Ver. 2023-04-25b

Course number: CSC.T341

コンピュータ論理設計 演習(3) Computer Logic Design Exercise(3)

情報工学系 荒堀喜貴 Yoshitaka ARAHORI, Department of Computer Science arahori_at_c.titech.ac.jp

Computer Logic Design support page https://www.arch.cs.titech.ac.jp/lecture/CLD/

CSC.T341 Computer Logic Design, Department of Computer Science, TOKYO TECH

2023年度版

コンピュータ論理設計 演習(Exercise)の注意点

- 演習はACRiルームを利用します。
- 3~4人のグループを作成します.そのグループ内で情報を共有しながら演習を 進めてください.
- 問題はグループ内で相談して解決する、あるいは、担当のTA(Teaching Assistant)や教員に質問してください.
- 演習には出席点があります.休まずにきちんと出席しましょう.
- 演習スライドにチェックポイントの図がある場所は、作業を確認してもらう場所です。すべてのチェックポイントをクリアしましょう.

Check Point 1

 演習時間でなくてもACRiルームを利用できます.現在は、1日に4枠(3時間 x 4 枠 = 12時間)を利用できます。独自のハードウェア設計などに挑戦しましょう.



Exercise(3)

- Project_3
 - CMT (Clock Management Tile) を使って、FPGAの内部で 20MHz と 33MHz のクロック信号を生成する方法を学ぶ。
 - この方法を用いることで、設計したハードウェアの高性能化のために動作周 波数を上げることができる。
- Project_4
 - 加算器, 論理演算, シフターなどの主要な算術論理演算のハードウェア量を 計測する. また, ALU全体のハードウェア量を計測する.
 - これにより、FPGA(xc7a35t)で利用できるハードウェア量を把握する.





新しいプロジェクトの作成

- Project_1と同様に、新しいプロジェクト project_3 を作成する.
 - 制約ファイルとして main11.xdc を登録する.
 - Verilog HDLファイルとして code057.v を登録する. vio_0 のコメン トアウトを削除する.

code057.v

```
module m_main (w_clk, w_led);
input wire w_clk;
output wire [3:0] w_led;
reg    r_out = 0;
reg [31:0] r_cnt = 0;
always@(posedge w_clk) begin
    r_cnt <= (r_cnt==9999999) ? 0 : r_cnt +1;
    r_out <= (r_cnt==0) ? ~r_out : r_out;
end
assign w_led = {r_out, r_out, r_out, r_out};
// vio_0 vio_00(w_clk, w_led[3], w_led[2], w_led[1], w_led[0]);
endmodule
```



新しい Vivado プロジェクトの作成とファイルの登録

- ・ 前回の演習を参考に、Vivado で新しいプロジェクト project_3 を作成する.
- Ubuntu で起動したターミナルで, 次のコマンドを実行してファイルをコピーする.
 - /home/tu_kise は automount のディレクトリなので、アクセスしないとファイルが見えない、tabキーによる補完がうまく動作しないことがあるので注意する。
 - 最後の ls コマンドで, code057.v, code105.v, code106.v, main11.xdc が表示される

```
$ ls /home/tu_kise
$ cd ~/cld/project_3
$ cp /home/tu_kise/cld/2023/code057.v .
$ cp /home/tu_kise/cld/2023/code105.v .
$ cp /home/tu_kise/cld/2023/code106.v .
$ cp /home/tu_kise/cld/2023/main11.xdc .
$ ls
```

- Vivado で, project_3 の制約ファイルとして main11.xdc を登録する.
 - main11.xdc ファイルの内容を変更する必要はない.

ことを確認

- Vivado で, project_3のVerilog HDLファイルとして code057.v を登録する.
- VIOの IP を生成して、次のスライドの通り、1秒毎に点滅することを確認する.

