Week 10

Objective : Make the Ipcore working for sure

Revisit the instructions :

Data section (32 bits)	instruction code (4 bits)	Signification
empty	0000	Does nothing just shift out the shift reg value
address (32 bits)	0001	Write the bus start address
byte_enable (4 bits)	0010	Write the byte enable reg
Size (32 bits)	0011	Write the burst_size of the transaction in number of words
	0100	Put the bus start address in the shift register
	0101	Put the byte enable reg in the shift register
	0110	Put the burst_size in the shift register
	0111	
data to send	1000	Write the data in the next place ping-pong buffer increassing the block size by one

Data section (32 bits)	instruction code (4 bits)	Signification
empty	1001	Put next word in the ping-pong buffer in the shift buffer decrease block size by one
empty	1010	Launch the write operation
empty	1011	launch the read operation
empty	1100	Wait for result of DMA and switch buffer
empty	1111	reset the registers

The ipcore

THe ipcore now has multiple registers:

- shift register: where the data is shifted in
- address_reg: store the DMA start address
- burst_size_reg: store the burst size that want to be used
- Byte_enable_reg: storet the byte enable that will be used by default it is 0xF
- shadow_reg: the next value the ipcore will shift out
- updated_data_reg: the value of the shift register the last time JUPDATE was one
- data_reg: store the result of the read buffer

Write register instructions

The instructions to write the configuration registers have one clock cycle delay. But this should not cause any issue since to shift in the next instruction we need 37 clock cycles hence the registers will be ready before executing the next instruction.

Read register instructions

This new version offers 3 instructions to read the config registers. THos instructions are a bit special. The role of those instructions are actually to put the value we want to read in the shift register so that the next time we shift in

data the instruction register will be shifted out and we can read it. Hence this instruction actually have a 1 instruction delay.

```
> irscan ecp5.tap 0x32
// Fill the registers
...
// Read the address buffer
> drscan ecp5.tap 37 0x4
<garbage value>
// shift out actual value
> drscan ecp5.tap 37 0x0
<content of address reg>
```

Using this technique we can simply read all the register by asking iteratively and it will work:

```
> irscan ecp5.tap 0x32
// Fill the registers
...
// put the address buffer in the shift reg
> drscan ecp5.tap 37 0x4
<garbage value>
// shift out actual value and put the byte_enable in the shift reg
> drscan ecp5.tap 37 0x5
<content of address reg>
// shift out actual value and put the burst_size in the shift reg
> drscan ecp5.tap 37 0x6
<content of byte_enable>
// shift out actual value
> drscan ecp5.tap 37 0x0
<content of burst_size>
```

Write to the buffer

THe 0x8 instruction write in the buffer. It will write incrementally unil the buffer is full. Once full this instruction will not do anything more. This instrcuction has 2 clock cycle delay before the data is actually puted in the buffer but same for the write register instruction it will not impact the system since we need 37 clock cycles to run another instruction.

Read the buffer

The 0x9 instruction read the buffer from the address 0x0 and increment after each read until it reaches the block size expected. This instruction works the same as the read register in a way that it simply put the data in the shift register and to actually read it we need to wait for the next instruction.

This time the ipcore needs 3 clock cycle to actually read the buffer

and put it in the shift register hence it nedds 3 clock cycle before sending another instruction

Reset instruction

the 0xf instruction will simply reset all the buffer of the ipcore.

Launch the DMA

In order to luanch a new instruction to the DMA those steps are followed implemented using a FSM:

1. Wait for the DMA to be not busy
2. Switch the buffer to retreive the value written by the DMA and get the corresponding block size
3. Launch the actual DMA instruction with signals such as DMA_launch_read, or DMA_launch_write
4. End the transaction to set back the signals to zero

This instruction needs at least 4 clock cycles before actaully launching th DMA and 6 to correctly reset all signals

The DMA can read the registers since they are mapped to its input signals.

Example of a write

Here we assume the DMA to be ready

```
// Select the 0x32 chain and set up the registers
// Write in the buffer
> drscan ecp5.tap 37 0x123456788
0000000007
> drscan ecp5.tap 37 0x12345678
0000000007
> drscan ecp5.tap 37 0x1234568
0000000007
> drscan ecp5.tap 37 0x123458
0000000007
> drscan ecp5.tap 37 0x12348
0000000007
> drscan ecp5.tap 37 0x1238
0000000007
> drscan ecp5.tap 37 0x128
0000000007
// Launch the write instruction
> drscan ecp5.tap 37 0xA
0000000007
> runtest 1
```

```
> runtest 1
> runtest 1
> runtest 1
> runtest 1
> runtest 1
```

Example of a read

Here we assume the DMA to be ready

```
// Select the 0x32 chain and set up the registers
// Launch the read
> drscan ecp5.tap 37 0xB
0000000007
> runtest 1
> runtest 1
> runtest 1
> runtest 1
> runtest 1
> runtest 1
// Switch the buffer when DMA is done
> drscan ecp5.tap 37 0xC
// Read the buffer
> drscan ecp5.tap 37 0x9
0000000007
> runtest 3
> drscan ecp5.tap 37 0x0
0012345678
```

Switch the buffer

I also added an instruction that allow to switch the buffer when the DMA is ready to switch. It simply follow the path of launching a new instruction but does not send any starting signal to the DMA.

Synchronisation

add the two synchronizers for both of the signal to launch the DMA

Question

1. What would be intersing to put in the status reg of the ipcore (operation running, status of the DMA)?
2. Differrence between the design and the implementation in the report ?
3. Should i explain the design before this week and why did we decided to change it, what about the presentation ?

4. In the report in the background is it meaning full to add all the tools i used ?