2023年度（令和5年）版

Ver. 2023-10-17d

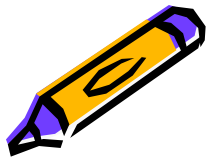
Course number: CSC.T363

コンピュータアーキテクチャ 演習(2)
Computer Architecture Exercise(2)

情報工学系　吉瀬謙二, Berjab Nesrine

Kenji KISE, Department of Computer Science

kise_at_c.titech.ac.jp



コンピュータアーキテクチャ 演習(Exercise)の注意点

- 連絡手段は Slack を利用します.
- 演習は 15:25～17:05 です.
  - 20分以上(15:45までに)遅刻したら欠席扱いになります.
  - 前半は課題の説明(15分程度)で, 後半は課題の解決とチェックポイントの確認.
- 演習は手元の FPGA ボードと 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 TA 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.

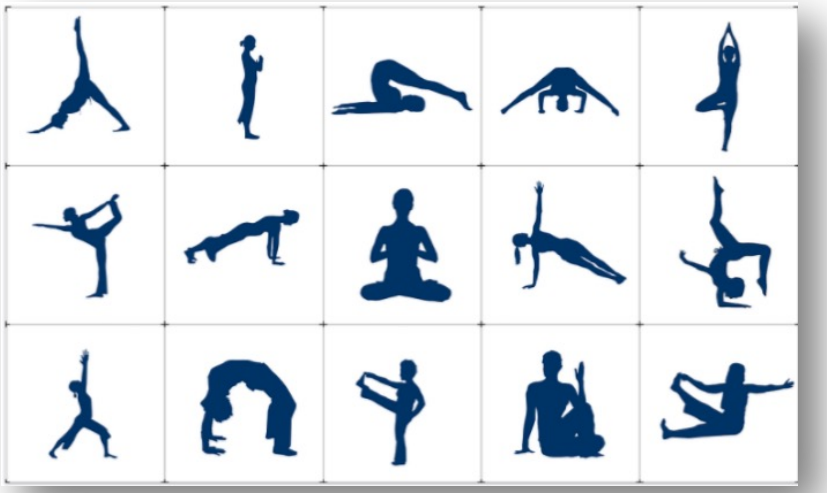
- 演習時間以外も手元の FPGA ボードと ACRi ルームを利用できます.
  - 手元の FPGA ボードを借りることができます.
  - 独自のハードウェア設計などに挑戦しましょう.

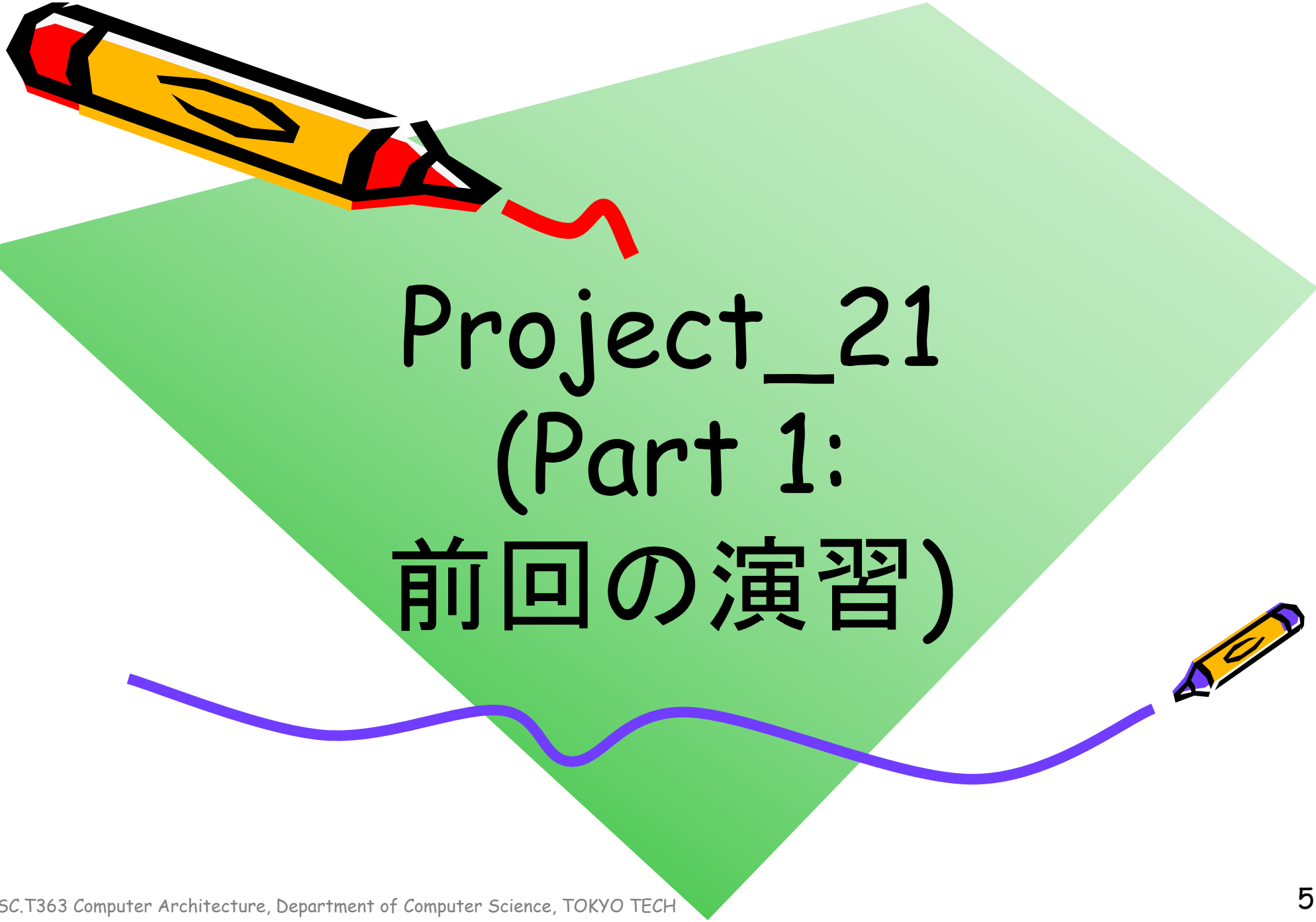
【重要】ACRiルームのサーバの予約

- ACRiルームのアカウントを使って，次のURLからログインする．
  - https://gw.acri.c.titech.ac.jp/wp
- 「予約ページトップ」から，vsで始まるサーバで演習の日の15:00～18:00の枠を予約すること．

Exercise(2)

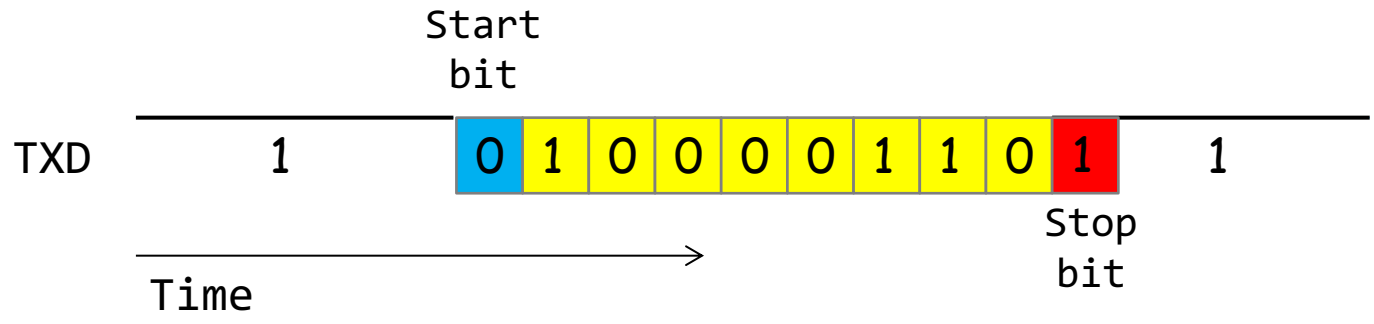
- Project_21
  - UARTの使い方を理解する.
    - 1ワード (4バイト) の add 命令を転送し，VIOを利用して正しく送信されていることを確認する.
    - 自分で作る命令列を FPGA で動かす.
      - 受信したデータのチェックサムを求めて VIOを利用して出力する．得られるチェックサムが正しいことを確認する.




