



UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA MECÂNICA

ES670 - Projeto de Sistemas Embarcados

Relatório - Projeto Prático Parte 1 Requisitos de Teclado, LEDs e Display de Sete Segmentos

Nome:

Daniel Dello Russo Oliveira

Davi Rodrigues

RA

101918

116581

15 de abril de 2016

1 Objetivo

O objetivo do projeto é, de maneira incremental, implementar no *target* os requisitos apresentados no roteiro[1] inicialmente desenvolvendo o modelo e depois implementando cada requisito. Estes requisitos são referentes à configuração e implementação de entradas de teclado, acionamento de LEDs, *display* de sete segmentos, protocolo de comunicação, *display LCD*, medição de velocidade de rotação, *PWM*, *ADC* e Controlador.

2 Modelagem

Utilizando o Rational Rhapsody Modeler e tomando como base os requisitos propostos mostrados na figura 1, complementamos o modelo inicial[2] (requisitos de teclado e LEDs) adicionando um bloco ao modelo referente aos *displays* de sete segmentos (REQ1C), conforme mostrado na figura 3. Adicionamos também alguns blocos auxiliares relacionados ao gerenciamento de pinos *GPIO* e a interrupções periódicas, que foram utilizados para nossa implementação do *display* de sete segmentos e do *buzzer*. Ao tratar o gerenciamento do *display* e do *buzzer* através de interrupções, livramos a *thread* principal para que essa lide com outros problemas sem precisar se preocupar com a atualização periódica dos *displays*.


 ES670 - Projeto de Sistemas Embarcados Projeto Prático - 1o semestre/2016 Diagrama de Requisitos de Sistema		
«Requirement» Requisito TECLADO ID = REQ1A O sistema deve permitir o monitoramento de 4 chaves tipo "push button".	«Requirement» Requisito LED ID = REQ1B O sistema deve ser capaz de acionar 4 LEDs.	«Requirement» Requisito DISPLAY7SEG ID = REQ1C O sistema deve ser capaz de acionar 4 displays de 7 segmentos.
«Requirement» Requisito PROTOCOLO ID = REQ2 O sistema deve possuir um protocolo de comunicação serial capaz de receber comandos e retornar status dos periféricos do sistema.	«Requirement» Requisito LCD ID = REQ3 O sistema deve ser capaz de exibir mensagens alfanuméricas num LCD.	«Requirement» Requisito VELOCIDADE ID = REQ4 O sistema deve ser capaz de realizar medições de velocidade de rotação.
«Requirement» Requisito PWM ID = REQ5 O sistema deve ser capaz de gerar um sinal de PWM.	«Requirement» Requisito ADC ID = REQ6 O sistema deve ser capaz de realizar a leitura de sinal analógico e também a conversão A/D deste sinal.	«Requirement» Requisito CONTROLE ID = REQ7 O sistema deve ser capaz de realizar a funcionalidade de controlador digital.

Figura 1: Diagrama de requisitos

 **ES670 - Projeto de Sistemas Embarcados**
Projeto Prático - 1o semestre/2016

Diagrama de Pacotes

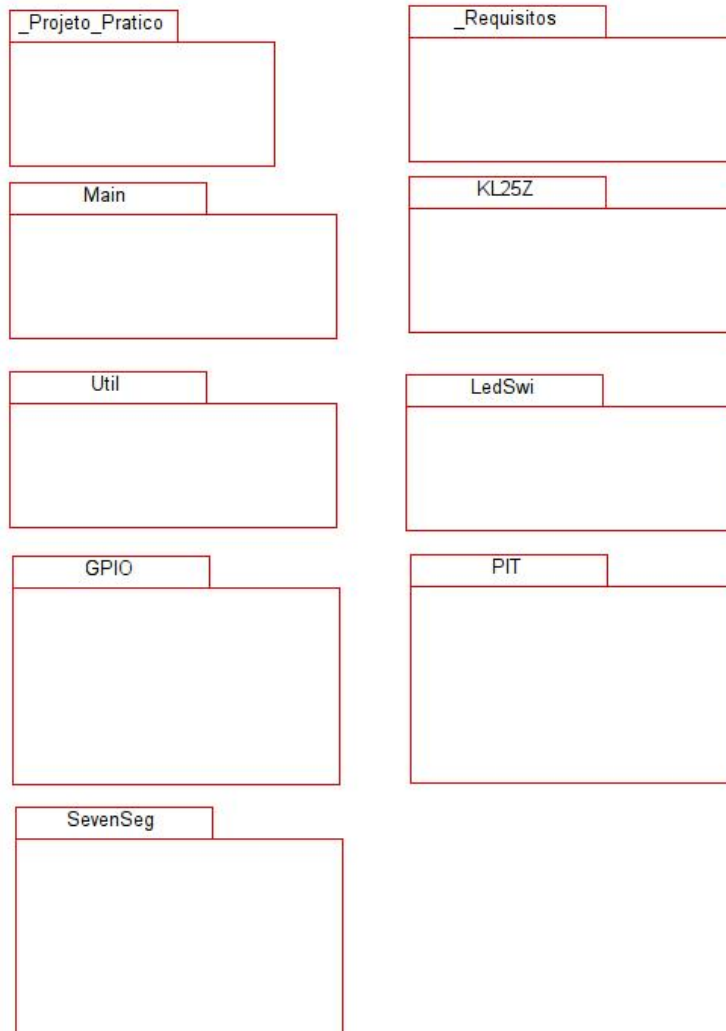


Figura 2: Diagrama de pacotes

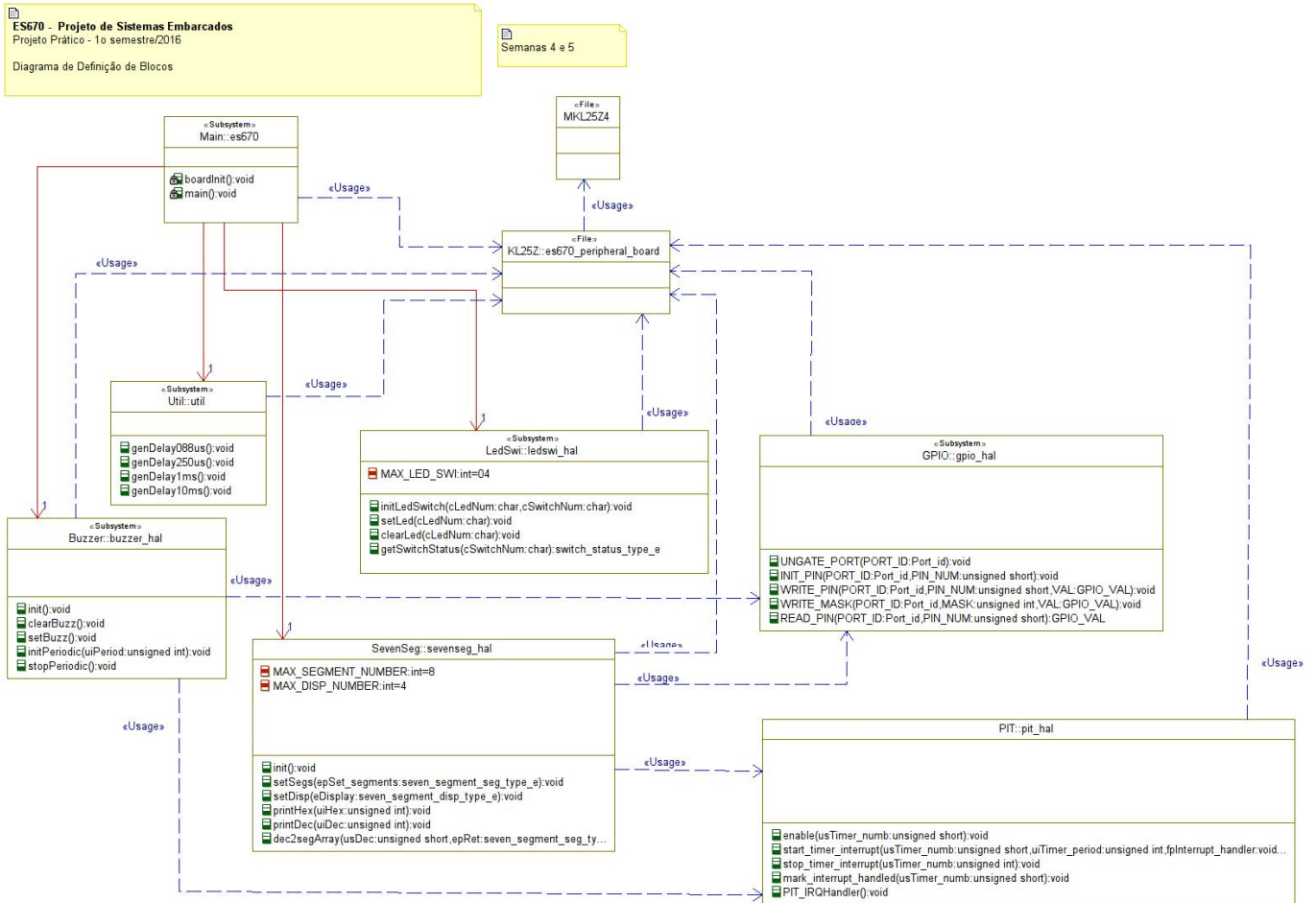


Figura 3: Diagrama de definição de blocos

O bloco de *GPIO_hal* tem operações para desbloquear o *clock* para uma porta, inicializar um pino em uma dada direção, escrever em um pino, escrever em um conjunto de pinos da mesma porta e ler a entrada em um pino.

O bloco *pit_hal* tem operações para inicializar o *PIT*, criar interrupções periódicas em um dos dois *timers* disponíveis, desativar as interrupções em um *timer* e marcar uma interrupção como tratada (deve ser feito pelos tratamentos de interrupção). Além disso esse bloco tem mais uma operação chamada *PIT_IRQHandler* que trata as interrupções do *PIT*, essa operação precisa ser visível para o *linker*, mas não deve ser chamada pelo usuário. É importante ressaltar que o *clock* do *PIT* é o *bus_clock* que no nosso caso é de 20MHz

As operações do bloco *sevenseg_hal* cobrem a inicialização dos *displays*, seleção manual de quais segmentos estarão ativos (feita através da passagem de um vetor com os segmentos desejados), ativação manual de um dos *displays* (desativando todos os outros), conversão de dígito hexadecimal ou decimal em vetor de segmentos (para ser passado para a função de seleção de segmentos) e impressão automática de um valor hexadecimal ou decimal através das interrupções de *timer*.

O bloco do *buzzer_hal* também ganhou duas novas operações, uma para criar uma onda quadrada no *buzzer* com um certo período (e *duty cycle* de 50%), através de interrupções de *timer*, e outra para remover essa onda.

