ARC® 700 IP Library

ARC® 700 Memory Components

Reference

5116-017

ARC® 700 Memory Components Reference

ARC® International

European Headquarters
ARC International,
Verulam Point,
Station Way,
St Albans, Herts, AL1 5HE, UK
Tel. +44 (0) 1727 891400
Fax. +44 (0) 1727 891401

North American Headquarters
3590 N. First Street, Suite 200
San Jose, CA 95134 USA
Tel. +1 408.437.3400
Fax +1 408.437.3401

www.arc.com

ARC Confidential Information

5116-017 April-2008

Contents

List of Figures

Chapter 1 — Introduction

This document is aimed at programmers of the ARC® 700 Memory Components.

The following memory component information is provided:

- Block Diagram
- Build Configuration Registers
- Closely Coupled Memories Architectures
- Instruction Closely Coupled Memory (ICCM)
- Data Closely Couple Memory (DCCM)
- Instruction Fetch Queue

Block Diagram

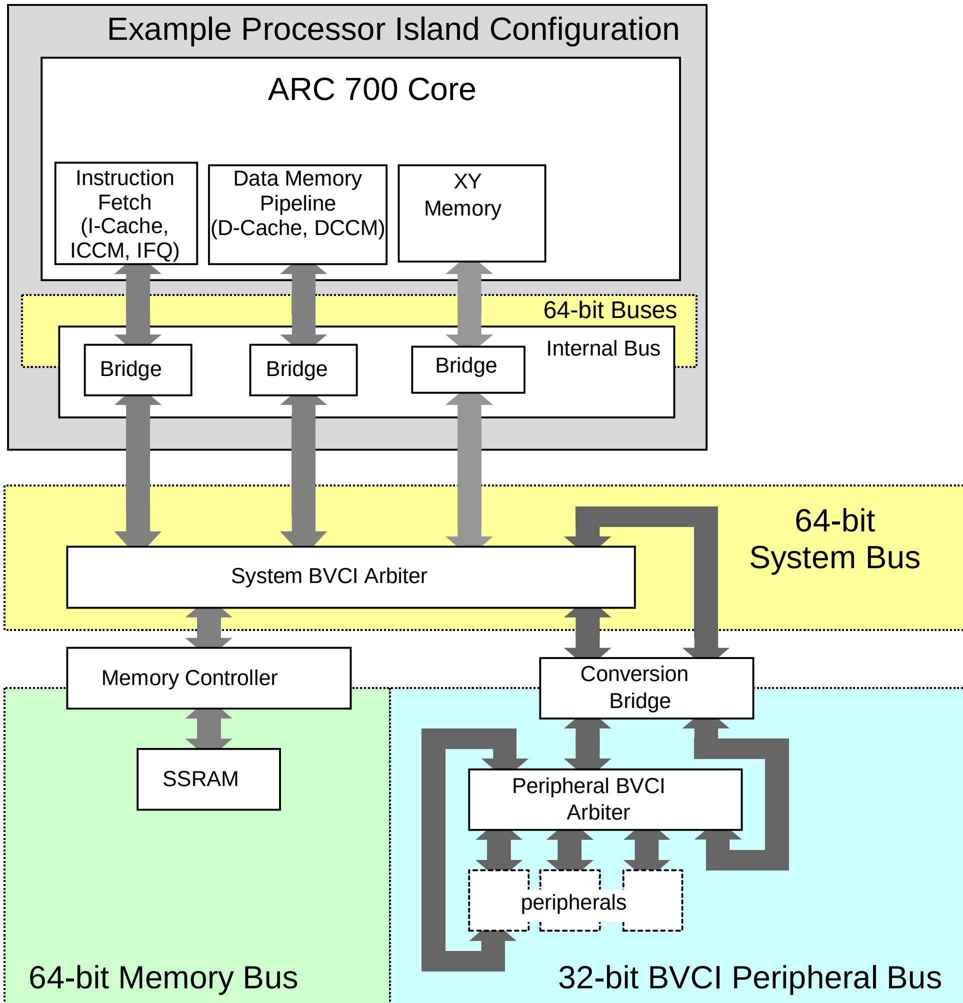

Figure 1 Example ARC 700 System Architecture

Chapter 2 — Register Set Details

The ARC 700 Memory Components use the following type of registers:

