UNIVERSIDADE FEDERAL DE OURO PRETO

ESCOLA DE MINAS

COLEGIADO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - CECAU

ANDRÉ LOPES MARINHO DOS SANTOS

DESENVOLVIMENTO DE SISTEMA DE CONTROLE PARA IRRIGAÇÃO UTILIZANDO MICROCONTROLADOR PIC16F873A E APLICATIVO ANDROID

MONOGRAFIA DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Ouro Preto, 2016

ANDRÉ LOPES MARINHO DOS SANTOS

DESENVOLVIMENTO DE SISTEMA DE CONTROLE PARA IRRIGAÇÃO UTILIZANDO MICROCONTROLADOR E APLICATIVO ANDROID

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para obtenção do Grau de Engenheiro de Controle e Automação.

Orientador: Alan Kardek Rêgo Segundo

Ouro Preto

Escola de Minas – UFOP

Março/2016

S237d André Lopes Marinho dos Santos.
Desenvolvimento de sistema de controle para irrigação utilizando microcontrolador PIC16F873A e aplicativo Android [manuscrito] / André Lopes Marinho dos Santos. – 2016.
57f. : il., color., tab.

Orientador: Prof. Dr. Alan Kardek Rêgo Segundo.

Monografia (Graduação) – Universidade Federal de Ouro Preto. Escola de Minas. Colegiado do Curso de Engenharia de Controle e Automação e Técnicas fundamentais.
Área de concentração: Engenharia de Controle e Automação.

1.Automação industrial. 2. Microcontroladores. 3. Android (Recurso eletrônico). 4. Redes de sensores sem fio – Bluetooth. I. Universidade Federal de Ouro Preto. II. Título.

CDU: 681.5

Fonte de catalogação: bibem@sisbin.ufop.br

Monografia defendida e aprovada, em 22 de março de 2016, pela comissão avaliadora constituída pelos professores:

Prof. Dr. Alan Kardek Rêgo Segundo - Orientador

Prof. Dr. Paulo Marcos de Barros Monteiro – Professor Convidado

Prof. Dr. Sávio Augusto Lopes da Silva – Professor Convidado

RESUMO

A proposta do presente trabalho é a criação de um sistema de controle de baixo custo utilizando microcontrolador, para utilização em sistemas de irrigação. São dimensionados os circuitos elétricos necessários para o bom funcionamento do microcontrolador, assim como os circuitos necessários para a conexão com seus periféricos. É utilizado um display de cristal líquido para a visualização do horário e demais configurações do sistema de controle, um relógio de tempo real para uma maior precisão na contagem de tempo e um módulo Bluetooth para receber dados de parametrização, enviados por meio de aplicativo Android. É desenvolvido o programa do microcontrolador em linguagem C utilizando o software CCS. Depois de realizada a compilação do programa, este é testado juntamente com seu respectivo circuito, utilizando o software Proteus. Após o teste de funcionamento do circuito, é feita a montagem em *protoboard*, para a análise do sistema físico em funcionamento. Por fim, é desenvolvido o aplicativo Android, o qual utiliza a linguagem de programação Java e o Android Studio, a plataforma oficial de desenvolvimento para Android.

Palavras chave: Microcontrolador, PIC16F873A, Android, Bluetooth, sistema de irrigação.

ABSTRACT

The purpose of this work is to create a low-cost control system using microcontroller for use in irrigation systems. The necessary electrical circuits are sized for the proper functioning of the microcontroller as well as the necessary circuitry for connecting to itsperipherals. It is used a liquid crystal display for viewing time and other control system settings, a real time clock for accuracy in counting time and a Bluetooth module to receive parameters sent by Android application. The microcontroller program is written in C language using CCS software. After finishing the program compilation, it is tested with their respective circuit using Proteus software. After the circuit function test, the montage is done in breadboard for analysis of the physical system in operation. Finally, it is developed an Android application, which uses Java programming language and Android Studio which is the official development platform for Android.

Key words: Microcontroller, PIC16F873A, Android, Bluetooth, irrigation system.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

No presente capítulo é apresentada a formulação do problema do qual se trata este trabalho, os objetivos gerais e específicos os quais se deseja alcançar, a justificativa pela escolha do tema abordado, a motivação que levou a escolha desta área de pesquisa e uma sucinta descrição da organização do trabalho.

1.1 Formulação do Problema

A água é um elemento essencial à sobrevivência de todos os seres vivos. Além disso, está diretamente envolvida em quaisquer processos produtivos como fabricação de roupas, carros, computadores, entre outros. É um recurso natural extremamente abundante no planeta terra. Sabe-se que mais de 75% de sua superfície terrestre é coberta por água, fator o qual se deve a crença de que a água é um recurso inesgotável (Planeta sustentável, 2012).

A quantidade de água utilizada per capita vem crescendo a uma taxa duas vezes maior do que a taxa de crescimento populacional mundial, fazendo com que em algumas décadas este cenário possa se tornar irreversível. Cerca de 1,2 bilhões de habitantes do planeta vivem hoje em áreas de escassez hídrica, o que representa quase 20% da população mundial, e outras 500 milhões de pessoas já se encontram em situações onde ocorram problemas causados pela falta de água (United Nations - Department of Economic and Social Affairs, 2014).

Essas pessoas habitam regiões as quais se tem uma oferta inferior a 1700 metros cúbicos de água anuais por habitante, que segundo a ONU (Organização das Nações Unidas) é o limite mínimo avaliado como seguro para a sobrevivência.

Previsões atingiram resultados ainda mais alarmantes. Em 2025, espera-se que 1,8 bilhões de pessoas vivam em situação de completa escassez de água e caso não ocorra uma mudança de cenário esse número chegará a quase metade da população mundial, em 2030 (United Nations - Department of Economic and Social Affairs, 2014).

Grande parte do desperdício de água no mundo é atribuída a atividades agrícolas. É estimado que uma redução de 10% de desperdício de água apenas no setor da agricultura, já seria suficiente para suprir as necessidades de toda a população mundial. No Brasil, este cenário mostra-se equivalente, com grande parte do consumo hídrico na agricultura, especialmente na irrigação. Os

métodos mais utilizados são pivôs, aspersores convencionais e espalhamento superficial. Todos os três mais utilizados no Brasil são considerados métodos pouco eficientes de irrigação e, além disso, pode-se ressaltar que aspersores e pivôs tem gasto energético elevado, em comparação com outros métodos existentes (REBOUÇAS, 2003).

O presente trabalho propõe a utilização de um sistema de controle de baixo custo para auxílio em processos agrícolas, em particular, processos os quais é necessária à irrigação. O controlador proposto é adaptável a diversos métodos de irrigação operados via acionamentos elétricos. Pode-se ajustá-lo, por exemplo, ao número de aspersores ou pivôs de uma plantação.

1.2 Objetivos Gerais

O presente trabalho tem como objetivo o desenvolvimento de um sistema de controle de baixo custo que auxilie processos agrícolas visando à diminuição do desperdício de água e energia elétrica. O controlador proposto poderá ser parametrizável remotamente, facilitando a operação e monitoramento da água utilizada nos processos, via aplicativo desenvolvido para celular com sistema operacional Android, por meio de conexão sem fio com o sistema físico, utilizando tecnologia Bluetooth.

1.3 Objetivos Específicos

- Projeto do circuito elétrico do controlador
- Programação do microcontrolador
- Desenvolvimento de aplicativo de fácil utilização para a parametrização
- Simulação do sistema em *protoboard*

1.4 Justificativa

O uso indevido da água, especialmente na agricultura deve-se especialmente a métodos ineficientes de irrigação. Por meio de um sistema de controle de fácil ajuste espera-se diminuir o desperdício provocado por esta atividade. Sistemas de controle utilizando microcontroladores podem ser facilmente adaptados a sistemas já existentes, visto que ocupam pouco espaço físico e despendem de baixíssimo consumo energético.

1.5 Motivação

O problema da escassez hídrica é global, porém a região centro-oeste de minas gerais, onde grande parte da família do autor reside, já sofreu as consequências do uso indevido deste recurso natural tão importante. Em meados de 2015 algumas cidades entraram em racionamento de água. Em Pará de Minas, em particular, foi decretado estado de calamidade pública. Fato que aumentou o interesse do autor em áreas de pesquisa como energia renovável e técnicas mais eficientes de utilização dos recursos naturais.

1.6 Organização do Trabalho

O presente trabalho foi dividido em quatro partes: Revisão de Literatura, Materiais e Métodos, Resultados e Conclusões e Recomendações para Trabalho Futuros.

Inicialmente é apresentada uma revisão bibliográfica sobre microcontroladores e componentes utilizados no trabalho e desenvolvimento de aplicações para Android.

Na segunda parte são mostrados os circuitos elétricos envolvendo os componentes utilizados, além do código armazenado no microcontrolador e códigos para criação do aplicativo.

Na terceira parte são apresentados os resultados da simulação do sistema de controle, utilizando um *protoboard*, assim como uma breve discussão de seu funcionamento.

Por fim, foram feitas as considerações finais a serem comentadas e as recomendações para trabalhos futuros.

2. REVISÃO DE LITERATURA

Neste capítulo, são retratados estudos realizados para obtenção de embasamento teórico para o desenvolvimento do sistema de controle para irrigação, por meio de pesquisas sobre funcionamento de componentes e softwares utilizados durante a fase de desenvolvimento.

2.1 **O Microcontrolador**

O microcontrolador é um componente eletrônico programável utilizado para aplicações que envolvem processos lógicos. Diz-se que o microcontrolador é programável, pois toda a lógica é estruturada e gravada em uma memória interna, a qual pode ser alterada conforme necessidade. Esta é executada toda vez que o dispositivo é alimentado (SOUZA, 2009).

Microcontroladores são considerados dispositivos de baixo custo e facilmente inseridos em sistemas de controle digital, devido a seus circuitos internos, os quais facilmente se interagem com os periféricos, de forma rápida e eficiente. Podem-se utilizar, por exemplo, os conversores A/D do microcontrolador para a leitura de sensores, portas de entrada e saída de dados que podem ser acessadas praticamente de forma simultânea (devido a sua grande velocidade de processamento), *timers* que permitem a inserção de lógicas de programação levando em conta a contagem de tempo (IBRAHIM, 2006).

Atualmente, a maior parte dos programadores e engenheiros desenvolve códigos para microcontroladores em linguagens de alto nível como C e PASCAL, em alternativa a linguagem Assembly usualmente utilizada há alguns anos. Essa mudança deve-se a simplicidade de programação em linguagens de alto nível, facilidades de testes e menor ocorrência de erros em geral. Dentre os pontos negativos, pode-se citar uma geração de código maior para ser gravado e executado pelo microcontrolador, o que faz com que ele opere com uma velocidade de processamento menor (IBRAHIM, 2006). O desenho esquemático de um microcontrolador é mostrado na Figura 2.1.

