ANDRÉ VIEDMA CESTAROLLI

SIMULAÇÃO DE FOTOPERÍODO ATRAVÉS DA MODULAÇÃO DA INTENSIDADE LUMINOSA DE LEDS POR MICROCONTROLADOR MSP430 OPERADO VIA COMUNICAÇÃO SEM FIO BLUETOOTH

Trabalho de Conclusão de Curso
apresentado à Escola de Engenharia de São
Carlos, da Universidade de São Paulo

Curso de Engenharia da Computação com
ênfase em sistemas embarcados

ORIENTADOR:

Professor Dr. Carlos Dias Maciel

São Carlos
2012

Cestarolli, André Viedma
C421s SIMULAÇÃO DE FOTOPERÍODO ATRAVÉS DA MODULAÇÃO DA INTENSIDADE LUMINOSA DE LEDS POR MICROCONTROLADOR MSP430 OPERADO VIA COMUNICAÇÃO SEM FIO BLUETOOTH / André Viedma Cestarolli; orientador Carlos Dias Maciel. São Carlos, 2012.

Monografia (Graduação em Engenharia de Computação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

1. MSP430. 2. PWM. 3. dimerização. 4. LED. 5. bluetooth. I. Título.

FOLHA DE APROVAÇÃO

Nome: André Viedma Cestarolli

Título: "Simulação de fotoperíodo através da modulação da intensidade luminosa de LEDS por microcontrolador MSP430 operado via comunicação sem fio Bluetooth"

Trabalho de Conclusão de Curso defendido em 30/11/12.

Comissão Julgadora:	**Resultado:**
M.Sc. Jen John Lee SEL/EESC/USP	APROVADO
M.Sc. Wagner Endo SEL/EESC/USP	APROVADO

Orientador:
Prof. Associado Carlos Dias Maciel - SEL/EESC/USP

Coordenador pela EESC/USP do Curso de Engenharia de Computação:

Prof. Associado Evandro Luís Linhari Rodrigues

Resumo

Os diodos emissores de luz – LEDs estão por toda parte em incontáveis aplicações e aparecem como opção promissora para iluminação por seu alto rendimento, barateamento do preço e grande durabilidade. Nesse contexto surge a necessidade de variar a intensidade luminosa (também conhecido como dimerização) para melhor aproveitamento da energia e outras aplicações: uso da intensidade para transmitir informação, efeitos luminosos, simulação do ciclo circadiano, etc. Usou-se modulação por largura de pulso (PWM) para realizar esse efeito.

O uso de tecnologias de transmissão digital de dados sem-fio também tem se popularizado bastante. Existem diversos padrões como Wi-Fi, 3G, Wi-Max, **Bluetooth**, infravermelho, ZigBee, separadas principalmente por requisitos de alcance e consumo de energia. Derivada dessas tecnologias e com grande potencial ainda a ser explorada temos a rede WPAN (wireless personal area network) ou simplesmente rede pessoal, criada a partir de dispositivos portáteis, pequenos e geralmente com energia limitada tais como celulares, relógios, tablets, tocadores de música entre outros.

Pensando na importância de dominar essas técnicas é proposto um projeto no qual LEDs são controlados por sinal PWM gerado em um microcontrolador MSP430. Um programa Java no computador permite controlar o MSP430 via **Bluetooth** e com isso variar a intensidade do led, mudando remotamente os parâmetros do PWM. O programa Java fornece uma interface prática com funcionalidades do tipo: mudar instantaneamente a intensidade dos leds para qualquer valor, modo de demonstração e mudar os parâmetros e realizar uma simulação de ciclo circadiano. Foi projetado também um circuito elétrico conhecido como *driver* de corrente para fornecer energia à carga que o microcontrolador não poderia prover e que possa chavear na velocidade necessária.

Abstract

Light emitting diodes are everywhere and are a promising option for lighting because of their high performance, low prices and great durability. In this scenario dimming is a very important feature but now it cannot be done the way it used to be done with old incandescent light bulbs. With pulse width modulation, or just PWM, one can save energy besides many other applications like signal transmission, lighting effects, circadian rhythm simulation and so on.

The use of wireless digital data transmission technologies is also widespread. There are lots of standards like Wi-Fi, 3G, Wi-Max, **Bluetooth**, infrared, ZigBee, separated by range and power consumption. Wireless personal networks, WPANs, are also a field yet to be explored.

Considering all these technologies and their relevance, a project is suggested: to dim leds through a pwm signal generated in a MSP430 microcontroller. A Java program controls the MSP430 remotely via serial **Bluetooth** communication and sets the PWM parameters to simulate the circadian rhythm. A current driver circuit is also conceived in order to provide enough power to the leds.

Sumário

Lista de Figuras

1. Introdução

Este trabalho tem como objetivo realizar o controle da intensidade luminosa em uma carga de LEDs utilizando microcontrolador MSP430, comunicação sem fio com o computador e uma interface com usuário no computador. O programa no computador irá realizar a simulação do fotoperíodo, como exemplo prático de utilização do *hardware* construído.

1.1 Objetivos e motivação

Para variar a intensidade luminosa nos LEDs o método usado será o PWM. Variar a tensão sobre os LEDs poderia diminuir o rendimento, deslocar a região do espectro cromático de operação, modificando a cor, ou mesmo apagá-lo caso saia da região de operação.

O ciclo circadiano é o ciclo de 24 horas que influencia todos os seres vivos. Seu principal fator é a luz do sol e chamamos de fotoperíodo o ciclo diário da intensidade luminosa e a maneira como se dá essa variação. Tem-se por objetivo realizar uma simulação simples desse ciclo: uma rampa crescente na qual a intensidade varia de zero ao valor máximo, permanecendo nesse valor por um determinado período, uma rampa agora decrescente até intensidade zero e uma etapa final com ausência de luz. O tempo total e os tempos de cada etapa serão configuráveis. As rampas de subida e descida terão incremento linear.

Para alcançar esse resultado propõe-se a construção de blocos funcionais de *hardware* e *software* que integrados corretamente irão resolver o problema. Vale ressaltar que a síntese desses blocos gera competências muito úteis pois estes mesmos podem ser reaproveitados e adaptados para uso em outros projetos.

1.2 Visão geral

Uma visão geral dos blocos do projeto é apresentada na Figura 1:

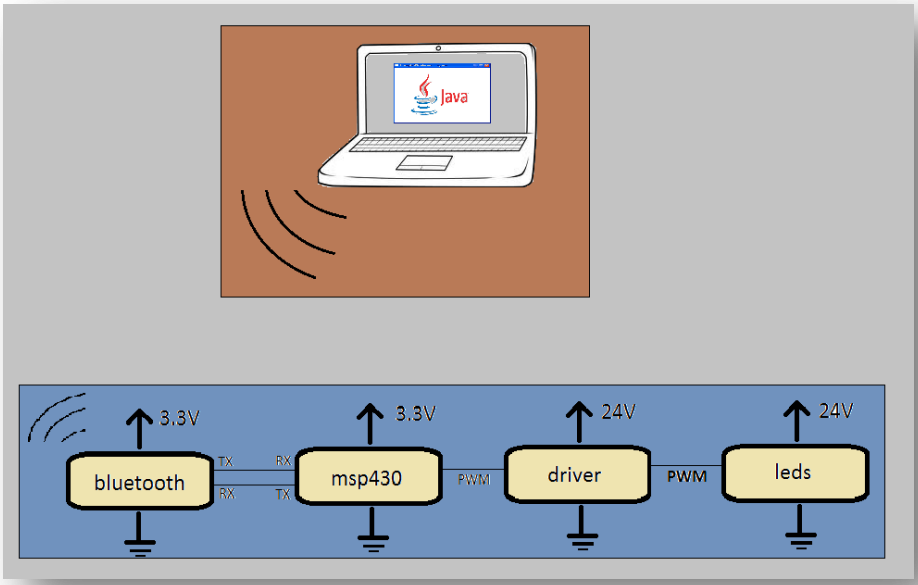


Figura 1 – Visão geral dos blocos do sistema: tensões de alimentação e sinais

Nela vemos que o MSP430 tem papel central: ele recebe os sinais serialmente do módulo **Bluetooth** e gera o PWM que será amplificado pelo driver de corrente. Mais detalhes sobre os módulos individualmente serão apresentados nos capítulos a seguir.

1.3 Organização do trabalho

O projeto foi dividido em 5 módulos independentes. São eles:

- Programa Java
- **Bluetooth**
- MSP430 serial
- MSP430 PWM
- Driver de corrente

A escolha da linguagem Java se deu pelas conhecidas facilidades que esta provê para desenvolvimento ágil da lógica em código e interfaces de usuário, comunicação serial através de bibliotecas e reuso de código.

O MSP430 foi escolhido tendo em vista a importância atual de mercado e a facilidade de aquisição desse chip. A linha PIC da Microchip Technology e a plataforma *open source* Arduíno também seriam escolhas plausíveis. Da arquitetura ARM, que está em evidência no

mercado hoje em dia com processadores embarcados poderosos, também existem chips de baixo custo que atenderiam bem as especificações.

Para formar a rede sem fio temos o **Bluetooth** e o Zigbee como padrões semelhantes em termos de alcance e consumo. Ambos podem se enquadar na definição de WPAN na qual dispositivos portáteis trocam informações a curta distância formando uma rede pessoal. Como principais diferenças temos que o Zigbee possui um tempo reduzido de reinicialização a partir do modo de economia de energia e permite que informação seja retransmitida por vários módulos até chegar ao destino, formando uma rede em malha (ou rede *mesh*). Já o **Bluetooth** permite taxas de dados maiores (com gasto superior de energia) e fácil integração com notebooks e celulares que normalmente já possuem antenas desse tipo. Como economia de energia não é o foco (temos uma fonte de alimentação externa disponível) e integração com o computador é parte deste trabalho esse padrão foi escolhido.

O **Bluetooth** usado é um CI de baixo custo com antena e interface serial integrados, *firmware* **Bluetooth** versão 2.1 + EDR e pareamento simples seguro (SSP). Pronto para receber comandos AT e usar. Com alcance de aproximadamente 10 metros é o suficiente para a aplicação.

O driver de corrente surge da necessidade de fornecer energia para uma carga genérica a partir de um pino do microcontrolador com sérias limitações de corrente. Apesar de cada LED usualmente consumir pouca energia (10 a 100mW) e consequentemente drenar pouca corrente, uma carga constituída por muitos LEDs ou mesmo LEDs de maior potência pode necessitar de vários ampères. É necessário uma maneira de converter o sinal do MSP430 para chavear uma fonte externa de corrente contínua (geralmente ligada a rede elétrica 127V), ou seja, fazer com que a carga receba o sinal do microcontrolador mas drene corrente apenas da fonte, e sem perder as características do sinal, sem distorcê-lo.

Com o objetivo de contextualizar o leitor a respeito das tecnologias utilizadas, termos e siglas o Capítulo 2 apresenta uma revisão de alguns conceitos técnicos, de eletrônica e de computação. O Capítulo 3 detalha a implementação e integração dos módulos. No Capítulo 4 são explicados detalhes a respeito do funcionamento e da implementação e no Capítulo 5 a conclusão do trabalho e resultados.

2. Revisão de conceitos

2.1 MSP430

O MSP 430 é uma linha de microcontroladores de 16 bits de baixo custo da **Texas Instruments - TI** voltada para baixo consumo e uso em sistemas embarcados.

Com varias gerações de famílias e modelos que podem chegar a 25MHz de *clock* o MSP430 se destaca por possuir linhas com oscilador interno, *timers*, USART, SPI, i²C, PWMs, conversores analógico digital – ADC e outros perifericos conhecidos embutidos. Também é conhecido pelos modos de economia de energia, no qual pode chegar a drenar apenas 1 µA e retornar rapidamente ao modo ativo em menos de 6 micro segundos, de acordo com seu *datasheet* [1].

A **Texas Instruments** lançou em 2010 uma plataforma de baixo custo pronta para usar chamada TI MSP430 Launchpad. Ela inclui 2 CIs MSP430, cristal de 32kHz, uma placa pronta com soquete, trilhas e conexões de entrada e saída e uma interface usb para gravação e *debug* . A Figura 2 mostra a placa utilizada no projeto.

