LAB 3 - Digital-to-Analog Conversion (DAC)

Ana Lopes (98587)[1] and Mariana Mourão (98473)[2]

[1] *Instituto Superior Técnico,*

Integrated Master's in Biomedical Engineering,

ana.rita.santos.lopes@tecnico.ulisboa.pt

[2] *Instituto Superior Técnico,*

Integrated Master's in Biomedical Engineering,

mariana.mourao@tecnico.ulisboa.pt

Notas: O presente relatório visa testar experimentalmente a conversão digital-analógico (DAC), caracterizando-se como um circuito eletrónico com a função de converter uma grandeza digital (código binário) numa grandeza analógica (tensão).

Para o *streaming* dos dados, adaptámos o código desenvolvido na sessão laboratorial anterior para transmitir as entradas analógicas A0 e A1 a 50 Hzs, juntamente com o tempo decorrido desde que a placa começou a executar o programa, através da porta série (com uma taxa de transmissão de 115200 bps). A transmissão dos valores fez-se sob a forma de uma sequência formatada de valores separados por vírgula (CSV), com a análise dos mesmos a ser realizada no Excel.

Questões:

1. Neste exercício, o *Seeeduino Nano* é configurado como um gerador de ondas quadradas com amplitude [0; Amplitude Vcc], frequência de 1 Hz e ciclo de trabalho (*duty*) de 50%. Para a realização deste exercício, utilizámos o código já desenvolvido no laboratório anterior, no qual o Arduino realiza amostragem e quantização dos valores analógicos bem como a transmissão dos códigos digitais correspondentes.

O gerador de ondas quadradas é um tipo de gerador usado para gerar uma onda em forma de quadrado. Este foi obtido configurando o pin digital (D2) enquanto *output*, através da função

pinMode(), estabelecendo a voltagem de operação Vcc (5V, para o *Seeeduino Nano*) quando é atribuído o estado *HIGH*, e 0V quando atribuído o estado *LOW*. Consoante a frequência da onda e do ciclo de trabalho, define-se o período temporal dos níveis da onda quadrada. O ciclo de trabalho corresponde à razão entre o período em que a tensão da onda é Vcc (*on-time*, em oposição ao período em que não está ativo, *off-time*) e o período total da onda. No caso de se definir o ciclo de trabalho a 0%, não são gerados pulsos, pois D2 está sempre a 0V (*LOW*). O oposto seria obtido se fosse definido 100% de ciclo de trabalho, em que o pulso seguinte começaria imediatamente após o anterior, fazendo com que o sistema estivesse sempre a Vcc. Já para um ciclo de trabalho de 50%, uma onda quadrada é obtida.

Note-se que de forma a visualizar a onda gerada, a sequência de valores analógicos foi transmitida para A0 (Figura 3), realizando a sua quantização através do ADC (códigos digitais 1023 e 0 atribuídos às tensões Vcc e 0V, respetivamente). As montagens do circuito, quer no *Seeeduino Nano* quer no simulador, encontram-se nas Figuras 1 e 2. No Anexo I, disponibiliza-se o código desenvolvido para este exercício, incorporando-se os comentários considerados necessários.

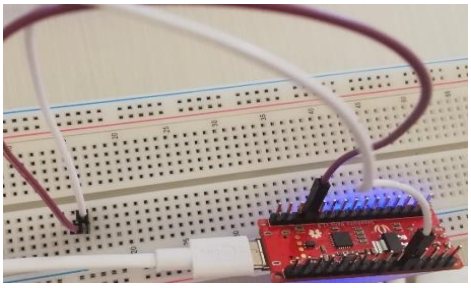

Figura 1- Montagem no Seeeduino Nano, para obter um gerador de onda quadrada. Pin digital D2 ligado à entrada analógica A0 através de dois cabos.

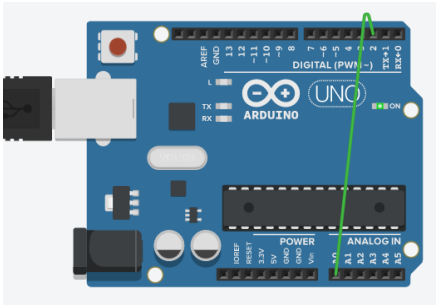

Figura 2- Montagem no simulador de Arduino do tinkerCad, para obter um gerador de onda quadrada.