Project_21
(Part 1:
前回の演習)

UART (Universal Asynchronous Receiver/Transmitter)

- 調歩同期方式によるシリアル信号をパラレル信号に変換したり，その逆方向の変換をおこなう集積回路をUARTと呼ぶ．8ビット(1バイト)単位でデータを送信・受信する．
- UARTを用いることで，FPGAとコンピュータの間でのお手軽なデータ通信が可能．
- 例えば，'a' という文字を送信する場合，'a' は 8'h61, 8'b01100001 (次スライドのASCII Tableを参照)なので，下図のタイミングで送信線TXDを制御する．
  - データが送信されるまで送信線TXD を1とする．
  - まず，青色で示した0 (これをスタートビットと呼ぶ)を送信することで，データ送信の開始を明示．
  - 次に，黄色で示した様に送信したいデータ 8'b01100001 の最下位ビットから順番に送信する．
  - 最後に，赤色で示した1(これをストップビットと呼ぶ)を送信する．
- 1ビットを送受信するための時間間隔は送信側と受信側で同じレートを用いる．これをボー・レート (baud) と呼ぶ．例えば，1000 baud であれば，1ビット送信の間隔は 1msec となる．



シリアル通信による送信回路 m_UartTx

- システムクロック 50MHz，1Mbaud を想定する送信回路
- トップのモジュール m_main では，2秒に1回の頻度で，文字 a を送信する．

code201.v

```
/*************************************************************************************/
/* code201.v                For CSC.T363 Computer Architecture, Archlab TOKYO TECH */
/*************************************************************************************/
`timescale 1ns/100ps
`default_nettype none
/*************************************************************************************/
module m_main (w_clk100, w_txd);
   input  wire w_clk100;
   output wire w_txd;
   reg r_clk = 0;
   always@(posedge w_clk100) r_clk <= ~r_clk; // 50MHz clock signal
   reg [31:0]  r_cnt = 1;
   always@(posedge r_clk) r_cnt <= (r_cnt>(100_000_000 - 1)) ? 0 : r_cnt+1;
   m_UartTx m_UartTx0(r_clk, 8'h61, (r_cnt==0), w_txd);
endmodule
/*************************************************************************************/
module m_UartTx (w_clk, w_din, w_we, w_txd);
   input wire       w_clk, w_we;
   input wire [7:0] w_din;
   output wire      w_txd;
   reg [8:0]        r_buf = 9'b111111111;
   reg [7:0]        r_wait= 0;
   always@(posedge w_clk) begin
      r_wait <= (w_we) ? 0 : (r_wait>=49) ? 0 : r_wait + 1;
      r_buf  <= (w_we) ? {w_din, 1'b0} : (r_wait>=49) ? {1'b1, r_buf[8:1]} : r_buf;
   end
   assign w_txd = r_buf[0];
endmodule
/*************************************************************************************/
```

シリアル通信による受信回路 m_UartRx

- システムクロック 50MHz，1Mbaud を想定する受信回路
- 受信した8ビットのデータを r_dout に出力し，そのことを伝えるために r_en を1にする．

code202.v

```
/*********************************************************************************/
module m_UartRx (w_clk, w_rxd, w_dout, r_en);
   input  wire       w_clk, w_rxd;
   output wire [7:0] w_dout;
   output reg        r_en = 0;
   reg [2:0] r_detect_cnt = 0; /* to detect the start bit */
   always @(posedge w_clk) r_detect_cnt <= (w_rxd) ? 0 : r_detect_cnt + 1;
   wire w_detected = (r_detect_cnt>2);
   reg       r_busy = 0;
   reg [3:0] r_cnt  = 0;
   reg [7:0] r_wait = 0;
   always@(posedge w_clk) r_wait <= (r_busy==0) ? 0 : (r_wait>=49) ? 0 : r_wait + 1;
   reg [8:0] r_data = 0;
   always@(posedge w_clk) begin
      if (r_busy==0) begin
         {r_data, r_cnt, r_en} <= 0;
         if(w_detected) r_busy <= 1;
      end
      else if (r_wait>= 49) begin
         r_cnt  <= r_cnt + 1;
         r_data <= {w_rxd, r_data[8:1]};
         if (r_cnt==8) begin r_en <= 1; r_busy <= 0; end
      end
   end
   assign w_dout = r_data[7:0];
endmodule
/*********************************************************************************/
```

シリアル通信のための m_main

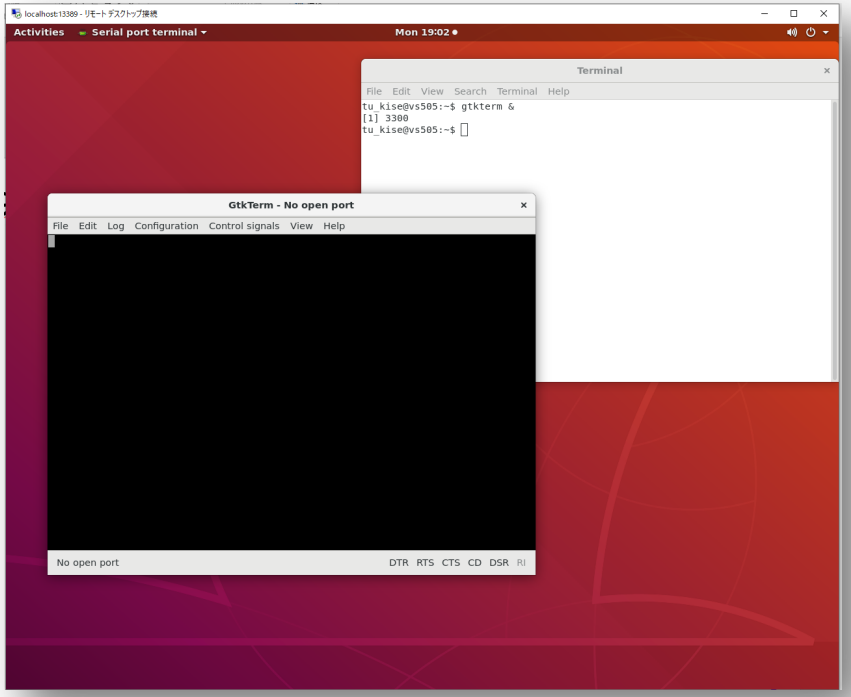
- 50MHz のシステムクロック，1Mbaud (1,000,000 baud) のUARTで，「受信した文字」をそのまま送信するモジュール m_main

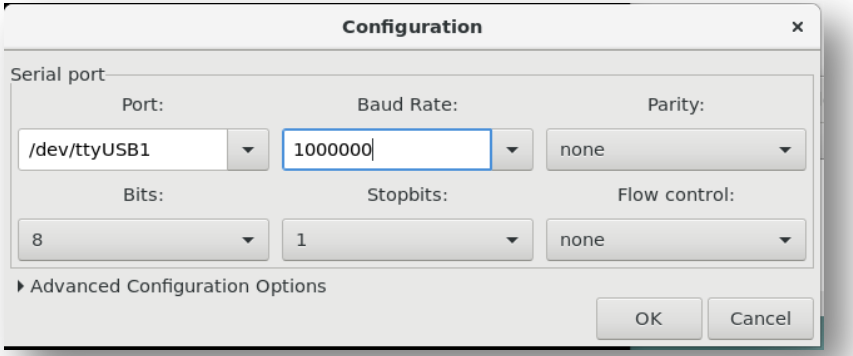
code202.v

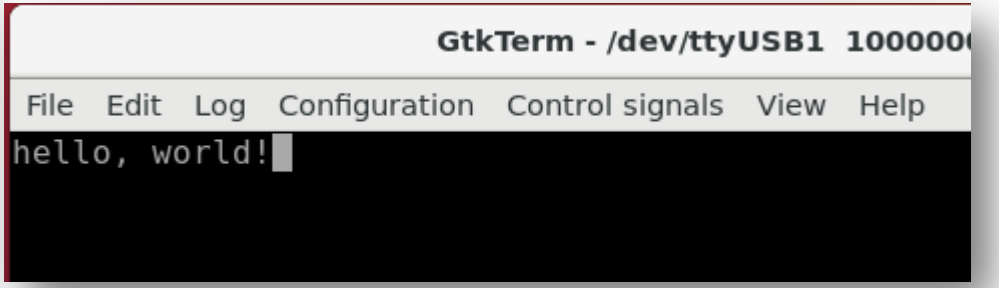
```
module m_main (w_clk100, w_txd, w_rxd);
   input  wire w_clk100, w_rxd;
   output wire w_txd;
   wire [7:0] w_dout;
   wire w_en;
   reg r_clk=0;
   always @(posedge w_clk100) r_clk=~r_clk;
   m_UartRx m_UartRx0(r_clk, w_rxd, w_dout, w_en);
   m_UartTx m_UartTx0(r_clk, w_dout, w_en, w_txd);
```