- Build Configuration Registers

Build Configuration Registers

There are various Build Configuration Registers (BCRs) which describe what type of CCM build that has been selected:

- DCCM Base Address, DCCM_BASE_BUILD, 0x61
- Memory Subsystem Configuration Register, MEMSUBSYS, 0x67
- DCCM RAM Configuration Register, DCCM_BUILD, 0x74
- ICCM Configuration Register, ICCM_BUILD, 0x78

NOTE *The Data Closely Coupled Memory (DCCM) utilizes two BCR's. 0x74 (DCCM_BUILD) should be read initially in order to determine the presence and size of the DCCM, and the DCCM_BASE_BUILD register (0x61) should be read to determine the physical base address of the DCCM in memory.

The registers as described in the following sections are arranged in numerical order.

DCCM Base Address, DCCM_BASE_BUILD, 0x61

31 … 8	7 … 0
ADDR[31:8]	VERSION

The field descriptions are shown in the following table.

Field	Description
VERSION	Current Version current version of the data closely coupled memory (DCCM) base address register
ADDR	Base Address This configurable field specifies the DCCM Base Address. Depending on the DCCM size the following address bit ranges are used: · 31 down to 13 – 8K · 31 down to 14 – 16K · 31 down to 15 – 32K · 31 down to 16 – 64K · 31 down to 17 – 128K · 31 down to 18 – 256K

Field	Description
	The default base address is set to 0x100.000.

Memory Subsystem Configuration Register, MEMSUBSYS, 0x67

31 – 3	2	1	0
Reserved	BE	R	EM

The field descriptions are shown in the following table.

Field	Description
EM	External Memory System Enabled - If there is an off chip RAM interface (i.e. memory arbitrator, sequencer) then this bit is set to 1.
BE	Big Endian System Enabled - If the ARC processor based system supports a big endian configured system then this bit is set to 1.

DCCM RAM Configuration Register, DCCM_BUILD, 0x74

31 – 11	10 – 8	7 – 0
Reserved	SIZE	VERSION

A zero is returned when the DCCM RAM is not present.

The field descriptions are shown in the following table.

Field	Description
VERSION	Current version
SIZE	Size of DCCM RAM
	0x0 = 2k
	0x1 = 4k
	0x2 = 8k
	0x3 = 16k
	0x4 = 32k
	0x5 = 64k
	0x6 = 128k
	0x7 = 256k

ICCM Configuration Register, ICCM_BUILD, 0x78

This register describes both the size and version number of the Instruction Closely Coupled Memory.

31 – x	– 11	10 – 8	7 – 0
BASE	Reserved	SIZE	VERSION

A zero is returned when the ICCM is not present.

The field descriptions are shown in the following table.

Field	Description
VERSION	Current version 0x1
SIZE	ICCM RAM Size
	This field refers to the size of the ICCM RAM:

	0x0 – No ICCM 0x1 – 8K Bytes 0x2 – 16K Bytes 0x3 – 32K Bytes 0x4 – 64K Bytes 0x5 – 128K Bytes 0x6 – 256K Bytes 0x7 – 512K Bytes
BASE	Base Address This configurable field specifies the ICCM Base Address. This address must be on a boundary defined by the size of the Instruction CCM RAM. · ICCM size 8K – x=13 · ICCM size 16K – x=14 · ICCM size 32K – x=15 · ICCM size 64K – x=16 · ICCM size 128K – x=17 · ICCM size 256K – x=18 · ICCM size 512K – x=19 The default base address is set to 0x0.

Instruction Fetch Queue Configuration Register, IFETCHQUEUE_BUILD, 0xFE

The Instruction Fetch Queue build configuration register, IFETCHQUEUE_BUILD, indicates that the Instruction Fetch Queue is present in a design.

31–8	7–0
Reserved	VERSION

The field descriptions are shown in the following table.

Field	**Description**
VERSION	Version 0x01 = Current version of Instruction Fetch Queue.

Chapter 3 — Closely Coupled Memories (CCM)

A Closely Coupled Memory architecture (CCM) has two separate memory buses: one for code and another for data.

