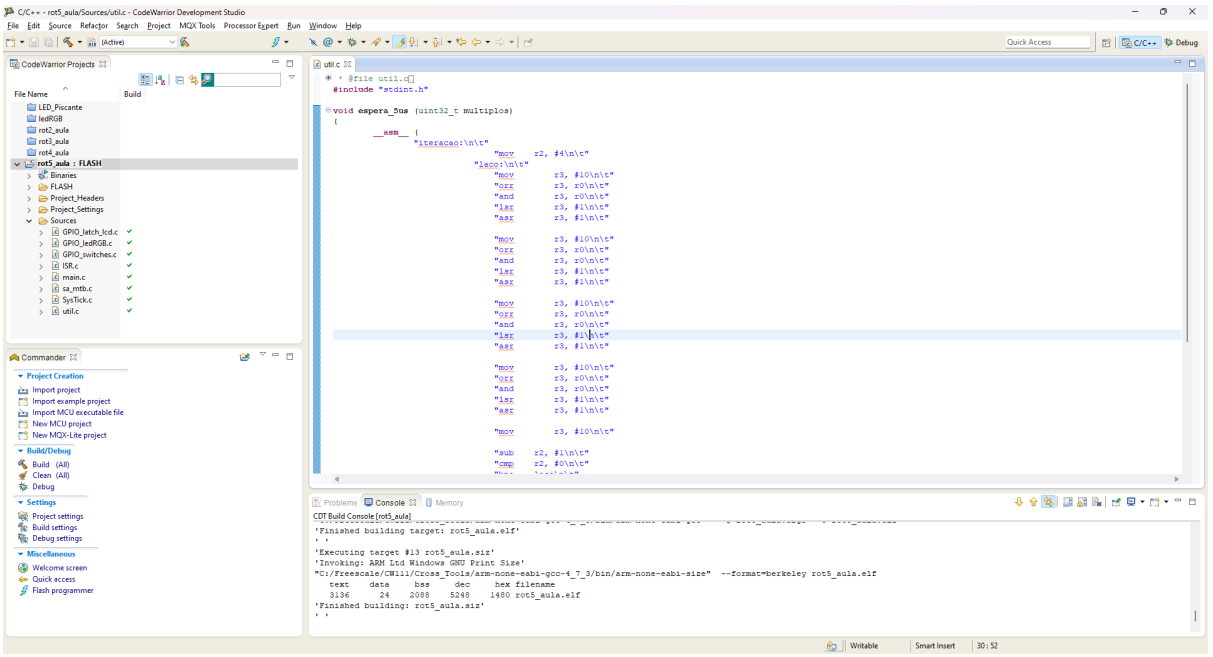
FEEC - UNICAMP;
EA871;
EXPERIMENTO DA TAREFA 5;
VINÍCIUS ESPERANÇA MANTOVANI, RA: 247395;

Ítem 1)

1.1) Substituindo a função e gerando o executável:



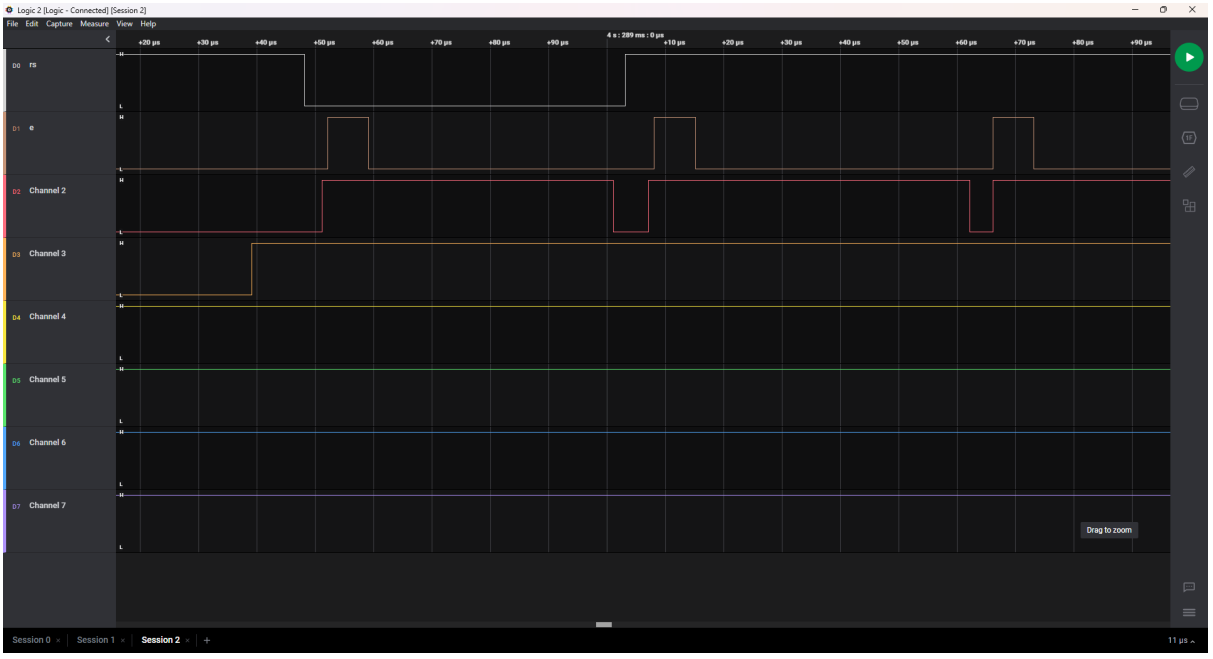
Ao executar o programa no modo run, o led azul é aceso e, em seguida os LEDs vão de verde para amarelo e, por fim, para vermelho, repetindo essa sequência com exceção do azul que só aparece no início da execução. E, o LCD apresenta em sua primeira linha, no início da execução, a palavra AZUL (na primeira linha) até que, passa a não apresentar nada ao mesmo tempo que o led azul e, em seguida passa a apresentar VERDE na primeira linha e três corações na segunda, depois AMARELO na primeira linha e, por fim, VERMELHO na primeira linha e três corações na segunda linha, repetindo essa sequência, com exceção do AZUL que somente aparece no início da execução. A frequência de mudança no LCD é igual à frequência de mudança no LED e, a cor amostrada no LED é sempre igual à cor cujo nome é escrito no LCD.

1.2)

a)


Logic 2 [Logic - Connected] [Session 2]
File Edit Capture Measure View Help
D0 rs
D1 e
D2 Channel 2
D3 Channel 3
D4 Channel 4
D5 Channel 5
D6 Channel 6
D7 Channel 7
Drag to zoom
Session 0
Session 1
Session 2
550 ms

Dando zoom:


Logic 2 [Logic - Connected] [Session 2]
File Edit Capture Measure View Help
D0 rs
D1 e
D2 Channel 2
D3 Channel 3
D4 Channel 4
D5 Channel 5
D6 Channel 6
D7 Channel 7
Drag to zoom
Session 0
Session 1
Session 2
11 µs

Foram registrados 17 pulsos em um dos casos e, para o outro caso de degrau foram registrados 13 pulsos.

