# Design And Integration of EMISS SoC Using 65nm Technology

By

**Jani Parthkumar Girishchandra**

**09MEC008**


**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**AHMEDABAD-382481**

**MAY 2011**

# Design And Integration of EMISS SoC Using 65nm Technology

**Major Project**

Submitted in partial fulfillment of the requirements

For the degree of

**Master of Technology in VLSI Design**

By

**Jani Parthkumar Girishchandra**

**09MEC008**

Under the Guidance of

**Prof. Usha Mehta**

**Associate Professor**


**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**AHMEDABAD-382481**

**MAY 2011**

# Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in VLSI Design at Nirma University and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

**Jani Parthkumar Girishchandra**

# Certificate

This is to certify that the Major Project entitled "Design And Integration of Mini SoC Using 65nm Technology" submitted by **JANI PARTHKUMAR GIRISHCHANDRA(09MEC008)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Prof. Usha Mehta
Associate Professor,
Department E & C Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Mr. Asif Ali Zargar
Technical Leader,
HED-HVD Design Team,
ST Microelectronics,
Greater Noida

Dr. N M Devashrayee
PG Co-ordinator,M.Tech VLSI Design,
Department E & C Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. A S Ranade
Head of Department,E&C Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. K Kotecha
Director,
Institute of Technology,
Nirma University ,Ahmedabad

# Abstract

Nowadays the circuits, with the feature of lower power, lower cost per gate area, faster operation, more reliable implementation, smaller physical size, greater design security are the main requirements. SoC are the best alternative in this scenario which satisfy all these criteria. SoC is the well developed branch of electronics and in todays era everything is built on SoC(silicon on chip). SoCs are highly efficient and have improved performance because they are programmable and we can implement desired functionality and makes our task easier. SoC contains many IP(Intellectual Property) blocks which contains many modules of required property. The different memory modules are use in project. In past , television antennas are used to transfer data. In that, Analog signals are transferred. In this process we cant get the good picture quality due to weakness of analog signals.

Different memories are added in the design like NAND flash memory, NOR flash memory, SPI flash memory, and MCC cache memory and also connected ST Bus. These memories are designed as a EMISS SoC. This design is verified using different tools. During verification different design bugs are arrived and using the trial and solution method these bugs are removed.

Tests cases were run for read, write, hit, miss, read mode write, continuous read, continuous write to check the design using Stymgen environment. The design was being tried to be failed and was being tried to find out as many bugs as possible.

# Acknowledgements

I am deeply indebted to my thesis supervisor Mr.Sanjeev Vwrshney for his constant guidance and motivation. He has devoted significant amount of his valuable time to plan and discuss the thesis work. Without his experience and insights, it would have been very difficult to do quality work.

I would also like to extend my gratitude to Asso. Prof.Usha Mehta and Dr. N.M.Devashrayee for fruitful discussions during Design And Integration of Mini SoC Using 65nm Technology meetings and for their encouragement.

My special thanks to Mr.Asif Rashid Zargar who was the first person I used to approach whenever I got stuck. I would also like to thank my friends and IP Design Team members (Mr.Sandeep,Mr.Kriti,Mr.Naveen,Mrs.Deboleena,Mrs.Parul)of my class for their delightful company which kept me in good humor throughout the year.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the degree program successfully.

**Jani Parthkumar Girishchandra**

**09MEC008**

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 STBus

The STBus is a set of protocols, interfaces, primitives and architectures specifying an interconnect subsystem, versatile in terms of performance, architecture and implementation. The STBus is the result of the evolution of the interconnect subsystem developed for microcontroller dedicated to consumer application, such as set top boxes, ATM networks, digital still cameras and others. Such an interconnect was born from the accumulation of ideas converging from different sources, such as the transporter (ST20), the Chameleon program (ST40, ST50), MPEG video processing and VCI (Virtual Component Interface)organization. Today the STBus is not only a communication system characterized by protocol, interfaces, transaction set and IPs, but also a technology allowing to design and implement communication networks for Systems On Chip with the support of a development environment including tools for system level design and architectural exploration, silicon design, physical implementation and verification.

## 1.2 EMISS(External Memory Interface Subsystem)

The EMISS supports NOR flash, NAND Flash, Serial Flash, and PCI interfaces. The design consists of two modules, namely, the subsystem core and pad logic. The subsystem core instantiates to support different interfaces all the responsible IPs. The arbitration logic to grant the mastership of the bus to a bus interface (or device) is implemented in the sub-system core. The pad logic multiplexes different signals on to a common bus which is fed to the pads. The necessary logic to control the tri-stating of the pads is also generated within the pad logic of the sub-system. There are different modules in EMISS block like NAND flash controller , NOR flash controller, serial flash controller.

### 1.2.1 NAND flash controller

The NAND controller consists of two independent NAND controllers known as Hamming NAND controller and BCH NAND controller. Both these controllers are described briefly in the sections below.

#### 1.Hamming NAND controller

The Hamming NAND flash controller supports boot, flex mode and advanced flex modes. In all the modes the external NAND flash could be 8 or 16 bits. It could be small page or large page. The NAND device can accept 3 bits, 4 bits or 5 bits of address. All the configurations for boot mode are inferred from the value of static pins.

#### 2.BCH NAND controller

The BCH NAND controller has been designed to support both MLC and SLC kind of devices. It supports boot and AFM modes. In boot mode only 8 bit parts are supported while in AFM mode 16bit parts are also supported. The BCH NAND

controller is equipped with dedicated hardware DMAs for efficient data transfers in AFM mode.

### 1.2.2 NOR flash controller

It supports NOR flash (AMD, Intel or ST), MPX master, DVBCI and ATAPI interfaces. The DVBCI and ATAPI interfaces are supported using additional glue that is designed around the EMI. EMI also supports SDRAM interface. The SDRAM pins are not bristled out and hence cannot be supported by the sub-system.

### 1.2.3 Serial Flash controller

The SPI Boot IP supports booting from the serial flash, which could be ST type or Atmel type. The type of flash is configured by the static pin which is bristled out of the chip. After boot, the writing into the external serial flash is supported by SPI functionality in comms or through the new FSM mode in the SPI controller. The SPI boot pins are multiplexed with the SPI pins of the comms. This multiplexing is done at the top level and is outside the scope of the document. The subsystem bristles the all the pins required for serial communication.

### 1.2.4 PCI controller

The PCI interface is supported by three blocks which are together known as PCISS. PCI IP purchased from Synopsys. STBus AHB converter from OCCS Glue to connect the STBus-AHB and AHB-PCI bridges. In addition to connecting the two bridges, the additional functionalities such as address translation are also implemented in the glue. The STBus-PCI bridge is explained in detail in chapter 2.

## 1.3 NAND Flash Memory

It is a multilevel cell device from the NAND flash 4224-byte page family of non-volatile flash memories. It has a density of 16 Gbits. The device operates from a 3 V power supply. The address lines are multiplexed with the data input/output signals on a multiplexed x8 input/output bus. This interface reduces the pin count and makes it possible to migrate to other densities without changing the footprint. Each block can be programmed and erased up to 10,000 cycles (with error correction code (ECC) on). The device also has hardware security features; a write protect pin is available to provide hardware protection against program and erase operations.

The devices feature an open-drain, ready/busy output that identifies if the program/erase/ read (P/E/R) controller is currently active. The use of an open-drain output allows the ready/busy pins of several memories to be connected to a single pull-up resistor. The memory array is split into 2 planes of 2048 blocks each. This multiplane architecture makes it possible to program 2 pages at a time (one in each plane), to erase 2 blocks at a time (one in each plane), or to read 2 pages at a time (one in each plane) dividing by two the average program, erase, and read times.

The device has the Chip Enable 'don't care' feature, which allows the bus to be shared between more than one memory at the same time, as Chip Enable transition during the latency time do not stop the read operation. Program and erase operations can never be interrupted by Chip Enable transition. There is the option of a unique identifier (serial number), which allows the NAND flash to be uniquely identified.

## 1.4 NOR Flash Memory

The NOR architecture is currently the most popular flash architecture. It is commonly used in EPROM and EEPROM designs. Aside from active transistors, the largest contributor to area in the cell array is the metal to diffusion contacts. NOR architecture requires one contact per two cells, which consumes the most area of all

the flash architecture alternatives. Electron trapping in the floating gate is done by hot-electron injection.

The NOR flah memory featuring single 3.0V power supply, is a 32Mbit NOR-type Flash Memory organized as 4Mx8 or 2M x16. The memory architecture of the device is designed to divide its memory arrays into 71 blocks to be protected by the block group. This block architecture provides highly flexible erase and program capability. The NOR Flash consists of two banks. This device is capable of reading data from one bank while programming or erasing in the other bank.

Access times of 70ns, 80ns and 90ns are available for the device. The device?s fast access times allow high speed microprocessors to operate without wait states. The device performs a program operation in units of 8 bits (Byte) or 16 bits (Word) and erases in units of a block. Single or multiple blocks can be erased.

## 1.5 Serial Flash Memory

SPI-compatible interface that allows for a low pin-count package which occupies less board space and ultimately lower total system costs. SPI serial flash memories are manufactured with SST proprietary, high-performance CMOS Super Flash technology. The split-gate cell design and thick-oxide tunneling injector attain better reliability and manufacturability compared with alternate approaches. SPI flash memory devices significantly improve performance and reliability, while lowering power consumption. The devices write (Program or Erase) with a single power supply of 1.65-1.95V. The total energy consumed is a function of the applied voltage, current, and time of application. Since for any given voltage range, the Super Flash technology uses less current to program and has a shorter erase time, the total energy consumed during any Erase or Program operation is less than alternative flash memory technologies.

## 1.6 PCI Flash Memory

The flash on this board, like the SRAM, is non-volatile. Unlike the SRAM, the flash contains various modes that it operates in. Before accessing the FLASH, it must be placed into the mode of operation desired via an ORDERED sequence of reads and writes to specific addresses with specific data. Ordered is stressed, because some systems will cache up accesses and burst them to the board in an undesired order; hence, the operation will be unsuccessful. This problem can be avoided by performing non-cacheable access or by inserting delays between accesses when performing certain activities.

## 1.7 MCC Cache Memory

The purpose of inserting the cache for inter prediction motion compensation is to reduce the typical bandwidth required for prediction, keeping the worst case band width smaller or same as without cache memory. The bandwidth has been studied on Allegro streams and the compromising size or associatively and complexity has been found: 8-ways associative, 32- byte or 16 bytes word, cache size is 128 KBytes. The one ways was simpler in term of implementation, but it creates a bottleneck which can slow down the cache. This motion compensation cache (MCC) is inserted between the XPred memory access and the STBus. It can be bypassed, in which case the XPred accesses the external memory in the same manner as in previous DELTA versions. In this case, XPred STBus plugs are directly connected to STBus.

But generally 128k bytes is too much and it acquires more space on chip so we will have to reduce the mrmory size from 128kB to 64kB or 32kB.

This second generation of MCC cache,uses an input buffer in order to be STbus latency independant whatever cache size. Because when cache memory become smaller than 128K, cache data collisions can introduce some delay on request sent to STbus which are not acceptable.With an input buffer we have no more collision possible.

# Chapter 2

# STBus overview

The objective of this document is to give all the informations required to understand what is the STBus, covering the communication protocols, the interfaces and the functionality of the building blocks through which it's possible to build interconnect systems at different level of complexity. It has been tried to mantain the level of detail of the different topics such that the understanding of the different concepts and problematics should be easy and fast enough; the description of the building blocks is carried out so that deep technical implementation details are omitted, for sake of simplicity. This document addresses as audience the interconnect architects, the interconnect designers, the application engineers and the project leaders. It can be used as complete and concise reference about the STBus communication system by any people starting architectural or design activities about the STBus.

## 2.1 STBus overview

### 2.1.1 STBus Protocols

Three different types of the STBus protocols exist, each having a different level of complexity in terms of both performance and implementation:

a. Type1 is the simplest and is intended to be used for peripherals registers access.

