⚠ You have previously submitted this assignment with David Rama Jimeno. Group can only change between different assignments.

This course has already ended.

« 14. Exercise 11 (/comp.ce.240/spring-202... 15. Exercise 12 » (/comp.ce.240/spring-202...

COMP.CE.240 (/comp.ce.240/spring-2024/) / 14. Exercise 11 (/comp.ce.240/spring-2024/E11/) / 14.1 Exercise 11: I2C-bus controller

Assignment description (/comp.ce.240/spring-2024/E11/i2c_config/)

My submissions (1/1000)

Exercise 11: I2C-bus controller

Your task is to implement an I2C-bus controller which configures the DA7212 audio codec before the synthesizer begins to feed data to it. After the configuration, the I2C-controller does not do anything.

After this exercise, you are even more experienced in interpreting real life specifications. You will also learn the basics of tri-state logic.

Useful datasheets / information sources:

- The Dialog DA7212 audio codec datasheet (https://www.renesas.com/eu/en/document/dst/da7212-datasheet)
- User guide of the ARD-AUDIO-DA7212 Audio Shield (https://www.renesas.com/eu/en/document/mat/ard-audio-da7212-audio-shield-user-guide) (the shield on top of the PYNQ board where the codec is installed)
- I2C-bus specification (https://plus.tuni.fi/graderA/static/compce240-s2024/UM10204_NXPSemiconductors.pdf).

The interface of the controller

- Entity name: `i2c_config`
- Integer type generics and default values
  - `ref_clk_freq_g, 50 000 000 (Hz)` (frequency of clk-signal as information to your block)
  - `i2c_freq_g, 20 000 (Hz)` (i2c-bus `(sclk_out)` frequency)

    - `n_params_g, 15` (number of configuration parameters)
    - `n_leds_g, 4` (number of leds on the board)
- Signals
    - `clk`
    - `rst_n`
    - `sdat_inout` (direction is `inout` and width is 1 bit)
    - `sclk_out` (width 1 bit)
    - `param_status_out` (width `n_leds_g`) --
    - `finished_out` (width 1 bit)

I2C-bus

> *"Philips Semiconductors (now NXP Semiconductors) developed a simple bidirectional 2-wire bus for efficient inter-IC control. This bus is called the Inter-IC or I2C-bus. Only two bus lines are required: a serial data line (SDA) and a serial clock line (SCL). Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, up to 1 Mbit/s in the Fast-mode Plus (Fm+), or up to 3.4 Mbit/s in the High-speed mode. The Ultra Fast-mode is a uni-directional mode with data transfers of up to 5 Mbit/s."*
> *- I2C-bus specification*

Start by reading the first three chapters from the I2C-bus specification (https://plus.tuni.fi/graderA/static/compce240-s2024/UM10204_NXPSemiconductors.pdf). Sections 3.1.3, 3.1.4, 3.1.5, 3.1.6 and 3.1.10 as well as Figure 9 are especially useful.

Notice

- I2C-bus consists of two signals (sdat and sclk)
    - sdat = serial data (SDA)
    - sclk = serial clock (SCL)
- Our block works as bus master on the FPGA and Dialog audio codec works as slave
    - We use Standard-mode
    - We use 7-bit addresses
    - **We make only write operations so the zero bit (= write) can be attached to the end of the address**

After eight sent bits, the receiver acknowledges the transfer using signal `sdat_inout`. `sdat_inout` is connected to *tri-state logic* (see, e.g., Wikipedia (http://en.wikipedia.org/wiki/Three-state_logic)) on the FPGA-chip which enables two-way data transmission. When transmitter writes value 'Z' to the inout-signal, it can read the value from the signal driven by the receiver.

In Section 3.1.6 in the I2C-specification, it is mentioned: "the transmitter releases the SDA line during the acknowledge clock pulse so the receiver can pull the SDA line LOW...". This means that the transmitter drives 'Z' to `sdat_inout` and reads the acknowledgement value driven by Dialog audio codec.

The datasheet of the DA7212 codec (https://www.renesas.com/eu/en/document/dst/da7212-datasheet) has also some information about the I2C protocol support on the chip on pages 55-58. There are multiple modes but with the given parameter setup (a table below) the "byte write" way of doing things is recommended (figure 22).
Notes

- 'Z'-value can be driven to any std_logic- or std_logic_vector-type signal or port
- Notice that tri-state logic cannot be used inside the FPGA-chip - only in some FPGA pins
- Ensure that SCL and SDA do not change simultaneously as it violates the I2C timing rules which cannot be noticed by the synthesis tool
- Strict timing rules are expressed in Table 10 and Figure 38 of the specification. (Violating these is the most common reason for a boomerang...)
- The I2C Bus Specification outlines that upon receiving a NACK, a stop OR repeated start condition can be use before re-transmitting the erroneous message. To make things easier, we ask that in your implementation, please use the **stop condition only**.

Controller functionality

Your controller (transmitter) sends messages via the I2C-bus to Dialog DA7212 audio codec. The transmission consists of start condition, address transmission, read-write-bit, target register address, actual data and stop condition.

The easiest way to implement the controller is probably by a state machine which contains few states, e.g., for start, end, data transmit and acknowledgement listening. You can implement your controller also in some other way. The requirement is that it works with the FPGA and follows the coding rules. Additionally, if the audio codec gives negative acknowledgement (NACK), the current three byte transmission is restarted from the beginning. I2C supports many types of transmissions but now we make transmissions always with a length of three bytes.

Your controller generates a slower clock signal `sclk_out` (frequency `i2c_freq_g`). The audio codec uses the rising edge of SCL for sampling signal `sdat_inout`, and at that moment the data signal has to be stable.