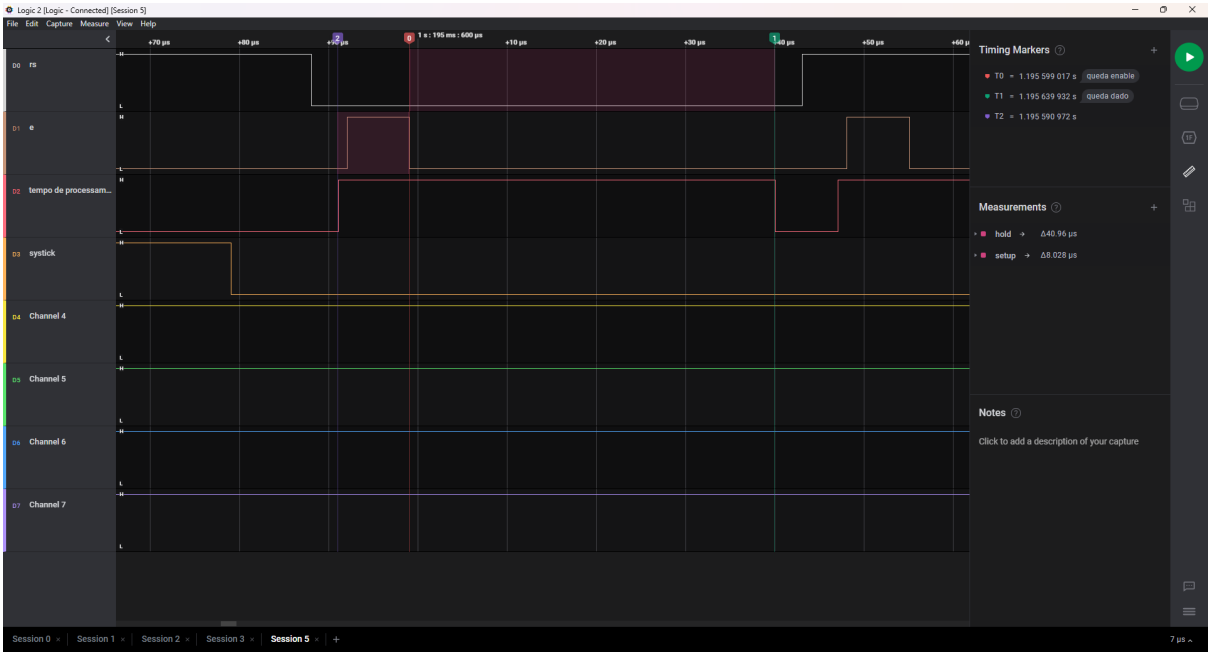
O caso de 17 pulsos ocorre na transição VERMELHO-VERDE, pois são impressas as strings: “VERDE ”, “ “ e três corações, ou seja, são impressos 14 bytes e outros 3 pulsos são causados pelos comandos de escrita das três strings, totalizando 17 pulsos. Enquanto que em VERDE-AMARELO, as strings são: “AMARELO”, “ ” e bells, cujo primeiro byte é ‘\0’, ou seja, nesse caso são impressos 10 bytes e os demais pulsos são causados pelos três comandos de escrita no lcd. Por fim, a transição AMARELO-VERMELHO é causadora de 17 pulsos, também, uma vez que as strings impressas são: “VERMELHO”, “ ” e três corações, ou seja, os caracteres que são impressos totalizam 14 bytes, enquanto que os demais pulsos são ocasionados pela execução dos comandos de escrita das três strings, totalizando 17 pulsos do sinal estudado.

Logo, notamos que, analisando o código, é esperado que tenhamos, mesmo, 13 e 17 pulsos dentro dos palitos.

usando a ferramenta de medida do app do analisador, temos que:

tempo de hold = 40,96 microssegundos;

tempo de setup = 8,028 microssegundos.



Medi esses tempos com a ferramenta de medida do aplicativo, conforme se percebe em rosa no print.

Comparando esses tempos com aqueles dados no datasheet, notamos que os que encontramos por meio do analisador são muito maiores que aqueles no datasheet, pois são da ordem de microssegundos, enquanto que aqueles especificados pelo fabricante são da ordem de nanosegundos.

b) Na função GPIO_escreveByteLCDH5Pins, o bloco de código:

```
GPIOC_PSOR = GPIO_PIN (9);
GPIOE_PSOR = GPIO_PIN(21);
espera_5us(1);
GPIOC_PCOR = GPIO_PIN (9);
GPIOE_PCOR = GPIO_PIN(21);
```

é um dos responsáveis pela forma de onda vista, uma vez que determina o envio de um pulso de sinal em E (enable) com um tempo maior que 450 ns. Além disso, entendemos por meio do código desta função, ainda, a forma de onda de PTE22 (intervalo de tempo de processamento em nível alto), pois é nesta função que se determina a subida e a descida desse sinal, nas linhas:

```
GPIOE_PSOR = GPIO_PIN(22);
```

e

```
GPIOE_PCOR = GPIO_PIN(22);
```

Já a função GPIO_setRSLCDH5Pin4 é responsável pela forma de onda em PTE20 (RS), nas linhas:

```
if(i == COMANDO) {
 GPIOC_PCOR = GPIO_PIN(8);              // Seta o LCD no modo de comando
 GPIOE_PCOR = GPIO_PIN(20);             // Mostra o estado do sinal no PTE20
}
else if (i == DADO) {
 GPIOC_PSOR = GPIO_PIN(8);      // Seta o LCD no modo de dados
 GPIOE_PSOR = GPIO_PIN(20);             // Mostra o estado do sinal no PTE20
}
```

pois essas linhas definem a mudança de RS para 1 (dados) e 0 (comandos).

