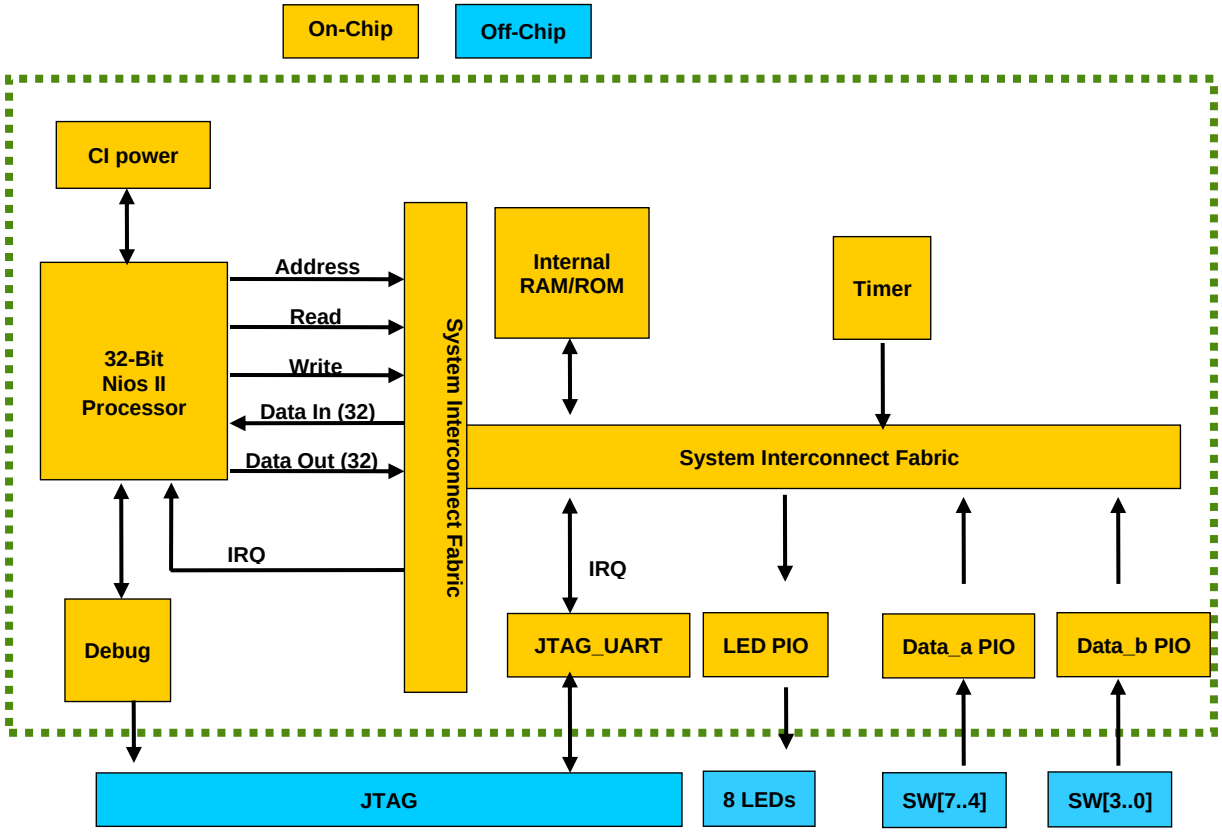
Задание labn_6

Введение:

Цель упражнения – познакомиться с возможностями процессора NIOSII по аппаратной акселерации с применением пользовательских инструкций.

Структура проекта



Алгоритм работы проекта:

Этап 1.

Под управлением процессора NIOSII обеспечивается:

- Программное выполнение умножения данных, поступающих со входов Data_a и Data_b;
- Измерение числа ticks, требуемых для реализации операции умножения и вывод соответствующей информации в окно консоли.

Этап 2.

Под управлением процессора NIOSII обеспечивается:

- Аппаратное (пользовательская инструкция CI_Power) выполнение умножения данных, поступающих со входов Data_a и Data_b;
- Измерение числа ticks, требуемых для реализации операции умножения и вывод соответствующей информации в окно консоли.

Часть 1 – Создание проекта

1. Запустите пакет QuarusII
2. В меню **File** менеджера пакета, укажите **New Project Wizard…**.
3. На экране появится окно введения - **Introduction** (если оно небыло отключено). Нажмите кнопку **next**.
4. В появившемся окне введите следующие данные:

What is the working directory for this project? Рабочая папка (*с помощью браузера найдите рабочую папку проекта*)	***C:\Intel_trn\Q_NIOS*****Lab6**
What is the name of this project? Имя проекта	Lab6
What is the name of the top-level design entity for this project? Имя модуля верхнего уровня в иерархии проекта.	Lab6

- в разделе **Available devices** укажите СБИС **EP4CE6E22C8**.

Часть 2 – Создание аппаратной части проекта

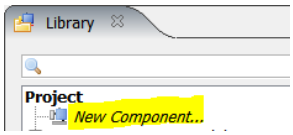
1. Скопируйте систему из проекта Lab5_nios.qsys в рабочую папку проекта.
2. Выполните команду **Tools => Qsys.** Будет запущен Qsys.
3. Откройте систему Lab5_nios.qsys и сохраните ее под именем Lab65_nios.qsys

4. В пакете QII создайте Verilog файл: CI_power.sv:

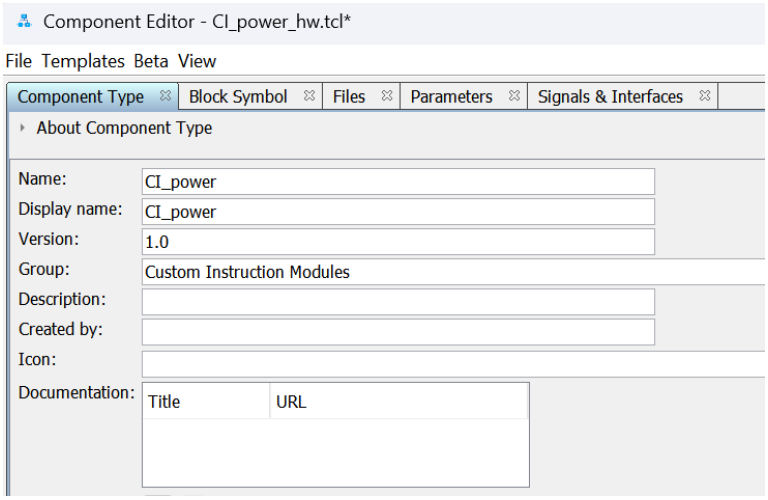
```
1    module CI_Power (
2        input  bit [31:0] dataa,    // Operand A (always required)
3        input  bit [31:0] datab,    // Operand B (optional)
4        output bit [31:0] result    //result (always required)
5    );
6    assign result = dataa*datab;
7    endmodule
```

5. Сохраните его и проверьте синтаксис (Analyse and Elaborate)

