LAB: True Randum Number Generators

KTH IL1331/FIL3030 Hardware Security

1 Introduction

In this lab tutorial[1], you will be introduced to Intel's Cyclone V architecture and get hands on experience with the ring-oscillator based True Random Number Generators (TRNGs) that you were introduced to during the course. The design software Quartus Prime will be used to synthesize and implement the design described by the provided source code written using the hardware description language Verilog. The procedure of implementing all parts of a functioning RO-TRNG using Quartus Prime will be described step-by-step. It will further be explained how to establish the communication between the FPGA and a computer via the JTAG interface and the the UART series port, and furthermore how to monitor on-chip signals using the Intel debug tool Signal Tap Logic Analyzer.

2 Background

The Random Number Generator (RNG) is an important cryptographic primitive which is used to generate private and public session keys, initialization vectors for stream and block ciphers, and random challenges for challenge-response authentication. Modern techniques for random number generation are divided into either computational methods or non-deterministic processes. The former are generally deterministic, and are therefore not sufficient for cryptographic use. To provide real security, an RNG requires a sufficiently high information entropy, a measure of uncertainty. The latter are therefore preferred as entropy sources. The randomness of a TRNG stems from a non-deterministic process of a physical device. Compared to its pseudo random number generator counterpart that produces deterministic, periodic sequences that seem random, the TRNG provides a higher level of security for cryptographic applications.

2.1 TRNG Approaches

There are different approaches for constructing TRNGs, these can be classified into four categories: noise-based TRNGs, ring oscillator-based TRNG (RO-TRNG), chaotic TRNGs, and metastable-based TRNGs. Noise-based TRNGs use analog circuitry to amplify, sample and quantize a noise source in physical devices. Ring oscillator-based TRNGs utilize timing jitter in free-running oscillators as a source of randomness. Metastable-based TRNGs extract randomness from the analog behavior of logic gates between two logic levels. Chaotic TRNGs produce randomness from repeated measurements of a physical chaotic system.

2.2 Intel Cyclone V FPGA Architecture

The basic building blocks of FPGAs are its look-up tables (LUTs). The Intel FPGAs Cyclone V provide these LUTs with additional logic blocks as a column structure. In addition to the

[1]This lab is partially based on the book chapter "Implementation of Delay-Based PUFs on Altera FPGAs", L. Feiten, M. Sauer, and B. Becker, Springer, 2017.

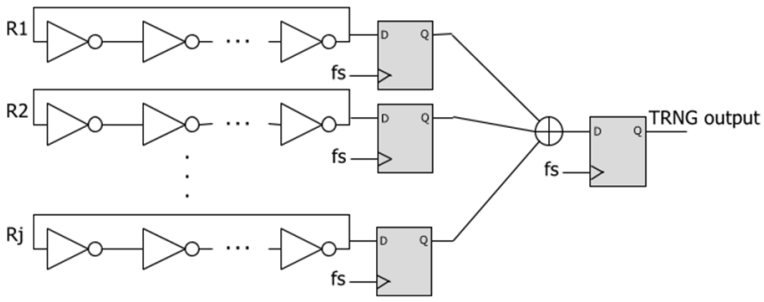


Figure 1: A typical RO-TRNG implementation.

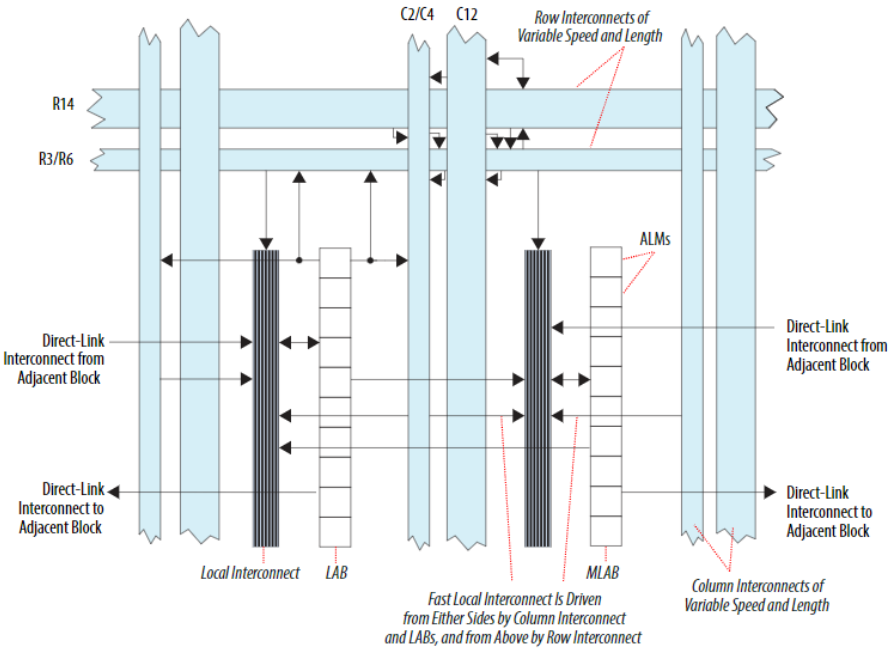


Figure 2: Cyclone V Fabric

FPGA fabric, our Cyclone V SoC also includes an HPS (Hard Processor System) consisting of a dual-core ARM Cortex-A9 but for the purposes of this lab, we will not utilize.

The logic array block (LAB) is composed of basic building blocks known as adaptive logic modules (ALMs) that you can configure to implement logic functions, arithmetic functions, register functions and in our case, ring oscillators.

