CS-476: Embedded System Design

Practical work 3

GPIO-module

Version:
1.0

Contents

1 Introduction **1**

2 Exercises **2**

2.1 prerequisites 2

2.2 Task 1 2

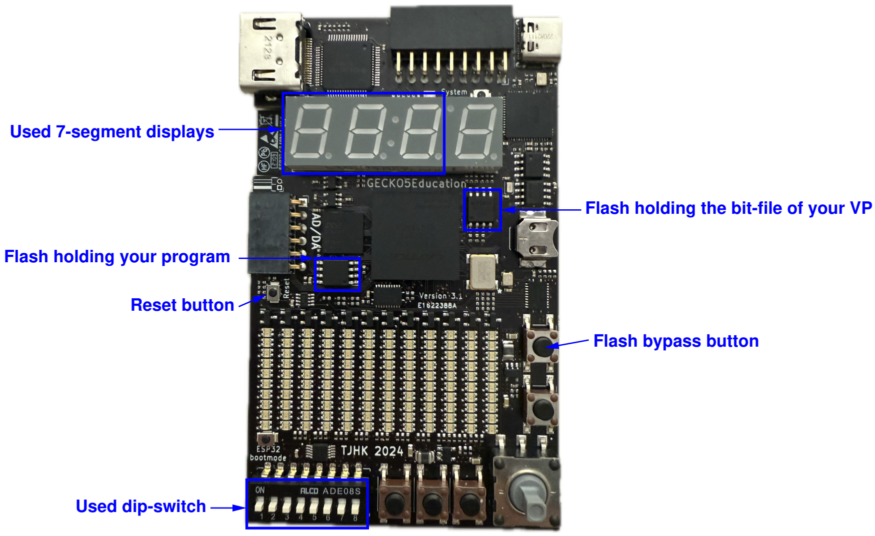
2.3 Task 2 2

2.4 Task 3 2

2.5 Adding a block to an existing system 3

1 Introduction

In this lab you will deal with a GPIO module. As the Sobel algorithm uses a threshold of 8-bits we are going to use a dip-switch to control this threshold. Furthermore we are going to display it's value in decimal on three 7-segment display's.



In the above picture you see the used elements. There are some nice features put in place to ease the usage of the platform:

- You can store your virtual-prototype on the GECKO5Education by usage of the flash that holds the bit-file. To make the bit-file persistent use `openFPGALoader -f or1420SingleCore.bit`.
- You can also store your program on the GECKO5Education by usage of a SPI-flash. In case the SPI-Flash holds a program, the bios will load and execute it and you will not go into *upload* mode.

To program the binary into the program flash you can perform following steps:

1. In the BIOS, upload the `.cmem`-file as usual to the vp.
2. Do not execute the program with `$`, but give the command `*f`. This will store the program into flash. Now each time you start your system the program in flash will be executed.

Of course, you have a way to go back to the upload mode:

1. Press and hold the `flash bypass button`.
2. Press and release the `reset button`.
3. You will now be again in the *upload mode*, and you can erase the program-flash with the `*e` command.

2 Exercises

2.1 prerequisites

Familiarize yourself with the data bus. Read the documentation and the timing diagrams in the slides. The GPIO module should be designed in such a way that it only supports single-word transactions, whereby the byte-enables all must be active. If this is not the case, the module should generate an error. Draw a time sequence of the bus transactions where an error is triggered (this is very easy with wavedrom). Note: for these situations it does not matter whether it is a read or write action.
Furthermore, all signals on the bus-in port should be provided with flip-flops to reduce the critical path in the bus and prevent oscillations. Draw two timing diagrams again where you clearly show a successful read and a successful write transaction. To do this, draw all signals from the bus-in port, the signals after the flip-flops, any control signals that you need internally and the signals at the bus-out port.

2.2 Task 1

Create a generic GPIO block where you use parameters to set the memory base address (`Base`), the `NrOfInputs` and `NrOfOutputs`. A maximum of 32 inputs and 32 outputs should be available, and a minimum of 1 input and 1 output. When describing the GPIO module, note that the CPU is big-endian and the bus is little-endian. Furthermore, writing to the address `Base` will set the value of the outputs, whilst reading from the address `Base` will read the value of the inputs. All inputs that are not used will return `0`. The inputs, as well as the outputs, are LSB-aligned.

2.3 Task 2

Add your GPIO module to the top-level `or1420SingleCore.v` so that you can read in a dipswitch and control three 7-segment displays. The GPIO block should be mapped to address `0x40000000`. The section 2.5 will help you to insert a new block. Note that the dip switches are low-active and the 7-segment display is low-active and scanning, meaning that you have to use a counter that selects each time one of the segments. For the dipswitch, the leftmost switch should represent the MSB and the rightmost switch the LSB. For the 7-segment displays, segment A is the LSB, segment B is bit 1, segment C is bit 2, ..., and the bit 7 is the dot. The LSB-byte is the right most 7-segment display, the second byte is the middle 7-segment display, and the third byte is the left-most 7-segment display. Hence the three 7-segment displays occupy a total of 24-bits.

2.4 Task 3

Download the grayscale example solution from moodle. This solution implements a custom instruction that performs the RGB565 to grayscale conversion on 4 pixels at a time (the optional exercise of last-week graded PW). Extend this program such that for each picture taken:

- The value of the dip-switches are read in.
- The read-in binary value of the dip-switches is displayed in decimal on the three 7-segment displays.

- Perform a binarisation of the image by setting a pixel to white (255) when the grayscale value is above the value read-in by the dipswitch, and to black (0) when the grayscale value is equal or below the value read-in by the dipswitch.

2.5 Adding a block to an existing system

To add your own block to an existing system, carry out the following steps:

- Create the Verilog file of your new block and save it under `virtual_prototype/modules/gpio/verilog/gpio.v`.
- Add the gpio-verilog file to the file:
  `yosysOr1420.script`
  in the directory:
  `virtual_prototype/systems/singleCore/scripts/`
- Edit the verilog of the toplevel `or1420SingleCore.v` as follows:
  - Modify the module definition by adding the inputs and outputs that come into the FPGA (dipswitch), and go out of the FPGA (7-segment displays).
  - Define the required signals, for example:
    - `sGpioBusError`
    - `sGpioAddressData`
    - `sGpioEndTransaction`
    - `sGpioDataValid`
  - Add your gpio-module as a component in the system.
  - Connect your gpio-module to the bus system (at the bottom of `or1420SingleCore.v`).
  - Modify the `gecko5_or1420.lpf` file in the directory:
    `virtual_prototype/systems/singleCore/scripts/`
    such that you correctly connect to the dipswitch and the 7-segment displays. The pin-information where these components are connected to the FPGA can be found in the gecko5.lpf file on moodle.
- Execute `../scripts/synthesizeOr1420.sh` and test whether synthesizing and place and route work.