MScTI_RES – Reconfigurable Embedded Systems

Assignment: Number 7 **Due Date:** 20.06.2022, 09:00 AM
Name: a single .zip file per upload on Moodle

For all problems Please submit to Moodle

- A brief document, where you describe your work and findings. Please add screenshots of waveforms from Simulation and SignalTap where applicable.
- .qar archives of your projects containing all relevant files to recreate your binaries (.vhd, .sdc, .stp, etc.)
- VHDL test benches used for simulation
- Scripts to run the simulation and display the waveforms
- All files to recreate the software (source, header, scripts, ...) and the output files created

Problem 1 Download material 7 from moodle. The .zip includes 3 directories

- **hdldoc**: contains a browser based documentation of the hardware implementation, generated with Doxygen. Top level file: *hdldoc/mini/html/index.html*
- **swdoc**: contains a browser based documentation of the software implementation, generated with Doxygen. Top level file: *swdoc/doc/html/index.html*
- **uc68hc08-minimal**: contains a minimal 68hc08 project, divided into the different subdirectories
  - **cyc1000**: contains the project files, constraints dependent on our target device, the CYC1000 board, debugging STP and a makefile for project generation
  - **doc**: contains some more documentation on the HC08 CPU: tutorial, application note on nested interrupts, reference manuals (by Motorola and Freescale), datasheet for an M68HC08, and technical data on the predecessor M68HC05
  - **hdl**: contains the source code. Within the subdirectory *arch*, some more target device dependent files are located
  - **host**: contains software for serial communication with the device
  - **ip**: contains IP cores
  - **sim**: test bench files and simulation startup scripts
  - **sw**: contains build scripts, the memory output .vhd file, an info file on interrupts and reentrant variables, and various subdirectories

- **doc**: contains more information on memory map and the minimal kernel
- **inc**: contains peripherals.h header file that contains register addresses and bit positions
- **lib**: contains hclib.lib
- **obj**: object file directory (empty upon download of .zip file)
- **src**: contains the main sw file main_simple.c

(a) After downloading the project from moodle, follow the steps from the lecture slides to compile the minimal software and the Quartus project and verify that both compile without errors. Familiarize yourself a little bit with the code. Start a simulation in ModelSim using the do-script located in *sim/hc08_tb_ghdl_modelsim.do* and observe the behaviour. You can also run a quick test in HW. For that, program the FPGA with the generated .sof file. There is a SignalTap file in the project that allows you to spy on some debug signals. For an interrupt to be triggered, you can send some dummy data via the UART. There is a simple example software for that located under *host/ftdi_write_data.cpp*.

(b) Add your timer peripheral from Assignment Sheet 6 to the Quartus project. Integrate your timer peripheral within the hc08_top.vhd - there is already a default timer component instantiated starting from line 753. Adapt the names of your timer's ports to match the interface implemented here. You can use the GPIO and UART entities as reference for the interface definition. Set the *hasTimer* boolean generic of the hc_top to **true** for your timer to be included. Compile and verify that you can compile the project without errors.

(c) In order to use your timer peripheral in your software: Update the register address map in peripherals.h to match your timer's internal register map and bit positions.

(d) Add an initialization program for your timer to the software (main_simple.c), so that your timer generates periodic interrupts. For synthesis, initialize your timer such that it generate interrupts at a rate of 1 Hz. For simulations, choose a higher frequency.

(e) Also add an interrupt handler for your timer peripheral to the hardware interrupt service routine. The interrupt handler shall clear the timer's interrupt, and increment a variable (check the GPIO and UART interrupt handlers for reference). Add the value of this variable to the variables that count the number of interrupts triggered by GPIO and UART. The sum of all 3 counters shall be written to the GPIO data output, which is connected to the LEDs.

(f) Compile your software, generate your memory file and include it in your Quartus project.

(g) Compile your firmware - and verify again that it compiles without errors.

(h) Test your project in simulation with ModelSim. Check that your timer generates periodic interrupts. You can check e. g. the IRQ signal or the LEDs to change.

(i) Test in hardware, e. g. by means of observing the LEDs updating at a rate of 1 Hz.