you must use many, separate them into different modules

Clock Buffer

- Avoid hand instantiating clock buffers in RTL code. They are normally inserted after synthesis as part of the physical design.
- Avoid internally generated clocks and resets, all the registers in the macro should be reset at the same time

Gated Clock

- Avoid coding gated clocks in RTL
  - Cannot be made part of a scan chain
  - If you must use, keep the clock and/or reset generation circuitry as a separate module at the top level of the design or model it using synchronous load registers e.g.

```
always @(posedge clk)
   if(p1_gate)
          …
```

Coding for Synthesis

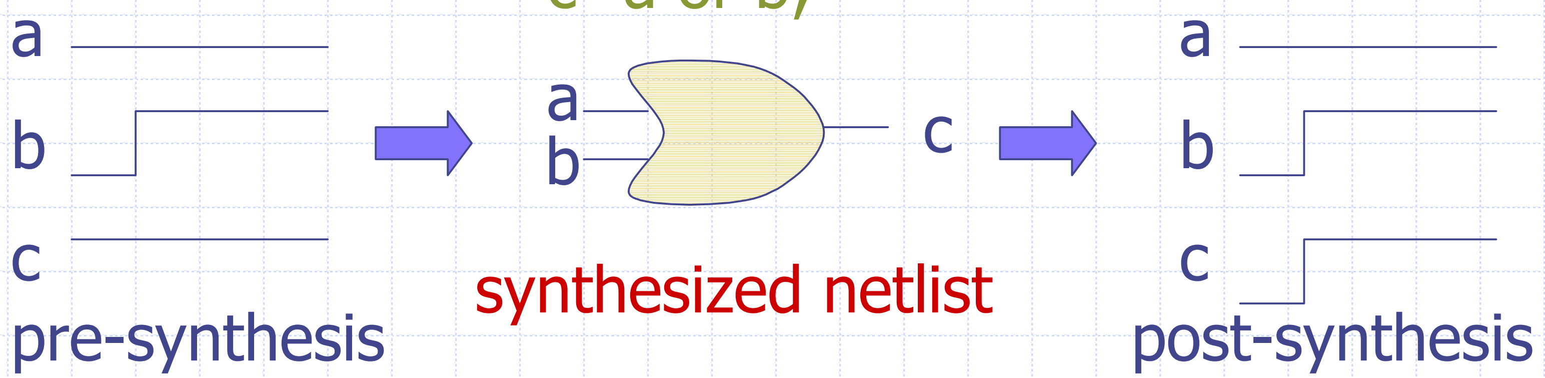
Infer Register

- Use reset signal to initialize registered signals instead of use an `initial` statement
- Avoid using any latches
  - Assign default value for all path
  - Assign outputs for all input conditions
  - Use else for the final priority branch
- Avoid combinational feedback

Sensitivity List

- Specify complete sensitivity list avoid difference between pre-synthesis and post-synthesis netlist in combinational blocks
- Include clock and reset in sequential blocks
- Avoid unnecessary signals in list

```
always @(a)
c=a or b;
```



Case v.s. if-then-else

- A case statement infers a single-level multiplexer, while an if-then-else one infers a priority-encoded, cascaded combination of multiplexers
- Synopsis directive about case statement

  case (*sel*) // synopsis parallel_case full_case
- if-then-else can be useful if you have a late arriving signal
- For large multiplexers, case is preferred because faster in cycle-based simulator
- Conditional assignment:

  e.g. assign *z1*=(*sel_a*) ? *a* : *b* ;

State Machines

- Separate state machines into two processes:combinational and sequential
  - Poor coding style:

```
always @(posedge clk)
  a <= b+c;
```

  - Recommended coding style:

```
always @(b or c)
  a_nst = b+c;
always @(posedge clk)
  a <= a_nst;
```

