INSTITUTO NACIONAL DE TELECOMUNICAÇÕES - “INATEL”

RELATÓRIO SOBRE TRABALHO FINAL DE E209

LIGHT CONTROL

Equipe:

1. Alexander Augusto Silva Fernandes	Matrícula 1333	período: 5º
2. Leandro de Aquino Pereira	Matrícula 1382	período: 5º
3. Pedro Bonfilio Lima	Matrícula 620	período: 5º

Professor: Evandro

Monitor: Lucas

Junho / 2019

SUMÁRIO

1. Introdução

O presente relatório está inserido na proposta de projeto final da disciplina de Sistemas Microcontrolados e Microprocessados (E209), relacionando os conceitos vistos em sala de aula e praticados em laboratório utilizando o microcontrolador MSP430, expondo uma determinada aplicação, possibilitando, assim, seu uso no dia-a-dia.

Visando tal objetivo, o relatório expõe esse processo de desenvolvimento por meio da metodologia utilizada, o objetivo do sistema, a descrição do hardware e firmware, assim como sua análise por meio de testes realizados.

Com o auxílio da tecnologia, pode-se obter resultados melhores para questões que humanidade enfrenta constantemente, sendo uma delas, o desperdício de energia por meio da utilização da luz. De acordo com o relatório da Associação Brasileira das Empresas de Serviço de Conservação de Energia (Abesco), entre o mês de setembro de 2014 e setembro de 2017 o desperdício de energia custou R$ 61,7 bilhões para o Brasil. Uma forma de minimizar tal desperdício é por meio de sistemas automatizados que buscam uma melhor eficiência dos sistemas de iluminação.

Dada a proposta inicial do projeto, "Controle de Luminosidade Ambiente para Economia de Energia", desenvolveu-se o *Light Control* que sintetiza todas exigências, apresentando uma solução ao problema proposto.

1.1. Metodologia

A metodologia utilizada para desenvolver este projeto foi o da Engenharia que consiste em identificar um problema, identificar uma solução e desenvolver um protótipo para efetuar testes. Ao receber a proposta, foi identificado a necessidade de aplicar os conceitos apresentados na disciplina de Sistemas Microcontrolados e Microprocessados, e, com isso, concluiu-se que o *Light Control* atenderia todos requisitos do sistema.

Foram empregados diversos conceitos adquiridos até o presente período de graduação em Engenharia. Utilizou-se a implementação de circuitos eletrônicos com sensores, fundamentos da lógica de programação, o software de simulação NI Multisim para executar o esquema elétrico e, para finalizar, realizou-se testes verificando o funcionamento.

1.2. Objetivos

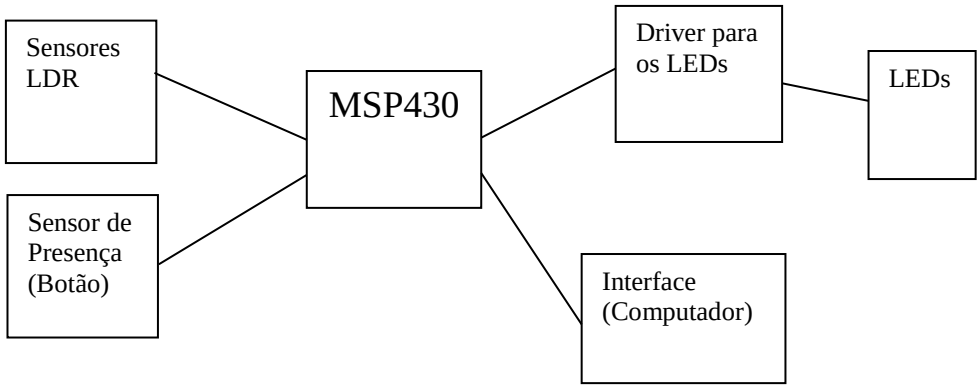
O objetivo deste projeto é desenvolver um protótipo que atenda as exigências do problema proposto, tendo em vista a eficiência do mesmo por meio da interação com o usuário e economia de energia. Posteriormente ao desenvolvimento, efetuou-se testes técnicos para comprovar sua solução.

