FPGA Implementation and Validation of the Asynchronous Array of simple Processors

Jeremy W. Webb
VLSI Computation Laboratory
Department of ECE
University of California, Davis
One Shields Avenue
Davis, CA 95616
jwwebb@ece.ucdavis.edu

ABSTRACT
In today's ASIC market validating a VLSI design in a field programmable gate array (FPGA) device before tape out can save a significant amount of time and therefore, money up front in the design process. At the University of California, Davis, we are working on the design of a highly parallel, reconfigurable processor chip, known as the Asynchronous Array of simple Processors (AsAP). This report will summarize my design approach and results in both implementing and validating the AsAP processor in a Xilinx Spartan 3 FPGA.

Categories and Subject Descriptors
B.7.1 [**ASIC**]: Features – *VLSI (very large scale integration), Microprocessors and microcomputers.*

General Terms
Performance, Design.

Keywords
Reconfigurable, array of processors, energy-efficient.

1. INTRODUCTION
The VLSI Computation Laboratory (VCL) at UC Davis is currently working on the design of a highly parallel, reconfigurable processor chip, referred to as AsAP. The AsAP processor design will eventually be fabricated into an ASIC chip. However, before the AsAP is fabricated in silicon a more cost-effective solution will be tried; the same design will be implemented in an FPGA in order to prove its feasibility. The fabricated AsAP chip will contain hundreds of independent processors that can be reconfigured to a chosen algorithm or application. An appropriate example application is a 16-tap FIR filter.[1] The first FPGA implementation will be of a single AsAP processor interacting with outside data sources.

2. AsAP DESIGN FLOW
The AsAP design flow consists of two main paths: simulation and synthesis. The following two sections provide a description of the steps involved in designing with the AsAP processor.

2.1 Simulation Design Flow
The AsAP simulation design flow (see Figure 1) consists of the following steps:

1. Write AsAP Verilog HDL code.
2. Write AsAP assembly code for each processor.
3. Write AsAP configuration file.
   - Define the configuration settings for each processor. For example, clock frequency and memory contents.
4. Write AsAP chip file.
   - Define X by Y processor array size of chip and input/output direction for each processor.
5. Write AsAP input data file for simulation stimulus.
6. Generate imem.dat and cmem.dat files.
7. Simulate AsAP processor using either NC-Verilog or ModelSim.
   - Verify functionality of AsAP processor and the assembly code.

The AsAP simulation design flow is simplified by the use of make files and Perl scripts to automate the intensive steps of the design process.

2.2 Synthesis Design Flow
The AsAP synthesis design flow (see Figure 1) consists of the preceding simulation steps, with the following:

1. Synthesize design using either Synopsys' Design Compiler, Synplicity's Synplify Pro, or Xilinx' XST Synthesis tools.
   - Define clock and pinout constraints.
2. Translate, map, and place & route AsAP processor using Xilinx ISE 6.3i design tools.
   - Define timing ignore, timing groups, and any device specific constraints.
3. Generate programming bit file.

The AsAP synthesis design flow is simplified by the use of Windows batch files, Tcl scripts, and Perl scripts to automate the intensive steps of the design process.

