**Principles of Embedded Software (Bruce Montgomry ) - Project 2**

**-Utkarsh Dviwedi and Harsh Rathore**

**Part 1** – Create a Work Breakdown Structure (WBS) (10 points)

Small (S) = less than 2 hours

Medium (M) = 2 to less than 4 hours

Large (L) = 4 to less than 8 hours

Extra Large = 8 hours or more

So each lowest level sub task of your WBS should say something like:

Test code – Bruce (M)

**Part 2**

**Make fb\_run**: This version of the program will run natively on the Freedom board.

1) The program is required to flash the LED in a specific provided pattern (see end of project description) of on and off periods which you will control with a simple timing delay loop. Your program must meet the timing requirements of each pattern command.

2) (Optional – 5 points extra credit) When in the timing loop you will look for a press event from the capacitive touch slider on the board. If you press the slider on its left side, the LED will change color to RED. If you press it in the middle, the LED will change to GREEN. On the right will cause it to go to BLUE. This change will occur the next time a command is sent to the LED to turn on (not immediate).

Note that the reference below for slider support will not work

directly with our IDE.

2) If not using the slider, the color of the LED should be changed after every three on and off

cycles.

3) The program will run through the provided timing pattern ten complete cycles, and will then

end.

**Make fb\_debug:** This version of the program will also run natively on the Freedom board

1) It must perform all the functions as for the fb\_run version

2) It will be able to send messages via UART to a serial terminal (you can use the built in

MCUXpresso terminal). These messages should include the current command LED GREEN ON or

LED BLUE OFF, for example, along with a timestamp and the elapsed time since the last

command. Individual output debug lines will look like this:

LED GREEN ON 14:22:17 973

2A) (Optional – 5 extra credit points) In order to do the above, you will need to initialize the

Real Time Clock on the KL25Z board, as there is no other timing source. If you do so you can

generate time stamps for the embedded version of the program as shown above.

2B) If you do not want to run the Real Time Clock, you may change the KL25Z board debug

lines to read as follows:

LED GREEN ON 72000

Where the number in the debug line is the value provided to your wait loop for that particular

LED state.

3) You may use the serial output for other messages such as program start and end.

4) Some form of IFDEF in the code will likely manage whether these messages are sent or not.

**Make pc\_run:** This version of the program will run locally on your PC in MCUXpresso, sending results to

the debug console.

1) This version of the code should generally run the same as the fb\_run version.

2) In the PC version of the program, instead of toggling the LED on and off, you must send the text

LED GREEN ON or LED BLUE OFF as an alternative to triggering the LED on the Freedom board.

This should be done in the same code modules that run on the Freedom board, perhaps using

IFDEF clauses for alternate behavior. NOTE – the timing of these cycles should still match the

timing of LED activations on the board, so you may need to change to alternate timing loops for

local PC execution.

3) Also, in the PC version of the code, there is no equivalent to the slider, so that functionality

should be disabled, and the color of the LED should be changed after every three on and off

cycles in a set of alternate code.

**Make pc\_debug:**

1) This version of the code should generally run as the pc\_run version.

2) This version of the program will add sending debug print lines to the debug console as above,

but you must include time stamps and time since last event in milliseconds on the print line,

which will look like this:

LED GREEN ON 14:22:17 973

3) You may use the debug console for other messages such as program start and end.

The timing Cycle is

3000,1000,2000,600,1000,400,1000,200,500,100,500,100,500,100,1000,200,1000,400,2000,600

Makefile Notes

WE MAKE AN EMPTY PROJECT AND ADD ALL THE NECESSARY BOARD FILES FROM A NEW COMPLETELY CREATED PROJECT. THEN WE ADD THE VARIABLES, ESSENTIALLY THE FLAGS WE WILL BE USING TO GENERATE AN ECECUTABLE VERSION OF THE C SOURCE. THEN WE DEFINE BUILD TARGETS FOR PC, IN ITS RUN AND DEBUG MODE, AND FOR ARM RUN AND DEBUG MODE. WE WRITE THE DIFFERENT VARIABLES FOR COMPILATION OF C SOURCE FILE IN ARM EXECUTABLE FILE AND PC EXECUTABLE VERSION

**The contents of the main.c file are**

/\*

\* Copyright 2016-2019 NXP

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without modification,

\* are permitted provided that the following conditions are met:

\*

\* o Redistributions of source code must retain the above copyright notice, this list

\* of conditions and the following disclaimer.

\*

\* o Redistributions in binary form must reproduce the above copyright notice, this

\* list of conditions and the following disclaimer in the documentation and/or

\* other materials provided with the distribution.

