UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA
EEL7802 - PROJETO EM ELETRÔNICA II

RELATÓRIO II

SISTEMA DE SEGURANÇA ADICIONAL E DE FÁCIL USO PARA ESTABELECIMENTOS COMERCIAIS

CÂMERA TRAP

Autores:
Luciano Silva do Lago
Pedro Henrique Kappler Fornari

Florianópolis, 04 de Novembro de 2015.

1. Introdução

Neste relatório será abordado o progresso obtido no desenvolvimento da *Câmera Trap*, a partir da data de entrega do Relatório I.

No primeiro relatório já havia sido alcançado alguns objetivos e *checkpoints* importantes, como a escolha e definição dos componentes, a modelagem do sistema por meio de diagramas, o estudo do funcionamento do sensor *PIR*, o estudo das ondas provenientes da câmera e de seus protocolos de comunicação, ideias para armazenar a imagem da forma mais rápida possível e, por fim, um breve estudo do módulo WiFi (ESP8266), além de desenvolver o *layout* de uma placa de testes para este módulo.

A segunda etapa do desenvolvimento teve maior ênfase no módulo WiFi e na comunicação serial SCCB com a câmera, visto que na primeira etapa o foco foi mais na câmera, porém viu-se a necessidade também de mudar a maneira de armazenar a imagem, que será descrita em breve.

Além disso, será apresentado os problemas enfrentados durante este período e as justificativas para estes erros, trazendo um cronograma atualizado com atividades já realizadas e as próximas etapas do projeto.

2. Desenvolvimento

2.1 Módulo WiFi ESP-07

Para iniciar os testes do módulo, foi preciso, inicialmente, conseguir um conversor USB-Serial, além de um regulador de tensão de 5V para 3V3, pois o módulo trabalha com tensão diferente caso quisesse-se testar diretamente com a porta USB do computador. Foi-se comprado um conversor e regulador integrado na loja *Brasil Robotics* de Florianópolis e apresentamos ele na imagem a seguir.

Simultaneamente a aquisição destes componentes foi estudada formas de programar o módulo, encontrando três possibilidades: envio de comandos AT, onde deveria fazer uma comunicação serial direta entre um microcontrolador externo e o módulo; programação direta do módulo em LUA, onde seria necessário instalar um interpretador da linguagem LUA no módulo e por meio de uma IDE que será apresentada abaixo seria gravado um código diretamente na memória interno do módulo; ou a programação pela IDE do Arduino, onde seria utilizado a IDE do Arduino em conjunto com as configurações adicionais dos módulos ESP-Generic, gerando um arquivo .hex compatível com o microcontrolador interno do módulo, este processo será melhor explicado posteriormente.

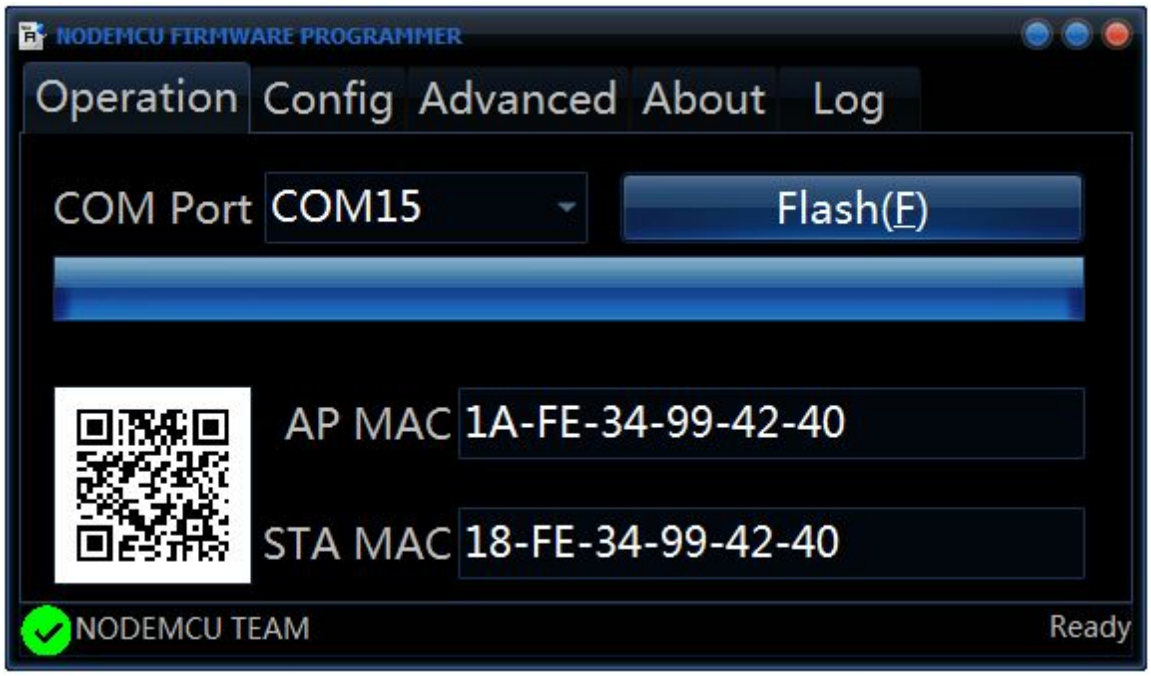
Para decidir qual forma será utilizada, estudou-se as vantagens e desvantagens de cada uma delas, os próximos parágrafos apresentam estas características.

