#ifndef DEBUG\_H

#define DEBUG\_H

#include <stdint.h>

#define MASK(x) (1UL << (x))

// Debug Signals on port B

#define DBG\_1 1

#define DBG\_2 2

#define DBG\_3 3

#define DBG\_4 8

#define DBG\_5 9

#define DBG\_6 10

#define DBG\_7 11

#define DBG\_LED\_ON DBG\_1

#define DBG\_CONTROLLER DBG\_2

#define DBG\_IRQ\_TPM DBG\_3

#define DBG\_IRQ\_ADC DBG\_4

//#define DBG\_ DBG\_5

//#define DBG\_ DBG\_6

//#define DBG\_ DBG\_7

void Init\_Debug\_Signals**(**void**);**

#endif // DEBUG\_H

#include <MKL25Z4.H>

#include "debug.h"

**void** Init\_Debug\_Signals**(void)** **{**

// Enable clock to port B

SIM**->**SCGC5 **|=** SIM\_SCGC5\_PORTB\_MASK**;**

// Make pins GPIO

PORTB**->**PCR**[**DBG\_1**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTB**->**PCR**[**DBG\_1**]** **|=** PORT\_PCR\_MUX**(**1**);**

PORTB**->**PCR**[**DBG\_2**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTB**->**PCR**[**DBG\_2**]** **|=** PORT\_PCR\_MUX**(**1**);**

PORTB**->**PCR**[**DBG\_3**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTB**->**PCR**[**DBG\_3**]** **|=** PORT\_PCR\_MUX**(**1**);**

PORTB**->**PCR**[**DBG\_4**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTB**->**PCR**[**DBG\_4**]** **|=** PORT\_PCR\_MUX**(**1**);**

PORTB**->**PCR**[**DBG\_5**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTB**->**PCR**[**DBG\_5**]** **|=** PORT\_PCR\_MUX**(**1**);**

PORTB**->**PCR**[**DBG\_6**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTB**->**PCR**[**DBG\_6**]** **|=** PORT\_PCR\_MUX**(**1**);**

PORTB**->**PCR**[**DBG\_7**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTB**->**PCR**[**DBG\_7**]** **|=** PORT\_PCR\_MUX**(**1**);**

// Set ports to outputs

PTB**->**PDDR **|=** MASK**(**DBG\_1**)** **|** MASK**(**DBG\_2**)** **|** MASK**(**DBG\_3**)** **|** MASK**(**DBG\_4**)** **|** MASK**(**DBG\_5**)** **|** MASK**(**DBG\_6**)** **|** MASK**(**DBG\_7**);**

// Initial values are 0

PTB**->**PCOR **=** MASK**(**DBG\_1**)** **|** MASK**(**DBG\_2**)** **|** MASK**(**DBG\_3**)** **|** MASK**(**DBG\_4**)** **|** MASK**(**DBG\_5**)** **|** MASK**(**DBG\_6**)** **|** MASK**(**DBG\_7**);**

**}**

#ifndef LEDS\_H

#define LEDS\_H

// Freedom KL25Z LEDs

#define RED\_LED\_POS (18) // on port B

#define GREEN\_LED\_POS (19) // on port B

#define BLUE\_LED\_POS (1) // on port D

// function prototypes

void Init\_RGB\_LEDs**(**void**);**

void Control\_RGB\_LEDs**(**unsigned int red\_on**,** unsigned int green\_on**,** unsigned int blue\_on**);**

#endif

#include <MKL25Z4.H>

#include "LEDs.h"

#include "gpio\_defs.h"

**void** Init\_RGB\_LEDs**(void)** **{**

// Enable clock to ports B and D

SIM**->**SCGC5 **|=** SIM\_SCGC5\_PORTB\_MASK **|** SIM\_SCGC5\_PORTD\_MASK**;;**

// Make 3 pins GPIO

PORTB**->**PCR**[**RED\_LED\_POS**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTB**->**PCR**[**RED\_LED\_POS**]** **|=** PORT\_PCR\_MUX**(**1**);**

