Ver. 2024-10-15b

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

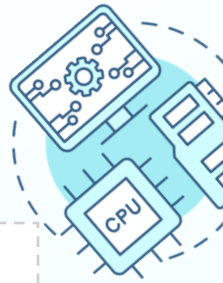
情報工学系　Berjab Nesrine

Computer Architecture support page :

https://www.arch.cs.titech.ac.jp/lecture/CA/

2024年度（令和6年）版

コンピュータアーキテクチャ 演習の注意点 (1)



❑ 連絡について

- 連絡は Slack を使用する。登録がまだの場合は速やかに行うこと。招待メールが来ていない場合は、教員あるいはTAにmアドレスを伝え再送要求すること。

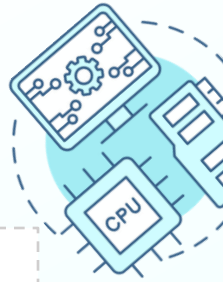
❑ 演習について

- 演習は 15:25～17:05 の時間で行う。15:20 までに学術国際情報センター 3階、情報工学系計算機室に集合すること。15:45 までに到着しない場合、欠席扱いになる。
- 最初の15分は課題の説明、その後は課題の進行とチェックポイントの確認を行う。演習ではACRi ルームを利用する。

❑ グループ作業

- 3人のグループを作成し、グループ内で情報を共有しながら演習を進める。問題が発生した場合、まずグループ内で相談し、それでも解決しない場合は TA や教員に質問すること。

コンピュータアーキテクチャ 演習の注意点 (2)

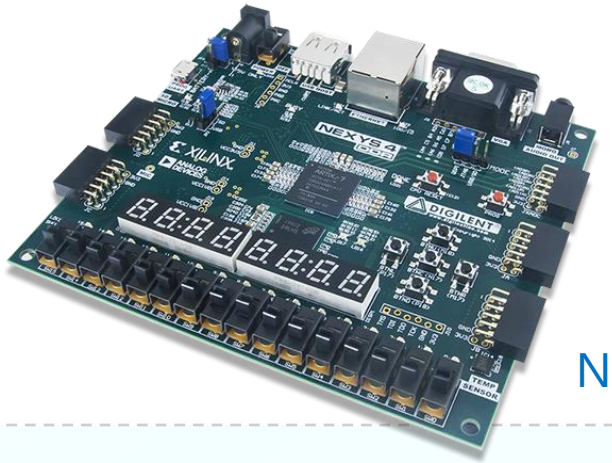


❑ 出席について

- 演習には出席点があるため、全ての授業に休まず参加すること。チェックポイントの図が演習スライドに示されている箇所で、作業の確認を受ける。全てのチェックポイントをクリアすることを目指す。

❑ 演習時間外について

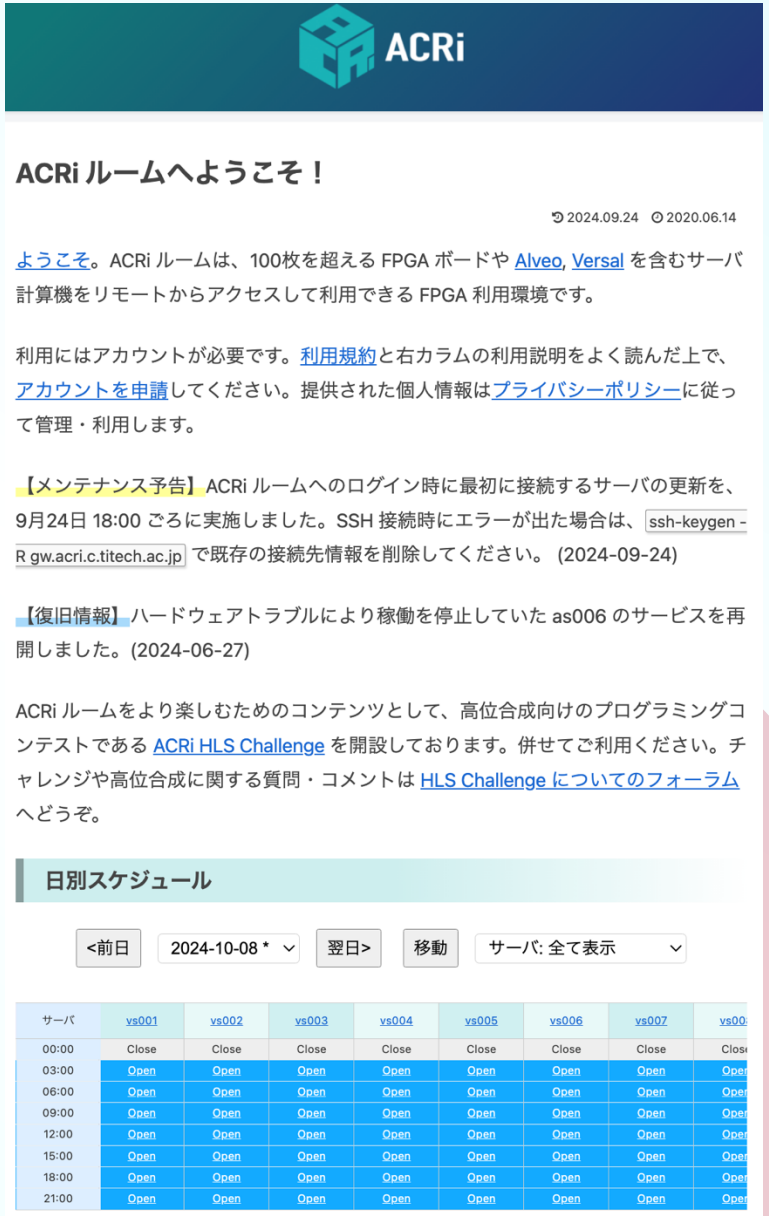
- 演習時間外にも手元の FPGA ボードや ACRi ルームを利用できる。
- 手元のFPGA ボードの貸出も可能なので、独自のハードウェア設計に挑戦してみよう！