3 Diagramas Esquemáticos

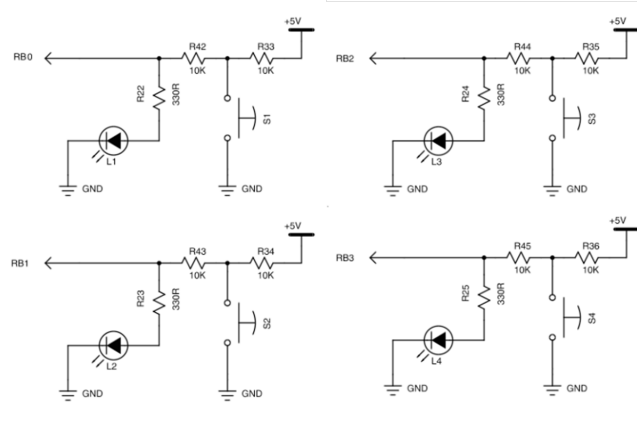


Figura 4: Esquema teclado e LEDs

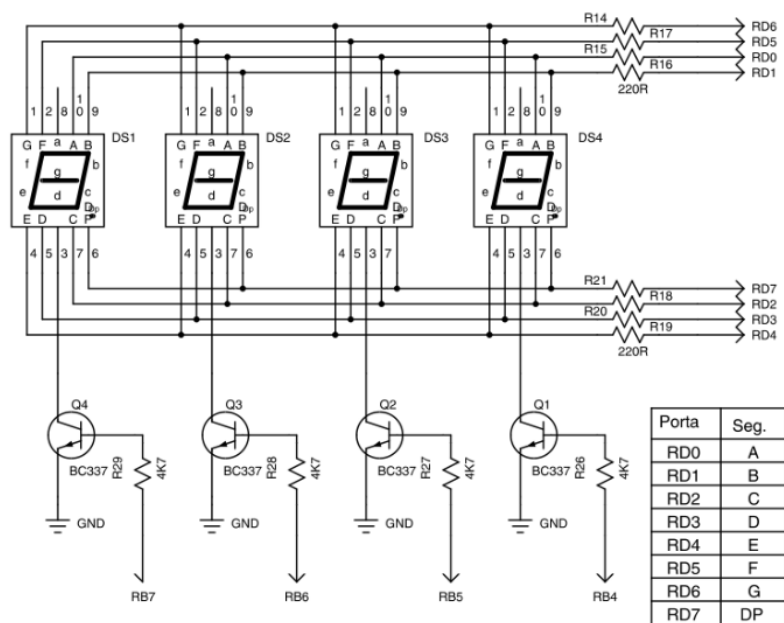


Figura 5: Esquema sete segmentos

Os pinos apresentados acima, devem ser mapeados para os da placa *FRDM-KL25Z* através do mapeamento apresentado na figura 6

KL25Z ADAPTER Marclio

FUNCAIONALIDADE	ESPECIFICAÇÃO	PINO	TIPO	PORTA DO ARM	HEADER ARM	SOQUETE PIC
Display 7S	Segmento A	RD0	GPIO	PTC0	PTC0	J1 03
Display 7S	Segmento B	RD1	GPIO	PTC1	PTC1LLWU P6RTC CLKIN	J10 12
Display 7S	Segmento C	RD2	GPIO	PTC2	PTC2	J10 10
Display 7S	Segmento D	RD3	GPIO	PTC3	PTC3LLWU P7	J1 05
Display 7S	Segmento E	RD4	GPIO	PTC4	PTC4LLWU P8	J1 07
Display 7S	Segmento F	RD5	GPIO	PTC5	PTC5LLWU P9	J1 09
Display 7S	Segmento G	RD6	GPIO	PTC6	PTC6LLWU P10	J1 11
Display 7S	Segmento DP	RD7	GPIO	PTC7	PTC7	J1 01
Display 7S	Display 1	RB7	GPIO	PTC13	PTC13	J2 03
Display 7S	Display 2	RB6	GPIO	PTC12	PTC12	J2 01
Display 7S	Display 3	RB5	GPIO	PTC11	PTC11	J1 15
Display 7S	Display 4	RB4	GPIO	PTC10	PTC10	J1 13
LCD	RS	RE0	GPIO	PTC8	PTC8	J1 14
LCD	ENABLE	RE1	GPIO	PTC9	PTC9	J1 16
LCD	DB0	RD0	GPIO	PTC0	PTC0	J1 03
LCD	DB1	RD1	GPIO	PTC1	PTC1LLWU P6RTC CLKIN	J10 12
LCD	DB2	RD2	GPIO	PTC2	PTC2	J10 10
LCD	DB3	RD3	GPIO	PTC3	PTC3LLWU P7	J1 05
LCD	DB4	RD4	GPIO	PTC4	PTC4LLWU P8	J1 07
LCD	DB5	RD5	GPIO	PTC5	PTC5LLWU P9	J1 09
LCD	DB6	RD6	GPIO	PTC6	PTC6LLWU P10	J1 11
LCD	DB7	RD7	GPIO	PTC7	PTC7	J1 01
BUZZER		RA5	GPIO	PTD0	FTM0_CH0	J2 06
Teclado/LED	S1/LED1	RB0	GPIO	PTA1	PTA1	J1 02
Teclado/LED	S2/LED2	RB1	GPIO	PTA2	PTA2	J1 04
Teclado/LED	S3/LED3	RB2	GPIO	PTA4	PTA4	J1 10
Teclado/LED	S4/LED4	RB3	GPIO	PTA5	PTA5	J1 12

Página 1

Figura 6: Mapeamento entre pinos dos esquemáticos e da placa FRDM-KL25Z

Como pode ser visto na figura 5, é necessário fazer um gerenciamento dos pinos PTC0 a PTC7 para selecionar os segmentos que serão ativados e PTC10 a PTC13 para selecionar quais *displays* estarão ativos. Para isso, é preciso alternar qual *display* está ativo e fazer a mudança nos segmentos para que cada *display* esteja mostrando um valor diferente. É importante lembrar que a frequência dessa alternância seja escolhida de modo que o olho humano não perceba que os *displays* estão ligando e desligando. Afim de garantir que essa alternância funcionará apropriadamente nós utilizamos o módulo *PIT* para gerar uma interrupção a cada $3.125ms$, conforme sugerido na aula 6 [4].

4 Matriz de Rastreabilidade

A matriz de rastreabilidade apresentada na tabela 1 relaciona cada um dos requisitos com a sua implementação.

Tabela 1: Matriz de Rastreabilidade

ID do Requisito	Implementação
REQ1A	<code>ledswi_hal.c</code> - <code>void ledswi_initLedSwitch(char cLedNum, char cSwitchNum)</code> - <code>switch_status_type_e ledswi_getSwitchStatus(char cSwitchNum)</code>
REQ1B	<code>ledswi.c</code> - <code>void ledswi_initLedSwitch(char cLedNum, char cSwitchNum)</code> - <code>void ledswi_setLed(char cLedNum)</code> - <code>void ledswi_clearLed(char cLedNum)</code>
REQ1C	<code>sevenSeg.c</code> - <code>void sevenseg_init(void)</code> - <code>void sevenseg_setSegs(seven_segment_seg_type_e* epSet_segments)</code> - <code>void sevenseg_setDisp(seven_segment_disp_type_e eDisplay)</code> - <code>void sevenseg_printDec(unsigned int uiDec)</code> - <code>void sevenseg_printHex(unsigned int uiHex)</code>

5 Notas

5.1 Gerenciamento de GPIO e macros

Detectamos logo no início do projeto um defeito estrutural no código fornecido quando lidando com GPIO: o identificador da porta e o número do pino utilizado eram referenciados em diversos locais diferentes do código dificultando de maneira agravante mudanças na configuração de hardware. Para resolver isso inicialmente pensamos em utilizar o arquivo *fsl_gpio_hal.h* da biblioteca da *FRDM-K125Z*, mas isso não nos foi permitido. Como calcular as posições na memória de cada registrador seria reimplementar a biblioteca, escolhemos por criar macros que geram o mesmo estilo de código utilizado no exemplo fornecido através do operador de concatenação (`##`) do pré processador. Esse operador apresenta algumas particularidades, a principal sendo que macros que o utilizam em seu corpo não tem seus argumentos expandidos [5]. Para circular essa dificuldade criamos uma outras macros que funcionam como uma *wrappers* para essas macros, fazendo assim que seus argumentos sejam expandidos antes da chamada da concatenação.

As macros que fazem a concatenação propriamente ditas não devem ser chamadas pelo usuário (sendo identificadas por um `_` no início de seus nomes).

Outra dificuldade relacionada a esse módulo é que a expansão dos argumentos das macros não para quando chega em alguma *token* não definida, no caso o identificador das portas (A,B,C,D,E). Para contornar esse problema utilizamos *typedefs* para definir esses identificadores como *tokens* válidas.

5.2 Interrupções por timer

Outro problema que enfrentamos foi com a implementação das interrupções por timer através do PIT, notavelmente pela dificuldade de encontrar uma documentação clara sobre o NVIC e pois a documentação fornecida para o PIT inverte a endianness dos registradores em relação aos registradores de GPIO.

5.3 Leitura de entradas digitais

Nesse laboratório tivemos dificuldades relacionadas à leitura do estado dos botões. O problema encontrado ocorreu pois quando o desbloqueamos o *clock* para uma porta do controlador no código fornecido estávamos desabilitando o *clock* para todas as outras portas. Como o *clock* só afeta significativamente os pinos configurados como *input*, se o módulo *ledswi_hal* fosse inicializado por último (como é feito no código do professor) o problema não seria notado.

5.4 Estilo de Documentação

Não conseguimos nos adaptar ao estilo de comentários sugerido pelo professor, que nos parece introduzir uma quantidade desnecessária de burocracia. Formatar todos os comentários para caber dentro daquele quadrado tomava mais tempo que o planejamento e a implementação do código e muita informação redundante estava sendo inserida (bastava ler a interface da função para saber seu nome e o tipo de seus argumentos). Escolhemos substituir todos os comentários do código pelo padrão *javadoc*, com o qual estamos mais familiarizados, que é capaz de documentar de maneira eficiente o código.

5.5 Outros

Também tivemos dificuldades com a instalação do *Rhapsody Rational Modeler* no *Linux* (através do *Wine*), pois ele necessita da instalação das seguintes dlls nativas do *Windows*: *comctl32*, *msvcirt* e *riched20*.

É relevante lembrar de utilizar o modificador *volatile* para variáveis que serão modificadas durante o tratamento de interrupções, dessa maneira o otimizador sabe que não deve alterar comandos que envolvem essa variável.

