UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA MECÂNICA

ES670 - Projeto de Sistemas Embarcados

Relatório - Projeto Prático Parte 6
Requisito 6: ADC

Nome:	*RA*
Daniel Dello Russo Oliveira	101918
Davi Rodrigues	116581

24 de junho 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*.

Continuamos a complementar nosso diagrama adicionando um bloco responsável pela comunicação serial e um pela interpretação dos comandos referentes ao protocolo do requisito REQ2, adicionamos também um bloco responsável pelo controle do display LCD referente ao REQ3. Blocos responsáveis pelo controle do ventilador e pela medição da velocidade do mesmo, referentes aos requisitos REQ4 e REQ5 foram acrescentados ao diagrama. Novos blocos responsáveis pelo aquecedor e pelo sensor de temperatura do REQ6 foram adicionados.

Integramos também o padrão executivo cíclico cooperativo ao nosso projeto, tornando nosso código mais determinístico.
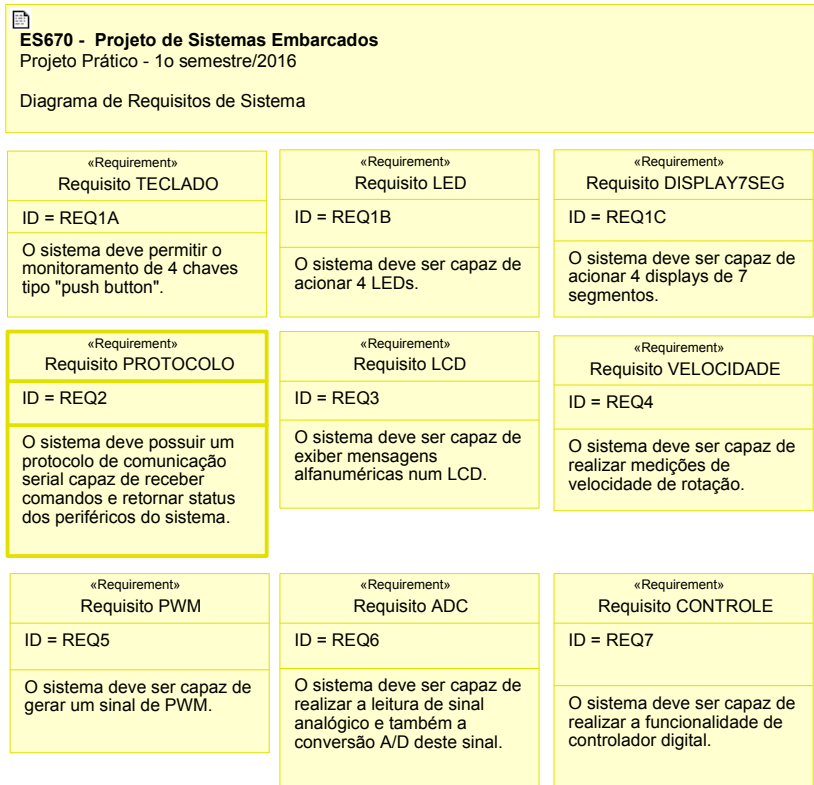
ES670 - Projeto de Sistemas Embarcados
Projeto Prático - 1o semestre/2016

Diagrama de Requisitos de Sistema

«Requirement» Requisito TECLADO	«Requirement» Requisito LED	«Requirement» Requisito DISPLAY7SEG
ID = REQ1A	ID = REQ1B	ID = REQ1C
O sistema deve permitir o monitoramento de 4 chaves tipo "push button".	O sistema deve ser capaz de acionar 4 LEDs.	O sistema deve ser capaz de acionar 4 displays de 7 segmentos.

«Requirement» Requisito PROTOCOLO	«Requirement» Requisito LCD	«Requirement» Requisito VELOCIDADE
ID = REQ2	ID = REQ3	ID = REQ4
O sistema deve possuir um protocolo de comunicação serial capaz de receber comandos e retornar status dos periféricos do sistema.	O sistema deve ser capaz de exibir mensagens alfanuméricas num LCD.	O sistema deve ser capaz de realizar medições de velocidade de rotação.

«Requirement» Requisito PWM	«Requirement» Requisito ADC	«Requirement» Requisito CONTROLE
ID = REQ5	ID = REQ6	ID = REQ7
O sistema deve ser capaz de gerar um sinal de PWM.	O sistema deve ser capaz de realizar a leitura de sinal analógico e também a conversão A/D deste sinal.	O sistema deve ser capaz de realizar a funcionalidade de controlador digital.

Figura 1: Diagrama de requisitos
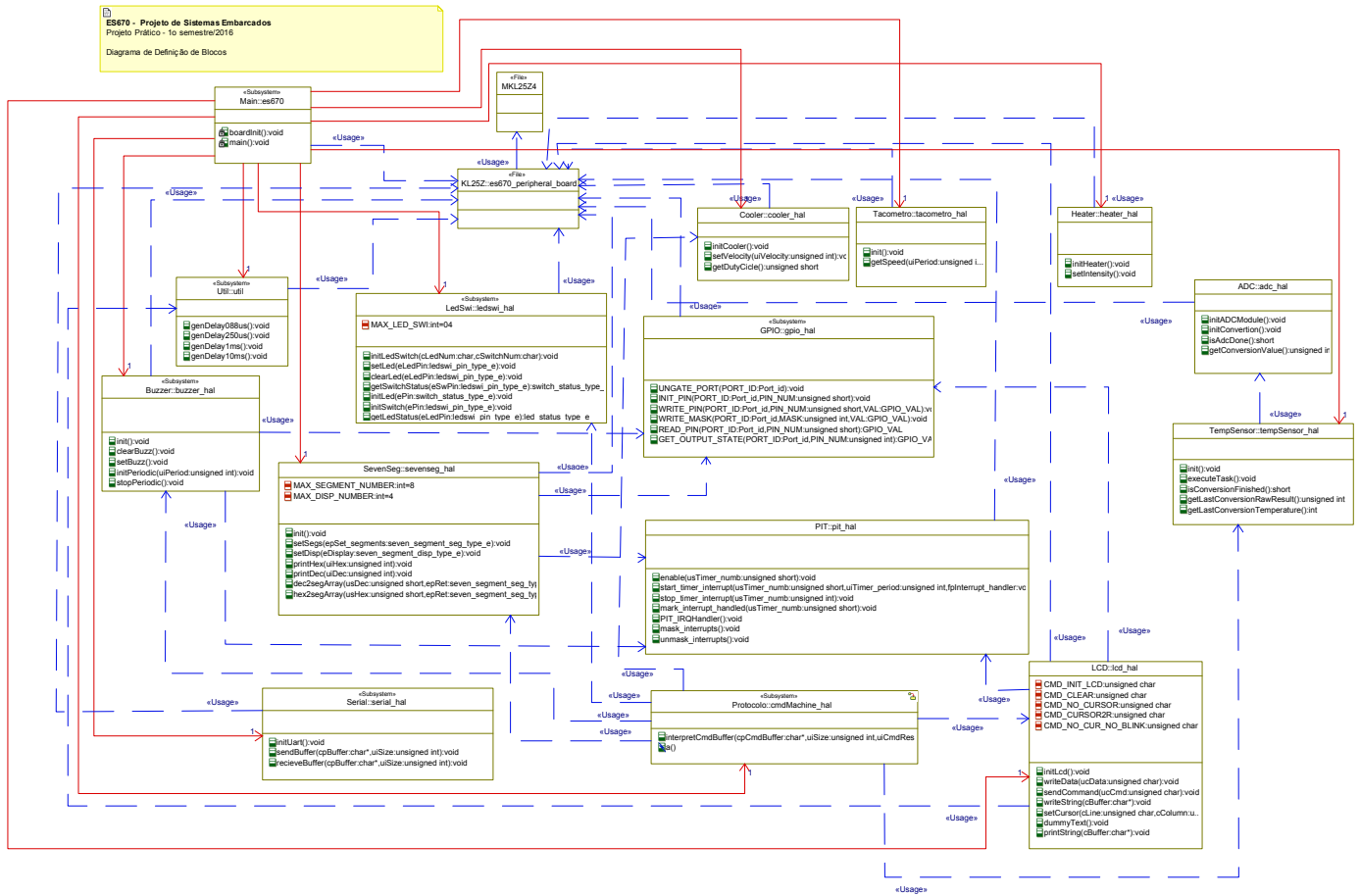
Figura 2: Diagrama de pacotes

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, ler a entrada em um pino e ler o valor de saída que está sendo controlado 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*, marcar uma interrupção como tratada (deve ser feito pelos tratamentos de interrupção), mascarar as interrupções de PIT e desmascarar essas interrupções. 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.

O bloco *ledswi_hal* foi alterado para lidar melhor com a inicialização separada dos pinos como Led e Switch e também foi adicionada uma função para ler o estado atual de um pino configurado como LED.

O bloco *serial_hal* é responsável pela configuração da UART para comunicação serial e pelo envio de recebimento de dados através desse protocolo.

O bloco *lcd_hal* é responsável pelo controle do display LCD, ele segue a implementação passada pelo professor, porém as chamadas as bibliotecas padrão foram substituídas por chamadas ao bloco *GPIO_hal* e manipulações de registradores. Uma função que imprime um texto ao display, após limpar a exibição e resetar a posição do cursor, foi acrescentada.

O bloco *cmdMachine_hal* lida com a interpretação dos comandos e geração da string de resposta. Em nossa implementação, escolhemos tratar uma string já bufferada de commandos ao invés de trabalhar caractere a caractere. A máquina de estados do tratamento dos comandos pode ser vista na figura 4.

O bloco *tacometro_hal* é responsável pela medição da velocidade do ventilador ligado à placa e sua função getCurrentSpeed deve ser chamada de maneira periódica a fim de resetar o contador usado para medir a velocidade.

O bloco *cooler_hal* é responsável pelo controle do ventilador ligado à placa.

Ele conta com uma função de inicialização, uma que controla a velocidade do mesmo, através do módulo PWM, e uma que nos retorna o duty cicle atual do PWM.

O bloco *heater_hal* é responsável pelo controle do aquecedor, possuindo uma função de inicialização e uma para controlar a intensidade do aquecedor.

O bloco *tempSensor_hal* é responsável pelo sensor de temperatura, contando com funções para sua inicialização, para inicializar uma tarefa de leitura, para verificar se uma tarefa de leitura está sendo executada no momento, para ler o resultado "raw"da última tarefa, e para ler a temperatura medida na última tarefa.

Figura 4: Diagrama de máquina de estados para interpretação dos comandos

A máquina de estados recebe um buffer de caracteres por vez, e avalia caracter a caracter percorrendo o buffer. Após cada ação a máquina retorna para o estado "state_idle"para a avaliação da próxima ação do buffer de dados. Sempre que a máquina de estados encontra um caractere inesperado ou sem significado, ela entra no estado "state_err", retornando um erro e descartando todos caracteres até encontrar o próximo espaço em branco. Os outros estados

são acionados através dos seguintes caracteres: 'B' para buzzer, 'S' para switch, 'L' para led, 'D' para o display sete segmentos, 'P' para imprimir no LCD.

No caso do buzzer, após interpretar o caracter 'B', a máquina espera ler até três caracteres numéricos que correspondem ao tempo em que o buzzer permanecerá ativo, retornando para o estado 'statez_idle' após, para avaliar a próxima ação do buffer de entrada.

Para o switch, após a leitura de 'S', a máquina de estados espera a leitura de um caractere numérico, de 1 a 4, que corresponde de qual switch faremos a leitura. Recebemos como retorno 'O' caso o switch esteje acionado, e 'C' caso contrário.

No caso do led, após a leitura de 'L' a máquina de estado espera mais um caractere indicando a ação desejada. Este caractere pode ser 'S' no caso de set, 'R' para read e 'C' para clear. Para todos estes casos, a máquina de estados esperará um caractere numérico indicando em qual dos leds a ação deverá ser executada.

Para o display sete segmentos, após leitura de um 'D' a máquina de estados espera ler até quatro caracteres hexadecimais que serão mostrados no display, retornando para o estado 'state_idle'. No caso do display LCD, após a leitura de um caractere 'P' a máquina de estados lê uma string de até 30 caracteres (sem espaços em branco) e a imprime no display LCD, retornando para o estado 'state_idle'.

O controle do ventilador também foi acrescentado à máquina de estados, ao receber um comando 'C' ela espera ler até quatro caracteres hexadecimais que determinarão o duty-cicle do PWM, retornando para o estado 'state_idle'.

Quando o comando 'T' é lido, o resultado "raw"da última tarefa de leitura é retornado via porta serial e máquina de estados retorna ao estado 'state_idle'.

Após cada ação executada com sucesso, o sistema responderá com 'ACK'.
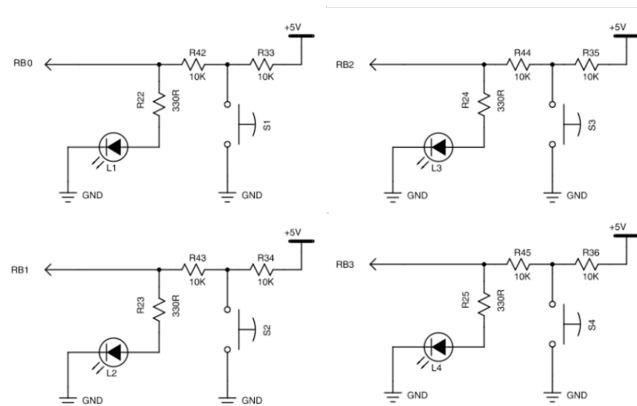
3 Diagramas Esquemáticos
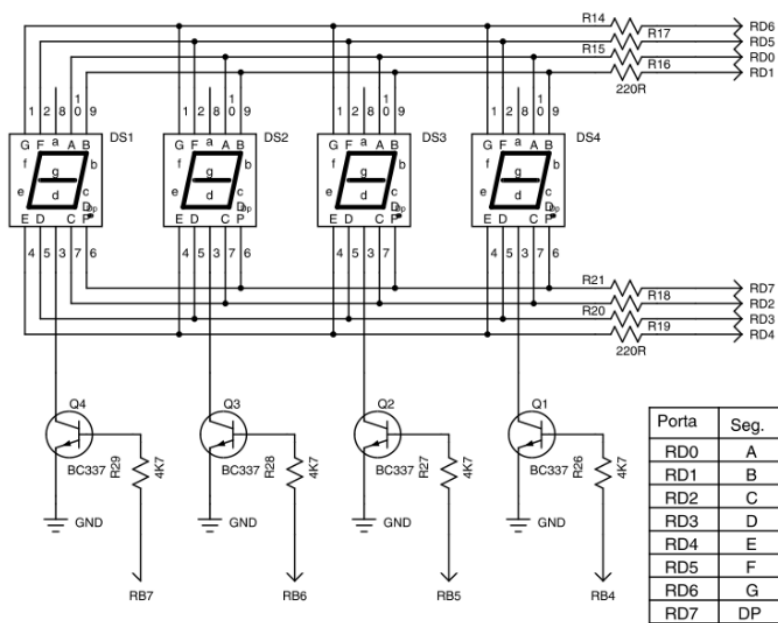

Figura 5: Esquema teclado e LEDs
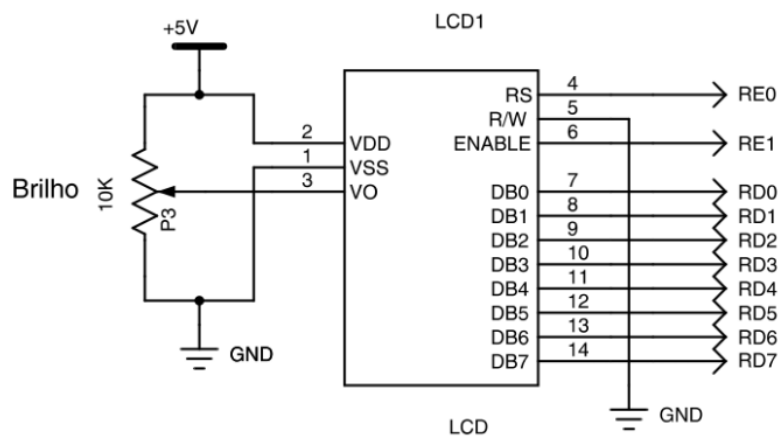

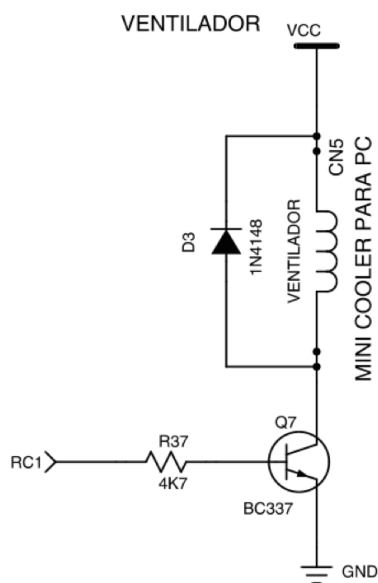Figura 6: Esquema sete segmentos

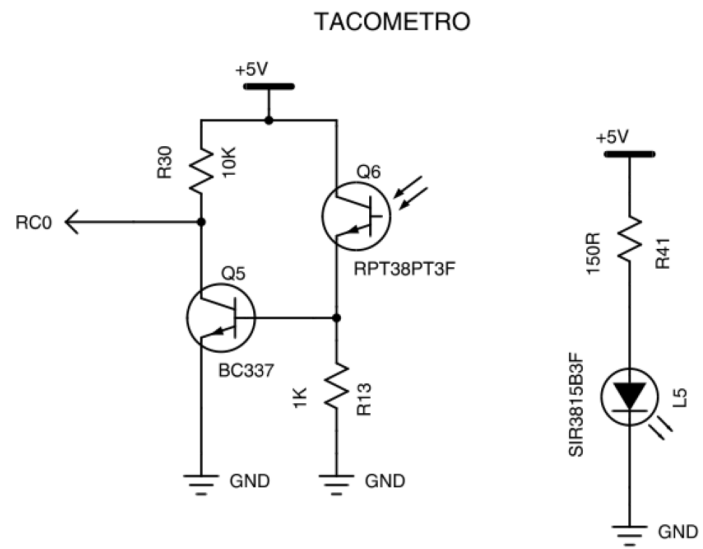Figura 7: Esquema display LCD

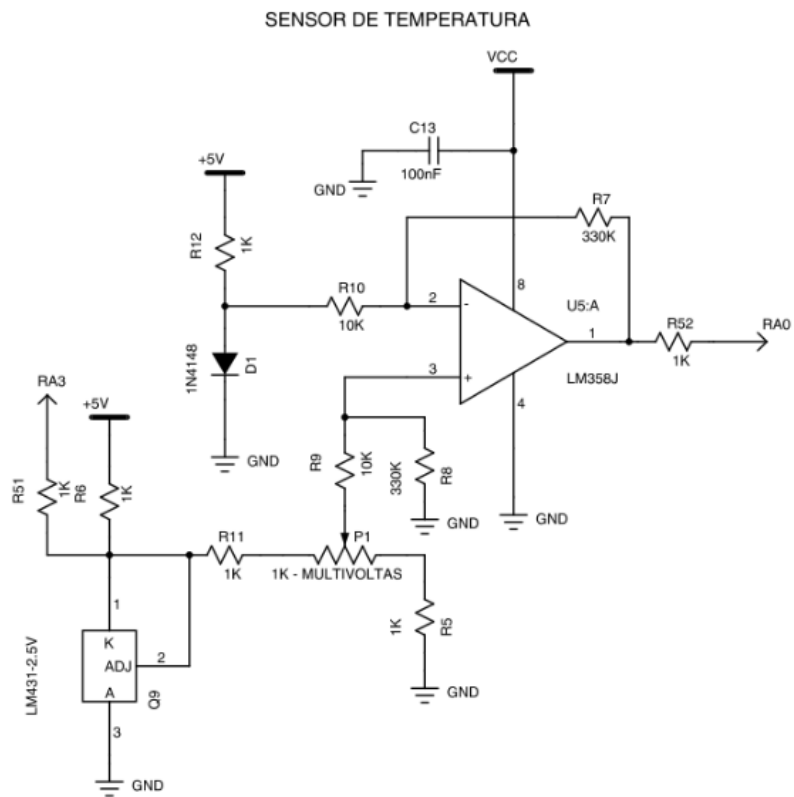Figura 8: Esquema do ventilador

Figura 9: Esquema do tacômetro

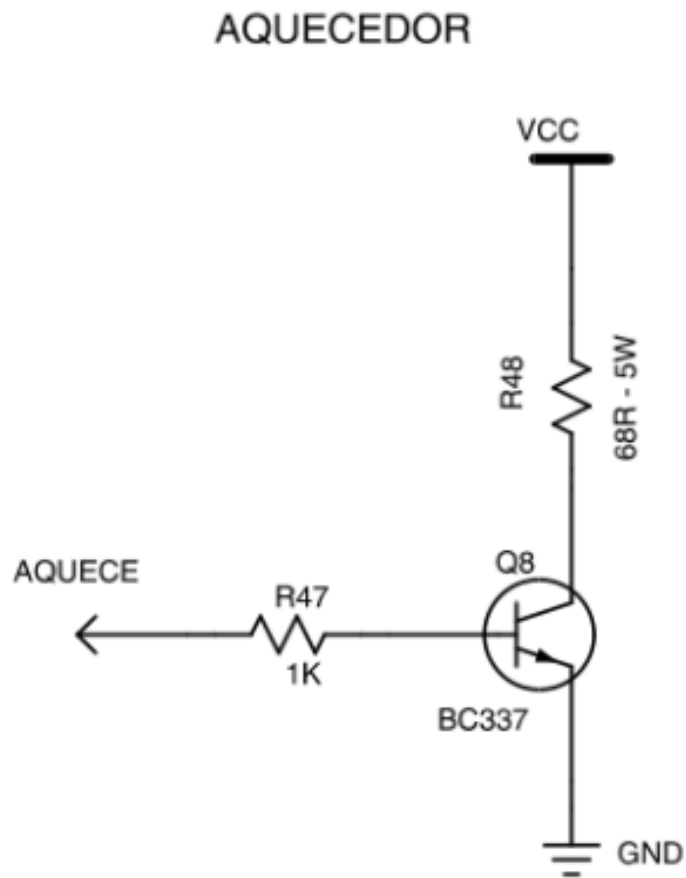Figura 10: Esquema do sensor de temperatura

Figura 11: Esquema do aquecedor

Os pinos apresentados acima, devem ser mapeados para os da placa *FRDM-KL25Z* através do mapeamento apresentado na figura 12

FUNCIONALIDADE	ESPECIFICAÇÃO	PINO	TIPO	PORTA DO ARM		HEADER ARM	SOQUETE PIC
Display 7S	Segmento A	RD0	GPIO	PTC0	PTC0	J1 03	19
Display 7S	Segmento B	RD1	GPIO	PTC1	PTC1/LLWU_P6/RTC_CLKIN	J10 12	20
Display 7S	Segmento C	RD2	GPIO	PTC2	PTC2	J10 10	21
Display 7S	Segmento D	RD3	GPIO	PTC3	PTC3/LLWU_P7	J1 05	22
Display 7S	Segmento E	RD4	GPIO	PTC4	PTC4/LLWU_P8	J1 07	27
Display 7S	Segmento F	RD5	GPIO	PTC5	PTC5/LLWU_P9	J1 09	28
Display 7S	Segmento G	RD6	GPIO	PTC6	PTC6/LLWU_P10	J1 11	29
Display 7S	Segmento DP	RD7	GPIO	PTC7	PTC7	J1 01	30
Display 7S	Display 1	RB7	GPIO	PTC13	PTC13	J2 03	40
Display 7S	Display 2	RB6	GPIO	PTC12	PTC12	J2 01	39
Display 7S	Display 3	RB5	GPIO	PTC11	PTC11	J1 15	38
Display 7S	Display 4	RB4	GPIO	PTC10	PTC10	J1 13	37
LCD	RS	RE0	GPIO	PTC8	PTC8	J1 14	18
LCD	ENABLE	RE1	GPIO	PTC9	PTC9	J1 16	19
LCD	DB0	RD0	GPIO	PTC0	PTC0	J1 03	19
LCD	DB1	RD1	GPIO	PTC1	PTC1/LLWU_P6/RTC_CLKIN	J10 12	20
LCD	DB2	RD2	GPIO	PTC2	PTC2	J10 10	21
LCD	DB3	RD3	GPIO	PTC3	PTC3/LLWU_P7	J1 05	22
LCD	DB4	RD4	GPIO	PTC4	PTC4/LLWU_P8	J1 07	27
LCD	DB5	RD5	GPIO	PTC5	PTC5/LLWU_P9	J1 09	28
LCD	DB6	RD6	GPIO	PTC6	PTC6/LLWU_P10	J1 11	29
LCD	DB7	RD7	GPIO	PTC7	PTC7	J1 01	30
BUZZER		RA5	GPIO	PTD0	FTM0_CH0	J2 06	7
Teclado/LED	S1/LED1	RB0	GPIO	PTA1	PTA1	J1 02	33
Teclado/LED	S2/LED2	RB1	GPIO	PTA2	PTA2	J1 04	34
Teclado/LED	S3/LED3	RB2	GPIO	PTA4	PTA4	J1 10	35
Teclado/LED	S4/LED4	RB3	GPIO	PTA5	PTA5	J1 12	36

Página 1

Figura 12: Mapeamento entre pinos dos esquemáticos e da placa FRDM-KL25Z

Como pode ser visto na figura 6, 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].

