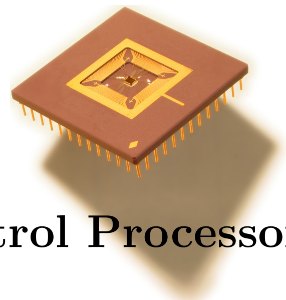
Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2007-003 January 11, 2007

Scale Control Processor Test-Chip

Chrstopher Batten, Ronny Krashinsky and Krste Asanovic

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu



Scale Control Processor Test-Chip

Christopher Batten, Ronny Krashinsky, and Krste Asanović

Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, 32-G736
Cambridge, MA 02139

January 5, 2007

We are investigating vector-thread architectures which provide competitive performance and efficiency across a broad class of application domains [1, 4]. Vector-thread architectures unify data-level, thread-level, and instruction-level parallelism, providing new ways of parallelizing codes that are difficult to vectorize or that incur excessive synchronization costs when multithreaded. To illustrate these ideas we have developed the Scale processor, which is an example of a vector-thread architecture designed for low-power and high-performance embedded systems. The prototype includes a single-issue 32-bit RISC control processor, a vector-thread unit which supports up to 128 virtual processor threads and can execute up to 16 instructions per cycle, and a 32 KB shared primary cache.

Since the Scale vector-thread processor is a large and complex design (especially for an academic project), we first designed and fabricated the Scale Test Chip (STC1). STC1 includes a simplified version of the Scale control processor, 8 KB of RAM, a host interface, and a custom clock generator. STC1 helped mitigate the risk involved in fabricating the full Scale chip in several ways. First, we were able to establish and test our CAD toolflow. Our toolflow included several custom tools which had not previously been used in any tapeouts. Second, we were able to better characterize our target package and process. For example, STC1 enabled us to better correlate the static timing numbers from our CAD tools with actual silicon and also to characterize the expected rise/fall times of our external signal pins. Finally, STC1 allowed us to test our custom clock generator. We used our experiences with STC1 to help us implement the Scale vector-thread processor. Scale was taped out on October 15, 2006 and it is currently being fabricated through MOSIS. This report discusses the fabrication of STC1 and presents power and performance results.

1 Overview

Figure 1 shows a high-level block diagram of STC1. The scalar processor (SP) is a single issue in-order processor which implements a subset of the MIPS-II ISA. The SP is fully interlocked and does not include programmer-visible branch delay slots or load-use delay slots. To interface with the memory system, the SP is divided into two decoupled units: the Fetch Unit and the Execute Unit. The Fetch Unit is responsible for generating instruction addresses and then issuing instruction fetch requests to the memory system. The Execute Unit is responsible for executing the instructions. For load and store instructions, the Execute Unit handles issuing the appropriate memory request and then writing back load data when it returns.

In the full Scale processor many requesters arbitrate for a single shared cache. The cache interface is fully decoupled: a requester issues a memory request and then some number of cycles later a memory response is returned to the requester. Therefore, in STC1 the decoupled Fetch and Execute Units are designed to continue operating even if one of the units is blocked from making a request. For example, if the Fetch Unit cannot issue an instruction fetch due to a conflict with another requester, the Execute Unit can continue to execute arithmetic instructions and even issue new load or store requests. Similarly, if the Execute Unit is blocked from issuing a memory request, the Fetch Unit can continue to run ahead and fetch additional instructions. In STC1, there are only three requesters and these requesters directly access a single bank of on-chip memory instead of accessing the Scale cache. The STC1 memory controller uses a fixed priority scheme to arbitrate between these three requesters.

The Fetch Unit consists of a single pipeline stage, while the Execute Unit consists of four pipeline stages: decode, execute, memory, and writeback. Note that the SP makes use of a two write port register file so that arithmetic instructions write the register file in the execute stage while load instructions write the register file in the writeback stage. The SP manages instruction dependencies to bypass or stall as needed.