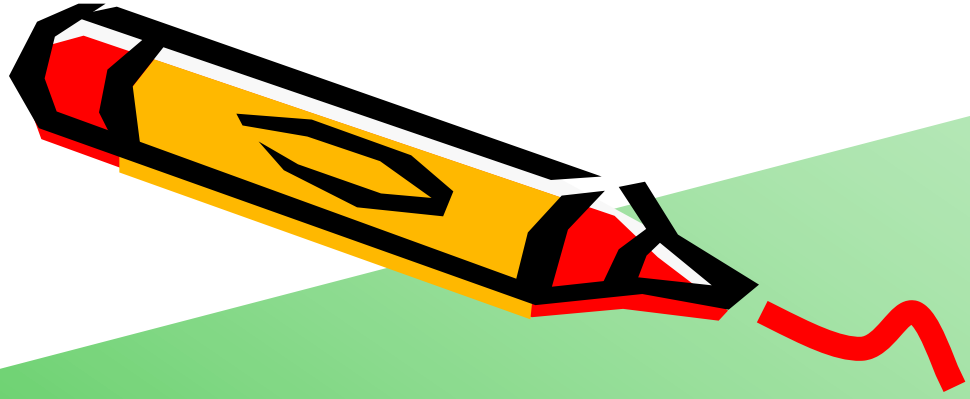
GtkTerm を立ち上げて動作確認

- コマンド gtkterm & で GtkTerm を起動する.
  - Port として /dev/ttyUSB1 を選択する.
  - Baud Rate に 1000000 を入力して，1Mbaud に設定する.
    - コマンド：gtkterm -p /dev/ttyUSB1 -s 1000000 &)
- GtkTerm で入力した文字が表示されることを確認する.

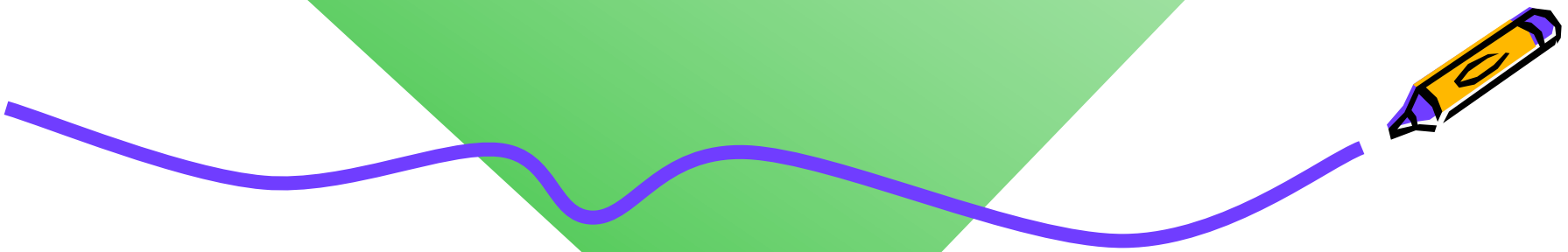






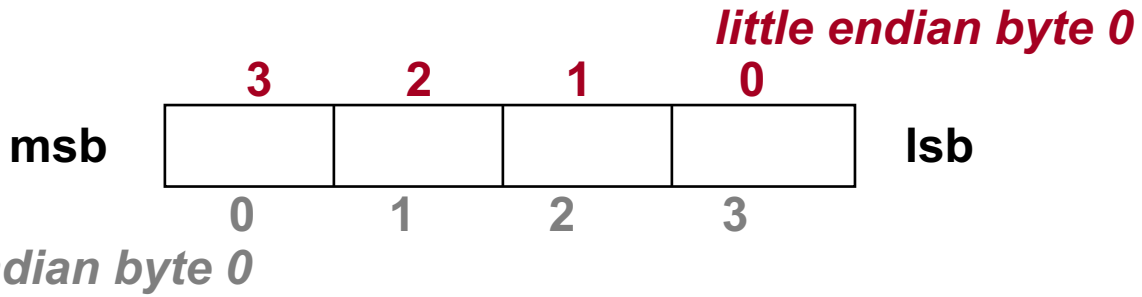


Project_21 (Part 2)



Byte Addresses

- Since 8-bit bytes are so useful, most architectures address individual bytes in memory
  - The memory address of a word must be a multiple of 4 (alignment restriction)
- Big Endian:
  - Leftmost byte is word address
  - IBM 360/370, Motorola 68k, MIPS, SPARC, HP PA
- Little Endian:
  - Rightmost byte is word address
  - Intel 80x86, DEC Vax, DEC Alpha (Windows NT)



RISC-V の add 命令のエンコード

- RISC-V の 次の命令を32ビットにエンコードして16進数で表示するコード
  - add x12, x1, x2
    - CLD-7 lecture materials (https://www.arch.cs.titech.ac.jp/lecture/CLD/Lec-2023-05-01.pdf)

code204.v

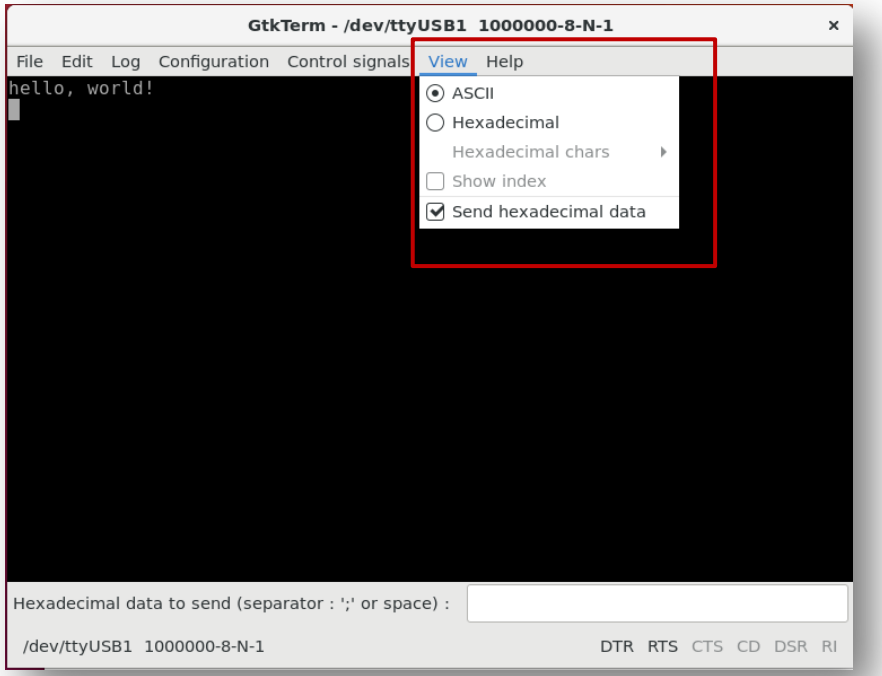
```
module m_top();
   reg [31:0] r_insn = {7'h0, 5'd2, 5'd1, 3'h0, 5'd12, 7'b0110011};  //add x12, x1, x2

   initial begin
      $write("%x\n", r_insn);
      $finish();
   end
endmodule
```

```
$ iverilog code204.v
$ ./a.out
00208633
```

