I J C T A, 9(17) 2016, pp. 8509-8517

A Reconfigurable Cache Design for Embedded Dynamic Data Cache

Shameedha Begum, T. Vidya, Amit D. Joshi and N. Ramasubramanian

ABSTRACT

Applications that are executed on devices such as automobiles home appliances mobile phones can be termed as embedded applications. Cache memories stores the most frequently used data and instructions. The desktop configurations execute wide range of applications but specific deadlines are involved in the applications that are to be executed on embedded systems. The proposed Reconfigurable Embedded Dynamic Data Cache is a hardware design that reconfigures the data cache during program execution with respect to two parameters, namely associativity and block size. Associativity can be configured to be one of Direct Mapped, 2-way Set Associative or 4-way set Associative while the block size can be configured to be one of 64 bytes, 128 bytes or 256 bytes. Miss rate of the cache during program execution has been used to determine when reconfiguration needs to be done. A 4-way set associative cache of size 64KB with LRU replacement policy and write-back policy is used as a base cache upon which dynamic reconfiguration can be done.

Keywords: Cache, Associativity, Block size, LRU.

I. INTRODUCTION

The Embedded applications are characterized by operations that must be performed to meet strict deadlines as well as power and energy requirements [1]. Hence the performance of CPU and cache are paramount in the case of embedded applications. Since embedded applications have specific functionality within a larger system, the cache can be tuned to that particular application to achieve better performance [2]. Cache memories are the first level in the memory hierarchy of any computer system and hold the most frequently accessed data and instructions by the processor. The main principle behind cache is the Principle of Locality which can be further divided as Spatial Locality and Temporal Locality [3]. The address of any item needed by the processor is sent to the cache first and only when the item is not available in the cache, the address is sent to the main memory. Whenever the referenced item is available in the cache, it is called a hit and if the item is not available, then it is called a miss. When a miss occurs, the referenced item must be fetched from the main memory and owing to spatial locality, instead of fetching only the referenced item, a set of items at contiguous memory addresses are fetched. This set of items that are fetched from the main memory whenever a miss occurs is called a block/ line.

The organization of a cache refers to the way a cache manages the subset of items stored in it and satisfies the processor's requests in a transparent manner. It depends on the parameters like Mapping of memory address to the cache,Replacement policy, Write strategy [4].

1.1. Mapping: The cache is divided into blocks and mapping specifies where in the cache can a block that is fetched from the main memory can be placed. Based on the mapping caches can be categorized as follows

Direct mapped caches: For every block that is fetched, there is exactly one block in the cache where it can be placed.

* Department of Computer Science and Engineering, National Institute Of Technology, Tiruchirappalli, Tamil Nadu, India, *E-mail: shameedha@nitt.edu; thiyagarajan.vidya@gmail.com; adj.comp@coep.ac.in, nrs@nitt.edu*

Fully Associative caches: A block can be placed anywhere in the cache

Set-associative caches: The cache is divided into sets with each set containing k blocks. A block that is fetched from the main memory is mapped to exactly only one set but within the set it can be placed anywhere in the k locations. Such a cache is said to be k-way set associative. A direct mapped cache is 1-way set associative while a fully associative cache is n-way set associative where n is the total number of blocks in the cache

1.2. Replacement Policy: Replacement policy dictates which block in the cache must be evicted whenever a new block has to be brought in from memory due to a miss. Some of the commonly used replacement policies are LRU, MRU, FIFO and Random.

1.3. Write Strategy: During a write operation, there is a choice for the designer with regard to whether the write must be performed on the main memory also or not. If the main memory is also simultaneously updated along with the cache during a write, then it is called a write-through strategy. If only the cache is updated during a write and main memory updated during replacement, then it is called a write-back strategy.

1.4. Cache Components: Data RAM, Tag RAM, Valid Bits, Dirty Bits, Cache Controller, Comparators and Multiplexers are the important components of a cache.

1.4.1 Data RAM: The subset of the main memory data held by the cache is stored in Data RAM.

1.4.2 Tag RAM: A portion of the main memory address that is used to check whether the cache holds the data corresponding to that address is called tag. Each block in the cache is associated with a tag and all the tags are stored in Tag RAM.

