Universidade do Vale do Paraíba

Faculdade de Engenharias, Arquitetura e Urbanismo

Engenharia Elétrica / Eletrônica

Estudo dos Microcontroladores MSP430 por intermédio do desenvolvimento de interface com a placa didática McLab2

Autor: Júlio César de Carvalho Vieira

Orientador: Prof. Msc. Helosman Valente de Figueiredo

Relatório do Trabalho de Conclusão de Curso apresentado à Banca Avaliadora da Faculdade de Engenharias, Arquitetura e Urbanismo da Universidade do Vale do Paraíba, como parte dos requisitos para obtenção do Título de Bacharel em Engenharia Elétrica / Eletrônica.

São José dos Campos – SP

Junho/2017

RESUMO

O uso de microcontroladores nas atividades industriais e domésticas vem crescendo a cada dia devido ao crescente desenvolvimento de sistemas automatizados, trazendo relevância cada vez maior ao estudo destes componentes pelos estudantes de Engenharia Elétrica/Eletrônica. Contudo, os microcontroladores da família **MSP430** desenvolvidos pela Texas Instruments e estudados atualmente na Univap/FEAU, possuem uma placa de desenvolvimento de baixo custo (*LaunchPad* MSP-EXP430G2) muito limitada no que diz respeito à quantidade de periféricos instalados, limitando também o estudo das funções que o componente oferece, o que resulta na necessidade de se montar circuitos externos com outros periféricos para a elaboração de experimentos que explorem todas as funções do microcontrolador. Assim, com o objetivo de ampliar-se o aprendizado do estudante de Engenharia Elétrica/Eletrônica na disciplina de microcontroladores, através de um rápido e prático acesso a experimentos que o auxiliem neste aprendizado, o presente trabalho apresenta o estudo dos MSP430 realizado com o auxilio de uma placa didática comercial, a McLab2 originalmente projetada para os microcontroladores da família PIC. Para o desenvolvimento deste estudo, as seguintes atividades foram cumpridas: estudo da placa didática McLab2, onde avaliou-se suas características elétricas, além dos circuitos e periféricos disponíveis, seguido do estudo das características elétricas e da distribuição dos pinos da *LaunchPad,* comparando-as com os dados analisados na McLab2, de modo a realizar-se o projeto de uma placa de interface para conexão entre a *LaunchPad* e a McLab2. Após o estudo das placas McLab2 e *LaunchPad* e do desenvolvimento do projeto da placa de interface, foi possível confeccioná-la e, por fim, elaborar-se e executar-se os experimentos destinados ao estudo das principais funções do MSP430, por meio de programação do microcontrolador em linguagem C. Dessa maneira, pode-se concluir que a interface entre *LaunchPad* e McLab2 traz grande funcionalidade ao estudante de Engenharia Elétrica/Eletrônica na medida em que é possível ampliar a gama de experimentos realizados com o microcontrolador e aprofundar-se no entendimento de suas funções principais, além de desenvolver a programação em linguagem C e otimizar a depuração de erros ou falhas na programação em tempo real, aumentando assim a eficiência do aprendizado em microcontroladores.

Palavras-chave: Microcontroladores. MSP430. LaunchPad. McLab2.

ABSTRACT

The use of microcontrollers in industrial and domestic activities is growing day by day due to the increasing of automated systems development, raising the importance of studying these components by students of Electrical/Electronic Engineering. However, the MSP430 microcontrollers family developed by Texas Instruments and currently studied at Univap/FEAU have a low-cost development board (LaunchPad MSP-EXP430G2) which is very limited in terms of the number of peripherals, resulting in the need of assembling external circuits with other peripherals for experiments that explore all the microcontroller functions. Thus, with the aim of expanding the student's learning in microcontrollers by a quick and practical access to experiments that help in this learning, this work presents the MSP430 study developed with the support of a commercial didactic board, the McLab2. For this study, the following activities were accomplished: the McLab2 board study, analysing its electrical characteristics and the available circuits and peripherals, followed by the LaunchPad board study, analysing its electrical characteristics and distribution of the pins, comparing them with the data analysed in the McLab2, in order to design of an interface board for connection between the LaunchPad and the McLab2. After these studies and with the interface board designed, it was possible build it and, finally, to prepare and run the experiments of studying the MSP430 main functions, by microcontroller programming in C-language. In this way, it can be concluded that the interface between LaunchPad and McLab2 brings great functionality to the student of Electrical/Electronic Engineering, because it is possible to extend the range of experiments performed with the microcontroller and deepen understanding of its core functions, as well as developing C-language programming and optimizing debugging of errors or faults in real-time programming, increasing the efficiency of microcontrollers learning.

Keywords: Microcontroladores. MSP430. LaunchPad. McLab2.

SUMÁRIO

INTRODUÇÃO

Nos dias atuais, com a evolução na automação de muitas das atividades industriais e domésticas que anteriormente não eram realizadas de maneira automatizada, o estudo dos microcontroladores – componentes responsáveis pelo controle dos sistemas automatizados – torna-se cada vez mais importante dentro da Engenharia Elétrica/Eletrônica e é indispensável que os estudantes dessa área possuam um amplo domínio destes componentes, conhecendo seu funcionamento, características e aplicabilidades. Desta forma, o estudo dos microcontroladores e suas aplicações deve ter uma metodologia que auxilie o estudante no entendimento do funcionamento e programação deste componente.

Dentre os vários modelos disponíveis no mercado, os microcontroladores da família MSP430 fabricados pela Texas Instruments são os que atualmente vêm sendo utilizados no ensino acadêmico pela Univap/FEAU. Os MSP430 são microcontroladores que possuem como características principais seu ultra-baixo consumo de energia, facilidade de compilação em linguagem C, CPU preparada para operar em modo de baixo consumo e periféricos inteligentes e de baixo consumo [1].

Contudo, a placa de desenvolvimento de baixo custo disponibilizada pela Texas Instruments (Figura 1, *LaunchPad* modelo MSP-EXP430G2) possui uma quantidade reduzida de dispositivos periféricos, 2 *leds* (*light-emitting diode*) e 2 *push buttons* (botões), o que consequentemente limita o estudo de todas as funcionalidades que o microcontrolador oferece, sendo que, para operações envolvendo outros periféricos, como *displays*, sensores e motores e estudos adicionais das aplicações do MSP430, se faz necessária a montagem de circuitos auxiliares para estes periféricos, o que concorre diretamente com o tempo dedicado ao estudo do MSP430.

Figura 1 – MSP430 *LaunchPad*, modelo MSP-EXP430G2.

Fonte: Texas Instruments (2016).

Dessa forma, para que o estudante de Engenharia Elétrica/Eletrônica mantenha seu foco exclusivamente no estudo do microcontrolador e de suas aplicações, é essencial o fácil acesso a estes circuitos auxiliares para utilização de periféricos, trazendo mais elementos à realização de experimentos que ampliem de forma robusta a compreensão do estudante. Todos estes circuitos podem ser compilados em uma placa ditática, também conhecida como placa de desenvolvimento destinada ao estudo de microcontroladores, sendo que atualmente existem no mercado placas ditáticas desenvolvidas para esta finalidade.

Assim, este trabalho tem por objetivo apresentar o estudo dos Microcontroladores MSP430 por intermédio do desenvolvimento de interface de uma placa didática comercial, que será suporte à elaboração dos experimentos complementares àqueles existentes na *LaunchPad* MSP-EXP430G2. Este estudo será baseado no estudo de uma placa de desenvolvimento comercial, no estudo da *LaunchPad* MSP-EXP430G2, no projeto de interface entre as placas e na programação em linguagem C e execução dos experimentos correpondentes ao estudo das principais características do MSP430.

MATERIAIS E MÉTODOS

O trabalho desenvolvido a seguir dividiu-se nas seguintes etapas: a primeira etapa consistiu-se na definição e estudo da placa didática a ser utilizada para complementar o estudo do MSP430, seguido do estudo da *LaunchPad* MSP-EXP430G2 e desenvolvimento de interface entre as placas, e, por fim, na utilização de todo conjunto, com a elaboração dos experimentos através de programação.

Estudo da placa didática comercial:

Inicialmente, fez-se necessária a escolha da placa didática com a qual os experimentos com o MSP430 seriam realizados. No mercado é possível encontrar modelos de placas didáticas destinadas ao desenvolvimento de projetos de microcontroladores, algumas delas utilizando os da família PIC. A própria Univap/FEAU possui placas didáticas para este modelo de microcontrolador, uma vez que o PIC esteve presente na grade curricular da instituição, sendo este o antecessor do MSP430 na disciplina de microcontroladores.

Dessa maneira, no intuito de se aproveitar os equipamentos já disponibilizados pela instituição e que atualmente estão subutilizados devido à substituição do microcontrolador estudado, optou-se por utilizar as placas didáticas existentes na Univap, que são as placas

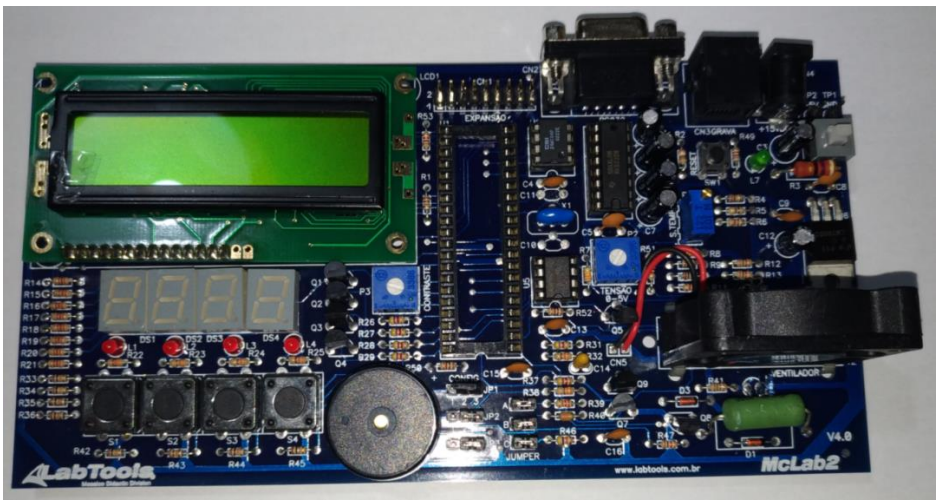
McLab2 – 16F, fabricadas pela Mosaico. Estas placas foram desenvolvidas para o estudo do PIC16F877A e, portanto, necessitam ser adaptadas para sua utilização com o MSP430.

Nesta placa, é possível encontrar diversos periféricos que auxiliam o estudo das funcionalidades de um microcontrolador. Estão disponíveis na McLab2 [2]:

- LCD (*Liquid Crystal Display*) alfanumérico;
- *Displays* de *led* de 7 segmentos;
- Teclas e *leds*;
- *Buzzer*;
- Medidor de tensão (conversão A/D);
- Sensor de temperatura;
- Aquecedor;
- Ventilador;
- Tacômetro;
- Leitura de *jumpers*;
- Memória *Serial* EEPROM 24C04;
- Comunicação *Serial* RS232.

A McLab2 é alimentada por uma fonte de 15 V_{DC}, contudo, os circuitos auxiliares para utilização dos periféricos presentes na placa foram projetados considerando tensões de 5 V_{DC} como alimentação, uma vez que este é o nível de tensão de operação do microcontrolador PIC16F877A. Por isso, a placa contém uma fonte de tensão interna, responsável pela conversão de 15 V_{DC} para 5 V_{DC}.

Figura 2 – Placa McLab2 16F



Fonte: Mosaico.

Assim como são utilizados para o PIC, os periféricos presentes na McLab2 também podem ser usados no estudo das funcionalidades do MSP430 e, portanto, necessita-se analisar a viabilidade de interface destes circuitos com a *LaunchPad,* de modo a disponibilizá-los para utilização no estudo deste microcontrolador.

Estudo da *LaunchPad* MSP-EXP430G2 e projeto de interface com a McLab2:

Uma vez definida e estudada a placa de desenvolvimento a ser utilizada, a etapa subsequente consistiu em estudar-se a *LaunchPad* MSP-EXP430G2, de modo a possibilitar o desenvolvimento de uma interface com a McLab2 para a utilização dos periféricos desta placa pelo MSP430. Assim, foi necessária uma avaliação das características elétricas da *LaunchPad*, verificando-se as tensões de alimentação e tensões dos pinos de entrada e saída do microcontrolador. Além disso, necessitou-se avaliar a distribuição dos circuitos presentes na McLab2 para conexão a *LaunchPad,* definindo-se quais pinos do MSP430 deveriam ser conectados a quais circuitos da McLab2.

Conforme se verificou no estudo da McLab2, a placa foi projetada para trabalhar com tensão de 5 V_{DC} para alimentação do PIC e da maioria dos periféricos como, por exemplo, o *display* LCD. Temos então uma diferença importante em relação ao MSP430: enquanto o PIC trabalha com tensão de 5 V_{DC} em seus pinos (alimentação e portas I/O), a tensão de operação no MSP430G2553, presente na *LaunchPad* é de 3,6 V_{DC} [3] .

Esta diferença na tensão de operação faz com que seja necessária avaliação da compatibilidade dos circuitos da McLab2 e o MSP430, de tal maneira que não haja risco de danificar o microcontrolador por sobretensão em seus pinos. Assim, analisaram-se as características da *LaunchPad*, comparando-as com as da McLab2, onde obteve-se às seguintes conclusões:

- **Alimentação do Microcontrolador:** na placa McLab2 a alimentação ocorre através de uma fonte externa de 15 V_{DC}, posteriormente convertida para 5 V_{DC} por intermédio de fonte interna contida na McLab2 e que, por fim, alimenta o microcontrolador, no caso o PIC16F877 [4]. No caso da *LaunchPad*, a alimentação é feita através de uma porta USB, que por sua vez também é de 5 V_{DC}. Ocorre que nesta placa existe uma fonte interna que converte esta tensão de 5 V_{DC} para os 3,6 V_{DC} necessários à alimentação do MSP430 [5]. Assim, não há restrições para a alimentação da *LaunchPad* via McLab2, desde que esta alimentação seja conectada juntamente à porta USB e não diretamente ao microcontrolador. Entretanto, como a

programação do MSP430 necessita da utilização do cabo USB, optou-se por manter a alimentação da *LaunchPad* exclusivamente pela entrada USB.

