A MIPS R2000 IMPLEMENTATION

Nathaniel Pinckney, Thomas Barr, Michael Dayringer, Matthew McKnett, Nan Jiang, Carl Nygaard, Joel Stanley, David Money Harris, and Braden Phillips
Harvey Mudd College, Claremont, CA 91711, USA
The University of Adelaide, SA 5005, Australia
Email: npinckney@hmc.edu, tbarr@hmc.edu, mdayringer@hmc.edu, mmcknett@hmc.edu, njiang@hmc.edu, cnygaard@hmc.edu, joel.stanley@adelaide.edu.au, David_Harris@hmc.edu, phillips@eleceng.adelaide.edu.au

ABSTRACT

Thirty-four undergraduates implemented a MIPS R2000 processor for an introductory CMOS VLSI design course. This included designing a microarchitecture in Verilog, developing custom PLA generation and ad-hoc random testing tools, creating a standard cell library, schematics, layout, and PCB test board. The processor was fabricated by MOSIS on an AMI 0.5-micron process, included 160,000 transistors, and ran at 7.25 MHz.

KEYWORDS: MIPS, RISC.

1 Introduction

MIPS is a family of 32-bit and 64-bit computer processors used for many embedded applications, including network routers, PDAs, and game consoles such as the Sony PlayStation Portable. A MIPS processor is classified as a Reduced Instruction Set Computer (RISC) processor, because of its small number of instructions and addressing modes, in contrast to Complex Instruction Set Computers (CISC), such as the Intel x86 architecture. RISC favors less complexity to streamline hardware implementation, relying on software optimization. Examples of previous RISC processors include ARM, DEC Alpha, and MIPS. Modern Intel and AMD x86 processors are RISC-like, implementing RISC execution units and using microcode to execute CISC instructions.

MIPS was originally invented as part of a Stanford research project [1] and later brought to market by newly-started MIPS Corporation in 1985, releasing the MIPS R2000 running at 8 MHz on 2.0 micron process. In 1988 the R3000 was released, improving performance to eventually 40 MHz on a 1.2 micron process. Both used approximately 110,000 transistors and included cache controllers, which could use external memory chips as processor cache. The R2000 supported 32 kB of data cache and 64 kB of instruction cache. The R3000 doubled the amount of data cache. Later revisions of the chip, including the R4000, expanded the chip to 64-bit instructions and processing [2].

As part of E158: Introduction to CMOS VLSI, thirty undergraduates at Harvey Mudd College, advised by Professor David Money Harris, and four students at the University of Adelaide advised by Professor Braden Phillips, designed, fabricated and tested an R2000-compatible MIPS processor in a semester. Unlike the original R2000/R3000, the data and instruction caches are on-chip. The project included a cell library, logic design, schematics, custom layout, a compiler chain, a test board, and custom tools. This project provided hands-on VLSI experience and exposed students to working in a large design team on a nontrivial problem.

2 MIPS Microarchitecture

At the beginning of the semester, four students quickly designed and implemented logic for the microarchitecture, so that the other students could begin work on schematics and layout. The RTL was coded in Verilog and simulated in Modelsim. More detailed information about the microarchitecture and downloadable code is available on the hmc-mips Google Code website [3] and is released as open source under the MIT license agreement.

The MIPS architecture includes thirty-two general-purpose 32-bit registers and fifty-eight instructions, each 32 bits long. Some R2000 processors have external floating-point units (FPUs). Our design did not include support for an FPU.

Figure 1 shows a high-level block diagram of the processor. The instructions are processed in a five-stage pipeline: fetch, decode, execute, memory, and writeback. Instructions are read from the instruction cache during the fetch stage, from the memory address stored in the program counter (PC). During the decode stage, data is read from the triple-ported register file and the controller configures how the instruction will be manipulated in each stage, by starting a state machine. Jumping or branching may also occur in the decode stage. In the execute stage arithmetic operations are performed and values are shifted. Additionally, reads and write from the dedicated multiply/divide unit are performed during this stage, as will be discussed later. Data cache reads and writes are performed during the memory stage. Finally, the writeback stage writes values to the register file.

