Computer Architecture

Caches

Filipe Salgado, PhD Student @ ESRG

R&D Centre ALGORITMI
Department of Industrial Electronics
School of Engineering,
University of Minho, Guimarães - PORTUGAL

Cache

Embedded Systems
Research Group

- Overview
- Introduction
- What to cache?
- Logical Organization
- Cachegen
- Cache implementation "from scratch"

Overview

- Introduction
  - Why to Cache?
  - Memory Hierarchy
- What to cache?
  - Locality of Reference
- Logical Organization
  - Concepts, words, lines, sets, tags
  - Types of Caches (organizational models)

Overview

Embedded Systems Research Group

- Cachegen
  - Caches generation
  - Generate RTL in Quartus II
- Cache implementation "from scratch‟
  - Direct mapped cache
  - Metrics and projection
  - Storage structures
  - test-bench

Cache

Embedded Systems
Research Group

- Overview
- Introduction
- What to cache?
- Logical Organization
- Cachegen
- Cache implementation "from scratch"

Why to cache?

- Why to cache?
    - Memory technology evolves differently from silicon technology
        - Different speeds:
            - DDR SDRAM: **100MHz-266MHz**
            - Pentium: **1.3GHz – 3.8GHz**
        - SDRAM requires **multiple cycles** to access content
    - Memory access is a performance bottleneck in typical computer architectures
    - Cache attenuates that performance difference
        - Storing memory content in a fast access (but area expensive) memory, closer to the processor core.

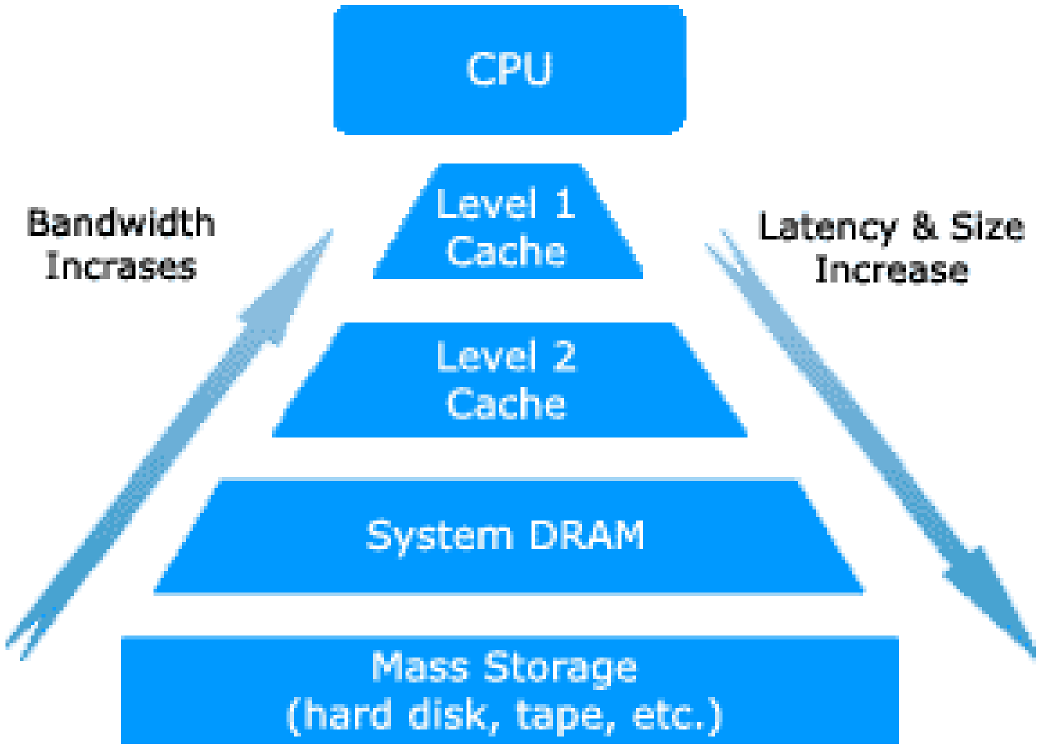
Memory Hierarchy

- Memory Hierarchy
  - Fast caches are very expensive!
    - In Power and area!!!
    - About 20% to 30% of chip Area!
    - So must have reduced storage capacity
  - Solution is to create slower caches with increased storage capacity!
    - Compose a memory hierarchy
      - Cache L1 – fastest, most expensive
      - Cache L2 – fast, less expensive, more storage capacity
      - ?Cache L3?
      - SDRam
      - Mass storage memory

Memory Hierarchy

- Memory Hierarchy
  - Cache L1 – fastest, expensivest
  - Cache L2 – fast, less expensive, more storage capacity
  - Cache L3 - …
  - SDRam
  - Mass storage memory

- Convenience Store Example ☺



http://www.extremetech.com/computing/47920-pc-motherboard-technology/4?print on 23/4/12

Cache

Embedded Systems Research Group

- Overview
- Introduction
- **What to cache?**
- Logical Organization
- Cachegen
- Cache implementation "from scratch"

What to cache?

Embedded Systems Research Group

- Data caching is done according to straight information management policies
  - How to decide what should be cached?

- Locality of Reference:
  - Principle of Temporal Locality
  - Principle of Spatial Locality

Locality of Reference

Embedded Systems Research Group

- Principle of Temporal Locality
  - "If a particular memory location at one point is required, it is very likely that it will be necessary again in a near future."
    - Convenience store example ;)
      - If you go there to buy eggs and they don't have, when they go to the warehouse, they'll get more than one box.
    - Usage ex.: Data memory (variables in loop cycles)

Locality of Reference

- Principle of Spatial Locality
  - "If a particular memory location at one point is required, it is very likely that the nearby locations will be referenced too in a near future."
    - Convenience store example ;)
      - If you go there to buy potatoes and they are out of stock, when they go to the market, they'll take advantage of the trip and also buy carrots, tomatoes, etc…
    - Usage ex.: Instructions stored in Code Memory

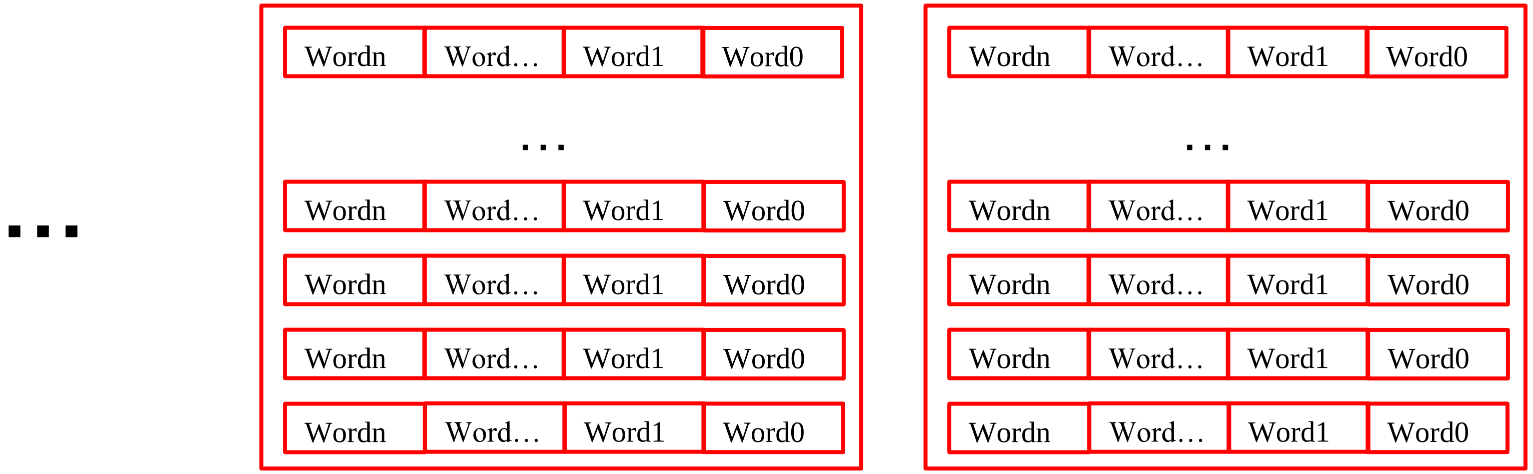
Cache

Embedded Systems Research Group

- Overview
- Introduction
- What to cache?
- Logical Organization
- Cachegen
- Cache implementation "from scratch"

Logical Organization

- How data is organized in cache?
  - Words
  - Lines
  - Sets



Logical Organization

- How to know **where** is **what**?