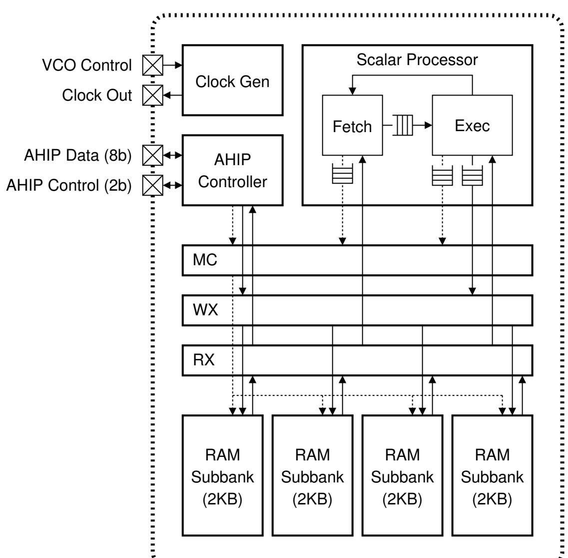


Figure 1: Scale Test-Chip (STC1) Block Diagram. Address buses are shown with dashed lines, while data buses are shown with solid lines. STC1 includes a scalar processor (SP), four 2 KB RAM subbanks, memory controller (MC), read crossbar (RX), write crossbar (WX), Asynchronous Host Interface Protocol (AHIP) controller, and a custom clock generator block. The SP is in turn composed of two decoupled units: a Fetch Unit responsible for fetching instructions from the on-chip RAM, and an Execute Unit responsible for actually executing the instructions.

The Fetch Unit always speculates that branches are not-taken so that it can continue to fetch instructions until the Execute Unit actually resolves the branch. When the Execute Unit resolves a branch as taken, it sends the new target PC and a special *branch token* to the Fetch Unit. The Fetch Unit then starts fetching from the correct target and in addition, it keeps the branch token with the correct branch target instruction. Since the Fetch Unit has already speculatively fetched instructions down the wrong path (and these instructions can be waiting in the various decoupling queues), the Execute Unit must discard instructions until it sees the branch token. Once the Execute Unit has seen the branch token it can start executing instructions normally. This simple token scheme enables the two units to be completely decoupled; the processor will function correctly regardless of how many cycles it takes for memory requests to return.

STC1 includes a host controller which allows the host to read and write the on-chip memory as well as several configuration registers. The host uses the Asynchronous Host Interface Protocol (AHIP) to communicate with STC1. AHIP is a simple read/write protocol; the on-chip memory and configuration registers are all memory mapped into the AHIP address space. The full Scale chip will use a similar controller and AHIP protocol. The approach for running a test program on STC1 involves the following steps.

1. STC1 includes a `block_cpu` signal from the AHIP controller to the memory controller, and on reset this signal is asserted. If `block_cpu` is asserted, then the memory controller refuses to grant any requests from the SP.

2. After resetting STC1, the host writes the program and any static data into the on-chip RAM using the AHIP protocol. When finished, the host writes a memory-mapped configuration register which deasserts the `block_cpu` signal. The scalar processor is now free to begin execution starting at the appropriate reset vector.

3. The host polls a specific `tohost` configuration register using the AHIP protocol. When STC1 is finished executing it writes this `tohost` register.

4. When the host sees the finish value in the `tohost` register, the host can read results from the on-chip memory using AHIP to verify proper functionality.

5. The host can now reset STC1 and load additional programs for further testing.

STC1 uses a custom clock generator which provides a 50% duty-cycle clock at an adjustable frequency up to 1 GHz with good supply voltage noise rejection. For testing purposes, a divided-by-16 clock is also generated and output off-chip. The clock generator includes a differential voltage-controlled oscillator (VCO), a differential to single-ended clock converter, and various clock-division logic. The VCO is similar to the one proposed by Hwang and Kang except that it has been adjusted for our process and frequency requirements [3]. The custom clock generator should enable us to experiment with much higher frequencies than what would be possible with an off-chip clock generator.

2 Implementation and Verification

A C++ functional model served as the reference implementation for STC1. This functional model is linked with a C++ test harness to enable loading test programs, running the tests, and reading the results. After implementing the STC1 RTL in Verilog, we used Tenison VTOC to translate the Verilog into C++. We were then able to link the RTL model with the same C++ test harness used for the functional model. We verified correctness by comparing the results from running the same tests on both the functional and RTL models. In addition to a suite of custom directed tests, we developed Torture, a random test program generator. The challenge in generating random tests is to create legal programs that run correctly and also stress different corner cases in the design. Torture randomly generates relatively simple instruction sequences of various types, and then randomly interleaves these sequences to construct complex yet correct programs. By tuning parameters which control the breakdown of instruction sequence types, we can stress different aspects of STC1.

