Advanced Digital Design [VU]
Lab Exercise III

Andreas Steininger & Florian Huemer

Vienna University of Technology
January 18, 2018

Workcraft

In this lab exercise you will use the tool Workcraft [1] to generate asynchronous control circuits from Signal Transition Graph (STG) specifications. These circuits will then be used in FPGA hardware implementations of asynchronous circuits to control their operation.

Before you start with the actual exercise it is a advisable to consult the Workcraft documentation. Good starting points are the help page on STGs [1] and the tutorial on how to model and verify a C gate [2].

We provide FPGA implementations of three C gate versions (normal, setable, resetable) in the `common/workcraft` directory.

Task 1: LED Pattern Generator

Implement asynchronous LED pattern generators, that produce the patterns shown in Figures 1 and 2, respectively.

The controller circuits should have the interface shown in Figure 3. The outputs $LED1-3$ are directly connected to an output of the FPGA. You will have to implement some suitable logic to generate the ack input of the controller. To slow down the operation speed of the circuit use the timeout module provided in the `common` directory. This (synchronous) module basically behaves like a buffer with an adjustable delay. Make sure that each step of the LED patterns lasts at least one second. Furthermore, devise a reset strategy for your circuit (i.e. if the reset signal is asserted all state holing elements must be set to a defined state).

Use the templates provided in the `lpgA` and `lpgB` directories for your solution.

[1] `http://www.workcraft.org/help/stg_plugin`
[2] `http://www.workcraft.org/tutorial/synthesis/celement/start`

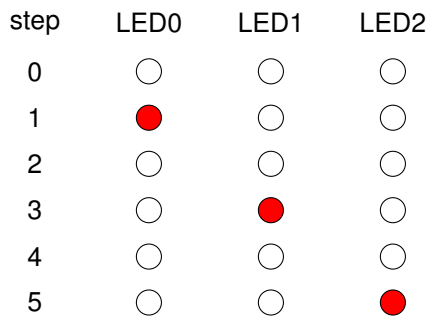

Figure 1: Pattern A

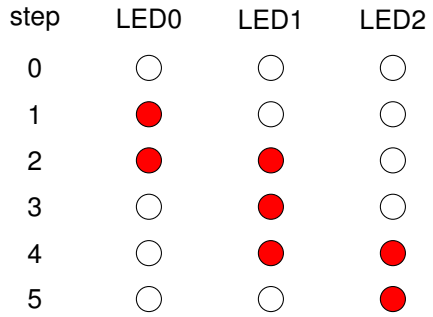

Figure 2: Pattern B

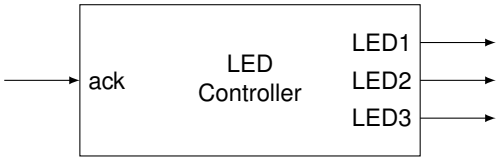

Figure 3: Asynchronous LED Controller

Task 2: Delay Insensitive Communication Link

Use the 1-of-4 encoding (discussed in the lecture) to implement an asynchronous delay-insensitive 4-phase communication link such as the one shown in Figure 4. We provide you with the UART/Bundled Data converter, that implements a sending and a receiving 8-bit wide 4-phase BD interface. This means that you will need to transmitt four 1-of-4 code words in parallel. Data received over the UART is buffered into a FIFO and transmitted over the BD interface only if the bd_tx_en signal is asserted. This input is connected to SW0 of the FPGA board. You can use this feature to generate burst transfers on the BD interface. Data received over the BD interface of the UART/BD converter is also buffered into a FIFO and sent by the UART transmitter.

Your task is to implement the transmitter (tx) and receiver (rx) modules. The tx module should receive data over the BD interface convert it to a 4-phase 1-of-4 encoding and transmit it over the DI link. On the other end of the link the rx module receives the DI data and converts it back to BD.

