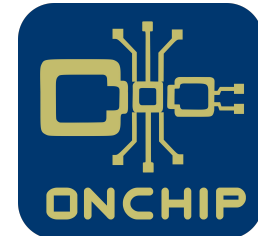


Register Access Controller Architecture For ASIC's

Hanssel Morales

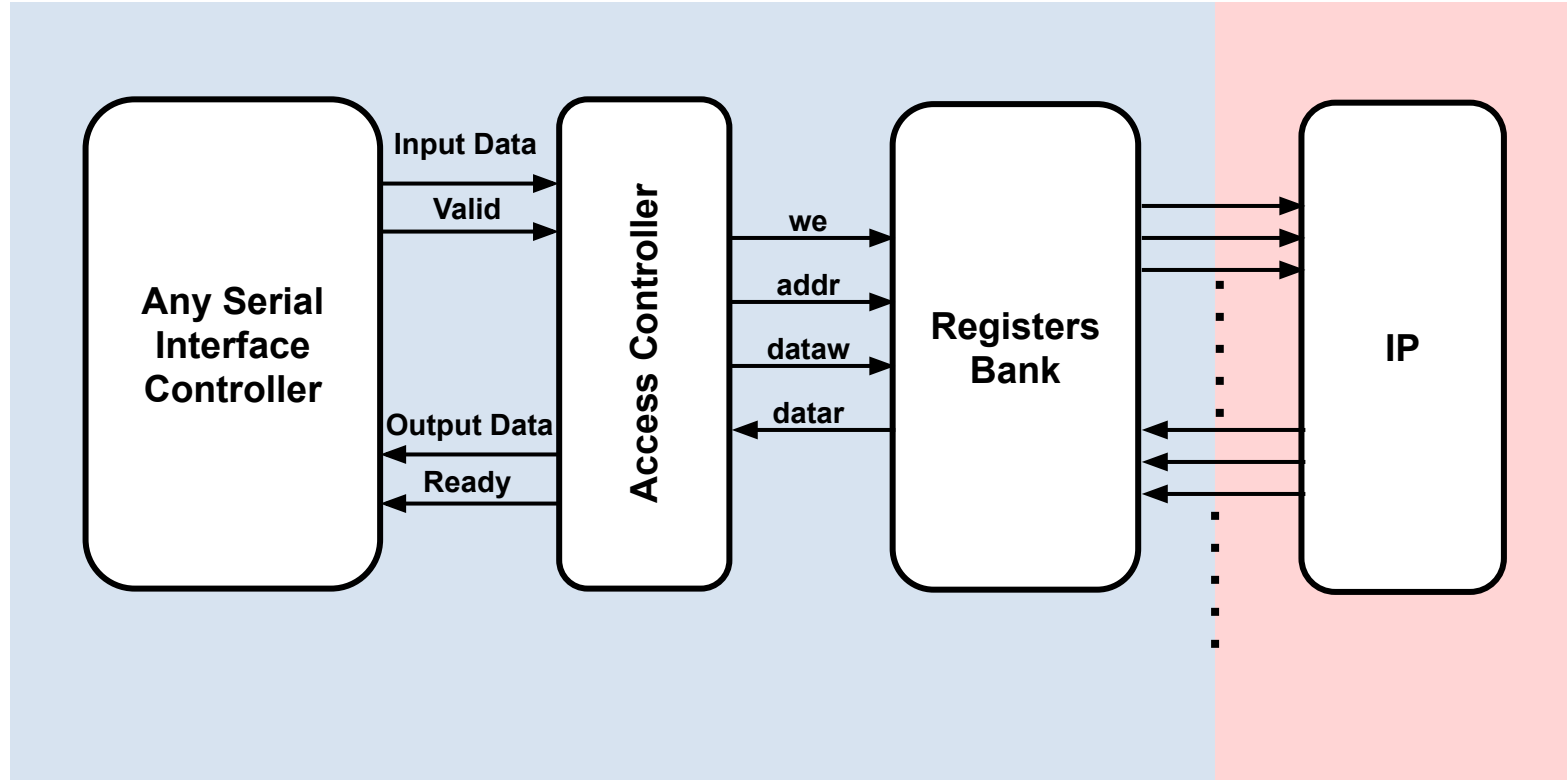
Integrated Systems Research Group – OnChip
Universidad Industrial de Santander, Bucaramanga - Colombia



