**Usando um Joystick Playstation**

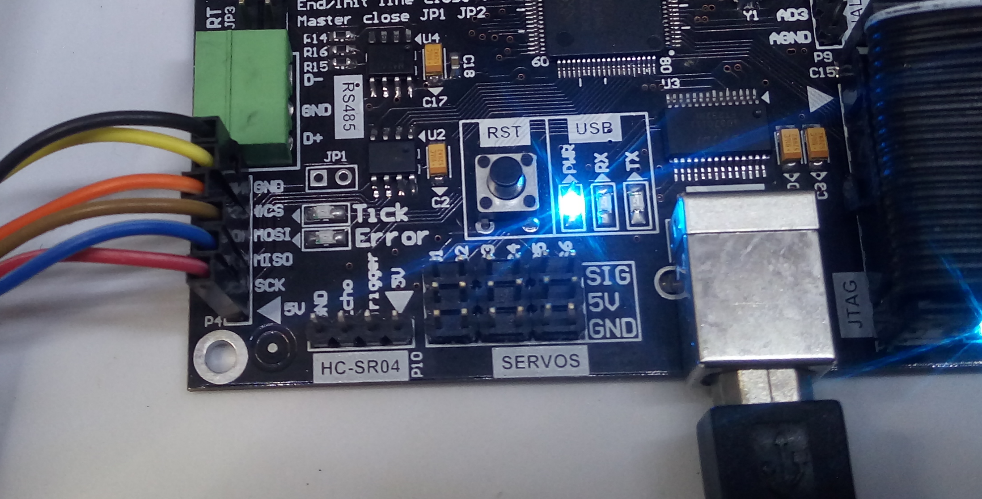
Neste exemplo faremos uma leitura de um joystick de um Playstation e mostraremos os resultados em um display texto 16x2:



**Conexão com o Kit ARM**

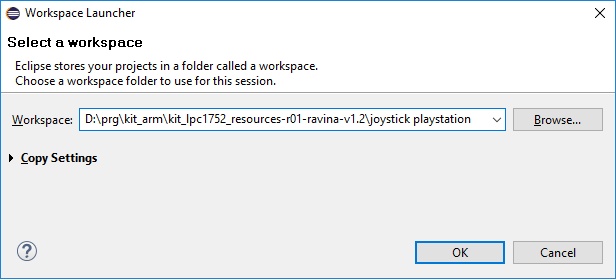
Este tipo de joystick requer uma comunicação SPI com uma configuração em peculiar em relação na maioria dos dispositivos que usam esse tipo de barramento. Usaremos a porta P4 que utiliza a porta SPI1 do ARM para acesso ao joystick e na tabela abaixo descreve a conexão:

|  |  |  |
| --- | --- | --- |
| **Joystick** | **Conector** | **Pino ARM** |
| Chip select (CS#) | P4 pino 5 | 45 (P0\_18) |
| SPI CLK | P4 pino 2 | 63 (P0\_7) |
| SPI MISO (SDO) | P4 pino 3 | 62 (P0\_8) |
| SPI MOSI (SDI) | P4 pino 4 | 61 (P0\_9) |
| VCC (3V a 5V) | P4 pino 1 |  |
| GND | P4 pino 6 |  |



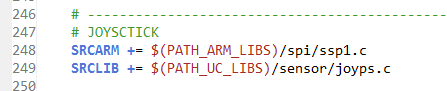
**Programa Exemplo**

Abra o exemplo da pasta “joystick playstation” com o Eclipse:



No makefile, na secção de inserção de fontes, foi incluindo as seguintes fontes:

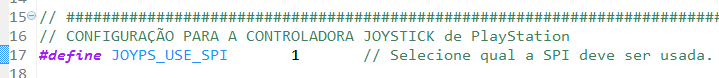
* ssp1.c: Código de acesso a porta SPI1 do ARM;
* joyps.c: Driver de acesso ao joystick;



Em app.h foi adicionado os includes das respectivas fontes de códigos inseridos anteriormente para serem usados na aplicação:



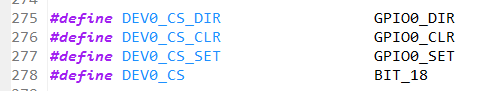
Em \_config\_cpu\_spi.h é feito a configuração de qual porta SPI de acesso pelo joystick:



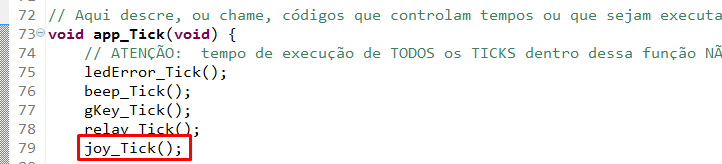
Em main.c dentro do int main(void), são inicializados a porta SPI1 e o driver do joystick. A inicialização da porta SPI é feita com configuração adequadas para este tipo de dispositivo, e a velocidade padrão de comunicação é de 400KHz, que é o suficiente para uma interface homem máquina:



Caso queira mudar o pino de chip select do joystick isto é feito nas definições do arquivo arch\_hardware\_resources.h:



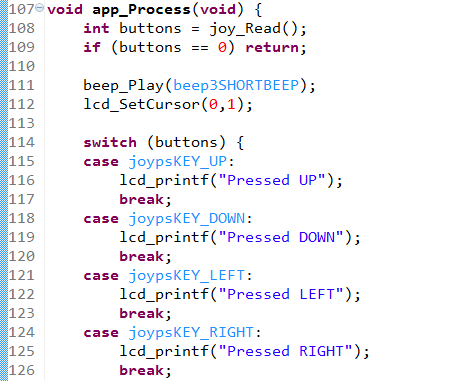
A leitura do joystick deve ser feita a cada 16ms, este tempo já é o suficiente para uma boa resposta de leitura das ações do usuário. Como se trata de leitura de tempos regulares é preciso chamar a função joy\_Tick dentro da função app\_Tick. A função app\_Tick chama todas as funções ticks do sistema:



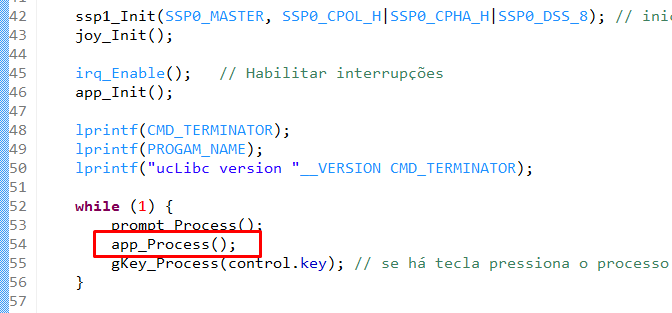
A função app\_Tick é chamada a cada 1ms, isto foi especificado no início do programa na função main:



A leitura do joystick pode ser feita a qualquer momento, então é feita dentro do loop principal do programa na chamada da função app\_Process. Dentro dessa função fazemos a leitura do joystick através da função joy\_Read, e quando há um botão pressionado emitimos um sinal sonoro via beep\_Play, e imprimimos no LCD a mensagem de acordo com o valor do botão pressionado:



A função app\_Process é chamada dentro do loop da função main:



**Compile e Execute**

Com tudo instalado e configurado, compile a aplicação, grave e execute no ARM.

Exemplo teclando o botão para cima:



Exemplo teclando o botão para baixo:



**Joystick do Playstation**

Entre muitos comandos para acessar o joystick, somente foi estudado o comando para ler os estados dos botões, que são necessários para nossos projetos, desconsiderando quaisquer outros comandos, como controle de vibrações e configurações do joystick.

**Conector do Playstation**

As tensões de trabalho nos pinos dos conectores do joystick do Playstation são de 0 e VCC, representando respectivamente os níveis lógicos 0 e 1. O VCC é a tensão de trabalho do joystick, segundo os documentos da Playstation, os joysticks podem trabalhar entre tensões de 3 a 5 Volts. Contudo, pela análise de sinais feitos e projetos semelhantes já analisados, as tensões de trabalho sempre são de 3.3 volts. Por tanto, adotaremos também está tensão de trabalho.



**Conector do cabo e imagens de dois modelos de joysticks**

|  |  |  |  |
| --- | --- | --- | --- |
| **Pino** | **BUS SPI** | **Descrição** | **Orientação** |
| 1 | MISO | DATA. Linha de dados do joystick para o HOST | Saída |
| 2 | MOSI | COMAND. Linha de comandos do HOST para o joystick | Entrada |
| 3 | VCC\_MOTOR | Pino de alimentação dos motores de vibrações. (7 a 9 VDC).  Não usamos para nossos projetos | Entrada |
| 4 | GND | GND | Entrada |
| 5 | VCC | Alimentação do joystick entre 3.3VDC a 5 VDC. Consumo máximo de 750mA. | Entrada |
| 6 | #CS | ATTENTION. Pino de chamada de atenção feito pelo HOST ao joystick para início de comunicação. | Entrada |
| 7 | CLK | CLK. Clock do barramento para transferência de dados. | Entrada |
| 8 |  |  |  |
| 9 | ACK | Pino que o joystick utiliza para sinalizar ao HOST o reconhecimento da recepção de dados | Saída |

Os pinos MISO e ACK devem está em PULLUP através de um resistor de 4K7 ohms, pois os PULLUP dos pinos dos uC não são o suficiente para manter a integridade dos sinais.