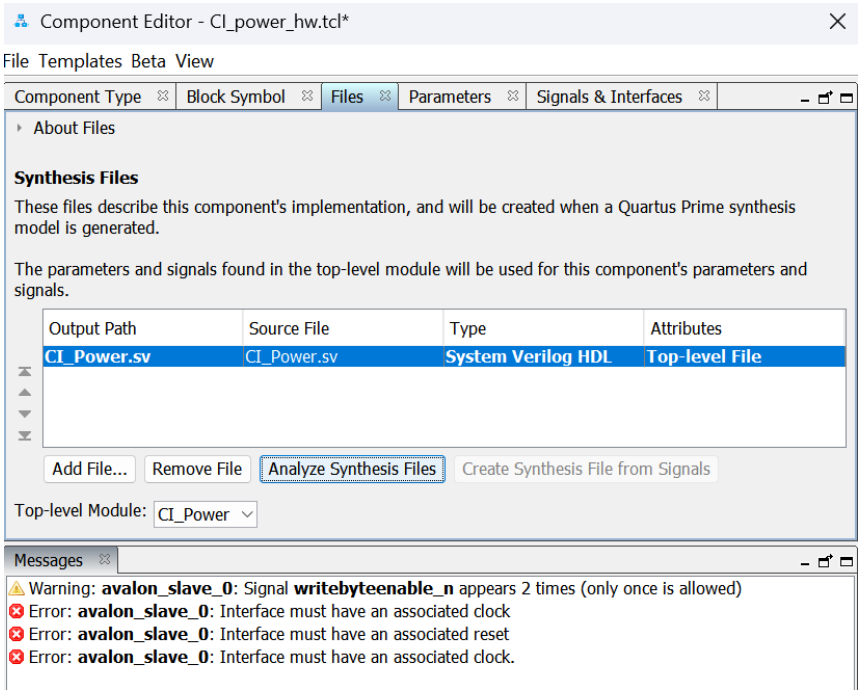
6. В пакете Qsys выполните команду New Components



7. На закладке Component Type укажите:

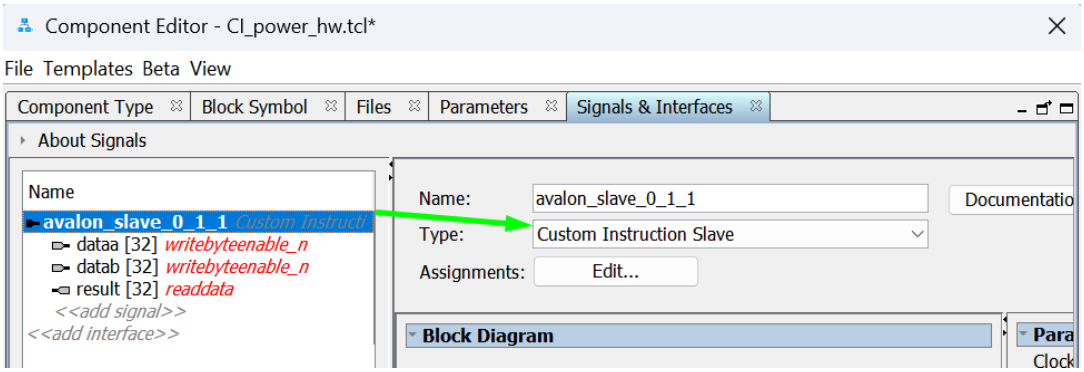


8. На закладке Files (в разделе Synthesis Files) укажите CI_power.sv и выполните Analyze Synthesis File (появляющиеся ошибки пока можно проигнорировать)

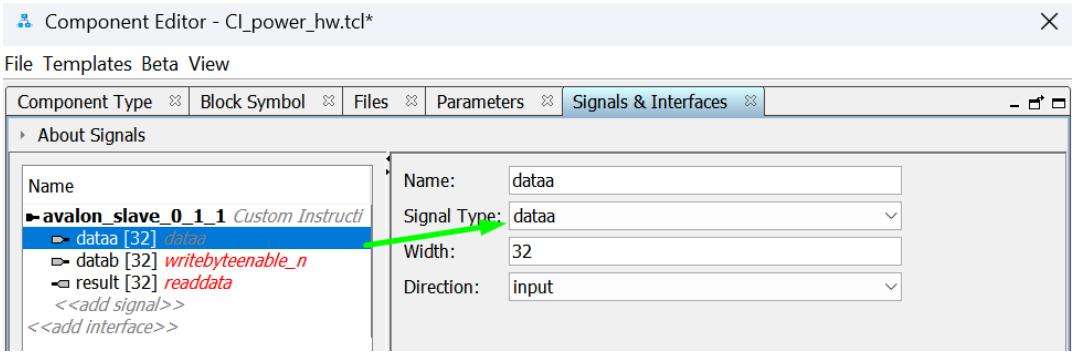


9. На закладке Signal & Interfaces задайте:

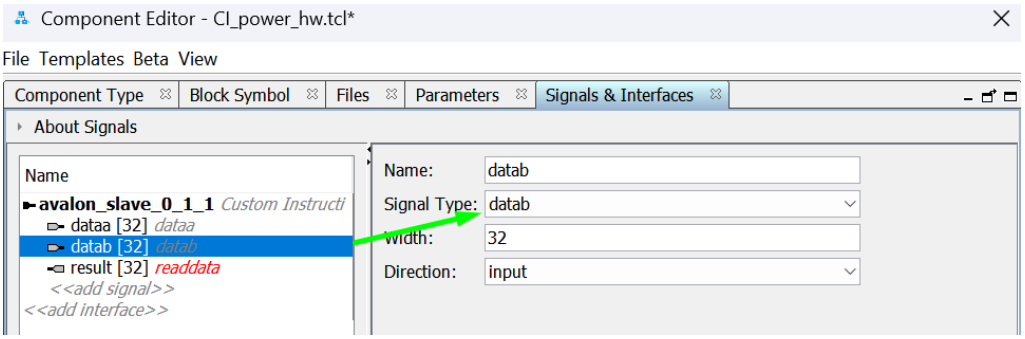
✓



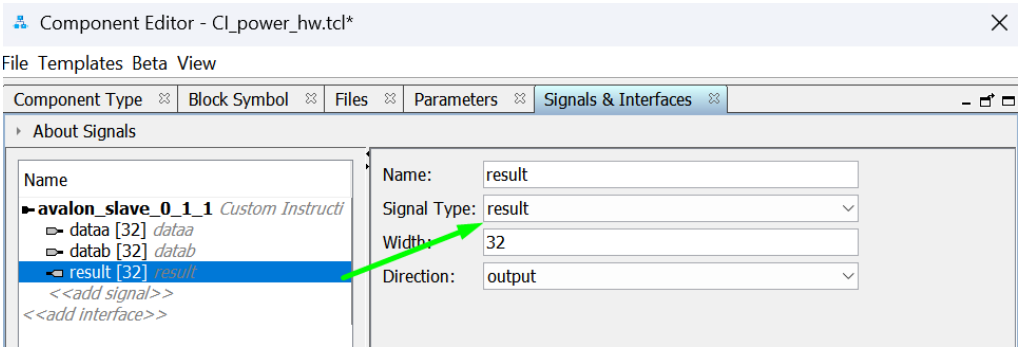
✓



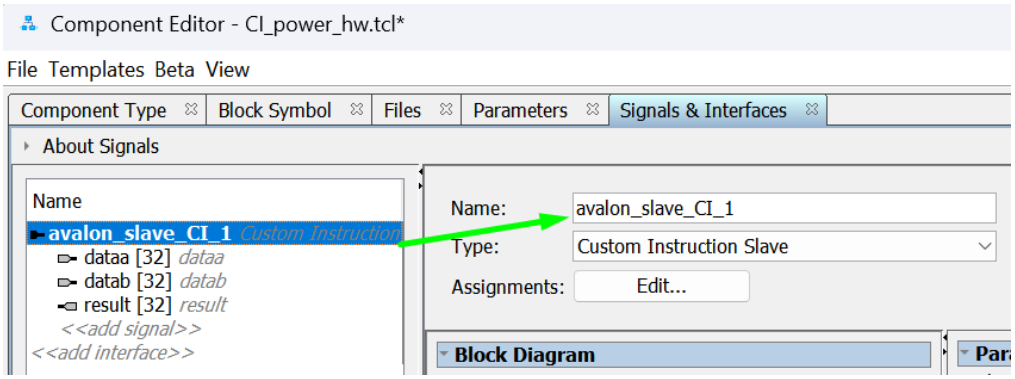
✓


Component Editor - CI_power_hw.tcl*
File Templates Beta View
Component Type
Block Symbol
Files
Parameters
Signals & Interfaces
About Signals
Name
avalon_slave_0_1_1 Custom Instructi
dataa [32] dataa
datab [32] datab
result [32] readdata
<<add signal>>
<<add interface>>
Name: datab
Signal Type: datab
Width: 32
Direction: input