6 Referências

- [1] Roteiro de Laboratório - Semanas 04 e 05 (disponibilizado para os alunos)
- [2] Projeto do Modelo Inicial do Sistema (disponibilizado para os alunos)
- [3] Código Fonte Inicial em Linguagem C (disponibilizado para os alunos)
- [4] Notas de Aula - Semanas 06 (disponibilizado para os alunos)
- [5] The C Preprocessor (Concatenation) <https://gcc.gnu.org/onlinedocs/cpp/Concatenation.html#Concatenation>

7 Apêndice

Listagem dos códigos fonte:

7.1 ../Sources/LedSwi/ledswi_hal.h

```
1  /* ***** */
2  /* File name:      ledswi_hal.h */
3  /* File description: Header file containing the function/methods */
4  /*                prototypes of ledswi.c */
5  /* Author name:    dloubach */
6  /* Creation date:   09jan2015 */
7  /* Revision date:   13abr2016 */
8  /* ***** */
9
10 #ifndef SOURCES_LEDSWI_LEDSWI_HAL_H_
11 #define SOURCES_LEDSWI_LEDSWI_HAL_H_
12
13 #define MAX_LED_SWI      04
14
15 typedef enum
16 {
17     SWITCH_OFF,
18     SWITCH_ON
19 } switch_status_type_e;
20
21 /**
22  * As the hardware board was designed with LEDs/Switches sharing
23  * the same pins, this method configures how many LEDS and switches
24  * will be available for the application
25  * @param cLedNum num of LEDs
26  * @param cSwitchNum num of Switches (cLedNum + cSwitchNum <= MAX_LED_SWI)
27  */
28 void ledswi_initLedSwitch(char cLedNum, char cSwitchNum);
29
30
31 /**
32  * set the led ON
33  * @param cLedNum which LED {1..4}
34  */
35 void ledswi_setLed(char cLedNum);
36
37
```

```
38 /**
39  * set the led OFF
40  * @param cLedNum which LED {1..4}
41  */
42 void ledswi_clearLed(char cLedNum);
43
44
45 /**
46  * return the switch status
47  * @param cSwitchNum which switch
48  * @return If the switch is ON or OFF
49  */
50 switch_status_type_e ledswi_getSwitchStatus(char cSwitchNum);
51
52
53 #endif /* SOURCES_LEDSWI_LEDSWI_HAL_H_ */
```

7.2 ../Sources/LedSwi/ledswi_hal.c

```
1  /* ***** */
2  /* File name:      ledswi_hal.c */
3  /* File description: This file has a couple of useful functions to */
4  /*                control LEDs and Switches from peripheral board */
5  /* Author name:    dloubach */
6  /* Creation date:   20jan2015 */
7  /* Revision date:   13abr2016 */
8  /* ***** */
9
10 #include "ledswi_hal.h"
11 #include "KL25Z/es670_peripheral_board.h"
12
13 #undef USING_OPENSDA_DEBUG
14
15 /**
16  * As the hardware board was designed with LEDs/Switches sharing
17  * the same pins, this method configures how many LEDS and switches
18  * will be available for the application
19  * @param cLedNum num of LEDs
20  * @param cSwitchNum num of Switches (cLedNum + cSwitchNum <= MAX_LED_SWI)
21  */
22 void ledswi_initLedSwitch(char cLedNum, char cSwitchNum)
23 {
24     /* un-gate port clock*/
25     SIM_SCGC5 |= SIM_SCGC5_PORTA(CGC_CLOCK_ENABLED);
26
27     /* set pin as gpio */
28 #ifndef USING_OPENSDA_DEBUG
29     PORTA_PCR1 = PORT_PCR_MUX(LS1_ALT);
30     PORTA_PCR2 = PORT_PCR_MUX(LS2_ALT);
31 #endif
32     PORTA_PCR4 = PORT_PCR_MUX(LS3_ALT);
33     PORTA_PCR5 = PORT_PCR_MUX(LS4_ALT);
34
35
36     /* check if the number to configured is according to
37     hardware dev kit */
38     if((cLedNum + cSwitchNum) <= MAX_LED_SWI)
39     {
40         /* max number of peripherals to configure is ok, carry on */
41         switch(cSwitchNum)
42         {
```

```

43         case 0:
44             /* no switches in system configuration */
45             /* all leds */
46             GPIOA_PDDR |= GPIO_PDDR_PDD(LS1_DIR_OUTPUT | LS2_DIR_OUTPUT |
LS3_DIR_OUTPUT | LS4_DIR_OUTPUT);
47             break;
48
49         case 1:
50             /* just 1 switch */
51             GPIOA_PDDR |= GPIO_PDDR_PDD(LS2_DIR_OUTPUT | LS3_DIR_OUTPUT |
LS4_DIR_OUTPUT);
52             GPIOA_PDDR &= ~GPIO_PDDR_PDD(LS1_DIR_INPUT);
53             break;
54
55         case 2:
56             /* just 2 switches */
57             GPIOA_PDDR |= GPIO_PDDR_PDD(LS3_DIR_OUTPUT | LS4_DIR_OUTPUT);
58             GPIOA_PDDR &= ~GPIO_PDDR_PDD(LS1_DIR_INPUT | LS2_DIR_INPUT);
59             break;
60
61         case 3:
62             /* 3 switches */
63             GPIOA_PDDR |= GPIO_PDDR_PDD(LS4_DIR_OUTPUT);
64             GPIOA_PDDR &= ~GPIO_PDDR_PDD(LS1_DIR_INPUT | LS2_DIR_INPUT |
LS3_DIR_INPUT);
65             break;
66
67         case 4:
68             /* 4 switches */
69             GPIOA_PDDR &= ~GPIO_PDDR_PDD(LS1_DIR_INPUT | LS2_DIR_INPUT |
LS3_DIR_INPUT | LS4_DIR_INPUT);
70             break;
71     } /* switch(cSwitchNum) */
72
73 } /* if((cLedNum + cSwitchNum) <= MAX_LED_SWI) */
74
75 }
76
77
78
79 /**
80  * set the led ON
81  * @param cLedNum which LED {1..4}
82  */

```

```

83 void ledswi_setLed(char cLedNum)
84 {
85     /* sanity check */
86     if(cLedNum <= MAX_LED_SWI)
87     {
88         switch(cLedNum)
89         {
90             case 1:
91                 GPIOA_PSOR = GPIO_PSOR_PTS0( (0x01U << LS1_PIN) );
92                 break;
93             case 2:
94                 GPIOA_PSOR = GPIO_PSOR_PTS0( (0x01U << LS2_PIN) );
95                 break;
96             case 3:
97                 GPIOA_PSOR = GPIO_PSOR_PTS0( (0x01U << LS3_PIN) );
98                 break;
99             case 4:
100                 GPIOA_PSOR = GPIO_PSOR_PTS0( (0x01U << LS4_PIN) );
101                 break;
102             } /* switch(cLedNum) */
103         } /* if(cLedNum <= MAX_LED_SWI) */
104     }
105 }
106
107
108
109 /**
110  * set the led OFF
111  * @param cLedNum which LED {1..4}
112  */
113 void ledswi_clearLed(char cLedNum)
114 {
115     /* sanity check */
116     if(cLedNum <= MAX_LED_SWI)
117     {
118         switch(cLedNum)
119         {
120             case 1:
121                 GPIOA_PCOR = GPIO_PCOR_PTC0( (0x01U << LS1_PIN) );
122                 break;
123             case 2:
124                 GPIOA_PCOR = GPIO_PCOR_PTC0( (0x01U << LS2_PIN) );
125                 break;
126             case 3:

```



```

127         GPIOA_PCOR = GPIO_PCOR_PTC0( (0x01U << LS3_PIN) );
128         break;
129     case 4:
130         GPIOA_PCOR = GPIO_PCOR_PTC0( (0x01U << LS4_PIN) );
131         break;
132     } /* switch(cLedNum) */
133
134 } /* if(cLedNum <= MAX_LED_SWI) */
135 }
136
137
138
139 /**
140  * return the switch status
141  * @param cSwitchNum which switch
142  * @return If the switch is ON or OFF
143  */
144 switch_status_type_e ledswi_getSwitchStatus(char cSwitchNum)
145 {
146     switch_status_type_e sstReturn = SWITCH_OFF;
147
148     /* sanity check */
149     if(cSwitchNum <= MAX_LED_SWI)
150     {
151         switch(cSwitchNum)
152         {
153             case 1:
154                 if(SWITCH_ON == ((GPIOA_PDIR & LS1_DIR_INPUT) >> LS1_PIN) )
155                     sstReturn = SWITCH_ON;
156                 break;
157
158             case 2:
159                 if(SWITCH_ON == ((GPIOA_PDIR & LS2_DIR_INPUT) >> LS2_PIN) )
160                     sstReturn = SWITCH_ON;
161                 break;
162
163             case 3:
164                 if(SWITCH_ON == ((GPIOA_PDIR & LS3_DIR_INPUT) >> LS3_PIN) )
165                     sstReturn = SWITCH_ON;
166                 break;
167
168             case 4:
169                 if(SWITCH_ON == ((GPIOA_PDIR & LS4_DIR_INPUT) >> LS4_PIN) )
170                     sstReturn = SWITCH_ON;

```

```
171         break;
172
173     } /* switch(cSwitchNum) */
174
175 } /* if(cSwitchNum <= MAX_LED_SWI) */
176
177 /* return the result */
178 return(sstReturn);
179
180 }
```

7.3 ../Sources/Mcg/mcg_hal.h

```
1  /* ***** */
2  /* File name:      mcg_hal.h */
3  /* File description: Header file containing the functions/methods */
4  /*                interfaces for handling the Multipurpose clock */
5  /*                generator module */
6  /* Author name:    dloubach */
7  /* Creation date:   21out2015 */
8  /* Revision date:   25fev2016 */
9  /* ***** */
10
11 #ifndef SOURCES_MCG_HAL_H_
12 #define SOURCES_MCG_HAL_H_
13
14 /* ***** */
15 /* Method name:      mcg_clockInit */
16 /* Method description: main board clk configuration */
17 /* Input params:     n/a */
18 /* Output params:    n/a */
19 /* ***** */
20 void mcg_clockInit(void);
21
22
23
24 #endif /* SOURCES_MCG_HAL_H_ */
```