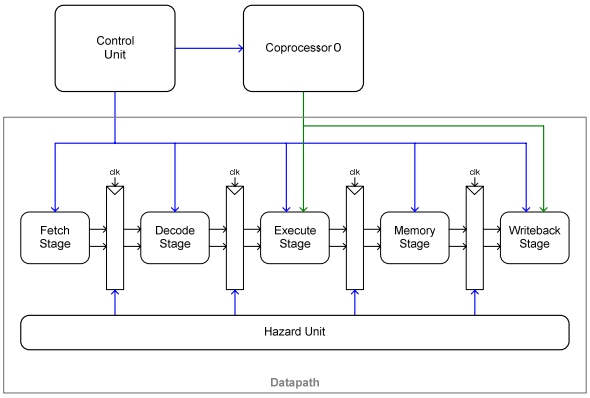

Figure 1: Block Diagram of MIPS Processor

Because jumps are processed during the decode stage, the next instruction has already been read from memory by the fetch stage. Instead of flushing the pipeline, the instruction is processed. This is known as the "branch delay slot" and must be

considered by the compiler or assembly programmer. A hazard detection unit in the controller detects when data is unavailable, because of a cache miss or unprocessed instruction, and stalls the CPU accordingly.

A coprocessor handles exceptions and holds configuration bits. Only a few configuration bits are used for the R2000 architecture, and are mostly used to enable/disable exceptions and to configure the caches. The MIPS architecture supports exception handling and interrupts using so-called "precise exceptions." This means that when an exception occurs, a single instruction is labeled at-fault. Every instruction before the offending instruction completes in its entirety and anything after (as well as the offending instruction itself) has no effect. From an exception handling standpoint, the architecture can be viewed as being a sequential, non-pipelined design.

To ensure that this happens, the processor only handles exceptions when the offending instruction is in the execute stage. Exceptions detected in the decode stage (such as the break instruction) are delayed until the execute stage. This allows any previous instructions to finish. The address of the offending instruction and the type of exception is stored in the coprocessor, and execution redirected to a hard-coded memory-address, where an exception handler resides. The chip supports a subset of the R2000 exceptions. The exceptions Breakpoint, Syscall, Invalid Opcode, and FPU Unavailable are all detected in the instruction decode stage. Misaligned Load, Misaligned Store, and Arithmetic Overflow are all detected in the execute stage. Interrupts are treated like any other exception handled in the execute stage, only they come from an external input pin.

Figure 2 shows the on-chip memory system. Unlike the original R2000 implementations, which had no on-chip caches, we have two separate 512 byte on-chip caches for instructions and data. The small size of these caches is due to constrained space on the die. The cache is write-through, where the cache is never more up-to-date than memory. By writing to a memory location, the corresponding cache line is invalidated. Normally, the processor would have to stall and wait for the external memory to complete a write before continuing execution. Instead a four-entry write buffer is used to store words for writing to memory. If the memory location is later read before the write-buffer has finished, a cache miss will occur and the CPU will stall until all writes have occurred. Data cache can read and write, but the instruction cache is read-only. There is a shared external memory bus for instruction and data, and since a data cache miss in the memory stage will stall the fetch stage, the data cache is given precedence for memory access after the write buffer. The physical caches can be swapped, so the instruction cache becomes the data cache and vice-versa. This is helpful during cache initialization. Data is cached when the memory address is between `0x8000 0000` to `0x9FFF FFFF` and uncached when the memory address is between `0xA000 0000` to `0xBFFF FFFF`.

Our MIPS implementation also includes a dedicated multiply/divide (multdiv) unit capable of multiplication and division on signed and unsigned integers. It uses a radix-4 Booth algorithm to multiply numbers and a successive shift-and-subtract algorithm to divide them. Since this can take up to 32 cycles, it is not desirable to stall the CPU during this operation. Instead the result is stored in two dedicated registers, prodh and prodl, representing the high and low 32-bits of a multiplication. A multiply or divide instruction will load the multdiv unit with the proper input values and start it, however execution will continue with the next instruction. The CPU only stalls for multdiv completion when prodh/prodl is read. A carefully written program can start a multiply, continue doing useful work, and only read the result when the computation is complete.