Comandos AT: Possui a vantagem por ser uma forma de rápida prototipação, porém o limite de velocidade estipulado pela serial torna esta forma de projeto ineficaz. A velocidade máxima possível seria de aproximadamente 921600bps.

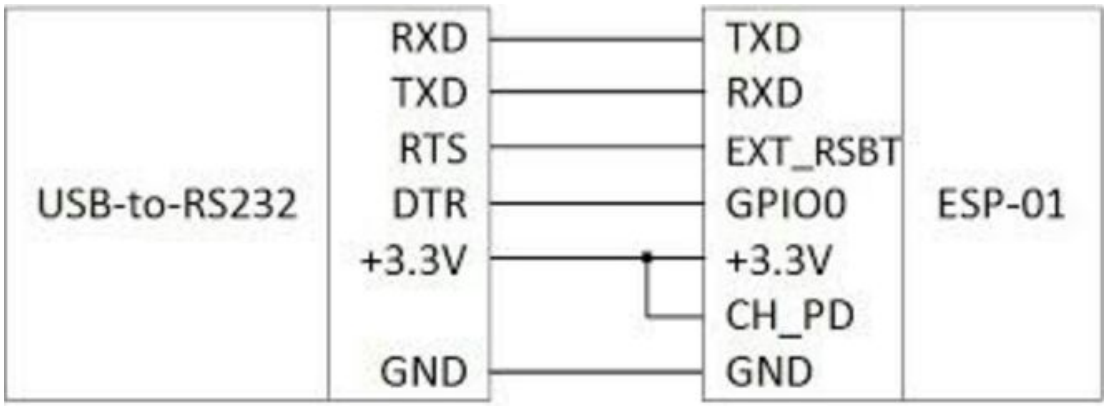
LUA: LUA é uma linguagem interpretada, na qual existe uma considerável quantidade de artigos, publicações online e tutoriais sobre o módulo ESP-Generic, e por isso ela foi considerada por um tempo como a padrão para o projeto.

O interpretador utilizado foi o NODEMCU, que foi utilizado pela empresa Adafruit na confecção de uma *breakout* para o módulo que já se encontra no mercado. Para gravar este interpretador o *link* para o tutorial presente foi utilizado para passo a passo: http://www.whatimade.today/loading-the-nodemcu-firmware-on-the-esp8266-windows-guide/

, onde basicamente é feito o download do *firmware* e da ferramenta e executa-se esta última. O software de gravação está apresentado na figura a seguir:

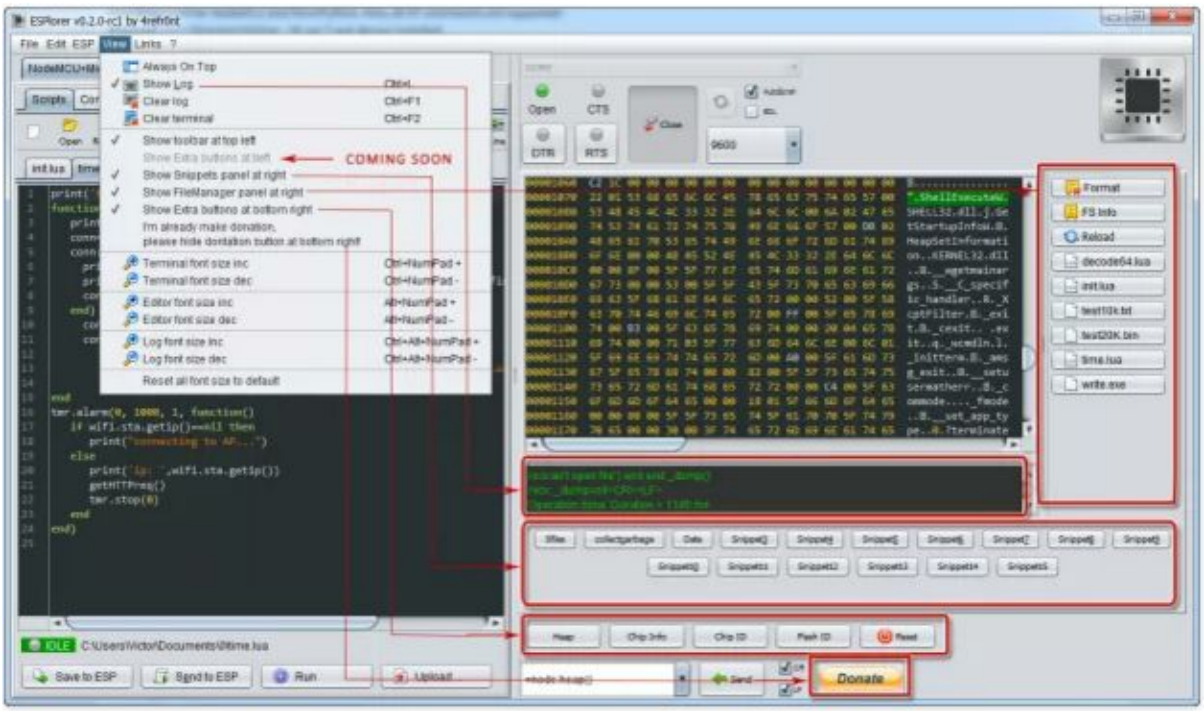


Com isso, a ferramenta é então configurada conforme as configurações no módulo ESP-07, e então este é conectado o módulo ao conversor USB-serial conforme a seguinte tabela:



E então, simplesmente, com um clique em “FLASH(F)”, o interpretador é gravado no módulo ESP-07.

