Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

LUAN ARAÚJO COELHO SILVEIRA CASTRO

PROJETO DE CAMPAINHA INTELIGENTE CONTROLADA POR SISTEMA DE VÍDEO

Campina Grande, Paraíba
Agosto de 2018

LUAN ARAÚJO COELHO SILVEIRA CASTRO

PROJETO DE CAMPAINHA INTELIGENTE CONTROLADA POR SISTEMA DE VÍDEO

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Eletrônica, controle e automação.

Orientadora:
Professora Doutora Luciana Ribeiro Veloso.

Campina Grande, Paraíba
Agosto de 2017

LUAN ARAÚJO COELHO SILVEIRA CASTRO

PROJETO DE CAMPAINHA INTELIGENTE CONTROLADA POR SISTEMA DE VÍDEO

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Eletrônica, controle e automação.

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professora Doutora Luciana Ribeiro Veloso
Universidade Federal de Campina Grande
Orientadora, UFCG

Dedico este trabalho aos meus pais Djannildo e Vilma, ao meu irmão Ruan e a minha noiva Linsey, que são o meu suporte diário e a base da minha felicidade.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais Djannildo e Vilma, que sempre me acompanharam e me deram forças em todas as etapas da minha vida, lutando dia após dia em busca da perfeita harmonia dentro e fora de casa.

À UFCG, por me permitir crescer intelectualmente, oferecendo um curso de excelência em minha área, construído por um corpo docente extremamente qualificado, essencial para minha formação profissional, e por uma equipe de funcionários administrativos que facilitaram o percurso que me permitiu chegar até aqui.

A professora Dra. Luciana Ribeiro Veloso, por ter sido uma grande tutora durante o curso, não apenas como simples professora, mas por ter sido minha orientadora enquanto fui monitor de uma de suas disciplinas e por ter aceitado ser minha orientadora neste trabalho de conclusão de curso bem como por acreditar sempre no meu potencial e por continuamente me entusiasmar com a profissão.

Aos meus amigos que me acompanharam durante toda esta trajetória, Mauricio, Thiago, Rodolfo e Luiz, não apenas como verdadeiros amigos, bem como companheiros de mesmo apartamento.

A todos que já foram meus professores, cada um foi de fundamental importância para minha trajetória profissional.

Por fim, agradeço àqueles colegas de curso, que me acompanharam e me ajudaram durante toda minha trajetória de graduação, sem eles o percurso teria sido muito mais árduo.

"—A realidade está errada.
Sonhos podem sim se tornar realidade"
Autor desconhecido.

RESUMO

Este trabalho de conclusão de curso apresenta a implementação de uma campainha inteligente de baixo custo, implementada junto à campainha das residências. Uma campainha inteligente é um dispositivo compacto, capaz de monitorar o fluxo de pessoas junto à entrada das residências bem como o controle da abertura da porta de entrada a qual está associada. Trata-se de um dispositivo largamente utilizado, porém no Brasil, este tipo de aparelho ainda é caro. Portanto, uma campainha inteligente de baixo custo e que apresente resultados precisos, pode expandir ainda mais a sua usabilidade. Este projeto envolve o desenvolvimento de um circuito analógico de aquisição e condicionamento dos sinais, controlados por um microcontrolador, que é também responsável por enviar os sinais amostrados para um aplicativo *Android* via Internet. Através do aplicativo é possível o recipiente conceder a abertura da porta de entrada associada a este sistema.

Palavras-chave: *wireless*, campainha, comunicação, módulos, Arduíno.

ABSTRACT

The current document introduces the implementation of a low-cost Smart Doorbell, which is installed alongside the doorbell outside the resident's front door. A smart doorbell is a compact device that can monitor the flow of movement from any individuals near the entrance of house, as well as, control the opening of the front door. It is a widely used device around the world, however in Brazil, this type of device is often too expensive for the average household. Therefore, an innovative and low-cost doorbell which presents accurate results can further expand its usability. This project involves the development of an analog acquisition and signal-conditioning device, controlled by a microcontroller, which is also responsible for sending the sampled signals to an Android application via the Internet. Through the application, it is possible for the user to grant the door's opening which is associated with this system.

Keywords: Wireless, doorbell, communication, modules, Arduino.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BLE	*Bluetooth Low Energy*
Bps	*Byte per second*
bps	*Bits per second*
BR	*Basic Rate*
BT	*Bluetooth*
CI	Circuito Integrado
CPU	*Central Process Unit*
CRC	*Cyclic redundancy check*
DC	*Direct current* (corrente contínua)
DQPSK	*Differential quadrature phase shift keying*
DTR	*Data terminal ready*
E/S	Entrada/ Saída
EDR	*Enhanced Data Rate*
FM	*Frequency Modulation*
FTDI	*Future Technology Devices International*
GFSK	*Gaussian frequency shift keying*
GND	*Ground*
GPIO	*General Purpose Input/output*
I/O	*Input/ Output*
ICSP	*In Circuit Serial Programming*
IDE	*Integrated Development Environment*
IEM	Interferência eletromagnética
ISM	*Industrial*, *Scientific*, *Medical*
MCU	Microcontrolador
NA	Normalmente aberto
NF	Normalmente fechado
PROM	*Programmable Read Only Memory*
PSK	*Phase-shift keying*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
RF	Rádio Frequência
RFI	Interferência de rádio frequência

RX	Receptor
SO	Sistema Operacional
SPI	*Serial Peripherical Interface*
SSR	*Solid State Relay*
TTL	*Trasnsistor – transistor logic*
TX	Transmissor
UART	*Universal Asynchronous Receiver/Transmitter*

LISTA DE SÍMBOLOS

μ	Micro (10^{-6})
A	Ampére
B	Byte
dBm	Decibel miliwatt
G	Giga (10^{9})
Hz	Hertz
K	Kilo(10^{3})
m	Mili(10^{-3})
M	Mega(10^{6})
n	Nano(10^{-9})
V	Volt
W	Watts
Ω	Ohm

SUMÁRIO

1 INTRODUÇÃO

A violência no Brasil é um problema persistente que atinge direta ou indiretamente a população. O país tem níveis acima da média mundial no que se refere a crimes violentos, com níveis particularmente altos no tocante a violência armada e homicídios. No primeiro semestre de 2017, diversas Secretarias de Segurança Pública no Brasil registraram um aumento significativo de roubos e furtos em residências. Por exemplo, em Campina Grande (PB), o aumento nos índices de furtos residenciais foi de 97% no último ano e, em João Pessoa (PB), 50% no mesmo período. Dados levantados pelo Centro Integrado de Operações Policiais (Ciop) da Polícia Militar apontam que, em média, uma casa é invadida por dia em Campina Grande.

Adicionalmente, é possível destacar que a maioria das campainhas residenciais encontradas no comércio brasileiro não permitem que uma pessoa com deficiência auditiva tenha a percepção que a campainha foi acionada. De acordo com estimativa do *Instituto Brasileiro de Geografia e Estatística - IBGE*, cerca de 9,7 milhões de brasileiros possuem deficiência auditiva, o que representa 5,1% da população do país. No que se refere a idade, quase 1 milhão são crianças e jovens até 19 anos. A acessibilidade para surdos ainda é um desafio. Essa parcela da população ainda enfrenta dificuldades para conseguir realizar atividades cotidianas, a exemplo identificar o acionamento da campainha de suas residências. Portanto, uma vez que o sistema desenvolvido agrega uma notificação no aparelho celular do morador da residência em questão, tem-se que o público com deficiência auditiva também será beneficiado uma vez da aquisição do sistema.

Visando ampliar ainda mais a usabilidade do projeto desenvolvido podemos destacar uma das informações obtidas pelo *IBGE* de que mais de 13,2 milhões de pessoas no Brasil afirmam ter algum grau de deficiência motora, o que equivale a 7% dos brasileiros. A deficiência motora severa foi declarada por mais de 4,4 milhões de pessoas. Destas, mais de 734,4 mil disseram não conseguir caminhar ou subir escadas de modo algum e mais de 3,6 milhões informaram ter grande dificuldade de locomoção. Portanto, visando promover o máximo de independência

no âmbito das capacidades e limitações das pessoas que se encontram nesta faixa de incapacidade física, o sistema desenvolvido torna-se uma ferramenta capaz de promover maior mobilidade as pessoas que se encontram neste espectro, uma vez que o sistema promove a abertura de uma porta à distância, não sendo necessário nenhum tipo de locomoção, por parte do indivíduo portador de tal insuficiência motora.

Dentro deste contexto, esta proposta de trabalho de conclusão de curso apresenta a implementação de uma campainha inteligente de baixo custo, implementada junto à campainha das residências.

Inicialmente são introduzidos os conceitos de microcontroladores e suas diversas aplicabilidades, bem como formas de operação. Em seguida, são apresentados módulos capazes de trazer maior funcionalidade aos microcontroladores, tornando possível a leitura de dados e comunicação à distância e sem fios entre as plataformas trabalhadas. Posteriormente são apresentados os resultados dos testes realizados com diferentes arquiteturas de hardware em busca do protótipo que apresentasse resultados mais satisfatórios para os objetivos desejados. Por fim, apresenta-se os resultados dos testes bem como o protótipo desenvolvido.

2 MICROCONROLADORES

Microcontroladores são utilizados em muitos processos e aplicações tais como: sistemas de controle automotivos, sistemas de segurança, máquinas residenciais, brinquedos, sistemas de supervisão, entre outros. A grande aplicabilidade dos microcontroladores deve-se as seguintes características: possuir tamanho reduzido e ter seu custo e consumo de energia cada vez menores, aliados as facilidades de implementação.

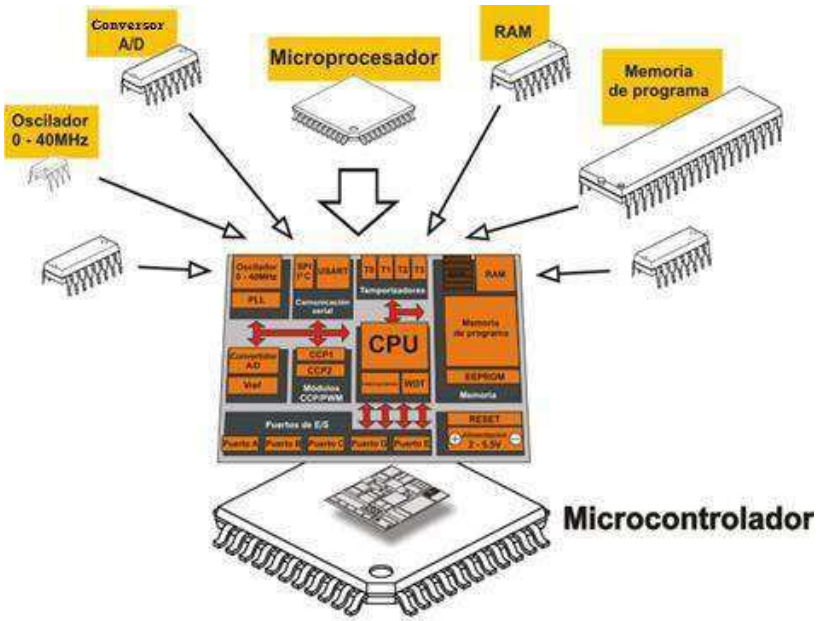
Um microcontrolador é um Circuito Integrado (CI) capaz de efetuar processos lógicos. A grande vantagem deste CI é a sua possibilidade de programação, o que o torna adaptável à finalidade desejada, (Sousa D. J., 2000).

O desenvolvimento da tecnologia de circuitos integrados, tornou possível o armazenamento de milhares de transistores em um único chip, suscitou a produção de microprocessadores. A crescente necessidade de busca pela redução dos tamanhos físicos de componentes e equipamentos com o passar do tempo, desencadeou a integração entre periféricos e processador em um único circuito integrado, que passou a ser designado de microcontrolador (MCU) (Assis, 2004).

O microcontrolador possui, basicamente, os seguintes dispositivos:

- CPU (*Central Processing Unit*), cuja finalidade é interpretar as instruções de programa;
- Memória PROM (*Programmable Read Only Memory*), na qual são gravadas as instruções do programa;
- Memória RAM (*Random Access Memory*), utilizada para armazenar as variáveis utilizadas pelo programa;
- Conjunto de entradas e saídas utilizadas para controlar dispositivos externos ou receber sinais dos sensores, interruptores, entre outros;
- Conjuntos de dispositivos auxiliares ao funcionamento. Dentre eles pode-se citar: gerador de *clock*, contadores, unidades para comunicação, conversores entre outros.

Figura 1 – Composição de um microcontrolador.



Fonte: (TOMIC[1],2017).

Geralmente, os microcontroladores possuem um sistema operacional (SO) simples quando comparado com os SO de computadores desktop convencionais. Os programas rodam diretamente no chip. O *software* que roda no microcontrolador é denominado *Firmware*. Esse *software* é programado em linguagens C ou Assembly na maior parte dos casos, embora seja possível usar outras linguagens em alguns MCUs. A programação é feita com o uso de uma ferramenta instalada em um computador, chamada de IDE (do inglês *Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado).

No mercado são diversas as empresas que fabricam microcontroladores, sendo a Mitsubishi, Atmel, Freescale, Texas Instruments, Intel e Microchip Technology as mais conhecidas. Esses chips variam de acordo com a aplicação, a velocidade de operação, a capacidade de memória, o tipo de memória, tamanho da palavra, entre outros. Alguns exemplos típicos de microcontroladores comuns encontrados no mercado e usados comumente em projetos eletrônicos estão listados a seguir:

- ARM Cortex-M;
- Atmel AVR / AVR 32;

[1] Mikroe. (s.d.). *MikroElektronika - Development tools* Acesso em 14 de Junho de 2018, disponível em www.edgefx.in: https://www.edgefx.in/difference-between-microprocessor-and-microcontroller

- Intel 8051;
- Microchip PIC;
- NXP LCP 2000 / 3000;
- Paralax Propeller;
- Texas Instruments MSP430.

Muito comum no mercado são os kits e placas para estudo e prototipagem de sistemas embarcados e programação de microcontroladores. Como exemplo desses kits tem-se a Plataforma Arduíno, o Microchip PIC Starter Kit, o Texas Instruments MSP430 Launchpad e o Keil MCB900 (NXP). A utilização desses kits facilita a realização dos testes associados aos microcontroladores no desenvolvimento dos equipamentos.

2.1 PLACA DE DESENVOLVIMENTO ARDUÍNO

Arduíno é uma plataforma para prototipagem eletrônica que utiliza o conceito de *hardware* livre. Neste conceito, deve-se prover a disponibilização irrestrita das informações sobre o projeto de *hardware*, tais como diagramas eletrônicos, estrutura de produtos, *layout* de placas de circuito impresso, rotinas de baixo nível e qualquer outra informação necessária para uma construção de um projeto (Silva & Sales, 2016).

A ideia inicial da criação da placa (que foi desenvolvida no *Ivrea Interaction Design Institute*) foi de um instrumento para prototipagem simples de sistemas para aqueles que não possuíam conhecimentos profundos das áreas de eletrônica e programação. A medida que foi se popularizando para uma comunidade mais especializada, a placa foi adquirindo adaptações que a tornou favorável a aplicações mais sofisticadas, como para IoT, impressão 3D, sistemas embarcados, automações residenciais e *wearables*.

Ao longo dos anos, o Arduíno tem sido o núcleo de milhares de projetos, desde objetos cotidianos até complexos instrumentos científicos. Uma comunidade mundial de criadores - estudantes, amadores, artistas, programadores e profissionais - se reuniram em torno desta plataforma de

hardware e *software* aberto e suas contribuições somaram uma incrível quantidade de conhecimento acessível[2].

O processo de utilização do microcontrolador é bastante simplificado utilizando o Arduíno. Além dessa facilidade, pode-se citar as seguintes vantagens associada ao uso do Arduino:

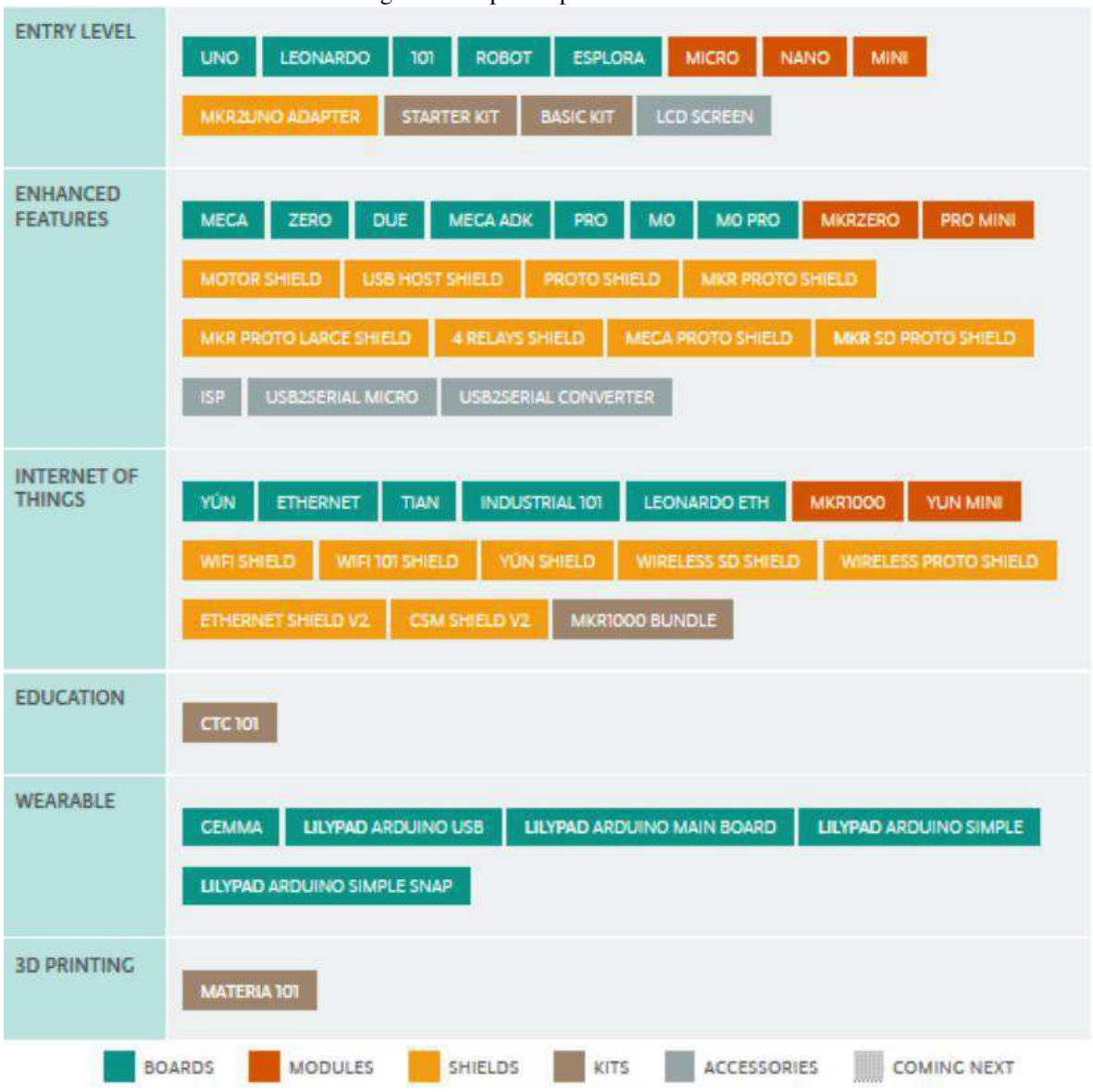
- Baixo custo: as placas são relativamente baratas em comparação com outras plataformas de microcontrolador;
- *Cross-platform* (Multiplataforma) - O *software* Arduíno (IDE) é executado em sistemas operacionais Windows, Macintosh OSX e Linux. A maioria dos sistemas de microcontroladores é limitada ao Windows;
- Ambiente de programação simples e claro - O *software* Arduíno (IDE) é fácil de usar, mas flexível o suficiente para que os usuários avançados também possam aproveitar;
- *Software open source* e extensível - O *software* Arduíno é publicado como ferramentas de código aberto, disponíveis para a extensão por programadores experientes. O idioma pode ser expandido por meio de bibliotecas C ++, e as pessoas que querem entender os detalhes técnicos podem fazer o salto do Arduíno para a linguagem de programação AVR C em que se baseia. Da mesma forma, se desejar, pode adicionar códigos AVR-C diretamente em seus programas;
- *Hardware open source* e extensível - Os planos das placas Arduíno são publicados sob uma licença *Creative Commons*, para que designers de circuitos experientes possam fazer sua própria versão do módulo, estendê-la e melhorá-la. Mesmo os usuários relativamente inexperientes podem construir a versão de painéis do módulo, a fim de entender como ele funciona e economizar dinheiro.

