Universidade Federal de Juiz de Fora
Programa de Pós-Graduação em Engenharia Elétrica
Mestrado em Engenharia Elétrica

Primeiro Trabalho

Aluno: Thiago Duque Saber de Lima

Juiz de Fora
7 de novembro de 2023

Sumário

1 Introdução

O primeiro trabalho da disciplina tem como objetivo criar um modulo de aquisição utilizando o Tiva e o Python. Desse modo, tem como pré-requisitos:

- O sistema deve possuir um filtro anti-aliasing;
- Deve-se utilizar o ADC com uma taxa de amostragem pré-determinada;
- Deve-se utilizar a DMA para realizar a cópia dos dados;
- Os dados amostrados devem ser representados graficamente no python, de maneira online (no domínio do tempo e/ou da frequência).

Dessa maneira, foi estabelecido um sistema com o objetivo de filtrar o sinal de um gerador através de um filtro sallen-key. A saída do filtro irá para um conversor AD periódico que, através da DMA, preenche um buffer. Com o buffer cheio, irá ser chamada uma interrupção que levantará uma flag. Essa flag permitirá transmitir os dados via uart, que será recebido por um computador. No computador será plotado, em tempo real, os dados recebidos. Na Figura 1 está ilustrado o funcionamento geral do trabalho.

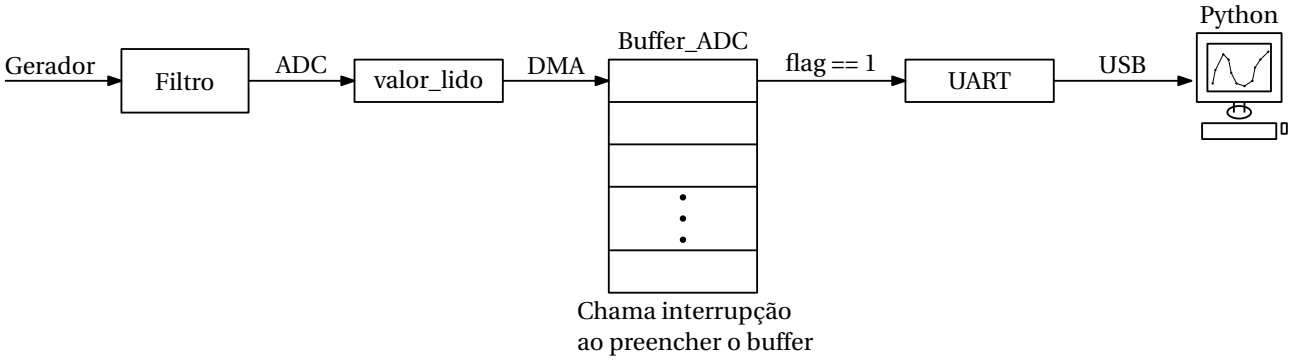

Figura 1: Caption

2 Filtro *anti-aliasing*

Na Figura 2 está ilustrado o filtro *anti-aliasing* projetado, sendo esse um filtro sallen-key de quarta ordem.

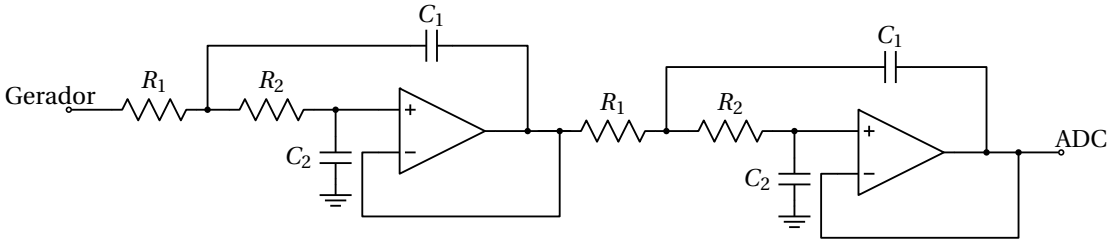

Figura 2: Filtro de quarta-ordem.

O sistema foi projetado para ter uma frequência de amostragem de 12 kHz. Sendo assim, foi projetado um filtro para ter frequência de corte em 3,3862 kHz com os componentes expostos na Tabela 1.

Componente	Valor	Unidade
R_1	$4,7$	kΩ
R_2	$4,7$	kΩ
C_1	10	nF
C_2	10	nF

Tabela 1: Valores dos componentes selecionados.

