Título: **EXPERIMENTAÇÕES COM COMUNICAÇÃO**

Equipe: Yves Augusto e Paulo Henrique Araujo

**UART**

**a. Verifique a freqüência de operação da UART (bits/s) com o osciloscópio.**

**Código Arduino comentado:**

#include <SoftwareSerial.h>

SoftwareSerial mySerial(3, 2);

void setup()

{

Serial.begin(115200);

Serial.println("Entrada");

mySerial.begin(38400);

mySerial.println("Saida");

}

void loop()

{

if (mySerial.available())

Serial.write(mySerial.read());

if (Serial.available())

mySerial.write(Serial.read());

}

**Imagens do osciloscópio:**

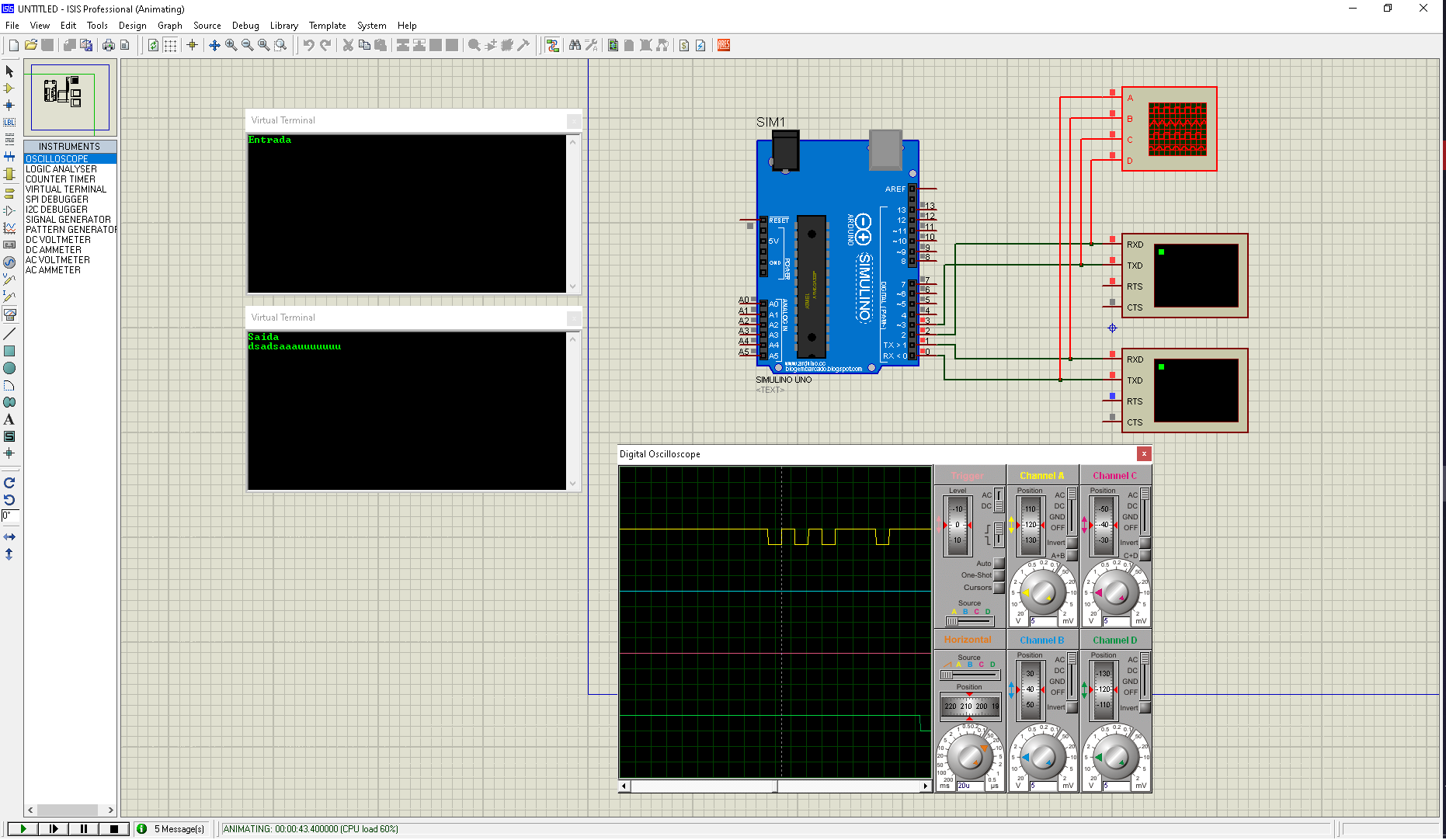


Imagem 1

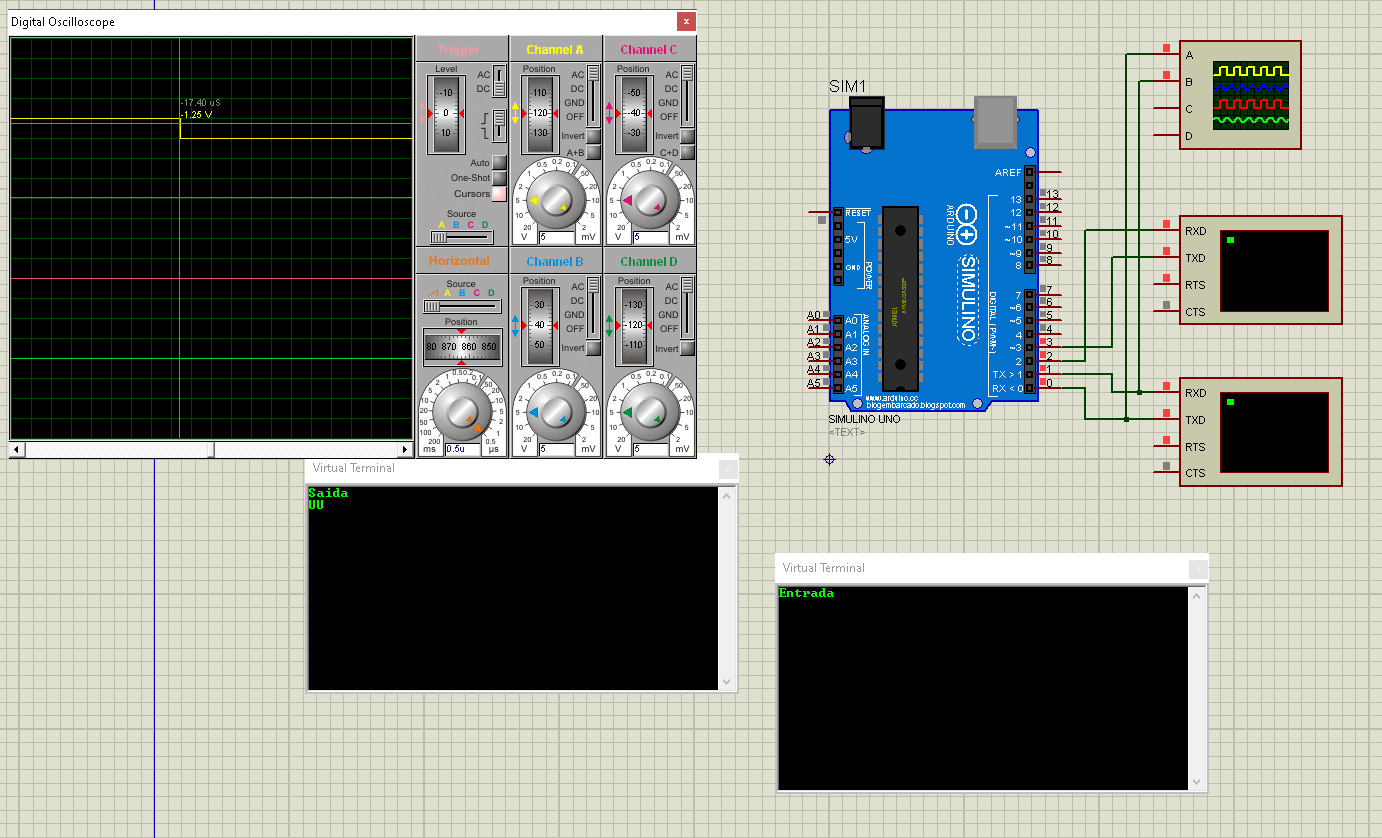


Imagem 2

**Resultados e Comentários:**

Com os baud rate dos terminais setados respectivamente 57600 e 38400, para a entrada e outro para a saída, vemos no osciloscópio a imagem em binário da entrada digitada. Vemos também no terminal de saída, a informação que foi dada na entrada.

