

UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
Departamento de Engenharia Elétrica - DEE



Igor Radichi, 2018020441
Nander Carmo, 2018019931

Sistema Embarcado para Medição de Luminância

Trabalho Prático

Módulo 1

Belo Horizonte
2021

Sumário

| | |
|---|-----------|
| Introdução | 3 |
| Desenvolvimento | 4 |
| Sensor de Iluminância | 4 |
| Condicionamento de Sinais | 8 |
| Sistema de Digitalização | 10 |
| Sistema de Controle e Interface de Dados | 15 |
| Apresentação e Análise de Resultados | 17 |
| Conclusão | 17 |
| Bibliografia | 17 |

1. Introdução

Os fotodiodos são dispositivos semicondutores capazes de realizar a transdutância de irradiância para corrente elétrica. A corrente é gerada pela absorção de fótons com energia suficiente para criar um par elétron-lacuna na camada de depleção do fotodiodo. As lacunas fluem para o ânodo, enquanto os elétrons fluem para o catodo. Essa movimentação elétrons-lacuna devido à incidência de fótons, caracteriza a fotocorrente.

Esses dispositivos podem funcionar em dois tipos básicos de configuração:

- a) **modo fotovoltaico:** esse tipo de configuração explora o efeito fotovoltaico e não requer polarização externa. Com o circuito aberto ou sob uma impedância de carga (Figura 1-a), gera uma tensão que polariza diretamente o fotodiodo (ânodo positivo em relação ao catodo). Já com o circuito curto-circuitado ou com uma impedância de carga extremamente baixa (Figura 1-b), uma corrente direta irá surgir proporcionalmente ao nível de luz (fotocorrente).

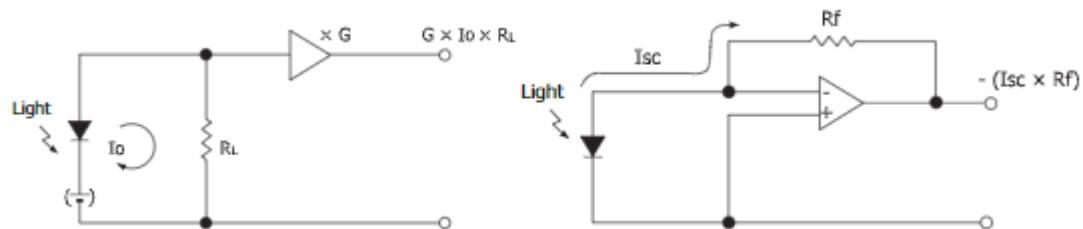


Figura 1 - Operação do fotodiodo em modo fotovoltaico, com tensão gerada por impedância de carga (1-a) e por curto-circuito virtual (1-b)

À esquerda, vemos um circuito que gera uma tensão proporcional à fotocorrente a um fator R_L da resistência de carga $V = I_o \cdot R_L$. Esse valor de tensão é então amplificado conforme desejado. À direita, vemos um circuito que tem o princípio do curto-circuito virtual gerado pelo amp. op., que possibilita um valor R_f muitas vezes menor que R_L e uma consequente maior linearidade, além de otimizações em relação a efeito de carga.

- b) **modo fotocondutor:** esse tipo de configuração requer uma polarização reversa do fotodiodo, que irá reduzir a largura da camada de depleção e possibilitar uma resposta de corrente mais linear e veloz quando comparada ao modo fotovoltaico. Esse método traz, por outro lado, maiores efeitos de corrente escura e de ruídos.

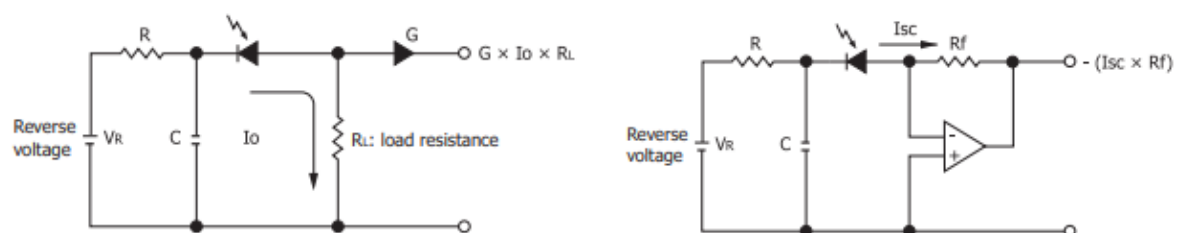


Figura 2 - Operação do fotodiodo em modo fotocondutor, com tensão gerado por impedância de carga (2-a) e curto-circuito virtual (2-b)

À esquerda, uma tensão reversa aplicada (com filtro passa baixas para reduzir os ruídos da fonte) com uma resistência de carga para converter corrente em tensão: $V = I_o \cdot R_L$. À direita, é utilizado um amplificador operacional que possibilita um curto-circuito virtual, reduz o efeito de carga e a influência de ruídos através da resistência, além de maior linearidade.