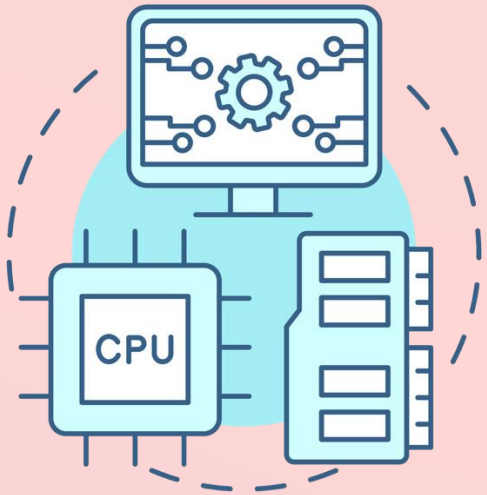


Nexys4 DDR Artix-7 FPGAボード

【重要】 ACRiルームのサーバの予約

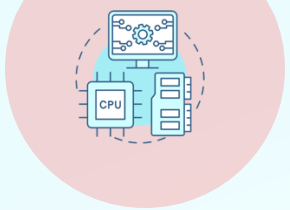
- ACRi ルームのアカウントを使って，次の URL からログインする。
  https://gw.acri.c.titech.ac.jp/wp
- アカウントがなければ、今、次の URL のページを参考にアカウントを申請すること。
  https://gw.acri.c.titech.ac.jp/wp/manual/apply-for-account
  - ACRi ルームでは、次の2種類のアカウントを使用している。
    1. Web 予約システム用アカウント
    2. Linux サーバログイン用アカウント
  - 2つのアカウントで同じ「ユーザアカウント名」を使っているが、別々の「パスワード」が設定されているので注意すること。
- 「予約ページトップ」から，vs0xx~vs7xx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。





Project 3

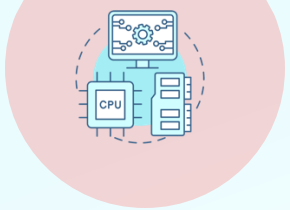
（Part 1）



演習第二回の内容 (Project 3 – Part 1)

❑ 目的:

- この演習の目的は、UART を用いてバイナリデータを FPGA に送信方法を理解すること。これによって、同じ方法で RISC-V の命令を FPGA 上のプロセッサに送ることができるようになる。
- この **Project 3 (part 2)**を通じて以下を学ぶ :
  1. RISC-V 命令のエンコード:
     - add 命令を16進数にエンコードする方法を学ぶ。
  2. UART によるシリアル通信:
     - エンコードした命令を UART を使って FPGA に送信する。
  3. FPGA での Verilog 実装:
     - FPGA 上で受信した命令を処理する Verilog コードを実装する。
  4. VIO を使った結果確認:
     - VIO を使って命令が正しく処理されたかを検証する。



ステップ1: RISC-V命令のエンコード (1/2)

- 最初に、簡単なRISC-V命令を16進数にエンコードする。この命令は後でFPGAに送信する。

- **命令例:**
  - 次のRISC-V命令をエンコードする： `add x12, x1, x2`
    - この命令は、レジスタx1とx2の内容を加算し、その結果をレジスタx12に格納する。
  - エンコードのプロセス:
    - RISC-V命令は特定のエンコードパターンに従う。add 命令の32ビットエンコードは以下のように構成される。

Instr rd, rs1, rs2	funct7	rs2	rs1	funct3	rd	opcode
add x12, x1, x2	0000000	00010 (x2)	00001 (x1)	000 (add)	01100 (x12)	0110011

  - この命令を16進数にエンコードすると、以下のようになる: 00208633

ステップ1: RISC-V命令のエンコード (2/2)

- この命令を Verilog を用いて以下のようにエンコードする：

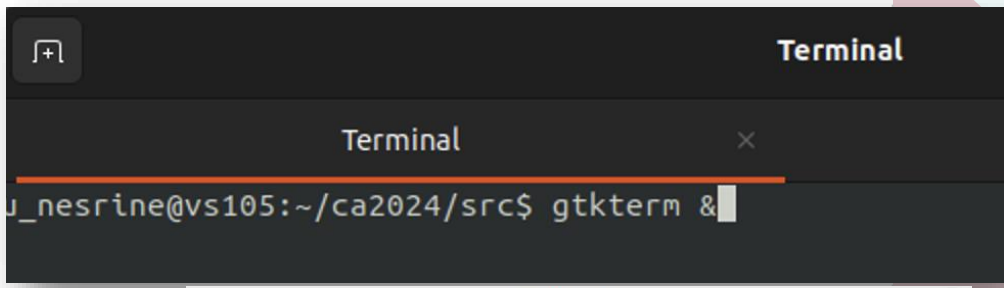
add_inst.v (Source code available in /home/u_nesrine/ca2024/src)

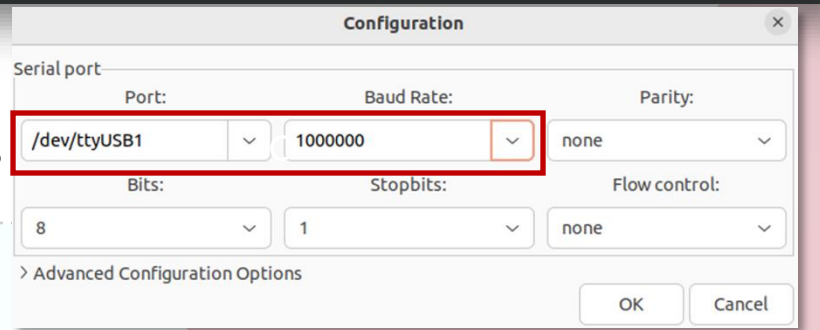
```
module m_top();
   reg [31:0] r_insn = {7'h0, 5'd2, 5'd1, 3'h0, 5'd12, 7'b0110011};  //add x12, x1, x2
   initial begin
      $write("%x\n", r_insn);
      $finish();
   end
endmodule
```

```
$  /tools/cad/bin/verilator --binary add_inst.v
$ obj_dir/Vadd_inst
00208633
- add_inst.v:8: Verilog $finish
- S i m u l a t i o n R e p o r t: Verilator 5.028 2024-08-21
- Verilator: $finish at 1ps; walltime 0.000 s; speed 10.479 ns/s
- Verilator: cpu 0.000 s on 1 threads; alloced 8 MB
```

ステップ2: UARTによるシリアル通信 (1/2)

