Design of a Quantum Microarchitecture Integrated Circuit

For Deep Cryogenic Operation

by

Elizabeth K. Hatfield

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Engineering

at the Delft University of Technology,
to be defended publicly on Monday December 17, 2018 at 2:00 PM.

Preface

If a practical quantum computer is to be built, it will not only need quantum programming languages and qubits, but hardware-software interfaces as well. These interfaces, also known as computer (micro)architectures are the key to creating a ’quantum stack’, where a programmer can directly execute programs onto quantum hardware. A functional quantum computer microarchitecture, the Central Controller-Light (CCLight), has been designed, implemented on a Field Programmable Gate Array (FPGA), and verified to control superconducting qubits successfully. The next logical step, in hopes of creating a more scalable quantum computing system, is to take the CCLight and implement it as an Application-Specific Integrated Circuit (ASIC). In this thesis, the CCLight was built into an ASIC. It was shown that this approach yields the potential for higher performance and lower power for the microarchitecture core, and the groundwork for more robust and automatic testing of similar microarchitectures was laid out. Lastly, the lack of scalability in the original approach, wherein the processor transmits multi-byte codewords to the analog/RF hardware controlling the qubits, was revealed.

Elizabeth K. Hatfield
Delft, December 2018

Contents

1 Introduction

1.1. Development of Quantum Computing

The idea of the 'quantum computer', which exploits quantum phenomena to perform computations, was first proposed by Dr. Richard Feynman of Caltech. He proposed that a quantum computer with quantum mechanical elements could feasibly simulate quantum systems, meaning a portion of the universe. Moreover, this quantum mechanical computer could be used to handle the probabilities of large classical systems, which become larger computationally with size [1]. The next major milestone for quantum computing was the birth of Shor's algorithm in 1994. Shor's algorithm, created by mathematician Peter Shor at Bell Labs, is an integer factorization algorithm that incorporates classical and quantum operations, leading to a much faster result than the best classical-only algorithms [2]. This sparked a wave of academic research interest not only in developing further quantum algorithms, but the hardware (qubits) needed to execute them. This spark later caught on with industry as well; companies such as D-Wave in Canada and Rigetti Computing in California were established hoping to build quantum computing devices. Even big name technology companies like IBM, Intel, and Google joined the race for 'quantum supremacy', meaning the point where quantum computers overtake classical computers in terms of computational power [3]. Today, qubit chips are still limited to 50-75 qubits, far less than the hundreds of thousands or millions needed for big speedups over classical computing systems. It remains to be seen who will win the 'race', what qubit technology will prevail, and what system approach will win out.

1.2. The Quantum Computing 'Stack'

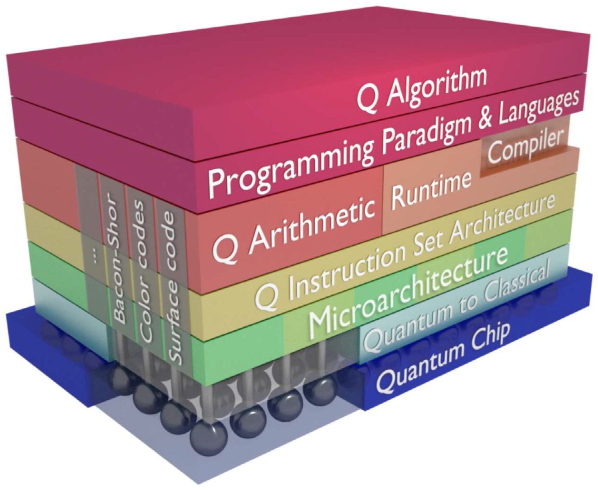
In a classical computing system, there is a 'stack' of elements, of increasing levels of abstraction from the silicon its built out of. At the highest level, there are software applications described by high level languages. These are then compiled to low level assembly languages, and finally assembled to opcodes that correspond to the computers processor's specific Instruction Set Architecture (ISA) and microarchitecture. The processor's microarchitecture is made up of digital circuits and ultimately transistors. All of these levels together make the computer a complete system.

Quantum computers too need to be built into a stack to constitute a fully-fledged and practical system [4].To make a full quantum computing system, quantum algorithms, high level languages, compilers, and microarchitectures will all be needed, just as in classical systems. On top of that - or in light of the stack, below that - lies error correction for the qubits, analog/RF control and readout, and finally the qubit chip itself, as shown in Figure 1.1. Indeed, there is much more to building working quantum computers than qubits and quantum algorithms.

1.3. Need for Scalable Electronic Control

With today's qubit chips, which have a few up to tens of qubits, there is no problem driving these qubits with off-the-shelf electronics. Furthermore, the design of these qubits, often in a planar scheme, call for multiple wires per qubit. This setup usually entails a full stack of control and readout hardware, such as Arbitrary Waveform Generator (AWG)s, Vector Switch Matrix (VSM)s, and Ultra High Frequency

Figure 1.1: The Quantum Stack, Image © Harald Homulle



Quantum Analyzer (UHFQA)s. As the number of qubits scales up, so too does the number of wires, AWGs, and UHFQAs. If a practical quantum computer is to be built, one that can run quantum algorithms with meaningful speedups over classical computing systems, it will need hundreds of thousands or even millions of qubits. This system would then need, if current approaches are to be used, tens of millions of wires and thousands of stacks full of quantum control and readout hardware. This simply is not feasible. Instead, a better approach would be to take advantage of:

- the guiding philosophy of CMOS VLSI by integrating the electronics from stack systems down to chips
- the cryogenic setup of the qubits by placing electronics physically closer to the qubits at a lower temperature

Both bring advantages and disadvantages, but the approach promises to help make practical quantum computing possible [5].

1.4. Objectives and Contributions of This Thesis

The objective of this thesis is to take the specification and implementation of the Central Controller-Light (CCLight) quantum microarchitecture processor on a Field Programmable Gate Array (FPGA), laid out in [6] and [7], and convert the design to an Application-Specific Integrated Circuit (ASIC), which can then be fabricated and operated not only at room temperature, but also at cryogenic temperatures.

The contributions of this thesis are as follows:

- Creation of an ASIC from the CCLight FPGA Firmware (and thereby making the firmware ASIC-compatible)
- Increased the fault coverage potential of eQASM-based designs by creation of a script to generate randomized eQASM programs for benchmarking
- Realization of the trade-off between the desire for system flexibility and scalability

2 The CCLight Processor

In order to tie together quantum hardware (qubits, analog/RF electronics), and software (quantum algorithms, high level languages), and to define another layer in the quantum stack, a software-hardware interface needs to be built. This means defining the lowest level of programming a programmer will see, called an Instruction Set Architecture (ISA). This ISA is then implemented with a microarchitecture, the highest level of abstraction in the digital hardware design from the transistors it is ultimately built from. The circuit defined by the microarchitecture will eventually push instructions to the analog/RF electronics that control the qubits. With this need in mind, the Central Controller-Light (CCLight), a quantum microarchitecture processor, was created.

2.1. The CBox

The precursor to the Central Controller-Light (CCLight) was the Control Box (CBox). The CBox was a system designed to control a 3-qubit chip with feedback. The innovation in the CBox was that all of the control electronics were integrated into a single 'box' (hence the name). Furthermore, the controller of the design was implemented on an Field Programmable Gate Array (FPGA), meaning that the control and management of the hardware interfacing to and from the CBox, as well as the other hardware components inside the CBox, are described with digital circuits. This is in contrast to a microcontroller (or microprocessor) implementation, which would simply be configured with a program written in C or assembly. This choice gives the design potentially enhanced functionality (i.e. the FPGA can do things the microcontroller cannot), as well as improved speed, at the cost of higher power consumption. The choice also paves the way for a future processor/microarchitecture controller, which can be implemented on an FPGA. This FPGA in turn controlled daughter FPGAs controlling Arbitrary Waveform Generator (AWG) boards within the CBox, a USB interface for communication to and from a user PC, and qubit control and readout. The CBox ultimately achieved its goal of controlling and reading out a 3-qubit system as an interface between user PC and quantum hardware [8].

Inside the CBox, there was one main FPGA, 3 AWG subsystems with daughter FPGAs and 6 DACs for IQ modulation, and 2 ADCs for readout, plus some power regulators. Inside of the main FPGA, there are 9 functional modules. These functions include communication (USB and SPI), error correction, and feedback signal processing. Furthermore, the software front-end of the CBox for the user PC was provided in the form of two Matlab® programs. One was written to create the waveforms for the DACs to send to the qubits, program the DACs with the waveform, and associate the waveform with a particular command operation to the AWG. The second program allows the user to configure the ADCs into different operational modes and ADC parameters like 'delay to integration' and 'integration length' [8].

2.2. Computer Architecture for Quantum Computing

Computer architecture is often defined simply as an Instruction Set Architecture. An ISA is a specification of instructions that would be executed by a processor, which itself is in many ways described by the ISA. Some details of the *implementation* of that processor, such as pipelining and memory sizes, are not a part of that definition however. Sometimes the implementation details are considered the 'microarchitecture' of a computer. These two elements, plus other 'hardware' details like clock speed make up the description and design of all processors [9]. Ultimately, both the ISA and microarchitecture are of importance in the design of a processor for controlling qubits, as they control the usability and performance of a system built from it.

To date, little research attention (relative to the top and bottom of the stack) has been made to the topic of computer architecture for quantum computers. Indeed, far more focus is placed on the topics at the top of the stack, namely quantum algorithms [2] [10] and high-level languages [11] [12] [13], and at the bottom of the stack, such as the design and manufacture of physical qubits [14] [15] [16]. This may be due to any number of reasons, such as researchers not being concerned with the systems approach, only the highest/lowest level of the quantum computer. However, in order to build a true full-stack, fully-fledged quantum computer, a hardware-software interface is needed; a direct execution of quantum algorithms onto qubits is needed.

Some previous work has been done to build these interfaces, in the form of ISAs. These include Open QASM, cQASM, and QuMIS. Open QASM was designed as an open-source quantum assembly language in conjunction with IBM. This language can do fundamental single and two qubit gates, allows for user definition of custom gates as subroutines, resembles C-family syntax, but lacks a feedback mechanism from the qubits within itself [17]. cQASM was created in hopes of a common quantum assembly language *syntax* so that QASM programs for various hardware systems, applications, and in different languages or "dialects" can be translated to different languages and quantum hardware [18]. Lastly, QuMIS, the precursor to eQASM, which aimed to be a flexible implementation in a practical microarchitecture and capable of controlling real experiments on superconducting qubits [6]. QuMIS failed however to provide a few key elements to its specification, most notably feedback from the qubits, which is imperative to executing quantum algorithms [7].

2.3. eQASM: the Executable Quantum Instruction Set Architecture

The executable Quantum Assembly (eQASM) is a quantum Instruction Set Architecture (ISA) that can be directly implemented as a physical microarchitecture. In other words, the ISA can be implemented in digital hardware with pipelines, memory elements, arithmetic blocks, which then interfaces with analog/RF hardware that controls and reads out qubits.

2.3.1. Innovations of eQASM

Unlike other quantum ISAs, the eQASM specification provides instructions allowing feedback from the qubits. The specification also alleviates the "quantum operation issue rate problem" [7] with the use of a Very Long Instruction Word (VLIW) architecture and Single Operation Multiple Qubit (SOMQ) execution. The first, VLIW, means that a single instruction in the instruction memory can correspond to two operations which can then be executed in parallel, increasing execution rate. The second, SOMQ, means that a qubit gate (single ot two-qubit) can be applied to multiple qubits at once. This is achieved with masks that contain multiple single qubits or multiple qubit pairs. Solving the "quantum operation issue rate problem" means that the ISA is feasible for a practical implementation operating a real qubit chip.

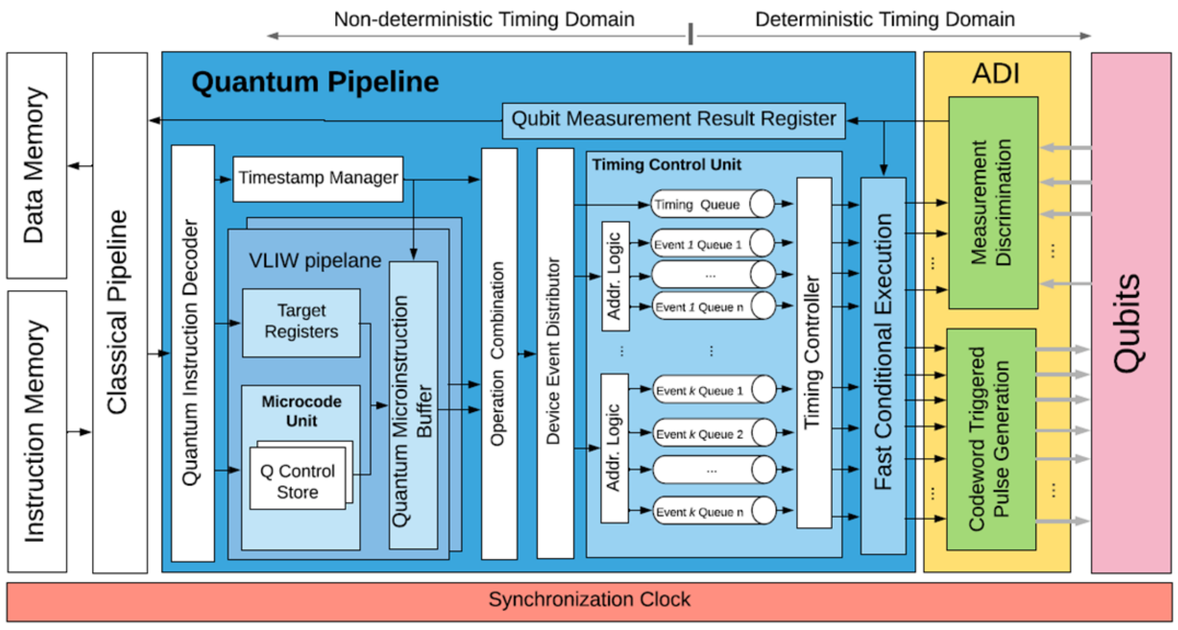
2.3.2. Instructions of the ISA

In the specification for eQASM, there are two classes of instructions: classical, and quantum. The classical instructions are mainly standard instructions, like logical and arithmetic instructions, whereas the quantum instructions are concerned with the execution of operations on the physical qubits. In total, there are 14 classical instructions, including 2 program control (branching) instructions, 6 data transfer instructions, 4 logical instructions, and two arithmetic instructions. This also includes the instruction FMR, fetch my result, which reads qubit measurement results into a general purpose register. This is considered as a classical instruction as the destination is a general purpose register, whose data is available to classical instructions, not quantum instructions. For the quantum instructions, there are only 4 explicitly defined, 2 for defining the target qubits, and 2 for specifying wait times between operations. However, the eQASM definition leaves a footprint for any number of quantum operations; this number is only limited by the bit size chosen for the microarchitecture implementation. These are the operations that are performed on the qubits. The quantum operations can be user defined, as the target microarchitecture and system implementation is codeword controlled quantum hardware (AWGs, VSM) based.

2.4. The CCLight Processor Microarchitecture

The Central Controller-Light (CCLight) microarchitecture is a practical validation of the eQASM specification. The CCLight uses 32-bit instructions, a 100 MHz core clock frequency and 50 MHz output frequency, and was implemented on an Altera® Cyclone V FPGA. The design leaves room for 37 single qubit operations, 7 flux operations, and 17 two qubit operations to be defined by the user. A microcode unit stores the information about each of these instructions, so that the processor knows the right codeword(s) to output for the quantum hardware and the desired operation.

Figure 2.1: The CCLight Microarchitecture, from [7]



The CCLight microarchitecture was implemented with over 100 modules and submodules. Hence, it is easier to analyze the microarchitecture, as visualized in Figure 2.1 by dividing it into the following regions:

- Classical pipeline
- Quantum pipeline - processing side
- Quantum pipeline - issue side
- Communication handlers

2.4.1. The Classical Pipeline

In the classical pipeline region there are three major elements: the instruction cache, the measurement result register file, and the classical pipeline itself. The instruction cache holds each 32-bit instruction, up to 32768 (32 K) individual instructions. The measurement result register file holds the results of the qubit measurements read back to the processor. It can hold 7 results (equal to the number of qubits), each consisting of a single bit. Lastly, there is a classical pipeline. It is named this way not only because it executes the classical instructions, but because it has the standard 5-stage design. That is, every classical instruction is executed in the following steps: instruction fetch, decode, execute, memory, write-back [19]. For the FPGA to operate the classical pipeline at 100 MHz, the pipeline had to be primarily implemented by direct instantiation of hardware blocks within the FPGA, namely flip-flops for inter-stage memory, and Look-Up Table (LUT)s for logical or arithmetic operations, as opposed to writing these parts behaviorally. There are two register files in the memory stage, for single and two-qubit mask specifications, each holds 32 masks of 32 bits. These register files were also direct instantiations of Altera® IP.

2.4.2. The Quantum Pipeline - Processing Side

In the quantum pipeline - even just the processing side of it - there are a great number of modules. The most important are the quantum decoder, the items of the VLIW pipelanes, the instruction joiner, and the timestamp manager. In the definition of eQASM, each quantum operation is given in a bundle, so that two operations can be processed by the microarchitecture at once. It is the job of the decoder to split the instruction into the two operations, which are then fed to the two pipelanes for parallel processing. In the pipelanes, the operation is translated to the corresponding microwave, measurement, and/or flux codewords by way of a module called the microcode unit. This is a programmable memory so that the user can define each of the operations and the codewords that go to the quantum hardware, which need to be configured so the codewords generate the right waveform. The two microcode units store words of 22 bits and have a capacity of 256 words. Next the instruction joiner takes the outputs from the pipelanes and puts them back into one word for singular processing into the issue side of the quantum pipeline. Lastly the timestamp manager creates a timestamp or issue trigger for each quantum instruction from the wait interval specified from `QWAITR` or the pre-interval (prefix `bs`).

2.4.3. The Quantum Pipeline - Issue Side

The issue side of the quantum pipeline is mainly concerned with the execution of quantum operations at precise timing points. Hence it is built of 7 FIFOs, in addition to an operation distributor and a timing controller. Most of these make up a unit aptly named the timing control unit, which assures the timing of quantum operations. Everything in this part of the processor is concerned with placing the codewords into the respective FIFOs - one for flux, two for measurement, two for microwave, one for VSM, and one for timing points - and requesting them at just the right time. Each FIFO has separate read and write clocks, along with status signals for the occupancy of them, which are used by dedicated finite state machines that control reading and writing to the FIFOs. At the end of the quantum pipeline, before the signals go to the final outputs, they are bit mapped to the codeword form expected by the quantum hardware. In addition, the VSM signal goes through a Double Data Rate (DDR) module. Lastly, the DDR module was implemented in order to give the VSM control signal a sharp rising and falling edge, by pushing it as close to the 1 ns resolution (1 GS/s) VSM module specification as possible.

2.4.4. Communication Handlers

To handle the programming of the processor, two separate Advanced Extensible Interface (AXI) buses are used. AXI is an interface protocol released by ARM for communications between processors [20]. This interface was chosen primarily for ease of use with the ARM cores within the host FPGA. One is for programming to the instruction memory and the microcode units, and the other to registers to enable and start the processor. the 'memory' AXI uses a data length of 64 and an address length of 24, while the 'register' AXI uses 32 and 16 respectively. On the back-end of the AXI buses (and the front-end of the processor) there are translation modules that simply pull out the data and address information from the transmitted word coming from the AXI buses. The 'register' translation module is also in control of the main reset signal across the processor.

2.5. The Development Environment

The source firmware for the CCLight is a complex one, hence its management primarily from a GitHub repository. GitHub is a platform for managing software projects, focusing heavily on the version management aspects with constructs like branches and forks for projects. These functions lend themselves equally to firmware projects like the CCLight, and the software packages made to facilitate its development. Among many other pieces to the CCLight development environment the most notable items include:

1. HDL Designer - for building and editing the CCLight firmware
2. QuestaSim - a simulation tool which has a plug-in to interface with HDL Designer directly

3. Python Assembly Script - a tool that assembles a QISA file into binary and loads it to the testbench for simulation in QuestaSim

2.5.1. HDL Designer

HDL Designer is a firmware design suite from Mentor Graphics. HDL Designer is different from most other firmware development tools in that it is graphically based. The intent is that the fundamental blocks of an HDL design will be correlated to physical blocks in a graphical view, with the interconnect and generics/parameters defined similarly. Changes made in the graphical view are automatically changed in the generated HDL for the modules. This is a very useful and powerful functionality for designs with many smaller submodules, as many complex designs like the CCLight are.

2.5.2. QuestaSim

QuestaSim is an HDL simulation tool from Mentor Graphics which allows simulation of designs via waveform, or visualizing the values of signals within the design over time. These waveforms can be analyzed at key time points for accuracy, or saved into a waveform file (.wlf) and then compared to another waveform file or new simulation. Both techniques can be very useful for debugging designs, and the latter for verifying correctness of a design. Additionally, the QuestaSim tool has an integrated console which takes TCL commands natively, so the entire simulation process can be easily scripted, which speeds up the process significantly. The most important reason this software was used is the native plug-in from HDL Designer, which allows a design to be loaded from HDL Designer to QuestaSim for simulation without rebuilding the project within QuestaSim itself. This keeps coherence between the design under development in HDL Designer and the design under test in QuestaSim.

2.5.3. Python Assembly Script

To allow for the assembly of any quantum program described in eQASM to executable binary for the CCLight, an assembly script in Python was provided. The script works by attempting to parse the eQASM (QISA file) into defined quantum instructions, which are read from a definition file called a QMAP file. The QMAP file defines by name all of the instructions in the CCLight, from the classical instructions to single-qubit instructions to flux instructions to two-qubit instructions. It also gives the corresponding binary opcode, which is what the processor receives and executes with in the output binary file. If an instruction is given in the QISA file that is not in the QMAP file, the script gives an error. Furthermore, the script checks the general syntax of the QISA file and assures that is correct before assembly. Then, once the QISA file is correctly parsed, the script creates two binary files: one with the original eQASM instructions as comments on the corresponding opcodes, and one with only opcodes. Both are actually written in hexadecimal notation to conserve length in character count and file size (and increase human readability). Lastly, the assembler rewrites the input file for the CCLight testbench to feed the instruction cache with the opcodes, so that on the next QuestaSim simulation launch, this new program is executed.

2.6. Description of FPGA Operation

The CCLight as it was implemented on an Altera® FPGA operates with a few distinct stages. First, the instructions and microcode are loaded to the respective memory elements, and some enable registers are written to. Next, the processor begins executing the instructions in the cache, either in the classical or quantum pipeline. This builds up codewords in the appropriate output FIFOs. At the timing points specified in the instructions, independently of the instructions running in the pipelines, the processor issues the codewords from the FIFOs. Lastly, the processor reads input from the qubit readout to some measurement result FIFOs, allowing the processor to fetch that data when the Fetch Measurement instructions are to be executed. There are some extra peripherals on the top level of the design that will help control the accuracy of the timing of the processor.

2.6.1. Initialization and Run Stage

The first stage of the processor operation is initialization. This means programming everything the processor needs to execute a program through the AXI interfaces. This is a standard procedure, with the following items in order:

1. Write the instructions to the instruction cache
2. Write the codewords to the microcode unit
3. Configure delays on output signals (if needed)
4. Write to the enable register
5. Write to the run register

First, the opcodes from an eQASM program are uploaded to the instruction cache via the 'memory' AXI buses. each opcode with target address is sent over the parallel interface to the 'memory' AXI communication module. This module translates the target address to the instruction cache, and pushes the opcode with address and write enabling signals to the instruction cache. This is done one by one for each opcode in the program. After this is complete, the microcode unit is written in an identical fashion. Next, the 'register' AXI becomes the target. If the user desires delay on certain output signals, such as microwave0 or flux, this can be configured by writing the desired value to the corresponding register, with the data and register address packaged in the same way as the memory writing. Next, the enable register is written to with a 1. This means in terms of the design the reset signal across all regions of the design is lowered, so all of the internal modules of the design can have their initial values, but are now ready to change. Lastly the run register is written in the same way as the enable register, meaning the the program counter of the classical pipeline can start, and the instructions inside executed.

2.6.2. Processing Stage

With the run register written, the processor starts executing its instructions. This means classical instruction go to the classical pipeline, and quantum instructions go to the quantum pipeline. The classical pipeline has the standard five-stage pipeline setup (fetch, decode, memory, execute, write-back), whereas the quantum pipeline essentially grabs quantum instructions, and in two *pipelanes* decodes the instruction to its corresponding codeword, and places it into the corresponding FIFO along with the timing info specifying when the codeword should be executed into the timing FIFO. This process continues for as long as the program goes, which could have loops extending this time past the physical length of the program.

2.6.3. Output Management and Execution

Once there are codewords in the output FIFOs, and the time comes to send some, the timing control unit pushes out the items corresponding to the current timestamp. This happens asynchronously with respect to the classical pipeline and processing side of the quantum pipeline. In this way it can be assured that the codewords are issued at precise timing points, regardless of the execution happening within the pipelines e.g. arithmetic instructions, NOPs, and branches. These codewords are sent out for measurement, microwave, flux, etc., and then translated to the codeword structure expected by the quantum hardware, before finally being outputted.

2.6.4. Feedback and Other Peripherals

After initialization, the processor is able to read in measurement results. The processor continuously reads the result input buses and stores any received data words in some measurement result FIFOs. These FIFOs then store the data until they are pushed out and into the classical pipeline when a fetch measurement instruction is executed. In addition, there are some phase-detecting modules that check the main clock frequency for drift or inaccuracy with respect to an external PLL signal. If the module detects too great a shift in the clock from the desired frequency, the processor is reset. This is due to the necessity for precise timing on output codewords, which is always defined by the main clock frequency.

3 Development of the CCLight for an ASIC Implementation

While a Field Programmable Gate Array (FPGA) implementation provides such advantages as faster development time, lower cost of entry, and design flexibility, an Application-Specific Integrated Circuit (ASIC) has three main advantages:

- Better Performance
- Lower Power
- Smaller Size

All of these advantages are highly suited for scalability, as well as cryogenic operation, over an FPGA. Hence, an ASIC implementation of the quantum Instruction Set Architecture (ISA) executable Quantum Assembly (eQASM) is the desired approach for cryogenic control of qubits.

In this chapter, the details of implementing the CCLight design as an ASIC in TSMC 40 nm technology are discussed.

3.1. IP Block Replacement in the CCLight Core

The first step in implementing the Central Controller-Light (CCLight) as an ASIC was to replace all of the Altera® IP blocks within the CCLight design with open-source blocks that could be legally implemented in the hard chip.

3.1.1. IP Blocks to be Replaced

First, an analysis of the existing CCLight HDL code was done. The following Altera® blocks were identified in the design:

1. DPRAM module, for the instruction cache, one instance
2. Register file, for the classical pipeline, 2 instances
3. FIFO module, for the codeword output and measurement fetch, 9 instances
4. Flip-flop module, for the classical pipeline, 30 instances
5. Look-up table (LUT), for the classical pipeline, 5 instances
6. Double Data Rate (DDR) output module, for the VSM output, one instance

Most of these modules were implemented in the design to meet the tight timing constraints on the target FPGA, as discussed in section 2.4. Especially the flip-flops and LUTs were crucial in making the classical pipeline meet the 100 MHz frequency requirement when synthesized on the physical FPGA. Likewise the utilization of the DPRAM, FIFOs, and register files allow for quicker memory access. This is especially critical for the FIFOs on the output of the CCLight, which need to have fast responsiveness to keep operation issue latency low. Lastly the DDR output module was implemented in order to bring the VSM control signal as close to the 1 ns resolution (1 GS/s) specification as possible.

3.1.2. Replacement Methodology

To replace each component, a 'black-box' approach was taken. That is, the individual component was considered as an arbitrary sub-module with some known inputs and known outputs within a larger design element. This approach was chosen because A) the components needing replacement did not have readily available testbenches (exclusive for the component) and B) none of the components in question had readable behavioral source code available, and only two (the FIFOs [21] and the DDR output [22]) had documentation available.

Conveniently, a replacement for the LUTs and flip flops was already in place in the design. The module encompassing the classical pipeline core was given a boolean generic value, `gBehavioralImpl`, which would switch the logic of the core from instantiations of the Altera® blocks to a simple behavioral description. Changing this value from 'False' to 'True' removed two of the Altera® components from the design completely.

For the rest of the Altera® components, the same testbench and benchmark program was used to make a replacement component. The waveform window was altered to focus on the inputs and outputs of the component in question, as well as other signals of interest within the larger module the component exists in. Based on the visible ports, the input/output response of the component, and known general functionality of the component, a rough behavioral HDL description was made. This description was swapped into the design in place of the original Altera® component, and then the testbench and benchmark were run on the whole CCLight with the replacement module in place. One by one, with some iterative debugging, most components were replaced within a day or two. The FIFO module proved the biggest challenge by far; it required two weeks of writing, testing, debugging, and rewriting to get it working.

3.1.3. Replacement Verification

Lastly, with all replacement components in place, all benchmarks were swept, to confirm the new implementation of the CCLight meets the original specification. First however a change was made to the architecture of the entire design. A boolean generic value was added to the top-level module of the design, `QuMATop`, and all of the submodules descending to the submodules implementing the replacement modules. Then with the Altera® modules back in place, and the replacement modules still in place, both units were placed into a switching (generate statements) architecture. This HDL architecture allows for either the Altera® module or the replacement module to be instantiated at compile/simulation time with a simple switch of the generic value. A similar approach was done in the original classical pipeline HDL code. This made running the benchmarks on the original implementation versus the new implementation much more convenient.