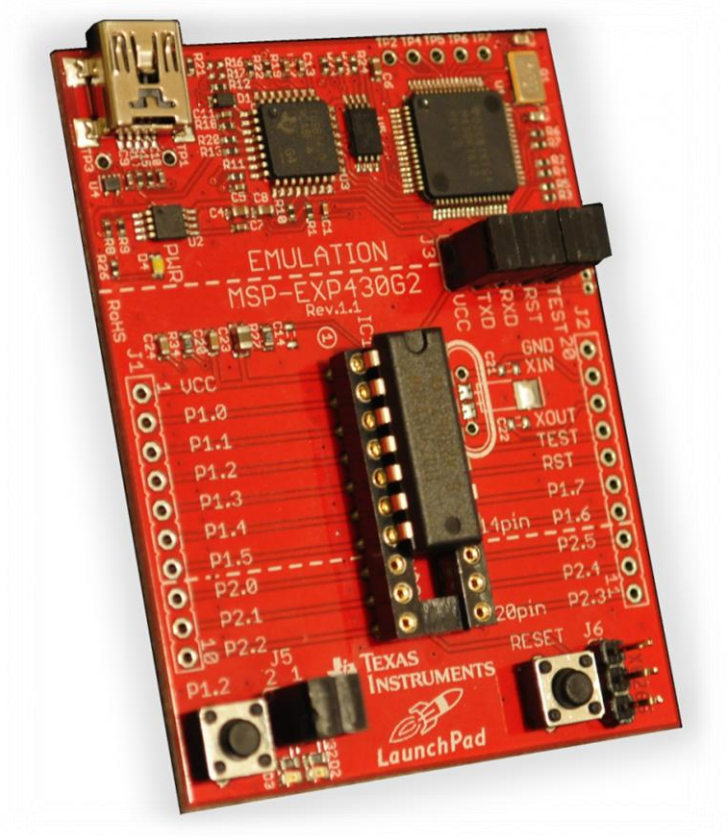


Figura 2 – MSP430 Launchpad v1.4 com chip MSP430G2231[1]

[1] Figura retirada de http://www.ti.com/tool/msp-exp430g2 (Nov/2012)

O chip usado é modelo MSP430G2553 e tem como características:

- *clock* de até 16 MHz;
- 16kB de memória flash;
- 512B memória RAM;
- 8 canais de ADC - 10 bits de resolução;
- 2 Timers;
- Módulo de comunicação serial (USCI – I2C, SPI, HW UART);
    - apesar deste modelo possuir UART em *hardware* foi usada uma comunicação half-duplex em *software* pois o modelo anteriormente usado não tinha essa UART disponível conforme será explicado mais adiante.

Para programá-lo foi usado o Code Composer Studio (CCS) versão 5 que é a plataforma para desenvolvimento para processadores embarcados da **Texas Instruments** como DSPs, dispositivos baseados em ARM e o próprio MSP430. O CCS é baseado no *framework open source* Eclipse e é uma solução completa para desenvolvimento, debug e *deploy*.

2.2 Modulação por largura de pulso - PWM

O PWM (*pulse width modulation*) é o método usado para dimerizar os LEDs [2]. É formado a partir do chaveamento entre nível lógico baixo e nível lógico alto de um pino no microcontrolador MSP430 (neste caso: 0V e 3.3V) a uma taxa bem definida. Como uma onda quadrada, mas com o tempo "ligado" não necessariamente simétrico ao tempo "desligado". O ciclo de trabalho (do Inglês *duty cycle*) é caracterizado pelo percentual de tempo que a saída permanece ativa. Assim um *duty cycle* de 100% representaria uma saída permanentemente ligada. Além de poder ser usado para regular a quantidade de energia entregue a uma carga o PWM também tem outras aplicações na eletrônica tais como: transmissão de sinais, regulação de voltagem, efeitos e amplificação de áudio .

Os LEDs precisam ser dimerizados desta maneira pois reduzir a tensão de alimentação, como era feita com as lâmpadas incandescentes, reduz o desempenho ou mesmo apaga o LED. A junção PN do LED, se diretamente polarizada, mantém-se com corrente relativamente constante mesmo com incremento de tensão. O gráfico da Figura 3 ilustra uma curva típica de tensão x corrente de um diodo, dos quais os LEDs são uma categoria especial e portanto compartilham muitas propriedades.

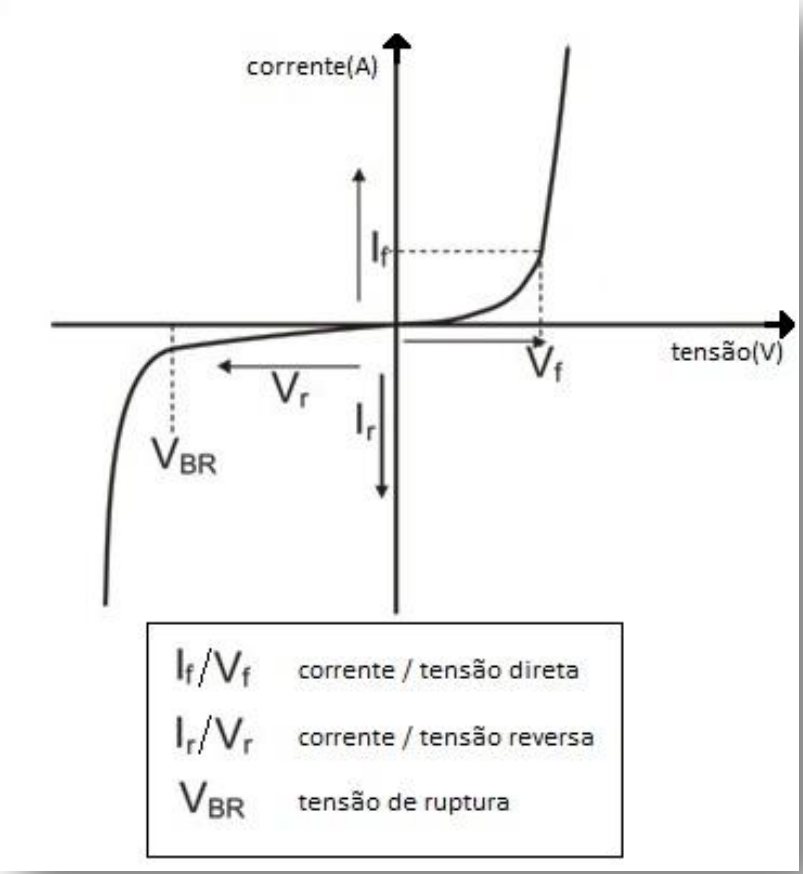


Figura 3 - Curva corrente x tensão do diodo.

A frequência de chaveamento do PWM também é um fator importante para caracterizá-lo, juntamente com o ciclo de trabalho. No caso de telecomunicações ela está relacionada com a taxa de amostragem ou a frequência da portadora. Se estiver regulando a energia entregue a um motor elétrico deve respeitar características físicas do mesmo: se for muito baixa pode causar solavancos e danos. Se for muito alta o motor pode não ter sensibilidade devido a sua construção e ser ignorada. No caso do PWM nos LEDs, se a frequência for muito baixa - abaixo de 30 quadros por segundo [3] - poderá ocorrer o fenômeno denominado *flickering* no qual o olho humano consegue perceber a taxa de atualização e visualizar o LED piscando. Se for muito alta o LED não conseguirá descarregar a energia acumulada na junção e não desligará completamente. Nesse caso o LED permaneceria aceso mais tempo do que deve e a impressão seria de um percentual de brilho maior que o percentual do duty cycle do PWM. A frequência utilizada neste caso é 100 Hz (mais detalhes no capítulo 3, Implementação).

Um exemplo de vários PWMs com *duty cycles* distintos é mostrado na Figura 4:

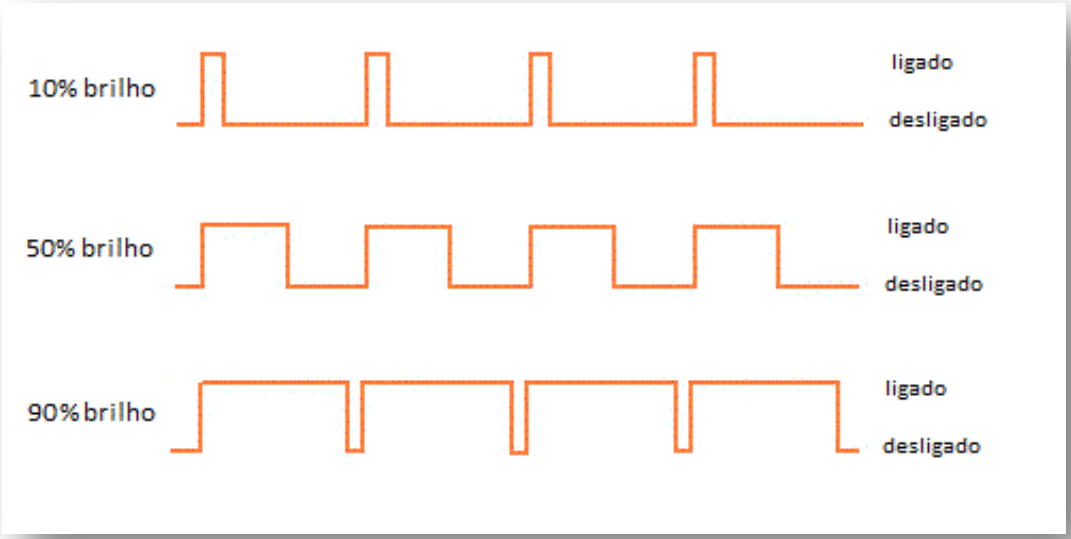


Figura 4 – Curvas de PWMs com duty cycle 10% - 50% - 90%[2]

2.3 UART

UART é a sigla de Transmissor/Receptor Universal Assíncrono (do ingles Universal Synchronous Receiver Transmitter). É um formato padrão para transmissão de dados serialmente utilizando 2 fios: um para enviar, outro para receber. No modo *full-duplex* a transmissão e recepção podem ocorrer concomitantemente e no modo *half-duplex* a comunicação se dá nos 2 sentidos, porém num sentido de cada vez.
O transmissor envia os bits de maneira sequencial, um de cada vez e o receptor reune os bits para formar o byte [4]. O quadro é constituido da seguinte maneira:

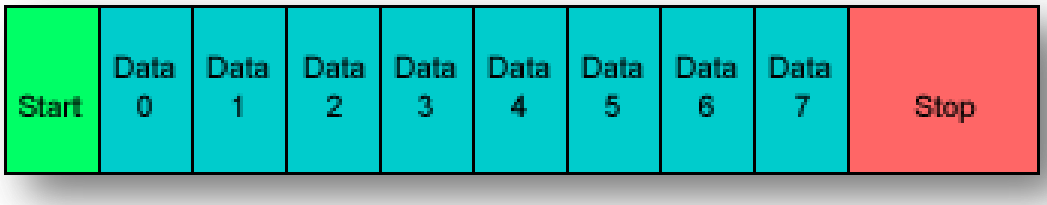


Figura 5 - Quadro UART para transmissão serial com 1 *start bit* e 2 *stop bits*

[2] Figura retirada e adaptada a partir do site: Dallas Personal Robotics Group - http://www.dprg.org/tutorials/2005-11a/index.html (Dez/2012)

O *start bit* é nível lógico baixo. Isso é um legado histórico da telegrafia onde manter a linha energizada mostrava que o transmissor não estava danificado. Em seguida são enviados os *bits* um após o outro (respeitando os intervalos de tempo entre um e outro). Pode existir um *bit* adicional de paridade e então um ou dois *stop bit(s)* (nivel lógico alto).

Os intervalos de tempo dependem do *baudrate*. Assim como o formato do quadro a ser enviado e o tipo de paridade essa informação dever ser conhecida por ambos os lados ou os dados serão perdidos durante a transmissão. De uma maneira simplificada o *baudrate* define a taxa de *bits* transmitidos por segundo ou então, para o microcontrolador, o tempo que ele deve manter o valor na saída antes de passar para o próximo para que o receptor possa identificá-lo corretamente.