CSC.T341 Computer Logic Design, Department of Computer Science, TOKYO TECH

VIOを用いてFPGAの動作をリモートで確認

- Click + button in hw_vio_1 window.
- Select w_led_OBUF[0:0], w_led_OBUF_1[3:3], w_led_OBUF_2[2:2], and w_led_OBUF_3[1:1] by clicking them pushing Shift button. Then click OK.
- You will see the values of four LEDs change every second!
- ここまでは Project_1 と同様の作業.

Add Probes	×	со	de057.v × hw_vios ×	
¥ \$	٠		huu uin 1	
earch: Q-		ы	IW_VI0_1	
Probes for hw_vio_1 (4)	~	tion	Q ¥ ♦ + ■	
√ 1 hw_vio_1		opi	Name Value Activity Direction	VI0
w_led_OBUF_1[3:3]		bard	🐌 w_led_0BUF_2[2:2] [B] 1 🔹 Input	hw_v
1 w_led_0BUF_2[2:2]		hbo	🐌 w_led_0BUF[0:0] [B] 1 💲 Input	hw_v
₩_led_OBUF_3[1:1]		Das	🐌 w_led_0BUF_1[3:3] [B] 1 🔹 Input	hw_v
			1_ w_led_OBUF_3[1:1] [B] 1	hw_v
	:			
	1			
\frown				
ОК	Cancel			
				



CSC.T341 Computer Logic Design, Department of Computer Science, TOKYO TECH

200

- 🗆 🗙

Clocking Wizard を起動する

- Click IP Catalog
- Double click Clocking Wizard in IP Catalog window

Flow Navigator ₹ ≑ ? .	
✓ PROJECT MANAGER	hll.xd
🔅 Settings	Cor
Add Sources	
Language Templates	4
IP Catalog	<u>S</u> ear
	Name
V IP INTEGRATOR	
Create Block Design	~
Open Block Design	
Generate Block Design	
✓ SIMULATION	
Run Simulation	>
	>
✓ RTL ANALYSIS	<
> Open Elaborated Design	Det

nll.xdc x hw_ila_1 x dashboard_1	× IP Catalog 🛛 🗸 🗸
Cores Interfaces	
Q 🗶 🗢 🕫 📭 🖉 🕮 🔳	
Search: Q-	
Name	^ 1 AXI4
> 🗅 Utility	
🗸 🚍 FPGA Features and Design	
🗸 🚍 Clocking	
👎 Clocking Wizard	AXI4
> 🚍 IO Interfaces	
> 🚍 Soft Error Mitigation	
> 🚍 XADC	
> 🚍 Kernels	
💫 🗁 Math Functions	
<	
Details	



20MHzのクロックを出力する IP を生成する

- In Output Clocks, change the frequency from 100.000 to 20 for clk_out1 to generate 20MHz clock signal. Click OK.
- In Generate Output Products window, click Generate.

	Customize I	P	×		Re-cus	tomize IP		
locking Wizard (6.0)			4	Clocking Wizard (6.0)				
Documentation 🛛 📄 IP Loca	ation C Switch to Defaults			🚯 Documentation 🛛 🖨 IP Locatio	n C Switch to Defau	ılts		
IP Symbol R ⊲ ▶ ≡	Component Name clk_wiz_	_0	0	IP Symbol Res 4 ▶ ≡	Component Name	clk_wiz_0		
Show disabled ports	Board Clocking Onti	ons Output Clocks Port	Be (↓ ► =	Show disabled ports	Board Clockin	g Options	Output Clocks	ort Renami 4 🕨
	Accession ID interface with				The phase is calc	ulated relative	to the active input cl	ock.
	IP Interface	Board Interface			Output Clock	Port Name	Output Freq (M	Hz)
"da a sul lin	CLK_IN1	Custom	•	+ s_ext_ite	Clk out1	clk out1 🤇	20	20.00000
+ CK_1ML0 + CK_1ML0	CLK_IN2	Custom		+ CLK_INI_D + CLK_INI_D + CLKFB IN D	□ _ □ clk out2	clk out2	100.000	N/A
+ CLARE_NUE = Sex[ext] = Sex	EXT_RESET_IN	Custom	•	s_axt_acte CLKFB_OUT_D +	clk out3	_ clk_out3	100.000	N/A
exert altered	Clear Board Paramete	ers		resect cite_cor(3.0)	clk out4	_ clk_out4	100.000	N/A
- secalu bated				user_citit in incread - user_citit incread - user_citit	clk out5	clk out5	100.000	N/A
- secolul				- user_cits - cit_ini	clk out6	clk out6	100.000	N/A
					clk out7	_ clk_out7	100.000	N/A
						_		
					<			>
		ок	Cancel				ОК	Cancel