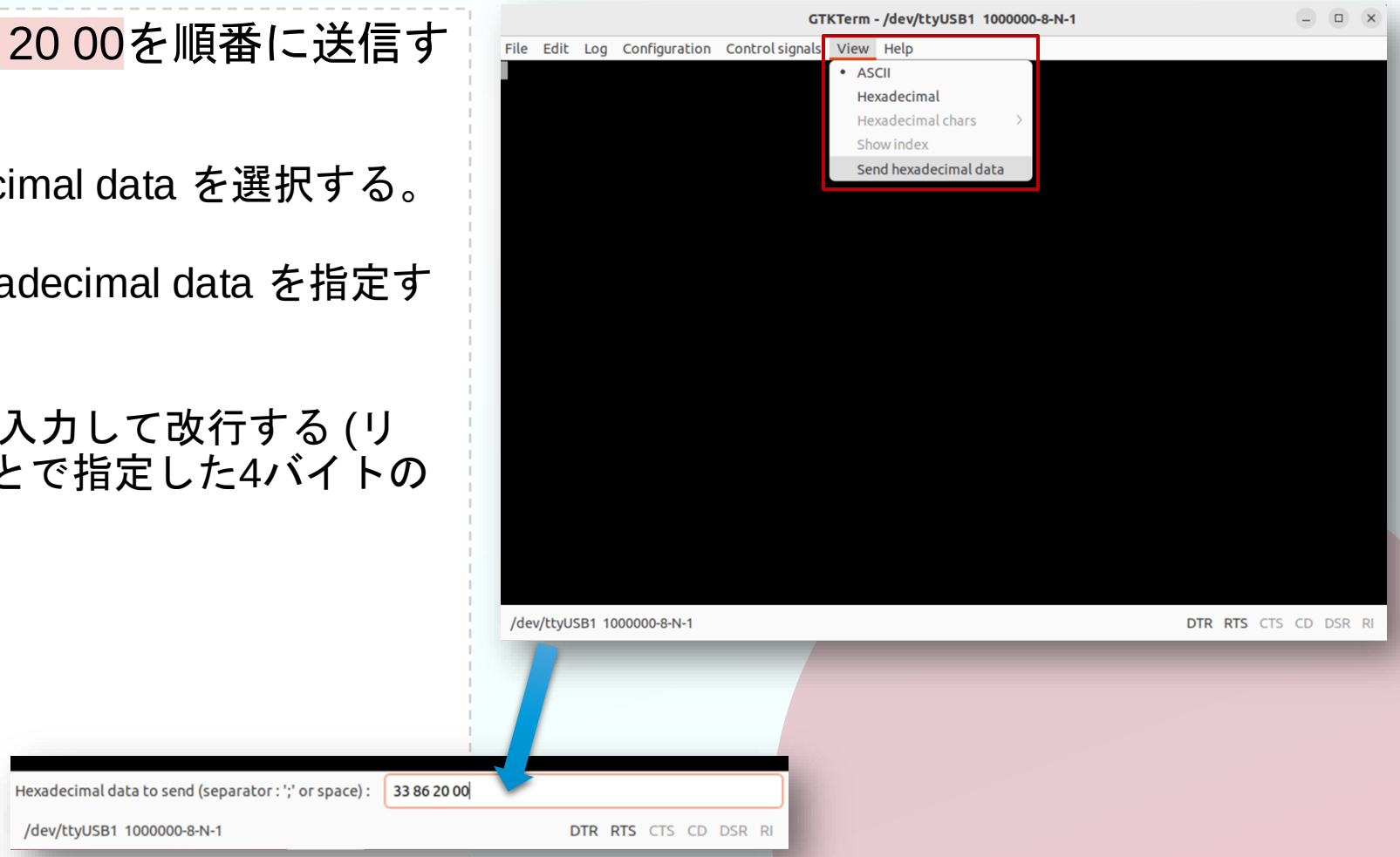
- UART を用いて、エンコードした命令 (00208633) を FPGA に送信する。
- エンディアンについて
  - エンディアンとは、多バイトの数値を送信する際のバイトの並び順のこと。今回使用するリトルエンディアン方式は、最下位のバイトから順に送信する並び順のことである。よって、4バイトで構成される 00208633 は次のように送信する: 33 86 20 00 。
- 送信手順:
  - ターミナルまたはシリアル通信プログラム（例：GtkTerm）を開く。
    - コマンド gtkterm & で GtkTerm を起動する。
    - Configuration から Port を選択する。
      - Port として /dev/ttyUSB1 を選択する。
      - Baud Rate に 1000000 を入力して、1Mbaud とする。





ステップ2: UARTによるシリアル通信 (2/2)

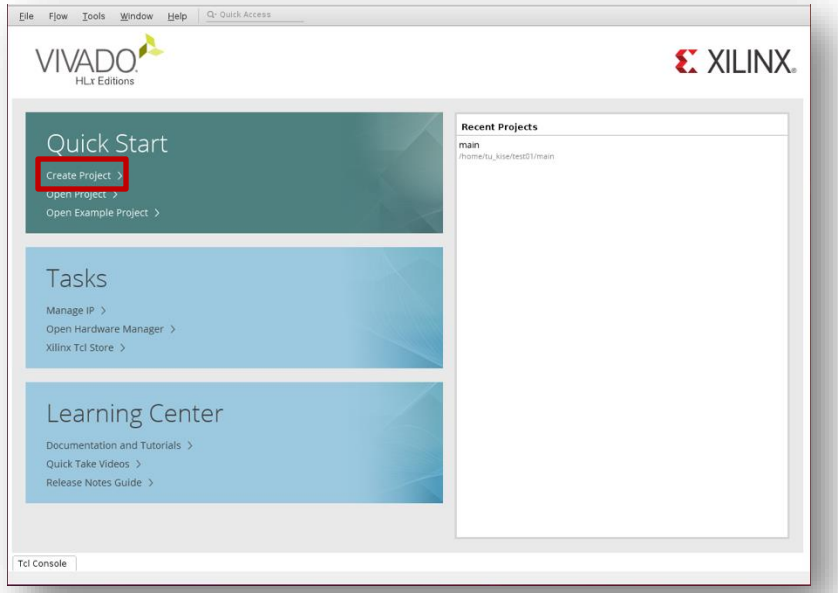
- 16進数のバイト列33 86 20 00を順番に送信する。
  - View → Send hexadecimal data を選択する。
  - View の下に send hexadecimal data を指定する欄が表示される。
  - ここに 33 86 20 00 と入力して改行する (リターンキーを押す) ことで指定した4バイトのデータを送信できる。



ステップ3: UART 受信のための Verilog 実装 (1/4)

- FPGA 上でシリアルデータを受信し、レジスタに格納するモジュールを Verilog を用いて実装する。
- シミュレーションを実行し、受信したデータが正しく 00208633 として組み立てられることを確認する。
- 新しいVivado Project 「project_3」 を作る。
- ターミナルで次のコマンドを入力し，Vivado を起動する.
  - 「Vivado 2024.1」を利用する。