No pipeline applies. It acts as a RG protocol. Load/store on 1/2/4/8 bytes are supported.

b. Type 2 adds pipelines features. It is equivalent to the "basic" RGV protocol. It supports all operation code for ordered transactions. The number of the requesting cells (i.e. in a packet) is the same than the number of the response ones.

c. Type 3 is an advanced protocol implementing split transactions for high bandwidth requirements (high performance systems). It supports out of order executions. The packet response size might be different than the packet request size (the number of cells differs between request and response). The interfaces maps the STBus transaction set on a physical set of wires defined by this interface.

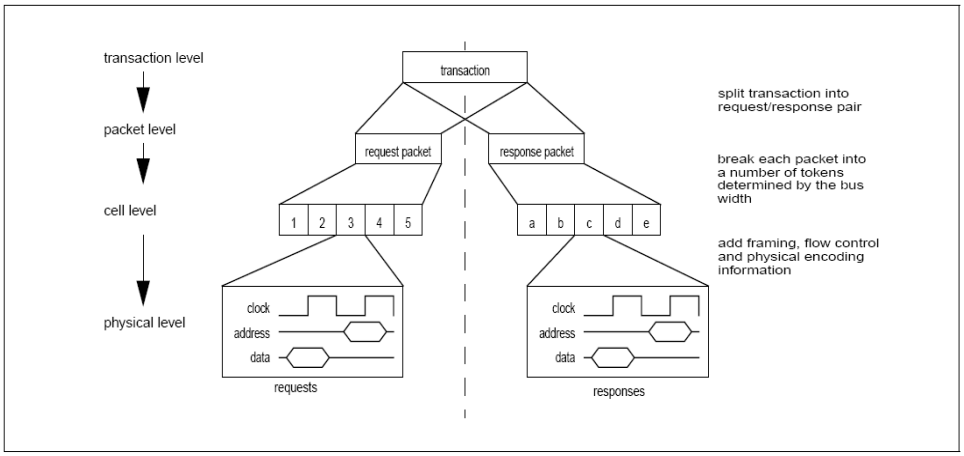


Figure 2.1: STBus protocols layers

Each operation is broken into one or more request/response packet pairs. Primitive operations have a single request/ response pair. Depending on the STBus type, compound operations may contain multiple pairs of packets. These packets are then mapped onto the physical interface as a series of cells. Request cells onto the signals associated with requests, and responses onto those associated with responses. The number of cells in each packet is determined by the amount of data to be transferred

and the width of the interface. Depending on the interface type, the amount of information to be transferred in the request phase is the same (type 1/ type 2) or may differ (type 3) from that to be transferred in the response phase.

## 2.2 STBus basic pin description

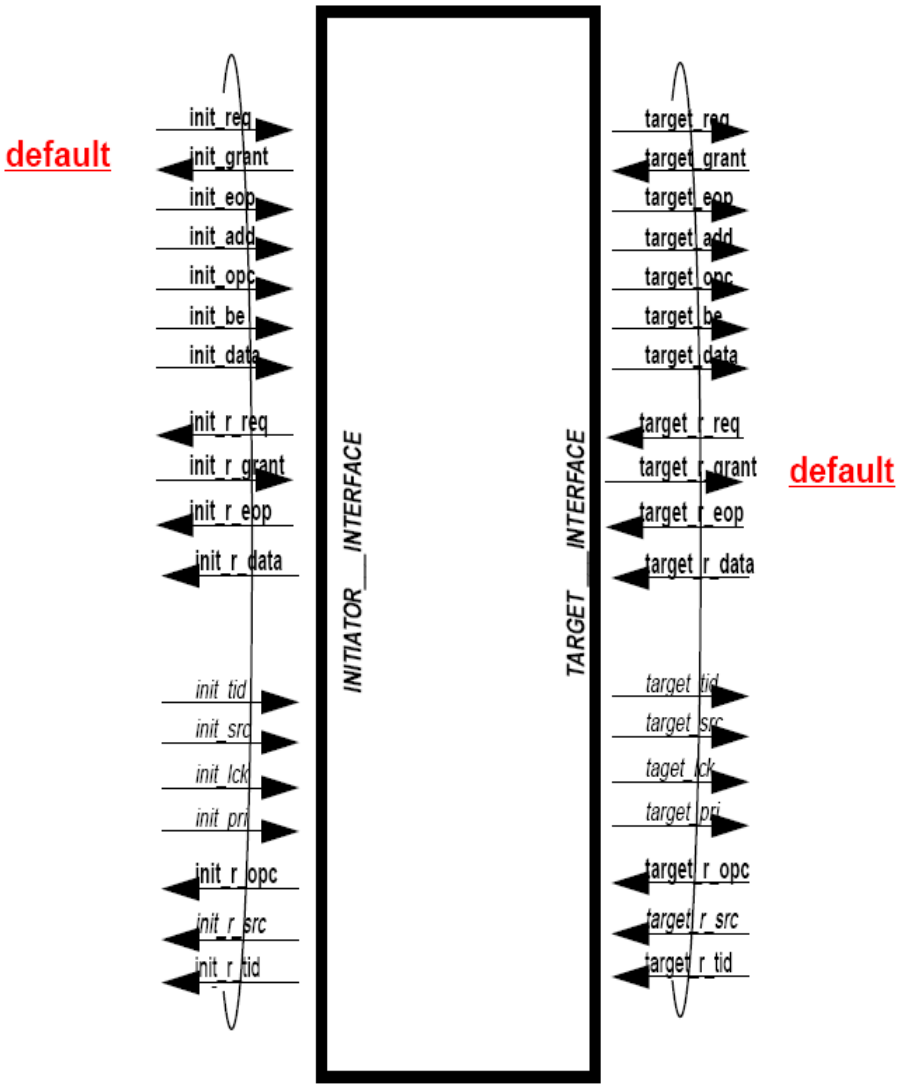


Figure 2.2: Signals and pin descriptions of STBus

- req(request)

  The request is the signal used by the initiator interface to communicate it wants to start a transaction. Once the request is asserted, all the signals associated to the packet to be transmitted must be kept constant until the grant is received from the target interface, meaning the transaction has effectively started.

- Gnt (grant)

  The grant is the signal with which a transaction is considered started after a request has been asserted. The grant generated by a node as consequence of the arbitration is a causal grant (grant on request), while the grant generated by a target module is a default grant and is an information about the availability of the target itself to accept a new request.

- Eop (end of packet)

  The end of packet is the signal marking the last cell of a packet being transmitted. It's used mainly by the node to ensure a packet cannot be interrupted.

- Lck (lock)

  The lock signal is used to link together two or more packets to build complex transactions, such as chunks and ReadModifyWrite operations

- r_req (response request)

  The response request is the signal used by the target interface to communicate it wants to complete a transaction. Once the response request is asserted, all the signals associated to the response packet to be transmitted must be kept constant until the response grant is received from the initiator interface, meaning the transaction has effectively been completed.

- r_gnt (response grant)

  The response grant is the signal with which a transaction is considered completed after a response request has been asserted. The response grant generated

by a node as consequence of the response arbitration is a causal response grant, while the response grant generated by an initiator module is a default response grant and is an information about the availability of the initiator itself to accept a new response.

- r_eop (response end of packet)

  The response end of packet is the signal marking the last cell of a response packet being transmitted. It's used mainly by the node to ensure a response packet cannot be interrupted.

- r_lck (response lock)

  The response lock signal is used to link together two or more response packets during the transmission of complex transactions, such as responses to chunks.

- data[8*2n-1:0] (data with n = 0,1, 2, 3, 4,5)

  This is the data the initiator wants to send to the target in case of store operations.

- add[31:n] (address with n = 0,1, 2, 3, 4,5)

  This is the address of the memory location the initiator wants to access. The address of a transaction must be transaction-aligned. For example an 8 bytes operation can be addressed to the addresses 0x0, 0x8, 0x10, but not to the addresses 0x4 and 0xC.

- opc[7:0] (opcode)

  This signal defines the type of operation the initiator wants to perform. The opcode can be considered composed of three parts: bits [3:0 indicate the operation type, e.g. load/store/cache operation etc. l bits [6:4 specify the size of the operation, e.g. the number of bytes the operation will involve l bit 7 is reserved and in multiplexed buses/systems is used to distinguish between request and

response packets. The STBus interconnect is a non-multiplexed system so bit 7 must always be low

- be[2n-1:0] (byte enables)

  The byte enables signal defines which bytes within a cell are significant. For operations whose size is smaller that the data width, the byte enables basically specifies the low order address bits. For example, with a LD1 in a 32 bits system, address bits [1:0 are implicitly encoded in the byte enables signal. In systems where byte enables are not defined all cells are assumed to be valid.

- src[7:0] (source)

  The src is used for two reasons: the lowest 4 bits can be used by the initiator to identify possible internal sub processes (up to 16), the highest 4 bits are used by the interconnect to unambiguously identify the initiator itself. If the initiator has less than 16 processes (even none), all the unused src bits can be used to identify the initiator in the interconnect.

- tid[7:0] (transaction identifier)

  This signal is used by the initiator to label a transaction with additional information. It is split into two fields:

  tid[3:0] uniqueness information

  tid[7:4] enhancement information The uniqueness information is used by type 3 initiators to unambiguosly identify the transaction, so to be able to detect the correct response in case of out of order traffic. Type 2 initiators haven't such a possibility. The enhancement information is used to enhance the processing of an operation without changing its functionality. The following fields are defined for it: [4] Not end of message

  '0' - this transaction has no relationship to the next transaction (end of message)
  '1' - the next transaction is related to this transaction (member of a message)

[5] Not device access '0' - this store operation will not be posted '1' - this store operation may be posted (that is the system may choose to return an early response, deleting the true response when it is returned to improve performance reducing the latency between grant and response)

[6] Store and forward '0' - each time a cell is received/assembled, it can be immediately propagated to the target interface '1' - first collect all the cells building a packet and then send the whole packet to the target interface

[7] Reserved The message information is used by the initiator to indicate that the system should attend to keep the message components together. This can increase performance in case of access to a page-based memory such as the SDRAM. The write posting information is used by the initiator to indicate that the information in the response to a write operation will be ignored, and then it may wish to have a 'dummy' response returned early. The store and forward information is used to drive the storage behavior at buffers.

- pri[3:0] (priority)

  This signal labels the request packet with an urgency level which the interconnect may use to implement preferential arbitration. The priority is encoded into a 4 bits field; "1111" corresponds to the most urgent or highest priority, "0000" to the least urgent or lowest priority.

- r_data[[8*2n-1:0] (response data with n = 1, 2, 4, 8, 16 or 32 -bytes)

  This is the data read from memory at a specified location during a load operation.

- r_opc[7:0] (response opcode)

  This is the information about the status of the response being presented to the initiator. The encoding of this field is as follows: Success r_opc = "10000000" Error r_opc = "10000001"

Most modules may choose to only implement bit zero of this field. Bits 1 to 6 are reserved for future applications whilst bit[7 is reserved for systems which implement multiplexed request/response transports. The construction and size of an error response may differ from that of a normal response as the target is unable to process the operation, however, typically it will be between 1 and N cells, where N is the size of the request packet. This signal should normally be sampled on the final cell (as marked with eop) of a response packet

- r_tid[7:0] (response transaction identifier)

  The r_tid signal is a copy of the tid signal sent during the transmission of the request packet, and is used by type 3 initiators to detect the request packet to which the response got is relative.

- r_src[7:0] (response source)

  This signal is a copy of the src signal associated to the respective request packet, and is used by the interconnect to identify the initiator toward which the response has to be routed. For an initiator, the lowest bits can be used to detect the internal sub process to which the response got is relative.

- r_pri[3:0] (response priority)

  This signal labels the response packet with an urgency level which the system interconnect may use to implement preferential arbitration. It is a copy of the pri signal sent with the request packet.

- tst_scanenable: Scan test enable

  The STBus interconnect is fully scannable for testability reasons. The tst_scanenable signal is used to enable the scan test operation mode.