Uma vez escolhido o modo de operação do dispositivo e a maneira de gerar uma tensão proporcional à intensidade luminosa, pode-se adquirir esses dados através de um circuito condicionado, tratá-los e apresentá-los em uma interface. A seguir, serão apresentados as etapas de construção do sensor de iluminância, desde a montagem de circuito do sensor até a visualização de dados em sua interface.

2. Desenvolvimento

2.1. Sensor de Iluminância

O sensor de iluminância será construído a partir de um fotodiodo BPW34 funcionando em modo fotocondutor, conforme Figura 3:

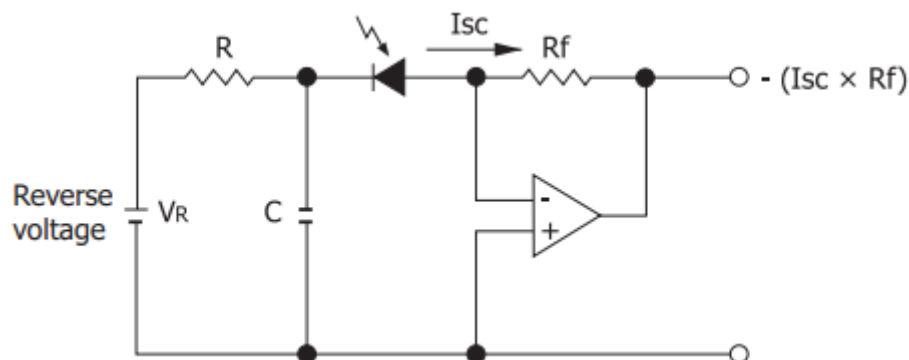


Figura 3 - Circuito com fotodiodo em operação fotocondutora

Com esse circuito, utiliza-se de uma tensão reversa no terminal catodo do fotodiodo, de tal forma a polarizar e deslocar sua curva de operação do quarto (modo fotovoltaico) para o terceiro quadrante (modo fotocondutor), considerando as curvas que definem a relação corrente e tensão do componente, conforme Figura 4. Isso será vantajoso em relação a operação no quarto quadrante pois localiza a operação do fotodiodo em uma região mais longa e de maior linearidade tensão-corrente.

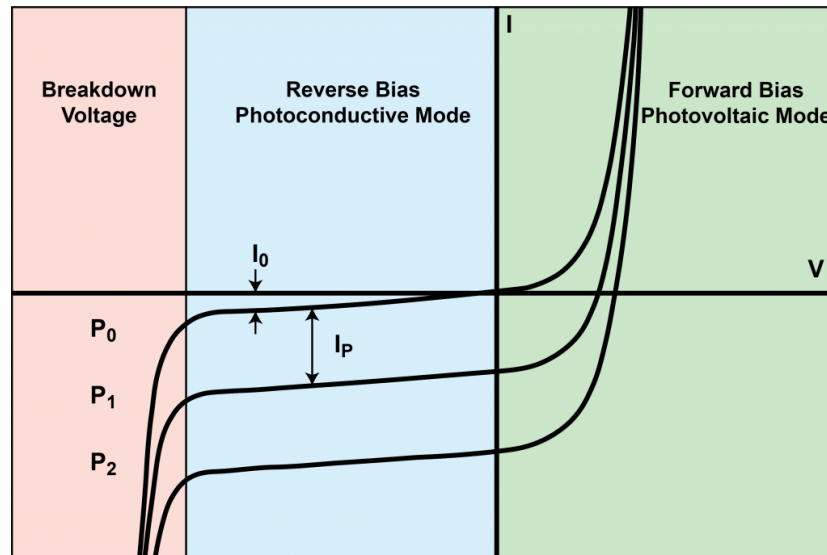


Figura 4 - Regiões de operação do fotodiodo

Essa fonte de tensão reversa não poderá ultrapassar a tensão de ruptura especificada pelo *datasheet* do fotodiodo utilizado, que é de 60V reversos. Para isso, utiliza-se de uma tensão $V_R = 5V$, que será obtida através da saída de tensão regulada do próprio microcontrolador Arduino.