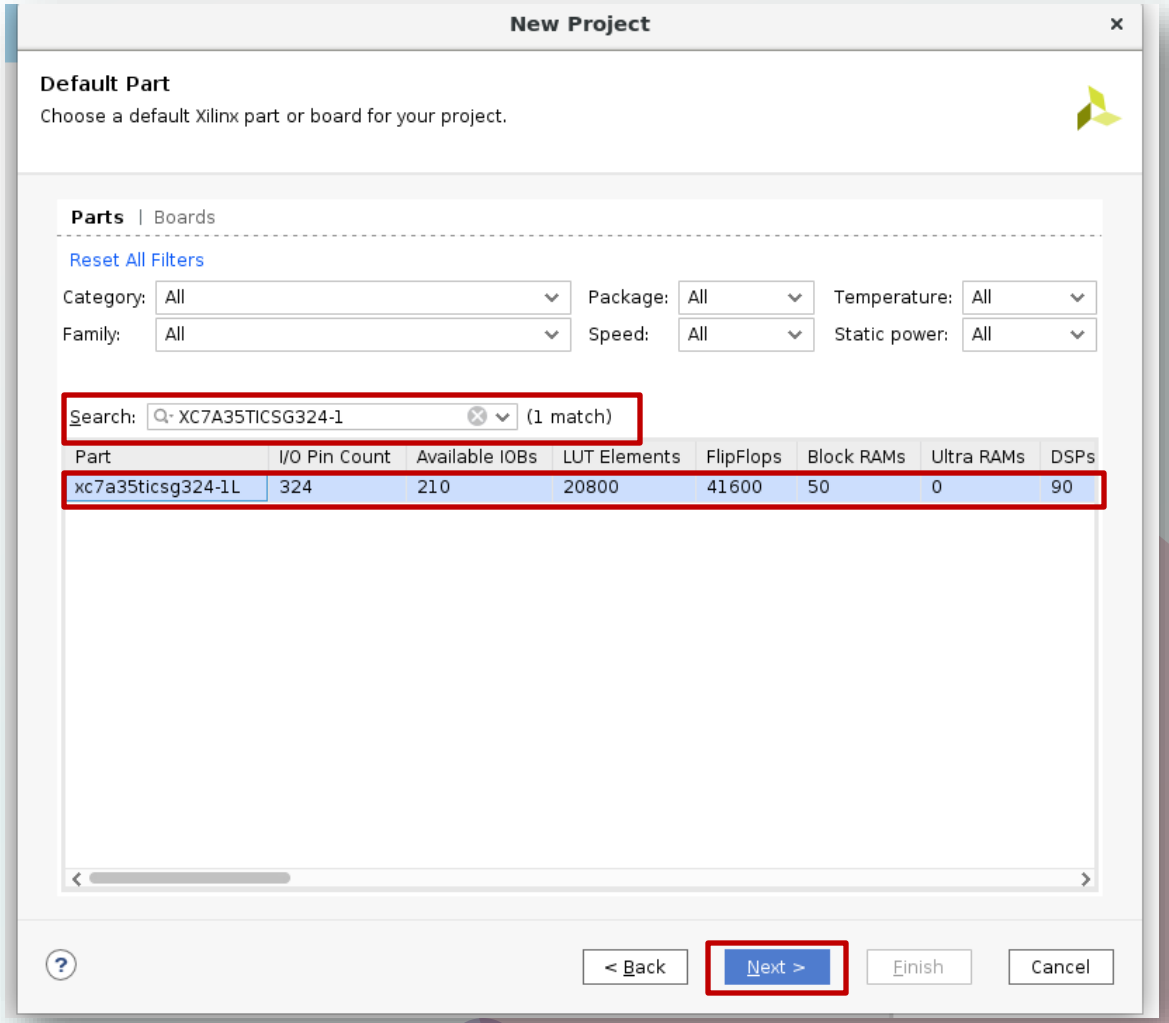
```
$ source /tools/Xilinx/Vivado/2024.1/settings64.sh
$ vivado &
```



- Select Create Project, Click Next
- Project name “project_3” and location “/home/your_username/ca2024” are selected.
  - Check “Create project subdirectory”.
- Click Next

ステップ3: UART 受信のための Verilog 実装 (2/4)

- In Project Type window, select RTL project and click Next.
- In Add Sources window, click Next.
- In Add Constraints (optional) window, click Next.
- In Default Part window, select Parts, and write XC7A35TICSG324-1L.
- Select XC7A35TICSG324-1L and click Next.
- Confirm the summary in New Project Summary window and click Finish.



ステップ3: UART 受信のための Verilog 実装 (3/4)

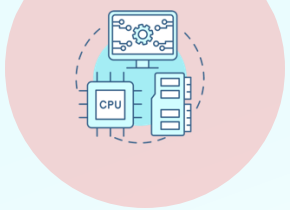
- Source codeをコピーする
  - ターミナルで，ファイルをコピーする。
  - /home/u_nesrine/ca2024/src/ に保存されている main3.v と main3.xdc を，作成したプロジェクトのディレクトリ ~/ca2024/project_3 にコピーする。

```
$ cd ~/ca2024/project_3
$ cp /home/u_nesrine/ca2024/src/main3.v .
$ cp /home/u_nesrine/ca2024/src/main3.xdc .
```

- m_uart_rx: このモジュールは、UARTで1バイトずつデータを受信する。
- r_data: 32ビットのレジスタで、受信した命令を保持する。各バイトが到着するたびにシフトしてデータを格納する。
  - VIOを m_main モジュール内の r_data に接続する。

main3.v (Source code available in /home/u_nesrine/ca2024/src)

```
module m_main (
    input  wire w_clk       , // 100 MHz clock signal
    input  wire w_rxd
);

    wire       w_en      ;
    wire [7:0] w_dout    ;
    m_uart_rx m_uart_rx0 (w_clk, w_rxd, w_dout, w_en);

    reg [31:0] r_data = 0;
    always @(posedge w_clk) if (w_en) r_data <= {w_dout, r_data[31:8]};

    vio_0 vio_00 (w_clk, r_data);

endmodule
```



ステップ3: UART 受信のための Verilog 実装 (4/4)

- Click Add Sources, then select Add or create design sources and click Next.
  - In Add or Create Design Sources window, click Add Files, select main3.v in project_3 directory, and click OK.
  - Click Finish.

