FRONTGRADE

PROGRAMING MRAM

UT24CP1008

CertusPro-NX-RT FPGA

4/7/2026

Version #:1.0.0

Introduction

Frontgrade has created example user code for programming external non-volatile storage devices using Lattice Serial Peripheral Interface (SPI) Flash Memory Controller IP. The process is comprised of a set of user modules for interfacing to Lattice SPI Flash IP. The user code interfaces to the IP to download Lattice FPGA image and store it in the external non-volatile memory device.

The IP provides an interface between the user code and an external non-volatile memory device. AHB-Lite interface is used to configure the IP through the IP Graphical User Interface (GUI), and the APB interface is used to issue any flash operations. This IP receives and processes data or control signals from user code, and then translates each flash command into serial form to be communicated to the external non-volatile memory device.

For details on Lattice's Flash Memory Controller IP, refer to FPGA-IPUG-02134-2-1-SPI-Flash-Memory-Controller-IP-User Guide.pdf.

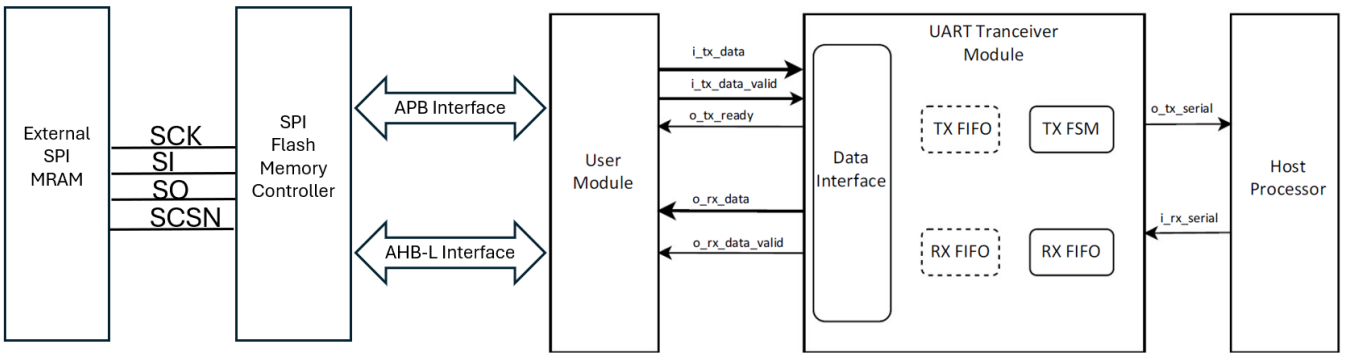
Note: The hex file was created from the accompanying **ADC app note** and following **Global** settings:

- MASTER_SPI_PORT: SERIAL
- JTAG_PORT: ENABLE
- PROGRAMN_PORT: DISABLE
- MCCLK_FREQ: 3.5

This app note goes through the following steps:

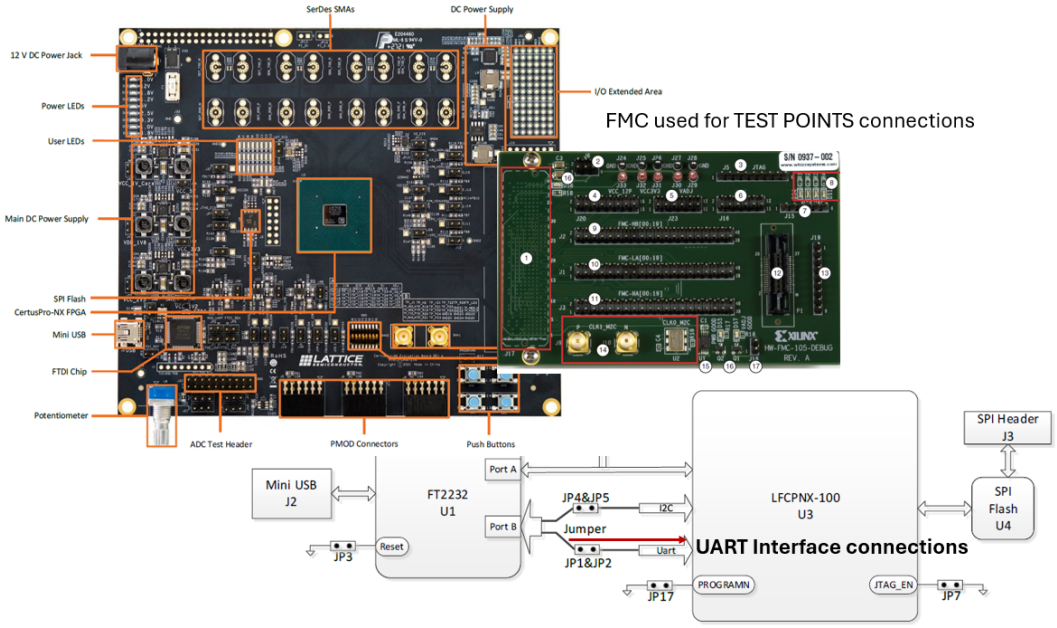
1. Use Lattice Tools to create the hex file.
2. Send intel hex file using <Intel Hex file name>.ttl and Terminal.

   Terminal > Control > Macro > <Intel Hex file name>.ttl.
3. User code receives the hex file line by line and assembles one memory page.
4. The SPI Flash Memory Controller sends one memory page at a time to MRAM until the image is complete.



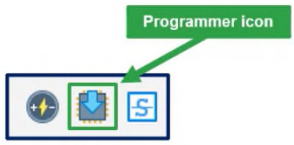
CertusPro-NX Device

CertusPro-NX Evaluation Board is the LFCPNX-100-9LFG672C. The Eval Board can be used in a wide range of applications. For more information on the capabilities of CertusPro-NX device, see CertusPro-NX Family Data Sheet (FPGA-DS-02086).

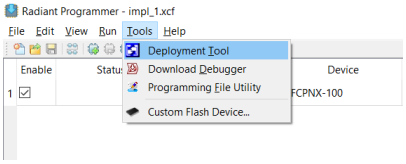


1.1 Preparing MRAM Image from CertusPro-NX-RT Bit File

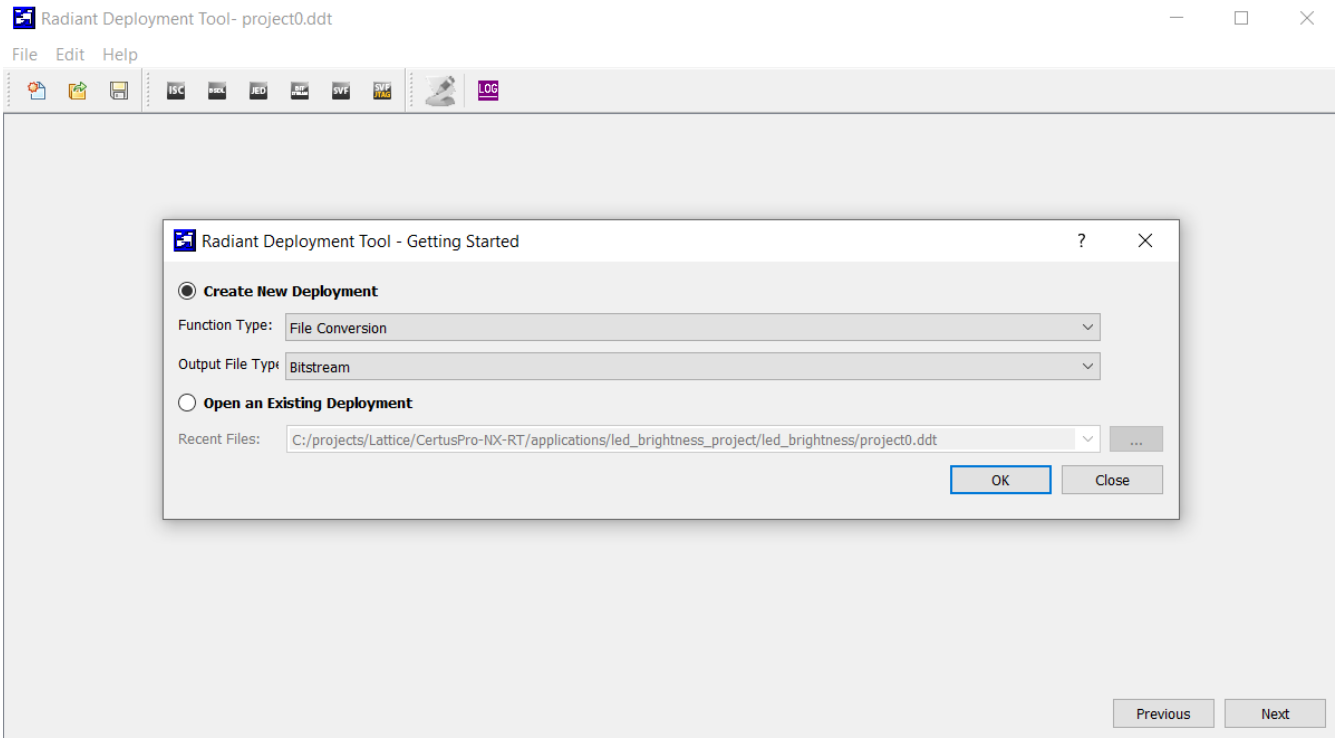
The Programmer tool from Lattice can be used to create the hex file. Launch the programming tool, see picture below.



Under Tools, click on Deployment Tool, see picture below.

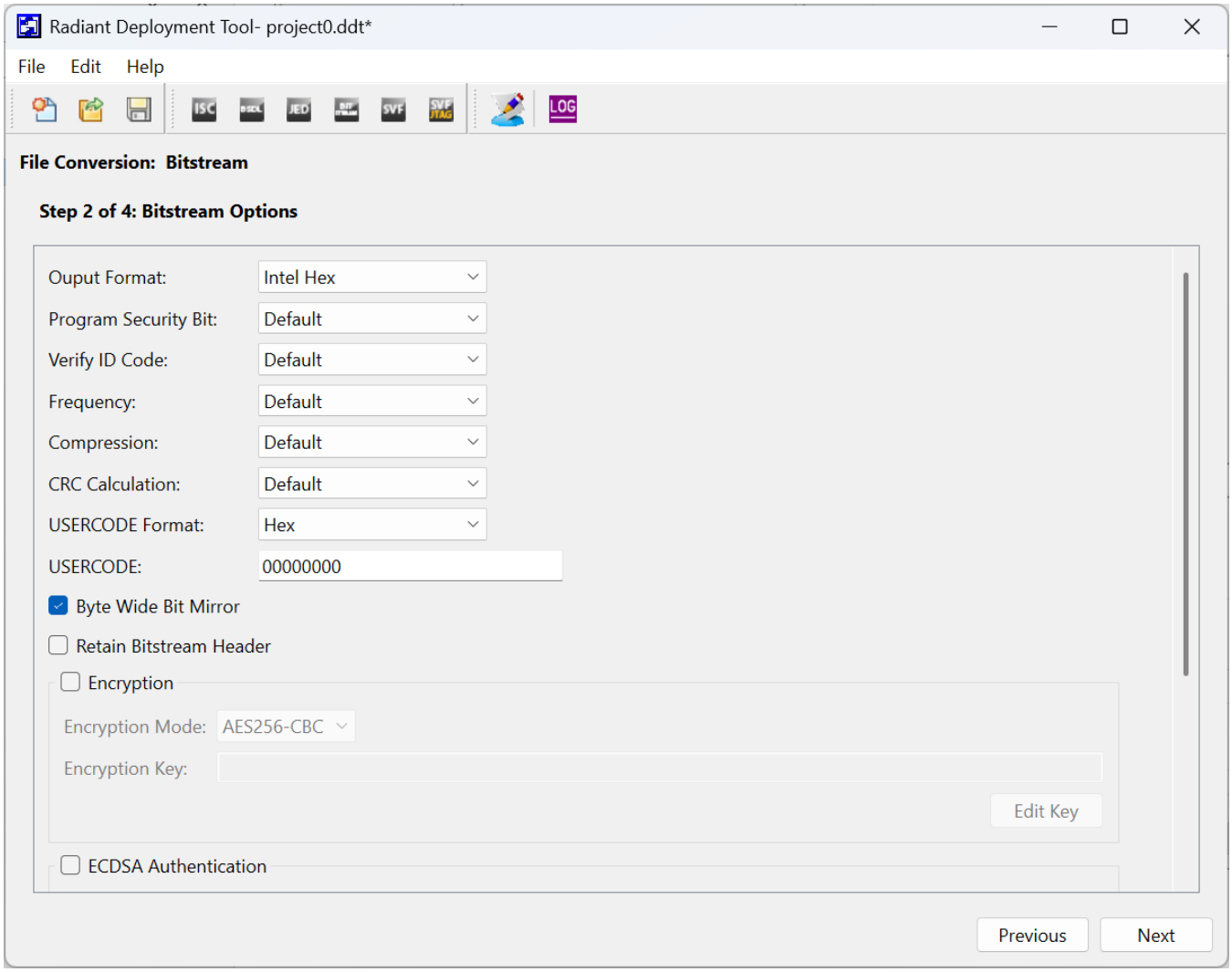


In the Radiant Deployment Tool, under Create New Deployment, choose the Function Type and Output File Type and click OK, see picture below.