Much of STC1 was implemented using standard cells and SRAM blocks provided by Artisan. To achieve a high-performance and efficient processor, we made extensive use of fine-grain standard-cell pre-placement. We wrote several custom tools and gate-level builders to pre-place individual standard cells into datapath bitslices and register file arrays. The control logic was synthesized using Synopsys DesignCompiler and placed & routed using Cadence Encounter. After integrating pre-placed blocks with automatically synthesized and placed & routed blocks, the final gate-level Verilog RTL was formally verified against the reference Verilog RTL using Synopsys Formality.

To verify the final GDSII layout for the chip, we ran DRC and LVS using Mentor Calibre. A final check of correct functionality was provided by running tests on the extracted Spice netlist using Synopsys Nanosim. VCD signal traces generated by the reference RTL model were used to drive the input pins of the chip and verify the output pins.

The STC1 chip was fabricated in the TSMC CLO18 process. It is composed of a mixture of standard cells, memories, and I/O's provided by Artisan, and our own custom clock generator module. STC1 has 14 digital I/O pads, 1 analog input pad, and 25 power and ground pads in a linear wire-bonding configuration. It uses a PGA121M package for pin-compatibility with previous test chips. The die size is 1.7 mm $\times$ 2.1 mm, and the core size is 1.0 mm $\times$ 1.4 mm. See Figure 2 for the chip plot and die photo.

3 Physical Test Setup

The physical testing and characterization of STC1 focused on measuring the power consumption across a range of supply voltages and clock frequencies. The infrastructure to test STC1 has been used to test several other custom VLSI chips, including the ATC1 test-chip which was also fabricated through the MOSIS Educational Research Program [2]. The test setup includes three primary components: a host computer, a general ATB0 test baseboard, and a daughter card (see Figure 3). The ATB0 test baseboard and the daughter card are custom boards designed for use in our research group. The daughter card includes a ZIF socket for the STC1 chip, and it connects the pins of STC1 to the ATB0 test baseboard. In addition, the daughter card provides probe points suitable for a logic analyzer or oscilloscope. One