7.4 ../Sources/Mcg/mcg_hal.c

```
1  /* ***** */
2  /* File name:      mcg_hal.c */
3  /* File description: Multipurpose clk generator hardware abstraction */
4  /*                  layer. Enables the clock configuration */
5  /* */
6  /*                  Modes of Operation */
7  /*                  FLL Engaged Internal (FEI)      = DEFAULT */
8  /*                  FLL Engaged External (FEE) */
9  /*                  FLL Bypassed Internal (FBI) */
10 /*                  FLL Bypassed External (FBE) */
11 /*                  PLL Engaged External (PEE) */
12 /*                  PLL Bypassed External (PBE) */
13 /*                  Bypassed Low Power Internal (BLPI) */
14 /*                  Bypassed Low Power External (BLPE) */
15 /*                  Stop */
16 /* */
17 /*                  For clock definitions, check the chapter */
18 /*                  5.4 Clock definitions from */
19 /*                  KL25 Sub-Family Reference Manual */
20 /* */
21 /* Author name:      dloubach */
22 /* Creation date:     21out2015 */
23 /* Revision date:     13abr2016 */
24 /* ***** */
25
26 #include "mcg_hal.h"
27
28 /* systems include */
29 #include "fsl_smc_hal.h"
30 #include "fsl_port_hal.h"
31 #include "fsl_clock_manager.h"
32
33 /* XTAL0 PTA18 */
34 #define XTAL0_PORT          PORTA
35 #define XTAL0_PIN           18U
36 #define XTAL0_PINMUX        kPortPinDisabled
37
38 /* XTAL0 PTA19 */
39 #define XTAL0_PORT          PORTA
40 #define XTAL0_PIN           19U
41 #define XTAL0_PINMUX        kPortPinDisabled
42
```

```

43 /* OSC0 configuration */
44 #define OSC0_INSTANCE          0U
45 #define OSC0_XTAL_FREQ        8000000U /* 08 MHz*/
46 #define OSC0_SC2P_ENABLE_CONFIG false
47 #define OSC0_SC4P_ENABLE_CONFIG false
48 #define OSC0_SC8P_ENABLE_CONFIG false
49 #define OSC0_SC16P_ENABLE_CONFIG false
50 #define MCG_HG00              kOscGainLow
51 #define MCG_RANGE0            kOscRangeVeryHigh
52 #define MCG_EREFS0            kOscSrcOsc
53
54 /* RTC external clock configuration. */
55 #define RTC_XTAL_FREQ          0U
56 #define RTC_SC2P_ENABLE_CONFIG false
57 #define RTC_SC4P_ENABLE_CONFIG false
58 #define RTC_SC8P_ENABLE_CONFIG false
59 #define RTC_SC16P_ENABLE_CONFIG false
60 #define RTC_OSC_ENABLE_CONFIG  false
61 #define RTC_CLK_OUTPUT_ENABLE_CONFIG false
62
63 /* RTC_CLKIN PTC1 */
64 #define RTC_CLKIN_PORT          PORTC
65 #define RTC_CLKIN_PIN           1U
66 #define RTC_CLKIN_PINMUX        kPortMuxAsGpio
67
68
69 #define CLOCK_VLPR              1U /* very low power run mode */
70 #define CLOCK_RUN               2U /* run mode */
71
72 #ifndef CLOCK_INIT_CONFIG
73 #define CLOCK_INIT_CONFIG CLOCK_RUN
74 #endif
75
76
77 /* Configuration for enter VLPR mode, Core clock = 4MHz */
78 const clock_manager_user_config_t g_defaultClockConfigVlpr =
79 {
80     .mcgConfig =
81     {
82         .mcg_mode          = kMcgModeBLPI,          // Work in BLPI mode
83         .irclkEnable        = true,                  // MCGIRCLK enable
84         .irclkEnableInStop = false,                  // MCGIRCLK disable in STOP mode
85         .ircs               = kMcgIrcFast,           // Select IRC4M
86         .fcrdiv             = 0U,                    // FCRDIV is 0

```

```

87
88     .frdiv    = 0U,
89     .drs      = kMcgDcoRangeSelLow,           // Low frequency range
90     .dmx32    = kMcgDmx32Default,           // DCO has a default range of 25%
91
92     .pll0EnableInFllMode = false,           // PLL0 disable
93     .pll0EnableInStop  = false,           // PLL0 disable in STOP mode
94     .prdiv0          = 0U,
95     .vdiv0           = 0U,
96 },
97 .simConfig =
98 {
99     .pllFllSel = kClockPllFllSelFll,       // PLLFLLSEL select FLL
100    .er32kSrc  = kClockEr32kSrcLpo,        // ERCLK32K selection, use LPO
101    .outdiv1    = 0U,
102    .outdiv4    = 4U,
103 },
104 .oscerConfig =
105 {
106     .enable      = true,                 // OSCERCLK enable
107     .enableInStop = false,              // OSCERCLK disable in STOP mode
108 }
109 };
110
111 /* Configuration for enter RUN mode, Core clock = 40 MHz */
112 /*
113  * 24.5.1.1 Initializing the MCG
114  * KL25 Sub-Family Reference Manual, Rev. 3, September 2012
115  *
116  * Refer also to
117  * Table 24-18. MCG modes of operation
118  *
119  * On L-series devices the MCGFLLCLK frequency is limited to 48 MHz max
120  * The DCO is limited to the two lowest range settings (MCG_C4[DRST_DRS] must be set
121    * to either 0b00 or 0b01).
122  *
123  * FEE (FLL engaged external)
124  * fext / FLL_R must be in the range of 31.25 kHz to 39.0625 kHz
125  * FLL_R is the reference divider selected by the C1[FRDIV] bits
126  * F is the FLL factor selected by C4[DRST_DRS] and C4[DMX32] bits
127  *
128  * (fext / FLL_R) * F = (8 MHz / 256 ) * 1280 = 40 MHz
129  * */

```

```

130 const clock_manager_user_config_t g_defaultClockConfigRun =
131 {
132     /* ----- multipurpose clock generator configurations ----- */
133     .mcgConfig =
134     {
135         .mcg_mode          = kMcgModeFEE,          // Work in FEE mode
136
137         /* ----- MCGIRCLK settings ----- */
138         .irclkEnable       = true,                 // MCGIRCLK enable
139         .irclkEnableInStop = false,                // MCGIRCLK disable in STOP mode
140         .ircs              = kMcgIrcSlow,          // Select IRC 32kHz
141         .fcrdiv            = 0U,                   // FCRDIV is 0
142
143         /* ----- MCG FLL settings ----- */
144         .frdiv             = 0b011,                // Divide Factor is 256 (EXT OSC 8
MHz / 256 = 31.250 kHz)
145
146         // The resulting frequency must be in
the range 31.25 kHz to 39.0625 kHz
147         .drs               = kMcgDcoRangeSelMid,    // frequency range
148         .dmx32             = kMcgDmx32Default,      // DCO has a default range of 25%
149
150         /* ----- MCG PLL settings ----- */
151         .pll0EnableInFllMode = false,              // PLL0 disable
152         .pll0EnableInStop    = false,              // PLL0 disable in STOP mode
153         .prdiv0              = 0x0U,
154         .vdiv0               = 0x0U,
155     },
156     /* ----- system integration module configurations ----- */
157     .simConfig =
158     {
159         .pllFllSel = kClockPllFllSelFll,          // PLLFLLSEL select PLL
160         .er32kSrc  = kClockEr32kSrcLpo,           // ERCLK32K selection, use LPO
161         .outdiv1   = 0U,                           // core/system clock, as well as the
bus/flash clocks.
162         .outdiv4   = 1U,                           // bus and flash clock and is in
addition to the System clock divide ratio
163     },
164     /* ----- system oscillator output configurations ----- */
165     .oscerConfig =
166     {
167         .enable     = true,                        // OSCERCLK enable
168         .enableInStop = false,                    // OSCERCLK disable in STOP mode
169     }
170 };

```

```

170
171
172 /**
173  * Oscillator configuration
174  */
175 void mcg_initOsc0(void)
176 {
177     /* OSC0 configuration */
178     osc_user_config_t osc0Config =
179     {
180         .freq          = OSC0_XTAL_FREQ,
181         .hgo           = MCG_HGO0,
182         .range         = MCG_RANGE0,
183         .erefs         = MCG_EREFS0,
184         .enableCapacitor2p = OSC0_SC2P_ENABLE_CONFIG,
185         .enableCapacitor4p = OSC0_SC4P_ENABLE_CONFIG,
186         .enableCapacitor8p = OSC0_SC8P_ENABLE_CONFIG,
187         .enableCapacitor16p = OSC0_SC16P_ENABLE_CONFIG,
188     };
189
190     /* oscillator initialization */
191     CLOCK_SYS_OscInit(OSC0_INSTANCE, &osc0Config);
192 }
193
194
195
196 /**
197  * Function to initialize RTC external clock base on board configuration
198  */
199 void mcg_initRtcOsc(void)
200 {
201
202     #if RTC_XTAL_FREQ
203         // If RTC_CLKIN is connected, need to set pin mux. Another way for
204         // RTC clock is set RTC_OSC_ENABLE_CONFIG to use OSC0, please check
205         // reference manual for details
206         PORT_HAL_SetMuxMode(RTC_CLKIN_PORT, RTC_CLKIN_PIN, RTC_CLKIN_PINMUX);
207     #endif
208
209     #if ((OSC0_XTAL_FREQ != 32768U) && (RTC_OSC_ENABLE_CONFIG))
210     #error Set RTC_OSC_ENABLE_CONFIG will override OSC0 configuration and OSC0 must be 32k.
211     #endif
212
213     rtc_osc_user_config_t rtcOscConfig =

```



```

214     {
215         .freq                = RTC_XTAL_FREQ ,
216         .enableCapacitor2p   = RTC_SC2P_ENABLE_CONFIG ,
217         .enableCapacitor4p   = RTC_SC4P_ENABLE_CONFIG ,
218         .enableCapacitor8p   = RTC_SC8P_ENABLE_CONFIG ,
219         .enableCapacitor16p  = RTC_SC16P_ENABLE_CONFIG ,
220         .enableOsc           = RTC_OSC_ENABLE_CONFIG ,
221     };
222
223     /* OSC RTC initialization */
224     CLOCK_SYS_RtcOscInit(0U, &rtcOscConfig);
225 }
226
227
228
229 /**
230  * System clock configuration
231  */
232 void mcg_initSystemClock(void)
233 {
234     /* Set system clock configuration. */
235     #if (CLOCK_INIT_CONFIG == CLOCK_VLPR)
236         CLOCK_SYS_SetConfiguration(&g_defaultClockConfigVlpr);
237     #else
238         CLOCK_SYS_SetConfiguration(&g_defaultClockConfigRun);
239     #endif
240 }
241
242
243
244 /* ***** */
245 /* Method name:      mcg_clockInit */
246 /* Method description: main board clk configuration */
247 /* Input params:     n/a */
248 /* Output params:    n/a */
249 /* ***** */
250 void mcg_clockInit(void)
251 {
252     /* enable clock for PORTs */
253     CLOCK_SYS_EnablePortClock(PORTA_IDX);
254     CLOCK_SYS_EnablePortClock(PORTC_IDX);
255     CLOCK_SYS_EnablePortClock(PORTE_IDX);
256
257     /* set allowed power mode to allow all */