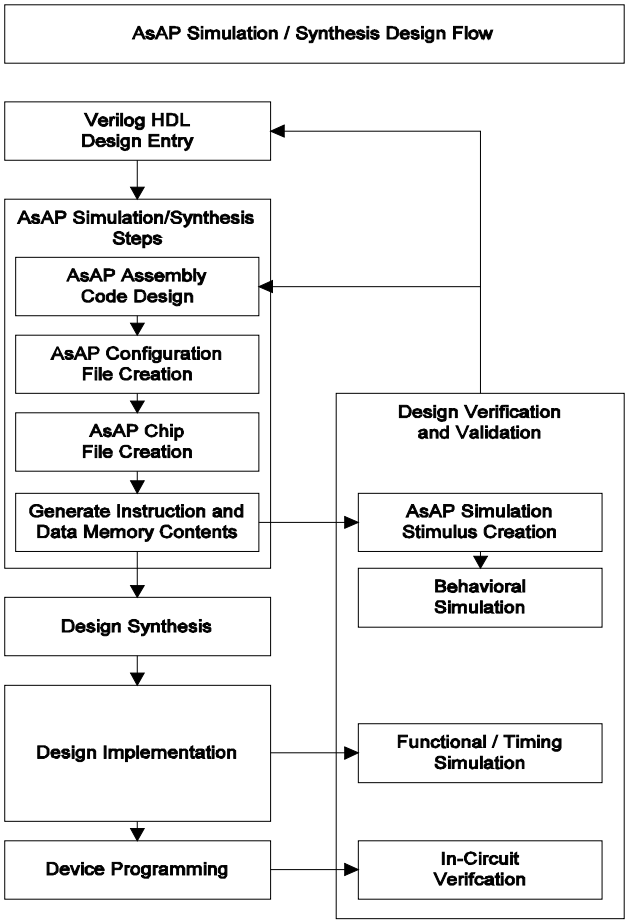

Figure 1. AsAP Simulation/Synthesis Design Flow diagram.

3. PROJECT DESIGN APPROACH

3.1 Block Diagram

The block diagram in Figure 2 illustrates the AsAP single processor implementation as it would appear in a fabricated ASIC chip:

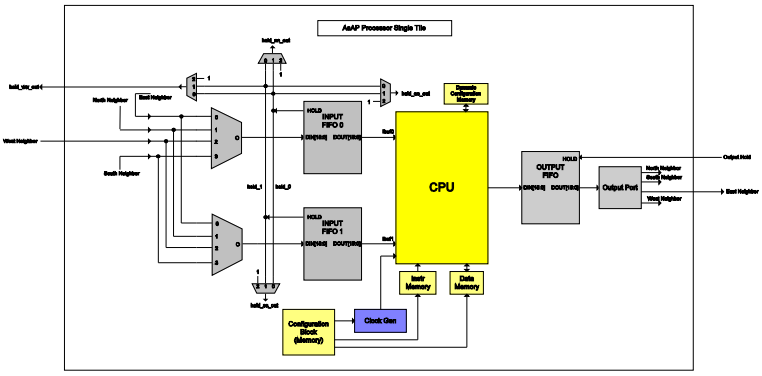

Figure 2. AsAP Single Processor ASIC implementation block diagram.

Due to limitations of FPGAs some compromises had to be made during the validation and implementation processes. These compromises were as follows:

- The programmable clock oscillator was removed.
  - AsAP driven by external clock input.
- All configuration registers were hard coded.
  - Essentially removing the configuration block.
- Instruction and data memories were pre-loaded

The block diagram of the AsAP Single Processor FPGA implementation can be seen in Figure 3 below:

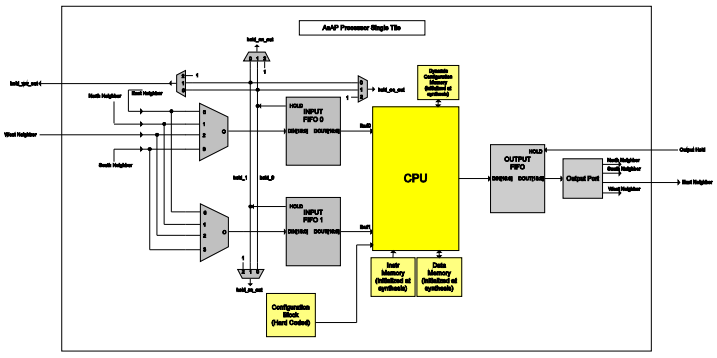

Figure 3. AsAP Single Processor FPGA Implementation block diagram.

3.2 Verilog HDL Implementation

When I started the validation and FPGA implementation of the AsAP processor, a Verilog HDL design had already existed. However, the existing code was not written with the possible FPGA implementation in mind. My task was to carefully mold into shape, such that it would fit neatly into an FPGA. The main pieces of code that were modified, due to FPGA limitations, were the compromises listed in section 3.1.