The ARC 700 processor can be configured to a CCM architecture by selecting Closely Coupled RAMs rather than caches. CCMs allow for faster program execution as both instruction code and data is held locally in the memories.

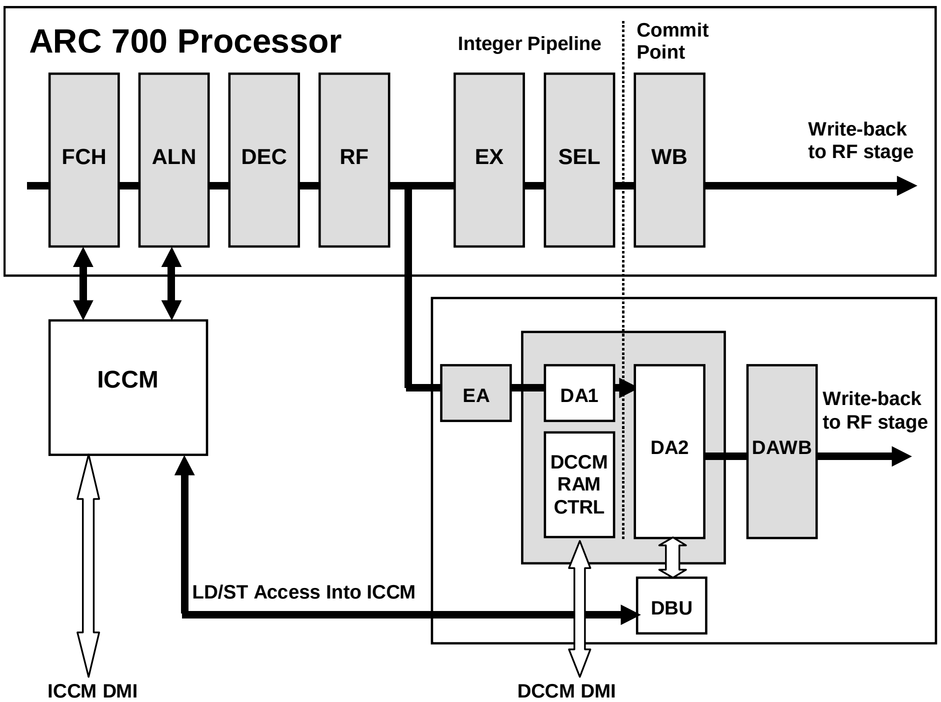

Figure 2 A Typical ICCM and DCCM Build

The following architecture options are available:

- ICCM Build
- DCCM Build

ICCM Build

The Instruction Closely Coupled Memory (ICCM) is similar to the Data Closely Coupled Memory (DCCM) in that it is a passive memory. The ICCM is mapped into physical memory space and its base address is configurable (refer to ICCM Configuration Register, `ICCM_BUILD`, for more details)

The size of the ICCM is configurable during the ARChitect build phase, and the RAM sizes supported are 8, 16, 32, 64, 128, 256 and 512 Kbytes.

All code accesses that fall within the range of the ICCM (0x0000 0000 to last word aligned address in the selected RAM size) result in an ICCM ‘hit’. All accesses to the ICCM are single cycle.

Accesses that exceed the physical boundary of the mapped RAM resource generate a memory exception.

In order to determine the ICCM RAM size, the ICCM build configuration register must be interrogated (auxiliary register 0x078).