[2] Arduíno. (2017). *Arduíno - What is Arduino?* Acesso em 15 de Junho de 2018, disponível em www.arduino.cc: https://www.arduino.cc/en/Guide/Introduction

Existe uma gama de placas Arduíno no mercado, cada uma com diferentes especificações (o microcontrolador, quantidade de portas e conexões, memória, entre outros). A escolha dependerá sempre da aplicação a qual a placa será destinada. A Figura 2 apresenta um resumo dos existentes atualmente.

Nos tópicos 2.2 e 3.3 são apresentadas as características do Arduíno UNO e da ESP32, que foram os utilizados neste projeto.

Figura 2 – Tipos de placa de Arduino.



Fonte: (Arduíno. 2017)

2.2 ARDINO UNO

O UNO possui o processador ATMEGA 328 e 14 portas digitais, sendo que seis delas podem ser usadas como saídas PWM, e 6 portas analógicas. A alimentação (selecionada automaticamente) pode vir da conexão USB ou do conector para alimentação externa (recomendável 7 a 12 Vdc). Possui uma tensão de operação de 5 V, cristal oscilador de 16 MHz, corrente máxima nas portas de entrada e saída (E/S) de 40 mA, uma conexão ICSP (*In Circuit Serial Programming*, protocolo de comunicação utilizado para gravação de dispositivos programáveis) e um botão de reset. Em relação à memória, o mesmo possui as seguintes características (todos do Atmega 328):

- Memória flash de 32 KB dos quais 0,5 KB são utilizadas pelo bootloader;
- SRAM de 2 KB;
- EEPROM de 1 KB.

O Arduíno possui um *firmware* chamado *bootloader* que é executado ao alimentar a placa ou quando pressionado o botão de *reset*. Este tem a função de carregar um programa do computador escrito na IDE, e escrever na memória do microcontrolador. Este *firmware* é responsável pela facilidade que o Arduíno oferece de ser programado diretamente pela porta serial (que na realidade ocorre pela USB, o que acontece é que existe outro microcontrolador nas placas do Arduíno UNO, conversor serial-USB, que cria uma porta serial virtual no computador quando o Arduíno é conectado).

Cada um dos 14 pinos digitais do Arduino UNO podem ser utilizados como uma entrada ou uma saída utilizando-se as funções pinMode(), digitalWrite(), e digitalRead(). Eles possuem um resistor *pull-up* interno (desconectado por padrão) de 20-50 kΩ. Além disso, alguns pinos têm funções especializadas:

- Serial: 0 (RX) e 1 (TX). Usados para receber (RX) e transmitir (TX) dados seriais TTL. Estes pinos são conectados aos pinos correspondentes do chip, conversor, serial USB-para-TTL ATmega8U2;

- Interruptores Externos: pinos 2 e 3. Estes pinos podem ser configurados para disparar uma interrupção de acordo com alguma variação sensível pelo circuito (função attachInterrupt());
- SPI: pino 10 (SS), pino 11 (MOSI), pino 12 (MISO),pino 13 (SCK). Estes pinos dão suporte à comunicação SPI utilizando a biblioteca SPI;
- LED: pino 13. Há um LED integrado ao pino digital 13;
- I2C: pino 4 (SDA) e pino 5 (SCL). Fornecem suporte à comunicação I2C (TWI) utilizando a biblioteca Wire;
- AREF: Tensão de referência para as entradas analógicas. Utilizado com a função analogReference ();
- Reset: Envio o valor LOW para esta linha para resetar o microcontrolador. Tipicamente usado para adicionar um botão de reset para *shileds* montados sobre a placa original.

As entradas analógicas, A0 à A5, tem 10 bits de resolução cada (variam então com valores de 0 a 1023). Por padrão elas medem de 0 a 5 V, embora seja possível alterar o limite superior utilizando o pino AREF e a função analogReference (). O anexo A apresenta o mapa de pinos de toda a placa.

3 COMUNICAÇÃO POR RÁDIO FREQUÊNCIA

Uma conexão *wireless* é qualquer forma de conexão entre dois sistemas transmissor e receptor de dados em que não requeira o uso de fios.

Em um sistema de comunicação sem fio que utilize ondas de rádio, a informação a ser transmitida é modulada em uma portadora, ou seja, ela é posicionada no espectro de frequências de modo que no mesmo meio físico possa trafegar informações de vários transmissores, desde que estejam utilizando uma faixa não ocupada. Por meio da modulação é possível fazer o deslocamento do espectro da informação para uma posição específica do espectro. O fato de não existirem fios ligando os dispositivos de comunicação permite que esses dispositivos ofereçam mobilidade. Para explorar esta vantagem, torna-se vital a utilização de transmissores e receptores de baixo consumo, para que estes possam ser alimentados com baterias pequenas (Sousa M. B., 2002).

Os sinais de radiofrequência são sinais que se propagam por um condutor e são irradiados no ar por meio de uma antena. Na prática, uma antena converte um sinal cabeado em um sinal *wireless* e vice-versa. Esses sinais são então irradiados no ar livre na forma de ondas de rádio e se propagam em linha reta e em todas as direções. Podem-se imaginar essas ondas como círculos concêntricos que aumentam o seu raio na medida em que se afastam da antena (Sampaio, 2008).

Para implementar um sistema de transmissão de dados por ondas de radiofrequência, necessita-se principalmente de módulos de radiofrequência, que são responsáveis pela transmissão e recepção dos dados via ondas de rádio. Nos tópicos subsequentes a este são retratados os módulos que, associados a microcontroladores (no caso, o Arduíno), cumprem a aplicação de enviar dados sem fio para o microcontrolador. Estes módulos que serão apresentados atuam como meios de comunicação capazes de pôr em prática diversas ações, como funções de telemetria (efetivar medidas de sensores à distância), acionamento de cargas por meio do acionamento direto de relés ou por acoplamento óptico, envio

e recepção de dados analógicos e/ou digitais para as mais variadas aplicações, entre outras.

3.1 MÓDULOS *BLUETOOTH*

Os tópicos subsequentes trazem uma explanação referente a módulos que utilizam a tecnologia Bluetooth como ferramenta de seu processamento.

3.1.1 TECNOLOGIA *BLUETOOTH*

A tecnologia Bluetooth é de curto alcance, onde dentre seus objetivos principais destacam-se sua capacidade de eliminar os cabos nas conexões entre dispositivos eletrônicos, tanto portáteis como fixos, bem como permitir troca de dados e arquivos de computadores, scanners, fones de ouvido e demais dispositivos de forma rápida e segura. Dentre as principais características desta tecnologia, elenca-se: confiabilidade, baixo consumo e mínimo custo. Várias das funções das especificações são opcionais, o que permite a diversificação dos produtos.

Os dispositivos *Bluetooth* operam na faixa ISM (*Industrial*, *Scientific*, *Medical*) centrada em 2,45 GHz. Inicialmente sua operação era formalmente reservada para alguns grupos de usuários profissionais em sua camada Rádio Frequência (RF). Nos Estados Unidos, a faixa ISM varia de 2400 a 2483,5 MHz. Na maioria da Europa a mesma banda também está disponível. No Japão a faixa varia de 2400 a 2500 MHz.

A faixa ISM é aberta, e assim, pode ser utilizada por qualquer sistema de comunicação. Portanto, é imprescindível garantir que o sinal do Bluetooth não sofra interferência e também não a gere. O esquema de comunicação FH-CDMA (*Frequency Hopping — Code-Division Multiple Access*), utilizado pelo Bluetooth, permite essa proteção, dividindo a frequência em vários canais.

O dispositivo que estabelece a conexão muda de um canal para outro de maneira bastante rápida. Esse “salto de frequência" permite que a largura de

banda da frequência seja muito pequena, diminuindo sensivelmente as chances de interferência. No *Bluetooth*, pode-se utilizar até 79 frequências (ou 23, dependendo do país) dentro da faixa ISM, cada uma "espaçada" da outra por intervalos de 1 MHz[3].

Os dispositivos são classificados, de acordo com a potência e alcance, em três níveis: classe 1 (100 mW, com alcance de até 100 m), classe 2 (2,5 mW e alcance até 10 m) e classe 3 (1 mW e alcance de 1 m, uma variante muito rara).

Os dispositivos *Bluetooth* comunicam-se entre si e formam uma rede denominada piconet (rede formada pela conexão entre dispositivos Bluetooth), na qual podem existir até oito dispositivos interligados, sendo um deles o mestre (*master*) e os outros dispositivos escravos (*slave*); uma rede formada por diversos "*masters*" (com um número máximo de 10) pode ser obtida para maximizar o número de conexões. Esse procedimento consiste em fazer uma piconet se comunicar com outra que está dentro do limite de alcance, esquema este denominado *scatternet*. Desta maneira, podemos definir estas características de operação da seguinte maneira:

- Modo *Master* (Mestre): O módulo pode se conectar a outros dispositivos *Bluetooth*;
- Modo *Slave* (Escravo): O módulo apenas recebe conexões de outros dispositivos *Bluetooth*;
- Modo *Loopback*: O módulo recebe os dados do módulo *Master* e envia de volta esses mesmos dados. É um modo utilizado geralmente para testes.

Existem dois modos de modulação: *Basic Rate* (BR), que é obrigatória e usa uma modulação binária FM para minimizar a complexidade de transmissão e a *Enhanced Data Rate* (EDR) que é um modo opcional e usa a modulação PKS (*PhaseShift Keying*) e tem duas variantes: π/4-DQPSK e 8DPSK. A taxa bruta de transmissão é 1 Mbps para a “Categoria Básica”, 2 Mbps para “Categoria de

[3] Info wester. (30 de Janeiro de 2008). *Info wester - Tecnologia Bluetooth: o que é e como funciona?* Acesso em 20 de Julho de 2018, disponível em https://www.infowester.com/bluetooth.php

Dados Melhorados" usando π/4-DQPSK e 3 Mbps para "Categoria de Dados Melhorados" usando 8DPSK (Okuda, 2008).

3.1.2 MÓDULOS HC-05 E HC-06

Os módulos mais comuns presentes do mercado que realizam a comunicação *Bluetooth* são o HC-05 e o HC-06, que trabalham com a tecnologia BT 3.0. A diferença básica entre esses módulos é que o HC-05 pode ser configurado nos modos *Master* (mestre), *Slave* (escravo) e *Loopback*, e o HC-06 apenas no modo *Slave*.

Tanto o HC-05 quanto o HC-06 podem ser configurados por meio de comandos AT, que se tratam da forma básica de configurar e acionar o Arduino quando ele está sob controle de um equipamento externo, a partir de programas de controle serial. O HC-05 possui mais comandos, como pode ser visualizado na Tabela 1.

Tabela 1 - Comandos AT dos módulos HC-05 e HC-06.

Comando	**Função**	**HC-05**	**HC-06**
AT	Teste	X	X
AT+RESET	*Reset*	X	
AT+VERSION	Mostra a versão do *software*	X	X
AT+ORGL	Restaura configurações padrão	X	
AT+ADDR?	Mostra o endereço do módulo BT (Bluetooth)	X	
AT+NAME	Mostra/ altera o nome do módulo BT	X	X
AT+RNAME?	Mostra o nome do módulo BT remoto	X	
AT+ROLE	Seleciona modo *master/ slave/ loopback*	X	
AT+PSWD	Altera a senha do módulo	X	X
AT+UART	Altera a velocidade (*baud rate*)	X	X
AT+RMAAD	Remove a lista dos dispositivos pareados	X	
AT+INQ	Inicia a varredura por dispositivos BT	X	
AT+PAIR	Efetua o pareamento com BT remoto	X	
AT+LINK	Efetua a conexão com o BT remoto	X	

Fonte:(KOYANAGI,2017)

Mais recentemente, apareceram os HC-08 e HC-10 que trabalham com tecnologia BT 4.0 ou BLE ("*Bluetooth Low Energy*"). Os módulos BLE são os

únicos que podem ser conectados a um *Iphone*, pois a *Apple* não fornece suporte ao BT 3.0.

O módulo HC-06 é um pouco mais simples de configurar que o HC-05. Resumidamente, basta conectá-lo a, por exemplo, um Arduíno, carregar o programa e enviar os comandos AT por meio da serial.

No caso do HC-05, para enviar esses comandos tem-se que utilizar um pino adicional, o pino KEY (como pode ser visualizado na Figura 3), que deve estar em estado ALTO (*HIGH*). Esse procedimento coloca o módulo em "modo AT". Alguns módulos vêm até mesmo com um pequeno botão para colocar esse pino em nível *HIGH*.

Figura 3 – Modulos *Bluetooth*.

Fonte:(KOYANAGI,2017).

A conexão dos pinos dos módulos é extremamente simples, os dois tipos possuem os pinos Vcc, GND, Rx e Tx. É importante ressaltar que o nível de sinal utilizado pela maioria dos módulos para a comunicação serial é de 3.3 V (não confundir com a alimentação do módulo, que pode ser de 3.6 a 6 V). Assim, para usar o módulo com as placas Arduíno, por exemplo, que trabalham com nível de sinal de 5 V, deve-se usar sempre um divisor de tensão para o pino RX do módulo para evitar danos ao mesmo.

Vale salientar que ao se usar a entrada serial do UNO para o uso do módulo, é muito importante lembrar que o mesmo não pode estar fisicamente conectado aos pinos 0 e 1 (Tx e Rx) do Arduíno durante a carga do programa. Isso porque o USB também usa essa mesma serial. Uma maneira simples de contornar esse problema (se o projeto não utiliza muitos GPIOs do UNO) é usar uma porta serial por SW por meio da *library SoftwareSerial*.

Para o corrente trabalho, visando uma maior diversificação do produto final em busca de melhores adaptações no mercado, é apresentada uma configuração da campainha inteligente na qual se utiliza o módulo Bluetooth HC-05 no tópico *Desenvolvimento*, de modo que, dependendo do tipo de configuração que o usuário escolher, esta implicará em economia do capital financeiro bem como uma maior versatilidade na utilização do produto.

Consta no *apêndice A* o código referente à configuração e modo de teste do módulo HC – 05, que foi gerado utilizando a plataforma Arduíno Uno em conjunto com a utilização da plataforma *MIT App Inventor,* que será apresentada no tópico 5 *Aplicativo Android*.

3.2 TECNOLOGIA WIFI

WiFi trata-se do nome popular atribuído ao padrão de rede 802.11 Ethernet. Esta tecnologia envolve a transmissão de dados por meio de ondas de rádio fazendo uso do protocolo Ethernet por meio de uma conexão com a Internet sem fio à um computador hospedeiro (host). Na maioria das vezes, a conexão com a Internet via *wifi* é mais rápida quando comparada com conexões por satélite, *Domain-specific language* (DSL) ou cabo. Essencialmente, esta tecnologia trata-se de uma ligação sem fios entre um computador e à Internet (por exemplo, router DSL ou modem por cabo) em uma rede de conexão local.

A forma como esta tecnologia funciona pode ser simplificada da seguinte maneira: inicialmente, um adaptador sem fio associado a um computador converte dados em um sinal de rádio e os transmite usando uma antena. Em seguida, um roteador recebe este sinal e o decodifica, e, por conseguinte, o roteador envia as informações para a Internet usando uma conexão Ethernet física com fio.

Uma rede de conexão *wifi* trabalha dentro de uma faixa de frequências que varia entre 2,4Ghz ou 5Ghz, onde sua faixa de operação está relacionada com a quantidade de dados que está sendo enviada pelo usuário. As transmissões de rádio têm várias restrições que tornam este tipo de comunicação sensível de ser implementada. Desta maneira, a camada física do padrão 802.11 define,

inicialmente, várias técnicas de transmissão que permitem limitar os problemas de interferência, tais como a técnica de escalonamento de espectro com salto de frequência, a técnica de escalonamento de espectro de sequência direta e a tecnologia infravermelha.

Em consequência, assim definida a forma de operação da rede de conexão *wifi*, é apresentada, no tópico *Desenvolvimento*, testes condizentes com o objetivo deste trabalho, que foram desenvolvidos tendo como base esta categoria de transmissão de dados.

3.3 MÓDULO ESP32

No tocante à criação de protótipos de engenharia, a redução no número de componentes necessários para a construção dos dispositivos propostos, é sempre bem-vinda. Neste sentido, surgiu-se a possibilidade da utilização da placa de desenvolvimento ESP32, o qual está descrita mais adiante.

O módulo ESP32 é um módulo de alta performance para aplicações envolvendo *wifi*, contando com um baixíssimo consumo de energia. É uma evolução do já conhecido ESP8266, porém com maior poder de processamento e bluetooth BLE 4.2 embutido.

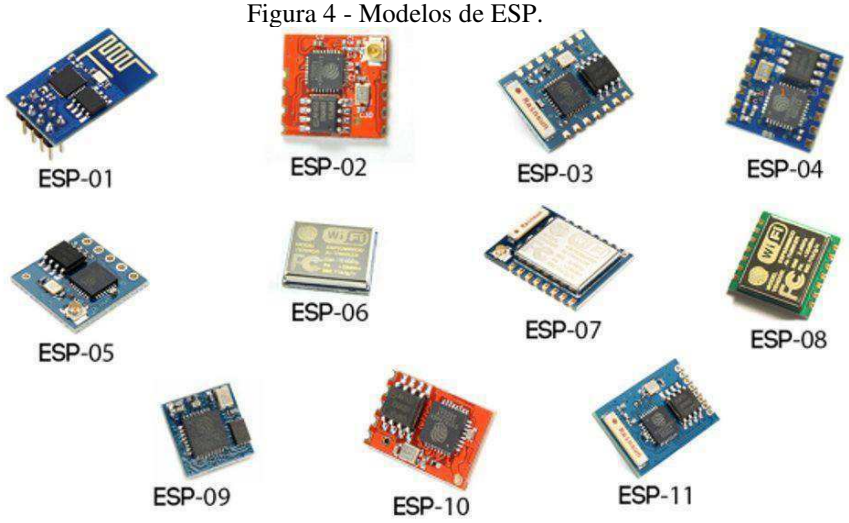
Na placa tem-se o chip ESP32 com antena embutida, uma interface usb-serial e regulador de tensão 3.3V. A programação pode ser feita utilizando a IDE do Arduino através de um cabo micro-usb. Com 4 MB de memória flash, o ESP32 permite criar variadas aplicações para projetos de IoT, acesso remoto, webservers e dataloggers, entre outros. Algumas de suas características estão listadas a seguir:

- CPU: Xtensa® Dual-Core 32-bit LX6
- ROM: 448 KBytes
- RAM: 520 Kbytes
- Flash: 4 MB
- Conexão *wifi* 2.4Ghz (máximo de 150 Mbps)
- Antena embutida

- Wi-Fi Direct (P2P), P2P Discovery, P2P Group Owner mode e P2P
- Bluetooth BLE 4.2
- GPIO com funções de PWM, I2C, SPI
- Suporte para upgrade remoto de firmware

Os módulos ESP são fornecidos numa ampla variedade de modelos, com diferenças perceptíveis principalmente no que tange à quantidade de entradas e saídas (IOs) disponíveis para acesso externo e no tamanho do módulo. A Figura 4 a seguir apresenta alguns desses modelos.