  (considering direct mapped and set-associative)

  - Addresses are stored in a dedicated memory - **tag memory**
  - Some part of the address is used to address the Tag memory – **line address**
  - Another part of the address is used to address the word – **word address**

(further explanations ahead)

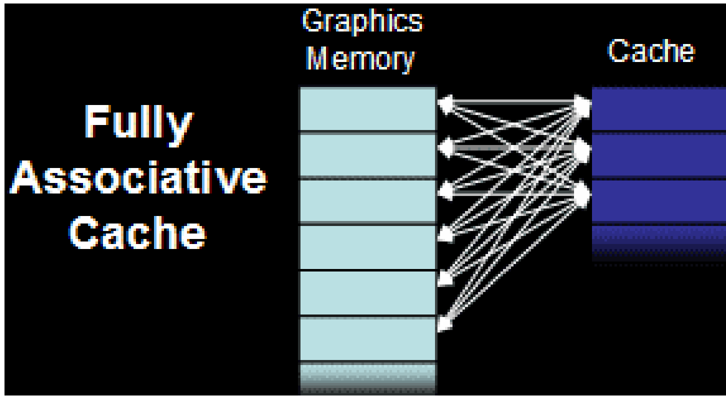
Types of cache

Embedded Systems
Research Group

- Some organizational models are used:
  - **Fully associative**
  - **Direct mapped**
  - **Set associative**
  - **Hyper associative**

Types of cache

- Fully associative:
  - Data is stored in the cache sets according with the access order.
    - Any memory content can be anywhere in the set
    - Required a bigger tag memory
  - Uses a CAM (content addressable memory)
    - Reverse memory: user provides the content, it gives the address where it is stored
  - Address is stored in the CAM
    - After read the cam user knows if requested location is available or not.

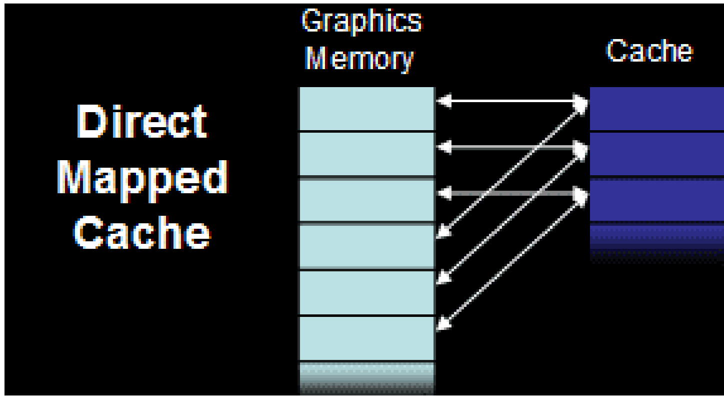


http://www.xbitlabs.com/articles/graphics/display/radeon-x1000_6.html on 23/4/12

Types of cache

Embedded Systems Research Group

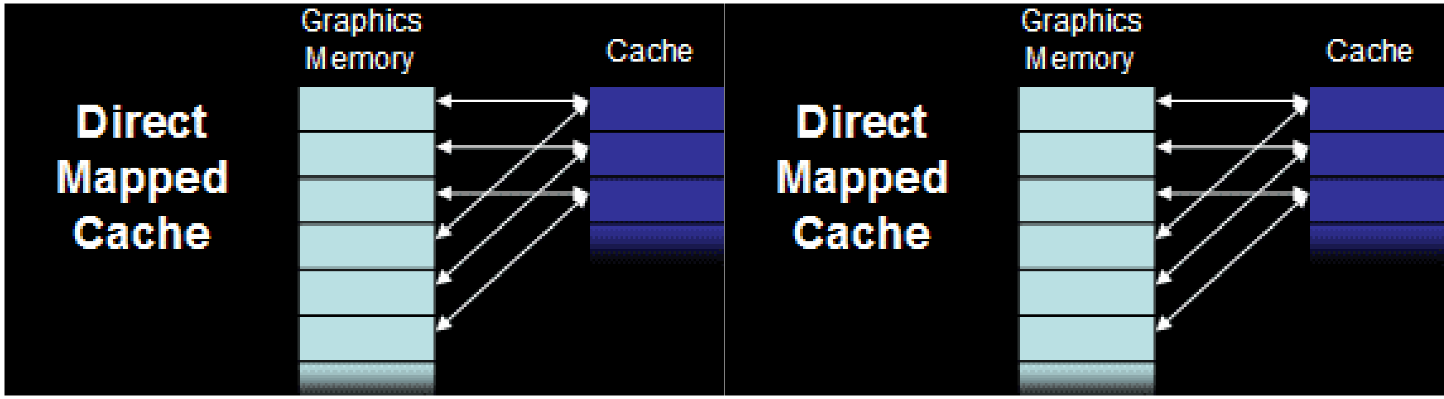
- Direct Mapped:
  - Easier to implement
  - Each cache line corresponds to a subset of memory positions
    - Smaller tag memory



http://www.xbitlabs.com/articles/graphics/display/radeon-x1000_6.html on 23/4/12

Types of cache

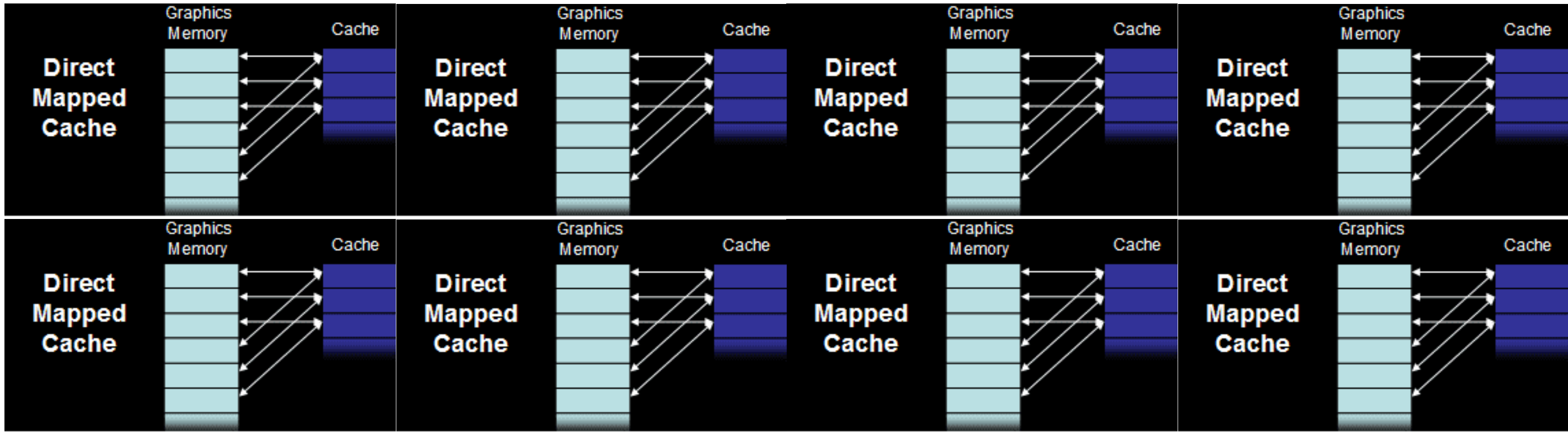
- N-way set associative (typically N is 2,4,8 sets):
  - Same principle of direct mapped cache
  - Associativity in powers of 2!
  - Implemented with the replications of groups of cache set+tag memory