A fonte reversa V_R passará por um filtro passa-baixas RC simples, de tal forma a filtrar ruídos de alta frequência que possa estar contido no sinal da fonte. Poderemos utilizar um resistor $R = 10k\Omega$ e um capacitor $C = 100nF$, que resultarão em uma frequência de corte satisfatória para os propósitos da montagem, cerca de 159Hz.

A saída do filtro é fornecida para o fotodiodo, agora polarizado, e é então transmitida para um amplificador operacional em configuração inversora, com um resistor de realimentação R_f . Essa configuração torna-se interessante pois a utilização do amplificador fornece:

- i) maior linearidade de operação ao circuito pois, devido à $Z_{in} \rightarrow \infty$, é criado um curto-circuito virtual no fotodiodo;
- ii) isolamento entre seu primário e secundário, também devido à $Z_{in} \rightarrow \infty$, que minimiza o efeito de carga provocado pelo sistema de medição;
- iii) ganho intrínseco considerável $A_{vo} \gg 1$.

Com o fotodiodo em curto-circuito, circula-se uma corrente de curto-circuito I_{SC} fornecida pelo *datasheet* do BPW34, que é em torno de $70\mu A$ para $1000lx$ incidindo sobre sua área ativa. Como vamos estimar uma tensão do ADC de 0 a 5V, estima-se de um ganho de transcondutância máximo $G_{max} = -I_{SC} \cdot R_f$ em torno de -3.5V, por segurança e devido à

tensão de saturação precoce dos amp. ops. reais, que provocaria um comportamento não linear do circuito. Para tanto, o valor dimensionado R_f será:

$$- 3.5V = - 70\mu A \cdot R_f \rightarrow R_f = 50k\Omega$$

O valor de R_f estimado poderá ser obtido por uma rede T de resistores de tal forma a alcançar o valor de resistência desejado e ao mesmo tempo minimizar a incidência de ruídos. Essa rede será composta por dois resistores $R_T = 10k\Omega$ e um resistor $R_G = 3.3k\Omega$, que irão produzir uma resistência de realimentação bem próxima da desejada:

$$R_f = R_T \cdot (2 + R_T/R_G) = 50.003k \approx 50k\Omega$$

Como obviamente não é permitido fornecer uma tensão negativa para as entradas do ADC do microcontrolador, esse sinal de tensão gerado foi passado por um outro amp. op., também em configuração inversora, mas de ganho unitário, de forma a rebater essa tensão para o polo positivo.

A alimentação dos amp. ops. será realizada com +5V e -3V, onde a tensão positiva é obtida do próprio microcontrolador, enquanto a tensão negativa será obtida através de uma configuração com 2 pilhas de 1.5V conectadas conforme a Figura 5. Essa alimentação assimétrica é suportada pelo TL071, que permite uma diferença de potencial (assimétrica ou simétrica) de até 42V.

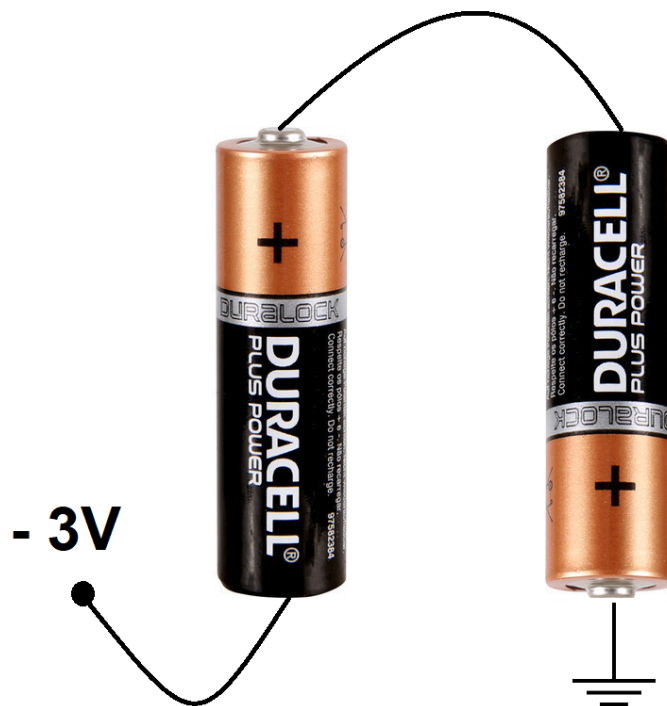


Figura 5 - Esquema para obtenção de tensão negativa

Por fim, teremos o seguinte circuito sensor de iluminância (estágio de entrada):

