Nios® II and the Interval Timers' Alarm and Timestamp Functionality on the Intel® MAX® 10-10M08 Evaluation Kit

By Sean D. Liming and John R. Malin
Annabooks, LLC. – www.annabooks.com

January 2023

Timing is everything. The Interval Time IP that comes with Quartus not only provides the date and time for a Nios II processor but also supports alarms and timestamp functionality. The paper walks through a couple of applications that test both.

Please see the article *Intel® Quartus® Prime Lite and Nios® II SBT for Eclipse Installation Instructions* on Annabooks.com to install the software needed for this hands-on exercise.

The Project Requirements:

- Intel Quartus Prime Lite Edition V21.0 and Nios® II SBT for Eclipse are already installed.
- Intel® MAX® 10 - 10M08 Evaluation Kit and the schematic for the evaluation board are required. The schematic PDF file can be downloaded from the Intel FPGA website.
- Intel FPGA Programming cable – USB Blaster II or EthernetBlaster II. The Intel® MAX® 10 - 10M08 Evaluation Kit doesn't have a built-in USB Blaster II onboard.
- *Intel® Quartus® Prime Lite and NIOS® II SBT for Eclipse Installation Instructions* on Annabooks.com

Note: There are equivalent MAX 10 development and evaluation boards available. These boards can also be used as the target, but you will have to adjust to the available features on the board. Please make sure that you have the board's schematic files as these will be needed to identify pins.

1.1 Nios II Timer Project

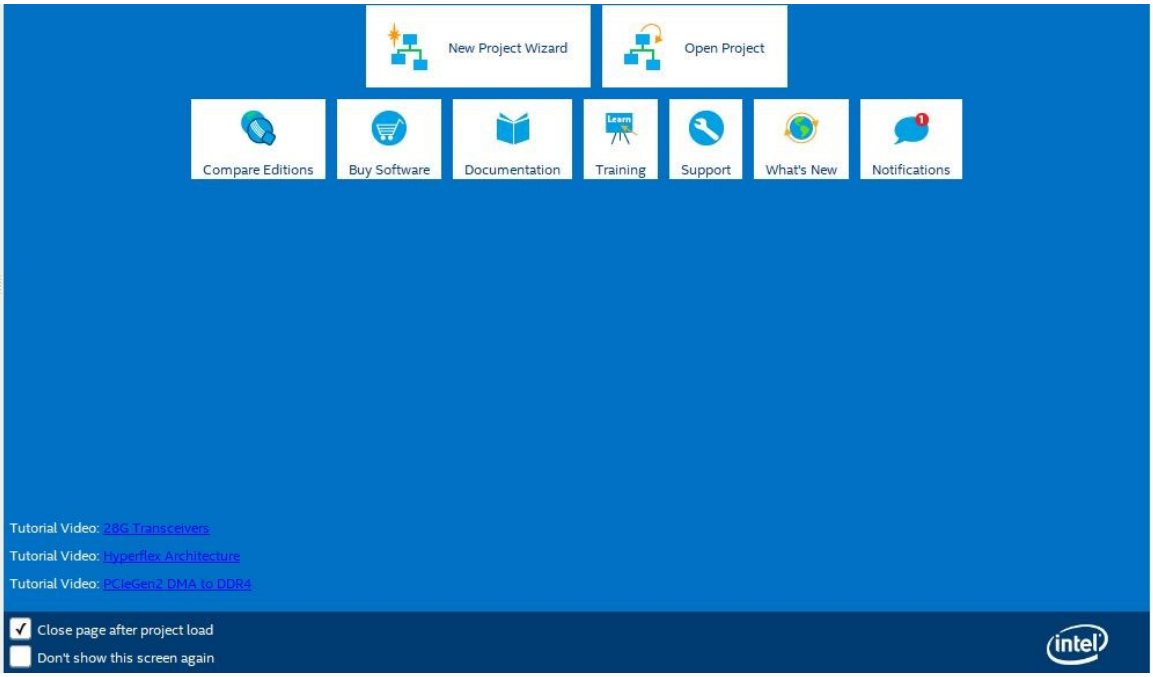
The custom MCU will comprise the following IP blocks:

- Nios II processor
- Onchip RAM
- Interval Timer
- Parallel IO for LEDs
- Sys ID
- JTAG UART

1.1.1 Create the Project

The first step is to create a design project.

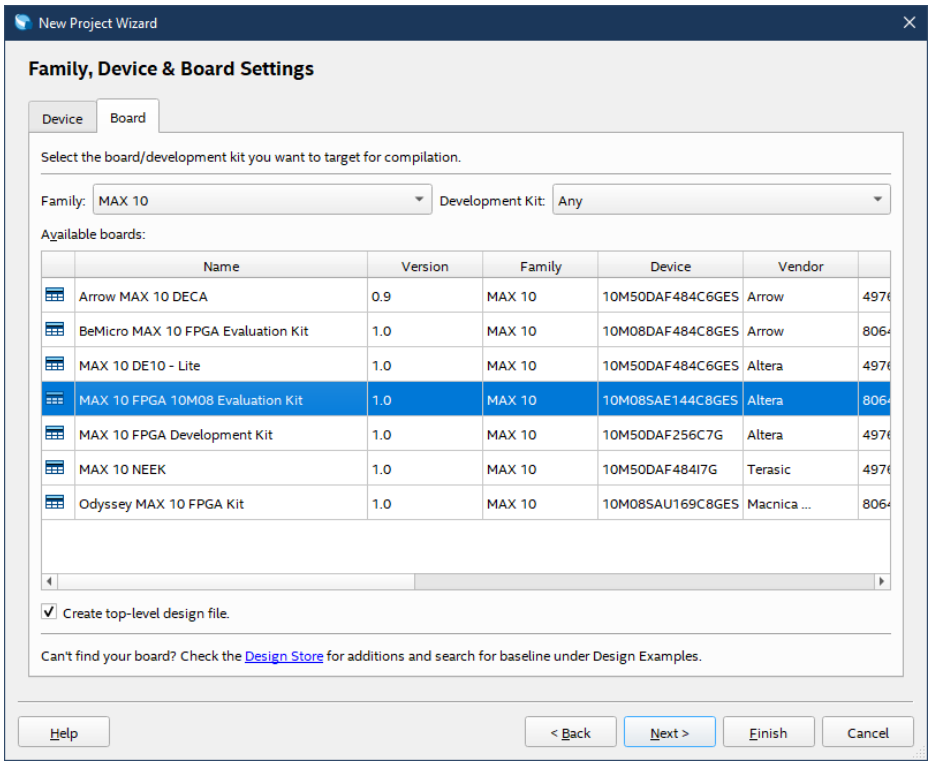
1. Open Quartus
2. Click on the New Project Wizard



3. Click Next to the Introduction dialog
4. Select or create a project directory \NIOS2_Timer (Do not use the Quartus installation directory) and name of the project: “NIOS2timer”. Click Next.

Note: By default, the root directory is the Quartus installation directory. Make sure the root project directory is a separate path from the Quartus installation files. Also, there can be no spaces in the name of the folders or projects.

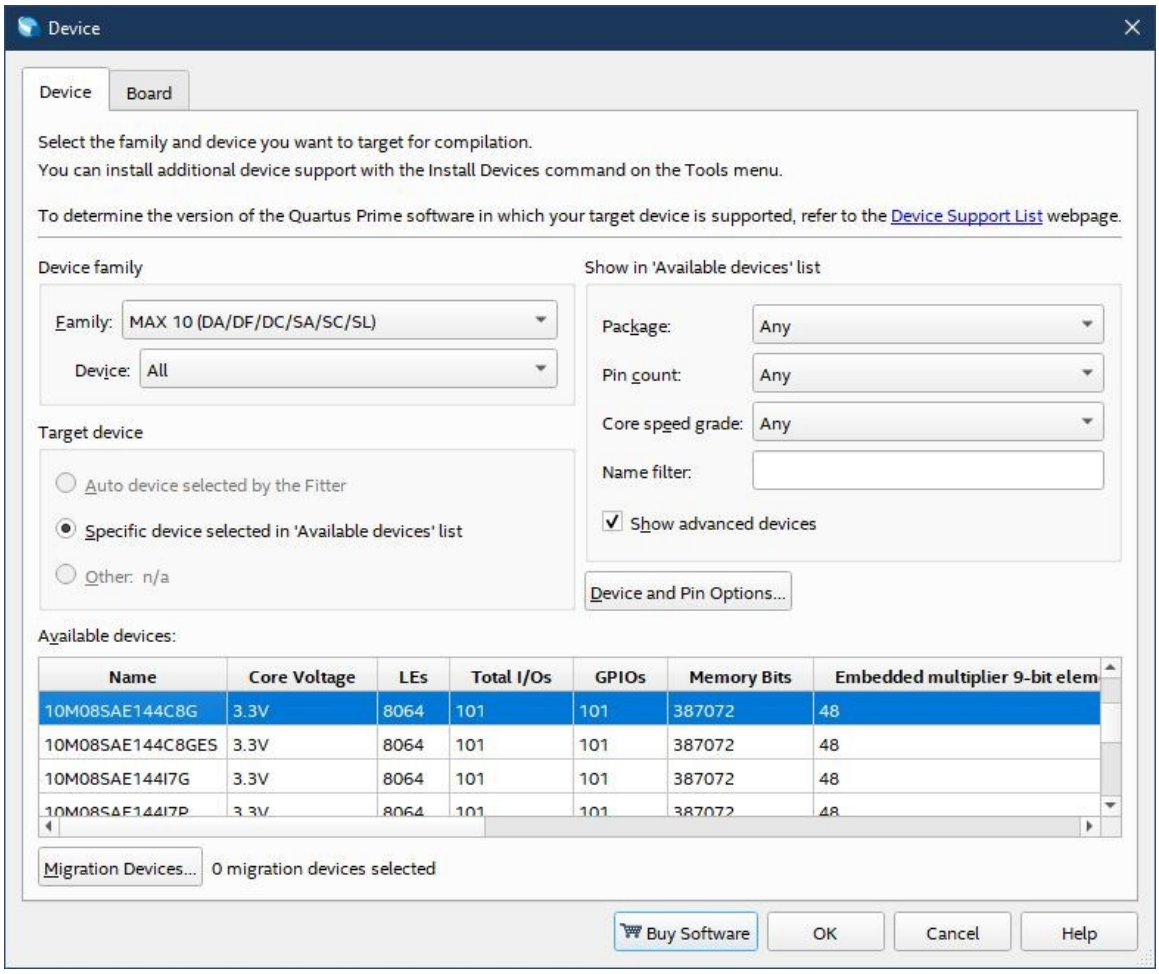
5. Project Type: Empty project, click Next
6. Add File no files to add, click Next.
7. Family, Device & Board Settings, click the Board tab and select: MAX 10 FPGA 10M08 Evaluation Kit, and click Next.