Next, the benchmarks needed to be swept. The CCLight design was verified with 10 benchmarks, especially testing key parts of the design including branching and the VSM trigger ouput timing. Additionally, 15 more benchmarks designed to test the timing and latency of the design were collected. Each of these benchmarks, all written in eQASM, were individually assembled to executable binary with an assembler provided with the CCLight development environment. The given testbench for the CCLight was altered to show results for the primary outputs to the AWGs and VSM, signals `B_Trigger`, and `DIO1-5`. Furthermore, the provided testbench was modified to save the waveform file (.wlf) after the benchmark and the implementation (Altera® or not). First, this procedure was performed for every benchmark on the original FPGA implementation. Then the design was switched to the new Altera® IP-free implementation, and the testbench was again modified to automatically run the QuestaSim comparison tool with the Altera® implementation result of the same benchmark used as a reference. The testbench was also modified to print any discrepancies reported to the QuestaSim console to a text file named after the benchmark being performed. Lastly the benchmarks were all swept again on the new implementation and the comparisons to the original design's responses were made.

This revealed two small bugs. One in the FIFOs where they were too fast in the case of the measurement result FIFOs, which was easily remedied by delaying the output of the delinquent signals (`wrreq` and `data`). The second bug wasn't as clear in its origin. On some of the benchmarks, incorrect outputs were observed on the microwave FIFOs exclusively. After swapping out each replacement component individually, the issue was isolated to the classical pipeline. Then a comparison was made on a faulting benchmark between the response of the Altera® module and the replacement module. This allowed the error to be traced back to its origin in time, which was revealed to be three branching signals, `BrXReg`, `BrYReg`, and `BrZReg`, which were not reset properly. This error was then fixed easily. Once both were remedied on the first benchmarks where they gave errors, the benchmarks were swept again, this time without any discrepancies with respect to the design utilizing Altera® components.

At this point, the Altera® IP-free design was ready to be altered for ASIC implementation. More specifically, the CCLight was ready for Input/Output (IO) redesign.

3.2. Pin-out Redesign

The original CCLight design had a few luxuries. One of the main ones was its host FPGA, the Altera® Cyclone V SOC 5CSTFD6D5F31I7N. This FPGA has over 200 pins at the designer's disposal for general IO, and 896 on the package [23]. For the original CCLight design, this was greatly utilized; in the end the CCLight FPGA design had a total of 625 pins for programming the chip, communicating to the quantum hardware, and other IOs.

In principle, any number of pins can be implemented in an ASIC implementation of the CCLight. This number is only limited by the physical number of bond pads (meaning the structure in silicon that is connected electrically to the package it is enclosed within) that can be placed within the allotted silicon area. Ideally a BGA or similar technique would be used to allow for much larger pinouts, but this technique is too complex to implement for the design. This leaves the classic bond pad ring technique for the ASIC. While the area of the ASIC could be increased to accommodate any number of bond pads on the ring, there are two costs:

1. The price of the chip increases per mm^2
2. Bond pads are only placed around the circumference of a chip, so most of the extra silicon area will go un-utilized in the core of the chip

With these considerations in mind, a trade-off for this ASIC design was made. An area constraint was set at $1.5mm^2$, as the CCLight design plus memory macros should fit within this area in the TSMC 40 nm technology targeted for the implementation. Memory macros can become very large with larger capacities (doubling from 4K to 8K addresses, etc), but the core of the design, even with some significant memory (2^8) elements that must be synthesized in the design, should fit within $1.0mm^2$ with TSMC 40 nm technology. The instruction cache must be an SRAM macro block (originally 2^{15} or 32K in size) as it is too large for standard synthesis tools (the results would be worse as well [24]). However, for the TSMC 40 nm technology, a 32K SRAM is not available. The closest available is (2^{13}), which needs a physical size of $0.265mm^2$. Since most eQASM programs provided for testing (and hence were written as real experiments) were well under 100 lines, this reduction was not considered fatal to the design and could be easily accommodated. With the lack of necessity for a higher capacity in the SRAM, and the desire for low power in mind, the 4K (2^12) SRAM block was chosen. This choice also gives a performance bonus; the 8K SRAM has a nominal cycle time just above 1 ns (1 GHz clock speed) while the 4K SRAM has one just below 1 ns, meaning this SRAM could be used theoretically at 1 GHz speed. The area of the 4K SRAM is $0.15mm^2$, pushing the upper limit of the total chip size to the next size bracket, $1.5mm^2$.

Given an area of $1.5mm^2$ and an aspect ratio of 0.5 (aspect ratio = width of die / height of die) the circumference of the chip will be $5,000\mu m$. Each corner of the chip will be occupied by a corner cell which is a square cell with a pitch of $120\mu m$, leaving $1260\mu m$ on the left and right side, and $760\mu m$ on the top and bottom sides of the chip. If the bond pads are placed in a linear ring around the core, then only one bond pad can be placed every $110\mu m$, which means there could only be 36 pins on the final chip. However, if a staggered bond pad placement is used, then an average of one bond pad per $55\mu m$ can be achieved. Thus, a maximum of 72 bond pads can be placed on the chip.

Therefore, the number of pins in the CCLight needs to be decreased from 625 pins down to 72 or less in order to be implemented as an ASIC economically. In order to do this, the pins were organized into the following categories: clocks, programming interface, and input/output pins. Each pin group was reduced with its own criteria and constraints.

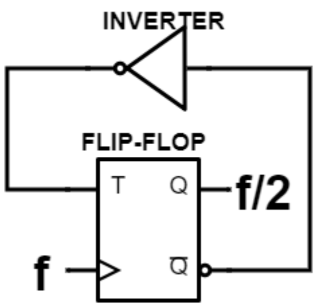
3.2.1. Clock Reduction

The original CCLight design employed six separate clock signals:

1. Clock_200MHz - a 200 MHz reference clock
2. clock_100MHz - 100 MHz clock used for the classical pipeline and most of the logic of the architecture
3. clock_50MHz - a 50 MHz reference clock
4. clock_50MHz_pi_shift - another 50 MHz reference clock, 180° phase shifted from clock_50MHz
5. pll_locked - a 100 MHz reference from an on-FPGA PLL (phase locked loop)
6. pll_50MHz_Locked - a 50 MHz PLL reference

To reduce the clocks that need to be fed to the chip, multiple clocks can be generated from a single source clock. This is done by employing a simple frequency divider, which cuts the output clock frequency by half from the input clock.

Figure 3.1: Simple Frequency Divider



A flip flop with its QBAR output tied to the D input, and the input clock fed as the flip flip clock, as shown in Figure 3.1 works by changing the output of QBAR only at the rising edge of the input clock (one clock period), meaning that the output clock at Q only returns to its original value (one period) after two rising edges of the input clock (two periods). Hence the input clock is halved for the output clock. Only one of these units is needed to make a 100 MHz clock from the 200 MHz clock, and two can be daisy-chained (or connected in tandem) to make a division by four, creating the 50 MHz clock and the phase shifted 50 MHz clock (from the QBAR output). This is employed to produce the 50 MHz PLL reference from the 100 MHz PLL reference as well. In the end, six input clocks are reduced to two.

3.2.2. Programming Interface Replacement

The largest pin users in the original CCLight design are the two Advanced Extensible Interface (AXI) modules. These interfaces were used to program to two separate parts of the architecture: one for the instruction cache and the microcode unit, and the other to control registers of the CCLight. Each of these uses about 40 signals, but some are parallel and can be expanded to cover any address or data size. This stacks up to 480 pins for both AXIs.

To reduce the pins further, the obvious solution is to replace the AXIs entirely in favor of a completely different communication scheme. There are many protocols available, but to decrease the number of pins the most a serial communication protocols must be chosen. The most popular serial communication protocols are:

- I^2C (Inter-Integrated Circuit)
- SPI (Serial Peripheral Interface)
- USART (Universal Synchronous and Asynchronous Receiver-Transmitter)
- SSI (Synchronous Serial Interface)

In the end, the SPI protocol was chosen for its versatility, low number of required pins, and simplicity. The interface uses 4 pins, which brings the new total for programming the CCLight to 8 pins, a reduction of 472 pins.

However, this pin reduction comes at the cost of programming speed and reliability. The AXI protocol uses parallel channels to communicate multiple pieces of data on parallel buses at once, such as the address and the value for an instruction in the instruction cache. Furthermore, the interface has reciprocated read channels so that the parent programming device can read what the child device (CCLight in this case) received, meaning that an error in sent data can be realized and corrected quickly. On the other hand, SPI has to communicate all data as one word on a single bit channel, which means the word is sent one bit at a time. SPI does have a channel for communication from child to parent, but this is also a single-bit channel, so error detection is possible, but slower than for AXI. Since the CCLight executes the program loaded to it in an autonomous fashion, in principle it doesn't matter how long it takes to load the program - only the speed of the outputs to the quantum hardware and reading in the measurement results are critical in speed and latency. This time only affects the user's wait time for the start and result of an experiment. Even with SPI the load up time will still be in the order of hundreds microseconds or tens of milliseconds, which is still not humanly perceptible.

3.2.3. Input/Output Reduction and Serialization

The other large pin user is the outputs that interface to the AWGs and VSM. This includes 16 pins to the VSM, 1 pin to toggle an LED, and 109 pins to control the AWGs.

Not all of these pins were being utilized in the design however. Specifically, the `DIO(3-5)_DIR` pins (12 total), and `DIO(3-5)_DOL` pins (3 total) were not actually loaded to anything within the CCLight design; thus they were discarded from the ASIC design. Furthermore, the LED toggle pin was removed from the design.

Next, a reduction of the remaining parallel output signals to the AWGs and VSM were considered. These signals on the original design were: `B_Trigger` (16 bits), `DIO1_out` (7 bits), `DIO2_out` (7 bits), `DIO3` (32 bits), `DIO4` (32 bits), and `DIO5` (16 bits). The DIO signals were all pushed at a frequency of 50 MHz, and the `B_Trigger`, clocked at 200 MHz, went through the DDR output module for a final frequency of 400 MHz. This means the CCLight was designed to push out 7.9 GBit/sec - a huge amount of data - to drive 7 qubits.

Clearly the specification for 110 parallel pins to push these signals cannot be achieved in the ASIC design, so another pin reduction scheme needs to be employed. The only viable option here is to serialize these outputs; in other words, push the bits of the output codewords sequentially on common pins instead of at once on unique pins. Of course, these bits must then be pushed at a higher frequency so that the resulting word still comes at the specified rate (50 MHz). The CCLight design provides a unique opportunity to achieve this, as it provides a 100 MHz and a 200 MHz clock that could be utilized for serialization. In this way, sending half the bits at twice the frequency or a quarter of the bits at four times the frequency is possible. The first option would need 56 pins, while the second option would need 28. With the very tight pin number constraint on this design, the latter option was taken for the design. This, along with the two output toggle pins, adds up to 30 pins needed for the output signals.

The cost of this choice is the need for an off-chip deserializer. There are some off-the-shelf shift registers available that meet the requirement for parallelizing 4 bits of data at 200 MHz. Furthermore, an FPGA can also be used to translate the codewords into the form they are needed in. Either of these options however add some latency to the execution of a quantum instruction from the CCLight to the AWG; for the shift register option, this number is usually around 10 ns or less.

Lastly, the input pins were considered. These are pins that give the measurement data from the qubits via Ultra High Frequency Quantum Analyzer (UHFQA). These pins include `DIO1_in` (7 bits), `DIO2_in` (7 bits), `DIO1_Toggle_in` (1 bit), and `DIO2_Toggle_in` (1 bit). Additionally, some inputs corresponding to the output AWGs were considered, namely `DIO(3-5)_CLKI` (1 bit each). It was found that the signals `DIO1_Toggle_in`, `DIO2_Toggle_in`, and `DIO(3-5)_CLKI` were not actually loaded to anything within the CCLight design, so they were simply discarded. The `DIO_in` signals could have been further reduced in the same way the DIO outputs were reduced, i.e. serialized, but this choice would force the use of even more external shift registers (parallel-in, serial-out this time), and possible complications interpreting the data internally. Besides, at most the pins could be decreased from 14 to 4. Hence, these signals were left in their original parallel configuration. This results in a small overall reduction for this group of signals, from 19 pins to 14.

3.2.4. Result

Ultimately, the goal of reducing the 625 pins to less than 72 pins was achieved. First the 6 clocks were reduced to 2. Next, the programming pins were reduced from 480 to 8. Then the output pins were reduced from 126 pins to 30. Finally the inputs were reduced from 19 pins down to 14 pins. This adds up to 54 pins in the design, which leaves a maximum of 18 pins for power and ground and any debugging pins that may be desired.

3.3. Functional Verification

The initial design and the redesign were both verified using a total of 35 benchmarks. These benchmarks were all written by hand in eQASM and designed to test various functions of the CCLight. While this number is acceptable for an FPGA design, it is not sufficient for an ASIC. With an FPGA design, if there is a flaw in the design, the design can simply be redone, tested, and reloaded to the FPGA. There is no major cost when the design doesn't work, only a loss in time to fix and redeploy it. On the other hand, the ASIC cannot be changed after being manufactured – the design must be right. Otherwise, the process of taping out and manufacturing a new design must be redone, costing months and thousands of dollars. Thus, there is a need for more robust testing of the design before tape-out.

Typically, a processor would be tested by running a program on it, then checking the data memory and comparing that to the expected data. However, the CCLight processor is a unique case. The fundamental principle of the CCLight architecture is to issue specific codewords at very precise timing points. Thus, the verification of the ASIC design must be done by comparing these outputs to the expected values with respect to time. In this case, the ASIC outputs can be compared to the outputs of the original design via waveform comparison in the QuestaSim® tool to verify the correctness of the design, as done in section 3.1. In this way, it is assumed that the original design is completely correct. Since the design was already validated in the field with actual qubits [6], this assumption is valid.

3.3.1. Requirements for Functional Verification of the Design

Before any new benchmarks can be made, some requirements need to be defined. In other words, some goals for the benchmarks need to bet set. For the CCLight ASIC, it is already assumed that the original design is functionally correct. However, it needs to be verified that what is in the ASIC corresponds to the specification of the FPGA design. This means not only verifying execution of instructions, but assuring that key elements that were changed for the ASIC, especially macro blocks, function as expected. The requirements for the benchmarks are defined as follows:

- Test every instruction (classical and quantum)
- Sweep as many qubit masks as possible
- Use each register slot within the classical pipeline
- Test maximum and minimum program size

With these goals, the benchmarks can be formulated.

In the CCLight, there are two kinds of instructions: classical and quantum. The difference between these two categories is primarily in which pipeline of the CCLight it is executed. This difference is very important when verifying the CCLight, as only the quantum instructions can be seen directly on the outputs. This means that any benchmarks made to test the classical instructions, if it is intended to be seen on the outputs, must be augmented with quantum instructions. Hence, these benchmarks will be far more complex to make. Therefore, it was decided that the classical instruction benchmarks and the quantum instruction benchmarks would be made separately.

3.3.2. Creation of the Benchmarks

Classical Benchmarks

Due to the complexity of making classical benchmarks, meaning they must be made by hand, the task of making them was shelved in favor of building the testing infrastructure and the more critical quantum benchmark set. The benchmarks will be based off the architecture of one of the original 35 benchmarks, `msmt_loop_trigger`. Each one will be varied in the qubit masks declared, and the carrier quantum instruction(s), but kept the same basic architecture, most importantly that the classical instruction under test will set a delay interval for the quantum instructions being executed. In this way, the result of the classical instruction can be seen on the CCLight outputs.

Quantum Benchmarks

Unlike the classical benchmarks, the quantum benchmarks can be more straightforward to create. Again, the architecture of these benchmarks were based on that of one of the original benchmarks, `msmt_trigger`. However, in this case, the nature of the quantum instructions provide a unique opportunity to create the contents of the benchmarks randomly. That is, the instructions all either want to operate on single qubit masks, or two qubit masks. In addition, the result of every quantum instruction is a codeword and timing point going to the respective FIFOs, to be outputted at the appropriate time; there is no internal feedback to worry about in this case. This means no special consideration needs to be taken to choose masks with respect to the chosen quantum instruction, except that it needs single qubit masks or two qubit masks (and if two instructions are to happen at the same time, that they don't both issue to the same qubit). Therefore, a program can be readily made to write benchmarks with randomly selected qubit masks and quantum instructions to execute on them. In this way, a huge number of quantum instruction benchmarks can be made quickly, and more importantly more potential discrepancies between the FPGA and ASIC designs can be caught. Ultimately, most of the goals for the verification of the ASIC design are achieved through the quantum benchmarks and the generator used to create them.

To make all of the quantum benchmarks, a Python 3 script was written to automatically generate quantum benchmarks with the same fundamental architecture, but different qubit masks, time intervals, quantum instructions, and lengths. Each of these parameters is created using the native python random number generator (and the related shuffle function). These randomization functions yield a uniform distribution by default [25]. The benchmark generator script, given in section A.1 works in the following steps. It stores the information about the possible quantum instructions, qubits, and qubit pairs in lists, which are then randomly plucked from later. First the script decides how many single qubits the benchmark will have. If there is enough to make a valid pair of qubits (3 qubits) then it will decide how many qubit pairs it will have. It then selects from the possible qubit pairs which is determined by the qubits chosen in the first step. It also makes masks of multiple single qubits based on the number of single qubits by taking the full list of available qubits in the benchmark, taking a subset of the list of a random length, then shuffling that list. Next the script decides how many wait intervals it will keep, and lastly how many quantum operations it will have, which is randomized between 1 and the maximum number of operations that can be placed into the benchmark. The benchmark must be bounded up to 4096 lines, so that makes the maximum number of operations equal to:

$$maxOPs = \frac{4096 - numSMasks - numTMasks - numIntervals - 3}{2}$$

Where the 3 comes from the branching instruction and two `NOP`s at the end of every benchmark. Due to the unusual behavior of the rounding function within Python, this is approximated with $maxOPs = 2048 - numSMasks - numTMasks - numIntervals - 3$. Finally the script can begin writing the benchmark file. First it declares all of the masks it chose previously, and assigns random values within the valid range to the chosen set of interval registers. Next it begins the loop that will contain the quantum instructions and intervals. On each loop of this sub-routine, first it writes a `QWAITR` instruction with random interval register selection, then decides the quantum instruction(s) for this line. First it must decide if it will do one quantum instruction this line or two. If only one, then it merely decides (when there are qubit pair masks) if it will pick a single or two qubit instruction, which is then selected from the appropriate list at random, then assigned to an appropriate mask at random. In the case of two quantum instructions on a line, the same decisions must be made except now the script must choose (when possible) if the first, second, or both instructions will be a two qubit instruction. It also has to take care that the qubits chosen in both instructions in the line are to exclusive qubits; the script will pick one mask and then iterate until it chooses one that does not match the first. This sub-routine is repeated until all quantum operations are written. Finally it writes the branch and two `NOP` instructions, completing the benchmark.

This entire process is computationally short; the script can make roughly 1,000 benchmarks per minute. For most of the parameters the script randomizes, number of single qubit masks, number of interval registers, and number of quantum operations, a uniform distribution can be observed on large samples (thousands) of benchmarks. For the number of two qubit masks and number of multiple single qubit masks, a decaying distribution to higher values (like a geometric distribution) is observed, due to the random number within a random number which generates these values. Ideally the tests would have a Gaussian distribution centered around the expected range of values (similar to Monte Carlo analysis) but since not enough is known about the real experiments performed with the CCLight, a uniform distribution to the created benchmarks will give the best fault coverage.

3.3.3. Running Verification on the Design

To perform the testing, 1,000 eQASM programs were created from the generator. These benchmarks were placed into a single folder. Next a simulation script was made to set the simulation time to always be 10 ms. This is a long time interval, but is approximately equal to the longest allowed wait interval allowed in the QWAITR instruction ($[2^{19} - 1] * 20ns = 10.5ms$), thus guaranteeing that most of the benchmarks created will produce outputs within the simulation timeframe. The simulation script also made a generic rename of the resulting waveform (.wlf) file, from vsim.wlf to result.wlf, so that the script made for automating the benchmark execution process could know when the simulation was finished. The automation script, written in PowerShell, went through each item in the generated benchmark directory, called the assembly script on it, then launched the simulation with scripted mouse clicks. The script then waits for the creation of the file result.wlf, at which point it can rename the file to indicate the benchmark that was tested, close the simulation window with a keystroke, and repeat the full procedure for the next program in the directory.

Ultimately, the task of running 1,000 eQASM programs on the CCLight FPGA design in HDL Designer/QuestaSim for the time interval of 10 ms proved a huge computational task for the hardware platform. In the end it took 9 days (twice the ideal run-time which assumed 7 minutes per benchmark, giving 5 days of run-time) to produce the results. Hence the same procedure has not been done for the CCLight ASIC design, which will also need to have the comparison done in the same time, increasing computation time still. First, the procedure will need to be redesigned to run on the platform hosting the CCLight ASIC design, namely UNIX and QuestaSim. The procedure of assembling the script in each iteration will have to be skipped; the assembly script is not compatible with UNIX. Instead, the assembled binary files created during the FPGA runs, which will be compatible to the ASIC design, will be used. Then after the simulation runs for 10 ms or perhaps longer, the result will need to be compared to the right FPGA result or reference waveform file. One major hurdle to this will be taking into account the difference in start time between the FPGA and the ASIC, since the SPI programming takes approximately 10 times less. Other than this delay in the start, the waveforms should look identical. Until these problems are solved, a preliminary chip will have to be verified by 'eyeballing' the two waveforms to confirm the same output codewords at the same intervals.

3.4. Synthesis

With the CCLight HDL design for the ASIC implementation functionally verified, the process of building the actual ASIC can begin. The first step in the process of converting an HDL behavioral description of a design into an ASIC is called synthesis. In this step, all of the constructs of the HDL code, such as if statements, case statements, and arithmetic operations are converted into a circuit described with standard cells, such as logic gates, flip flops, and multiplexers. The input to this step is a behavioral HDL description, a set of timing constraints (i.e. the target clock frequency), and the output is a netlist of standard cells. In later steps this netlist will be placed into a physical layout of a chip, which is what is ultimately sent for manufacturing. At this point in the design process, the clock frequency can be extracted, as this is directly related to the performance of the standard cells that make up the resulting circuit.

3.4.1. The Genus Tool

The tool used to synthesize the CCLight ASIC design to the target TSMC 40 nm standard cell library was Genus®. Genus® is the most recent branding of the Cadence flagship synthesis tool, as part of their EDA suite including the other tools used in creating the CCLight ASIC (Innovus®, Virtuoso®). Until recently this tool was branded as RTL Compiler®, meaning that a lot of information about the tool from forums comes from discussions about the previous branding, although there are some discrepancies between the two tools. A significant amount of time was spent merely learning how to use the tools to build the ASIC, just as was done with the CCLight development environment.

Genus® is a console-based tool, meaning that it is best used with scripts. It runs mainly proprietary commands that perform the most important functions such as compiling the behavioral code, synthesizing the HDL to generic cells, and retiming, as well as TCL, shell, and SDC commands. Information about the proprietary commands is given in a command reference with the tool's documentation. Genus® also has a GUI for schematic viewing, which is very convenient for verifying correct synthesis of the blocks within the design to standard cells, as well as the interconnect between them.

3.4.2. Genus Workflow

Although the use of a synthesis tool may seem linear, i.e. an HDL design and some timing constraints are entered into it, and the tool produces a netlist which can be verified and then used within further ASIC development tools, it is actually a feedback loop and an iterative process. These inputs are given, the netlist and timing reports are produced, but then the report must be analyzed, and the netlist verified to match the input HDL behavioral description. Hence the HDL design and/or the timing constraints must be altered, and the process repeated until a working result is achieved.

The CCLight HDL design has an initial set of timing constraints, meaning the clock frequency used in the FPGA design. Due to the direct mapping of the logic to the corresponding hardware blocks in the ASIC design, and the control to place these blocks suitably close to each other, the ASIC design is virtually guaranteed to meet the timing constraints from the FPGA, and even exceed them. However, there is no guarantee that the behavioral description will be compiled and synthesized correctly. The Altera® tools used to compile and load the original design may be more forgiving than Genus®, i.e. the latter may not allow some HDL constructs present in the CCLight design. For example, the Genus® tool will not compile VHDL process statements that are sensitive to multiple clocks.

HDL Synthesis Issues

Unfortunately there were many problems found within the CCLight HDL during the synthesis process. First, many modules within the CCLight FPGA design were given 'initial values' for some signals. This means that at start up, these signals would start with the specified value, instead of being undefined. This is valid for an FPGA, as they typically employ some hardware that allows blocks within the design to be initialized in this way. However, this is not available in the ASIC design. Instead, the signals must be reset with a reset signal at start up. For the most part, the initial values weren't needed, especially in the modules within the classical and quantum pipelines, as the initial undefined values are flushed out as valid data propagates through the pipelines. However, the main reset signal to the pipelines was not defined at start up nor until it was explicitly dropped by a register write, as the block defining that signal from the external inputs, `QuMALwAXIComMan`, too had initial values within its design. This reset signal was defined with a delay chain connected to the 'enable' register as an active-low reset, meaning that pulling the reset signal to a logic low or zero value enables the CCLight. The same issue and architecture was present for the 'run' register. After changing the sensitivity lists of the module's processes to be sensitive to the external reset, and changing the clock from the SPI clock to the pipeline clock, the internal reset signal was finally defined from start up, allowing the majority of the internal modules to start up properly. This change allowed the delay chains in the processes to be reset from the external reset, instead of propagating undefined values constantly, and to have a more continuous sensitivity as the SPI clocks are by definition turned off (not switching) between transmissions, which allows for data delay from the SPI modules (re-parallelization) and is more in line with the original design with the always-on AXI clocks.

Second, some of the modules in the CCLight design were missing resets on internal signals. This was a particularly difficult bug to fix, as the only apparent issue is that valid signals are sent to the module, but the outputs remain undefined. Under further inspection of the modules from the netlist in the QuestaSim® simulation tool, none of these internal signals are ever defined. When the HDL is simulated behaviorally, the QuestaSim® tool compiles these signals to be initialized, even when initial statements aren't used, so these flaws in the HDL architecture are hard to catch beforehand. In all cases, the modules had to be carefully rewritten so that these internal signals would have a reset, without changing the fundamental logic.

One example of a module with this issue is the unit `MjuOpAlign` (within `MicroOperationJoin`). The original firmware is given in section A.2. Compare to the synthesis-compatible version of the firmware given in section A.3. The first thing that is different is the lack of initial values on the synthesis compatible version (lines 53-59 in both versions). These assignments have simply been commented out. The critical difference however is on line 78. The output signal `MJ_MajorTimestamp` could perhaps go without being reset, but `i_MajorTimestamp` must have a reset, because it appears in the conditional expression of a concurrent assignment. The result of any conditional expression with at least one undefined value is an undefined output. This undefined signal in turn propagates the undefined value to `i_vec_VerConf`, `i_vec_MicroOperation_qubit`, `MJ_MajorTimestamp`, `QMB_vec_MicroOperation_qubit`, and `QMB_VerConf` - nearly every output signal from the module. This is why this module and some others gave constantly undefined outputs. In each case the issue had to be fixed carefully so as to not reset the signals to the wrong value or change the logic of the module.

Third, the FIFO module was not functioning after synthesis. After some inspection in the QuestaSim® tool, it was determined that the issue was due to the way the internal memory was handled within the FIFO. More specifically, the writing to the memory was controlled with an arithmetic loop and exit statement to determine the position of the input data in the queue. This block did not function at all, as all outputs were always unchanged from the value after reset regardless of any input, along with other indication signals in the design such as `wrused`, which indicates the number of occupied slots in the queue with respect to the write clock. Therefore the entire process was rewritten with a counter, and ultimately the entire module as changing this functional block had ramifications on the logic of some indicator signals – mainly how complex or simple their logical blocks needed to be.

Fourth, there were timing issues in the design. There was negative slack on one particular path, specifically within the DDR output module. The DDR output module was put into the design to give the CCLight FPGA design a 2.5 ns resolution signal to the VSM, on hopes of achieving a signal with a sharper rising and falling edge. This negative slack means that the data from one register doesn't have enough time to go through the logical path (combinational circuit path) and become valid on the next register within the clock period. However, the implementation of the DDR output within the CCLight ASIC design will have no effect on the precision on the final output signal. This will be dictated by the performance of the pad cells driving the signal off the chip. Hence the module was removed from the design, eliminating all remaining timing issues.

Final Script

Ultimately, the CCLight design was successfully synthesized after many iterations using the script given in section A.4. The script is long as it has to pluck every necessary file from the source directories, since they also contain other files that are not used in the design. The first section sets attributes (or settings) for the synthesis tool. For example `set_db auto_ungroup none` forces the tool to keep the abstracts of modules within the design, instead of completely decomposing them to standard cells where possible. This is useful for debugging purposes. The next section loads all of the CCLight source files into the program and compiles them. Next all of the clocks within the design are declared, then some pre-synthesis reports on the design are created. Finally the design can be synthesized first to generic gates, then the standard cells of the TSMC 40 nm library. Lastly post synthesis reports are written, and the resulting netlist and constraint files can be created.

Results

The resulting netlist has the characteristics shown in Table 3.1:

CCLight Post Synthesis Report		
Item	Value	Units
Standard Cells	53,782	integer
Cell Area	156,268.526	μm^2
Total Area (Cell + SRAM)	303,131.730	μm^2
Slack, 200 MHz Clock	3,461.2	picoseconds
Slack, 100 MHz Clock	2,570.8	picoseconds
Leakage Power	566.688	μW

Table 3.1: Excerpts from the CCLight Post-Synthesis Reports

The fist value shows the number of standard cells that CClight design was synthesized to. This includes logic elements like XOR gates and sequential elements like flip flops. The next number gives an area estimation for these cells. This number could be considered as a minimum area, as routing signals and clocks properly in the real chip core may take more area than this. The third number is simply the second plus the area of the SRAM, which as a macro is fixed in size. The SRAM will also need a halo around it for its own power routing, which will add to the occupied area in the chip core. The next two numbers indicate the slack for the source 200 MHz clock, and the pipelines' 100 MHz clock. The slack of the 200 MHz clock indicates that it could actually be doubled (period of 5,000 ps but only 1,539 ps needed), but the 100 MHz cannot be reduced in the same way. Analyzing the reports further shows that the critical path(s) happen on certain pins to the instruction cache/SRAM. If the design is to have the clocks increased, these paths will need to be redesigned. This amount of slack is very generous for routing purposes. Lastly the leakage power figure, which again is a minimum estimate, and not the complete picture. This number is heavily dominated (84%) by the SRAM. The core power will certainly be much higher when the power routing and design placement are done in the chip and these factors taken into account.