```

```
258     SMC_HAL_SetProtection(SMC, kAllowPowerModeAll);
259
260     /* configure OSC0 pin mux */
261     PORT_HAL_SetMuxMode(EXTALO_PORT, EXTALO_PIN, EXTALO_PINMUX);
262     PORT_HAL_SetMuxMode(XTALO_PORT, XTALO_PIN, XTALO_PINMUX);
263
264     /* setup OSC0 */
265     mcg_initOsc0();
266
267     /* setup OSC RTC */
268     mcg_initRtcOsc();
269
270     /* setup system clock */
271     mcg_initSystemClock();
272 }
```

7.5 ../Sources/PIT/pit_hal.h

```
1  /* ***** */
2  /* File name:      pit_hal.h */
3  /* File description: Header file containing the functions/methods */
4  /*                interfaces for handling the Periodic Interruption*/
5  /*                timer module */
6  /* Author name:    ddello */
7  /* Creation date:   10abr2016 */
8  /* Revision date:   10abr2016 */
9  /* ***** */
10
11 #ifndef SOURCES_PIT_PIT_HAL_H_
12 #define SOURCES_PIT_PIT_HAL_H_
13
14 /**
15  * Enables Periodic Interruption Timer module.
16  * (With the stop on debug flag set to on)
17  */
18 void pit_enable(void);
19
20 /**
21  * Start interruptions for given timer, unchained mode.
22  * Timer interruptions are masked.
23  *
24  * @param usTimer_numb The number for the desired timer (0,1)
25  * @param uiTimer_period The number of bus_clock cycles between interrupts
26  * @param fpInterrupt_handler Timer interrupt handler routine address pointer
27  */
28 void pit_start_timer_interrupt(unsigned short usTimer_numb, unsigned int
    uiTimer_period, void (*fpInterrupt_handler)(void));
29
30 /**
31  * Stop interruptions for given timer, unchained mode.
32  *
33  * @param usTimer_numb The number for the desired timer (0,1)
34  */
35 void pit_stop_timer_interrupt(unsigned short usTimer_numb);
36
37 /**
38  * Mark interruption as handled for the given timer, this should be called by timer
39  * interruption handlers once they are finished.
40  *
41  * @param usTimer_numb The number for the desired timer (0,1)
```

```
42  */
43  void pit_mark_interrupt_handled(unsigned short usTimer_numb);
44
45  /**
46   * Pit interruption handler. Checks what timer caused the interruption and call the
47   * correct timer interruption handler.
48   */
49  void PIT_IRQHandler(void);
50  #endif /* SOURCES_PIT_PIT_HAL_H_ */
```

7.6 ../Sources/PIT/pit_hal.c

```
1  /* ***** */
2  /* File name:      pit_hal.c */
3  /* File description: File containing the functions/methods */
4  /*                for handling the Periodic Interruption */
5  /*                Timer module */
6  /* Author name:    ddello */
7  /* Creation date:   10abr2016 */
8  /* Revision date:   13abr2016 */
9  /* ***** */
10 //Careful when handling PIT DOC! Bit endianness is inverted in relation to GPIO doc
11
12 #include "pit_hal.h"
13 #include "KL25Z/es670_peripheral_board.h"
14
15 #define PIT_IRQ_NUMBER PIT_IRQn
16
17 /**
18  *Default timer interruption handler. Does nothing.
19  */
20 static void _nop_handler(void){
21     PIT_TFLG0 |= PIT_TFLG_TIF(0x1u);
22     PIT_TFLG1 |= PIT_TFLG_TIF(0x1u);
23 }
24
25 static void (*fpTimer0Handler)(void) = &_nop_handler;
26 static void (*fpTimer1Handler)(void) = &_nop_handler;
27
28 /**
29  * Pit interruption handler. Checks what timer caused the interruption and call the
30  * correct timer interruption handler.
31  */
32 void PIT_IRQHandler(void){
33     if(PIT_TFLG0){
34         (*fpTimer0Handler)();
35     }
36     if(PIT_TFLG1){
37         (*fpTimer1Handler)();
38     }
39 }
40
41 /**
42  * Enables Periodic Interruption Timer module.
```

```

43  * (With the stop on debug flag set to on)
44  */
45  void pit_enable(void){
46      SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;
47      PIT_MCR &= ~PIT_MCR_MDIS(0x1u);
48      //Freeze in debug mode
49      PIT_MCR |= PIT_MCR_FRZ(0x1u);
50      NVIC_ClearPendingIRQ(PIT_IRQ_NUMBER);
51      NVIC_EnableIRQ(PIT_IRQ_NUMBER);
52  }
53
54  /**
55   * Start interruptions for given timer, unchained mode.
56   * Timer interruptions are masked.
57   *
58   * @param usTimer_numb The number for the desired timer (0,1)
59   * @param uiTimer_period The number of bus_clock cycles between interrupts
60   * @param fpInterrupt_handler Timer interrupt handler routine address pointer
61   */
62  void pit_start_timer_interrupt(unsigned short usTimer_numb, unsigned int
      uiTimer_period, void (*fpInterrupt_handler)(void)){
63      if(!usTimer_numb){
64          fpTimer0Handler = fpInterrupt_handler;
65          PIT_LDVAL0 = PIT_LDVAL_TSV(uiTimer_period);
66          PIT_TCTRL0 &= ~PIT_TCTRL_CHN(0x1u); /*Disable chain mode*/
67          PIT_TCTRL0 |= PIT_TCTRL_TIE(0x1u); /*Enable interrupts for timer 0*/
68          PIT_TCTRL0 |= PIT_TCTRL_TEN(0x1u); /*Enable timer 0*/
69      }else{
70          fpTimer1Handler = fpInterrupt_handler;
71          PIT_LDVAL1 = PIT_LDVAL_TSV(uiTimer_period);
72          PIT_TCTRL1 &= ~PIT_TCTRL_CHN(0x1u); /*Disable chain mode*/
73          PIT_TCTRL1 |= PIT_TCTRL_TIE(0x1u); /*Enable interrupts for timer 1*/
74          PIT_TCTRL1 |= PIT_TCTRL_TEN(0x1u); /*Enable timer 1*/
75      }
76  }
77
78  /**
79   * Stop interruptions for given timer, unchained mode.
80   *
81   * @param usTimer_numb The number for the desired timer (0,1)
82   */
83  void pit_stop_timer_interrupt(unsigned short usTimer_numb){
84      if(!usTimer_numb){
85          PIT_TCTRL0 &= ~PIT_TCTRL_TIE(0x1u);

```

```

86     PIT_TCTRL0 &= ~PIT_TCTRL_TEN(0x1u);
87 }else{
88     PIT_TCTRL1 &= ~PIT_TCTRL_TIE(0x1u);
89     PIT_TCTRL1 &= ~PIT_TCTRL_TEN(0x1u);
90 }
91 }
92
93 /**
94  * Mark interruption as handled for the given timer, this should be called by timer
95  * interruption handlers once they are finished.
96  *
97  * @param usTimer_numb The number for the desired timer (0,1)
98  */
99 void pit_mark_interrupt_handled(unsigned short usTimer_numb){
100     if(!usTimer_numb){
101         PIT_TFLG0 |= PIT_TFLG_TIF(0x1u);
102     }else{
103         PIT_TFLG1 |= PIT_TFLG_TIF(0x1u);
104     }
105 }

```

7.7 ../Sources/SevenSeg/sevenseg_hal.h

```
1  /* ***** */
2  /* File name:      sevenseg_hal.h */
3  /* File description: Header file containing the functions/methods */
4  /*                interfaces for handling SEVEN SEGMENT DISPLAY */
5  /*                from the peripheral board */
6  /* Author name:    ddello */
7  /* Creation date:   18mar2016 */
8  /* Revision date:   13abr2016 */
9  /* ***** */
10
11 #ifndef SOURCES_SEVEN_SEGMENT_HAL_H_
12 #define SOURCES_SEVEN_SEGMENT_HAL_H_
13
14 #include "KL25Z/es670_peripheral_board.h"
15
16 #define MAX_SEGMENT_NUMBER 8
17 #define MAX_DISP_NUMBER 4
18
19
20 typedef enum
21 {
22     SEG_A = SEGA_PIN,
23     SEG_B = SEGB_PIN,
24     SEG_C = SEGC_PIN,
25     SEG_D = SEGD_PIN,
26     SEG_E = SEGE_PIN,
27     SEG_F = SEGF_PIN,
28     SEG_G = SEGG_PIN,
29     SEG_DP = SEGDP_PIN,
30     SEG_END = -1
31 } seven_segment_seg_type_e;
32
33 typedef enum
34 {
35     DISP_1 = SEG_DISP1_PIN,
36     DISP_2 = SEG_DISP2_PIN,
37     DISP_3 = SEG_DISP3_PIN,
38     DISP_4 = SEG_DISP4_PIN,
39 } seven_segment_disp_type_e;
40
41 /**
42 * Initialize the seven segment display
```



```

43 */
44 void sevenseg_init(void);
45
46 /**
47  * Sets only the selected segments as high. Setting the others as low
48  * @param epDet_segments = Array with the segments that should be set as on (Last
49    element should be SEG_END)
50 */
51 void sevenseg_setSegs(seven_segment_seg_type_e* epSet_segments);
52
53 /**
54  * Shows the value written in the segment pins to the
55  * given display after clearing the others
56  * @param eDisplay the display to initialize.
57 */
58 void sevenseg_setDisp(seven_segment_disp_type_e eDisplay);
59
60 /**
61  * Shows the passed value in hexadecimal format in the seven segment display.
62  * @param uiHex the value to be printed
63 */
64 void sevenseg_printHex(unsigned int uiHex);
65
66 /**
67  * Shows the passed value in decimal format in the seven segment display.
68  * @param uiDec the value to be printed
69 */
70 void sevenseg_printDec(unsigned int uiDec);
71
72 /**
73  * Converts the less significative decimal digit of the argument into it's seven
74  * segment display configuration
75  * @param usDec the value to be converted (-1 if none should be displayed)
76  * @param epRet address for results (should be a allocated array of minimal 9 elements)
77  * @return epRet
78 */
79 seven_segment_seg_type_e* sevenseg_dec2segArray(unsigned short usDec,
80    seven_segment_seg_type_e* epRet);
81
82 /**
83  * Converts the less significative hexadecimal digit of the argument into it's seven
84  * segment display configuration
85  * @param usHex the value to be converted (-1 if none should be displayed)

```

```
85  * @param epRet address for results (should be a allocated array of minimal 9 elements)
86  *
87  * @return epRet
88  */
89  seven_segment_seg_type_e* sevenseg_hex2segArray(unsigned short usHex,
           seven_segment_seg_type_e* epRet);
90
91 #endif /* SOURCES_SEVEN_SEGMENT_HAL_H_ */
```

