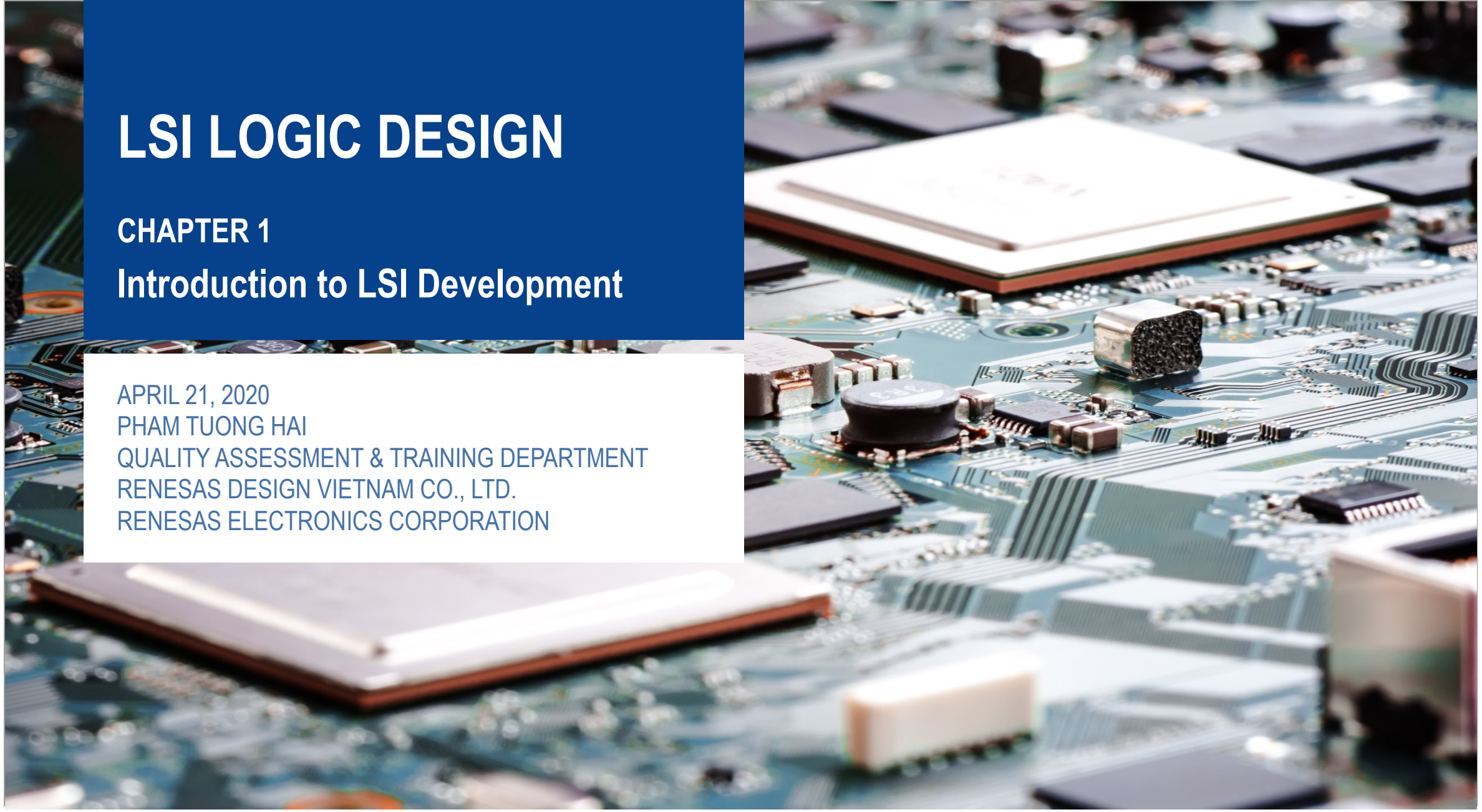
LSI LOGIC DESIGN

CHAPTER 1

Introduction to LSI Development



APRIL 21, 2020
PHAM TUONG HAI
QUALITY ASSESSMENT & TRAINING DEPARTMENT
RENESAS DESIGN VIETNAM CO., LTD.
RENESAS ELECTRONICS CORPORATION

CHAPTER 1. Introduction to LSI Development

1.1. Semiconductor Products.

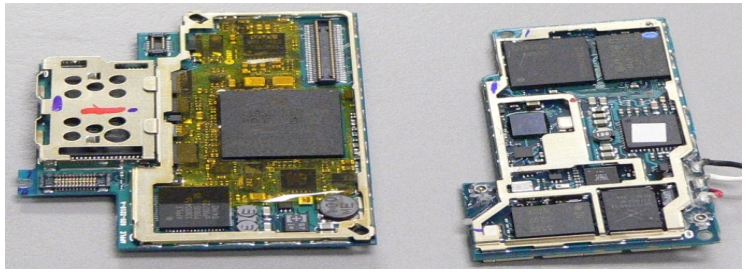
1.2. LSI Development Flow.

1.2. LSI Design Flow and Methodologies.

1.4. An Example.

1.1 Semiconductor Products

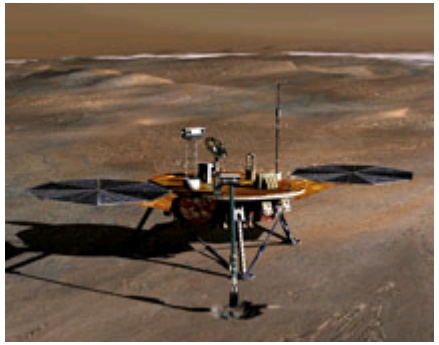
Micro Controller
Memory, Game
Data Communicate
Windows (Software)



Mobile Phone
E-mail, E-tag, Music
Camera, Digital-TV

PC

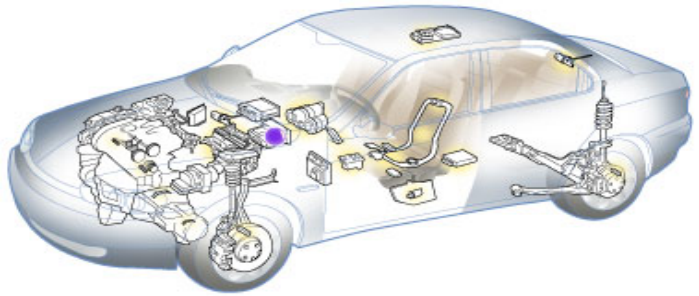
Mobile



Life

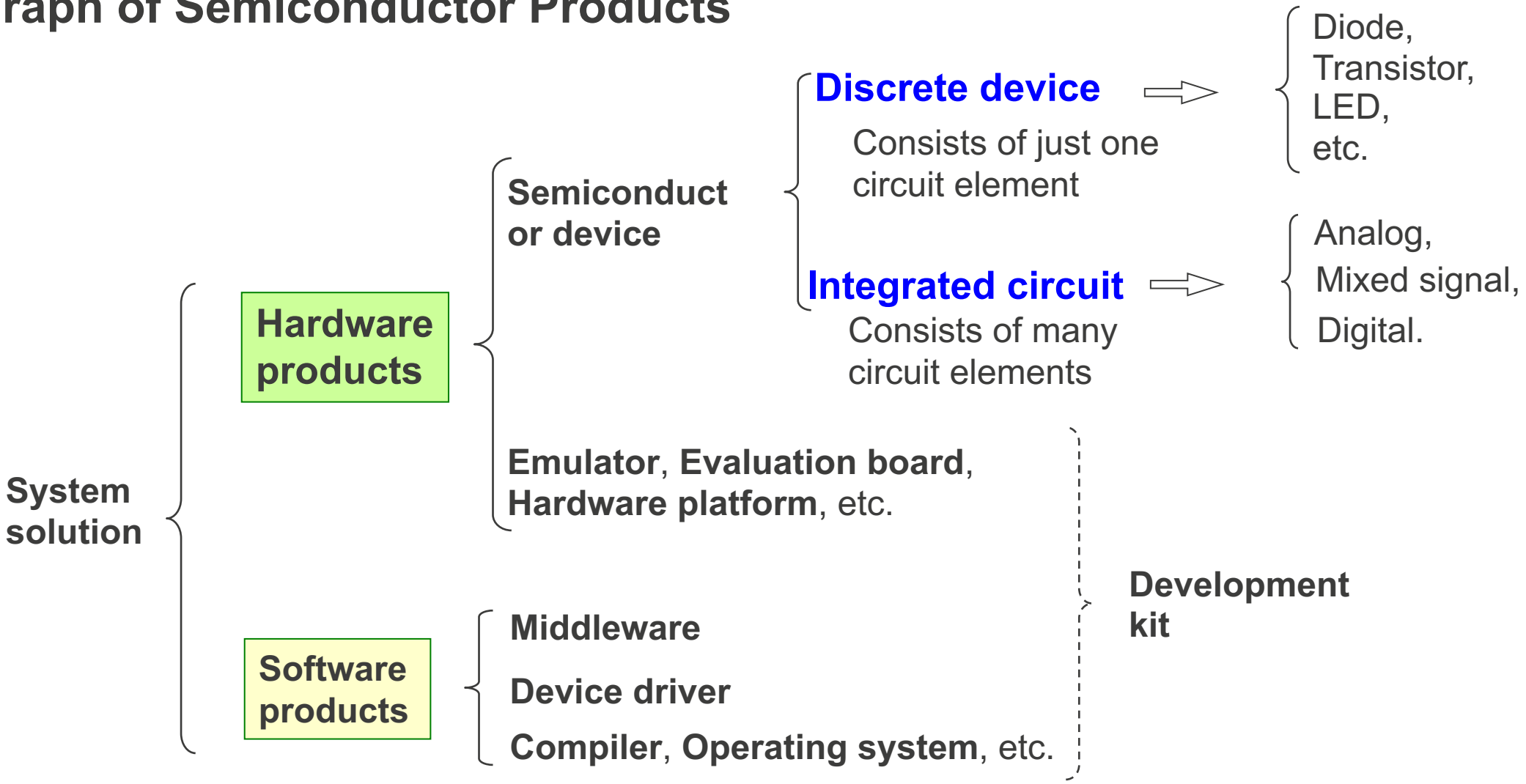
Car

Pace Maker（Health）
Robot
Sensor, Monitor
Environmental



Hybrid Car
Navigation
Anti-crash Control
Gas Control

Graph of Semiconductor Products



Classification of IC

Integration Scale

low → high

SSI MSI LSI VLSI ULSI

Small Scale Integration, Medium, Large, Very, Ultra

Integration level

SOC (System-on-a-chip)

SiP (System in Package)

Functional characteristic

Memory device

Logic device

Classification of IC from the view point of user

General purpose

Standard product

DRAM, SRAM, FPGA, etc.

ASSP (Application Specific Standard Product)

DSP, Embedded type micro processors, etc.

ASCP (Application Specific Custom Product)

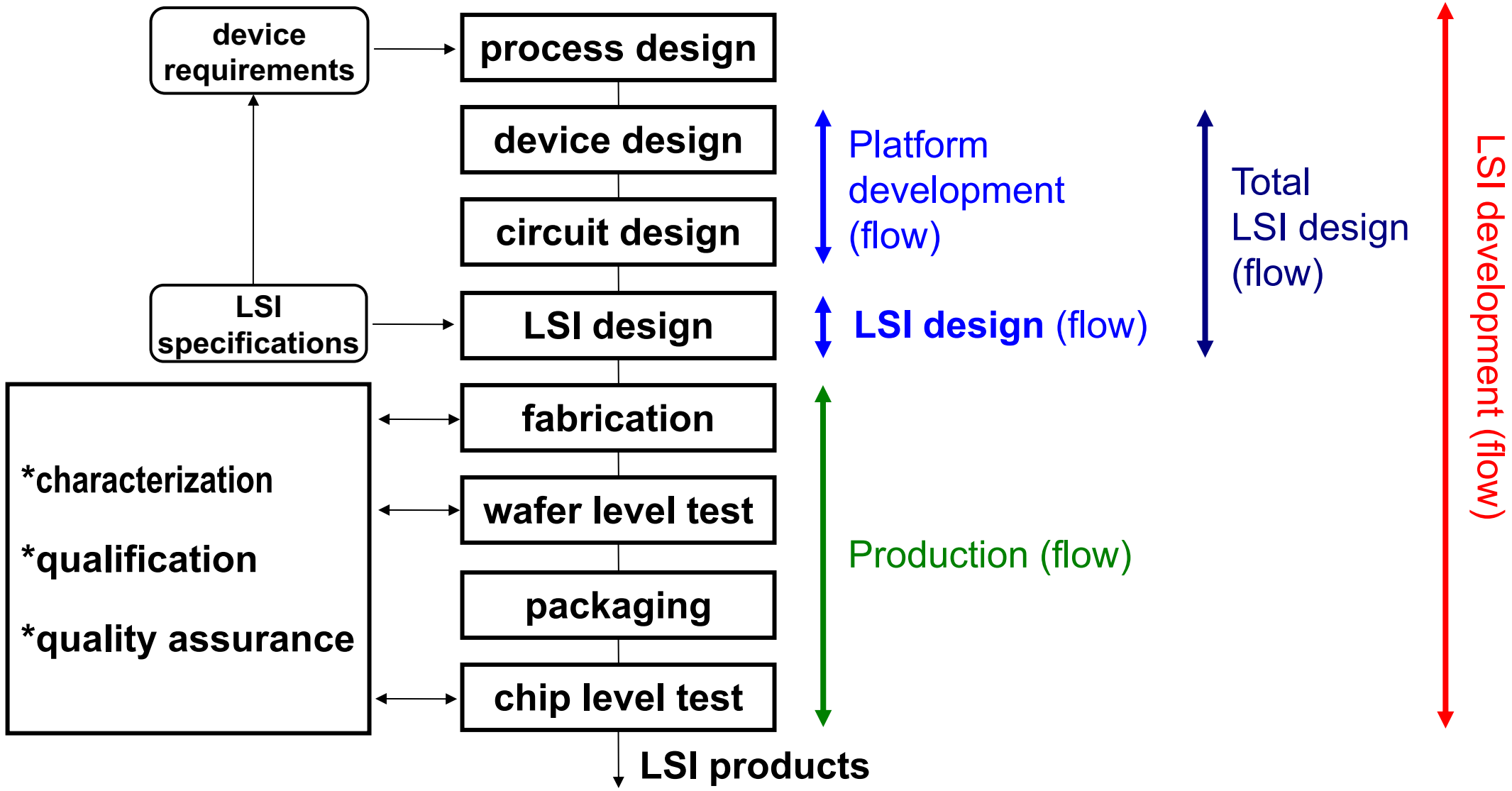
ASIC (Application Specific Integrated Circuit)