生成した IP を用いるようにコードを修正

- Verilog HDL のコードを、生成した IP を用いる様に code105.v の内容となるように変更する(赤色の文字の部分を入力あるいは変更する).
- Synthesis, Implementation をおこない、Bitstreamファイルを作成、FPGAにコ ンフィギュレーションする。
- 20MHz で動作するので, レジスタの変化が遅くなる.

code105.v

```
module m main (w clk, w led);
  input wire w clk;
                                                                                     w_clk_IBUF_inst
  output wire [3:0] w led;
                                                                          w clk
                                                                                                                    clk w0
  wire w_clk2, w_locked;
                                                                                                                         clk out1
                                                                                                             clk in1
  clk wiz 0 clk w0 (w clk2, 0, w locked, w clk);
                                                                                                              reset
                                                                                                                   clk wiz 0
              r out = 0;
  reg
  reg [31:0] r cnt = 0;
  always@(posedge w clk2) begin
    r cnt <= (r cnt==99999999) ? 0 : r cnt +1;</pre>
    r out <= (r cnt==0) ? ~r out : r out;</pre>
  end
  assign w_led = {r_out, r_out, r_out, r_out};
  vio_0 vio_00(w_clk2, w_led[3], w_led[2], w_led[1], w_led[0]);
endmodule
```



VIOを用いてFPGAの動作をリモートで確認

- Click + button in hw_vio_1 window.
- Select w_led_OBUF[0:0], w_led_OBUF_1[3:3], w_led_OBUF_2[2:2], and w_led_OBUF_3[1:1] by clicking them pushing Shift button. Then click OK.
- You will see the values of four LEDs change every five seconds because the 20MHz clock signal w_clk2 is used !

** Search: ? Probes for hw_vio_1 (4) * w_led_OBUF(0:0) * w_led_OBUF_2[2:2] * w_led_OBUF_2[2:2] * w_led_OBUF_3(1:1) * w_led_OBUF_1[3:3] * w_led_OBUF_1[3:3] * w_led_OBUF_1[3:3] * w_led_OBUF_2[2:2] * w_led_OBUF_3(1:1) * w_led_OBUF_1[3:3] * w_led_OBUF_3(1:1)	Add Probes X	со	ode057.v × hw_vios ×	1
Search: Q: Probes for hw_vio_1 Value & w_led_OBUF[0:0] w_led_OBUF_2[2:2] & w_led_OBUF_2[2:2] B) 1 & w_led_OBUF_3[1:1] Input hw_vio_1 Image: Search: & w_led_OBUF_2[2:2] Image: Search: & w_led_OBUF_3[1:1] Image: Search: W_led_OBUF_3[1:1] Image: Search: Image: Search: Image: Search:	€ ♦ 💠		hurvia 1	_
Probes for hw_vio_1 w_led_OBUF[0:0] w_led_OBUF_1[3:3] w_led_OBUF_2[2:2] w_led_OBUF_3[1:1] w_led_OBUF_1[3:3] w_led_OBUF_3[1:1] w_led_OBUF_3[1:1] w_led_OBUF_3[1:1] w_led_OBUF_3[1:1] w_led_OBUF_3[1:1] w_led_	earch: Q-	ы		-
<pre> W_vio_1 w_led_OBUF[0:0] w_led_OBUF_1[3:3] w_led_OBUF_2[2:2] w_led_OBUF_2[2:2] w_led_OBUF_3[1:1] w_led_OBUF_3[1:1] w_led_OBUF_3[1:1] W_vio_1 w_led_OBUF_3[1:1] W_vio_1 w_led_OBUF_3[1:1] W_vio_1 W_vi</pre>	robes for hw_vio_1 (4) V	tion		
ib m_led_OBUF_1[3:3] ib w_led_OBUF_2[2:2] ib w_led_OBUF_3[1:1]	T hw_vio_1	do	Name Value Activity Direction VIO	
w_led_OBUF_2(2:2) w_led_OBUF_3(1:1) w_led_OBUF_3(1:1) (B) 1 (B) 1 <td>w_led_OBUF_1[3:3]</td> <td>Dard</td> <td>L w_led_OBUF_2[2:2] [B] 1</td> <td></td>	w_led_OBUF_1[3:3]	Dard	L w_led_OBUF_2[2:2] [B] 1	
	1 w_led_0BUF_2[2:2]	hbo	🐌 w_led_OBUF[0:0] [B] 1 💲 Input hw_vio_1	
w_led_OBUF_3[1:1] [B] 1 Input hw_vio_1	1 w_led_OBUF_3[1:1]	Das	🐌 w_led_OBUF_1[3:3] [B] 1 💲 Input hw_vio_1	
			🐌 w_led_OBUF_3[1:1] 🕻 [B] 1 🔷 🛊 Input hw_vio_1	