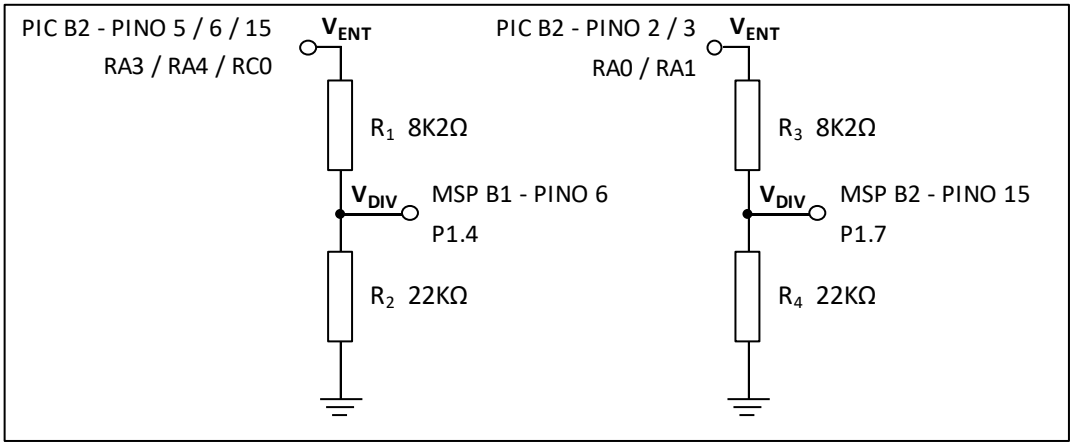
- **Pinos de Entrada:** para os circuitos onde a conexão com o microcontrolador é feita através de uma entrada, ou seja, quando o microcontrolador recebe o sinal proveniente de um dos circuitos auxiliares, fez-se necessária a realização de dois tratamentos diferentes. O primeiro tratamento foi necessário para regular a tensão nos pinos responsáveis por receber os sinais analógicos da McLab2, provenientes dos circuitos de Medição de Tensão, Sensor de Temperatura e Leitura de *Jumpers*, além do sinal digital proveniente do circuito Tacômetro. Nestes circuitos, o sinal de saída pode ser de até 5 V_{DC} e, portanto, necessitam se convertidos para um sinal máximo de 3,6 V_{DC} na interface entre as duas placas, uma vez que o MSP430 não pode operar com nível de tensão acima deste último valor. Para a realização desta tarefa, projetou-se montar na placa de interface divisores de tensão para os pinos 6 e 15 do MSP430. A escolha pelo divisor de tensão deve-se ao fato de que os pinos 6 e 15 serão utilizados para leitura de sinais analógicos e, por isso, um conversor de nível lógico não seria apropriado uma vez que neste tipo de componente, a conversão só ocorre na faixa de nível lógico alto, enquanto que para os sinais analógicos é necessária a conversão de tensão em toda a sua faixa.

O divisor de tensão foi definido conforme cálculo abaixo e seu esquema elétrico de acordo com a Figura 3:

$V_{Divisor} = V_{Entrada} \times \frac{R_2}{R_1+R_2}$, onde R_1 (8,2 KΩ) e R_2 (22 KΩ) são os resistores do divisor.

$V_{Divisor} = 5 \times \frac{22K}{8,2K + 22K} = 3,64$ V.

Figura 3 – Esquema elétrico dos divisores de tensão da placa de interface

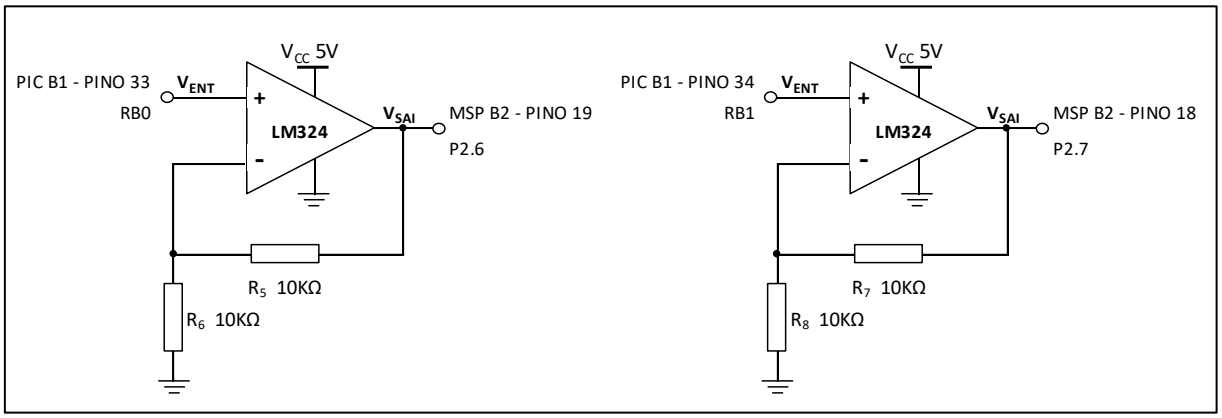


Fonte: O autor.

O segundo tratamento necessário à interface das placas, foi realizado nos circuitos das Teclas (S1 a S4) da placa McLab2. Devido à característica construtiva da placa didática, onde há a presença de *Leds* (L1 a L4) instalados no mesmo circuito das teclas, de tal forma que este circuito possa trabalhar tanto como entrada quanto saída dependendo da necessidade do programador, o sinal de tensão proveniente destes circuitos quando estão trabalhando como saída é da ordem de 1,8V. Este nível de tensão é inferior ao limite máximo necessário para que o MSP430 assuma o sinal como nível lógico alto (0,75xVcc = 2,7V), o que prejudica o funcionamento das teclas quando conectadas a este microcontrolador [6].

Dessa maneira, para que o sinal de tensão fosse regulado de forma a atingir o nível ideal ao correto funcionamento das teclas quando conectadas ao MSP430, fez-se necessário o projeto de instalação de um amplificador operacional na saída dos circuitos. Nesta implementação foi utilizado o modelo LM324, conforme esquema elétrico da Figura 4:

Figura 4 – Esquema elétrico do circuito amplificador operacional conectado às teclas do McLab2



Fonte: O autor.

Em função desta implementação, os circuitos onde os amplificadores operacionais necessitaram ser acoplados tiveram sua função reduzida, passando a operar apenas como saída. Assim, para que se pudessem utilizar teclas e *leds* da McLab2, definiram-se os circuitos das teclas S1 e S2, aos quais projetou-se acoplar os amplificadores operacionais, para serem utilizadas exclusivamente como saída, enquanto os circuitos dos *leds* L3 e L4 foram designados para trabalhar exclusivamente como entrada.

- **Pinos de Saída:** para os circuitos onde a conexão com o microcontrolador é através de uma saída, ou seja, quando o microcontrolador emite o sinal para o funcionamento do circuito auxiliar, verificou-se que estes mantêm seu funcionamento normal mesmo recebendo

um sinal de 3,6 V_{DC}, incluindo o *display* de LCD que apesar de operar com 5 V_{DC}, pode receber em seus pinos de controle e dados o nível de tensão gerado pelo MSP430 [7]. Neste caso, a comutação entre as placas pode ser realizada diretamente, sem a necessidade de adição de conversores tensão, seja analógica ou digital.

- **Distribuição dos circuitos da McLab2 para conexão com a *LaunchPad* MSP-EXP430G2:** após a verificação das características elétricas das placas, seguiu-se com a etapa de avaliação da distribuição dos circuitos da placa McLab2 para o MSP430. Nesta análise, a quantidade de pinos existentes no PIC e no MSP430 foi o grande desafio para a elaboração da interface entre as placas. Isto devido a grande diferença de quantidade entre elas, pois enquanto o PIC16F877A possui um total de 40 pinos, sendo 33 portas I/O, o MSP430G2553 utilizado na *LaunchPad* possui 20 pinos apenas, com 16 portas I/O.

Por esta razão, diferentemente do que originalmente ocorre na McLab2, onde cada circuito é conectado a um pino exclusivo no PIC, para a conexão da *LaunchPad* foi necessário o compartilhamento dos pinos do microcontrolador em diferentes circuitos da placa didática, sendo a comutação entre um ou outro circuito realizado por intermédio de *jumpers*. Este compartilhamento trouxe como desvantagem a limitação dos circuitos que puderam ser conectados ao MSP430, além da impossibilidade de utilização de todos os circuitos de forma simultânea. Desta forma, fez-se a distribuição dos circuitos da placa de desenvolvimento conforme Quadro 1:

Quadro 1 – Distribuição dos circuitos da McLab2 para operação com o MSP430

Pino PIC	**Nome PIC**	**Placa McLab2**	**Pino MSP**	**Nome MSP**
1	MLCR	Botão de reset manual	NC	
2	RA0	Entr analóg do sensor de temperatura	15	P1.7/A7/CA7
3	RA1	Entr analóg do potenciômetro P2	15	P1.7/A7/CA7
4	RA2	Conector de expansão	NC	
5	RA3	Entr analóg fixa em 2,5V (Vref+)	6	P1.4/SMCLK/VREF+/VEREF+/A4/CA4
6	RA4	Leitura de *jumpers*/tacômetro	6	P1.4/SMCLK/VREF+/VEREF+/A4/CA4
7	RA5	*Buzzer*	4	P1.2/TA0.1/A2/CA2
8	RE0	RS do LCD alfanumérico	7	P1.5/TA0.0/A5/CA5
9	RE1	*ENABLE* do LCD alfanumérico	8	P2.0/TA1.0
10	RE2	Conector de expansão	NC	
11	Vdd	+5V	NC	
12	Vss	GND	20	DVSS

Pino PIC	Nome PIC	Placa McLab2	Pino MSP	Nome MSP
13	OSC1	Ressonador cerâmico de 4MHz	NC	
14	OSC2	Ressonador cerâmico de 4MHz	NC	
15	RC0	Tacômetro	6	P1.4/SMCLK/VREF+/VEREF+/A4/CA4
16	RC1	Ventilador	9	P2.1/TA1.1
17	RC2	Aquecedor	14	P1.6/TA0.1/A6/CA6
18	RC3	*Clock* memória *serial* 24C04	NC	
19	RD0	LCD (LSB) / Segmento A *display*	2	P1.0/TA0CLK/ACLK/A0/CA0
20	RD1	LCD / Segmento B *display*	3	P1.1/TA0.0/A1/CA1
21	RD2	LCD / Segmento C *display*	4	P1.2/TA0.1/A2/CA2
22	RD3	LCD / Segmento D *display*	5	P1.3/A3/CA3
23	RC4	*Data* memória *serial* 24C04	NC	
24	RC5	Conector de expansão	NC	
25	RC6	Comunicação *serial* (RX)	NC	
26	RC7	Comunicação *serial* (TX)	NC	
27	RD4	LCD / Segmento E *display*	10	P2.2/TA1.1
28	RD5	LCD / Segmento F *display*	11	P2.3/TA1.0
29	RD6	LCD / Segmento G *display*	12	P2.4/TA1.2
30	RD7	LCD / Segmento DP *display*	13	P2.5/TA1.2
31	Vss	GND	20	DVSS
32	Vdd	+5V	NC	
33	RB0	Tecla S1 / ~~*Led* L1~~	19	P2.6/TA0.1
34	RB1	Tecla S2 / ~~*Led* L2~~	18	P2.7
35	RB2	~~Tecla S3~~ / *Led* L3	7	P1.5/TA0.0/A5/CA5
36	RB3	~~Tecla S4~~ / *Led* L4	8	P2.0/TA1.0
37	RB4	Comum do *Display* DS4	7	P1.5/TA0.0/A5/CA5
38	RB5	Comum do *Display* DS3	8	P2.0/TA1.0
39	RB6	Comum do *Display* DS2	9	P2.1/TA1.1
40	RB7	Comum do *Display* DS1	14	P1.6/TA0.1/A6/CA6

**NC = Não Conectado*

Fonte: O autor.

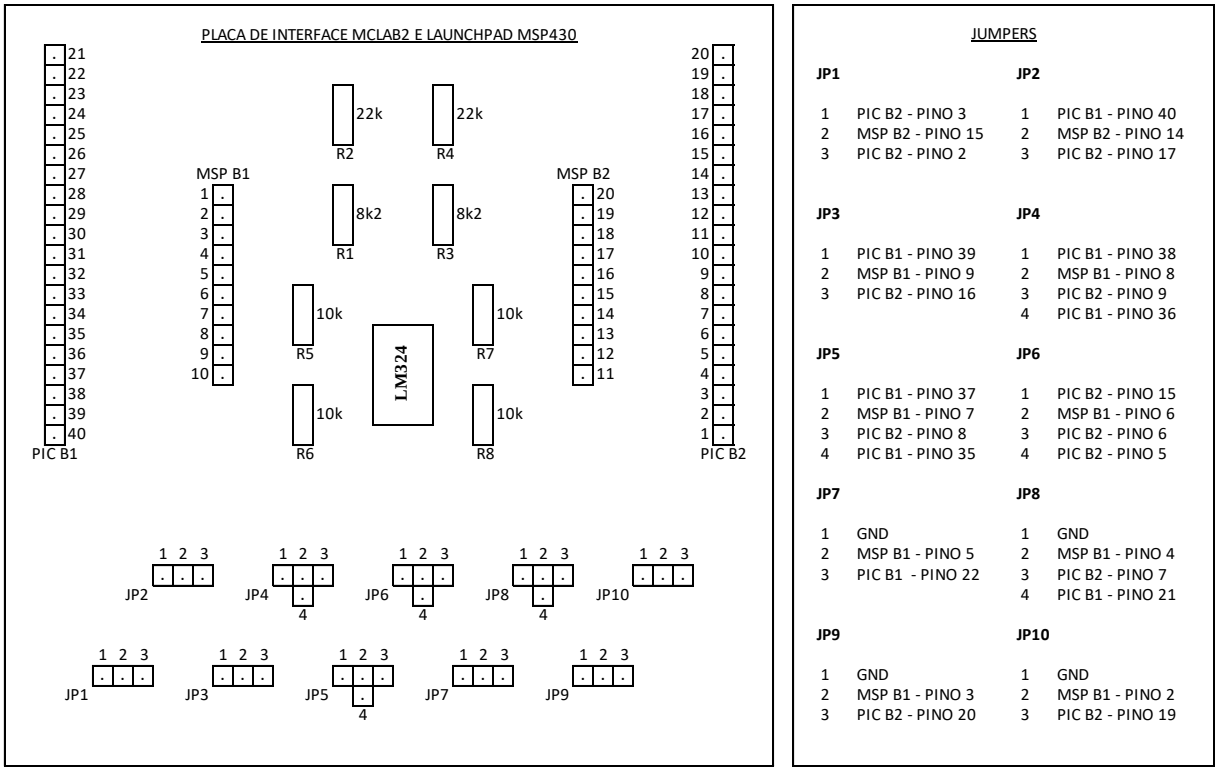
Conforme se verifica no Quadro 1, foi possível realizar a interface dos seguintes circuitos da McLab2 com a *LaunchPad*: LCD alfanumérico, *displays* de *led* de 7 segmentos, teclas e *leds*, *buzzer*, medidor de tensão, sensor de temperatura, aquecedor, ventilador, tacômetro e leitura de *jumpers*. Já entre os circuitos que não puderam ser conectados temos:

- Ressonador cerâmico de 4MHz: devido à possibilidade de instalação de um cristal oscilador diretamente na LaunchPad, a não utilização deste componente não impede os estudos referentes a osciladores no MSP430 [8].

- Comunicação *Serial* RS232: similar ao caso anterior, é possível emular a comunicação serial RS232 pela porta USB da *LaunchPad* e, portanto, sua não utilização também não prejudica o aprendizado.
- Memória *Serial* EEPROM 24C04: neste caso, devido a sua operação em 5 V_{DC}, não é possível realizar a interface com o MSP430, uma vez que o pino SDA é utilizado para transmissão e recepção de dados, o que implica em receber e emitir sinais por uma única via. Dessa forma, para sua interface seria necessária a utilização de um conversor de nível lógico 3,3V - 5V bidirecional com detecção automática de sentido de sinal, o que não foi possível implementar.

