Interconnections
Second Laboratory Session

Prof. Claudio Passerone

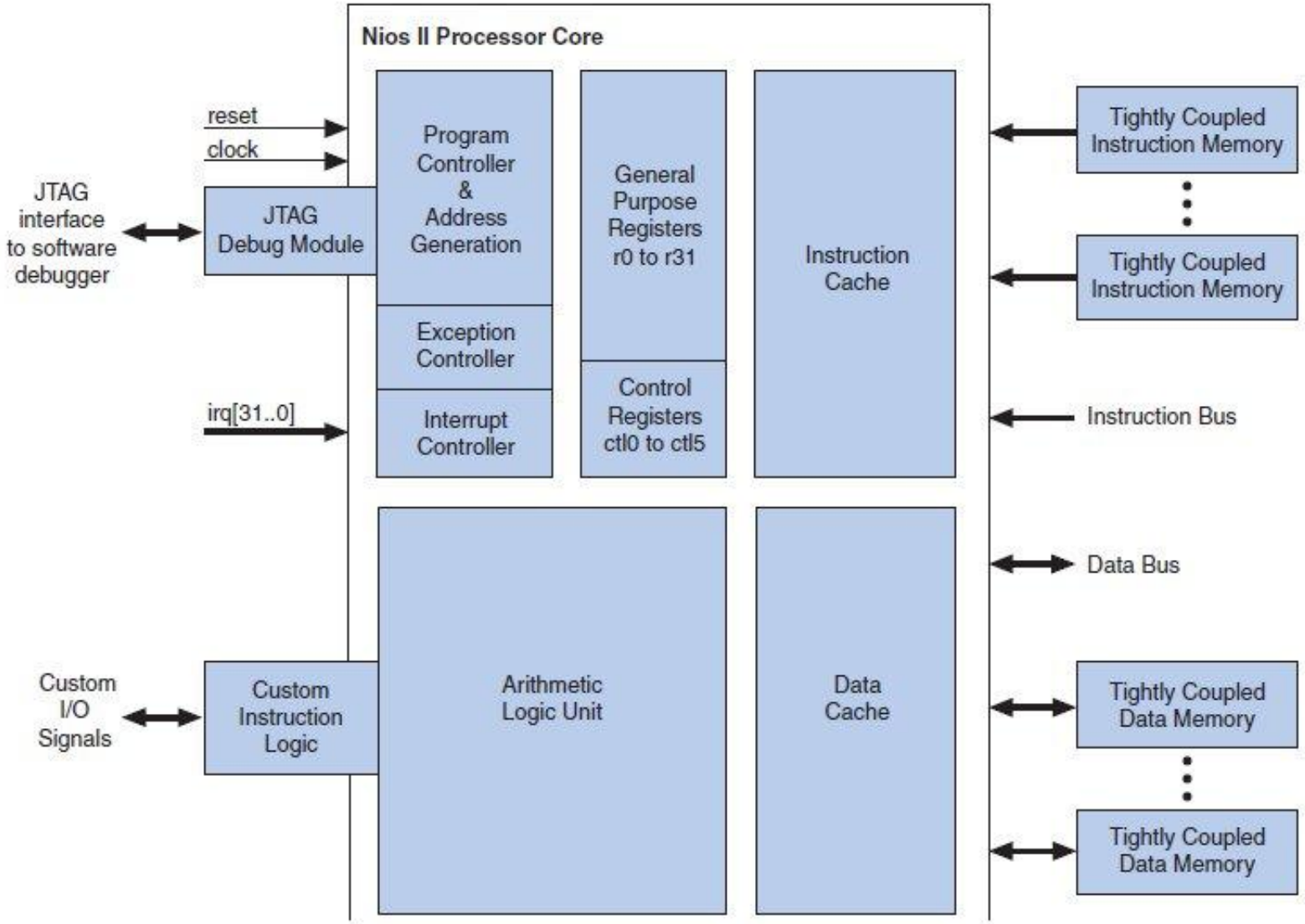
(Electronics for Embedded Systems)

Altera Nios II Processor

- Altera Nios II Processor
  - Soft processor
    - VHDL and Verilog sources
      - Intel/Altera does not sell Nios II integrated circuits
    - Synthesized to IC (rarely) or FPGA (Cyclone V or Cyclone 10 LP in our case)
    - Can be customized before synthesis
      - Trade offs between area (costs) and performance
        - Cache memory
        - Hardware multiplier and/or divider
        - Branch prediction
        - Custom instructions
    - Good compared to a standard microprocessor if the FPGA is needed for other reasons
      - Otherwise a normal microprocessor is cheaper and faster

Altera Nios II Processor

- Nios II Processor architecture
  - General purpose 32 bit RISC processor
    - Load & store architecture
  - 32 GP registers and 32 special registers
  - Integrated interrupt controller
  - ALU for arithmetic and logic functions
  - Modified Harvard architecture
    - Separate instruction and data caches
    - Common address space main memory



Altera Nios II Processor

- Altera Nios II Configuration Options
  - Different pipeline architecture
    - /e: economy, no pipeline
    - /s: standard, 5 stages (fetch, decode, execute, memory, writeback)
    - /f: fast, 6 stages (align after memory), branch prediction
  - Amount of cache
    - No cache, only I$ (decide size), both I$ and D$ (decide size)
    - Implemented using M9K or M10K blocks
  - Multipliers and dividers
    - SW emulation or HW implementation
    - HW can be on LEs or using embedded multipliers on Cyclone V or Cyclone 10 LP
  - After configuration and synthesis
    - Check compilation report for area and timing analysis (define a timing constraint for the clock)
    - Performance do not depend on the maximum clock frequency only
    - NiosII/e @ 50MHz gives lower performance than NiosII/f @ 40 MHz

Altera Nios II Processor

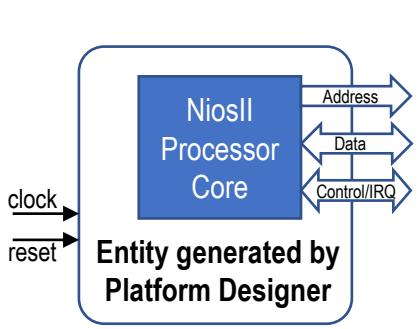
- Altera Nios II Instruction Set Categories
  - Data transfer (ldw, stw, ldwio, stwio, mov)
    - Also for single bytes or words
  - Integer arithmetic and logic
    - Always applied to registers or immediate values (add, sub, mul, div, and, or, xor, rol, ror, sll, srl, sra)
  - Comparison and conditional branches
    - Always applied to registers and immediate values (eq, ne, ge, gt, le, lt)
  - Unconditional jump
    - call, jmp, br, ret
  - Other control instruction
    - trap, break, rdctl, wrctl, flush, sync, nop
  - You will be using C code to program the NiosII processor
    - If you want to use assembly code, it is fine
      - Use the asm construct in C

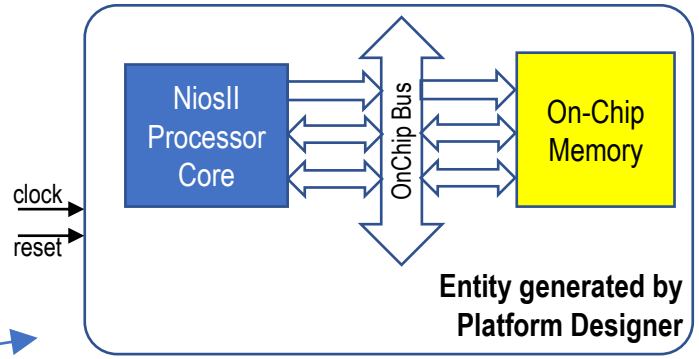
Altera Nios II Processor

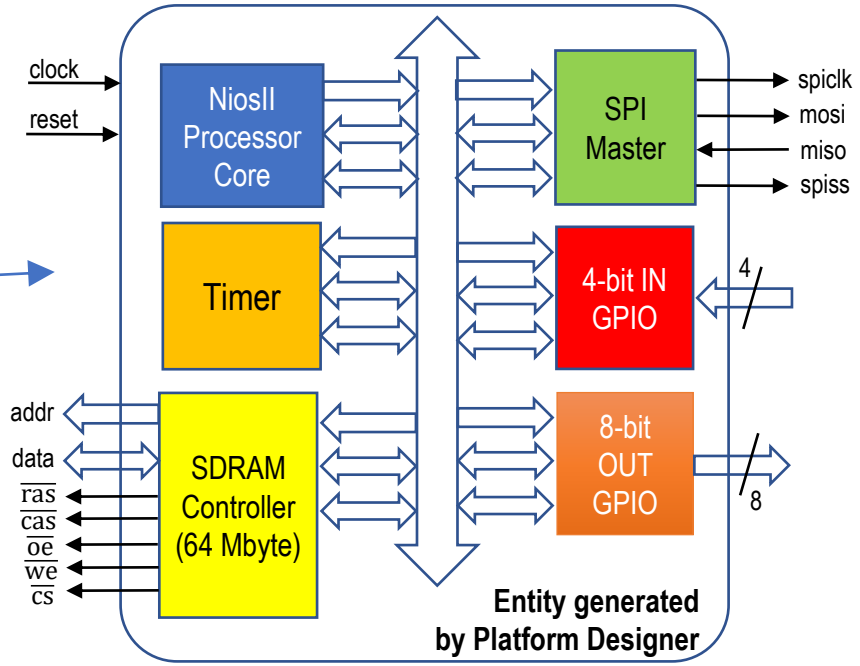
- Altera Platform Designer
  - Launched within Quartus Prime
    - Older versions with similar interfaces
      - SOPC Builder (System On Programmable Chip), QSys
  - Generates a VHDL entity containing the processor
    - It can be the only entity in your design
    - It can be instantiated in a larger VHDL design
  - Allows you to graphically configure the Altera Nios II processor
  - Allows you to add other blocks to the system on the FPGA
    - On-chip memory (program and data)
    - On-chip peripherals and coprocessors
    - On-chip bus for internal communications
    - Automatically instantiated, configurable

Altera Nios II Processor

- Altera Nios II Examples
  - Processor core alone
    - Clock, reset, address, data, control, irq
    - Not used most of the times
  - Processor + bus + memory
    - Clock, reset
    - Minimal working system
      - But usually you need something more for input and output
  - Processor + bus + memory + peripherals
    - Clock, reset, peripheral pins (e.g., SDRAM memory controller, GPIO, timer, SPI master, …)
  - Processor + bus + memory controller + timer + GPIO
    - Laboratory session to read UART line
    - See next slide

