CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Paulo Mellin Gimenes

Desenvolvimento de um *Front-end* Web para a plataforma Raspberry Pi

Americana, SP

2015

CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Paulo Mellin Gimenes

Desenvolvimento de um *Front-end* Web para a plataforma Raspberry Pi

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do Prof. Me. Rossano Pablo Pinto

Área de concentração: engenharia de software

Americana, SP

2015

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

Gimenes, Paulo Mellin
G399d Desenvolvimento de um Front-end web para a plataforma Raspberry Pi. / Paulo Mellin Gimenes. – Americana: 2015.
49f.

Monografia (Graduação em Tecnologia em Análise e Desenvolvimento de Sistemas). - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza.
Orientador: Prof. Me. Rossano Pablo Pinto

1.Engenharia de software I. Pinto, Rossano Pablo II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana.

CDU: 681.3.05

Paulo Mellin Gimenes

Desenvolvimento de um *Front-end* Web para a plataforma Raspberry Pi

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo CEETEPS/Faculdade de Tecnologia – Fatec/ Americana.
Área de concentração: engenharia de software.

Americana, 07 de dezembro de 2015.

Banca Examinadora:

Rossano Pablo Pinto (Presidente)
Mestre
Faculdade de Tecnologia de Americana

Eduardo Antonio Vicentini (Membro)
Mestre
Faculdade de Tecnologia de Americana

Rogério Nunes de Freitas (Membro)
Especialista
Faculdade de Tecnologia de Americana

Resumo

Este trabalho tem como objetivo propor uma solução para o controle dos pinos de entrada e saída - GPIO(*General Purpose Input/Output*) - do Raspberry Pi através do desenvolvimento de um *Front-end* web com os frameworks Django e jQuery para, assim, facilitar ao usuário com pouco ou nenhum conhecimento de técnicas de programação desenvolver soluções utilizando este recurso.

Palavras-chave: Desenvolvimento de sistemas web, Raspberry Pi, Django, jQuery, AJAX.

Abstract

This monography aims to propose a solution to control Raspberry Pi's GPIO(General Purpose Input/Output pins) through the development of a web front-end with the frameworks Django and jQuery to facilitate the user with little or no programming knowledge techniques to develop solutions using this feature.

Keywords: Web systems development, Raspberry Pi, Django, jQuery, AJAX.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AJAX	*Asynchronous Javascript and XML*
app	*Application*
API	*Application Programming Interface*
ID BCM	*Broadcom SOC channel*
DRY	*Don't repeat yourself*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
ER	Engenharia de Requisitos
GPIO	*General Purpose Input/Output*
GPS	*Global Positioning System*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
LED	*Light Emitting Diode*
PC	*Personal Computer*
PWM	*Pulse Width Modulation*
RPi	Raspberry Pi
SPI	*Serial Peripheral Interface*
SoC	*System On Chip*
UART	*Universal Asynchronous Receiver/Transmitter*
UML	*Unified Modelling Language*

Sumário

Introdução

As plataformas computacionais embarcadas estão presentes em larga escala no cotidiano e, com o menor custo dos componentes eletrônicos, suas influências na humanidade tendem sempre a aumentar. Pode-se enxergá-las de modo abrangente em elementos da vida, como por exemplo em celulares, nos transportes públicos com sistemas embarcados para pagamento de passagem, automóveis com os sistemas GPS (*Global Positioning System*) e sistemas audiovisuais automotivos, nas redes de computadores com roteadores, *switches* e pontos de acesso Wi-Fi, modens, receptores de sinal digital, *Smart Tvs*, ou seja, uma lista grande de exemplos que por si só mostram a importância dos sistemas embarcados na vida moderna (CARRO; WAGNER, 2003).

Uma grande vantagem da popularização dos sistemas embarcados é o fácil acesso a plataformas para o "usuário comum" com vastas APIs(*Application Programming Interface*) e documentação para aprender e desenvolver soluções que um sistema embarcado pode realizar. Atualmente as plataformas mais difundidas vão desde microcontroladoras de *hardware* livre como Arduino(ARDUINO, 2015) até plataformas computacionais completas como a Beaglebone(BEAGLEBOARD.ORG, 2015) e a Raspberry Pi, esta última utilizada para este trabalho de graduação.

O Raspberry Pi(RPi) é uma plataforma computacional completa do tamanho de um cartão de crédito tendo como um de seus principais recursos a possibilidade de se conectar ao mundo através de GPIO (*General Purpose Input/Output*) que são pinos digitais programáveis de entrada e saída em que é possível conectar sensores, microcontroladoras, leds e outros elementos externos (RPI FOUNDATION, 2015c).

Através do conhecimento de técnicas de programação e eletrônica básica é possível para qualquer pessoa operar os pinos da GPIO. Entretanto, para os que ainda não tem este conhecimento, mas desejam utilizar a GPIO, é uma tarefa árdua para conseguir realizar qualquer tipo de uso dos pinos.

Por isso, este trabalho de graduação tem como objetivo desenvolver um *front-end* gráfico para web oferecendo uma maneira mais fácil de operar os pinos da GPIO do Raspberry Pi sem conhecimento profundo em técnicas de programação. Vale ressaltar que a solução proposta ainda requer um conhecimento básico de eletrônica.

No Capítulo 1 é traçado um panorama sobre plataformas computacionais que existem para os estudantes destinadas a prototipação de projetos eletrônicos e programação, já no Capítulo 2 são demonstrados detalhes sobre o Raspberry Pi bem como exemplos de como acessar os pinos da GPIO.

O Capítulo 3 trata sobre a concepção, visão geral, análise e desenvolvimento do **ProtoPi**, *front-end* criado para operar os pinos da GPIO. Em sequência no Capítulo 4 é dado um exemplo de uso prático do **ProtoPi**.

Finalmente, nas Considerações Finais se traça um panorama geral do projeto e expectativas para a evolução da solução proposta (**ProtoPi**)

1 Referencial Teórico

Com a evolução dos equipamentos associados às tecnologias da informação é possível perceber que há um afastamento dos estudantes das ciências da computação com o desenvolvimento de aplicações que interajam com o mundo real. Todo o estudo é voltado para o alto nível de computação em que as interações externas são o pressionar de uma tecla do teclado, o clique de um mouse e o toque de uma tela.

Com a redução constante de custos para componentes eletrônicos, houve o surgimento de plataformas para prototipação de projetos que unem o mundo físico com processamento computacional em que um dos objetivos é eliminar uma lacuna no ensino de informática. Essas plataformas podem ser baseadas em microcontroladores e/ou plataformas computacionais completas em apenas um chip, conhecidas como *System on a Chip*(SoC).

As próximas seções abordam este universo que cada vez mais se aproxima dos estudantes de computação, bem como as principais diferenças entre os dispositivos e também sobre a arquitetura de processador que estes elementos utilizam.

1.1 Microcontroladores

Os microcontroladores são elementos computacionais utilizados no controle de processos lógicos (TANENBAUM, 2007).

Para entender a presença desses elementos computacionais no cotidiano das pessoas, é importante relembrar de alguns exemplos de itens bem conhecidos que utilizam microcontroladores:

- **Eletrodomésticos**: micro-ondas, lavadouras de roupa, geladeiras, secadoras de roupa.
- **Periféricos de computadores**: placa de rede, placa de vídeo, impressoras e webcams.
- **Aparelhos de reprodução de imagens**: CD, DVD e BluRay players, televisores e conversores de sinal digital/analógico.
- **Dispositivos de vendas**: Máquinas de cartão de débito/crédito.
- **Brinquedos**: Bonecos de ação, bonecas, carrinhos de controle remoto.

Basicamente, todo microcontrolador tem um processador, memória e I/O (entrada e saída). As funções de entrada e saída dos microcontroladores normalmente incluem detectar botões, interruptores, controle de luz, sons e monitores(TANENBAUM, 2007).

O uso de microcontroladores em projetos eletrônicos há alguns anos atrás não era trivial para profissionais da tecnologia da informação e para o público em geral. Com o surgimento de algumas plataformas baseadas em microcontroladores como o Arduino, hoje é possível encontrar uma grande variedade de projetos desenvolvidos por pessoas que não possuem um vasto conhecimento em eletrônica. A próxima seção explora duas dessas plataformas.

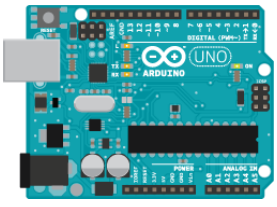
1.1.1 Principais plataformas baseadas em microcontroladores

As plataformas baseadas em microcontroladores não executam um sitema operacional e geralmente executam um programa por vez em *loop*. Existe uma vasta gama de plataformas baseadas em microcontroladores, duas principais delas são abordadas a seguir.

Arduino

Arduino é uma placa *open-source* para ser ampla e facilmente usada. As placas Arduino são capazes de ler entradas de sensores diversos e processarem uma saída. Através de conjuntos de instruções, é possível dizer ao microcontrolador ações que devem ser executadas (ARDUINO, 2015). Na Figura 1 é demonstrado o modelo mais popular do Arduino (UNO) que tem o custo aproximado de € 20.

Figura 1 – Arduino UNO



Fonte: https://www.arduino.cc/ acesso em 06/11/2015 às 14h30

Os sensores e atuadores podem ser conectados ao Arduino por meio de elementos conhecidos como *shields*. Esses elementos oferecem uma maneira simples de interação com os diversos sensores e atuadores disponíveis para a plataforma.