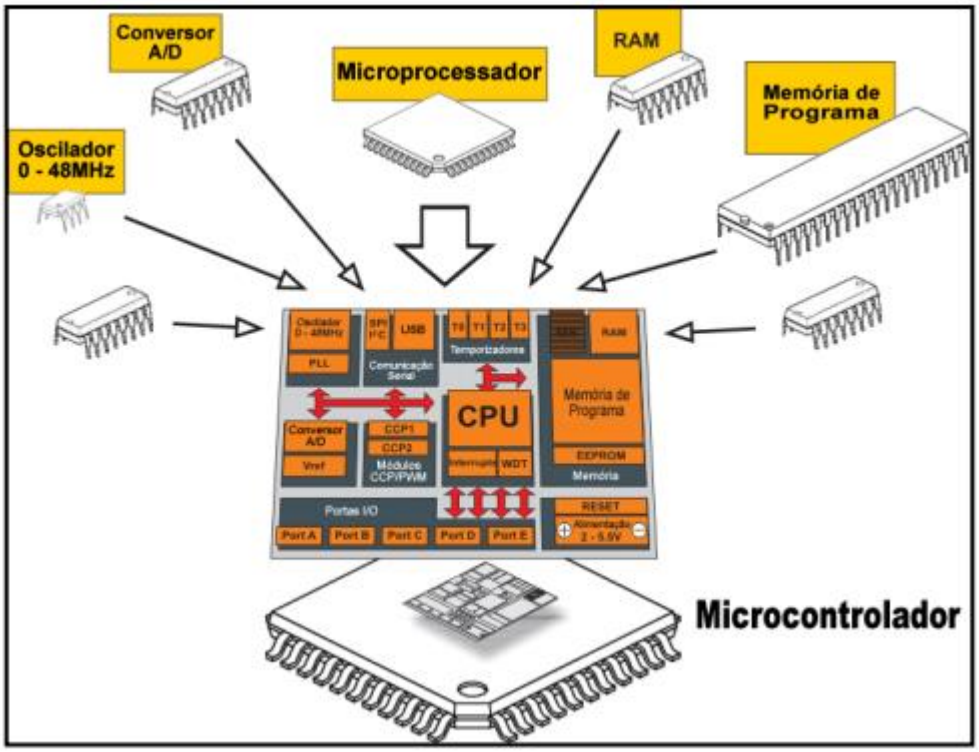

Figura 2.1 - Desenho esquemático de um microcontrolador

Fonte: TORRES; MARTINS

2.1.1 PIC16F873A

Existe uma grande variedade de microcontroladores disponíveis no mercado. Eles apresentam uma variabilidade quanto a tamanho, velocidade de processamento, preço, quantidade de portas de entrada e saída, conversores A/D e D/A, *timers*, entre outras funcionalidades necessárias a cada tipo de projeto.

Figura 2.2 - Microcontrolador PIC16F873A

Fonte: MICROCHIP, 2013

No presente projeto optou-se pela utilização do PIC16F873A (Figura 2.2), fabricado pela empresa Microchip. É um microcontrolador de 28 pinos que pode operar com velocidade de processamento de 20 MHz, ou seja, consegue executar uma instrução em um intervalo de apenas 200 nano segundos.

Observando a "pinagem" (Figura 2.3) do PIC16F873A, pode-se verificar que alguns dos pinos podem ser configurados de diversas maneiras. Assim, um pino que é programado para ler um valor analógico de um sensor, pode ser reprogramado, por exemplo, para ser um pino de saída digital acionando um relé.

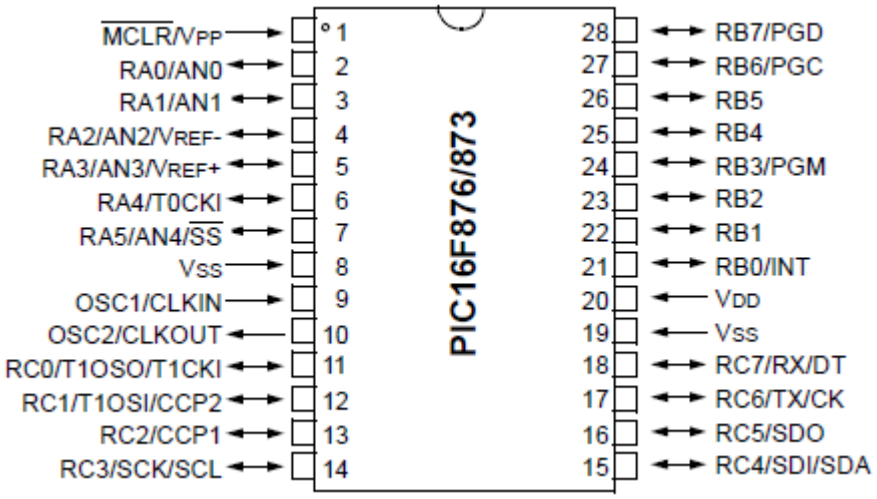

Figura 2.3 - Pinagem do microcontrolador PIC16F873A

Fonte: MICROCHIP, 2013

2.2 Display de cristal líquido (LCD)

Displays de informação têm se tornando um elemento importante nos dias atuais, sendo uma maneira de estabelecer interação homem-máquina. Existem diversos tipos de displays como display de 7 segmentos (encontrados em balanças e caixas eletrônicos), telas de plasma e LCD's. O LCD (Figura 2.4), em particular, é bastante utilizado por dispor de vantagens como ser um dispositivo leve, ter grande área com alta definição, necessitar de baixas tensões para operar e consumir pouca energia elétrica (YEH; GU, 2010).

Figura 2.4 - Display de cristal líquido

Fonte: (Arduino Tutorial Hello Word, 2016).

.

2.3 Relógio de tempo real (RTC-DS1307)

Microcontroladores possuem circuitos internos os quais tornam possível a contagem de tempo, porém, para aplicações onde espaços temporais são fatores primordiais, utiliza-se um circuito integrado externo, para garantir uma maior precisão.

O circuito integrado DS1307 (Figura 2.5) realiza contagem de segundos, minutos, horas, dia de mês, dia da semana, mês, ano e correção de ano bissexto. Opera em faixa de temperatura industrial (-40ºC a +85ºC) e possui sistema de detecção de falhas que faz com que o dispositivo passe a ser alimentado por uma bateria, caso ocorra algum problema com a fonte de alimentação (NETO, 2011).

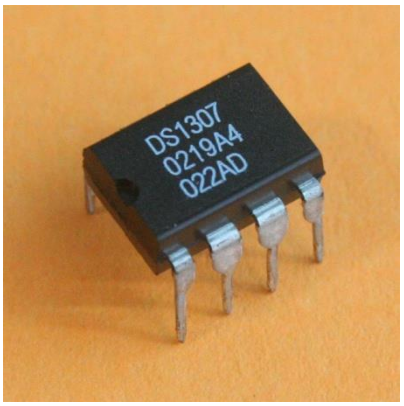

Figura 2.5 - Relógio de tempo real (RTC-DS1307)

Fonte: (MARIAN, 2015)

2.4 LED (Doido emissor de luz)

Recentemente, a fabricação de grande parte dos displays digitais e fontes de luz têm como um dos principais elementos um tipo especial de diodo, o LED, capaz de emitir luz visível quando polarizado corretamente. Quando se energiza um semicondutor do tipo p-n, ele apresenta liberação de energia em forma de calor e fótons (ROBERT L. BOYLESTAD, 2002). Em alguns materiais a porcentagem de fótons emitida é desprezível, como ocorre com o silício e o germânio. Por outro lado, quando se têm materiais como o Arseneto Fosfeto de Gálio (GaAsP) e o Fosfeto de Gálio (GaP), é criada uma fonte de luz altamente visível, com a vantagem de não ocorrer uma grande dissipação de calor.

Os LED's, assim como os demais tipos de diodo, são dispositivos bipolares contendo um lado denominado catodo e outro anodo, que podem ser identificados conforme a Figura 2.6.

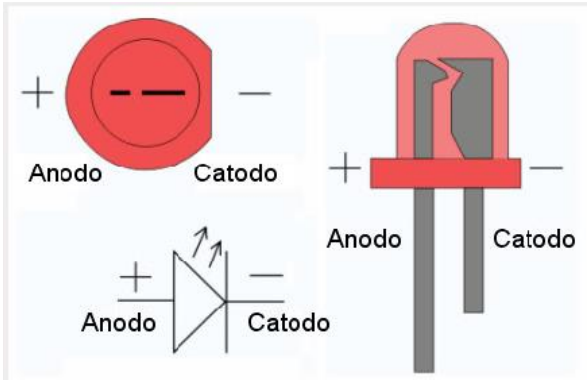

Figura 2.6 - Representação esquemática de um LED

Dentre as vantagens que os LED’s oferecem, podem-se destacar a vida útil elevada, eficiência em termos energéticos, baixa voltagem de operação, acionamento instantâneo, possível dimerização e ausência de radiações infravermelho e ultravioleta.

2.5 Bluetooth

O Bluetooth é uma interface de comunicação de alta confiabilidade e fácil utilização, por estar disponível na maior parte dos smartphones encontrados no mercado. A comunicação é realizada por meio de ondas de rádio, e seu alcance pode variar de 1 m chegando até 100 m, dependendo da potência dos dispositivos utilizados. A comunicação via Bluetooth não é recomendada para aplicações onde há demanda de alta velocidade de transmissão de dados, por contar com uma taxa de apenas 24MB/s, em sua versão mais recente (FERREIRA; ALVES, 2013).

Na Figura 2.7 é exibido um módulo comumente utilizado para comunicação Bluetooth em circuitos digitais.

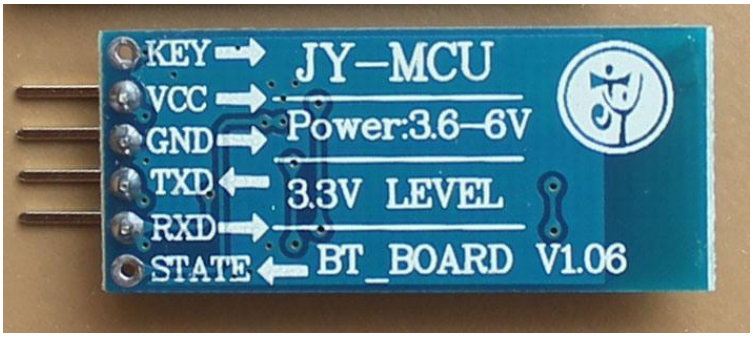

Figura 2.7 - Modulo Bluetooth

2.6 Proteus

O Proteus é uma plataforma de desenvolvimento para sistemas embutidos. Por meio dessa ferramenta, pode-se criar tanto o esquema elétrico envolvendo o microcontrolador, quanto o programa que será posteriormente inserido no mesmo. Como apresentado na Figura 2.8, pode-se acelerar o desenvolvimento de projetos, além da redução de custos, visto que hardware e software podem ser testados sem a necessidade de um protótipo físico ser fabricado.

