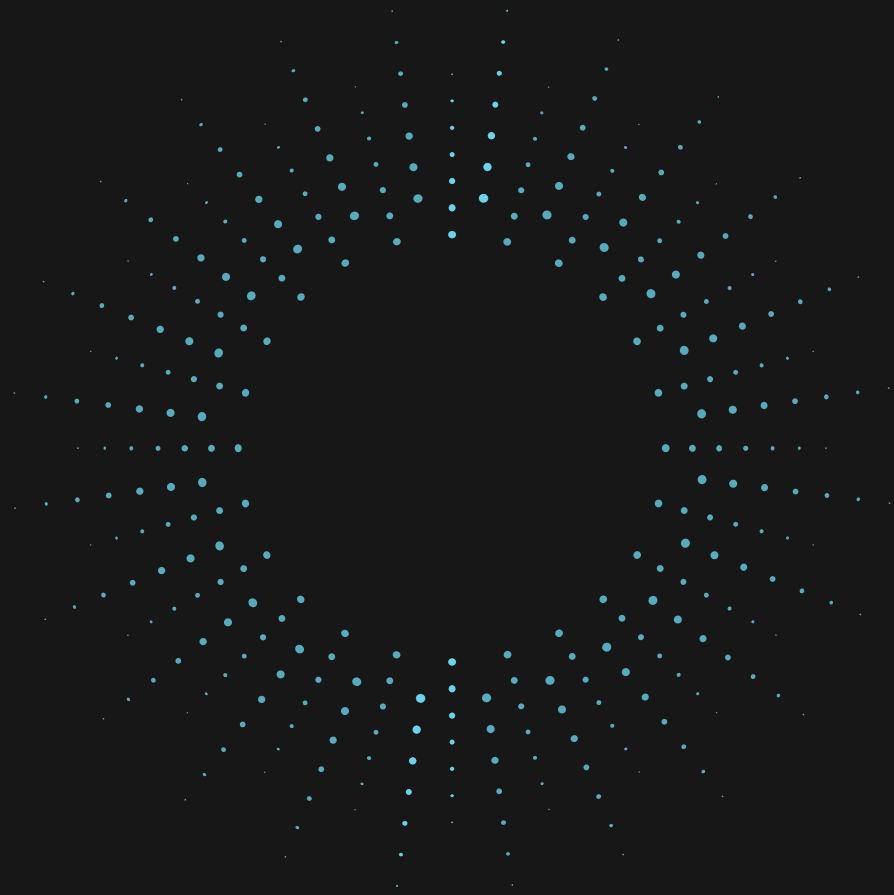
KD240 PWM LED

Field Application Engineer
Adaptive and Embedded Computing Group (AECG)

Revision History

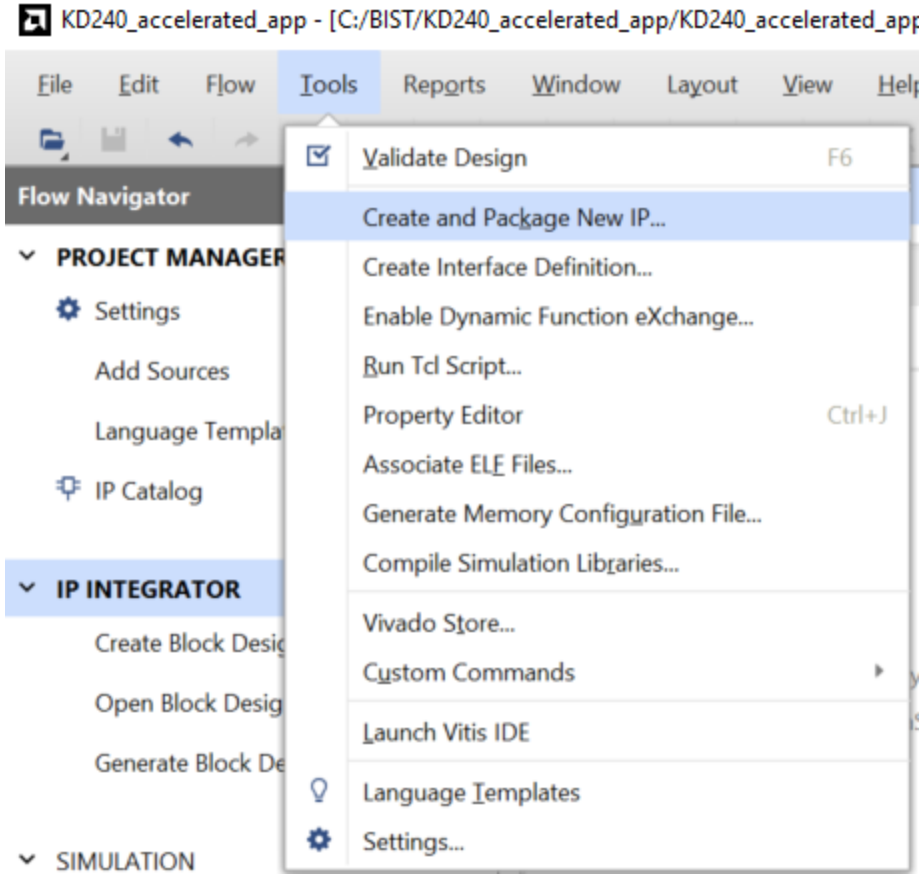
Date	Version	Description
01/26/24	1.1	Add new chapter – Using PYNQ overlay to light LED.
01/16/24	1.0	Initial version for flow introduction.