Por fim, avaliadas as diferenças nas características elétricas entre as placas e realizado o projeto dos circuitos necessários para a adequação da interface entre elas, além da análise da distribuição dos circuitos da McLab2 para os pinos do MSP430, projetou-se o *layout* da placa de interface e a listagem de conexões dos *jumpers*, conforme Figuras 5:

Figura 5 – *Layout* da placa para interface e lista de conexões dos *Jumpers*.



Fonte: O autor.

Para a confecção da placa de interface projetada na Figura 5, necessita-se dos seguintes componentes descritos no Quadro 2:

Quadro 2 – Lista de componentes necessários à confecção da placa de interface

Componente	**Quantidade**
Placa ilhada 10 x 10 (cm)	01 und.
Barra de Pinos 1x20 - 90 Graus	02 und.
Barra de Pinos 1x10 - 180 Graus	02 und.
Barra de Pinos 1x40 - 180 Graus	01 und.
Conector mini *jumper*	10 und.
Resistor 8K2 Ω	02 und.
Resistor 22K Ω	02 und.
Resistor 10K Ω	04 und.
Soquete 14 pinos estampado	01 und.
Amplificador Operacional LM324N	01 und.
Cabo alumínio Ø 1mm	03 m.
Cabo *Jumper* Macho-Fêmea	40 und.

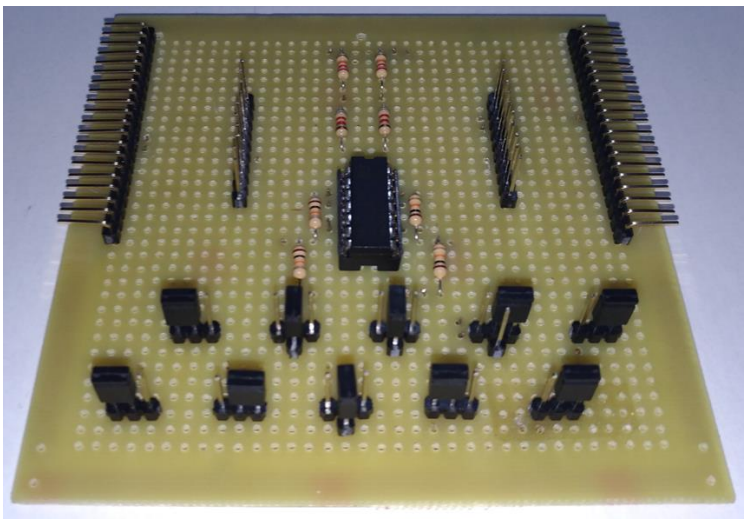
Fonte: O autor.

RESULTADOS E DISCUSSÃO

Desenvolvimento da placa de interface entre LaunchPad e McLab2

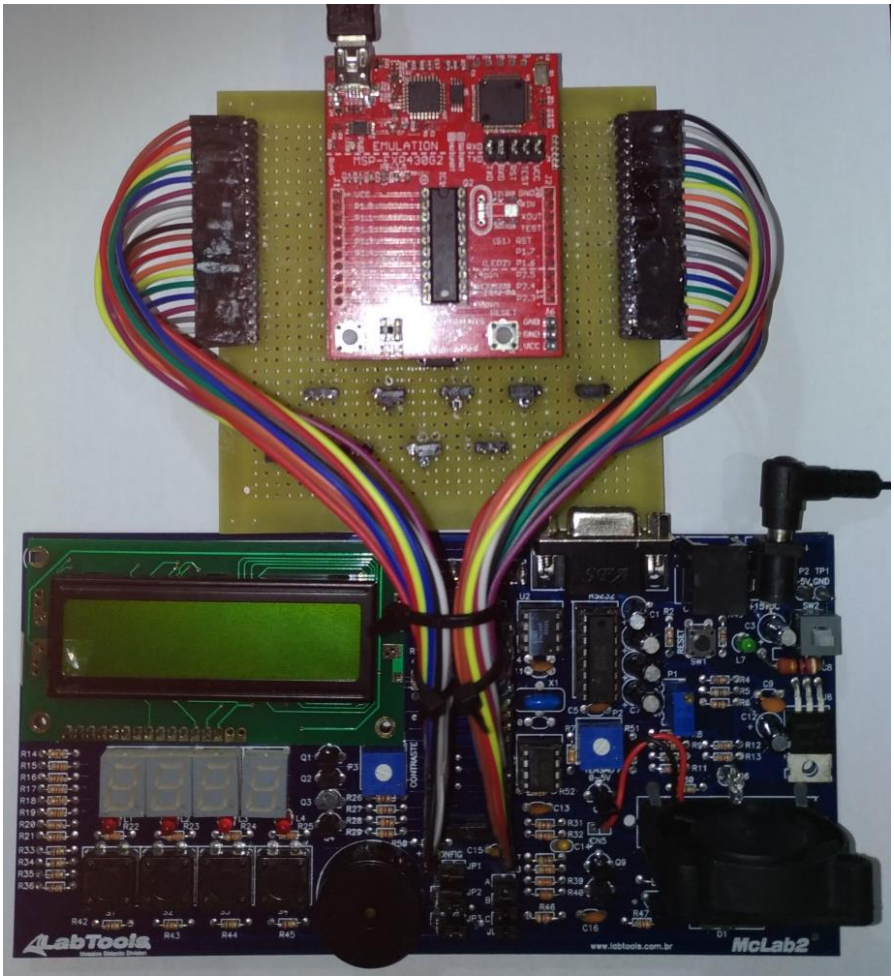
Finalizadas as etapas de estudo das placas didáticas (*LaunchPad* e McLab2) e também o projeto da placa de interface entre elas, realizou-se o desenvolvimento do protótipo da placa. Para esta construção, utilizaram-se os componentes discriminados no Quadro 2 e o resultado da montagem pode ser observado nas Figuras 6 e 7:

Figura 6 – Placa para interface entre a *LaunchPad* e a McLab2, onde observa-se os conectores para a *LaunchPad* (centrais), conectores para a placa McLab2 (laterais), divisores de tensão e amplificador operacional LM324 (centro) e conjunto de *jumpers* para conexão dos circuitos (inferior).



Fonte: O autor.

Figura 7 – Placa para interface conectada à McLab2 e à *LaunchPad*



Fonte: O autor.

De posse da placa de interface finalizada, tornou-se possível realizar a conexão da *LaunchPad* com os diversos circuitos presentes na McLab2. A relação de circuitos e os respectivos jumpers que devem ser conectados para sua utilização são apresentados no Quadro 3:

Quadro 3 – Circuitos e conexões da McLab2 para operação com o MSP430

Circuito	**Posição de Operação dos *Jumpers* na Placa de Interface**									
	JP1	**JP2**	**JP3**	**JP4**	**JP5**	**JP6**	**JP7**	**JP8**	**JP9**	**JP10**
Display de 7 segmentos	-	12	12	12	12	-	23	24	23	23
Display LCD	-	-	-	23	23	-	-	12	12	12
Botão S1/*Led* L1	Não utiliza *jumpers* – Conectado diretamente									
Botão S2/ *Led* L2	Não utiliza *jumpers* – Conectado diretamente									
Botão S3/*Led* L3	-	-	-	-	24	-	-	-	-	-
Botão S4/ *Led* L4	-	-	-	24	-	-	-	-	-	-

Circuito	Posição de Operação dos *Jumpers* na Placa de Interface									
	JP1	**JP2**	**JP3**	**JP4**	**JP5**	**JP6**	**JP7**	**JP8**	**JP9**	**JP10**
Buzzer	-	-	-	-	-	-	-	23	-	-
Ventilador	-	-	23	-	-	-	-	-	-	-
Tacômetro	-	-	-	-	-	12	-	-	-	-
Aquecedor	-	23	-	-	-	-	-	-	-	-
Sensor de Temperatura	23	-	-	-	-	24	-	-	-	-
Medidor de Tensão	12	-	-	-	-	-	-	-	-	-
Leitor de *Jumpers*	-	-	-	-	-	23	-	-	-	-

Fonte: O autor.

Programação e execução de experimento para o estudo do MSP430

Com a finalização da montagem da placa de interface, tem-se o material disponível para o início do estudo dos microcontroladores MSP430, através da execução de experimentos utilizando os periféricos presentes na McLab2. Dessa maneira, realizaram-se os testes de funcionamento da placa didática através da elaboração de experimentos destinados ao estudo do MSP430 com os periféricos presentes na McLab2, onde obteve-se o desenvolvimento de exemplos que abrangem as seguintes características dos MSP430:

- **Portas I/O (*Input/Output*):** o MSP430G2553 de 20 pinos, presente na *LaunchPad*, possui um total de 16 pinos de I/O, subdivididos em 2 Portas (Port1 e Port2) com 8 *bits* cada [9]. Estes pinos são configuráveis via *software*, isto é, cada um deles pode ser configurado ora como entrada ora como saída de maneira independente. Ainda via *software*, é possível determinar o nível lógico dos pinos programados como saída.

Para o estudo das portas I/O, se faz necessária a realização de um experimento onde o estudante possa praticar as configurações das portas I/O, tanto na determinação de sua função como Entrada ou Saída, quanto na definição do nível lógico que as portas devem ter em determinado momento do experimento. Dessa maneira, elaborou-se um contador numérico utilizando os *displays* de 7 segmentos presentes na McLab2. Existem 4 destes displays na placa, o que permite a criação de um contador de 0 a 9999.

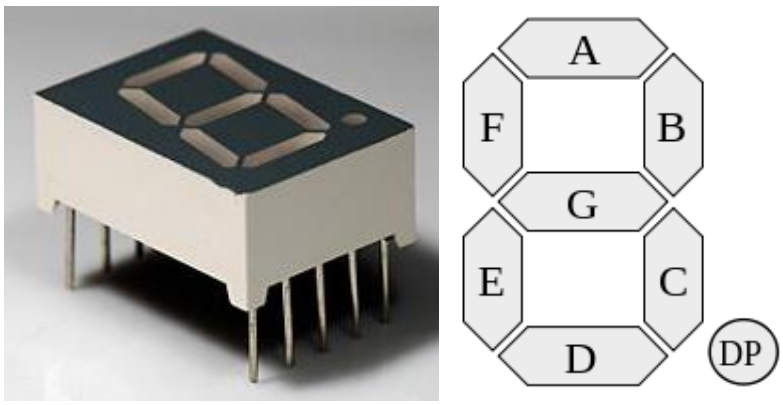
Assim, para se programar o contador fez-se necessária a configuração como saída das Portas I/O que estão conectadas aos segmentos do display, conforme Quadro 4 e Figura 8. Esta configuração é realizada através do registrador PxDIR, sendo que os *bits* habilitados em nível lógico alto (1) neste registrador referem-se às portas que estão configuradas como saída.

Quadro 4 – Portas I/O e pinos correspondentes aos segmentos do *display*

Instrução	**MSP430G2553**	
	Porta	**Pino**
Segmento A	P1.0	2
Segmento B	P1.1	3
Segmento C	P1.2	4
Segmento D	P1.3	5
Segmento E	P2.2	10
Segmento F	P2.3	11
Segmento G	P2.4	12
Segmento DP	P2.5	13
DS1 (Milhar)	P1.5	7
DS2 (Centena)	P2.0	8
DS3 (Dezena)	P2.1	9
DS4 (Unidade)	P1.6	14
GND	GND	20

Fonte: O autor.

Figura 8 – *Display* de 7 segmentos e configuração dos segmentos



Fonte: Wikipedia.

Para a configuração dos *displays*, os registradores P1DIR e P2DIR foram configurados da seguinte maneira:

```
P1DIR = 0x6f; // Config. P1.0, P1.1, P1.2, P1.3, P1.5 & P1.6 as outputs.
P2DIR = 0x3f; // Config. P2.0, P2.1, P2.2, P2.3, P2.4 & P2.5 as outputs.
```

Em seguida foi necessária a programação da sequência de controle do nível logico de cada uma das portas, determinando quais delas deveriam ser definidas com nível baixo (0) ou nível alto (1) à medida que a contagem ocorre, através da configuração do registrador

PxOUT. Por exemplo, inicialmente a portas P1.0, P1.1, P1.2, P1.3, P2.2 e P2.3 devem estar definidos com nível alto, enquanto a porta P2.4 fica em nível baixo, para se formar o algarismo “0”. Em seguida, apenas P1.1 e P1.2 devem estar definidos em nível alto e as demais portas em nível baixo para ter-se o algarismo “1” e assim sucessivamente. Adicionalmente, como temos segmentos do *display* conectados tanto a pinos da Porta1 quanto da Porta2 do MSP430, foi necessária a avaliação da sequência de controle de nível lógico de cada uma das portas de maneira distinta, de tal forma que a combinação destas sequências resultasse no algarismo a ser representado, conforme Quadro 5.

Quadro 5 – Portas I/O e pinos correspondentes aos segmentos do display

Algarismo	**dp**	**g**	**f**	**e**	**d**	**c**	**b**	**a**	**Port1**	**Port2**
	P2.5	P2.4	P2.3	P2.2	P1.3	P1.2	P1.1	P1.0		
0	0	0	1	1	1	1	1	1	0x0f	0x0c
1	0	0	0	0	0	1	1	0	0x06	0x00
2	0	1	0	1	1	0	1	1	0x0b	0x14
3	0	1	0	0	1	1	1	1	0x0f	0x10
4	0	1	1	0	0	1	1	0	0x06	0x18
5	0	1	1	0	1	1	0	1	0x0d	0x18
6	0	1	1	1	1	1	0	1	0x0d	0x1c
7	0	0	0	0	0	1	1	1	0x07	0x00
8	0	1	1	1	1	1	1	1	0x0f	0x1c
9	0	1	1	0	1	1	1	1	0x0f	0x18

Fonte: O autor.

Além disso, as portas P1.5, P1.6, P2.0 e P2.1 também necessitam de um controle espacial, uma vez que elas funcionam como “chave *on/off*” dos *displays*, ou seja, quando uma destas portas tem nível alto em sua saída o *display* ao qual está conectado entra em operação. Já quando a saída está em nível baixo, então o *display* é desligado e não mostra nenhuma informação, mesmo que as portas dos segmentos estejam em nível alto. Contudo, como os pinos que alimentam os segmentos do *display* são comuns aos 4 existentes na placa, se todos os *displays* estiverem ligados ao mesmo tempo, inevitavelmente eles mostrariam o mesmo algarismo. Neste caso, é necessária a aplicação da técnica de multiplexação dos *displays*, que consiste em ligar alternadamente os *displays* de maneira sequencial e numa frequência superior à nossa percepção, de tal forma que o olho humano não consiga enxergar essa alternância dos *displays*.

Adicionalmente, deve haver uma sincronização entre o momento que o *display* será ativado e o respectivo algarismo que deve ser representado neste *display*, de tal forma que não haja representação do algarismo de um *display* desligado naquele em operação no momento.

Com este experimento, foi possível conhecer a configuração necessária para operação das portas I/O no modo *Output* (saída), através do registrador responsável pela determinação do modo de operação das portas (PxDIR), bem como praticar o controle dos níveis lógicos das portas via *software*, através do registrador PxOUT.

