The Introduction and Applications of PBMCUSLK (Project Board)

Reported by: Shen Xu

Date: 12-06-2007
Contents

Section 1: Review of PBMCUSLK Project Board

Section 2: The Features of PBMCUSLK Project Board

Section 3: Benefit from Using PBMCUSLK Project Board with other controllers

Section 4: Review of Programming Software

Section 5: Evaluation of PBMCUSLK with two applications

Section 6: Problems of PBMCUSLK

Section 6: Future work of PBMCUSLK

Appendix A: the code of first application

Appendix B: the code of second application
Section 1: Review of PBMCUSLK Project Board

The PBMCUSLK is a full-featured prototyping platform intended for interfacing and programming Freescale MCU development modules in an educational environment. A line of HC(S) 12(X), HC(S) 08, DSP, and ColdFire modules plug directly into the project board. Other MCU boards can be interfaced directly to the project board by ribbon cable. The PBMCUSLK may also be used as an electronic circuit prototyping environment without MCU support.

An integrated USB BDM POD has been provided to allow the user to program, erase, and debug supported Freescale MCU modules.

The Project Board Student Learning Kit is recommended for use across multiple courses ranging from introductory electronics to microprocessor applications, and senior design projects. The Project Board is best suited for lab-based courses requiring students to prototype a stand-alone electronic circuit or interface control logic to a microprocessor.
Section 2: The Features of PBMCUSLK Project Board

- **Power:**

  The PBMCUSLK may be used as a stand-alone prototyping platform or in conjunction with the NI-ELVIS platform. The project board will accept power input from the included wall-plug transformer or from the NI-ELVIS workstation. The project board may also be powered from the integrated USB BDM.

- **User's I/O:**

  The PBMCUSLK provides an array of User I/O to allow connection of auxiliary components such as signal input, test equipment, Keypads, or LCD displays. Many of these user features are by default enabled, but can be disconnected through jumper settings.

And here, let me describe some details of the I/O in the project board.

**LCD Port**

The PBMCUSLK includes* an 8-char x 2-line LCD module to support application development requiring character display output. The display is connected by default to the MCU PORT pins through jumpers allowing direct interface to Freescale line of plug-in application modules.

**Push button Switch**
Each push button switch is configured for active-low operation. When pressed (closed) the associated signal line is pulled to GND through a 1 kΩ, current-limit resistor. A 10k ohm resistor pulls each signal line to VDD when the switch is released (open).

**DIP Switch**

Each DIP switch is configured for active-high operation. When ON (closed), each switch leg is individually pulled to VDD through a 100 Ω series, current limit resistor. A 10k ohm resistor pulls each signal line to GND when the switch is OFF (open).

**LED**

The PBMCUSLK provides 8, green LED’s, for use as output indicators. Each LED is configured for active-high operation. Each LED is individually driven by an ACT buffer allowing either 5V or 3.3V input levels. The input level is determined by VDD selection. A 10K ohm resistor holds each buffer input low to prevent inadvertent LED activation.

**Keypad**

The KEYPAD connector supports connection of a passive 12-key or 16-key keypad. The KEYPAD connector is routed directly to the signal breakout header labeled KEYPAD located adjacent to the breadboard. No current-limit is provided on this connection and should be provided by the user if required.

**Potentiometer**
The PBMCUSLK provides a single-turn, 5K ohm trim potentiometer for use in circuit prototyping. Most commonly, the POT may be used to provide analog input signals to the microcontroller.

**Banana Jack**

The PBMCUSLK provides one BNC jack for use as auxiliary I/O. This connector may be used for auxiliary signal input or for signal output to test equipment.

**Buzzer**

The PBMCUSLK features an external drive buzzer for audible applications. The buzzer is connected to a TIMER / PWM port on all current MCU modules.

**Section 3: Benefit from Using PBMCUSLK Project Board with other controllers**

In this section, I will introduce the main benefits for us to use this PBMCUSLK Project Board.

For the lab, we will use the MCU microchip to do the experiments. But in our microchip, such as MC9S12C32, which is a 12 bit micro controller, there are just two LED in the
board. We can not design experiments if the output of the experiments limit within these two LED. More over, in some other experiments we want to design, we need design some external circuit and use two or even more microchip for communication. It is obviously the single microchip can not fit for our requirements.

Based on the requirements below, the PBMCUSLK project board is our best choice. In the last section, I present the I/O parts of it. We can find out that the I/O parts, such as buzzer, LCD, and switches, are all can be used for the students’ experiments. More over, we can also build our own circuit in the project board for deep development. What we need do is just push our microchip into the J ports of this PBMCUSLK.

Section 4: Review of Programming Software

In this section, I will introduce some basic feature of the development software for our micro controllers, CodeWarrior. And here, I describe detail advantages of using CodeWarrior.

- Multiple-language support

Choose from multiple programming languages when developing software. The IDE supports high-level languages, such as C, C++, and Java, as well as in-line assemblers for most processors.