8. EDA Tools: click Next.
9. Summary: click Finish

Note: The actual MAX 10 on our board is the 10M08SAE144C8G, thus it is not an Engineering Sample (ES). The next two steps change the device to the production device. Your experience might be different. These next two optional steps change the device.

10. In the project navigation pane on the left, right-click on 10: 10M08SAE144C8GE, and select Device from the context menu.
11. In the Available devices, scroll down and select the 10M08SAE144C8G. Click OK.



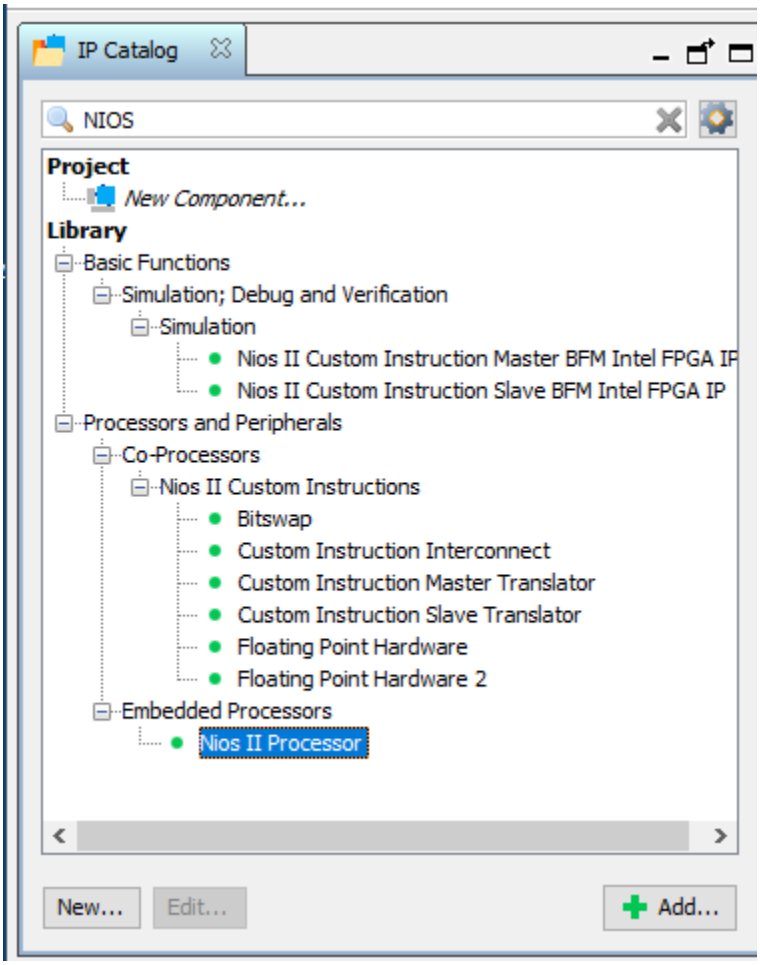
1.1.2 Create the Design in Platform Designer

Quartus supports many design types to create an FPGA design. The Platform Designer tool will be used for this hands-on exercise. Platform Designer makes it easy to add already-built IP blocks and interconnect them.

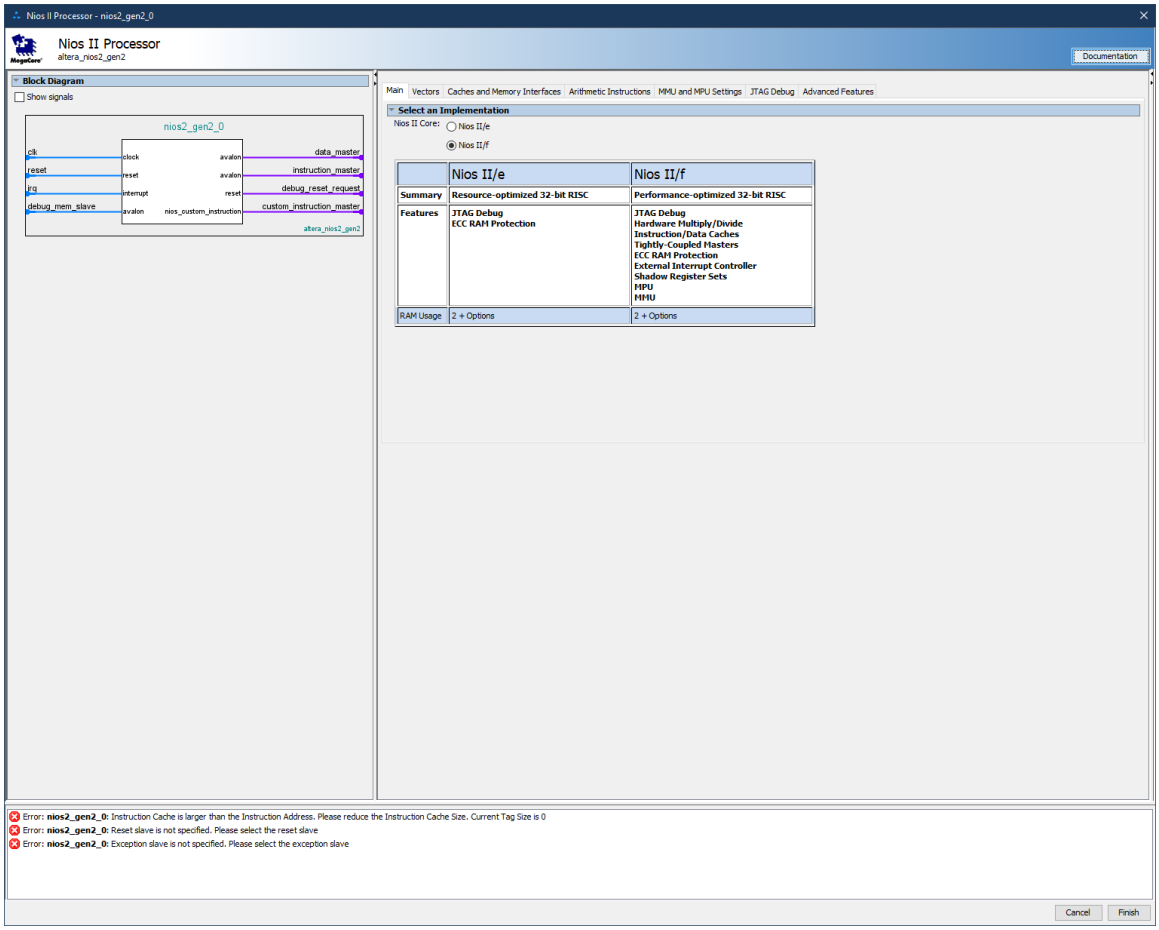
1. From the menu, select Tools->Platform Designer, or the Platform Designer icon  from the toolbar.

The Platform Designer tool is launched. By default, a clock (clk_0) is added to the design. Platform Designer makes it easy to add IP blocks and make interconnections between the blocks.

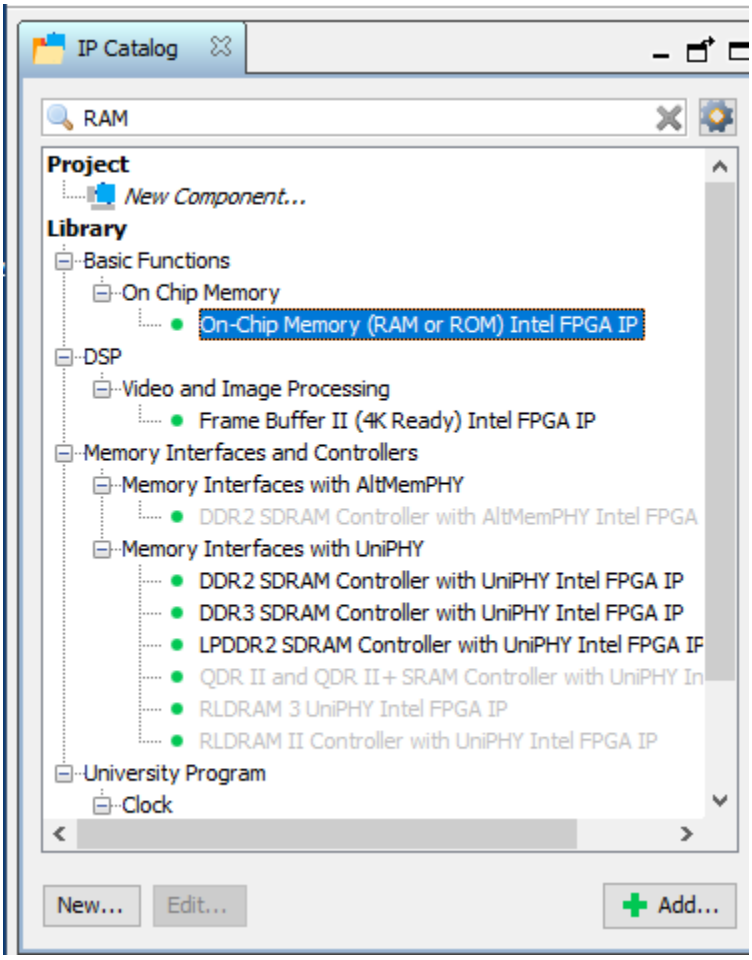
2. The top left pane contains the IP Catalog with all the available IP blocks that come with Quartus Prime. In the search box, type NIOS.