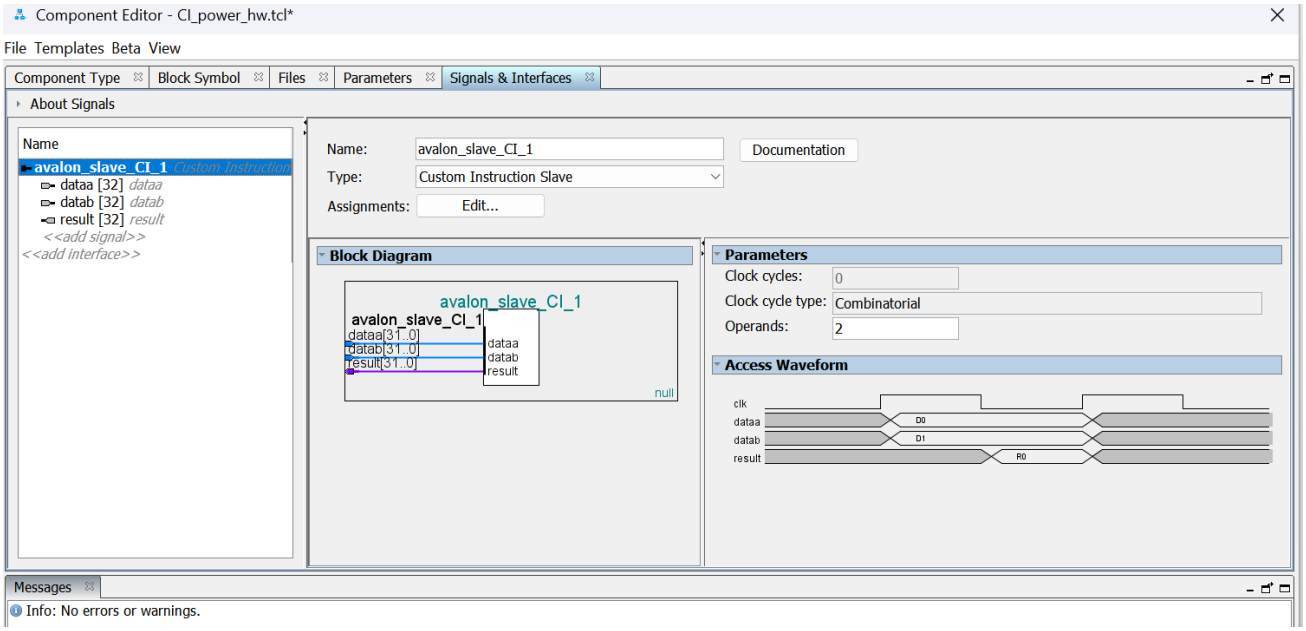
✓


Component Editor - CI_power_hw.tcl*
File Templates Beta View
Component Type
Block Symbol
Files
Parameters
Signals & Interfaces
About Signals
Name
avalon_slave_0_1_1 Custom Instructi
dataa [32] dataa
datab [32] datab
result [32] result
<<add signal>>
<<add interface>>
Name: result
Signal Type: result
Width: 32
Direction: output

✓


Component Editor - CI_power_hw.tcl*
File Templates Beta View
Component Type
Block Symbol
Files
Parameters
Signals & Interfaces
About Signals
Name
avalon_slave_CI_1 Custom Instruction
dataa [32] dataa
datab [32] datab
result [32] result
<<add signal>>
<<add interface>>
Name: avalon_slave_CI_1
Type: Custom Instruction Slave
Assignments: Edit...
Block Diagram

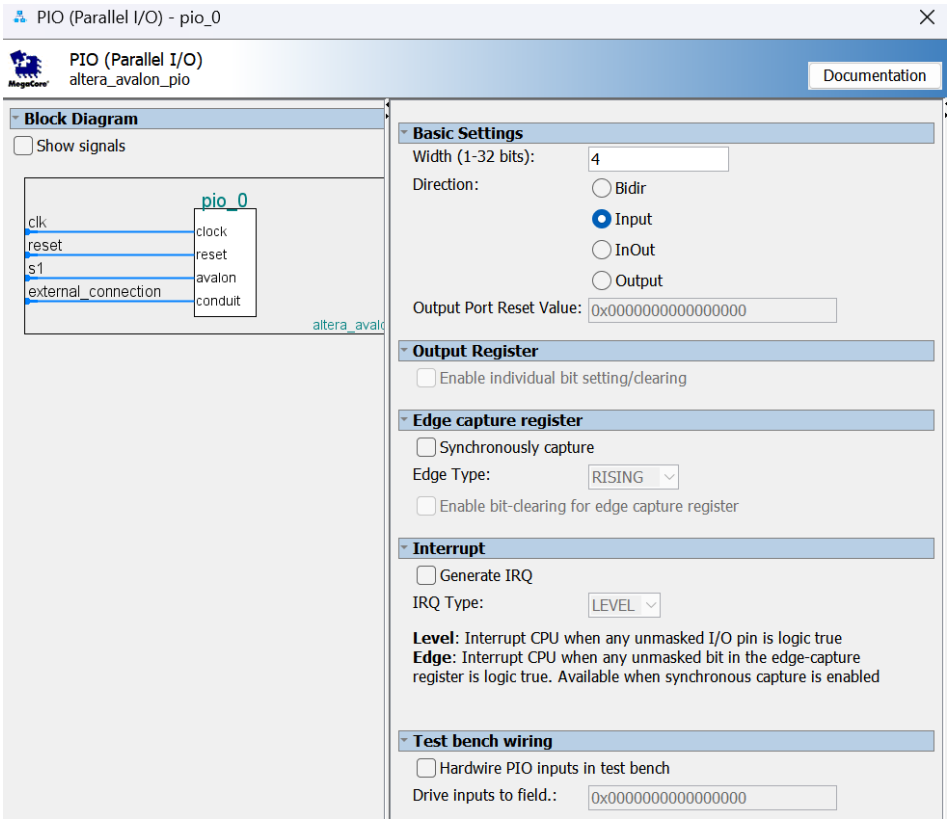
10. Проверьте сделанные назначения



Ошибки и предупреждения должны отсутствовать.