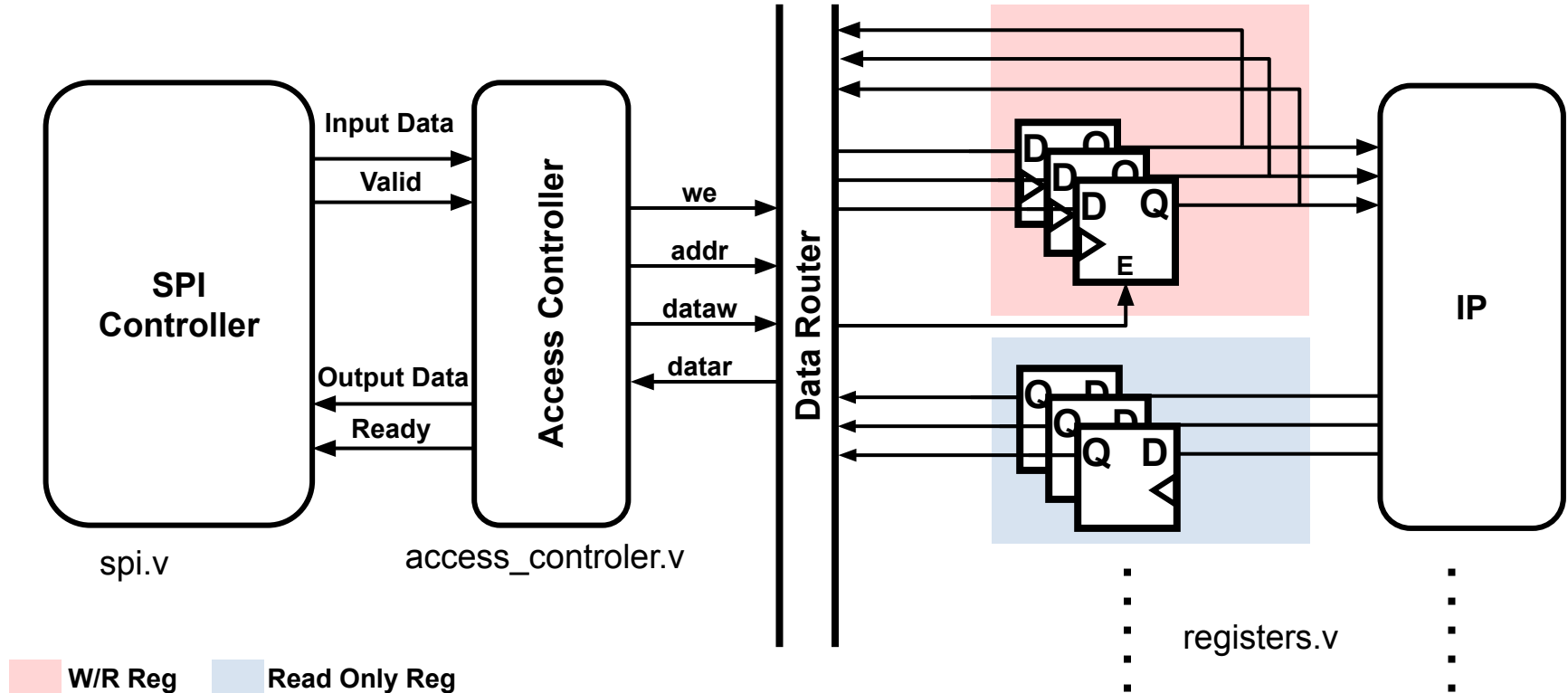
Outline

- Implemented Architecture
- Register Access Flow
- Command Line Use
- Scripting Capabilities
- FPGA Prototype and C232h SPI Standard Accomplishment