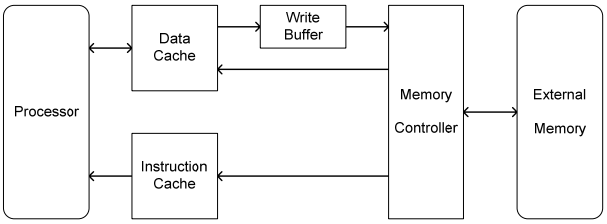

Figure 2: Block Diagram of On-Chip Memory System

When two numbers are divided, the integer part of the quotient is stored in prodl and the remainder is stored in prodh. In signed division, the unit first computes the quotient and remainder of the magnitude of the inputs, then adjusts the result. The quotient is negated if the signs of the divisor and dividend disagree, and the sign of the remainder is set to the sign of the dividend.

3 Schematics and Layout

The schematics and layout were done in Electric, an open source VLSI CAD program developed by Steven Rubin of Sun Microsystems. Electric is written in Java, and so is available for Linux, Mac, and Windows operating systems. Electric provides verification tools, including design rule check, electrical rule check, and schematic/layout netlist check. It has auto-route, auto-stitch, and mimic-stitch features to speed repetitive layouts, such as connecting a datapath.

We developed a programmable logic array (PLA) ROM generator for the controller. A PLA structure is an AND plane and an OR plane, used to compute sum of products functions. The PLAs generated by our software are read-only. The PLA generator reads a Verilog case statement and outputs a corresponding Electric library file, including a schematic, layout, and symbol for the PLA. The tool is written in Java and is freely released on the MIPS project website [4].

To aid in layout, the team developed a standard cell library before starting on the processor layout. The cell library contains 257 cells of 57 types, such as a 2-input NAND or settable flip-flop with enable. Most types come in a variety of transistor sizes, and some are optimized for use in the controller or datapath, by following different convention. For example, the width of cells in the datapath can be reduced by accepting complemented inputs, such as clock for the flip-flops, instead of including an inverter within the cell. Because the controller is mostly random logic, controller cells do not include complemented inputs. Figure 3 shows the schematic and layout for a datapath flip-flop, with complimented “ph1” and “ph2” clock inputs. In the layout, green is a diffusion layer, pink is polysilicon, and blue and purple are metal layers.

