DSPLab
User's Manual

First Edition
Kyanoosh Shafaei

DSPgig.com

Registered at Library of Congress.

First printing edition 2019.

kshafaei@dspgig.com

www.dspgig.com

IMPORTANT NOTICE

DSPgig, Inc. reserves the right to make changes to its products or to discontinue any product or service without notice. Customers are advised to obtain the latest version of relevant information to verify that the data being relied on is current before placing orders. DSPgig, Inc. warrants performance of its products and related software to current specifications in accordance with DSPgig's standard warranty. Testing and other quality control techniques are utilized to the extent deemed necessary to support this warranty. Please be aware that the products described herein are not intended for use in life-support appliances, devices, or systems. DSPgig does not warrant nor is DSPgig liable for the product described herein to be used in other than a development environment. DSPgig, Inc. assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does DSPgig warrant or represent any license, either express or implied, is granted under any patent right, copyright, or other intellectual property right of DSPgig, Inc. covering or relating to any combination, machine, or process in which such Digital Signal Processing development products or services might be or are used.

WARNING

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures necessary to correct this interference.

Contents

Preface

1) History

The first Digital Signal Processor (DSP) was introduced by Texas Instrument (TI) in 1978. Next, in 1983, TI presented TMS32010, which was a bigger success. (From then on, all TI DSP names started with TMS320, for example TMS320C30 or TMS320C50.) DSPs have been used in signal processing applications and especially in military equipment. DSPs such as TMS320C25[1] continue to be used widely in many military applications and are still in production.

The early generations of DSP do not have a JTAG connection, which is used for debugging. Later, TI introduced new families of DSPs with JTAG interfaces and increased the product number from two digits to four digits, such as TMS320C5509. When a processor has a bug, the next generation of the same processor has an extra letter at the end, for example TMS320C5509A.

Over the years, TI worked hard to create the most advanced DSPs and introduced a lot of innovations and improvements. Although other manufacturers also made good DSPs, with TI DSPs you never go wrong.

2) "DSPLab User's Manual"

This book is about learning TI-DSPs. There are six Labs in the book, and each Lab has multiple examples and exercises based on DSPLab hardware. This book focuses on Code Composer Studio (CCS) and some available software packages.

Figure 1: DSPLab hardware

[1] - TI made TMS320C25 at 1987.

2.1) Prerequisite

The Code Composer Studio has a powerful C/C++ compiler, so it is essential for students to be familiar with C programming. Knowledge of C++ is not necessary, but would be helpful. Before reading the book, make sure you are familiar with the following C syntaxes:

1. Global and local variables.
2. ‘int’, ‘char’, ‘long’, ‘float’, structures, and unions.
3. 'for', 'while', 'if', 'else', 'extern'.
4. '#include', '#if', '#ifdef', '#else', '#endif', '#pragma'.

2.2) Before the first laboratory session

The DSPLab hardware has a USB connection that can be connected to any computer, so you may choose to use your own laptop. Students should download and install the latest version of Code Composer Studio (CCS) from the TI website (www.ti.com). The software is free, but currently TI requires a registered account. Be aware that creating a registered account may take one or two days.

2.3) Group activity

It is recommended that all examples and exercises are done individually so that students better learn the basics. For this reason, team working is discouraged, especially for the first few Labs. If there is not enough DSPLab hardware, one DSPLab hardware can be shared by connecting and disconnecting it from your computer.

3) Other documentation

Besides “DSPLab User’s Manual” (this book), there are 4 additional documents, each of which can be used independently:

1- “DSPLab Teacher’s Manual”: The teacher’s guide is very similar to this document with more explanations for some complex topics.
2- “DSPLab Hardware Manual”: The DPLab hardware manual covers both hardware and software design for the DSP. Each chapter covers one of the peripherals, and starts with hardware and ends with software design. This is a very useful book if you are planning to design your own hardware.
3- "Real-Time Digital Signal Processing, Implementation, and Application" covers signal processing principals and real-time applications for 55xx families. Based on your interest, you may choose to read and then implement a few chapters using DSPLab hardware.
4- “TI-DSP Reference Manual, 2000, 5000, and 6000” is scheduled to be published by the end of 2020 and can be used as a reference for each Lab topic.