1.4.3 Valid Bit: This bit is associated with each block to indicate whether the data held is valid or not.

1.4.4 Dirty Bit: This bit is associated with each block and indicates whether the block has been modified or not. This bit is needed only for write-back strategy and is not necessary for write-through strategy.

1.4.5 Cache Controller : The cache controller is a finite state machine that is responsible for the proper operation of the cache and interfaces with the processor and memory.

1.5. Cache Operation

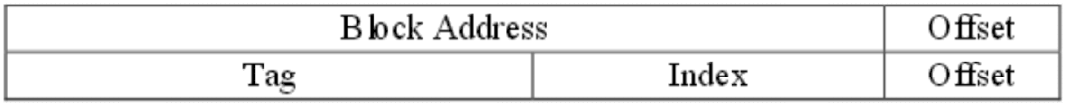

Figure 1: Memory Address Fields

The operation of a cache involves finding whether the referenced memory address by the processor is available in the cache and if so it performs the needed read/write operation and generates a hit signal. If the address is not available it generates a miss signal and fetches the desired block containing the word from the memory and places it in the appropriate set in the cache. If all the ways in a set contain valid entries, then a block is replaced from the set to make way for the new block according to the replacement policy. For the purpose of determining whether an item referenced by an address is available or not, the memory address is divided into three fields namely offset, index and tag.

Offset: This field selects the desired word from the block where a word is the unit of data transfer between processor and memory system.

Index: This field selects the needed set in the cache

Tag: This field is used to compare with the tag value stored in the cache to check for a hit

The tag and the index field put together is called the block address. For a direct mapped cache, the cache block that is mapped to a memory address is given by Block Address MOD (Number of Blocks in the cache) and for a k-way set associative cache with n sets, the cache set to which a memory address is mapped is Block Address MOD (Number of sets in the cache) and for a fully associative cache there is no mapping since a memory block can be anywhere in the cache. After obtaining the correct cache set/block, the valid bit is checked to see if the entry is valid and then the tag portion of the address is compared with the tag value stored in the cache. If the tag values match, then it is a hit else it is a miss.

2. LITERATURE REVIEW

Present day computer systems have usually more than one level cache between the processor and main memory and they are known as L1, L2 or L3 caches depending on their proximity to the processor with L1 being closest and L3 being farthest. L1 caches are split in the sense that there are separate caches for instruction and data while L2 and L3 are unified that is they hold both instructions and data.

The most common metric used to measure the performance of any memory hierarchy is the Average Memory Access Time (AMAT) which is given by *AMAT = Hit Time + Miss Rate * Miss Penalty*, where Hit Time: Time needed to access a word in the cache Miss Rate: Number of misses per memory reference Miss Penalty: Time needed to fetch a block from the main memory to cache on a miss From the equation for AMAT, it is clear the that AMAT can be improved by decreasing Hit Time, decreasing Miss Rate or decreasing Miss Penalty. Cache organizations can be varied to reduce AMAT by decreasing one the three above mentioned parameters. Of the three parameters, miss rate can be reduced by either increasing the block size or increasing associativity. But the flip side to increasing the block size is a higher miss penalty and a higher associativity can cause an increase in the hit time. So while increasing the block size and associativity, it must be taken care to see that the negatives do not overpower the positives leading to an increase in the AMAT.

2.1. RE-Configurable Caches

The performance of cache architecture is dependent on the kind of applications that use the particular design. Desktop systems usually execute a wide range of applications and do not have stringent requirements with respect to execution deadlines and power. Hence, a cache architecture is chosen that would reasonably fit all the applications. Embedded systems are designed to execute a small set of well-defined applications for their entire lifetime. In such systems caches are configured/tuned to those set of applications so as to increase performance. The tuning can be done with respect to the cache organization parameters such as associativity, block size, replacement policy and size of cache. Using a single cache, different embedded applications can be executed but with different configuration parameters in such a way that would maximize their performance. This change of configuration for every application is done through software mechanisms [5].