The AsAP's programmable clock oscillator had to be eliminated from the FPGA implementation, as a result of the FPGA's global clock tree construction. I therefore bypassed the programmable oscillator and tied the CPU clock to the external clock input of the AsAP processor. The purpose of the programmable clock oscillator in the AsAP architecture is to have the ability to generate different CPU clocks for different processors. Since this FPGA implementation is for a single processor only, the elimination of the programmable oscillator is not too large of a compromise. The two input FIFOs still contain there asynchronous clock inputs, so the asynchronous operation of the AsAP architecture remains in tact. As a result of bypassing the programmable oscillator, the Verilog HDL programmable clock oscillator module was almost completely optimized out of the design during synthesis. The only logic remaining in the module was the buffer that tied the external clock input to the CPU clock.

Since no method existed at the time of this project to stream live configuration data to the AsAP processor within the FPGA, it was necessary to hard code the configuration registers. There were a total of twenty single-bit and multi-bit configuration registers. As a result the Verilog HDL configuration module was optimized out of the AsAP processor design during synthesis.

Both the instruction and data memories mapped well to the Block RAM located inside the Xilinx Spartan 3 FPGA. The Block RAM contains 16,384 bits of data memory. The instruction memory was organized as 32-bits wide x 512 words deep; however, the AsAP processor only uses a 32-bits wide x 64 word deep memory. Therefore the excess words are left unused. The data memory was organized as 16-bits wide x 1024 words deep; however, the AsAP processor only uses a 16-bits wide x 128 word deep memory. As with the instruction memory, the excess words are left unused. Both the instruction and data memories were initialized using synthesis attributes in the Verilog HDL code. The Verilog HDL *defparam* keyword was used for simulation. Initializing the memories in the Verilog HDL code allowed for the project to progress.

3.3 Simulation Results

Simulation of the single AsAP processor was necessary to ensure the same functionality of the synthesizable design as the original Verilog HDL design. To simulate the AsAP design I used both the NC-Verilog simulator and the SimVision waveform viewer. The following is the AsAP assembly code that I used to simulate the design:

```
begin 0,0
MOVE dcmem 18 #1 nop3 // set OBUF to east, wait 3 cycles
NOP nop2  // wait 2 more cycles
RPT #0 // repeat forever
NOP nop2
ADD Obuf Ibuf0 SMem 0
end
```

This assembly code simply adds 1 to the input data and passes the result to the output. After simulating the two designs,
I discovered the designs were equivalent. Figures 4 and 5 show simulation results of both designs.

The critical simulation tasks for this project were to verify that the synthesizable AsAP design functioned with both the hard-coded configuration registers and the new Block RAMs for the instruction and data memories. As can be seen in Figure 5, the synthesizable design functions correctly.

3.4 FPGA Identification

Many factors were evaluated during the process of choosing an FPGA for the validation and implementation of a single AsAP processor. The main factors involved were area requirements for the single AsAP processor, FPGA speed, and FPGA development board cost and availability. To estimate the area requirements I ran a preliminary place & route after the modifications were made to the AsAP Verilog HDL design. The slice count estimate from the Xilinx ISE 6.3i tools was 1,676. Due to the small working budget, I chose to use the Spartan-3 Starter Kit from Digilent, Inc. and Xilinx, Inc.. The starter kit used an XC3S200 containing 1,920 slices. The XC3S200 could also be run at a maximum clock rate of 326 MHz.[2] The single AsAP processor used 87% of the XC3S200 Spartan 3 FPGA. As a general design rule, I try not to use more than 75% of an FPGA for any design. Using more than 75% of an FPGA reduces the flexibility to modify a design as it matures and bugs are discovered. Unfortunately, using a larger FPGA was not feasible due to budgetary reasons.

Table 1. Comparison of evaluation factors.

