Title
Post-Silicon Hardware Validation of a Many-Core System

Author
Tsoi, Wai Cheong

Publication Date
2022

Peer reviewed|Thesis/dissertation

Post-Silicon Hardware Validation of a Many-Core System

By

WAI CHEONG TSOI
THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

_____

Bevan Baas, Chair

_____

Venkatesh Akella

_____

Hussain Al-Asaad

Committee in Charge

2022

Acknowledgements

I would like to express my deepest gratitude to Professor Bevan Baas for his guidance and support throughout my undergraduate and master's study. He introduced me into the field of VLSI digital design and research. Despite all the challenges and setbacks, he led me through all the obstacles and taught me persistence and perseverance. His insight has led me through this journey.

I am grateful to Professor Venkatesh Akella and Professor Hussain Al-Asaad for their teaching and advising, and serving in my thesis committee. I learnt a lot regarding digital design, computer architecture and digital testing in their classes. I cherish their feedback regarding my thesis.

I am thankful to Brent Bohnenstiehl and Timothy Andreas as their contribution and continuous support is what made this research possible. They designed the foundation of the Kilocore2 chip and the companion hardware for the chip, and provided continuous support after their graduation.

Special thanks to everyone in the VLSI Computation Lab, especially Satyabrata Sarangi, Shifu Wu, Ziyuan Dong and Yikai Mao for their help in lab works. It is my pleasure to work with you all.

Lastly, I'd like to recognize everyone that supported me along the way.

Abstract

Post-Silicon validation is a fundamental step in integrated circuits fabrication, and designing the setup is an essential step to the process. The setup requires a lot of design effort, especially under the specification and constraint of a many-core processor, which has a large state space and requires high speed and bandwidth of data communication. In this paper, a custom test station design is proposed for the Kilocore2 many-core processor. It consists of a Linux test station that controls the instruments via GPIB, and communicates with an FPGA via UART and PCIe that act as the test driver and monitor for the DUT. Tests are performed using this setup and the results are presented. This setup shall establish the foundation for future validation efforts of the Kilocore2 chip.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

Post-Silicon Validation is a fundamental step in integrated circuits fabrication. After the behavior of the RTL design is verified in pre-silicon verification, the design is synthesized and implemented into physical design floorplans, which could introduce potential design rule faults or missed timing faults that are not reported by the CAD tools. The fabrication process might also introduce faults due to defects. The faults introduced in these two steps calls for a validation process to ensure the behavior of the chip fabricated matches with specification.

The setup for Post-Silicon Validation requires a lot of design effort, each with their own tradeoffs. The solution to interface the DUT with, the instruments to drive the high-speed data signals to the DUT and streaming data out of, the solution to connect the DUT to the instruments to conduct the high-speed data signals, and the test station to control the instruments – all requires consideration to match the specifications of the DUT. After confirming that the options match with the specifications, the choices should balance between cost, ease of use, future expandability, and various other factors.

1.2 Thesis Organization

This thesis is organized as follows.

Chapter 2 introduces the concept of many-core processor, the background of post-silicon validation, and the background and architecture of our DUT, the Kilocore2. Chapter 3 describes the requirements and specifications of a test station used for post-silicon validation.

Chapter 4 explains the composition of our custom test station and the design considerations into creating this test station.

Chapter 5 details the design of the FPGA board used as part of the test station to interface with the control and data ports of our DUT.

Chapter 6 details the design of the Python setup as part of the test station to control the instruments interfacing with our DUT.

Chapter 7 highlights the tests performed using our setup and the results of these tests.

Chapter 8 provides some ideas for future work.

2. Background

2.1 Many-core Processor

A many-core processor is a processor array with many physical cores that provides high degree of parallelism. The processor cores on a many-core processor typically runs at a lower frequency than conventional processor cores to save power, or even turned off when unused. Due to the number of cores, the array often consists of more complicated interconnect networks to communicate with internal memory and between cores. The number of possible states of the processor increases exponentially with respect to the number of cores in the processor, as each core would interact with each other on a physical level that cannot be decoupled. The validation effort thus also scales with the number of possible states, with test automation essential to effectively verify design of such complexity.

2.2 AsAP Architecture

The asynchronous array of simple processors (AsAP) architecture focuses on a large amount of processor cores that can run in high clock frequency and are globally asynchronous locally synchronous (GALS) to implement hardware task-level parallel data streams at low power. The goals of AsAP are to target workloads that can be parallelized into simple tasks, and would be benefited by high throughput and low power, which applies to most digital signal processing (DSP) tasks, such as video streaming and telecommunication. The distinct features of this architecture is that the processing elements are simplistic in design such that it can be clocked at higher frequency to achieve high throughput; each processing element is paired with a packet router that forms a two-dimensional mesh grid that are clocked separate to the

processing element, allowing the clock of the core to halt if the block is only routing traffic; and completely asynchronous clocking as each core has its own clock oscillator, which eliminates global clock trees routed to potentially a thousand processor elements [1]-[3].

2.3 Post-Silicon Validation

Post-silicon validation points to the validation process after the chip is fabricated, as opposed to pre-silicon verification, the verification process before the chip is fabricated, usually in computer simulations. And unlike pre-silicon verification, the chip needs to be physically powered on and input vectors need to be set by a signal generator, and the output vectors are often streamed to a checker. The debugging process is also different as, with the scale of modern silicon transistors, it is very costly and potentially destructive process to probe the internal signals, as compared to viewing the waveform of a simulation. There is also more potential fault points as the physical design of the chip is also into question, where the placement and routing of the transistors or register-transfer level to gate level synthesis discrepancy could be the root cause of a fault. This leads to extensive analysis to be conducted when a fault is found, and tests to be designed to discover the root-cause of the problem.

2.3.1 Wafer Test

There are two stages of post-silicon validation: wafer test and package test. Wafer test is conducted before the wafer is diced and packaged. As the wafer is a slab of silicon that is fragile and easily contaminated, the testing of it is done in clean room with special instruments. The wafer is placed on a wafer chuck that uses vacuum to

hold the wafer in place, and mechanical probes are placed on the contact pads to conduct testing [4].

2.3.2 Package Test

Packaging refers to the process where after the wafer is diced, the die is embedded in a case known as a package, and its input and output contacts are routed to pins for the ease of soldering the chip to a printed circuit board (PCB) or installing the chip onto a socket. Package test, hence, refers to the testing after the packaging process. Packaging tests are often conducted using automatic test equipment (ATE) that connects to an integrated circuit socket or PCB to interface with the chip being the device-under-test (DUT). As packaging of IC varies widely in non-commercial products [5], custom PCB designs typically offer more freedom and capabilities from IC socket connections, as extra features such as power-on self-test can be added to the DUT board [6]. Power rails with decoupling capacitors and extra sense lines can be added to the DUT board to reduce noises between power lines while providing extra entry points for testing. High speed signals can be physically isolated on the board to minimize noise coupling, and custom parallel data ports to be connected to signal generators and logic analyzers as part of the ATE for testing. If feasible, it is possible to include programmable logic, such as an FPGA to perform some versions of built-in self-test (BIST) or batch programming through the configuration data bus for register configurations. It could also act as the return path for loopback testing to verify the ATE setup, which is also very valuable during debugging.

2.4 Kilocore and Kilocore2 Chip

The VLSI Computation Laboratory at University of California, Davis designed and fabricated a many-core processor named Kilocore, which is the third generation of AsAP. It consists of 1000 RISC-based processing elements, each communicates with its neighbor in a 2D mesh network via packet routers. It contains all features and characteristics enabled by the AsAP architecture mentioned above [8]. Its successor, Kilocore2, would be the fourth generation of AsAP [9], and the DUT of this research.

The Kilocore2 chip contains similar amount processor cores and packet routers, DSP accelerators, and memory modules as the Kilocore. And just like the Kilocore, each processor core inside this chip is paired with a packet router, and creates a two-dimensional mesh network to efficiently route data between cores. A glue module act as a bridge between the chip's configuration input and the processor array, and houses the temperature-voltage sensing (TVS) module, configuration packet assemblers and disassemblers, configuration acknowledgement logic and test signal generation and selection logic. External clocks can also be supplied as the clock signal for specific processor cores or packet routers. The packet routers are routed by a hardware-implemented route shaping algorithm. Three levels of voltage, VHI, VMED and VLOW can be selected to each processor core for more options in voltage control [9], [10].

3. Test Station

A test station setup provides the platform to validate the DUT that establishes high-speed connections to the DUT, creates test agents that initiate the tests and check the results, and preferably able to be remotely accessed. One solution would be purchasing an automatic test equipment (ATE) that shall handle all initial bring-up efforts and provide the test agents and high-speed connections to the DUT. The ATE architecture consists of a master controller, often a computer, to control instruments that are responsible for fundamental operations, such as power supplies, multimeters and oscilloscopes needed to verify the chip is powered-on properly and internal clocks are operational, and to control instruments that facilitates feature testing, such as pulse generators and logic analyzers needed to send configuration commands and input vectors and stream output vectors from the DUT.

While this solution offers the ease and comfort that enables the chip validation team to concentrate their efforts on test generation, it often shifts the setup design effort to a third-party vendor like Teradyne and National Instruments with a cost. With the complexity of the design of our chip and its custom socket design, it is likely that it requires a lot of design effort, which in turn incur a huge cost. Thus, it is more feasible to design the validation setup in house.