- シリアル通信で (1) 0x33 (2) 0x86 (3) 0x20 (4) 0x00 という順番で送信して，レジスタの値を 00_20_86_33 に設定したい．

GtkTerm で送信するデータを16進数で指定

- View -> Send hexadecimal data を選択する.
- View の下に send hexadecimal data を指定する欄が表示される.
- ここに 33 86 20 00 と入力して改行する (リターンキーを押す) ことで指定した4バイトのデータを送信できる.



add 命令を転送し，VIOを利用して確認 (1/4)

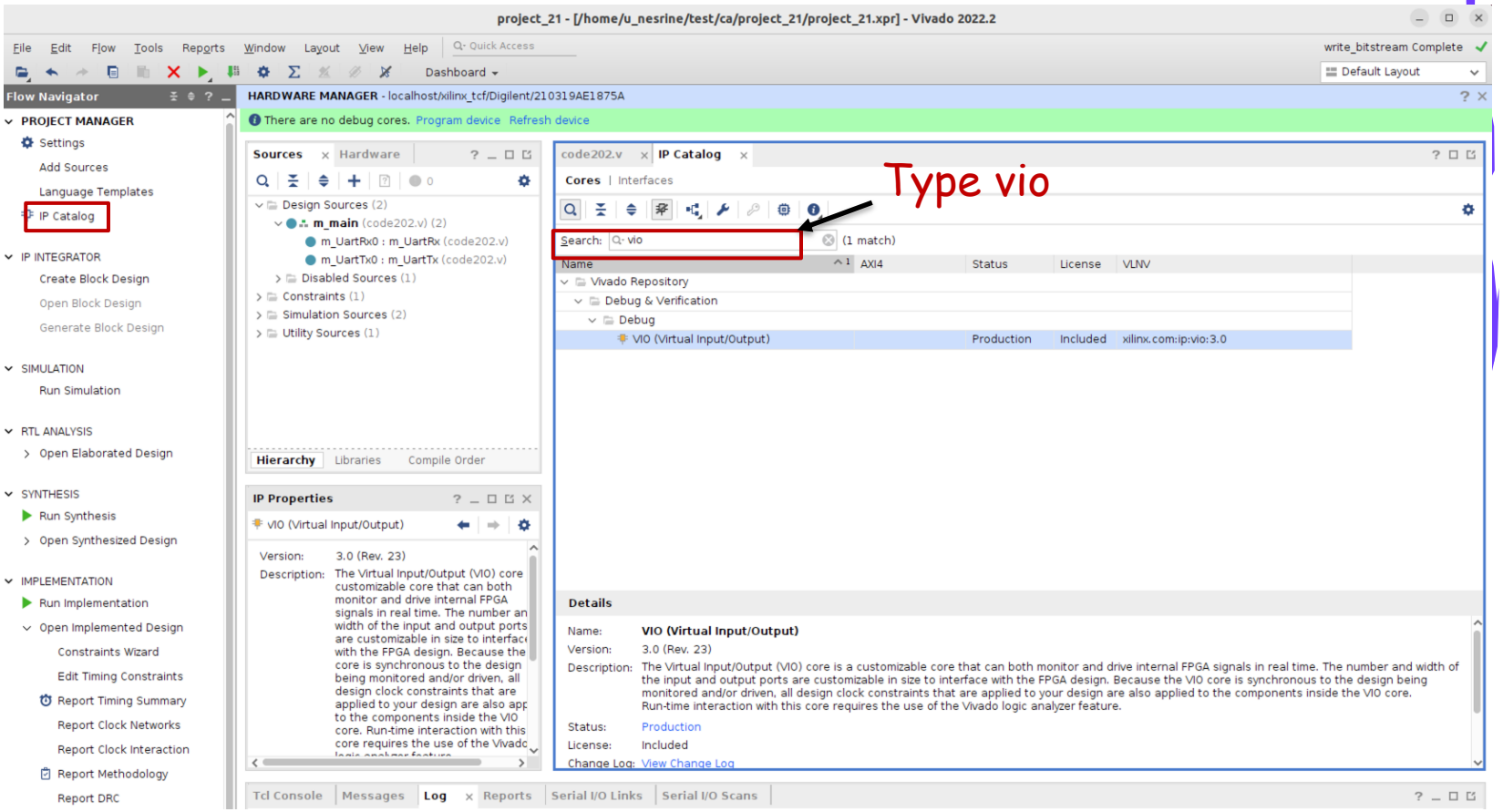
- code203.v の論理合成，配置・配線，コンフィギュレーション
  - 1バイト受信するたびに32ビットのレジスタを8ビット右にシフトして，上位の8ビットに受信した8ビットを連結する．
  - Little Endian で送信した4バイトを受信して，正しく32ビットのレジスタに格納して，その値をVIO で表示する．

code203.v

```
module m_main (w_clk100, w_txd, w_rxd);
   input  wire w_clk100, w_rxd;
   output wire w_txd;
   wire [7:0] w_dout;
   wire w_en;
   reg r_clk=0;
   always @(posedge w_clk100) r_clk=~r_clk;

   m_UartRx m_UartRx0(r_clk, w_rxd, w_dout, w_en);
   m_UartTx m_UartTx0(r_clk, w_dout, w_en, w_txd);

   reg[2:0] r_cnt=0;
   reg[31:0] r_data=0;
   always@(posedge r_clk) if(w_en) r_data<={w_dout, r_data[31:8]};
   always@(posedge r_clk) r_cnt<=(r_cnt==4) ? 0 : (w_en)?r_cnt+1:r_cnt;
   vio_0 vio_00(r_clk, r_data);

endmodule
```

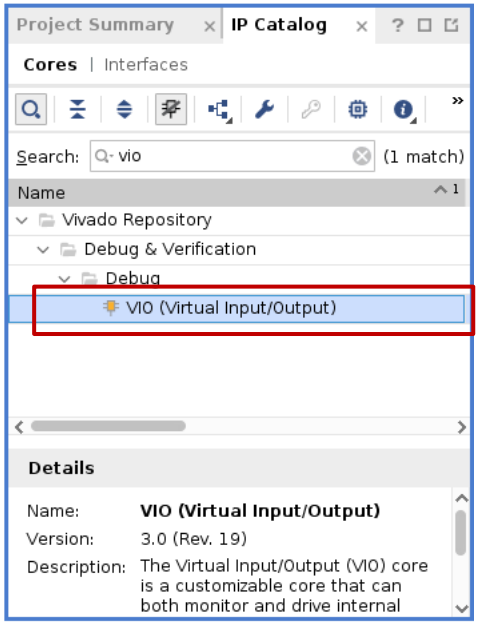
add 命令を転送し，VIOを利用して確認 (2/4)

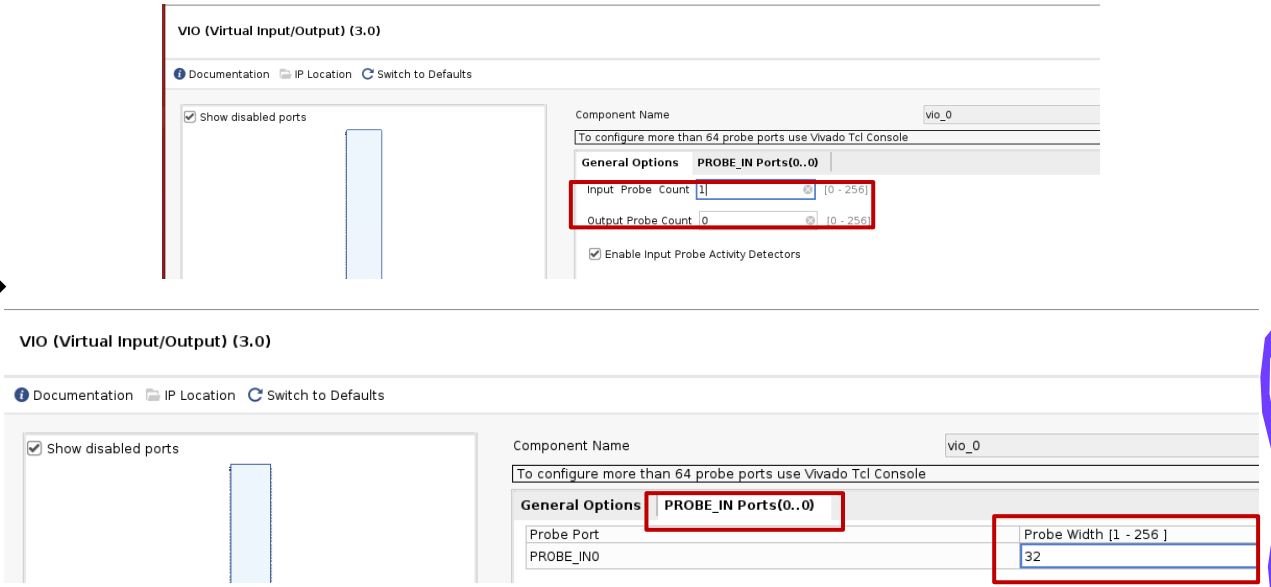
- VIO configuration:
  - Click IP Catalog, and type vio in Search area to use VIO (Virtual Input/Output)



addi命令を転送し，VIOを利用して確認 (3/4)