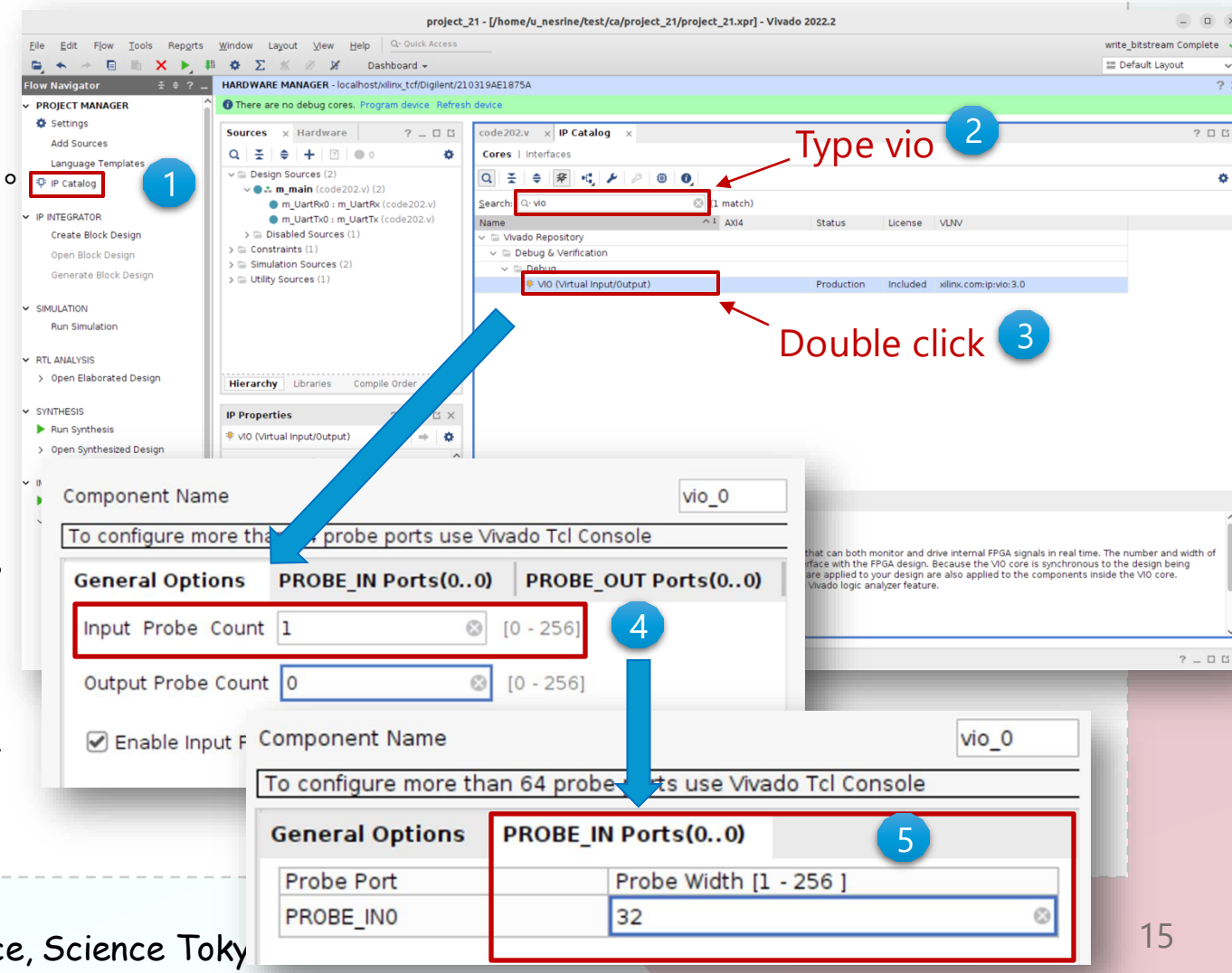
- Click Add Sources, then select Add or create constraints and click Next.
  - In Add or Create Constraints window, click Add Files, select main3.xdc in project_3 directory, and click OK.
  - Click Finish.

ステップ4: VIO（仮想入出力）を使った確認 (1/2)

- 最後に、r_data の値を確認するために VIO (Virtual Input/Output)を使用し、受信した命令が正しく処理されたかを確認する。

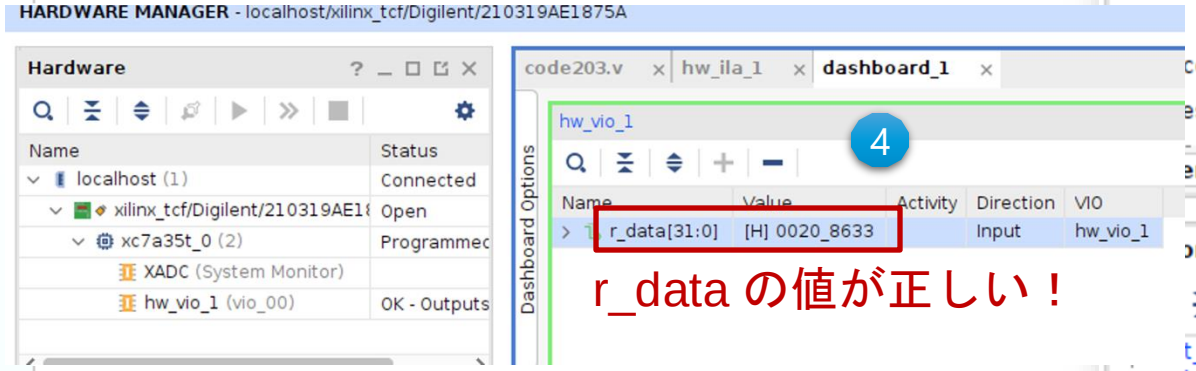
- VIO のコンフィギュレーション：
  1. Vivadoで IP Catalog を開き、vio を検索する。
  2. 以下のようにVIOを設定する：
     - Input Probe Count: 1
     - Probe Width: 32
  3. Click Generate and click OK if asked in Generate Output Products window.

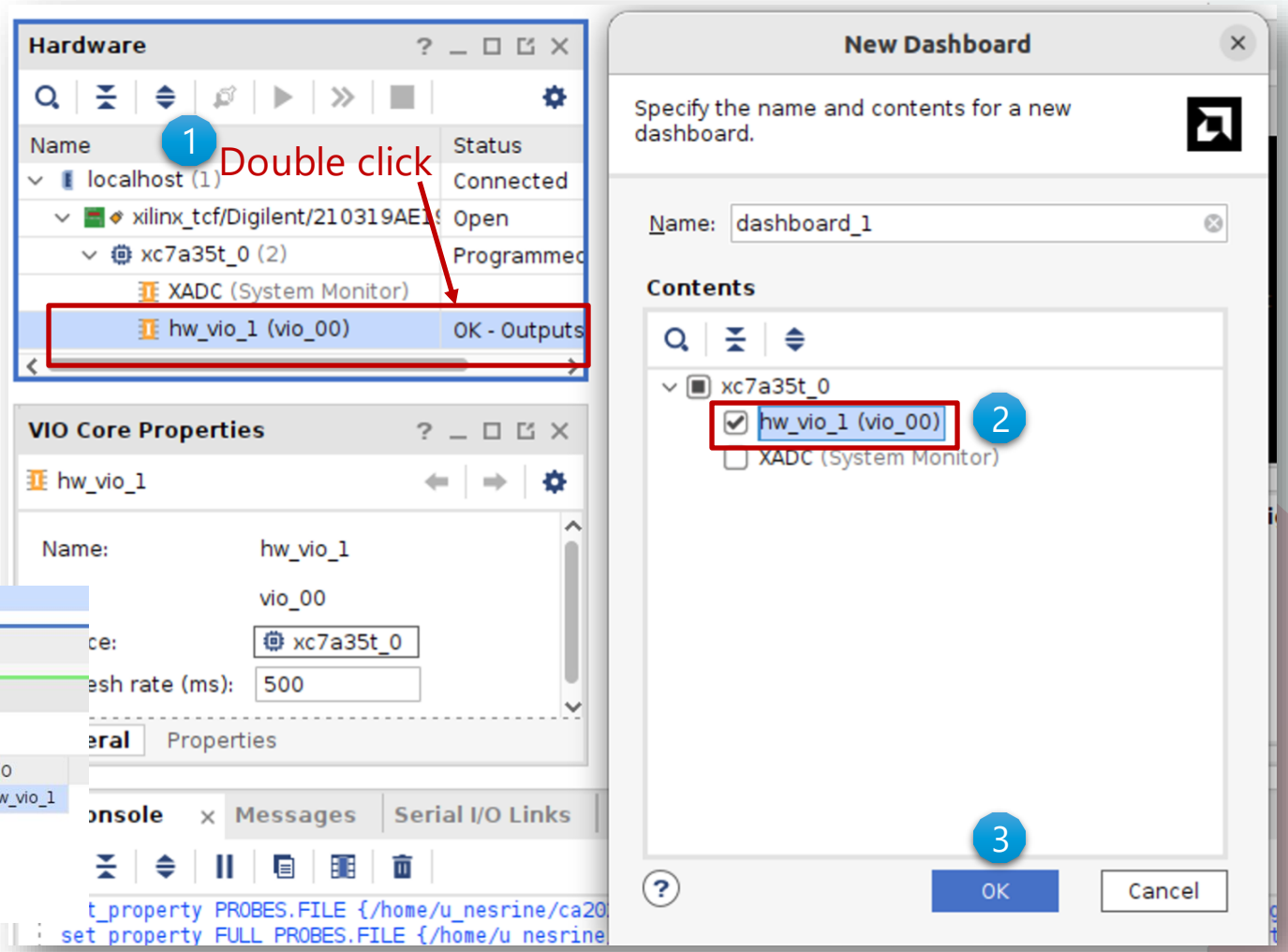
- ビットストリームを生成し、FPGAにプログラムする。
- UART を用いてバイナリデータを FPGA に送信する (スライド10)。