7.8 ../Sources/SevenSeg/sevenseg_hal.c

```
1  /* ***** */
2  /* File name:      sevenseg_hal.c */
3  /* File description: File containing the functions/methods */
4  /*                for handling SEVEN SEGMENT DISPLAY */
5  /*                from the peripheral board */
6  /* Author name:    ddello */
7  /* Creation date:   18mar2016 */
8  /* Revision date:   13abr2016 */
9  /* ***** */
10
11 #include "GPIO/gpio_hal.h"
12 #include "sevenseg_hal.h"
13 #include "math.h"
14 #include "KL25Z/es670_peripheral_board.h"
15 #include "PIT/pit_hal.h"
16
17 #define SEV_SEG_SEGMENT_MASK GPIO_HIGH << SEGA_PIN | GPIO_HIGH << SEGB_PIN | GPIO_HIGH
18     << SEGC_PIN | GPIO_HIGH << SEGD_PIN | GPIO_HIGH << SEGE_PIN | GPIO_HIGH << SEGF_PIN
19     | GPIO_HIGH << SEGG_PIN | GPIO_HIGH << SEGDP_PIN
20 #define SEV_SEG_DISP_MASK GPIO_HIGH << SEG_DISP1_PIN | GPIO_HIGH << SEG_DISP2_PIN |
21     GPIO_HIGH << SEG_DISP3_PIN | GPIO_HIGH << SEG_DISP4_PIN
22
23 #define SEVEN_SEG_PIT_PERIOD 0x0000F423 /* 62500 cycles = 3.125ms (20MHz)*/
24
25 static unsigned short usIsHex = 0;
26 static unsigned int uiPrintVal = -1;
27
28 /**
29  * Interrupt handler for updating in display configuration
30  */
31 void _sevenseg_interrupt_handler(void){
32     static seven_segment_disp_type_e epDisplays[] = {DISP_1, DISP_2, DISP_3, DISP_4};
33     static seven_segment_seg_type_e epSeg_array[MAX_SEGMENT_NUMBER+1];
34     static volatile unsigned short usCur_disp = 0;
35     if(usIsHex){
36         sevenseg_hex2segArray(uiPrintVal/pow(16,MAX_DISP_NUMBER-1-usCur_disp),
37             epSeg_array);
38     }else{
39         sevenseg_dec2segArray(uiPrintVal/pow(10,MAX_DISP_NUMBER-1-usCur_disp),
40             epSeg_array);
41     }
42     sevenseg_setSegs(epSeg_array);
43 }
```

```

38     sevenseg_setDisp(epDisplays[usCur_disp]);
39     usCur_disp = (usCur_disp+1)%MAX_DISP_NUMBER;
40     pit_mark_interrupt_handled(SEV_SEG_PIT_TIMER_NUMB);
41 }
42
43 /**
44  * Initialize the seven segment display
45  */
46 void sevenseg_init(void){
47     GPIO_UNGATE_PORT(SEV_SEG_PORT_ID);
48
49     // Init the Seven Segment segment control pins as OUTPUT
50     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGA_PIN, GPIO_OUTPUT);
51     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGB_PIN, GPIO_OUTPUT);
52     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGC_PIN, GPIO_OUTPUT);
53     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGD_PIN, GPIO_OUTPUT);
54     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGE_PIN, GPIO_OUTPUT);
55     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGF_PIN, GPIO_OUTPUT);
56     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGG_PIN, GPIO_OUTPUT);
57     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGDP_PIN, GPIO_OUTPUT);
58     // Init the Seven Segment segment display pins as OUTPUT
59     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEG_DISP1_PIN, GPIO_OUTPUT);
60     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEG_DISP2_PIN, GPIO_OUTPUT);
61     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEG_DISP3_PIN, GPIO_OUTPUT);
62     GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEG_DISP4_PIN, GPIO_OUTPUT);
63
64     //Init pit interrupts
65     pit_enable();
66     //Init timer 0
67     pit_start_timer_interrupt(SEV_SEG_PIT_TIMER_NUMB, SEVEN_SEG_PIT_PERIOD,
        &_sevenseg_interrupt_handler);
68 }
69
70
71 /**
72  * Sets only the selected segments as high. Setting the others as low
73  * @param epDet_segments = Array with the segments that should be set as on (Last
        element should be SEG_END)
74  */
75 void sevenseg_setSegs(seven_segment_seg_type_e* epSet_segments){
76     //Clear all segments.
77     GPIO_WRITE_MASK(SEV_SEG_PORT_ID, SEV_SEG_SEGMENT_MASK, GPIO_LOW);
78     //Set the selected segments to high
79     for(unsigned short usCounter = 0; epSet_segments[usCounter] != SEG_END; usCounter++){

```

```

80     GPIO_WRITE_PIN(SEV_SEG_PORT_ID, epSet_segments[usCounter], GPIO_HIGH);
81 }
82 }
83
84 /**
85  * Shows the value written in the segment pins to the
86  * given display after clearing the others
87  * @param eDisplay the display to initialize.
88  */
89 void sevenseg_setDisp(seven_segment_disp_type_e eDisplay){
90     //Clear all displays
91     GPIO_WRITE_MASK(SEV_SEG_PORT_ID, SEV_SEG_DISP_MASK, GPIO_LOW);
92     //Activate the selected display
93     GPIO_WRITE_PIN(SEV_SEG_PORT_ID, eDisplay, GPIO_HIGH);
94 }
95
96 /**
97  * Shows the passed value in hexadecimal format in the seven segment display.
98  * @param uiHex the value to be printed
99  */
100 void sevenseg_printHex(unsigned int uiHex){
101     usIsHex = 1;
102     uiPrintVal = uiHex;
103 }
104
105 /**
106  * Shows the passed value in decimal format in the seven segment display.
107  * @param uiDec the value to be printed
108  */
109 void sevenseg_printDec(unsigned int uiDec){
110     usIsHex = 0;
111     uiPrintVal = uiDec;
112 }
113
114 /**
115  * Converts the less significative decimal digit of the argument into it's seven
116  * segment display configuration
117  * @param usDec the value to be converted (-1 if none should be displayed)
118  * @param epRet address for results (should be a allocated array of minimal 9 elements)
119  *
120  * @return epRet
121  */
122 seven_segment_seg_type_e* sevenseg_dec2segArray(unsigned short usDec,
        seven_segment_seg_type_e* epRet){

```

```

123  if(usDec < 0){
124      epRet[0] = SEG_END;
125      return epRet;
126  }
127  epRet[0] = SEG_A;
128  epRet[1] = SEG_B;
129  epRet[2] = SEG_C;
130  epRet[3] = SEG_D;
131  epRet[4] = SEG_E;
132  epRet[5] = SEG_F;
133  epRet[6] = SEG_G;
134  epRet[7] = SEG_END;
135  switch(usDec%10){
136  case 0:
137      //{SEG_A,SEG_B,SEG_C,SEG_D,SEG_G,SEG_E,SEG_F,SEG_END};
138      epRet[7] = SEG_END;
139      break;
140  case 1:
141      //{SEG_B,SEG_C,SEG_END};
142      epRet[0] = SEG_B;
143      epRet[1] = SEG_C;
144      epRet[2] = SEG_END;
145      break;
146  case 2:
147      //{SEG_A,SEG_B,SEG_G,SEG_D,SEG_E,SEG_END};
148      epRet[2] = SEG_G;
149      epRet[5] = SEG_END;
150      break;
151  case 3:
152      //{SEG_A,SEG_B,SEG_C,SEG_D,SEG_G,SEG_END}
153      epRet[4] = SEG_G;
154      epRet[5] = SEG_END;
155      break;
156  case 4:
157      //{SEG_G, SEG_B,SEG_C,SEG_F, SEG_END}
158      epRet[0] = SEG_G;
159      epRet[3] = SEG_F;
160      epRet[4] = SEG_END;
161      break;
162  case 5:
163      //{SEG_A,SEG_G,SEG_C,SEG_D,SEG_F,SEG_END}
164      epRet[1] = SEG_G;
165      epRet[4] = SEG_F;
166      epRet[5] = SEG_END;

```

```

167     break;
168 case 6:
169     //{SEG_A,SEG_G,SEG_C,SEG_D,SEG_E,SEG_F,SEG_END}
170     epRet[1] = SEG_G;
171     epRet[6] = SEG_END;
172     break;
173 case 7:
174     //{SEG_A,SEG_B,SEG_C,SEG_END}
175     epRet[3] = SEG_END;
176     break;
177 case 8:
178     //{SEG_A,SEG_B,SEG_C,SEG_D,SEG_E,SEG_F,SEG_G,SEG_END}
179     break;
180 case 9:
181     //{SEG_A,SEG_B,SEG_C,SEG_F,SEG_G,SEG_END}
182     epRet[3] = SEG_F;
183     epRet[4] = SEG_G;
184     epRet[5] = SEG_END;
185     break;
186 }
187 return epRet;
188 }
189
190 /**
191  * Converts the less significative hexadecimal digit of the argument into it's seven
192  * segment display configuration
193  * @param usHex the value to be converted (-1 if none should be displayed)
194  * @param epRet address for results (should be a allocated array of minimal 9 elements)
195  *
196  * @return epRet
197  */
198 seven_segment_seg_type_e* sevenseg_hex2segArray(unsigned short usHex,
199         seven_segment_seg_type_e* epRet){
200     if(usHex < 0){
201         epRet[0] = SEG_END;
202         return epRet;
203     }
204     epRet[0] = SEG_A;
205     epRet[1] = SEG_B;
206     epRet[2] = SEG_C;
207     epRet[3] = SEG_D;
208     epRet[4] = SEG_E;
209     epRet[5] = SEG_F;
210     epRet[6] = SEG_G;

```

```

210 epRet[7] = SEG_END;
211 switch(usHex%16){
212     case 0:
213     case 1:
214     case 2:
215     case 3:
216     case 4:
217     case 5:
218     case 6:
219     case 7:
220     case 8:
221     case 9:
222         return sevenseg_dec2segArray(usHex%16, epRet);
223         break;
224     case 10: //A
225         //{SEG_A,SEG_B,SEG_C,SEG_G,SEG_E,SEG_F,SEG_END}
226         epRet[3] = SEG_G;
227         epRet[6] = SEG_END;
228         break;
229     case 11: //B (b)
230         //{SEG_G,SEG_F,SEG_C,SEG_D,SEG_E,SEG_END}
231         epRet[0] = SEG_G;
232         epRet[1] = SEG_F;
233         epRet[5] = SEG_END;
234         break;
235     case 12: //C
236         //{SEG_A,SEG_E,SEG_F,SEG_D,SEG_END}
237         epRet[1] = SEG_E;
238         epRet[2] = SEG_F;
239         epRet[4] = SEG_END;
240         break;
241     case 13: //D (d)
242         //{SEG_G,SEG_B,SEG_C,SEG_D,SEG_E,SEG_END}
243         epRet[0] = SEG_G;
244         epRet[5] = SEG_END;
245         break;
246     case 14: //E
247         //{SEG_A,SEG_G,SEG_F,SEG_D,SEG_E,SEG_END}
248         epRet[1] = SEG_G;
249         epRet[2] = SEG_F;
250         epRet[5] = SEG_END;
251         break;
252     case 15: //F
253         //{SEG_A,SEG_E,SEG_F,SEG_G,SEG_END}