4) Labs

There are 6 Labs in the book:

- Lab1 introduces the CCS software and basic features.
- Lab2 shows how easy to use DSPLab hardware is for signal processing applications.
- Lab3 goes more in depth about DSP internal structure and explains how C compiler manages memories.
- Lab4 covers advanced topics about signal processing libraries.
- Lab5 uses FFT as an example of how to practice more inside CCS.
- Lab6 is about Image Processing library.

Lab 1

CCS introduction

DSP Gig Inc.
Kyanoosh Shafaei

1) Prerequisite

Knowledge of 'C/C++' programing language is needed for programming most of the TI DSP[1]s. Before continuing, read the first 100 pages of any C language book containing essential instructions such as 'for,' 'while', 'switch-case', 'if', variable definition, pointer, array, and structure.

The software used for TI-DSP programming is Code Composer Studio (CCS). First, install CCS software. The latest version of the software can be downloaded from the TI website at the following address:

http://processors.wiki.ti.com/index.php/Download_CCS [2]

2) Introduction

The objective of this lab is to learn how to use Code Composer Studio and its debugging features. This chapter is focused on the basics of CCS and how to load the code from the computer into a TI DSP memory.

3) How to connect DSPLab to the PC:

DSPLab is connected to the computer by a USB cable located on the back of the box. Note that there is also a USB port on the top-front. But embedded JTAG can only be connected to PC using the USB connection on the back of the box.

To start, follow these simple steps:

• Connect the DSPLab hardware, using the USB connection on the back to the computer[3].

---

[1] - Texas Instrument Digital Signal Processors

[2] - If you choose to download an older version of CCS(before version 8), when opening the software for the first time, you are asked to enter a license file. Select XDS100 free option. Also TI recently released a free license file for older CCS version which can be downloaded from the TI website.

[3] - The DSPLab has two USB connectors: One on the top of the box and one on the back. The one on the back is for JTAG interface and can be used to connect CCS to the DSP and load the code.

Lab 2

DSPLab introduction

DSP Gig Inc.
Kyanoosh Shafaei

1) Introduction

In this session, different parts of the DSPLab will be introduced. We will learn how to:

1- Work with the keys and LEDs.

2- Write to digital-to-analog converter (DAC), and generate a waveform on the output.

3- Read analog-to-digital converter (ADC), and using an integrated oscilloscope to display the signal.

4- Create an image and display it on LCD.

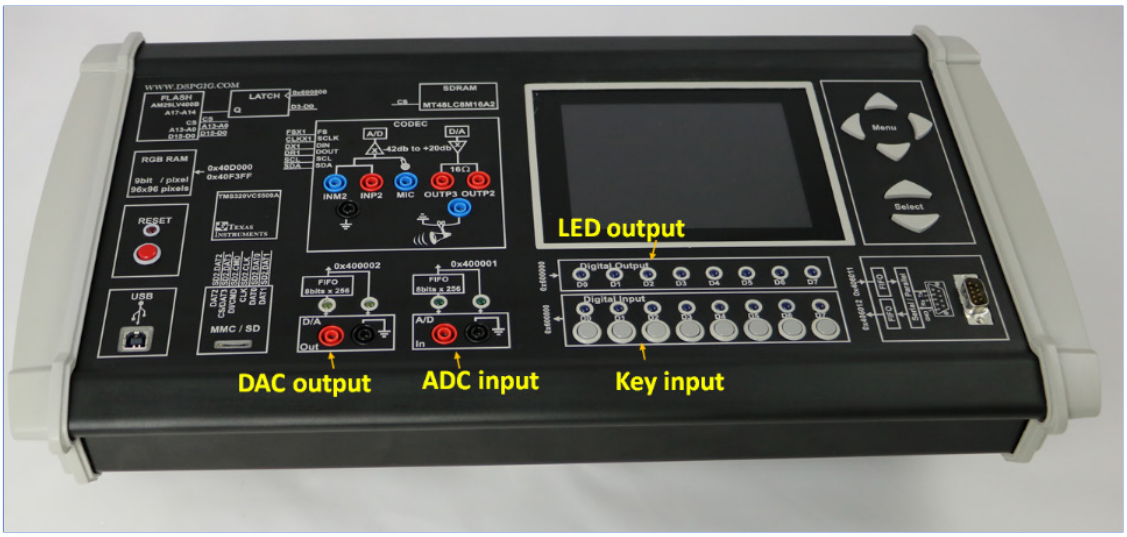


Figure 1: Location of LED, Key, ADC, and DAC in DSPLab

DSPLab has some integrated features which make DSP learning easy. To achieve this goal, all the hardware complexity is hidden from the final user, and a simple user interface is provided [1]. To work with each peripheral in DSPLab, usually two steps needs:

Step1: Apply the required settings from the LCD user interface.