http://www.xbitlabs.com/articles/graphics/display/radeon-x1000_6.html on 23/4/12

Types of cache

- Hyper-associative (typically 16,32,64,128,… sets):
  - Its an implementation of a N-way set associative with a huge number of small sets
  - Common in super scalar implementations
  - Increased number of sets to increase hit rate in applications with several small data work sets



http://www.xbitlabs.com/articles/graphics/display/radeon-x1000_6.html on 23/4/12

Cache

Embedded Systems Research Group

- Overview
- Introduction
- What to cache?
- Logical Organization
- Cachegen
- Cache implementation "from scratch"

Cachegen

Embedded Systems Research Group

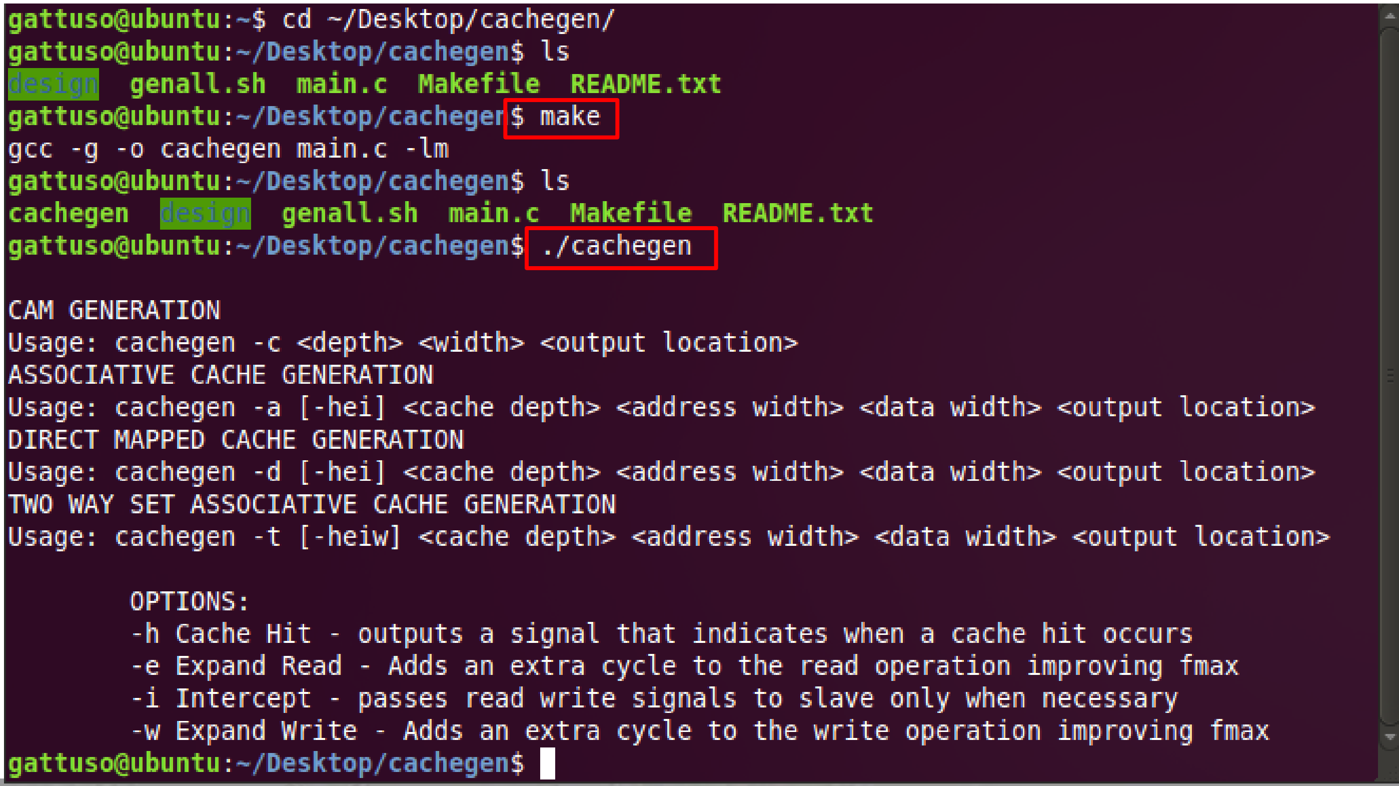
- It's a software developed by
  - Peter Yiannacouras for partial fulfillment of **BASc** at the University of Toronto
  - For Linux

- Simple cache generator, able to generate synthesizable **Verilog** code for **associative**, **direct mapped** and **2way set associative** caches.
- Generates only "one word per line" cache sets
- Useful for educational purposes!
  - Lets try it ;)

Cachegen

- Go to Linux
- Extract the tarball

Run the following commands:

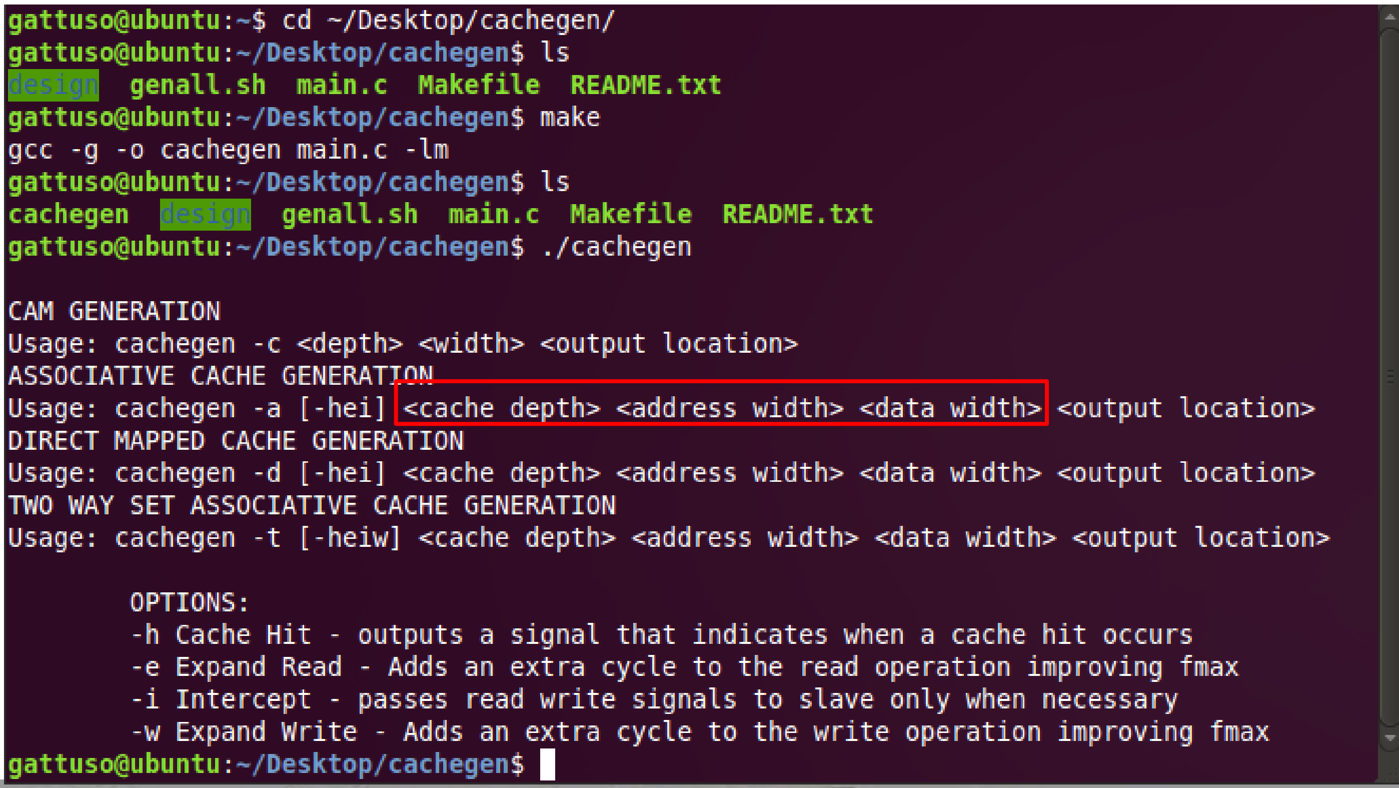


```
gattuso@ubuntu:~$ cd ~/Desktop/cachegen/
gattuso@ubuntu:~/Desktop/cachegen$ ls
design  genall.sh  main.c  Makefile  README.txt
gattuso@ubuntu:~/Desktop/cachegen$ make
gcc -g -o cachegen main.c -lm
gattuso@ubuntu:~/Desktop/cachegen$ ls
cachegen  design  genall.sh  main.c  Makefile  README.txt
gattuso@ubuntu:~/Desktop/cachegen$ ./cachegen

CAM GENERATION
Usage: cachegen -c <depth> <width> <output location>
ASSOCIATIVE CACHE GENERATION
Usage: cachegen -a [-hei] <cache depth> <address width> <data width> <output location>
DIRECT MAPPED CACHE GENERATION
Usage: cachegen -d [-hei] <cache depth> <address width> <data width> <output location>
TWO WAY SET ASSOCIATIVE CACHE GENERATION
Usage: cachegen -t [-heiw] <cache depth> <address width> <data width> <output location>

        OPTIONS:
        -h Cache Hit - outputs a signal that indicates when a cache hit occurs
        -e Expand Read - Adds an extra cycle to the read operation improving fmax
        -i Intercept - passes read write signals to slave only when necessary
        -w Expand Write - Adds an extra cycle to the write operation improving fmax
gattuso@ubuntu:~/Desktop/cachegen$ 
```

Cachegen

Embedded Systems Research Group

- Cache depth?
- Address width?
- Data width?



```
gattuso@ubuntu:~$ cd ~/Desktop/cachegen/
gattuso@ubuntu:~/Desktop/cachegen$ ls
design  genall.sh  main.c  Makefile  README.txt
gattuso@ubuntu:~/Desktop/cachegen$ make
gcc -g -o cachegen main.c -lm
gattuso@ubuntu:~/Desktop/cachegen$ ls
cachegen  design  genall.sh  main.c  Makefile  README.txt
gattuso@ubuntu:~/Desktop/cachegen$ ./cachegen

CAM GENERATION
Usage: cachegen -c <depth> <width> <output location>
ASSOCIATIVE CACHE GENERATION
Usage: cachegen -a [-hei] <cache depth> <address width> <data width> <output location>
DIRECT MAPPED CACHE GENERATION
Usage: cachegen -d [-hei] <cache depth> <address width> <data width> <output location>
TWO WAY SET ASSOCIATIVE CACHE GENERATION
Usage: cachegen -t [-heiw] <cache depth> <address width> <data width> <output location>

        OPTIONS:
        -h Cache Hit - outputs a signal that indicates when a cache hit occurs
        -e Expand Read - Adds an extra cycle to the read operation improving fmax
        -i Intercept - passes read write signals to slave only when necessary
        -w Expand Write - Adds an extra cycle to the write operation improving fmax
gattuso@ubuntu:~/Desktop/cachegen$
```

Cachegen

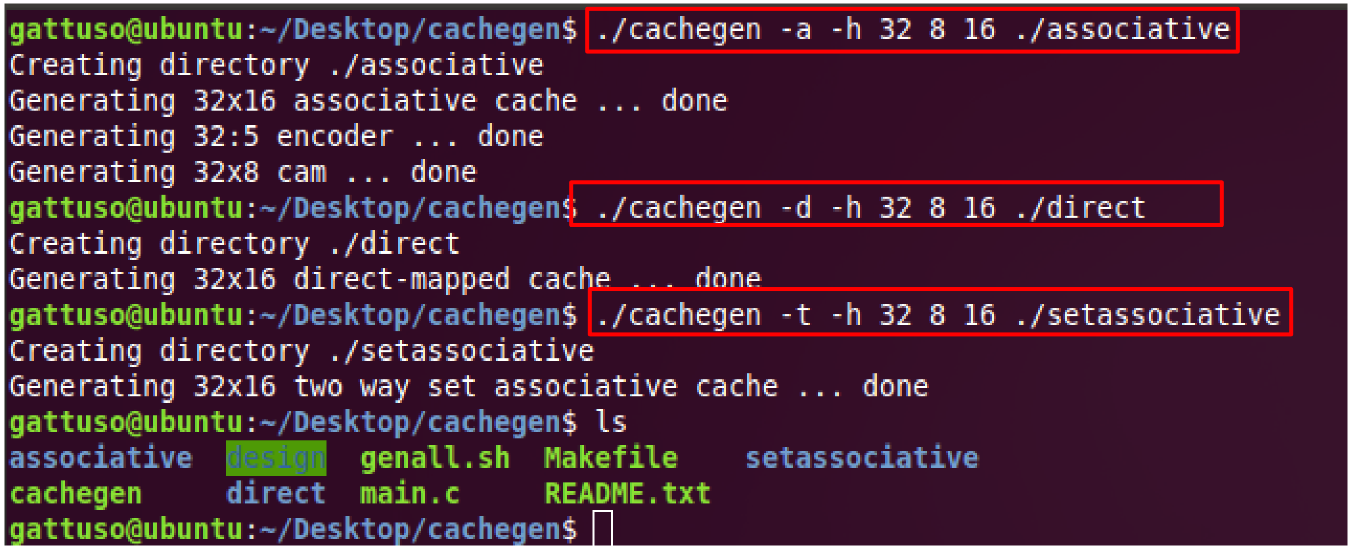
- **Cache depth -** is the number of cache lines present in the cache set and tag memory. In other words is the number of **cache lines**

- **Address width –** is the number of bits available to address memory. It's a consequence of the architecture and relates with the addressable space.

- **Data width –** is the number of bits per cache line. In other words is the **line size**.