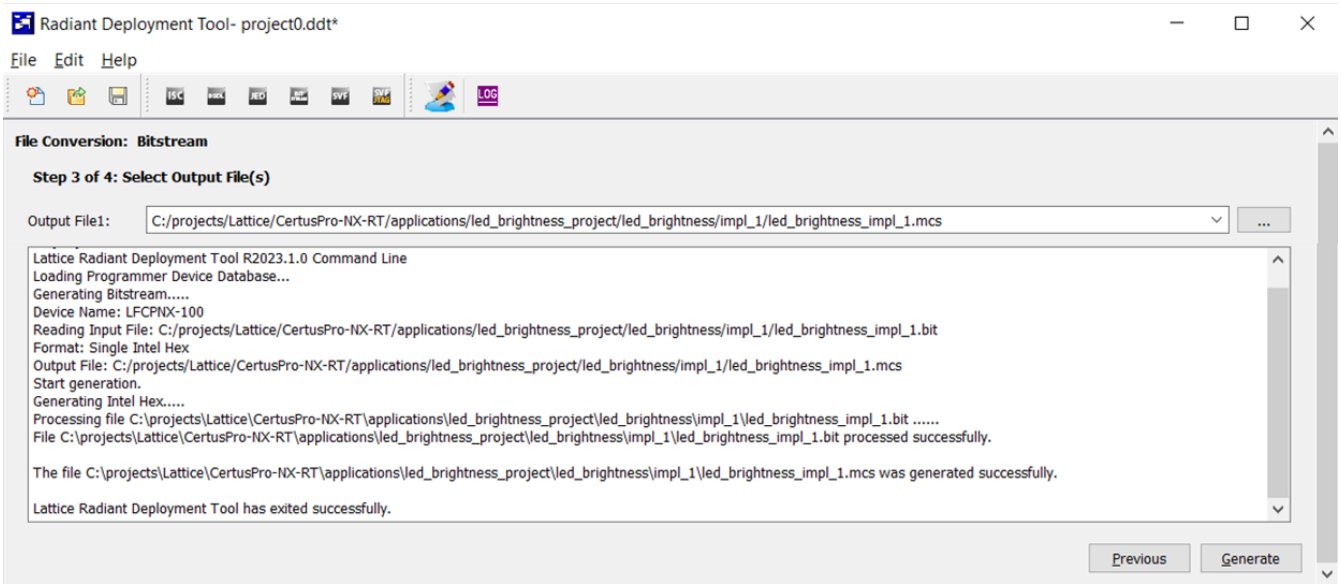


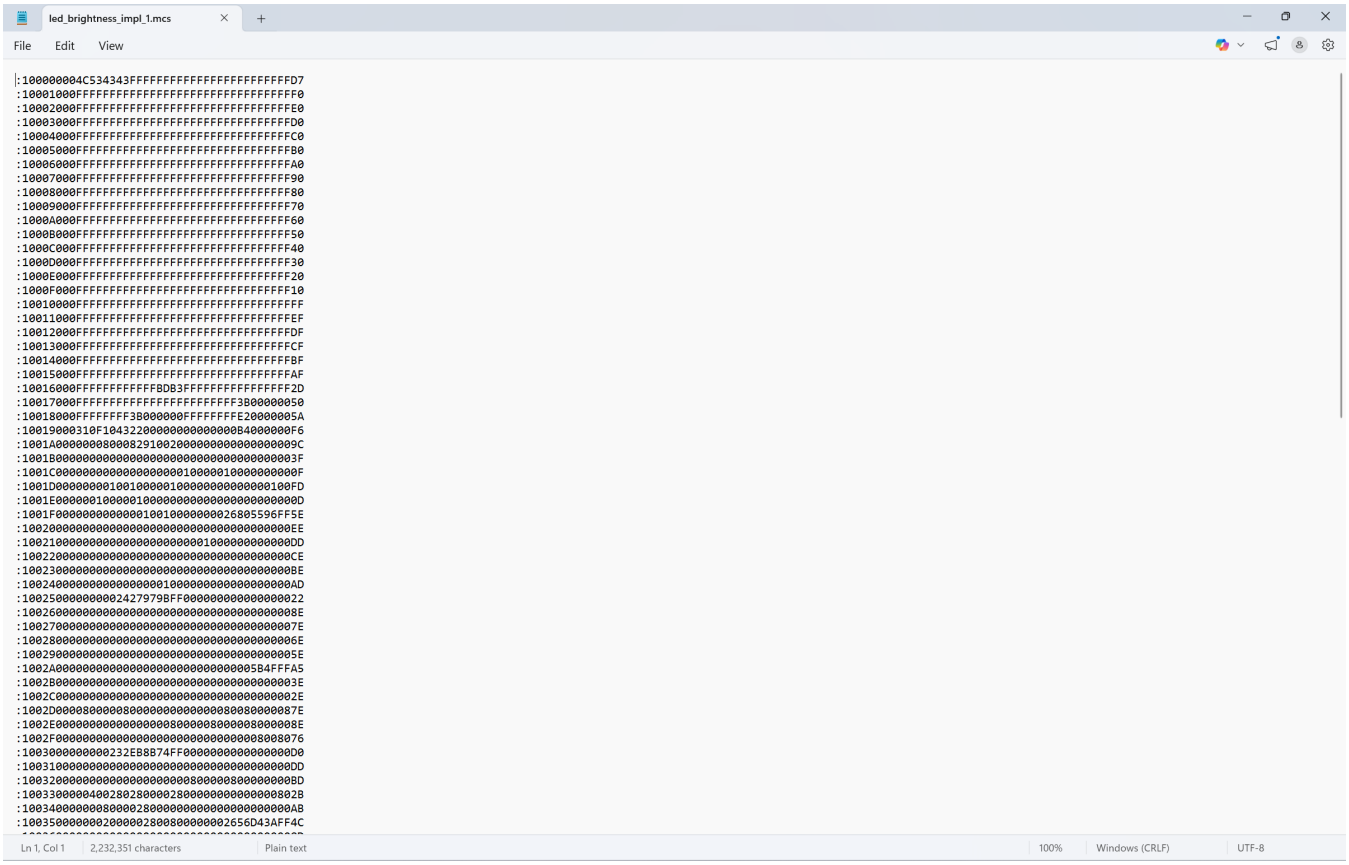
Under File Conversion, choose the bit file and click Next, see picture below.

Under File Conversion, choose the Output Format and click Next, see picture below.



Verify the output file, click Next, and finally generate the hex file, see picture below.





Below is Intel Hex Format for references.

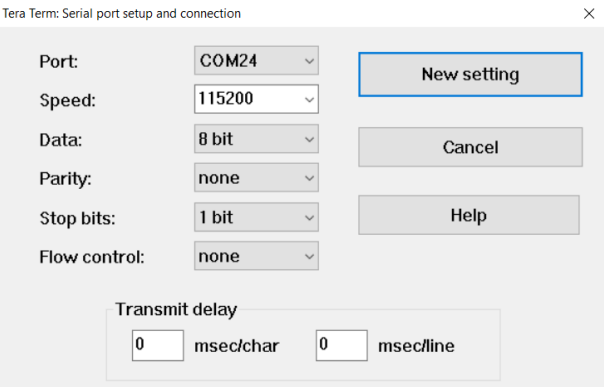
```
:10010000214601360121470136007EFE09D2190140
:10011000214601 7E17C20001FF5F16002148011928
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

Start code | Byte count | Address | Record type | Data | Checksum

Hex code	Record type	Description	Example
00	Data	Contains data and a 16-bit starting address for the data. The byte count specifies number of data bytes in the record. The example shown to the right has 0B (eleven) data bytes (61, 64, 64, 72, 65, 73, 73, 20, 67, 61, 70) located at consecutive addresses beginning at address 0010.	:0B0010006164647265737320676170A7
01	End Of File	Must occur exactly once per file in the last line of the file. The data field is empty (thus byte count is 00) and the address field is typically 0000.	:00000001FF
02	Extended Segment Address	The data field contains a 16-bit segment base address (thus byte count is always 02) compatible with 80x86 real mode addressing. The address field (typically 0000) is ignored. The segment address from the most recent 02 record is multiplied by 16 and added to each subsequent data record address to form the physical starting address for the data. This allows addressing up to one megabyte of address space.	:020000021200EA
03	Start Segment Address	For 80x86 processors, specifies the initial content of the CS:IP registers (i.e., the starting execution address). The address field is 0000, the byte count is always 04, the first two data bytes are the CS value, the latter two are the IP value.	:0400000300003800C1
04	Extended Linear Address	Allows for 32 bit addressing (up to 4GiB). The record's address field is ignored (typically 0000) and its byte count is always 02. The two data bytes (big endian) specify the upper 16 bits of the 32 bit absolute address for all subsequent type 00 records; these upper address bits apply until the next 04 record. The absolute address for a type 00 record is formed by combining the upper 16 address bits of the most recent 04 record with the low 16 address bits of the 00 record. If a type 00 record is not preceded by any type 04 records then its upper 16 address bits default to 0000.	:02000004FFFFFC
05	Start Linear Address	The address field is 0000 (not used) and the byte count is always 04. The four data bytes represent a 32-bit address value (big-endian). In the case of 80386 and higher CPUs, this address is loaded into the EIP register.	:0400000500000CD2A

1.2 Upload Intel Hex File to External Non-volatile Memory using Terminal

Before sending the hex file, Tera Term needs to be set to 115200 Baud rate and the following settings.



Send intel hex file using ut32_can_upload.ttl and Terminal
Terminal > Control > Macro > ut32_can_upload.ttl

```
; Open a file.
fileopen fhandle 'led_brightness_impl_1.mcs' 0

while 1
;for i 1 32
    ; Read a line from the file.
    filereadln fhandle line
    if result=1 then
        break
    endif
    ; Display the line.
    sendln line
    wait 'z'
    ;wait 't444'
    ;wait #13
    ;break
    ; Repeat until the end of the file.
;next
endwhile


; Close the file.
fileclose fhandle
```

1.3 User code receives the hex file line by line and assembles one memory page.

Data received from the UART is parsed and saved to a buffer. Each line in the Intel Hex Format contains thirty-two characters, which two characters combine to make one byte. The code saves 512 characters to make a page of 256 bytes.

The 256 bytes are saved to a buffer The SPI flash controller takes the 256 bytes and writes them to the external SPI MRAM.

