Ver. 2024-10-29

コンピュータアーキテクチャ 演習 (4)
Computer Architecture Exercise (4)

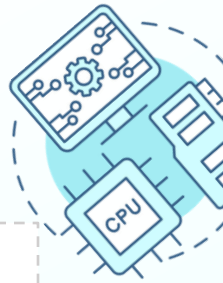
情報工学系　Berjab Nesrine

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。

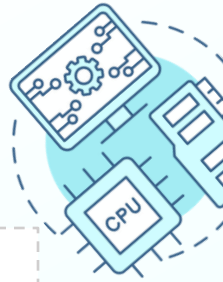
❑ 演習について

- 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)

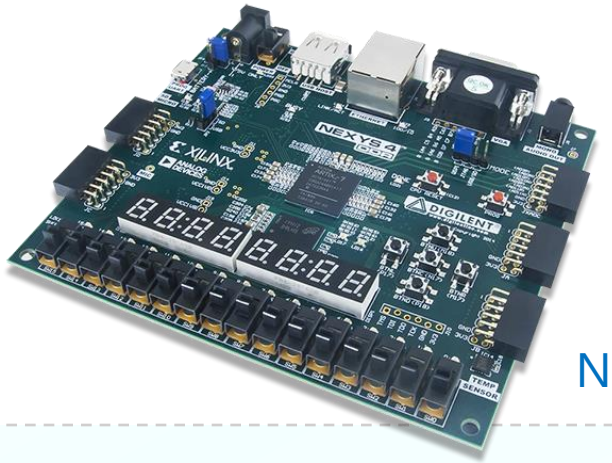


❑ 出席について

- 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

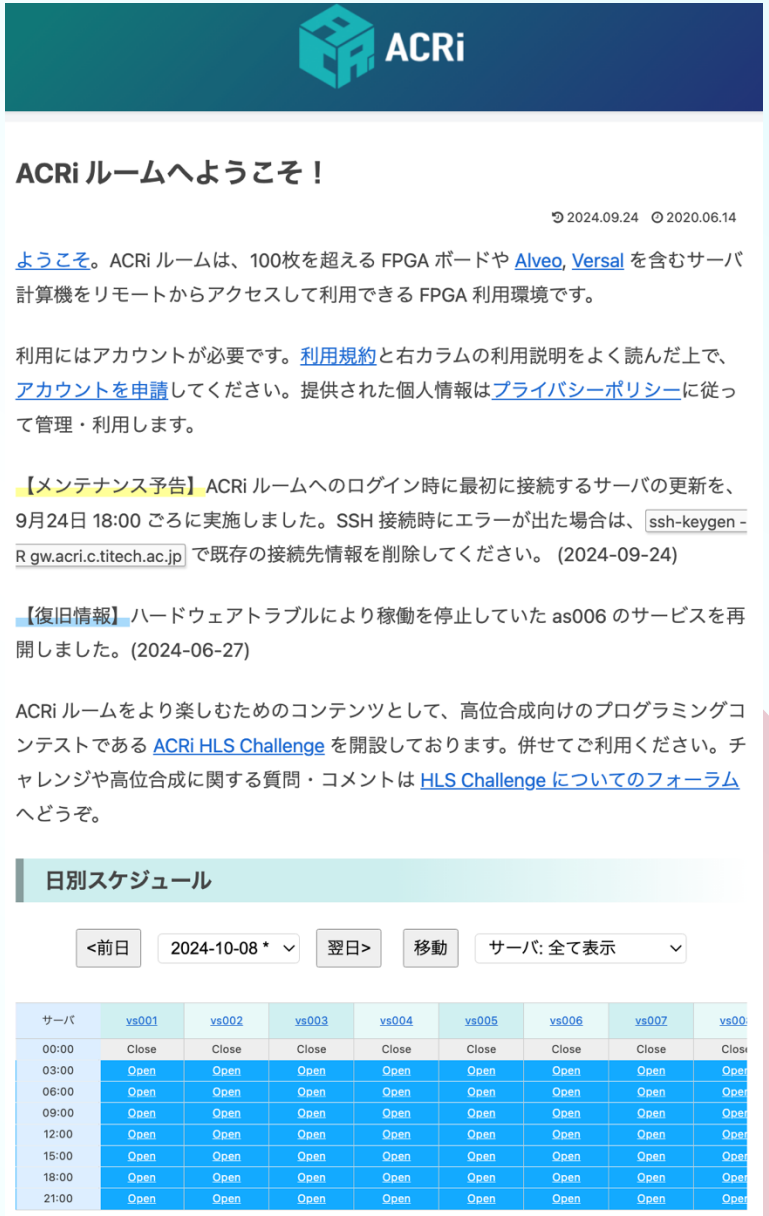
- 演習時間外にも手元の FPGA ボードや ACRi ルームを利用できる。
- 手元のFPGA ボードの貸出も可能なので、独自のハードウェア設計に挑戦してみよう！

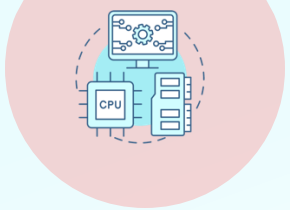


Nexys4 DDR Artix-7 FPGAボード

【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。

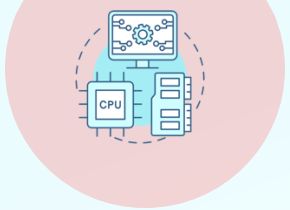




演習第五回の内容

❑目的:

- **Project 5:**
  - 32エントリ、ブロックサイズ1ワードのキャッシュのエントリ数を増やして、64エントリに修正する。
- **Project 6:**
  - MIG (Memory Interface Generator) を使って DRAM を動かし、読み出しと書き込みにかかる最大のサイクル数を求める。



演習第五回の内容 (Project 5)

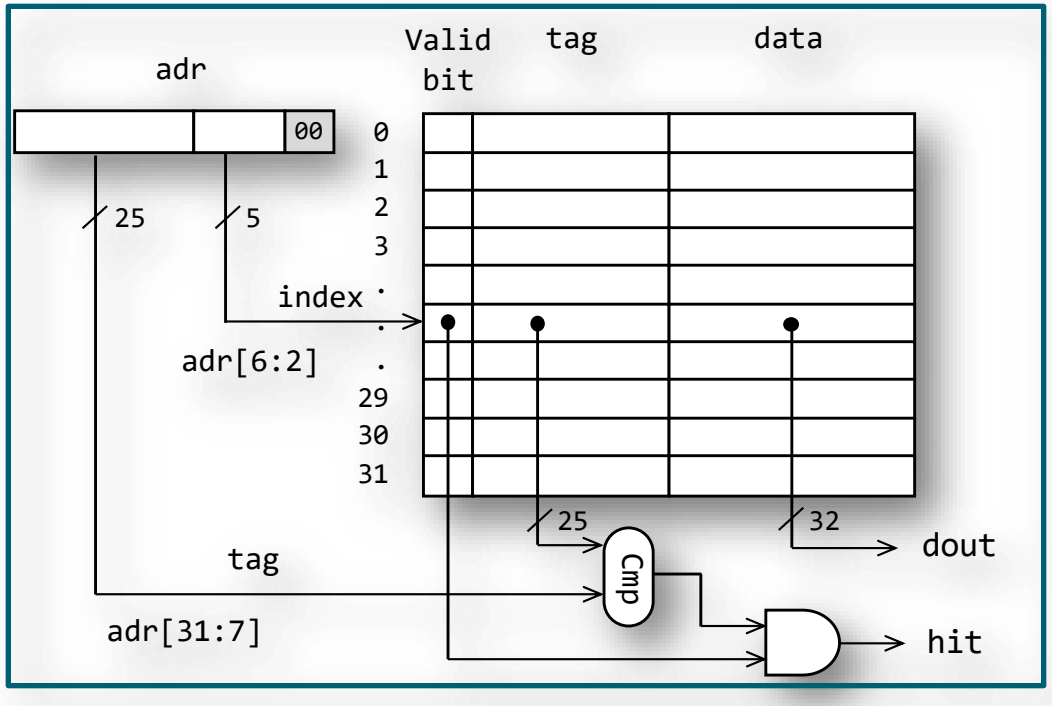
❑目的:

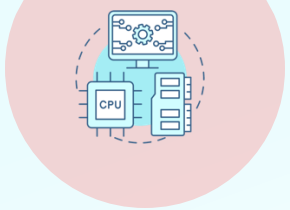
- ダイレクトマップ方式のキャッシュがどのように動作し、タグやインデックスを使ってデータが保存・アクセスされるかを理解する。
- キャッシュメモリのサイズやインデックスの幅を変更して、キャッシュエントリ数を調整する方法を理解する。
- キャッシュエントリ数を増やすことでキャッシュヒット・ミス率にどのような影響があるか、またシステム全体の性能にどう関わるかを確認する。

