An Abstract of the Thesis Of

Christopher D. Edmonds for the degree of Honors Baccalaureate of Science in Electrical Engineering and the degree of Honors Baccalaureate of Science in Computer Science presented on March 10, 2009. Title: Design of a Process Independent Tool for the Creation of On-Chip Serial Test Interfaces

Abstract Approved:

_____

Pavan Kumar Hanumolu

The push towards higher performing and more sensitive mixed signal circuitry has required the parallel development of increasingly more complex and sensitive test and calibration harnesses. Current off-chip methods of test and calibration may require higher pin counts or induce unwanted parasitic interference.

In this thesis, the design of a process independent tool for the creation of on-chip serial test interfaces is presented. This tool is part of a complete on-chip calibration solution designed by the author and his group for use in mixed signal integrated circuits. The system accomplishes several goals reducing pin count, minimizing parasitic interference, and facilitating test automation for the calibration and test harnesses by moving the test harness onto the die of the mixed signal integrated circuit. The tool presented in this paper was designed with the goals of easing test harness implementation, facilitating multiple fabrication processes, and providing flexibility to the chip designer. A set of serial test interfaces generated by the tool was fabricated in a 0.5um CMOS process in order to verify the both tool and its output.

Design of a Process Independent Tool for the Creation of On-Chip Serial Test Interfaces

by

Christopher D. Edmonds

A THESIS

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degrees of

Honors Baccalaureate of Science in Electrical Engineering
and
Honors Baccalaureate of Science in Computer Science

Presented March 10, 2009
Commencement June 2009

Honors Baccalaureate of Science in Electrical Engineering and Honors Baccalaureate of Science in Computer Science project of Christopher D. Edmonds presented on March 10, 2009.

APPROVED:

_____

Mentor, representing Electrical Engineering

_____

Committee Member, representing Electrical Engineering

_____

Committee Member, representing Electrical Engineering

_____

Director, School of Electrical Engineering and Computer Science

_____

Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

_____

Christopher D. Edmonds, Author

Acknowledgements

I would like to thank:

Richie and Dan for making this project a success.

Dr. Hanumolu for helping us along the way and the initial project concept.

Deirdre’ for her unfailing love and support even when I had to work late.

Table of Contents

List of Figures

List of Tables

List of Appendices

Section 1 Introduction

Current trends in integrated circuit design point towards tighter integration of both analog and digital systems on the same piece of silicon. As feature sizes decrease these mixed-signal systems continue to incorporate a greater number of circuits and transistors. In order to accommodate decreasing feature sizes and increasing integration many mixed signal circuits especially those that are prototypical in nature require some sort of characterization and calibration.

Traditionally this sort of calibration is accomplished by attaching each node in need of calibration to an external pin. While simple, this method of test and calibration is undesirable for several reasons. First, allocating a pin to every node that needs to be calibrated can greatly increase the overall pin count of a chip. Increased pin count requires more bonding pads to be placed on the die along with a need for larger packages with higher pin densities. These increases in turn raise both the cost of packaging and the cost of the die itself. Second, each node that is attached to an external pin is subjected to higher parasitic interference than it is through on-chip connections. Off-chip signal generators, power supplies, and probes can further increase unwanted parasitic interference. Finally, there is no simple or standardized manner in which to automate tests. Current methods of testing require the use of precision power supplies and function generators.

Past test interface development has traditionally had a focus on purely digital systems. The JTAG interface, for example, is designed to stimulate digital circuitry using a simple scan chain interface. Test systems such as the Joint Test Action Group (JTAG) IEEE 1149.1 lack methods to test or provide calibration signals to analog circuitry. There

is an effort to add mixed signal functionality to the JTAG interface[2]. IEEE 1149.4 is an extension to the JTAG standard which adds some analog functionality to the test interface, but it is designed to measure external component values and internal impedances rather than for calibration and characterization [2].

The design of a new test interface should then work towards meeting three goals. First, the test interface must not require nodes to be routed off-chip for test purposes increasing pin count and unwanted parasitic influence. Second, the test interface must be process independent. Third, the test interface should decrease the cost and increase the simplicity of test automation.

As a solution to this problem a group of three senior design students including myself proposed, designed, and implemented the Mixed-Signal Test Interface (MTI). MTI is an on-chip solution designed to provide both digital and analog stimuli to mixed signal circuitry. In order to ease implementation of the MTI our group determined to create an easy to use software application (MTI-Designer) for automatic generation of large portions of the interface. This thesis will cover the design and implementation of MTI-Designer.

Section 1.1 Thesis Scope

The MTI is comprised of several components that were all designed and implemented by a senior design project group that the author was a member of. This paper will focus on the author's work, which includes the architecture of the MTI digital layout macro (DLM), and the software suite designed to automate its creation, MTI-Designer. The design of the DACs used in the MTI has been covered in more depth by another member of the project group [7].

Section 2 Project Overview

The Mixed-Signal Test Interface (MTI) is an on-chip test and calibration harness designed to provide multiple low frequency analog and digital stimuli simultaneously to a user's circuit. MTI interfaces with the outside world through a simple four-wire serial interface, which allows bonding pad and pin counts to remain low. The MTI system consists of the MTI architecture itself and a software tool, MTI-Designer, to aid users in their implementation of the MTI.

Section 2.1 Mixed-Signal Test Interface Architecture

The MTI architecture is comprised of three primary components (Figure 1). The first of these is an automatically generated digital layout macro (DLM) that allows the MTI to receive stimuli instructions from off-chip. The outputs of the DLM may be directly connected to the circuitry under test as digital stimuli. The MTI DLM also provides digital input to the DACs to create the required analog stimuli. Analog and digital stimuli generated by the MTI are then connected to the circuitry under test in order to test, characterize and calibrate it.

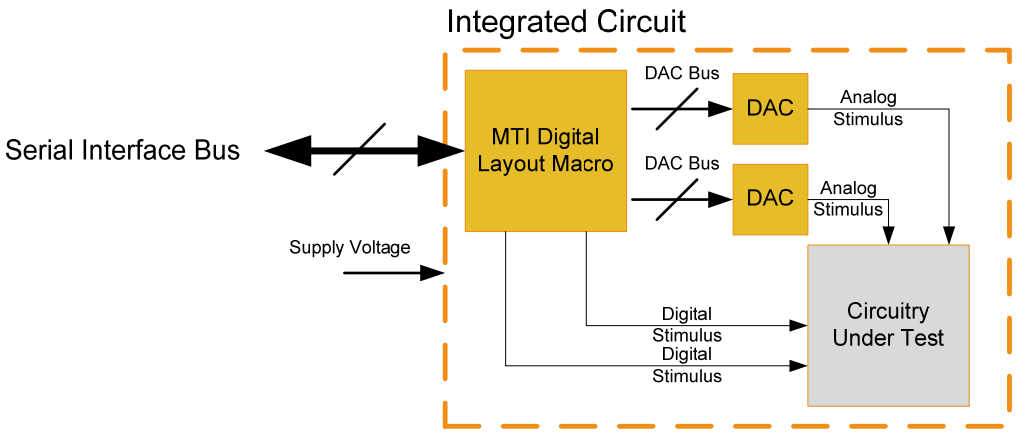

Figure 1: Top-level block diagram of the MTI

Section 2.2 MTI Designer

MTI Designer is a software suite designed to aid users in implementing the MTI. MTI Designer assists the user in designing the MTI DLM by collecting information about the design such as the number of outputs needed, the frequency at which the interface must run, and the physical dimensions of the macro. Additionally MTI-Designer collects process information to be used in creating the layout for the digital circuitry in the MTI implementation.

Section 3 Approach

The requirements for the MTI DLM and MTI-Designer were solicited from graduate students at Oregon State University working on prototypical analog-mixed-signal (AMS) circuits.

Section 3.1 Design Requirements

In order to best support the widest range of mixed-signal projects there are three primary goals for the MTI DLM and the software that generates it, MTI-Designer. First, a wide range of CMOS processes must be supported. Second, the software and the resulting layout should be relatively easy to generate and to integrate as opposed to a custom layout. Finally, not all process design kits include a standard cell library, and not all standard libraries have the same cells, so the design should depend on a minimum number of standard cells.

Section 3.1.1 Process Independence

Users of the MTI utilize a wide variety of CMOS processes for the development of mixed-signal circuits. The number of metal layers varies from process to process, as well as the minimum pitch sizes. In an attempt to support this wide variation there are two sub-goals that have been identified. First, the design of the digital architecture should be as simple and robust as possible. Second, the digital circuitry must be constructed using standard cell design. Standard cell design allows for digital circuits to be constructed from a library of pre-routed cells that implement basic logic functions (AND, OR, XOR).

Section 3.1.2 Automation

To increase adoption and usefulness of the MTI it is important that implementing the MTI be as automated as possible. In order to make the MTI a turnkey solution the digital circuitry necessary to implement the MTI must be generated as a layout macro based on user needs and specifications. Ideally the user should not need to understand the internal construction of the MTI.

