Validation and verification

of real-time data processing system in programmable devices for Digital J-PET scanner DAQ system

3rd Symposium on Positron Emission Tomography

Cracow, 2018

Karol Farbaniec

European Funds Smart Growth
Republic of Poland
FNP Foundation for Polish Science
European Union European Regional Development Fund

This work is supported by the Foundation for Polish Science under Grant TEAM/2017-4/39

Validation and verification

"Validation. The **assurance** that a product, service, or system meets the **needs of the customer and other identified stakeholders**..."

"Verification. The **evaluation** of whether or not a product, service, or system **complies with a regulation, requirement, specification, or imposed condition**..."

Validation of developed system

- Key functionality assumptions based on previous project iterations
  - continuous triggered readout
  - modularity and parametrization of subsystems
- Bottlenecks candidates
  - resources
  - throughput
- Next steps in further development...

Verification in reference to programmable devices

- Data source emulation
- HLS (High Level Synthesis) simulation environment for implemented modules
- Complementary HDL simulation

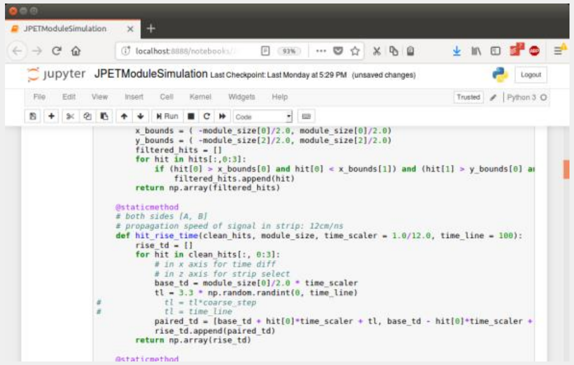
- Testing and evaluating design on hardware

Data source emulation

Digital J-PET scanner module data generator in python.

Why python? Why Jupyter notebook?

- Jupyter: No hardware setup or configuration needed. Work and further development can be easily done online in web browser
- Python is high-level programming language, so creation of utility applications is <u>fast</u> and code is <u>brief and legible</u>.
- Python contains <u>scientific libraries</u>, so we can easily <u>extend and integrate</u> current application with <u>complex mathematical models</u>

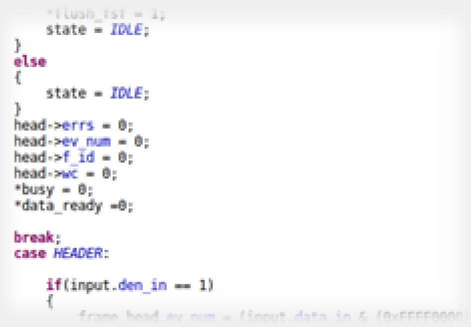

JPETModuleSimulation
localhost:8888/notebooks/
93%
jupyter JPETModuleSimulation Last Checkpoint: Last Monday at 5:29 PM (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
        x_bounds = ( -module_size[0]/2.0, module_size[0]/2.0)
        y_bounds = ( -module_size[2]/2.0, module_size[2]/2.0)
        filtered_hits = []
        for hit in hits[:,0:3]:
            if (hit[0] > x_bounds[0] and hit[0] < x_bounds[1]) and (hit[1] > y_bounds[0] a
                filtered_hits.append(hit)
        return np.array(filtered_hits)

    @staticmethod
    # both sides [A, B]
    # propagation speed of signal in strip: 12cm/ns
    def hit_rise_time(clean_hits, module_size, time_scaler = 1.0/12.0, time_line = 100):
        rise_td = []
        for hit in clean_hits[:, 0:3]:
            # in x axis for time diff
            # in z axis for strip select
            base_td = module_size[0]/2.0 * time_scaler
            tl = 3.3 * np.random.randint(0, time_line)
#             tl = tl*coarse_step
#             tl = time_line
            paired_td = [base_td + hit[0]*time_scaler + tl, base_td - hit[0]*time_scaler +
            rise_td.append(paired_td)
        return np.array(rise_td)

    @staticmethod
```

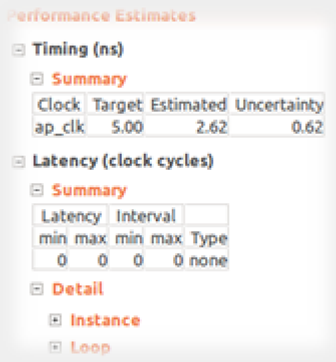
HLS simulation environment for designed modules

FPGA subsystems prototyping and testing in Vivado HLS IDE.

- HLS programming language abstraction enables implementation and rough debug on main functionality faster than traditional Hardware Description languages
- HLS enables agile design optimizations with pragmas
- Software algorithms migration to programmable logic with HLS is easier because of same abstraction level implementation (C/C++)



