Projeto Final

EA076 - Laboratório de Sistemas Embarcados

Turma C - Grupo 6

Gustavo Ciotto Pinton RA 117136

Anderson Une Bastos RA 093392

Campinas, 18 de Junho de 2016

Introdução

O objetivo principal do projeto final foi a integração das funcionalidades desenvolvidas nos experimentos anteriores com a adição de um desafio à escolha da equipe.

Conforme os relatórios passados, foram abordados diversos conceitos relacionados à programação de dispositivos embarcados, como a utilização do LCD para fornecer *outputs* visuais para o usuário e do teclado para capturar e processar eventos gerados através do pressionamento dos botões, o uso do conversor analógico digital para coletarmos valores de temperatura produzidos por um sensor de temperatura, a integração do *kit* a uma memória EEPROM através do protocolo de comunicação serial I2C, a utilização do conversor digital analógico com o intuito de implementarmos um gerador de funções e, por fim, o uso do *bluetooth* como um canal de comunicação *master/slave* com o usuário na placa FRDM-KL25Z [1].

De uma maneira geral, através dos experimentos citados anteriormente, pudemos verificar como funcionam as operações de leitura e escrita em uma memória EEPROM, comparar diferenças entre a varredura de um teclado por *polling* e por interrupção, analisar os aspectos temporais de uma comunicação serial assíncrona com um dispositivo *bluetooth* e verificar a realização de conversões de sinais analógicos em valores discretos.

Neste projeto, englobamos as funcionalidades explicitadas acima e adicionamos um dispositivo *touchscreen,* cujo nome é *STM32F7 Discovery* [2], para servir como uma interface gráfica de comunicação amigável com o usuário. A placa *KL25Z* servirá, portanto, como um servidor que calculará e fornecerá os dados que o cliente, isto é, o *kit* STM32F7, requisitar.

Características do Projeto

O projeto pode ser dividido em duas partes principais, sendo elas, o servidor e o cliente. O primeiro trata-se de um componente totalmente passivo que espera os comandos enviados pelo cliente através de uma transmissão serial assíncrona.

A conexão entre os dois módulos foi realizada através de 3 pinos: o terra, que deve ser o mesmo para os dois, e os canais *Rx* e *Tx*. As duas placas operam em 3.3V e utilizam os mesmos limiares para os níveis lógicos alto e baixo. Sendo assim, nenhum circuito transformador de tensões foi necessário.

Nas próximas subseções, buscaremos explicitar os detalhes de implementação dos dois módulos, assim como o protocolo de comunicação, isto é, as mensagens escolhidas para transmissão de dados entre os dois *kits* de desenvolvimento.

Módulo Cliente

Para a implementação do módulo cliente, utilizamos o *kit STM32F7 Discovery* da família *STM32* (circuitos integrados microcontroladores de 32 *bits*), desenvolvido pela *STMicroeletronics*. Além

do dispositivo *LCD Touchscreen*, tal *kit* apresenta diversas outras funcionalidades, detalhadas no *website* [2] da empresa.

Da mesma maneira que o *Processor Expert* abstrai os detalhes de *hardware* do KL25Z, uma biblioteca, chamada de *STM32CubeF7,* também é fornecida e provê diversas funções que manipulam todos os módulos da placa. Os nomes de algumas destas funções conterão o prefixo *HAL*, que significa, em inglês, *hardware abstraction layer,* explicitando seu propósito, já descrito anteriormente. Outras funções conterão o prefixo *BSP*, representando métodos que fazem parte da biblioteca *Utilities*, também disponibilizada junto com o *kit.* As funções providas por esta biblioteca facilitam algumas operações, como, por exemplo, o desenho de formas geométricas na tela.