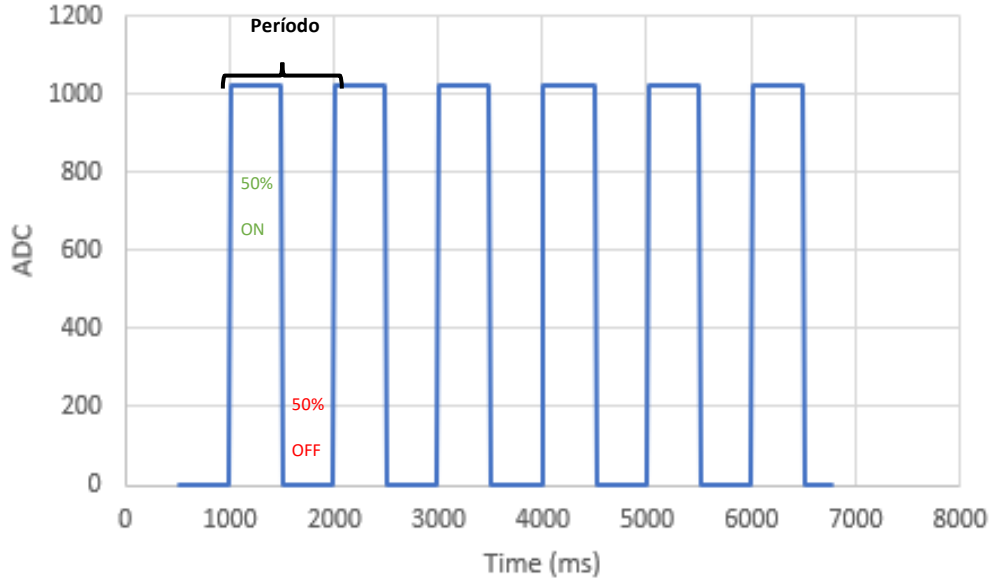

Figura 3 - Onda quadrada de 1 Hz e amostrada a 50Hz, tendo em conta os valores transmitidos através da entrada analógica A0 e apresentados no Serial Monitor do Arduino IDE, a partir da montagem no Seeeduino Nano (Figura 1).

Nos dados recolhidos experimentalmente, com ambos os microcontroladores (Simulador e *Seeeduino Nano*), obteve-se uma onda quadrada com período de 1 segundo (1000 ms, visto a sua frequência ser de 1 Hz), ou seja, durante o período de 1 segundo obtivemos 50% em 1023 e 50% em 0, o que corresponde a 0,5 segundos (t) em cada ciclo de trabalho. Dada a frequência de amostragem de 50 Hz (fs), o período de amostragem (intervalo de tempo entre amostras) corresponde a 0,02 segundos, pelo que cada ciclo de trabalho é definido por $N = t \times fs = 25$ amostras. Visto o período total da onda ser representado por 50 amostras, obtém-se uma onda quadrada. Recorrendo ao Excel, analisou-se a onda quadrada gerada, tendo-se confirmado as 25 amostras referentes a cada ciclo de trabalho.

2. O *Seeeduino Nano* é configurado como um gerador de ondas quadradas, como no exercício anterior, no entanto com as seguintes alterações: o utilizador tem, agora, controlo sobre a frequência da onda, sendo ajustada por um comando enviado através da *Serial Port* (como por exemplo, ao enviar F5 para a *Serial Port*, define-se a frequência da onda a 5 Hz); a amplitude da onda quadrada definida para [0; Vcc/2]. De forma a estabelecer a amplitude pretendida,

recorreu-se a um divisor de tensão, com duas resistências de igual valor (R = 10k ohm), obtendo-se no nó entre as duas resistências uma tensão $Vout = \frac{Vin}{2}$, com $Vin = Vcc$.

Os valores da onda gerada foram transmitidos para a entrada analógica A1, adicionalmente à onda quadrada gerada no exercício 1, com o dobro da amplitude, a qual é transmitida para a entrada analógica A0. As montagens do circuito, quer no *Seeeduino Nano* quer no simulador, encontram-se nas Figuras 4 e 5. No Anexo II, disponibiliza-se o código desenvolvido para este exercício, incorporando-se os comentários considerados necessários.

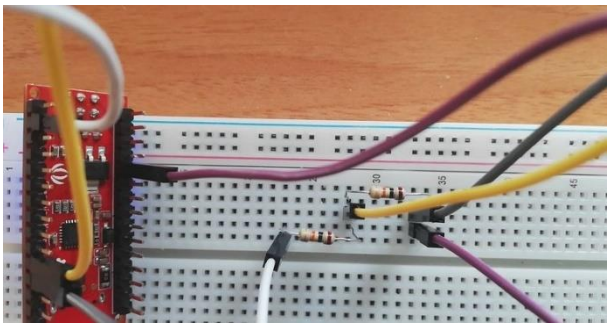

Figura 4- Montagem no Seeeduino Nano, incorporando um divisor de tensão (2 resistências de 10 KOhm) para obtenção da onda a ser transmitida para a entrada analógica A1, estabelecendo-se a ligação através do cabo amarelo. O cabo cinzento estabelece a ligação entre o pin digital D2 (gerador de onda quadrada de amplitude [0; Vcc]) com a entrada analógica A0. O cabo branco estabelece a ligação com o ground GND.

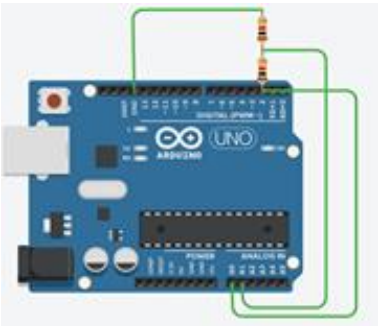

Figura 5- Montagem no simulador de Arduino do tinkerCad.

Nos dados recolhidos experimentalmente, com ambos os microcontroladores (Simulador e *Seeeduino Nano*), para uma onda quadrada de 1 Hz (Figura 6) voltou-se a verificar o referido no exercício 1, acerca do número de amostras que caracteriza cada nível (25 amostras), indo ao encontro do que é teoricamente previsto. As oscilações observadas relativamente aos códigos digitais atribuídos, devem-se ao erro de quantização inerente.