Para o desenvolvimento, o Arduino oferece uma API (*Application Programming Interface*) que deve ser utilizada através da IDE(*Integrated Development Environment*) "Arduino Software" para a programação e compilação dos códigos fonte, bem como o envio dos programas para a placa.

As especificações técnicas do arduino (modelo UNO) são detalhadas na Tabela 1.

Tabela 1 – Especificações técnicas Arduino.

Microcontrolador	ATmega328
Tensão para operação	5V
Input Voltage (recomendável)	7-12V
Input Voltage (limite)	6-20V
Digital I/O Pins	14
PWM Digital I/O Pins	6
Analog Input Pins	6
Corrente contínua *I/O Pin*	40 mA
Corrente contínua *3.3V Pin*	50 mA
Flash Memory	32 KB
Flash Memory para *Bootloader*	0.5 KB
SRAM[1]	2 KB
EEPROM[2]	1 KB
Clock Speed	16 MHz

Fonte: Arduino (2015)

Verifica-se pela Tabela 1 que o Arduino não tem o poder computacional de um Raspberry Pi (descrito no Capítulo 2), porém ele atende perfeitamente a protótipos eletrônicos dedicados a uma função específica. É necessário se executar um programa por vez, mas com os pinos analógicos e digitais é simples e pedagógico para a criação de projetos eletrônicos e robóticos.

Na prototipação de projetos eletrônicos com Arduino é necessário ter conhecimento em eletrônica, solda e/ou manipulação de uma protoboard e componentes eletrônicos como capacitores e resistores. Para atender a necessidade de estudantes que ainda não sabem soldar ou manipular componentes eletrônicos pela protoboard, surgiu o littleBits, solução abordada a seguir.

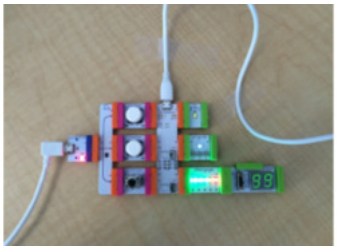
littleBits

Assim como o Arduino, o littleBits é uma placa *open-source* de uso fácil para prototipação de projetos eletrônicos. A grande diferença é que os projetos que a envolvem podem ser montados através de blocos, exatamente como um Lego (LITTLEBITS ELECTRONICS, 2015).

Na Figura 2 é ilustrado um projeto utilizando littleBits. É possível notar que esta plataforma dispensa o uso de solda, conectores, elementos eletrônicos, bem como o uso de *protoboard*.

Na Figura 3 é possível verificar que os componentes eletrônicos são utilizados como blocos para prototipação no littleBits e os mesmos dispensam de soldas ou protoboard por utilizar conectores magnéticos.

Figura 2 – Projeto utilizando littleBits



Fonte: http://littlebits.cc/ acesso em 06/11/2015 às 14h30

Figura 3 – Componente eletrônicos para littleBits

Fonte: http://littlebits.cc/ acesso em 06/11/2015 às 14h30

Pode-se observar na Figura 3 a presença de diversos componentes como um resistor sensível a força, que pode ser pressionado com os dedos e um *display* com 14 segmentos além de diversos outros componentes.

A próxima seção aborda plataformas mais completas denominadas SoC(*System on a Chip*).

1.2 *System on a Chip*(SoC)

System on a Chip(SoC) são plataformas computacionais em um único chip completas que possuem tudo o que um PC convencional pode fornecer. Geralmente estas plataformas utilizam um sistema operacional, além de possibilitar a conexão de sensores e processamento

de entradas e saídas através dos dados fornecidos por esses sensores(RAJSUMAN, 2000).

A grande diferença dessa plataformas para baseadas em microcontroladores é exatamente a possibilidade de, além de realizar projetos eletrônicos, desempenhar funções que um PC convencional.

1.2.1 Principais Plataformas SoC

Dentre as diversas plataformas SoC, duas destacam-se por sua popularidade: o Raspberry Pi explorada no Capítulo 2 e o BeagleBone explorado a seguir.

BeagleBone

BeagleBone é um pequeno computador com a capacidade de um PC Desktop e que pode ser utilizada com sistemas operacionais Windows, distribuições Linux e Mac OS. Também, possui funções de entrada e saída e utiliza um microprocessador AM335x 720MHz ARM (BEAGLEBOARD.ORG, 2015).

Figura 4 – BeagleBone Black

Fonte: http://beagleboard.org/bone acesso em 06/11/2015 às 14h30

É oferecido também pelo BeagleBone uma interface para entrada e saída: GPIO. Esta interface ofere 46 pinos programáveis para entrada ou saída.

O custo mínimo do BeagleBone é de US$ 89 e para programação é oferecido nativamente em seus sistemas operacionais uma biblioteca em JavaScript chamada de BoneScript (BEAGLEBOARD.ORG, 2015).

1.3 Plataformas Web

Para facilitar o desenvolvimento da ferramenta apresentada no Capítulo 3, foram necessárias ferramentas para o desenvolvimento do back-end e front-end do projeto. A fim de elucidar esses *Frameworks*, esta seção apresenta seus benefícios.

A principal justificativa de se utilizar um *framework* web é o desenvolvimento ágil com bibliotecas comuns a diversos sistemas o que evita gasto de tempo programando esses elementos comuns, ou seja, essas ferramentas oferecem os requisitos não funcionais de um sistema web, conforme definido no Capítulo 3, seção 3.2.

Dentre as ferramentas existentes vale ressaltar duas delas: uma para back-end (Django) e outra para front-end (jQuery).

1.3.1 Django

O *framework* web Django utiliza a plataforma de desenvolvimento Python[3]. Segundo a *Django Software Foundation* em sua página principal de seu *website*[4], a melhor definição para esta ferramenta é:

> *Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.(DSF, 2015b)*

Conforme a citação, é possível ver que o principal objetivo do Django é impedir que o programador gaste o tempo "'reinventando a roda", ou seja, já contém elementos comuns a diversos sistemas web e que podem ser reaproveitados em projetos diversos. Vale a pena ressaltar que o Django é voltado para escrita de códigos limpos de maneira rápida.

O *framework* Django permite o desenvolvimento em camadas, para assim, separar as lógicas de negócio, das classes modeladas e do design para o navegador de internet (mais detalhes no Apêndice A).

1.3.2 jQuery

Para o *front-end*, ou seja, a parte que é visualizada pelo usuário em seu navegador de internet, o jQuery é uma ferramenta que contém diversos métodos JavaScript para agilizar a montagem dos layouts, inserir ações nas páginas (como formulários dinâmicos, validações no cliente) e a submissão de formulários sem a necessidade de troca de páginas utilizando AJAX (*Asynchronous Javascript and XML*) (JQUERY, 2015).

O AJAX é uma tecncnologia que permite através de códigos-fontes JavaScript e XML a possibilidade de interações com o *back-end* de maneira que as requisições feitas do cliente para o servidor não possua a necessidade da troca de páginas (W3C, 2015).

A seguir no Capítulo 2 uma abordagem do Raspberry Pi, objeto de estudo deste trabalho de graduação.

[3] https://www.python.org/
[4] https://www.djangoproject.com/

2 Raspberry Pi

O Raspberry Pi, segundo sua fundação mantenedora (RASPBERRY PI FOUNDATION), é um computador de baixo custo que pode ser conectado a um monitor ou televisor com um mouse e teclado, assim, utilizando-o como um PC. É capaz de realizar as operações de um computador *desktop* além de interagir com o mundo externo através de seus pinos digitais de entrada e saída (GPIO) (RPI FOUNDATION, 2015c).

Na Figura 5 é oferecida uma relação entre os modelos existentes do Raspberry Pi, do mais antigo ao mais recente.

Figura 5 – Relação de modelos do Raspberry Pi.

Raspberry Pi:	Model A+	Model B	Model B+	2, Model B
Chip:	Broadcom BCM2835			Broadcom BCM2836
Processador	ARMv6 single core			ARMv7 quad core
Clock	700 MHz			900 MHz
Tensão e corrente para operação	600mA @ 5V			650mA @ 5V
GPU	Dual Core VideoCore IV Multimedia Co-Processor			
Tamanho	65x56mm	85x56mm		
Memória RAM	256 MB SDRAM @ 400 MHz	512 MB SDRAM @ 400 MHz		1 GB SDRAM @ 400 MHz
Armazenamento	SD Card		Micro SD Card	
GPIO	26		40	
USB 2.0	1	2	4	
Ethernet	None	10/100mb Ethernet RJ45 Jack		
Audio	Multi-Channel HD Audio over HDMI, Analog Stereo from 3.5mm Headphone Jack			

Fonte: Autoria Própria

É possível notar que com o lançamento dos modelos houve uma melhora em termos de componentes e desempenho, a memória RAM era de 256 MB no primeiro modelo, subiu para 512 MB nos modelos B e, no modelo mais recente, já se oferece 1 GB. Pode-se verificar que o número de portas USB também aumentou, e surgiu a capacidade de se conectar o Raspberry Pi à rede cabeada através da adição do conector RJ45 para uma rede **Ethernet**.

Para este capítulo é utilizado como referência o modelo B+ apresentado a seguir na Figura 6. Mais detalhes sobre seus componentes podem ser verificados no Apêndice B.

Figura 6 – Raspberry Pi: um computador do tamanho de um cartão de crédito(Modelo B+).