Na Figura 3 está ilustrado o diagrama de bode do filtro projetado. É possível perceber que o filtro não atinge os -74 dB recomendado em [1], porém atinge uma boa atenuação de aproximadamente -40 dB em 12 kHz e $-12{,}5$ dB em 6 kHz.

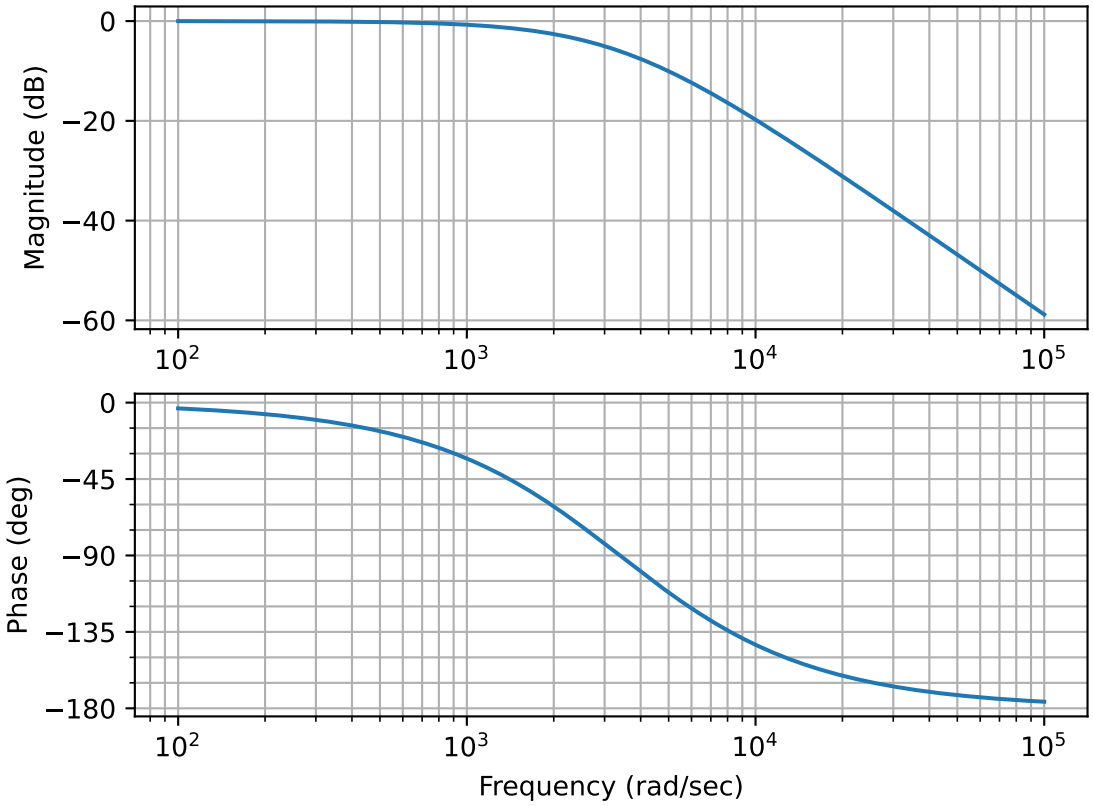

Figura 3: Diagrama de bode do filtro projetado.

O filtro projetado acima foi implementado em protoboard.

3 Amostragem

Para fazer a amostragem em um tempo pré-determinado, foi utilizado um timer periódico, conforme demonstrado em Código 1.

Código 1: Programação TIMER

```
// ------------- Configura o Timer -------------
uint32_t myCount = systemClock / SAMPLE_RATE;
                                                    //
   Define o valor maximo do contador para o timer
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); // Habilita o
   periferico de timer
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_TIMER0)){ }
TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC); // Seta o timer
   0 como periodico
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
TimerLoadSet(TIMER0_BASE, TIMER_A, myCount/2-1);
IntMasterEnable();
TimerEnable(TIMER0_BASE, TIMER_A);
```

A configuração do ADC está demonstrada em Código 2, em que é feita a leitura de ADC com o trigger pelo timer 0, inicializado em Código 1. Para o ADC será utilizado o sample sequencer 3, que configura uma única leitura de ADC.