PORTB**->**PCR**[**GREEN\_LED\_POS**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTB**->**PCR**[**GREEN\_LED\_POS**]** **|=** PORT\_PCR\_MUX**(**1**);**

PORTD**->**PCR**[**BLUE\_LED\_POS**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTD**->**PCR**[**BLUE\_LED\_POS**]** **|=** PORT\_PCR\_MUX**(**1**);**

// Set ports to outputs

PTB**->**PDDR **|=** MASK**(**RED\_LED\_POS**)** **|** MASK**(**GREEN\_LED\_POS**);**

PTD**->**PDDR **|=** MASK**(**BLUE\_LED\_POS**);**

**}**

**void** Control\_RGB\_LEDs**(unsigned** **int** red\_on**,** **unsigned** **int** green\_on**,** **unsigned** **int** blue\_on**)** **{**

**if** **(**red\_on**)** **{**

PTB**->**PCOR **=** MASK**(**RED\_LED\_POS**);**

**}** **else** **{**

PTB**->**PSOR **=** MASK**(**RED\_LED\_POS**);**

**}**

**if** **(**green\_on**)** **{**

PTB**->**PCOR **=** MASK**(**GREEN\_LED\_POS**);**

**}** **else** **{**

PTB**->**PSOR **=** MASK**(**GREEN\_LED\_POS**);**

**}**

**if** **(**blue\_on**)** **{**

PTD**->**PCOR **=** MASK**(**BLUE\_LED\_POS**);**

**}** **else** **{**

PTD**->**PSOR **=** MASK**(**BLUE\_LED\_POS**);**

**}**

**}**

#ifndef HBLED\_H

#define HBLED\_H

// Switching parameters

#define PWM\_HBLED\_CHANNEL (4)

#define PWM\_PERIOD (400)

/\* 48 MHz input clock.

PWM frequency = 48 MHz/(PWM\_PERIOD\*2)

Timer is in count-up/down mode. \*/

#define LIM\_DUTY\_CYCLE (PWM\_PERIOD)

// Control approach configuration

#define USE\_ASYNC\_SAMPLING 0

#define USE\_SYNC\_NO\_FREQ\_DIV 1

#define USE\_SYNC\_SW\_CTL\_FREQ\_DIV 0

#define USE\_SYNC\_HW\_CTL\_FREQ\_DIV 0

#define SW\_CTL\_FREQ\_DIV\_FACTOR (1) // Software division in ISR

#define HW\_CTL\_FREQ\_DIV\_CODE (0) // Not used

#if USE\_ASYNC\_SAMPLING

#define USE\_TPM0\_INTERRUPT 0

#define USE\_ADC\_HW\_TRIGGER 0

#define USE\_ADC\_INTERRUPT 1

#endif

#if USE\_SYNC\_NO\_FREQ\_DIV

#define USE\_TPM0\_INTERRUPT 0

#define USE\_ADC\_HW\_TRIGGER 1

#define USE\_ADC\_INTERRUPT 1

#endif

#if USE\_SYNC\_SW\_CTL\_FREQ\_DIV

#define USE\_TPM0\_INTERRUPT 1

#define USE\_ADC\_HW\_TRIGGER 0

#define USE\_ADC\_INTERRUPT 1

#endif

#if USE\_SYNC\_HW\_CTL\_FREQ\_DIV

#define USE\_TPM0\_INTERRUPT 0

#define USE\_ADC\_HW\_TRIGGER 1

#define USE\_ADC\_INTERRUPT 1

#endif

// Control Parameters

// default control mode: OpenLoop, BangBang, Incremental, PID, PID\_FX

#define DEF\_CONTROL\_MODE (Incremental)

// Incremental controller: change amount

#define INC\_STEP (PWM\_PERIOD/40)

// Proportional Gain, scaled by 2^8

#define PGAIN\_8 (0x0028)

// PID (floating-point) gains. Guaranteed to be non-optimal.

#define I\_GAIN\_FL (0.000f)

#define P\_GAIN\_FL (0.600f)

#define D\_GAIN\_FL (0.000f)

// PID\_FX (fixed-point) gains. Guaranteed to be non-optimal.