Each LAB can drive 30 ALMs through fast-local and direct-link interconnects, but any given LAB has just ten ALMs.

The ALMs each contains two 6-input LUTs and four programmable registers.

Figure 4 shows the floorplan of a Cyclone V FPGA as viewed in the Chip Planner of Quartus Prime software. Each little rectangle in the full view (left) stands for one LAB. The right shows a zoomed-in view in which a single ALM of of a LAB is visible.

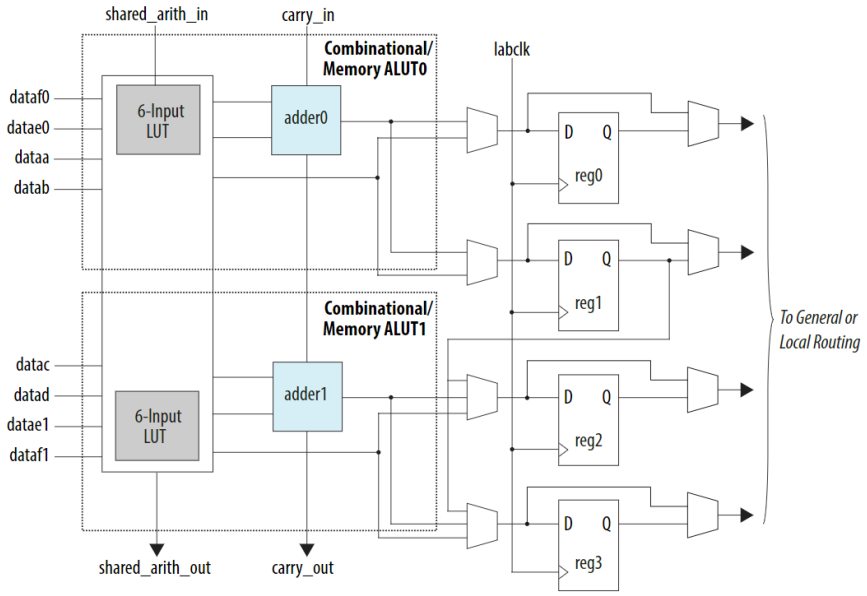


Figure 3: Cyclone V adaptive logic modules

3 Getting Started

This section aims to introduce the hardware prototyping board and computer-aided design software. We will also provide instructions to download and install the required software. In this lab, we will implement a classical RO-TRNG using Cyclone V on a DE0-Nano-SoC board. The DE0-Nano-SoC is designed to support a wide range of experiments. Figure 5 depicts the layout of the board and indicates the location of the connectors and key components. The DE0-Nano-SoC board combines a variety of logic and I/O devices onto a single printed-circuit board and allows you to configure and control these devices to create different applications. You should have a power supply and a USB cable to connect the board to your computer.

Lab materials, including user guide for the FPGA board and the installers of the software (Windows version) used in this lab, can be downloaded from: KTH-Archive