Por fim, tratando da função GPIO_escreveStringLCDH5Pins, percebe-se que, apesar de não ter bloco que defina diretamente as formas de onda, é nesta função que são chamadas as duas discutidas anteriormente, ou seja, ela é responsável por chamar as funções que definem as formas de onda.

c) Ao multiplicar por dois o valor passado na chamada de espera_5us percebemos que, visualmente, a mudança não é relevante, embora possamos prever que o tempo entre a escrita dos caracteres é maior. Já se considerarmos um produto por 5000, vemos um intervalo entre a impressão dos caracteres. E, ao zerar o valor passado para a função, ocorre um problema não esperado, uma vez que, por o bloco em assembly que escrevi sair do loop quando o valor de 'multiplos' (valor que entra na função) for decrementado até zero, ocorre que, ao entrar valendo 0, ele é decrementado para -1 antes de ser comparado com o 0 na instrução de condição, portanto, o loop nunca termina. Então, ao arrumar esse problema colocando uma comparação no início do bloco em assembly, vemos que a escrita no LCD é prejudicada, pois, pela falta de espera adequada, várias letras e símbolos são suprimidos. Por fim, analisando os sinais por meio do analisador, percebemos que as falhas na escrita ocorrem por conta da alteração para 0 do valor passado na espera do fim da função, /*! * Aguarda pelo processamento */ espera_5us (t);, isso porque, ao medir os tempos de setup e hold pelo programa do analisador, é possível perceber que, apesar de serem menores que aqueles registrados anteriormente, ainda estão na ordem de microssegundos, ou seja, são muito maiores que aquele recomendado pelo produtor no datasheet, não representando, portanto, um malefício.

d) A função responsável por executar a sequência de inicialização proposta pelo fabricante é GPIO_initLCD(). Ela é chamada logo após a função GPIO_ativaConLCD.

e) Os registradores usados nesse momento são:

- GPIOC_PCOR
- GPIOC_PSOR
- GPIOC_PDOR
- GPIOE_PSOR
- GPIOE_PSOR
- GPIOE_PCOR

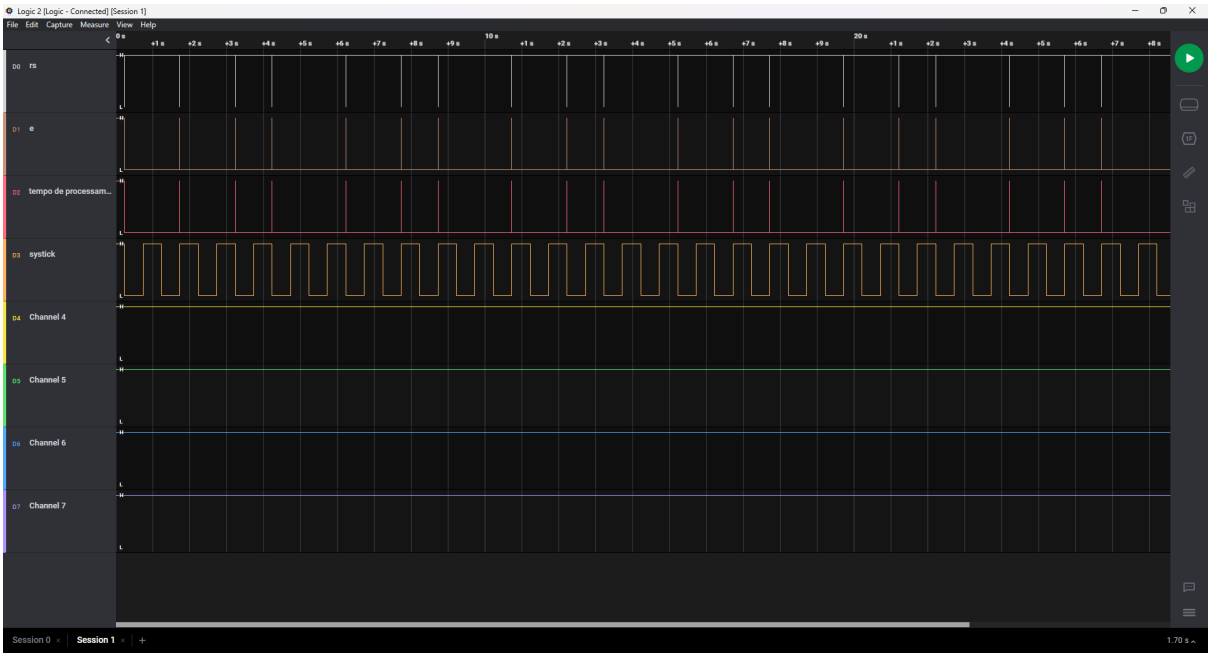
Dos quais, apenas os três primeiros são responsáveis pela renderização das mensagens, uma vez que os últimos três apenas são usados para exibir os sinais do lcd nas saídas PTE 20, 21, 22.