3.4.3. Netlist Verification

To test the synthesized netlist, a testbench was developed to load the benchmarks to the instruction cache, execute the benchmarks, then compare those results to the verified results of the pre-synthesis CCLight design. This testbench is virtually the same as that for the pre-synthesis CClight design. The testbenches were derived from the original one used for the FPGA testing, replacing AXI writing protocols with SPI, and generating source clocks and other input signals from within the single testbench file.

Without the availability of the increased benchmark files and an automated comparison tool for more comprehensive verification of the design, as described in section 3.3, a few of the original benchmarks (`br_test`, `flux_trigger`, and `msmt_trigger`) were tested on the post synthesis netlist. The results of them, inspected visually, showed that the core of the CCLight ASIC design gave the same outputs as the CCLight FPGA design. This result is promising enough to proceed with building a preliminary chip out of the netlist.

3.5. Place and Route

With a verified netlist, the chip can finally be built. This means the standard cells in the netlist can be placed into the core of a chip, power routed to it, clock tree synthesized, pad ring built around it, etc. Once these steps are complete, a layout design is the resulting output. Although it won't be quite ready for manufacturing, this is the basis for what is ultimately sent to the TSMC foundry for fabrication.

3.5.1. The Innovus Tool

Like Genus®, the Innovus® tool is a re-branding from an older tool, in this case the place and route tool SOC Encounter®. Unlike Genus®, Innovus® has a GUI that is meant to be used in the standard workflow. This workflow involves importing the design and any macros desired for the chip, and assembling them into a floorplan within the chip core using the GUI. This is also where power routing and pad ring assembly can be conveniently done. After this is successfully done, the chip design data can be saved for future loading so that these steps do not have to be repeated each time (optionally the exact commands to perform these tasks can be extracted from the Innovus® logs). Next, the console-side of the tool can be used to place and route the actual design within this chip footprint. Namely, loading the design and process/standard cell/macro information, then placing the design, synthesizing the clock tree, and routing the design. Before extracting the Netlist for functional verification and GDS for DRC/LVS checking, the design can be tested in-tool for meeting timing constraints and passing DRC (albeit an abridged check; this will come in section 3.6).

3.5.2. Innovus Workflow

Setup

In order to load the CClight netlist into the place and route tool, two files need to be created. The first is a Multi-Mode, Multi-Corner (MMMC) file (for physical verification of the chip design) and a pin assignment file (to declare where the pad cells will be placed around the edges of the chip.

The MMMC file can be easily made in a prompt from within the prompt to initially open a design in Innovus®. This simply involves linking together the standard cell library data about the RC corners, Process, Voltage, Temperature (PVT) variations, and timing for best, worst, and typical cases. Then when the chip is physically verified, these values are swept multi-dimensionally to give a more accurate estimation on the real performance of the chip post fabrication.

The pad cells must be given an assignment around the edges of the chip in a .io file, which can be created using the GUI, or can be arranged by manual pin management and file manipulation; the latter method was used. 54 pins were needed in the design for various I/O and signals, and 14 pins were given for chip power, in order to get as close to the 1 power/ground pair per 8 I/O pins rule-of-thumb as possible. This totals to 68 pins. With 68 pins, and an aspect ratio of 0.5, the top and bottom sides will get 12 pins, and the left and right sides will get 22 pins to keep the pin distribution even on all sides.

Floorplanning

Now the floorplan can be built in the GUI. First, the chip must be sized - $1000\mu m$ by $1500\mu m$. Then, some space for the power routing needs to be given between the core and the pad ring. Since this design has an SRAM macro block, a slightly larger than typical power routing gap is given, $25\mu m$. Next, the macro block must be placed within the core of the chip. In principle the macro can be placed anywhere within the core, but it is ideal to place it in such a way that not many signals need to be routed around the block to reach pins on the far side. Since all of the pins on this particular SRAM macro appear on the bottom and left side, and the CCLight netlist cells will be placed around the center of the core (by default of the tool), the bottom left corner of the macro block is placed at the center of the core. Then power rings can be built into the gap, and power stripes placed within the core. The rings are made in metals 5 and 6 with $4\mu m$ width, and the cell power stripes are made in metal 1. Finally the power routing for the SRAM can be done. This is done as a halo around the SRAM in metals 4 and 5, with metal 4 stripes placed vertically and metal 5 stripes placed horizontally over the SRAM block (as specified in the SRAM datasheets). With the power rings and stripes in place, the routing from the rails to the pad cells is done. To complete the floorplanning, IO fillers are added, and well taps are placed in the design at intervals of $32\mu m$, which is a rule-of-thumb figure.

Netlist Placement, Clock Tree Synthesis, and Routing

For this stage of the chip development, the chip is built using a script. The result from the floorplanning stage is loaded, then the remaining steps to build the chip within the Innovus® tool are performed, with the design saved at intermediate points along the way. First the chip footprint is loaded and the CCLight netlist is loaded into the tool. Then the power domains are set using a CPF file. Next the design is placed within the core. At this point, the physical cells are visible within the core of the chip. Afterwards some optimization is performed on the placement of those cells. Then the clock trees are synthesized, which means that the clock net is routed and given enough repeaters that the clock maintains its integrity through every module it is routed to. Even with the large slack reported from Genus®, the time constraints of 200 ps for setup and 100 ps for hold time were added to keep the clock tree synthesis conservative. Again, some optimization on the placement of the cells in the design were performed, then tie high and low cells were placed, which add the constant logic level signals where needed. Next, the design is routed, which means the interconnect between cells is finally added, and once more placement optimizations are performed. Once the filler cells are placed into the core, the design is fully constructed (aside from bond pad placement, which must be done in section 3.6 because those models don't exist for Innovus®). Finally, the CCLight ASIC layout has been built, as seen in Figure 3.2.

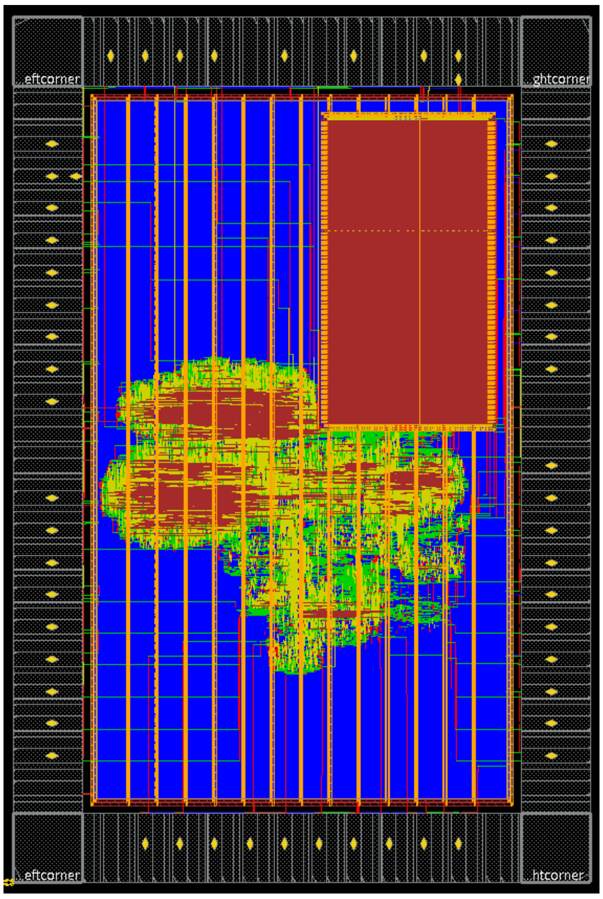
3.5.3. Verification

First some in-tool verification is done to assure the timing constraints are met after place and route operations. Next, in-tool DRC is done to assure the design can even pass a basic set of design rules before going through more rigorous testing in section 3.6. Functional verification would happen in the RC extraction phase, which goes along with the DRC/LVS steps. Since this is only a preliminary chip, and this is a complex and time consuming process, it has not yet been done for a CCLight ASIC. This step must be saved for when a fully verified netlist can be created and used in the chip design.

3.5.4. Results

The resulting chip has the characteristics shown in Table 3.2:

Figure 3.2: The CCLight ASIC Post Place and Route



For typical digital chip designs, a core utilization of about 70% is desired. In this way, the chip core can be filled with cells, macros, etc while giving enough room for routing, and not leaving much effectively to waste. In this case, the design size is dictated by the pins, which makes it a 'pad-limited' design. Although a core utilization of 36.8% is not ideal, it is the only option without adding things like decoupling capacitors, JTAG, etc to the chip. With the 54 pins on the design, and 14 power/ground pins added, the grand total pins for the CCLight ASIC is 68. 6 power pins were given to power the cells and the SRAM, and 8 were given to power the 54 pins on the design. Ideally there would be 12 such pins, to keep the to 1 power/ground pair per 8 IO pins rule-of-thumb, but the pin limit of the package (CLCC 68; the CLCC has been verified at cryogenic operation, hence its choice here) has been reached, and the absolute pin limit of the die nearly has too. Furthermore, if it is assumed that the 8 SPI pins will not be active when the DIO etc outputs are, which is true, and neither will the DIO inputs, which may or may not be true based on testing conditions, the 8 IO power pins will indeed be sufficient. Power analysis was run for this report with input activity and sequential activity both set to 0.5 (from 0 to 1).The power consumed by the chip is mostly internal, with 75.74% of the total power usage from the IO. Lastly the timing of the chip on the 100 MHz clock, which had the least slack in the post-synthesis report, was verified and even shown to have more slack in layout than before (now 3,920 ps).

3.6. Layout/Physical Design

The last step in creating an ASIC out of the CCLight design is finalizing the physical layout, and then verifying that the design can be fabricated, or design rule checking (DRC), that the physical design matches the post-synthesis netlist, or layout versus schematic (LVS), and that the final circuit meets specification. Once these tests are passed, the chip is ready to be sent to the TSMC foundry for fabrication.

CCLight Post Place and Route Report		
Item	Value	Units
Die Area	1.5	mm^2
Core Utilization	36.8%	percentage
Total Pins	68	integer
Core Power Pins	6 (3 VDD/VSS pairs)	integer
IO Power Pins	8 (4 VDD/VSS pairs)	integer
Power Dissipation	129.5	mW
Timing Closure	Passed	—

Table 3.2: Chip Layout Data

3.6.1. The Virtuoso and Calibre Tools

The last step to complete the building of the CCLight ASIC will be to add the bond pads. The bond pads are the metal contacts that are bond wired to the epoxy package containing the silicon chip, and electrically connected to the pad cells inside the silicon. As determined in section 3.2 the bond pads must be placed in a staggered arrangement in order to fit all of the needed pins in the limited silicon area provided. Hence every other bond pad will be placed a full bond pad length, $55\mu m$ from the pad ring (and slightly farther than that to pass DRC) while the other half are to be placed directly over the pad cells they are connected to on the pad ring. Each bond pad will need to be carefully aligned to its corresponding pad cell; a resolution of $0.01\mu m$ is necessary for the grid of the Virtuoso® layout editor. All bond pads need to be connected with metal 6 wires and layer 5 vias. The cost of staggered bond pads becomes apparent at this point, as the bond pads left 'dangling' over the pad ring add area to the final chip – increasing the total die size to about $1.6\mu m^2$.

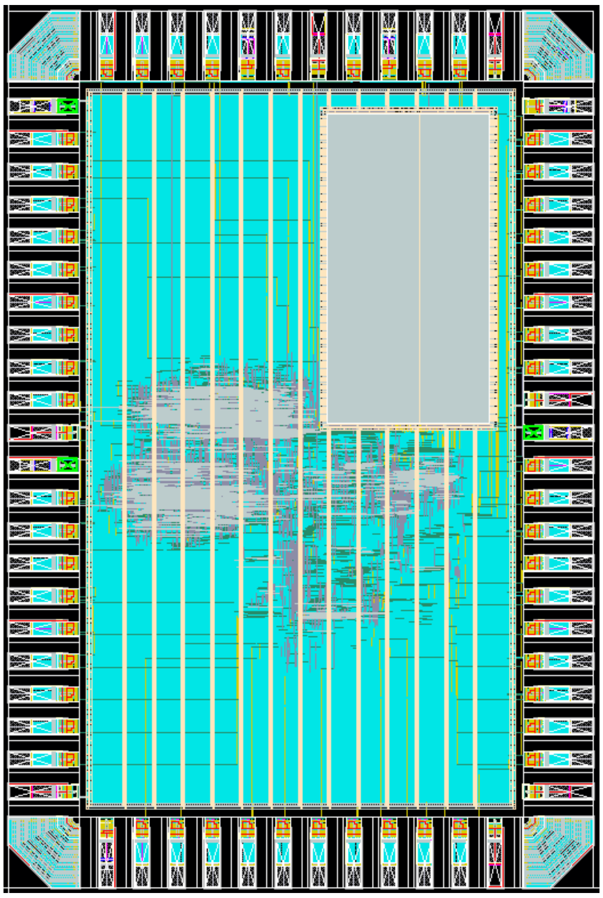
3.6.2. Passing DRC

Running design rule checks (DRC) verifies that the layout designed can be fabricated by the foundry. Hence it is imperative to have the most up to date DRC file (file that contains the information about what violates the fabrication rules) for the process, as the foundry constantly updates this information. Some DRC errors can be ignored, as they cannot be fixed by the designer anyway, such as the ESD errors often found on the pad cells. However, many of them reveal flaws in the layout design. In the case of a previous preliminary CClight ASIC, two meaningful errors were found. The first was a multitude of metal and metal density errors. Upon inspection, it was found that these errors only popped up around the bond pads. Unfortunately the wrong bond pads had been used for the 'hanging' bond pads in the design. With even more meticulous inspection, it was revealed that the bond pads were all slightly misaligned to their respective pad cells, as the layout editor did not have fine enough grid precision set. Both of these issues were quickly resolved, removing all related DRC errors. The second flaw was again related to metal geometry errors. This time the errors appeared on two power pad cells, and on visual inspection it was quickly realized that the pads were simply not able to connect to the power rails correctly because the two pads connections were crossing each other. So the simple solution was to flip the positions of the two pads in Innovus® and then bring the design back through the relevant steps. Ultimately this solution resolved all of the related DRC errors. In the end, the older CCLight ASIC was brought to the point of having no meaningful errors on the most up-to-date DRC file available. This paved the way for the CCLight ASIC presented in this thesis, shown in Figure 3.3, which managed to yield the same result on the first DRC attempt.

3.6.3. Passing LVS

After passing DRC, the next test for the chip is layout versus schematic (LVS). This means comparing the GDS layout to the schematic, which is converted from the netlist exported from Innovus® by the Virtuoso® tool. This step was performed on a previous preliminary CCLight ASIC. Converting the netlist to a schematic is typically a straightforward task. However, in the case of some of the pad cells, namely the digital power, ground, and IO pads, some of the power/ground ports are missing from the cell models when they are extracted from Innovus®. This causes errors for LVS that don't really exist in the design, as the ports and connections exist as they should in the GDS. To fix this problem, a small script was made to edit the netlist before schematic conversion by finding every instance of

Figure 3.3: The DRC-Clean CCLight ASIC in Virtuoso



the problematic pad cells in the netlist, and adding the missing ports to them. This solution fixed the problem and made the schematic import correctly, which could be seen in the schematic viewer, and from the LVS report. A copy of this script is given in Appendix B.

The next step in LVS cleaning was the the comparison results. Here, the CCLight gave three kinds of errors:

1. Incorrect Nets
2. Incorrect Instances
3. Property Errors

The first two are a result of the SRAM macro not being truly present in the layout; only the footprint from the metal layers is given by the foundry. This is done to protect the IP of the SRAM design. The information about the physical size and the ports is generally sufficient to design with at the layout level. Some information was given in the PDK to mitigate some of these false errors by editing the LVS rule file to treat the SRAM as a block box. Unfortunately, this only served to change what errors were showing up; the false errors were still there. In the end, this technique was not used, and the original false errors were waived. The last type of error, the property errors, gave the largest number of errors by far - 4,579. Upon visual inspection, these errors were all appearing on the pad cells only. With more investigation, it became clear that the issue was the schematic *extraction* having incorrectly parsed the transistors and even resistors' values incorrectly, even though the actual values present in the layout and the schematic being the same. This is a problem for all users of the PDK and pad cells, and not an isolated issue. To clean these false errors, another script was developed to take the extracted SPICE netlist from the schematic, and replace the incorrect definitions with the correct ones. The result was the removal of all false property errors, which left zero real property errors in the LVS report. A copy of this script is given in Appendix B.

4 Testing

The goal in this project is to replace the original CCLight FPGA implementation with the CCLight ASIC in place to show that a smaller and more power efficient custom chip can take the place in the Quantum Stack, and to run the Surface-7 qubits. Furthermore, the chip should operate at deep cryogenic temperatures, namely 4 Kelvin in order to show that this part of the Quantum Stack can not only be scaled down, but also brought closer to the qubits themselves, which typically operate in the milliKelvin temperature range [14] [15]. Since the CCLight drives complex hardware that cannot be brought down to cryogenic temperatures, there will need to be two distinct tests on the CCLight ASIC:

1. Room temperature tests – to prove that the ASIC can fit into the operating conditions and specifications of the FPGA

2. Cryogenic tests – to prove that the ASIC will still work under this condition and could drive the target hardware at cryogenic temperatures in principle

4.1. Testing Considerations

Due to design decisions for the ASIC made in section 3.2, some extra hardware is needed to translate the CCLight ASIC outputs to the form needed by the target hardware. Furthermore, the original CCLight FPGA needed some extra hardware to interface some target hardware units which the CCLight ASIC will need as well. More specifically, the CCLight ASIC needs 4-bit (or higher) deserializers operating at 200 MHz on each `DIO(1-5)` output, which was not needed for the FPGA, and a 2.5 V to 5 V voltage level shifter on the the `B_Trigger` output, which was needed for the FPGA. In total, 28 74LVC595A shift registers and 2 74LVC8T245 level shifters are needed for the CCLight ASIC to operate the target quantum hardware. Depending on the quality of the source clock and the requirements of the testing, one reference PLL 100MHz clock can be used, such as the CDCEL913.

In total, to run the tests on the CCLight ASIC with the target hardware (AWGs, VSM) a schematic and PCB with the ASIC, a control FPGA to program the chip (or an USB-to-SPI module), the shift registers, the level shifters, some single-ended to LVDS converters, voltage regulators, and break-outs to the AWGs and VSM needs to be designed and assembled. Furthermore, with the scale and complexity of this test system, some additional hardware may be needed to drive critical signals, such as clock generators and buffers. Ultimately the goal is to replace the CCLight FPGA within the larger system effectively, and the most critical aspect of the CCLight system design to consider is the timing of the signals to the quantum hardware, in terms of delay, integrity, and resolution. The CCLight ASIC and the quantum hardware need to communicate properly in order to perform experiments on the Surface-7 chip, which is the ultimate purpose of the CCLight project.

However, at cryogenic temperatures, some of these modules may not work. Neither will the quantum hardware (AWGs, VSM). Furthermore, the physical space allowed for the electronics is very limited [26] [27]. Therefore, it makes the most sense to design the cryogenic test system to be a bare-bones PCB, one that only contains the essential components, namely the ASIC, programming FPGA/module, and voltage regulator. In this way, it can be shown that the ASIC itself still functions at cryogenic temperatures, regardless of whether or not the peripheral modules can. Measuring the raw outputs of the ASIC with respect to time would be sufficient to prove this.

Since what is needed for the cryogenic tests is a subset of what is needed for room temperature tests, it could be possible to design a PCB system in which the modules necessary for cryogenic testing are lumped onto a daughterboard. This daughterboard could then be plugged into a motherboard containing the modules necessary for room temperature tests. In this way the design challenges from designing two PCBs can be condensed to one PCB system. However, a system like this may prove to be more complex than what its worth.

4.2. Proposed Room Temperature Tests

The first thing to test on the CCLight ASIC when it is returned from manufacturing is that it works electrically. This will mean powering it on and looking for some expected outputs even with nothing being programmed to the chip, in this case looking for a toggle signal from `DIO(1-2)_Toggle_out`. Once it is confirmed that the chip powers up correctly, the next thing to test is the SPI modules for programming the chip. The SPI module can be configured to repeat what words it reads on input to its output. In this way it can be verified that the SPI modules of the CCLight ASIC function as expected and the chip is ready to be tested functionally.

Ideally, the room temperature tests will go so far as physically interfacing with the target quantum hardware (AWGs, VSM). First, the CCLight ASIC needs to be verified to work with the full room temperature PCB system designed for it. This means verifying that the output signals of the CCLight ASIC can be distributed across the PCB correctly and interpreted by the shift registers, level shifters, etc correctly so that a target signal and timing specification can be met at the breakout points for the quantum hardware. To run these tests, some benchmarks with meaningful outputs and experiments that are easily understood will be chosen from the original set of 35 benchmarks, so that specifications for the PCB level can be derived from them. In total, 31 of the 35 original benchmarks can be potentially used for this testing (i.e. they have outputs to the quantum hardware). If the timing specification isn't met, it is possible to add buffers to signals that are too fast. This means adding a delay in the 1-10 ns regime, as many commercial modules are specified, which matches the resolution of the CCLight ASIC, which is the output signal period, 5 ns. Other signal issues can be resolved similarly (if not with the configurable delay within the CCLight design), the costs being physical area, execution latency, and redesign price and time. The critical point at this stage is getting the signals from the CCLight ASIC to be shifted to the needed parallel signals correctly.The challenge here is a lack of shift registers/deserializers that operate at 200 MHz, near the 2.5 voltage level of the chip IO, and take single-ended input. Especially the 200 MHz speed is a difficult specification to meet with off-the-shelf components.

The goal then for this functional system would be to hook it up to the quantum hardware, which is in turn running the Surface-7 chip. This would be the ultimate test for the CCLight ASIC, as the purpose of the CCLight is to control the qubits of the Surface-7. If the CCLight ASIC can successfully execute on the qubits and read measurement results into its memory, and respond to that input in a timely and correct fashion, the CCLight FPGA has been completely and successfully replaced by the CCLight ASIC.

4.3. Proposed Low Temperature Tests

The goal for cryogenic testing of the CCLight ASIC is to verify that it still works under the low temperature condition. As discussed in section 4.1, this is mainly due to the peripheral modules likely not working at cryogenic temperatures, the quantum hardware not working at cryogenic temperatures, and the whole system not fitting in the small physical testing space available for the experiment. The cryogenic test setup will then be a PCB with the minimum components: the CCLight ASIC, a driver/readout FPGA, and voltage regulator. The FPGA and voltage regulator must be chosen from what components have previously shown to work under these conditions, specifically the Xilinx® Artix-7 [28] and a handful of op-amps and discrete components to build a voltage regulator [29]. Due to the large cost in time to cool down and reheat the experiment, the FPGA will have to be programmed to test a burst of benchmarks at once, as well as collecting the data and communicating it back outside of the dilution (cryogenic) fridge.

For cryogenic testing of the CCLight ASIC, the key question is whether or not the SRAM for the instruction cache will work. SRAM is inherently sensitive to PVT/mismatch, especially at smaller technology nodes, due to its transistor density and leakage power [30] and the importance of V_T in SRAM operation and the higher sensitivity of the minimally sized SRAM cells to changes in V_T [31] [32]. Furthermore, mismatch is known to be an exacerbated issue at cryogenic temperatures [33] [34]; this adds up to a highly critical component at cryogenic temperatures. Ideally this component would be tested by sending instructions for it to store, then fetching them back to the FPGA to see if the data is unchanged. Since no structure in the design of the CCLight ASIC exists for this, a modified testing scheme will be performed. First, the SPI modules will be verified in the same way as in the room temperature testing. Then, some of the 31 viable benchmarks will be sent to the CCLight ASIC with the SPI configured to repeat the input back to the FPGA immediately. In this case, the benchmarks containing infinite branching loops will be the most revealing of the SRAM's performance. If the outputs are measured over a long enough period of time to see multiple loops though the benchmark, it can be seen on the outputs if incorrect instructions start to be executed, as the instructions coming out of the SRAM are no longer the same, and hence the outputs from the CCLight ASIC will change over time. This does not isolate the testing on the SRAM however; the core of the CCLight ASIC could cause a fault as well. However, the CCLight ASIC core is expected to be more robust to the cryogenic environment as CMOS logic cells are more robust to PVT [24], and the timing slack in the design with the nominal clock speeds was high, which gives a lot of room for reduced performance in the core.

Lastly, if the SRAM can be shown to work at cryogenic temperatures, verifying the functionality of the CCLight ASIC will not require much further testing complexity. This will involve again running the 31 benchmarks, SPI readout configuration not needed, and reading the outputs to the FPGA. The time length can be reduced to what is needed for one loop where branching is used or a standard time interval that is long enough for some outputs intended for the quantum hardware to be collected by the FPGA for all benchmarks. Then the FPGA can attempt to put the (what should be correct) serialized codewords back into their original parallel scheme as the shift registers would do at room temperature, and transmit them back to room temperature. If it can do that, the CCLight ASIC works at cryogenic temperatures.

5 Conclusions and Future Work

5.1. Summary of Work and Results

In this thesis, the Central Controller-Light (CCLight) was converted from an FPGA to an ASIC. First, the CCLight development environment was learned. Then the firmware of the CCLight was analyzed and the components needing replacement due to IP restrictions were identified. These components were replaced and verified within the design. Next the number of pins in the design was reduced by a factor of 10. A lack of sufficient benchmarks (QISA files) from the original set for verifying the ASIC design was identified, and a Python script to write 1,000 randomized benchmarks was created. Then the synthesis, place and route, and chip sign-off tools were learned. The CCLight firmware was synthesized to the TSMC 40 nm standard cell library, and the firmware debugged until the resulting synthesized netlist passed a set of benchmarks. Then the netlist was placed and routed into a chip layout, with timing [and power, functionality] verified. Some issues in the final chip verification processes (LVS) were identified and scripts were created to provide a fix. Lastly, DRC and LVS checks were performed and the layout design altered until all meaningful DRC/LVS errors were eliminated.

5.2. Conclusions

The conclusions of this thesis work are as follows:

- eQASM-based microarchitectures can be implemented on ASICs to increase performance and reduce core power consumption
- ASICs are the necessary implementation for digital control at cryogenic temperatures due to strict cooling power constraints
- The codeword-controlled quantum hardware approach in its current form, with tens of pins on the digital controller per qubit, is not scalable

5.3. Future Work

First, the chip built in this thesis needs to be finalized, fabricated, and tested. There is still need for further benchmarking of the ASIC design with respect to the FPGA design. The work for creating the randomized benchmarks for completing this task has already been done in this thesis; the setback is the large computational time needed to run the benchmarks on both the FPGA design and the ASIC design. Once this is completed, and any discovered bugs fixed, the process of getting the chip fabricated can be done. This mainly involves getting the chip DRC/LVS ”clean” i.e. there are no critical errors from either test. The chip as presented here is already ”clean” but any further changes from continued post-verification debugging may change this. Additionally there may be an updated DRC/LVS file provided by the foundry at time of final testing, which could potentially reveal more errors. After the chip comes back from the foundry it will need to be tested; the guidelines for doing so are discussed in chapter 4.

Second, further development to test the CCLight and other eQASM-based processors in a formalized manner needs to be done. One of the biggest gaps in the procedure for building an ASIC out of the

CCLight was the lack of robust tests for the complete picture of the design: the feedback, the PLLs, the programming mechanism(s), and of course the timing accuracy of the outputs to the quantum hardware. Ideally a tool or software package for an existing tool like QuestaSim® would be created providing robust testing and verification of all these aspects of the CCLight and future eQASM-based microarchitectures. This tool could easily incorporate the quantum benchmark generator created in section 3.3, and see it expanded to create benchmarks with different/more complex structures such as making the classical benchmarks too.

Third, a new eQASM-based processor could be designed and built for a spin qubit system. The specification of eQASM gives enough flexibility to create a design for spin qubits instead of the superconducting qubits targeted for the CCLight. The main difference would be the timing resolution needed. This could at the very least be achieved by an ASIC, if not an FPGA. The other issue is whether or not there are codeword-based control electronics available that can run spin qubits (superconducting and spin qubits have different fundamental frequencies for operation).

Fourth, an eQASM-based microarchitecture could be integrated onto a single chip with integrated AWGs/transmitters, along with other peripherals like the PLL and voltage regulator, in order to create a ”Qubit Control System-on-Chip”. In this scheme, a single chip would be able to do all of the control for a qubit chip. This would eliminate the issues encountered in section 3.2 as the interconnect is now within the chip and doesn't need the same care as when transmitted off-chip. Each final output becomes an IQ pair instead of 32 bits. This would also alleviate the issues with the timing of output signals between AWG and VSM outputs, since this issue was governed by the long physical distance between the VSM and the AWGs/CCLight. The only drawback would be that this resulting chip would have a fixed maximum number of qubits it could drive; this is the same for any eQASM microarchitecture however.

Bibliography

[1] R. P. Feynman, *Simulating physics with computers,* International journal of theoretical physics **21**, 467 (1982).

[2] P. W. Shor, *Algorithms for quantum computation: Discrete logarithms and factoring,* in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on* (Ieee, 1994) pp. 124–134.

[3] J. Preskill, *Quantum computing and the entanglement frontier,* arXiv preprint arXiv:1203.5813 (2012).