Código 2: Programação ADC

```
// ------------- Configura o ADC -------------
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE); // Habilita o GPIO E
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOE)){ }
GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3); // Configura o pino
   E3 como ADC
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0); // Habilita o ADC0
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_ADC0)){ }
IntDisable(INT_ADC0SS3);
ADCIntDisable(ADC0_BASE, 3);
ADCSequenceDisable(ADC0_BASE, 3);
ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0);
                                                // Configura o ADC
    para ter o Sample Sequencer 3 (1 medicao) com trigger por
   timer e prioridade 0
                                                    // Liga o
   timer a entrada de trigger do adc
ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH0 |
   ADC_CTL_IE | ADC_CTL_END);                  // Configura a
   leitura do canal 0 e habilita interrupcao de termino de
   leitura ao final.
ADCSequenceEnable(ADC0_BASE, 3);     // Habilita a sequencia 3 no
    ADC0
ADCIntClear(ADC0_BASE, 3);
ADCSequenceDMAEnable(ADC0_BASE, 3);
ADCIntEnable(ADC0_BASE, 3);
IntEnable(INT_ADC0SS3);
```

4 µDMA e UART

A µDMA foi configurada para fazer a copia dos dados do ADC para um buffer de 1024 posições, que posteriormente será enviada para o programa em python. Em Código 3 está demonstrado o código para a µDMA.

Código 3: Programação µDMA

```
// ------------- Configura a DMA -------------
SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_UDMA)){ }
uDMAEnable();
uDMAControlBaseSet(pu8ControlTable);
uDMAChannelAttributeDisable(UDMA_CHANNEL_ADC3,
                            UDMA_ATTR_ALTSELECT |
                               UDMA_ATTR_HIGH_PRIORITY |
                            UDMA_ATTR_REQMASK);
uDMAChannelControlSet(UDMA_CHANNEL_ADC3,  UDMA_SIZE_16 |
   UDMA_SRC_INC_NONE |
                      UDMA_DST_INC_16 | UDMA_ARB_1);

                         // Configura o canal da DMA
uDMAChannelTransferSet(UDMA_CHANNEL_ADC3,
                        UDMA_MODE_BASIC,
                        (void *)(ADC0_BASE + ADC_O_SSFIFO3),
                        bufferADC,
                        BUFFER_LENGHT);

uDMAChannelAttributeEnable(UDMA_CHANNEL_ADC3, UDMA_ATTR_USEBURST)
   ;
uDMAChannelEnable(UDMA_CHANNEL_ADC3 | UDMA_PRI_SELECT);
SysCtlDelay(10);
```

O código utilizado para configuração da UART está demonstrado em Código 4.

Código 4: Programação UART

```
// ---------------------------------------- Configura a UART
   ----------------------------------------
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_UART0)){ }
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA)){ }
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0 | GPIO_PIN_1);
UARTConfigSetExpClk(
```

```
        UART0_BASE, systemClock, 115200,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
           UART_CONFIG_PAR_NONE));
```

O código exposto em Código 5 demonstra o código completo implementado no microcontrolador.

5 Recebimento de Dados

Para o recebimento dos dados foi utilizado o código exposto nos anexos, demonstrado em Código 8, Código 6 e Código 7. Tais códigos, utilizando a função **animate** da biblioteca **matplotlib** fazem a atualização do plot a cada 86 ms com os valores vindos da UART.

Na Figura 4 está demonstrada a plotagem em tempo real para uma leitura continua de aproximadamente 1,65 V. É possível perceber que existe um erro no código em python, uma vez que não são plotados valores condizentes.

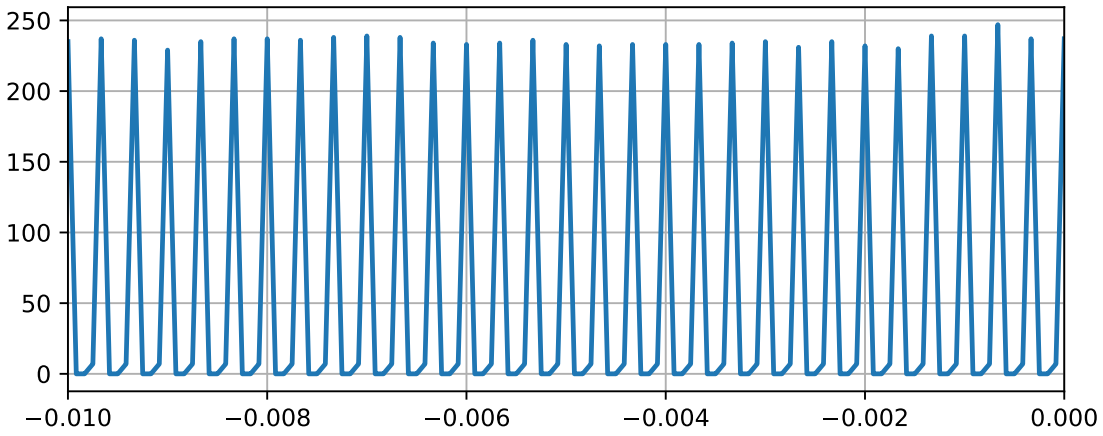

Figura 4: Plot em tempo real dos valores do ADC.