Fonte: Autoria Própria

O modelo B+ tem processador da arquitetura ARMv6 e o *clock* de 700MHz. Existe a possibilidade de se realizar o procedimento de *overclock*, porém é recomendado que seja feito com cautela para não causar super aquecimento na placa e danificar o dispositivo, já a quantidade de RAM(Memória de Acesso Aleatório) é de 512MB.

Este modelo pode ter seu poder computacional comparável a um PC *Desktop* com processador Pentium 4 e 512 MB de RAM, o que é um desempenho significativo dado o tamanho e custo de dispositivo, bem como sua possibilidade de interagir com o mundo real.

2.1 Sistema Operacional

O sistema operacional deve ser armazenado em um cartão Micro SD, que também serve de memória de armazenamento para o RPi.

O *Raspbian*, sistema operacional baseado na distribuição GNU/Linux Debian, [1] foi o escolhido para este trabalho como referência. De acordo com o site oficial do sistema operacional[2]:

> *Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, precompiled software bundled in a nice format for easy installation on your Raspberry Pi. The initial build of over 35,000 Raspbian packages, optimized for best performance on the Raspberry Pi, was completed in June of 2012. However, Raspbian is still under active development with an*

[1] http://www.debian.org
[2] http://www.raspbian.org/

> *emphasis on improving the stability and performance of as many Debian packages as possible.*

2.2 Usos do Raspberry Pi

O uso do Raspberry Pi pode ser destinado a diversas funções, muitas delas feitas por um PC *Desktop* comum. Dentro deste amplo espectro de utilizações duas abordadas a seguir se destacam.

2.2.1 Educação

Através do baixo custo do RPi é possível para escolas com baixo orçamento para investir em tecnologia da informação com gastos menores com relação a aquisição de PCs tradicionais (RPI FOUNDATION, 2015c).

Além da vantagem de se gastar menos com o RPi, é possível para os estudantes expandirem o paradigma da programação para interações diferentes de uma tela, mouse e teclado, envolvendo, assim, sensores e componentes que interagem com o mundo externo de maneiras diversas (RPI FOUNDATION, 2015c).

2.2.2 Internet das Coisas (IoT)

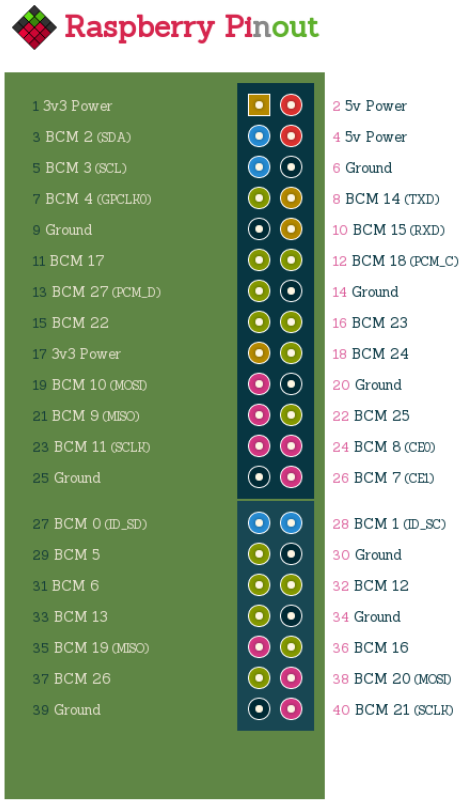
A principal definição para "Internet das Coisas"(*Internet of Things* - IoT) é a presença de diversos objetos que não são necessariamente plataformas computacionais completas na rede como sensores, telefones móveis,televisores e eletrodomésticos em que estes dispositivos interagem entre si por objetivos comuns. Inquestionavelmente, a principal vertente da ideia é o alto impacto no cotidiano e comportamento do usuário comum tanto no ambiente de trabalho como doméstico (ATZORI; IERA; MORABITO, 2010).

Por fornecer a GPIO, abordada na seção a seguir, pode-se interligar o Raspberry Pi com outros elementos de nosso cotidiano, principalmente com a conexão de sensores para assim atuar com situações da vida real e, portanto, aplicar o conceito de IoT.

2.3 GPIO

Para conectar o RPi a sensores, microcontroladores, periféricos, displays e LEDs, ou seja, conectá-lo ao mundo real, é necessário utilizar a GPIO(*general purpose input/output*, pinos digitais de entrada e saída para uso geral) que são pinos disponíveis para se conectar sensores e módulos ao RPi com o ambiente externo. Na Figura 7 é possível visualizar os pinos da GPIO.

Figura 7 – GPIO.



Fonte: Rpi Foundation (2015b)

Traçando um primeiro panorama, é possível notar que a GPIO no modelo B+ é composta por 40 pinos sendo que 4 pinos são reservados exclusivamente para alimentação (2 de 3.3 V e 2 de 5V), e 8 para terra (*Ground*).

Os pinos do Raspberry Pi seguem um padrão em que se pode referenciá-los no código fonte pelo seu ID BCM[3] ou pela sua posição física(*Board*) em que a contagem é feita de maneira vertical olhando para duas colunas de 20 pinos cada. Por exemplo, na Figura 7, que apresenta o RPi na posição vertical com os pinos da GPIO no lado direito, a numeração é feita da esquerda para a direita e de baixo para cima.

Mais detalhes sobre a GPIO podem ser vistos no Apêndice C.

2.4 Acesso aos pinos da GPIO

Para utilizar a GPIO do Raspberry Pi, o sistema operacional fornece diretórios como forma de abstração, ou seja, uma maneira de se acessar os pinos por intermédio de escrita e leitura de arquivos.

[3] *Broadcom SOC channel*

Além do acesso aos pinos via arquivos, também, é oferecido nativamente uma biblioteca Python para acesso à GPIO. As próximas seções tratam dessas duas maneiras.

2.4.1 Diretórios do Sistema Operacional

Os pinos da GPIO são abstraídos no Raspbian por meio de arquivos no diretório "/sys". Para manipulá-los deve-se acessar o seguinte caminho:

```
/sys/class/gpio/
```

Após ter acessado o diretório correto, deve-se escrever no arquivo "export" o id BCM do pino que deseja-se trabalhar, como exemplo são utilizados os ids "4" e "7":

```
echo "4" > /sys/class/gpio/export
echo "7" > /sys/class/gpio/export
```

Após a definição dos pinos que serão trabalhados através da escrita no arquivo "export" o kernel cria diretórios nomeados como "/sys/class/gpio/gpio4/" e "/sys/class/gpio/gpio7/" (note que isso é um padrão de nome "gpio+id"). Os pinos podem ser usados como entrada ou saída, por isso, deve-se definir sua direção através da escrita no arquivo "direction" presente nos diretórios criados. Intuitivamente, deve-se escrever "in" para entrada ou "out" para saída. No exemplo a seguir o pino id "4" é destinado para saída e o de id "7" para entrada:

```
echo "out" > /sys/class/gpio/gpio4/direction
echo "in" > /sys/class/gpio/gpio7/direction
```

No caso do pino destinado à saída (id "4"), define-se seu estado lógico, ou seja, se a saída é 1 ou 0, escrevendo no arquivo "value" dentro de seu diretório ("gpio4"):

```
echo "1" > /sys/class/gpio/gpio4/value
echo "0" > /sys/class/gpio/gpio4/value
```

Já para o pino destinado à entrada (id "7"), para saber seu estado lógico, ou seja se ele está retornando 1 ou 0, deve-se ler o arquivo "value" dentro de seu respectivo diretório (nesse caso "gpio7"):

```
cat /sys/class/gpio/gpio7/value
```

Após a realização de todos os testes, é prudente restaurar as configurações originais dos pinos, através do arquivo "unexport" no diretório "/sys/class/gpio/":

```
echo "4" > /sys/class/gpio/unexport
echo "7" > /sys/class/gpio/unexport
```

Encerrando-se, vemos que os diretórios "gpio4" e "gpio7" não estão mais presentes.

2.4.2 Bibliotecas Python

O Raspberry Pi oferece, como maneira nativa de programação, a linguagem Python[4] que, já pré-instalado no Raspbian, disponibiliza um conjunto de bibliotecas para o uso da plataforma.

A biblioteca RPi oferece elementos necessários para se operar a GPIO e deve ser importada em um código fonte da seguinte forma:

```
import RPi.GPIO as GPIO
```

Note que a dependência RPi.GPIO foi importada como um objeto chamado GPIO para se poder trabalhar todos os seus métodos no código fonte.

Após a importação, deve-se definir o modo de trabalhar com os pinos (BOARD ou BCM), mais detalhes sobre a diferença desses modos são esclarecidos na seção 2.3. No exemplo a seguir, é utilizado o modo "BCM":

```
GPIO.setmode(GPIO.BCM)
```

Para trabalhar com algum pino específico basta utilizar o método GPIO.setup() que deve ser invocado com os atributos, respectivamente, id e direção (saída ou entrada). No exemplo a seguir é utilizado o id "4" para saída e "7" para entrada:

```
GPIO.setup(4, GPIO.OUT)
GPIO.setup(7, GPIO.IN)
```

No caso do pino de saída pode-se alterar seu estado lógico para 1 ou 0 através do método GPIO.output() que deve ser invocado com os atributos id do pino e estado lógico:

```
GPIO.output(4, 1)
GPIO.output(4, 0)
```

No caso do pino configurado como entrada, é possível conhecer seu estado lógico através do método GPIO.input() que tem como atributo o id do pino:

```
input = GPIO.input(7)
```

O método acima retorna o estado lógico do pino de id "7" para a variável "input". No término do código fonte é aconselhável restaurar as configurações da GPIO através do seguinte método:

```
GPIO.cleanup()
```

[4] https://www.python.org/

2.5 Qual método de acesso utilizar?

Algumas linguagens de programação não possuem bibliotecas para o acesso a GPIO, mas permitem acesso à diretórios e leitura e escrita de arquivos. Para pessoas que não querem programar em Python, e sim em sua linguagem de expertise e essa linguagem não possui biblioteca para acesso à GPIO, o método de acesso por diretórios é o mais indicado.