```
            else begin
                rcvd_line_cnt      <= 0;
                send_resp_val  <= 2;
                send_resp_type <= RESP_DATA;
                state          <= SEND_RESP;
            end
        end else if ( (rcvd_line_cnt == 10'd8) &  (rx_data[6:0] == 8'h31) ) begin
            eof_done <= 1'b1;
            rcvd_line_cnt <= 10'd41;
        end else if ( (rcvd_line_cnt == 10'd8) &  ((rx_data[6:0] == 8'h32)|||(rx_data[6:0] == 8'h33)|||(rx_data[6:0] == 8'h34)|||(rx_data[6:0] == 8'h35) ))
            rcvd_line_cnt <= 10'd41;
        end else begin
            rcvd_line_cnt      <= rcvd_line_cnt + 1;
            if(rcvd_line_cnt >= 9 && rcvd_line_cnt < 41)begin
                //rx_line_valid <= 1;
                if (char_2_hex == 0)begin
                    data_8b_hi <= rx_data[6:0];
                end
                else begin
                    image_page[page_ptr*8 +: 8]   <= {ascii_to_hex(data_8b_hi),ascii_to_hex(rx_data[6:0])};
                    page_ptr      <= page_ptr + 1;
                end
                char_2_hex <= ~char_2_hex;
          end
        end
    end


end
SEND_RESP: begin
    if( new_char_check )begin
      if (send_resp_type == RESP_OK)
      begin
        case (char_cnt) // synthesis full_case
          0 : tx_data <= "-";
          2 : tx_data <= "O";
     //apb_wdata_o   <= 32'h100; // PAGE_PROGRAM_SIZE               = 256;
     init_sm_cs    <= ST_INIT_WRITE;
     apb_psel_o <= 1'b1;
     apb_penable_o<=0;
     apb_wr_rdn_o  <= 1'b1;

     if(flash_wr_rdn) begin
         apb_offset_o  <= PAGE_RD_LEN_0;
         page_prg_cmds          <= 3'b110;
     end
     else begin
         apb_offset_o  <= PAGE_PRG_LEN_0; //Set the page program length
         page_prg_cmds          <= 3'b011;
     end

     apb_wdata_o   <= 32'h100;// PAGE_PROGRAM_SIZE               = 256;
     data_8b_2d_32b   <= 0;
     page_ptr      <= 0;

 end
 // PAGE_PRG_BUF; //Generate word data
 3'b011 : begin

     init_sm_cs    <= ST_INIT_WRITE;
     apb_psel_o <= 1'b1;
     apb_penable_o<=0;
     apb_wr_rdn_o  <= 1'b1;
     apb_offset_o  <= page_ptr*4 + PAGE_PRG_BUF; //Generate word data
     apb_wdata_o   <= image_page[page_ptr*32 +: 32];
     page_ptr      <= page_ptr + 1;

     if(page_ptr>=64)begin
         page_prg_cmds          <= 3'b100;
         //init_ptr      <= init_ptr + PAGE_PROGRAM_SIZE;
     end
 end
```

1.4 SPI Flash Memory Controller Sends One Memory Page to MRAM.

The APB bus is used to directly read from or write to any memory location within the SPI flash memory controller. It is used to define and set how SPI MRAM transactions are executed by the controller. For writing one page to Flash, the following APB interface register accesses were used to perform this transaction:

1. Register write to '0x040' with '32-bit flash address'.
2. Register write to '0x044' to set the Flash Addressing mode to 32-bits.
3. Register write to '0x104' to set the page program length.
4. Register write to '0x200' to set the Page Program Buffer register.
5. Finally, register write to '0x000', Page Program register, to write the page to MRAM.

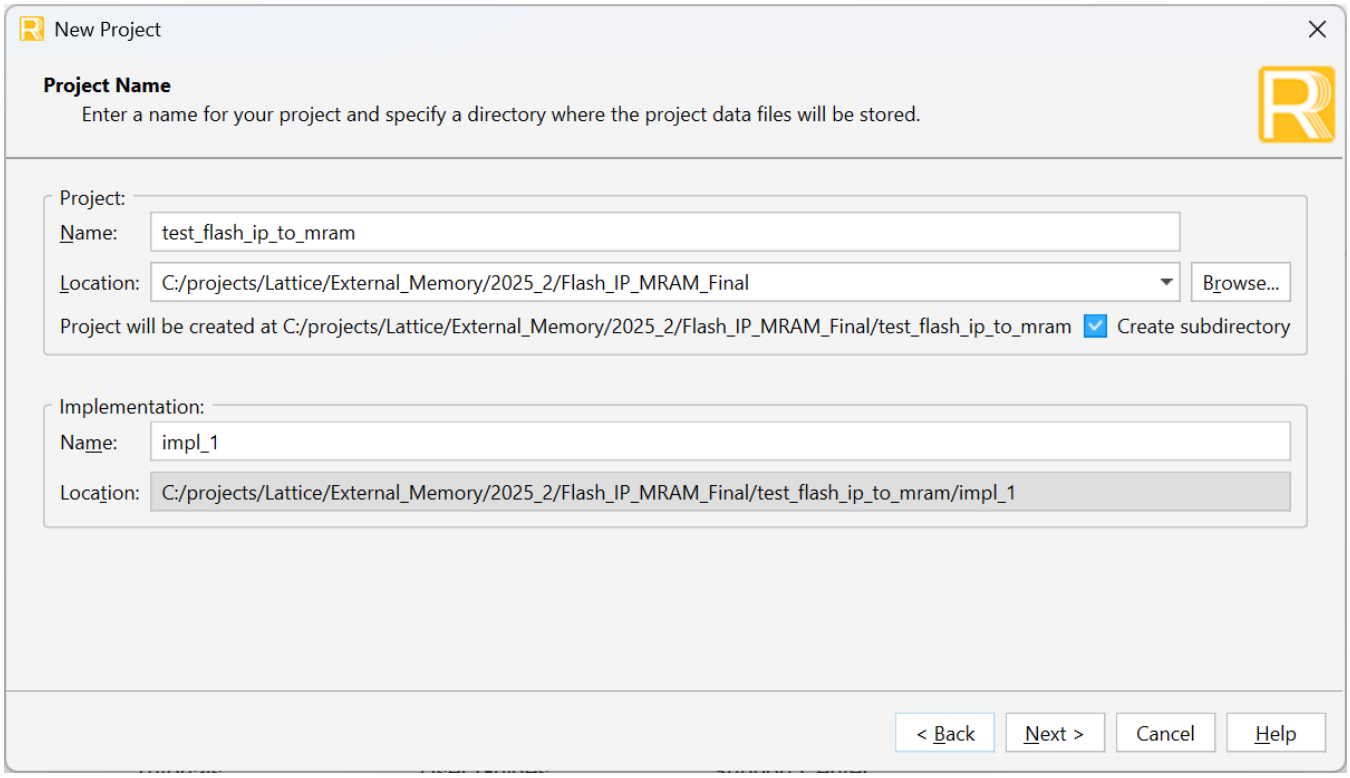
```
case (page_prg_cmds[2:0])
// SPI_ADDR_BYTE0; //Set the spi flash address
3'b000: begin
    init_sm_cs    <= ST_INIT_WRITE;
    apb_request_o <= 1'b1;
    apb_wr_rdn_o  <= 1'b1;
    page_prg_cmds          <= 3'b001;
    apb_offset_o  <= SPI_ADDR_BYTE0; //Set the spi flash address
    apb_wdata_o   <= init_ptr;//32'h00;
end
// Core Volts on ADC0
3'b001: begin
    init_sm_cs    <= ST_INIT_WRITE;
    apb_request_o <= 1'b1;
    apb_wr_rdn_o  <= 1'b1;
    page_prg_cmds          <= 3'b010;
    apb_offset_o  <= SPI_ADDR_MODE; //Set the flash addressing mode to 24-bit or 32-bit mode
    apb_wdata_o   <= 32'h01;
end
// SPI_ADDR_MODE; //Set the flash addressing mode to 24-bit or 32-bit mode
3'b010: begin
    //page_prg_cmds        <= 3'b011;
    //apb_offset_o  <= PAGE_PRG_LEN_0; //Set the page program length
    //apb_wdata_o   <= 32'h100; // PAGE_PROGRAM_SIZE               = 256;
    init_sm_cs    <= ST_INIT_WRITE;
    apb_request_o <= 1'b1;
    apb_wr_rdn_o  <= 1'b1;

    page_prg_cmds          <= 3'b011;
    apb_offset_o  <= PAGE_PRG_LEN_0; //Set the page program length
    apb_wdata_o   <= 32'h100;// PAGE_PROGRAM_SIZE               = 256;
    data_8b_2d_32b   <= 0;
    page_ptr      <= 0;
    //tx_ready   <= 1;

end

 // PAGE_PRG_BUF; //Generate word data
 3'b011 : begin

     init_sm_cs    <= ST_INIT_WRITE;
     apb_psel_o <= 1'b1;
     apb_penable_o<=0;
     apb_wr_rdn_o  <= 1'b1;
     apb_offset_o  <= page_ptr*4 + PAGE_PRG_BUF; //Generate word data
     apb_wdata_o   <= image_page[page_ptr*32 +: 32];
     page_ptr      <= page_ptr + 1;