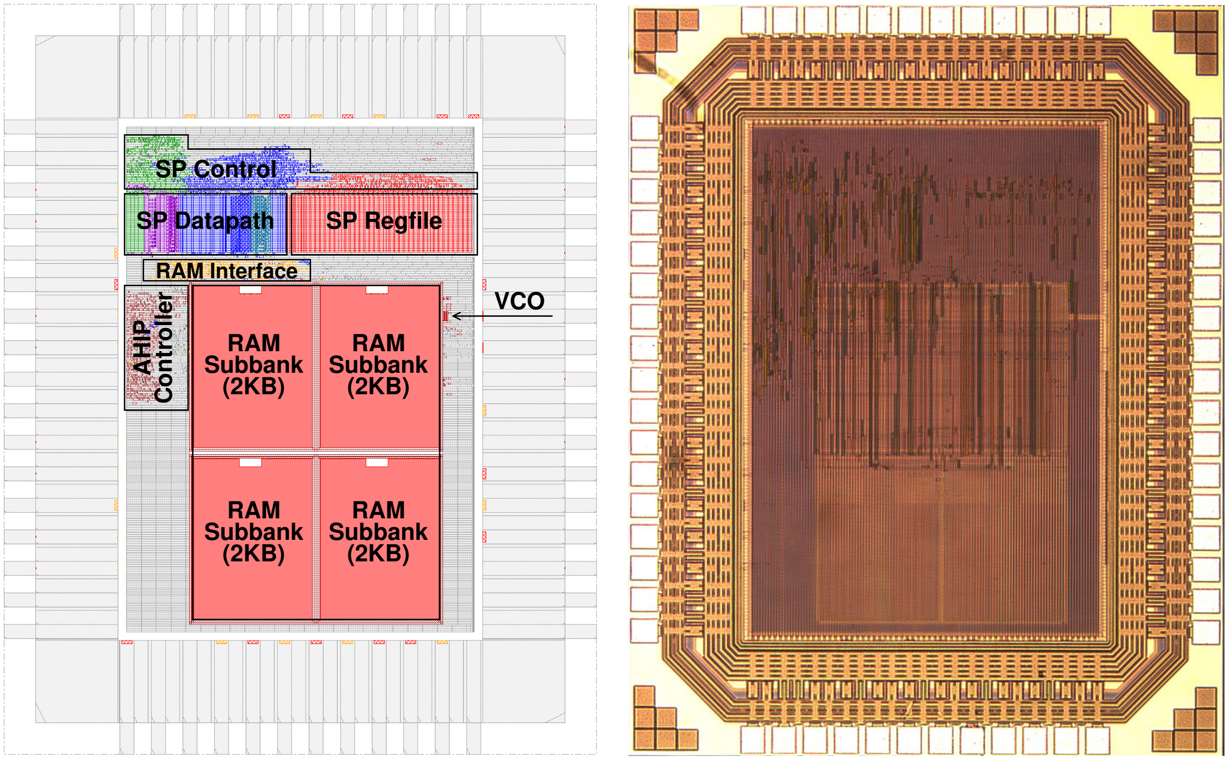


Figure 2: STC1 chip plot and die photo. On the chip plot, the Exec Unit is colored blue, and the Fetch Unit is colored green and purple.

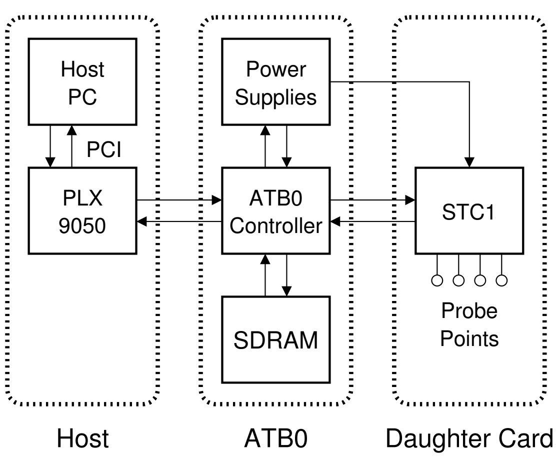


Figure 3: Test System Block Diagram. The test system includes the ATB0 test baseboard, a daughter card with the STC1 chip, and a host computer to control the test system.

of these probe points is connected to the clock output of STC1 enabling us to measure the performance of the custom on-chip clock generator.

The ATB0 test baseboard includes an ATB0 controller, 96 MB of SDRAM, and several adjustable power supplies with current measurement. The ATB0 controller is implemented in a Xilinx FPGA and it is able to communicate directly with STC1 using the AHIP protocol. Additionally, the ATB0 controller can adjust the voltage of the STC1 power supplies and the analog input of the voltage-controlled oscillator on STC1. The ATB0 controller is also able to read the current being drawn by each power supply. Using this system, we were able to accurately measure the power consumption of STC1 for various voltages and frequencies.

Software on the host PC communicates with the ATB0 controller using a PLX 9050 PCI card. A command protocol allows the software to adjust the power supplies, read data from the SDRAM, and communicate with STC1. More information about our test setup can be found online at `http://www.cag.csail.mit.edu/scale/hardware`.

4 Results

After receiving parts back from MOSIS, we were able to quickly plug the packaged chips into our test system and start running programs. For our basic evaluation, we used our test suite of 47 custom directed test programs which each primarily test a single instruction type. We also wrote tests to validate all of the SRAM memory bits. Out of 20 packaged parts 17 were functional, a yield of 85%.

Figure 4 shows a shmoo plot of frequency and supply voltage for all of the working parts. We observed a narrow range of variation between the chips. The performance difference between the fastest and slowest chip is around 10–20 MHz, less than 5%. At the nominal supply voltage of 1.8 V, all of the chips run at 440 MHz and 6 chips run at 450 MHz. This is around 15% faster than we expected from static timing analysis using the typical process parameter set.