#define I\_GAIN\_FX (FL\_TO\_FX(0.0065f))

#define P\_GAIN\_FX (FL\_TO\_FX(0.0626f))

#define D\_GAIN\_FX (FL\_TO\_FX(0.1831f))

#define FLASH\_PERIOD (20)

#define FLASH\_CURRENT\_MA (180)

// Hardware configuration

#define ADC\_SENSE\_CHANNEL (8)

#define R\_SENSE (2.2f)

#define R\_SENSE\_MO ((int) (R\_SENSE\*1000))

#define V\_REF (3.3f)

#define V\_REF\_MV ((int) (V\_REF\*1000))

#define ADC\_FULL\_SCALE (0x10000)

#define MA\_SCALING\_FACTOR (1000)

#define DAC\_POS 30

#define DAC\_RESOLUTION 4096

// #define MA\_TO\_DAC\_CODE(i) (i\*2.2\*DAC\_RESOLUTION/V\_REF\_MV) // Introduces timing delay and interesting bug!

#define MA\_TO\_DAC\_CODE(i) ((i)\*(2.2f\*DAC\_RESOLUTION/V\_REF\_MV))

#define MIN(a,b) ((a<b)?a:b)

#define MAX(a,b) ((a>b)?a:b)

#endif // HBLED\_H

#ifndef DELAY\_H

#define DELAY\_H

#include <stdint.h>

extern void Delay**(**uint32\_t dlyTicks**);**

extern void ShortDelay**(**uint32\_t dlyTicks**);**

#endif

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ARM University Program Copyright © ARM Ltd 2013\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <MKL25Z4.H>

**void** Delay **(uint32\_t** dly**)** **{**

**volatile** **uint32\_t** t**;**

**for** **(**t**=**dly**\***10000**;** t**>**0**;** t**--)**

**;**

**}**

**void** ShortDelay **(uint32\_t** dly**)** **{**

**volatile** **uint32\_t** t**;**

**for** **(**t**=**dly**;** t**>**0**;** t**--)**

**;**

**}**

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ARM University Program Copyright © ARM Ltd 2013\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*----------------------------------------------------------------------------

\*----------------------------------------------------------------------------\*/

#include <MKL25Z4.H>

#include <stdio.h>

#include <stdint.h>

#include "gpio\_defs.h"

#include "debug.h"

#include "timers.h"

#include "delay.h"

#include "LEDS.h"

#include "HBLED.h"

#include "FX.h"