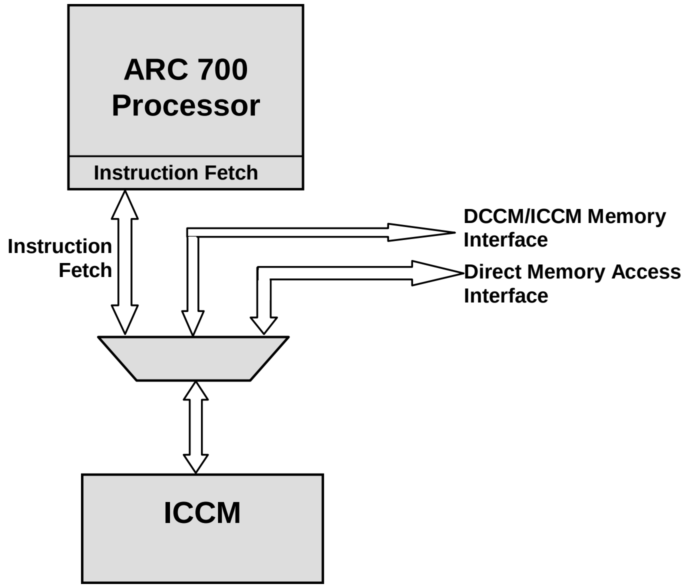

Figure 3 Simplified Processor/ICCM Diagram

Instruction Fetch in I-Cache and ICCM Mixed Builds

As a power saving feature in the ARC 700 processor, only one instruction memory is access for each instruction fetch in a mixed I-Cache and ICCM build. An instruction memory that has been fetched will keep on being fetched until a change is detected. A two cycle overhead is incurred to change from fetching one instruction memory to another. To remove uncertainty in handling interrupts, instruction fetch after an interrupt is assumed to start from ICCM. For performance critical interrupt handling routine (ISR), it is recommended to relocate the interrupt and exception vectors and the ISR to ICCM.

DCCM Build

The Data Closely Coupled Memory (DCCM) is a fast on-chip RAM that can either complement or replace the standard data cache architecture.

It has a direct memory interface that allows any device that is connected to it to perform burst operations to its memory (both read and write).

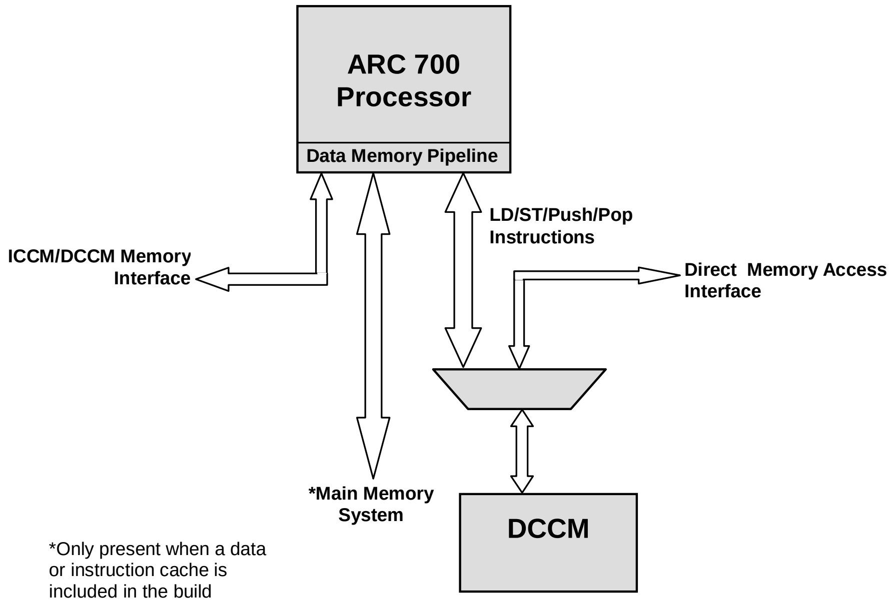

Figure 4 Simplified Processor/DCCM Diagram

The DCCM is mapped into the physical memory space and its base address is configurable (refer to DCCM Base Address, `DCCM_BASE_BUILD`, for more details). The RAM size of the DCCM can be configured to be 8, 16, 32, 64, 128, 256 or 512k bytes in size (configuration occurs during the ARChitect build phase).

In order to determine the DCCM RAM size, the DCCM build configuration register must be interrogated (auxiliary register 0x074).

All memory operations that fall within the boundaries of the DCCM result in a memory ‘hit’ within the RAM. The timing behavior of a load or store access into the DCCM is identical to that of a data cache ‘hit’.

DCCM/ICCM Memory Accesses

There are several destinations for a load or store for builds that contain both an ICCM and DCCM.