[4] X. Fu, L. Riesebos, L. Lao, C. G. Almudever, F. Sebastiano, R. Versluis, E. Charbon, and K. Bertels, *A heterogeneous quantum computer architecture,* in *Proceedings of the ACM International Conference on Computing Frontiers* (ACM, 2016) pp. 323–330.

[5] E. Charbon, F. Sebastiano, M. Babaie, A. Vladimirescu, M. Shahmohammadi, R. B. Staszewski, H. A. Homulle, B. Patra, J. P. Van Dijk, R. M. Incandela, *et al.*, *Cryo-cmos circuits and systems for scalable quantum computing,* in *Proc. 2017 International Solid-State Circuits Conference* (2017).

[6] X. Fu, M. Rol, C. Bultink, J. Van Someren, N. Khammassi, I. Ashraf, R. Vermeulen, J. De Sterke, W. Vlothuizen, R. Schouten, *et al.*, *An experimental microarchitecture for a superconducting quantum processor,* in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (ACM, 2017) pp. 813–825.

[7] X. Fu, L. Riesebos, M. Rol, J. van Straten, J. van Someren, N. Khammassi, I. Ashraf, R. Vermeulen, V. Newsum, K. Loh, *et al.*, *eqasm: An executable quantum instruction set architecture,* arXiv preprint arXiv:1808.02449 (2018).

[8] E. Garrido, L. Riesebos, J. Somers, and S. Visser, *User's Manual Feedback system for 3 Qubit bit flip code* (2014), cBox User's Manual.

[9] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach* (Elsevier, 2011).

[10] L. K. Grover, *A fast quantum mechanical algorithm for database search,* in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing* (ACM, 1996) pp. 212–219.

[11] P. Selinger, *Towards a quantum programming language,* Mathematical Structures in Computer Science **14**, 527 (2004).

[12] B. Ömer, *A procedural formalism for quantum computing,* (1998).

[13] S. Liu, X. Wang, L. Zhou, J. Guan, Y. Li, Y. He, R. Duan, and M. Ying, *$q|si\rangle$: A quantum programming environment,* in *Symposium on Real-Time and Hybrid Systems* (Springer, 2018) pp. 133–164.

[14] D. Riste, S. Poletto, M.-Z. Huang, A. Bruno, V. Vesterinen, O.-P. Saira, and L. DiCarlo, *Detecting bit-flip errors in a logical qubit using stabilizer measurements,* Nature communications **6**, 6983 (2015).

[15] U. Mukhopadhyay, J. P. Dehollain, C. Reichl, W. Wegscheider, and L. M. Vandersypen, *A 2× 2 quantum dot array with controllable inter-dot tunnel couplings,* Applied Physics Letters **112**, 183505 (2018).

[16] E. Lucero, R. Barends, Y. Chen, J. Kelly, M. Mariantoni, A. Megrant, P. O'Malley, D. Sank, A. Vainsencher, J. Wenner, *et al.*, *Computing prime factors with a josephson phase qubit quantum processor,* Nature Physics **8**, 719 (2012).

[17] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, *Open quantum assembly language,* arXiv preprint arXiv:1707.03429 (2017).

[18] N. Khammassi, G. Guerreschi, I. Ashraf, J. Hogaboam, C. Almudever, and K. Bertels, *cqasm v1. 0: Towards a common quantum assembly language,* arXiv preprint arXiv:1805.09607 (2018).

[19] D. Harris and S. Harris, *Digital design and computer architecture* (Morgan Kaufmann, 2010).

[20] AMBA and AXI, *Protocol specification,* ARM, June (2003).

[21] Intel, *Fifo intel fpga ip user guide,* https://www.intel.com/content/www/us/en/programmable/documentation/eis1414462767872.html (2018), accessed: 2018-12-01.

[22] Intel, *Double data rate i/o ip cores user guide,* https://www.intel.com/content/altera-www/global/en_us/index/documentation/eis1415168884929.html (2018), accessed: 2018-12-01.

[23] Intel, *Cyclone v datasheet,* https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_51002.pdf (2018), accessed: 2018-12-01.

[24] N. H. Weste and D. Harris, *CMOS VLSI design: a circuits and systems perspective* (Pearson Education, 2011).

[25] P. S. Foundation, *Python 3 pseudo-random number generator function documentation,* https://docs.python.org/2/library/random.html (2018), accessed: 2018-12-01.

[26] H. Homulle, S. Visser, B. Patra, G. Ferrari, E. Prati, F. Sebastiano, and E. Charbon, *A reconfigurable cryogenic platform for the classical control of quantum processors,* Review of Scientific Instruments **88**, 045103 (2017).

[27] B. Patra, R. M. Incandela, J. P. Van Dijk, H. A. Homulle, L. Song, M. Shahmohammadi, R. B. Staszewski, A. Vladimirescu, M. Babaie, F. Sebastiano, *et al.*, *Cryo-cmos circuits and systems for quantum computing applications,* IEEE Journal of Solid-State Circuits **53**, 309 (2018).

[28] H. Homulle and E. Charbon, *Performance characterization of altera and xilinx 28 nm fpgas at cryogenic temperatures,* in *Field Programmable Technology (ICFPT), 2017 International Conference on* (IEEE, 2017) pp. 25–31.

[29] H. Homulle and E. Charbon, *Cryogenic low-dropout voltage regulators for stable low-temperature electronics,* Cryogenics **95**, 11 (2018).

[30] S. Lin, Y.-B. Kim, and F. Lombardi, *Design and analysis of a 32 nm pvt tolerant cmos sram cell for low leakage and high stability,* Integration, the VLSI journal **43**, 176 (2010).

[31] S. Mukhopadhyay, H. Mahmoodi-Meimand, and K. Roy, *Modeling and estimation of failure probability due to parameter variations in nano-scale srams for yield enhancement,* in *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on* (IEEE, 2004) pp. 64–67.

[32] B. H. Calhoun and A. P. Chandrakasan, *Static noise margin variation for sub-threshold sram in 65-nm cmos,* IEEE Journal of solid-state circuits **41**, 1673 (2006).

[33] K. Das and T. Lehmann, *Sos current mirror matching at 4k: A brief study,* in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on* (IEEE, 2010) pp. 3405–3408.

[34] P. 't Hart, J. van Dijk, M. Babaie, E. Charbon, A. Vladimircscu, and F. Sebastiano, *Characterization and model validation of mismatch in nanometer cmos at cryogenic temperatures,* in *2018 48th European Solid-State Device Research Conference (ESSDERC)* (IEEE, 2018) pp. 246–249.

Acronyms

ASIC	Application-Specific Integrated Circuit. iii, 2, 9
AWG	Arbitrary Waveform Generator. 1, 3
AXI	Advanced Extensible Interface. 6, 7, 12, 19
CBox	Control Box. 3
CCLight	Central Controller-Light. iii, 2, 3, 5, 9, 29
DDR	Double Data Rate. 6
DRC	Design Rule Check. 19
eQASM	executable Quantum Assembly. 4–6, 9
FPGA	Field Programmable Gate Array. iii, 2, 3, 9
ISA	Instruction Set Architecture. 1, 3, 4, 9
LUT	Look-Up Table. 5
LVS	Layout Versus Schematic. 19
MMMC	Multi-Mode, Multi-Corner. 19
PVT	Process, Voltage, Temperature. 20
SOMQ	Single Operation Multiple Qubit. 4
SPI	Serial Peripheral Interface. 3, 19
UHFQA	Ultra High Frequency Quantum Analyzer. 1, 2, 13
VLIW	Very Long Instruction Word. 4, 6
VSM	Vector Switch Matrix. 1

Glossary

Firmware A common term for an hardware description language design, as this is written like ’software’ but describes ’hardware’, putting ’firmware’ somewhere in the middle. 2

GDS Stands for Graphic Database System, is the standard format that stores information about a layout design. 19

Netlist A (Verilog) module described using only generic gates or standard cells. 19

List of Code Blocks

A
Excerpts From the Thesis

A.1. eQASM Generator

```
#python script to create random benchmark QISA programs to functionally test the CCLight
#written by Lizzy Hatfield 5 Oct 18
#to implement:
#multis in bundled instructions
#multi two qubit masks


#imports
import random


numFiles = 5000
qubits = [0, 1, 2, 3, 4, 5, 6]
qubitPairRanges = [0, 0, 2, 6, 8, 12, 16]#index is number of qubits in experiment, value is max number of qubit pairs
validQubitPairs = [(0,2), (2,0), (0,3), (3,0), (1,3), (3,1), (1,4), (4,1), (2,5), (5,2), (3,5), (5,3), (3,6), (6,3), (4,6), (6,4)]
#cInstructions = [”ADD”, ”SUB”, ”AND”, ”OR”, ”XOR”, ”NOT”]
#otherClassical = [”NOP”, ”STOP”, ”CMP”, ”BR”, ”LDUI”, ”FBR”, ”FMR”]
qSInstructions = [”QNOP”, ”PrepX”, ”PrepZ”, ”MeasX”, ”MeasZ”, ”I”, ”X”, ”Y”, ”H”, ”S”, ”Sdag”, ”X90”, ”Xm90”, ”Y90”, ”Ym90”, ”T”, ”Tdag”, ”X45”, ”Xm45”]
validQSInstructions = [”CW_01”, ”CW_02”, ”CW_03”, ”MeasZ”, ”CW_08”, ”CW_09”, ”CW_0a”, ”CW_0b”, ”CW_0c”, ”CW_0d”, ”CW_0e”, ”CW_0f”,
          ”CW_10”, ”CW_11”, ”CW_12”, ”CW_13”, ”CW_14”, ”CW_15”, ”CW_16”, ”CW_17”, ”CW_18”, ”CW_19”, ”CW_1A”, ”CW_1B”, ”CW_1C”, ”CW_1D”, ”CW_1E”, ”CW_1F”,
          ”CW_20”, ”CW_21”, ”CW_22”, ”CW_23”, ”CW_24”, ”x90”, ”PrepZ”, ”Y”, ”X”, ”FLUX_01”, ”FLUX_02”, ”FLUX_03”, ”FLUX_04”, ”FLUX_05”, ”FLUX_06”, ”FLUX_07”,
          ”CW_04”, ”CW_05”, ”CW_06”, ”CW_07”]
qTInstructions = [”CNOT”, ”CZ”, ”SWAP”]
validQTInstructions = qTInstructions + [”CZ_3”, ”CZ_4”, ”CZ_5”, ”CZ_6”, ”CZ_7”, ”CZ_8”, ”CZ_9”, ”CZ_a”, ”CZ_b”, ”CZ_c”, ”CZ_d”, ”CZ_e”, ”CZ_f”, ”CZ_10”]
#quantumInstructions = validQSInstructions + validQTInstructions
metadata = ’./generated-qisa-files/benchmark-data.txt’
```

```

for index in range(numFiles):
  #some generation code
  #choose number of control qubits02
  numQubits = random.randint(0,6)#zero indexed
  #choose multi single qubits++++++
  numMultiS = random.randint(0,numQubits)
  if (numMultiS > 0):
    multiSes = []
    for x in range(numMultiS):
      tempList = qubits[0:numQubits]
      random.shuffle(tempList)
      #tempList = tempList[0:random.randint(0,numQubits)]
      multiSes.append(tempList[0:(len(tempList))])
  else:
    multiSes = []
  #choose number of qubit pairs
  if (numQubits >= 2):
    numPairs = random.randint(0,qubitPairRanges[numQubits])
    #create list of qubit pairs
    if (numPairs > 0): qubitPairs = validQubitPairs[0:numPairs]
    else: qubitPairs = []
    #shuffle pairs
    random.shuffle(qubitPairs)
  else:
    numPairs = 0
    qubitPairs = []
  #choose multi qubit pairs
  if (numPairs > 1):
    multiPs = []

  else:
    multiPs = []
  #choose number of regs
  numRegs = random.randint(0,31)#zero indexed
  #choose number of qwaits + gates
  OPmaxRange = 2048 - 3 - numQubits - numPairs - numRegs - numMultiS#?
  numOPs = random.randint(1,OPmaxRange)
  #keep line count
  numLines = 0
  #name the benchmark file
  filename = './generated-qisa-files/generated-benchmark-'+str(index+1)+'.
    qisa'

  with open(filename, 'w') as curr_qisa_file:
    #make mask for every single qubit
    for x in range(numQubits+1):#iterates through 0-n-1 hence +1 offset
      curr_qisa_file.write('SMIS s' + str(x) + ', { ' + str(x) + ' }\n')
      numLines += 1
    #make mask for every single qubit multi
    for x in range(numMultiS):#iterates through 0-n-1 hence +1 offset
      curr_qisa_file.write('SMIS s' + str(x+numQubits+1) + ', { ')
      for q in multiSes[x]:
        curr_qisa_file.write(str(q) + ', ')
      curr_qisa_file.seek(curr_qisa_file.tell()-2,0)#remove last two chars
    /bytes
    curr_qisa_file.write(’ }\n’)
    numLines += 1
  #make qubit pair masks
  for qp in qubitPairs:
    curr_qisa_file.write(’SMIT t’ + str(qubitPairs.index(qp)) + ’, { ’ + str(qp) + ’ }\n’)#list index out of range?
    numLines += 1
  #add LDIs
  for x in range(numRegs+1):
    curr_qisa_file.write(’LDI r’ + str(x) + ’, ’ + str(random.randint(0,2**(20-1)-1)) + ’\n’)
    numLines += 1
  #add ’Loop:’
  curr_qisa_file.write(’Loop: \n’)
  numLines += 1
  #add QINST(s) and QWAIT(s)
  for x in range(numOPs):
    curr_qisa_file.write(’\tqwaitr r’ + str(random.randint(0,numRegs)) + ’\n’)
    numLines += 1
    #choose two or one quantum instructions in this iteration
    if (random.randint(0,1) > 0 and numQubits > 0): #two instructions this iteration
      ###need to assure that the two chosen qubits arent the same
      if (random.randint(0, numQubits+numPairs) > numQubits): #first instruction will be two qubit op
        if (random.randint(0, numQubits+numPairs) > numQubits and numPairs > 4): #first and second will be two qubit ops
          #choose exclusive qubits for each bundled operation
          q1 = random.randint(0,numPairs-1)#pair index
          q2 = random.randint(0,numPairs-1)#pair index
          while qubitPairs[q1][0] == qubitPairs[q2][0] or qubitPairs[q1][1] == qubitPairs[q2][0] or qubitPairs[q1][0] == qubitPairs[q2][1] or qubitPairs[q1][1] == qubitPairs[q2][1]:
            q1 = random.randint(0,numPairs-1)
            q2 = random.randint(0,numPairs-1)
          curr_qisa_file.write(’\tbs ’ + str(random.randint(0,7)) + ’ ’ + random.choice(validQTInstructions) + ’ t’ + str(q1) + ’ | ’ + random.choice(validQTInstructions) + ’ t’ + str(q2) +’\n’)
        else: #first op will be two qubit op, second will be single qubit op
          #choose exclusive qubits for each bundled operation
          q1 = random.randint(0,numPairs-1)#pair index
          q2 = random.randint(0,numQubits)
          while qubitPairs[q1][0] == q2 or qubitPairs[q1][1] == q2:
            q2 = random.randint(0,numQubits)
          curr_qisa_file.write(’\tbs ’ + str(random.randint(0,7)) + ’ ’ + random.choice(validQTInstructions) + ’ t’ + str(q1) + ’ | ’ + random.choice(validQSInstructions) + ’ s’ + str(q2) +’\n’)
      else: #first op will be single qubit gate
        if (random.randint(0, numQubits+numPairs) > numQubits): #first op will be single qubit, second will be two qubit
          #choose exclusive qubits for each bundled operation
          q2 = random.randint(0,numPairs-1)#pair index
          q1 = random.randint(0,numQubits)
          while qubitPairs[q2][0] == q1 or qubitPairs[q2][1] == q1:
              q1 = random.randint(0,numQubits)
            curr_qisa_file.write(’\tbs ’ + str(random.randint(0,7)) + ’ ’ + random.choice(validQSInstructions) + ’ s’ + str(q1) + ’ | ’ + random.choice(validQTInstructions) + ’ t’ + str(q2) +’\n’)
          else: #both ops will be single qubit
            #choose two exclusive qubits
            q1 = random.randint(0,numQubits)
            q2 = random.randint(0,numQubits)
            while q1 == q2:
              q2 = random.randint(0,numQubits)
            curr_qisa_file.write(’\tbs ’ + str(random.randint(0,7)) + ’ ’ + random.choice(validQSInstructions) + ’ s’ + str(q1) + ’ | ’ + random.choice(validQSInstructions) + ’ s’ + str(q2) +’\n’)
      else: #one instruction this iteration
        #choose single or two qubit operation
        if (random.randint(0, numQubits+numPairs) > numQubits): #two qubit operation
          curr_qisa_file.write(’\tbs ’ + str(random.randint(0,7)) + ’ ’ + random.choice(validQTInstructions) + ’ t’ + str(random.randint(0, numPairs-1)) + ’\n’)
        else: #single qubit operation
          curr_qisa_file.write(’\tbs ’ + str(random.randint(0,7)) + ’ ’ + random.choice(validQSInstructions) + ’ s’ + str(random.randint(0, numQubits+numMultiS)) + ’\n’)
      numLines += 1
    #add branch
    curr_qisa_file.write(’\tBR always, loop\n’)
    numLines += 1
    #add NOPs
    curr_qisa_file.write(’\tNOP\n’)
    numLines += 1
    curr_qisa_file.write(’\tNOP\n’)
    numLines += 1
  print(’benchmark #’ + str(index+1) + ’ created’)
  with open(metadata, ’a+’) as metadata_file:
    #print(’numQubits = ’ + str(numQubits))
    #print(’numPairs = ’ + str(numPairs))
    #print(’qubitPairs = ’ + str(qubitPairs))
    #print(’numRegs = ’ + str(numRegs))
    #print(’numOPs = ’ + str(numOPs))
    #print(’numLines = ’ + str(numLines))
    #print(’filename = ’ + str(filename))
    #print(’numMultiS = ’ + str(numMultiS))
    #print(’multiSes = ’ + str(multiSes))
    metadata_file.write(str(filename) + ’;’ + str(numQubits) + ’;’ + str(numPairs)  + ’;’ + str(numMultiS) + ’;’ + str(numRegs) + ’;’ + str(numOPs) + ’;’ + str(numLines) + ’\n’)

#def pick_qubits(q1, q2):
  #pick qubits here
```

Code Block A.1: eQASM Generator

A.2. MjuOpAlign Original Firmware

```vhdl
-- ====================================================================================
-- VHDL Architecture QuantumPipeline.MJuOpAlign.ArchMJuOpAlign
-- ====================================================================================
-- Engineer          :  FU Xiang
-- Module            :  MJuOpAlign
-- File name         :  MJuOpAlign.vhd
-- Creation time     :  14:50:51 08/ 4/2017
-- Generated by      :  Mentor Graphics HDL Designer(TM) 2016.1 (Build 8)
-- ====================================================================================
-- Description:
--
-- ====================================================================================

-- ====================================================================================
-- Define synthesis directive for Quartus to indicate which library name to create for this component
-- synthesis library QuantumPipeline
-- ====================================================================================
--
-- Delft University of Technology  Project:  CC_Light
-- <enter comments here>
-- Title:  QuantumPipeline.MJuOpAlign
-- Path:  ./fpga/HDL Design/QuantumPipeline/MJuOpAlign/interface
-- Edited:  by FU Xiang on 04 Aug 2017
-- hds interface_start
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

LIBRARY QuantumPipeline;
USE QuantumPipeline.QuantumPipeline_pkg.all;

entity MJuOpAlign is
   port(
      AR_MajorTimestamp             : in     t_MajorTimestamp;
      AR_QuantumValid               : in     std_logic;
      vec_ConflictOps               : in     std_logic_vector (c_NumQubits - 1 downto 0);
      vec_MicroOperation_qubit      : in     t_vec_MicroOperationPerQubit;
      QMB_HorConf                   : out    std_logic_vector (c_NumQubits - 1 downto 0);
      QMB_VerConf                   : out    std_logic_vector (c_NumQubits - 1 downto 0);
      QMB_vec_MicroOperation_qubit  : out    t_vec_MicroOperationPerQubit;
      MJ_MajorTimestamp              : out    t_MajorTimestamp;
      Reset_Internal                 : in     std_logic;
      Clock_Internal                 : in     std_logic
   );

-- Declarations

end entity MJuOpAlign ;
-- hds interface_end

--
architecture ArchMJuOpAlign of MJuOpAlign is
   signal i_MajorTimestamp             : t_MajorTimestamp := (others => '0');

   signal i_vec_MicroOperation_qubit   : t_vec_MicroOperationPerQubit := (others => (others => '0'));
   signal i_Buffered                   : std_logic := '0';
   signal i_TimestampMatch             : std_logic := '0';

   signal i_vec_VerConf                : std_logic_vector(c_NumQubits - 1 downto 0) := (others => '0');

begin

   -- ==========================================================================================================
   -- Process: ProcAlignAndRelease
   -- Purpose: Align consecutive operations with the same timestamp but specified by different
   --          instructions into the same bundle.
   --          None zero output only when all operations of the same bundle have been collected.
   --          Default output: "0"s
   -- ==========================================================================================================
   ProcAlignAndRelease : process(Clock_Internal)
   begin
      if (rising_edge(Clock_Internal)) then

         QMB_vec_MicroOperation_qubit <= (others => (others => '0'));

         if (Reset_Internal = '1') then
            i_Buffered <= '0';
         elsif (AR_QuantumValid = '1') then
            i_Buffered <= '1';
         end if ;
         if (Reset_Internal = '1') then
            i_vec_MicroOperation_qubit <= (others => (others => '0'));
         else

            if (AR_QuantumValid = '1') then
               if (i_Buffered = '0') then
                  i_vec_MicroOperation_qubit <= vec_MicroOperation_qubit;

            else -- There are already operations buffered.

               LoopARAlignOps : for i in 0 to (c_NumQubits - 1) loop
                  -- old operations are released and new operations are
 put in place
                  if (i_TimestampMatch /= '1') then
                     i_MajorTimestamp <= AR_MajorTimestamp;
                     i_vec_MicroOperation_qubit(i) <=
vec_MicroOperation_qubit(i);

                     MJ_MajorTimestamp <= i_MajorTimestamp;
                     QMB_vec_MicroOperation_qubit <=
i_vec_MicroOperation_qubit;
                  else -- New operation is aligned with the old
operation
                     i_vec_MicroOperation_qubit(i) <=
i_vec_MicroOperation_qubit(i) or vec_MicroOperation_qubit(i);
                  end if ;
               end loop ; -- LoopARAlignOps

            end if ;
         end if ;
      end if;
   end if ;

end process ; -- ProcAlignAndRelease
i_TimestampMatch <= '1' when (AR_MajorTimestamp = i_MajorTimestamp)
else '0';

--
==========================================================================

-- Process: ProcVerConfCheck
-- Purpose: Combinatorial check if two instructions specifies
operations on the same qubit
--          with the same timestamp.
--
==========================================================================

ProcVerConfCheck : process(i_Buffered, AR_QuantumValid,
i_TimestampMatch, i_vec_MicroOperation_qubit, vec_MicroOperation_qubit)
begin
   LoopVerticalConflictCheck : for i in 0 to (c_NumQubits - 1) loop
      i_vec_VerConf(i) <= '0';

      if ((i_Buffered = '1') and (AR_QuantumValid = '1') and (
i_TimestampMatch = '1')) then
         if ( (i_vec_MicroOperation_qubit(i)(t_MicroOpTypeRange) /= (
t_MicroOpTypeRange => '0')) and
              (vec_MicroOperation_qubit(i)(t_MicroOpTypeRange) /= (
t_MicroOpTypeRange => '0')) ) then
            i_vec_VerConf(i) <= '1';
         end if;
      end if;

```

```vhdl
      end loop ; -- LoopVerticalConflictCheck
   end process ; -- ProcVerConfCheck

   --
    ===========================================================================

   -- Process: ProcVerConfCheck
   -- Purpose: Combinatorial check if two instructions specifies
    operations on the same qubit
   --          with the same timestamp.
   --
    ===========================================================================

   ProcConfAss : process(Clock_Internal)
   begin
      if (rising_edge(Clock_Internal)) then

         if (Reset_Internal = '1') then
            QMB_HorConf <= (others => '0');
            QMB_VerConf <= (others => '0');
         else
            QMB_HorConf <= vec_ConflictOps;
            QMB_VerConf <= i_vec_VerConf;
         end if ;

      end if ;
   end process ; -- ProcConfAss
end architecture ArchMJuOpAlign;
```

Code Block A.2: MjuOpAlign Original Firmware

A.3. MjuOpAlign Synthesis Compatible Firmware

```vhdl
--
    ===========================================================================

-- VHDL Architecture QuantumPipeline.MJuOpAlign.ArchMJuOpAlign
--
    ===========================================================================

-- Engineer           :  FU Xiang
-- Module             :  MJuOpAlign
-- File name          :  MJuOpAlign.vhd
-- Creation time      :  14:50:51 08/ 4/2017
-- Generated by       :  Mentor Graphics HDL Designer(TM) 2016.1 (Build 8)
--
    ===========================================================================

-- Description:
--
--
    ===========================================================================


--
    ===========================================================================
-- Define synthesis directive for Quartus to indicate which library name
    to create for this component
-- synthesis library QuantumPipeline
--
    ==================================================================================
--
-- Delft University of Technology  Project:  CC_Light
-- <enter comments here>
-- Title:  QuantumPipeline.MJuOpAlign
-- Path:  ./fpga/HDL Design/QuantumPipeline/MJuOpAlign/interface
-- Edited:  by FU Xiang on 04 Aug 2017
-- hds interface_start
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

LIBRARY QuantumPipeline;
USE QuantumPipeline.QuantumPipeline_pkg.all;

entity MJuOpAlign is
   port(
      AR_MajorTimestamp              : in     t_MajorTimestamp;
      AR_QuantumValid                : in     std_logic;
      vec_ConflictOps                : in     std_logic_vector (c_NumQubits
    - 1 downto 0);
      vec_MicroOperation_qubit       : in     t_vec_MicroOperationPerQubit;
      QMB_HorConf                    : out    std_logic_vector (c_NumQubits
    - 1 downto 0);
      QMB_VerConf                    : out    std_logic_vector (c_NumQubits
    - 1 downto 0);
      QMB_vec_MicroOperation_qubit : out    t_vec_MicroOperationPerQubit;
      MJ_MajorTimestamp              : out    t_MajorTimestamp;
      Reset_Internal                 : in     std_logic;
      Clock_Internal                 : in     std_logic
   );

-- Declarations

end entity MJuOpAlign ;
-- hds interface_end

--
architecture ArchMJuOpAlign of MJuOpAlign is
   signal i_MajorTimestamp              : t_MajorTimestamp;-- := (others =>
     '0');

   signal i_vec_MicroOperation_qubit    : t_vec_MicroOperationPerQubit;--
    := (others => (others => '0'));
   signal i_Buffered                    : std_logic;-- := '0';
   signal i_TimestampMatch              : std_logic;-- := '0';

   signal i_vec_VerConf                 : std_logic_vector(c_NumQubits - 1
    downto 0);-- := (others => '0');

```

```
begin

   --
   ================================================================================

   -- Process: ProcAlignAndRelease
   -- Purpose: Align consecutive operations with the same timestamp but
   specified by different
   --          instructions into the same bundle.
   --          None zero output only when all operations of the same
   bundle have been collected.
   --          Default output: "0"s
   --
   ================================================================================

   ProcAlignAndRelease : process(Clock_Internal)
   begin
      if (rising_edge(Clock_Internal)) then

         QMB_vec_MicroOperation_qubit <= (others => (others => '0'));

         if (Reset_Internal = '1') then
            i_Buffered <= '0';
      i_MajorTimestamp <= (others => '0');
            MJ_MajorTimestamp <= (others => '0');
         elsif (AR_QuantumValid = '1') then
            i_Buffered <= '1';
         end if ;
         if (Reset_Internal = '1') then
            i_vec_MicroOperation_qubit <= (others => (others => '0'));
         else

            if (AR_QuantumValid = '1') then
               if (i_Buffered = '0') then
                  i_vec_MicroOperation_qubit <= vec_MicroOperation_qubit;

               else -- There are already operations buffered.

                  LoopARAlignOps : for i in 0 to (c_NumQubits - 1) loop
                     -- old operations are released and new operations are
    put in place
                     if (i_TimestampMatch /= '1') then
                        i_MajorTimestamp <= AR_MajorTimestamp;
                        i_vec_MicroOperation_qubit(i) <=
   vec_MicroOperation_qubit(i);

                        MJ_MajorTimestamp <= i_MajorTimestamp;
                        QMB_vec_MicroOperation_qubit <=
   i_vec_MicroOperation_qubit;
                     else -- New operation is aligned with the old
   operation
                        i_vec_MicroOperation_qubit(i) <=
   i_vec_MicroOperation_qubit(i) or vec_MicroOperation_qubit(i);
                     end if ;
                  end loop ; -- LoopARAlignOps

            end if ;
         end if ;
      end if;
   end if ;

end process ; -- ProcAlignAndRelease
i_TimestampMatch <= '1' when (AR_MajorTimestamp = i_MajorTimestamp)
else '0';

--
=========================================================================================

-- Process: ProcVerConfCheck
-- Purpose: Combinatorial check if two instructions specifies
operations on the same qubit
--          with the same timestamp.
--
=========================================================================================

ProcVerConfCheck : process(i_Buffered, AR_QuantumValid,
i_TimestampMatch, i_vec_MicroOperation_qubit, vec_MicroOperation_qubit)
begin
   LoopVerticalConflictCheck : for i in 0 to (c_NumQubits - 1) loop
      i_vec_VerConf(i) <= '0';

      if ((i_Buffered = '1') and (AR_QuantumValid = '1') and (
i_TimestampMatch = '1')) then
         if ( (i_vec_MicroOperation_qubit(i)(t_MicroOpTypeRange) /= (
t_MicroOpTypeRange => '0')) and
              (vec_MicroOperation_qubit(i)(t_MicroOpTypeRange) /= (
t_MicroOpTypeRange => '0')) ) then
            i_vec_VerConf(i) <= '1';
         end if;
      end if;

   end loop ; -- LoopVerticalConflictCheck
end process ; -- ProcVerConfCheck

--
=========================================================================================

-- Process: ProcVerConfCheck
-- Purpose: Combinatorial check if two instructions specifies
operations on the same qubit
--          with the same timestamp.
--
=========================================================================================

ProcConfAss : process(Clock_Internal)
begin
   if (rising_edge(Clock_Internal)) then

      if (Reset_Internal = '1') then
         QMB_HorConf <= (others => '0');
         QMB_VerConf <= (others => '0');
      else
```

```
            QMB_HorConf <= vec_ConflictOps;
            QMB_VerConf <= i_vec_VerConf;
         end if ;

      end if ;
   end process ; -- ProcConfAss
end architecture ArchMJuOpAlign;
```