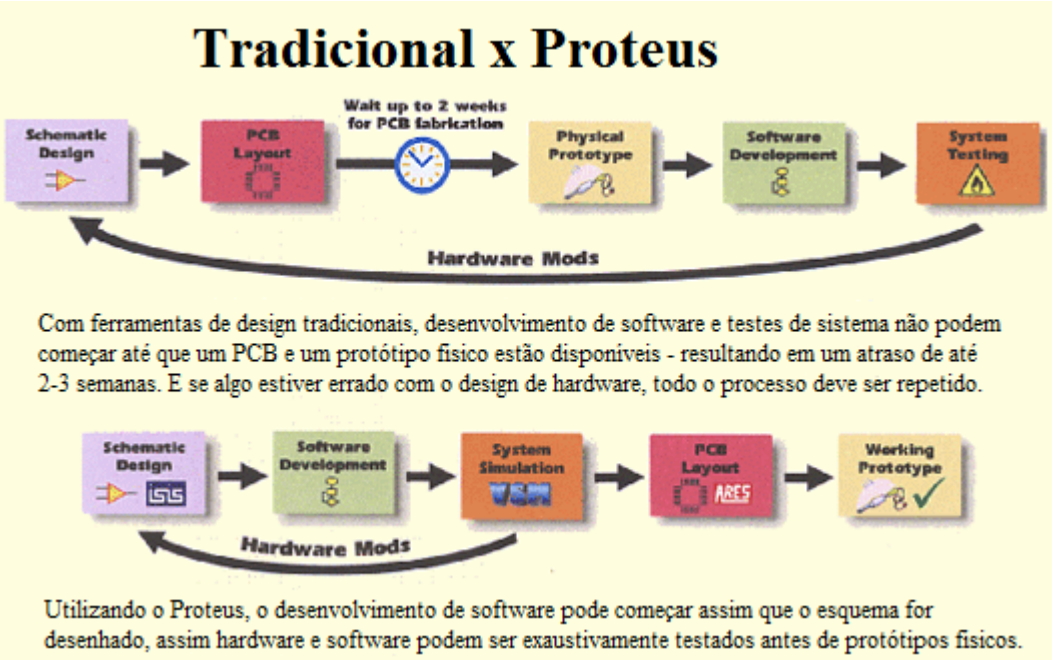

Figura 2.8 - Esquema comparativo de processo tradicional e com utilização do Proteus

Fonte: Adaptado de Labcenter Electronics, 2016.

2.7 Android OS

O sistema operacional Android despertou, desde seu lançamento, grande interesse de empresas, desenvolvedores e público em geral. O sistema passa por um constante aperfeiçoamento tanto de suas características como no aumento da quantidade de hardwares onde ele é suportado (MAIA; NOGUEIRA; PINHO, 2010). Atualmente, mais de 1 bilhão de dispositivos em todo o mundo, operam por meio de um sistema operacional Android, incluindo *smartphones*, *tablets*, relógios, televisões, entre outros (Android, 2016).

Existem três aspectos significativos que tornam o Android de grande interesse para a indústria em geral. O primeiro deve-se ao fato de ser um projeto de código aberto, ou seja, pode ser analisado e entendido por completo, torna mais rápida a detecção de *bugs* e criação de novas funcionalidades e mais fácil adaptação a novos hardwares. Outro fator consiste em ter uma arquitetura de sistema baseada no Linux, o que torna possível a utilização das características já conhecidas e oferecidas pelo sistema Linux. Por fim, aplicações Android são baseadas e na linguagem de programação Java, possuindo sua própria máquina virtual, possibilitando o

desenvolvedor de usufruir de suas vantagens e ponderar seus problemas já conhecidos (MAIA; NOGUEIRA; PINHO, 2010).

2.8 Android Studio

Andoid Studio é a plataforma oficial para desenvolvimento de aplicativos para Android. Possui editor de código inteligente, fazendo com que o desenvolvedor tenha mais agilidade e eficiência na programação. Na plataforma já existem alguns modelos de código prontos, que podem ser facilmente inseridos no projeto, como por exemplo, menus de navegação. Possui gerenciador de dispositivos virtuais integrado, tornando possível a simulação das aplicações nos dispositivos Android mais comumente utilizados. Como mostrado na Figura 2.9, conta com um editor de layout com suporte para *drag and drop*, facilitando a criação de telas para os aplicativos. Por meio de ferramentas pode-se averiguar a compatibilidade, desempenho, usabilidade e possíveis problemas nas aplicações (Android developers, 2016).

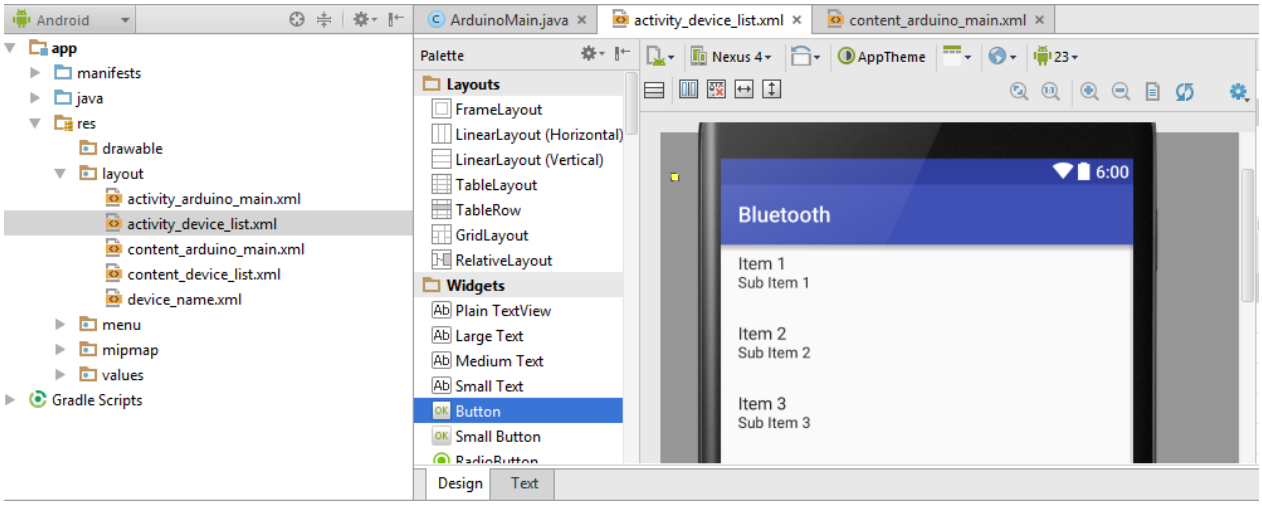

Figura 2.9 - Editor de layout do Android Studio

3. MATERIAIS E MÉTODOS

Neste capítulo são descritos os materiais e métodos utilizados no presente projeto, bem como os esquemas de ligação entre os componentes e os principais elementos do código desenvolvido.

3.1 Circuito do microcontrolador

3.1.1 Hardware básico para funcionamento

Circuitos digitais em geral, como é o caso dos sistemas onde há utilização de microcontroladores, são alimentados por corrente continua. Quando ocorre grande variação de corrente, há possibilidade de o circuito não apresentar o funcionamento esperado. Em aplicações práticas, é utilizado o capacitor de desacoplamento (Figura 3.1), também chamado de filtro. Neste projeto será utilizado um capacitor de capacitância igual a 100 nF, valor suficiente para garantir boa estabilidade do circuito de alimentação. Sua função é atenuar efeitos de uma tensão de entrada flutuante. Com a utilização do capacitor de desacoplamento, observa-se uma tensão flutuante da ordem de 10 mV, o que pode ser considerado adequado para garantir o bom funcionamento do dispositivo (Microchip forums, 2016).

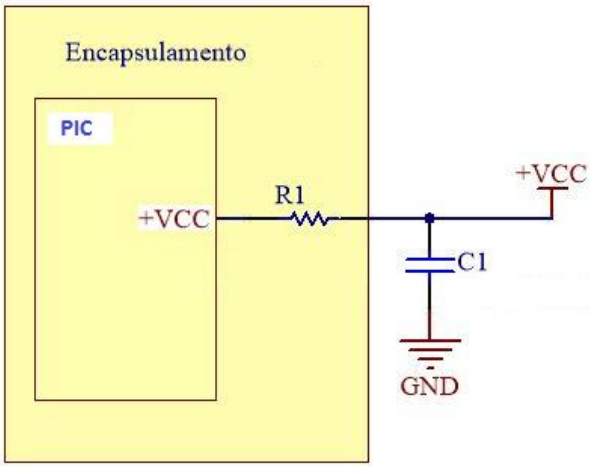

Figura 3.1 - Esquema de ligação do capacitor de desacoplamento

O Master Clear é uma funcionalidade do PIC que, quando acionada, interrompe o funcionamento do programa, e faz com que o mesmo se reinicie. A utilização da função Master Clear pode ser implementada por meio de hardware ou software. Neste projeto foi optado pela opção de

implementação usando hardware (Figura 3.2). O pino do Master Clear (MCLR) possui normalmente uma entrada em nível alto. A função Master Clear é ativada quando esta entrada é modificada para nível baixo, ou seja, basta acionar um botão manualmente para que o programa seja reiniciado.

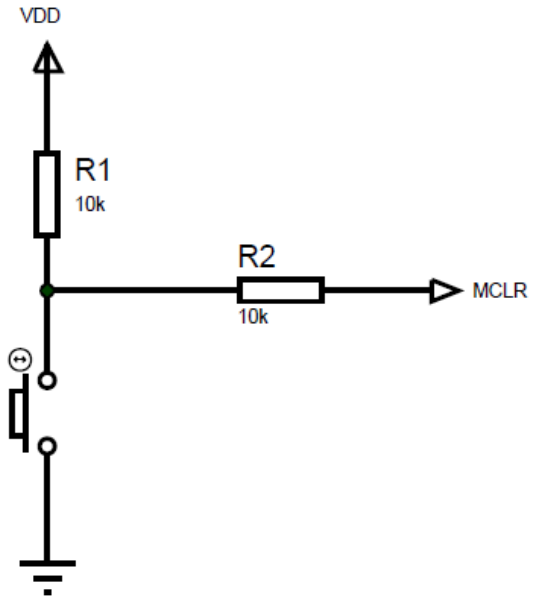

Figura 3.2 - Esquema de ligação de circuito de reset

O circuito de *clock* é composto por um cristal de quartzo e dois capacitores. O cristal vibra a uma alta frequência e enorme precisão, e é utilizado para sincronizar os ciclos de máquina do microcontrolador. O funcionamento do microcontrolador e periféricos são coordenados por esse relógio. A configuração do cristal utilizado é feita via software, por meio das diretivas de configuração. Neste projeto é utilizada a configuração HS (*high speed*) para o oscilador, por tratar-se de um cristal de alta frequência, operando a 20 MHz. Observa-se que no Datasheet do PIC16F873A é indicado a utilização de capacitores de 33 pF, quando o oscilador é configurado para operar em modo HS. Conforme a Figura 3.3 os capacitores tem um lado aterrado e outro conectado ao cristal, que por sua vez é interligado as suas respectivas entradas do microcontrolador, conforme informado no Datasheet.

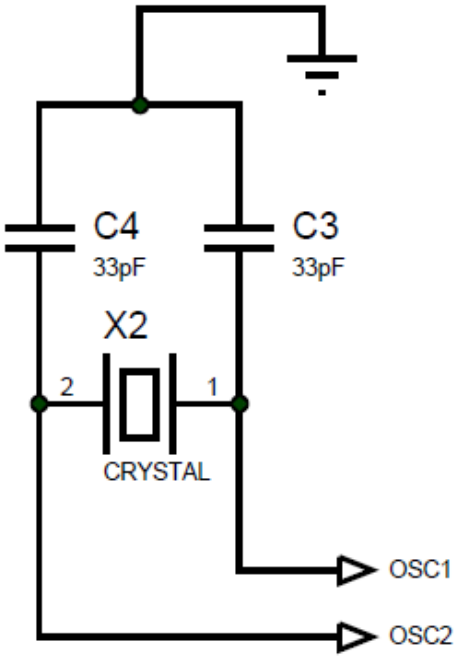

Figura 3.3 - Esquema de ligação do circuito do cristal

3.1.2 Circuito do LCD

O LCD é o periférico utilizado neste projeto para a visualização do comportamento do sistema de controle. Utilizam-se 10 pinos deste dispositivo para realizar a conexão com o microcontrolador e o recebimento de dados por parte do LCD. Esses pinos são conectados a porta B e C do microcontrolador, que são configurados como pinos de saída de dados. Para se obter uma imagem nítida e de fácil leitura, o pino VEE do LCD é conectado a um divisor de tensão por meio um potenciômetro, que possui a função de regular o contraste do display. Segue abaixo o esquema de ligação do LCD (Figura 3.4), envolvendo também o potenciômetro para ajuste de contraste.