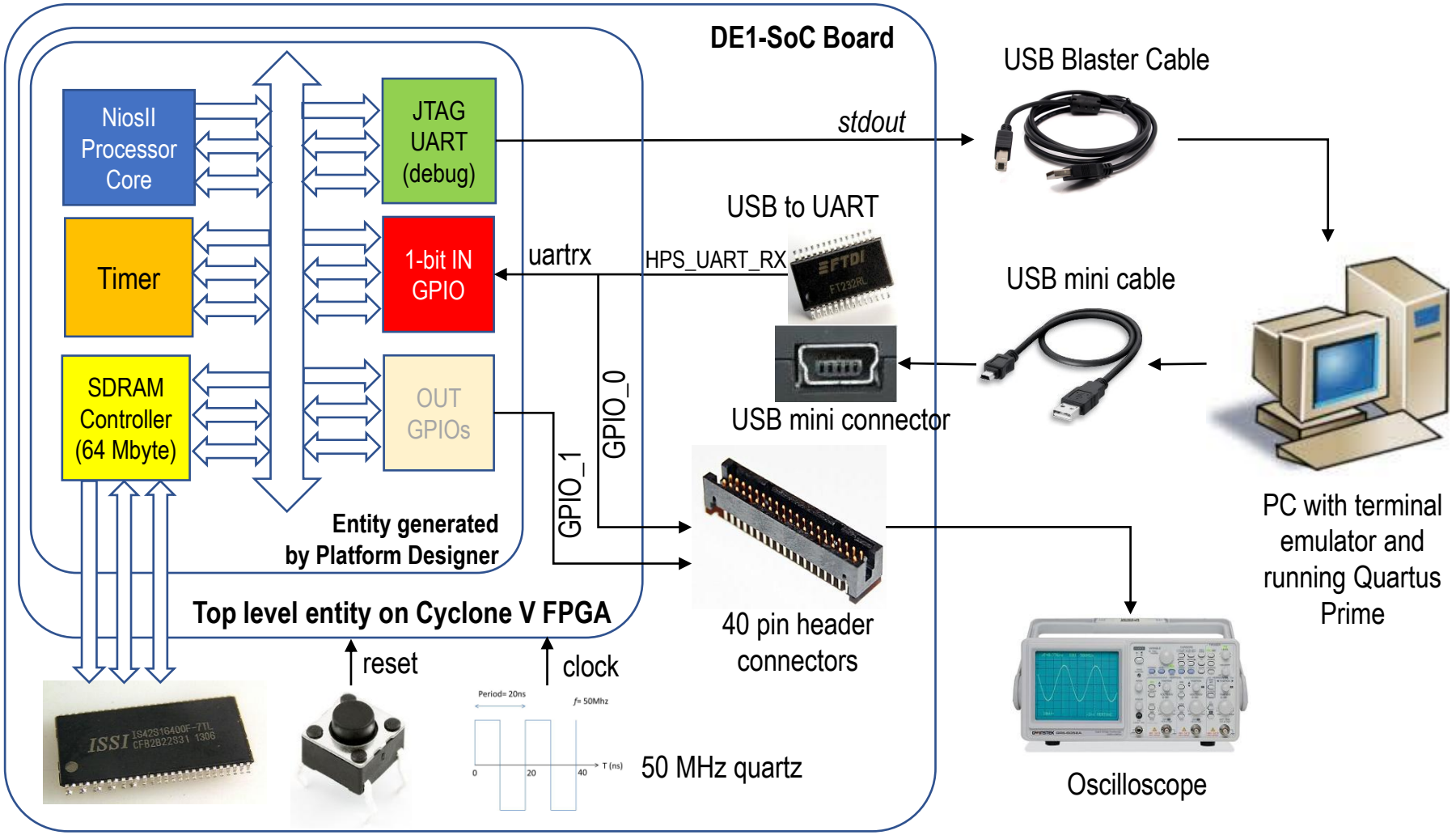






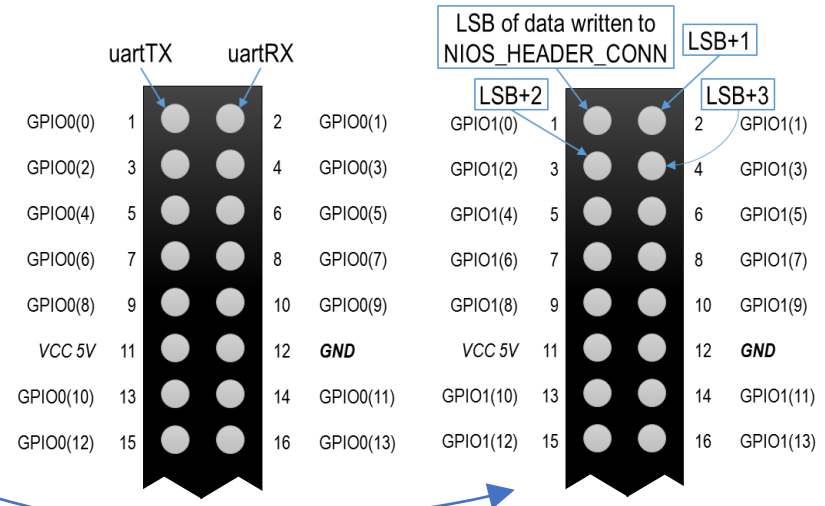
Altera Nios II Processor

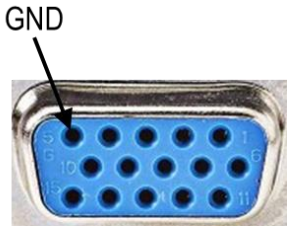
- Altera Nios II Example for Software UART laboratory session



Altera Nios II Processor

- Connections on the DE1-SoC
  - USB type A Host-to-Device cable for JTAG
    - Programming the hardware
    - Downloading the software
    - Implement stdin and stdout over JTAG
  - USB mini cable for UART connection
    - Using the FDTI232R integrated circuit
  - Female/Male jumpers for the oscilloscope
    - To display UART RX and TX signals
      - On the left header connector
    - To display software controlled pins
      - On the right header connector
  - Power Supply





Altera Nios II Processor

- Altera Nios II Synthesis
  - Platform Designer generates a VHDL description of an entity
    - You can instantiate the entity in your VHDL design
      - It may be your top level entity
    - It can be part of a higher level entity
      - As a component using port map
  - Quartus Prime synthesizes the entity to the FPGA
    - Pin assignments
    - Timing constraints
    - Simulation is hard and slow
      - Need a full license for /s and /f
    - Device configuration
  - What happens now?

Altera Nios II Processor

- Altera Nios II Software development
  - Integrated Development Environment (IDE)
    - Based on Eclipse
    - Already contains many libraries (Board Support Package, BSP)
    - Write C code
      - Start from `main()` function
      - Standard C libraries
        - With some limitations
      - Debugging through JTAG interface
      - Don’t use floating point arithmetic
    - Library handles
      - Processor initialization
      - Cache initialization (if present)
      - Peripheral initialization

Altera Nios II Processor

- Altera Nios II Software development
  - Board Support Package (BSP)
    - Based on the processor configuration
      - Processor boot at power-up (C Run-Time)
        - Assembler file crt0.s
          - Initialize Instruction and Data caches
          - Setup Global and Stack Pointers
          - Clear BSS segment (uninitialized global and static variables)
          - Call alt_main()
        - C file alt_main.c
          - Initialize interrupts
          - Initialize file descriptors
          - Initialize device drivers
          - Call main()

Altera Nios II Processor

- Altera Nios II Software development
  - Board Support Package (BSP)
    - Based on the processor configuration
      - Initialize device drivers
        - Configure device
        - Setup interrupt service routine
    - Provides header files and libraries to be included in your software
      - Macros to access devices
      - Functions to operate on devices
    - Any change in the hardware configuration requires to
      - Regenerate the BSP
      - Recompile the BSP
      - Recompile the application software that uses the BSP

Software UART Laboratory session

- Small Nios II system
  - Processor core
    - /e or any other
    - Get different performance
  - SDRAM Controller
    - External 64 MByte SDRAM
  - 1-bit input GPIO (General Purpose Input Output) for UART RX
    - Macro to read pin
  - Timer to measure time
    - Library functions to measure time
  - JTAG UART to send output to IDE
    - fprintf(stdout, "…");
- But there is a problem with the Cyclone V and the DE1-SoC board…

Software UART Laboratory session

- Large Nios II system
  - The real system is much more complex
  - The Cyclone V FPGA contains a dual core ARM processor
  - The UART lines (RX and TX) on the DE1-SoC are connected to the ARM processor
    - Not to the FPGA fabric
  - To make the UART lines (RX and TX) available to the FPGA
    - Need to customize the ARM processor in Platform Designer
      - Configure it to route those signals to the FPGA fabric
      - Generate a preloader for the ARM processor
      - Store it in a MicroSD Card
    - Insert the MicroSD Card in the reader of the DE1-SoC board before switching it on
    - When switched on
      - The ARM processor boots and configures the UART lines to the FPGA fabric
      - You can now program the FPGA with the Nios II processor

Software UART Laboratory session

- Large Nios II system
  - The whole process of creating the system is pretty long
    - Cannot be done in the 3 hours of the laboratory session
  - I provide you a full project that includes
    - The Platform Designer system
      - ARM Processor configuration
      - Nios II Processor configuration
      - A number of peripherals
        - More than those listed before
        - It can be used for other purposes
    - A top-level VHDL entity (uarthwsw.vhd)
      - It instantiates the Platform Designer system
      - It make all the needed connections
      - It is already compiled and verified (compilation alone takes 10 minutes on my laptop)

Software UART Laboratory session

- Large Nios II system
  - Examine the code for the top-level entity
    - GPIO_0(1) shows the HPS_UART_RX signal if you want to see it on the oscilloscope
    - GPIO_1(0) can be written by the Nios II processor (can be shown on the oscilloscope)
    - You can write the HPS_UART_TX signal from the Nios II processor
      - It will also appear on GPIO_0(0) if you want to see it on the oscilloscope
    - You can use the LEDs
    - You can use the 7 segment displays
      - The 7 segment display encoder is also already designed
    - Don’t worry about the I2C signals
      - They are not used in this lab
      - Unless you want to experiment
        - They are connected to an accelerometer with I2C slave interface

Software UART Laboratory session

- Large Nios II system
  - Examine the code for the top-level entity
    - The Nios II processor includes a hardware UART peripheral
      - Not used in this laboratory session
      - It will be used in the next laboratory session
    - The Nios II processor includes a hardware I2C peripheral
      - Not used in this laboratory session
      - It will NOT be used in the next laboratory session either
        - Unless you want to experiment
    - The Nios II processor includes a hardware SPI peripheral
      - Not used in this laboratory session
      - It will NOT be used in the next laboratory session either
        - Unless you want to experiment

Software UART Laboratory session

- Large Nios II system
  - Examine the code for the top-level entity
    - The system includes a VGA oscilloscope
      - It draws the UART lines over time on a VGA monitor
      - Useful to perform the laboratory session at home
      - Separate instructions on how to use it
        - It is not really straightforward
        - The real oscilloscope in the laboratory is somewhat easier
  - The same hardware project will be used for the 3rd laboratory session
    - UART using a processor peripheral
    - Select the right components using switch SW(9)
      - SW(9) = 0 → 2nd laboratory session (this one)
      - SW(9) = 1 → 3rd laboratory session (next time)
  - <u>Keep SW(9) = 0 for this laboratory session</u>

Software UART Laboratory session

- Reading a GPIO pin
  - NIOS_UARTRX is the name given to the PIO in Platform Designer
  - val is a 32-bit integer variable, but the PIO is a single bit
    - The most significant 31 bits may contain random data
    - Need to mask unwanted bits using logical operators
  - More detailed explanation when describing processor peripherals

```
int val;
/* Read data from pin */
val = IORD_ALTERA_AVALON_PIO_DATA(NIOS_UARTRX_BASE);
/* Keep only least significant bit (mask the others) */
val = val & 0x01;
```