- **Sistema de Interrupções:** o MSP430 possui um sistema de interrupções composto por vetores de 16 *bits*, cada um contendo uma sequência própria de tratamento da interrupção. Possuem prioridade de execução não-configurável, o que significa dizer que os eventos de interrupção ocorrem de acordo com a prioridade pré-definida no microcontrolador, onde o vetor 31 é o de maior prioridade e o vetor 0 o de menor prioridade.

Além disso, alguns vetores de interrupção podem ter mais de uma origem de ativação e, por isso, possuem um ou mais flags de interrupção, que são indicadores desta origem [10]. Para o estudo das interrupções, é importante que o estudante realize experimentos onde se faça necessária a implementação de uma rotina de tratamento quando de um evento gerador de interrupção no programa executado pelo microcontrolador. Para este estudo, elaborou-se a programação do LCD alfanumérico, que terá a função de exibir as interações realizadas através do botão S2 da *LaunchPad*.

Inicialmente, realizaram-se as conexões necessárias à operacionalização do LCD, conforme Quadro 6:

Quadro 6 – Portas I/O e pinos correspondentes aos segmentos do display

Instrução	**MSP430G2553**	
	Porta	**Pino**
RS do LCD alfanumérico	P1.5	7
ENABLE do LCD alfanumérico	P2.0	8
LCD Seg D0	GND / NC	20
LCD Seg D1	GND / NC	20
LCD Seg D2	GND / NC	20
LCD Seg D3	GND / NC	20
LCD Seg D4	P2.2	10
LCD Seg D5	P2.3	11
LCD Seg D6	P2.4	12
LCD Seg D7	P2.5	13

**NC = Não Conectado*

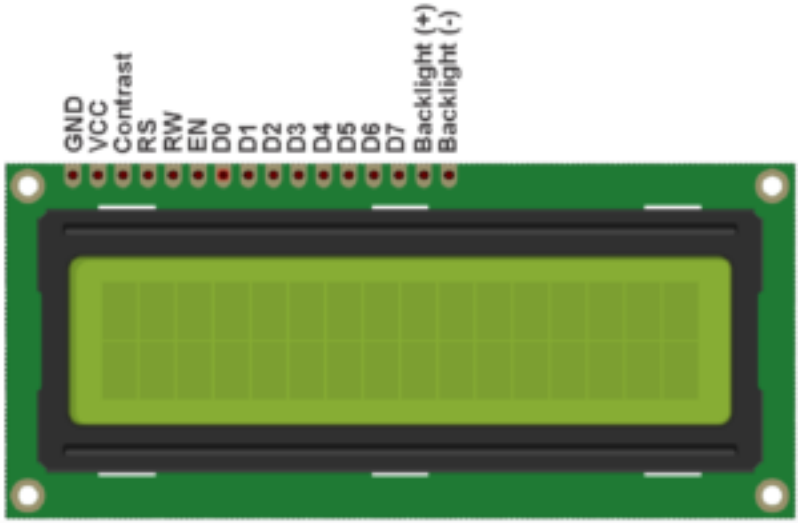
Fonte: O autor.

Como é possível observar, o LCD disponível na McLab2, possui 2 *bits* de controle (RS e *ENABLE*) disponíveis e mais 8 *bits* de dados (D0 a D7) pelos quais a informação é enviada ao LCD para que seja mostrada no *display*. Porém como é possível utilizar o LCD em dois modos de operação, com 8 *bits* ou 4 *bits*, no intuito de se economizar pinos I/O para outras funções a conexão com a *LaunchPad* utilizará o LCD no modo 4 *bits* (D4 a D7), reduzindo a necessidade de pinos de dados à metade.

Existem ainda outros 4 pinos no LCD, conforme é possível verificar na Figura 9: o primeiro, denominado R/W, é responsável pelo controle do fluxo de dados entre o *display* e o microcontrolador, sendo que quando o pino está no modo R (*READ*) o microcontrolador está "lendo" as informações contidas no LCD e no modo W (*WRITE*) o microcontrolador está "escrevendo" no LCD. No caso da McLab2, o LCD é utilizado exclusivamente no modo *WRITE* e, portanto, este pino está conectado diretamente a GND.

O segundo pino adicional (Contrast) é responsável pelo ajuste de contraste do LCD. Este pino está conectado ao potenciômetro P3 na McLab2 e pode ser ajustado manualmente. Por fim, os dois outros pinos adicionais (*Backlight*+ / *Backlight*-) são responsáveis pelo controle da iluminação *backlight* do LCD e, no caso da McLab2 não são utilizados.

Figura 9 – Modelo de display LCD e configuração dos pinos.



Fonte: Circuits4you.com.

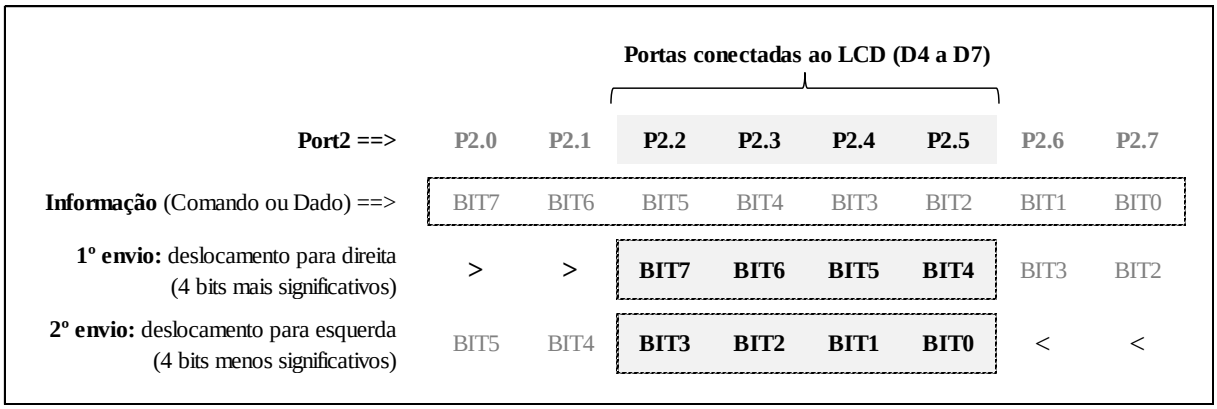
Dessa forma, partiu-se para a programação do LCD, através da verificação do seu *datasheet*, que contém as configurações necessárias à inicialização e controle do *display*, além das instruções para o procedimento de "escrita" no LCD [11]. Adicionalmente, realizaram-se pesquisas na internet para verificação de comandos adicionais para controle do LCD, como

posicionamento do cursor em pontos diferentes da tela, através do endereçamento do *display* [12].

Para o desenvolvimento da implementação, fez-se necessária a criação de rotinas de inicialização, envio de comandos, envio de texto, envio de dados e escrita no LCD. Devido à configuração do *display* no modo de operação em 4 *bits*, o envio de comandos e dados que são constituídos por informações de 8 *bits*, necessitam ser enviados em dois pacotes de 4 *bits*, sendo primeiramente enviados os 4 *bits* mais significativos e, posteriormente, os 4 *bits* menos significativos.

Por essa razão, nas rotinas de envio de comandos e envio de dados, foi preciso tratar as informações que seriam transmitidas em função da necessidade de envio dividido em dois pacotes e também em virtude das portas que foram conectadas ao LCD. Este tratamento do dado consistiu em deslocar os bits à esquerda e/ou à direita de tal forma que os dados ficassem disponíveis nas portas conectadas ao LCD para posterior envio, conforme demonstrado na Figura 10.

Figura 10 – Tratamento das informações de comando e dados



Fonte: O autor.

Após a implementação do *display* de LCD, iniciou-se a segunda etapa do experimento que consistia em atualizar a exibição do *display* ao se pressionar o botão S2 presente na *LaunchPad*. Este botão está conectado a porta P1.3 do microcontrolador, que por sua vez, deve operar como entrada neste experimento, recebendo a informação de acionamento do botão. Neste caso, isto significa que o registrador P1DIR deve ter seu BIT3 habilitado em nível baixo (0) para que a porta P1.3 seja considerada como entrada. Além disso, fez-se necessária a configuração de P1.3 em outros registradores, de tal maneira que garanta o correto funcionamento do botão S2. Estes registrados são responsáveis pela habilitação de

Resistor de *Pull-up/Pull-down* (P1REN), seleção de modo *Pull-up* (P1OUT), habilitação da interrupção (P1IE), seleção do sentido de borda da interrupção (P1IES) e limpeza do *flag* de interrupção (P1IFG). No experimento em questão, os registradores foram configurados conforme a seguinte sequência:

```
P1REN |= BIT3;       // Enable Pull-up/Down Resistor in P1.3
P1OUT |= BIT3;       // Select pull-up mode in P1.3
P1IE  = BIT3;        // Enable interrupt in P1.3
P1IES = BIT3;        // Hi-Lo edge in P1.3
P1IFG &= ~BIT3;      // Clear interrupt flag in P1.3
```

Outro registrador essencial para o correto funcionamento da interrupção é o registrador de habilitação de interrupções. Sem a configuração deste registrador as interrupções não são ativadas, mesmo que o evento gerador de uma interrupção ocorra, como por exemplo o acionamento do botão S2 do experimento em questão. O registrador de habilitação é configurado através do código:

```
__bis_SR_register (GIE);     // Enable global interrupt
```

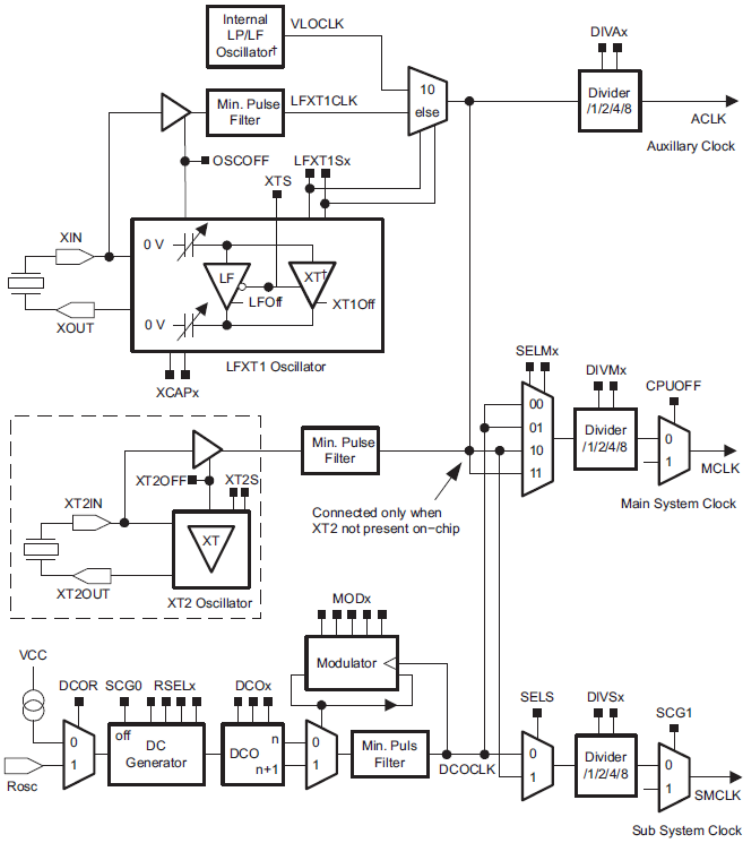
Desta forma, configurados os registradores de P1.3 e de habilitação de interrupções. A última etapa consistiu em realizar o tratamento da interrupção gerada pelo acionamento de S2. No experimento, foi criado um código que incrementa a quantidade de vezes que o botão foi pressionado e mostra esta informação no *display*. Nesta etapa do experimento, foi necessária a implementação de uma função que converte valores de uma variável do tipo *int* (inteiro) em *char* (caractere), uma vez que o LCD somente aceita a escrita deste último tipo de dado. Esta necessidade deve-se ao fato de que o dado a ser exibido no display é declarado na programação como variável *int*, uma vez que se faz necessária sua utilização em cálculos. Após uma série de pesquisas, identificou-se que a função necessária para esta conversão é a *itoa* (*integer to array*), mas que não é implementada na IDE (*Integrated Development Environment*) IAR do MSP430. Como alternativa, encontrou-se na internet uma forma de se implementar a função *itoa* de diretamente no programa do experimento, o que foi realizado [13]. Por fim, com a rotina de tratamento da interrupção foi programada com o seguinte código:

```
#pragma vector = PORT1_VECTOR  //
__interrupt void Port_1(void){
  counter++;                    // Incremental counter of button pressed
  itoa(counter, char_counter);  // Convert integer count to char char_count
  send_command(0x01);           // Clear display screen
  send_command(0x80);           // Set cursor to beginning to 1st line
  send_string("Button counter:");  // Write on 1st line
  send_command(0xC0);           // Set cursor to beginning to 2nd line
  send_string("Pressed: ");     // Write on 2nd line
  send_string(char_counter);    // Write char_counter on 2nd line
  send_string(" x");            // Write on 2nd line
  send_command(0x0C);           // Display on, cursor off
  __delay_cycles(50000);        // Delay
  P1IFG &= ~BIT3;               // Clear interrupt flag in P1.3
}
```

Com este experimento, foi possível conhecer a operação das portas I/O no modo *Input* (entrada), configurando os diversos registradores necessários para a operação desta entrada como um botão, além de estudar os registradores de interrupção, suas configurações e desenvolvimento de código para tratamento de uma interrupção.

- **Módulo Oscilador:** os microcontroladores MSP430 possuem sistema de *clock* que permite a realização de operações internas e externas a partir de diferentes fontes. Conhecido como BCS (*Basic Clock System*), esse sistema possui osciladores de baixa/alta frequência (LFXT1), além de oscilador controlado digitalmente (DCO). Alguns modelos de MSP430 possuem ainda um terceiro oscilador também de alta frequência (XT2). Esses osciladores podem disponibilizar o sinal de *clock* de três diferentes formas: pelo *clock* principal (MCLK), utilizado normalmente no sincronismo da CPU e outros periféricos do sistema, pelo *clock* secundário (SMCLK), também utilizado para sincronizar periféricos alternativamente ao MCLK e, por fim, pelo *clock* auxiliar (ACLK), utilizado principalmente como fonte de *clock* de precisão para periféricos durante o modo de baixo consumo do microcontrolador. Todos estes sinais podem ser divididos por 1, 2 4 ou 8 vezes, de acordo com a necessidade, antes de serem disponibilizados para utilização, conforme é possível verificar na Figura 11 [14].

Figura 11 – Diagrama de blocos: Basic Clock Module



Fonte: Texas Instruments.

Nos MSP430G2552 presentes na *LaunchPad*, estão disponíveis para utilização os osciladores LFXT1 e DCO. Para o estudo do módulo oscilador, o estudante necessita elaborar experimentos onde seja possível verificar as diferentes configurações de sinais de *clock*, determinado fontes de oscilação, frequência de oscilação e forma de saída. Assim, propôs-se configurar um sinal de *clock* operando a partir do DCO, numa faixa de frequência de 0,06 - 0,14 MHz e disponibiliza-lo via ACLK com fator 2 de divisão, na porta P1.0 da *LaunchPad*. Em seguida, ao se pressionar o botão S2, a frequência de saída deve ser alterada para o fator de divisão 8, mantendo-se nessa condição enquanto o botão estiver pressionado e retornando à condição anterior ao liberar o botão S2. Adicionalmente, alternar o acendimento dos *leds* L3 e L4 da McLab2 com o acionamento do botão S2.