Após isso a interface gráfica ESPlorer é usada para programar e gravar o módulo, a qual pode ser encontrada no site http://esp8266.ru/esplorer/ e, já pelo nome, mostra que é voltada para o módulo ESP, a imagem abaixo mostra a interface gráfica ESPlorer.



Arduino IDE: Existe uma forma de adicionar às placas da IDE do Arduino, as placas do módulo ESP8266, por meio de uma pasta de arquivos desenvolvida por um meio coloborativo. Nesta pasta existe também um tutorial de como integrar os modelos de placas do ESP8266 e o software de desenvolvimento do Arduino.

Para isso basta seguir os passos citados no tutorial, que são: abrir a janela de preferencias da IDE do Arduino; adicionar o link apresentado no tutorial no campo *Additional Board Manager URLs;* abrir a janela *Boards Manager* from *Tools* > *Board* e instalar a plataforma ESP8266. Com isso o módulo pode ser programado em C/C++ a partir da IDE do Arduino.

Após esta análise das possibilidades, foi testada a programação com LUA e a IDE do Arduino, primeiramente fazendo um LED piscar e após isso utilizando um exemplo de criação de rede TCP e enviando um pacote via TCP.

Por fim, o Arduino foi escolhido pela maior familiarização, tanto com C++ quanto com a própria interface.

2.2. Redes TCP Client/Server

Após fazer os primeiros testes com o módulo, um estudo foi iniciado de como funcionam protocolos de comunicação WiFi, como receber estes dados no computador e como devem ser enviados. Como os exemplos são apresentados a partir da comunicação TCP/IP, abaixo segue uma breve explicação do protocolo.

A comunicação TCP/IP funciona basicamente da seguinte forma: quando um dispositivo é conectado à rede, este recebe um IP, o qual serve para identificar este dispositivo. A partir deste IP pode-se abrir portas TCP, que servem para identificar qual serviço você está utilizando na rede. O padrão para TCP é 80, o qual será usado neste projeto, pois o módulo WiFi será configurado como servidor.

No TCP, o cliente envia um pacote “SYN”, solicitando a abertura da conexão, caso a porta estiver fechada, o servidor retorna um pacote “RST” finalizando a comunicação. Caso contrário o servidor retorna outro pacote “SYN”, seguido de um “ACK”, informando a disponibilidade da porta e abrindo a conexão.

Assim que o cliente enviar um pacote “ACK” ao servidor a comunicação se inicia, de forma que o cliente envie até 1500 bytes por pacote, logo deve-se tomar cuidado em relação a isso, de forma a manter uma confiabilidade na informação recebida pelo servidor.

Essa confiabilidade pode ser monitorada por um pacote de 4 bytes adicionais na mensagem, que são um código CRC. Assim que todos os pacotes forem enviados, um ultimo pacote “FYN” é enviado do cliente para o servidor e, posteriormente, do servidor para o cliente, encerrando oficialmente a comunicação.

2.3. Testes de comunicação

A partir do estudo da teoria do protocolo de comunicação TCP/IP e do exemplo encontrado sobre este protocolo para o módulo, foram feitos alguns testes para verificar se a taxa de transmissão de pacotes satisfaria nossas necessidades. Inicialmente foram feitos os testes com o interpretador da linguagem LUA e posteriormente carregando o código pelo Arduino.

Foi escrito um programa em C# e outro em Python para receber os dados e calcular a velocidade da comunicação. No programa carregado no módulo, inicialmente em LUA, enviamos um pacote de dados com tamanho e strings conhecidas, para poder calcular a velocidade dividindo o tamanho pelo tempo que o recebimento de dados terminou. Considerando uma rede conectada no modo n, uma taxa de transmissão próxima de 40Mbits/s é válida. Não foi possível realizar os testes de velocidade ainda por falta de infraestrutura para fazer o mesmo.

