Cache Design Project

Class: CMPE 413

Semester: Fall 2022

Completed by: Mick Harrigan and Daniel Cleaver

Date: 2022-11-22

Description

This system from the top level chip all the way down uses many different connected parts to achieve the desired results.

The modules contained within this system are listed here as follows:

- State Machine
- Cache Block
- Tag Block
- Valid Chip Enable Generator
- Registers
- Output Enable Controllers
- Muxes
- 5 bit Counter
- 5 bit Comparator
- Hit/Miss Detector

This is further represented below with a deeper hierarchical design here below.

```
Chip
├── 5 Bit Comparator
├── 5 Bit Counter
│   ├── 5 Bit Adder
│   ├── 5 Bit Buffer
│   └── 5 Bit Register
├── Cache Block
│   ├── 4x Cache 4 byte
│   │   ├── Cache Byte Decoder
│   │   └── Cache Cell 8
│   │       └── 8x Cache Cell 1
│   └── Cache Row Decoder
├── Hit/Miss Detector
│   └── 2 Bit Comparator
├── Muxes
│   └── 2 to 1 Mux
├── Output Enable Controllers
│   ├── CPU Data Output Enable
│   └── Memory Address Output Enable
├── Registers
│   ├── CPU Address Register
│   ├── CPU Data Register
│   └── RD_WR Register
├── State Machine
├── Tag Block
│   ├── 4x Cache Cell 2
│   └── Tag Decoder
└── Valid Chip Enable Generator
    └── Valid Bit Decoder
```

We used github to store files and track changes. Our repo can be found here.

Design Strategy

State Machine

The state machine acts as a controller for all other modules in the top-level chip. The current state is stored in a register, and the output signals are calcualted using combinational logic. In addition, the next state is determined based on the current state and the inputs. The behavior of the state machine is described by the following tables.

Table 1 shows the list of states and a description of each.

Table 1: List of States

State name	State Code (Dec)	State Code (Bin)	Action
idle	0	0000	
rd_init	4	0100	Store inputs, read data, check for hit
rd_hit	5	0101	Send data to CPU
rd_miss_mem_e	12	1100	Send address to Mem
rd_miss_mem_w	13	1101	Wait for Mem
rd_miss_wr	8	1000	Write data to row
rd_miss_rd	6	0110	Read data
rd_miss_send	7	0111	Send data to CPU
wr_init	14	1110	Store inputs, check for hit
wr_hit	9	1001	Write data
wr_miss	15	1111	Do nothing
reset	1	0001	Reset

Table 2 shows the outputs for each state.

Table 2: State Output Table

Current State	cpu_busy	counter_ce	counter_rst	cpu_data_oe	mem_add_oe	mem_enable	cb_d_wr_control	cb_ce	cb_rd_wr	cb_offset_control	tb_ce	tb_rd_wr	valid_ce	valid_ce_all	valid_rd_wr	valid_d_wr
idle	0	0	0	0	0	0	x	0	x	x	0	x	0	0	x	x
rd_init	1	0	0	0	0	0	x	1	1	0	1	1	1	0	1	x
rd_hit	0	0	0	1	0	0	x	1	1	0	0	x	0	0	x	x
rd_miss_mem_enable	1	1	0	0	1	1	x	0	x	x	0	x	0	0	x	x
rd_miss_mem_wait	1	1	0	0	0	0	x	0	x	x	0	x	0	0	x	x
rd_miss_wr	1	1	0	0	0	0	1	1	0	1	1	0	1	0	0	1
rd_miss_rd	1	0	1	0	0	0	x	1	1	0	0	x	0	0	x	x
rd_miss_send	0	0	0	1	0	0	x	1	1	0	0	x	0	0	x	x
wr_init	1	0	0	0	0	0	x	0	x	x	1	1	1	0	1	x
wr_hit	1	0	0	0	0	0	0	1	0	0	0	x	0	0	x	x
wr_miss	1	0	0	0	0	0	x	0	x	x	0	x	0	0	x	x
reset	0	0	1	0	0	0	x	0	x	x	0	x	x	1	0	0

Table 3 shows the possible state transitions, based on the current state and inputs.