Specific application

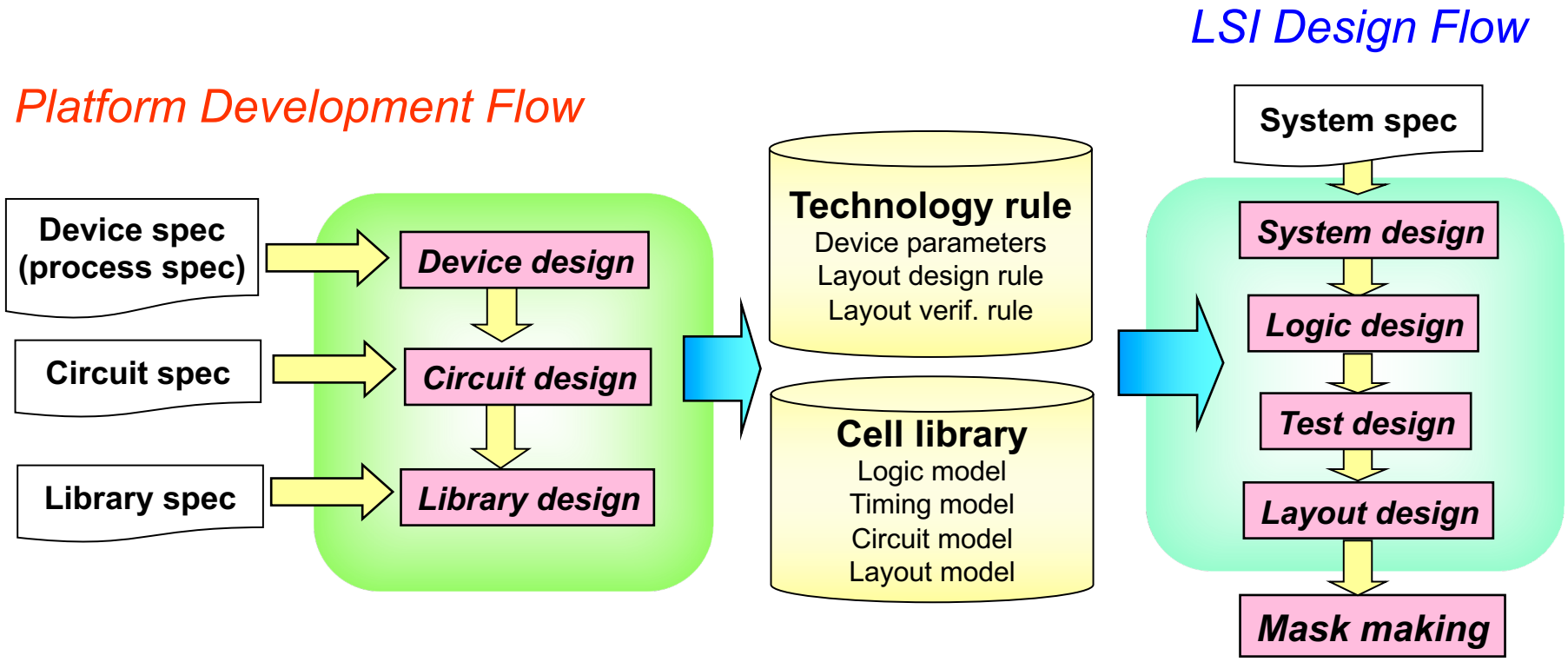
off-the-shelf components

1.2 LSI Development Flow

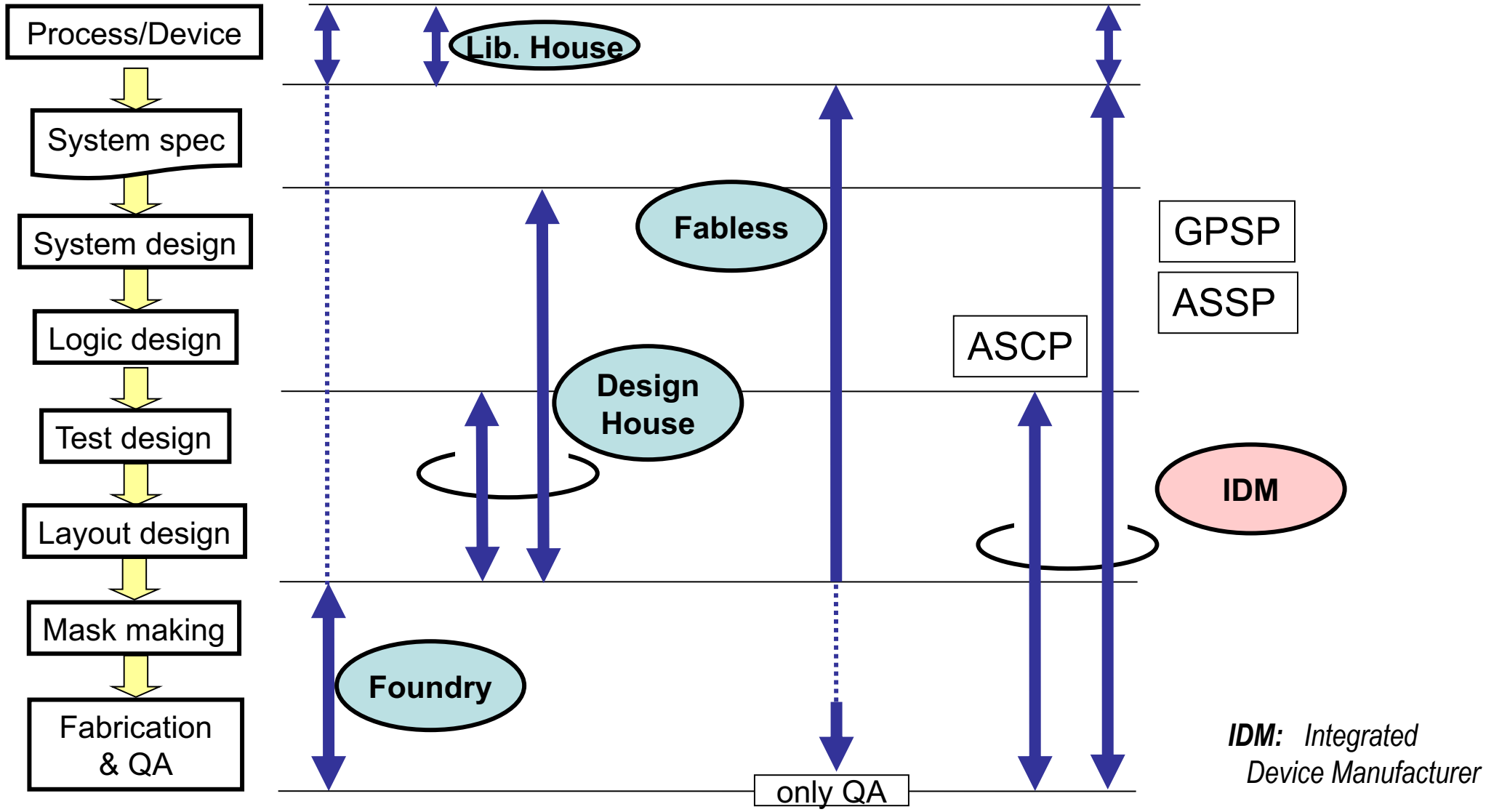
LSI Development Flow



Total LSI Design Flow

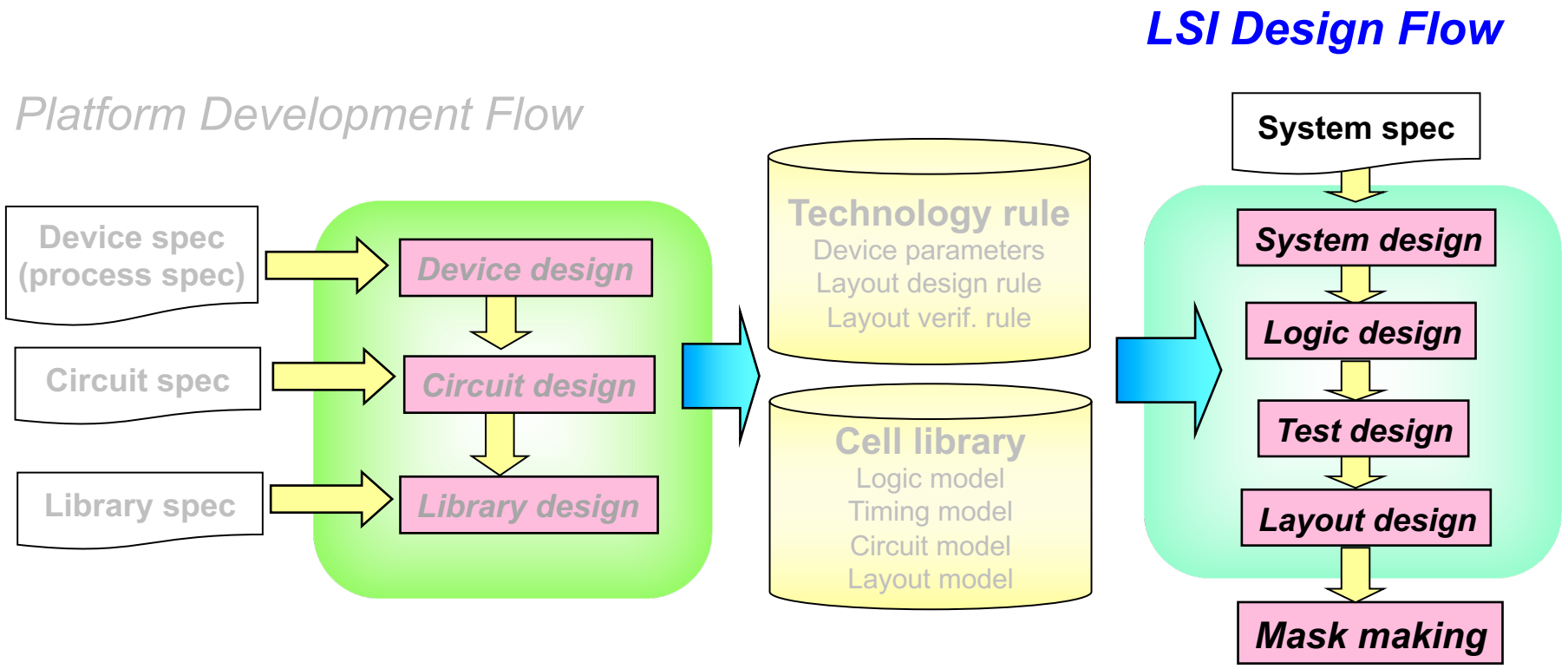


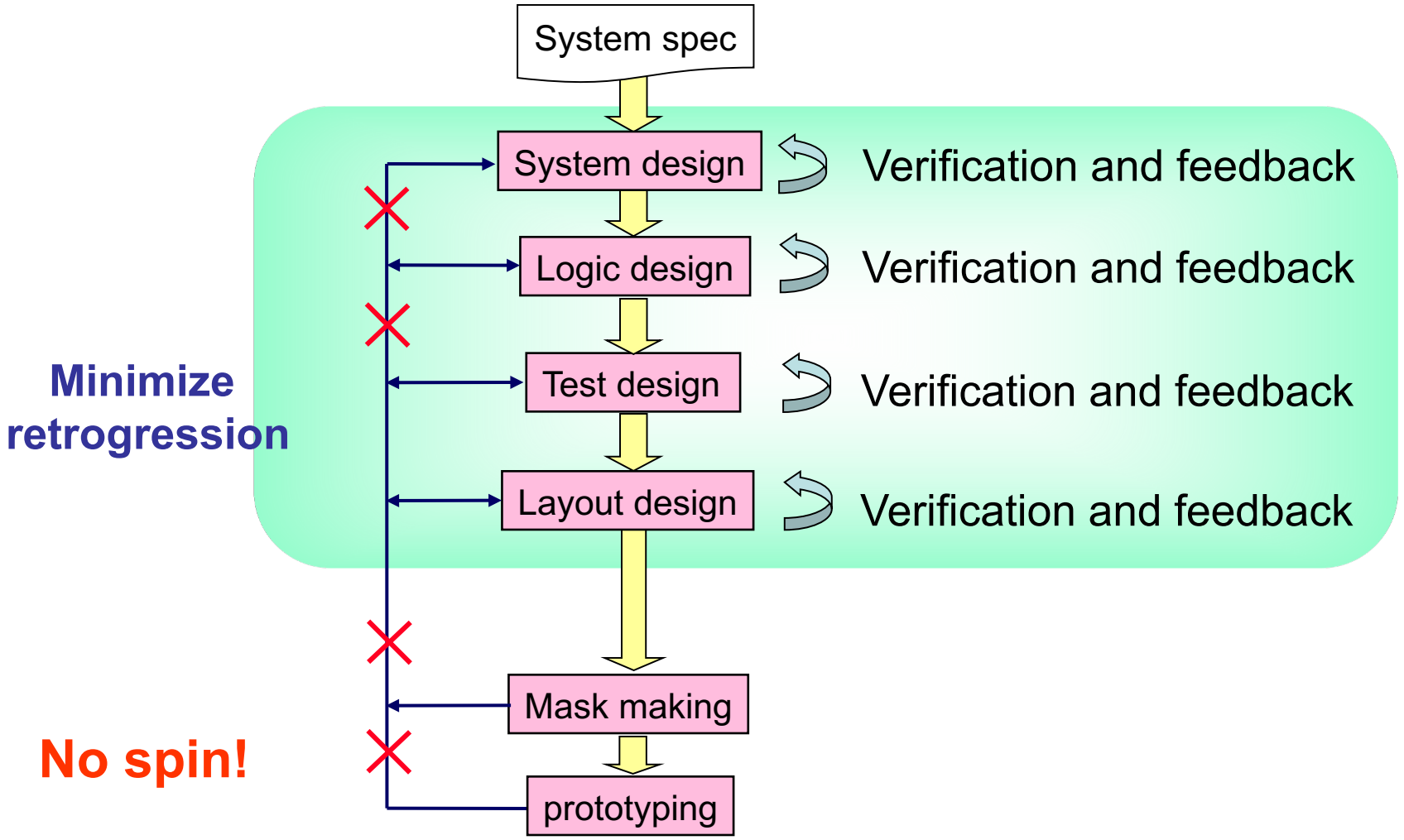
LSI Development and Business Models



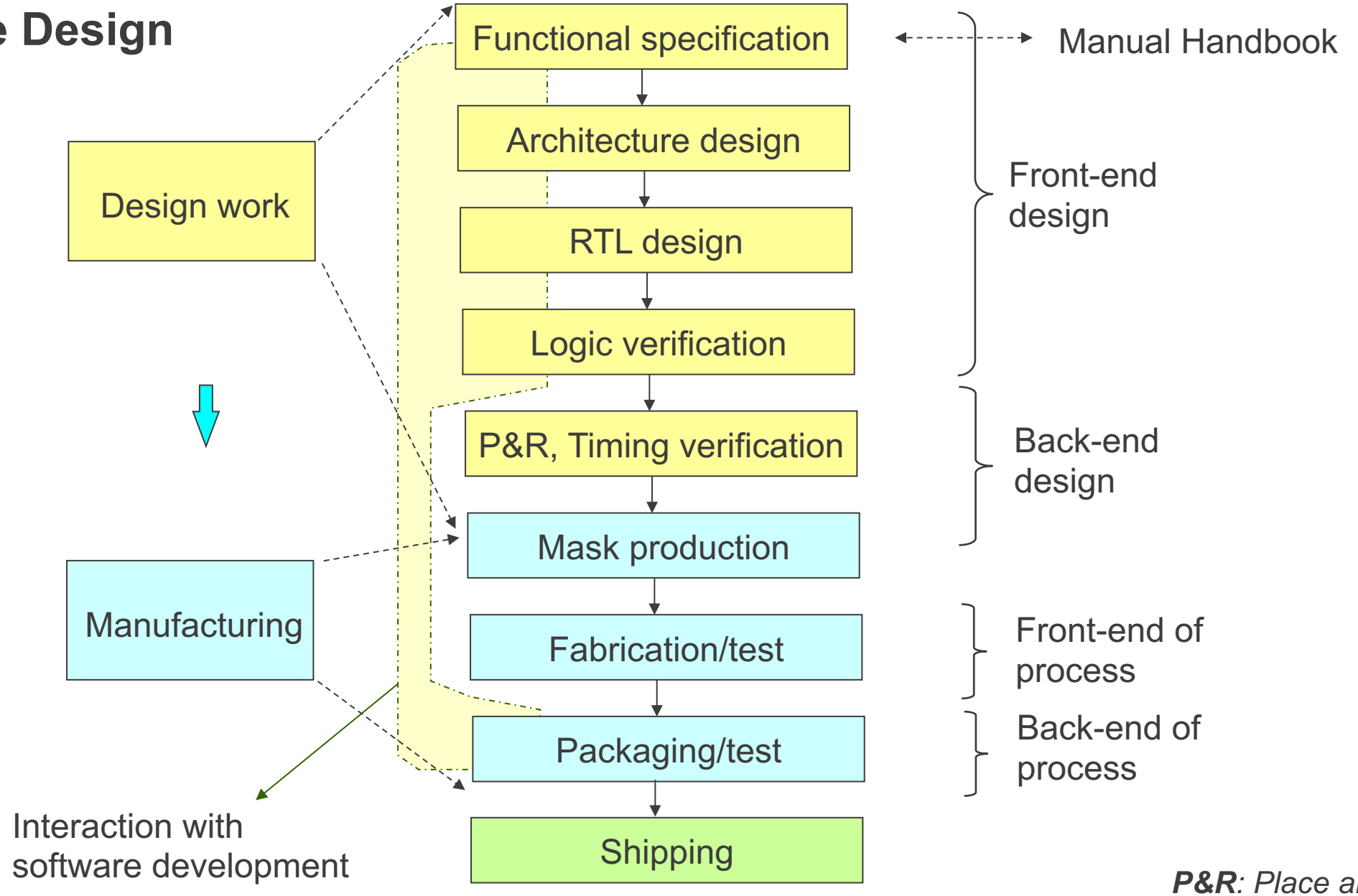
1.3 LSI Design Flow and Methodologies

LSI Design Flow

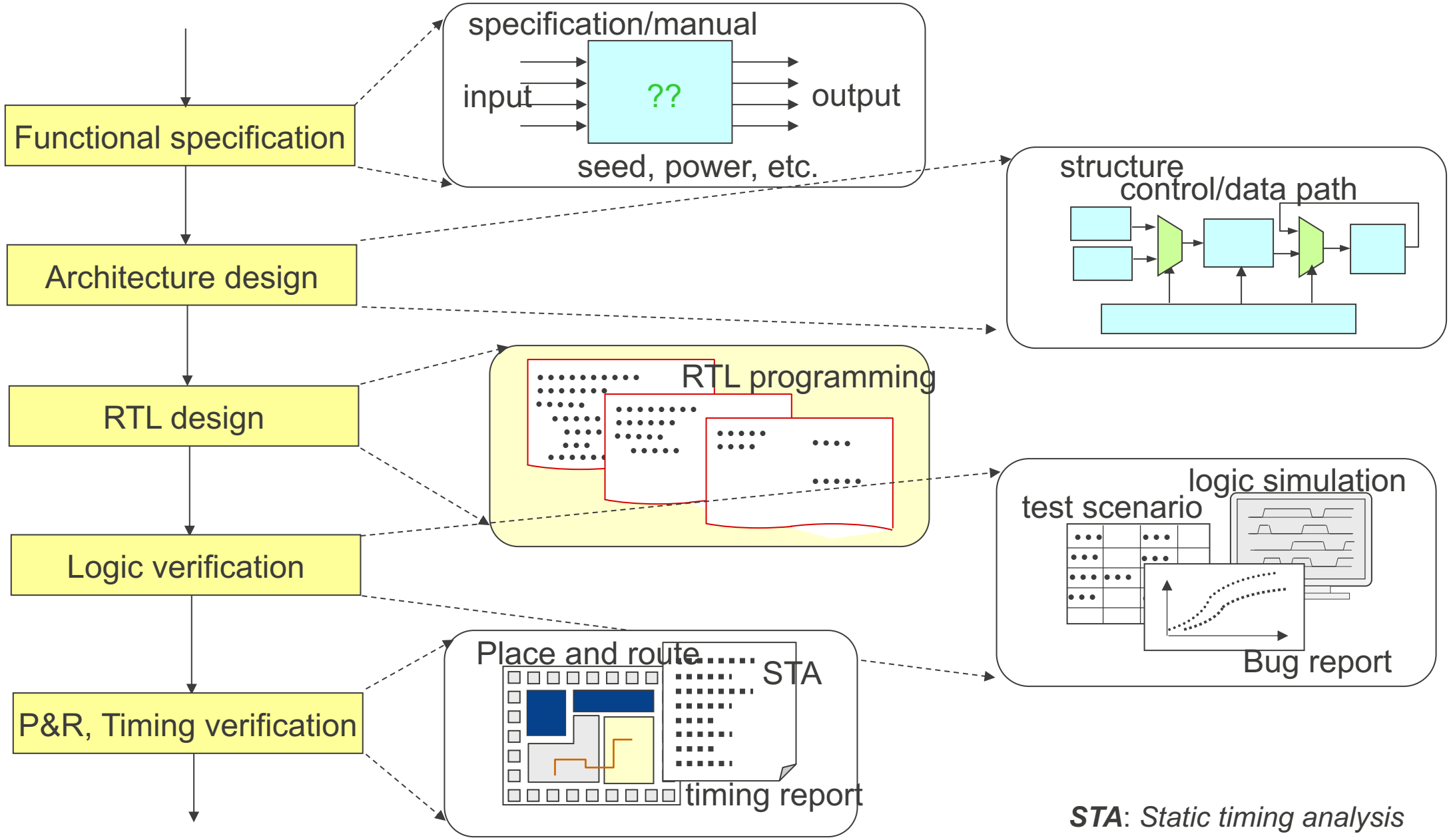



System spec
System design
Logic design
Test design
Layout design
Mask making
prototyping
Verification and feedback
Verification and feedback
Verification and feedback
Verification and feedback
Minimize retrogression
No spin!

