CS254 (Spring 2017): Lab Assignment 1

January 25-February 1, 2017

In this lab, we are going to design the encryption-decryption module of our ATM controller. This module is intended to encrypt and decrypt data using a simple algorithm called *Tiny Encryption Algorithm (TEA)*. Towards this end, you are required to do the following.

1. Please go through the Wikipedia page on *Tiny Encryption Algorithm* at `https://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm` to understand TEA. The following points are worth noting:

   - The Wiki page gives C-style **(not VHDL style)** reference codes for `encrypt` and `decrupt`.
     The function `encrypt` encrypts two 32-bit plaintext blocks (`v[0]` and `v[1]` in the code) with a 128-bit key (`k[0]`, `k[1]`, `k[2]`, `k[3]` in the code), and generates two 32-bit cipher (encrypted) blocks, stored as `v[0]` and `v[1]` after the "for" loop terminates. This encryption uses 64 rounds of TEA, organized as 32 cycles (each iteration of the `for` loop.
     The function `decrypt` decrypts two 32-bit cipher blocks (`v[0]` and `v[1]` in the code) with a 128-bit key (`k[0]`, `k[1]`, `k[2]`, `k[3]` in the code), and generates two 32-bit plaintext (decrypted) blocks, stored as `v[0]` and `v[1]` after the "for" loop terminates. This decryption also uses 64 rounds of TEA, organized as 32 cycles (each iteration of the `for` loop.
     Note that the functions are written in C-style, and may not work as expected if you directly want to copy them as VHDL code. When writing code in VHDL, please keep in mind how processes work, and also how simulation behaves.

2. You must design a circuit that reads in 64 bits of plaintext or ciphertext, and depending on whether we want to encrypt or decrypt it, generates 64 bits of ciphertext or plaintext. Your circuit, when mapped down to the FPGA board, must behave as follows:

   - The circuit reads in the data to be encrypted/decrypted from the slider input switches on the board. These inputs are called `data_in_sliders` in the VHDL design. Note that there are only eight slider input switches on the board, so `data_in_sliders` must be declared as `STD_LOGIC_VECTOR (7 downto 0)`.
   - The circuit displays the encrypted/decrypted result on the LEDs available on the board. These outputs are called `data_out_leds` in the VHDL design. Note that there are only eight LEDs on the board, so `data_out_leds` must be declared as `STD_LOGIC_VECTOR (7 downto 0)`.
   - The circuit also has a few 1-bit inputs that it reads from the push-button inputs on the FPGA board. These inputs and their functions are as follows:
     - `next_data_in_button:` Recall that we need to read in 64 bits of plaintext/ciphertext for encryption/decryption; however we have only 8 slider input switches on the FPGA board. The way we want to solve this problem is by breaking up the 64 bits into chunks of 8 bits

each, and reading each chunk at a time. The `next_data_in_button`, which is intended to be mapped to a push button input on the FPGA board, tells the circuit when the next chunk of 8 bit input is ready at `data_in_sliders`, and can be read in.
So to read in 64 bits of plaintext/ciphertext prior to encryption/decryption, we must proceed as follows. We first present the least significant 8 bits of plaintext/ciphertext at `data_in_sliders`, and then push `next_data_in_button` once and release it. This should cause your design to read in the 8 bits of data from `data_in_sliders` and store them as the least significant 8 bits of the required plaintext/ciphertext. After `next_data_in_button` is released, we must present the next significant 8 bits of plaintext/ciphertext at `data_in_sliders`, and then push `next_data_in_button` again and release it. This should cause your design to read in the next 8 bits of data from `data_in_sliders` and store them as the next significant 8 bits of the required plaintext/ciphertext. This process is repeated until all the 8-bit chunks of the required 64-bit plaintext/ciphertext is read in.