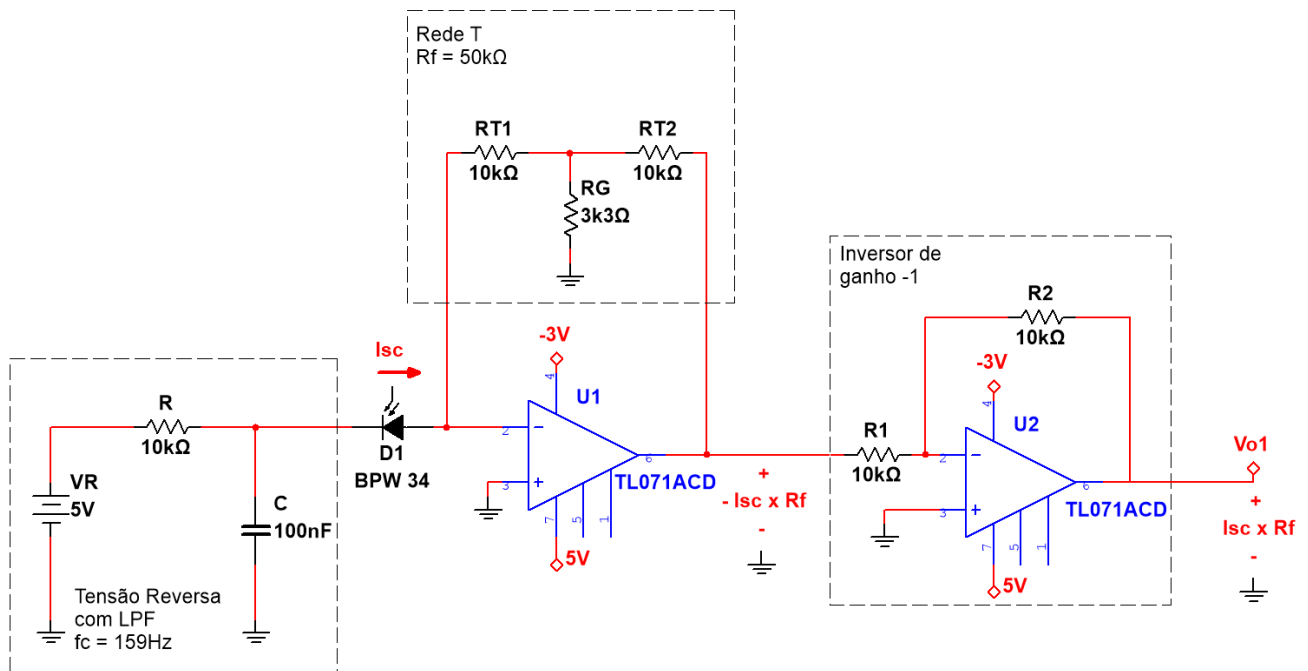


Figura 6 - Esquemático do estágio de entrada - sensor de iluminância

2.2. Condicionamento de Sinais

O condicionamento de sinais para o circuito montado precede o ADC e deve levar em consideração os seguintes aspectos:

- i) efeito de carga: já tratado pelos amplificadores operacionais utilizados no circuito de entrada, devido à alta impedância de entrada, o que minimiza os efeitos sistemáticos provocados por esse efeito de circuitos de medição no ADC.
- ii) efeitos de modo comum: já tratados, uma vez que a medição já é referenciada ao terra, fornecido pelo próprio microcontrolador.
- iii) efeitos de *aliasing*: deve-se inserir um filtro *anti-aliasing* (AAF) antes do amostrador do ADC de tal forma a respeitar o Teorema de Amostragem de Nyquist-Shannon.

O AAF será utilizado para restringir a largura de banda do sinal a ser amostrado, para que seja possível uma amostragem íntegra do sinal a ser medido. Isso poderá ser obtido através de um filtro passa-baixas ativo de 3ª ordem (Bessel), com frequência de corte em torno de 1kHz e ganho unitário. A frequência de corte foi definida considerando-se a frequência de amostragem do microcontrolador de 9615.38Hz. O filtro anti-aliasing será projetado, portanto, da seguinte maneira:

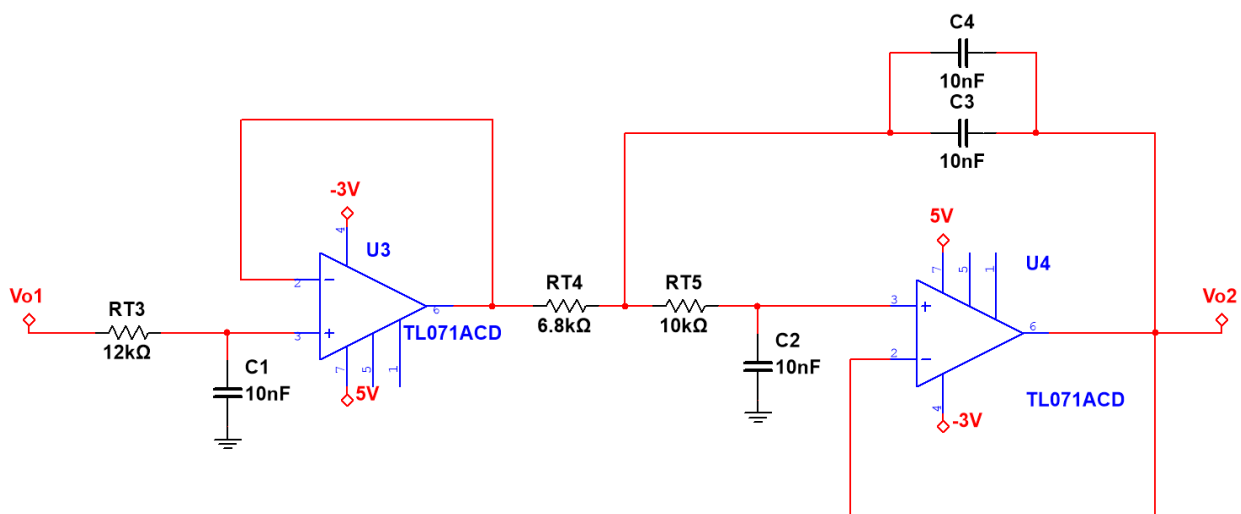


Figura 7 - Esquemático do estágio de condicionamento de sinais (AAF)

9

2.3. Sistema de Digitalização

Para a aquisição de dados, optou-se por usar o *Free-Running Mode*, já que essa configuração permite uma maior taxa de amostragem e precisão na periodicidade da coleta das amostras, quando comparado à configuração *Polling*. Apesar disso, como o projeto total não exige uma frequência de aquisição muito elevada, por ter como foco o estudo de um sistema de medição em condições de trabalho simples (pretende-se medir a variação da intensidade luminosa mediante a aproximação de uma lanterna, por exemplo) o modelo *Polling* poderia ser usado sem maiores consequências. Assim, seguindo o modelo proposto em aula para a operação em modo *Free-Running* do Arduino Uno, dentro da função *setup*, típica de um código desenvolvido para essa plataforma, definiu-se todas as portas analógicas (A0, A1, ..., A6, A7) como entrada de dados e a comunicação serial foi iniciada com um *baud rate* de 19200.

Entrando na função *loop* é chamada a função *handleSerial*, responsável por monitorar a porta serial do Arduino e por atualizar as variáveis responsáveis por iniciar e parar a aquisição, por definir o valor do *prescaler* do ADC e por selecionar o canal de leitura do ADC. Depois de tratar os eventos da serial, foram configurados os registradores ADMUX, ADCSRA, ADCSRB e DIDR0, responsáveis por definir o comportamento do ADC presente no microcontrolador e, por fim, o registrador SREG, responsável por habilitar as interrupções globais do processador. As análises da configuração desses registradores será feita considerando os bits ordenados na forma *Little-endian* de 0 a 7 (cada registrador tem 8 bits, ou 1 byte). Esses estão sendo setados no *loop* de forma a permitir que seus valores sejam controlados pela interface, conforme será descrito posteriormente.