Code Block A.3: MjuOpAlign Synthesis Compatible Firmware

A.4. Genus CCLight Synthesis Script

```
###set script variables
set top QuMA_Wrapper
set runTime [clock format [clock seconds] -format %b_%d_%T]


###check script
#check_script ./QuMA_Wrapper.tcl


###setup
#set_db init_blackbox_for_undefined true
set_db information_level 4
set_db lib_search_path {/data/cad/DesignKits/TSMC/tsmcN40/tsmc_40lp_std/
    tcbn40lpbwp_v120c/TSMCHOME/digital/Front_End/timing_power_noise/ECSM/
    tcbn40lpbwp_120b/}
set_db library {tcbn40lpbwpwc_ecsm.lib /users/lhatfield/
    memory_voor_lizzy_11july/memories_2018_Jul_11_Ariston/Front_end/
    tsdn40lpa4096x36m4m_130a/NLDM/tsdn40lpa4096x36m4m_130a_tt1p1v125c.lib}
set_db design_process_node 40
set_db init_hdl_search_path {../HDL_Design/CCLight_QuMA/hdl/ ../HDL_Design
    /AXI_Interface_QuMA/hdl/ ../HDL_Design/ClassicalPipeline/hdl/ ../
    HDL_Design/QuantumPipeline/hdl/ ../HDL_Design/Common_IP/hdl/../
    HDL_Design/SignalReshape/hdl/../HDL_Design/AQUA_IP/hdl/}
set_db auto_ungroup none
set_db delete_unloaded_insts false
#set_db syn_generic_effort high
#set_db cmd_file ./cclight/genus.cmd
#set_db log_file ./cclight/genus.log

#::legacy::set_attribute dft_scan_map_mode force_all
#set_db delete_unloaded_seqs false
#set_db delete_hier_insts_on_preserved_net false
#set_db remove_assigns true
#dc::set_time_unit ps

set_db optimize_constant_1_flops false
set_db optimize_constant_0_flops false
set_db optimize_constant_latches false
set_db merge_combinational_hier_instances false
set_db optimize_merge_flops false
set_db optimize_merge_latches false
set_db optimize_constant_feedback_seqs false


```

```
###hdl packages
read_hdl -language vhdl ~/HDL_Design/AXI_Interface_QuMA/hdl/
    AXI_Interface_QuMA_pkg.vhd -library AXI_Interface_QuMA
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/CCLight_QuMA_pkg.vhd
     -library CCLight_QuMA
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/
    CCLight_SimUtils_PKG.vhd -library CCLight_QuMA
read_hdl -language vhdl ~/HDL_Design/ClassicalPipeline/hdl/
    CCLight_Classical_PKG.vhd -library ClassicalPipeline
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/
    CCLight_ClassicalPipeline_PKG.vhd -library CCLight_QuMA
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    QuantumPipeline_pkg.vhd -library QuantumPipeline
read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/AQUA_IP_pkg.vhd -library
    AQUA_IP


###common ip items (9)
#read_hdl -language vhdl ~/HDL_Design/ElecLib_Common_IP/Common_IP/hdl/
    Altera_AXI_Lightweight.vhd
#read_hdl -language vhdl ~/HDL_Design/ElecLib_Common_IP/Common_IP/hdl/
    AXI3_FULL_S00_AXI.vhd
read_hdl -language vhdl ~/HDL_Design/ElecLib_Common_IP/Common_IP/hdl/
    CDC_Sync_Bit.vhd
read_hdl -language vhdl ~/HDL_Design/ElecLib_Common_IP/Common_IP/hdl/
    CDC_Sync_Vector.vhd
read_hdl -language vhdl ~/HDL_Design/ElecLib_Common_IP/Common_IP/hdl/
    Mux2to1.vhd
read_hdl -language vhdl ~/HDL_Design/ElecLib_Common_IP/Common_IP/hdl/
    simple_dual_port_ram_dual_clock.vhd
read_hdl -language vhdl ~/HDL_Design/ElecLib_Common_IP/Common_IP/hdl/
    simple_dual_port_ram_single_clock.vhd
read_hdl -language vhdl ~/HDL_Design/ElecLib_Common_IP/Common_IP/hdl/
    VectorToBit_OR.vhd
read_hdl -language vhdl ~/HDL_Design/ElecLib_Common_IP/Common_IP/hdl/
    RegisterFile.vhd -library Common_IP


###altera crap (4-7)
read_hdl -language vhdl ~/HDL_Design/ClassicalPipeline/hdl/
    CCLight_Classical.vhd
read_hdl -language vhdl ~/HDL_Design/ClassicalPipeline/hdl/
    CCLight_ClassicalPipeline_pls.vhd
read_hdl -language vhdl ~/HDL_Design/ClassicalPipeline/hdl/
    CCLight_IcacheScratch-dpram.vhd
#read_hdl -language vhdl ~/HDL_Design/SignalReshape/hdl/VSM_DDR.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/MsmtValidFifo0.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/MsmtValidFifo1.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    LPM_FIFO_Flux_DIO_Event.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    LPM_FIFO_Msmt_DIO_Event.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    LPM_FIFO_MW_DIO_Event.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    LPM_FIFO_VSM_Event.vhd
```

```
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    LPM_FIFO_Timing.vhd


###who needs you, altera? (25)
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    MaskRegisterFile.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/MaskMux.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/MaskRFInPrep.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    SimpleRegisterFile.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/MicrocodeUnit.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    mcu_AddrDecoder.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/mcu_CS_Memory.vhd
read_hdl -language vhdl ~/HDL_Design/AXI_Interface_QuMA/hdl/
    UhfqcDioWrapper.vhd
read_hdl -language vhdl ~/HDL_Design/AXI_Interface_QuMA/hdl/
    DioToggleGen.vhd
#read_hdl -language vhdl ~/HDL_Design/AXI_Interface_QuMA/hdl/
    AXI_Interface_QuMA.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    FluxEventEmitter.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    MsmtEventEmitter.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    TimestampComparator.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    MicrowaveEventEmitter.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    TimeStampBroadCast.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    CounterControl.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/QPipeTop.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    mcu_ValidLabelDelay.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    FluxQueueManager.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    MsmtQueueManager.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    MWQueueManager.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    TimingQueueManager.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    VSMQueueManager.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    EventQueFrontMaster.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    VSMEventDirectEmitter.vhd
#read_hdl -language vhdl ~/HDL_Design/AXI_Interface_QuMA/hdl/
    UhfqcDioWrapper1.vhd


###AQUA IP (2)
#read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/AQUA_SYNCRAM.vhd
```

```
#we use an SRAM block for synthesis/implementation
#read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/AQUA_DDRO.vhd
read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/AQUA_FIFO_new.vhd
read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/AQUA_Broadcaster.vhd
read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/AQUA_Clock_Divider.vhd
#read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/AQUA_Mem_SPI.vhd
#read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/AQUA_Reg_SPI.vhd
read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/spi_slave.vhd
read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/AQUA_SPI_output_MEM.vhd
read_hdl -language vhdl ~/HDL_Design/AQUA_IP/hdl/AQUA_SPI_output_REG.vhd
read_hdl -language vhdl ~/HDL_Design/AXI_Interface_QuMA/hdl/
    QuMA_Wrapper_deparam_final_2.vhd


###to compile (44)
#read_hdl -language vhdl ~/HDL_Design/AXI_Interface_QuMA/hdl/
    ClockOutput.vhd
read_hdl -language vhdl ~/HDL_Design/SignalReshape/hdl/
    ConfigurableDelay.vhd
read_hdl -language vhdl ~/HDL_Design/SignalReshape/hdl/ProgDelay_5b.vhd
read_hdl -language vhdl ~/HDL_Design/ClassicalPipeline/hdl/
    CCLight_IcacheSlice.vhd
read_hdl -language vhdl ~/HDL_Design/ClassicalPipeline/hdl/
    CCLight_MeasRegFile.vhd
read_hdl -language vhdl ~/HDL_Design/ClassicalPipeline/hdl/
    CCLight_MeasRegFileSlice.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/BTriggerOutMap.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/Delay_n_CondExe.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/
    QubitDelayAndCondExe.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    ApplyCondition.vhd
read_hdl -language vhdl ~/HDL_Design/SignalReshape/hdl/HalfCycleDelay.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/
    ProgrammableDelay.vhd
read_hdl -language vhdl ~/HDL_Design/SignalReshape/hdl/MultipleDelay.vhd
read_hdl -language vhdl ~/HDL_Design/SignalReshape/hdl/VsmAddPoints.vhd
#read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/FlipPulseGen.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/
    MsmtResDIOAnalysis.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/ExeCondGen.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/ValidMsmtMaskGen.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/AddrResolver.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/OperationMux.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/MaskManager.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/MM_ValidDelay.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    DeviceEventDistributor.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    FluxTriggerGen.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    QubitEventValidGen.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    Flux_DIO_Mapping.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    MsmtTriggerGen.vhd
```

```
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    MicrowaveTriggerGen.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/DeviceEventOR.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    MW_DIO_Mapping.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    VSMTriggerMapping.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/MeasIssueGen.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    MicroOperationJoin.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    MicroOperationOR.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/MJuOpAlign.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    QuantumDecoder.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/DelayRegister.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    TimestampManager.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    TimingControlUnit.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/QuMAAXIComMan.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/QuMALwAXIComMan.vhd
read_hdl -language vhdl ~/HDL_Design/CCLight_QuMA/hdl/QuMATop.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/counter.vhd
read_hdl -language vhdl ~/HDL_Design/QuantumPipeline/hdl/
    Msmt_DIO_Mapping.vhd


###elaboration
elaborate


###hdl options
set_dont_touch QuMA_Wrapper/inst_QuMATop/inst_CCLight_Classical/IcacheInst
    /NormalMemGen.Memory
#set_dont_touch QuMA_Wrapper


###set constraints
set_top_module $top
create_clock -name clock_200MHz -period 5 [get_ports clock_200MHz]
create_clock -name clock_100MHz -period 10 QuMA_Wrapper/Gen100MHz/
    out_clock
create_clock -name clock_50MHz -period 20 QuMA_Wrapper/Gen50MHz/out_clock
create_clock -name clock_50MHz_pi_shift -period 20 QuMA_Wrapper/Gen50MHz/
    n_out_clock
create_clock -name PLL_50MHz_Locked -period 20 QuMA_Wrapper/Gen50MHzPLL/
    n_out_clock
create_clock -name pll_locked -period 10 [get_ports pll_locked]
create_clock -name sclk_REG -period 10 [get_ports sclk_REG]
create_clock -name sclk_MEM -period 10 [get_ports sclk_MEM]
set_clock_uncertainty -setup 0.2 [all_clocks]
set_clock_uncertainty -hold 0.1 [all_clocks]
#set_dont_touch_network [all_clocks]
#set_ideal_network clock_200MHz

```

```

###retime
#retime -prepare
#retime -min_delay -clock clock_200MHz

###write pre-synthesis reports
#report sequential $top -hier >> cclight/${top}_${runTime}_reports.txt
report datapath $top -all >> cclight/${top}_${runTime}_reports.txt
report clocks [all_clocks] >> cclight/${top}_${runTime}_reports.txt
report hierarchy >> cclight/${top}_${runTime}_reports.txt
#report hierarchy
#report port $top *


###check_constraints
check_design $top -all
#validate_constraints -netlist /designs/$top


###synthesis
syn_generic
syn_map
#syn_opt
echo "synthesis performed" >> cclight/${top}_${runTime}_reports.txt


###synthesis reports
report_runtime >> cclight/${top}_${runTime}_reports.txt
report qor $top >> cclight/${top}_${runTime}_reports.txt
report summary >> cclight/${top}_${runTime}_reports.txt
report design_rules >> cclight/${top}_${runTime}_reports.txt
report timing -lint >> cclight/${top}_${runTime}_reports.txt
report area >> cclight/${top}_${runTime}_reports.txt
report hierarchy >> cclight/${top}_${runTime}_reports.txt
report gates >> cclight/${top}_${runTime}_reports.txt
report power >> cclight/${top}_${runTime}_reports.txt
report datapath >> cclight/${top}_${runTime}_reports.txt


###write outputs
write_reports -directory cclight/ -tag ${top}_${runTime}
write_hdl $top > cclight/${top}_${runTime}_mapped.v
write_sdf -design $top > cclight/${top}_${runTime}_mapped.sdf
write_sdc $top > cclight/${top}_${runTime}_mapped.sdc
#write_script > cclight/${top}_constraints.g
#write_design -innovus -basename cclight/${top}_${runTime}


gui_show
```

Code Block A.4: Genus CCLight Synthesis Script

B

Miscellaneous Toolchain Scripts

B.1. Netlist to Schematic Script

To solve the issue with Innovus® exporting some digital pad cells without all of the needed ports, a script was created to fix the exported netlist. The script simply runs a regex command for each delinquent pad cell, finding it in the netlist (possibly multiple times) and cutting its port declarations from the netlist, then adding them back in (see the `\1`), then appending the missing ports to the end of the declarations. This script is written in sh/csh, and the code is given below:

```
#!/bin/sh

#made by lizzy hatfield 8-6-18
#call with "source netlist-import-prep.sh <name-of-source-netlist-file.v>"

#fixes PDDW04DGZ_G
sed -i -r 's/PDDW04DGZ_G (.*) \(/PDDW04DGZ_G \1 (\n.POC(POC), \n.VDD(VDD), \n.VDDPST(VDDPST), \n.VSS(VSS), \n.VSSPST(VSSPST), /' $1
#fixes PDDW04SDGZ_G
sed -i -r 's/PDDW04SDGZ_G (.*) \(/PDDW04SDGZ_G \1 (\n.POC(POC), \n.VDD(VDD), \n.VDDPST(VDDPST), \n.VSS(VSS), \n.VSSPST(VSSPST), /' $1
#fixes PVDD2POC_G
sed -i -r 's/PVDD2POC_G (.*) \(/PVDD2POC_G \1 (\n.POC(POC), \n.VDD(VDD), \n.VDDPST(VDDPST), \n.VSS(VSS), \n.VSSPST(VSSPST) /' $1
#fixes PVDD2DGZ_G
sed -i -r 's/PVDD2DGZ_G (.*) \(/PVDD2DGZ_G \1 (\n.VDDPST(VDDPST), \n.VSS(VSS), \n.VSSPST(VSSPST) /' $1
#fixes PVSS2DGZ_G
sed -i -r 's/PVSS2DGZ_G (.*) \(/PVSS2DGZ_G \1 (\n.VDD(VDD), \n.VDDPST(VDDPST), \n.VSS(VSS), \n.VSSPST(VSSPST) /' $1
#fixes PVDD1DGZ_G
sed -i -r 's/PVDD1DGZ_G (.*) \(/PVDD1DGZ_G \1 (\n.VSS(VSS), \n.VSSPST(
    VSSPST), /' $1
#fixes PVSS1DGZ_G
sed -i -r 's/PVSS1DGZ_G (.*) \(/PVSS1DGZ_G \1 (\n.VDD(VDD), \n.VDDPST(
    VDDPST), \n.VSSPST(VSSPST), /' $1
```

Code Block B.1: Netlist to Schematic Script

B.2. SPICE Netlist Script

In order to solve the bug in the TSMC 40 nm PDK where the pad cells are not correctly parsed from schematic to SPICE netlist (for LVS), a script was developed to fix the SPICE netlist before running LVS. This script works by traversing the SPICE netlist and searching for the incorrect definition of each pad cell one at a time, and replacing it entirely with the correct one (a regex operation). This script was made initially to cover all digital pad cells from the TSMC 40 nm IO PDK, but was later expanded to cover all of the pad cells, including the analog ones. The code is written for Python 2.6.6, the version installed on the CoolGroup server. Note that the details on the parameters of the transistors in the pad cells, while the most important part of the actual script, are not given here as they are proprietary to TSMC. They are replaced here with Xes. The code is given below:

```
#python

#made by Lizzy Hatfield and Mark Cilissen Aug-8-18
#runs on python 2.6.6; python 3 and higher needs a different re.compile(),
     etc
#call with "python spice-netlist-fixer.py <name-of-source-netlist-file.src
    .net>"
#works by replacing pad cell definitions in SPICE netlist exported from
    schematic view with the correct ones
#should help you pass LVS
#known model issues within the adapted TSMC pad cells:
# transistors nch_* and pch_* give either swapped length and width values
    or finger width as the actual width after SPICE export from Virtuoso
# resistors (rppolywo, rm*w) give default values or possibly the values
    from sumW/L after SPICE export from virtuoso
#how to add a new pad or other cell description to this script:
# 1) export the SPICE netlist from the Virtuoso scheatic view of the cell
    using the Export > CDL tool.
# 2) copy and paste everything from .SUBCKT to .ENDS, as is done below
# 3) put that definition in triple quotes ("""<def>""") and give it a name
     eg replace_PAD123 = """<def>"""
# 4) then find and replace the incorrect values with the right ones
# 5) run LVS to confirm the property errors are gone

##fixes PDDW04DGZ_G
##fixes PDDW04SDGZ_G
##fixes PVDD2POC_G
##fixes PVDD2DGZ_G
##fixes PVDD1DGZ_G
##fixes PVSS3A_G
##fixes PVDD3A_G
##fixes PVSS2A_G
##fixes PVDD3AC_G
##fixes PVSS2AC_G?
##fixes PVSS3AC_G?
##fixes PVDD1ANA_G
```

```


import re
import sys


##********************************************************************
##* Library Name: io_7m4x1z1u_lvs_atef
##* Cell Name:    PDDW04DGZ_G
##* View Name:    schematic
##********************************************************************

replace_PDDW04DGZ_G = """.SUBCKT PDDW04DGZ_G C I OEN PAD POC REN VDD
    VDDPST VSS VSSPST
*.PININFO C:B I:B OEN:B PAD:B POC:B REN:B VDD:B VDDPST:B VSS:B VSSPST:B
MM_64 net_23 net_19 net_24 VSS nch_25od33 l=XXX w=XX m=X
MM_67 net_24 net_19 net_25 VSS nch_25od33 l=XXX w=XX m=X
MM_65 net_25 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_2 net_5 net_1 net_6 VSS nch_25od33 l=XXX w=XX m=X
MM_3 net_4 net_1 net_7 VSS nch_25od33 l=XXX w=XX m=X
MM_12 net_11 net_1 net_14 VSS nch_25od33 l=XXX w=XX m=X
MM_11 net_12 net_1 net_13 VSS nch_25od33 l=XXX w=XX m=X
MM_98 net_38 net_37 net_36 VSS nch_25od33 l=XXX w=XX m=X
MM_105 net_19 net_9 net_38 VSS nch_25od33 l=XXX w=XX m=X
MM_76 net_29 net_2 net_30 VSS nch_25od33 l=XXX w=XX m=X
MM_77 net_30 net_10 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_66 VSS VSS VSS VSS nch_25od33 l=XXX w=XX m=X
MM_20 PAD net_16 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_21 PAD net_16 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_22 PAD net_16 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_23 PAD net_16 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_24 PAD net_16 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_25 PAD net_16 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_26 PAD net_16 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_27 PAD net_16 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_60 net_21 net_22 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_57 net_19 net_20 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_61 net_20 REN VSS VSS nch_25od33 l=XXX w=XX m=X
MM_59 net_20 POC VSS VSS nch_25od33 l=XXX w=XX m=X
MM_99 net_1 POC VSS VSS nch_25od33 l=XXX w=XX m=X
MM_101 net_34 net_23 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_102 VDD net_34 net_35 VSS nch_25od33 l=XXX w=XX m=X
MM_100 net_35 net_23 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_104 net_17 net_37 net_33 VSS nch_25od33 l=XXX w=XX m=X
MM_103 net_36 net_10 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_13 net_10 POC VSS VSS nch_25od33 l=XXX w=XX m=X
MM_4 net_3 POC VSS VSS nch_25od33 l=XXX w=XX m=X
MM_72 net_27 net_9 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_73 net_27 net_2 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_80 net_31 net_27 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_82 net_32 net_29 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_84 net_33 net_32 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_97 net_33 net_32 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_86 net_15 net_31 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_18 PAD net_15 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_19 PAD net_15 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
```

```
MM_69 net_26 net_19 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_70 net_23 net_19 net_26 VDDPST pch_25od33 l=XXX w=XX m=X
MM_74 net_28 net_9 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_75 net_27 net_2 net_28 VDDPST pch_25od33 l=XXX w=XX m=X
MM_28 PAD net_17 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_29 PAD net_17 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_30 PAD net_17 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_31 PAD net_17 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_32 PAD net_17 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_33 PAD net_17 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_34 PAD net_17 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_35 PAD net_17 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_36 PAD net_17 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_37 PAD net_17 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_38 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_39 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_40 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_41 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_42 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_43 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_44 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_45 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_46 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_47 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_48 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_49 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_50 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_51 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_52 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_53 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_54 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_55 PAD net_18 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_68 VDDPST VDDPST VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_95 net_18 net_37 net_19 net_18 pch_25od33 l=XXX w=XX m=X
MM_96 net_17 net_37 net_19 net_18 pch_25od33 l=XXX w=XX m=X
MM_58 net_18 net_21 net_18 net_18 pch_25od33 l=XXX w=XX m=X
MM_62 net_21 net_20 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_63 net_20 net_21 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_71 net_23 net_1 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_89 net_1 POC VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_90 net_34 net_23 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_91 net_35 net_23 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_92 net_36 net_37 net_19 net_18 pch_25od33 l=XXX w=XX m=X
MM_94 net_18 net_36 VDDPST net_18 pch_25od33 l=XXX w=XX m=X
MM_93 net_17 net_36 net_33 net_18 pch_25od33 l=XXX w=XX m=X
MM_14 net_10 net_9 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_15 net_9 net_10 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_5 net_3 net_2 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_6 net_2 net_3 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_78 net_29 net_2 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_79 net_29 net_10 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_81 net_31 net_27 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_83 net_32 net_29 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_85 net_33 net_32 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_87 net_15 net_31 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_88 net_15 net_31 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
```

```
MM_8 net_6 I VSS VSS nch l=XXX w=XX m=X
MM_7 net_7 net_0 VSS VSS nch l=XXX w=XX m=X
MM_16 net_14 net_8 VSS VSS nch l=XXX w=XX m=X
MM_17 net_13 OEN VSS VSS nch l=XXX w=XX m=X
MM_113 C net_39 VSS VSS nch l=XXX w=XX m=X
MM_111 net_22 REN VSS VSS nch l=XXX w=XX m=X
MM_112 net_39 net_35 VSS VSS nch l=XXX w=XX m=X
MM_107 net_0 I VSS VSS nch l=XXX w=XX m=X
MM_109 net_8 OEN VSS VSS nch l=XXX w=XX m=X
MM_114 C net_39 VDD VDD pch l=XXX w=XX m=X
MM_110 net_22 REN VDD VDD pch l=XXX w=XX m=X
MM_115 net_39 net_35 VDD VDD pch l=XXX w=XX m=X
MM_106 net_0 I VDD VDD pch l=XXX w=XX m=X
MM_108 net_8 OEN VDD VDD pch l=XXX w=XX m=X
XXR_116 net_16 VSSPST rnpolywo l=XXX w=XX m=X
XXR_117 VSS net_40 rppolywo l=XXX w=XX m=X
XXR_118 net_41 VDD rppolywo l=XXX w=XX m=X
XX_119 net_19 PAD rppolywo l=XXX w=XX m=X
XX_120 VDDPST net_37 rppolywo l=XXX w=XX m=X
DD_56 VSSPST PAD ndio_esd area=XXX pj=XX m=X
MM_1 net_3 I net_5 VSS nch_na25od33 l=XXX w=XX m=X
MM_0 net_2 net_0 net_4 VSS nch_na25od33 l=XXX w=XX m=X
MM_9 net_9 net_8 net_11 VSS nch_na25od33 l=XXX w=XX m=X
MM_10 net_10 OEN net_12 VSS nch_na25od33 l=XXX w=XX m=X
.ENDS"""



##*************************************************************************
##* Library Name: io_7m4x1z1u_lvs_atef
##* Cell Name:    PDDW04SDGZ_G
##* View Name:    schematic
##*************************************************************************

replace_PDDW04SDGZ_G = """.SUBCKT PDDW04SDGZ_G C I OEN PAD POC REN VDD
    VDDPST VSS VSSPST
*.PININFO C:B I:B OEN:B PAD:B POC:B REN:B VDD:B VDDPST:B VSS:B VSSPST:B
MM_14 C net_10 VDD VDD pch l=XXX w=XX m=X
MM_21 net_14 OEN VDD VDD pch l=XXX w=XX m=X
MM_13 net_10 net_11 VDD VDD pch l=XXX w=XX m=X
MM_17 net_12 I VDD VDD pch l=XXX w=XX m=X
MM_19 net_13 REN VDD VDD pch l=XXX w=XX m=X
MM_110 net_36 net_14 VSS VSS nch l=XXX w=XX m=X
MM_111 net_35 OEN VSS VSS nch l=XXX w=XX m=X
MM_119 net_41 net_12 VSS VSS nch l=XXX w=XX m=X
MM_120 net_40 I VSS VSS nch l=XXX w=XX m=X
MM_16 C net_10 VSS VSS nch l=XXX w=XX m=X
MM_22 net_14 OEN VSS VSS nch l=XXX w=XX m=X
MM_15 net_10 net_11 VSS VSS nch l=XXX w=XX m=X
MM_18 net_12 I VSS VSS nch l=XXX w=XX m=X
MM_20 net_13 REN VSS VSS nch l=XXX w=XX m=X
MM_106 net_33 net_1 net_36 VSS nch_25od33 l=XXX w=XX m=X
MM_105 net_34 net_1 net_35 VSS nch_25od33 l=XXX w=XX m=X
MM_115 net_38 net_1 net_41 VSS nch_25od33 l=XXX w=XX m=X
MM_114 net_39 net_1 net_40 VSS nch_25od33 l=XXX w=XX m=X
MM_31 net_22 net_7 net_19 VSS nch_25od33 l=XXX w=XX m=X
```

```
MM_30 net_0 net_21 net_22 VSS nch_25od33 l=XXX w=XX m=X
MM_50 net_29 net_20 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_49 net_26 net_28 net_29 VSS nch_25od33 l=XXX w=XX m=X
MM_3 net_3 net_0 net_4 VSS nch_25od33 l=XXX w=XX m=X
MM_1 net_4 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_66 PAD net_9 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_67 PAD net_9 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_68 PAD net_9 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_69 PAD net_9 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_70 PAD net_9 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_71 PAD net_9 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_72 PAD net_9 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_73 PAD net_9 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_26 net_19 net_20 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_28 net_1 POC VSS VSS nch_25od33 l=XXX w=XX m=X
MM_107 net_20 POC VSS VSS nch_25od33 l=XXX w=XX m=X
MM_24 net_17 net_7 net_15 VSS nch_25od33 l=XXX w=XX m=X
MM_29 net_11 net_2 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_25 VDD net_18 net_11 VSS nch_25od33 l=XXX w=XX m=X
MM_27 net_18 net_2 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_23 net_15 net_16 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_45 net_15 net_16 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_41 net_16 net_26 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_53 net_27 net_21 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_54 net_27 net_28 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_43 net_24 net_27 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_47 net_23 net_24 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_64 PAD net_23 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_65 PAD net_23 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_58 net_31 net_13 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_57 net_32 POC VSS VSS nch_25od33 l=XXX w=XX m=X
MM_116 net_37 POC VSS VSS nch_25od33 l=XXX w=XX m=X
MM_2 VDDPST net_2 net_3 VSS nch_25od33 l=XXX w=XX m=X
MM_0 net_2 net_0 net_3 VSS nch_25od33 l=XXX w=XX m=X
MM_62 net_0 net_32 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_59 net_32 REN VSS VSS nch_25od33 l=XXX w=XX m=X
MM_55 net_30 net_21 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_56 net_27 net_28 net_30 VDDPST pch_25od33 l=XXX w=XX m=X
MM_84 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_85 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_86 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_87 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_88 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_89 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_90 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_91 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_92 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_93 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_94 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_95 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_96 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_97 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_98 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_99 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_100 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_101 PAD net_25 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
```

```
MM_108 net_20 net_21 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_109 net_21 net_20 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_37 net_25 net_7 net_0 net_25 pch_25od33 l=XXX w=XX m=X
MM_5 net_5 net_0 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_4 VSS net_2 net_5 VDDPST pch_25od33 l=XXX w=XX m=X
MM_6 net_2 net_0 net_5 VDDPST pch_25od33 l=XXX w=XX m=X
MM_7 net_2 net_1 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_33 net_25 net_19 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_35 net_19 net_7 net_0 net_25 pch_25od33 l=XXX w=XX m=X
MM_36 net_1 POC VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_38 net_17 net_7 net_0 net_25 pch_25od33 l=XXX w=XX m=X
MM_74 PAD net_17 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_75 PAD net_17 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_76 PAD net_17 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_77 PAD net_17 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_78 PAD net_17 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_79 PAD net_17 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_80 PAD net_17 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_81 PAD net_17 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_82 PAD net_17 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_83 PAD net_17 VDDPST net_25 pch_25od33 l=XXX w=XX m=X
MM_34 net_17 net_19 net_15 net_25 pch_25od33 l=XXX w=XX m=X
MM_63 net_25 net_31 net_25 net_25 pch_25od33 l=XXX w=XX m=X
MM_39 net_11 net_2 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_40 net_18 net_2 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_46 net_15 net_16 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_42 net_16 net_26 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_52 net_26 net_20 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_51 net_26 net_28 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_44 net_24 net_27 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_32 net_23 net_24 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_48 net_23 net_24 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_60 net_31 net_32 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_61 net_32 net_31 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_117 net_37 net_28 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_118 net_28 net_37 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
XXR_8 VSS net_6 rppolywo l=XXX w=XX m=X
XXR_11 net_8 VDD rppolywo l=XXX w=XX m=X
XX_9 net_0 PAD rppolywo l=XXX w=XX m=X
XX_10 VDDPST net_7 rppolywo l=XXX w=XX m=X
XXR_12 net_9 VSSPST rnpolywo l=XXX w=XX m=X
DD_102 VSSPST PAD ndio_esd area=XXX pj=XX m=X
MM_103 net_21 net_14 net_33 VSS nch_na25od33 l=XXX w=XX m=X
MM_104 net_20 OEN net_34 VSS nch_na25od33 l=XXX w=XX m=X
MM_112 net_28 net_12 net_38 VSS nch_na25od33 l=XXX w=XX m=X
MM_113 net_37 I net_39 VSS nch_na25od33 l=XXX w=XX m=X
.ENDS"""



##***********************************************************************
##* Library Name: io_7m4x1z1u_lvs_atef
##* Cell Name:    PVDD2POC_G
##* View Name:    schematic
##***********************************************************************