- Writing is similar
  - Use it if you want to write to the HPS_UART_TX
  - Use it if you want to write pins of the header connector

```
int val = 0x01;
/* Write data to pins */
IOWR_ALTERA_AVALON_PIO_DATA(NIOS_UARTTX_BASE, val);
val = 0x05;
/* Write data to header connector pins */
IOWR_ALTERA_AVALON_PIO_DATA(NIOS_HEADER_CONN_BASE, val);
```

Software UART Laboratory session

- Using the timer
  - It is a counter that counts clock ticks
  - alt_timestamp_freq() returns the number of ticks per second
    - Using a 50 MHz clock it returns 50.000.000
  - Start the timer
    - (void) alt_timestamp_start();
    - Clears the counter
  - Read the timer
    - int t1 = alt_timestamp();
    - Reads the counter value
  - Examples
    - Measure elapsed time
    - Wait for a certain time

```
int ticksPerSec = alt_timestamp_freq();
int t1 = alt_timestamp();
/* Some code here to be measured */
int t2 = alt_timestamp();
int nticks = t2 – t1;
printf("Number of ticks: %d\n", nticks);
printf("Number of us: %d\n", (nticks * 1000000) / ticksPerSec);
```

```
/* Wait n microseconds */
int ticksPerSec = alt_timestamp_freq();
int n = 8;
int nticks = (ticksPerSec / 1000000) * n;
alt_timestamp_start();
while (alt_timestamp() < nticks) {  /* Do nothing */  }
```

Software UART Laboratory session

- Controlling the LEDs
  - Write a value to the NIOS_LEDS GPIO included in the system
  - There are only 10 LEDs, so the variable should not exceed the value 0x3FF (1023_{10})
    - Higher significant bits are simply ignored

```
int val = 0x15A;   /* Maximum 10 bits */
/* Write data to the LED pins */
IOWR_ALTERA_AVALON_PIO_DATA(NIOS_LEDS_BASE, val);
```

- Controlling the 7-segment displays
  - Write a value to the NIOS_7SEG GPIO included in the system
  - Each 7-segment display shows an hexadecimal character of 4 bits (0 to F)
  - The variable should not exceed the value 0xFFFFFF (16777215_{10})
    - Higher significant bits are simply ignored

```
int val = 0x2B73C8;   /* Maximum 24 bits */
/* Write data to the 7-seg display pins */
IOWR_ALTERA_AVALON_PIO_DATA(NIOS_7SEG_BASE, val);
```

Software UART Laboratory session

- What should the software do?
  - Implement the UART RX protocol by reading the GPIO pin
    - While pin is high (UART idle) do nothing
    - When pin becomes low, start timer
    - After half bit, sample START BIT (check that it is still low)
      - Restart timer (or compute subsequent sampling instants)
    - Loop n times (where n is the number of bits of a character):
      - After one bit time, sample bit (in the middle), restart timer
    - After one bit time, sample parity or STOP BITs
    - Return to the beginning
  - You may make the software configurable (e.g., using #define) for
    - Baud Rate
    - Number of bits in the character
    - Presence and kind of parity
    - Number of STOP bits

Software UART Laboratory session

- Beware of integer arithmetic
  - Avoid underflowing a division
    - Example: $1/2 * 10 = 0$
    - Use parenthesis
      - First multiply at the numerator, then divide
      - Example: $(1 * 10)/2 = 5$
  - Avoid overflowing the timer ticks
    - They are on 32 bits, and roll back to 0 when the end is reached
- Bits arrive LSB first
  - Use bit masking to store them in the correct position
    - Use shifts and logic operations
- Baud rate may not be exact
  - Quartz tolerance
  - Integer division and software delays
  - Try to compute your real tolerance

Software UART Laboratory session

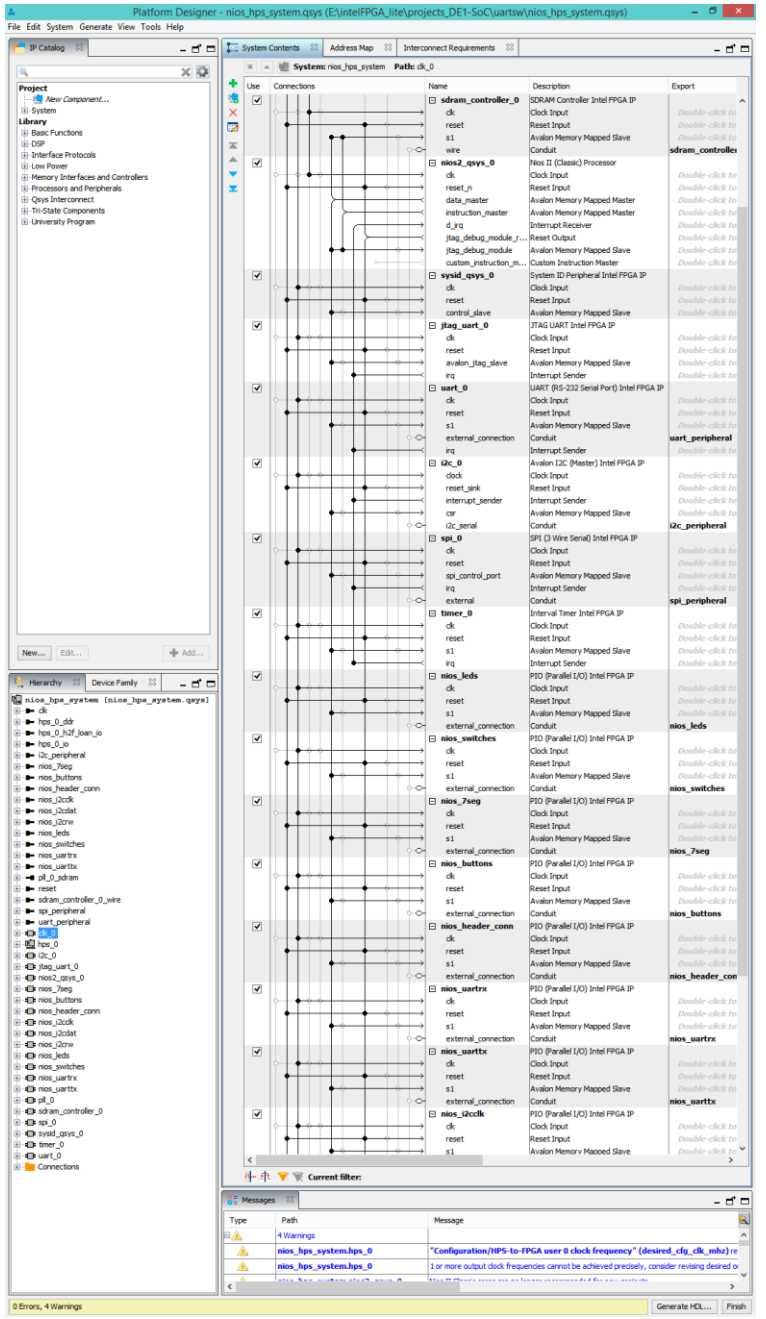
- How fast can you go without errors?
  - Try different Baud Rates
    - Start with 300, which should be easy to handle
    - Then increase it till you get wrong results
      - Need to change it in the SW for the NiosII and in the terminal emulator program on the PC
      - Need to recompile the SW when you change a #define
  - Look for optimizations
    - Precompute values (if possible) before entering a loop
    - Avoid multiplications if you can use additions
    - Division is implemented in a SW library, not in HW!
    - Loop unrolling increases performance
      - Avoids incrementing a loop variable
      - Avoids testing and jumping to the beginning of the loop

Software UART Laboratory session

- Step by Step instructions
  - Copy the provided project on your laptop or on the desktop PC of the lab
  - Open the uarthwsw/uarthwsw.qpf **project**
  - Open the uarthwsw/uarthwsw.vhd **file of the project**
    - This is the top-level entity
    - Look at the entity ports
      - There are many, some are commented out
    - Look at the components and their instances
      - The Platform Designer system u0
      - The Seven-segment display drivers hd*i*
      - The VGA oscilloscope v0
    - Look at the signal assignments

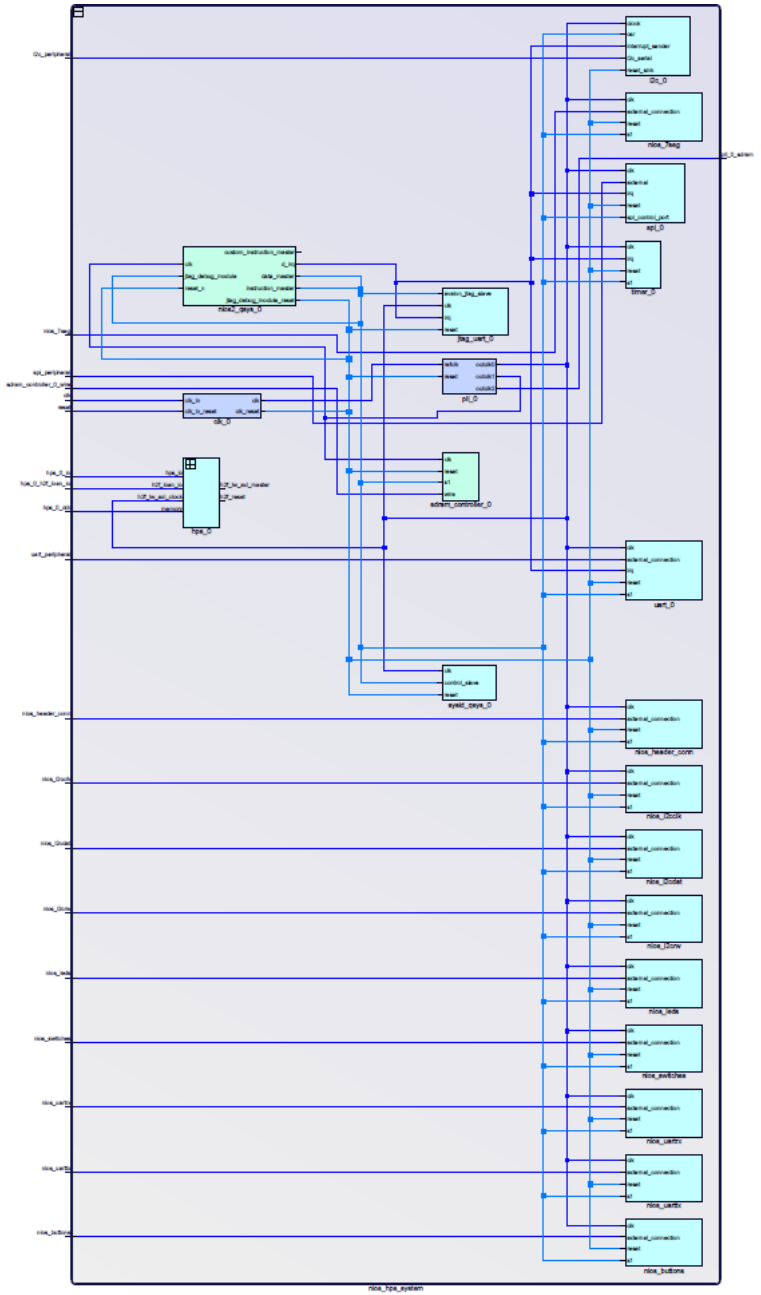
Software UART Laboratory session

- Step by Step instructions
  - Open the Platform Designer (in the Tools menu)
    - Load the nios_hps_system.qsys **system**
    - Look at the various components
      - nios2_qsys_0: Nios II processor
      - nios_uartrx: GPIO to read the UART_RX line
      - …
    - Look at their interconnections
      - Clocks and reset
      - Data and Instruction Masters
      - IRQ lines
      - Conduit
        - These are the external connections of the Platform Designer system
    - If you double click on a component you can see its parameters