Xilinx Development Kits[3]		
Description	**# of Slices**	**Price ($)**
Virtex-2 Pro FF1152 P20 Development Kit	9,280	1,595.00
Virtex-2 Pro P4 Development Kit	3,008	895.00
Spartan-3 Development Kit	13,312	749.00
Spartan-3 MB 3S1500 Development Kit	13,312	695.00
Virtex-2 MB 2V1000 Development Kit	5,120	595.00
Virtex-2 DSP Design Kit (XC2V1000)	5,120	400.00
Virtex-2 LC1000 Development Kit	5,120	395.00
Spartan-3 Starter Kit (XC3S200)	1,920	99.00

.

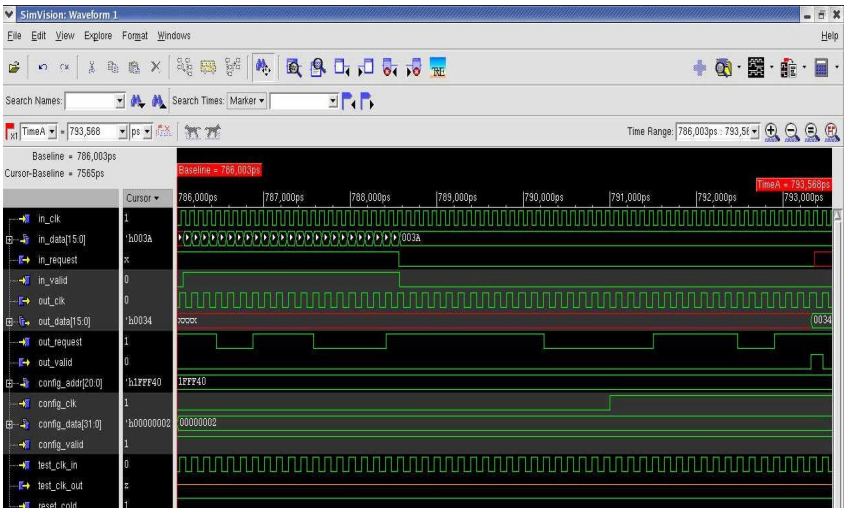

Figure 4. AsAP Single Processor ASIC simulation.

3.5 AsAP FPGA Implementation

To synthesize the AsAP design I used Synplicity's Synplify Pro software. To translate, map, and place & route the design I used Xilinx' ISE 6.3i software. The Synplify Pro tool synthesizes the Verilog HDL design to an Electronic Design Interchange Format (EDIF) netlist, which is used by the Xilinx translate tool. The Xilinx translate tool then reads in the EDIF file and generates a new file that describes the logical design in terms of logic elements such as AND gates, OR gates, decoders, flip-flops, and RAMs.

During the synthesis process I was able to evaluate the RTL and Technology design views. The evaluation consisted of ensuring that unintentional latches were not inferred, checking the inferred Xilinx primitives, and evaluating the critical path of the design. Analyzing the technology view I discovered that I needed to instantiate a signed multiplier, rather than rely on the synthesis tool, to infer the correct Xilinx primitive. Upon re-coding the multiply-accumulate (MAC) module, the signed multiplier was properly synthesized by Synplify Pro.

used during initial hardware tests. The final implementation of the AsAP single processor design produced the following mapping results:

Target Device : x3s200 (Xilinx Spartan 3)
Target Package : ft256, Target Speed : -4
Logic Utilization:
Number of Slice Flip Flops: 1,148 out of 3,840 29%
Number of 4 input LUTs: 2,222 out of 3,840 57%
Logic Distribution:
Number of occupied Slices: 1,670 out of 1,920 86%
Total Number 4 input LUTs: 2,390 out of 3,840 62%
Number used as logic: 2,222
Number used as a route-thru: 21
Number used for Dual Port RAMs: 128
Number of Block RAMs: 3 out of 12 25%
Number of MULT18X18s: 1 out of 12 8%
Number of GCLKs: 4 out of 8 50%
Number of DCMs: 1 out of 4 25%