O controle do LCD é feito escrevendo o dado desejado nos pinos de dados (DB*), sinalizando se o dado enviado é um comando (Low) ou um caractere a ser escrito (HIGH) através do pino RS e enviando um pulso de duração bem definida no pino ENABLE. Algum dos comandos que podem ser enviados são mostrados na tabela 1.

Tabela 1: Protocolo de comandos para o display LCD

Nome	Código	Descrição
CMD_INIT_LCD	00001111	Inicializa o display LCD
CMD_CLEAR	00000001	Limpa exibição
CMD_NO_CURSOR	00001100	Esconde a posição do cursor
CMD_CURSOR2R	00000110	Cursor a direita do texto
CMD_NO_CUR_NO_BLINK	00111000	Esconde o cursor e não pisca
CMD_MOV_CUR	1L00CCCC	Move cursor para posição [L, X]. L = linha (0, 1) CCCC = coluna(0 a 15)

Para medir a velocidade do ventilador, ligamos o tacômetro a um contador e contamos o número de pulsos dentro de um período pré determinado de tempo.

4 Matriz de Rastreabilidade

A matriz de rastreabilidade apresentada na tabela 2 relaciona cada um dos requisitos com a sua implementação.

Tabela 2: Matriz de Rastreabilidade

ID do Requisito	Implementação
REQ1A	ledswi_hal.c - void ledswi_initLedSwitch(char cLedNum, char cSwitchNum) - switch_status_type_e ledswi_getSwitchStatus(char cSwitchNum) - void ledswi_initSwich(ledswi_pin_type_e ePin)
REQ1B	ledswi_hal.c - void ledswi_initLedSwitch(char cLedNum, char cSwitchNum) - void ledswi_setLed(char cLedNum) - void ledswi_clearLed(char cLedNum) - void ledswi_initLed(ledswi_pin_type_e ePin) - led_status_type_e ledswi_getLedStatus(ledswi_pin_type_e eLedPin)
REQ1C	sevenseg_hal.c - void sevenseg_init(void) - void sevenseg_setSegs(seven_segment_seg_type_e* epSet_segments) - void sevenseg_setDisp(seven_segment_disp_type_e eDisplay) - void sevenseg_printDec(unsigned int uiDec) - void sevenseg_printHex(unsigned int uiHex)
REQ2	serial_hal.c - void serial_initUart(void) - void serial_sendBuffer(char *cpBuffer, unsigned int uiSize) - int serial_recieveBuffer(char *cpBuffer, unsigned int uiSize) cmdMachine_hal.c - void cmdmachine_interpretCmdBuffer(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes) - int handleError(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes) - int handleBuzzer(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes); - int getBuzzMs(char *cpCmdBuffer) - unsigned int handleSwitch(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes) - unsigned int handleLed(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes) - ledswi_pin_type_e parseLedNum(char cLedInput) - unsigned int handleIdle(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes) - unsigned int handleLCD(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes) - unsigned int handleSevenSeg(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes) - int getSevenSegHex(char *cpCmdBuffer)
REQ3	lcd_hal.c - void lcd_initLcd(void) - void lcd_writeData(unsigned char ucData) - void lcd_sendCommand(unsigned char ucCmd) - void lcd_writeString(const char *cBuffer) - void lcd_printString(const char *cBuffer) - void lcd_setCursor(unsigned char cLine, unsigned char cColumn) - void lcd_dummyText(void)
REQ4	tacometro_hal.c - void tacometro_init(void) - unsigned int tacometro_getSpeed(unsigned int uiPeriod)
REQ5	cooler_hal.c - void cooler_initCooler(void) - void cooler_setVelocity(uint16_t uiVelocity) - unsigned short cooler_getDutyCicle()
REQ6	tempSensor_hal.c - void tempSensor_init() - void tempSensor_executeTask() - short tempSensor_isConversionFinished() - uint16_t tempSensor_getLastConversionRawResult() - int tempSensor_getLastConversionTemperature()

5 Notas

5.1 Greve e Piquetes

Devido à greve quase não conseguimos acesso ao laboratório e não conseguimos mostrar que nosso projeto está funcional (ou testá-lo). Acreditamos que a postura adotada pelo professor de continuar cobrando laboratórios e projetos que estão acoplados ao acesso ao hardware, sem levar em consideração que esse acesso está na maior parte do tempo interditado, poderia ser revista. Concordando ou não com a greve, entendendo ou não que os grevistas faltaram com respeito com os professores, atrelar a aprovação dos alunos a sua capacidade de driblar o piquete e ao fato deles não aderirem à greve nos parece uma atitude questionável, que prende muitos dos estudantes (vários dos quais trabalham e não tem tempo para ficar esperando a boa vontade dos piqueteiros para entrar no laboratório) no meio de uma briga de poderes entre dois partidos que estão tentando provar seu ponto. Não apoiamos os meios adotados por alguns grevistas, mas acreditamos em compromissos e, enquanto nenhum dos partidos estiver disposto a fazer nenhum, pedimos mais compreensão por parte do professor, que se mostrou bastante inflexível quanto as datas de entrega.

5.2 Comunicação Serial e Executivo Cíclico

Até então quase todo o processamento do nosso sistema estava atrelado à execução de um comando vindo por comunicação serial, logo utilizávamos chamadas que bloqueavam nossa thread principal no aguardo do próximo comando. Agora não podemos fazer mais isso e devemos incluir um pooling da porta serial durante todo o período ocioso do nosso loop principal para garantir que não vamos perder informações vindas por comunicação serial. Uma outra alternativa (que julgamos desnecessária dada a periodicidade do nosso sistema) seria a implementação de interrupções da UART.

5.3 PWM

Uma coisa que deve ser levada em consideração quando configuramos o PWM é que sempre que executamos uma instrução de escrita para alterar a configuração do módulo de PWM o timer deve estar desabilitado e essa instrução leva um tempo para que a escrita seja efetuada, logo é recomendado escrever no registrador em uma única instrução e bit a bit.

16

5.4 LCD e Interrupções

Para escrever no LCD precisamos gerar um pulso no seu pino de enable. Para garantir que esse pulso terá a duração certa, foi preciso mascarar as interrupções durante esse pulso. Como as interrupções do display tem um período de aproximadamente 3ms e o pulso do LCD tem a mesma duração encontramos várias dificuldades ao controlar os dois sem mascarar as interrupções pela duração do pulso, logo adaptamos nosso código. Quando atualizamos as informações do painel LCD de maneira regular, notamos um leve efeito de "flicker"nos displays 7 segmentos.

5.5 Leitura do Estado dos LEDs

Como uma extensão ao protocolo proposto no roteiro [1], acrescentamos o comando *LRd* onde *d* é um digito de 1 a 4, que nos retorna o estado atual do LED referenciado. Para recuperar esse estado devemos saber qual valor está sendo controlado para um pino de saída digital, o que é feito lendo o registrador GPIOx_PDOR.

5.6 Interrupções do Buzzer e mudança de Baud Rate

Ao utilizar interrupções para cuidar do acionamento e controle do Buzzer, conseguímos executar essa funcionalidade de maneira assíncrona, garantindo que essa funcionalidade não vai interferir com ou ser interferida pelas outras. Isso é notavelmente importante com a comunicação serial, uma vez que a velocidade de leitura é crítica e se essa for alterada, corremos o risco de perder informações ou alterar o tempo de execução de nosso ciclo.

5.7 Bufferização dos comandos

Para diminuir a chance de perder informações que são enviadas por serial, escolhemos ler uma série de caracteres e armazená-los em um buffer antes de interpretá-los, ao invés de ler e interpretar caractere a caractere. Interrompemos esse processo de leitura no caso de buffer overflow ou quando encontramos uma quebra de linha. Interpretar uma série de comandos bufferizados introduziu uma dificuldade extra ao projeto, pois agora nossa máquina de estados tem que levar em consideração que um comando pode estar errado e o próximo não, logo o primeiro deve ser ignorado e o segundo executado.

5.8 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-Kl25Z*, 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.

5.9 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. Nota: Documentação disponível no site arm.com, na referência do registrador Arm-M0+ sobre o NVIC [6] encontramos a seguinte frase: "See the ARMv6-M Architecture Reference Manual for more information about the NVIC registers and their addresses, access types, and reset values.". Já na página ARMv6-M Architecture Reference Manual [7] somos informados que esse manual só está disponível em formato PDF e para clientes registrados.

5.10 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.

5.11 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.

5.12 Rhapsody Modeler e Wine

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*: *mfc42, comctl32, msvcirt* e *riched20*.
Ainda temos um problema com o Rhapsody quando tentamos editar textos dentro de um estado, por isso nossa máquina de estados só executa sua lógica nas transições.

5.13 Outros

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`

[6] Cortex-M0+ Technical Reference Manual `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0413c/Cihfihfe.html`

[7] ARMv6-M Architecture Reference Manual `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0413c/Cihfihfe.html`

7 Apêndice

Listagem dos códigos fonte:

7.1 ../Sources/ADC/adc.c

```
1  /*
2   * File name: adc.c
3   *
4   *  Created on: 06/06/2016
5   *       Author: msoliveira
6   */
7
8  #include "KL25Z/es670_peripheral_board.h"
9
10 #define ADC0_SC1A_COCO (ADC0_SC1A >> 7)
11
12 /**
13  *  Configure ADC module
14  */
15 void adc_initADCModule(void) {
16   /* un-gate port clock*/
17   SIM_SCGC6 |= SIM_SCGC6_ADC0(CGC_CLOCK_ENABLED); //Enable clock for ADC
18
19     /* un-gate port clock*/
20     SIM_SCGC5 |= SIM_SCGC5_PORTE(CGC_CLOCK_ENABLED);
21
22     /* set pin as ADC In */
23     PORT_PCR_REG(ADC_PORT_BASE_PNT, ADC_VOLT_PIN) =
       PORT_PCR_MUX(ADC_VOLT_PIN_MUX_ALT);   //Voltage Sensor
24     PORT_PCR_REG(ADC_PORT_BASE_PNT, ADC_TEMP_PIN) =
       PORT_PCR_MUX(ADC_TEMP_PIN_MUX_ALT);   //Temperature Sensor
25
26     ADC_WR_CFG1(ADC_NUM_BASE_PNT,
27         (
28             ADC_CFG1_ADICLK(0x1)      // bus/2 clock selection
29             | ADC_CFG1_MODE(0x3)       // 16-bit Conversion mode selection
30           | ADC_CFG1_ADLSMP(0x0)    // Short sample time configuration
31           | ADC_CFG1_ADIV(0x0)    // Clock Divide Select (Divide by 1)
32           | ADC_CFG1_ADLPC(0x1)   // Normal power Configuration
33       )
34   );
35
```

```
36    ADC_WR_SC2 ( ADC_NUM_BASE_PNT ,
37        (
38            ADC_SC2_REFSEL (0x0)        //external pins
39          | ADC_SC2_DMAEN (0x0)       //dma disabled
40          | ADC_SC2_ACFE (0x0)        //compare disabled
41          | ADC_SC2_ADTRG (0x0)       //sw trigger
42        )
43    );
44
45    ADC_WR_CFG2 ( ADC_NUM_BASE_PNT ,
46        (
47            ADC_CFG2_ADLSTS (0)        // default time
48          | ADC_CFG2_ADHSC (0)        // normal conversion sequence
49          | ADC_CFG2_ADACKEN (0)        // adack clock disabled
50          | ADC_CFG2_MUXSEL ( ADC_MUX_IDX )  // 'a' channels
51        )
52    );
53 }
54
55
56 /**
57  * Init conversion from A to D
58  */
59 void adc_initConvertion ( void ) {
60    ADC_WR_SC1 ( ADC_NUM_BASE_PNT , ADC_MUX_IDX ,
61        (
62        ADC_SC1_ADCH ( ADC_CHANNEL_SEL )        //Channel 4 - SE4A
63        | ADC_SC1_DIFF (0)               //single ended
64        | ADC_SC1_AIEN (0)              //interrupt disabled
65        )
66    );
67 }
68
69 /**
70  * Check if conversion is done
71  * @return 1 if done , 0 otherwise
72  */
73 short adc_isAdcDone ( void )
74 {
75    return ADC_BRD_SC1_COCO ( ADC_NUM_BASE_PNT , ADC_MUX_IDX );
76 }
77
78 /**
79  * Retrieve converted value
```

```
80   * @return The conversion result
81   */
82  uint16_t adc_getConversionValue(void)
83  {
84    return ADC_RD_R(ADC_NUM_BASE_PNT, ADC_MUX_IDX); // return the register value that
         keeps the result of convertion
85  }
```

7.2 ../Sources/ADC/adc.h

```
1  /*
2   * File name: adc.h
3   *
4   *   Created on: 06/06/2016
5   *       Author: msoliveira
6   */
7
8  #include <stdint.h>
9
10 /**
11  *  Configure ADC module
12  */
13 void adc_initADCModule(void);
14
15 /**
16  * Init conversion from A to D
17  */
18 void adc_initConvertion(void);
19
20 /**
21  * Check if conversion is done
22  * @return 1 if done, 0 otherwise
23  */
24 short adc_isAdcDone(void);
25
26 /**
27  * Retrieve converted value
28  * @return The conversion result
29  */
30 uint16_t adc_getConversionValue(void);
```

7.3 ../Sources/Buzzer/buzzer_hal.c

```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 static volatile int interrupt_counter = -1;
16
17 /**
18  * Initialize the buzzer device
19  */
20 void buzzer_init(void)
21 {
22   GPIO_UNGATE_PORT(BUZZER_PORT_ID);
23   GPIO_INIT_PIN(BUZZER_PORT_ID, BUZZER_PIN, GPIO_OUTPUT);
24   pit_enable();
25 }
26
27
28
29 /**
30  * Clear the buzzer
31  */
32 void buzzer_clearBuzz(void)
33 {
34     GPIO_WRITE_PIN(BUZZER_PORT_ID, BUZZER_PIN, GPIO_LOW);
35 }
36
37
38
39 /**
40  * Set the buzzer
41  */
42 void buzzer_setBuzz(void)
```

```
43 {
44   GPIO_WRITE_PIN(BUZZER_PORT_ID, BUZZER_PIN, GPIO_HIGH);
45 }
46
47 /**
48  * Handler for buzzer interruptions
49  */
50 void _buzzer_interrupt_handler(void){
51   static volatile unsigned short usBusOn = 0;
52   if(!usBusOn){
53     buzzer_setBuzz();
54   }else{
55     buzzer_clearBuzz();
56   }
57   usBusOn = !usBusOn;
58   //If not in undefined mode decrease counter
59   if(interrupt_counter > 0){
60     interrupt_counter--;
61   }
62   //Stop interruptions when signal duration is finished
63   if(!interrupt_counter){
64     buzzer_stopPeriodic();
65   }
66   //Mark interruption as handled
67   pit_mark_interrupt_handled(BUZZER_PIT_TIMER_NUMB);
68 }
69
70 /**
71  * Starts the buzzer with the specified period
72  *
73  * @param uiBuzzFreq_hz The frequency of the buzzer signal, in Hz
74  * @param uiDuration_ms How many milliseconds the buzzer should be producing sound
75  *                 if 0 buzzer will stay on indeterminaly.
76  */
77 void buzzer_initPeriodic(unsigned int uiBuzzFreq_hz, unsigned int uiDuration_ms){
78   unsigned int uiPeriod_us = 500000/uiBuzzFreq_hz;  /* 50% duty cycle */
79   if(uiDuration_ms > 0){
80     interrupt_counter = uiDuration_ms/(uiPeriod_us/1000);
81   }else{
82     interrupt_counter = -1;
83   }
84   //Init timer 1
85   pit_start_timer_interrupt(BUZZER_PIT_TIMER_NUMB, uiPeriod_us,
86     &_buzzer_interrupt_handler);
```

```
86 }
87
88 /**
89  * Stops any periodic buzzer signal
90  */
91 void buzzer_stopPeriodic( void ){
92   interrupt_counter = -1;
93   pit_stop_timer_interrupt ( BUZZER_PIT_TIMER_NUMB );
94   buzzer_clearBuzz ();
95 }
```

7.4   ../Sources/Buzzer/buzzer_hal.h

```c
/* ***************************************************************** */
/* File name:        buzzer_hal.h                                   */
/* File description: Header file containing the functions/methods   */
/*                   interfaces for handling BUZZER from the        */
/*                   peripheral board                               */
/* Author name:      dloubach                                       */
/* Creation date:    12jan2016                                      */
/* Revision date:    13abr2016                                      */
/* ***************************************************************** */

#ifndef SOURCES_BUZZER_HAL_H_
#define SOURCES_BUZZER_HAL_H_

/**
 * Initialize the buzzer device
 */
void buzzer_init(void);


/**
 * Clear the buzzer
 */
void buzzer_clearBuzz(void);


/**
 * Set the buzzer
 */
void buzzer_setBuzz(void);

/**
 * Starts the buzzer with the specified period
 *
 * @param uiBuzzFreq_hz The frequency of the buzzer signal, in Hz
 * @param uiDuration_ms How many milliseconds the buzzer should be producing sound
 *              if 0 buzzer will stay on indeterminaly.
 */
void  buzzer_initPeriodic(unsigned int uiBuzzFreq_hz, unsigned int uiDuration_ms);

/**
 * Stops any periodic buzzer signal
 */
```

```
43  void buzzer_stopPeriodic(void);
44
45
46  #endif /* SOURCES_BUZZER_HAL_H_ */
```

7.5 ../Sources/Cooler/cooler_hal.c

```c
/* ***************************************************************** */
/* File name:        cooler_hal.c                                   */
/* File description: File containing the functions/methods          */
/*                   for cooler control                             */
/* Author name:      ddello                                         */
/* Creation date:    13maio2016                                     */
/* Revision date:    13maio2016                                     */
/* ***************************************************************** */

#include "cooler_hal.h"
#include "KL25Z/es670_peripheral_board.h"
#include "GPIO/gpio_hal.h"

/**
 * Initialize the cooler module
 */
void cooler_initCooler(void){
  /* Init cooler pin */
  SIM_SCGC5 |= SIM_SCGC5_PORTA(1u);
  PORT_PCR_REG(COOLER_PORT_BASE_PNT , COOLER_PIN) = PORT_PCR_MUX(COOLER_PIN_MUX_ALT);

  /* INIT TPM */
  //Init TPM Clock source as OSCERCLOCK
  SIM_WR_SOPT2_TPMSRC(SIM,0x2);
  //Enable clock to TPM1
  SIM_SCGC6 |= SIM_SCGC6_TPM1(0x1);
  //PWM mode
  TPM_WR_SC_CPWMS(COOLER_TPM_BASE_PNT, 0x0);  //Edge Aligned
  /* Channel mode configuration should be done with the channel disabled and need
     acknowledge*/
  TPM_WR_CnSC(COOLER_TPM_BASE_PNT, COOLER_TPM_CHANNEL_INDEX, TPM_CnSC_ELSB(1) |
     TPM_CnSC_MSB(1));
    /* Wait till mode change is acknowledged */
    while (!
        (
        TPM_RD_CnSC(COOLER_TPM_BASE_PNT, COOLER_TPM_CHANNEL_INDEX)
        & (TPM_CnSC_ELSB(1) | TPM_CnSC_MSB(1))
      )
      );
  //Same period as OSCRCLK
  TPM_WR_MOD(COOLER_TPM_BASE_PNT, 0xFFFE);
  //Duty cycle
```

```
41    TPM_WR_CnV_VAL(COOLER_TPM_BASE_PNT, COOLER_TPM_CHANNEL_INDEX, 0);
42    //Prescale 1:1
43    TPM_WR_SC_PS(COOLER_TPM_BASE_PNT, 0x0);
44    //Select LTPM Clock Source for TPM module
45    TPM_WR_SC_CMOD(COOLER_TPM_BASE_PNT, 0x1);
46 }
47
48
49 /**
50  * Set the cooler velocity
51  *
52  * @param uiVelocity Cooler velocity from 0 (stopped) to 0xFFFF (max)
53  */
54 void cooler_setVelocity(uint16_t uiVelocity){
55    TPM_WR_CnV_VAL(COOLER_TPM_BASE_PNT, COOLER_TPM_CHANNEL_INDEX, uiVelocity);
56 }
57
58 /**
59  * Get the cooler PWM duty cicle in %.
60  *
61  * @return Duty Cicle (0 to 100)
62  */
63 unsigned short cooler_getDutyCicle(){
64    return (TPM_RD_CnV_VAL(COOLER_TPM_BASE_PNT, COOLER_TPM_CHANNEL_INDEX)*100)/0xFFFF;
65 }
```

7.6 ../Sources/Cooler/cooler_hal.h

```c
/* ***************************************************************** */
/* File name:        cooler_hal.h                                    */
/* File description: Header file containing the functions/methods    */
/*                   interfaces for cooler control                   */
/* Author name:      ddello                                          */
/* Creation date:    13maio2016                                      */
/* Revision date:    13maio2016                                      */
/* ***************************************************************** */

#ifndef SOURCES_COOLER_HAL_H_
#define SOURCES_COOLER_HAL_H_

#include <stdint.h>

/**
 * Initialize the cooler module
 */
void cooler_initCooler(void);


/**
 * Set the cooler velocity
 *
 * @param uiVelocity Cooler velocity from 0 (stopped) to 0xFFFF (max)
 */
void cooler_setVelocity(uint16_t uiVelocity);

/**
 * Get the cooler PWM duty cicle in %.
 *
 * @return Duty Cicle (0 to 100)
 */
unsigned short cooler_getDutyCicle();

#endif /* SOURCES_COOLER_HAL_H_ */
```