- Double click VIO (Virtual Input/Output).
- In VIO window:
  - General Options: set 1 for Input Probe Count, set 0 for Output Probe Count, and click OK.
  - PROBE_IN PORTS(0..0): set 32 for Probe Width [1-256]

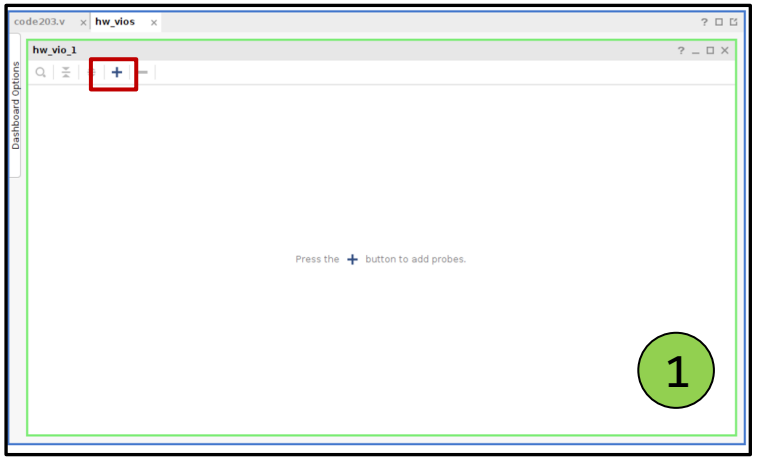


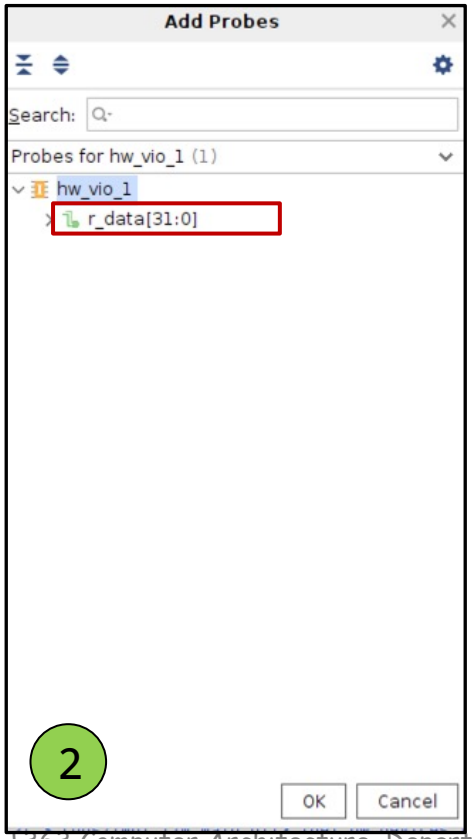


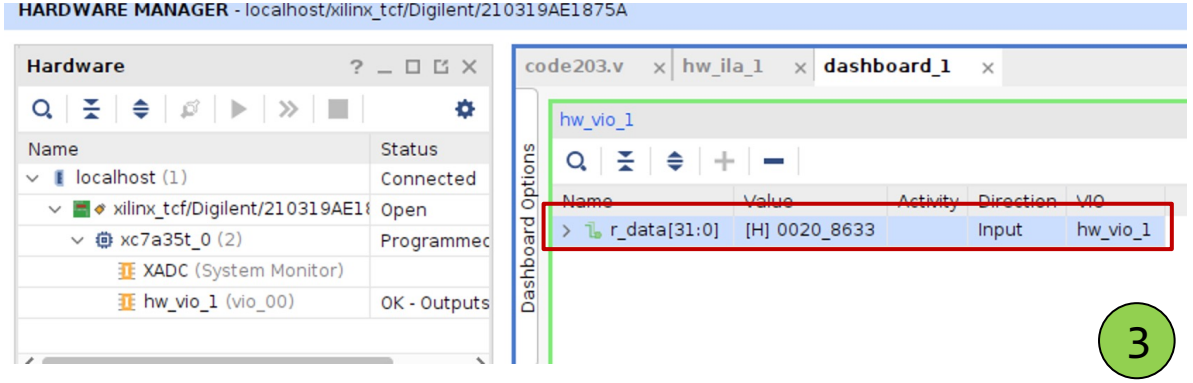
- Click Generate, and click OK if asked in Generate Output Products window.
- Click Generate Bitstream, click Yes, click OK, and wait.

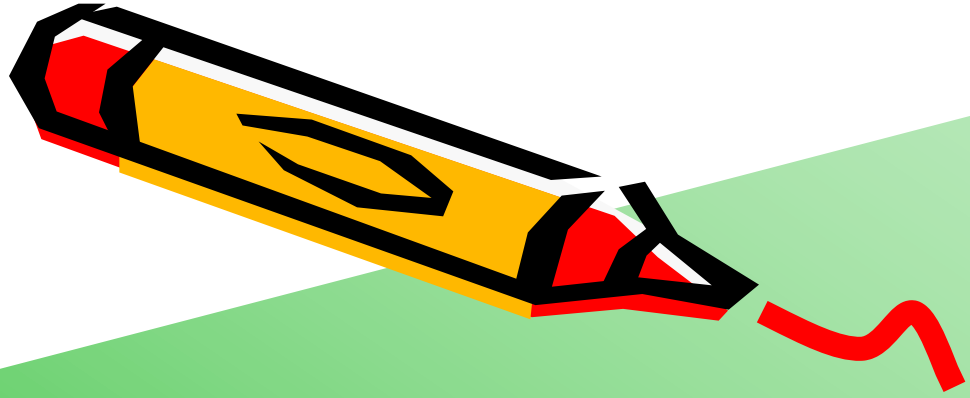
add 命令を転送し，VIOを利用して確認 (4/4)

- Click +button in hw_vio_1 window, select r_data[31:0] and the click OK.