Section 3.1.3 Standard Cell Library

While many Process Design Kits (PDKs) include some sort of standard cell library there is wide variability in the number and types of cells provided. Digital designs can be made more power and space efficient with larger libraries. In order to support a wide range of processes, PDKs, and standard cell libraries it is important that the digital portions of the MTI be relatively simple in order to decrease size and space inefficiencies that may come from smaller libraries.

Section 3.2 Serial Interface Selection

The serial interface is the means by which the MTI will receive information about which stimuli signals to apply to the circuitry under test. The serial controller needs to be simple to implement and small in size. Table 1 lists a broad cross section of the available serial communications and test interfaces that are currently used in the market. The interfaces range from the PC based USB protocol to the JTAG test protocols. This section will elaborate on the information presented in Table 1.

Table 1: Comparison of serial interfaces

Interface Name	References	Flip-Flop Overhead[1]	Pins	Typical Applications	Standard	PC Interface	Throughput	Device Addressable	Internal Clock	Data Verification
USB – Universal Serial Bus		499	2	PC Peripherals	Yes, Intel	No	12Mbps	Yes	Yes	Yes
JTAG – Joint Test Action Group	[1]	72	4-5	Digital Self Test	IEEE 1149.1	Yes	System Dependent	Yes	No	No
JTAG-AMS – Analog Mixed Signal Extension	[2], [10]	N/A[2]	6-7	AMS Self Test	IEEE 1149.4	Yes	System Dependent	Yes	No	No
I2C – Inter-chip Interface	[5], [4]	113	2	Motherboard, Embedded	Yes, Philips	Yes	400Kbps, 1Mbps, 3.4Mbps	Yes	No	No
SPI – Serial Peripheral Interface	[9], [4]	None[3]	3-4	Motherboard, Embedded	No[4]	Yes	1~25Mbps	No	No	No
One-Wire	[4], [1]	128	1	Sensors	Yes, Maxim	Yes	15Kbps,125Kbps	Yes	Yes	Yes[5]
UART – Universal Asynchronous Receive/Transmit	[9], [4]	90	2	Peripherals, Embedded	RS-232[6]	No	~1Mbps	No	Yes	Yes[7]
CAN – Controller Area Network	[9], [4], [2]	525	2	Automotive, Industrial	Yes, Bosch	Yes	up to 1Mbps	No	No	Yes[8]
Firewire	[3]	N/A[9]	4	Video, Hard disk drives	IEEE 1394	No	400Mbps	Yes	Yes	Yes

1 Estimates based on implementations found at www.opencores.org. This is merely meant to be a way to rank the chip area cost of an implementation.
2 This is not a purely digital interface therefore an analysis of flip-flop overhead would not correlate with the chip space required to implement the interface.
3 Since SPI stores incoming data in the same register from which data is read, it does not have to perform any addressing or decoding, and there is not flip-flop overhead, just logic overhead.
4 There is no official standard for SPI, though it is used by many different manufacturers in their embedded devices.
5 One-wire can implement an 8bit CRC [1].
6 RS-232 Applies to the data format if not the electrical interface. There would need to be an off chip charge pump to meet the standard.
7 Data can be verified with a parity bit.
8 CAN implements a 15 or 16 bit CRC for every frame [9].
9 Unable to find an implementation of IEEE-1394 at www.opencores.org.

In order to minimize die area used by the MTI it is important that the serial control logic be of minimal size. In order to determine the area of each serial controller, open source HDL code from www.opencores.org was compiled using Xilinx design tools for implementation in an FPGA. Table 1 details the number of flip-flops required to implement a controller for each protocol. These flip-flops represent only the overhead for the controller and do not include the cells that will be necessary to store the stimuli data.

The list includes two current test interfaces standards. These are IEEE 1149.1 (JTAG) and IEEE 1149.4 (JTAG-AMS). IEEE task forces architected these interfaces in order to help industry standardize their tests. Use of these interfaces would allow standard test tools to be used to communicate with the MTI.

JTAG is designed as a boundary scan chain. This means that the test controller communicates with a number of scan cells each associated with one of the chips pins. These cells can read the digital value on a pin or replace the external logic level with a simulated one [1]. While useful for testing both the chip and the surrounding circuitry this concept does not entirely meet our requirements. In contrast, the MTI is more concerned with signals that are not present on pins.

The JTAG-AMS standard is an extension of the JTAG standard [2]. In addition to providing boundary scan cells for digital circuitry it provides the means for implementing analog boundary scan cells. This system also seems quite applicable on first glance. While the inconsistencies in test ideology that exist in JTAG also exist in JTAG-AMS there is another issue. The analog boundary scan cells are designed to multiplex in an input in place of a real world signal [10]. Unfortunately this signal must be brought in through a pin. The JTAG-AMS standard thus requires a separate pin for each stimulus

and does not reduce the overall pin count for the chip, which is one of the primary goals of the MTI.

Clocked interfaces are those that require an independent clock in order to function properly. Some examples of these are USB, Firewire, 1-Wire, UART, and CAN [3], [1],[4],[9]. Generating a clock for these interfaces would require the use of an oscillator and on-chip clock distribution. In addition, the space requirements (determined by the number of flip flops needed) are in general higher for the interfaces that require their own clock. Due to the complexity and larger area required by these protocols they will not fit the needs of the MTI.

I2C is used commonly as a low frequency bus. It requires that there be a single master device. The slave devices are addressable and must each have unique codes associated with them. In addition I2C provides fault tolerance through the use of a checksum along with ACKs and NAKs [9]. The low number of pins required to implement I2C is a definite advantage of the interface. The checksum would guarantee that communications occurred in a fault free manner.

I2S is another low frequency embedded bus. This interface is specifically designed to accommodate digital audio signals. Since the DACs in the test circuitry are required to operate at frequencies within the audio range the data-rate of this bus would be appropriate. I2S also incorporates a word clock in addition to a bit clock [6]. The word clock is designed to sync the DACs together. This could allow the DAC transitions to be more monotonic in nature. I2S requires four pins to implement and has a larger controller overhead than SPI or I2C.

SPI is the last interface considered. SPI has separate pins for data input and output and does not have any transmission acknowledgment built in [9]. An SPI interface consists of a shift register that takes in data on one end and outputs on the other. For each clock one bit is shifted in and one bit is shifted out [9]. For this reason SPI has almost no overhead since the data does not need to be addressed or decoded. Since there is no addressing scheme built into SPI a multi-chip system would require each chip to have its own chip select line. SPI has a higher pin count than CAN or I2C.

SPI was selected as the serial interface for the MTI. While SPI has a higher external pin count than many of the other interfaces SPI has the least design complexity and flip-flop overhead. Flip-flop overhead dictates the size of the state that the communications controller must store in order to communicate. This state allows for more complex communications but also makes the logic to control serial communications more complex. In order to enhance the robustness and decrease the area overhead of the MTI DLM simplicity was chosen over pin-count.

Section 3.3 Software Selection

There are two ways in which the serial interface and stimuli registers could be constructed for the MTI. The first is a full custom solution that requires every piece of the MTI DLM must be placed by hand. A custom layout would save both space and power, but is would take significantly more time and effort to implement. The second option is to describe the operation of the MTI DLM in RTL register transfer logic (RTL) and then use tools to convert it into a layout appropriate to the user's CMOS process.

In order to turn the description of the serial interface into a physical layout that can be incorporated into the users design it must first pass through a set of tools. These

tools perform two primary steps in turning the RTL description into physical gates; these steps are referred to as synthesis and place and route (PnR).

Section 3.3.1 Synthesis Tool

Synthesis is the process by which a high-level architectural description of a digital design is turned into individual gates. The synthesis tools take as input the RTL code to convert into gates and a description of the digital cell library for a given process. The RTL is then converted into a Verilog net-list containing a circuit logically equivalent to that described in the RTL but comprised only of cells in the standard cell library.

Section 3.3.2 Place and Route Tool

Many analog and mixed-signal designers rely on Cadence Virtuoso in order to create layouts for their designs. Cadence System-on-Chip (SoC) Encounter is a widely used PnR tool for digital designs and is compatible with Virtuosos. Encounter is able to take in a net list of standard cells and create a functioning layout that can be imported into Virtuoso to be combined with the user's mixed-signal circuit.

Section 3.3.3 User Interface

The software tools selected for the MTI project are primarily run on Linux systems though they can also be run under Windows or other flavors of Unix. In order to accommodate this diversity of operating systems the user interface must rely on platform independent software solutions. Due to its widespread availability on Linux machines and the availability of a Windows interpreter Python was selected as the primary programming language for MTI Designer.

The GIMP toolkit (GTK+) is a cross-platform widget toolkit for the creation of graphical user interfaces (GUIs). The Python bindings for GTK+ (PyGTK) allow Python

code to run unmodified on Unix, Linux, Mac OSX, and Windows machines [8]. The use of PyGTK and GTK+ will also allow the user interface to maintain uniform appearance and functionality across all platforms.