- tst_scanin: Scan test input

  This is the scan chain(s) input to feed during the scan test operation mode.

- tst_scanout: Scan test output This is the scan chain(s) output to be monitored during the scan test operation mode.

# Chapter 3

# EMISS(External Memory Interface Subsystem)

The EMISS supports NOR flash, NAND Flash, Serial Flash,and PCI interfaces. The design consists of two modules, namely, the subsystem core and pad logic. The subsystem core instantiates to support different interfaces all the responsible IPs. The arbitration logic to grant the mastership of the bus to a bus interface (or device) is implemented in the sub-system core. The pad logic multiplexes different signals on to a common bus which is fed to the pads. The necessary logic to control the tri-stating of the pads is also generated within the pad logic of the sub-system. There are different modules in EMISS block like NAND flash controller , NOR flash controller, serial flash controller.

## 3.1 Features lists

The proposed EMIPCI subsystem supports following interfaces

- NOR Flash support for different vendors like Intel, Numonyx, AMD etc.(a)
- Synch and Asynch flash support
- 8 bit /16 bit data bus support

- NAND Flash support
- Small page, large page, very large page, very very large page devices supported
- 8bit / 16 bit data bus support
- ECC support for upto 30bit error correction
- Boot as well as non-boot capability
- Error free devices supported
- Host managed AFM mode with dedicated HW DMA support.
- Multi CS and 2X8 bit mode supported in non-boot AFM mode.
- Serial Flash support
- x1, x2, x4 pad support for boot and well as non-boot mode
- Flexible Fast Sequence Mode available for large data transfers with FDMA support.
- Programmable clock division ratios available.
- Flexibility to support Flashes from different vendors with varying commands/protocols.
- DVB-CI
- ATAPI supports for register read/write.
- MPX master and slave
- PCI master or target, host and device configured on boot
- Support for 6 different banks of 128MB(configurable) each

- Different interface pin outs muxed efficiently to provide a low pad count. Muxing is supported along with bus arbitration logic to ensure no bus contention Note: IThe full details of each particular feature are given in the relevant sections, and not in the features list.

## 3.2 IP block context

The sub-system instantiates two blocks viz., sub-system core and the pad logic as shown in Figure 1. The sub-system core instantiates different IPs responsible to implement the different functionalities envisaged in the EMI-PCI-EMPI sub-system. The padlogic multiplexes the different signals coming from the sub-system core on to a common shared bus.The System view of the EMI-PCI-EMPI subsystem is shown in the Figure 1. The block diagram of the EMI-PCI-EMPI subsystem is shown in the Figure 1 below. The scope of this document is only the sub-system core excluding the padlogic. The padlogic has to be added at SOC level as shown in Figure 1. However, for the sake of clarity and completeness some details of the padlogic block is also added here in this document.

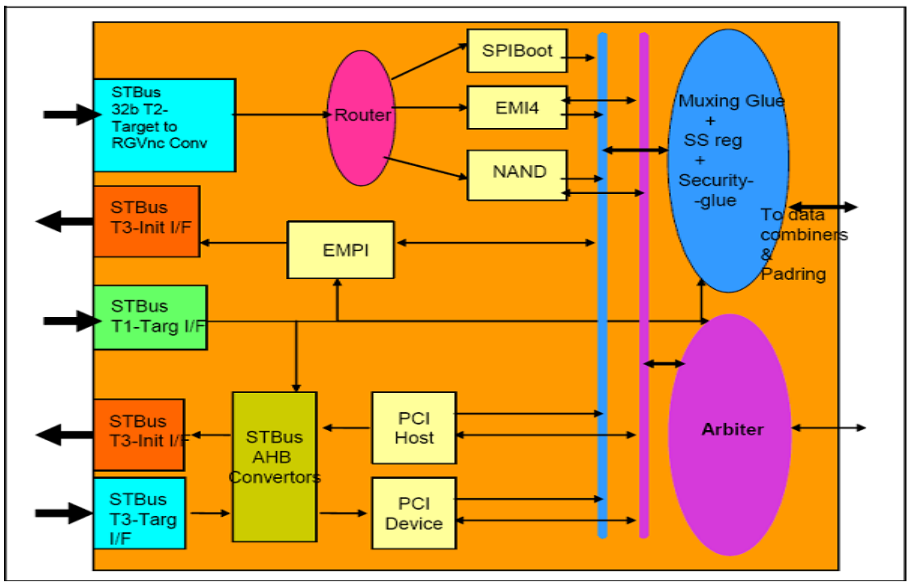


Figure 3.1: EMIPCISS Block diagram

## 3.3 Normal functional behavior

The sub-system core is described in this chapter. The chapter gives an overview of individual blocks. For the detailed description of the blocks instantiated to refer to the functional specification document of the corresponding IP. The glue and the arbitration logic are described in detail in this chapter. The interfaces of each IP and the issues in integrating the IP are also documented in this chapter. The IP details described in this section are as follows:

- EMI Buffer
- Router
- EMI4
- Serial Flash controller
- NAND integrated conrtoller
- BCH controller
- Hamming controller
- PCISS
- EMPI
- Arbiter
- Glue logic (internal to EMIPCISS) All the above IPs are inside the sub-system core which is delivered as a part of EMIPCISS. Figure 2 below shows the sub-system core with the different components.The padlogic glue is added on top of the sub-system core at the SOC level. Deyails related to the padlogic are given in section.
- Padlogic (external to EMIPCISS)

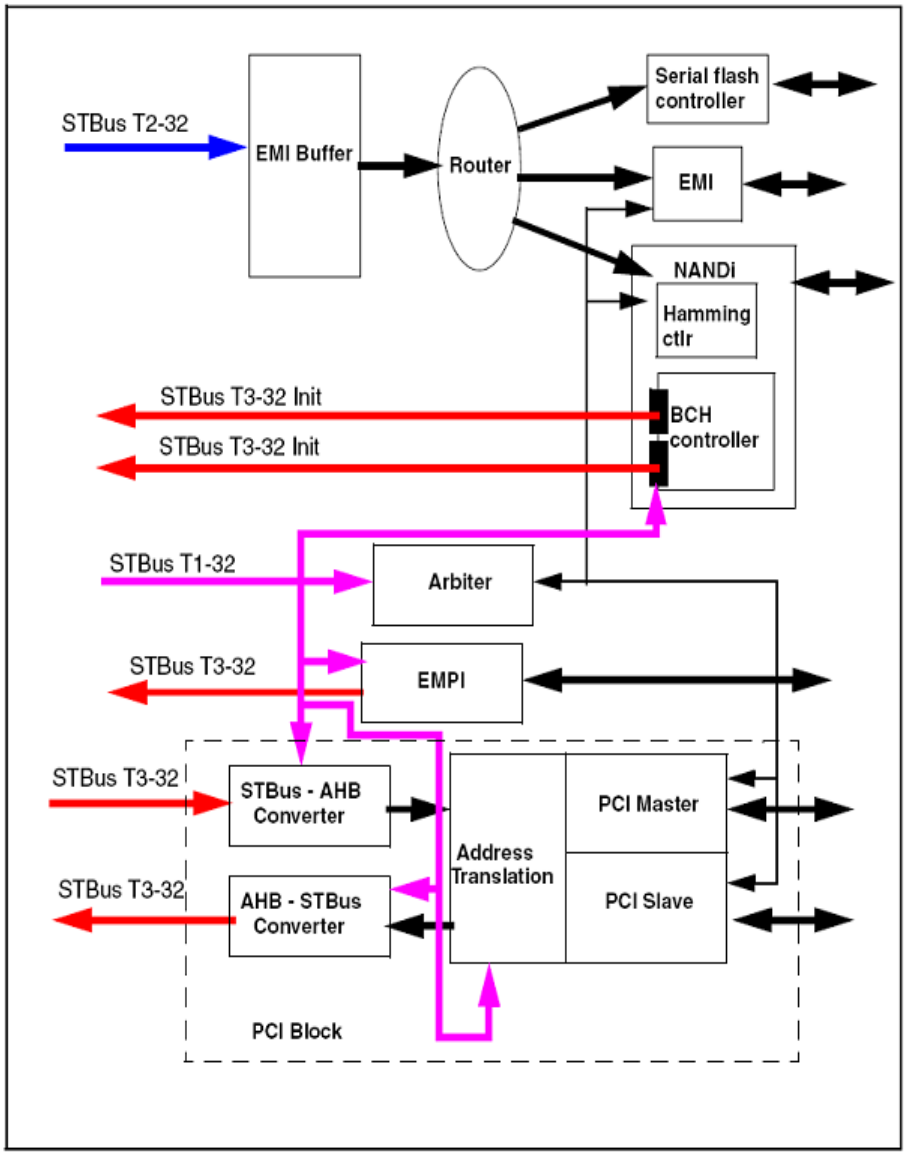


Figure 3.2: EMISS sub system core

## 3.4 EMI Buffer

The EMI buffer translates the STBus cells (received on T2-32 bit interface) into Request- Grant-Validnext cycle (RGVnc). It also houses registers that specify the top address of each bank. The EMI buffer houses transmit and receive buffers. The transmit buffer stores the cells received on STBus interface before they are mapped

onto RGVnc interface. The receive buffer stores the response data received from RGVnc interface before the responses are forwarded on to STBus interface.

- Router

  The router used is similar to the one described in 7200 FMI subsystem router Functional specification: ADCS 7966966. The router is responsible to route the transactions to SPI Boot, EMI or NAND flash controller. The router logic is expected to work on the following rules.

- If the current cell and next cell are to the same IP (that is either SPI Boot, EMI or nand flash controller), the cell will be forwarded without waiting for the valid_nextcycle signal.

- If the current and next cell are to different IPs, the next cell is forwarded a cycle after the valid_nextcycle assertion is detected.

- The router is packet based, which means that unless the reponse of all the cells of a packet are returned, the control is not changed to another IP.

  Following the above rules eliminates possibility of out of order responses on STBus interface.The memory mapped transactions to the boot bank are forwarded to the interface of the "boot_device" depending on the static pin values. The memory mapped transactions to any other bank are forwardef to the EMI4.

## 3.5 EMI4

EMI supports NOR flash (AMD, Intel or ST), MPX master, DVBCI and ATAPI interfaces. The DVBCI and ATAPI interfaces are supported using additional glue that is designed around the EMI. EMI also supports SDRAM interface. The SDRAM pins are not bristled out and hence cannot be supported by the sub-system.

## 3.6 Serial Flash controller

The SPI Boot IP supports booting from the serial flash, which could be ST type or Atmel type. The type of flash is configured by the static pin which is bristled out of the chip. After boot, the writing into the external serial flash is supported by SPI functionality in comms or through the new FSM mode in the SPI controller. The SPI boot pins are multiplexed with the SPI pins of the comms. This multiplexing is done at the top level and is outside the scope of the document. The subsystem bristles the all the pins required for serial communication.

