Application Note, V 1.1, June 2004

AP32066

TriCore® 1 Using Synchronization Primitives

TC1M

Microcontrollers

Never stop thinking

Revision History:	**2004-06**	V 1.1
Previous Version:	-	

Page	Subjects (major changes since last revision)
	Minor Changes

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

ipdoc@infineon.com

1 An Introduction To Synchronization Primitives

This application note introduces the TriCore® synchronization primitives (DSYNC and ISYNC instructions) and provides guidelines for using them.

The DYSNC and ISYNC primitives provide a mechanism through which software can guarantee the ordering of various events within the machine. In other words, they provide synchronization of instruction (ISYNC) and data (DSYNC) streams on the multistage pipelines.

The DSYNC primitive provides a mechanism through which a data memory barrier can be implemented. The DSYNC instruction guarantees that all data accesses associated with instructions semantically prior to the DSYNC instruction, are completed before any data memory accesses associated with an instruction semantically after DSYNC are initiated. This includes all accesses to the system bus and local data memory.

The ISYNC primitive, provides a mechanism through which the following can be guaranteed:

– If an instruction semantically prior to ISYNC makes a software visible change to a piece of architecture state, then the effects of this change are seen by all instructions semantically after ISYNC. For example, if an instruction changes a code range in the protection table, the use of an ISYNC instruction will guarantee that all instructions after the ISYNC are fetched and matched against the new protection table entry. All cached states in the pipeline, such as loop cache buffers, are invalidated.

The operation of the ISYNC instruction is described as follows:

1. Wait until all instructions semantically prior to the ISYNC have completed.
2. Flush the CPU pipeline and cancel all instructions semantically after the ISYNC.
3. Invalidate all cache state in the pipeline.
4. Re-fetch the next instruction after the ISYNC.

DSYNC and ISYNC allow programmers to control the order of events inside TriCore with various real-time applications. In some circumstances, as the examples in this document will demonstrate, it is still recommended to use these primitives despite the fact that they may cause pipeline side effects (*see* **Section 2.2 Silicon Bug Workaround**).

2 DSYNC Primitive

DSYNC (Synchronize Data) forces all data access to complete before any data accesses associated with an instruction semantically after the DSYNC are initiated.

2.1 Cache Coherency

One of the applications of DSYNC is for cache coherency. TC1M provides software support for cache coherency by invalidating I-cache, as well as invalidating and/or flushing D-cache with the following instructions:

1. Cache Flush Instruction: CACHEA.W
2. Cache Invalid Instruction: CACHEA.I
3. Cache Flush & Invalidate instruction: CACHEA.WI

It is recommended that a DSYNC instruction be placed after any of these instructions to ensure that all data access is completed and the pipeline synchronized, before cache coherency takes effect.

2.2 Silicon Bug Workaround

Aside from cache coherency applications, DSYNC can also be used to workaround some silicon bugs found in TC1M, such as solving unexpected context related traps for example:

1. CPU13 Bug: A context restore from CSA (Context Save Area) stack may trigger a FCD (Free Context List Depleted) trap unexpectedly, even if the CSA management completed properly.
2. CPU14 Bug: A FCU (Free Context List Underflow) trap will be triggered when CSA stack depth exceeds 8; i.e. when 8 nested subroutine calls are taking place.

The workaround for these two silicon bugs is to place a DSYNC instruction at the very top of all the subroutines, including the Interrupt Service Routine (ISR), in TriCore applications.

However, it must be noted that the DSYNC instruction may cause corrupted execution of the instructions which follow it. Up to two instructions after a DSYNC may see the wrong source value when A12-15 or D8-15 are used as a source operand for example.

To solve the DSYNC corruption problem, one NOP (No Operation) instruction has to be inserted after the DSYNC instruction. The pseudo code for the cache coherency application for example, would therefore be as follows:

```
CACHE.I or CACHE.W or CACHE.WI
DSYNC
#If (the next two instructions are handling A12-15 or D8-15)
NOP
#endif
```

The pseudo code for the context related silicon bug workaround would be:

Subroutine_Name_Label:

```
DSYNC
#If (the next two instructions are associated with A12-15 or D8-15)
NOP
#endif
{ Body of Subroutine }
RET or RFE
```

3 ISYNC Primitive

ISYNC (Synchronize Instructions) forces completion of all previous instructions, flushes the CPU pipelines and invalidates any cached pipeline state before proceeding to the next instruction.

3.1 Core Register Initialization

The major use of ISYNC is for core register initialization's. For this document the core registers that ISYNC relates to have been divided into six simplified categories - *see* **Table 1 Core Registers That Require An ISYNC After Being Initialized (Simplified)**.

TC1M implements an MTCR (Move To Control Register) instruction to initialize core registers by moving the contents (initialization value) in a data register to the core register. A MTCR instruction should be performed together with an ISYNC instruction (which synchronizes the instruction stream) in order to ensure completion of that operation before the execution of the following instructions.

Usually programmers have to handle / initialize the core registers of the first three categories in their code, while debug tools (like emulator) and the OS or RTOS will take care of the rest of the core registers.