Cachegen

- Metrics examples:
  - cache dept 32
  - addr width (address space) 8
  - data width (line size) 16
- **What's the storage capacity of this cache?**
- **What's the addressable space?**



```
gattuso@ubuntu:~/Desktop/cachegen$ ./cachegen -a -h 32 8 16 ./associative
Creating directory ./associative
Generating 32x16 associative cache ... done
Generating 32:5 encoder ... done
Generating 32x8 cam ... done
gattuso@ubuntu:~/Desktop/cachegen$ ./cachegen -d -h 32 8 16 ./direct
Creating directory ./direct
Generating 32x16 direct-mapped cache ... done
gattuso@ubuntu:~/Desktop/cachegen$ ./cachegen -t -h 32 8 16 ./setassociative
Creating directory ./setassociative
Generating 32x16 two way set associative cache ... done
gattuso@ubuntu:~/Desktop/cachegen$ ls
associative  design  genall.sh  Makefile    setassociative
cachegen     direct  main.c     README.txt
gattuso@ubuntu:~/Desktop/cachegen$ 
```

Cachegen

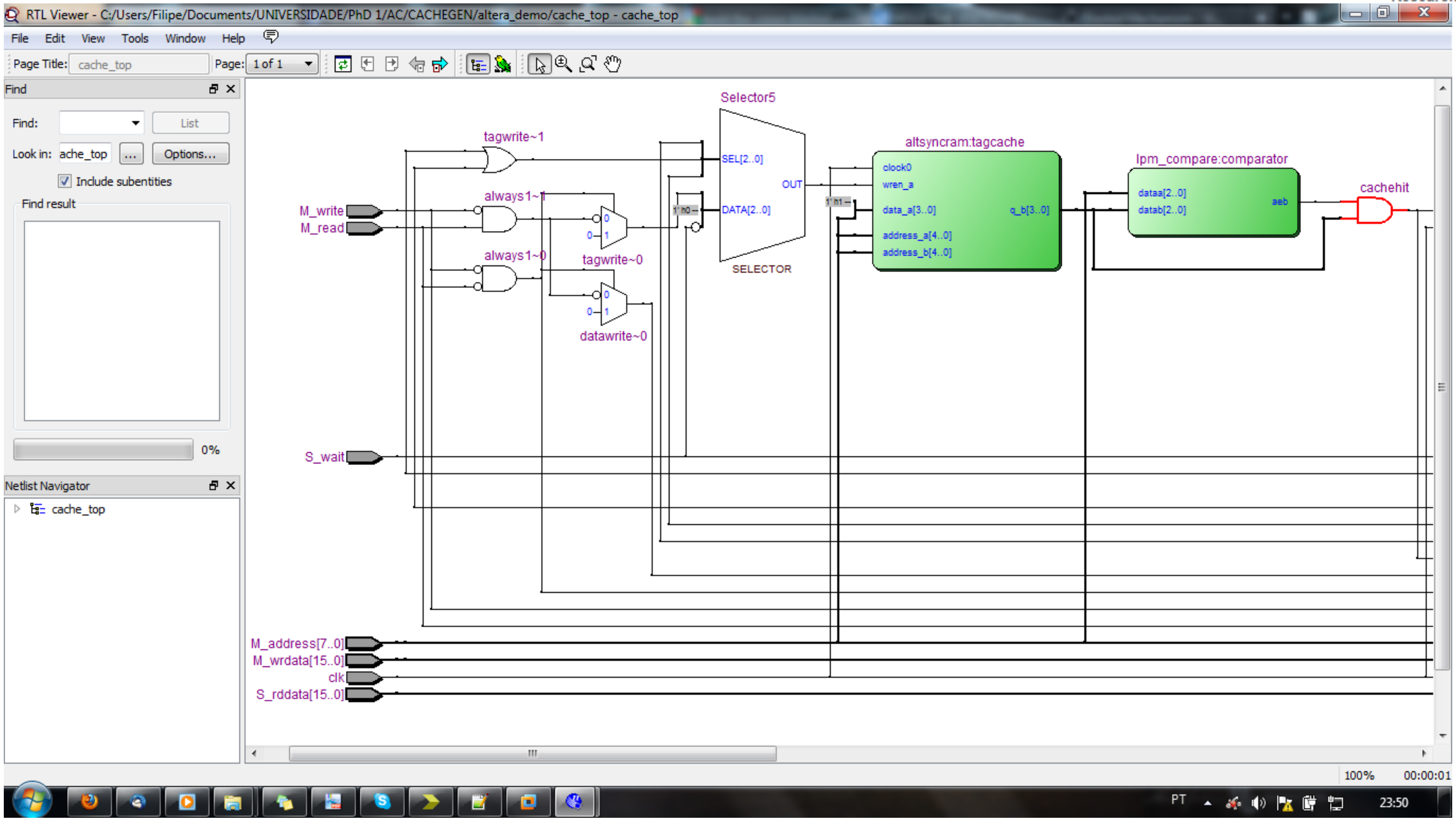
Embedded Systems
Research Group

- Back to windows
  - Copy the Direct mapped Design
  - Create a Vivado Project for Zybo board
  - Include the Verilog file
  - **Synthetise!**