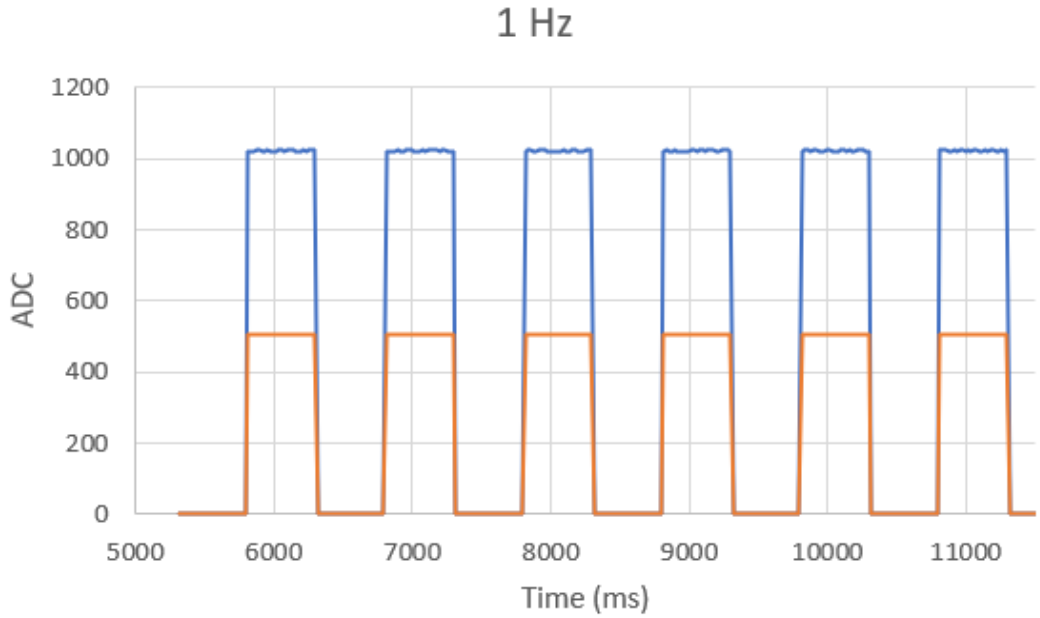

Figura 6 - Ondas quadradas de 1 Hz e amostradas a 50 Hz, tendo em conta os valores transmitidos através das entradas analógicas A0 (azul – onda de amplitude [0; Vcc]) e A1 (laranja – onda de amplitude [0; Vcc/2]) e apresentados no Serial Monitor do Arduino IDE, a partir da montagem no Seeeduino Nano (Figura 4).

Ao definir-se uma frequência de 10 Hz (Figura 7), o período da onda será de, teoricamente, 0.1 segundos, sendo que para um ciclo de trabalho de 50%, cada nível será definido por 0.05 segundos (t). Dada a frequência de amostragem (fs) de 50 Hz, o período de amostragem corresponde a 0,02 segundos, pelo que cada ciclo de trabalho é definido por $N = t \times fs = 2.5$ amostras. Recorrendo ao Excel, analisou-se a onda quadrada gerada, tendo-se confirmado que o nível de 1023 era consistentemente caracterizado por 2 amostras, sendo o nível de 0 por 3 amostras. Este resultado vai de encontro ao teoricamente esperado, uma vez que a frequência de amostragem e da onda conduzem a que 1 ciclo da onda seja amostrado por 5 amostras, não existindo, no domínio digital, uma exata simetria da onda.

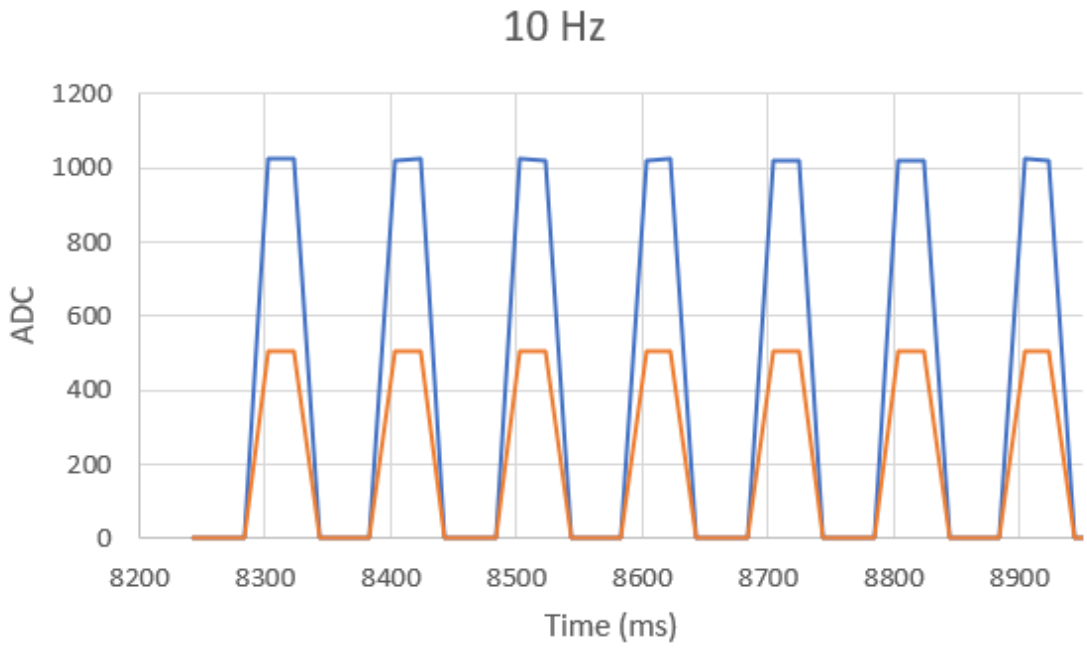

Figura 7 - Ondas quadradas de 10 Hz e amostradas a 50 Hz, tendo em conta os valores transmitidos através das entradas analógicas A0 (azul – onda de amplitude [0; Vcc]) e A1 (laranja – onda de amplitude [0; Vcc/2]) e apresentados no Serial Monitor do Arduino IDE, a partir da montagem no Seeeduino Nano (Figura 4).