Na imagem 2, vemos no osciloscópio, vemos que um bit equivale a (0 - (-17.4 uS)) = 17.4 uS, convertendo para frequência, vemos o resultado de 57.471.26, um resultado bem próximo da frequência utilizada (57.600).

**I2C (*Inter Integrated Circuit)***

**a. Verifique experimentalmente o endereço atribuído ao mestre da comunicação.**

**Código Arduino comentado:**

**------- CÓDIGO DO MASTER --------**

#include <Wire.h>

void setup()

{

Wire.begin();

}

byte x = 0;

void loop()

{

Wire.beginTransmission(4);

Wire.write("x is ");

Wire.write(x);

Wire.endTransmission();

x++;

delay(500);

}

**-------- CÓDIGO DO SLAVE ---------**

#include <Wire.h>

byte x = 0;

void setup()

{

Wire.begin(4); // atribui endereço 4 ao escravo

Wire.onReceive(receiveEvent); // registra a função receiveEvent

Serial.begin(9600);

}

void loop()

{

// este bloco de código resolve o problema de descobrir

// o endereço atribuído ao mestre

byte error, address;

// tenta enviar mensagens a todos os endereços de 1 a 255

for(address = 1; address < 127; address++){

Wire.beginTransmission(address);

error = Wire.endTransmission();

// quando a mensagem é bem sucedida, imprime o número

// do endereço no monitor

if (error == 0){

// no caso, o valor impresso é sempre 127

Serial.print("Foi achado um endereco: ");

Serial.print(address);

Serial.println("");

}

}

delay(100000);

}

void receiveEvent(int howMany)

{

while(1 < Wire.available())

{

char c = Wire.read();

Serial.print(c);

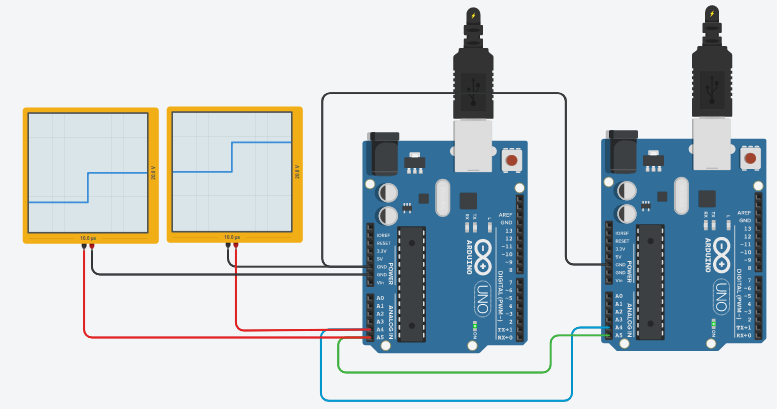
}

int x = Wire.read(); // quando não há mais loop,

Serial.println(x); // imprime o último valor de x

}

**Imagens do osciloscópio:**

****



**Resultados e Comentários:**

O endereço encontrado foi 1. O código varre todos os endereços até esbarrar num endereço válido, para, com isso, descobrir qual foi o endereço padrão que o Mestre atribuiu a si mesmo quando executou a linha “Wire.begin()”.

A ideia foi retirada deste tutorial: <https://playground.arduino.cc/Main/I2cScanner/>

**b. Qual a taxa de comunicação (bits/s) obtida no teste?**

**Código Arduino comentado:**

**------- CÓDIGO DO MASTER --------**

#include <Wire.h>

void setup()

{

Wire.begin();

}

byte x = 0;

void loop()

{

Wire.beginTransmission(4);

Wire.write("x is ");

Wire.write(x);

Wire.endTransmission();

x++;

delay(500);

}

**-------- CÓDIGO DO SLAVE ---------**

#include <Wire.h>

byte x = 0;

void setup()

{

Wire.begin(4); // atribui endereço 4 ao escravo

Wire.onReceive(receiveEvent); // registra a função receiveEvent

Serial.begin(9600);

}

void loop()

{

// este bloco de código resolve o problema de descobrir

// o endereço atribuído ao mestre

byte error, address;

// tenta enviar mensagens a todos os endereços de 1 a 255

for(address = 1; address < 127; address++){

Wire.beginTransmission(address);

error = Wire.endTransmission();