11. Нажмите кнопку Finish. И сохраните изменения.
12. В папке Project появится раздел Custom Instruction Modules, а в нем – модуль CI_power.
13. Добавьте модуль к системе.
14. Переименуйте модуль CI_power_0: новое имя CI_PWR
15. Подключите вывод avalon_slave_CI_1 модуля CI_PWR к выводу custom_instruction_master модуля nios2_qsys

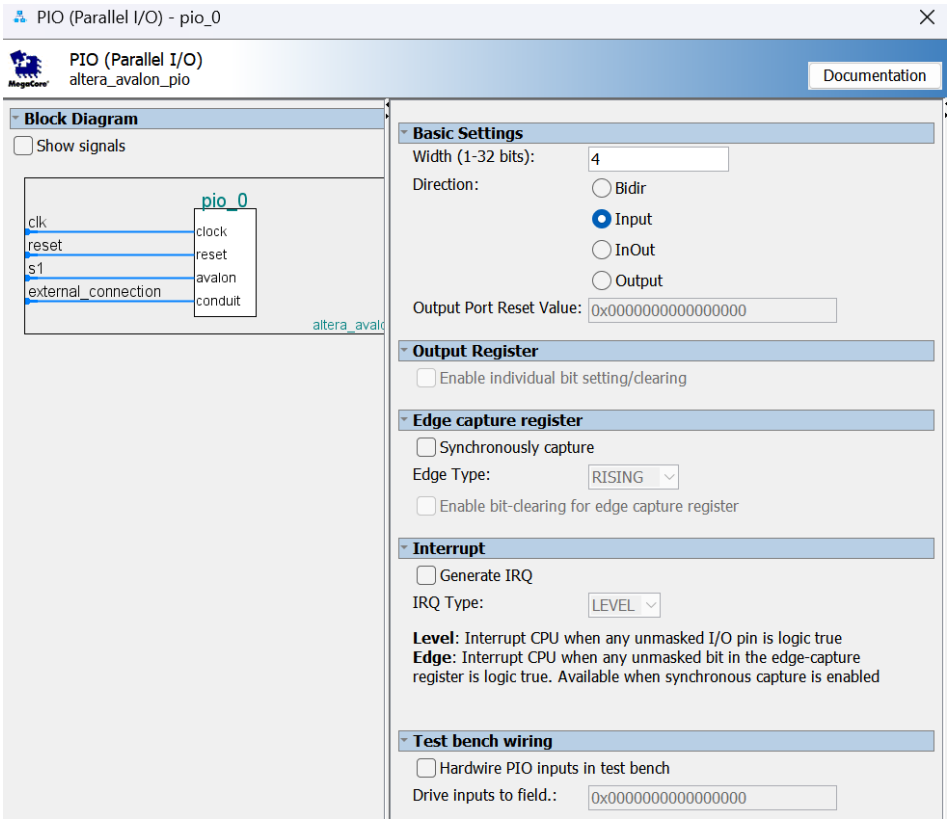
16. Добавьте к системе модуль PIO для входов dataa.



17. Переименуйте добавленный модуль: новое имя Data_a.

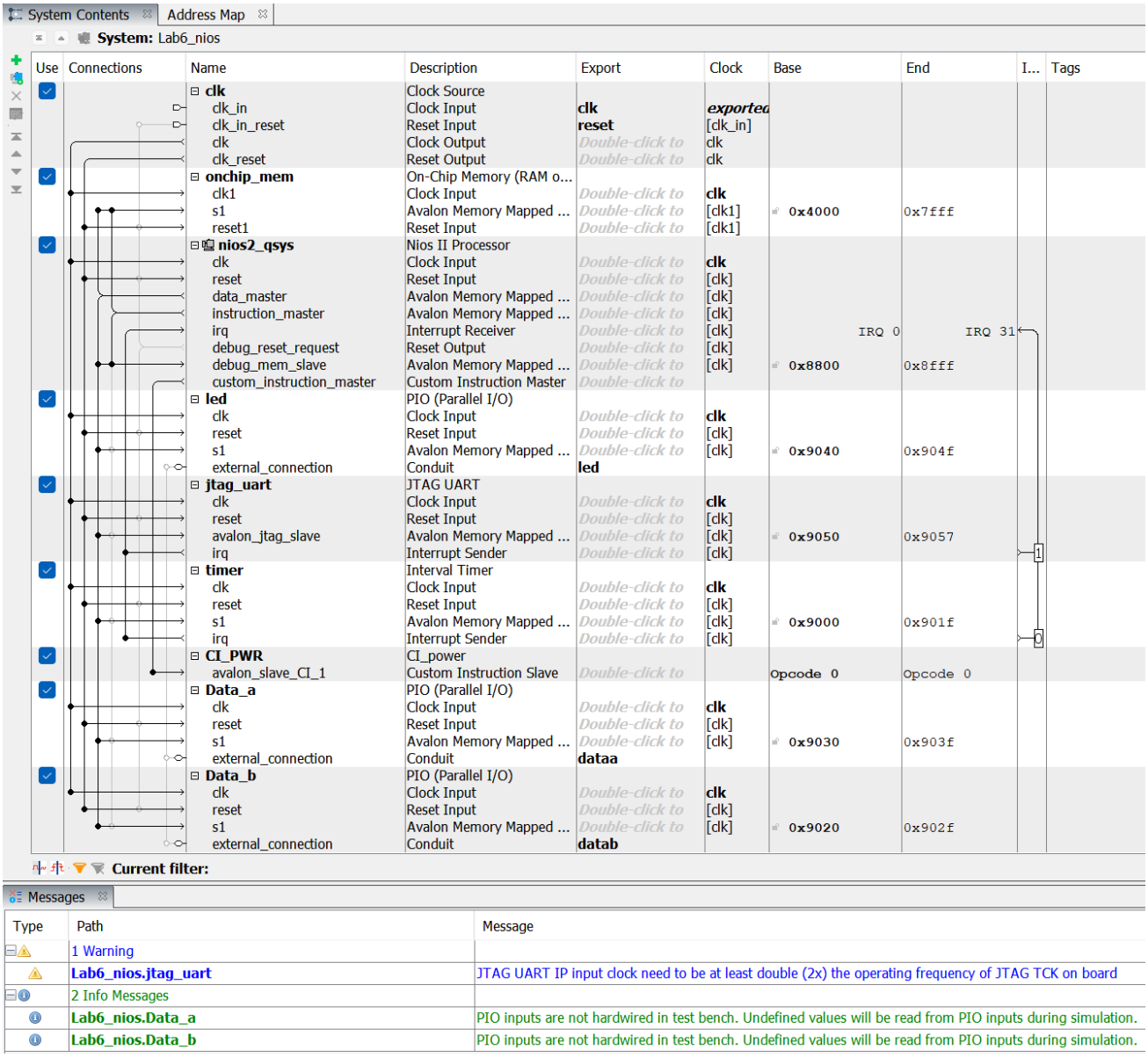
18. Подключите модуль Data_a к создаваемой системе (экспортируемый вывод назовите dataa).

19. Добавьте к системе модуль PIO для входов datab.