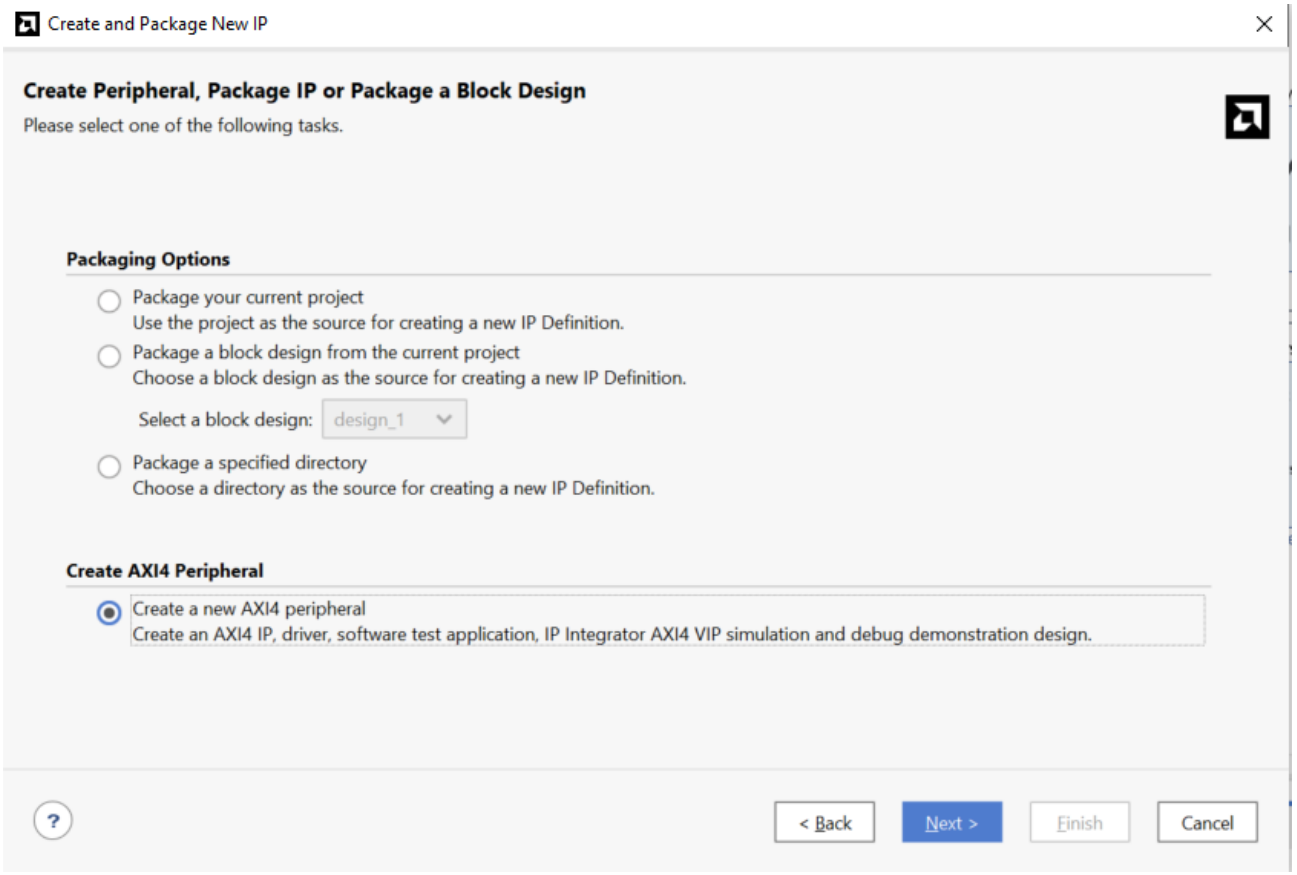
Vivado 2023.2.1 Part



KD240 PWM LED

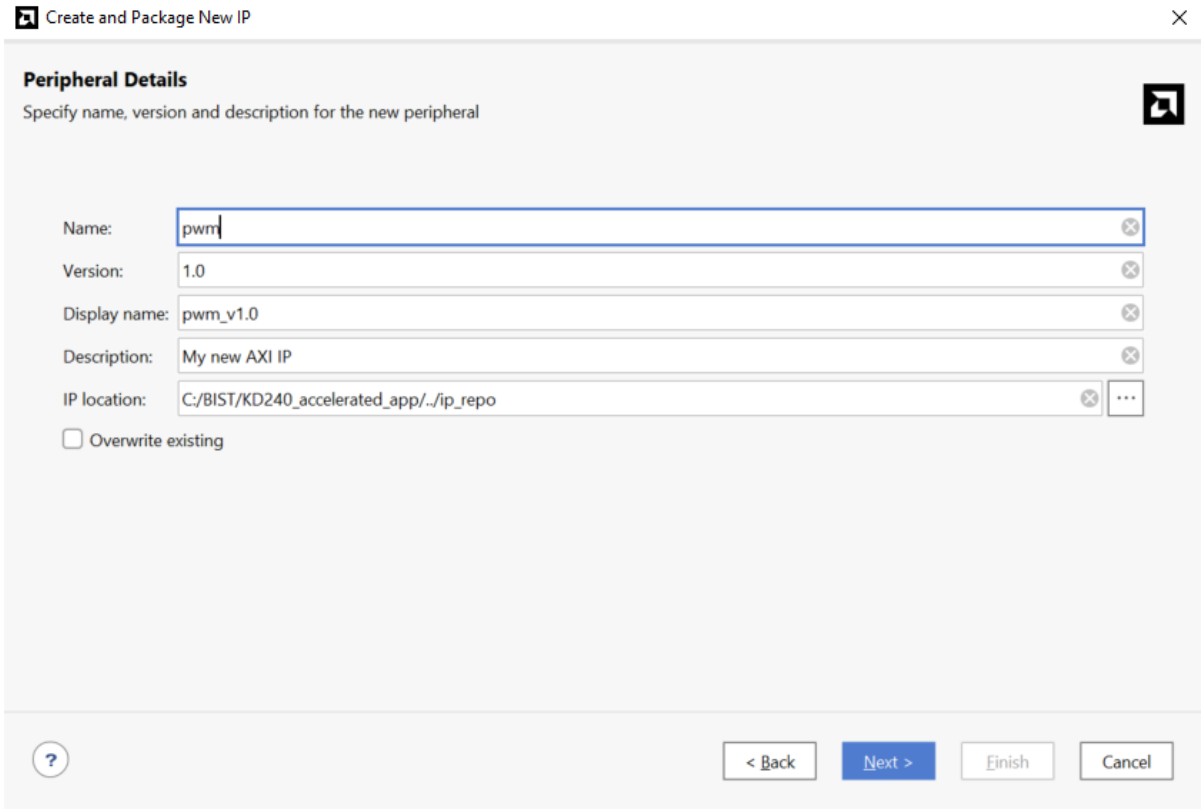
1. 自定義 PWM IP

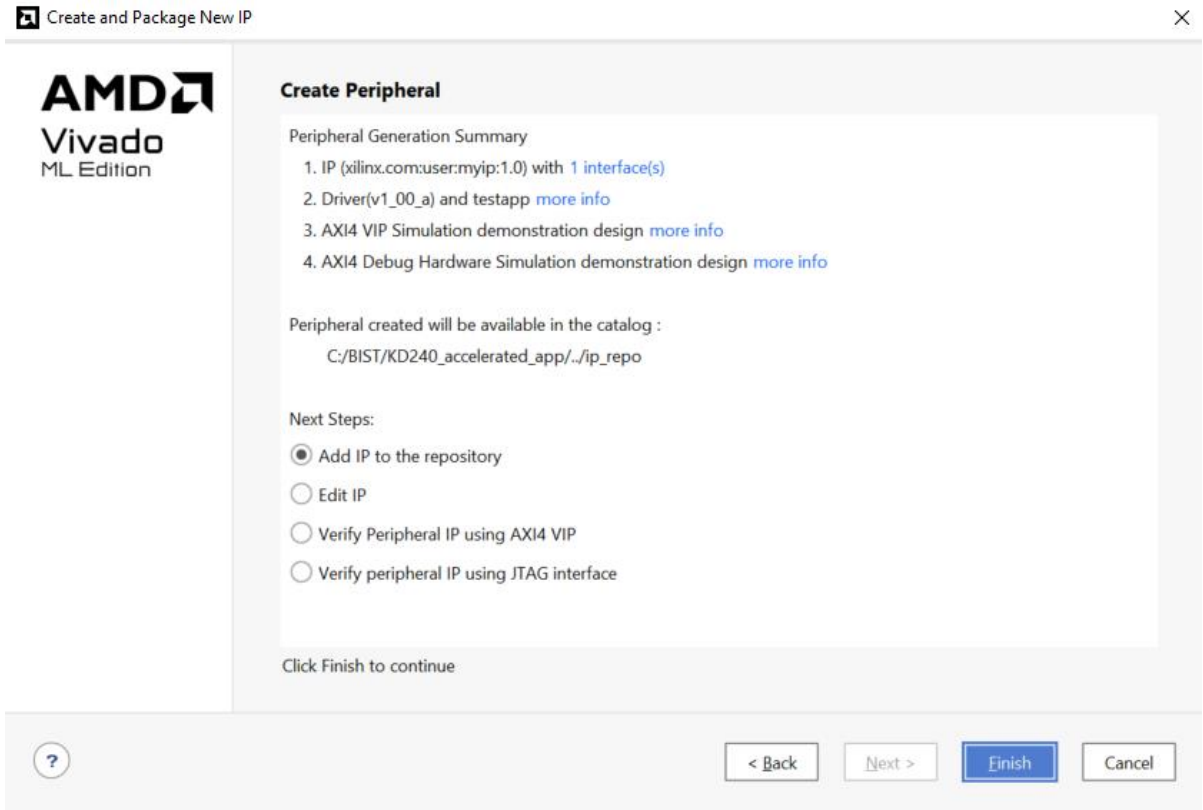




KD240 PWM LED

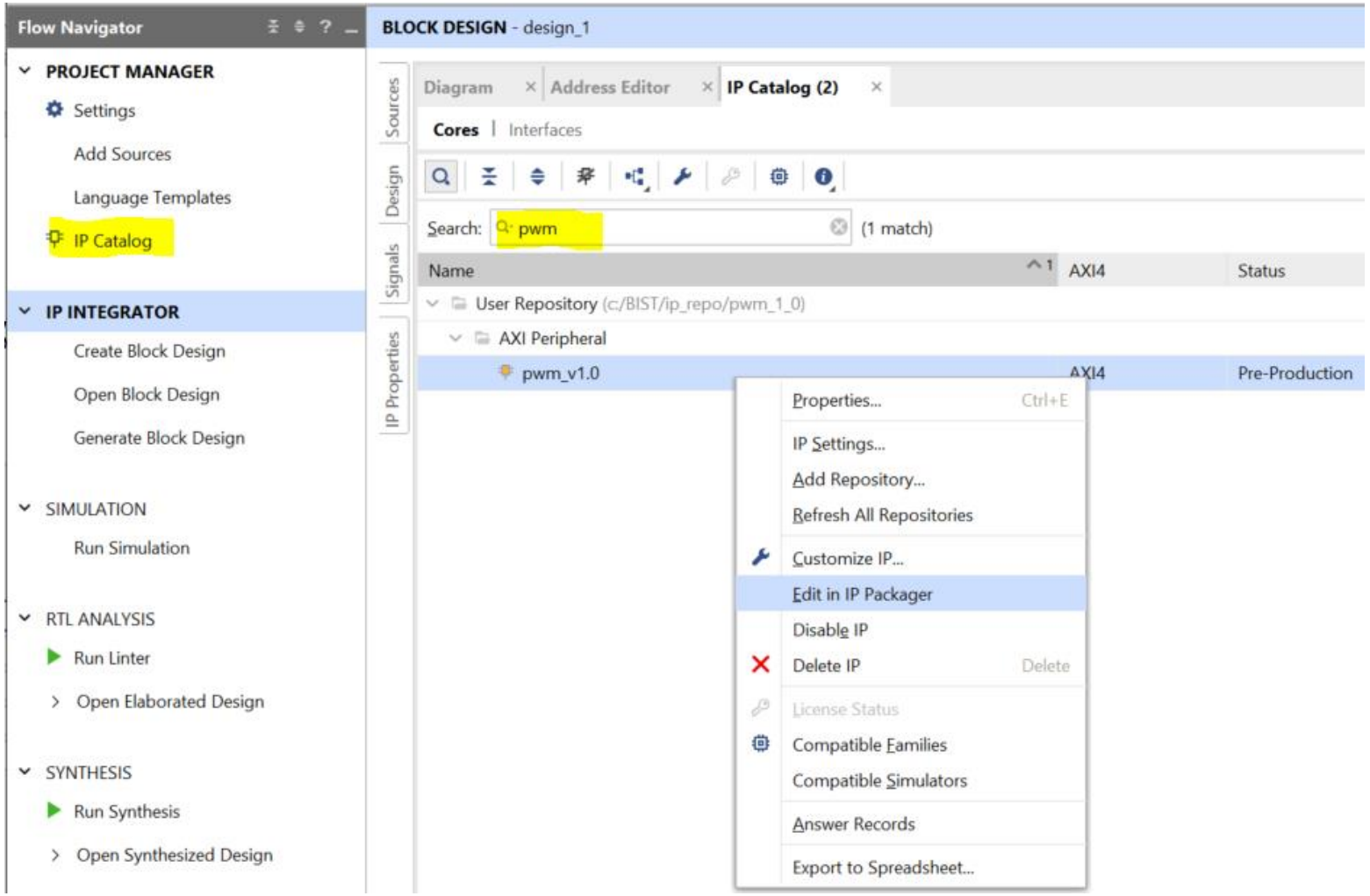
1. 自定義 PWM IP - 前面設定都直接點選 Next 就好





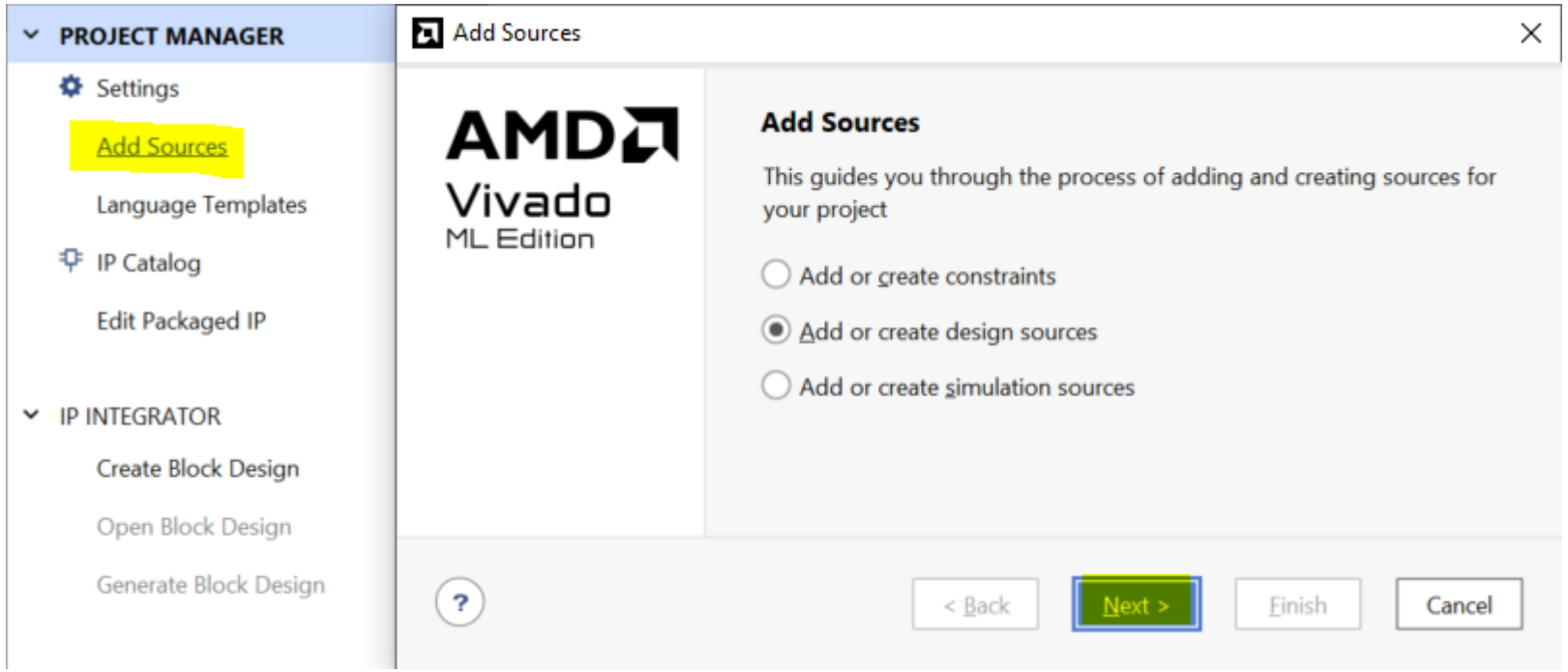
KD240 PWM LED

2. 設定 PWM IP - 會開一個暫時的 project 來設定 IP



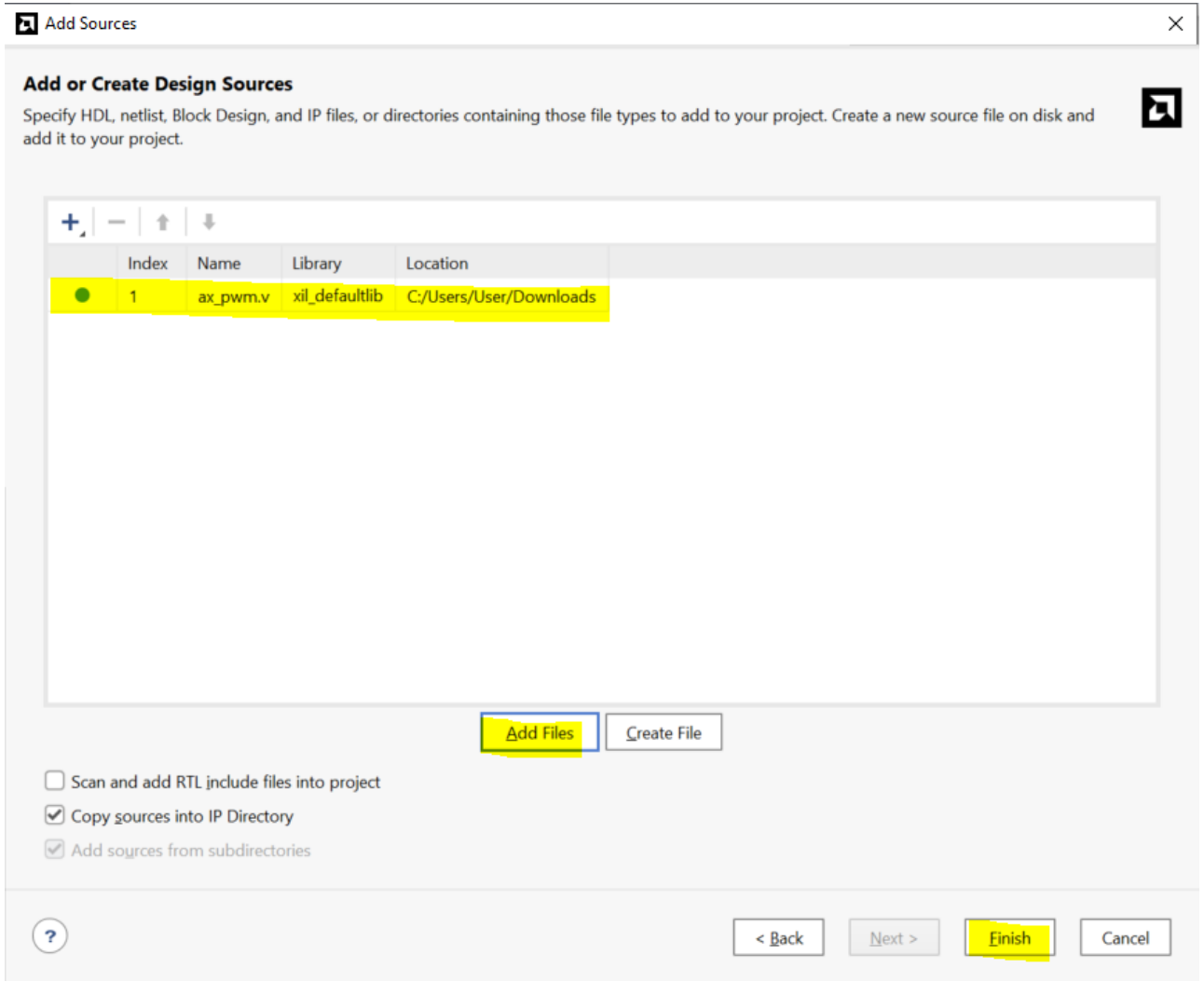
KD240 PWM LED

2. 設定 PWM IP - 加入 PWM RTL Code



KD240 PWM LED

2. 設定 PWM IP - 加入 PWM RTL Code



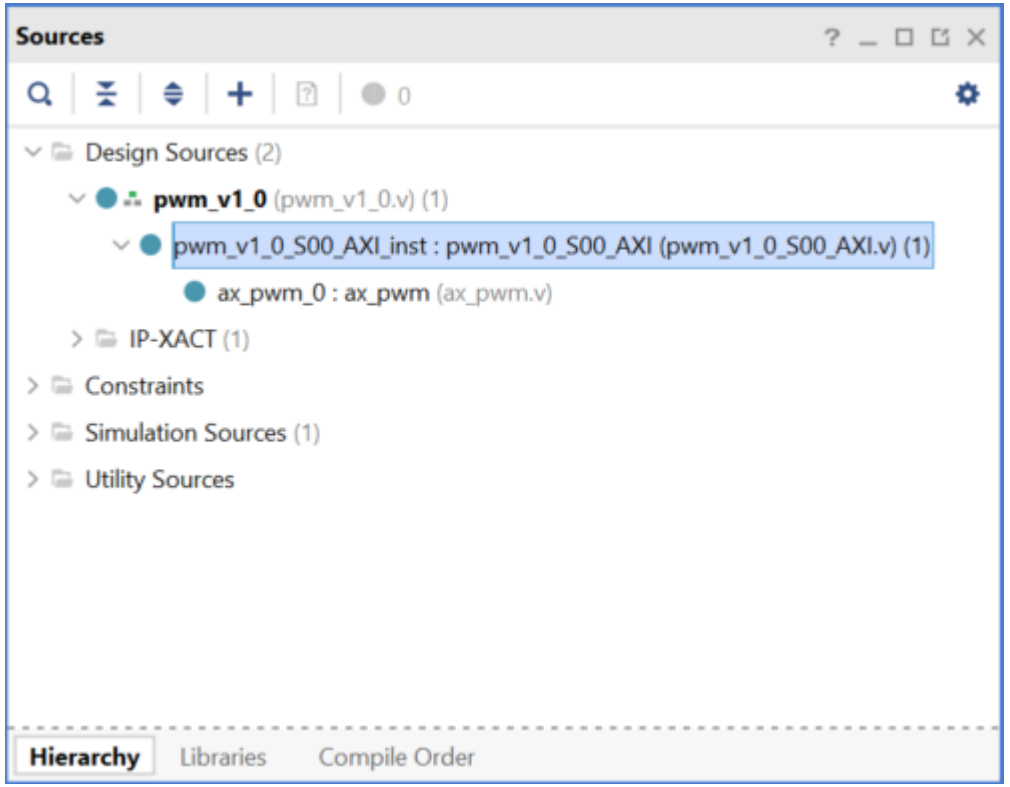
KD240 PWM LED

PWM RTL Code Review

ax_pwm.v

KD240 PWM LED

2. 設定 PWM IP - 修改 AXI Top RTL Code 來引入剛剛的 PWM RTL Code



KD240 PWM LED

2. 設定 PWM IP - 修改 AXI Top RTL Code 來引入剛剛的 PWM RTL Code

Project Summary | Package IP - pwm | ax_pwm.v | pwm_v1_0_S00_AXI.v

c:/BIST/ip_repo/pwm_1_0/hdl/pwm_v1_0_S00_AXI.v

```
1
2   `timescale 1 ns / 1 ps
3
4       module pwm_v1_0_S00_AXI #
5       (
6           // Users to add parameters here
7
8           // User parameters ends
9           // Do not modify the parameters beyond this line
10
11          // Width of S_AXI data bus
12          parameter integer C_S_AXI_DATA_WIDTH    = 32,
13          // Width of S_AXI address bus
14          parameter integer C_S_AXI_ADDR_WIDTH    = 4
15      )
16      (
17          // Users to add ports here
18          output pwm,
19          // User ports ends
20          // Do not modify the ports beyond this line
21
22          // Global Clock Signal
23          input wire  S_AXI_ACLK,
24          // Global Reset Signal. This Signal is Active LOW
25          input wire  S_AXI_ARESETN,
26          // Write address (issued by master, acceped by Slave)
27          input wire [C_S_AXI_ADDR_WIDTH-1 : 0] S_AXI_AWADDR,
28          // Write channel Protection type. This signal indicates the
29              // privilege and security level of the transaction, and whether
30              // the transaction is a data access or an instruction access.
31          input wire [2 : 0] S_AXI_AWPROT,
32          // Write address valid. This signal indicates that the master signaling
```

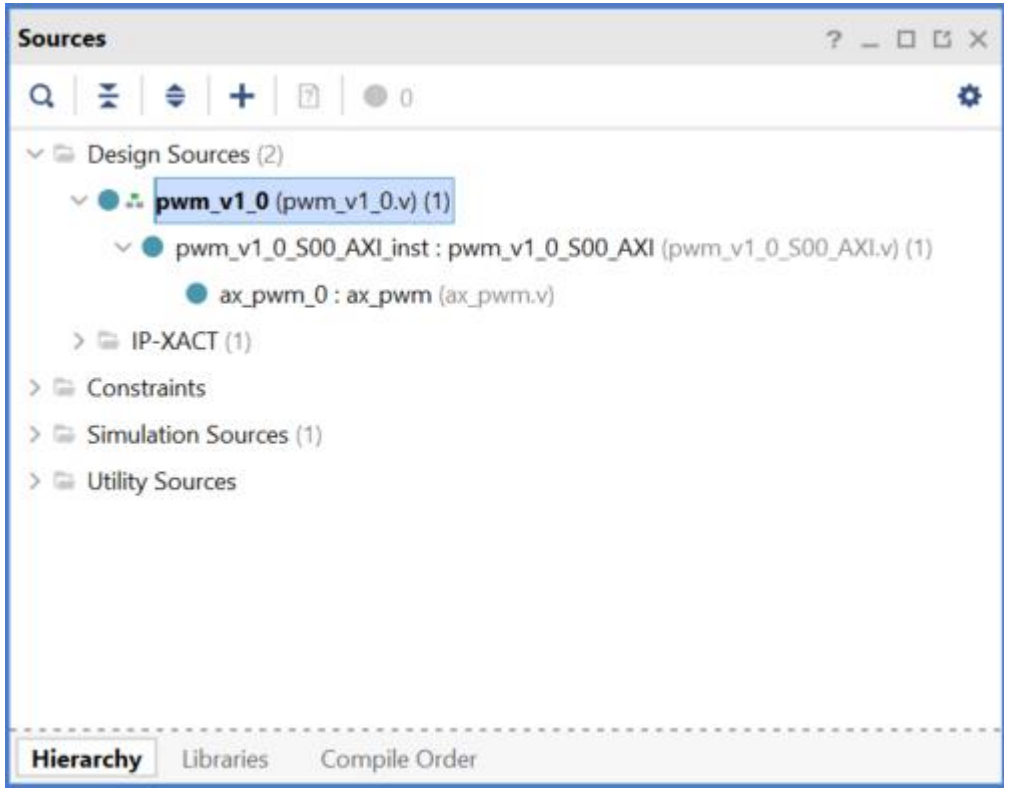
Project Summary | Package IP - pwm | ax_pwm.v | pwm_v1_0_S00_AXI.v

c:/BIST/ip_repo/pwm_1_0/hdl/pwm_v1_0_S00_AXI.v

```
380
381     // Output register or memory read data
382     always @( posedge S_AXI_ACLK )
383     begin
384       if ( S_AXI_ARESETN == 1'b0 )
385         begin
386           axi_rdata  <= 0;
387         end
388       else
389         begin
390           // When there is a valid read address (S_AXI_ARVALID) with
391           // acceptance of read address by the slave (axi_arready),
392           // output the read dada
393           if (slv_reg_rden)
394             begin
395               axi_rdata <= reg_data_out;     // register read data
396             end
397         end
398     end
399
400     // Add user logic here
401     ax_pwm ax_pwm_0(
402         .clk(S_AXI_ACLK),
403         .rst(S_AXI_ARESETN),
404         .period(slv_reg0),
405         .duty(slv_reg1),
406         .pwm_out(pwm)
407     );
408     // User logic ends
409
410     endmodule
411
```

KD240 PWM LED

2. 設定 PWM IP - 修改 AXI Top RTL Code 來引入剛剛的 PWM RTL Code



KD240 PWM LED

2. 設定 PWM IP - 修改 AXI Top RTL Code 來引入剛剛的 PWM RTL Code

Project Summary | Package IP - pwm | ax_pwm.v | pwm_v1_0_S00_AXI.v | pwm_v1_0.v

c:/BIST/ip_repo/pwm_1_0/hdl/pwm_v1_0.v

```
1
2   `timescale 1 ns / 1 ps
3
4       module pwm_v1_0 #
5       (
6           // Users to add parameters here
7
8           // User parameters ends
9           // Do not modify the parameters beyond this line
10
11
12          // Parameters of Axi Slave Bus Interface S00_AXI
13          parameter integer C_S00_AXI_DATA_WIDTH  = 32,
14          parameter integer C_S00_AXI_ADDR_WIDTH  = 4
15      )
16      (
17          // Users to add ports here
18          output pwm,
19          // User ports ends
20          // Do not modify the ports beyond this line
21
22
23          // Ports of Axi Slave Bus Interface S00_AXI
24          input wire  s00_axi_aclk,
25          input wire  s00_axi_aresetn,
26          input wire [C_S00_AXI_ADDR_WIDTH-1 : 0] s00_axi_awaddr,
27          input wire [2 : 0] s00_axi_awprot,
28          input wire  s00_axi_awvalid,
29          output wire  s00_axi_awready,
30          input wire [C_S00_AXI_DATA_WIDTH-1 : 0] s00_axi_wdata,
31          input wire [(C_S00_AXI_DATA_WIDTH/8)-1 : 0] s00_axi_wstrb,
32          input wire  s00_axi_wvalid,
```

Project Summary | Package IP - pwm | ax_pwm.v | pwm_v1_0_S00_AXI.v | pwm_v1_0.v

c:/BIST/ip_repo/pwm_1_0/hdl/pwm_v1_0.v

```
43          output wire  s00_axi_rvalid,
44          input wire  s00_axi_rready
45      );
46  // Instantiation of Axi Bus Interface S00_AXI
47      pwm_v1_0_S00_AXI # (
48          .C_S_AXI_DATA_WIDTH(C_S00_AXI_DATA_WIDTH),
49          .C_S_AXI_ADDR_WIDTH(C_S00_AXI_ADDR_WIDTH)
50      ) pwm_v1_0_S00_AXI_inst (
51          .pwm(pwm),
52          .S_AXI_ACLK(s00_axi_aclk),
53          .S_AXI_ARESETN(s00_axi_aresetn),
54          .S_AXI_AWADDR(s00_axi_awaddr),
55          .S_AXI_AWPROT(s00_axi_awprot),
56          .S_AXI_AWVALID(s00_axi_awvalid),
57          .S_AXI_AWREADY(s00_axi_awready),
58          .S_AXI_WDATA(s00_axi_wdata),
59          .S_AXI_WSTRB(s00_axi_wstrb),
60          .S_AXI_WVALID(s00_axi_wvalid),
61          .S_AXI_WREADY(s00_axi_wready),
62          .S_AXI_BRESP(s00_axi_bresp),
63          .S_AXI_BVALID(s00_axi_bvalid),
64          .S_AXI_BREADY(s00_axi_bready),
65          .S_AXI_ARADDR(s00_axi_araddr),
66          .S_AXI_ARPROT(s00_axi_arprot),
67          .S_AXI_ARVALID(s00_axi_arvalid),
68          .S_AXI_ARREADY(s00_axi_arready),
69          .S_AXI_RDATA(s00_axi_rdata),
70          .S_AXI_RRESP(s00_axi_rresp),
71          .S_AXI_RVALID(s00_axi_rvalid),
72          .S_AXI_RREADY(s00_axi_rready)
73      );
74
```

KD240 PWM LED

2. 設定 PWM IP - 須注意 AXI Reset 是為 0 時 initial register value，這點要跟 PWM Code 一致

Project Summary | Package IP - pwm | ax_pwm.v | pwm_v1_0_S00_AXI.v | pwm_v1_0.v

c:/BIST/ip_repo/pwm_1_0/hdl/pwm_v1_0_S00_AXI.v

```
128     // axi_awready is asserted for one S_AXI_ACLK clock cycle when both
129     // S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
130     // de-asserted when reset is low.
131
132     always @( posedge S_AXI_ACLK )
133     begin
134       if ( S_AXI_ARESETN == 1'b0 )
135         begin
136           axi_awready <= 1'b0;
137           aw_en <= 1'b1;
138         end
139       else
140         begin
141           if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID && aw_en)
142             begin
143               // slave is ready to accept write address when
144               // there is a valid write address and write data
145               // on the write address and data bus. This design
146               // expects no outstanding transactions.
147               axi_awready <= 1'b1;
148               aw_en <= 1'b0;
149             end
150           else if (S_AXI_BREADY && axi_bvalid)
151               begin
152                 aw_en <= 1'b1;
153                 axi_awready <= 1'b0;
154               end
155         else
156           begin
157             axi_awready <= 1'b0;
158           end
159       end
```

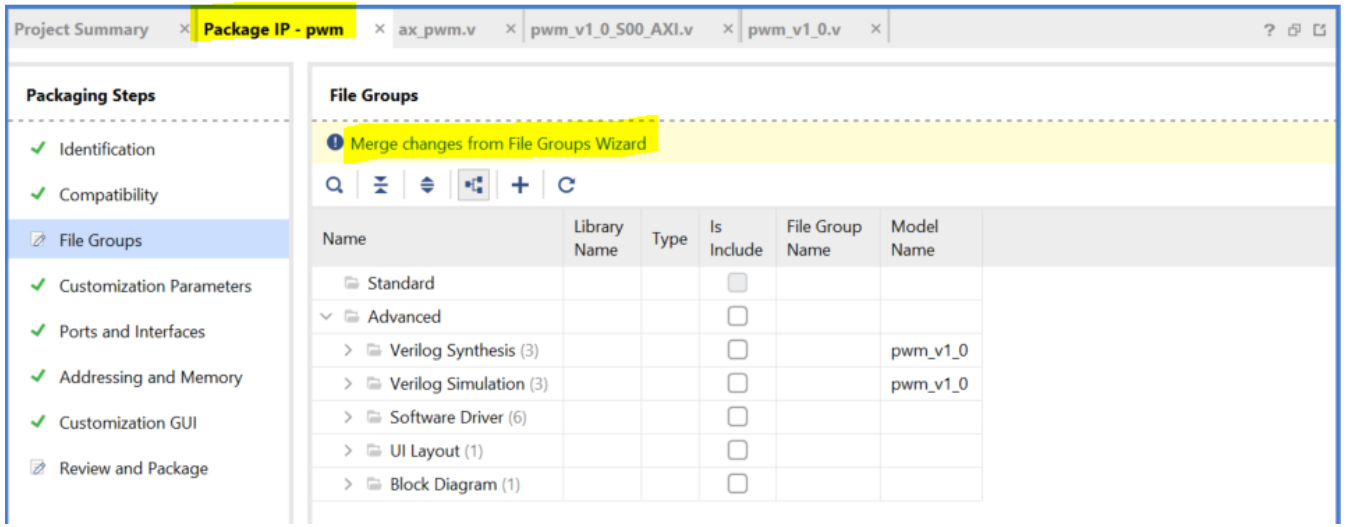
Project Summary | Package IP - pwm | ax_pwm.v | pwm_v1_0_S00_AXI.v | pwm_v1_0.v

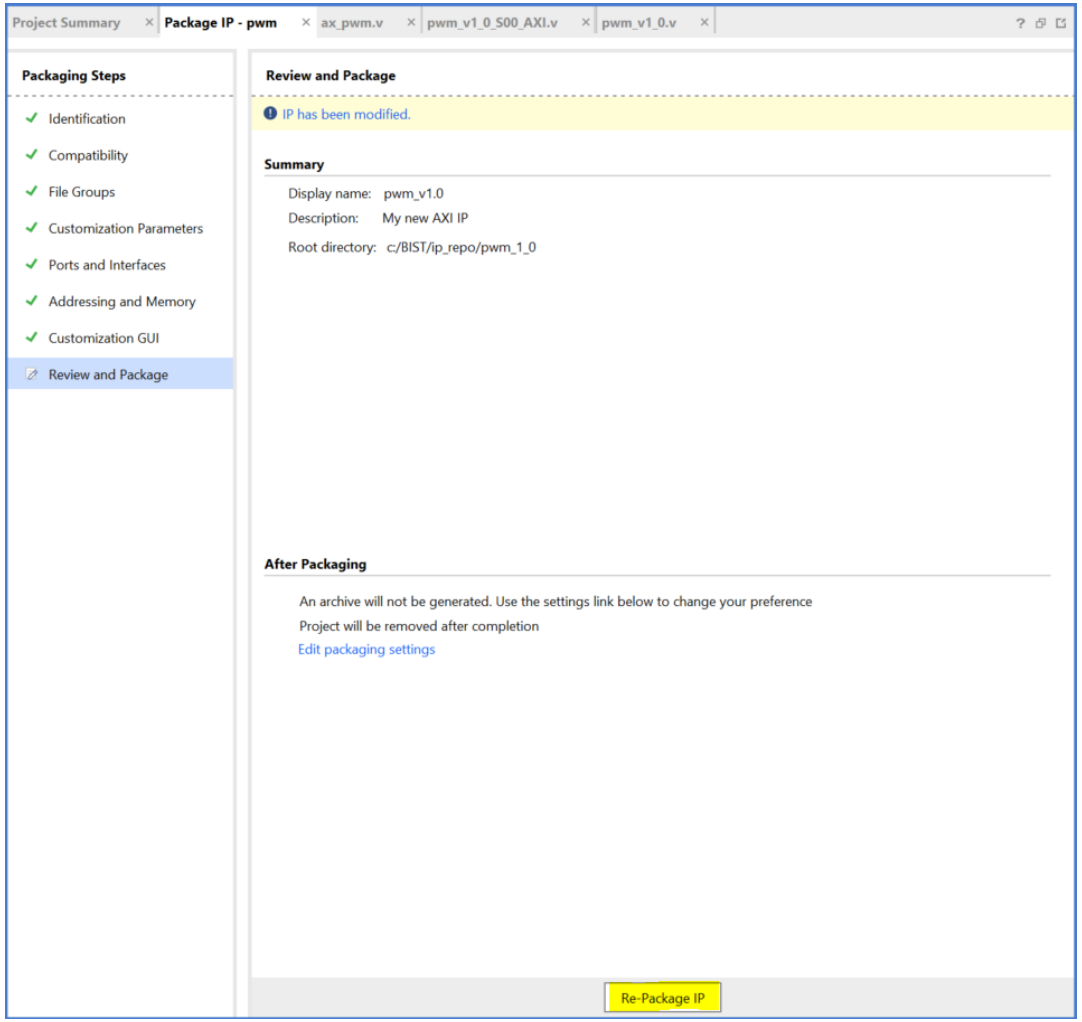
c:/BIST/ip_repo/pwm_1_0/src/ax_pwm.v

```
43      output          pwm_out
44      );
45
46  reg[N - 1:0] period_r;
47  reg[N - 1:0] duty_r;
48  reg[N - 1:0] period_cnt;
49  reg pwm_r;
50  assign pwm_out = pwm_r;
51  always@(posedge clk)
52  begin
53      if(rst==0)
54      begin
55          period_r <= { N {1'b0} };
56          duty_r <= { N {1'b0} };
57      end
58      else
59      begin
60          period_r <= period;
61          duty_r   <= duty;
62      end
63  end
64
65  always@(posedge clk)
66  begin
67      if(rst==0)
68          period_cnt <= { N {1'b0} };
69      else
70          period_cnt <= period_cnt + period_r;
71  end
72
73  always@(posedge clk)
74  begin
```