```



```
254         epRet[1] = SEG_E;
255         epRet[2] = SEG_F;
256         epRet[3] = SEG_G;
257         epRet[4] = SEG_END;
258         break;
259     }
260     return epRet;
261 }
```

7.9 ../Sources/GPIO/gpio_hal.h

```
1  /* ***** */
2  /* File name:      gpio_hal.h */
3  /* File description: This file has a couple of useful macros to */
4  /*                control and init the GPIO pins from the KLM25Z */
5  /* Author name:    ddello */
6  /* Creation date:   01abr2016 */
7  /* Revision date:   13abr2016 */
8  /* ***** */
9
10 #ifndef SOURCES_GPIO_GPIO_HAL_H_
11 #define SOURCES_GPIO_GPIO_HAL_H_
12
13 #include "KL25Z/es670_peripheral_board.h"
14
15 /* GPIO input / output */
16 #define GPIO_INPUT      0x00U
17 #define GPIO_OUTPUT      0x01U
18
19 #define GPIO_MUX_ALT      0x01u
20
21 #define GPIO_HIGH      1
22 #define GPIO_LOW      0
23
24 /**
25  * Ungates the clock for a gpio port
26  * @param PORT_ID the GPIO port id(A,B)
27  */
28 #define GPIO_UNGATE_PORT(PORT_ID)\
29     _GPIO_UNGATE_PORT(PORT_ID)
30
31 //Wrapper macro above is needed for argument expansion when using concatenation
32 #define _GPIO_UNGATE_PORT(PORT_ID)\
33     /* un-gate port clock*/\
34     SIM_SCGC5 |= SIM_SCGC5_PORT ## PORT_ID (CGC_CLOCK_ENABLED)
35
36 /**
37  * inits a pin as GPIO in the given direction
38  * @param PORT_ID the GPIO port id(A,B)
39  * @param PIN_NUM pin number in port
40  * @param DIR pin direction (GPIO_HIGH, GPIO_LOW)
41  */
42 #define GPIO_INIT_PIN(PORT_ID, PIN_NUM, DIR)\
```

```

43     _GPIO_INIT_PIN(PORT_ID, PIN_NUM, DIR)
44
45 //Wrapper macro above is needed for argument expansion when using concatenation
46 #define _GPIO_INIT_PIN(PORT_ID, PIN_NUM, DIR)\
47     /* set pin as gpio */\
48     PORT ## PORT_ID ## _PCR ## PIN_NUM = PORT_PCR_MUX(GPIO_MUX_ALT);\
49     /* Set pin direction */\
50     if(DIR == GPIO_OUTPUT){\
51         GPIO ## PORT_ID ## _PDDR |= GPIO_PDDR_PDD(0x01 << PIN_NUM);\
52     }else{\
53         GPIO ## PORT_ID ## _PDDR &= ~GPIO_PDDR_PDD(0x01 << PIN_NUM);\
54     }
55
56
57 /**
58  * Writes a pin with the given value
59  * @param PORT_ID the GPIO port id(A,B)
60  * @param PIN_NUM pin number in port
61  * @param VAL pin value (GPIO_HIGH, GPIO_LOW)
62  */
63 #define GPIO_WRITE_PIN(PORT_ID, PIN_NUM, VAL)\
64     _GPIO_WRITE_PIN(PORT_ID, PIN_NUM, VAL)
65
66 //Wrapper macro above is needed for argument expansion when using concatenation
67 #define _GPIO_WRITE_PIN(PORT_ID, PIN_NUM, VAL)\
68     if(VAL == GPIO_HIGH){\
69         GPIO ## PORT_ID ## _PSOR = GPIO_PSOR_PTSO( (0x01U << PIN_NUM) );\
70     }else{\
71         GPIO ## PORT_ID ## _PCOR = GPIO_PCOR_PTCO( (0x01U << PIN_NUM) );\
72     }
73
74 /**
75  * Writes the given value to the pins given in the MASK
76  * @param PORT_ID the GPIO port id(A,B)
77  * @param MASK 31 bit Mask with 1 in the bits corresponding to the pins of interest.
78  * @param VAL pins value (GPIO_HIGH, GPIO_LOW)
79  */
80 #define GPIO_WRITE_MASK(PORT_ID, MASK, VAL)\
81     _GPIO_WRITE_MASK(PORT_ID, MASK, VAL)
82
83 #define _GPIO_WRITE_MASK(PORT_ID, MASK, VAL)\
84     if(VAL == GPIO_HIGH){\
85         GPIO ## PORT_ID ## _PSOR = GPIO_PSOR_PTSO(MASK);\
86     }else{\

```

```

87     GPIO ## PORT_ID ## _PCOR = GPIO_PCOR_PTCO(MASK);\
88 }
89
90
91 /**
92  * Reads the status of a GPIO PIN
93  * @param PORT_ID the GPIO port id(A,B)
94  * @param PIN_NUM pin number in port
95  * @param VAL pin value (GPIO_HIGH, GPIO_LOW)
96  */
97 #define GPIO_READ_PIN(PORT_ID, PIN_NUM)\
98     _GPIO_READ_PIN(PORT_ID, PIN_NUM)
99
100 //Wrapper macro above is needed for argument expansion when using concatenation
101 #define _GPIO_READ_PIN(PORT_ID, PIN_NUM)\
102     ((GPIO ## PORT_ID ## _PDIR & (0x01u << PIN_NUM)) >> PIN_NUM) )
103
104 #endif /* SOURCES_GPIO_GPIO_HAL_H_ */

```

7.10 ../Sources/Main/es670.c

```
1 #include "KL25Z/es670_peripheral_board.h"
2 #include "LedSwi/ledswi_hal.h"
3 #include "McG/mcg_hal.h"
4 #include "Buzzer/buzzer_hal.h"
5 #include "SevenSeg/sevenseg_hal.h"
6 #include "PIT/pit_hal.h"
7 #include "Util/util.h"
8
9
10 int main(void)
11 {
12     mcg_clockInit();
13     ledswi_initLedSwitch(1,3);
14     sevenseg_init();
15     sevenseg_printHex(0xABCDu);
16     buzzer_initPeriodic(0xB18Eu);    //440Hz (Base 20MHz)
17     ledswi_setLed(4);
18     while(1){
19         if(ledswi_getSwitchStatus(3) == SWITCH_OFF || ledswi_getSwitchStatus(2) ==
20             SWITCH_OFF || ledswi_getSwitchStatus(1) == SWITCH_OFF){
21             ledswi_setLed(0x4);
22         }else{
23             ledswi_clearLed(0x4);
24         }
25     }
26     /* Never leave main */
27     return 0;
28 }
```

7.11 ../Sources/Buzzer/buzzer_hal.c

```
1  /* ***** */
2  /* File name:      buzzer_hal.c */
3  /* File description: File dedicated to the hardware abstraction layer*/
4  /*                related buzzer from the peripheral board */
5  /* Author name:    dloubach */
6  /* Creation date:   12jan2016 */
7  /* Revision date:   13abr2016 */
8  /* ***** */
9
10 #include "GPIO/gpio_hal.h"
11 #include "buzzer_hal.h"
12 #include "KL25Z/es670_peripheral_board.h"
13 #include "PIT/pit_hal.h"
14
15 /**
16  * Initialize the buzzer device
17  */
18 void buzzer_init(void)
19 {
20     GPIO_UNGATE_PORT(BUZZER_PORT_ID);
21     GPIO_INIT_PIN(BUZZER_PORT_ID, BUZZER_PIN, GPIO_OUTPUT);
22     pit_enable();
23 }
24
25
26
27 /**
28  * Clear the buzzer
29  */
30 void buzzer_clearBuzz(void)
31 {
32     GPIO_WRITE_PIN(BUZZER_PORT_ID, BUZZER_PIN, GPIO_LOW);
33 }
34
35
36
37 /**
38  * Set the buzzer
39  */
40 void buzzer_setBuzz(void)
41 {
42     GPIO_WRITE_PIN(BUZZER_PORT_ID, BUZZER_PIN, GPIO_HIGH);
```

```

43 }
44
45 /**
46  * Handler for buzzer interruptions
47  */
48 void _buzzer_interrupt_handler(void){
49     static volatile unsigned short usBusOn = 0;
50     if(!usBusOn){
51         buzzer_setBuzz();
52     }else{
53         buzzer_clearBuzz();
54     }
55     usBusOn = !usBusOn;
56     pit_mark_interrupt_handled(BUZZER_PIT_TIMER_NUMB);
57 }
58
59 /**
60  * Starts the buzzer with the specified period
61  *
62  * @param uiPeriod The period of the buzzer signal, in clock cycles (40MHz)
63  */
64 void buzzer_initPeriodic(unsigned int uiPeriod){
65     //Init timer 1
66     pit_start_timer_interrupt(BUZZER_PIT_TIMER_NUMB, uiPeriod/2,
67         &_buzzer_interrupt_handler);
68 }
69
70 /**
71  * Stops any periodic buzzer signal
72  */
73 void buzzer_stopPeriodic(void){
74     pit_stop_timer_interrupt(BUZZER_PIT_TIMER_NUMB);
75 }

```

7.12 ../Sources/Buzzer/buzzer_hal.h

```
1  /* ***** */
2  /* File name:      buzzer_hal.h */
3  /* File description: Header file containing the functions/methods */
4  /*                interfaces for handling BUZZER from the */
5  /*                peripheral board */
6  /* Author name:    dloubach */
7  /* Creation date:   12jan2016 */
8  /* Revision date:   13abr2016 */
9  /* ***** */
10
11 #ifndef SOURCES_BUZZER_HAL_H_
12 #define SOURCES_BUZZER_HAL_H_
13
14 /**
15  * Initialize the buzzer device
16  */
17 void buzzer_init(void);
18
19
20 /**
21  * Clear the buzzer
22  */
23 void buzzer_clearBuzz(void);
24
25
26 /**
27  * Set the buzzer
28  */
29 void buzzer_setBuzz(void);
30
31 /**
32  * Starts the buzzer with the specified period
33  *
34  * @param uiPeriod The period of the buzzer signal, in clock cycles (40MHz)
35  */
36 void buzzer_initPeriodic(unsigned int uiPeriod);
37
38 /**
39  * Stops any periodic buzzer signal
40  */
41 void buzzer_stopPeriodic(void);
42
```