**Barramento de Dados**

O barramento de dados que o joystick utiliza é o padrão SPI, com exceção do pino ACK. Todas as transferências são iniciadas pelo HOST (console de jogo ou uC), ou seja, o HOST sempre é o mestre do barramento e todos os joysticks são os escravos. As relações entre os pinos do joystick com o padrão SPI são:

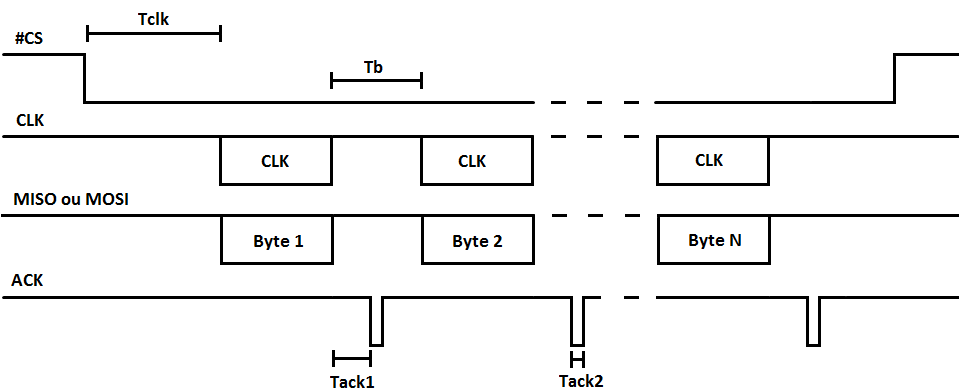
|  |  |
| --- | --- |
| **SPI** | **Joystick** |
| SCK (Serial Clock) | CLK |
| #CS (Chip select) | ATT (Attention) |
| MISO (Master IN Slave Out) | DATA |
| MOSI (Master Out Slave In) | CMD (Command) |

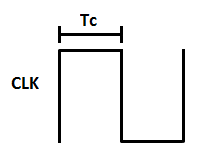
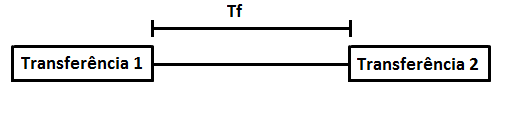
No barramento SPI para acessar o joystick os bits menos significativos são transmitidos primeiro (LSB). O clock deve permanecer em nível alto quando o barramento estiver ocioso (CPOL=1), e os bits são capturados na segunda borda do clock (CPHA=1), isto é, neste caso quando a borda do clock cai os bits são enviados e quando a borda sobe os bits são capturados. Para mais informação consulte documento do seu uC na secção SPI.

O barramento SPI pode operar em velocidades muito altas, e pelo breve estudo que fiz sobre o joystick, a sua velocidade típica é de 250Khz, em alguns casos de até 500Khz. A velocidade do joystick é baixa em relação a muitos dispositivos, contudo, é mais que suficiente para uma interface humana.

Em nossos projetos ignoramos o sinal ACK, e para saber se existe ou não um joystick conectado ao sistema, usaremos polling fazendo leituras do IDs dos joysticks.

Os sinais descritos aqui é referente a ciclos de clock de 4uS. Para iniciar a transferência de dados o sinal #CS, do joystick desejado, vai ao nível baixo fazendo a seleção do dispositivo para comunicação. Após 1uS, tipicamente 39.4uS, da seleção do joystick, começa a transferência dos bytes. O joystick ativa o sinal de reconhecimento ACK em torno de 10uS depois que receber cada byte, e permanece ativo pelo menos metade do ciclo de clock. Se o HOST não receber o ACK após alguns uS, o HOST considera a ausência do joystick. Entretanto, podemos ignorar o sinal ACK, não o esperando e fazemos a transmissão do próximo byte. Após a inicialização da comunicação (#CS=0), não pode haver um intervalo muito grande entre as transferências dos bytes, não mais que 19.4uS, senão o joystick cancela a comunicação e é preciso recomeçar a comunicação novamente. Para uma interface humana o período de leitura do joystick pode ser feito entre intervalos de 16.15mS.



