Management and analysis of physics datasets, Part. 1

Final Project

Stefano Pavinato
14/1/2019

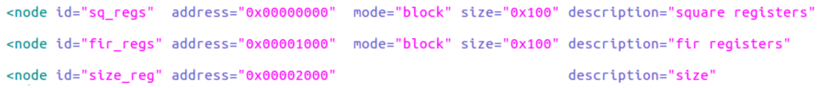
Outline

Outline

Goals

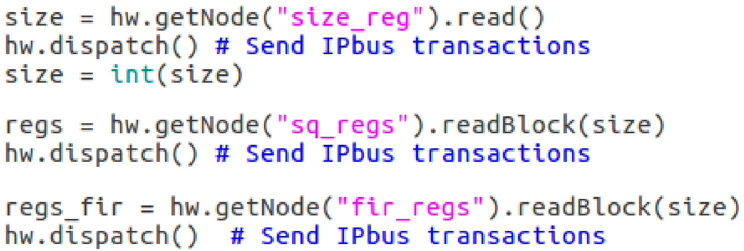
1. Read the two first locations of the SPI flash memory, then concatenate the data of the two registers. The concatenated data give the size value (software side) to read the following registers blocks.
2. Generate a square wave with period and duty-cycle configurable. The samples generated are stored in the dpram *sq_regs*.
3. Read the data stored in the dpram *sq_regs*, apply at them a fir filter and write the data filtered in the dpram *fir_regs*.
4. Finally the python script reads the last two dpram (*sq_regs* and *fir_regs*) and plots the data generated by the square wave generator and data filtered in time and frequency domain.

Memory mapped resources



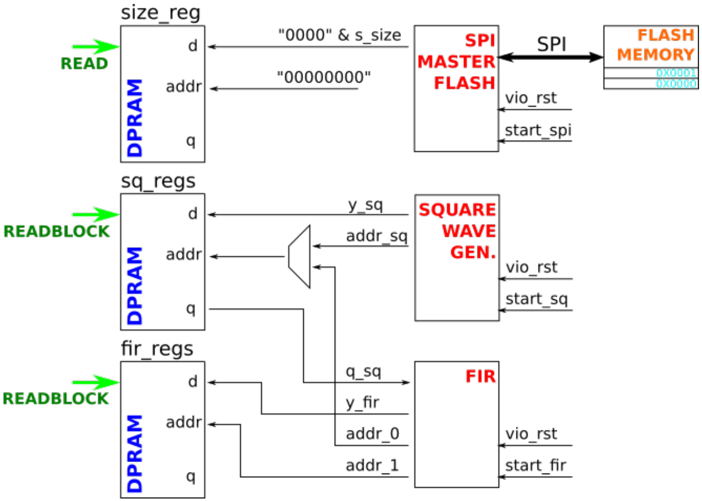
```
<node id="sq_regs"  address="0x00000000"  mode="block" size="0x100" description="square registers"
<node id="fir_regs" address="0x00001000"  mode="block" size="0x100" description="fir registers"
<node id="size_reg" address="0x00002000"                            description="size"
```

Python script

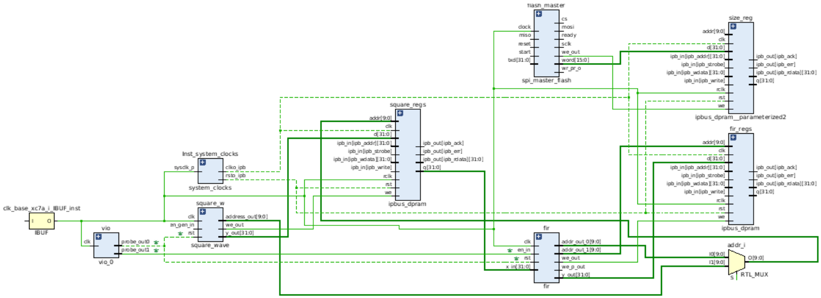


```python
size = hw.getNode("size_reg").read()
hw.dispatch() # Send IPbus transactions
size = int(size)

regs = hw.getNode("sq_regs").readBlock(size)
hw.dispatch() # Send IPbus transactions

regs_fir = hw.getNode("fir_regs").readBlock(size)
hw.dispatch()  # Send IPbus transactions
```

Firmware blocks



Firmware blocks - synthesis



Outline

What to do?

- **No** xml.
- **No** python. In python if you want, you can ONLY change the code to make the png file.
- **Just** VHDL. You have to write/modify the files that implement the spi flash reader, the square wave generator and the fir processing. Obviously also VIO and ILA cores are necessary. Indispensable is at least a VIO core.

SPI Master Flash (1)

- The SPI Master Flash implemented in the file *spi_master_flash.vhd* allows to read a number *NBYTES* of flash memory registers and write their contents in a dual port RAM.
- The flash memory is set with the same mcs file given in the last laboratory.
- You have to modify the FSM in the file *spi_master_flash.vhd* in order to read the first two registers, concatenate the data of these two registers and then write it in the dual port ram `size_reg`.
- According with the mcs file in the first two locations of the flash memory are written:
  - @ address 0x000000 the value 0x00;
  - @ address 0x000001 the value 0x01.