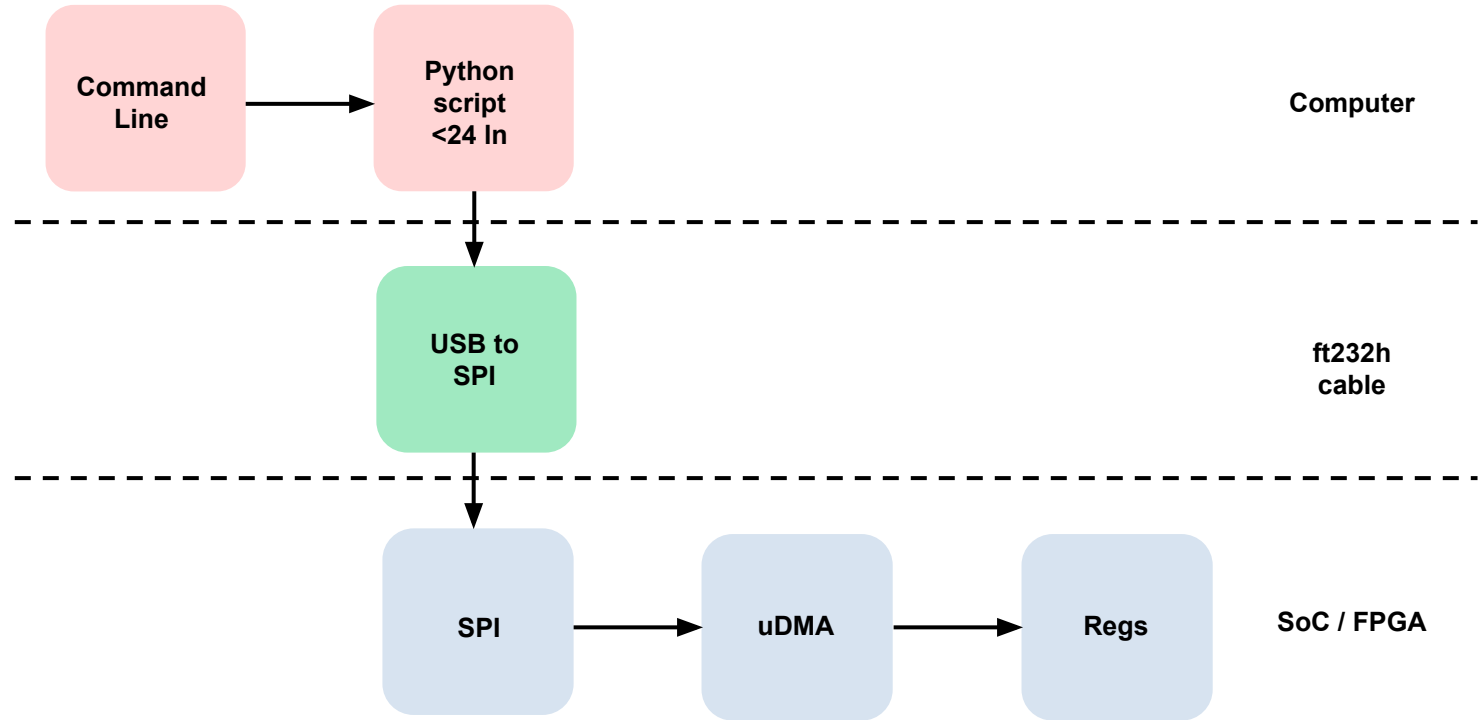
Implemented Architecture



Amazilia Implemented Architecture



Register Access Flow



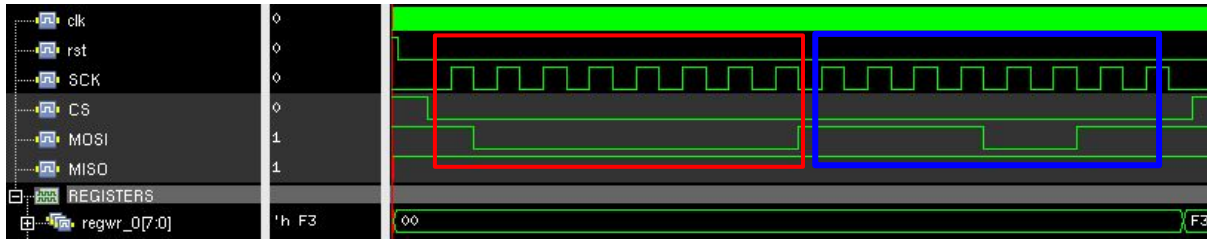
Command Line Use

Single Read and Single Write Transactions been used:

```
askartos@askartos-Lenovo-Y520-15IKBN:~$ sudo ./ftdiControl.py r 0x0
0x2f Open Target
Its working
askartos@askartos-Lenovo-Y520-15IKBN:~$ sudo ./ftdiControl.py w 0x0 0x3c
Its working
askartos@askartos-Lenovo-Y520-15IKBN:~$ sudo ./ftdiControl.py r 0x0
0x3c
Its working