KD240 PWM LED

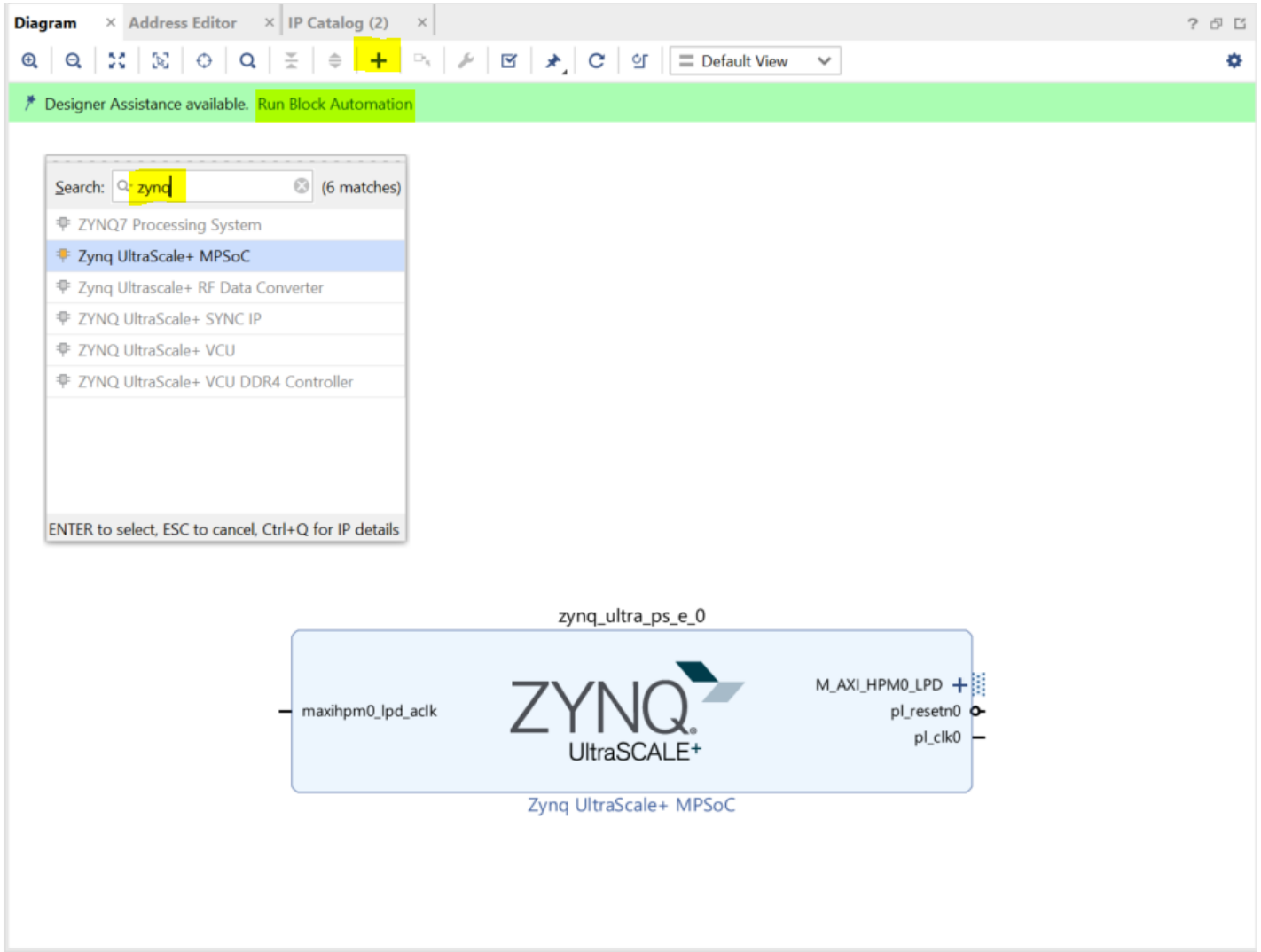
2. 設定 PWM IP - Merge Changes and Re-Package IP