Além disso, escolhemos utilizar o *FreeRTOS*, que consiste de um sistema operacional simplificado fornecendo suporte a *multitasking* e gerenciamento de memória. Apesar do suporte ao *multitasking*, faremos uso de apenas uma *task*, cuja função é detectar eventuais toques na tela que, por questões de simplicidade, realizada através da técnica de *polling*. A função relacionada a esta *task* é apresentada na seção Implementação da Detecção de Toques, abaixo.

O *kit* STM32F7 Discovery oferece 2 *headers* de pinos, conforme [3]. Utilizamos um dos terras *GND* disponíveis, o pino *PC_6* como saída (*Tx*) e *PC_7*, como entrada (*Rx*).

Implementação da interface gráfica

O arquivo `interface.c` possui a implementação de diversas funções responsáveis pelo desenho de botões e caixas de texto. Contrariamente a aplicações baseadas em *Linux* embarcado, no contexto de microcontroladores, não há bibliotecas de processamento gráfico de botões, *checkboxes* ou *textboxes* já implementadas e, portanto, cabe a nós desenvolvê-las.

Sendo assim, definimos as seguintes *structs* para representar botões na tela e painéis na tela.

Programa 1: *struct* que simboliza um botão.

```
typedef struct {

    uint16_t x_pos, y_pos;
    uint8_t isPushed, hasBeenAcknowledged, isEnabled;
    char title[12];
    button_handler_t buttonPressedHandler;

} button_t;
```

Cada botão possui um par de coordenadas, um título, 3 *flags* que indicam, respectivamente, se ele está pressionado, se o evento já foi tratado e se ele está habilitado, além do ponteiro da função que deve ser chamada para tratar o evento.

Programa 2: *struct* que simboliza um painel.

```
typedef struct {

    board_t board;
```

```
    uint16_t x, y;
    uint8_t isEnabled, isPressed, order;
    board_refresh_screen_t draw_initial_screen;

} pboard_t;
```

Como os botões, cada painel possui um par de coordenadas e algumas *flags* que indicam seu estado. Além destes atributos, cada painel possui um ponteiro para a função que desenhará a tela caso tal painel seja apertado e uma variável do tipo `board_t` que indica o tipo do painel. Tais componentes serão utilizados somente na tela inicial e representarão as funcionalidades fornecidas pelo cliente. No total, quatro foram programadas, sendo elas:

i. Operações com a *EEPROM*: o cliente poderá iniciar e interromper que dados sejam escritos neste componente, assim como requerer estatísticas sobre tais dados e listá-los na tela. A função associada a tal funcionalidade e responsável por atualizar a tela quando ela é escolhida chama-se `draw_EEPROM()`.

ii. Operações com o acelerômetro presente no *kit* KL25Z: o cliente poderá habilitar a leitura do acelerômetro nos três eixos a cada 1 segundo. A função `draw_ACCEL()` é responsável por desenhar botões e caixas de texto associados a tais funcionalidades.

iii. Operação com o conversor analógico digital (ADC): o cliente poderá requisitar valores de tensão, temperatura e pressão, bem como transmitir um relatório com tais dados ao módulo *bluetooth.* Bem como os painéis anteriores, a operação ADC utiliza a função `draw_ADC()` para atualizar a tela.

iv. Operação com o conversor digital analógico (DAC): o cliente poderá gerar ondas nos formatos senoidal, dente-de-serra e quadrada, bem como atribuir um valor constante de 0 a 3.3V na saída deste componente. A mudança de amplitude e período poderá ser feito em tempo de execução. Enfim, o método `draw_DAC()` pinta na tela as opções disponíveis nesta opção.