Table 3: State Transition Table

Curr state	Inputs						Next state
	cpu_rd_wrn	cpu_start	cpu_reset	count1	count2	hit_miss	
idle	1	1	0				rd_init
idle	0	1	0				wr_init
idle	x	x	1				reset
idle	x	0	0				idle
rd_init			0			0	rd_miss_mem_enable
rd_init			0			1	rd_hit
rd_init			1			x	reset
rd_miss_mem_enable			0				rd_miss_mem_wait
rd_miss_mem_enable			1				reset
rd_miss_mem_wait			0	0			rd_miss_mem_wait
rd_miss_mem_wait			0	1			rd_miss_wr
rd_miss_mem_wait			1	x			reset
rd_miss_wr			0		0		rd_miss_wr
rd_miss_wr			0		1		rd_miss_rd
rd_miss_wr			1		x		reset
rd_miss_rd			0				rd_miss_send
rd_miss_rd			1				reset
rd_miss_send			0				idle
rd_miss_send			1				reset
rd_hit			0				idle
rd_hit			1				reset
wr_init			0			0	wr_miss
wr_init			0			1	wr_hit
wr_init			1			x	reset
wr_miss			0				idle
wr_miss			1				reset
wr_hit			0				idle
wr_hit			1				reset
reset	1	1	0				rd_init
reset	0	1	0				wr_init
reset	x	x	1				reset
reset	x	0	0				idle

1 Bit Cache Cell

The single bit cell is built using a modified DFF, transmission gate, and a specialized decoder. Each of these parts are required for the operation of the cell as it is defined.

The modified DFF is used as the single bit storage system, with a write enable signal as its chip enable and a constant tie low for its reset.

The transmission gate controls whether or not the data bit is being read from or not.

The decoder selects if the cell should be reading or writing, and thus affects the output of the transmission gate and input of the DFF.

4x4 Byte Cache Block

The 4x4 Byte cache block stores all 16 Bytes of data through the use of 4 rows of 4 bytes of 1 bit cache cells.

This top level module for the cache takes the data byte and decodes other input signals to determine a write or read.

The data given to this system is then decoded and passed to the specific row to further parse the data. This is done through passing signals through to lower modules within this top level one.

Once the specified row is chosen that module gets the specific offset to either read or write from where it gets passed down into the specific byte and then cells themselves.

VHDL Code

The source code for the project is located here, in the src directory.

Simulations

The following sections include waveforms for the major components of the cache. For each one, we used the top level testbench provided by the TA and Professor. This test shows full functionality of the design and includes each of the four major scenarios (read miss, read hit, write miss, write hit). To test the state machine and cache block, we replaced the chip's signals with signals specific to that module.

The testbench vhd file and input and output text files are located here.

Chip

Figure 1 shows the waveforms for the top-level chip. These results match the pdf that was provided near the beginning of the project.

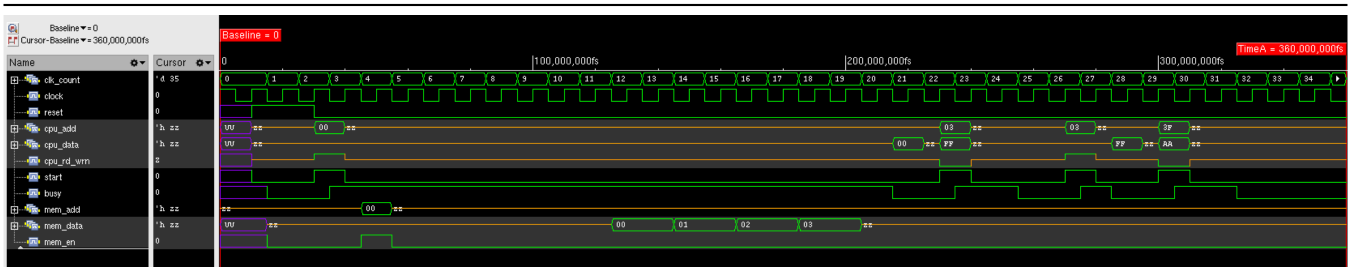

Figure 1: Testbench Waveforms for Chip

State Machine

Figure 2 shows the waveforms for the state machine. All the inputs and outputs are shown, along with the current state.