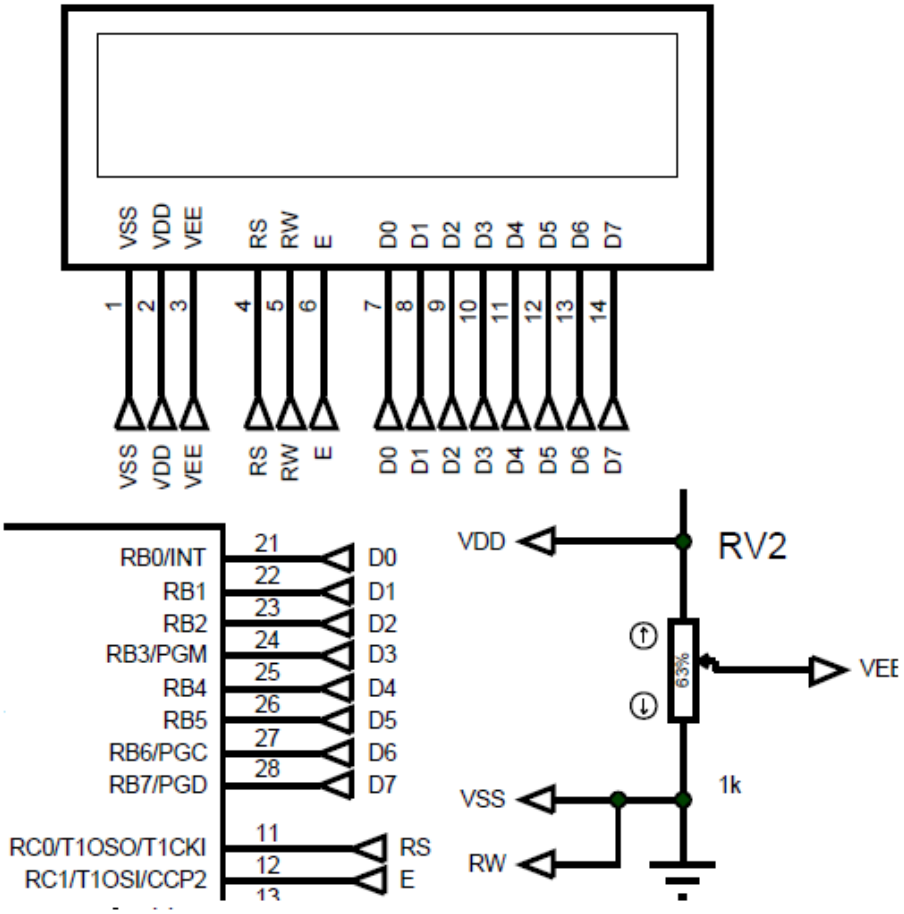

Figura 3.4 - Esquema de ligação do LCD

3.1.3 Circuito do relógio de tempo real (DS1307)

O relógio de tempo real (Figura 3.5) é um dispositivo que pode ser alimentado por uma fonte independente de energia. Necessita-se um cristal e uma fonte de alimentação externa, porém, caso o PIC sofra alguma queda de energia, o periférico não é prejudicado e se mantém em funcionamento. Esse dispositivo é conectado ao microcontrolador utilizando barramento de dados I2C, que é utilizado para conexão de dispositivos de alta velocidade com alguns dispositivos mais lentos, como é o caso do relógio de tempo real.

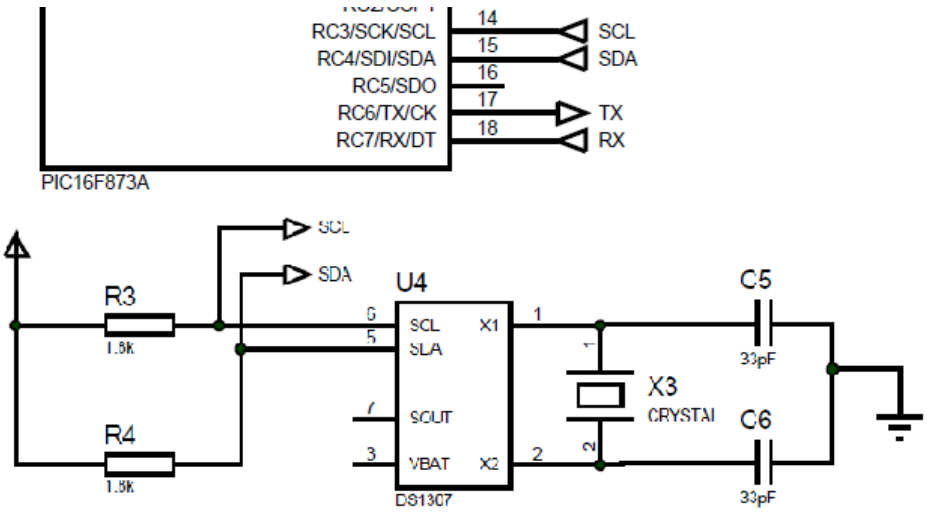

Figura 3.5 - Esquema de ligação do relógio de tempo real

Neste projeto, optou-se pela utilização do módulo RTC com I2C (Figura 3.6). Desta forma o esquema de ligação se torna mais simples, visto que não é necessária a inserção e um cristal, dos capacitores e das conexões para barramento de dados I2C, pois o módulo já vem com estes componentes embutidos, sendo assim é necessário apenas conectar os pinos SCL e SDA do módulo aos seus respectivos pinos do microcontrolador. Além disso, o módulo já vem com estrutura pronta para recebimento de bateria de lítio, garantindo um funcionamento independente por pelo menos 9 anos, com uma média de duração de 17 anos.

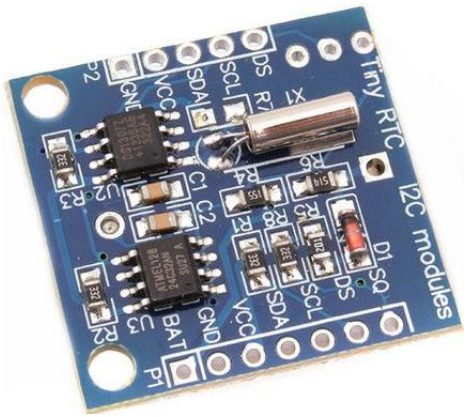

Figura 3.6 - Módulo RTC com I2C

3.1.4 Circuito do módulo Bluetooth

O módulo Bluetooth possui 6 pinos, os quais 2 deles são utilizados apenas para alterar configurações e os demais são necessários para o funcionamento do módulo. Deve-se conectar o pino de alimentação e o pino de terra de forma que se obtenha uma diferença de potencial de 5 V entre eles. O pino TX do módulo Bluetooth deve ser conectado ao RX do microcontrolador e o pino RX do módulo ao TX do microcontrolador. Porém, o pino RX do módulo Bluetooth suporta aproximadamente 3,3 V. Para alcançar esse valor, utiliza-se um divisor de tensão na saída do pino TX do PIC, como mostrado na Figura 3.7.

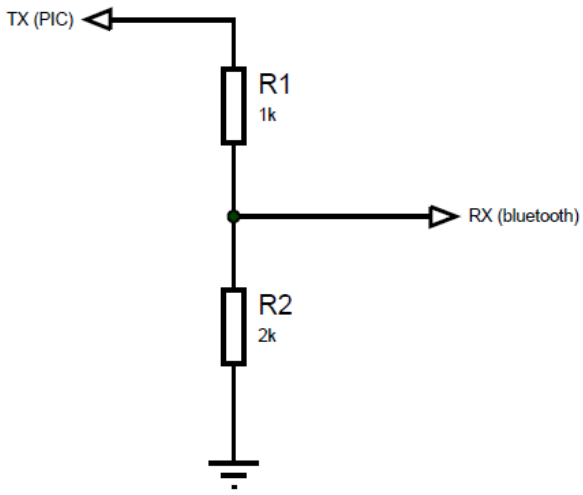

Figura 3.7 - Divisor de tensão

Sabe-se que o microcontrolador opera com diferença de potencial de 5 V. Verifica-se que o pino RX do módulo Bluetooth recebe uma alimentação de 3,3 V por meio das equações 3.1 e 3.2:

$$i = \frac{V}{Req} \tag{3.1}$$

Onde V é igual a 5 V e R_{eq} é igual a 3kΩ, sendo assim a corrente total i é igual a 1,67mA.

Utilizando a lei das malhas verifica-se que:

$$V_{rx} = V_{tx} - V_{R1} \tag{3.2}$$

Onde V_{tx} é a tensão de saída do pino Tx microcontrolador, V_{rx} é a tensão de entrada no pino Rx do módulo Bluetooth, e V_{R1} é a queda de tensão no resistor R1. Sendo assim, a tensão em Rx é igual a 3,3 V.

Os valores de R1 e R2 foram escolhidos como 1 kΩ e 2 kΩ, respectivamente, pois são suficientemente grandes de forma que a corrente que passa pelos pinos do microcontrolador e do módulo Bluetooth não seja maior que 20mA, podendo acarretar a queima do pino TX do microcontrolador.

3.1.5 Circuito de saída

A função do controlador desenvolvido é o acionamento de cargas utilizadas em sistemas de irrigação. Para isso, seria necessária a utilização de um circuito para acionamento de cargas as quais são utilizadas tensões mais elevadas, o que resulta em mais componentes a serem utilizados no protótipo. Devido ao fato de que a lógica de controle é a mesma, foram utilizados LED's, precedidos por resistores (Figura 3.8), como saídas do sistema, pois se pode visualizar facilmente o estado da saída, ou seja, se ela está energizada ou não.

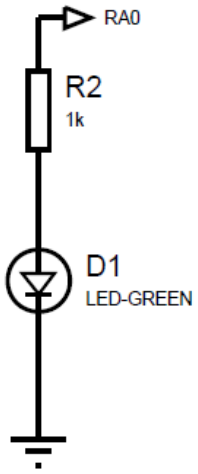

Figura 3.8 - Esquema de ligação do LED

3.2 Programação do microcontrolador

A programação do microcontrolador foi feita em linguagem C, utilizando o software CCS. Foram realizadas inclusões de bibliotecas como a *#include <lcd.h>* para o LCD e *#include <ds1307a.h>* para o relógio de tempo real. Desta maneira não é necessária a criação de novos códigos para a troca de informações com esses periféricos. Uma vez incluída a biblioteca, pode-se utilizar comandos já desenvolvidos e testados, melhorando a eficiência da programação e reduzindo a complexidade de código. Foi necessária a declaração das configurações de comunicação com o módulo Bluetooth, inserindo a velocidade de comunicação, bits de paridade, entre outros parâmetros. Também foi necessária a inserção de configurações para comunicação I2C, para que o relógio de tempo real opere corretamente. Essas configurações são mostradas abaixo:

```
#use rs232(baud=9600,parity=N, xmit=PIN_C6,rcv=PIN_C7,bits=8)
#use I2C(master, sda = PIN_C4, scl = PIN_C3)
```

Quando o microcontrolador recebe um comando informando que alguma saída deve ser acionada em determinada hora e por determinado espaço de tempo, essa informação é armazenada em variáveis de programa. Nesta aplicação esse comando é composto por 7 caracteres que são exemplificados na Tabela 3.1 e descritos em seguida.