// quando a mensagem é bem sucedida, imprime o número

// do endereço no monitor

if (error == 0){

// no caso, o valor impresso é sempre 127

Serial.print("Foi achado um endereco: ");

Serial.print(address);

Serial.println("");

}

}

delay(100000);

}

void receiveEvent(int howMany)

{

while(1 < Wire.available())

{

char c = Wire.read();

Serial.print(c);

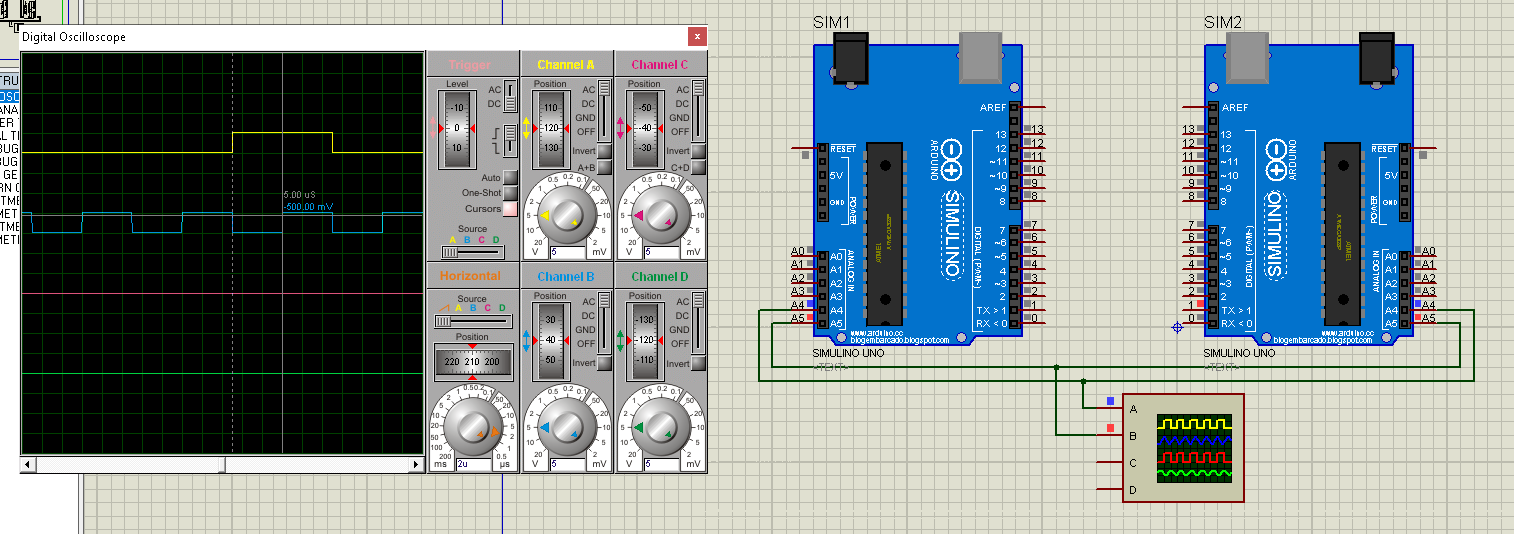
}

int x = Wire.read(); // quando não há mais loop,

Serial.println(x); // imprime o último valor de x

}

**Imagens do osciloscópio:**



**Resultados e Comentários:**

Vemos no osciloscópio que 1 bit corresponde a 5 uS, convertendo os valores, 1 segundo passa 200000 bits, logo sua taxa de transmissão de bit equivale a 200kbps

**SPI (*Serial Peripheral Interface Bus*)**

**a. Verifique experimentalmente como o destino da mensagem é indicado no padrão SPI? Explique.**

**Código Arduino comentado:**

**------- CÓDIGO DO MASTER --------**

#include <SPI.h>

void setup (void)

{

digitalWrite(SS, HIGH); // ensure SS stays high for now

SPI.begin ();

SPI.setClockDivider(SPI\_CLOCK\_DIV8);

}

void loop (void)

{

char c;

digitalWrite(SS, LOW); // SS is pin 10

for (const char \* p = "Experimento SPI\n" ; c = \*p; p++)

SPI.transfer (c);

digitalWrite(SS, HIGH);

delay (1000);

