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 is used to speed-up the processor-memory operation. A cache contains the most recently accessed pieces of main memory. The question arises why we need this high speed memory?. The time for processor to fetch a data from main memory is very long or worst for the disk.So, we need some thing in between memory and processor to speed up the whole process of data fetching, instruction fetching, read and write process.For example, a memory used in Pentium its access time is typical is 60ns for main memory (DRAM). [1]. It means 100 MHz processor can execute most instructions in 1 CLK or 10 ns. Its a bottleneck of this processor. Cache memory helps by reducing the time it takes to read/write information to and from the processor. A typical access time for SRAM is 15 ns [1]. Therefore cache memory allows small portions of main memory to be accessed three to four times faster than main memory But,How can this small piece of memory decrease the access time. In theory this concept called **Locality of Reference**. It means at any given time the processor access only a memory in a small or localized region of a memory. The cache loads this region allowing the processor to access the memory region faster. So question arises, why not replace main memory with cache. The reason for not replacing memory with cache is cost of cache because for cache cell we need six transistor to store one bit. Cache memory (SRAM) is several times more expense than main memory (DRAM). The power consumption of SRAM is also higher than DRAM.

II. 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 **??**. 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 1.

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.

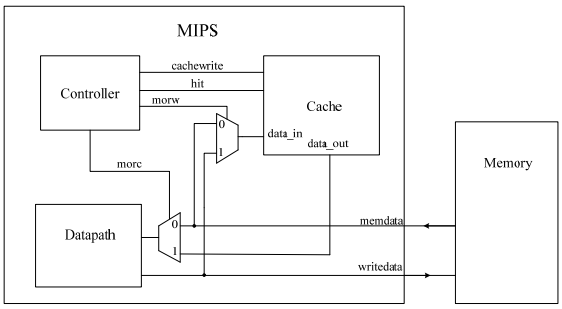

Fig. 1. MIPS Microarchiture

In Figure 1, three output signals (*cachewrite, morw, morc*) and one input signal (*hit*) are added to the controller.

- *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 **??**, 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.

III. 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 2.

Figure 3 indicates that the comparator design was valid, passing all three design rule checks.

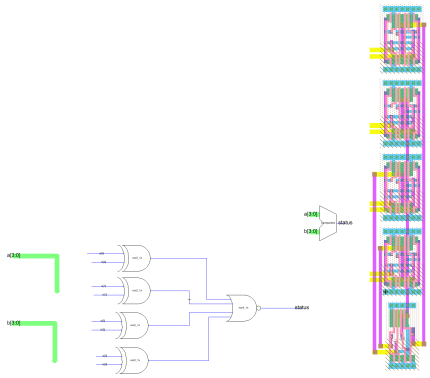

Fig. 2. The Schematic and Layout of Comparator Design

```
═══════════════════════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. 3. 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 4. 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.

It can be shown from the Figure 5 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 6.

The design rule check results can be found in Figure 7. The current design level of SRAM column did not pass NCC for unmatched number of *gnd* wire between schematic and layout, an error originated from SRAM bit standard cell. This error was appropriately ignored, as the issue will be resolved in cache level when all components share common *vdd* and *gnd* tracks.

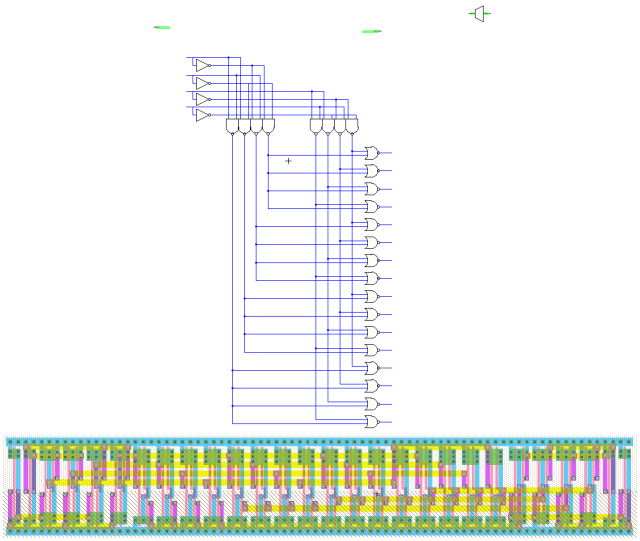

Fig. 4. The Schematic and Layout of Decoder Design