Além destes dois tipos, muitas funções foram implementadas para a manipulação e renderização dos botões e demais componentes. Entre elas, destacam-se:

i. `void init_interface(void)`: desenha a tela inicial e instancia todos os botões e mais recursos gráficos.

ii. `void back_initial_screen (void* button)`: retorna à tela inicial após o toque no botão *Home*, passado como parâmetro*,* disponível após que um dos painéis tenha sido escolhido.

iii. `void print_initial_screen()`: imita comportamento da função `init_interface`, mas não instancia nenhum componente.

iv. `uint8_t isTouched(ts_event, *button, width, heigth)`: dado o parâmetro `ts_event` do tipo `TS_StateTypeDef`, que detém informações sobre um evento de toque na tela, como coordenadas, por exemplo, verifica se o botão `button` está pressionado ou não, baseando-se na largura e altura, também enviados à função. Caso esteja, retorna 1 e, caso contrário, 0.

v. `uint8_t isTouchedBoard(ts_event, *board)`: assim como no caso anterior, esta função determina se um painel foi pressionado.

vi. `void placeButton(button, color_back, color_text, width, heigth)`: pinta o botão `button` de largura `width` e altura `height`, utilizando as cores `color_back` e `color_text` para, respectivamente, o *background* e o seu texto.

vii. `void placeBoard(*board, color_back, color_text)`: semelhante à função anterior, porém desenha um painel. Neste caso, não é necessário a largura e altura, já que todos os painéis possui os mesmos valores para tais atributos, contrariamente aos botões.

viii. `void refreshButtonState(ts_event, *button, width, heigth)`: atualiza o estado de um botão. Caso ele esteja apertado, chama-se a função *handler* associada a ele e atualiza as *flags* indicando este novo estado. Caso contrário, verifica-se se é necessário atualizar novamente as *flags*.

ix. `void refreshBoardState(ts_event, *board)`: realiza as mesmas operações descritas no item anterior para um painel.

Implementação da comunicação serial assíncrona

A biblioteca *HAL* fornece métodos para a transmissão serial assíncrona. Para o *envio* de dados do cliente para o servidor, escolhemos o método que utiliza *interrupções*, enquanto que, para a *recepção*, escolhemos o método *bloqueante*, isto é, que bloqueia a execução do programa até que a função receba corretamente os dados ou até que o *timeout* seja atingido. Tal escolha foi realizada para checarmos periodicamente se a conexão entre as duas placas ainda está estabelecida.

O envio de dados, via interrupção, é realizado através da função `HAL_UART_Transmit_IT`, que recebe por parâmetro os *bytes* a serem enviados. Tal função ativa a interrupção que é chamada quando a transmissão foi bem-sucedida, cujo nome é `HAL_UART_TxCpltCallback`. Ela realiza uma única operação, que consiste em modificar o estado da *flag* `isReadyToSend`. Tal *flag* indica se um novo *byte* pode ser transmitido.

O recebimento de dados, de maneira bloqueante, é feito por meio do método `HAL_UART_Receive`, que recebe, além do *buffer* onde o *bytes* serão armazenado, o período *timeout*, em milissegundos, que a função deve esperar antes de retornar. Se tal retorno é `HAL_TIMEOUT`, o programa considera que a conexão entre os dois *kits* foi perdida.

Módulo Servidor

O servidor foi implementado na placa KL25Z, utilizada durante os laboratórios da disciplina. As portas e configurações utilizados nos experimentos anteriores foram preservadas e, desta forma, podem ser conferidas nos roteiros e relatórios anteriores.

Baseando-se na tabela *Pin Assignments* de [1], escolhemos as portas *PTB20* e *PTB21* para a comunicação serial assíncrona com o módulo cliente. O componente utilizado no *Processor Expert* foi semelhante àquele empregado na comunicação entre o *bluetooth* e a MCU no experimento 6.

Implementação da comunicação serial assíncrona

A comunicação com o *STM32F7 Discovery* é feita com o auxílio de interrupções. Como no experimento 6, duas interrupções foram ativadas: a que é chamada quando o módulo está pronto para receber um novo *byte* e a outra que é executada quando um novo *byte* pode ser transmitido. Nas funções de tratamento destas interrupções, modificamos as *flags* `hasReceivedCharFromSTM32` e `isReadyToSendToSTM32`, respectivamente, de forma que o programa principal tome conhecimento e possa prosseguir com sua execução.