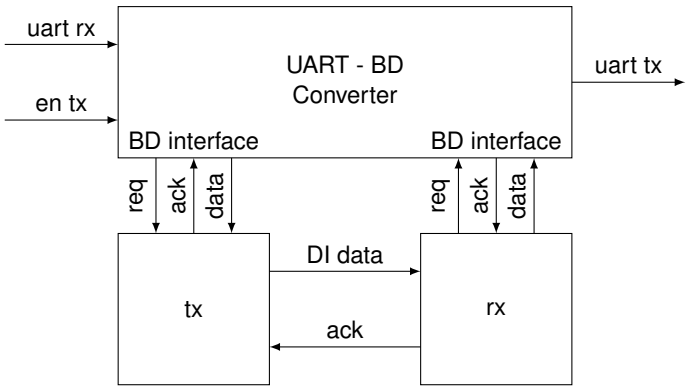

Figure 4: DI Link Overview

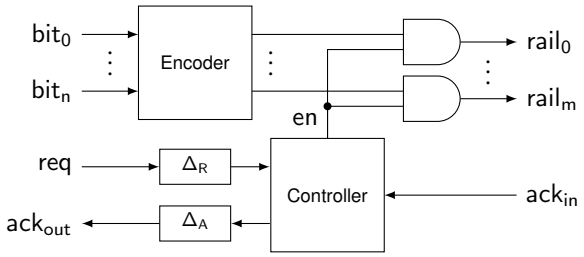

Figure 5: Transmitter Circuit (BD to DI)

In the following we provide descriptions for the transmitter and receiver circuits, which can be used as basis for your own implementation.

Consider the circuit shown in Figure 5 which converts a 4-phase BD to a return-to-zero (i.e. 4-phase) DI protocol. The converter is very generic and can basically be used with arbitrary DI codes (here we will use the 1-of-4 encoding). The actual code is defined by the encoder block. Note that it is not required for the DI encoder to be a QDI circuit, since it operates in the BD domain. It may even produce glitches at its outputs. However, it must be guaranteed that the delay Δ_{req} is long enough such that the outputs of the encoder are stable and valid when the req signal reaches the controller. In the beginning the en output of the controller is zero, thus the AND gates are masking the output of the encoder. The encoder can now make arbitrary transitions until it finally stabilizes. As soon as the controller receives a rising edge on the req input it asserts the en output to activate the AND gates. The DI data emerges on the output of the converter $(rail_0, ..., rail_m)$ and eventually causes the succeeding logic to assert the ack_{in} signal. The controller now asserts the ack_{out} signal, while it simultaneously deasserts the en output to switch the AND gates into the masking state again. This generates the null phase on the output. The delay element Δ_{ack} in the acknowledge path must ensure that all AND gates are definitely in the masking state, before the acknowledgment causes the preceding logic to change, i.e. invalidate, the input data to the converter $(bit_0, ..., bit_n)$. Eventually the null phase is acknowledged by the succeeding logic stage by deassertion of ack_{in}. Meanwhile the handshaking protocol on the input side of the converter is completed (deassertion of req and ack_{out}) and the circuit is finally ready to process the next input request.

Based on the given circuit description devise an STG specification of the controller and use Workcraft to synthesize a circuit out of it.

The receiver circuit, as shown in Figure 6, is a little bit simpler as it does not require a controller. Again, the actual DI code is defined by the decoder block, which is also not required to be a QDI circuit. Additionally a completion detector (CD) is required, to generate the req signal for the BD handshaking protocol. The DI input code word is presented to the CD and the decoder to produce the req as well as the binary coded output data. The delay Δ_R ensures that the data is consistent and stable at the successive pipeline stage, when the req signal arrives. However, since the CD already delays the req signal an actual delay element may not even be required in all cases.

Note that not all delay elements may actually be required for your implementation. Be prepared to argue, why you did or didn't include certain delays in your solution. Hint: Logic Lock Regions may be helpful to place some parts of your circuit on fixed locations of the FPGA.