**volatile** **int** g\_enable\_control**=**1**;**

**volatile** **int** g\_set\_current**=**0**;**

**volatile** **int** measured\_current**;**

**volatile** **int16\_t** g\_duty\_cycle**=**0**;** // global to give debugger access

**volatile** **int** error**;**

**enum** **{**OpenLoop**,** BangBang**,** Incremental**,** Proportional**,** PID**,** PID\_FX**}**

control\_mode**=**DEF\_CONTROL\_MODE**;**

**int32\_t** pGain\_8 **=** PGAIN\_8**;** // proportional gain numerator scaled by 2^8

**volatile** **int** g\_enable\_flash**=**1**;**

**typedef** **struct** **{**

**float** dState**;** // Last position input

**float** iState**;** // Integrator state

**float** iMax**,** iMin**;** // Maximum and minimum allowable integrator state

**float** iGain**,** // integral gain

pGain**,** // proportional gain

dGain**;** // derivative gain

**}** SPid**;**

**typedef** **struct** **{**

FX16\_16 dState**;** // Last position input

FX16\_16 iState**;** // Integrator state

FX16\_16 iMax**,** iMin**;** // Maximum and minimum allowable integrator state

FX16\_16 iGain**,** // integral gain

pGain**,** // proportional gain

dGain**;** // derivative gain

**}** SPidFX**;**

SPid plantPID **=** **{**0**,** // dState

0**,** // iState

LIM\_DUTY\_CYCLE**,** // iMax

**-**LIM\_DUTY\_CYCLE**,** // iMin

I\_GAIN\_FL**,** // iGain

P\_GAIN\_FL**,** // pGain

D\_GAIN\_FL // dGain

**};**

SPidFX plantPID\_FX **=** **{**FL\_TO\_FX**(**0**),** // dState

FL\_TO\_FX**(**0**),** // iState

FL\_TO\_FX**(**LIM\_DUTY\_CYCLE**),** // iMax

FL\_TO\_FX**(-**LIM\_DUTY\_CYCLE**),** // iMin

I\_GAIN\_FX**,** // iGain

P\_GAIN\_FX**,** // pGain

D\_GAIN\_FX // dGain

**};**

**float** UpdatePID**(**SPid **\*** pid**,** **float** error**,** **float** position**){**

**float** pTerm**,** dTerm**,** iTerm**;**

// calculate the proportional term

pTerm **=** pid**->**pGain **\*** error**;**

// calculate the integral state with appropriate limiting

pid**->**iState **+=** error**;**

**if** **(**pid**->**iState **>** pid**->**iMax**)**

pid**->**iState **=** pid**->**iMax**;**

**else** **if** **(**pid**->**iState **<** pid**->**iMin**)**

pid**->**iState **=** pid**->**iMin**;**

iTerm **=** pid**->**iGain **\*** pid**->**iState**;** // calculate the integral term

dTerm **=** pid**->**dGain **\*** **(**position **-** pid**->**dState**);**

pid**->**dState **=** position**;**

**return** pTerm **+** iTerm **-** dTerm**;**

**}**

FX16\_16 UpdatePID\_FX**(**SPidFX **\*** pid**,** FX16\_16 error\_FX**,** FX16\_16 position\_FX**){**

FX16\_16 pTerm**,** dTerm**,** iTerm**,** diff**,** ret\_val**;**

// calculate the proportional term

pTerm **=** Multiply\_FX**(**pid**->**pGain**,** error\_FX**);**

// calculate the integral state with appropriate limiting

pid**->**iState **=** Add\_FX**(**pid**->**iState**,** error\_FX**);**

**if** **(**pid**->**iState **>** pid**->**iMax**)**

pid**->**iState **=** pid**->**iMax**;**

**else** **if** **(**pid**->**iState **<** pid**->**iMin**)**

pid**->**iState **=** pid**->**iMin**;**

iTerm **=** Multiply\_FX**(**pid**->**iGain**,** pid**->**iState**);** // calculate the integral term

diff **=** Subtract\_FX**(**position\_FX**,** pid**->**dState**);**

dTerm **=** Multiply\_FX**(**pid**->**dGain**,** diff**);**

pid**->**dState **=** position\_FX**;**

ret\_val **=** Add\_FX**(**pTerm**,** iTerm**);**

ret\_val **=** Subtract\_FX**(**ret\_val**,** dTerm**);**

**return** ret\_val**;**

**}**

**void** Control\_HBLED**(void)** **{**

**uint16\_t** res**;**

FX16\_16 change\_FX**,** error\_FX**;**

FPTB**->**PSOR **=** MASK**(**DBG\_CONTROLLER**);**

#if USE\_ADC\_INTERRUPT

// already completed conversion, so don't wait

#else

**while** **(!(**ADC0**->**SC1**[**0**]** **&** ADC\_SC1\_COCO\_MASK**))**

**;** // wait until end of conversion

#endif

res **=** ADC0**->**R**[**0**];**

measured\_current **=** **(**res**\***1500**)>>**16**;** // Extra Credit: Make this code work: V\_REF\_MV\*MA\_SCALING\_FACTOR)/(ADC\_FULL\_SCALE\*R\_SENSE)

**switch** **(**control\_mode**)** **{**

**case** OpenLoop**:**

// don't do anything!

