第 11 章

FPGA への実装

本章では, 図 4.3 で示したフローの「コーディング」から「回路の実装」に従い, Xilinx 製開発環境である Vivado を使用した FPGA への実装方法について述べる。なお，実装する回路は 4.2.5 項に示される半加算器を対象とする。

１１．１　使用環境

以下に使用環境について示す。

・開発環境：Vivado2022.1（Windows 版）
・開発対象キット：Xilinx Artix-7 35T "Arty" FPGA evaluation kit
・実装対象 FPGA：Artix 7（XC7A35T-L1CSG324-1L（Arty に実装済み））

Xilinx 製 Vivado を使用することで Verilog HDL をはじめとしたハードウェア記述言語による記述および論理合成, シミュレーション, FPGA への実装が可能である。Vivado を Windows もしくは Linux 版を選択した上で, Xilinx のホームページよりダウンロードし，インストールする必要がある。また，開発対象キットである Arty（**図 11.1**）には，Xilinx 製 FPGA である Artix 7 のほかに，RGB LED やスイッチ，DDR3LDRAM(667MHz，256MB) や DA コンバータやフラッシュメモリ（16MB）等が実装されている。

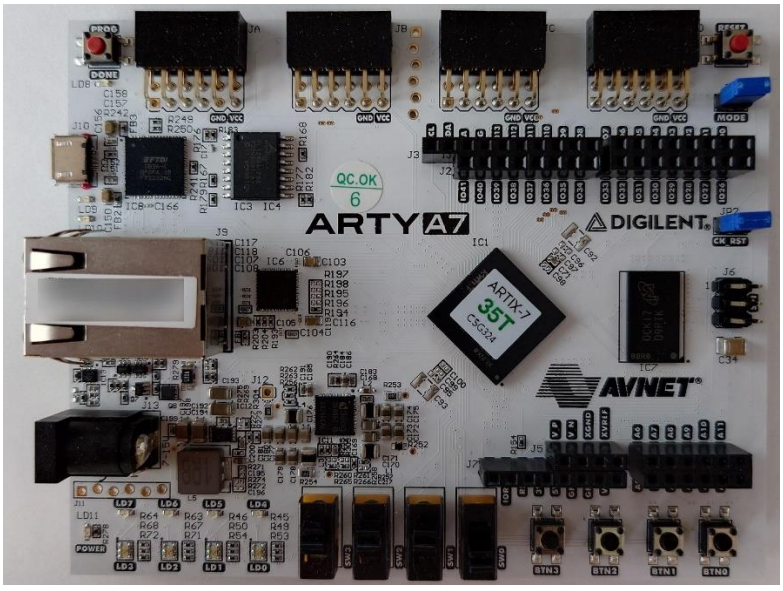


図 11.1　開発対象キット（Arty）

また，開発環境を使用する上での注意点を以下に示す。

・ファイル名や保存場所に日本語を入れない（文字化けにより正常に動作しない場合がある）
・Windows のユーザー名に日本語やスペースを入れない（インストーラが実行されない場合がある）

１１．２　プロジェクトの作成および論理合成

1)　Vivado を開き「Create Project」を選択する（**図 11.2**）。

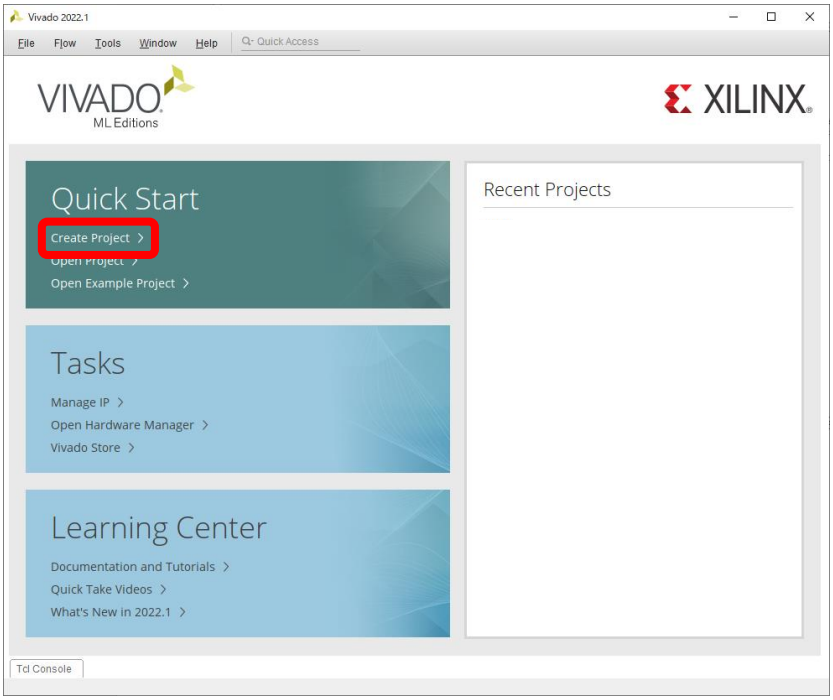


図 11.2　プロジェクトの作成および論理合成の手順 1

2)　「Next」を選択する（**図 11.3**）。

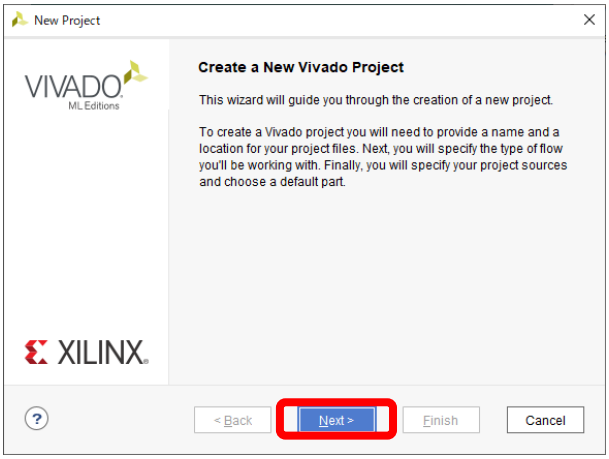


図 11.3　プロジェクトの作成および論理合成の手順 2

3)　「Project name」（半加算器ということで HA）および「Project location」を適宜設定する（**図 11.4**）。

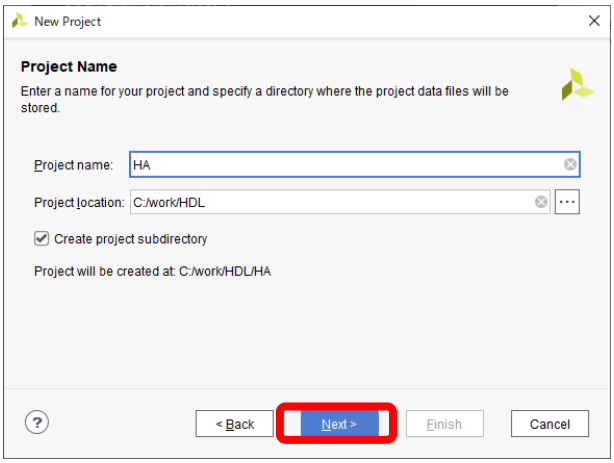


図 11.4　プロジェクトの作成および論理合成の手順 3

4)　「RTL Project」を選択する（**図 11.5**）。

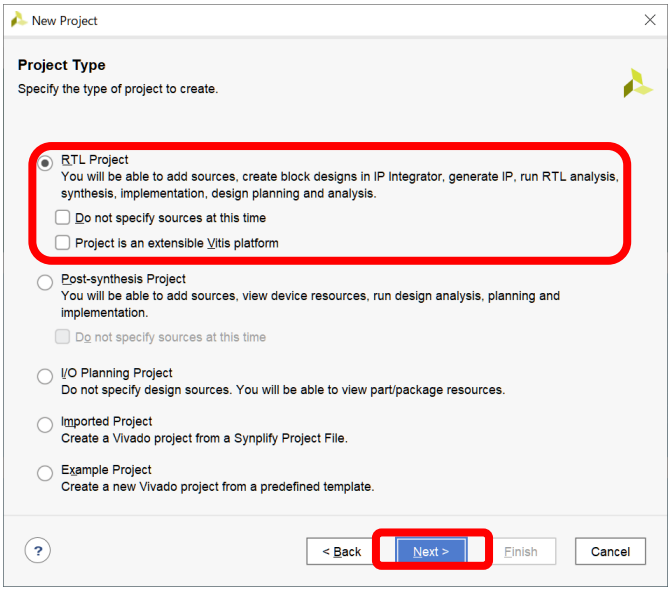


図 11.5　プロジェクトの作成および論理合成の手順 4

5)　「Create File」をクリックし，回路用 Verilog HDL ファイルとして「File name:」を入力し「OK」を選択する（**図 11.6**）。この場合は，プロジェクト名と同一の「HA」とする。これにより，ファイル名「HA.v」とする Verilog HDL のファイルが生成される。なお，Xilinx の開発環境の場合，不具合が発生する可能性があるため，プロジェクト名とトップモジュールが含まれるファイル名は同一とすることを推奨する。