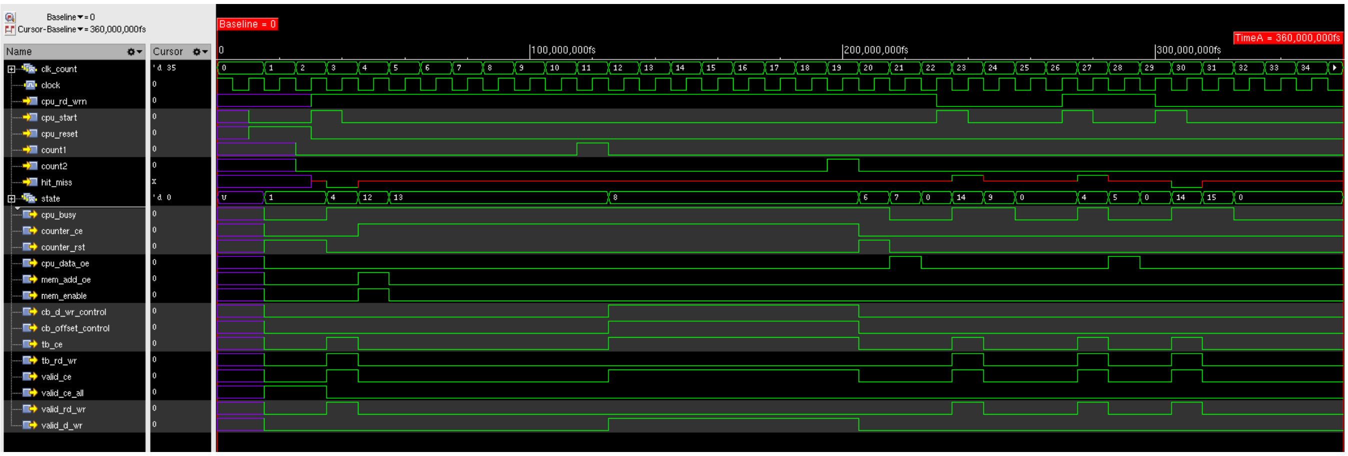

Figure 2: Testbench Waveforms for State Machine

Cache Block

Figure 3 shows the waveforms for the cache block. This shows what happens when the cache is written to or read from.

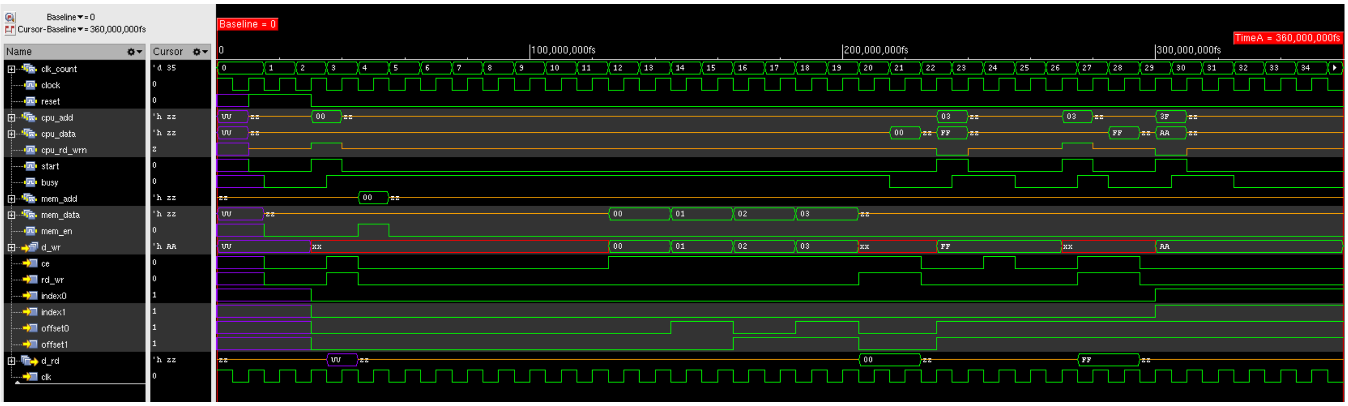

Figure 3: Testbench Waveforms for Cache Block

Work Breakdown

Breakdown of commits to the repo are listed here. This is a chronicle of all changes and updates that each person did over the course of the development of this project. Looking deeper into the commit history shows a list of all changes that were pushed to the repo and from which user. Clicking on any of the commits will show which files were changed, added, removed, or moved.

In terms of lines written by each person, more were written in Dan's commits due to his dealing with longer files, whereas Mick spent more time on creating more smaller low-level files that were used throughout the porject. The overall amount of code used in the final version of this VHDL library is fairly even with a similarly even split of the workload. Lastly, writing of the documentation for the project was done simultaneously by both team members through the use of live coding.

Moving forward, the plan for the layouts is to split up time on the primitives and then each work with the modules that we are most familiar with. This should result in an even time spent on the layouts as well as the VHDL.

Conclusion

This project has taught many different skills and tools to be used later in both of our careers. From learning more simple things such as Git and software control structures, to more specific to this class with VHDL and hierarchical design. We expect this to continue with the second half of the project with creating the layouts of the files and systems we designed to eventually have a fully functional cache system.

With all of this in mind this project has been a success in both learning and applying the topics learned in class as well as applicable to our careers as we prepare for our time after graduation.