Tabela 3.1 - Exemplo de comandos de parametrização

Caractere 1	Caractere 2	Caractere 3	Caractere 4	Caractere 5	Caractere 6	Caractere 7
Comando	**Hora**		**Minutos**		**Tempo**	
	Digito1	**Digito2**	**Digito1**	**Digito2**	**Digito1**	**Digito2**
0	1	5	1	5	1	0
1	2	2	0	0	1	5
2	0	8	4	5	4	5
3	1	3	1	3	x	x

O primeiro caractere representa o tipo de comando a ser enviado ao microcontrolador. Os caracteres 0 a 2 indicam que o comando ajustará parâmetros das saídas 0 a 2 do sistema, respectivamente. O caractere 3 indica que o comando ajustará parâmetros do periférico relógio de tempo real, que está conectado ao microcontrolador. O segundo, o terceiro, o quarto e o quinto caracteres representam o horário a ser ajustado para as saídas do sistema, ou para o relógio externo. O sexto e sétimo caracteres são caracteres que dizem respeito ao tempo de irrigação das

saídas do sistema. Caso se trate de um comando para o periférico, deve-se digitar quaisquer números, e eles serão ignorados.

Comandos inadequados, como por exemplo, comandos com caractere inicial igual a 5, ou os caracteres que dizem respeito as horas iguais a 8 e 4 (84horas) respectivamente, são ignorados.

Exemplo: ao receber o comando 2151245, o microcontrolador irá ajustar saída 2 do sistema para ser acionada as 15 horas e 12 minutos e permanecer ligada por 45 minutos.

O código que realiza este ajuste é mostrado na Figura 3.9, onde são utilizadas subfunções para que ele não se torne muito extenso, facilitando o entendimento do programa principal.

```
200         if(kbhit())
201         {
202           for(i=0;i<7;i++){
203              string[i] = getc();
204              palavra[i] = string[i] - '0';       // converte char para int
205              }
206            switch (string[0])
207            {
208              case '0': ajusta_saida0(palavra[1],palavra[2],palavra[3],palavra[4], palavra[5], palavra[6]);
209                 break;
210              case '1': ajusta_saida1(palavra[1],palavra[2],palavra[3],palavra[4], palavra[5], palavra[6]);
211                 break;
212              case '2': ajusta_saida2(palavra[1],palavra[2],palavra[3],palavra[4], palavra[5], palavra[6]);
213                 break;
214              case '3': ajusta_hora(palavra[1],palavra[2],palavra[3],palavra[4]);
215                 break;
216                 }
217         //printf(string);     // apenas p/ visualizacao no monitor
218         }
```

Figura 3.9 – Trecho de código para decodificação de comandos recebidos

As variáveis de programa são comparadas a todo instante com as variáveis armazenadas no periférico que indica a hora real (Figura 3.10). Quando o horário real for o mesmo que o horário armazenado nas variáveis de programa, a saída é acionada.

```
if(hora0 == h && minuto0 == m && flag0 == 0){tempo0 = tempo0 * 60; flag0 = 1;}
if(tempo0 == 0){led0 = 0; flag0 = 0;}

if(hora1 == h && minuto1 == m && flag1 == 0){tempo1 = tempo1 * 60; flag1 = 1;}
if(tempo1 == 0){led1 = 0; flag1 = 0;}

if(hora2 == h && minuto2 == m && flag2 == 0){tempo2 = tempo2 * 60; flag2 = 1;}
if(tempo2 == 0){led2 = 0; flag2 = 0;}
```

Figura 3. 10 – Trecho de código para acionamento das saídas do sistema

Para a contagem do tempo em que a saída deve permanecer acionada, é utilizado o *timer0* (Figura 3.11), que realiza um contagem de tempo independente do relógio externo. O *timer0* não é tão preciso quanto o periférico, porém, para contagem de apenas algumas horas ou minutos, pode-se utilizá-lo de forma eficiente.

```
68
69  //Tratamento de Interrupções
70  #int_timer0
71  void trata_tmr0 ()
72  {
73     set_timer0(131 + get_timer0());
74     cont++;
75     if(cont > 624)
76     {
77        if(flag0){led0=1;tempo0--;}
78        if(flag1){led1=1;tempo1--;}
79        if(flag2){led2=1;tempo2--;}
80        flaghora = 1;
81        cont = 0;
82        segundos++;
83     }
84  }
85
```

Figura 3.11 - Tratamento do timer0

Este projeto possui 3 saídas digitais que acionarão os LED's, por isso são utilizados 3 contadores de tempo chamados de *tempo0*, *tempo1* e *tempo2*, além de um contador adicional usado para mostrar a hora atual no LCD a cada segundo. Após uma saída ser acionada seu respectivo contador é ajustado com o valor, em segundos, que ela deve permanecer em nível alto. Esse contador tem seu valor decrementado em uma unidade a cada segundo, e coloca a saída em nível baixo novamente quando seu valor chega a 0. Desta forma, as saídas permanecerão acionadas no intervalo de tempo desejado, mesmo que posteriormente ocorra uma mudança no horário atual do periférico, como podem acontecer em ajustes de fuso horário.

3.3 Simulação e gravação

Depois de finalizada a parte de implementação de código em linguagem C, o próprio CCS é utilizado para compilar o programa e gerar um código hexadecimal correspondente, ou seja, a

programação realizada é transformada em linguagem de máquina. Utilizando esse código hexadecimal e o programa Proteus, pode-se simular o funcionamento do sistema. Sabe-se que o princípio de funcionamento do módulo Bluetooth é receber dados sem fio e enviá-los ao microcontrolador de forma como é feito na comunicação serial. Utiliza-se então o recurso do Proteus chamado virtual terminal, onde é possível digitar caracteres no teclado do computador, e o programa simula como se eles fossem enviados por meio da porta serial (pinos RX/TX) do microcontrolador. Na Figura 3.12 é mostrada uma simulação na qual a saída 2 acaba de ser parametrizada, recebendo um comando para ser acionada às 15 horas e 30 minutos e permanecer ligada por 10 minutos.

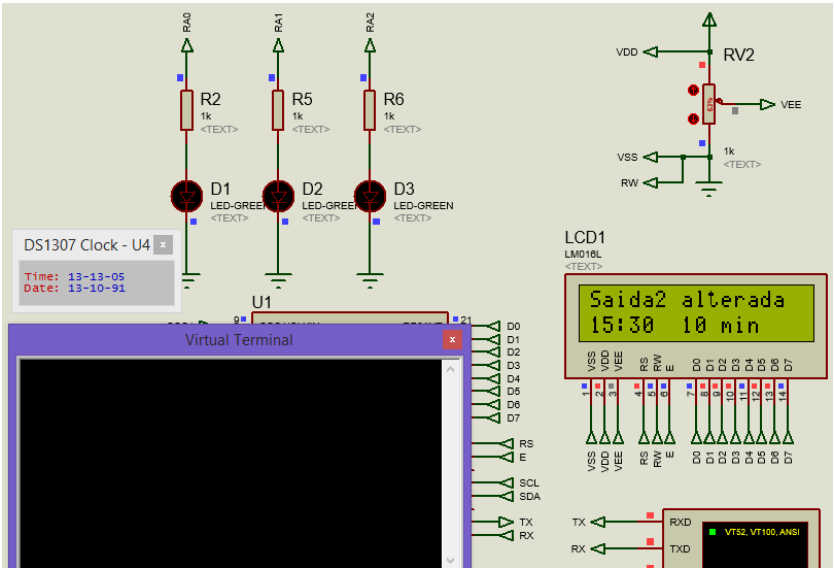

Figura 3.12 - Simulação no Proteus

Posteriormente, verificado que o funcionamento do sistema está de acordo com a necessidade da aplicação, o programa em formato hexadecimal é gravado no PIC, utilizando a gravadora MicroICD (Figura 3.13) e o software PicKit2.

Figura 3.13 - Gravadora MicroICD

O PIC é conectado à gravadora que por sua vez é conectada à porta USB do computador. O software PicKit2 (Figura 3.14) é gratuito e pode ser baixado no site oficial da Microchip. Faz-se então a importação do firmware para o software, sendo em seguida gravado no microcontrolador.

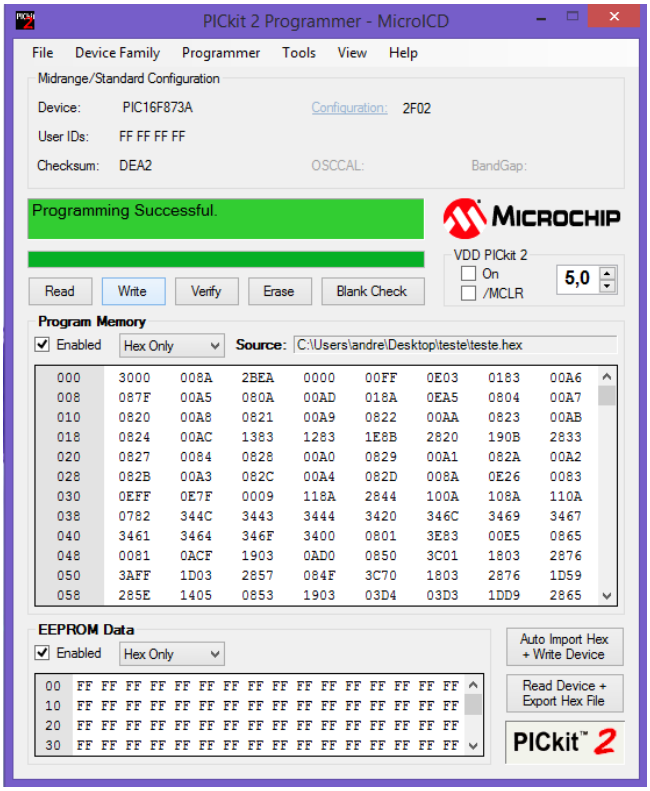

Figura 3.14 - Software para gravação de firmware (PicKit)

3.4 Desenvolvimento e funcionamento do aplicativo

O aplicativo é desenvolvido utilizando o software Android Studio. Pelo fato de o sistema operacional Android ser um projeto de código aberto, quando se instala a plataforma de

desenvolvimento, são baixados automaticamente alguns trechos de código, que podem ser posteriormente utilizados, como menus de navegação, códigos para criação de mapas, entre outros. A possibilidade de utilizar o recurso *drag and drop* faz com que o desenvolvedor possa criar o layout de forma rápida e eficaz. Como mostrado na Figura 3.15, tem-se uma parte da tela principal da plataforma de desenvolvimento, onde são inseridos, por exemplo, botões, caixas de texto, tabelas, entre outras opções da palheta.