**break;**

**case** BangBang**:**

**if** **(**measured\_current **<** g\_set\_current**)**

g\_duty\_cycle **=** LIM\_DUTY\_CYCLE**;**

**else**

g\_duty\_cycle **=** 0**;**

**break;**

**case** Incremental**:**

**if** **(**measured\_current **<** g\_set\_current**)**

g\_duty\_cycle **+=** INC\_STEP**;**

**else**

g\_duty\_cycle **-=** INC\_STEP**;**

**break;**

**case** Proportional**:**

g\_duty\_cycle **+=** **(**pGain\_8**\*(**g\_set\_current **-** measured\_current**))/**256**;** // - 1;

**break;**

**case** PID**:**

g\_duty\_cycle **+=** UpdatePID**(&**plantPID**,** g\_set\_current **-** measured\_current**,** measured\_current**);**

**break;**

**case** PID\_FX**:**

error\_FX **=** INT\_TO\_FX**(**g\_set\_current **-** measured\_current**);**

change\_FX **=** UpdatePID\_FX**(&**plantPID\_FX**,** error\_FX**,** INT\_TO\_FX**(**measured\_current**));**

g\_duty\_cycle **+=** FX\_TO\_INT**(**change\_FX**);**

**break;**

**default:**

**break;**

**}**

// Update PWM controller with duty cycle

**if** **(**g\_duty\_cycle **<** 0**)**

g\_duty\_cycle **=** 0**;**

**else** **if** **(**g\_duty\_cycle **>** LIM\_DUTY\_CYCLE**)**

g\_duty\_cycle **=** LIM\_DUTY\_CYCLE**;**

PWM\_Set\_Value**(**TPM0**,** PWM\_HBLED\_CHANNEL**,** g\_duty\_cycle**);**

FPTB**->**PCOR **=** MASK**(**DBG\_CONTROLLER**);**

**}**

#if USE\_ADC\_INTERRUPT

**void** ADC0\_IRQHandler**()** **{**

FPTB**->**PSOR **=** MASK**(**DBG\_IRQ\_ADC**);**

Control\_HBLED**();**

FPTB**->**PCOR **=** MASK**(**DBG\_IRQ\_ADC**);**

**}**

#endif

**void** Set\_DAC**(unsigned** **int** code**)** **{**

// Force 16-bit write to DAC

**uint16\_t** **\*** dac0dat **=** **(uint16\_t** **\*)&(**DAC0**->**DAT**[**0**].**DATL**);**

**\***dac0dat **=** **(uint16\_t)** code**;**

**}**

**void** Set\_DAC\_mA**(unsigned** **int** current**)** **{**

**unsigned** **int** code **=** MA\_TO\_DAC\_CODE**(**current**);**

// Force 16-bit write to DAC

**uint16\_t** **\*** dac0dat **=** **(uint16\_t** **\*)&(**DAC0**->**DAT**[**0**].**DATL**);**

**\***dac0dat **=** **(uint16\_t)** code**;**

**}**

**void** Init\_DAC**(void)** **{**

// Enable clock to DAC and Port E

SIM**->**SCGC6 **|=** SIM\_SCGC6\_DAC0\_MASK**;**

SIM**->**SCGC5 **|=** SIM\_SCGC5\_PORTE\_MASK**;**

// Select analog for pin

PORTE**->**PCR**[**DAC\_POS**]** **&=** **~**PORT\_PCR\_MUX\_MASK**;**

PORTE**->**PCR**[**DAC\_POS**]** **|=** PORT\_PCR\_MUX**(**0**);**

// Disable buffer mode

DAC0**->**C1 **=** 0**;**

DAC0**->**C2 **=** 0**;**

// Enable DAC, select VDDA as reference voltage

DAC0**->**C0 **=** DAC\_C0\_DACEN\_MASK **|** DAC\_C0\_DACRFS\_MASK**;**

Set\_DAC**(**0**);**

**}**

**void** Init\_ADC**(void)** **{**

// Configure ADC to read Ch 8 (FPTB 0)

SIM**->**SCGC6 **|=** SIM\_SCGC6\_ADC0\_MASK**;**

ADC0**->**CFG1 **=** 0x0C**;** // 16 bit

// ADC0->CFG2 = ADC\_CFG2\_ADLSTS(3);