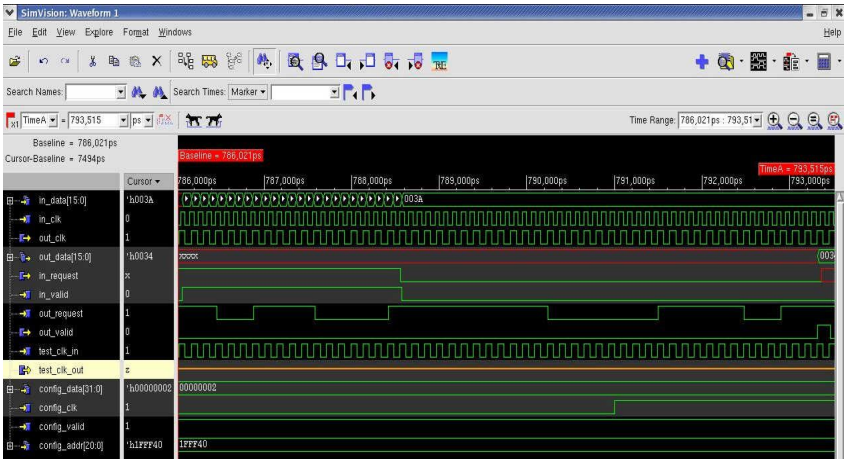

Figure 5. AsAP Single Processor FPGA simulation.

During the translate, mapping, and place & route process I discovered that the design was not meeting my timing constraint of 100 MHz. At first I investigated the possibility of adding extra pipeline stages to improve the timing of the design. Even though the added pipeline stages improved the timing of the design I realized I could find better ways to improve timing. Using the Xilinx Timing Analyzer I discovered the place & route tool was trying to push the configuration clock to operate at the specified global timing constraint set by the synthesis tool. The configuration clock is used by the instruction memory module to transfer data to the data memory when necessary, and is only running at 1 MHz. To mitigate this timing issue I placed a timing ignore constraint on the portion of the design operating on the 1 MHz configuration clock. The initial design only achieved a maximum operating clock frequency of ~52.7 MHz, but after adding proper constraints the design achieved a maximum operating frequency of ~201.9 MHz.

At the time of my project presentation, I was only using a single clock to operate the entire single AsAP processor. I have since utilized one of the digital clock managers (DCM) in the Spartan 3 FPGA to generate a 90º out-of-phase clock to drive the input FIFO, thus completely testing the dual clock input FIFO. The DCM was also used to generate the 200 MHz external clock input

3.6 AsAP Hardware Validation

The single AsAP processor design was validated under the following test conditions:

Test program: add 1 to the input 16-bit data stream.
AsAP clock rate: 50MHz
- Initial tests ran at 200MHz, but due to limitations of pc board design clock rate was reduced to 50MHz.

Input data set by a bank of 8 slide-switches.
- asap_input[15:0] = {sw[7:0],sw[7:0]}.

Configuration hard-coded.
- Input multiplexer selects, oscillator frequency, instruction memory, data memory.

Figure 6 shows the hardware test setup for validating the design. An Agilent Technologies 1.5 GHz / 5 GSa/s Infiniium Oscilloscope was used to measure both the output clock and the output data's least significant bit (LSB). Screen shots of the single AsAP processor's results are shown in Figures 7 and 8. Figure 7 displays the result of adding 1 to a 16-bit input value of zero, and Figure 8 shows the result of adding 1 to a 16-bit input value of 1.