Referências

[1] Bonnie C. Baker and Microchip Technology Inc. An 699 anti-aliasing , analog filters for data acquisition systems. 1999.

Anexos

Código 5: Arquivo microcontrolador

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "inc/hw_adc.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"

   // Inclui a biblioteca de timer
#include "driverlib/interrupt.h"

   // Inclui a biblioteca de interrupt
#include "driverlib/adc.h"

   // Inclui a biblioteca de adc
#include "driverlib/uart.h"

   // Inclui a biblioteca de uart
#include "driverlib/udma.h"

   // Inclui a biblioteca de DMA

#define SAMPLE_RATE 12000

   // Define a frequencia de amostragem desejada
#define BUFFER_LENGHT 1024

   // Define o tamanho do buffer (12000/2400 = 200ms)

#pragma DATA_ALIGN(pu8ControlTable, 1024);

   // Diretiva de compila  para a DMA
uint8_t pu8ControlTable[1024];

   // Enderecos de memoria para configuracao da DMA


uint16_t bufferADC[BUFFER_LENGHT];

   // Buffer para armazenar as leituras do ADC
```

```
volatile uint16_t flag = 0;

   // Flag para buffer verificar o preenchimento do buffer
volatile uint16_t teste = 0;
//void sendInt(int dado);
//void ADCIntHandler();

void sendInt(int dado)
{

    char* char_ptr = (char*)&dado;
    int i;
    for(i = 0; i < sizeof(int); i++)
    {
        UARTCharPut(UART0_BASE, *(char_ptr+i));
    }
}

void ADCIntHandler(){
    ADCIntClear(ADC0_BASE, 3);
    flag++;
}

int main(void)
{
    uint32_t systemClock = SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ
       | SYSCTL_OSC_MAIN
    | SYSCTL_USE_PLL | SYSCTL_CFG_VCO_480), 120000000);

       // Define o clock do sistema (clock interno, 120 MHz)


    // ---------------------------------------- Configura a
       UART ----------------------------------------
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_UART0)){ }
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA)){ }
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
    GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0 |
       GPIO_PIN_1);
    UARTConfigSetExpClk(
            UART0_BASE, systemClock, 115200,
```

```
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
           UART_CONFIG_PAR_NONE));


// ---------------------------------------- Configura a DMA
    ----------------------------------------
SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_UDMA)){ }
uDMAEnable();
uDMAControlBaseSet(pu8ControlTable);
uDMAChannelAttributeDisable(UDMA_CHANNEL_ADC3,
                                UDMA_ATTR_ALTSELECT |
                                   UDMA_ATTR_HIGH_PRIORITY |
                                UDMA_ATTR_REQMASK);
uDMAChannelControlSet(UDMA_CHANNEL_ADC3,  UDMA_SIZE_16 |
   UDMA_SRC_INC_NONE |
                      UDMA_DST_INC_16 | UDMA_ARB_1);

                        // Configura o canal da DMA
uDMAChannelTransferSet(UDMA_CHANNEL_ADC3,
                       UDMA_MODE_BASIC,
                       (void *)(ADC0_BASE + ADC_O_SSFIFO3),
                       bufferADC,
                       BUFFER_LENGHT);

uDMAChannelAttributeEnable(UDMA_CHANNEL_ADC3,
   UDMA_ATTR_USEBURST);
uDMAChannelEnable(UDMA_CHANNEL_ADC3 | UDMA_PRI_SELECT);
SysCtlDelay(10);


    // ---------------------------------------- Configura o
        ADC ----------------------------------------
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE); // Habilita
       o GPIO E
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOE)){ }
    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3); // Configura
        o pino E3 como ADC