- Consistent development environment

Port software to new processors without having to learn new tools or lose an existing code base. The IDE supports many common desktop and embedded processor families.
Plug-in tool support

Extend the capabilities of the IDE by adding a plug-in tool that supports new services. The IDE currently supports plug-ins for compilers, linkers, pre-linkers, post-linkers, preference panels, version controls, and other tools. Plug-ins make it possible for the CodeWarrior IDE to process different languages and support different processor families.

Here is the feature of CodeWarrior, which is copied from the Guide of it.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Purpose</th>
<th>CodeWarrior IDE Features</th>
</tr>
</thead>
</table>
| **Project Manager** | Manipulate items associated with a project | • Handles top-level file management for the software developer  
                    |                                                       | • Organizes project items by major group, such as files and targets  
                    |                                                       | • Tracks state information (such as file-modification dates)  
                    |                                                       | • Determines build order and files to be included in each build  
                    |                                                       | • Coordinates with plug-ins to provide version-control services |
| **Editor**   | Create and modify source code         | • Uses color to differentiate programming-language keywords  
                    |                                                       | • Allows definition of custom keywords for additional color schemes  
                    |                                                       | • Automatically verifies parenthesis, brace, and bracket balance  
                    |                                                       | • Allows use of menus for navigation to any function or into the header files used by the program |
| **Search Engine** | Find and replace text                  | • Finds a specific text string  
                    |                                                       | • Replaces found text with substitute text  
                    |                                                       | • Allows use of regular expressions  
                    |                                                       | • Provides file-comparison and differencing functionality |
Section 5: Evaluation of PBMCUSLK with two applications

To evaluate this PBMCUSLK, I developed some simple code of using a HCS12 family microcontroller (MCU) in conjunction with this project board.

In the two experiments, I used CodeWarrior for programming and burning the code into my HCS 12 microcontroller (MC9S12C32). I put this microcontroller into the J1 ports of PBMCUSLK, and connect the project board to my computer with USB cable. Then, the USB cable takes care of power and the communication tasks.

- First experiment: Test the connection

In this experiment, I use a simple program which burned into MC9S12C32 to control two LED in the microcontroller. This program was written in the CodeWarrior platform. Here is

<table>
<thead>
<tr>
<th>Tool</th>
<th>Purpose</th>
<th>CodeWarrior IDE Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Manage and view program symbols</td>
<td>• Maintains a symbolic database for the program. Sample symbols include names and values of variables and functions.</td>
</tr>
<tr>
<td>Browser</td>
<td></td>
<td>• Uses the symbolic database to assist code navigation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Links every symbol to other locations in the code related to that symbol</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Processes both object-oriented and procedural languages</td>
</tr>
<tr>
<td>Build</td>
<td>Convert source code into an executable file</td>
<td>• Uses compiler to generate object code from source code</td>
</tr>
<tr>
<td>System</td>
<td></td>
<td>• Uses linker to generate final executable file from object code</td>
</tr>
<tr>
<td>Debugger</td>
<td>Resolve errors</td>
<td>• Uses symbolic database to provide source-level debugging</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Supports DWARF (1.1 and 2.0) and the HIWAVE object file format</td>
</tr>
</tbody>
</table>
If it works, it means that the connection between this microcontroller and the project board is ready to work.

The code of this part is in the Appendix A.

- **Second experiment: Test the Project Board**

From the first experiment, we can see that the connection is OK. However, in that experiment, we did not use any I/O part of the project board. It also means that the advantage of this project board does not be shown. So in the second experiment, I develop a program to test the I/O parts of the project board.

Such as the last one, I use CodeWarrior to do the programming; and also use it to burn program into MC9S12C32. This is the figure of debugging and burning, which show the assembling code of it, the value in memory of microcontroller, and the value in registers.
After burning the code into MC9S12C32, the program does work independently. At this time, LED 1-4 on the project board are beginning to cycle and the LCD displays PBS12C32SLK_DEMO. Press PB1 on the project board and adjust the potentiometer. The buzzer is enabled with a 440Hz frequency and the potentiometer value is sampled to control the tone of the buzzer while the button is pressed. Also, note that while pressed the LCD display is now showing “PBMCUSLK BUZZER2”.

This experiment shows that the control of I/O part such as buzzer, potentiometer, and LCD on the project from MC9S12C32 in it. It means that we can develop some other experiment based on the project board.
Section 6: Problems of PBMCUSLK

At first, I want to evaluate PBMCUSLK of using the 8 bit microcontroller, which we have a lot. But I find out that the 8 bit microcontroller can not connect to this project board directly. More over, almost all the documents and demo from Freescale are using 12 bit microcontroller.

So should we use 12 bit microcontroller in our experiment design in future?

Section 6: Future work of PBMCUSLK

Based on the I/O parts I have tested in this report, I will design some lab experiments with the use of oscilloscope we have.