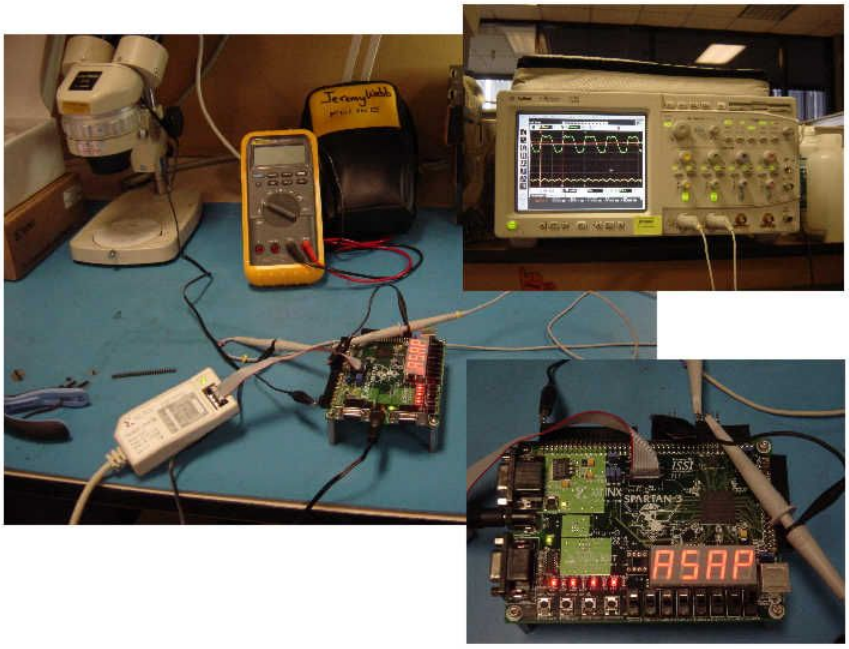

Figure 6. AsAP Single Processor HW Test Setup.

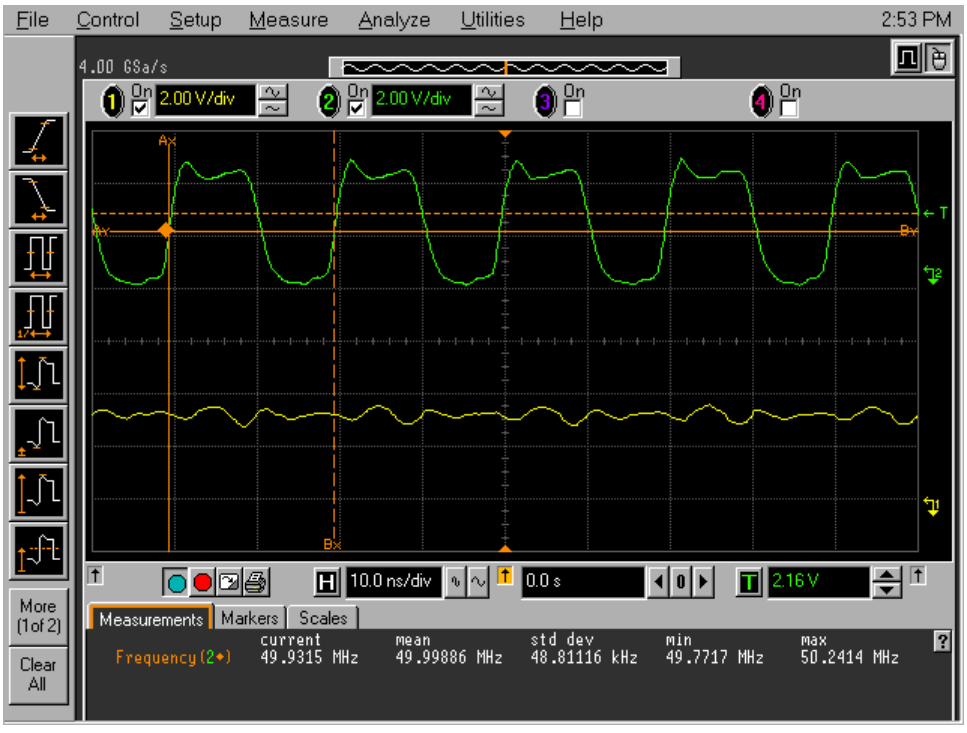

Figure 7. Result of adding 1 to an input value of zero.