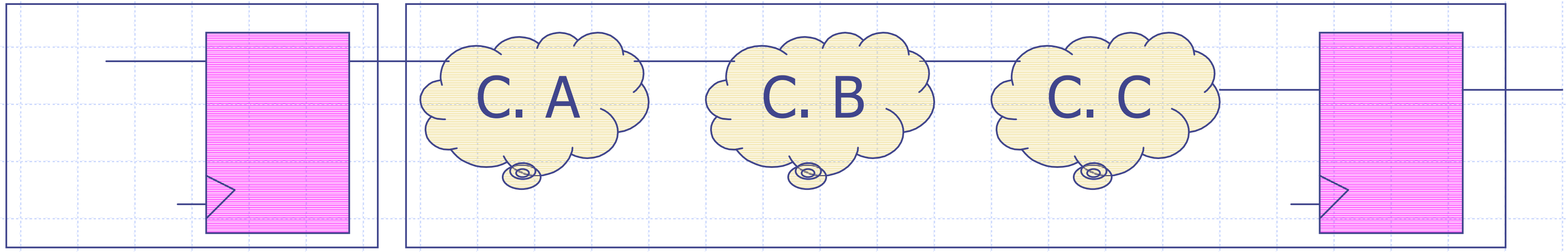
Partitioning for Synthesis

Partition for Synthesis

- Good partition provides advantages:
  - Better synthesis results
  - Faster compile runtimes
  - Ability to use simpler synthesis strategies to meet timing

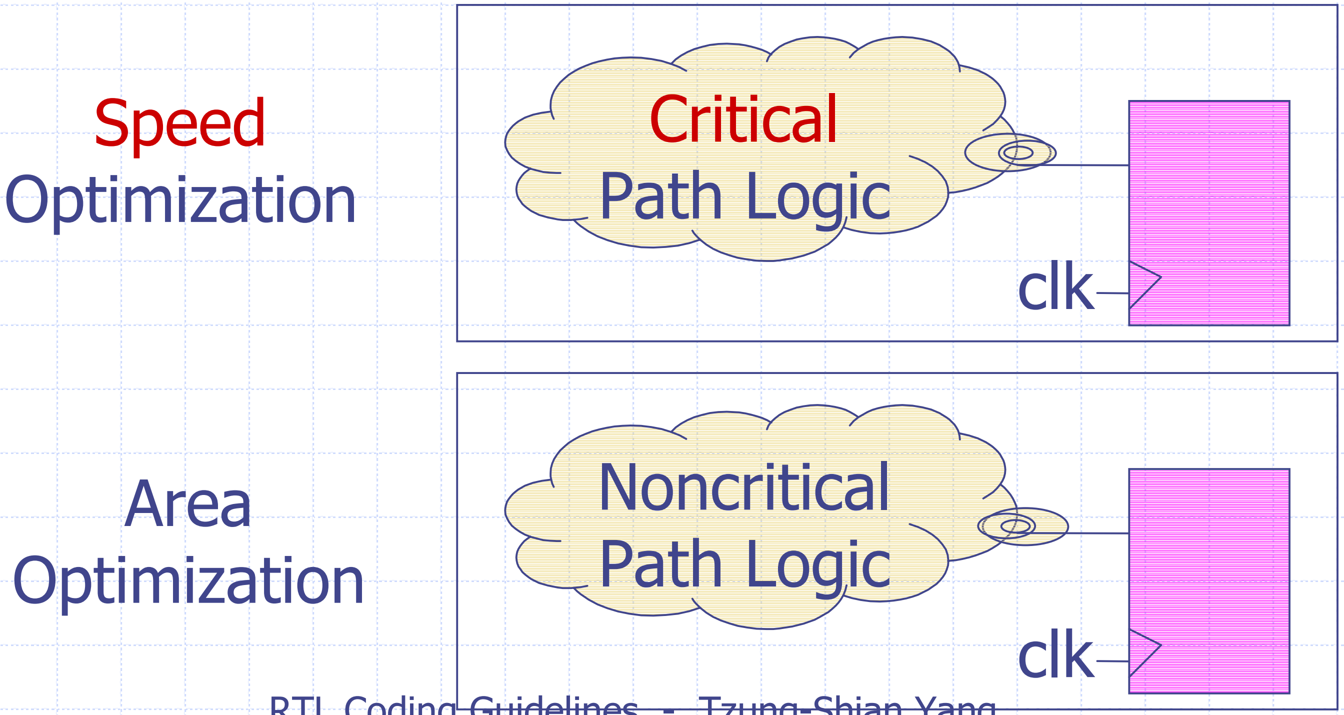
Register Output

- For each block of a hierarchical design, register all output signals
  - Output drive strengths equal to the drive strength of the average flip-flop
  - Input delays predictable
- Keep related combinational logic together in the same module



Different Design Goals

- Keep critical path logic in a separate module, optimize the critical path logic for speed while optimizing the noncritical path logic for area



Asynchronous Logic

- Avoid asynchronous logic
  - If required, partition in a separate module from the synchronous logic
- Merging resources
  - mux input than mux output for complicated processing unit

Partition for Synthesis Runtime

- Most important considerations in partition: logic function, design goals and timing and area requirements
- Grouping related functions together
- Eliminate glue logic at the top level

Conclusion

- Practice makes perfect