43

44 `#endif /* SOURCES_BUZZER_HAL_H_ */`

7.13 ../Sources/KL25Z/es670_peripheral_board.h

```
1  /* ***** */
2  /* File name:      es670_peripheral_board.h */
3  /* File description: Header file containing the peripherals mapping */
4  /*                of the peripheral board for the ES670 hardware*/
5  /* Author name:    dloubach */
6  /* Creation date:   16out2015 */
7  /* Revision date:   25fev2016 */
8  /* ***** */
9
10 #ifndef SOURCES_ES670_PERIPHERAL_BOARD_H_
11 #define SOURCES_ES670_PERIPHERAL_BOARD_H_
12
13 /* system includes */
14 #include <MKL25Z4.h>
15
16 /*                General uC definitions */
17
18 /* Clock gate control */
19 #define CGC_CLOCK_DISABLED      0x00U
20 #define CGC_CLOCK_ENABLED       0x01U
21
22 /* GPIO DIRECTION */
23 #define GPIO_OUTPUT              0x01U
24
25 /* Workaround for PORT_ID macro expansion to stop at port level*/
26 typedef int A;
27 typedef int B;
28 typedef int C;
29 typedef int D;
30 typedef int E;
31
32 /*                END OF General uC definitions */
33
34
35 /*                BUZZER Definitions */
36 #define BUZZER_PORT_BASE_PNT     PORTD /*
37     peripheral port base pointer */
37 #define BUZZER_GPIO_BASE_PNT    PTD /*
38     peripheral gpio base pointer */
38 #define BUZZER_PORT_ID          D /* peripheral
39     port identifier*/
39
```

```

40 #define BUZZER_PIT_TIMER_NUMB    1
41
42 #define BUZZER_PIN                0                                /* buzzer
    pin */
43 #define BUZZER_DIR                kGpioDigitalOutput
44 #define BUZZER_ALT                0x01u
45 /*                                END OF BUZZER definitions */
46
47
48 /*                                LED and SWITCH Definitions */
49 #define LS_PORT_BASE_PNT          PORTA                            /*
    peripheral port base pointer */
50 #define LS_PORT_ID                A                                /*
    peripheral port identifier*/
51 #define LS_GPIO_BASE_PNT          PTA                              /*
    peripheral gpio base pointer */
52
53 /* THIS PIN CONFLICTS WITH PTA1 USED AS UART0_RX IN THE OPENSDA SERIAL DEBUG PORT */
54 #define LS1_PIN                   1U                                /*
    led/switch #1 pin */
55 #define LS1_DIR_OUTPUT             (GPIO_OUTPUT << LS1_PIN)
56 #define LS1_DIR_INPUT             (GPIO_OUTPUT << LS1_PIN)
57 #define LS1_ALT                   0x01u                            /* GPIO
    alternative */
58
59 /* THIS PIN CONFLICTS WITH PTA2 USED AS UART0_TX IN THE OPENSDA SERIAL DEBUG PORT */
60 #define LS2_PIN                   2U                                /*
    led/switch #2 pin */
61 #define LS2_DIR_OUTPUT             (GPIO_OUTPUT << LS2_PIN)
62 #define LS2_DIR_INPUT             (GPIO_OUTPUT << LS2_PIN)
63 #define LS2_ALT                   LS1_ALT
64
65 #define LS3_PIN                   4U                                /*
    led/switch #3 pin */
66 #define LS3_DIR_OUTPUT             (GPIO_OUTPUT << LS3_PIN)
67 #define LS3_DIR_INPUT             (GPIO_OUTPUT << LS3_PIN)
68 #define LS3_ALT                   LS1_ALT
69
70 #define LS4_PIN                   5U                                /*
    led/switch #4 pin */
71 #define LS4_DIR_OUTPUT             (GPIO_OUTPUT << LS4_PIN)
72 #define LS4_DIR_INPUT             (GPIO_OUTPUT << LS4_PIN)
73 #define LS4_ALT                   LS1_ALT
74

```

```

75  /*                      END OF LED and SWITCH definitions                      */
76
77  /*                      SEVEN SEGMENT DISPLAY Definitions                      */
78  #define SEV_SEG_PORT_BASE_PNT      PORTC      /*
       peripheral port base pointer */
79  #define SEV_SEG_PORT_ID            C          /*
       peripheral port identifier*/
80  #define SEV_SEG_GPIO_BASE_PNT      PTC        /*
       peripheral gpio base pointer */
81
82  #define SEV_SEG_PIT_TIMER_NUMB      0          /* timer number for seven seg
       PIT */
83
84  #define SEGA_PIN                    0          /* Segment
       A*/
85  #define SEGA_DIR_OUTPUT              (GPIO_OUTPUT << SEGA_PIN)
86  #define SEGA_ALT                    0x01u     /* GPIO
       alternative */
87
88  #define SEGB_PIN                    1
89  #define SEGB_DIR_OUTPUT              (GPIO_OUTPUT << SEGB_PIN)
90  #define SEGB_ALT                    SEGA_ALT
91
92  #define SEGC_PIN                    2
93  #define SEGC_DIR_OUTPUT              (GPIO_OUTPUT << SEGC_PIN)
94  #define SEGC_ALT                    SEGA_ALT
95
96  #define SEGD_PIN                    3
97  #define SEGD_DIR_OUTPUT              (GPIO_OUTPUT << SEGD_PIN)
98  #define SEGD_ALT                    SEGA_ALT
99
100 #define SEGE_PIN                    4
101 #define SEGE_DIR_OUTPUT              (GPIO_OUTPUT << SEGE_PIN)
102 #define SEGE_ALT                    SEGA_ALT
103
104 #define SEGF_PIN                    5
105 #define SEGF_DIR_OUTPUT              (GPIO_OUTPUT << SEGF_PIN)
106 #define SEGF_ALT                    SEGA_ALT
107
108 #define SEGG_PIN                    6
109 #define SEGG_DIR_OUTPUT              (GPIO_OUTPUT << SEGG_PIN)
110 #define SEGG_ALT                    SEGA_ALT
111
112 #define SEGDP_PIN                    7

```

```

113 #define SEGDP_DIR_OUTPUT          (GPIO_OUTPUT << SEGDP_PIN)
114 #define SEGDP_ALT                  SEGA_ALT
115
116 #define SEG_DISP1_PIN              13
117 #define SEG_DISP1_DIR_OUTPUT      (GPIO_OUTPUT << SEG_DISP1_PIN)
118 #define SEG_DISP1_ALT              SEGA_ALT
119
120 #define SEG_DISP2_PIN              12
121 #define SEG_DISP2_DIR_OUTPUT      (GPIO_OUTPUT << SEG_DISP2_PIN)
122 #define SEG_DISP2_ALT              SEGA_ALT
123
124 #define SEG_DISP3_PIN              11
125 #define SEG_DISP3_DIR_OUTPUT      (GPIO_OUTPUT << SEG_DISP3_PIN)
126 #define SEG_DISP3_ALT              SEGA_ALT
127
128 #define SEG_DISP4_PIN              10
129 #define SEG_DISP4_DIR_OUTPUT      (GPIO_OUTPUT << SEG_DISP4_PIN)
130 #define SEG_DISP4_ALT              SEGA_ALT
131
132 /*                      END of SEVEN SEGMENT DISPLAY Definitions          */
133
134
135 #endif /* SOURCES_ES670_PERIPHERAL_BOARD_H_ */

```

7.14 ../Sources/Util/util.h

```
1  /* ***** */
2  /* File name:      util.h */
3  /* File description: Header file containing the function/methods */
4  /*                prototypes of util.c */
5  /*                Those delays were tested under the following: */
6  /*                core clock @ 40MHz */
7  /*                bus clock @ 20MHz */
8  /* Author name:    dloubach */
9  /* Creation date:   09jan2015 */
10 /* Revision date:   13abr2016 */
11 /* ***** */
12
13 #ifndef UTIL_H
14 #define UTIL_H
15
16 /**
17  * generates ~ 088 micro sec
18  */
19 void util_genDelay088us(void);
20
21
22 /**
23  * generates ~ 250 micro sec
24  */
25 void util_genDelay250us(void);
26
27
28 /**
29  * generates ~ 1 mili sec
30  */
31 void util_genDelay1ms(void);
32
33
34 /**
35  * generates ~ 10 mili sec
36  */
37 void util_genDelay10ms(void);
38
39
40 #endif /* UTIL_H */
```

7.15 ../Sources/Util/util.c

```
1  /* ***** */
2  /* File name:      util.c */
3  /* File description: This file has a couple of useful functions to */
4  /*                make programming more productive */
5  /* */
6  /*                Remarks: The soft delays consider */
7  /*                core clock @ 40MHz */
8  /*                bus clock @ 20MHz */
9  /* */
10 /* Author name:     dloubach */
11 /* Creation date:    09jan2015 */
12 /* Revision date:    13abr2016 */
13 /* ***** */
14
15 #include "util.h"
16
17 /**
18  * generates ~ 088 micro sec
19  */
20 void util_genDelay088us(void)
21 {
22     char i;
23     for(i=0; i<120; i++)
24     {
25         __asm("NOP");
26         __asm("NOP");
27         __asm("NOP");
28         __asm("NOP");
29         __asm("NOP");
30         __asm("NOP");
31         __asm("NOP");
32         __asm("NOP");
33         __asm("NOP");
34         __asm("NOP");
35         __asm("NOP");
36         __asm("NOP");
37         __asm("NOP");
38         __asm("NOP");
39         __asm("NOP");
40     }
41 }
42
```

```

43
44
45 /**
46  * generates ~ 250 micro sec
47  */
48 void util_genDelay250us(void)
49 {
50     char i;
51     for(i=0; i<120; i++)
52     {
53         __asm("NOP");
54         __asm("NOP");
55         __asm("NOP");
56         __asm("NOP");
57         __asm("NOP");
58         __asm("NOP");
59         __asm("NOP");
60         __asm("NOP");
61         __asm("NOP");
62         __asm("NOP");
63     }
64     util_genDelay088us();
65     util_genDelay088us();
66 }
67
68
69
70 /**
71  * generates ~ 1 mili sec
72  */
73 void util_genDelay1ms(void)
74 {
75     util_genDelay250us();
76     util_genDelay250us();
77     util_genDelay250us();
78     util_genDelay250us();
79 }
80
81
82 /**
83  * generates ~ 10 mili sec
84  */
85 void util_genDelay10ms(void)
86 {

```



```
87     util_genDelay1ms();
88     util_genDelay1ms();
89     util_genDelay1ms();
90     util_genDelay1ms();
91     util_genDelay1ms();
92     util_genDelay1ms();
93     util_genDelay1ms();
94     util_genDelay1ms();
95     util_genDelay1ms();
96     util_genDelay1ms();
97 }
```