```
═══════════════════════6═══════════════════════
Checking schematic cell 'muddlib07:inv_1x{sch}'
   No errors found
Checking schematic cell 'muddlib07:nand2_1x{sch}'
   No errors found
Checking schematic cell 'muddlib07:nor2_1x{sch}'
   No errors found
Checking schematic cell 'decoder16_1x{sch}'
   No errors found
0 errors and 0 warnings found (took 0.005 secs)
═══════════════════════7═══════════════════════
Checking Wells and Substrates in 'vlsi:decoder16_1x{sch}' ...
No Well errors found (took 0.001 secs)
═══════════════════════8═══════════════════════
Hierarchical NCC every cell in the design: cell 'decoder16_1x{sch}'  cell 'decoder16_1x{lay}'
Comparing: muddlib07:inv_1x{sch} with: muddlib07:inv_1x{lay}
  exports match, topologies match, sizes match in 0.004 seconds.
Comparing: muddlib07:nand2_1x{sch} with: muddlib07:nand2_1x{lay}
  exports match, topologies match, sizes match in 0.002 seconds.
Comparing: muddlib07:nor2_1x{sch} with: muddlib07:nor2_1x{lay}
  exports match, topologies match, sizes match in 0.002 seconds.
Comparing: vlsi:decoder16_1x{sch} with: vlsi:decoder16_1x{lay}
  exports match, topologies match, sizes match in 0.01 seconds.
Summary for all cells: exports match, topologies match, sizes match
NCC command completed in: 0.023 seconds.
```

Fig. 5. Deisgn Rule Checks for Decoder

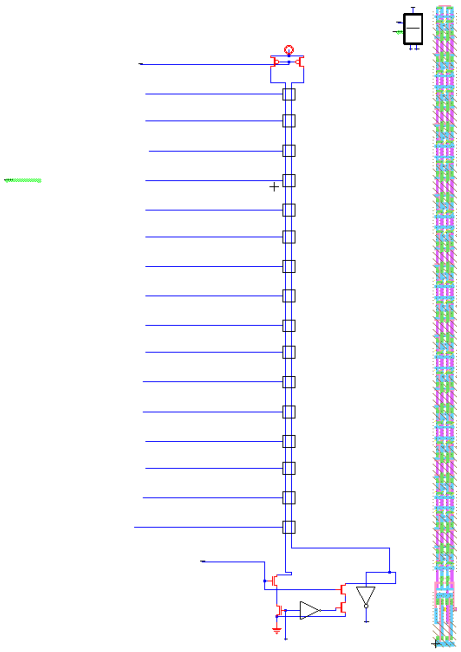

Fig. 6. The Schematic and Layout of SRAM Column

```
                    14
Checking schematic cell 'muddlib07:inv_1x{sch}'
   No errors found
Checking schematic cell 'inv_hi{sch}'
   No errors found
Checking schematic cell 'muddlib07:srambit{sch}'
   No errors found
Checking schematic cell 'sramcol{sch}'
   No errors found
0 errors and 0 warnings found (took 0.002 secs)
                    15
Checking Wells and Substrates in 'vlsi:sramcol{sch}' ...
No Well errors found (took 0.001 secs)
```

Fig. 7. Deisgn Rule Checks for SRAM Column

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.

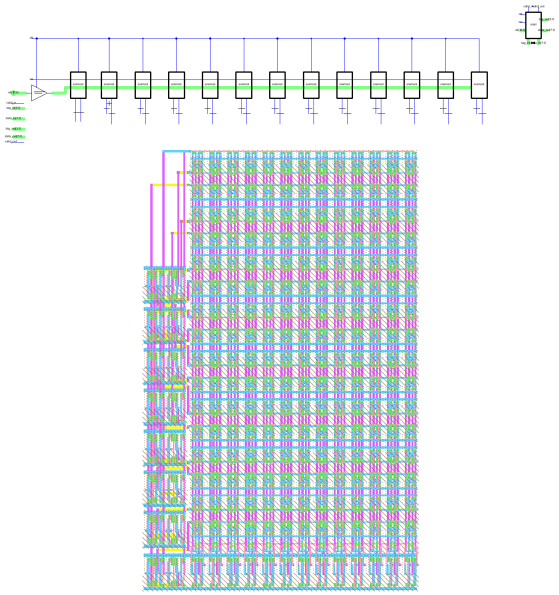

Fig. 8. The Schematic and Layout of SRAM Array

Similar to Section III-C, the SRAM array was not able to pass NCC for the same reason; similarly the error was ignored at this level as well. The DRC and ERC check results can be found in Figure 9.

```
                    16
Checking schematic cell 'muddlib07:buftri_c_1x{sch}'
   No errors found
Checking schematic cell 'muddlib07:inv_1x{sch}'
   No errors found
Checking schematic cell 'inv_hi{sch}'
   No errors found
Checking schematic cell 'muddlib07:srambit{sch}'
   No errors found
Checking schematic cell 'sramcol{sch}'
   No errors found
Checking schematic cell 'sram{sch}'
   No errors found
0 errors and 0 warnings found (took 0.002 secs)
                    17
Checking Wells and Substrates in 'vlsi:sram{sch}' ...
No Well errors found (took 0.005 secs)
```

Fig. 9. Deisgn Rule Checks for SRAM Array

IV. Testing

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

A. Components Testing

The main components to be tested are shown below:

- Comparator
- Decoder
- SRAM: this is the storage elements in cache.
- Cache:

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         j end                 # 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

V. Design Quality

VI. 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.