ダイレクトマップ方式のキャッシュの基本

❑ダイレクトマップ方式のキャッシュの仕組み:

- キャッシュラインの位置はインデックスによって一意に定まる（直接写像される)。
- 主要な要素:
  - **index**: キャッシュ内の特定のラインを選択する。
  - **tag**: 正しいメモリブロックがそのラインに格納されているか識別するために使用される。
  - **valid bit**: そのキャッシュラインのデータが有効かどうかを示す。
  - **data**: 実際に格納されているデータ
    - この場合、32ビットのワード





現在のキャッシュコード (1/2)

1) キャッシュは32エントリの配列を使って実装されており、それぞれにバリッドビット、タグ、データを格納している。

2) 現在のコードでは、adr[6:2] の 5ビットをインデックスとして、キャッシュにアクセスする。

3) アドレスタグ（adr[31:7]）は格納されたタグと比較され、ヒットかミスかを判断する。

proc2.v

```
module m_cache_direct_mapped (
  input  wire         w_clk,
  input  wire         w_we,
  input  wire [31:0] w_adr,
  input  wire [31:0] w_wadr,
  input  wire [57:0] w_wd,
  output wire         w_hit,
  output wire [31:0] w_dout
);

  reg [57:0] mem [0:31];  1
  integer i; initial for (i=0; i<32; i=i+1) mem[i] = 0;

  wire [4:0]  w_index = w_adr[6:2];  2
  wire        w_v;
  wire [24:0] w_tag;  3
  assign {w_v, w_tag, w_dout} = mem[w_index];
  assign w_hit = w_v & (w_adr[31:7]==w_tag);  3

  always @(posedge w_clk) if (w_we) mem[w_index] <= w_wd;
endmodule
```

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

現在のキャッシュコード (2/2)

❑動作確認:

- proc2.v、 sample2.txt を同じディレクトリーにコピーする。
- シミュレーションにより、動作確認を行う。

```
$ cd ~/ca2024/
$ cp /home/u_nesrine/ca2024/src/proc2.v .
$ cp /home/u_nesrine/ca2024/src/sample2.txt .
$ /tools/cad/bin/verilator --binary -o simv proc2.v
$./obj_dir/simv
```

- シミュレーションを実行すると以下の結果が出力される。

```
simulation finished.
cache hit            11643
cache miss           13774
total (hit+miss)     25417
- proc_ex4.v:410: 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 3us; walltime 0.195 s; speed 21.203 us/s
- Verilator: cpu 0.120 s on 1 threads; alloced 9 MB
```

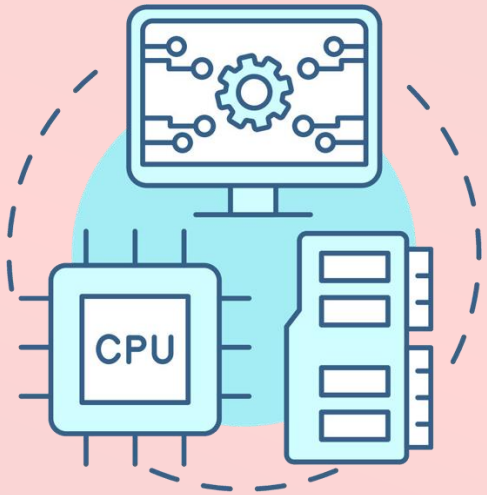
キャッシュサイズを64エントリにする

❑ 目標：

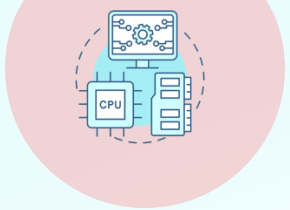
- 32エントリと1ワードのブロックサイズを持つダイレクトマップ方式のキャッシュが与えられている。
- proc2.v の m_cache_direct_mapped モジュールを変更し、エントリ数を32から64に増やすこと 。
- 変更が完了したら、シミュレーションを実行し、出力が以下と一致するか確認する。

```
simulation finished.
cache hit            21559
cache miss            3858
total (hit+miss)     25417
- proc_ex4.v:411: 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 3us; walltime 0.129 s; speed 23.262 us/s
- Verilator: cpu 0.109 s on 1 threads; alloced 9 MB
```

Check Point 6



Project 6



演習第五回の内容 (Project 6)

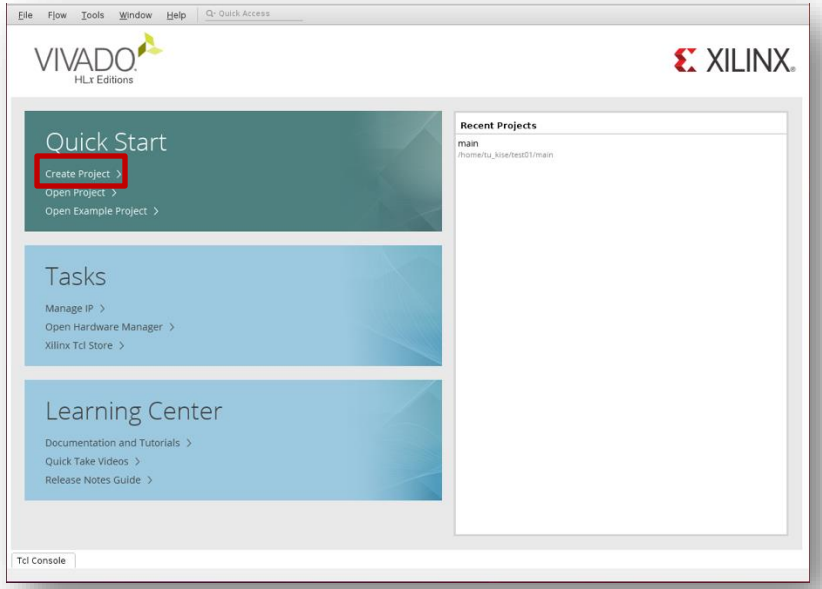
❑課題:

- DRAM の読み出しおよび書き込みにかかる最大のサイクル数を求める。
  - **目的**： DRAM の読み出しおよび書き込みにどのくらいのサイクル数がかかるかを知る。そして、DRAM のアクセスが遅いことを理解する。
  - **方法**： DRAM の読み出し（リード）および書き込み（ライト）を繰り返し、サイクル数を測定する。読み出しおよび書き込みのそれぞれについて、「測定したサイクル数」がそれまでの「最大のサイクル数」を上回った場合、「最大のサイクル数」を更新する。
  - **出力**：VIO で読み出しおよび書き込みにかかった最大のサイクル数を出力する。

新しい Vivado Project を作る (1/2)

- 新しいVivado Project 「project_6」 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_6” and location “/home/your_username/ca2024” are selected.
  - Check “Create project subdirectory”.
- Click Next
- In Default Part window, select Parts, and write XC7A35TICSG324-1L.

ステップ1:新しい Vivado Project を作る (2/2)

```
$ cd ~/ca2024/project_6
$ cp /home/u_nesrine/ca2024/src/main6.v .
$ cp /home/u_nesrine/ca2024/src/main6.xdc .
$ cp /home/u_nesrine/ca2024/src/arty_a7_mig.ucf .
```

- 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 main6.v 、 main6.xdcと arty_a7_mig.ucf を作成したプロジェクトのディレクトリ ~/ca2024/project_6 にコピーする。
- Click Add Sources, then select Add or create design sources and click Next.
  - In Add or Create Design Sources window, click Add Files, select main6.v in project_6 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 main6.xdc in project_6 directory, and click OK.
  - Click Finish.

main6.v コードの説明 (1/3)

- main6.v のコードは、DDR3 メモリに対する読み書き処理を行うモジュール。
  - DDR3 メモリとのインターフェースには、Xilinx のメモリインターフェース生成（MIG）IP コアを使用している。
  - MIG コアに対して、書き込みコマンドと読み出しコマンドを送ることによって、メモリにデータを書き込んだり読み出したりする処理が実装されている。
  - また、メモリ初期化完了などの状態をLEDに表示する。
- 入出力ポート:
  - w_clk: 100MHz のクロック入力。
  - w_led: 4ビットの LED 出力で、動作状態を表示する。
  - ddr3_*の各信号は DDR3 メモリとの接続用。

main6.v

```
module m_main (
    input  wire w_clk,                          // 100MHz clock signal
    output wire [3:0] w_led,                    // LED
    //rest of code
);
```