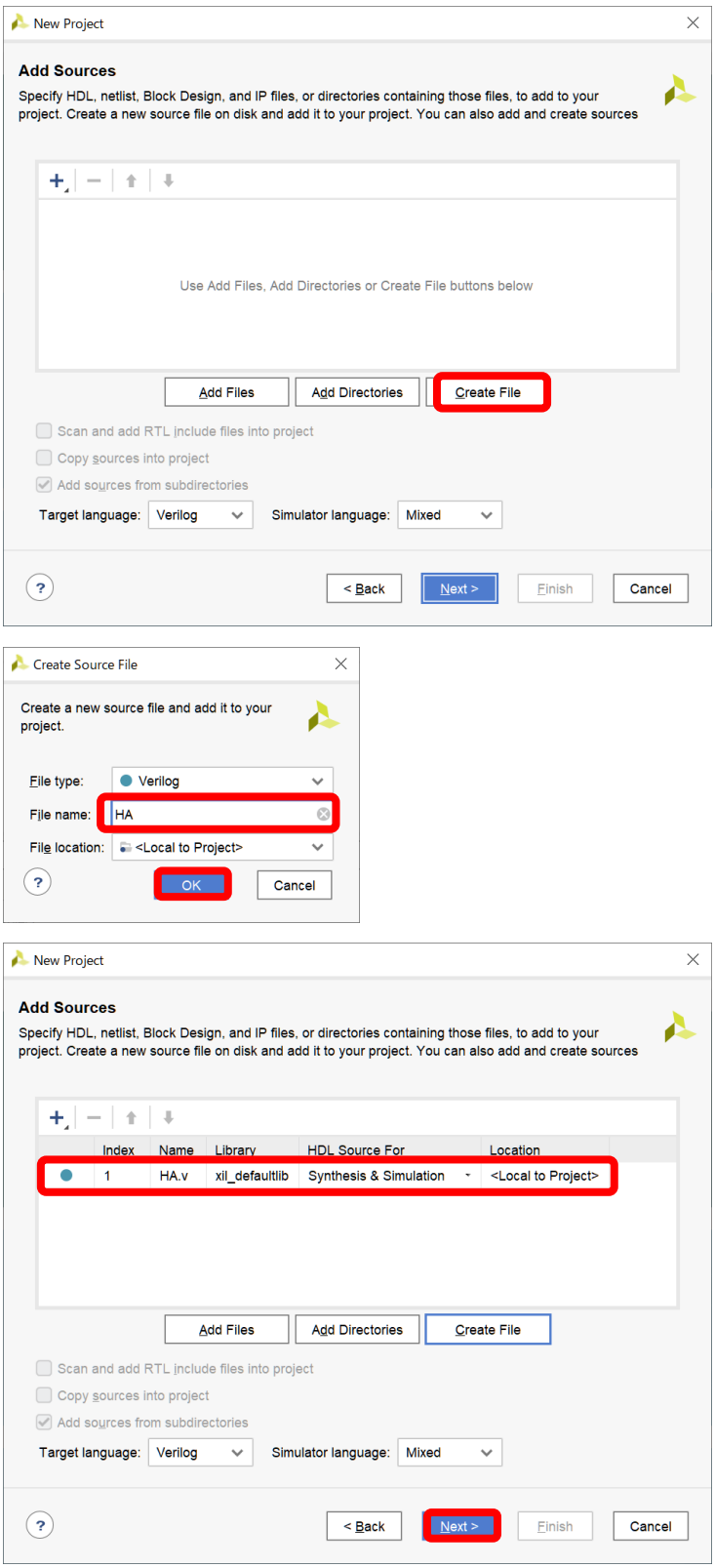


図 11.6　プロジェクトの作成および論理合成の手順 5

6） 後ほど別途追加するため，この手順では「Next」を選択する（**図 11.7**）。

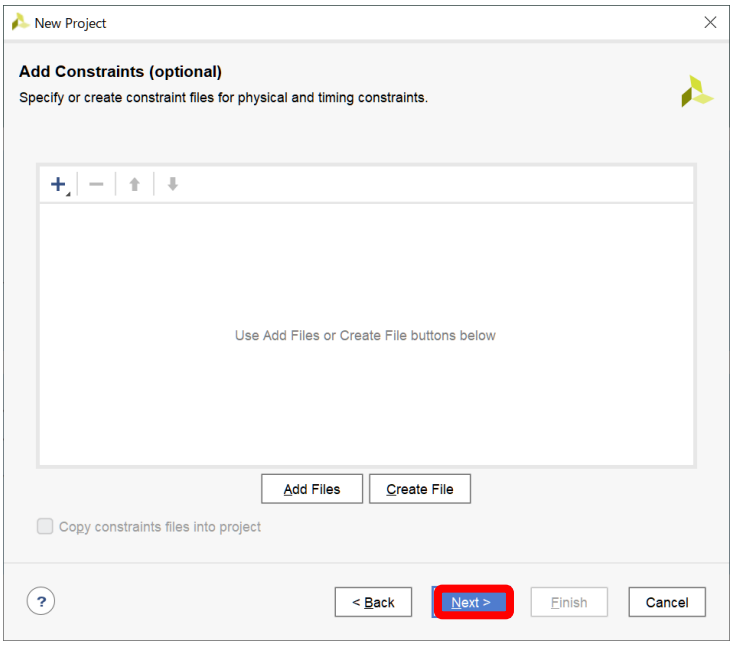


図 11.7　プロジェクトの作成および論理合成の手順 6

7） 「Family」および「Package」，「Speed」を選択した後，「xc7a35ticsg324-1L」を選択する（**図 11.8**）。

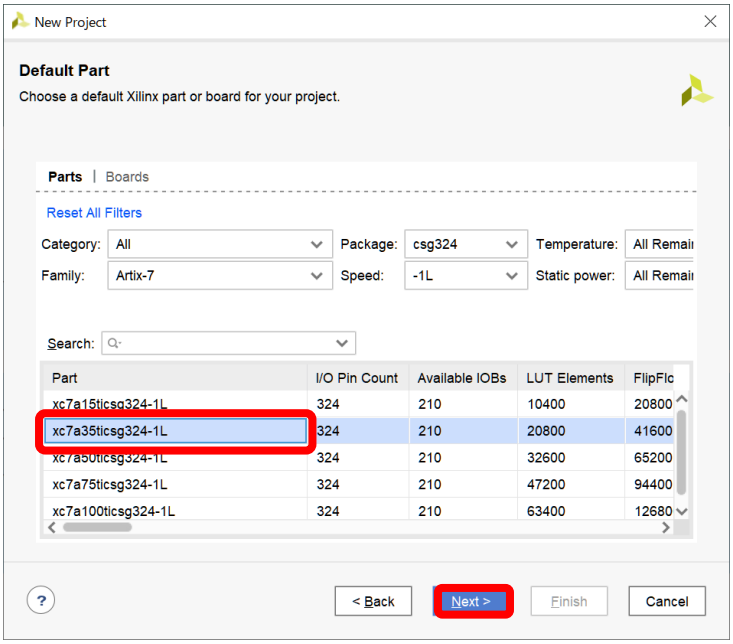


図 11.8　プロジェクトの作成および論理合成の手順 7

8)　「Finish」を選択する（**図 11.9**）。

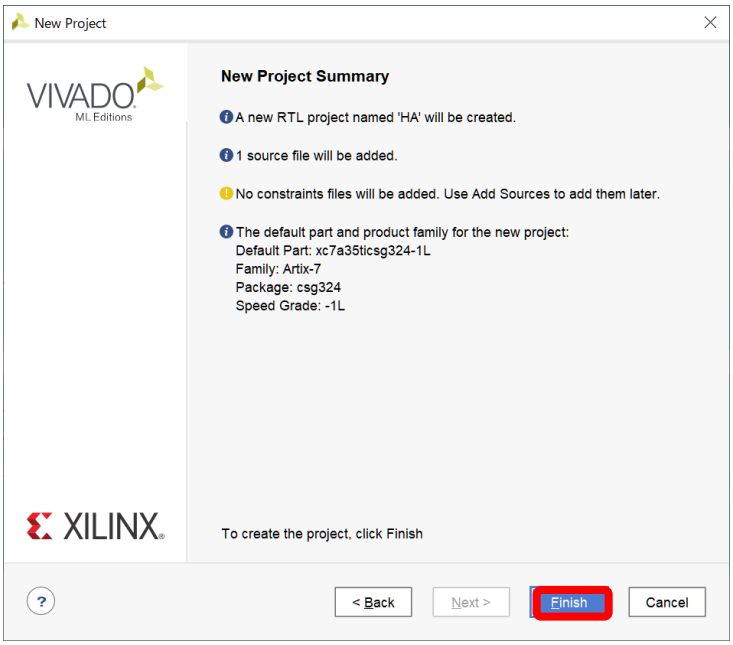


図 11.9　プロジェクトの作成および論理合成の手順 8

9)　「OK」を選択し，その後「Yes」を選択する（**図 11.10**）。

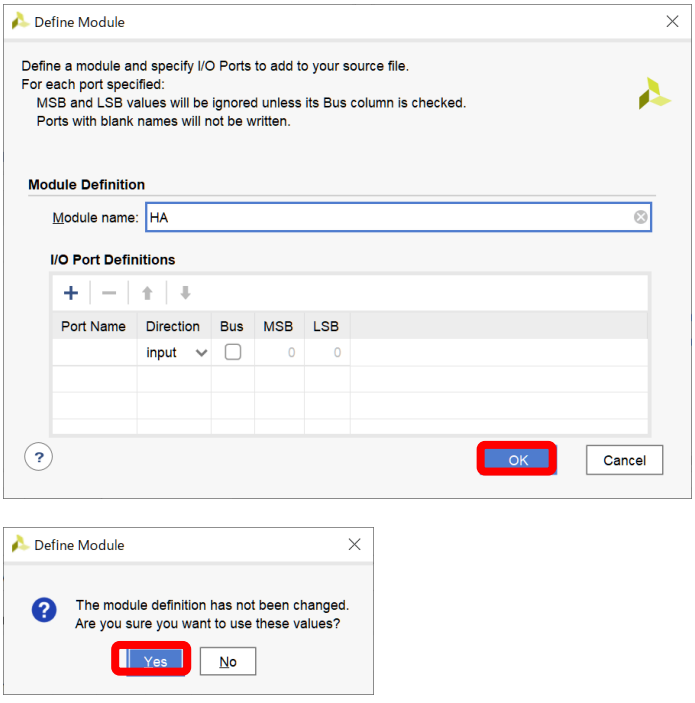


図 11.10　プロジェクトの作成および論理合成の手順 9

10）「Design Sources」直下の表記（この場合は HA）をダブルクリックする。その後，右のエリアにプログラムを記述するスペースが表示される（**図 11.11**）。