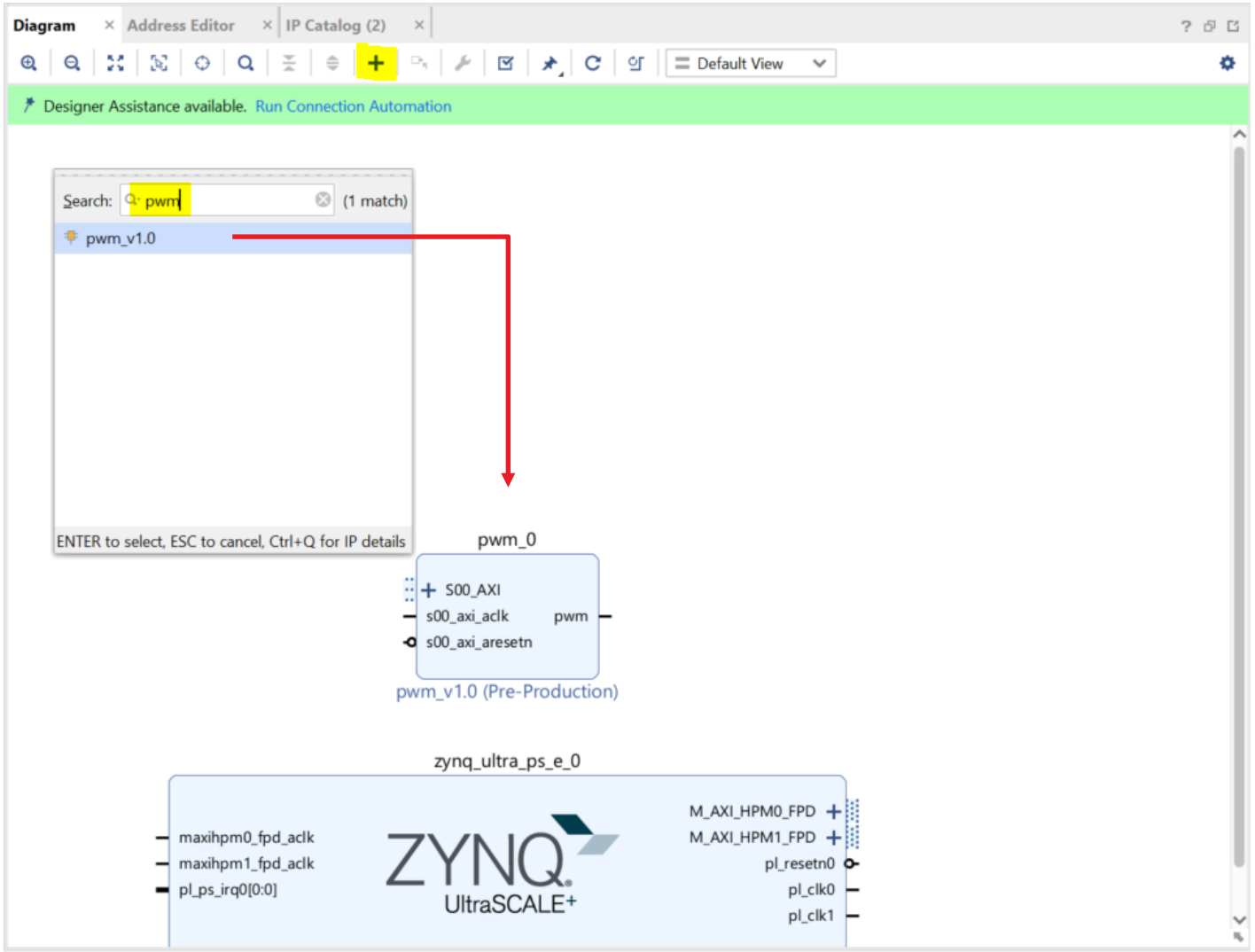
KD240 PWM LED

3. Create Block Design - 開始兜設計



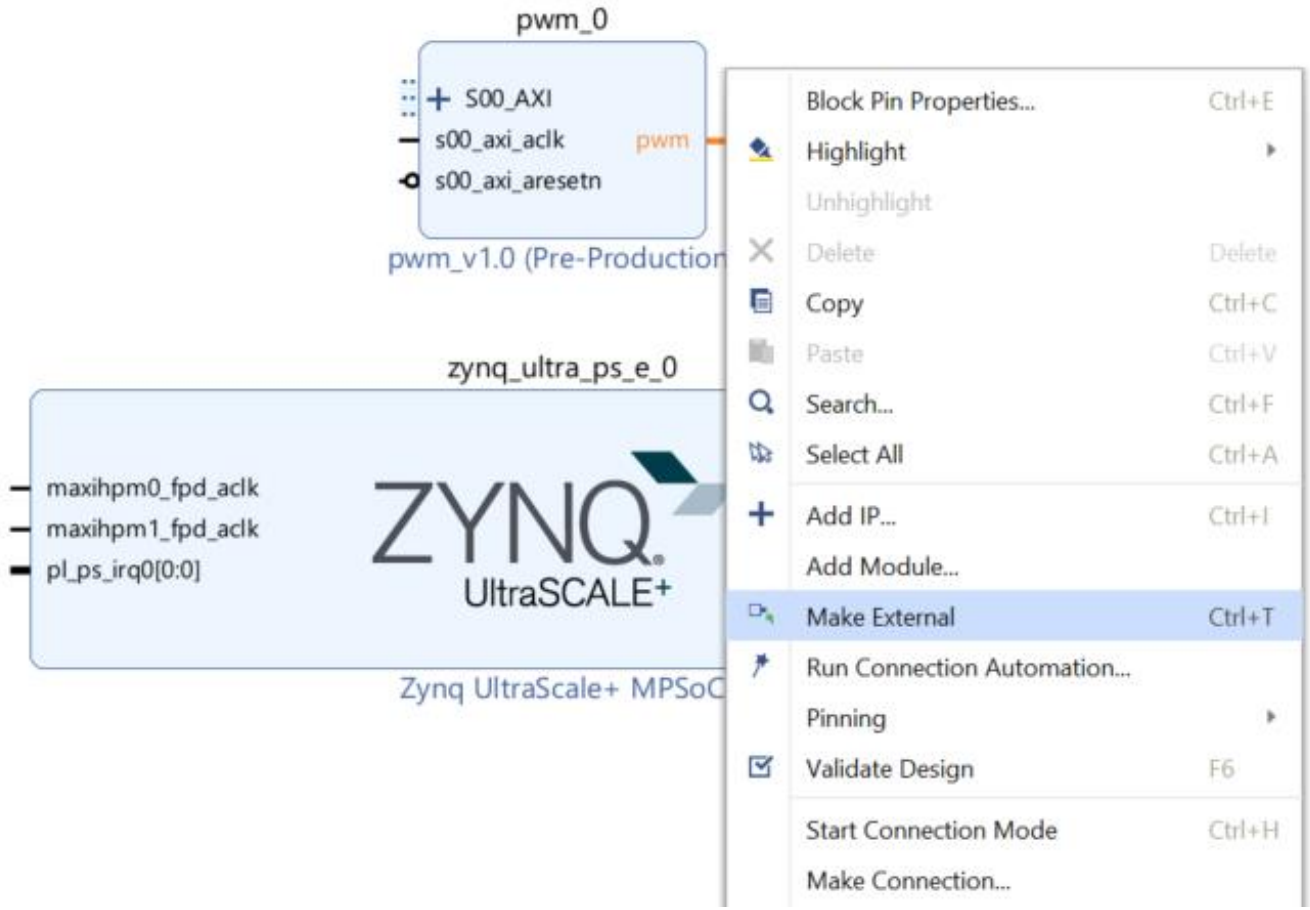
KD240 PWM LED

3. Create Block Design - 開始兜設計



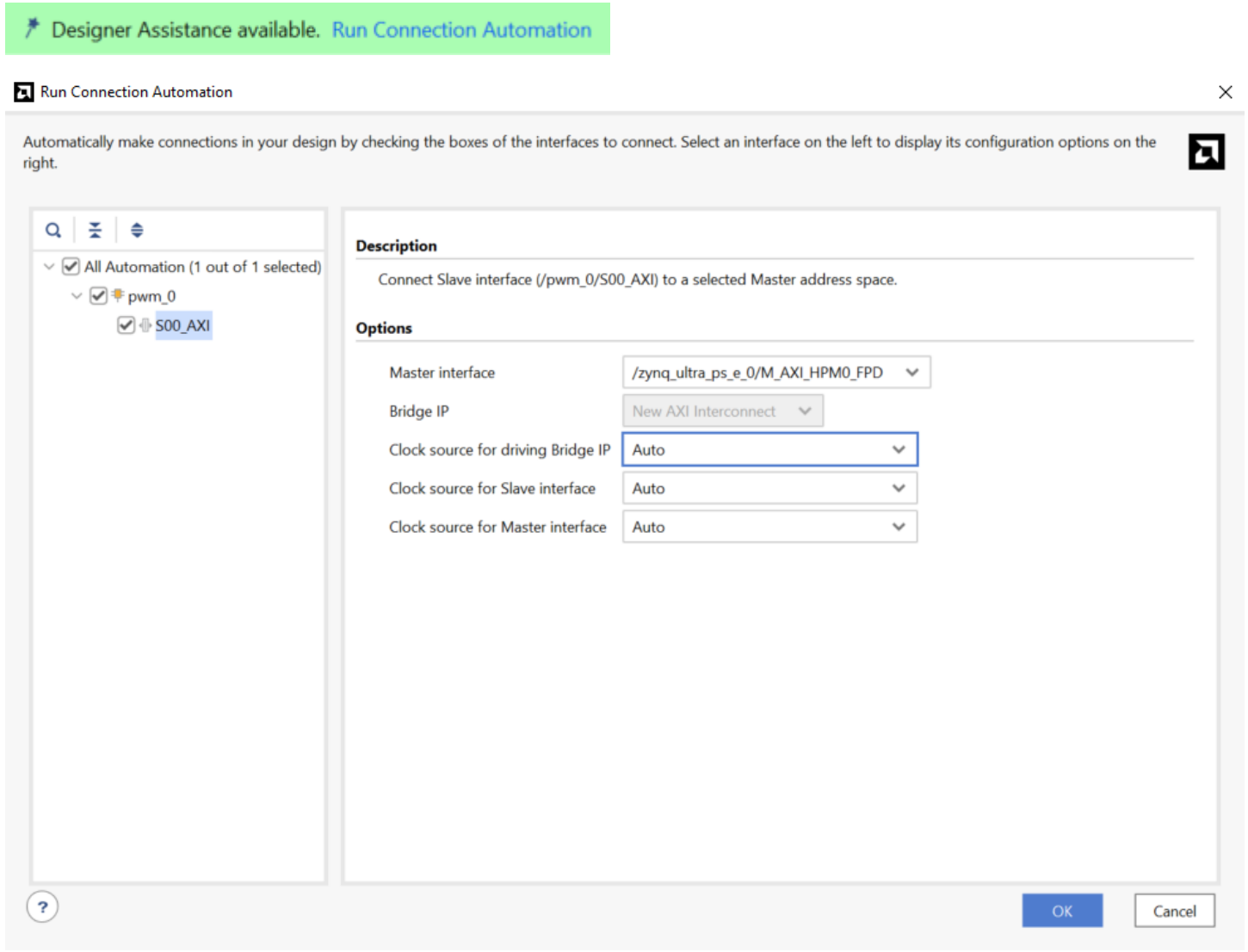
KD240 PWM LED

3. Create Block Design - 開始兜設計



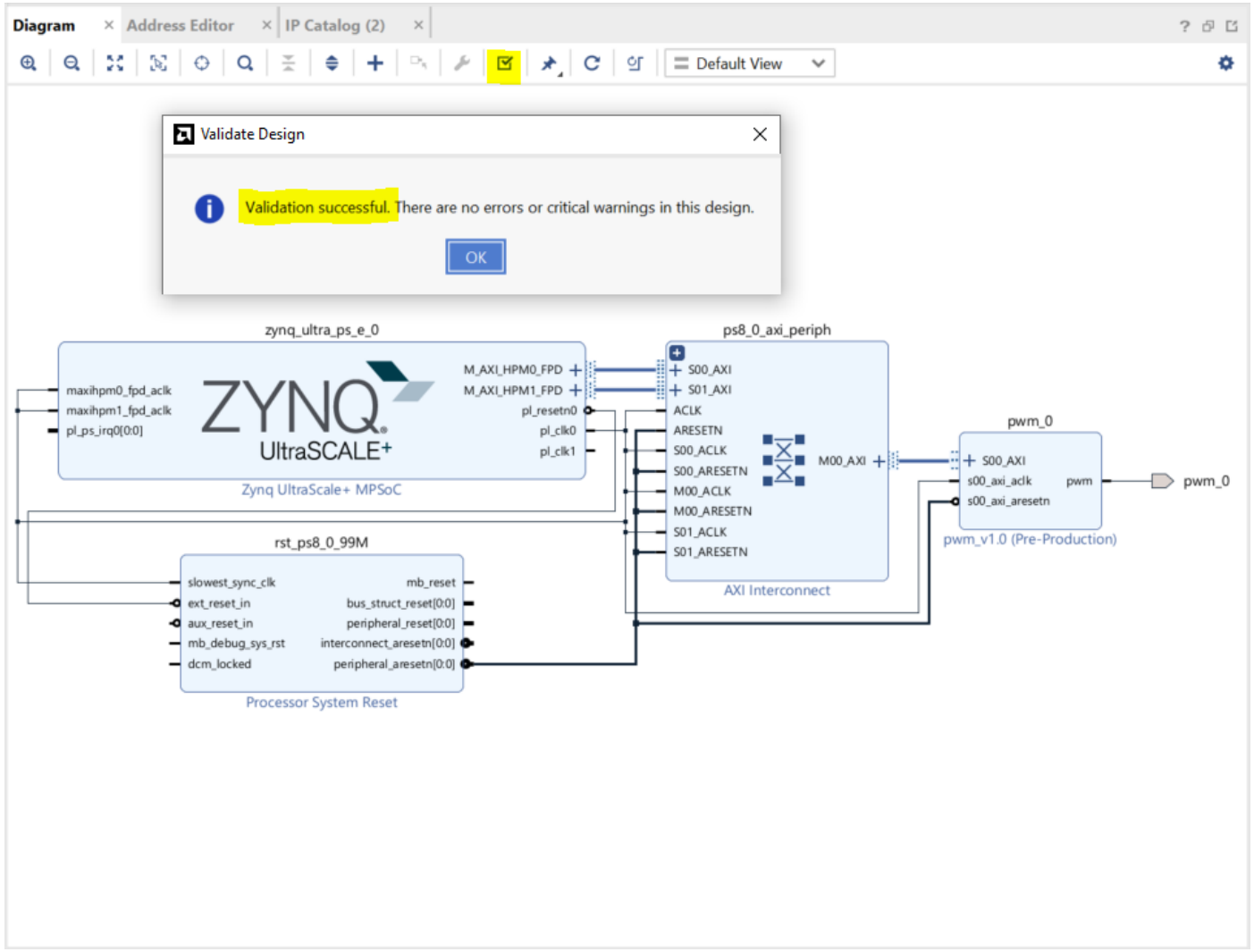
KD240 PWM LED

3. Create Block Design - 開始兜設計



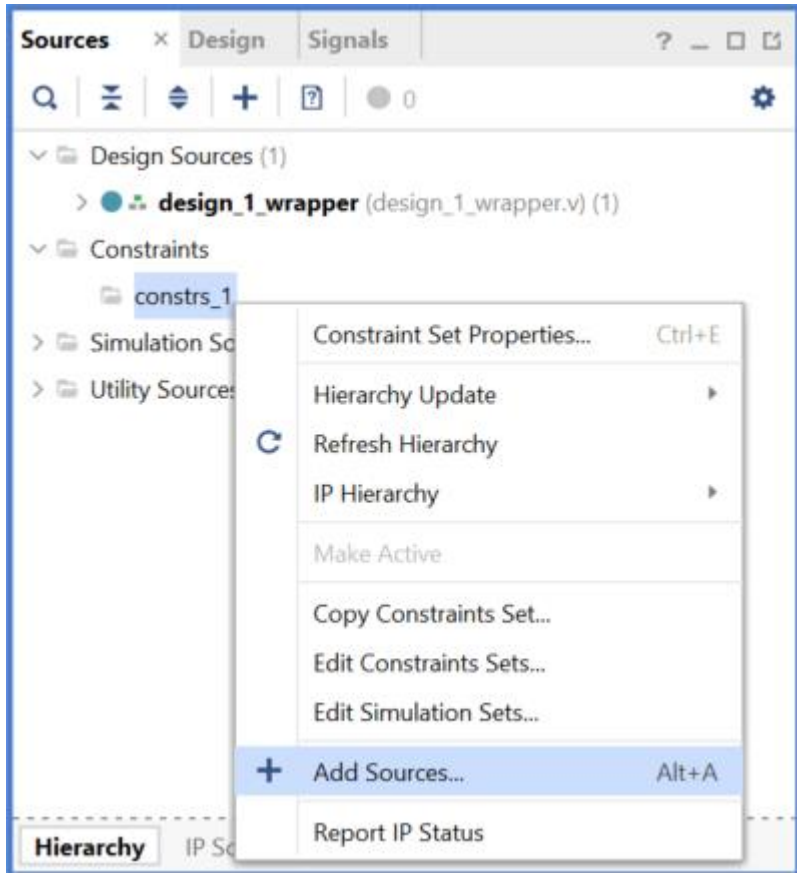
KD240 PWM LED

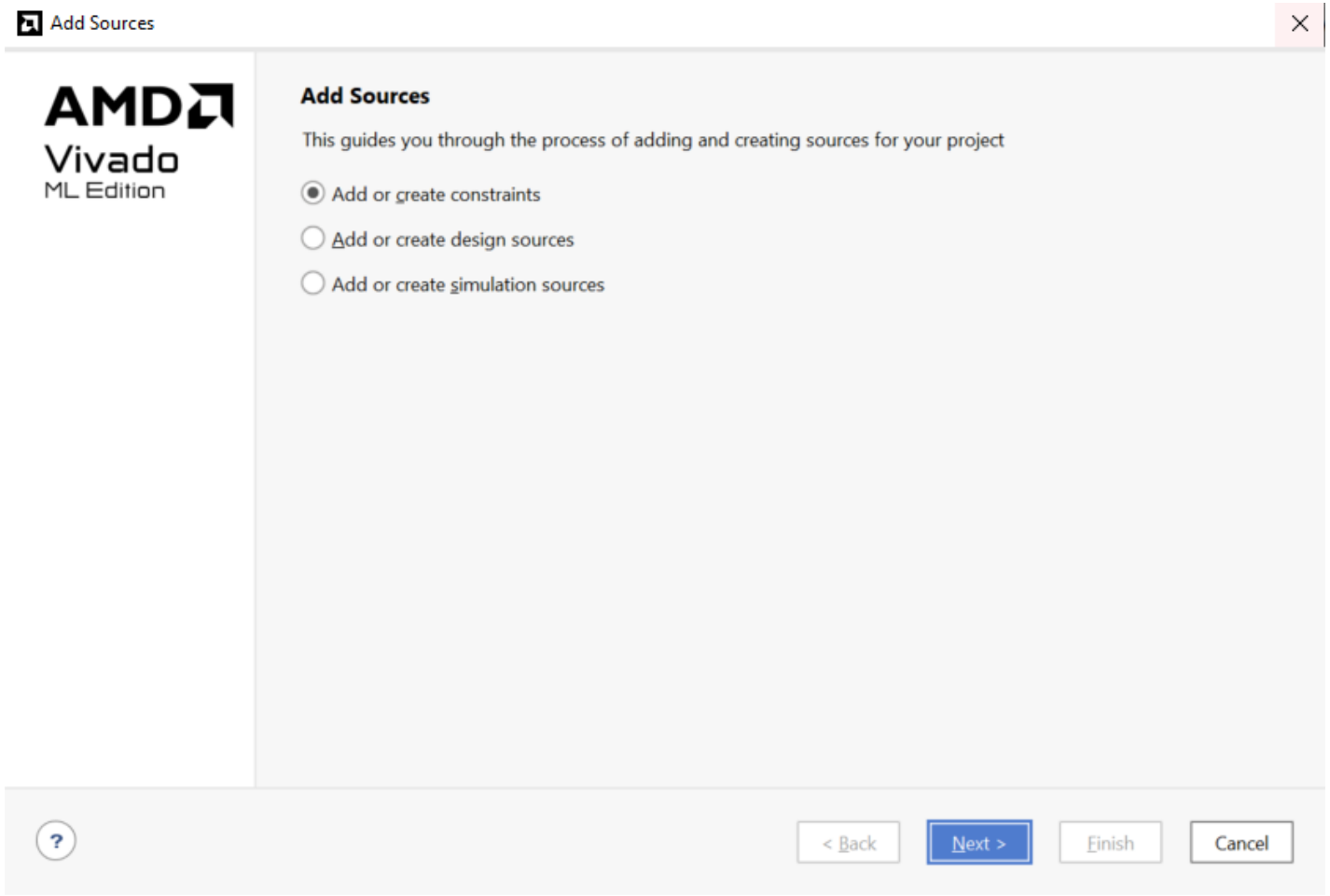
3. Create Block Design - 開始兜設計



KD240 PWM LED

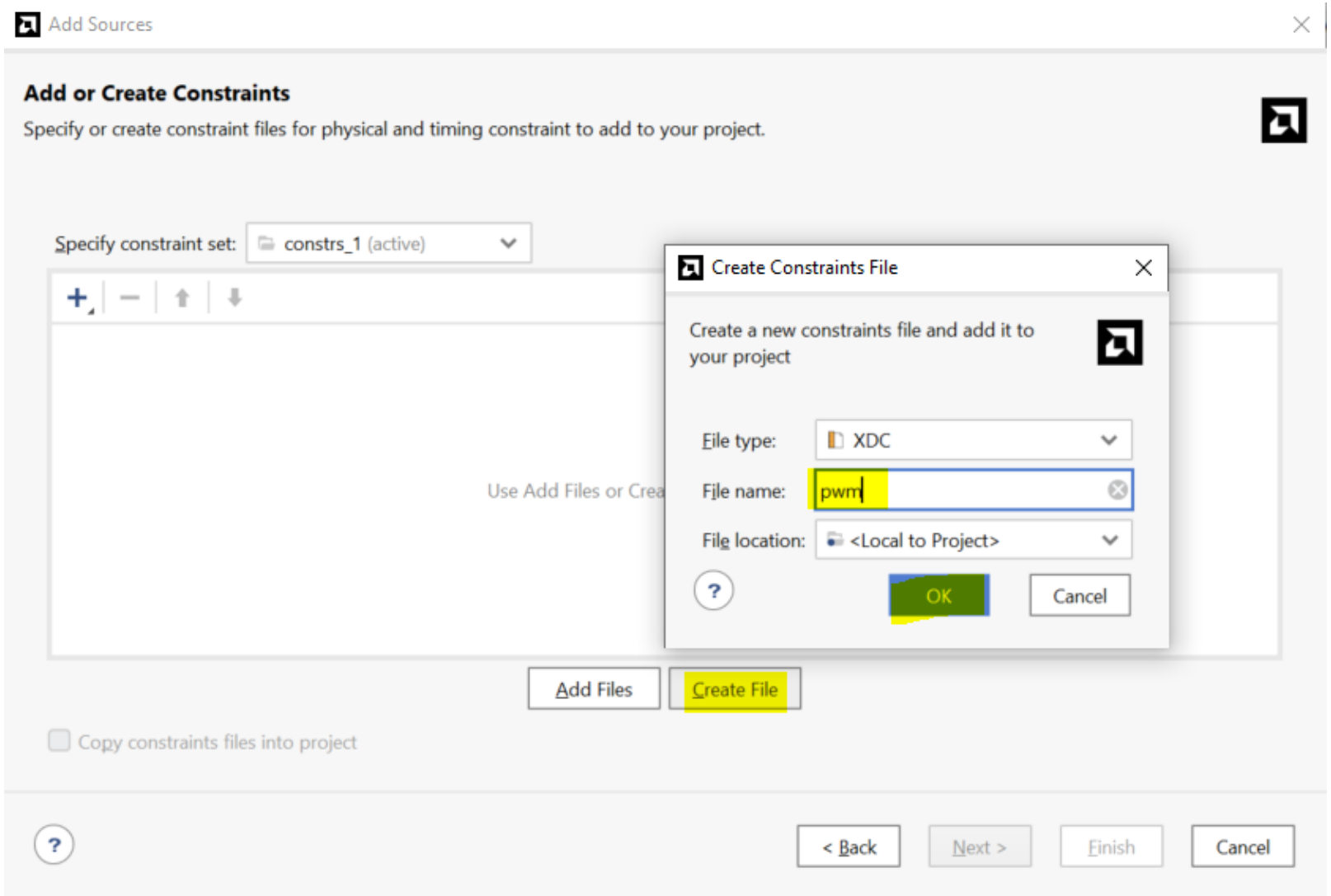
3. Create Block Design - 添加 XDC 内容





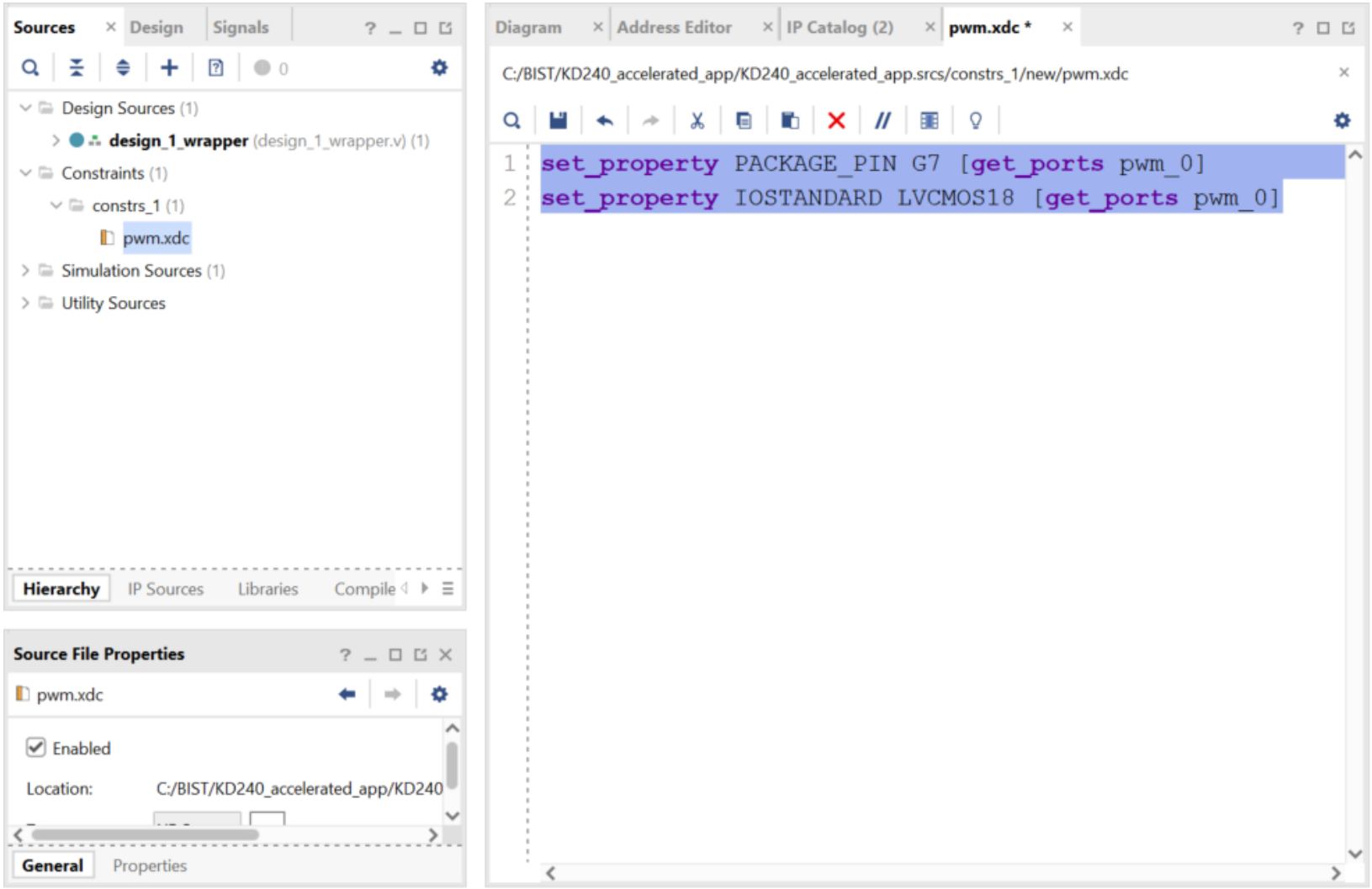
KD240 PWM LED

3. Create Block Design - 添加 XDC 内容



KD240 PWM LED

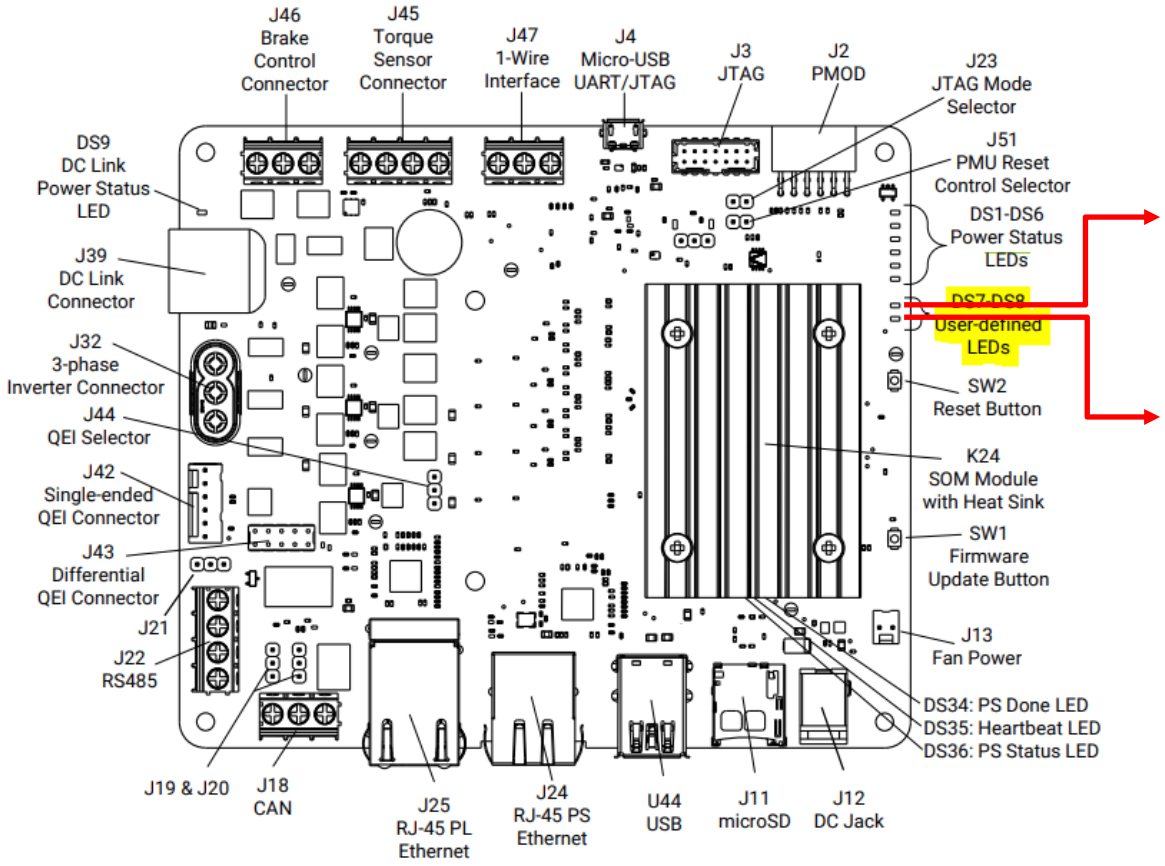
3. Create Block Design - 添加 XDC 内容



KD240 PWM LED

XDC 內容對照

Figure 2: **Interfaces and Connectors—Top of Card**

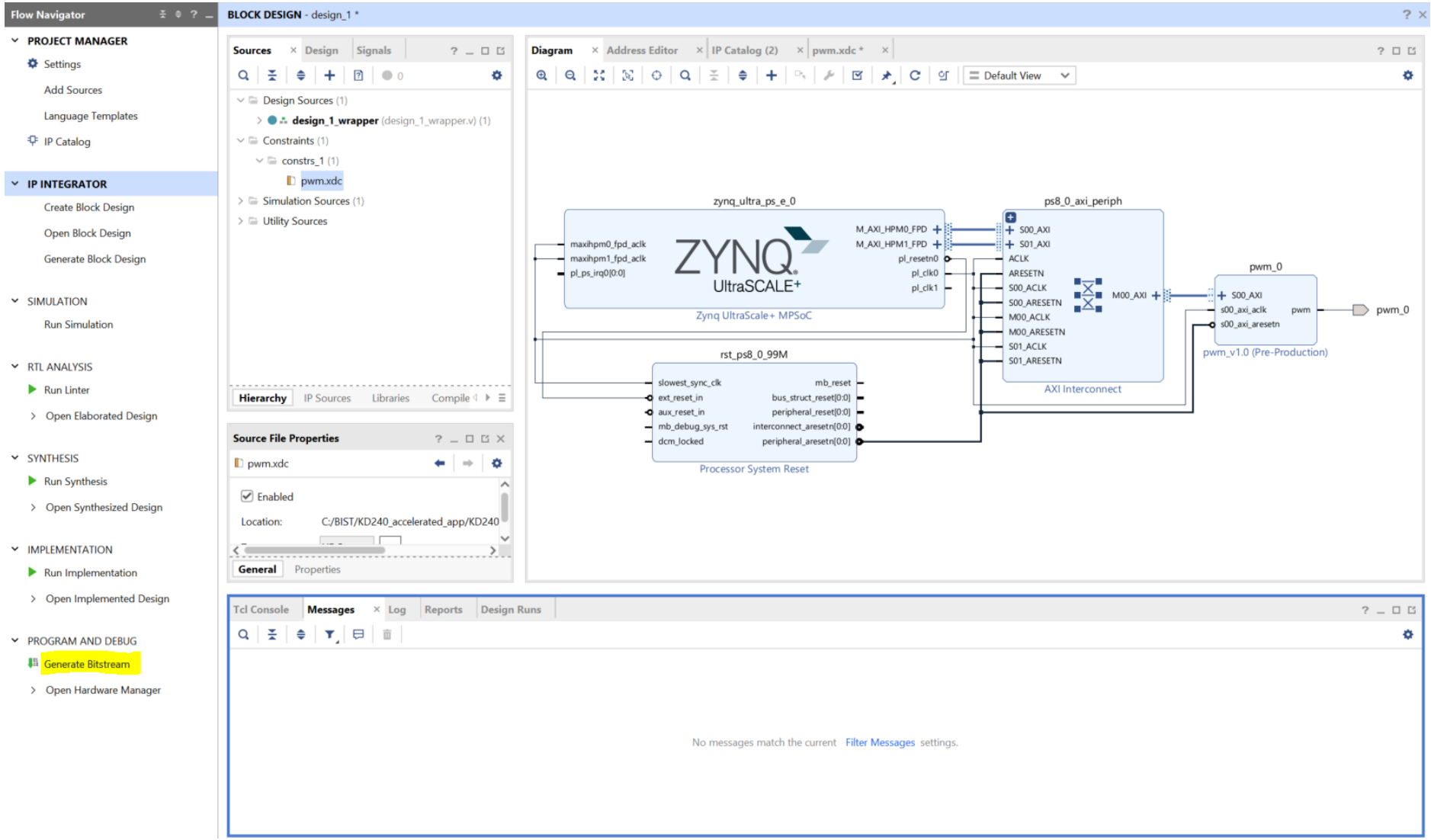


DS8：pin is H7, LVCMOS18

DS7：pin is G7, LVCMOS18

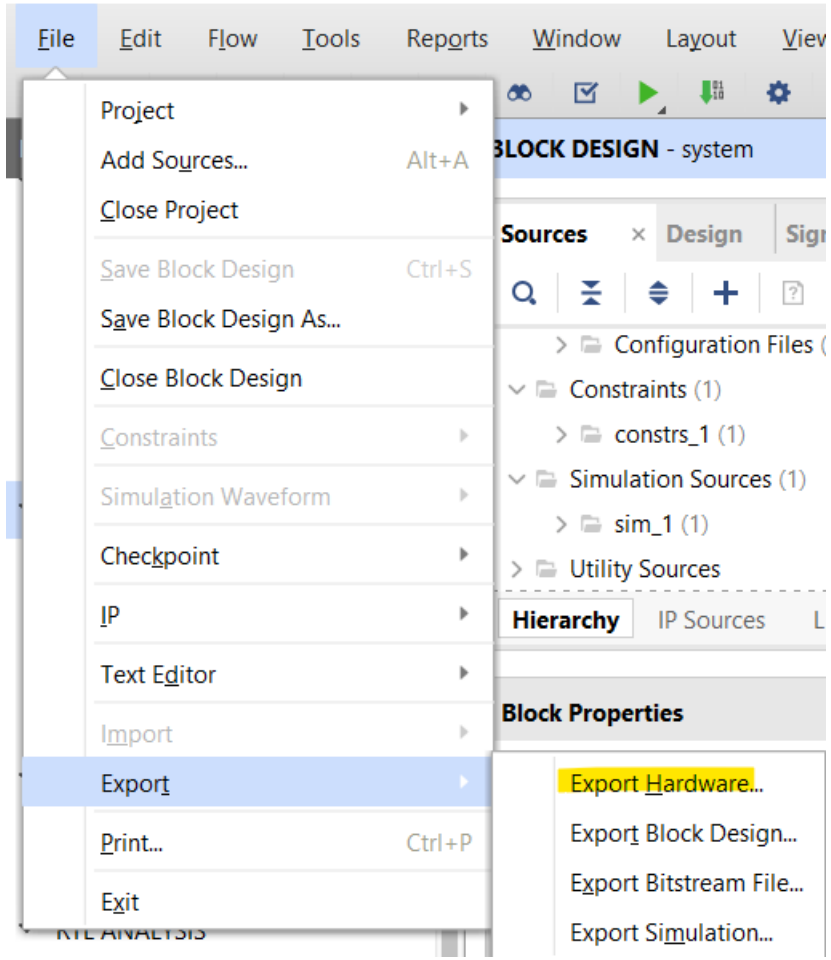
KD240 PWM LED

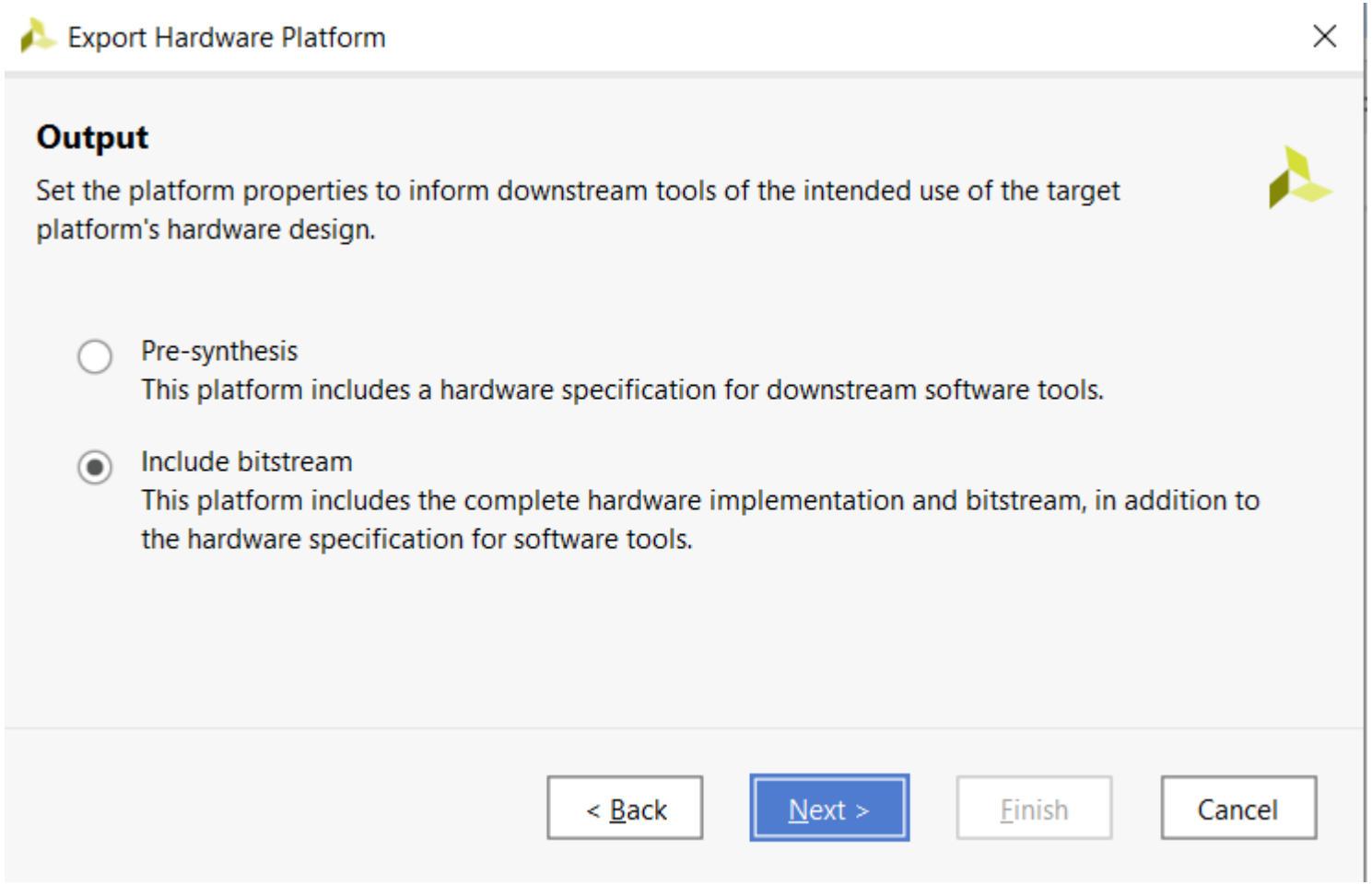
4. Generate Bitstream




ONE HOUR
LATER...

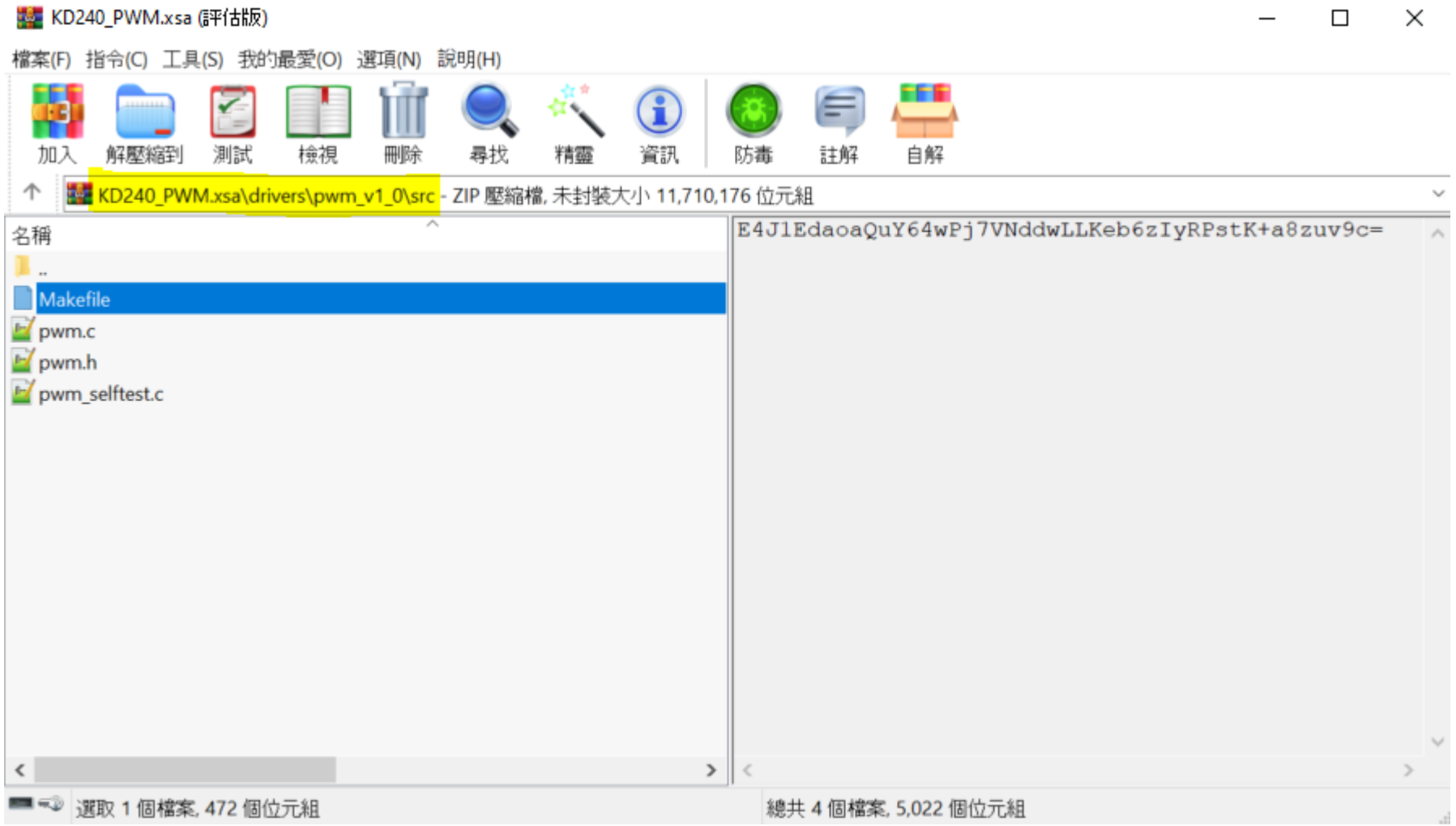
KD240 PWM LED





KD240 PWM LED