}

**-------- CÓDIGO DO SLAVE ---------**

#include <SPI.h>

char buf [100];

volatile byte pos;

volatile boolean process\_it;

void setup (void)

{

Serial.begin (115200);

SPCR |= bit (SPE);

pinMode(MISO, OUTPUT);

pos = 0; // buffer empty

process\_it = false;

SPI.attachInterrupt();

}

ISR (SPI\_STC\_vect)

{

byte c = SPDR; // grab byte from SPI Data Register

if (pos < (sizeof (buf) - 1))

buf [pos++] = c;

if (c == '\n')

process\_it = true;

}

void loop (void)

{

if (process\_it)

{

buf [pos] = 0;

Serial.println (buf);

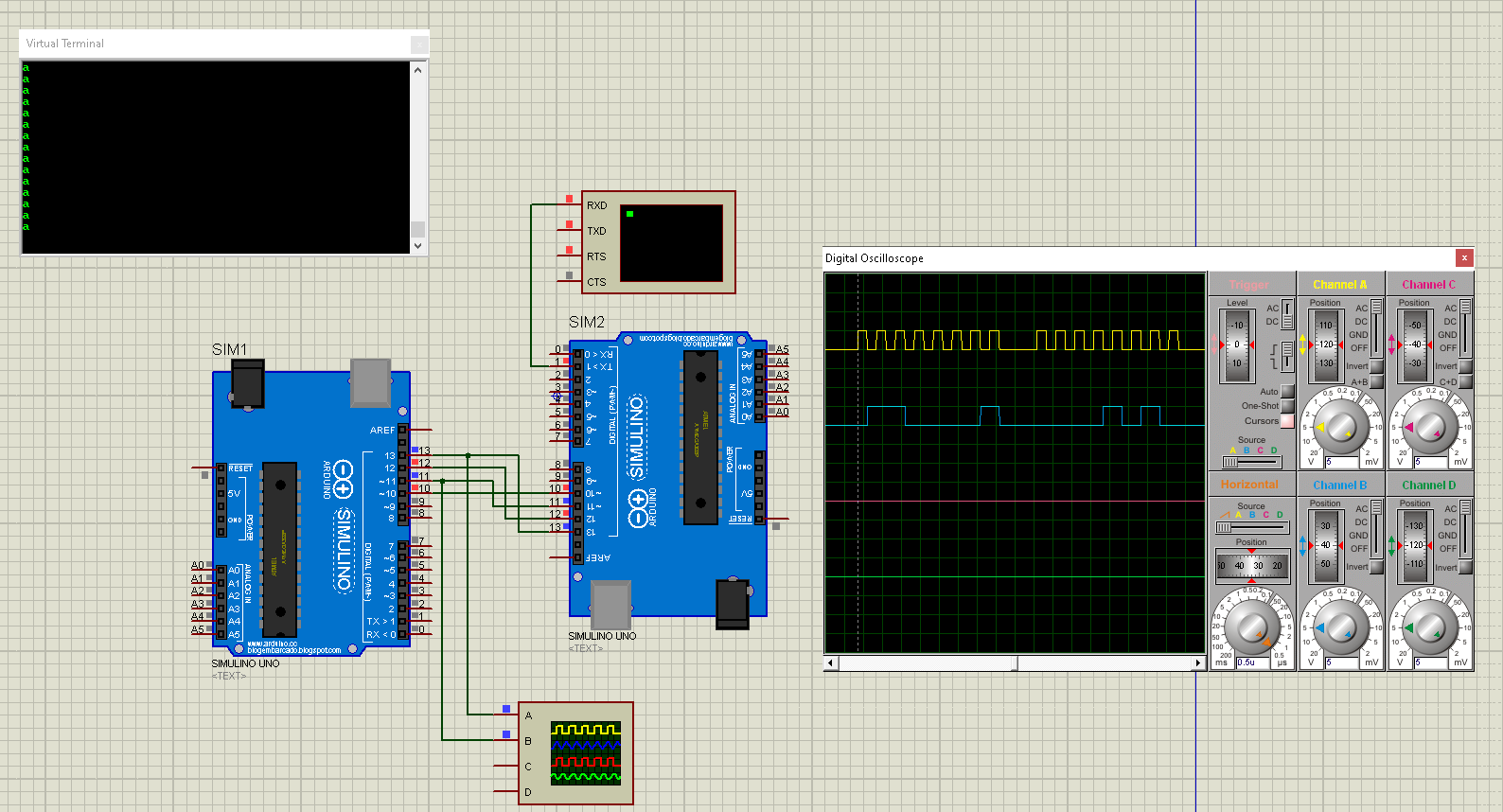
pos = 0;

