Experimento 4

Dado digital não viciado

Mattos, A. Guilherme
Universidade de Brasília
Faculdade Gama
Brasília, Brasil
guilherme.mattos.gam@gmail.com

Souza, Paulo Augusto M. F. de
Universidade de Brasília
Faculdade Gama
Brasília, Brasil
pauloaugustomiguelfonseca@gmail.com

***Abstract*— Using the MSP430 microcontroller to design a digital dice no addicted using low power mode, interrupts and Assembly in C.**

***Keywords*— MSP430, low power mode, interrupts, Assembly in C.**

I. OBJETIVOS

Utilizar o microcontrolador MSP430 para projetar um dado digital não viciado utilizando modo de baixo consumo, interrupções e Assembly em C.

II. INTRODUÇÃO

Microcontroladores, computer on a chips, diferenciam dos processadores, pois além dos componentes lógicos e aritméticos usuais de um microprocessador de uso geral, o microcontrolador integra elementos adicionais em sua estrutura interna, como memória de leitura e escrita para armazenamento de dados, memória somente de leitura para armazenamento de programas, EEPROM para armazenamento permanente de dados, dispositivos periféricos como conversores analógico/digitais (ADC), conversores digitais/analógicos (DAC) em alguns casos; e, interfaces de entrada e saída de dados. Um microcontrolador não utiliza um sistema operacional, somente um software especifico para uma aplicação. Para a programação geralmente é usado a linguagem C, ou Assembly, muitas vezes pode se mesclar as duas para que haja uma melhor performance, como utilizado neste experimento.

Uma das maiores vantagens do MSP430, utilizado no experimento, é o baixo consumo de energia, onde dependendo dos casos, é quase nenhum. O seu processador permite uma aritmética diretamente com valores da memória. O modo de economia de energia, faz com que o hardware economize energia enquanto não estiver sendo utilizado. O modo 4, utilizado nessa prática por exemplo, praticamente desliga todas as funcionalidade do microcontrolador, fazendo com que todos os clocks não sejam usados, desligando a CPU. Esses modos de baixo consumo deixa a corrente na faixa de µA conforme a Tabela 1. O MSP pode sair desses modos de baixo consumo utilizando interrupções como será mostrado neste relatório.

Tabela 1 - Tabela de consumo de cada modo *Low-Power*.

Low-Power Mode Supply Currents (Into V_{CC}) Excluding External Current

over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)(1) (2)

PARAMETER		TEST CONDITIONS	T_A	V_{CC}	MIN	TYP	MAX	UNIT
$I_{LPM0,1MHz}$	Low-power mode 0 (LPM0) current(3)	f_{MCLK} = 0 MHz, f_{SMCLK} = f_{DCO} = 1 MHz, f_{ACLK} = 32768 Hz, BCSCTL1 = CALBC1_1MHZ, DCOCTL = CALDCO_1MHZ, CPUOFF = 1, SCG0 = 0, SCG1 = 0, OSCOFF = 0	25°C	2.2 V		55		µA
I_{LPM2}	Low-power mode 2 (LPM2) current(4)	f_{MCLK} = f_{SMCLK} = 0 MHz, f_{DCO} = 1 MHz, f_{ACLK} = 32768 Hz, BCSCTL1 = CALBC1_1MHZ, DCOCTL = CALDCO_1MHZ, CPUOFF = 1, SCG0 = 0, SCG1 = 1, OSCOFF = 0	25°C	2.2 V		22		µA
$I_{LPM3,LFXT1}$	Low-power mode 3 (LPM3) current(4)	f_{DCO} = f_{MCLK} = f_{SMCLK} = 0 MHz, f_{ACLK} = 32768 Hz, CPUOFF = 1, SCG0 = 1, SCG1 = 1, OSCOFF = 0	25°C	2.2 V		0.7	1.0	µA
$I_{LPM3,VLO}$	Low-power mode 3 current, (LPM3)(4)	f_{DCO} = f_{MCLK} = f_{SMCLK} = 0 MHz, f_{ACLK} from internal LF oscillator (VLO), CPUOFF = 1, SCG0 = 1, SCG1 = 1, OSCOFF = 0	25°C	2.2 V		0.5	0.7	µA
I_{LPM4}	Low-power mode 4 (LPM4) current(5)	f_{DCO} = f_{MCLK} = f_{SMCLK} = 0 MHz, f_{ACLK} = 0 Hz, CPUOFF = 1, SCG0 = 1, SCG1 = 1, OSCOFF = 1	25°C	2.2 V		0.1	0.5	µA
			85°C			0.8	1.5	