5. 自定義 IP 要改 xsa 內容，透過 winrar 開啟 xsa 並點選以下路徑開啟 Makefile



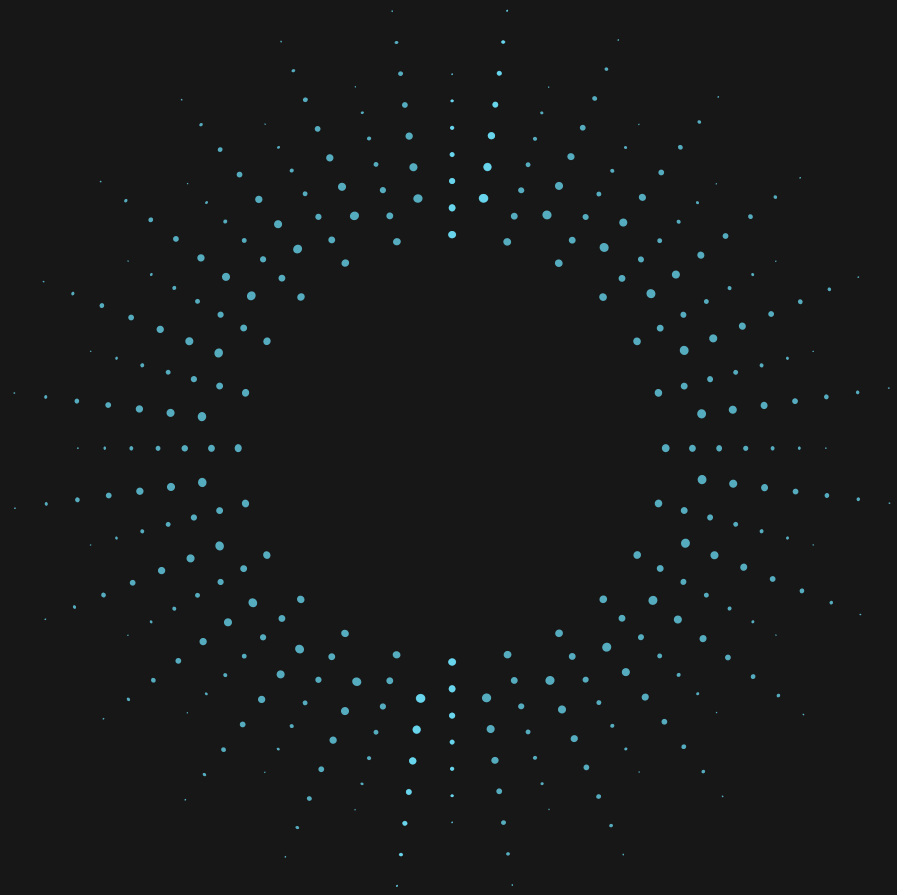
KD240 PWM LED

5. 自定義 IP 要改 xsa 內容，透過 winrar 開啟 xsa 並點選以下路徑開啟 Makefile，修改後儲存

```
COMPILER=
ARCHIVER=
CP=cp
COMPILER_FLAGS=
EXTRA_COMPILER_FLAGS=
LIB=libxil.a

RELEASEDIR=../../../lib
INCLUDEDIR=../../../include
INCLUDES=-I./. -I${INCLUDEDIR}

INCLUDEFILES=*.h
LIBSOURCES=*.c
OUTS = *.o

libs:
    echo "Compiling pwm..."
    $(COMPILER) $(COMPILER_FLAGS) $(EXTRA_COMPILER_FLAGS) $(INCLUDES) $(LIBSOURCES)
    $(ARCHIVER) -r ${RELEASEDIR}/${LIB} ${OUTS}
    make clean

include:
    ${CP} $(INCLUDEFILES) $(INCLUDEDIR)

clean:
    rm -rf ${OUTS}
```

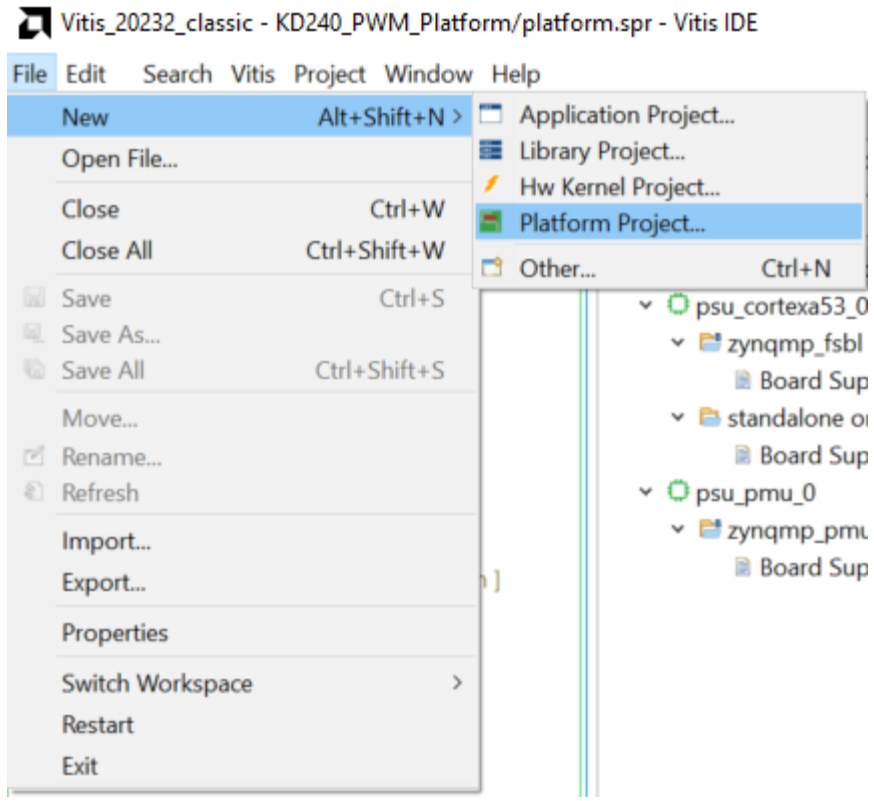
```
COMPILER=
ARCHIVER=
CP=cp
COMPILER_FLAGS=
EXTRA_COMPILER_FLAGS=
LIB=libxil.a

RELEASEDIR=../../../lib
INCLUDEDIR=../../../include
INCLUDES=-I./. -I${INCLUDEDIR}

INCLUDEFILES=$(wildcard *.h)
LIBSOURCES=$(wildcard *.c)
OUTS = $(wildcard *.o)

libs:
    echo "Compiling pwm..."
    $(COMPILER) $(COMPILER_FLAGS) $(EXTRA_COMPILER_FLAGS) $(INCLUDES) $(LIBSOURCES)
    $(ARCHIVER) -r ${RELEASEDIR}/${LIB} ${OUTS}
    make clean

include:
    ${CP} $(INCLUDEFILES) $(INCLUDEDIR)

clean:
    rm -rf ${OUTS}
```

