ECSE 548 (VLSI) Project: MIPS Processor On-Chip Cache Design and Integration

***Abstract*—In this paper, we present an on-chip cache design and integration for 8-bit MIPS processor. The cache that we adopt is a direct-mapped cache with 1 byte cache blocks. It has a size of 16 bytes and is used for both instructions and data. We apply a top-down design methodology, i.e. we start from the architecture level and go down to its layout, with testings on each level. To integrate the cache, MIPS Finite State Machine (FSM) is modified accordingly, which results in controller architecture modifications. Assembly code is also rewritten to verify cache functionality. Design quality is tested using SPICE.**

I. Introduction

Cache is a small memory that contains the most recently accessed pieces of main memory, used to speed-up the processor-memory operation. Due to **Locality of Reference**, where a process at any given time accesses only a small region of memory, a cache helps significantly reduce processor I/O overhead by loading the small region that can be accessed by the processor with a much faster rate. For example, the typicall memory access time of 60 ns of a Pentium processor can be reduced to 15 ns with the presence of a cache [1].

In this report, we propose a design for a simple direct-mapped cache for MIPS processor using SRAM memory bits. The report is divided into seven sections. Section II and Section III provides detailed descriptions of propsed design for cache architecture and modifications on the existing MIPS processor for the integration between the two parts; then Section IV presents the process of implementing the design. In Sections V and VI, both functional and performance testing on the design is discussed. Section VII concludes the design process along with the whole report.

II. Cache Architecture

The MIPS processor presented in the course takes an 8-bit address line and 8-bit data line every time it reads data from external memory; to maintain a manageable project scale while ensuring basic functioanlity of the cache, the cache was designed to split up the address line into a 4-bit tag line and a 4-bit set line, where set signals are used for row selection in cache whereas tag signals are used for comparison to generate cache hits. The designed configuration results in a 16 8-bit data row in cache, making it capable of storing 16 bytes (4 distinctive assembly instructions) for MIPS.

The resulting design of the cache is shown in Figure 1. Set signals are used to decode data rows inside SRAM arrays, and tag signals are used to compare the tags stored in cache to generate cache hit signals. Wite enable (*we* as shown in figure) signal ensures the cache is writeable whenever it set to high. Additionally, for better stability, a validation bit (*v*) was added to each row of SRAM array inside the cache to implement *Compulsory Miss Strategy*, where the first read from cache is always guaranteed to generate a cache miss signal.

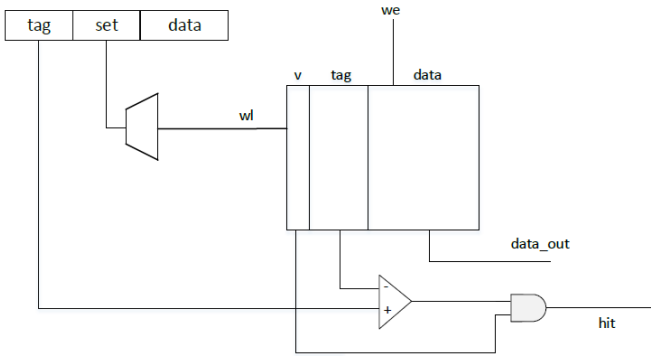

Fig. 1. Cache Microarchiture

III. MIPS Integration Architecture

We adopt hierarchical top-down design methodology. We go through architecture level, schematic level and layout level. At architecture level, the Finite State Machine (FSM) is modified and the controller is changed accordingly, see Figure 2. This is implemented in SystemVerilog and the functional simulation has been verified successfully before going down to schematic level. The overall MIPS microarchitecture is shown in Figure 3.