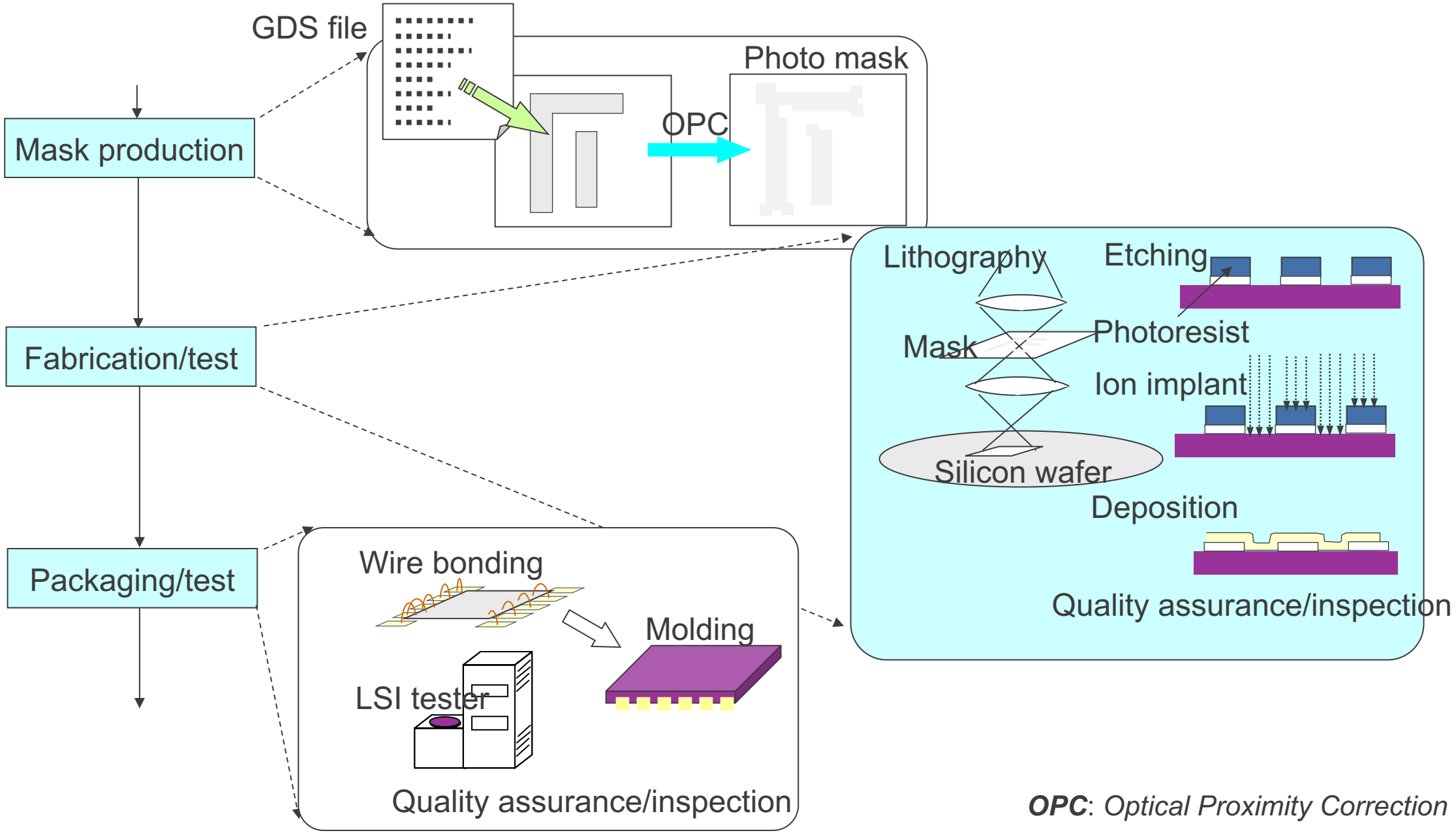
Hardware Design



***P&R**: Place and Route*

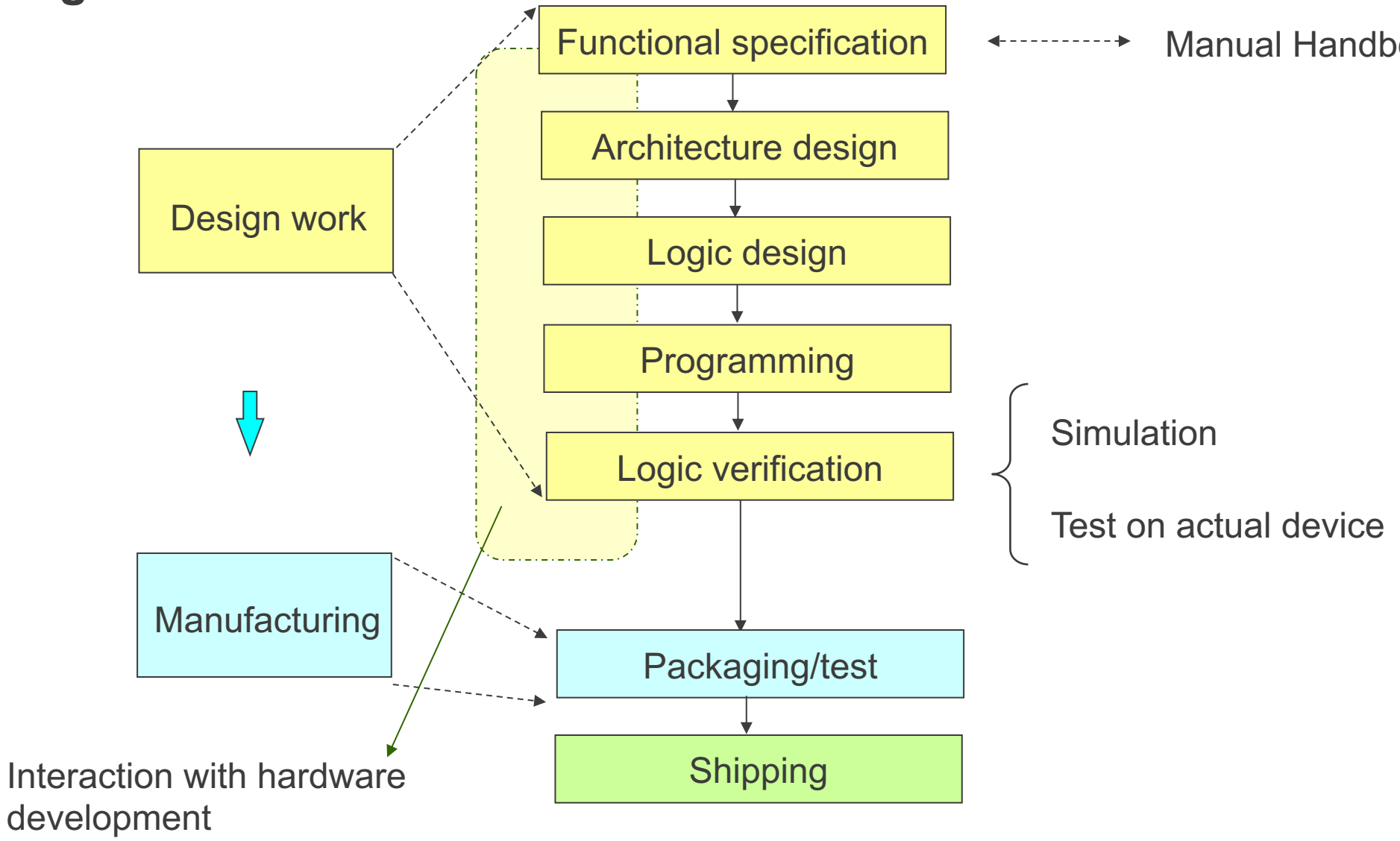


STA: *Static timing analysis*

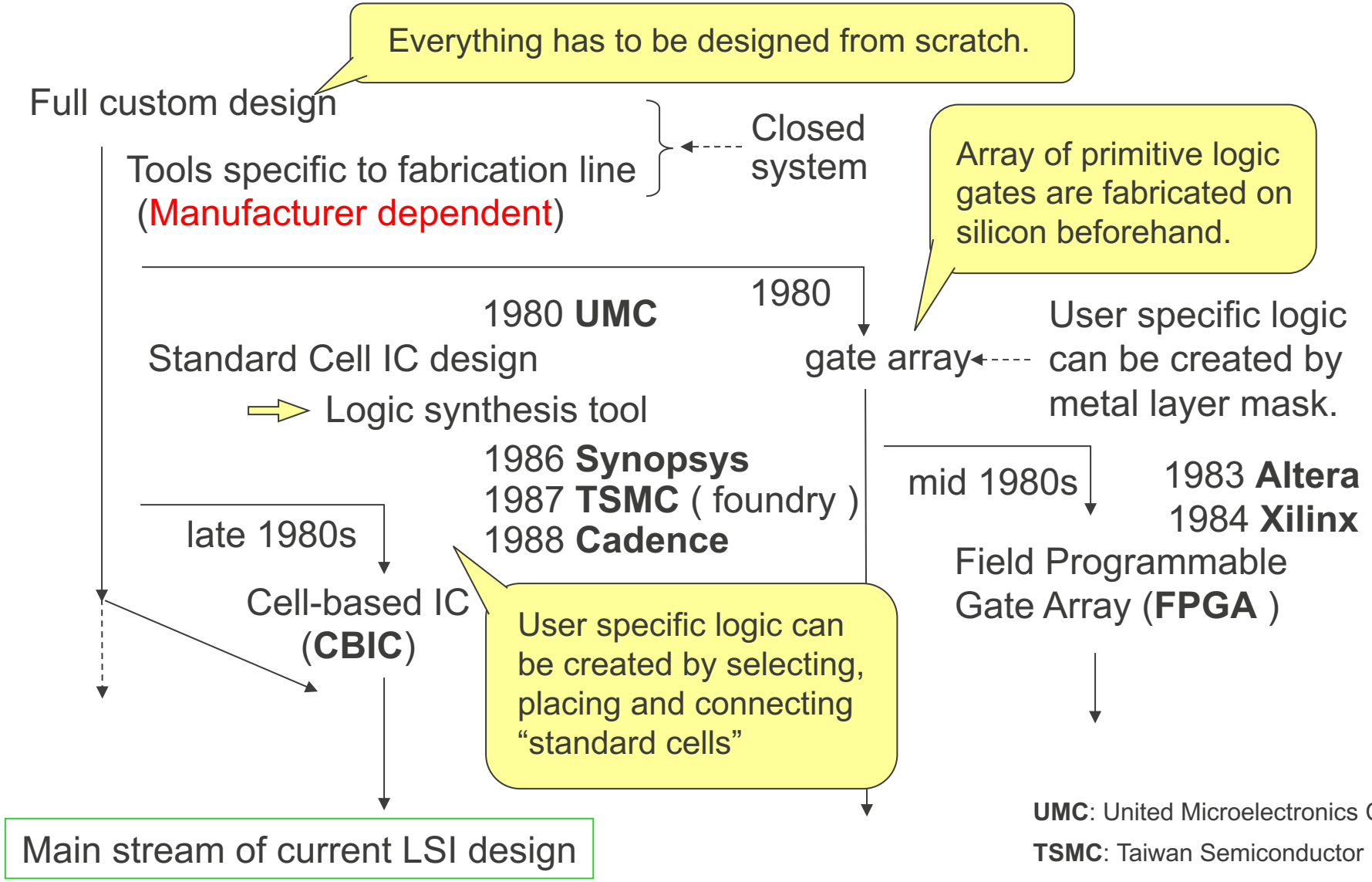


OPC: *Optical Proximity Correction*

Software Design



History of LSI Design Methodologies



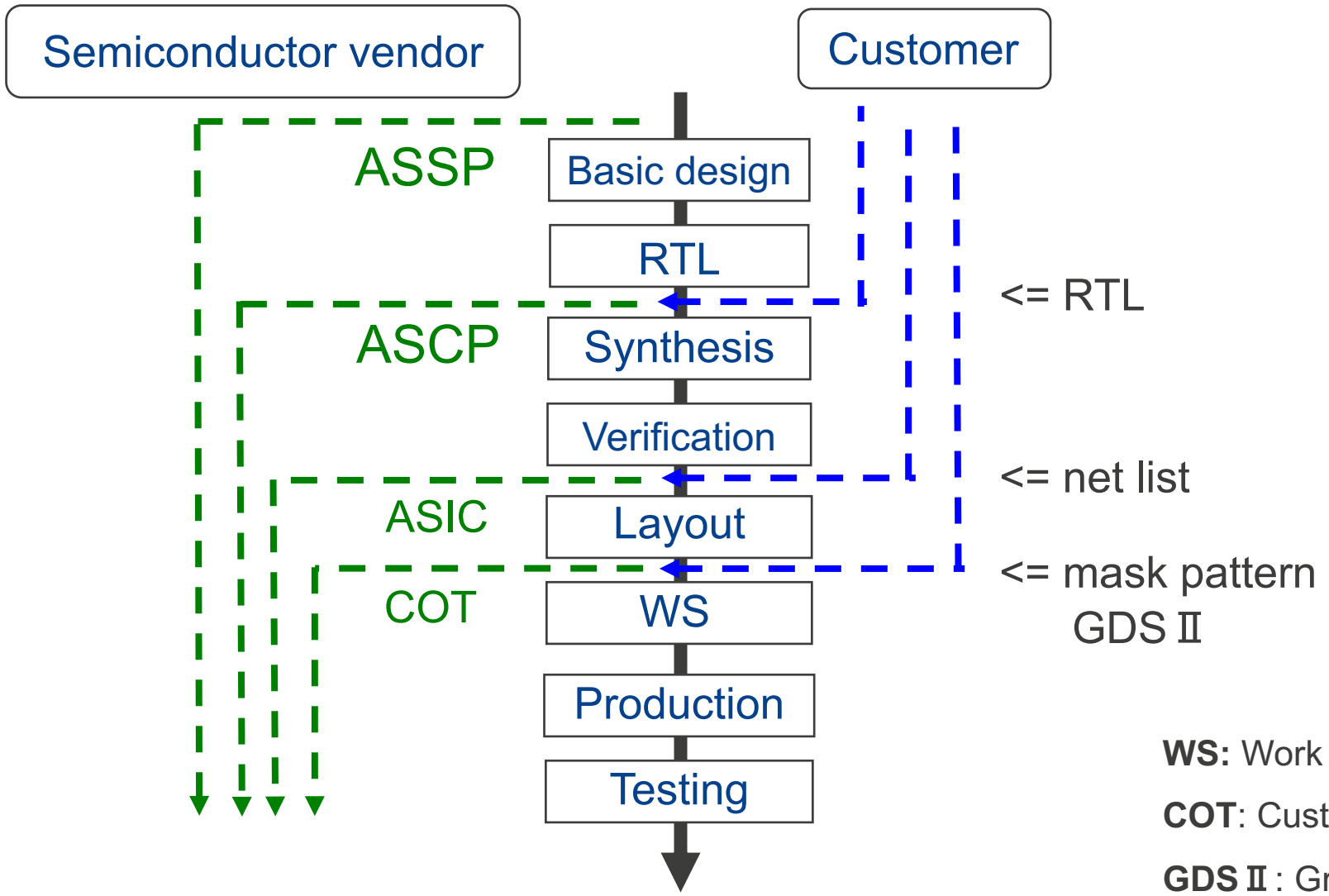
UMC: United Microelectronics Corp.

TSMC: Taiwan Semiconductor Manufacturing Company

Old vs. Current Design

	Old LSI design	Current LSI design
General	Manufacture dependent (Closed system)	Cell base design (Open system)
Design functions	Manufacture specific tools.	C language, Verilog, VHDL, etc.
Create gate level logic	Manual mapping into transistor logic gates and hand optimization needed because of poor synthesis tools. Created gate diagrams have poor portability among LSI production lines.	Automatically mapped into standard cells and optimized by powerful synthesis tools. Created gate diagrams have portability among LSI production lines.
Create MASK pattern	Manual work needed because of poor layout tools.	Automatic optimization by powerful layout tools.