(1) All inputs are tied to 0 V or to V_{CC}. Outputs do not source or sink any current.
(2) The currents are characterized with a Micro Crystal CC4V-T1A SMD crystal with a load capacitance of 9 pF.
(3) Current for brownout and WDT clocked by SMCLK included.
(4) Current for brownout and WDT clocked by ACLK included.
(5) Current for brownout included.

III. DESENVOLVIMENTO

A. Descrição do Software

O experimento consiste em produzir um dado digital não viciado, como se pode ter certeza que um dado realmente consiga colocar números aleatoriamente, sem favorecer nenhum número? Ao criar um código em linguagem C que ao o usuário apertar o botão venha um valor aleatório entre 1 e 6 não se sabe o número de instruções que a máquina está fazendo para cada número, caso haja algum número com mais instruções este será favorecido, logo o dado é viciado. Para criar um dado não viciado, o uso da linguagem Assembly é requerido, já que pode criar o mesmo número de instruções pra qualquer um dos números de 1 a 6, garantindo assim um dado não viciado. O problema em utilizar essa linguagem é sua alta complexidade e extensão. Uma solução para esse problema é a junção das duas linguagens, C e Assembly, a linguagem C utilizada no corpo do código, e onde se encontra a lógica pra selecionar os números de 1 a 6 o Assembly é usado para garantir que o dado não seja viciado. Essa junção é feita utilizando o comando "#include<legacymsp430.h>", onde possui uma biblioteca já pronta pra utilizar o código em Assembly junto ao C. O código em C é colocado durante o corpo e criando funções, e a inserção do Assembly é feita com

o comando "__asm__", após isso entre parênteses e aspas fica o código que se deseja implementar na linguagem Assembly.

Além de juntar duas linguagens de programação, neste experimento foi necessário desenvolver o modo de baixo consumo e utilizar interrupções. O modo de baixo consumo 4 como mostrado na Tabela 1, desliga os geradores de clock e a CPU, criando assim um estado de hibernação onde a corrente utilizada está na faixa de 0,1 µA, essa é uma das grandes vantagens de utilizar o MSP430, permanecer num estado onde quase não há gasto de energia torna desse microcontrolador uma ótima ferramenta. A interrupção, como o próprio nome já diz, interrompe o programa tanto utilizando um botão, como uma contagem, uma borda de descida ou subida de algum sinal, após o programa sofrer uma interrupção é necessário que a flag seja apagada para o programa sair da interrupção e o código voltar a ser executado, colocando o P1IFG igual a 0. No programa criado para o dado não viciado o usuário pressionará o botão, o MSP430 sairá do estado de hibernação, fará o que foi proposto, no caso escolher um valor de 1 a 6 e retorna para seu modo de baixo consumo.