Figure 5 shows results for a wider range of supply voltages. Scaling the voltage down to 1.1 V, the chips continue to operate at around 200 MHz. We were somewhat surprised that the frequency continued to improve at extremely high supply voltages up to 700 MHz at 3.6 V. However, the rate of frequency improvement did taper off at the higher voltages.

```
500    . . . . . . . . 2
490    . . . . . . . 1 9
480    . . . . . . 2 9 *
470    . . . . . . 9 * *
460    . . . . . 6 * * *
450    . . . . 4 9 * * *
440    . . . 1 * * * * *
430    . . . 9 * * * * *
420    . . 6 * * * * * *
410    . 2 * * * * * * *
400    . 9 * * * * * * *

       1 1 1 1 1 1 1 1 2
       6 6 7 7 8 8 9 9 0
       0 5 0 5 0 5 0 5 0
       0 0 0 0 0 0 0 0 0
```

Figure 4: Shmoo plot for all 17 working parts. The horizontal axis plots supply voltage in mV, and the vertical axis plots frequency in MHz. A * indicates that all of the chips functioned correctly at that operating point, and a dot indicates that none of the chips functioned correctly. A number indicates the percentage of chips which functioned correctly (e.g. 4 means 40%).

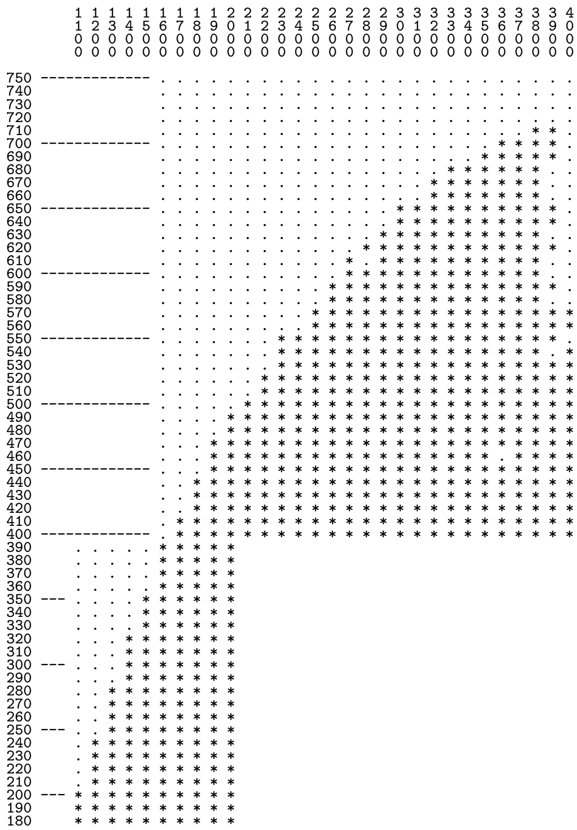


Figure 5: Shmoo plot for a wide voltage range. The horizontal axis plots supply voltage in mV, and the vertical axis plots frequency in MHz. A * indicates correct operation at that operating point and a dot indicates failure. The shmoo plot shows combined results from two different chips.

```
    la t9, input_end
forever:
    la t8, input_start
    la t7, output
loop:
    lw   t0, 0(t8)
    addu t8, 4
    bgez t0, pos
    subu t0, $0, t0
pos:
    sw   t0, 0(t7)
    addu t7, 4
    bne  t8, t9, loop
    b    forever
```

Figure 6: Test program for measuring typical power consumption. The inner loop calculates the absolute values of integers in an input array and stores the results to an output array. The input array had 10 integers during testing, and the program repeatedly runs this inner loop forever.

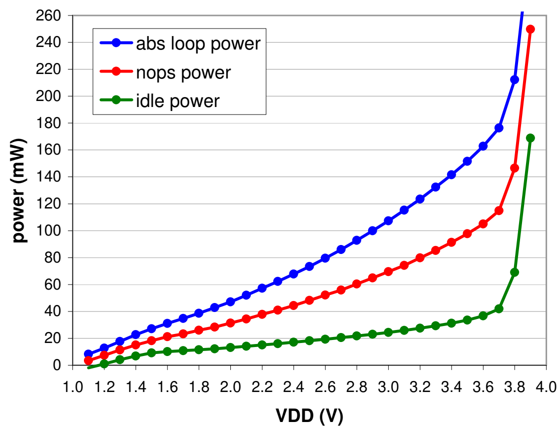


Figure 7: Power consumption versus supply voltage at 100 MHz. Results are shown for idle (clock only), no-operation (nop) instructions, and the absolute-value test program.