Section 4 Digital Architecture Design

The design description is broken into two pieces. This first section covers the architectural design of the digital logic necessary to create the MTI DLM. The MTI DLM provides the MTI with off-chip communication and its outputs can be directly sent to the circuitry under test as digital stimuli. Additionally, its outputs can be passed through a DAC in order to create analog stimuli.

Section 4.1 Top Level Design

The digital outputs of the MTI DLM are addressable as 10-bit registers that can be both read and written. The decision to use 10-bit words as opposed to more traditional byte oriented word sizes was a result of the needs of the analog architecture of the MTI. This design decision is elaborated upon elsewhere [3]. It is also important for the DACs attached to the MTI that all bits of the 10-bit word change simultaneously. This prevents the node attached to the DAC from cycling through voltage levels as the 10-bit word is shifted in.

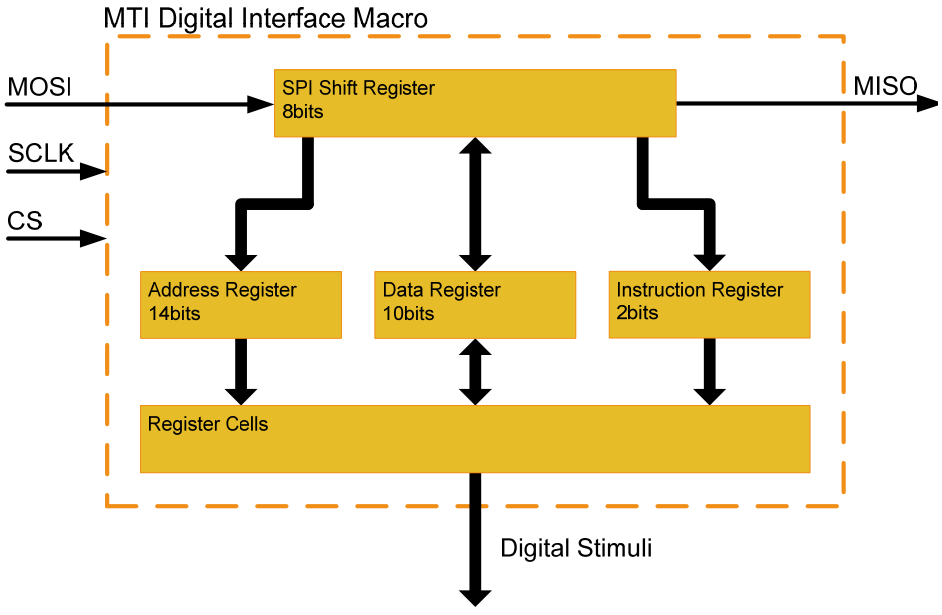

Figure 2: MTI Digital Layout Macro – Major Internal Blocks

Figure 2 shows the internal architecture of the MTI DLM. SPI communications are byte oriented so instructions are read in as 8-bit words and then stored in the appropriate registers. Communication starts with the selection of the MTI when CS goes low. The first byte clocked into the chip is stored in the two-bit instruction and the upper six bits of the address. The second byte clocked in is stored as the lower eight bits of the address.

If the instruction bits specified a write operation then the next byte will be the upper eight bits to be written to the register and the final byte will contain the lower two bits of the data. The data will not be written to the register until the CS signal goes high deselecting the chip. Timing for the write operation can be seen in Figure 3.

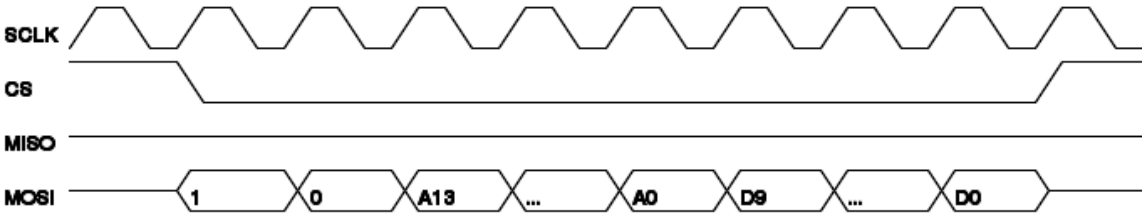

Figure 3: Write timing diagram

If the instruction bits specified a read-operation then the next byte will be the upper eight bits of the data register being read and the final byte will contain the last two bits of the register being read. Timing for a read operation can be seen in Figure 4.

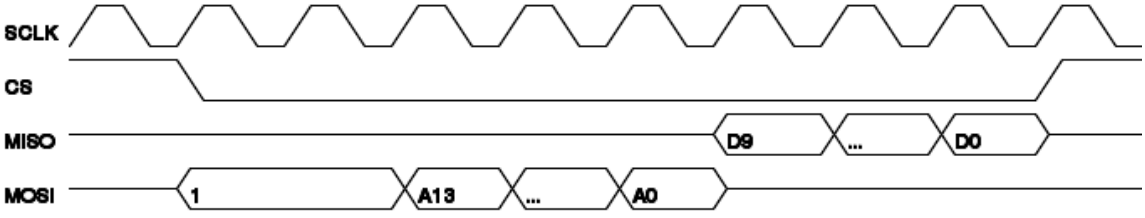

Figure 4: Read timing diagram

Section 5 Software Tool-chain Design

MTI Designer is the software suite for generating the MTI DLM. It works in conjunction with Cadence SoC Encounter and Cadence Virtuoso to create an implementation of the MTI DLM.

Section 5.1 Top Level Description

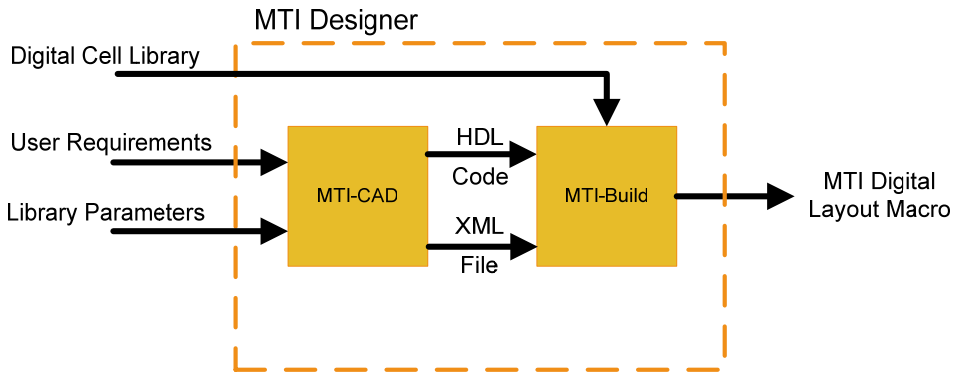

Figure 5: MTI Designer – Top Level Block Diagram

MTI-Designer consists of two components MTI-CAD and MTI-Build. MTI-CAD gathers user requirements and process information through its user interface. User requirements are used to modify a Verilog code template with the desired number of outputs. This Verilog code along with an XML file containing both the user requirements and process information are then passed to MTI-Build.

MTI-Build orchestrates the electronic design automation (EDA) tools in the implementation of the MTI DLM. In order to accomplish this MTI-Build takes EDA script templates and modifies them with information provided by the user about their CMOS process. The status of MTI-Build is fed back to MTI-CAD in order to keep the user informed about implementation status.

Section 5.2 MTI-CAD Functional Block Description

MTI -CAD provides a graphical user interface (Section 3.3.3) that is used to input the number and type of stimuli that the MTI will need to provide. With this information the MTI-CAD software can produce Verilog code that describes an MTI DLM that meets user requirements.

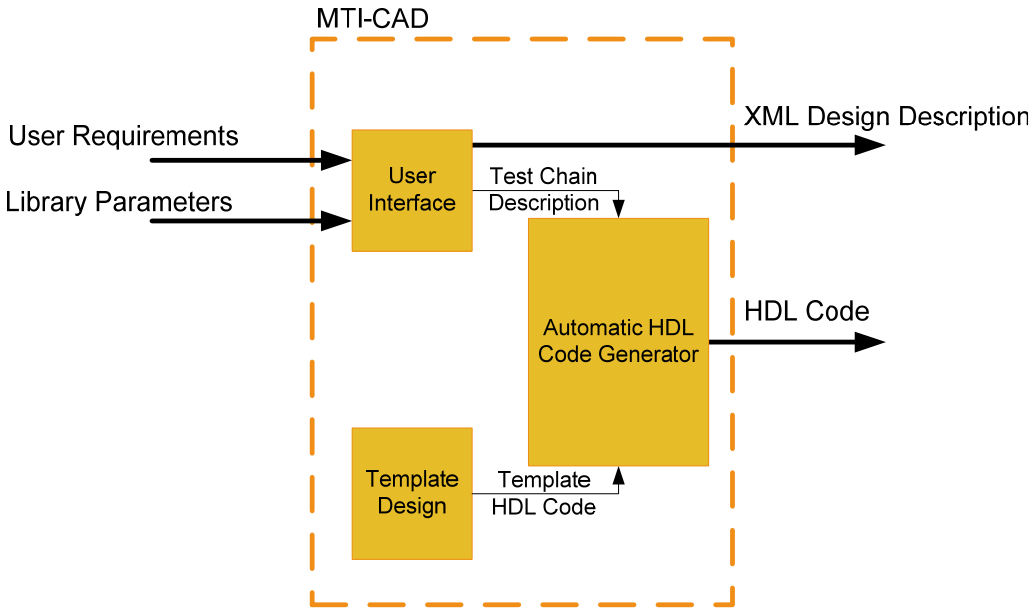

Figure 6: MTI-CAD – Major Internal Blocks

Section 5.2.1 User Interface

The user interface for MTI-CAD was designed using Python and the GTK+ bindings for Python, PyGTK. The user interface is the only portion of MTI-Designer that the user must interface with and it orchestrates all other tasks performed by MTI-Designer. Organization of the user interface is split into four panes.