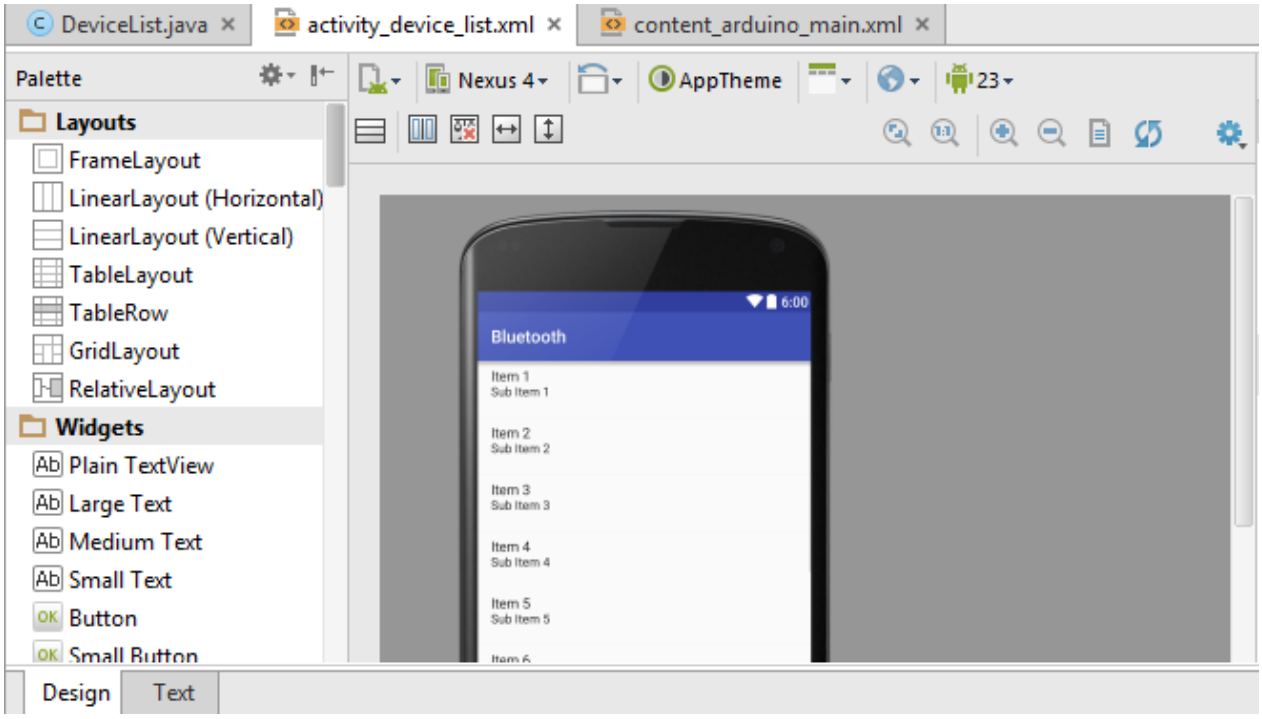

Figura 3.15 - Plataforma de desenvolvimento para aplicativos Android

Quando os botões são adicionados ao layout da página, seu código XML é automaticamente gerado e pode ser acessado clicando em "Text", como se observa na Figura 3.15. O desenvolvedor pode interpretar o código gerado e manipulá-lo textualmente. Um exemplo desta prática seria, por exemplo, se por alguma razão se queira inserir um botão A exatamente 10 cm abaixo de um botão B, não importando onde o botão B esteja. Enquanto os layouts são desenvolvidos por códigos XML, as funcionalidades do programa são baseadas na linguagem de programação Java. Como Java é uma linguagem de programação orientada a objetos, têm-se grandes vantagens no trabalho com interfaces com o usuário, pois pode-se aproveitar objetos e métodos já existentes para uma melhor manipulação dos dados.

Na tela principal deste aplicativo pede-se a permissão de ativar o Bluetooth, caso este não se encontre previamente ativado. Em seguida, é mostrada ao usuário uma lista com os dispositivos previamente pareados. O usuário deve escolher o dispositivo desejado fazer um clique simples sobre o campo com o nome e endereço do dispositivo. Após o clique do usuário, tenta-se estabelecer conexão sem fio do Android com o respectivo dispositivo escolhido. Caso esta tentativa não obtenha sucesso, aparece uma mensagem ao usuário informando que a conexão não foi estabelecida, provavelmente por não ter encontrado o dispositivo escolhido. Caso contrário, é efetuada a conexão e o aplicativo é direcionado para a tela para inserção de dados a serem enviados (Figura 3.16).

Figura 3.16 - Tela do aplicativo para envio de dados

Na caixa de texto mostrada no canto superior da tela, escrito “Digite uma string”, pode-se inserir uma *string* e enviá-la ao microcontrolador. Um botão aparecerá, logo após o clique na caixa de texto, e por meio deste é feita a confirmação de uma operação de atualização de horário, de um novo horário para acionar alguma saída ou um código inválido. Têm-se botões numerados de 0 a 9 que são utilizados para enviar um caractere separadamente.

4. RESULTADOS

4.1 Circuito elétrico

A parte elétrica do projeto produziu um resultado como esperado. O display de cristal líquido funciona corretamente, mostrando ao usuário algumas informações relevantes do sistema de controle. No circuito elétrico, este periférico é o componente que realiza a interação homem-máquina. A hora atual é exibida na tela utilizando o formato de horas, minutos e segundos, sendo atualizada a cada segundo. Caso o usuário altere os parâmetros do sistema, por meio do aplicativo Android, é mostrado no display uma mensagem indicando que estes foram alterados. Logo em seguida, são exibidos os parâmetros com seus respectivos valores atualizados. O potenciômetro conectado ao LCD é utilizado para ajuste de contraste do display, fazendo com que se possa alterá-lo se houver necessidade. O relógio de tempo real DS1307 armazena dados de horário e data, já com a correção para anos bissextos. Suas configurações são lidas ou alteradas por meio de subfunções do programa gravado no microcontrolador. O módulo Bluetooth tem a função de receber dados via transmissão sem fio e transmiti-los ao meio físico, por meio de conexão com os pinos do microcontrolador. Na Figura 4.1 pode-se observar o circuito elétrico completo, montado em protoboard.

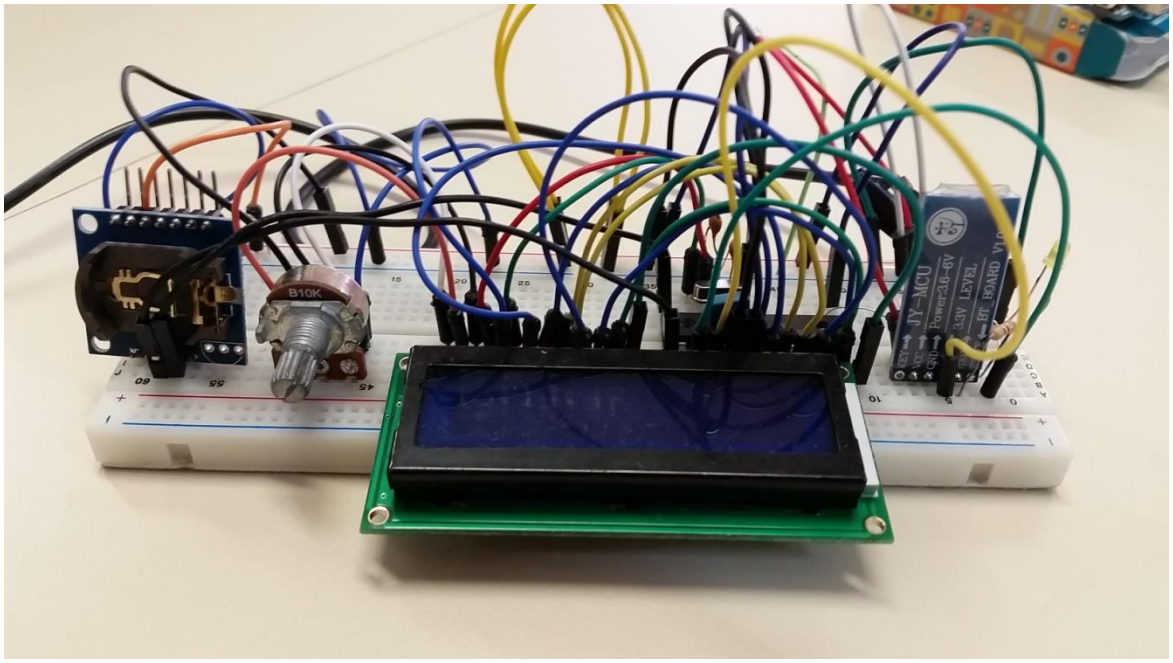

Figura 4.1 – Montagem do circuito elétrico em protoboard

4.2 Programação

O programa do microcontrolador realiza funções de envio e recebimento de dados ao periférico DS1307 e recebimento de dados do aplicativo Android. Quando há ocorrência de alguma mudança de parâmetros do sistema, o microcontrolador também tem a função de informar ao usuário as alterações realizadas, exibindo-as no display de cristal líquido. O programa armazena um conjunto de parâmetros para as saídas do sistema. Para cada uma das três saídas do sistema são armazenados o horário (horas e minutos) e o tempo de irrigação. O horário é comparado a todo instante com as informações relativas ao horário atual, obtidas a partir do periférico DS1307. Quando essa comparação resulta em uma resposta positiva, ou seja, o horário real corresponde ao horário para determinada saída ser acionada, o microcontrolador realiza o acionamento da respectiva saída. Há outra função que realiza a contagem de tempo que esta saída ficará energizada, conforme parametrizado pelo usuário. Sendo assim, quando o contador de tempo alcança o seu valor final, sua respectiva saída é desenergizada. Observando o Apêndice A pode-se verificar o código de uma forma mais detalhada.

4.3 Aplicativo

O aplicativo desenvolvido funciona conforme esperado, pode-se observar o código desenvolvido para a aplicação por meio do Apêndice B. Primeiramente, deve-se abrir o menu de configurações do telefone e ativar a opção Bluetooth. Em seguida é realizada a busca por dispositivos e o pareamento com o módulo Bluetooth utilizado no projeto. Iniciando o aplicativo, é exibida uma lista com todos os dispositivos pareados. Ao realizar um clique sobre o dispositivo desejado, é iniciada automaticamente uma conexão. Caso contrário, será exibido um aviso na tela informando que a conexão desejada não foi estabelecida, a aplicação retornará para a tela inicial. Quando a conexão é realizada é mostrada ao usuário a interface para envio de dados, com uma caixa de texto e dez botões. Por meio da caixa de texto pode-se inserir uma *string* e está é enviada ao microcontrolador. Outra opção é o envio de caracteres separadamente, por meio dos botões enumerados de 0 a 9. Independentemente da forma de envio escolhida, a aplicação exibe uma mensagem informando ao usuário qual a informação enviada, e logo em seguida retorna a tela para envio de dados.

O aplicativo desenvolvido pode ser instalado e utilizado por quaisquer dispositivos que disponibilizem de sistema operacional Android 2.2 ou versão superior, ou seja, é compatível com a maioria dos dispositivos encontrados atualmente (Android developers, 2016).