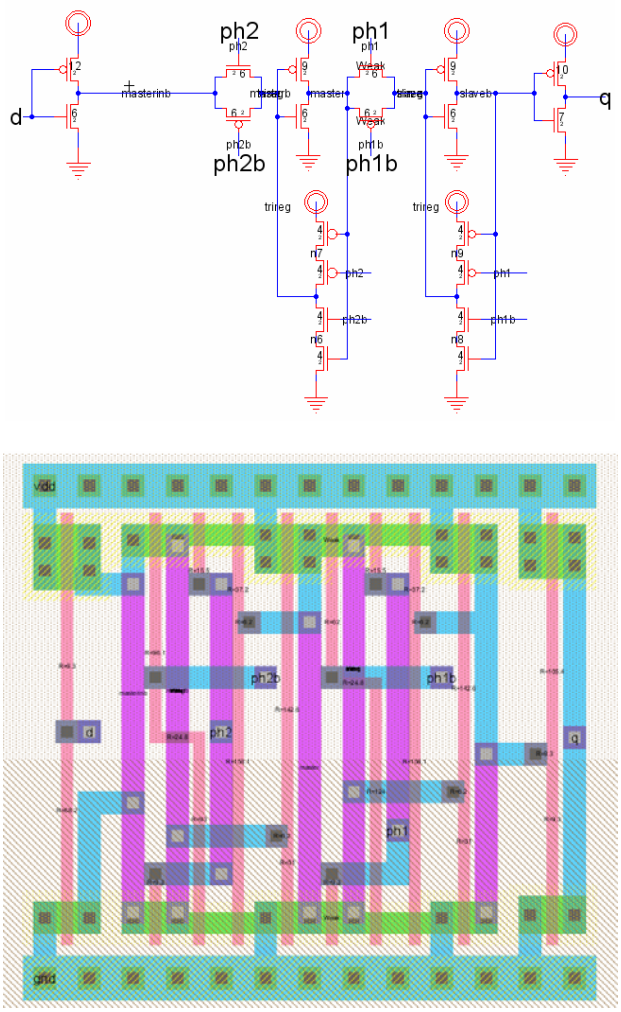

Figure 3: Schematic and Layout of Datapath Flip-flop

A preliminary floorplan was created immediately after the RTL was complete. Our die was 4.5mm x 4.5mm in a 0.5-micron process, but we designed for a 4mm x 4mm die size to leave padding for unanticipated routing. This tested the feasibility of the design, and provided insight on size constraints. Parts of the RTL were updated based on the preliminary floorplan, such as the cache size reducing from 2 kB to 512 bytes per cache.

The class was split up into unit teams, of about five students each, responsible for different modules, such as datapath, controller, and cache system. Within each team, blocks of the unit were assigned to different team members. Each member was responsible for creating a schematic from the RTL, calculating a floorplan for their block, and designing a layout. Unit managers collected floorplans for individual blocks and assembled them into detailed unit floorplans. Finally, the unit floorplans were assembled into a detailed floorplan shown in Figure 4. Layouts were assembled analogously. A 2-phase non-overlapping clock was used to eliminate hold time risks. The finished layout contains 160,000 transistors.

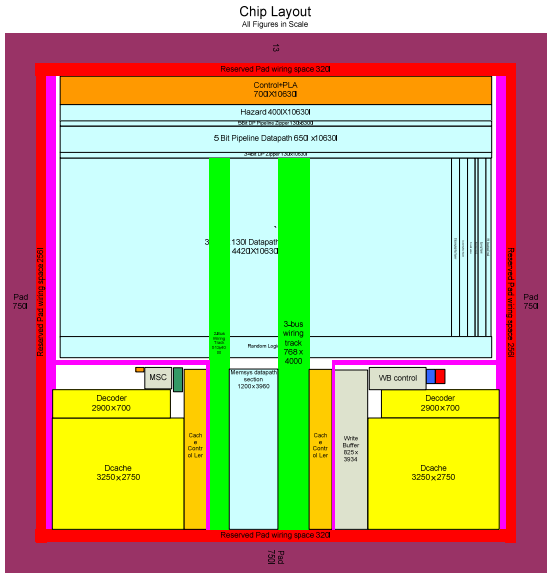

Figure 4: MIPS Layout Floorplan

4 Verification

To aid in the incremental development of the processor, a set of ad-hoc tests were written along with the RTL. These tests consisted of several assembly language test programs that targeted specific MIPS instructions. A Verilog testbench runs each program through the simulated processor, allowing flaws to be easily traced to a specific subsystem. When possible, these tests were also run through SPIM, a free MIPS simulator, to ensure compatible behavior with existing MIPS implementations. This was not possible to test non-standard exception conditions and interrupts, as these features are not implemented in SPIM.

To verify the proper functionality of the processor in unexpected and corner cases, two random directed test generators were developed, one for the multiply/divide unit and one for the entire CPU.

Due to the algorithmic complexity of the signed radix-4 Booth multiply/divide unit, we decided to test the device as thoroughly as possible by using a random testvector generator. This generator picked input vectors from a set of known corner cases (negative and positive one, zero and the maximum and minimum signed numbers), and generated the expected output from standard integer multiplication and division. This testbench allowed us to identify a bug in the multdiv unit wherein the sign of the remainder of negative quotients was miscalculated. This bug resulted from a misinterpretation of a signed integer division specification. It was caught because the testvector generator used an independently developed reference implementation of signed integer division which highlighted the inconsistency.