2.4 Driver de Corrente

Todos os microcontroladores tem limitação da corrente que pode passar por um pino de ES (entrada e saída). O MSP430 não é diferente. Para poder chavear com segurança uma carga devemos passar o sinal por um circuito conhecido como driver de corrente. Este circuito dará um grande ganho (amplificará o sinal) e irá transferir para uma fonte externa a drenagem de corrente. Essa fonte não terá problemas em prover corrente da ordem de alguns ampères se ela foi projetada para tal. Um driver simples pode ser concebido a partir de um transistor bipolar de junção (TBJ) NPN conforme o esquema da Figura 6:

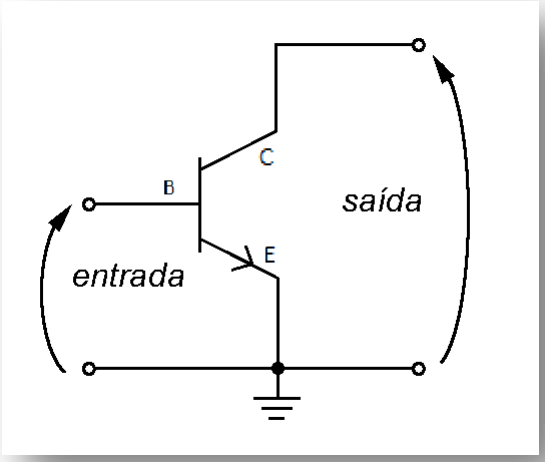


Figura 6 – Driver de corrente simples: transistor NPN em montagem emissor comum

Essa disposição é conhecida como emissor comum [5]. Nela uma pequena corrente, da ordem de miliampères, entrando na base do transistor (no caso de um NPN), irá “fechar a chave” entre coletor e emissor como um curto circuito (a menos de um Vce, tensão coletor emissor, geralmente da ordem de 0,5 V) e a corrente irá fluir livremente agora restrita pela impedância da

carga. O transistor também limita a corrente que pode passar pelo coletor-emissor, e esse valor é fornecido na especificação e depende de parâmetros de sua construção. A razão entre a corrente emissor-coletor e a corrente da base é também conhecida como ganho, h_{fe} ou parâmetro β do transistor, e é geralmente maior que 100.

Existem transistores projetados especificamente para trabalhar dessa maneira conhecidos como Darlington (Figura 7) que possuem 2 ou mais transistores num mesmo encapsulamento e podem atingir ganho de corrente de 1000 vezes ou maior.

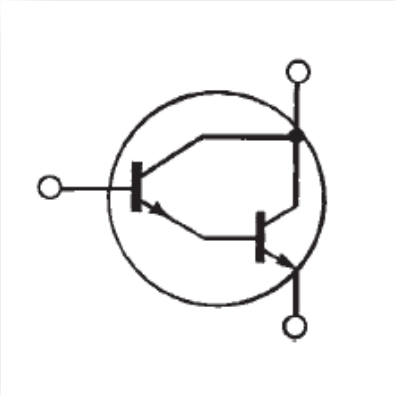


Figura 7 - Símbolo transistor Darlington: dois transistores NPN em série

A configuração usada é chamada de par de Sziklai, semelhante a de Darlington. Utiliza um NPN para o primeiro estágio e um PNP para o segundo. Mais detalhes sobre o circuito, bem como cálculos das correntes e resistores utilizados serão demonstrados no próximo capítulo.

2.5 Ciclo circadiano

Ciclo circadiano, ou ritmo circadiano, é qualquer processo biológico que possui uma oscilação com período de aproximadamente 24 horas. Esse ritmo tem sido largamente observado em plantas, animais, fungos e bactérias. O termo circadiano é derivado do latim “circa” (cerca, aproximadamente) e “diem” (dia): cerca de um dia. Embora o ciclo seja endógeno (criado e sustentado por processos internos) eles são ajustados e atrelados ao meio ambiente por fatores conhecidos como *zeitgebers* (cronômetro, na tradução livre do alemão) dos quais o mais importante é a luz do sol. Esse processo no corpo humano, bem como em diversos animais, regula ritmos físicos e psicológicos como humor, temperatura corporal, estado de vigília, passando por crescimento e renovação das células, liberação de hormônios e enzimas . Tem também influência conhecida nas plantas [6]. Iremos criar um dispositivo que simula essa variação da luz e realiza o ciclo com parâmetros configuráveis. Em um ambiente isolado de luz externa isso pode ser usado para testes biológicos em animais e plantas.

3. Implementação

Este capítulo descreve o processo de implementação dos módulos, listagem e custo dos materiais usados bem como os detalhes de funcionamento, problemas durante a modelagem e soluções escolhidas. Em seguida temos como se deu a etapa de integração.

3.1 Lista de Materiais

Lista de materiais utilizados na confecção do projeto (2012):

- 1x MSP430 Launchpad: U$4.30 ~ R$9,00 – adquirido no site da **Texas Instruments**
- 1x Módulo **Bluetooth**: U$6.60 ~ R$14,00 – adquirido na internet
- 2x resistor 4k7 ohms: R$1,50 (pacote com 50) – adquiridos em São Carlos/SP
- 1x resistor 50k ohms: R$1,50 (pacote com 50)
- 1x resistor 470 ohms: R$1,50 (pacote com 50)
- 1x transistor NPN bc546b (~R$2,00)
- 1x transistor PNP 2n6248 (~R$2,00)
- 1x placa padrão (~R$2,00)
- 1x led verde (~R$1,00)

Custo total do projeto: aproximadamente 30 reais

3.2 Implementação dos módulos

Primeiramente foi feito um estudo dos aspectos relevantes da arquitetura do MSP430. O documento de especificação (arquivo slau144i.pdf) do site da **Texas Instruments** é bastante completo. Entre os itens analisados vale ressaltar: diagrama de blocos, conjunto de instruções, barramentos, sistema de *clock*, E.S., interrupções, modos de operação, registradores especiais, watchdog timer e, principalmente timers. Os timers são de grande importância para esse projeto pois eles contam tempo para gerar o *baudrate* da comunicação serial e o PWM também é gerado a partir deles.

Os módulos foram então concebidos na seguinte ordem:

MSP430 PWM

MSP430 serial

Programa Java

Driver de corrente

Bluetooth

3.2.1 MSP430 PWM

O programa C foi obtido no site da **Texas Instruments** e modificado para adequar-se ao modelo e as necessidades. O trecho de codigo a seguir foi utilizado no MSP430G2231:

```
#include "msp430G2231.h"

void main(void)
{
 WDTCTL = WDTPW + WDTHOLD;  // Stop WDT

 P1DIR |= BIT6;             // P1.6 to output
 P1SEL |= BIT6;             // P1.6 to TA0.1

 CCR0 = 1000-1;             // PWM Period
 CCTL1 = OUTMOD_7;          // CCR1 reset/set
 CCR1 = 250;                // CCR1 PWM duty cycle
 TACTL = TASSEL_2 + MC_1;   // SMCLK, up mode

 _BIS_SR(LPM0_bits);        // Enter LPM0
}
```

O *watchdog timer* é desligado pois não será usado por enquanto e para que não efetue um *reset* periódico no sistema. O bit 6 do porto 1 é configurado como saída e tem o PWM como fonte de dados desse pino. Nas linhas que se seguem é configurado o valor máximo para o qual o timerA deverá contar, o modo *set/reset* do PWM , o valor até o qual a saída será mantida em alta (*duty cycle*) e o periférico é ligado ao *clock* sub principal (SMCLK) para que comece a contar. A CPU é desligada – ou colocada para dormir, indefinidamente.

Após esta breve inspeção podemos afirmar que o código é bastante imediato e elegante. Nele o perférico usado (Timer para PWM) é configurado e o processador é colocado em modo de enconomia de energia. Não é, entretanto, de fácil confecção para um programador novo com MSP430 pois utiliza várias diretivas *#define* dentro do arquivo de cabeçalhos incluído (arquivo .h). Essas constantes definidas no programa são, principalmente, associadas a registradores especiais (*special function registers*, ou apenas SFR) mapeados na memória do chip onde são feitas as configurações principais. Podemos acessar essas regiões diretamente e configurar os

bits que nos interessam usando operações bit-a-bit da linguagem C, mas essas constantes servem para facilitar o uso e deixar o código portável entre as famílias de MSP430.

O MSP430 é uma máquina de arquitetura von-Neumann na qual um espaço de endereçamento é compartilhado por SFR, periféricos, memória RAM, Flash/ROM, vetor de interrupções. O acesso pode ser feito por bytes ou palavras de dados.

A localização dos SFR no mapeamento da memória é ilustrada na Figura 8:

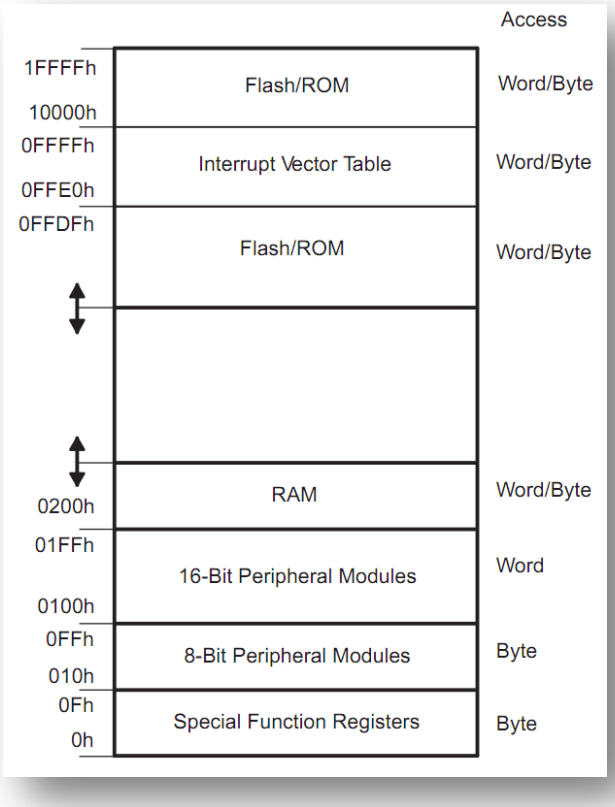
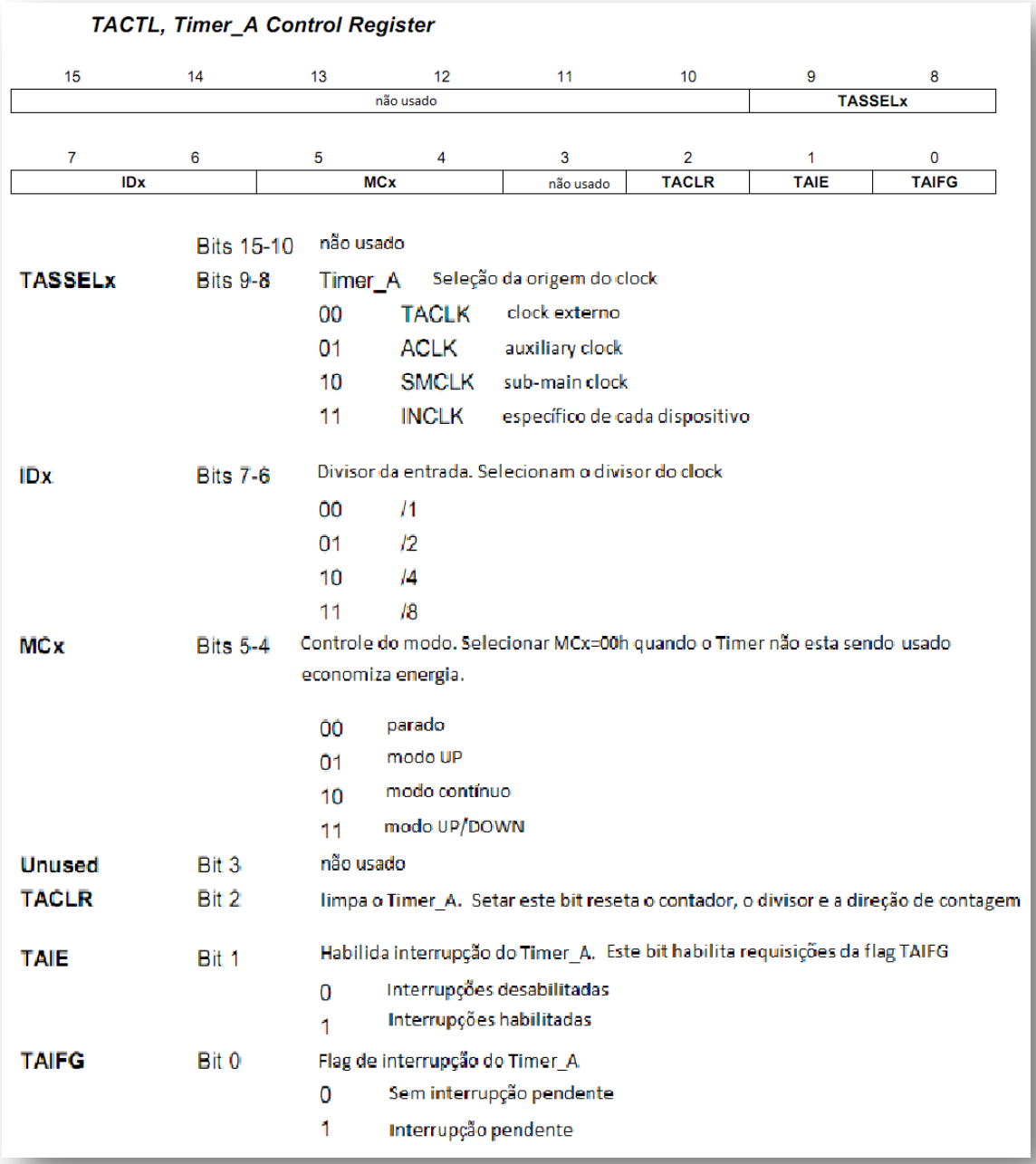