4. Custom Test Station Design

Besides cost, building the validation setup in-house also allow the implementation of extra features that provides efficiency or convenience during debugging. The output vectors can be processed into readable debug messages or values, such as floating-point numbers representation from a bitstring. Scoreboard checkers can be added to compare the output vectors to a golden reference, and only print out error messages to declutter the interface, and further automate the validation process. Allowing remote access also helps the validation process as physical access might not be feasible in some situations, as demonstrated by a global pandemic.

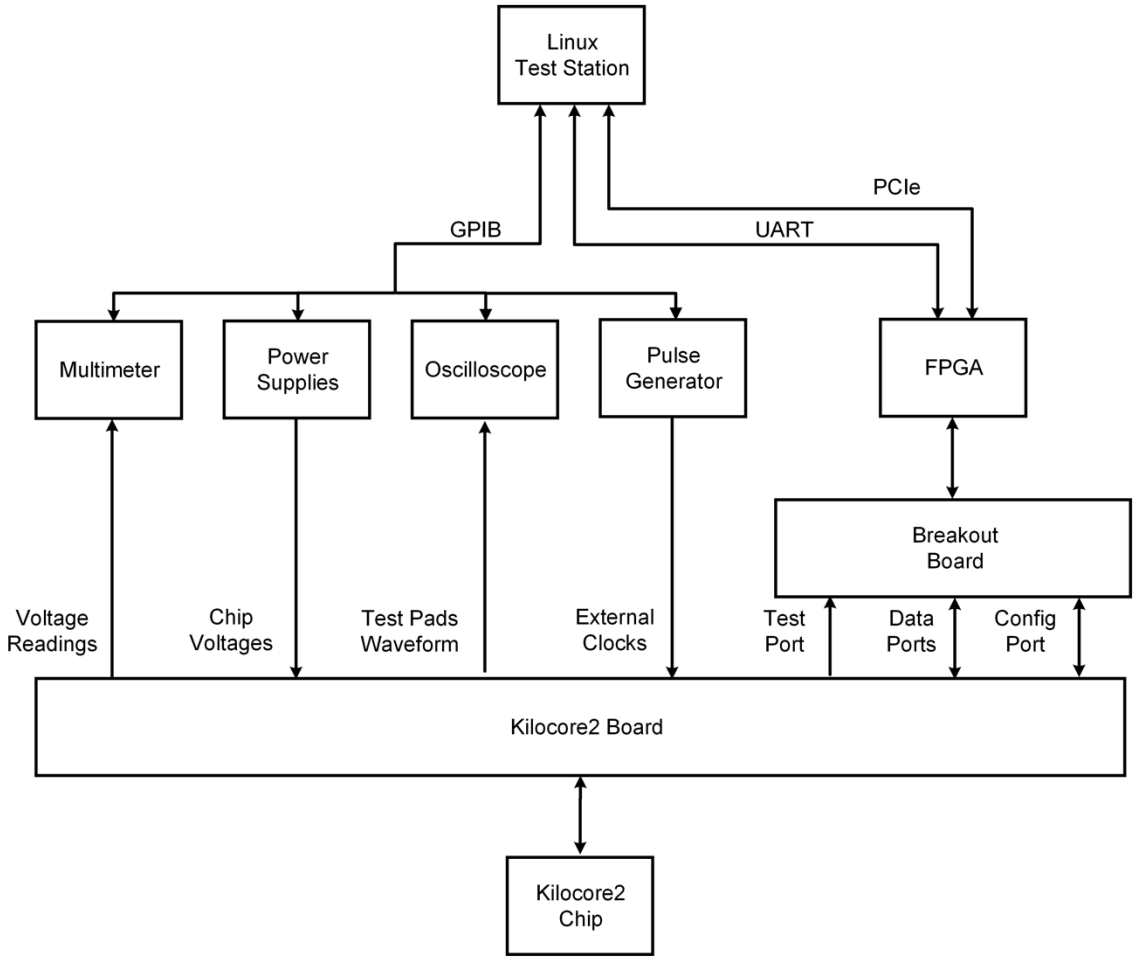

Figure 1. Test Station Setup for Kilocore2 Chip

4.1 Linux-Based Operating System

Linux-based Fedora Linux is installed on our test station master controller. While CAD tools and adapter drivers are best supported on Windows as compared to Linux, Linux offers more control over various settings like file system and security that made Linux more feasible. For our purpose, the difference between different Linux distributions, such as Fedora Linux as compared to, for example, CentOS, is negligible as the test station does not utilize any features unique to any Linux distribution.

The server is installed with the CAD tools such as Vivado, ISE and iMPACT, which is required to synthesize our RTL designs, and thus create the bitstreams used to program the FPGA and the on-board CPLD used in our setup. Python is chosen as our programming interface for it is simple to program, and the language contains a lot of open-source libraries that can be used to simplify our design process. It also does not rely on compilation and can be launched from the terminal directly.

4.2 GPIB

GPIB is a common interface is utilized to communicate with instruments that are not impacted by the standard's low throughput, such as power supplies and multimeters. Compared to other standards like PCIe 3.0 x8 that has a bandwidth of almost 8000 MB/s, the GPIB interface has its maximum bandwidth of less than 2 MB/s [7], making it undesirable for high-speed data transfer. These instruments are only responsible during powering on and taking measurements that are not sensitive to timing. GPIB also allow daisy chaining instruments, where after programming its address, instead of each instrument establishing separate connections with the master controller, only one is required for all instruments, freeing up port space and reducing cost in USB/GPIB adapters. With GPIB, the chip can be powered up in a specific power

sequence with voltage and current limit in place and set specific voltages, all contained within one Python script.

Figure 2. Agilent 82357A USB/GPIB Interface

4.3. Power Supplies

Power supplies of lower current rating, such as the Agilent 6611C, are used for initial testing, as compared to power supplies that can reach maximum current rating of our DUT's power rails. This intrinsically provides protection against misconfiguration when debugging the setup. In our setup, the chip core voltage uses the Agilent E3633A that provides a higher current limit than other power rails to reap this benefit while delaying the time when future tests demand a higher current. DUTs that are sensitive to

voltage levels might be damaged when faults occur and the power supplies switches to current-limited mode and should disregard this.

4.4. Custom Kilocore2 PCB

To access the pins of our chip easier, the Kilocore2 chip is soldered onto a custom designed Kilocore2 PCB, such that the package pins are routed to various power terminals and high-speed parallel data ports. The PCB improves connectivity to the chip during testing, with separate port connectors as opposed to exposed pins on the chip. The board provides clear indication of which ports correspond to compared with bare pins on the packaging that is difficult to distinguish between. The board can be fitted with high-current rated power terminals with decoupling capacitors that provides stability to core voltages. Mounting holes can be added to the board for mounting a heat sink to cool the chip under high loads.

Dedicated ports for separate data bus provide better signal integrity, especially when two signal standards, LVDS and 1.8V CMOS, are both used in different data buses. Low Voltage Differential Signaling (LVDS) uses the difference in voltage as signal and thus by design is less susceptible to noise. Along with its lower voltage swing, it is anticipated to achieve higher data transfer rates than 1.8V CMOS. 1.8V CMOS, however, is still used in some data buses as expansion boards that utilizes these data ports might not be compatible with LVDS standard. The data ports are in a parallel configuration, as it provides higher bandwidth as opposed to serial, where digital signal processing uses would find beneficial. SMA connectors are added to provide low-noise external clock inputs.

Extra features can be added by the board's custom design. For the Kilocore2 board, by daisy chaining multiple boards' configuration ports, multiple chips can be

configured by only one host. This is achieved by assigning unique IDs to each board's CPLD and multiplexing the configuration bus depending on ID selection. Ports for external memory module are also included on the board for future expansion.

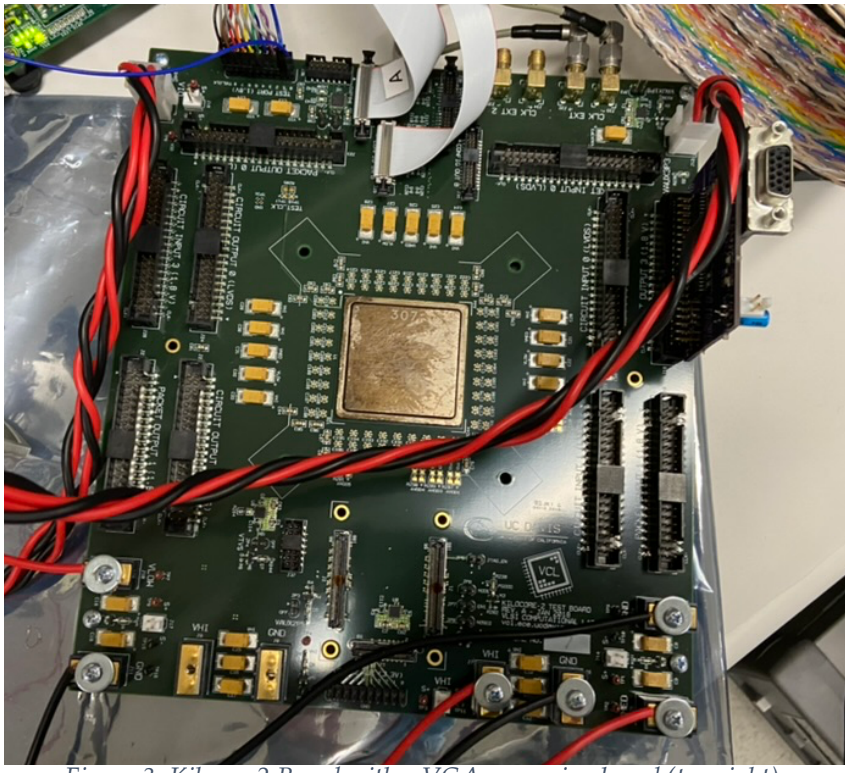

Figure 3. Kilocore2 Board with a VGA expansion board (top right)

4.5. Xilinx VC709 FPGA

To accommodate the high-speed signals and have more control over timing, instead of using multiple pulse generators and logic analyzers, a Xilinx VC709 FPGA will be sending input vectors to the board and receiving the output vectors streamed out of it.