O código consiste de 3 funções e uma interrupção, a função "DELAY" que é responsável pelo atraso nas saídas dos displays para que o usuário possa ver o número ligado, a função "numface" que vai ligar os display com os números de 1 a 6, essa função foi a que apresentou uma dificuldade considerável, ao chamar ela em Assembly, não fica claro onde está a variável "num", então foi necessário que ao compilar o código em C utilizando o CODE COMPOSER fosse identificado o registrador onde essa variável estava sendo utilizada, feito isso foi averiguado que o registrador era o R11, então foi passado o valor correspondente da face que estava no R15 para esse R11, esse processo foi perigoso, pois é necessário ter cuidado com os registradores, e que valores se coloca neles, para futuros experimentos não é recomendável a utilização desse método, no entanto para essa prática mais simples esse processo não acarretou nenhum problema. Imaginando que o R11 tendo algum valor de 1 a 6, que será a variável chamada "num", foi criada uma lógica para que a saída dependesse desse valor alocado em "num", utilizando Ifs, dependendo do valor ele ligaria os leds respectivos para mostrar o valor da face, após ligar o display é chamada a função "DELAY" que já foi mencionada, bem como sua função, depois foi utilizado um return para um label criado no final da interrupção, para o programa apagar o display, finalizar a interrupção e voltar pro modo baixo consumo 4. A função principal segue a mesma lógica dos outros experimentos com o adicional de alguns comandos novos, utilizados na interrupção e no modo de baixo consumo. São declaradas as saídas como os bit0, bit1,bit2,bit4,bit5 e bit6, a entrada como o bit3 que é o próprio botão do MSP. É utilizado também no início da função main o comando P1IE (interrupt enable) que é colocado como o botão "BTN", já definido como o BIT3, definindo assim que quem habilita a interrupção é o botão, depois é utilizado o comando P1IES, que define se a interrupção será chamada na borda de descida ou subida, colocando igual ao botão, é o mesmo que definir o valor 1, ou seja, borda de descida, isso quer dizer que quando o botão é pressionado já ativa a interrupção, o que teve um uso muito importante para a lógica do código. Ao colocar "_BIS_SR (LPM4_bits + GIE) " seta os registradores responsáveis pelo modo de baixo consumo 4 e pela interrupção, habilitando assim interrupções mascaráveis e deixando o MSP430 em modo de baixo consumo 4. Logo abaixo da "main" é colocado o formato de interrupção para o gcc, e o código assembly para que no momento em que a interrupção seja ativada já sejam lidas as instruções. O código em Assembly consiste em uma lógica que ao soltar o botão, lembrando que o botão pressionado tem nível baixo, e solto nível alto, logo ao soltar o botão a entrada volta pra nível alto, e o código para na instrução em que ele solta, carregando um valor no registrador R15 que será utilizado na função "numface" como já mencionado.

B. Descrição do Hardware

Componentes utilizados:

- 1 protoboard;
- 1 display de sete segmentos cátodo comum;
- MSP430.

Conforme especificado no código anexado, o pino P1.3 foi usado para receber a entrada do botão quando ele fosse acionado ou até mesmo quando ele fosse liberado, de acordo com a lógica da programação. Os pinos P1.0, P1.1, P1.2, P1.4, P1.5 e P1.6 foram usados como saídas. O P1.0, cor vermelha no esquemático abaixo, foi usado para os leds A e D do display, pois como verificado eles sempre ligavam juntos ou apagavam juntos considerando os números de 1 a 6, essa foi uma forma de tentar minimizar as portas utilizadas, utilizando esse aproveitamento foi economizada um pino. Após verificar o datasheet do display de 7 segmentos (Figura 2), o P1.0 foi usado para o led A e D, o P1.1 para o B o P1.2 para o C, o P1.4 para o E, o P1.5 para o F e o P1.6 para o G. A saída GND foi ligada no comum do display, pois ele é catodo comum e os leds são ativos com nível alto de energia. Como no experimento, foi usado o modo 4 de economia de energia, todos os sistemas do MSP estavam desligados. Ele só ativava ao se apertar o botão, e logo depois de mostrar a saída do dado, ele retornava ao modo de baixo consumo, apagando os displays.

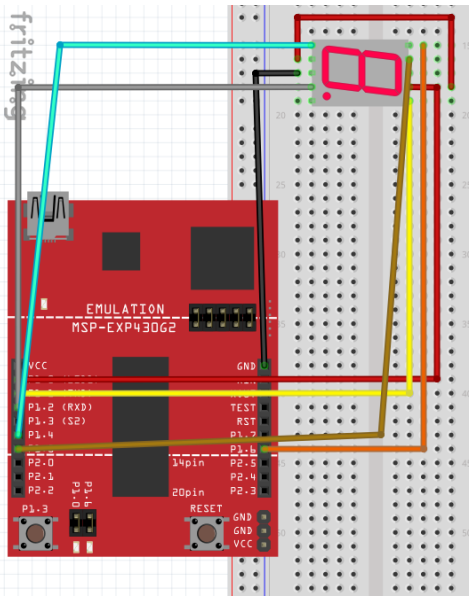

Figura 1 – Esquemático da montagem.