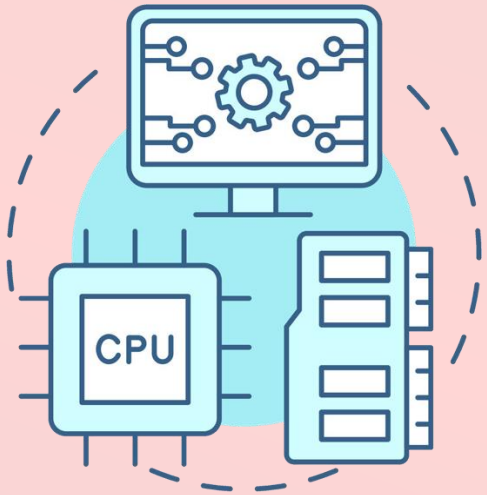


ステップ4: VIO（仮想入出力）を使った確認 (2/2)

- UART を用いてバイナリデータを FPGA に送信する。
- Hardware Manager を開き、VIO コアを追加する。
- VIOを使用して r_data[31:0] の値が正しいかを確認する。
  - Click +button in hw_vio_1 window, select r_data[31:0] and the click OK

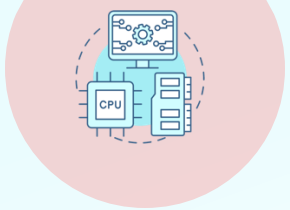






Project 3

（Part 2）



演習第二回の内容 (Project 3 – Part 2)

❑ 目的:

- この演習の目的は、UART を用いてバイナリデータを FPGA に送信方法を理解すること。
- この **Project 3 (part 2)** を通じて以下を学ぶ :
  1. RISC-V 命令のエンコード:
     - **RISC-Vアセンブリコードのエンコードする方法を学ぶ。**
  2. FPGA での Verilog 実装:
     - **受信した命令を受け取るために Verilog コード (main3.v)を修正する。**
  3. UART によるシリアル通信:
     - **エンコードしたRISC-Vアセンブリコードを UART を使って FPGA に送信する。**
  4. VIO を使った結果確認:
     - VIO を使って命令が正しく処理されたかを検証する。

ステップ1: RISC-Vアセンブリコードの作成 (1/5)

- RISC-Vアセンブリコードの作成 例として、Fibonacci のアセンブリコードを作成する。
  - 例として、Fibonacciのアセンブリコードを使っているが、別のプログラムを作成することできる。
- Venus （オンラインシミュレーター）を使用:
  - ブラウザで下の URL にアクセスし、Venusのウェブバージョンを開く。

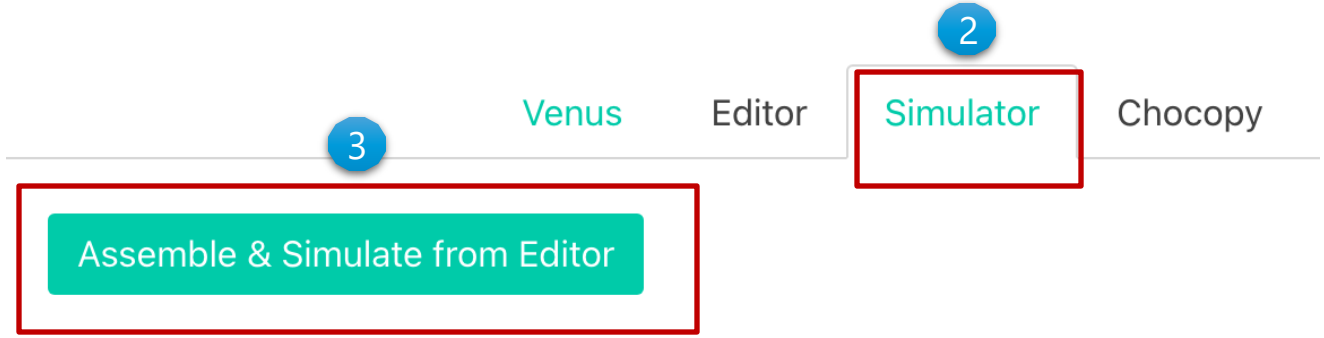
https://venus.cs61c.org

fib.s

```
    .data
n:  .word 10              # The Fibonacci sequence will be calculated up to this number (10)

    .text
    .globl main
main:
    add t0, zero, zero    # t0 = 0 (Fibonacci(0))
    addi t1, zero, 1      # t1 = 1 (Fibonacci(1))
    la t3, n              # Load the address of n into t3
    lw t3, 0(t3)          # Load the value of n (number of iterations) into t3

fib:
    beq t3, zero, finish  # If t3 (counter) == 0, jump to finish
    add t2, t1, t0        # t2 = t1 + t0 (Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2))
    mv t0, t1             # Move t1 to t0 (t0 = Fibonacci(i-1))
    mv t1, t2             # Move t2 to t1 (t1 = Fibonacci(i))
    addi t3, t3, -1       # Decrement the counter (t3 = t3 - 1)
    j fib                 # Jump back to fib to continue the loop

finish:
    addi a0, zero, 1      # a0 = 1 (for printing or exit, depending on the syscall convention)
    mv a1, t0             # Move the final Fibonacci result into a1 (Fibonacci(n))
    ecall                 # Make system call (print or exit depending on environment)

    addi a0, zero, 10     # a0 = 10 (Exit syscall)
    ecall                 # Exit program
```