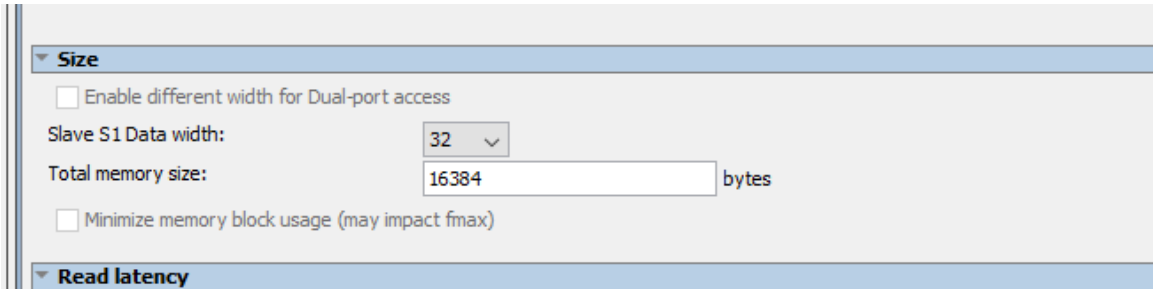
3. Expand the Processors and Peripherals and Embedded Processors branches and double-click on the Nios II Processor.
4. This will open the Nios II Configuration page. The first tab is to select the type of core Nios II/e or Nios II/f. We will keep the defaults for now. Click Finish.



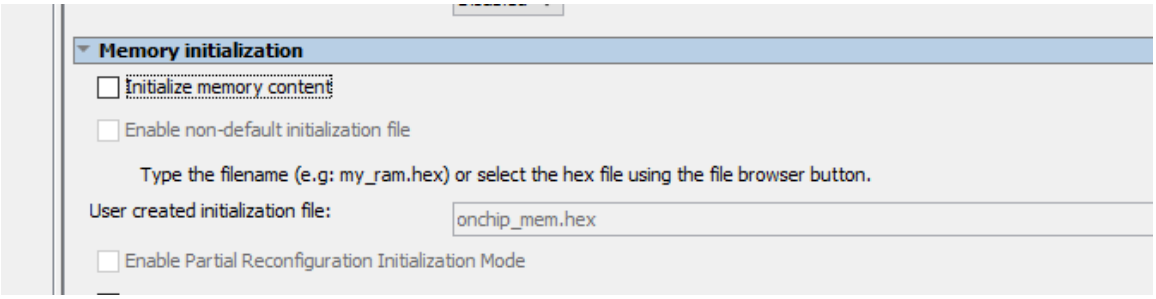
1. The processor will be added to the design. Right-click on the name nios2_gen2_cpu, and rename it to nios2.
2. Now let's add the RAM IP block. In the IP Catalog enter RAM in the search box.
3. Double-click on On-chip Memory (RAM or ROM) in the Intel FPGA IP.



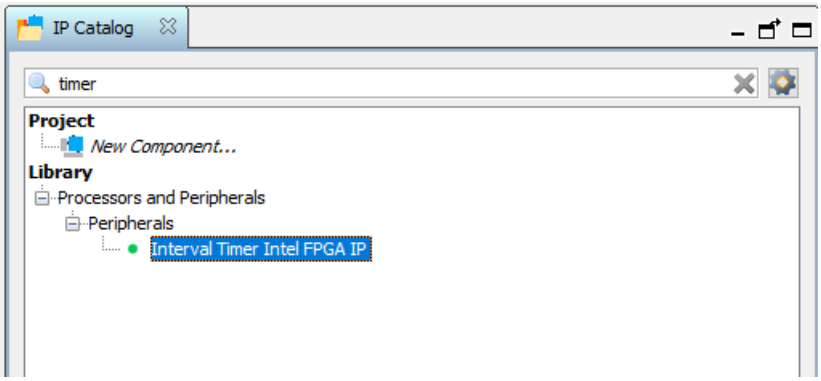
4. The configuration page will appear. Change the Total memory size to 16384. We need more memory to run the application.



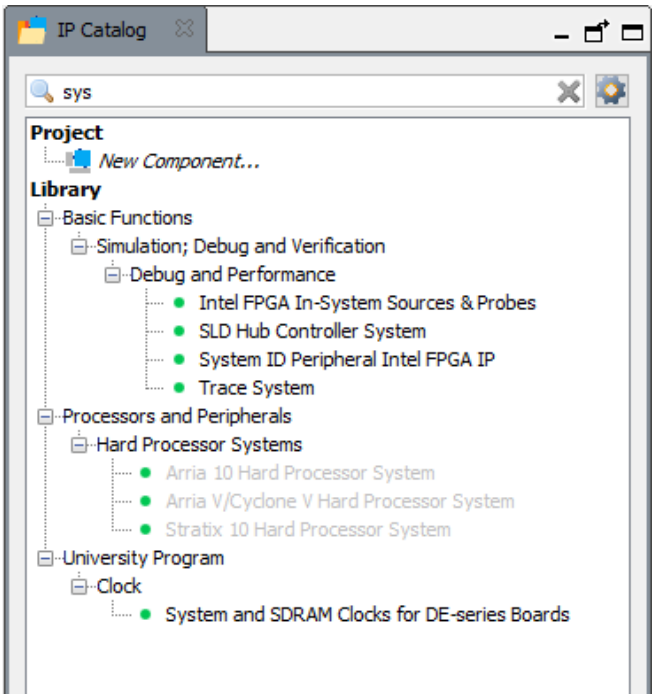
5. Uncheck the box for “Initialize memory content”, and click Finish.



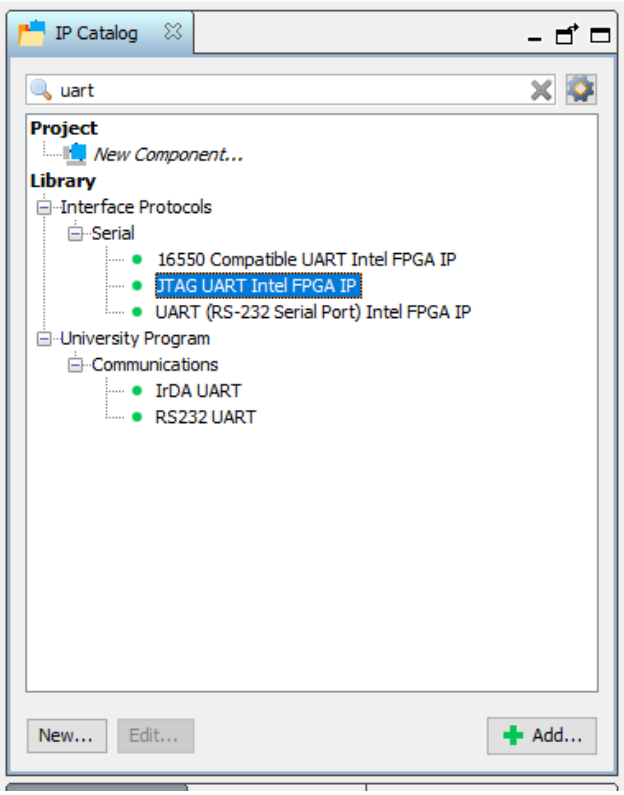
6. The On-chip Memory (RAM or ROM) in the Intel FPGA IP will be added to the design, right-click on the name, and rename it to onchip_RAM.
7. In the IP Catalog search, enter timer.
8. Double-click on the Interval Timer Intel FPGA IP.



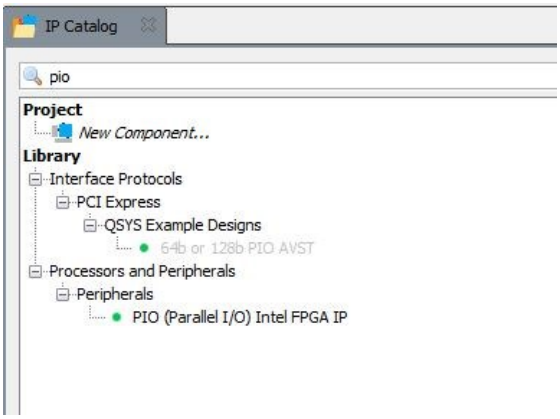
9. Keep the settings as they are and click Finish.
10. In the IP Catalog search, enter system ID.
11. Double-click on the System ID Peripheral Intel FPGA IP.