1.3)

a) O canal três apresenta pulsos de aproximadamente 500 ms (algumas medições são: 499.272 ms, 500.324 ms, 500.259 ms). Enquanto que um pulso de subida mais um de descida totalizam aproximadamente 1 s juntos.

b) Na linha GPIOE_PTOR = GPIO_PIN(23); na função SysTick_Handler, notamos que o pulso gerado no canal três é causado por um toggle, sendo que, portanto, o canal 3 reflete o período do Systick. Ou seja, o sinal em PTE23 registra os pulsos de Systick na saída por meio do registrador de Toggle, que permite que o sinal em PTE23 seja o mesmo que o produzido pelo Systick para um mesmo momento. Com base nas configurações realizadas na função systick_init em Systick.c, espera-se que a largura do pulso seja de um "periodo", onde "periodo" é o parâmetro da função. Portanto, neste caso, quando chamada em main.c com "periodo" = 10485760, percebemos que systick é configurado para ter um pulso de 500

ms. Logo, a largura medida por meio do programa do analisador oscila em valores muito próximos à largura esperada.



c) Se aumentarmos o valor de periodo passado em main.c para a função SysTick_init, notamos que o espaçamento entre palitos aumenta no programa do analisador. Enquanto que se diminuirmos o valor do periodo, percebemos um decréscimo no espaçamento entre palitos.
A relação entre o tempo medido e o valor de periodo está principalmente fundamentada no fato de que esse valor é o número de clocks do contador até que o pulso do Systick acabe, ou seja, ao configurar um valor de 10485760 para periodo, temos que, a uma taxa de 20971520 Hz do timer, o valor em segundos de um pulso é de aproximadamente 0,5 s. Sendo assim, ao aumentar esse valor, o tempo de um pulso aumenta e, ao diminuí-lo, o tempo de um pulso diminui.