Já para aqueles que programam em Python, o segundo método apresentado é mais interessante, pois abre a possibilidade de utilizar *frameworks* web, como no caso deste trabalho que utiliza o Django (Apêndice A).

Portanto, pode-se observar que a maneira de configurar os pinos é bastante simples, tanto com o uso de diretórios quanto da linguagem de programação Python. Entretanto, isso é somente um nível acima de abstração da GPIO, o que pode ser complicado para alguns usuários da plataforma que almejam uma forma mais alto nível para utilizar os pinos. Sendo assim, uma interface web para ajustar configurações e a possibilidade de prototipar sistemas se faz necessária, o que é abordado no próximo capítulo.

3 Uma ferramenta web para prototipação no Raspberry Pi

Esse capítulo apresenta o processo de análise e concepção de uma ferramenta Web para manipulação da GPIO.

A principal motivação para a realização do projeto é fornecer aos estudantes que não tem contato intenso com técnicas de programação a possibilidade de prototipação de projetos eletrônicos com rapidez.

A vantagem de fazer este sistema em web é a disponibilidade e abrangência, já que a maioria dos dispositivos que acessam a internet possuem navegadores de internet. Assim, independentemente do usuário utilizar um *smartphone* ou um PC com qualquer sistema operacional, a ferramenta está disponível.

3.1 ProtoPi

O sistema em desenvolvimento foi batizado como ProtoPi e teve como logotipo a imagem apresentada na Figura 8.

Figura 8 – Logotipo do Sistema.

Fonte: Autoria Própria

De uma maneira geral, o sistema oferece duas funcionalidades:

1. Acesso Rápido: O usuário acessa cada Pino da GPIO individualmente, através de um mapa e executa funções em tempo real. Esta funcionalidade é para realização de testes rápidos e não é possível salvar o que é feito.

2. Criação de protótipos: Os protótipos são elaborados e salvos. O usuário pode criar seus próprios módulos e executá-los.

3.2 Engenharia de requisitos

Os requisitos de um sistema são apresentados como descrições sobre o que ele deve fazer, ou seja, através de serviços, modos de operações e restrições ofertadas aos usuários. A análise e documentação dessas necessidades é o que define a Engenharia de Requisitos (ER) (SOMMERVILLE, 2011).

Dentro dos requisitos para o desenvolvimento do sistema pode-se classificá-los em: requisitos de sistema e requisitos de usuário. O primeiro trata-se de quais serviços são esperados pelo sistema para os usuários e em que restrições o mesmo deve operar, já o segundo trata das necessidades computacionais para a realização do projeto de desenvolvimento (PRESSMAN, 2011).

Trazendo os conceitos trabalhados acima para o **ProtoPi** pode-se imaginar os requisitos da seguinte forma:

- **Requisitos de Usuário**: o projeto deve ser acessível através de um navegador de internet, deve ter uma ferramenta gráfica para acessar os pinos rapidamente através de um mapa da GPIO e deve ter uma ferramenta gráfica para montar e salvar protótipos eletrônicos.
- **Requisitos de Sistema**: deve funcionar através de um servidor web que rode o *framework* Web Django (informações adicionais no Apêndice A), seu *Design* deve ser através do *framework* bootstrap[1].

3.2.1 Requisitos não funcionais

Dentro dos requisitos de sistemas os não funcionais são aqueles mais gerais que independem da lógica de negócio adotada no desenvolvimento e refletem as qualidades esperadas (SOMMERVILLE, 2011).

É possível definir como requisitos não funcionais para o **ProtoPi** os seguintes tópicos:

- **Segurança**, ou seja, deve ter um controle de acesso para mitigar a possibilidade de acessos simultâneos, pois a execução de protótipos deve ser realizada uma por vez.
- **Sistema gerenciador de banco de dados**, o que significa que o usuário deve ter a possibilidade de salvar seu protótipo para executá-lo e alterá-lo a qualquer momento.

[1] http://getbootstrap.com/

3.2.2 Requisitos funcionais

Os requisitos funcionais descrevem explicitamente as funcionalidades e serviços em termos de lógica e técnicas de programação associadas a lógica de negócio de um sistema(SOMMERVILLE, 2011)

São definidos como requisitos funcionais para o **ProtoPi** os seguintes itens:

- **Configurações da GPIO ProtoPi** deve definir a configuração da GPIO em modo BCM (seção 2.3 do Capítulo 2);
- **Configurações dos pinos pelo sistema**, ou seja, deve definir a direção dos pinos (entrada ou saída);
- **Execução de métodos relacionadas aos pinos**: fornecer para execução de operações básicas com os pinos como alteração e leitura do estado lógico.

3.3 Casos de uso

Para um melhor entendimento do sistema e seus requisitos, sempre é necessário realizar abstrações mais próximas da realidade, pois, às vezes a capacidade humana de criação não atende plenamente a técnica pura e necessita de um **cenário** prático para seu exercício (SOMMERVILLE, 2011).

Para criar um **cenário**, pode-se utilizar um diagrama clássico da UML (*Unified Modeling Language*)[2]: caso de uso.

No caso do **ProtoPi**, é possível elaborar os cenários nomeados como **Acesso Rápido** e **Prototipação**. Estes cenários, elaborados a partir das duas funcionalidades abordadas na seção 3.1, serão descritos na subseção 3.3.2 e subseção 3.3.3.

A próxima seção apresenta os atores para os diagramas de casos de uso.

3.3.1 Atores

Dentro do **ProtoPi** é possível identificar os seguintes atores:

- **Usuário**: define quais os pinos que devem ser acessados, como devem ser utilizados e elabora protótipos;
- **ProtoPi**: fornece a estrutura necessária para o usuário visualizar os pinos da GPIO, acessá-los e realizar protótipos;
- **GPIO**: executa o que é definido no **ProtoPi**.

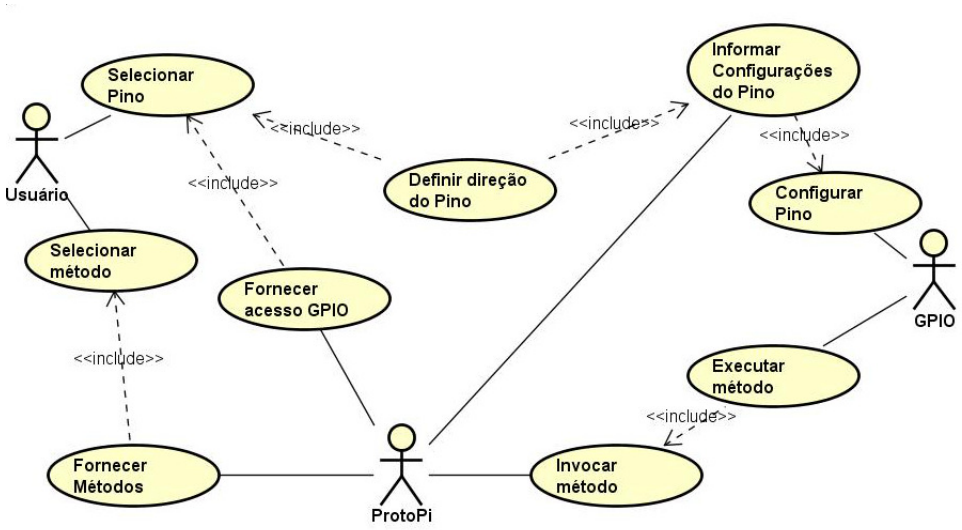
[2] http://www.uml.org/

3.3.2 Acesso Rápido

O caso de uso para funcionalidade **Acesso Rápido** pode ser visto na Figura 9 em que é observável os papéis de cada ator no cenário da realização da funcionalidade "Acesso Rápido" do sistema. É possível verificar que o ator **Usuário** define os pinos que ele deseja acessar com o fornecimento do mapa e acesso do **ProtoPi** e o mesmo requisita a configuração e execuções de método ao ator **GPIO**.

Basicamente, é concluível que **ProtoPi** é um intermediário entre os demais atores servindo de tradutor dos comandos gráficos que o **Usuário** seleciona para uma linguagem de mais baixo nível que o ator **GPIO** "entenda".

Figura 9 – Diagrama de caso de uso Acesso Rápido.



Fonte: Autoria Própria

É possível visualizar que para cada ação do **Usuário**, existe a obrigatoriedade do **ProtoPi** fornecer uma funcionalidade. Por exemplo, ao **selecionar um pino** o **ProtoPi** deve **fornecer acesso à GPIO**, ou seja, através de uma representação gráfica os pinos devem estar disponíveis. Após a seleção do pino, o **Usuário** deve **definir a direção do pino**, o que significa que **ProtoPi** deve **informar as configurações do pino** ao ator **GPIO** que, de fato, configura o pino (**configurar pino**).