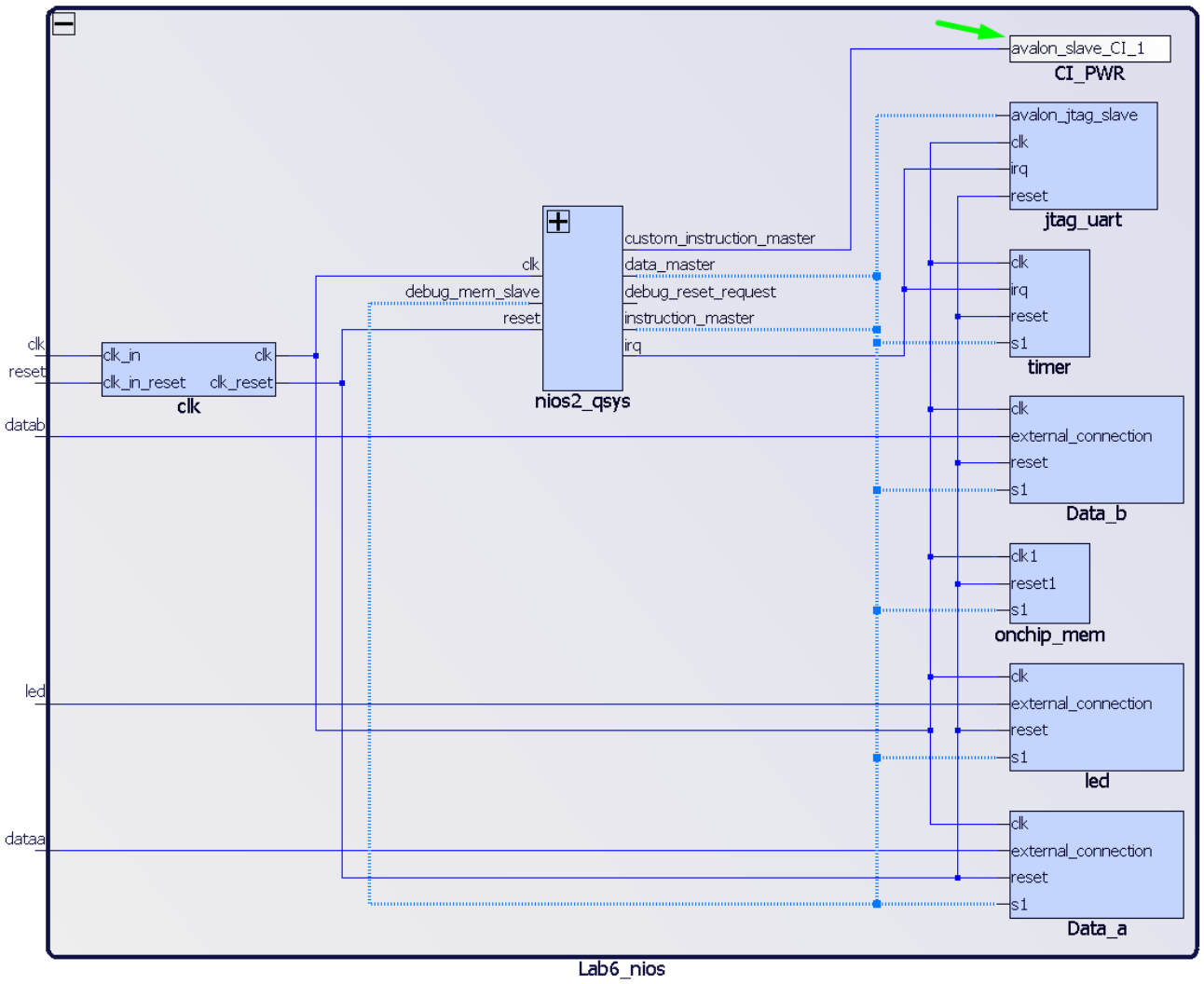
20. Переименуйте добавленный модуль: новое имя Data_b.
21. Подключите модуль Data_b к создаваемой системе (экспортируемый вывод назовите datab).
22. Выполните команду System => Assign Base Addresses

23. Проверьте созданную систему (она должна быть похожа на приведенную на рисунке)



24. Выполните команду View => Schematic.

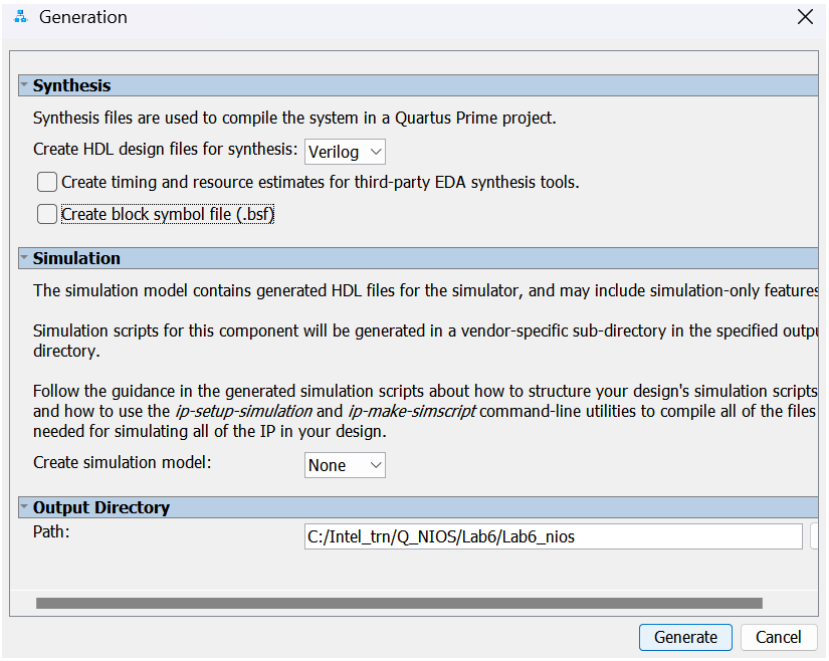
    Проверьте созданную систему (она должна быть похожа на приведенную на рисунке)



25. Выполните команду File=>Save

26. Выполните команду Generate=>Generate HDL

27. В окне задайте параметры как показано на рисунке



28. Нажмите кнопку Generate
29. Не должно появится ошибок генерации. Предупреждения

Warning: **Lab6_nios.jtag_uart**: JTAG UART IP input clock need to be at least double (2x) the operating frequency of JTAG TCK on board
Info: **Lab6_nios**: Generating **Lab6_nios "Lab6_nios"** for QUARTUS_SYNTH
Warning: **Lab6_nios**: "No matching role found for nios2_qsys:custom_instruction_master:E_ci_multi_clock (clk)"
Warning: **Lab6_nios**: "No matching role found for nios2_qsys:custom_instruction_master:E_ci_multi_reset_req (reset_req)"
Warning: **Lab6_nios**: "No matching role found for nios2_qsys:custom_instruction_master:E_ci_multi_reset (reset)"

можно проигнорировать.

Часть 3 – Интеграция аппаратной части проекта

1. Создайте в текстовом редакторе файл (имя файла Lab6.sv) верхнего уровня в иерархии проекта (для этого целесообразно использовать файл Lab6_nios_inst.v из папки C:\Intel_trn\Q_NIOS\Lab6\Lab6_nios)
    a. Пример файла приведен на рисунке

```
module Lab6 (
    input  bit clk,     // Clock
    input  bit pbb,     // System reset active low
    input  bit [7:0] sw,// dataa: sw[7:4]; datab sw[3:0]
    output bit [7:0] led
);
    Lab6_nios u0 (
        .clk_clk       (clk),       //   clk.clk
        .led_export    (led),    //   led.export
        .reset_reset_n (pbb), // reset.reset_n
        .dataa_export  (sw[7:4]),  // dataa.export
        .datab_export  (sw[3:0])   // datab.export
    );
endmodule
```

2. Назначьте файл Lab6.sv файлом верхнего уровня в иерархии проекта.
3. Добавьте к проекту файл Lab6_nios.qip из папки C:\Intel_trn\Q_NIOS\Lab5\Lab5_nios\synthesis
4. Проверка синтаксиса проекта.
    a. Выполните команду Processing=>Start=>Start Analysis and Elaboration
5. Назначение выводов проекта. Назначьте выводы так, как показано на рисунке ниже