Step2: Read or write to an address in memory to access the peripheral.

In DSPLab, each of the LED, key, ADC, DAC, and the graphical display has a unique address in the DSP memory. To access these peripherals, the code should read or write from the specific

[1] - Even though this user interface can be more advanced but the designers decided to keep the interface as simple as possible so the student's time be focused more on DSP learning.

Lab 3

Memory Management

DSP Gig Inc.
Kyanoosh Shafaei

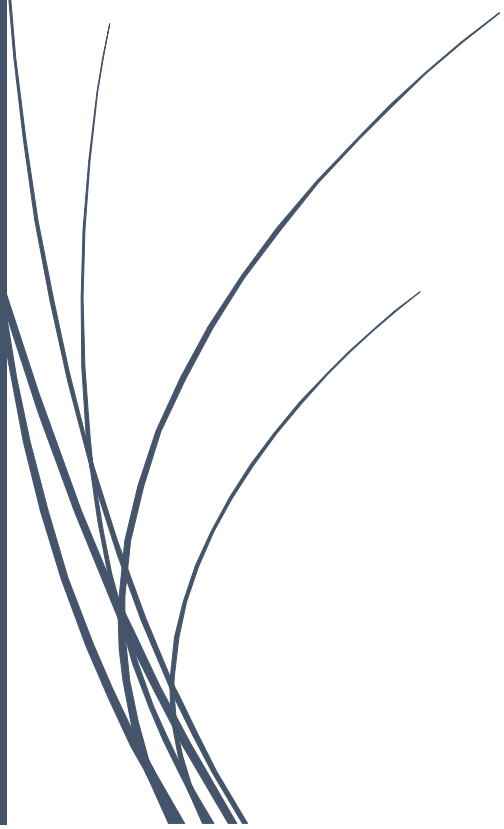
1) Introduction

The Code Composer Studio supports embedded C programming. The difference between embedded C and traditional C is extensive memory management. In C programming, sometimes a part of the code should be in a specific area in memory, or a big array (like multiple images) needs to be in external memory. In assembly, the memory is under user control, and user can access any part of the memory. Embedded C is very similar to the assembly language. The embedded C supported by CCS is one of the most advanced memory management used in C.

A command file is a powerful tool used by CCS to manage memory. The command file puts codes and variables in any desired place in memory. A simple usage is placing the code in one memory and the data in another memory, so the processor can access both code and data in a single cycle. The next picture shows the internal structure of TMS320VC5509A. As shown in the picture, the hardware has multiple internal buses for program and data. So the internal core can use one program bus to load the instructions from memory while it can use two other data buses to load the data needed by those instructions from memory. This can happen in one cycle if there is no conflict in the memory access. But, if both code and data are in the same internal memory (SARAM or DARAM), then there is a bus conflict, and one memory transaction has to wait for the other one to finish.

Lab 4

DSP Libraries



DSP Gig Inc.
Kyanoosh Shafaei

Chapter Goal

In many practical signal processing applications, there is a need to have optimized assembly functions. A simple code analysis shows which part of the code uses the most cycle count and needs more optimization. C compiler introduces lots of optimizations, but still, the human brain (assembly coding) is much better. It is important to note that never should write the code entirely in assembly. Usually, a small portion of the code consumes most of the processor cycles and is the right candidate for assembly coding.

TI has made a lot of optimized assembly packages for various signal processing applications. In this chapter, DSPLIB is chosen as a sample. The TI libraries are not installed as part of the CCS, so CCS, by default, cannot find them. Almost all TI packages need some necessary preparation steps before being used. These steps are not complicated, but in this chapter, the students practice how to prepare (build) a TI library after downloading from the website and then how to reuse the examples and functions.

Due to continue software change, the library has multiple issues when being used in a newer version of CCS. This example is a good candidate to show the majority of the library problems. Most of the other libraries are not as old as DSPLIBv2.4 and have fewer problems, but this is the best option for training purposes.