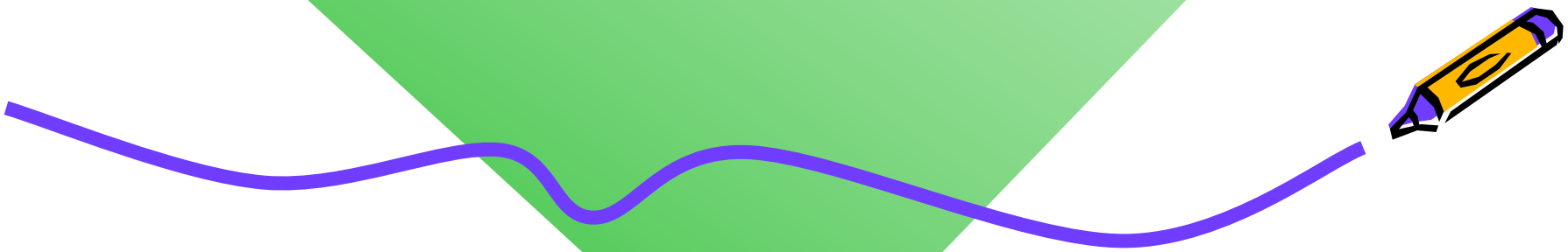








Project_21 (Part 3)



自分で作った命令列を FPGA で動かす (1/4)

- Example: FPGAで Fibonacci 数を実行する
  - Step1: Translating from C to RISC-V

fib.c

```
/*************************************************************************
/
/* fib.c           For CSC.T363 Computer Architecture, Archlab TOKYO TECH */
/*************************************************************************
/
#include <stdio.h>

int n = 9;

// Function to find the nth Fibonacci number
int main(void) {
    int current_fib = 0, next_fib = 1;
    int new_fib;
    for (int i = n; i > 0; i--) {
        new_fib = current_fib + next_fib;
        current_fib = next_fib;
        next_fib = new_fib;
    }
    printf("%d\n", current_fib);
    return 0;
}
```

fib.s (translation of fib.c into RISC-V)

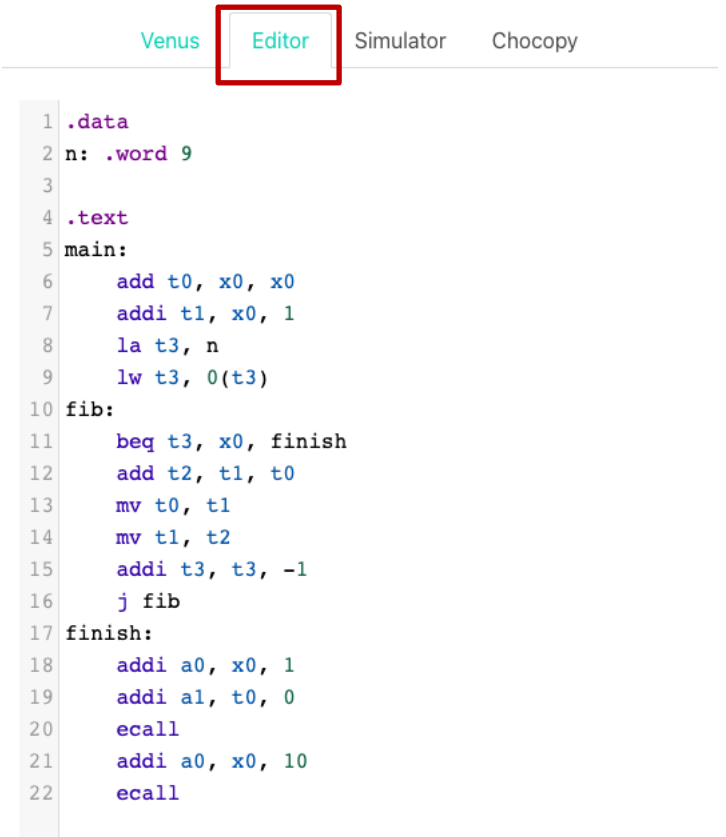
```
/*************************************************************************/
/* fib.s           For CSC.T363 Computer Architecture, Archlab TOKYO TECH */
/*************************************************************************/

.data
n: .word 9

.text
main:
    add t0, x0, x0 # current_fib = 0
    addi t1, x0, 1 # next_fib = 1
    la t3, n # load the address of the label n
    lw t3, 0(t3) # get the value that is stored at the adddress with the label n
fib:
    beq t3, x0, finish # exit loop once n iterations are completed
    add t2, t1, t0 # new_fib = current_fib + next_fib;
    mv t0, t1 # current_fib = next_fib;
    mv t1, t2 # next_fib = new_fib;
    addi t3, t3, -1 # decrement counter
    j fib # loop
finish:
    addi a0, x0, 1 # argument to ecall to execute print integer
    addi a1, t0, 0 # argument to ecall, the value to be printed
    ecall # print integer ecall
    addi a0, x0, 10 # argument to ecall to terminate
    ecall # terminate ecall
```

自分で作った命令列を FPGA で動かす (2/4)

Step2: Venus シミュレーターの使い方を理解する.

- ウェブブラウザで https://venus.cs61c.org を開く.
- 「ヴィーナス」を使い始めるには、「Editor」と「Simulator」タブを主に利用します。

- エディタータブをクリックします. エディタータブはアセンブリコードの編集を行う場所です.
- シミュレーション用のプログラムを用意:
  - シミュレーションしたいアーキテクチャのプログラムやコードを用意します.
  - これは通常、アセンブリ言語や機械語で書かれたものです.

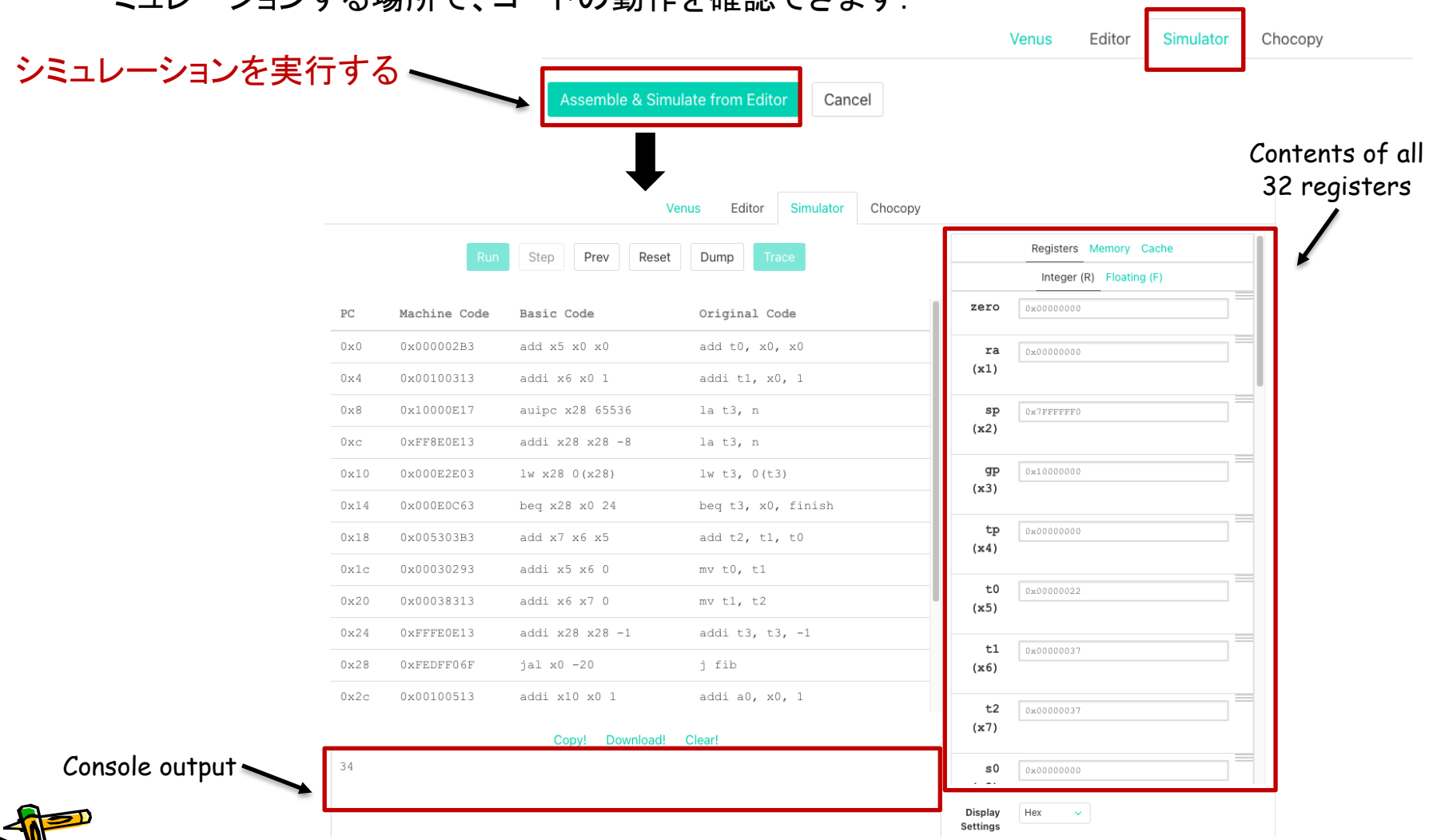



```
.data
n: .word 9

.text
main:
    add t0, x0, x0
    addi t1, x0, 1
    la t3, n
    lw t3, 0(t3)
fib:
    beq t3, x0, finish
    add t2, t1, t0
    mv t0, t1
    mv t1, t2
    addi t3, t3, -1
    j fib
finish:
    addi a0, x0, 1
    addi a1, t0, 0
    ecall
    addi a0, x0, 10
    ecall
```