Figura 4 - Modelos de ESP.



Fonte:(KOYANAGI,2017).

Assim como o Arduino, o ESP32 faz uso do mesmo software para sua programação, o Arduino IDE, possuindo inclusive compatibilidades com a maioria de suas bibliotecas (Santos, Pereira, Pereira, & Neto, 2016).

Alguns dos módulos vêm pré-carregados com um *firmware* que os transformam em "Pontes Serial-WiFi". Para realizar essa ponte, a interface serial dos módulos obedece a uma tabela de comandos. Os comandos seriais seguem o padrão AT, e variam de *firmware* para *firmware*, diferenciando principalmente no tipo de resposta dada pelo módulo a cada comando, e na quantidade de comandos suportados. Vale lembrar que a tensão de operação dos pinos do módulo é de 3,3 V. A placa de desenvolvimento utilizada para os testes aqui realizados foi a NodeCMU-32S e seu mapa de pinos está localizado no anexo B.

4 PROCESSAMENTO DIGITAL DE IMAGENS

A primeira etapa de um sistema de processamento de imagens e vídeo é a etapa de aquisição dos dados, onde uma imagem deve ser capturada por uma câmera e convertida em uma entidade gerenciável. Este processo é conhecido como aquisição de imagem. O processo de aquisição de imagens consiste em três etapas: energia refletida do objeto de interesse, um sistema óptico que concentra a energia e, finalmente, um sensor que mede a quantidade de energia.

Imagens digitais são essencialmente matrizes de números, cujos elementos são chamados pixels. Estes representam a intensidade de luz em pontos da imagem. Em uma imagem binária o branco é dado pelo 1 e o preto é dado pelo valor 0.

Dentro dos objetivos deste trabalho, encontra-se a aquisição de imagens por meio de uma câmera à ser instalada nas proximidades da porta a ser controlada pelo protótipo aqui desenvolvido. Em busca da melhor compatibilidade com a plataforma de gerenciamento de dados escolhida, optou-se por utilizar o módulo câmera VGA 0V7670 compatível com Arduino, o qual apresenta-se uma breve introdução a seguir.

4.1 MÓDULO CÂMERA VGA OV7670

O módulo câmera VGA OV7670 é um módulo que permite a captura e armazenamento de imagens coloridas pelo Arduino, com uma taxa de atualização de até 30 frames por segundo, com resolução máxima de 640 x 480 Pixels.

Este pequeno sensor é capaz de prover em um único chip uma câmera VGA e funções de processamento de imagens, ajudando por exemplo em aplicações com Arduino. Através de um barramento de controle, SCCB, é possível visualizar um frame inteiro e sub-sampling com uma resolução de 8 bits de dados.

É possível ainda controlar a qualidade, formato e transmissão da imagem bem como o processamento de imagens como curvas gamma, balanço, saturação e cor através da interface SCCB de programação.

A seguir apresenta-se suas principais características bem como sua pinagem:

- Resolução máxima : 640 x 480
- Taxa de transferência : Máximo de 30 fps em modo VGA
- Tensão de operação : 2.5v à 3.0v – Alta sensibilidade em ambientes com pouca luz
- Baixa tensão de operação, ideal para projetos compactos
- Interface SCCB (Serial Camera Control Bus), compativel com interface I2C
- Suporta VGA, CIF – Formatos de saída : YUV/YCBCr 4:2:2, RGB 565/555, GRB 4:2:2, RAW RGB Data
- Controle automático de funções de imagem – Controle automático de qualidade de imagem, como saturação, matiz, gama, nitidez e anti-blooming
- Auto detecção de flicker (50/60 Hz) – Tamanho da lente : 1/6 ” – Angulo de visão : 24 graus

Figura 5 – Módulo câmera 0V7670 para Arduino.

Fonte: Próprio autor.

A seguir apresenta-se duas tabelas contendo as informações necessárias ao projeto, a primeira contendo a pinagem do módulo e a posterior contendo o significado de cada pino, assim como apresentado no *datasheet* do módulo em questão. É importante destacar que as cores apresentadas nas tabelas foram

geradas apenas para melhor associação do leitor com cada um dos pinos e sua devida descrição.

Tabela 2 – Esquema de pinagem referente ao módulo VGA OV7670.

VDD	· ·	GND
SDIOC	· ·	SDIOD
VSYNC	· ·	HREF
PCLK	· ·	XCLK
D7	· ·	D6
D5	· ·	D4
D3	· ·	D2
D1	· ·	D0
RESET	· ·	PWDN

Fonte: Próprio autor.

Tabela 3 – Tabela contendo descrição de cada pino e suas funções do módulo OV7670.

Pino	Tipo	Descrição
VDD	Fonte	Fonte de alimentação
GND	Fonte	Pino de aterramento
SDIOC	Entrada	SCCB clock
SDIOD	Entrada/Saída	Dados do SCCB
VSYNC	Saída	Sincronização vertical
HREF	Saída	Sincronização horizontal
PCLK	Saída	Clock de pixels
XCLK	Entrada	Clock do sistema
D0-D7	Saída	Saída de vídeo paralela
RESET	Entrada	Reset (Nível lógico ativo baixo)
PWDN	Entrada	Desligamento (Nível lógico ativo alto)

Fonte: Próprio autor.

É importante destacar que este módulo trabalha dentro da faixa de operação do protocolo I²S (Inter-IC Sound). O padrão I²S trata-se de um interface de comunicação de barramento serial desenvolvida pela *Phillips Semiconductor* que é utilizada para conexão de dispositivos de áudio. Este padrão é utilizado para comunicação de dados em formato *Pulse-code Modulation* (*PCM)* entre circuitos integrados em um dispositivo eletrônico. O padrão I²S separa o clock do sistema e os sinais em formato serial, resultando em baixa perda de dados, consistindo basicamente de: dados, clock e linha de seleção.

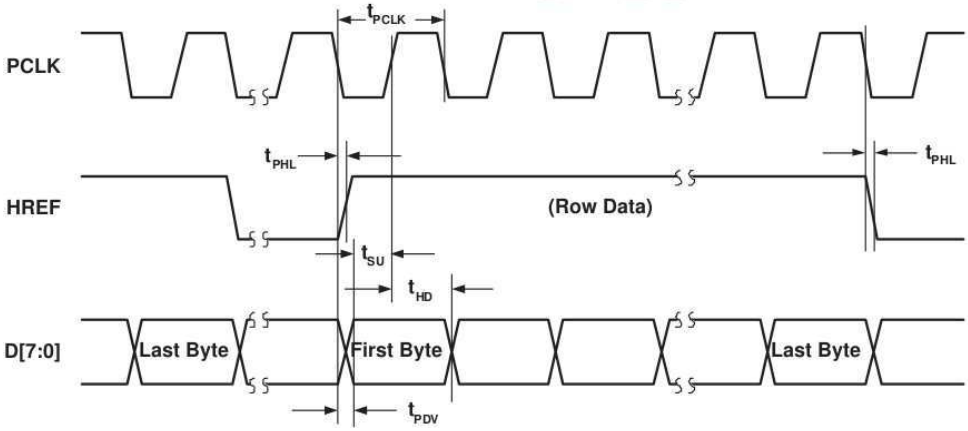
A interface de barramento da I²S forma um conjunto de sinais unidirecionais que se conectam aos pinos de entrada e saída do módulo a ser utilizado para formar os sinais de barramento I²S fora do chip: clock serial (I2SCLK), seleção de palavras (I2SWS), entrada de dados (I2SSDI), e dados de

saída (I2SSDO). Para reduzir a contagem de pinos no nível do chip, os sinais da interface do barramento I²S podem ser compartilhados com outras funções no chip por meio de um controlador GPIO (General Purpose I / O).

Em resumo, o sensor OV7670 envia os dados em um formato síncrono paralelo ao microcontrolador. Primeiramente, para obter quaisquer dados provenientes do sensor, faz-se necessário fornecer um sinal de clock no pino XCLK. De acordo com o datasheet do fabricante, este clock deve ter uma freqüência entre 10 e 48MHz.

Depois que um sinal de clock for aplicado ao pino XCLK, o OV7670 começará a conduzir seus pinos VSYNC, HREF e D0-D7. A Figura 6 apresenta o comportamento dos sinais acima mencionados.

Figura 6 – Sincronização horizontal do *módulo OV7670.*



Fonte: (OMNIVISION,2005).

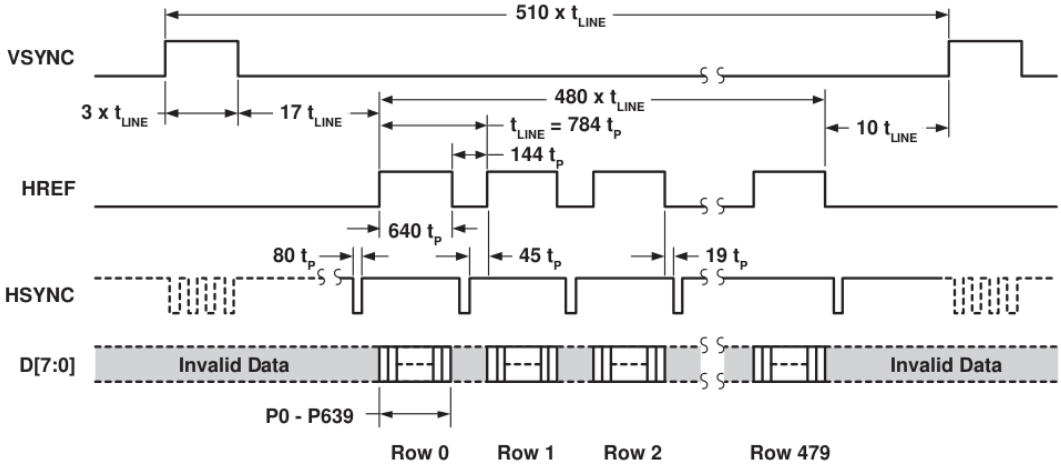
Como é possível notar por meio da Figura 6, deve-se amostrar os pinos D0-D7 na borda de descida do sinal PCLK. Adicionalmente, estes pinos devem ser amostrados apenas na borda de subida de HREF.

Nota-se que todos esses bytes amostrados quando o HREF estava alto correspondem aos pixels em uma linha. É possível observar que um byte não é um pixel, este depende do formato escolhido. Por padrão, o formato é YCbCr422, e isso significa que, em média, dois bytes correspondem a um pixel.

A Figura 7 a seguir apresenta os sinais associados a um quadro "VGA" (640 x 480). Durante o estado alto HSYNC, deve-se capturar 640 pixels, o que é equivalente a uma linha. As 480 linhas, equivalentes a um quadro, são capturadas durante o estado baixo do VSYNC. Isso significa que a borda de descida do

VSYNC sinaliza o início de um quadro e sua borda de subida sinaliza o final de um quadro.

Figura 7 – Temporização VGA do *módulo OV7670.*

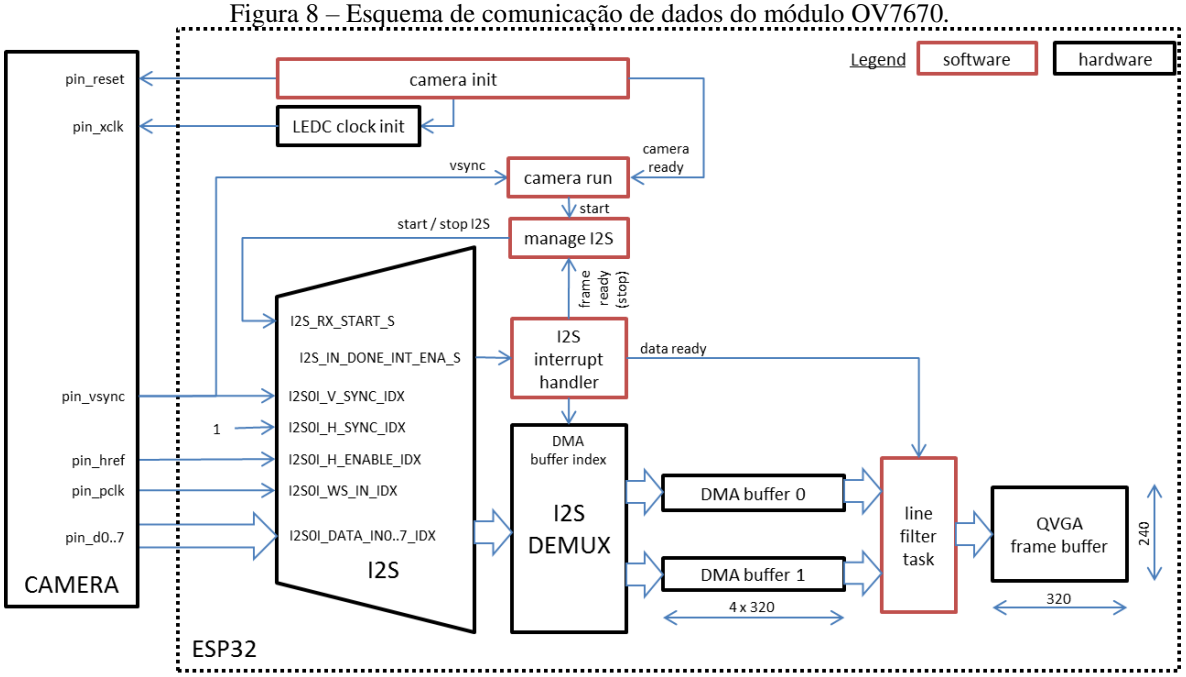


Fonte: (OMNIVISION,2005).

Para a inicialização e testes do módulo câmera OV7670, foram desenvolvidos códigos em linguagem C++, linguagem de programação compatível com o compilador do arduino, que estão contidos nos *apêndices B e D.*

Todas as informações sobre a comunicação da câmera estão descritas no diagrama exemplificado da Figura 8 , diagrama esse fornecido pela fabricante da placa ESP32, *Espressif*.

Figura 8 – Esquema de comunicação de dados do módulo OV7670.



Fonte: (GROKHOTKOV,2017).

Na Figura 9 a seguir apresenta-se um exemplo demonstrativo de utilização do módulo OV7670 em conjunto com utilização com módulo ESP32, realizado por Ivan Grokhotkov, engenheiro de software da *Espressif*. Maiores detalhes da execução do projeto proposto por este trabalho serão encontrados no tópico *Desenvolvimento,* no decorrer deste trabalho.

Figura 9 – Demonstração de uso do módulo OV7670.

Fonte: (GROKHOTKOV,2017).

5 APLICATIVO ANDROID

Devido a necessidade de controlar o sistema desenvolvido nesse projeto de forma remota, foi desenvolvido um aplicativo para celulares androids. Tal aplicativo foi desenvolvido dentro da plataforma *MIT App Inventor,* a qual será tratada a seguir.

O App Inventor para Android é um aplicativo da Web de código aberto fornecido originalmente pelo Google e agora mantido pelo Massachusetts Institute of Technology (MIT).

Ele permite que iniciantes em programação criem aplicativos de software para o SO Android. Fazendo uso de uma interface gráfica, muito semelhante ao Scratch da interface de usuário do StarLogo TNG, a interface do *APP Inventor* permite aos usuários arrastar e soltar objetos visuais para criar um aplicativo que pode ser executado em dispositivos Android.

O App Inventor permite desenvolver aplicativos para telefones que utilizam Android, usando um navegador da Web e um telefone ou emulador conectado. Os servidores do App Inventor armazenam seu trabalho e ajudam o usuário a acompanhar seus projetos. Na plataforma de criação, o usuário trabalha dentro de duas vertentes:

- O *App Inventor Designer*, no qual o usuário seleciona os componentes para seu aplicativo.
- O *App Inventor Blocks Editor*, no qual o usuário monta blocos de programa que especificam como os componentes devem se comportar. É possível realizar montagens visualmente, juntando as peças.

Dentro do tópico *Desenvolvimento* é apresentado o aplicativo que foi desenvolvido nesse trabalho bem como o diagrama de blocos à este associado.

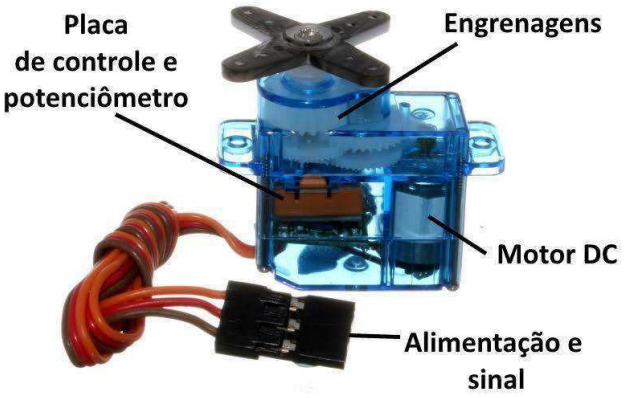
6 COMPONENTES ELETRÔNICOS E ELETRO-MECÂNICOS

Neste tópico são apresentados componentes e circuitos que foram utilizados para os testes e aplicações dos módulos descritos anteriormente.

6.1 SERVO MOTORES

O servo-motor (Figura 10) é um componente eletromecânico que recebe sinais elétricos, os transforma em um movimento giratório de seu eixo, sendo esse movimento proporcional ao sinal recebido. As suas características mais importantes são o seu torque (força rotacional), velocidade de operação, peso, tamanho e consumo.

Figura 10 - Servo-motor.

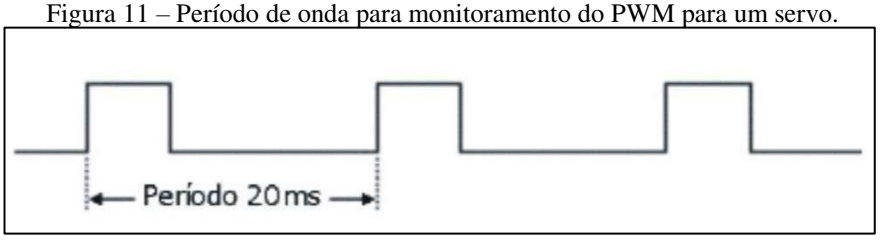


Fonte:(KOYANAGI,2017).

Os servo-motores possuem três terminais: terra (GND), alimentação (Vcc) e o terceiro terminal é destinado ao pino de controle. Este pino de controle define o ângulo de rotação do motor servo, e é determinado pela duração do pulso que se aplica nessa entrada. O servo funciona recebendo um sinal no formato PWM (*Pulse Width Modulation*, modulação por largura de pulso), sistema que consiste em gerar uma onda quadrada em que se varia a duração do pulso, mantendo o período da onda. Este sinal é de 0 volts ou 5 volts. O circuito

de controle do servo monitora este sinal em intervalos de 20 ms, se dentro deste intervalo ele perceber uma alteração do sinal de 0 V para 5 V durante 1 ms até 2 ms, ele modifica a posição do seu eixo para coincidir com o sinal que recebeu (Figura 11).