\*

\* o Neither the name of NXP Semiconductor, Inc. nor the names of its

\* contributors may be used to endorse or promote products derived from this

\* software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND

\* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED

\* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE

\* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR

\* ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

\* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

\* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON

\* ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

\* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS

\* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

/\*\*

\* @file MKL25Z128xxx4\_Project\_LedBlink.c

\* @brief Application entry point.

\*/

**#include** <stdio.h>

**#include** "board.h"

**#include** "peripherals.h"

**#include** "pin\_mux.h"

**#include** "clock\_config.h"

**#include** "MKL25Z4.h"

**#include** "fsl\_debug\_console.h"

/\* **TODO**: insert other include files here. \*/

/\* **TODO**: insert other definitions and declarations here. \*/

/\*

\* @brief Application entry point.

\*/

//Function to cout delay

**void** **delay**(**volatile** int32\_t number)

{

**int** multiplier = 4000; //for the Mhz system clock

**int** delay = number\*multiplier;

**while** (delay!=0)

{

**\_\_asm** **volatile** ("nop");

delay--;

}

}

//Function to Print the on the Semihost Console the PC mode values

**void** **print\_pc\_on**(**int** LED\_no, **int** delay\_total, **int** cumulative, **int** Debug\_mode)

{

**if** ((LED\_no == 0) && (Debug\_mode ==0)) //PC\_run Mode RED

{

**printf**("\n The Red LED is ON ");

delay(delay\_total);

**printf**("\n The Red LED is OFF");

}

**if** ((LED\_no == 0) && (Debug\_mode ==1)) //PC\_Debug Mode RED

{

**printf**("\n The Red LED is ON %d ms",cumulative);

delay(delay\_total);

**printf**("\n The Red LED is OFF %d ms", cumulative + delay\_total);

}

**if** ((LED\_no == 1) && (Debug\_mode ==0)) //PC\_run Mode Green

{

**printf**("\n The Green LED is ON ");

delay(delay\_total);

**printf**("\n The Red LED is OFF");

}

**if** ((LED\_no == 1) && (Debug\_mode ==1)) //PC\_Debug Mode Green

{

**printf**("\n The Green LED is ON %d ms",cumulative);

delay(delay\_total);

**printf**("\n The Green LED is OFF %d ms", cumulative + delay\_total);

}

**if** ((LED\_no == 0) && (Debug\_mode ==0)) //PC\_run Mode Blue

{

**printf**("\n The Blue LED is ON ");

delay(delay\_total);

**printf**("\n The Blue LED is OFF");

}

**if** ((LED\_no == 0) && (Debug\_mode ==1)) //PC\_Debug Mode Blue

{

**printf**("\n The Blue LED is ON %d ms",cumulative);

delay(delay\_total);

**printf**("\n The Blue LED is OFF %d ms", cumulative + delay\_total);

}

}

// Function to Print on FRB the debug values

**void** **print\_frb\_on**(**int** LED\_no, **int** delay\_total,**int** cumulative)

{

**if** (LED\_no == 0)

{

PRINTF("\n The Red LED is ON %d ms",cumulative);

delay(delay\_total);

PRINTF("\n The Red LED is OFF %d ms", cumulative + delay\_total);

}

**if** (LED\_no == 1)

{

PRINTF("\n The Green LED is ON %d ms",cumulative);

delay(delay\_total);

PRINTF("\n The Green LED is OFF %d ms", cumulative+ delay\_total);

}

**if** (LED\_no == 2)

{

PRINTF("\n The Blue LED is ON %d ms",cumulative);

delay(delay\_total);

PRINTF("\n The Blue LED is OFF %d ms",cumulative+ delay\_total);

}

}

/\*

void print\_frb\_off(int LED\_no, int delay\_total)

{

if (LED\_no == 0)

{

PRINTF("\n The Red LED is OFF %d ms",delay\_total);

}

if (LED\_no == 1)

{

PRINTF("\n The Green LED is OFF %d ms",delay\_total);

}

if (LED\_no == 2)

{

PRINTF("\n The Blue LED is OFF %d ms",delay\_total);

}

}

\*/

//Function to blink the Freedom Board LEDS

**void** **blink**(**int** LED\_no, **int** delay\_counter)

{

**if** (LED\_no==0) //LED is Red

{

LED\_RED\_ON();

delay(delay\_counter);

LED\_RED\_OFF();

}

**if** (LED\_no==1) //LED is Green

{

LED\_GREEN\_ON();

delay(delay\_counter);

LED\_GREEN\_OFF();

}

**if** (LED\_no==2) //LED is Blue

{

LED\_BLUE\_ON();

delay(delay\_counter);

LED\_BLUE\_OFF();