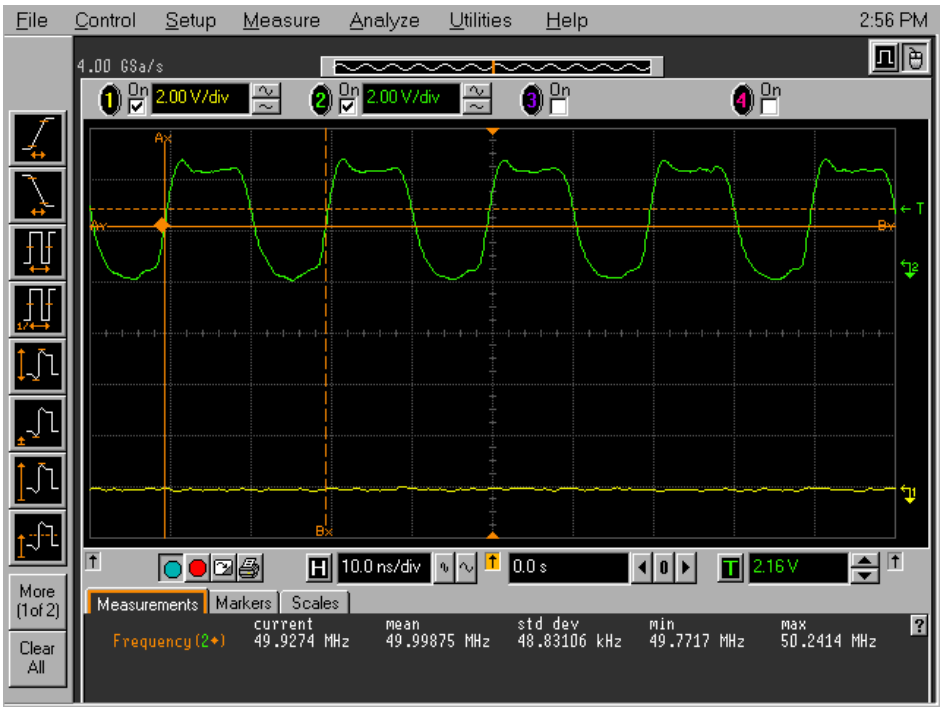

Figure 8. Result of adding 1 to an input value of one.

The hardware validation of the single AsAP processor design was very successful, and there were few problems encountered during this process. The problems that did arise, were unrelated to the AsAP processor design. During the validation process I discovered a problem with the Digilent, Inc.'s Spartan 3 Starter Kit board; it was exhibiting severe signal integrity issues. While the AsAP processor was operating at 200 MHz, the 16-bit AsAP output signals showed a DC offset of ~200 mV. To eliminate the excessive DC offset I slowed the AsAP external clock input down to 50 MHz. A possible reason for the elimination of the DC offset at a 50 MHz clock rate could be that the Starter Kit board was designed with a 50 MHz clock oscillator.

4. FUTURE ENHANCEMENTS

For future implementations of the AsAP processor, the following is a list of possible ideas to help improve the overall results:

1. Design a method for streaming live configuration data to the AsAP processor after the FPGA has been configured.
    a. Possible methods are to design a UART/RS-232 FIFO interface.
2. Design a custom PC board with a large enough FPGA to implement multi-processor arrays.
    a. Route PCB traces using matched length, controlled impedance transmission lines.
    b. Provide high-performance peripheral devices (e.g., High-Speed A/D and D/A's, Video Outputs, SDRAM, and a serial I/O interface).

5. CONCLUSION

Overall, the hardware implementation and validation of the single AsAP processor design was very successful. The AsAP processor in the FPGA performed the programmed addition and the results agreed with the simulation shown in section 3.3. In addition, improvements were made to the current AsAP design that will benefit the final ASIC implementation.

6. ACKNOWLEDGMENTS

I would like to thank Professor Bevan Baas and the entire AsAP design team for their help in learning more about the AsAP processor. I would also like to thank my loving wife for helping me with the editing of this project report.

7. REFERENCES

[1] Jeremy W. Webb, "UC Davis AsAP FPGA Implementation," In EEC 289Q: Reconfigurable Computing, ECE Department, University of California, Davis, October 2004.

[2] "Spartan-3 FPGA Family: Introduction and Ordering Information". *DS099-1. Xilinx, Inc.,* (July 13, 2004), 1.

[3] "*http://www.ece.ucdavis.edu/~jwwebb/asap_demo_boards.html*", Webb, Jeremy W., 2004.