Figura 8 – Mapeamento da memória do MSP430 em endereços

Para exemplificar os SFR vamos mostrar o uso do registrador TACTL (Timer A control) que no código acima recebe as constantes TASSEL2 e MC_1. No documento (*datasheet*) desse modelo de MSP verificamos a seguinte disposição:



TACTL, Timer_A Control Register

15	14	13	12	11	10	9	8
não usado						TASSELx	

7	6	5	4	3	2	1	0
IDx		MCx		não usado	TACLR	TAIE	TAIFG

	Bits 15-10	não usado
TASSELx	Bits 9-8	Timer_A Seleção da origem do clock
	00	TACLK clock externo
	01	ACLK auxiliary clock
	10	SMCLK sub-main clock
	11	INCLK específico de cada dispositivo
IDx	Bits 7-6	Divisor da entrada. Selecionam o divisor do clock
	00	/1
	01	/2
	10	/4
	11	/8
MCx	Bits 5-4	Controle do modo. Selecionar MCx=00h quando o Timer não esta sendo usado economiza energia.
	00	parado
	01	modo UP
	10	modo contínuo
	11	modo UP/DOWN
Unused	Bit 3	não usado
TACLR	Bit 2	limpa o Timer_A. Setar este bit reseta o contador, o divisor e a direção de contagem
TAIE	Bit 1	Habilida interrupção do Timer_A. Este bit habilita requisições da flag TAIFG
	0	Interrupções desabilitadas
	1	Interrupções habilitadas
TAIFG	Bit 0	Flag de interrupção do Timer_A
	0	Sem interrupção pendente
	1	Interrupção pendente

Figura 9 – Registrador TACTL (Timer A Control) e função de seus bits[3]

Pode-se configurar os campos desse registrador de 16 bits diretamente atribuindo um valor hexadecimal ou binário com as configurações desejadas. Mas isso não é recomendado pois iria sobrepor os valores anteriores. Para manter os valores atuais e modificar apenas alguns campos pode-se usar operações bit-a-bit do C como AND (símbolo &) ou OR (símbolo |). Mas essa ainda não é a maneira mais prática. No arquivo de cabeçalhos (msp430G2231.h) incluído:

[3] Figura retirada adaptada do *datasheet* slau144i.pdf [1]

```
#define TASSEL_2     (2*0x100u)  /* Timer A clock source select: 2 - SMCLK */
#define MC_1         (1*0x10u)   /* Timer A mode control: 1 - Up to CCR0 */
```

Pode-se agora usar essas constantes e saber que valores serão atribuídos ao registrador de controle TACTL.

Quanto a PWM gerada por esse código: a cada pulso do *clock* SMCLK o registrador de 16 bits do TimerA TAR é incrementado. Ao atingir o valor de 250 ele é comparado com CCR1 , e verificando a igualdade, a saída (no porto1, pino 6) é colocada em nível lógico alto de acordo com o modo *reset/set* configurado. Depois, ao atingir 999 (valor de CCR0) a saída vai para 0 caracterizando um PWM com duty cycle de 25%.

O pino 6 na placa MSP430 Launchpad está ligado ao led vermelho. É possível ver o LED dimerizado, brilhando com menos intensidade.

A frequência de operação do PWM depende da velocidade que o MSP430 está operando: quanto maior for o *clock* mais rápido o valor do período em CCR0 é atingido. A versão final do código (ANEXO 2) usa um *clock* principal no MSP430 de 1,1MHz (que também alimenta o SMCLK) então o valor do período deve ser 11000 para termos um PWM a 100Hz.

3.2.2 MSP430 serial

O processo de desenvolvimento da comunicação serial UART foi um pouco mais complexo e lento pois envolve o uso de rotinas de interrupção e temporização precisa. Foi criado, primeiramente, um projeto que apenas incrementava e enviava periodicamente um byte do microcontrolador para o computador, através do porto serial virtual que o MSP430 Launchpad cria pela porta USB. Do lado do computador, um programa terminal qualquer (teraterm, realterm) no Windows conectado ao porto serial COM correspondente, lia os dados. O código está disponível no ANEXO 1.

Em seguida foi concebido o *software half duplex*. Este modo de transmissão UART permite tanto transmissão quanto recepção de dados contanto que não ocorram ao mesmo tempo. Um pino do porto P1 (chamado RXD) deve ficar preparado para, ao mudar de 1 para 0, gerar uma interrupção e na rotina de atendimento preparar o timer para efetuar o recebimento. O timer passa a gerar periodicamente uma interrupção interna de acordo com o baudrate e amostrar o pino RXD.

Após 10 valores terem sido lidos, o start bit, o byte transmitido e o stop bit, o método monta o valor em RXBbyte, copia para TXByte e chama o método Transmit(). Dessa maneira todo valor recebido pelo controlador é “ecoado” de volta para o emissor. Esse valor retornado é usado pelo

software do lado do computador para validar que o valor foi recebido e manter o sistema num estado consistente com o que é apresentado na tela.

O código referente a versão final está disponível no ANEXO 2. Ele possui algumas modificações em relação ao UART *half duplex* original pois:

1. Foi preciso adaptá-lo para que funcionasse junto ao PWM
2. Código de tratamento de alguns comandos foram incluídos para que o programa do computador pudesse controlá-lo. Mais detalhes na próxima seção.
3. Os pinos TXD e RXD anteriormente em P1.1 e P1.2 (padrão do Lauchpad que permite comunicação com o pc via usb) passaram para P1.5 e P1.6, liberando a comunicação original para deploy e debug. Isso apenas depois da integração com o **Bluetooth**

Até este momento foi usado o chip MSP430G2231 da versão 1.4 do kit Launchpad. Esse chip apenas possui 1 timer, o TimerA e com ele era possível ou gerar o PWM ou fazer a comunicação serial em *software*. Para poder utilizar ambos no mesmo chip era necessário um chip com mais de um timer ou fazer alguma modificação nos códigos para que eles coexistissem, compartilhassem o timer e não inteferissem um no outro. Tivemos acesso a versão 1.5 da Launchpad com o chip MSP430G2553, uma versão mais recente e revisada. Esse chip possui UART em *hardware* e apenas configurando os registradores de controle do dispositivo ele faria a comunicação serial automaticamente. Para aproveitar o código desenvolvido anteriormente, foi escolhido não trabalhar com essa UART em *hardware*. O PWM passou a utilizar o Timer1_A3 e o UART continuou com o timer original, agora chamado de Timer0_A0. Vale ressaltar que a portabilidade de código C de um modelo de chip para outro foi muito facilitada devido ao fato das famílias do MSP430 serem muito próximas necessitando apenas a troca de alguns registradores pelos equivalentes e, como foi neste caso, do nome do vetor de interrupção.

3.2.3 Programa Java

Após a concepção da UART em *software* no MSP430 foi necessário desenvolver um programa no computador para testar a UART criada. O projeto Java TwoWaySerialComm utiliza a biblioteca RXTX, uma biblioteca Java sob licença GNU LGPL que usa implementação nativa (JNI – Java Native Interface) para disponibilizar acesso as portas seriais no computador.

O TwoWaySerialComm conecta na COMx (x variável) com baudrate, paridade, *start* e *stop bits* pré-definidos, e envia caracteres do console do computador para o MSP430. Nessa versão os seguintes comandos foram implementados:

- Letra a (codigo ascii 97) – incrementa o brilho em 10%
- Letra z (codigo ascii 122) – decrementa o brilho em 10%
- Letra Q

Com a UART do MSP respondendo aos comandos era possível controlar a PWM pelo programa Java TwoWaySerialComm. O programa RXTXComm, baseado no anterior, agora com interface gráfica SWING possibilita configurar a simulação do ciclo circadiano. Ele é composto por 2 classes: Connection.java (ANEXO 3) e SerialScreen.java. A interface do RXTXComm é apresentada na Figura 10:

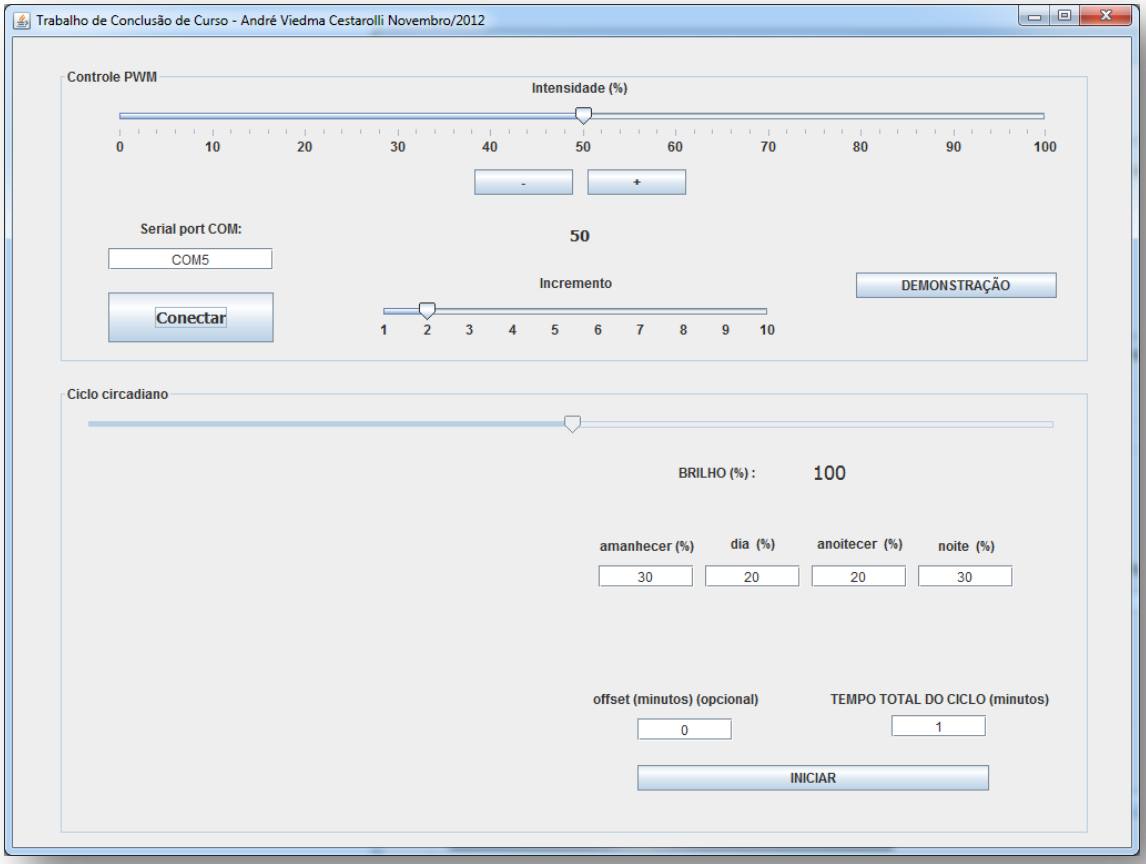


Figura 10 – Tela do programa RXTXComm em execução

Na parte de “Controle PWM” temos operações básicas como configurar o valor de intensidade para qualquer valor desejado a qualquer instante (usando o *slider* ou os botoes +/-), mudar o incremento mínimo do passo do PWM (de 1% a 10%) ou ainda ativar um modo de demonstração” no qual o PWM sobe e desce periodicamente (de acordo com o passo mínimo ajustável). Essas operações demonstram controle total sobre a intensidade gerada com o PWM.

Na segunda parte, “Ciclo circadiano”, é apresentada uma simulação do ciclo de intensidade luminosa do sol. Podemos escolher a porcentagem de tempo de cada uma das 4 etapas : amanhecer, dia, anoitecer e noite assim como o tempo total. O *offset* é opcional e permite iniciar o ciclo após x minutos de espera. Por exemplo, num ciclo de 1440 minutos (1 dia) se desejarmos iniciar o amanhecer em 6 horas basta colocar 1440 em tempo total e 360 no *offset*. O programa pode ser interrompido a qualquer momento. O programa foi feito com resolução mínima de 1 segundo. Ele calcula, de acordo com os parâmetros inseridos, a quantidade de tempo que deve permanecer em cada etapa e o passo mínimo do brilho. Assim numa simulação curta podemos ter um passo de brilho grande e uma transição pouco suave de dia/noite/dia. Com ao menos 30 minutos de tempo total do ciclo a resolução já cai para 1% da intensidade, realizando transições suaves.