Iniciando a configuração dos registradores do ADC, o registrador ADMUX teve seus 4 bits menos significativos setados para 0, de forma a selecionar o canal 0 do conversor. O bit 5 desse registrador foi setado em 0, de forma a definir que as amostras do ADC serão ajustadas à direita e os dois bits mais significativos foram setados em 01 de forma a selecionar a tensão de referência como a V_{cc} . Já o registrador ADCSRA foi configurado da seguinte forma: os 3 bits menos significativos foram definidos como 111, de forma a ajustar o *prescaler* como 128, resultando em uma frequência de amostragem de 9615,38Hz (para um canal apenas); o bit 3 foi setado para 1, habilitando as interrupções do ADC; o bit 5 foi setado para 1 de forma a habilitar o *auto-trigger* do ADC, permitindo a operação em modo *Free-running*; Em seguida, o registrador ADCSRB teve seus 3 bits menos significativos setados como 000 de forma a selecionar o *Trigger Source* como modo *Free-running*; o registrador DIDR0 é então definido com todos os bits em nível lógico alto, de forma a desabilitar o comportamento das portas analógicas como digitais; por fim, o bit 7 do registrador SREG é posto em nível lógico alto habilitando as interrupções externas do processador e os bits 7 e 4 do registrador ADCSRA são colocados em nível lógico alto de forma a habilitar o conversor AD e a inicializá-lo, respectivamente. No caso, a inicialização ou não da aquisição do conversor AD, através da alteração do bit 4 do ADCSRA, será habilitada de acordo com o valor da variável `MUST_READ` que é definida pelo usuário na interface, conforme será explicado nos próximos parágrafos.

Após configurados os registradores, dentro da função *loop*, existe segundo loop infinito que só será executado enquanto a variável `MUST_READ` estiver setada de forma a permitir a aquisição de dados. Essa variável é definida pela interface ao pressionar os botões “Começar Leitura” e “Parar Leitura”. Dentro desse loop a *flag isProcessing* que indica que os dados podem ser processados é constantemente monitorada, de forma que quando ela for habilitada pela rotina de tratamento de interrupção do ADC, quando o vetor de amostras tiver sido preenchido, os dados serão processados. No caso do sistema de aquisição desenvolvido, o processamento de dados corresponde à transmissão serial dos dados para o computador de controle, responsável por converter e plotar os dados na interface gráfica de visualização. Essa transmissão foi implementada de forma a usar a comunicação via USB com um computador. Ao término do processamento dos dados essa rotina volta a desabilitar a *flag* de controle, permitindo que a rotina de interrupção volte a armazenar os valores coletados. Caso essa *flag* não estiver habilitada o loop chama novamente a função *handleSerial* de forma a tratar os comandos da interface.

A função *handleSerial* basicamente monitora a porta serial do arduino, verificando se algum byte foi recebido via serial através da conexão USB do Arduino. Caso seja identificado que um byte foi recebido, a função começa a tratar os dados da serial. Assim, se esse byte for exatamente o responsável por iniciar a aquisição (`START_READ = 0x80`), a função realiza a leitura de 2 outros bytes enviados pela interface, responsável por setar o *prescaler* e o canal analógico do ADC selecionados na interface. Caso o byte recebido for diferente do comando para iniciar a aquisição a variável `MUST_READ` recebe o valor `STOP_READ = 0x7F`, que desabilita a aquisição de dados do conversor AD.