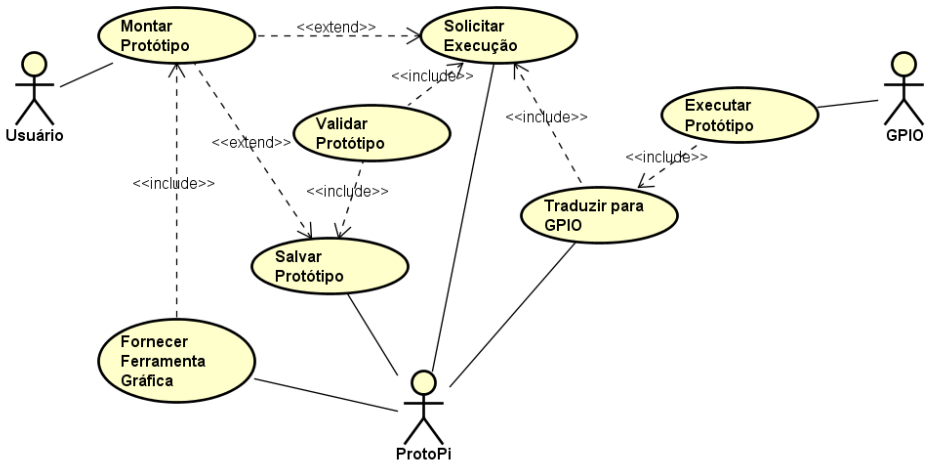
Após o término da configuração, o **Usuário** passa a **selecionar método** o que implica que **ProtoPi** deve **fornecer métodos** para esta seleção. Concluído, **ProtoPi** deve **invocar o método** e a GPIO executá-lo (**executar método**).

3.3.3 Prototipação

Já o caso de uso para **Prototipação** pode ser visto na Figura 10, em que há a presença da ferramenta gráfica oferecida pelo ator **ProtoPi** ao **Usuário**. O **Usuário** deve, ao invés de programar através das tradicionais linhas de códigos, montar o conjunto de instruções de seu protótipo através da ferramenta gráfica. **ProtoPi** mantém seu papel de tradutor e, após validação, pode armazenar em forma de conjunto de instruções codificadas e solicitar a execução pelo ator **GPIO**.

O caso de uso **Prototipação** ressalta ainda mais o papel do **ProtoPi** como intermediário/tradutor do usuário ao **GPIO**.

Figura 10 – Diagrama de caso de uso Prototipação.



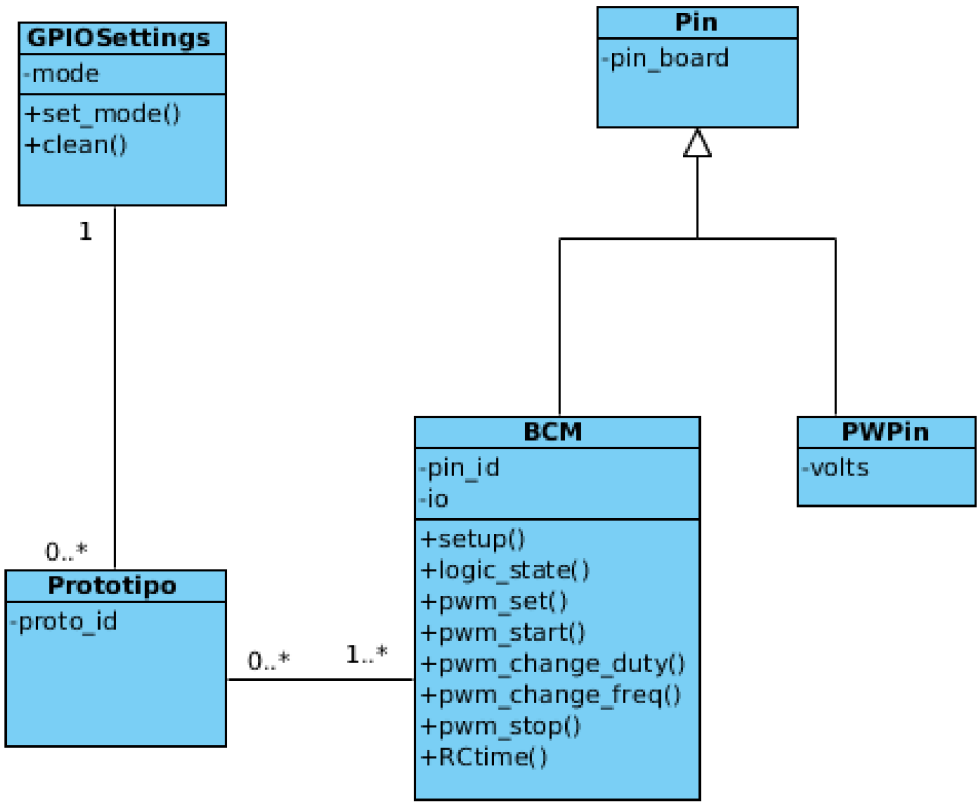
Fonte: Autoria Própria

Ao **Usuário** realizar a montagem do protótipo (**montar protótipo**), há a obrigação do **ProtoPi** em **fornecer uma ferramenta gráfica** para realização do projeto. Concluído o protótipo, o **Usuário** pode **salvar o protótipo** e **solicitar execução**, etapas que devem passar por processo de validação (**validar protótipo**). Com a devida validação, **ProtoPi** deve **traduzir para GPIO** as estruturas lógicas, em forma de código fonte Python e por fim o ator **GPIO** realiza a execução (**executar protótipo**)

3.4 Diagrama de Classes

Para melhor entender a estrutura funcional do projeto ,a Figura 11 apresenta as principais classes do **ProtoPi**.

Figura 11 – Diagrama de classes ProtoPi.



Fonte: Autoria Própria

É possível verificar que a estrutura do sistema é composta por cinco classes, sendo a classe **Pin** abstrata em que **BCM** e **PWPin** herdam seu atributo "pin_board" que representa a posição física do pino na GPIO.

A classe **PWPin** representa os pinos destinados a alimentação sendo o atributo "volts" a tensão que o pino fornece, em que 0 representa quando o pino é *Ground*. Esta classe não é usada na estrutura lógica dos protótipos mas é importante para a renderização das ferramentas gráficas das duas funcionalidades do sistema.

Já a classe **BCM** representa os pinos de entrada e saída em que o atributo "pin_id" é seu id BCM (detalhado no Capítulo 2 na seção 2.3) e "io" a direção do pino, ou seja, entrada e saída. A seguir são detalhados os métodos presentes nesta classe.

- **setup()**: este método configura a direção do pino através do atributo "io" (entrada ou saída).
- **logic_state()**: caso o pino seja de saída, este método define seu estado lógico, já se o pino é definido como de entrada, o método retorna seu estado lógico.
- **pwm_set()**: configura a função de PWM (*Pulse-width modulation*) destinados apenas aos pinos com sua direção definida como "saída" para controlar a variação do estado lógico do pino durante o tempo definida pela frequência e razão cíclica.
- **pwm_start()**: inicia a função de PWM.
- **pwm_change_duty()**: altera a razão cíclica da função PWM.
- **pwm_change_freq()**: altera a frequência da função PWM.
- **pwm_stop()**: encerra a função PWM.
- **RCtime()**: método destinado aos pinos de entrada que demonstra o tempo que o pino leva para atingir o estado lógico 1.

A classe **GPIOSettings** é destinada a fornecer ao protótipo as configurações da GPIO e a inicialização da mesma. O atributo "mode" representa o modo a ser trabalhado ("BOARD" ou "BCM"), e o método "set_mode" inicializa a GPIO e o "clean" restaura a GPIO ao seu estado inicial para a execução de outros protótipos.

Por fim, a classe **Prototipo** é destinada a todas as estruturas lógicas de cada projeto criado pelos usuários do **ProtoPi**. A classe **Prototipo** necessita obrigatoriamente de um objeto proveniente da classe **GPIOSettings** e pode conter vários objetos provenientes da classe **BCM**.

3.5 Detalhes de implementação

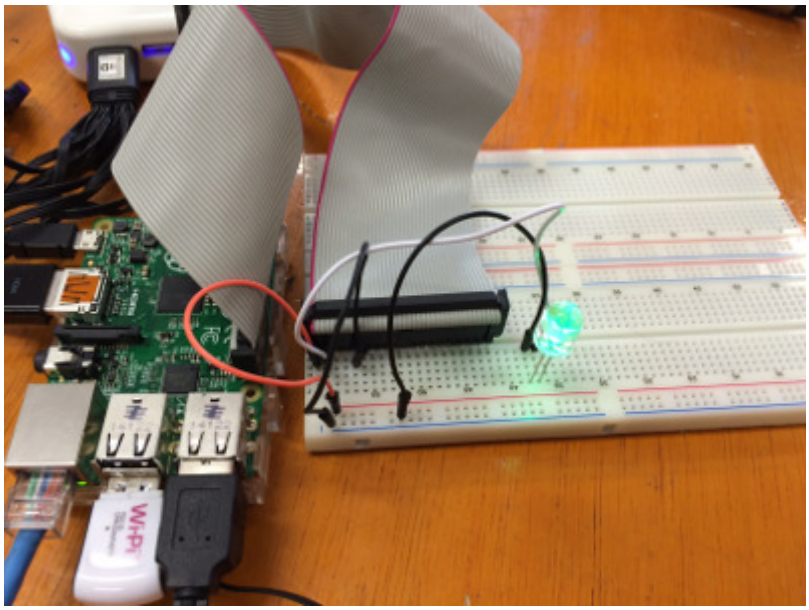
No desenvolvimento do **ProtoPi** foi utilizado para *back-end* para operação dos pinos da GPIO a biblioteca Python RPi, detalhada no Capítulo 2 na seção 2.4. O Django (Apêndice A), versão 1.8, foi utilizado como *Framework* web por ser em Python e, portanto, conseguir ter a biblioteca RPi integrada ao desenvolvimento.