7.7 ../Sources/fsl_debug_console.c

```
1  /*
2   * Copyright (c) 2013 - 2014, Freescale Semiconductor, Inc.
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without modification,
6   * are permitted provided that the following conditions are met:
7   *
8   * o Redistributions of source code must retain the above copyright notice, this list
9   *   of conditions and the following disclaimer.
10  *
11  * o Redistributions in binary form must reproduce the above copyright notice, this
12  *   list of conditions and the following disclaimer in the documentation and/or
13  *   other materials provided with the distribution.
14  *
15  * o Neither the name of Freescale Semiconductor, Inc. nor the names of its
16  *   contributors may be used to endorse or promote products derived from this
17  *   software without specific prior written permission.
18  *
19  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
20  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
21  * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
22  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
23  * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
24  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
25  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
26  * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
27  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
28  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
29  */
30
31  #include <stdarg.h>
32  #include <stdio.h>
33  #include <stdlib.h>
34  #include "fsl_device_registers.h"
35  #include "fsl_debug_console.h"
36  #if defined(UART_INSTANCE_COUNT)
37  #include "fsl_uart_hal.h"
38  #endif
39  #if defined(LPUART_INSTANCE_COUNT)
40  #include "fsl_lpuart_hal.h"
41  #endif
42  #if defined(UART0_INSTANCE_COUNT)
```

```
43 #include "fsl_lpsci_hal.h"
44 #endif
45 #include "fsl_clock_manager.h"
46 #include "fsl_os_abstraction.h"
47 #include "print_scan.h"
48
49 #if (defined(USB_INSTANCE_COUNT) &&  (defined(BOARD_USE_VIRTUALCOM)))
50    #include "usb_device_config.h"
51    #include "usb.h"
52    #include "usb_device_stack_interface.h"
53    #include "usb_descriptor.h"
54    #include "virtual_com.h"
55 #endif
56
57 extern uint32_t g_app_handle;
58 #if __ICCARM__
59 #include <yfuns.h>
60 #endif
61
62 static int debug_putc(int ch, void* stream);
63
64 /********************************************************************************
65  * Definitions
66  ********************************************************************************/
67
68 /*! @brief Operation functions definiations for debug console. */
69 typedef struct DebugConsoleOperationFunctions {
70     union {
71         void (* Send)(void *base, const uint8_t *buf, uint32_t count);
72 #if defined(UART_INSTANCE_COUNT)
73         void (* UART_Send)(UART_Type *base, const uint8_t *buf, uint32_t count);
74 #endif
75 #if defined(LPUART_INSTANCE_COUNT)
76         void (* LPUART_Send)(LPUART_Type* base, const uint8_t *buf, uint32_t count);
77 #endif
78 #if defined(UART0_INSTANCE_COUNT)
79         void (* UART0_Send)(UART0_Type* base, const uint8_t *buf, uint32_t count);
80 #endif
81 #if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
82         void (* USB_Send)(uint32_t base, const uint8_t *buf, uint32_t count);
83 #endif
84     } tx_union;
85     union{
86         void (* Receive)(void *base, uint8_t *buf, uint32_t count);
```

```c
87  #if defined(UART_INSTANCE_COUNT)
88          uart_status_t (* UART_Receive)(UART_Type *base, uint8_t *buf, uint32_t count);
89  #endif
90  #if defined(LPUART_INSTANCE_COUNT)
91          lpuart_status_t (* LPUART_Receive)(LPUART_Type* base, uint8_t *buf, uint32_t
        count);
92  #endif
93  #if defined(UART0_INSTANCE_COUNT)
94          lpsci_status_t (* UART0_Receive)(UART0_Type* base, uint8_t *buf, uint32_t
        count);
95  #endif
96  #if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))
97          usb_status_t (* USB_Receive)(uint32_t base, uint8_t *buf, uint32_t count);
98  #endif
99
100     } rx_union;
101 } debug_console_ops_t;
102
103 /*! @brief State structure storing debug console. */
104 typedef struct DebugConsoleState {
105     debug_console_device_type_t type;/*<! Indicator telling whether the debug console
        is inited. */
106     uint8_t instance;                /*<! Instance number indicator. */
107     void*   base;                    /*<! Base of the IP register. */
108     debug_console_ops_t ops;         /*<! Operation function pointers for debug uart
        operations. */
109 } debug_console_state_t;
110
111 /*******************************************************************************
112  * Variables
113  ******************************************************************************/
114 /*! @brief Debug UART state information.*/
115 static debug_console_state_t s_debugConsole;
116
117 /*******************************************************************************
118  * Code
119  ******************************************************************************/
120 /* See fsl_debug_console.h for documentation of this function.*/
121 debug_console_status_t DbgConsole_Init(
122         uint32_t uartInstance, uint32_t baudRate, debug_console_device_type_t device)
123 {
124     if (s_debugConsole.type != kDebugConsoleNone)
125     {
126         return kStatus_DEBUGCONSOLE_Failed;
```

```
127        }
128
129        /* Set debug console to initialized to avoid duplicated init operation.*/
130        s_debugConsole.type = device;
131        s_debugConsole.instance = uartInstance;
132
133        /* Switch between different device. */
134        switch (device)
135        {
136  #if (defined(USB_INSTANCE_COUNT) && defined(BOARD_USE_VIRTUALCOM))  /*&& defined()*/
137            case kDebugConsoleUSBCDC:
138                {
139                    VirtualCom_Init();
140                    s_debugConsole.base = (void*)g_app_handle;
141                    s_debugConsole.ops.tx_union.USB_Send = VirtualCom_SendDataBlocking;
142                    s_debugConsole.ops.rx_union.USB_Receive =
        VirtualCom_ReceiveDataBlocking;
143                }
144                break;
145  #endif
146  #if defined(UART_INSTANCE_COUNT)
147            case kDebugConsoleUART:
148                {
149                    UART_Type * g_Base[UART_INSTANCE_COUNT] = UART_BASE_PTRS;
150                    UART_Type * base = g_Base[uartInstance];
151                    uint32_t uartSourceClock;
152
153                    s_debugConsole.base = base;
154                    CLOCK_SYS_EnableUartClock(uartInstance);
155
156                    /* UART clock source is either system or bus clock depending on
        instance */
157                    uartSourceClock = CLOCK_SYS_GetUartFreq(uartInstance);
158
159                    /* Initialize UART baud rate, bit count, parity and stop bit. */
160                    UART_HAL_SetBaudRate(base, uartSourceClock, baudRate);
161                    UART_HAL_SetBitCountPerChar(base, kUart8BitsPerChar);
162                    UART_HAL_SetParityMode(base, kUartParityDisabled);
163  #if FSL_FEATURE_UART_HAS_STOP_BIT_CONFIG_SUPPORT
164                    UART_HAL_SetStopBitCount(base, kUartOneStopBit);
165  #endif
166
167                    /* Finally, enable the UART transmitter and receiver*/
168                    UART_HAL_EnableTransmitter(base);
```

```
169                     UART_HAL_EnableReceiver ( base );
170
171                     /* Set the funciton pointer for send and receive for this kind of
       device . */
172                     s_debugConsole . ops . tx_union . UART_Send = UART_HAL_SendDataPolling ;
173                     s_debugConsole . ops . rx_union . UART_Receive = UART_HAL_ReceiveDataPolling ;
174                 }
175                 break ;
176  #endif
177  #if defined ( UART0_INSTANCE_COUNT )
178         case kDebugConsoleLPSCI :
179               {
180                     /* Declare config sturcuture to initialize a uart instance . */
181                     UART0_Type * g_Base [ UART0_INSTANCE_COUNT ] = UART0_BASE_PTRS ;
182                     UART0_Type * base = g_Base [ uartInstance ];
183                     uint32_t uartSourceClock ;
184
185                     s_debugConsole . base = base ;
186                     CLOCK_SYS_EnableLpsciClock ( uartInstance );
187
188                     uartSourceClock = CLOCK_SYS_GetLpsciFreq ( uartInstance );
189
190                     /* Initialize LPSCI baud rate , bit count , parity and stop bit . */
191                     LPSCI_HAL_SetBaudRate ( base , uartSourceClock , baudRate );
192                     LPSCI_HAL_SetBitCountPerChar ( base , kLpsci8BitsPerChar );
193                     LPSCI_HAL_SetParityMode ( base , kLpsciParityDisabled );
194  #if FSL_FEATURE_LPSCI_HAS_STOP_BIT_CONFIG_SUPPORT
195                     LPSCI_HAL_SetStopBitCount ( base , kLpsciOneStopBit );
196  #endif
197
198                     /* Finally , enable the LPSCI transmitter and receiver */
199                     LPSCI_HAL_EnableTransmitter ( base );
200                     LPSCI_HAL_EnableReceiver ( base );
201
202                     /* Set the funciton pointer for send and receive for this kind of
       device . */
203                     s_debugConsole . ops . tx_union . UART0_Send = LPSCI_HAL_SendDataPolling ;
204                     s_debugConsole . ops . rx_union . UART0_Receive =
       LPSCI_HAL_ReceiveDataPolling ;
205                 }
206                 break ;
207  #endif
208  #if defined ( LPUART_INSTANCE_COUNT )
209         case kDebugConsoleLPUART :
```

```
210              {
211                  LPUART_Type* g_Base[LPUART_INSTANCE_COUNT] = LPUART_BASE_PTRS;
212                  LPUART_Type* base = g_Base[uartInstance];
213                  uint32_t lpuartSourceClock;
214
215                  s_debugConsole.base = base;
216                  CLOCK_SYS_EnableLpuartClock(uartInstance);
217
218                  /* LPUART clock source is either system or bus clock depending on
     instance */
219                  lpuartSourceClock = CLOCK_SYS_GetLpuartFreq(uartInstance);
220
221                  /* initialize the parameters of the LPUART config structure with
     desired data */
222                  LPUART_HAL_SetBaudRate(base, lpuartSourceClock, baudRate);
223                  LPUART_HAL_SetBitCountPerChar(base, kLpuart8BitsPerChar);
224                  LPUART_HAL_SetParityMode(base, kLpuartParityDisabled);
225                  LPUART_HAL_SetStopBitCount(base, kLpuartOneStopBit);
226
227                  /* finally, enable the LPUART transmitter and receiver */
228                  LPUART_HAL_SetTransmitterCmd(base, true);
229                  LPUART_HAL_SetReceiverCmd(base, true);
230
231                  /* Set the funciton pointer for send and receive for this kind of
     device. */
232                  s_debugConsole.ops.tx_union.LPUART_Send = LPUART_HAL_SendDataPolling;
233                  s_debugConsole.ops.rx_union.LPUART_Receive =
     LPUART_HAL_ReceiveDataPolling;
234
235              }
236              break;
237 #endif
238          /* If new device is requried as the low level device for debug console,
239           * Add the case branch and add the preprocessor macro to judge whether
240           * this kind of device exist in this SOC. */
241          default:
242              /* Device identified is invalid, return invalid device error code. */
243              return kStatus_DEBUGCONSOLE_InvalidDevice;
244      }
245
246      /* Configure the s_debugConsole structure only when the inti operation is
     successful. */
247      s_debugConsole.instance = uartInstance;
248
```

```
249        return kStatus_DEBUGCONSOLE_Success;
250 }
251
252 /* See fsl_debug_console.h for documentation of this function.*/
253 debug_console_status_t DbgConsole_DeInit(void)
254 {
255        if (s_debugConsole.type == kDebugConsoleNone)
256        {
257            return kStatus_DEBUGCONSOLE_Success;
258        }
259
260        switch(s_debugConsole.type)
261        {
262 #if defined(UART_INSTANCE_COUNT)
263            case kDebugConsoleUART:
264                CLOCK_SYS_DisableUartClock(s_debugConsole.instance);
265                break;
266 #endif
267 #if defined(UART0_INSTANCE_COUNT)
268            case kDebugConsoleLPSCI:
269                CLOCK_SYS_DisableLpsciClock(s_debugConsole.instance);
270                break;
271 #endif
272 #if defined(LPUART_INSTANCE_COUNT)
273            case kDebugConsoleLPUART:
274                CLOCK_SYS_DisableLpuartClock(s_debugConsole.instance);
275                break;
276 #endif
277            default:
278                return kStatus_DEBUGCONSOLE_InvalidDevice;
279        }
280
281        s_debugConsole.type = kDebugConsoleNone;
282
283        return kStatus_DEBUGCONSOLE_Success;
284 }
285
286 #if (defined(__KSDK_STDLIB__))
287 int _WRITE(int fd, const void *buf, size_t nbytes)
288 {
289        if (buf == 0)
290        {
291            /* This means that we should flush internal buffers.  Since we*/
292            /* don't we just return.  (Remember, "handle" == -1 means that all*/
```

```
293            /* handles should be flushed.)*/
294            return 0;
295        }
296
297
298        /* Do nothing if the debug uart is not initialized.*/
299        if (s_debugConsole.type == kDebugConsoleNone)
300        {
301            return -1;
302        }
303
304        /* Send data.*/
305        s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t const *)buf,
           nbytes);
306        return nbytes;
307
308 }
309
310 int _READ(int fd, void *buf, size_t nbytes)
311 {
312
313        /* Do nothing if the debug uart is not initialized.*/
314        if (s_debugConsole.type == kDebugConsoleNone)
315        {
316            return -1;
317        }
318
319        /* Receive data.*/
320        s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, buf, nbytes);
321        return nbytes;
322 }
323 #elif __ICCARM__
324
325 #pragma weak __write
326 size_t __write(int handle, const unsigned char * buffer, size_t size)
327 {
328        if (buffer == 0)
329        {
330            /* This means that we should flush internal buffers.  Since we*/
331            /* don't we just return.  (Remember, "handle" == -1 means that all*/
332            /* handles should be flushed.)*/
333            return 0;
334        }
335
```

```
336      /* This function only writes to "standard out" and "standard err",*/
337      /* for all other file handles it returns failure.*/
338      if ((handle != _LLIO_STDOUT) && (handle != _LLIO_STDERR))
339      {
340          return _LLIO_ERROR;
341      }
342
343      /* Do nothing if the debug uart is not initialized.*/
344      if (s_debugConsole.type == kDebugConsoleNone)
345      {
346          return _LLIO_ERROR;
347      }
348
349      /* Send data.*/
350      s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t const *)buffer,
         size);
351      return size;
352  }
353
354  #pragma weak __read
355  size_t __read(int handle, unsigned char * buffer, size_t size)
356  {
357      /* This function only reads from "standard in", for all other file*/
358      /* handles it returns failure.*/
359      if (handle != _LLIO_STDIN)
360      {
361          return _LLIO_ERROR;
362      }
363
364      /* Do nothing if the debug uart is not initialized.*/
365      if (s_debugConsole.type == kDebugConsoleNone)
366      {
367          return _LLIO_ERROR;
368      }
369
370      /* Receive data.*/
371      s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, buffer, size);
372
373      return size;
374  }
375
376  #elif (defined(__GNUC__))
377  #pragma weak _write
378  int _write (int handle, char *buffer, int size)
```

```
379 {
380     if (buffer == 0)
381     {
382         /* return -1 if error */
383         return -1;
384     }
385
386     /* This function only writes to "standard out" and "standard err",*/
387     /* for all other file handles it returns failure.*/
388     if ((handle != 1) && (handle != 2))
389     {
390         return -1;
391     }
392
393     /* Do nothing if the debug uart is not initialized.*/
394     if (s_debugConsole.type == kDebugConsoleNone)
395     {
396         return -1;
397     }
398
399     /* Send data.*/
400     s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (uint8_t *)buffer, size);
401     return size;
402 }
403
404 #pragma weak _read
405 int _read(int handle, char *buffer, int size)
406 {
407     /* This function only reads from "standard in", for all other file*/
408     /* handles it returns failure.*/
409     if (handle != 0)
410     {
411         return -1;
412     }
413
414     /* Do nothing if the debug uart is not initialized.*/
415     if (s_debugConsole.type == kDebugConsoleNone)
416     {
417         return -1;
418     }
419
420     /* Receive data.*/
421     s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, (uint8_t *)buffer, size);
422     return size;
```

```
423 }
424 #elif defined(__CC_ARM) && !defined(MQX_STDIO)
425 struct __FILE
426 {
427     int handle;
428     /* Whatever you require here. If the only file you are using is */
429     /* standard output using printf() for debugging, no file handling */
430     /* is required. */
431 };
432
433 /* FILE is typedef in stdio.h. */
434 #pragma weak __stdout
435 FILE __stdout;
436 FILE __stdin;
437
438 #pragma weak fputc
439 int fputc(int ch, FILE *f)
440 {
441     /* Do nothing if the debug uart is not initialized.*/
442     if (s_debugConsole.type == kDebugConsoleNone)
443     {
444         return -1;
445     }
446
447     /* Send data.*/
448     s_debugConsole.ops.tx_union.Send(s_debugConsole.base, (const uint8_t*)&ch, 1);
449     return 1;
450 }
451
452 #pragma weak fgetc
453 int fgetc(FILE *f)
454 {
455     uint8_t temp;
456     /* Do nothing if the debug uart is not initialized.*/
457     if (s_debugConsole.type == kDebugConsoleNone)
458     {
459         return -1;
460     }
461
462     /* Receive data.*/
463     s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, &temp, 1);
464     return temp;
465 }
466
```

```
467 #endif
468
469 /*************Code for debug_printf/scanf/assert******************************/
470 int debug_printf(const char *fmt_s, ...)
471 {
472     va_list  ap;
473     int  result;
474     /* Do nothing if the debug uart is not initialized.*/
475     if (s_debugConsole.type == kDebugConsoleNone)
476     {
477         return -1;
478     }
479     va_start(ap, fmt_s);
480     result = _doprint(NULL, debug_putc, -1, (char *)fmt_s, ap);
481     va_end(ap);
482
483     return result;
484 }
485
486 static int debug_putc(int ch, void* stream)
487 {
488     const unsigned char c = (unsigned char) ch;
489     /* Do nothing if the debug uart is not initialized.*/
490     if (s_debugConsole.type == kDebugConsoleNone)
491     {
492         return -1;
493     }
494     s_debugConsole.ops.tx_union.Send(s_debugConsole.base, &c, 1);
495
496     return 0;
497
498 }
499
500 int debug_putchar(int ch)
501 {
502     /* Do nothing if the debug uart is not initialized.*/
503     if (s_debugConsole.type == kDebugConsoleNone)
504     {
505         return -1;
506     }
507     debug_putc(ch, NULL);
508
509     return 1;
510 }
```

```
511
512  int debug_scanf(const char  *fmt_ptr, ...)
513  {
514      char    temp_buf[IO_MAXLINE];
515      va_list ap;
516      uint32_t i;
517      char result;
518
519      /* Do nothing if the debug uart is not initialized.*/
520      if (s_debugConsole.type == kDebugConsoleNone)
521      {
522          return -1;
523      }
524      va_start(ap, fmt_ptr);
525      temp_buf[0] = '\0';
526
527      for (i = 0; i < IO_MAXLINE; i++)
528      {
529          temp_buf[i] = result = debug_getchar();
530
531          if ((result == '\r') || (result == '\n'))
532          {
533              /* End of Line */
534              if (i == 0)
535              {
536                  i = (uint32_t)-1;
537              }
538              else
539              {
540                  break;
541              }
542          }
543
544          temp_buf[i + 1] = '\0';
545      }
546
547      result = scan_prv(temp_buf, (char *)fmt_ptr, ap);
548      va_end(ap);
549
550      return result;
551  }
552
553  int debug_getchar(void)
554  {
```

45

```
555     unsigned char c;

556

557     /* Do nothing if the debug uart is not initialized.*/
558     if (s_debugConsole.type == kDebugConsoleNone)
559     {
560         return -1;
561     }
562     s_debugConsole.ops.rx_union.Receive(s_debugConsole.base, &c, 1);

563

564     return c;
565 }
566 /*******************************************************************************
567  * EOF
568  ******************************************************************************/
```

7.8 ../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 /**
16  * Ungates the clock for a gpio port
17  * @param PORT_ID the GPIO port id(A,B)
18  */
19 #define GPIO_UNGATE_PORT(PORT_ID)\
20   _GPIO_UNGATE_PORT(PORT_ID)
21
22 //Wrapper macro above is needed for argument expansion when using concatenation
23 #define _GPIO_UNGATE_PORT(PORT_ID)\
24   /* un-gate port clock*/\
25     SIM_SCGC5 |= SIM_SCGC5_PORT ## PORT_ID (CGC_CLOCK_ENABLED)
26
27 /**
28  * inits a pin as GPIO in the given direction
29  * @param PORT_ID the GPIO port id(A,B)
30  * @param PIN_NUM pin number in port
31  * @param DIR pin direction (GPIO_HIGH, GPIO_LOW)
32  */
33 #define GPIO_INIT_PIN(PORT_ID, PIN_NUM, DIR)\
34     _GPIO_INIT_PIN(PORT_ID, PIN_NUM, DIR)
35
36 //Wrapper macro above is needed for argument expansion when using concatenation
37 #define _GPIO_INIT_PIN(PORT_ID, PIN_NUM, DIR)\
38     /* set pin as gpio */\
39   PORT_PCR_REG(PORT ## PORT_ID , PIN_NUM) = PORT_PCR_MUX(GPIO_MUX_ALT);\
40     /* Set pin direction */\
41     if(DIR == GPIO_OUTPUT){\
42       GPIO ## PORT_ID ## _PDDR |= GPIO_PDDR_PDD(0x01 << PIN_NUM);\
```

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

```
87   */
88  #define GPIO_READ_PIN ( PORT_ID , PIN_NUM )\
89      _GPIO_READ_PIN ( PORT_ID , PIN_NUM )
90
91  //Wrapper macro above is needed for argument expansion when using concatenation
92  #define _GPIO_READ_PIN ( PORT_ID , PIN_NUM )\
93      ( (( GPIO ## PORT_ID ## _PDIR & (0 x01u << PIN_NUM )) >> PIN_NUM ) )
94
95  /**
96   * Reads the driven status of a output GPIO PIN
97   * @param PORT_ID the GPIO port id(A,B)
98   * @param PIN_NUM pin number in port
99   * @param VAL pin value ( GPIO_HIGH , GPIO_LOW )
100   */
101  #define GPIO_GET_OUTPUT_STATE ( PORT_ID , PIN_NUM )\
102      _GPIO_GET_OUTPUT_STATE ( PORT_ID , PIN_NUM )
103
104  //Wrapper macro above is needed for argument expansion when using concatenation
105  #define _GPIO_GET_OUTPUT_STATE ( PORT_ID , PIN_NUM )\
106      ( (( GPIO ## PORT_ID ## _PDOR & (0 x01u << PIN_NUM )) >> PIN_NUM ) )
107
108  #endif /* SOURCES_GPIO_GPIO_HAL_H_ */
```

7.9 ../Sources/Heater/heater_hal.c

```
1   /* ******************************************************************* */
2   /* File name:        heater_hal.c                                     */
3   /* File description: File containing the functions/methods            */
4   /*                   for heater control                               */
5   /* Author name:      ddello                                           */
6   /* Creation date:    14jun2016                                        */
7   /* Revision date:    14jun2016                                        */
8   /* ******************************************************************* */
9
10  #include "heater_hal.h"
11  #include "KL25Z/es670_peripheral_board.h"
12  #include "GPIO/gpio_hal.h"
13
14  /**
15   * Initialize the heater module
16   */
17  void heater_initHeater(void){
18    /* Init HEATER pin */
19    SIM_SCGC5 |= SIM_SCGC5_PORTA(1u);
20    PORT_PCR_REG(HEATER_PORT_BASE_PNT , HEATER_PIN) = PORT_PCR_MUX(HEATER_PIN_MUX_ALT);
21
22    /* INIT TPM */
23    //Init TPM Clock source as OSCERCLOCK
24    SIM_WR_SOPT2_TPMSRC(SIM,0x2);
25    //Enable clock to TPM1
26    SIM_SCGC6 |= SIM_SCGC6_TPM1(0x1);
27    //PWM mode
28    TPM_WR_SC_CPWMS(HEATER_TPM_BASE_PNT, 0x0);  //Edge Aligned
29    /* Channel mode configuration should be done with the channel disabled and need
         acknowledge*/
30    TPM_WR_CnSC(HEATER_TPM_BASE_PNT, HEATER_TPM_CHANNEL_INDEX, TPM_CnSC_ELSB(1) |
         TPM_CnSC_MSB(1));
31      /* Wait till mode change is acknowledged */
32      while (!
33          (
34          TPM_RD_CnSC(HEATER_TPM_BASE_PNT, HEATER_TPM_CHANNEL_INDEX)
35          & (TPM_CnSC_ELSB(1) | TPM_CnSC_MSB(1))
36        )
37        );
38    //Same period as OSCRCLK
39    TPM_WR_MOD(HEATER_TPM_BASE_PNT, 0xFFFE);
40    //Duty cycle
```

```
41    TPM_WR_CnV_VAL(HEATER_TPM_BASE_PNT, HEATER_TPM_CHANNEL_INDEX, 0);
42    //Prescale 1:1
43    TPM_WR_SC_PS(HEATER_TPM_BASE_PNT, 0x0);
44    //Select LTPM Clock Source for TPM module
45    TPM_WR_SC_CMOD(HEATER_TPM_BASE_PNT, 0x1);
46  }
47
48
49  /**
50   * Set the heater intensity
51   *
52   * @param uiIntensity heater intensity from 0 (stopped) to 0xFFFF (max)
53   */
54  void heater_setIntensity(uint16_t uiIntensity){
55    TPM_WR_CnV_VAL(HEATER_TPM_BASE_PNT, HEATER_TPM_CHANNEL_INDEX, uiIntensity);
56  }
```

7.10   ../Sources/Heater/heater_hal.h

```
1  /* ***************************************************************** */
2  /* File name:         heater_hal.h                                   */
3  /* File description: Header file containing the functions/methods    */
4  /*                   interfaces for heater control                   */
5  /* Author name:      ddello                                          */
6  /* Creation date:    14jun2016                                       */
7  /* Revision date:    14jun2016                                       */
8  /* ***************************************************************** */
9
10 #ifndef SOURCES_HEATER_HAL_H_
11 #define SOURCES_HEATER_HAL_H_
12
13 #include <stdint.h>
14
15 /**
16  * Initialize the heater module
17  */
18 void heater_initHeater(void);
19
20
21 /**
22  * Set the heater intensity
23  *
24  * @param uiIntensity heater intensity from 0 (stopped) to 0xFFFF (max)
25  */
26 void heater_setIntensity(uint16_t uiIntensity);
27
28 #endif /* SOURCES_HEATER_HAL_H_ */
```

7.11 ../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 #include "MKL25Z4_extension.h"
16
17 /*                  General uC definitions                */
18
19 /* Clock gate control */
20 #define   CGC_CLOCK_DISABLED          0x00U
21 #define   CGC_CLOCK_ENABLED           0x01U
22
23 /* GPIO input / output */
24 #define GPIO_INPUT                    0x00U
25 #define GPIO_OUTPUT                   0x01U
26
27 #define GPIO_MUX_ALT                  0x01u
28
29 #define GPIO_HIGH          1
30 #define GPIO_LOW           0
31
32 /*  Workaround for PORT_ID macro expansion to stop at port level*/
33 typedef int A;
34 typedef int B;
35 typedef int C;
36 typedef int D;
37 typedef int E;
38
39 /*                  END OF General uC definitions          */
40
41
42 /*                  BUZZER Definitions                     */
```