}

}

**int** **main**(**void**) {

/\* Init board hardware. \*/

BOARD\_InitBootPins();

BOARD\_InitBootClocks();

BOARD\_InitBootPeripherals();

/\* Init FSL debug console. \*/

BOARD\_InitDebugConsole();

/\* Force the counter to be placed into memory. \*/

**volatile** **static** **int** i = 0 ;

**volatile** **static** **int** mode = 1;

**volatile** **static** **int** LED\_no = 0 ;

//Lookup table

**int** Delay\_Arr[20]={3000,1000,2000,600,1000,400,1000,200,500,100,500,100,500,100,1000,200,1000,400,2000,600};

//Initialize the LEDs and GPIO

LED\_RED\_INIT(1);

LED\_GREEN\_INIT(1);

LED\_BLUE\_INIT(1);

//Tetting Printf functions on console and terminal

PRINTF("Hello World\n");

**printf**("hellow world pc debug mode here");

//////////////////////Declarations for the different run modes///////////////

**#ifdef** pc\_run

mode = 0;

PRINTF("\n PC Run mode")

**#endif**

**#ifdef** pc\_debug

mode = 1;

PRINTF("\n PC Debug mode")

**#endif**

**#ifdef** frb\_run

mode = 2;

PRINTF("\n Freedom Board Run mode")

**#endif**

**#ifdef** frb\_debug

mode = 3;

PRINTF("\n Freedom Board debug mode")

**#endif**

**int** delay\_count = 0;

/\* Enter an infinite loop, just incrementing a counter. \*/

**int** cumulative\_delay = 0;

**int** cumulative\_count = 0;

////////////////////////////////////PC run Mode//////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////

**if** (mode == 0)

{

**for** (i=1;i<=18;i++)

{

**if** (i == 20)

{

cumulative\_count++;

}

**if** ( cumulative\_count == 3)

{

**break**;

}

**switch** (i)

{

**case** 1: LED\_no = 0; // R on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,0);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 2: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 3: LED\_no = 0; //R on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,0);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 4: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 5: LED\_no = 0; //R on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,0);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 6: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 7: LED\_no = 1; //G on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,0);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 8: delay(Delay\_Arr[delay\_count]);

delay\_count++; //G off

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 9: LED\_no = 1; //G on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,0);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 10: delay(Delay\_Arr[delay\_count]); //G off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 11: LED\_no = 1;

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,0); //G on

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 12: delay(Delay\_Arr[delay\_count]); //G off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 13: LED\_no = 2;

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,0); //B on

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 14: delay(Delay\_Arr[delay\_count]); //B off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 15: LED\_no = 2; //B on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,0);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 16: delay(Delay\_Arr[delay\_count]); //B off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 17: LED\_no = 2; //Blue on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,0);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 18: delay(Delay\_Arr[delay\_count]); //Blue off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

} ////end of switch

**if** (i==18)

{

i=1;

}

**if** (delay\_count == 19)

{

delay\_count = 0;

}

/\* 'Dummy' NOP to allow source level single stepping of

tight while() loop \*/

//\_\_asm volatile ("nop");

} //end of for

} ///end of mode 0

/////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////PC Debug Mode//////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////

**if** (mode == 1)

{

**for** (i=1;i<=18;i++)

{

**if** (i == 20)

{

cumulative\_count++;

}

**if** ( cumulative\_count == 3)

{

**break**;

}

**switch** (i)

{

**case** 1: LED\_no = 0; // R on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,1);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 2: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 3: LED\_no = 0; //R on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,1);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 4: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 5: LED\_no = 0; //R on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,1);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 6: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 7: LED\_no = 1; //G on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,1);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 8: delay(Delay\_Arr[delay\_count]);

delay\_count++; //G off

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 9: LED\_no = 1; //G on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,1);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 10: delay(Delay\_Arr[delay\_count]); //G off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 11: LED\_no = 1;

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,1); //G on

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 12: delay(Delay\_Arr[delay\_count]); //G off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 13: LED\_no = 2;

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,1); //B on

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 14: delay(Delay\_Arr[delay\_count]); //B off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 15: LED\_no = 2; //B on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,1);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 16: delay(Delay\_Arr[delay\_count]); //B off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 17: LED\_no = 2; //Blue on

print\_pc\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay,1);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 18: delay(Delay\_Arr[delay\_count]); //Blue off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

} ////end of switch

**if** (i==18)

{

i=1;

}

**if** (delay\_count == 19)

{

delay\_count = 0;

}

/\* 'Dummy' NOP to allow source level single stepping of

tight while() loop \*/

//\_\_asm volatile ("nop");

