CS254 (Spring 2018): Lab Assignment 5 Instructions

March 11, 2018

In this lab, you are required to write a C program that uses the FPGAlink C API to communicate with your FPGA board to execute a specified set of actions. You will also need to modify the VHDL code of the railway signal controller implemented on your FPGA board, such that your program (runinng on a host computer) and your VHDL code (implemented on the FPGA board) together achieve what the system is supposed to do.

It may be useful here to recall a few things about our overall objective. We are trying to design a network of communicating railway signaling controllers to be placed at railway track junctions, with a backend host computer that has information about the overall railway network. Specifically, we expect the host computer to have information about all segments of railways tracks between all junctions, while each railway signaling controller (implemented in an FPGA board) will only have local information around the junction where the controller is located. Each controller is required to communicate periodically with the host computer (in our case, through the USB port) to obtain information about tracks its neighbourhood. It is also required to periodically update the host computer about changes in the status of tracks passing through the junction where it is located. The information received by a railway singal controller from the host computer must be used along with sensor inputs (implemented by the slider switches on the FPGA board) to determine the final sequence of signaling (displayed on the LEDs). In the final phase of the project, we will also allow two railway signaling controllers (your design mapped down to two different FPGA boards) to communicate with each other via the UART port on the FPGA board. This is particularly useful when the link with the host computer may be temporarily down.

As discussed in the last lab assignment, each railway signaling controller has a pair of co-ordinates – essentially, a pair of unsigned 4-bit integers, that identifies its location in the railway network grid. These coordinates should be declared as constants in your VHDL code, so that if we require you to change the coordinates, you can change the values of these constants, re-synthesize and re-program the FPGA board, and everything should work. Each controller also has a pair of dedicated channels (unlike in the last assignment where it used a single channel) for communicating with the backend controller. The FPGALink interface makes available 128 channels; let's call them channels 0 through 127. For a value of $i \in \{0, \ldots 63\}$, a railway signaling controller uses channel $2i$ for sending messages to the host computer, and uses channel $2i + 1$ to receive messages from the host computer. For example, you could choose channel 30 to send messages from your controller to the host computer, and channel 31 to receive messages from the host computer. Your VHDL design of the controller should declare these channel identifiers as constants that can be changed if we require you to do so later (of course, this means your VHDL code will need to be re-synthesized and re-programmed on the FPGA board).

We expect you to complete the assignment in two parts – one in-lab and one over the week.

Note that all communication between the host computer and the FPGA board must be encrypted – we will use the same 32-bit encryption and decryption algorithms we used in Lab 3 (week of Jan 23-30, 2018). Thus, any message to be sent from the FPGA must first be padded up or broken into 32-bit chunks, encrypted using a secret key (you can use 11001100110011001100110011000001 for example) and then sent

one byte at a time over a channel in the FPGALink interface. Similarly, on receiving an encrypted message from the host, the FPGA board must receive the 32 bits, one byte at a time, concatenate them together to form a 32-bit vector and then decrypt it. The situation at the host computer's end is similar – except that the FPGALink APIs directly allow you to read multiple bytes on a channel in the C program.

Part 1 (in-lab): Modify `main.c` for `flcli` such that the program does the following:

[H1] It reads a table in the following format from a file named "network.txt".

X-coord	Y-coord	Direction	TrackOK	NextSignal
2,	3,	2,	1,	2
1,	3,	5,	0,	1
⋮	⋮	⋮	⋮	⋮

Each line in this table contains five comma-separated unsigned integers. Each row describes the track in a particular direction at a specified junction. The first two entries in the row give the coordinates of the junction – these range from $0 - 15$ each. The third entry in the row encodes a direction from amon `N, NE, E, SE, S, SW, W, NW`. For convenience, we will assign the codes 0 to `N`, 1 for `NE`, 0 to `E` and so on, until 7 to `NW`. The fourth entry in a row is 0 or 1 value indicating whether the track in the specified direction through the specified junction is ok or not (0 means not ok, 1 means ok). The final entry in a row indicates the number of hops in the specified direction from the current coordinates in the railway network, where the next signaling contoller exists. This is an unsigned integer in 0 through 7, where 0 means that there are no further signaling controllers beyond the current one in the specified direction.

If a pair of co-ordinates doesn't appear in the table at all, it means there is no railway signaling junction at those co-ordinates. If a direction code doesn't appear for a pair of co-ordinates anywhere in the table, it means there are no tracks in that direction at the railway signaling junction with the specified co-ordinates.

[H2] It reads from channel 0 the encrypted co-ordinates of a railway signaling controller and decrypts it.

[H3] It re-encrypts the co-ordinates received on channel 0 and sends it back on channel 1 to figure out if this really came from a signaling controller (or if it was some junk values read on the channel).

[H4] It then waits to hear a special 32-bit encrypted acknowledgement, say `Ack1`, from the signaling controller (FPGA) on channel 0. You can design what this acknowledgement message from the FPGA should be (this could be your favourite sequence of 32 bits). Once the host computer receives this acknowledgement on channel 1, it knows that the coordinates that came on channel 0 indeed came from a genuine controller.

[H5] It then sends its own special 32-bit encrypted reverse acknowledgement, say `Ack2`, to the signaling controller on channel 1. You are free to design what this reverse acknowledgement should be (another of your favourite sequence of 32 bits).

[H6] The host computer now consults the table it read to figure out the 8 bits per direction it needs to send to the railway signaling controller. This requires searching the table for the coordinates of the controller and putting together all of the required information as 8 bytes.

[H7] The host computer then encrypts the first 4 bytes and sends them to the railway signaling controller on channel 1.

- It waits for the encrypted acknowledgment `Ack1` from the controller on channel 0.

[H8] On receiving the `Ack1`, it encrypts the last 4 bytes and sends these to the railway signaling controller over channel 1.

[H9] It again waits for the encrypted acknowledgment `Ack1` from the controller on channel 1.

[H10] Once the above cycle is over, it sends the encrypted acknowledgment `Ack2` on channel 0.

[H11] Then the host computer waits for 16 seconds and re-starts the entire process.

You are also required to modify `cksum_rtl.vhdl` so that it does the following:

[C1] It constructs 32 bits out of the 8 bits of its coordinates, encrypts them and sends them on channel 0.

[C2] It listens on channel 1 to receive 32 encrypted bits, decrypts them and checks whether its coordinates are embedded in the derypted message. The embedding of the 8-bit co-ordinates in the 32-bit message should be exactly the same as was used to construct the 32-bit message from the 8-bit coordinates.

[C3] It then sends the special encrypted acknowledgment `Ack1` on channel 0 and listens on channel 1 until it gets the special encrypted acknowledgment `Ack2` from the host computer.

[C4] After receiving the encrypted `Ack2` from the host computer, it receives the next 32 bits on channel 1, decrypts it and re-constructs the information about tracks in 4 directions passing through this junction.

[C5] It then sends the encrypted acknowledgment `Ack1` back to the host computer on channel 0.

[C6] Next, it receives the remaining 32 bits on channel 1, decrypts it and re-constructs the information about tracks in the remaining 4 directions passing through this junction.

[C7] It again sends the encrypted acknowledgment `Ack1` back to the host computer on channel 0.

[C8] Next, it waits to receive the encrypted acknowledgment `Ack2` from the host computer on channel 1.

[C9] It then displays the information about tracks in all 8 directions passing through this junction, as received from the host-computer on the LEDs, exactly as was done in the last assignment.

[C10] Finally, it waits for 16s and repeats the entire cycle.