2.4. Módulo para armazenar a imagem

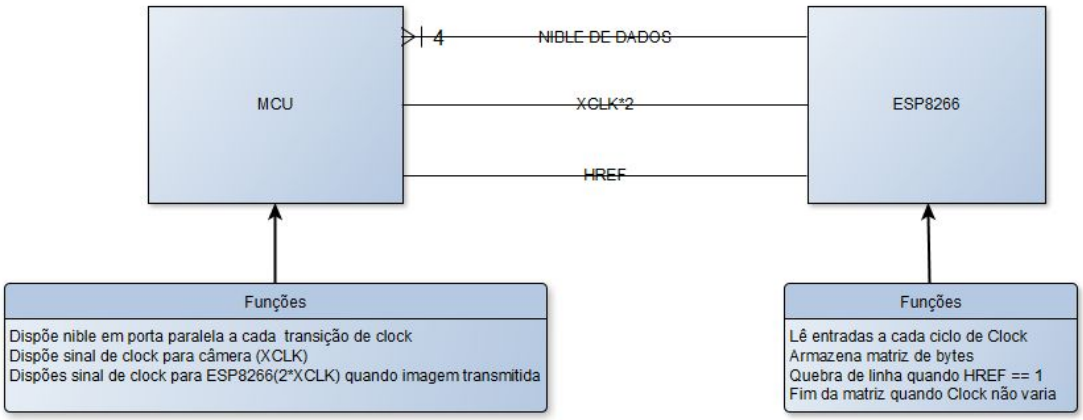
Visto que o módulo EPS-07 possui uma memória interna relativamente grande de memória RAM, aproximadamente 34kB, o projeto irá se distanciar do planejado no Relatório I. Será feita uma comunicação paralela de bits, ficando muito semelhando à comunicação usada para comunicar-se com *displays* LCD 16x2.

Além disso não necessários sinais para informar uma quebra de linha e um final de imagem, que será feito da seguinte forma: a primeira opção é fornecer o (XCLK*2) quando houver um (VSYNC) no microcontrolador externo, indicando o inicio da imagem, e a quebra de linha pelo por outro GPIO, de forma que, após um determinado número de quebras de linha, contado pelo controlador do módulo, pode-se inferir que a imagem está no fim; a segunda opção inicia a leitura da mesma forma da primeira opção, porém indica o termino da imagem pela estabilização do sinal enviado pelo microcontrolador em nível lógico alto ou nível lógico baixo.

Por fim, deve-se desenvolver um código para o microcontrolador externo que seja capaz de colocar 4 bits de dados da câmera por vez em pinos de saída e gerar um clock duas vezes maior do que o clock XCLK.

Isto deve-se por causa da limitação da velocidade que uma memória flash possui para gravação e leitura de dados, podendo atingir um máximo de 20 Mbits/s, o que torna

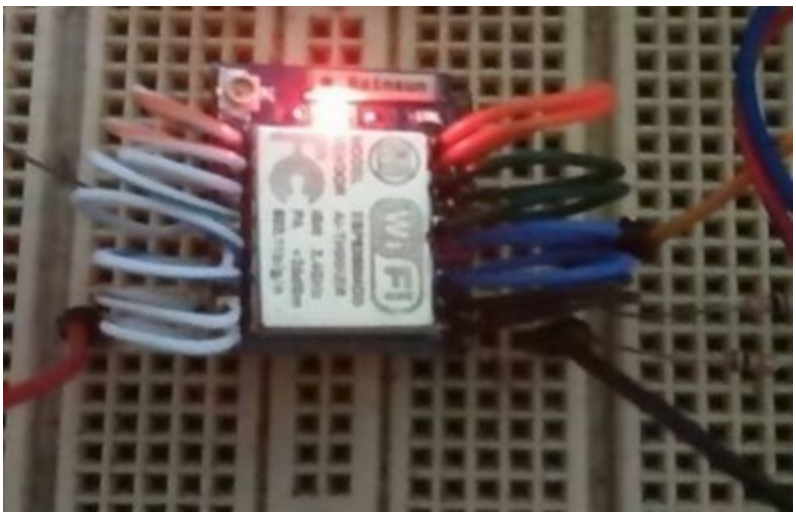
esta forma de salvar os dados é inviável. A imagem a seguir ilustra melhor a linha de raciocínio que será seguida.



2.5 Problemas com o módulo ESP-07

O plano que constava no Relatório I era o de utilizar uma placa de testes, *breakout board*, para conectar o módulo em uma matriz de contatos. A matriz de contatos possui espaçamentos de 100mils já o módulo ESP-07 possui espaçamentos de 1mm.

A placa seria produzida pelos integrantes do grupo, e que o *layout* da mesma foi até anexada no Relatório I. Mas devido a dificuldade de conseguir fresadora e a dificuldade de se fazer por transferência de calor (distância curta entre trilhas e pads e a alta umidade do ar) foi feito um “circuito aranha” para fazer essa adaptação de conectores.



2.6. Configuração da Câmera

A câmera OV7670 utiliza um protocolo de comunicação própria para a configuração da câmera. Este protocolo é muito semelhante ao protocolo I²C da NXP, tanto que pode-se utilizar I²C para configurar a câmera. Mas para este projeto, como pretende-se usar os hardwares configuráveis do microcontrolador para outras finalidades, optou-se por implementar o protocolo SCCB via software.

O protocolo SCCB é síncrono e com dados de 9 bits para leitura e escrita. Sendo 8 de dados e 1 “Don’t Care” para escrita ou “NA” para leitura.

Para a escrita é necessário passar por 3 fases. Enviar endereço do dispositivo (no caso da câmera OV7670, 0x42), enviar endereço do registrador a ser gravado e por último enviar o dado a ser gravado.

Para a leitura é necessário fazer uma escrita em 2 fases e depois uma leitura de 2 fases. Para a escrita deve-se enviar o endereço do dispositivo e depois o endereço do registrador a ser lido. Para a leitura deve-se enviar o endereço do dispositivo a ser lido e depois ler os bits enviados pelo escravo.