The design pane collects basic information about the serial controller to be implemented. This information includes the desired aspect ratio of the controller, the number of outputs, and the clock frequency at which the serial interface is expected to be run. In addition the user can specify which side of the macro they require the serial

interface and outputs to be on. Table 2 provides a description of each field and the range of valid inputs.

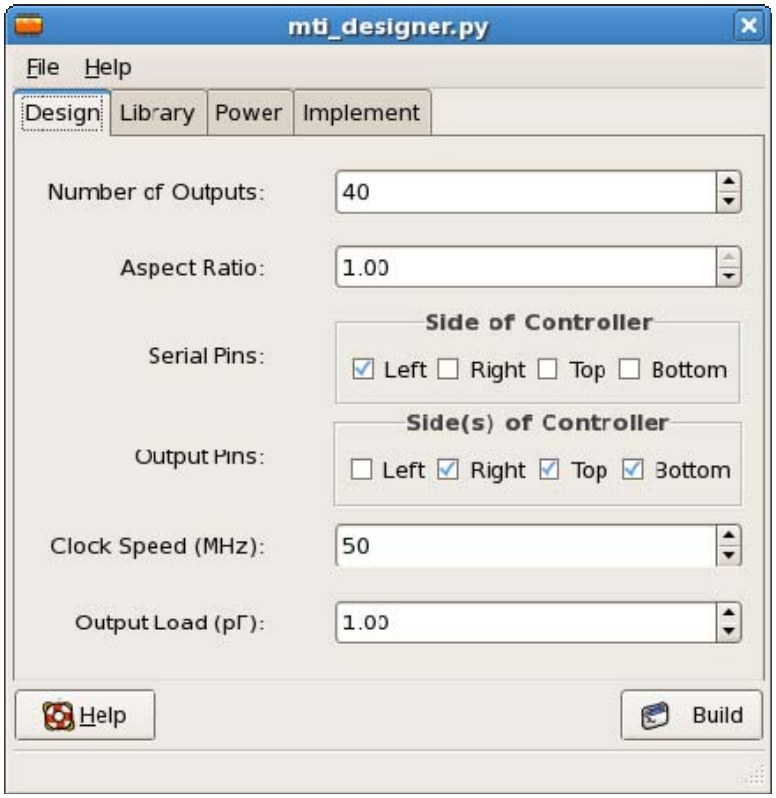

Figure 7: The serial-controller design tab

Table 2: Description of design tab components

Interface Component	Description	Valid Range
Number of Outputs	The total number of parallel outputs that the serial controller should have.	10 - 1000
Aspect Ratio	This value controls how rectangular or square the serial controller layout is.	0.10 – 1.00
Serial Pins	Specifies Which side of the serial controller layout should the serial control lines be on.	N/A
Output Pins	Which side of the serial controller layout should the output pins be on?	N/A
Clock Speed	The desired clock speed in MHz of the serial clock.	1 - 100
Output Load	The capacitive load in picofarads that the outputs of the serial controller will see.	1.00 – 5.00

The second pane of the user interface is designed to allow the user to input necessary information about their digital cell library. Information required includes the locations of timing files, technology files, and a Verilog file describing the digital cells. Information collected in this pane is passed on to both the synthesis and PnR tools. Table 3 provides a description of the information gathered in the library pane.

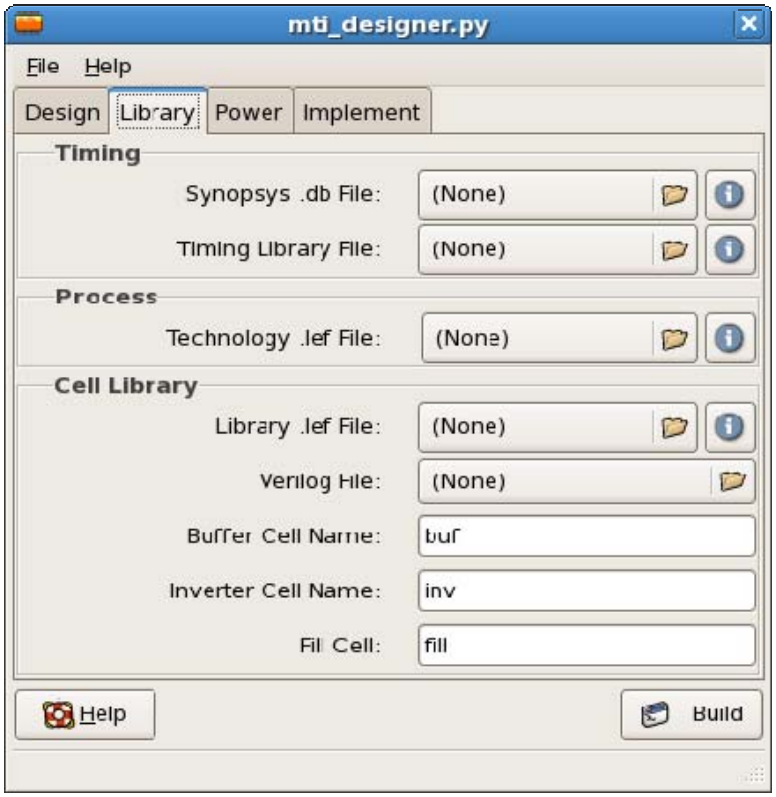

Figure 8: The cell library tab

Table 3: Description of library tab components

Interface Component	Description
Synopsys .db File	Contains information about the digital cell libraries timing and logical properties.
Timing Library File	Timing information used by the PnR tool.
Technology .lef File	Provides information about the CMOS process.
Library .lef File	Provides information about the cells in the digital library.
Verilog File	A Verilog description of each cell in the library.
Buffer Cell Name	The name of the cell to be used as a buffer.
Inverter Cell Name	The name of the library cell to be used as an inverter.
Fill Cell	The name of the library cell to be used as a filler. The filler cell normally only contains power and ground rails.

Power and ground rings must be routed around the MTI DLM so that the PnR tool can connect each digital cell to power and ground. The power pane of the user interface allows the user to input the names of the global power and ground nets along with the desired widths and layers for the power rings.

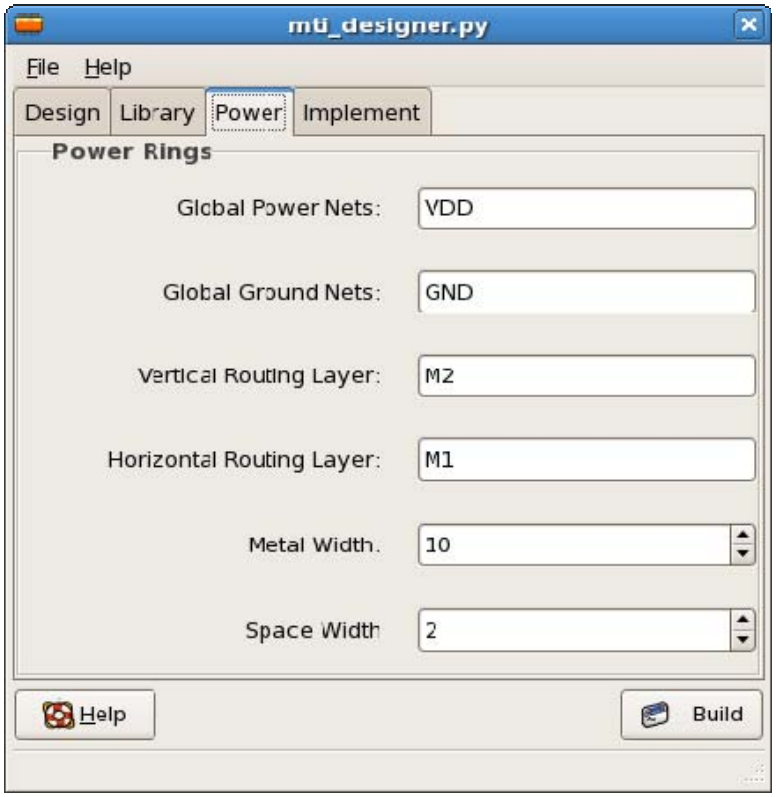

Figure 9: The power planning tab

Table 4: Description of power planning tab components

Interface Component	Description
Global Power Nets	Nets common to all digital cells that should be connected to power.
Global Ground Nets	Nets common to all digital cells that should be connected to ground.
Vertical Routing Layer	The name of the metal layer that should be used for vertical power/ground nets.
Horizontal Routing Layer	The name of the metal layer that should be used for horizontal power/ground nets.
Metal Width	The width of the metal in the power/ground rings.
Space Width	The width of the space between power/ground rings.