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

```
 76
 77 #define LS4_PIN                      5                                              /*
       led/switch #4 pin */
 78 #define LS4_DIR_OUTPUT               (GPIO_OUTPUT << LS4_PIN)
 79 #define LS4_DIR_INPUT                (GPIO_OUTPUT << LS4_PIN)
 80 #define LS4_ALT                      LS1_ALT
 81
 82 /*                END OF LED and SWITCH definitions                */
 83
 84 /*                SEVEN SEGMENT DISPLAY Definitions                      */
 85 #define SEV_SEG_PORT_BASE_PNT     PORTC                                            /*
       peripheral port base pointer */
 86 #define SEV_SEG_PORT_ID           C                                                /*
       peripheral port identifier*/
 87 #define SEV_SEG_GPIO_BASE_PNT     PTC                                              /*
       peripheral gpio base pointer */
 88
 89 #define SEV_SEG_PIT_TIMER_NUMB    0                          /* timer number for seven seg
       PIT */
 90
 91 #define SEGA_PIN                     0                                            /* Segment
       A*/
 92 #define SEGA_DIR_OUTPUT              (GPIO_OUTPUT << SEGA_PIN)
 93 #define SEGA_ALT                     0x01u                                          /* GPIO
       alternative */
 94
 95 #define SEGB_PIN                     1
 96 #define SEGB_DIR_OUTPUT              (GPIO_OUTPUT << SEGB_PIN)
 97 #define SEGB_ALT                     SEGA_ALT
 98
 99 #define SEGC_PIN                     2
100 #define SEGC_DIR_OUTPUT              (GPIO_OUTPUT << SEGC_PIN)
101 #define SEGC_ALT                     SEGA_ALT
102
103 #define SEGD_PIN                     3
104 #define SEGD_DIR_OUTPUT              (GPIO_OUTPUT << SEGD_PIN)
105 #define SEGD_ALT                     SEGA_ALT
106
107 #define SEGE_PIN                     4
108 #define SEGE_DIR_OUTPUT              (GPIO_OUTPUT << SEGE_PIN)
109 #define SEGE_ALT                     SEGA_ALT
110
111 #define SEGF_PIN                     5
112 #define SEGF_DIR_OUTPUT              (GPIO_OUTPUT << SEGF_PIN)
```

```c
113  #define SEGF_ALT                    SEGA_ALT
114
115  #define SEGG_PIN                    6
116  #define SEGG_DIR_OUTPUT             (GPIO_OUTPUT << SEGG_PIN)
117  #define SEGG_ALT                    SEGA_ALT
118
119  #define SEGDP_PIN                   7
120  #define SEGDP_DIR_OUTPUT            (GPIO_OUTPUT << SEGDP_PIN)
121  #define SEGDP_ALT                   SEGA_ALT
122
123  #define SEG_DISP1_PIN               13
124  #define SEG_DISP1_DIR_OUTPUT        (GPIO_OUTPUT << SEG_DISP1_PIN)
125  #define SEG_DISP1_ALT               SEGA_ALT
126
127  #define SEG_DISP2_PIN               12
128  #define SEG_DISP2_DIR_OUTPUT        (GPIO_OUTPUT << SEG_DISP2_PIN)
129  #define SEG_DISP2_ALT               SEGA_ALT
130
131  #define SEG_DISP3_PIN               11
132  #define SEG_DISP3_DIR_OUTPUT        (GPIO_OUTPUT << SEG_DISP3_PIN)
133  #define SEG_DISP3_ALT               SEGA_ALT
134
135  #define SEG_DISP4_PIN               10
136  #define SEG_DISP4_DIR_OUTPUT        (GPIO_OUTPUT << SEG_DISP4_PIN)
137  #define SEG_DISP4_ALT               SEGA_ALT
138
139  /*                END of SEVEN SEGMENT DISPLAY Definitions                  */
140
141
142  /*                END OF General uC definitions           */
143
144
145  /*                LCD definitions                  */
146
147  /* LCD Register Selector
148   * Used as register selector input
149   * When (LCD_RS = LCD_RS_HIGH) => DATA register is selected
150   * When (LCD_RS = LCD_RS_LOW)  => INSTRUCTION register is selected
151  */
152  #define LCD_PORT_BASE_PNT           PORTC                                    /*
          peripheral port base pointer */
153  #define LCD_PORT_ID         C
154  #define LCD_GPIO_BASE_PNT           PTC                                      /*
          peripheral gpio base pointer */
```

```
155
156 #define LCD_RS_PIN                  8                                              /* register
        selector */
157 #define LCD_RS_DIR                  GPIO_OUTPUT
158
159 #define LCD_ENABLE_PIN              9                                              /* enable
        pin */
160 #define LCD_ENABLE_DIR              GPIO_OUTPUT
161
162 #define LCD_RS_HIGH                 GPIO_HIGH
163 #define LCD_RS_DATA                 LCD_RS_HIGH
164
165 #define LCD_RS_LOW                  GPIO_LOW
166 #define LCD_RS_CMD                  LCD_RS_LOW
167
168 #define LCD_ENABLED                 1U
169 #define LCD_DISABLED                0U
170
171 #define LCD_DATA_DIR                GPIO_OUTPUT                          /* LCD data pins
        */
172
173 #define LCD_DATA_DB0_PIN            0
174 #define LCD_DATA_DB1_PIN            1
175 #define LCD_DATA_DB2_PIN            2
176 #define LCD_DATA_DB3_PIN            3
177 #define LCD_DATA_DB4_PIN            4
178 #define LCD_DATA_DB5_PIN            5
179 #define LCD_DATA_DB6_PIN            6
180 #define LCD_DATA_DB7_PIN            7
181 /*                  END OF LCD definitions                    */
182
183 /*          COOLER DEFINITIONS              */
184 #define COOLER_PORT_ID          A
185 #define COOLER_PORT_BASE_PNT    PORTA
186 #define COOLER_PIN              13
187 #define COOLER_PIN_MUX_ALT      0x3
188
189 #define COOLER_TPM_BASE_PNT     TPM1
190 #define COOLER_TPM_CHANNEL_INDEX  1
191
192
193 /*          END OF COOLER DEFINITIONS              */
194
195 /*          HEATER DEFINITIONS              */
```

```c
196 #define HEATER_PORT_ID         A
197 #define HEATER_PORT_BASE_PNT    PORTA
198 #define HEATER_PIN            12
199 #define HEATER_PIN_MUX_ALT      0x3

201 #define HEATER_TPM_BASE_PNT     TPM1
202 #define HEATER_TPM_CHANNEL_INDEX  0


205 /*          END OF HEATER DEFINITIONS              */

207 /*          TACOMETER DEFINITIONS         */
208 #define TACOMETRO_PORT_ID      E
209 #define TACOMETRO_PORT_BASE_PNT   PORTE
210 #define TACOMETRO_PIN        29
211 #define TACOMETRO_PIN_ALT      0x4    /*ALT4 FTM_CLOCKIN0*/

213 #define TACOMETRO_TPM_BASE      TPM0

215 /*          END OF TACOMETER DEFINITIONS    */

217 /*          ADC DEFINITIONS              */
218 #define ADC_PORT_ID          E
219 #define ADC_PORT_BASE_PNT      PORTE
220 #define ADC_NUM_BASE_PNT       ADC0_BASE_PTR
221 #define ADC_MUX_IDX          0    //MUX A
222 #define ADC_CHANNEL_SEL        0x4
223 #define ADC_TEMP_PIN         21
224 #define ADC_VOLT_PIN         20
225 #define ADC_TEMP_PIN_MUX_ALT    0x0
226 #define ADC_VOLT_PIN_MUX_ALT    0x0

228 #define HEATER_TPM_BASE_PNT     TPM1
229 #define HEATER_TPM_CHANNEL_INDEX  0


232 /*          END OF ADC DEFINITIONS             */


235 #endif /* SOURCES_ES670_PERIPHERAL_BOARD_H_ */
```

7.12 ../Sources/LCD/lcd_hal.c

```c
/* ******************************************************************* */
/* File name:        lcd_hal.c                                         */
/* File description: File dedicated to the hardware abstraction layer*/
/*                   related to the LCD HARDWARE based on the KS006U */
/*                   controller                                        */
/* Author name:      dloubach                                          */
/* Creation date:    16out2015                                         */
/* Revision date:    13mai2016                                         */
/* ******************************************************************* */

#include "lcd_hal.h"
#include "KL25Z/es670_peripheral_board.h"
#include "Util/util.h"
#include "GPIO/gpio_hal.h"

/* line and columns */
#define LINE0         0U
#define COLUMN0         0U

#define L0C0_BASE     0x80 /* line 0, column 0 */
#define L1C0_BASE     0xC0 /* line 1, column 0 */
#define MAX_COLUMN   15U

/**
 * Initialize the LCD function
 */
void lcd_initLcd(void)
{
    /* pins configured as outputs */

    /* un-gate port clock*/
    GPIO_UNGATE_PORT(LCD_PORT_ID);

    /* set pin as gpio output*/
    GPIO_INIT_PIN(LCD_PORT_ID, LCD_RS_PIN, LCD_RS_DIR);
    GPIO_INIT_PIN(LCD_PORT_ID, LCD_ENABLE_PIN, LCD_ENABLE_DIR);
    GPIO_INIT_PIN(LCD_PORT_ID, LCD_DATA_DB0_PIN, LCD_DATA_DIR);
    GPIO_INIT_PIN(LCD_PORT_ID, LCD_DATA_DB1_PIN, LCD_DATA_DIR);
    GPIO_INIT_PIN(LCD_PORT_ID, LCD_DATA_DB2_PIN, LCD_DATA_DIR);
    GPIO_INIT_PIN(LCD_PORT_ID, LCD_DATA_DB3_PIN, LCD_DATA_DIR);
    GPIO_INIT_PIN(LCD_PORT_ID, LCD_DATA_DB4_PIN, LCD_DATA_DIR);
    GPIO_INIT_PIN(LCD_PORT_ID, LCD_DATA_DB5_PIN, LCD_DATA_DIR);
```

```
43      GPIO_INIT_PIN(LCD_PORT_ID, LCD_DATA_DB6_PIN, LCD_DATA_DIR);
44      GPIO_INIT_PIN(LCD_PORT_ID, LCD_DATA_DB7_PIN, LCD_DATA_DIR);
45
46      // turn-on LCD, with no cursor and no blink
47      lcd_sendCommand(CMD_NO_CUR_NO_BLINK);
48
49      // init LCD
50      lcd_sendCommand(CMD_INIT_LCD);
51
52      // clear LCD
53      lcd_sendCommand(CMD_CLEAR);
54
55      // LCD with no cursor
56      lcd_sendCommand(CMD_NO_CURSOR);
57
58      // cursor shift to right
59      lcd_sendCommand(CMD_CURSOR2R);
60
61  }
62
63
64
65  /**
66   * Send command or data to LCD
67   * @param ucBuffer Char to be send
68   * @param cDataType Command LCD_RS_CMD or data LCD_RS_DATA
69   */
70  void lcd_write2Lcd(unsigned char ucBuffer,  unsigned char cDataType)
71  {
72      /* writing data or command */
73      if(LCD_RS_CMD == cDataType){
74          /* will send a command */
75          GPIO_WRITE_PIN(LCD_PORT_ID, LCD_RS_PIN, LCD_RS_CMD);
76      } else{
77          /* will send data */
78        GPIO_WRITE_PIN(LCD_PORT_ID, LCD_RS_PIN, LCD_RS_DATA);
79      }
80
81      /* write in the LCD bus */
82      GPIO_WRITE_PIN(LCD_PORT_ID, LCD_DATA_DB0_PIN, ((ucBuffer & (1u << 0u)) >> 0u));
83      GPIO_WRITE_PIN(LCD_PORT_ID, LCD_DATA_DB1_PIN, ((ucBuffer & (1u << 1u)) >> 1u));
84      GPIO_WRITE_PIN(LCD_PORT_ID, LCD_DATA_DB2_PIN, ((ucBuffer & (1u << 2u)) >> 2u));
85      GPIO_WRITE_PIN(LCD_PORT_ID, LCD_DATA_DB3_PIN, ((ucBuffer & (1u << 3u)) >> 3u));
86      GPIO_WRITE_PIN(LCD_PORT_ID, LCD_DATA_DB4_PIN, ((ucBuffer & (1u << 4u)) >> 4u));
```

```
87      GPIO_WRITE_PIN(LCD_PORT_ID, LCD_DATA_DB5_PIN, ((ucBuffer & (1u << 5u)) >> 5u));
88      GPIO_WRITE_PIN(LCD_PORT_ID, LCD_DATA_DB6_PIN, ((ucBuffer & (1u << 6u)) >> 6u));
89      GPIO_WRITE_PIN(LCD_PORT_ID, LCD_DATA_DB7_PIN, ((ucBuffer & (1u << 7u)) >> 7u));
90
91      pit_mask_interrupts();
92      /* enable, delay, disable LCD */
93      /* this generates a pulse in the enable pin */
94      GPIO_WRITE_PIN(LCD_PORT_ID, LCD_ENABLE_PIN, LCD_ENABLED);
95      util_genDelay1ms();
96      GPIO_WRITE_PIN(LCD_PORT_ID, LCD_ENABLE_PIN, LCD_DISABLED);
97      util_genDelay1ms();
98      util_genDelay1ms();
99      pit_unmask_interrupts();
100 }
101
102
103
104 /**
105  * Write data to be displayed
106  * @param ucData Char to be written
107  */
108 void lcd_writeData(unsigned char ucData)
109 {
110     /* just a relay to send data */
111     lcd_write2Lcd(ucData, LCD_RS_DATA);
112 }
113
114
115
116 /**
117  * Write command to LCD
118  * @param ucCmd Command to be executed
119  */
120 void lcd_sendCommand(unsigned char ucCmd)
121 {
122     /* just a relay to send command */
123     lcd_write2Lcd(ucCmd, LCD_RS_CMD);
124 }
125
126
127
128 /**
129  * Set cursor line and column
130  * @param cLine = LINE0..LINE1
```

```
131  * @param  cColumn = COLUMN0..MAX_COLUMN
132  */
133  void lcd_setCursor(unsigned char cLine, unsigned char cColumn)
134  {
135      char cCommand;
136
137      if(LINE0 == cLine)
138          /* line 0 */
139          cCommand = L0C0_BASE;
140      else
141          /* line 1 */
142          cCommand = L1C0_BASE;
143
144      /* maximum MAX_COLUMN columns */
145      cCommand += (cColumn & MAX_COLUMN);
146
147      // send the command to set the cursor
148      lcd_sendCommand(cCommand);
149  }
150
151
152
153  /**
154   * Write string to be displayed
155   * @param cBuffer String to be written in LCD
156   */
157  void lcd_writeString(const char *cBuffer)
158  {
159      while(*cBuffer)
160      {
161          lcd_writeData(*cBuffer++);
162      };
163  }
164
165  /**
166  /* Print string to the display, clears display and handles cursor
167  /* @param cBuffer String to be written in LCD
168   */
169  void lcd_printString(const char *cBuffer)
170  {
171      // clear LCD
172      lcd_sendCommand(CMD_CLEAR);
173
174      // set the cursor line 0, column 1
```

```
175        lcd_setCursor(0,1);
176        unsigned int uiCursorCollum = 0;
177        while(*cBuffer)
178        {
179            lcd_writeData(*cBuffer++);
180            if(++uiCursorCollum == MAX_COLUMN){
181                lcd_setCursor(1,0);
182                uiCursorCollum = 0;
183            }
184        };
185 }
186
187
188 /**
189  * Write a dummy hard coded text
190  */
191 void lcd_dummyText(void)
192 {
193     // clear LCD
194     lcd_sendCommand(CMD_CLEAR);
195
196     // set the cursor line 0, column 1
197     lcd_setCursor(0,1);
198
199     // send string
200     lcd_writeString("*** ES670 ***");
201
202     // set the cursor line 1, column 0
203     lcd_setCursor(1,0);
204     lcd_writeString("Prj Sis Embarcad");
205 }
```

7.13 ../Sources/LCD/lcd_hal.h

```c
/* ****************************************************************** */
/* File name:        lcd_hal.h                                       */
/* File description: Header file containing the functions/methods    */
/*                   interfaces for handling the LCD hardware from    */
/*                   the hardware kit                                 */
/* Author name:      dloubach                                        */
/* Creation date:    16out2015                                       */
/* Revision date:    13mai2016                                       */
/* ****************************************************************** */

#ifndef SOURCES_LCD_HAL_H_
#define SOURCES_LCD_HAL_H_

/* lcd basic commands list */
#define CMD_INIT_LCD        0x0F
#define CMD_CLEAR           0x01
#define CMD_NO_CURSOR       0x0C
#define CMD_CURSOR2R        0x06 /* cursor to right */
#define CMD_NO_CUR_NO_BLINK 0x38 /* no cursor, no blink */


/**
 * Initialize the LCD function
 */
void lcd_initLcd(void);


/**
 * Send command or data to LCD
 * @param ucBuffer Char to be send
 * @param cDataType Command LCD_RS_CMD or data LCD_RS_DATA
 */
void lcd_writeData(unsigned char ucData);


/**
 * Write command to LCD
 * @param ucCmd Command to be executed
 */
void lcd_sendCommand(unsigned char ucCmd);

```

```
43  /**
44   * Write string to be displayed
45   * @param cBuffer String to be written in LCD
46   */
47  void lcd_writeString(const char *cBuffer);
48
49  /**
50  /* Print string to the display, clears display and handles cursor
51  /* @param cBuffer String to be written in LCD
52   */
53  void lcd_printString(const char *cBuffer);
54
55  /**
56   * Set cursor line and column
57   * @param cLine = LINE0..LINE1
58   * @param cColumn = COLUMN0..MAX_COLUMN
59   */
60  void lcd_setCursor(unsigned char cLine, unsigned char cColumn);
61
62
63  /**
64   * Write a dummy hard coded text
65   */
66  void lcd_dummyText(void);
67
68
69  #endif /* SOURCES_LCD_HAL_H_ */
```

7.14 ../Sources/LedSwi/ledswi_hal.c

```c
/* ****************************************************************** */
/* File name:        ledswi_hal.c                                     */
/* File description: This file has a couple of useful functions to    */
/*                   control LEDs and Switches from peripheral board  */
/* Author name:      dloubach                                         */
/* Creation date:    20jan2015                                        */
/* Revision date:    13abr2016                                        */
/* ****************************************************************** */

#include "ledswi_hal.h"
#include "GPIO/gpio_hal.h"

#define USING_OPENSDA_DEBUG

/**
 * As the hardware board was designed with LEDs/Switches sharing
 *  the same pins, this method configures how many LEDS and switches
 *  will be available for the application
 * @param cLedNum num of LEDs
 * @param cSwitchNum num of Switches (cLedNum + cSwitchNum <= MAX_LED_SWI)
 */
void ledswi_initLedSwitch( char cLedNum, char cSwitchNum )
{
    /* un-gate port clock*/
    SIM_SCGC5 |= SIM_SCGC5_PORTA( CGC_CLOCK_ENABLED );

    /* set pin as gpio */
#ifndef USING_OPENSDA_DEBUG
    PORTA_PCR1 = PORT_PCR_MUX( LS1_ALT );
    PORTA_PCR2 = PORT_PCR_MUX( LS2_ALT );
#endif
    PORTA_PCR4 = PORT_PCR_MUX( LS3_ALT );
    PORTA_PCR5 = PORT_PCR_MUX( LS4_ALT );


    /* check if the number to configured is according to
    hardware dev kit */
    if(( cLedNum + cSwitchNum ) <= MAX_LED_SWI)
    {
        /* max number of peripherals to configure is ok, carry on */
        switch( cSwitchNum )
        {
```

66

```
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  * initializes pin as LED
80  * @param ePin which pin {1..4}
81  */
82 void ledswi_initLed(ledswi_pin_type_e ePin){
```

```
83    GPIO_INIT_PIN(LS_PORT_ID, ePin, GPIO_OUTPUT);
84  }
85
86  /**
87   * initializes pin as SWITCH
88   * @param ePin which pin {1..4}
89   */
90  void ledswi_initSwitch(ledswi_pin_type_e ePin){
91    GPIO_INIT_PIN(LS_PORT_ID, ePin, GPIO_INPUT);
92  }
93
94  /**
95   * set the led ON
96   * @param eLedPin which LED {1..4}
97   */
98  void ledswi_setLed(ledswi_pin_type_e eLedPin)
99  {
100     GPIO_WRITE_PIN(LS_PORT_ID, eLedPin, GPIO_HIGH);
101 }
102
103
104
105 /**
106  * set the led OFF
107  * @param eLedPin which LED {1..4}
108  */
109 void ledswi_clearLed(ledswi_pin_type_e eLedPin)
110 {
111     GPIO_WRITE_PIN(LS_PORT_ID, eLedPin, GPIO_LOW);
112 }
113
114
115 /**
116  * return the led status
117  *
118  * @param eLedPin which LED {1..4}
119  *
120  * @return If the led is ON or OFF
121  */
122 led_status_type_e ledswi_getLedStatus(ledswi_pin_type_e eLedPin){
123     led_status_type_e lstReturn = LED_OFF;
124     if(GPIO_GET_OUTPUT_STATE(LS_PORT_ID, eLedPin) == LED_ON){
125       lstReturn = LED_ON;
126     }
```

```
127        return(lstReturn);
128 }
129
130 /**
131  * return the switch status
132  *
133  * @param eSwPin which Switch {1..4}
134  *
135  * @return If the switch is ON or OFF
136  */
137 switch_status_type_e ledswi_getSwitchStatus(ledswi_pin_type_e eSwPin){
138     switch_status_type_e sstReturn = SWITCH_OFF;
139     if(GPIO_READ_PIN(LS_PORT_ID, eSwPin) == SWITCH_ON){
140        sstReturn = SWITCH_ON;
141     }
142     return(sstReturn);
143 }
```

7.15 ../Sources/LedSwi/ledswi_hal.h

```c
/* ***************************************************************** */
/* File name:        ledswi_hal.h                                   */
/* File description: Header file containing the function/methods    */
/*                   prototypes of ledswi.c                         */
/* Author name:      dloubach                                       */
/* Creation date:    09jan2015                                      */
/* Revision date:    13abr2016                                      */
/* ***************************************************************** */

#ifndef SOURCES_LEDSWI_LEDSWI_HAL_H_
#define SOURCES_LEDSWI_LEDSWI_HAL_H_

#include "KL25Z/es670_peripheral_board.h"


#define MAX_LED_SWI        04

typedef enum
{
    LS_1 =  LS1_PIN,
  LS_2 =  LS2_PIN,
  LS_3 =  LS3_PIN,
  LS_4 =  LS4_PIN,
  UNKNOWN = -1
} ledswi_pin_type_e;

typedef enum
{
    SWITCH_ON,
    SWITCH_OFF
} switch_status_type_e;

typedef enum
{
    LED_OFF,
    LED_ON
} led_status_type_e;

/**
 * As the hardware board was designed with LEDs/Switches sharing
 *  the same pins, this method configures how many LEDS and switches
 *  will be available for the application
```

```
43   * @param cLedNum num of LEDs
44   * @param cSwitchNum num of Switches (cLedNum + cSwitchNum <= MAX_LED_SWI)
45   */
46  void ledswi_initLedSwitch( char cLedNum, char cSwitchNum );
47
48
49  /**
50   * initializes pin as LED
51   * @param ePin which pin {1..4}
52   */
53  void ledswi_initLed( ledswi_pin_type_e ePin );
54
55  /**
56   * initializes pin as SWITCH
57   * @param ePin which pin {1..4}
58   */
59  void ledswi_initSwitch( ledswi_pin_type_e ePin );
60
61  /**
62   * set the led ON
63   * @param eLedPin which LED {1..4}
64   */
65  void ledswi_setLed( ledswi_pin_type_e eLedPin );
66
67
68
69  /**
70   * set the led OFF
71   * @param eLedPin which LED {1..4}
72   */
73  void ledswi_clearLed( ledswi_pin_type_e eLedPin );
74
75
76  /**
77   * return the led status
78   *
79   * @param eLedPin which LED {1..4}
80   *
81   * @return If the led is ON or OFF
82   */
83  led_status_type_e ledswi_getLedStatus( ledswi_pin_type_e eLedPin );
84
85  /**
86   * return the switch status
```

```
87   *
88   * @param eSwPin which Switch {1..4}
89   *
90   * @return If the switch is ON or OFF
91   */
92  switch_status_type_e ledswi_getSwitchStatus(ledswi_pin_type_e eSwPin);
93  #endif /* SOURCES_LEDSWI_LEDSWI_HAL_H_ */
```

7.16 ../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  #include "Serial/serial_hal.h"
9  #include "Protocolo/cmdmachine_hal.h"
10 #include "LCD/lcd_hal.h"
11 #include "Util/tc_hal.h"
12 #include "Cooler/cooler_hal.h"
13 #include "Tacometro/tacometro_hal.h"
14 #include "Heater/heater_hal.h"
15 #include "TempSensor/tempSensor_hal.h"
16 #include <string.h>
17 #include <stdio.h>
18
19 /* defines */
20 #define RCV_BUF_SIZE 100
21 #define SND_BUF_SIZE 100
22 #define CYCLIC_EXECUTIVE_PERIOD        500 * 10000 /* 1000000 micro seconds */
23
24 /* globals */
25 volatile unsigned int uiFlagNextPeriod = 0;        /* cyclic executive flag */
26 uint16_t temp = 0;
27
28
29 /**
30  * cyclic executive interrupt service routine
31  */
32 void main_cyclicExecuteIsr(void)
33 {
34     /* set the cyclic executive flag */
35     uiFlagNextPeriod = 1;
36 }
37
38
39 /**
40  * Initialize board and periferics
41  */
42 void main_boardInit(){
```

```
43   mcg_clockInit();
44   ledswi_initLedSwitch(1,3);
45   serial_initUart();
46   lcd_initLcd();
47   lcd_dummyText();
48   sevenseg_init();
49   buzzer_init();
50   tacometro_init();
51   cooler_initCooler();
52   heater_initHeater();
53   tempSensor_init();
54 }
55
56 /**
57  * Reads serial comm, interprets received commands and prints output
58  */
59 void main_protocolCheck(){
60   static char rcvBuffer[RCV_BUF_SIZE];
61   static char sndBuffer[SND_BUF_SIZE];
62   if(UART0_BRD_S1_RDRF(UART0)){
63     int iCmdSize = 0;
64     iCmdSize = serial_recieveBuffer(rcvBuffer, RCV_BUF_SIZE);
65     if(iCmdSize > 0){
66       cmdmachine_interpretCmdBuffer(rcvBuffer, iCmdSize, sndBuffer);
67       serial_sendBuffer(sndBuffer, strlen(sndBuffer));
68     }
69   }
70 }
71
72 /**
73  * Checks current cooler speed and prints it to lcd panel
74  */
75 void main_checkCoolerSpeed(){
76   static char coolerSpeedBuffer[15];
77   sprintf(coolerSpeedBuffer, "%d rps - %d",
78     tacometro_getSpeed(CYCLIC_EXECUTIVE_PERIOD/1000), cooler_getDutyCicle());
78   lcd_printString(coolerSpeedBuffer);
79 }
80
81 /**
82  * Main function
83  */
84 int main(void)
85 {
```

