Experimento 3 - Teclado

EA076 - Laboratório de Sistemas Embarcados

Turma C - Grupo 6

Gustavo Ciotto Pinton RA 117136

Anderson Une Bastos RA 093392

Campinas, 11 de abril de 2016

Introdução

O objetivo deste experimento foi a manipulação de um teclado com 12 teclas. A primeira etapa foi dedicada aos estudos do funcionamento do respectivo componente a partir de [1] e da maneira de ligá-lo ao microcontrolador a partir de pinos de entrada e saída. Conforme explicado nas próximas seções, utilizamos 4 pinos de saída, um para cada linha, e 3 de entrada, um para cada coluna. Dois métodos de verificação das teclas foram implementadas: detecção via *polling* e via interrupções. No primeiro método, laço verifica continuamente o estado das teclas enquanto que, na segunda, o processamento é interrompido quando uma tecla é pressionada.

Em seguida, avaliamos algumas técnicas de *debouncing,* tanto de *software* quanto de *hardware*, a fim de melhorar a performance da solução implementada. Escolhemos uma solução de *software,* baseada no tempo de pressionamento.

Funcionamento do Teclado e Varredura por *polling* (Partes 1, 2, 3 e 4 do Roteiro)

O teclado utilizado neste experimento contém 12 chaves, organizadas em 4 linhas e 3 colunas, conforme figura 1, abaixo. Cada uma das linhas é ligada a um pino de *saída* do microprocessador, enquanto que cada coluna é conectada a uma *entrada*. Quando uma chave posicionada em `(i, j)`, em que `i` representa uma linha e `j`, uma coluna, é pressionada, a tensão aplicada na linha `i` é transmitida à coluna `j`, uma vez que o contato entre os dois barramentos é estabelecido.

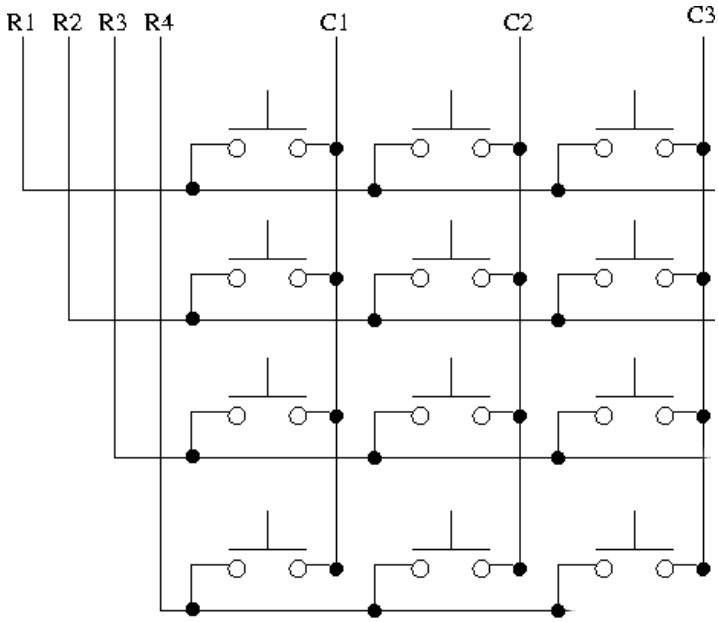

Figura 1: Circuito presente em um teclado. Adaptado de [1].

Antes de implementar o *firmware* que detectará as teclas, é necessário estabelecer os níveis de tensão que serão aplicados às colunas e linhas e que permitirão determinar, assim, o estado de um *botão*. Foi definido que, caso nenhuma tecla pertencente a uma coluna esteja apertada, então tal coluna deve possuir uma tensão de 3.3V, correspondente ao nível lógico alto. Tal comportamento é similar a aquele apresentado no **Relatório 1**, quando um pino de entrada era utilizado para determinar se um botão foi pressionado ou não. Sendo assim, cada coluna será conectada a um resistor *pull up*,

conforme figura 2 abaixo. Quando nenhuma das chaves está pressionada, a corrente que passa pelo resistor é praticamente nula, assim como a queda de tensão. Foram utilizados resistores de 4kΩ a fim de proteger o sistema contra correntes muito altas, já que, em termos simples, as teclas são chaves que ligam V_{cc} a portas do microcontrolador, sendo que correntes elevadas podem danificá-lo. O valor de 4kΩ foi pensado para evitar danos mesmo na ocorrência do pior caso, ou seja, em que 4 teclas de uma mesma coluna são pressionadas ao mesmo tempo. Quando este cenário acontece, quatro diferentes correntes se somam em um mesmo barramento. Com a utilização de resistores de 4kΩ, a corrente atinge, no máximo (pior caso), 3mA, protegendo, portanto, o circuito.