The implement pane guides the user through the build process of the serial controller. Green check marks are displayed after each action if it completed successfully. A red "x" is displayed if a step fails. If a step fails the designer may clock on the "View Log" button in order to view the error.

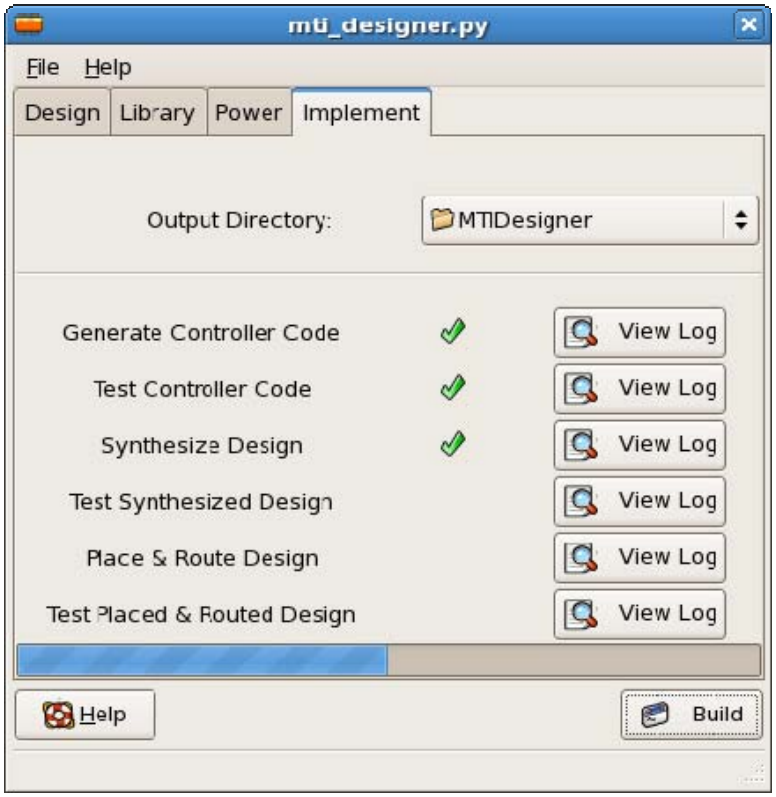

Figure 10: The design implementation tab.

Table 5: Description of implement tab components

Interface Component	Description
Output Directory	The directory that the generated layout should be placed in.
Generate Controller Code	Indicates whether the Verilog code for the serial controller has been generated or not.
Test Controller Code	Indicates whether the generated Verilog code has been tested against the test vectors or not.
Synthesize Design	Indicates if the design has been synthesized or not.
Test Synthesized Design	Indicates if the synthesized design has been tested or not.
Place & Route Design	Indicates whether the PnR tool has completed or not.
Test Placed & Routed Design	Indicates if the layout macro has been tested or not

Information gathered from the user interface is saved into an XML file to be read by other scripts in the MTI-Designer suite. Information stored in the XML file reflects information gathered by the user interface. A user can choose to save a session to the XML file before implementation is complete and resume from the saved XML file at a later point in time. An example of the XML format is given in Appendix C.

Section 5.2.2 Template Verilog

The template design consists of a set of Verilog files that contain special tags. These tags indicate to the automatic hardware description language (HDL) generator which sections of code need to be customized based on the user's input as gathered by the user interface. Currently there are three Verilog source files required to describe the digital architecture of the MTI DLM. Two of these files must be modified in order for the design to match user requirements. The third file contains basic building blocks for the MTI DLM such as shift-registers.

The first template code file contains a description of a ten-bit wide register file. The register file takes as input an instruction and address. Based on the instruction the register file either waits to receive data or provides the contents of its registers on the data bus. This file is modified by MTI-CAD in order to include the number of registers desired by the chip designer. In addition address decoders are inserted based on the depth of the register file.

The third piece of template code describes the serial controller. This piece of Verilog code will read the serial signal in from the SPI interface and create the appropriate instruction, data, and address signals to be sent to the register file. This file

must be modified so that address decoders may be added for each new register file entry the user requests.

Section 5.2.3 Code Generator

The automatic code generation for the HDL description of the MTI digital circuitry is performed in the Automatic HDL generator. The generator takes the user input from the user interface in the form of an XML file combined with the template Verilog code and creates a Verilog description of a specific MTI implementation. This Verilog description is then passed on to the MTI-Build component of the MTI-Designer Suite.

Section 5.3 MTI-Build Functional Block Description

MTI-Build is the portion of MTI-Designer that synthesizes the logical gates required for the digital portions of the user specified MTI implementation. It then proceeds to place the digital cells and route interconnects between them. At the end of the MTI-Build process the designer will be presented with a file containing all of the physical layout information for the MTI DLM. This layout can then be imported into the user's design tools to be combined with the user's mixed-signal circuitry.

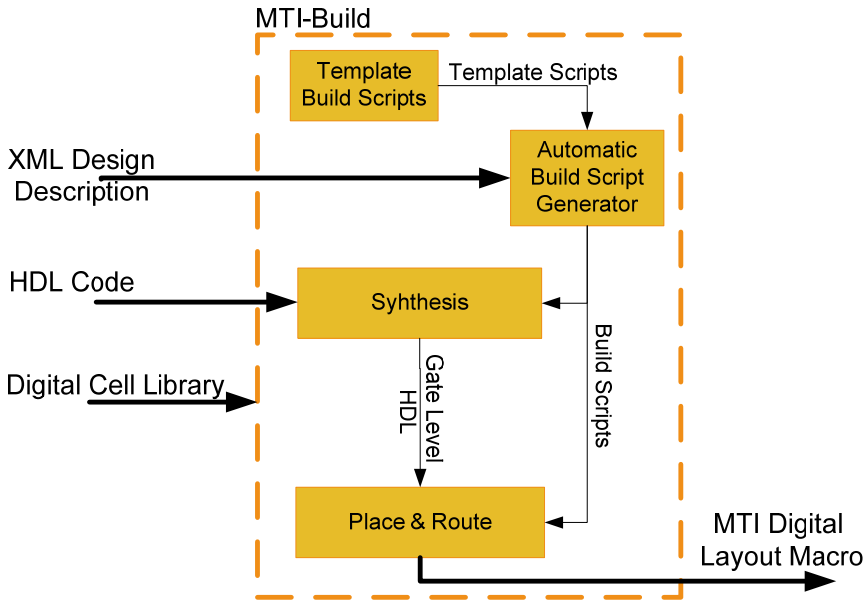

Figure 11: MTI-Build – Major Internal Blocks

Section 5.3.1 Automatic Build Script Generation

Automatic build script generation takes as inputs the XML description containing user requirements and process information along with template build scripts. Template build scripts must be filled in with process information. This information includes the location of timing libraries, Verilog cell descriptions, and physical descriptions of the cells.

Section 5.3.2 Synthesis

The synthesis script is responsible for calling the external tools that convert the behavioral Verilog description into a gate level Verilog description. This gate level Verilog description specifies which gates from the library will be used in the design and how they are connected to each other. The script used for synthesis is available in Appendix D.

Section 5.3.3 Place and Route

The PnR script is responsible for taking the gate-level Verilog from the synthesis step and converting it into a physical layout. This is accomplished using the Cadence tool SoC Encounter. The template PnR script can be found in Appendix E.

Section 5.4 Testing

Since the implementation of the DLMs is completely automated is important that testing be performed after each stage of implementation. Testing is accomplished through the use of a Verilog test-bench as seen in Appendix B. Figure 12 shows the results from part of the test-bench with a waveform viewer.