```

```
replace_PVDD2POC_G = """.SUBCKT PVDD2POC_G POC VDD VDDPST VSS VSSPST
*.PININFO POC:B VDD:B VDDPST:B VSS:B VSSPST:B
MM_22 net_0 net_1 net_3 VDDPST pch_25od33 l=XXX w=XX m=X
MM_1 net_3 net_1 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_2 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_3 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_4 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_5 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_6 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_7 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_8 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_9 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_10 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_11 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_12 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_13 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_21 net_9 net_6 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_20 net_10 net_9 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_17 POC net_10 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_19 POC net_10 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_16 net_8 net_9 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_14 VSS net_6 net_7 VDDPST pch_25od33 l=XXX w=XX m=X
MM_24 net_6 net_0 net_7 VDDPST pch_25od33 l=XXX w=XX m=X
MM_23 net_7 net_0 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_15 POC net_8 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_18 POC net_8 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_57 net_6 net_0 net_11 VSS nch_25od33 l=XXX w=XX m=X
MM_25 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_26 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_27 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_28 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_29 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_30 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_31 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_32 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_33 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_34 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_35 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_36 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_37 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_38 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_39 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_40 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_41 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_42 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_43 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_44 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_45 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_46 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_47 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_48 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_49 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_50 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_51 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_52 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_53 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
```

```
MM_54 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_55 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_56 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_58 net_11 net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_59 VDDPST net_6 net_11 VSS nch_25od33 l=XXX w=XX m=X
MM_64 net_9 net_6 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_61 net_10 net_9 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_63 POC net_10 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_65 POC net_10 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_60 POC net_8 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_62 POC net_8 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_66 net_8 net_9 VSS VSS nch_25od33 l=XXX w=XX m=X
XXR_69 net_1 VDD rnpolywo l=XXX w=XX m=X
DD_67 VSSPST VDDPST ndio_esd area=XXX pj=XX m=X
XXR_70 VDDPST net_5 rppolywo l=XXX w=XX m=X
MM_0 net_0 net_1 net_2 VSS nch_na25od33 l=XXX w=XX m=X
MM_68 net_2 net_1 VSS VSS nch l=XXX w=XX m=X
.ENDS"""



##***********************************************************************
##* Library Name: io_7m4x1z1u_lvs_atef
##* Cell Name:    PVDD2DGZ_G
##* View Name:    schematic
##***********************************************************************

replace_PVDD2DGZ_G = """.SUBCKT PVDD2DGZ_G VDDPST VSS VSSPST
*.PININFO VDDPST:B VSS:B VSSPST:B
MM_22 net_0 net_1 net_3 VDDPST pch_25od33 l=XXX w=XX m=X
MM_1 net_3 net_1 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_2 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_3 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_4 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_5 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_6 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_7 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_8 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_9 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_10 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_11 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_12 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_13 net_4 net_5 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_21 net_10 net_6 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_20 net_11 net_10 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_17 net_8 net_11 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_19 net_8 net_11 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_16 net_9 net_10 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_14 VSS net_6 net_7 VDDPST pch_25od33 l=XXX w=XX m=X
MM_24 net_6 net_0 net_7 VDDPST pch_25od33 l=XXX w=XX m=X
MM_23 net_7 net_0 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_15 net_8 net_9 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_18 net_8 net_9 VDDPST VDDPST pch_25od33 l=XXX w=XX m=X
MM_57 net_6 net_0 net_12 VSS nch_25od33 l=XXX w=XX m=X
MM_25 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_26 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
```

```
MM_27 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_28 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_29 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_30 VSSPST net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_31 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_32 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_33 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_34 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_35 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_36 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_37 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_38 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_39 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_40 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_41 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_42 net_4 net_5 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_43 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_44 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_45 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_46 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_47 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_48 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_49 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_50 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_51 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_52 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_53 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_54 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_55 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_56 VDDPST net_4 VSSPST VSSPST nch_25od33 l=XXX w=XX m=X
MM_58 net_12 net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_59 VDDPST net_6 net_12 VSS nch_25od33 l=XXX w=XX m=X
MM_64 net_10 net_6 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_61 net_11 net_10 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_63 net_8 net_11 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_65 net_8 net_11 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_60 net_8 net_9 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_62 net_8 net_9 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_66 net_9 net_10 VSS VSS nch_25od33 l=XXX w=XX m=X
XXR_69 net_1 VSS rnpolywo l=XXX w=XX m=X
DD_67 VSSPST VDDPST ndio_esd area=XXX pj=XX m=X
XXR_70 VDDPST net_5 rppolywo l=XXX w=XX m=X
MM_0 net_0 net_1 net_2 VSS nch_na25od33 l=XXX w=XX m=X
MM_68 net_2 net_1 VSS VSS nch l=XXX w=XX m=X
.ENDS"""



##*************************************************************************
##* Library Name: io_7m4x1z1u_lvs_atef
##* Cell Name:    PVDD1DGZ_G
##* View Name:    schematic
##*************************************************************************

replace_PVDD1DGZ_G = """.SUBCKT PVDD1DGZ_G VDD VSS VSSPST
*.PININFO VDD:B VSS:B VSSPST:B
```

```
MM_1 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_2 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_3 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_4 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_5 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_6 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_7 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_8 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_9 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_10 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_11 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_12 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_13 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_14 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_15 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_16 net_0 net_1 VDD VDD pch_25od33 l=XXX w=XX m=X
MM_17 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_18 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_19 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_20 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_21 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_22 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_23 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_24 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_25 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_26 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_27 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_28 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_29 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_30 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_31 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_32 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_33 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_34 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_35 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_36 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_37 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_38 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_39 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_40 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_41 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_42 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_43 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_44 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_45 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_46 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_47 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_48 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_49 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_50 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_51 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_52 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_53 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_54 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_55 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_56 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
```

```
DD_0 VSSPST VDD ndio_25od33 area=XXX pj=XX m=X
MM_57 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_58 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_59 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_60 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_61 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_62 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_63 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_64 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_65 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_66 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_67 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_68 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_69 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_70 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_71 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_72 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_73 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_74 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_75 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_76 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_77 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_78 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_79 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_80 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_81 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_82 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_83 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_84 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_85 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_86 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_87 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_88 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_89 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_90 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_91 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_92 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_93 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_94 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_95 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_96 VDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
XXR_97 VDD net_1 rppolywo l=XXX w=XX m=X
.ENDS"""


###analog pads begin here

#************************************************************************
#* Library Name: io_7m4x1z1u_lvs_atef
#* Cell Name:    PVSS2A_G
#* View Name:    schematic
#************************************************************************

replace_PVSS2A_G=""".SUBCKT PVSS2A_G TAVDD TAVSS VSS
*.PININFO TAVDD:B TAVSS:B VSS:B
MM_0 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
```

```
MM_1 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_2 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_3 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_4 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_5 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_6 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_7 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_8 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_9 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_10 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_11 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_12 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_13 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_14 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_15 VSS TAVDD VSS VSS nch_25od33 l=XXX w=XX m=X
DD_18 VSS TAVSS pdio area=XXX pj=XX m=X
DD_17 VSS TAVDD pdio area=XXX pj=XX m=X
DD_16 TAVSS VSS ndio area=XXX pj=XX m=X
.ENDS"""

#************************************************************************
#* Library Name: io_7m4x1z1u_lvs_atef
#* Cell Name:     PVSS3A_G
#* View Name:     schematic
#************************************************************************

replace_PVSS3A_G = """.SUBCKT PVSS3A_G AVSS TAVDD TAVSS VSS
*.PININFO AVSS:B TAVDD:B TAVSS:B VSS:B
MM_0 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_1 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_2 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_3 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_4 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_5 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_6 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_7 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_8 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_9 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_10 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_11 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_12 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_13 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_14 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_15 TAVSS TAVDD TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
DD_20 TAVSS VSS pdio area=XXX pj=XX m=X
DD_19 TAVSS TAVDD pdio area=XXX pj=XX m=X
XXR_16 TAVSS AVSS rm2w l=XXX w=XX m=X
XXR_17 TAVSS AVSS rm1w l=XXX w=XX m=X
DD_18 VSS TAVSS ndio area=XXX pj=XX m=X
.ENDS"""

#************************************************************************
#* Library Name: io_7m4x1z1u_lvs_atef
#* Cell Name:     PVDD3A_G
#* View Name:     schematic
#************************************************************************
```

```

replace_PVDD3A_G=""".SUBCKT PVDD3A_G AVDD TAVDD TAVSS VSS
*.PININFO AVDD:B TAVDD:B TAVSS:B VSS:B
MM_0 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_1 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_2 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_3 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_4 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_5 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_6 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_7 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_8 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_9 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_10 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_11 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_12 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_13 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_14 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_15 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_16 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_17 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_18 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_19 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_20 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_21 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_22 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_23 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_24 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_25 TAVDD net_0 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_26 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_27 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_28 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_29 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_30 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_31 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_32 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_33 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_34 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_35 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_36 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_37 net_2 net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_38 TAVSS net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_39 TAVSS net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_40 TAVSS net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_41 TAVSS net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_42 TAVSS net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_43 TAVSS net_1 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_44 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_45 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_46 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_47 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_48 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_49 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_50 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_51 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_52 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
```

```
MM_53 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_54 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_55 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_56 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_57 TAVSS net_2 TAVSS TAVSS nch_25od33 l=XXX w=XX m=X
MM_58 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_59 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_60 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_61 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_62 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_63 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_64 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_65 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_66 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_67 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_68 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_69 net_2 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_70 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_71 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_72 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_73 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_74 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_75 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_76 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_77 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_78 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_79 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_80 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
MM_81 net_0 net_1 TAVDD TAVDD pch_25od33 l=XXX w=XX m=X
DD_82 VSS TAVDD ndio_esd area=XXX pj=XX m=X
XX_85 TAVDD net_1 rppolywo l=XXX w=XX m=X
XXR_83 TAVDD AVDD rm1w l=XXX w=XX m=X
XXR_84 TAVDD AVDD rm2w l=XXX w=XX m=X
.ENDS"""

#**************************************************************************
#* Library Name: io_7m4x1z1u_lvs_atef
#* Cell Name:    PVDD3AC_G
#* View Name:    schematic
#**************************************************************************

replace_PVDD3AC_G = """.SUBCKT PVDD3AC_G AVDD TACVDD TACVSS VSS
*.PININFO AVDD:B TACVDD:B TACVSS:B VSS:B
MM_0 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_1 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_2 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_3 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_4 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_5 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_6 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_7 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_8 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_9 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_10 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_11 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_12 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
```

```
MM_13 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_14 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_15 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_16 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_17 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_18 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_19 net_0 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_20 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_21 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_22 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_23 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_24 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_25 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_26 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_27 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_28 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_29 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_30 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_31 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_32 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_33 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_34 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_35 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_36 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_37 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_38 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_39 net_2 net_1 TACVDD TACVDD pch_25od33 l=XXX w=XX m=X
MM_40 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_41 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_42 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_43 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_44 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_45 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_46 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_47 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_48 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_49 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_50 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_51 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_52 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_53 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_54 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_55 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_56 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_57 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_58 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_59 net_2 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_60 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_61 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_62 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_63 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_64 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_65 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_66 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_67 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_68 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
```

```
MM_69 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_70 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_71 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_72 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_73 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_74 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_75 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_76 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_77 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_78 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_79 net_0 net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_80 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_81 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_82 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_83 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_84 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_85 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_86 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_87 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_88 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_89 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_90 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_91 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_92 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_93 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_94 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_95 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_96 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_97 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_98 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_99 TACVSS net_1 TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_102 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_103 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_104 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_105 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_106 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_107 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_108 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_109 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_110 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_111 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_112 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_113 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_114 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_115 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_116 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_117 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_118 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_119 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_120 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_121 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_122 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_123 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_124 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_125 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_126 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
```

```
MM_127 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_128 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_129 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_130 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_131 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_132 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_133 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_134 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_135 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_136 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_137 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_138 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_139 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_140 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_141 TACVDD net_2 VSS VSS nch_hvt l=XXX w=XX m=X
MM_142 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_143 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_144 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_145 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_146 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_147 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_148 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_149 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_150 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_151 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_152 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_153 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_154 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_155 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_156 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_157 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_158 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_159 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_160 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_161 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_162 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_163 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_164 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_165 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_166 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_167 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_168 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_169 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_170 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_171 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_172 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_173 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_174 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_175 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_176 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_177 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_178 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_179 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_180 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
MM_181 TACVSS net_0 TACVSS TACVSS nch_hvt l=XXX w=XX m=X
XX_182 TACVDD net_1 rppolywo l=XXX w=XX m=X
```

```
XXR_100 TACVDD AVDD rm2w l=XXX w=XX m=X
XXR_101 TACVDD AVDD rm1w l=XXX w=XX m=X
.ENDS"""

#***********************************************************************
#* Library Name: io_7m4x1z1u_lvs_atef
#* Cell Name:    PVSS2AC_G
#* View Name:    schematic
#***********************************************************************

replace_PVSS2AC_G = """.SUBCKT PVSS2AC_G TACVDD TACVSS VSS
*.PININFO TACVDD:B TACVSS:B VSS:B
MM_0 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_1 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_2 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_3 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_4 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_5 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_6 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_7 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_8 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_9 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_10 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_11 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_12 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_13 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_14 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
MM_15 VSS TACVDD VSS VSS nch_25od33 l=XXX w=XX m=X
DD_18 VSS TACVSS pdio area=XXX pj=XX m=X
DD_17 VSS TACVDD pdio area=XXX pj=XX m=X
DD_16 TACVSS VSS ndio area=XXX pj=XX m=X
.ENDS"""

#***********************************************************************
#* Library Name: io_7m4x1z1u_lvs_atef
#* Cell Name:    PVSS3AC_G
#* View Name:    schematic
#***********************************************************************

replace_PVSS3AC_G = """.SUBCKT PVSS3AC_G AVSS TACVDD TACVSS VSS
*.PININFO AVSS:B TACVDD:B TACVSS:B VSS:B
MM_0 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_1 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_2 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_3 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_4 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_5 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_6 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_7 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_8 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_9 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_10 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_11 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_12 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_13 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
MM_14 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
```

```
MM_15 TACVSS TACVDD TACVSS TACVSS nch_25od33 l=XXX w=XX m=X
DD_20 TACVSS VSS pdio area=XXX pj=XX m=X
DD_19 TACVSS TACVDD pdio area=XXX pj=XX m=X
XXR_16 TACVSS AVSS rm2w l=XXX w=XX m=X
XXR_17 TACVSS AVSS rm1w l=XXX w=XX m=X
DD_18 VSS TACVSS ndio area=XXX pj=XX m=X
.ENDS"""

#************************************************************************
#* Library Name: io_7m4x1z1u_lvs_atef
#* Cell Name:    PVDD1ANA_G
#* View Name:    schematic
#************************************************************************

replace_PVDD1ANA_G = """.SUBCKT PVDD1ANA_G AVDD VSS VSSPST
*.PININFO AVDD:B VSS:B VSSPST:B
MM_1 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_2 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_3 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_4 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_5 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_6 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_7 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_8 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_9 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_10 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_11 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_12 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_13 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_14 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_15 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_16 net_0 net_1 AVDD AVDD pch_25od33 l=XXX w=XX m=X
MM_17 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_18 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_19 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_20 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_21 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_22 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_23 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_24 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_25 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_26 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_27 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_28 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_29 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_30 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_31 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_32 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_33 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_34 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_35 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_36 VSS net_1 VSS VSSPST nch_25od33 l=XXX w=XX m=X
MM_37 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_38 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_39 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_40 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
```

```
MM_41 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_42 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_43 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_44 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_45 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_46 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_47 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_48 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_49 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_50 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_51 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_52 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_53 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_54 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_55 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
MM_56 net_0 net_1 VSS VSS nch_25od33 l=XXX w=XX m=X
DD_0 VSSPST AVDD ndio_25od33 area=XXX pj=XX m=X
MM_57 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_58 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_59 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_60 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_61 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_62 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_63 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_64 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_65 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_66 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_67 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_68 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_69 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_70 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_71 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_72 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_73 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_74 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_75 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_76 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_77 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_78 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_79 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_80 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_81 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_82 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_83 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_84 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_85 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_86 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_87 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_88 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_89 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_90 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_91 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_92 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_93 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_94 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
MM_95 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
```

```
MM_96 AVDD net_0 VSS VSS nch_hvt l=XXX w=XX m=X
XXR_97 AVDD net_1 rppolywo l=XXX w=XX m=X
.ENDS"""



#create regex
sections = {
    'PDDW04DGZ_G': replace_PDDW04DGZ_G,
    'PDDW04SDGZ_G': replace_PDDW04SDGZ_G,
    'PVDD2POC_G': replace_PVDD2POC_G,
    'PVDD2DGZ_G': replace_PVDD2DGZ_G,
    'PVDD1DGZ_G': replace_PVDD1DGZ_G,
    'PVSS2A_G': replace_PVSS2A_G,
    'PVSS3A_G': replace_PVSS3A_G,
    'PVDD3A_G': replace_PVDD3A_G,
    'PVDD3AC_G': replace_PVDD3AC_G,
    'PVSS2AC_G': replace_PVSS2AC_G,
    'PVSS3AC_G': replace_PVSS3AC_G,
    'PVDD1ANA_G': replace_PVDD1ANA_G
}

#read original file (create buffer)
filename = sys.argv[1]
with open(filename, 'r') as f:
    contents = f.read()

#find and replace in original file's buffer
for name, replacement in sections.items():
    pattern = re.compile('.SUBCKT ' + name + '.*?.ENDS', re.DOTALL)
    contents = pattern.sub(replacement, contents)

#write buffer to filename.new
with open(filename + '.new', 'w') as f:
    f.write(contents)
```

Code Block B.2: SPICE Netlist Script

C
Newsletter Software

To increase the communication within the CoolGroup, the weekly newsletter was started. The idea is that every student in the lab reports on what they've been working on in the past week, any problems they've come across, and what they plan to do in the next week. In this way everyone knows what everyone else is doing, and the progress of the various projects can be tracked. However, the process of collecting and formulating this data into a newsletter email each week was a lengthy process, and often resulted in an email with inconsistent formatting, as each person's submission was copied from their email responses. Of course, this problem was solved with automation. A JavaScript (Google Script API) software was created to collect this data through Google Forms and Sheets, and then place the data into an HTML formatted string which can then be emailed to everyone.

The newsletter software can be very briefly described in the following manner. The members of the lab receive weekly email reminders sent from the software at the user's discretion. The email contents are dictated by a Google Doc which is parsed to HTML by the JS back-end. The lab submits their weekly reports and optionally comics to Google Forms which store this data in Google Sheets. When the user wants to send the email, they press a button on the 'GUI'. This triggers the back-end to build an HTML template which is pre-defined, then parse the lab reports Google Sheets and place the items in that template. In this process it also checks the items for the HTML tags that would 'break' the template. It also adds a comic from the respective Google Sheet. Finally once the full template has been assembled with the desired data, it send the email to the lab. The user also has the option to send themselves a test email first, which makes the same email sent to the lab otherwise, but without cleaning the data from the Google Sheets. Other options include turning on and off name sorting, announcements, and comics within the newsletters.

Examples of the GUI with success/error messages on sending the newsletter are given below in Figure C.1 and Figure C.2.

The front-end HTML 'GUI' code is given below:

```
<!--

-if you dont have a div inside a div the whole table will disappear after
   a click

-delete handlers after 1 use?

-->
<!DOCTYPE html>
<html>
<head>
   <base target="_top">
<style>
table {
    width: 600px;
```

Figure C.1: Newsletter GUI, Email Success Handler

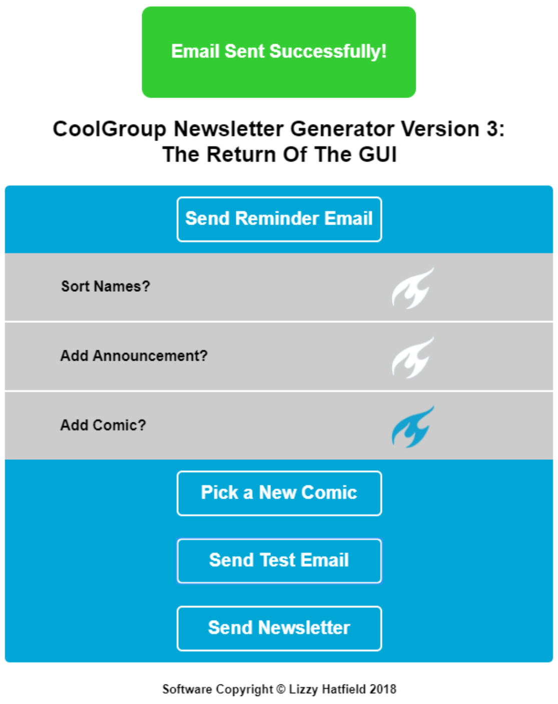
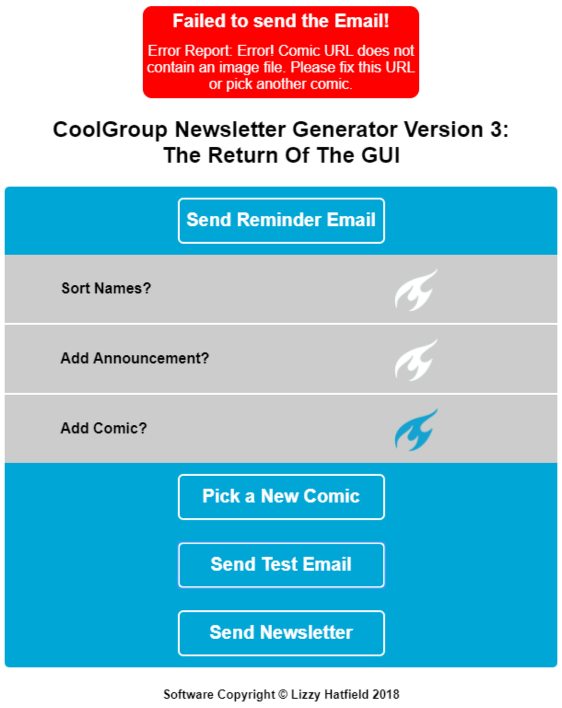


Figure C.2: Newsletter GUI, Email Failure Handler



Error Report: Error! Comic URL does not contain an image file. Please fix this URL or pick another comic.

```
  /*height: 400px;*/
    margin: 0;
    border-collapse: collapse;
    font-family: arial;
    table-layout: fixed;
}
```

```

/*reports table definition*/
#CGReports th, #CGReports td {
    height: 49px;
    padding: 12.5px;
    /*margin: 0;*/
    vertical-align: middle;
    text-align: center;
}
#CGReports th {
  background-color: #00A6D6;
  color: white;
  font-size: 125%;
}
/*describe the custom input elements*/
input[type="button"] {
    background-color: #00A6D6;
    color: #fff;
    width: 225px; /* width of image */
    height: 50px; /* height of image */
    border: 2px solid #fff;
    font-family: arial;
    font-size: 100%;
    font-weight: bold;
    border-radius: 6px 6px 6px 6px;
}

input[type=checkbox].css-checkbox {/*margin or sthg needs to be fixed*/
  position:absolute;
    z-index:-1000;
    left:-1000px;
    overflow: hidden;
    clip: rect(0 0 0 0);
    height:1px; width:1px;
    margin:-1px;
    padding:0;
    border:0;
}

input[type=checkbox].css-checkbox + label.css-label, input[type=checkbox].
    css-checkbox + label.css-label.clr {
  padding-left:55px;
  height:50.4px;
  display:inline-block;
  line-height:50px;
  background-repeat:no-repeat;
  background-position: 0 0;
  font-size:50px;
  vertical-align:middle;
  cursor:pointer;

}

input[type=checkbox].css-checkbox:checked + label.css-label, input[type=
    checkbox].css-checkbox + label.css-label.chk {
  background-position: 0 -50px;
}
label.css-label {
  background-image:url(http://csscheckbox.com/checkboxes/u/
    csscheckbox_feef178dfbea01689537ae950e54fb64.png);
  -webkit-touch-callout: none;
  -webkit-user-select: none;
  -khtml-user-select: none;
  -moz-user-select: none;
  -ms-user-select: none;
  user-select: none;
}
</style>
</head>

<body>

<div style="overflow-x:auto;" align="center">
<div id="CGNewsletter" style="width: 300px; height: 100px" align="center">
    </div>
<h2 style="width: 600px; font-family:arial;text-align: center;">CoolGroup
    Newsletter Generator Version 3:<br>The Return Of The GUI</h2>

<table id="CGReports">
  <tr>
    <th style="border-radius: 6px 6px 0 0;" colspan="2"> <input type="
    button" id="Reminder" value="Send Reminder Email"></th>
  </tr>
  <tr style="background-color: #ccc;" >
    <td style="text-align: left;"><b>&emsp;&emsp;&emsp;Sort Names?</b></td
    >
    <td><input id="Sort" name="Sort" value="true" type="checkbox" class="
    css-checkbox"><label for="Sort" class="css-label radGroup1"></label></
    td>
  </tr>
  <tr style="background-color: #ccc; border-top: 2px solid #fff; border-
    bottom: 2px solid #fff;" >
    <td style="text-align: left;" max-height="49px"><b>&emsp;&emsp;&emsp;
    Add Announcement?</b></td>
    <td><input id="Announcement" name ="Announcement" value="true" type="
    checkbox" class="css-checkbox"><label for="Announcement" class="css-
    label radGroup1"></label></td>
  </tr>
  <tr style="background-color: #ccc;">
    <td style="text-align: left; max-height: 49;"><b>&emsp;&emsp;&emsp;Add
     Comic?</b></td>
    <td><input id="Comic" name="Comic" value="true" type="checkbox" class=
    "css-checkbox" checked><label for="Comic" class="css-label radGroup1"><
    /label></td>
  </tr>
  <tr>
    <th colspan="2"><input type="button" id="PickComic" value="Pick a New
    Comic"></th>
  </tr>
  <tr>
    <th colspan="2"><input type="button" id="Test" value="Send Test Email"
    ></th>
  </tr>
  <tr>
    <th style="border-radius: 0 0 6px 6px;" colspan="2"><input type="
   button" id="Newsletter" value="Send Newsletter"></th>
  </tr>
</table>

<script>
    /*---make event listeners--*/
    document.getElementById('Reminder').addEventListener('click',
   SendReminder);//execute on change in reminder
    document.getElementById('PickComic').addEventListener('click',
   ShuffleComics);
    document.getElementById('Test').addEventListener('click', SendTest);//
   execute on change in test
    document.getElementById('Newsletter').addEventListener('click',
   SendNewsletter);//execute on change in newsletter

    /*---define success handler--*/
    function success()
    {
      var message = [
        '<table style="font-family:arial; border-collapse: collapse; width
   : 300px; height: 100px; margin-left: 0;">',
        '<tr>',
        '<td style="background-color: #33cc33; color: white; border-radius
   : 10px 10px 10px 10px; font-size: 125%" align="center">',
        '<b>Email Sent Successfully!</b>',
        '</td>',
        '</tr>',
        '</table>'
        ].join('\n')
      document.getElementById('CGNewsletter').innerHTML = message;
    }

    /*---define failure handler--*/
    function failure(error)
    {
      var message = [
        '<table style="font-family:arial; border-collapse: collapse; width
   : 300px; height: 100px; margin-left: 0;">',
        '<tr>',
        '<td style="background-color: #ff0000; color: white; border-radius
   : 10px 10px 0 0; font-size: 125%" align="center">',
        '<b>Failed to send the Email!</b>',
        '</td>',
        '</tr>',
        '<tr>',
        '<td style="background-color: #ff0000; color: white; border-radius
   : 0 0 10px 10px;" align="center">',
        'Error Report: ' + error.message,
        '</td>',
        '</tr>',
        '</table>'
        ].join('\n')
      document.getElementById('CGNewsletter').innerHTML = message;
      google.script.run.errorEmail(error.message);
    }

    /*---functions that call the backend code---*/
    function ShuffleComics()
    {
      google.script.run.shuffleSheet();
    }
    function SendReminder()
    {
      document.getElementById(’CGNewsletter’).innerHTML = ”Sending Reminder Email...”;
      google.script.run.withFailureHandler(failure).withSuccessHandler(success).sendReminder();
    }
    function SendTest()
    {
      //var progress_text=”Sending Test Email...”;
      var sort=document.getElementById(’Sort’).checked;
      var ann=document.getElementById(’Announcement’).checked;
      var comic=document.getElementById(’Comic’).checked;
      document.getElementById(’CGNewsletter’).innerHTML = ”Sending Test Email...”;
      google.script.run.withFailureHandler(failure).withSuccessHandler(success).testEmail(sort, ann, comic);
    }
    function SendNewsletter()
    {
      var sort=document.getElementById(’Sort’).checked;
      var ann=document.getElementById(’Announcement’).checked;
      var comic=document.getElementById(’Comic’).checked;
      document.getElementById(’CGNewsletter’).innerHTML = ”Sending Newsletter Email...”;
      google.script.run.withFailureHandler(failure).withSuccessHandler(success).sendNewsletter(sort, ann, comic);
    }
  </script>

<h5 style=”width: 600px; font-family:arial;text-align: center;”>Software Copyright © Lizzy Hatfield 2018</h5>
</div>
</body>
</html>
```