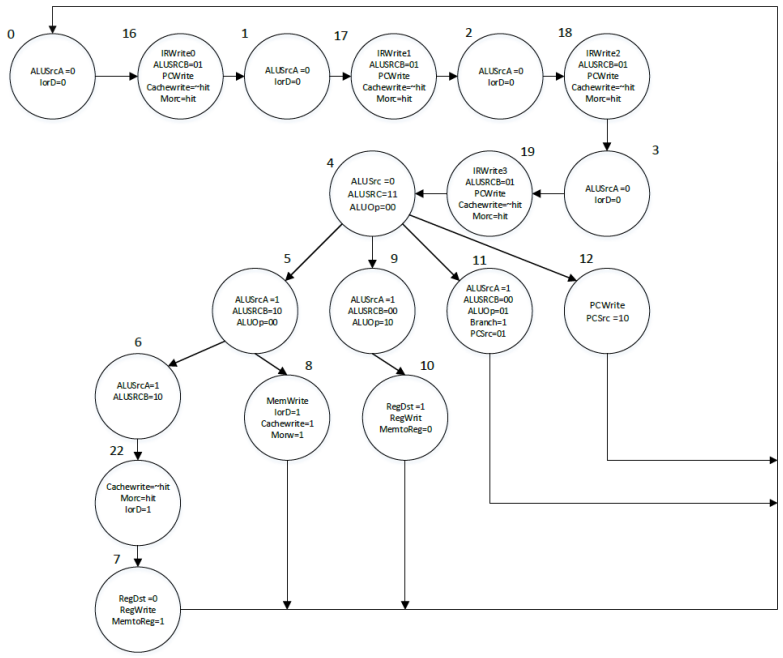

Fig. 2. Modified FSM of Controller

Since our cache is used for both instructions and data, in following sections, when we say data, we refer to both instructions and data unless specified otherwise.

In Figure 3, three output signals (*cachewrite, morw, morc*) and one input signal (*hit*) are added to the controller.

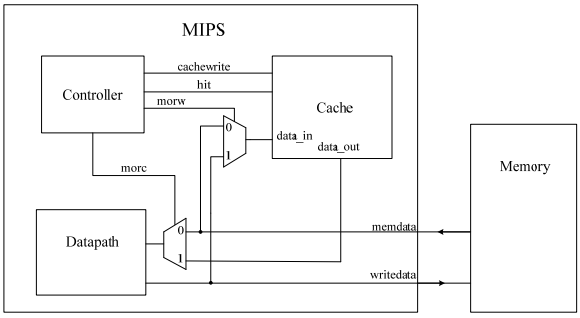

Fig. 3. MIPS Microarchiture

- *cachewrite* is used to enable write/read access to the cache with a value of 1/0.
- *morw* is used to select data from memory or datapath. If a cache miss happens, it is set to 0 and the data from memory is written into the cache.
- *morc* is used to select data from memory or from cache. If there is a cache hit, it is set to 1 and the data from cache is sent to MIPS. Otherwise it is set to 0 and memory data is used.
- *hit* is used to show status of cache. 1 represents a cache hit and 0 represents a cache miss. A compulsory miss strategy was implemented for an ensured miss for a first-time read from cache.

In Figure 2, we split one state for data read access into two states. The first state is used to see if there is a cache hit. If so, at second state, data from cache is read; otherwise data is fetched from main memory.

We adopt write-through policy, so the data is always written to both cache and memory. For data write access, we still use one state.

IV. Implementaion

This section discusses the implementation process of cache components and integration in Electric.

A. Comparator

A 4-bit comparator was implemented to compare the *tag* bits from the memory and cache to generate corresponding *cache hit* signals. The designed schematic and layout can be found in Figure 4.