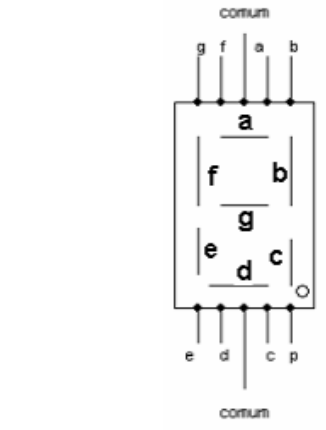

Figura 2 - Pinagem do display de 7 segmentos.

IV. RESULTADOS

Ao construir o código muitas dificuldades foram apresentadas, como o problema da função "numface" que ao ser chamada indicava que precisava do valor que seria colocado na variável "num", já dito anteriormente. Outra dificuldade era estar aparecendo o primeiro número da instrução, no caso o valor 1, isso é porque a lógica da parte em Assembly estar usando o "JNZ", que comparava a flag Z de forma que o resultado da operação de uma máscara no caso o valor 8 (00001000) que ao fazer uma AND com o botão (BIT3) resultava sempre em 0 quando o botão estava pressionado, como a operação AND dava 0 ele não saltava e já carregava o valor 1, que é a primeira instrução (if1), a solução pra esse problema está foi utilizar o "JZ" que ao fazer o AND saltava com o valor 0, fazendo várias instruções, e somente ao soltar o botão, o que faz a operação AND resultar em 1, zerando a flag Z, logo não salta mais e carrega o valor da instrução que ele parou. Resultando assim em um dado não viciado conforme os requisitos solicitados. Na hora da implementação foi identificado outro erro, ao chamar a função "numface" o display ligava vários números, ele não ligava apenas um e retornava onde foi chamado, talvez porque ao voltar ele não estivesse zerando o flag de interrupção e carregando outros valores. A solução para esse problema foi colocar um Assembly em cada if, ao ligar o display e aguardar um tempo antes de apagar, ele apresenta um JMP para voltar a interrupção, apagar o display, zerar a flag de interrupção e retornar ao modo de baixo consumo. Após efetuar essa solução o experimento funcionou conforme as condições solicitadas.

V. CONCLUSÃO

A criação de um dado digital não viciado aparenta ser bem simples, no entanto com este trabalho é possível perceber diversos conceitos ligados ao que seria um simples dado. Mesmo com os conceitos já definidos da linguagem de programação C, da linguagem de máquina Assembly, do microcontrolador MSP430, ainda assim foram inseridos novos conceitos, de interrupção, baixo consumo e junção de C e Assembly. Essas novas ferramentas apresentadas fornecem novas maneiras de projetar, o fato de deixar em modo de baixo consumo 4 faz do MSP430 um microcontrolador mais adequado para condições onde o aparelho terá que ter pouca energia, como em ambientes onde não há disponibilidade de energia deixando o aparelho com apenas uma bateria e com o modo de baixo consumo ele pode permanecer por um bom tempo ligado, a interrupção também é uma ótima ferramenta a ser utilizada junto ao modo de baixo consumo, para detectar um determinado dado, ele sai do modo de baixo consumo, efetua seu papel no qual foi programado e logo após pode voltar ao modo de baixo consumo, isso torna o MSP430 útil para diversas formas de trabalho. O ato de juntar uma linguagem que é mais fácil de se utilizar, no entanto apresenta algumas limitações, que é o caso da linguagem C, com uma linguagem mais complexa que é o Assembly, torna o processo de programar bem mais abrangente, criando um número maior de possiblidades e soluções.

REFERÊNCIAS

[1] John H. Davies, MSP430 Microcontroller Basics, Elsevier, 2008.
[2] Software: Energia.
[3] Software: Fritzing.
[4] Software: Code Composer Studio.