Code Block C.1: Newsletter Software HTML Source Code

The back-end JavaScript is given below:

```
/* *********************************************************************************

 * Engineer: Lizzy Hatfield (4625129)
 * Name: CoolGroup Newsletter Generator
 * Description: Script that generates and sends weekly newsletter emails via Google Forms, Sheets, and Docs
 * Ver: 3.0
 * © Lizzy Hatfield
```

```
****************************************************************************************************
*/

/* ------------------------------------------------------------------------------------ Documentation -------------------------------------------------------------------------------------

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                           .".
                          /  |
                         /  /
                        / ,"
            .-------.--- /
           "._ __.-/ o. o\
              "   (    Y  )
                   )     /
                  /     (
                 /       Y
             .-"         |
            /  _     \    \
           /    `. ". ) /' )
          Y       )( / /(,/
         ,|      /     )
        ( |     /     /
         " \_  (___  (___       [nabis]
             "-._,)--._,)

© Chris Johnson,
http://www.chris.com/ascii/index.php?art=animals/rabbits
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


New in Version 1.1:
++reports can be sorted alphabetically by name
++depth of the ranges can be programmically defined
++added copyright
++announcement/comic are now optional
++HTML saved in a separate folder
++clearData takes range as an input

New in Version 1.2:
++function testEmail() to test the output email
++function testAnnouncement() to test announcement for correct formatting added from Test-Funcs
++function testThisHTML() to test new HTML formatting file for email compatability added from Test-Funcs
++function sendReminder() to send a reminder email

New in Version 2.0:
++added GUI for running the script to control all the bells and whistles more cleanly
++main renamed to sendNewsletter

New in Version 3.0:
++bug in clearing old submission data from form spreadsheet fixed
```

```
++email sent to developer on error in GUI
++announcements and reminder read from Docs text to HTML directly
++comics can be added by anyone and are chosen at random by the script
++malice checking of progress submissions (</td>)
++sendNewsletter and testEmail called functions bundled to keep consistency between the two
++code is now fully portable to any group/lab/university/etc


for the future when you want to read announcements.doc as html:
   https://stackoverflow.com/questions/47299478/google-apps-script-docs-convert-selected-element-to-html/47313357#47313357

-a big THANK YOU to Jarva for translating the alpha version of this algorithm from Python to JavaScript/Google Script (and writing the getWeek() function!)
------------------------------------------------------------------------------------ */

//front end code


function doGet() {
  var html = HtmlService.createTemplateFromFile('Webpage').evaluate();
  html.setTitle("coolGroup Newsletter");
  return html;
}

function include(filename) {
  return HtmlService.createHtmlOutputFromFile(filename)
      .setSandboxMode(HtmlService.SandboxMode.IFRAME)
      .getContent();
}


//backend code


/*==================================================================Global Variables ================================================================================

All of the global variables are defined here. These are the pieces of information that are the same for every function, although most functions are written to use
these values as if they were local variables declared in the main functions.

==================================================================================== */
var form_sheet_id = '1nFi6f98GuG_46Xmg3V8WPoY0rtMK4LykOIW5QilFRiY';
var archive_sheet_id = '1PkE5F7MWh-nS67w-PKAxgaYXsdleA_mdD_QpZUI8aAQ';
var announcement_doc_id = '12qAcj4a-SQ-1Nwd9QyQeCYToXA-objeBrvE58iUg8j8';
var work_folder = '16bmSg6V75RwXc6dWlQpWBCxgBWoJ92Fp';
var HTML_folder = '1ewwc7cRepBUsKh-DGxD9rIwir5LC1WzM';
var comic_sheet_id = '1xt5fVg-G1on9X6igIvaILBIdB_wKg7GFd9Bp0aAURMc';
var reminder_doc_id = '1R4l1rokHQLdFt2LmVRIlty-Tew9W6OOPptG3vgVuQcg';
var archiveEmails = true;

var devEmail = 'yoyoultimate@gmail.com';
var userEmail = 'fabio.sebastiano@gmail.com';
var labEmail = 'coolgroup-ewi@tudelft.nl';

var labName = 'CoolGroup';
var headerBGColor = '#00A6D6'; //cyan
var headerTextColor = 'white';
var accentColor1 = '#eee'; //light gray
var accentColor2 = '#fff'; //white



/*==================================================================Main Functions ==========================================================================

These are the top-level functions that execute all the tasks needed respectively. They are directly called by the GUI

-rewrite with functions bundled to keep consistency between testEmail and sendNewsletter
============================================================================================ */
function sendNewsletter(doSort, AddAnn, AddComic) {

var subject = "Newsletter Week " + getWeek();
var body = createEmail(AddAnn, AddComic, doSort);
var recipient = labEmail;

sendAndSaveEmail(subject, recipient, body, archiveEmails);
cleanup();
}//end


/* ----------------------------------------------------------------------------- testEmail() ----------------------------------------------------------------------------


Description: Sends a test email to a single recipient of the generated newsletter

Arguments:
-none
Returns:
-none
Notes:
-allows for the creation and sending of an email in the same way as main(), but without performing the cleanup functions main() does such as saving items to the Archive
```

```
-should be rewritten in the future to run the same code as main() so that
    the functions stay coherent (and leave main() to run some cleanup()
    procedures)
---------------------------------------------------------------------------------
    */
function testEmail(doSort, AddAnn, AddComic) {

var subject = "Test Newsletter Week " + getWeek();
var body = createEmail(AddAnn, AddComic, doSort);
var recipient = userEmail;

sendAndSaveEmail(subject, recipient, body, false);
}//end


/*
    ------------------------------------------------------------------------
    sendReminder()
    -------------------------------------------------------------------------------


Description: Sends a reminder email to the CoolGroup to fill in the Google
     Form

Arguments:
-none
Returns:
-none
Notes:
---------------------------------------------------------------------------------
    */
function sendReminder() {
//create the HTML file and send the email
var email = retrieveAnnouncement(reminder_doc_id);
//send email
GmailApp.sendEmail(labEmail, "Newsletter Submission Reminder", "Error
    Sending Email", {htmlBody: email});
}



/*======================================================================
    Utility Functions
    ===========================================================================


These functions are here to test functionalities without sending the
    newsletter to the entire coolgroup. They are not called anywhere, but
    rather are meant to be called
by the user from the Google Script Development environment.

=================================================================================
    */


/*-------------------------------------------------------------------------
```

```
    testAnnouncement()
    --------------------------------------------------------------------------------


Description: Sends an email with only the Announcement box from the
    Newsletter to test the HTML formatting of the announcement itself

Arguments:
-none
Returns:
-none
Notes:
-needs to be rewritten so it calls the same functions used
-should no longer be self-contained
--------------------------------------------------------------------------------------------
    */
function testAnnouncement(){
 //open announcement doc
 var docID = '12b9LNsjeh6ac3qlhRHTjvDQeNBgbsBVlhkbHlIk5NCU';
 var announcement = [];
 announcement = convertToHTML(docID);

 //create email
 var email = [
    '<!DOCTYPE html>',
    '<html>',
    '<head>',
    '<style>',
    'table {width: 800px;margin: 20px;}',
    'table, th, td {border-collapse: collapse;}',
    'th, td {padding: 12px; text-align: left;}',
    'table#t01 tr:nth-child(even) {background-color: ' + accentColor1 + '
    ;}',
    'table#t01 tr:nth-child(odd) {background-color: ' + accentColor2 + ';}
    ',
    'table#t01 th {background-color: #00A6D6; color: white;}',
    '</style>',
    '</head>',
    '<body>',
    '<table style="font-family:arial;" id="t01">',
    '<tr>',
    '<th style="text-align:center">Announcements</th>',
    '</tr>',
    '<tr style = "background-color: ' + accentColor1 + ';">',
    '<td>' + announcement + '</td>',
    '</tr>',
    '</table>',
    '</body>',
    '</html>'
  ].join('\n')
 //send email
 GmailApp.sendEmail(userEmail, "Announcement Test", "error", {htmlBody:
    email});
}


```

```
/* ------------------------------------------------------------------------------ testThisHTML() ----------------------------------------------------------------------------------

Description: Reads from a test format HTML file and sends it as an email to the recipient to test if the format is email-compatible

Arguments:
-none
Returns:
-none
Notes:
-could be linearized (one line)
-------------------------------------------------------------------------------------------- */
/* use this guy to test a new html style file for email compatability */
function testThisHTML(){
 var files = DriveApp.getFilesByName('new-style.html');
 var file = files.next();
 var html = file.getAs('text/html');
 var email = html.getDataAsString()
 GmailApp.sendEmail(userEmail, "HTML Test", "error", {htmlBody: email});
}


/*=========================================================================== Bundle Functions ================================================================================

These functions execute the functions to perform general groups of tasks. This aliasing is done to keep testEmail and sendAnnouncement congruent

==================================================================================================== */


function createEmail(AddAnn, AddComic, doSort){
 //get data to build email with
 var ann = retrieveAnnouncement(announcement_doc_id);
 var com = retrieveComic(comic_sheet_id);
 //check for image file in URL
 if ((String(com[0]).search(".png") == -1) && (String(com[0]).search(".jpg") == -1) && (String(com[0]).search(".jpeg") == -1) && (String(com[0]).search(".gif") == -1))//search for image file extension
   throw "Error! Comic URL does not contain an image file. Please fix this URL or pick another comic.";
 if (doSort)
 {
  alphabetizeSheet(form_sheet_id, full_data_range);
 }
 var data = SpreadsheetApp.openById(form_sheet_id).getSheets()[0].getDataRange().offset(1, 1).getValues();//open sheet, grab table as far as valid data extends, remove top row and leftmost column

 //create an array with the tables of the weelky reports
 var groupTables = [];
 for (i in data) {
  var row = data[i];
  for (j in row)//malice check every entry in this submission
  {
    if (String(row[j]).search(”</td>”) != -1)
    {
      var num = Number(j) + 2;//unexpected strings are unexpected
      var col = (num + 9).toString(16).toUpperCase();
      var error_message = ”Error! Someone is trying to break the newsletter HTML! Check the Newsletter Submission spreadsheet, row ” + (Number(i) + 2) + ”, column ” + col + ”, for the tag \”&#60;/td&#62;\”.”;
      throw error_message;
    }
    if (String(row[j]).search(”</tr>”) != -1)
    {
      var num = Number(j) + 2;//unexpected strings are unexpected
      var col = (num + 9).toString(16).toUpperCase();
      var error_message = ”Error! Someone is trying to break the newsletter HTML! Check the Newsletter Submission spreadsheet, row ” + (Number(i) + 2) + ”, column ” + col + ”, for the tag \”&#60;/tr&#62;\”.”;
      throw error_message;
    }
    if (String(row[j]).search(”</table>”) != -1)
    {
      var num = Number(j) + 2;//unexpected strings are unexpected
      var col = (num + 9).toString(16).toUpperCase();
      var error_message = ”Error! Someone is trying to break the newsletter HTML! Check the Newsletter Submission spreadsheet, row ” + (Number(i) + 2) + ”, column ” + col + ”, for the tag \”&#60;/table&#62;\”.”;
      throw error_message;
    }
  }
  if (row[0])//check that there is contents in this row via the first element, which is ’Name’
     {
       groupTables.push(buildEmailRow(row));
     }
 }

 //make html email
 var email = buildEmail(AddAnn, AddComic, groupTables, announcement_doc_id, comic_sheet_id);
 return email
}



function sendAndSaveEmail(subject, recipient, html, save){
 //send email
 GmailApp.sendEmail(recipient, subject, ”testing”, {htmlBody: html});

 //save email if thats what we want to do
 if(save){
  saveEmail(html, HTML_folder);
 }
}



function cleanup(){
//copy the data from the Forms Sheet to the Archive Sheet
updateArchive(form_sheet_id, archive_sheet_id);
//remove the data from the Form Sheet
clearData(form_sheet_id);
//update comic sheet
updateComics(comic_sheet_id);
}


/*=======================================================================
    Helper Functions
    =================================================================================

These functions perform the heavy-lifting of the script such as reading
    from the Google Sheets/Docs, compiling the newsletter HTML file, etc

====================================================================================
    */


/*
    -------------------------------------------------------------------------
    getWeek()
    ---------------------------------------------------------------------------------

Description: Calculates the week number using Date() and Math functions,
    which is returned as an integer

Arguments:
-none
Returns:
-([week number] integer)
Notes:
-credits to Jarva for writing this function
------------------------------------------------------------------------------------
    */
function getWeek() {
  var d = new Date();
  d.setHours(0, 0, 0);
  d.setDate(d.getDate() + 4 - (d.getDay() || 7));
  return Math.ceil((((d - new Date(d.getFullYear(), 0, 1)) / 8.64e7) + 1)
    / 7);
}

```

```

/*------------------------------------------------------------
    retrieveAnnouncement()
    -----------------------------------------------------------------------------------------


Description: Grabs the announcement from the Announcement Google Doc as a
    string

Arguments:
-doc_id([GoogleDocId] string)
Returns:
-announcement(string)
Notes:
-------------------------------------------------------------------------------------------
    */
function retrieveAnnouncement(doc_id){
  var announcement = convertToHTML(doc_id);
  return announcement

}


/*----------------------------------------------------------------------
    retrieveComic()
    -----------------------------------------------------------------------------------------


Description: Grabs the comic URL from the Comic Google Doc as a string

Arguments:
-doc_id([GoogleDocId] string)
Returns:
-comic(string array)
Notes:
-------------------------------------------------------------------------------------------
    */
function retrieveComic(doc_id){
  var comic_data = [];
  var comic = SpreadsheetApp.openById(doc_id).getRange('B2').getValues();
  if(comic == null)
    throw "Error! No URL found for the comic. Please check the comic
    spreadsheet."
  var name = SpreadsheetApp.openById(doc_id).getRange('C2').getValues();
  if(name == null || name == '')
    name = 'Anonymous';
  comic_data[0] = comic;
  comic_data[1] = name;
  return comic_data
}


/*--------------------------------------------------------------------
    getSpreadsheetData()
    -------------------------------------------------------------------------------------
```

```

Description: getSpreadsheetData() returns the values from the specified
    range of the specified sheet as a [string?] array

Arguments:
-range([GoogeSheetRange] string)
-sheet_id([GoogleSheetId] string)
Returns:
-([array] string)
Notes:
------------------------------------------------------------------------------------
    */
function getSpreadsheetData(range, sheet_id) {
  var data_range = SpreadsheetApp.openById(sheet_id).getRange(range);
  return data_range.getValues()
}


/*-------------------------------------------------------buildEmailRow()
    ---------------------------------------------------------------------------------


Description: buildEmailRow() writes the tables that contain the weekly
    report data in HTML as a string array

Arguments:
-row([weekly report data item] string)
Returns:
-([HTML] string)
Notes:
------------------------------------------------------------------------------------
    */
function buildEmailRow(row) {
  return [
    '<table style="font-family:arial;" id="t01">',
    '<tr>',
    '<th>' + row[0] + ', ' + row[1] + '</th>',
    '<th>Project Name: ' + row[2] + '</th>',
    '</tr>',
    '<tr style = "background-color: ' + accentColor1 + ';">',
    '<td><b>This Week\'s Achievement</b></td>',
    '<td width=\"67%\">' + row[3] + '</td>',
    '</tr>',
    '<tr style = "background-color: ' + accentColor2 + ';">',
    '<td><b>This Week\'s Problem(s)</b></td>',
    '<td>' + row[4] + '</td>',
    '</tr>',
    '<tr style = "background-color: ' + accentColor1 + ';">',
    '<td><b>Next Week\'s Plan</b></td>',
    '<td>' + row[5] + '</td>',
    '</tr>',
    '</table>'
  ].join('\n')
}


```

```
/*-----------------------------------------------------------buildEmailHead
    ()
    ---------------------------------------------------------------------------

Description: Writes the header for the HTML file, which includes the CSS
    definitions for the tables/overall layout of the file, as a string
    array

Arguments:
-none
Returns:
-([HTML] string)
Notes:
-uncomment "'th {font-size:125%}'," if destination is gmail
------------------------------------------------------------------------------
    */
function buildEmailHead() {
  return [
    '<!DOCTYPE html>',
    '<html>',
    '<head>',
    '<style>',
    'table {width: 800px;margin: 20px;}',
    'table, th, td {border-collapse: collapse;}',
    'th, td {padding: 12px; text-align: left;}',
    /*'th {font-size:125%}',*/
    'table#t01 tr:nth-child(even) {background-color: ' + accentColor1 + '
    ;}',
    'table#t01 tr:nth-child(odd) {background-color: ' + accentColor2 + ';}
    ',
    'table#t01 th {background-color: ' + headerBGColor + '; color: ' +
    headerTextColor + ';}',
    '</style>',
    '</head>'
  ].join('\n')
}


/*-----------------------------------------------------------buildEmailAnn()
    ---------------------------------------------------------------------------

Description: Writes the header for the HTML file, which includes the CSS
    definitions for the tables/overall layout of the file, as a string
    array

Arguments:
-ann_doc_id([GoogleDocId] string)
Returns:
-([HTML] string)
Notes:
-add "font-size:125%;" to 'th style' if destination is gmail
------------------------------------------------------------------------------
    */
function buildEmailAnn(ann_doc_id) {
  return [
    '<table style="font-family:arial;" id="t01">',
    '<tr>',
    '<th style="text-align:center">Announcements</th>',
    '</tr>',
    '<tr style = "background-color: ' + accentColor1 + ';">',
    '<td>' + retrieveAnnouncement(ann_doc_id) + '</td>',
    '</tr>',
    '</table>'
  ].join('\n')
}


/*-----------------------------------------------------------buildEmailComic
    ()
    ------------------------------------------------------------------------------------------

Description: Writes the header for the HTML file, which includes the CSS
    definitions for the tables/overall layout of the file, as a string
    array

Arguments:
-comic_doc_id([GoogleDocId] string)
Returns:
-([HTML] string)
Notes:
-add "font-size:125%;" to 'th style' if destination is gmail
-----------------------------------------------------------------------------------------------
    */
function buildEmailComic(com_doc_id) {
  var com_data = retrieveComic(com_doc_id);
  return [
    '<table style="font-family:arial;" id="t01">',
    '<tr>',
    '<th style="text-align:center">Comic of the Week</th>',
    '</tr>',
    '<tr style = "background-color: ' + accentColor1 + ';">',
    '<td style="text-align:center"><img src="' + com_data[0] + '" alt="You
     win this round, [email client name]" width="776">Comic submitted by: '
     + com_data[1] + '</td>',
    '</tr>',
    '</table>'
  ].join('\n')
}


/*-----------------------------------------------------------buildEmail()
    ------------------------------------------------------------------------------------------

Description: Writes all of the HTML for the email as a string array

Arguments:
-ann(boolean)
-comic(boolean)
```

```
-groupTables([Google Sheet rows] string array)
-ann_doc_id([GoogleDocId] string)
-com_doc_id([GoogleDocId] string)
Returns:
-([HTML] string)
Notes:
-uses the [A?B:C] conditional to enable or disable the tables correspoding
    to the announcement and comic
-------------------------------------------------------------------------------------------------
    */
function buildEmail(ann, comic, groupTables, ann_doc_id, com_doc_id) {
  return [
    buildEmailHead(),
    '<body>',
    '<h2 style="font-family:arial;">'+ labName + ' Newsletter Week ' +
   getWeek() + '</h2>',
    ann ? buildEmailAnn(ann_doc_id) : ' ',
    groupTables.join('\n'),
    comic ? buildEmailComic(com_doc_id) : ' ',
    '<h5 style="font-family:arial;"> Software Copyright © Lizzy Hatfield
   2018 </h5>',
    '</body>',
    '</html>'
  ].join('\n')
}


/*
    ---------------------------------------------------------------------------
    saveEmail()
    ------------------------------------------------------------------------------------------


Description: Creates the physical HTML file out of the HTML code that was
    generated before

Arguments:
-email([HTML] string)
-dest_id([GoogleFolderId] string)
Returns:
-nothing
Notes:
-------------------------------------------------------------------------------------------------
    */
function saveEmail(email, dest_id) {
  var folder = DriveApp.getFolderById(dest_id);
  var date = new Date();
  folder.createFile('newsletter-wk' + getWeek() + '-' + date.getFullYear()
     +'.html', email, MimeType.HTML);
}


/*------------------------------------------------------------updateArchive
    ()
    ------------------------------------------------------------------------------------------
```

```

Description: Appends a new row of data to the Archive Sheet

Arguments:
-source_range([GoogeSheetRange] string)
-source_id([GoogeSheetId] string)
-dest_id([GoogeSheetId] string)
Returns:
-nothing
Notes:
-check for row[0] not being null, as the name field will be filled for
    every valid entry into the Form Sheet
-----------------------------------------------------------------------------------------
    */
function updateArchive(source_id, dest_id) {
  var dest_sheet = SpreadsheetApp.openById(dest_id);
  var source_data = SpreadsheetApp.openById(source_id).getSheets()[0].
    getDataRange().offset(1, 0).getValues();
  for (i in source_data) {
    var row = source_data[i];
    if (row[0])//if row[0] == anything
    {
      dest_sheet.appendRow(row);
    }
  }
}


/*--------------------------------------------------------------------
    clearData()
    ------------------------------------------------------------------------------------


Description: Deletes the rows of the Form Sheet in ascending order

Arguments:
-id([GoogeSheetId] string)
-depth([unsigned] integer)
Returns:
-nothing
Notes:
-----------------------------------------------------------------------------------------
    */
function clearData(id) {
  var range = SpreadsheetApp.openById(id).getSheets()[0].getDataRange().
    offset(1,0);
  range.clear();
}


/*--------------------------------------------------------------------
    alphabetizeSheet()
    ------------------------------------------------------------------------------------


Description: Sorts the rows of the Form Sheet alpabetically by name
```

```

Arguments:
-sheet_id([GoogeSheetId] string)
-sort_range([unsigned] integer)
Returns:
-nothing
Notes:
-two is for the second column which holds the ’Name’ field of the Form
    data
---------------------------------------------------------------------------------------------------
    */
function alphabetizeSheet(sheet_id, sort_range){
  var data = SpreadsheetApp.openById(sheet_id).getSheets()[0].getDataRange
    ().offset(1, 0);
  data.sort(2);
}


/*-----------------------------------------------------------errorEmail()
    -----------------------------------------------------------------------------------------


Description: Sends an email to the developer when there is an error from
    the GUI with the error message

Arguments:
-error_message(error.message, string)
Returns:
-nothing
Notes: emails developer on error
---------------------------------------------------------------------------------------------------
    */
function errorEmail(error_message){
GmailApp.sendEmail(devEmail, ”Newsletter Generator Error!”, ”Error Message
    : ” + error_message);
}


/*-----------------------------------------------------------shuffleSheet
    ()
    -----------------------------------------------------------------------------------------


Description:

Arguments:
-none
Returns:
-nothing (randomizes rows of the comic sheet)
Notes: only works on the comic sheet
---------------------------------------------------------------------------------------------------
    */
function shuffleSheet(){
  var sheet = SpreadsheetApp.openById(comic_sheet_id).getSheets()[0];
  var full_range = sheet.getDataRange();
  var range = full_range.offset(1, 0);
  range.setValues(range.randomize());
}//need to check for bad links, and an empty comic sheet


/*-----------------------------------------------------------updateComics
    ()
    ------------------------------------------------------------------------

Description:

Arguments:
-none
Returns:
-nothing
Notes:
-----------------------------------------------------------------------------
    */
function updateComics(comic_id){
  var d = new Date();
  //grab row 2
  var row = SpreadsheetApp.openById(comic_id).getSheets()[0].getDataRange
    ().getValues()[1];
  //place todays date in row[0]
  row[0] = String((d.getMonth() + 1) + '/' + d.getDate() + '/' + d.getYear
    ());
  //copy to sheet #2
  SpreadsheetApp.openById(comic_id).getSheets()[1].appendRow(row);
  //delete row 2 in sheet #1
  SpreadsheetApp.openById(comic_id).getSheets()[0].getRange("A2:C2").clear
    ();
}


/*-----------------------------------------------------------convertToHTML
    ()
    ------------------------------------------------------------------------

Description: Sends an email to the developer when there is an error from
    the GUI with the error message

Arguments:
-GoogleDocID(string)
Returns:
-output(string array)
Notes: This code is adapted from the work by Omar Al Zabir, source at:
    https://github.com/oazabir/GoogleDoc2Html
-----------------------------------------------------------------------------
    */
function convertToHTML(doc_id){
  var body = DocumentApp.openById(doc_id).getBody();
  var numChildren = body.getNumChildren();
  var output = [];
  var images = [];
  var listCounters = {};

  // Walk through all the child elements of the body.
  for (var i = 0; i < numChildren; i++) {
    var child = body.getChild(i);
    output.push(processItem(child, listCounters, images));
  }

  return output.join('\n')
}


function processItem(item, listCounters, images) {
  var output = [];
  var prefix = "", suffix = "";

  if (item.getType() == DocumentApp.ElementType.PARAGRAPH) {
    switch (item.getHeading()) {
        // Add a # for each heading level. No break, so we accumulate the
     right number.
      case DocumentApp.ParagraphHeading.HEADING6:
        prefix = "<h6>", suffix = "</h6>"; break;
      case DocumentApp.ParagraphHeading.HEADING5:
        prefix = "<h5>", suffix = "</h5>"; break;
      case DocumentApp.ParagraphHeading.HEADING4:
        prefix = "<h4>", suffix = "</h4>"; break;
      case DocumentApp.ParagraphHeading.HEADING3:
        prefix = "<h3>", suffix = "</h3>"; break;
      case DocumentApp.ParagraphHeading.HEADING2:
        prefix = "<h2>", suffix = "</h2>"; break;
      case DocumentApp.ParagraphHeading.HEADING1:
        prefix = "<h1>", suffix = "</h1>"; break;
      default:
        prefix = "<p>", suffix = "</p>";
    }

    if (item.getNumChildren() == 0)
      return "";
  }
  //else if (item.getType() == DocumentApp.ElementType.INLINE_IMAGE)
  //{
    //processImage(item, images, output);
  //}
  else if (item.getType()===DocumentApp.ElementType.LIST_ITEM) {
    var listItem = item;
    var gt = listItem.getGlyphType();
    var key = listItem.getListId() + '.' + listItem.getNestingLevel();
    var counter = listCounters[key] || 0;

    // First list item
    if ( counter == 0 ) {
      // Bullet list (<ul>):
      if (gt === DocumentApp.GlyphType.BULLET
          || gt === DocumentApp.GlyphType.HOLLOW_BULLET
          || gt === DocumentApp.GlyphType.SQUARE_BULLET) {
        prefix = '<ul class="small"><li>', suffix = "</li>";

```

```
          suffix += ”</ul>”;
        }
      else {
        // Ordered list (<ol>):
        prefix = ”<ol><li>”, suffix = ”</li>”;
      }
    }
    else {
      prefix = ”<li>”;
      suffix = ”</li>”;
    }

    if (item.isAtDocumentEnd() || item.getNextSibling().getType() !=
   DocumentApp.ElementType.LIST_ITEM) {
      if (gt === DocumentApp.GlyphType.BULLET
          || gt === DocumentApp.GlyphType.HOLLOW_BULLET
          || gt === DocumentApp.GlyphType.SQUARE_BULLET) {
        suffix += ”</ul>”;
      }
      else {
        // Ordered list (<ol>):
        suffix += ”</ol>”;
      }

    }

    counter++;
    listCounters[key] = counter;
  }

  output.push(prefix);

  if (item.getType() == DocumentApp.ElementType.TEXT) {
    processText(item, output);
  }
  else {


    if (item.getNumChildren) {
      var numChildren = item.getNumChildren();

      // Walk through all the child elements of the doc.
      for (var i = 0; i < numChildren; i++) {
        var child = item.getChild(i);
        output.push(processItem(child, listCounters, images));
      }
    }

  }

  output.push(suffix);
  return output.join(’’);
}


function processText(item, output) {
  var text = item.getText();
  var indices = item.getTextAttributeIndices();

  if (indices.length <= 1) {
    // Assuming that a whole para fully italic is a quote
    if(item.isBold()) {
      output.push('<b>' + text + '</b>');
    }
    else if(item.isItalic()) {
      output.push('<blockquote>' + text + '</blockquote>');
    }
    else if (text.trim().indexOf('http://') == 0) {
      output.push('<a href="' + text + '" rel="nofollow">' + text + '</a>'
 );
    }
    else {
      output.push(text);
    }
  }
  else {

    for (var i=0; i < indices.length; i ++) {
      var partAtts = item.getAttributes(indices[i]);
      var startPos = indices[i];
      var endPos = i+1 < indices.length ? indices[i+1]: text.length;
      var partText = text.substring(startPos, endPos);

      Logger.log(partText);

      if (partAtts.ITALIC) {
        output.push('<i>');
      }
      if (partAtts.BOLD) {
        output.push('<b>');
      }
      if (partAtts.UNDERLINE) {
        output.push('<u>');
      }

      // If someone has written [xxx] and made this whole text some
 special font, like superscript
      // then treat it as a reference and make it superscript.
      // Unfortunately in Google Docs, there's no way to detect
 superscript
      if (partText.indexOf('[')==0 && partText[partText.length-1] == ']')
 {
        output.push('<sup>' + partText + '</sup>');
      }
      else if (partText.trim().indexOf('http://') == 0) {
        output.push('<a href="' + partText + '" rel="nofollow">' +
 partText + '</a>');
      }
      else {
        output.push(partText);
      }

      if (partAtts.ITALIC) {
        output.push('</i>');
      }
      if (partAtts.BOLD) {
        output.push('</b>');
      }
      if (partAtts.UNDERLINE) {
        output.push('</u>');
      }

    }
  }
}
```

Code Block C.2: Newsletter Software JavaScript Source Code

D

Digital Flow Handbook

In order to help future digital designers working on the tools used by the CoolGroup (Genus, Innovus, etc). A handbook was started to compile all of the knowledge and helpful tips about these tools. It was created in hopes that continued documentation of engineering tools could help future designers learn faster, and make their designs better and more efficiently. What is complete from this work-in-progress is given in the following pages.