The multdiv testbench generator was used as the basis for a much larger random code generator. Since the ad-hoc tests tested fewer than five hundred unique instruction words, the team created a directed random assembly generator as a black-box test. This allowed us to test a much larger number of possible combinations of instructions likely to appear in production code. The test creates assembly, with a mix of ALU, multiply/divide, and memory load/store operations divided into

blocks that are conditionally executed using the different branch types. The relative frequency of each instruction appearing can be controlled to isolate specific subunits of the CPU. This tester was not designed to explore the behavior of the CPU during exceptions, so the generated code is created to avoid possible overflows or divide by zeros. Branches are also constrained to jump forward to avoid infinite loops. The code generator is written in Python and is also freely available on the project website [4].

As the generator creates a block of code, it simulates the code to determine the resultant state of the CPU after execution. This uses a software implementation of the MIPS instruction set developed from published documentation to serve as an independent reference implementation. An arbitrary number of instructions may be generated and written to an assembly source file. The generated code terminates by XOR'ing the contents of the registers together. The final result of that XOR is predicted deterministically by the code generator, and verification is achieved by comparing the predicted output with the actual output, thus indicating consistency with the reference design.

Electric's design rule checker (DRC) and schematic/layout netlist checker (NCC) were used to verify the layout matched the schematic and did not have any geometrical errors. We generated a transistor-level Verilog deck for the layout using Electric and simulated it in Modelsim against the RTL. The team ran hundreds of thousands of randomly generated instructions through the RTL, and five thousand through the transistor-level schematic. Transistor-level simulation was very time consuming, with the simulated chip running at about 5 Hz, or 10 minutes to complete 3000 tests. In all runs, the resultant state of the simulated design was consistent with the calculated result from the code generator. The large number of instructions executed minimized the probability of a wiring mistake on the chip or an improperly handled pipeline hazard, and the independently developed reference implementation minimized the probability of a specification error.

Individual modules requiring more precise transistor sizing and timings, such as the SRAM bits for the caches, were simulated individually in IRSIM. Additionally, a PLA was simulated in SPICE to verify the PLA generator's correctness. The Adelaide team worked on verifying the final layout against the RTL using IRSIM. IRSIM is a switch level simulator, treating each transistor as a switch, series resistor, and parallel capacitor to ground. The RC circuit is a simplified model of transistor delay, known as the Elmore delay model of transistor delay.

The ad-hoc tests were run against the RTL to generate a VCD, which describes the inputs and outputs that were asserted by the tests over time. This VCD was then parsed with a custom Perl script, changing the VCD format into an IRSIM command file. During debugging the chip initially failed testing due to compiled C code tests jumping to random memory locations instead of entering an endless loop. This put the simulation into an unknown state, because the memory locations were not initialized, and caused future to tests to experience errors.

The tests were run and the chip passed simulation at a maximum clock frequency of 40 MHz, though the critical path was not determined. This maximum frequency is a very rough estimate because of the simplified transistor model.

5 System Tools

Because we did not have a MIPS computer to test the chip with after fabrication, five students composed the systems team, which was responsible for designing a test system, creating a toolchain to compile code, and writing small test programs.

A Xilinx FPGA development board was chosen to emulate memory, provide memory-mapped I/O devices, and generate a 2-phase clock signal. Table I shows the memory map of the test setup. Our test memory only maps the lower 17 bits to a physical address. Hence, the upper 15 bits are ignored by the external memory system. During reset the processor fetches the instruction at address `0xBFC0 0000`, by convention, which maps to the physical address `0x000 0000`. The upper three bits of the address are used to bypass the cache in the chip, as mentioned in the previous section.