Serial flash controller can work as per below modes as shown in fig:

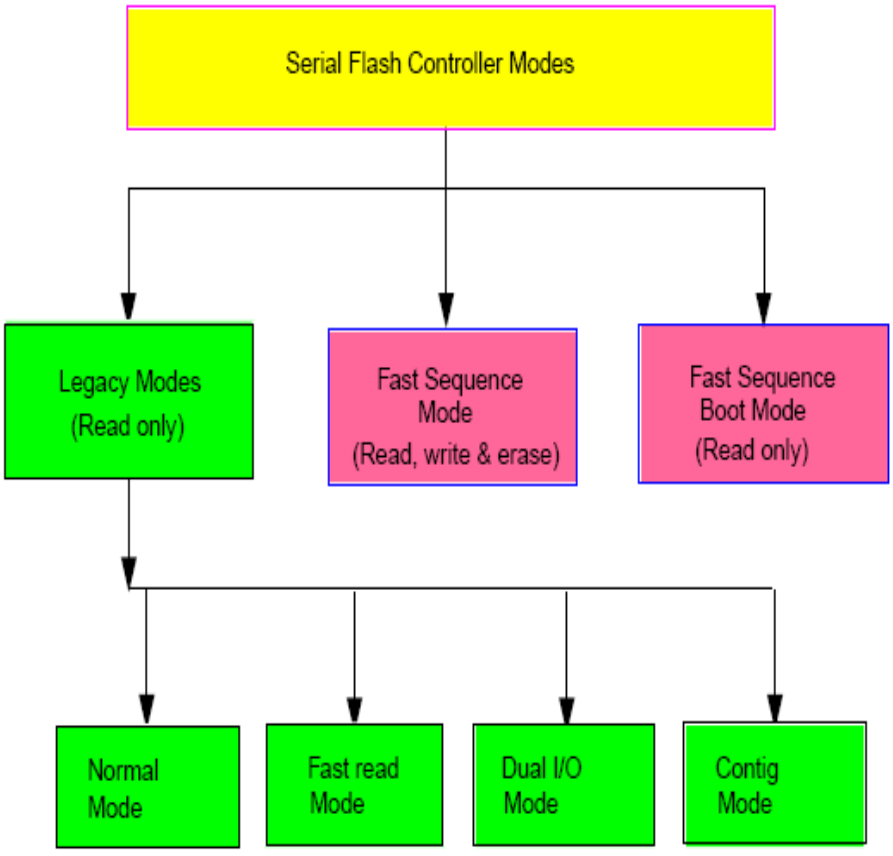


Figure 3.3: Serial flash controller modes

### 3.6.1 Legacy mode

- Normal mode: This mode supports read access to the serial Flash. All the SPI outputs are generated on the rising edge of the system clock and all the inputs are latched on the rising edge of the system clock.
- Contiguous mode: This is the performance enhancement mode. When enabled, it improves the performance of the controller for back-to-back contiguous accesses made to flash memory locations.
- Fast read mode: That is operations at higher frequencies. It allows reads at bit rates up to 50 MHz.
- Dual I/O mode: The data is output on two pins, DI and DO. This doubles the data transfer rate as compared to the normal mode

### 3.6.2 Fast sequence mode

It offers a flexible, software programmable engine which may be used to perform various serial Flash operations including read, write and erase.The FSM mode enables the controller to execute multiple command/address/data sequences in one go based on the configuration of sequence registers.

### 3.6.3 Fast sequence boot mode

This mode is designed to support x1, x2 and x4 booting.Fast sequence boot mode is designed to support x1, x2 and x4 booting.

## 3.7 NAND integrated controller

The NANDi controller consists of two independent NAND controllers known as Hamming NAND controller and BCH NAND controller. Both these controllers are described briefly in the sections below.

### 3.7.1 Hamming NAND controller

The Hamming NAND flash controller supports boot, flex mode and advanced flex modes. In all the modes the external NAND flash could be 8 or 16 bits. It could be small page or large page. The NAND device can accept 3 beats, 4 beats or 5 beats of address. All the configurations for boot mode are inferred from the value of static pins. When the external NAND device used is small page, 3 beats of address are generated when address_short_not_long is '1', else 4 beats of address are generated. In the case of large page NANDs 4 address beats are generated when 'address_short_not_long' is '1' else 5 address beats are generated. It is expected that only chip select don't care type of NAND devices are supported. In the current sub-system NAND device is supported on one Bank. This bank however can be configurable at boot and later can be re-configured by the software. The HAMMING controller is cabable of correcting one error per 128Bytes during boot mode. It can also detect 1 error in 512B in AFM mode. Correction in AFM mode is left on the software. The Nand controller always functions in slave mode. The arbiter is always assumed to have the bus ownership and nand controller will always work in slave mode. When the nand controller receives a request to access the external memory block from the router, it forwards the request to the arbiter by asserting the "nand_bus_req" signal. The arbiter grants the bus to the nand controller by asserting the "nand_bus_grant" signal. The controller starts accessing the memory block only when it receives the grant for the external shared bus. When the nand controller transaction is complete, it relinquishes the bus by deasserting the "nandbus_engaged_not_free" signal. The controller would request for the bus before every boot, flex or AFM transaction.

### 3.7.2 BCH NAND controller

The BCH NAND controller has been designed to support both MLC and SLC kind of devices. It supports boot and AFM modes. In boot mode only 8 bit parts are supported while in AFM mode 16bit parts are also supported. The BCH NAND

controller is equipped with dedicated hardware DMAs for efficient data transfers in AFM mode. During boot mode, static pins are sampled by the controller to determine the type of NAND flash it is connected to. The BCH controller can support 2K, 4K and 8K page flash devices. It also has an inherent error correcting BCH engine which can correct up to18b or up to 30b errors per 1KB depening on the configuration used. Devices with 512B pages cannot be supported by this controller. The controller is capable of interfacing with multi chip select devices. It also supports 2x8 bit configuration. Like the HAMMING controller, the BCH controller always functions in bus slave mode. It requests for the shared bus form the arbiter whenever it has a boot or AFMrequest to service.

## 3.8 PCISS

The PCI interface is supported by three blocks which are together known as PCISS.

- PCI IP purchased from Synopsys.
- STBus AHB converter from OCCS
- Glue to connect the STBus-AHB and AHB-PCI bridges. In addition to connecting the two bridges, the additional functionalities such as address translation are also implemented in the glue.

## 3.9 EMPI

The MPX slave interface is implemented using the EMPI. The MPX slave is never a bus master, hence does not request for the mastership of the bus. The external MPX master is expected to request for the bus mastership. However the glue around the EMPI has to be updated to support strobes on falling feature. The EMPI supports the modes described below.

### 3.9.1 EMPI modes supported

The EMPI is configured to support single chip select mode only. The chipselect[5] (or the chip select of last bank) is connected to the EMPI chip select. The EMPI within EMISS is configured to infer the burst size information from address bits 31 to 29. The address bits 63 to 61 are not required. Strobes on falling feature can be programmed to be enabled on EMPI. This is essentially done in the padlogic.

### 3.9.2 Arbiter

The arbiter is primarily responsible to arbitrate the requests from different agents (Interfaces) requesting the mastership of the bus and grant the mastership to a given interface. It always ensures that the bus contention does not occur when the bus mastership is changed. The primary purpose of the arbiter is to ensure that any of the bus agents does not lock the bus for a very long time. Latency critical requests, such as CPU accesses to the external peripheral viz., DVBCI are serviced with a greater priority to ensure an efficient overall system performance. The arbiter uses a bandwidth limit arbitration scheme instead of a simple bandwidth based arbitration scheme.

## 3.10 clocking

The sub-system could be configured as clock master or clock slave for synchronous flash and MPX modes. This is done by asserting appropriate logic on the pin 'emi_clock_slavenot master' at the boot. It could be configured either as the clock master or the clock slave in PCI mode. When PCI mode is supported, synchronous flash and MPX mode are not supported. The following four scenarios are envisaged for clocking.

- PCI clock master
- PCI clock slave

- MPX and Flash clock master
- MPX clock slave Each of the three clocking scenarios are described in following sections. The Figure below gives a picture of the clocking structure in EMIPCISS.

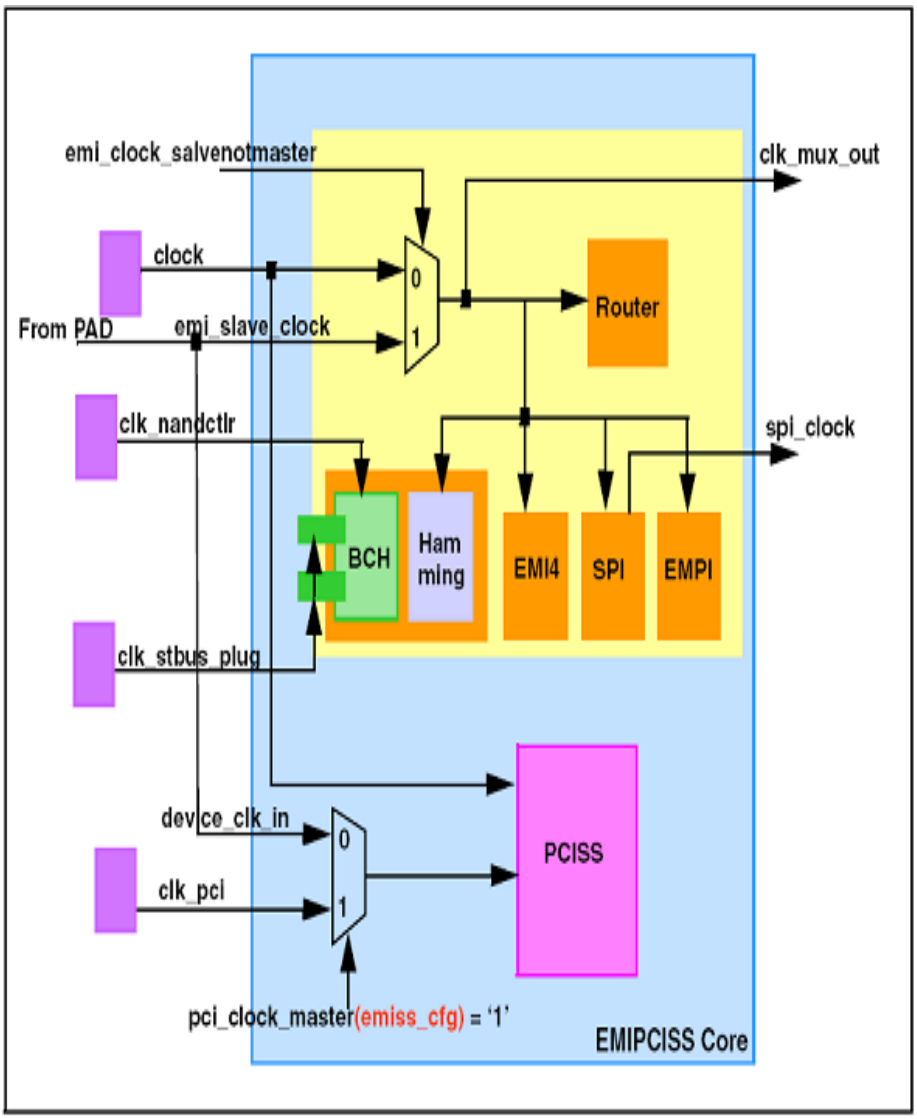


Figure 3.4: EMISS clocking

### 3.10.1 Frequencies

| clock name | C.F. Min | C.F. Typical | C.F. Max |
|---|---|---|---|
| Clock | 0 | 100MHz | 100MHz |
| Clock_pci | 0 | 33MHz | 66MHz |
| device_clock_in | 0 | 33MHz | 66MHz |
| clk_nandctrl | 0 | 200MHz | 200MHz |
| clk_stbus_plug | 0 | 200MHz | 200MHz |
| emi_slave_clock | 0 | 100MHz | 100MHz |

Table 3.1: clock names and frequencies
Where C.F. = Clock Frequency

### 3.10.2 Clock relationships

The sub-system has a total of 6 clocks as given in Table 10. When the EMISS is clock slave, emi_slave_clock is used instead of clock. When PCI is used in clock slave mode then device_clock_in is used instead of clock_pci. These scenarios are described in the sections below. There are essentially four clock domains in the EMISS- clock, clock_pci, clk_nandctrl and clk_stbus_plug. All the clocks are assumed to be of 50% duty cycle +/-10%. However there is no requirement for system clock to meet this constraint. The PCI clock requirements are defined in STBus-PCI bridge functional specification document. Appropriate re-timing of signals is performed within EMISS for signals that are crossing clock domains. The system clock is expected to have three branches, one fed to EMI, second fed to the padlogic and third fed to glue responsible for generating the Flash clock/MPX clock. It is essential that a timing relation is maintained between these three branches. The timing relation is described later.