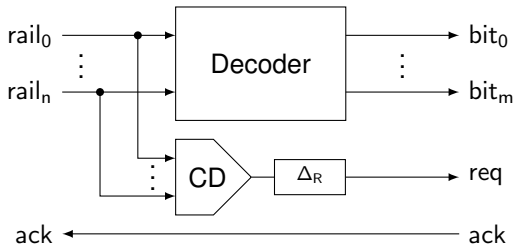

Figure 6: Receiver Circuit (DI to BD)

The DI bus as well as the DI ack signal will be connected to 40-pin headers of the FPGA board (see Figure 7). A (short) ribbon cable, which will be provided in the lab, is then used to connect the tx and rx modules. Use the middle connector of the cable to attach the logic analyzer and visualize the signal transitions on the DI bus. To deliberately generate different delays on the individual rails of the DI bus use the vector delay module. This entity as well as the UART/BD, the rx and tx module are already instantiated in the top level entity provided in the template.

Use the template provided in the `di_link` directory for your solution.

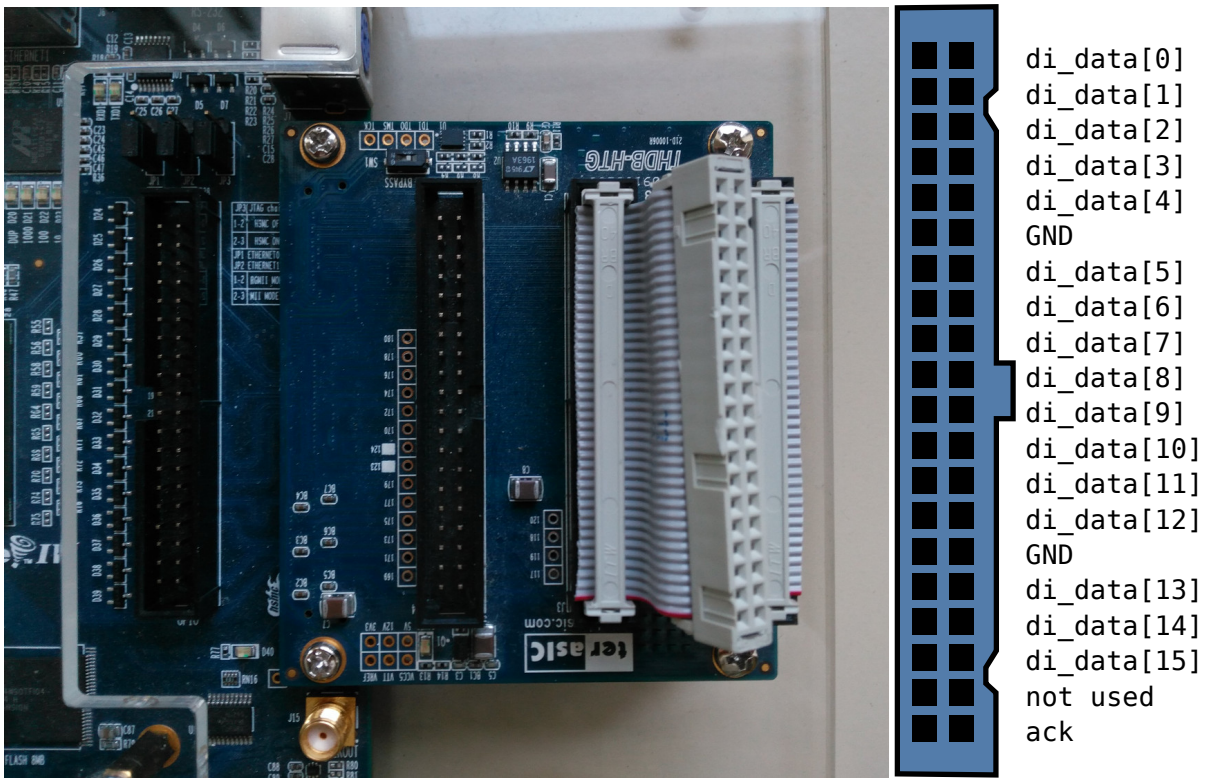

Figure 7: External FPGA connector

References

[1] *Workcraft Website*, 2016. `www.workcraft.org`.