Testando para uma frequência de 25 Hz (Figura 8), o período da onda (T) será de, teoricamente, 0.04 segundos. Dada a frequência de amostragem (fs) de 50 Hz, o período de amostragem (Ts) corresponde a 0,02 segundos. Segundo o teorema de *Nyquist*, para uma determinada frequência de amostragem, a frequência máxima que pode ser representada sem sofrer *aliasing* designa-se frequência de *Nyquist*, correspondendo a metade da frequência de amostragem ($f_N = \frac{fs}{2}$). Desta forma, para o sinal analógico ser recuperado a partir do sinal digital, as frequências contidas no mesmo têm de respeitar a expressão 1, com Ts o período de amostragem e T o período da onda. Os sinais com componentes de frequência acima da frequência de *Nyquist* sofrem *aliasing*, passando a assumir um valor entre DC e a frequência de *Nyquist* (as frequências são subestimadas quando o sinal é reconstruído). Por exemplo, um componente de frequência $f_N <$ f0 $<$ fs é representado como $fs - f0$, sendo a fórmula geral para determinar a *alias frequency* (fa) dada pela expressão 2, com *m* como sendo o múltiplo inteiro da frequência de amostragem mais próximo da frequência do sinal analógico (f0). Ora, o caso analisado corresponde à situação limite, com $f = \frac{fs}{2}$, sendo que pela expressão 2, $fa = f0 = 25\ Hz$, com $m = 1$, pelo que se concluiria que frequência seria corretamente

representada. Note-se que, contudo, dependente da primeira amostra considerada (processo não determinístico), o sinal pode na realidade ser representado por uma constante (amostragem acontece sempre no mesmo ponto, não produzindo oscilação), pelo que a frequência de 50 Hz no sinal original passaria a ser 0Hz aquando reconstruído. Na figura 9 ilustram-se duas possibilidades aquando da amostragem do sinal, produzindo uma onda de frequência 25 Hz (1 amostra por ciclo de trabalho de 50%, quando fs = 50Hz) ou de 0Hz (todas as amostras referentes ao mesmo nível). Analisando o experimentalmente obtido (Figura 8), constata-se que o primeiro comportamento (1 amostra por nível) foi o obtido. Assim, conclui-se que uma onda quadrada de frequência $\frac{fs}{2}$ apresenta-se distorcida no domínio digital, tendo-se obtido uma onda triangular.

$$f < \frac{fs}{2} \Leftrightarrow T > Ts \times 2 \qquad \text{Expressão 1}$$

$$fa = |m \times fs - f0| \qquad \text{Expressão 2}$$

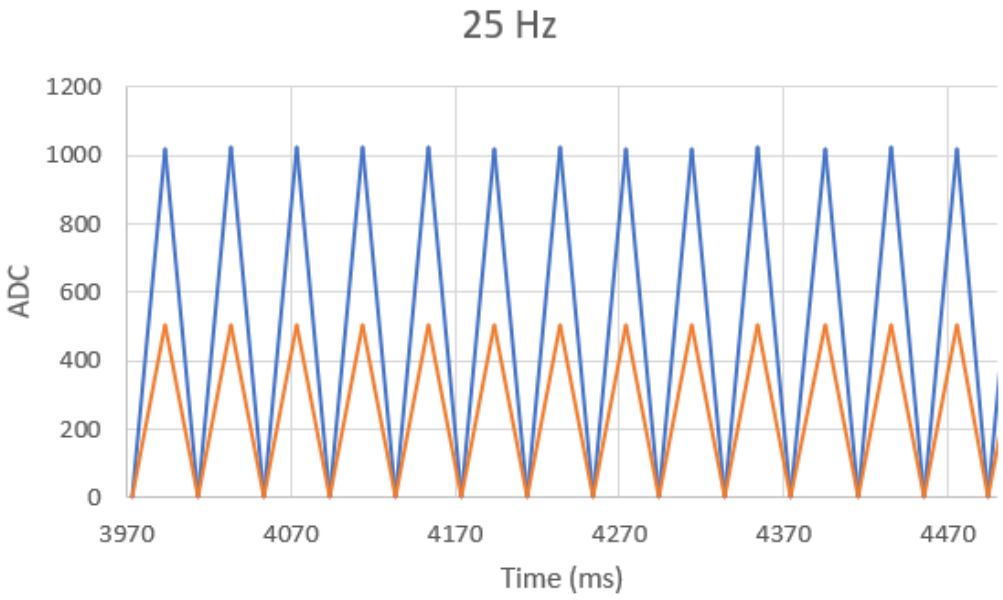

Figura 8 - Ondas quadradas de 25 Hz e amostradas a 50 Hz, tendo em conta os valores transmitidos através das entradas analógicas A0 (azul – onda de amplitude [0; Vcc]) e A1 (laranja – onda de amplitude [0; Vcc/2]) e apresentados no Serial Monitor do Arduino IDE, a partir da montagem no Seeeduino Nano (Figura 4). Ondas triangulares são obtidas experimentalmente.