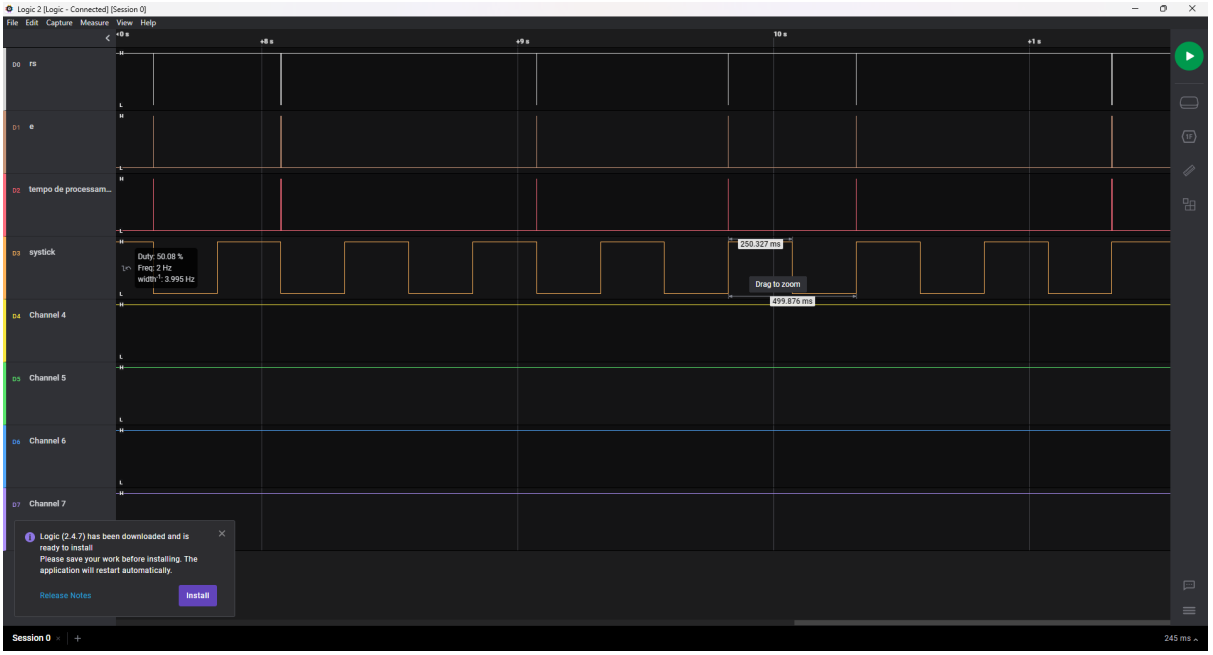
d) Um evento que pode causar interrupção em Systick é o overflow que ocorre quando se chega ao valor máximo de contagem, o que ocasiona a chamada de uma rotina de interrupção para tratar esse overflow e recomeçar a contagem.

e) Ao aumentar “valor” os leds RGB ficam mais tempo em uma mesma cor até mudarem para outra. Ao diminuí-lo, o tempo em uma mesma cor diminui, aumentando o número de trocas de cor por um mesmo intervalo. Isso faz sentido, pois como o postscaler divide a frequência de ocorrência de overflows, ele aumenta o tempo até que um overflow ocorra, logo, ao aumentar o “valor”, estamos aumentando o tempo até um overflow e, ao diminuí-lo estamos diminuindo esse tempo, e, contribuindo para que a mudança de cor seja mais rápida. Ainda assim, o período de Systick não é alterado por conta disso, uma vez que, apenas é alterada a frequência de overflows.

f) A função usada na main para zerar SYST_CVR é SysTick_resetaCVR(). E, isso não impacta no controle do espaçamento na mudança de cores.

g) A interrupção do SYSTICK é ativada na main após a inicialização do próprio SYSTICK, na linha SysTick_ativaInterrupt (). No entanto, a interrupção nunca é desativada, pois o programa roda indeterminadamente.

h) Usando o valor 5242880, máximo divisor comum desejado, temos o que se segue no print:



No print, nota-se que o período do SystIck foi alterado de 1s (pulso completo, alto e baixo) para aproximadamente 0,5s (pulso completo, alto e baixo).

1.4) A variável em questão é “estado”, que garante a correspondência correta entre main e a SysTick_Handler. Essa variável é declarada em ISR.c como estática e do tipo “tipo_estado”.

1.5)

a) São strings os arranjos: verde, vermelho, amarelo, azul, bells, hearts. Enquanto que bell e heart são bitmaps. O último byte não é identificado na função GPIO_escreveBitmapLCD, uma vez que o bitmap deve ser preenchido por completo e, o loop da função acaba sempre no oitavo e último byte. A função GPIO_escreveStringLCD não escreve lixo quando o segundo parâmetro é um bitmap, pois o bitmap não é terminado em ‘\0’, logo, não é interpretado como string e, não será, portanto, impresso.

b) Isso ocorre porque como a posição do bitmap bell está em 0x00, ocorre conflito com o elemento ‘\0’. Logo, para alterar isso, basta mudar a posição do bitmap na tabela, alterando o valor 0x00 por 0x02 em GPIO_escreveBitmapLCD, por exemplo.

c) O que ocorre ao retirarmos esses espaços brancos é a sobreposição de palavras de forma que, ao escrever uma palavra menor que a anterior, as letras da anterior que não forem substituídas por letras da nova, continuarão impressas, ocorrendo, por exemplo, “verdelo”, onde as três últimas letras não são substituídas nem por letras de verde, nem por caracteres vazios como deveriam ser. Portanto, o papel dos espaços em branco nas mensagens é impedir a junção da mensagem do momento com caracteres da mensagem anterior em posições do LCD onde não existiriam caracteres novos caso não existissem os espaços brancos.

d) O operador aplicado no segundo argumento para compatibilizar os tipos de dado foi *, operador cujo significado é “valor de”, ou seja, ele retorna o valor armazenado no endereço representado pelo ponteiro que o acompanha.