FPGA provides a reprogrammable platform that is useful for design changes and adding features, as well as temporarily reroute signals for debugging. The FPGA board chosen includes a PCIe port for fast data transfer and a UART port for control and less time-sensitive data signals. Both communication protocols shall be used by the test

station and controlled by the Python interface. The FPGA also includes an FMC connector that provides the bandwidth for routing the input and output ports from the Kilocore2 board to the FPGA board.

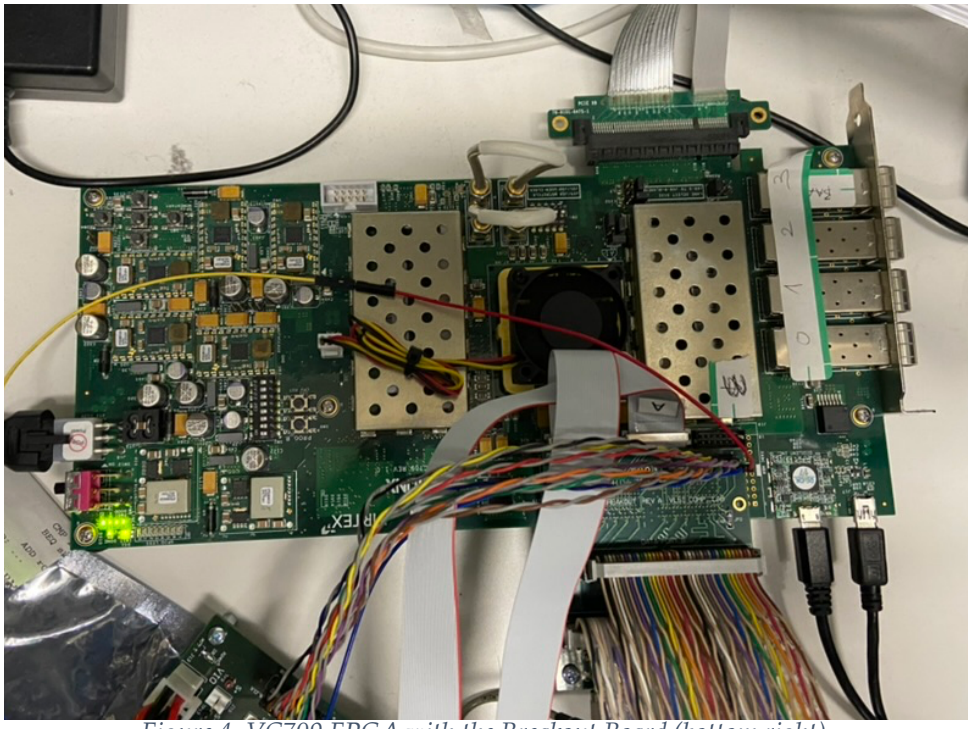

Figure 4. VC709 FPGA with the Breakout Board (bottom right)

4.6 Breakout Board

Yet, the FMC connector itself with its exposed pins are not ideal to interface with our various port standards on our Kilocore2 board. Hence, a breakout board would route the pins to matching ports to the Kilocore2 board. The breakout board contains the same configuration ports and test port as the Kilocore2 board, and only two data ports that can be programmed as either an input or output during synthesis. With the LVDS standard requiring a pair of pins per bit, the 400-pin FMC connector is not sufficient to contain all the pins used in an entire Kilocore2 ports. Two data ports were chosen as it was the bare minimum for one as input and one as output. And after verifying some of the data ports, those ports can be used to verify other data ports, or

by using a verified board, use its data ports to generate the test sequence for another DUT board.

Besides the basic connectivity, the breakout board also includes extra features that are useful for debugging. It adds two pairs of external clock outputs to be used to generate test clocks for the external clock inputs of the Kilocore2 board, providing a more convenient solution as compared to setting up the pulse generator through GPIB. The breakout board also features a 14-pin JTAG port that allows programming the CPLD through that instead of a Xilinx Platform Cable USB II cable.

Figure 5. Xilinx 14-Pin JTAG Cable

5. FPGA Design

The FPGA is responsible for decoding commands from the Linux test station, and set configuration signals and input buses to the Kilocore2 board accordingly, as well as receiving data from the Kilocore2 board and sending it back to the Linux test station, via UART or PCIe.

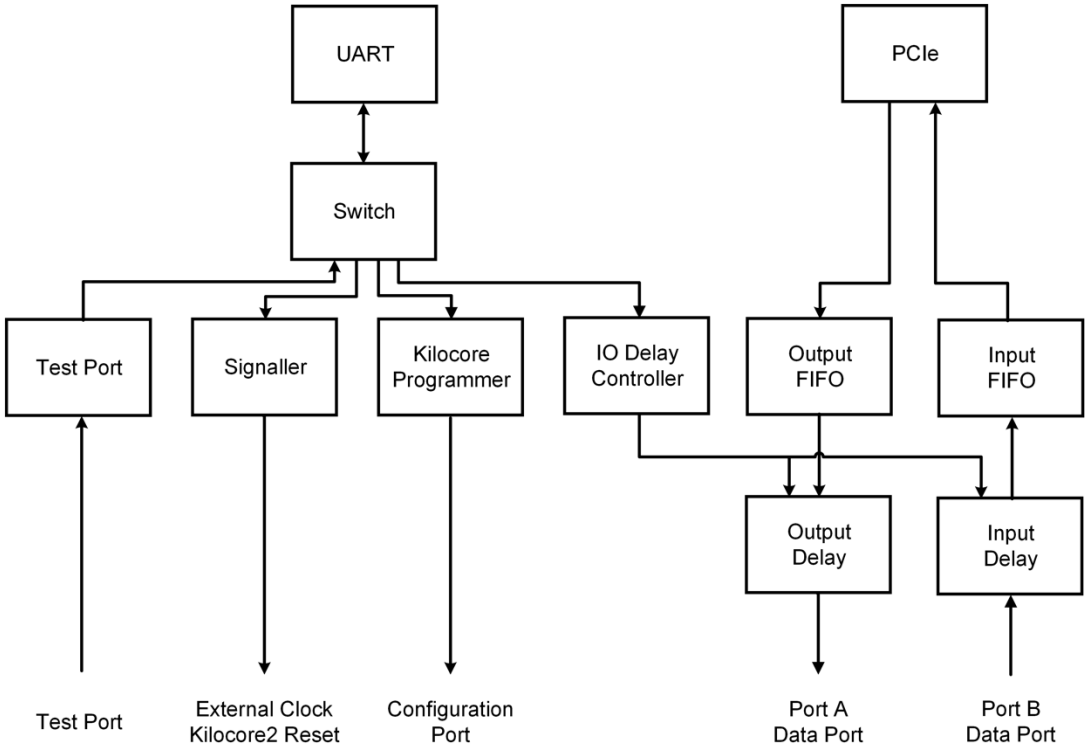

Figure 6. FPGA Top Module with PCIe

5.1. UART Interface

With its simplistic design, UART is used for sending configuration words to the FPGA. After receiving the bits from the serial bus of the FPGA, the bits are assembled to 32-bit words by a shift register and a counter, which is then sent to a FIFO. The FIFO is needed as the downstream block may backpressure this block, but the UART block has no mechanism to backpressure the Python host to halt. It is expected, however, that

with the serial nature of the input bus and the slow baud rate as compared to the FPGA clock, a buffer with a depth of 2 should be sufficient, which further saves FPGA space for other features. The datapath for sending words back to the Python host, however, should have an adequate FIFO depth, depending on the baud rate to the FPGA clock ratio, as the producer from the FPGA side has a significantly higher bandwidth than the consumer through the UART interface. This indicates that the bandwidth of UART is not sufficient for high-speed data transfer, and thus is not used for streaming data into and out of the Kilocore2, and PCIe is used instead.

As the 32-bit configuration words need to be sent to their corresponding modules, the first three bits are encoded as the destination of the word, which is decoded by the switch during routing.