TABLE I
MEMORY MAP OF TEST SETUP

Memory Range	Description	
`0x000 0000`	Reset Vector	Instruction ROM
`0x000 0004`	Exception Vector	
`0x000 0100` to `0x000 01FC`	Boot loader	
`0x001 0200` to `0x001 6A7C`	Program memory	
`0x001 6A80` to `0x004 3FFC`	Data RAM	
`0x004 4000`	LED Array	I/O Devices
`0x004 4004` to `0x004 4010`	DIP Switch	
`0x004 4014` to `0x004 4024`	Push buttons	
`0x004 4028`	LCD Display	

The compiler toolchain we used was a modified version of GNU GCC 4.1.1, compiled by Professor James Stine at University of Oklahoma. A small library of C routines was developed to use the toolchain. Functions include setting LEDs, reading switches, and sending data to the LCD text display. A small boot loader was also created to initialize the caches and stack pointer, and jump to the start of a program. We did not implement any standard C libraries, such as stdlib.h or math.h.

The system team also designed a PCB layout to mount the MIPS processor, provide regulated power, and interface with an FPGA development board. The PCB and FPGA were tested by wiring the chip socket to an additional FPGA, programmed with synthesized RTL of the processor. Test programs were run on this dual-FPGA setup and found to work.

6 Fabrication and Testing

Tapeout was done in Electric, which generated a CIF file appropriate for fabrication. The chip was fabricated by MOSIS using an AMI 0.5 micron process, with a die size of 4.5 mm x 4.5 mm, and packaged in a 108-pin pin grid array (PGA), measuring 1.2 inches on a side. Figure 5 shows a photomicrograph of the fabricated chip.

A photograph of our test setup, described in the last section, is shown in Figure 6. The custom PCB on the left includes a 2x20 LCD for text display and a reset button. After no shorts between power and ground were found, and we were satisfied that the test setup was correct, the chip was placed in the socket on the test board.

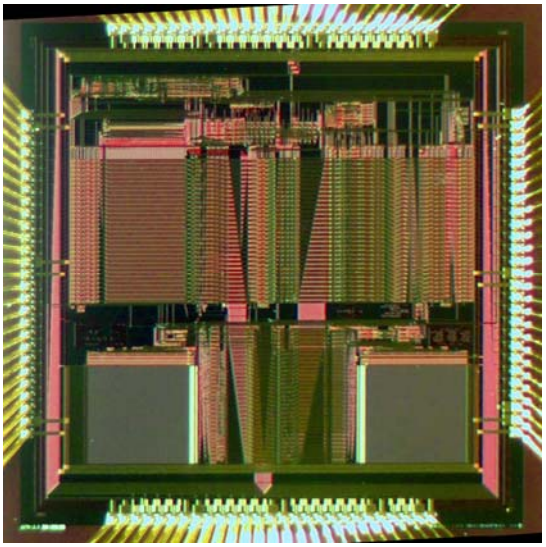

Figure 5: Photomicrograph of Fabricated MIPS Chip

The chip functioned perfectly on first silicon, though some errors were found in the test setup. It passed all of the pre-tapeout regression tests, successfully controlled LEDs, and played a simple "lights out" game written in C.

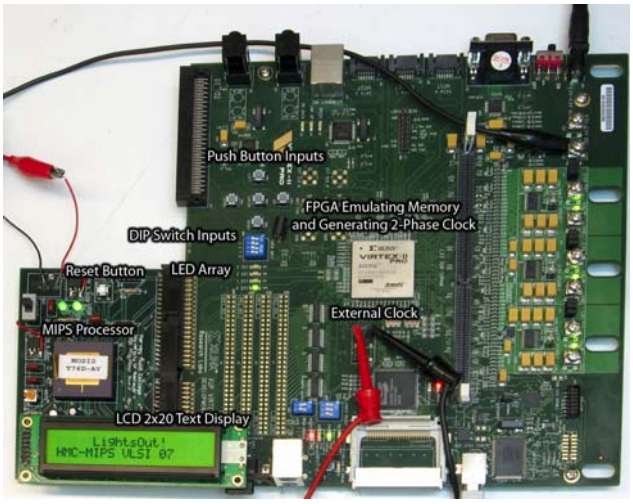

Figure 6: Test Setup with FPGA Board

We were able to run the processor at a maximum of 7.25 MHz for the 2-phase clock, which we found to be limited by the cache. Above 7.25 MHz the chip manifested erratic sporadic behavior, such as displaying random characters on the LCD during the "lights out" game, and eventually froze. Though the simulated chip was able to operate at a higher clock speed, we believe these simulations were flawed because of simplified transistor models. With the cache disabled, we were able to run some tests at 15 MHz. At 7.25 MHz the power draw was 52 milliwatts.