演習(IPの構成方法, 100MHzと33MHzクロックを使う回路)

- 100MHzのクロック信号 w_clk と、33MHzのクロック信号 w_clk2 で異なるカウンタ r_cnt, r_cnt2 をインクリメントする次のコード code106.v をFPGAで動かす.
- プロジェクトに登録されている Verilog HDL のコードを code106.v の内容となるように変 更すること.
- r_out2 と比べて r_out は3倍の動作周波数 で動作しているので、例えば、VIO r_out2の 値が10の時に、r_outの値が約30になって いるはずである。

```
module m_main (w_clk, w_led);
input wire w_clk;
output wire [3:0] w_led;
```

```
wire w_clk2, w_locked; // 33MHz clock
clk_wiz_0 clk_w0 (w_clk2, 0, w_locked, w_clk);
```

```
reg [31:0] r_out = 0;
reg [31:0] r_cnt = 0;
always@(posedge w_clk) begin // 100MHz
r_cnt <= (r_cnt==99999999) ? 0 : r_cnt +1;
r_out <= (r_cnt==0) ? r_out+1 : r_out;
end
```

```
reg [31:0] r_out2 = 0;
reg [31:0] r_cnt2 = 0;
always@(posedge w_clk2) begin // 33MHz
r_cnt2 <= (r_cnt2==99999999) ? 0 : r_cnt2 +1;
r_out2 <= (r_cnt2==0) ? r_out2+1 : r_out2;
end
```

```
assign w_led = r_out[3:0];
vio_0 vio_00(w_clk, r_out);
vio_0 vio_01(w_clk2, r_out2);
endmodule
```

code106.v

演習(IPの構成方法, 100MHzと33MHzクロックを使う回路)

- Clocking Wizard でパラメータを適切に修正して, clk_wiz_0 を生成する.
- 32ビットのレジスタの値を VIO で表示できるように修正した vio_0 を生成する.
- 論理合成, 配置配線, bitstreamファイルを生成し, FPGAをコンフィギュレーションする.
- hw_vio_1 ウィンドウと hw_vio_2 ウィンドウを開く.
- それぞれのウィンドウで, r_out の値と r_out2 の値を表示する.
- それぞれのウィンドウで表示する32ビットのレジスタの値を、符号なし10進数の表示 (Unsigned Decimal) に変更する. (右クリック -> Radix -> Unsigned Decimal)
- 2つのウィンドウを左右に表示して、その画面を「担当の教員あるいはTAに」示すこと.