C. Zhang et al. [5] have proposed a configurable cache which can be configured as a direct-mapped, 2-way set associative or 4-way set associative cache under software control using a configuration register. The design also has the capability to shut down certain regions of the cache to reduce energy consumption. In [6], the cache is reconfigured to have varying sizes depending on the requirements of the application. The researchers have used a Hardware Description Language (HDL) to implement a reconfigurable cache with associativity as the configuration parameter [7].

2.2. Dynamically Re-configurable Caches

While re-configurable caches change the cache parameters before starting the execution of a new application, dynamically re-configurable caches can modify the parameters when an application is in execution.

Dynamically Re-configurable caches use phase based behavior of a program to determine whether re-configuration is needed and what the configuration parameters must be for that phase. A program phase is a set of intervals within a program's execution that have similar behavior regardless of temporal adjacency [8]. The similarity in behavior are characterized by values of metrics such as the number of cache misses, number of branch mispredicts, energy consumed, number of instructions executed per cycle (IPC). Program intervals whose values are close for the above given metrics are considered to belong to the same phase. For caches, miss rates are used to determine program phases. The proposed work also utilizes miss rates to dynamically reconfigure the cache.

Sherwood et al. [8] have done extensive research to show that programs exhibit similar characteristics across different metrics in different time intervals. Such time intervals having similar behaviour are called phases. The phase behaviour of programs have inspired many researchers to dynamically reconfigure the caches whenever phase behaviour changes. Zhang et al. in [5] presents a heuristic to determine the best cache configuration over the cache organization parameters of size, block size and associativity using hardware. In [9] Gordon-Ross, researchers have used offline phase classification using basic block vectors and phase prediction to reconfigure a two level cache over size, associativity and block size. Researchers in [10] have proposed a hardware mechanism to categorize phases based on conflict miss pattern to reconfigure the cache by dynamically enabling or disabling the ways in the cache. A decision tree model based on Dead Set count and Stack Distance has been proposed in [10] to allow reconfiguration on both size and associativity.

3. MOTIVATION

The proposed design has used ideas presented from [2][5][7]. Taking clue from [4], miss rate has been used to categorize a program into phases and apply a cache configuration to that phase. Implementation of LRU using counter method as proposed by [12] has been adopted to implement the LRU replacement policy.

4. PROPOSED DESIGN

The proposed cache is a dynamically reconfigurable cache that can be configured based on associativity or block size. The associativity of the cache can be configured to be direct-mapped, 2-way set associative or 4-way set associative while the block size can be configured to be of 64 bytes (16 words), 128 bytes (32 words) or 256 bytes (64 words). Of the total 9 configurations possible, cache tuning heuristic discussed in [5] is applied in intervals and the configuration having the minimum miss rate is chosen for the application till the observed miss rate exceeds beyond a threshold at which point, the cache goes into configuration determination mode to again find the best configuration. The cache has been designed using two modules namely Red Cache and Dynamic Red Cache.

The block diagram of proposed cache design is shown in Figure 2. The Red Cache is a reconfigurable cache that can be configured to work with different associativities and block sizes. The Dynamic Red

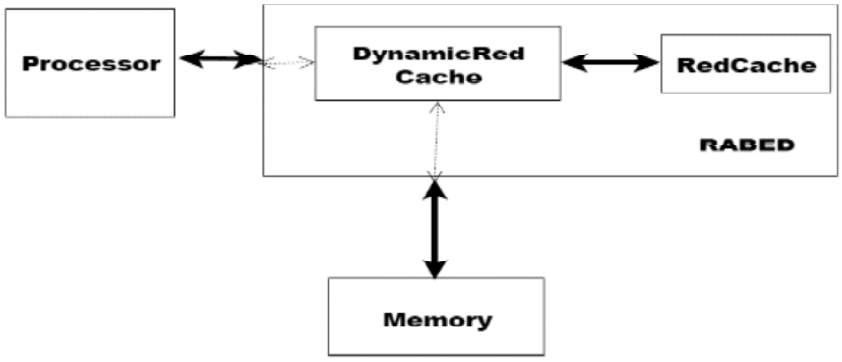

Figure 2: Block Diagram proposed Design

Cache module configures the Red Cache module at appropriate intervals to obtain an optimal miss rate. The processor and memory interface with the Dynamic Red Cache module.

