Using the –edma_warnN Compiler Switch to Detect a CPU L2 EDMA Lockout

Ivan Garcia, Vineet Ganju *C6000 Hardware Applications*

ABSTRACT

This document discusses the use of the –edma_warnN compiler switch (where N is the deadline in CPU clock cycles that the EDMA must meet) for detecting potential scenarios on which EDMA accesses are blocked by the CPU from accessing L2.

Contents

List of Figures

1 Description and Use

The –edma_warnN compiler switch is used to find potential EDMA lockouts by the CPU. Details for this problem can be found in the Silicon Errata for C671x devices. This switch generates a warning only when the following two criteria are met:

1. **Criterion One.** For a given sequence of code: If the total number of stores (including single and parallel stores) is greater than or equal to the total number of cycles that contain no stores then the first criterion is met.
2. **Criterion Two.** For a given sequence of code: If the EDMA must service a peripheral in N CPU clock cycles to meet a hard deadline, but there is also a loop of at least N CPU clock cycles, then the second criterion is met. For example, if there is a hard deadline of 586 cycles that the EDMA must meet to service a peripheral, the –edma_warn586 switch is used.

An example of the warning issue when both criteria have been met:

"source_file.c", line 274: WARNING: An EDMA lockout problem may exist

NOTE: In some cases the compiler may not be able to determine the number of iterations a loop will run. Therefore it won't be able to guarantee the duration of the loop will be less than N cycles. In this case the compiler will be conservative and issue a warning. The warning contains the source code location that produced the potential problem loop as noted on the example.

A detailed description of these two criteria now follows.

1.1 Criterion One

To meet Criterion One, in a given sequence of code, the total number of stores must be greater than or equal to the number of cycles on which no store occurs. In the case of looped code, in one iteration of a loop, the total number of stores must be greater than or equal to the number of cycles on which no store occurs, or, in other words, the length of one iteration in cycles is less than or equal to twice the number of stores.

Here are a few examples of instruction sequences that outline this criterion:

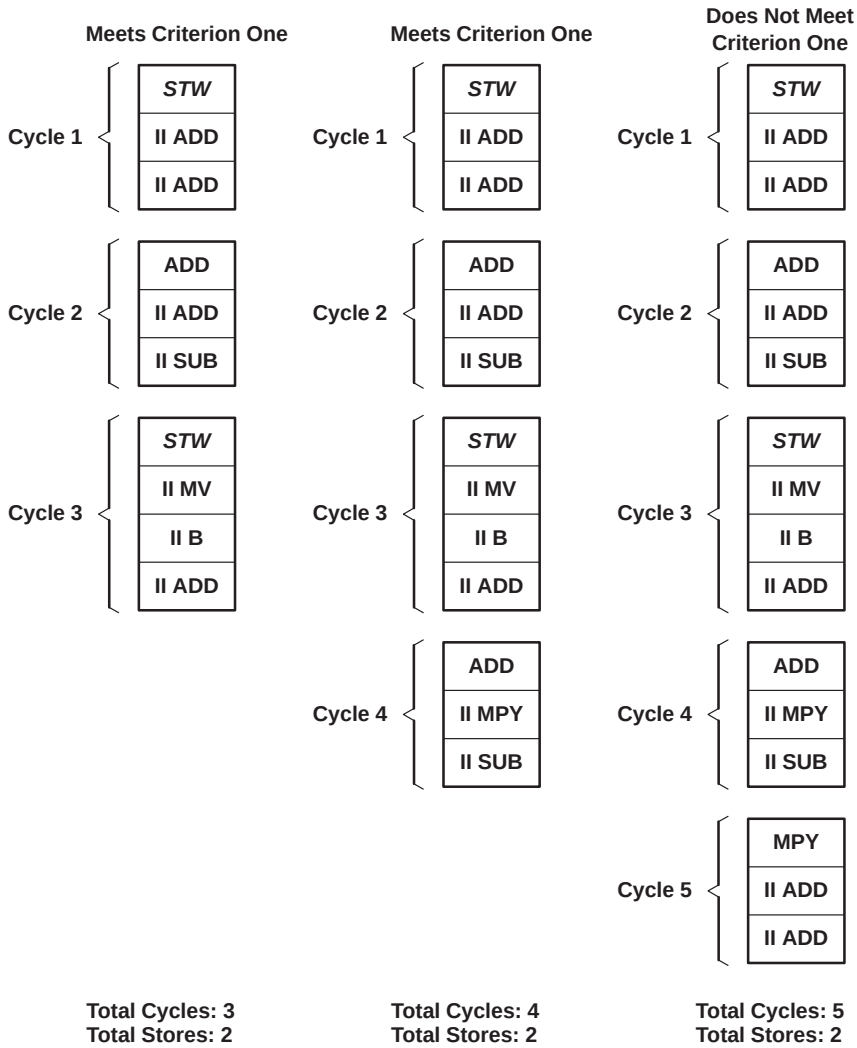

Figure 1. Pseudo Code Example With Parallel Instructions (Criterion 2)

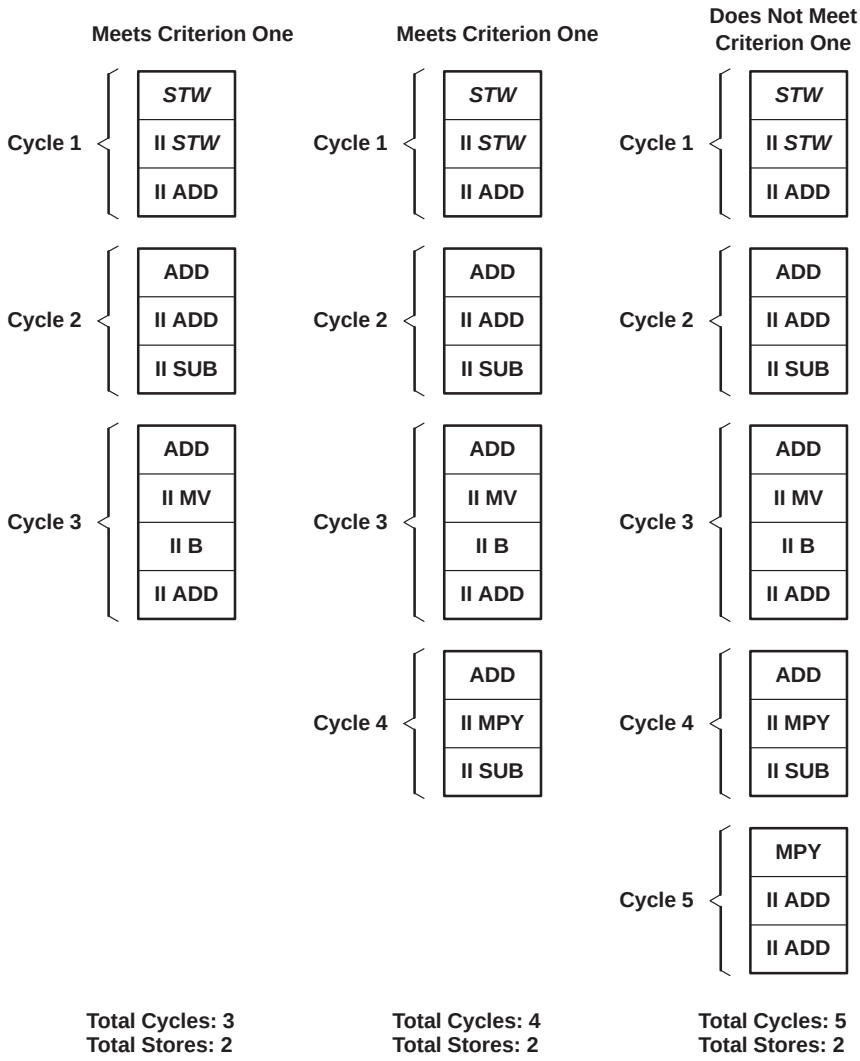

Figure 2. Pseudo Code Example Parallel Stores (Criterion 2)