LSI fabrication	Possible only on the specific fabrication lines. No compatible lines among vendors.	Possible on the compatible fabrication lines among vendors.

Classification of IC from the view point of design



WS: Work Sign Off

COT: Customer Own Tool

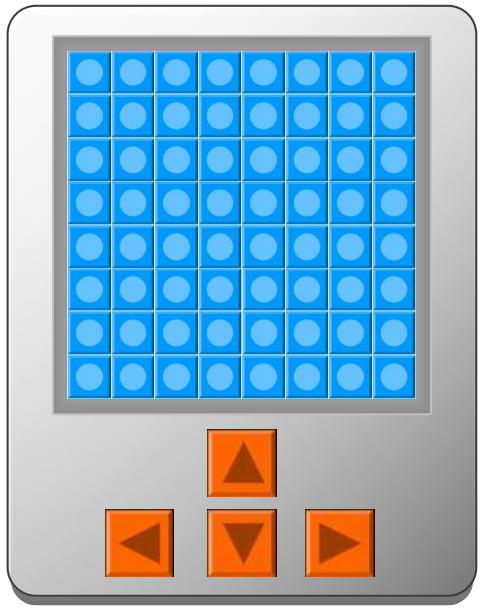
GDS Ⅱ : Graphic Data System II

1.4 An Example

Implement a logic for Snake game in Verilog self-learning material on an actual chip.

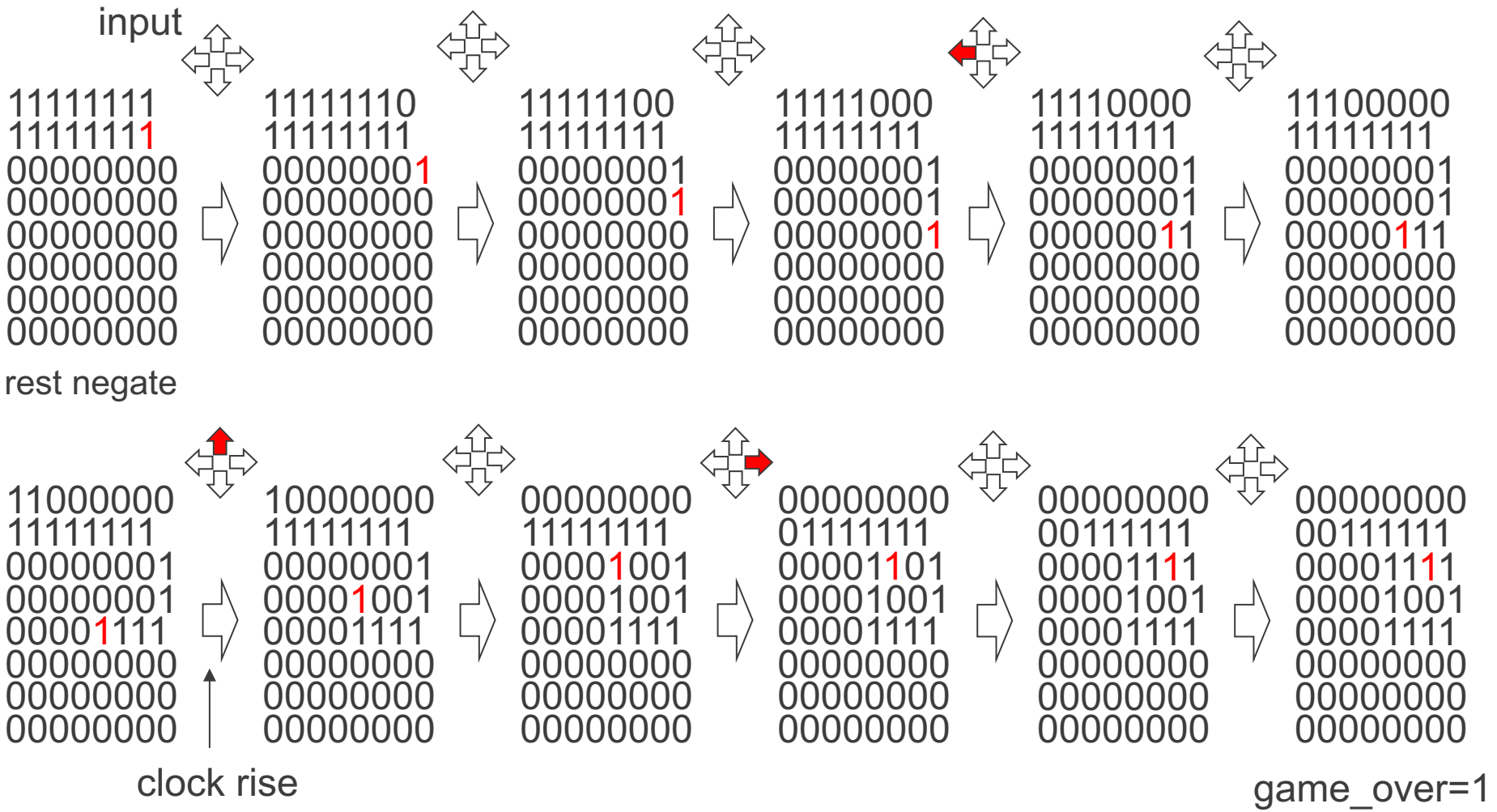
Snake game
Handy game terminal
Battery drive
Low price
LCD display

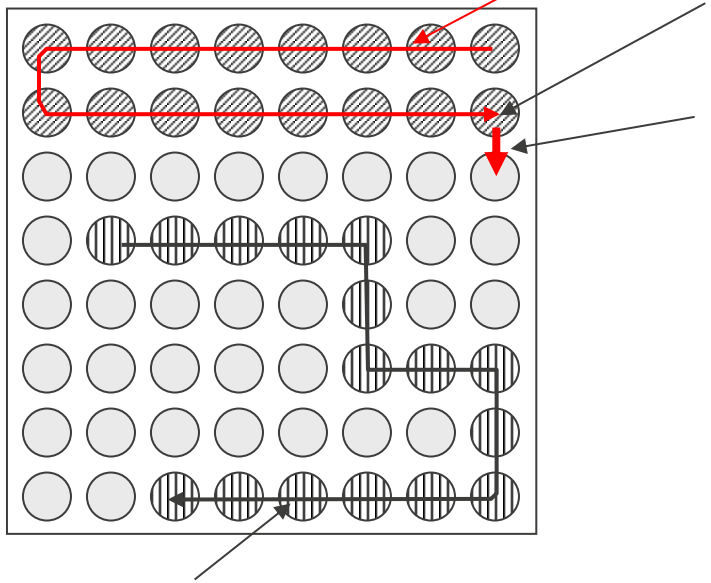
Let's design and create a chip to be used in this snake game terminal.



Game specification

Following figures show 16-bit length snake moving around in 8x8 lamp field. 1 means lamp on, and 0 means off. The on lamps represent shake body. IT moves one lamp at clock rise time as shown below.

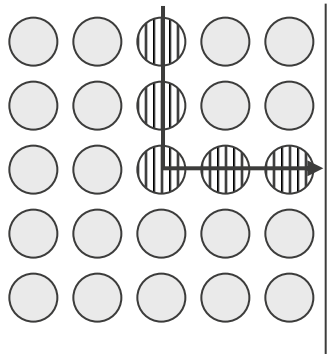




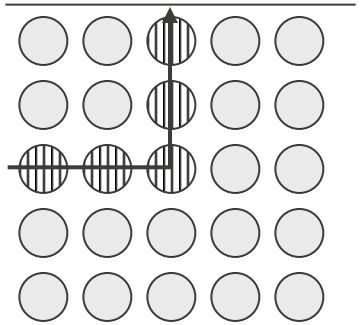
When reset signal is negated, the snake begins to move downward. At each rising edge of clock, it moves one lamp at a time. The snake length must always be 16 lamps.