The main modules in the proposed cache design are Red Cache and Dynamic Red Cache

4.1. Red Cache

This is a reconfigurable cache which has been designed using a base cache having the following parameters

Size: 64KB
Associativity: 4-way set associative having 256 sets
Block Size: 64 bytes (16 words, 1 word = 32 bits)
Replacement Policy: LRU
Write-Strategy: Write-through
Address Bus width: 24 bits, memory is assumed to be word addressable
Data Bus width: 32 bits

The cache is reconfigurable with respect to associativity and block size. Associativity of the cache can be direct-mapped, 2-way or 4-way and block size can be 16 words, 32 words or 64 words. The organization of the base cache is shown in Figure 3.

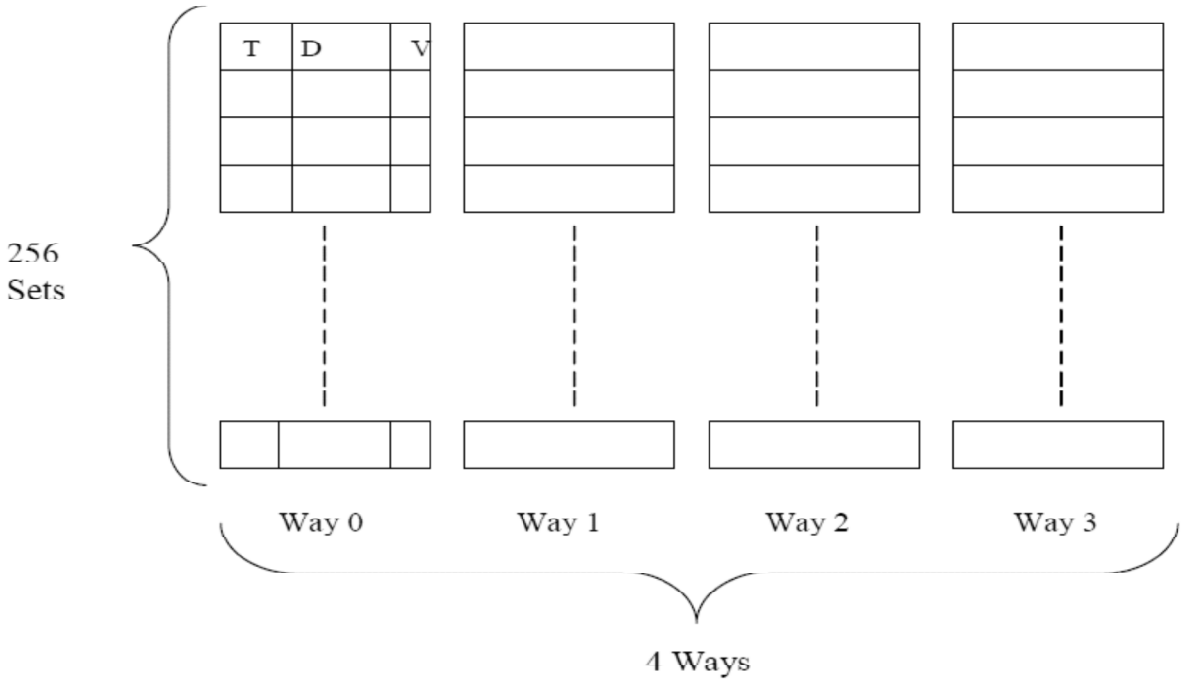

Figure 3: Organization of base cache of proposed design

4.1.1. Storage elements

The Tag RAM, Data RAM and Valid Bits of the cache have been designed using the Register File module available with the BSV library. Dirty bit has not been used since the cache follows a write-through policy. Width of the Tag RAM has been set to accommodate the maximum tag bits needed which is for 4-way set associative. Each block also has a two bit counter for LRU implementation.