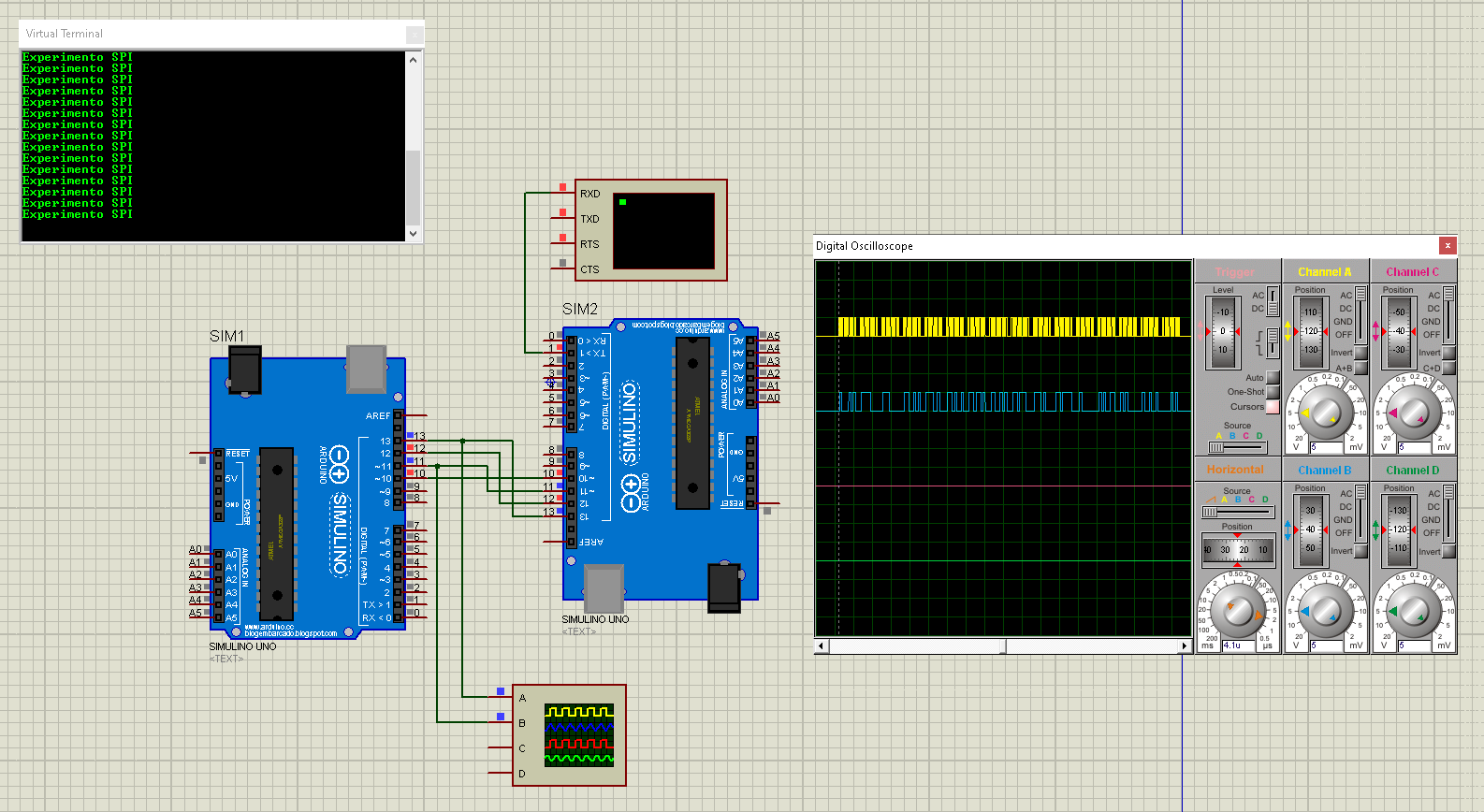
process\_it = false;

}

}

**Imagens do osciloscópio:**





**Resultados e Comentários:**

Observamos no osciloscópio que a frequência em azul representa a informação em binário enviada pelo master e a amarela representa slave. Na imagem 1 vemos, em azul, ele imprimindo o código binário de ‘a’ e na imagem 2 vemos ele imprimindo o texto ‘Experimento SPI’, já em amarelo vemos os bits variando de 0 e 1 para obtenção de dados e uma pausa em 0 para o fim/começo de uma nova letra.