3.2.4 Driver de corrente

Até este ponto estava em funcionamento o PWM, a comunicação serial (ambos no MSP430) e o programa Java que se comunicava com o microcontrolador. Dessa maneira era possível enviar comandos pelo computador e dimerizar o LED SMD na placa Launchpad. Nosso objetivo é dimerizar uma carga de leds que utilizará uma corrente de 440 miliampères. Para tal é necessário o uso de uma fonte externa e, conforme descrito anteriormente, um circuito eletrônico com transistores em configuração de Sziklai. Na Figura 11 temos o esquemático referido:

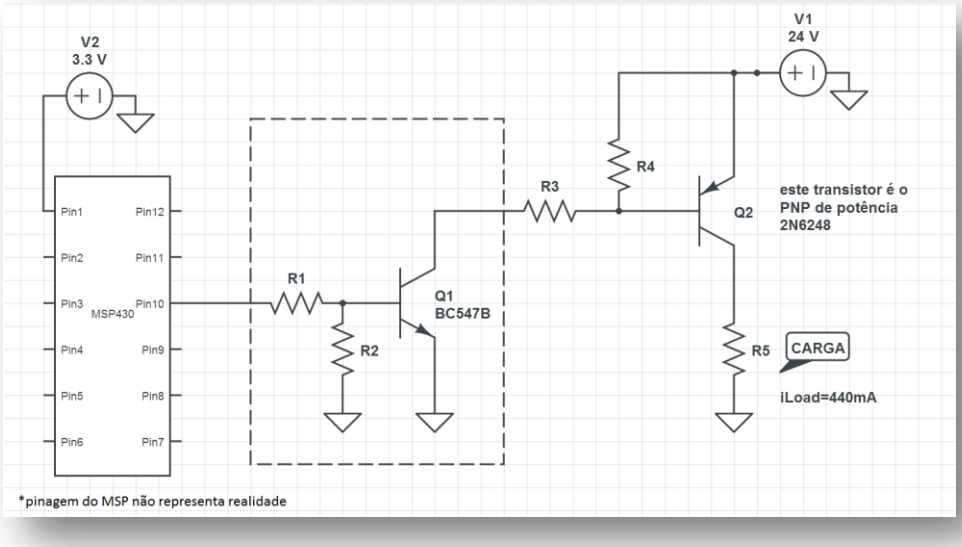


Figura 11 - Esquemático do *driver* de corrente.

Destaque para o primeiro módulo: transistor NPN em modo emissor comum

Uma pequena corrente proveniente do MSP430 entrando na base de Q1 faz o transistor conduzir no modo saturado: corrente irá fluir de seu coletor (ligado em R3) para o emissor (ligado ao terra). Essa corrente saindo da base de Q2 irá abaixar o potencial ali para próximo de terra (antes era 24V) e mudar este transistor também para o modo saturado no qual a corrente flui de seu emissor (24V) para a carga.

Para calcular os valores de R1, R2, R3 e R4 de maneira a garantir que os transistores trabalhem numa faixa normal de operação e forneçam a corrente desejada a carga precisamos dos parâmetros: Ic max – corrente máxima de coletor, Vce – tensão máxima coletor-emissor, Vce sat - tensão coletor-emissor na saturação, Vbe – tensão base emissor, Hfe – ganho. Os dados estão dispostos na Figura 12 para os transistores BC547B e 2n6248

	BC547B	2n6248
tipo	NPN	PNP
Vce (max)	45V	-100V
Ic	100mA	-15A
Vce(sat)	0,3V	-1,3V
Vbe	1V	-1,8V
hFE	200	100

Figura 12 - Parâmetros de funcionamento dos transistores usados: BC547B e 2N6248

Será iniciado pelo cálculo da corrente no PNP Q2.

Para meio ampère passando na carga e considerando Ic = Ie

$$Ic = Ib \times h_{FE} \qquad [I]$$

$$Ic = Ie = 0.5\ A \qquad h_{FE} = 100 \qquad => \quad Ib = 5\ mA$$

Usou-se Ib = 50 mA que o driver seja capaz de conduzir ate 5A na carga.

A questão é, se o PNP está com corrente suficiente na base e saturado ele passa a atuar quase que como um curto circuito (pois Vce=1,3V é muito menor que os 24V da fonte) e a carga passa a limitar sua própria corrente. Com Ib=5mA será chaveada uma corrente de ate meio ampère,

conforme calculado. Ainda que a carga tentasse solicitar mais corrente, por exemplo, 1 ampère, o transistor passaria a limitar a corrente de acordo com a fórmula (I) Ic = Ib x h_{FE}. Com Ib = 50mA a carga é chaveada corretamente e drena meio ampere da fonte com possibilidade de uma carga maior que drene até 5 amperes.

Com a corrente passando por R3 (chamada de I3) igual a 50mA pode-se agora calcular o valor do resistor R3. Desconsiderando por agora o resistor R4 para simplificar as contas e utilizando a malha fechada desde a fonte de 24V até o terra passando por Q2, R3 e Q1 temos:

$$24 - Vbe(Q2) - Vce(Q1) = 21{,}9\ V \quad \text{[que é a queda de tensão no R3]}$$

Ainda,

$$\frac{V}{I} = R \qquad \text{[II]}$$

$$R3 = \frac{21{,}9}{0{,}050} = 438\ \Omega \text{ [o valor comercial mais próximo é 470 Ω]}$$

Agora em Q1 o calculo da corrente da base novamente para chavear no mínimo 50 mA entre coletor-emissor. Usando novamente a fórmula [I] e considerando Ic=Ie:

$$Ic = Ib \times h_{FE}$$

$$Ic = Ie = 0{,}050\ A \qquad hFE = 200 \qquad => \quad Ib = 0{,}25\ mA$$

Que é uma corrente de base extremamente pequena. Será dobrada para 0,5 mA e considerando que próximo do Ic máximo do transistor o ganho diminui Q1 poderá chavear até quase 100 mA. Como R3 estará limitando a corrente em 50 mA não haverá problemas em chegar perto desse limite.

Mais uma vez o cálculo da malha fechada desde a saída do MSP até o terra, passando por R1:

$$3.3V - Vbe(Q1) = 2{,}3\ V \qquad \text{[queda de tensão em R1]}$$

E mais uma vez por [II] $\frac{V}{I} = R$

$$R1 = \frac{2{,}3}{0{,}0005} = 4600\ \Omega \qquad \text{[vamos utilizar 4k7 Ω].}$$

Os resistores R2 e R4 servem para garantir estabilidade do sistema em situação de alta impedância da entrada. R2 irá manter a base de Q1 em terra enquanto o MSP estiver inicializando. R4 irá manter a base de Q2 em 24V quando Q1 estiver cortado. Dessa maneira será evitado que a carga receba alguma corrente indesejada. Para esses valores um fator de 10 vezes R1 para R2 e 10 vezes R3 para R4 é suficiente. Assim:

$$R2 = 50k\ \Omega$$
$$R4 = 5k\ \Omega$$

e

$$R1 = 5k\ \Omega$$
$$R3 = 470\ \Omega$$

Isso encerra os cálculos para o driver de corrente. Um sinal saindo do MSP430 com uma corrente de 0,5mA poderá, através desse circuito, chavear e fornecer corrente de até 5 A – desde que a fonte externa permita – caracterizando um ganho de 10 mil vezes. Vale lembrar que existem no mercado componentes prontos que tem ganho maior que mil e seriam suficientes para a aplicação proposta. Para efeito prático será usado uma fonte de 1 A da qual será drenado apenas 440mA usando uma tira de LEDs de potência.

A seguir o esquemático projetado no *software* EAGLE (Figura 13), o layout da placa gerada para corrosão manual ou produção em maior escala (Figura 14), uma imagem gerada com o *software* EAGLE3D (Figura15) e uma foto do circuito soldado numa placa padrão (furada) (Figura 16):

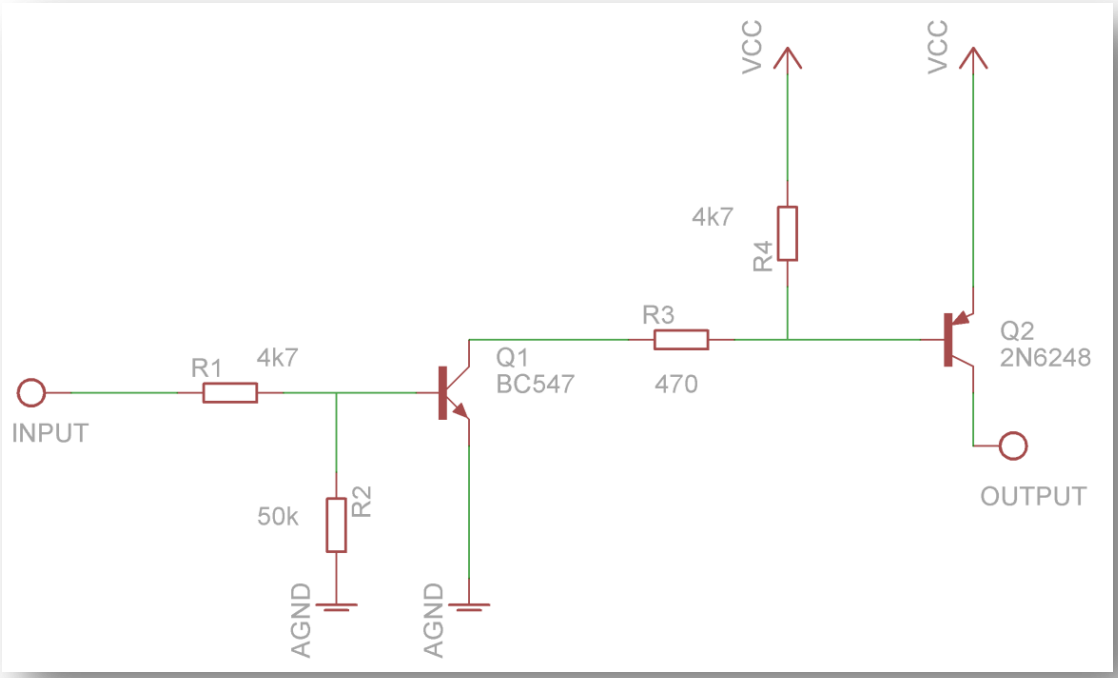


Figura 13 – Esquemático do *driver* de corrente projetado no *software* EAGLE

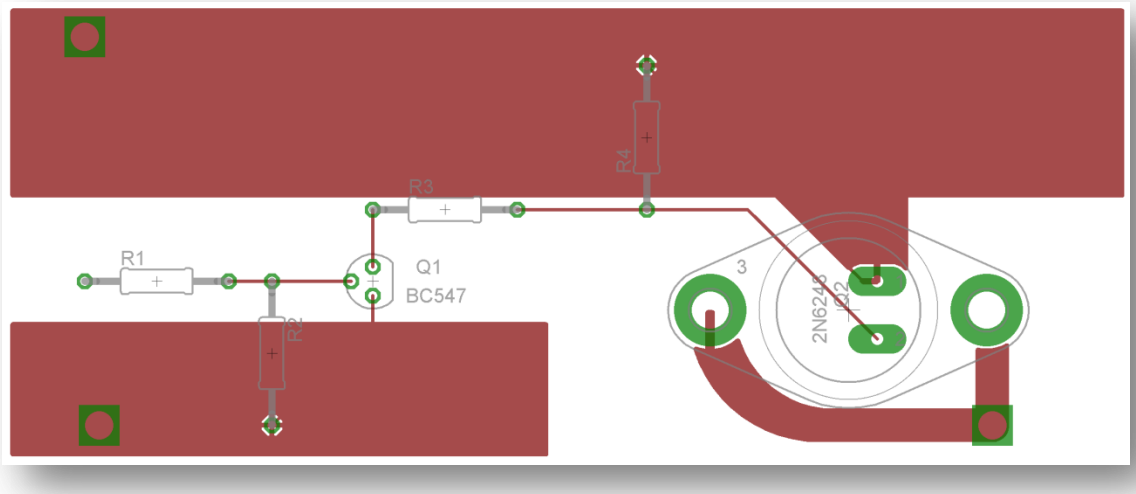


Figura 14 - Layout do *driver* de corrente para corrosão gerado no *sofware* EAGLE