4.1.2. Interface Signals: The Figure 4 shows the Interface signals of Red Cache.The Table 1 illustrates the signals and its direction along with appropriate description. Direction indicates whether the signal is sent into the module or signal is sent out of the module.

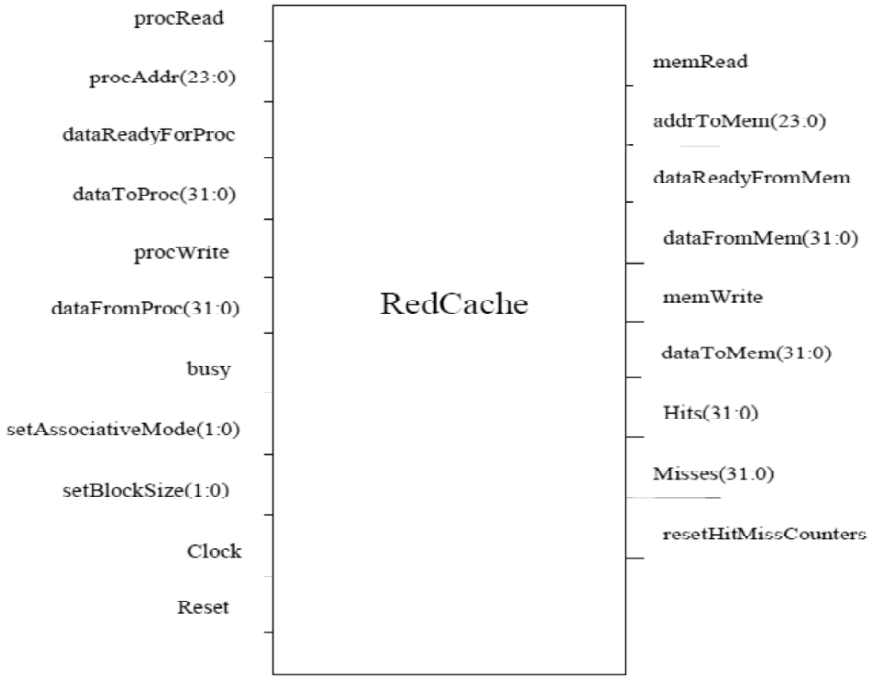

Figure 4: Red Cache Interface Signals

Table 1
Interface Signals of Red Cache

Signals	*Direction*	*Description*
procRead	In	Asserted by processor to perform a read operation on the cache
procAddr	In	A 24-bit address sent by the processor to the cache to read/write a word
dataReadyForProc	Out	Asserted by cache to the processor to indicate that data from the requested address is available
dataToProc	Out	A 32-bit data that is sent from the cache after a read operation
procWrite	In	Asserted by processor to perform a write operation on the cache
dataFromProc	In	A 32-bit data sent by the processor to be written
Busy	Out	Asserted True by cache to the processor to indicate it is not yet ready to accept new requests since cache state is being updated. Asserted False when the cache is idle
setAssociativeMode	In	A 2-bit value indicating the associativity to be used by cache which can be one of direct-mapped, 2-way or 4-way set associative
setBlockSize	In	A 2-bit value indicating the block size to be used which can be one of 64 bytes, 128 bytes or 256 bytes
memRead	Out	Asserted by cache to the memory to read a word from memory
addrToMem	Out	A 24-bit address sent to the memory by the cache to read/write a word
dataReadyFromMem	In	Signal asserted by memory to indicate that data from the requested address is available

contd. table 1

Signals	*Direction*	*Description*
dataFromMem	In	A 32-bit data sent by memory to cache after a read operation
memWrite	Out	Signal asserted by cache to memory to perform a write operation
dataToMem	Out	A 32-bit value sent by cache to be written into memory
Hits	Out	A 32-value giving the number of hits since the last reset of hit/miss counters
Misses	Out	A 32-bit value giving the number of misses since the last reset to the hit/miss counter
resetHitMissCounters	In	Signal to be asserted by any external module to reset the hit/miss counters

4.1.3 Address Mapping: For the base cache, the memory address can be divided into the following fields to find the correct block in the cache and retrieve the needed data.