* Tclk. Tempo entre o sinal #CS habitado e a transferência do primeiro byte. Mínimo 1uS, tipicamente é 39.4uS;
* Tb. Tempo de transmissões entre os bytes máximo de 19.4uS;
* Tack1. Tempo do ACK do joystick após de receber o byte do HOST. Tipicamente 10.6uS;
* Tack2. Duração do pulso do ACK pelo menos 2uS;
* Máximo período de clock de 4uS, logo Tc = 2uS;
  + 
* Tf. Tempo entre consulta do HOST ao joystick. Tipicamente de 16.15mS.
  + 

**Protocolo de Comunicação**

Em uma transferência de dados entre o HOST ao joystick pode ser em uma rajada de 5 ou mais bytes, dependendo do tipo de dispositivo conectado ao barramento, e o tipo de comando enviado. Existem muitos comandos envolvidos no protocolo como configuração do joystick, leitura de status, controle de vibrações, entre outros. Contudo, os únicos comandos necessários em nossos projetos é o comando de leitura do ID e de leitura dos estados dos botões.

Por conversão, os bits são representados neste artigo com os bits mais significativo primeiro (MSB) que é ao contrário do que é lido no barramento.

O protocolo é dividido em cabeçalho e corpo de dados, O cabeçalho onde é enviado o comando ao joystick e o retorno de seu ID. O corpo de dados é o local que encontra os estados dos botões e dos manches, ou os demais dados que são enviados e recebido do joystick.

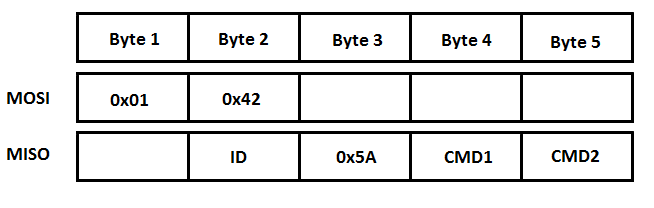
O ID lido do joystick sinaliza em que modo de trabalho o joystick se encontra no momento, se está no modo digital ou no modo analógico. Também o ID sinaliza qual é o tamanho do pacote de dados que será enviado pelo joystick. O ID é dividido em nibble alto e nibble baixo:

* Nibble alto: Sinaliza o modo de trabalho do joystick;
  + 0x4 = modo digital
  + 0x7 = modo analógico
* Nibble baixo: Sinaliza o tamanho em 16 bits de pacote de dados.
  + 0x1 = Dados de 16 bits. Exemplo: Joystick digital somente manda dois bytes de comandos dos botões;
  + 0x3 = Dados de 48 bits. Exemplo: Joystick modo analógico manda 6 bytes de comandos dos botões e manches.

Logo um ID de um joystick analógico seria 0x73 e o ID de um joystick digital o ID seria 0x41.

**Comando para leitura do joystick no modo digital**

Para o joystick digital a transferência de dados se dá a uma rajada de 5 bytes por acesso. Se o joystick for analógico, com manches, ele também transmite no modo digital, desde que o botão analógico esteja desligado. Abaixo mostra essa sequência do protocolo.

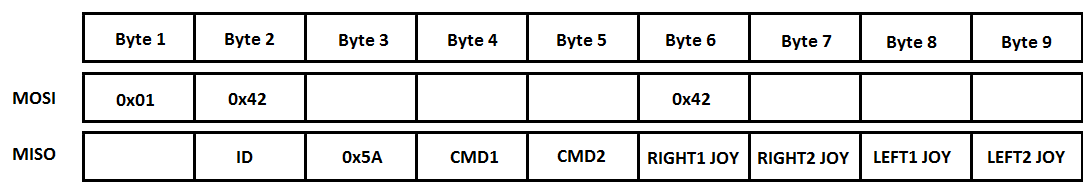
****

|  |  |  |
| --- | --- | --- |
| **Nº Byte** | **MOSI (HOST transmite)** | **MISO (Joystick transmite)** |
| 1 | HOST transmite o comando de iniciar transferência (0x01) | Joystick transmite byte irrelevante |
| 2 | HOST transmite o comando de quesito de dados do joystick (0x42) | Joystick transmite o seu identificador (ID). 0x41 para digital e 0x73 para analógico. |
| 3 | HOST transmite byte irrelevante | Joystick transmite o valor 0x5A indicando que está iniciando a transmissão de dados |
| 4 | HOST transmite byte irrelevante | Joystick transmite comando 1 |
| 5 | HOST transmite byte irrelevante | Joystick transmite comando 2 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bits** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Comando1** | RIGHT | DOWN | LEFT | UP | START |  |  | SELECT |
| **Comando2** | Quadrado | X | Circulo | Triangulo | R1 | L1 | R2 | L2 |

**Comando para leitura do joystick no modo analógico**

Para o joystick analógico a transferência de dados se dá a uma rajada de 9 bytes por acesso. Abaixo é exibido essa sequência do protocolo.