fib.s

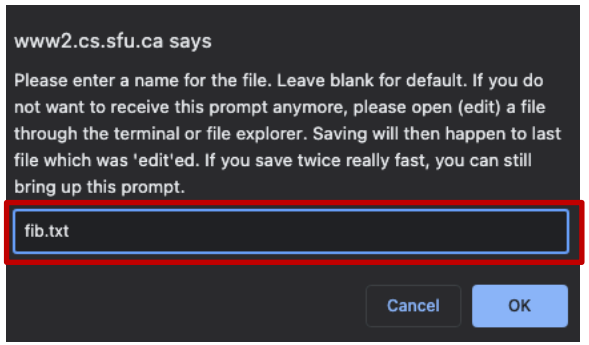
自分で作った命令列を FPGA で動かす (3/4)

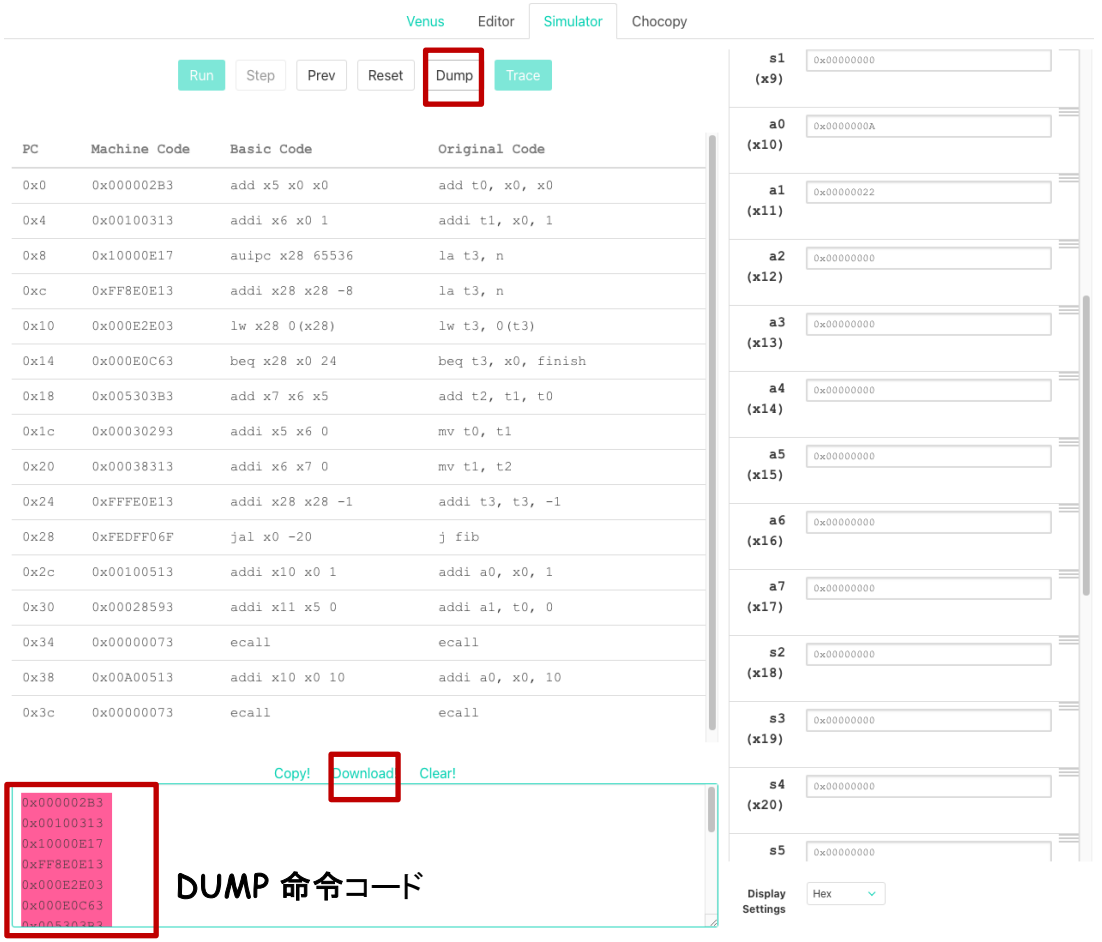
- シミュレータータブをクリックします．シミュレータータブは、アセンブリコードを実行またはシミュレーションする場所で、コードの動作を確認できます．



自分で作った命令列を FPGA で動かす (4/4)

- Step3: Dump タブをクリックします.
  - “Dump ”は、自分のコード中のすべての命令の16進表現のダンプを提供します.
- Step4: Download ボタンをクリックします.
  - fib.txtを入力します.





Conversion to binary file and checksum calculation

main01.c

```
/*****************************************************************************/
/* main01.c          For CSC.T363 Computer Architecture, Archlab TOKYO TECH */
/*****************************************************************************/

#include <stdio.h>

int main() {
    FILE* file = fopen("fib.txt", "r"); // Open the file for reading

    FILE *fp =fopen("test.bin", "wb"); //To generate test.bin


    int checksum = 0;

    if (file == NULL) {
        perror("Error opening file");
        return 1;
    }

    int data;
    char line[100]; // Assuming lines won't exceed 100 characters

    while (fgets(line, sizeof(line), file) != NULL) {
        if (sscanf(line, "%x", &data) == 1) {
            fwrite(&data, 4, 1, fp);
            checksum += data;

        } else {
            printf("Failed to read a hexadecimal value from the file.\n");
        }
    }

    printf("checksum %x\n", checksum);
    fclose(file);
    fclose(fp); // Close the files

    return 0;
}
```

- Step5: 「fib.txt」ファイルに含まれるマシンコードをバイナリファイルに変換します.
  - test.bin が生成されます.
- Fibonocci のマシンコードのチェックサムの値は 0xfa474d0 となります.

```
$ gcc main01.c
$ ./a.out
checksum fa474d0
```

転送されたデータのチェックサムを出力

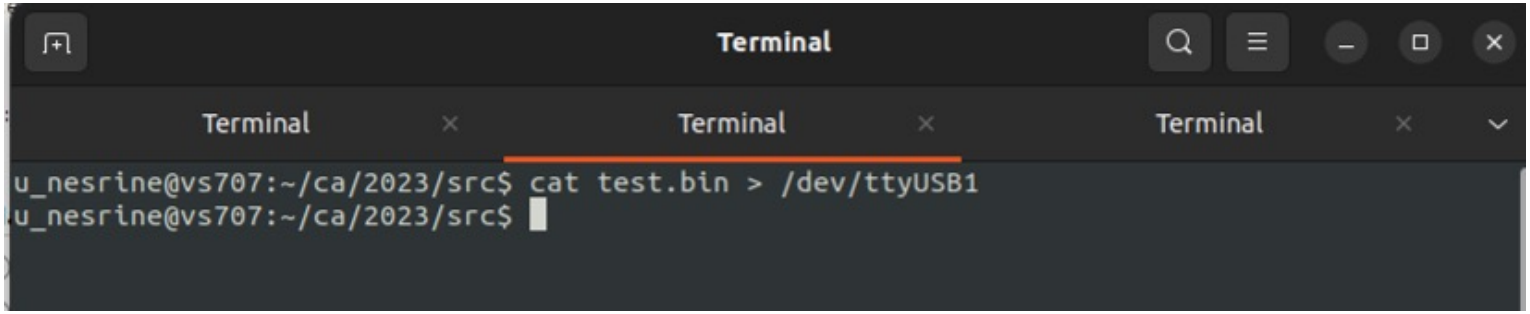
- Step6: code203.v を修正し，論理合成，配置・配線，コンフィギュレーション

code203.v

```
module m_main (w_clk100, w_txd, w_rxd);
   input  wire w_clk100, w_rxd;
   output wire w_txd;
   wire [7:0] w_dout;
   wire w_en;
   reg r_clk=0;
   always @(posedge w_clk100) r_clk=~r_clk;

   m_UartRx m_UartRx0(r_clk, w_rxd, w_dout, w_en);
   m_UartTx m_UartTx0(r_clk, w_dout, w_en, w_txd);

   //checksum calculation
   reg[2:0] r_cnt=0;
   reg[31:0] r_sum=0;
   reg[31:0] r_data=0;
   always@(posedge r_clk) if(w_en) r_data<={w_dout, r_data[31:8]};
   always@(posedge r_clk) r_cnt<=(r_cnt==4) ? 0 : (w_en)?r_cnt+1:r_cnt;
   always@(posedge r_clk) if(r_cnt==4) r_sum<=r_sum+r_data;
   vio_0 vio_00(r_clk, r_sum);
endmodule
```