### 3.10.3 PCI clock master

When the sub-system is configured as PCI clock master, the PCI clock is generated with in the on chip (from clock gen) is used instead of the one fed from the pads. A logic within the sub-system selects the PCI clock and feeds it to the clock out pad. The clock out pad is enabled by the logic within the sub-system core, so that the PCI clock is output from the chip.

### 3.10.4 PCI clock slave

When the sub-system is configured as PCI clock slave, the external glue selects the PCI clock fed from the pads instead of the one generated from the clock gen. A logic within the sub-system disables the clock out pad (by de-asserting the 'device_clock_en' pin. This tristates the clock pad. The sub-system clock is also generated by the clock gen. and is fed to the sub-system core and the padlogic. The constraints on the PCI clock are documented in the STBus-PCI bridge functional specifications. The constraints in the scenario of both PCI clock master and PCI clock slave are documented.

### 3.10.5 MPX and flash clocking

MPX and PCI modes are complimentary, hence PCI will not be used when MPX mode is selected. When the sub-system is configured as the clock master (by asserting a '0' on emi_clock_slavenotmaster), the clock from clock gen is fed to EMI and padlogic, the MPX clock (or flash clock ) is generated by the padlogic. The MPX clock generated is bristled out through the clock pad. The clock pad has to be configured as output. In the MPX and synchronous flash mode, the EMISS is configured as clock slave by asserting a logic '1' on pin 'emi_clock_slavenotmaster'. In this mode, the clock from the pad is fed to the EMI and padlogic. A glue external to the EMISS has to select the EMI STBus clock. The mapping of STBus transactions to the interconnect clock is outside the scope of the EMISS. The clock balancing has to be such that the delay

in the EMISS clock source point (output of mux) to clock input of the FF in EMI has to be same as the delay from clock source to clock input of FF (essentially clocked by curr_clock) in padlogic. The data at the pad output should be delayed by a very small amount (say 2ns) with respect to the rising edge of the clock at the pad.

# Chapter 4

# Memories used in EMISS

## 4.1 NAND Memory

The NAND16GW3D2A is a multilevel cell (MLC) device from the NAND flash 4224-byte page family of non-volatile flash memories. The NAND16GW3D2A has a density of 16 Gbits. The device operates from a 3 V power supply.

The address lines are multiplexed with the data input/output signals on a multiplexed x8 input/output bus. This interface reduces the pin count and makes it possible to migrate to other densities without changing the footprint.

Each block can be programmed and erased up to 10,000 cycles (with error correction code (ECC) on). The device also has hardware security features; a write protect pin is available to provide hardware protection against program and erase operations.

The devices feature an open-drain, ready/busy output that identifies if the program/erase/ read (P/E/R) controller is currently active. The use of an open-drain output allows the ready/busy pins of several memories to be connected to a single pull-up resistor. The memory array is split into 2 planes of 2048 blocks each. This multiplane architecture makes it possible to program 2 pages at a time (one in each plane), to erase 2 blocks at a time (one in each plane), or to read 2 pages at a time (one in each plane) dividing by two the average program, erase, and read times.

The device has the Chip Enable 'don't care' feature, which allows the bus to be shared between more than one memory at the same time, as Chip Enable transition during the latency time do not stop the read operation. Program and erase operations can never be interrupted by Chip Enable transition.

There is the option of a unique identifier (serial number), which allows the NAND16GW3D2A to be uniquely identified. It is subject to an NDA (non-disclosure agreement) and is, therefore, not described in the datasheet. For more details of this option contact your nearest Numonyx sales office. The device is available in TSOP48 (12  20 mm) package. and is shipped from the factory with block 0 always valid and the memory content bits, in valid blocks, erased to '1'.

### 4.1.1 NAND Memory feature lists

- High density multilevel cell (MLC) flash memory 16 Gbits of memory array 512 Mbits of spare area Cost-effective solutions for mass storage applications
- NAND interface x8 bus width Multiplexed address/data
- Supply voltage: VDD = 2.7 to 3.6 V
- Page size: (4096 + 128 spare) bytes
- Block size: (512K + 16K spare) bytes
- Multiplane architecture Array split into two independent planes All operations can be performed on both planes simultaneously
- Memory cell array (4 K + 128) bytes x 128 pages x 4096 blocks
- Page read/program Random access: 60 us (max) Sequential access: 25 ns (min) Page program operation time: 800 us (typ)
- Multipage program time (2 pages): 800 us (typ)
- Copy-back program Fast page copy

- Fast block erase Block erase time: 2.5 ms (typ)
- Multiblock erase time (2 blocks): 2.5 ms (typ)
- Status register
- Electronic signature
- Serial number option
- Chip enable 'don't care'
- Data protection Hardware program/erase locked during power transitions
- Development tools Error correction code models Bad block management and wear leveling algorithm HW simulation models
- Data integrity 10,000 program/erase cycles (with ECC) 10 years data retention

### 4.1.2 NAND flash memory Logic diagram and pin description

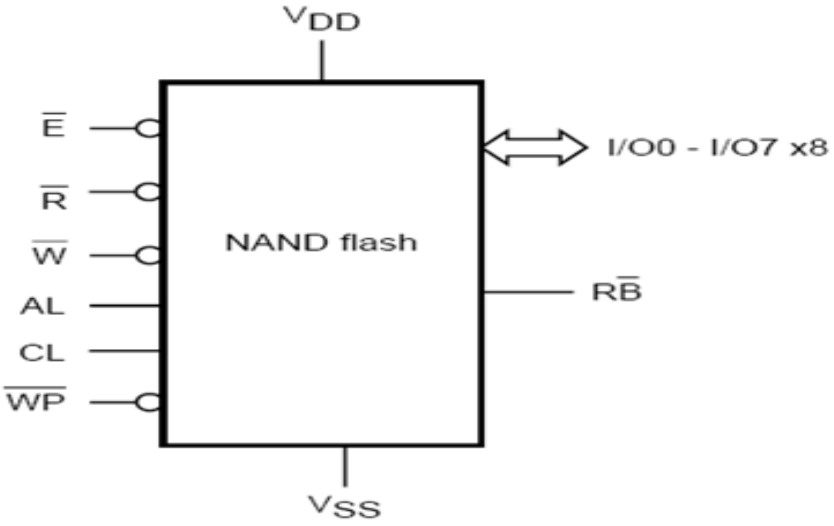


Figure 4.1: Logic Diagram of NAND flash memory

| Signal | Function | Direction |
|---|---|---|
| I\O0 - I\O7 | Data input\outputs | Input\output |
| CL | Command latch enable | Input |
| AL | Address latch enable | Input |
| E(bar) | Chip enable | Input |
| R(bar) | Read enable | Input |
| W(bar) | Write enable | Input |
| WP(bar) | Write protect | Input |
| RB(bar) | Ready\Busy (open drain output) | Output |
| VDD | Power supply | Power supply |
| VSS | Ground | Ground |
| N.C. | Not connected | - |
| DU | Do not used | - |

Table 4.1: Signal Names And its Function of NAND flash memory

### 4.1.3 Bus Operation

- Command input

  Command input bus operations give commands to the memory. Commands are accepted when Chip Enable is Low, Command Latch Enable is High, Address Latch Enable is Low and Read Enable is High. They are latched on the rising edge of the Write Enable signal. Only I/O0 to I/O7 are used to input commands.

- Address input

  Address input bus operations input the memory addresses. Five bus cycles are required to input the addresses. The addresses are accepted when Chip Enable is Low, Address Latch Enable is High, Command Latch Enable is Low and Read Enable is High. They are latched on the rising edge of the Write Enable signal. Only I/O0 to I/O7 are used to input addresses.

- Data input

  Data input bus operations input the data to be programmed. Data is only accepted when Chip Enable is Low, Address Latch Enable is Low, Command

Latch Enable is Low and Read Enable is High. The data is latched on the rising edge of the Write Enable signal. The data is input sequentially using the Write Enable signal.

- Data output

  Data output bus operations read the data in the memory array, the status register, the electronic signature, and the unique identifier. Data is output when Chip Enable is Low, Write Enable is High, Address Latch Enable is Low, and Command Latch Enable is Low. The data is output sequentially using the Read Enable signal. If the Read Enable pulse frequency is lower then 33 MHz (tRLRL higher than 30 ns), the output data is latched on the rising edge of Read Enable signal For higher frequencies (tRLRL lower than 30 ns), the extended data out (EDO) mode must be considered. In this mode, data output is valid on the input/output bus for a time of tRLQX after the falling edge of Read Enable signal

- Write protect

  Write protect bus operations protect the memory against program or erase operations. When the Write Protect signal is Low the device does not accept program or erase operations, therefore, the contents of the memory array cannot be altered. The Write Protect signal is not latched by Write Enable to ensure protection, even during power-up.

- Standby

  The memory enters standby mode by holding Chip Enable, E, High for at least 10 us. In standby mode, the device is deselected, outputs are disabled and power consumption is reduced.

- Command set All bus write operations to the device are interpreted by the command interface. The commands are input on I/O0-I/O7 and are latched

on the rising edge of Write Enable when the Command Latch Enable signal is High. Device operations are selected by writing specific commands to the command register. The two-step command sequences for program and erase operations are imposed to maximize data security.

| Function | 1st cycle | 2nd cycle | 3rd cycle | 4th cycle | Acceptable during Command busy |
|---|---|---|---|---|---|
| Page read | 00h | 30h | | | |
| Read for copy back | 00h | 35h | | | |
| Read ID | 90h | | | | |
| Reset | FFh | | | | Yes |
| Page Program | 80h | 10h | | | |
| Multiple page program | 80h | 11h | 81h | 10h | |
| Multiple read | 60h | 60h | 30h | | |
| Copy back program | 85h | 10h | | | |
| Multiple copy back program | 85h | 11h | 81h | 10h | |
| Multiple copy back read | 60h | 60h | 35h | | |
| Block erase | 60h | D0h | | | |
| Multiple block erase | 60h | 60h | D0h | | |
| Read status register | 70h | | | | Yes |
| Random data input | 85h | | | | |
| Random data output | 05h | E0h | | | |
| Multiple random data output | 00h | 05h | E0h | | |

Table 4.2: command sets and its hex condition

## 4.2 NOR flash Memory

### 4.2.1 NOR Flash Memory Features List

- Single Voltage, 2.7V to 3.6V for Read and Write operations
- Organization 4,194,304 x 8 bit (Byte mode) / 2,097,152 x 16 bit (Word mode)

- Fast Read Access Time : 70ns
- Read While Program/Erase Operation
- Dual Bank architectures

  Bank 1 / Bank 2 : 8Mb / 24Mb
- Secode(Security Code) Block : Extra 64K Byte block
- Power Consumption (typical value @5MHz) Read Current : 14mA Program/Erase Current : 15mA Read While Program or Read While Erase Current : 25mA Standby Mode/Auto Sleep Mode : 5uA
- WP/ACC input pin Allows special protection of two outermost boot blocks at VIL, regardless of block protect status Removes special protection of two outermost boot block at VIH, the two blocks return to normal block protect status Program time at VHH : 9us/word
- Erase Suspend/Resume
- Unlock Bypass Program
- Hardware RESET Pin
- Command Register Operation
- Block Group Protection / Unprotection
- Supports Common Flash Memory Interface
- Industrial Temperature : -40C to 85C
- Endurance : 100,000 Program/Erase Cycles Minimum
- Data Retention : 10 years
- Package : 48 Pin TSOP1 : 12 x 20 mm / 0.5 mm Pin pitch 48 Ball TBGA : 6 x 8.5 mm / 0.8 mm Ball pitch 48 Ball FBGA : 6 x 8.5 mm / 0.8 mm Ball pitch

### 4.2.2 NOR flash Memory logic diagram and pin description

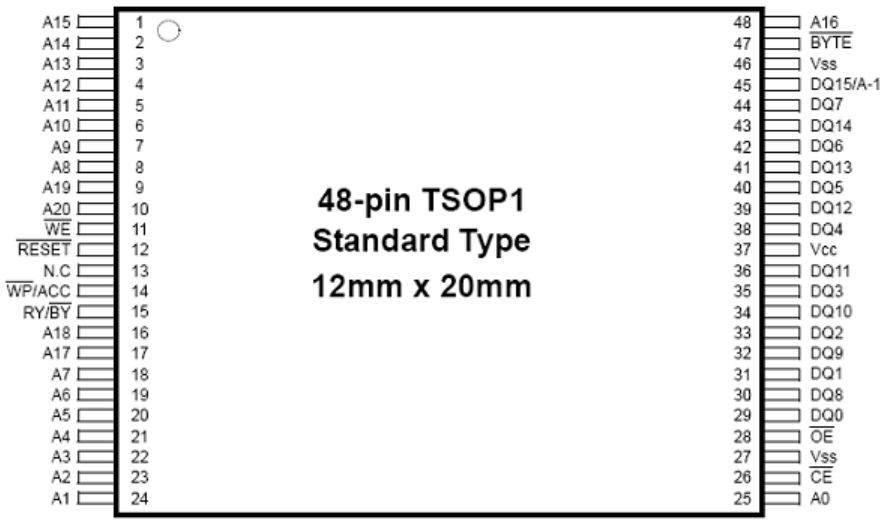


Figure 4.2: Signals and pin descriptions of NOR flash memory

### 4.2.3 Bus Operation

- Byte/Word Mode

  If the BYTE pin is set at logical "1" , the device is in word mode, DQ0-DQ15 are active. Otherwise the BYTE pin is set at logical "0" the device is in byte mode, DQ0-DQ7 are active. DQ8-DQ14 are in the High-Z state and DQ15 pin is used as an input for the LSB (A-1) address pin.