O documento com todas as especificações do protocolo e forma de onda pode ser encontrado no seguinte endereço:

http://www.ovt.com/download_document.php?type=document&DID=63.

Abaixo segue a implementação para o PSoC 4 da Cypress usando o framework disponibilizada pela mesma. Em seguida segue a leitura realizada de todos os registradores da câmera.

sccb.h

```
#ifndef SCCB_H
#define SCCB_H

#include <stdint.h>

void SCCB_Write(uint8_t addr, uint8_t reg, uint8_t data);
uint8_t SCCB_Read(uint8_t addr, uint8_t reg);
```

```
void SCCB_CleanUp(void);
int SCCB_Start(void);

#endif /* SCCB_H */
```

sccb.c

```
#include <project.h>
#include "iprintf.h"
#include "sccb.h"

static void SCCB_StartTransmission(void){
       OV_SDIOD_SetDriveMode(OV_SDIOD_DM_STRONG);

       OV_SDIOD_Write(0);
       CyDelayUs(10);
}

static void SCCB_WriteByte(uint8_t data){
       int i;

       OV_SDIOD_SetDriveMode(OV_SDIOD_DM_STRONG);

       for (i = 7; i >= 0; i--){
              OV_SDIOC_Write(0);
              CyDelayUs(5);

        OV_SDIOD_Write((data >> i) & 0x01);
              OV_SDIOC_Write(1);
              CyDelayUs(5);
    }

       OV_SDIOC_Write(0);
       OV_SDIOD_Write(1);
       OV_SDIOD_SetDriveMode(OV_SDIOD_DM_DIG_HIZ);
       CyDelayUs(5);

       OV_SDIOC_Write(1);
       CyDelayUs(5);
}

static uint8_t SCCB_ReadByte(void){
       int i;
       uint8_t data;

       OV_SDIOD_SetDriveMode(OV_SDIOD_DM_DIG_HIZ);

       OV_SDIOC_Write(0);
    CyDelayUs(5);

       data = 0;
       for (i = 7; i >= 0; i--){
              OV_SDIOC_Write(1);
              CyDelayUs(5);
```

```
            data = data << 1;
        data &= ~0x01;
            data |= OV_SDIOD_Read();

            OV_SDIOC_Write(0);
            CyDelayUs(5);
      }

      OV_SDIOD_SetDriveMode(OV_SDIOD_DM_STRONG);
      OV_SDIOD_Write(1);
      CyDelayUs(5);

      OV_SDIOC_Write(1);
      CyDelayUs(5);

      return data;
}

static void SCCB_EndTransmission(void){
      OV_SDIOC_Write(0);
      CyDelayUs(10);

      OV_SDIOD_SetDriveMode(OV_SDIOD_DM_STRONG);
      OV_SDIOD_Write(0);
      CyDelayUs(90);

      OV_SDIOC_Write(1);
      CyDelayUs(10);

      OV_SDIOD_Write(1);
      CyDelayUs(90);

      OV_SDIOD_SetDriveMode(OV_SDIOD_DM_DIG_HIZ);
}

void SCCB_Write(uint8_t addr, uint8_t reg, uint8_t data){
      SCCB_StartTransmission();
      SCCB_WriteByte(addr);
      SCCB_WriteByte(reg);
      SCCB_WriteByte(data);
      SCCB_EndTransmission();
}

uint8_t SCCB_Read(uint8_t addr, uint8_t reg){
      uint8_t data;

      SCCB_StartTransmission();
      SCCB_WriteByte(addr);
      SCCB_WriteByte(reg);
      SCCB_EndTransmission();

      SCCB_StartTransmission();
      SCCB_WriteByte(addr | 0x01);
      data = SCCB_ReadByte();
      SCCB_EndTransmission();

      return data;
```

```
}

int SCCB_Start(void){
    OV_SDIOC_SetDriveMode(OV_SDIOC_DM_STRONG);
       OV_SDIOD_SetDriveMode(OV_SDIOD_DM_STRONG);

       OV_SDIOC_Write(1);
       OV_SDIOD_Write(1);
       CyDelayUs(10);

       return 0;
}

void SCCB_CleanUp(void){
    OV_SDIOC_SetDriveMode(OV_SDIOC_DM_DIG_HIZ);
       OV_SDIOD_SetDriveMode(OV_SDIOD_DM_DIG_HIZ);
}
```

main.c

```
#include <project.h>
#include <stdint.h>
#include "iprintf.h"
#include "sccb.h"

int main(){
    CyGlobalIntEnable;

    DEBUG_Start();
    SCCB_Start();

    int i = 0;
    for(i = 0x00; i < 0xca; i++){
        iprintf("ADDR[%x] <= \'%x\'\n\r",
            i,
            SCCB_Read(0x42, i));
    }

    while(1){
    }
}
```

