

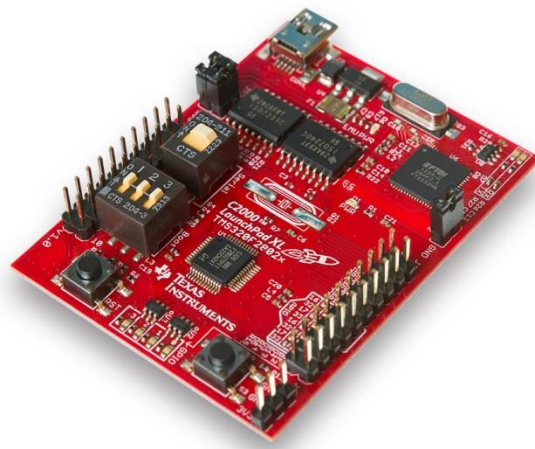
MICHIGAN STATE UNIVERSITY

Introduction to Programming C2000 Piccolo Launchpad

ECE480 Team 8

Daniel S. Chen

11/11/2014



The C2000 Piccolo Launchpad is an evaluation platform that allows the user to practice real-time control programming on the C2000 Piccolo microcontrollers. The Launchpad is based on the Piccolo TMS320F28027 with features such as 12bit ADC, 8PWM channels, I2C, SPI, UART, and 64KB of on board flash memory, etc. It utilizes an integrated isolated XDS100 JTAG emulator for easy programming and debugging. In addition, it also has 40 PCB pins that are available for easy access to the pins of the F28027 microcontroller and programmable/reset button.



To help an individual how to program, debug, and communicate with real-world using the C2000 Piccolo Launchpad. The main focuses are on using IDE (Integrated Development Environment), programming methods, and access to helpful information.

Overview

To successfully understand and integrate a microcontroller, one would need to become proficient with IDE, which is an interface that allows user to program and debugging the Launchpad. Secondly, there are various ways to communicate with the Launchpad such as programming language and programming structure that is also important to utilize the microcontroller to its full potential. Lastly, it is also important for the user to understand how to gain access to useful resources such as peripheral guidelines, register maps, or potential application tutorials.

Recommendation

Integrated Development Environment (IDE)

Code Composer Studio (CCS) is a common IDE that supports Texas Instruments' Microcontroller and embedded processors. It comprises a suite of tools that can be used to develop and debug embedded applications. It includes an optimizing C++/C compiler, debugger, profiler, project build environment, and source code editor.