OBS: Baudrate utilizado = 115200.

**b. Qual a taxa de comunicação (bits/s) obtida no teste?**

**Código Arduino comentado:**

#include <SPI.h>

void setup (void)

{

digitalWrite(SS, HIGH); // ensure SS stays high for now

SPI.begin ();

SPI.setClockDivider(SPI\_CLOCK\_DIV8);

}

void loop (void)

{

char c;

digitalWrite(SS, LOW); // SS is pin 10

for (const char \* p = "Experimento SPI\n" ; c = \*p; p++)

SPI.transfer (c);

digitalWrite(SS, HIGH);

delay (1000);

}

**-------- CÓDIGO DO SLAVE ---------**

#include <SPI.h>

char buf [100];

volatile byte pos;

volatile boolean process\_it;

void setup (void)

{

Serial.begin (115200);

SPCR |= bit (SPE);

pinMode(MISO, OUTPUT);

pos = 0; // buffer empty

process\_it = false;

SPI.attachInterrupt();

}

ISR (SPI\_STC\_vect)

{

byte c = SPDR; // grab byte from SPI Data Register

if (pos < (sizeof (buf) - 1))

buf [pos++] = c;

if (c == '\n')

process\_it = true;

}

void loop (void)

{

if (process\_it)

{

buf [pos] = 0;

Serial.println (buf);

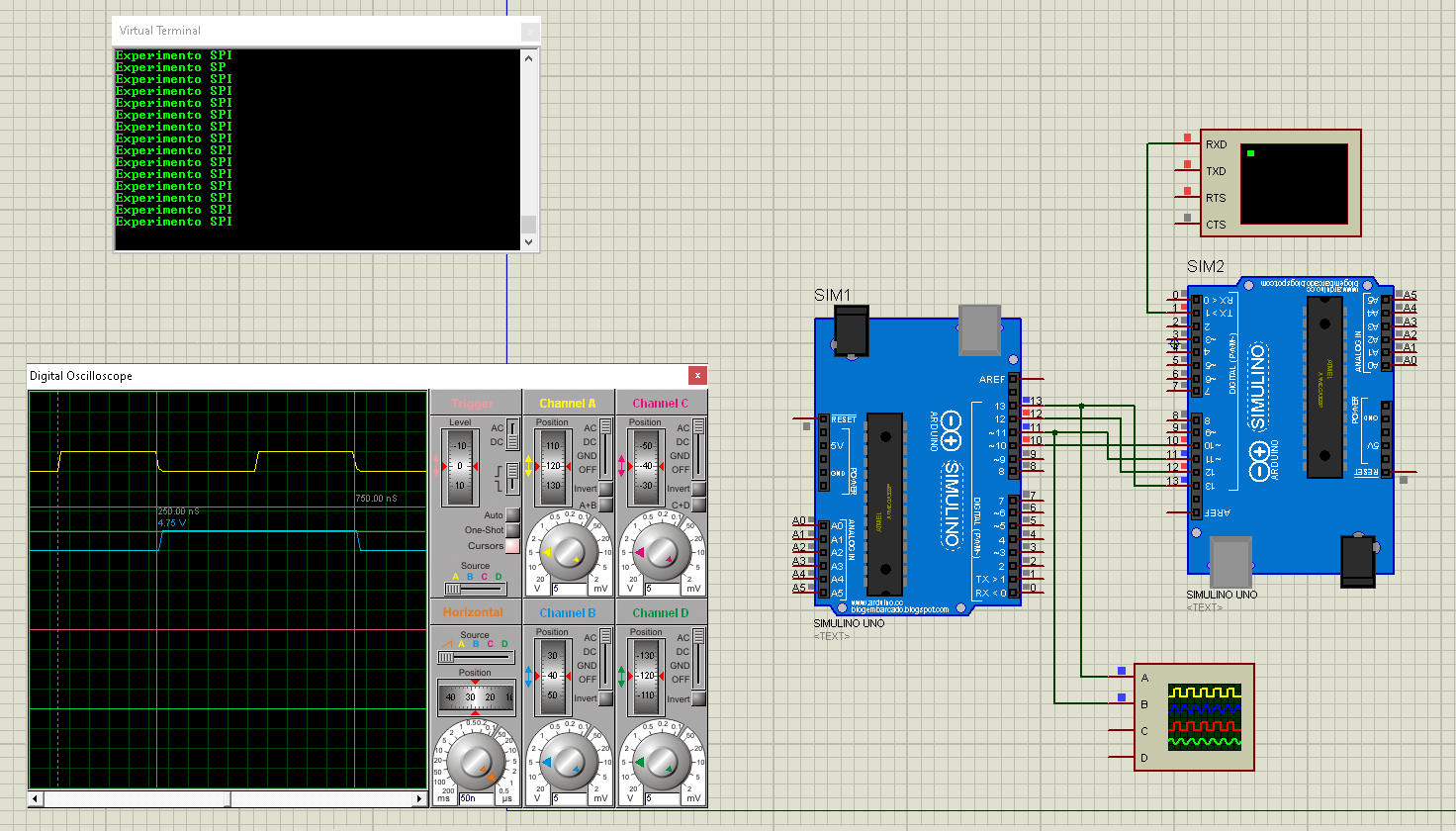
pos = 0;