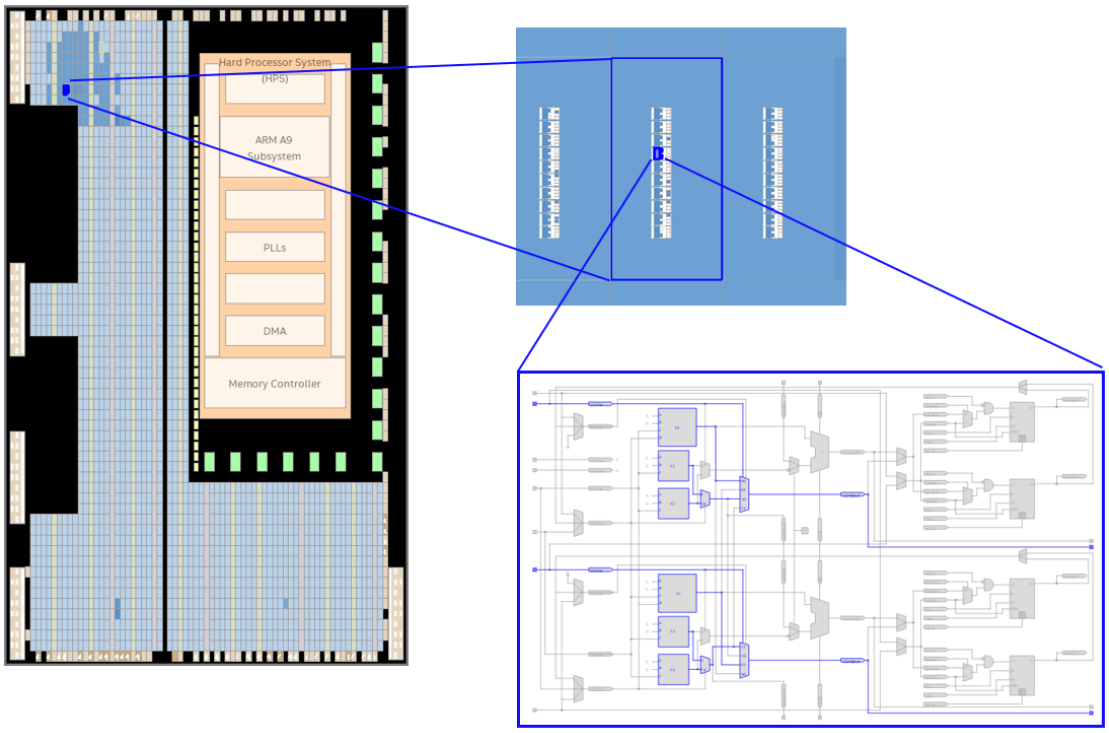


Figure 4: Cyclone V floorplan

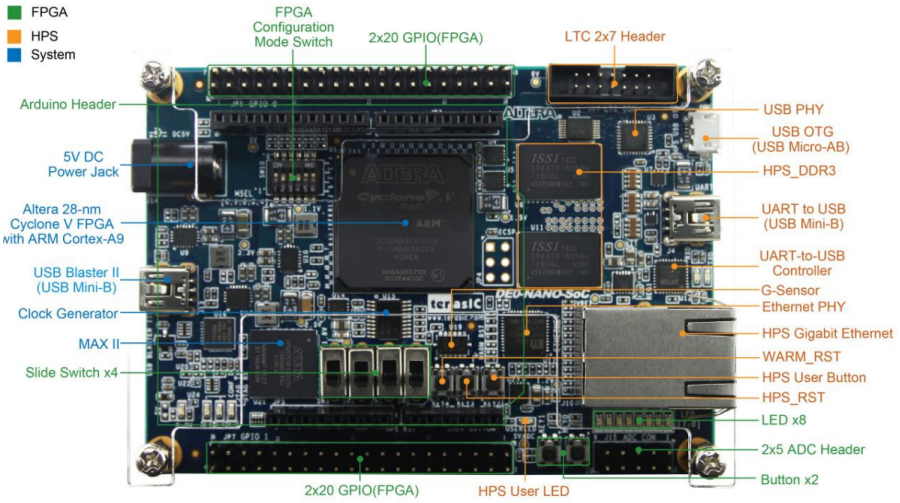


Figure 5: DE0-Nano-SoC development board overview

A pdf user guide of the DE0-Nano-SoC development board can be downloaded from: Terasic.

Intel Quartus Prime is a toolchain that enables analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the program-

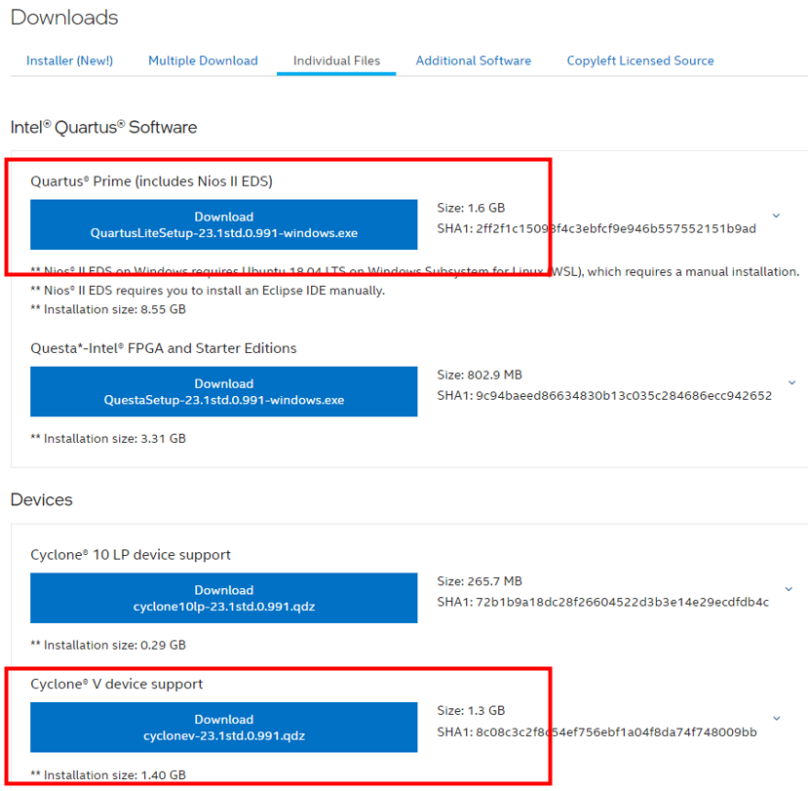


Figure 6: Quartus Prime Lite download page.

mer. It also makes the task of creating an FPGA design easier for the designer by automatizing the mapping, placement and routing to the FPGA hardware. The current version of Quartus Prime is 23.1 but the instructions of this lab are applicable for any version previous versions. The "Lite" edition is a license-free version of Quartus Prime that can be downloaded. This edition provides compilation and programming for a limited number of Intel devices. As for this lab, this free version can provide the functionalities/device supports that are needed.

3.0.1 Downloading and Installing Quartus Prime

As mentioned earlier, we will be using Intel's Quartus Prime Lite. This can be downloaded from the Intel website at: Intel Quartus Prime Lite

On the version-specific download page, click Individual Files. Make sure to download both Quartus Prime and Cyclone V device support (see Figure 6)

Direct Links for Windows version:

- Quartus Prime Lite 25.1
- Cyclone V Device File 25.1

3.1 Downloading and Installing USB-Blaster Drivers

Plug in the power adapter that was included with the board. Use the provided USB cable to connect the connector on the DE-series board labeled USB Blaster to a USB port on the

computer. Press the power button to turn on the DE-series board. The computer will recognize the new hardware connected to its USB port, but it will be unable to proceed if it does not have the required driver already installed. The DE0-Nano-SoC is programmed by using Intel's USB-Blaster driver software. If the driver software is not already installed, then follow the steps below to install the required driver.