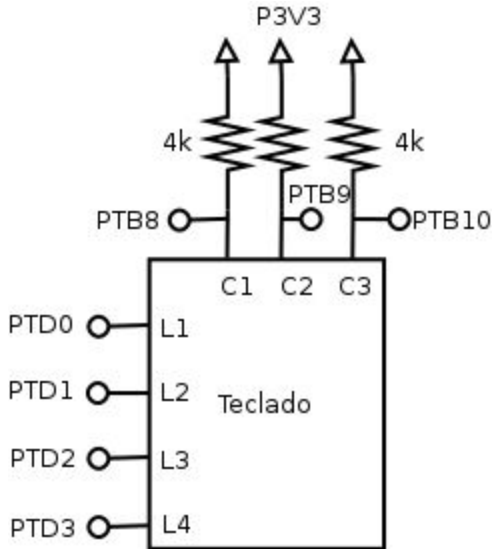

Figura 2: Ligações dos pinos do microcontrolador ao teclado.

Para determinar se uma tecla `(i, j)` foi acionada, aplica-se uma tensão nula, correspondente ao nível lógico baixo, somente ao pino de saída ligado à linha `i` e tensões altas em todos os demais pinos. Se a respectiva tecla foi de fato pressionada, uma tensão baixa será submetida ao pino de entrada ligado à `j` e o evento poderá ser detectado pelo *firmware* através da verificação do estado de tal pino. Repete-se tal processo para toda combinação de `i` e `j`. Considere, por exemplo, a seguinte situação: a tecla `(x+1, y)` está pressionada, sendo que estamos testando o estado da tecla `(x, y)`, que está solta. O fato do primeiro botão estar pressionado não influencia na leitura do segundo, uma vez que a linha `x+1` possui um nível de tensão alto. A coluna `y` continua, corretamente, em nível lógico alto.

A função `getkey(int **keys_matrix)` abaixo foi escrita a fim de determinar quais botões foram apertados. Ela recebe como parâmetro uma matriz 4x3 de inteiros, que armazenará os estados dos botões: caso o botão `(i, j)` esteja pressionado, o elemento `(i, j)` dessa matriz conterá 1. Caso contrário, possuirá 0.

Programa 1: Função `get_keys` que verifica estados dos botões.

```
void get_keys(int **keys_matrix) {

    check_key_L1(keys_matrix[0]);

    check_key_L2(keys_matrix[1]);

    check_key_L3(keys_matrix[2]);

    check_key_L4(keys_matrix[3]);
}
```

A função acima faz chamadas a quatro outras funções, responsáveis por testar cada linha individualmente. Cada uma das `check_key_Ln()` recebe a respectiva linha da matriz que ela deve modificar, sendo que a linha 0 representa o barramento `L1`, a 1 representa `L2` e assim por diante. O programa 2 abaixo contem `check_key_L1()`. A primeira etapa realizada por ela é limpar o vetor recebido como parâmetro, atribuindo 0 a todas as posições. Em seguida, modificamos as tensões dos pinos de saída, ligados às linhas, de maneira que somente `L1` possua a tensão correspondente ao nível lógico baixo. Enfim, verificamos as 3 colunas individualmente e modificamos as posições do vetor referentes as colunas que apresentarem o nível lógico 0.

Programa 2: Função `check_key_L1()` que verifica estados dos botões ligados à linha L1.

```
void check_key_L1(int* result) {

    memset(result, 0, 3*sizeof(int)); /* Limpa vetor */

    /* Atribui tensão baixa a somente uma das linhas */
    L1_ClrVal();
    L2_SetVal();
    L3_SetVal();
    L4_SetVal();

    /* Verifica colunas */
    if (C1_GetVal() == 0)
      result[0] = 1;

    if (C2_GetVal() == 0)
      result[1] = 1;

    if (C3_GetVal() == 0)
      result[2] = 1;

    L1_SetVal();
}
```

Na função `main()`, a leitura das teclas é efetuada através de uma técnica chamada *polling*. Esta técnica consiste na varredura dos botões de maneira contínua através de um *loop* sem condições de parada, por exemplo. O seguinte programa contém o trecho referente à implementação desta técnica. A cada iteração do `while`, a função `get_key()` é chamada e a varredura dos botões ocorre. Em seguida, percorremos essa matriz à procura das posições que apresentam 1 e que, portanto, representam teclas apertadas. Se um 1 é encontrado, então obtemos o caracter que deve ser impresso no LCD a partir de uma outra matriz, chamada `buttons_map`, usando a mesma coordenada da posição onde o 1 foi encontrado.