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

main6.v コードの説明 (2/3)

main6.v

```
//code
reg  [`APP_ADDR_WIDTH-1 : 0] r_app_addr = 0;
reg  [`APP_CMD__WIDTH-1 : 0] r_app_cmd  = 0;
//rest of code
```

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

❑ レジスタと信号:

- ❑ いくつかのレジスタやワイヤが宣言されており、主に DDR3 メモリコントローラ（mig_7series_0）とのインターフェース用に使用される。
- ❑ r_app_addr と r_app_cmd はメモリ操作のためのアドレスとコマンドを指定するためのも。
- ❑ r_app_en はコマンドが有効であること、r_app_wdf_wren は書き込みが有効であることを示す。
- ❑ r_app_wdf_data にはメモリへの書き込みデータである。
- ❑ r_app_wdf_mask は書き込みデータがバイトごとに無効であるか否かを表す (0: 有効、1: 無効)。
- ❑ app_rd_data はメモリからの読み出しデータである。

main6.v コードの説明 (3/3)

❑ メモリ操作用のステートマシン

- r_state が書き込みと読み取り処理を管理するステートマシン。
- init_calib_complete が1になったときに、DRAM の初期化が完了する。
- **書き込みプロセス**:
  - ステート0で、メモリが準備完了 (app_rdy) で書き込みデータが受け入れ可能 (app_wdf_rdy) の場合、r_app_en と r_app_wdf_wren を1に設定し書き込み開始。
  - 開始後、 ステート1に進み、書き込み完了と r_app_addr のインクリメントを実行。
- **読み出しプロセス:**
  - ステート3でメモリが準備完了の場合、r_app_cmd を CMD_READ に設定して読み取りを開始。
  - ステート4で、app_rd_data_valid が有効なときデータを読み出し、r_sum に加算し、アドレスをインクリメント。

main6.v

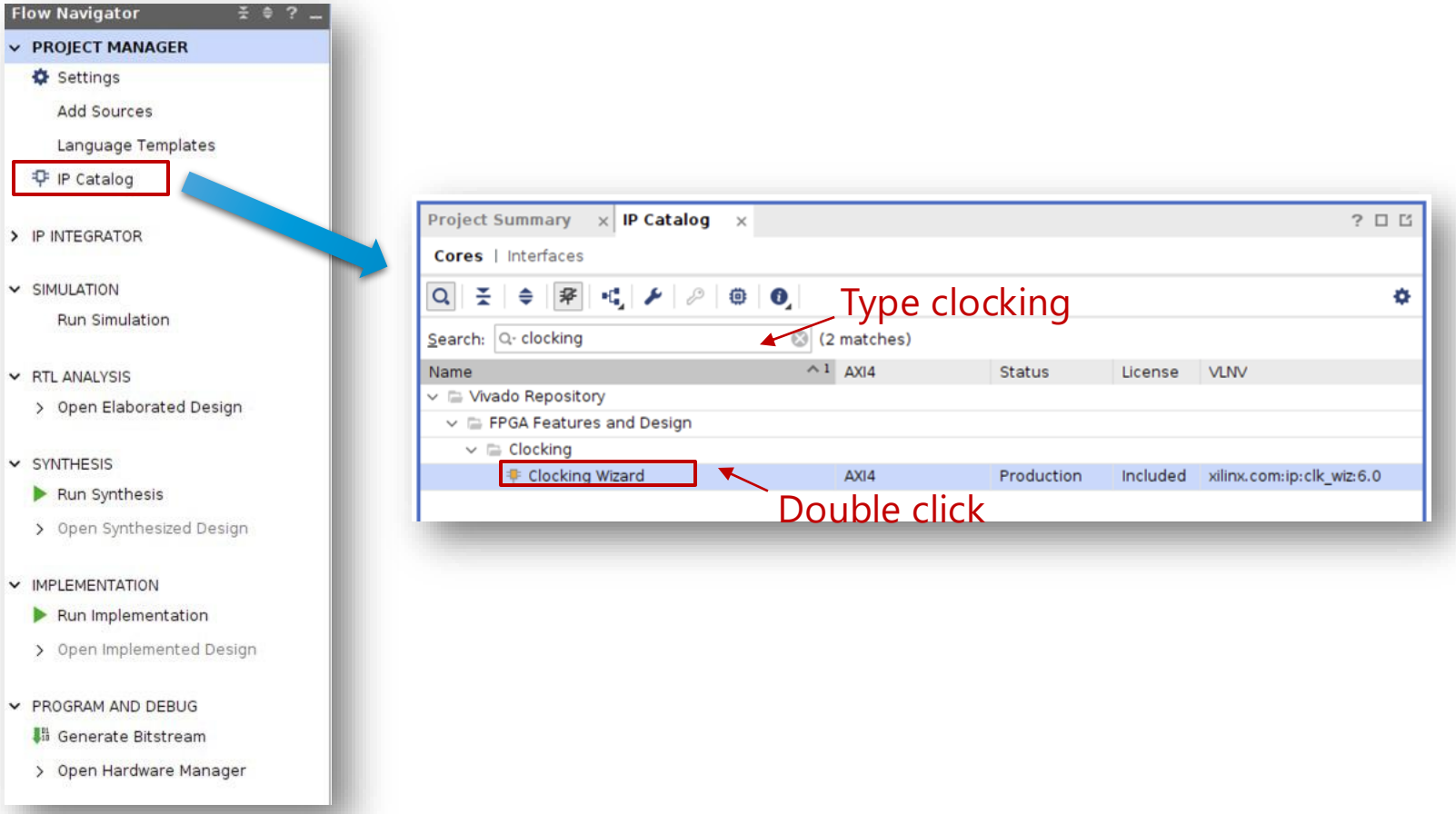
```
//code

reg [3:0] r_state = 0;
always @(posedge w_ui_clk) if (init_calib_complete) begin
    if (r_state==0 && app_rdy && app_wdf_rdy) begin  ///// WRITE_1
        r_app_en       <= 1;
        r_app_wdf_wren <= 1;
        r_app_cmd      <= `CMD_WRITE;
        r_state        <= 1;
    end
    //rest of code
end
```

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

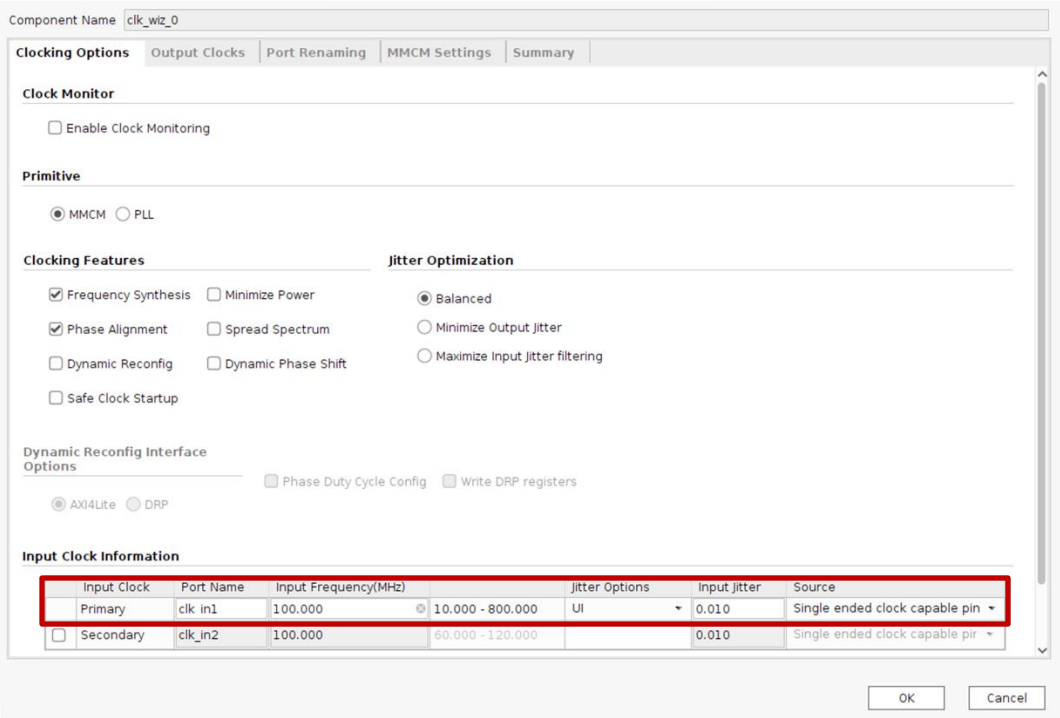
Clocking Wizard でclock を変化させる (1/2)