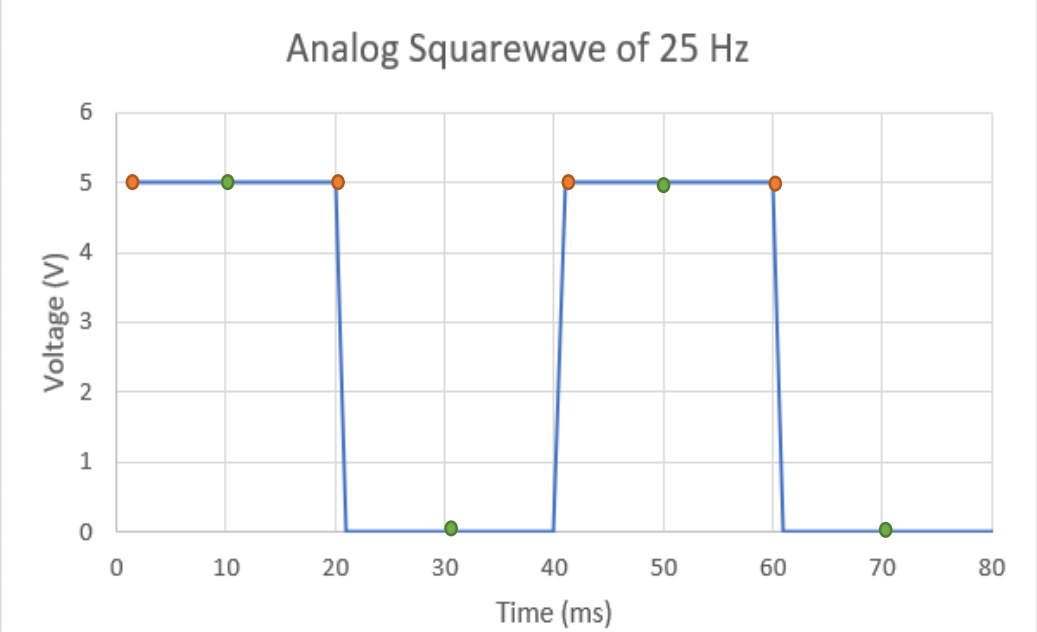

Figura 9 – Ilustração de uma onda analógica de 25 Hz, amostrada de duas formas diferentes, com o espaçamento teórico entre amostras de 20 ms (frequência de amostragem de 50 Hz). As amostras a laranja conduzem a uma reta constante no domínio digital. As amostras a verde permitem representar cada um dos níveis, ainda que com apenas 1 amostra, pelo que não se representa uma onda quadrada no domínio digital, mas sim triangular.

Por fim, de forma a testarmos o fenómeno de *aliasing* mais evidentemente, definiu-se 100 Hz enquanto frequência de onda, não verificando o Teorema de *Nyquist* ($f = 100 > \frac{fs}{2} = 25$). Pela expressão 2, a frequência $f0 = 100\ Hz$ é representada por $fa = 0\ Hz$ ($m = 2$), caso em que o sinal estaria sempre a ser amostrado no mesmo ponto em cada período. Em termos experimentais, a amostragem não se traduz desta forma determinística (comportamento não regular da onda obtida – Figura 10), dependendo também do microcontrolador que se usa. Analisando o experimentalmente obtido (Figura 10), constata-se que as ondas estão distorcidas face ao desejado, em que cada ciclo de trabalho deveria ter uma duração de 0.005 segundos, contudo não possibilitado pelo período de amostragem de 0.02 segundos (reconstrução do sinal original não é conseguida, encontrando-se mascarado).

Enquanto conclusão final, tem-se que para corretamente representar uma onda quadrada no domínio digital, com ciclo de trabalho 50%, tem-se que, em primeira análise, o período da mesma deverá ser representado por um número par de amostras ($N = \frac{fs}{f}$), de forma que seja atribuído a cada nível o mesmo número de amostras, ao contrário do observado para a

frequência de 10 Hz. Por outro lado, a frequência da onda quadrada terá de respeitar o Teorema de *Nyquist*, de forma que não sofra *aliasing*, sendo que quanto maior a frequência de amostragem, mais amostras são adquiridas num dado período, permitindo uma melhor representação do sinal original do que uma frequência de amostragem inferior.