Visando alcançar tal objetivo, fracionou-se o projeto em diferentes partes, sendo a primeira a alimentação do MSP430 e circuito externo com sensores. Na segunda parte, projetou-se e efetuou o desenvolvimento circuito externo com sensores. Logo em seguida, na terceira parte, desenvolveu-se a programação e efetuou-se testes por meio do circuito externo desenvolvido anteriormente.

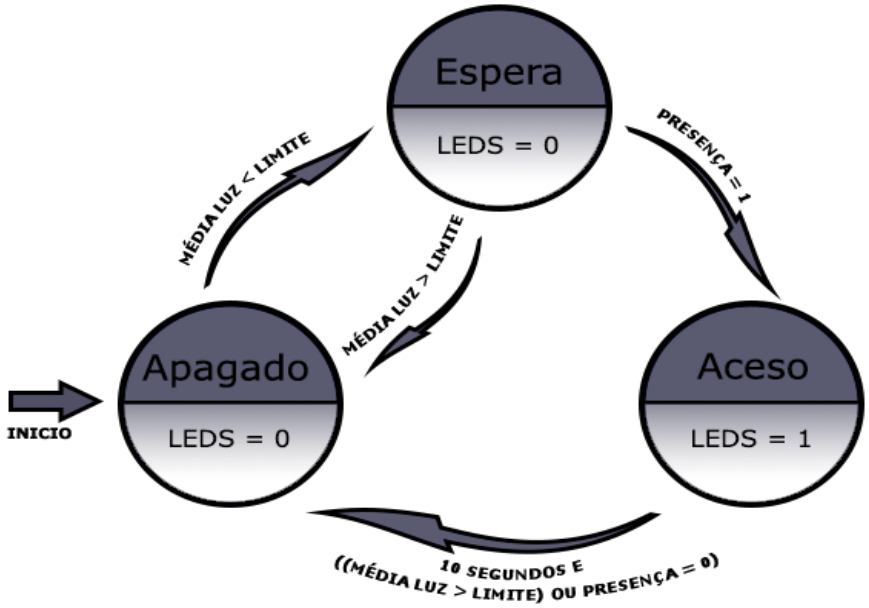
2. Descrição Geral

O *Light Control* foi desenvolvido visando a solução do problema proposto como projeto final da disciplina de Sistemas Microcontrolados e Microprocessados. Para a solução foram utilizados sensores LDR, driver, botão de acionamento e o MSP430G2553. O funcionamento do projeto é demonstrado no diagrama em blocos e na máquina de estados abaixo:

2.1. Diagrama em blocos:



2.2. Máquina de estados:



Os sensores LDR pegam a luminosidade do ambiente a fim de verificar se os LEDs devem ser acesos ou não. O botão é utilizado para simular um sensor de presença, ou seja, quando pressionado demonstra que uma pessoa chegou ao local e caso o sensor esteja identificando luz baixa no ambiente (o valor limite de luminosidade é inserido pelo usuário por meio de uma interface), os LEDs serão acesos. Como tem-se 3 LEDs para serem acesos, utilizou-se o driver para aumentar a corrente sobre eles e acender os 3 de uma só vez.