Já para o *front-end*, ou seja, o parte cliente do sistema, foi utilizado o bootstrap (TWITTER, 2015) para elaboração do layout e o jQuery (JQUERY, 2015) para efeitos visuais e requisições AJAX (*Asynchronous Javascript and XML*) (W3C, 2015) por parte do cliente, para assim, a experiência de uso da ferramenta ficasse mais agradável evitando excessivas trocas e recargas de páginas.

4 Exemplo de uso do ProtoPi

Para uma melhor elucidação do ProtoPi, este Capítulo apresenta um exemplo prático de uso da funcionalidade de **Acesso Rápido**. Para que o esta funcionalidade possa ser utilizada, é necessário que exista um protótipo eletrônico real. Para fins didáticos, um projeto eletrônico bem simples é utilizado como exemplo neste capítulo que permite o controle da intensidade do brilho de um LED(*Light Emitting Diode*). O protótipo eletrônico pode ser visto na Figura 12.

Figura 12 – Protótipo eletrônico para um LED.



Fonte: autoria própria

Pode-se observar na Figura 12 os seguintes componentes:

- Uma conexão entre a GPIO e a protoboard através de um cabo *flat* (cabo geralmente utilizado para conexão de um HD IDE e uma placa mãe de um PC). Para este projeto, o cabo foi modificado para permitir a interconexão entre a GPIO do Raspberry Pi e a protoboard;
- Uma protoboard, item necessário para evitar o uso de soldas;
- LED de 3.3 V na cor verde;
- Cabos jumpers para interconexão dos componentes na protoboard.

Tradicionalmente, é possível, se tratando de pinos digitais, trabalhar apenas com dois níveis de intensidade do brilho: aceso e apagado, ou seja, há uma limitação no controle do brilho de LEDs através dos pinos digitais. Isso se dá pelos pinos da GPIO somente fornecerem dois tipos de voltagem 0 ou 3.3 volts, o que seriam os estados lógicos digitais 1 ou 0 (verdadeiro ou falso).

Uma maneira de se ter o efeito de variação do brilho do LED é utilizando o método PWM que, através da frequência determinada em Hertz[1] e a razão cíclica[2], oferece a ilusão de redução ou aumento da intensidade do LED.

No caso do circuito apresentado, na Figura 12, verifica-se que o pino id BCM "'3" está conectado ao anodo (terminal positivo) do LED através do cabo *jumper* da cor branca e seu catodo (terminal negativo) está conectado a um pino terra (*Ground*) através do cabo na cor preta, ou seja, toda vez que o pino id BCM "3" tem seu estado lógico definido como verdadeiro, o LED se acende. Nesse exemplo, é possível alterar o brilho do LED definindo uma frequência alta, imperceptível ao olho humano, e variando a razão cíclica de 0 a 100 (em porcentagem) cria-se o efeito de diminuição e aumento da intensidade do brilho do LED. Quanto mais próximo de 100 maior o brilho.

A maneira tradicional, em python, para fazer o que foi descrito se apresenta no código fonte a seguir. Esse *script* é o mais simples possível para realizar a função pretendida sem a possibilidade do usuário interferir na razão cíclica, somente diminuindo o brilho do LED e após dois segundos aumentando. Para se ter maior interação do usuário para modificar a razão cíclica e frequência seria necessário incluir mais linhas de códigos, o que tornaria o algoritmo mais complexo.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(2, GPIO.OUT)

p = GPIO.PWM(2,50)

p.start(100)
time.sleep(2)

p.ChangeDutyCycle(20)
time.sleep(2)
```

[1] O número de vezes que o estado lógico do pino sofrerá variações ao longo de 1 segundo.
[2] Porcentagem de tempo em que o pino permanece no estado lógico 1 durante cada segundo.

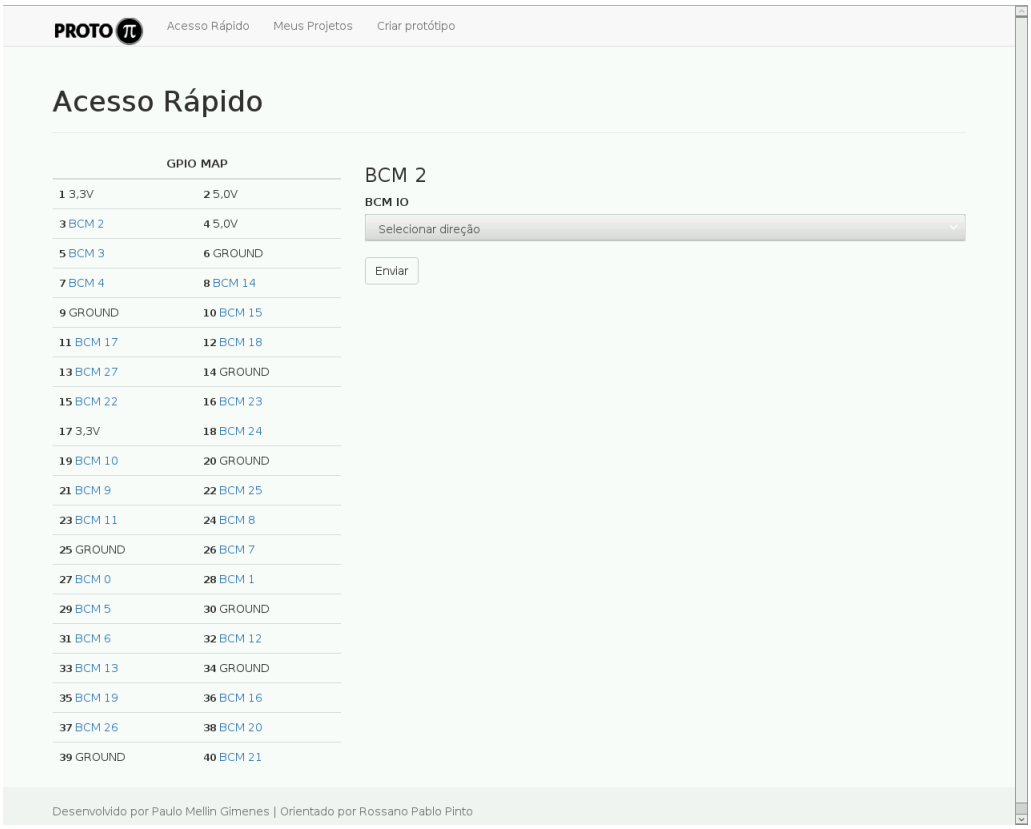
```
p.ChangeDutyCycle(90)
time.sleep(2)

p.stop()