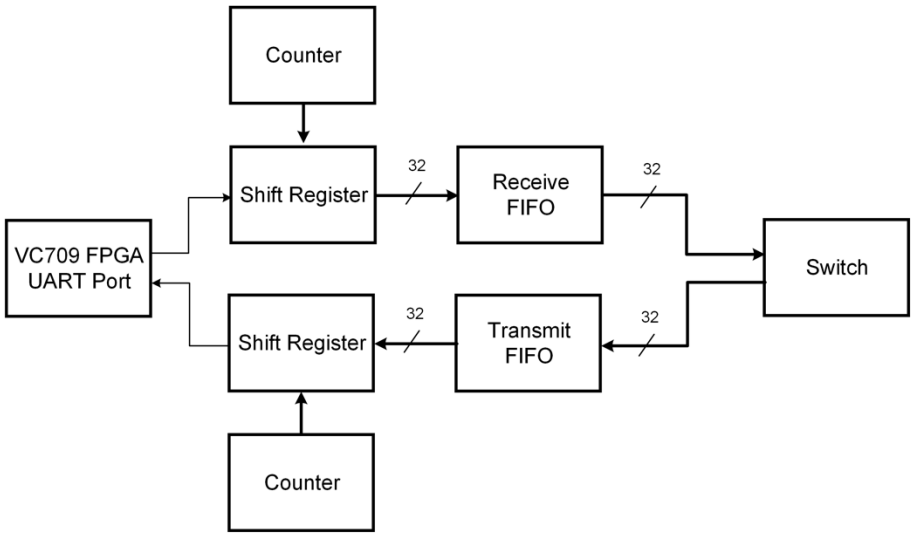

Figure 7. Datapath of UART Module

5.2. Switch

The switch module is responsible for sending data between modules inside the FPGA. To simplify the design, no FIFOs are implemented inside switch. Instead, the switch uses valid ready interface and backpressures the sender if the receiver is not

ready. In other words, the switch pairs the sender's valid with the receiver's ready, and routes the data words if the sender has valid data and the receiver is ready. This is uncommon as stalling at the switch might stall multiple blocks at once, and is to be avoided in most cases. In our case, however, all traffic either originates from or destinates to the UART module, which contains a FIFO in both directions. If the depth of the UART FIFO is sufficient, this guarantees that no traffic would be stalled by the switch. Along with the simplistic nature of other modules that can complete most transactions within one cycle, implementing FIFOs inside switch or other blocks appears to be unnecessary.

For blocks that send data back to the Python interface, a unique header ID is still assigned as it is used to differentiate between different senders. Therefore, some blocks that does not intend to receive any data, such as the test port module, still has a valid receive input port. While the Python interface does not allow sending data to these blocks, as these blocks never intend to accept any input data, to simplify the hardware design, it is possible to send data to it by setting the input ready signal to be always high, but promptly disregarded by leaving the input bus open. This preserves consistency and the simplistic design of the switch without causing any potential stalling in the datapath.

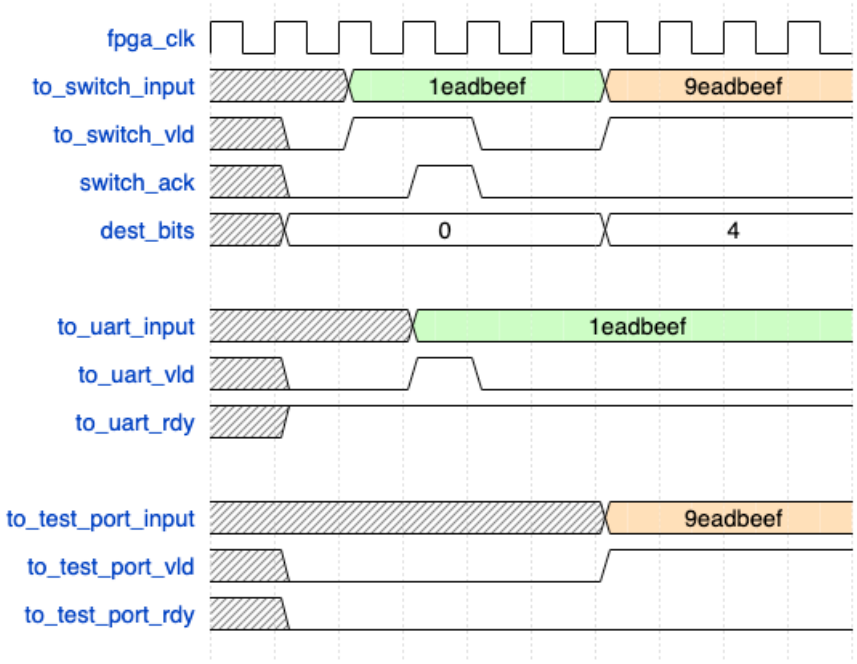

Figure 8. Timing Diagram of Switch

5.3. Kilocore Programmer

The Kilocore Programmer module is responsible for sending configuration subwords to program the Kilocore2 chip. It achieves this by turning on the configuration clock and sending the 18-bit subword along with a valid bit through the configuration port of the Kilocore2 board whenever it receives a valid configuration word from the Python interface via UART. Through the configuration port of the Kilocore2 board, the glue module inside the Kilocore2 chip receives this subword to assemble a complete 64-bit instruction to program the processor array. To acknowledge receiving the subword, glue contains a bit selector and a counter that returns one bit per cycle of the subword received, starting with the least significant bit. The Kilocore Programmer module would receive this return bit, and assemble the complete 18-bit subword with a shift register, a simplistic way to compare and verify if the Kilocore2 chip received the correct bits. The Kilocore Programmer module will also send the

subword back to the Python test station. Although an LED is implemented to light up if the subword returned does not match with the subword sent, considering the situation where the test station is being accessed remotely, the LED would not be conceived by the user. However, the fixed length of 32-bit words to be sent through UART indicates the fixed cost of sending data to be the same if it is a single status bit or 18 bits of the returned subword. Thus, the returned subword is sent to the Python test station after padding the header bits. In turn, the Python test station also compares the two subwords and raise an exception if a mismatch is found.

As mentioned in Chapter 4, the CPLD added to the Kilocore2 board also allow daisy chaining Kilocore2 boards to allow multiple boards to be programmed by a single host. This is achieved by setting a chip select ID in Kilocore Programmer, which when matched with the programmed parameter in the CPLD of the board, will route the configure subword into the chip's glue module, as well as the return bit back to the Kilocore Programmer.