    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0); // Habilita o
        ADC0
    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_ADC0)){ }
    IntDisable(INT_ADC0SS3);
    ADCIntDisable(ADC0_BASE, 3);
    ADCSequenceDisable(ADC0_BASE, 3);
    ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_TIMER, 0);
```

```
                                                                    //
        Configura o ADC para ter o Sample Sequencer 3 (1
        medicao) com trigger por timer e prioridade 0
                                                                  //
        Liga o timer a entrada de trigger do adc
    ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH0 |
        ADC_CTL_IE | ADC_CTL_END);                    //
        Configura a leitura do canal 0 e habilita interrupcao
        de termino de leitura ao final.
    ADCSequenceEnable(ADC0_BASE, 3);     // Habilita a
        sequencia 3 no ADC0
    ADCIntClear(ADC0_BASE, 3);
    ADCSequenceDMAEnable(ADC0_BASE, 3);
    ADCIntEnable(ADC0_BASE, 3);
    IntEnable(INT_ADC0SS3);

// ---------------------------------------- Configura o
    Timer ----------------------------------------
uint32_t myCount = systemClock / SAMPLE_RATE;
                                                                  //
    Define o valor maximo do contador para o timer

SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0); // Habilita o
    periferico de timer
while(!SysCtlPeripheralReady(SYSCTL_PERIPH_TIMER0)){ }
TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC); // Seta o
    timer 0 como periodico
TimerControlTrigger(TIMER0_BASE, TIMER_A, true);
TimerLoadSet(TIMER0_BASE, TIMER_A, myCount/2-1); // Configura
    o periodo do timer
                                                                //
                                                                    In
                                                                    o
                                                                    ti

IntMasterEnable();

TimerEnable(TIMER0_BASE, TIMER_A);


while(1){
    if(flag == BUFFER_LENGHT) {
        IntDisable(INT_ADC0SS3);
```

```
            flag = 0;
            for(int i = 0; i < BUFFER_LENGHT; i++){
                sendInt(bufferADC[i]);
            }

            uDMAChannelTransferSet(UDMA_CHANNEL_ADC3,
                                   UDMA_MODE_BASIC,
                                   (void *)(ADC0_BASE +
                                      ADC_O_SSFIFO3),
                                   bufferADC,
                                   BUFFER_LENGHT);
            IntEnable(INT_ADC0SS3);
            uDMAChannelEnable(UDMA_CHANNEL_ADC3 | UDMA_PRI_SELECT
               );
        }
    }
}
```

Código 6: Arquivo main.py

```
from realtimeplot import RealTimePlot
import matplotlib.pyplot as plt

my_signal = [0x80, 0x83, 0x87, 0x8A, 0x8E, 0x91, 0x95, 0x98, 0x9B
   , 0x9E, 0xA2, 0xA5, 0xA7, 0xAA, 0xAD, 0xAF,
    0xB2, 0xB4, 0xB6, 0xB8, 0xB9, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF, 0
       xBF, 0xBF, 0xC0, 0xBF, 0xBF, 0xBF,
    0xBE, 0xBD, 0xBC, 0xBB, 0xB9, 0xB8, 0xB6, 0xB4, 0xB2, 0xAF, 0
       xAD, 0xAA, 0xA7, 0xA5, 0xA2, 0x9E,
    0x9B, 0x98, 0x95, 0x91, 0x8E, 0x8A, 0x87, 0x83, 0x80, 0x7C, 0
       x78, 0x75, 0x71, 0x6E, 0x6A, 0x67,
    0x64, 0x61, 0x5D, 0x5A, 0x58, 0x55, 0x52, 0x50, 0x4D, 0x4B, 0
       x49, 0x47, 0x46, 0x44, 0x43, 0x42,
    0x41, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x41, 0x42, 0
       x43, 0x44, 0x46, 0x47, 0x49, 0x4B,
    0x4D, 0x50, 0x52, 0x55, 0x58, 0x5A, 0x5D, 0x61, 0x64, 0x67, 0
       x6A, 0x6E, 0x71, 0x75, 0x78, 0x7C
    ]

port = 'COM14'
baudrate = 115200
comm = RealTimePlot(port = port, baudrate = baudrate, pong =
   False)
comm.getSignal(signal=my_signal)
comm.initAnimate()
plt.show()
```

Código 7: Arquivo realtimeplot.py