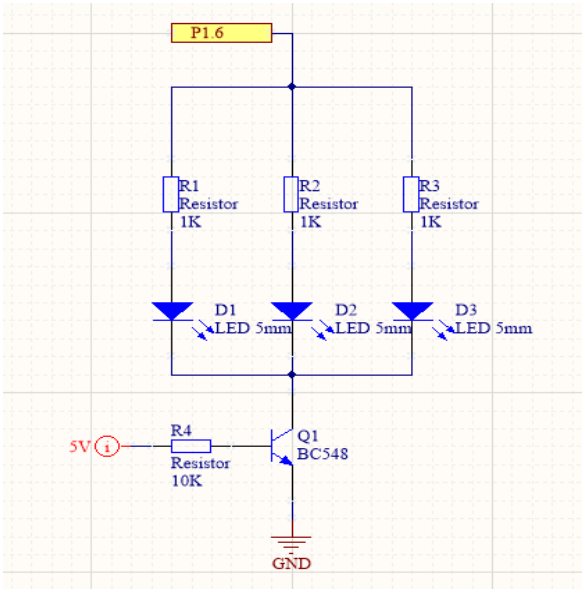
2.3. Hardware

O hardware do Light Control é composto pela placa padrão do MSP430G2553, e o circuito geral montado no protoboard. O circuito do protoboard é composto por um driver, para o acionamento dos LEDs com 5V, e o circuito com os sensores de luminosidade.

Foi usado o software Altium Designer para o esquema elétrico e montagem teórica do circuito utilizado, devido a facilidade de montagem e o conhecimento prévio do funcionamento da aplicação.