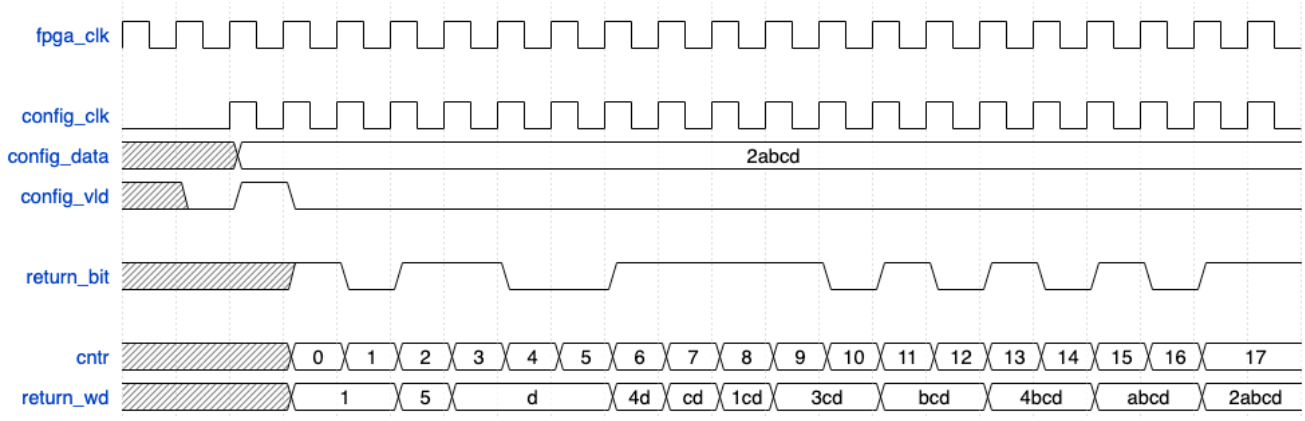

Figure 9. Return Bit Timing Diagram

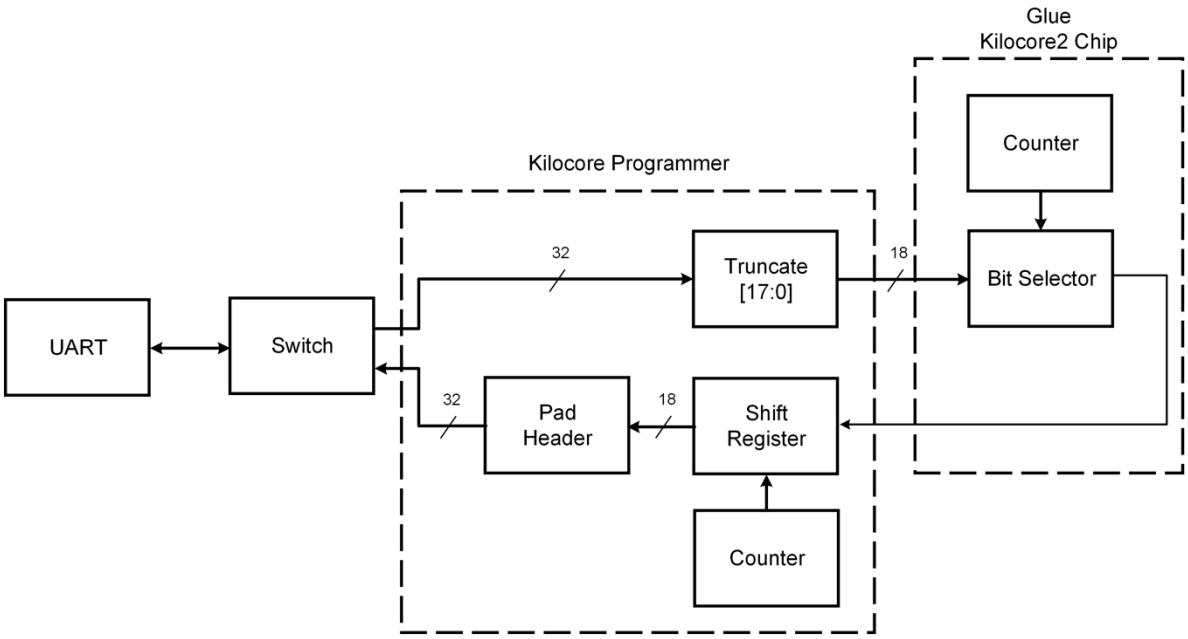

Figure 10. Glue Return Bit Logic

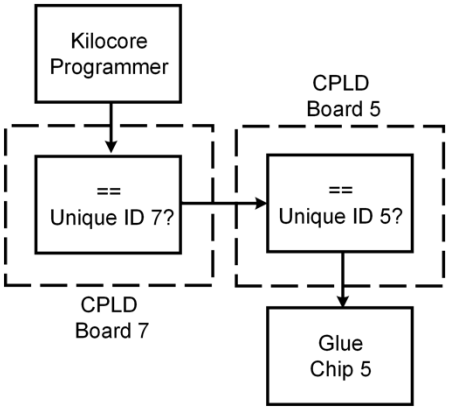

Figure 11. Chip Select Logic

5.4. Test Port

This module samples data from the test port and sends the data to the test station via UART. There are two modes of sampling, automatic and manual, where automatic mode sends data whenever there is a positive edge of the valid bit at the test port, whereas manual mode sends one data after the request is received and a positive edge of the valid bit at the test port is received. While automatic sampling is convenient in

simple tests, manual sampling should be chosen to avoid congesting the switch network and cluttering in the Python terminal.

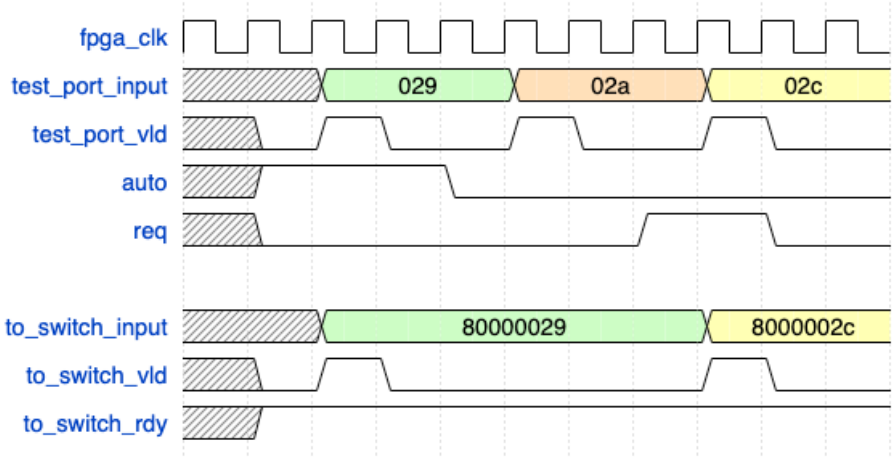

Figure 12. Timing Diagram of Test Port

5.5. IO Delay and IO Delay Controller

The input and output delay modules each contain an IP block that delays the data from the input and output of the data port with respect to the skew of individual pins. Take the output delay module as example. The output skew is controlled by ODELAYE2, an IP block by Xilinx that uses a 31-tap chain to delay the output signal, and the tap can be loaded or increased by a custom value set by IO Delay Controller. As the output standard can be changed between 1.8V CMOS and LVDS, an output buffer is added to accommodate this variance, which thanks to the re-programmability of FPGA, can be synthesized differently depending on which ports are connected to it.

This module is needed as the length and the delay of each trace connecting the Kilocore2 chip to the data ports on the Kilocore2 board could be different, resulting in signal skew. While the trace lengths of most data ports are matched, the length for Packet Input 1 and Packet Output 1 could not be matched [10]. This module thus compensates the skew by delaying each bit by different amounts.

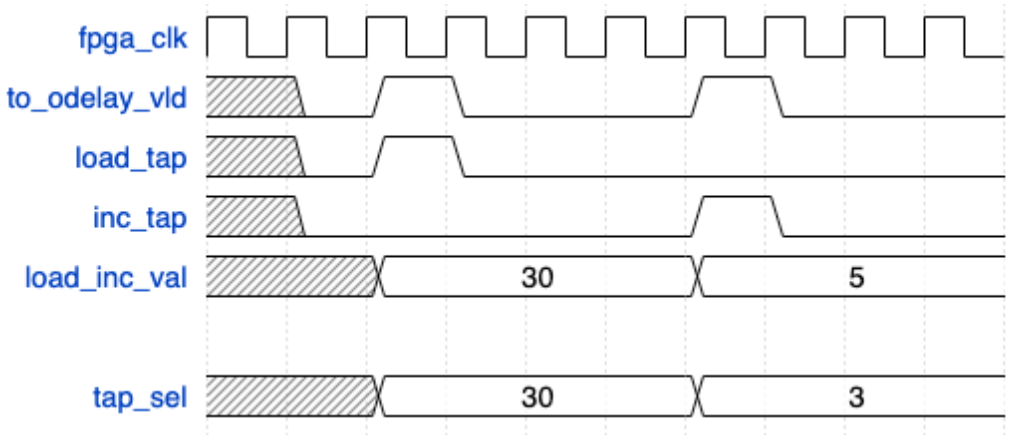

Figure 13. Timing Diagram of Output Delay

5.6. PCIe Module and Data Port FIFO

The PCIe module is responsible for sending and receiving data via the PCIe Gen3 x8 bus on the FPGA. It contains the Xillybus IP block, which handles all PCIe handshakes, and two FIFOs handling input and output traffic. The FIFOs are needed as the data ports use separate clocks to the FPGA clock.

In early stages of debugging, when bandwidth of data port is not a concern, the Data Port FIFOs were connected to the switch module. Obviously, sharing the switch's bandwidth with the data ports is non-ideal, and this configuration should only be used for early stages of debugging to verify the data port FIFOs and the delay taps configuration.

When using the PCIe bus exclusively for data ports, the input and output modules fully utilize the high bandwidth that PCIe offers. The delay taps configuration, while uses UART to configure, should be considered a static register input, and thus does not affect the bandwidth of the data port buses. As the data transfer path is separated from the slower UART switch network, PCIe should be used as the default

data transfer protocol for the data ports in the expense of higher utilization of LUTs in the FPGA.

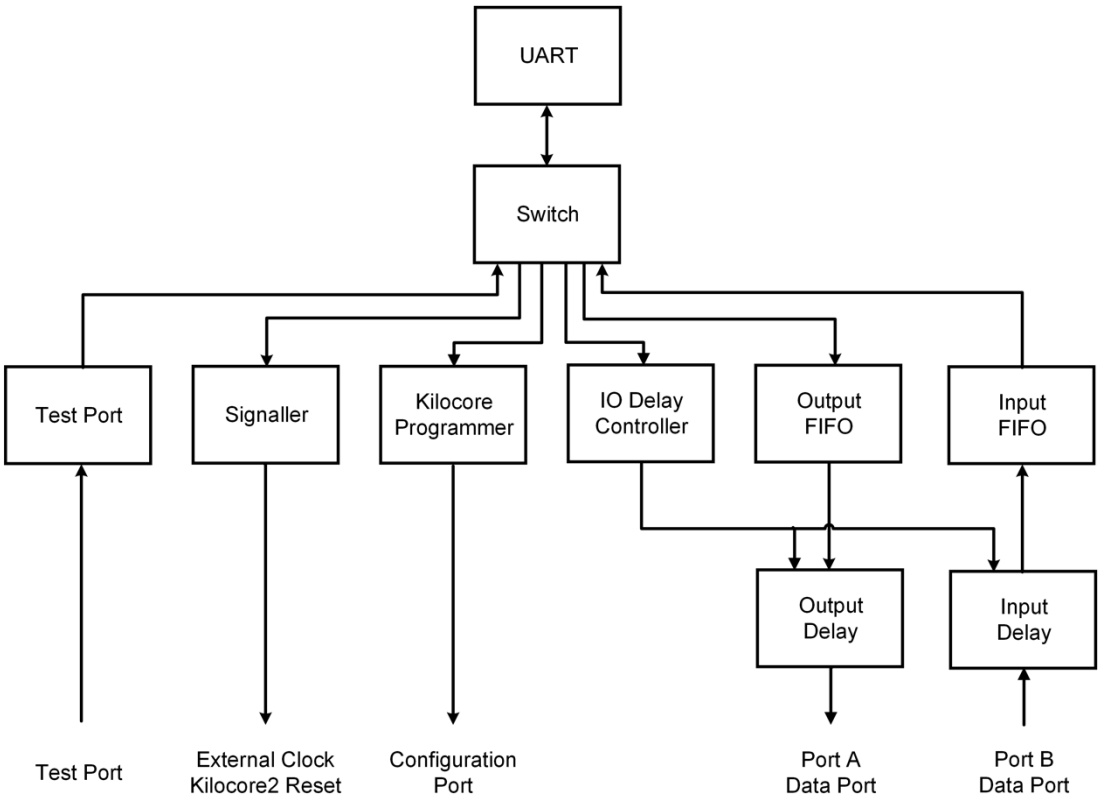

Figure 14. FPGA Top Module with UART Only, Used During Debugging

6. Python Setup

Python is used to control GPIB instruments, and to send and receive data words to the FPGA's UART and PCIe port. Python is chosen as this scripting language provides a simple and readable format for its scripts, making it clear to collaborators, and thus easier to debug during testing. When compared to other languages like C, Python scripts can be launched without compiling, which saves time and effort when scripts are changed to reproduce a trace that previously caused a fault during testing.