```

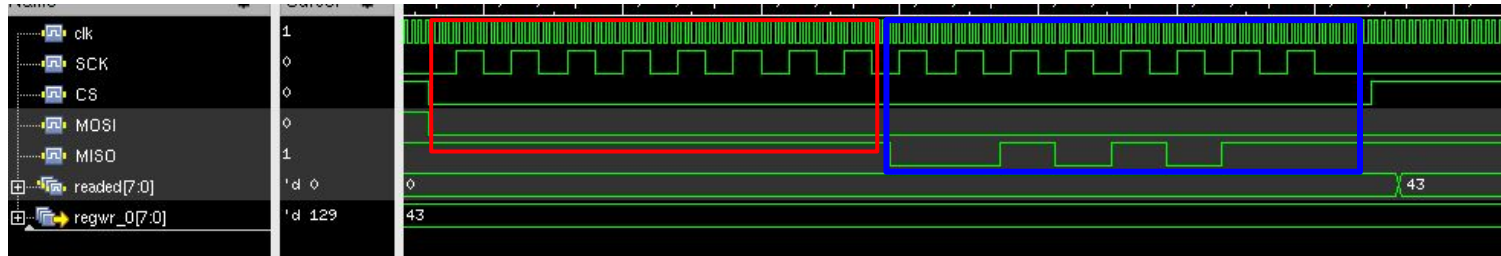
Single Write Transaction Wave Form

Example of a write transaction over address 0 :



Single Read Transaction Wave Form

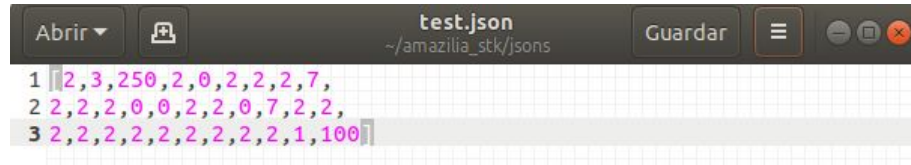
Example of a read transaction over address 0 :



Command Line Use

Burst transaction been used :

```
askartos@askartos-Lenovo-Y520-15IKBN:~$ sudo ./ftdiControl.py b test.json
21: Aplicaciones Google Keep Google Calendar Sci-Hub: remo... (2) WhatsA
Its working
askartos@askartos-Lenovo-Y520-15IKBN:~$
```