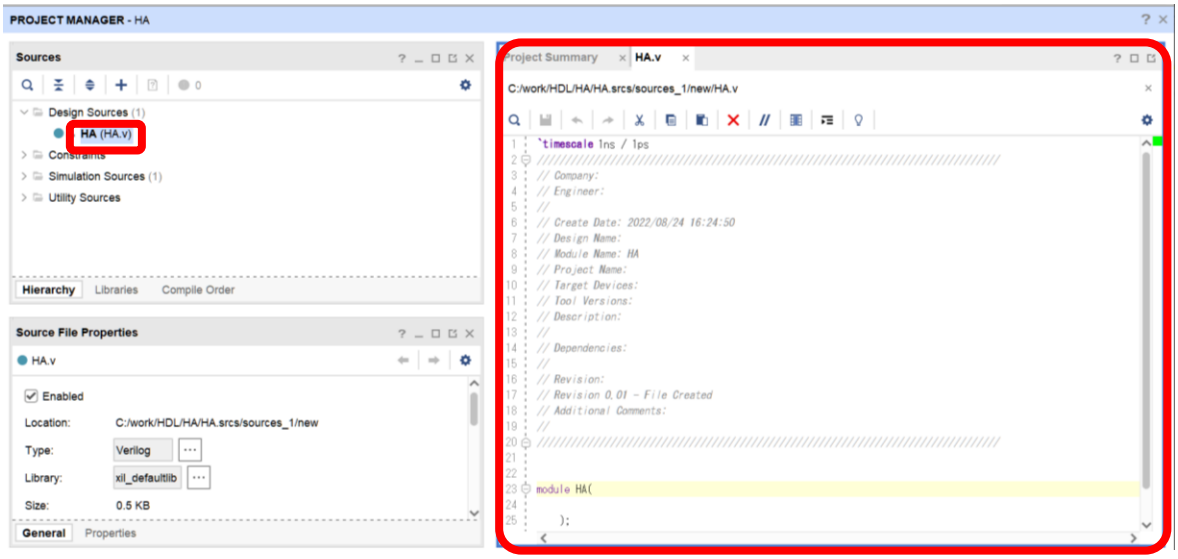


図 11.11　プロジェクトの作成および論理合成の手順 10

11）以下のように 4.2.5 項のコード 4.3 と同様に記述する（**図 11.12**）。

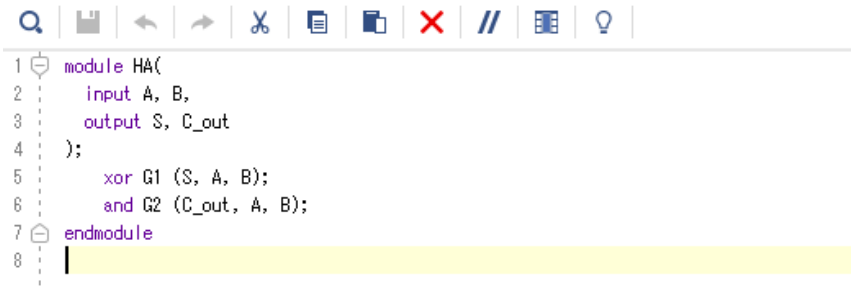
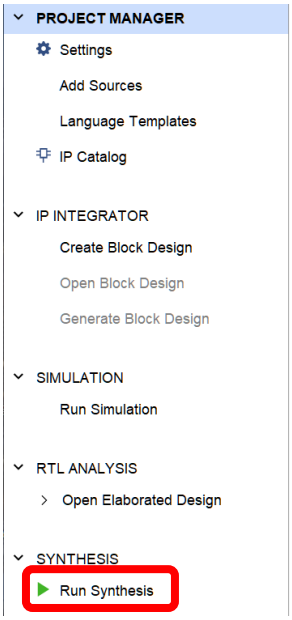


図 11.12　プロジェクトの作成および論理合成の手順 11

12）「Run Synthesis」→「Save」の順に選択する（**図 11.13**）。



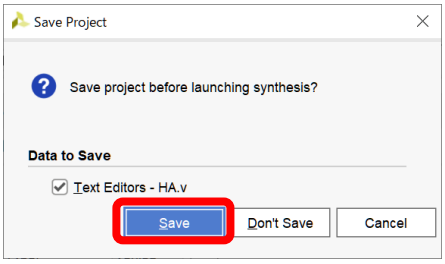


図 11.13　プロジェクトの作成および論理合成の手順 12

13）「Number of jobs」の数を調整し「OK」を選択すると論理合成が開始される（**図 11.14**）。この数は，コンピュータのプロセッサ数の仕様に依存し，大きいほど処理にかかる時間を短縮できる。ただし，ほかのアプリケーションを実行し作業する際は，最大の数を指定しないことを推奨する。

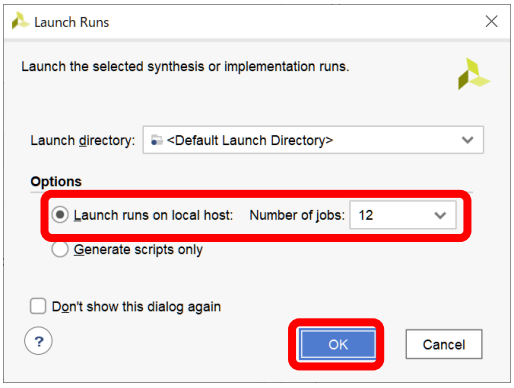


図 11.14　プロジェクトの作成および論理合成の手順 13

14）論理合成実施中はウインドウ右上の表示がこのような表示となる（**図 11.15**）。処理が完了するまで待機する。

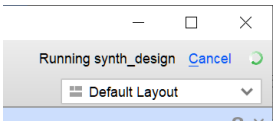


図 11.15　プロジェクトの作成および論理合成の手順 14

15）論理合成が正常に終了すると，このような表示となる。ここでは「Cancel」を選択する（**図 11.16**）。

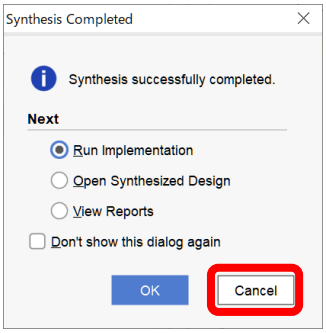


図 11.16　プロジェクトの作成および論理合成の手順 15

16）**図 11.17** のように「Synthesis failed」の表示となった場合は「OK」を選択し，ウインドウ下部に表示されるメッセージを確認する。この際，「Error」と「Critical warning」にチェックが入っていることを確認する。「Critical warning」がトリガーとなって，「Error」が発生することが多いため，この 2 種類のメッセージに着目する。

また，**図 11.17** のプログラム記述において，意図的に 7 行目の「endmodule」の「e」を抜いて，エラーを発生させた例であり，記述ミスの場合は記述の段階で波線が表示される。また，メッセージ末尾の[HA.v:7]は，ファイル名「HA.v」の 7 行目でエラーが発生していることを示している。この場合は「endmodule」に記述ミスが存在することから，「Critical warning」として構文エラーが検出され，「Error」としてプログラムの終了を検出できないことから，7 行目で EOF を検出できないというメッセージが表示されている。