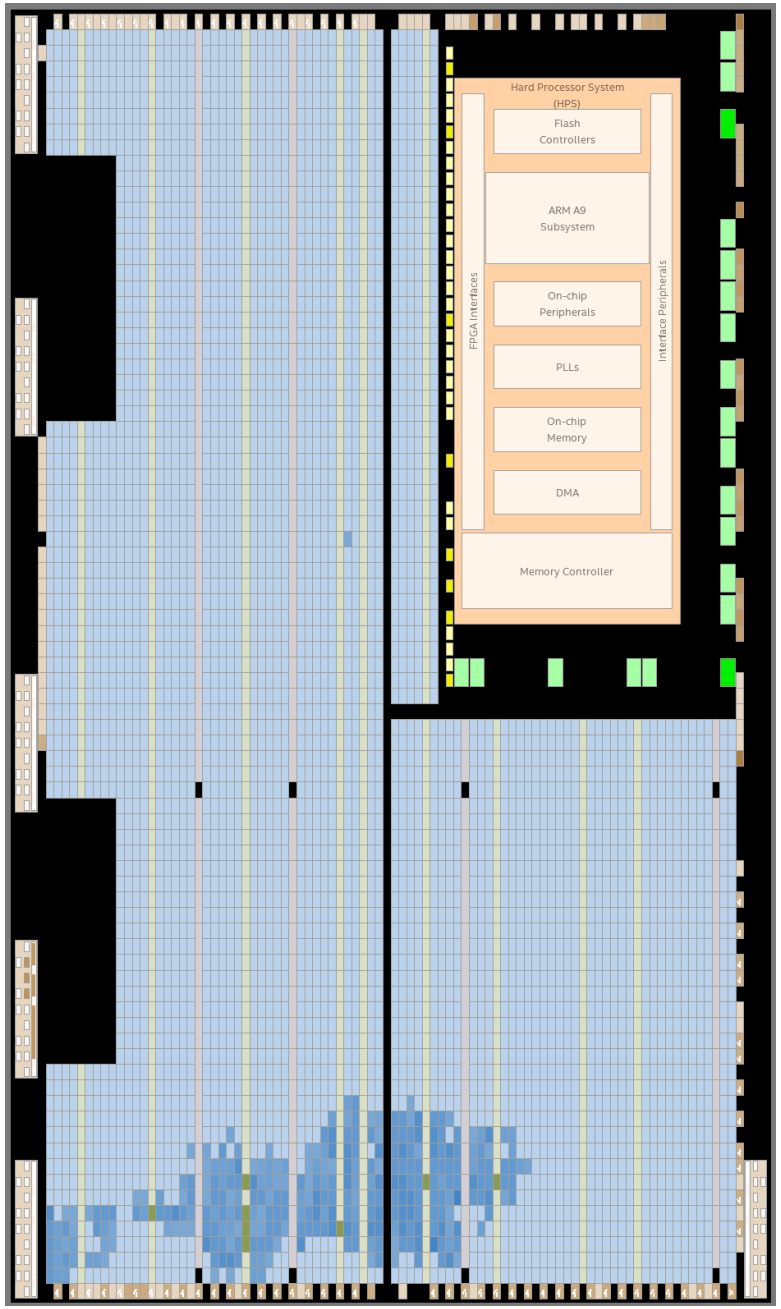
Software UART Laboratory session

- Step by Step instructions
  - Still in Platform Designer
    - Select View → Schematic
    - You see the same system as a block diagram
    - If you hover the mouse over a block
      - You see a popup list of all interconnections
    - If you hover the mouse over an interconnections
      - You see a popup list of all connected blocks
        - The master is driving the connection
    - In the Hierarchy tab on the left
      - List of all blocks and interconnections
      - Details of all wires and their directions
        - You must expand blocks



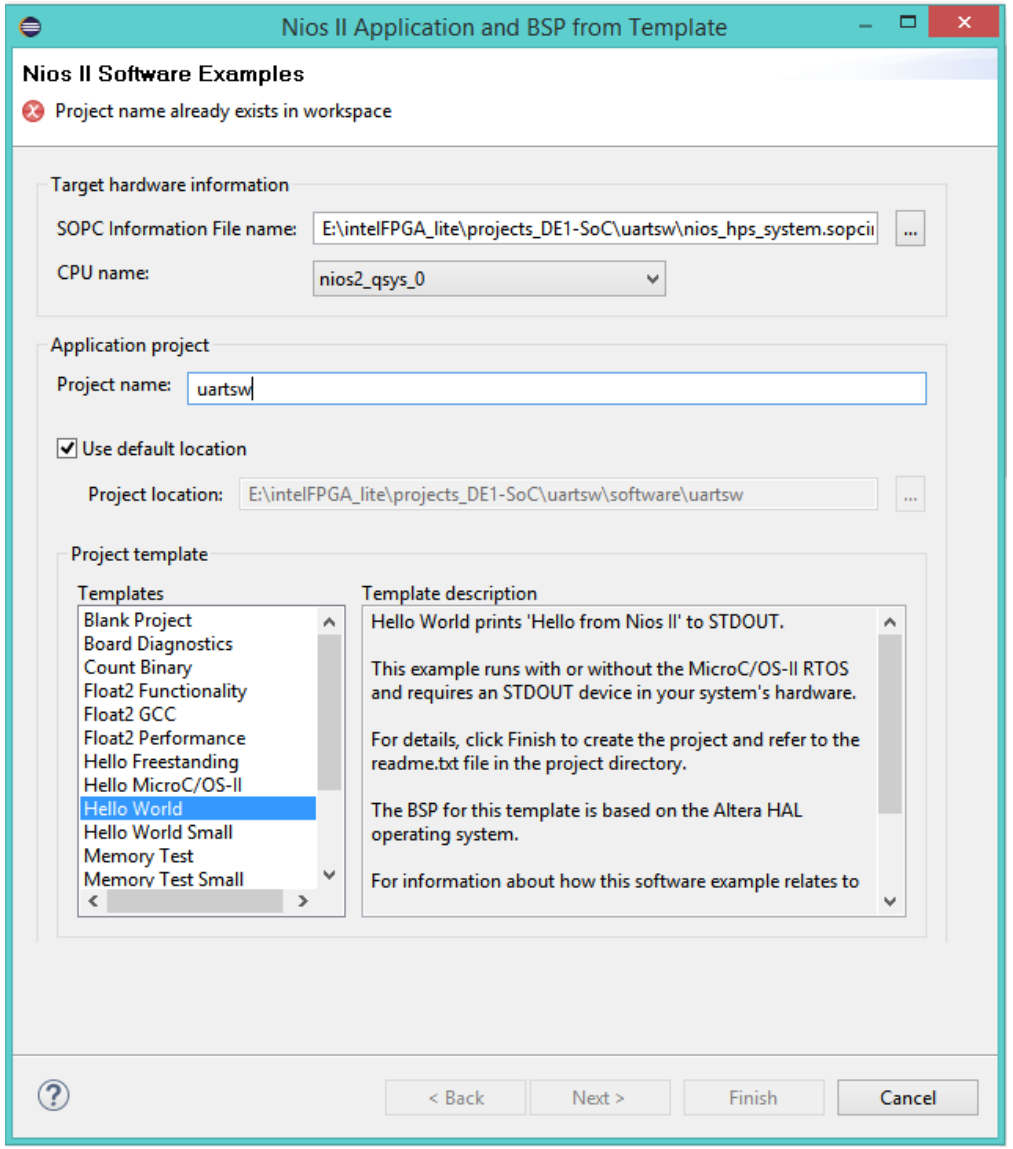
Software UART Laboratory session

- Step by Step instructions
  - Open the Chip Planner (under Tools menu in Quartus Prime)
    - The ARM processor is in the top right
    - The Nios II and all peripherals are at the bottom
    - Most of the FPGA is unused
      - The selected Nios II Processor is very simple
        - You can change it in Platform Designer
        - Then regenerate the system
        - Instantiate it in the top-level entity
        - Recompile the entire design
        - DON'T DO IT in the lab, it takes a long time
  - Insert the MicroSD Card in the DE1-SoC card reader
  - Switch on the board
  - Program the board



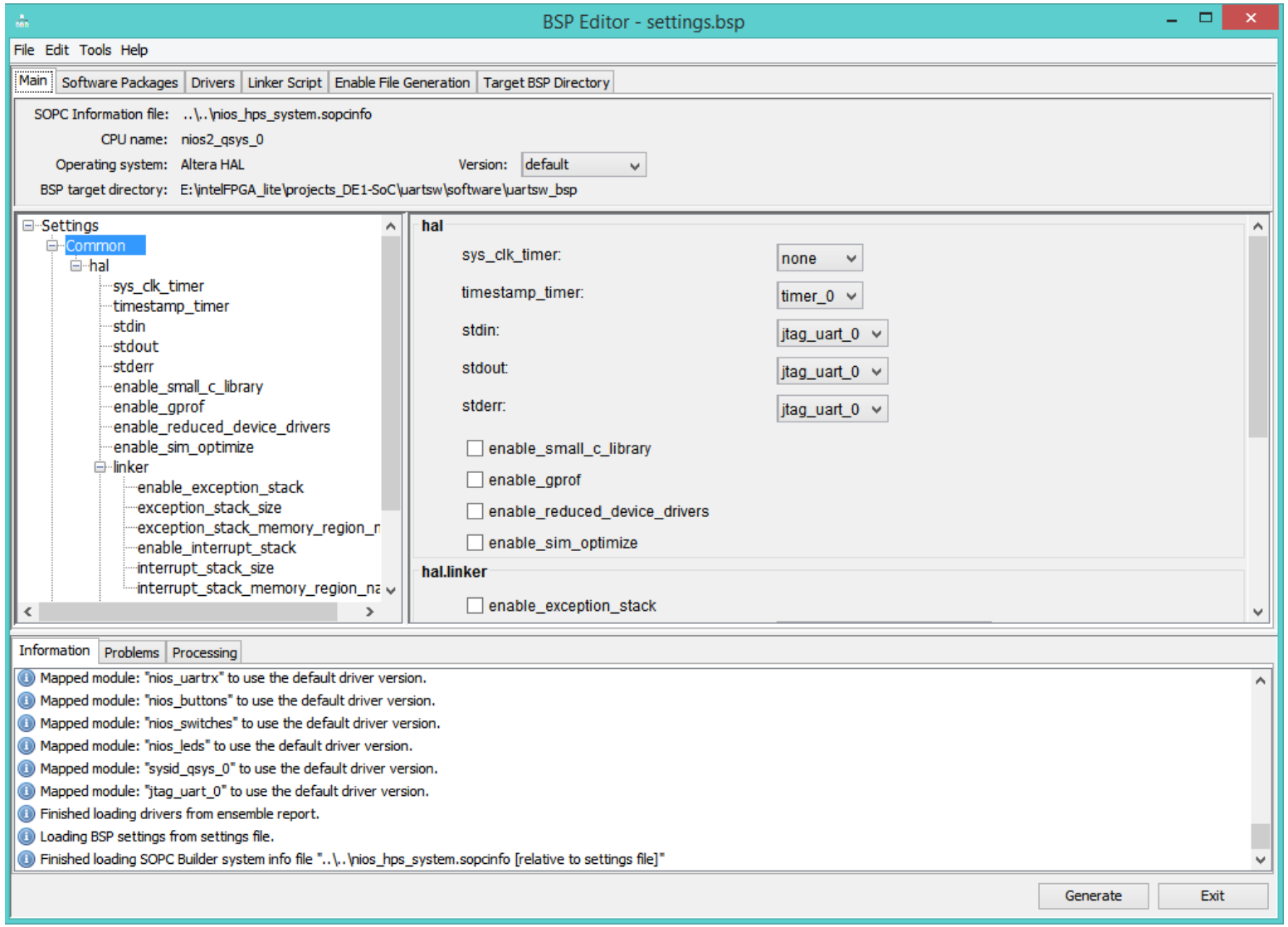
Software UART Laboratory session

- Step by Step instructions
  - Start the Nios II Software Build Tools for Eclipse
    - It is in the Tools menu in Quartus Prime
      - Or you can run it from the Windows taskbar
    - Choose a new directory for the workspace
    - Create it in the project directory
  - Create a new Nios II Application and BSP
    - File → New Nios II Application and BSP from template
    - Select the nios_hps_system.sopcinfo file in the Target hardware information
    - Give a name to the project
    - Choose a location
    - Choose the Hello World template
    - Also create a new BSP (after clicking Next)
      - With default parameters
    - Click Finish