Vitis 2023.2.1 Part



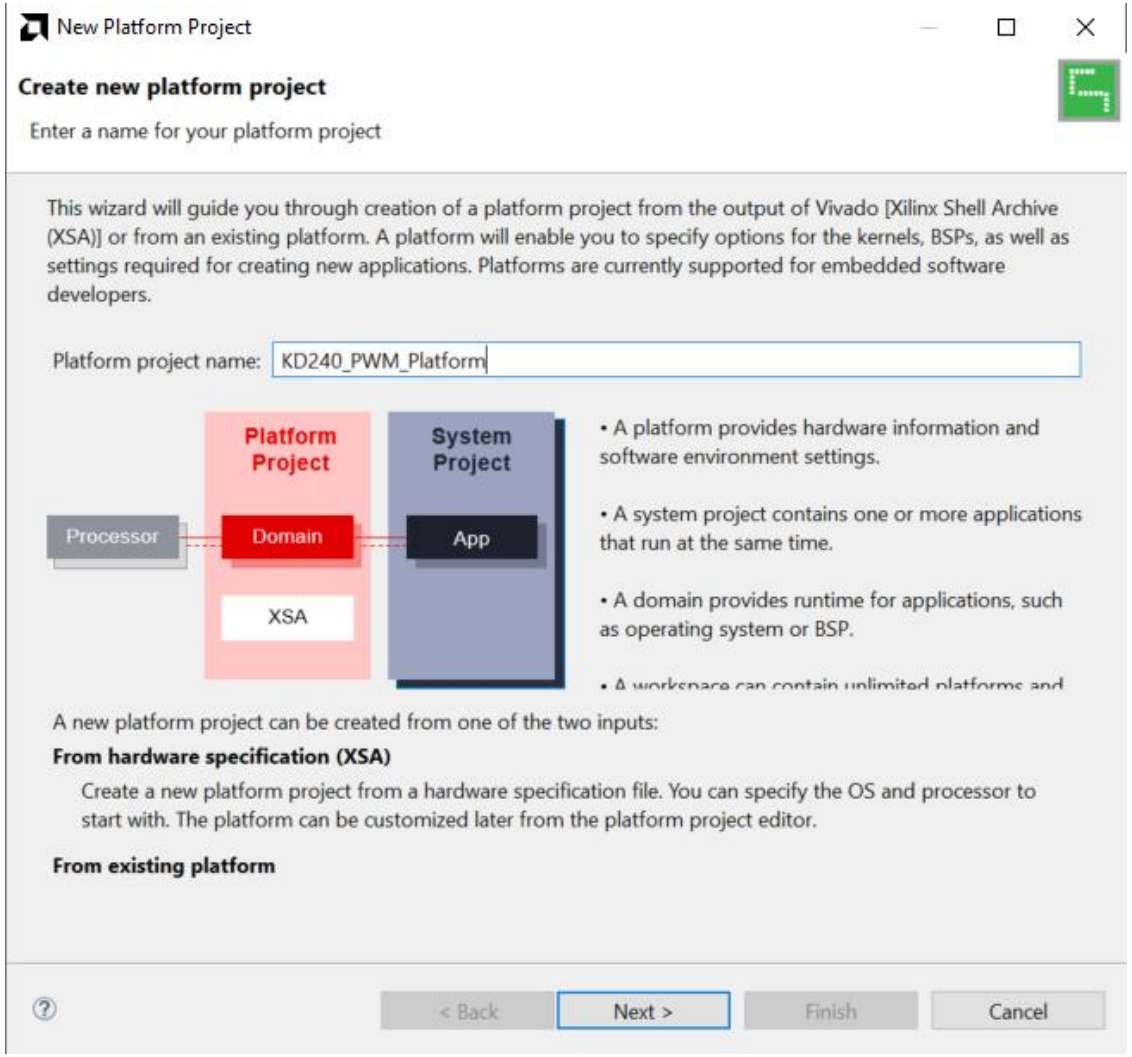
KD240 PWM LED

1. Create Platform and Application



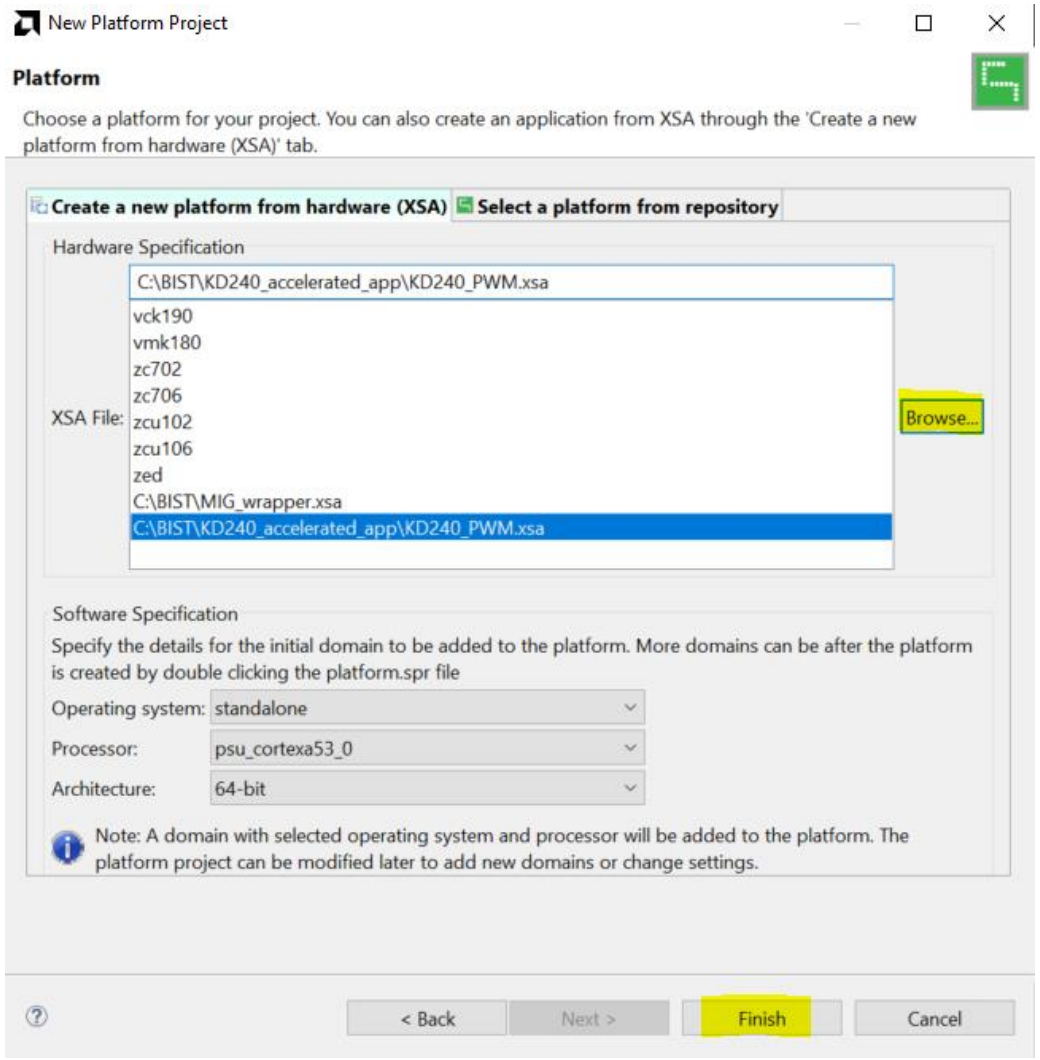
KD240 PWM LED

1. Create Platform and Application



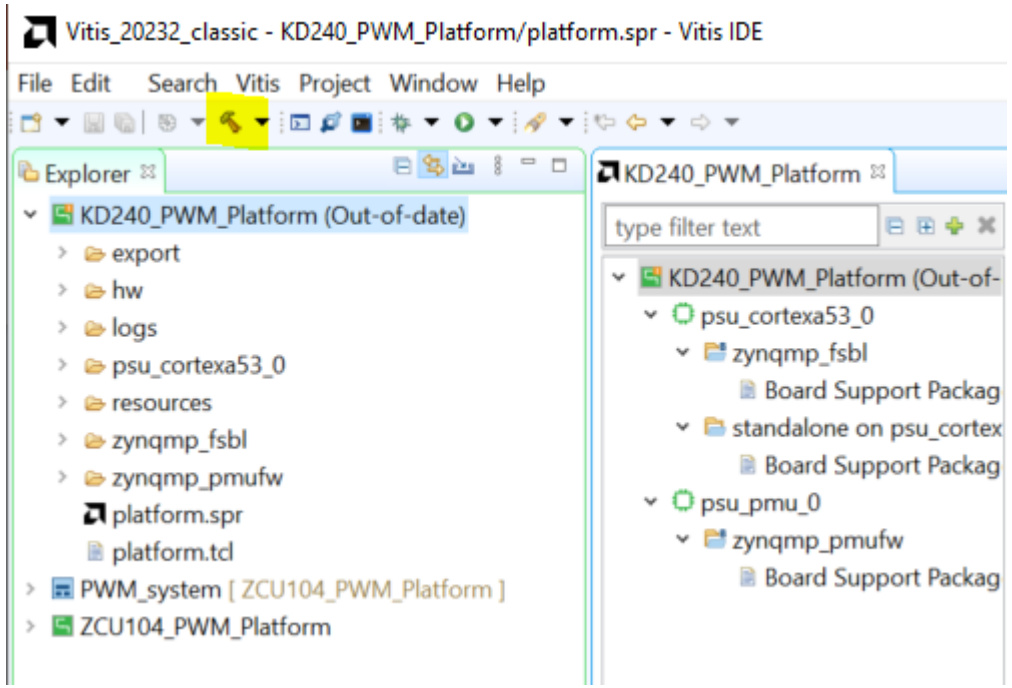
KD240 PWM LED

1. Create Platform and Application



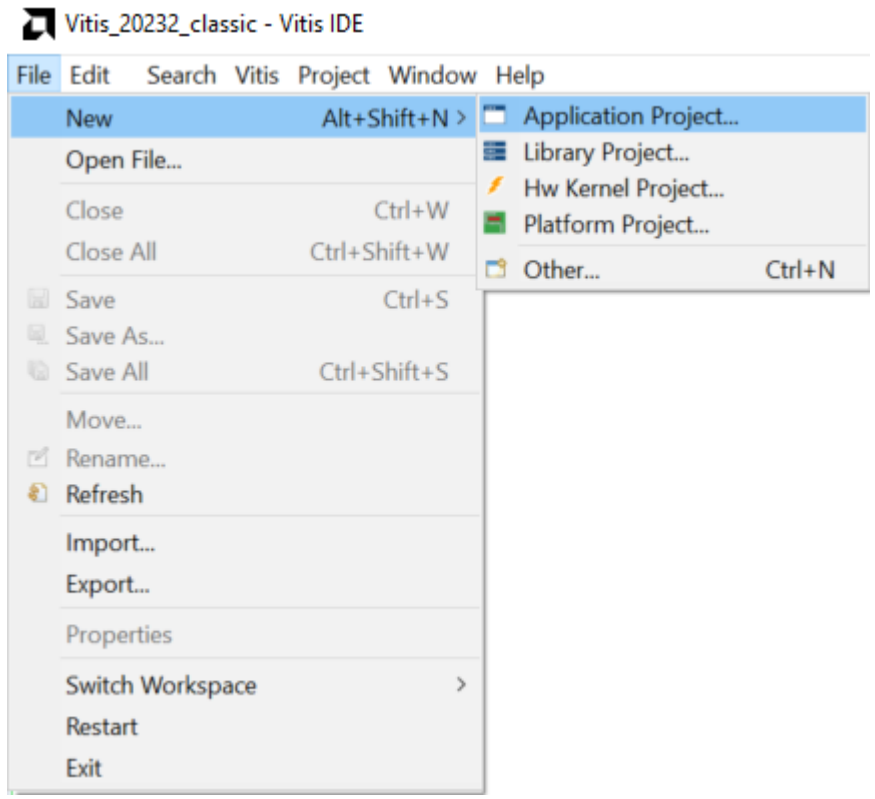
KD240 PWM LED

1. Create Platform and Application －Create 完要先 build 才會有 Library 連結檔案



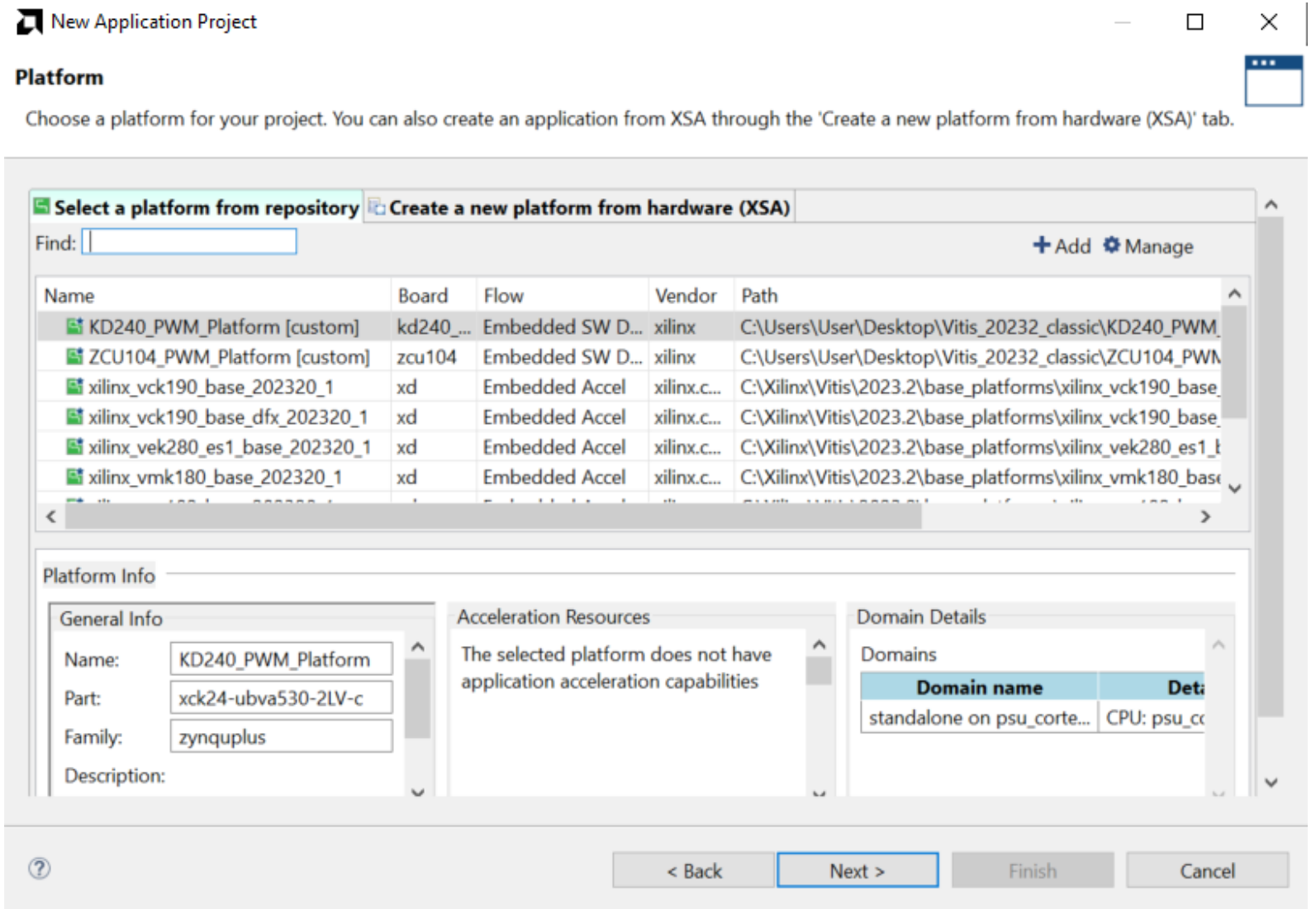
KD240 PWM LED

1. Create Platform and Application



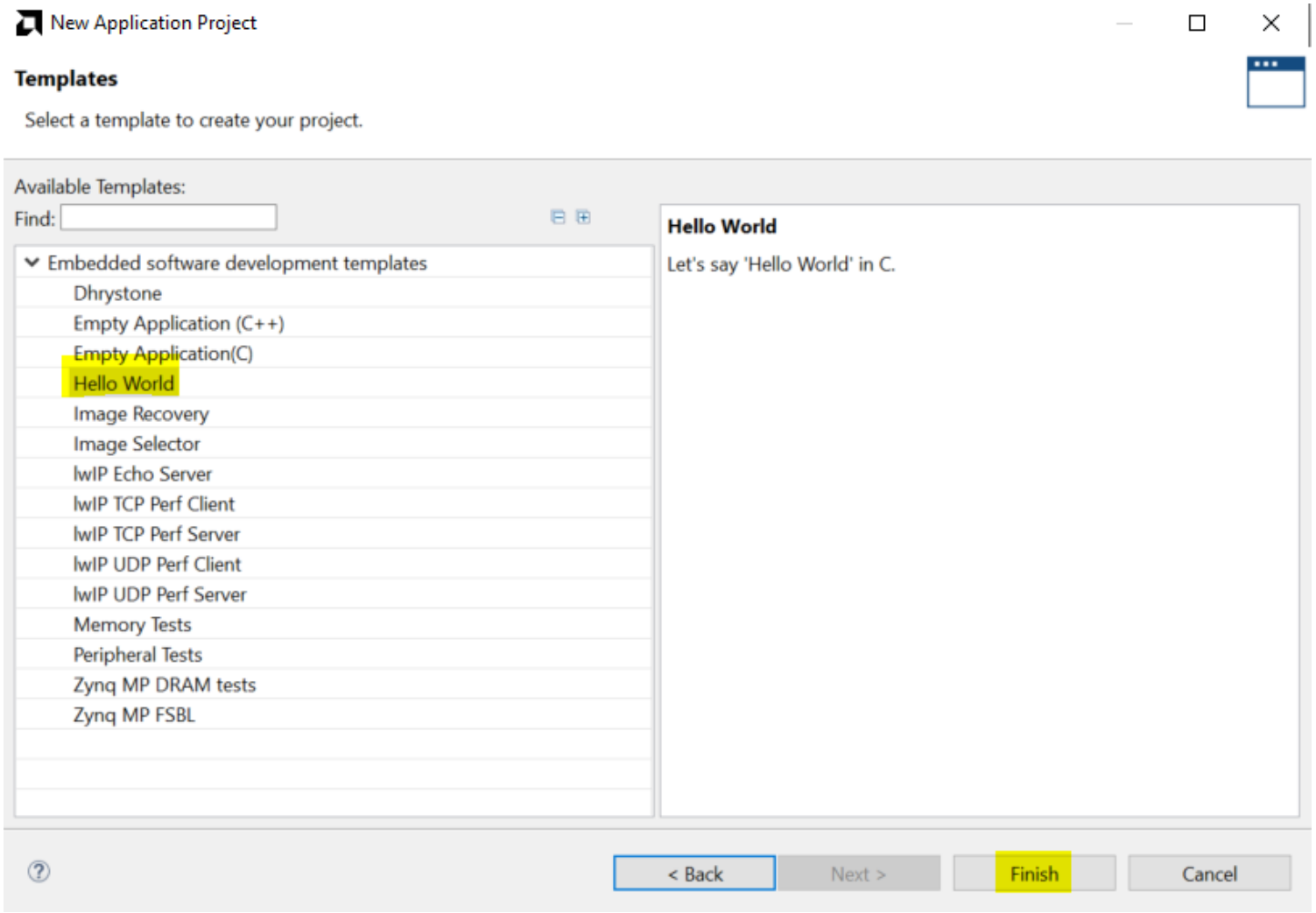
KD240 PWM LED

1. Create Platform and Application



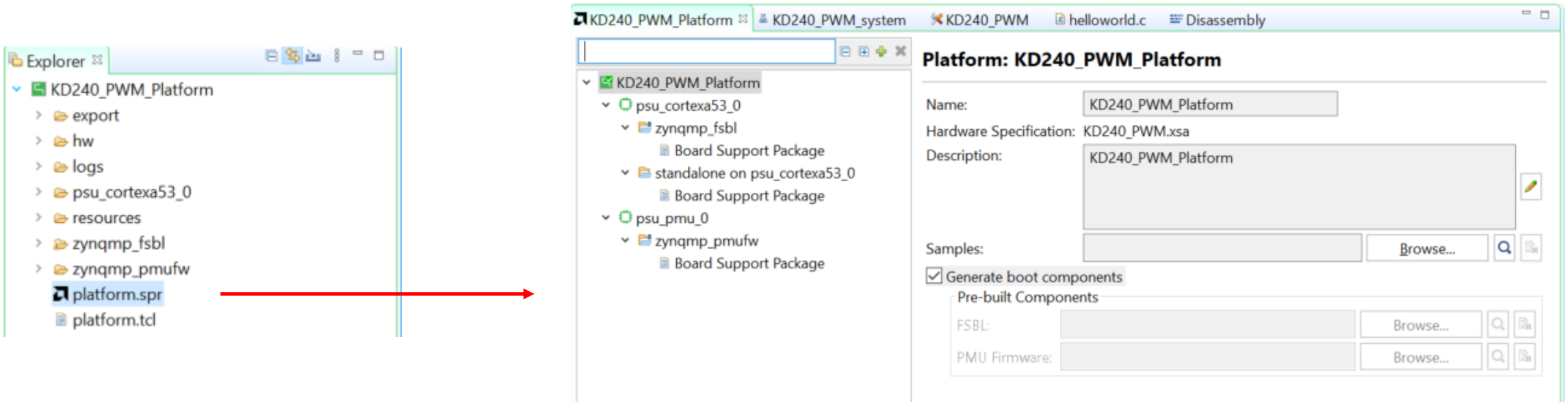
KD240 PWM LED

1. Create Platform and Application - Next 直到選擇 Hello World 作為程式模板



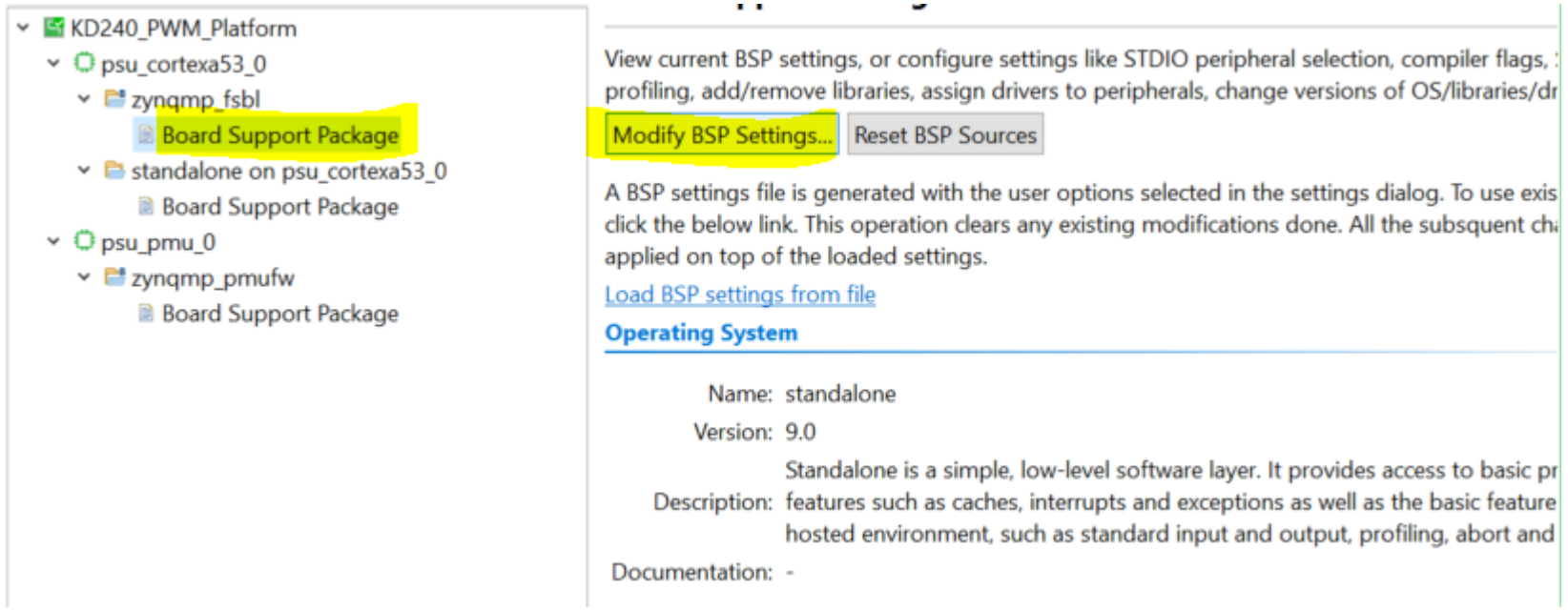
KD240 PWM LED

2. 更改 KD240 BSP - 因為 Kria Uart 預設為 1，但在 Vitis 內原始設定為 0



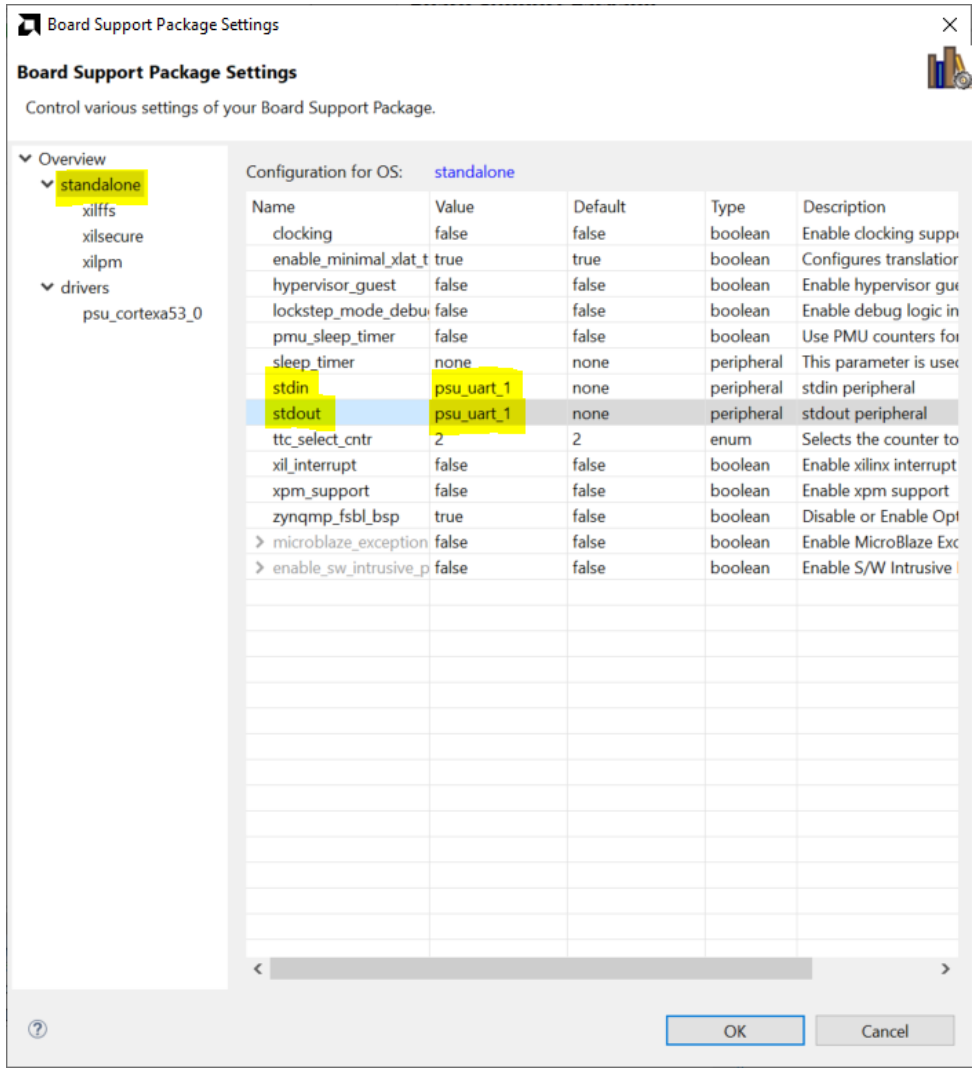
KD240 PWM LED

2. 更改 KD240 BSP - 因為 Kria Uart 預設為 1，但在 Vitis 內原始設定為 0



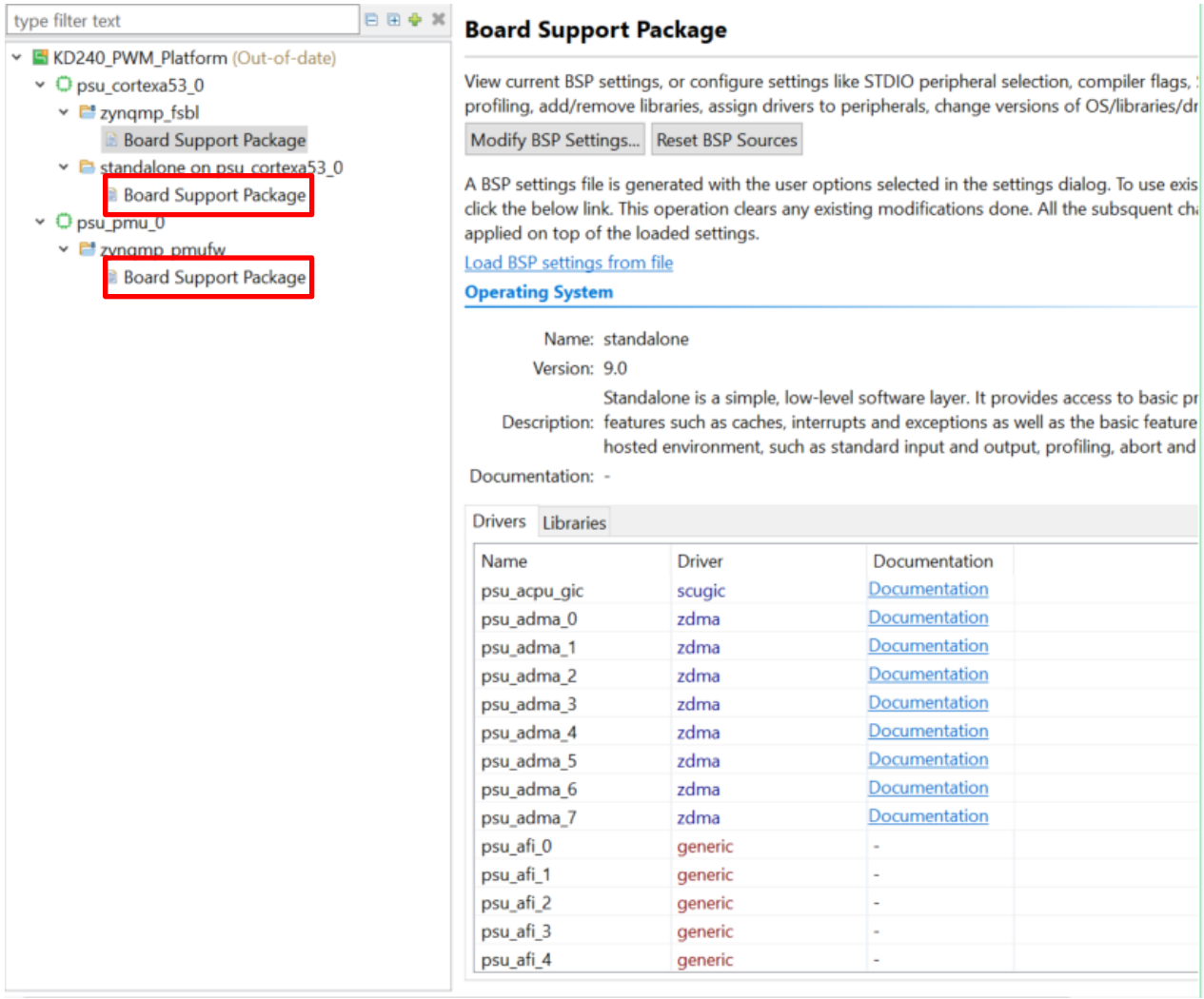
KD240 PWM LED

2. 更改 KD240 BSP - 因為 Kria Uart 預設為 1，但在 Vitis 內原始設定為 0



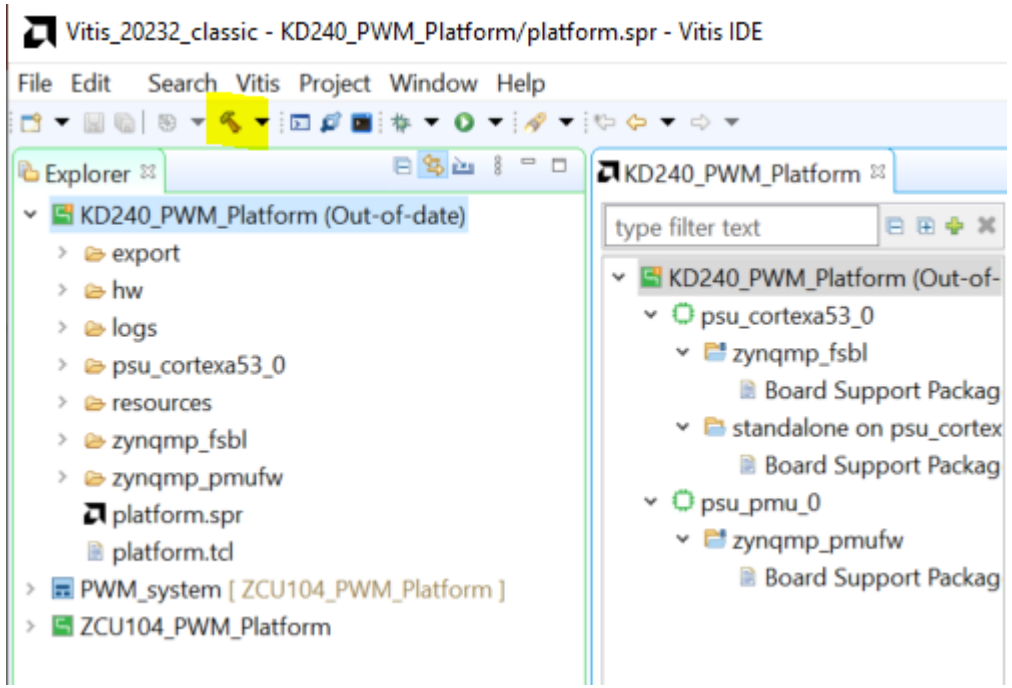
KD240 PWM LED

2. 更改 KD240 BSP - 其他兩個也一樣這樣改



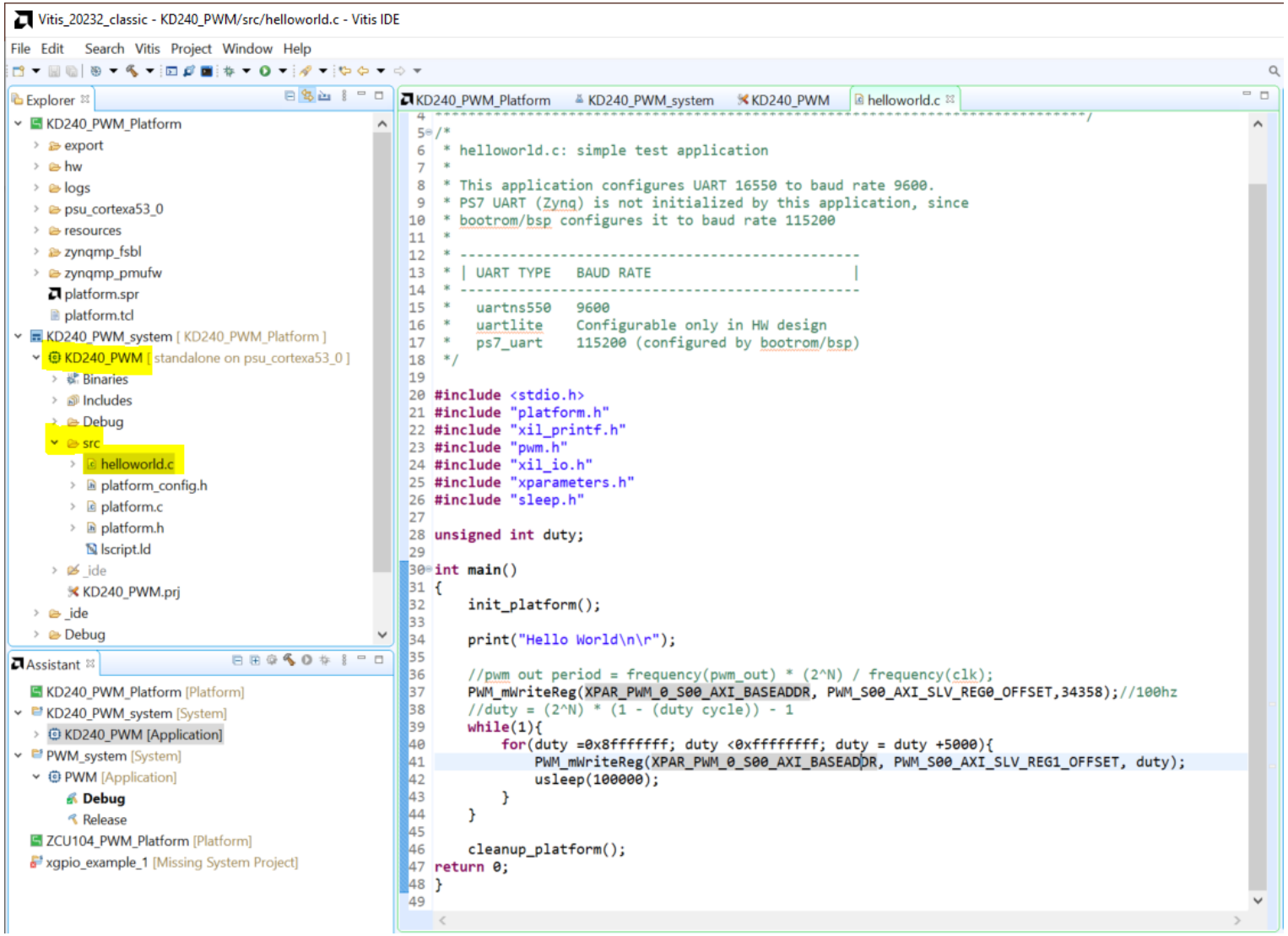
