OpenCache: An Open-Source OpenRAM Based Cache Generator

Eren Dogan
Ozyegin University
Istanbul, Turkey 34794
eren.dogan@ozu.edu.tr

H. Fatih Ugurdag
Ozyegin University
Istanbul, Turkey 34794
fatih.ugurdag@ozyegin.edu.tr

Matthew R. Guthaus
University of California Santa Cruz
Santa Cruz, CA 95064
mrg@ucsc.edu

***Abstract*—Hardware caches are used in order to maximize the average latency of large memory blocks. The client logic is usually a CPU core but it may well be an application specific logic. However, designing a cache manually from scratch is difficult. In this paper, we describe OpenCache, an open-source parameterized IP core generator. OpenCache calls OpenRAM on the fly, while considering OpenRAM efficiency issues. The current version of OpenCache supports a single pipelined and in-order read-write port on the client side. It outputs a synthesizable Verilog module for cache logic and configuration files for OpenRAM to compile internal SRAM blocks holding data and tags of the generated cache.**

I. Introduction

Processing units in computer hardware can be designed to run much faster than memory units, thanks to hardware techniques such as instruction pipelining and branch prediction. Since processing units are limited by memory frequency, sophisticated chip designs include hardware caches in order to decrease memory access delays. Caches are quite fast compared to especially Dynamic Random Access Memories (DRAMs); however, they are complex, expensive, and hard to design.

There are not many free and open-source tools, which also support open-source Static Random Access Memory (SRAM) compilation, available for designers that do not have enough time and funding to focus on designing a custom cache. There have been some studies on cache generators in the past. [1] focused on FPGA deployment and [2] is a "lock cache" generator addressing multi-core synchronization problem. Neither of them considers SRAM compilation and they are not available for designers. Therefore, we propose OpenCache with the aim to make designing caches faster and easier for research and industry while enabling tape-out of these caches using the OpenRAM memory compiler [3].

OpenCache is a generic hardware generator, which can be further improved with new features. It builds on the OpenRAM project [3], an open-source SRAM compiler available on GitHub [4]. OpenCache takes the specifications of a cache design as a configuration file input and generates configuration files for OpenRAM to compile the internal SRAMs of the cache. It also generates a synthesizable Verilog module for the cache logic using the nMigen library [5].

II. Architecture

In this section, some variables are used to explain the architecture of OpenCache. These variables are defined in Table I, and some of the variables on the right side of equations are taken as parameters in the configuration file, which are defined in Table IV of the Implementation section.

TABLE I
VARIABLES IN THE OPENCACHE.

Variable	Equation
line_size	word_size * words_per_line
row_size	line_size * num_ways
num_rows	total_size/row_size
num_masks	word_size/write_size
offset_size	$\log_2$(words_per_line)
set_size	$\log_2$(num_rows)
tag_size	address_size − set_size − offset_size

Input and output ports of a cache are illustrated in Figure 1, and pins in the CPU interface and DRAM interface are shown in Tables II and III, respectively. In this paper, we are referring to a CPU and DRAM; however, OpenCache is not specific to them. OpenCache can also be used for caches within an ASIC. It can also be used for large SRAMs or it can interface to a higher level cache instead of DRAM.

TABLE II
CPU INTERFACE OF THE CACHE.

Pin	Size	Direction	Description
clk	1	Input	System clock
rst	1	Input	Reset
flush	1	Input	Flush
csb	1	Input	Chip select
web	1	Input	Write enable
wmask	num_masks	Input	Write mask
addr	address_size	Input	Address
din	word_size	Input	Data input
dout	word_size	Output	Data output
stall	1	Output	Pipeline stall

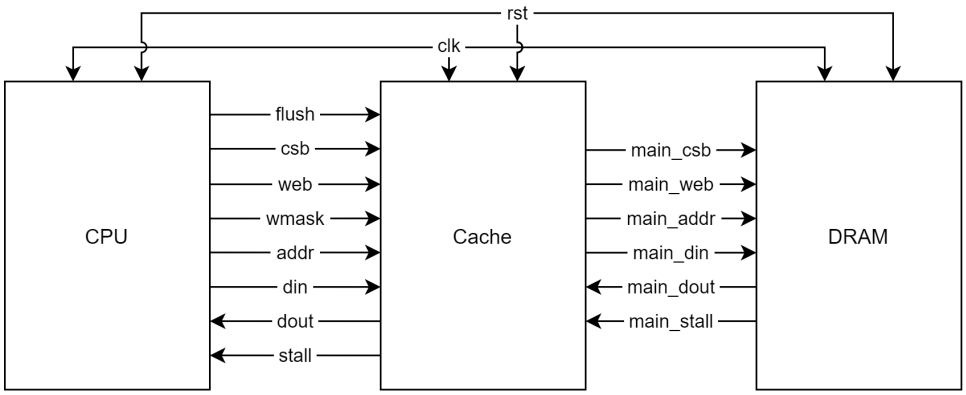

Fig. 1. Ports of a cache.

TABLE III
DRAM INTERFACE OF THE CACHE.

