CS254 (Spring 2018): Final Project Description

April 3, 2018

[Clarifications highlighted in red: April 8, 2018.]

The final project for this course will involve putting together various parts that you have already designed over the semester, along with some additional features that you need to add for the final project. The final project description, as given below, is in two parts: (a) a mandatory part, which all groups must complete, and (b) an optional part, necessary if a group wishes to shoot for AP.

The overall system we wish to design is a communicating railway signal controller, along with a backend software, to be run on a laptop, for communicating with the controller. The mandatory part and the optional part differ in what your controller communicates with over UART – either your laptop or another controller (your VHDL design mapped to another Digilent Atlys board).

The controller, implemented on the FPGA board, should have the following inputs/outputs:

- Slider switches S_0 through S_7 used to indicate status of tracks, *and* also to provide data to be sent to the backend computer and neighbouring controller (see details below).
- The "centre" push-button switch, which is pressed to reset the controller. On pushing this switch, the controller should come back to its `reset` state from whichever state it is in.
- The "up" push-button switch, which is pressed to indicate that the controller wishes to send data about tracks to the backend computer using `fpgalink`.
- The "down" push-button switch, which is intended to be pressed after the "up" push-button switch has been pressed, and after the slider switches have been set to correspond to the data that has to be sent to the backend computer. The "down" push-button switch is pressed to indicate that the data from the slider switches is now ready to be read and sent to the backend computer using `fpgalink`.
- The "left" push-button switch, which is pressed to indicate that the controller wishes to send data about tracks over UART port to its neighbouring controller (for those doing the optional part) or to the back-end computer (for others).
- The "right" push-button switch, which is intended to be pressed after the "left" push-button switch has been pressed, and after the slider switches have been set to correspond to the data that has to be sent to the neighbouring controller or backend computer (depending on whether you are doing the mandatory or optional part). The "down" push-button switch is pressed to indicate that the data from the slider switches is now ready to be read and sent over UART port to the neighbouring controller (for those doing the optional part) or to the back-end computer (for others).
- LEDs L_0 through L_7 used to denote the signaling status.

- `PROG` port used to connect the controller to the backend computer for communication over USB (using `fpgalink`). Note that this port is also used to download the `.bit` file from your laptop to program the FPGA.

- `UART` port used to connect the controller to the backend computer (for those doing the mandatory part) or neighbouring controller (for those doing the optional part) for communication over UART.

The controller, to be implemented on the FPGA board, should implement a state machine with the following macro-states and macro-state-specific actions:

S1: On pressing the `reset` button, all LEDs should light up for 3 seconds, and the controller should go to this state (also called the "reset" state). The controller should stay in this macro-state for 3 seconds with all LEDs lit up, and then go to macro-state S2.

S2: In this macro-state, the controller should communicate with the backend controller, as in the lab assignment statement of Mar 13. However, at the end of the communication protocol, instead of waiting for 8 seconds and restarting the cycle (as in step C10 of the problem statement of Mar 13), it should go to macro-state S3.

There is one modification to the signaling display to be implemented using the LEDs. If there is a train waiting in a direction, the track in that direction is ok, and if there is no train waiting in the opposite direction, but the next signal is only 1 stop away, the signal should be amber for 3 seconds.

S3: In this macro-state, the controller checks if the "Up" push-button switch has been pressed (i.e. user wants to send data to the back-end computer using `fpgalink`). If so, it waits in this state until the data is available (detected as a press of the "Down" push-button switch), sends the data from the slider switches to the backend computer and then goes to macro-state S4.

If the controller finds the "Up" push-button switch not pressed when it enters macro-state S3, it simply goes to macro-state S4.

S4: In this macro-state, the controller checks if the "Left" push-button switch has been pressed (i.e. user wants to send data to the neighbouring controller (for those doing the optional part) or to the back-end computer (for others) using `UART`. If so, it waits in this state until the data is available (detected as a press of the "Down" push-button switch), sends the data from the slider switches to the neighbouring controller (those doing the optional part) or to the backend computer (for others) and then goes to macro-state S5.

If the controller finds the "Up" push-button switch not pressed when it enters macro-state S3, it simply goes to macro-state S4.

S5: In this macro-state, the controller checks if any data has arrived on its UART port. If so, it updates its locally saved status of tracks passing through its junction using the received data. Note that for those implementing the optional part, the received data is the data from a neighbouring controller, while for those implementing the mandatory part, it is data from the back-end computer. Assume that the data received is already corrected/adjusted (for directions) for the controller that is receiving it. The controller then moves to macro-state S6.

If no data is received, the controller directly moves to macro-state S6.

S6: In this macro-state, the controller waits for T_0 seconds and then goes to macro-state S2. You are free to choose the value of T_0 based on your convenience.

The backend computer should behave exactly as in the lab assignment of Mar 13 when using `fpgalink`. The only exception is that in step H10, it should try to read encrypted data (data should be 1 byte, but has to be encrypted as 32 bits, as earlier) from the controller (FPGA board) over channel $2i$ (using `fpgalink`) and timeout after T_1 seconds if it cannot read any data. You are free to choose the value of T_1 based on your convenience. If the backend computer is able to read the data successfully, it should extract the byte from the data and update the back-end table with the data it read (only the `TrackOk` and `NextSignal` fields in the specified direction for the junction are to be updated).

The communication over UART must be unencrypted and using 8-bits with no parity bit and 1 stop bit. All communication using `fpgalink` should be encrypted and decrypted in chunks of 32 bits.

Note that the controller receives information about tracks from two sources – backend computer using `fpgalink` and neighbouring controller/backend computer using `UART`. Your controller needs to consolidate information received from these two sources in order to maintain its local view of the tracks, based on which actual signaling is to be done.

Recall that the controller receives 64 bits of track status from the backend computer in macro-state S2, with 8 bits used for each of the 8 directions. The 8 bits for each direction are to be interpretted exactly as in the assignment of Feb 20, i.e. most significant three bits to encode the direction, next significant bit to encode `TrackExists`, next significant bit to encode `TrackOk` and the least significant three bits to encode `NextSignal`.

When the controller receives a data byte of information from a neighbouring controller/back-end computer over `UART` in macro-state S5, the 8 bits in the data byte are to be interpretted exactly as above. Also, as mentioned above, information received in macro-state S5 can be used only to update `TrackOk` and `NextSignal` components of track information. There are a few cases to consider here. These are elaborated below:

- Whether a track exists in a particular direction or not is always obtained from the back-end computer in macro-state S2. Thus, if in macro-state S5, the data byte received over `UART` specifies a direction in which no track exists (as per information received from the backend computer in macro-state S2), then the data byte received over `UART` must be ignored.

- An existing track in a particular direction is to be considered ok by the controller for signaling purposes only if both the back-end computer considers this track ok (as communicated in macro-state S2) and the data byte (if any) received in macro-state S5 doesn't say that the track in this direction is not ok. If the information from either of these sources says that an existing track in a particular direction is not ok, the track in that direction must be considerd not ok by your controller for signaling purposes.

Groups that are able to successfully implement communication with a neighbouring controller (on an FPGA board) over `UART` may skip communication with the backend-computer over UART. However, groups that are unable to successfully implement communication with a neighbouring controller (on an FPGA board) over `UART` must implement communication with the backend-computer over UART.

Groups that are implementing communication with the backend-computer over UART must allow the facility to the user/evaluator to enter a byte (in hexadecimal) to be communicated to the controller over UART. This can be done by reading from a file or by asking the user to enter a byte in hexadecimal on the terminal in the appropriate state of the backend software.