KD240 PWM LED

2. 更改 KD240 BSP - 改完要重 Build



KD240 PWM LED

3. 開啟 helloworld.c，更改程式內容如下



KD240 PWM LED

4. 修改 KD240 開機模式 - 因為 Kria 系列預設是從 QSPI 開機，其次是 SD 卡，可以透過 TCL 修改開機模式

Setting Bootmodes — Kria™ SOM 2022.1 documentation (xilinx.github.io)

Setting Bootmodes

Once applications and custom HW designs are generated, developers need to move them to target. If using the Kria Starter Kit, developers can use various boot-modes to test monolithic boot to application software using the following TCL scripts to set the preferred development boot process. Developers will first put the functions in a `<boot>.tcl` script. Then, with the host machine connected with their SOM kit, they use the following commands in XSDB or XSCT:

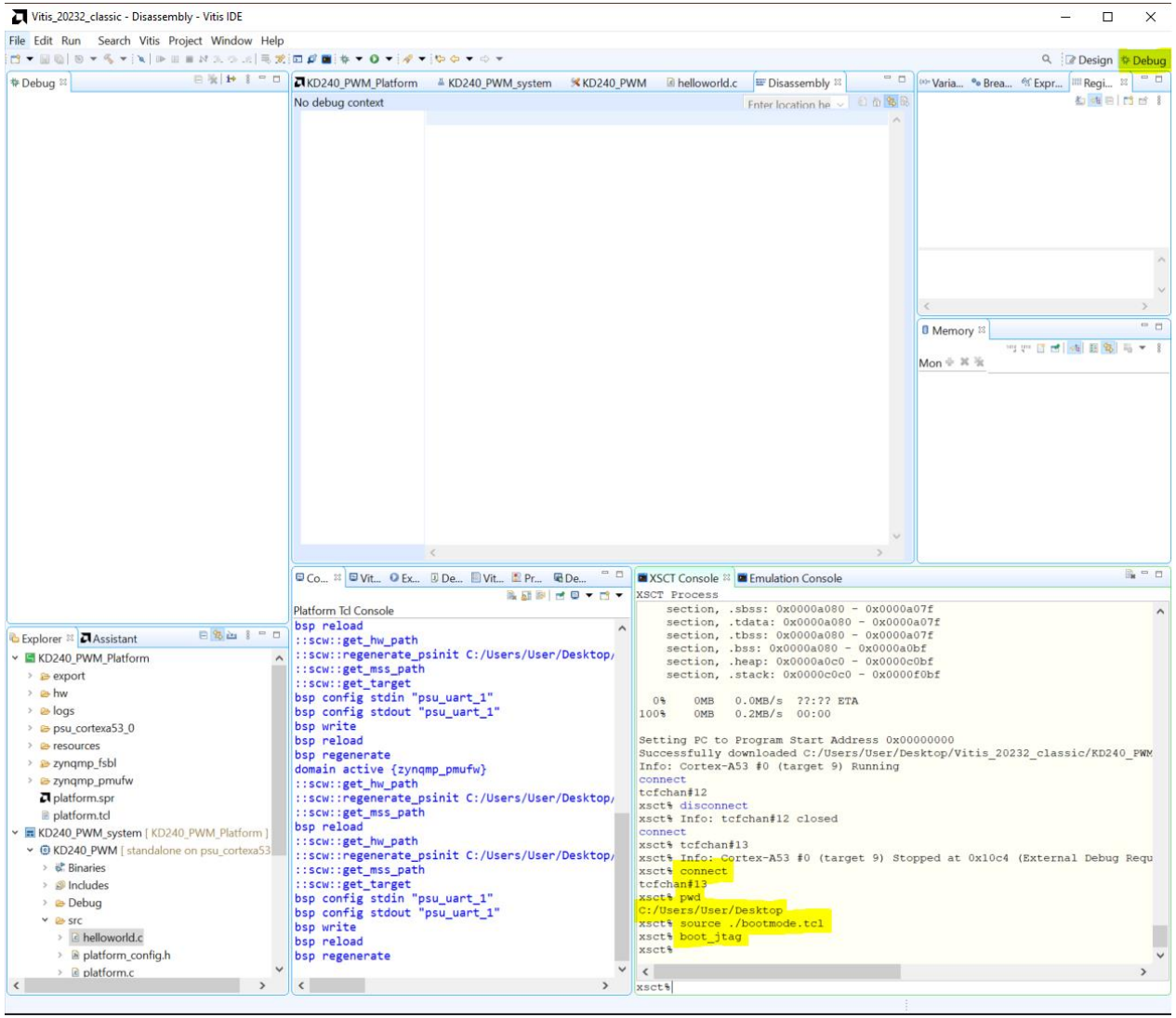
```
connect
source <boot>.tcl
boot_<mode>
```

To set K26 to JTAG bootmode using XSDB/XSCT, add the following TCL scripts and call the function:

```
proc boot_jtag { } {
############################
# Switch to JTAG boot mode #
############################
targets -set -filter {name =~ "PSU"}
# update multiboot to ZERO
mwr 0xffca0010 0x0
# change boot mode to JTAG
mwr 0xff5e0200 0x0100
# reset
rst -system
}
```