A rotina ISR é chamada sempre que o conversor AD finaliza uma coleta de amostra, interrompendo a execução sequencial normal do processador e, basicamente, realiza a leitura da amostra coletada pelo conversor AD e a armazena em um vetor de amostras de tamanho definido na posição adequada. Quando esse vetor é totalmente preenchido, essa rotina é responsável por definir um *flag* que indica que os dados coletados podem ser processados pela rotina principal e que desabilita o armazenamento das amostras coletadas pelo ADC.

```
/*  
 * DataAcquisition.ino  
 * Universidade Federal de Minas Gerais  
 * Created on: Fev 2021  
 * Author: Nander Carmo, Igor Radichi  
 * Version: 1.0  
 * License: MIT  
 */  
  
const uint8_t STOP_READ = 0x7F;  
const uint8_t START_READ = 0x80;
```

```
const uint8_t channelsCount = 1; // number of channels
const uint8_t dataBufferSize = 100; // size of the data buffers
uint8_t isProcessing = false; // flag to start data processing
uint16_t dataVector[channelsCount][dataBufferSize]; // data vectors for
each channel
uint32_t dataReadCount = 0; // controls the number of samples

uint8_t RESET = 0x00; // Leitura pausada
uint8_t ANALOG_CHANNEL = 0x00; // Leitura pausada
uint8_t VOLTAGE_REF = 0x40; // Leitura pausada
uint8_t PRESCALER = 0x07; // Leitura pausada
uint8_t EN_ADC_IRQ = 0x08; // Leitura pausada
uint8_t EN_ADC_AUTO_TRIGGER = 0x20; // Leitura pausada
uint8_t FREE_RUNNING_MODE = 0x00; // Leitura pausada
uint8_t DIS_ADC_DIG = 0xFF; // Leitura pausada
uint8_t EN_GLOBAL_IRQ = 0x80; // Leitura pausada
uint8_t MUST_READ = START_READ; // Leitura pausada
uint8_t ADC_START = 0x40; // Leitura pausada

void handleSerial();

void setup() {

    Serial.begin(19200);

    pinMode(A0, INPUT);
    pinMode(A1, INPUT);
    pinMode(A2, INPUT);
    pinMode(A3, INPUT);
    pinMode(A4, INPUT);
    pinMode(A5, INPUT);
    pinMode(A6, INPUT);
    pinMode(A7, INPUT);
}

void loop() {

    handleSerial();

    ADMUX = ANALOG_CHANNEL; // MUX[3:0] = 0000 -> select analog channel 0,
ADLAR = 0 -> AD samples are right adjusted
    ADMUX |= VOLTAGE_REF; // REFS[1:0] = 01, set voltage reference to AVcc

    ADCSRA = RESET;
    ADCSRA |= PRESCALER; // ADPS[2:0] = 111, set prescaler to 128 -> fs =
4807.69Hz (2 channels)
```

```
ADCSRA |= EN_ADC_IRQ; // ADIE = 1, enable ADC interrupts
ADCSRA |= EN_ADC_AUTO_TRIGGER; // ADATE = 1, enable auto-trigger

ADCSRB = FREE_RUNNING_MODE; // ACME = 0, ADTS[2:0] = 000 -> trigger
source = free running mode

DIDR0 = DIS_ADC_DIG; // disable the ADC digital input buffers

SREG |= EN_GLOBAL_IRQ; // enable global interrupts

ADCSRA |= MUST_READ; // ADEN = 1, enable AD converter
ADCSRA |= ADC_START; // ADSC = 1, start AD conversion

while(MUST_READ == START_READ) {

    if(isProcessing) {

        for(uint8_t i = 0; i < channelsCount; i++) {

            for(uint8_t j = 0; j < dataBufferSize; j++) {

                Serial.print((uint16_t) dataVector[i][j]);
            }
        }

        noInterrupts();
        isProcessing = false;
        interrupts();

    } else handleSerial();
}

void handleSerial() {

    if(Serial.available()) {

        uint8_t readChar = Serial.read();

        if(readChar == START_READ) {

            while(!Serial.available()) continue;

            PRESCALER = Serial.read();

            while(!Serial.available()) continue;
        }
    }
}
```

```
    ANALOG_CHANNEL = Serial.read();
    MUST_READ = START_READ;

    } else MUST_READ = STOP_READ;
}
}

ISR(ADC_vect) {

    //Serial.println("$ ISR");

    uint16_t sample;
    uint8_t CH;

    sample = ADCL; // read the lower byte
    sample += ADCH << 8; // read the upper byte shift by 8 bits left

    // char message[100] = "";
    // sprintf(message, "$ isProcessing = %s\n", isProcessing ? "true" :
    "false");
    // Serial.println(message);

    if(!isProcessing) {

        CH = ADMUX & 0x0F; // get AD channel
        dataVector[CH][dataReadCount] = sample; // store data read

        if(++CH < channelsCount) ADMUX += 1; // verify if all channels
        were acquired, if not, go to the next channel
        else {

            ADMUX &= 0xF0; // if so, turn to channel 0
            dataReadCount++; // update the number of samples
        }

        if(dataReadCount == dataBufferSize) {

            dataReadCount = 0; // data vector full, restart dataReadCount
            isProcessing = true; // set the flag to start processing
        }

    }
}
```

2.4. Sistema de Controle e Interface de Dados

Para a apresentação dos dados e para fazer o tratamento e processamento dos valores coletados pelo ADC, foi desenvolvida uma interface gráfica simples, usando o *framework* QT (C++). A interface apresenta um gráfico para visualização dos valores coletados pelo ADC, transmitidos pela USB e processados pelo computador de controle e permite a configuração de alguns parâmetros, como o valor do *prescaler* (e consequentemente da frequência de amostragem) que o ADC irá trabalhar, qual o canal do conversor AD que será utilizado (em qual porta do Arduino o sensor foi conectado) e, posteriormente poderia ser ampliada de forma a permitir que todos os registradores correspondentes ao ADC (ADMUX, ADCSRA, ADCSRB e DIDR0) e ao Arduino (SREG) sejam configurados através dela.