A player has to control the moving direction of the head by 4-bit control input ctl_dir.

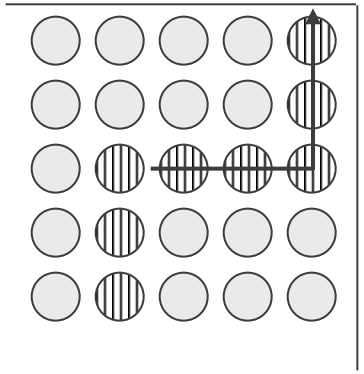
The game ends if the head cannot move any more. The head can neither go out of the lamp field nor cross its body.



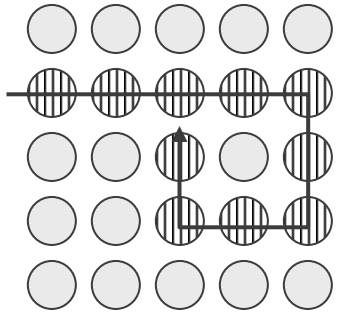
Game ends unless the head is controlled to go up or down.



Game ends unless the head is controlled to go right or left.



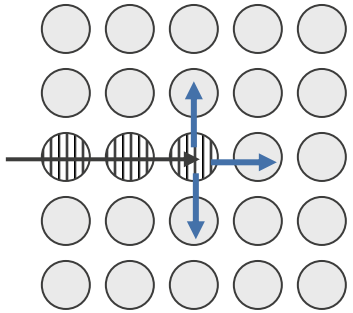
Game end unless the head is controlled to go left.



Game end unless the head is controlled to go left or right.

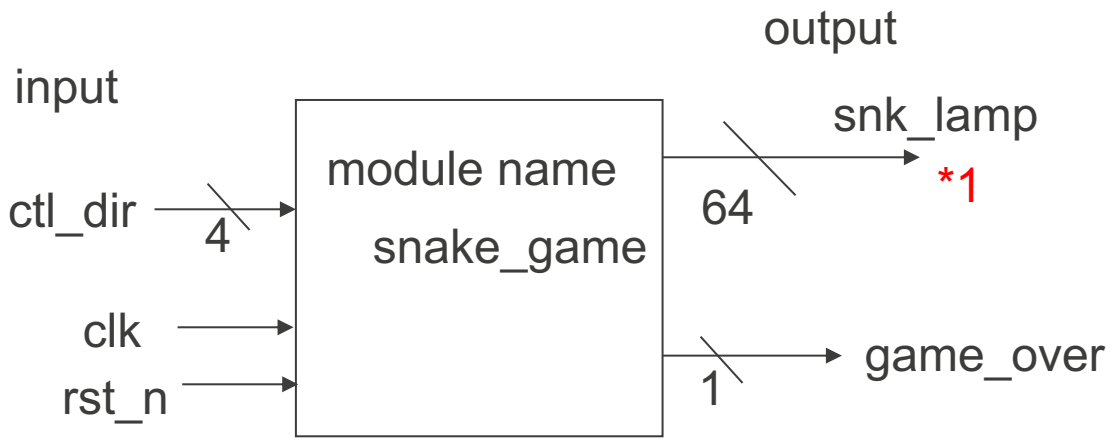
(go_right will end up “game over” in the next clock cycle in this case)

The head can move at most three directions as shown below. It can not go back. If go backward input given, the game is over.



: snake's current moving direction

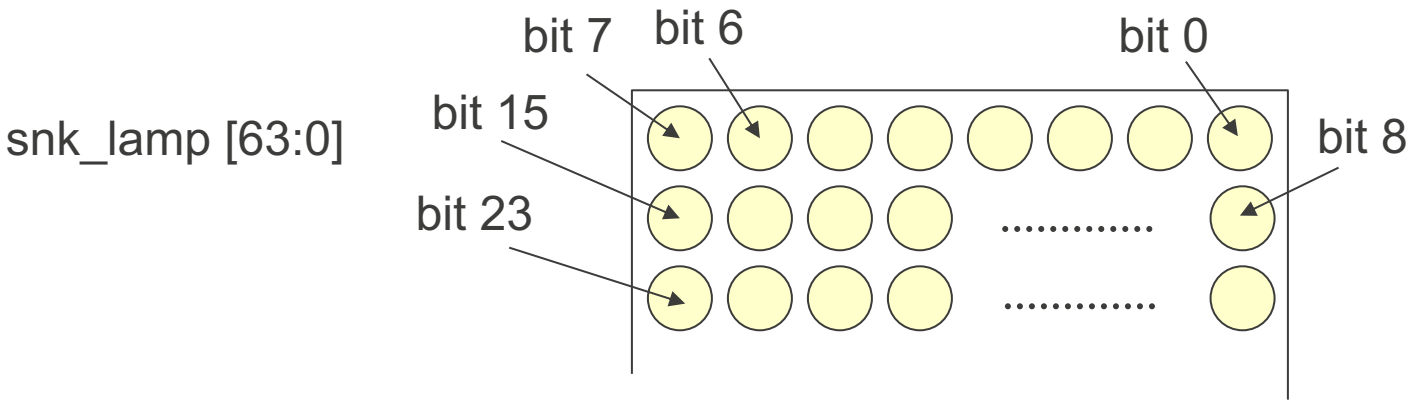
: movable direction



When game_over is asserted, the snake must stop moving and keep its position until reset asserted.

game_over does not becomes 0 once it is asserted unless reset asserted.

*1: In Verilog1995, array cannot be used as a port. Therefore, output is defined as 8x8=64-bit length vector data. Each bit represents one lamp in the lamp field. Bit 0 corresponds to upper right corner lamp, bit 7 to upper left corner, bit 56 to bottom right corner, and bit 63 to bottom left corner lamp as shown below.



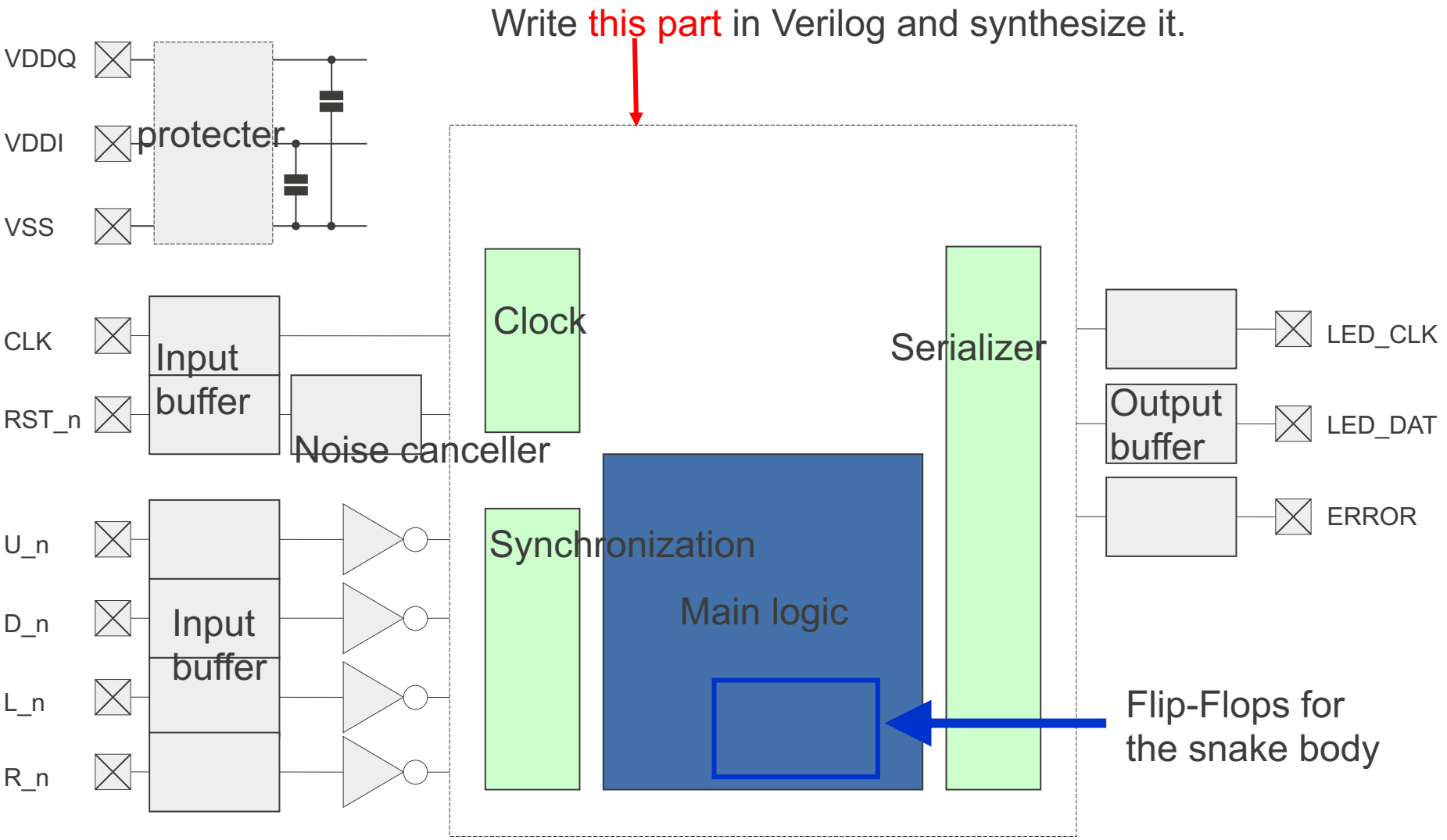
In the actual implementation, we use a serial interface to reduce the number of pins, that is, we use a serializer for 64 bits parallel interface of snake lamps.
The additional specifications needed to design the chip are as follows.

Snake moving speed: 2 lamps per second (2Hz)
Clock : 2Hz*128*32 = 8KHz
Use 32KHz clock and gate it to slow down the speed.
Maxmum lamps: 128 lamps
(Implementation 64 lamps)
Package: 14 pin so ,
internal connection: 5 PADs x 2 lows

pin:

(1)VDDQ	(6)U_n	(10)LED_CLK
(2)VDDI	(7)D_n	(11)LED_DATA
(3)VSS	(8)L_n	(12)ERROR
(4)CLK	(9)R_n	
(5)RST_n		

Block structure of the chip



A part of the RTL code for the main logic

```
module snake_game(clk, rst_n, clk_en, ctl_dir, snk_lamp, game_over);
//
// -- parameter definition --
parameter A_WIDTH = 3;
parameter X_WIDTH = 8;      // 2 ** A_WIDTH : must be powerd by 2
parameter Y_WIDTH = X_WIDTH; // must be X_WIDTH = Y_WIDTH
parameter MV_U = 2'b00;
parameter MV_D = 2'b01;
parameter MV_R = 2'b10;
parameter MV_L = 2'b11;
parameter INIT_SNK_LAMP = 64'h0000000000FFFF;
parameter INIT_BODY    = 96'o10_11_12_13_14_15_16_17_07_06_05_04_03_02_01_00;
parameter INTL = 2'b00;
parameter GO   = 2'b01;
parameter ERR  = 2'b11;
//
// -- I/O --
input  clk, rst_n, clk_en;
input  [3:0] ctl_dir ; //U D R L
output [(Y_WIDTH*X_WIDTH-1):0] snk_lamp ;
output game_over;
//
// -- wire --
wire clk, rst_n, clk_en;
wire [3:0] ctl_dir;
wire hitall;
wire [(Y_WIDTH*X_WIDTH-1):0] temp_snk_lamp ;
wire [(Y_WIDTH*X_WIDTH-1):0] next_snk_lamp ;
wire [(Y_WIDTH*X_WIDTH-1):0] head_bit ;
wire hitbody ;
wire [(A_WIDTH*2*16-1):0] next_body_arry ;
wire [A_WIDTH-1:0] tail_x, tail_y ;
wire err_stop ;
wire game_over ;
//
// -- reg --
reg [(Y_WIDTH*X_WIDTH-1):0] snk_lamp ;    //FF
reg [1:0] move ;                          //FF
reg [1:0] next_move ;
reg [A_WIDTH  :0] head_x, head_y ; // w/i carry bit
reg [(A_WIDTH*2*16-1):0] body_arry ;
reg [1:0] state ;                         //FF
reg [1:0] next_state ;
//
// -- keep moving direction --
always@(posedge clk or negedge rst_n) begin
  if(!rst_n) begin
    move <= MV_D;
  end
  else begin
    if(clk_en) begin
      move <= next_move;
    end
  end
end
always@(ctl_dir or move) begin
  case(ctl_dir)
    4'b1000: begin next_move = MV_U ; end
    4'b0100: begin next_move = MV_D ; end
    4'b0010: begin next_move = MV_R ; end
    4'b0001: begin next_move = MV_L ; end
    4'b0000: begin next_move = move ; end
    default: begin next_move = move ; end
  endcase
end
//
// -- lamps & hit --
always@(posedge clk or negedge rst_n) begin
  if(!rst_n) begin
    snk_lamp <= INIT_SNK_LAMP;
  end
  else begin
    if(clk_en) begin
      snk_lamp <= next_snk_lamp;
    end
  end
end
assign temp_snk_lamp = snk_lamp & ~bin2bit(tail_x, tail_y) ; //Erase Tail bit
assign head_bit      = bin2bit(head_x, head_y) ;          //Decode Head position to bitmap
assign hitbody       = ((temp_snk_lamp & head_bit) == {(Y_WIDTH*X_WIDTH){1'b0}}) ? 1'b0 : 1'b1 ;//Check head hit in body
assign next_snk_lamp = err_stop ? snk_lamp : temp_snk_lamp | head_bit ;                    //Set Head bit
//
// -- decoder --
// Decode binry data to bitmap like a 10(binary) -> 0100(bitmap)
function [(X_WIDTH*Y_WIDTH-1):0] bin2bit;
  input [A_WIDTH-1:0] xx ; //Lower
  input [A_WIDTH-1:0] yy ; //Upper
  reg [A_WIDTH*2-1:0] pos ;
  begin
    pos = (yy << A_WIDTH) + xx ;
    bin2bit = {(Y_WIDTH*X_WIDTH) {1'b0}} ;
    bin2bit[pos] = 1'b1 ;
  end
endfunction
```

```
// -- body stack --
// Keeping head-body-tail position
// (MSB)Head_Y--Head_X--Body--Tail_Y--Tail_X(LSB)
always@(posedge clk or negedge rst_n) begin
  if(!rst_n) begin
    body_arry <= INIT_BODY ;
  end
  else begin
    if(clk_en) begin
      body_arry <= next_body_arry ;
    end
  end
end
assign tail_x = body_arry[A_WIDTH-1:0] ;       //Get Tail position
assign tail_y = body_arry[A_WIDTH*2-1:A_WIDTH] ; //Get Tail position
always@(next_move or body_arry) begin  //Y-motion
  case(next_move)
    MV_L: begin head_y = body_arry[(A_WIDTH*2*16-1):(A_WIDTH*2*16-A_WIDTH)] + {{A_WIDTH {1'b0}},1'b0} ; end
    MV_R: begin head_y = body_arry[(A_WIDTH*2*16-1):(A_WIDTH*2*16-A_WIDTH)] + {{A_WIDTH {1'b0}},1'b0} ; end
    MV_D: begin head_y = body_arry[(A_WIDTH*2*16-1):(A_WIDTH*2*16-A_WIDTH)] + {{A_WIDTH {1'b0}},1'b1} ; end //w/i caryy
    MV_U: begin head_y = body_arry[(A_WIDTH*2*16-1):(A_WIDTH*2*16-A_WIDTH)] - {{A_WIDTH {1'b0}},1'b1} ; end //w/i caryy
  endcase
end
always@(next_move or body_arry) begin  //X-motion
  case(next_move)
    MV_L: begin head_x = body_arry[(A_WIDTH*2*16-A_WIDTH-1):(A_WIDTH*2*15)] - {{A_WIDTH {1'b0}},1'b1} ; end //w/i carry
    MV_R: begin head_x = body_arry[(A_WIDTH*2*16-A_WIDTH-1):(A_WIDTH*2*15)] + {{A_WIDTH {1'b0}},1'b1} ; end //w/i carry
    MV_D: begin head_x = body_arry[(A_WIDTH*2*16-A_WIDTH-1):(A_WIDTH*2*15)] + {{A_WIDTH {1'b0}},1'b0} ; end
    MV_U: begin head_x = body_arry[(A_WIDTH*2*16-A_WIDTH-1):(A_WIDTH*2*15)] + {{A_WIDTH {1'b0}},1'b0} ; end
  endcase
end
assign next_body_arry = err_stop ? body_arry : {head_y[A_WIDTH-1:0], head_x[A_WIDTH-1:0], body_arry[(A_WIDTH*2*16-1):A_WIDTH*2]} ; //Add new head position to the
stack
//
// -- state --
always@(posedge clk or negedge rst_n) begin
  if(!rst_n) begin
    state <= INTL;
  end
  else begin
    if(clk_en) begin
      state <= next_state;
    end
  end
end
//
always@(state or hitall or ctl_dir) begin
  case(state)
    INTL : begin next_state = (((ctl_dir == 4'b0100) | (ctl_dir == 4'b0000)) & (hitall == 1'b0)) ? GO : ERR ; end
    GO   : begin next_state = (hitall == 1'b1) ? ERR : state ; end
    ERR  : begin next_state = ERR ; end
    default : begin next_state = 2'bxx ; end
  endcase
end
// The range of position is head_?[A_WIDTH-1:0], so head_?[A_WIDTH] should be 0 in nomal opsition
assign hitall = hitbody | head_x[A_WIDTH] | head_y[A_WIDTH] ;
assign err_stop  = (next_state == ERR) ? 1'b1 : 1'b0 ;
assign game_over = (state == ERR)      ? 1'b1 : 1'b0 ;
//
endmodule
~
~
```