Programa 3: Trecho da função `main()` que implementa a técnica de *polling*

```
const char buttons_map [4][3]
    = {{'1', '2', '3'},{'4', '5', '6'},{'7', '8', '9'},{'#', '0', '*'}};

int main() {
    [...]
    while (!Button_GetVal()){ // Condicao de parada estabelecida para
                              // fim do polling
      int i,j;
```

```
      get_keys(buttons); // Chamada da funcao que realiza varredura
                         // das teclas.

      for (i = 0; i < 4; i++)
           for (j = 0; j < 3; j++) {

                // Caso a posicao (i,j) seja 1,isto eh, botao (i,j)
                // apertado.
                if (buttons[i][j]) {

                    // Verifica se eh necessario limpar display (linha
                    // completa)
                     if (!( count % 16 ) && count != 0 ) {
                          send_cmd(0x01, 26);
                          count  =0;
                     }

                    // Envia caracter ao LCD
                     send_data(buttons_map[i][j]);
                     count++;
                }
           }
     }
     [...]
}
```

Pelo fato de consumir processamento durante as verificações, esta técnica não representa a melhor solução. Na próxima seção, será apresentada a técnica de varredura por interrupção, que permite realizar a varredura das teclas somente quando uma delas for pressionada.

Varredura por ocorrência de Interrupção e Tratamento de *bouncing* **(Partes 5, 6, 7 e 8 do Roteiro)**

Varredura por ocorrência de Interrupção

Como explicado na seção anterior, o programa realiza um *loop* infinito em que, a cada iteração, ocorre a leitura das colunas e linhas do teclado em busca do botão que está pressionado. Esta solução, embora factível, não é ótima, pois o processador acaba gastando tempo para realizar este *loop* infinito que poderia ser melhor utilizado em outras tarefas. Nesta linha, uma forma de liberar o processador para outras tarefas é implementando um código que é ativado a partir de uma interrupção. Ou seja, caso uma tecla seja apertada, uma *flag* de interrupção é ativada e entra-se em uma rotina de leitura das colunas e linhas para identificar qual tecla foi apertada.

Em nosso projeto, utilizamos uma porta `AND` de 3 entradas (TTL SN74LS15) e resistores de proteção de 4kΩ por coluna. É necessário observar, porém, que as saídas de tal componente devem apresentar a característica *open-collector*, pelo fato de que a sua tensão de referência (5V) difere daquela usada no microcontrolador (3.3V). Neste tipo de componente, a saída de cada porta não é ligada a nenhuma tensão de referência, conforme figura 3 (retirada de [4]) cabendo a nós determinarmos e implementarmos estas ligações de acordo com as características do nosso projeto, isto é, tensão de

referência igual a 3.3V e corrente máxima da ordem de 5mA. Dessa forma, quando o transistor mais a direita estiver “aberto”, a saída Y possuirá 3.3V (e não 5V!) e, quando estiver “fechado”, Y possuirá a mesma tensão do terra.

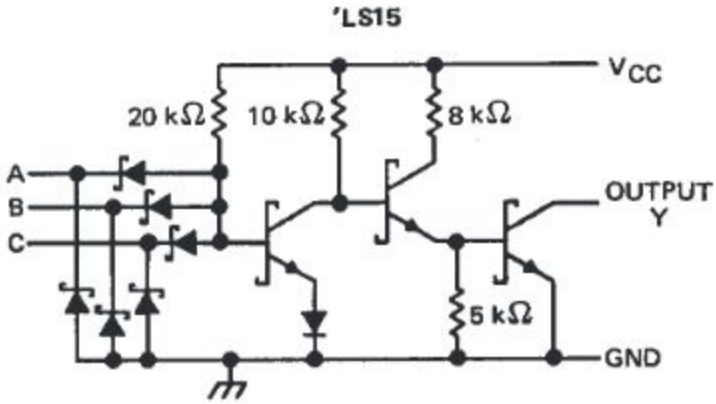

Figura 3: Porta AND com saída *open-collector*. Retirado de [4].