In order to measure the performance of the chip against other similar MIPS processors, we used the Dhrystone benchmark program. It runs a series of routines that test the string handling and integer arithmetic facilities of a processor. The results of the benchmark are reported in Dhrystones per second and normalized to the score of the VAX 11/780, a 1 million instruction per second machine. Here we report the VAX score (microseconds per Dhrystone cycle divided by 1,757) and the DMIPS/MHz (VAX score divided by clock speed). Though the Dhrystone benchmark is generally considered to be an inaccurate measure of system performance, we believe it is useful for measuring our chip's rough performance against other R2000/R3000 chips. Furthermore, the Dhrystone is an integer-only benchmark while many other benchmarking programs rely on a floating point unit, which our chip does not have.

We were forced to modify the Dhyrstone source code because our test setup does not provide time information to the processor, and so it is impossible to measure the time Dhrystone takes to run. We also modified the source code because of a limitation of our compiler toolchain, so that all of the functions called by `main()` were moved into the second of Dhyrstone's two source code files. This bends one of the rules of the Dhrystone benchmark [5] meant to make the compilation process more "real-world." We do not believe that this minor point affects our results. The Dhrystone benchmark was compiled using GCC 4.1.1 with default optimizations.

Unlike a standard Dhrystone benchmark, we had to manually time the modified benchmark using a stopwatch. We used twenty-five thousand runs of the Dhrystone main loop, which took about 13 seconds to complete. The timing error introduced by human reaction time (roughly ±0.05 s) is considered to be the greatest source of timing error and constitutes a ±0.001 error in the reported VAX score and ±0.0001 DMIPS/MHz error. Benchmark results of the processor compared to other R2000 processors [6] is shown in Table II.

Our fabricated chip achieved a VAX score of 1.1 DMIPS at 7.25 MHz and 0.15 DMIPS/MHz, about six times slower than other R2000 processors. Prior to chip fabrication we measured the performance of our processor in the dual-FPGA emulated test setup and found the DMIPS/Mhz to be identical to the fabricated chip.

The Dhrystone benchmark also helped to validate functionality of the chip. The program reports success or failure of Dhyrstone calculations on the LCD and LED array. No incorrect calculations were reported by Dhrystone program during testing of the fabricated chip at 7.25 MHz.

TABLE II
COMPARISON OF DESIGN WITH SIMILAR R2000 PROCESSORS

Processor Name	Vax Score (DMIPS)	Clock Speed (MHz)	DMIPS /MHz
HMC MIPS (cache disabled)	0.36	7.25	0.050
HMC MIPS	1.08	7.25	0.150
DECstation 2100 R2000	11.193	12	0.93275
SGI Personal Iris 4D/20 R2000	9.812	12.5	0.78496
SGI Personal Iris 4D/20 R2000	9.799	12.5	0.78392

7 Conclusions

A team of 34 undergraduates designed, fabricated, and tested a 32-bit MIPS processor compatible with the original R2000. The design was completed as part of a semester-long CMOS VLSI course. The layout includes 160,000 transistors on a 4.5 mm x 4.5 mm 0.5-micron die. The chip was fabricated, tested, and found to operate at a maximum 2-phase clock frequency of 7.25 MHz. The finished chip was able to validate against the RTL, pass all benchmarks, and run a small "lights out" game.

ACKNOWLEDGEMENT

The authors would like to thank the developers of Electric, especially Steven Rubin, for producing a great CAD software package. Sun Microsystems for supporting Electric development. MOSIS for subsidizing chip fabrication. Professor James Stine at University of Oklahoma for his modified version of GCC. Lastly, the authors thank all of the students who participated in the project, at Harvey Mudd College and University of Adelaide.