- Read Mode

  The K8D3216U is controlled by Chip Enable (CE), Output Enable (OE) and Write Enable (WE). When CE and OE are low and WE is high, the data stored at the specified address location,will be the output of the device. The outputs are in high impedance state whenever CE or OE is high.

- Standby Mode

| PIN NAME | PIN FUNCTION |
|---|---|
| A0-A20 | Address inputs |
| DQ0-DQ10 | Data inputs backslash outputs |
| Byte(bar) | Word\Byte(bar) selection |
| CE(bar) | chip enable |
| OE(bar) | output enable |
| RESET(bar) | hardware reset pin |
| RY backslash BY(bar) | Ready\Busy output |
| WE(bar) | Write enable |
| WP(bar) backslash ACC | Hardware write protection \program Acceleration |
| Vcc | power supply |
| Vss | ground |
| N.C. | Not connected |

Table 4.3: Signal Names And its Function of NOR flash memory

The features Stand-by Mode to reduce power consumption. This mode puts the device on hold when the device is deselected by making CE high (CE = VIH). Refer to the DC characteristics for more details on stand-by modes. Output Disable The device outputs are disabled when OE is High (OE = VIH). The output pins are in high impedance state.

- Automatic Sleep Mode

  The features Automatic Sleep Mode to minimize the device power consumption. Since the device typically draws 5uA of the current in Automatic Sleep Mode, this feature plays an extremely important role in battery-powered applications. When addresses remain steady for tAA+50ns, the device automatically activates the Automatic Sleep Mode. In the sleep mode, output data is latched and always available to the system. When addresses are changed, the device provides new data without wait time.

- Autoselect Mode

  This flash offers the Autoselect Mode to identify manufacturer and device type by reading a binary code. The Autoselect Mode allows programming equipment

to automatically match the device to be programmed with its corresponding programming algorithm. In addition, this mode allows the verification of the status of write protected blocks. This mode is used by two method. The one is high voltage method to be required VID (8.5V 12.5V) on address pin A9. When A9 is held at VID and the bank address or block address is asserted, the device outputs the valid data via DQ pins(see Table 9 and Figure 2). The rest of addresses except A0, A1 and A6 are Donut Care. The other is autoselect command method that the autoselect code is accessible by the commamd sequence without VID. The manufacturer and device code may also be read via the command register. The autoselect operation of block protect verification is initiated by first writing two unlock cycle. The third cycle must contain the bank address and autoselect command (90H). If Block address while (A6, A1, A0) = (0,1,0) is finally asserted on the address pin, it will produce a logical "1" at the device output DQ0 to indicate a write protected block or a logical "0" at the device output DQ0 to indicate a write unprotected block. To terminate the autoselect operation, write Reset command (F0H) into the command register..

- Write (Program/Erase) Mode

  The K8D3216U executes its program/erase operations by writing commands into the command register. In order to write the commands to the register, CE and WE must be low and OE must be high. Addresses are latched on the falling edge of CE or WE (whichever occurs last) and the data are latched on the rising edge of CE or WE (whichever occurs first). The device uses standard microprocessor write timing.

- Program

  This can be programmed in units of a word or a byte. Programming is writing 0's into the memory array by executing the Internal Program Routine. In order to perform the Internal Program Routine, a four-cycle command sequence is necessary. The first two cycles are unlock cycles. The third cycle is assigned

for the program setup command. In the last cycle, the address of the memory location and the data to be programmed at that location are written. The device automatically generates adequate program pulses and verifies the programmed cell margin by the Internal Program Routine. During the execution of the Routine, the system is not required to provide further controls or timings. During the Internal Program Routine, commands written to the device will be ignored. Note that a hardware reset during a program operation will cause data corruption at the corresponding location.

- Block Erase

  To erase a block is to write 1us into the desired memory block by executing the Internal Erase Routine. The Block Erase requires six bus cycles to write the command sequence shown in Table 8. After the first two "unlock" cycles, the erase setup command (80H) is written at the third cycle. Then there are two more "unlock" cycles followed by the Block Erase command. The Internal Erase Routine automatically pre-programs and verifies the entire memory prior to erasing it. The block address is latched on the falling edge of WE or CE, while the Block Erase command is latched on the rising edge of WE or CE. Multiple blocks can be erased sequentially by writing the six bus-cycle operation in Figure. Upon completion of the last cycle for the Block Erase, additional block address and the Block Erase command (30H) can be written to perform the Multi-Block Erase. 50us (typical) "time window" is required between the Block Erase command writes. The Block Erase command must be written within the 50us "time window", otherwise the Block Erase command will be ignored. The 50us "time window" is reset when the falling edge of the WE occurs within the 50us of "time window" to latch the Block Erase command. During the 50us of "time window", any command other than the Block Erase or the Erase Suspend command written to the device will reset the device to read mode. After the 50us of "time window", the Block Erase command will

initiate the Internal Erase Routine to erase the selected blocks. Any Block Erase address and command following the exceeded "time window" may or may not be accepted. No other commands will be recognized except the Erase Suspend command during Block Erase operation.

## 4.3 SPI flash Memory

### 4.3.1 SPI flash Memory feature lists

- serial peripheral interface compatible- mode 0 and mode 3 268,435,456x1 bit structure or 134,217,728x2 bits (two i/o mode) structure or 67,108,864 x 4 bits (for i/o mode structure)
- power supply operation 2.7 to 3.6 volt for read,erase and program operation
- latch up protected to 100mA from 1V to Vcc+1V
- high performance VCC=2.7 3.6V
- normal read 50MHZ
- Fast read 1 I/O: 80 MHZ with 8 dummy cycles 2 I/O: 70 MHZ with 4 dummy cycles 4 I/O: 70 MHZ with 6 dummy cycles
- fats program time: 1.4ms(typ.) and 5 ms(max.)/page(256-byte per page)
- Low power consumption
- typical 100000 erase /program cycles

| SYMBOL | DESCRIPTION |
|---|---|
| CS | Chip select |
| SI\ | SIO0 & Serial Data input\output |
| SO\SIO1\PO7 | serial Data input\output parallel data output |
| SCLK | clock input |
| WP/SIO2 | write protection:connect to GND or serial data input\output |
| GND | ground |
| PO0 ~PO6 | parallel data output\input |
| N.C. | Not connected |

Table 4.4: flash memory pin description and its functions

### 4.3.2 SPI flash Memory logic diagram and pin description

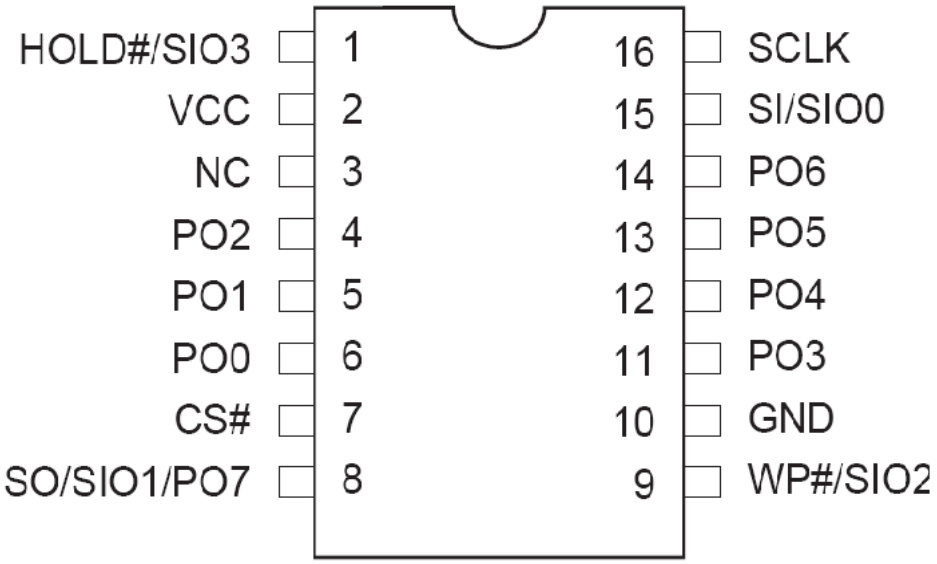


Figure 4.3: SPI flash memory pin diagram

### 4.3.3 SPI flash memory Bus operations

- CS (chip select): This signal will enable the chip
- SCLK (serial clock): clock input

| Mode | SIO0 | SIO1 | SIO2 | SIO3 |
|---|---|---|---|---|
| 1 | Input | Output | WP to GND | Hold |
| 2 | I\P & O\P | I\P & O\P | WP to GND | Hold |
| 4 | I\P & O\P | I\P & O\P | I\P & O\P | I\P & O\P |

Table 4.5: Input / Output Data Table

- PO (parallel data input/output): It will not be connected in serial data operation
- WP(write protect): It will be connected to ground

# Chapter 5

# MCC Cache Memory

The purpose of inserting the cache for inter prediction motion compensation is to reduce the typical bandwidth required for prediction, keeping the worst case band width smaller or same as without cache memory. The bandwidth has been studied on Allegro streams and the compromising size/associativity and complexity has been found: 8-ways associative, 32- byte or 16 bytes word, cache size is 128 KBytes. The one ways was simplier in term of implementation, but it creates a bottleneck which can slow down the cache.

This motion compensation cache (MCC) is inserted between the XPred memory access and the STBus. It can be bypassed, in which case the XPred accesses the external memory in the same manner as in previous DELTA versions. In this case, XPred STBus plugs are directly connected to STBus .

## 5.1 cache mode specification

- The cache mode has been specified having in mind next constraints: XPred is "latency independent", which means we may have 2 MBs in the pipeline between the request by the XPred and the data coming back to XPred.