Enfim, o circuito, incluindo as colunas, pode ser conferido no esquema da figura 4.

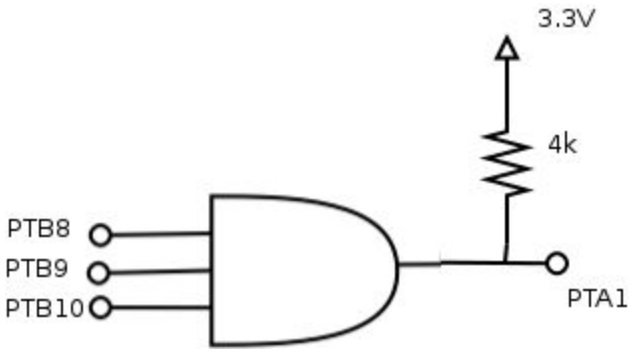

Figura 4: Conexões das colulas à porta AND com saída *open-collector*.

A saída de cada uma das colunas serve como um pino de entrada de um AND e também é ligada a resistores de 4k ohms, conforme explicado na seção anterior.

Voltando a análise do circuito, a saída de cada uma das colunas serve como um pino de entrada de uma porta AND. Assim que uma tecla é apertada, a tensão aplicada na linha `i` passa para a coluna `j`, fazendo com que o estado da coluna se altere e, consequentemente, o estado da saída da porta `AND` também. Em nossa lógica, caso nenhuma tecla seja apertada, o estado de cada coluna é de HIGH (Alto). Quando uma tecla é apertada, a coluna muda seu estado para LOW (Baixo), e portanto a saída da porta AND também se torna LOW. Esta mudança (*falling edge*) na saída da porta `AND` é captada pela porta `PTA1` da placa KL25Z (como definido pelo roteiro) e então uma interrupção é ativada.

A rotina da interrupção faz uma chamada à função `get_keys()` que é a rotina de leitura das linhas e colunas para identificação da tecla que foi apertada, que é similar ao da seção anterior.

Tratamento do *Bouncing*

Nas seções anteriores explicamos o funcionamento lógico do teclado. No plano digital, uma parte muito importante deve ser analisada, que é o tempo de execução do programa.

A interação do ser humano com o teclado, por mais ágil que seja, é da ordem de dezenas de milisegundos, enquanto que a velocidade de funcionamento das máquinas pode ser da ordem de nanosegundos. Esta diferença no tempo faz com que eventos inesperados ocorram no decorrer da

execução do programa. Um exemplo de evento inesperado é o *bouncing,* que consiste de trepidações de tensão que ocorrem nos instantes anteriores em que a chave é apertada. Estas trepidações de tensão são chamados de *spikes* e afetam significativamente na execução do programa. Como dito anteriormente, o pressionamento do botão por um ser humano pode durar dezenas de milisegundos, no entanto, este tempo é extenso para um circuito digital, e portanto, ele acaba interpretando estas trepidações dentro de seu código, fornecendo resultados inesperados.