Flip-Flops for the snake body

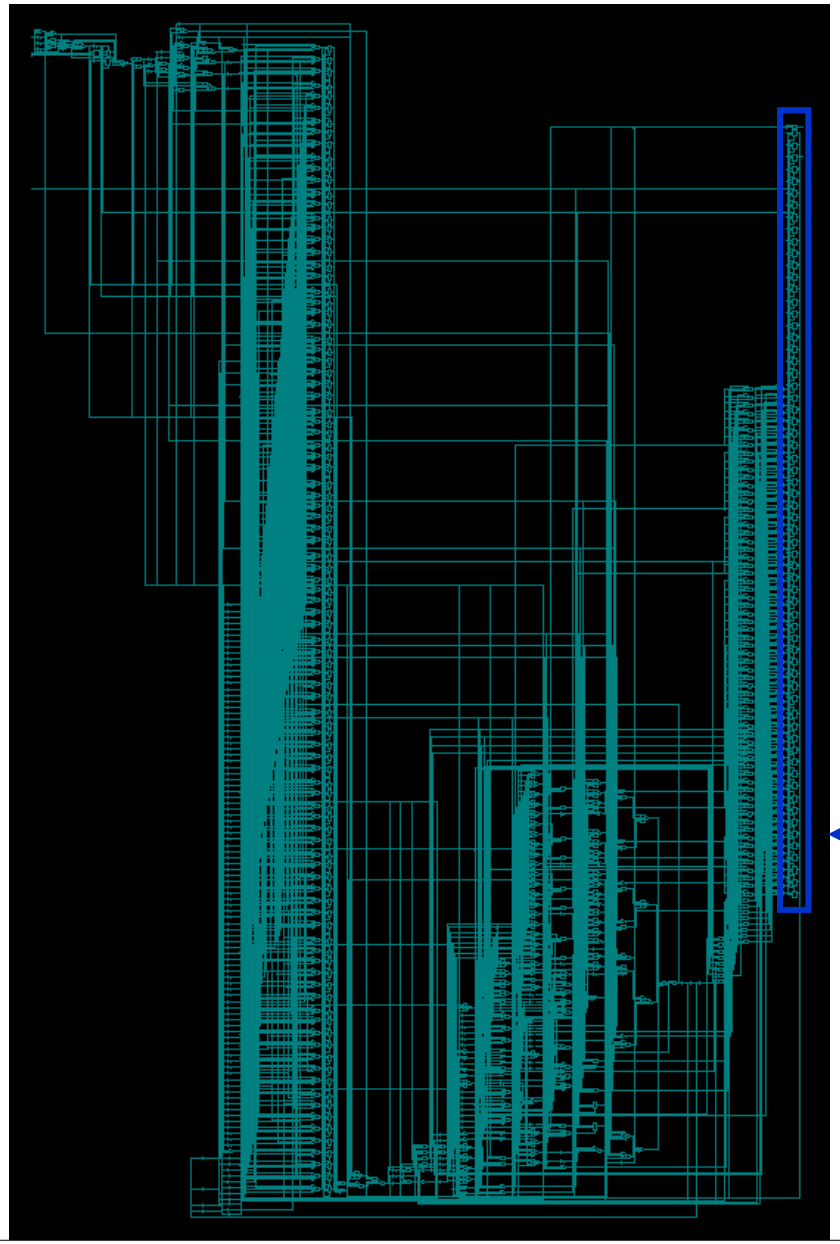
154 lines including comments.

```
// -- lamps & hit --
always@(posedge clk or negedge rst_n) begin
  if(!rst_n) begin
    snk_lamp <= INIT_SNK_LAMP;
  end
  else begin
    if(clk_en) begin
      snk_lamp <= next_snk_lamp;
    end
  end
end
assign temp_snk_lamp = snk_lamp & ~bin2bit(tail_x, tail_y) ; //Erase Tail bit
assign head_bit      = bin2bit(head_x, head_y) ;           //Decode Head position to bitmap
assign hitbody       = ((temp_snk_lamp & head_bit) == {(Y_WIDTH*X_WIDTH){1'b0}}) ? 1'b0 :
1'b1 ;//Check head hit in body
assign next_snk_lamp = err_stop ? snk_lamp : temp_snk_lamp | head_bit ;
```

Flip-Flops for
the snake body

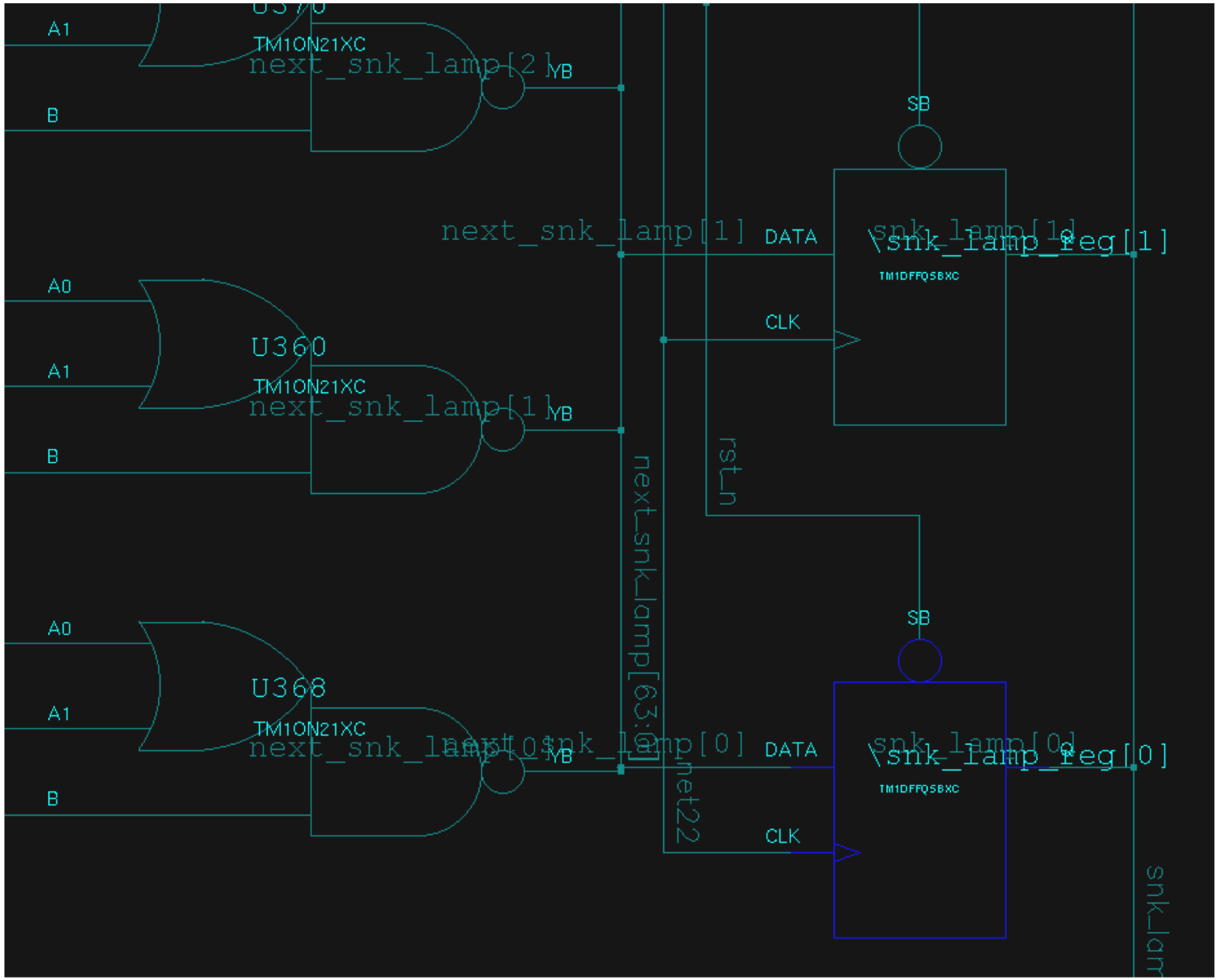
The main logic written in Verilog can be synthesized into the following gate netlist.

After the synthesis, mail logic part looks like the following gate giagram.



Flip-Flops for
the snake body

741 cells and Gated-clock cells.



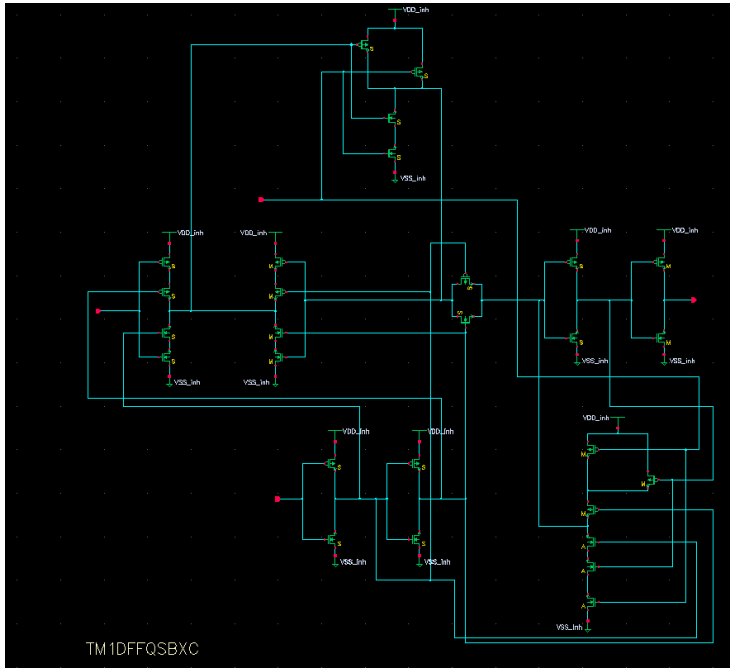
A part of the gate diagram including Flip-Flops of the snake body

```
always@(posedge CLK or negedge SB) begin
  if(!SB) begin
    Q <= 1'b1;
  end
  else begin
    Q <= DATA;
  end
end
```