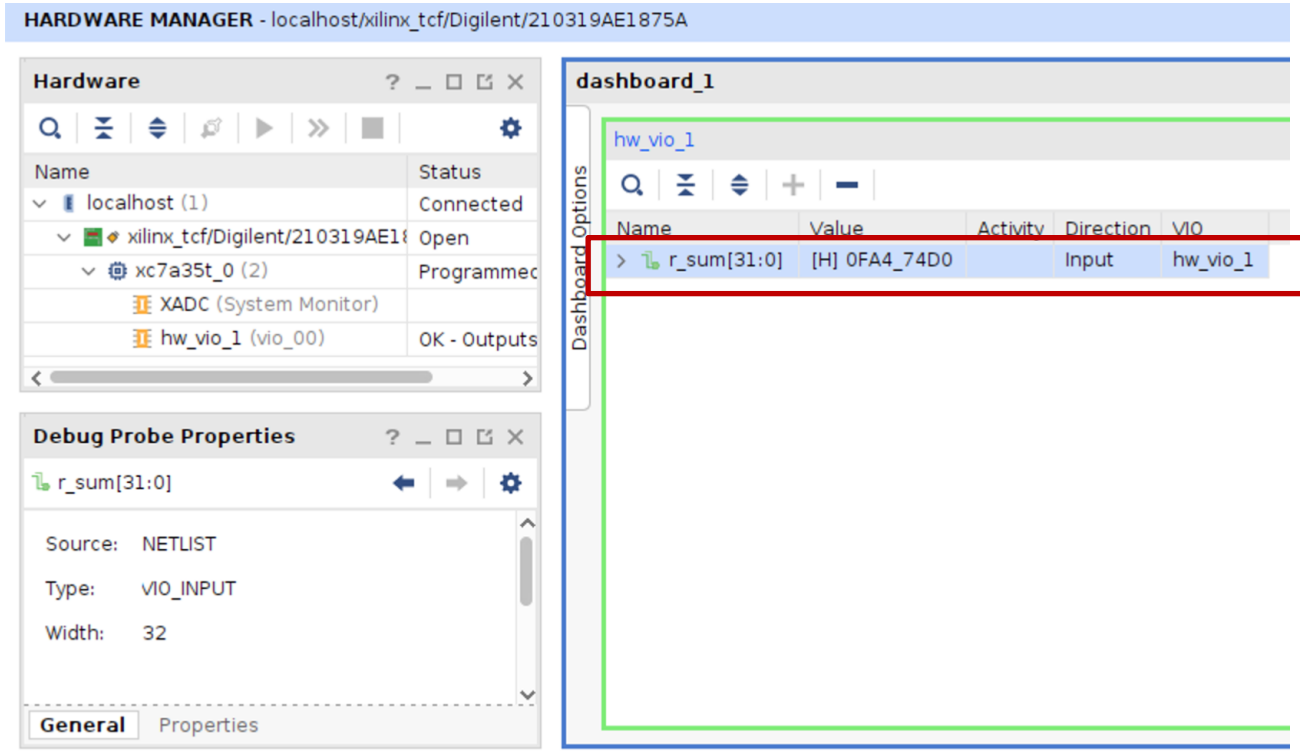
ファイル test.bin の内容を送信

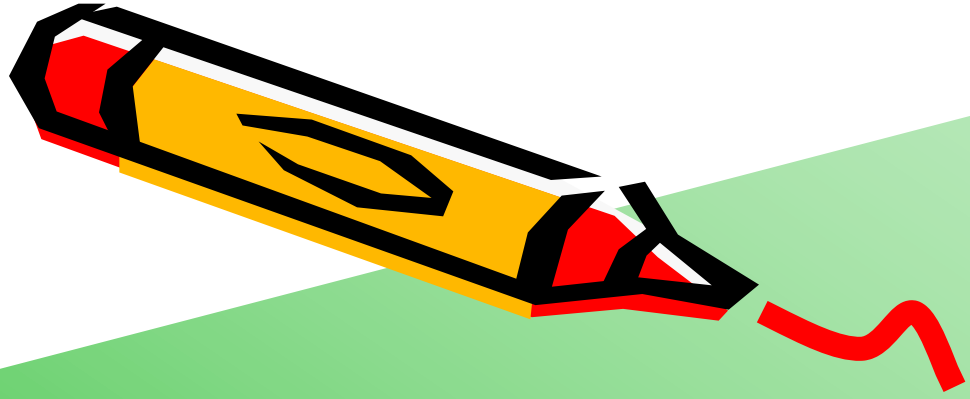
- Step7:ターミナルから次のコマンドを実行
  - cat test.bin > /dev/ttyUSB1
- GtkTerm でボーレートなどを設定してからコマンドを実行すること.
  - 次のコマンドで設定を確認できる. stty -a < /dev/ttyUSB1



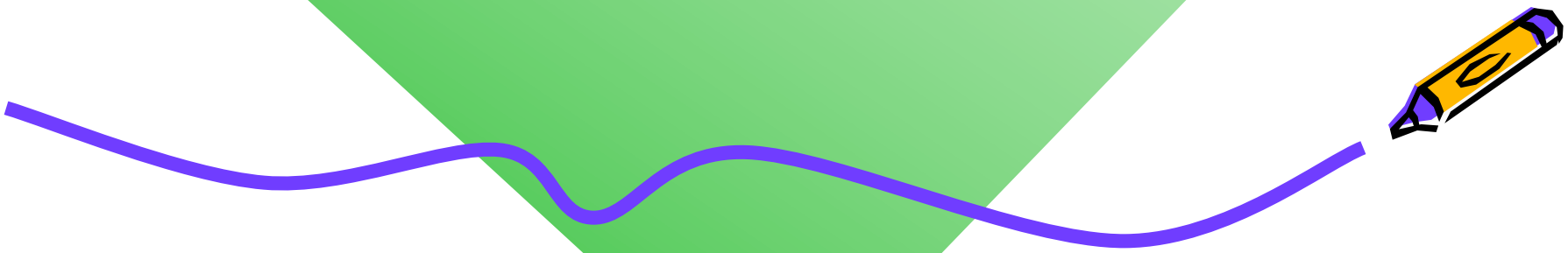
転送されたデータのチェックサムを出力

- Step8:生成した test.bin を送信して，転送されたデータの値が 0xfa474d0 となることを VIO を用いて確認します．





References



References (1/2)

- Computer Architecture support page
  - http://www.arch.cs.titech.ac.jp/lecture/CA/
- Computer Logic Design support page
  - http://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/

References (2/2)

- Xilinx Vivado Design Suite
  - https://japan.xilinx.com/products/design-tools/vivado.html
- Digilent Arty A7-35 A7: FPGA Trainer Board
  - https://reference.digilentinc.com/reference/programmable-logic/arty-a7/start
- Digilent Nexys 4 DDR Atrix-7 FPGA
  - https://store.digilentinc.com/nexys-4-ddr-artix-7-fpga-trainer-board-recommended-for-ece-curriculum/
- Verilog HDL
  - https://ja.wikipedia.org/wiki/Verilog