process\_it = false;

}

}

**Imagens do osciloscópio:**



**Resultados e Comentários:**

Na imagem, vemos no osciloscópio, vemos que, no sinal do SCK(azul) um bit equivale a (750 nS - 250 nS) = 500 nS, convertendo para frequência, vemos o resultado de 2Mbps, já o MOSI(amarelo), como seu tempo é metade do SCK, temos o dobro da frequência, já que é inversamente proporcional chegando a um resultado de 4Mbps.

**Mais Questões:**

Compare os três padrões quanto a robustez e custo.

De um lado, o protocolo I2C possui recursos para lidar com problemas de comunicação, como as funcionalidades ACK/NACK que tornam mais sofisticado o processo de manipulação de erros, utilizando 8 bits para informar se a recepção dos dados foi bem sucedida ou não. De outro, o modelo de comunicação SPI não possui controle de fluxo, nem meios para confirmar se os dados foram adequadamente recebidos. Por esses motivos, o protocolo I2C é mais robusto que o SPI. Mas podemos também afirmar que há mais robustez no I2C que no UART, pois este último só dispõe, para fins de auto-checagem, de bits de paridade, recurso que é inferior aos meios empregados pela tecnologia I2C. Não obstante, a UART ainda possui algum método de manipulação de erros, que é a paridade dos bits, coisa que não se encontra no modelo SPI.

Fica, portanto, ordenada a classificação:

I2C > UART > SPI, quanto à robustez.

I2C < SPI < UART, quanto ao preço.

fontes:

[1] <https://www.researchgate.net/publication/224376183_An_introduction_to_I2C_and_SPI_protocols>

[2] <https://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html>

[3] <https://www.allaboutcircuits.com/technical-articles/introduction-to-the-i2c-bus/>

[4]<http://www.gammon.com.au/spi>

Faça recomendações de uso desses padrões em aplicações embarcadas considerando o custo.

**UART:** Como a tecnologia UART não é um protocolo de comunicação como o SPI e o I2C, e possui limitações seja no campo da transmissão como no número de dispositivos, é melhor que esta modalidade seja empregada quando se quiser enviar dados de um dispositivo a outro, no máximo dois, sem nutrir elevadas expectativas acerca da velocidade de transmissão.

**I2C:** Devido a este protocolo possuir um esquema pré-definido — Master e Slave —, capaz de relacionar os dispositivos, orquestrando múltiplas tarefas de transmissão e recepção, é recomendado para situações onde há diversos dispositivos, não mais que 127. A velocidade de envio dos dados é superior à da UART.

**SPI:** Este aqui se combina muito bem com aplicações onde a velocidade das transmissões seja um fator crucial, e haja necessidade de coordenar muitos dispositivos: companhias, empresas, etc.

Algumas placas de desenvolvimento oferecem um conector padrão RS-232, como o da figura. Por que vc não poderia conectar TX/RX do Arduino diretamente a um conector RS-232?