- `next_data_out_button:` We need to display 64 bits of ciphertext/plaintext after encryption/decryption; however we have only 8 LED outputs on the FPGA board. The way we want to solve this problem is by breaking up the 64 bits into 8 chunks of 8 bits and displaying each chunk at a time. The `next_data_out_button`, which is intended to be mapped to a push button input on the FPGA board, tells the circuit when the next chunk of 8 bit output is ready to be displayed, and can be displayed on the LEDs.
  So to display 64 bits of ciphertext/plaintext, we first present the least significant 8 bits of ciphertext/plaintext at `data_out_leds`. This should cause the least significant 8 bits to be displayed on the 8 LEDs on the FPGA board. To display the next significant 8 bits on the LEDs on the FPGA board, we must push `next_data_out_button` once and release it. This should cause your design to output the next significant 8 bits of ciphertext/plaintext on `data_out_leds` so that this is displayed on the LEDs on the board. On pressing `next_data_in_button` again and releasing it, the next significant 8 bits of ciphertext/plaintext should be output at `data_out_leds`, and so on. This process is repeated until all the 8-bit chunks of the required 64-bit ciphertext/plaintext are displayed on the 8 LEDs on the board.
- `start_encrypt_button` and `start_decrypt_button:` These are intended to be mapped to two push button inputs on the FPGA board. After reading in all 64 bits of plaintext (or ciphertext), as described above, if we push `start_encrypt_button` (or `start_decrypt_button` once and release it, the design is supposed to compute the TEA-encrypted 64-bit ciphertext (or TEA-decrypted 64-bit plaintext) and present the least significant 8 bits of the result on `data_out_leds`. Subsequent chunks of 8 bits of the result can be read out (or displayed) by pushing `next_data_out_button`, as described above.
  Note that displaying the least significant 8 **bits of the result does not require pressing** `next_data_out` **at all. Note also that your design must remember which of** `start_encrypt_button` **or** `start_decrypt_button` **was pushed last, so that it knows whether to display the ciphertext output of the encrypter, or plaintext output of the decrypter on** `data_out_leds`**.**

- The circuit has a single-bit output `done` that is reset (i.e. has value '0') immediately after pressing `start_encrypt_button` or `start_decrypt_button`, and gets set (i.e. has value '1') after TEA-based encryption or decryption (as the case may be) gets done.

A skeleton of the VHDL code has been provided for you to fill in. You must use this skeleton to design your circuit. The skeleton code has the following components. **DO NOT CHANGE THE NAMES OF MODULES AND THEIR INPUT/OUTPUT PORTS**

- `debouncer`: When push button switches are used to give inputs to a circuit, often the switch bounces (i.e. goes on and off) initially a few times before settling to a stable position (although the user thinks the switch has been pressed only once). This can give rise to multiple inputs being read from the switch, although the user wanted to send only a single input. To protect us against this, the input coming from a push-button switch needs to be "de-bounced". In other words, we must wait for a certain amount of time to see if the input coming from the switch has stayed constant for the entire duration, before reading in the input from the switch.
  A debouncer module, adapted from
  `http://fpga-tutorials.blogspot.in/2012/12/deouncing-push-buttons.html` has been provided with the skeleton code, for you to get started on this.
  Note that slider input switches can also bounce. However, since we are reading inputs from the slider input switches only after some time (e.g. after `next_data_in` is pushed), the data read in from these switches can be expected to be stable when they are being read. Hence, we are not going to use debouncers for the slider inputs.
- `encrypter` and `decrypter`: These are the core modules you must design so that they do TEA encryption and decryption of 64 bits. The input and output ports of these modules are self-explanatory from their names, as given in the skeleton code. The computation (encryption/decryption) should start once the input `start` of the module goes to '1'. Note that the `start` input comes from a (debounced) push-button input switch, and hence it will go to '0' shortly after becoming '1', and your design must take care of this.
- `read_multiple_data_bytes`: This module helps in reading 8 chunks of 8 bit data input from its `data_in` port, as discussed above, and presents it as a 64 bit data output on its output port `data_read`. The input `next_data` is supposed to be the debounced version of the `next_data_in` push-button input of the overall design.
- `display_multiple_data_bytes`: This module helps in presenting (or displaying) 8 chunks of 8 bit data on its `data_out` port, as discussed above, given a a 64 bit data input on its `data_in` port. The input `next_data` is supposed to be the debounced version of the `next_data_out` push-button input of the overall design.

3. You must fill in all the missing parts of the skeleton design, design a testbench, simulate it and submit your code as follows.

   - On Wed, Jan 25, you must get started with designing some of the modules in VHDL during lab hours, and submit whatever code you have by 5pm on moodle. TAs will evaluate the progress of your work in the lab and your group will be given a marks out of 5 (out of a total of 20 for this assignment).
   - On Wed, Feb 1, you must come to the lab with a fully working design and testbench, and must make sure that you can generate a .bit file for mapping your design down to the FPGA board. We will provide you a constraints file and an FPGA board in the lab, and you will be evaluated (out of 15 marks) in the lab based on the following:
     - Discipline in your design (well-documented, modular designs with proper naming of variables, signals etc. are desirable);
     - How well you have tested your design by simulation (testbench creation and simulation results).
     - How well your design mapped to the FPGA board actually works.

     You must submit your final design files on moodle on Wed, Feb 1, as well. More instructions regarding the submission on Feb 1 will be provided closer to Feb 1.