Node Name	Direction	Location	I/O Bank	/REF Group	I/O Standard	Current Strength
clk	Input	PIN_23	1	B1_N0	3.3-V LVTTL	8mA (default)
led[7]	Output	PIN_65	4	B4_N0	2.5 V	8ma
led[6]	Output	PIN_66	4	B4_N0	2.5 V	8ma
led[5]	Output	PIN_67	4	B4_N0	2.5 V	8ma
led[4]	Output	PIN_68	4	B4_N0	2.5 V	8ma
led[3]	Output	PIN_69	4	B4_N0	2.5 V	8ma
led[2]	Output	PIN_70	4	B4_N0	2.5 V	8ma
led[1]	Output	PIN_71	4	B4_N0	2.5 V	8ma
led[0]	Output	PIN_72	4	B4_N0	2.5 V	8ma
pbb	Input	PIN_58	4	B4_N0	2.5 V	8mA (default)
sw[7]	Input	PIN_88	5	B5_N0	3.3-V LVTTL	8mA (default)
sw[6]	Input	PIN_89	5	B5_N0	3.3-V LVTTL	8mA (default)
sw[5]	Input	PIN_90	6	B6_N0	3.3-V LVTTL	8mA (default)
sw[4]	Input	PIN_91	6	B6_N0	3.3-V LVTTL	8mA (default)
sw[3]	Input	PIN_49	3	B3_N0	3.3-V LVTTL	8mA (default)
sw[2]	Input	PIN_46	3	B3_N0	3.3-V LVTTL	8mA (default)
sw[1]	Input	PIN_25	2	B2_N0	3.3-V LVTTL	8mA (default)
sw[0]	Input	PIN_24	2	B2_N0	3.3-V LVTTL	8mA (default)

30. Назначение опции проекта:
    Assignment=>Device=>Device and Pin Options=>Unused pin=>As input tri-stated with weak pull-up resistor

31. С помощью Timing Analyzer или в текстовом редакторе создайте файл (**Lab6.sdc**) с требованиями к временным параметрам проекта. Пример файла приведен на рисунке

```
##
## DEVICE  "EP4CE6E22C8"
##
#**************************************************************
# Time Information
#**************************************************************

set_time_format -unit ns -decimal_places 3

#**************************************************************
# Create Clock
#**************************************************************

create_clock -name {clock} -period 40.000 -waveform { 0.000 20.000 } [get_ports {clk}]

#**************************************************************
# Set Clock Uncertainty
#**************************************************************

derive_clock_uncertainty

#**************************************************************
# Set Input Delay
#**************************************************************

set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {pbb}]
set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {sw[0]}]
set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {sw[1]}]
set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {sw[2]}]
set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {sw[3]}]
set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {sw[4]}]
set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {sw[5]}]
set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {sw[6]}]
set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {sw[7]}]

set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {altera_reserved_tdi}]
set_input_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {altera_reserved_tms}]

#**************************************************************
# Set Output Delay
#**************************************************************

set_output_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {led[0]}]
set_output_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {led[1]}]
set_output_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {led[2]}]
set_output_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {led[3]}]
set_output_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {led[4]}]
set_output_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {led[5]}]
set_output_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {led[6]}]
set_output_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {led[7]}]

set_output_delay -add_delay  -clock [get_clocks {clock}]  10.000 [get_ports {altera_reserved_tdo}]
```

32. Подключите файл Lab6.sdc к проекту.
33. Осуществите полную компиляцию проекта: команда **Processing => Start Compilation**.
34. Компиляция должна завершиться без ошибок. Все требования к временным параметрам должны быть выполнены.
35. Осуществите программрование СБИС ПЛ на плате: *на плате загорится зеленый светодиод.*

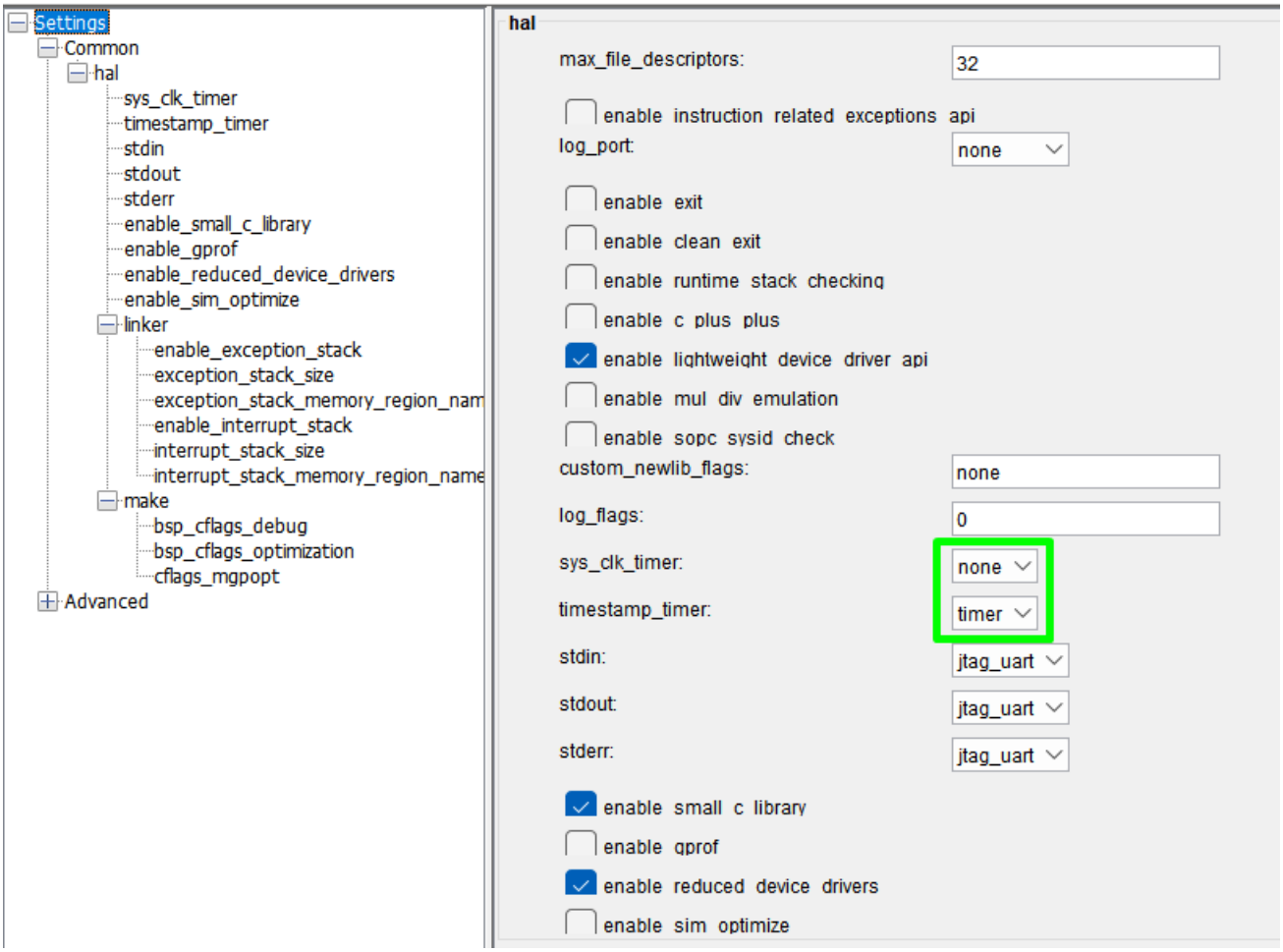
Часть 4 – Создание программной части проекта