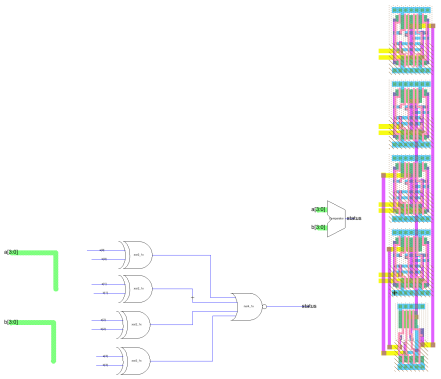

Fig. 4. The Schematic and Layout of Comparator Design

Figure 5 indicates that the comparator design was valid, passing all three design rule checks.

```
================================3================================
Checking schematic cell 'muddlib07:nor4_1x{sch}'
   No errors found
Checking schematic cell 'muddlib07:xor2_1x{sch}'
   No errors found
Checking schematic cell 'comp2_1x_4{sch}'
   No errors found
0 errors and 0 warnings found (took 0.01 secs)
================================4================================
Checking Wells and Substrates in 'vlsi:comp2_1x_4{sch}' ...
No Well errors found (took 0.005 secs)
================================5================================
Hierarchical NCC every cell in the design: cell 'comp2_1x_4{sch}'  cell 'comp2_1x_4{lay}'
Comparing: muddlib07:nor4_1x{sch} with: muddlib07:nor4_1x{lay}
  exports match, topologies match, sizes match in 0.003 seconds.
Comparing: muddlib07:xor2_1x{sch} with: muddlib07:xor2_1x{lay}
  exports match, topologies match, sizes match in 0.002 seconds.
Comparing: vlsi:comp2_1x_4{sch} with: vlsi:comp2_1x_4{lay}
  exports match, topologies match, sizes match in 0.003 seconds.
Summary for all cells: exports match, topologies match, sizes match
NCC command completed in: 0.012 seconds.
```

Fig. 5. Deisgn Rule Checks for Comparator

B. 4-to-16 Row Decoder

In order for the cache to make use of the 4-bit set signals, a row decoder was applied to decode the set signal and select 1 out of 16 8-bit data lines lying inside the cache. In order to reduce the delay and space usage of the decoder, *pre-decoding* technology was used, resulting in a design shown in Figure 6. It was worth noting that the layout of the decoder was delibrately designed to be wide with a minimum height in order to pitch-match with the sram array inside the cache.

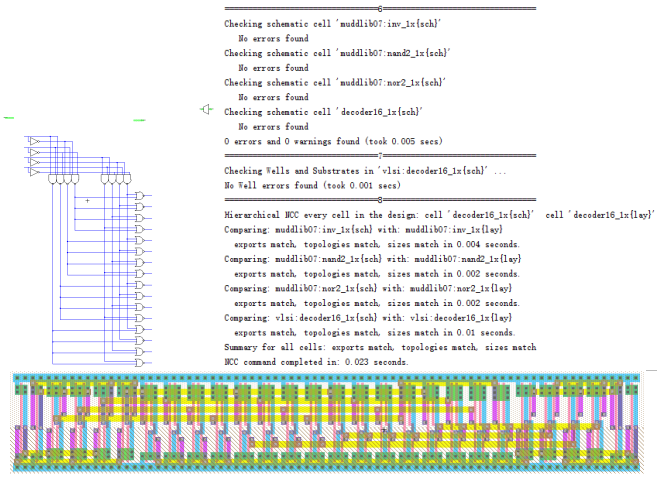

Fig. 6. The Schematic and Layout of Decoder Design With Design Rule Check Results

It can be shown from the Figure 6 that the decoder design passed all three design rule checks and was ready for the integration into the cache deisgn.

C. SRAM Column

An SRAM column represents a column of SRAM bits inside the SRAM array of the cache, consisting of 16 SRAM bits sharing bit and inverting bit lines. The design was presented in Figure 7.

The design rule check results can be found in Figure 7. It can be shown from the Figure that all three of the design rule requirements were satisfied by the design.