SPI Master Flash (2)

- The data concatenated is $0x0100$, that is 256.
- This value in the *top_level.vhd* file is represented by the signal *s_size*, a 16 bit signal.
- The value is then retrieve by the read function in python and so assigned to the variable `size`.

Square Wave Generator (1)

- The square wave generator has to be implemented in the file *square_wave.vhd*.
- The output of the generator is a std_logic_vector.
- The square wave has two logic levels:
  - low level: $y <= std\_logic\_vector(to\_signed(-1024, y'length));$
  - high level: $y <= std\_logic\_vector(to\_signed(1024, y'length));$
- It generates *SAMPLE_N* samples. The default *SAMPLE_N* value is 1024. You do not need to change it.
- After *SAMPLE_N* samples the FSM implementing the square wave generator goes in idle state.

Square Wave Generator (2)

- Each sample has to be stored in the dpram *sq_regs*.
- That is the first sample generated is stored at the dpram address $0x0000$, the second sample at the address $0x0001$, the third at $0x0002$ and the 1024th sample at the address $0x3fff$.
- The first 256 (variable `size` in python) values stored in this dpram are read with the script python using the `readblock` function.
- The samples generated by the square wave generator, in python, are represented by the variable `regs`.

Square Wave Generator (3)

- The generator is configurable with the parameters (*generics*):
  1. PERIOD. The period of the square wave is $\mathrm{PERIOD} * T_{clock}$. With T_{clock}, as usual, 10 ns;
  2. DUTY_CYC. It is the duty cycle of the square wave. The square wave is at the high logic level for a time equals to $\mathrm{PERIOD} * \frac{\mathrm{DUTY\_CYC}}{100} * T_{clock}$. Else it is at low level. DUTY_CYC is an integer with values from 0 to 100.
- You can use as model what you made for the seventh laboratory.
- It can be implemented as a 3 state FSM. The states are labeled as: `s_idle`, `s_high` and `s_low`.

FIR

- The fir filter has to be implemented in the file *fir.vhd*.
- You can reuse what you made in the fifth laboratory.
- You have to integrate it with a FSM in order to read the data from the dpram *sq_regs*.
- With the same FSM after the signal processing, the data "filtered" has to be written in the dpram *fir_regs*.
- The first 256 (variable `size` in python) value stored in *fir_regs* dpram are read with the script python using the `readblock` function.
- In python the data stored in the *fir_regs* dpram are represented by the variable `regs_fir`.

MUX

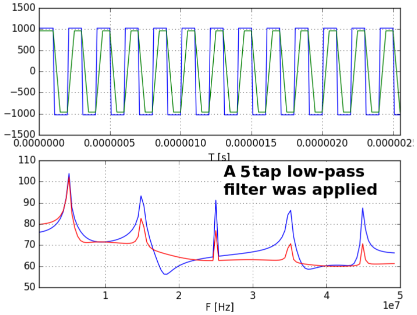
- You see that the dpram *sq_regs*, it is used to write the samples generated by the square wave generator and it is used by the fir component as input port.
- So the ram memory implemented into the file *dpram.vhd* has to be addressed by the square wave generator when it writes its generated samples and by the fir filter to retrieve its input.
- So the address input port of the dpram component has to be shared between the two design logic components in different moments.
- How do you share the input address port of the dpram, in this case?
- The more straightforward way is a mux 2-1.

VIO

- In the slide 7 there is the block schema representation of what you have to implement.
- You see that each design logic block as a reset signal and a start signal.
- These signals are driven with a VIO core.
- The reset is the same for each block.
- Regarding on the start signals there are two policies:
  1. Three start signals provided, with the right sequence, by the VIO core.
  2. One start signal and then a FSM (in the *top_level.vhd* file) that after the SPI flash memory reading starts with the square wave generation. After the generation of the square wave starts with the fir processing.
- You can choose what policy you prefer.

Final result

After you have implemented the three blocks, if you run the python script given, you have to get a picture like this:



The .bit and .ltx file generated with the default values (*generics*) are given. This picture was obtained with this bitstream.

Outline

Homework

- **This homework is for the final exam.**
- You download all the folder *final_project/firmware*, open the project (the file .xpr) and work here.
- You have to implement the three design logic blocks.
- At the final exam you have to bring with you a report.
- Bring with you also the files .bit and .ltx. Possibly we can check live the working of your project.

Report (1)

In the report there must be:

- The fir filter you applied. You can copy the picture produced by the python script *coeff.py*
- The picture produced by the python script (*final_project.py*) that you find in the *software* folder with:
  - PERIOD = 20, DUTY_CYC = 50 (the default values);
  - PERIOD = 10, DUTY_CYC = 50;
  - PERIOD = 20, DUTY_CYC = 30;
- The VIO policy you chose.

Report (2)

In the report there must be also the following pieces of code:

- what you modified in the FSM in the file *spi_master_flash.vhd*. Not the whole FSM;
- the mux implementation, indicating the three input signals;
- the FSM implemented in the fir architecture;
- the state `s_low` of the FSM used to implement the square wave generator.