FPGA notes

Nicola Zaupa

July 13, 2023

Contents

Info about the setup:

a. BOARD: DE4 Stratix IV Development BOARD

b. CORE: Stratix IV GX

c. CHIP: EP4SGX230KF40C2

d. LANGUAGE: Verilog

1 Installing

Some notes about installing the necessary softwares.

LICENSE

1. go to `https://licensing.intel.com/psg/s/` and access with the credential already linked to the device.
2. go to *Computers and License Files* to create a new license (licenses are linked to the PC and also to the ethernet network (MAC address)
3. add a new Computer (*New*)
    - Computer name: right click on `start` and then *System* → *Device name*
    - Computer Type: NIC ID, go to Command Line in Win "`cmd`" and type `ipconfig/all`. Find the `Physical Adress` of the interface (connection) that you are using (you also find the host name) [probably other interfaces on the same PC can be added]
4. go to *Licenses-All Licenses* and choose the one related to the product you need to activate
5. `Generate License` → *Assign an Existing Computer* [you should be able to rehost... otherwise you're ...]
6. Choose the computer
7.

2 Coding

I code in Verilog. Now testing the coding in Visual Studio Code.

Type of signal:

- `wire` can used with `assign`: `assign a = b & c`
- `reg`: values can be assigned with procedural blocks (`always, initial` : <=) but not with continuous (`assign`). Because `reg` is capable of storing and does not require to be driven continuously.

Initialize a variable

- at declaration: `reg [31:0] data = 32'hdead_cafe;`
- in `initial begin`: `data = 32'hdead_cafe;`
- can also initialize `wire`

Avoid to do it both, since is not sure which is executed first. Underscores '_' are not counted as bytes, their are useful to separate long numbers.

Could be interesting `assign-deassign` and `force-release`.

Structure for the loop `always`

Operators

`&&`	`&`	and	`*`	multi	`===`	equal, including x and z
`||`	`|`	or	`/`	divided	`!==`	not equal, including x and z
`!`	~	not	`%`	modulo	`==`	equal, can be unknown
	∧	xor	`**`	power	`!=`	not equal, can be unknown

Shift

Logical shift	`<<`	`>>`	add zeros
Arithmetic shift	`<<<`	`>>>`	keep the (sign?) bit (not sure about `<<<`)

```
data = 00101   →   data << 1  = 01010
                   data << 2  = 10100
                   data <<< 1 = 01011
```

Conditional operator: `assign out = cond ? true : false`
Concatenation: use the curly brackets $\{\ \}$ → $\{a, b, c\,[1:0]\,, 2'b00, \{2\{a\}\}\}$
Replication: $\{y, y, y\}$ = $\{3\{y\}\}$, it can be nested.

3 Simulation

Describe how to setup a simulation in Modelsim (Questasim)
→ is possible to generate a file "**.do*" to lunch for the simulation (should simplify the steps).

4 Quartus

- go to the RTL[1] view
- assign the PIN (use csv file to define the connection)

NEW PROJECT

1. File `->` New Project Wizard
2. Select/Create the folder for the project
3. Give a name to the project and select/create the TOP file of the code
4. Select `Empty Project`
5. In case select existing files (blocks), then `Next`
6. Select the device on the board
7. If necessary link simulation programs or others
8. Finish

PIN ASSIGNEMENT

1. *Assignments*
2. *Import Assignment*
3. Select the `*.csv` file, then *Ok*

Press the button play to compile the code
LOAD the CODE

1. Open the Programmer [small image]
2. Press on `Hardware Setup`
3. On *Currently selected hardware* select `USB-Blaster [USB-0]`. If not present maybe is necessary to update the driver of the USB blaster (on your PC go to device-Select the USB-Update Driver-Make windows search in the installation folder of Quartus)
4. Press `Start` to load the code [A blue led should be ON on the board]
5.

POSSIBLE PROBLEMS
Error: "`Error: Can't open project -- you do not have permission to write to all the files or create new files in the project's database directory`" → look at `https://www.intel.com/content/www/us/en/support/programmable/articles/000084612.html`

5 Data flow and manipulation

i_C and v_C are measured through the ADC. Chain of amplification:

1. analog electronics (attenuation) to match ADC requirements
2. ADC (from analog to digital)

[1]Register Transfer Level

6 Program blocks

6.1 TOP block

Is the block that is directly interfaced with the hardware. It defines the connection with the output PINs, and manage the main code. Then the code can take advantages of blocks in order to simplify it by making it more clearer and simplifying the debugging.

6.2 debounce

```
module debounce (
    output o_switch,
    input i_clk,
    input i_reset,
    input i_switch
);
```

Given an single bit input signal, the output switch to the input signal only when the input is stable for a certain amount of time. The time is defined trough a counter, hence depends on the counter limit and on the clock that activates the counter.

6.3 ABS

Compute the absolute value of a number.

6.4 dead_time

Introduce a delay on a rising edge of a signal.

6.5 hybrid_control

Implement the control to sustain the self oscillation.

- hybrid_control_theta – frequency modulation
- hybrid_control_phi – amplitude modulation
- hybrid_control_theta_phi – combination of amplitude and frequency modulation

6.6 LPF

Implement a digital Low Pass Filter on a signal. Important parameter is the number of samples that are considered. It is a moving average filter.

6.7 peak_detector

Detects a peak on a signal by computing the two points derivative.

6.8 PI

Block that implement the PI controller. Take the error as input and give the correction as output.

6.9 trigonometry

Compute sine and cosine of an angle in degree(?)

6.10 PLL

Manage the clock generation in the FPGA. Script generated by Quartus.

6.11 angle_control

Need to be designed, now there are two separate version, one for θ and one for φ.

The idea is to create a single block that has as inputs

- button signal
- upper limit
- lower limit
- step magnitude
- reset button

7 PIN mapping of the board

Maybe some figures on the PIN mapping that could be useful to have at hand.