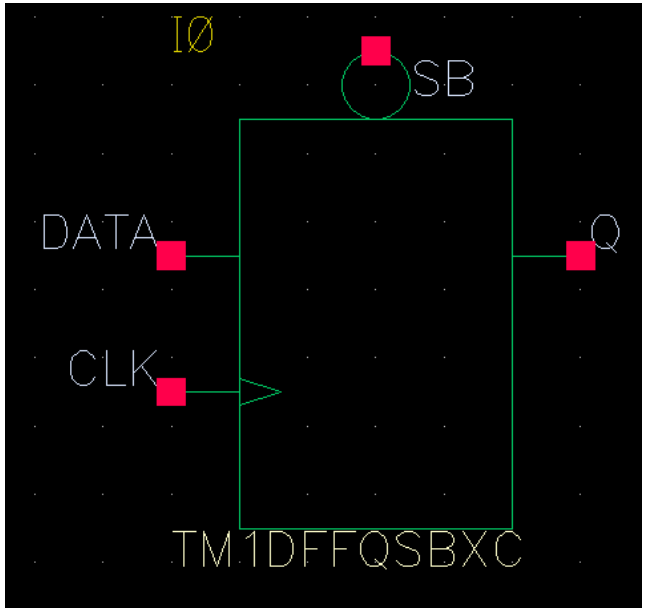
Verilog

The Flip-Flop of the snake body

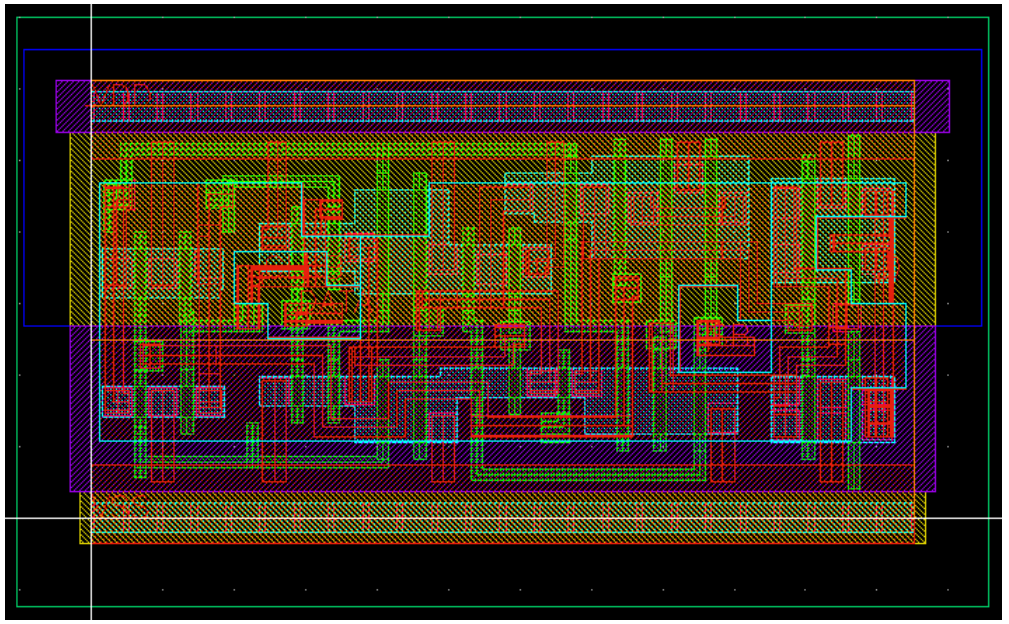
MOS circuit



Gate diagram



Layout



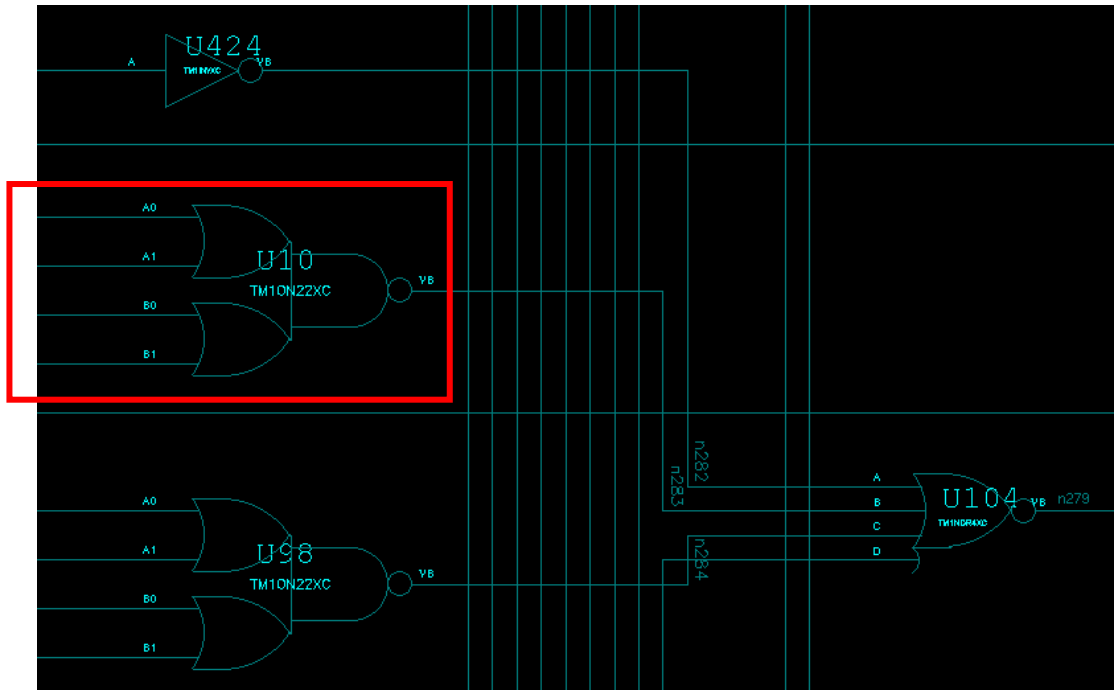
A part of the gate netlist including Flip-Flops of the snake body

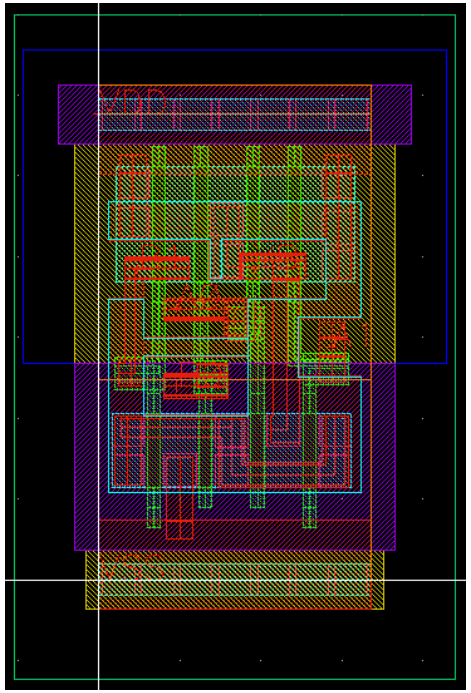
```
SNPS_CLOCK_GATE_HIGH_snake_game clk_gate_move_reg ( .CLK(clk), .EN(clk_en),
      .ENCLK(net22) );
 TM1DFFQRBXC ¥state_reg[0]  ( .DATA(next_state[0]), .CLK(net22), .RB(rst_n),
      .Q(state[0]) );
 TM1DFFQRBXC ¥move_reg[1]  ( .DATA(next_move[1]), .CLK(net22), .RB(rst_n),
      .Q(move[1]) );
 TM1DFFQRBXC ¥state_reg[1]  ( .DATA(next_state[1]), .CLK(net22), .RB(rst_n),
      .Q(state[1]) );
( snip )
 TM1DFFQSBXC ¥snk_lamp_reg[11]  ( .DATA(next_snk_lamp[11]), .CLK(net22),
      .SB(rst_n), .Q(snk_lamp[11]) );
 TM1DFFQSBXC ¥snk_lamp_reg[8]  ( .DATA(next_snk_lamp[8]), .CLK(net22),
      .SB(rst_n), .Q(snk_lamp[8]) );
 TM1DFFQSBXC ¥snk_lamp_reg[0]  ( .DATA(next_snk_lamp[0]), .CLK(net22),
     .SB(rst_n), .Q(snk_lamp[0]) );
 TM1INVXA U3 ( .A(n9), .YB(n412) );
( snip )
```

```
TM1ON22XC U10 ( .A0(n33), .A1(n289), .B0(n3), .B1(n290), .YB(n283) );
TM1INVXH U11 ( .A(n6), .YB(n33) );
TM1AND2XN U12 ( .A(state[1]), .B(state[0]), .Y(game_over) );
TM1INVXH U13 ( .A(n5), .YB(n15) );
```

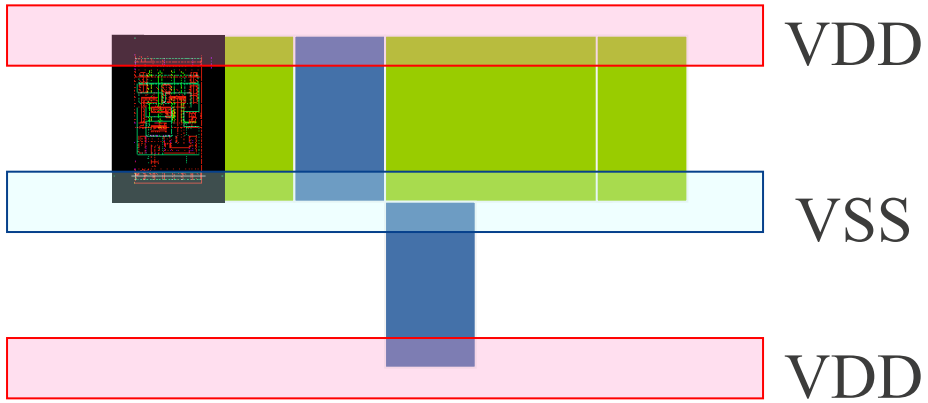
Netlist is a code describing gate connections.

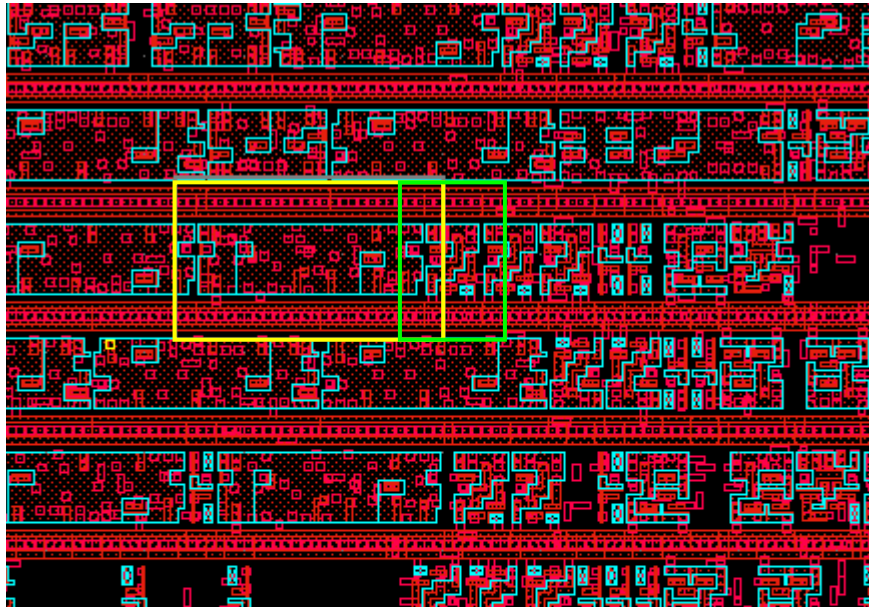
Netlist has less readability, therefore we use schematics as shown below.



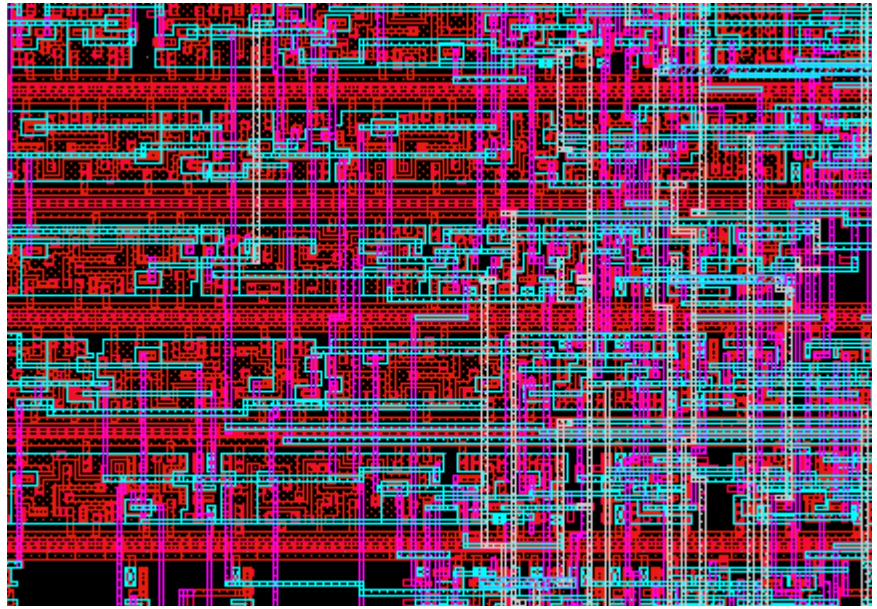


Circuits can be built by connecting cells.
Many cells have the same hight but different width depending on their functionality.