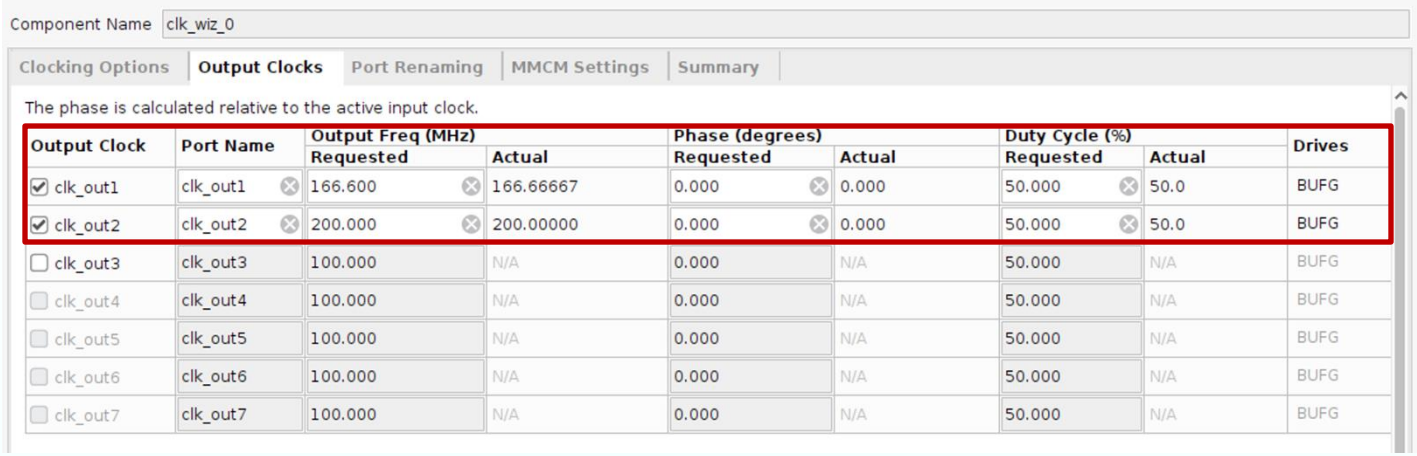
- Click IP Catalog
- Double click Clocking Wizard in IP Catalog window

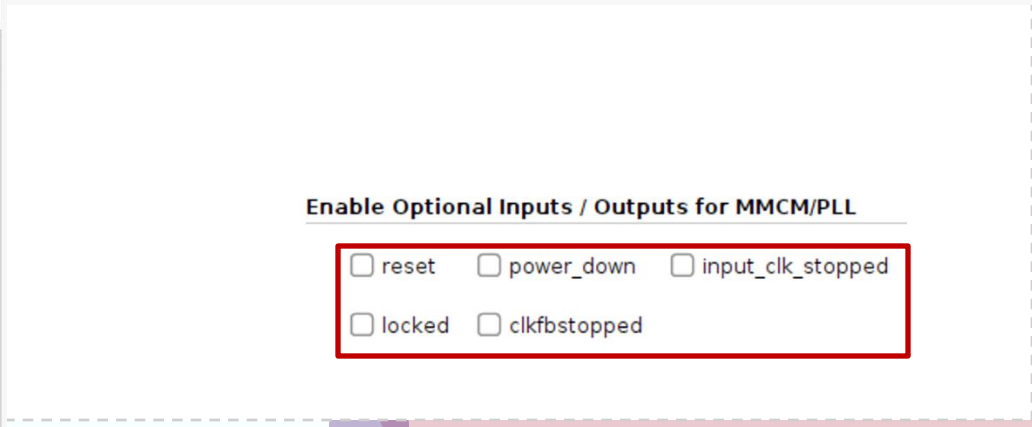


Clocking Wizard でclock を変化させる (2/2)

- Clocking Options, Input Clock Information : 100.000 MHz Input Frequency
- Output Clocks, clk_out1 : 166.6 MHz Output Freq is Requested
- Output Clocks, clk_out2 : 200 MHz Output Freq is Requested
- Output Clocks: disable reset, power_down, input_clk_stopped, locked, clksbstopped

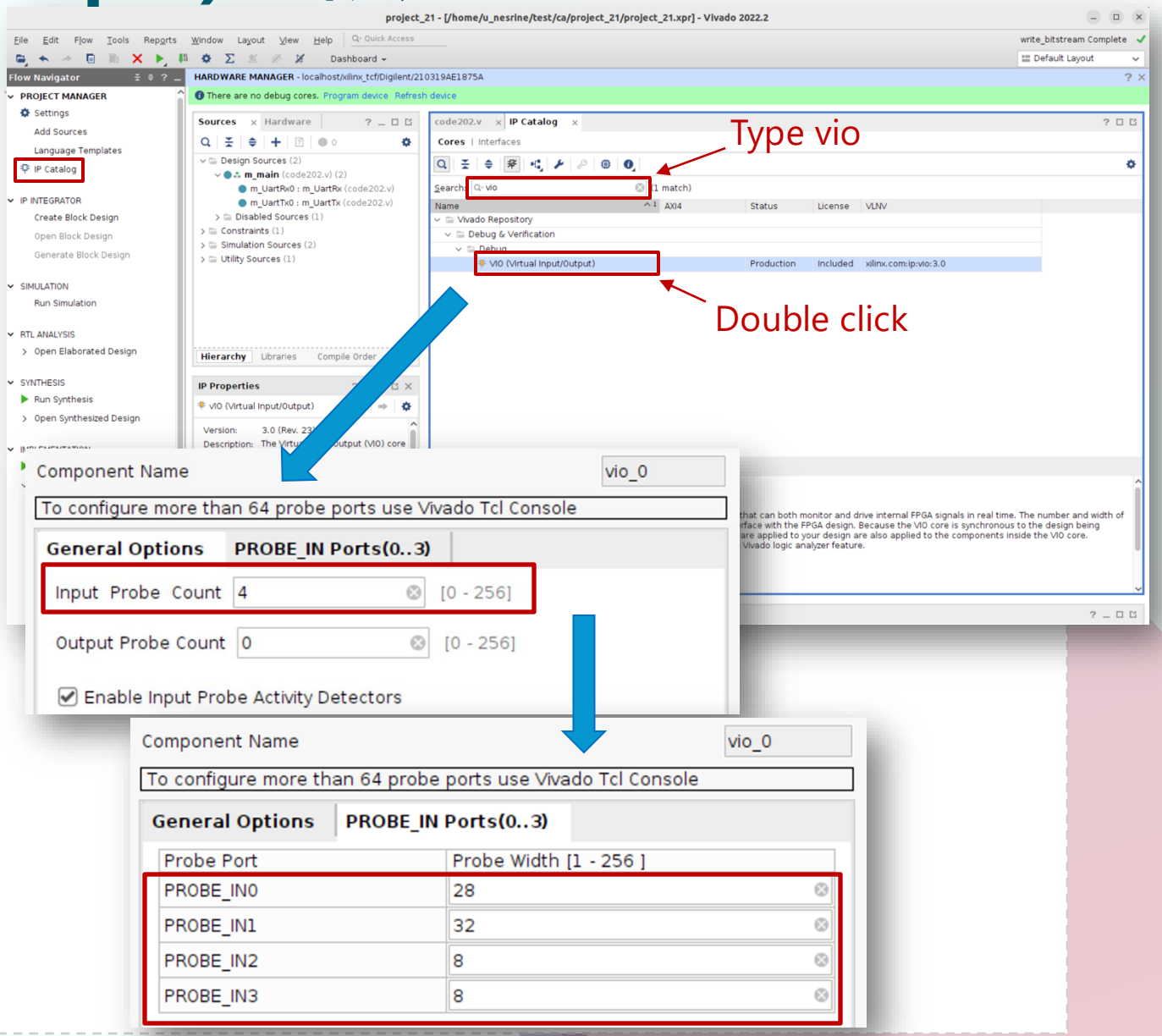






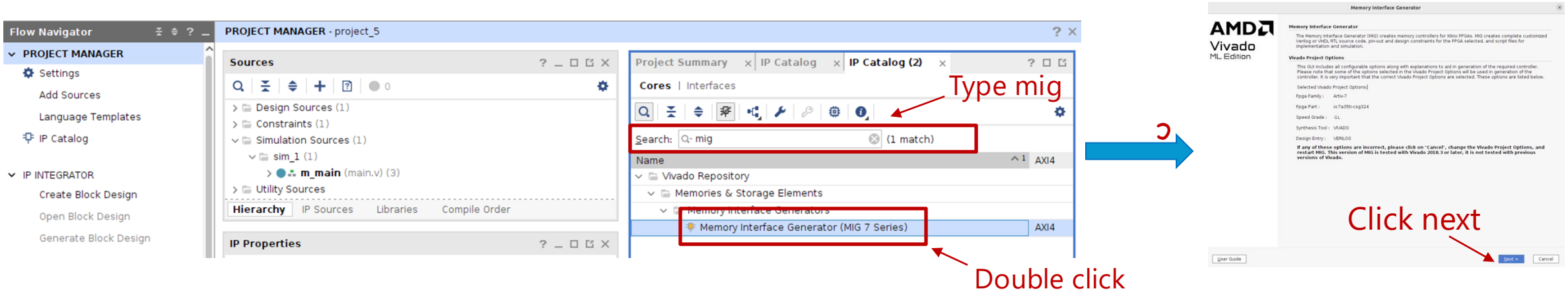
VIO (Virtual Input/Output) の設定

- 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: 4
     - Probe Width:
       - In0: 28
       - In1: 32
       - In2: 8
       - In3: 8
  3. Click Generate and click OK if asked in Generate Output Products window.