6.1. GPIB

As mentioned in Chapter 4, GPIB is used to connect the Linux test station to the instruments that does not connect to the high-speed data buses of the DUT, such as power supplies, multimeters, oscilloscopes, and pulse generators. The Linux test station can turn on/off instrument outputs, set voltage/current limit of power supplies, set output values of the instruments, and request readings from the instruments.
GPIB commands are launched from the test station to each instrument to program them and request data from. The commands are composed of the unique primary address of the instrument was programmed to, and a command that either complies with SCPI standard or is defined by the vendor of that instrument.

On our Linux test station, Python is unable to send GPIB commands directly without using existing libraries or launching subroutines using libraries compiled in a different programming language. In our case, for simplicity, the C library that complements the Agilent 82357A USB/GPIB adapter was compiled to create subroutines that the Python scripts would later launch. This eliminates the need of debugging any potential incompatibility issues when using third-party libraries.

A custom Python object that contains the GPIB address was created such that instruments can be addressed by variable names instead of their GPIB primary addresses in the scripts, so to improve readability and avoid the potential of misconfiguration by a typo in the address field. Common functions, such as turning on/off the instrument, are also added to the object to further improve readability, as GPIB commands are often obscured in their simplified form, and thus often calls for comments to explain its use. For example, to check the connection to an instrument, one SCPI command that is often used is **IDN?*, which asks the target device for its identification string. The mnemonic itself does not describe its function, which prompted for custom Python functions with more descriptive names to improve readability of the script. Therefore, this transforms the action of pinging the power supply responsible for our IO power rail into this one line of Python code: *vddio.ping()*, which is much more descriptive than the subroutine syntax this function launches: *gpib 6 *IDN?*.

GPIB instruments cannot send error messages back to the host actively, and thus queries to the instruments to check error status should be performed often during setup. For example, when setting the voltage and current maximum of a power supply via GPIB, if the value provided should exceeds the maximum ratings of the instrument, the instrument might disregard the command. The instrument would flag it as error, and some instruments might light up an indicator LED; but the Linux test station does not know the error status until it queries the instrument.

When using GPIB during remote setting, queries to the instruments should be made to verify the command is set, along with extra consideration on the accuracy of the reading from the instruments. For example, after enabling the output of a power supply, the voltage and current reading from the power supply should be queried as a

sanity check. But, due to voltage drop-off from power cables, the voltage reading from the power supply is likely an upper estimate. A separate multimeter can be connected to the terminal, or in the case of the Kilocore2 board, separate test pads on the board, to obtain a more accurate reading of the power rail voltage.

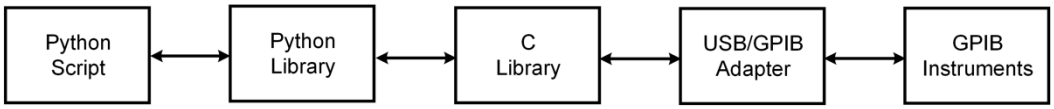

Figure 15. Datapath of Sending and Receiving GPIB Commands to Instruments

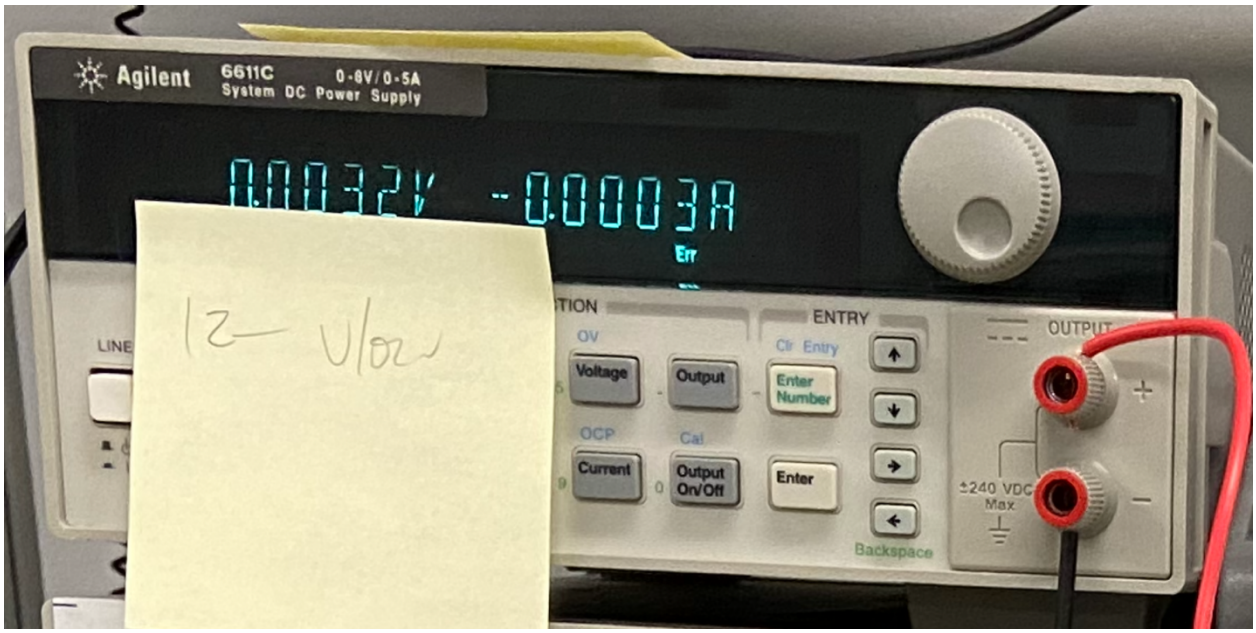

Figure 16. DC Power Supply with "Err" light on

6.2. FPGA Control

The Python scripts control and configure the FPGA hardware by sending configuration words through UART. This is achieved in Python by using the PySerial library to interface with the serial port and setting the baud rate agreed with the FPGA's UART module. By sending 4-Byte bitstrings with the *write()* function of the PySerial library, the UART module in the FPGA will receive and assemble the 32-bit bitstrings that will then be routed by switch to their respective destinations, such as setting the

reset signal of the Kilocore2 chip, or turning on the external clock port on the breakout board. Messages returned from the FPGA can be received with the *read()* function, which then can be decoded and further processed in Python.

A custom Python package is created, with commonly sent configuration words are implemented as Python functions to reduce error by writing bitstring directly and improve readability. For example, a custom function of *prog.Unreset_Kilocore()* would be replacing the PySerial function of *write(b'\x2f\x00\x00\x00')*, where this bitstring cannot be comprehended by anyone beside the designer without comments. Other functions can be assembled by referring to the field of the command and specifying the value of such to further discourage writing in bitstring that harms the debugging process. This also allows more sanity checks to be performed at the Python layer, such as disallowing sending configuration words to modules that are not designed to accept inputs, and thus simplify the FPGA design and decrease FPGA utilization.

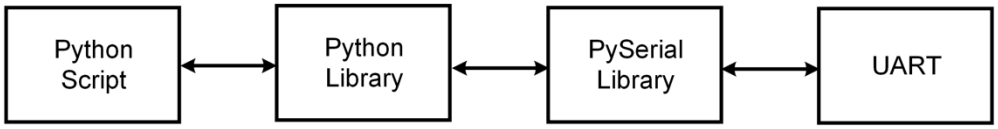

Figure 17. Datapath of Sending and Receiving Configuration Words

6.3. PCIe

The Python scripts send high-speed data to the data ports on the Kilocore2 board via the PCIe port of FPGA. This is achieved in Python by sending data directly to the ports instantiated in the Linux test station by the Xillybus driver, which complements the Xillybus IP block used in the FPGA that the PCIe port connects to.

Of its simplistic nature, the Python interface to write to and read from the PCIe ports does not require custom Python objects and functions to be written to improve

readability and reduce potential errors. However, due to our design that allows the FPGA synthesis to change between using UART and PCIe to send and receive data from the data ports, the function to write and read data from the data ports are currently bundled with the package that sends control words as mentioned above. The Python library shall read from the FPGA defines Verilog header and choose to send data between using PCIe or UART accordingly.