```
        *flush_fsf = 1;
        state = IDLE;
}
else
{
        state = IDLE;
}
head->errs = 0;
head->ev_num = 0;
head->f_id = 0;
head->wc = 0;
*busy = 0;
*data_ready =0;

break;
case HEADER:

        if(input.den_in == 1)
        {
                frame_head.ev_num = (input.data_in & (0xFFFF0000)
```



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00	2.62	0.62

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
0	0	0	0	none

Detail

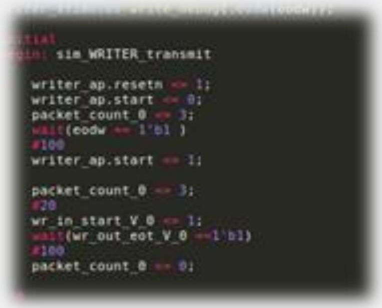
Instance

Loop

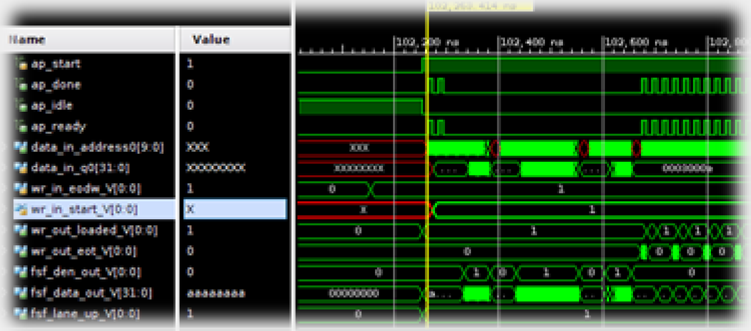
Complementary HDL simulation

HDL testbench and transactions simulation

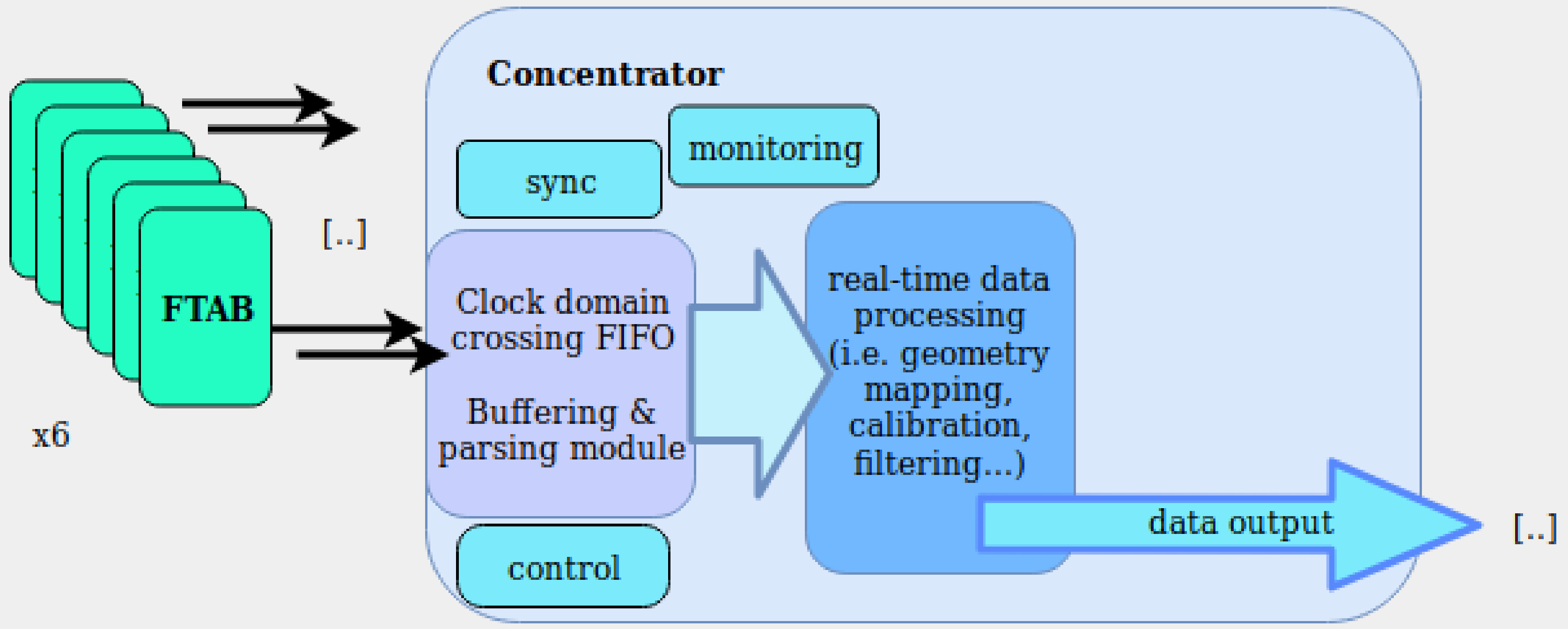
- Examination of low level signals in intergration testing and subsystems simulation
- SystemVerilog based verification for HLS modules – testing on low-level abstraction
- AXI transactions as freuqent method for in hardware FPGA testing, monitoring, slow-control and debug. Common interface in proposed system design