```
86    main_boardInit();
87    heater_setIntensity(0x8000);
88
89      /* configure cyclic executive interruption */
90      tc_installLptmr0(CYCLIC_EXECUTIVE_PERIOD, main_cyclicExecuteIsr);
91
92      /* cooperative cyclic executive main loop */
93    while(1){
94 //     main_checkCoolerSpeed();
95      tempSensor_executeTask();
96
97       /* WAIT FOR CYCLIC EXECUTIVE PERIOD */
98      while(!uiFlagNextPeriod){
99        main_protocolCheck();
100     }
101     uiFlagNextPeriod = 0;
102
103   }
104     /* Never leave main */
105     return 0;
106 }
```

7.17 ../Sources/Mcg/mcg_hal.c

```
1  /* ****************************************************************** */
2  /* File name:          mgc_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 /* EXTAL0 PTA18 */
34 #define EXTAL0_PORT                 PORTA
35 #define EXTAL0_PIN                  18U
36 #define EXTAL0_PINMUX               kPortPinDisabled
37
38 /* XTAL0 PTA19 */
39 #define XTAL0_PORT                  PORTA
40 #define XTAL0_PIN                   19U
41 #define XTAL0_PINMUX                kPortPinDisabled
42
```

```c
/* OSC0 configuration */
#define OSC0_INSTANCE                 0U
#define OSC0_XTAL_FREQ                8000000U /* 08 MHz*/
#define OSC0_SC2P_ENABLE_CONFIG       false
#define OSC0_SC4P_ENABLE_CONFIG       false
#define OSC0_SC8P_ENABLE_CONFIG       false
#define OSC0_SC16P_ENABLE_CONFIG      false
#define MCG_HGO0                      kOscGainLow
#define MCG_RANGE0                    kOscRangeVeryHigh
#define MCG_EREFS0                    kOscSrcOsc

/* RTC external clock configuration. */
#define RTC_XTAL_FREQ                 0U
#define RTC_SC2P_ENABLE_CONFIG        false
#define RTC_SC4P_ENABLE_CONFIG        false
#define RTC_SC8P_ENABLE_CONFIG        false
#define RTC_SC16P_ENABLE_CONFIG       false
#define RTC_OSC_ENABLE_CONFIG         false
#define RTC_CLK_OUTPUT_ENABLE_CONFIG  false

/* RTC_CLKIN PTC1 */
#define RTC_CLKIN_PORT                PORTC
#define RTC_CLKIN_PIN                 1U
#define RTC_CLKIN_PINMUX              kPortMuxAsGpio


#define CLOCK_VLPR                    1U /* very low power run mode */
#define CLOCK_RUN                     2U /* run mode */

#ifndef CLOCK_INIT_CONFIG
#define CLOCK_INIT_CONFIG CLOCK_RUN
#endif


/* Configuration for enter VLPR mode, Core clock = 4MHz */
const clock_manager_user_config_t g_defaultClockConfigVlpr =
{
    .mcgConfig =
    {
        .mcg_mode          = kMcgModeBLPI,     // Work in BLPI mode
        .irclkEnable       = true,             // MCGIRCLK enable
        .irclkEnableInStop = false,            // MCGIRCLK disable in STOP mode
        .ircs              = kMcgIrcFast,      // Select IRC4M
        .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 disalbe 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
        to either 0b00 or 0b01).
121  *
122  * FEE (FLL engaged external)
123  * fext / FLL_R must be in the range of 31.25 kHz to 39.0625 kHz
124  * FLL_R is the reference divider selected by the C1[FRDIV] bits
125  * F is the FLL factor selected by C4[DRST_DRS] and C4[DMX32] bits
126  *
127  * (fext / FLL_R) * F = (8 MHz / 256 ) * 1280 = 40 MHz
128  *
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          /* ------------------ MCGIRCCLK 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                                                    // The resulting frequency must be in
    the range 31.25 kHz to 39.0625 kHz
146          .drs     = kMcgDcoRangeSelMid,            // frequency range
147          .dmx32   = kMcgDmx32Default,              // DCO has a default range of 25%
148
149          /* -------------------- MCG PLL settings ---------------------- */
150          .pll0EnableInFllMode = false,             // PLL0 disable
151          .pll0EnableInStop  = false,               // PLL0 disabLe in STOP mode
152          .prdiv0             = 0x0U,
153          .vdiv0              = 0x0U,
154      },
155      /* ----------- system integration module configurations   -------------- */
156      .simConfig =
157      {
158          .pllFllSel = kClockPllFllSelFll,          // PLLFLLSEL select PLL
159          .er32kSrc  = kClockEr32kSrcLpo,           // ERCLK32K selection, use LPO
160          .outdiv1   = 0U,                          // core/system clock, as well as the
    bus/flash clocks.
161          .outdiv4   = 1U,                          // bus and flash clock and is in
    addition to the System clock divide ratio
162      },
163      /* ----------- system oscillator output configurations    -------------- */
164      .oscerConfig =
165      {
166          .enable       = true,                     // OSCERCLK enable
167          .enableInStop = false,                    // OSCERCLK disable in STOP mode
168      }
169  };
```

```
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(EXTAL0_PORT , EXTAL0_PIN , EXTAL0_PINMUX );
262        PORT_HAL_SetMuxMode(XTAL0_PORT , XTAL0_PIN , XTAL0_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 }
```

7.18 ../Sources/Mcg/mcg_hal.h

```c
/* **************************************************************** */
/* File name:        mcg_hal.h                                      */
/* File description: Header file containing the functions/methods   */
/*                   interfaces for handling the Multipurpose clock */
/*                   generator module                               */
/* Author name:      dloubach                                       */
/* Creation date:    21out2015                                      */
/* Revision date:    25fev2016                                      */
/* **************************************************************** */

#ifndef SOURCES_MCG_HAL_H_
#define SOURCES_MCG_HAL_H_


/* ********************************************** */
/* Method name:        mcg_clockInit             */
/* Method description: main board clk configuration */
/* Input params:       n/a                       */
/* Output params:      n/a                       */
/* ********************************************** */
void mcg_clockInit(void);




#endif /* SOURCES_MCG_HAL_H_ */
```

7.19 ../Sources/PIT/pit_hal.c

```c
/* ***************************************************************** */
/* File name:        pit_hal.c                                      */
/* File description: File containing the functions/methods          */
/*                   for handling the Periodic Interruption         */
/*                   Timer module                                   */
/* Author name:      ddello                                         */
/* Creation date:    10abr2016                                      */
/* Revision date:    10mai2016                                      */
/* ***************************************************************** */
//Careful when handling PIT DOC! Bit endianness is inverted in relation to GPIO doc

#include "pit_hal.h"
#include "KL25Z/es670_peripheral_board.h"
#include "fsl_clock_manager.h"

#define PIT_IRQ_NUMBER PIT_IRQn

static unsigned char pit_enabled = 0;

/**
 *Default timer interruption handler. Does nothing.
 */
static void _nop_handler(void){
  PIT_TFLG0 |= PIT_TFLG_TIF(0x1u);
  PIT_TFLG1 |= PIT_TFLG_TIF(0x1u);
}

static void (*fpTimer0Handler)(void) = &_nop_handler;
static void (*fpTimer1Handler)(void) = &_nop_handler;

/**
 * Pit interruption handler. Checks what timer caused the interruption and call the
 * correct timer interruption handler.
 */
void PIT_IRQHandler(void){
  if(PIT_TFLG0){
    (*fpTimer0Handler)();
  }
  if(PIT_TFLG1){
    (*fpTimer1Handler)();
  }
}
```

84

```
43
44  /**
45   * Enables Periodic Interruption Timer module.
46   * (With the stop on debug flag set to on)
47   */
48  void pit_enable(void){
49    SIM_SCGC6 |= SIM_SCGC6_PIT_MASK;
50    PIT_MCR &= ~PIT_MCR_MDIS(0x1u);
51    //Freeze in debug mode
52    PIT_MCR |= PIT_MCR_FRZ(0x1u);
53    NVIC_ClearPendingIRQ(PIT_IRQ_NUMBER);
54      NVIC_EnableIRQ(PIT_IRQ_NUMBER);
55      pit_enabled = 1;
56  }
57
58  /**
59   * Start interruptions for given timer, unchained mode.
60   * Timer interruptions are masked.
61   *
62   * @param usTimer_numb  The number for the desired timer (0,1)
63   * @param uiTimer_period_ms  The number of microseconds between interrupts
64   * @param fpInterrupt_handler    Timer interrupt handler routine address pointer
65   */
66  void pit_start_timer_interrupt(unsigned short usTimer_numb, unsigned int
        uiTimer_period_us, void (*fpInterrupt_handler)(void)){
67    uint32_t ui32BusFreq = CLOCK_SYS_GetBusClockFreq()/1000000; //Freq in MHz
68    uint32_t ui32cyclePeriod = ui32BusFreq*uiTimer_period_us - 1;
69    if(!usTimer_numb){
70      fpTimer0Handler = fpInterrupt_handler;
71      PIT_LDVAL0 = PIT_LDVAL_TSV(ui32cyclePeriod);
72      PIT_TCTRL0 &= ~PIT_TCTRL_CHN(0x1u);   /*Disable chain mode*/
73      PIT_TCTRL0 |= PIT_TCTRL_TIE(0x1u);    /*Enable interrupts for timer 0*/
74      PIT_TCTRL0 |= PIT_TCTRL_TEN(0x1u);    /*Enable timer 0*/
75    }else{
76      fpTimer1Handler = fpInterrupt_handler;
77      PIT_LDVAL1 = PIT_LDVAL_TSV(ui32cyclePeriod);
78      PIT_TCTRL1 &= ~PIT_TCTRL_CHN(0x1u);   /*Disable chain mode*/
79      PIT_TCTRL1 |= PIT_TCTRL_TIE(0x1u);    /*Enable interrupts for timer 1*/
80      PIT_TCTRL1 |= PIT_TCTRL_TEN(0x1u);    /*Enable timer 1*/
81    }
82  }
83
84  /**
85   * Stop interruptions for given timer, unchained mode.
```

```
86   *
87   * @param usTimer_numb   The number for the desired timer (0,1)
88   */
89  void pit_stop_timer_interrupt( unsigned short usTimer_numb ){
90    if(!usTimer_numb){
91      PIT_TCTRL0 &= ~PIT_TCTRL_TIE(0x1u);
92      PIT_TCTRL0 &= ~PIT_TCTRL_TEN(0x1u);
93    }else{
94      PIT_TCTRL1 &= ~PIT_TCTRL_TIE(0x1u);
95      PIT_TCTRL1 &= ~PIT_TCTRL_TEN(0x1u);
96    }
97  }
98
99  /**
100  * Mark interruption as handled for the given timer, this should be called by timer
101  * interruption handlers once they are finished.
102  *
103  * @param usTimer_numb   The number for the desired timer (0,1)
104  */
105 void pit_mark_interrupt_handled( unsigned short usTimer_numb ){
106   if(!usTimer_numb){
107     PIT_TFLG0 |= PIT_TFLG_TIF(0x1u);
108   }else{
109     PIT_TFLG1 |= PIT_TFLG_TIF(0x1u);
110   }
111 }
112
113 /**
114  * Disables PIT interruptions temporarily
115  */
116 void pit_mask_interrupts(){
117   if(pit_enabled){
118 //     NVIC_ClearPendingIRQ(PIT_IRQ_NUMBER);
119     NVIC_DisableIRQ(PIT_IRQ_NUMBER);
120     PIT_TCTRL0 &= ~PIT_TCTRL_TIE(0x1u);
121     PIT_TCTRL1 &= ~PIT_TCTRL_TIE(0x1u);
122   }
123 }
124
125 /**
126  * Re-enables PIT interruptions
127  */
128 void pit_unmask_interrupts(){
129   if(pit_enabled){
```

```
130 //     NVIC_ClearPendingIRQ(PIT_IRQ_NUMBER);
131     NVIC_EnableIRQ(PIT_IRQ_NUMBER);
132     PIT_TCTRL0 |= PIT_TCTRL_TIE(0x1u);     /*Enable interrupts for timer 0*/
133     PIT_TCTRL1 |= PIT_TCTRL_TIE(0x1u);     /*Enable interrupts for timer 1*/
134   }
135 }
```

7.20 ../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:    10mai2016                                      */
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_ms  The number of microseconds 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_us, 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
51   /**
52    * Disables PIT interruptions temporarily
53    */
54   void pit_mask_interrupts();
55
56   /**
57    * Re-enables PIT interruptions
58    */
59   void pit_unmask_interrupts();
60   #endif /* SOURCES_PIT_PIT_HAL_H_ */
```

7.21 ../Sources/print_scan.c

```
1   /***********************************************************************
2    * File:    print_scan.c
3    * Purpose: Implementation of debug_printf(), debug_scanf() functions.
4    *
5    * This is a modified version of the file printf.c, which was distributed
6    * by Motorola as part of the M5407C3BOOT.zip package used to initialize
7    * the M5407C3 evaluation board.
8    *
9    * Copyright:
10   *       1999-2000 MOTOROLA, INC. All Rights Reserved.
11   *   You are hereby granted a copyright license to use, modify, and
12   *   distribute the SOFTWARE so long as this entire notice is
13   *   retained without alteration in any modified and/or redistributed
14   *   versions, and that such modified versions are clearly identified
15   *   as such. No licenses are granted by implication, estoppel or
16   *   otherwise under any patents or trademarks of Motorola, Inc. This
17   *   software is provided on an "AS IS" basis and without warranty.
18   *
19   *   To the maximum extent permitted by applicable law, MOTOROLA
20   *   DISCLAIMS ALL WARRANTIES WHETHER EXPRESS OR IMPLIED, INCLUDING
21   *   IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
22   *   PURPOSE AND ANY WARRANTY AGAINST INFRINGEMENT WITH REGARD TO THE
23   *   SOFTWARE (INCLUDING ANY MODIFIED VERSIONS THEREOF) AND ANY
24   *   ACCOMPANYING WRITTEN MATERIALS.
25   *
26   *   To the maximum extent permitted by applicable law, IN NO EVENT
27   *   SHALL MOTOROLA BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING
28   *   WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS
29   *   INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY
30   *   LOSS) ARISING OF THE USE OR INABILITY TO USE THE SOFTWARE.
31   *
32   *   Motorola assumes no responsibility for the maintenance and support
33   *   of this software
34   **********************************************************************/
35
36  #include "print_scan.h"
37  #include <stdio.h>
38  #include <stdlib.h>
39  #include <ctype.h>
40  #include <stdint.h>
41  #include <stdbool.h>
42  // Keil: suppress ellipsis warning in va_arg usage below
```

```
43  #if defined(__CC_ARM)
44  #pragma diag_suppress 1256
45  #endif
46
47  #define FLAGS_MINUS     (0x01)
48  #define FLAGS_PLUS      (0x02)
49  #define FLAGS_SPACE     (0x04)
50  #define FLAGS_ZERO      (0x08)
51  #define FLAGS_POUND     (0x10)
52
53  #define IS_FLAG_MINUS(a)    (a & FLAGS_MINUS)
54  #define IS_FLAG_PLUS(a)     (a & FLAGS_PLUS)
55  #define IS_FLAG_SPACE(a)    (a & FLAGS_SPACE)
56  #define IS_FLAG_ZERO(a)     (a & FLAGS_ZERO)
57  #define IS_FLAG_POUND(a)    (a & FLAGS_POUND)
58
59  #define LENMOD_h        (0x01)
60  #define LENMOD_l        (0x02)
61  #define LENMOD_L        (0x04)
62  #define LENMOD_hh       (0x08)
63  #define LENMOD_ll       (0x10)
64
65  #define IS_LENMOD_h(a)  (a & LENMOD_h)
66  #define IS_LENMOD_hh(a) (a & LENMOD_hh)
67  #define IS_LENMOD_l(a)  (a & LENMOD_l)
68  #define IS_LENMOD_ll(a) (a & LENMOD_ll)
69  #define IS_LENMOD_L(a)  (a & LENMOD_L)
70
71  #define SCAN_SUPPRESS               0x2
72
73  #define SCAN_DEST_MASK              0x7c
74  #define SCAN_DEST_CHAR              0x4
75  #define SCAN_DEST_STRING            0x8
76  #define SCAN_DEST_SET               0x10
77  #define SCAN_DEST_INT               0x20
78  #define SCAN_DEST_FLOAT             0x30
79
80  #define SCAN_LENGTH_MASK            0x1f00
81  #define SCAN_LENGTH_CHAR            0x100
82  #define SCAN_LENGTH_SHORT_INT       0x200
83  #define SCAN_LENGTH_LONG_INT        0x400
84  #define SCAN_LENGTH_LONG_LONG_INT   0x800
85  #define SCAN_LENGTH_LONG_DOUBLE     0x1000
86
```

```
87  #define SCAN_TYPE_SIGNED          0x2000
88
89  /*!
90   * @brief Scanline function which ignores white spaces.
91   *
92   * @param[in] s The address of the string pointer to update.
93   *
94   * @return String without white spaces.
95   */
96  static uint32_t scan_ignore_white_space(const char **s);
97
98  #if defined(SCANF_FLOAT_ENABLE)
99  static double fnum = 0.0;
100 #endif
101
102 /*!
103  * @brief Converts a radix number to a string and return its length.
104  *
105  * @param[in] numstr    Converted string of the number.
106  * @param[in] nump      Pointer to the number.
107  * @param[in] neg       Polarity of the number.
108  * @param[in] radix     The radix to be converted to.
109  * @param[in] use_caps  Used to identify %x/X output format.
110
111  * @return Length of the converted string.
112  */
113 static int32_t mknumstr (char *numstr, void *nump, int32_t neg, int32_t radix, bool
        use_caps);
114
115 #if defined(PRINTF_FLOAT_ENABLE)
116 /*!
117  * @brief Converts a floating radix number to a string and return its length.
118  *
119  * @param[in] numstr            Converted string of the number.
120  * @param[in] nump              Pointer to the number.
121  * @param[in] radix             The radix to be converted to.
122  * @param[in] precision_width   Specify the precision width.
123
124  * @return Length of the converted string.
125  */
126 static int32_t mkfloatnumstr (char *numstr, void *nump, int32_t radix, uint32_t
        precision_width);
127 #endif
128
```

```
129
130  static void fput_pad ( int32_t c , int32_t curlen , int32_t field_width , int32_t * count ,
         PUTCHAR_FUNC func_ptr , void * farg , int * max_count );
131
132  double modf ( double input_dbl , double  * intpart_ptr );
133
134  #if ! defined ( PRINT_MAX_COUNT )
135  #define n_putchar ( func , chacter , p , count )      func ( chacter , p )
136  #else
137  static int n_putchar ( PUTCHAR_FUNC func_ptr , int chacter , void *p , int * max_count )
138  {
139      int result = 0;
140      if (* max_count )
141      {
142          result = func_ptr ( chacter , p );
143          (* max_count ) --;
144      }
145      return result ;
146  }
147  #endif
148
149  /* FUNCTION ******************************************************************
150   *
151   * Function Name : _doprint
152   * Description   : This function outputs its parameters according to a
153   * formatted string . I/O is performed by calling given function pointer
154   * using following (* func_ptr )( c , farg );
155   *
156   * END ***********************************************************************/
157  int _doprint ( void * farg , PUTCHAR_FUNC func_ptr , int max_count , char * fmt , va_list ap )
158  {
159      /* va_list ap ; */
160      char *p;
161      int32_t c;
162
163      char vstr [33];
164      char * vstrp ;
165      int32_t vlen ;
166
167      int32_t done ;
168      int32_t count = 0;
169      int temp_count = max_count ;
170
171
```

93

```c
172         uint32_t flags_used;
173         uint32_t field_width;
174
175         int32_t ival;
176         int32_t schar, dschar;
177         int32_t *ivalp;
178         char *sval;
179         int32_t cval;
180         uint32_t uval;
181         bool use_caps;
182         uint32_t precision_width;
183         //uint32_t length_modifier = 0;
184 #if defined(PRINTF_FLOAT_ENABLE)
185         double fval;
186 #endif
187
188         if (max_count == -1)
189         {
190             max_count = INT32_MAX - 1;
191         }
192
193         /*
194          * Start parsing apart the format string and display appropriate
195          * formats and data.
196          */
197         for (p = (char *)fmt; (c = *p) != 0; p++)
198         {
199             /*
200              * All formats begin with a '%' marker.  Special chars like
201              * '\n' or '\t' are normally converted to the appropriate
202              * character by the __compiler__.  Thus, no need for this
203              * routine to account for the '\' character.
204              */
205             if (c != '%')
206             {
207                 n_putchar(func_ptr, c, farg, &max_count);
208
209                 count++;
210
211                 /*
212                  * By using 'continue', the next iteration of the loop
213                  * is used, skipping the code that follows.
214                  */
215                 continue;
```

94

```
216          }
217
218          /*
219           * First check for specification modifier flags.
220           */
221          use_caps = true;
222          flags_used = 0;
223          done = false;
224          while (!done)
225          {
226              switch (/* c = */ *++p)
227              {
228                  case '-':
229                      flags_used |= FLAGS_MINUS;
230                      break;
231                  case '+':
232                      flags_used |= FLAGS_PLUS;
233                      break;
234                  case ' ':
235                      flags_used |= FLAGS_SPACE;
236                      break;
237                  case '0':
238                      flags_used |= FLAGS_ZERO;
239                      break;
240                  case '#':
241                      flags_used |= FLAGS_POUND;
242                      break;
243                  default:
244                      /* we've gone one char too far */
245                      --p;
246                      done = true;
247                      break;
248              }
249          }
250
251          /*
252           * Next check for minimum field width.
253           */
254          field_width = 0;
255          done = false;
256          while (!done)
257          {
258              switch (c = *++p)
259              {
```