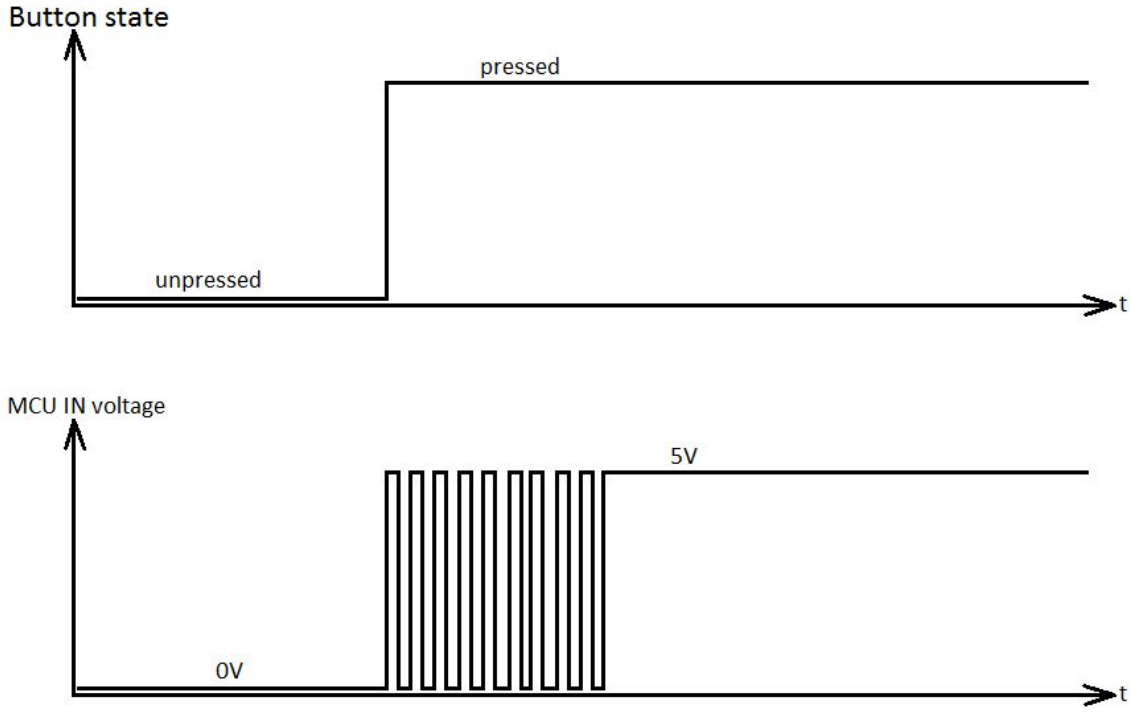

Figura 5: Demonstração do *bouncing*

Este fenômeno (negativo) também ocorreu dentro de nossa implementação do teclado. Quando pressionávamos um dígito, por mais rápido que pressionávamos, o dígito era enviado repetidas vezes no LCD. Por isso, descobrimos a necessidade da implementação de um circuito *debouncer* para eliminar este problema.

O circuito *debouncer* pode ser feito através de *hardware* ou de *software*. Em nosso projeto implementamos o tratamento do bouncer via *software*, como explicado a seguir.

Tratamento do *Bouncing* via *software*

O tratamento de *bouncing* via *software* faz uso do *timer* que é utilizado também para o LCD. A cada interrupção gerada pelo *timer*, uma variável, chamada `ticks_counter`, do tipo `unsigned int` é incrementada. A função de interrupção possui uma variável, chamada `last_tick`, persistente (`static`) e `unsigned int`, que armazena o valor de `ticks_counter` na última vez que a função de interrupção foi executada. Quando uma nova interrupção é captada, a função verifica se a diferença entre `ticks_counter` e `last_tick` é maior que certo valor e, caso seja, o estado dos botões é verificado. Caso contrário, a interrupção é ignorada. Dessa forma, interrupções geradas pelo *bouncing* dentro de um pequeno intervalo de tempo são simplesmente ignoradas. O valor escolhido para a diferença mínima garante que interrupções geradas por *bounces* dentro de 100ms desde o pressionamento do botão sejam descartadas. O programa a seguir contém o tratamento descrito neste parágrafo. Pelo fato do tipo das variáveis ser `unsigned int`, não precisamos nos preocupar com o *overflow* (número máximo suportado pelo tipo), já que a subtração também gera valores `unsigned int`. Por exemplo, se subtrairmos 0xFFFFFFFF de 0x00000002 o resultado é 0x00000003, conforme esperado.

Programa 4: função tratadora de interrupções, que elimina interrupções geradas por *bounces*

```
void EInt1_OnInterrupt(void)
{
    static unsigned int last_tick = 0;
    // 1666 equivale a 1ms
    if (tick_counter - last_tick > 1666) {
      get_keys(buttons);
      isReady = 1; //comunica a funcao main que pode imprimir caracteres
    }

    last_tick = tick_counter;
}
```

Referências Bibliográficas

[1] Keypad Scan, disponível em http://esd.cs.ucr.edu/labs/decode_key/decode_key.html

[2] Configuração do Keypad, disponível em ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/complementos/Configuracao_do_TECLADO2.pdf

[3] A Guide to Debouncing, or, How to Debounce a Contact in Two Easy Pages, disponível em http://www.ganssle.com/debouncing.htm

[4] Triple 3-input positive-and gates with open-collector outputs, disponível em http://www.ti.com/lit/ds/sdls133/sdls133.pdf