Serial Game Controller and Score Monitor

Tiarles da Rocha Moralles Guterres[1], Moisés de Oliveira Goulart[1]

[1] Engenharia de Computação – Universidade Federal de Santa Maria (UFSM)

97.105-900 – Santa Maria – RS – Brazil

Resumo. *Relatório escrito para a disciplina de Projeto de Sistemas Embarcados, ministrada pelo Profº Carlos Henrique Barriquelo no 1º/2017. Tendo como objetivo a introdução do Projeto Final da disciplina.*

1. Introdução ao projeto

A ideia do projeto visa simplificar o trabalho ao mesmo tempo em que se aborda com suficiência o uso de microcontroladores, de seus sistemas embarcados (comunicação USART e módulo OLED1 por exemplo) e do armazenamento de informação em memória (Flash e/ou EEPROM).

2. Material utilizado

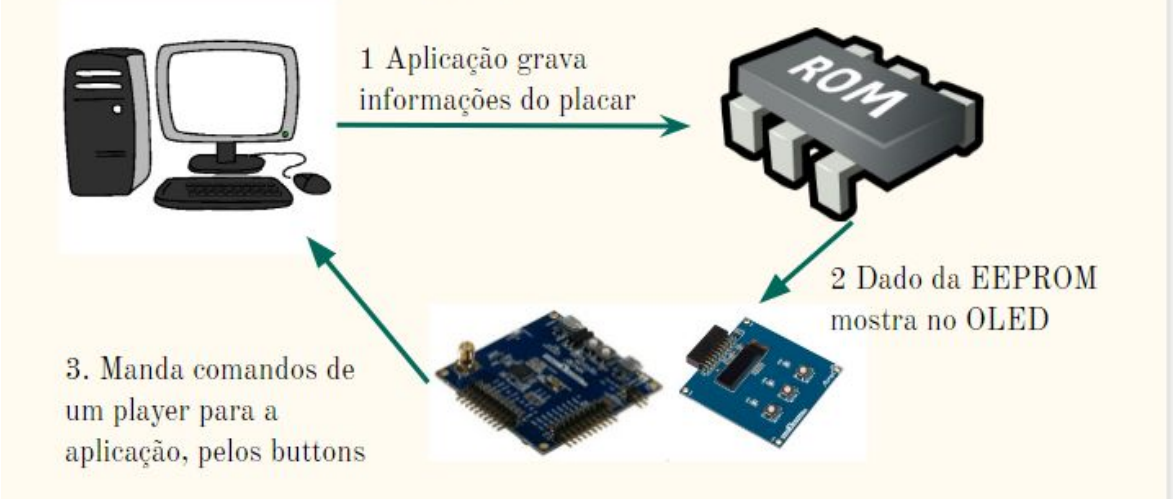
- SAMR21 Xplained Pro (O microcontrolador);
- OLED1 Xplained Pro (Kit com um monitor OLED (128x32) e um teclado simples).

3. A ideia

Utilizamos o teclado do OLED1 Xplained Pro para controlar um jogo via comunicação serial, o jogo é uma aplicação rodando no computador utilizando este kit como um joystick.

A tela OLED mostra a pontuação atual no jogo (Dados estes que serão lidos da EEPROM, gravada em tempo de execução, ao decorrer do jogo atual).

4. O fluxo da comunicação no projeto



4.1. Aplicação grava informações do placar

Obviamente com o auxílio do embarcado a aplicação se comunica via USART e grava um dado padronizado e reconfigurado na EEPROM.

A EEPROM possui todas as preparações necessárias (inclusão do BOD -Brown Out Detector-) para evitar perdas na chamada "memória não-volátil".

Para este processo usa-se uma função básica de leitura de um buffer via usart:

```
//usart_read_buffer_job(&usart_instance, (uint8_t *)rx_buffer, MAX_RX_BUFFER_LENGTH);
usart_read_buffer_wait (&usart_instance, rx_buffer, MAX_RX_BUFFER_LENGTH);
```

Ela faz leitura síncrona da via serial.