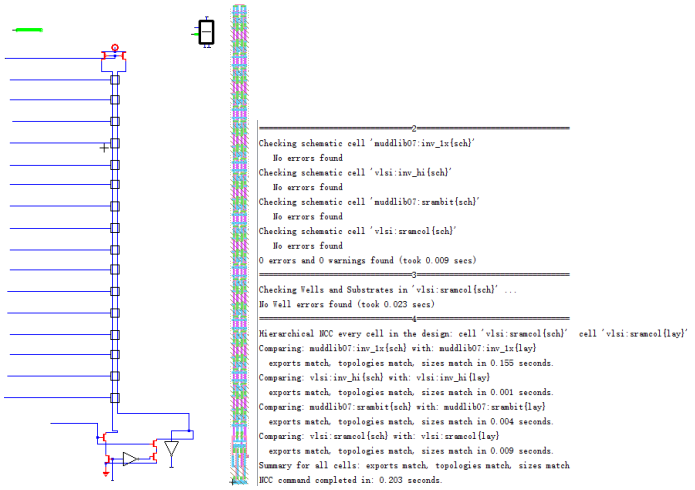

Fig. 7. The Schematic and Layout of SRAM Column With Design Rule Check Results

D. SRAM Array

The SRAM array is the most essential part of the cache, consisting of 16 *13-bit* SRAM bit rows, with each row having 1 bit for validation (compulsory miss), 4 bits for tag line and 8 bits for data storage. Each row also has a wordline buffer for a stable charging-up. The design is presented in Figure 8; the design rule check results can be found in the same figure as well.

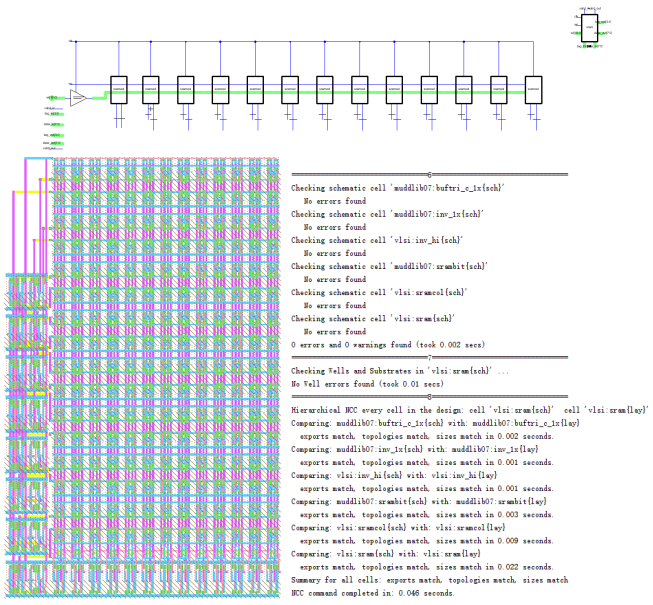

Fig. 8. The Schematic and Layout of SRAM Array With Design Rule Check Results

E. Cache

After the implementation of all circuit components, the cache was built based on the microarchitecture of the proposed cache circuit design. For easy integrations with the MIPS processor, in the layout, universal *vdd* and *gnd* tracks were added that connects *vdd* and *gnd* of sub-components together. The Cache circuit design, along with the evidence that it passes all three design checks, is presented in Figure 9.

F. PLA Controller & Controller

As indicated in Section III, with updated FSM for the controller, the PLA controller which represents the FSM logic