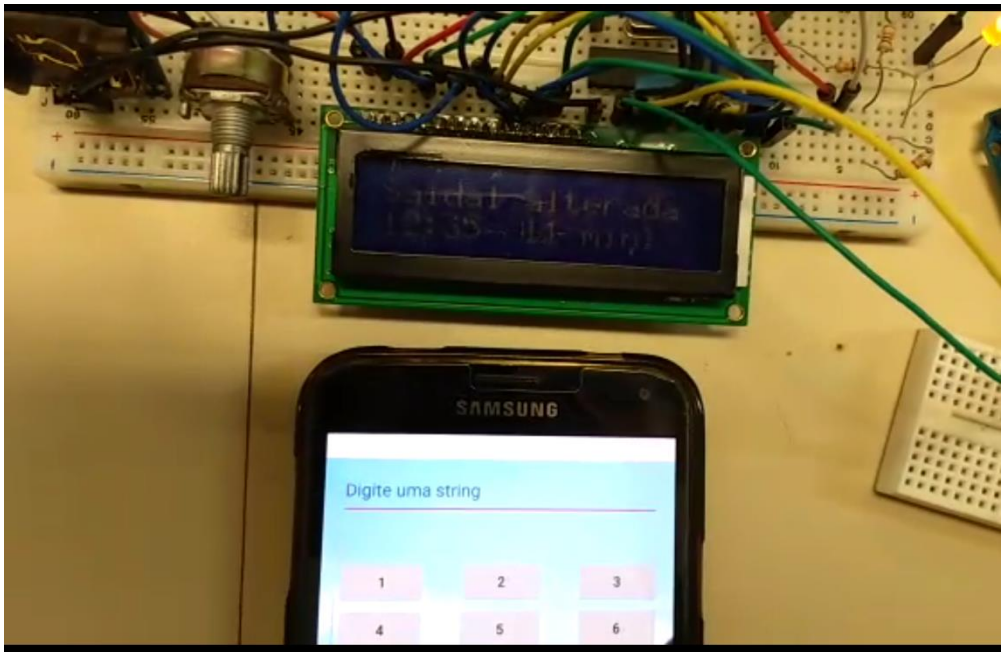

Figura 4.2 - Envio de dados do aplicativo para o microcontrolador

5. CONSIDERAÇÕES FINAIS E RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Tendo em vista a grande quantidade de processos produtivos que demandam o consumo de água, bem como a necessidade de utilização da mesma pela população, percebe-se a necessidade de uma boa gestão e planejamento, tanto de forma local como global, da utilização apropriada dos recursos hídricos existentes. O controlador proposto tem aplicação direta na agricultura, setor agroindustrial que mais consome água no Brasil e em todo o mundo. Por meio da utilização de microcontroladores pode-se obter um produto de baixo custo e consumo energético, que auxilia no controle e monitoramento dos sistemas de irrigação. O controlador pode ser facilmente parametrizado, bastando o usuário portar um smartphone com Bluetooth e o software desenvolvido no presente projeto instalado. Feita a parametrização, o sistema de controle poderá operar durante anos sem perder a precisão na contagem de tempo, visto que pequenas oscilações ou falhas na alimentação não interferem no bom funcionamento do controlador.

Para os próximos trabalhos sugere-se a inserção de uma funcionalidade no aplicativo para atualização de horário automático, ou seja, quando o aplicativo é conectado ao sistema de controle, ele já enviaria automaticamente o horário real. Pode-se verificar a viabilidade de utilização de dispositivos Bluetooth de maior potência (classe 1), pois eles podem realizar comunicação em distâncias maiores, chegando a até 100 metros. Outra ideia seria inserir neste mesmo produto uma forma alternativa de parametrização como ajuste por botões ou comunicação Wi-Fi, caso uma comunicação Bluetooth por meio de smartphone não seja possível no momento. Adicionalmente, caso se tenha optado pela inserção de uma comunicação Wi-Fi, seria possível a criação de um sistema supervisório, e então pode-se realizar um monitoramento no sistema de forma remota.

6. REFERÊNCIAS BIBLIOGRÁFICAS

Bibliografia

ANDROID. **Android**, 2016. Disponivel em: <https://www.android.com/>. Acesso em: 07 mar. 2016.

ANDROID developers, 2016. Disponivel em: <http://developer.android.com/sdk/index.html>. Acesso em: 07 mar. 2016.

ARDUINO Tutorial Hello Word. **Arduino**, 2016. Disponivel em: <https://www.arduino.cc/en/Tutorial/HelloWorld>. Acesso em: 17 mar. 2016.

FERREIRA, E. D. P.; ALVES, N. D. L. A. **braço articulado com controle proporcional de movimento comandado via bluetooth por um aplicativo desenvolvido para plataformas android**. São José dos Campos: [s.n.], 2013.

IBRAHIM, D. **Microcontroller Based Applied Digital Control**. [S.l.]: [s.n.], 2006.

MAIA, C.; NOGUEIRA, L. M.; PINHO, L. M. **Evaluating Android OS for embedded real-time-systems**. Polytechnic institute of porto. Porto, p. 10. 2010.

MARIAN, P. Electroschematics DS1307 Datasheet. **Electroschematics**, 2015. Disponivel em: <http://www.electroschematics.com/8886/ds1307-datasheet/>. Acesso em: 17 mar. 2016.

MICROCHIP forums. **Microchip**, 2016. Disponivel em: <http://www.microchip.com/forums/m44927.aspx>. Acesso em: 17 mar. 2016.

NETO, Á. P. SISTEMA DE IRRIGAÇÃO AUTOMATIZADO PARA AGRICULTURA, Belém, 2011. 1-6.

PLANETA sustentável. **Abril.com**, dez. 2012. Disponivel em: <http://planetasustentavel.abril.com.br/noticia/ambiente/populacao-falta-agua-recursos-hidricos-graves-problemas-economicos-politicos-723513.shtml>. Acesso em: 12 Julho 2015.

REBOUÇAS, A. D. C. B. Bahia Análise e Dados, Salvador, 2003. 341-345.

ROBERT L. BOYLESTAD, L. N. **Electronic Devices and Circuit Theory**. Eighth Edition. ed. [S.l.]: Pearson Education, 2002.

SOUZA, D. J. **Desbravando o PIC:** ampliado e atualizado para PIC 16F628A. 12. ed. São Paulo: Érica, 2009.

TORRES, F.; MARTINS, H. **Apostila Didática PICMinas - Sistemas microcontrolados**. [S.l.]: [s.n.].

UNITED Nations - Department of Economic and Social Affairs. **United Nations**, 2014. Disponivel em: <http://www.un.org/waterforlifedecade/scarcity.shtml>. Acesso em: 03 março 2016.

YEH, P.; GU, C. **Optics of liquid crytal displays**. second edition. ed. [S.l.]: [s.n.], 2010.

APÊNDICE A – Código de microcontrolador

```
// O microcontrolador recebe 7 caracteres de cada vez

// O primeiro indica a opção desejada:

//    0 - ajusta saída 0

//    1 - ajusta saída 1

//    2 - ajusta saída 2

//    3 - ajusta horário real

// Os quatro caracteres seguintes indicam as horas e os minutos, e os dois últimos indicam o tempo de irrigação em minutos

// Caso a opção 3 for selecionada, os dois últimos caracteres são ignorados

//

// Exemplo: 1144510 (a saída 1 é acionada as 14h45 e permance ligada por 1o minutos)


#include <16F873A.H>   // arquivo de definições do microcontrolador usado

#fuses HS,NOWDT,PUT,NOBROWNOUT,NOLVP     // bits de configuração


// Configurações do Projeto

#use delay(clock=20000000) // informa ao sistema o frequência de clock, para temporização


#use rs232(baud=9600,parity=N, xmit=PIN_C6,rcv=PIN_C7,bits=8)

//#use rs232(baud=9600, xmit=PIN_C6,rcv=PIN_C7)
```

```
#include<stdlib.h>


//variáveis
int8 day = 13,month = 10, year = 91,dow = 4,h=13,m=13,s=0, i =0;
char string[7];
int palavra[7];
int hora0, minuto0, hora1, minuto1, hora2, minuto2;
static long int cont;
static long int segundos = 0, tempo0, tempo1, tempo2;
boolean flaginicio = 1, flaghora=0, flag0 = 0, flag1 = 0, flag2 = 0;


// Definição e inicialização dos ports
#use   fast_io(a)
#use   fast_io(b)
#use   fast_io(c)


//para saber o endereço de memoria, olhar datasheet do pic
#byte   porta = 0x05
#byte   portb = 0x06
#byte   portc = 0x07
```

```
// Definições de Hardware


// Entradas


// Saídas
#bit led0 = porta.0
#bit led1 = porta.1
#bit led2 = porta.2



#bit rs = portc.0     // via do lcd que sinaliza recepção de dados ou comando
#bit enable = portc.1 // enable do lcd
#byte   DISPLAY = portb  //tem que ser a mesma porta escolhida dentro do arquivo lcd.h

// Bibliotecas
#include   <lcd.h>

#use I2C(master, sda = PIN_C4, scl = PIN_C3)

#include <ds1307a.h>
```

```
// Interrupções


//Tratamento de Interrupções
#int_timer0
void trata_tmr0 ()
{
  set_timer0(131 + get_timer0());
  cont++;
  if(cont > 624)
  {
    if(flag0){led0=1;tempo0--;}
    if(flag1){led1=1;tempo1--;}
    if(flag2){led2=1;tempo2--;}
    flaghora = 1;
    cont = 0;
    segundos++;
  }
}
```

```
// Sub-funções
void mostra_hora_lcd(){
      ds1307_get_time(h,m,s);
      limpa_lcd();
      comando_lcd(0x80);
      printf(escreve_lcd, "%02i:%02i:%02i", h, m,s);    // "%02i" é usado para saída com 2 digitos, e 0 no lugar dos espaços vazios

      flaghora = 0;
}
void inicio(){

  comando_lcd(0x80);
  escreve_lcd("LCD ligado");
  delay_ms(2000);
  limpa_lcd();
  ds1307_set_date_time(day,month,year,dow,h,m,s);

  flaginicio = 0;
}
```

```
void ajusta_hora(int h1, int h0, int m1, int m0){
  h = h1*10 + h0;
  m = m1*10 + m0;
  ds1307_set_date_time(day,month,year,dow,h,m,s);
  comando_lcd(0x80);
  escreve_lcd("Horario alterado! ");
  delay_ms(1000);
}

void ajusta_saida0(int h1, int h0, int m1, int m0, int t1, int t0){
  hora0 = h1*10 + h0;
  minuto0 = m1*10 + m0;
  tempo0 = t1*10 + t0;
  comando_lcd(0x80);
  escreve_lcd("Saida0 alterada ");
  comando_lcd(0xC0);
  printf(escreve_lcd, "%02i:%02i  %2ld", hora0, minuto0, tempo0);     // "%02i" é usado para saída com 2 digitos, e 0 no lugar dos espaços vazios
  escreve_lcd(" min");
  delay_ms(2000);
}
void ajusta_saida1(int h1, int h0, int m1, int m0, int t1, int t0){
```

```
  hora1 = h1*10 + h0;
  minuto1 = m1*10 + m0;
  tempo1 = t1*10 + t0;
  comando_lcd(0x80);
  escreve_lcd("Saida1 alterada ");
  comando_lcd(0xC0);
   printf(escreve_lcd, "%02i:%02i   %2ld", hora1, minuto1, tempo1);     // "%02i" é usado para saída com 2 digitos, e 0 no lugar dos espaços vazios
  escreve_lcd(" min");
  delay_ms(2000);
}
void ajusta_saida2(int h1, int h0, int m1, int m0, int t1, int t0){
  hora2 = h1*10 + h0;
  minuto2 = m1*10 + m0;
  tempo2 = t1*10 + t0;
  comando_lcd(0x80);
  escreve_lcd("Saida2 alterada ");
  comando_lcd(0xC0);
  printf(escreve_lcd, "%02i:%02i   %2ld", hora2, minuto2, tempo2);     // "%02i" é usado para saída com 2 digitos, e 0 no lugar dos espaços vazios
  escreve_lcd(" min");delay_ms(2000);
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                    Configurações do Pic                    *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


  void main()
{
    // configura os tris
    set_tris_a(0b00000000);
    set_tris_b(0b00000000);
    set_tris_c(0b10111000);

    // inicializa os ports
    porta=0b00000000;          // limpa porta
    portb=0b00000000;          // limpa portb
    portc=0b00000000;          // limpa portc



  inicializa_lcd();        // configura o lcd
```

```
  ds1307_init();      //inicializa o ds1307 (real time clock)

  // CONFIGURAÇÃO DO TIMER0
  setup_timer_0 (RTCC_INTERNAL | RTCC_8_BIT | RTCC_DIV_64);
  set_timer0(131);
  enable_interrupts(GLOBAL);//chave principal das interrupções
  enable_interrupts(int_timer0); // interrupção do TIMER 0

  PORT_B_PULLUPS(1);
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                        Rotina principal                        *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

  while(TRUE)           // rotina principal
  {

    RESTART_WDT(); // incia o watch-dog timer
    if(flaginicio){inicio();}

    if(flaghora){mostra_hora_lcd();}
```

```
if(hora0 == h && minuto0 == m && flag0 == 0){tempo0 = tempo0 * 60; flag0 = 1;}
if(tempo0 == 0){led0 = 0; flag0 = 0;}

if(hora1 == h && minuto1 == m && flag1 == 0){tempo1 = tempo1 * 60; flag1 = 1;}
if(tempo1 == 0){led1 = 0; flag1 = 0;}

if(hora2 == h && minuto2 == m && flag2 == 0){tempo2 = tempo2 * 60; flag2 = 1;}
if(tempo2 == 0){led2 = 0; flag2 = 0;}



 if(kbhit())
 {
   for(i=0;i<7;i++){
      string[i] = getc();
      palavra[i] = string[i] - '0';       // converte char para int
      }
    switch (string[0])
    {
      case '0': ajusta_saida0(palavra[1],palavra[2],palavra[3],palavra[4], palavra[5], palavra[6]);
         break;
      case '1': ajusta_saida1(palavra[1],palavra[2],palavra[3],palavra[4], palavra[5], palavra[6]);
```

```
            break;
          case '2': ajusta_saida2(palavra[1],palavra[2],palavra[3],palavra[4], palavra[5], palavra[6]);
            break;
          case '3': ajusta_hora(palavra[1],palavra[2],palavra[3],palavra[4]);
            break;
            }
      //printf(string);     // apenas p/ visualizacao no monitor
       }


   }


}
```