- Chunks bordered inside the MB and having max size defined by user registers

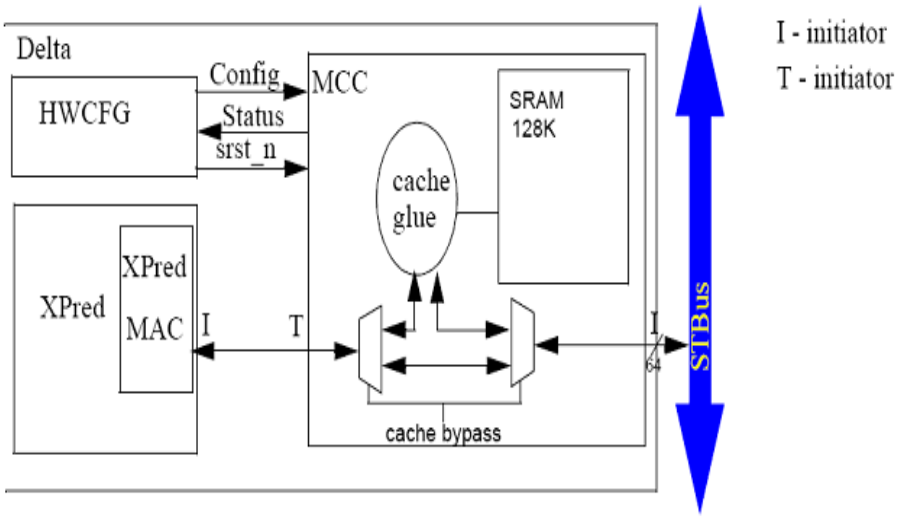


Figure 5.1: MCC cache block diagram

should be respected, even if some of data from original chunk requested by XPred are in the cache

- For DDR efficiency, The cache access can be either LD32 or LD16.
- cache is working on STBus clock to avoid multiple clock domain crossing and to be able to use the MCC as embedded SRAM even though DELTA is not used and its clock disabled.
- Tthe target frequency is 300 Mhz in 45 nm. Xpred request can be read every 2 cycles clocks. Xpred data can be sent by MCC every clock cycle.

## 5.2 Cache general description

Requests from XPred are processed one by one with 2 cycles per request. At this stage, MCC verifies if the requested pixels are in the cache or not. Nothing is read/written from/to cache at this stage. In case of cache miss, a 32-byte memory word in cache is reserved. In any case of hit/miss, number of outstandig reads at given cache address

is incremented. If there is a cache miss:the request is put in ReqFIFO (request FIFO), from which the requests will be sent to DDR, once the chunk is completed

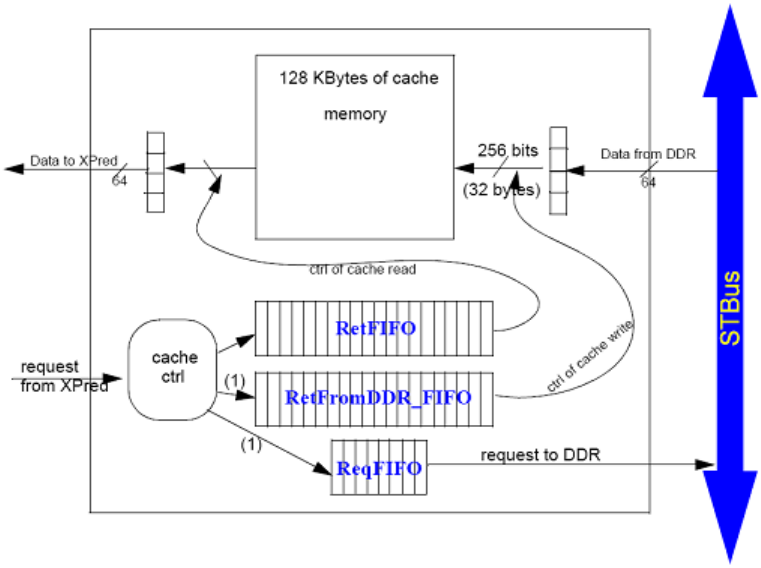


Figure 5.2: Memory cache block diagram

the cache address where the pixels will be written to, is put in RetFromDDR_FIFO (return from DDR FIFO) used to control the "write in cache" process In any case (hit or miss), the address where the pixels are (or will be after DDR access) is stored in RetFIFO (return to XPred FIFO). This FIFO is used to control the "read from cache" process Pixels requested by XPred are either in cache (cache hit) or not (cache miss) and they have to be accessed from the DDR. In this last case pixels are stored in the cache when back from the DDR. In any of these two cases, pixels are read from the cache before being sent to the XPred. To make the implementation uniform and as small as possible, 8 memory cuts of 4096x32bits each have been chosen. The control logic uses a 2x512x76 bits table, having tags,u_ord,in_cpt. The way this ctrl table is used will be described later in text. In addition, there are few FIFOs (DPreg) used to store requests from XPred: 512x12bits, 512x17bits and 16x28bits

## 5.3 Cache Control

- Cache control used a control table that is used to store different parameters for each of words in cache. This table contains in_cpt, V bits, U_ord and address tag for each word in cache.

- V: Valid bit is 0 if the word in cache has not yet been used for the picture (after reset). Once it has been used at least once, it remains to 1, which means there is a valid word that may be used for MC, which can only be replaced by another valid word.

- in_:cpt counts the number of times a word in cache is to be read. In other words, it counts the number of outstanding requests from XPred for a given word. Or again, it counts the number of times same word is used in the pipeline: XPred - Cache - DDR - Cache - XPred

- u_ord: corresponds to used order, each time a Xpred request is granted, the used order of each bank are refreshed. Then when there is a cache miss, the corresponding request can reserve the word in cache which has been least recenlty used. control table can be updated by 2 differents processes: the "Hit/Miss Detection" and the "Read from cache" process. dual port memory should be used for ctrl table. One port is used for "cache hit/miss" process and one port for "read from cache process". In addition to keep the right processing rate two "cache hit/miss process" are running in parallel, as well as two "read from cache process".

In above section, 128KB memory is used. If 128KB memory is used then the area of memory will increase.so we will have to reduce the area and so we will have to reduce the memory of the cache from 128KB to 64KB or 32KB.

This second generation of cache, uses an input buffer in order to be STbus latency independant whatever the cache size. Because when the cache memory become smaller than 128K, cache data collisions can introduce some delay on request sent

to STbus which are not acceptable. with an input buffer we have no more collision possible.

## 5.4 Cache Performance Tables

| **32 K Cache efficiency in term of Bandwidth & Opcode** | | | | | |
|---|---|---|---|---|---|
| | | **without cache** | | **128K (075mm2 in 40n)LRU** | |
| | | **BW** | **nb Opcode** | **BW** | **nb Opcode** |
| **Allergo Streams** | Max | 809 | 46813 | 862(+6%) | 26939(-42%) |
| | Avg | 487 | 24538 | 275(-44%) | 8587(-65%) |
| **DVD Streams** | Max | 474 | 24814 | 288(-39%) | 8993(-63%) |
| | Avg | 245 | 12332 | 191(-22%) | 5967(-51%) |
| **Real Life Streams** | Max | 195 | 10077 | 206(+5%) | 6444(-36%) |
| | Avg | 106 | 4935 | 102(-3%) | 3186(-35%) |

Table 5.1: 32 K Cache efficienty in term of Bandwidth & Opcode

| **64 K Cache efficiency in term of Bandwidth & Opcode** | | | | | |
|---|---|---|---|---|---|
| | | **without cache** | | **128K (075mm2 in 40n)LRU** | |
| | | **BW** | **nb Opcode** | **BW** | **nb Opcode** |
| **Allergo Streams** | Max | 809 | 46813 | 862(+6%) | 26939(-42%) |
| | Avg | 487 | 24538 | 271(-44%) | 8467(-65.5%) |
| **DVD Streams** | Max | 474 | 24814 | 288(-39%) | 8987(-63%) |
| | Avg | 245 | 12332 | 183(-25%) | 5703(-53%) |
| **Real Life Streams** | Max | 195 | 10077 | 192(-1.5%) | 5995(-40%) |
| | Avg | 106 | 4935 | 86(-18%) | 2693(-45%) |

Table 5.2: 64K Cache efficienty in term of Bandwidth & Opcode

| 128 K Cache efficiency in term of Bandwidth & Opcode | | | | | |
|---|---|---|---|---|---|
| | | without cache | | 128K (075mm2 in 40n)LRU | |
| | | BW | nb Opcode | BW | nb Opcode |
| **Allergo Streams** | Max | 809 | 46813 | 862(+6%) | 26935(-42%) |
| | Avg | 487 | 24538 | 258(-47%) | 8051(-67%) |
| **DVD Streams** | Max | 474 | 24814 | 267(-43%) | 8340(-66%) |
| | Avg | 245 | 12332 | 151(-38%) | 4709(-61%) |
| **Real Life Streams** | Max | 195 | 10077 | 147(-24%) | 4603(-54%) |
| | Avg | 106 | 4935 | 83(-21%) | 2608(-47%) |

Table 5.3: 128K Cache efficienty in term of Bandwidth and Opcode

## 5.5 Cache Area Estimation

a. A quick Webgen exploration give the following area for 128K cache in 40nm:

SRAM 4096x64=0.117 (4 cuts =0.468)

DPRAM 1024x32 =0.044mm2 (5 cuts = 0.222)

DPREG 512x17 =0.011mm2

DPREG 512x13 =0.009mm2

DPREG 128x64 = 0.011mm2

DPREG 16x30 = 0.003mm2 (should probably implemented with Flip-flops)

it means a total of about **0.72 mm2** for memories in 40 nm

a. For 32 KB cache

For 32 KB cache with fifo replacement and input buffer in 40nm:

DPREG 2x416x128 =0.090(input buffer)

SRAM 1024x32 =0.019 (8 cuts = 0.15) or 1024x128=0.069 (2 cuts=0.14) (cache)

DPREG 256x112 =0.027(cache control)

DPREG 512x15 =0.01

DPREG 128x20 =0.005

DPREG 16x30 = 0.003mm2 (should probably implemented with Flip-flops)

It means a total of about **0.27 mm2** for memories in 40 nm

a. For 64 KB cache with fifo replacement and input buffer in 40nm:

DPREG 2x416x128 =0.090(input buffer)

SRAM 1024x32 =0.019 (8 cuts = 0.15) or 2048x128=0.13 (2 cuts=0.26) (cache)

DPREG 512x104 =0.048(cache control)

DPREG 512x15 =0.01

DPREG 128x20 =0.005

DPREG 16x30 = 0.003mm2 (should probably implemented with Flip-flops)

it means a total of about **0.41 mm2** for memories in 40 nm with input buffer, XpredMac could be simplified by removing a part of its memory (replace 288x176 by 42x176). but in that case the bypass mode of the cache should still use input buffer, which is not planed yet.

Cache control converts also the Load32 or Load16 from external memory or cache into Load8 or Load16 when sending data to XPred. The address corresponding to the opcode, received from XPred MAC is considered in next way by the cache:

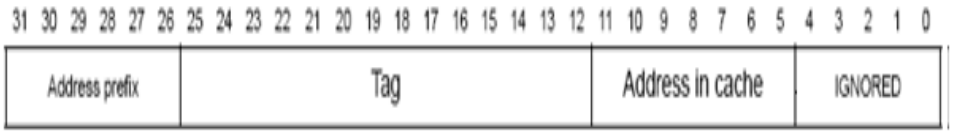


Figure 5.3: 32 Kbytes cache with cache line 32 bytes

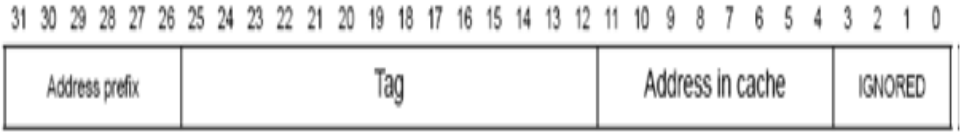


Figure 5.4: 32 Kbytes cache with cache line 16 bytes

To simplify the design, the prediction address space should be contiguous. Having the max DPB size in mind, the Address prefix may be ignored by the cache control hardware. The maximum range of address that Cache can access is defined by max DPB size = 6 (pict) x 8160(MBs) x 384(byte per MB) = 18.8MBytes. But to secure future application the cache can work in 64 Mbytes memory range it explains why tag+address_byte is on 26 bits. A functional schema of cache control system is shown here below. There are 128 addresses in the cache 32 bytes and 256 addresses in cache 16 bytes, which corresponds to 10 MBs . In orther words, if we access all block of pixels one after another, same cache address will be used only after 10 MBs. The 8 memories having the same cache address space (going from 0 to 127) in LD32 configuration, and 4 memories having the same cache address space (going from 0 to 127) and 4 memories having the same cache address space (going from 128 to 255) in LD16 configuration. with each address, it corresponds 8-way associativity, will be called banks. A content of any address in DDR may be at single address in cache (equal to DDR addr(11:4)), but in any of eight banks. To know to which DDR address the content of cache address correspond to, tag part of DDR address (equal to DDR address(25:12)) is stored together with each 32-byte or 16 byte data.