saída na serial

```
ADDR[00] <= '00'
ADDR[01] <= '80'
ADDR[02] <= '80'
ADDR[03] <= '00'
ADDR[04] <= '00'
ADDR[05] <= '00'
ADDR[06] <= 'ff'
ADDR[07] <= '00'
ADDR[08] <= '00'
ADDR[09] <= '01'
ADDR[0a] <= '76'
ADDR[0b] <= '73'
```

```
ADDR[65] <= '30'
ADDR[66] <= '00'
ADDR[67] <= '80'
ADDR[68] <= '00'
ADDR[69] <= '00'
ADDR[6a] <= '80'
ADDR[6b] <= '0a'
ADDR[6c] <= '02'
ADDR[6d] <= '55'
ADDR[6e] <= 'c0'
ADDR[6f] <= '9a'
ADDR[70] <= '3a'
```

```
ADDR[0c] <= '00'
ADDR[0d] <= '00'
ADDR[0e] <= '01'
ADDR[0f] <= '43'
ADDR[10] <= 'ff'
ADDR[11] <= '80'
ADDR[12] <= '00'
ADDR[13] <= '8f'
ADDR[14] <= '4a'
ADDR[15] <= '00'
ADDR[16] <= '00'
ADDR[17] <= '11'
ADDR[18] <= '61'
ADDR[19] <= '03'
ADDR[1a] <= 'ff'
ADDR[1b] <= '00'
ADDR[1c] <= '7f'
ADDR[1d] <= 'a2'
ADDR[1e] <= '01'
ADDR[1f] <= '00'
ADDR[20] <= '04'
ADDR[21] <= '02'
ADDR[22] <= '01'
ADDR[23] <= '00'
ADDR[24] <= 'ff'
ADDR[25] <= '63'
ADDR[26] <= 'd4'
ADDR[27] <= '80'
ADDR[28] <= '80'
ADDR[29] <= '07'
ADDR[2a] <= '00'
ADDR[2b] <= '00'
ADDR[2c] <= '80'
ADDR[2d] <= '0f'
ADDR[2e] <= '00'
ADDR[2f] <= '00'
ADDR[30] <= '08'
ADDR[31] <= '30'
ADDR[32] <= '80'
ADDR[33] <= '08'
ADDR[34] <= '11'
ADDR[35] <= '1a'
ADDR[36] <= '00'
ADDR[37] <= 'ff'
ADDR[38] <= '01'
ADDR[39] <= '00'
ADDR[3a] <= '0d'
ADDR[3b] <= '00'
ADDR[3c] <= '68'
ADDR[3d] <= '88'
ADDR[3e] <= '00'
ADDR[3f] <= '00'
ADDR[40] <= 'c0'
ADDR[41] <= 'ff'
ADDR[42] <= '00'
ADDR[43] <= '14'
ADDR[44] <= 'f0'
```

```
ADDR[71] <= 'ff'
ADDR[72] <= '11'
ADDR[73] <= '00'
ADDR[74] <= '00'
ADDR[75] <= '0f'
ADDR[76] <= '01'
ADDR[77] <= '10'
ADDR[78] <= '00'
ADDR[79] <= '00'
ADDR[7a] <= '24'
ADDR[7b] <= 'ff'
ADDR[7c] <= '07'
ADDR[7d] <= '10'
ADDR[7e] <= '28'
ADDR[7f] <= '36'
ADDR[80] <= '44'
ADDR[81] <= '52'
ADDR[82] <= '60'
ADDR[83] <= '6c'
ADDR[84] <= '78'
ADDR[85] <= 'ff'
ADDR[86] <= '9e'
ADDR[87] <= 'bb'
ADDR[88] <= 'd2'
ADDR[89] <= 'e5'
ADDR[8a] <= '00'
ADDR[8b] <= '00'
ADDR[8c] <= '00'
ADDR[8d] <= '0f'
ADDR[8e] <= '00'
ADDR[8f] <= '00'
ADDR[90] <= '00'
ADDR[91] <= '00'
ADDR[92] <= '00'
ADDR[93] <= '00'
ADDR[94] <= '50'
ADDR[95] <= '50'
ADDR[96] <= '01'
ADDR[97] <= '01'
ADDR[98] <= 'ff'
ADDR[99] <= '40'
ADDR[9a] <= '40'
ADDR[9b] <= '20'
ADDR[9c] <= '00'
ADDR[9d] <= '99'
ADDR[9e] <= '7f'
ADDR[9f] <= 'c0'
ADDR[a0] <= '90'
ADDR[a1] <= '03'
ADDR[a2] <= 'ff'
ADDR[a3] <= '00'
ADDR[a4] <= '00'
ADDR[a5] <= '0f'
ADDR[a6] <= 'f0'
ADDR[a7] <= 'c1'
ADDR[a8] <= 'f0'
ADDR[a9] <= 'c1'
```