} //end of for

} ///end of mode 1

/////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////Freedom Board run Mode//////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////

**if** (mode == 2)

{

**for** (i=1;i<=18;i++)

{

**if** (i == 20)

{

cumulative\_count++;

}

**if** ( cumulative\_count == 3)

{

**break**;

}

**switch** (i)

{

**case** 1: LED\_no = 0; // R on

blink(LED\_no,Delay\_Arr[delay\_count]);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 2: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 3: LED\_no = 0; //R on

blink(LED\_no,Delay\_Arr[delay\_count]);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 4: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 5: LED\_no = 0; //R on

blink(LED\_no,Delay\_Arr[delay\_count]);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 6: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 7: LED\_no = 1; //G on

blink(LED\_no,Delay\_Arr[delay\_count]);delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 8: delay(Delay\_Arr[delay\_count]);

delay\_count++; //G off

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 9: LED\_no = 1; //G on

blink(LED\_no,Delay\_Arr[delay\_count]);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 10: delay(Delay\_Arr[delay\_count]); //G off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 11: LED\_no = 1;

blink(LED\_no,Delay\_Arr[delay\_count]); //G on

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 12: delay(Delay\_Arr[delay\_count]); //G off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 13: LED\_no = 2;

blink(LED\_no,Delay\_Arr[delay\_count]); //B on

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 14: delay(Delay\_Arr[delay\_count]); //B off

delay\_count++;

**break**;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**case** 15: LED\_no = 2; //B on

blink(LED\_no,Delay\_Arr[delay\_count]);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 16: delay(Delay\_Arr[delay\_count]); //B off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 17: LED\_no = 2; //Blue on

blink(LED\_no,Delay\_Arr[delay\_count]);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 18: delay(Delay\_Arr[delay\_count]); //Blue off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

} ////end of switch

**if** (i==18)

{

i=1;

}

**if** (delay\_count == 19)

{

delay\_count = 0;

}

/\* 'Dummy' NOP to allow source level single stepping of

tight while() loop \*/

//\_\_asm volatile ("nop");

} //end of for

} ///end of mode 2

/////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////Freedom Board debug Mode//////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////

**if** (mode == 3)

{

**for** (i=1;i<=18;i++)

{

**if** (i == 20)

{

cumulative\_count++;

}

**if** ( cumulative\_count == 3)

{

**break**;

}

**switch** (i)

{

**case** 1: LED\_no = 0; // R on

print\_frb\_on(LED\_no, Delay\_Arr[delay\_count],cumulative\_delay);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 2: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 3: LED\_no = 0; //R on

print\_frb\_on(LED\_no,Delay\_Arr[delay\_count],cumulative\_delay);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 4: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 5: LED\_no = 0; //R on

print\_frb\_on(LED\_no,Delay\_Arr[delay\_count],cumulative\_delay);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 6: delay(Delay\_Arr[delay\_count]); //R off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 7: LED\_no = 1; //G on

print\_frb\_on(LED\_no,Delay\_Arr[delay\_count],cumulative\_delay);

;delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 8: delay(Delay\_Arr[delay\_count]);

delay\_count++; //G off

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 9: LED\_no = 1; //G on

print\_frb\_on(LED\_no,Delay\_Arr[delay\_count],cumulative\_delay);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 10: delay(Delay\_Arr[delay\_count]); //G off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 11: LED\_no = 1;

print\_frb\_on(LED\_no,Delay\_Arr[delay\_count],cumulative\_delay); //G on

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 12: delay(Delay\_Arr[delay\_count]); //G off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 13: LED\_no = 2;

print\_frb\_on(LED\_no,Delay\_Arr[delay\_count],cumulative\_delay); //B on

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 14: delay(Delay\_Arr[delay\_count]); //B off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 15: LED\_no = 2; //B on

print\_frb\_on(LED\_no,Delay\_Arr[delay\_count],cumulative\_delay);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 16: delay(Delay\_Arr[delay\_count]); //B off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 17: LED\_no = 2; //Blue on

print\_frb\_on(LED\_no,Delay\_Arr[delay\_count],cumulative\_delay);

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

**case** 18: delay(Delay\_Arr[delay\_count]); //Blue off

delay\_count++;

cumulative\_delay = cumulative\_delay + Delay\_Arr[delay\_count];

**break**;

} ////end of switch

**if** (i==18)

{

i=1;

}

**if** (delay\_count == 19)

{

delay\_count = 0;

}

/\* 'Dummy' NOP to allow source level single stepping of

tight while() loop \*/

//\_\_asm volatile ("nop");

} //end of for

} ///end of mode 3

/////////////////////////////////////////////////////////////////////////////////

**return** 0 ;

}