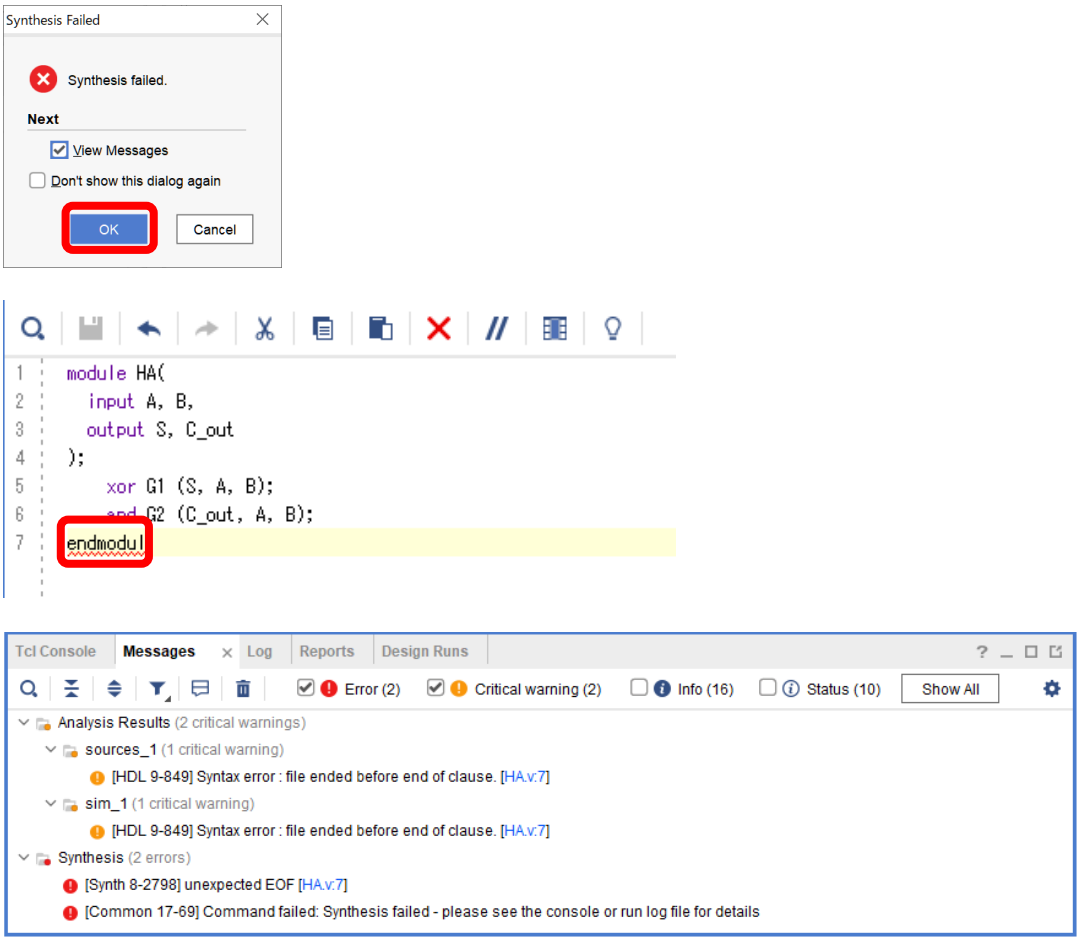


図 11.17　Synthesis failed の例

１１．３　シミュレーション

1)　ウインドウ左側の「Add Sources」を選択する（**図 11.18**）。

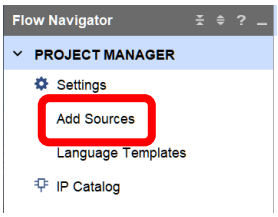


図 11.18　シミュレーションの手順 1

2)　「Add or create simulation sources」を選択する（**図 11.19**）。

図 11.19　シミュレーションの手順 2

3)　「Create File」を選択する（**図 11.20**）。

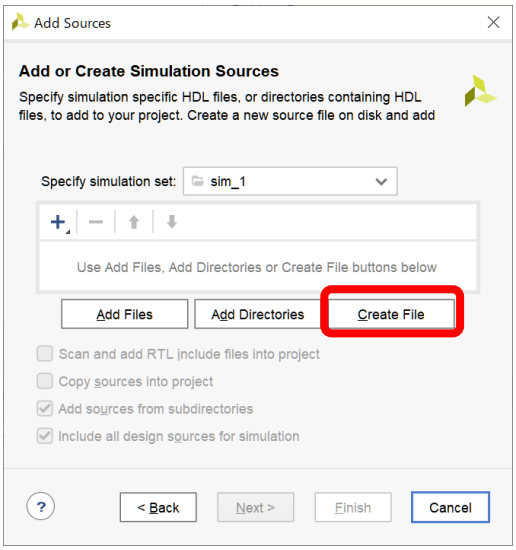


図 11.20　シミュレーションの手順 3

4)　**図 11.21** のように「File name:」にシミュレーションに使用するテストベンチ用の Verilog HDL ファイル名を指定する。回路用のファイルと同一の命名はできないため，同一でないファイル名を指定する。なお，回路用ファイル名に「_sim」を付加すると，付加しやすく明確に示すことができる。この場合は，「HA_sim」とする。

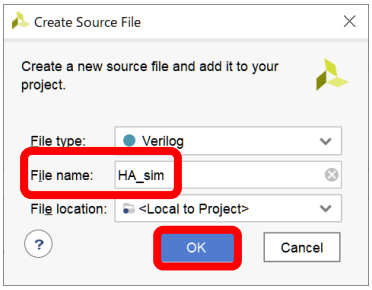


図 11.21　シミュレーションの手順 4

5)　「Finish」を選択する（**図 11.22**）。

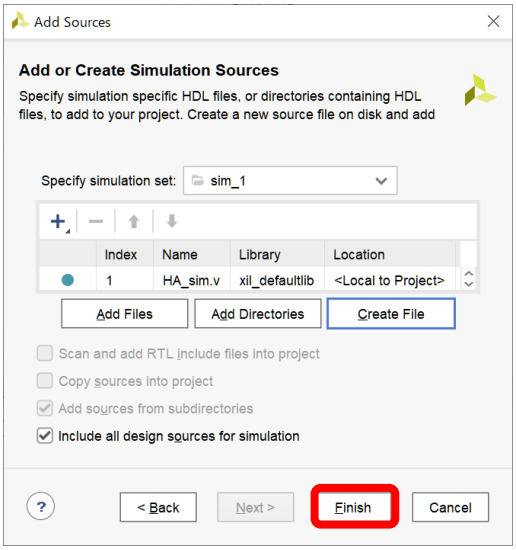


図 11.22　シミュレーションの手順 5

6)　「OK」を選択し，その後「Yes」を選択する（**図 11.23**）。

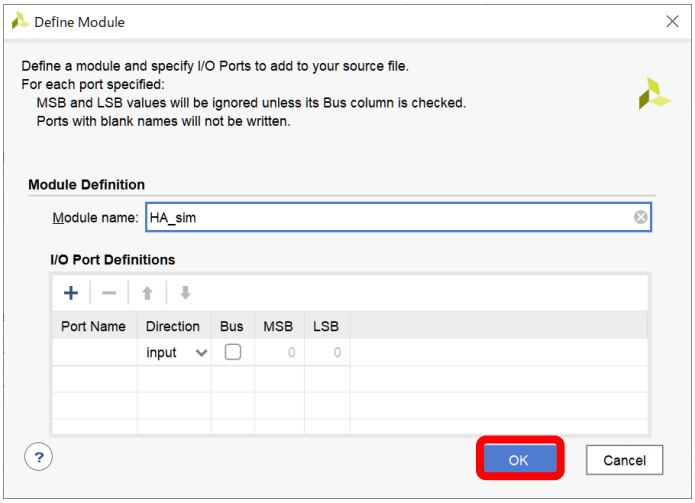


図 11.23　シミュレーションの手順 6

7)　「Simulation Sources」→「sim_1」をそれぞれダブルクリックする。その後，「HA_sim」をダブルクリックする（**図 11.24**）。

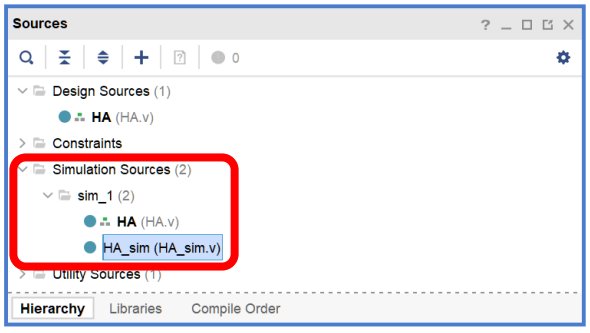


図 11.24　シミュレーションの手順 7

8)　コード 6.4 と同様に以下のプログラム（**図 11.25**）を右のエリア（HA_sim.v のタブ）に記述する（**図 11.26**）。

```
`timescale 1ns / 1ps
module HA_sim(

    );

    reg simA = 1'b0;
    reg simB = 1'b0;

    wire simS,simC_out;

    initial begin
    simA = 1'b0;
    simB = 1'b0;

    simA = 1'b0;
    simB = 1'b0;
    #10
    simA = 1'b1;
    simB = 1'b0;
    #10
    simA = 1'b0;
    simB = 1'b1;
    #10
    simA = 1'b1;
    simB = 1'b1;
    #10
    $finish;
    end

  HA HA0(
        .A(simA),
        .B(simB),
        .S(simS),
        .C_out(simC_out)
     );

endmodule
```

図 11.25　HA のシミュレーション記述例

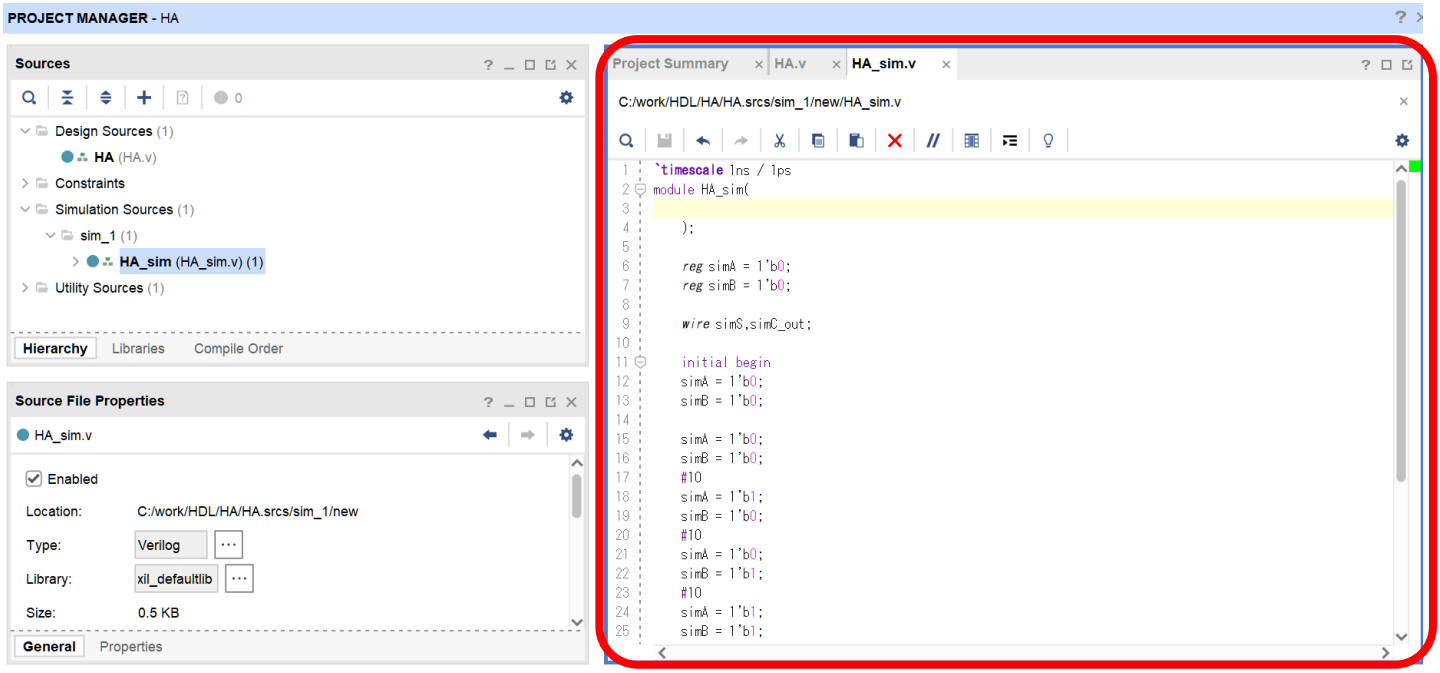


図 11.26　シミュレーションの手順 8

9)　**図 11.27** のように，左のメニューより「Run Simulation」→「Behavioral Simulation」を選択する。その後，シミュレーションが実行される。なお，Behavioral Simulation は，回路の遅延等を考慮せず設計した回路の振る舞いを確認するシミュレーションである。したがって，配線の遅延等は考慮されていないため，FPGA に実装した際のシミュレーションを行うためには，これ以外のシミュレーションを選択する。