Figura 11 – Período de onda para monitoramento do PWM para um servo.

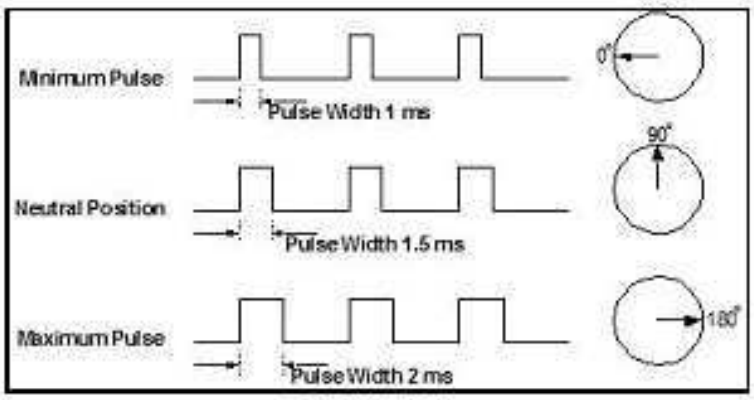


Fonte:(KOYANAGI,2017).

A largura mínima e máxima do pulso (sinal recebido) depende do tipo de servo. No entanto, e no caso geral, se o servo receber na sua entrada um pulso com a duração de 1.5ms, o seu eixo roda até atingir estabilidade, correspondendo a um ângulo de 90º para este intervalo. Se receber pulso com a duração de 1ms, roda, no sentido anti-horário, até atingir o limite do intervalo de rotação correspondente a 0º. Se receber pulso com a duração de 2ms, roda, no sentido horário.

Em resumo, pulsos ente 1ms e 1.5ms farão com que o servo rode para posições intermédias entre 0º e 90º, enquanto pulsos entre 1.5ms e 2ms farão com que o servo rode para posições intermédias entre 90º e 180º (Figura 12). Se um mesmo pulso for aplicado ciclicamente na entrada do servo ele mantém-se na mesma posição angular. Sabe-se que a duração dos impulsos e os valores angulares de rotação do eixo dependem dos fabricantes. Os valores indicados acima são valores médios.

Figura 12 – PWM para controle do servo.



Fonte:(KOYANAGI,2017).

Para as montagens realizadas utilizando os servos, quando o mesmo possuía uma característica de torque mais elevada, não era possível sua alimentação por meio do Arduíno, pois a corrente deste não era suficiente para o acionamento do servo, então, foi necessária a utilização de uma fonte externa.

Os canais de alimentação do servo (Vcc e GND, fios vermelho e preto), são conectados diretamente na bateria, e o pino de controle (fio amarelo) no Arduíno. É importante lembrar que os terras da placa e da bateria devem estar conectados, para que todo o circuito tenha a mesma referência.

7 DESENVOLVIMENTO

Inicialmente foram desenvolvidos testes individuais com o módulo Arduino Uno, no sentido de provar a eficácia dos códigos em linguagem de programação C++ gerados para controle e aquisição dos sinais de interesse.

Uma vez da possível transferência dos mesmos códigos para a placa de desenvolvimento ESP32, com pequenas modificações, e, utilizando a mesma plataforma para transferência de dados que o Arduino Uno, testes também foram realizados fazendo uso desta arquitetura de hardware.

Os testes a seguir foram desenvolvidos no sentido de oferecer uma maior mobilidade no momento em que se deseja criar um produto para o mercado, consequentemente oferecer uma maior gama de opções para o cliente escolher qual será o seu melhor custo benefício. A seguir encontra-se uma descrição mais detalhada dos testes realizados.

7.1 CONTROLE DA FECHADURA

No sentido de criação de uma campainha inteligente e de baixo custo, torna-se interessante o controle de abertura e fechamento da porta associada ao sistema.

Inicialmente, buscou-se no mercado brasileiro soluções para controle de portas por meio de fechaduras que fossem capazes de ser acionadas por meio do envio e recebimento de sinais elétricos. Após um período de intensa pesquisa, a tabela 4 a seguir apresenta os modelos mais comuns encontrados no mercado bem como seus respectivos preços, em cotação realizada entre os dias 04/06/2018 à 13/06/2018.

Tabela 4 – Comparativo entre preços de fechaduras eletrônicas encontradas no Brasil.

Modelo	Fabricante	Preço
AGL-10	AGL Brasil	R$ 87,96
Fx 2000	Intelbras	R$ 134,99
F90	Ipec Eletrônica	R$ 87,96
Trava eletromecânica	Bulher	R$ 84,90
C-90	HDL	R$ 253,90
KIT2ECO	Ipec Eletrônica	R$ 148,99
FV33ICR	Amelco	R$ 299,57
Fe 786	Soprano	R$ 169,00
YRD 221	Yale	R$ 999,00
Gate Lock RCG	Leroy Merlin	R$ 134,90
E150	G-Locks	R$ 799,00

Fonte: Próprio autor.

Diante do exposto, manifestou-se a necessidade de inovação do produto em busca da maior acessibilidade ao público alvo deste trabalho. O conceito de inovação é bastante variado, dependendo, principalmente, da sua aplicação. De forma sucinta, inovação é a exploração com sucesso de novas ideias.

Desta forma, foi desenvolvida uma unidade de acionamento mecânica composta por engrenagens que possuem seu controle baseadas no funcionamento de um motor servo, previamente visto nos tópicos anteriores. Após repetitivos testes, foi comprovada que esta unidade possui a capacidade de abertura de uma fechadura convencional. Outrossim, seu peso é suficientemente pequeno para que este possa ser edificado a qualquer tipo de porta, no caso da criação de um novo produto para o mercado.

A ideia fundamental no momento da criação desta unidade se deu no fato de que este dispositivo pode adaptar-se a qualquer tipo de chave, uma vez que seu princípio de funcionamento está relacionado ao encaixe de uma chave comum, na extremidade do seu eixo de rotação. Portanto, não há necessidade do cliente fazer a troca de sua fechadura por completo, no momento da aquisição do produto aqui desenvolvido. O que não é verdade, para fechaduras convencionais encontradas no mercado brasileiro atualmente.

Em suma, a unidade mecânica aqui exposta possui um terminal onde se é possível fazer o encaixe de qualquer chave comum, onde a unidade estará fixada à porta associada a esta chave. A Figura 13 a seguir apresenta a estrutura

do módulo desenvolvida. A mesma estrutura aqui apresentada foi utilizada para os testes descritos nos tópicos subsequentes.

Figura 13 – Protótipo desenvolvido para fechadura inteligente.

Fonte: Próprio autor

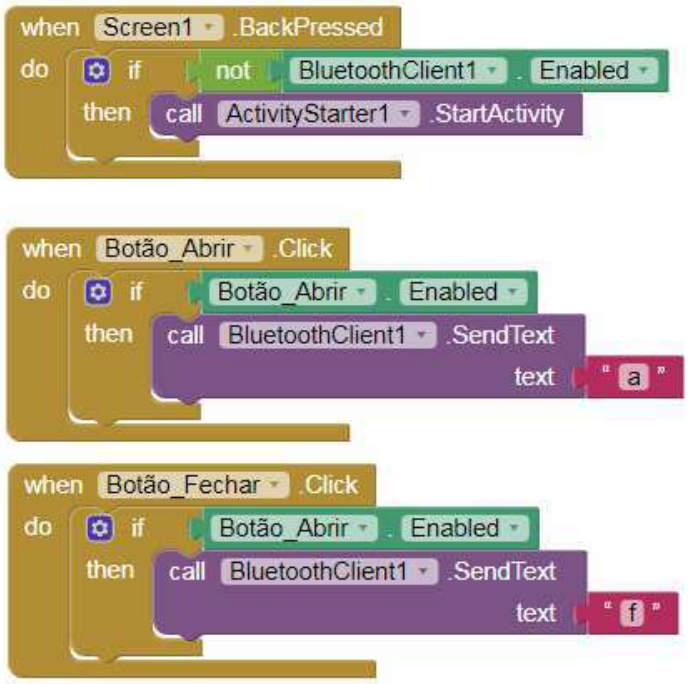
7.2 UTILIZAÇÃO DOS MÓDULOS VIA BLUETOOTH

A seguir apresenta-se uma descrição referente aos testes realizados com os módulos HC-05 e OV7670 em conjunto com o Arduino Uno, descrito no tópico 7.1.1, bem como a utilização com a placa ESP32, descrito no tópico 7.1.2, e, consequentemente, o desenvolvimento de cada aplicativo de controle do sistema proposto a ser utilizado na plataforma Android.

7.2.1 CONEXÃO ARDUINO UNO COM MÓDULOS HC-05 E 0V7670 E MOTOR SERVO

A princípio foram desenvolvidos os códigos em linguagem de programação C++, que constam no *apêndice A* e *apêndice B,* para controle dos módulos utilizados para este teste. Em um segundo momento, foi realizado o desenvolvimento de um aplicativo para controle dos sinais de interesse. A Figura 16 apresenta a interface do aplicativo utilizado para controle e aquisição dos sinais gerados e adquiridos e a Figura 19 posteriormente apresenta o diagrama em blocos do núcleo principal do aplicativo.

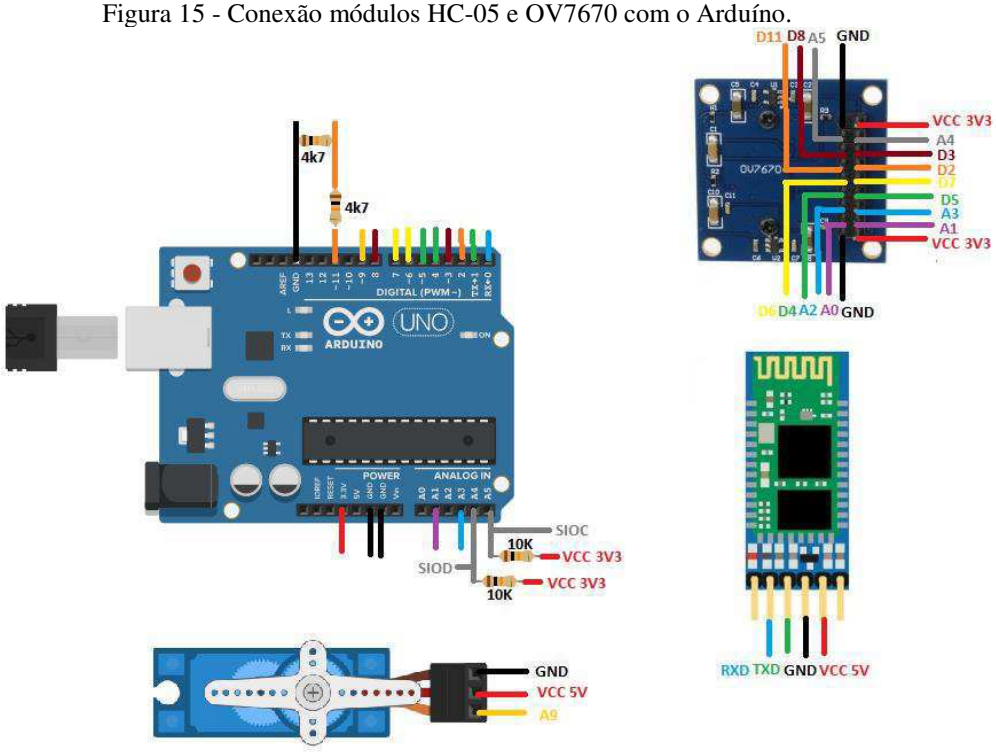
Figura 14 – Diagrama de blocos do aplicativo Android para testes com HC-05 e OV7670.



Fonte: Próprio autor

Em um terceiro momento, realizou-se a conexão física dos componentes para enfim realizar o teste. A conexão dos componentes é ilustrada na figura 15.

Figura 15 - Conexão módulos HC-05 e OV7670 com o Arduíno.



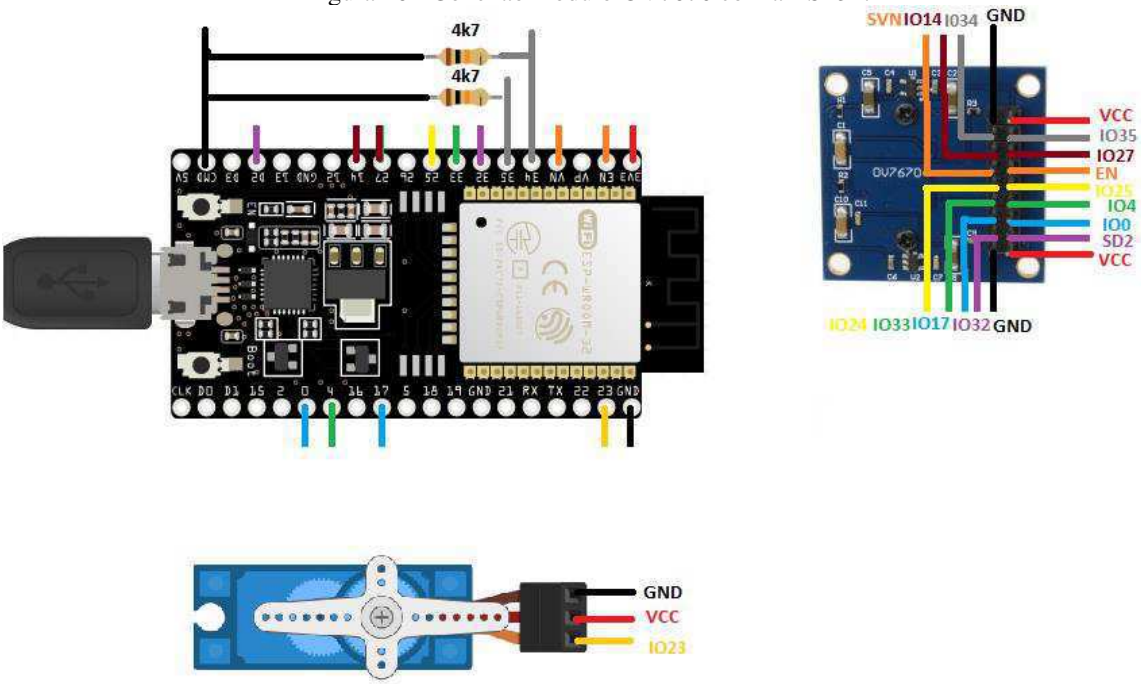
Fonte: Próprio autor.

7.2.2 UTILIZAÇÃO DA PLACA ESP32 COM MÓDULO OV7670 E MOTOR SERVO VIA BLUETOOTH

Para realização da montagem proposta por este tópico, é importante destacar que o módulo ESP32 já inclui dentro de suas diversas funcionalidades um formato para conexão Bluetooth, e, consequentemente, dispensa o uso do módulo HC-05 que foi utilizado na montagem do tópico anterior.

Inicialmente foi desenvolvido o código em linguagem de programação C++ que consta no apêndice *C* para controle de sinais entre a placa ESP32 e o aplicativo Android à esta associado, que foi incrementado posteriormente. A Figura 16 a seguir apresenta a conexão realizada para o teste. A interface do aplicativo desenvolvido bem com seu diagrama de blocos segue o padrão desenvolvido na Figura 18 a ser revelada posteriormente no decorrer deste trabalho.

Figura 16 - Conexão módulo OV7670 com a ESP32.



Fonte: Próprio autor.

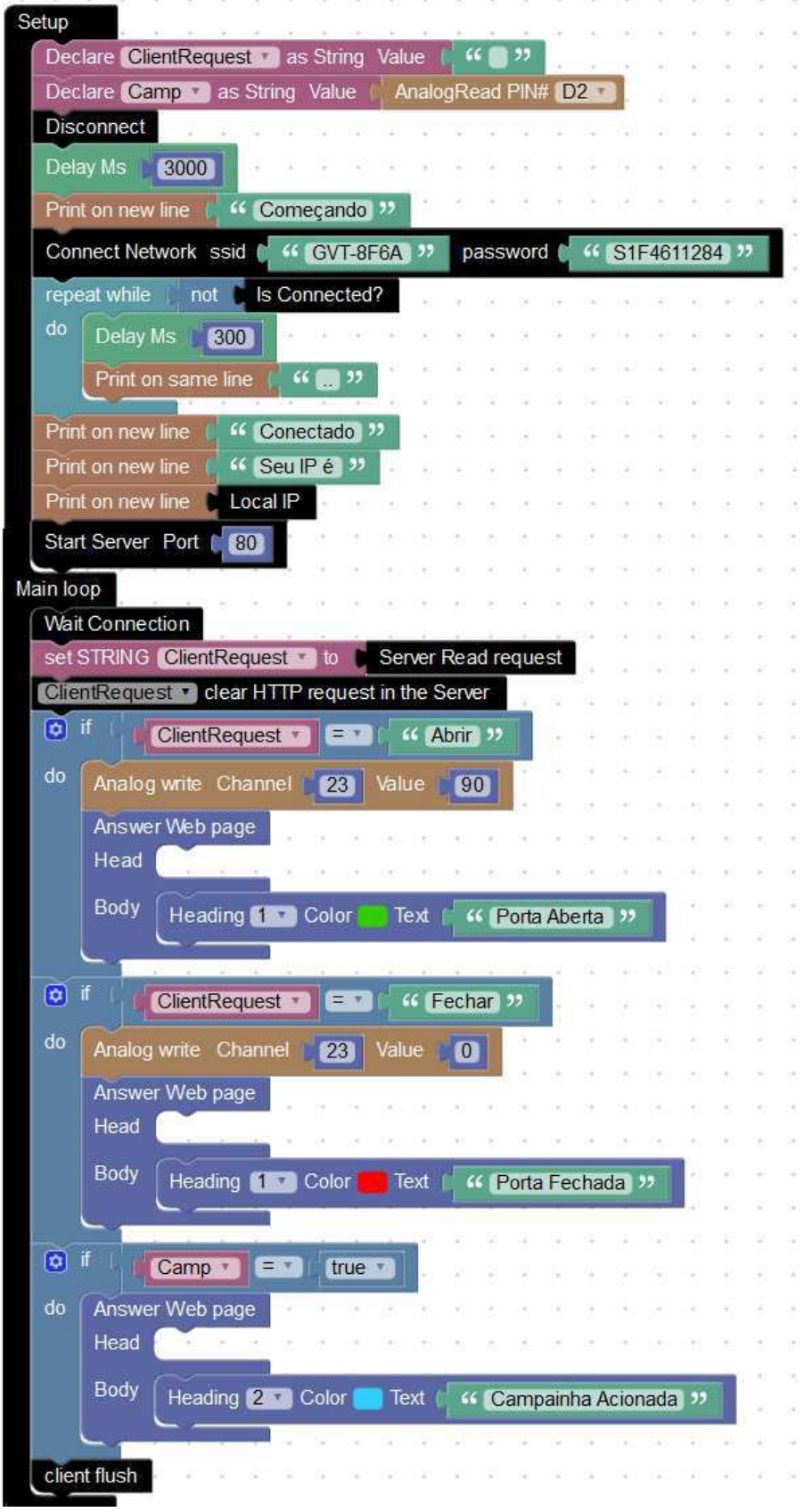
7.3 UTILIZAÇÃO DA PLACA ESP32 COM MÓDULO OV7670 E MOTOR SERVO VIA WIFI

Neste tópico será apresentado o teste realizado com a placa de desenvolvimento ESP32 no modo de utilização pela rede *Wifi*. Em virtude do fato de que a ESP32 já possui a função de *Wifi* integrado ao seu chip, não foram realizados testes utilizando o Arduino Uno em conjunto a algum módulo compatível com *Wifi* em consequência da constante busca da redução de custos proposta por este trabalho.