  - **Generate RTL View**

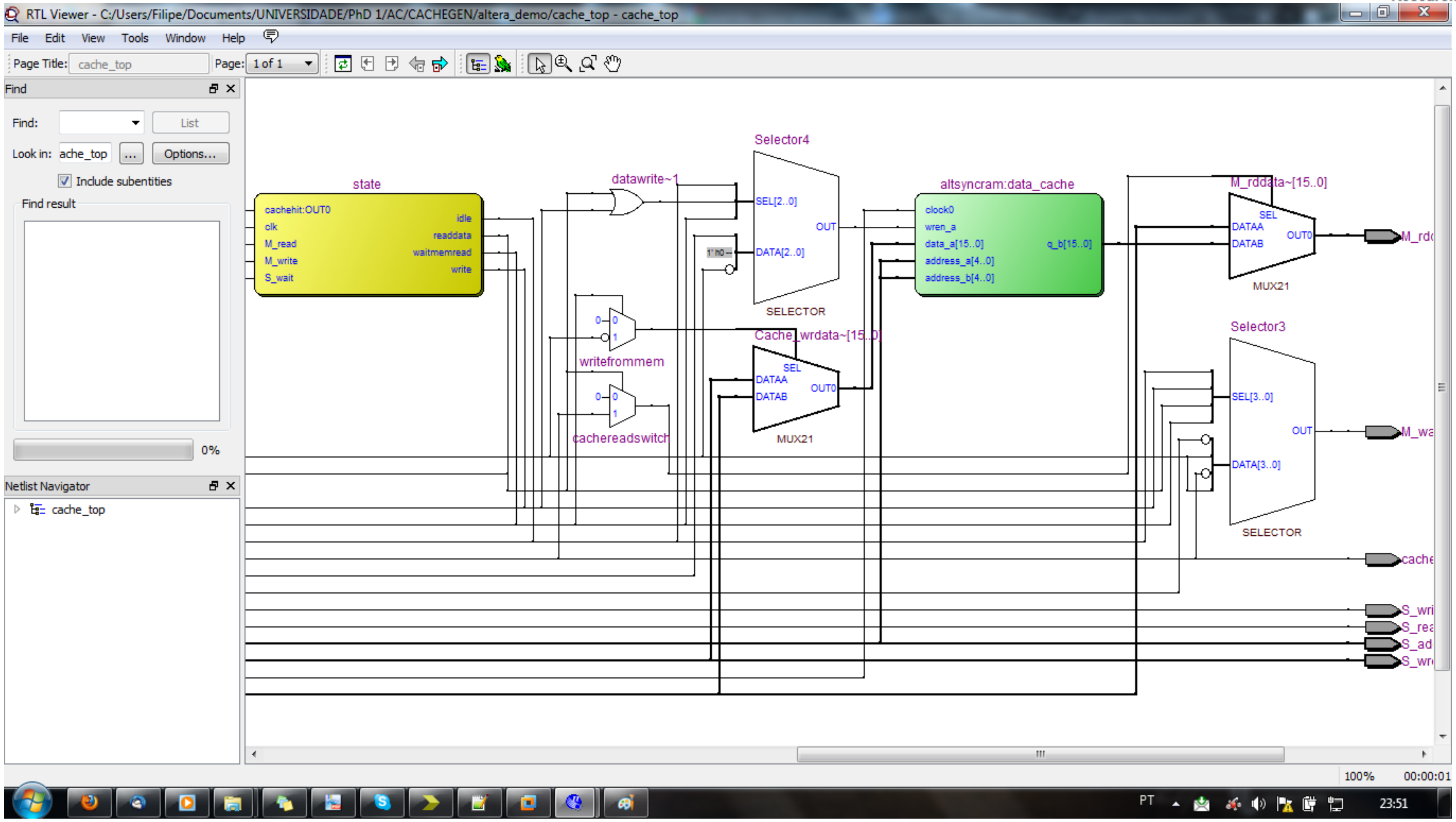
Cachegen

Embedded Systems Research Group



Cachegen

Embedded Systems Research Group



Cache

Embedded Systems Research Group

- Overview
- Introduction
- What to cache?
- Logical Organization
- Cachegen
- Cache implementation "from scratch‟

Working principle basis

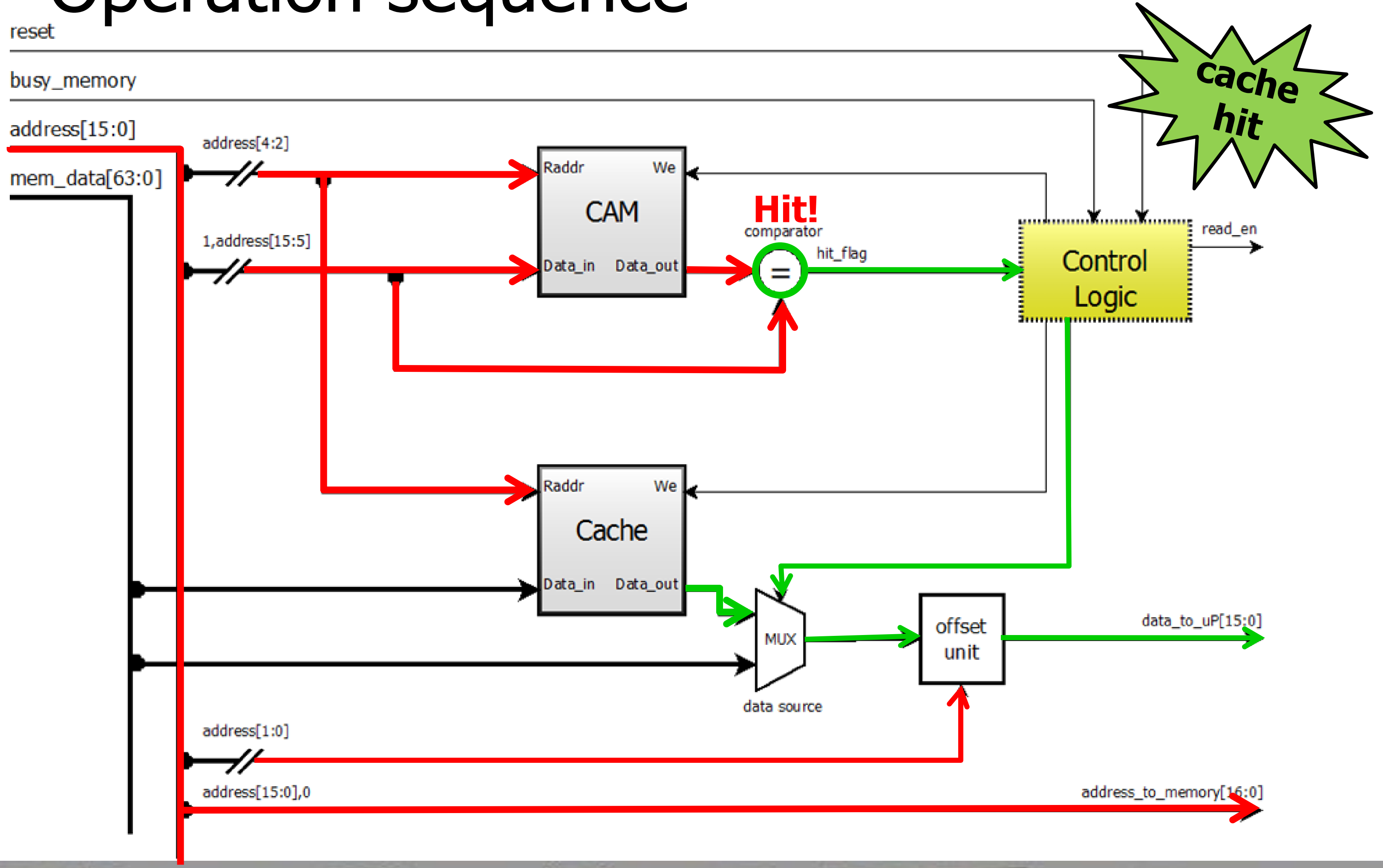
- What about if Cachegen doesn't fit your requirements? (performance, words per line, latency, etc)

- **Lets create a cache from scratch!**

  - **Simplest design: Direct mapped Instruction Cache**
    - What's the main difference between **Instruction** and **Data** caches?
  - **How does it works??**