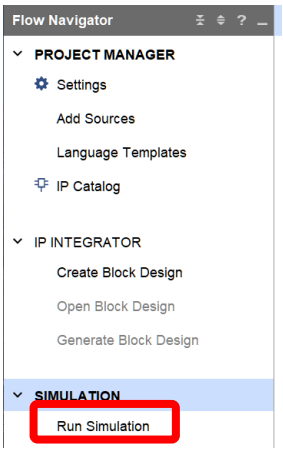


図 11.27　シミュレーションの手順 9

10) シミュレーションが正常に終了すると，「Untitled 1」のタブが追加されるので，選択するとタイミングチャートとしてシミュレーション結果が表示される。その後，「Zoom Fit」ボタンをクリックすると，シミュレーション全体を表示できる（**図 11.28**）。

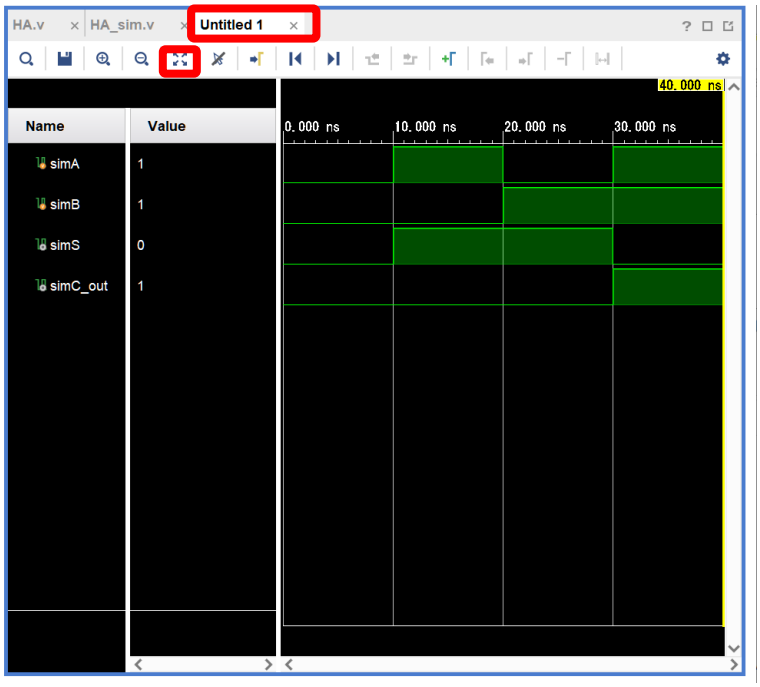


図 11.28　シミュレーションの手順 10

11）図 11.29 のように「ERROR」の表示となった場合は「OK」を選択し，表示されるメッセージを確認する。ただし，シミュレーションの場合は論理合成と異なり，具体的なエラー箇所が直接表示されないため，メッセージ内に表示され，ログ情報が格納されている「xvlog.log」ファイルをメモ帳等より開いて確認する。

図 11.29 は，意図的に 6 行目の「reg simA = 1'b0;」をコメントアウトし，エラーを発生させた例となる。また，図 11.30 に示す「xvlog.log」内のメッセージ末尾の[HA_sim.v:12]等は，回路の論理合成と同様にファイル名「HA_sim.v」の 12 行目でエラーが発生していることを示している。この場合はテストベンチ内に「simA」を使用しているものの，「simA」を宣言していないことから，登録されていない simA には代入できない旨のエラーメッセージが表示されている。

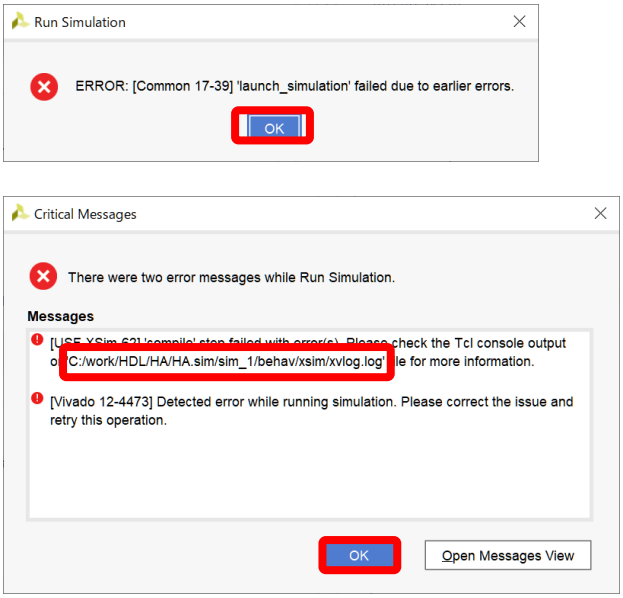


図 11.29　シミュレーションエラーの例

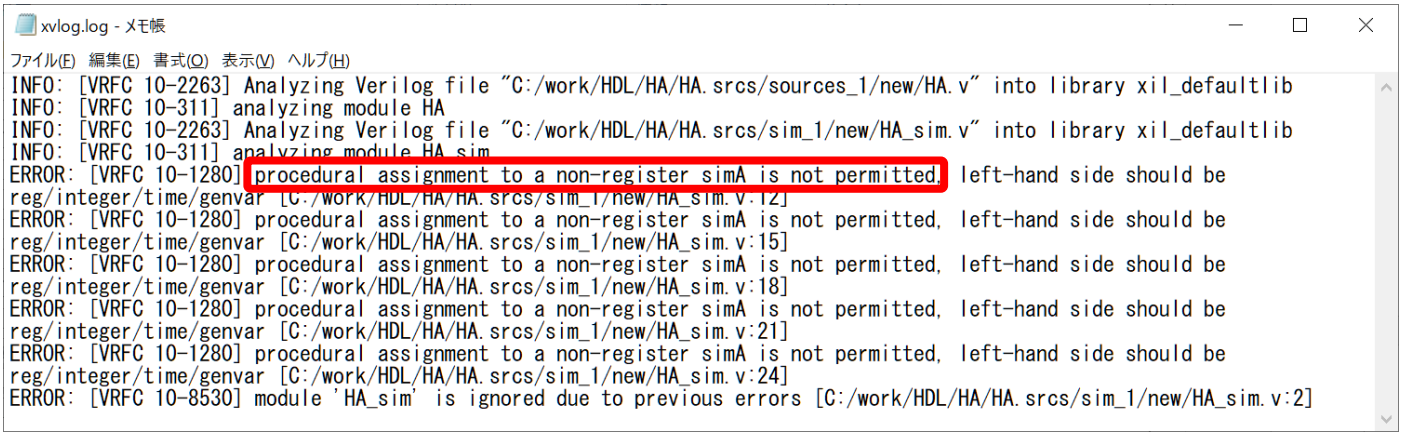


図 11.30　xvlog.log の出力例

12）手順 10 で表示されたシミュレーション結果と期待値を比較し期待どおりの動作をしているかを確認する。タイミングチャートと**表 11.1** に示す真理値表を照合すると値が一致しているため，期待どおりの機能が確認できたことになる。

表 11.1　HA の真理値表

入力		出力	
simA（入力 A）	simB（入力 B）	simS（加算結果）	simC_out（桁上がり出力）
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

１１．４　FPGA への実装

1）評価ボード（Arty）に実装されている FPGA の I/O 設定を記述する Constraints ファイルを，以下のアドレスの「Raw」ボタン上で右クリックし「名前を付けてリンク先を保存」を選択し，保存する。
https://github.com/Digilent/digilent-xdc/blob/master/Arty-A7-35-Master.xdc

2）左のメニュー内の「Add Sources」を選択する（**図 11.31**）。

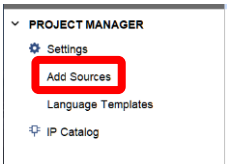
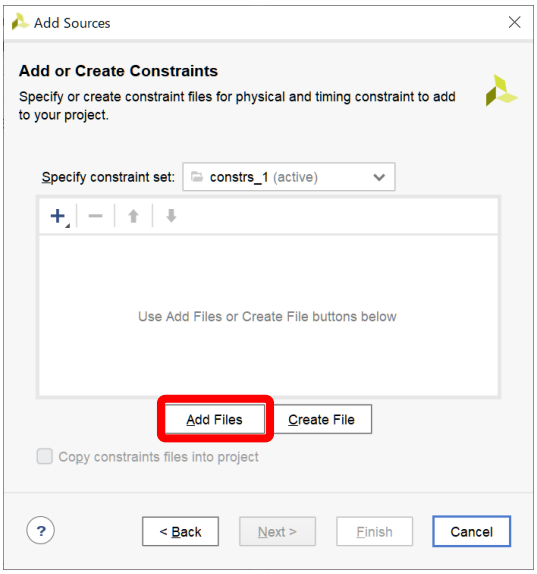