- Apply power to the board and your computer to the board's programming port (the mini-USB port closes to power barrel jack).
- Select Start → Control Panel → Hardware and Sound and click Device Manager under Devices and Printers.
- Expand the Other devices category and double-click the entry labeled USB-Blaster II under "JTAG cables".
- In the General tab select Update Driver. The driver is available within the Quartus Prime software. Select Browse my computer for driver software.
- In the pop-up window, check Include subfolders and click Browse.
- Find the driver in your Quartus installation folder for example:

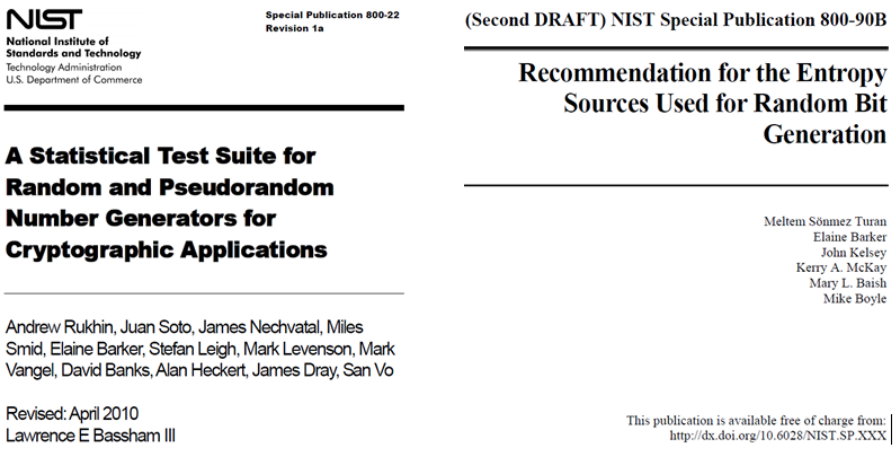
  `C:\intelFPGA_lite\23.1std\quartus\drivers`
- In the end, you will see a notification showing Windows has finished installing the driver software for this device Altera USB-blaster. Click Close and you can start using the DE0-Nano-SoC board.

3.2 Downloading and Installing Docklight

In addition to the USB-Blaster which are used to program and debug the FPGA devices, the Universal Asynchronous Receiver / Transmitter (UART) series communication may also be needed to transmit generated random back to the PC. A USB-UART adapter is provided with your lab kit and will be required for such series communication. The transmitted and recorded bit-stream can then be saved in a data file which can be further loaded into a test suit for randomness evaluation. Commonly used test suits include NIST 800-22 statistical test suite (cf. Figure 7 left) and NIST 800-90B entropy estimation suite (cf. Figure 7 right) both are standardized by the National Institute of Standards and Technology (NIST). The NIST 800-22 has been used extensively for accessing random numbers, but it only tests the statistical properties, so even a PRNG can past the test. On the other hand, the NIST 800-90B entropy estimation suit has been recently developed and is specially dedicated to testing physical/true random number generators. Docklight allows you to monitor and control serial communications between FPGA and PC. Docklight for Windows can be downloaded at Docklight.

4 Implementing the RO-TRNG on Cyclone V

This section details how to use the Quartus Prime software to design a RO-based TRNG. Our running example is the implementation of a basic RO-TRNG as described in the previous sections. For this running example, we assume that the name of the Quartus project is `top` and that a Intel/Altera Cyclone V SE 5CSEMA4U23C6N on a Terasic DE0-Nano-SoC development board is used.


NIST
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce

Special Publication 800-22
Revision 1a

A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications

Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, San Vo

Revised: April 2010
Lawrence E Bassham III

(Second DRAFT) NIST Special Publication 800-90B

Recommendation for the Entropy Sources Used for Random Bit Generation

Meltem Sönmez Turan
Elaine Barker
John Kelsey
Kerry A. McKay
Mary L. Baish
Mike Boyle

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.SP.XXX

Figure 7: NIST test suits for statistical/entropy evaluation.

4.1 Defining the Hardware Components

The hardware components of this tutorial are defined in Verilog HDL. When Quartus compiles a design, an analysis for logical redundancies is made and nodes just passing on a value like the RO's delay elements (cf. inverters in R1 → Rj, Figure 1) are removed automatically. To prevent the compiler from "optimizing away" the delay elements, a special comment should be added to the internal wires of each RO. Figure 8 shows the Verilog code for such an enabled RO, consisting of a single NAND gate and a number of delay elements (even number of inverters, in this case 10). When the enable input is set to 1, the RO starts oscillating. As explained in sec. 2.1, an extra D flip-flop after each RO is added, to prevent too frequent transients feeding into the XOR-tree.

When constructing an RO, it is important to avoid introducing deterministic properties. As seen in Figure 8. The RO given in the source file ro.v consists of 11 delay elements. If the number of delay elements were to exceed 20, they would no longer fit in the LE slots available within a single LAB of the Cyclone V architecture, meaning that the RO would be split into several LABs. The routing between two LEs within a LAB is shorter than that between two LABs, meaning that this would have negative implications for the RO-TRNG performance.

Constraining the RO to a single LAB column requires advanced FPGA programming (placement constrains) that is beyond the scope of this learning activity and therefore will be ignored in our implementation.

In the design provided in this lab, 128 ROs together with their sampling D-FF are fed into an XOR-tree. The output of the XOR-tree is then sampled by a cascade of D-FFs producing the output data stream. In order to transmit the TRNG output to a PC for performance evaluation, a UART block is also implemented in this design. The UART code (cf. rxtx.v) is adapted from[2]. Information sent through UART is sent in packets which usually consist of one start bit, one stop bit, and a byte of data. The receiver on PC and the transmitter on FPGA need to know at which rate the data is going to be sent; this rate is known as the BAUD rate (number of bits sent per second). In our running example, the BAUD rate is 9600 and the clock rate is 50 MHz. The clock signal comes from a 50MHz oscillator clock provided by the DE0-Nano-SoC).