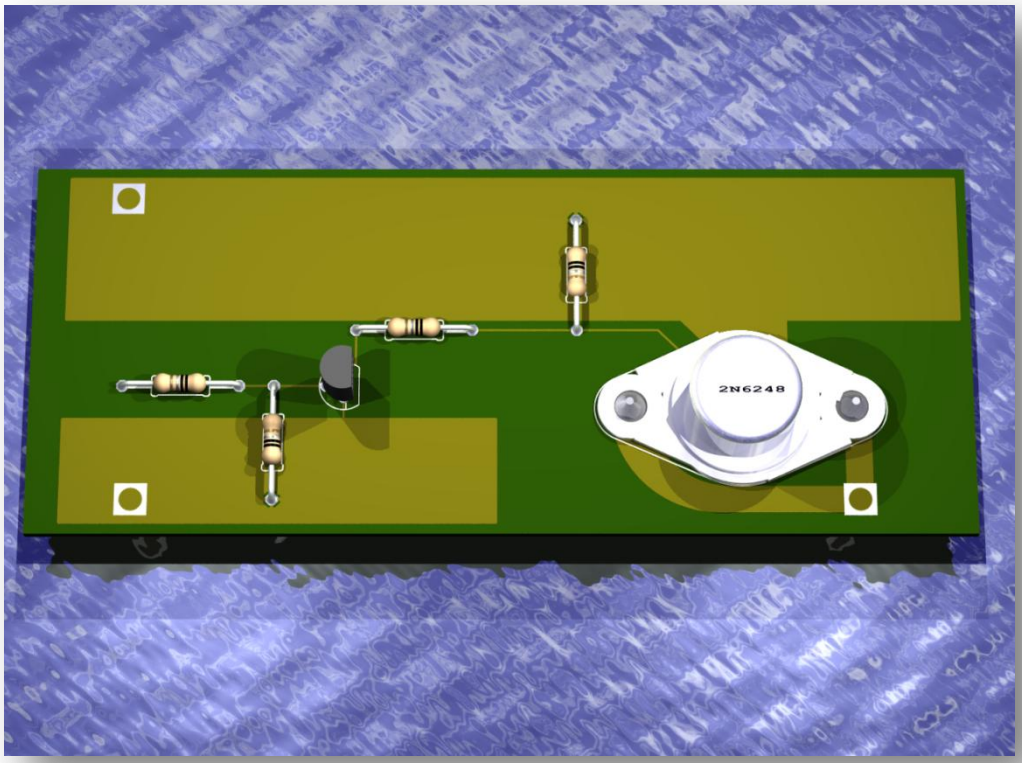


Figura 15 - Representação 3D do layout. Gerado no *software* EAGLE 3D

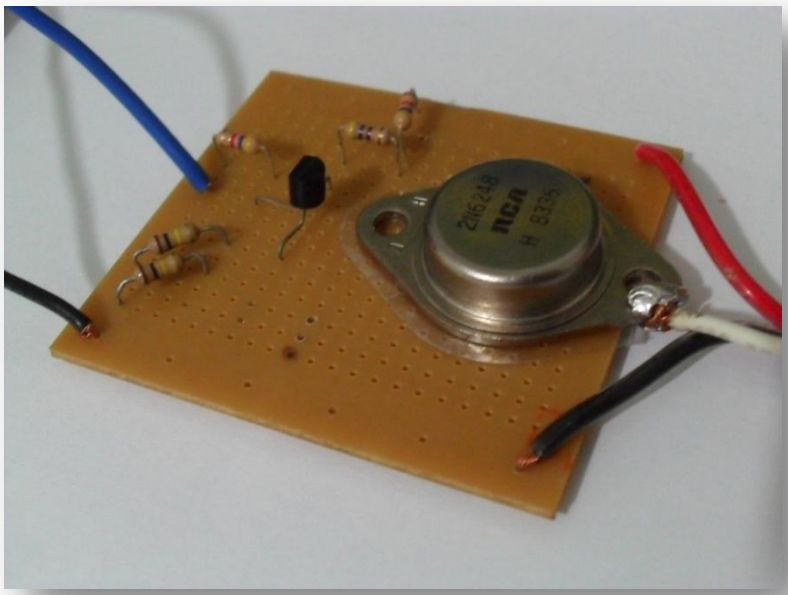


Figura 16 - Foto do driver projetado soldado em uma placa padrão

3.2.5 Bluetooth

O **Bluetooth** foi o último módulo. Ele adiciona uma funcionalidade de alto valor agregado e comodidade eliminando a necessidade de fios para conectar o computador a placa e possibilitando operação a uma distancia de até aproximadamente 10 metros.

O módulo escolhido foi com o chip CSR BC 417 que contém uma interface serial, um adaptador **Bluetooth** e *firmware* HC06 (também conhecido como "linvor 1.5"). Trabalha na faixa de frequência 2,4 – 2,48 GHz ISM. Por padrão opera no modo *slave*, interface serial com baudrate de 9600, 8 bits de dados, 1 stop bit e sem paridade.

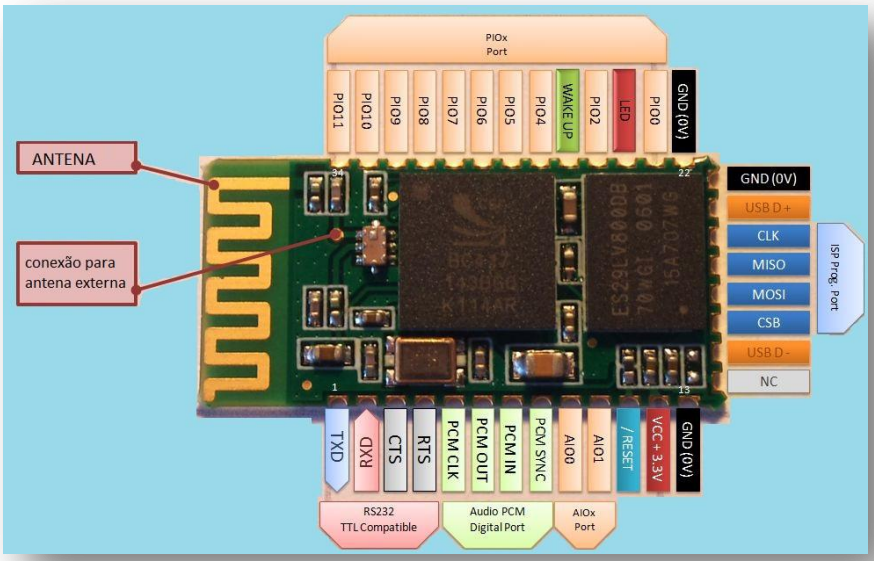


Figura 17 – Pinagem de entrada e saída do módulo Bluetooth e sua função

Esse *firmware* é conhecido por ser o mais simples da família (HC-03, HC-04, HC-05, HC-06) com poucos comandos AT e aceitar apenas o modo *slave*. O módulo foi escolhido pelo baixo custo. Para nosso objetivo de transmissão sem fio já é o suficiente. Os comandos AT disponíveis estão listados na Figura a seguir:

COMANDO	RESPOSTA	NOTA
AT	OK	útil para checar a conexão
AT+VERSION	Linvor1.5	retorna a versão do módulo
AT+BAUDx	OKyyyy	seta o baudrate. x pode assumir estes valores: 1 para 1200 bps 2 para 2400 bps 3 para 4800 bps 4 para 9600 bps 5 para 19200 bps 6 para 38400 bps 7 para 57600 bps 8 para 115200 bps 9 para 230400 bps A para 460800 bps B para 921600 bps C para 1382400 bps
AT+NAMEnome	OKnome	muda o nome do dispositivo (até 20 caracteres)
AT+pinxxxx	OKsenha	muda a senha de pareamento (1234 é o padrão)
AT+PN		muda para sem paridade (padrão)
AT+PE		muda para paridade par
AT+PO		muda para paridade impar

Figura 18 - Comandos AT disponíveis para o módulo Bluetooth

Para testes foi usado o programa de eco pela comunicação serial half-duplex no MSP430G2231 e o programa Java TwoWaySerialComm. Após a gravação do programa na flash do MSP os fios TX e RX foram desconectados da usb – a Launchpad oferece jumpers para tal, para serem ligados ao RX e TX (propositalmente trocados) do **Bluetooth**. O Launchpad também forneceu alimentação de 3.3V e o terra para o módulo **Bluetooth** (a visualização dessa conexão poderá ser vista no próximo item: integração dos módulos. Com apenas 4 fios ligados o módulo ja estava pronto para uso.

Após o pareamento do **Bluetooth** com o computador o TwoWaySerialComm permite enviar um caracter pelo terminal Java, esse byte é transmitido por uma porta serial COM e pelo **Bluetooth** do computador. O módulo **Bluetooth** recebe o byte e transmite serialmente para o MSP430. Nele o *software* recebe serialmente o byte e o envia de volta pelo mesmo canal, fazendo agora o caminho contrário. O terminal Java no computador exibe tanto o caracter enviado quanto o mesmo caracter respondido confirmando a transmissão. O canal serial feito pelo **Bluetooth** é transparente para o programa Java e para o MSP assim como o canal serial do MSP430 Launchpad emulado pelo USB.

3.3 Integração dos módulos

A integração dos módulos é simples, bastando apenas conectar os fios RX e TX da UART e os fios de sinais PWM – haja vista que foram projetados para trabalhar juntos. Uma visão completa

do esquemático final (Figura 19) bem como uma foto do sistema pronto (Figura 20) podem ser visualizados abaixo:

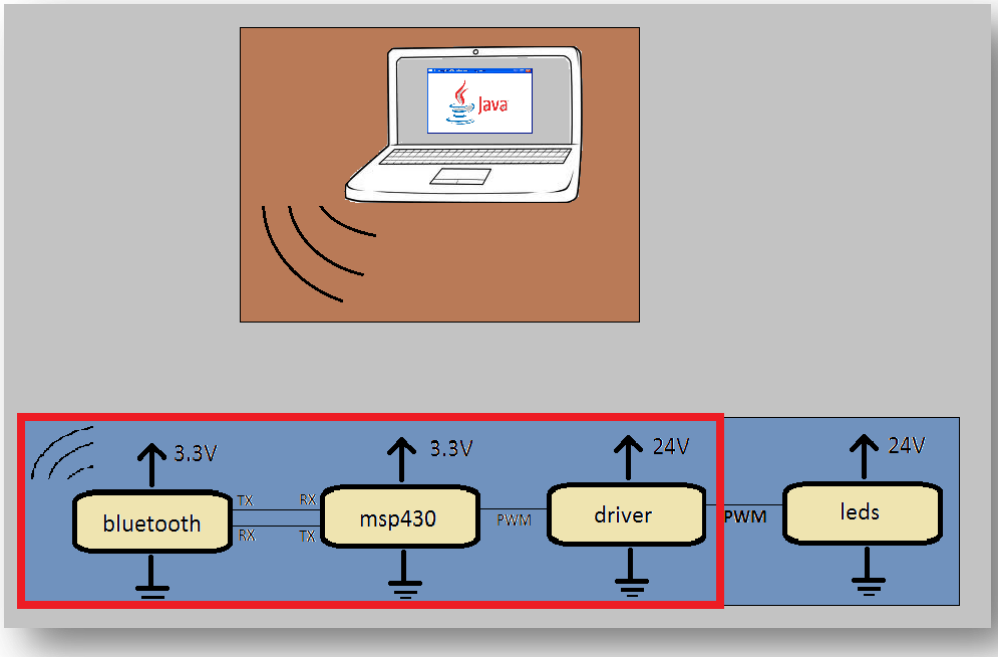


Figura 19 - Esquemático com destaque para o *hardware* implementado

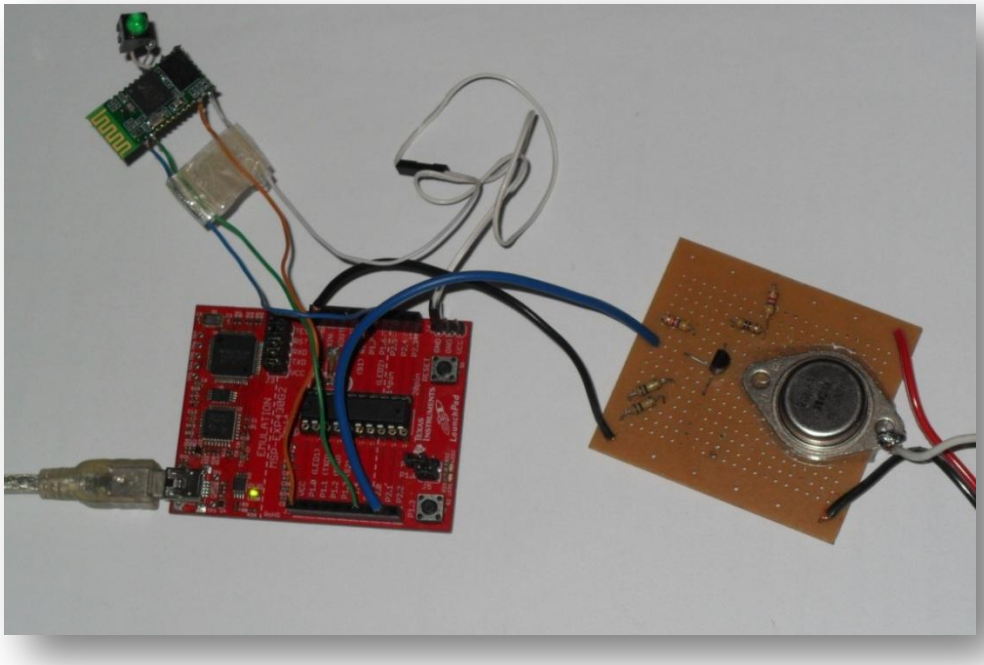


Figura 20 – *Hardware* completo e interligado

Na Figura 20 temos o MSP fornecendo 3.3V para o modulo **Bluetooth** (fio laranja). Os fios vermelho e preto entrando pela direita no driver de corrente são provenientes da fonte externa de 24V ligada a rede elétrica. Todos os aterramentos estão interligados (fios preto e branco). Fios verde e azul: TX e RX do **Bluetooth**.