図 11.31　FPGA への実装の手順 1

3)　「Add or create constraints」を選択する（**図 11.32**）。

図 11.32　FPGA への実装の手順 2

4)　「Add Files」を選択し，手順 1 で保存した「Arty-A7-35-Master.xdc」を指定する（**図 11.33**）。



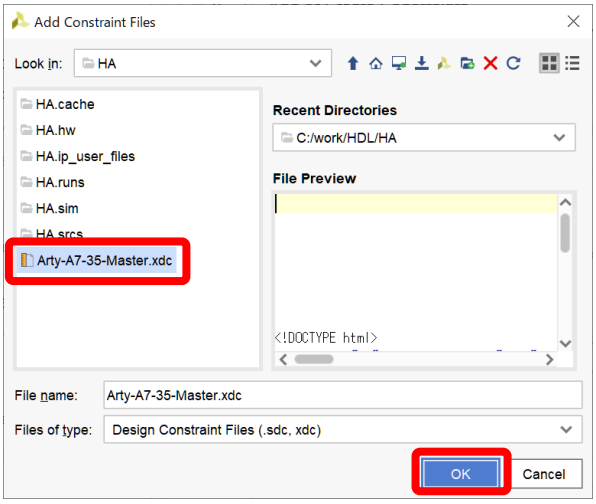


図 11.33　FPGA への実装の手順 3

5)　「Finish」を選択する（**図 11.34**）。

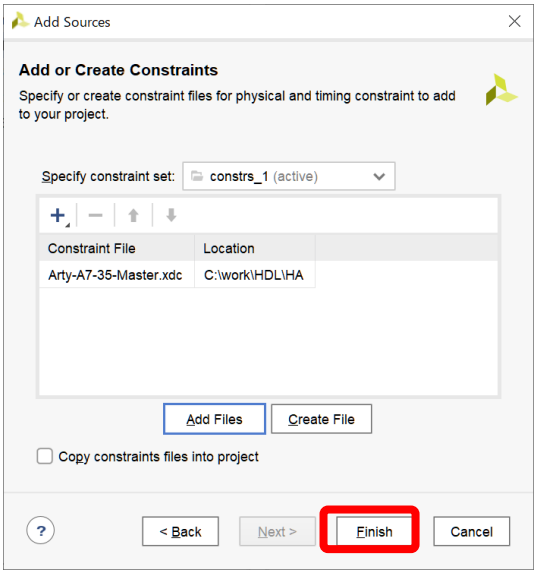


図 11.34　FPGA への実装の手順 4

6)　「Constraints」→「constrs_1」をそれぞれダブルクリックする。その後，「Arty-A7-35-Master.xdc」をダブルクリックする。**図 11.35** のように右のエリアに「Arty-A7-35-Master.xdc」タブが追加され，内容が表示される。

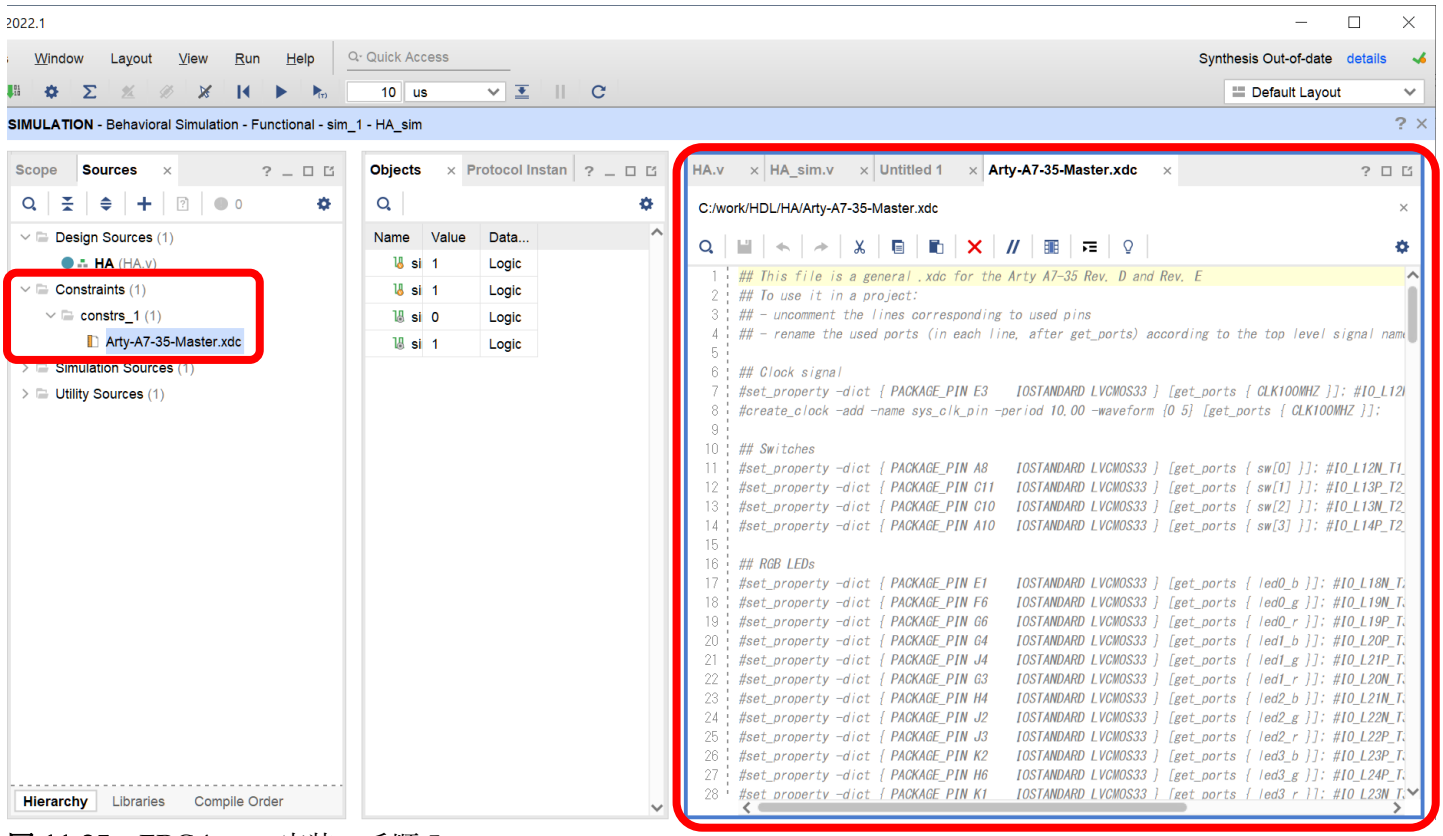


図 11.35　FPGA への実装の手順 5

7) Constraints ファイル「Arty-A7-35-Master.xdc」の抜粋を**図 11.36** に示す。Constraints ファイルは，行頭に「#」を付記することでコメントアウトすることができる。「led[0]」等の「get_ports」後の括弧内が回路の Verilog HDL ファイルで宣言した入出力ポートと対応する。また，開発対象キット Arty の場合，**図 11.37** のように評価ボードのシルク印刷としておおよそ対応する部品が記されている。

表 11.2 に示すように各信号名の対応を把握した上で，回路の信号名を割り当てる。例えば，作成した回路の Verilog-HDL ファイルの信号名「A」を，評価ボード上の押しボタンスイッチ「BTN0」に割り当てる場合は，Constraints ファイル中の「btn[0]」を「A」に変更することで割り当てることができる。これらを適用すると，図 11.38 のとおりとなるので，該当する行のみを変更する。

```
## LEDs
#set_property -dict { PACKAGE_PIN H5    IOSTANDARD LVCMOS33 } [get_ports
{ led[4] }]; #IO_L24N_T3_35 Sch=led[4]
#set_property -dict { PACKAGE_PIN J5    IOSTANDARD LVCMOS33 } [get_ports
{ led[5] }]; #IO_25_35 Sch=led[5]

## Buttons
#set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 } [get_ports
{ btn[0] }]; #IO_L6N_T0_VREF_16 Sch=btn[0]
#set_property -dict { PACKAGE_PIN C9    IOSTANDARD LVCMOS33 } [get_ports
{ btn[1] }]; #IO_L11P_T1_SRCC_16 Sch=btn[1]
```