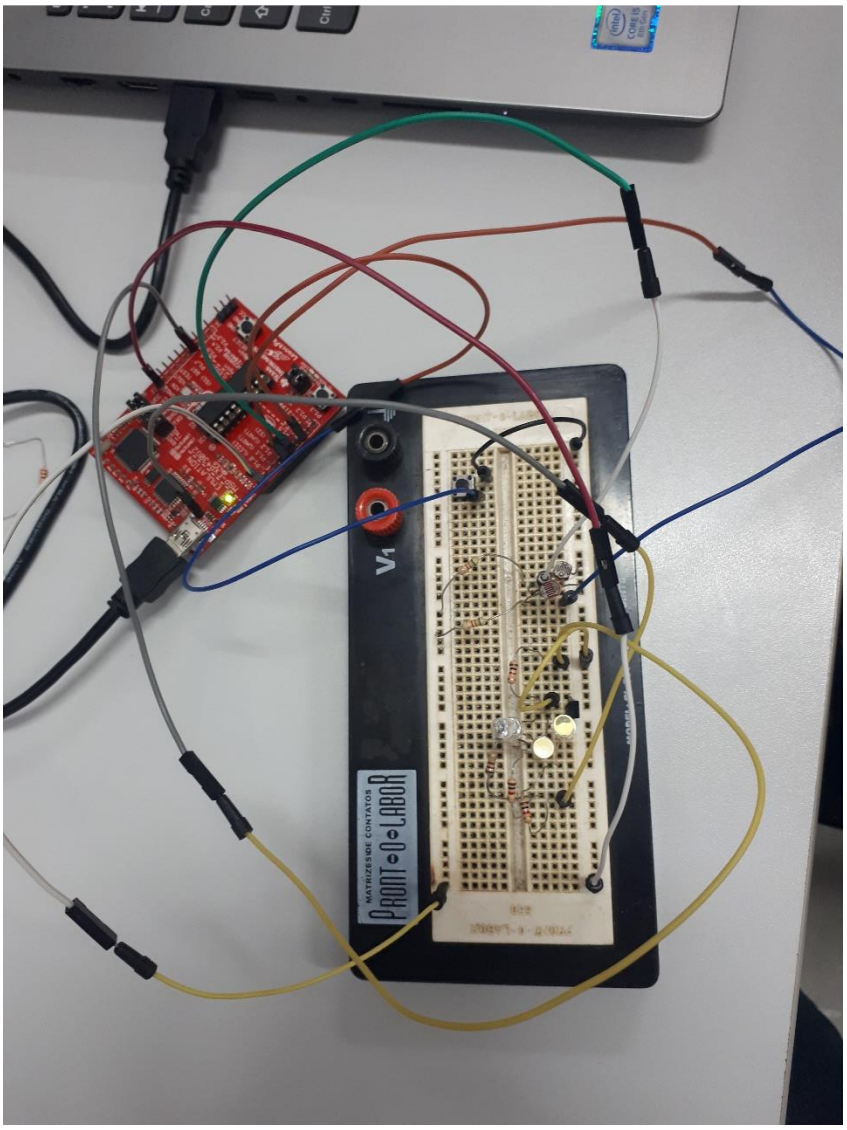
2.3.1. Driver

Nesse circuito, o intuito é aumentar a corrente nos 3 LEDs para acionar todos de uma só vez. Foi utilizado um transistor BC548 do tipo NPN, 3 LEDs de 5mm, 3 resistores de 1 kΩ para proteger os LEDs e um resistor de 1k Ω para proteger a base do transistor. Abaixo tem-se o esquema elétrico:



2.3.2. Circuito Geral

No circuito geral, está o driver para o acionamento dos LEDs, o botão para simulação de um sensor de presença, os sensores LDR para capturar a luminosidade do ambiente e acionar os LEDs caso a luminosidade ultrapasse o valor limite fornecido pelo usuário e conectado a ele tem-se o microcontrolador que realiza toda lógica da aplicação. Abaixo tem-se o circuito elétrico:



2.4. Firmware

O programa gravado na memória do MSP430 realiza toda sequência lógica da aplicação, desde a leitura da luminosidade do ambiente até o acendimento dos LEDs.

Como o microcontrolador utilizado é o MSP430G2553, então todo código e lógica foram feitos por meio da linguagem C e pela plataforma de desenvolvimento de códigos (IDE) IAR Embedded Workbench. A escolha da plataforma foi feita por meio de alguns fatores, como: Entre os seus dispositivos compatíveis, o microcontrolador utilizado no projeto também é compatível e também por ser um software utilizado no processo de aprendizado da disciplina de Microcontroladores do Inatel.

3.3[V]	VCC	1	20	VSS	GND
NO	P1.0	2	19	P2.6	NO
UART	P1.1	3	18	P2.7	NO
UART	P1.2	4	17	TEST	NO
Conversor AD (Sensor LDR)	P1.3	5	16	RST	RST
Conversor AD (Sensor LDR)	P1.4	6	15	P1.7	NC
NO	P1.5	7	14	P1.6	PWM (LEDs)
Botão (Sensor de presença)	P2.0	8	13	P2.5	NO
NO	P2.1	9	12	P2.4	NO
NC	P2.2	10	11	P2.3	N

3. Análise e Resultados

Comportamento desejado	**Transição de descida do botão (3.3 - 0 [V])**				
	1º teste	2º teste	3º teste	4º teste	5º teste
Botão (P2.0)	OK	OK	OK	OK	OK

Comportamento desejado	**Atracar ao jogar 3.3V na base do transistor**				
	1º teste	2º teste	3º teste	4º teste	5º teste
Driver	OK	OK	OK	OK	OK

Comportamento desejado	**Variar a tensão com mudança na luminosidade**				
	1º teste	2º teste	3º teste	4º teste	5º teste
Sensor 1 (P1.3)	OK	OK	OK	OK	OK

Comportamento desejado	**Variar a tensão com mudança na luminosidade**				
	1º teste	2º teste	3º teste	4º teste	5º teste
Sensor 2 (P1.4)	OK	OK	OK	OK	OK

Após a montagem e ligação dos pinos correspondentes do driver o dos sensores no MSP, foi feito os testes com os sensores e os LEDs, e tudo funcionou corretamente. Quando foi feito o teste com UART os valores recebidos estavam invertidos, e quando verificado, os valores do vetor “mensagem” não tinham sido zerados. Quando os valores foram modificados, e o projeto testado novamente, tudo funcionou como deveria.

Conclusão

Conclui-se que o projeto Light Control está em perfeito funcionamento, portanto atende todos os requisitos propostos e necessários para a utilização. O valor do limite de luminosidade que é usado para que os LEDs acendam, foi escolhido com base em testes com o sensor de luminosidade. Para a aplicação em um projeto real, poderia ser utilizado o mesmo sensor e uma lâmpada LED sendo chaveado em uma tensão de 127V, que é a utilizada para lâmpadas em geral.

