EFES Project Proposal

The idea of the project I want to realize is an embedded room temperature controller and management. It is thought to be used in a server room or something like this, so there are operators that can setup it.

It's a configurable system by an operator that can set several things in how the system works, like the interval time between a temperature sample and another, the average value to compute (how much sample consider), different thresholds and different speeds of fans for each threshold.

The system is based on an Altera Cyclone IV FPGA (inside we'll found a System On a Chip based on NIOS 2 by Altera and other externa peripherals) + various electronics components around it.

How the EFES's chapters will be covered?

Memories

In order to cover the Memories chapter, I decided to give to the system a sort of "history" of the samples. The history will be saved on one (or more, depends on how many data will be saved and the width (not decided yet)) on-chip memories implemented inside the Altera FPGA chip (they are known as M9K).

This memory will be accessed by the SoC via internal GPIO lines.

Programmable Logic Devices

I decided to cover this chapter by writing some VHDL code on my FPGA. In particular, my FPGA will be loaded with a NIOS 2 SoC so I'll write some code to interface it with the external pins (the Top Level Entity).

Not only, probably an entire SAR Logic will be implemented in VHDL (Actually I implemented it inside a microcontroller for testing purpose in order to have a feasibility analysis before proceeding, I hope it's not too difficult to translate its logic into VHDL).

AD and DA converter

To cover this chapter, I decided to create an "homemade" SAR Analog to Digital converter to sample the output of the analog temperature sensor (it's a LM35).

The sensor outputs an analog value that is of 10mV at every 1°C. I decided to cover a range of temperatures that spaces from 0°C to 40°C and this means an analog value that spaces from 0V to 400mV. The ADC input range is between 2.5V and 3.3V (I've to look if Quartus lets me set the output to 2.5V, I prefer it). Anyway, the input signal from the sensor is too small so I've to amplify it and then convert it.

The conversion is done on 7 bits, it should give me enough precision (similar to the one offered by the temperature sensor). On my FPGA I should have 7 pins available for interfacing with my ADC (it doesn't have a GPIO dedicated pin so I've to stole them from leds/7 segment display). If not, I will try with lower bits or with a serial transmission.

The DAC inside the ADC (on the feedback path) is done with a ladder network with resistors of 10K and 20K.

For the comparator, I have to look if I can found a good comparator in time for the exams session of February. If not, I was thinking to use a microcontroller with a two channel ADC to compare the output of the DAC and the input from the sensor and outputs on a digital pin if it's higher or lower (a comparator done in software in other words).

The output from the sensor is stable enough to do a complete sampling so a S&H it's not needed probably.

Interconnections

For this part of the course, the idea is to give to an operator the ability to configure and administrate the device through a serial interface. The FPGA will be interfaced with the UART and connected to a modern computer trough an UART to USB converter (I've a CP2102 converter).

The operator in this way can enter in the system in an admin mode (through, I think, a python visual interface) where it can see all the entire history (read from the m9k as said before) and configure the system behaviour like the sample period, on how many samples do the average, the thresholds and the speed of the fans.

Maybe, if there will be time, I can add an "Advanced mode" where it's possible to change the clock frequency of the converter (changing the total Tconv).

Processor Peripherals

So, as said, we have an entire SoC available on the FPGA. There will be different peripherals (for sure we have an SDRAM Controller, a timer, different GPIOs, an UART peripherals and some PWM controllers).

For sure I will program manually (without using NIOS's drivers) the GPIO and UART peripherals (probably I will write my own very basic drivers in order to not do everything manually).

The PWM peripherals will be used for driving at different speeds the two DC motors.

UART will be used through interrupts (using the internal interrupt controller of the NIOS 2 core).

Power Management

For this last chapter, I decided to manage some actuators. In particular, I decided to use two DC motors driven with PWM.

The problem is that the output pins from the FPGA (or others microcontrollers) don't have enough current to drive them correctly so I will use MOSFETs to drive them (with heatsink, I don't want to burn them). The PWM signal is connected to the gate in this way the DC motor will be driven with a separate voltage supply (I didn't decide the voltage yet).

If there will be time, I will build and add to the circuit two H-bridges too to drive them in different polarity.