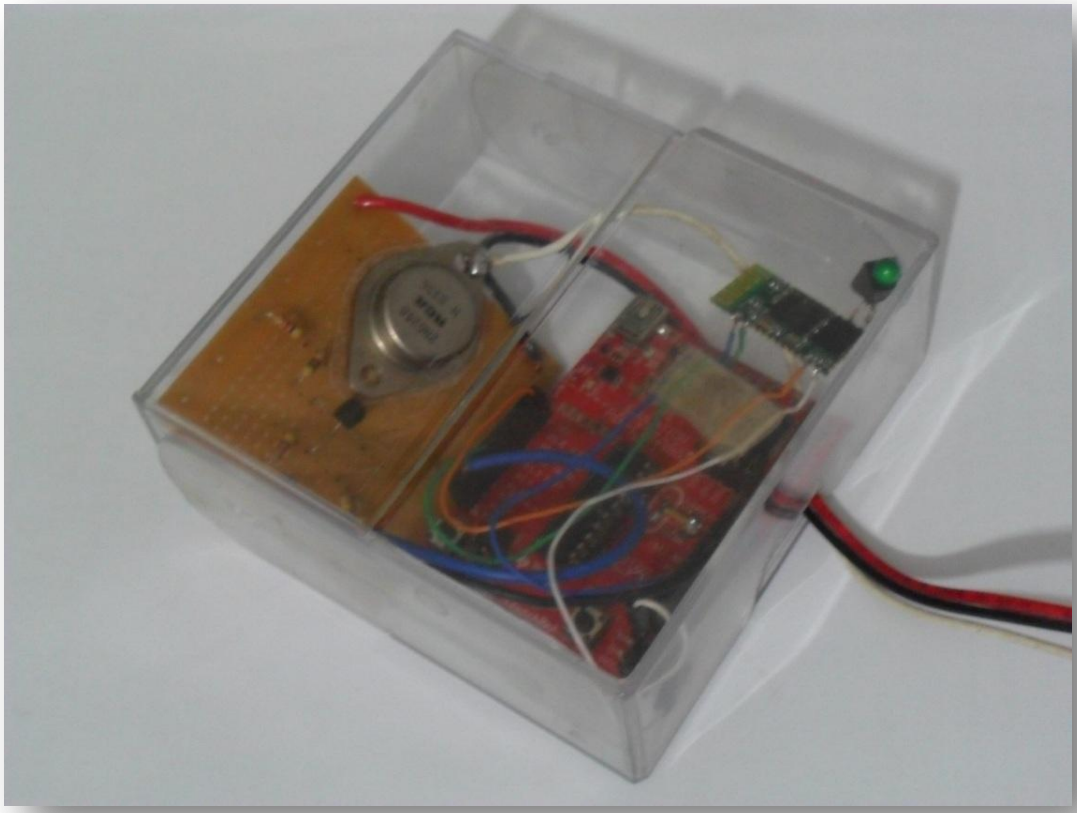


Figura 21 - Sistema final dentro da caixa

Na Figura 21 podem ser visualizados o MSP, o driver e o módulo **Bluetooth** (em verde) dentro da caixa plástica.

4. Funcionamento

Para a implementação das operações básicas do PWM (variar intensidade, entre outras) foi necessário criar um protocolo entre o programa C (sendo executado no MSP430) e o programa JAVA. O protocolo é feito transmitindo bytes especiais que o MSP irá interpretar como comandos. O MSP pode retornar o comando enviado ou o valor da intensidade de brilho atual para o JAVA manter o estado correto de sincronia entre eles.

Os bytes especiais usados foram:

- DEMO = 170; //liga / desliga o modo de demonstração
- INC = 171; //incrementa o brilho atual em um "step"
- DEC = 172; // decrementa o brilho atual em um "step"
- STEP = 173; //*seta* um novo valor para o passo step (passa parâmetro)
- FORCE = 174; //força um novo valor para o brilho (passa parâmetro)
- UPDATE = 175; //chama o metodo update() do MSP que atualiza o brilho
- RETINT = 176; //retorna a intensidade atual do MSP

O programa JAVA tem esses valores como constantes definidas na classe Connection. O MSP recebe o valor e dentro do loop da função main interpreta o comando, executa, retorna algo pela serial (quando tem que retornar) ou apenas aguarda por um valor quando o comando enviado precisa passar parâmetro.
Comando STEP: após recebê-lo o MSP aguarda por um valor entre 1 e 10 para atualizar o valor do passo de incremento/decremento da intensidade.
Comando FORCE: após recebê-lo o MSP aguarda por um valor entre 0 e 100 para atualizar o valor da intensidade (em porcentagem).

5. Conclusões

Os módulos funcionam e a proposta foi cumprida. O ciclo circadiano tem conhecida importância para os seres vivos e a simulação do fotoperíodo aqui apresentada pode, conforme mencionado anteriormente, servir como ferramenta útil para pesquisas em plantas (sobre a fotossíntese, variando a intensidade luminosa ao longo do ciclo ou outra métrica) e em animais (comportamento dos animais influenciado por variações na luz ou mesmo em humanos e a influência da luz no sono, por exemplo).

Sobre os módulos desenvolvidos o programa Java Connection executa comunicação serial genérica e reaproveitável. O programa UART halfDuplex do MSP também é bem modularizado e pode ser aproveitado. A comunicação **Bluetooth** pode ser usada em qualquer projeto se consideradas as limitações de alcance e *throughput*. A experiência projetando e implementando o circuito do driver é valiosa.

Existem alternativas comerciais para dimerizar LEDS a partir de aproximadamente 30 reais até algumas centenas de reais (dezembro/2012). Apesar dos custos apresentados estarem no valor mínimo desse intervalo poderíamos reduzi-lo ainda mais com uma eletrônica mais simples (sem microcontrolador e **Bluetooth**, gerando PWM com CI 555, por exemplo) mas o foco deste projeto foram as funcionalidades desenvolvidas e a possibilidade de realizar o comando remotamente pelo computador abrindo possibilidades para diferentes programas de controle conforme necessidade e a comodidade de uso com celulares, *tablets* e similares.

Apesar do projeto ser um projeto simples, a utilidade e o caráter interdisciplinar dos módulos motivou-me durante sua execução e tornou o resultado deste trabalho ainda mais satisfatório.

Anexos

ANEXO 1

Programa MSP430 UART_TXTest:

```
#include  "msp430g2553.h"

#define     TXD           BIT6// TXD on P1.1
#define     Bitime        104//9600 Baud, SMCLK=1MHz (1MHz/9600)=104
unsigned char BitCnt; // Bit count, used when transmitting byte
unsigned int TXByte; // Value sent over UART when Transmit() is called

// Function Definitions
void Transmit(void);

void main(void) {
        WDTCTL = WDTPW + WDTHOLD; // Stop WDT

        unsigned int uartUpdateTimer = 10; // Loops until byte is sent
        unsigned int i = 0; // Transmit value counter

        BCSCTL1 = CALBC1_1MHZ; // Set range
        DCOCTL = CALDCO_1MHZ; // SMCLK = DCO = 1MHz

        P1SEL |= TXD; //
        P1DIR |= TXD; //

        __bis_SR_register(GIE);
        // interrupts enabled
        /* Main Application Loop */
        while (1) {
                if ((--uartUpdateTimer == 0)) {
                        TXByte = i;
                        Transmit();
                        //delay para ver os caracteres enviados no terminal do pc
                        __delay_cycles(80000);

                        i++;
                        uartUpdateTimer = 10;
                }
        }
}

// Function Transmits Character from TXByte
void Transmit() {
        CCTL0 = OUT; // TXD Idle as Mark
        TACTL = TASSEL_2 + MC_2; // SMCLK, continuous mode

        BitCnt = 0xA; // Load Bit counter, 8 bits + ST/SP
        CCR0 = TAR;

        CCR0 += Bitime; // Set time till first bit
        TXByte |= 0x100; // Add stop bit to TXByte (which is logical 1)
        TXByte = TXByte << 1; // Add start bit (which is logical 0)

        CCTL0 = CCIS0 + OUTMOD0 + CCIE; // Set signal, intial value, enable interrupts
        while (CCTL0 & CCIE)
                ; // Wait for TX completion     bad programming practice!
        TACTL = TASSEL_2; // SMCLK, timer off (for power consumption)
}

// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A(void) {
        CCR0 += Bitime; // Add Offset to CCR0
        if (BitCnt == 0) // If all bits TXed, disable interrupt
                CCTL0 &= ~CCIE;
        else {
```

```
                CCTL0 |= OUTMOD2; // TX Space
                if (TXByte & 0x01)
                        CCTL0 &= ~OUTMOD2; // TX Mark
                TXByte = TXByte >> 1;
                BitCnt--;
        }
}
```