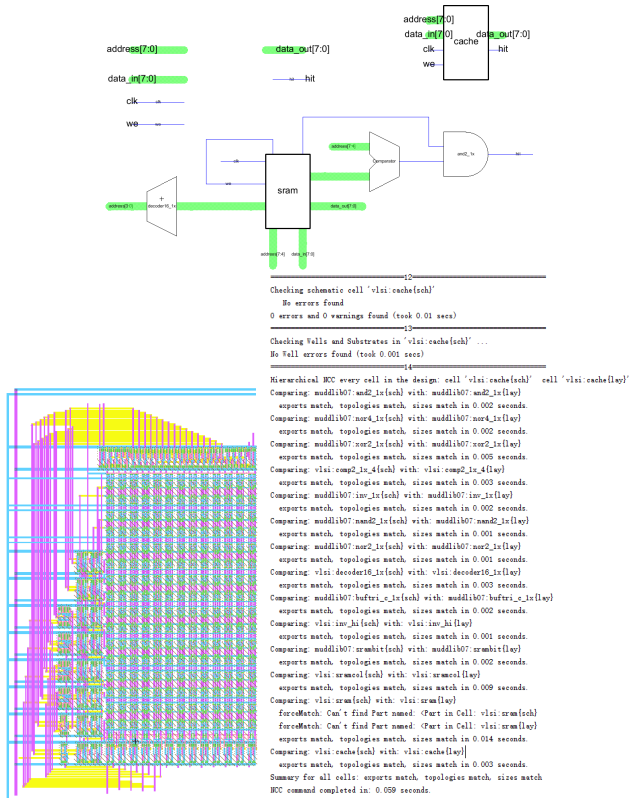

Fig. 9. The Schematic and Layout of Cache With Design Rule Check Results

has been modified accordingly. With an updated FSM description file, a new PLA controller was generated and re-integrated into the controller citcuit of MIPS processor. New exports corresponding to cache signals have been added to the controller circtui as well. The resulting circuit is presented in the next subsection.

G. Integration

Upon the completion and tests of each component, the cache was then integrated with MIPS processor (with updated controller component), resulting in a circuit presented in Figure 10.

The pad frame was then generated from the new MIPS processor with cache implemented. To contain the much-expanded size of the MIPS, the size of the pad frame was extended from *40* pads to *64* pads. The resulting chip, along with error-free design rule check results, is shown in Figure 11. Due to page limits, a truncated version of design rule checks for the chip is presented in this report; the full check has been presented during the demo with the professor.

V. Testing

In this section, we explain our testing methodology. We tested components individually in Section V-A. After assembly, the SystemVerilog MIPS processor is simulated for functional testing in Section V-B.

A. Components Testing

The main components to be tested are shown below:

- Comparator
- Decoder The 4-to-16 decoder was tested in ModelSim with all 16 test cases; the results showing that the design was operating properly is presented in Figure 12.