図 11.36　Constraints ファイル「Arty-A7-35-Master.xdc」抜粋

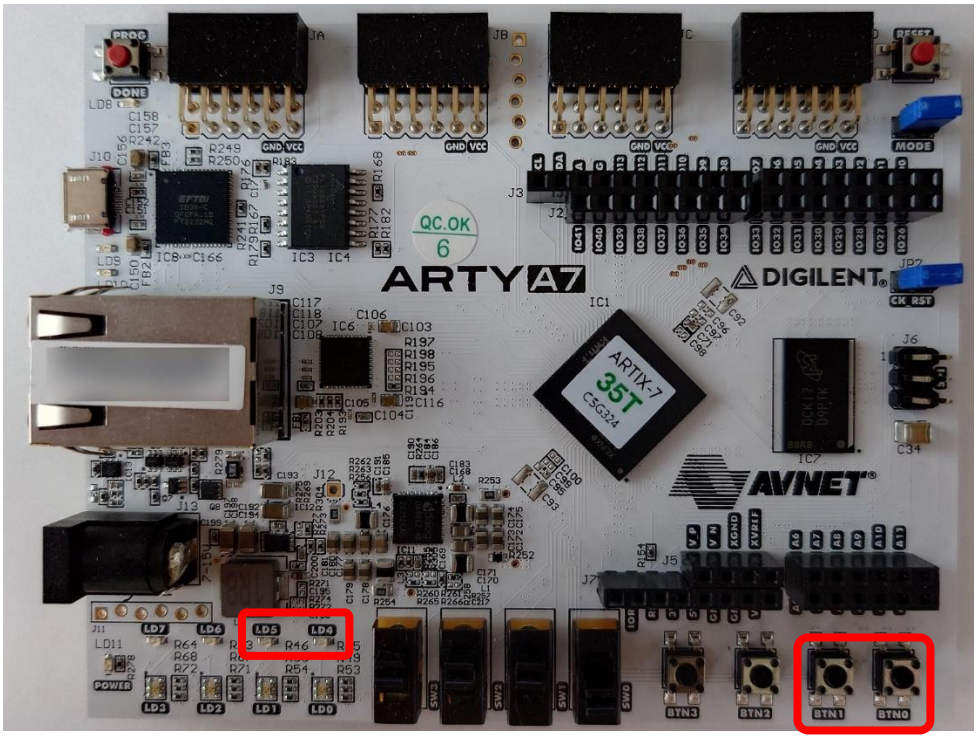


図 11.37　開発対象キット Arty

表 11.2　信号の割り当て

Constraints ファイル	評価ボードのシルク印刷	HA.v	HA_sim.v
led[0]	LD4	S	simS
led[1]	LD5	C_out	simC_out
btn[0]	BTN0	A	simA
btn[1]	BTN1	B	simB

```
## LEDs
set_property -dict { PACKAGE_PIN H5    IOSTANDARD LVCMOS33 } [get_ports
{ S }]; #IO_L24N_T3_35 Sch=led[4]
set_property -dict { PACKAGE_PIN J5    IOSTANDARD LVCMOS33 } [get_ports
{ C_out }]; #IO_25_35 Sch=led[5]

## Buttons
set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 } [get_ports
{ A }]; #IO_L6N_T0_VREF_16 Sch=btn[0]
set_property -dict { PACKAGE_PIN C9    IOSTANDARD LVCMOS33 } [get_ports
{ B }]; #IO_L11P_T1_SRCC_16 Sch=btn[1]
```

図 11.38　HA に関する Constraints ファイルの適用例

8)　Constraints ファイルの変更を，フロッピーディスクのボタンをクリックし，保存する（**図 11.39**）。

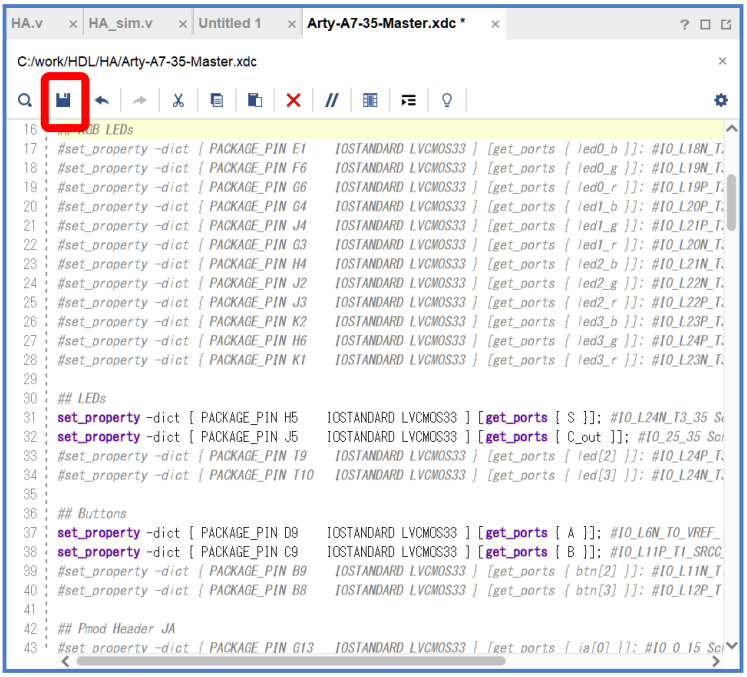


図 11.39　FPGA への実装の手順 6

9) FPGA 上における回路の配置配線を行うために，「Run Implementation」を実行する。また，論理合成と同様に「Number of jobs:」の数を適宜調整する（**図 11.40**）。

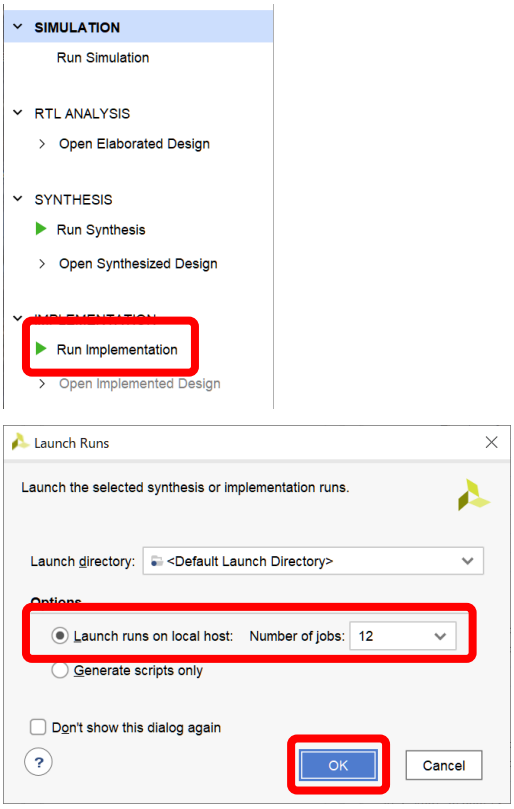


図 11.40　FPGA への実装の手順 7

10) 配置配線が完了すると以下が表示されるので，「Cancel」を選択する（**図 11.41**）。

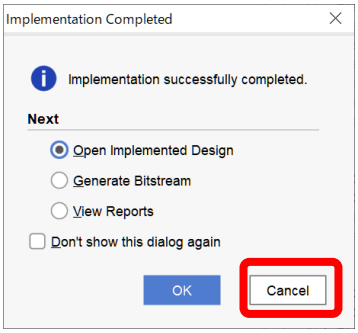


図 11.41　FPGA への実装の手順 8

11) 「Open Implemented Design」を選択する（**図 11.42**）。

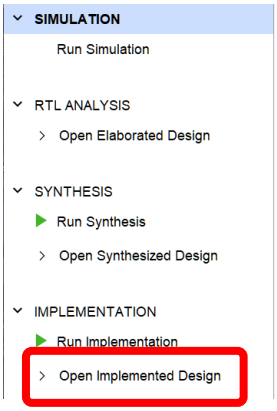


図 11.42　FPGA への実装の手順 9

12) 上部のメニューにおける「Layout」の「I/O Planning」を選択する。その後，「Scalar ports」を選択し，設定表を表示させる。この際，「Fixed」の列にチェックマークが入っていることと，「I/O Std」列に「default」の語句が含まれる場合は「LV」に設定し，FPGA の IO ポートの電圧を LED 駆動するために 3.3V を指定するべく，「Vcco」が「3.300」であることを確認する（**図 11.43**）。

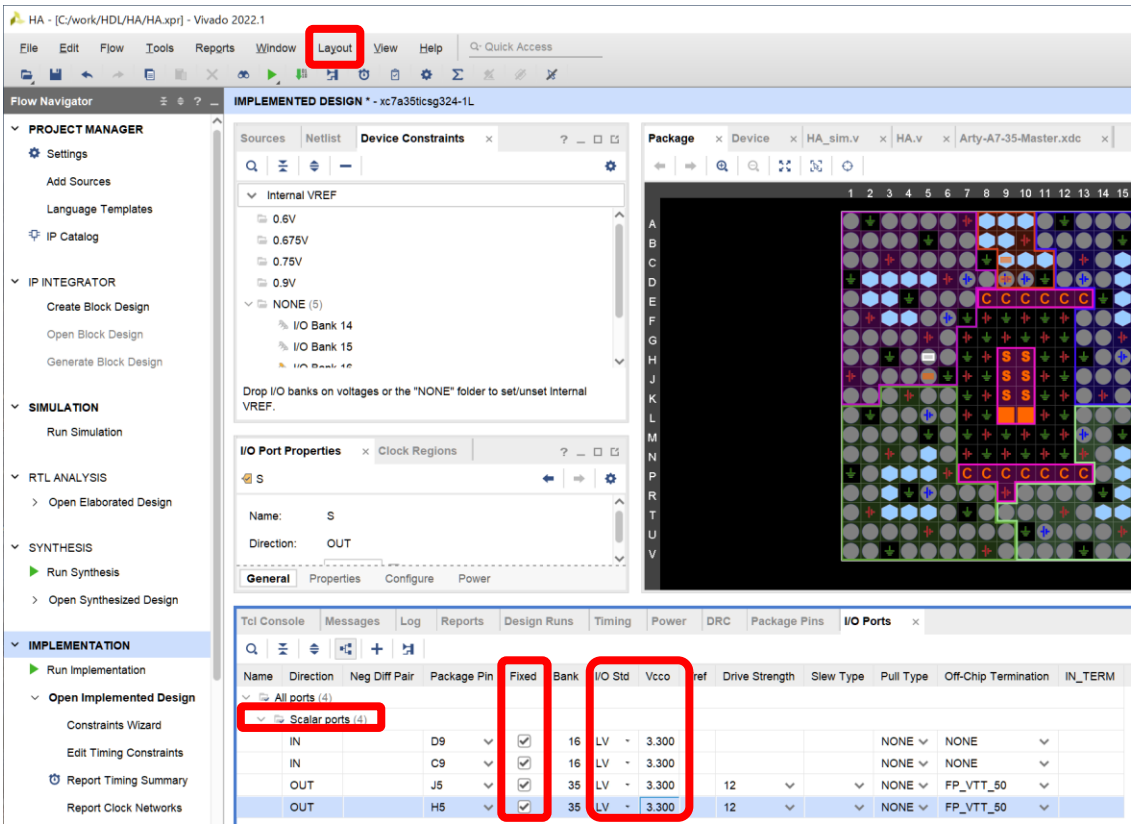
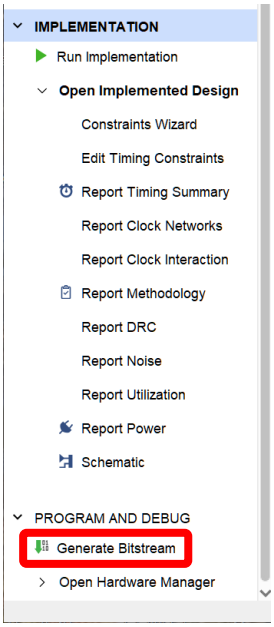