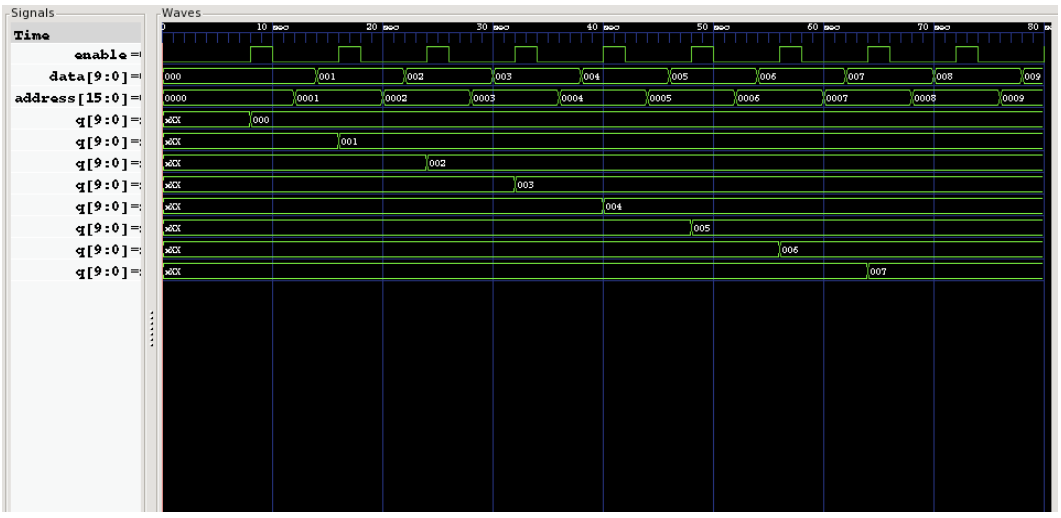

Figure 12: Test results for MTI Digital Layout Macro

Testing is performed three times during the process of implementing the MTI DLM. The first time the test-bench is run it is used to verify that the automated HDL generator generated working RTL code. After the RTL code has been synthesized into gate level logic the test-bench is run again. In order to ensure that the synthesized logic is meeting timing requirements delay information generated by the synthesis script is back annotated into the RTL so that simulation is performed with realistic gate delays. The final test pass occurs after the PnR script has successfully completed. In this run of the test-bench not only are realistic gate delays back annotated but accompanying wire delays are back annotated as well. If any of these test fail the MTI-Designer user interface will notify the user of the error. The template test bench can be viewed in Appendix B.

Section 6 Results

In order to verify the MTI the group designed and had manufactured a silicon implementation of the MTI. Manufacturing was performed in AMI Semiconductors' C5N 0.5-µm CMOS process. Digital portions of the MTI were constructed using the standard cell library provided in AMI Semiconductor's process design kit. Other members of the senior project group laid out the analog portions of the MTI.

The chip contained three separate implementations of the MTI DLM along with five DACs. The digital circuitry consisted of 10, 100 and 1000 bit controllers that were fully implemented by the MTI-Designer software. The DLM designs were selected over several orders of magnitude in order to test out the flexibility of the MTI-Designer software and the digital architecture.

Figure 13 shows a photograph taken of the implementation die after it was received back from manufacturing. In the image the 100-bit serial controller is highlighted. The 10-bit serial controller resides in the small rectangle to the right of the 100-bit controller and the 1000-bit controller dominates the center of the chip. All of the serial controllers were backfilled with metal, in order to meet metal layer usage requirements, so the majority of the circuitry in these modules is not visible.