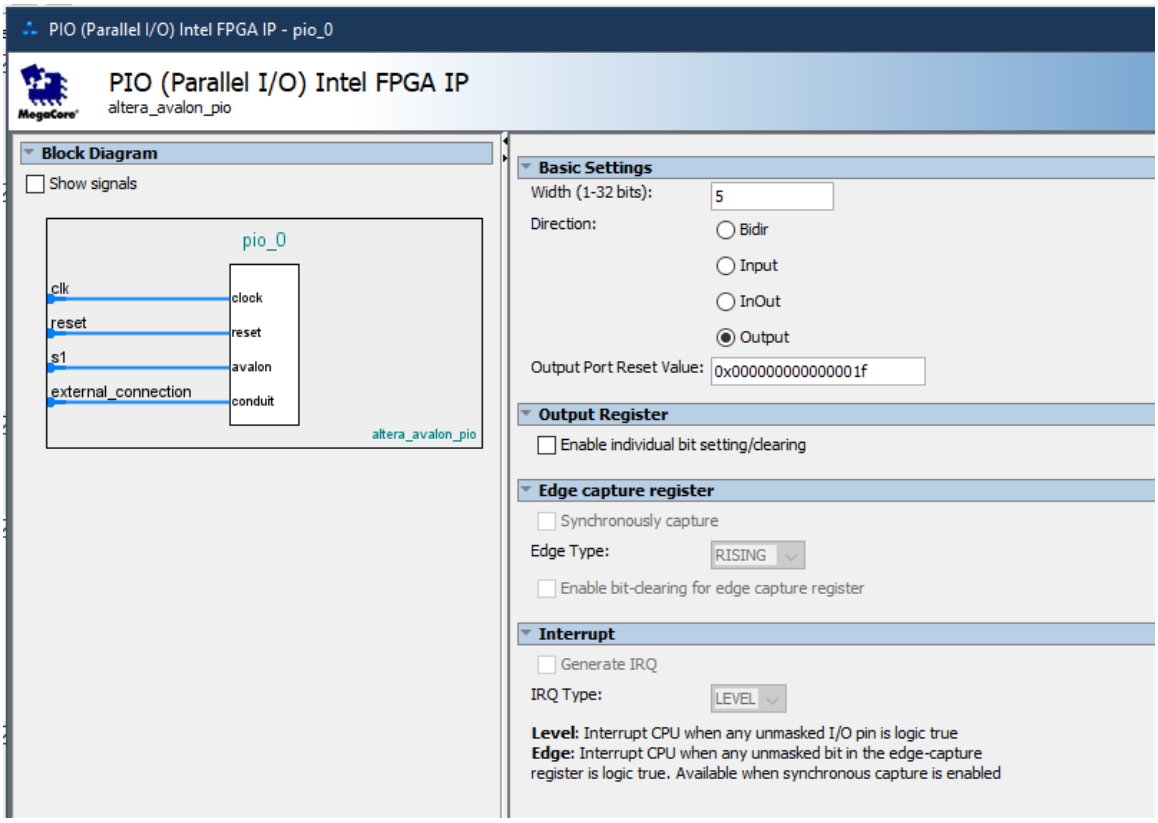
12. A configuration page will appear. There are no changes to be made. Click Finish.
13. In the IP Catalog search, enter uart.
14. Double-click on the JTAG UART Intel FPGA IP.



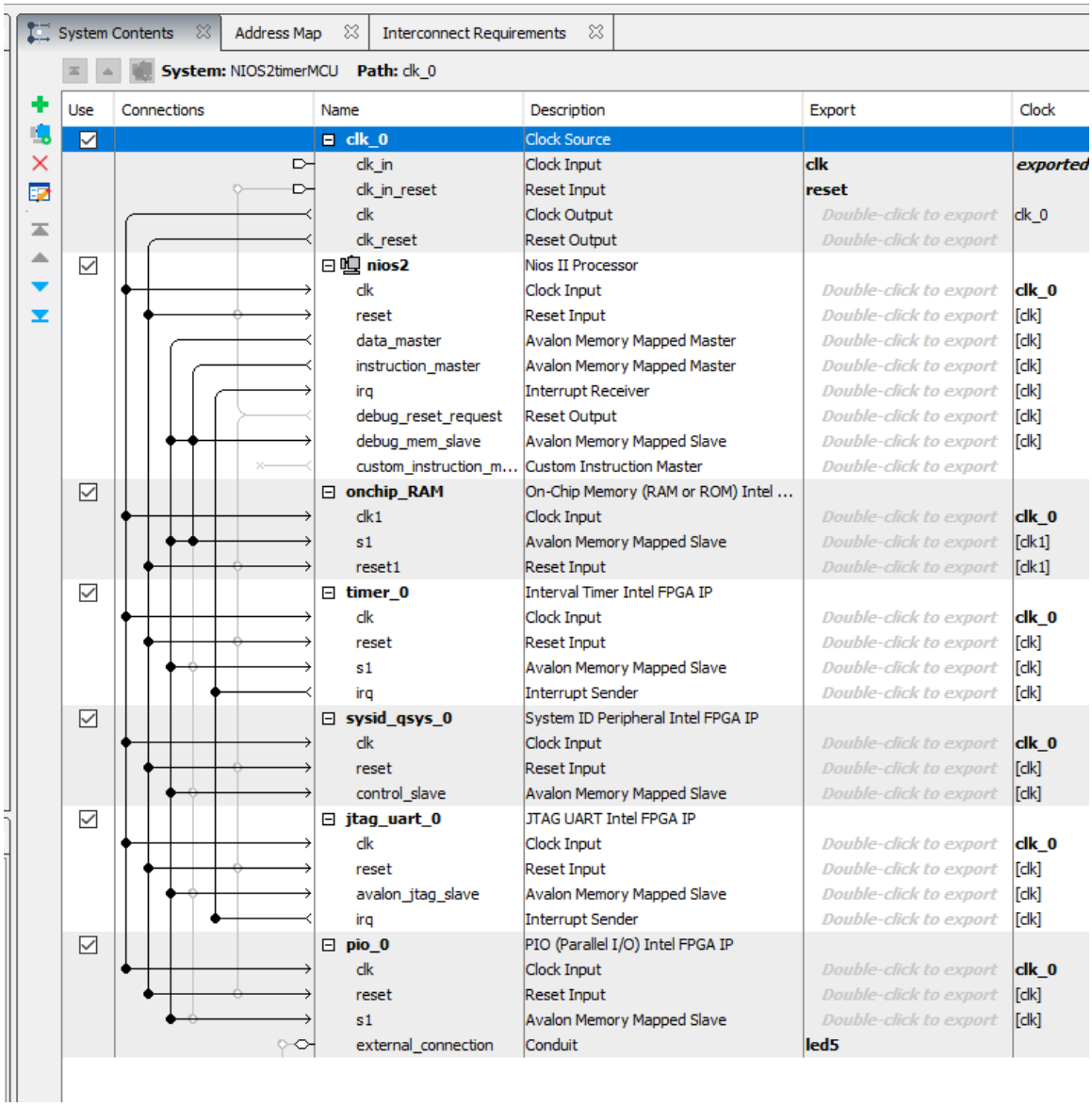
15. A configuration page will appear. There are no changes to be made. Click Finish.
16. In the IP Catalog enter pio in the search box.
17. Add the PIO (Parallel I/O) Intel FPGA IP to the design.



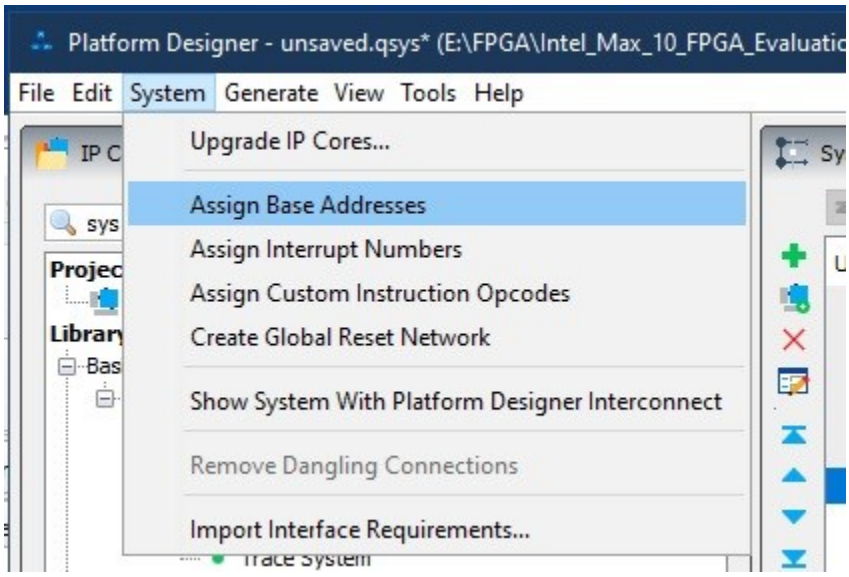
18. In the configuration page, set the Width to 5, leave the Direction as Output, and set the Output Port Reset Value to 0x1f. Since the LEDs are active low, the value turns 5 LEDs off on startup.
19. Click Finish.



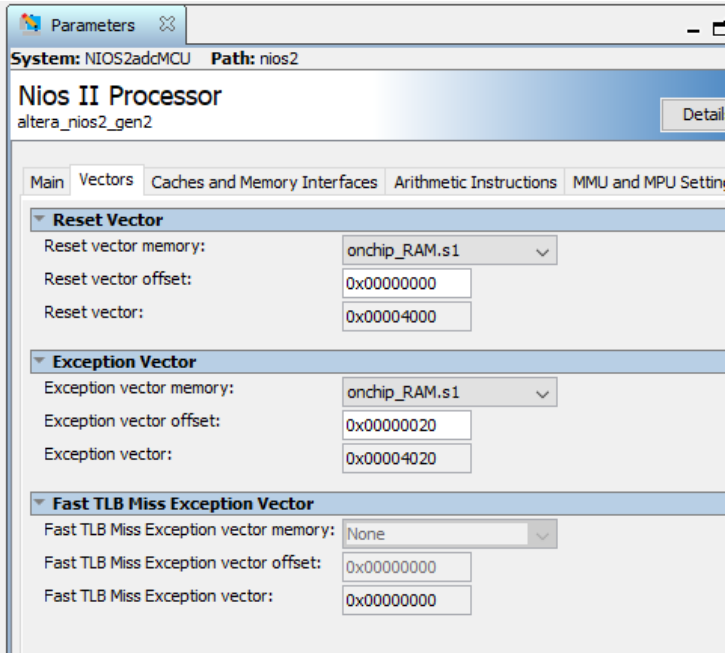
20. The PIO will be added to the design. Rename the PIO to pio_0.
21. For the PIO exernal_connection, double-click on the Export column and set the value to led5. This will provide a base name for connecting the signals to the PINs on the chip. The connection will be made in PIN Planner.
22. Now we need to wire the IP blocks together. The picture below shows all the writing connections for the design.



23. Let's assign a base address. From the menu, select System->Assign Base Address. This will remove a number of errors from the message box. You will see the base address values for each IP change in the System Contents tab.