Inicialmente, faz-se necessário fazer com que a ESP32 trabalhe no formato de um servidor web. Isto significa que a ESP32 funcionará como um sistema de computadores em que irá hospedar um site e gerenciar as suas informações. A comunicação entre o usuário e o servidor se dá pela decomposição do endereço da página (URL), pelo browser (domínio, nome e protocolo da página) e, em seguida, o domínio (DNS) traduz a informação fornecida pelo usuário para o endereço IP (combinação numérica do endereço real do site na web) para que o browser determine qual ação será tomada. Para instauração do servidor web proposto, foi utilizada a rede *Wifi* local.

Consequentemente, uma vez estabelecida essa conexão faz-se possível a troca de informações entre a ESP32 e quaisquer meios externos via internet. Neste momento, é incorporado ao sistema o aplicativo Android desenvolvido e responsável pelo envio das informações de demanda provenientes do usuário do aplicativo. O aplicativo, por sua vez, faz acesso ao site previamente desenvolvido pelo servidor web para que seja possível a troca de informações entre o aparelho celular e o protótipo final. A montagem experimental utilizada foi a mesma apresentada na figura 16.

Figura 17 – Configuração da web server.

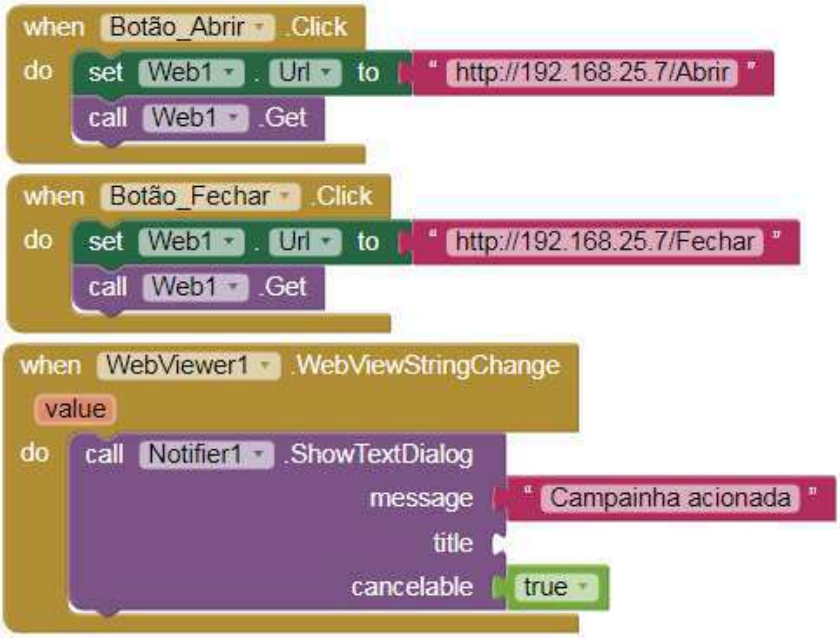


Fonte: Próprio autor.

Figura 18 – Interface do aplicativo.

Fonte: Próprio autor.

Figura 19 – Configuração do aplicativo para funcionamento via wifi.



Fonte: Próprio autor.

7.4 RESULTADOS

Os testes realizados via tecnologia Bluetooth demonstraram razoável resposta com relação aos objetivos propostos. Conseguiu-se a abertura e fechamento à distância da fechadura associada ao teste, porém constatou-se que o envio de imagens via tal técnica não seria viável devido a limitações técnicas provenientes das plataformas empregadas nos testes. Adicionalmente, os testes realizados via Bluetooth demonstraram uma maior limitação com relação ao raio de alcance dos dispositivos associados aos testes.

É de fundamental importância identificar pontos críticos no desenvolvimento de novos produtos para o mercado, consequentemente, vale salientar que para o protótipo aqui confeccionado, a tecnologia Bluetooth demonstrou-se de baixa segurança, uma vez que o acesso e controle ao dispositivo poderia ser alcançado por diferentes aparelhos, dentro do raio de alcance da campainha, com maior facilidade quando comparado com os testes realizados utilizando uma rede *Wifi*.

Com relação aos testes realizados via *Wifi*, estes demonstraram uma maior confiabilidade com relação aos quesitos de segurança bem como um maior raio de alcance quando comparado com os testes executados via tecnologia Bluetooth. O fato do aplicativo de controle da fechadura estar associado ao endereço configurado pelo webserver local, traz ao usuário uma maior confiabilidade nos quesitos de segurança. Além disso, a montagem proposta utilizando uma configuração via *Wifi* demonstrou que é possível o envio e recebimento de imagens uma vez da utilização dos módulos aqui propostos. Portanto, a configuração de controle do protótipo que utiliza *Wifi* em sua base, por meio da placa ESP32, é a mais indicada em casos em que deseja-se ter maior segurança com relação à fechadura associada ao protótipo.

No que se diz respeito ao preço do protótipo, tem-se que o objetivo de criação de um item de menor valor, quando comparado com o mercado brasileiro, atingiu o seu objetivo pois o valor estimado do protótipo tornou-se menor que todos os outros tipos de fechaduras inteligentes encontradas no mercado brasileiro, como evidenciado na tabela 5 a seguir, além de sua funcionalidade não requerer que o usuário troque inteiramente sua fechadura já existente, uma vez que o protótipo desenvolvido possui a característica de adaptação à qualquer tipo de fechadura existente no mercado brasileiro.

Não foi objetivo deste trabalho a implementação de uma fonte de tensão para o protótipo, uma vez que o objetivo principal foi o de desenvolvimento de um dispositivo que demonstrasse funcionalidade e autonomia. A simulação de um pulso de tensão proveniente de uma campainha convencional foi executado por meio de uma botoeira, ficando a par de uma melhor adaptação ao mercado convencional ser desenvolvida mais futuramente, uma vez que maior parte dos objetivos principais aqui propostos foram atingidos.

Tabela 5 – Orçamento campainha inteligente.

Modelo	**Preço**
ESP-32	R$ 22,00
Camera OV7670	R$ 16,00
Base para chave	R$ 5,00
Motor servo	R$ 8,00
Fios	R$ 5,39
Resistores	R$ 0,26
---	---
Total	R$ 56,65

Fonte: Próprio autor.

8 CONCLUSÃO

A execução deste projeto de conclusão de curso proporcionou ao concluinte a possibilidade de aprender duas novas arquiteturas de hardware, o microcontrolador Arduino Uno, bem como a ESP32. Além de um software de interface gráfica, o App Inventor 2, e o software de comunicação com os microcontroladores, o Arduino IDE. Apesar das dificuldades e da necessidade de adaptação a cada etapa do projeto, seja por necessidade de adaptação devida falta de materiais disponíveis no mercado, seja por parte da falta de material encontrado em livros e na internet, acredita-se que o conhecimento adquirido pelo autor durante a execução do projeto seja capaz de no futuro proporcionar maior aprendizado sobre tais microcontroladores e gerar correções sobre os erros de execução do projeto para que este seja implementado completamente como foi pensado no começo do período letivo, ou seja, o protótipo ser controlado remotamente e possuir acesso à vídeo sobre quem está a tocar a campainha.

Mesmo com tais adaptações necessárias para que o projeto fosse implementado com sucesso, e mesmo fugindo um pouco a ideia inicial, considera-se que o projeto como bem sucedido, pois, a interface gráfica do software para plataforma Android, foi elaborada com sucesso, assim como almejado desde o começo do trabalho, e a comunicação entre o hardware, o microcontrolador ESP32, e a interface gráfica, por meio da comunicação *wifi* foi executada com sucesso, onde considerou-se como um dos maiores desafios a serem superados na execução do projeto, dentre outros desafios superados durante o vigente período letivo.

BIBLIOGRAFIA

Aníbal Ferreira, [et al.]; *Comunicações audiovisuais : tecnologias, normas e aplicações*. ed. Fernando Pereira ; rev. Luís Filipe Coelho. - 1ª ed. - Lisboa : IST Press, cop. 2009. - XXII, 938 p. : il. ;

G. Côté, B. Erol, M. Gallant, F. Kossentini, H,263+: vídeo coding at low bit rates, *IEEE Trans. on Circuits and Systems for Video Technology* 8(7):849-866,1998

M. Horowitz, A. Joch, F. Kossentini, A. Hallapuro, H.264/AVC baseline profile decoder complexity analysis, *IEEE Trans. on Circuits and Systems for Video Technology* 13(7): 704-716, 2003.

ISO, Home Page. http://www.iso.org

ISO/IEC:2001, Coding of audio-visual objects, Part 1: Systems, 2nd Edition, ISO/IEC Standard 14496-1, 2001

ISO/IEC:2001, Coding of audio-visual objects, Part 2: Visual, 2nd Edition, ISO/IEC Standard 14496-2, 2001

A. Puri, A. Wong, Spatial domain resolution scalable video coding, *Visual Communications and Image Processing,* Proc. of the SPIE, vol. 1199, pp. 718-729, 1993.

Assis, P. D. (Dezembro de 2004). MICROCONTROLADOR, Trabalho de conclusão de curso da UNIPAC - UNIVERSIDADE PRESIDENTE ANTÔNIO CARLOS. Barbacena, Minas Gerais, Brasil.

Azevedo, C. M. (Setembro de 2010). Comando e Monitorização de Sistemas de Actuação Via Rede Wireless - ZigBee. *Dissertação de Mestrado da Faculdade de Engenharia - Universidade do Porto*.

Canzian, E. (2009). Comunicação Serial – RS232. Cotia, São Paulo, Brasil.

Cunha, L. (2009). Relés e Contatores. *Setor Elétrico*, 54 - 60.

Ferreira, I. A. (Março de 2008). Sistemas de controle e supervisão de sistemas embebidos - Tipo SCADA. *Dissertação de Mestrado da Faculdade de Engenharia - Universidade do Porto*.

Leite, S. G. (Março de 2007). Comunicação Wireless - CURSO DE GESTÃO DE REDES - UNIFACS – UNIVERSIDADE SALVADOR. Salvador, Bahia, Brasil.

Neto, B. B., Monteiro, P. d., & Queiroga, S. L. (2012). Aplicabilidade dos Microcontroladores em Inovações Tecnológicas. Palmas, Tocantins, Brasil.

Okuda, F. M. (2008). Bluetooth, Trabalho de conclusão de curso da Universidade de São Paulo. São Paulo.

Penido, É. d., & Trindade, R. S. (2013). Microcontroladores - IFMG. Ouro Preto, Minas Gerais, Brasil.

Reis, F. d. (24 de Abril de 2015). Introdução aos Microcontroladores. São Paulo, São Paulo, Brasil.

Ribeiro, T. J., & Oliveira, S. C. (2016). Projeto de um Gateway para Automação Residencial. *Revista de Engenharia e Pesquisa Aplicada*, 43-49.

Sampaio, A. F. (2008). SISTEMA DE AUTOMAÇÃO wireless para controle de dispositivos autônomos. Brasília, DF, Brasil.

Santos, W. S., Pereira, M. F., Pereira, R. B., & Neto, E. J. (2016). Miniestacão agrometeorológica baseada na plataforma ESP8266 para aplicações agrícolas, UFMT. Cuiabá, Mato Grosso, Brasil.

Silva, A. O., & Sales, V. M. (2016). Desenvolvimento de um controlador feedback adaptativo para posicionamento de uma antena parabólica receptora - Trabalho de Conclusão do curso Engenharia de Controle e Automação - IFRJ - Fluminense. Campos dos Goytacazes, Rio de Janeiro, Brasil.

Sousa, D. J. (2000). *Desbravando o PIC: Baseado no microcontrolador PIC 16F84. 5ª ed.* São Paulo: Érica.

Sousa, M. B. (2002). *Wireless: Sistemas de rede sem fio.* Rio de janeiro: Brasport.

APÊNDICE A – UPLOAD DO PROGRAMA PARA O UTILIZAÇÃO DA MÓDULO BLUETOOTH HC-05 COM ARDUÍNO UNO

```
#include <Servo.h>
static const int servoPin = 9;

//Programa para simples controle do servo por meio do bluetooth
void setup() {
  Serial.begin(9600);
  myservo.attach(9);
}

void loop() {
  if(Serial.available()>0)
  {
    char data = Serial.read();
    if (data == 'a'){
    myservo.write(90);
    delay(15);    }
    else if(data == 'f'){
      myservo.write(0);
      delay(15);
    }
  }
}
```