The pseudo code to initialize a core register is:

```
{Instructions for loading initialization value to data register D0}
MTCR#CORE_REGISTER, D0
ISYNC
```

Table 1 Core Registers That Require An ISYNC After Being Initialized (Simplified)

Type of Register	Abbreviation	Explanation
CPU (Central Processing Unit) related	PSW	Program Status Word
	SYSCON	System Configuration Register
Context related	PCXI	Previous Context Information
	FCX	Free Context List Head Pointer
	LCX	Free Context List Limit Pointer
Interrupt & Trap related	ISP	Interrupt Stack
	ICR	Interrupt Control Register
	BIV	Base Address of Interrupt Vector Table
	BTV	Base Address of Trap Vector Table
Memory Protection related	DPRx_x	Data Segment Protection Register Sets
	DPMx_x	Data Protection Mode Register Sets
	CPRx_x	Code Segment Protection Register Sets
	CPMx_x	Code Protection Mode Register Sets
MMU (Memory Management Unit) related	MMUCON	MMU Configuration Register
	ASI	MMU Address Space Identifier
	TVA	MMU Translation Virtual Address
	TPA	MMU Translation Physical Address
	TPX	MMU Translation Page Index
	TFA	MMU Translation Fault Address
OCDS (On Chip Debug Support) related	DBGSR	Debug Status Register
	EXEVT	External Break Event Specifier
	SWEVT	Software Break Event Specifier
	CREVT	Core SFR Event Specifier
	TR0EVT	Trigger Event 0 Specifier
	TR1EVT	Trigger Event 1 Specifier

3.2 Interrupt Control

TC1M implements three instructions to simplify the control of the Interrupt Control Register (ICR):

1. Interrupt Enable Instruction: ENABLE
   Sets the bit ICR.IE of ICR to enable interrupt

2. Interrupt Disable Instruction: DISABLE
   Clears the bit ICR.IE of ICR to disable interrupt

3. Interrupt Service Routine Begin Instruction: BISR
   Sets the bits ICR.CCPN value of ICR and prepares an Interrupt Service Routine (ISR) including save lower context and enable interrupts.

It should be noted that these three instructions are executed such that the CPU is blocked from taking interrupt requests until the instruction is completely finished, therefore eliminating the need for an ISYNC instruction and so avoiding pipeline side effects.

3.3 Enabling I-Cache & D-Cache

Another recommended use of ISYNC is when enabling I-cache and D-cache. The purpose is to flush and invalidate the CPU pipeline states and therefore ensure the pipeline is fully synchronized, before proceeding to the next instructions. The pseudo code for this is illustrated as follows:

```
{Instructions for initializing PMUCON0 to enable/configure I-Cache}
ISYNC
{Instructions for initializing DMUCON to enable/configure D-Cache}
ISYNC
```

References

The following **Infineon** documents should be referred to for additional reference information:

- TriCore 1 Architecture Manual

For TriCore information on the Internet, visit:

http://www.infineon.com/tricore

Glossary

Abbreviation	Definition
ASI	MMU Address Space Identifier
BISR	Begin Interrupt Service Routine
BIV	Base Address of Interrupt Vector Table
BTV	Base Address of Trap Vector Table
CACHEA.I	Cache Address Invalidate
CACHEA.W	Cache Address Writeback
CACHEA.WI	Cache Address Writeback & Invalidate
CPM	Code Protection Mode Register Sets
CPR	Code Segment Protection Register Sets
CPU	Central Processing Unit
CREVT	Core SFR Event Specifier
CSA	Context Save Area
DBGSR	Debug Status Register
DMU	Data Memory Unit
DPM	Data Protection Mode Register Sets
DPR	Data Segment Protection Register Sets
DSYNC	Synchronize Data
EXEVT	External Break Event Specifier
FCD	Free Context List Depleted
FCU	Free Context List Underflow
FCX	Free Context List Head Pointer
ICR	Interrupt Control Register
ICR	Interrupt Control Register
ISP	Interrupt Stack
ISR	Interrupt Service Routine
ISR	Interrupt Service Routine
ISYNC	Synchronize Instructions
LCX	Free Context List Limit Pointer
MMU	Memory Management Unit
MMUCON	MMU Configuration Register

Abbreviation	Definition
MTCR	Move To Context Register
NOP	No Operation
OCDS	On-Chip Debug Support
OS	Operating System
PCXI	Previous Context Information
PMU	Program Memory Unit
PSW	Program Status Word
RET	Return From Call
RFE	Return From Exception
RTOS	Real-Time Operating System
SFR	Special Function Register
SWEVT	Software Break Event Specifier
SYSCON	System Configuration Register
TFA	MMU Translation Fault Address
TPA	MMU Translation Physical Address
TPX	MMU Translation Page Index
TR0EVT	Trigger Event 0 Specifier
TR1EVT	Trigger Event 1 Specifier
TVA	MMU Translation Virtual Address

http://www.infineon.com

Published by Infineon Technologies AG