Digital IC Design Handbook

Applied Quantum Architectures

E. K. Hatfield

A Guide
to help you implement your digital crap in silicon

TUDelft

Acknowledgements and Preface

This guide was made in collaboration with and with contributions by Gerd Kiene, Augusto Carimatto...

This document was last updated December 2, 2018 and may be out of date...

Questions or comments may be emailed to: `e.k.hatfield@student.tudelft.nl`

Contents

1 Introduction

1.1. About this Guide

The purpose of this handbook is to help students through the process of implementing a digital design as an ASIC using the tools available for the CoolGroup at TU Delft, specifically targeting a 40nm TSMC process. This guide was written on December 2, 2018 and may be out of date with the server and/or tools. If you think that something is missing/incorrect/needs to be updated, then update this guide!

This guide assumes that you are familiar with the UNIX server environment, shell scripting, and are starting with a digital design that you are ready to synthesize etc, but some helpful hints and tips will be provided for these topics in the appendices.

1.2. Getting Set Up with the Tools in this Guide

1.2.1. Genus

To setup Genus in your fernet account, you only need to execute a few commands:

```
cd ~
mkdir <name-of-genus-run-directory>
cp /eda/cadence/2017-18/scripts/GENUS_17.11.000_RHELx86.csh ./<name-of-
    genus-run-directory>/
```

Code Snippet 1.1: Genus Setup

Then you can simply source the Genus sourceme (after sourcing the Virtuoso sourceme) and run the program with the comamnd ’genus’.

1.2.2. Innovus

Likewise for Innovus, you only need to run a few commands:

```
cd ~
mkdir <name-of-innovus-run-directory>
cp /eda/cadence/2017-18/scripts/INNOVUS_17.11.000_RHELx86.csh ./<name-of-
    innovus-run-directory>/
```

Code Snippet 1.2: Innovus Setup

The same goes for launching Innovus as it does with Genus.

1.2.3. Virtuoso & Calibre

To set up Virtuoso is simple enough. First you need to create a working directory for Virtuoso, as done for the other tools. Next you need to create a sourceme file inside that directory that has the following contents:

```
source /eda/scripts/flexlm.csh
source /eda/cadence/2016-17/scripts/IC_6.1.7.704_RHELx86.csh
source /eda/cadence/2016-17/scripts/MMSIM_15.10.627_RHELx86.csh
source /eda/mentor/2016-17/scripts/CALIBRE_2017.1-25.22_RHELx86_64.csh
source /eda/cadence/2016-17/scripts/ASSURA_04.15.107_IC617OA_RHELx86.csh

setenv CDS_AUTO_64BIT ALL
setenv USE_CALIBRE_64 YES
setenv SPECTRE_DEFAULTS -E
setenv CDS_Netlisting_Mode Analog
setenv CDS_AHDLCMI_ENABLE NO
```

Code Snippet 1.3: Virtuoso sourceme

Second, you need to create a file in the same place called "cds.lib" with the following contents:

```
INCLUDE $CDSHOME/share/cdssetup/cds.lib


DEFINE tsmcN40 /data/cad/DesignKits/TSMC/tsmcN40/kit_oa_1p7m_4X0Y1Z0R1U/
    tsmcN40
INCLUDE /u/55/55/cadence/IC615HF506/share/cdssetup/cds.lib
DEFINE avTech /u/55/55/cadence/ASSURA41_615_OA/tools/assura/etc/avtech/
    avTech


DEFINE tphn40lpgv2od3_sl /data/cad/DesignKits/TSMC/tsmcN40/
    tphn40lpgv2od3_sl/TSMCHOME/digital/Back_End/cdk_oa/
    tphn40lpgv2od3_sl_200a/mt_2/7lm
DEFINE tcbn40lpbwp_120b /data/cad/DesignKits/TSMC/tsmcN40/tsmc_40lp_std/
    tcbn40lpbwp_v120c/TSMCHOME/digital/Back_End/cdk_oa/tcbn40lpbwp_120b/
    tcbn40lpbwp

DEFINE BPADS /u/58/58/babaie/cadence_40/BPADS
DEFINE MOSLAY /u/58/58/babaie/cadence_40/MOSLAY
DEFINE IORING /u/58/58/babaie/cadence_40/IORING

DEFINE io_7m4x1z1u_lvs_atef /data/cad/DesignKits/TSMC/tsmcN40/
    kit_oa_1p7m_4X0Y1Z0R1U/io_7m4x1z1u_lvs_atef
#
DEFINE run_nov_2015 /users/atef/cadence_work37/run_nov_2015
DEFINE run_nov_2015_m /users/atef/cadence_work37/run_nov_2015_m
DEFINE run_nov_2015_odpo /users/atef/cadence_work37/run_nov_2015_odpo
DEFINE run_nov_2015_total /users/atef/cadence_work37/run_nov_2015_total
#
#
DEFINE tijdelijk_run_nov_2015 /users/atef/cadence_work37/
    tijdelijk_run_nov_2015
DEFINE tijdelijk_run_nov_2015_m /users/atef/cadence_work37/
    tijdelijk_run_nov_2015_m
DEFINE tijdelijk_run_nov_2015_odpo /users/atef/cadence_work37/
    tijdelijk_run_nov_2015_odpo
DEFINE tijdelijk_run_nov_2015_total /users/atef/cadence_work37/
    tijdelijk_run_nov_2015_total
#
```

Code Snippet 1.4: Virtuoso cds.lib

You can create these with emacs/gedit/vi/etc. The first is a script to prepare the shell for launching Virtuoso, and the second is a file Virtuoso relies on to manage cell libraries. Next, to set up the Calibre environment inside Virtuoso, you need to do the following [inside the CIW window once you launch Virtuoso]: [this is the current ad-hoc solution, please update with the robust solution later]

```
setenv MGC_HOME /u/55/55/mentor/Calibre/ixl_cal_2009.3_32.24
load(strcat("/u/55/55/mentor/Calibre/ixl_cal_2009.3_32.24/shared/pkgs/
    icv.ixl/tools/queryskl/calibre.skl"))
```

Code Snippet 1.5: Calibre Setup

Now when you open an item in the Layout Viewer in Virtuoso, you should be able to see a Calibre tab on the toolbar with options like Run nmDRC, Run nmLVS, etc. If that is the case, then you are ready to begin building your digital chip!

2 Design Synthesis with Genus

2.1. Introduction

The first step in taking a digital design from HDL behavioral description to chip is automatic synthesis. This means translating all of the constructs withing the HDL code such as case statements, if statements, arithmetic operations, and more into basic digital components like AND gates and flip flops from a standard cell library.

This guide is written for Genus version 17.11.000, and may become out of date. Genus is the successor of the Cadence tool RTL Compiler, and much of the information about RTL Compiler is relevant to Genus. Useful documents and references for Genus can be found on fernet; see Appendix B for a listing of these documents.

2.2. Building Your Script

2.2.1. Introduction

The Genus environment is built out of a console and this the the primary way to use it. Just like a UNIX shell, you can get an auto-complete/auto-suggest from the TAB key. Additionally, you can access help or man pages by using `help` and `man` with the name of the command. To get information about a Genus attribute, use `help root:` plus the name of the attribute.

Since Genus has mainly a console interface, it is best to run it by way of TCL scripts. The next sections will discuss the structure of these scripts.

2.2.2. Basic Structure of a Genus Synthesis Script

Within the Genus script, there are a few steps that are always performed in order to synthesize the HDL code to a netlist. These steps are as follows:

1. Load technology libraries
2. Load HDL code
3. Set constraints
4. Synthesize
5. Export netlist and constraints

The first step loads the technology files that describe the function and performance of the standard cells that the design will be synthesized into. The second step loads the behavioral HDL code that needs to be replicated as a netlist of standard cells. The third step sets synthesis constraints on timing and/or area for the final netlist. The fourth step performs the actual of the HDL to standard cells while trying to meet the constrains set in step three. Lastly step five extracts the Verilog netlist of standard cells and corresponding timing constraint file.

2.2.3. A Basic Script Example

These five basic steps can be implemented with the following script:

```
1  ###Step 1: load technology libraries and files
2  set_db lib_search_path {/data/cad/DesignKits/TSMC/tsmcN40/tsmc_40lp_std/
       tcbn40lpbwp_v120c/TSMCHOME/digital/Front_End/timing_power_noise/ECSM/
       tcbn40lpbwp_120b/}
3  set_db library {tcbn40lpbwpwc_ecsm.lib}
4  set_db design_process_node 40
5  set_db init_hdl_search_path {HDLSourceDir/}
6  
7  ###Step 2: load HDL code
8  read_hdl -language vhdl HDLSourceDir/Behavioral.vhd
9  elaborate
10 
11 ###Step 3: set constraints
12 set_top_module Behav
13 create_clock -name Clk -period 10 [get_ports Clk]
14 set_dont_touch_network [all_clocks]
15 
16 ###Step 4: synthesis
17 syn_generic
18 syn_map
19 syn_opt
20 
21 ###Step 5: export netlist and constraints
22 write_hdl Behav > ResultsDir/Behav_mapped.v
23 write_sdf -design Behav > ResultsDir/Behav_mapped.sdf
24 write_sdc Behav > ResultsDir/Behav_mapped.sdc
```

Code Snippet 2.1: Basic Synthesis Script

This script can be analyzed step-by-step:

Step 1 In this step, the tool is configured to target the desired standard cell library and technology node. In line 2, the tool is pointed towards the path of the standard cell library models, and in line 3 it is pointed towards the models themselves. Line 4 designates the process node (in nm), and line 5 sets the path of the HDL file(s).

Step 2 This step is straightforward; line 8 reads the source HDL file, and line 9 'elaborates' it. The process of reading in the HDL file and elaborating it syntax checks the HDL in a basic sense and also to verify that the constructs in the HDL are synthesizable/compatible with the tool.

Step 3 This step sets the clock frequency and optionally the setup and hold times [rise/fall?] of the clock(s). First, line 12 asserts the top module of the design, which is important if multiple designs are loaded (separate designs, not necessarily separate HDL files). Line 13 defines a clock in the design named 'Clk' and gives it a period of 10 ns (or a frequency of 100 MHz). The portion in the square brackets is an inline command which returns the ports connected to 'Clk' as an argument to this command (create_clock). This is the standard syntax for this operation. Lastly line 14 adds a protection to the clocks defined for this design that they should not be altered by the synthesis tool.

Step 4 Step 4 performs the synthesis of the design to a netlist of standard cells. Line 17 translates the HDL constructs into generic components (gates, flip flops, etc). Then line 18 translates those generic components to components in the loaded standard cell library while trying to meet the timing constraints set in step 3. Finally line 19 tries to optimize the final netlist of standard cells further.

Step 5 The last step in the script is very straightforward; it exports the netlist and corresponding constraints. Line 22 exports the netlist, line 23 exports the SDC (timing constraints) file, and line 24 exports the SDF (timing data) file.

2.2.4. More Genus Tricks

Although the script given in 2.2.3 will completely synthesize a design to the TSMC 40nm Standard Cell Library, there is still more that can be achieved with a synthesis script.

Lastly, consulting the PDFs/documentation for Genus will give the best information about the tool. See the relevant appendix for a listing of the most useful documentation on the fernet server.

2.3. Using the GUI

3

Place and Route with Innovus

3.1. Introduction

The second step in implementing a digital design is called 'place and route'. In this step, the standard cells instantiated in the netlist generated by the synthesis tool are placed within a floorplan of a chip and routed in a way that meets the timing constraints of the design.

3.2. Importing Your Design

3.3. Floorplanning Your Design

3.4. Making Your Pad Ring

3.5. Building Your Design with a Script

3.6. Verification

3.7. Exporting Your GDS and Netlist

3.7.1. Netlist Export

4 Verification with Virtuoso and Calibre

A

UNIX, Scripting, and QuestaSim

A.1. UNIX Tips & Tricks

In this section, some helpful UNIX tip and tricks will be presented. First, the host server for the CoolGroup will be discussed. Second, some helpful tips for using the command line will be presented. Lastly, a few of the most common UNIX commands will be explained.

A.1.1. Introduction to fernet.ewi.tudelft.nl

All of the data, software, and documentation regarding the TSMC 40nm process for the CoolGroup is on a server called fernet.ewi.tudelft.nl. The fernet server launches with the tcsh shell by default, and runs on the CentOS operating system, version 6.7. The server can be accessed with any SSH tool such as putty or MobaXterm. If you do not already have an account on this server, please speak with the server admin, Antoon Frehe. On this server, all of the CoolGroupers have accounts under the directory /users/, which is one among many other top level directories. The folders of interest are the /data/ and /eda/ folders. These folders are mentioned elsewhere in this guide. Lastly, any submissions of GDS etc files to IMEC will happen in the /users/qegds2/ directory. Note that this folder is always accessible via an explicit change directory call, but may become invisible after a period of inactivity.

A.1.2. Command Line Tips

When running in the command line, you can do a few things that will make your life easier:

- **The TAB Key** - the shell will list all of the possible items, like an auto complete function. Works for file names and commands.
- **The Up and Down Arrow Keys** - the shell will log all of the commands you run, so you can easily rerun an old command by 'scrolling up' in the log with the Up Arrow Key.
- **The .** - in ever directory file, there are two entries. One of them is the ., which indicates the location of the directory. You can think of it as 'here'. For example `cp /users/example/file.txt .` will copy the file file.txt to 'here', which is your current directory.
- **The ..** - the other entry in every folder is the .. entry. This simply indicates the location of the parent directory. For example, `cd ..` will move you to the parent directory of your current directory.
- **The Wildcard Character, *** - the wildcard character can be used in place of a character or string you don't know or don't care about. For example, if you have a bunch of files named 'sample' with different extensions like .txt, .tcl, .png, and you want to remove them all, you can simply run `rm sample.*`
- **Input Redirect, <** - the input redirect character will allow you to redirect the input to a program or command to a file you explicitly choose. For example, `sort < new.txt` will sort the items in new.txt.
- **Output Redirect, >** - the output redirect character will allow you to redirect the output from a program or command to a file you explicitly choose. For example, `echo "Hello World!" > new-file.txt` writes the line "Hello World!" to the file new-file.txt, overwriting any previous contents of the file.

- **Output Append,** >> - similar to redirect, except this one will append the output to the end of the destination file without overwriting the previous contents.
- **Pipelining with |** - the 'pipe' character allows you to directly feed the output of one program to another. For example, `ls | grep example | less` will take all of the files in the current directory from ls, feed it to grep, which then searches those files for the string "example", and finally sends those results to less to print out as a scrolling page.
- **Run in Background with &** - a useful command for any program that is run purely from a GUI, running a program such as `virtuoso &` will launch an instance of Virtuoso, but leave the shell open for further commands.
- **Multiple Commands at Once with ;** - you can send multiple commands to the shell at once if you separate them with a semicolon (;). The commands will be executed from left to right, and any invalid commands will print an error as normal, but not stop the execution of the following command(s).
- **The Escape Character, \** - whenever you want the shell or program to take a character as a literal instead of as a command, such as the wildcard character *, you must 'escape' the character command using \* so that the shell knows to take the character as its own character and not as a wildcard indicator.

A.1.3. List of Useful UNIX Commands

Some of the most common UNIX commands are explained below:

- `cd` - change directory. Will change your current working directory to the specified destination folder, or the home directory if none is given. Example: `cd ~/example/some-folder/`
- `man` - manual. Will open the manual page for a given command, listing all of the options and functions of the command. Example: `man grep`
- `ls` - list. Will list every file in your current working directory.
- `ll` - list long. Will list every file in your current working directory, along with basic information about each file including size, read-write permissions, ownership, etc.
- `cat` - concatenate. Will print out the contents of a text/ASCII file to the command prompt. Example: `cat textfile.txt`
- `tar` - tape archive. Will compress or decompress .tar or .tar.gz files.
  Compress example: `tar -czf dest-file.tar.gz source-folder` Decompress example: `tar -zxvf compressed.tar.gz`
- `vi` - visual. In-shell text editor. Has a 'command' mode and an 'edit' mode. There is a similar program called 'vim'. Example: `vi new.txt`
- `emacs` - A GUI-based text editor. Has plug-ins for various programming languages such as csh, tcl, and VHDL. The similar 'gedit' is also available.
- `find` - find. Traverses through the current working current and though child directories for the file specified. Example: `find . -name "*.tcl" -print`
- `pwd` - print working directory. Will print the full path name of your current working directory.
- `diff` - difference. Will print the differences between two files, if any exist.
  Example: `diff file.txt similar.txt`
- `cp` - copy. Will copy a specified file or directory to a specified location.
- `mv` - move. Will move a specified file or directory to a specified location.
- `rm` - remove. Will delete a specified file or directory.
- `mkdir` - make directory. Will create a directory with the given name in the current working directory.

- `grep` - global regular expression print. Searches through a specified file for a given string. Example: `grep string file.txt`
- `ps` - process status. Lists all running processes (or programs) with process ID on the server. Example: `ps -u aUserName`
- `kill` - Ends a process by ID. Used in conjunction with ps. Example: `kill -9 1337`
- `sed` - stream editor. Can perform regex operations on files.
  Examples: `sed -n '/^.u.an$/p' words.txt` and `sed 's/old thing/new thing/'`

A.2. Shell Scripting

This section will cover basic scripting concepts, the different kinds of shells, how to launch a script, how to make a file executable, how to make a basic script, and some extra useful tips.

A.2.1. Basic Concepts

The main goal of scripting is to automate your workflow in the shell environment. Anything you can do in the command prompt, you can do in a script (and more, even). So really the trick is to know what's possible, what you want to accomplish, and make a solution out of these two things. As with any programming problem, Google is your best friend, and as time goes on your scripts will get more powerful and sophisticated. Happy scripting!

A.2.2. The Different Shells

Although all the UNIX command prompts might look the same, there are actually many different shells available, each with different features and syntax w.r.t. scripting. The two main kinds of shells are Bourne shells and C shells. The former include the sh and bash shells, while the latter includes csh, and tcsh, which is the default shell for the fernet server. On the fernet server, the following shells are available: sh, bash, nologin, dash, tcsh, csh, ksh, zsh. The main difference between these two families of shells is that the C shells have a C language-like syntax. Many guides on the internet suggest to start with the Bourne shells if you are unfamiliar with shell scripting, but if you are reading this guide then you are certainly familiar with C and programming in general, meaning tcsh and the other C shells will work fine for you. All of the examples in this guide will be in tcsh syntax, but could be readily converted to Bourne syntax.

A.2.3. How to Make a Basic Script

Example of something to be scripted: Every time you want to run Genus or Innovus, you need to first source the sourceme file for Virtuoso, and then go to your Genus/Innovus working directory and source the sourceme file for Genus/Innovus, and finally run the program. Instead of typing these commands every time you want to run one of these programs, you can make a script that will do this all in one go.

In this case, the contents of the script is very clear – it will just have the five commands it takes to start Genus in this example. Next, a reference to the intended shell is needed on the first line of the file. In this case, the desired shell is tcsh, so that makes the first line `#!/bin/tcsh`. Lastly, we put all of these items together in a file using a text editor like vim, emacs, or nano, and save it with the extension [.csh]. The full example file is given below:

```
1 #!/bin/tcsh
2 #GenusStartScript.csh
3
4 cd ~/CadenceRunDir
5 source sourceme
6 cd ~/GenusRunDir
7 source /eda/cadence/2017-18/scripts/GENUS_17.11.000_RHELx86.csh
8 genus
```

Code Snippet A.1: Basic Script Example

The first line is a comment, but tells the shell that the tcsh shell is intended for this script, and hence is necessary. The second line is a comment that can be ignored. Lines 4 and 5 move the shell to the Cadence run directory which contains a copy of the Cadence sourceme file and sources it, which configures the shell to run Virtuoso. This is also necessary for running Genus (and Innovus) which are also Cadence tools. Lines 6 and 7 move the shell to the Genus run directory and sources the Genus sourceme file and sources it. Lastly line 8 starts Genus.

It is also possible to make a script executable using `chmod`. However, for this particular script it would not be useful to run it as an executable, i.e. with `./GenusStartScript.csh`, because then you would be sourcing `sourceme` and `.../GENUS_17.11.000_RHELx86.csh` (which set some global variables to run the programs) to a child shell, but these effects are not seen in the parent shell that the call is made from. Hence you will not be able to run Genus if you run the script in that way.

A.2.4. More Advanced Scripting Tips

Example of something more complicated to be scripted: imagine that you are running Genus in a directory called ’GenusRunDir’, but don’t yet know how to log the automatically generated ’cmd’ and ’log’ files in a user-specified location. Now you’ve got about 50 cmd files and 50 log files bogging down your directory, and you want to clean it up so you can find the things you are looking for. You know the commands to do this, but since this is happening every time you run Genus, you know it’s wise to make it as a script so you can simply run the script each time you want to clean out the genus directory.

The first thing to do is to decide what exactly you want the script to do. In this case, you want to move the files to another folder with the name "logs-up-to-" plus the date in the format "ddMonyy". Then you want to move these files to that folder so that the creation date information is retained and more importantly these files are removed from the Genus run directory.The full example file is given below:

```
1 #!/bin/tsch
2 #script to clean up log and command files from genus
3
4 set d=`date +%d%b%y`
5 mkdir ~/GenusRunDir/logs-up-to-$d
6 mv ~/GenusRunDir/genus.* ~/GenusRunDir/logs-up-to-$d
```

Code Snippet A.2: More Advanced Script Example

The first line is the usual comment to designate this file as a tcsh script. The second line is just a normal comment. Line 4 sets a variable called "d" using the command `date` as an argument. It is very important that the back tick `` ` `` is used as quotes for this command so that the shell knows this is not a string but a command and that the output of this command should go to the variable "d". Line 5 creates the sub-directory we want with the name "logs-up-to-ddMonyy" in the Genus run directory. The ’$d’ places the value of the variable ’d’ at that place in the string. Lastly line 6 moves all of the files starting with the name "genus." to the sub-directory made from line 5. This affects the Genus log files which have the name pattern "genus.log<num>", and the cmd log files which have the name pattern "genus.cmd<num>".

A.3. Digital Design with QuestaSim

There are a number of tools for simulating and verifying digital designs, mainly Xilinx and Altera/Mentor Graphics tools. In this section some tips for using the QuestaSim tool from Mentor Graphics will be presented. This information is based on software version 10.5c and may become out of date with newer software versions.

A.3.1. Making and Simulating a Project Using the GUI

The best way to start with QuestaSim is to build a project. A project construct in QuestaSim is really just a collection of source files (and configurations) which don’t all necessarily fall into the same library. A library is a folder with some special QuestaSim files that should correspond to libraries defined in your HDL design. HDL files are always compiled with respect to a particular library, be it defined in the HDL design or the ’default’ library called ’work’.

First, start up QuestaSim by sourcing the Cadence sourceme file, then the QuestaSim sourceme file, and lastly running the command `vsim`. Here you will see a ’Library’ window, a ’Transcript’ window, and a toolbar at the top. To make a project, go to the toolbar and click on "File > New > Project". A pop-up will appear. Choose your project’s name, directory, and default library. Now a ’Project’ window will appear, displaying the files within the project, so for now it is empty. There will also be a new pop-up, which asks to add new items to the project. Add any existing VHDL/Verilog/etc files you like by clicking "Add Existing File" then specifying the path to the file in the "File Name" prompt, and finally clicking "OK". Creating a new source file is a similar procedure: click "Create New File" then specifying the path and filename in the "File Name" prompt, selecting the file type under "Add file as type" and lastly clicking "OK". After this pop-up disappears, these prompts can be found again by right-clicking the "Project" window.

Once all of the files of the project are ready, they can be compiled by going to the toolbar, and clicking "Compile > Compile All". This is compatible for mixed language designs. Additionally right-clicking the "Project" window and selecting "Compile > Compile All" will compile all of the source files of the project. If the compile order matters, which it usually does, go to the toolbar, and click "Compile > Compile Order...". This brings up a prompt where the order of compile can be manually altered or automatically generated via the "Auto Generate" button. Again, this option can be accessed via right-clicking the "Project" window.

When the entire design has been compiled, it can finally be simulated. First, the simulation environment needs to be launched by going to the toolbar and clicking "Simulation > Start Simulation...". A pop-up appears, and the 'Design Unit' needs to be selected. This means the testbench entity name, which will be found within the library it was compiled into, as `libName.TBEntityName`. With the design unit selected, press "OK" and the simulation environment will launch. To be able to see any signals during the simulation, they need to be added to the waveform viewer. In the "sim" window of the simulation environment, a hierarchy of the entire design is visible to the left, and on the right the available signals can be seen (in the "Objects" sub-window). Signals can be dragged from the "Objects" window to the waveform viewer (window titled "Wave - Default"), or selected and added with a right-click and selecting "Add Wave". When all of the desired signals are on the waveform viewer, the column widths of the signal name, value, and waveform columns can be adjusted as desired. Once the waveform viewer is configured as desired, the simulation can be done. Under the toolbar the "Run Length" sets the run time length, either by the arrows or by manual type-in. Lastly press the "Run" button (next to the "Run Length" toggle) and the design will be simulated for the specified length of time. Press the "Zoom Full" button (or the 'F' key) to see the full waveform.

To save time and frustration, all of the commands corresponding to the configurations made to the waveform viewer, including the signals added, color settings, columns widths, etc., click the "Save Format..." button under the toolbar and press "OK". This file can be executed after launching the simulation environment using `do wave.do` in the QuestaSim console (window name "Transcript").

A.3.2. How to Use the Compare Tool

A very useful tool in QuestaSim for verifying your design is the comparison tool. This tool will sweep the signals between two waveform files and highlight any differences through time.

The first step is to save a reference waveform file. With all of the signals from a reference design point present on the waveform for the desired time period of evaluation, save the waveform file by clicking on "File > Save Dataset..." on the toolbar.

Next, click on "Tools > Waveform Compare > Comparison Wizard". Click on "Browse..." and choose the waveform that you previously saved, then click on "Next >". Select "Compare All Signals" (or another if you prefer) then click "Next >" again. Click "Next >" one more time, then finally "Compare Differences Now" and "Finish".

On the waveform viewer under any signals that were already present, new signals labeled "compare:" will now be visible. Any signals that have no differences will appear with a yellow triangle, while those with differences will have a red X over the triangle. To see where the differences are in time, look for red highlighting on the signal in the waveform. To see the differences in the values, expand the "compare:" signal and the two compared base signals will appear below. Highlighting is preserved on these signals too.

Note that the comparison wizard will stop comparing in time once a maximum number of differences has been found. This setting can be changed at "Tools > Waveform Compare > Options... > Comparison Limit Count" on the toolbar.

Note that this method requires the signals to have the same name and same timing to work.

Note that this procedure will print the respective commands to the console, which can be saved into a TCL script to make the comparison easily repeatable.

A.3.3. Using the QuestaSim Console

The best way to use QuestaSim efficiently is to make use of scripts and the console. It is possible to run commands from a TCL script in the QuestaSim console; it is even possible to execute a python script from it.

In general, the QuestaSim console works like a UNIX/Linux console: you can scroll through a log of previous commands, there is an auto-complete function, etc. In addition, QuestaSim has its own set of commands, mainly relating to the simulation environment, which can be executed. Below is a listing of some useful QuestaSim commands:

- **`quit <option>`** - quits the QuestaSim tool. Using the -f option forces the exit without any pop ups. Using the -sim option only quits the simulation environment
- `help` - acts like the command `man` from UNIX; gives information about a command or item
- `source` - [shell command] executes a script file
- `do` - executes a DO file (similar to source)
- `vsim` - launches specified design into the simulation environment.
  Example: `vsim -t ps work.hdlDesign(arch)` launches the 'arch' architecture of the entity 'hdl-design' of the library work with the timescale of picoseconds. Specifying the timescale is always necessary for a mixed-language design/project, and specifying the architecture is only necessary when multiple architectures for the design under test are defined.
- `run` - runs the simulation for the specified time e.g. `run 400 us` or `run -all` (the latter runs the simulation until a breakpoint is reached)
- `restart <option>` - restarts the simulation back to time zero. Signals, waveform/window settings are preserved (unless certain options are used). Example: `restart -f`
- `file <option>` - [TCL command] allows for the manipulation of files.
  Examples: `file copy vsim.wlf newFile.wlf` and `file delete oldFile.wlf`
- `log -r /*` - logs all of the signals in the design loaded into the simulation environment and saves them to the default waveform file (vsim.wlf)
- `vcom` and `vlog`
- `vlib` - creates a library with the specified name. Example: `vlib work`
- `vdel` - deletes QuestaSim files. Example: `vdel -all -lib work`

Happy scripting!

List of Code Snippets