As shown in Figure 9, the top level block diagram displayed in RTL viewer consists of the RO-TRNG block (trng) and the UART vlock (rxtx). The trng block is further divided into two parts (shown in Figure 10), the RO-TRNG circuitry (trng_core) consisting of ROs, XOR tree,

[2]https://github.com/progranism/Open-Source-FPGA-Bitcoin-Miner

```
//enabled ro with metastable prevention
module ro (ro_en,clk,ro_out);
  input       ro_en;
  input       clk;
  output      ro_out;

  reg meta;

  wire        r0/* synthesis syn_keep=1 */;
  wire        r1/* synthesis syn_keep=1 */;
  wire        r2/* synthesis syn_keep=1 */;
  wire        r3/* synthesis syn_keep=1 */;
  wire        r4/* synthesis syn_keep=1 */;
  wire        r5/* synthesis syn_keep=1 */;
  wire        r6/* synthesis syn_keep=1 */;
  wire        r7/* synthesis syn_keep=1 */;
  wire        r8/* synthesis syn_keep=1 */;
  wire        r9/* synthesis syn_keep=1 */;
  wire        r10/* synthesis syn_keep=1 */;

always@(posedge clk)
if(clk)
  begin
    meta <= r0;
  end

  //generate the ro
  nand    n1 (r1, r0, ro_en);
  not     n2 (r2, r1);
  not     n3 (r3, r2);
  not     n4 (r4, r3);
  not     n5 (r5, r4);
  not     n6 (r6, r5);
  not     n7 (r7, r6);
  not     n8 (r8, r7);
  not     n9 (r9, r8);
  not     n10(r10, r9);
  not     n11(r0, r10);

  assign ro_out = meta;

endmodule
```

Figure 8: ro.v

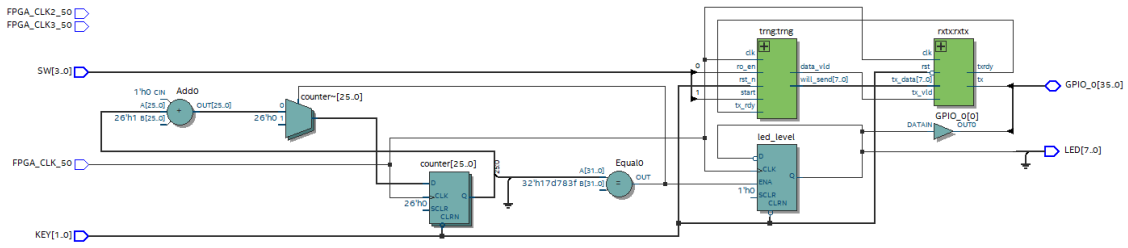


Figure 9: Quartus RTL Viewer for top.v.

and sampling D-FFs, and the 1-bit to 1-byte data conversion circuitry which prepares the output data for UART transmission. The circuitry at the bottom is a clock divider to flash an LED on