4. Referências

Texas Instruments. MIXED SIGNAL MICROCONTROLLER. Disponível em: < http://www.ti.com/lit/ds/symlink/msp430g2553.pdf> . Acesso em: 28 maio 2017 Texas Instruments. MSP430x2xx Family User's Guide . Disponível em: < http://www.ti.com/lit/ug/slau144j/slau144j.pdf >. Acesso em 15 junho 2019

Gonçalves, Ana Letícia Gomes. Conversor AD. Disponível em: < https://siteseguro.inatel.br/PortalAcademico/PortalUniversitario/WebMaterialAulaUsuario.aspx>. Acesso em 15 jun. 2019.

Apêndice

4.1. Firmware (código fonte)

```
#include "msp430g2553.h"
#include <stdio.h>

void configTimer0(void);
void configTimer1(void);
void configInterrupt();
void configADC();
unsigned int leADC(unsigned int porta, unsigned int canal);
void configUART(void);
void UART_TX(char * tx_data);

// Variaveis
char mensagem_tx[20];
char mensagem_rx[32];
int tam=0;
int TAMANHO = 3;
int aux = 0;
unsigned int valorLimite = 0;
int valorAD1 = 0;
int valorAD2;
int valorAD;
unsigned int valorPWM = 0;
unsigned char cont = 0;
```

```
// FUNCAO MAIN
void main(void)
{
  // DESABILITANDO O WATCHDOG
  WDTCTL = WDTPW + WDTHOLD;

  BCSCTL1 = CALBC1_1MHZ;
  DCOCTL = CALDCO_1MHZ;

  // Para o botao
  P2REN |= BIT0;
  P2OUT |= BIT0;
  // Para o PWM
  P1DIR |= BIT6;
  P1SEL |= BIT6;

  configUART();
  configADC();
  configInterrupt();
  configTimer0();
  configTimer1();

  _BIS_SR(GIE); // HABILITANDO INTERRUPÇÃO GLOBAL

  mensagem_rx[0] = 0;
  mensagem_rx[1] = 0;
```

```
  mensagem_rx[2] = 0;
  TAMANHO = 2;

  // Entrando com o valor limite na UART
  UART_TX("Digite o limite de luminosidade ambiente: \r\n");
  do{
     valorLimite = (( mensagem_rx[0]-48)*10 +(mensagem_rx[1]-48)*1);
  }while(valorLimite != 45);
  sprintf(mensagem_tx, "<%.3d>\n\r",valorLimite);
  UART_TX(mensagem_tx);

  mensagem_rx[0] = -1;
  mensagem_rx[1] = -1;
  mensagem_rx[2] = -1;
  TAMANHO = 3;

  // Entrando com o valor maximo de luminosidade na UART
  UART_TX("Digite o valor maximo de luminosidade do led: \r\n");
  do{
     valorPWM = ((mensagem_rx[0] - 48)*100 +( mensagem_rx[1] - 48)*10
+(mensagem_rx[2] - 48)*1);
  }while(valorPWM < 0 || valorPWM > 100);
  sprintf(mensagem_tx, "<%.3d>\n\r",valorPWM);
  UART_TX(mensagem_tx);

  //valorLimite = 45;
  valorLimite =  (valorLimite*10.23);
```

```
  //valorPWM = 20;

  for(;;){
        valorAD1 = leADC(BIT4,INCH_4);
        valorAD2 = leADC(BIT3,INCH_3);
        valorAD = (valorAD1 + valorAD2)/2;

        sprintf(mensagem_tx, "<%.3d>\n\r",valorAD);
        UART_TX(mensagem_tx);