95

```
260                case '0':
261                case '1':
262                case '2':
263                case '3':
264                case '4':
265                case '5':
266                case '6':
267                case '7':
268                case '8':
269                case '9':
270                    field_width = (field_width * 10) + (c - '0');
271                    break;
272                default:
273                    /* we've gone one char too far */
274                    --p;
275                    done = true;
276                    break;
277            }
278        }
279
280        /*
281         * Next check for the width and precision field separator.
282         */
283        precision_width = 6;
284        if (/* (c = *++p) */ *++p == '.')
285        {
286            /* precision_used = true; */
287
288            /*
289             * Must get precision field width, if present.
290             */
291            precision_width = 0;
292            done = false;
293            while (!done)
294            {
295                switch (c = *++p)
296                {
297                    case '0':
298                    case '1':
299                    case '2':
300                    case '3':
301                    case '4':
302                    case '5':
303                    case '6':
```

```
304                     case '7':
305                     case '8':
306                     case '9':
307                         precision_width = (precision_width * 10) + (c - '0');
308                         break;
309                     default:
310                         /* we've gone one char too far */
311                         --p;
312                         done = true;
313                         break;
314                 }
315             }
316         }
317         else
318         {
319             /* we've gone one char too far */
320             --p;
321         }
322
323         /*
324          * Check for the length modifier.
325          */
326         /* length_modifier = 0; */
327         switch (/* c = */ *++p)
328         {
329             case 'h':
330                 if (*++p != 'h')
331                 {
332                     --p;
333                 }
334                 /* length_modifier |= LENMOD_h; */
335                 break;
336             case 'l':
337                 if (*++p != 'l')
338                 {
339                     --p;
340                 }
341                 /* length_modifier |= LENMOD_l; */
342                 break;
343             case 'L':
344                 /* length_modifier |= LENMOD_L; */
345                 break;
346             default:
347                 /* we've gone one char too far */
```

```
348                     --p;
349                     break;
350             }
351
352             /*
353              * Now we're ready to examine the format.
354              */
355             switch (c = *++p)
356             {
357                 case 'd':
358                 case 'i':
359                     ival = (int32_t)va_arg(ap, int32_t);
360                     vlen = mknumstr(vstr,&ival,true,10,use_caps);
361                     vstrp = &vstr[vlen];
362
363                     if (ival < 0)
364                     {
365                         schar = '-';
366                         ++vlen;
367                     }
368                     else
369                     {
370                         if (IS_FLAG_PLUS(flags_used))
371                         {
372                             schar = '+';
373                             ++vlen;
374                         }
375                         else
376                         {
377                             if (IS_FLAG_SPACE(flags_used))
378                             {
379                                 schar = ' ';
380                                 ++vlen;
381                             }
382                             else
383                             {
384                                 schar = 0;
385                             }
386                         }
387                     }
388                     dschar = false;
389
390                     /*
391                      * do the ZERO pad.
```

```
392                    */
393                   if (IS_FLAG_ZERO(flags_used))
394                   {
395                       if (schar)
396                       {
397                           n_putchar(func_ptr, schar, farg, &max_count);
398                           count++;
399                       }
400                       dschar = true;
401
402                       fput_pad('0', vlen, field_width, &count, func_ptr, farg,
     &max_count);
403                       vlen = field_width;
404                   }
405                   else
406                   {
407                       if (!IS_FLAG_MINUS(flags_used))
408                       {
409                           fput_pad(' ', vlen, field_width, &count, func_ptr, farg,
     &max_count);
410                           if (schar)
411                           {
412                               n_putchar(func_ptr, schar, farg, &max_count);
413                               count++;
414                           }
415                           dschar = true;
416                       }
417                   }
418
419                   /* the string was built in reverse order, now display in */
420                   /* correct order */
421                   if ((!dschar) && schar)
422                   {
423                       n_putchar(func_ptr, schar, farg, &max_count);
424                       count++;
425                   }
426                   goto cont_xd;
427  #if defined(PRINTF_FLOAT_ENABLE)
428             case 'f':
429             case 'F':
430                   fval = (double)va_arg(ap, double);
431                   vlen = mkfloatnumstr(vstr,&fval,10, precision_width);
432                   vstrp = &vstr[vlen];
433
```

```
434                    if (fval < 0)
435                    {
436                        schar = '-';
437                        ++vlen;
438                    }
439                    else
440                    {
441                        if (IS_FLAG_PLUS(flags_used))
442                        {
443                            schar = '+';
444                            ++vlen;
445                        }
446                        else
447                        {
448                            if (IS_FLAG_SPACE(flags_used))
449                            {
450                                schar = ' ';
451                                ++vlen;
452                            }
453                            else
454                            {
455                                schar = 0;
456                            }
457                        }
458                    }
459                    dschar = false;
460                    if (IS_FLAG_ZERO(flags_used))
461                    {
462                        if (schar)
463                        {
464                            n_putchar(func_ptr, schar, farg, &max_count);
465                            count++;
466                        }
467                        dschar = true;
468                        fput_pad('0', vlen, field_width, &count, func_ptr, farg,
       &max_count);
469                        vlen = field_width;
470                    }
471                    else
472                    {
473                        if (!IS_FLAG_MINUS(flags_used))
474                        {
475                            fput_pad(' ', vlen, field_width, &count, func_ptr, farg,
       &max_count);
```

```
476                        if (schar)
477                        {
478                            n_putchar(func_ptr, schar, farg, &max_count);
479                            count++;
480                        }
481                        dschar = true;
482                    }
483                }
484                if (!dschar && schar)
485                {
486                    n_putchar(func_ptr, schar, farg, &max_count);
487                    count++;
488                }
489                goto cont_xd;
490 #endif
491            case 'x':
492                use_caps = false;
493            case 'X':
494                uval = (uint32_t)va_arg(ap, uint32_t);
495                vlen = mknumstr(vstr,&uval,false,16,use_caps);
496                vstrp = &vstr[vlen];
497
498                dschar = false;
499                if (IS_FLAG_ZERO(flags_used))
500                {
501                    if (IS_FLAG_POUND(flags_used))
502                    {
503                        n_putchar(func_ptr, '0', farg, &max_count);
504                        n_putchar(func_ptr, (use_caps ? 'X' : 'x'), farg, &max_count);
505                        count += 2;
506                        /*vlen += 2;*/
507                        dschar = true;
508                    }
509                    fput_pad('0', vlen, field_width, &count, func_ptr, farg,
    &max_count);
510                    vlen = field_width;
511                }
512                else
513                {
514                    if (!IS_FLAG_MINUS(flags_used))
515                    {
516                        if (IS_FLAG_POUND(flags_used))
517                        {
518                            vlen += 2;
```

101

```
519                              }
520                              fput_pad(' ', vlen, field_width, &count, func_ptr, farg,
         &max_count);
521                              if (IS_FLAG_POUND(flags_used))
522                              {
523                                   n_putchar(func_ptr, '0', farg, &max_count);
524                                   n_putchar(func_ptr, (use_caps ? 'X' : 'x'), farg,
         &max_count);
525                                   count += 2;
526
527                                   dschar = true;
528                              }
529                         }
530                    }
531
532                    if ((IS_FLAG_POUND(flags_used)) && (!dschar))
533                    {
534                         n_putchar(func_ptr, '0', farg, &max_count);
535                         n_putchar(func_ptr, (use_caps ? 'X' : 'x'), farg, &max_count);
536                         count += 2;
537                         vlen += 2;
538                    }
539                    goto cont_xd;
540
541               case 'o':
542                    uval = (uint32_t)va_arg(ap, uint32_t);
543                    vlen = mknumstr(vstr,&uval,false,8,use_caps);
544                    goto cont_u;
545               case 'b':
546                    uval = (uint32_t)va_arg(ap, uint32_t);
547                    vlen = mknumstr(vstr,&uval,false,2,use_caps);
548                    goto cont_u;
549               case 'p':
550                    uval = (uint32_t)va_arg(ap, uint32_t);
551                    uval = (uint32_t)va_arg(ap, void *);
552                    vlen = mknumstr(vstr,&uval,false,16,use_caps);
553                    goto cont_u;
554               case 'u':
555                    uval = (uint32_t)va_arg(ap, uint32_t);
556                    vlen = mknumstr(vstr,&uval,false,10,use_caps);
557
558                    cont_u:
559                         vstrp = &vstr[vlen];
560
```

```
561                     if (IS_FLAG_ZERO(flags_used))
562                     {
563                         fput_pad('0', vlen, field_width, &count, func_ptr, farg,
    &max_count);
564                         vlen = field_width;
565                     }
566                     else
567                     {
568                         if (!IS_FLAG_MINUS(flags_used))
569                         {
570                             fput_pad(' ', vlen, field_width, &count, func_ptr, farg,
    &max_count);
571                         }
572                     }
573
574             cont_xd:
575                     while (*vstrp)
576                     {
577                         n_putchar(func_ptr, *vstrp--, farg, &max_count);
578                         count++;
579                     }
580
581                     if (IS_FLAG_MINUS(flags_used))
582                     {
583                         fput_pad(' ', vlen, field_width, &count, func_ptr, farg,
    &max_count);
584                     }
585                 break;
586
587             case 'c':
588                 cval = (char)va_arg(ap, uint32_t);
589                 n_putchar(func_ptr, cval, farg, &max_count);
590                 count++;
591                 break;
592             case 's':
593                 sval = (char *)va_arg(ap, char *);
594                 if (sval)
595                 {
596                     vlen = strlen(sval);
597                     if (!IS_FLAG_MINUS(flags_used))
598                     {
599                         fput_pad(' ', vlen, field_width, &count, func_ptr, farg,
    &max_count);
600                     }
```

```
601                         while (* sval )
602                         {
603                             n_putchar ( func_ptr , * sval ++ , farg , & max_count );
604                             count ++;
605                         }
606                         if ( IS_FLAG_MINUS ( flags_used ))
607                         {
608                             fput_pad ( ' ' , vlen , field_width , & count , func_ptr , farg ,
     & max_count );
609                         }
610                     }
611                 break ;
612             case 'n ':
613                 ivalp = ( int32_t *) va_arg ( ap , int32_t *);
614                 * ivalp = count ;
615                 break ;
616             default :
617                 n_putchar ( func_ptr , c , farg , & max_count );
618                 count ++;
619                 break ;
620         }
621     }
622
623     if ( max_count )
624     {
625         return count ;
626     }
627     else
628     {
629         return temp_count ;
630     }
631 }
632
633 /* FUNCTION *******************************************************************
634  *
635  * Function Name : _sputc
636  * Description   : Writes the character into the string located by the string
637  * pointer and updates the string pointer.
638  *
639  * END ************************************************************************/
640 int _sputc ( int c , void * input_string )
641 {
642     char ** string_ptr = ( char  **) input_string ;
643
```

104

```
644    *(*string_ptr)++ = (char)c;
645    return c;
646  }
647
648  /*FUNCTION*******************************************************************
649   *
650   * Function Name : mknumstr
651   * Description   : Converts a radix number to a string and return its length.
652   *
653   *END***********************************************************************/
654  static int32_t mknumstr (char *numstr, void *nump, int32_t neg, int32_t radix, bool
         use_caps)
655  {
656      int32_t a,b,c;
657      uint32_t ua,ub,uc;
658
659      int32_t nlen;
660      char *nstrp;
661
662      nlen = 0;
663      nstrp = numstr;
664      *nstrp++ = '\0';
665
666      if (neg)
667      {
668          a = *(int32_t *)nump;
669          if (a == 0)
670          {
671              *nstrp = '0';
672              ++nlen;
673              goto done;
674          }
675          while (a != 0)
676          {
677              b = (int32_t)a / (int32_t)radix;
678              c = (int32_t)a - ((int32_t)b * (int32_t)radix);
679              if (c < 0)
680              {
681                  c = ~c + 1 + '0';
682              }
683              else
684              {
685                  c = c + '0';
686              }
```

105

```
687                a = b;
688                *nstrp++ = (char)c;
689                ++nlen;
690            }
691        }
692        else
693        {
694            ua = *(uint32_t *)nump;
695            if (ua == 0)
696            {
697                *nstrp = '0';
698                ++nlen;
699                goto done;
700            }
701            while (ua != 0)
702            {
703                ub = (uint32_t)ua / (uint32_t)radix;
704                uc = (uint32_t)ua - ((uint32_t)ub * (uint32_t)radix);
705                if (uc < 10)
706                {
707                    uc = uc + '0';
708                }
709                else
710                {
711                    uc = uc - 10 + (use_caps ? 'A' : 'a');
712                }
713                ua = ub;
714                *nstrp++ = (char)uc;
715                ++nlen;
716            }
717        }
718    done:
719        return nlen;
720 }
721
722 #if defined(PRINTF_FLOAT_ENABLE)
723 /*FUNCTION**********************************************************************
724  *
725  * Function Name : mkfloatnumstr
726  * Description   : Converts a floating radix number to a string and return
727  * its length, user can specify output precision width.
728  *
729  *END**************************************************************************/
```

```
730  static int32_t mkfloatnumstr ( char *numstr , void *nump , int32_t radix , uint32_t
          precision_width )
731  {
732      int32_t a ,b ,c ,i ;
733      double fa ,fb ;
734      double r , fractpart , intpart ;
735
736      int32_t nlen ;
737      char *nstrp ;
738      nlen = 0;
739      nstrp = numstr ;
740      *nstrp ++ = '\0';
741      r = *( double *) nump ;
742      if (r == 0)
743      {
744          *nstrp = '0';
745          ++ nlen ;
746          goto done ;
747      }
748      fractpart = modf (( double )r , ( double *) & intpart );
749      /* Process fractional part */
750      for (i = 0; i < precision_width ; i ++)
751      {
752          fractpart *= radix ;
753      }
754      //a = ( int32_t ) floor ( fractpart + ( double )0.5);
755      fa = fractpart + ( double )0.5;
756      for (i = 0; i < precision_width ; i ++)
757      {
758          fb = fa / ( int32_t ) radix ;
759          c = ( int32_t )( fa - ( uint64_t ) fb * ( int32_t ) radix );
760          if (c < 0)
761          {
762              c = ~c + 1 + '0';
763          } else
764          {
765              c = c + '0';
766          }
767          fa = fb ;
768          *nstrp ++ = ( char )c ;
769          ++ nlen ;
770      }
771      *nstrp ++ = ( char )'.';
772      ++ nlen ;
```

107

```
773     a = (int32_t)intpart;
774     while (a != 0)
775     {
776         b = (int32_t)a / (int32_t)radix;
777         c = (int32_t)a - ((int32_t)b * (int32_t)radix);
778         if (c < 0)
779         {
780             c = ~c + 1 + '0';
781         }else
782         {
783             c = c + '0';
784         }
785         a = b;
786         *nstrp++ = (char)c;
787         ++nlen;
788     }
789     done:
790     return nlen;
791 }
792 #endif
793
794 static void fput_pad(int32_t c, int32_t curlen, int32_t field_width, int32_t *count,
795     PUTCHAR_FUNC func_ptr, void *farg, int *max_count)
795 {
796     int32_t i;
797
798     for (i = curlen; i < field_width; i++)
799     {
800         func_ptr((char)c, farg);
801         (*count)++;
802     }
803 }
804
805 /*FUNCTION*****************************************************************
806  *
807  * Function Name  : scan_prv
808  * Description    : Converts an input line of ASCII characters based upon a
809  * provided string format.
810  *
811  * END*********************************************************************/
812 int scan_prv(const char *line_ptr, char *format, va_list args_ptr)
813 {
814     uint8_t base;
815     /* Identifier for the format string */
```

108

```
816      char *c = format;
817      const char *s;
818      char temp;
819     /* Identifier for the input string */
820      const char *p = line_ptr;
821     /* flag telling the conversion specification */
822      uint32_t flag = 0 ;
823     /* filed width for the matching input streams */
824      uint32_t field_width;
825     /* how many arguments are assigned except the suppress */
826      uint32_t nassigned = 0;
827     /* how many characters are read from the input streams */
828      uint32_t n_decode = 0;
829
830      int32_t val;
831      char *buf;
832      int8_t neg;
833
834     /* return EOF error before any convernsion */
835      if (*p == '\0')
836      {
837          return EOF;
838      }
839
840     /* decode directives */
841      while ((*c) && (*p))
842      {
843          /* ignore all white-spaces in the format strings */
844          if (scan_ignore_white_space((const char **)&c))
845          {
846              n_decode += scan_ignore_white_space(&p);
847          }
848          else if (*c != '%')
849          {
850              /* Ordinary characters */
851              c++;
852 ordinary:   if (*p == *c)
853              {
854                  n_decode++;
855                  p++;
856                  c++;
857              }
858              else
859              {
```

109

```
860                    /* Match failure. Misalignment with C99, the unmatched
861                     * characters need to be pushed back to stream. HOwever
862                     * , it is deserted now. */
863                    break;
864                }
865            }
866        else
867        {
868            /* convernsion specification */
869            c++;
870            if (*c == '%')
871            {
872                goto ordinary;
873            }
874
875            /* Reset */
876            flag = 0;
877            field_width = 0;
878            base = 0;
879
880            /* Loop to get full conversion specification */
881            while ((*c) && (!(flag & SCAN_DEST_MASK)))
882            {
883                switch (*c)
884                {
885                    case '*':
886                        if (flag & SCAN_SUPPRESS)
887                        {
888                            /* Match failure*/
889                            return nassigned;
890                        }
891                        flag |= SCAN_SUPPRESS;
892                        c++;
893                        break;
894                    case 'h':
895                        if (flag & SCAN_LENGTH_MASK)
896                        {
897                            /* Match failure*/
898                            return nassigned;
899                        }
900                        flag |= SCAN_LENGTH_SHORT_INT;
901
902                        if (c[1] == 'h')
903                        {
```

```
904                                    flag |= SCAN_LENGTH_CHAR;
905                                    c++;
906                                }
907                                c++;
908                                break;
909                        case 'l':
910                                if (flag & SCAN_LENGTH_MASK)
911                                {
912                                    /* Match failure*/
913                                    return nassigned;
914                                }
915                                flag |= SCAN_LENGTH_LONG_INT;
916
917                                if (c[1] == 'l')
918                                {
919                                    flag |= SCAN_LENGTH_LONG_LONG_INT;
920                                    c++;
921                                }
922                                c++;
923                                break;
924 #if defined(ADVANCE)
925                        case 'j':
926                                if (flag & SCAN_LENGTH_MASK)
927                                {
928                                    /* Match failure*/
929                                    return nassigned;
930                                }
931                                flag |= SCAN_LENGTH_INTMAX;
932                                c++
933                        case 'z'
934                                if (flag & SCAN_LENGTH_MASK)
935                                {
936                                    /* Match failure*/
937                                    return nassigned;
938                                }
939                                flag |= SCAN_LENGTH_SIZE_T;
940                                c++;
941                                break;
942                        case 't':
943                                if (flag & SCAN_LENGTH_MASK)
944                                {
945                                    /* Match failure*/
946                                    return nassigned;
947                                }
```

```
948                                    flag |= SCAN_LENGTH_PTRDIFF_T;
949                                    c++;
950                                    break;
951 #endif
952 #if defined(SCANF_FLOAT_ENABLE)
953                          case 'L':
954                              if (flag & SCAN_LENGTH_MASK)
955                              {
956                                  /* Match failure*/
957                                  return nassigned;
958                              }
959                              flag |= SCAN_LENGTH_LONG_DOUBLE;
960                              c++;
961                              break;
962 #endif
963                          case '0':
964                          case '1':
965                          case '2':
966                          case '3':
967                          case '4':
968                          case '5':
969                          case '6':
970                          case '7':
971                          case '8':
972                          case '9':
973                              if (field_width)
974                              {
975                                  /* Match failure*/
976                                  return nassigned;
977                              }
978                              do {
979                                  field_width = field_width * 10 + *c - '0';
980                                  c++;
981                              } while ((*c >= '0') && (*c <= '9'));
982                              break;
983                          case 'd':
984                              flag |= SCAN_TYPE_SIGNED;
985                          case 'u':
986                              base = 10;
987                              flag |= SCAN_DEST_INT;
988                              c++;
989                              break;
990                          case 'o':
991                              base = 8;
```

112

```
992                              flag |= SCAN_DEST_INT;
993                              c++;
994                              break;
995                        case 'x':
996                        case 'X':
997                              base = 16;
998                              flag |= SCAN_DEST_INT;
999                              c++;
1000                             break;
1001                       case 'i':
1002                             base = 0;
1003                             flag |= SCAN_DEST_INT;
1004                             c++;
1005                             break;
1006 #if defined(SCANF_FLOAT_ENABLE)
1007                       case 'a':
1008                       case 'A':
1009                       case 'e':
1010                       case 'E':
1011                       case 'f':
1012                       case 'F':
1013                       case 'g':
1014                       case 'G':
1015                             flag |= SCAN_DEST_FLOAT;
1016                             c++;
1017                             break;
1018 #endif
1019                       case 'c':
1020                             flag |= SCAN_DEST_CHAR;
1021                             if (!field_width)
1022                             {
1023                                  field_width = 1;
1024                             }
1025                             c++;
1026                             break;
1027                       case 's':
1028                             flag |= SCAN_DEST_STRING;
1029                             c++;
1030                             break;
1031 #if defined(ADVANCE) /* [x]*/
1032                       case '[':
1033                             flag |= SCAN_DEST_SET;
1034                             /*Add Set functionality */
1035                             break;
```

113

```
1036 #endif
1037                     default:
1038 #if defined(SCAN_DEBUG)
1039                         printf("Unrecognized expression specifier: %c format: %s,
     number is: %d\r\n", c, format, nassigned);
1040 #endif
1041                         return nassigned;
1042                 }
1043             }
1044
1045             if (!(flag & SCAN_DEST_MASK))
1046             {
1047                 /* Format strings are exausted */
1048                 return nassigned;
1049             }
1050
1051             if (!field_width)
1052             {
1053                 /* Larget then length of a line */
1054                 field_width = 99;
1055             }
1056
1057             /* Matching strings in input streams and assign to argument */
1058             switch (flag & SCAN_DEST_MASK)
1059             {
1060                 case SCAN_DEST_CHAR:
1061                     s = (const char *)p;
1062                     buf = va_arg(args_ptr, char *);
1063                     while ((field_width--) && (*p))
1064                     {
1065                         if (!(flag & SCAN_SUPPRESS))
1066                         {
1067                             *buf++ = *p++;
1068                         }
1069                         else
1070                         {
1071                             p++;
1072                         }
1073                         n_decode++;
1074                     }
1075
1076                     if (((((!(flag)) & SCAN_SUPPRESS) && (s != p))
1077                     {
1078                         nassigned++;
```

114

```
1079                          }
1080                          break;
1081                  case SCAN_DEST_STRING:
1082                          n_decode += scan_ignore_white_space(&p);
1083                          s = p;
1084                          buf = va_arg(args_ptr, char *);
1085                          while ((field_width--) && (*p != '\0') && (*p != ' ') &&
1086                                  (*p != '\t') && (*p != '\n') && (*p != '\r') && (*p !=
     '\v') && (*p != '\f'))
1087                          {
1088                              if (flag & SCAN_SUPPRESS)
1089                              {
1090                                  p++;
1091                              }
1092                              else
1093                              {
1094                                  *buf++ = *p++;
1095                              }
1096                              n_decode++;
1097                          }
1098
1099                          if ((!(flag & SCAN_SUPPRESS)) && (s != p))
1100                          {
1101                              /* Add NULL to end of string */
1102                              *buf = '\0';
1103                              nassigned++;
1104                          }
1105                          break;
1106                  case SCAN_DEST_INT:
1107                          n_decode += scan_ignore_white_space(&p);
1108                          s = p;
1109                          val = 0;
1110                          /*TODO: scope is not testsed */
1111                          if ((base == 0) || (base == 16))
1112                          {
1113                              if ((s[0] == '0') && ((s[1] == 'x') || (s[1] == 'X')))
1114                              {
1115                                  base = 16;
1116                                  if (field_width >= 1)
1117                                  {
1118                                      p += 2;
1119                                      n_decode += 2;
1120                                      field_width -= 2;
1121                                  }
```

```
1122                        }
1123                    }
1124
1125                    if (base == 0)
1126                    {
1127                        if (s[0] == '0')
1128                        {
1129                            base = 8;
1130                        }
1131                        else
1132                        {
1133                            base = 10;
1134                        }
1135                    }
1136
1137                    neg = 1;
1138                    switch (*p)
1139                    {
1140                        case '-':
1141                            neg = -1;
1142                            n_decode++;
1143                            p++;
1144                            field_width--;
1145                            break;
1146                        case '+':
1147                            neg = 1;
1148                            n_decode++;
1149                            p++;
1150                            field_width--;
1151                            break;
1152                        default:
1153                            break;
1154                    }
1155
1156                    while ((*p) && (field_width--))
1157                    {
1158                        if ((*p <= '9') && (*p >= '0'))
1159                        {
1160                            temp = *p - '0';
1161                        }
1162                        else if((*p <= 'f') && (*p >= 'a'))
1163                        {
1164                            temp = *p - 'a' + 10;
1165                        }
```

116

```c
                        else if((*p <= 'F') && (*p >= 'A'))
                        {
                            temp = *p - 'A' + 10;
                        }
                        else
                        {
                            break;
                        }

                        if (temp >= base)
                        {
                            break;
                        }
                        else
                        {
                            val = base * val + temp;
                        }
                        p++;
                        n_decode++;
                    }

                    val *= neg;
                    if (!(flag & SCAN_SUPPRESS))
                    {
                        switch (flag & SCAN_LENGTH_MASK)
                        {
                            case SCAN_LENGTH_CHAR:
                                if (flag & SCAN_TYPE_SIGNED)
                                {
                                    *va_arg(args_ptr, signed char *) = (signed
        char)val;
                                }
                                else
                                {
                                    *va_arg(args_ptr, unsigned char *) = (unsigned
        char)val;
                                }
                                break;
                            case SCAN_LENGTH_SHORT_INT:
                                if (flag & SCAN_TYPE_SIGNED)
                                {
                                    *va_arg(args_ptr, signed short *) = (signed
        short)val;
                                }
```

```
1207                                    else
1208                                    {
1209                                        *va_arg(args_ptr, unsigned short *) = (unsigned
        short)val;
1210                                    }
1211                                    break;
1212                                case SCAN_LENGTH_LONG_INT:
1213                                    if (flag & SCAN_TYPE_SIGNED)
1214                                    {
1215                                        *va_arg(args_ptr, signed long int *) = (signed
        long int)val;
1216                                    }
1217                                    else
1218                                    {
1219                                        *va_arg(args_ptr, unsigned long int *) = (unsigned
        long int)val;
1220                                    }
1221                                    break;
1222                                case SCAN_LENGTH_LONG_LONG_INT:
1223                                    if (flag & SCAN_TYPE_SIGNED)
1224                                    {
1225                                        *va_arg(args_ptr, signed long long int *) =
        (signed long long int)val;
1226                                    }
1227                                    else
1228                                    {
1229                                        *va_arg(args_ptr, unsigned long long int *) =
        (unsigned long long int)val;
1230                                    }
1231                                    break;
1232                                default:
1233                                    /* The default type is the type int */
1234                                    if (flag & SCAN_TYPE_SIGNED)
1235                                    {
1236                                        *va_arg(args_ptr, signed int *) = (signed int)val;
1237                                    }
1238                                    else
1239                                    {
1240                                        *va_arg(args_ptr, unsigned int *) = (unsigned
        int)val;
1241                                    }
1242                                    break;
1243                            }
1244                        nassigned++;
```

```
1245                    }
1246                    break;
1247 #if defined(SCANF_FLOAT_ENABLE)
1248            case SCAN_DEST_FLOAT:
1249                    n_decode += scan_ignore_white_space(&p);
1250                    fnum = strtod(p, (char **)&s);
1251
1252                    if ((fnum == HUGE_VAL) || (fnum == -HUGE_VAL))
1253                    {
1254                        break;
1255                    }
1256
1257                    n_decode += (int)(s) - (int)(p);
1258                    p = s;
1259                    if (!(flag & SCAN_SUPPRESS))
1260                    {
1261                        if (flag & SCAN_LENGTH_LONG_DOUBLE)
1262                        {
1263                            *va_arg(args_ptr, double *) = fnum;
1264                        }
1265                        else
1266                        {
1267                            *va_arg(args_ptr, float *) = (float)fnum;
1268                        }
1269                        nassigned++;
1270                    }
1271                    break;
1272 #endif
1273 #if defined(ADVANCE)
1274            case SCAN_DEST_SET:
1275                    break;
1276 #endif
1277            default:
1278 #if defined(SCAN_DEBUG)
1279                    printf("ERROR: File %s  line: %d\r\n", __FILE__, __LINE__);
1280 #endif
1281                    return nassigned;
1282            }
1283        }
1284    }
1285    return nassigned;
1286 }
1287
1288 /*FUNCTION*********************************************************************
```

```
1289    *
1290    * Function Name : scan_ignore_white_space
1291    * Description   : Scanline function which ignores white spaces.
1292    *
1293    *END**************************************************************************/
1294   static uint32_t scan_ignore_white_space(const char **s)
1295   {
1296       uint8_t count = 0;
1297       uint8_t c;
1298
1299       c = **s;
1300       while ((c == ' ') || (c == '\t') || (c == '\n') || (c == '\r') || (c == '\v') ||
              (c == '\f'))
1301       {
1302           count++;
1303           (*s)++;
1304           c = **s;
1305       }
1306       return count;
1307   }
```