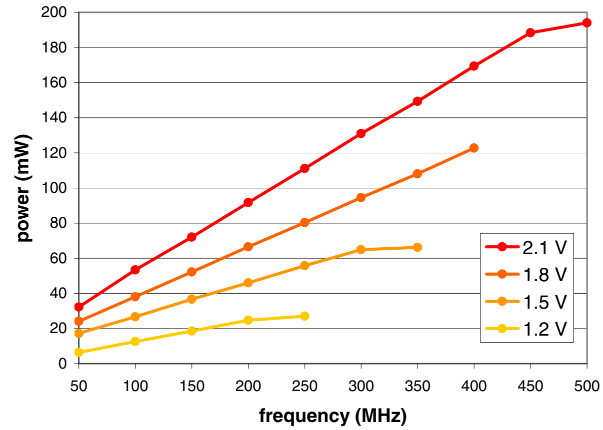


Figure 8: Power versus frequency for various supply voltages. The results are typical power consumption as measured while executing the test program.

We also measured power consumption using our testing infrastructure. We did this by measuring steady-state average current draw during operation. The power measurements are for the core of the chip and do not include the input and output pads. We measured the idle power draw when the clock is running but the processor is not fetching or executing any instructions. We also measured the power consumption when the processor is executing only no-operation (nop) instructions. To estimate typical power consumption, we used the simple test program shown in Figure 6.

Figure 7 shows how the power consumption scales with supply voltage with the chip running at 100 MHz. At 1.8 V, the idle power is 11.5 mW, the power while executing nops is 26.0 mW, and the typical power is 38.6 mW. The power scaling is roughly linear around the nominal supply voltage, but becomes nonlinear at higher voltages.

Figure 8 shows how the power consumption scales with frequency for various supply voltages. The scaling is mostly linear for a given supply voltage. At 1.8 V the power increases from 38 mW to 123 mW as the frequency scales from 100 MHz to 400 MHz. At a given frequency, the supply voltage can be lowered to reduce power. At 250 MHz the chip consumes 80 mW at 1.8 V but only 27 mW at 1.2 V.

As another way to evaluate the chip power consumption, Figure 9 shows how energy scales with frequency for a wide range of supply voltages. To calculate the average energy per cycle, we multiply the average power consumption by the clock period. This is a measurement of the energy to perform a fixed amount of work (one clock cycle), so it does not tend to change very much with frequency as long as the supply voltage remains fixed. However, we can reduce energy consumption at a given frequency by reducing the supply voltage as much as possible. At low clock frequencies the chip can run at a low supply voltage, but as the speed increases the supply voltage must also increase. Looking at the outer edge of the data points in Figure 9, we see that the minimum energy per clock cycle increases as the clock frequency increases. Another way to interpret this is that the energy used to execute a given program will increase as the program executes faster. The rate of energy increase is moderate at lower frequencies, for example doubling the speed from 240 MHz to

480 MHz increases the energy consumption by about 3×. But the energy increase becomes more drastic at higher frequencies and voltages.

To characterize the package performance, we used a oscilloscope to measure the rise/fall time of STC1 driving the digital output pad corresponding to the AHIP acknowledge signal (see Figure 10). With a rise time of 5.55 ns and a fall time of 6.37 ns we can reasonably assume that STC1 can switch its digital outputs at 50 MHz. We also measured the rise/fall time of the ATB0 controller driving the AHIP request signal which is connected to a STC1 digital input. We found this signal to have a rise time of 4.72 ns and a fall time of 3.64 ns. This data helped us set reasonable off-chip I/O expectations for the Scale processor when packaged in the same PGA121M package and placed in our standard ATB0 testboard. We also intend to later support a full memory system for Scale using a better package and a custom circuit board.