E para a escrita na EEPROM, aqui temos de criar um vetor, chamado de *page_data* e essa estrutura será a chave de comunicação com a memória EEPROM:

```
    uint8_t page_data[EEPROM_PAGE_SIZE];
    eeprom_emulator_write_page(0, page_data);
    eeprom_emulator_commit_page_buffer();
```

Na função de escrita utilizamos como parâmetro o próprio *page_data* e o valor '0', a primeira página da EEPROM.

4.2 Dado da EEPROM para o OLED

Para importar os dados da EEPROM para serem mostrados no OLED primeiro utilizamos a função de read page do buffer, utilizando a estrutura anteriormente descrita a *page_data*:

```
eeprom_emulator_read_page(0, page_data);
```

E por fim, com as informações no *page_data* são gravadas no OLED, junto com o já preparado para ser imprimido:

```
uint8_t placar[] = "Placar: ";
uint8_t ox[]     = " x ";// + a info da EEPROM

// Desenho do placar
for(i = 0; i < sizeof(placar) - 1; i++)
    gfx_mono_draw_char(placar[i], i*SYSFONT_WIDTH, 0, &sysfont);
```

4.3. Manda comandos de um player para a aplicação, pelos buttons

Com as funções do OLED1 podemos tratar o button para fazer o que quisermos, neste projeto além de enviar os comandos via USART para a aplicação por:

```
if(oled1_get_button_state(&oled1, OLED1_BUTTON1_ID)){
    usart_write_buffer_wait(&usart_instance, pressButton1, sizeof(pressButton1));
    usart_write_buffer_wait(&usart_instance, '\n', 1);
```

A aplicação também mostra no OLED o que foi enviado para a USART, usando a "*gfx_mono_draw_char(...)*" descrita anteriormente.

5. Controle no computador

O programa em python recebe o estado dos botões através da porta serial utilizando o módulo pySerial:

```
import serial

self.ser = serial.Serial('/dev/ttyACM0', timeout = 0.0001)

if self.ser.readable():
    read = self.ser.read(23)
    if 'U' in read:
        self.player2.move("up")
    if 'D' in read:
        self.player2.move("down")
    if 'N' in read:
        self.player2.move("stop")
```

E a partir da obtenção dos dados seriais a aplicação opera com o outros módulos como o pyGame para criar desde a interface até a física do jogo (velocidade da barra de jogo, velocidade da bola, arte gráfica, etc).

6. Outras partes do código e documentação das funções criadas

O código possui 3 funções criadas de configuração (EEPROM, BOD e USART), além do SYSCTRL_Handler e do próprio main.

```
void configure_usart(void);
void configure_eeprom(void);
static void configure_bod(void);
```

A "*configure_usart()*" configura a **struct** usart_module *usart_instance* criada no escopo do código, seta informações de baudrate e pinos do módulo, além de habilitá-lo;

A "*configure_eeprom()*" configura a própria EEPROM simulada na memória flash disponível, para uma conclusão com sucesso desta função deve ser pré-definido em fuses o tamanho de EEPROM que a aplicação pode usar, caso contário toda vez que for solicitado uma inicialização o *status_code* recebe um define, o STATUS_ERR_NO_MEMORY, que impede a inicialização e uso da EEPROM.

A "*configure_bod()*" vem do exemplo de 'Quick Start for EEPROM' que seria a configuração de um hardware dedicado para fornecimento de energia para possíveis quedas de tensão para a não perda de memória não-volátil.

```
#if (SAMD || SAMR21)
void SYSCTRL_Handler(void)
{
    if (SYSCTRL->INTFLAG.reg & SYSCTRL_INTFLAG_BOD33DET) {
        SYSCTRL->INTFLAG.reg = SYSCTRL_INTFLAG_BOD33DET;
        eeprom_emulator_commit_page_buffer();
    }
}
#endif
```

A SYSCTRL_Handler trabalha em conjunto com a EEPROM e o BOD para comitar páginas e não perder memória.

7. Apêndice A - Sobre Licenciamento e Disponibilização

Como solicitado, o Projeto passará pela análise documentada do software Doxygen e já está no GitHub em código aberto através da licença MIT e pode ser encontrado no seguinte repositório: https://github.com/moisesoliveira/Embarcados.