****

|  |  |  |
| --- | --- | --- |
| **Nº Byte** | **MOSI (HOST transmite)** | **MISO (Joystick transmite)** |
| 1 | HOST transmite o comando de iniciar transferência (0x01) | Joystick transmite byte irrelevante |
| 2 | HOST transmite o comando de quesito de dados do joystick (0x42) | Joystick transmite o seu identificador (ID). 0x41 para digital e 0x73 para analógico. |
| 3 | HOST transmite byte irrelevante | Joystick transmite o valor 0x5A indicando que está iniciando a transmissão de dados |
| 4 | HOST transmite byte irrelevante | Joystick transmite comando 1 |
| 5 | HOST transmite byte irrelevante | Joystick transmite comando 2 |
| 6 | HOST transmite byte irrelevante | Joystick transmite comando HIGH1 JOY |
| 7 | HOST transmite byte irrelevante | Joystick transmite comando HIGH2 JOY |
| 8 | HOST transmite byte irrelevante | Joystick transmite comando LEFT1 JOY |
| 9 | HOST transmite byte irrelevante | Joystick transmite comando LEFT2 JOY |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bits** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Comando1** | RIGHT | DOWN | LEFT | UP | START |  |  | SELECT |
| **Comando2** | Quadrado | X | Circulo | Triangulo | R1 | L1 | R2 | L2 |
| **RIGHT1 JOY** | Manche da direita no sentido direita e esquerda. Valor 0x00 estrema esquerda e valor 0xFF estrema direita | | | | | | | |
| **RIGHT2 JOY** | Manche da direita no sentido cima e baixo. Valor 0x00 estrema acima e valor 0xFF estremo baixo | | | | | | | |
| **LEFT1 JOY** | Manche da esquerda sentido direita esquerda. Valor 0x00 estrema esquerda e valor 0xFF estrema direita | | | | | | | |
| **LEFT2 JOY** | Manche da esquerda no sentido cima e baixo. Valor 0x00 estrema acima e valor 0xFF estremo baixo | | | | | | | |

**Comandos do joystick analógico com LED vermelho**

**A interpretação dos estados dos botões**

* Botão pressionado tem nível lógico 0;
* Botão não pressionado tem nível lógico 1.

**Procedimento para Leitura do Joystick**

Este procedimento se repete a cada 16.75mS. Não necessariamente precisa ser nesse exato tempo, pode adotar outros intervalos de acordo com a aplicação de uso.

**Procedimento para ler o joystick**

1. Habilitando o joystick para comunicação (#CS = 0);
2. Esperar por 39.4uS;
3. HOST manda o byte 0x1 para o joystick sinalizado início de transação;
4. HOST manda comando de leitura dos estados dos botões (byte 0x42) e o joystick responde com seu ID;
5. HOST manda qualquer valor para que o joystick transmita de volta o valor de início de transmissão de dados 0x5A;
6. HOST manda qualquer valor de byte para que e o joystick transmita o comando 1;
7. HOST manda qualquer valor de byte para que e o joystick transmita o comando 2;
8. HOST manda qualquer valor de byte para que e o joystick transmita o RIGHT1 JOY;
9. HOST manda qualquer valor de byte para que e o joystick transmita o RIGHT2 JOY;
10. HOST manda qualquer valor de byte para que e o joystick transmita o LEFT1 JOY;
11. HOST manda qualquer valor de byte para que e o joystick transmita o LEFT2 JOY;
12. Desabilitando o joystick para comunicação (#CS = 1).

Se no passo 4 o joystick respondeu que no modo digital leia somente os comandos 1 e 2 e pule para o passo 12. Pois os passos 8 a 11 são para leituras dos comandos analógicos.

**Referencias**

http://www.gamesx.com/controldata/psxcont/psxcont.htm

http://pinouts.ru/Game/playstation\_9\_pinout.shtml

http://www.webx.dk/robot-crawler/ps-joy.htm

http://www.curiousinventor.com/guides/ps2