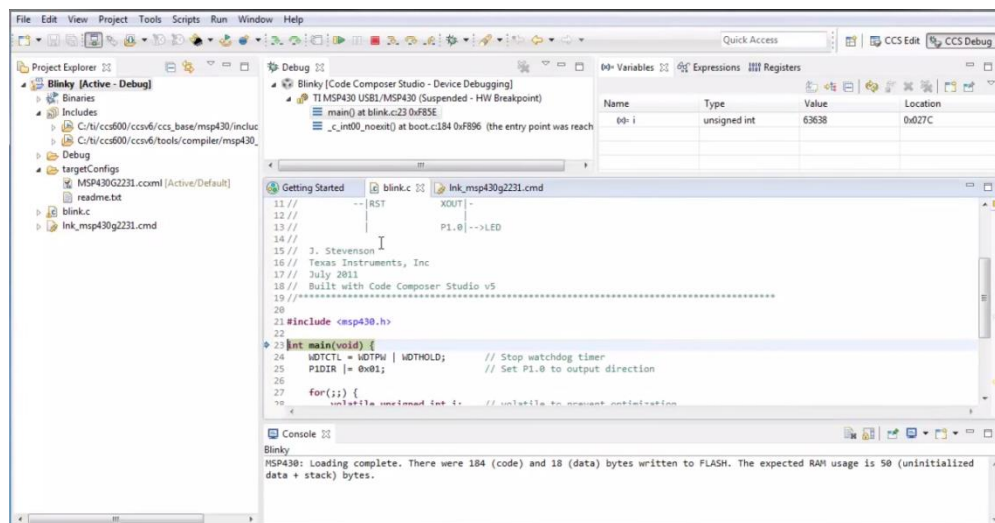


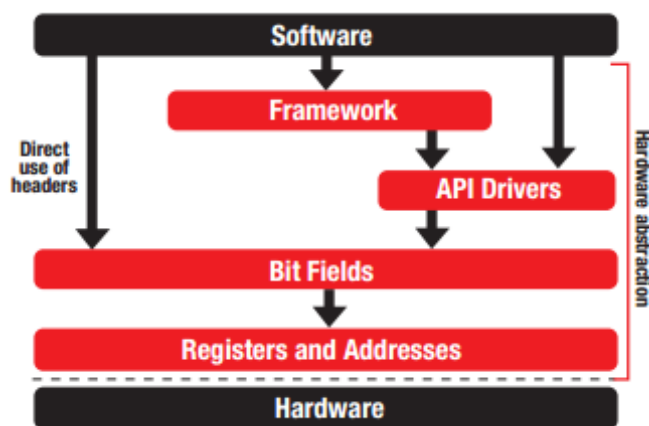
Figure 2: CCS working interface

Some of the advantages of developing using the Code Composer Studio:

- Include tools for C2000 such as BIOS real-time operation system, an emulator, flash programmer, etc.
- Interrupt debugging is possible by interrupt servicing even when the main program is halted

- Real-time watch windows and graphical aid in software verification and debugging
- Real-time debugging circuitry on C2000 devices

In addition to the IDE, it is also critical to have controlSUITE available during the development process. controlSUITE is a cohesive set of software infrastructure and software tools that is designed to minimize the development time. It provides device-specific driver as well as support software and system examples for some of the sophisticated system application. The controlSUITE comprise of 4-level Hardware Abstraction Layer.



Level 1 – Register and Addresses

- Baseline assembly or C communication to all hardware registers and addresses

Level 2 – Bit Fields

- Flexibility to access a register as a whole or by bits
- Bit fields can be manipulated without masking
- View in CCStudio watch window as variables

Level 3 – API Drivers

- Accelerate learning experiences for new programmer
- Common tasks and peripheral modes supported

Level 4 – Framework

- Function-based device initialization
- System management via state machines set to run at specific frequency
- Pre-configured GPIO mapping

Programming the C2000 Launchpad

Programming Language

The programming of the C2000 microcontroller can be done either using C or C++ language. The selection is based on the user's preferences since that there are advantages and disadvantages between the two. Below are some common comparisons between the two programming language. It is upon the user to determine the best way to program the microcontroller for maximum performance.

	C	C++
Typing Discipline	Weak, Static	Strong, Unsafe, Nominative
Paradigms	Imperative systems implementation language	Object-oriented programming
Garbage Collection	Allows better management of memory manually	No
Speed	Faster to compile and execute than C++	Slightly slower if not proficient with the language

Table 1: Comparison between C and C++

Programming Model

In addition to the two programming language, there are two programming methods that can be implemented, Direct Register Access Model and Software Driver Model. Both of the programming models can be implemented independently or a combination of both is also valid. Each of the programming models has its own advantages and disadvantages as showing below:

	Advantages	Disadvantages
Direct Register Access Model	Smaller code footprint Faster code execution	Codes are obscure Detailed knowledge of each register is required
Software Driver Model	Larger code footprint Slower code execution	Codes are viewable and understandable

Table 2: Comparison between programming models

Direct Register Access Model

As the name suggest, the Direct Register Access Model (DRAM) writes value directly to the individual peripheral registers. All of the peripheral registers are defined in the corresponding header file for specific device. It is therefore important for the user to include such header file along with the developed program. An example of DRAM is demonstrated as follow:

```
AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;  
AdcRegs.ADCINTFLGCLR.bit.ADCINT2 = 1;
```

This particular code clears the ADC interrupt flag for ADC interrupt 1 and 2.