ADC0**->**SC2 **=** ADC\_SC2\_REFSEL**(**0**);**

#if USE\_ADC\_HW\_TRIGGER

// Enable hardware triggering of ADC

ADC0**->**SC2 **|=** ADC\_SC2\_ADTRG**(**1**);**

// Select triggering by TPM0 Overflow

SIM**->**SOPT7 **=** SIM\_SOPT7\_ADC0TRGSEL**(**8**)** **|** SIM\_SOPT7\_ADC0ALTTRGEN\_MASK**;**

// Select input channel

ADC0**->**SC1**[**0**]** **&=** **~**ADC\_SC1\_ADCH\_MASK**;**

ADC0**->**SC1**[**0**]** **|=** ADC\_SC1\_ADCH**(**ADC\_SENSE\_CHANNEL**);**

#endif

#if USE\_ADC\_INTERRUPT

// enable ADC interrupt

ADC0**->**SC1**[**0**]** **|=** ADC\_SC1\_AIEN**(**1**);**

// Configure NVIC for ADC interrupt

NVIC\_SetPriority**(**ADC0\_IRQn**,** 128**);** // 0, 64, 128 or 192

NVIC\_ClearPendingIRQ**(**ADC0\_IRQn**);**

NVIC\_EnableIRQ**(**ADC0\_IRQn**);**

#endif

**}**

**void** Update\_Set\_Current**(void)** **{**

**static** **int** delay**=**FLASH\_PERIOD**;**

**if** **(**g\_enable\_flash**){**

delay**--;**

**if** **(**delay**==**1**)** **{**

FPTB**->**PSOR **=** MASK**(**DBG\_LED\_ON**);**

Set\_DAC\_mA**(**FLASH\_CURRENT\_MA**);**

g\_set\_current **=** FLASH\_CURRENT\_MA**;**

**}** **else** **if** **(**delay**==**0**)** **{**

delay**=**FLASH\_PERIOD**;**

FPTB**->**PSOR **=** MASK**(**DBG\_LED\_ON**);**

Set\_DAC\_mA**(**FLASH\_CURRENT\_MA**/**4**);**

g\_set\_current **=** FLASH\_CURRENT\_MA**/**4**;**

**}** **else** **{**

FPTB**->**PCOR **=** MASK**(**DBG\_LED\_ON**);**

Set\_DAC\_mA**(**0**);**

g\_set\_current **=** 0**;**

**}**

**}**

**}**

/\*----------------------------------------------------------------------------

MAIN function

\*----------------------------------------------------------------------------\*/

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

Init\_Debug\_Signals**();**

Init\_DAC**();**

Init\_ADC**();**

Init\_RGB\_LEDs**();**

// Configure driver for buck converter

// Set up PTE31 to use for SMPS with TPM0 Ch 4

SIM**->**SCGC5 **|=** SIM\_SCGC5\_PORTE\_MASK**;**

PORTE**->**PCR**[**31**]** **&=** PORT\_PCR\_MUX**(**7**);**

PORTE**->**PCR**[**31**]** **|=** PORT\_PCR\_MUX**(**3**);**

PWM\_Init**(**TPM0**,** PWM\_HBLED\_CHANNEL**,** PWM\_PERIOD**,** 40**);**

Control\_RGB\_LEDs**(**1**,**1**,**0**);**

Delay**(**100**);**

Control\_RGB\_LEDs**(**0**,**0**,**1**);**

// Configure flash timer

Init\_PIT**(**423456**);**

Start\_PIT**();**

**while** **(**1**)** **{**

#if USE\_ASYNC\_SAMPLING

// Start conversion

ADC0**->**SC1**[**0**]** **=** ADC\_SC1\_AIEN**(**1**)** **|** ADC\_SENSE\_CHANNEL**;**

Control\_HBLED**();** // Blocks until ADC done, then updates control

#endif

// else do nothing but wait for interrupts

**}**

**}**

#ifndef TIMERS\_H

#define TIMERS\_H

#include "MKL25Z4.h"

void PWM\_Init**(**TPM\_Type **\*** TPM**,** uint8\_t channel\_num**,** uint16\_t period**,** uint16\_t duty**);**

void PWM\_Set\_Value**(**TPM\_Type **\*** TPM**,** uint8\_t channel\_num**,** uint16\_t value**);**

void Init\_PIT**(**unsigned period**);**

void Start\_PIT**(**void**);**

void Stop\_PIT**(**void**);**

#endif

#include "timers.h"