     if(page_ptr>=64)begin
         page_prg_cmds          <= 3'b100;
         //init_ptr      <= init_ptr + PAGE_PROGRAM_SIZE;
     end
 end
3'b100 : begin
    init_sm_cs    <= ST_INIT_WRITE;
    apb_request_o <= 1'b1;
    apb_wr_rdn_o  <= 1'b1;
    page_prg_cmds          <= 3'b000;
    apb_offset_o  <= PAGE_PRG; //Run the page program in SPI interface
    apb_wdata_o   <= 32'h01; // PAGE_PROGRAM_SIZE               = 256;
    init_last     <= 1;
end
```

Creating User Code Project with Radiant Software

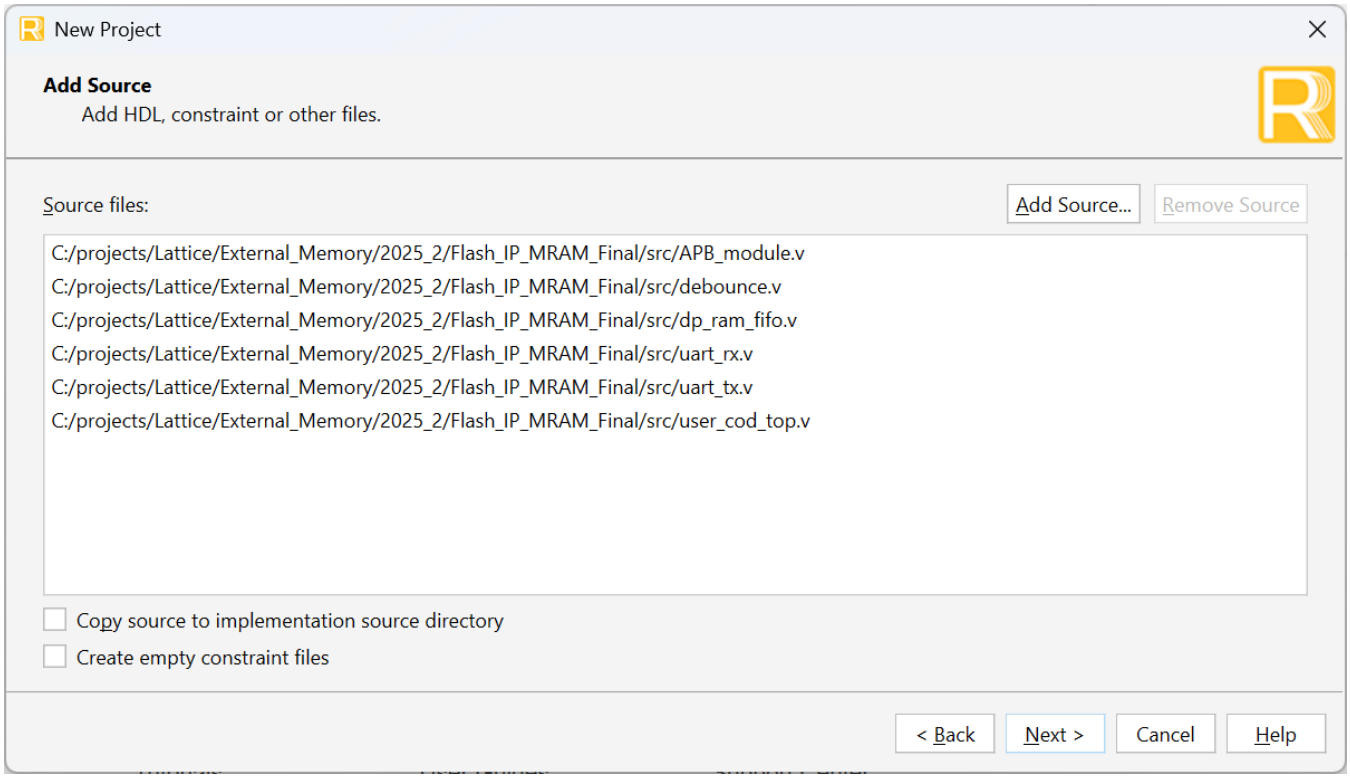
Launch Radiant Software 

From the File menu, select New > Project….

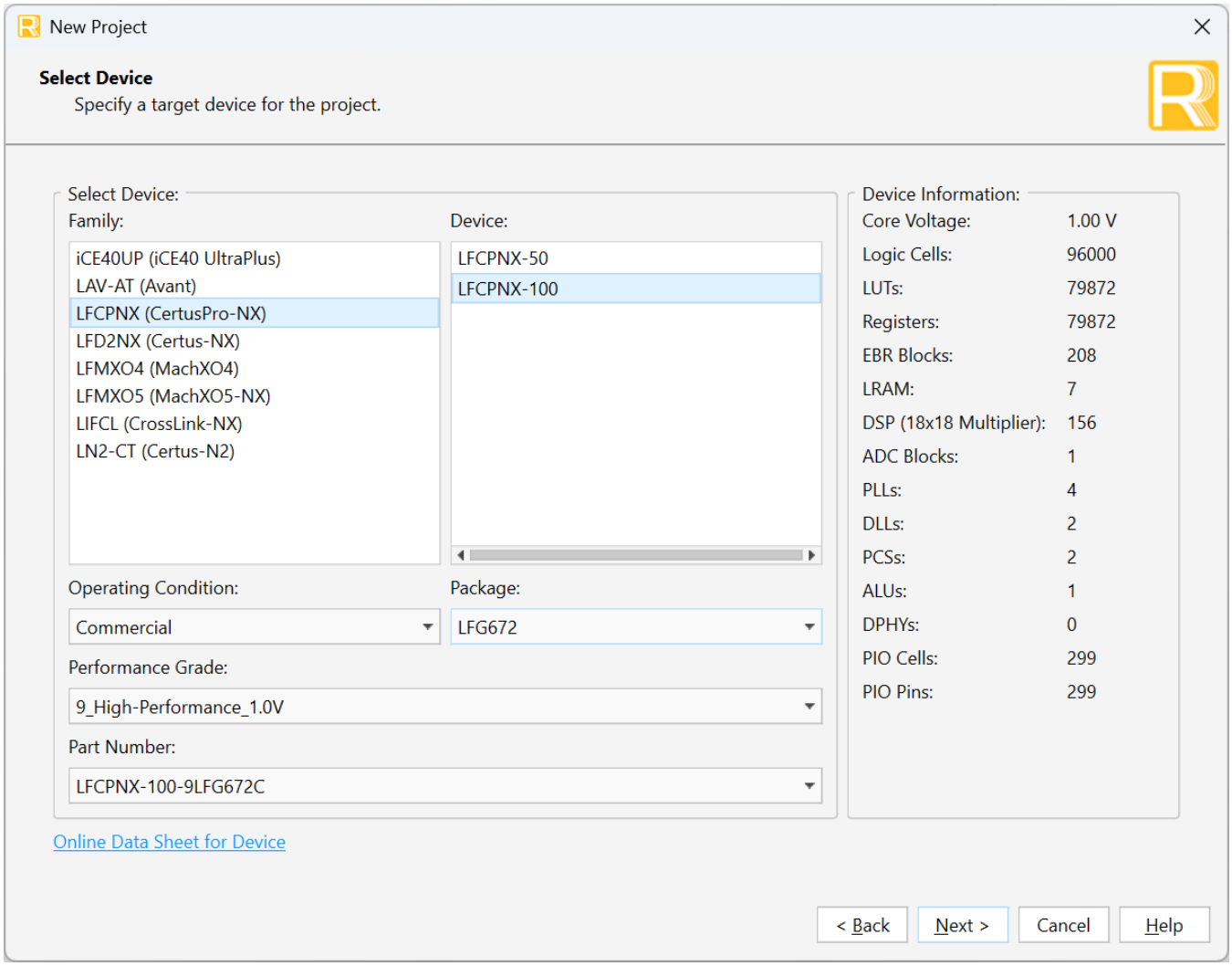
Specify the project name as test_flash_ip_to_mram, navigate to the location of your choice and click Next, see figure below.



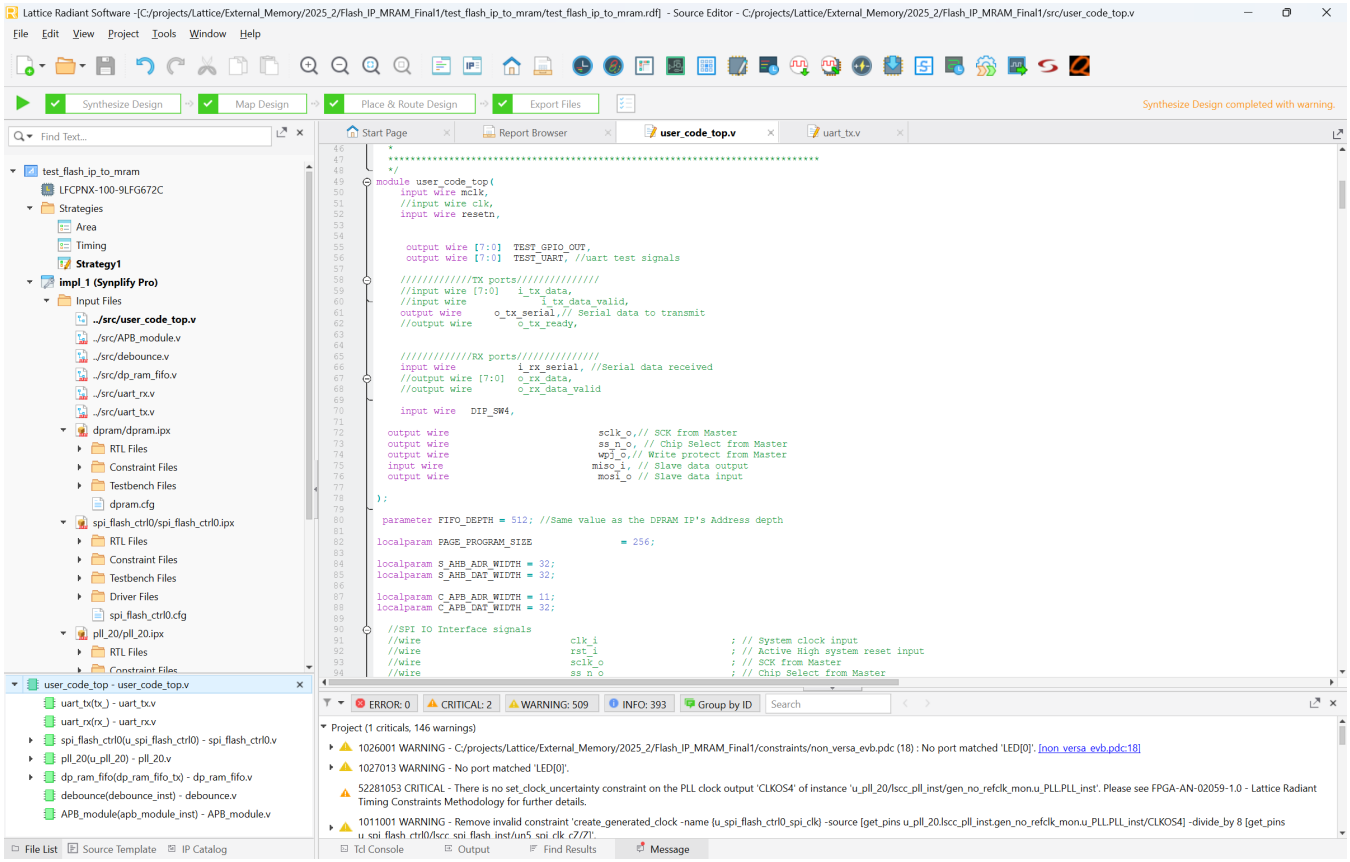
Select Add Source… and click Open, see picture below, click Next.



Select the appropriate device, click Next and choose Sinplify Pro; click Next and Finish, see picture below.