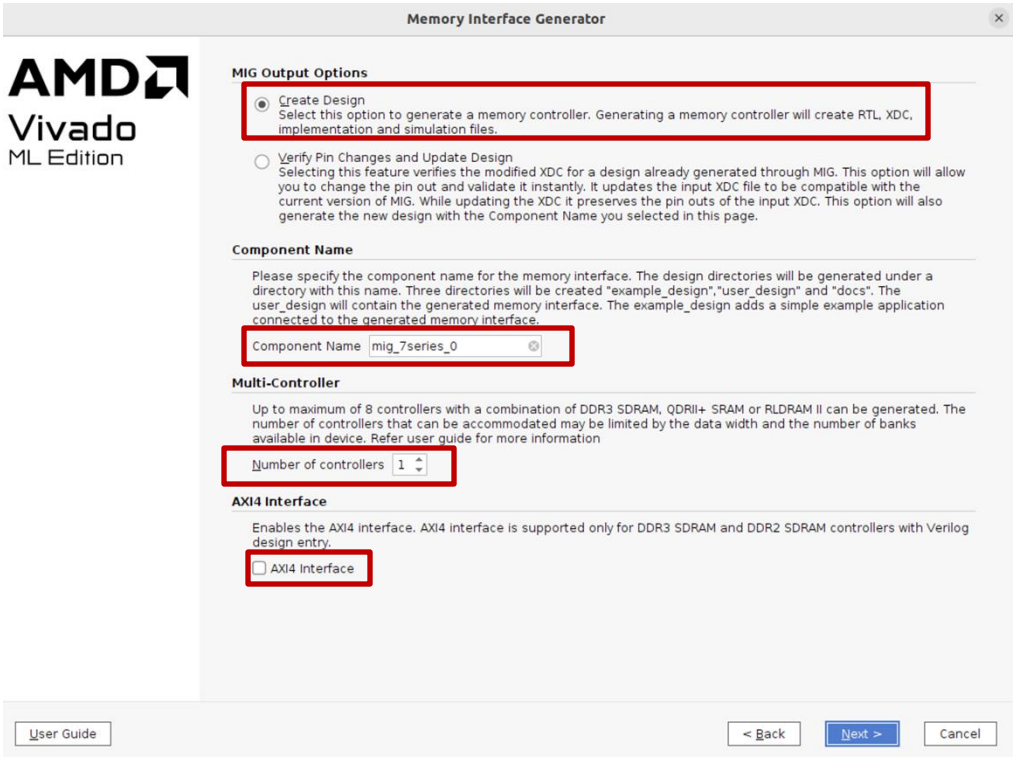
MIG (Memory Interface Generator) の設定 (1/8)

- Xilinx Memory Interface Generator (MIG) という IP コアをベースに Xilinx FPGA で DRAM メモリを動かすこと
- MIG を生成するための IP コアの起動方法
  - Click IP Catalog
  - Double click Memory Interface Generator (MIG 7 Series) in IP Catalog window



MIG (Memory Interface Generator) の設定 (2/8)

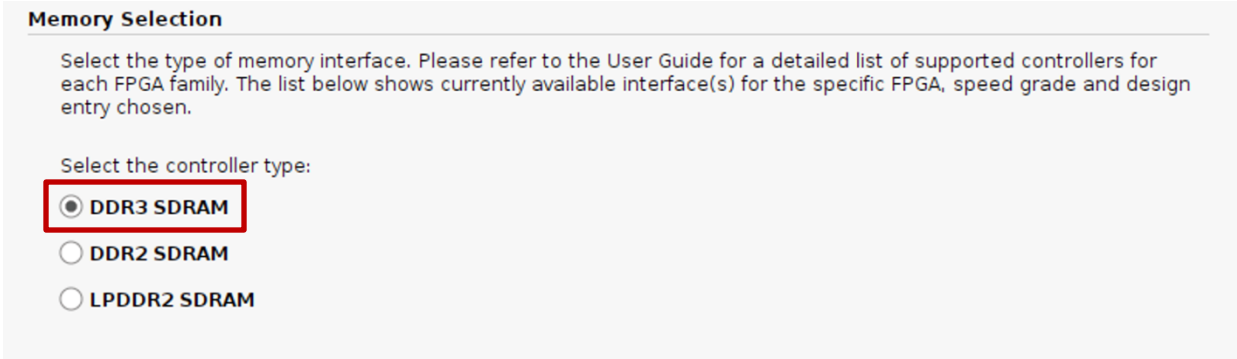
- MIG Output Options: check Create Design
- MIG Output Options, Component Name: mig_7series_0
- MIG Output Options, Multi-Controller: 1
- MIG Output Options, AXI4 Interface: uncheck



MIG (Memory Interface Generator) の設定 (3/8)