#include <MKL25Z4.h>

#include "HBLED.h"

#include "GPIO\_defs.h"

#include "debug.h"

**extern** **void** Update\_Set\_Current**(void);**

**void** PWM\_Init**(**TPM\_Type **\*** TPM**,** **uint8\_t** channel\_num**,** **uint16\_t** period**,** **uint16\_t** duty**)**

**{**

//turn on clock to TPM

**switch** **((int)** TPM**)** **{**

**case** **(int)** TPM0**:**

SIM**->**SCGC6 **|=** SIM\_SCGC6\_TPM0\_MASK**;**

**break;**

**case** **(int)** TPM1**:**

SIM**->**SCGC6 **|=** SIM\_SCGC6\_TPM1\_MASK**;**

**break;**

**case** **(int)** TPM2**:**

SIM**->**SCGC6 **|=** SIM\_SCGC6\_TPM2\_MASK**;**

**break;**

**default:**

**break;**

**}**

//set clock source for tpm

SIM**->**SOPT2 **|=** **(**SIM\_SOPT2\_TPMSRC**(**1**)** **|** SIM\_SOPT2\_PLLFLLSEL\_MASK**);**

//load the counter and mod

TPM**->**MOD **=** period**;**

//set channel to center-aligned low-true PWM

TPM**->**CONTROLS**[**channel\_num**].**CnSC **=** TPM\_CnSC\_MSB\_MASK **|** TPM\_CnSC\_ELSA\_MASK**;**