Anexos

```
////////////////////////////////////PINAGEM/////////////////////////////////////////////

/*            |
     BIT0 v
            _____
           |     |
BIT5-> |_____| <- BIT1
           |BIT6^|
BIT4-> |_____| <- BIT2

         BIT0 ^|


*/

////////////////////////////////////////////////////////////////////////////////////////

#include <msp430g2553.h>
#include <legacymsp430.h>
#define BTN  BIT3
#define DLY 19000

void DELAY(volatile unsigned long int t)
{
volatile unsigned long int i;
for(i=0;i<t;i++) { }
}



void numface(int num)
{
__asm__(        "numface:\n"
"mov.w r15,r11");  // DE ACORDO COM A COMPILAÇÃO A VARIAVEL NUM RECEBE O VALOR DE R11

// LÓGICA PARA LIGAR OS LEDS DEPENDENDO DO VALOR DA FACE DO DADO

if (num==1)
{
P1OUT|= BIT1+ BIT2 ;
DELAY(DLY);
__asm__("jmp RETURN");
}


if (num==2)
{
P1OUT|= BIT0 + BIT1 + BIT6 + BIT4 ;
DELAY(DLY);
__asm__("jmp RETURN");
}
```

```
if (num==3)
{
P1OUT|= BIT0 + BIT1 + BIT6 + BIT2;
DELAY(DLY);
__asm__("jmp RETURN");
}

if (num==4)
{
P1OUT|= BIT1+ BIT2 + BIT5 + BIT6;
DELAY(DLY);
__asm__("jmp RETURN");
}

if (num==5)
{
P1OUT|= BIT0+ BIT2 + BIT5 + BIT6;
DELAY(DLY);
__asm__("jmp RETURN");
}

if (num==6)
{
P1OUT|= BIT0+ BIT2 + BIT4 + BIT5 + BIT6 ;
DELAY(DLY);
__asm__("jmp RETURN");
}

else __asm__("jmp RETURN");

}


int main (void)
{
WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer
P1SEL  = 0;                     // Utilizar pinos para a porta P1
P1SEL2 = 0;                     // Utilizar pinos para a porta P1
P1REN  = BTN;                  // Habilitar resistor de pull-up/down no pino do botão
P1OUT  =  BTN;                 // LED2 on, Resistor do botão conectado a VCC
P1DIR  = (BIT0 + BIT1 + BIT2 + BIT4 + BIT5 + BIT6);    // setar pinos > saídas
P1DIR  = ~(BTN); // entrada
P1IES = BTN; // Configurar interrupção no botão por borda de descida
P1IE = BTN ; // Habilitar interrupção via botão



_BIS_SR ( LPM4_bits + GIE ) ;




}
```

```
interrupt ( PORT1_VECTOR ) P1_ISR ( void ) // Formato de interrupção para o gcc para MSP430
{
__asm__(

//if (1)
"if_face1: \n "
"mov.b &__P1IN, r15 \n "
"and.b #8, r15 \n "
"jz if_face2 \n "
"mov.b #1, r15 \n "
"call #numface \n "

//if (2)
"if_face2: \n "
"mov.b &__P1IN, r15 \n "
"and.b #8, r15 \n "
"jz if_face3 \n "
"mov.b #2, r15 \n "
"call #numface \n "

//if (3)
"if_face3: \n "
"mov.b &__P1IN, r15 \n "
"and.b #8, r15 \n "
"jz if_face4 \n "
"mov.b #3, r15 \n "
"call #numface \n "

//if (4)
"if_face4: \n "
"mov.b &__P1IN, r15 \n "
"and.b #8, r15 \n "
"jz if_face5 \n "
"mov.b #4, r15 \n "
"call #numface \n "

//if (5)
"if_face5: \n "
"mov.b &__P1IN, r15 \n "
"and.b #8, r15 \n "
"jz if_face6 \n "
"mov.b #5, r15 \n "
"call #numface \n "

//if (6)
"if_face6: \n "
"mov.b &__P1IN, r15 \n "
"and.b #8, r15 \n "
"jz if_face1 \n "
"mov.b #6, r15 \n "
"call #numface \n "

//ELSE
"jmp if_face1" ) ;

__asm__("RETURN:");
```

```
P1OUT &= ~((BIT0+BIT1+BIT2+BIT4+BIT5+BIT6));

P1IFG =0;  // Apagar flag de interrupção

}
```