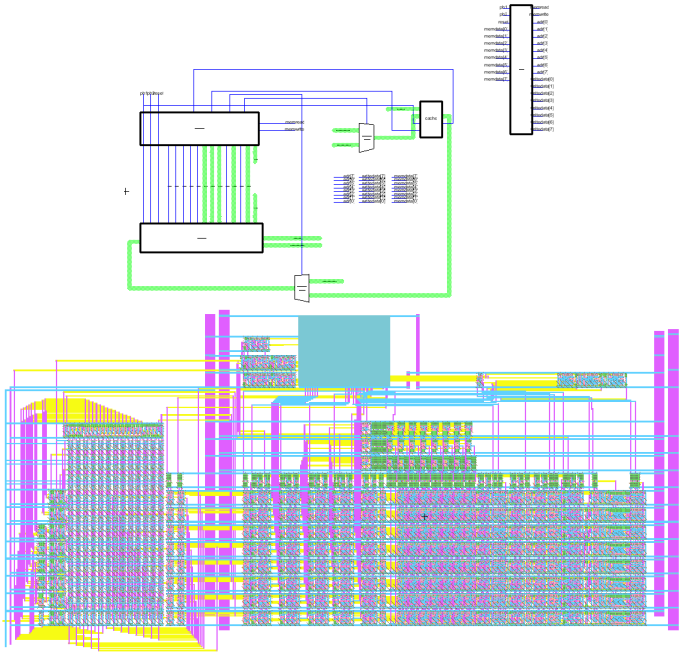

Fig. 10. The Schematic and Layout of MIPS integrated with Cache

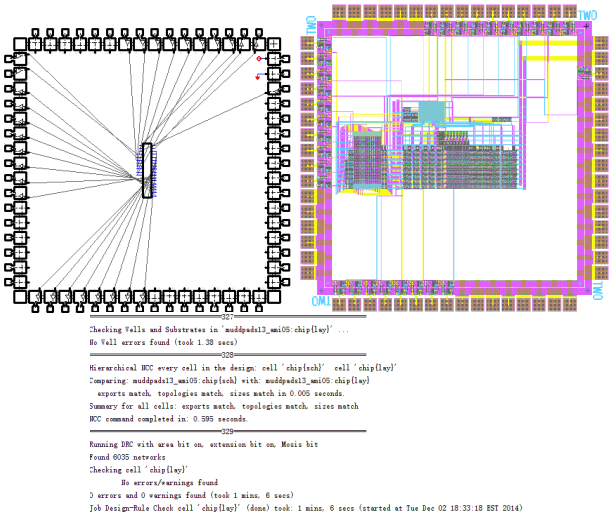

Fig. 11. The Schematic and Layout of Chip Integration With Design Rule Check Results

```
# Input: 0000, Ouput: 0000000000000001, Expected: 0000000000000001
# Input: 0001, Ouput: 0000000000000010, Expected: 0000000000000010
# Input: 0010, Ouput: 0000000000000100, Expected: 0000000000000100
# Input: 0011, Ouput: 0000000000001000, Expected: 0000000000001000
# Input: 0100, Ouput: 0000000000010000, Expected: 0000000000010000
# Input: 0101, Ouput: 0000000000100000, Expected: 0000000000100000
# Input: 0110, Ouput: 0000000001000000, Expected: 0000000001000000
# Input: 0111, Ouput: 0000000010000000, Expected: 0000000010000000
# Input: 1000, Ouput: 0000000100000000, Expected: 0000000100000000
# Input: 1001, Ouput: 0000001000000000, Expected: 0000001000000000
# Input: 1010, Ouput: 0000010000000000, Expected: 0000010000000000
# Input: 1011, Ouput: 0000100000000000, Expected: 0000100000000000
# Input: 1100, Ouput: 0001000000000000, Expected: 0001000000000000
# Input: 1101, Ouput: 0010000000000000, Expected: 0010000000000000
# Input: 1110, Ouput: 0100000000000000, Expected: 0100000000000000
# Input: 1111, Ouput: 1000000000000000, Expected: 1000000000000000
# Completed     16 tests with      0 errors.
```

Fig. 12. ModelSim Test Results for decoder

- SRAM:

  This is the storage elements in cache. As shown in Figure 1, it has 16 sets and each set consists of 1 *valid* bit, 4 *tag* bits and 8 *data* bits. The input signal *we* is set to 0/1 for data read/write and the signal *wl* is used to select the set for access. Initially, the *valid* bit is 0. Once a cache set is written, its corresponding *valid* becomes 1. *valid* is used to indicate if the data has been loaded or not. We produce some random inputs for testing. Due to the complexity of memory accesses, we generate test vectors from the inputs using our *Python* script, as shown in Figure 13. The signal representation is listed in Table I. The functionality of SRAM is verified successfully according to test vectors.

```
00000000000000100011111111000000000000000010
00000000000000100011111111000000000000000010
00000000000000100011100111000000000000000011
00000000000000100111111111000000000000100001
11000111001111000111111111000000000000000010
11001111111110100011111111000000000000100000
11000111001111010111111111000000000000000011
11010111111111110111111111000000000000000011
11110111111111110111111111000000000000000010
```

Fig. 13. SRAM Test Vectors

TABLE I. SRAM TEST VECTOR

bit position	signal
0	we
16:1	wl
24:17	data_in
28:25	tag_in
36:29	expected_data
40:37	expected_tag
41	expected_valid

B. *Functional Testing*

In this section, the functional testing of MIPS SystemVerilog model is demonstrated. For this purpose, assembly code is written, as shown in Listing 1.

```
1  main:    # assembly code       # effect
2           lb $2, 80($0)         # $2 = 5
3           lb $7, 76($0)         # $7 = 3
4           lb $3, 81($7)         # $3 = 12
5           or $4, $7, $2         # $4 <=3 or 5 = 7
6           and $5, $3, $4        # $5 <=12 and 7 = 4
7           add $5, $5, $4        # $5 <= 4+7 = 11
8           beq $5, $7, end       # not taken
9           slt $6, $3, $4        # $6 <= 12 < 7 = 0
10          beq $6, $0, around    # not taken
11          lb $5, 0($0)          # not taken
12 around:  slt $6, $7, $2        # $6 <= 3 < 5 = 1
13          add $7, $6, $5        # $7 <= 1 + 11 = 12
14          sub $7, $7, $2        # $7 <= 12 - 5 = 7
15          add $3, $7, $0        # $3 = 7
16 loop:    beq $3, $0, end       # taken if $3 = 0
17          sub $3, $3, $6        # $3 = $3 - 1
18          j loop                # taken
19          lb $7, 0($0)          # not taken
20 end:     sb $7, 71($2)         # write adr 76 <=7
21          .dw 3
22          .dw 5
23          .dw 12
```