Pin	Size	Direction	Description
main_csb	1	Output	Chip select
main_web	1	Output	Write enable
main_addr	address_size	Output	Address
main_din	line_size	Output	Data input
main_dout	line_size	Input	Data output
main_stall	1	Input	Stall

csb and web pins are active low since SRAMs generated by OpenRAM also have active low csb and web. Size of the dout pin depends on the return_type parameter. It is word_size by default; however, some caches such as L2 and L3, return a whole data line. In this case, size of the dout pin is line_size. Returning a data line is not yet implemented in OpenCache but will be available soon.

The address pin of the cache consists of 3 parts: tag, set, and offset. Offset is used to select a word in a cache line. Set is used to find the row (set) of data array, which has the requested address' data. Tag is used to identify an address in the set. It is used in tag comparison to find whether the request is a hit or miss.

Address		
Tag	Set	Offset
tag_size	set_size	offset_size

OpenCache has four parameters which together decide on the features of the cache generated. These parameters are is_data_cache, num_ways, replacement_policy, and write_policy.

A. Cache Type

The cache type is chosen with the is_data_cache parameter. A data cache is generated if is_data_cache is true; otherwise, an instruction cache is generated. Instruction caches do not have write operation. CPUs can fetch instruction and data words in parallel if it has two separate caches for instruction and data. This way, the performance of the design can be improved.

Only data caches can be generated using OpenCache at the moment; however, instruction caches will be available in the future.

B. Associativity

The associativity of a cache is chosen with the num_ways parameter. Each way has its own entry in the tag and data arrays, which have the following structures.

	Tag Way			Data Way		
Set 1	V	D	Tag	Word Y	...	Word 1
Set 2	V	D	Tag	Word Y	...	Word 1
...	...					
Set X	V	D	Tag	Word Y	...	Word 1

V stands for "valid bit", which shows whether the way has valid data. If it is low, the way is empty and new data can be placed in this way. D stands for "dirty bit" and shows whether the data in the way has been modified. If a way is dirty, it needs to be written back when cache miss occurs or during flush. X is equal to num_rows and Y is equal to words_per_line. In OpenCache, there are 2 associativities implemented.

1) Direct-mapped Cache: If num_ways is 1, the cache generated by OpenCache will be a direct-mapped cache. Direct-mapped caches have only 1 way for data placement. If a cache miss occurs, the data in the set, which corresponds to the address, is replaced.

2) N-way Set Associative Cache: If num_ways is more than 1, the cache generated by OpenCache will be an N-way set associative cache. Set associative caches have multiple ways for data placement. If a cache miss occurs, a way in the set corresponding to the address is replaced. The way to be evicted is chosen according to the replacement policy of the cache. This type of caches have N data arrays for each way since combining all ways in a single array can cause problems for large word_size and words_per_line parameters.

C. Replacement Policy

The replacement policy of a cache is chosen with the `replacement_policy` parameter. Replacement policies select the way of data to replace after a cache miss. There are 3 different replacement policies implemented in OpenCache.

1) First In First Out (FIFO): In FIFO replacement, the set to be evicted is chosen according to a queue of placement. The data that has entered the queue the first gets evicted first. When a cache uses FIFO replacement policy, it has an SRAM array for FIFO pointer numbers.

There is a FIFO pointer number for each set in the cache, which are used to decide the way to replace. These pointers start from zero and increase each time a data is placed in their corresponding set. When the cache places a data, it selects the way that is pointed by the FIFO number of that set.

2) Least Recently Used (LRU): In LRU replacement, the set to be evicted is chosen according to an order of access. The data that is accessed earliest in time gets evicted first. When the cache uses LRU replacement policy, it has an SRAM array for LRU numbers. There are `num_ways` many LRU numbers for each set in the cache, which are used together to decide the way to replace.

Every time a way is used, its corresponding LRU number is brought to the top of the order (maximum value) and all the other LRU numbers, which are greater than its previous value, are decreased by one. Every time a data is replaced, the way that corresponds to the LRU number, which is equal to zero, is chosen by the algorithm.

3) Random Replacement (RR): In random replacement, the set to be evicted is chosen randomly. The cache has a register which acts like a counter incremented by 1 at every positive edge of the system clock. When the cache needs to evict a way, it is chosen according to the counter register. Since there is no guarantee when the cache will need to evict data or how long it takes for DRAM to return the requested data, this approach essentially replaces data randomly.

D. Write Policy

The write policy of a cache is chosen with the `write_policy` parameter, and they decide how to perform the write operation on cache data. Only "write-back policy" is currently implemented in OpenCache. "Write-through policy" will be available in the future.

1) Write-back: When a write request comes to a write-back cache, it writes the data to its internal SRAM. Since the internal SRAM is not synchronized with DRAM, DRAM's data may be old. When the modified data needs to be evicted, which is either a flush or cache miss, it is written back to the DRAM before overwriting the dirty data.

2) Write-through: In situations where atomicity is critical such as banking, write-through caches are used in order to provide a more reliable and secure system. When a write request comes to a write-through cache, it writes the data to the DRAM immediately.