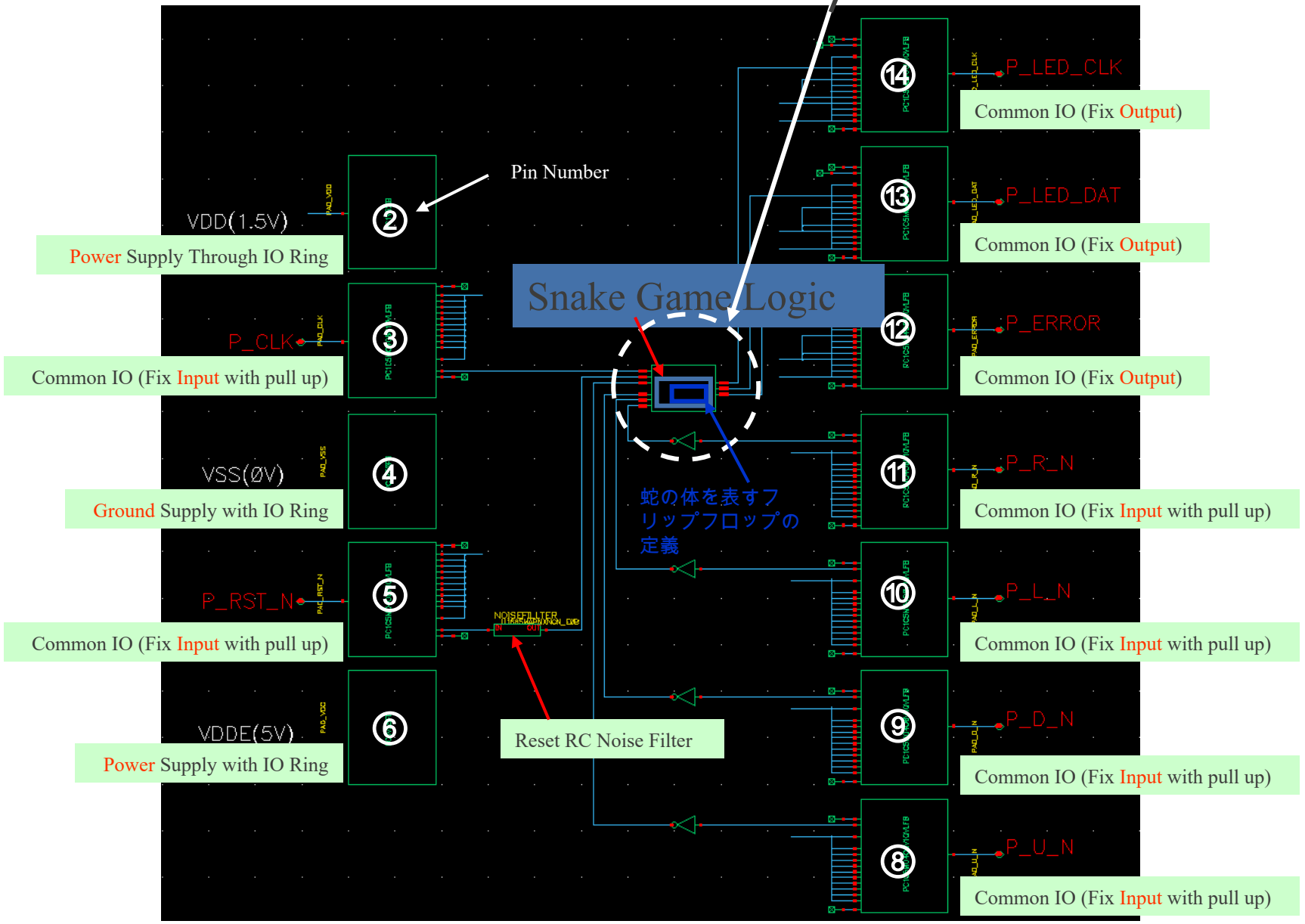
Before wiring



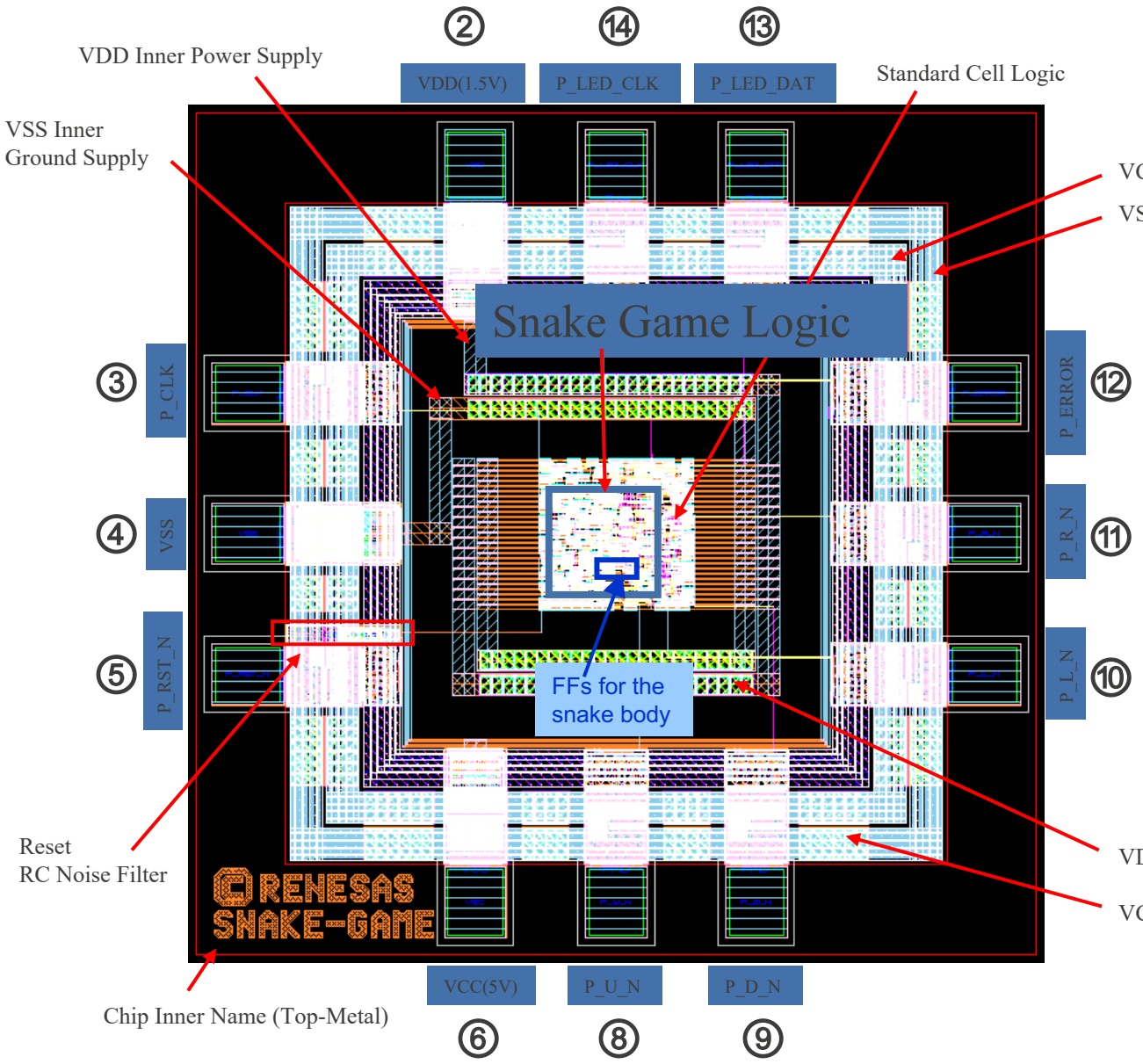
After wiring

The schematic for all the chip.

The logic part shown in the previous pages.



The total chip layout.

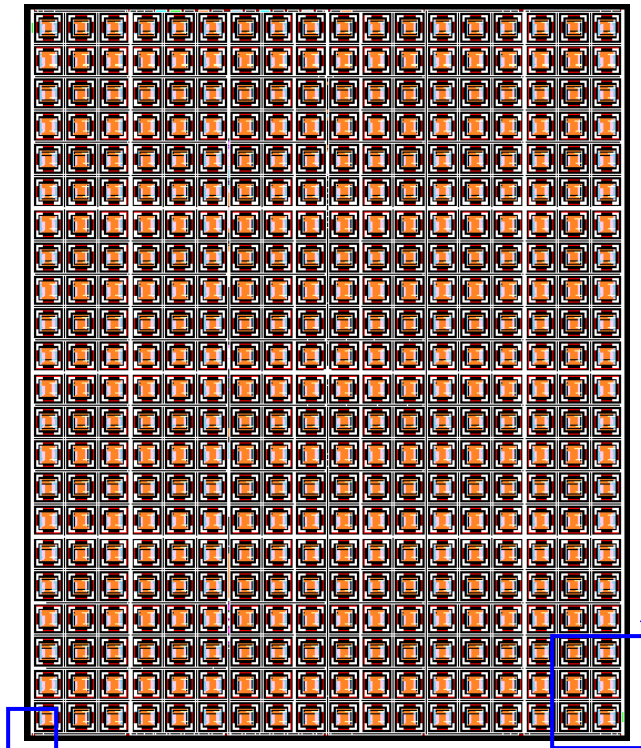


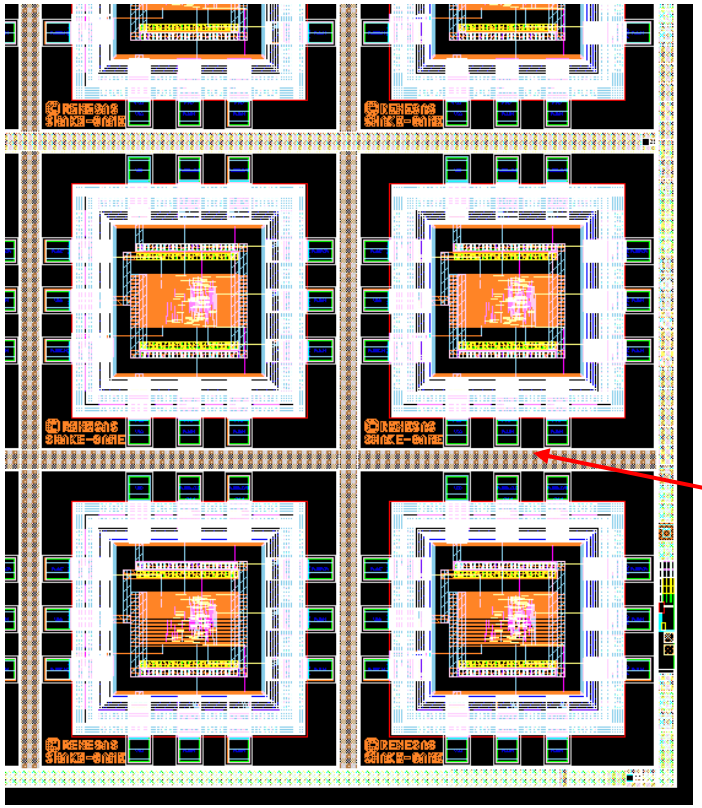
- Process Information
  (0.15um process)
  5-Metal Layers
  12-Pitch Standard Cell
  (0.48um x 5.76um)

- Chip Information
  Chip Size (Scribe Line 60um)
  X = 940um + 60um = 1.0mm
  Y = 940um + 60um = 1.0mm

The shot image

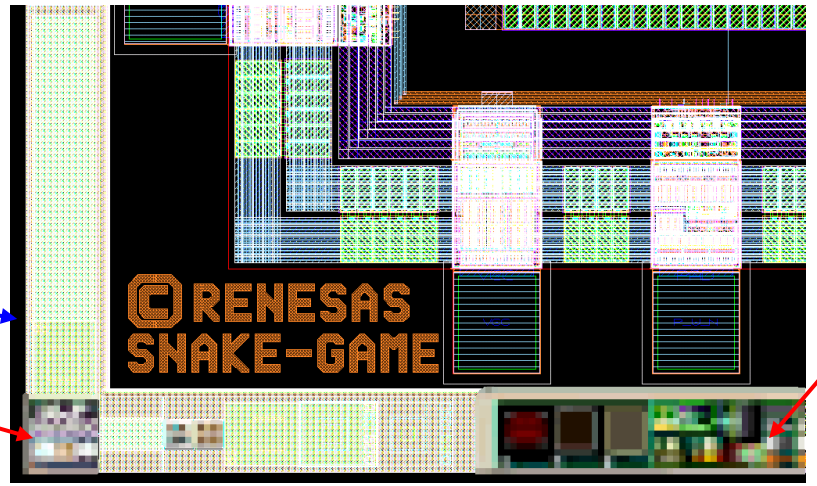
1-Shot : 396 Chips





Cut here to isolate each die.

Scribe Line (60um)



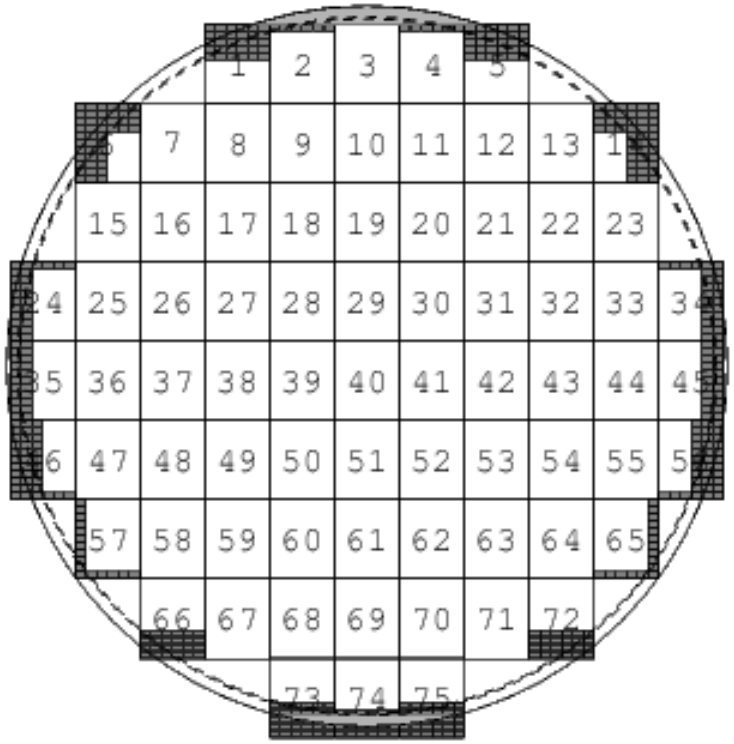
Process monitor pattern

Process Side Monitor (Scribe TEG)

Alignment mark

Wafer Map

The mask is photo printed as shown below on a silicon wafer.
Each shot has 396 chips.

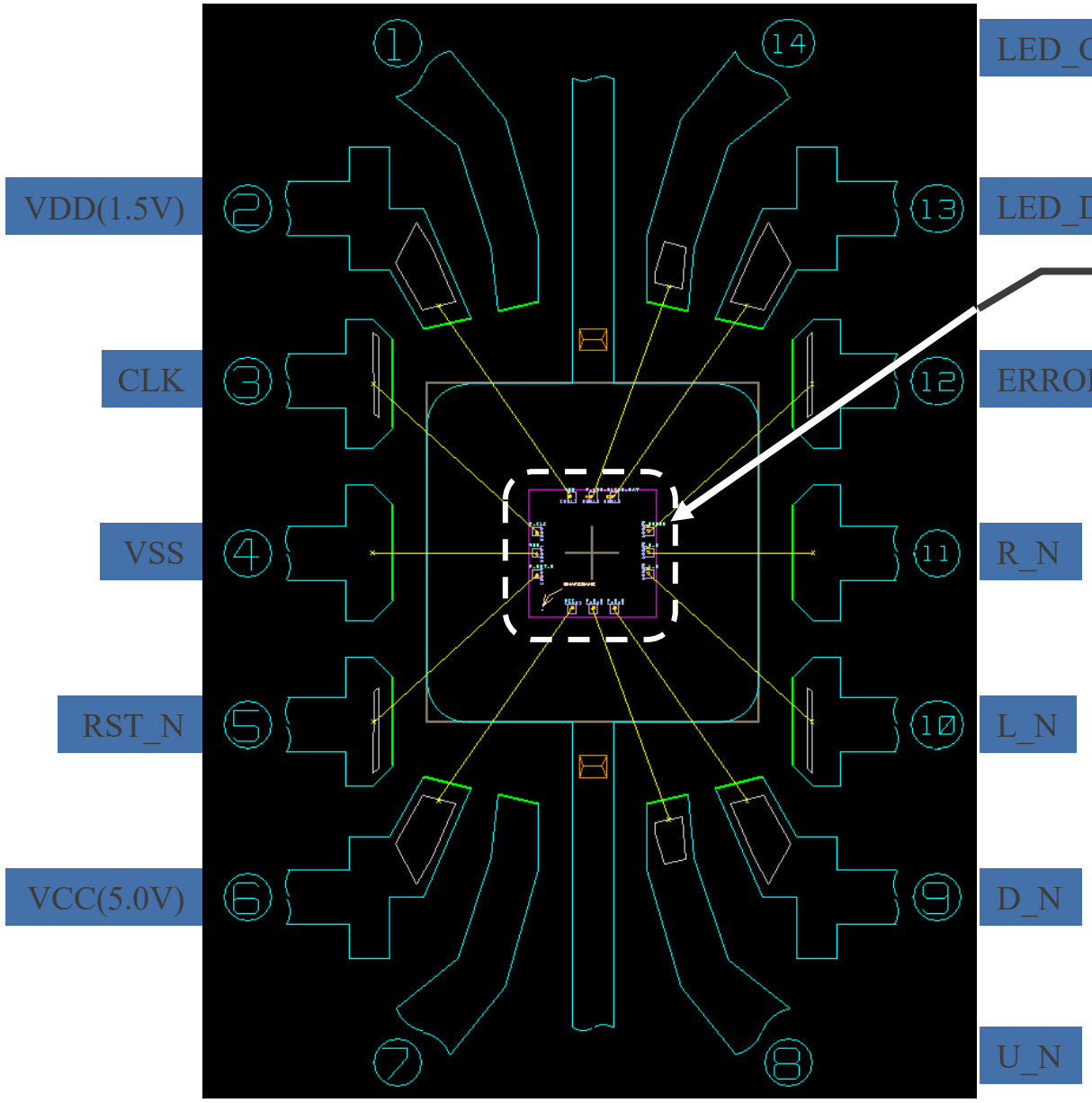


- Wafer Information

Wafer Size : 8-Inch
Number of Chips : 26736

Wire Diagram

Structure inside the package.



This part correspond to the total chip layout.

◆Wire Diagram Information

PAD Opening : 68um x 82um
Total Wire Length : 2.4cm
Max Wire Length : 1724.1um
Min Wire Length : 1215.2um

Renesas.com