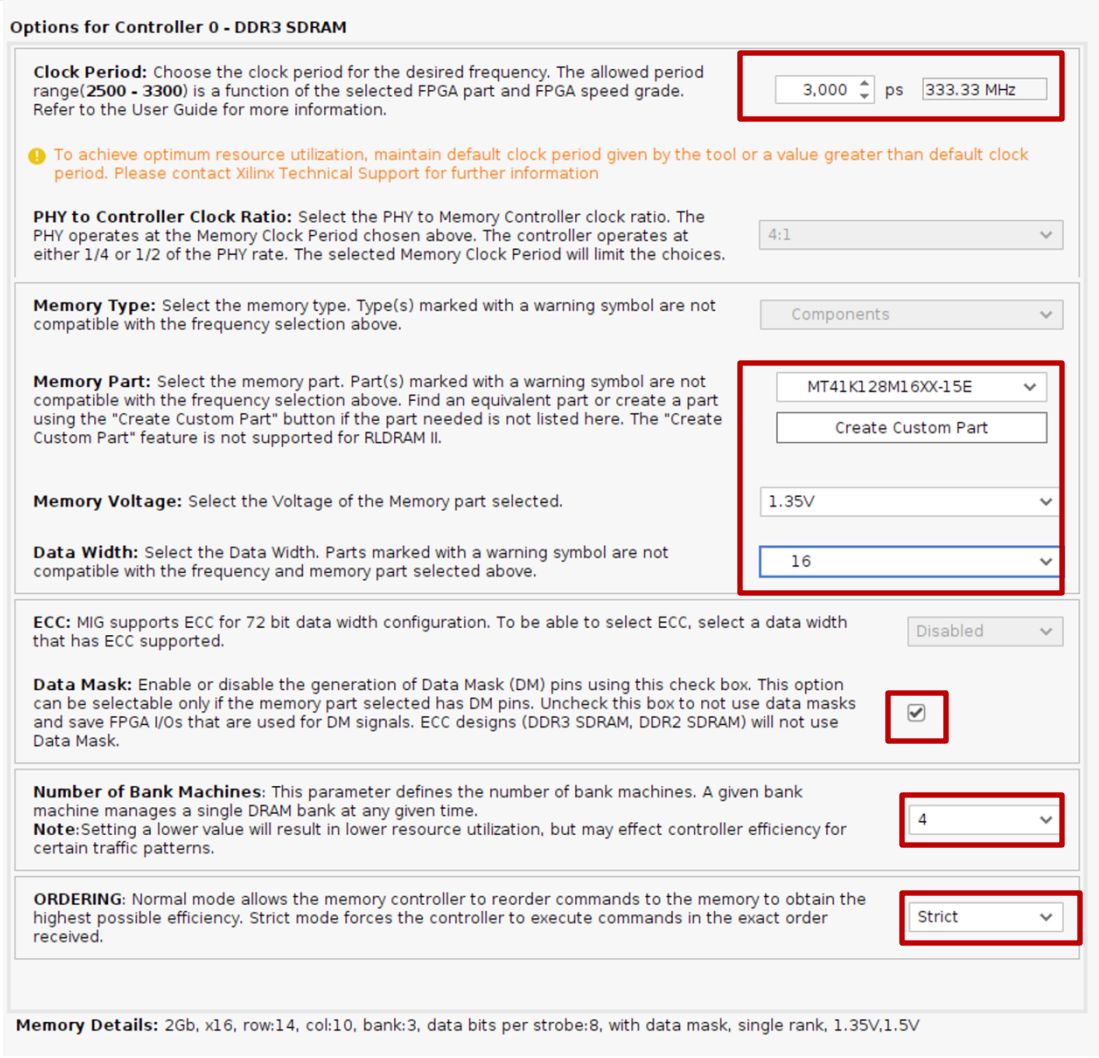
- Pin Compatible FPGAs: uncheck

- Memory Selection: check DDR3 SDRAM



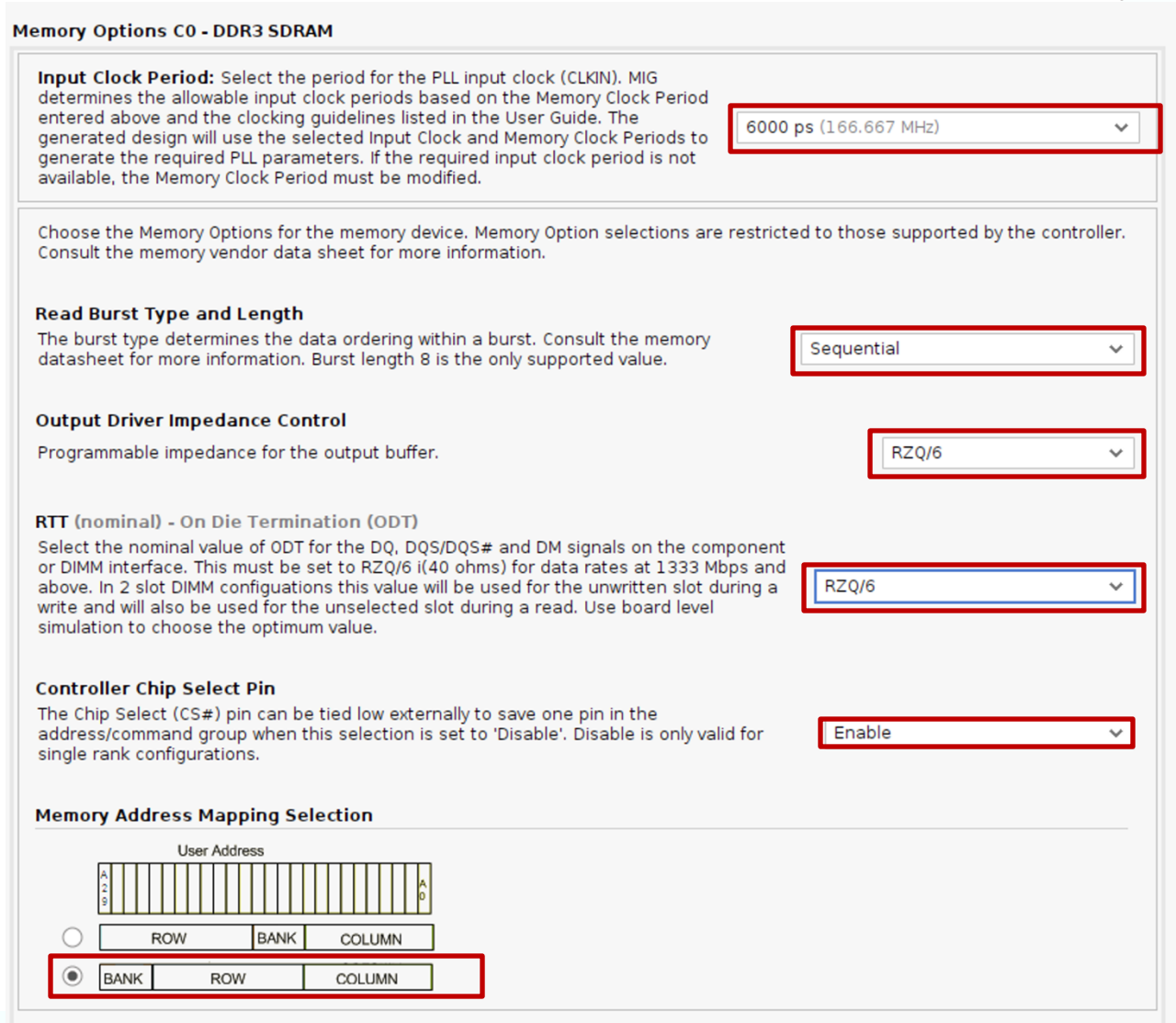
MIG (Memory Interface Generator) の設定 (4/8)

- Options for Controller 0, Clock Period: 3000
- Options for Controller 0, Memory Type: MT41K128M16XX-15E
- Options for Controller 0, Memory Type: Create Custom Part
- Options for Controller 0, Memory Voltage: 1.35V
- Options for Controller 0, Data Width: 16
- Options for Controller 0, Data Mask: check
- Options for Controller 0, Number of Bank Machines: 4
- Options for Controller 0, Ordering: Strict



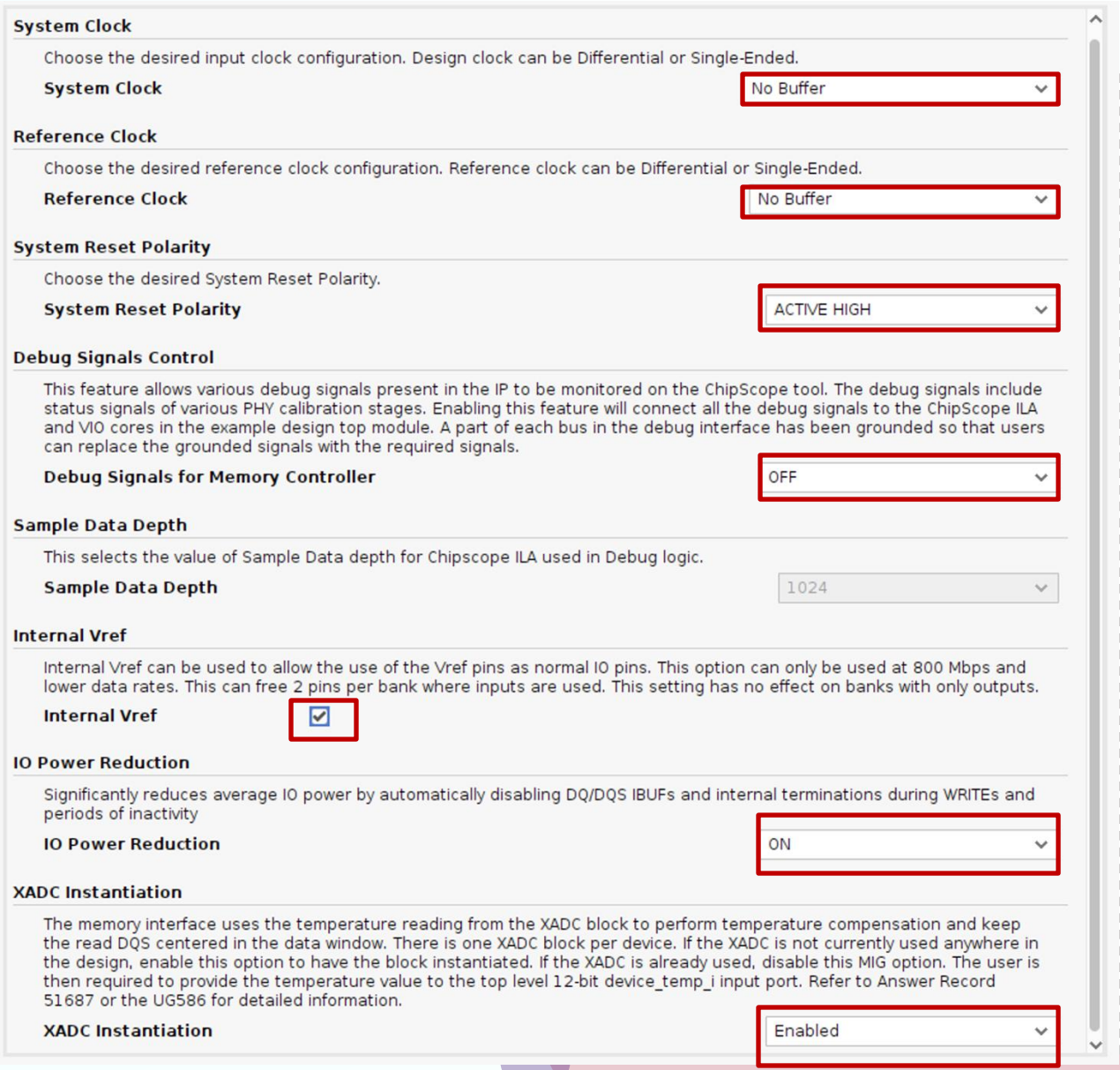
MIG (Memory Interface Generator) の設定 (5/8)