APÊNDICE B – Trecho de código do aplicativo para envio de dados

```
package com.example.simpleBluetooth;


        import java.io.IOException;

        import java.io.OutputStream;
        import java.util.UUID;

        import android.app.Activity;
        import android.Bluetooth.BluetoothAdapter;
        import android.Bluetooth.BluetoothDevice;
        import android.Bluetooth.BluetoothSocket;
        import android.content.Intent;
        import android.os.Bundle;
        import android.view.KeyEvent;
        import android.view.View;
        import android.view.View.OnClickListener;
        import android.view.View.OnKeyListener;
        import android.widget.Button;
        import android.widget.EditText;
        import android.widget.Toast;


public class ArduinoMain extends Activity {

    //Declare buttons & editText
    Button functionOne, functionTwo, button3, button4, button5, button6,
button7, button8, button9,button0;

    private EditText editText;

    //Memeber Fields
    private BluetoothAdapter btAdapter = null;
    private BluetoothSocket btSocket = null;
    private OutputStream outStream = null;

    // UUID service - This is the type of Bluetooth device that the BT module
is
    // It is very likely yours will be the same, if not google UUID for your
manufacturer
    private static final UUID MY_UUID = UUID.fromString("00001101-0000-1000-
8000-00805F9B34FB");

    // MAC-address of Bluetooth module
    public String newAddress = null;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_arduino_main);

        addKeyListener();

        //Initialising buttons in the view
```

```
        //mDetect = (Button) findViewById(R.id.mDetect);
        functionOne = (Button) findViewById(R.id.functionOne);
        functionTwo = (Button) findViewById(R.id.functionTwo);
        button3 = (Button) findViewById(R.id.button3);
        button4 = (Button) findViewById(R.id.button4);
        button5 = (Button) findViewById(R.id.button5);
        button6 = (Button) findViewById(R.id.button6);
        button7 = (Button) findViewById(R.id.button7);
        button8 = (Button) findViewById(R.id.button8);
        button9 = (Button) findViewById(R.id.button9);
        button0 = (Button) findViewById(R.id.button0);

        //getting the Bluetooth adapter value and calling checkBTstate
function
        btAdapter = BluetoothAdapter.getDefaultAdapter();
        checkBTState();


/*************************************************************************
*****************************************8
         *  Buttons are set up with onclick listeners so when pressed a method
is called
         *  In this case send data is called with a value and a toast is made
         *  to give visual feedback of the selection made
         */

        functionOne.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                sendData2('1');
                Toast.makeText(getBaseContext(), "caracter 1 enviado",
Toast.LENGTH_SHORT).show();
            }
        });

        functionTwo.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                sendData2('2');
                Toast.makeText(getBaseContext(), "caracter 2 enviado",
Toast.LENGTH_SHORT).show();
            }
        });
        button3.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                sendData2('3');
                Toast.makeText(getBaseContext(), "caracter 3 enviado",
Toast.LENGTH_SHORT).show();
            }
        });
        button4.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                sendData2('4');
                Toast.makeText(getBaseContext(), "caracter 4 enviado",
Toast.LENGTH_SHORT).show();
            }
        });
        button5.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
```

```
                sendData2('5');
                Toast.makeText(getBaseContext(), "caracter 5 enviado",
Toast.LENGTH_SHORT).show();
            }
        });
        button6.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                sendData2('6');
                Toast.makeText(getBaseContext(), "caracter 6 enviado",
Toast.LENGTH_SHORT).show();
            }
        });
        button7.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                sendData2('7');
                Toast.makeText(getBaseContext(), "caracter 7 enviado",
Toast.LENGTH_SHORT).show();
            }
        });
        button8.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                sendData2('8');
                Toast.makeText(getBaseContext(), "caracter 8 enviado",
Toast.LENGTH_SHORT).show();
            }
        });
        button9.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                sendData2('9');
                Toast.makeText(getBaseContext(), "caracter 9 enviado",
Toast.LENGTH_SHORT).show();
            }
        });
        button0.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                sendData2('0');
                Toast.makeText(getBaseContext(), "caracter 0 enviado",
Toast.LENGTH_SHORT).show();
            }
        });

    }

    @Override
    public void onResume() {
        super.onResume();
        // connection methods are best here in case program goes into the
background etc

        //Get MAC address from DeviceListActivity
        Intent intent = getIntent();
        newAddress = intent.getStringExtra(DeviceList.EXTRA_DEVICE_ADDRESS);

        // Set up a pointer to the remote device using its address.
        BluetoothDevice device = btAdapter.getRemoteDevice(newAddress);

        //Attempt to create a Bluetooth socket for comms
```

```
        try {
            btSocket = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e1) {
            Toast.makeText(getBaseContext(), "ERROR - Could not create
Bluetooth socket", Toast.LENGTH_SHORT).show();
        }

        // Establish the connection.
        try {
            btSocket.connect();
        } catch (IOException e) {
            try {
                btSocket.close();        //If IO exception occurs attempt to
close socket
            } catch (IOException e2) {
                Toast.makeText(getBaseContext(), "ERROR - Could not close
Bluetooth socket", Toast.LENGTH_SHORT).show();
            }
        }

        // Create a data stream so we can talk to the device
        try {
            outStream = btSocket.getOutputStream();
        } catch (IOException e) {
            Toast.makeText(getBaseContext(), "ERROR - Could not create
Bluetooth outstream", Toast.LENGTH_SHORT).show();
        }
        //When activity is resumed, attempt to send a piece of junk data ('x')
so that it will fail if not connected
        // i.e don't wait for a user to press button to recognise connection
failure
        sendData("x");
    }

    @Override
    public void onPause() {
        super.onPause();
        //Pausing can be the end of an app if the device kills it or the user
doesn't open it again
        //close all connections so resources are not wasted

        //Close BT socket to device
        try     {
            btSocket.close();
        } catch (IOException e2) {
            Toast.makeText(getBaseContext(), "ERROR - Failed to close
Bluetooth socket", Toast.LENGTH_SHORT).show();
        }
    }
    //takes the UUID and creates a comms socket
    private BluetoothSocket createBluetoothSocket(BluetoothDevice device)
throws IOException {

        return  device.createRfcommSocketToServiceRecord(MY_UUID);
    }

    //same as in device list activity
```

```
    private void checkBTState() {
        // Check device has Bluetooth and that it is turned on
        if(btAdapter==null) {
            Toast.makeText(getBaseContext(), "ERROR - Device does not support
Bluetooth", Toast.LENGTH_SHORT).show();
            finish();
        } else {
            if (btAdapter.isEnabled()) {
            } else {
                //Prompt user to turn on Bluetooth
                Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBtIntent, 1);
            }
        }
    }

    private void sendData3(int message){
        //byte[] msgBuffer = message.getBytes();
        try {
            //attempt to place data on the outstream to the BT device
            outStream.write(message);
        } catch (IOException e) {
            //if the sending fails this is most likely because device is no
longer there
            Toast.makeText(getBaseContext(), "ERROR - Device not found",
Toast.LENGTH_SHORT).show();
            finish();
        }
    }

    private void sendData2(char message){
        byte msgBuffer = ((byte)message);
        //byte[] msgBuffer = message.getBytes();
        try {
            //attempt to place data on the outstream to the BT device
            outStream.write(msgBuffer);
        } catch (IOException e) {
            //if the sending fails this is most likely because device is no
longer there
            Toast.makeText(getBaseContext(), "ERROR - Device not found",
Toast.LENGTH_SHORT).show();
            finish();
        }
    }

    // Method to send data
    private void sendData(String message) {
        byte[] msgBuffer = message.getBytes();

        try {
            //attempt to place data on the outstream to the BT device
            outStream.write(msgBuffer);
        } catch (IOException e) {
            //if the sending fails this is most likely because device is no
longer there
            Toast.makeText(getBaseContext(), "ERROR - Device not found",
```

```
Toast.LENGTH_SHORT).show();
            finish();
        }
    }
    public void addKeyListener() {

        // get edittext component
        editText = (EditText) findViewById(R.id.editText1);

        // add a keylistener to keep track user input
        editText.setOnKeyListener(new OnKeyListener() {
            public boolean onKey(View v, int keyCode, KeyEvent event) {

                // if keydown and send is pressed implement the sendData
method
                if ((keyCode == KeyEvent.KEYCODE_ENTER) && (event.getAction()
== KeyEvent.ACTION_DOWN)) {
                    //I have put the * in automatically so it is no longer
needed when entering text
                    sendData('*' + editText.getText().toString());
                    Toast.makeText(getBaseContext(), "enviando string",
Toast.LENGTH_SHORT).show();

                    return true;
                }

                return false;
            }
        });
    }

}
```