ステップ1: RISC-Vアセンブリコードの作成 (2/5)

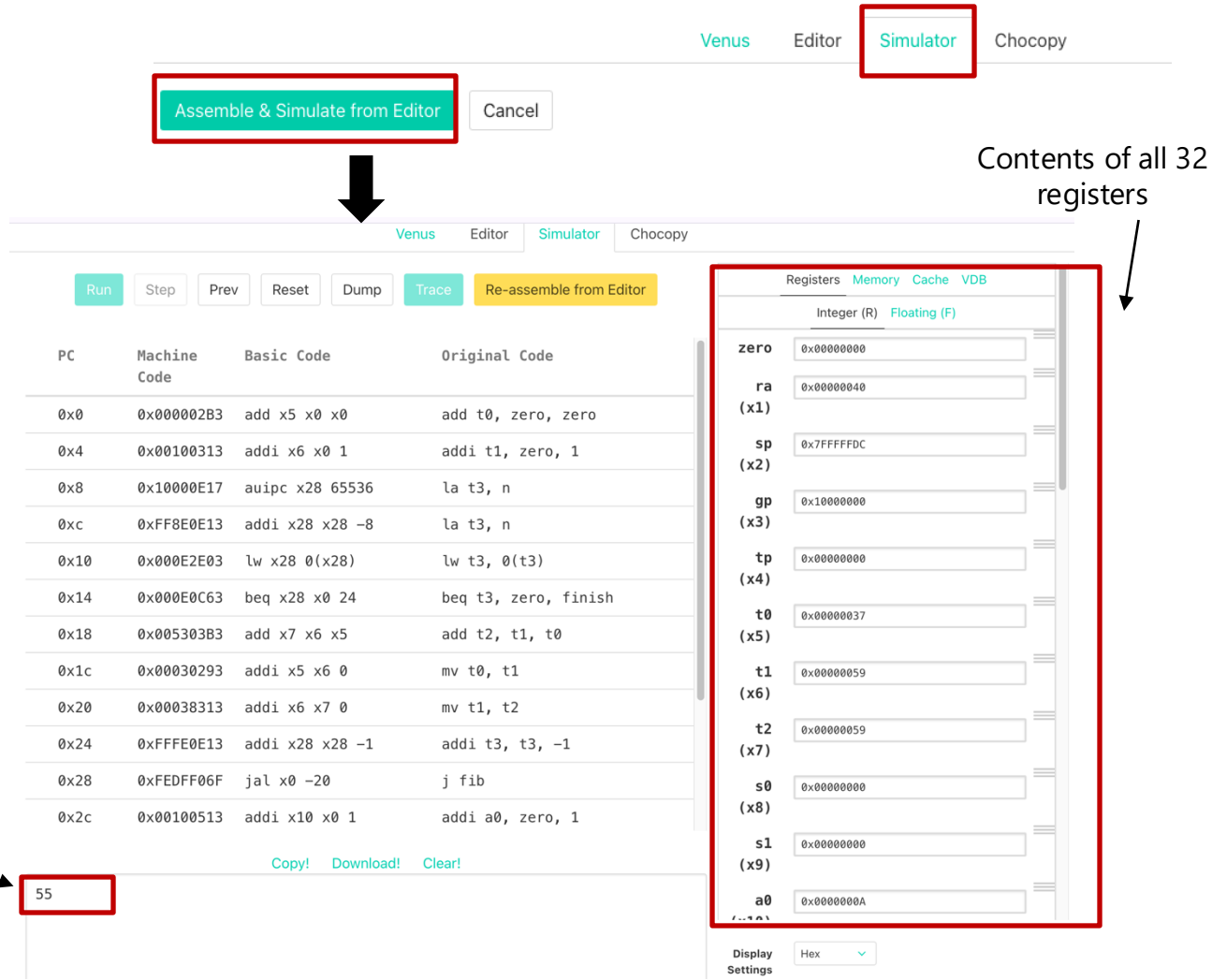
- 左側のタブ Editor を使って、RISC-V アセンブリコードを書く。
- 次はタブ Simulator を使って、 Assemble & Simulate ボタンをクリックして、シミュレーターでコードを実行する。




```
1     .data
2  n: .word 9
3
4     .text
5     .globl main
6  main:
7     add t0, zero, zero
8     addi t1, zero, 1
9     la t3, n
10    lw t3, 0(t3)
11
12 fib:
13    beq t3, zero, finish
14    add t2, t1, t0
15    mv t0, t1
16    mv t1, t2
17    addi t3, t3, -1
18    j fib
19
20 finish:
21    addi a0, zero, 1
22    mv a1, t0
23    ecall
24
25    addi a0, zero, 10
26    ecall
```

ステップ1: RISC-Vアセンブリコードの作成 (3/5)

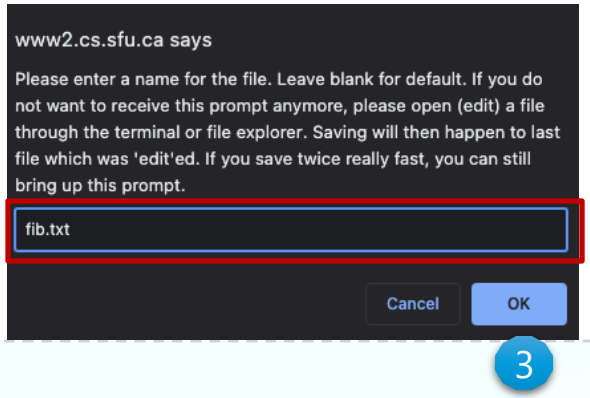
- デバッグと出力の確認:
  - コードをステップ実行したり、ブレークポイントを設定したり、実行中のレジスタやメモリの状態を確認することができる。