- Memory Options C0, Input Clock Period: 6000 ps
- Memory Options C0, Read Burst Type and Length: Sequential
- Memory Options C0, Output Driver Impedance Control: RZQ/6
- Memory Options C0, RTT: RZQ/6
- Memory Options C0, Controller Chip Select Pin: Enable
- Memory Options C0, Memory Address Mapping Selection: check below (BANK, ROW, COKUMN)



MIG (Memory Interface Generator) の設定 (5/8)

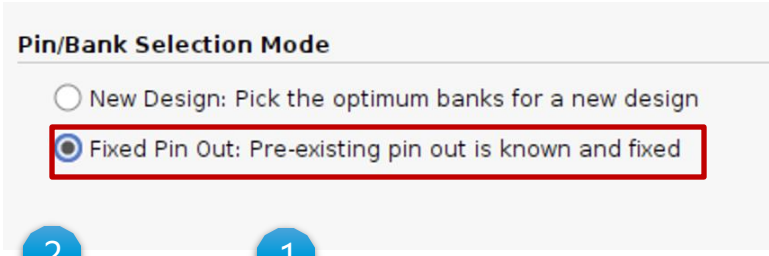
- FPGA Options, System Clock: No Buffer
- FPGA Options, Reference Clock: No Buffer
- FPGA Options, System Reset Polarity: ACTIVE HIGH
- FPGA Options, Debug Signals for Memory Controller: OFF
- FPGA Options, Internal Vref: Check
- FPGA Options, IO Power Reduction: ON
- FPGA Options, XADC Instantiation: Enabled



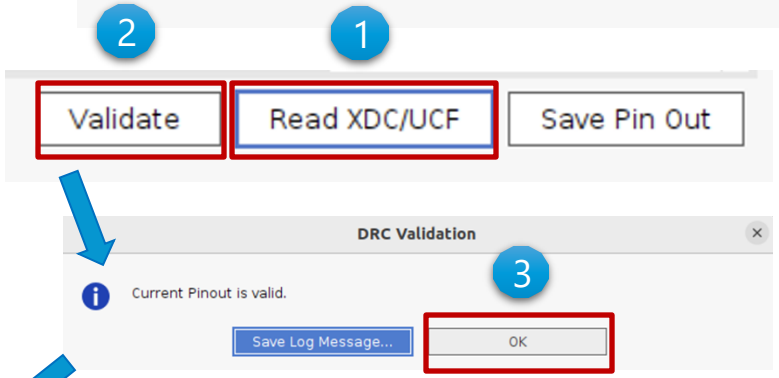
MIG (Memory Interface Generator) の設定 (6/8)

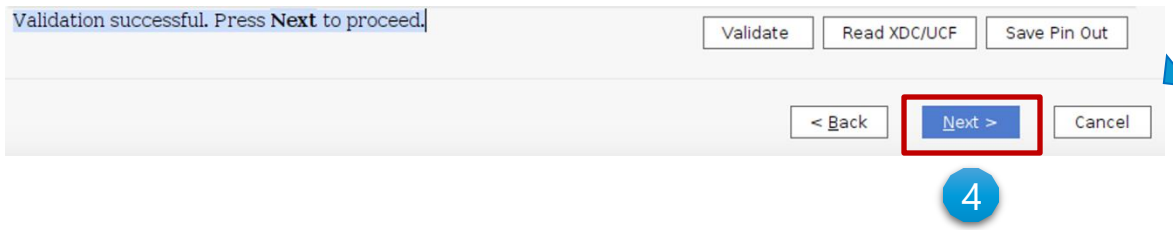
- Extended FPGA Options, Internal Termination Impedance: 50 Ohms

- Pin/Bank Selection Mode: check Fixed Pin Out: Pre-existing pin out is known and fixed



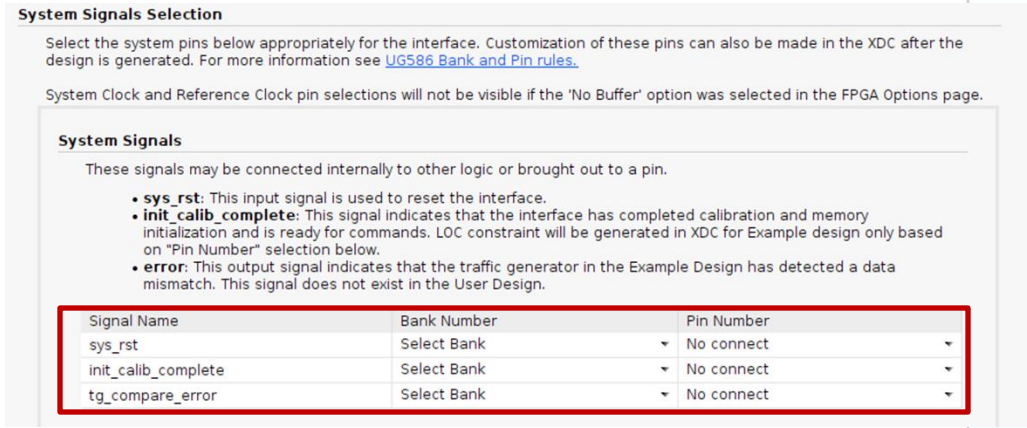
- Pin Selection: click Read XDC/UCF, and select a file named arty_a7_mig.ucf, and click Validate