REFERENCES

[1] M. Horowitz, et. al., "MIPS-X: a 20-MIPS peak, 32-bit microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 22, no. 5, pp. 790-799, Oct. 1987.

[2] D. Sweetman, *See MIPS Run*, San Diego: Academic Press, 2002.

[3] "Google Code hmc-mips," http://code.google.com/p/hmc-mips/

[4] "E158 CMOS VLSI Design Spring 2007 MIPS Project," http://www4.hmc.edu:8001/Engineering/158/07/project/

[5] A. Weiss, "Dhrystone Benchmark: History, Analysis, `Scores,' and Recommendations White Paper," Nov. 2002, http://www.synchromeshcomputing.com/pdf/dhrystoneWhitePaper.pdf

[6] D. Grevenstein, "Dhyrstone Benchmark Results," http://sites.inka.de/pcde/dbp/dhrystone.html

Nathaniel Pinckney is a 2006-2008 Clay-Wolkin fellow and senior engineering major at Harvey Mudd College specializing in VLSI design and embedded system. He served as the memory microarchitect for the MIPS project. He is currently researching cryptographic hardware accelerators and plans to pursue a Ph.D. in electrical engineering after graduation.

Thomas Barr is a senior at Harvey Mudd College, and a Clay-Wolkin fellow for 2007-2008. He works for The Aerospace Corporation in High-Performance Computing research, and intends to pursue a Ph.D. after graduation. He was the Exception Microarchitect for the MIPS project.

Michael Dayringer is a senior engineer major at Harvey Mudd College. His interests are in VLSI and computer architecture and he intends to pursue a master's degree after graduation. He was part of the memory team for the MIPS project.

Matt McKnett is a senior computer science major at Harvey Mudd College. He is focusing on Computer Graphics and User Interface Design and will go into industry after graduation. He was co-manager of the systems team for the MIPS project, in charge of compiler toolchain setup, and developing the demo and benchmarking programs.

Nan Jiang graduated Harvey Mudd College in 2007 and is now a Ph.D. student at Stanford University. His research interests are in computer architecture and VLSI. During the MIPS project he served as Chief Circuit Designer.

Carl Nygaard graduated from Harvey Mudd in 2007 in Computer Science and now works in Google's Kirkland office as a Software Engineer in Test. He served as the Chief Microarchitect.

Joel Stanley is a computer systems Bachelor of Engineer and Bachelor of Economics final year student, from the University of Adelaide, South Australia. Since then, he has been working for the One Laptop per Child project, both as an intern at their office on MIT's campus, and as a Google Summer of Code participant in Australia. He worked with the Adelaide team, in collaboration with HMC, on the cache subsystem and verification of the final chip layout before tapeout.

David Money Harris is an Associate Professor of Engineering at Harvey Mudd College. David received his Ph.D. from Stanford University in 1999 and his S.B. and M. Eng. degrees from MIT in 1994. His research interests include high speed CMOS VLSI design and computer arithmetic. He is the author of *CMOS VLSI Design: A Circuits and Systems Perspective*, *Logical Effort*, and *Skew-Tolerant Circuit Design*. He holds twelve patents, has written numerous papers, and has designed chips at Sun Microsystems, Intel, Hewlett-Packard, and Evans & Sutherland. When he is not teaching or building chips, David enjoys hiking with his family.

Braden Phillips is a lecturer in the School of Electrical and Electronic Engineering at the University of Adelaide. Prior to the completion of his PhD thesis, `An Optimised Implementation of Public Key Cryptography for Smart Card Processors', Braden worked as a process control engineer and was a founding partner in Current Dynamics, an electronic hardware design venture. In September 2000 he took up a lecturing position at Cardiff University in South Wales, a post he held for 2 years before returning to Adelaide. Braden's research interests include digital arithmetic, digital microelectronics, computer architecture, real time systems and information security.