Incialmente, para se programar o modulo oscilador do MSP430, faz-se necessária a configuração dos registradores deste módulo, perfazendo um total de 4 registradores de 8 *bits* cada. São eles:

- ✓ DCOCTL: responsável pela configuração da frequência de operação do DCO, através dos *bits* DCOx e MODx.

✓ BCSCTL1: responsável pela seleção do oscilador XT2, definição do modo de operação de LFXT1, definição do fator de divisão do sinal ACLK e seleção do range de frequência do DCO, através dos *bits* XT2OFF, XTS, DIVAx e RSELx, respectivamente.

✓ BCSCTL2: responsável pela seleção do oscilador fonte de MCLK, definição do fator de divisão do sinal MCLK, seleção do oscilador fonte de SMCLK, definição do fator de divisão do sinal SMCLK e definição do resistor de DCO, através dos *bits* SELMx, DIVMx, SELS, DIVSx e DCOR, respectivamente.

✓ BCSCTL3: responsável pela definição da faixa de frequência de XT2, definição da faixa de frequência de LFXT1, seleção do capacitor do oscilador LFXT1, flag de falha de XT2 e flag de falha de LFXT1, através dos *bits* XT2Sx, LFXT1Sx, XCAPx, XT2OF e LFXT1OF, respectivamente [15].

De posse do *datasheet* do MSP430, faz-se então a programação das configurações dos registradores, conforme segue:

```
DCOCTL = 0x00;  // DCOx = 000 e MODx = 00000
BCSCTL1 = 0x90;  // XT2OFF = 1, XTS = 0, DIVAx = 01 e RSELx = 0000
BCSCTL2 = 0x00;  // SELMx = 00, DIVMx = 00, SELS = 0, DIVSx = 00, DCOR = 0
BCSCTL3 = 0x0c;  // XT2Sx = 00, LFXT1Sx = 00, XCAPx = 11, XT2OF = 0,
                    LFXT1OF = 0
```

Posteriormente, faz-se a configuração dos pinos I/O de acordo com o proposto no experimento, através dos registradores PxDIR, PxOUT, PxREN, definindo entradas e saídas, resistores de *pull-up/pull-down* e níveis lógicos nas saídas dos pinos. Além destes registradores, devido à necessidade de disponibilizar o sinal de *clock* ACLK na porta P1.0, é necessária a configuração do registrador PxSEL. Este registrador é responsável, em conjunto com o registrador PxSEL2, de selecionar funções especiais aos pinos de I/O, conforme configuração pré-definida no microcontrolador. No caso de P1.0, a saída ACLK é selecionada quando P1SEL está em nível alto no BIT0. Sendo assim, os registradores de I/O foram configurados da seguinte forma:

```
P1SEL = BIT0;                   // Define special function to P1.0 (ACLK)
P1DIR = BIT0+BIT5;              // Define P1.0 e P1.5 as outputs
P1REN = BIT3;                   // Enable pullup/pulldown of P1.3
```

```
P1OUT = BIT3+BIT5;          // Set pullup (high) for P1.3 e light on L3 (P1.5)
P2DIR = BIT0;               // Define P2.0 output
P2OUT = 0;                  // Set Port2 low
```

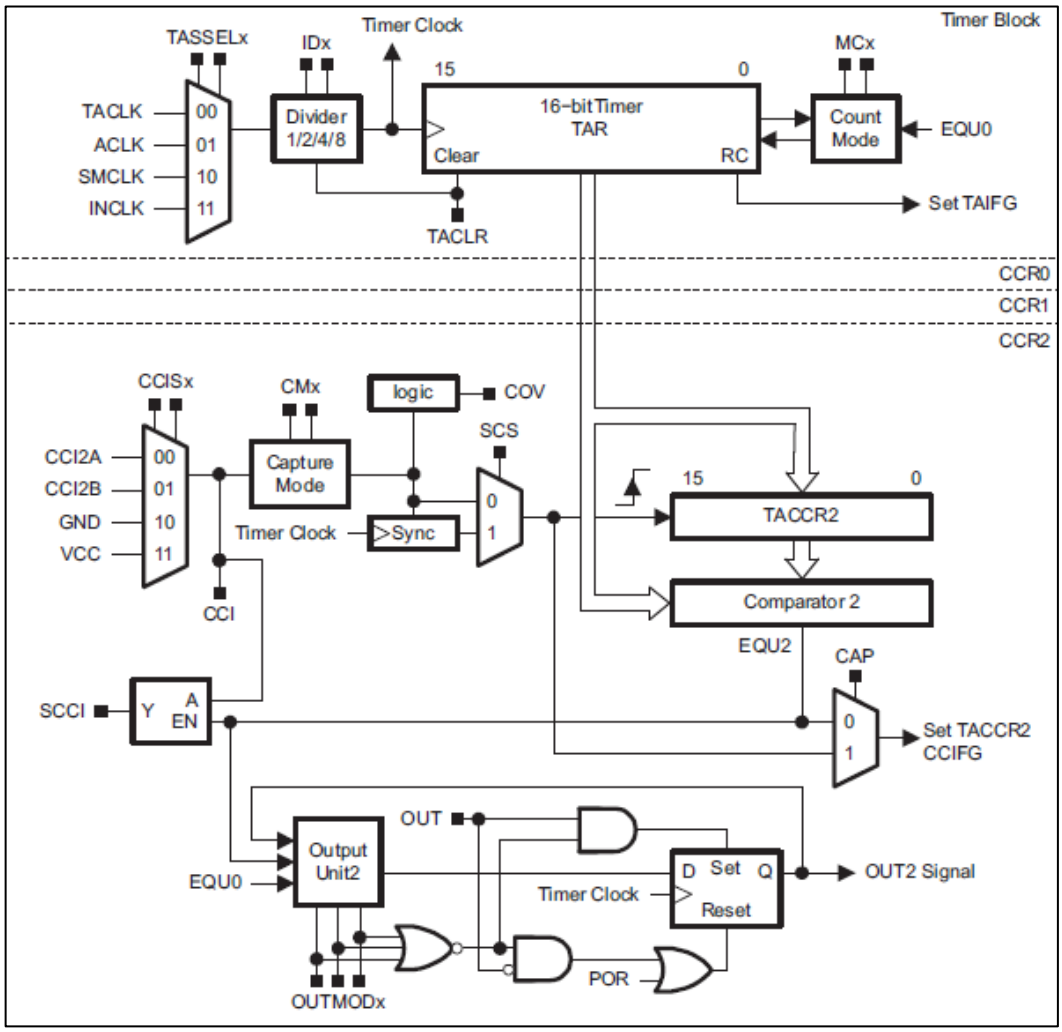
Por fim, programa-se a rotina de verificação do botão S2. Como os *bits* necessários à seleção do fator de divisão de ACLK encontram-se no registrador BCSCTL1, basta realizar a reconfiguração deste registrador para termos o sinal ACLK divido conforme proposto. Temos então a seguinte programação:

```
 if( (P1IN & BIT3) == 0 )    // Check if P1.3 is low (S2 pressed)
 {                           // If S2 pressed, ACLK frequency is divided by 8.
  BCSCTL1 = 0xb0;            // XT2OFF = 1, XTS = 0, DIVAx = 11 e RSELx = 0000
  P1OUT &= ~BIT5;            // Light off L3 (P1.5)
  P2OUT |= BIT0;             // Light on L4 (P2.0)
 }
 else {                      // If S2 not pressed, ACLK frequency divided by 2
  BCSCTL1 = 0x90;            // XT2OFF = 1, XTS = 0, DIVAx = 01e RSELx = 0000
  P1OUT |= BIT5;             // Light on L3 (P1.5)
  P2OUT &= ~BIT0;}           // Light off L4 (P2.0)
}
```

Através deste experimento, foi possível estudar o modulo oscilador do MSP430, conhecendo os registradores que compõem este módulo e a maneira com a qual estes registradores devem ser configurados de acordo com o tipo de oscilador que se deseja utilizar, bem como o sinal de *clock* que será disponibilizado a partir deste oscilador.

- **Timer A:** o MSP430G2553 possui um contador/temporizador de 16 *bits* programável, incremental ou decremental e com capacidade de interrupção, que pode operar a partir das diferentes fontes de *clock* fornecidas pelo microcontrolador e também a partir de *clock* externo. Possui até três registradores de CCP com funções de captura, comparação e PWM (*Pulse Width Modulation*), que podem ser utilizados para medição de períodos, contagem de eventos e geração de sinais PWM. Também pode realizar captura a partir de um sinal analógico, além da possibilidade de realizar conversão A/D a partir o comparador [16].

Figura 12 – Diagrama de blocos: Timer A



Fonte: Texas Instruments.

No MSP430G2553 presente na *LaunchPad*, este contador/temporizador é disponibilizado em dois blocos, o *Timer* A0 e o *Timer* A1. Para a análise do funcionamento do *Timer* A, o estudante precisa realizar experimentos onde sejam necessários realizar medições ou controles de potência de periféricos, configurando o *Timer* A para desempenhar tais funções. Por isso, elaborou-se experimento no qual um ventilador é acionado através de um sinal PWM gerado a partir do *Timer* A1, de tal maneira que a velocidade de rotação inicie em 25% da capacidade nominal e seja incrementada em 25% a cada acionamento do botão S2. Adicionalmente, é calculada a rotação do motor através de um tacômetro IR (*Infrared*) e apresentado o resultado no LCD.

Neste experimento, serão utilizados os circuitos de ventilador e tacômetro, além do *display* LCD. Para tanto, as conexões entre a McLab2 e o MSP430 seguem a configuração apresentada no Quadro 7 e obtida através da comutação dos *jumpers* da placa de interface:

Quadro 7 – Conexões LCD, ventilador e tacômetro

Instrução	**MSP430G2553**	
	Porta	**Pino**
RS do LCD alfanumérico	P1.5	7
ENABLE do LCD alfanumérico	P2.0	8
LCD Seg D0	GND / NC	20
LCD Seg D1	GND / NC	20
LCD Seg D2	GND / NC	20
LCD Seg D3	GND / NC	20
LCD Seg D4	P2.2	10
LCD Seg D5	P2.3	11
LCD Seg D6	P2.4	12
LCD Seg D7	P2.5	13
Ventilador	P2.1	9
Tacômetro	P1.4	6

**NC = Não Conectado*

Fonte: O autor.

Para a programação deste experimento, primeiramente é necessária a configuração do sinal de *clock* do sistema, uma vez que a operação dos *Timers* A0 e A1 ocorrerá a partir deste sinal. Assim faz-se necessária a programação dos registradores de *clock* do microcontrolador (DCOCTL, BCSCTL1, BCSCTL2, BCSCTL3), de modo a definir a frequência de operação do sistema.

```
DCOCTL = CALDCO_1MHZ;          // DCO = 1 MHz
BCSCTL1 = CALBC1_1MHZ;         // BC1 = 1 MHz
 BCSCTL2 = 0x00;               // 0
 BCSCTL3 = 0x00;               // 0
```

Em seguida, realiza-se a configuração dos pinos I/O, através dos registradores PxDIR, PxREN, PxOUT, PxSEL, PxSEL2, PxIE, PxIES e PxIFG

```
 P2DIR |= 0x02;          // Set P2.1 as output.
 P2SEL |= 0x02;          // Set P2.1 as PWM(TA1.1) output special function
 P2SEL2 = 0x00;          // Clear P2SEL2
```

```
P1REN |= 0X08;          // Enable Pull-up/Down Resistor in P1.3
P1OUT |= 0X08;          // Select pull-up mode in P1.3
P1IE  |= 0X18;          // Enable interrupts in P1.3 & P1.4
P1IES |= 0x08;          // Set Hi-Lo edge in P1.3
P1IES &= ~0x10;         // Set Lo-Hi edge in P1.4
P1IFG &= ~0X18;         // Clear interrupt flags in P1.3 & P1.4
```

Posteriormente, configura-se os registradores do *Timer* A0 e A1, de forma que o contador opere de acordo com a necessidade do programador. Neste experimento, o *Timer* A0 será utilizado como contador para a contagem de tempo e o *Timer* A1 será utilizado na geração do sinal PWM. Para tanto, devemos configurar os seguintes registradores:

- ✓ TACTL: responsável pela definição da fonte de *clock*, seleção do fator de divisão do sinal de *clock*, seleção do modo de operação do contador, definição de *reset* do contador, habilitação de interrupção e *flag* de interrupção, através dos *bits* TASSELx, IDx, MCx, TACLR, TAIE, TAIFG respectivamente.
- ✓ TACCTL: responsável por definir o modo de captura, definir o sinal de entrada do modo de captura, definir sincronia da entrada do modo comparação/captura, definir modo (captura/comparação), definir modo de saída, definir interrupção, definir entrada, definir nível da saída (alto/baixo), definir estouro do contador e setar *flag* de interrupção, através dos *bits*, CMx, CCISx, SCS, SCCI, CAP, OUTMODx, CCIE, CCI, OUT, COV e CCIFG respectivamente.
- ✓ TACCRx: responsável por definir o valor de referência do registrador de captura comparação, através dos *bits* TACCRx [17].