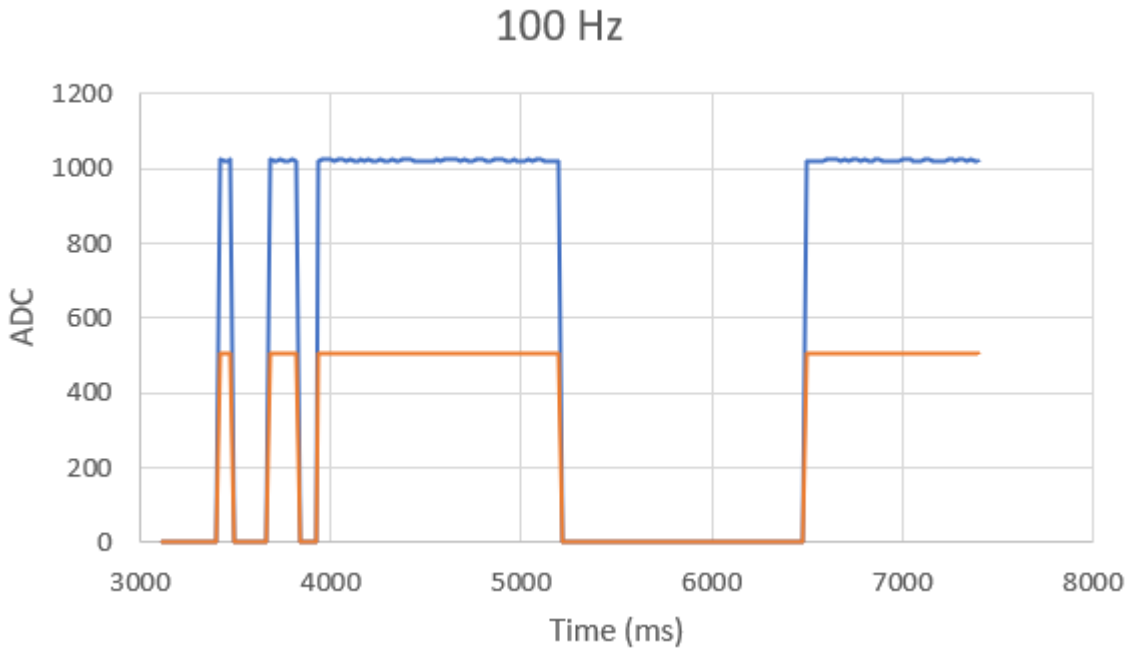

Figura 10 - Ondas quadradas de 100 Hz e amostradas a 50 Hz, tendo em conta os valores transmitidos através das entradas analógicas A0 (azul – onda de amplitude [0; Vcc]) e A1 (laranja – onda de amplitude [0; Vcc/2]) e apresentados no Serial Monitor do Arduino IDE, a partir da montagem no Seeeduino Nano (Figura 4). Obtém-se uma onda que sofreu aliasing.

3. Uma das falhas associadas ao *Arduino Uno* e ao *Seeduino Nano* é o facto de estes não possuírem um conversor digital-analógico (DAC), o que torna impossível a conversão direta dos pins em tensão analógica. Tal pode ser resolvido recorrendo à *R2R ladder*. A *R2R ladder* é uma forma simples de realizar conversão digital-analógica, utilizando arranjos repetitivos de redes de resistência precisas, apenas requerendo valores de R e 2R (dobro de R), em que estas atuam como divisores de tensão.

Um circuito DAC de n = 3 bits, construído a partir de um *R2R ladder*, obtém-se ao representar-se os 3 bits (bit a_{n-1} – bit mais significativo, MSB – até o bit a_0 – bit menos significativo, LSB) enquanto 3 pins digitais do microcontrolador, sendo acionados consoante o

estado atribuído ao pin digital (LOW – 0V ou HIGH – Vcc). A rede *R2R* faz com que a contribuição dos bits digitais para a tensão de saída (Vout) seja ponderada, a qual representa uma versão quantizada do sinal que se pretende gerar. O MSB, quando ativado, causa a maior alteração em Vout (contribuição de $\frac{Vcc}{2^1}$ V), com o LSB, quando ativado, a causar a menor alteração em Vout (contribuição de $\frac{Vcc}{2^3}$ V). Dependendo dos bits que são definidos para HIGH ou para LOW, Vout poderá ser quantizado em 8 níveis (2^n, com n = 3), visto cada bit poder representar dois estados. Como o primeiro nível digital diz respeito a VAL = 0 = 000_2, tem-se que o nível digital máximo para n = 3 é VAL = 7 = 111_2. Deste modo, a resolução (step size) é determinada dividindo Vcc pelo número total de níveis (8 níveis), obtendo-se $\Delta V_0 = 5V \times \frac{1}{2^3} = 0.625\ V$, correspondendo a VAL = 1 = 001_2. A voltagem analógica Vout máxima é obtida para VAL = 7 = 111_2 (nível máximo), tendo-se Vout = $\Delta V_0 \times 7 = 4.375\ V$. Assim, as voltagens Vout irão variar entre 0V a 4.375 V, para uma discretização de ΔV_0=$0.625\ V$. A fórmula geral para o cálculo do Vout é dada pela expressão 3, em que os bits a0, a1 e a2 podem assumir os valores de 1 e 0 consoante estejam ativos ou não, respetivamente. Na tabela 1 dispõem-se as voltagens analógicas correspondentes a um dado nível, calculadas a partir da expressão 3, assim como os códigos digitais para um ADC de resolução de 10 bits (expressão 4).

$$\text{Vout} = Vcc \times \left(\frac{a0}{2^3} + \frac{a1}{2^2} + \frac{a2}{2^1}\right) = Vcc \times \frac{VAL}{2^3} \qquad \text{Expressão 3}$$

$$valorADC = \frac{Vout * 2^{10}}{Vcc} \qquad \text{Expressão 4}$$

Tabela 1 – Valores das voltagens analógicas para cada combinação de bits, apresentando-se os códigos digitais correspondentes, obtidos pelo ADC integrado no comando analogRead.