In this report, I did not test the development of external circuit and the communication function of microcontrollers in the project board. I will figure out how to use these function if we need it in our future lab experiments.
Appendix A: the code of first application

```c
#include <hidef.h>         /* common defines and macros */
#include <mc9s12c32.h>     /* derivative information */
#include "aps12c32slk.h"   /* I/O definitions and init routines */

#pragma LINK_INFO DERIVATIVE "mc9s12c32"

void main(void) {

    INIT_APSLK_PBs;
    INIT_APSLK_LEDs;
    // INIT_APSLK_POT;

    EnableInterrupts;

    for(;;) {
        /* PB1 = LED1 */
        if (APSLK_PB1) {
            APSLK_LED1 = APSLK_PB1;     // LED 1 Active
        }
        /* PB2 = LED2 */
        else if(APSLK_PB2) {
            APSLK_LED2 = APSLK_PB2;     // LED 2 Active
        }
        else{
            APSLK_LED2 = OFF;
            APSLK_LED1 = OFF;
        }

        } /* wait forever */
    }
```
Appendix B: the code of second application

```c
#include <hidef.h>         /* common defines and macros */
#include <mc9s12c32.h>     /* derivative information */
#include "pbs12c32slk.h"   /* I/O definitions and init routines */
#include "lcd.h"           /* LCD definitions and init routines */

#pragma LINK_INFO DERIVATIVE "mc9s12c32"

/* Global Variables */

long int count = 0;
int divisor = 0;
int refresh = 5;
int up = 1;
int LED_SPEED = 255;

void main(void) {

    LCDInit(); /* Initialize LCD Display */

    INIT_PBMCLUSLK_PBs;
    INIT_PBMCLUSLK_LEDS;
    INIT_PBMCLUSLK_POT;
    INIT_PBMCLUSLK_BUZZER;
    INIT_PBMCLUSLK_MOTOR;

    EnableInterrupts;
```
for(;;) {
    /* 440 Hz Buzzer Routine */
    if (!PBMCUSLK_PB1)
        {
            if (refresh != 1){
                refresh = 1;
            }
            PBMCUSLK_LED1 = PBMCUSLK_PB1;     //LED 1 Active
            PBMCUSLK_BUZZER_TONE = A4_440Hz;  //440Hz Default Tone
            PBMCUSLK_BUZZER_VOLUME = (PBMCUSLK_BUZZER_TONE * PBMCUSLK_POT) >> 9;
        }
    /* 880 Hz Buzzer Routine */
    else if(!PBMCUSLK_PB2)
        {
            if (refresh != 2){
                LCDPutString("PBMCUSLK\n");
                LCDPutString("BUZZER2\n");
                refresh = 2;
            }
            PBMCUSLK_LED2 = PBMCUSLK_PB2;     //LED 2 Active
            PBMCUSLK_BUZZER_TONE = A5_880Hz;  //880 Hz Default Tone
            PBMCUSLK_BUZZER_VOLUME = (PBMCUSLK_BUZZER_TONE * PBMCUSLK_POT) >> 9;
        }
else if (!PBMCUSLK_PB3) {
    if (refresh != 3) {
        LCDPutString("MOTOR\n");
        LCDPutString("CONTROL\n");
        refresh = 3;
    }

    PBMCUSLK_LED3 = PBMCUSLK_PB3;
    PBMCUSLK_MOTOR_SPEED = PBMCUSLK_POT;
}

else if (!PBMCUSLK_PB4) {
    if (refresh != 4) {
        LCDPutString("LEDSPD\n");
        LCDPutString("CONTROL\n");
        refresh = 4;
    }
    LED_SPEED = PBMCUSLK_POT + 0x03; // Set LED speed, the +0x03 is to give LEDs enough time to refresh
}
else {
    /* Default Welcome Screen Display */
    if (refresh != 5) {
        LCDPutString("PBS12C32\n");
        LCDPutString("SLK_DEMO\n");
        refresh = 5;
    }
    PBMCUSLK_BUZZER_TONE = 0;
}
PBMCUSLK_BUZZER_VOLUME = 0;
}

if (up == 1 | count == 0){
    up = 1;
    count ++;
} else {
    count --;
}

divisor = (count/(LED_SPEED*50));
switch(divisor){
    case 0:
        PBMCUSLK_LED1 = ON;
        PBMCUSLK_LED2 = OFF;
        PBMCUSLK_LED3 = OFF;
        PBMCUSLK_LED4 = OFF;
        break;
    case 1:
        PBMCUSLK_LED1 = OFF;
        PBMCUSLK_LED2 = ON;
        PBMCUSLK_LED3 = OFF;
        PBMCUSLK_LED4 = OFF;
        break;
    case 2:
        PBMCUSLK_LED1 = OFF;
        PBMCUSLK_LED2 = OFF;
        PBMCUSLK_LED3 = ON;
        PBMCUSLK_LED4 = OFF;
        break;
case 3:
    PBMCUSLK_LED1 = OFF;
    PBMCUSLK_LED2 = OFF;
    PBMCUSLK_LED3 = OFF;
    PBMCUSLK_LED4 = ON;
    break;

case 4:
    up = ~up;
    break;

default:
    PBMCUSLK_LED1 = OFF;
    PBMCUSLK_LED2 = OFF;
    PBMCUSLK_LED3 = OFF;
    PBMCUSLK_LED4 = OFF;
    count = 0;
    break;

} /* wait forever */

}