//set TPM to up-down and divide by 1 prescaler and clock mode

TPM**->**SC **=** **(**TPM\_SC\_CPWMS\_MASK **|** TPM\_SC\_PS**(**0**));**

//set trigger mode and keep running in debug mode

TPM**->**CONF **|=** TPM\_CONF\_TRGSEL**(**0xA**)** **|** TPM\_CONF\_DBGMODE**(**3**);**

// Set initial duty cycle

TPM**->**CONTROLS**[**channel\_num**].**CnV **=** duty**;**

#if USE\_TPM0\_INTERRUPT // if using interrupt

TPM0**->**SC **|=** TPM\_SC\_TOIE\_MASK**;**

// Configure NVIC

NVIC\_SetPriority**(**TPM0\_IRQn**,** 128**);** // 0, 64, 128 or 192

NVIC\_ClearPendingIRQ**(**TPM0\_IRQn**);**

NVIC\_EnableIRQ**(**TPM0\_IRQn**);**

#endif

// Start the timer counting

TPM**->**SC **|=** TPM\_SC\_CMOD**(**1**);**

**}**

**void** PWM\_Set\_Value**(**TPM\_Type **\*** TPM**,** **uint8\_t** channel\_num**,** **uint16\_t** value**)** **{**

TPM**->**CONTROLS**[**channel\_num**].**CnV **=** value**;**

**}**

**extern** **void** Control\_HBLED**(void);**

**void** TPM0\_IRQHandler**()** **{**

**static** **uint32\_t** control\_divider **=** SW\_CTL\_FREQ\_DIV\_FACTOR**;**

FPTB**->**PSOR **=** MASK**(**DBG\_IRQ\_TPM**);**

//clear pending IRQ flag

TPM0**->**SC **|=** TPM\_SC\_TOF\_MASK**;**

control\_divider**--;**

**if** **(**control\_divider **==** 0**)** **{**

control\_divider **=** SW\_CTL\_FREQ\_DIV\_FACTOR**;**

// Start conversion

ADC0**->**SC1**[**0**]** **=** ADC\_SC1\_AIEN**(**1**)** **|** ADC\_SENSE\_CHANNEL**;**

#if USE\_ADC\_INTERRUPT

// can return immediately

#else

// Call control function, which will wait for ADC coco

Control\_HBLED**();**

#endif

**}**

FPTB**->**PCOR **=** MASK**(**DBG\_IRQ\_TPM**);**

**}**

**void** Init\_PIT**(unsigned** period**)** **{**

// Enable clock to PIT module

SIM**->**SCGC6 **|=** SIM\_SCGC6\_PIT\_MASK**;**

// Disable clocks for configuration,

PIT**->**MCR **|=** PIT\_MCR\_MDIS\_MASK**;**

// Initialize PIT0 to count down from argument

PIT**->**CHANNEL**[**0**].**LDVAL **=** PIT\_LDVAL\_TSV**(**period**);**

// No chaining

PIT**->**CHANNEL**[**0**].**TCTRL **&=** **~**PIT\_TCTRL\_CHN\_MASK**;**

PIT**->**CHANNEL**[**0**].**TCTRL **|=** PIT\_TCTRL\_TIE\_MASK**;**

#if 1 // generate interrupts

NVIC\_SetPriority**(**PIT\_IRQn**,** 128**);** // 0, 64, 128 or 192

NVIC\_ClearPendingIRQ**(**PIT\_IRQn**);**

NVIC\_EnableIRQ**(**PIT\_IRQn**);**

#endif

**}**

**void** Start\_PIT**(void)** **{**

// Enable counter

PIT**->**CHANNEL**[**0**].**TCTRL **|=** PIT\_TCTRL\_TEN\_MASK**;**

PIT**->**MCR **&=** **~**PIT\_MCR\_MDIS\_MASK**;**

**}**

**void** Stop\_PIT**(void)** **{**

// Disable counter

PIT**->**CHANNEL**[**0**].**TCTRL **&=** **~**PIT\_TCTRL\_TEN\_MASK**;**

**}**

**void** PIT\_IRQHandler**()** **{**

// check to see which channel triggered interrupt

**if** **(**PIT**->**CHANNEL**[**0**].**TFLG **&** PIT\_TFLG\_TIF\_MASK**)** **{**

// clear status flag for timer channel 0

PIT**->**CHANNEL**[**0**].**TFLG **&=** PIT\_TFLG\_TIF\_MASK**;**

// Do ISR work

Update\_Set\_Current**();**

**}** **else** **if** **(**PIT**->**CHANNEL**[**1**].**TFLG **&** PIT\_TFLG\_TIF\_MASK**)** **{**

// clear status flag for timer channel 1

PIT**->**CHANNEL**[**1**].**TFLG **&=** PIT\_TFLG\_TIF\_MASK**;**

**}**

**}**

#ifndef FX\_H

#define FX\_H

#include <stdint.h>

**typedef** int32\_t FX16\_16**;**

#define FL\_TO\_FX(x) ((FX16\_16)((x)\*65536.0))

#define INT\_TO\_FX(x) ((FX16\_16)((x)\*65536))

#define FX\_TO\_INT(x) ((int32\_t)((x)/65536))

#define FX\_TO\_FL(x) ((float)((x)/65536.0))

FX16\_16 Multiply\_FX**(**FX16\_16 a**,** FX16\_16 b**);**

FX16\_16 Add\_FX**(**FX16\_16 a**,** FX16\_16 b**);**

FX16\_16 Subtract\_FX**(**FX16\_16 a**,** FX16\_16 b**);**

#endif // FX\_H

#include "FX.h"

FX16\_16 Multiply\_FX**(**FX16\_16 a**,** FX16\_16 b**)** **{**

**int64\_t** p**,** pa**,** pb**;**

// Long multiply first.

pa **=** a**;**

pb **=** b**;**

p **=** pa **\*** pb**;**

// normalize after multiplication

p **>>=** 16**;**

**return** **(**FX16\_16**)(**p**&**0xffffffff**);**

**}**

FX16\_16 Add\_FX**(**FX16\_16 a**,** FX16\_16 b**)** **{**

FX16\_16 p**;**

// Add. This will overflow if a+b > 2^16

p **=** a **+** b**;**

**return** p**;**

**}**

FX16\_16 Subtract\_FX**(**FX16\_16 a**,** FX16\_16 b**)** **{**

FX16\_16 p**;**

p **=** a **-** b**;**

**return** p**;**

**}**