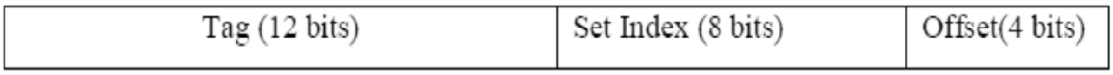

Figure 5: Address Fields in proposed cache design

The Red Cache for all its 9 configurations always uses the index field (8 bit) in the mapping of the memory address to base cache to find the appropriate set and uses additional bits in the tag field to select the appropriate way for direct-mapped and 2- way set associative configurations. Since the base cache has a block size of 16 words, block sizes of 32 words and 64 words are configured by allocating 2 consecutive blocks in the same way for block size of 32 words and allocating 4 blocks in the same way for block size of 64 words. The way selection for direct mapped and 2-way set associativity have been implemented in such a way that whenever the configuration of the cache moves from lower associativity to higher one, addresses that are hits in the lower associativity remain hits even in the higher one. The complete tag address is matched for all the configurations. The address mapping, way selection and block offset for the configurations are shown below.

Table 2
Address Mapping Red Cache

Configuration(Associativity + Block Size)	*Tag bits used for way selection*	*Size of Block Offset*
Direct mapped + 16 words	Bits 13:1200 – Way 001 – Way 110 – Way 211 – Way 3	4 bits
Direct mapped + 32 words	Same as above	5 bits
Direct mapped + 64 words	Same as above	6 bits
2 way + 16 words	Bit 120 – Way 1 or Way 21 – Way 1 or Way 3	4 bits
2 way + 32 words	Same as above	5 bits
2 way + 64 words	Same as above	6 bits
4 way + 16 words	No extra tag bits used. All 4ways are selected	4 bits
4 way + 32 words	No extra tag bits used. All 4ways are selected	5 bits
4 way + 64 words	No extra tag bits used. All 4ways are selected	6 bits

4.2. Dynamic Red Cache

This module implements the Reconfigurable Associativity and Block Size Embedded Dynamic Data Cache. It utilizes the Red Cache module and reconfigures its parameters dynamically. Any dynamic reconfiguration involves two decisions which are Identification of the best configuration and Determination as to when to change the configuration.

4.2.1. Identifying the best configuration

Of the 9 configurations with which the Red Cache can be configured, this module determines the best configuration using miss rate as the yard stick. The best configuration is determined by adopting the heuristic proposed by [5]. The determination of the best configuration is done in two stages

Stage 1: Starting with associativity to be direct-mapped and block size to be 16 words, the best block size is determined changing the block size from 16 words to 32 words and to 64 words. The best block size is the one that gives the minimum miss rate. Each configuration is executed for a certain number of memory accesses which has been determined to be 128 memory accesses after many trials. This value is termed as Configuration Determination Interval (CDI).

Stage 2: For the best block size identified, the best associativity is found by changing it to 2-way from direct-mapped and to 4-way from 2-way and again executing each configuration for 128 memory accesses. Once the best configuration is identified, it is executed for a certain interval which is called a phase. Best configuration is always determined at the beginning of execution.

4.2.2. Interface signals

The Interface signals of the Dynamic Red Cache module are shown in Table 3. This module interfaces with the processor, memory and the Red Cache.

Table 3
Interface Signals of Dynamic Red Cache

Signal	*Direction*	*Description*
procRead	In	Asserted by processor to perform a read operation on the cache
procAddr	In	A 24-bit address sent by the processor to the cache to read/write a word
dataReadyForProc	Out	Asserted by cache to the processor to indicate that data from the requested address is available
dataToProc	Out	A 32-bit data that is sent from the cache after a read operation
procWrite	In	Asserted by processor to perform a write operation on the cache
dataFromProc	In	A 32-bit data sent by the processor to be written
busy	Out	Asserted True by cache to the processor whenever either the Red Cache is asserting busy or Dynamic Red Cache is updating its state. Asserted False when both Red Cache and Dynamic Red Cache are IDLE
memRead	Out	Asserted by cache to the memory to read a word from memory
addrToMem	Out	A 24-bit address sent to the memory by the cache to read/write a word
dataReadyFromMem	In	Signal asserted by memory to indicate that data from the requested address is available
dataFromMem	In	A 32-bit data sent by memory to cache after a read operation
memWrite	Out	Signal asserted by cache to memory to perform a write operation
dataToMem	Out	A 32-bit value sent by cache to be written into memory
Hits	Out	A 32-value giving the number of hits since the last reset of hit/miss counters
Misses	Out	A 32-bit value giving the number of misses since the last reset to the hit/miss counter
hitMissValuesReady	Out	Signal asserted by Dynamic Red Cache at the end of every phase to indicate that the hit and miss values can be obtained