GPIO.cleanup()
```

Por isso, a próxima seção demonstra como criar este efeito utilizando a funcionalidade **Acesso Rápido** do ProtoPi para aumentar e diminuir o brilho do LED apresentado na Figura 12 em que o usuário pode modificar os atributos em tempo real e criar sem ter que programar o efeito de variação de brilho.

4.1 Alterando brilho do LED com ProtoPi

Na funcionalidade de acesso rápido o usuário tem disponível um mapa em que pode selecionar o pino com o qual deseja trabalhar, como demonstrado na Figura 13.

Figura 13 – Exemplo alteração brilho de LED - Tela inicial.



Fonte: autoria própria

Através do mapa representado no lado esquerdo da Figura 13, há a possibilidade do usuário selecionar qualquer pino de entrada e saída, definir sua direção (“IN” ou “OUT”) no menu BCM IO, o que acarretará na exibição dos métodos relacionadas com a direção escolhida. A seguir na Figura 14 é demonstrada estas seleções em que o pino de BCM 2 é selecionado e sua direção definida como “OUT”.

Figura 14 – Exemplo alteração brilho de LED - Seleção de direção.

Fonte: autoria própria

Selecionada a direção como “OUT”, para manipular a intensidade de brilho deve-se selecionar o método PWM como demonstrado na Figura 15.

Figura 15 – Exemplo alteração brilho de LED - Seleção do método PWM.

Fonte: autoria própria

Feita a seleção do método PWM, deve-se selecionar uma ação, dividida entre “Iniciar”, “Alterar freqûencia”, “Alterar razão cíclica” e “Parar”. A seguir, na Figura 16 é selecionada a ação “iniciar”.

Figura 16 – Exemplo alteração brilho de LED - Seleção da ação iniciar.

Fonte: autoria própria

É possível observar que na Figura 16 a frequência inicial foi determinada em 50 Hz com uma razão cíclica de 100 %, o que significa que o LED estará com seu brilho máximo.

Para diminuir o brilho do LED, basta alterar a razão cíclica na ação “alterar razão cíclica” o que é demonstrado na Figura 17.

Figura 17 – Exemplo alteração brilho de LED - Alteração da razão cíclica.

Fonte: autoria própria

Com a razão cíclica menor, há um efeito de diminuição do brilho do LED. Com o *ProtoPi* é possível o usuário interagir com o recurso de PWM sem ter a obrigação de

codificar e com interação em tempo real.

Como observado, o uso da ferramenta ProtoPi possibilitou a elaboração do mesmo programa apresentado no início do capítulo sem a necessidade de codificar uma única linha de código.

A seguir as Considerações Finais são apresentadas.

Considerações finais

Com o desenvolvimento do **ProtoPi**, com sua proposta, visão geral, requisitos e análise detalhados no Capítulo 3, é possível notar o quão importante são as ferramentas para que o público em geral consiga realizar projetos eletrônicos com uma ferramenta mais próxima da linguagem humana e mais afastada da tradicional codificação.

Codificação, na maneira tradicional, pode parecer um "bicho de sete cabeças" e às vezes pode desmotivar o público em adquirir ferramentas para prototipação de projetos eletrônicos. Por isso, um sistema que humanize a relação entre pessoa e a plataforma de maneira mais próxima ao linguajar humano e mais distante das linguagens de baixo nível se faz necessária.

O uso de plataformas web para o desenvolvimento do **ProtoPi** oferece o benefício de independência dos dispositivos e sistemas operacionais, ou seja, a ferramenta pode ser utilizada em qualquer dispositivo que ofereça um navegador de internet.

Com o uso do ProtoPi é possível desenvolver sistemas que fazem uso da GPIO do Raspberry Pi sem a necessidade de se codificar. Para isso, **ProtoPi** oferece dois modos de uso: **Acesso Rápido** e **Prototipação**.

Para a evolução do ProtoPi, as seguintes sugestões de trabalhos futuros são elencadas:

- O uso em laboratório de informática para diversos alunos, ou seja, a possibilidade de prototipação concorrente. Hoje a ferramenta possui apenas possibilidade de uso individual;
- A utilização de vários dispositivos Raspberry Pi em forma de *clusters*. A solução apresentada contempla apenas um RPi, a sugestão é o uso da ferramenta em vários dispositivos conectados em uma mesma rede, o que permitiria construir sistemas na linha da Internet das Coisas (IoT).

Referências

ARDUINO. *Arduino documentation*. 2015. Disponível em: <https://www.arduino.cc/en/Guide/Introduction>. Acesso em: 06 nov. 2015. Citado 3 vezes nas páginas 11, 14 e 15.

ATZORI, L.; IERA, A.; MORABITO, G. "The Internet of Things: A survey". *ScienceDirect - Computer Networks*, v. 54, n. 15, p. 2787–2805, June 2010. Citado na página 21.

BEAGLEBOARD.ORG FOUNDATION. *What is BeagleBone?* 2015. Disponível em: <http://beagleboard.org/bone>. Acesso em: 06 nov. 2015. Citado 2 vezes nas páginas 11 e 17.

CARRO, L.; WAGNER, F. R. Sistemas computacionais embarcados. *Jornadas de atualização em informática. Campinas: UNICAMP*, 2003. Citado na página 11.

DJANGO SOFTWARE FOUNDATION. *Django documentation*. 2015. Disponível em: <https://docs.djangoproject.com/en/1.8/>. Acesso em: 29 mai. 2015. Citado 2 vezes nas páginas 43 e 45.

DJANGO SOFTWARE FOUNDATION. *Meet Django*. 2015. Disponível em: <https://www.djangoproject.com/>. Acesso em: 29 mai. 2015. Citado 2 vezes nas páginas 18 e 43.

LITTLEBITS ELECTRONICS, INC. *What is Littlebits?* 2015. Disponível em: <http://littlebits.cc/how-it-works>. Acesso em: 06 nov. 2015. Citado na página 15.

PRESSMAN, R. S. *Engenharia de software*. [S.l.]: McGraw Hill Brasil, 2011. Citado na página 27.

RAJSUMAN, R. *System-on-a-chip: Design and Test*. [S.l.]: Artech House, Inc., 2000. Citado na página 17.

RASPBERRY PI FOUNDATION. *Raspberry Pi FAQs*. 2015. Disponível em: <https://www.raspberrypi.org/help/faqs/>. Acesso em: 29 mai. 2015. Citado na página 47.

RASPBERRY PI FOUNDATION. *Raspberry Pi GPIO Pinout*. 2015. Disponível em: <http://pi.gadgetoid.com/pinout>. Acesso em: 29 mai. 2015. Citado 3 vezes nas páginas 22, 48 e 49.

RASPBERRY PI FOUNDATION. *What Is a Raspberry Pi?* 2015. Disponível em: <https://www.raspberrypi.org/help/what-is-a-raspberry-pi/>. Acesso em: 29 mai. 2015. Citado 3 vezes nas páginas 11, 19 e 21.

SOMMERVILLE, I. *Software Engineering*. 9. ed. Harlow, England: Addison-Wesley, 2011. Citado 2 vezes nas páginas 27 e 28.

TANENBAUM, A. S. *Organização estruturada de computadores*. 5. ed. São Paulo: Pearson Prentice Hall, 2007. Citado 2 vezes nas páginas 13 e 14.

THE JQUERY FOUNDATION. *jQuery API Documentation*. 2015. Disponível em: <http://api.jquery.com/>. Acesso em: 06 nov. 2015. Citado 2 vezes nas páginas 18 e 32.

TWITTER. *Bootstrap - Getting Started*. 2015. Disponível em: <http://getbootstrap.com/getting-started/>. Acesso em: 06 nov. 2015. Citado na página 32.

WORLD WIDE WEB CONSORTIUM. *AJAX Tutorial*. 2015. Disponível em: <http://www.w3schools.com/ajax/>. Acesso em: 06 nov. 2015. Citado 2 vezes nas páginas 18 e 32.

Apêndices

APÊNDICE A – Django

A.1 Definição

O *framework* web Django utiliza a plataforma de desenvolvimento Python[1]. Segundo a *Django Software Foundation* em sua página principal de seu *website*[2], a melhor definição para esta ferramenta é:

> *Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.(DSF, 2015b)*

A.2 Conferindo a versão e instalando

Para instalar o Django é necessário ter a plataforma tecnológica Python instalada no sistema operacional. Este trabalho recomenda a utilização da distribuição GNU/Linux Debian em sua versão 8.0 (codinome Jessie) pois o sistema operacional utilizado para o projeto final é baseado nesta distribuição (Raspbian), por padrão Debian já tem Python em seu pacote de instalação.

Para conferir se o Django está em instalado em sua versão mais recente deve-se utilizar o seguinte comando no terminal do Sistema Operacional:

```
python -c "import django; print(django.get_version())"
```

Se o Django estiver instalado será informado sua versão, senão a seguinte mensagem de erro relatando que não foi encontrado o módulo com o nome Django será apresentada[3](DSF, 2015a).

É importante ressaltar que as próximas etapas desta seção deve ser realizada com permissão de administrador do sistema (root):

A.2.1 Instalação

Caso o Django não estiver instalado ou desatualizado, deve-se utilizar o gerenciador de pacote pip[4]. Em distribuições Debian, se utiliza-se o Gerenciador de Pacote Apt para instalar este programa:

---

1 https://www.python.org/
2 https://www.djangoproject.com/
3 Até a última versão deste trabalho a versão mais recente foi constada como a 1.8
4 https://pip.pypa.io/en/stable/

```
apt-get install python-pip
```

Após instalar o pip é possível instalar ou atualizar a versão mais recente do Django através do comando:

```
pip install Django
```

A.3 Elaborando um projeto e aplicação Django

Nesta seção será demonstrado como criar um projeto e uma aplicação Django.

A.3.1 Projeto

Para criar um projeto Django, deve-se estar no diretório de preferência e digitar o seguinte comando:

```
django-admin startproject meuprojeto
```

Após a execução do comando é possível verificar a seguinte estrutura de diretório com os seguintes arquivos:

```
meuprojeto/
 manage.py
  meuprojeto/
   __init__.py
   settings.py
   urls.py
```

1. **manage.py**: script para intarface *command-line* para interagir com o projeto django.
2. **___init___.py**: Um arquivo vazio para indicar que o diretório em que está listado deve ser considerado com um pacote Python.
3. **settings.py**: Arquivo com os principais atributos para a configuração do projeto.
4. **urls.py**: arquivo destinado a mapear as URLs do projeto e atribuir métodos a cada URL.

A.3.2 Diferença entre Projeto e Aplicação

Antes de abordar sobre a criação de uma aplicação dentro do projeto criado é importante definir a diferença entre Projeto e Aplicação.

Uma aplicação (app) em django pode ser vista como algo específico como um WebBlog, um banco de gravações ou até mesmo um simples sistema de enquetes. Já o

projeto contém uma coleção de configurações e apps, ou seja, um projeto pode conter muitos apps e um app pode estar em múltiplos projetos(DSF, 2015a).

A.3.3 Criando uma Aplicação

Para criar uma aplicação em seu projeto basta utilizar o arquivo manage.py estando no diretório do projeto criado ("meuprojeto") da seguinte forma:

```
python manage.py startapp minhaapp
```

Após a execução, será criado um diretório "minhaapp" com a seguinte estrutura:

```
minhaapp/
    __init__.py
    admin.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

O Django trabalha com três camadas: *Model*, *View* e *Template*(DSF, 2015a). O arquivo **models.py** contém todas as classes relativas a aplicação representando a camada Model do Django. A ideia principal é modelar as classes em um único fonte seguindo, assim, o princípio DRY (*Don't repeat yourself*)(DSF, 2015a).