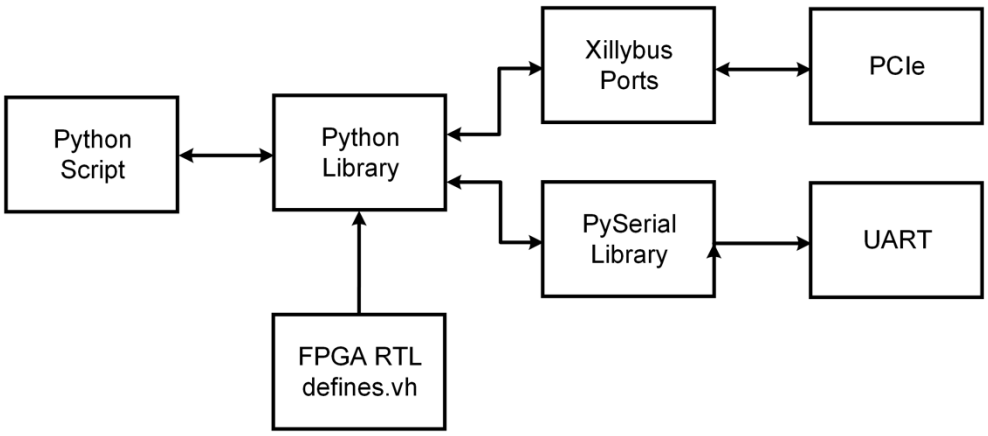

Figure 18. Datapath of Sending and Receiving High-speed Data

7. System Bring-up and Testing

Tests are performed to validate the behavior of our setup and our DUT. Testing our setup is important as faults from the setup will poison the results obtained from DUT testing and create more uncertainty in our fault model. Loopback tests are useful as cut points are created in the middle of the datapath to verify the integrity of data to that point, and extend it to the next block in the datapath until the test covers the entire datapath, essentially creating a divide-and-conquer methodology.

This testing philosophy is applied to our DUT tests. After confirming that the glue module is able to accept configuration instructions with return bits, as detailed in Chapter 4, the next test would be extending one block further from glue. The TVS module fits this description, and also returns the temperature and the core voltage of the chip, values that can be cross-checked. External clock is a feature of the Kilocore2 chip that overrides the local clock of a specified core, and combined with the test pads on the Kilocore2 board to show the clock waveform of the core, also can be tested with a simple test.

7.1. TVS

The TVS test is performed as follows. By configuring the TVS module out of sleep and reset, the temperature or the core voltage of the chip will be sent to the test port of the Kilocore2 board, and then be sent back to the Python test station via UART. The values are converted into Celsius or millivolts by equations specified by the vendor that created the TVS module in Python. The sampled temperature values are verified by spraying cold air onto the chip packaging and observe that the temperature values drop

after. The sampled voltage values are confirmed by measuring and matching the voltage across the test pads that corresponds to chip voltage on the Kilocore2 board.

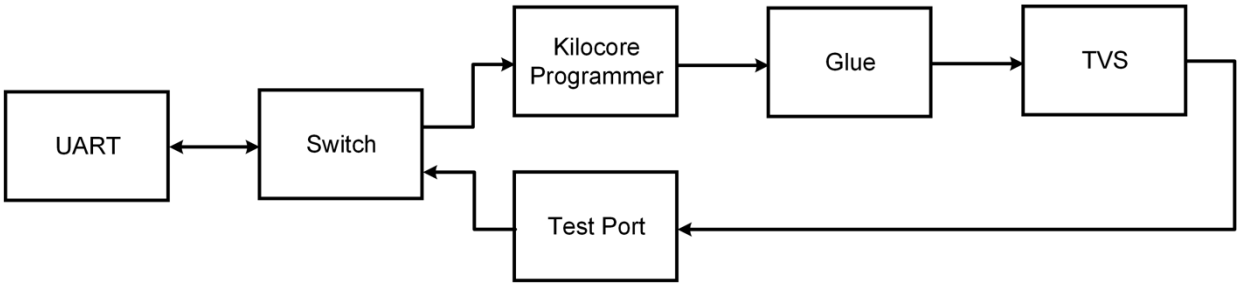

Figure 19. Datapath of Configuring TVS

7.2. External Clock

The external clock test is performed as follows. The test starts with turning on the external clock output on the FPGA breakout board, which is verified beforehand with an oscilloscope. The positive and negative SMA terminals are connected from the breakout board to the Kilocore2 board correspondingly. Then, by programming core 0 inside the Kilocore2 chip to use the external clock, the test pads on the Kilocore2 board should be showing the waveform of the clock on the oscilloscope, which the frequency reading would be sent to the Python test station via GPIB.

Instead, the external clock is currently only implicitly validated by correlation. The differential probes on the test pads currently only detects noise when the board is powered on, and the waveform does not change after various programming attempts that changes the core it probes from, or switching between internal clock and external clock. Therefore, the testing approach changed to minimize influence from other cores and modules inside the chip, and measure the change in core current draw as the external clock is turned on and increases in frequency. The anticipated positive correlation between external clock frequency and core current draw is observed. But we

hope to rectify the issue that caused the test pads to not show the waveform, redo the tests and verify programming processor cores to clock with external clock with waveforms.

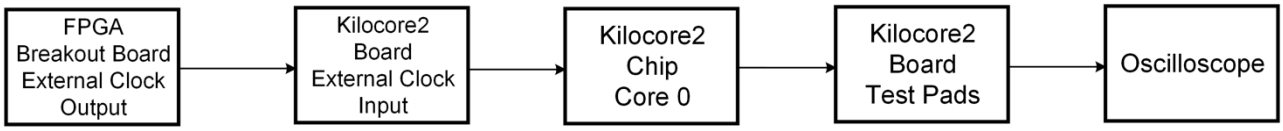

Figure 20. External Clock Datapath

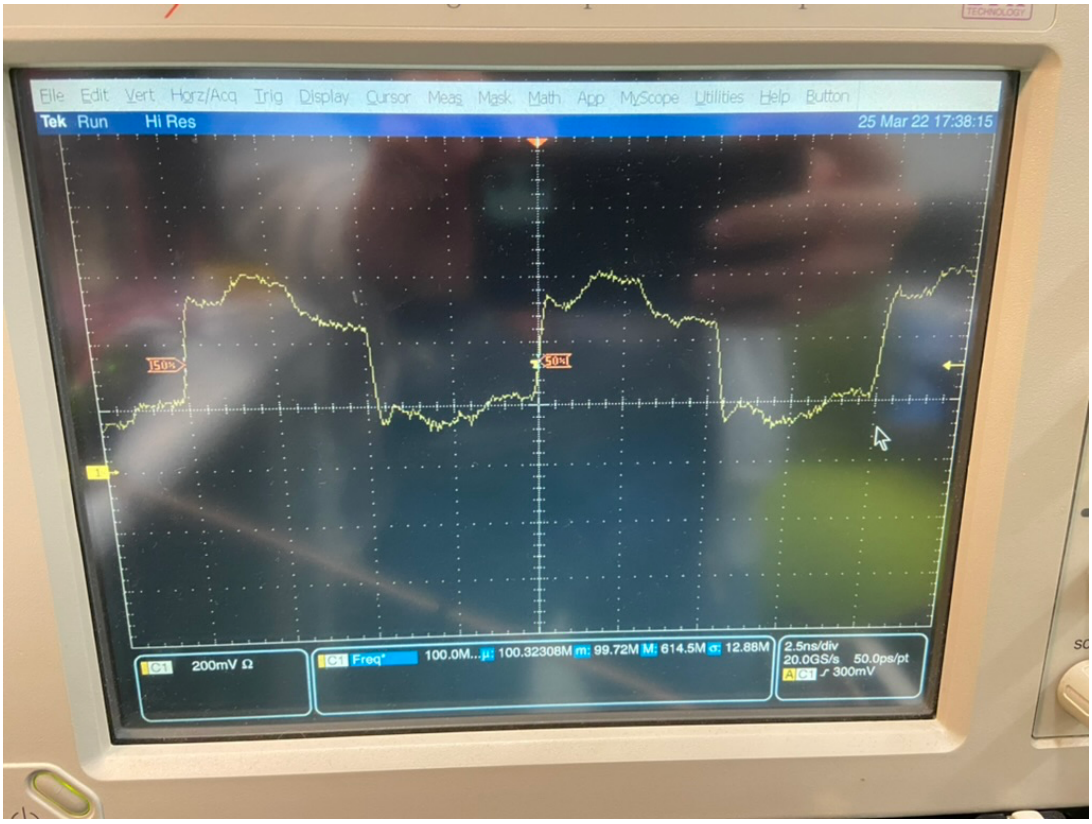

Figure 21. 100MHz Clock Output from FPGA Breakout Board

f (MHz)	50	75	100	150	200	300	400	600
I (A)	1.786	1.883	1.970	2.146	2.321	2.658	2.983	3.579

Table 1. Current Reading of Core Voltage Power Rail with respect to External Clock Frequency

8. Future Work

8.1. Kilocore2 Validation

The Kilocore2 board still has a lot to be verified. It is suggested to test the packet routers feature as they are of less complexity yet crucial for operation. The packet routing algorithm is implemented in hardware; and while the specific route can't be specified, shunts can be programmed to constrain routing hardware, which provides more insight during testing and debugging. Then, the functionality of each processor, where the feature list also differs from the fast processors and regular processors, the Viterbi and FFT accelerators and the internal memory modules are all in need of validation. Examining the power draw from the processors, and thus benchmarking the chip would be the next step after that. As the Kilocore2 board reserved the port for a TE0712 FPGA to be used as external memory, validation of that setup can also be performed in parallel of that.