APÊNDICE B – UPLOAD DO PROGRAMA PARA O UTILIZAÇÃO DA MÓDULO CÂMERA OV7670 COM ARDUÍNO UNO

```
#include <stdint.h>
#include <avr/io.h>
#include <util/twi.h>
#include <util/delay.h>
#include <avr/pgmspace.h>

#define F_CPU 16000000UL
#define vga   0


#define qvga  1
#define qqvga   2
#define yuv422  0
#define rgb565  1
#define bayerRGB  2
#define camAddr_WR  0x42
#define camAddr_RD  0x43

/* Registradores */
#define REG_GAIN            0x00  /* Ganho menor que 8 bits */
#define REG_BLUE            0x01  /* Ganho da cor azul */
#define REG_RED             0x02  /* Ganho da cor Vermelha */
#define REG_VREF            0x03  /* Registradores GAIN, VSTART, VSTOP */
#define REG_COM1            0x04  /* Controle da porta 1 */
#define COM1_CCIR656        0x40    /* CCIR656 ativado */
#define REG_BAVE            0x05  /* Nível médio U/B */
#define REG_GbAVE           0x06  /* Nível médio Y/Gb */
#define REG_AECHH           0x07  /* AEC MS 5 bits */
#define REG_RAVE            0x08  /* Nível médio V/R */
#define REG_COM2            0x09  /* Controle porta 2 */
#define COM2_SSLEEP         0x10  /*Modo Soft sleep */
#define REG_PID             0x0a  /* Produto ID MSB */
#define REG_VER             0x0b  /* Produto ID LSB */
#define REG_COM3            0x0c  /* Controle porta 3 */
#define COM3_SWAP           0x40  /*Scan de bytes */
#define COM3_SCALEEN        0x08  /* Escalonamento */
#define COM3_DCWEN          0x04  /* Habilitar downsamp/crop/window */
#define REG_COM4            0x0d  /* Controle porta 4 */
#define REG_COM5            0x0e  /* Todos "reservedos" */
#define REG_COM6            0x0f  /* Controle porta 6 */
#define REG_AECH            0x10  /* Bits de valor AEC */
#define REG_CLKRC           0x11  /* Controle do Clock */
#define CLK_EXT             0x40  /* Uso do clock externo diretamente */
#define CLK_SCALE           0x3f  /* Mascara para escala do clock interno */
```

```
#define REG_COM7            0x12  /* Controle da porta 7 */
#define COM7_RESET          0x80  /*Reset dos registradores */
#define COM7_FMT_MASK       0x38    /* Formato MASK */
#define COM7_FMT_VGA        0x00    /* Formato VGA */
#define COM7_FMT_CIF        0x20  /* Formato CIF */
#define COM7_FMT_QVGA       0x10  /* Formato QVGA */
#define COM7_FMT_QCIF       0x08  /* Formato QCIF */
#define COM7_RGB            0x04  /* Formato RGB para bits 0 e 2 */
#define COM7_YUV            0x00  /*Formato YUV */
#define COM7_BAYER          0x01  /*Formato Bayer */
#define COM7_PBAYER         0x05  /* "Processador bayer" */
#define REG_COM8            0x13  /* Controle porta 8 */
#define COM8_FASTAEC        0x80  /* Habilitar AGC/AEC */
#define COM8_AECSTEP        0x40  /* Tamanho de passo ilimitado para AEC step*/
#define COM8_BFILT          0x20 /* Habilitação do filtro de Banda de passagem */
#define COM8_AGC            0x04  /* Habilitação do auto-ganho */
#define COM8_AWB            0x02  /* Habilitação do balanço de branco */
#define COM8_AEC            0x01  /* Habilitação de auto-exposição */
#define REG_COM9            0x14  /* Controle 9 – teto de ganho */
#define REG_COM10           0x15  /* Controle 10 – teto de ganho */
#define COM10_HSYNC         0x40  /* HSYNC no lugar de HREF */
#define COM10_PCLK_HB       0x20  /* PCLK para espaço horizontal */
#define COM10_HREF_REV      0x08  /* HREF reverso */
#define COM10_VS_LEAD       0x04 /* VSYNC sobre a borda de interesse do clock*/
#define COM10_VS_NEG        0x02  /* VSYNC negativo */
#define COM10_HS_NEG        0x01  /* HSYNC negativo */
#define REG_HSTART          0x17  /* Começo horizontal dos bits */
#define REG_HSTOP           0x18  /* Parada horizontal dos bits*/
#define REG_VSTART          0x19  /* Começo vertical dos bits */
#define REG_VSTOP           0x1a  /* Parada vertical dos bits */
#define REG_PSHFT           0x1b  /* Atraso de pixel após HREF */
#define REG_MIDH            0x1c  /* Manuf. ID high */
#define REG_MIDL            0x1d  /* Manuf. ID low */
#define REG_MVFP            0x1e  /* Espelhamento / vflip */
#define MVFP_MIRROR         0x20  /* Imagem do refletida */
#define MVFP_FLIP           0x10  /* Espelhamento vertical */
#define REG_AEW             0x24  /* AGC limite superior */
#define REG_AEB             0x25    /* AGC limite inferior */
#define REG_VPT             0x26  /* AGC/AEC modo de operação da região */
#define REG_HSYST           0x30  /* HSYNC atraso da borda de subida */
#define REG_HSYEN           0x31  /* HSYNC atraso da borda de descida */
#define REG_HREF            0x32  /* Dados HREF */
#define REG_TSLB            0x3a  /* Dados TSLB*/
#define TSLB_YLAST          0x04  /* UYVY ou VYUY – ver porta com13 */
#define REG_COM11           0x3b  /* Controle  porta 11 */
#define COM11_NIGHT         0x80  /* Modo night habilitado */
#define COM11_NMFR          0x60  /* Taxa de 2 bits/quadro */
#define COM11_HZAUTO        0x10  /* Auto-detecção 50/60 Hz */
#define COM11_50HZ          0x08  /* Manual 50Hz selecionado */
#define COM11_EXP           0x02 /* Controle de exposição */
```

```
#define REG_COM12          0x3c  /* Controle porta 12 */
#define COM12_HREF         0x80  /*Sempre HREF  */
#define REG_COM13          0x3d  /* Controle porta 13 */
#define COM13_GAMMA        0x80  /* Habilitação Gamma */
#define COM13_UVSAT        0x40  /*Auto ajuste de saturação UV */
#define COM13_UVSWAP       0x01  /* V antes de U – com TSLB */
#define REG_COM14          0x3e  /* Controle porta 14 */
#define COM14_DCWEN        0x10  /* DCW/PCLK – habilitação de escala */
#define REG_EDGE           0x3f  /* Fator de aprimoramento da borda */
#define REG_COM15          0x40  /* Controle porta 15 */
#define COM15_R10F0        0x00  /* Alcance de dados - 10 à F0 */
#define COM15_R01FE        0x80  /* 01 à FE */
#define COM15_R00FF        0xc0  /*  00 à FF */
#define COM15_RGB565       0x10  /* RGB565 saída */
#define COM15_RGB555       0x30  /* RGB555 saída */
#define REG_COM16          0x41  /* Controle porta 16 */
#define COM16_AWBGAIN      0x08  /* AWB - habilitação de ganho */
#define REG_COM17          0x42  /* Controle porta 17 */
#define COM17_AECWIN       0xc0  /* anela AEC – precisa combinar com COM4 */
#define COM17_CBAR         0x08  /* Barra de cores DSP */

/*
* Esta matrix define como as cores são geradas This matrix defines how the colors are
generated, e precisa ser alterada para ajuste dos níveis UHE e Saturação
*
* Ordem: v-red, v-green, v-blue, u-red, u-green, u-blue
* Estes são sinais de 9 bits, com armazenamento no formato in0x58.
* Sinal para v-red é bit 0, e sobem a partir deste ponto.
*/

#define CMATRIX_LEN        6
#define REG_BRIGHT         0x55  /* Ajuste de brilho */
#define REG_REG76          0x76  /* Nome do OV */
#define R76_BLKPCOR        0x80  /* Correção de pixels pretros */
#define R76_WHTPCOR        0x40  /* Correção de pixels  brancos*/
#define REG_RGB444         0x8c  /* RGB 444 controle */
#define R444_ENABLE        0x02  /* Liga RGB444, escreve 5x5 */
#define R444_RGBX          0x01  /* Limpeza de memória no fim */
#define REG_HAECC1         0x9f  /* Hist AEC/AGC controle 1 */
#define REG_HAECC2         0xa0  /* Hist AEC/AGC controle 2 */
#define REG_BD50MAX        0xa5  /* 50hz limite de banda */
#define REG_HAECC3         0xa6  /* Hist AEC/AGC controle 3 */
#define REG_HAECC4         0xa7  /* Hist AEC/AGC controle 4 */
#define REG_HAECC5         0xa8  /* Hist AEC/AGC controle 5 */
#define REG_HAECC6         0xa9  /* Hist AEC/AGC controle 6 */
#define REG_HAECC7         0xaa  /* Hist AEC/AGC controle 7 */
#define REG_BD60MAX        0xab  /* 60hz banding step limit */
#define MTX1               0x4f  /* Matrix Coef. 1 */
#define MTX2               0x50  /* Matrix Coef. 2 */
#define MTX3               0x51  /* Matrix Coef. 3 */
```

```
#define MTX4                    0x52  /* Matrix Coef. 4 */
#define MTX5                    0x53  /* Matrix Coef. 5 */
#define MTX6                    0x54  /* Matrix Coef. 6 */
#define REG_CONTRAS             0x56  /* Controle de contraste */
#define MTXS                    0x58  /* Matrix de sinais de coef. */
#define AWBC7                   0x59  /* AWB Controle 7 */
#define AWBC8                   0x5a  /* AWB Controle 8 */
#define AWBC9                   0x5b  /* AWB Controle 9 */
#define AWBC10                  0x5c  /* AWB Controle 10 */
#define AWBC11                  0x5d  /* AWB Controle 11 */
#define AWBC12                  0x5e  /* AWB Controle 12 */
#define REG_GFI                 0x69  /* Fix gain Controle */
#define GGAIN                   0x6a  /* Canal G  ganho AWB */
#define DBLV                    0x6b   /*Ganho DBLV*/
#define AWBCTR3                 0x6c  /* AWB Controle 3 */
#define AWBCTR2                 0x6d  /* AWB Controle 2 */
#define AWBCTR1                 0x6e  /* AWB Controle 1 */
#define AWBCTR0                 0x6f  /* AWB Controle 0 */

struct regval_list{
  uint8_t reg_num;
  uint16_t value;
};

const struct regval_list qvga_ov7670[] PROGMEM = {
  { REG_COM14, 0x19 },
  { 0x72, 0x11 },
  { 0x73, 0xf1 },

  { REG_HSTART, 0x16 },
  { REG_HSTOP, 0x04 },
  { REG_HREF, 0xa4 },
  { REG_VSTART, 0x02 },
  { REG_VSTOP, 0x7a },
  { REG_VREF, 0x0a },
  { REG_HSTART, 0x16 },
  { REG_HSTOP, 0x04 },
  { REG_HREF, 0x24 },
  { REG_VSTART, 0x02 },
  { REG_VSTOP, 0x7a },
  { REG_VREF, 0x0a },
  { 0xff, 0xff }, /* Fim de marcador */
};

const struct regval_list yuv422_ov7670[] PROGMEM = {
  { REG_COM7, 0x0 },  /* Seleciona modo YUV */
  { REG_RGB444, 0 },
  { REG_COM1, 0 },
  { REG_COM15, COM15_R00FF },
  { REG_COM9, 0x6A }, /* 128x limite de ganho; 0x8 bit reservado */
```

```
  { 0x4f, 0x80 },   /* "matrix coef. 1" */
  { 0x50, 0x80 },   /* "matrix coef. 2" */
  { 0x51, 0 },    /* vb */
  { 0x52, 0x22 },   /* "matrix coef. 4" */
  { 0x53, 0x5e },   /* "matrix coef. 5" */
  { 0x54, 0x80 },   /* "matrix coef. 6" */
  { REG_COM13, COM13_UVSAT },
  { 0xff, 0xff },   /* Fim de marcador */
};

const struct regval_list ov7670_default_regs[] PROGMEM = {
  { REG_COM7, COM7_RESET },
  { REG_TSLB, 0x04 }, /* OV */
  { REG_COM7, 0 },  /* VGA */

  { REG_HSTART, 0x13 }, { REG_HSTOP, 0x01 },
  { REG_HREF, 0xb6 }, { REG_VSTART, 0x02 },
  { REG_VSTOP, 0x7a }, { REG_VREF, 0x0a },

  { REG_COM3, 0 }, { REG_COM14, 0 },
  /* Escalonamento */
  { 0x70, 0x3a }, { 0x71, 0x35 },
  { 0x72, 0x11 }, { 0x73, 0xf0 },
  { 0xa2,/* 0x02 mudou para to 1*/1 }, { REG_COM10, 0x0 },
  /* Valores das curvas Gamma */
  { 0x7a, 0x20 }, { 0x7b, 0x10 },
  { 0x7c, 0x1e }, { 0x7d, 0x35 },
  { 0x7e, 0x5a }, { 0x7f, 0x69 },
  { 0x80, 0x76 }, { 0x81, 0x80 },
  { 0x82, 0x88 }, { 0x83, 0x8f },
  { 0x84, 0x96 }, { 0x85, 0xa3 },
  { 0x86, 0xaf }, { 0x87, 0xc4 },
  { 0x88, 0xd7 }, { 0x89, 0xe8 },
  /* Parametros AGC e AEC.  Primeiramente são desabilitados,
  e são abilitados novamente após os valores ajustados. */
  { REG_COM8, COM8_FASTAEC | COM8_AECSTEP },
  { REG_GAIN, 0 }, { REG_AECH, 0 },
  { REG_COM4, 0x40 }, /* magic reserved bit */
  { REG_COM9, 0x18 }, /* 4x gain + magic rsvd bit */
  { REG_BD50MAX, 0x05 }, { REG_BD60MAX, 0x07 },
  { REG_AEW, 0x95 }, { REG_AEB, 0x33 },
  { REG_VPT, 0xe3 }, { REG_HAECC1, 0x78 },
  { REG_HAECC2, 0x68 }, { 0xa1, 0x03 }, /* magic */
  { REG_HAECC3, 0xd8 }, { REG_HAECC4, 0xd8 },
  { REG_HAECC5, 0xf0 }, { REG_HAECC6, 0x90 },
  { REG_HAECC7, 0x94 },
  { REG_COM8, COM8_FASTAEC | COM8_AECSTEP | COM8_AGC | COM8_AEC
},
  { 0x30, 0 }, { 0x31, 0 },//disable some delays
  /* Almost all of these are magic "reserved" values.  */
```

```
  { REG_COM5, 0x61 }, { REG_COM6, 0x4b },
  { 0x16, 0x02 }, { REG_MVFP, 0x07 },
  { 0x21, 0x02 }, { 0x22, 0x91 },
  { 0x29, 0x07 }, { 0x33, 0x0b },
  { 0x35, 0x0b }, { 0x37, 0x1d },
  { 0x38, 0x71 }, { 0x39, 0x2a },
  { REG_COM12, 0x78 }, { 0x4d, 0x40 },
  { 0x4e, 0x20 }, { REG_GFIX, 0 },
 /*{0x6b, 0x4a},*/{ 0x74, 0x10 },
  { 0x8d, 0x4f }, { 0x8e, 0 },
  { 0x8f, 0 }, { 0x90, 0 },
  { 0x91, 0 }, { 0x96, 0 },
  { 0x9a, 0 }, { 0xb0, 0x84 },
  { 0xb1, 0x0c }, { 0xb2, 0x0e },
  { 0xb3, 0x82 }, { 0xb8, 0x0a },

  /* Balaço de branco */
  { 0x43, 0x0a }, { 0x44, 0xf0 },
  { 0x45, 0x34 }, { 0x46, 0x58 },
  { 0x47, 0x28 }, { 0x48, 0x3a },
  { 0x59, 0x88 }, { 0x5a, 0x88 },
  { 0x5b, 0x44 }, { 0x5c, 0x67 },
  { 0x5d, 0x49 }, { 0x5e, 0x0e },
  { 0x6c, 0x0a }, { 0x6d, 0x55 },
  { 0x6e, 0x11 }, { 0x6f, 0x9e },
  { 0x6a, 0x40 }, { REG_BLUE, 0x40 },
  { REG_RED, 0x60 },
  { REG_COM8, COM8_FASTAEC | COM8_AECSTEP | COM8_AGC | COM8_AEC |
COM8_AWB },

/* Matrix de coeficientes */
  { 0x4f, 0x80 }, { 0x50, 0x80 },
  { 0x51, 0 },    { 0x52, 0x22 },
  { 0x53, 0x5e }, { 0x54, 0x80 },
  { 0x58, 0x9e },

  { REG_COM16, COM16_AWBGAIN }, { REG_EDGE, 0 },
  { 0x75, 0x05 }, { REG_REG76, 0xe1 },
  { 0x4c, 0 },     { 0x77, 0x01 },
  { REG_COM13, /*0xc3*/0x48 }, { 0x4b, 0x09 },
  { 0xc9, 0x60 },   /*{REG_COM16, 0x38},*/
  { 0x56, 0x40 },

  { 0x34, 0x11 }, { REG_COM11, COM11_EXP | COM11_HZAUTO },
  { 0xa4, 0x82}, { 0x96, 0 },
  { 0x97, 0x30 }, { 0x98, 0x20 },
  { 0x99, 0x30 }, { 0x9a, 0x84 },
  { 0x9b, 0x29 }, { 0x9c, 0x03 },
  { 0x9d, 0x4c }, { 0x9e, 0x3f },
  { 0x78, 0x04 },
```

```
  /* Multiplexadores de registradores */
  { 0x79, 0x01 }, { 0xc8, 0xf0 },
  { 0x79, 0x0f }, { 0xc8, 0x00 },
  { 0x79, 0x10 }, { 0xc8, 0x7e },
  { 0x79, 0x0a }, { 0xc8, 0x80 },
  { 0x79, 0x0b }, { 0xc8, 0x01 },
  { 0x79, 0x0c }, { 0xc8, 0x0f },
  { 0x79, 0x0d }, { 0xc8, 0x20 },
  { 0x79, 0x09 }, { 0xc8, 0x80 },
  { 0x79, 0x02 }, { 0xc8, 0xc0 },
  { 0x79, 0x03 }, { 0xc8, 0x40 },
  { 0x79, 0x05 }, { 0xc8, 0x30 },
  { 0x79, 0x26 },
  { 0xff, 0xff }, /* Fim de marcador */
};

void error_led(void){
  DDRB |= 32;//LED como saída
  while (1){/espera reset
    PORTB ^= 32;// piscar led
    _delay_ms(100);
  }
}

void twiStart(void){
  TWCR = _BV(TWINT) | _BV(TWSTA) | _BV(TWEN);//Envia start
  while (!(TWCR & (1 << TWINT)));//Espera transmissão de start
  if ((TWSR & 0xF8) != TW_START)
    error_led();
}

void twiWriteByte(uint8_t DATA, uint8_t type){
  TWDR = DATA;
  TWCR = _BV(TWINT) | _BV(TWEN);
  while (!(TWCR & (1 << TWINT))) {}
  if ((TWSR & 0xF8) != type)
    error_led();
}

void twiAddr(uint8_t addr, uint8_t typeTWI){
  TWDR = addr;//Envia endereço
  TWCR = _BV(TWINT) | _BV(TWEN);/* Limpeza interrupt para
                                   começar transmissão */
  while ((TWCR & _BV(TWINT)) == 0); /* Espera para transmissão */
  if ((TWSR & 0xF8) != typeTWI)
    error_led();
}

void wrReg(uint8_t reg, uint8_t dat){
```

```
//Envio da condição de start
  twiStart();
  twiAddr(camAddr_WR, TW_MT_SLA_ACK);
  twiWriteByte(reg, TW_MT_DATA_ACK);
  twiWriteByte(dat, TW_MT_DATA_ACK);
  TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);//Envio do stop
  _delay_ms(1);
}

static uint8_t twiRd(uint8_t nack){
  if (nack){
    TWCR = _BV(TWINT) | _BV(TWEN);
    while ((TWCR & _BV(TWINT)) == 0); /* Espera transmissão */
    if ((TWSR & 0xF8) != TW_MR_DATA_NACK)
      error_led();
    return TWDR;
  }
  else{
    TWCR = _BV(TWINT) | _BV(TWEN) | _BV(TWEA);
    while ((TWCR & _BV(TWINT)) == 0); /* Espera transmissão */
    if ((TWSR & 0xF8) != TW_MR_DATA_ACK)
      error_led();
    return TWDR;
  }
}

uint8_t rdReg(uint8_t reg){
  uint8_t dat;
  twiStart();
  twiAddr(camAddr_WR, TW_MT_SLA_ACK);
  twiWriteByte(reg, TW_MT_DATA_ACK);
  TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);//Envia stop
  _delay_ms(1);
  twiStart();
  twiAddr(camAddr_RD, TW_MR_SLA_ACK);
  dat = twiRd(1);
  TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);//Envia stop
  _delay_ms(1);
  return dat;
}

void wrSensorRegs8_8(const struct regval_list reglist[]){
  uint8_t reg_addr, reg_val;
  const struct regval_list *next = reglist;
  while ((reg_addr != 0xff) | (reg_val != 0xff)){
    reg_addr = pgm_read_byte(&next->reg_num);
    reg_val = pgm_read_byte(&next->value);
    wrReg(reg_addr, reg_val);
    next++;
```

```
  }
}

void setColor(void){
  wrSensorRegs8_8(yuv422_ov7670);
}

void setRes(void){
  wrReg(REG_COM3, 4); // REG_COM3 habilitação de escalonamento
  wrSensorRegs8_8(qvga_ov7670);
}

void camInit(void){
  wrReg(0x12, 0x80);
  _delay_ms(100);
  wrSensorRegs8_8(ov7670_default_regs);
  wrReg(REG_COM10, 32);//PCLK não se altera(toggle) em HBLANK.
}

void arduinoUnoInut(void) {
  cli();//disabilita interrupções

    /* Setup do clock de 8MHz do PWM
  * Estara associado ao pino 11*/
  DDRB |= (1 << 3);//pino 11
  ASSR &= ~(_BV(EXCLK) | _BV(AS2));
  TCCR2A = (1 << COM2A0) | (1 << WGM21) | (1 << WGM20);
  TCCR2B = (1 << WGM22) | (1 << CS20);
  OCR2A = 0;//(F_CPU)/(2*(X+1))
  DDRC &= ~15;//Baixo d0-d3 da camera
  DDRD &= ~252;//d7-d4 e pinos de interrupção
  _delay_ms(3000);

    //Inicia p/ 100khz
  TWSR &= ~3;//Desabilita preescalador para TWI
  TWBR = 72;//Inicia para 100khz
     //habilita serial
  UBRR0H = 0;
  UBRR0L = 1; /*Taxas de trans.: 0 = 2M. 1 = 1M. 3 = 0.5M. 7 = 250k ,207 é 9600 */.
  UCSR0A |= 2;//velocidade aysnc
  UCSR0B = (1 << RXEN0) | (1 << TXEN0); //Habilita trans. e receptor
  UCSR0C = 6; //async 1 p/ bit 8bit char sem bits de paridade
}
void StringPgm(const char * str){
  do{
     while (!(UCSR0A & (1 << UDRE0)));//Espera transmissão de byte pelo serial
     UDR0 = pgm_read_byte_near(str);
     while (!(UCSR0A & (1 << UDRE0)));//Espera byte de transmissão
  } while (pgm_read_byte_near(++str));
}
```

```
static void captureImg(uint16_t wg, uint16_t hg){
  uint16_t y, x;

  StringPgm(PSTR("*RDY*"));

  while (!(PIND & 8));//Espera para high
  while ((PIND & 8));//Espera para low
    y = hg;
  while (y--){
       x = wg;
     //while (!(PIND & 256));//Espera para high
    while (x--){
     while ((PIND & 4));//Espera para low
          UDR0 = (PINC & 15) | (PIND & 240);
        while (!(UCSR0A & (1 << UDRE0)));//espera byte de transmissão
     while (!(PIND & 4));//Espera para high
     while ((PIND & 4));//Espera para low
     while (!(PIND & 4));//Espera para high    }
   //  while ((PIND & 256));//Espera para low
}
   _delay_ms(100);}
void setup(){
  arduinoUnoInut();
  camInit();
  setRes();
  setColor();
  wrReg(0x11, 10); }

void loop(){
  captureImg(320, 240);}
```

APÊNDICE C – UPLOAD DO PROGRAMA PARA O UTILIZAÇÃO DA PLACA ESP32 VIA BLUETOOTH

```
/*
  Código para criação de uma conexão BLE que uma vez estabelecida, irá enviar dados de
confirmação periodicamente
  6E400002-B5A3-F393-E0A9-E50E24DCCA9E - usado para receber dados "WRITE"
  6E400003-B5A3-F393-E0A9-E50E24DCCA9E - usado para enviar dados  "NOTIFY"
*/

#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

#include <Servo.h>
static const int servoPin = 23;
Servo servo1;

BLECharacteristic *pCharacteristic;
bool deviceConnected = false;
float txValue = 0;

#define SERVICE_UUID "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
// Servico UART UUID
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-
E50E24DCCA9E"

#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-
E50E24DCCA9E"

class MyServerCallbacks: public BLEServerCallbacks {
   void onConnect(BLEServer* pServer) {
     deviceConnected = true;
   };

   void onDisconnect(BLEServer* pServer) {
     deviceConnected = false;
   }
};

class MyCallbacks: public BLECharacteristicCallbacks {
   void onWrite(BLECharacteristic *pCharacteristic) {
     std::string rxValue = pCharacteristic->getValue();

     if (rxValue.length() > 0) {
       Serial.println("*********");
       Serial.print("Received Value: ");
```

```
      for (int i = 0; i < rxValue.length(); i++) {
        Serial.print(rxValue[i]);
      }

      Serial.println();

      // Do stuff based on the command received from the app
      if (rxValue.find("A") != -1) {
        Serial.println("Abrindo porta");
        servo1.write(0);
        delay(20);
      }
      else if (rxValue.find("F") != -1) {
        Serial.println("Fechando Porta");
        servo1.write(90);
        delay(20);
      }
      }

      Serial.println();
      Serial.println("*********");
    }
  }
};

void setup() {
  Serial.begin(115200);
  servo1.attach(servoPin);

  // Criando um dispositivo BLE
  BLEDevice::init("ESP32 UART Test"); // Give it a name

  // Criando um servidor BLE
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());

  // Criando uma BLE
  BLEService *pService = pServer->createService(SERVICE_UUID);

  // Criando uma BLE
  pCharacteristic = pService->createCharacteristic(
                      CHARACTERISTIC_UUID_TX,
                      BLECharacteristic::PROPERTY_NOTIFY
                    );

  pCharacteristic->addDescriptor(new BLE2902());

  BLECharacteristic *pCharacteristic = pService->createCharacteristic(
                                         CHARACTERISTIC_UUID_RX,
```

```
                                  BLECharacteristic::PROPERTY_WRITE
                                );

  pCharacteristic->setCallbacks(new MyCallbacks());

  // Comeco de servico
  pService->start();

  // Aviso
  pServer->getAdvertising()->start();
  Serial.println("Espera de uma conexao com cliente...");
}

void loop() {
  if (deviceConnected) {

  }
  delay(1000);
}
```