Check Point 3



新しい Vivado プロジェクトの作成とファイルの登録

- 前回の演習を参考に、Vivado で新しいプロジェクト project_4 を作成する.
- Ubuntu で起動したターミナルで、次のコマンドを実行してファイルをコピーする.
 - /home/tu_kise は automount のディレクトリなので、アクセスしないとファイルが見えない、tabキーによる補完がうまく動作しないことがあるので注意する。
 - 最後の ls コマンドで, code101.v, main11.xdc, cp4.txt が表示されることを確認.

```
$ ls /home/tu_kise
$ cd ~/cld/project_4
$ cp /home/tu_kise/cld/2023/code101.v .
$ cp /home/tu_kise/cld/2023/main11.xdc .
$ cp /home/tu_kise/cld/2023/cp4.txt .
$ ls
```

- Vivado で, project_4 の制約ファイルとして main11.xdc を登録する.
 - main11.xdc ファイルの内容を変更する必要はない.
- Vivado で, project_4のVerilog HDLファイルとして code101.v を登録する.

Code101.v ALU(Arithmetic and Logic Unit)の記述

```
module m_main (w_clk, w_a, w_c, w_dout, w_zero);
                   input wire w clk, w a, w c;
                   output wire w dout, w zero;
                   reg [31:0] r a=0, r b=0;
                   reg [3:0] r c=0;
                   wire [31:0] w out;
                   assign w dout = ^w out;
                   always@(posedge w_clk) begin
                       r_a <= {w_a, r_a[31:1]};</pre>
                       r b <= w out;
                       r_c <= {w_c, r_c[3:1]};</pre>
                   end
                   m_ALU m_ALU0 (r_c, r_a, r_b, w_out, w_zero);
               endmodule
               module m_ALU (w_Ctl, w_A, w_B, r_Out, w_Zero);
                   input wire [3:0] w Ctl;
                   input wire [31:0] w A, w B;
                   output reg [31:0] r_Out;
                   output wire w Zero;
                   assign w Zero = (r Out==0);
                   always@(*)
                     case (w Ctl)
                         0: r_Out <= w_A & w_B;
                         1: r Out <= w A | w B;
                         2: r Out <= w A + w B;
                         6: r Out <= w A - w B;
                         7: r Out <= (w A < w B) ? 1 : 0;
                         12: r Out <= \sim(w A | w B);
                         default: r Out <= 0;</pre>
                     endcase
code101.v
               endmodule
```



CSC.T341 Computer Logic Design, Department of Computer Science, TOKYO TECH

設定の変更(モジュール間の最適化をしない,モジュールのハードウェア 量を計測するため)

- Click Settings.
- Select none for -flatten_hierarchy and click Apply.
- If you have Create New Run window, select No in the window.
- Click OK in Setting window.



Project Settings	Specify various settings assoc	iated to Synthesis
General Simulation Elaboration	Constraints	
Synthesis	Default constraint set:	constrs 1 (active)
Implementation Bitstream	Report Options	-
> IP	Strategy: 🗯 Vivado Synt	hesis Default Reports (Vivado Synthes
Fool Settings	Options	
Project IP Defaults Board Repository	Write Incremental Synthe	esis
Example Project Repository	Incremental synthesis: No	t set
Source File Display	S <u>t</u> rategy:	Vivado Synthesis Defaults* (Vi 🗸
WebTalk	Description: Viv	ado Synthesis Defaults
Help	✓Svnth Design (vivado)	
> Text Editor	tcl.pre	
3rd Party Simulators	tcl.post	
Selection Bules	-flatten_hierarchy*	none
Shortcuts	-gated_clock_conversion	off
Strategies	-bufg	12
Remote Hosts	-fanout_limit	10,000
Window Behavior	-directive	Default
	-flatten_hierarchy* Flatten hierarchy during LUT n	napping.