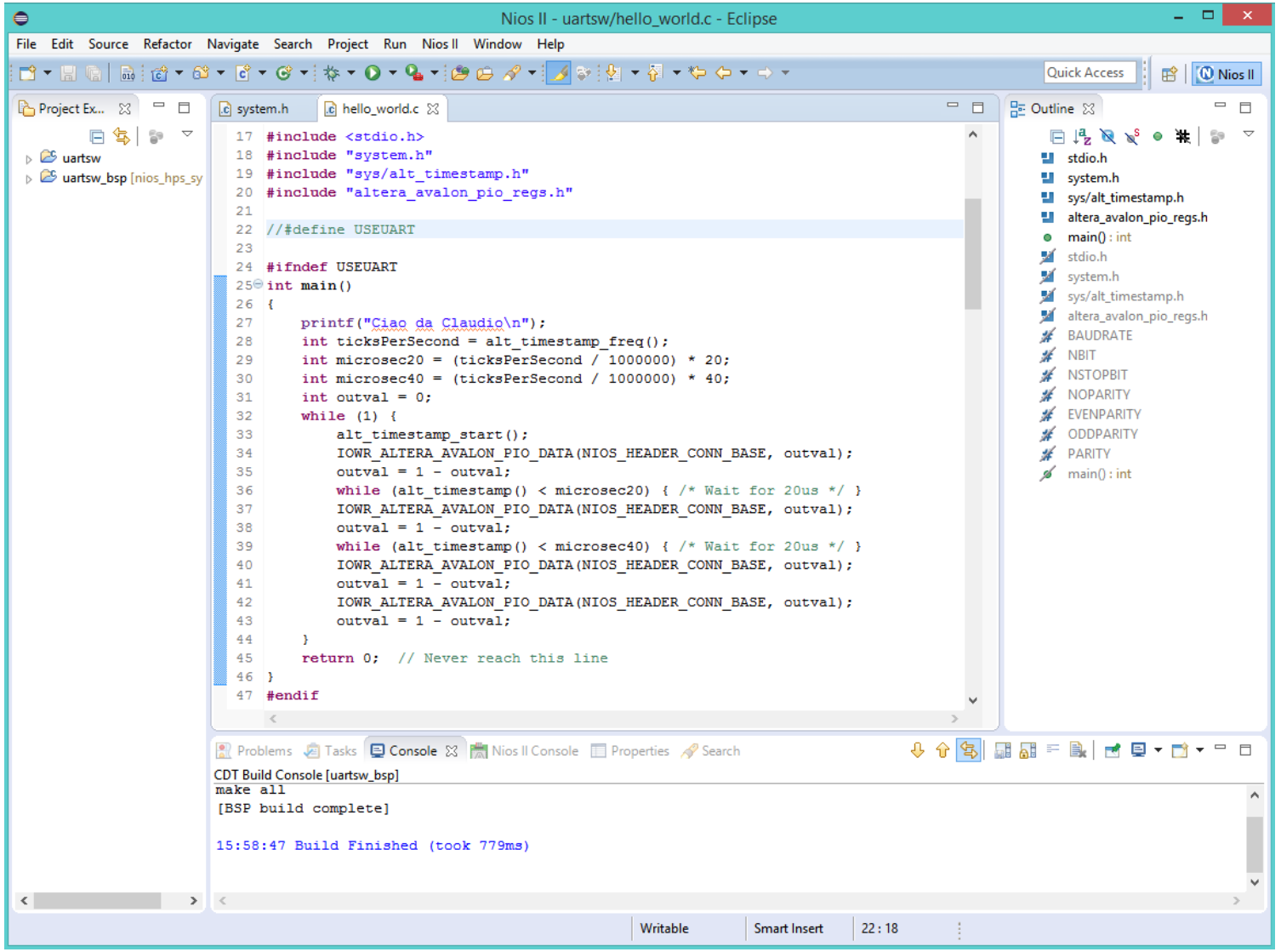
Software UART Laboratory session

- Step by Step instructions
  - Check the BSP configuration
    - Run it by right-clicking on the software project and choosing Nios II → BSP Editor…
    - Verify stdout
      - Should be the jtag_uart_0
    - Verify sys_clk_timer
      - Should be none
    - Verify timestamp_timer
      - Should be timer_0
    - If you make any change then click on Generate before exiting



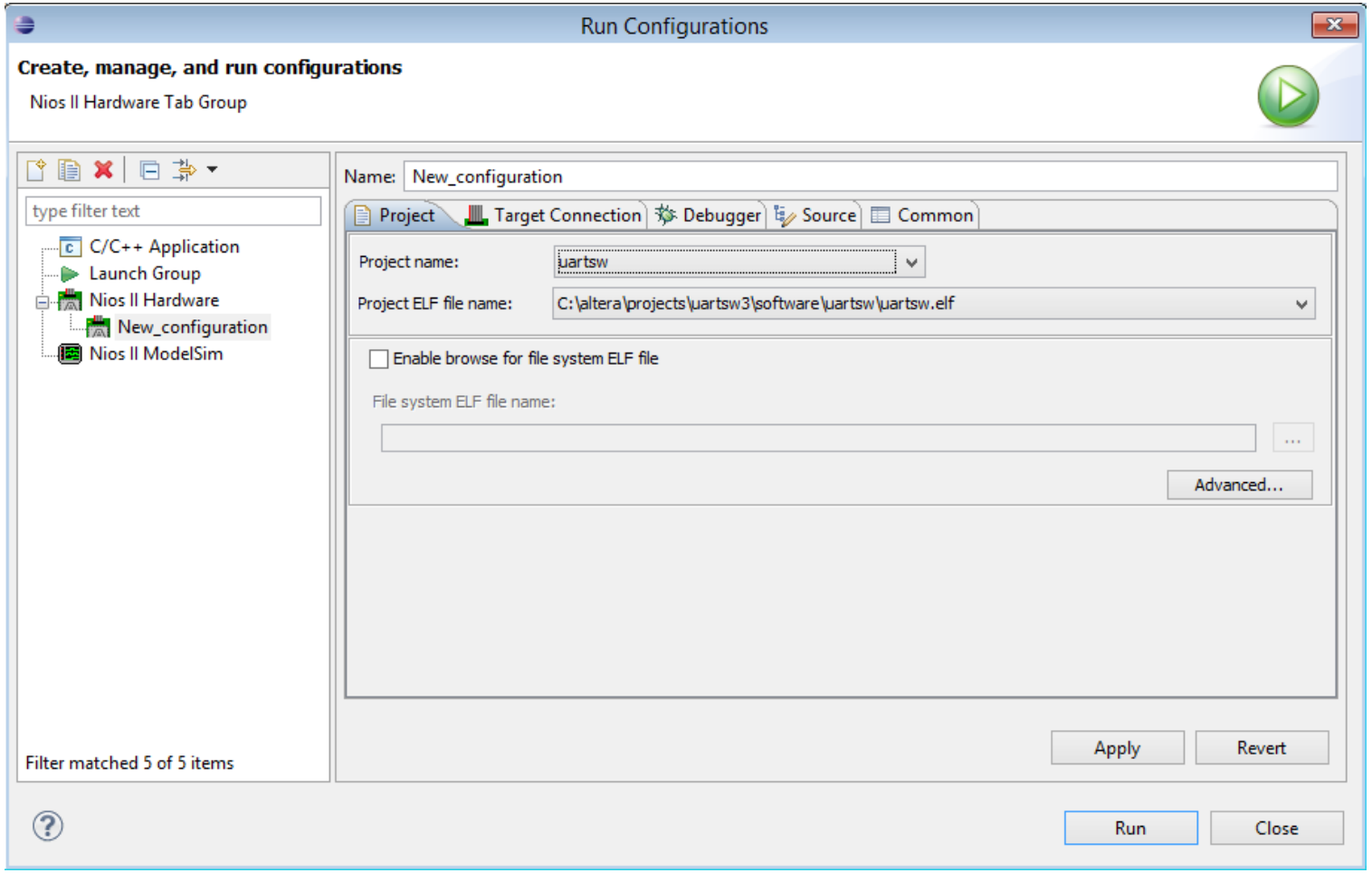
Software UART Laboratory session

- Step by Step instructions
  - Change the hello_world.c code
    - It is in the project folder
    - Begin with simple code
    - Then add further details
    - Code should never terminate
  - Compile the design and the BSP
    - Use Project → Build All
    - You should get no errors
    - Check the tab Problems