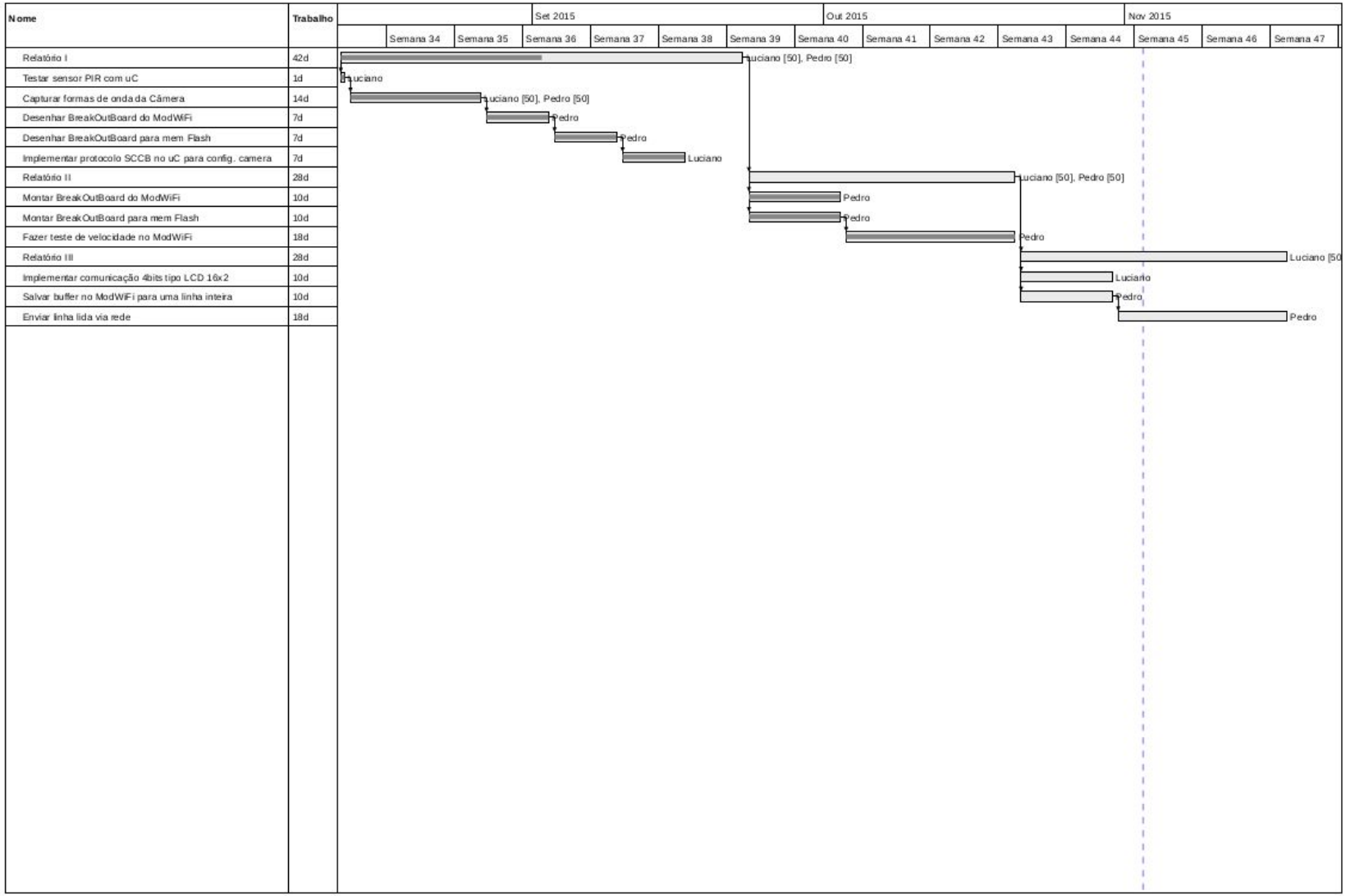
```
ADDR[45] <= '45'
ADDR[46] <= '61'
ADDR[47] <= '51'
ADDR[48] <= '79'
ADDR[49] <= '00'
ADDR[4a] <= 'ff'
ADDR[4b] <= '00'
ADDR[4c] <= '00'
ADDR[4d] <= '04'
ADDR[4e] <= '00'
ADDR[4f] <= '40'
ADDR[50] <= '34'
ADDR[51] <= '0c'
ADDR[52] <= '17'
ADDR[53] <= '29'
ADDR[54] <= 'ff'
ADDR[55] <= '00'
ADDR[56] <= '40'
ADDR[57] <= '80'
ADDR[58] <= '1e'
ADDR[59] <= '91'
ADDR[5a] <= '94'
ADDR[5b] <= 'aa'
ADDR[5c] <= '71'
ADDR[5d] <= '8d'
ADDR[5e] <= 'ff'
ADDR[5f] <= 'f0'
ADDR[60] <= 'f0'
ADDR[61] <= 'f0'
ADDR[62] <= '00'
ADDR[63] <= '00'
ADDR[64] <= '50'
```

```
ADDR[aa] <= '14'
ADDR[ab] <= '0f'
ADDR[ac] <= '00'
ADDR[ad] <= '80'
ADDR[ae] <= '80'
ADDR[af] <= '80'
ADDR[b0] <= '00'
ADDR[b1] <= '00'
ADDR[b2] <= '00'
ADDR[b3] <= '80'
ADDR[b4] <= '00'
ADDR[b5] <= 'ff'
ADDR[b6] <= '00'
ADDR[b7] <= '66'
ADDR[b8] <= '00'
ADDR[b9] <= '06'
ADDR[ba] <= '00'
ADDR[bb] <= '00'
ADDR[bc] <= '00'
ADDR[bd] <= '00'
ADDR[be] <= '00'
ADDR[bf] <= 'ff'
ADDR[c0] <= '00'
ADDR[c1] <= '00'
ADDR[c2] <= '00'
ADDR[c3] <= '00'
ADDR[c4] <= '00'
ADDR[c5] <= '00'
ADDR[c6] <= '00'
ADDR[c7] <= '00'
ADDR[c8] <= '06'
ADDR[c9] <= 'ff'
```

Alguns valores padrões lidos diferem do encontrado no *datasheet* do produto, mas grande parte dos valores são os mesmos encontrados na folha de dados, pode-se dizer que a implementação feita é bastante confiável.