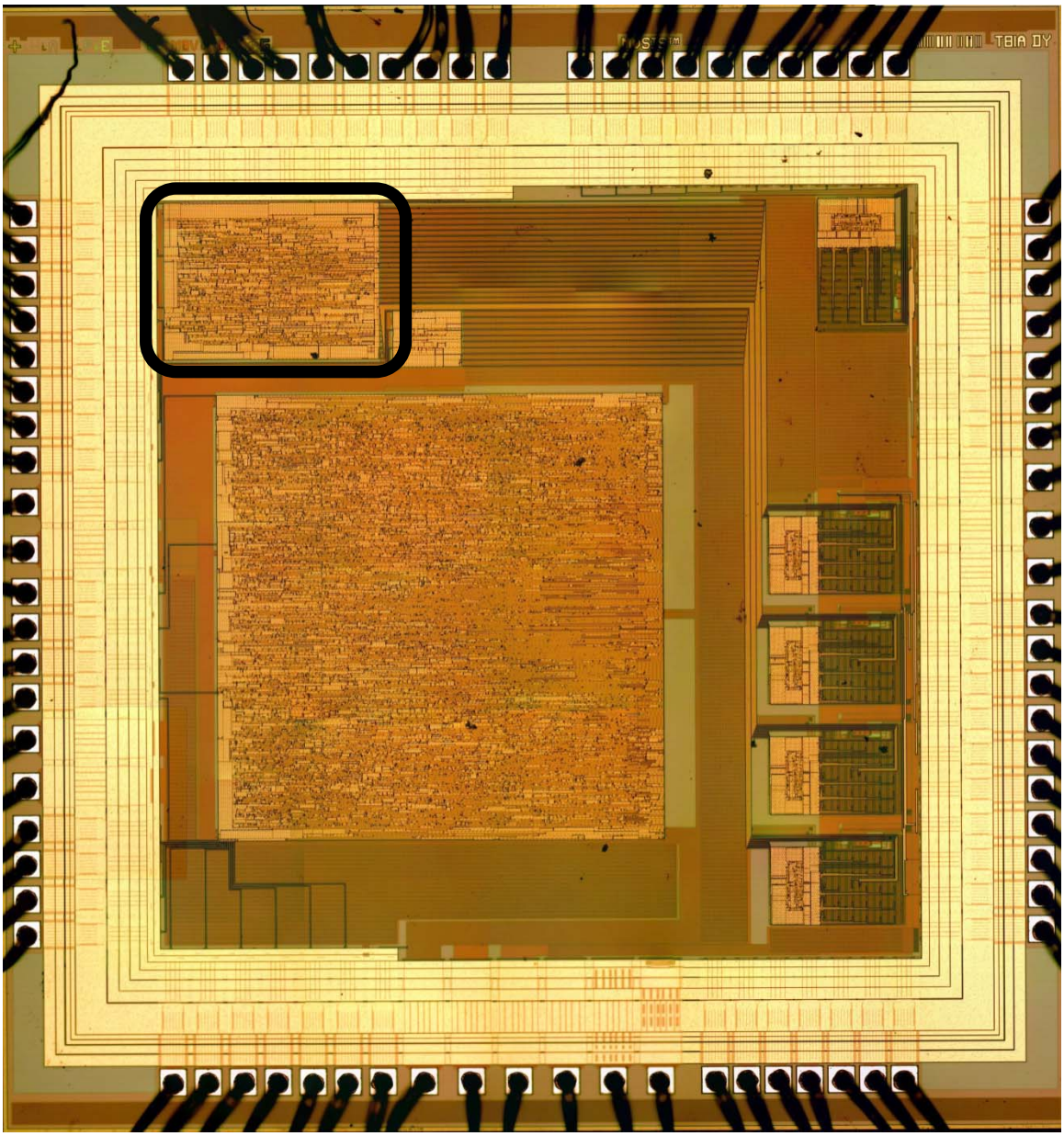

Figure 13: Image of die with 100-bit serial controller highlighted

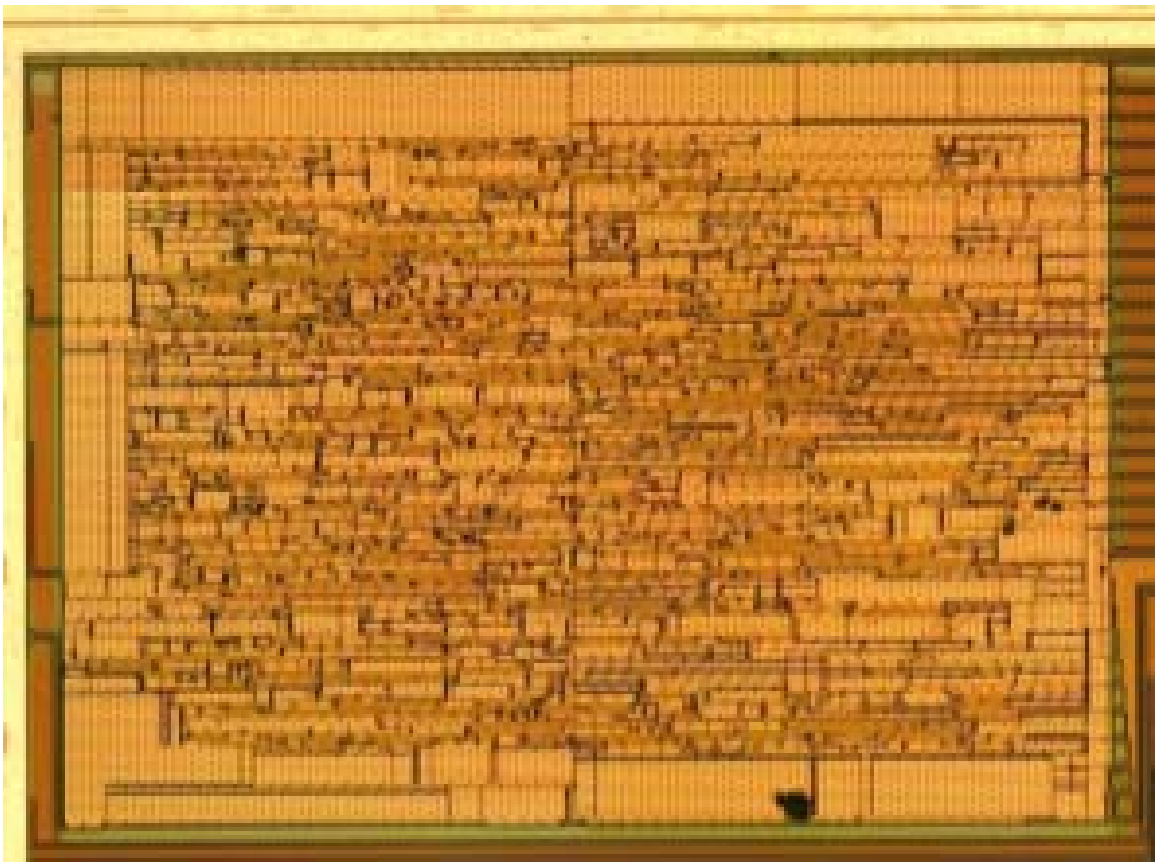

Figure 14: Close-up view of the 100-bit serial controller

Section 6.1 Testing Results

The digital circuitry used in the implementation chip was generated to operate at a frequency of up to 25MHz. Due to limitations in the equipment used to test the implementation the maximum rate that the serial controllers have been tested at is 6MHz. With each 10-bit wide register requiring 26-bits to program (14 address, 2 instruction, 10 data) the maximum frequency that the serial controller is able to change output is approximately 230KHz.

Initial tests confirmed that the all three serial controllers were operating as designed. A program was written to write data into each of the 100 10-bit entries in the 1000-bit serial controller and then to read out the data and verify its integrity. Similar

tests were performed on the 100 and 10 bit controllers. During these tests no malfunctions were observed.

In order to demonstrate the capabilities of the project as a whole the author wrote a program to translate waveform audio format (WAV) files into commands to be sent to the serial controllers. The output of one of the DACs was sent off-chip to an audio amplifier, which was then connected to a set of speakers. This allowed the demonstration of the entire MTI system operating at a sustained frequency of 48KHz with 10-bit DAC resolution.

Section 7 Future Work

At the conclusion of the project several areas were identified that could benefit from further research and future work [3]. Some of these future work items pertained to the MTI system as a whole, some to the analog design and some for MTI-Designer. The future work identified in this section refers specifically to the digital architecture of the MTI DLM and MTI-Designer.

Section 7.1 Improved Support for Custom Libraries

Currently if a digital cell library does not exist for a process it is up to the user to obtain or design a cell library. In future versions of MTI-Designer it is desirable that the tool be able to guide the user through the creation of their own custom cell library in the event that one is not accessible or does not exist for the user's process.

Section 7.2 Physical Verification in other Processes

Currently the MTI and MTI-Designer have only been verified in AMI's 0.5µm C5N process. MTI-Designer has been used to create demonstration circuits utilizing a handful of other process design kits including the (North Carolina State University) NCSU 45nm kit. None of these other processes have been physically verified though. In order to increase user confidence in MTI-Designer and the MTI system physical verification in other processes is desired.

Section 7.3 User Acceptance Testing

The MTI-Designer user interface, documentation, and design flow have, to date, only been utilized by members of the original senior design group. More thorough testing with users would undoubtedly discover ways in which the documentation and user

interface could be better designed to meet the needs of users. User testing would also reveal and design issues which impede adoption of the MTI.

Section 7.4 Automated Testbench Design Tool

In order to increase support for automation original design discussions for the MTI-Designer included a GUI from which to assemble and run automated test sequences for the MTI. This system was eventually scoped out of the requirements due to time constraints, but would have significantly increased the utility of the MTI. Such a system would allow the user to select stimuli to be applied to their circuit and the time intervals at which those signals should change.

Section 8 Conclusions

The MTI provides a method for test and calibration of mixed-signal circuits. It accomplishes the goals set forth of creating a process independent test harness that reduces pin count and aids test automation. Furthermore, the MTI has been demonstrated through its implementation and verification in AMI Semiconductors' C5N 0.5-µm CMOS process. The MTI consists of two primary components, its RTL behavioral description and the MTI-Designer tool. This paper has addressed the design methodology, design, implementation, and future work for the MTI-Designer tool and the architecture of the MTI DLM.

The DLM implements SPI in order to allow for off-chip communications. SPI was selected due to its low overhead and its relatively simple and ubiquitous design. The tools used in the creation of the MTI-Designer application were chosen from those available to the team. Synthesis is performed with Synopsys Design Compiler and PnR is accomplished with Cadence SoC Encounter. The user interface and accompanying scripts for MTI-Designer were created using a combination of the Python programming language, the GTK graphics toolkit, the TCL programming language, and makefiles.

The digital architecture of the MTI DLM consists of a 10-bit wide addressed register file accessed by a simple controller. Data can be read and written to any entry and all bits in an entry change simultaneously. The software for MTI-Designer presents the user with a simple but powerful GUI frontend. The backend software consists of a series of template files and automated scripts. These scripts fill in template files based on user input and orchestrate the process of creating the MTI DLM.

In order to prove operation and verify the architectural design of the MTI several serial controllers and accompanying DACs were implemented in AMI Semiconductors' C5N 0.5-μm CMOS process. Tests performed on the resulting chip revealed no architectural or implementation flaws.

Further testing is desired in other CMOS processes. To make the MTI more useful to users additional research needs to be done in how the MTI can be implemented in CMOS processes where the user does not have access to a standard cell library for the process. In addition the user interface for MTI-Designer needs to be subjected to user acceptance testing.

The project was a success on all counts producing fully operational silicon in what was for all members of the group their first chip. The MTI provides meaningful and user friendly solutions to problems that will continue to vex mixed-signal developers for the foreseeable future.

Section 9 References

[1] Awtrey, Dan, “Transmitting Data and Power over a One-Wire Bus,” Sensors, Feb. 1997.

[2] Bosch, CAN 2.0 Specification, Stuttgart, 1991.

[3] C. D. Edmonds, D. T. Keese, and R. J. Przybyla, “Mixed Signal Test Interface: Design Description.” Senior Project Description, Sch. of Elect. Engr. & Comp. Sci., Oregon State University, Corvallis, OR, 2001. Available online: http://classes.engr.oregonstate.edu/eecs/fall2007/ece441/2007/g26/files/MTI_Project_Specification_0.pdf.

[4] Ganssle, Noergaard, et. al, Embedded Hardware, Burlington, Elsevier, 2007, 301-387.

[1] IEEE 1149.1-2001, IEEE Standard Test Access Port and Boundary-Scan Architecture.

[2] IEEE 1149.4-1999, IEEE Standard for a Mixed-Signal Test Bus.

[3] IEEE 1394-1995, IEEE standard for a high performance serial bus.

[4] Peacock, Craig, “USB in a Nutshell,” Website, April 6th, 2007. Available: http://www.beyondlogic.org/usbnutshell/

[5] Philips I2C-Rev 03, IC-bus specification and user manual.

[6] Philips I2S-1996, I2S bus specification.

[7] Richie’s Paper

[8] "PyGTK: GTK+ for Python." PyGTK. Available online: http://www.pygtk.org/.

[9] “Selecting a Serial Bus,” Application Note, Maxim Semiconductor, Nov. 2006.

[10] Sunter, Stephen, “Implementing and Using a Mixed-Signal Test Bus,” 2004.