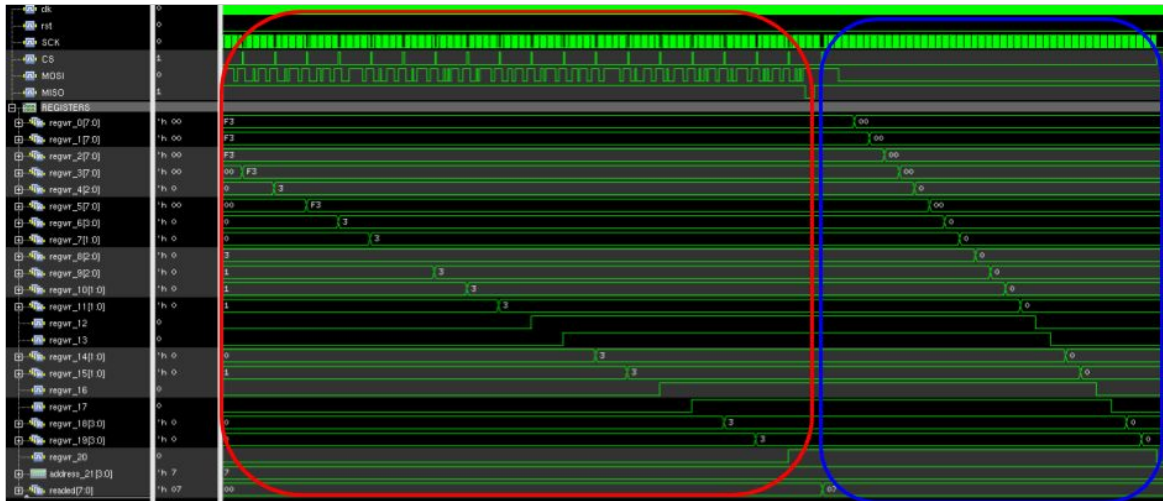


The screenshot shows a text editor window titled "test.json" with the path "~/amazilia_stk/jsons". The editor contains three lines of JSON data, each representing a burst transaction. The first line is highlighted in blue, the second in yellow, and the third in green. The data is as follows:

```
1 [2,3,250,2,0,2,2,2,7,
2 2,2,2,0,0,2,2,0,7,2,2,
3 2,2,2,2,2,2,2,2,2,1,100]
```

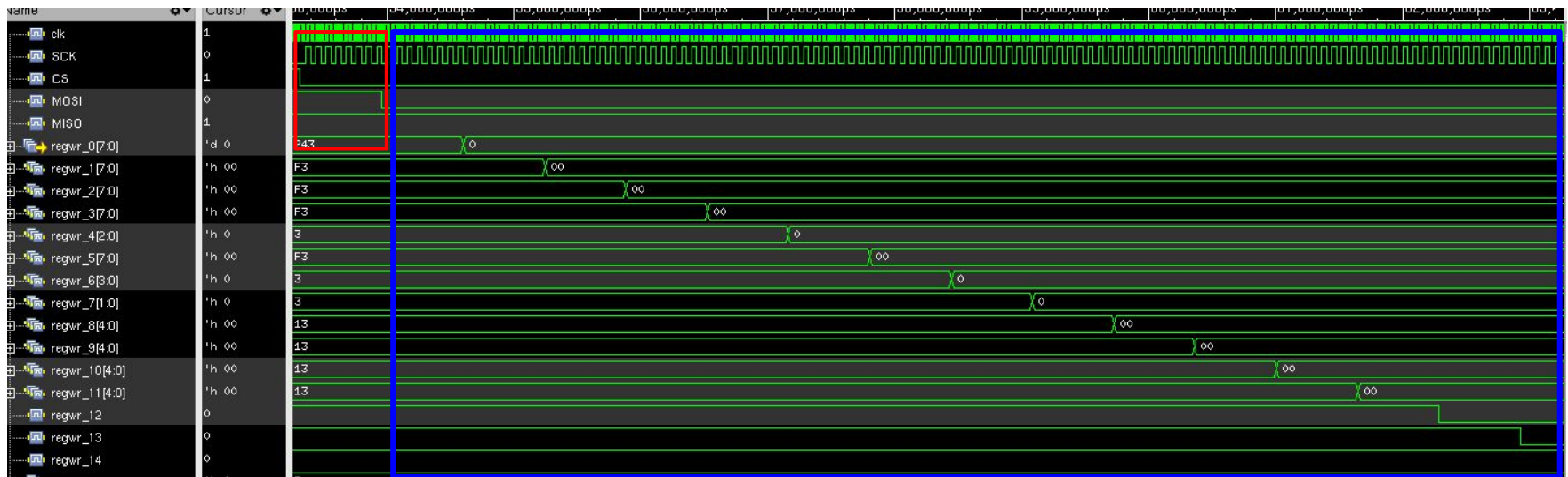
Burst Write Transaction

Burst transaction compared with multiple write transactions:



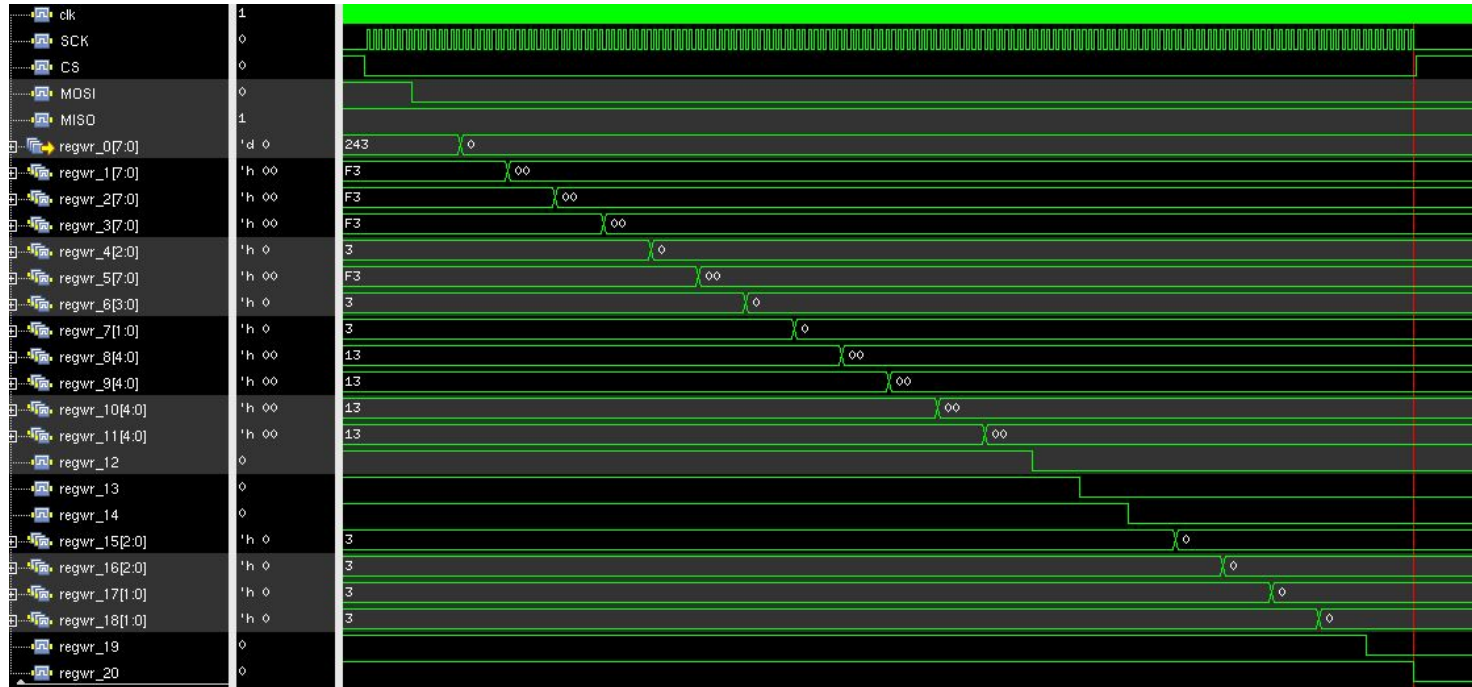
Burst Write Transaction

Burst transaction stages:



Burst Write Transaction

Burst transaction stages:



Command Line Use

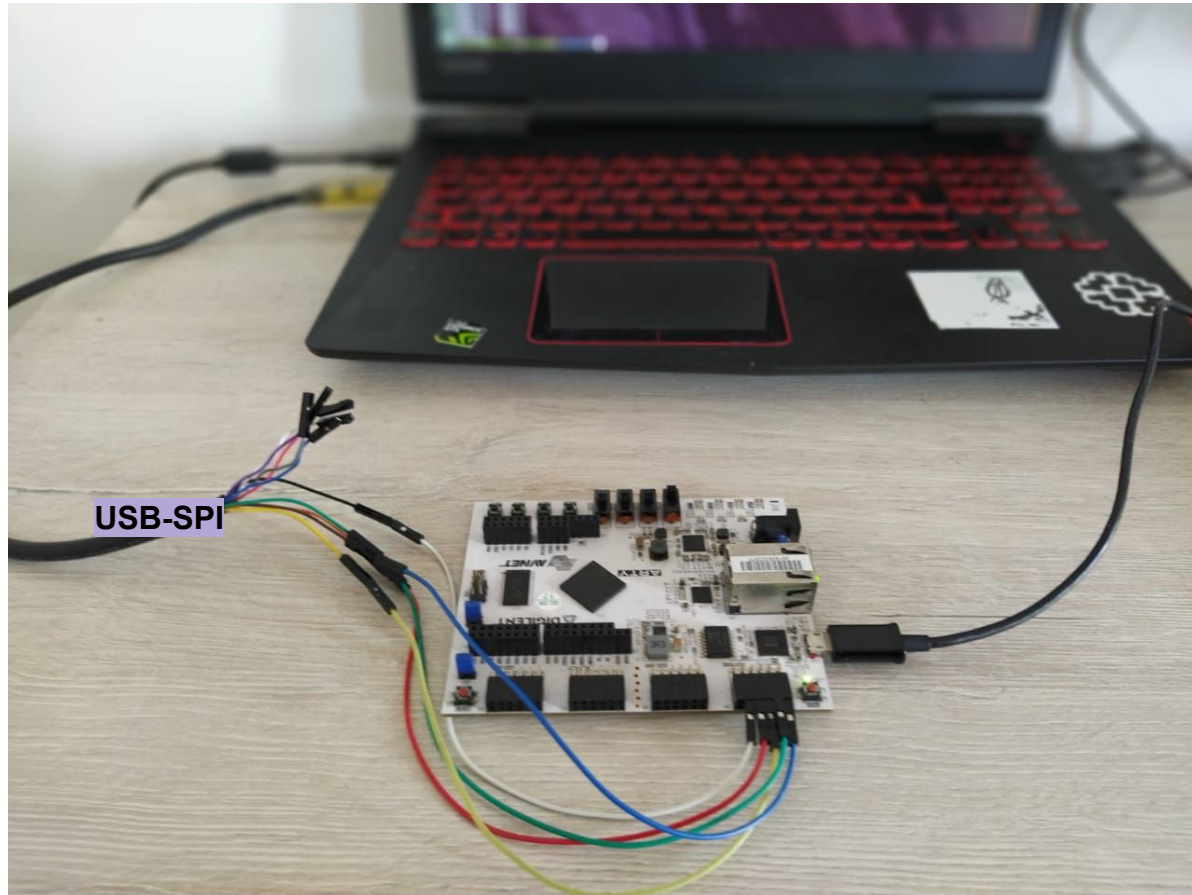
Multiple read software implemented transaction :

```
askartos@askartos-Lenovo-Y520-15IKBN:~$ sudo ./rttControl.py ra 0x0 0x15
addr: 0x0      data: 0x2
addr: 0x1      data: 0x3
addr: 0x2      data: 0xfa
addr: 0x3      data: 0x2
addr: 0x4      data: 0x0
addr: 0x5      data: 0x2
addr: 0x6      data: 0x2
addr: 0x7      data: 0x2
addr: 0x8      data: 0x7
addr: 0x9      data: 0x2
addr: 0xa      data: 0x2
addr: 0xb      data: 0x2
addr: 0xc      data: 0x0
addr: 0xd      data: 0x0
addr: 0xe      data: 0x2
addr: 0xf      data: 0x2
addr: 0x10     data: 0x0
addr: 0x11     data: 0x0
addr: 0x12     data: 0x2
addr: 0x13     data: 0x2
addr: 0x14     data: 0x0
addr: 0x15     data: 0x7
Its working
askartos@askartos-Lenovo-Y520-15IKBN:~$
```