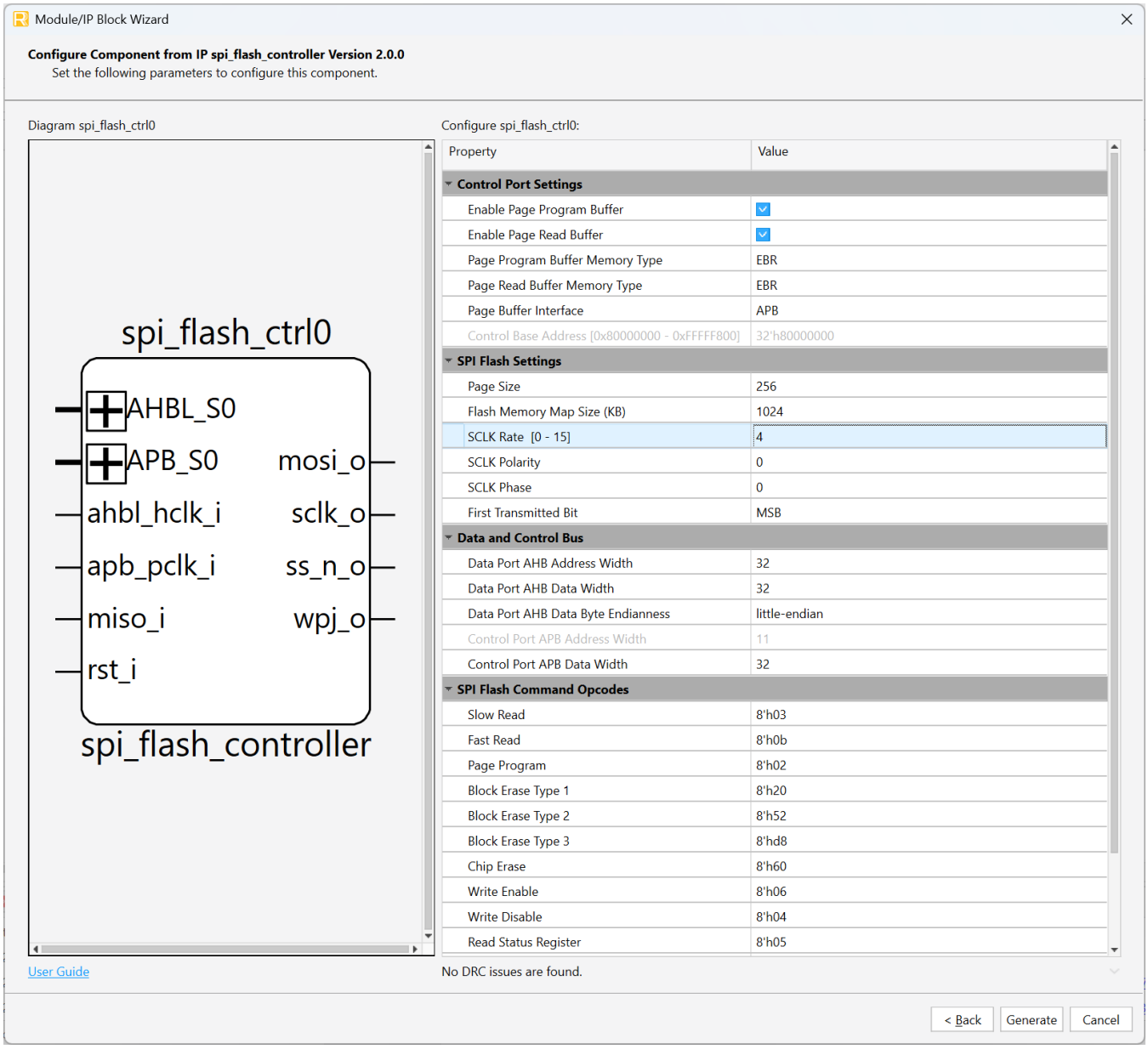


The initial project with provided files should look like the figure below.



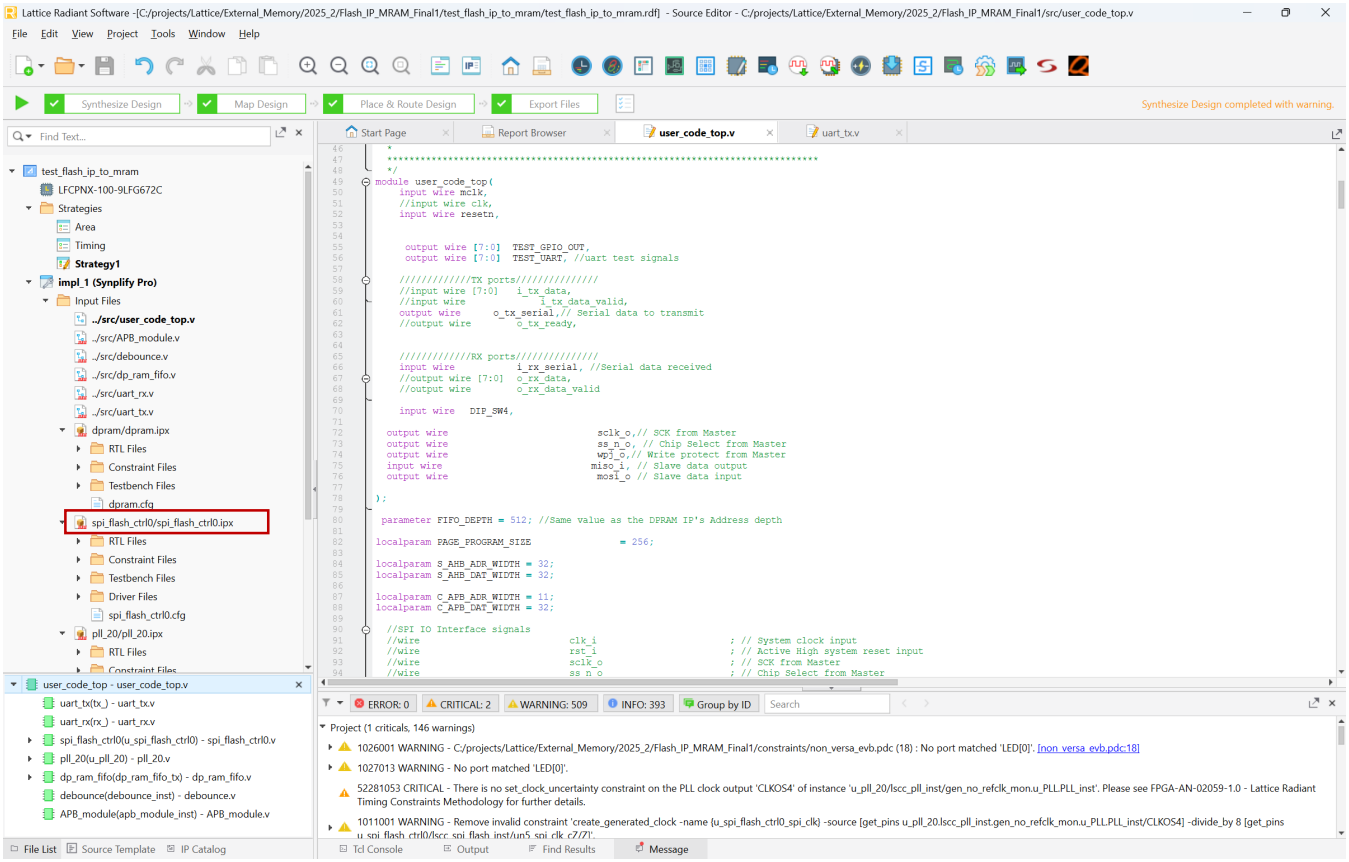
2.1 SPI Flash Controller IP

From the IP Catalog Architecture Modules, add a SPI Flash Controller IP module with the options shown below and name it spi_flash_ctrl0, see picture below.



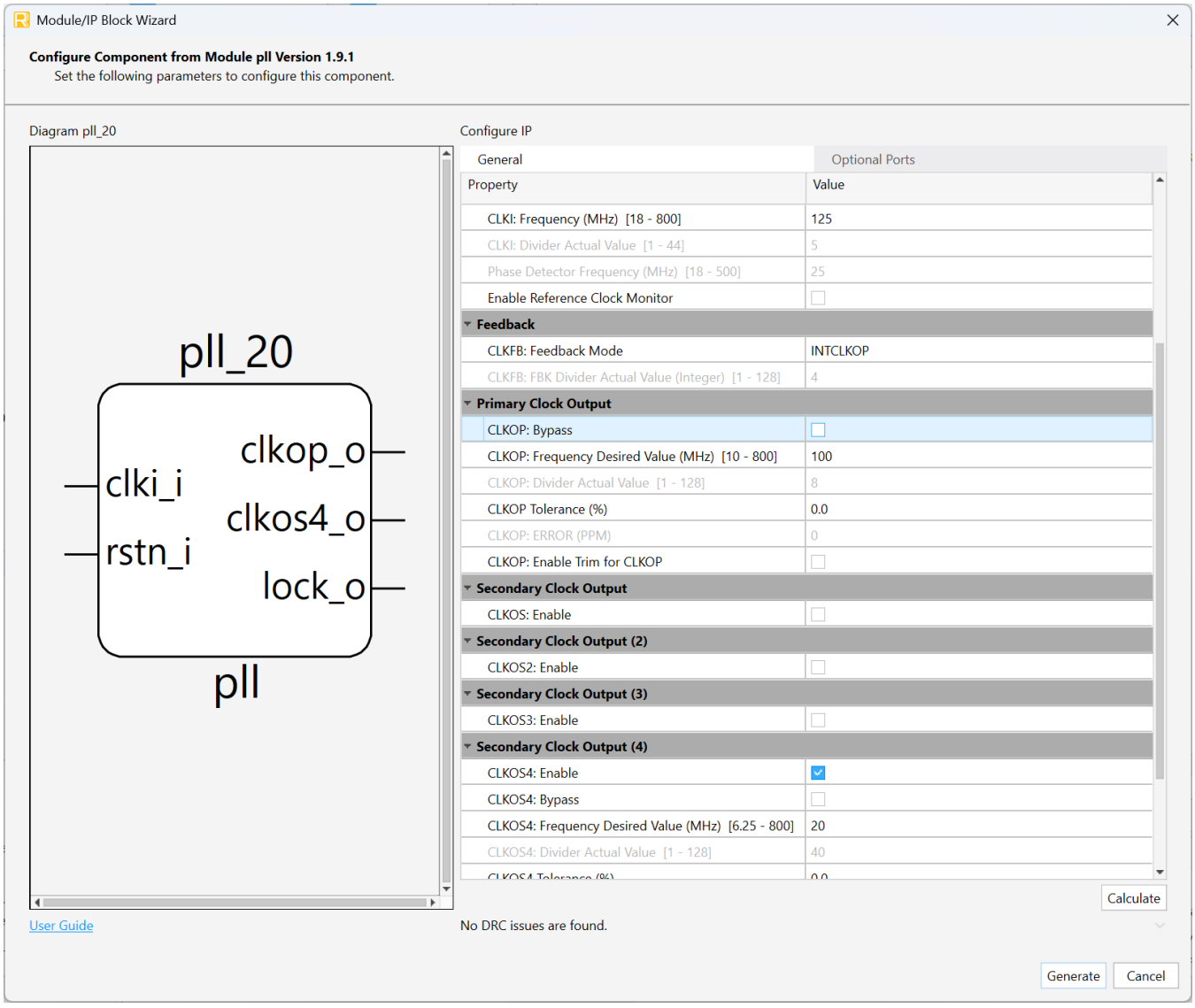
Once the spi_flash_ctrl0 options are entered, click the Generate button to start the creation of the module. On the next screen, click the Finish button to complete the creation of the module.

Once the spi_flash_ctrl0 is created, it is added to the project files, see picture below.



2.2 PLL IP

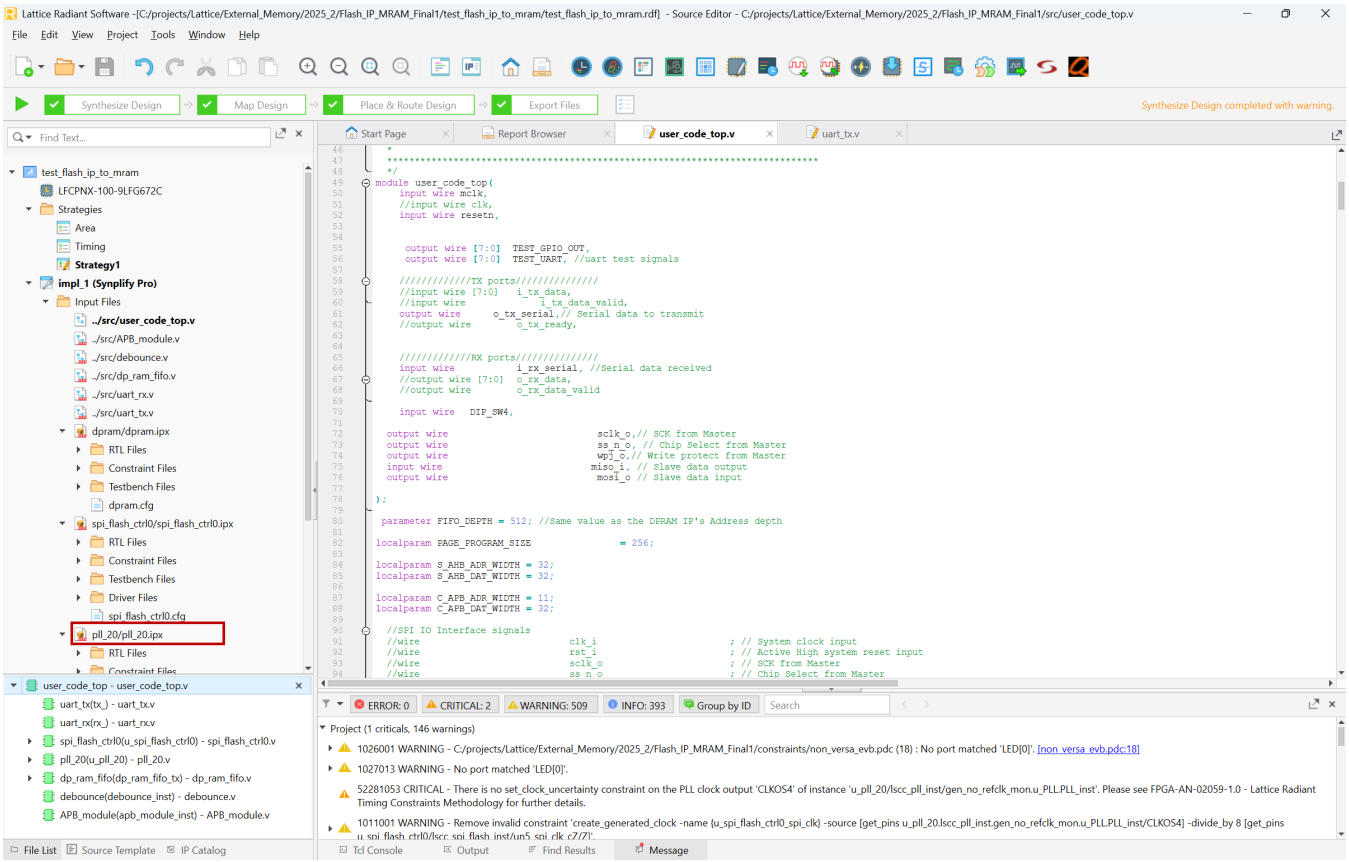
From the IP Catalog Architecture Modules, add a PLL module with the options shown below and name it pll_20, see picture below.



Once the PLL options are entered, click the Generate button to start the creation of the module.