Desta maneira, procedeu-se com a configuração dos registradores da seguinte maneira:

```
// Timer A0
TA0CTL = TASSEL_2 + MC_1 + ID_0; // Set source SMCLK, control "Up to CCR0"
                                    e input divider "/1"
TA0CCTL0 |= CCIE;                // Enable Timer A0 interrupt
TA0CCR0 = 1000-1;                // Set cycle period 1 ms (frequency 1 KHz)
```

```
// Timer A1:
TA1CTL = TASSEL_2 + MC_1 + ID_0;  // Set source SMCLK, control "Up to CCR0"
                                     e input divider "/1")
TA1CCTL1 = OUTMOD_7;              // PWM2 output mode: 7 - PWM reset/set
TA1CCR0 = 2000-1;                 // Set cycle period 2 ms (frequency 500 Hz)
TA1CCR1 = 500-1;                  // Set PWM2 duty cycle in 25%
```

Conforme os registradores foram configurados, temos no *Timer* A0 um temporizador de contagem de tempo, na qual é disparado uma interrupção a cada 1 milissegundos (ms). Já o *Timer* A1 utiliza a comparação entre os registradores de captura TA1CCR0 e TA1CCR1 para gerar um pulso de PWM com 25% de *duty cycle*. Por fim, realizam-se os tratamentos das interrupções: no *Timer* A0 é feita a contagem de tempo em milissegundos e no botão S2 é definido um novo valor de TA1CCR1 de forma que o pulso PWM seja incrementado em 25% a cada acionamento do botão. Já a interrupção da entrada do tacômetro (P1.4) é utilizada para a contagem das voltas do ventilador e posterior cálculo da velocidade em RPM (Rotações por Minuto):

```
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void){
   miliseconds++;}                // Increment time counter

#pragma vector = PORT1_VECTOR
__interrupt void Port_1(void) {
  if(P1IFG&0x08) {                // Is there P1.4 interrupt pending?
    if((P1IES&0x08)) {            // Is this the Hi-Lo edge?
      if (TA1CCR1 == (2000-1)){
        TA1CCR1 = (500-1); }      // Returns PWM to duty cycle 25%
      else {
        TA1CCR1 += 500;           // Add 25% to PWM duty cycle
        __delay_cycles (50000); }
    }
  }
  if(P1IFG&0x10) {              // Is there P1.3 interrupt pending?
```

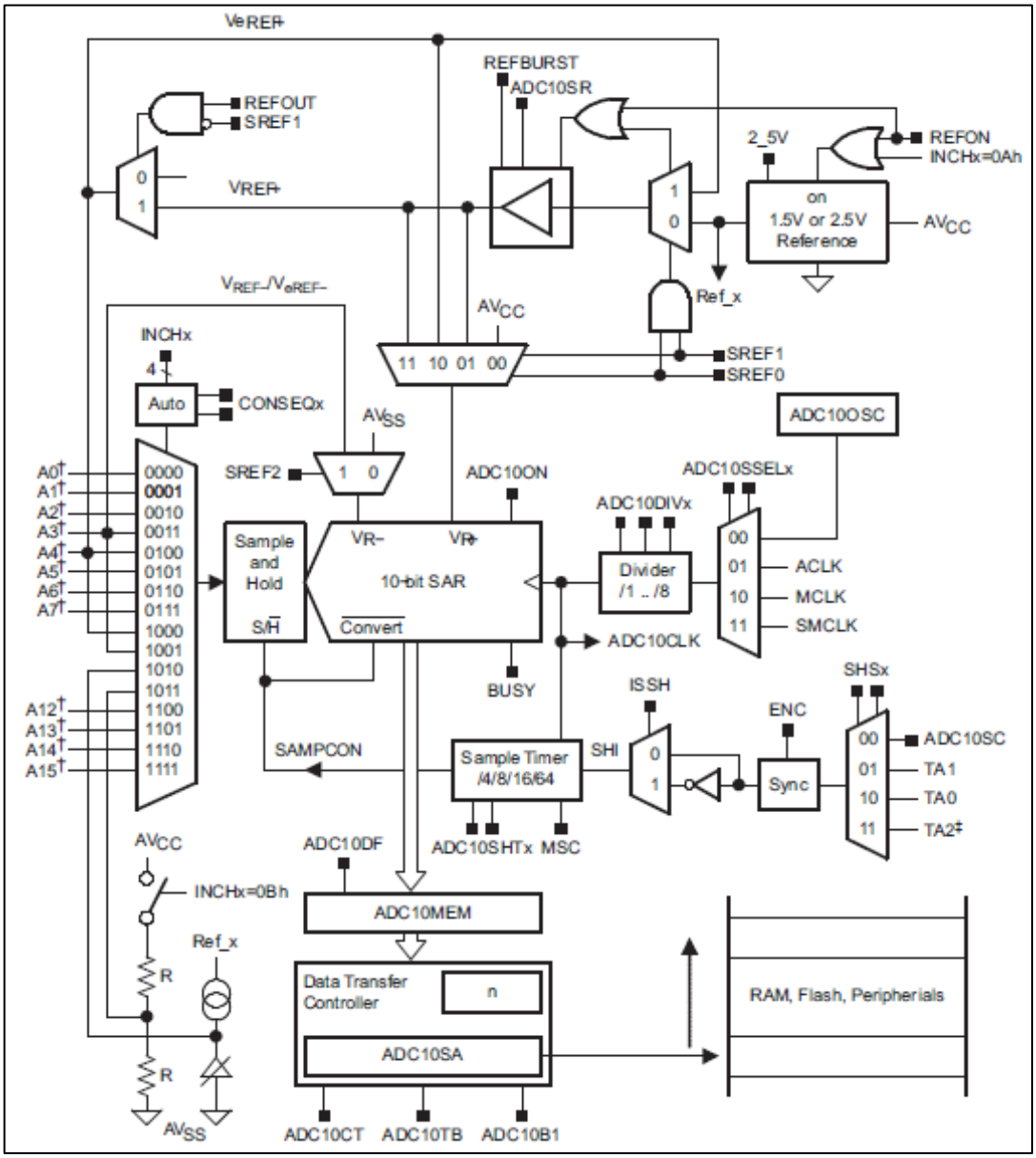
```
  if(!(P1IES&0x10)) {            // Is this the Lo-Hi edge?
    if (turn_counter < 179){
      turn_counter++;}           // Increment turn_counter
    else {                       // If turn_counter is 180, the fan has completed 20 turns
      rpm=1200000/miliseconds  // RPM = 20(turns) x 60000(ms = 1min) / miliseconds
      turn_counter=0;            // Reset turn counter
      TA0CTL|=TACLR;             // Clears timer A0
      miliseconds = 0; }         // Reset time counter
   }
  }
  P1IFG &= ~0x18;                // Clear interrupt flags in P1.3 & P1.4
}
```

Com este experimento, foi possível compreender o funcionamento do *Timer* A e sua aplicabilidade prática no controle de potência pelo sinal de PWM, além da sua utilização como contador/temporizador. Foi possível verificar os registradores necessários à configuração e compreender a função de cada um de seus *bits*.

- **Conversor Analógico/Digital (A/D):** o MSP430G2553 possui um conversor analógico/digital de 10 *bits* (ADC10) de resolução, o que permite a leitura de 1024 níveis analógicos discretos de um sinal. Possui período de amostragem e fontes de *clock* programável por *software*. Além disso, pode iniciar uma conversão por uma saída do *Timer* A.

O conversor ADC10 pode operar com tensão de referencia interna ou externa, sendo que internamente esta tensão é selecionável entre 1,5V e 2,5V. Além disso, possui um controlador de transferência de dados que permite a transmissão automática das informações para a memória do microcontrolador, evitando a interferência da CPU [18].

Figura 13 – Diagrama de blocos: ADC10



Fonte: Texas Instruments.

Para o estudo do ADC10, é fundamental que o estudante realize experimentos onde sejam necessárias a leitura de níveis analógicos de tensão pelo microcontrolador como, por exemplo, no caso de sensores de temperatura, de forma que este sinal analógico deva ser convertido em sinal digital para tratamento posterior. Dessa forma, propôs-se a programação de um medidor de tensão, a partir do circuito existente na McLab2 onde é possível variar a tensão de saída através do ajuste de um potenciômetro. Dessa forma, o programa deverá realizar seguidas leituras de tensão e disponibilizá-las para visualização no LCD, para que se possa verificar a variação de tensão proveniente da alteração na resistência. Neste

experimento, será necessária a utilização do circuito medidor de tensão da placa didática, além do LCD. As conexões com a *LaunchPad* devem ser realizadas conforme Quadro 8, por intermédio dos *jumpers* da placa de interface.

Quadro 8 – Conexões LCD e medidor de tensão

Instrução	**MSP430G2553**	
	Porta	**Pino**
RS do LCD alfanumérico	P1.5	7
ENABLE do LCD alfanumérico	P2.0	8
LCD Seg D0	GND / NC	20
LCD Seg D1	GND / NC	20
LCD Seg D2	GND / NC	20
LCD Seg D3	GND / NC	20
LCD Seg D4	P2.2	10
LCD Seg D5	P2.3	11
LCD Seg D6	P2.4	12
LCD Seg D7	P2.5	13
Medidor de Tensão	P1.7	15

**NC = Não Conectado*

Fonte: O autor.

Assim como o *Timer* A, a configuração do ADC10 deve ser iniciada pela configuração do sinal de *clock* do sistema, uma vez que esta será fonte de operação do conversor A/D:

```
DCOCTL = CALDCO_1MHZ;          // DCO = 1 MHz
BCSCTL1 = CALBC1_1MHZ;         // BC1 = 1 MHz
BCSCTL2 = 0x00;                // 0
BCSCTL3 = 0x00;                // 0
```

Na sequência, realiza-se a configuração dos registradores do conversor A/D e do pino de entrada do conversor. É importante notar que, no caso ADC10, basta o pino de entrada estar habilitado no registrador ADC10AEx, que o mesmo já assume a função especial de entrada analógica, sem necessidade de configuração dos registradores PxSEL e PxSEL2.

Para o funcionamento do conversor, é necessário que seus registradores estejam corretamente configurados, sendo eles:

- ✓ ADC10AE0: responsável pela definição da entrada analógica do registrador 0, através dos *bits* ADC10AE0x.

✓ ADC10AE1: responsável pela definição da entrada analógica do registrador 1, através dos *bits* ADC10AE1x.

✓ ADC10CTL0: responsável pela definição da referência do conversor, definição do período de amostragem, definição da taxa de amostragem, definição da referência de saída, definição da referência de *buffer*, definição de múltiplas amostras e conversão, definição do nível da referência interna (1,5 ou 2,5V), definição da referência interna (on/off), acionamento do ADC10, habilitação da interrupção, *flag* de interrupção, habilitação da conversão e habilitação do início da conversão, através do *bits* SREFx, ADC10SHTx, ADC10SR, REFOUT, REFBURST, MSC, REF2_5, REFON ADC10ON, ADC10IE, ADC10IFG, ENC e ADC10 SC respectivamente.

✓ ADC10CTL1: responsável pela definição do canal de entrada do conversor, definição da fonte de período de amostragem, definição do formato do dado, definição do sinal da amostra (invertido ou não), definição do fator de divisão do *clock*, definição da fonte de *clock*, definição da sequencia de conversão e *busy flag*, através dos *bits* INCHx, SHSx, ADC10DF, ISSH, ADC10DIVx, ADC10SSELx, CONSEQx, ADC10BUSY.

✓ ADC10MEM: responsável por armazenar o resultado da conversão.

✓ ADC10DTC0: responsável pela definição de modo de dois blocos, definição do modo de transferência contínua, definição do bloco preenchido e definição de *reset*, através dos *bits* ADC10TB, ADC10CT, ADC10B1, ADC10FETCH.

✓ ADC10DCT1: responsável pela definição do numero de transferência por bloco, através dos *bits* DTC Transfers.

✓ ADC10SA: responsável por iniciar o registro de endereço para a transferência de dados [19].

No experimento proposto, os registradores devem ser configurados da seguinte forma:

```
ADC10AE0 |= BIT7                  // Config. P1.7 as intput of ADC10
ADC10CTL1 |= INCH_7               // Config. channel 7 of ADC
          + SHS_0                 // Config. sample-and-hold by software
          + ADC10DIV_7            // Config. division /8
          + ADC10SSEL_3           // Config. cock source SMCLK
          + CONSEQ_0;             // Config. ADC conversion
```

```
ADC10CTL0 |= SREF_0              // Config. Vcc = Vr+ e GND = Vr-
          + ADC10SHT_0           // Config. sample-and-hold time 4 clock cycles
          + ADC10ON              // Activate conversor
          + ADC10IE;             // Enable ADC10 interrupt
```

Por fim, se configura o registrador ADC10CTL0 para iniciar a conversão, faz-se o tratamento dos dados dentro da interrupção que é gerada ao ser realizada a amostragem e conversão, reprogramando um novo ciclo de amostragem, de tal maneira que a conversão seja contínua enquanto o microcontrolador estiver em funcionamento.

Início da conversão:

```
ADC10CTL0 |= ENC + ADC10SC;     // Start conversion
```

Rotina de tratamento da interrupção:

```
#pragma vector=ADC10_VECTOR        // Interrupt vector
__interrupt void ADC10_ISR(void) {
  ADC_value = ADC10MEM;            // Data from register ADC10MEM to variable
  ADC_value = (5*ADC_value)/1024;  // Calculate voltage (máx resol. 5V = 0x03FF)
  __bis_SR_register(GIE);          // Enable global interrupt
 ADC10CTL0 |= ENC + ADC10SC; }     // Start new conversion
```

Através da realização deste experimento, pode-se compreender o processo de conversão A/D realizada pelo microcontrolador, através das definições de período de amostragem, taxa de amostragem, modo de amostragem, além das configurações de canal de entrada A/D e armazenamento das conversões, de acordo com a configuração de seus respectivos registradores.

CONCLUSÃO

Após realização de todas as atividades propostas neste trabalho, conclui-se que os resultados obtidos foram satisfatórios, uma vez que com placa de interface, foi possível utilizar a *LaunchPad* com a McLab2. Contudo, devido a grande diferença na quantidade de pinos do PIC16F877A utilizado originalmente na McLab2 e o MSP430G2553 utilizado na *LaunchPad*, 40 contra 20, houve a necessidade de compartilhamento dos pinos do MSP430 em diferentes circuitos da placa McLab2, com a comutação entre os circuitos sendo realizada por intermédio de *jumpers*, o que resultou por limitar os circuitos que puderam ser conectados ao MSP430, além da impossibilidade de utilização de todos os circuitos de forma simultânea. Adicionalmente, houve a necessidade de conversão nos níveis de tensão para as portas que deveriam trabalhar como entrada no MSP430, de forma a garantir o correto funcionamento do microcontrolador, além de evitar que este seja danificado por sobretensão.

A partir do desenvolvimento da placa de interface entre a *LaunchPad* MSP-EXP430G2 e a placa de desenvolvimento McLab2, tornou-se possível explorar com muito mais profundidade as características e funções existentes no MSP430, haja vista que os experimentos realizados apresentam aplicações reais das funções presentes no microcontrolador e contribuem para a compreensão destas características. A elaboração dos experimentos contidos na McLab2 através da placa de interface ocorre de forma muito rápida, bastando apenas realizar o ajuste dos *jumpers* e prosseguir com a programação do microcontrolador, evitando o despendimento de tempo para a preparação de circuitos e trazendo o foco que o estudante necessita para a compreensão do componente.

Também por conta das práticas, cria-se a necessidade de contato constante com a linguagem de programação C, contribuindo com o desenvolvimento da capacidade de raciocínio lógico no desenvolvimento de *softwares* destinados ao controle do microcontrolador. Além disso, com a interface entre a *LaunchPad* e McLab2 o estudante pode depurar a programação em tempo real, reduzindo o tempo de análise de erros ou falhas e aumentando a eficiência do aprendizado.

Por fim, conclui-se que o projeto da placa de interface pode ser aperfeiçoado no sentido de se incluir os circuitos de ressonador 4 MHz, comunicação *serial* RS232 e a memória serial EEPROM, aumentando assim a gama de experimentos possíveis de serem realizados com a interface entre *LaunchPad* e McLab2.

REFERENCIAS

[1] TEXAS INSTRUMENTS. **Category: MSP430.** Disponível em: <http://processors.wiki.ti.com/index.php/Category:MSP430?keyMatch=MSP430&tisearch=Search-EN-Everything>. Acesso em: 23 ago. 2016.

[2] MOSAICO. **Guia do Usuário Placa de Desenvolvimento McLab2.** Disponível em: <http://www.mosaico.com.br/Midias/Documentacao/Manual%20McLab2%20(16F877A)_rev_07.pdf>. Acesso em: 10 jun. 2017.

[3] TEXAS INSTRUMENTS. Features. In: TEXAS INSTRUMENTS. **Mixed Signal Microcontroller**, p. 1. Disponível em: <http://www.ti.com/lit/ds/symlink/msp430g2253.pdf>. Acesso em: 10 jun.2017.

[4] MOSAICO. **Placa Lab – Modulo II.** Disponível em: <https://pt.scribd.com/document/349411630/Esquema-Eletrico-McLab2-pdf>. Acesso em: 10 jun. 2017.