1) Introduction: use of optimized assembly functions for mathematical application

Signal processing is the main application for DSP processors. When DSP first showed up in the market, they become famous for having mathematical calculation capabilities. On the early versions of DSP, coding has to be done in assembly language. Later C language introduced by CCS but still assembly language was the only option for optimized code. Then, software such as MATLAB became a big player. MATLAB used widely for mathematical algorithms implementation, and it can produce C code for DSPs. However, the C code was not efficient at all. Then TI introduced some outstanding levels of optimization in the C compiler. That helps a lot with the inefficiency of the C language, but still, there was a noticeable gap between the assembly and C compiler. To this

Lab 5

FFT

DSP Gig Inc.
Kyanoosh Shafaei

1) How to implement a fixed-point algorithm?

The fixed-point implementation is always challenging. The fixed-point calculation is considered less accurate compare to floating-point. The next picture shows the difference between a 32-bit multiplier in a fixed-point and floating-point DSP.

In a fixed-point DSP, the output of a 32-bit multiplier is stored in a 64-bit accumulator, while usually, the output of a 32-bit floating-point multiplier goes to a 32-bit accumulator.

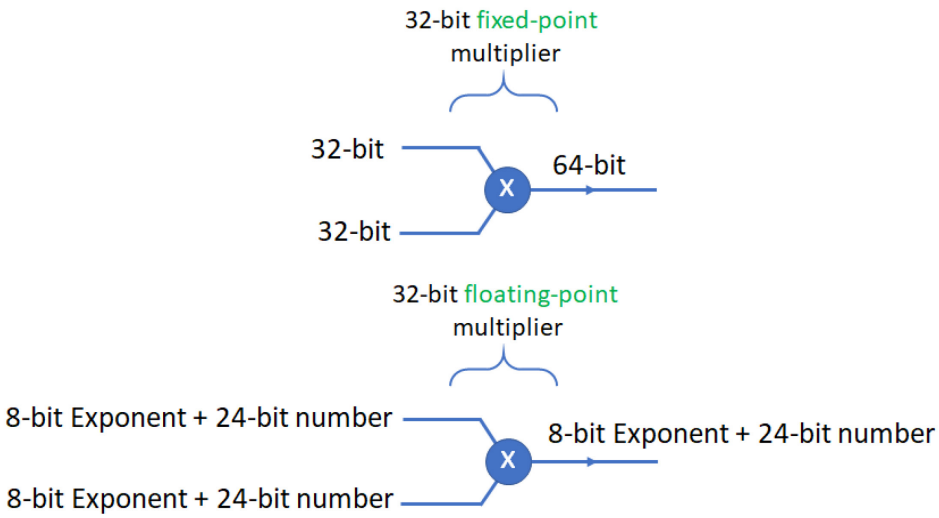


Figure 1: 32-bit fixed-point multiplier vs. floating-point multiplier

A 32-bit floating-point number is actually a 24-bit number (The extra 8-bit is the shift). The output of a 24-bit fixed-point multiplier is 48-bit, but in floating-point, when 24-bit numbers are multiplied together, the result is truncated to a 24-bit number. So in a floating-point multiplier, the 24-bit LSB is ignored after each multiplication.

Because no bit is lost in fixed-point multiplication, a 32-bit fixed-point multiplier can be more accurate than a 32-bit floating-point multiplier. On the other hand, in fixed-point, a 64-bit number has to be truncated to a 32-bit number before being used as an input to a new multiplier. As an example, if the number inside the 64-bit accumulator is only 48-bit, after truncating the 32-bit LSB, only 16-bit is remaining.

Lab 6

Image

DSP Gig Inc.
Kyanoosh Shafaei

1) Introduction

In this chapter, the image processing library is introduced, and a few functions from the library are tested. The image processing applications usually need huge memories. So the memory is important. The 'Memory Model' is also introduced. In this chapter, instead of rebuilding an example project, a library is directly used in the project.

2) Memory Models:

TI has made multiple addressing modes in DSP processors. Most DSPs support the traditional 'absolute' and 'relative' addressing mode. Also, TI adds multiple flavors of these addressing modes with better performance. For example, a processor with 16MB of addressing range uses a 24-bit address. So when the address is stored as part of the instruction, the encoding is at least 32-bit (24-bit address+ 8-bit instruction). Now assume a typical application that only uses the memory from 0 to 64KB. To address the first 64KB memory, only a 16-bit address is needed. So if the processor has extra instructions with a 16-bit address range, then the code space is reduced, and the load speed from memory is improved (because the instructions are smaller).