APÊNDICE D – UPLOAD DO PROGRAMA PARA O CONTROLE DA PLACA ESP32 COM MÓDULO OV7670 VIA WIFI

```
/*ESP32_I2S_Camera*/
#include "OV7670.h"

#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library

#include <WiFi.h>
#include <WiFiMulti.h>
#include <WiFiClient.h>
#include "BMP.h"

const int SIOD = 25; //SDA
const int SIOC = 23; //SCL

const int VSYNC = 22;
const int HREF = 26;

const int XCLK = 27;
const int PCLK = 21;

const int D0 = 35;
const int D1 = 17;
const int D2 = 34;
const int D3 = 5;
const int D4 = 39;
const int D5 = 18;
const int D6 = 36;
const int D7 = 19;

#define ssid1        "Republica"
#define password1    "republica01"

OV7670 *camera;

WiFiMulti wifiMulti;
WiFiServer server(80);

unsigned char bmpHeader[BMP::headerSize];

void serve()
```

```
{
  WiFiClient client = server.available();
  if (client)
  {
    //Serial.println("New Client.");
    String currentLine = "";
    while (client.connected())
    {
      if (client.available())
      {
        char c = client.read();
        //Serial.write(c);
        if (c == '\n')
        {
          if (currentLine.length() == 0)
          {
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println();
            client.print(
              "<style>body{margin: 0}\nimg{height: 100%; width: auto}</style>"
              "<img id='a' src='/camera' onload='this.style.display=\"initial\"; var b = document.getElementById(\"b\"); b.style.display=\"none\"; b.src=\"camera?\"+Date.now(); '>"
              "<img id='b' style='display: none' src='/camera' onload='this.style.display=\"initial\"; var a = document.getElementById(\"a\"); a.style.display=\"none\"; a.src=\"camera?\"+Date.now(); '>");
            client.println();
            break;
          }
          else
          {
            currentLine = "";
          }
        }
        else if (c != '\r')
        {
          currentLine += c;
        }

        if(currentLine.endsWith("GET /camera"))
        {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-type:image/bmp");
          client.println();

          for(int i = 0; i < BMP::headerSize; i++)
            client.write(bmpHeader[i]);
          for(int i = 0; i < camera->xres * camera->yres * 2; i++)
            client.write(camera->frame[i]);
```

```
        }
       }
     }
     // close the connection:
     client.stop();
     //Serial.println("Client Disconnected.");
   }
}

void setup()
{
  Serial.begin(115200);

  wifiMulti.addAP(ssid1, password1);
  Serial.println("Connecting Wifi...");
  if(wifiMulti.run() == WL_CONNECTED) {
      Serial.println("");
      Serial.println("WiFi connected");
      Serial.println("IP address: ");
      Serial.println(WiFi.localIP());
  }

  camera = new OV7670(OV7670::Mode::QQVGA_RGB565, SIOD, SIOC, VSYNC,
HREF, XCLK, PCLK, D0, D1, D2, D3, D4, D5, D6, D7);
  BMP::construct16BitHeader(bmpHeader, camera->xres, camera->yres);

  server.begin();
}

void loop()
{
  camera->oneFrame();
  serve();
}

/* BMP.h*/

#pragma once

//assuming pixel lines have multiples of 4 bytes sizes
class BMP
{

  static void setChar(void *buffer, int pos, char ch)
  {
    *(char*)(buffer+pos) = ch;
  }

  static void setLong(void *buffer, int pos, long l)
  {
```

```
    *(long*)(buffer+pos) = l;
  }

  static void setShort(void *buffer, int pos, short s)
  {
    *(short*)(buffer+pos) = s;
  }

public:
  static const int headerSize = 54 + 12;

  static void construct16BitHeader(void *buffer, long xres, long yres)
  {
    setChar(buffer, 0, 'B');
    setChar(buffer, 1, 'M');

    long bytesPerLine = xres * 2;
    setLong(buffer, 2, bytesPerLine * yres  + 54 + 12); //filesize
    setLong(buffer, 6, 0);

    setLong(buffer, 10, 54 + 12); //offset

    setLong(buffer, 14, 40); //header size
    setLong(buffer, 18, xres);
    setLong(buffer, 22, yres);
    setShort(buffer, 26, 1); //planes
    setShort(buffer, 28, 16); //bits

    setLong(buffer, 30, 3); //compression 3 = bit fields
    setLong(buffer, 38, 0); //x pix per meter
    setLong(buffer, 42, 0); //y pix per meter

    setLong(buffer, 46, 0); //biClrUsed
    setLong(buffer, 50, 0); //biClrImportant

    setLong(buffer, 54, 0xF800); //R mask
    setLong(buffer, 58, 0x07E0); //G mask
    setLong(buffer, 62, 0x001F); //B mask
  }
};

/*DMABuffer.h */

#pragma once

class DMABuffer
{
  public:
  lldesc_t descriptor;
  unsigned char* buffer;
```

```
  DMABuffer(int bytes)
  {
    buffer = (unsigned char *)malloc(bytes);
    descriptor.length = bytes;
    descriptor.size = descriptor.length;
    descriptor.owner = 1;
    descriptor.sosf = 1;
    descriptor.buf = (uint8_t*) buffer;
    descriptor.offset = 0;
    descriptor.empty = 0;
    descriptor.eof = 1;
    descriptor.qe.stqe_next = 0;
  }

  void next(DMABuffer *next)
  {
    descriptor.qe.stqe_next = &(next->descriptor);
  }

  int sampleCount() const
  {
    return descriptor.length / 4;
  }

  ~DMABuffer()
  {
    if(buffer)
      delete(buffer);
  }
};

/* I2C.h*/
#include "Arduino.h"

class I2C
{
  void inline DELAY()
  {
    delayMicroseconds(1);
  }

  void inline SCLLOW()
  {
    pinMode(SCL, OUTPUT);
    digitalWrite(SCL, 0);
  }

  void inline SCLHIGH()
  {
    pinMode(SCL, INPUT_PULLUP);
```

```
  digitalWrite(SCL, 1);
}

void inline CLOCK()
{
  DELAY();
  SCLHIGH();
  DELAY();
  DELAY();
  SCLLOW();
  DELAY();
}

void inline SDALOW()
{
  pinMode(SDA, OUTPUT);
  digitalWrite(SDA, 0);
}

void inline SDAHIGH()
{
  pinMode(SDA, OUTPUT);
  digitalWrite(SDA, 1);
}

void inline SDAPULLUP()
{
  pinMode(SDA, INPUT_PULLUP);
}

void pushByte(unsigned char b)
{
  for(char i = 0; i < 8; i++)
  {
    if(b & 0x80)
      SDAHIGH();
    else
      SDALOW();
    b <<= 1;
    CLOCK();
  }
}

bool getAck()
{
  SDAPULLUP();
  DELAY();
  SCLHIGH();
  DELAY();
  int r = digitalRead(SDA);
```

```
  SDALOW();
  DELAY();
  SCLLOW();
  DELAY();
  return r == 0;
}

void start()
{
  SDAPULLUP();
  DELAY();
  SCLHIGH();
  DELAY();
  SDALOW();
  DELAY();
  SCLLOW();
  DELAY();
}

void end()
{
  SCLHIGH();
  DELAY();
  SDAPULLUP();
  DELAY();
}

public:
int SDA;
int SCL;
I2C(const int data, const int clock)
{
  SDA = data;
  SCL = clock;
  pinMode(SDA, INPUT_PULLUP);
  pinMode(SCL, INPUT_PULLUP);
  digitalWrite(SDA, 0);
  digitalWrite(SCL, 0);
}

bool writeRegister(unsigned char addr, unsigned char reg, unsigned char data)
{
  start();
  pushByte(addr);

  if(!getAck())
  {
    end();
    return false;
  }
```

```
    pushByte(reg);
    if(!getAck())
    {
      end();
      return false;
    }

    pushByte(data);
    if(!getAck())
    {
      end();
      return false;
    }

    end();
    return true;
  }
};

/*I2SCamera.cpp*/
#include "I2SCamera.h"
#include "Log.h"

int I2SCamera::blocksReceived = 0;
int I2SCamera::framesReceived = 0;
int I2SCamera::xres = 640;
int I2SCamera::yres = 480;
gpio_num_t I2SCamera::vSyncPin = (gpio_num_t)0;
intr_handle_t I2SCamera::i2sInterruptHandle = 0;
intr_handle_t I2SCamera::vSyncInterruptHandle = 0;
int I2SCamera::dmaBufferCount = 0;
int I2SCamera::dmaBufferActive = 0;
DMABuffer **I2SCamera::dmaBuffer = 0;
unsigned char* I2SCamera::frame = 0;
int I2SCamera::framePointer = 0;
int I2SCamera::frameBytes = 0;
volatile bool I2SCamera::stopSignal = false;

void IRAM_ATTR I2SCamera::i2sInterrupt(void* arg)
{
    I2S0.int_clr.val = I2S0.int_raw.val;
    blocksReceived++;
    unsigned char* buf = dmaBuffer[dmaBufferActive]->buffer;
    dmaBufferActive = (dmaBufferActive + 1) % dmaBufferCount;
    if(framePointer < frameBytes)
      for(int i = 0; i < xres * 4; i += 4)
      {
        frame[framePointer++] = buf[i + 2];
        frame[framePointer++] = buf[i];
```

```
      }
    if (blocksReceived == yres)
    {
      framePointer = 0;
      blocksReceived = 0;
      framesReceived++;
      if(stopSignal)
      {
        i2sStop();
        stopSignal = false;
      }
    }
    //    i2sStop();
}

void IRAM_ATTR I2SCamera::vSyncInterrupt(void* arg)
{
    GPIO.status1_w1tc.val = GPIO.status1.val;
    GPIO.status_w1tc = GPIO.status;
    if(gpio_get_level(vSyncPin))
    {
      //frame done
    }
}

void I2SCamera::i2sStop()
{
    esp_intr_disable(i2sInterruptHandle);
    esp_intr_disable(vSyncInterruptHandle);
    i2sConfReset();
    I2S0.conf.rx_start = 0;
}

void I2SCamera::i2sRun()
{
    DEBUG_PRINTLN("I2S Run");
    while (gpio_get_level(vSyncPin) == 0);
    while (gpio_get_level(vSyncPin) != 0);

    esp_intr_disable(i2sInterruptHandle);
    i2sConfReset();
    blocksReceived = 0;
    dmaBufferActive = 0;
    framePointer = 0;
    DEBUG_PRINT("Sample count ");
    DEBUG_PRINTLN(dmaBuffer[0]->sampleCount());
    I2S0.rx_eof_num = dmaBuffer[0]->sampleCount();
    I2S0.in_link.addr = (uint32_t)&(dmaBuffer[0]->descriptor);
    I2S0.in_link.start = 1;
    I2S0.int_clr.val = I2S0.int_raw.val;
```

```
    I2S0.int_ena.val = 0;
    I2S0.int_ena.in_done = 1;
    esp_intr_enable(i2sInterruptHandle);
    esp_intr_enable(vSyncInterruptHandle);
    I2S0.conf.rx_start = 1;
}

bool I2SCamera::initVSync(int pin)
{
  DEBUG_PRINT("Initializing VSYNC... ");
  vSyncPin = (gpio_num_t)pin;
  gpio_set_intr_type(vSyncPin, GPIO_INTR_POSEDGE);
  gpio_intr_enable(vSyncPin);
  if(gpio_isr_register(&vSyncInterrupt,                    (void*)"vSyncInterrupt",
ESP_INTR_FLAG_INTRDISABLED          |          ESP_INTR_FLAG_IRAM,
&vSyncInterruptHandle) != ESP_OK)
  {
    DEBUG_PRINTLN("failed!");
    return false;
  }
  DEBUG_PRINTLN("done.");
  return true;
}

void I2SCamera::deinitVSync()
{
  esp_intr_disable(vSyncInterruptHandle);
}

bool I2SCamera::init(const int XRES, const int YRES, const int VSYNC, const int HREF,
const int XCLK, const int PCLK, const int D0, const int D1, const int D2, const int D3,
const int D4, const int D5, const int D6, const int D7)
{
  xres = XRES;
  yres = YRES;
  frameBytes = XRES * YRES * 2;
  frame = (unsigned char*)malloc(frameBytes);
  if(!frame)
  {
    DEBUG_PRINTLN("Not enough memory for frame buffer!");
    return false;
  }
  i2sInit(VSYNC, HREF, PCLK, D0, D1, D2, D3, D4, D5, D6, D7);
  dmaBufferInit(xres * 2 * 2);  //two bytes per dword packing, two bytes per pixel
  initVSync(VSYNC);
  return true;
}
```

```
bool I2SCamera::i2sInit(const int VSYNC, const int HREF, const int PCLK, const int D0,
const int D1, const int D2, const int D3, const int D4, const int D5, const int D6, const int
D7)
{
 int pins[] = {VSYNC, HREF, PCLK, D0, D1, D2, D3, D4, D5, D6, D7};
 gpio_config_t conf = {
  .pin_bit_mask = 0,
  .mode = GPIO_MODE_INPUT,
  .pull_up_en = GPIO_PULLUP_DISABLE,
  .pull_down_en = GPIO_PULLDOWN_DISABLE,
  .intr_type = GPIO_INTR_DISABLE
 };
  for (int i = 0; i < sizeof(pins) / sizeof(gpio_num_t); ++i) {
     conf.pin_bit_mask = 1LL << pins[i];
     gpio_config(&conf);
  }

  // Route input GPIOs to I2S peripheral using GPIO matrix, last parameter is invert
  gpio_matrix_in(D0,   I2S0I_DATA_IN0_IDX, false);
  gpio_matrix_in(D1,   I2S0I_DATA_IN1_IDX, false);
  gpio_matrix_in(D2,   I2S0I_DATA_IN2_IDX, false);
  gpio_matrix_in(D3,   I2S0I_DATA_IN3_IDX, false);
  gpio_matrix_in(D4,   I2S0I_DATA_IN4_IDX, false);
  gpio_matrix_in(D5,   I2S0I_DATA_IN5_IDX, false);
  gpio_matrix_in(D6,   I2S0I_DATA_IN6_IDX, false);
  gpio_matrix_in(D7,   I2S0I_DATA_IN7_IDX, false);
  gpio_matrix_in(0x30,  I2S0I_DATA_IN8_IDX, false);
  gpio_matrix_in(0x30,  I2S0I_DATA_IN9_IDX, false);
  gpio_matrix_in(0x30,  I2S0I_DATA_IN10_IDX, false);
  gpio_matrix_in(0x30,  I2S0I_DATA_IN11_IDX, false);
  gpio_matrix_in(0x30,  I2S0I_DATA_IN12_IDX, false);
  gpio_matrix_in(0x30,  I2S0I_DATA_IN13_IDX, false);
  gpio_matrix_in(0x30,  I2S0I_DATA_IN14_IDX, false);
  gpio_matrix_in(0x30,  I2S0I_DATA_IN15_IDX, false);

  gpio_matrix_in(VSYNC, I2S0I_V_SYNC_IDX, true);
  gpio_matrix_in(0x38,  I2S0I_H_SYNC_IDX, false);  //0x30 sends 0, 0x38 sends 1
  gpio_matrix_in(HREF,  I2S0I_H_ENABLE_IDX, false);
  gpio_matrix_in(PCLK,  I2S0I_WS_IN_IDX, false);

  // Enable and configure I2S peripheral
  periph_module_enable(PERIPH_I2S0_MODULE);

  // Toggle some reset bits in LC_CONF register
  // Toggle some reset bits in CONF register
  i2sConfReset();
  // Enable slave mode (sampling clock is external)
  I2S0.conf.rx_slave_mod = 1;
  // Enable parallel mode
  I2S0.conf2.lcd_en = 1;
```

```
    // Use HSYNC/VSYNC/HREF to control sampling
    I2S0.conf2.camera_en = 1;
    // Configure clock divider
    I2S0.clkm_conf.clkm_div_a = 1;
    I2S0.clkm_conf.clkm_div_b = 0;
    I2S0.clkm_conf.clkm_div_num = 2;
    // FIFO will sink data to DMA
    I2S0.fifo_conf.dscr_en = 1;
    // FIFO configuration
    //two bytes per dword packing
    I2S0.fifo_conf.rx_fifo_mod = SM_0A0B_0C0D;  //pack two bytes in one dword see
:https://github.com/igrr/esp32-cam-demo/issues/29
    I2S0.fifo_conf.rx_fifo_mod_force_en = 1;
    I2S0.conf_chan.rx_chan_mod = 1;
    // Clear flags which are used in I2S serial mode
    I2S0.sample_rate_conf.rx_bits_mod = 0;
    I2S0.conf.rx_right_first = 0;
    I2S0.conf.rx_msb_right = 0;
    I2S0.conf.rx_msb_shift = 0;
    I2S0.conf.rx_mono = 0;
    I2S0.conf.rx_short_sync = 0;
    I2S0.timing.val = 0;

    // Allocate I2S interrupt, keep it disabled
    esp_intr_alloc(ETS_I2S0_INTR_SOURCE, ESP_INTR_FLAG_INTRDISABLED |
ESP_INTR_FLAG_LEVEL1 | ESP_INTR_FLAG_IRAM, &i2sInterrupt, NULL,
&i2sInterruptHandle);
    return true;
}

void I2SCamera::dmaBufferInit(int bytes)
{
  dmaBufferCount = 2;
  dmaBuffer = (DMABuffer**) malloc(sizeof(DMABuffer*) * dmaBufferCount);
  for(int i = 0; i < dmaBufferCount; i++)
  {
    dmaBuffer[i] = new DMABuffer(bytes);
    if(i)
      dmaBuffer[i-1]->next(dmaBuffer[i]);
  }
  dmaBuffer[dmaBufferCount - 1]->next(dmaBuffer[0]);
}

void I2SCamera::dmaBufferDeinit()
{
    if (!dmaBuffer) return;
    for(int i = 0; i < dmaBufferCount; i++)
      delete(dmaBuffer[i]);
    delete(dmaBuffer);
    dmaBuffer = 0;
```

```
    dmaBufferCount = 0;
}

/*I2SCamera.h*/
#pragma once

#include "soc/soc.h"
#include "soc/gpio_sig_map.h"
#include "soc/i2s_reg.h"
#include "soc/i2s_struct.h"
#include "soc/io_mux_reg.h"
#include "driver/gpio.h"
#include "driver/periph_ctrl.h"
#include "rom/lldesc.h"
#include "XClk.h"
#include "DMABuffer.h"

class I2SCamera
{
  public:
  static gpio_num_t vSyncPin;
  static int blocksReceived;
  static int framesReceived;
  static int xres;
  static int yres;
  static intr_handle_t i2sInterruptHandle;
  static intr_handle_t vSyncInterruptHandle;
  static int dmaBufferCount;
  static int dmaBufferActive;
  static DMABuffer **dmaBuffer;
  static unsigned char* frame;
  static int framePointer;
  static int frameBytes;
  static volatile bool stopSignal;

  typedef enum {
    /* camera sends byte sequence: s1, s2, s3, s4, ...
     * fifo receives: 00 s1 00 s2, 00 s2 00 s3, 00 s3 00 s4, ...
     */
    SM_0A0B_0B0C = 0,
    /* camera sends byte sequence: s1, s2, s3, s4, ...
     * fifo receives: 00 s1 00 s2, 00 s3 00 s4, ...
     */
    SM_0A0B_0C0D = 1,
    /* camera sends byte sequence: s1, s2, s3, s4, ...
     * fifo receives: 00 s1 00 00, 00 s2 00 00, 00 s3 00 00, ...
     */
    SM_0A00_0B00 = 3,
  } i2s_sampling_mode_t;
```

```
  static inline void i2sConfReset()
  {
    const uint32_t lc_conf_reset_flags = I2S_IN_RST_M | I2S_AHBM_RST_M |
I2S_AHBM_FIFO_RST_M;
    I2S0.lc_conf.val |= lc_conf_reset_flags;
    I2S0.lc_conf.val &= ~lc_conf_reset_flags;

    const uint32_t conf_reset_flags = I2S_RX_RESET_M | I2S_RX_FIFO_RESET_M |
I2S_TX_RESET_M | I2S_TX_FIFO_RESET_M;
    I2S0.conf.val |= conf_reset_flags;
    I2S0.conf.val &= ~conf_reset_flags;
    while (I2S0.state.rx_fifo_reset_back);
  }

  void start()
  {
   i2sRun();
  }

  void stop()
  {
   stopSignal = true;
   while(stopSignal);
  }

  void oneFrame()
  {
   start();
   stop();
  }

  static void i2sStop();
  static void i2sRun();

  static void dmaBufferInit(int bytes);
  static void dmaBufferDeinit();

  static bool initVSync(int pin);
  static void deinitVSync();

  static void IRAM_ATTR i2sInterrupt(void* arg);
  static void IRAM_ATTR vSyncInterrupt(void* arg);

  static bool i2sInit(const int VSYNC, const int HREF, const int PCLK, const int D0, const
int D1, const int D2, const int D3, const int D4, const int D5, const int D6, const int D7);

  static bool init(const int XRES, const int YRES, const int VSYNC, const int HREF, const
int XCLK, const int PCLK, const int D0, const int D1, const int D2, const int D3, const
int D4, const int D5, const int D6, const int D7);
```

```
};
/*Log.h*/
#pragma once
#include "Arduino.h"

/*
#define DEBUG_PRINTLN(a) Serial.println(a)
#define DEBUG_PRINT(a) Serial.print(a)
#define DEBUG_PRINTLNF(a, f) Serial.println(a, f)
#define DEBUG_PRINTF(a, f) Serial.print(a, f)
*/
#define DEBUG_PRINTLN(a)
#define DEBUG_PRINT(a)
#define DEBUG_PRINTLNF(a, f)
#define DEBUG_PRINTF(a, f)

/*OV7670.cpp*/
#include "OV7670.h"
#include "XClk.h"
#include "Log.h"

OV7670::OV7670(Mode m, const int SIOD, const int SIOC, const int VSYNC, const int
HREF, const int XCLK, const int PCLK, const int D0, const int D1, const int D2, const
int D3, const int D4, const int D5, const int D6, const int D7)
  :i2c(SIOD, SIOC)
{
  ClockEnable(XCLK, 20000000); //base is 80MHz

  DEBUG_PRINT("Waiting for VSYNC...");
  pinMode(VSYNC, INPUT);
  while(!digitalRead(VSYNC));
  while(digitalRead(VSYNC));
  DEBUG_PRINTLN(" done");

  mode = m;
  switch(mode)
  {
    case VGA_RGB565:
    xres = 640;
    yres = 480;
    break;
    case QVGA_RGB565:
    xres = 320;
    yres = 240;
    break;
    case QQVGA_RGB565:
    xres = 160;
    yres = 120;
    QQVGARGB565();
    break;
```

```
    case QQQVGA_RGB565:
    xres = 80;
    yres = 60;
    QQQVGARGB565();
    break;
    default:
    xres = 0;
    yres = 0;
  }
  //testImage();
  I2SCamera::init(xres, yres, VSYNC, HREF, XCLK, PCLK, D0, D1, D2, D3, D4, D5,
D6, D7);
}

void OV7670::testImage()
{
  i2c.writeRegister(ADDR, 0x71, 0x35 | 0x80);
}

void OV7670::saturation(int s)  //-2 to 2
{
  //color matrix values
  i2c.writeRegister(ADDR, 0x4f, 0x80 + 0x20 * s);
  i2c.writeRegister(ADDR, 0x50, 0x80 + 0x20 * s);
  i2c.writeRegister(ADDR, 0x51, 0x00);
  i2c.writeRegister(ADDR, 0x52, 0x22 + (0x11 * s) / 2);
  i2c.writeRegister(ADDR, 0x53, 0x5e + (0x2f * s) / 2);
  i2c.writeRegister(ADDR, 0x54, 0x80 + 0x20 * s);
  i2c.writeRegister(ADDR, 0x58, 0x9e);  //matrix signs
}

void OV7670::frameControl(int hStart, int hStop, int vStart, int vStop)
{
  i2c.writeRegister(ADDR, REG_HSTART, hStart >> 3);
  i2c.writeRegister(ADDR, REG_HSTOP,  hStop >> 3);
  i2c.writeRegister(ADDR, REG_HREF, ((hStop & 0b111) << 3) | (hStart & 0b111));

  i2c.writeRegister(ADDR, REG_VSTART, vStart >> 2);
  i2c.writeRegister(ADDR, REG_VSTOP, vStop >> 2);
  i2c.writeRegister(ADDR, REG_VREF, ((vStop & 0b11) << 2) | (vStart & 0b11));
}

void OV7670::QQQVGA()
{
    i2c.writeRegister(ADDR, REG_COM3, 0x04);  //DCW enable
    i2c.writeRegister(ADDR, REG_COM14, 0x1b); //pixel clock divided by 4, manual
scaling enable, DCW and PCLK controlled by register
    i2c.writeRegister(ADDR, REG_SCALING_XSC, 0x3a);
    i2c.writeRegister(ADDR, REG_SCALING_YSC, 0x35);
    i2c.writeRegister(ADDR, REG_SCALING_DCWCTR, 0x33); //downsample by 8
```

```
   i2c.writeRegister(ADDR, REG_SCALING_PCLK_DIV, 0xf3); //pixel clock divided
by 8
   i2c.writeRegister(ADDR, REG_SCALING_PCLK_DELAY, 0x02);
}

void OV7670::QQVGA()
{
 //160x120 (1/4)
 //i2c.writeRegister(ADDR, REG_CLKRC, 0x01);
 i2c.writeRegister(ADDR, REG_COM3, 0x04);  //DCW enable

 i2c.writeRegister(ADDR, REG_COM14, 0x1a); //pixel clock divided by 4, manual
scaling enable, DCW and PCLK controlled by register
 i2c.writeRegister(ADDR, REG_SCALING_XSC, 0x3a);
 i2c.writeRegister(ADDR, REG_SCALING_YSC, 0x35);

 i2c.writeRegister(ADDR, REG_SCALING_DCWCTR, 0x22); //downsample by 4
 i2c.writeRegister(ADDR, REG_SCALING_PCLK_DIV, 0xf2); //pixel clock divided by
4
 i2c.writeRegister(ADDR, REG_SCALING_PCLK_DELAY, 0x02);
}

void OV7670::QQVGARGB565()
{
 i2c.writeRegister(ADDR, REG_COM7, 0b10000000);  //all registers default

 i2c.writeRegister(ADDR, REG_CLKRC, 0b10000000); //double clock
 i2c.writeRegister(ADDR, REG_COM11, 0b1000 | 0b10); //enable auto 50/60Hz detect
+ exposure timing can be less...

 i2c.writeRegister(ADDR, REG_COM7, 0b100); //RGB
 i2c.writeRegister(ADDR, REG_COM15, 0b11000000 | 0b010000); //RGB565

 QQVGA();

 frameControl(196, 52, 8, 488); //no clue why horizontal needs such strange values,
vertical works ok

 //i2c.writeRegister(ADDR, REG_COM10, 0x02); //VSYNC negative
 //i2c.writeRegister(ADDR, REG_MVFP, 0x2b);  //mirror flip

 i2c.writeRegister(ADDR, 0xb0, 0x84);// no clue what this is but it's most important for
colors
 saturation(0);
 i2c.writeRegister(ADDR, 0x13, 0xe7); //AWB on
 i2c.writeRegister(ADDR, 0x6f, 0x9f); // Simple AWB
}

void OV7670::QQQVGARGB565()
{
```