```
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import numpy as np
from communicate import Communicate


class RealTimePlot():
    _fig_width_cm = 18/2.4
    _fig_height_cm = 7/2.4
    _SAMPLE_TIME = 8.333e-5
    _WINDOW_TIME = 10e-3
    __i = 0

    def __init__(self, port: str, baudrate: int, pong = False):
        '''
        Classe para fazer o plot em tempo real.
        :param str port: porta de comunica  o com o
           microcontrolador
        :param int baudrate: baudrate da comunica  o
        :param bool pong: configura o envio de um sinal para o
           microcontrolador
        '''
        self._pong = pong
        self._y = []
        self._t = np.arange(0, -self._WINDOW_TIME, -self.
           _SAMPLE_TIME)
        self._fig, self._axs = plt.subplots(figsize=(self.
           _fig_width_cm, self._fig_height_cm), nrows = 1, ncols
           = 1)
        self._communicate = Communicate(port=port, baudrate=
           baudrate)

    def initAnimate(self) -> None:
        '''
        M todo para iniciar o plot em tempo real.
        '''
        print("teste")
        self._ani = FuncAnimation(self._fig, self._update,
           interval = 86)
        print("teste 2")


    def getSignal(self, signal : list) -> None:
        '''
        M todo para pegar o sinal que deve ser reenviado.
```

```
        :param list signal: sinal que deve ser reenviado
        '''
        self._signal = signal

    def _update(self, fig) -> None:
        print("Teste 3")
        try:
            if self._pong == True:
                print(f'Enviando a amostra {self.__i}: {self.
                    _signal[self.__i]}')
                self._communicate.send_int_to_uart(self._signal[
                    self.__i])
                self.__i = (self.__i + 1) % len(self._signal)
            data = self._communicate.read_int_from_uart()
            self._updateData(data)
            self._updatePlot()
        except Exception as e:
            print(f"Erro: {e}")

    def _updateData(self, y : int) -> None:
        '''
        M todo que atualiza o vetor de dados recebidos.
        '''
        # self._y.insert(0, y)
        # print(self._y)
        self._y = np.concatenate((y, self._y))
        if len(self._y) > len(self._t):
            self._y = self._y[:len(self._t)]
        # print(f'y: {self._y}')

    def _updatePlot(self) -> None:
        '''
        M todo que atualiza o gr fico em tempo real
        '''
        try:
            self._axs.cla()
            print(self._y)
            self._axs.plot(self._t[:len(self._y)], self._y,
                linewidth = 2, color = "tab:blue")
            self._axs.set_xlim([-self._WINDOW_TIME,0])
            # self._axs.set_ylim([60, 130])
            self._axs.grid()
            print("teste")
        except:
            print("Erro ao plotar gr fico")
```

Código 8: Arquivo communicate.py

```
from realtimeplot import RealTimePlot
import matplotlib.pyplot as plt

my_signal = [0x80, 0x83, 0x87, 0x8A, 0x8E, 0x91, 0x95, 0x98, 0x9B
   , 0x9E, 0xA2, 0xA5, 0xA7, 0xAA, 0xAD, 0xAF,
    0xB2, 0xB4, 0xB6, 0xB8, 0xB9, 0xBB, 0xBC, 0xBD, 0xBE, 0xBF, 0
       xBF, 0xBF, 0xC0, 0xBF, 0xBF, 0xBF,
    0xBE, 0xBD, 0xBC, 0xBB, 0xB9, 0xB8, 0xB6, 0xB4, 0xB2, 0xAF, 0
       xAD, 0xAA, 0xA7, 0xA5, 0xA2, 0x9E,
    0x9B, 0x98, 0x95, 0x91, 0x8E, 0x8A, 0x87, 0x83, 0x80, 0x7C, 0
       x78, 0x75, 0x71, 0x6E, 0x6A, 0x67,
    0x64, 0x61, 0x5D, 0x5A, 0x58, 0x55, 0x52, 0x50, 0x4D, 0x4B, 0
       x49, 0x47, 0x46, 0x44, 0x43, 0x42,
    0x41, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x41, 0x42, 0
       x43, 0x44, 0x46, 0x47, 0x49, 0x4B,
    0x4D, 0x50, 0x52, 0x55, 0x58, 0x5A, 0x5D, 0x61, 0x64, 0x67, 0
       x6A, 0x6E, 0x71, 0x75, 0x78, 0x7C
    ]

port = 'COM14'
baudrate = 115200
comm = RealTimePlot(port = port, baudrate = baudrate, pong =
   False)
comm.getSignal(signal=my_signal)
comm.initAnimate()
plt.show()
```