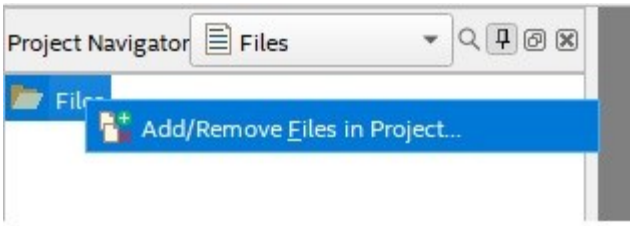


24. Finally, let's set the reset and exception vector addresses. Double-click on the nios2 to open the configuration page.
25. Click on the Vectors tab.
26. Change the Reset vector memory drop-down to onchip_RAM.s1.
27. Change the Exception vector memory drop-down to onchip_RAM.s1.

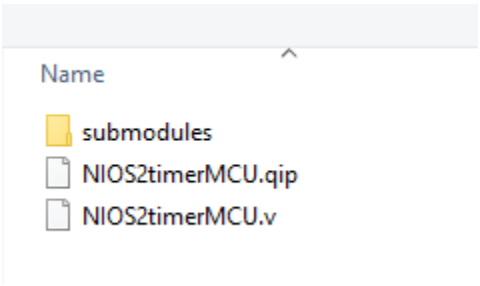


28. Click on Generate HDL…
29. Keep the defaults and click the Generate button.
30. A dialog will appear asking you to save the design, click Save.
31. Give the name as NIOS2timerMCU.qsys and click Save.
32. Once the save has been completed, click Close.
33. The generate process kicks off. The processes should succeed, click Close.

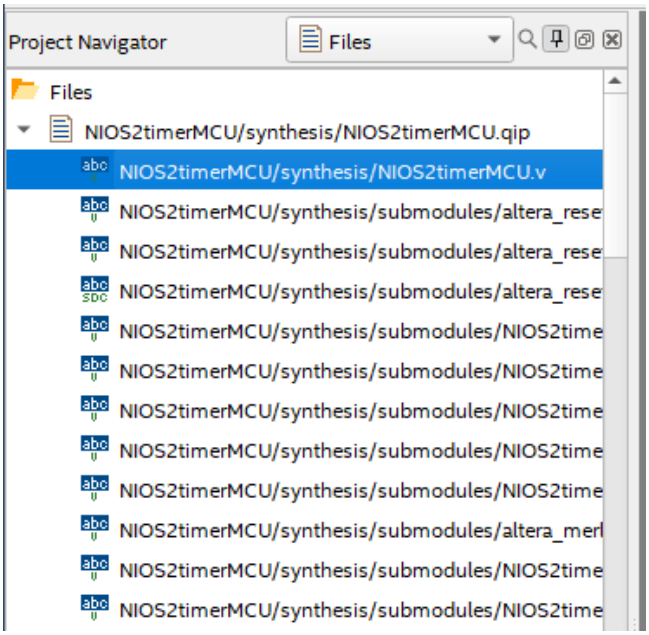
34. Click Finish to close the design.
35. Quartus then reminds you to add the new design to the project. Click Ok.
36. In the Project Navigator, click on the drop-down and select Files.
37. Right-click on Files and select Add/Remote Files in Project.



38. A Settings – NIOS2timer page appears with Files on the left highlighted. Click the three dots browse button for File name, and navigate to \NIOS2_Timer\NIOS2timerMCU\synthesis folder.
39. Click on NIOS2timerMCU.qip file and click open



40. Click OK to close the Settings- NIOS2timer page. The qip file is added to the Project navigator list. Underneath are all the Verilog files that were generated by Platform Designer.

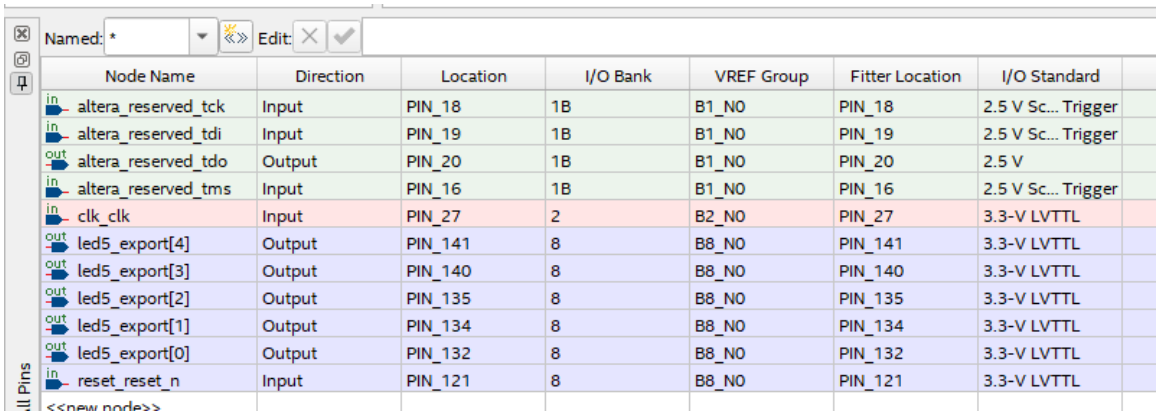


41. In the Project Navigator Right-click on the NIOS2timerMCU/synthesis/NIOS2timerMCU.v file, and select Set as Top-Level Entity from the context menu.
42. Save the project.
43. In the Task pane on the left, double-click on Fitter (Place & Route) to start the task. The analysis will take some time, and it should succeed in the end. This step helps to diagnose any errors and finds the Node Names for the pin assignments in the next step.
44. Once the process completes, the pin assignments need to be set, from the menu select Assignments->Pin Planner or click on the  icon from the toolbar. The analysis just run populated the Node Name list at the bottom of the Pin Planner dialog.

45. Using the board schematic, locate the pins for the SW1 and the 50MHz clock. Set the Location values for both node names. For the MAX 10 – 10M08 Evaluation Board, these values are as follows:

Node Name	Location
SW1	PIN_121
Clk_50MHz:	PIN_27
altera_reserved_tck	PIN_18
altera_reserved_tdi	PIN_19
altera_reserved_tdo	PIN_20
altera_reserved_tms	PIN_16
Led5_export[4]	PIN_141
Led5_export[3]	PIN_140
Led5_export[2]	PIN_135
Led5_export[1]	PIN_134
Led5_export[0]	PIN_132

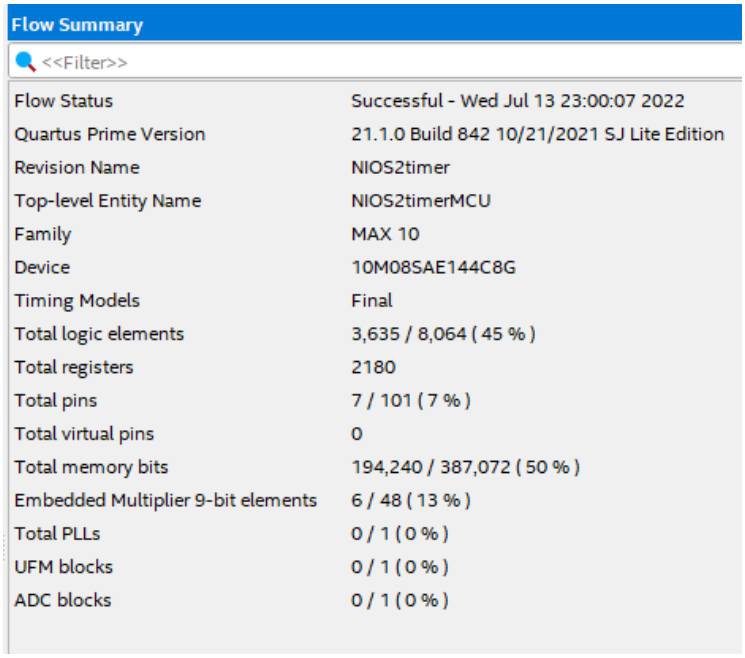
46. Set the I/O Standard to 3.3V-LVTTL for all pins except JTAG. You can see from the schematic that the I/O are all tied to 3.3V.



47. Close the Pin Planner when finished. The diagram gets updated with the pin numbers.
48. Save the project.

Note: Quartus can crash unexpectedly, which may be due to the fact that it was written in Java and is not a native Windows application based on .NET. Therefore, a best practice at this point is to make a backup of the project folder. Archiving is simple. From the menu, Project->Archive Project.

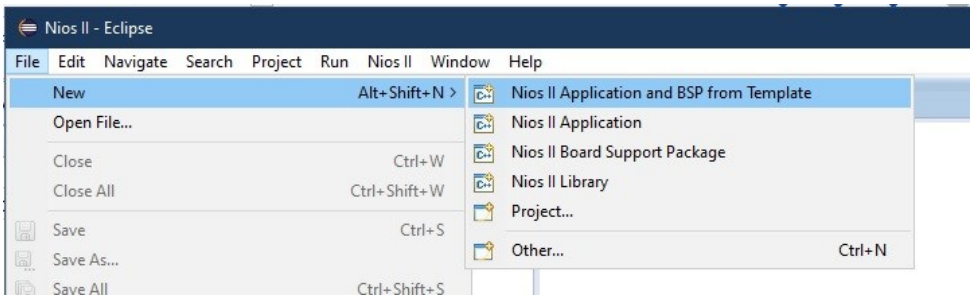
49. Finally, compile the design. In the Task pane, right-click on Compile and Design and select Start from the context menu, or you can click on the  symbol in the toolbar. The design should compile successfully.