Appendices

Appendix A. Verilog Template

```
module serial_controller (sclk, cs, mosi, miso, out);
      input sclk, cs, mosi;
      output out, miso;

      parameter length = 60;

      reg [length-1:0] serial_data;
      reg [length-1:0] out;

      assign miso = serial_data[length-1];

      always @(posedge sclk)
      begin
            if(cs == 0)
            begin
                  serial_data <= {serial_data[length-2:0], mosi};
            end
      end

      always @(posedge cs)
            out <= serial_data;

endmodule
```

Appendix B. Verilog Test-bench

```
`timescale 1ns/1ns;

module tb ();

      parameter length = 60;
      parameter clk_period = 20;

      integer file, file_o, c, r, bit_count;
      integer reg_length = length;
      reg clk, cs;
      reg [31:0] i;
      reg [length-1:0] data;
      reg [length-1:0] temp;
      wire miso, mosi;
      wire [length-1:0] out;


      assign mosi = data[length-1];

      serial_controller sc(clk, cs, mosi, miso, out);

      initial
      begin : file_block
            cs=1;
            clk=0;
            data=0;
            file = $fopen("vectors/golden_vectors", "r");
            file_o = $fopen("vectors/vectors", "w");
            if (file == 0) //If error opening file
                  disable file_block;

            while (!$feof(file))
            begin
                  @(negedge cs)
                  begin
                        r = $fscanf(file, "%b\n", data);
                        temp = data;
                           #((length+1)*clk_period)


                        $fdisplay(file_o, "%10d\t %b %b", $stime,
temp, out);
                  end
            end
```

```
            $fclose(file);
            $fclose(file_o);

            $finish;
      end

      always @(negedge clk)
      begin
            data = data << 1;
      end

      always
      begin
            #(clk_period*2) cs = !cs;
            for (i=0; i<length*2; i=i+1)
            begin
                  #(clk_period / 2) clk = !clk; // Clock generator
            end
            #(clk_period/2) cs = !cs;
      end

endmodule
```

Appendix C. Example XML Description

```
<MixedSignalTestInterface>
<SerialController>
      <Outputs>60<Outputs/>
</SerialController>
<Controller>
      <North>
            SerialInterface
      </North>
      <South>
            outputs[0-19]
      </South>
      <East>
            outputs[20-29]
      </East>
      <West>
            outputs[30-39]
      </West>
</Controller>
</MixedSignalTestInterface>
<DigitalLibrary>
      <TlfFile>
            Mylib.tlf
      </TlfFile>
      <DbFile>
            Mylib.db
      </DbFile>
      <LefFile>
            MyProccess.lef
</LefFile>
      <LefFile>
            Mylib.lef
      </LefFile>
      <VerilogFile>
            Mylib.v
      </VerilogFile>
      <BufferCell>
            Bufx01
      </BufferCell>
      <InverterCell>
            Invx01
      </InverterCell>
</DigitalLibrary>
```

Appendix D. Synthesis Script

```
#/*************************************************/
#/* Compile Script for Synopsys                   */
#/*                                               */
#/* dc_shell-xg-t -f compile_dc.tcl               */
#/*                                               */
#/* Christopher Edmonds, OSU                      */
#/* edmondsc@onid.orst.edu                        */
#/*************************************************/

set hdlin_auto_save_templates true
set hdlin_check_no_latch      true
set hdlin_warn_sens_list      true

#/*************************************************/
#/* User Set Variables                            */
#/*************************************************/

#/* List of Verilog files                         */
set design_Verilog_files [list rtl_src/serial_controller.rtl.v]

#/* Top-level Module Name                         */
set design_toplevel serial_controller

#/* The name of the clock pin. If no clock-pin    */
#/* exists, pick anything                         */
set design_clock_pin sclk

#/* Target frequency in MHz for optimization      */
set design_clk_freq_MHz 50

#/* Delay of input signals                        */
set design_input_delay_ns 1

#/* Reserved time for output signals              */
set design_output_delay_ns 1

#/*************************************************/
#/* Shouldn't Need To Change the Rest             */
#/*************************************************/
set AMIS_Cells [format "%s%s" [getenv "AMIS"]
"/lib/amis500cx/logic/synopsys/amis500cxascm/current"]
set search_path [concat $search_path $AMIS_Cells]
set alib_library_analysis_path $AMIS_Cells
```

```
set target_library "amis500cxascm_wc_4.5v_150c.db"
set link_library "* $target_library"


set bad_cells [ get_lib_cells {amis500cxascm/*} ]
set good_cells [list "amis500cxascm/df001_m"
"amis500cxascm/no21_m" "amis500cxascm/inv1_m"]
foreach re $good_cells {
    set good_cell [get_lib_cells $re]
    query_object $good_cell
    query_object $bad_cells
    set bad_cells [remove_from_collection $bad_cells $good_cell]
}
set_dont_use $bad_cells


set Verilogout_show_unconnected_pins "true"
set_ultra_optimization true
set_ultra_optimization -force

analyze -f Verilog $design_Verilog_files

elaborate $design_toplevel

check_design

current_design $design_toplevel
link
uniquify

set design_period [expr 1000 / $design_clk_freq_MHz ]

set clk_name $design_clock_pin
create_clock -period $design_period $clk_name

set_input_delay $design_input_delay_ns -clock $clk_name
[remove_from_collection [all_inputs] $design_clock_pin]
set_output_delay $design_output_delay_ns -clock $clk_name
[all_outputs]

#compile -ungroup_all -map_effort medium
compile -map_effort medium
compile -incremental_mapping -map_effort medium
```

```
check_design
report_constraints -all_violators


#change_names -rules "MYrules" -hierarchy
change_names -rules Verilog -hierarchy


# Compile with completely dissolved design
compile -ungroup_all -map_effort medium
#
compile -incremental_mapping -map_effort medium

# Make sure we are at the top level
set current_design  serial_controller

# Generate area and constraints reports on the optimized design
report_area    > reports/serial_controller.area.rpt

# Generate timing report for worst case path
report_timing  > reports/serial_controller.delay.rpt

report_hierarchy > reports/serial_controller.hierarchy.rpt

report_resources > reports/serial_controller.resources.rpt

# Save the compiled design
write -format Verilog -hierarchy -output  [format "%s%s%s"
"rtl_src/" $design_toplevel ".gate.v"]

# Write out the delay information to the sdf file
write_sdf [format "%s%s%s" "sdfout/" $design_toplevel
".gate.sdf"]

quit
```

Appendix E. Place and Route Script

```
###################################
# Run the design through Encounter
###################################

# Setup design and create floorplan
loadConfig ./encounter.conf
commitConfig

# Create Floorplan
floorplan -r 1.0 .95 40.05 40.8 40.05 42

# Add supply rings around core
addRing -spacing_bottom 9.9 -width_left 9.9 -width_bottom 9.9 -
width_top 9.9 -spacing_top 9.9 -layer_bottom metal1 -width_right
9.9 -around core -center 1 -layer_top metal1 -spacing_right 9.9 -
spacing_left 9.9 -layer_right metal2 -layer_left metal2 -
offset_top 9.9 -offset_bottom 9.9 -offset_left 9.9 -offset_right
9.9 -nets { gnd vdd }

#placeInstance $custom_cell  342 343 R0 -fixed

# Place standard cfells
placeDesign

# Route power nets
sroute -noBlockPins -noPadRings

# Perform trial route and get initial timing results
trialroute
buildTimingGraph
setCteReport
reportTA -nworst  10 -net > timing.rep.1.placed

# Run in-place optimization
# to fix setup problems
setIPOMode -mediumEffort -fixDRC -addPortAsNeeded
initECO ./ipo1.txt
fixSetupViolation
endECO
buildTimingGraph
setCteReport
reportTA -nworst  10 -net > timing.rep.2.ipo1
```

```
# Run Clock Tree Synthesis
createClockTreeSpec -output encounter.cts -bufFootprint buf -
invFootprint inv
specifyClockTree -clkfile encounter.cts
ckSynthesis -rguide cts.rguide -report report.ctsrpt -macromodel
report.ctsmdl -fix_added_buffers

# Output Results of CTS
trialRoute -highEffort -guide cts.rguide
extractRC
reportClockTree -postRoute -localSkew -report
skew.post_troute_local.ctsrpt
reportClockTree -postRoute -report report.post_troute.ctsrpt

# Run Post-CTS Timing analysis
setAnalysisMode -setup -async -skew -autoDetectClockTree
buildTimingGraph
setCteReport
reportTA -nworst  10 -net > timing.rep.3.cts

# Perform post-CTS IPO
setIPOMode -highEffort -fixDrc -addPortAsNeeded -incrTrialRoute
-restruct -topomap
initECO ipo2.txt
setExtractRCMode -default -assumeMetFill
extractRC
fixSetupViolation -guide cts.rguide

# Fix all remaining violations
setExtractRCMode -detail -assumeMetFill
extractRC
if {[isDRVClean -maxTran -maxCap -maxFanout] != 1} {
fixDRCViolation -maxTran -maxCap -maxFanout
}

endECO
cleanupECO

# Run Post IPO-2 timing analysis
buildTimingGraph
setCteReport
reportTA -nworst  10 -net > timing.rep.4.ipo2

# Add filler cells
addFiller -cell FILL -prefix FILL -fillBoundary
```

```
# Connect all new cells to VDD/GND
globalNetConnect vdd -type tiehi
globalNetConnect vdd -type pgpin -pin vdd -override

globalNetConnect gnd -type tielo
globalNetConnect gnd -type pgpin -pin gnd -override

# Run global Routing
globalDetailRoute

# Get final timing results
setExtractRCMode -detail -noReduce
extractRC
buildTimingGraph
setCteReport
reportTA -nworst  10 -net > timing.rep.5.final

# Output GDSII
streamOut final.gds -mapFile
/import/vlsi1/osu_stdcells_2.4/flow/ami05.stacks/gds2_encounter.m
ap -stripes 1 -units 1000 -mode ALL
saveNetlist -excludeLeafCell final.v

# Output DSPF RC Data
rcout -spf final.dspf

# Run DRC and Connection checks
verifyGeometry
verifyConnectivity -type all -noAntenna

exit
```