contd. table 1

Signal	*Direction*	*Description*
getAssociativeMode	Out	Returns the associative mode used for the just ended phase
getBlockSize	Out	Returns the block size used for the just ended phase
setCacheConfiguration	In	The Dynamic Red Cache can be configured to use either StaticConfiguration or Dynamic Configuration. In the case of static configuration, the cache parameters that are set at the beginning are used for the entire program execution else dynamic reconfiguration can be done. A 2-bit value is used00-Static :: 01-Dynamic
setStaticConfigurationParams	In	If the Dynamic Red Cache is to use static configuration, then the cache parameters are specified through this method which include associativity and block size

5. CONCLUSION

The work focuses on a new cache design for embedded applications. A reconfigurable cache design for embedded data cache is proposed with dynamic configuration. Red cache and Dynamic Red cache are used in this design. The cache parameters like associativity and block size can be used with this cache design to achieve a higher AMAT and miss penalty. It can also help to reduce energy requirements along with strict deadline achievements.

REFERENCES

[1] Wolf, M. (2012). *Computers as components: principles of embedded computing system design*. Elsevier.

[2] Zhang, C., Vahid, F., &Najjar, W. (2003, June). A highly configurable cache architecture for embedded systems. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on* (pp. 136-146). IEEE.

[3] Patterson, D. A., & Hennessy, J. L. (2013). *Computer organization and design: the hardware/software interface*. Newnes.

[4] Hennessy, J. L., & Patterson, D. A. (2011). *Computer architecture: a quantitative approach*. Elsevier.

[5] Zhang, C., Vahid, F., &Lysecky, R. (2004). A self-tuning cache architecture for embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, *3*(2), 407-425.

[6] Naz, A., Kavi, K., Oh, J., &Foglia, P. (2007, March). Reconfigurable split data caches: a novel scheme for embedded systems. In *Proceedings of the 2007 ACM symposium on Applied computing* (pp. 707-712). ACM.

[7] Bani, R. R., Mohanty, S. P., Kougianos, E., &Thakral, G. (2010, December). Design of a Reconfigurable Embedded Data Cache. In *Electronic System Design (ISED), 2010 International Symposium on* (pp. 163-168). IEEE.

[8] Sherwood, T., Perelman, E., Hamerly, G., Sair, S., & Calder, B. (2003). Discovering and exploiting program phases. *Micro, IEEE*, *23*(6), 84-93.

[9] Gordon-Ross, A., Lau, J., & Calder, B. (2008, May). Phase-based cache reconfiguration for a highly-configurable two-level cache hierarchy. In*Proceedings of the 18th ACM Great Lakes symposium on VLSI* (pp. 379-382). ACM.

[10] Banerjee, S., &Nandy, S. K. (2007, January). Program phase directed dynamic cache way reconfiguration for power efficiency. In *Proceedings of the 2007 Asia and South Pacific Design Automation Conference* (pp. 884-889). IEEE Computer Society.

[11] Sundararajan, K. T., Jones, T. M., &Topham, N. (2011, July). Smart cache: A self adaptive cache architecture for energy efficiency. In *Embedded Computer Systems (SAMOS), 2011 International Conference on* (pp. 41-50). IEEE.

[12] Sudarshan, T. S. B., Mir, R. A., &Vijayalakshmi, S. (2004, December). Highly efficient LRU implementations for high associativity cache memory. In*Proceedings of 12th IEEE International Conference on Advanced Computing and Communications* (pp. 24-35).