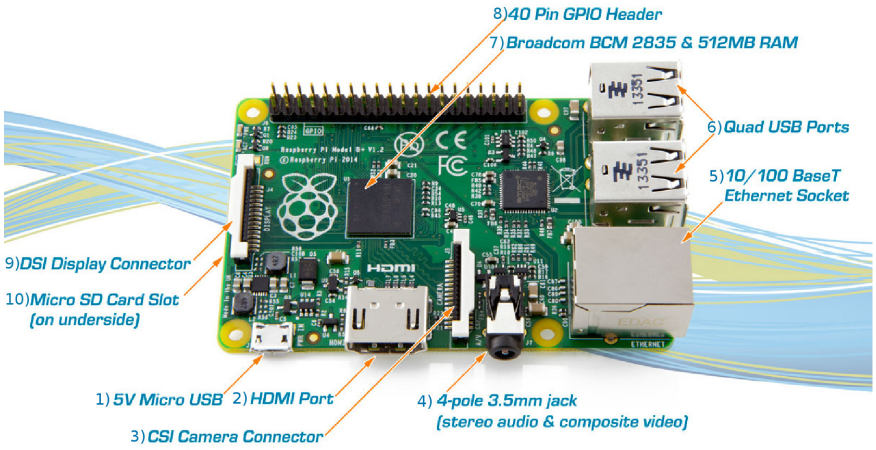
Já o arquivo **views.py** contém toda a lógica para processar requisições e retornar respostas, representando a camada *View*(DSF, 2015a).

Já a camada *Template* contém todos os arquivos referente ao *design* do projeto, ou seja, os arquivos html, css e também javascript que, por padrão, devem estar em um diretório nomeado como "template" a ser criado no diretório "minhaapp/" (DSF, 2015a).

APÊNDICE B – Componentes do Raspberry Pi

Os componentes do Raspberry Pi estão indicados na Figura 18 em que é referenciado com um número o nome de cada componente:

Figura 18 – Componentes do Raspberry Pi(Modelo B+).



Fonte:http://www.cnet.com/uk/products/raspberry-pi-model-b-plus/ - Editado pelo autor com o uso do Software Gimp 2.8

A descrição de cada componente apresentado na Figura 18 está presente na Tabela 2 em que a coluna “Número” remete ao número indicado anteriormente e “Componente” ao componente também representado na imagem seguido de sua descrição.

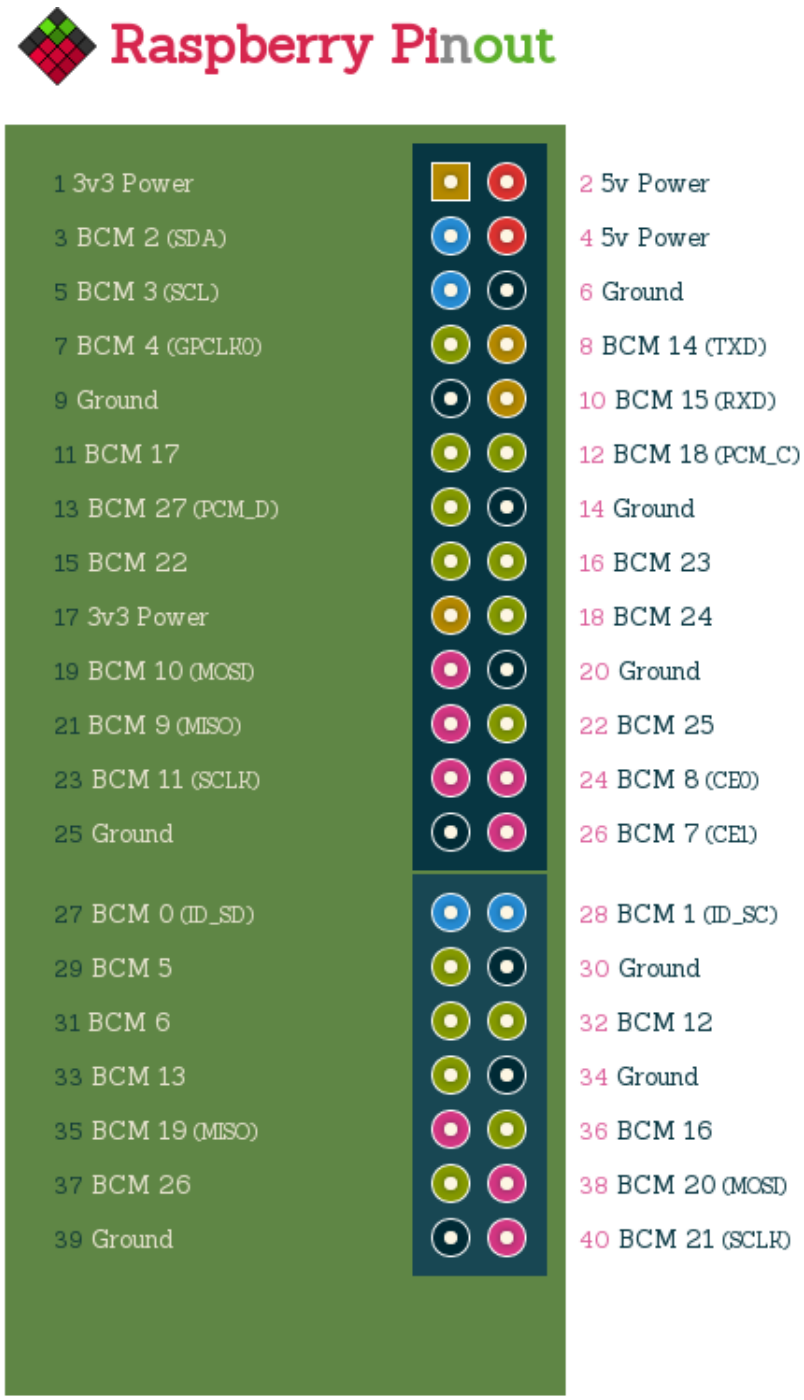
Tabela 2 – Componentes do Raspberry Pi.

Número	**Componente**	**Descrição**
1	5V Micro USB	Porta de alimentação Micro USB
2	HDMI Port	Saída de vídeo HDMI
3	CSI Camera Connector	Entrada para Câmera padrão CSI
4	4-pole 3.5 mm Jack	Entrada padrão P2 para áudio e vídeo
5	10/100 BaseT Ethernet Socket	Entrada padrão Ethernet para rede
6	Quad USB ports	4 Entradas padrão USB
7	Broadcom BCM 2835 & 512 RAM	Processador e quantidade de RAM do dispositivo
8	40 Pin GPIO Header	40 pinos de uso geral para entrada e saída
9	DSI Display Connector	Interface para conexão de *display*
10	Micro SD Card Slot	*Slot* para cartão de memória.

Fonte: Rpi Foundation (2015a)

APÊNDICE C – Detalhes sobre a GPIO

Figura 19 – Mapa da GPIO.



Fonte: Rpi Foundation (2015b)

Na Tabela 3 a seguir é possível localizar mais precisamente cada pino que tem uso

especial além do de entrada e saída.

Tabela 3 – GPIO: Pinos para Uso Especial.

Código BCM	**Posição**	**Descrição**	**Função**
BCM 0	27	ID_SD	i2c com HAT EEPROM (*Data*)
BCM 1	28	ID_SC	i2c com HAT EEPROM (*Clock*)
BCM 2	3	SDA	i2c(*Data*)
BCM 3	5	SCL	i2c(*Clock*)
BCM 4	7	GPCLK0	Pino para(*Clock*)
BCM 7	26	SC1	SPI *Chip Select 1*
BCM 8	24	SC0	SPI *Chip Select 0*
BCM 9	21	MISO	SPI MISO
BCM 10	19	MOSI	SPI MOSI
BCM 11	23	SCLK	SPI SCLK
BCM 14	8	TXD	UART *Transmit*
BCM 15	10	RXD	UART *Receive*
BCM 18	12	PCM_C	PCM (*Clock*)
BCM 19	35	MISO	SPI *Master In*
BCM 20	38	MOSI	SPI *Master Out*
BCM 21	40	SCLK	SPI (*Clock*)
BCM 27	13	PCM_D	PCM (*Data*)

Fonte: Rpi Foundation (2015b)

1. **i2c**: desenvolvido pela Phillips no início da década de 1980, é um barramento serial para comunicar periférico. Pode-se conectar através desta interface conversores analógicos digitais, dispositivos de entrada e saída, microcontroladoras, EEPROM e sensores.Duas conexões, SDA e SDC, feitas por fio são necessárias para estabelecer comunicação. [1].

2. **EEPROM**: *Electrically Erasable Programmable Read-Only Memory* é uma memória reservada para leitura que pode ser apagada ou reprogramada através da eletricidade [2].

3. **UART**: *Universal asynchronous receiver/transmitter* é a interface recomendada para conexão de uma microcontroladora Arduino ao RPi. Esta interface é exclusiva para conexão de dispositivos seriais[3].

4. **SPI**: *Serial Peripheral Interface* é uma interface para conectar o RPi a periféricos a uma curta distância. Assim como i2c, pode ser utilizada para microcontroladoras. Utiliza 3 linhas para comunicação: MISO, MOSI e SCLK [4].

[1] **Fonte:**http://i2c.info/. Acesso às 19h15 em 22/04/2015

[2] **Fonte:** http://whatis.techtarget.com/definition/EEPROM-electrically-erasable-programmable-read-only-memory. Acesso às 19h30 em 22/04/2015

[3] **Fonte:** http://whatis.techtarget.com/definition/UART-Universal-Asynchronous-Receiver-Transmitter. Acesso às 19h40 em 22/04/2015

[4] **Fonte:** http://www.arduino.cc/en/Reference/SPI. Acesso às 19h40 em 22/04/2015