Your controller makes `n_params_g` (= 15) transmissions. The target of every transimission is the audio codec so the device address is always the same 7-bit address **"0011 010"**. Attach the write-bit **'0'** also to the address. Every transmission consists of two data bytes (byte = 8 bits). The first byte defines the internal register address of the audio codec and the latter contains the value. These address-value pairs are listed in the following table.
All the registers in the device are listed in the table on the pages 65-70 and individually on the pages after them, with more information.
The registers' addresses are written in the datasheet with hexadecimal values. In the table below they have been translated to binary.

The following configuration data is used to get the audio codec to the desired state.

Register address [15:8]	Value [7:0]	Register name	Description
0001 1101	1000 0000	cif_ctrl	Reset configuration and set control interface (I2C) mode.
0010 0111	0000 0100	pll_ctrl	System clock is MCLK, and MCLK = 10-20MHz (in our case 12.288MHZ)
0010 0010	0000 1011	sr	Sampling rate. 1011 means 48kHz.
0010 1000	**0000 0000****	dai_clk_mode	Digital audio interface clock (directions etc.)
0010 1001	**deduce yourself**	dai_ctrl	Digital audio interface controller
0110 1001	0000 1000	dac_l_ctrl	Don't mute nor filter output audio (left channel)
0110 1010	0000 0000	dac_r_ctrl	Don't mute nor filter output audio (right channel)
0100 0111	1110 0001	cp_ctrl	Charge pump enable
0110 1011	0000 1001	hp_l_ctrl	Activate headphone jack's left output channel
0110 1100	0000 1000	hp_r_ctrl	Activate headphone jack's right output channel
0100 1011	0000 1000	mixout_l_select	Drive output from DAC* (left channel)
0100 1100	0000 1000	mixout_r_select	Drive output from DAC* (right channel)
0110 1110	1000 1000	mixout_l_ctrl	Enable mixer amplifier (left channel)
0110 1111	1000 1000	mixout_r_ctrl	Enable mixer amplifier (right channel)
0101 0001	1111 0001	system_modes_output	Activate all the selected output sub-systems

* DAC = Digital-analog converter

There are two missing parameters for you to find out. **To find out the missing parameters, search (ctrl-F!) for registers *DAI_CLK_MODE*** and *DAI_CTRL*.** They should be somewhere between pages 60 and 80.

(Note that the search function might break for longer names if they span multiple lines in the document (e.g. dai_clk_mode), so search wisely.)

Digital audio interface control register

****: This parameter was originally meant to be deduced by the student, but because of possible problems with the automatic tests, it is now given beforehand. You can see it in the table above (the bit sequence matches the settings below).**

The settings for the Digital audio interface were left for your own consideration. This is one of the registers that have quite a lot to do with your previously implemented audio controller block (exercises 7 and 8).

Below are the desired settings. They are set up in DAI_CLK_MODE and DAI_CTRL registers. All the corresponding bits for every setting can be found in the datasheet on the pages about these registers.

DAI_CLK_MODE

Parameter	Setting
DAI master mode enable	slave mode
DAI word clock polarity	normal polarity
DAI bit clock polarity	normal
BCLK number per WCLK period	32

DAI_CTRL

Parameter	Setting
DAI enable	enabled
DAI output enable	DATOUT pin high impedance
DAI TDM mode enable	DAI normal mode
DAI mono mode enable	DAI stereo mode
DAI data word length	16 bits per channel
DAI data format	left justified mode

Parameter status

The purpose of `param_status_out` is to function as debug information which tells the configuration phase. The signals of `param_status_out` are connected to leds on the PYNQ board.
If we had enough leds on the PYNQ board for all the parameters (total of 15), we could use one led for one parameter. We don't, but luckily we have 4 leds, which is enough to represent an unsigned value of max 15).

One easy way to keep track of the status is a signal of type `unsigned` (width `n_leds_g (=4)`) and output this signal to `param_status_out`. Add +1 to this signal every time you have succesfully sent one parameter. In the end it should show 15 (when synthesized on the board in later exercise, all the leds on the board should be lit).

When all the configuration data is succesfully sent, the signal `finished_out` is also raised up. Until then, it remains down. This signal can be used for starting the operation of the rest of the project work.

Other requirements

- Compile using command `vcom -check_synthesis`. The check_synthesis parameter gives warnings about errors which would cause a difference between simulation and synthesis results. It is not recommended to use it with test benches as they are not synthesized.

Return:

Obtain the files specified below and put them under E11 folder in your Git repository. Return them also by file return in the bottom of this page.

- Files to return:
  - `answer11.txt`
  - `fsm.png`

- In `answer11.txt`:
  - A small plan of your block in file
    - Brief textual description of your processes
    - The missing configuration values from the table and where you found them

- In `fsm.png`:
  - State diagram of the state machine as a figure (`fsm.png`) which you can draw the way you want

- The code is not returned yet, only the answers to the questions

- The code is returned in Exercise 12 (https://plus.tuni.fi/comp.ce.240/spring-2024/E12/tb_i2c_config/)

answer11.txt

Choose File No file chosen

fsm.png

Choose File No file chosen

Submit with David Rama Jimeno Submit

Earned points

1 / 1

Exercise info

Assignment category
VHDL exercises

Your submissions
1 / 1000

Points required to pass
1

Deadline
Sunday, 14 April 2024, 23:59

Late submission deadline
Friday, 31 May 2024, 23:59

Group size
1-2

Total number of submitters
55

« 14. Exercise 11 (/comp.ce.240/spring-202… 15. Exercise 12 » (/comp.ce.240/spring-202…