1.1.3 Eclipse Application: Alarm

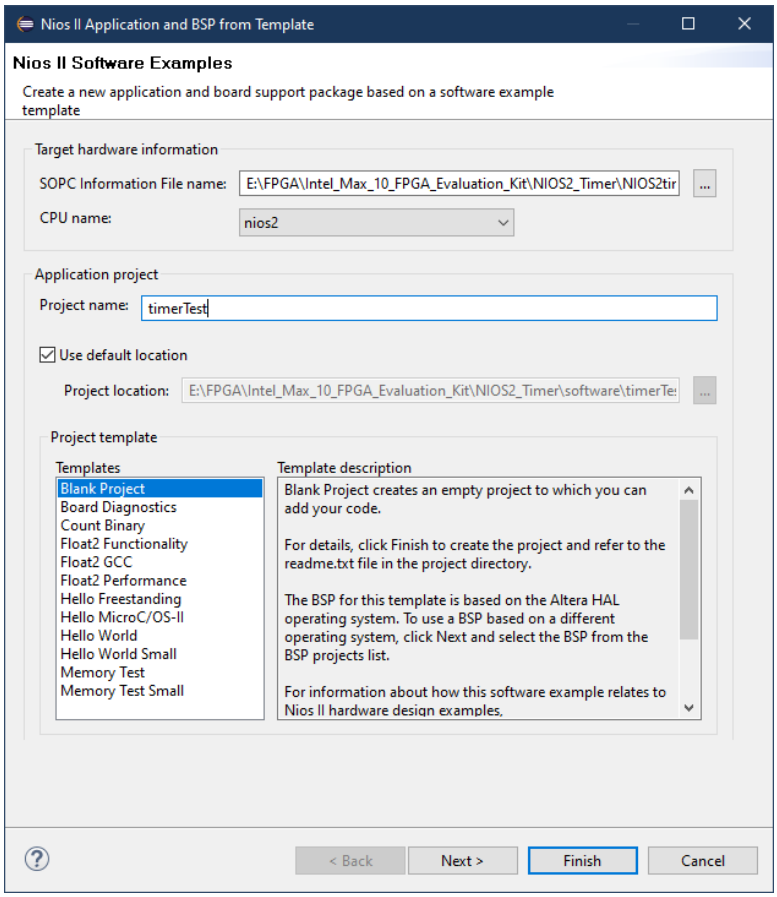
The first application will test the Alarm functionality.

1. In Quartus Prime, from the menu, select Tools->Nios II Software Build Tools for Eclipse.
2. Eclipse will open and ask for the root workspace directory. Set the workspace folder to something like \Documents\FPGA\Apps, and hit ok. It doesn't matter the location of the workspace, since the actual applications for the project will exist within the \NIOS2_Timer\software folder.
3. In Eclipse, from the menu, select File->New-> Nios II Application and BSP from Template.



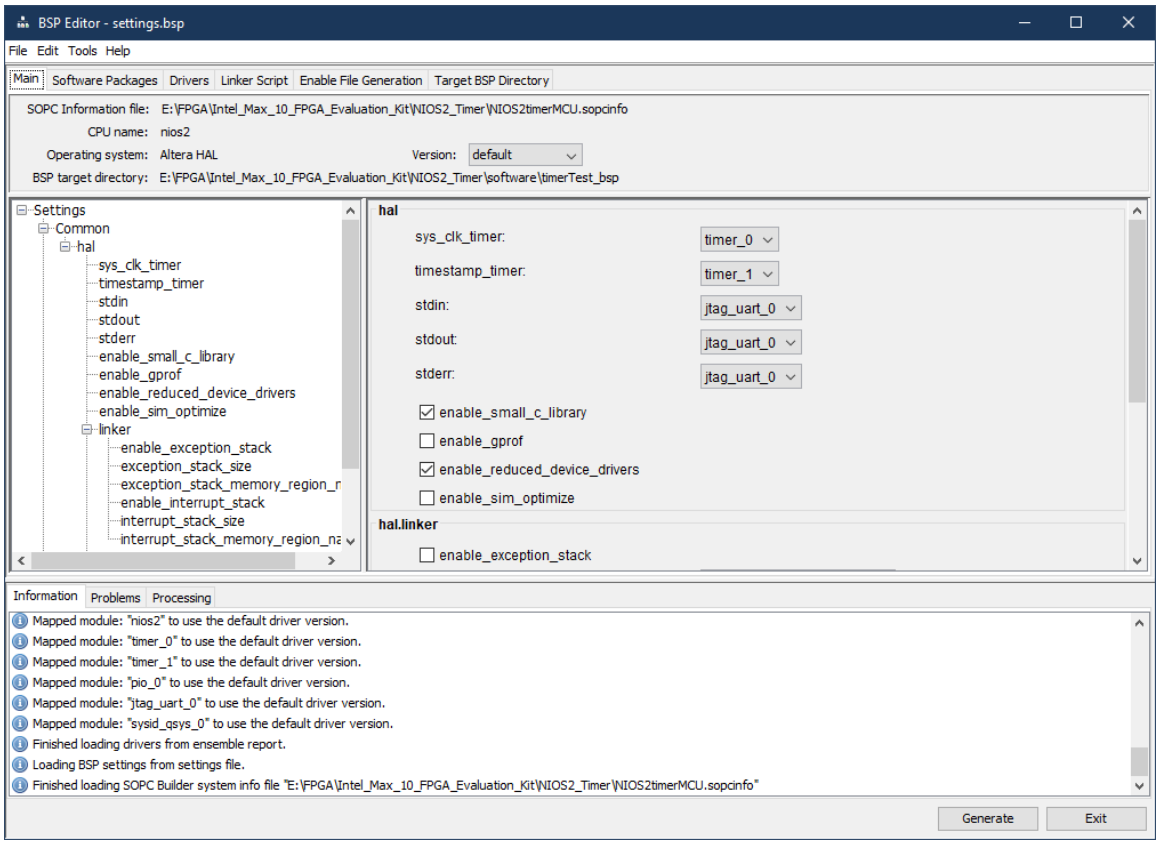
4. The first step is to open the SOPC file that was generated for the hardware design. Click on the three dots button.
5. Navigate to the \NIOS2_Timer folder and open the NIOS2timerMCU.sopcinfo file. The CPU name will reflect the name we gave the CPU in Platform Designer.
6. Enter the project name: timerTest.
7. In the Project Template, select Blank Project

8. Click Finish.



Two projects will be generated. The timerTest _bsp is generated to give you the HAL drivers and API based on the hardware design. The timerTest is the application that will run on the hardware.

9. We need to edit the BSP to use the small C library and drivers. The BSP Editor tool allows you to edit the settings.bsp file to make specific changes for the target. Right-click on timerTest_bsp and select Nios II->BSP Editor from the context menu.
10. The BSP Editor opens and opens the settings.bsp file automatically. If you started the BSP Editor from the main menu, you would have to manually navigate to open the file. In the BSP Editor, tick the box for enable_small_c_library and enable_reduced_device_drivers.



The interval timer can either be a sys_clk_timer or a timestamp_timer, but a single interval timer cannot be both. Two intervale timers could have been added to the design. One timer (timer_0) for the system clock timer (alarms / count down) and another timer (timer_1) for the timestamp timer. For this project, we will flip timer_0 between the two application projects. The first project will be the alarm so leave sys_clk_timer as is.

11. Click Generate to generate the changes.
12. Click Exit when finished.

The timerTest_bsp contains the key files that will help with filling in the code to access the timers and pio port. System.h contains the definitions that can be used for how the timer and PIO were set up in Platform Designer. Since we are using the small_C_library for space contain reasons, the standard C io calls cannot be used. Instead, we will be using the Nios II HAL API to access the timer, pio, and the standard output via Jtag uart. The header files for the timestamp.h and alarm.h contain the APIs needed for the application.

13. We need to add a main.c file to the project. Right-click on the timerTest project, and select New->File from the context menu.
14. Enter the file name, main.c, and click Finish.
15. Add the following code to the main.c file.

```
1.    #include "sys/alt_stdio.h"
2.    #include "system.h"
3.    #include "priv/alt_busy_sleep.h"
4.    #include "sys/alt_sys_wrappers.h"
5.    #include "altera_avalon_pio_regs.h"
6.    #include <sys/alt_timestamp.h>
7.    #include <sys/alt_alarm.h>
```

```
8.
9.
10.     static alt_u32 toggleFlag = 0;
11.
12.     //Alarm callback will toggle the LEDs
13.     alt_u32 alarm_callback(void* context){
14.
15.             if(toggleFlag){
16.                     toggleFlag = 0;
17.                     IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, 0x15);
18.             }
19.             else{
20.                     toggleFlag = 1;
21.                     IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, 0xa);
22.             }
23.             //return alt_ticks_per_second(); //periodic alarm
24.             return 0;
25.     }
26.
27.
28.     int main()
29.     {
30.
31.             alt_putstr("Alarm test\n");
32.             IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, 0x18);
33.
34.             ///Set the alarm
35.             static alt_alarm countDownAlarm;
36.             if(alt_alarm_start(&countDownAlarm, 5000, alarm_callback,
        NULL)<0){
37.                     alt_putstr("No System Clock Found\n");
38.             }
39.
40.             while(1){
41.
42.                     for(int i = 0; i < 100; i++){
43.                             alt_printf("Application is running %x\n", i);
44.                             usleep(50000);
45.                     }
46.                     usleep(50000);
47.             }
48.
49.             return 0;
50.     }
```

The basic concept for programming on top of the provided HAL drivers is the HAL API Wrappers. The various driver header files contain the wrapper APIs that are used to access the timer and PIO.

The alarm needs a callback when the alarm triggers. More than one alarm and callback can be in the code. The callbacks are put into a linked list and managed by the interrupt handler. Lines 10-25 are the callback function for this exercise. The alarm will toggle the LEDs in a pattern based on

the toggleFlag. There is a return value of 0. The value 0 tells the timer driver that this is a one-shot alarm and removes the callback from the alarm link list. If an integer value or alt_ticks_per_second() is in the return, the alarm will fire off over and over again at the periodic rate until the alt_alarm_stop() function is called.

Lines 35-38 create an instance of the alarm and start the alarm countdown with a time of 5 seconds and assign the alarm_callback to respond when the countdown reaches zero. The rest of the program keeps the application alive.

16. Save the file.
17. Right-click on timerTest project again, and select Build Project. The build should complete successfully, and the timerTest.elf file has been created.
18. Close Eclipse

Now, we are ready to program the board with the design and debug the application.