3-bit data - $a_0a_1a_2$/VAL	Vout (V)	Valor ADC
000_2 = 0	0	0
001_2 = 1	0.625	128
010_2 = 2	1.25	256
011_2 = 3	1.875	384
100_2 = 4	2.5	512
101_2 = 5	3.125	640
110_2 = 6	3.75	768
111_2 = 7	4.375	896

4. Com o intuito de criar um *firmware* que gere uma onda com forma *sawtooth* e rampa positiva, recorremos à montagem do circuito descrito no exercício 3 com 2 valores de resistências (R e 2R) no *Seeeduino Nano* e no simulador. O valor de R foi estabelecido a 10k ohm, sendo que 2R foi obtido por 2 resistências R em série, resultando numa rede de 10 resistências de 10K Ohm. A montagem dos circuitos pode visualizar-se nas Figuras 11 e 12. No Anexo III, disponibiliza-se o código desenvolvido para este exercício, incorporando-se os comentários considerados necessários.

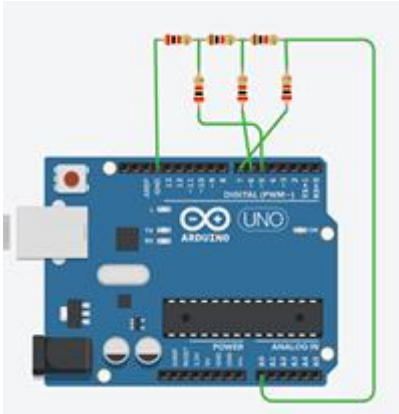

Figura 11 - Montagem no simulador de Arduino do tinkerCad do R2R ladder, para n = 3 bits.

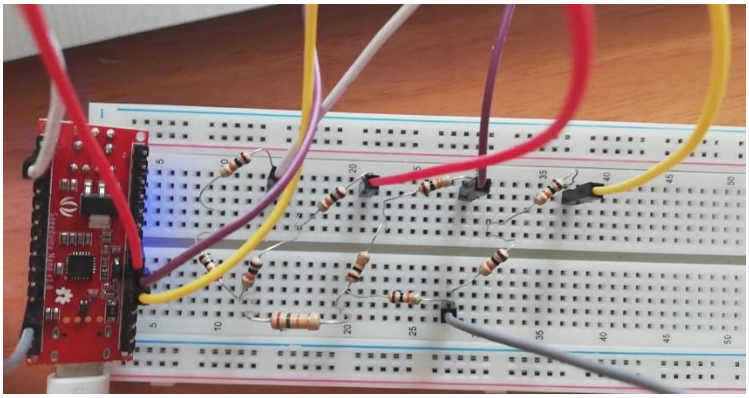

Figura 12 - Montagem no Seeeduino Nano do R2R ladder, com 10 resistências de 10k ohm. Os pins digitais D5, D6 e D7 encontram-se ligados à rede de resistências através dos cabos vermelho, roxo e amarelo, respetivamente. Vout está ligado à entrada analógica A0, através do cabo cinzento. A ligação do ground é estabelecida pelo cabo branco.

A partir da montagem do *R2R ladder* caracterizado no exercício 3, ao controlar-se a combinação de bits pode-se gerar uma onda com forma *sawtooth*, em que para n = 3 bits tem 7 degraus (VAL a assumir valores inteiros entre 0 e 7). Como tal, para um dado VAL, é avaliada a combinação de bits necessária, sendo para isso realizadas operações *bit-wise* para aceder a cada um dos bits que se pretende controlar (aplicam-se as máscaras 001, 010 e 100 através da operação *left-shift* <<=, respetivamente acedendo aos bits a0, a1 e a2), avaliando se os pins digitais correspondentes precisam de ser ativos para produzir a saída VAL desejada. Após isto, Vout ligado à entrada analógica A0 apresenta a voltagem analógica correspondente a VAL, sendo posteriormente amostrado e quantizado. Para a amostragem, tem de se ter em conta o período pré-estabelecido (TW) da onda. Visto que se pretende representar 8 níveis com igual período de duração, tem-se que cada um dos níveis terá um período dado por $\frac{TW}{8}$. Durante o

período de cada nível, a onda é amostrada consoante o período de amostragem (TS) definido, novamente relembrando que para não sofrer *aliasing* ter-se-á de verificar a expressão 1. Experimentalmente, testaram-se os seguintes parâmetros: TW = 100 ms e TS = 1 ms ($TW > 2 \times TS$), permitindo obter uma boa representação da onda pretendida (Figura 13); TW = 100 ms e TS = 63 ms ($TW < 2 \times TS$), obtendo-se uma onda que sofreu *aliasing* (Figura 14).

Analisando a Figura 13, conclui-se que a onda gerada corresponde ao teoricamente pretendido, com a quantização das voltagens analógicas a assumirem valores aproximados dos apresentados na tabela 1 (valor ADC), com as discrepâncias devidas aos erros de quantização inerentes. Quanto ao período de cada nível, apresenta uma duração de $t = \frac{TW}{8} = 12.5\ ms$, pelo que para uma frequência de amostragem de 1000 Hz, cada nível é representado por $N = t \times fs = 12.5$ amostras. No Excel, confirmou-se que, de facto, os níveis apresentam 12 ou 13 amostras.