ハードウェア量の計測



AND と OR のそれぞれのハードウェア量を計測する場合

- Verilog HDL のコードを編集して AND (&) のみのALUとして, 同様にハードウェア量を 計測する.
- 次に, OR ())のみのALUとして、同様にハードウェア量を計測する.
- 同様に,加算 (+),減算 (-),大小比較 (<), NOR (~ |)のハードウェア量を計測する.
- また,全てのコメントアウトを削除して,全ての演算を含む ALU 全体のハードウェア量を 計測する.

```
module m_ALU (w_Ctl, w_A, w_B, r_Out, w_Zero);
input [3:0] w Ctl;
```

```
input [31:0] w_A, w_B;
output reg [31:0] r_Out;
output wire w_Zero;
assign w Zero = (r Out==0);
```

```
always@(*)
     case (w_Ctl)
            0: r Out <= w A & w B;
           1: r Out <= w A \mid w B;
  11
  11
          2: r Out <= w A + w B;
  11
           6: r Out <= w A - w B;
  11
           7: r_Out <= (w_A < w_B) ? 1 : 0;
            12: r Out <= \sim(w A | w B);
  11
          default: r Out <= 0;</pre>
      endcase
endmodule
```

AND のハードウェア量を計測する場合

```
module m ALU (w Ctl, w A, w B, r Out, w Zero);
    input [3:0] w Ctl;
    input [31:0] w A, w B;
    output reg [31:0] r Out;
    output wire w Zero;
    assign w Zero = (r Out==0);
    always@(*)
      case (w_Ctl)
            0: r Out <= w A & w B;
   11
            1: r Out <= w A | w B;
           2: r Out <= w A + w B;
   11
           6: r Out <= w A - w B;
  11
            7: r Out <= (w_A < w_B) ? 1 : 0;
  11
   11
            12: r Out <= \sim(w A | w B);
          default: r Out <= 0;</pre>
      endcase
endmodule
```

OR のハードウェア量を計測する場合

演習(計測したハードウェア量をまとめる)

- cp4.txt の xxxxx, yyyyy の部分を適切な数字で置き換えること.
- それぞれの演算を含む ALU で必要となる LUT の数 (# of LUTs), それを FPGA に搭載できる数 (# of modules on an FPGA) を記入する. FPGA に搭載されている LUT が 20,800個であることから, 20,800/(# of LUTs) によって, FPGA に搭載できるモジュールの数を計算すること.
- また、&、 |、 +、 -、 <、 ~ | の6種類の演算を含む ALU で必要となる「LUTの数」の和を計算して、 Sum の行に記入すること、このLUTの数から、 FPGA に搭載できるモジュール数を計算して記入すること、
- すべての演算を含む ALU (code101.v の記述)のために必要となる LUT の数を ALU の行に記入 する. 同様に, FPGA に搭載できるモジュール数を記入すること.
- 「Sum の行に記入した LUT の数と, ALU の行に記入した LUT の数が異なる理由」を考えて, 「担当の教員あるいはTA」に伝えること。

cp4.txt	operation	# of LUTs	# of modules on an FPGA
	<pre>r_Out <= w_A & w_B; r_Out <= w_A w_B; r_Out <= w_A + w_B; r_Out <= w_A - w_B; r_Out <= (w_A < w_B) ? 1 : 0; r_Out <= ~(w_A w_B);</pre>	xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx	УУУУУ УУУУУ УУУУУ УУУУУ УУУУУ УУУУУ
	Sum	xxxxx	ууууу
	ALU	xxxxx	ן אַעאַע





CSC.T341 Computer Logic Design, Department of Computer Science, TOKYO TECH

ALE.

References

- Computer Logic Design support page
 - https://www.arch.cs.titech.ac.jp/lecture/CLD/
- ACRi Room
 - https://gw.acri.c.titech.ac.jp
- ACRi Blog
 - https://www.acri.c.titech.ac.jp/wordpress/
- 情報工学系計算機室
 - http://www.csc.titech.ac.jp/
- Xilinx Vivado Design Suite
 - https://japan.xilinx.com/products/design-tools/vivado.html
- Digilent Arty A7-35T
 - https://reference.digilentinc.com/reference/programmable-logic/arty-a7/start
- Verilog HDL
 - https://ja.wikipedia.org/wiki/Verilog