- ICCM Access
  A load or store memory access that is targeted to hit the memory map between 0x0000 0000 and the maximum address of the ICCM RAM, results in an access to the ICCM. This is particularly useful for breakpoint insertion and self-modifiable code.
- DCCM Access
  A load or store memory request that lies between 0x100000 and the maximum address of the DCCM RAM results in an access to the DCCM.
- ICCM Illegal Memory Access
  An instruction fetch that occurs outside the physical boundaries of the ICCM results in a memory exception error.
- DCCM Illegal Memory Access
  A load or store memory access that falls beyond the ICCM memory boundary or DCCM memory boundary results in a memory exception error.

- Overlapping CCMs with Off-Core Memory
  It is possible to map the ICCM and/or the DCCM to a memory region that is also populated by off-core memory. However, ARC strongly discourage this practice as it can lead to confusing behavior. For example, if an instruction code fragment is written to the DCCM and the software subsequently jumps to it, the instruction fetch path will fetch from the off-core memory instead of the DCCM. This is because the instruction fetch path does not cover the DCCM. If there is no off-core memory populated in the same region as the DCCM, this triggers a memory error that signals the instruction fetch is invalid. If this did not occur, the instruction word in the off-core memory overlapping the DCCM would be fetched and issued to the processor pipeline and this might not be what the programmer had expected.
- Cache and CCM mixed builds
  Builds that contain a mixture of caches and CCM change the behavior of the CPU when accesses fall outside the physical boundaries of the RAM. In the event that there is a build with an instruction cache, ICCM, data cache and DCCM, rather than generate a memory exception error for accesses that fall outside the physical boundaries of the CCM, they are simply passed onto either the instruction or data cache (depending on the access type, e.g. instruction fetch or load\store access).
  In the event that there is a cache on one interface and CCM on the other, the access that falls outside the physical boundary of the CCM is presented to the main memory system. For example, if the build contained an instruction cache and DCCM, but no ICCM or data cache, then any load\store accesses that fall outside the physical boundaries of the DCCM are presented to main memory.

Big-Endian Configuration

When the ARC 700 system is configured as a big-endian system the DCCM will operate to provide the correct data, in big-endian format, to the processor as documented in the *ARCompact Programmer's Reference*.

Chapter 4 — Instruction Fetch Queue

The Instruction Fetch Queue allows ARC 700 systems, without an Instruction Cache, to fetch instructions, through the Island Bridge (BVCI, AHB, AXI or *ARC legacy*), from external memory. The Instruction Fetch Queue (IFQ) is often used for boot code or infrequently used code that is not present in the ICCM.

The processor instruction fetch port is connected, via the Instruction Fetch Queue, to the island bridge. The IFQ uses a linear pre-fetching scheme to read instructions from memory ahead of instruction fetches from the ARC 700 pipeline. The Instruction Fetch Queue can be use with or without an ICCM. See Figure 1.

The IFQ configuration register, IFETCHQUEUE_BUILD, indicates that the Instruction Fetch Queue is present in a design.

The IFQ incorporates a buffer containing four 64-bit wide entries. Entries in the buffer are always a linear sequence of pre-fetches from memory. The IFQ logic maintains the buffer by making individual read requests from memory in 64-bit quantities whenever the buffer is not full. For this reason, the IFQ is set to be the lowest priority component for the arbiter.

When a new instruction fetch request is received from the ARC 700 pipeline, the address of top entry in the IFQ buffer is checked against the requested address.

- If the top address matches, the pre-fetched instruction is provided from the IFQ
- If the top address does not match, the entire contents of the IFQ are discarded (including any pending requests from memory). A new pre-fetch sequence is started at the fetch address provided by the fetch stage.

Note that no caching is performed - at least one memory request is made for each instruction issued. Any non-linear changes in the fetch address (a branch for example), will cause the IFQ to be emptied and re-filled, even if the target address is held somewhere in the IFQ buffer.

Self-modifying code running on the ARC 700 processor should ensure that the IFQ has been emptied and refilled by executing a SYNC instruction after code in memory is changed. This ensures that any outstanding memory writes have been completed, and causes a pipeline flush which in turn causes the IFQ to be emptied.

Breakpoints written through the debug interface will be successfully recognized since all debug transactions cause pipeline flushes on completion, causing the IFQ to be refilled.