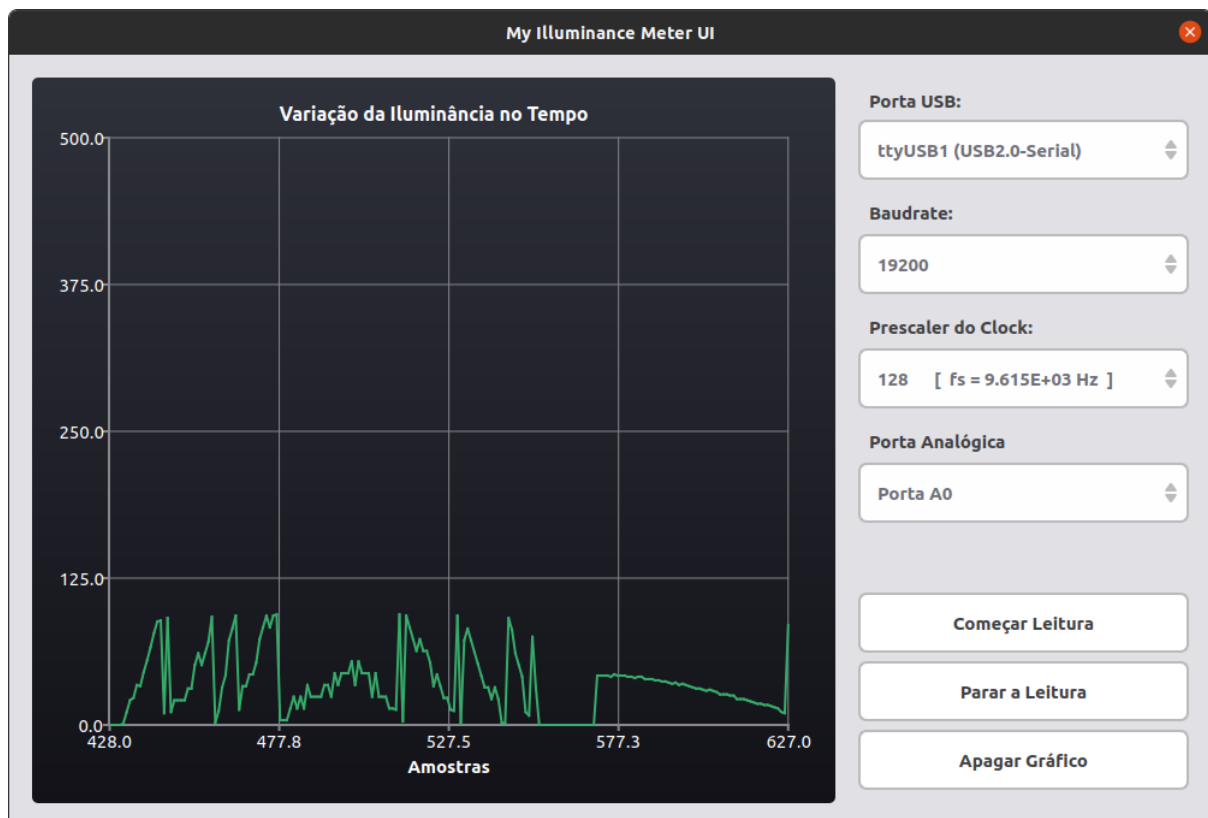


Figura 7 - Interface gráfica para apresentação dos dados coletados desenvolvida em QT.

Não será abordado aqui como a interface foi desenvolvida, porque foge do propósito deste documento que visa tratar da parte de aquisição e processamento de dados e do condicionamento de sinal necessário para realizar a medição de iluminância. Mas resumidamente, ao pressionar o botão “Começar Leitura” a interface envia para o microcontrolador os 3 bytes responsáveis por configurar e iniciar a aquisição de dados e passa a escutar a porta serial de forma a receber os dados enviados pela aquisição. Quando o botão “Parar a Leitura” é pressionado a interface envia o byte responsável por interromper o processo de coleta de amostras do ADC, interrompendo a leitura do sensor.

Como o circuito do condicionador de sinais e da polarização do BPW34 ainda não foram construídos fisicamente, os dados apresentados na figura acima ainda não correspondem a valores de iluminância. Esses valores são resultado da flutuação da porta analógica A0 selecionada na interface e que se encontra desconectada. A porta foi deixada dessa forma para que fosse possível capturar oscilações de sinal.

3. Apresentação e Análise de Resultados

4. Conclusão

5. Bibliografia

[1] Hilton de Oliveira Mota. ELE029 - Sistemas de Medição. Anotações de Aula, 2020/2.

[2] MathWorks. MATLAB. Acessado em: 04 de Fevereiro de 2021. Disponível em:

<<https://www.mathworks.com/help/matlab>>