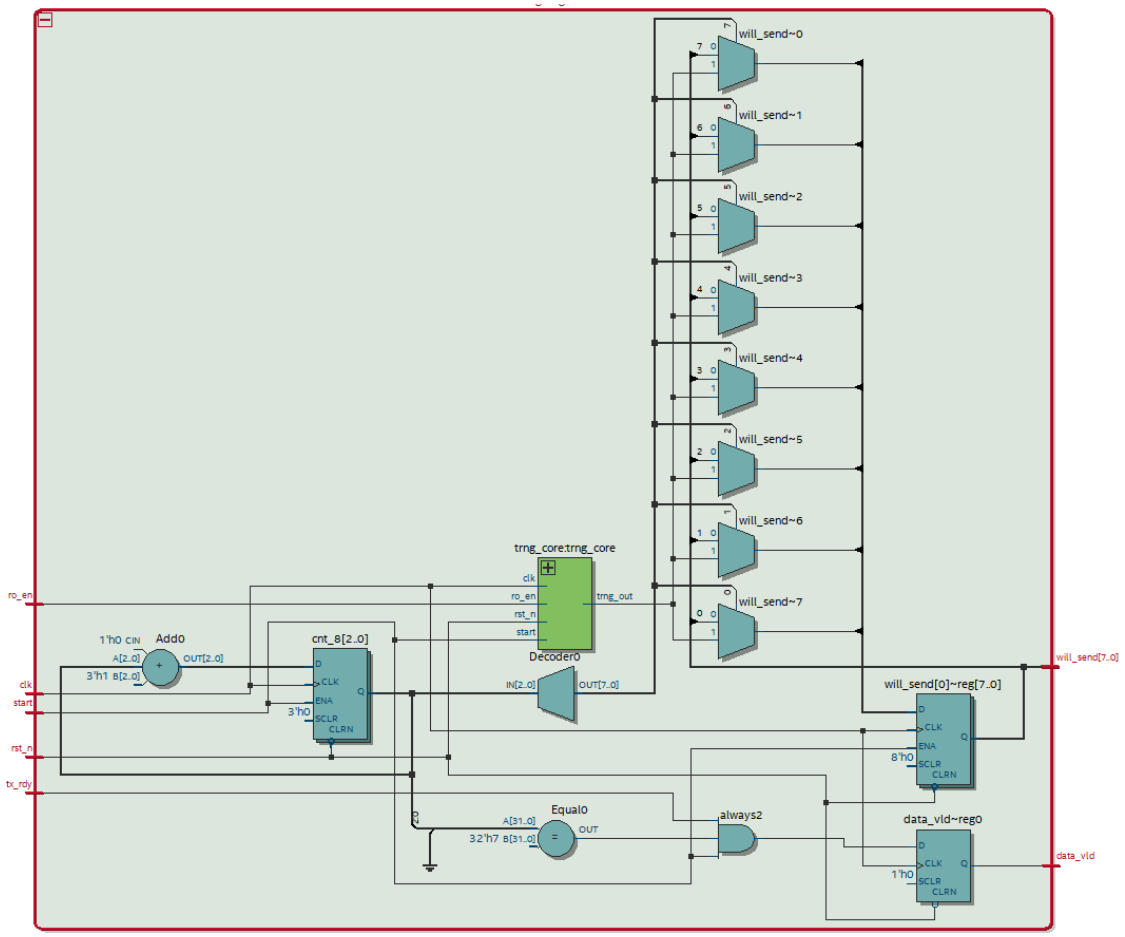


Figure 10: Quartus RTL Viewer for trng.v.

the DE0-Nano-SoC and is unrelated to the TRNG.

4.2 Implementation Procedure using Quartus Prime

This section illustrates the procedure of implementing the complete RO-TRNG design in Quartus given the provided source code. All the necessary Verilog source files are provided in the folder **\src** in the zipped directory **lab_TRNG_2024** available at the course web page on Canvas.

1. Create a new directory in your home folder, e.g., Documents\IL1331\Lab1 Unzip the downloaded zip folder into the created directory.

2. Start Quartus by double clicking the "Quartus (Quartus Prime 23.1std) Lite Edition)" icon or executing the "quartus" executable from the command line (Linux).

3. From the Quartus Menu Bar, select **FILE** → **New Project Wizard**

   Fill-in the Working Directory and Project Name as follows and click Next:

Working Directory	C:\Users\student\Documents\IL1331\Lab1
Project Name	trng
Top-Level Design Entity	top

4. Choose "Empty Project". Now we will add the source files for the project, these are located in \src. Click on browse ("...") and navigate to /Lab1/src Select all the Verilog source files (.v) and click **OPEN**. Click Next to advance

5. Select the "Board" tab and choose "Atlas-Soc (DE0-Nano-Soc)". Make sure to uncheck "Create top-level design file" and click Next.

6. Under "Simulation", make sure Verilog HDL is selected. Click Next.

7. Here you can see a summary of your project setting selections. Click Finish to complete the New Project Wizard.

8. Now that the project has been created. We need to import the pin assignments for our FPGA design. Under the "Assignments" menu, select "Import Assignments" and select the provided file in `src\pin_assignment.qsf`.

   Select Assignments → Pin Planner to open the Pin Planner window. Here you can view graphically how the I/O pins have been assigned.

9. From the Quartus Prime Menu Bar, select FILE → Open Navigate to /Lab1, select the generated top.qsf, and click OPEN.

   In this file, we can see the clock timings and other timing constraints in the design. We have done this already by the previous assignment import operation. Alternatively, you may define the timing by selecting Tools → Timing Analyzer.

10. Now, the project is set to be compiled. Synthesize the design by clicking the Start Compilation button on the toolbar.

5 Debugging and Recording the RO-TRNG's Outputs

5.1 Communication Between PC and FPGA

When the Quartus project is compiled successfully, the TRNG implementation is ready to be tested on real FPGAs. This section describes how the communication between an FPGA and a PC can be achieved via the FPGA's JTAG interface.

- On the Terasic DE0-Nano-SoC board, plug the USB Cable into the USB Blaster Port (closest to power jack). Plug the other end of the USB Cable into the computer. Do not forget to plug in power for the board.

- In the Quartus Prime window click the Programmer button on the Toolbar. Click the Hardware Setup... button and select DE-SoC [USB-0] from the drop-down menu for Currently selected hardware and click close to return to the programming menu. Mode should be set to JTAG.

- Click the "Autodetect" button and select "5CSEMA4", and select Yes when it prompts you to update the programming chain.

  The Programmer should now display two devices on the JTAG chain, the first device is the HPS ARM core (which we will ignore) while the "5CSEMA4" is the FPGA fabric. Select the fabric (5CSEMA4) from the list and click the "Change File..." button. Select the synthesized bitstream under `output_files\top.sof`.