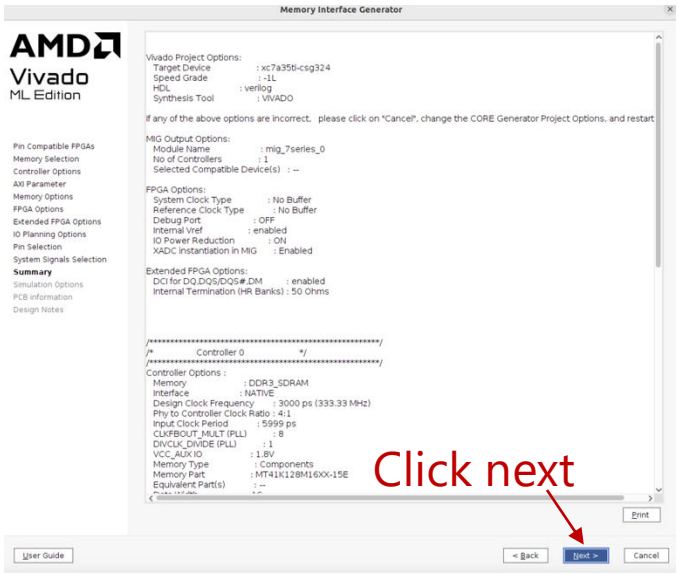


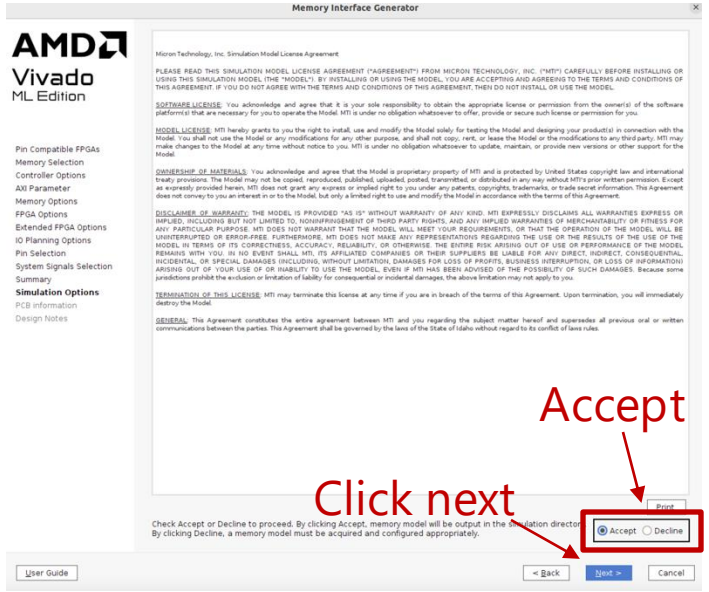


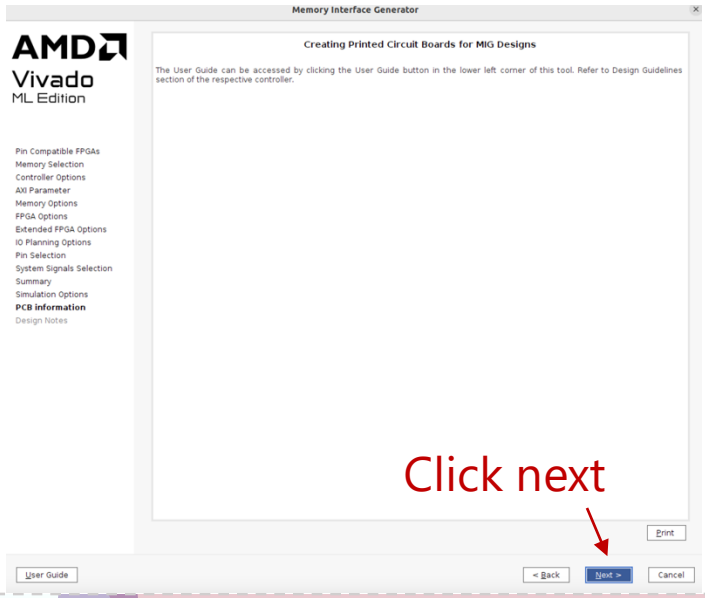
MIG (Memory Interface Generator) の設定 (7/8)

- ❑ System Signals Selection, sys_rst: No connect
- ❑ System Signals Selection, init_calib_complete: No connect
- ❑ System Signals Selection, tg_compare_error: No connect

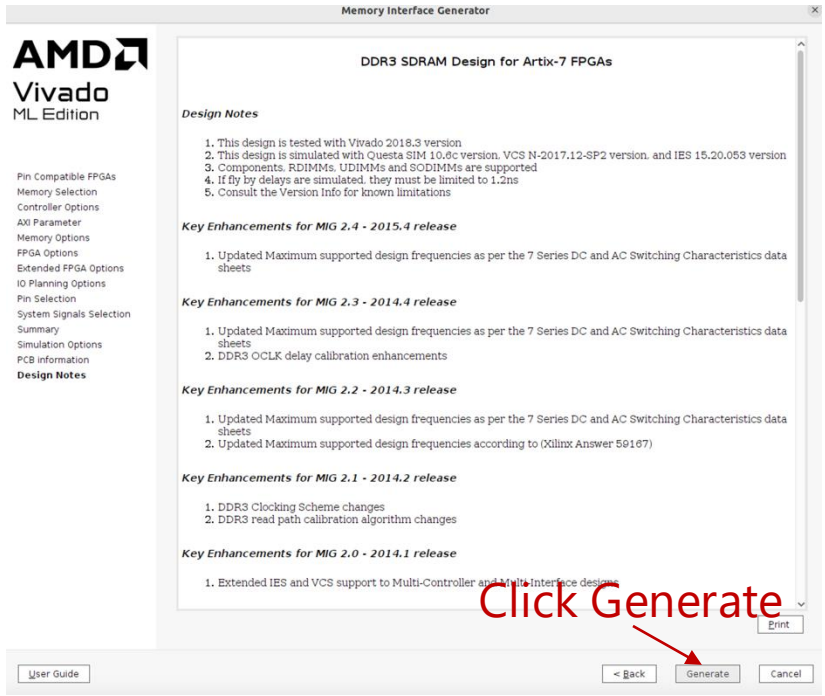


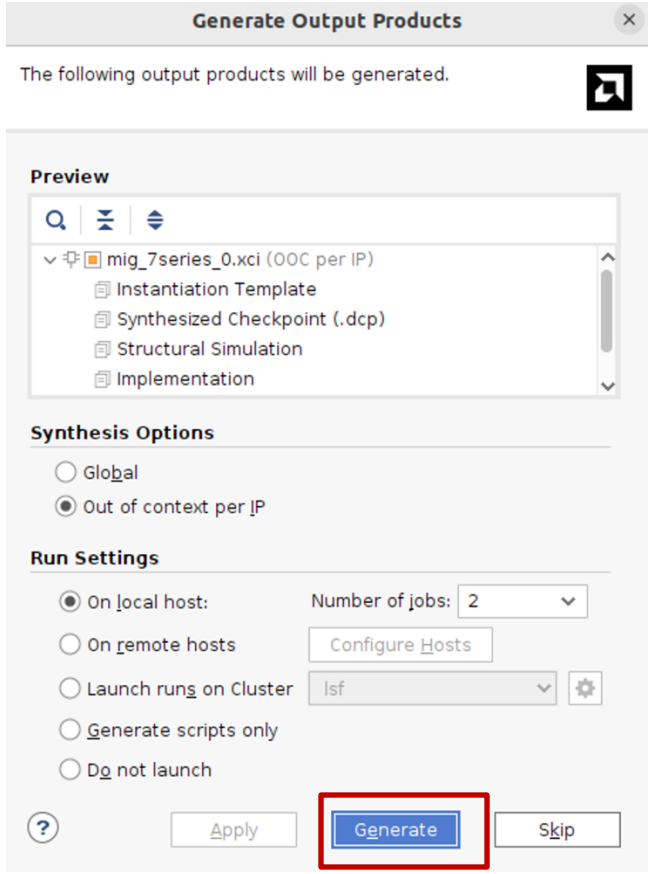






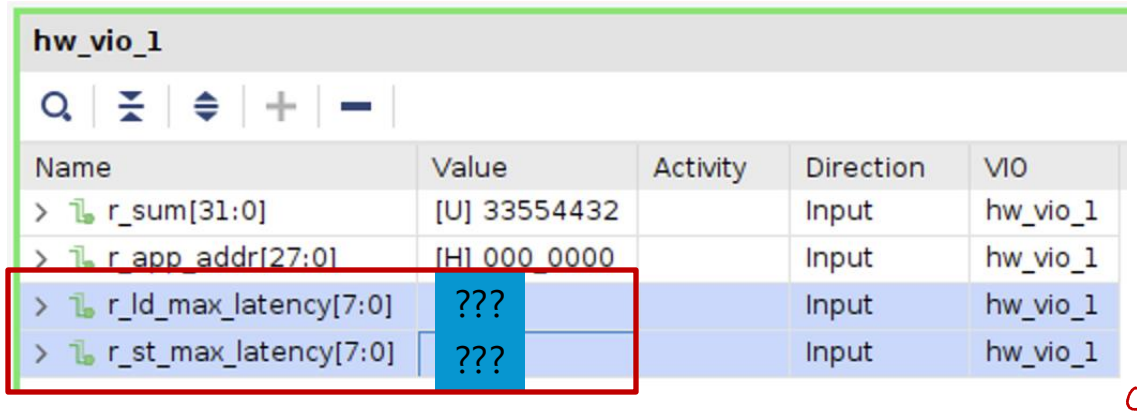
MIG (Memory Interface Generator) の設定 (8/8)





DRAM からのリードとライトのレイテンシを求める

- DRAM の読み出しおよび書き込みにかかる最大のサイクル数を測定する。
- main6.v の m_main モジュールを修正して、DRAM の読み出しと書き込みにかかる最大のサイクル数を求める。
  - DRAM の読み出しおよび書き込みにかかるサイクル数を測定する。
  - 最大のサイクル数を記録する。
  - VIO で確認する。



main6.v

```
//code

reg [9:0] r_ld_latency = 0; //register that stores the latency for a load operation
reg [9:0] r_ld_max_latency = 0; //maximum latency encountered during load operations
reg [9:0] r_st_latency = 0; //register that stores the latency for a store operation
reg [9:0] r_st_max_latency = 0; //maximum latency encountered during store operations

always @(posedge w_ui_clk) if (init_calib_complete) begin
    //Complete here
end

vio_0 vio0 (w_ui_clk, r_app_addr, r_sum, r_st_max_latency, r_ld_max_latency);


//rest of code
```

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

References

- https://www.acri.c.titech.ac.jp/wordpress/archives/6767
- https://github.com/kisek/fpga_arty_a7_dram