7.22 ../Sources/print_scan.h

```
1  /*
2   * Copyright (c) 2013 - 2014, Freescale Semiconductor, Inc.
3   * All rights reserved.
4   *
5   * Redistribution and use in source and binary forms, with or without modification,
6   * are permitted provided that the following conditions are met:
7   *
8   * o Redistributions of source code must retain the above copyright notice, this list
9   *   of conditions and the following disclaimer.
10  *
11  * o Redistributions in binary form must reproduce the above copyright notice, this
12  *   list of conditions and the following disclaimer in the documentation and/or
13  *   other materials provided with the distribution.
14  *
15  * o Neither the name of Freescale Semiconductor, Inc. nor the names of its
16  *   contributors may be used to endorse or promote products derived from this
17  *   software without specific prior written permission.
18  *
19  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
20  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
21  * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
22  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
23  * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
24  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
25  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
26  * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
27  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
28  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
29  */
30 #ifndef __print_scan_h__
31 #define __print_scan_h__
32
33 #include <stdio.h>
34 #include <stdarg.h>
35 #include <stdint.h>
36 #include <stdbool.h>
37 #include <string.h>
38
39 //#define PRINTF_FLOAT_ENABLE   1
40 //#define PRINT_MAX_COUNT       1
41 //#define SCANF_FLOAT_ENABLE    1
42
```

121

```
43  #ifndef HUGE_VAL
44  #define HUGE_VAL          (99.e99)///wrong value
45  #endif
46
47  typedef int (*PUTCHAR_FUNC)(int a, void *b);
48
49  /*!
50   * @brief This function outputs its parameters according to a formatted string.
51   *
52   * @note I/O is performed by calling given function pointer using following
53   * (*func_ptr)(c,farg);
54   *
55   * @param[in] farg      Argument to func_ptr.
56   * @param[in] func_ptr  Function to put character out.
57   * @param[in] max_count Maximum character count for snprintf and vsnprintf.
58   * Default value is 0 (unlimited size).
59   * @param[in] fmt_ptr   Format string for printf.
60   * @param[in] args_ptr  Arguments to printf.
61   *
62   * @return Number of characters
63   * @return EOF (End Of File found.)
64   */
65  int _doprint(void *farg, PUTCHAR_FUNC func_ptr, int max_count, char  *fmt, va_list ap);
66
67  /*!
68   * @brief Writes the character into the string located by the string pointer and
69   * updates the string pointer.
70   *
71   * @param[in]      c            The character to put into the string.
72   * @param[in, out] input_string This is an updated pointer to a string pointer.
73   *
74   * @return Character written into string.
75   */
76  int _sputc(int c, void * input_string);
77
78  /*!
79   * @brief Converts an input line of ASCII characters based upon a provided
80   * string format.
81   *
82   * @param[in] line_ptr The input line of ASCII data.
83   * @param[in] format   Format first points to the format string.
84   * @param[in] args_ptr The list of parameters.
85   *
86   * @return Number of input items converted and assigned.
```

```
87   * @return IO_EOF - When line_ptr is empty string "".
88   */
89  int scan_prv(const char *line_ptr, char *format, va_list args_ptr);
90
91  #endif
```

```c
/* ****************************************************************** */
/* File name:          cmdmachine_hal.c                             */
/* File description: File containing the functions/methods          */
/*                    interfaces for protocol command machine       */
/* Author name:       ddello                                        */
/* Creation date:     27abr2016                                     */
/* Revision date:     13mai2016                                     */
/* ****************************************************************** */
#include <string.h>
#include <stdio.h>
#include "cmdmachine_hal.h"
#include "LedSwi/ledswi_hal.h"
#include "Buzzer/buzzer_hal.h"
#include "SevenSeg/sevenseg_hal.h"
#include "LCD/lcd_hal.h"
#include "Util/util.h"
#include "Cooler/cooler_hal.h"
#include "TempSensor/tempSensor_hal.h"

#define ERR_STR "ERR\n"
#define ACK_STR "ACK\n"

#define STATE_IDLE 0
#define STATE_LED_CMD 1
#define STATE_BUZZER_CMD 2
#define STATE_SWITCH_CMD 3
#define STATE_SEVENSEG_CMD 4
#define STATE_LCD_CMD 5
#define STATE_COOLER_CMD 6
#define STATE_TEMP_CMD 7
#define STATE_ERR 99

static int iState = STATE_IDLE;

//============================================================================
// IDLE STATE MACHINE
//============================================================================
/**
 * Handles parsing while in IDLE state and checks for transitions
 *
 * @param cpCmdBuffer The start of the command string to parse
 * @param uiSize The size of the command string
```

124

```
43  * @param cpCmdRes Buffer for concatenating the commmand response
44  *
45  * @return The number of characters parsed while in the IDLE state
46  */
47  unsigned int handleIdle(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes){
48    unsigned int uiCounter = 0;
49    while(uiCounter < uiSize && iState == STATE_IDLE){
50      switch(cpCmdBuffer[uiCounter++]){
51        case 'L':
52          iState = STATE_LED_CMD;
53          break;
54        case 'S':
55          iState = STATE_SWITCH_CMD;
56          break;
57        case 'B':
58          iState = STATE_BUZZER_CMD;
59          break;
60        case 'D':
61          iState = STATE_SEVENSEG_CMD;
62          break;
63        case 'P':
64          iState = STATE_LCD_CMD;
65          break;
66        case 'C':
67          iState = STATE_COOLER_CMD;
68          break;
69        case 'T':
70          iState = STATE_TEMP_CMD;
71          break;
72        case ' ':
73        case '\t':
74        case '\r':
75        case '\n':
76        case '\0':
77          break;
78        default:
79          iState = STATE_ERR;
80      }
81    }
82    return uiCounter;
83  }
84
85  //========================================================================
86  // LED CMD STATE MACHINE
```

125

```
87  //=============================================================================
88  /**
89   * Parses character ledNumberReference into a led number
90   *
91   * @param ccLedInput Character between '1' and '4' refering to the desired
92   *                   led number
93   *
94   * @return Led number reference (number between 1 and 4) or -1 in case of invalid
95   *         input
96   */
97  ledswi_pin_type_e parseLedNum(char cLedInput){
98      switch(cLedInput){
99          case '1':
100             return LS_1;
101         case '2':
102             return LS_2;
103         case '3':
104             return LS_3;
105         case '4':
106             return LS_4;
107         default:
108             iState = STATE_ERR;
109             return UNKNOWN;
110     }
111 }
112
113 /**
114  * Handles parsing while in LED_COMMAND state and checks for transitions
115  *
116  * @param cpCmdBuffer The start of the command string to parse
117  * @param uiSize The size of the command string
118  * @param cpCmdRes Buffer for concatenating the command response
119  *
120  * @return The number of characters parsed while in the LED_COMMAND state
121  */
122 unsigned int handleLed(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes){
123     unsigned int uiCounter = 0;
124     ledswi_pin_type_e eLedNum = UNKNOWN;
125     void (*fpLedFunction)(ledswi_pin_type_e);
126     while(uiCounter < uiSize && iState == STATE_LED_CMD){
127         switch(cpCmdBuffer[uiCounter++]){
128             case 'C':
129                 eLedNum = parseLedNum(cpCmdBuffer[uiCounter++]);
130                 fpLedFunction = &ledswi_clearLed;
```

```
131            break;
132         case 'S':
133            eLedNum = parseLedNum(cpCmdBuffer[uiCounter++]);
134            fpLedFunction = &ledswi_setLed;
135            break;
136         case 'R':
137            eLedNum = parseLedNum(cpCmdBuffer[uiCounter++]);
138            if(eLedNum != UNKNOWN){
139               strcat(cpCmdRes, ACK_STR);
140               ledswi_setLed(eLedNum);
141               if(ledswi_getLedStatus(eLedNum) == LED_ON){
142                  strcat(cpCmdRes, "O\n");
143               }else{
144                  strcat(cpCmdRes, "C\n");
145               }
146               iState = STATE_IDLE;
147            }else{
148               iState = STATE_ERR;
149            }
150            return uiCounter;
151            break;
152         default:
153            iState = STATE_ERR;
154      }
155   }
156   if(eLedNum != UNKNOWN){
157      strcat(cpCmdRes, ACK_STR);
158      ledswi_initLed(eLedNum);
159      (*fpLedFunction)(eLedNum);
160      iState = STATE_IDLE;
161   }else{
162      iState = STATE_ERR;
163   }
164   return uiCounter;
165 }
166 //==========================================================================
167 // SWITCH CMD STATE MACHINE
168 //==========================================================================
169 /**
170  * Handles parsing while in SWITCH_COMMAND state and checks for transitions
171  *
172  * @param cpCmdBuffer The start of the command string to parse
173  * @param uiSize The size of the command string
174  * @param cpCmdRes Buffer for concatenating the command response
```

```
175    *
176    * @return The number of characters parsed while in the SWITCH_COMMAND state
177    */
178   unsigned int handleSwitch( char *cpCmdBuffer , unsigned int uiSize , char* cpCmdRes ){
179     unsigned int uiCounter = 0;
180     ledswi_pin_type_e eSwiNum = UNKNOWN ;
181     while( uiCounter < uiSize && iState == STATE_SWITCH_CMD ){
182       switch( cpCmdBuffer[ uiCounter ++]){
183         case '1':
184           eSwiNum = LS_1;
185           break;
186         case '2':
187           eSwiNum = LS_2;
188           break;
189         case '3':
190           eSwiNum = LS_3;
191           break;
192         case '4':
193           eSwiNum = LS_4;
194           break;
195         default:
196           eSwiNum = UNKNOWN;
197           break;
198       }
199       if( eSwiNum !=  UNKNOWN ){
200         strcat( cpCmdRes , ACK_STR );
201         ledswi_initSwitch( eSwiNum );
202         if( ledswi_getSwitchStatus( eSwiNum ) == SWITCH_ON ){
203           strcat( cpCmdRes , "O\n" );
204         }else{
205           strcat( cpCmdRes , "C\n" );
206         }
207         iState = STATE_IDLE ;
208       }else{
209         iState = STATE_ERR ;
210       }
211     }
212     return uiCounter ;
213   }
214
215
216   // ======================================================================
217   // BUZZER CMD STATE MACHINE
218   // ======================================================================
```

```
219 /**
220  * Parses Buzzer command milliseconds input into integer
221  *
222  * @param cpCmdBuffer The start of the Buzzer command milliseconds input string
223  *
224  * @return The parsed number of milliseconds contained in the start of the command
225  *         string or -1 in case of parsing failure
226  */
227 int getBuzzMs(char *cpCmdBuffer){
228   int iMs = -1;
229   if(!sscanf(cpCmdBuffer, "%3d", &iMs)){
230     iMs = -1;
231   }
232   return iMs;
233 }
234
235 /**
236  * Handles parsing while in BUZZER_COMMAND state and checks for transitions
237  *
238  * @param cpCmdBuffer The start of the command string to parse
239  * @param uiSize The size of the command string
240  * @param cpCmdRes Buffer for concatenating the command response
241  *
242  * @return The number of characters parsed while in the BUZZER_COMMAND state
243  */
244 int handleBuzzer(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes){
245   unsigned int uiCounter = 0;
246   int iBuzzMs = -1;
247   if(uiCounter < uiSize){
248     iBuzzMs = getBuzzMs(cpCmdBuffer);
249   }
250   if(iBuzzMs >= 0){
251     strcat(cpCmdRes, ACK_STR);
252     buzzer_initPeriodic(440, iBuzzMs);    //440Hz
253     iState = STATE_IDLE;
254     //Exactly how many characters where read.
255     while(uiCounter < 3 && uiCounter < uiSize && cpCmdBuffer[uiCounter] >= '0' &&
          cpCmdBuffer[uiCounter] <= '9'){
256       uiCounter++;
257     }
258   }else{
259     iState = STATE_ERR;
260   }
261   return uiCounter;
```

```
262  }
263
264
265  //==========================================================================
266  // Seven Segment CMD STATE MACHINE
267  //==========================================================================
268  /**
269   * Parses SevenSeg command hexadecimal input into integer
270   *
271   * @param cpCmdBuffer The start of the Buzzer command hexadecimal input string
272   *
273   * @return The parsed number contained in the start of the command
274   *         string or -1 in case of parsing failure
275   */
276  int getSevenSegHex( char *cpCmdBuffer ){
277    int iCommand = -1;
278    if(!sscanf( cpCmdBuffer, "%4x", &iCommand )){
279      iCommand = -1;
280    }
281    return iCommand;
282  }
283
284  /**
285   * Handles parsing while in SEVSEG_COMMAND state and checks for transitions
286   *
287   * @param cpCmdBuffer The start of the command string to parse
288   * @param uiSize The size of the command string
289   * @param cpCmdRes Buffer for concatenating the command response
290   *
291   * @return The number of characters parsed while in the SEVSEG_COMMAND state
292   */
293  int handleSevenSeg( char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes ){
294    unsigned int uiCounter = 0;
295    int iHex = -1;
296    if( uiCounter < uiSize ){
297      iHex = getSevenSegHex( cpCmdBuffer );
298    }
299    if( iHex >= 0 ){
300      strcat( cpCmdRes, ACK_STR );
301      sevenseg_printHex( iHex );
302      iState = STATE_IDLE;
303      //Exactly how many characters where read.
304      while( uiCounter < 4 && uiCounter < uiSize && (( cpCmdBuffer[ uiCounter ] >= '0' &&
         cpCmdBuffer[ uiCounter ] <= '9') || ( cpCmdBuffer[ uiCounter ] >= 'A' &&
```

```
         cpCmdBuffer[uiCounter] <= 'F')) ){
305         uiCounter++;
306      }
307   }else{
308      iState = STATE_ERR;
309   }
310   return uiCounter;
311 }
312
313 //==========================================================================
314 // Cooler CMD STATE MACHINE
315 //==========================================================================
316 /**
317  * Parses Cooler command hexadecimal input into integer
318  *
319  * @param cpCmdBuffer The start of the Buzzer command hexadecimal input string
320  *
321  * @return The parsed number contained in the start of the command
322  *         string or -1 in case of parsing failure
323  */
324 int getCoolerHex(char *cpCmdBuffer){
325   int iCommand = -1;
326   if(!sscanf(cpCmdBuffer, "%4x", &iCommand)){
327     iCommand = -1;
328   }
329   return iCommand;
330 }
331
332 /**
333  * Handles parsing while in COOLER_CMD state and checks for transitions
334  *
335  * @param cpCmdBuffer The start of the command string to parse
336  * @param uiSize The size of the command string
337  * @param cpCmdRes Buffer for concatenating the command response
338  *
339  * @return The number of characters parsed while in the COOLER_COMMAND state
340  */
341 int handleCooler(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes){
342   unsigned int uiCounter = 0;
343   int iHex = -1;
344   if(uiCounter < uiSize){
345     iHex = getCoolerHex(cpCmdBuffer);
346   }
347   if(iHex >= 0){
```

```
348      strcat(cpCmdRes, ACK_STR);
349      cooler_setVelocity(iHex);
350      iState = STATE_IDLE;
351      //Exactly how many characters where read.
352      while(uiCounter < 4 && uiCounter < uiSize && ((cpCmdBuffer[uiCounter] >= '0' &&
         cpCmdBuffer[uiCounter] <= '9') || (cpCmdBuffer[uiCounter] >= 'A' &&
         cpCmdBuffer[uiCounter] <= 'F')) ){
353        uiCounter++;
354      }
355    }else{
356      iState = STATE_ERR;
357    }
358    return uiCounter;
359 }
360
361 //============================================================================
362 // TEMP CMD STATE MACHINE
363 //============================================================================
364 /**
365  * Handles parsing while in TEMP_COMMAND state and checks for transitions
366  *
367  * @param cpCmdBuffer The start of the command string to parse
368  * @param uiSize The size of the command string
369  * @param cpCmdRes Buffer for concatenating the command response
370  *
371  * @return The number of characters parsed while in the TEMP_COMMAND state
372  */
373 int handleTemp(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes){
374    char iPrintBuff[30];
375    strcat(cpCmdRes, ACK_STR);
376    sprintf(iPrintBuff, "%d", tempSensor_getLastConversionRawResult());
377    strcat(cpCmdRes, iPrintBuff);
378    iState = STATE_IDLE;
379    return 0;
380 }
381
382 //============================================================================
383 // LCD CMD STATE MACHINE
384 //============================================================================
385 /**
386  * Handles parsing while in LCD_COMMAND state and checks for transitions
387  *
388  * @param cpCmdBuffer The start of the command string to parse
389  * @param uiSize The size of the command string
```

```
390    * @param cpCmdRes Buffer for concatenating the command response
391    *
392    * @return The number of characters parsed while in the LCD_COMMAND state
393    */
394   int handleLCD(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes){
395     unsigned int uiCounter = 0;
396     char iPrintBuff[30];
397     if(uiCounter < uiSize){
398       sscanf(cpCmdBuffer, "%30s", iPrintBuff);
399       uiCounter += strlen(iPrintBuff);
400       strcat(cpCmdRes, ACK_STR);
401       lcd_printString(iPrintBuff);
402       iState = STATE_IDLE;
403     }else{
404       iState = STATE_ERR;
405     }
406     return uiCounter;
407   }
408
409   //=============================================================================
410   // ERROR STATE MACHINE
411   //=============================================================================
412   /**
413    * Handles parsing while in ERR state and checks for transitions
414    *
415    * @param cpCmdBuffer The start of the command string to parse
416    * @param uiSize The size of the command string
417    * @param cpCmdRes Buffer for concatenating the command response
418    *
419    * @return The number of characters parsed while in the ERR state
420    */
421   int handleError(char *cpCmdBuffer, unsigned int uiSize, char* cpCmdRes){
422     int uiCounter = 0;
423     while(uiCounter < uiSize && iState == STATE_ERR){
424       switch(cpCmdBuffer[uiCounter++]){
425         case ' ':
426         case '\t':
427         case '\r':
428         case '\n':
429         case '\0':
430           iState = STATE_IDLE;
431           break;
432         default:
433           break;
```

133

```
434        }
435      }
436      strcat ( cpCmdRes , ERR_STR ) ;
437      return uiCounter ;
438  }
439
440
441  /**
442   * Interpret all commands in the given command buffer.
443   * Will trigger the command execution an format an output string for printing
444   * Ignores blanks (\r,\n, ,\t,\0) between commands.
445   *
446   * For each valid command, the string ACK will be concatenated to the response string
447   * followed by a line-break and the command response (if any).
448   *
449   * If invalid command syntax is found ERR will be concatenated to the response string
450   * and all the characters before the next blank (\r,\n, ,\t,\0) will be ignored.
451   *
452   * @param cpCmdBuffer Pointer to command buffer
453   * @param uiSize Size of the command buffer
454   * @param cpCmdRes Pointer for response string.
455   *
456   * @after cpCmdRes with the response string for this cmdBuffer ,
457   *        this will contain a ACK for each valid commmand in the cmdBuffer
458   *        followed by a line break and the command output (if any).
459   *        If an invalid command syntax is found the output string will contain
460   *        an ERR\n for that command and all characters before the next blank
461   *        (\r,\n, ,\t,\0) will be ignored.
462   */
463  void cmdmachine_interpretCmdBuffer ( char *cpCmdBuffer , unsigned int uiSize , char*
         cpCmdRes ){
464      iState = STATE_IDLE ;
465      *cpCmdRes = '\0'; //Start response as an empty string.
466      unsigned int uiCounter = 0;
467      while ( uiCounter < uiSize ){
468        switch ( iState ){
469          case STATE_IDLE :
470            uiCounter += handleIdle (& cpCmdBuffer [ uiCounter ], uiSize - uiCounter , cpCmdRes );
471            break ;
472          case STATE_LED_CMD :
473            uiCounter += handleLed (& cpCmdBuffer [ uiCounter ], uiSize - uiCounter , cpCmdRes );
474            break ;
475          case STATE_SWITCH_CMD :
```

134

```c
476        uiCounter += handleSwitch(&cpCmdBuffer[uiCounter], uiSize - uiCounter,
       cpCmdRes);
477          break;
478        case STATE_BUZZER_CMD:
479          uiCounter += handleBuzzer(&cpCmdBuffer[uiCounter], uiSize - uiCounter,
       cpCmdRes);
480          break;
481        case STATE_SEVENSEG_CMD:
482          uiCounter += handleSevenSeg(&cpCmdBuffer[uiCounter], uiSize - uiCounter,
       cpCmdRes);
483          break;
484        case STATE_LCD_CMD:
485          uiCounter += handleLCD(&cpCmdBuffer[uiCounter], uiSize - uiCounter, cpCmdRes);
486          break;
487        case STATE_COOLER_CMD:
488          uiCounter += handleCooler(&cpCmdBuffer[uiCounter], uiSize - uiCounter,
       cpCmdRes);
489          break;
490        case STATE_TEMP_CMD:
491          uiCounter += handleTemp(&cpCmdBuffer[uiCounter], uiSize - uiCounter, cpCmdRes);
492          break;
493        case STATE_ERR:
494          uiCounter += handleError(&cpCmdBuffer[uiCounter], uiSize - uiCounter,
       cpCmdRes);
495          break;
496      }
497    }
498  }
```

7.24 ../Sources/Protocolo/cmdmachine_hal.h

```c
/* ***************************************************************** */
/* File name:        cmdmachine_hal.h                               */
/* File description: Header file containing the functions/methods    */
/*                   interfaces for protocol command machine         */
/* Author name:      ddello                                          */
/* Creation date:    27abr2016                                       */
/* Revision date:    27abr2016                                       */
/* ***************************************************************** */

#ifndef SOURCES_CMDMACHINE_HAL_H_
#define SOURCES_CMDMACHINE_HAL_H_

/**
 * Interpret all commands in the given command buffer.
 * Will trigger the command execution an format an output string for printing
 * Ignores blanks (\r,\n, ,\t,\0) between commands.
 *
 * For each valid command, the string ACK will be concatenated to the response string
 * followed by a line-break and the command response (if any).
 *
 * If invalid command syntax is found ERR will be concatenated to the response string
 * and all the characters before the next blank (\r,\n, ,\t,\0) will be ignored.
 *
 * @param cpCmdBuffer Pointer to command buffer
 * @param uiSize Size of the command buffer
 * @param cpCmdRes Pointer for response string.
 *
 * @after cpCmdRes with the response string for this cmdBuffer,
 *         this will contain a ACK for each valid commmand in the cmdBuffer
 *         followed by a line break and the command output (if any).
 *         If an invalid command syntax is found the output string will contain
 *         an ERR\n for that command and all characters before the next blank
 *         (\r,\n, ,\t,\0) will be ignored.
 */
void cmdmachine_interpretCmdBuffer(char *cpCmdBuffer, unsigned int uiSize, char*
    cpCmdRes);

#endif /* SOURCES_CMDMACHINE_HAL_H_ */
```