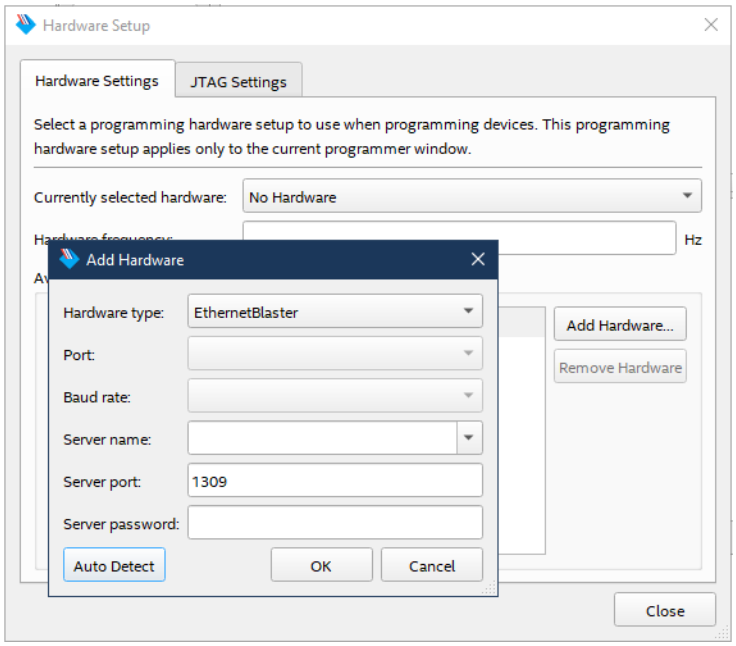
1.1.4 Program the Board

With the design compiled, application ready, and circuit connect, we can now test the design on the board.

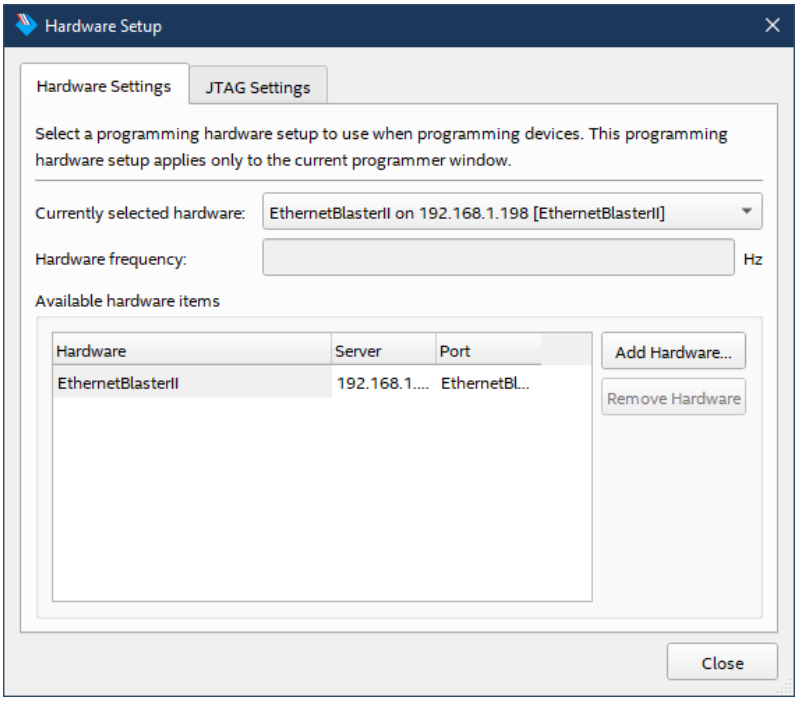
1. Connect the board and the programming cable together per the cable instructions.

Note: The MAX 10 – 10M08 Evaluation Kit doesn't come with a programming cable or built-in JTAG USB Blaster II. You will have to use either the USB Blaster II or EthernetBlaster II external cables. The EthernetBlaster II was used for this example. DHCP setup was not working so a direct Ethernet cable connection was made between a PC and the EthernetBlaster II. The static IP was set for the PC network card to 198.162.0.1. The EthernetBlaster II was accessed via a browser and then the IP address was changed to a static IP that matched the network. The new IP address was used as the Server name.

2. Power on the board and the programming cable box.
3. In Quartus Prime, from the Task pane, right-click on Program Device (Open Programmer) and select Open from the context menu or click on the  icon on the toolbar.
4. The Programmer dialog appears, click on the "Hardware Setup" button.
5. Click the Add hardware button, select the Hardware type and fill in any remaining information, and click OK.

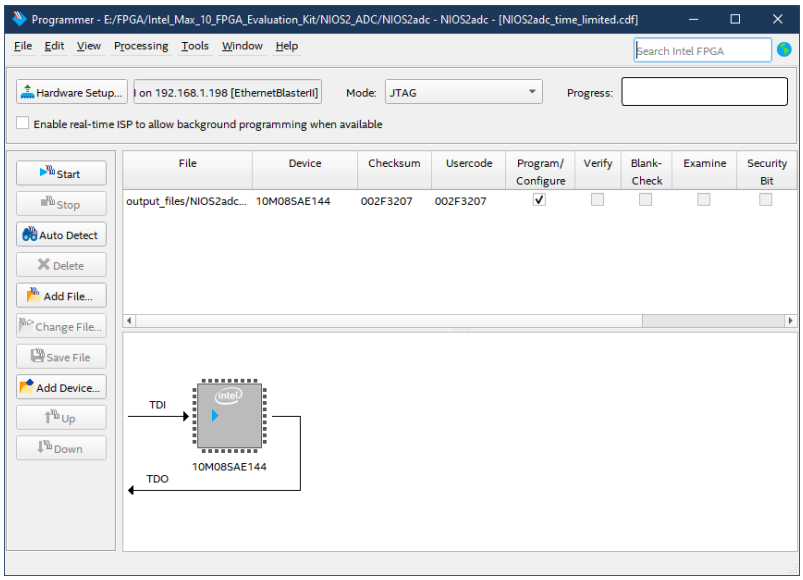


6. The tool allows you to connect to a number of programming cables. We need to select the one for our board. In the “Currently selected hardware”, click the drop-down and select the hardware cable for the board, and click Close when finished

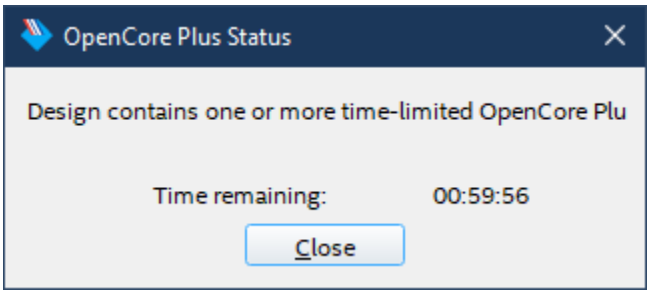


7. A NIOS2timer_time_limited.sof file gets created during the Compile Design flow. The file is automatically filled in. There is only one FPGA on the board and in the JTAG chain so the file already has the Program/Configure checkbox checked. Click the Start button to program the board. The process takes a few seconds and shows that the task was completed successfully.

Note: The reason for the “time_limited” in the name of the .sof file is that we chose a Nios II/f, which requires a license. The design must be connected to the JTAG cable or the system will shut off after an hour.



A dialog will appear that the design is time limited to one hour. The design can always be reloaded when the timeout occurs.



Important: This dialog acts as a tether to the time-limited IP. You must leave this dialog running while you are running applications.

1.1.5 Deploy the Application and Other Tests

With the design loaded and the connection to JTAG up and running, we can test the application.

1. From the Quartus menu, select Tools-> Nios II Software Build Tools for Eclipse.
2. Open the main.c application.
3. Right-click on timerTest and select Run As->Nios II Hardware. The program will load and start running.

The first 3 LEDs will turn on and after the alarm goes off, the LEDs will change to an on-off-on-off-on pattern. The application continues to run in the loop, but the alarm is done.

4. Edit main.c at line 24, and change the return from 0 to 3000.
5. Save the main.c file.
6. Rebuild the application.

7. Right-click on timerTest and select Run As->Nios II Hardware. The program will load and start running.

The first 3 LEDs will turn on and after the alarm goes off, the LEDs start flashing the two patterns periodically every 3 seconds. If we had alt_arlam_stop() in the code somewhere, this would stop the alarm.

8. Edit main.c again at line 23, and change the return from 3000 to 0. The code is going back to a one-shot alarm.
9. At line 36 change the countdown value from 5000 to 300.
10. After line 38 add the following code:

```
for(int i = 0; i < 10; i++){
      alt_printf("Time stamp running %x\n", i);
      usleep(50000);
}

if(alt_alarm_start(&countDownAlarm, 10000, alarm_callback, NULL)<0){
            alt_putstr("No System Clock Found\n");
}
```

11. Save the main.c file.
12. Rebuild the application.
13. Right-click on timerTest and select Run As->Nios II Hardware. The program will load and start running.

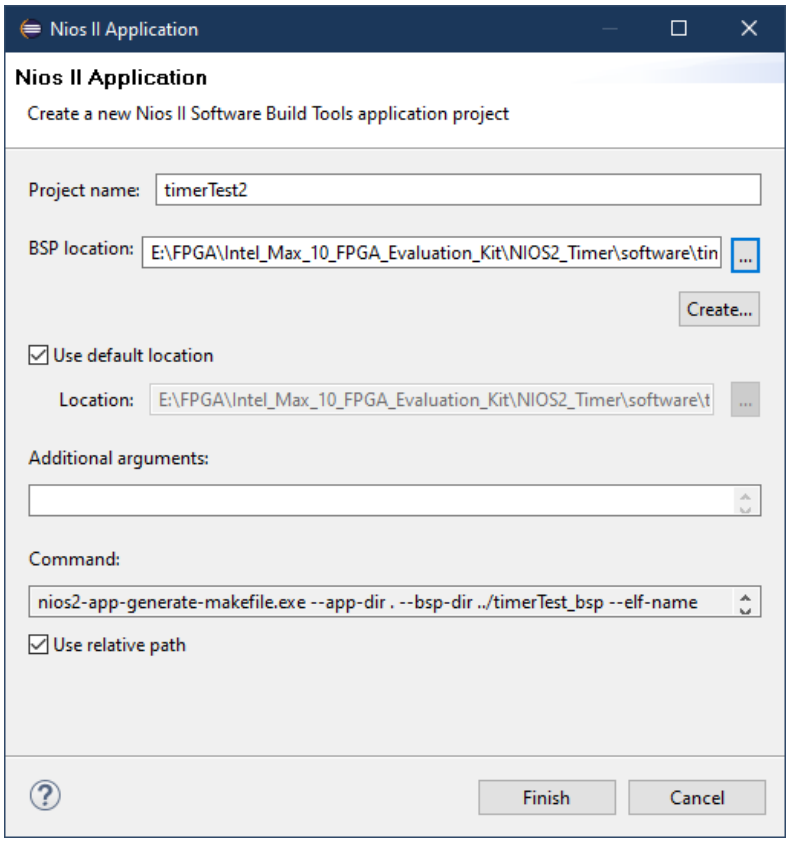
The alarm callback is set for a single-shot operation. Two alarms are set up for the same callback, but for the application to NOT crash, the first alarm has to finish before the second alarm can be set. Setting the second alarm just after the first for the same callback will crash the application. The loop between the two alarm_start() calls provides some delay. You will see the LEDs go through the pattern once and then the alarms are no longer set. This demonstrates that an alarm callback can be reused after it has been triggered.