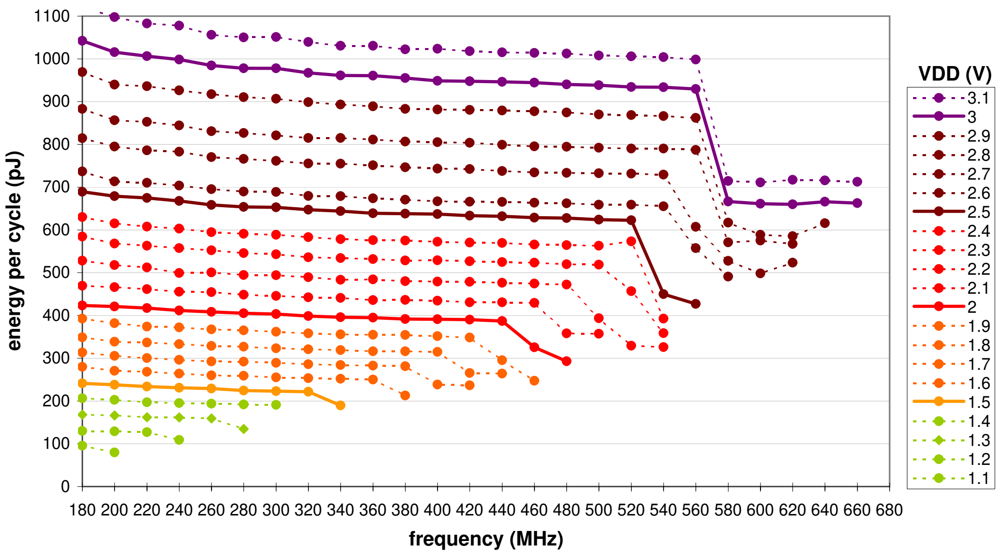
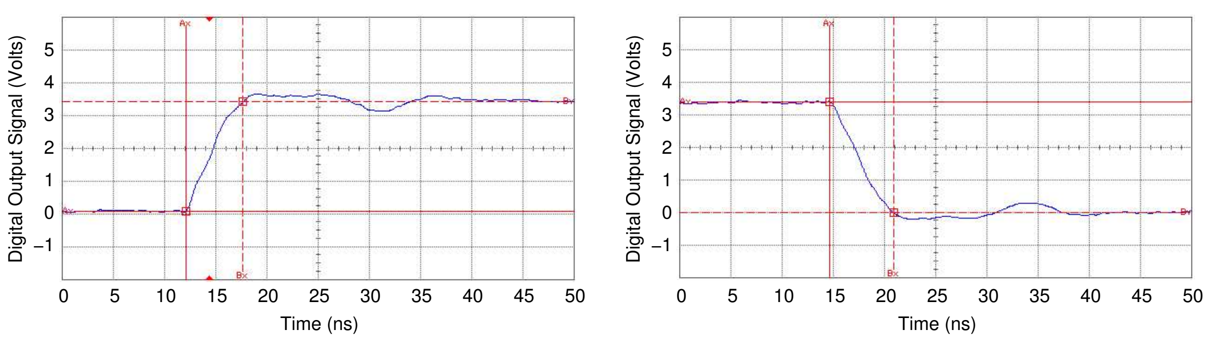


Figure 9: Energy versus frequency for various supply voltages. The results are typical energy per cycle as measured while executing the test program.



(a) Rise time is measured at 5.55ns

(b) Fall time is measured at 6.27ns

Figure 10: Full rail rise/fall time of STC1 digital output signal.

5 Acknowledgments

This chip fabrication was made possible by the MOSIS Educational Program. This work was partially supported by NSF CAREER award CCR-0093354. We acknowledge and thank Albert Ma for designing the VCO and providing extensive CAD tool support. Jaime Quinonez developed the initial datapath tiler used for fine-grain standard-cell pre-placement. We also thank Tenison for the use of the VTOC Verilog-to-C++ translation tool.

References

[1] C. Batten, R. Krashinsky, S. Gerding, and K. Asanović. Cache refill/access decoupling for vector machines. In *37th International Symposium on Microarchitecture (MICRO)*, Dec 2004.

[2] S. Heo and K. Asanović. Testchip for AHIP protocol, ASIC flow, and leakage control through body biasing. http://www.cag.csail.mit.edu/scale/hardware/atc1/atc1-mosisreport.pdf, April 2005.

[3] I.-C. Hwang and S.-M. Kang. A self-regulating VCO with supply sensitivity of <0.15%-Delay/1%-Supply. In *IEEE International Solid-State Circuits Conference (ISSCC)*, pages 140–141, February 2002.

[4] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, and K. Asanović. The vector-thread architecture. In *31st International Symposium on Computer Architecture (ISCA)*, June 2004.