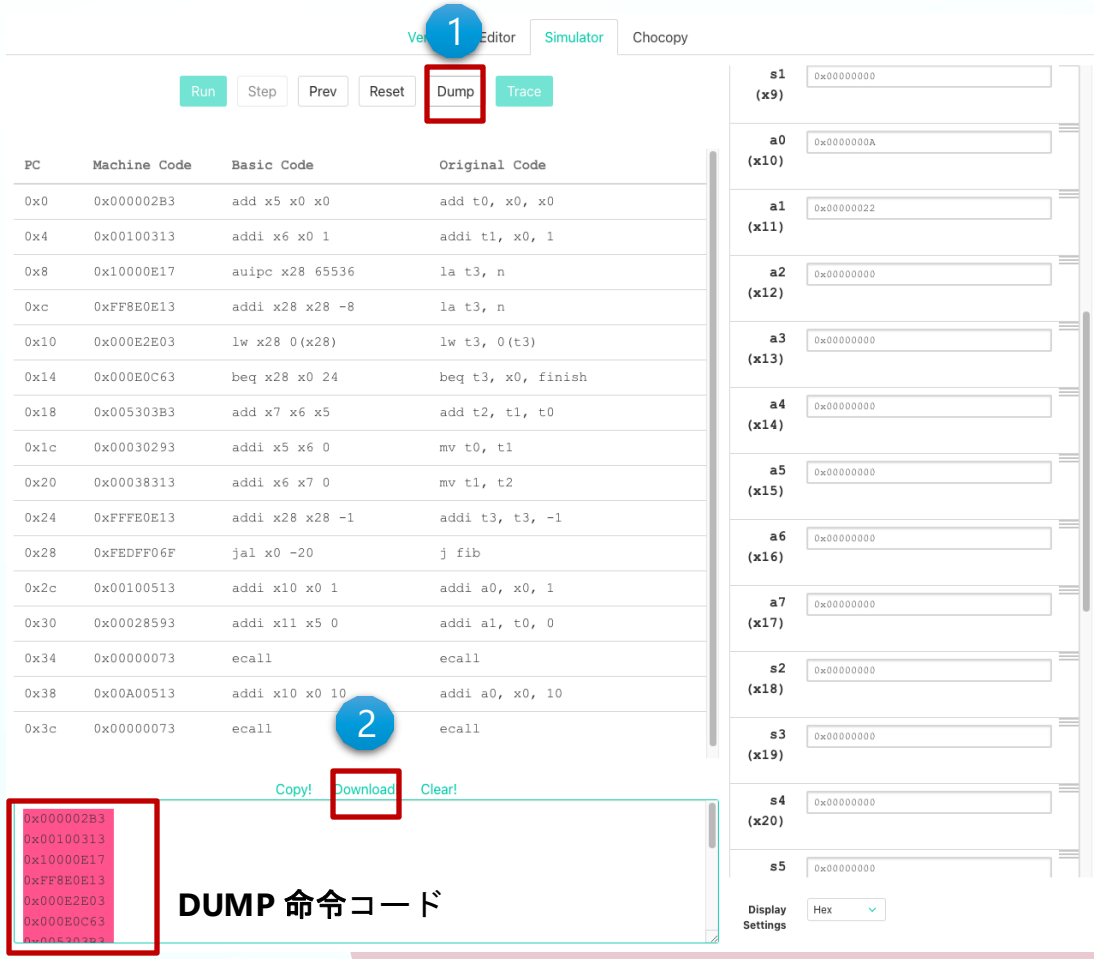


Console output
For Fibonacci(10), the value is **55**.

ステップ1: RISC-Vアセンブリコードの作成 (4/5)

- Dumpタブを使って、コード中のすべての命令の16進数表現を生成します。
- その後、Download を押すと、ファイルがダウンロードされる。
  - ファイル名（例: fib.txt）を入力し保存する





ステップ1: RISC-Vアセンブリコードの作成 (5/5)

(Source code available in /home/u_nesrine/ca2024/src)

- このコード (main3_2.c) は、fib.txt から16進数データを読み取り、バイナリに変換して test.binに書き込み、全データのチェックサムを計算するプログラムです。

- 変換後のファイルのチェックサムを確認
  - 次のコマンドを実行する

```
$ gcc main3_2.c
$ ./a.out
checksum fa474d0
```

- Fibonocci のマシンコードのチェックサムの値は 0xfa474d0 となる。

main3_2.c

```c
#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); // Close the file
    fclose(fp); // Close the file

    return 0;
}
```

ステップ2: UART 受信のための Verilog 実装

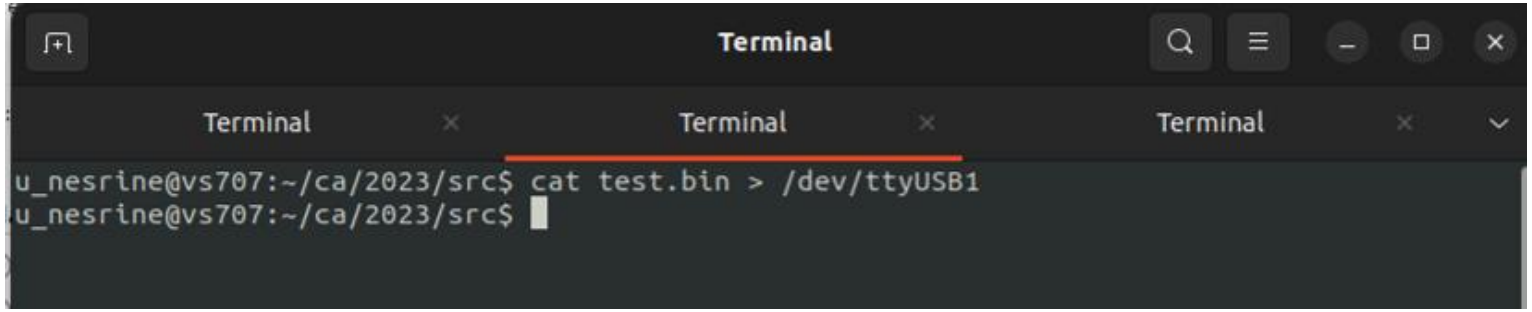
- m_main モジュールを完成させ、受信データからチェックサム r_sum を計算できるようにし、その後、VIO に接続する。

main3.v (Source code available in /home/u_nesrine/ca2024/src)

```
module m_main (
    input  wire w_clk       , // 100 MHz clock signal
    input  wire w_rxd
);

    wire       w_en      ;
    wire [7:0] w_dout    ;
    m_uart_rx m_uart_rx0 (w_clk, w_rxd, w_dout, w_en);

    reg [31:0] r_data = 0;
    always @(posedge w_clk) if (w_en) r_data <= {w_dout, r_data[31:8]};

    vio_0 vio_00 (w_clk, r_data);

    reg [31:0] r_sum = 0;

    // Complete here to calculate the checksum r_sum
    // Connect VIO to r_sum to verify the checksum value


endmodule
```

ステップ3: UARTによるシリアル通信

- ターミナルを開いて、次のコマンドを実行して test.bin をFPGAに送信する。
  - 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。
  - cat test.bin > /dev/ttyUSB1



ステップ4: VIO（仮想入出力）を使った確認

- 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 を用いて確認する。