But, can compiler use these reduced-size instructions when compiling a C code? When the compiler compiles the code, it does not know the address of any section in memory. The sections address is assigned in the link phase (after compilation). So it is risky for the compiler to assume, the user will put sections in the first 64kB of memory because the compiler has no access to the linker (the linker can start only when all files already compiled). To solve this problem, TI introduced the 'Memory Model'. The 'Memory Model' ensures the compiler that the command file follows specific rules. So the compiler can safely use more efficient instructions. If the wrong 'Memory Model' is chosen for a project, it can create very odd and sophisticated bugs.

So the 'Memory Model' and the linker command file are dependent on each other. Fortunately, each family has only a few memory models, and rules for each memory model are simple.

2.1) Memory model Rules:

The 'Memory Model' names in 5000 and 2000 series are 'small' and 'large'. In 6000 family, the "Memory Models" are 'near' and 'far'.

Appendix A

DSPLab Registers

DSP Gig Inc.
Kyanoosh Shafaei

1) Introduction

Inside DSPLab hardware, there are two DSPs and two FPGAs. The user only has access to one of the DSPs. The other DSP and two FPGAs are for creating essential peripherals such as ADC, DAC, UART, Image, PLL, and timers. On the top side of the DSPLab aluminum box, the schematic for most of the peripherals is shown.

Designing DSP and FPGA hardware is beyond the scope of this book, but in this chapter, all the peripheral registers and their programming are covered.

2) DSPLab peripherals

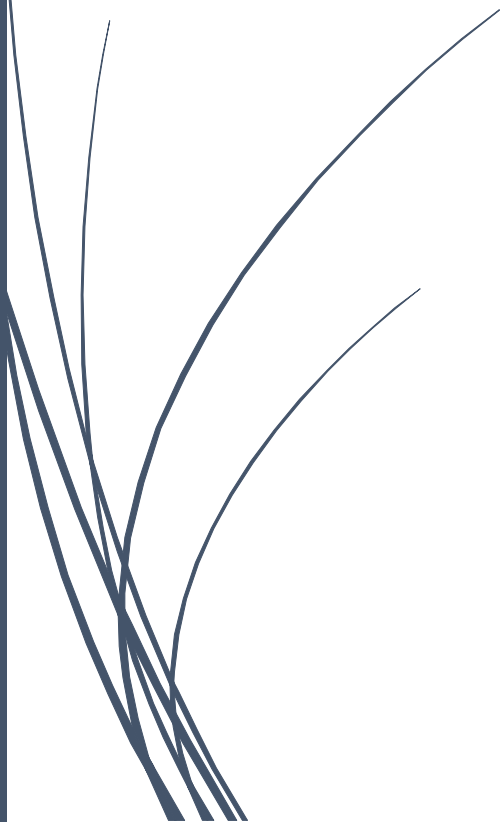
Inside DSPLab, several external devices are connected to the DSP through internal FPGA. This chapter is a good example of how to design a new peripheral for the DSP and program it easily using memory-mapped registered. To make DSP code even simpler, a FIFO is added to all new peripherals. The FIFO keeps the data and can send an interrupt to the processor.

In real-time software, the data movement, between the peripheral and the core, requires interrupt. The interrupt handler usually needs a few hundred cycles. If the data rate is high, a lot of cycles can be used by the interrupt. Reducing the number of interrupts is always part of successful real-time software design. The FIFO can keep the data and effectively can reduce the number of interrupts.

The next table lists most of the registers added inside the FPGA and can be accessed from the DSP (using EMIF).

Appendix B

‘struct’ and ‘union’ in embedded C



DSP Gig Inc.
Kyanoosh Shafaei

This appendix, is recommended for advanced C programmers.

1) Introduction

C language has some excellent features which can be used in embedded system programming. To program registers with multiple fields, a combination of 'union' and 'structure' can create the most optimized C code. In this chapter, as an example, the ADC registers for DSPLab are accessed using this method. The HAL (hardware Abstraction Layer) libraries for embedded systems (like TI Chip Support Library) usually use a similar approach.

2) What is 'union'?

In C, 'union' is very similar to 'structure'. But in 'structure', every member has a separated place in memory, while all members in 'union' share the same memory location.

For example, the following structure needs 2 words in memory:

```
struct A{
  short  i;   // Occupies word 1 in memory
  short  j;   // Occupies word 2 in memory
}obj1;
```

while the following union uses only one word in memory:

```
union B{
  short  i;   // Occupies word 1 in memory
  short  j;   // overlap with i. Occupies no extra memory
  short  k;   // overlap with i. Occupies no extra memory
  short  m;   // overlap with i. Occupies no extra memory
} obj2;
```

In the above 'union', 'obj2.i', 'obj2.j', 'obl2.k', and 'obj2.m' are pointing to the same location in memory.