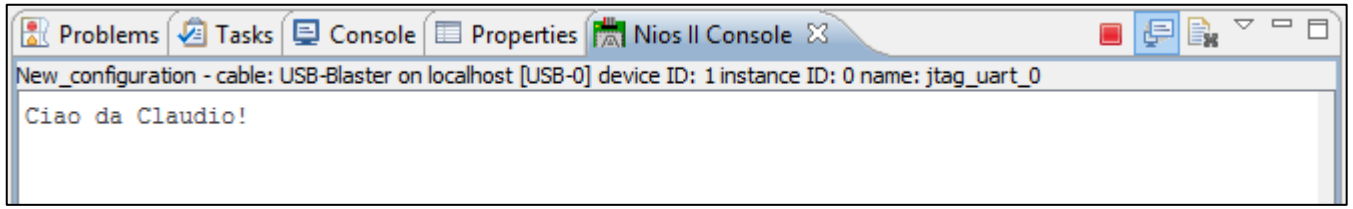
Software UART Laboratory session

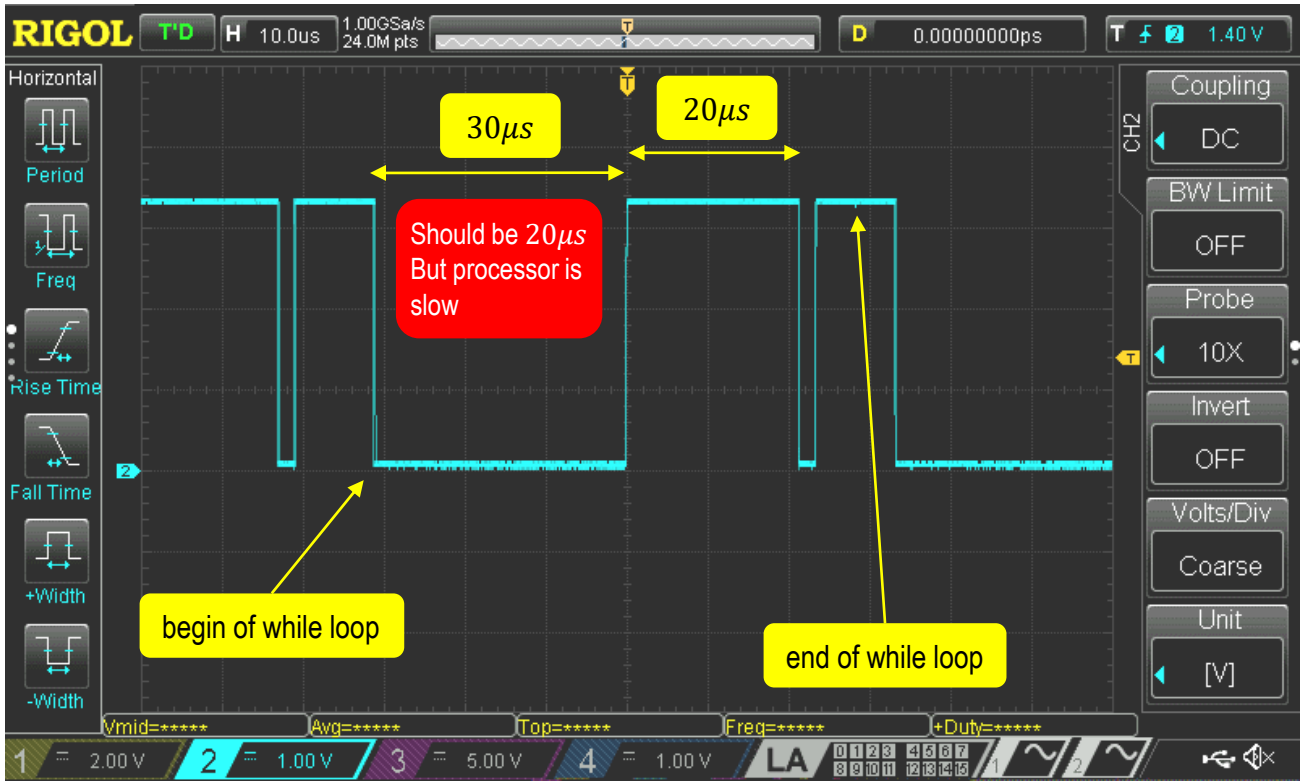
- Step by Step instructions
  - Create a Run Configuration
    - Provide the project name
    - Check the ELF name
    - Check the Target Connection
  - Run the software
    - Read output on the Nios II console
    - Connect an oscilloscope if you have any output on the header connector



Software UART Laboratory session

- Step by Step instructions
  - Example output on the Nios II console
  - Example output on the oscilloscope
    - Blue waveform is the GPIO1(0) pin
    - It is alternatively written with the values 1 and 0 in the software
    - One square is $10\mu s$
    - One clock cycle is 20 ns
    - Even using the timer, some operations cause significant delays





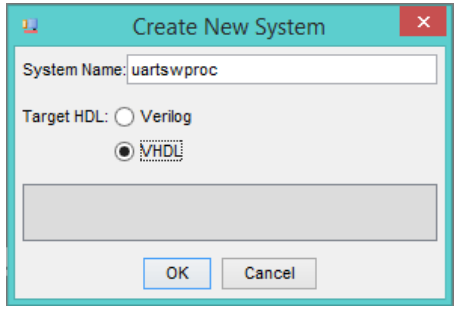
Software UART Laboratory session

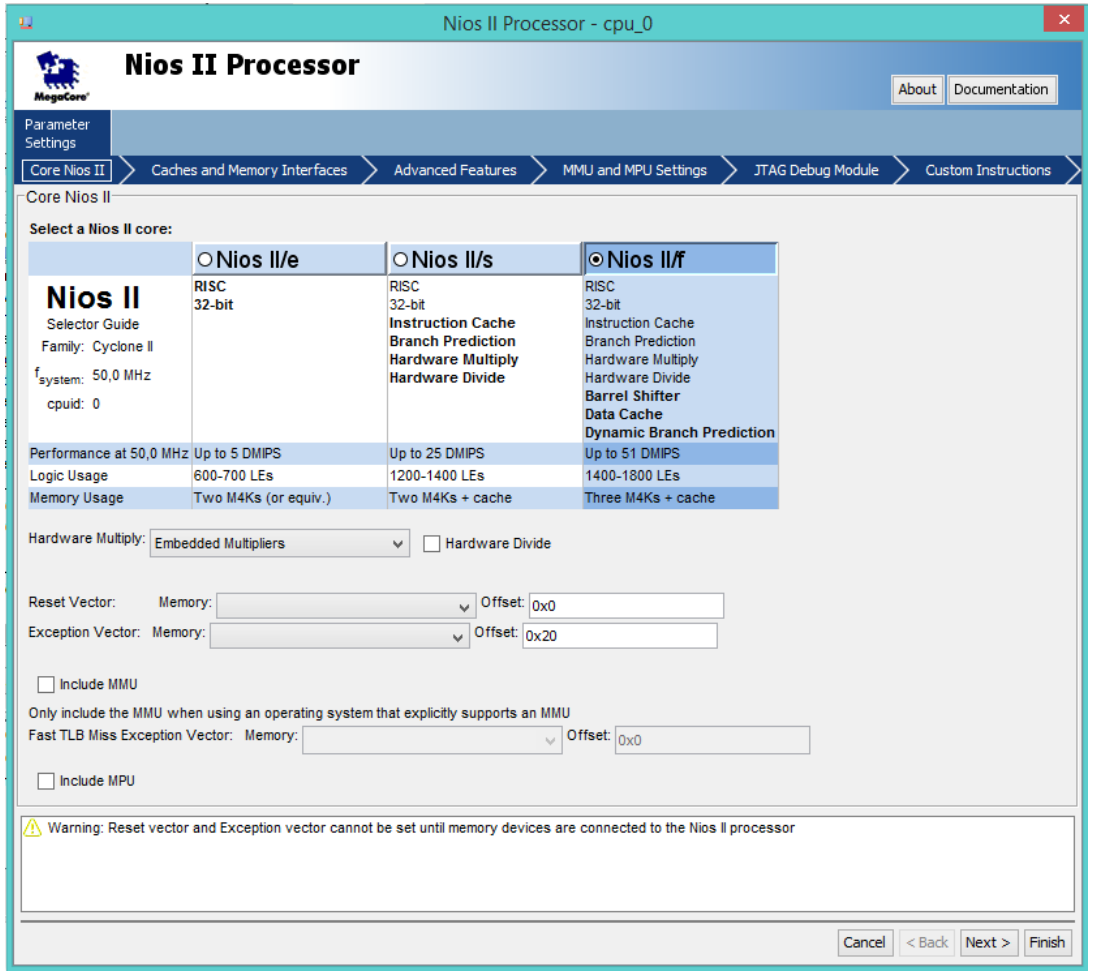
- Old hardware design instructions
  - Before 2020, with the DE2 board
  - Not to be used with the DE1-SoC board
    - To use UART with the Nios II processor the HPS (ARM9 processor) should also be included in the design
    - Can be used with the DE1-SoC if you don't need to access UART but just want to have a Nios II processor
  - Can be used with a Cyclone 10 LP
  - Can be used with other boards (DE0, DE10, …)
- All pictures are for Quartus 11.0sp1 and SOPC Builder
  - But Platform Designer is not so different from SOPC Builder

Software UART Laboratory session

- Old instructions for Cyclone II using SOPC Builder
- Don't use for Cyclone V on DE1-SoC
- May be used for Cyclone 10 LP

- Step by Step instructions
  - Create a new project
    - Top level entity uartsw
  - Launch SOPC Builder
    - Top level entity uartswproc
    - VHDL language
  - Instantiate a Nios II processor
    - Choose kind of processor (/e, /s, /f)
    - Setup Caches and Memory Interfaces
    - Setup JTAG Debug Module
    - Setup multipliers and divider implementation options