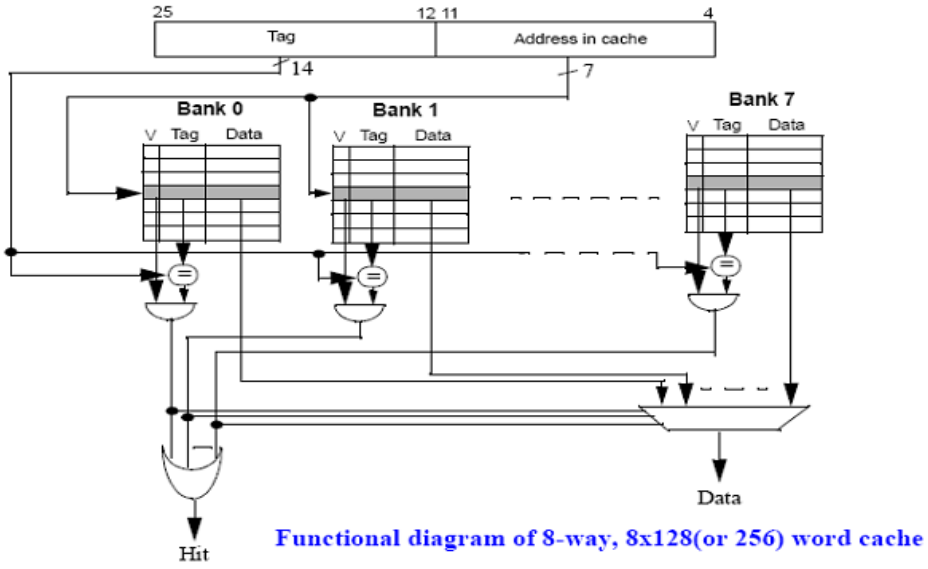


Figure 5.5: Functional Diagram

When valid bit V is one and the "Tag" part of address is found, the word is considered "Hit" and read from the cache. If not, the word is fetched from external memory and written on the least recently written word of 8 having the same address in the cache as address[11:5] bits of DDR.

## 5.6 Functional Specification

In this specification, The cache is verified is verification engineer. If there is any bug in the design then verification engineer will find it out and will inform about the bug to the designer. Designer will modify the design and try to make it correct.

In MCC cache below are the tests which have run to check the design and as per the results of the tests modification are done to design.

### 5.6.1 Cache Glue and Reset

Check that MCC is bypassed when bypass bit is set.

Check that on soft reset pulse on srst_n:

- All the state machines go back to idle state
- The outstanding STBUS transactions are properly terminated before soft reset is applied to internal logic
- Status output port (MCC_X_CNT) are cleared
- The valid (V) control bits stored in SRAM are reseted.
- When STBUS transactions are terminated and soft reset finished MCC_SRSEND goes high during the soft reset initialization the Xpred requests are not taken into account (grant is forced to 0)
- At the end of soft reset the configuration is stored in internal registers.

### 5.6.2 Cache Control

- Proper Hit& Miss detection, Check that on Cache HIT, the requests are not sent to DDR. The data should be returned to XPred from internal cache memory.
- Lock management: Check the functionality of lock management block. Generate traffic to hit same cache address for the parallel running processes: Cache Hit & Miss P1, Cache Hit & Miss P2, Cache Read P1
- Check that the entire cache memory is accessible and can be filled up resulting in cache hit for all subsequent transactions within the specified address range.
- Replacement policy: Check that when all sets for a cache address are valid, the least recently used cache word is chosen for replacement in case a miss occurs for next transaction on same cache address.
- In order response: Check that cache always returns data in order of request.
- Check that data coherency is maintained: data returned by cache for any address should match memory content at that address.
- Check proper cache functioning in case of cache miss and all eight banks valid with pending request greater then 0.
- Check proper cache functioning in case of cache hit and if corresponding number of pending request = 15.
- Check that cache write gets higher priority, if cache read & write occur at same cache address.

## 5.7 Verification Environment

This verification environment consists of following components:

- Everest EVC for driving/observing all STBus Ports.

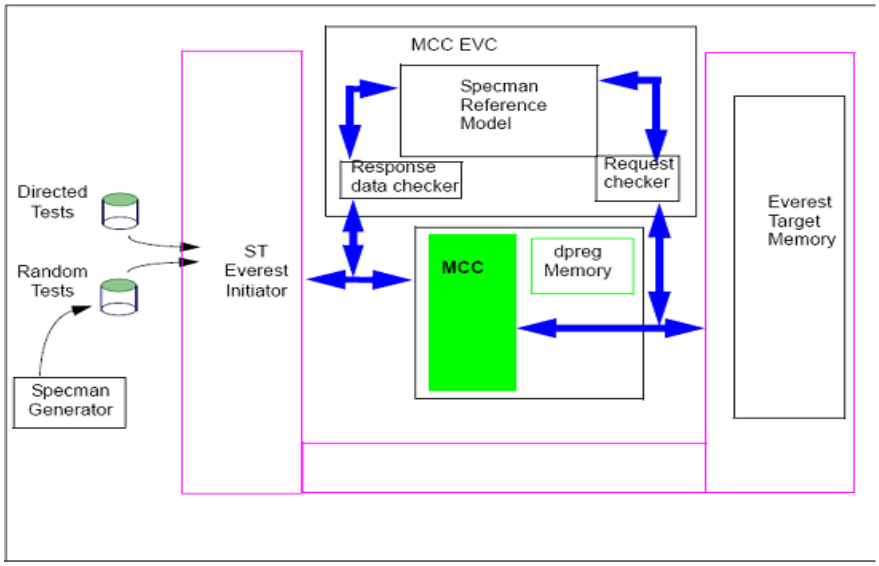


Figure 5.6: Verification Environment

- VHDL Test bench instantiating the Device under test (DUT)
- Specman test generator for random tests generation.
- Directed tests.

### 5.7.1 Verification Platform

The verification platform consists of Everest EVC, MCC EVC and HDL test bench. Everest EVC implements XPred initiator and system memory target ports. MCC EVC has a cache reference model and checkers to compare rtl and reference model transactions.

#### Specman Reference Model

A reference model for delta cache is implemented in Specman. The model closely follows functional specification. The reference model has following components and functions:

- Control table struct - implements control table bits and valid bits for each cache address
- Cache mem struct - implements cache memory, with variable cache line size 16 or 32
- Fifos - ReqFifo, RetFifo, RetFromDDRFifo, STBus Rx Fifo - used to store transactions.
- ProcessCacheHitMissDetect: The process detects if the data for incoming request is in cache (cache hit), or has to be requested from DDR (cache miss). It then populates the fifos as per spec. and updates control table. Two cache hit miss detect process run in parallel to handle Xpred requests every 2 cycles
- ProcessCacheRead: It reads data from cache and stores the response data in RxRsp fifo.
- ProcessCacheWrite: On receiving response from DDR, it writes data in cache.
- LruUpdate: The function updates 'used_order' field in control table according to functional spec. and returns least recently used 'way' for replacement in case of cache miss.
- Performance counters: counters for input opcodes received for a picture frame, cache hit or miss and output opcodes generated by cache on DDR initiator ports are implemented.

### MCC EVC

A MCC EVC wrapper is written on top of cache reference model. This block translates STBus transactions to cache transactions which are driven on reference model ports. The EVC implements checkers for comparing request and response transactions. Following checks are implemented:

- Request generated by RTL should match reference model requests on DDR initiator port
- Response generated by RTL should match reference model response on target port
- Response Data should always match contents of memory for the given address

## 5.8 HDL Testbench Architecture

The HDL test bench has MCC RTL along with webgen memories and a hwcfg block to configure cache.

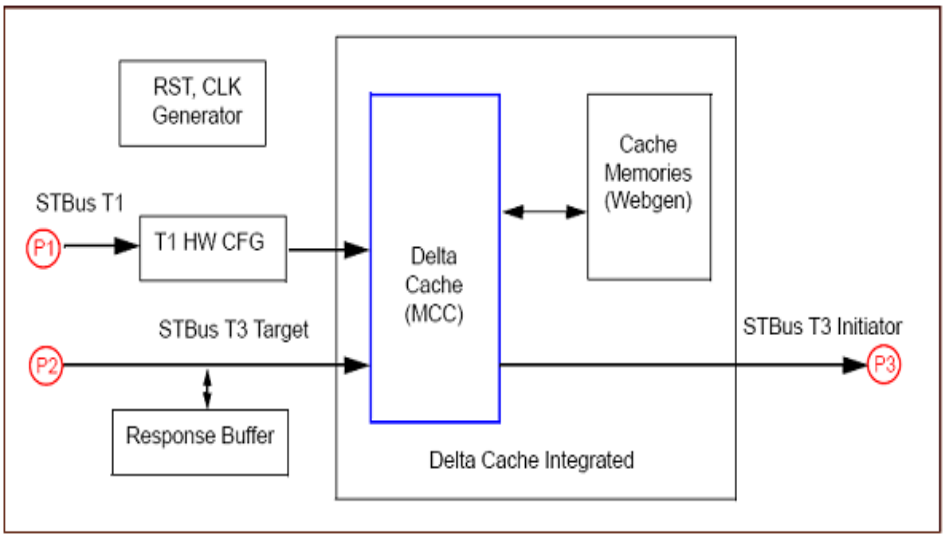


Figure 5.7: Verification Process

### 5.8.1 RTL Simulation Environment

IUS tool is used for simulating design. A script 'build_cmd' is used to precompile rtl and specman environment. For running a test, 'run_cmd' script is used which loads the precompiled snapshot, recompiles it along with the test and runs a simulation.Memory contents are dumped at the end of test for debugging in case of test failure.

### 5.8.2 Test Scenarios

Test are written in specman with constrained random generation of address and opcode on DDR initiator port. The timing delays for target and initiator ports are also randomized. To run allegro streams, 'stream.e' test reads the opcodes, address and end of chunk information from a file and drives them on the initiator port.

### 5.8.3 Checking Mechanism

All tests are self checking, dut_error is reported in case of any check failures. STBus protocol checks are enabled in all tests to ensure proper STBus compliant traffic at MCC input and output ports.

# Chapter 6

# Conclusion

We can say that, by using the MCC cache memory and other memories, which are described above, are very useful in TV. We can improve the quality of TV picture by digitizing the analog signal and using these kinds of memory we can reduce the delay of the broadcasted picture. By using this kind of memories, we can store the data in advance and so the delay is reduced. Also using MCC cache memory we can control the traffic of video data signals.

STBus is also a very useful protocol which is used for data transfer in between two modules. By using STBus we can apply input as well as output as per our requirement. And also we can check the modules by applying the different kinds of tests to our modules. So STBus is also a very useful protocol. And it is specially STMicroelectronics product which is used in only STMicroelectronics.

# References


[1] Davide Di Blasi, Run Tests User manual, 2001

[2] STBus Communication System : Concepts and definitions by Alberto Scandurra , Giuseppe Falconeri, Bruno Jego CMG division -OCCS team,Catania ,Granoble,.2002.

[3] Spyglass design checks by Jean-Christophe Mas ,Technology R&D, CCDS February 2008.

[4] EMIPCISS Generic IP block functional specification by Sanjeev varshney ,HED,Greater Noida,2010

[5] RTLGUI 5.X HEG R&D, Design Support, greater noida, India.

[6] Websites used
www.toodoc.com
www.ieee.com