3. Cronograma

As mudanças que precisaram ser feitas no desenvolvimento do projeto podem ser encontradas no cronograma atualizado abaixo. Foram retiradas as tarefas que tinham qualquer relação com a memória flash e foram adicionadas tarefas para se fazer a lógica de aquisição de dados por parte do módulo WiFi do microcontrolador externo.


Nome
Trabalho
Set 2015
Out 2015
Nov 2015
Semana 34
Semana 35
Semana 36
Semana 37
Semana 38
Semana 39
Semana 40
Semana 41
Semana 42
Semana 43
Semana 44
Semana 45
Semana 46
Semana 47
Relatório I
42d
Testar sensor PIR com uC
1d
Capturar formas de onda da Câmera
14d
Desenhar BreakOutBoard do ModWiFi
7d
Desenhar BreakOutBoard para mem Flash
7d
Implementar protocolo SCCB no uC para config. camera
7d
Relatório II
28d
Montar BreakOutBoard do ModWiFi
10d
Montar BreakOutBoard para mem Flash
10d
Fazer teste de velocidade no ModWiFi
18d
Relatório III
28d
Implementar comunicação 4bits tipo LCD 16x2
10d
Salvar buffer no ModWiFi para uma linha inteira
10d
Enviar linha lida via rede
18d
Luciano [50], Pedro [50]
Luciano
Luciano [50], Pedro [50]
Pedro
Pedro
Luciano
Luciano [50], Pedro [50]
Pedro
Pedro
Pedro
Luciano [50
Luciano
Pedro
Pedro

4. Conclusão

4.1. Objetivos Alcançados

Durante esta segunda etapa consegui-se avançar bastante com o módulo WiFi, desenvolvemos a rede TCP/IP, que será primordial para transmitir a imagem para o proprietário do estabelecimento comercial pela internet.

Conseguiu-se também verificar e iniciar o desenvolvimento de um *firmware* que excluirá a necessidade de uma memória externa para o microcontrolador, utilizando a própria memória do módulo ESP8266, que até agora foi a melhor solução encontrada para um armazenamento rápido e de qualidade da imagem da câmera.

Por fim ainda foi possível estudar a técnica de *bit-banging* para configurar a câmera como desejado, o que é mais fácil do que a configuração pelo protocolo I2C, que exigiria mais hardware.

4.2. Problemas e Dificuldades Encontradas

O maior desafio nessa etapa foi tentar manter o cronograma com a necessidade de desenvolver a *breakout* a mão, pois os testes do módulo WiFi utilizando fios era muito ruim, mas o processo de soldagem de fios no módulo foi relativamente complicado, principalmente pelo pequeno tamanho do módulo e a quantidade de fios que soltavam durante este processo.

Fora isso, houve problema com o desenvolvimento do teste de velocidade de transmissão WiFi do módulo, no programa de recepção dos dados em C#.

4.3 Conclusão Final

Pode-se observar que esta segunda etapa foi bastante produtiva, pois foi estudado muitos conceitos novos de comunicação WiFi, redes, protocolos de comunicação, etc.

Houve a chance de estar em contato com várias linguagens que não era de conhecimento de ambos, trazendo um conhecimento importante na área de software, como Python, C# e eLua.

Foi descoberto formas de incluir interpretadores a um microcontrolador, em especial o microcontrolador embutido no módulo ESP8266, que podem ser chave para um rápido desenvolvimento em projetos futuros, pois, para projetos com poucas I/O's o microcontrolador embutido no ESP pode ser muito útil, visto que tem grande memória interna e é de baixo custo.

Foi estudado técnicas de comunicação serial que utilizam software ao invés de hardware dedicado para facilitar a configuração de equipamentos quando não há um hardware embarcado para a comunicação desejada, o que também pode ser levado como um grande conhecimento para projetos futuros.

Também é bom observar que, em sua maioria, os estudos desta etapa do desenvolvimento do projeto não são ementa das matérias lecionadas no nosso curso,

porém são fundamentais no mercado de trabalho atual, que está, cada vez mais, investindo em IoT e em equipamentos conectados.

Finalmente, nossos próximos passos estão focados em integrar a comunicação do microcontrolador com o módulo WiFi, de forma que a imagem seja armazenada de forma rápida e enviada primeiramente para um documento ou até mesmo o endereço IP do computador sendo testado, em formato hexadecimal. Também é desejado enviar uma mensagem para um e-mail com informações do estado do sistema, como forma alternativa de aviso ao usuário caso o aplicativo que recebe e converte a imagem não estiver pronto até o final do semestre.

Caso essas metas forem cumpridas até a data de entrega do projeto passaremos a desenvolver o aplicativo de recebimento e conversão das imagens, para que o usuário possa receber uma imagem da loja, caso a câmera trap detecte algum movimento no interior do estabelecimento, de qualquer lugar do mundo, contanto que esteja com um celular conectado à internet e ao aplicativo do sistema.