[5] TEXAS INSTRUMENTS. Schematics. In: TEXAS INSTRUMENTS. **MSP-EXP430G2 LaunchPad™ Development Kit User's Guide**, p. 15-20. Disponível em: <http://www.ti.com/lit/ug/slau318g/slau318g.pdf>. Acesso em: 10 jun.2017.

[6] TEXAS INSTRUMENTS. Schmitt-Trigger Inputs, Ports Px. In: TEXAS INSTRUMENTS. **Mixed Signal Microcontroller**, p. 24. Disponível em: <http://www.ti.com/lit/ds/symlink/msp430g2253.pdf>. Acesso em: 10 jun.2017.

[7] TEXAS INSTRUMENTS. Output Interfaces. In: TEXAS INSTRUMENTS. **Interfacing the 3-V MSP430 to 5-V Circuits**, p. 10. Disponível em: < http://www.ti.com/lit/an/slaa148/slaa148.pdf>. Acesso em: 10 jun.2017.

[8] TEXAS INSTRUMENTS. Connecting a Crystal Oscillator. In: TEXAS INSTRUMENTS. **MSP-EXP430G2 LaunchPad™ Development Kit User's Guide**, p. 12. Disponível em: <http://www.ti.com/lit/ug/slau318g/slau318g.pdf>. Acesso em: 10 jun.2017.

[9] TEXAS INSTRUMENTS. Table 1. Available Options. In: TEXAS INSTRUMENTS. **Mixed Signal Microcontroler**, p. 2. Disponível em: <http://www.ti.com/lit/ds/symlink/msp430g2253.pdf>. Acesso em: 10 jun.2017.

[10] PEREIRA, FÁBIO. Sistema de Interrupções. In: PEREIRA, FÁBIO. **Microcontroladores MSP430 – Teoria e Prática.** São Paulo: Érica, 2005. p. 107-115.

[11] SUNROM. **LCD Module Specification JHD162A**. Disponível em: <www.sunrom.com/get/526000>. Acesso em: 10 jun.2017.

[12] MACIEL, MARCELO. **Conectando um display LCD no PIC**. Disponível em: <http://www.marcelomaciel.com/2012/03/configuracao-display-lcd-no-pic.html>. Acesso em: 10 jun.2017.

[13] GRUPOS DO GOOGLE. **MSP430 - Convertendo Inteiro para Char**. Disponível em: <https://groups.google.com/forum/#!topic/sis_embarcados/oRIM3VFz23o>. Acesso em: 12 jun. 2017.

[14] PEREIRA, FÁBIO. Módulo Oscilador. In: PEREIRA, FÁBIO. **Microcontroladores MSP430 – Teoria e Prática.** São Paulo: Érica, 2005. p. 116-137.

[15] TEXAS INSTRUMENTS. Basic Clock Module+ Registers. In: TEXAS INSTRUMENTS. **MSP430x2xx Family – User's Guide**, p. 282-285. Disponível em: <http://www.ti.com/lit/ug/slau144j/slau144j.pdf>. Acesso em: 10 jun.2017.

[16] PEREIRA, FÁBIO. Timer A. In: PEREIRA, FÁBIO. **Microcontroladores MSP430 – Teoria e Prática.** São Paulo: Érica, 2005. p. 143-156.

[17] TEXAS INSTRUMENTS. Timer_A Registers. In: TEXAS INSTRUMENTS. **MSP430x2xx Family – User's Guide**, p. 369-373. Disponível em: <http://www.ti.com/lit/ug/slau144j/slau144j.pdf>. Acesso em: 10 jun.2017.

[18] PEREIRA, FÁBIO. Conversor A/D de 10 Bits. In: PEREIRA, FÁBIO. **Microcontroladores MSP430 – Teoria e Prática.** São Paulo: Érica, 2005. p. 239-250.

[19] TEXAS INSTRUMENTS. ADC10 Registers. In: TEXAS INSTRUMENTS. **MSP430x2xx Family – User's Guide**, p. 552-558. Disponível em: <http://www.ti.com/lit/ug/slau144j/slau144j.pdf>. Acesso em: 10 jun.2017.

APÊNDICE A

Programa em linguagem C: Contador 0 a 9999.

```
#include "msp430g2553.h"

// Define Port1 and Port2 outputs for 0-9 sequence {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

int NumP1[10] = { 0x0f,       0x06, 0x0b, 0x0f, 0x06,0x0d, 0x0d, 0x07, 0x0f, 0x0f};
int NumP2[10] = { 0x0c, 0x00, 0x14, 0x10, 0x18, 0x18, 0x1c, 0x00, 0x1c, 0x18};

// Define Port1 and Port2 outputs for mcdu (milhar, centena, dezena, unidade)  displays

int DplP1[4] = {0x20, 0x00, 0x00, 0x40};
int DplP2[4] = {0x00, 0x01, 0x02, 0x00};

// Matrix for Port1 and Port2 mcdu data

int mcduP1 [4];
int mcduP2 [4];

int main(void){
  WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog to prevent time out reset

  P1DIR = 0x6f;                   // Config. P1.0, P1.1, P1.2, P1.3, P1.5 & P1.6 as outputs
  P2DIR = 0x3f;                   // Config. P2.0, P2.1, P2.2, P2.3, P2.4 & P2.5 as outputs
  P1OUT = 0;                      // Set Port1 outputs low (0)
  P2OUT = 0;                      // Set Port2 outputs low (0)

  int i, j, k, l;                 // Variables for mcdu sequence (l=m, k=c, j=d, i=u)
  volatile unsigned int t, w, x;  // Auxiliary variables

// Start  counting to 0000
  mcduP1[0] = NumP1[0];
  mcduP1[1] = NumP1[0];
  mcduP1[2] = NumP1[0];
  mcduP1[3] = NumP1[0];

  mcduP2[0] = NumP2[0];
  mcduP2[1] = NumP2[0];
  mcduP2[2] = NumP2[0];
  mcduP2[3] = NumP2[0];
  while (1){
```

```
   for (l=0; l<10;l++){                    // Thousand count loop

    for (k=0; k<10; k++){                  // Hundred count loop

     for (j=0; j<10; j++){                 // Ten count loop

      for (i=0; i<10; i++){                // Unit count loop

       mcduP1[0] = NumP1[i];               // P1 unit counting
       mcduP2[0] = NumP2[i];               // P2 unit counting

       for (t=0; t<155; t++){              // Counting delay
        for (x=0; x<4; x++){               // Multiplexing loop
         P1OUT = mcduP1[x]+DplP1[x];       // P1 sync between mcdu display and value
         P2OUT = mcduP2[x]+DplP2[x];       // P2 sync between mcdu display and value
         for(w=0;w<150;w++);               // Multiplexing delay
        }
       }
      }                                    // End unit count loop

      mcduP1[0] = NumP1[0];
      mcduP1[1] = NumP1[j+1];
      mcduP2[0] = NumP2[0];
      mcduP2[1] = NumP2[j+1];
     }                                     // End ten count loop

     mcduP1[1] = NumP1[0];
     mcduP1[2] = NumP1[k+1];
     mcduP2[1] = NumP2[0];
     mcduP2[2] = NumP2[k+1];
    }                                      // End hundred count loop

    mcduP1[2] = NumP1[0];
    mcduP1[3] = NumP1[l+1];
    mcduP2[2] = NumP2[0];
    mcduP2[3] = NumP2[l+1] | 0x20;         // Add thousand separator point (0x20)
   }                                       // End thousand count loop

   mcduP1[3] = NumP1[0];
   mcduP2[3] = NumP2[0];
  }                                        // End WHILE loop
}
```

APÊNDICE B

Programa em linguagem C: Comunicação do LCD e varredura de botão por interrupção.

```
#include <msp430g2553.h>

// LCD control pins definitions
#define DATA_REG P1OUT = BIT5                   // set RS high (data register)
#define INST_REG P1OUT = (~BIT5)                // set RS low (instruction register)
#define ENABLE_PIN_HIGH P2OUT |= BIT0           // set Enable high
#define ENABLE_PIN_LOW P2OUT &= (~BIT0)   // set Enable Low

// Variable declaration

int counter = 0;
char char_counter[5];
```

// Implementation of itoa function (convert values in strings to show on LCD)

```
char *itoa(int from, char to[])
{
   char const digit[]  = "0123456789";
   char* p             = to;
   int shifter;

   if (from < 0) {
      *p++ = '-';
      from *= -1;
   }

   shifter = from;

   do {
      ++p;
      shifter = shifter / 10;
   } while (shifter);

   *p = '\0';

   do {
      *--p = digit[(from % 10)];
      from = from / 10;
   } while (from);

   return to;
}
void configure_clocks()
```

```
{
  WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog to prevent time out reset
  DCOCTL = CALDCO_1MHZ;           // DCO = 1 MHz
  BCSCTL1 = CALBC1_1MHZ;          // BC1 = 1 MHz
  BCSCTL2 = 0x00;                 // 0
  BCSCTL3 = 0x00;                 // 0
}

void delay_us(unsigned int us)
{
  while (us)
  {
    __delay_cycles(1);            // Frequency 1 MHz = Period 1 us
    us--;
  }
}

void delay_ms(unsigned int ms)
{
  while (ms)
  {
    __delay_cycles(1000);       // Frequency 1 MHz = Period 1 us  =>  (1ms = 1000 us)
    ms--;
  }
}

void data_write(void)
{
  ENABLE_PIN_HIGH;
  delay_ms(5);
  ENABLE_PIN_LOW;
}

void send_data(unsigned char data)
{
  unsigned char higher_nibble = 0x3c & (data >> 2);
  unsigned char lower_nibble = 0x3c & (data << 2);

  delay_us(200);

  DATA_REG;

  P2OUT = (P2OUT & 0xc3) | (higher_nibble);       // Send first 4 bits
  data_write();
  P2OUT = (P2OUT & 0xc3) | (lower_nibble);        // Send last 4 bits
  data_write();
}


void send_string(char *s)
```

```
{
  while(*s)
  {
    send_data(*s);
    s++;
  }
}

void send_command(unsigned char cmd)
{
  unsigned char higher_nibble = 0x3C & (cmd >> 2);
  unsigned char lower_nibble = 0x3C & (cmd << 2);

  INST_REG;
  P2OUT = (P2OUT & 0xC3) | (higher_nibble);      // Send first 4 bits
  data_write();

  P2OUT = (P2OUT & 0xC3) | (lower_nibble);       // Send last 4 bits
  data_write();
}

void lcd_init(void)
{
  P1DIR = 0x20;                          // Set P1.5 as output
  P1OUT = 0x00;                          // Set P1 outputs low
  P2DIR = 0x3D;                          // Set P2.0 P2.2 P2.3 P2.4 P2.5 as output
  P2OUT = 0x00;                          // Set P2 outputs low

  delay_ms(15);
  send_command(0x33);                    // Initialization code

  delay_us(200);
  send_command(0x32);                    // Initialization code

  delay_us(40);
  send_command(0x28);                    // 2 lines and 5×7 matrix (4-bit mode)

  delay_us(40);
  send_command(0x0E);                    // Display on, cursor on

  delay_us(40);
  send_command(0x01);                    // Clear display screen

  delay_us(40);
  send_command(0x06);                    // Increment cursor (shift cursor to right)

  delay_us(40);
  send_command(0x80);                    // Set cursor to beginning to 1st line
}
```

```
int main(void)
{
  configure_clocks();                 // Start clock

  lcd_init();                         // Start LCD

  send_string("Julio's TCC 2017");    // Write on 1st line
  send_command(0xC0);                 // Set cursor to beginning to 2nd line
  send_string("Hello World!!!");      // Write on 2nd line
  send_command(0x0C);                 // Display on, cursor off

  P1REN |= BIT3;                      // Enable Pull-up/Down Resistor in P1.3
  P1OUT |= BIT3;                      // Select pull-up mode in P1.3
  P1IE  = BIT3;                       // Enable interrupt in P1.3
  P1IES = BIT3;                       // Hi-Lo edge in P1.3
  P1IFG &= ~BIT3;                     // Clear interrupt flag in P1.3

  __bis_SR_register (GIE);            // Enable global interrupt

  while (1)
  {}
}

#pragma vector = PORT1_VECTOR
__interrupt void Port_1(void){
  counter++;                          // Incremental counter of button pressed
  itoa(counter, char_counter);        // Convert integer count to char char_count
  send_command(0x01);                 // Clear display screen
  send_command(0x80);                 // Set cursor to beginning to 1st line
  send_string("Button counter:");     // Write on 1st line
  send_command(0xC0);                 // Set cursor to beginning to 2nd line
  send_string("Pressed: ");           // Write on 2nd line
  send_string(char_counter);          // Write char_counter on 2nd line
  send_string(" x");                  // Write on 2nd line
  send_command(0x0C);                 // Display on, cursor off
  __delay_cycles(50000);              // Delay
  P1IFG &= ~BIT3;                     // Clear interrupt flag in P1.3
}
```

APÊNDICE C

Programa em linguagem C: Configuração do modulo oscilador e acendimento de led.

```
#include "msp430g2553.h"

int main( void ){
WDTCTL = WDTPW + WDTHOLD;     // Stop watchdog to prevent time out reset

// Set DCOCTL, BCCTL1, BCCTL2, BCCTL3 to produce an oscillation by DCO
// Frequency range 0,06 - 0,14 MHz - Registers settings:

  DCOCTL = 0x00;    // DCOx = 0(dec) e MODx = 0(dec)
  BCSCTL1 = 0x90;   // XT2OFF = 1(bin), XTS = 0(bin), DIVAx = 01(bin) (ACLK /2) e
                                RSELx = 0(dec)
  BCSCTL2 = 0x00;   // SELMx = 00(bin), DIVMx = 00(bin), SELS = 0(bin),
                                DIVSx = 00(bin), DCOR = 0(bin)
  BCSCTL3 = 0x0c;   // XT2Sx = 00(bin), LFXT1Sx = 00(bin), XCAPx = 11(bin),
                                XT2OF = 0(bin), LFXT1OF = 0(bin)

  P1SEL = BIT0;                     // Define special function to P1.0 (ACLK)
  P1DIR = BIT0 +BIT5;               // Define P1.0 e P1.5 as outputs
  P1REN = BIT3;                     // Enable pullup/pulldown of P1.3
  P1OUT = BIT3+BIT5;                // Set pullup (high) for P1.3 e light on L3 (P1.5)
  P2DIR = BIT0;                     // Define P2.0 output
  P2OUT = 0;                        // Set Port2 low

  while(1) {
    if( (P1IN & BIT3) == 0 )        // Check if P1.3 is low (S2 pressed)
    {                               // If S2 pressed, ACLK frequency is divided by 8.
      BCSCTL1 = 0xb0;               // XT2OFF = 1(bin), XTS = 0(bin), DIVAx = 11(bin)
                                       (ACLK /8) e RSELx = 0(dec)
      P1OUT &= ~BIT5;               // Light off L3 (P1.5)
      P2OUT |= BIT0;                // Light on L4 (P2.0)
    }
    else {                      // If S2 not pressed, ACLK frequency divided by 2
      BCSCTL1 = 0x90;   // XT2OFF = 1(bin), XTS = 0(bin), DIVAx = 01(bin) (ACLK /2) e
                                RSELx = 0(dec)
      P1OUT |= BIT5;        // Light on L3 (P1.5)
      P2OUT &= ~BIT0;} // Light off L4 (P2.0)
  }
}
```