Software Driver Model

The Software Driver Model (SDM) utilize an API(application programming interface) provided by the peripheral driver library. In which can be used by applications to control peripherals. The driver header file needs to be included in this case to utilize SDM and a “handle” to specific peripheral needs to be initialized. An example of SDM is shown as follow:

```
/** Include for the pwm.h header file */  
#include "f2802x_common/include/pwm.h"  
/** Initialize handles for ePWM1 and ePWM2 */  
myPwm1 = PWM_init((void *)PWM_ePWM1_BASE_ADDR,  
sizeof(PWM_Obj));  
myPwm2 = PWM_init((void *)PWM_ePWM2_BASE_ADDR,  
sizeof(PWM_Obj));  
/** Examples of API function calls using handles */  
PWM_setPeriod(myPwm1, 3000);  
PWM_setClkDiv(myPwm2, PWM_ClkDiv_by_1);
```

Suggestion for choosing programming model

How the programming models are implemented are affect by its application. A software driver model is recommended if code size and execution time is not a consideration for the particular application. A combination of both software driver and direct register access model is recommended if code size is not an issue but timing is critical and speed of execution is significant. If both the code size and timing constraints are critical, then the direct register access model is recommended.

Another DRAM Example

```
/*Configure Clock configuration for 50MHz*/
SYSCTL_RCC_R = 0x01C00380;

/*Enable port A (SPI)*/
SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOA;

/*Configure input/output*/
GPIO_PORTA_DIR_R = 0x2C;

/*Enable P(in)A SPI pins*/
GPIO_PORTA_DEN_R = 0x3C;
GPIO_PORTA_AFSEL_R |= 0x34;

/*Enable the peripheral clock of SSI*/
SYSCTL_RCGC1_R = SYSCTL_RCGC1_SSI0;

/*Configure role as master*/
SSI0_CR1_R = 0x00000000;

/*SSI Clock pre-scale register: Will run at 2MHz*/
SSI0_CPSR_R = 0x00000005;

/*Enable SSI*/
SSI0_CR1_R = 0x00000002;
```

Another SDM Example

```
//Blink the blue LEDs in sequence until the pushbutton is pressed
while(GPIO_getData(myGpio, GPIO_Number_12) != 1)
{
    GPIO_setHigh(myGpio, GPIO_Number_0);
    GPIO_setHigh(myGpio, GPIO_Number_1);
    GPIO_setHigh(myGpio, GPIO_Number_2);
    GPIO_setLow(myGpio, GPIO_Number_3);
    DELAY_US(50000);

    GPIO_setHigh(myGpio, GPIO_Number_0);
    GPIO_setHigh(myGpio, GPIO_Number_1);
    GPIO_setLow(myGpio, GPIO_Number_2);
    GPIO_setHigh(myGpio, GPIO_Number_3);
    DELAY_US(50000);

    GPIO_setHigh(myGpio, GPIO_Number_0);
    GPIO_setLow(myGpio, GPIO_Number_1);
    GPIO_setHigh(myGpio, GPIO_Number_2);
    GPIO_setHigh(myGpio, GPIO_Number_3);
}
```

```
    DELAY_US(50000);

    GPIO_setLow(myGpio, GPIO_Number_0);
    GPIO_setHigh(myGpio, GPIO_Number_1);
    GPIO_setHigh(myGpio, GPIO_Number_2);
    GPIO_setHigh(myGpio, GPIO_Number_3);
    DELAY_US(500000);
}
```

Resources for optimal experiences

It is always helpful to have the most information available when working with a microcontroller. Therefore, resources such as specific microcontroller's driver, peripheral datasheet, or even training tutorial can be helpful during the development process of the embedded system. Such useful information is listed in the Reference section at the end of this document.

Reference

http://processors.wiki.ti.com/index.php/Main_Page

<http://www.ti.com/ww/en/launchpad/launchpads-c2000-launchxl-f28027.html#tabs>

<http://www.ti.com/lit/ml/sprb199a/sprb199a.pdf>

<http://www.ti.com/tool/launchxl-f28027>

<http://www.ti.com/lit/ds/sprs523j/sprs523j.pdf>

<http://www.ti.com/lit/ug/spru566k/spru566k.pdf>

http://www.diffen.com/difference/C_vs_C%2B%2B