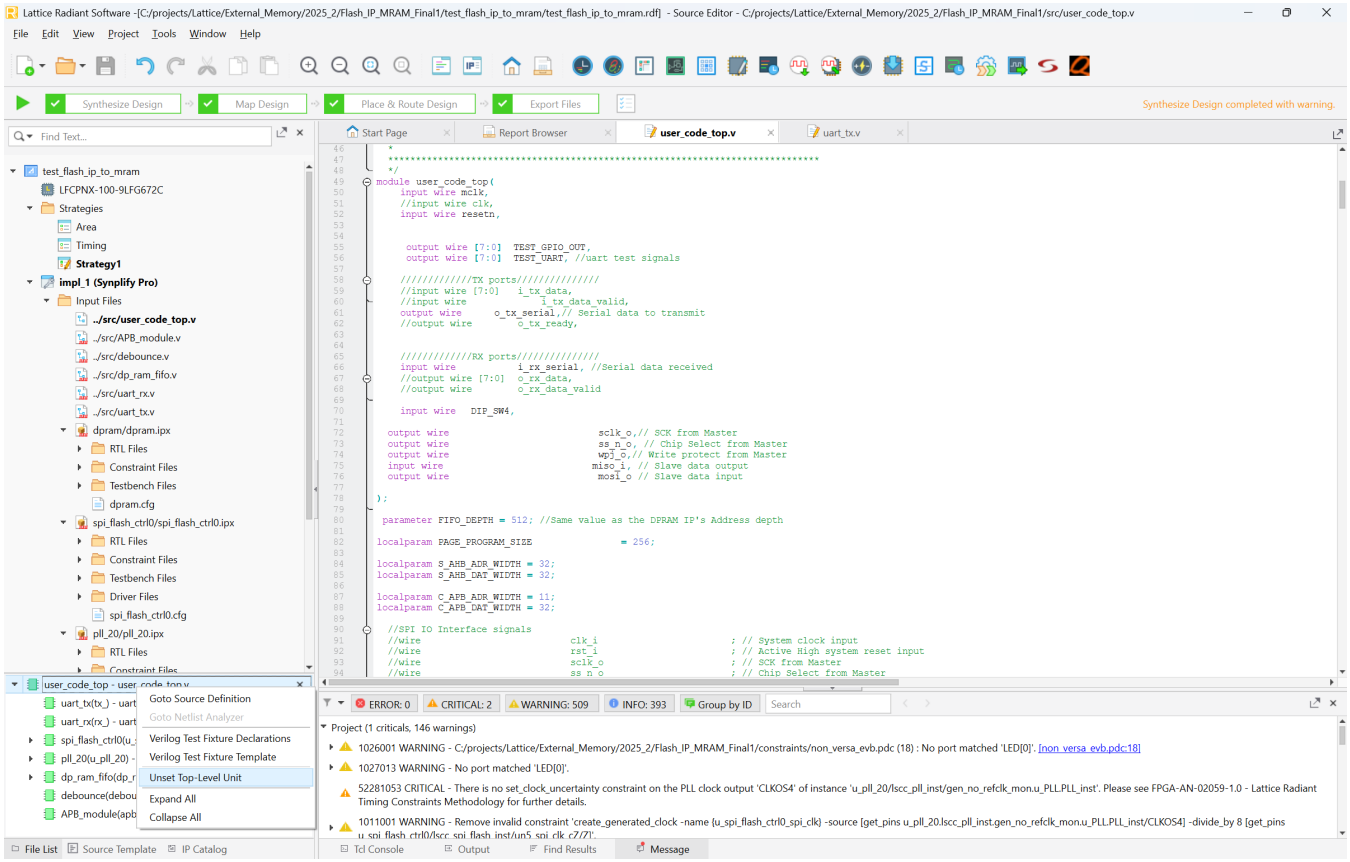
ON the next screen, click the Finish button to complete the creation of the module.

Once the PLL is created, it is added to the project files, see picture below.

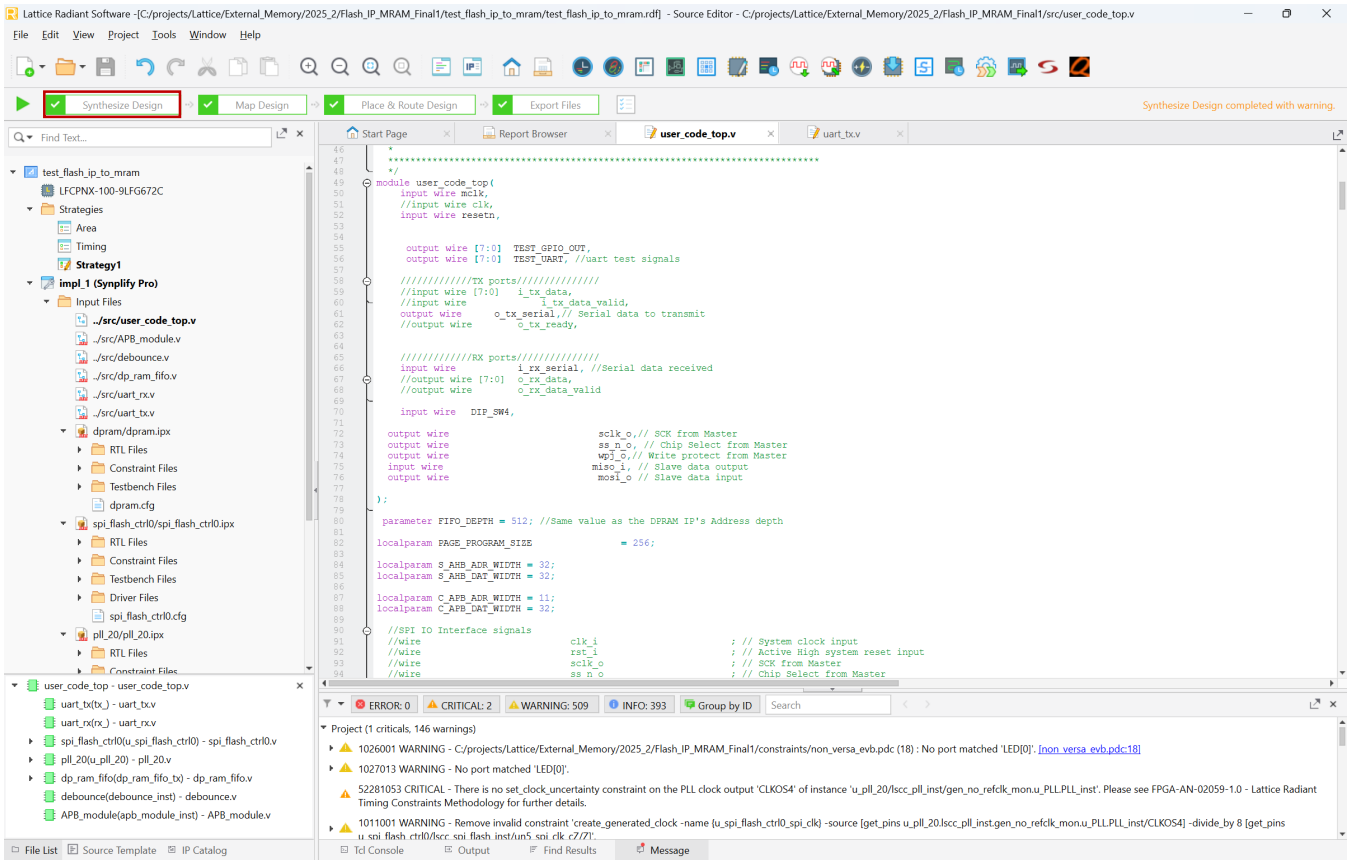


2.3 Compile Design Using Radiant Software

From the Project File List, right click on user_code_top.v and set is as Set Top-Level Unit, see picture below.

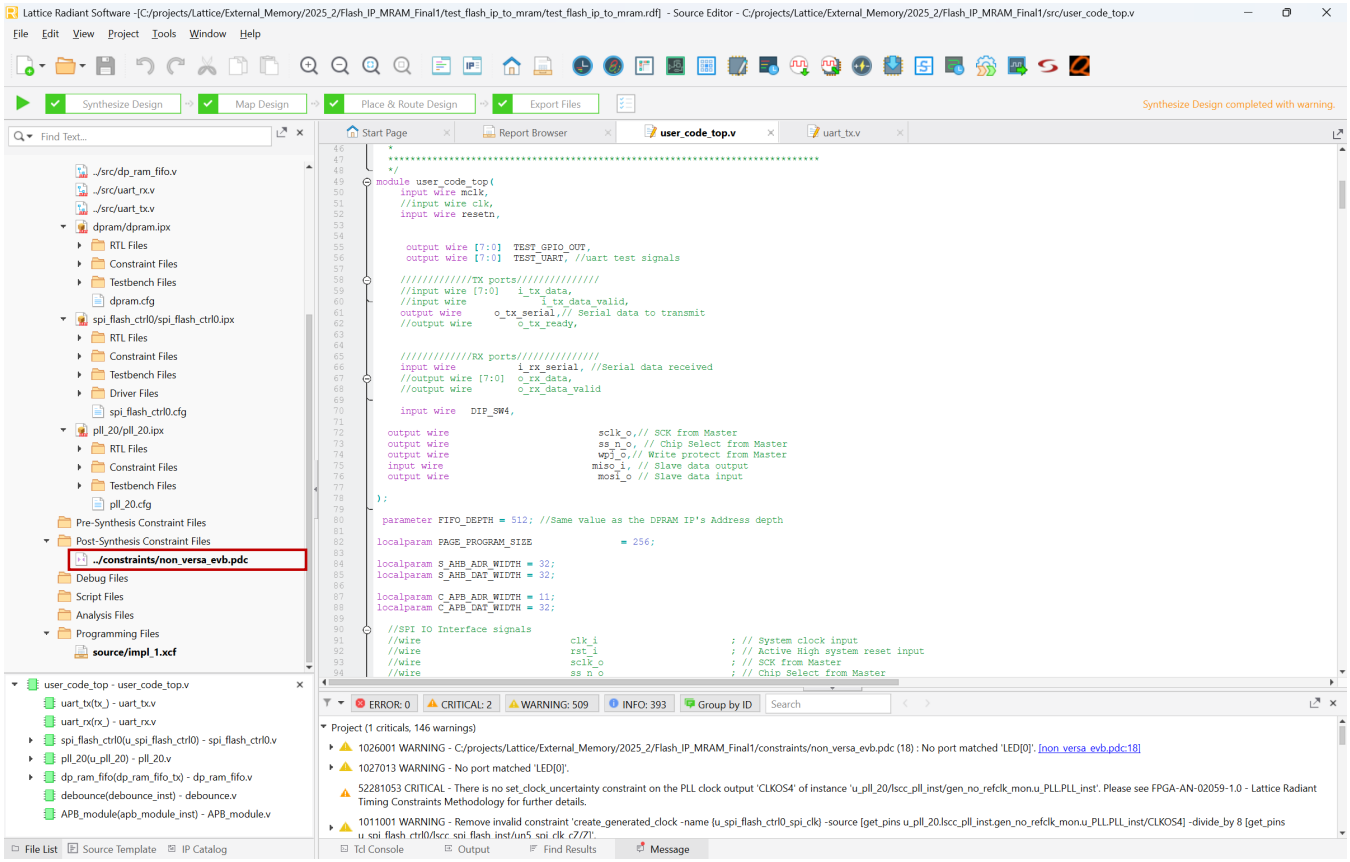


Synthesize the design by clicking on the Synthesize Design arrow, see figure below.

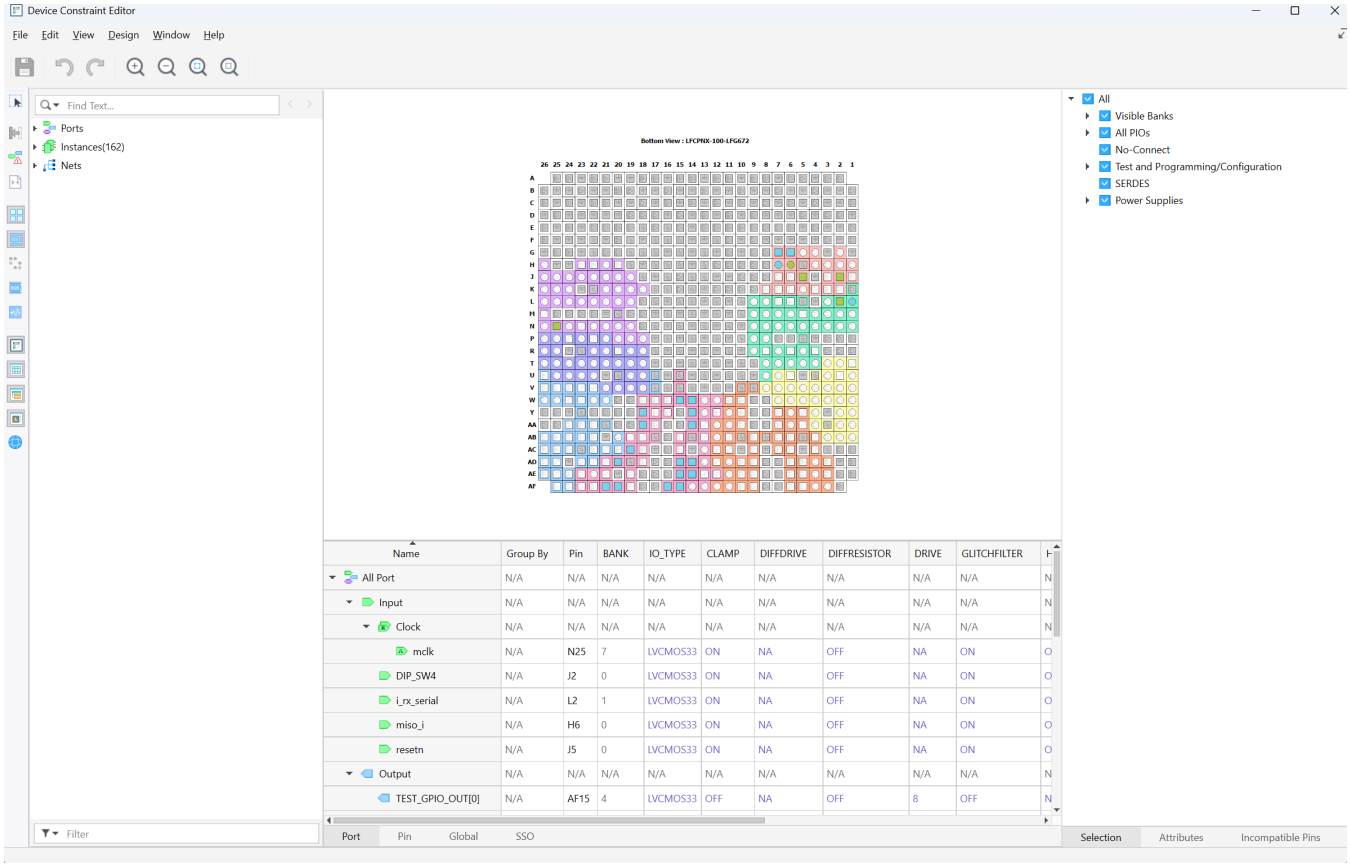


2.4 Pin Assignment (Device Constraint Editor)

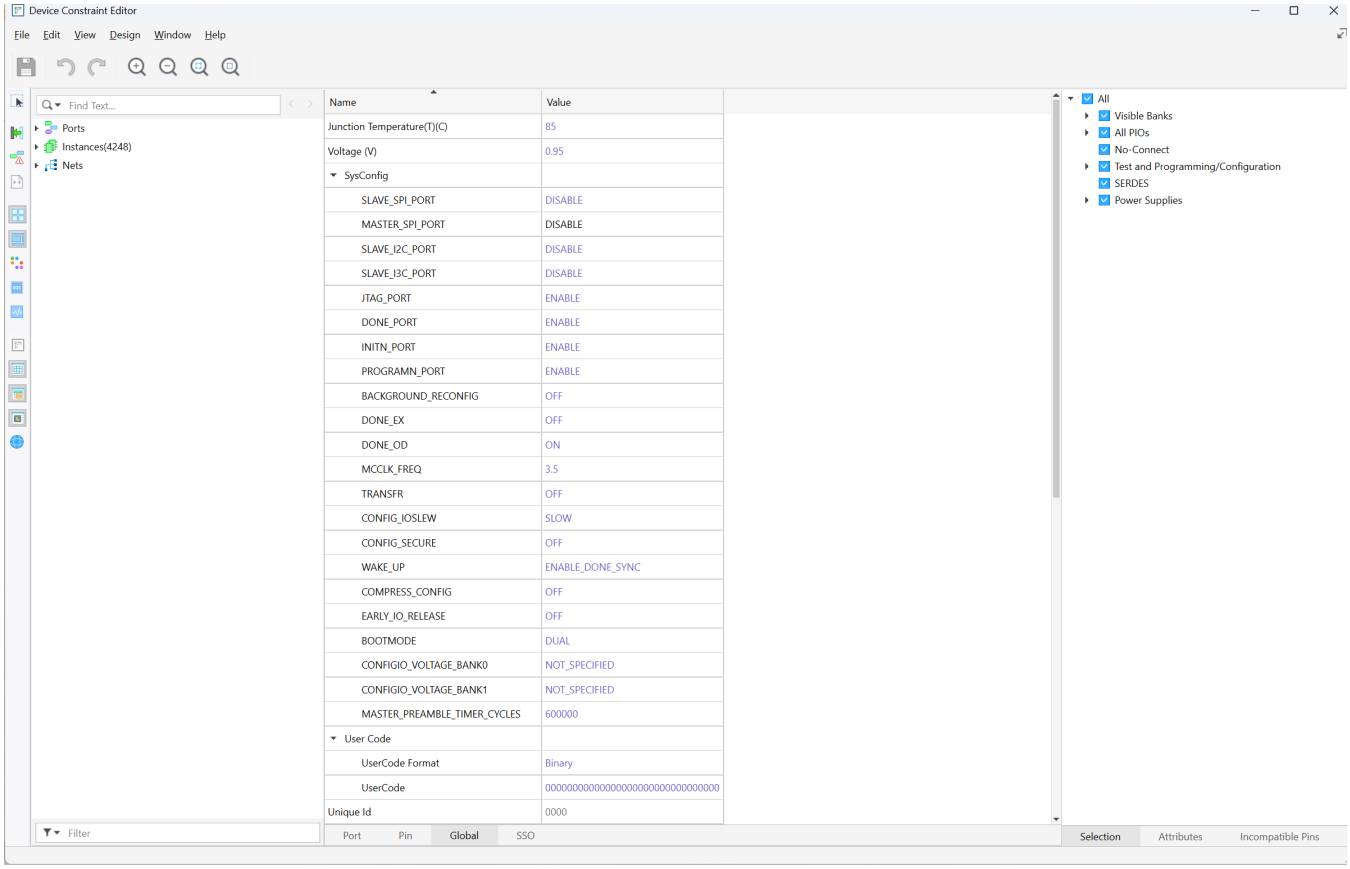
Add the provided non_versa_evb.pdc constraint files to its respective folder, see picture below.



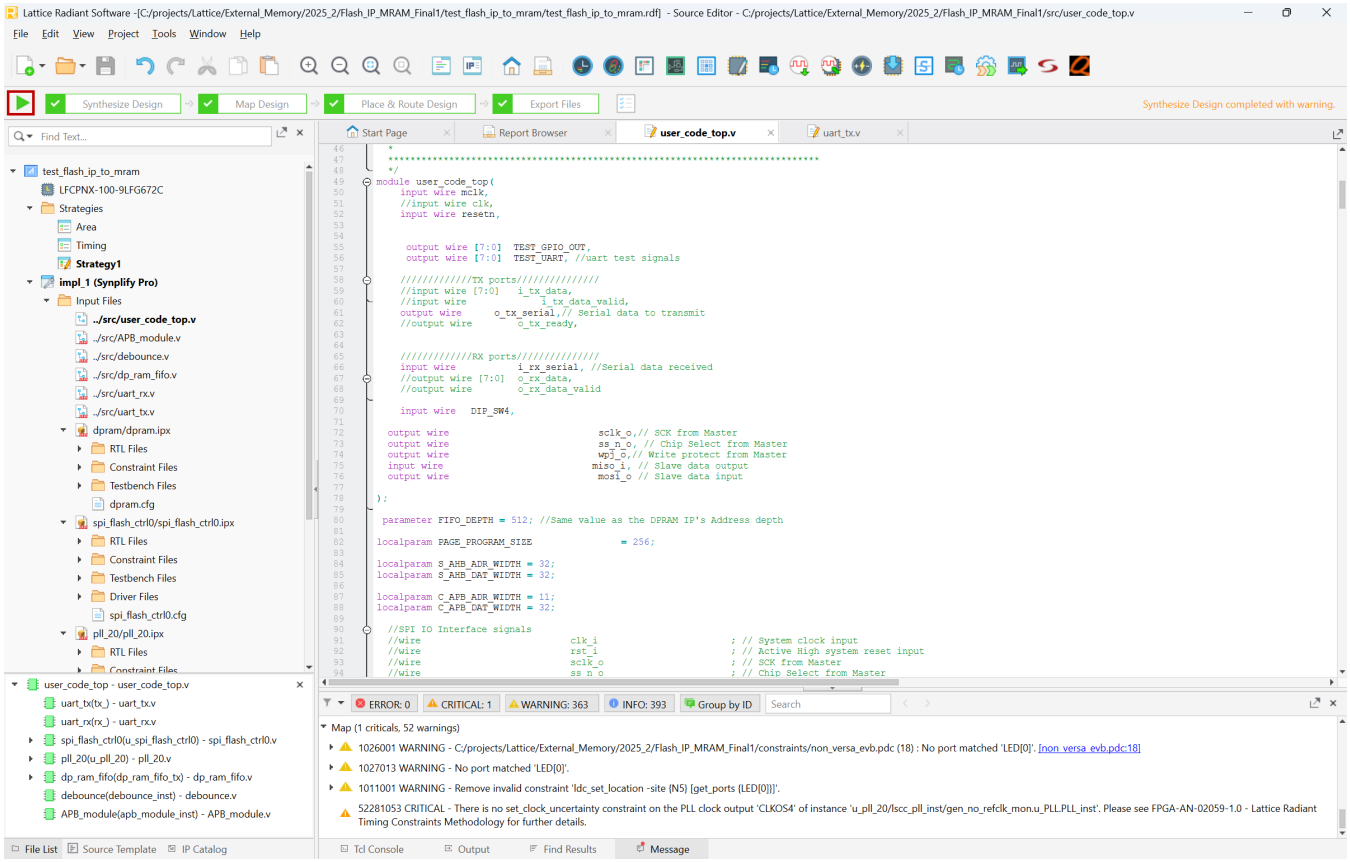
Once the design has been synthesized, use the Device Constraints Editor option of the Tools drop down menu to confirm the pin numbers to their respective signals, see picture below.



