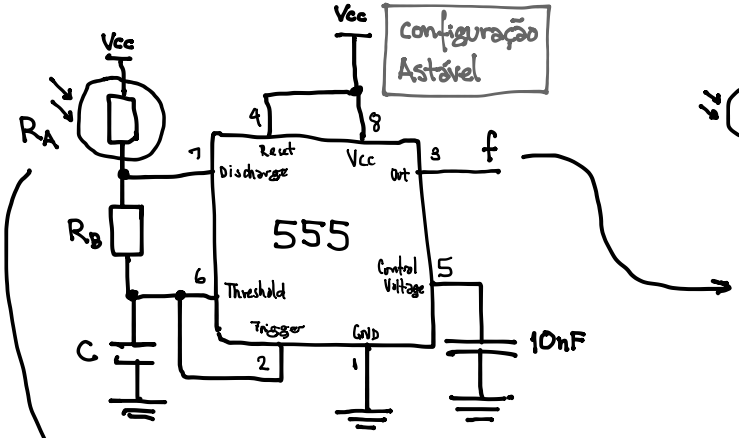
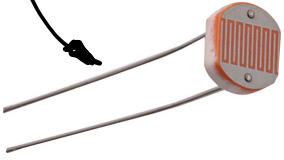
PROJETO: Monitor de intensidade Luminosa

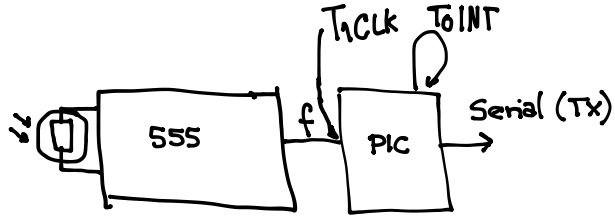


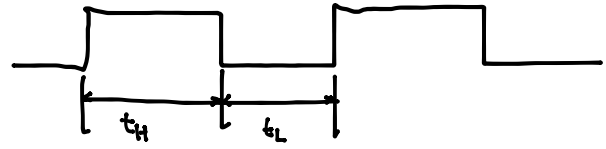


LDR - Resistor Variável por Luz

no claro: 400Ω

no escuro: >1MΩ (típico: 10MΩ)





$t_H = 0{,}693(R_A + R_B).C$

$t_L = 0{,}693(R_B).C$

$T = 0{,}693(R_A + 2R_B).C$

Sugestão para R_B e C:

$R_B = 680K\Omega$ e $C = 1nF$

Por que esses valores são bons?

1) Quais são as frequencias mínima e máxima produzidas pelo 555?
2) Determine um periodo de amostragem adequado para ser gerado um número que represente a intensidade luminosa.
3) Escreva o software para o PIC

```c
/*
 * File:   main.c
 * Author: Jun Okamoto Jr.
 *
 * Created on June 11, 2014, 2:59 PM
 */

#include <stdio.h>
#include <stdlib.h>


// PIC16F886 Configuration Bit Settings

#include <xc.h>

// #pragma config statements should precede project file includes.
// Use project enums instead of #define for ON and OFF.

// CONFIG1
#pragma config FOSC = INTRC_NOCLKOUT// Oscillator Selection bits (INTOSCIO oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled and can be enabled by SWDTEN bit of the WDTCON register)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = ON       // RE3/MCLR pin function select bit (RE3/MCLR pin function is MCLR)
#pragma config CP = OFF         // Code Protection bit (Program memory code protection is disabled)
#pragma config CPD = OFF        // Data Code Protection bit (Data memory code protection is disabled)
#pragma config BOREN = OFF      // Brown Out Reset Selection bits (BOR disabled)
#pragma config IESO = ON        // Internal External Switchover bit (Internal/External Switchover mode is enabled)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is enabled)
#pragma config LVP = OFF        // Low Voltage Programming Enable bit (RB3 pin has digital I/O, HV on MCLR must be used for programming)

// CONFIG2
#pragma config BOR4V = BOR40V   // Brown-out Reset Selection bit (Brown-out Reset set to 4.0V)
#pragma config WRT = OFF        // Flash Program Memory Self Write Enable bits (Write protection off)

// Bibliotecas locais do projeto
#include "always.h"
#include "delay.h"
#include "serial.h"

/*
 *
 */
volatile int timer0Counter; // contador de interrupções do Timer 0
volatile unsigned int timer1Count; // contagem capturada pelo Timer 1

void interrupt isr () {
    if (TMR0IE && TMR0IF) { // se for interrupção de Timer 0
        if (++timer0Counter >= 3) { // interrompe 3 vezes que é equivalente a 98,30 ms
            TMR1ON = 0; // para contagem do Timer 1
            timer1Count = (unsigned int)(TMR1H * 256 + TMR1L); // armazena a contagem
            TMR1H = 0; TMR1L = 0; // zera a contagem do Timer 1
            timer0Counter = 0;    // zera contador do Timer 0
            RB5 = ~RB5; // alterna LED para para uso como Trigger do osciloscópio
        }
        TMR0IF = 0; // reseta flag de interrupção do Timer 0
    }
}

int main(int argc, char** argv) {
    // Configuração do Timer 1 como contador de eventos externos
    T1CKPS0 = 0; // prescale de 1:1, LSB
    T1CKPS1 = 0; // prescale de 1:1, MSB
    T1OSCEN = 0; // oscilador LP desligado
    TMR1ON = 0;  // inicialmente, desligado
    TMR1CS = 1;  // clock externo para contagem de eventos

    // Configuração do Timer 0 para geração de interrupção a cada 100 ms
    T0CS = 0; // usa clock interno FOSC/4
    PSA = 0; // prescaler associado ao Timer 0
    PS2 = 1; PS1 = 1; PS0 = 1; // prescaler de 256

    // Habilita interrupções
    GIE = 1;    // habilita interrupções em geral
    TMR0IE = 1; // habilita interrupção do Timer 0


    // Inicializações
    TRISB5 = 0; // bit onde está o LED para saída
    serial_init();

    while (1) { // loop principal
        if (TMR1ON == 0) { // se o timer 1 estiver desligado
            // mostra o seu valor pelo serial
            printf("%05u | ",  timer1Count); // manda contagem pelo serial
            TMR1ON = 1; // re-liga Timer 1
        }

    } // fim do loop principal

    return (EXIT_SUCCESS);
}
```