図 11.43　FPGA への実装の手順 10

13) FPGA に書き込む回路構成情報である Bitstream データを生成するために，「Generate Bitstream」を選択する。また，「Save Project」ウインドウが表示された場合は，「Save」を選択する。その後，「Save Constraints」ウインドウが表示された場合は，「Update」を選択し，「Select an existing file」にチェックが入っていることを確認し，「OK」を選択する。さらに，「Implementation is Out-of-date」が表示された場合は，「Yes」を選択する。また，論理合成と同様に「Number of jobs:」の数を適宜調整する（**図 11.44**）。



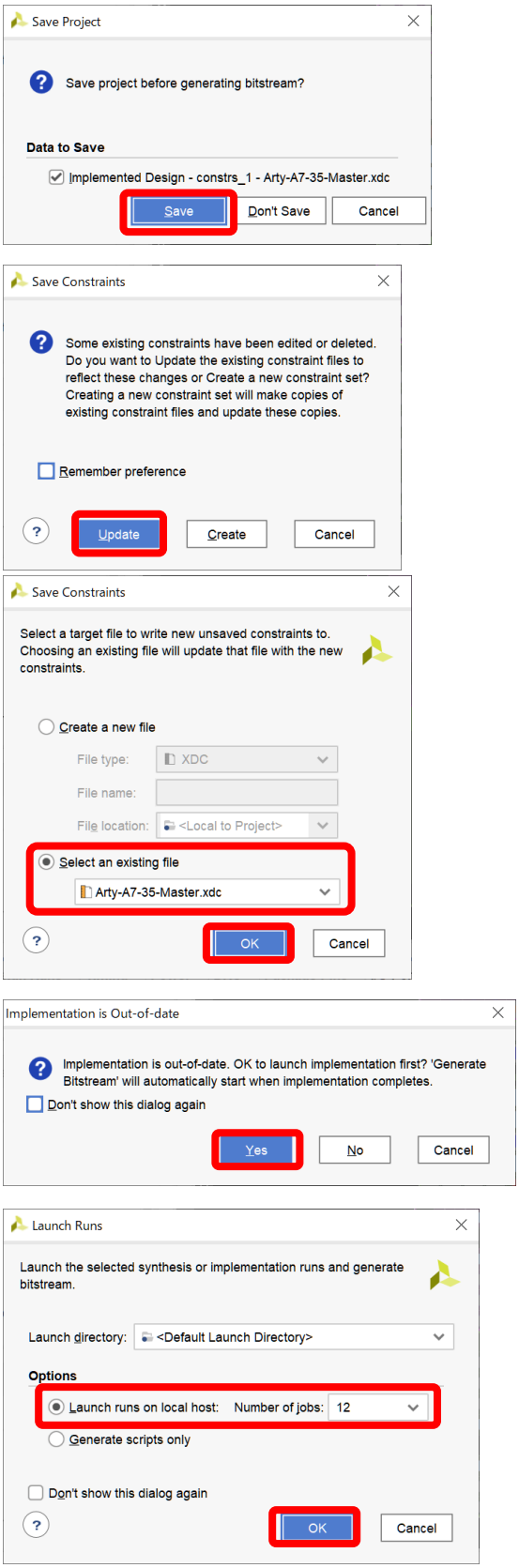


図 11.44　FPGA への実装の手順 11

14）Bitstream データが生成されると以下のウインドウが表示されるので，「Cancel」を選択する（**図 11.45**）。

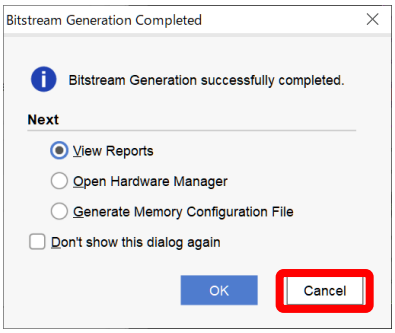


図 11.45　FPGA への実装の手順 12

15）評価ボードとコンピュータを USB で接続する（**図 11.46**）。

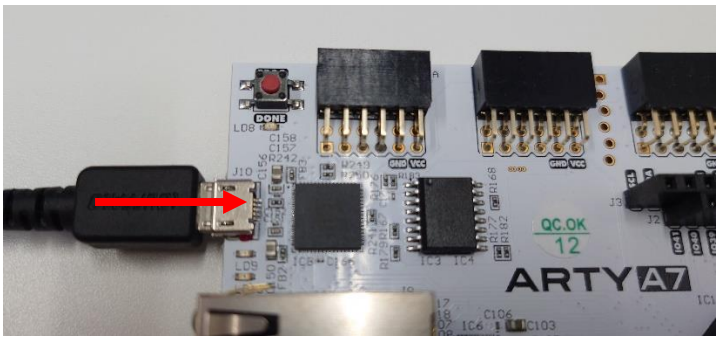


図 11.46　評価対象キットとの USB 接続

16）「Open Target」を選択し，「Auto Connect」を選択する（**図 11.47**）。

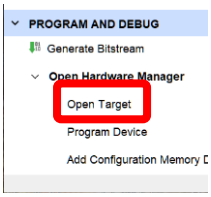


図 11.47　FPGA への実装の手順 13

17）「Program Device」を選択し，対象の FPGA を選択する。この場合は，「xc7a35t_0」を選択する（**図 11.48**）。

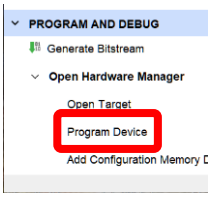


図 11.48　FPGA への実装の手順 14

18) 「Bitstream file」を選択する。書き込む Bitstream ファイルは，プロジェクトが保存されているフォルダ内の「(プロジェクト名) .runs」フォルダ内の「impl_1」フォルダ内に「(プロジェクト名) .bit」として保存されているので，「…」を選択し，該当するファイルを選択する（**図 11.49**）。

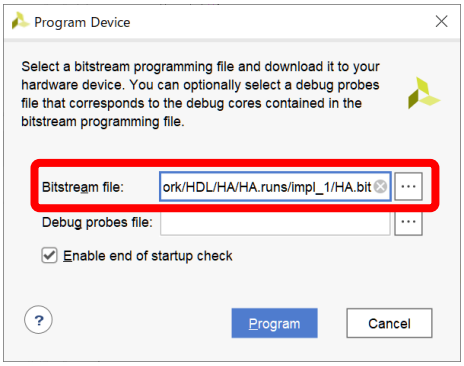


図 11.49　FPGA への実装の手順 15

19) 「Program」を選択する（**図 11.50**）。

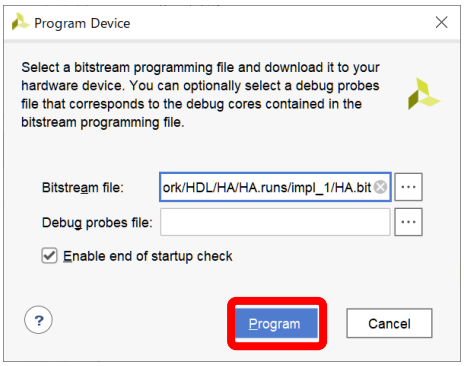


図 11.50　FPGA への実装の手順 16

20) 以下の書き込みの進捗を示すプログレスバーが表示され，100%に到達し表示が消えた段階で書き込みが完了となる。この状態で動作を実機にて確認することができる（**図 11.51**）。

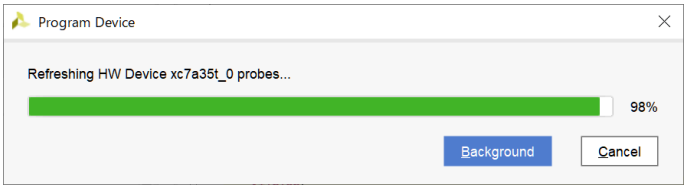


図 11.51　書き込みのプログレスバー

21) 図 11.37 および表 11.2 に示されるように，評価ボードに記されたシルク印刷と設計した回路の信号名はそれぞれ BTN0=A，BTN1=B，LD4=S，LD5=C_out に対応しているので，実際に BTN0 および BTN1 を押下し，LD4 や LD5 が期待どおりに点灯するか確認する。

演習問題

1 四つのスライドスイッチの入力値を四つの LED に出力する回路を設計し，シミュレーションと実機で動作確認しなさい。

2 つぎの真理値表に示すデコーダ回路を設計し，シミュレーションと実機で動作確認しなさい。なお，入力 X を押しボタンスイッチまたはスライドスイッチとし，出力 Y を LED に割り当てなさい。

X1	X0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

3 1Hz の速度でカウントアップし，スタート／ストップを制御可能な 4 ビットカウンタを設計し，シミュレーションと実機で動作確認しなさい。なお，スタート／ストップの制御は押しボタンスイッチを使用し，カウンタの出力を LED に割り当てなさい。