CS254 (Spring 2018): Lab Assignment 4 Instructions

February 6, 2018

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` is 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 is 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 32 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 32 bits into 8 chunks of 8 bits, and reading each chunk at a time. The `next_data_in_button`, which is 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 32 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 causes the 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 causes the 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 32-bit plaintext/ciphertext is read in. If you press the button more than 4 times, it will again read and store the bits in the least significant 8 bits of plaintext/ciphertext. So don't press the button more than four times for one input.
  - `next_data_out_button`: We need to display 32 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 32 bits into 8 chunks of 8 bits and displaying each chunk at a time. The `next_data_out_button`, which is 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 32 bits of ciphertext/plaintext, we first present the least significant 8 bits of ciphertext/plaintext at data out leds. This causes 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 causes the 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_out_button` again and releasing it, the next significant 8 bits of ciphertext/plaintext are displayed at `data_out_leds`, and so on. This process is repeated until all the 8-bit chunks of the required 32-bit ciphertext/plaintext are displayed on the 8 LEDs on the board.

  - `start_encrypt_button` and `start_decrypt_button`: These are mapped to two push button inputs on the FPGA board. After reading in all 32 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 encrypted 32-bit ciphertext (or decrypted 32-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.

- 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 the 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. 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 encryption and decryption of 32 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 enable of the module goes to 1.

- `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 32 bit data output on its output port data read. Please read and understand the module description in the VHDL files given.

- `display_multiple_data_bytes`: This module helps in displaying 8 chunks of 8 bit data on its data out port, as discussed above, given a 32 bit data input on its data in port. Please read and understand the module description in the VHDL files given.