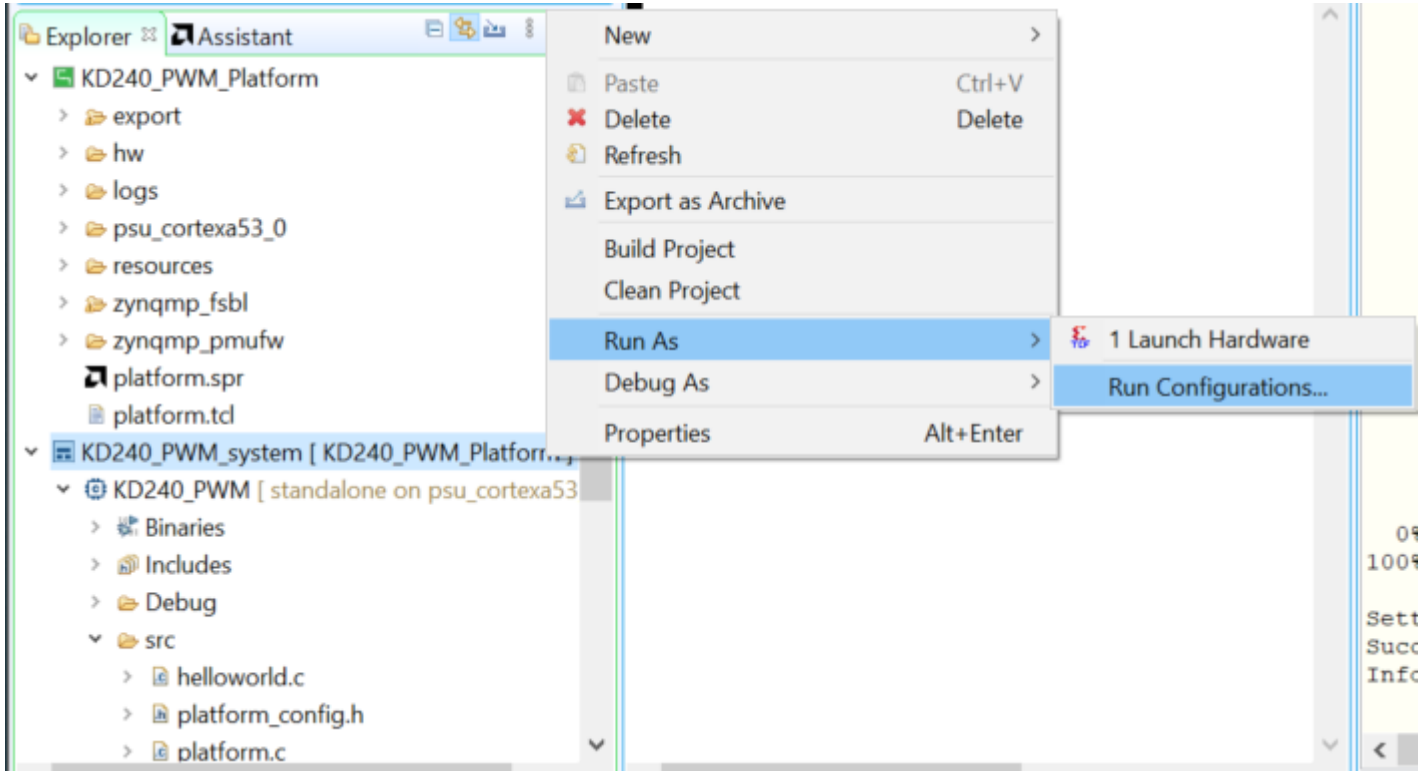
KD240 PWM LED

4. 修改 KD240 開機模式 - 建立好 TCL 檔之後，在 Vitis 的 xsct 內輸入以下



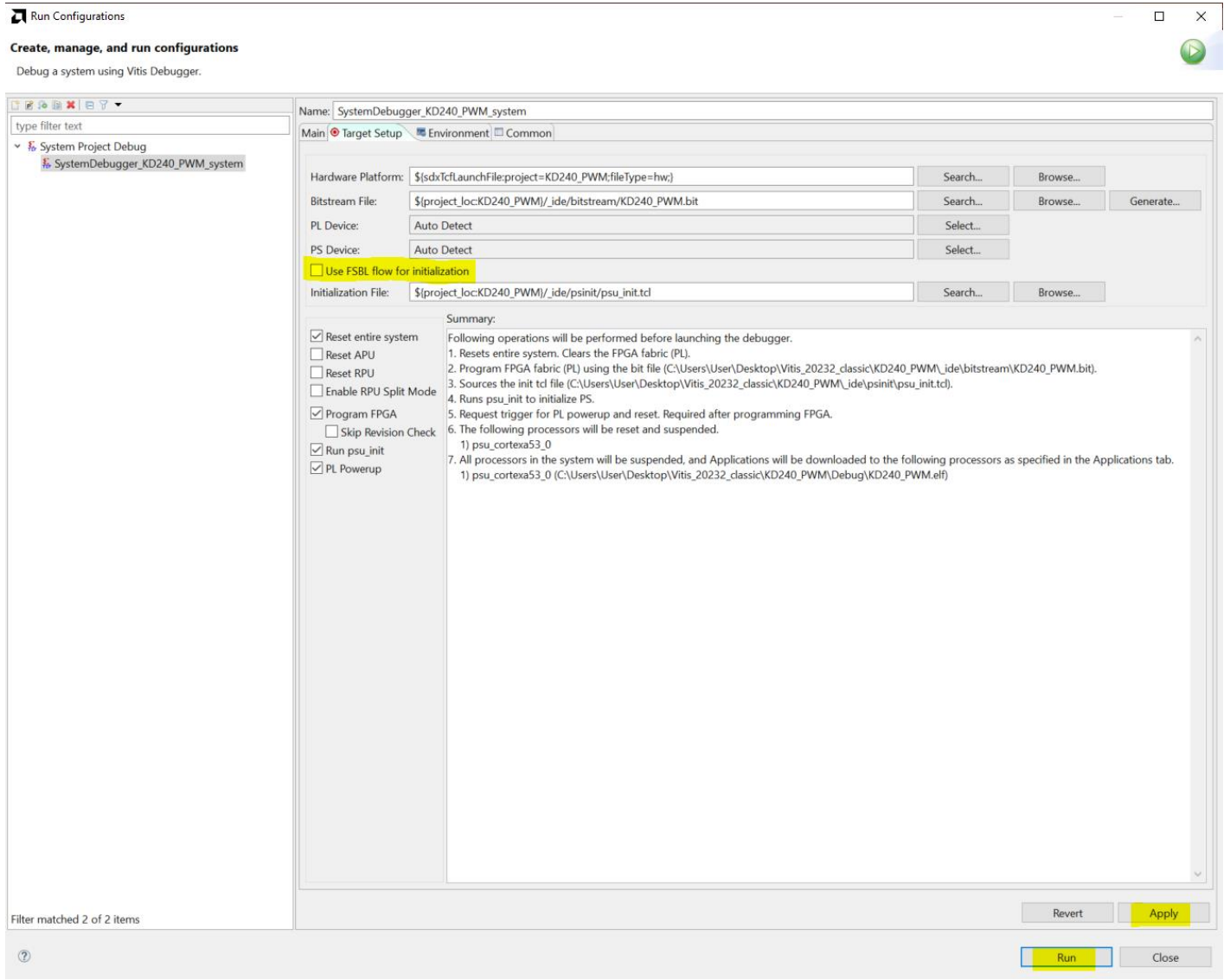
KD240 PWM LED

4. (Optional) 修改 Run Configuration - 或是以另一種方式來讓 KD240 可以被燒錄成功



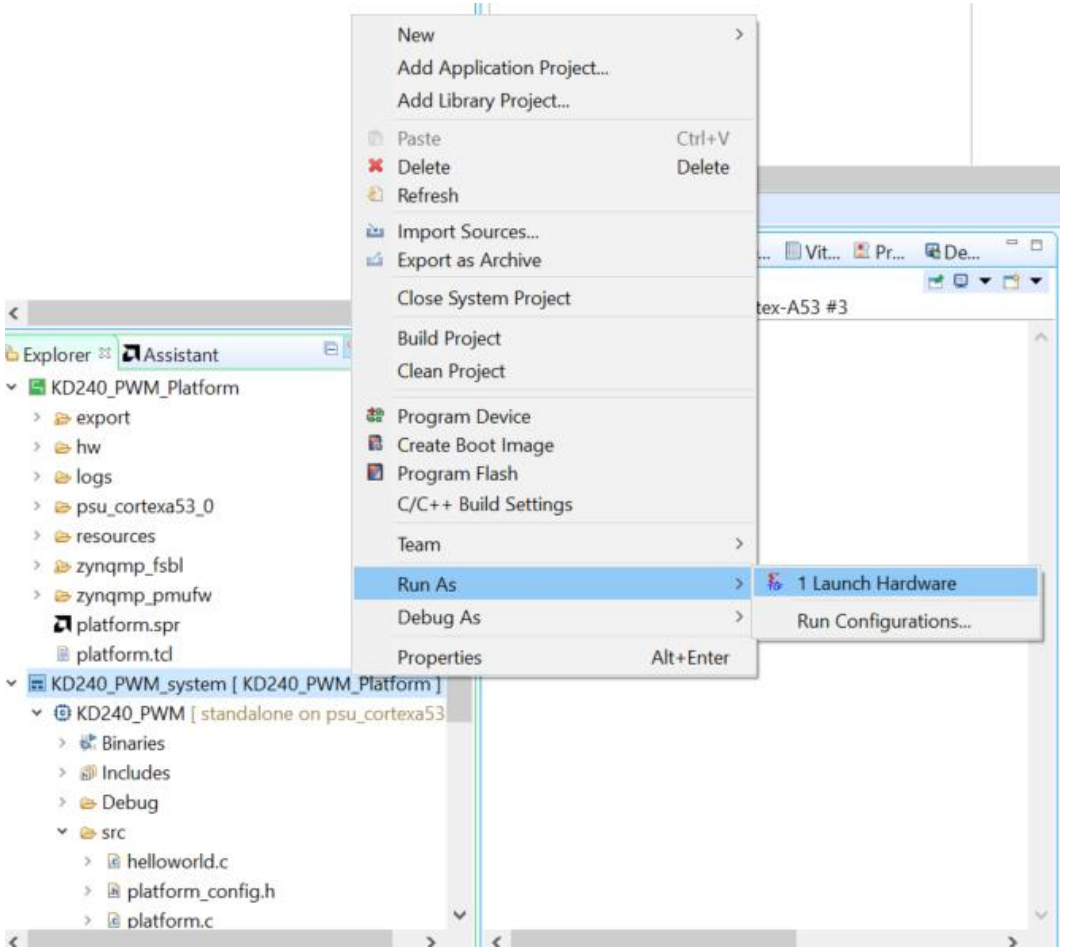
KD240 PWM LED

4. (Optional) 修改 Run Configuration - Use FSBL flow for initialization 不要打勾

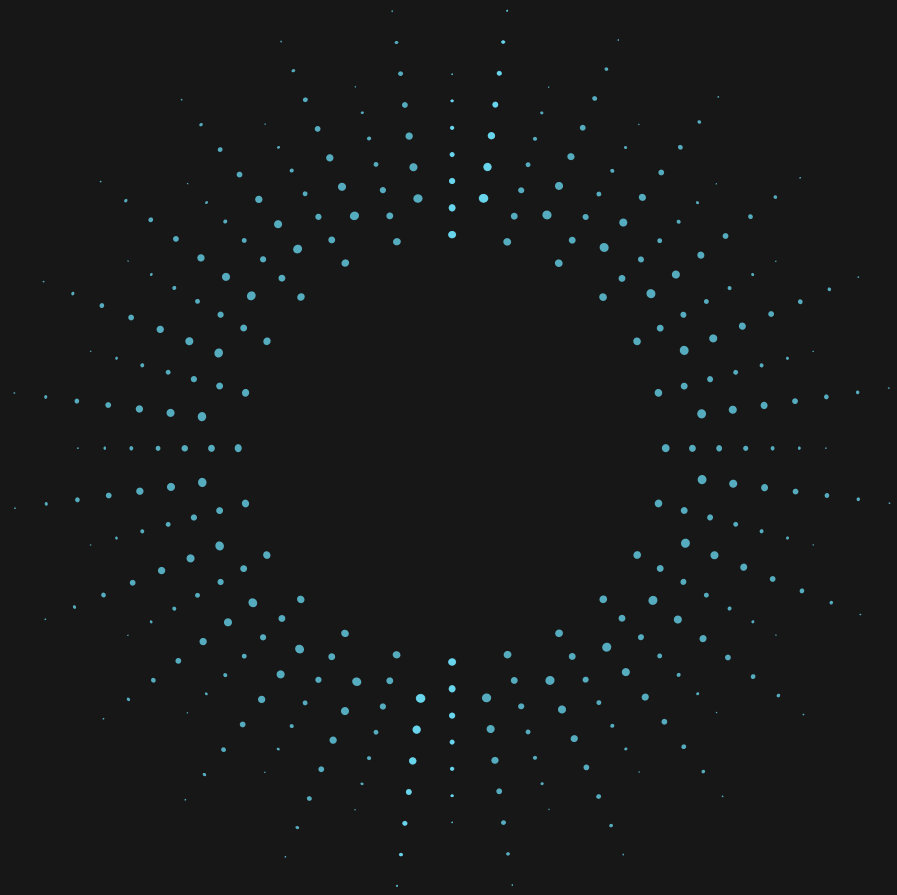


KD240 PWM LED

5. Program - 就成功了

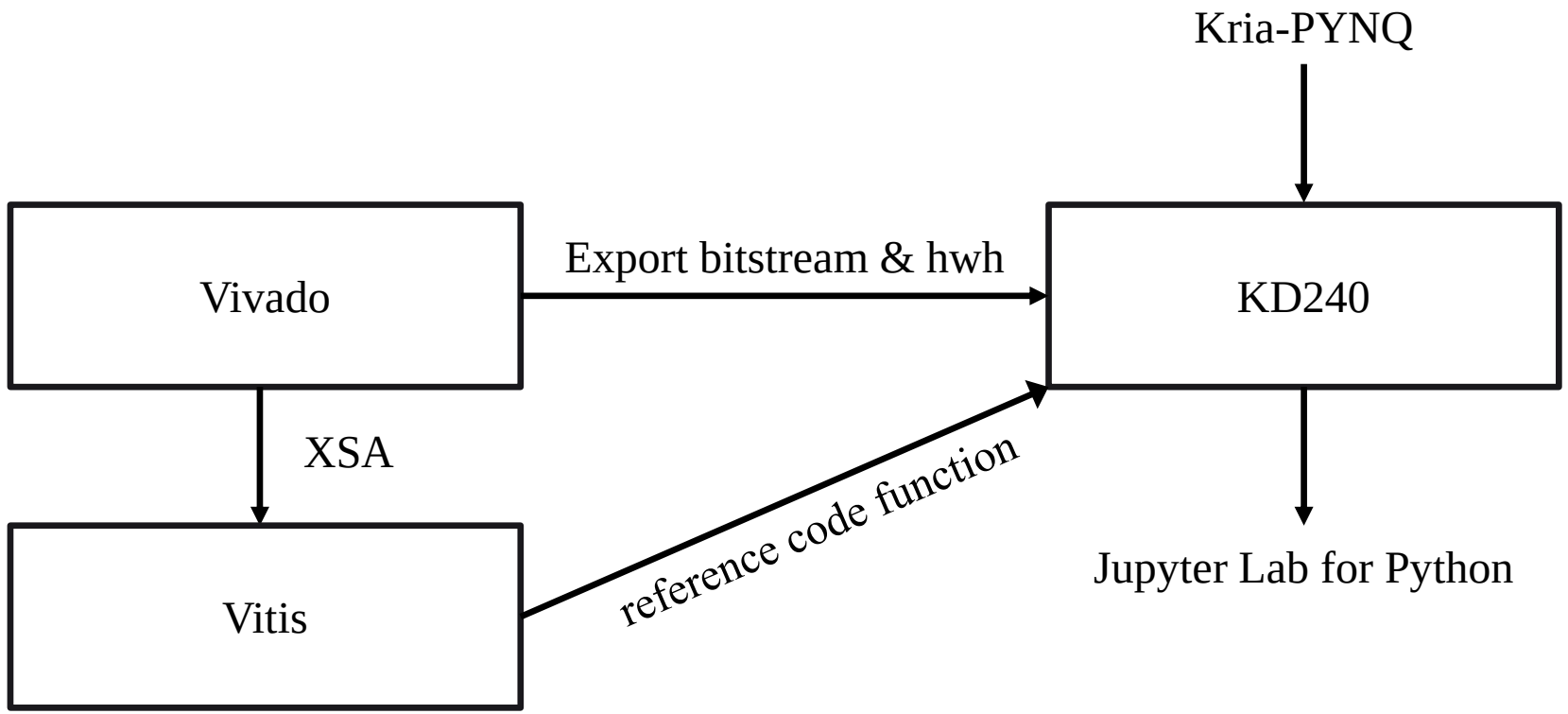


Using PYNQ overlay to light LED



Using PYNQ overlay to light LED

- 這邊開始要介紹如何透過 Python 控制在 Vivado 用 RTL Code 寫好的 IP
- 大致流程如下 (Python 控制 IP 的 function 需要參考 Vitis)



Using PYNQ overlay to light LED

1. Download Ubuntu 22.04 and put it in SDcard through balenaetcher
2. Boot Ubuntu 22.04 from KD240
3. sudo add-apt-repository ppa:xilinx-apps (Optional)
4. Update Ubuntu package
   - sudo apt update
   - sudo apt upgrade
5. Install Xilinx system management snap package ---> sudo snap install xlnx-config --classic (Optional)
6. git clone https://github.com/Xilinx/Kria-PYNQ.git
7. cd Kria-PYNQ
8. ~~更改 install.sh 241 行 ---> cp -r pynq_dpu/kd240_notebooks /root/jupyter_notebooks/ (官方已修正)~~
9. sudo bash install.sh -b KD240

Using PYNQ overlay to light LED

成功後出現以下訊息：

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
Collecting pytest
  Downloading pytest-7.4.4-py3-none-any.whl (325 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 325.3/325.3 KB 1.2 MB/s eta 0:00:00
Collecting tomli≥1.0.0
  Downloading tomli-2.0.1-py3-none-any.whl (12 kB)
Requirement already satisfied: packaging in /usr/local/share/pynq-venv/lib/python3.10/site-packages (from pytest)
Collecting pluggy<2.0,≥0.12
  Downloading pluggy-1.4.0-py3-none-any.whl (20 kB)
Collecting iniconfig
  Downloading iniconfig-2.0.0-py3-none-any.whl (5.9 kB)
Collecting exceptiongroup≥1.0.0rc8
  Downloading exceptiongroup-1.2.0-py3-none-any.whl (16 kB)
Installing collected packages: tomli, pluggy, iniconfig, exceptiongroup, pytest
Successfully installed exceptiongroup-1.2.0 iniconfig-2.0.0 pluggy-1.4.0 pytest-7.4.4 tomli-2.0.1
PYNQ Installation completed.


To continue with the PYNQ experience, connect to JupyterLab via a web browser using this url: 10.8.3.238:9090/lab
:9090/lab - The password is xilinx

root@kria:/home/root/Kria-PYNQ#
```

這時候在同網域下的瀏覽器輸入黃字部分的網址，即可開啟 Jupyter Lab

Using PYNQ overlay to light LED

Jupyter Lab

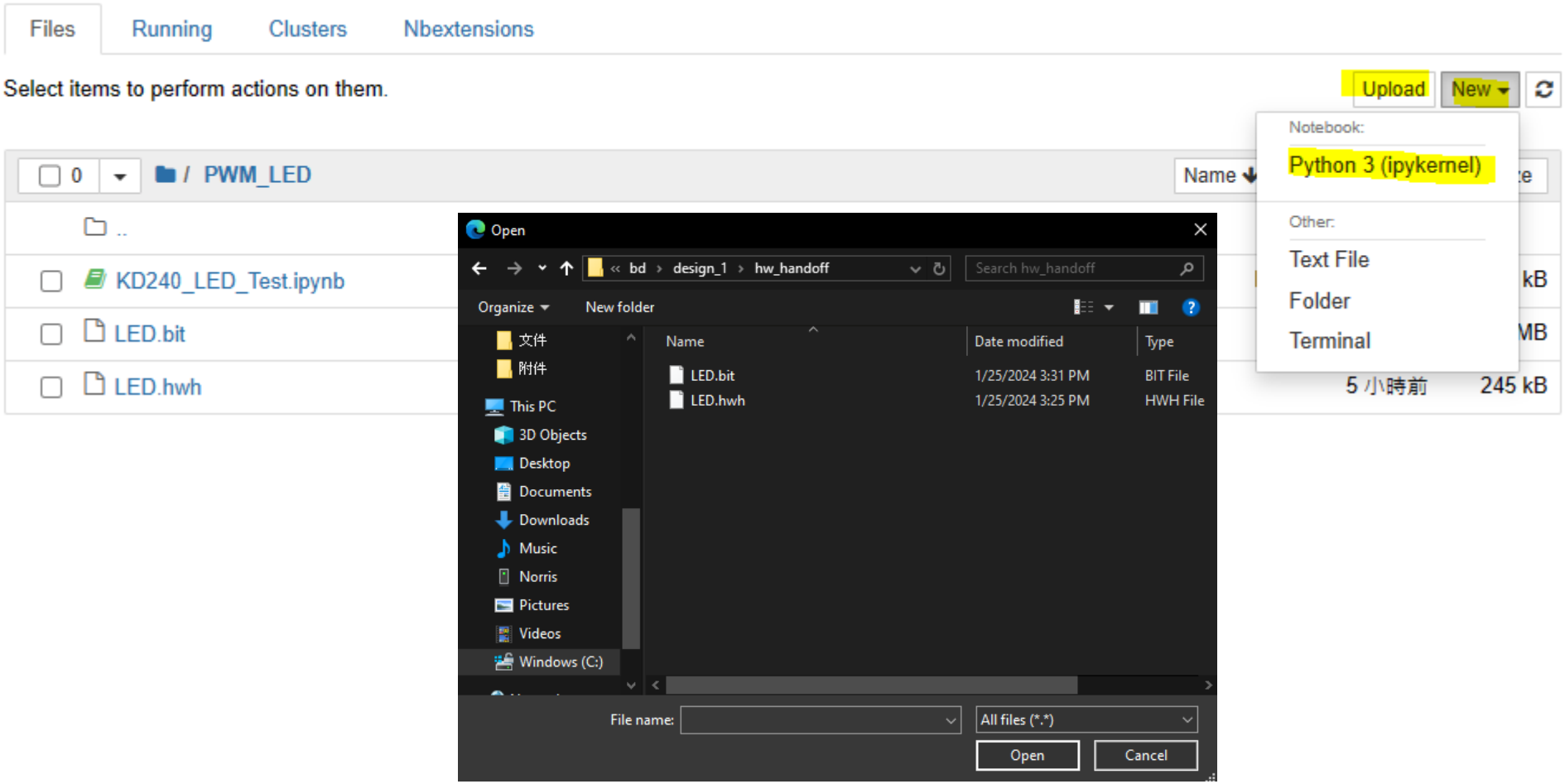
Using PYNQ overlay to light LED

建立一個名為 PWM_LED 的資料夾

Using PYNQ overlay to light LED

進入 PWM_LED 資料夾，將 Vivado 所產生的 .bit 以及 .hwh 皆放進來，並創建一新 python 文件

.gen\sources_1\bd\design_1\hw_handoff



Using PYNQ overlay to light LED

進入 python 文件開始撰寫，以下介紹 Coding 流程與 API 用法

首先要 import python 與 pynq 的 library package 進來

In [139]:
```
from pynq import Overlay
from pynq.lib import AxiGPIO
import time
from pynq import MMIO
import multiprocessing as mp
```

1. overlay 為 loading bitstream 時用到
2. AxiGPIO 針對設計內有用到 GPIO 的區塊可以進行 register 讀寫
3. time 做為 delay 時間控制用
4. MMIO 為 PS 控制 PL 端時讀寫 register 用，此篇則是以控制 PL PWM IP 為目的
5. multiprocessing 可以執行多個迴圈同時運作

Using PYNQ overlay to light LED

```
In [140]: overlay = Overlay("LED.bit")
```

Overlay loading bitstream，注意一定會需要 .bit 和 .hwh

```
In [141]: help(overlay)
Help on Overlay in module pynq.overlay:

<pynq.overlay.Overlay object>
    Default documentation for overlay LED.bit. The following
    attributes are available on this overlay:

    IP Blocks
    ----------
    pwm_0               : pynq.overlay.DefaultIP
    axi_gpio_0          : pynq.lib.axigpio.AxiGPIO
    zynq_ultra_ps_e_0   : pynq.overlay.DefaultIP

    Hierarchies
    -----------
    None

    Interrupts
    ----------
    None
```

help(overlay) 可以查看當前 .bit 內用到什麼 IP，架構如何，以此與 Vivado 內的設計相呼應

Using PYNQ overlay to light LED

初始化 IP

```
led_ip = overlay.ip_dict['axi_gpio_0']
pwm_ip = overlay.ip_dict['pwm_0']
```

因 KD240 PL ethernet 上的 LED 有兩組，因此要額外再設定 channel (只有一組時也要設定 channel1)

```
led_top = AxiGPIO(led_ip).channel1
led_down = AxiGPIO(led_ip).channel2
```

設定 GPIO direction，即 mask，0x0 為 input，0xf 為 output，並設定 LED 交互發光的 delay 數值 (秒為單位)

```
mask = 0xf
sec = 0.05
```

Using PYNQ overlay to light LED

撰寫 Ethernet LED 發亮動作時的 Function

```
def led_ethernet():
    led_top.write(1, mask)
    led_down.write(1, mask)
    time.sleep(1)

    while(1):
        led_top.write(0, mask)
        time.sleep(sec)

        if led_down.read() == 1:
            led_down.write(1, mask)
        else:
            led_down.write(1, mask)
            time.sleep(sec)

        led_top.write(2, mask)
        time.sleep(sec)
        led_top.write(3, mask)
        time.sleep(sec)

        led_down.write(3, mask)
        time.sleep(sec)
        led_top.write(1, mask)
        time.sleep(sec)
        led_down.write(2, mask)
        time.sleep(sec)
        led_down.write(0, mask)
        time.sleep(sec)
```

PL ethernet 亮燈數值為

0: 亮右邊

1: 全不亮

2: 全亮

3: 亮左邊

Using PYNQ overlay to light LED

設定 PWM IP 的記憶體和 Register 數值

```
base_address = 0xA0000000
address_range = 0x10000
REG0 = 0x00
REG1 = 0x04
```

這幾點實質可以參照 Vitis 以及 Vivado 內的 Address Editor

Diagram | Address Editor

Assigned (2) | Unassigned (0) | Excluded (0) | Show All

Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/zynq_ultra_ps_e_0					
/zynq_ultra_ps_e_0/Data (40 address bits : 0x00A0000000 [256M] ,0x0400000000 [4G] ,0x1000000000 [224G] ,0x00B0000000 [256M] ,0x0500000000 [					
/axi_gpio_0/S_AXI	S_AXI	Reg	0xA001_0000	64K	0xA001_FFFF
/pwm_0/S00_AXI	S00_AXI	S00_AXI_reg	0xA000_0000	64K	0xA000_FFFF

```
/* Definitions for peripheral PWM_0 */
#define XPAR_PWM_0_DEVICE_ID 0
#define XPAR_PWM_0_S00_AXI_BASEADDR 0xA0000000
#define XPAR_PWM_0_S00_AXI_HIGHADDR 0xA000FFFF
```

```
#define PWM_S00_AXI_SLV_REG0_OFFSET 0
#define PWM_S00_AXI_SLV_REG1_OFFSET 4
#define PWM_S00_AXI_SLV_REG2_OFFSET 8
#define PWM_S00_AXI_SLV_REG3_OFFSET 12
```

Using PYNQ overlay to light LED

撰寫 PWM IP Function

```
def led_pwm():
    mmio = MMIO(base_address, address_range)
    mmio.write(REG0, 17179)

    while(1):
        for duty in range(17179, 9999999, 5000):
            mmio.write(REG1, duty)
            time.sleep(0.1)
```

MMIO 會對記憶體位置去讀寫數值，基本上跟 Vitis 的邏輯概念一樣，可以照搬前面章節 Vitis C Code 改為 Python Code

Using PYNQ overlay to light LED

為了讓兩個迴圈同時動作，這邊利用 Multi-Process 將兩個 Function 加入到不同的 Process 中

並且同步運行

```
p1 = mp.Process(target=led_ethernet)
p2 = mp.Process(target=led_pwm)

p1.start()
p2.start()

p1.join()
p2.join()
```

Using PYNQ overlay to light LED

Result

Reference

- 【ZYNQ Ultrascale+ MPSOC FPGA教程】第三十章 自定义IP实验 - ALINX官方博客 - 博客园 (cnblogs.com)
- Bootmodes — Kria™ SOM 2022.1 documentation (xilinx.github.io)
- Development process and questions about KR260 standalone (xilinx.com)
- ug1093-kd240-starter-kit.pdf • 查看器 • AMD 自适应计算文档门户 (xilinx.com)
- Loading an Overlay — Python productivity for Zynq (Pynq) v1.0
- pynq.lib.axigpio Module — Python productivity for Zynq (Pynq) v1.0
- pynq框架下自定义ip的使用_pynq ip-CSDN博客


AMD