Sendo assim, para *enviar* um *byte* ao módulo cliente, realiza-se

Programa 3: trecho para enviar um *byte* ao módulo cliente;

```
while (!isReadyToSendToSTM32);
isReadyToSendToSTM32 = 0;
STM32_SendChar(count_a[0]);
```

Da mesma forma, para *receber* um *byte,* executa-se

Programa 4: trecho para receber um *byte* ao módulo cliente;

```
while (!hasReceivedCharFromSTM32);
char rcv_byte;
hasReceivedCharFromSTM32 = 0;
STM32_RecvChar(&rcv_byte);
```

Além destes trechos, implementados algumas funções que enviam ou recebem *floats,* inteiros ou *strings* do cliente, sendo elas:

i. `send_string_STM32(char* name)`: envia uma *string* ao cliente.
ii. `send_float_STM32(float f)`: envia um ponto flutuante à placa *STM32F7 Discovery*.
iii. `get_float_STM32(float *f)`: recebe um *float* através da entrada serial.
iv. `get_int_STM32(int *f)`: recebe um inteiro através da entrada serial.
v. `get_char_STM32()`: recebe um *byte*.

Implementação do programa principal

O programa principal trata-se de um *loop* infinito que espera comandos transmitidos do cliente. Cada comando é composto por apenas um *byte*. Ao receber um comando, o servidor envia uma resposta de confirmação, sendo ora “OK”, ora “SEND”. A primeira opção indica que o servidor recebeu corretamente o respectivo comando, mas que nenhum dado será transmitido entre os dois módulos, contrariamente ao segundo caso, em que um dos dois módulos deverá enviar algum *byte* adicional posteriormente.

No total, 22 comandos foram implementados para a comunicação sendo eles:

i. Conectar-se (‘C’) ou desconectar-se (‘D’) do servidor.

ii. Iniciar (0x10) e interromper (0x11) gravação na EEPROM, assim como transmitir estatísticas de média, máximo e mínimo (0x12) ou valores contidos na memória (0x13).

iii. Iniciar (0x80) ou interromper (0x81) funcionamento do acelerômetro.

iv. Requisitar temperatura (0x50) e tensão (0x51) lidos pelo conversor ADC. No nosso caso, utilizamos um sensor de temperatura. Adicionalmente, o comando 0x53 permite imprimir um relatório completo destes valores em um dispositivo conectado via *bluetooth* ao servidor.

v. Gerar ondas sinoidais (0x60), dente de serra (0x61) ou quadráticas (0x62) através do conversor DAC, assim como interrompê-las (0x64). É permitido aumentar (0x65) ou diminuir (0x66) 25ms do período das ondas e de sua amplitude (0x67 para aumentar em 0.1V e 0x68 para diminuir). Além disso, o comando 0x63 permite atribuir um nível de tensão constate à saída analógico do servidor.

Conclusão

Os objetivos do projeto final foram alcançados com sucesso, desde a criação da interface com o *touchscreen* do dispositivo *STM32F7 Discovery*, a comunicação com *o FRDM-KL25Z* e a implementação da leitura de dados do sensor de temperatura, escrita/leitura na memória *EEPROM,* comunicação com o acelerômetro e, por fim, a geração de ondas através do *DAC*. Como uma sugestão para projetos futuros pode-se incluir alguma funcionalidade que utilize a comunicação via *bluetooth* para que a MCU envie comandos ao *STM32F7 Discovery* vindos de um dispositivo móvel.

Referências

[1] KL25 Sub-Family Reference Manual – Freescale Semiconductors, disponível em *ftp://*ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf

[2] Discovery kit with STM32F746NG MCU, disponível em *http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/32f746gdiscovery.html*.

[3] STM32F7 Discovery Pinout, disponível em *http://www.cnx-software.com/wp-content/uploads/2015/08/STM32F7-DISCOVERY_Pinout.jpg*