Listing 1. Assembly Code

At the beginning, instructions are fetched, and there are compulsory cache misses. The assembly code contains a loop to test cache hits. When dealing with this loop, there are cache hits. In Figure 14, we can see that there is a cache miss. This happens because the instructions are used for the first time here. During program execution, cache hits occur for the loop. This is illustrated in Figure 15. The instructions in the loop are put in the cache and then fetched soon. As a result, cache hits happen here.

Apart from cache read access, we also checked cache write access, as shown in Figure 16. In this case, signal *morw* is set

to 1 to select data and *cachewrite* is set to 1 to write the data from datapath into cache.

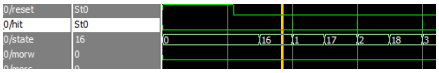

Fig. 14. Cache Read Miss

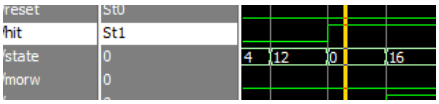

Fig. 15. Cache Read Hit

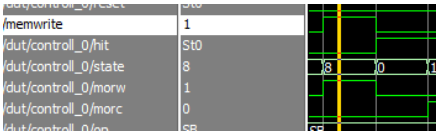

Fig. 16. Cache Write

The result is shown in Figure 16. If cache is absent, the write address *adr* is 76 and write data *writedata* is 7. After we integrated on-chip cache, the result is still correct. Our cache functionality is thus verified.

VI. Design Quality

In this section, performances of certain components, such as regular inverter, high-skew inverter as well as and gate, are evaluated using CAD tool(i.e.PSPICE) since we are not capable to perform evaluations directly on integrated circuits blocks due to the circuit size limitation from CAD tool. The corresponding results are shown in **Fig.17** for the setup time analysis of each component respectively. As can be seen clearly from the output waveform, the high-skew inverter pulls up faster than the regular inverter as expected. As for the power dissipation, the corresponding total power consumption of a and gate, inverter and high-skew inverter are $18.9\mu W$, $22.2\mu W$ and $12.7\mu W$, respectively. Therefore, skewed logic is clearly one of the high performance and low power-carry logic components available currently.

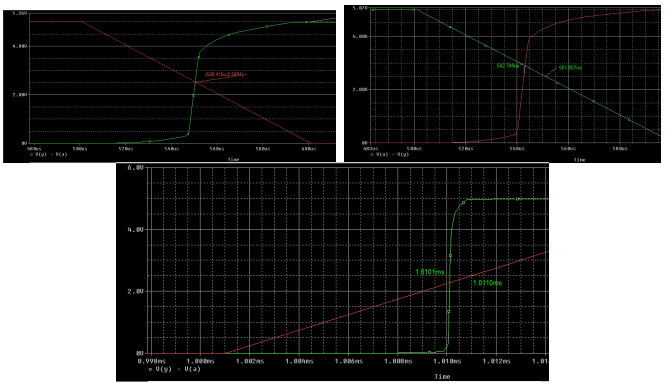

Fig. 17. SPICE simulation of INV, high-skew INV and AND gate

VII. Conclusion

The Cache implementation can be mould in a number of ways, in terms of associativity which can decrease access time but increase complexity so in engineering world need to live with this trade-off. But basic working principal remains same.As discussed earlier we have successfully implemented the direct mapped cache and successful integration with MIPS. cross severa the initially defined boundaries. It is important to understand that the Pentium(R) Processor uses only one method to implement cache. We have successfully tested the all components their simulation verification done in ModelSim, Schematic and layout implementation and testing done in Electric. We also employed Spice simulation.In short Cache is simply a high speed memory that stores a piece of main memory which helps processor for high performance he conclusion goes here.

References

[1] http://download.intel.com/design/intarch/papers/cache6.pdf The conclusion goes here.