Этап 1 – выполнение операции умножения программным образом

1. Запустите оболочку для разработки/отладки программ - NIOSII SBT
   a. Рабочую папку задайте как: C:\Intel_trn\Q_NIOS\Lab6\workspace
2. Создайте проект lab6_sw (lab6_sw и lab6_sw_bsp)
3. В проекте lab6_sw создайте исходный файл lab6_source_sw.c
4. Введите исходный код программы

```c
#include <stdio.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include <time.h>
#include <unistd.h>
#include <sys/alt_timestamp.h>
int main(){
  int led;
  int in_a;
  int in_b;
  alt_u32 num_ticks = 0;
  alt_u32 time1, time2, timer_overhead;
  // Check Timer:
      if (alt_timestamp_start() < 0)
        printf("Îðîãðàììà èíèöèàëèçàöèè òàéìåðà \n");
      else
        printf("Îðîöåññîð  Nios II çàïóùåí!\n");
        printf("\n ×àñòîòà ðàáîòû îðîöåññîðà  - CPU clock (Ãö): %u \r \n",(unsigned int)alt_timestamp_freq());
  // Initialize the registers in the LED_PIO peripheral:
  IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE,0x00);
   while (1) {
    if (alt_timestamp_start() < 0)
        printf("Îðîãðàììà èíèöèàëèçàöèè òàéìåðà \n");
    // Determine the timer overhead involved to record time stamp:  ***
      time1 = alt_timestamp();//alt_nticks();
      time2 = alt_timestamp();//alt_nticks();
      timer_overhead = time2 - time1;
    //Input data
      in_a = IORD_ALTERA_AVALON_PIO_DATA(DATA_A_BASE);
      in_b = IORD_ALTERA_AVALON_PIO_DATA(DATA_B_BASE);
    //start time
      time1 = alt_timestamp();
    // Multiplication
      led=(in_a*in_b);
    //END time
      time2 = alt_timestamp();
      num_ticks = time2 - time1 - timer_overhead;
    //output
      IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE,~led);
     // Print out the time it took to perform this loop:
      printf("Ðåçóëüòàò %u         ×èñëî  ticks %u \n", (unsigned int) led, (unsigned int)num_ticks);
      usleep(500000);
  }//end while loop
  return 0;
}
```

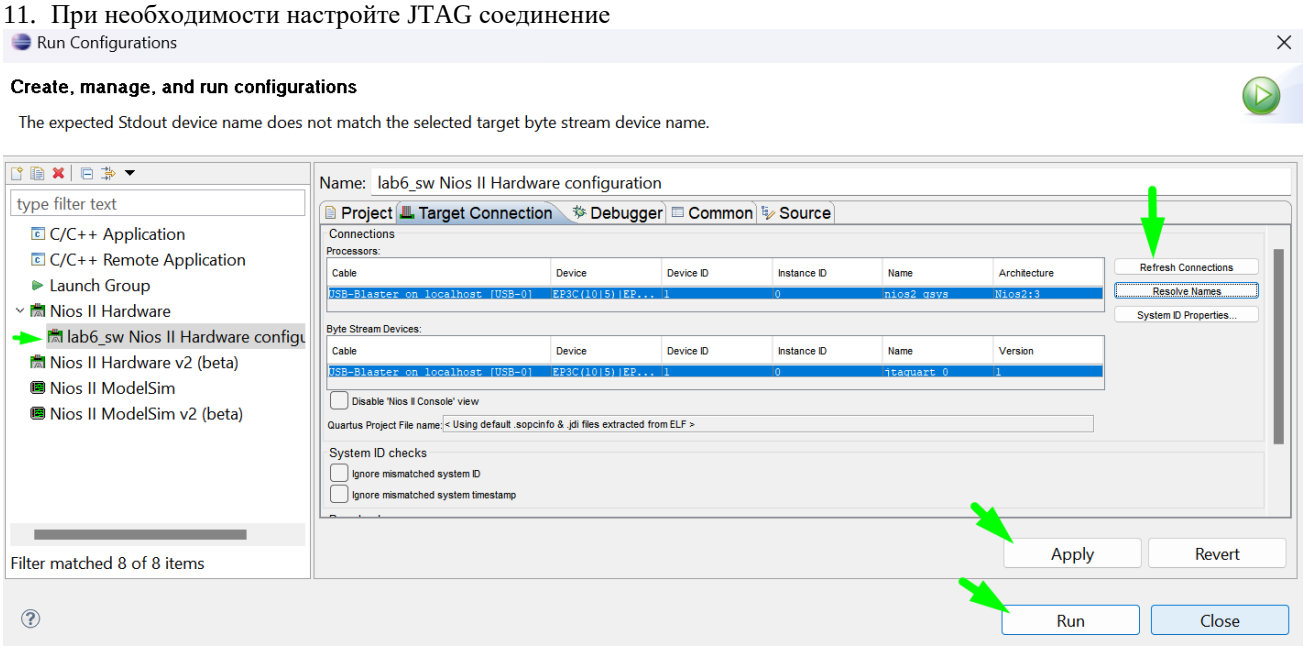
5. В lab6_sw_bsp установите опции, минимизирующие объем кода исполняемого файла и настройте таймеры как показано на рисунке, приведенном ниже.



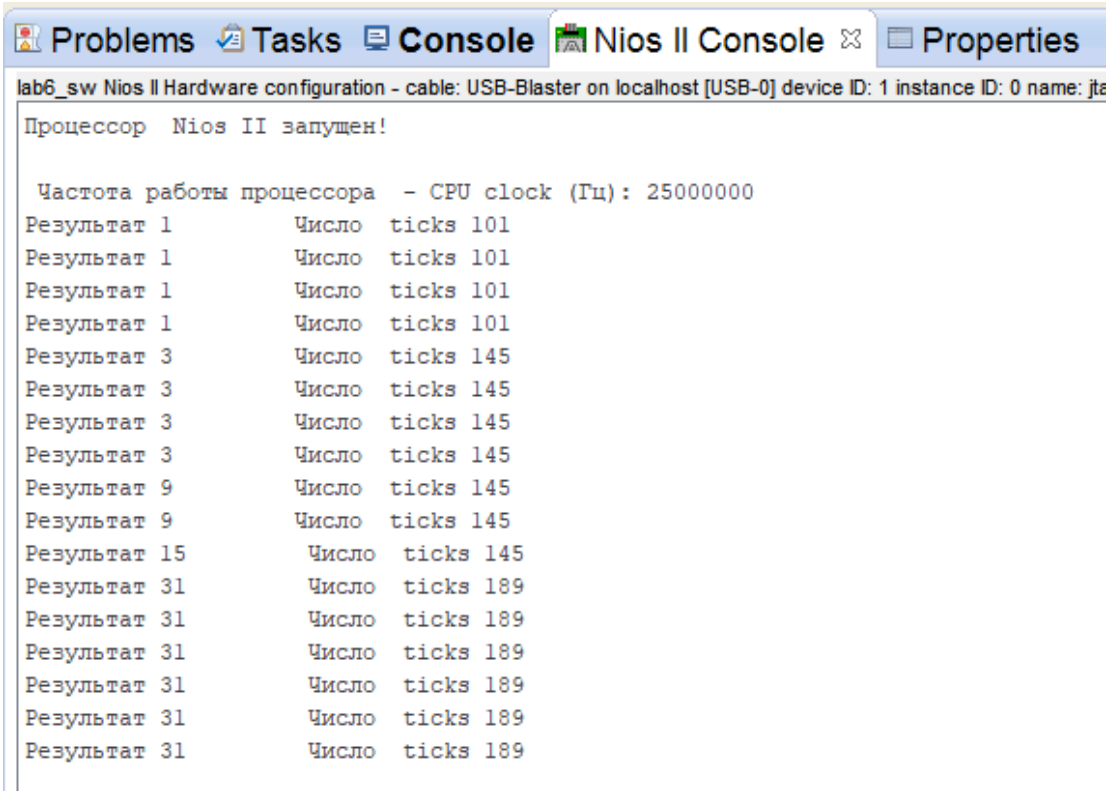
6. Нажмите кнопку generate, затем exit.
7. Запустите порождение испольняемого кода для проекта lab6_sw.
8. Проверьте результат компиляции:

```
CDT Build Console [lab6_sw]
nios2-elf-g++  -T'../lab6_sw_bsp//linker.x' -msys-crt0='../lab6_sw_bsp//obj/HAL/src/crt0.o' -
nios2-elf-insert lab6_sw.elf --thread_model hal --cpu_name nios2_qsys --qsys true --simulatio
Info: (lab6_sw.elf) 4868 Bytes program size (code + initialized data).
Info:               11 KBytes free for stack + heap.
Info: Creating lab6_sw.objdump
nios2-elf-objdump --disassemble --syms --all-header --source lab6_sw.elf >lab6_sw.objdump
[lab6_sw build complete]
```

9. ***На переключателях платы задайте dataa (sw [7:4])=1 и datab(sw[3:0])=1***
10. В NIOSII SBT выберите папку lab6_sw и запустите загрузку и выполнение программы на плате
11. При необходимости настройте JTAG соединение



12. Проверьте работу программы (изменяя значения data и datab):



13. Зафиксируйте (для нескольких наборов dataa и datab) число ticks, потраченное на выполнение 1 умножения чисел (при разных числах, введенных с переключателей sw, число ticks будет разным).