8.2. FPGA Features

It's expected that more features would be added into the VC709 FPGA for future validation efforts. The current post-implementation utilization of the FPGA is less than 1% of logic units and BRAM, which hints to the potential of tests written in the future, such as data bandwidth stress testing between the FPGA and the Kilocore2 board's external memory with some of the FPGA BRAM used as buffer. The initial programming of static registers of the Kilocore2 chip with default values can be added to the FPGA to streamline the power-on sequence. It might also be feasible to develop a sanity check routine to be implemented in FPGA that will initialize the static registers,

perform some basic operations on some processors, and return the status of those tests through UART to the Python interface.

8.3. Test Automation

The Python package and interface allows automating the testing process. Along with the Kilocore2 assembler and program loader script, it might be feasible to develop a verification interface, where after loading the program and input data, this script shall also take in reference data and compare the output data streamed from the FPGA with it, and report mismatches automatically. It might also be useful for debugging if the reference data is generated by a Verilog simulation of the chip, which combined with the simulation waveform, could provide us insights to where the potential failure originates from.

8.4 Daisy Chain KC2 Boards

The CPLD on the Kilocore2 board enables a common configuration bus for a daisy chain configuration of boards. As the breakout board only consists of two data ports, testing multiple ports simultaneously sending data may be achieved by this daisy chain configuration, by utilizing one validated board as part of the test setup and thus send data words from its multiple output data ports simultaneously to the DUT board's multiple input data ports. The validated board may also be programmed as an automated test pattern generator for future testing purposes.

Glossary

AsAP *Asynchronous Array of Simple Processors.* An architecture designed to implement an array of simple cores that clocks independently to achieve high throughput.

ATE *Automatic Test Equipment*. An equipment that performs automated tests.

BRAM *Block Random Access Memory.* Hardware memory modules in FPGA.

CMOS *Complementary Metal-Oxide Semiconductor.* The technology used in constructing integrated circuit chips.

CPLD *Complex Programmable Logic Device.* A device that is reprogrammable to emulate hardware, often less complex than an FPGA.

DUT *Device-Under-Test.* Refers to the device that is being tested. Within the context of this thesis, refers to the Kilocore2 chip.

DSP *Digital Signal Processing.* The field where signals are processed in the digital domain, such as video encoding and decoding, and data compression.

FIFO *First-in-first-out.* A hardware module that implements a queue.

FMC *FPGA Mezzanine Card.* A parallel standard that connects expansion cards to FPGAs.

FPGA *Field-Programmable Gate Array.* A device that can be reprogrammed to emulate hardware designs.

GALS *Globally Asynchronous Locally Synchronous*. An architecture where the physical cores of the processors can be clocked independently without a synchronous global clock.

GPIB *General Purpose Interface Bus.* A parallel connection standard used in connecting various test instruments, such as power supplies and oscilloscopes.

JTAG *Joint Test Action Group*. Within the context of this thesis, a serial standard used to program CPLD and FPGA devices.

LVDS *Low-Voltage Differential Signalling.* A signalling standard that uses a pair of wires to send symbols represented by the difference in voltage between the pair.

PCB *Printed Circuit Board.* A component that connects various electrical devices with printed traces between the layers on the board.

PCIe *Peripheral Component Interconnect Express*. A serial connection bus that provides multiple lanes for high-bandwidth connections.

RISC *Reduced Instruction Set Computer.* A computer architecture that favors a more simplistic instruction set to increase the execution speed of each instruction.

SCPI *Standard Commands for Programmable Instruments.* A set of commands that are implemented to instruments that are compatible with.

SMA *SubMiniature version A.* A type of coaxial connector used for low noise high frequency signals.

TVS *Temperature-Voltage Sensing*. A module inside the Kilocore2 chip that senses the temperature and voltage of the chip.

UART *Universal Asynchronous Receiver-Transmitter.* A serial standard that transmits and receives data through an agreed baud rate without the use of a clock.

USB *Universal Serial Bus.* A serial connection standard found in computers and test stations.

Bibliography

[1] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, M. Singh, B. M. Baas. "An Asynchronous Array of Simple Processors for DSP Applications." In Proceedings of the IEEE International Solid-State Circuits Conference, (ISSCC '06) , February 2006, pp.428-429.

[2] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, D. Truong, T. Mohsenin, B. Baas, "AsAP: An Asynchronous Array of Simple Processors," IEEE Journal of Solid-State Circuits (JSSC), vol. 43, no. 3, pp. 695-705, March 2008.

[3] B. Baas, Z. Yu, M. Meeuwsen, O. Sattari, R. Apperson, E. Work, J. Webb, M. Lai, T. Mohsenin, D. Truong, J. Cheung "AsAP: A Fine-Grained Many-Core Platform for DSP Applications," IEEE Micro, Volume 27, Number 2, March/April 2007.

[4] R. Jaeger, *Introduction to Microelectronic Fabrication*, Prentice-Hall, 2002.

[5] J. Rabaey, A. Chandrakasan, B. Nikolic, *Digital Integrated Circuits*, Pearson, 2002.

[6] EESemi, *Test Equipment Load Boards | Interface Boards*, EESemi.com, 2005. [Accessed May 2022]

[7] "Instrument Bus Performance – Making Sense of Competing Bus Technologies for Instrument Control", National Instruments, 2021. [Accessed May 2022]

[8] B. Bohnenstiehl et al., *Kilocore: A 32nm 1000-Processor Array*, IEEE Journal of Solid-State Circuits (JSSC), 2009.

[9] *KiloCore2 Specification and Implementation Manual*, VLSI Computation Laboratory, University of California, Davis, 2018.

[10] T. Andreas, *KiloCore-2 Test System Manual*, VLSI Computation Laboratory, University of California, Davis, 2021.

[11] Xillybus, *Getting started with Xillybus on a Linux host*, n.d. [Accessed Feb. 2022]

[12] Xilinx, *VC709 Evaluation Board for the Virtex-7 FPGA User Guide*, Xilinx, 2019.

[13] Xilinx, *7 Series FPGAs SelectIO Resources User Guide*, Xilinx, 2018.

[14] A. T. Tran, D. N. Truong and B. M. Baas, "A GALS many-core heterogeneous DSP platform with source-synchronous on-chip interconnection network," 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip, 2009, pp. 214-223, doi: 10.1109/NOCS.2009.5071470.

[15] B. M. Baas, "A parallel programmable energy-efficient architecture for computationally-intensive DSP systems," The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003, 2003, pp. 2185-2189 Vol.2, doi: 10.1109/ACSSC.2003.1292368.

[16] Z. Yu and B. Baas, "Implementing Tile-based Chip Multiprocessors with GALS Clocking Styles," 2006 International Conference on Computer Design, 2006, pp. 174-179, doi: 10.1109/ICCD.2006.4380812.

[17] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, P. Mejia, A. Tran, J. Webb, E. Work, Z. Xiao, B. M. Baas. "A 167-processor 65 nm Computational Platform with Per-Processor Dynamic Supply Voltage and Dynamic Clock Frequency Scaling." Symposium on VLSI Circuits, (VLSI '08), June 2008, pp. 22-23.

[18] D. N. Truong, W. H. Cheng, T. Mohsenin, Z. Yu, A. T. Jacobson, G. Landge, M. J. Meeuwsen, A. T. Tran, Z. Xiao, E. W. Work, J. W. Webb, P. V. Mejia, B. M. Baas, "A 167-Processor Computational Platform in 65 nm CMOS" IEEE Journal of Solid-State Circuits (JSSC), vol. 44, no. 4, pp. 1130-1144, April 2009.

[19] B. Bohnenstiehl, A. Stillmaker, J. Pimentel, T. Andreas, B. Liu, A. Tran, E. Adeagbo, B. Baas, "A 5.8 pJ/Op 115 Billion Ops/sec, to 1.78 Trillion Ops/sec 32nm 1000-Processor Array," Symposium on VLSI Circuits, Honolulu, HI, June 2016.

[20] B. Baas, Z. Yu, M. Meeuwsen, O. Sattari, R. Apperson, E. Work, J. Webb, M. Lai, D. Gurman, C. Chen, J. Cheung, D. Truong, T. Mohsenin. "Hardware and Applications of AsAP: An Asynchronous Array of Simple Processors." In Proceedings of the IEEE HotChips Symposium on High-Performance Chips (HotChips 2006), August 2006.

[21] M. Meeuwsen, Z. Yu, B. M. Baas, "A Shared Memory Module for Asynchronous Arrays of Processors," EURASIP Journal on Embedded Systems, vol. 2007, Article ID 86273, 13 pages, 2007.

[22] R. Apperson, Z. Yu, M. Meeuwsen, T. Mohsenin, B. Baas, "A Scalable Dual-Clock FIFO for Data Transfers Between Arbitrary and Haltable Clock Domains," IEEE Transactions on Very Large Scale Integration Systems (TVLSI), vol. 15, no. 10, pp. 1125-1134, October 2007.

[23] Z. Yu, "High Performance and Energy Efficient Multi-core Systems for DSP Applications," Technical Report ECE-CE-2007-5, Computer Engineering Research Laboratory, ECE Department, University of California, Davis, 2007.

[24] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, B. Baas, "Architecture and Evaluation of an Asynchronous Array of Simple Processors," Journal of Signal Processing Systems, March 2008.