E. Pipeline

OpenCache generates in-order pipelined caches, which help provide better CPU performance. When a pipelined cache is returning the data requested by the CPU, it also reads the corresponding lines from its internal SRAMs for the next request of the CPU. However, pipeline can also bring data hazard if SRAMs are not read-after-write.

F. Data Hazard

OpenCache has a parameter named `data_hazard` to enable data hazard control. Since OpenCache generates pipelined caches, when reading a line from the internal SRAMs, old data may be received if there is a write to the same line at the same cycle. If `data_hazard` parameter is true, generated caches avoid causing this kind of data hazard by entering a stall state.

Some standard cell libraries might have bitcells that can cause data hazard since they are not read-after-write. However, if users of OpenCache can guarantee that their standard cell library is going to be data hazard proof, they can set `data_hazard` to false and generate the cache accordingly.

OpenCache does not support cache coherence, which ensures data uniformity across multiple cores via communication between caches that use the same shared memory. There can be data hazard in a system where multiple processing units access the same memory address and have different values because of cache incoherence.

III. Implementation

OpenCache is implemented in Python and is available on GitHub [6]. The structure of the generator is similar to OpenRAM's structure for convenience. OpenCache uses nMigen library [5], which is a Python toolbox for hardware design. It takes a Python file as input which is supposed to be the configuration file including the parameters of the cache to be generated. All OpenCache parameters are listed in Table IV.

TABLE IV
PARAMETERS OF OPENCACHE.

Parameter	Description
`total_size`	Size of the data array
`word_size`	Size of a machine word
`words_per_line`	Number of words per line
`address_size`	Size of the address pin
`write_size`	Size of data for a write mask bit
`num_ways`	Number of ways of the cache
`replacement_policy`	Replacement policy of the cache
`write_policy`	Write policy of the cache
`is_data_cache`	Data or instruction cache
`return_type`	Return either a word or line
`data_hazard`	If SRAMs are not read-after-write
`simulate`	Enable simulation (disabled by default)
`synthesize`	Enable synthesis (disabled by default)

The generator is used by running the $opencache.py$ file with a Python interpreter. $opencache.py$ accepts parameters, instantiates design modules, performs design verification, and saves output files. The $cache$ class decides and instantiates a specific design module according to the associativity and replacement policy of the desired cache. Logic blocks in the synthesizable Verilog are implemented in $block$ classes using the nMigen library.

In addition to parameterization, OpenCache supports design verification for regression testing. This feature of OpenCache makes it possible to detect bugs and verify new features. OpenCache uses FuseSoC [7] in order to use EDA tools for simulation and synthesis of generated caches. Currently, Icarus [8] is the supported tool for simulation and Yosys [9] is the supported tool for synthesis. Support for more EDA tools can be added for verification in the future, thanks to FuseSoC's support for many EDA tools.

IV. Conclusion and Future Work

Our work introduces an open-source cache generator called OpenCache. OpenCache generates synthesizable Verilog files for the cache logic and builds on top of the OpenRAM project of Guthaus et al. [3] to compile the SRAMs, which are used inside the generated cache.

The OpenCache project is implemented to enable easier and faster custom cache design for research. Our intention is to make OpenCache generic and flexible so that new features can be added in the future. Some missing features are already expressed in this paper. We are actively developing OpenCache to implement the missing and new features.

OpenCache does not take into account the read-write race conditions that might occur (depending on the DRAM used). This problem may be addressed in the future by adding a control mechanism to OpenCache.

Currently, OpenCache is an in-order memory, meaning that caches respond to requests in the same order they arrive. OpenCache can be improved to support out-of-order execution for better performance. Since OpenCache generates single port memory, multi port cache support can also be added in the future.

References

[1] P. Yiannacouras and J. Rose, "A Parameterized Automatic Cache Generator for FPGAs," Proceedings of the International Conference on Field-Programmable Technology (FPT), pp. 324-327, 2003.

[2] B. E. S. Akgul and V. J. Mooney, "PARLAK: Parametrized Lock Cache Generator," Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), pp. 1138-1139, 2003.

[3] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "OpenRAM: An Open-Source Memory Compiler," Proceedings of the International Conference on Computer-Aided Design (ICCAD), pp. 1-6, 2016.

[4] M. R. Guthaus, "OpenRAM," GitHub. [Online]. Available: https://github.com/VLSIDA/OpenRAM. [Accessed: 29-Aug-2021].

[5] "nMigen," GitHub. [Online]. Available: https://github.com/nmigen/nmigen. [Accessed: 29-Aug-2021].

[6] "OpenCache," GitHub. [Online]. Available: https://github.com/VLSIDA/OpenCache. [Accessed: 29-Aug-2021].

[7] O. Kindgren, "FuseSoC," GitHub. [Online]. Available: https://github.com/olofk/fusesoc. [Accessed: 29-Aug-2021].

[8] S. Williams, "Icarus Verilog," GitHub. [Online]. Available: https://github.com/steveicarus/iverilog. [Accessed: 29-Aug-2021].

[9] C. X. Wolf, "Yosys," GitHub. [Online]. Available: https://github.com/YosysHQ/yosys. [Accessed: 29-Aug-2021].