APÊNDICE D

Programa em linguagem C: Controle de velocidade da ventilador via PWM e medição de velocidade com tacômetro IR.

```
#include <msp430g2553.h>

// LCD control pins definitions
#define DATA_REG P1OUT = BIT5                   // set RS high (data register)
#define INST_REG P1OUT = (~BIT5)                // set RS low (instruction register)
#define ENABLE_PIN_HIGH P2OUT |= BIT0           // set Enable high
#define ENABLE_PIN_LOW P2OUT &= (~BIT0)         // set Enable Low

// Variable declaration
int turn_counter = 0;
int miliseconds = 0;
int rpm = 0;
char char_rpm[5];
char char_miliseconds[5];

// Implementation of itoa function (convert values in strings to show on LCD)
char *itoa(int from, char to[])
{
   char const digit[]  = "0123456789";
   char* p             = to;
   int shifter;

   if (from < 0) {
      *p++ = '-';
      from *= -1;
   }

   shifter = from;

   do {
      ++p;
      shifter = shifter / 10;
   } while (shifter);

   *p = '\0';

   do {
      *--p = digit[(from % 10)];
      from = from / 10;
   } while (from);

   return to;
}
```

```
void configure_clocks()
{
  WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog to prevent time out reset
  DCOCTL = CALDCO_1MHZ;           // DCO = 1 MHz
  BCSCTL1 = CALBC1_1MHZ;          // BC1 = 1 MHz
  BCSCTL2 = 0x00;                 // 0
  BCSCTL3 = 0x00;                 // 0
}

void delay_us(unsigned int us)
{
  while (us)
  {
    __delay_cycles(1);            // Frequency 1 MHz = Period 1 us
    us--;
  }
}

void delay_ms(unsigned int ms)
{
  while (ms)
  {
    __delay_cycles(1000);       // Frequency 1 MHz = Period 1 us  =>  (1ms = 1000 us)
    ms--;
  }
}

void data_write(void)
{
  ENABLE_PIN_HIGH;
  delay_ms(5);
  ENABLE_PIN_LOW;
}

void send_data(unsigned char data)
{
  unsigned char higher_nibble = 0x3c & (data >> 2);
  unsigned char lower_nibble = 0x3c & (data << 2);

  delay_us(200);

  DATA_REG;
  P2OUT = (P2OUT & 0xc3) | (higher_nibble);       // Send first 4 bits
  data_write();
  P2OUT = (P2OUT & 0xc3) | (lower_nibble);        // Send last 4 bits
  data_write();
}

void send_string(char *s)
{
```

```
  while(*s)
  {
    send_data(*s);
    s++;
  }
}

void send_command(unsigned char cmd)
{
  unsigned char higher_nibble = 0x3C & (cmd >> 2);
  unsigned char lower_nibble = 0x3C & (cmd << 2);

  INST_REG;

  P2OUT = (P2OUT & 0xC3) | (higher_nibble);      // Send first 4 bits
  data_write();

  P2OUT = (P2OUT & 0xC3) | (lower_nibble);       // Send last 4 bits
  data_write();
}

void lcd_init(void)
{
  P1DIR = 0x20;                          // Set P1.5 as output
  P1OUT = 0x00;                          // Set P1 outputs low
  P2DIR = 0x3D;                          // Set P2.0 P2.2 P2.3 P2.4 P2.5 as output
  P2OUT = 0x00;                          // Set P2 outputs low

  delay_ms(15);
  send_command(0x33);                    // Initialization code

  delay_us(200);
  send_command(0x32);                    // Initialization code

  delay_us(40);
  send_command(0x28);                    // 2 lines and 5×7 matrix (4-bit mode)

  delay_us(40);
  send_command(0x0E);                    // Display on, cursor on

  delay_us(40);
  send_command(0x01);                    // Clear display screen

  delay_us(40);
  send_command(0x06);                    // Increment cursor (shift cursor to right)

  delay_us(40);
  send_command(0x80);                    // Set cursor to beginning to 1st line
}
```

```
void calc_rpm (void){
  rpm=1200000/miliseconds;              // RPM = 20(turns) x 60000(ms = 1min) / miliseconds
  itoa(miliseconds, char_miliseconds);  // Convert int miliseconds to char_miliseconds
  itoa(rpm, char_rpm);                  // Convert int rpms to char char_rpm
  send_command(0x89);                   // Set cursor to 2nd half of 1st line
  send_string("ms=");                   // Write on LCD
  send_string(char_miliseconds);        // Write char_miliseconds on LCD
  send_command(0xC0);                   // Set cursor to beginning to 2nd line
  send_string("Vel= ");                 // Write on LCD
  send_string(char_rpm);                // Write char_rpm on LCD
  send_string(" RPM");                  // Write on LCD
  send_command(0x0C);                   // Display on, cursor off
}

int main(void)
{
 configure_clocks();

 lcd_init();
 send_command(0x80);              // Set cursor to beginning to 1st line
 send_string("PWM 25% ");         // Write on LCD

 P2DIR |= 0x02;                   // Set P2.1 as output.
 P2SEL |= 0x02;                   // Set P2.1 as PWM(TA1.1) output
 P2SEL2 = 0x00;                   // Clear P2SEL2

 P1REN |= 0X08;                   // Enable Pull-up/Down Resistor in P1.3
 P1OUT |= 0X08;                   // Select pull-up mode in P1.3
 P1IE  |= 0X18;                   // Enable interrupts in P1.3 & P1.4
 P1IES |= 0x08;                   // Set Hi-Lo edge in P1.3
 P1IES &= ~0x10;                  // Set Lo-Hi edge in P1.4
 P1IFG &= ~0X18;                  // Clear interrupt flags in P1.3 & P1.4

 // Timer A0
 TA0CTL = TASSEL_2 + MC_1 + ID_0;  // Set source SMCLK (TASSEL_2), control "Up
                                   to CCR0" (MC_1) e input divider "/1" (ID_0)
 TA0CCTL0 |= CCIE;                 // Enable Timer A0 interrupt
 TA0CCR0 = 1000-1;                 // Set cycle period 1 ms (frequency 1 KHz)

 // Timer A1:
 TA1CTL = TASSEL_2 + MC_1 + ID_0;  // Set source SMCLK (TASSEL_2), control "Up
                                   to   CCR0" (MC_1) e input divider "/1" (ID_0)
 TA1CCTL1 = OUTMOD_7;              // PWM2 output mode: 7 - PWM reset/set
 TA1CCR0 = 2000-1;                 // Set cycle period 2 ms (frequency 500 Hz)
```

```
  TA1CCR1 = 500-1;                          // Set PWM2 duty cycle in 25%

  __bis_SR_register (GIE);                  // Enable global interrupt

  miliseconds = 0;                          // Start time counter
  turn_counter = 0;                         // Start turn counter

  while (1){
  }
}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
   miliseconds++;                           // Increment time counter
}

#pragma vector = PORT1_VECTOR
__interrupt void Port_1(void)
{
  if(P1IFG&0x08)                            // Is there P1.4 interrupt pending?
  {
    if((P1IES&0x08))                        // Is this the Hi-Lo edge?
    {
      if (TA1CCR1 == (2000-1)){
        TA1CCR1 = (500-1);                  // Returns PWM to duty cycle 25%
        send_command(0x80);                 // Set cursor to beginning to 1st line
        send_string("PWM 25% ");            // Write on LCD
      }
      else {
        TA1CCR1 += 500;                     // Add 25% to PWM duty cycle
        __delay_cycles (50000);
        switch (TA1CCR1){
        case (1000-1):
          send_command(0x80);               // Set cursor to beginning to 1st line
          send_string("PWM 50% ");          // Write on LCD
          break;
        case (1500-1):
          send_command(0x80);               // Set cursor to beginning to 1st line
          send_string("PWM 75% ");          // Write on LCD
          break;
        case (2000-1):
          send_command(0x80);               // Set cursor to beginning to 1st line
          send_string("PWM 100%");          // Write on LCD
```

```
        break;
       default:
        send_command(0x80);                 // Set cursor to beginning to 1st line
        send_string("PWM 25% ");            // Write on LCD
        break;
       }
     }
   }
 }
 if(P1IFG&0x10)                 // Is there P1.3 interrupt pending?
 {
  if(!(P1IES&0x10))             // Is this the Lo-Hi edge?
  {
    if (turn_counter < 179){
      turn_counter++;           // Increment turn_counter
    }
    else {                      // If turn_counter is 180, the fan has completed 20 turns
      calc_rpm();
      turn_counter=0;           // Reset turn counter
      TA0CTL|=TACLR;            // Clears timer A0
      miliseconds = 0;          // Reset time counter
    }
  }
 }
 P1IFG &= ~0x18;                // Clear interrupt flags in P1.3 & P1.4
}
```

APÊNDICE E

Programa em linguagem C: Medidor de Tensão.

```
#include <msp430g2553.h>

// LCD control pins definitions
#define DATA_REG P1OUT = BIT5                  // set RS high (data register)
#define INST_REG P1OUT = (~BIT5)               // set RS low (instruction register)
#define ENABLE_PIN_HIGH P2OUT |= BIT0          // set Enable high
#define ENABLE_PIN_LOW P2OUT &= (~BIT0)    // set Enable Low

// Variable declaration
float ADC_value = 0;
int ADC_value_int = 0;
int ADC_value_dec = 0;
char int_ADC_value[5];
char dec_ADC_value[5];

// Implementation of itoa function (convert values in strings to show on LCD)
char *itoa(int from, char to[])
{
   char const digit[]  = "0123456789";
   char* p             = to;
   int shifter;

   if (from < 0) {
      *p++ = '-';
      from *= -1;
   }

   shifter = from;

   do {
      ++p;
      shifter = shifter / 10;
   } while (shifter);

   *p = '\0';

   do {
      *--p = digit[(from % 10)];
      from = from / 10;
   } while (from);

   return to;
}
```

```
void configure_clocks()
{
  WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog to prevent time out reset
  DCOCTL = CALDCO_1MHZ;           // DCO = 1 MHz
  BCSCTL1 = CALBC1_1MHZ;          // BC1 = 1 MHz
  BCSCTL2 = 0x00;                 // 0
  BCSCTL3 = 0x00;                 // 0
}

void delay_us(unsigned int us)
{
  while (us)
  {
    __delay_cycles(1);            // Frequency 1 MHz = Period 1 us
    us--;
  }
}

void delay_ms(unsigned int ms)
{
  while (ms)
  {
    __delay_cycles(1000);       // Frequency 1 MHz = Period 1 us  =>  (1ms = 1000 us)
    ms--;
  }
}

void data_write(void)
{
  ENABLE_PIN_HIGH;
  delay_ms(5);
  ENABLE_PIN_LOW;
}

void send_data(unsigned char data)
{
  unsigned char higher_nibble = 0x3c & (data >> 2);
  unsigned char lower_nibble = 0x3c & (data << 2);

  delay_us(200);

  DATA_REG;

  P2OUT = (P2OUT & 0xc3) | (higher_nibble);       // Send first 4 bits
  data_write();
  P2OUT = (P2OUT & 0xc3) | (lower_nibble);        // Send last 4 bits
  data_write();
}
```

```
void send_string(char *s)
{
  while(*s)
  {
    send_data(*s);
    s++;
  }
}

void send_command(unsigned char cmd)
{
  unsigned char higher_nibble = 0x3C & (cmd >> 2);
  unsigned char lower_nibble = 0x3C & (cmd << 2);

  INST_REG;
  P2OUT = (P2OUT & 0xC3) | (higher_nibble);       // Send first 4 bits
  data_write();

  P2OUT = (P2OUT & 0xC3) | (lower_nibble);        // Send last 4 bits
  data_write();
}

void lcd_init(void)
{
  P1DIR = 0x20;                          // Set P1.5 as output
  P1OUT = 0x00;                          // Set P1 outputs low
  P2DIR = 0x3D;                          // Set P2.0 P2.2 P2.3 P2.4 P2.5 as output
  P2OUT = 0x00;                          // Set P2 outputs low

  delay_ms(15);
  send_command(0x33);                    // Initialization code

  delay_us(200);
  send_command(0x32);                    // Initialization code

  delay_us(40);
  send_command(0x28);                    // 2 lines and 5×7 matrix (4-bit mode)

  delay_us(40);
  send_command(0x0E);                    // Display on, cursor on

  delay_us(40);
  send_command(0x01);                    // Clear display screen

  delay_us(40);
  send_command(0x06);                    // Increment cursor (shift cursor to right)

  delay_us(40);
  send_command(0x80);                    // Set cursor to beginning to 1st line
}
```

```
void voltage_measure(void){
  ADC_value = ADC10MEM;                        // Data from register ADC10MEM to variable
  ADC_value = (5*ADC_value)/1024;              // Calculate voltage (máx resol. 5V = 0x03FF)
  ADC_value_int = ADC_value;                             // Sets integer part
  ADC_value_dec = (ADC_value-ADC_value_int)*100;   // Sets decimal part
  itoa(ADC_value_int, int_ADC_value);                    // Convert to integer part to string
  itoa(ADC_value_dec, dec_ADC_value);                    // Convert decimal part to string
  if (ADC_value_dec < 10){
    send_command(0xC0);                          // Set cursor to beginning to 2nd line
    send_string("V= ");                          // Write on LCD
    send_string(int_ADC_value);                  // Write on LCD
    send_string(".0");                           // Write on LCD
    send_string(dec_ADC_value);                  // Write on LCD
    send_string(" Volts   ");                    // Write on LCD
    send_command(0x0C);                          // Display on, cursor off
  }
  else {
    send_command(0xC0);                          // Set cursor to beginning to 2nd line
    send_string("V= ");                          // Write on LCD
    send_string(int_ADC_value);                  // Write on LCD
    send_string(".");                            // Write on LCD
    send_string(dec_ADC_value);                  // Write on LCD
    send_string(" Volts   ");                    // Write on LCD
    send_command(0x0C);                          // Display on, cursor off
  }
}

int main(void){
  configure_clocks();

  lcd_init();
  send_command(0x80);                    // Set cursor to beginning to 1st line
  send_string("Voltage Measure");        // Write on LCD

   ADC10AE0 |= BIT7;                     // Config. P1.7 as input of ADC10

   ADC10CTL1 |= INCH_7                   // Config. channel 7 of ADC
           + SHS_0                       // Config. sample-and-hold by software
           + ADC10DIV_7                  // Config. division /8
           + ADC10SSEL_3                 // Config. cock source SMCLK
           + CONSEQ_0;                   // Config. ADC conversion

   ADC10CTL0 |= SREF_0                   // Config. Vcc = Vr+ e GND = Vr-
           + ADC10SHT_0                  // Config. sample-and-hold time 4 clock cycles
```

```
            + ADC10ON                   // Activate conversor
            + ADC10IE;                  // Enable ADC10 interrupt

   __bis_SR_register(GIE);              // Enable global interrupt

   ADC10CTL0 |= ENC + ADC10SC;          // Start conversion

  while (1)
  { }
}
#pragma vector=ADC10_VECTOR             // Interrupt vector
__interrupt void ADC10_ISR(void)
{
 voltage_measure();

 __bis_SR_register(GIE);                // Enable global interrupt

 ADC10CTL0 |= ENC + ADC10SC;            // Start new conversion
}
```