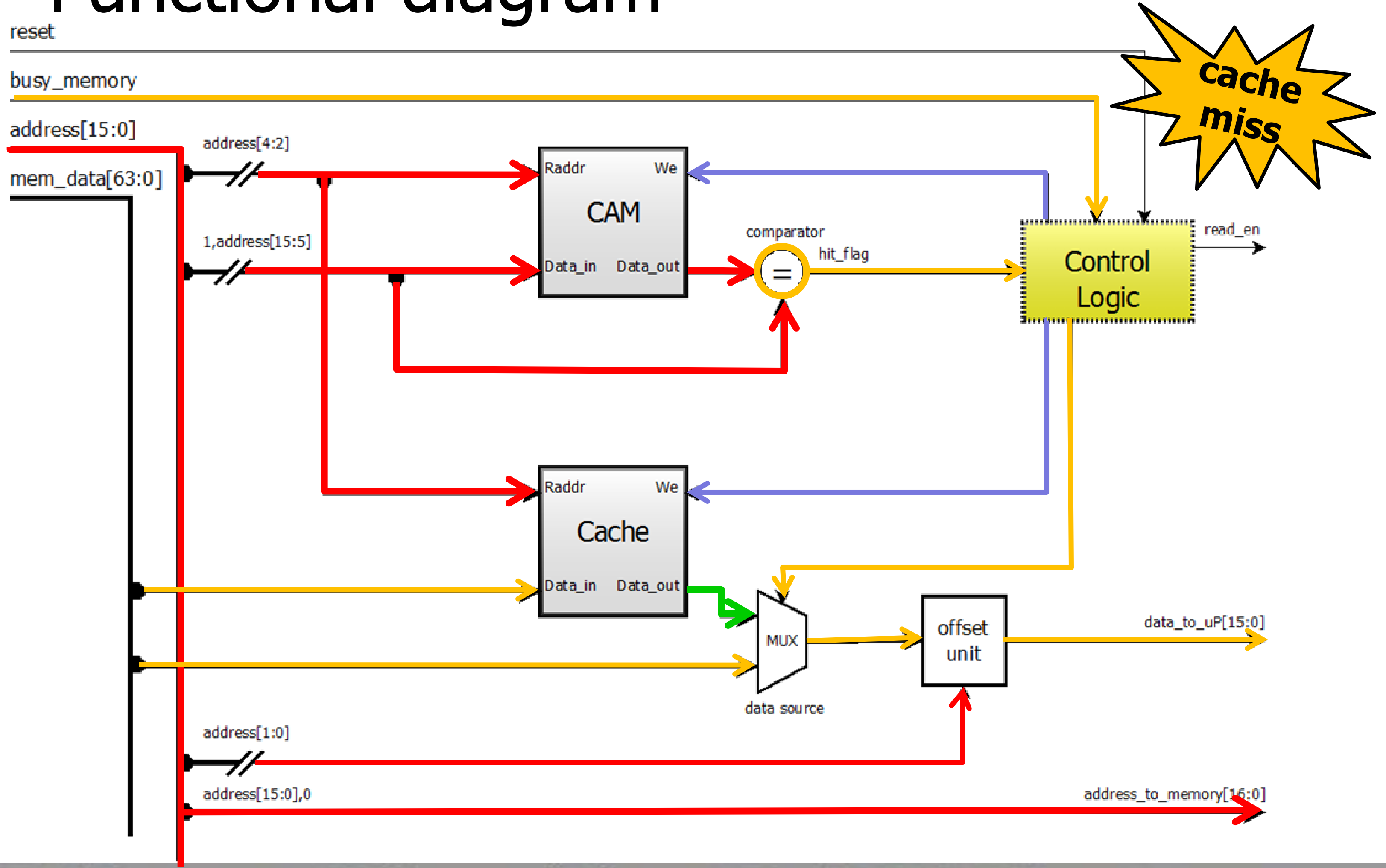
Working principle basis

- Operation sequence



Working principle basis

- Functional diagram



Working principle basis

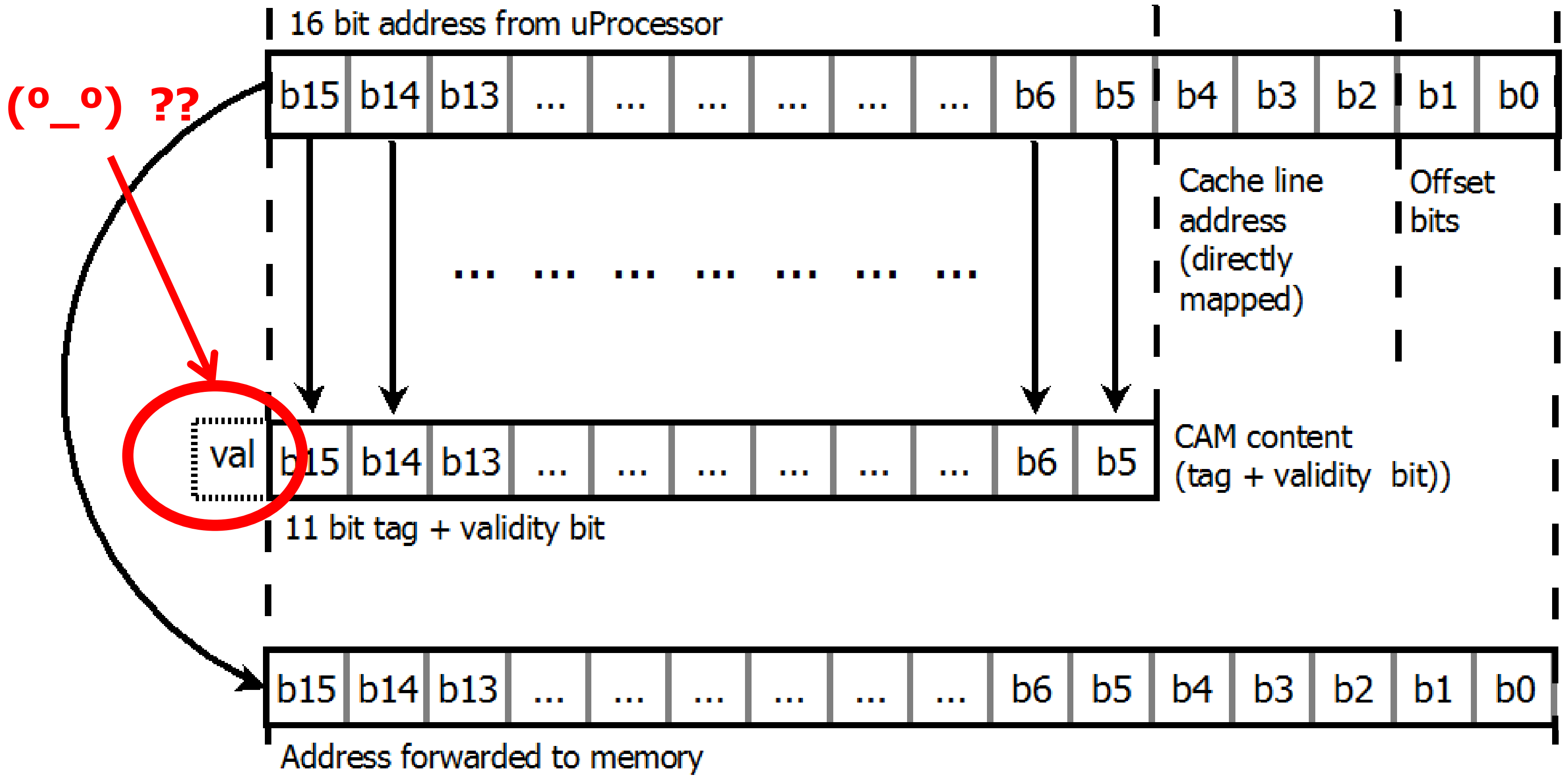
Embedded Systems Research Group

- And what about the **address**? What goes where?

- In multiple words per cache line the **address** must be **split** in
  - Offset , or Word address
  - Line address
  - Tag

Working principle basis

- Address interpretation



16 bit address from uProcessor
b15
b14
b13
...
b6
b5
b4
b3
b2
b1
b0
Cache line address (directly mapped)
Offset bits
(º_º) ??
val
CAM content (tag + validity bit))
11 bit tag + validity bit
Address forwarded to memory

Working principle basis

- Address interpretation
  - Validity bit?
    - Indicates if a cache line is valid or not.
    - How to know if the cache has stored the address 0x0000 or is just empty? -> **VAL bit**
  - **How many words per line?**
  - **How many cache lines?**
  - **Addressable space?**

Working principle basis

Embedded Systems Research Group

- Now on Vivado;)
  - Create a project
  - Include the "ins_cache.v"

  - Metrics:
    - **How to start…**

Metrics

- SDRAM retrieved big pieces of data
    - Usually memory is organized in long lines 256 bit
    - Retrieved 64 or 72 bits at a time
    - Cache lines should take advantage of that
- Lets assume a **256 byte** memory, with **64 bit lines**
- 32 lines
- Requires **5 bit** to address the lines

Metrics

Embedded Systems Research Group

- Cache set metrics
  - 64 bit memory interface
  - 16 bit architecture
  - 64 Bytes of storage

    - 64 bit line
    - 64 bit line/16 bit words = 4 Words per line
    - 64B/(4word*2byte) = 8 lines

Metrics

- Address Division
  - Address length (PC) 8 bit
  - 4 words per line
  - 8 lines

    - 2 bit - word address
    - 3 bit - line address
    - 8 bit address - (2bit+3bit) = 4 tag bits + val!