After the particular design has been synthesized, select the following **Global** settings, see picture below.

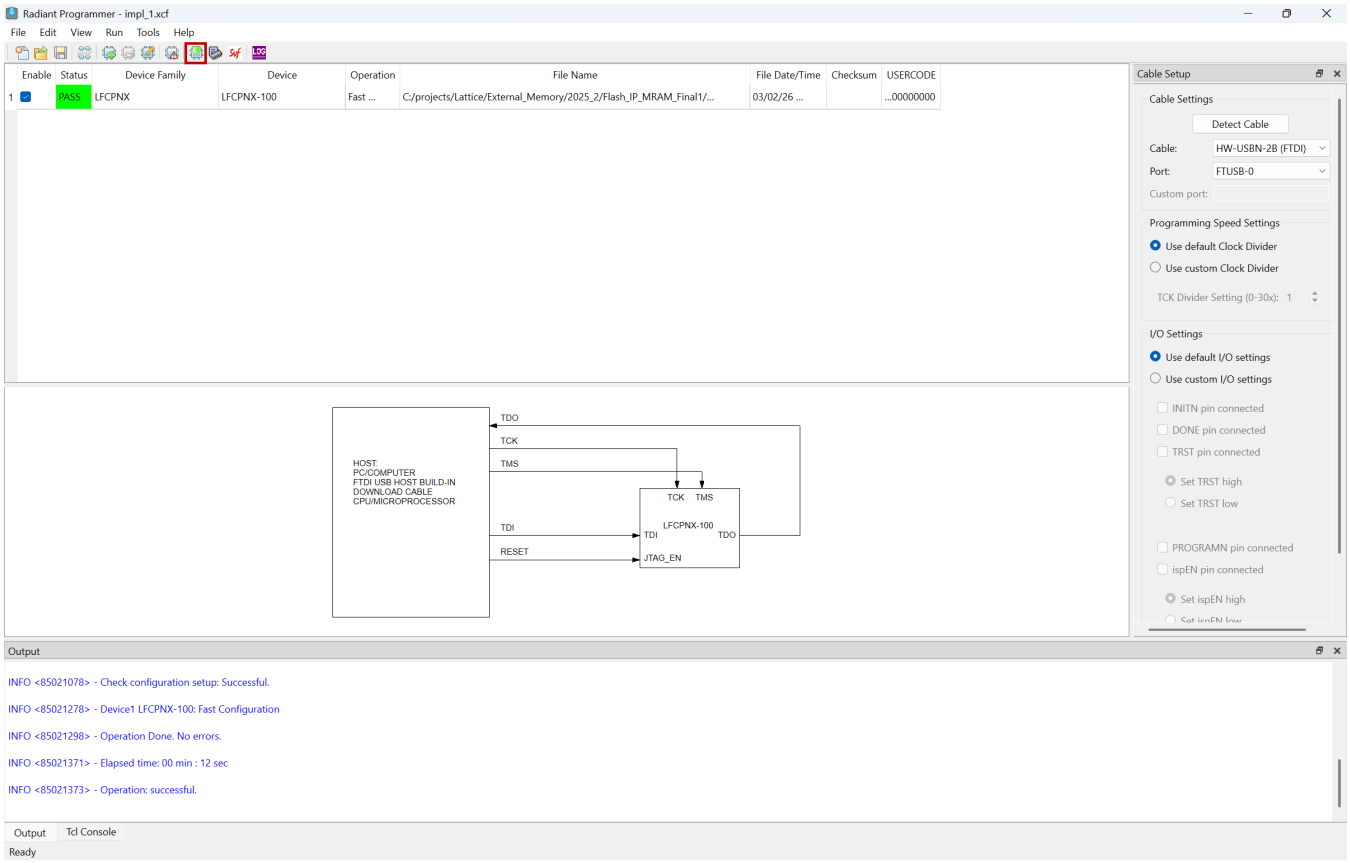


Compile the design by clicking on the Run All arrow, see picture below.



2.5 Program the Device with Radiant Programmer

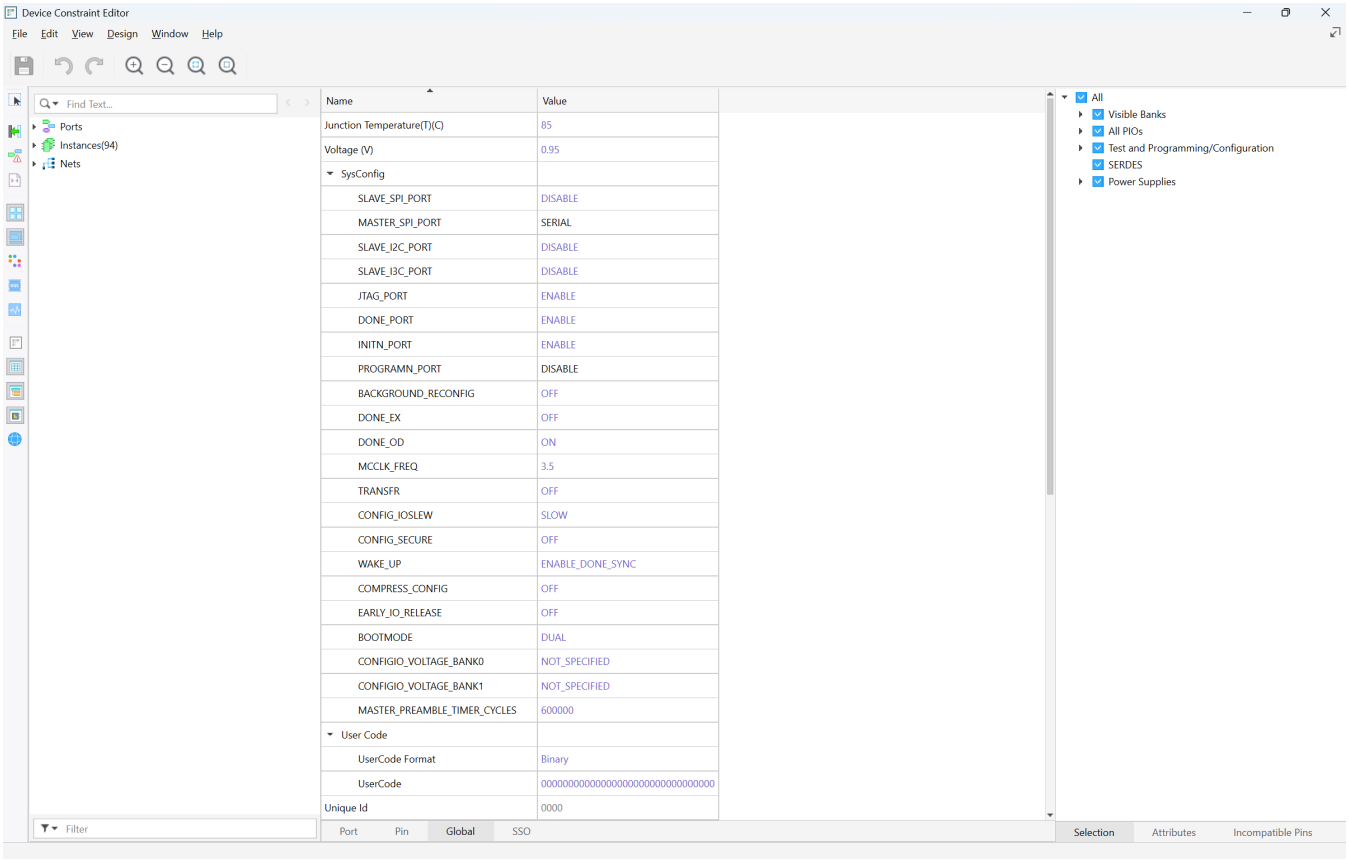
From the **Tools** menu, select **Programmer** and a new window opens, see picture below. Once the cable is detected and settings are set, program the device by clicking the Program Device icon , see picture below.



2.6 Verify the Hardware Interface

The project named **led_brightness** provided with the user code app note is provided for testing the hardware. The design uses switches SW[3:0] for controlling the duty cycle to the pwm output, which controls the brightness to LED[0]; LED[7:1] are used as heartbeat of the system.

Note: make sure the image configuring the FPGA has the following Global settings.



References

For more information, refer to the following documents:

- FPGA-IPUG-02134-2-1-SPI-Flash-Memory-Controller-IP-User Guide
- FPGA-EB-02046-1-1-CertusPro-NX-Evaluation-Board
- FPGA-RD-02270-1-8-Universal-Asynchronous-ReceiverTransmitter-Reference-Design
- FPGA-TN-02099-3-3-sysCONFIG-User-Guide-for-Nexus-Platform
- CertusPro-NX 100k Pinout File (FPGA-SC-02022)
- fpga-tn-02145-2-1-multiboot-user-guide-for-nexus-platform
- Avalanche_Serial-Dual-QSPI-P-SRAM-for-Xilinx-Versal-ACAP-VCK-190-Evaluation-Kit-User-Guide-v1-1
- datasheet_-_ut8mrq2g
- FPGA-IPUG-02191-1-1-Distributed-Memory-Modules

Revision History

Date	Revision #	Author	Change Description	Page #
4/7/2026	1.0.0	JA	Initial release.	N/A

The following United States (U.S.) Department of Commerce statement shall be applicable if these commodities, technology, or software are exported from the U.S.: These commodities, technology, or software were exported from the United States in accordance with the Export Administration Regulations. Diversion contrary to U.S. law is prohibited.