If you want, you could create a different callback for the second alarm, that puts out a different LED pattern.

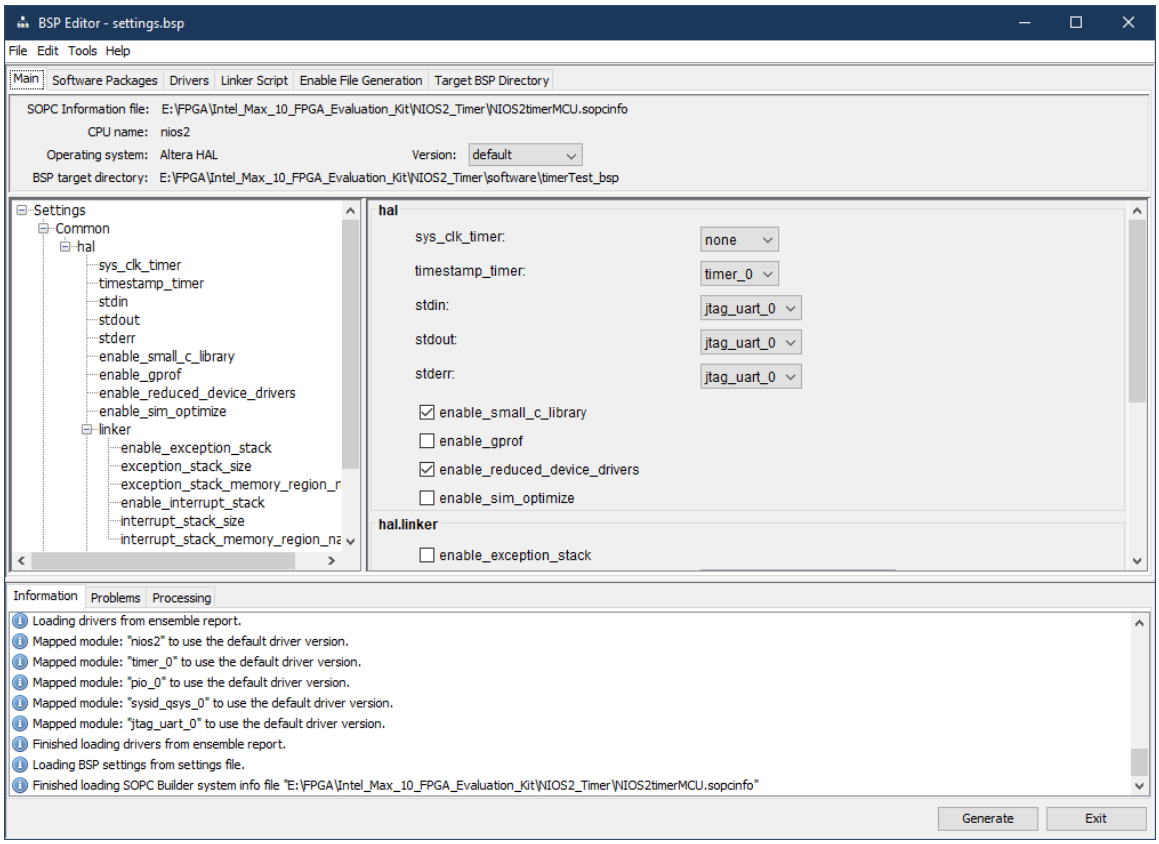
1.1.6 Eclipse Application: Timestamp

This second application will demonstrate the timestamp functionality of the Interval Timer.

1. From the menu in Eclipse, select File->Nios II Application.
2. Enter the project name timerTest2.
3. For the BSP Location, click on the 3 dot button.
4. Select timerTest_bsp and click OK.



5. Click Finish.
6. Now, we need to edit the BSP to change timer_0 to be assigned as a timestamp_timer. Right-click on timerTest_bsp and select Nios II->BSP Editor.
7. Change sys_clk_timer to none.
8. Change timerstamp_timer to timer_0.



9. Click Generate.
10. Click Finish.
11. Right-click on timerTest2 and select New->File.
12. Name the file main.c and click finish.
13. Add the following code to main.c:

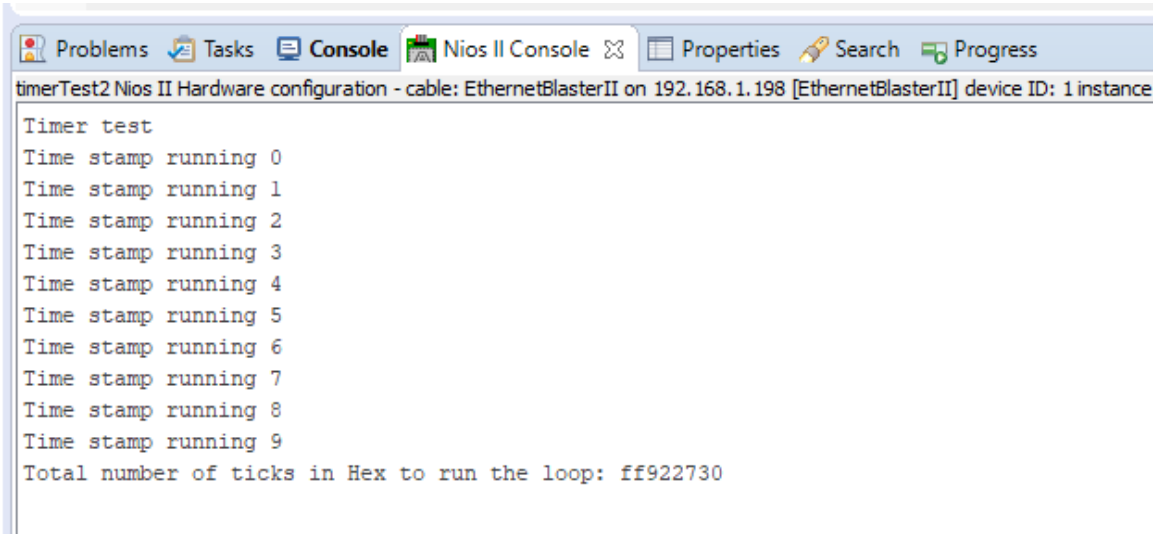
```
1.  #include "sys/alt_stdio.h"
2.  #include "system.h"
3.  #include "priv/alt_busy_sleep.h"
4.  #include "sys/alt_sys_wrappers.h"
5.  #include "altera_avalon_pio_regs.h"
6.  #include <sys/alt_timestamp.h>
7.  #include <sys/alt_alarm.h>
8.
9.
10. int main()
11. {
12.
13.         alt_putstr("Timer test\n");
14.         IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, 0x18);
15.
16.         alt_u8 timestampavailable = 0;
17.         alt_u32 timeBefore, timeAfter, timerOverhead, totalTicks;
18.
19.         if(alt_timestamp_start() <0){
20.                 alt_putstr("No timestamp timer found\n");
21.         }
22.         else{
23.                 timestampavailable = 1;
24.                 timeBefore = alt_timestamp();
```

```
25.                 timeAfter = alt_timestamp();
26.                 timerOverhead = timeAfter - timeBefore;
27.           }
28.
29.           //Test the time stamp functionality.
30.           if(timestampavailable){
31.
32.                 timeBefore = alt_timestamp();
33.                 for(int i = 0; i < 10; i++){
34.                       alt_printf("Time stamp running %x\n", i);
35.                       alt_busy_sleep(6000);
36.                 }
37.                 timeAfter = alt_timestamp();
38.
39.                 totalTicks = timeBefore-timeAfter-timerOverhead;
40.
41.                 alt_printf("Total number of ticks in Hex to run the loop:
      %x\n", totalTicks);
42.
43.           }
44.           while(1){
45.
46.                 usleep(5000);
47.           }
48.
49.           return 0;
50.     }
```

The application tests the number of timer-ticks it takes to run through the for-loop at lines 33-36. The application sets up the variables to be used to get the time stamp before and after the loop runs. Before it can perform the test, the overhead of performing both timestamp operations is calculated. The test is performed and the resulting number of ticks is presented as a hex value.

14. Save the application.
15. Build the application.
16. With the design's sopc file programmed to the FPGA, run the application.

The application simply outputs from the loop and posts the result. Timestamps can be helpful when diagnosing code that is time critical.



17. When finished close Eclipse, the OpenCore Plus dialog, and the JTAG programming application.

1.2 Summary: It is About Time

As a test, you can go back to the design and add a second timer (timer_1), and then in the timerTest_bsp edit the BSP to have one timer be the sys_clk_timer and the other be the timestamp_timer. A single application can support both alarms and timestamps.

1.3 References

The following references were used for this article:

Nios® II Processor: Hardware Abstraction Layer Exercise Manual, Intel Corporation,

Nios® II Software Developer Handbook, V21.3, Intel Corporation, 10/4/21