Implementation

Embedded Systems
Research Group

- Dive into Vivado
  - Analyze and explain
  - Create a Verilog test bench to the project
    - Copy code from “cache_tb”
  - **Run TB!**

Implementation

Embedded Systems
Research Group

- How to adapt for **2-way** set associative
  - N-way set associative is basically a direct mapped cache with replicated storage resources
    - N Cache sets
    - N Tag memories
    - N Cache hit comparators
    - …

Implementation

- How to adapt for **2-way** set associative
  - But where to store the data coming from memory?
  - What's the criteria to select the set?
    - The one which is empty :)
  - And if both addressed lines in all the sets have content?
    - Replacement mechanism must be employed

Implementation

- Replacement Mechanisms
  - Least Frequently Used (LFU)
  - Least Recently Used (LRU)
  - Random
  - Variants of the previous
  - **Pseudo LRU**
    - One of the Simplest and Most used
    - Easy to implement

Implementation

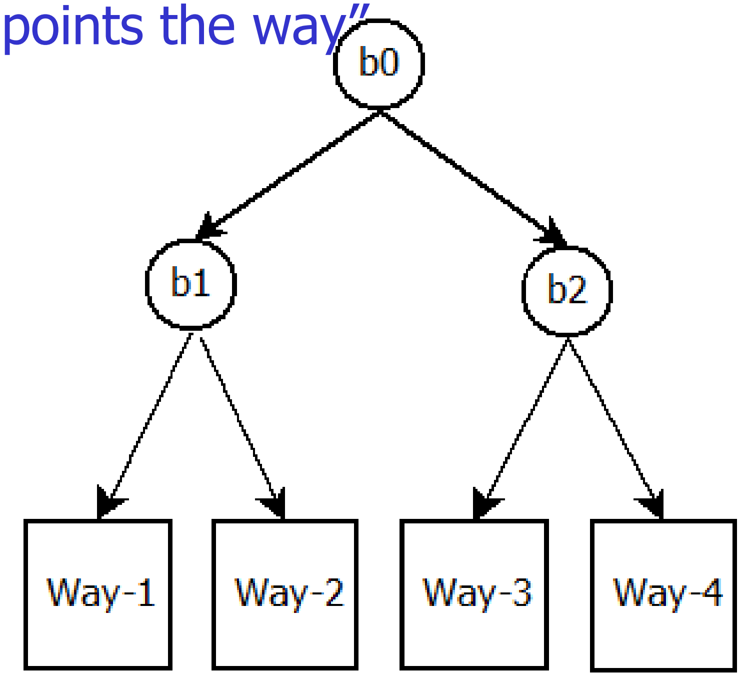
Embedded Systems
Research Group

- Replacement Algorithm
  - Mechanism used reduce the chances of discard a cache line being used
  - PLRU (Pseudo Least Recently Used) – is a light implementation of the LRU algorithm
  - It has a great efficiency
  - Requires few hardware

Implementation

Embedded Systems Research Group

- Replacement Algorithm
  - **Pseudo LRU**
    - Uses a binary tree to decide which *way* should be replaced
    - Each binary branch (b0, b1, …) "points the way" to be replaced
      - **0** means "**go left"**
      - **1** means **"go right"**
    - And "pointer bits" are updated on every access to point to opposite way of the access

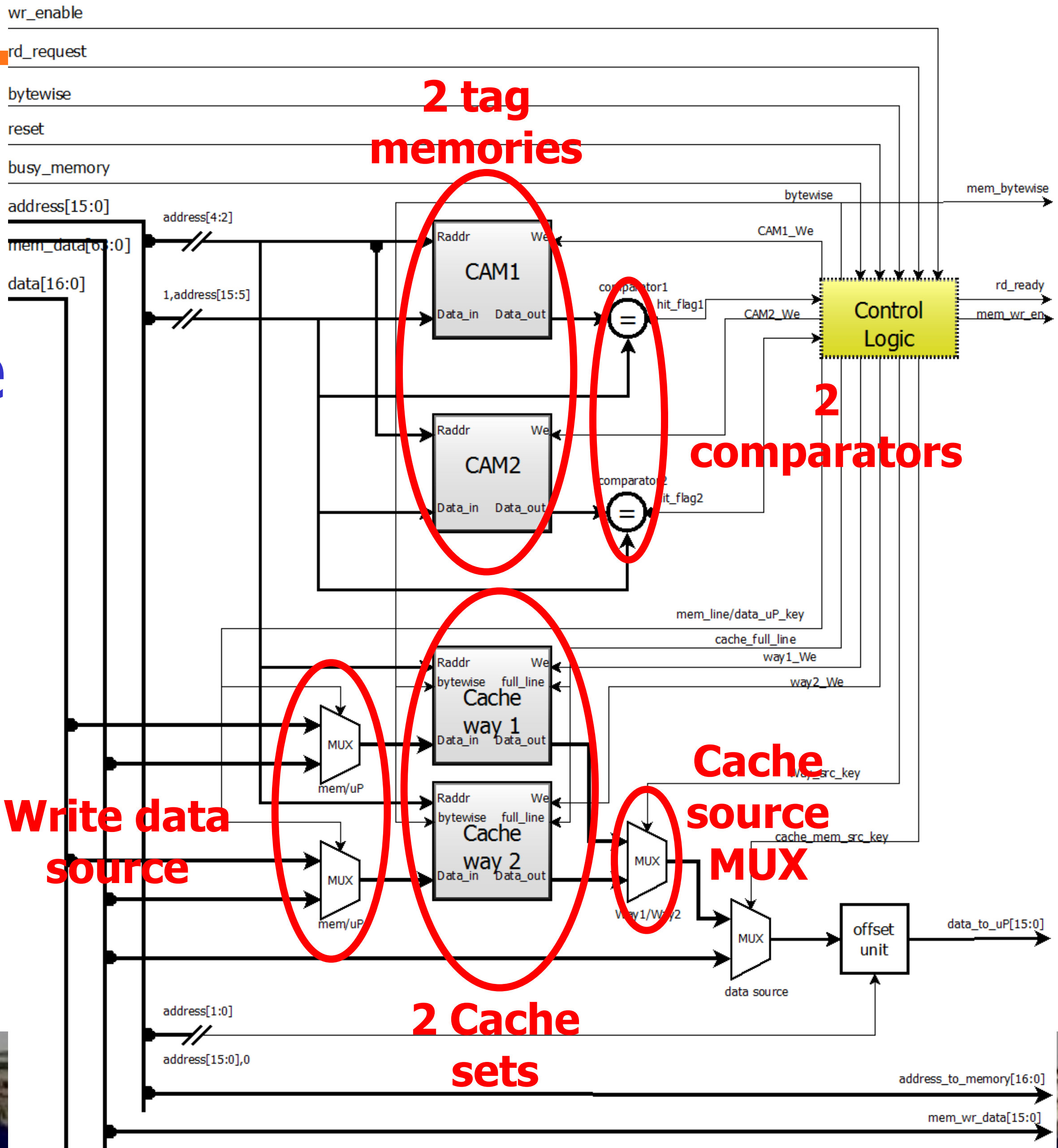


Implementation

Two-way

Set assoc.:

Data cache



Implementation

Embedded Systems Research Group

- Try it by yourselves!
  - Create a new Project on Vivado
  - Start from the Direct mapped implementation code
  - Adapt
    - (follow the previous mentioned steps)

Implementation

Embedded Systems
Research Group

- Test it
  - Use the provided “mem_hierarchy.v” file, include it in the project
    - It is the Top module that connects the cache and a simulation of an external memory
  - External Memory simulation
    - “Offchip_mem.v” – uses on-chip memory but adds latency cycles
    - Add it to the project

Implementation

Embedded Systems Research Group

- Thank you for your attention,
  - Questions?