ANEXO 2

Programa MSP430 UartPwmFinal:

```
/*****************************************************************************
 *                  MSP-EXP430G2-LaunchPad User Experience Application
 *
 * 1. Pisca o led P2.1 usando a PWM do TIMER0_A3
 * 2. Comunica pela UART modo half duplex (Timer1_A3)
 * 3. Baseado no PwmUartHalfDuplex
 * 4. Usa o WatchDogTimer para o modo Demonstração (PWM up/down)
 *
 *
 * Andre VC novembro/2012
 *****************************************************************************/

#include  "msp430g2553.h"
#include  "stdbool.h"

#define     PWM         BIT1    // signal output on P2.1
#define     LEDV        BIT0    // led VERMELHO
#define     BUTTON      BIT3    // botao
#define     TXD         BIT1    // TXD on P1.5
#define     RXD         BIT2    // RXD on P1.6
#define     Bit_time    104     // 9600 Baud, SMCLK=1MHz (1MHz/9600)=104
#define     Bit_time_5  52      // Time for half a bit.
#define     periodo     11000

unsigned char BitCnt; // Bit count, used when transmitting byte
unsigned int TXByte; // Value sent over UART when Transmit() is called
unsigned int RXByte; // Value received once hasRecieved is set
bool isReceiving; // Status for when the device is receiving
bool hasReceived; // Lets the program know when a byte is received

unsigned int data;
unsigned int param;
bool hasParam; //used to receive 2-bytes instructions
bool demoMode;
int upMode = 1; //pwm mode 1=increasing 0=decreasing
int atual = 2200;
int step = 110;

// Function Definitions
void Transmit(void);
void inline parseIt(unsigned int p, unsigned int d);
void inline up();
void inline down();
void inline update();
void inline toggleDemo();

void main(void) {

        WDTCTL = WDTPW + WDTHOLD; // Stop WDT

        BCSCTL1 = CALBC1_1MHZ; // Set range
        DCOCTL = CALDCO_1MHZ; // SMCLK = DCO = 1MHz

        //PWM CONFIG
        TA1CCR0 = periodo; // PWM Period
```

```
TA1CCR1 = atual; // CCR1 PWM duty cycle
TA1CCTL1 = OUTMOD_7; // CCR1 reset/set
TA1CTL = TASSEL_2 + MC_1; // SMCLK, up mode

P2DIR |= PWM;
P2SEL |= PWM; // P2.1 to TA1.1 PWM out
P2SEL2 &= ~PWM; // P2.1 to TA1.1

P1DIR |= LEDV;
P1OUT &= ~LEDV;

P1DIR &= ~BUTTON;
P1OUT |= BUTTON;
P1REN |= BUTTON;
P1IES |= BUTTON;
P1IFG &= ~BUTTON;
P1IE |= BUTTON;

//UART CONFIG
P1SEL |= TXD;
P1DIR |= TXD;

P1IES |= RXD; // RXD Hi/lo edge interrupt
P1IFG &= ~RXD; // Clear RXD (flag) before enabling interrupt
P1IE |= RXD; // Enable RXD interrupt

isReceiving = false;
hasReceived = false;
hasParam = false;
demoMode = false;

__bis_SR_register(GIE);
// interrupts enabled

while (1) {
        if (hasReceived) {
                hasReceived = false;
                data = RXByte;
                // Load the received byte into the register TX to be transmitted
                //TXByte = RXByte;

                if (hasParam) {
                        //parse the param value
                        hasParam = false;
                        parseIt(param, data);

                } else {
                        switch (data) {

                        case 170:
                                toggleDemo();
                                break;
                        case 171:
                                upMode = 1;
                                up();
                                break;
                        case 172:
                                upMode = 0;
                                down();
                                break;
                        case 173:
                        case 174:
                        case 175:
                                hasParam = true;
                                param = data;
                                break;
                        default:
                                break;
                        }

                }
                TXByte = atual / 10; //load intensity to be transmitted (in %)
                Transmit(); //transmit the value stored in TXByte
        }
```

```
                if (~hasReceived) // Loop again if another value has been received
                        __bis_SR_register(CPUOFF + GIE);
        }
}
/* ********************************************************
 * Function Transmits Character from TXByte
 * ******************************************************** */
void Transmit() {
        // Wait for RX completion
        while (isReceiving)
                ;

        CCTL0 = OUT; // TXD Idle as Mark
        TACTL = TASSEL_2 + MC_2; // SMCLK, continuous mode

        BitCnt = 0xA; // Load Bit counter, 8 bits + ST/SP
        CCR0 = TAR; // Initialize compare register

        CCR0 += Bit_time; // Set time till first bit
        TXByte |= 0x100; // Add stop bit to TXByte (which is logical 1)
        TXByte = TXByte << 1; // Add start bit (which is logical 0)

        CCTL0 = CCIS0 + OUTMOD0 + CCIE; // Set signal, intial value, enable interrupts

        // Wait for previous TX completion
        while (CCTL0 & CCIE)
                ;
}

/* ********************************************************
 * Port 1 interrupt service routine
 * ******************************************************** */
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void) {

        if ((P1IFG & BIT3) != 0) { // SE BOTAO PRESSIONADO
                P1IFG &= ~BIT3; // P1.3 IFG cleared
                //liga/desliga o modo Demonstração
                toggleDemo();
        } else {
                isReceiving = true;

                P1IE &= ~RXD; // Disable RXD interrupt
                P1IFG &= ~RXD; // Clear RXD IFG (interrupt flag)

                TACTL = TASSEL_2 + MC_2; // SMCLK, continuous mode
                CCR0 = TAR; // Initialize compare register
                CCR0 += Bit_time_5; // Set time till first bit
                CCTL0 = OUTMOD1 + CCIE; // Dissable TX and enable interrupts

                RXByte = 0; // Initialize RXByte
                BitCnt = 0x9; // Load Bit counter, 8 bits + ST
        }
}

/* ********************************************************
 * Timer A0 interrupt service routine
 * ******************************************************** */
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A(void) {

        if (!isReceiving) {
                CCR0 += Bit_time; // Add Offset to CCR0
                if (BitCnt == 0) // If all bits TXed
                                {
                        TACTL = TASSEL_2; // SMCLK, timer off (for power consumption)
                        CCTL0 &= ~CCIE; // Disable interrupt
                } else {
                        CCTL0 |= OUTMOD2; // Set TX bit to 0
                        if (TXByte & 0x01)
                                CCTL0 &= ~OUTMOD2; // If it should be 1, set it to 1
                        TXByte = TXByte >> 1;
                        BitCnt--;
                }
```

```
        } else {
                CCR0 += Bit_time; // Add Offset to CCR0
                if (BitCnt == 0) {
                        TACTL = TASSEL_2; // SMCLK, timer off (for power consumption)
                        CCTL0 &= ~CCIE; // Disable interrupt

                        isReceiving = false;

                        P1IFG &= ~RXD; // clear RXD IFG (interrupt flag)
                        P1IE |= RXD; // enabled RXD interrupt

                        if ((RXByte & 0x201) == 0x200) // Validate the start and stop bits are
correct
                                        {
                                RXByte = RXByte >> 1; // Remove start bit
                                RXByte &= 0xFF; // Remove stop bit
                                hasReceived = true;
                        }
                        __bic_SR_register_on_exit(CPUOFF);
                        // Enable CPU so the main while loop continues
                } else {
                        if ((P1IN & RXD) == RXD) // If bit is set?
                                RXByte |= 0x400; // Set the value in the RXByte
                        RXByte = RXByte >> 1; // Shift the bits down
                        BitCnt--;
                }
        }
}
/* ************************************************************
 *  Watchdog Interval Timer interrupt service
 * ************************************************************ */
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void) {
        update();
}

void inline update() {

        if (upMode == 1) {
                if (atual < (periodo - step)) {
                        atual += step;
                } else {
                        upMode = 0;
                        atual = periodo;
                }
        } else {
                if (atual > step) {
                        atual -= step;
                } else {
                        upMode = 1;
                        atual = 0;
                }
        }
        TA1CCR1 = atual;
}

void inline up() {
        if (atual < (periodo - step)) {
                upMode = 1;
                atual += step;
        } else {
                atual = periodo;
                upMode = 0;
        }
        TA1CCR1 = atual;
}

void inline down() {
        if (atual > step) {
                upMode = 0;
                atual -= step;
        } else {
                atual = 0;
        }
```

```
        TA1CCR1 = atual;
}
void inline parseIt(unsigned int p, unsigned int d) {
        switch (p) {
        case 173: { //set STEP
                if ((d > 0) && (d < 11)) { //1% <= STEP valido <= 10%
                        step = d * 10;
                }
                break;
        }
        case 174: { //FORCE PWM
                atual = d * 4;
                TA1CCR1 = atual;
                break;
        }
        case 175: { //UPDATE
                update();
                break;
        }
        }
}
void inline toggleDemo() {

        if (demoMode) {
                demoMode = false;
                WDTCTL = WDTPW + WDTHOLD; // Stop WDT

        } else {
                demoMode = true;
                WDTCTL = WDT_MDLY_32; // WDT as interval timer (period 0,5 ms)
                IE1 |= WDTIE; // Enable WDT interrupt
        }

}
```

ANEXO 3

Programa RXTXCOMM

Connection.java:

```
package tcc;

import gnu.io.CommPortIdentifier;
import gnu.io.PortInUseException;
import gnu.io.RXTXPort;
import gnu.io.UnsupportedCommOperationException;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Enumeration;
import java.util.List;

public class Connection {

private final int BAUD = 9600;
private String port;
private RXTXPort rxtxPort = null;
private List<CommPortIdentifier> ports;
private List<String> serialPortNames;
private boolean connected = false;
private OutputStream out = null;
private InputStream in = null;
public  final static int DEMO = 170; // 100; //toggles demonstration mode
public  final static int INC = 171;// 97; //increment one step
public  final static int DEC = 172;// 122; //decrement one step
```

```
public  final static int STEP = 173;// 115; //set step x mode
public  final static int FORCE = 174;// 102; //force PWM x mode
public  final static int UPDATE = 175;// 117; //force update
public  final static int RETINT = 176;// 114; //return current intensity

        boolean connect(String portName) {
                try {
                        port = portName;
                        rxtxPort = new RXTXPort(port); // PortInUseException
                        rxtxPort.setSerialPortParams(BAUD, RXTXPort.DATABITS_8,
RXTXPort.STOPBITS_1, RXTXPort.PARITY_NONE);
                        System.out.println("connected to " + port);
                        connected = true;
                        out = rxtxPort.getOutputStream();
                        in = rxtxPort.getInputStream();
                        return true;

                } catch (PortInUseException e) {
                        System.out.println("Erro ao abrir a porta " + port);
                } catch (UnsupportedCommOperationException e) {
                        e.printStackTrace();
                }
                return false;
        }

public  void disconnect() {
                rxtxPort.close();
                connected = false;
                System.out.println("disconnected from " + port);
        }

public  boolean isConnected() {
                return connected;
        }

public  void transmit(int a) {
                if (out != null) {
                        if ((a >= 0) && (a < 256)) {
                                try {
                                        out.write(a);
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }
                }

        }

public  OutputStream getOutPutStream() {
                return out;
        }

public  InputStream getInputStream() {
                return in;
        }
}
```

Glossário

(conforme ordem de aparecimento)

- LED – light emitting diode, diodo emissor de luz
- MSP430 – microcontrolador fabricado pela **Texas Instruments**
- Java – linguagem de programação orientada a objeto
- **Bluetooth** – padrão de tecnologia de troca de dados sem fio a curta distância
- PWM – *pulse width modulation*, modulação por largura de pulso
- Driver de corrente – circuito eletrônico que amplifica um pequeno sinal
- Dimerização – efeito de reduzir a intensidade luminosa
- Open source – programa de computador de código fonte aberto
- ARM – Advanced RISC Machine, arquitetura de processadores RISC
- CI – circuito integrado (eletrônica)
- **TI – Texas Instruments**
- USART – Universal Asynchronous Receiver/Transmitter, *hardware* que se comunica serialmente
- SPI – serial peripheral interface, barramento sincrono de comunicação serial full duplex
- I2C – inter-integrated circuit, barramento sincrono de comunicação serial
- ADC – analog-to-digital converter, conversor analógico/digital
- Datasheet – documento com as especificações técnicas, geralmente fornecido pelo fabricante
- Debug – processo de teste para procurar/retirar erros em um *software*
- Deploy – implantação do *software* onde ele será executado
- *Clock* – sinal em forma de onda quadrada que dita a velocidade que um circuito eletrônico irá trabalhar. Medido em hertz e seus múltiplos.
- *Timer* – periférico em um microcontrolador usando para contar tempo, gerar interrupções, PWMs, etc.
- Duty cycle – ciclo de trabalho, percentual de tempo que o PWM permanece em alta
- Diodo – componente eletrônico formado por uma junção PN em silício ou semicondutor semelhante
- Full-duplex – modo de transmissão serial no qual é possível enviar e receber ao mesmo tempo
- Half-duplex – modo de transmissão serial no qual é possível enviar e receber mas não concomitantemente
- ES – entrada e saída

- Baudrate – taxa de transmissão de dados
- WDT – watchdog *timer*, *timer* especial em microcontroladores usado para *resetar* o chip em caso de travamentos
- SFR – special funcion register, registradores de função especial em microcontroladores que mapeiam dispositivos ou outras funções especiais de configuração
- TACTL – SFR do MSP430: controle do *timer* A
- SWING – biblioteca gráfica para desenhar interface de usuário
- SMD – surface-mount device, dispositivos pequenos soldados sobre a placa de circuito impresso.
- Slave – modo de operação no qual o dispositivo após ligado espera por instruções de um dispositivo em modo mestre (master)

Referências bibliográficas

[1] Texas Instruments (2004). "MSP430x2xx Family User's Guide (Rev. I)" Literature Number: SLAU144I http://www.ti.com/lit/ug/slau144i/slau144i.pdf (acessado em setembro/2012)

[2] Prathyusha Narra and Zinger, D.S. (Oct 2004). "An effective LED dimming approach". *Industry Applications Conference, 2004. 39th IAS Annual Meeting*. 1671-1676

[3] Farrel *et al.*(1987). Farrell, J.E., Benson,B.L., Haynie, C.R., "PREDICTING FLICKER THRESHOLDS FOR VIDEO DISPLAY TERMINALS" *Stanford Center for Image System Engineering*,1987.
http://scien.stanford.edu/jfsite/Papers/ImageRendering/Displays/Farrell_Flicker_1987.pdf

(acessado em outubro/2012)

[4] Adam Osborne (1980). "An Introduction to Microcomputers Volume 1: Basic Concepts". *Osborne-McGraw Hill Berkeley California USA*, 116-126

[5] Jones, R.V. (2001). "Basic BJT Amplifier Configurations" Electronic Devices and Circuits

Engineering Sciences 154

http://people.seas.harvard.edu/~jones/es154/lectures/lecture_3/bjt_amps/bjt_amps.html

(acessado em novembro/2012)

[6] Dijk, Derk-Jan; Malcolm von Schantz (2005). "Timing and Consolidation of Human Sleep, Wakefulness, and Performance by a Symphony of Oscillators". J Biol Rhythms (SagePub) 20 (4): 279–290

- **Texas Instruments** (2012) "MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. G)" (agosto,2012)

http://www.ti.com/lit/ds/symlink/msp430g2253.pdf (acessado em setembro/2012)

- Microeletrônica – 5ª Edição Sedra, Adel S.