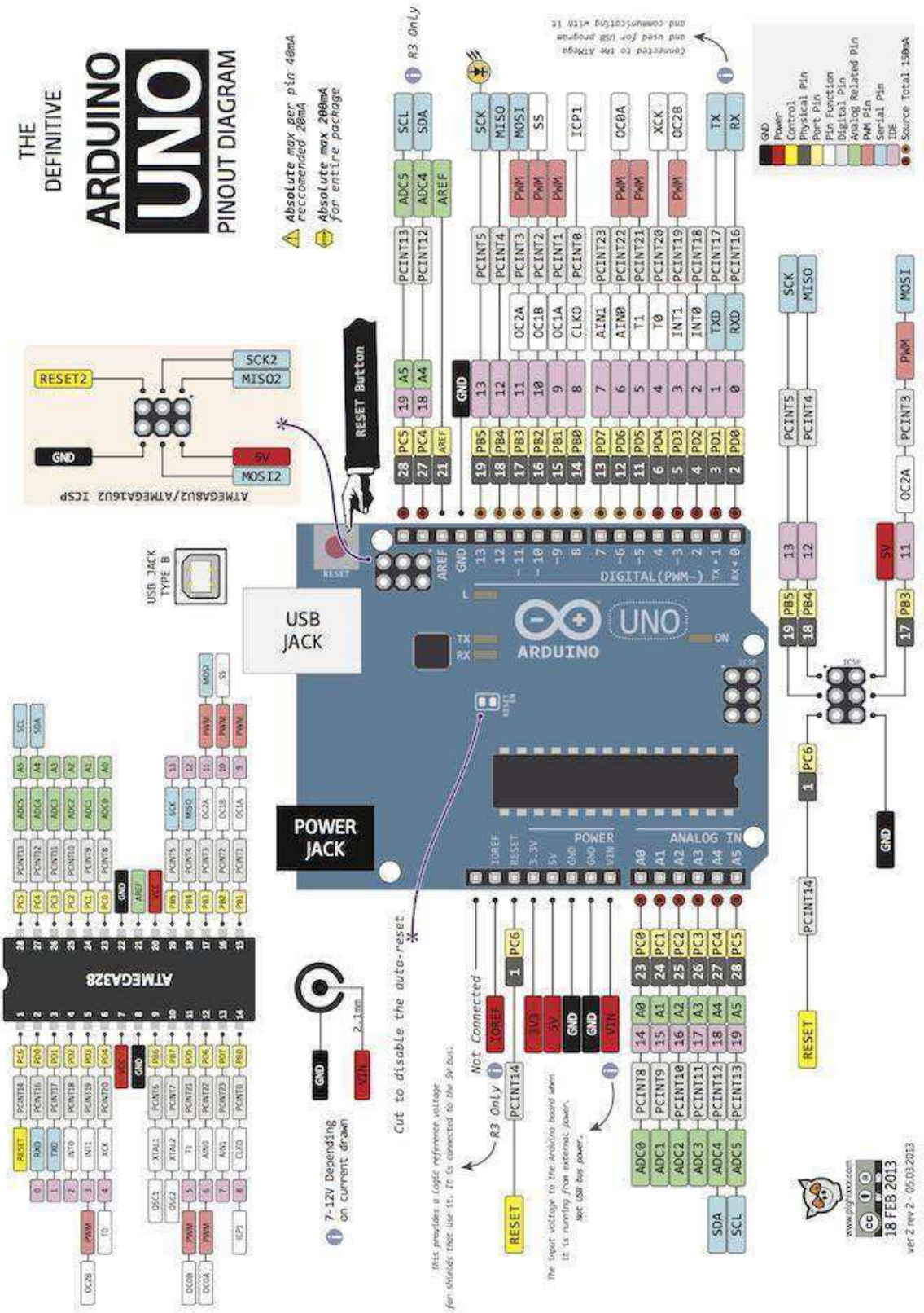
```
  i2c.writeRegister(ADDR, REG_COM7, 0b10000000);  //all registers default

  i2c.writeRegister(ADDR, REG_CLKRC, 0b10000000); //double clock
  i2c.writeRegister(ADDR, REG_COM11, 0b1000 | 0b10); //enable auto 50/60Hz detect
+ exposure timing can be less...

  i2c.writeRegister(ADDR, REG_COM7, 0b100); //RGB
  i2c.writeRegister(ADDR, REG_COM15, 0b11000000 | 0b010000); //RGB565

  QQQVGA();

  frameControl(196, 52, 8, 488); //no clue why horizontal needs such strange values,
vertical works ok

  //i2c.writeRegister(ADDR, REG_MVFP, 0x2b);  //mirror flip

  i2c.writeRegister(ADDR, 0xb0, 0x84);// no clue what this is but it's most important for
colors
  saturation(0);
  i2c.writeRegister(ADDR, 0x13, 0xe7); //AWB on
  i2c.writeRegister(ADDR, 0x6f, 0x9f); // Simple AWB
}

/*OV7670.h*/

#pragma once
#include "I2SCamera.h"
#include "I2C.h"

class OV7670: public I2SCamera
{
  public:
  enum Mode
  {
    QQQVGA_RGB565,
    QQVGA_RGB565,
    QVGA_RGB565,
    VGA_RGB565,
  };
  int xres, yres;

  protected:
  static const int ADDR = 0x42;

  Mode mode;
  I2C i2c;

  void testImage();
  void saturation(int s);
  void frameControl(int hStart, int hStop, int vStart, int vStop);
```

```
  void QQVGA();
  void QQVGARGB565();
  void QQQVGA();
  void QQQVGARGB565();
  void inline writeRegister(unsigned char reg, unsigned char data)
  {
    i2c.writeRegister(ADDR, reg, data);
  }

  public:
  OV7670(OV7670::Mode m, const int SIOD, const int SIOC, const int VSYNC, const
int HREF, const int XCLK, const int PCLK, const int D0, const int D1, const int D2, const
int D3, const int D4, const int D5, const int D6, const int D7);


//camera registers
  static const int REG_GAIN = 0x00;
  static const int REG_BLUE = 0x01;
  static const int REG_RED = 0x02;
  static const int REG_COM1 = 0x04;
  static const int REG_VREF = 0x03;
  static const int REG_COM4 = 0x0d;
  static const int REG_COM5 = 0x0e;
  static const int REG_COM6 = 0x0f;
  static const int REG_AECH = 0x10;
  static const int REG_CLKRC = 0x11;
  static const int REG_COM7 = 0x12;
    static const int COM7_RGB = 0x04;
  static const int REG_COM8 = 0x13;
    static const int COM8_FASTAEC = 0x80;    // Enable fast AGC/AEC
    static const int COM8_AECSTEP = 0x40;    // Unlimited AEC step size
    static const int COM8_BFILT = 0x20;    // Band filter enable
    static const int COM8_AGC = 0x04;    // Auto gain enable
    static const int COM8_AWB = 0x02;    // White balance enable
    static const int COM8_AEC = 0x0;
  static const int REG_COM9 = 0x14;
  static const int REG_COM10 = 0x15;
  static const int REG_COM14 = 0x3E;
  static const int REG_COM11 = 0x3B;
  static const int COM11_NIGHT = 0x80;
  static const int COM11_NMFR = 0x60;
  static const int COM11_HZAUTO = 0x10;
  static const int COM11_50HZ = 0x08;
  static const int COM11_EXP = 0x0;
  static const int REG_TSLB = 0x3A;
  static const int REG_RGB444 = 0x8C;
  static const int REG_COM15 = 0x40;
    static const int COM15_RGB565 = 0x10;
    static const int COM15_R00FF = 0xc0;
  static const int REG_HSTART = 0x17;
```

```
  static const int REG_HSTOP = 0x18;
  static const int REG_HREF = 0x32;
  static const int REG_VSTART = 0x19;
  static const int REG_VSTOP = 0x1A;
  static const int REG_COM3 = 0x0C;
  static const int REG_MVFP = 0x1E;
  static const int REG_COM13 = 0x3d;
    static const int COM13_UVSAT = 0x40;
  static const int REG_SCALING_XSC = 0x70;
  static const int REG_SCALING_YSC = 0x71;
  static const int REG_SCALING_DCWCTR = 0x72;
  static const int REG_SCALING_PCLK_DIV = 0x73;
  static const int REG_SCALING_PCLK_DELAY = 0xa2;
  static const int REG_BD50MAX = 0xa5;
  static const int REG_BD60MAX = 0xab;
  static const int REG_AEW = 0x24;
  static const int REG_AEB = 0x25;
  static const int REG_VPT = 0x26;
  static const int REG_HAECC1 = 0x9f;
  static const int REG_HAECC2 = 0xa0;
  static const int REG_HAECC3 = 0xa6;
  static const int REG_HAECC4 = 0xa7;
  static const int REG_HAECC5 = 0xa8;
  static const int REG_HAECC6 = 0xa9;
  static const int REG_HAECC7 = 0xaa;
  static const int REG_COM12 = 0x3c;
  static const int REG_GFIX = 0x69;
  static const int REG_COM16 = 0x41;
  static const int COM16_AWBGAIN = 0x08;
  static const int REG_EDGE = 0x3f;
  static const int REG_REG76 = 0x76;
  static const int ADCCTR0 = 0x20;

};

/*XClk.cpp*/
#include "XClk.h"
#include "driver/ledc.h"

bool ClockEnable(int pin, int Hz)
{
   periph_module_enable(PERIPH_LEDC_MODULE);

   ledc_timer_config_t timer_conf;
   timer_conf.bit_num = (ledc_timer_bit_t)1;
   timer_conf.freq_hz = Hz;
   timer_conf.speed_mode = LEDC_HIGH_SPEED_MODE;
   timer_conf.timer_num = LEDC_TIMER_0;
   esp_err_t err = ledc_timer_config(&timer_conf);
   if (err != ESP_OK) {
```

```
        return false;
    }

    ledc_channel_config_t ch_conf;
    ch_conf.channel = LEDC_CHANNEL_0;
    ch_conf.timer_sel = LEDC_TIMER_0;
    ch_conf.intr_type = LEDC_INTR_DISABLE;
    ch_conf.duty = 1;
    ch_conf.speed_mode = LEDC_HIGH_SPEED_MODE;
    ch_conf.gpio_num = pin;
    err = ledc_channel_config(&ch_conf);
    if (err != ESP_OK) {
        return false;
    }
    return true;
}

void ClockDisable()
{
    periph_module_disable(PERIPH_LEDC_MODULE);
}

/*XClk.h*/

#pragma once

bool ClockEnable(int pin, int Hz);
void ClockDisable();
```

ANEXO A – MAPA DE PINOS DO ARDUÍNO UNO



ANEXO B – MAPA DE PINOS DA NodeMCU-32S