Scripting Capabilities

```
1 #!/usr/bin/python3
2 import amazilia
3
4
5
6 print(amazilia.read(0)) #READS ADDRESS 0
7
8 amazilia.write(0,243)    #WRITES ADDRESS 0
9
10 print(amazilia.read(0)) #READS ADDRESS 0
11
12
13 print("old data")
14 amazilia.multiread(0,31) #READS COMPLETE MEMORY MAP
15
16 data=[20,3,11,2,10,2,2,2,11,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,100]
17
18 amazilia.burst(data)      #WRITE COMPLETE MEMORY MAP
19
20 print("new data")
21 amazilia.multiread(0,31) #READS COMPLETE MEMORY MAP
```

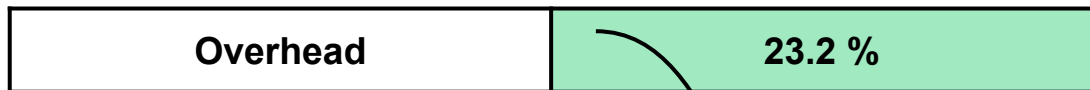
FPGA Prototype



RAC WC Synthesis Report 180nm

Registers Bank (115 registers)	10774 $[(\mu\text{m})^2]$
Total RAC	14036 $[(\mu\text{m})^2]$ eq 108.5 μm x 108.5 μm
Overhead	23.2 %
Frequency Target	333 MHz

WC Synthesis Report

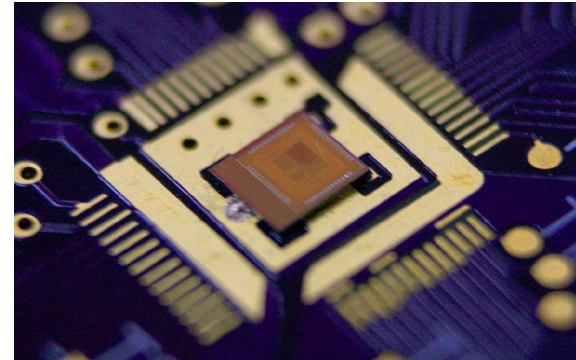
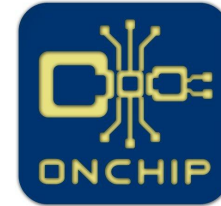
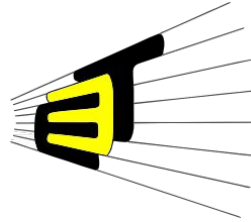


Take into account that you are only using 115 registers and this system its able to bring access with the same circuitry to 1024 registers

WC Area Report 180nm

Registers Bank (115 registers)	10774 $[(\mu\text{m})^2]$
Total RAC	14036 $[(\mu\text{m})^2]$ eq 108.5 μm x 108.5 μm
Chip Percentage (1.6mm x 1.6mm)	0.54 %
Frequency Target	333 MHz

Thanks! Questions?

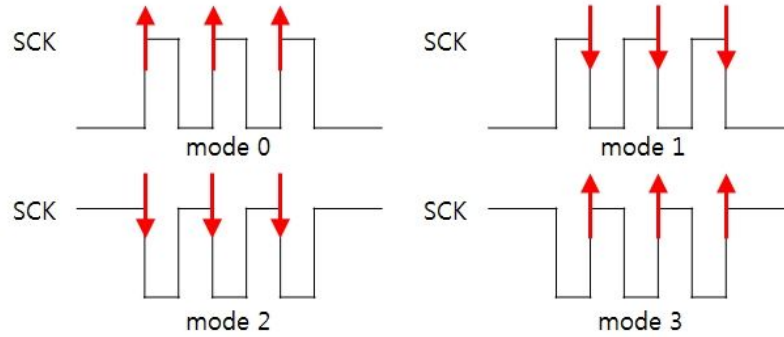


hanssel.morales@correo.uis.edu.co
onchip@uis.edu.co



[@onchipUIS](https://twitter.com/onchipUIS)

SPI MODES



Register Access Controller