Этап 2 – использование пользовательской инструкции CI_power

1. В файле lab6_source_sw.c внесите следующие изменения:

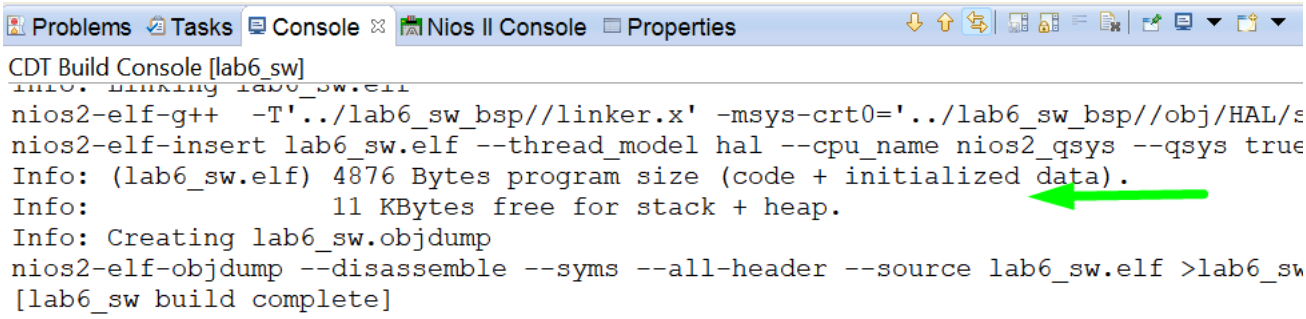
   a. Было:
   ```
   led=(in_a*in_b);
   ```

   b. Стало:
   ```
   led=ALT_CI_CI_PWR(in_a, in_b);
   ```

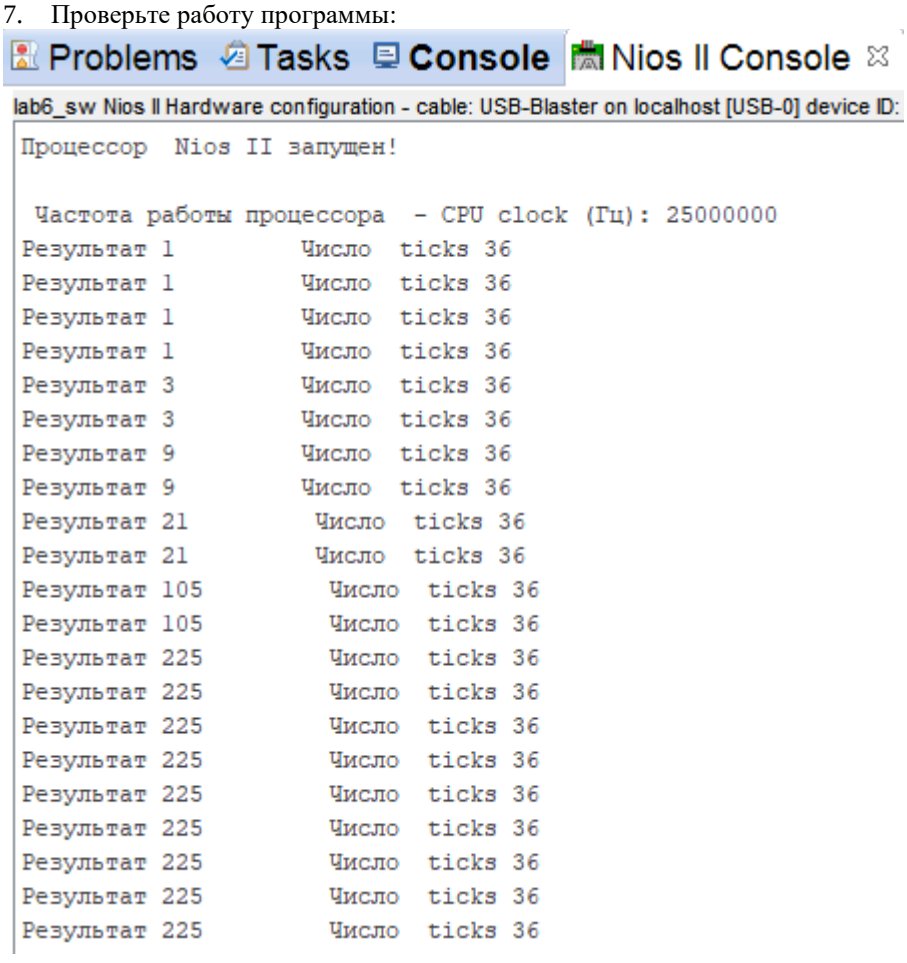
2. Сохраните его под именем lab6_source_hw.c

3. Исключите файл lab6_source_sw.c из компиляции

4. Запустите порождение испольняемого кода для проекта lab6_sw
5. Проверьте результат компиляции:



6. В NIOSII SBT выберите папку lab6_sw и запустите загрузку и выполнение программы на плате
7. Проверьте работу программы:



8. Зафиксируйте (для нескольких наборов dataa и datab) число ticks, потраченное на выполнение 1 умножения чисел (при разных числах, введенных с переключателей sw, число ticks будет **одинаковым**).
9. Сравните значения, полученные на данном этапе, со значениями полученным в рамках выполнения этапа 1.

Часть 5 – Дополнительное задание

10. Для Вашей операции (соответствует номеру по списку группы)

    a. Создайте пользовательскую инструкцию (код на языке Verilog)

    b. Интегрируйте ее в QSYS

    c. Обновите систему (CI_PWR надо убрать и добавить Вашу CI_xxx)

    d. Обновите программы lab6_source_sw.c (новое имя lab6_MY_sw.c) и lab6_source_hw.c (новое имя lab6_MY_hw.c)

    e. Зафиксируйте число ticks для программной и аппаратной реализации Вашей инструкции (обратите внимание: при переполнении 8 бит результат на светодиодах будет отображаться неправильно. Это допустимо.)

Номер в списке группы	Пользовательская инструкция
1	a*b+a
2	a*b+b
3	a*b-a
4	a*b-b
5	a*b+a-b
6	a*b-a+b
7	a*b-a-b
8	a*b+a+b
9	a*a+a+b
10	a*a+a-b
11	a*a-a+b
12	a*a-a-b
13	a*a+a+b
14	(a+b)*a
15	(a+b)*b
16	(a+a)*b
17	(b+b)*a
18	(a+b)*a -a
19	(a+b)*a + a
20	(a+b)*a - b
21	(a+b)*a + b
22	(a+b)*b -a
24	(a+b)*b + a
25	(a+b)*b - b
26	(a+b)*b + b

Упражнение 6 завершено.