Note the following to meet Criterion One:

- The order of the store instructions does not matter
- When two store instructions are performed in parallel (instructions performed in parallel are denoted by the || symbol), each store must be counted individually. That is, even though both stores occur on the same cycle, they still must be counted as two stores
- It does not matter if the store is storing a 32-bit word (STW), 16-bit word (STH), or 8-bit word (STB)

1.2 Criterion Two

The second criterion can be viewed pictorially using a timeline.

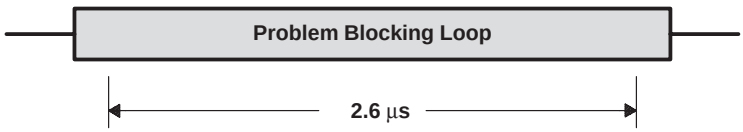

Figure 3. Criterion Two

There is a hard deadline, say 2.6us, during which time the EDMA must access L2. If the DSP system clock is running at 225MHz, having a hard deadline of 2.6us translates to 586 CPU cycles. The criteria is met when the CPU has a loop of at least 586 (N = 586) clock cycles, the length of the EDMA hard deadline. The switch used for this particular hard deadline would then be –edma_warn586.

NOTE: In some cases the compiler may not be able to determine the number of iterations a loop will run. Therefore it won't be able to guarantee the duration of the loop will be less than N cycles. In this case the compiler will be conservative and issue a warning. The warning contains the source code location that produced the potential problem loop.

2 Analyzing Potential Problems

Once a list of potential problem loops is generated by the –edma_warnN compiler flag, the programmer must now examine each of these potential problem loops to eliminate the compiler's warnings due to not being able to guarantee the number of iterations. Furthermore, the programmer should also examine each potential problem loop to see if they meet the two additional criteria for blocking the EDMA from accessing L2. This is accomplished by closely examining the source code and assembly output of that source code. The assembly output of source files are retained by using the compiler option –k.

The following checklist should be used to qualify a loop as a problem loop. If the potential problem loop fails any of these checks, it is *not* a problem loop and the EDMA will not be blocked from accessing L2.

1. Examine the source code to determine the number of iterations of a loop. The compiler switch does its best to determine the number of iterations of a loop, but it cannot always determine this. If a programmer knows that a potential problem loop will only run for a few iterations, then it can be eliminated as a potential problem. However, if the loop runs a number of times close to or greater than the hard deadline, then it can still be a potential problem.
2. Check to see if the loop contains stores to the same bank in L2. This can be done by checking the source code to see if the index increment is a factor of 8, or some factor that would cause all the stores to be in the same bank.

For example, this loop would likely cause a problem, because the stores are striding by 8 elements at a time (hitting the same bank of L2):

```
int buffer[];
for (i=0; i< 1000; i++)
      buffer[i*8] = 0;
```

But this loop would not cause a problem since it is only striding by one element at a time:

```
int buffer[];
for (i=0; i<1000; i++)
      buffer[i] = 0;
```

3. Check to see if the loop has any misses in L1. This is done by examining the assembled output of the source. Find the particular loop of interest in the assembly file associated with the source file containing the potential problem loop. Examine the kernel of that loop to see if there are any loads performed in the kernel. Any load which misses L1 immediately disqualifies the loop from being a problem loop.

All these criteria would typically occur in a loop that is filling or clearing a buffer. Even then, it is only filling/clearing a certain section of a buffer (every eighth element). For this reason, a problem loop is rarely found in a typical post-processing routine.

After performing these steps, the list of potential problem loops should be reduced when using the –edma_warnN compiler switch.

For more details on this problem please refer to the corresponding C621x/C671x Silicon Errata. Please contact your local customer support to obtain the tools that support the –edma_warnN compiler switch.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products & application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265