1.6) Da forma como são colocados os blocos de instruções no roteiro, a substituição não causaria problemas. No entanto, essa substituição não poderia ser feita no programa, uma vez que o bloco de instruções de cada caso tem de ser diferentes uns dos outros.

2)

main.c | ISR.c | SysTick.c | util.c | GPIO_latch_lcd.c | GPIO_ledRGB.c | GPIO_switches.c | derivative.h | MKL25Z4.h | util.h

```
/* @file GPIO_switches.c

#include "derivative.h"

void GPIO_initSwitchNMI (uint8_t IRQC, uint8_t prioridade) {
    /**
     * Configura o modulo SIM: ativacao dos sinais de relogio de PORTB
     * Ativacao dos sinais de relogio de PORTA
     */
    *(uint32_t volatile *) SIM_SCGC5 |= (1<<9);

    /**
     * Configura o modulo PORTA_PCR4
     * registrador PORTA_PCR4 mapeado em 0x40049010
     */
    *(uint32_t volatile *) PORTA_PCR4 &= ~0x700; // Zera bits 10, 9 e 8 (MUX) de PTA4
    (*(uint32_t volatile *) PORTA_PCR4) |= 0x00000100; // Seta bit 8 do MUX de PTA4

    /**
     * Configura o modulo GPIOA: sentido de entrada do sinal no pino PTA4, sem
     * interrupcao habilitada
     * registrador GPIOA_PDDR mapeado em 0x400ff014u
     */
    (*(uint32_t volatile *) GPIOA_PDDR) &= ~(1<<4);

    /**
     * Limpa flag de interrupcao do pino 4
     * registrador PORTA_PCR4 mapeado em 0x40049010u
     */
    (*(uint32_t volatile *) PORTA_PCR4) |= (1<<24);

    /**
     * Habilita a interrupcao do pino 4 para IRQC
     */
    *(uint32_t volatile *) PORTA_PCR4 &= ~0xF0000;
    (*(uint32_t volatile *) PORTA_PCR4) |= (IRQC << 16);

    /**
```

main.c | ISR.c | SysTick.c | util.c | GPIO_latch_lcd.c | GPIO_ledRGB.c | GPIO_switches.c | derivative.h | MKL25Z4.h | util.h

```
     */
    (*(uint32_t volatile *) GPIOA_PDDR) &= ~(1<<4);

    /**
     * Limpa flag de interrupcao do pino 4
     * registrador PORTA_PCR4 mapeado em 0x40049010u
     */
    (*(uint32_t volatile *) PORTA_PCR4) |= (1<<24);

    /**
     * Habilita a interrupcao do pino 4 para IRQC
     */
    *(uint32_t volatile *) PORTA_PCR4 &= ~0xF0000;
    (*(uint32_t volatile *) PORTA_PCR4) |= (IRQC << 16);

    /**
     * Configura o modulo NVIC: habilita IRQ 30
     * registrador NVIC_ISER mapeado em 0xe000e100u
     */
    (*(uint32_t volatile *) NVIC_ISER) = (1<<30);

    /**
     * Configura o modulo NVIC: limpa pendencias IRQ 30
     * registrador NVIC_ICPR mapeado em 0xe000e280u
     */
    (*(uint32_t volatile *) NVIC_ICPR) = (1<<30);

    /**
     * Configura o modulo NVIC: seta prioriodade para IRQ30
     * registrador NVIC_IPR7 mapeado em 0xe000e41cu (Tabela 3-7/p. 54 do Manual)
     */
    (*(uint32_t volatile *) NVIC_IPR7) |= (prioridade<<22);     //(Secao 3.3.2.1/p. 52 do Manual)

    return;
}
```