```
itial
gin: sim_WRITER_transmit

  writer_ap.resetn <= 1;
  writer_ap.start <= 0;
  packet_count_0 <= 3;
  wait(eodw == 1'b1 )
  #100
  writer_ap.start <= 1;

  packet_count_0 <= 3;
  #20
  wr_in_start_V_0 <= 1;
  wait(wr_out_eot_V_0 ==1'b1)
  #100
  packet_count_0 <= 0;
```

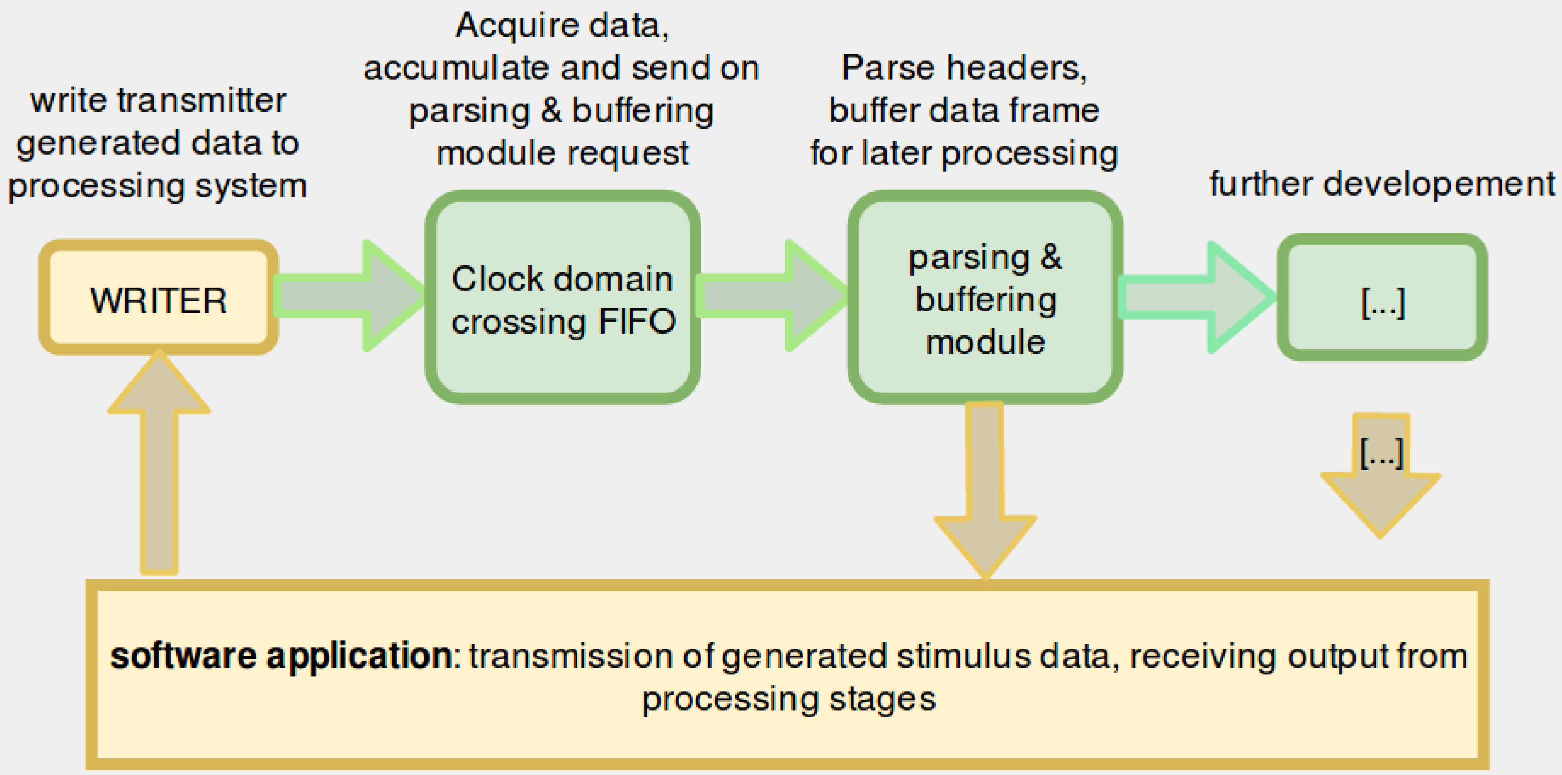


Data processing system – conceptual design



FTAB
x6
[..]
Concentrator
sync
monitoring
Clock domain crossing FIFO
Buffering & parsing module
real-time data processing (i.e. geometry mapping, calibration, filtering...)
data output
control
[..]

Testing and design evaluation in hardware


write transmitter generated data to processing system
Acquire data, accumulate and send on parsing & buffering module request
Parse headers, buffer data frame for later processing
further developement
WRITER
Clock domain crossing FIFO
parsing & buffering module
[...]
[...]
software application: transmission of generated stimulus data, receiving output from processing stages

Summary & conclusions

- Suitable verification enviroment enables faster developement of modules
- Variety use of dedicated tools, software and programming languages is essential in complex HW/SW systems developement
- Scripts, applications and designs reuse its a nice to have approach while working on R&D

- Parser and interfacing tests ongoing
- Module-wise coincidence finder as a next step