- Tick the "Program/Configure" box. Settings should look like Figure 11

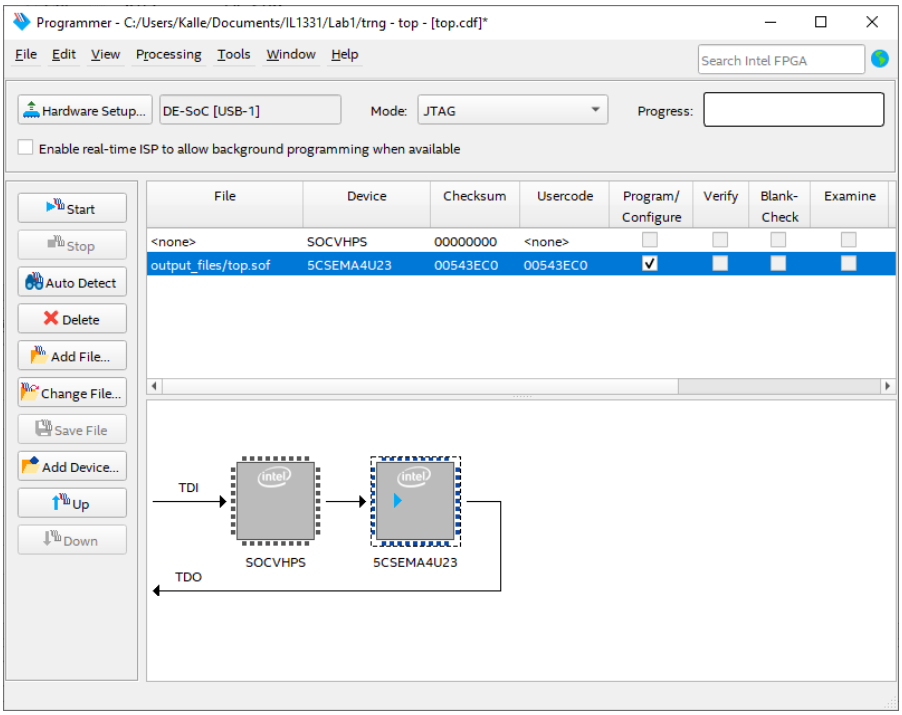


Figure 11: Quartus Prime Programmer Settings

Click Add File and choose top.sof in the output files folder you compiled into, then click the Start button to program the DE0-Nano-SoC board. When the progress bar reaches 100%, programming is complete.

You can now test the program on the DE0-Nano-SoC board by using the toggle switches located on the board. SW0 controls the ro_en signal which enables and disables the oscillation of all the ROs. SW1 controls the start signal which enables the UART transmission from the FPGA device. KEY0 (bottom right most button near the LEDs) controls the rst signal which resets some of the D-FFs in the design. The output of the RO-TRNG can be either monitored in real time using either Signal Tap Logic Analyzer (Section 5.2) or on PC through series port communication (Section 5.3).

5.2 Debugging Internal Signals using Signal Tap Logic Analyzer

- In the navigation bar, choose "Tools→Signal Tap Logic Analyzer"
- In JTAG Chain Configuration, choose Setup... and select USB-Blaster then click Close
- In the setup tab of the main window, double click to add nodes.

  Select Filter as Signal Tap: pre-synthesis

  Click List and select at least the signals shown in Figure 12

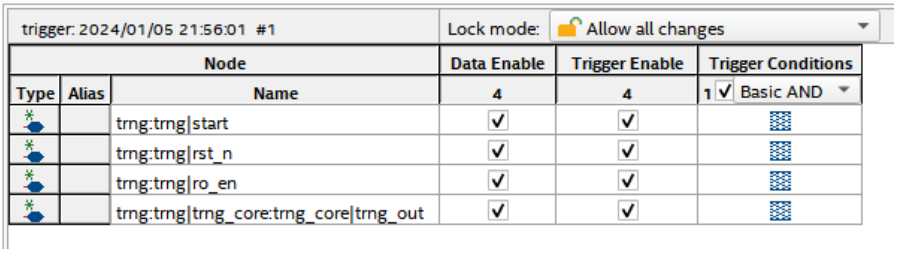


Figure 12: Node setup in Signal Tap Logic Analyzer

Click ">" to add the selected signals and click OK

- Right Click on the "trigger conditions" box and choose Basic AND.
- In the Signal Configuration Window: Next to "Clock" click "···"

  Click "List" and choose "clk"

  Click ">" and "OK"
- In the navigation bar, choose "File→ save"

  Name the file "top.stp"

  Click "Yes" when prompted to enable the file for the current project.
- In the Signal Tap Logic Analyzer window, recompile your design using the "Start Compilation" button.
- After compilation select the bitstream file in the top right JTAG Configuration. Should look similar to Figure 13. Press "Program Device" to upload the new bitstream with our TRNG design including the Logic Analyzer core.

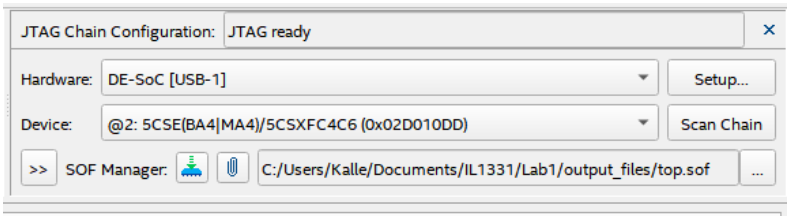


Figure 13: Bitstream setup in Signal Tap Logic Analyzer

- Now you can Click the purple play button Autorun Analysis at the top of the SignalTap window to begin signal acquisition.

  Don't forget to switch on SW0 and SW1 to enable the ROs as well as the UART transmission. Click KEY0 and watch the changes in the waveform.
- You can of course add other internal signals in the design to watch.
- To finish real time signal acquisition, click Stop Analysis.

5.3 Transmitting and Recording of Bit-streams

In order to further evaluate the output data stream, we need to transmit it to a PC through series communication. A USB-RS232 adapter is supplied with your lab kits. Only the ground (GND) and receive (RX) pins on the adapter is required for our purposes (see Appendix Figure 16).

On the PC, we need to run a RS232 series communication terminal to display and record the received data stream. Here we will use the PC software Docklight which you should have installed in sec. 3.2.