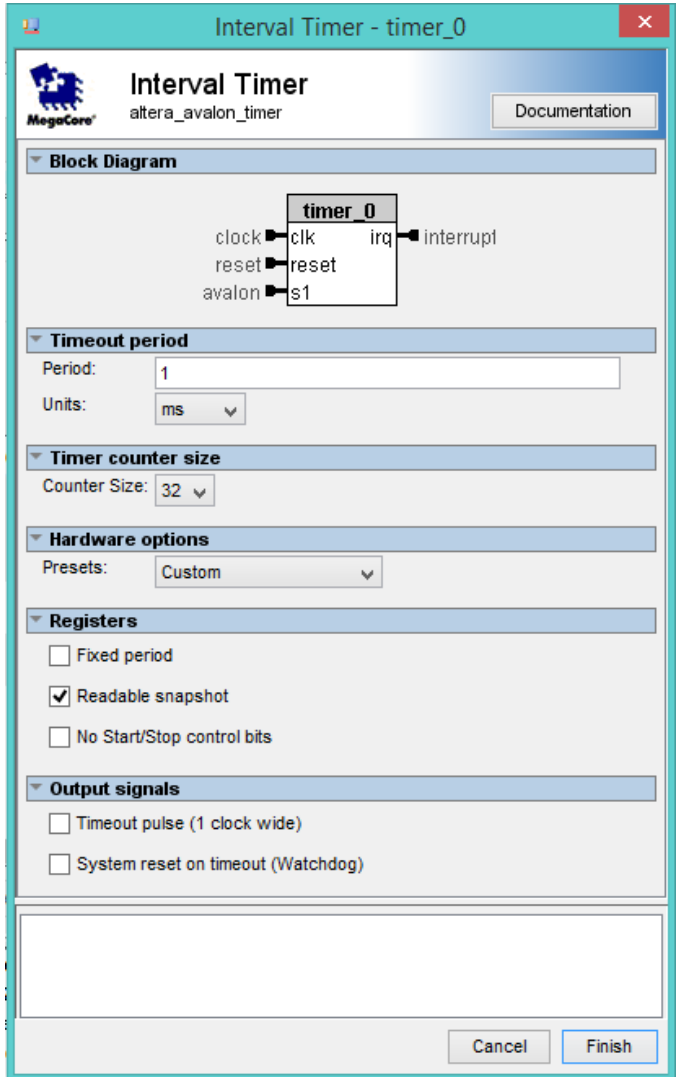


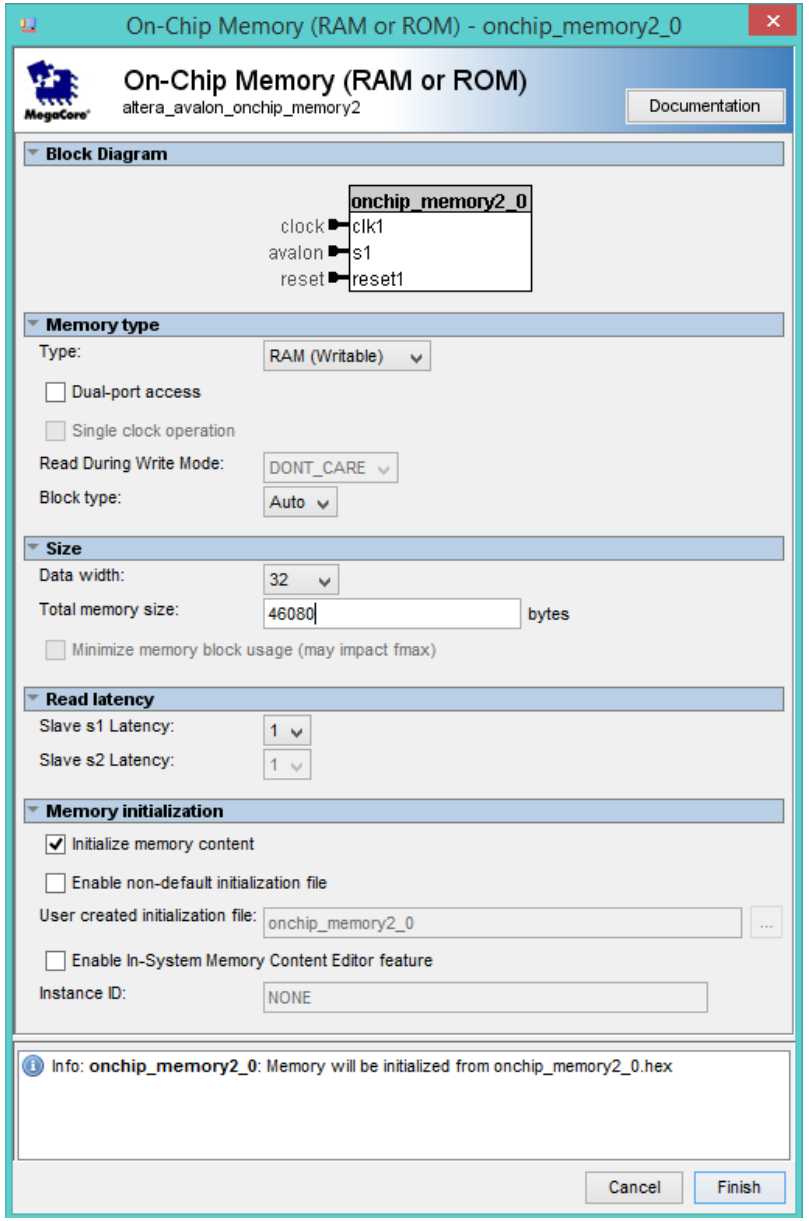


Software UART Laboratory session

- Step by Step instructions
  - Instantiate an On-Chip RAM Memory
    - Choose size (at least 40 Kbytes)
      - Max size depends on size of caches
      - They all use M4K blocks in the FPGA
  - Instantiate an Interval Timer
    - Choose period
    - Choose counter size
    - Enable reading a snapshot

- Old instructions for Cyclone II using SOPC Builder
- Don't use for Cyclone V on DE1-SoC
- May be used for Cyclone 10 LP

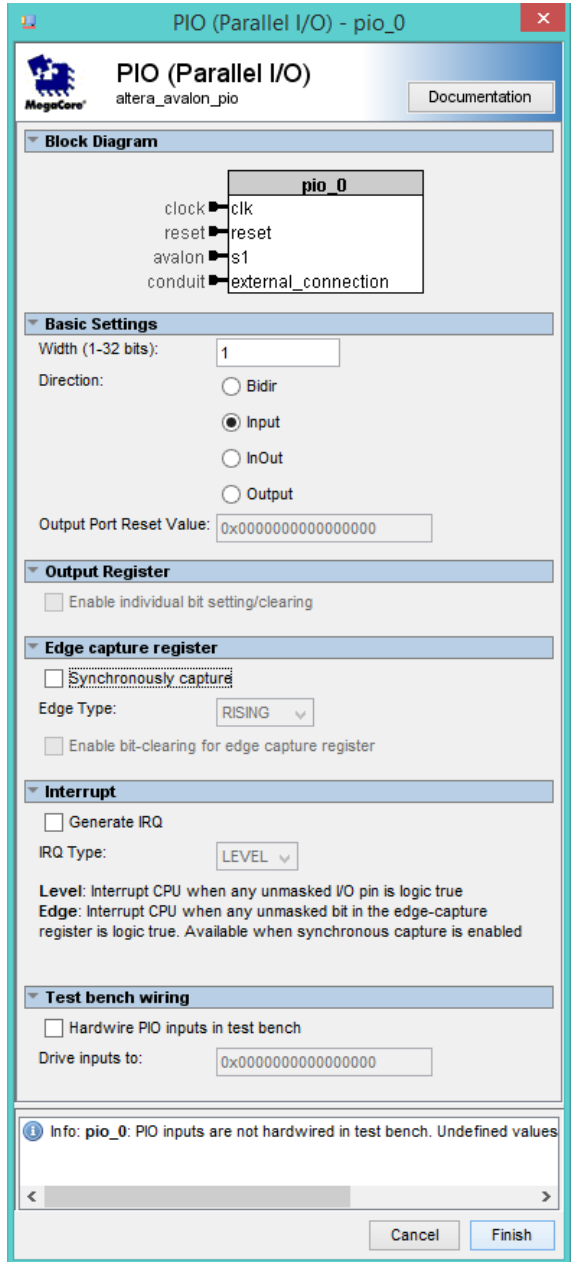




Software UART Laboratory session

- Step by Step instructions
  - Instantiate a Parallel I/O (PIO)
    - 1-bit input to read the UART line
      - Don't enable the edge capture register
      - Don't enable IRQ generation
        - Unless you want to use interrupts
          - You need the external SDRAM in this case
    - 1-bit output for the oscilloscope
      - Only if you plan to generate a waveform on an output pin on the header connector
      - Useful for debugging
    - Any other output PIO to control LEDs or 7-Seg Displays
      - Write values using software
  - Instantiate a SysID peripheral
    - Not strictly needed, but sometimes helps avoiding problems

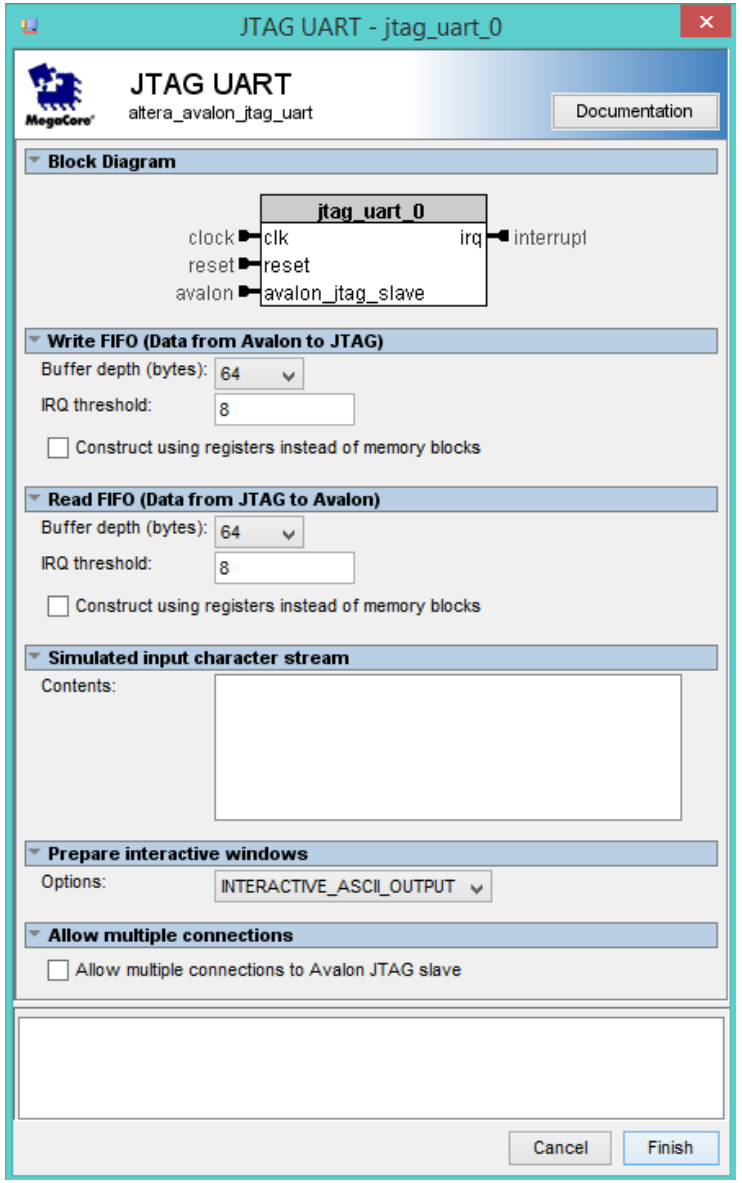
- Old instructions for Cyclone II using SOPC Builder
- Don't use for Cyclone V on DE1-SoC
- May be used for Cyclone 10 LP



Software UART Laboratory session

- Step by Step instructions
  - Instantiate a JTAG UART
    - Useful for debugging
    - Implements stdout over the Board-to-PC USB connections
    - Allows printf to show on the PC screen
    - This UART is NOT the one you must develop in software
    - JTAG pins are pre-assigned
      - You don't need to set them in the Pin Planner
  - Assign Base Addresses
  - Assign IRQ numbers
  - Assign Reset and Exception vectors
    - In the microprocessor configuration window