7.25 ../Sources/Serial/serial_hal.c

```
1  /* ******************************************************************* */
2  /* File name:        serial_hal.c                                      */
3  /* File description: File containing the functions/methods             */
4  /*                   interfaces for serial communication               */
5  /* Author name:      ddello                                            */
6  /* Creation date:    27abr2016                                         */
7  /* Revision date:    27abr2016                                         */
8  /* ******************************************************************* */
9
10 #include "serial_hal.h"
11 #include "KL25Z/es670_peripheral_board.h"
12 #include "fsl_clock_manager.h"
13 #include "fsl_debug_console.h"
14
15
16 /* The UART to use for debug messages */
17 #ifndef BOARD_DEBUG_UART_INSTANCE
18     #define BOARD_DEBUG_UART_INSTANCE   0
19     #define BOARD_DEBUG_UART_BASEADDR   UART0
20 #endif
21 #ifndef BOARD_DEBUG_UART_BAUD
22     #define BOARD_DEBUG_UART_BAUD       9600
23 #endif
24
25
26 /**
27  * Initialize the serial device configuration
28  */
29 void serial_initUart(void){
30     /* LPSCI0 */
31     /* UART0_RX */
32   PORT_PCR_REG(PORTA , 1) = PORT_PCR_MUX(2U);
33     /* UART0_TX */
34   PORT_PCR_REG(PORTA , 2) = PORT_PCR_MUX(2U);
35
36
37     /* Select different clock source for LPSCI */
38     CLOCK_SYS_SetLpsciSrc(BOARD_DEBUG_UART_INSTANCE , kClockLpsciSrcPllFllSel);
39
40     /* init the debug console */
41     DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE , BOARD_DEBUG_UART_BAUD ,
     kDebugConsoleLPSCI);
```

```
42 }
43

44
45 /**
46  * Write buffer content to serial port
47  *
48  * @param cpBuffer Pointer to the start of the buffer
49  * @param uiSize Number of characters to write
50  */
51 void serial_sendBuffer( char *cpBuffer, unsigned int uiSize){
52   while( uiSize --){
53       while (!UART0_BRD_S1_TDRE(UART0)){
54          //Wait for room in the transmission buffer
55       }
56       UART0_D = (* cpBuffer ++);
57   }
58 }
59

60
61 /**
62  * Receive content from serial port to buffer.
63  *
64  * This function will read until one of {\n, \0} is found
65  *
66  * @param cpBuffer Pointer to the start of the buffer
67  * @param uiSize Maximum number of characters to be read from serial port
68  *
69  * @return The number of characters actually read if the successful,
70  *          -1 in case of buffer overRun
71  */
72 int serial_recieveBuffer( char *cpBuffer, unsigned int uiSize){
73   int ret = 0;
74   while( uiSize --){
75     while(!UART0_BRD_S1_RDRF(UART0)){
76       //Wait for transmission buffer to be filled
77     }
78     *cpBuffer = UART0_D;
79     ret ++;
80     //In case of a buffer overRun
81     if(UART0_BRD_S1_OR(UART0)){
82       //Clear OR flag so we can continue to receive
83       UART0_BWR_S1_OR(UART0,1U);
84       return -1;
85     }
```

```
86
87     switch (* cpBuffer ){
88        case '\n':
89        case '\r':
90          *(++ cpBuffer ) = '\0';
91        case '\0':
92          return ret ;
93        default :
94          cpBuffer ++;
95      }
96   }
97   * cpBuffer = '\0';
98   return ret ;
99 }
```

7.26 ../Sources/Serial/serial_hal.h

```
1  /* ***************************************************************** */
2  /* File name:         serial_hal.h                                   */
3  /* File description: Header file containing the functions/methods     */
4  /*                   interfaces for serial communication            */
5  /* Author name:      ddello                                         */
6  /* Creation date:    27abr2016                                       */
7  /* Revision date:    27abr2016                                       */
8  /* ***************************************************************** */
9
10 #ifndef SOURCES_SERIAL_HAL_H_
11 #define SOURCES_SERIAL_HAL_H_
12
13 /**
14  * Initialize the serial device configuration
15  */
16 void serial_initUart(void);
17
18
19 /**
20  * Write buffer content to serial port
21  *
22  * @param cpBuffer Pointer to the start of the buffer
23  * @param uiSize Number of characters to write
24  */
25 void serial_sendBuffer(char *cpBuffer, unsigned int uiSize);
26
27
28 /**
29  * Receive content from serial port to buffer.
30  *
31  * This function will read until one of {\n, \0} is found
32  *
33  * @param cpBuffer Pointer to the start of the buffer
34  * @param uiSize Maximum number of characters to be read from serial port
35  *
36  * @return The number of characters actually read if the successful,
37  *         -1 in case of buffer overRun
38  */
39 int serial_recieveBuffer(char *cpBuffer, unsigned int uiSize);
40
41 #endif /* SOURCES_SERIAL_HAL_H_ */
```

7.27 ../Sources/SevenSeg/sevenseg_hal.c

```c
/* ******************************************************************** */
/* File name:        sevenseg_hal.c                                     */
/* File description: File containing the functions/methods             */
/*                   for handling SEVEN SEGMENT DISPLAY                 */
/*                    from the peripheral board                        */
/* Author name:      ddello                                            */
/* Creation date:    18mar2016                                         */
/* Revision date:    13abr2016                                         */
/* ******************************************************************** */

#include "GPIO/gpio_hal.h"
#include "sevenseg_hal.h"
#include "math.h"
#include "KL25Z/es670_peripheral_board.h"
#include "PIT/pit_hal.h"

#define SEV_SEG_SEGMENT_MASK GPIO_HIGH << SEGA_PIN | GPIO_HIGH << SEGB_PIN | GPIO_HIGH
    << SEGC_PIN | GPIO_HIGH << SEGD_PIN | GPIO_HIGH << SEGE_PIN | GPIO_HIGH << SEGF_PIN
    | GPIO_HIGH << SEGG_PIN | GPIO_HIGH << SEGDP_PIN
#define SEV_SEG_DISP_MASK GPIO_HIGH << SEG_DISP1_PIN | GPIO_HIGH << SEG_DISP2_PIN |
    GPIO_HIGH << SEG_DISP3_PIN | GPIO_HIGH << SEG_DISP4_PIN

#define SEVEN_SEG_PIT_PERIOD  3125    /*3.125ms*/

static unsigned short usIsHex = 0;
static unsigned int uiPrintVal = 0;

static seven_segment_seg_type_e epSeg_Matrix[MAX_DISP_NUMBER][MAX_SEGMENT_NUMBER+1];
/**
 * Interrupt handler for updating in display configuration
 */
void _sevenseg_interrupt_handler(void){
  static seven_segment_disp_type_e epDisplays[] = {DISP_1, DISP_2, DISP_3, DISP_4};
  static volatile unsigned short usCur_disp = 0;
  sevenseg_setSegs(epSeg_Matrix[usCur_disp]);
  sevenseg_setDisp(epDisplays[usCur_disp]);
  usCur_disp = (usCur_disp+1)%MAX_DISP_NUMBER;
  pit_mark_interrupt_handled(SEV_SEG_PIT_TIMER_NUMB);
}


/**
 * Initialize the seven segment display
```

```
40  */
41  void sevenseg_init(void){
42    GPIO_UNGATE_PORT(SEV_SEG_PORT_ID);
43
44    // Init the Seven Segment segment control pins as OUTPUT
45    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGA_PIN, GPIO_OUTPUT);
46    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGB_PIN, GPIO_OUTPUT);
47    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGC_PIN, GPIO_OUTPUT);
48    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGD_PIN, GPIO_OUTPUT);
49    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGE_PIN, GPIO_OUTPUT);
50    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGF_PIN, GPIO_OUTPUT);
51    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGG_PIN, GPIO_OUTPUT);
52    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEGDP_PIN, GPIO_OUTPUT);
53    // Init the Seven Segment segment display pins as OUTPUT
54    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEG_DISP1_PIN, GPIO_OUTPUT);
55    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEG_DISP2_PIN, GPIO_OUTPUT);
56    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEG_DISP3_PIN, GPIO_OUTPUT);
57    GPIO_INIT_PIN(SEV_SEG_PORT_ID, SEG_DISP4_PIN, GPIO_OUTPUT);
58
59    sevenseg_printDec(0);
60    //Init pit interrupts
61    pit_enable();
62    //Init timer 0
63    pit_start_timer_interrupt(SEV_SEG_PIT_TIMER_NUMB, SEVEN_SEG_PIT_PERIOD,
         &_sevenseg_interrupt_handler);
64  }
65
66
67  /**
68   * Sets only the selected segments as high. Setting the others as low
69   * @param epDet_segments = Array with the segments that should be set as on (Last
         element should be SEG_END)
70   */
71  void sevenseg_setSegs(seven_segment_seg_type_e* epSet_segments){
72    //Clear all segments.
73    GPIO_WRITE_MASK(SEV_SEG_PORT_ID, SEV_SEG_SEGMENT_MASK, GPIO_LOW);
74    //Set the selected segments to high
75    for(unsigned short usCounter = 0; epSet_segments[usCounter] != SEG_END; usCounter++){
76      GPIO_WRITE_PIN(SEV_SEG_PORT_ID, epSet_segments[usCounter], GPIO_HIGH);
77    }
78  }
79
80  /**
81   * Shows the value written in the segment pins to the
```

```
82    *   given  display  after  clearing  the  others
83    *  @param  eDisplay  the  display  to  initialize.
84    */
85   void  sevenseg_setDisp ( seven_segment_disp_type_e  eDisplay ){
86     // Clear  all  displays
87     GPIO_WRITE_MASK ( SEV_SEG_PORT_ID ,  SEV_SEG_DISP_MASK ,  GPIO_LOW );
88     // Activate  the  selected  display
89     GPIO_WRITE_PIN ( SEV_SEG_PORT_ID ,  eDisplay ,  GPIO_HIGH );
90   }
91
92   /**
93    *  Shows  the  passed  value  in  hexadecimal  format  in  the  seven  segment  display.
94    *  @param  uiHex  the  value  to  be  printed
95    */
96   void  sevenseg_printHex ( unsigned  int  uiHex ){
97     for ( unsigned  short  usCur_disp  =  0;  usCur_disp  <  MAX_DISP_NUMBER;  usCur_disp ++){
98       sevenseg_hex2segArray ( uiHex/pow ( 16 , MAX_DISP_NUMBER -1- usCur_disp ) ,
         epSeg_Matrix [ usCur_disp ]);
99     }
100  }
101
102  /**
103   *  Shows  the  passed  value  in  decimal  format  in  the  seven  segment  display.
104   *  @param  uiDec  the  value  to  be  printed
105   */
106  void  sevenseg_printDec ( unsigned  int  uiDec ){
107    for ( unsigned  short  usCur_disp  =  0;  usCur_disp  <  MAX_DISP_NUMBER;  usCur_disp ++){
108      sevenseg_dec2segArray ( uiDec/pow ( 10 , MAX_DISP_NUMBER -1- usCur_disp ) ,
         epSeg_Matrix [ usCur_disp ]);
109    }
110  }
111
112  /**
113   *  Converts  the  less  significative  decimal  digit  of  the  argument  into  it's  seven
114   *  segment  display  configuration
115   *  @param  usDec  the  value  to  be  converted  (-1  if  none  should  be  displayed )
116   *  @param  epRet  address  for  results  (should  be  a  allocated  array  of  minimal  9  elements )
117   *
118   *  @return  epRet
119   */
120  seven_segment_seg_type_e*  sevenseg_dec2segArray ( unsigned  short  usDec ,
         seven_segment_seg_type_e*  epRet ){
121    if ( usDec  <  0 ){
122      epRet [0]  =  SEG_END ;
```

143

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

```
167      //{SEG_A,SEG_G,SEG_C,SEG_D,SEG_E,SEG_F,SEG_END}
168      epRet[1] = SEG_G;
169      epRet[6] = SEG_END;
170      break;
171    case 7:
172      //{SEG_A,SEG_B,SEG_C,SEG_END}
173      epRet[3] = SEG_END;
174      break;
175    case 8:
176      //{SEG_A,SEG_B,SEG_C,SEG_D,SEG_E,SEG_F,SEG_G,SEG_END}
177      break;
178    case 9:
179      //SEG_A,SEG_B,SEG_C,SEG_F,SEG_G,SEG_END}
180      epRet[3] = SEG_F;
181      epRet[4] = SEG_G;
182      epRet[5] = SEG_END;
183      break;
184    }
185    return epRet;
186 }
187
188 /**
189  * Converts the less significative hexadecimal digit of the argument into it's seven
190  * segment display configuration
191  * @param usHex the value to be converted (-1 if none should be displayed)
192  * @param epRet address for results (should be a allocated array of minimal 9 elements)
193  *
194  * @return epRet
195  */
196 seven_segment_seg_type_e* sevenseg_hex2segArray(unsigned short usHex,
        seven_segment_seg_type_e* epRet){
197    if(usHex < 0){
198      epRet[0] = SEG_END;
199      return epRet;
200    }
201    epRet[0] = SEG_A;
202    epRet[1] = SEG_B;
203    epRet[2] = SEG_C;
204    epRet[3] = SEG_D;
205    epRet[4] = SEG_E;
206    epRet[5] = SEG_F;
207    epRet[6] = SEG_G;
208    epRet[7] = SEG_END;
209    switch(usHex%16){
```

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

```
254        epRet[3] = SEG_G;
255        epRet[4] = SEG_END;
256        break;
257    }
258    return epRet;
259 }
```

7.28 ../Sources/SevenSeg/sevenseg_hal.h

```c
/* **************************************************************** */
/* File name:        sevenseg_hal.h                                 */
/* File description: Header file containing the functions/methods   */
/*                   interfaces for handling SEVEN SEGMENT DISPLAY   */
/*                    from the peripheral board                     */
/* Author name:      ddello                                         */
/* Creation date:    18mar2016                                      */
/* Revision date:    13abr2016                                      */
/* **************************************************************** */

#ifndef SOURCES_SEVEN_SEGMENT_HAL_H_
#define SOURCES_SEVEN_SEGMENT_HAL_H_

#include "KL25Z/es670_peripheral_board.h"

#define MAX_SEGMENT_NUMBER 8
#define MAX_DISP_NUMBER 4


typedef enum
{
    SEG_A =  SEGA_PIN,
    SEG_B =  SEGB_PIN,
    SEG_C =  SEGC_PIN,
    SEG_D =  SEGD_PIN,
    SEG_E =  SEGE_PIN,
    SEG_F =  SEGF_PIN,
    SEG_G =  SEGG_PIN,
    SEG_DP = SEGDP_PIN,
  SEG_END = -1
} seven_segment_seg_type_e;

typedef enum
{
    DISP_1 =   SEG_DISP1_PIN,
  DISP_2 =   SEG_DISP2_PIN,
  DISP_3 =   SEG_DISP3_PIN,
  DISP_4 =   SEG_DISP4_PIN,
} seven_segment_disp_type_e;

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

7.29 ../Sources/Tacometro/tacometro_hal.c

```c
/* ****************************************************************** */
/* File name:        tacometro_hal.c                                 */
/* File description: File containing the functions/methods           */
/*                   for tacometer                                   */
/* Author name:      ddello                                          */
/* Creation date:    13maio2016                                      */
/* Revision date:    13maio2016                                      */
/* ****************************************************************** */
#include "tacometro_hal.h"
#include "KL25Z/es670_peripheral_board.h"

#define TACOMETRO_SCALER 7

/**
 * Initialize the tacometer module
 */
void tacometro_init(void){
  /* Init tacometer pin */
  SIM_SCGC5 |= SIM_SCGC5_PORTE(1u);
  PORT_PCR_REG(TACOMETRO_PORT_BASE_PNT , TACOMETRO_PIN) =
    PORT_PCR_MUX(TACOMETRO_PIN_ALT);

  /* INIT TPM */
  //Init TPM Clock source as OSCERCLOCK
  SIM_WR_SOPT2_TPMSRC(SIM,0x2);
  //TPM0 ClockIn0
  SIM_WR_SOPT4_TPM0CLKSEL(SIM, 0x0);
  //Enable clock to TPM0
  SIM_SCGC6 |= SIM_SCGC6_TPM0(0x1);
  //Select External Clock Source for TPM module
  TPM_WR_SC_CMOD(TACOMETRO_TPM_BASE, 0x2);
  //Prescale 1:1
  TPM_WR_SC_PS(TACOMETRO_TPM_BASE, 0x0);
  //Init mod as maximum value
  TPM_WR_MOD(TACOMETRO_TPM_BASE, 0xFFFF);
}


/**
 * @param uiPeriod Time ellapsed since last getSpeedCall in ms
 *
 * @return current tacometer speed in rps
```

151

```
42    */
43  unsigned int tacometro_getSpeed(unsigned int uiPeriod){
44    unsigned int ret = TPM_RD_CNT(TACOMETRO_TPM_BASE)/TACOMETRO_SCALER*1000/uiPeriod;
45    TPM_WR_CNT(TACOMETRO_TPM_BASE, 0x0);  /*Reset counter */
46    return ret;
47  }
```

7.30 ../Sources/Tacometro/tacometro_hal.h

```c
/* ************************************************************* */
/* File name:        tacometro_hal.h                           */
/* File description: Header file containing the functions/methods  */
/*                   interfaces for tacometer                  */
/* Author name:      ddello                                    */
/* Creation date:    13maio2016                                */
/* Revision date:    13maio2016                                */
/* ************************************************************* */

#ifndef SOURCES_TACOMETRO_HAL_H_
#define SOURCES_TACOMETRO_HAL_H_

/**
 * Initialize the tacometer module
 */
void tacometro_init(void);



/**
 * @param uiPeriod Time ellapsed since last getSpeedCall in ms
 *
 * @return current tacometer speed in rps
 */
unsigned int tacometro_getSpeed(unsigned int uiPeriod);

#endif /* SOURCES_TACOMETRO_HAL_H_ */
```

7.31 ../Sources/TempSensor/tempSensor_hal.c

```c
/* ***************************************************************** */
/* File name:         tempSesnor_hal.c                              */
/* File description: File containing the functions/methods          */
/*                    for temperature sensor                        */
/* Author name:       ddello                                        */
/* Creation date:    16jun2016                                      */
/* Revision date:    16jun2016                                      */
/* ***************************************************************** */

#include "tempSensor_hal.h"
#include "ADC/adc.h"

#define TEMP_ADC_TRANSF_EQ_PARAM_A 0.4073
#define TEMP_ADC_TRANSF_EQ_PARAM_B -123.75

static uint16_t rawReading = 0;

/**
 * Init temperature sensor
 */
void tempSensor_init(){
  adc_initADCModule();
}

/**
 * Updates temperature sensor readings and starts a new conversion
 */
void tempSensor_executeTask(){
  if(tempSensor_isConversionFinished()){
    rawReading = adc_getConversionValue();
  }
  adc_initConvertion();
}

/**
 * Checks if last triggered conversion is finished
 *
 * @return 1 if conversion is complete, 0 otherwise
 */
short tempSensor_isConversionFinished(){
  return adc_isAdcDone();
}
```

154

```
43
44 /**
45  * @return Raw result from last successful conversion
46  */
47 uint16_t tempSensor_getLastConversionRawResult(){
48    return rawReading;
49 }
50
51 /**
52  * @return Last read temperature (in Celcius Degrees)
53  */
54 int tempSensor_getLastConversionTemperature(){
55    return TEMP_ADC_TRANSF_EQ_PARAM_A * tempSensor_getLastConversionRawResult() +
         TEMP_ADC_TRANSF_EQ_PARAM_B;
56 }
```

7.32 ../Sources/TempSensor/tempSensor_hal.h

```
1  /* ***************************************************************** */
2  /* File name:        tempSesnor_hal.h                              */
3  /* File description: Header file containing the functions/methods   */
4  /*                   for temperature sensor                        */
5  /* Author name:      ddello                                        */
6  /* Creation date:    16jun2016                                     */
7  /* Revision date:    16jun2016                                     */
8  /* ***************************************************************** */
9
10 #include <stdint.h>
11
12 /**
13  * Init temperature sensor
14  */
15 void tempSensor_init();
16
17 /**
18  * Updates temperature sensor readings and starts a new conversion
19  */
20 void tempSensor_executeTask();
21
22 /**
23  * Checks if last triggered conversion is finished
24  *
25  * @return 1 if conversion is complete, 0 otherwise
26  */
27 short tempSensor_isConversionFinished();
28
29 /**
30  * @return Raw result from last successful conversion
31  */
32 uint16_t tempSensor_getLastConversionRawResult();
33
34 /**
35  * @return Last read temperature (in Celcius Degrees)
36  */
37 int tempSensor_getLastConversionTemperature();
```

7.33 ../Sources/Util/tc_hal.c

```c
/* File name:        tc_hal.c                                    */
/* File description: This file has a couple of useful functions to   */
/*                   timer and counter hardware abstraction layer    */
/*                                                              */
/* Author name:      dloubach                                   */
/* Creation date:    23out2015                                  */
/* Revision date:    25fev2016                                  */
/* ************************************************************** */

#include "tc_hal.h"

/* system includes */
#include "fsl_lptmr_driver.h"

#include "fsl_clock_manager.h"
#include "fsl_port_hal.h"
#include "fsl_gpio_hal.h"


/* LPTMR configurations */
lptmr_user_config_t lptmrConfig =
{
    .timerMode             = kLptmrTimerModeTimeCounter,
    .freeRunningEnable     = false,
    .prescalerEnable       = true,
    .prescalerClockSource  = kClockLptmrSrcLpoClk,
    .prescalerValue        = kLptmrPrescalerDivide2,
    .isInterruptEnabled    = true,
};

/* LPTMR driver state information */
lptmr_state_t lptmrState;


/* LPTMR IRQ handler that would cover the same name's APIs in startup code */
/* Do not edit this part */
void LPTMR0_IRQHandler(void)
{
    LPTMR_DRV_IRQHandler(0U);
}


```

```
43
/* dummy code for hardware debug purpose */
void tc_testDelaySetup(void)
{
    CLOCK_SYS_EnablePortClock(PORTE_IDX);
    PORT_HAL_SetMuxMode(PORTE, 0u, kPortMuxAsGpio);
    GPIO_HAL_SetPinDir(PTE, 0u, kGpioDigitalOutput);
    GPIO_HAL_ClearPinOutput(PTE, 0u);
}
/* dummy code for hardware debug purpose */
void tc_TestDelay(void)
{
    static char cTest = 0;

    if(!cTest)
        GPIO_HAL_ClearPinOutput(PTE, 0u);
    else
        GPIO_HAL_SetPinOutput(PTE, 0u);

    cTest ^= 1u;
}
/* dummy code for hardware debug purpose */
void tc_lptmrCallBack(void)
{
    tc_TestDelay();
}


/* ************************************************* */
/* Method name:        tc_installLptmr              */
/* Method description: Low power timer 0            */
/*                     initialization and start     */
/* Input params:       uiTimeInUs:                  */
/*                     time in micro seconds        */
/*                     tUserCallback                */
/*                     function pointer to be called*/
/*                     when counter achieves        */
/*                     uiTimeInUs                    */
/* Output params:      n/a                          */
/* ************************************************* */
void tc_installLptmr0(uint32_t uiTimeInUs, lptmr_callback_t tUserCallback)
{
    /* Initialize LPTMR */
    LPTMR_DRV_Init(LPTMR0_IDX, &lptmrState, &lptmrConfig);
```

```
87
88      /* Set timer period for TMR_PERIOD mili seconds */
89      LPTMR_DRV_SetTimerPeriodUs(LPTMR0_IDX, uiTimeInUs);
90
91      /* Install interrupt call back function for LPTMR */
92      LPTMR_DRV_InstallCallback(LPTMR0_IDX, tUserCallback);
93
94      /* Start LPTMR */
95      LPTMR_DRV_Start(LPTMR0_IDX);
96 }
```

7.34 ../Sources/Util/tc_hal.h

```c
/* ***************************************************************** */
/* File name:        tc_hal.c                                       */
/* File description: Header file containing the functions/methods   */
/*                   interfaces for handling timers and counter     */
/*                   from the FRDM-KL25Z board                      */
/* Author name:      dloubach                                       */
/* Creation date:    23out2015                                      */
/* Revision date:    25fev2016                               s      */
/* ***************************************************************** */

#ifndef SOURCES_TC_HAL_H_
#define SOURCES_TC_HAL_H_

#include "fsl_lptmr_driver.h"

/* ********************************************** */
/* Method name:        tc_installLptmr           */
/* Method description: Low power timer 0          */
/*                     initialization and start   */
/* Input params:       uiTimeInUs:                */
/*                     time in micro seconds      */
/*                     tUserCallback              */
/*                     function pointer to be called */
/*                     when counter achieves       */
/*                     uiTimeInUs                  */
/* Output params:      n/a                        */
/* ********************************************** */
void tc_installLptmr0(uint32_t uiTimeInUs, lptmr_callback_t tUserCallback);


#endif /* SOURCES_TC_HAL_H_ */
```

7.35 ../Sources/Util/util.c

```c
/* *************************************************************** */
/* File name:       util.c                                        */
/* File description: This file has a couple of useful functions to */
/*                   make programming more productive             */
/*                                                                */
/*                   Remarks: The soft delays consider            */
/*                   core clock @ 40MHz                           */
/*                   bus clock @ 20MHz                            */
/*                                                                */
/* Author name:     dloubach                                      */
/* Creation date:   09jan2015                                     */
/* Revision date:   13abr2016                                     */
/* *************************************************************** */

#include "util.h"

/**
 * generates ~ 088 micro sec
 */
void util_genDelay088us(void)
{
    char i;
    for(i=0; i<120; i++)
    {
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
        __asm("NOP");
    }
}
```

```
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  }
```

7.36 ../Sources/Util/util.h

```c
/* ***************************************************************** */
/* File name:        util.h                                         */
/* File description: Header file containing the function/methods     */
/*                   prototypes of util.c                            */
/*                   Those delays were tested under the following:   */
/*                   core clock @ 40MHz                              */
/*                   bus clock @ 20MHz                               */
/* Author name:      dloubach                                        */
/* Creation date:    09jan2015                                       */
/* Revision date:    13abr2016                                       */
/* ***************************************************************** */

#ifndef UTIL_H
#define UTIL_H

/**
 * generates ~ 088 micro sec
 */
void util_genDelay088us(void);


/**
 * generates ~  250 micro sec
 */
void util_genDelay250us(void);


/**
/* generates ~  1 mili sec
 */
void util_genDelay1ms(void);


/**
 * generates ~  10 mili sec
 */
void util_genDelay10ms(void);


#endif /* UTIL_H */
```