  }
}
// FUNCAO DE CONFIGURACAO DO TIMER1
void configTimer1(void){
  TA1CTL |= MC_1;
  TA1CTL |= TASSEL_2;
  TA1CTL |= ID_1;
  TA1CCR0 |= 49999;
  TA1CCTL0 |= CCIE;
}

// FUNCAO DE CONFIGURACAO DO TIMER0 COMO PWM
void configTimer0(void){
  TA0CCTL1 = OUTMOD_7;
  TA0CCR0 = 19999;
  TA0CCR1 = 0;
  TA0CTL = TASSEL_2 + ID_0 + MC_1;
  TA0CCTL0 &= ~CCIE;
```

```
}

// FUNCAO DE CONFIGURACAO DE INTERRUPCAO
void configInterrupt(){
  P2IES |= BIT0;
  P2IE |= BIT0;
  P2IFG &=~ BIT0;
}

// FUNCAO DE CONFIGURACAO DO CONVERSOR AD (COM SENSOR)
void configADC()
{
  ADC10CTL1 |= SHS_0 + ADC10SSEL_3;
  ADC10CTL0 = SREF_0 + ADC10ON;
}

// FUNCAO PARA LER O CONVERSOR AD
unsigned int leADC(unsigned int porta, unsigned int canal)
{
  unsigned long int valor = 0;
  int media;
  ADC10AE0 |= porta;

  ADC10CTL0 &=~  ADC10ON;
  ADC10CTL1 &=~ INCH_7;
  ADC10CTL1 |= canal;
  ADC10CTL0 |=  ADC10ON;
```

```
  for(int i = 0;i<100;i++){
    ADC10CTL0 |= ENC + ADC10SC;
    while((ADC10CTL0 & ADC10IFG) == 0);
    valor += ADC10MEM;
  }

   ADC10CTL0 &= ~ADC10IFG;
   valor = valor/100;
   media = (int) valor;
   return media;
}

// FUNCAO PARA CONFIGURAR O UART
void configUART(void)
{
  P1SEL |= BIT1 + BIT2;
  P1SEL2 |= BIT1 + BIT2;
  UCA0CTL1 |= UCSSEL_2;
  UCA0BR0 = 104;
  UCA0BR1 = 0;
  UCA0MCTL = UCBRS0;
  UCA0CTL1 &= ~UCSWRST;
  IE2 |= UCA0RXIE;
}

//Função de envio de mensagens “STRINGS”
```

```
void UART_TX (char * tx_data)
{
  unsigned int i=0;
  while(tx_data[i]) // Espera enviar todos caracteres da STRING
  {
    while ((UCA0STAT & UCBUSY)); //espera terminar o envio da ultima informação
    UCA0TXBUF = tx_data[i]; // envia o elemento na posição i
    i++; // incrementa posição do vetor
  }
}

// ROTINA DE INTERRUPÇÃO DO TIMER1
#pragma vector = TIMER1_A0_VECTOR
__interrupt void Timer1_A0_ISR(void)
{

   if(((P2IN&BIT0) == 0) && (valorAD > valorLimite)){
      cont = 0;
    }else{
      cont++;
    }

   if(cont >= 100){
    TACCR1 = 0;
   }

  TA1CCTL0 &=~ CCIFG; // LIMPANDO A FLAG DE INTERRUPÇÃO
```

```
}

// ROTINA DE INTERRUPÇÃO EXTERNA
#pragma vector = PORT2_VECTOR
__interrupt void INTERRUPCAO_EXTERNA_ISR(void)
{

  if(valorAD > valorLimite){
    TA0CCR1 = valorPWM * 499;
  }



  P2IFG &=~ BIT0; // LIMPANDO A FLAG DE INTERRUPÇÃO
}

// ROTINA DE INTERRUPÇÃO PARA RECEPÇÃO DE MENSAGENS
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
  mensagem_rx[tam] = UCA0RXBUF;

  tam++;
  if(tam == TAMANHO)
  {
    tam=0;
  }
```

```
  IFG2=IFG2&~UCA0RXIFG;
}
```