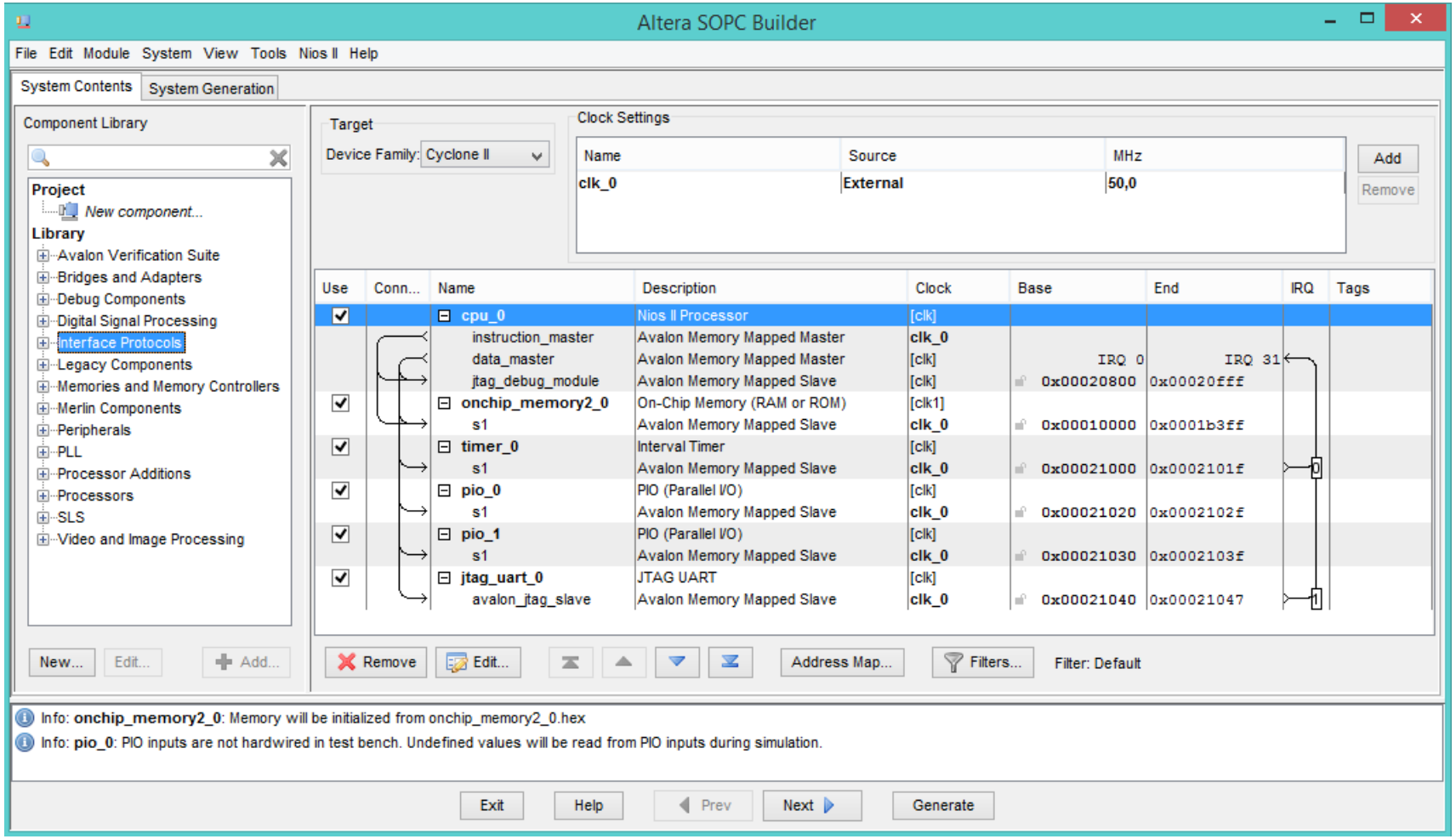
- Old instructions for Cyclone II using SOPC Builder
- Don't use for Cyclone V on DE1-SoC
- May be used for Cyclone 10 LP



Software UART Laboratory session

- Old instructions for Cyclone II using SOPC Builder
- Don't use for Cyclone V on DE1-SoC
- May be used for Cyclone 10 LP

- Step by Step instructions
  - Check final design
  - Save and generate VHDL code



Software UART Laboratory session

- Step by Step instructions
  - Create top level entity
    - Instantiate processor
    - Add any additional hardware

```
2344  -- altera message_off 10034 10035 10036 10037 10230 10240 10030
2345
2346  library altera;
2347  use altera.altera_europa_support_lib.all;
2348
2349  library ieee;
2350  use ieee.std_logic_1164.all;
2351  use ieee.std_logic_arith.all;
2352  use ieee.std_logic_unsigned.all;
2353
2354  entity uartswproc is
2355          port (
2356                -- 1) global signals:
2357                   signal clk_0 : IN STD_LOGIC;
2358                   signal reset_n : IN STD_LOGIC;
2359
2360                -- the_pio_0
2361                   signal in_port_to_the_pio_0 : IN STD_LOGIC;
2362
2363                -- the_pio_1
2364                   signal out_port_from_the_pio_1 : OUT STD_LOGIC
2365                );
2366  end entity uartswproc;
2367
2368
2369  architecture europa of uartswproc is
2370  component cpu_0_jtag_debug_module_arbitrator is
```

File uartswproc.vhd generated by SOPC Builder

- Old instructions for Cyclone II using SOPC Builder
- Don't use for Cyclone V on DE1-SoC
- May be used for Cyclone 10 LP

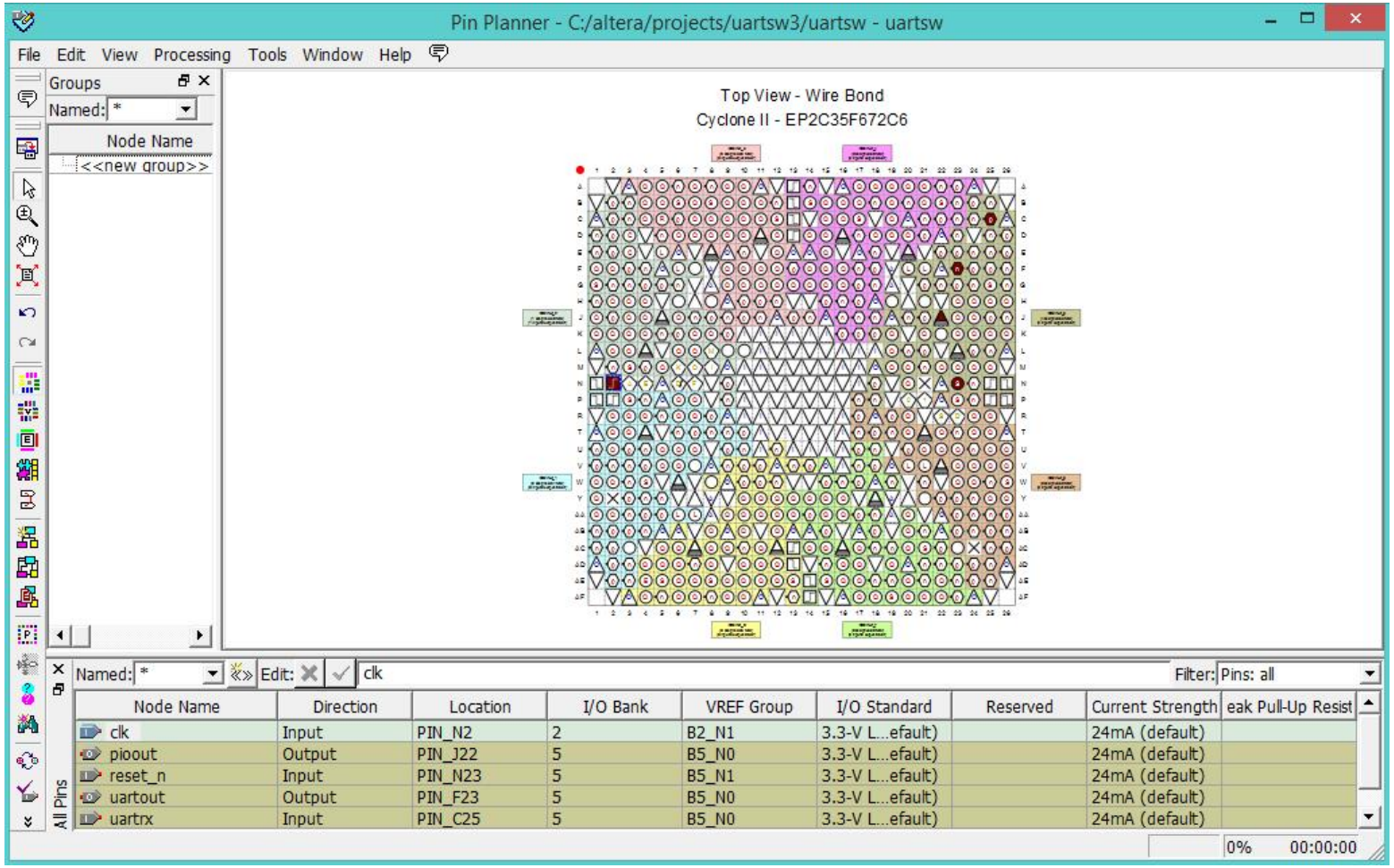
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_arith.all;
4   use ieee.std_logic_unsigned.all;
5
6   entity uartsw is
7      port (
8         signal clk : IN STD_LOGIC;
9         signal reset_n : IN STD_LOGIC;
10        signal uartrx : IN STD_LOGIC;
11        signal uartout: out std_logic;
12        signal pioout: out std_logic
13     );
14  end entity uartsw;
15
16  architecture struct of uartsw is
17
18     component uartswproc is
19        port (
20           -- 1) global signals:
21           signal clk_0 : IN STD_LOGIC;
22           signal reset_n : IN STD_LOGIC;
23           -- the_pio_0
24           signal in_port_to_the_pio_0 : IN STD_LOGIC;
25           -- the_pio_1
26           signal out_port_from_the_pio_1 : OUT STD_LOGIC
27        );
28     end component uartswproc;
29
30  begin
31
32     myproc: uartswproc port map (clk, reset_n, uartrx, pioout);
33     uartout <= uartrx;
34
35  end architecture struct;
```

File uartsw.vhd written by you

Software UART Laboratory session

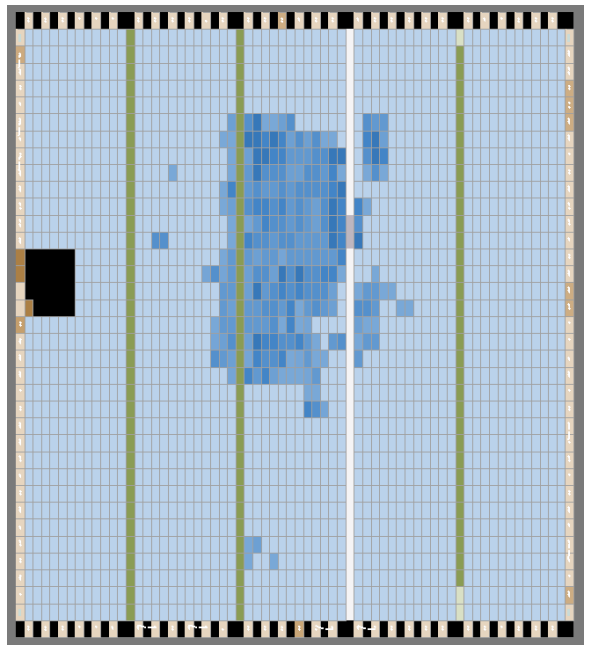
- Old instructions for Cyclone II using SOPC Builder
- Don't use for Cyclone V on DE1-SoC
- May be used for Cyclone 10 LP

- Step by Step instructions
  - Assign pins
    - Clock
    - Reset
      - To a push button
    - UART RX line
    - Any other pin that you decided to use
      - Header connector
      - LEDs
      - 7-Seg Displays
      - Switches
  - Add timing constraints
    - Clock @ 20ns



Software UART Laboratory session

- Step by Step instructions
  - Compile design
    - Check reports
    - Check Chip Planner
    - Check Timing Analysis
  - Program the board



Flow Summary

Flow Status	Successful - Thu Nov 08 14:45:43 2018
Quartus II Version	11.0 Build 208 07/03/2011 SP 1 SJ Web Edition
Revision Name	uartsw
Top-level Entity Name	uartsw
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	3,148 / 33,216 (9 %)
Total combinational functions	2,833 / 33,216 (9 %)
Dedicated logic registers	1,895 / 33,216 (6 %)
Total registers	1895
Total pins	5 / 475 (1 %)
Total virtual pins	0
Total memory bits	391,232 / 483,840 (81 %)
Embedded Multiplier 9-bit elements	4 / 70 (6 %)
Total PLLs	0 / 4 (0 %)

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	83.25 MHz	83.25 MHz	clock	

- Old instructions for Cyclone II using SOPC Builder
- Don't use for Cyclone V on DE1-SoC
- May be used for Cyclone 10 LP