- Start Docklight
- In pop-up window, select start with a blank project / blank script and click continue
- Switch the communication format to Binary
- Select Tools → Project settings, fill in the option as shown in Figure 14, and click OK

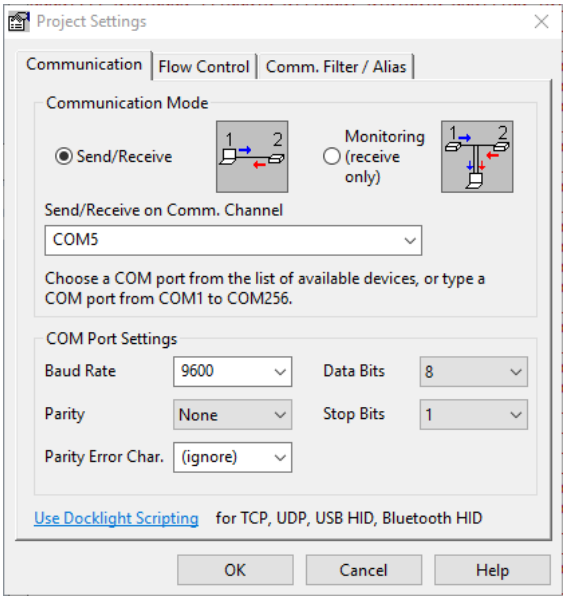


Figure 14: Docklight Settings

  Notice that communication channel in your case may not be COM5, and you need to select the correct COM port from the drop-down list. It should appear something similar as `Silicon Labs CP210x USB to UART Bridge`[3]

- Select Run → Start Communication and you will see the received real-time data byte stream as shown in Figure 15.

[3]Drivers for the UART adapter can be found at Link if needed.

Figure 15: Docklight data stream

5.4 Evaluate TRNG performance using a NIST test suite

In order to finish this task in time, it is strongly recommended that you download and familiarize yourselves with the test suites before the lab session! This step requires that the TRNG output is saved to a file, and unfortunately, this cannot be done using the free version of Docklight. One method of accomplishing this is to use a different serial terminal, such as RealTerm.

- RealTerm can be downloaded at: Link
- The NIST 800-22 statistical test suite can be downloaded at: Link
- The NIST 800-90B entropy estimation suite can be downloaded at: Link

A Jupyter notebook python project has been supplied in your lab package **lab_TRNG_2024** under "python_files" that implements some of the NIST tests if you have trouble running the above suites.

5.5 Self Exploration (Optional)

For those that have some familiarity with HDL and like to experiment. Try changing the TRNG design (change the number of RO and/or the number of inverters in each RO), what impact does this have on the test results? Speculate on your results, we would like to hear your theories!

6 Results

Please present your results from Section 5 to a lab assistant.

Appendix

USB to TTL Wiring Diagram

On the DE0-Nano-SoC, we defined in the RTL that `GPIO_1[1]` is the output of the UART block (`Tx`), therefore *Data* "comes out" of this pin and "into" the USB-UART adapter pin (`RXD`). Connect the ground pins of both the DE0-Nano-SoC and the USB-UART adapter to ensure proper voltage reference. **DO NOT connect any other pins** (ie. 5V or 3V3) as you may risk damage to the devboard, UART adapter or your computer!

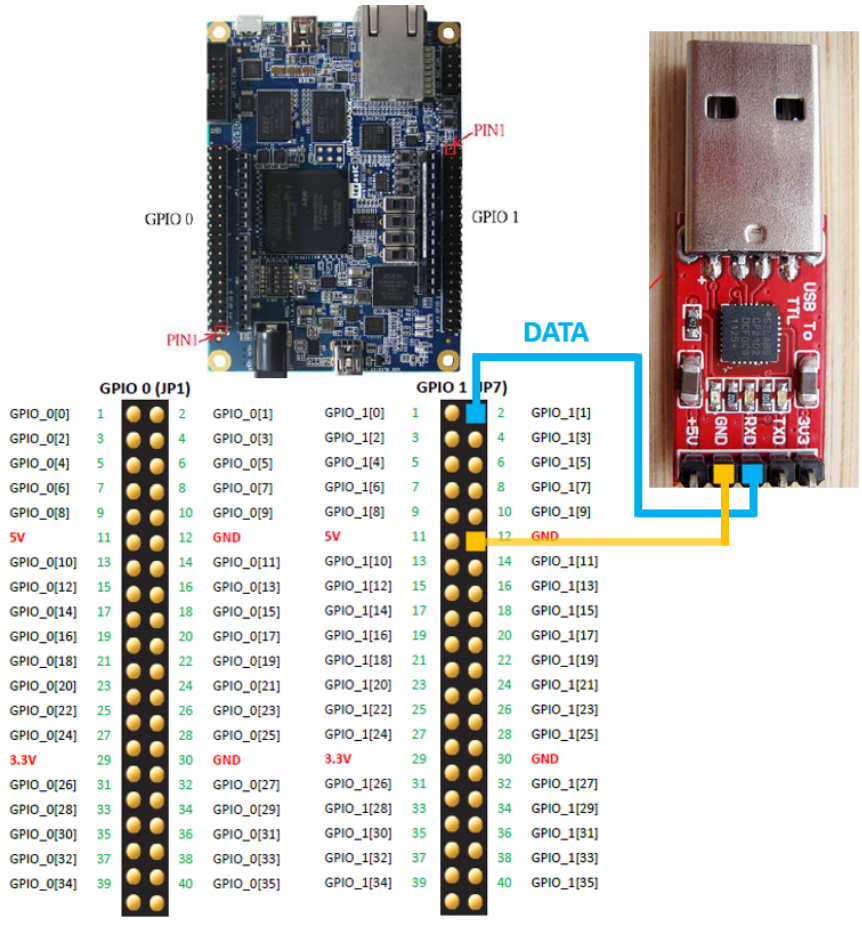


Figure 16: Wiring Diagram