Faculty of Science
Department of Computer Science
Chair for Embedded Systems
Prof. Dr. O. Bringmann

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Digital Design and Synthesis of Embedded Systems

Summer Term 2025

Exercise 5 : Deadline 08.06.2025 - 23:59 o'clock

The solutions are submitted via ILIAS as a packed archive (ZIP/TAR.GZ). This ZIP contains the written answers to questions, the source code (not as a listing in the PDF), the simulation results as VCD files, and the Makefile. Each of your programming tasks **must** be executable with your Makefile or with a standard python interpreter. If your python solution uses non-standard packages you have to provide a requiretmens.txt file. If you do not follow this format, your handing may be graded with 0 points!

Exercise 1 Process communication [15 Points]

In this task, you will create a small process pipeline that reads the temperature from a model of an 8-bit signed temperature sensor, calculates the average of the last three measurements and passes this to a status process. The model of the temperature sensor outputs a new random temperature value each time the `sample_in` signal changes. The status process reacts to the measured temperature and outputs it on a 14-segment display (14SA) [1]. An overview of the pipeline can be seen in Figure 1.

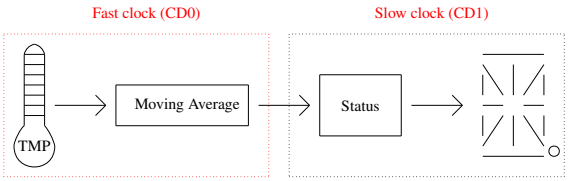

Figure 1: Process pipeline

As you can see from the illustration, the design has two different *Clock Domains*. A clock domain is an area of a design that receives the same clock. The domain of the fast clock (CD0) should have a clock period of **2ns**, the slow domain (CD1) a clock period of **3.7ns**. The status process works as follows:

- In the event of an asynchronous reset, an "R" (reset) is output on the 14SA
- After an asynchronous reset, at the next positive clock edge:
  - Displays an "O" (Okay) on the 14SA if the temperature t: $-16°C < t \leq 45°C$
  - Displays a "W" (Warm) on the 14SA if the temperature t: $\leq 45°C < t$
  - Displays a "C" (Cold) on the 14SA if the temperature t: $t \leq -16°C$

  The "R" remains displayed on the 14SA until the change.

[1] `https://www.mouser.de/datasheet/2/216/PSC39-21EWA-82862.pdf`

- The same applies to each of the outputs described above. For example, the output changes from "C" to "W" if the temperature t: $\leq 45°C < t$.
- From the Warm and Cold case, a transition to "W**.**" (Too Warm), or "C**.**" (Too Cold) can take place. The following must apply for Too Warm: Temperature $t > 60°C$. The following must apply for Too Cold: Temperature $t < -45°C$.
- From the Too Warm/Cold cases, an "R" is output on the 14SA in the following clock cycle.

Note: Create a separate subfolder in your submission for each of the following programming tasks. In this exercise you might need to cast a bit-vector to a signed value. In SystemVerilog you can to this as follows: `signed'(<VALUE>)`. Please use our provided testbench top source file for your simulations.

(a) Make yourself familiar with the provided source files. Note that the file `pipeline.sv` uses a pre-processes macro `` `USE_CDC ``. During simulation and synthesis. You can set this macro by invoking the makefile as follow:

`make` <*target name*> `USE_CDC=1`

(b) Create a state diagram that shows the states of the status process and their transition conditions. Highlight the start node. You can use graphviz (dot), yED, tikz, etc. to create the graph.

(c) Create a truth-table for the 14SA that shows which segements are for which character. Use the linked data sheet to appropriately control the display. Your table must contain at least the controls for the characters used. *Note:* Ignore the 12th pin of the 14SA.

(d) Implement the status module and test the behavior by running through every possible case described above. Make sure to include all specifications of the status module.

(e) Implement the Moving Average Module (AVGM) and test it with a series of 16 data values. Make sure that there are positive and negative numbers in your sample data. Make sure to include all specifications of the AVGM.

(f) Create the illustrated process pipeline from Figure 1 taking into account the specifications mentioned above. Add the synthesisable parts (all non-testbench and non-model files) of your pipeline to the correct variable in the Makefile and synthesise (`make synthesis` your design with `USE_CDC=0`. Simulate the design for 1000ns. *Note:* You will learn what this kind of synthesis (called RTL-Synthesis) is doing later in the lecture. For interested students: A RTL-Synthesis translates the hardware independet netlist (your design) into a hardware dependent design. The output file after synthesis (located in the `/synthesis` directory where your Makefile is will look significatly different than before.

(g) Analyze the generated VCD file. In particular, look at the times at which the data at the AVGM output and Status Module input changes. Pay attention to the behavior of the two clock signals at these points in time. Examine the output on the command line. Do you see any points-of-failures? Explain what you see and why this occurs in a few sentences.

(h) Implement the new module shown in Figure 2.

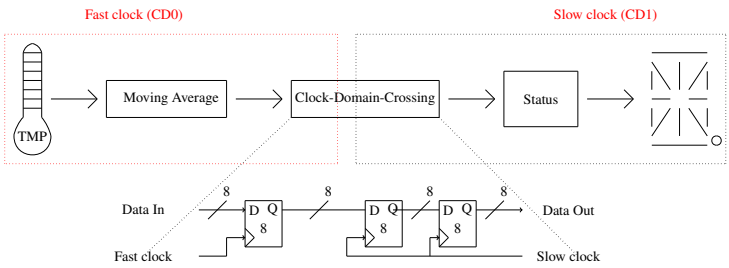

Figure 2: Process pipeline with clock domain crossing

The module has 3 inputs and 1 output. The inputs are: The data from the AVGM, the fast clock and the slow clock. The output is the data that is forwarded to the status module. The input data is written to a register/memory on a positive clock edge of the fast clock. The following two registers/memories read the data at their data inputs (D) on a positive clock edge of the slow clock. Pay attention to the correct bit width.

(i) Simulate the design again for 1000ns and compare the behavior of the circuit before and after the new module. What differences do you notice?

Exercise 2 High-Level-Synthese [5 Points]

Create a C file and implement a function `modulo` which is passed two unsigned numbers a, b and returns the value $a \bmod b$. You can use the built-in modulo operation of the C programming language here. Enter the name of your function in the variable `HLS_FUNC_TOP` and the file path to your C file in the variable `HLS_SRC`. Select `ns` as the time unit and `1` as the period duration. Then work on the following tasks. FYI: The HLS library for this is for the Xiling FPGA device `xc7vx485t-ffg1761-2`.

(a) Perform the HLS of your function. The output of the HLS, if everything has worked, is located in the created folder `hls`. Within this folder, you will find the output files under the folder `syn`.

(b) Take a look at the `_csyntch.rpt` report of your function and briefly describe what you see.

(c) Add the generated Verilog source files to the Makefile. The file `modulo.sv` should be last in the list.

(d) Create a testbench for the synthesized module. The top module of the HLS is located in the file with the same name as your function. Here: `modulo.sv`. Take a look at the port list of this module. In addition to the ports for the numbers a, b, the input ports `ap_clk`, `ap_rst`, `ap_start` and the output port `ap_return` are important. The `ap_` input ports are all high-active. The start signal may only be set to logic 1 if the reset is disabled. Create a clock with a period of **1ns**. Simulate the design for **at least 500ns**. Based on the waveform, can you infere how the module works, explain briefly.

(e) Expand the testbench to a total of 3 test cases.

(f) **BONUS:** Implement the 14SA module in C, perform a HLS and run a simulation (no synthesis is required). Do you think HLS made the implementation easiert?