Appendix C

Initialization code for DSPLab

DSP Gig Inc.
Kyanoosh Shafaei

This appendix is recommended for advanced DSPLab users.

1) Introduction

A DSP hardware needs some necessary register initialization for proper operation. These registers should be initialized after hardware reset. There are 4 common methods for initialization:

- 'Gel' file: The 'gel' file is another approach with one significant limitation. It is only for debugging (while JTAG is connected) and cannot be used in the standalone product.
- Bootloader: The bootloader is usually a ROM code that is executed after reset and loads the code from external storage (such as a flash) into internal or external memory. The first part of the data in external storage can have optional commands for bootloader. The bootloader can program internal registers based on the commands in external storage. These commands can initialize memories before any data is loaded. Or they can program the 'PLL' to increase chip frequency to increase the boot speed.
- Before '_c_int00': An initialization assembly function can be executed before any other function. The assembly function can be executed as the first function (even before '_c_int00'[1]). Then at the end, with a Jump/Branch instruction, the code execution is transferred to '_c_int00'. This kind of initialization is useful for memory section without any initial value after reset (such as '.bss' section).
- After '_c_int00': A C/assembly initialization function can be called as part of the 'main()' function. This kind of initialization is for peripheral programmings, such as 'PLL' (to increase the processor frequency). Many peripherals can be programmed inside 'main()' before being used by other parts of the code. Just memory is needed to be detected by the processor before any code is loaded into that memory.

The next picture shows why sometimes an early initialization may be needed. Although it is not necessary to have all these initialization processes for every application, it is helpful to know when to use them.

[1] - '_c_int00' is the first function executed in any C project and does the C initializations.

Appendix D

Flash programming

DSP Gig Inc.
Kyanoosh Shafaei

This appendix is for the users how are interested in using the internal parallel 512kB flash memory in DSPLab.

1) Introduction

DSPLab has a 512kB parallel flash memory. The flash part number is AM29LV400B. The next picture shows the flash connection to the processor.

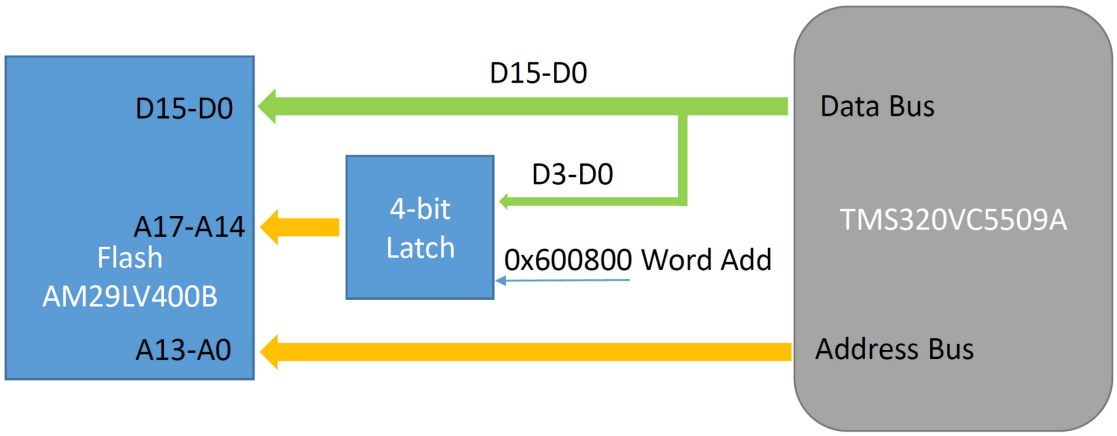


Figure 1: The extra 4-bit latch at address 0xC01000 B (0x600800 word)

Due to a hardware limitation, a latch for the four upper address bits is added to the hardware at address 0x600800 Word. Because the AM29LV400B is 256kWord, it has 18 address pins (A0 to A17). The lowest 14 pins (A0 to A13) are directly connected to the processor's address bus. But the 4 MSB pins (A14 to A17) are conneted to a 4-bit Flip-Flop and they need to be updated when the 4-bit MSB address is changing.

Because writing to a flash envolves special steps, the code for writing to the flash are explained here. The source code can be downloaded from *wiki.dspgig.com/dsplab/flash.zip*.