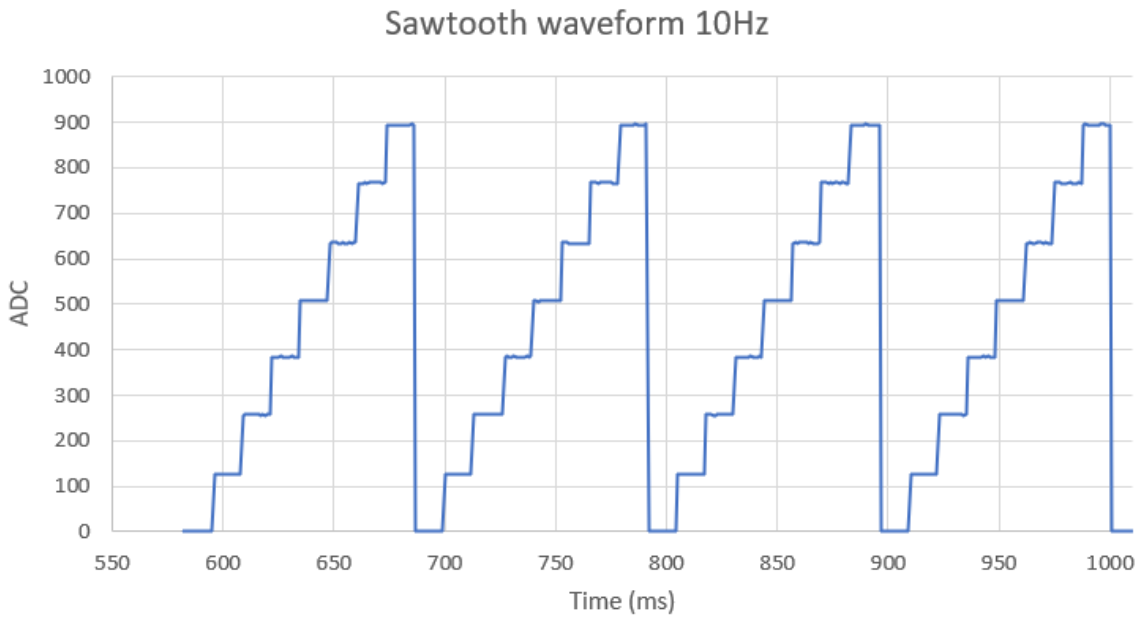

Figura 13 – Onda com forma sawtooth gerada a partir de um R2R ladder com n = 3 bits, com 10 Hz de frequência e amostrada a 1000 Hz.

Quanto à segunda onda gerada, o facto de o teorema de *Nyquist* não ser verificado, faz com que a onda obtida esteja distorcida, refletindo-se novamente o carácter não determinístico da amostragem.

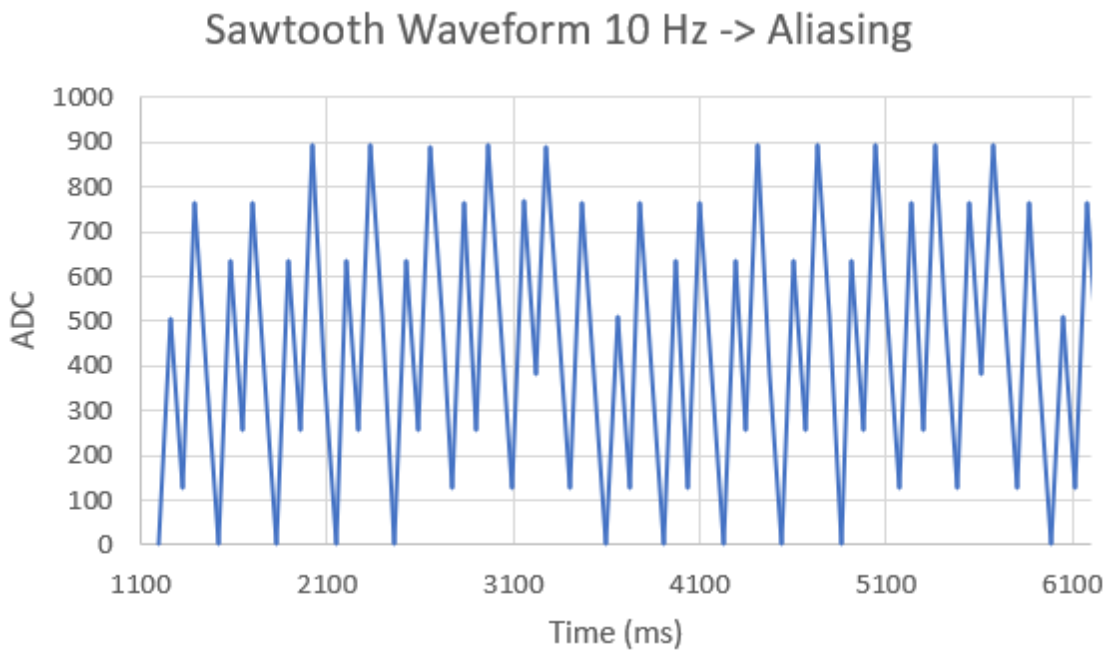

Figura 14 – Onda com forma sawtooth gerada a partir de um R2R ladder com n = 3 bits, com 10 Hz de frequência e amostrada a 63 ms (frequência de amostragem 15.87 Hz).

Anexos

Anexo I – Código para o exercício 1

```
int sensorPin_0 = A0; // selects the analog pin (A0) for transmiting the square wave

bool digitalpin_state = HIGH; // initializes the state of the digital pin
const byte digitalpin = 2; // selects the digital pin (D2)

// Set the wave frequency
const byte frequency = 1; // In Hz

// duty cycle 50%, corresponding to half of the wave period (in ms)
int interval_duty50 = pow(10,3)/(frequency*2.0);

// creates the variable for storing the char coming from the serial port
char incoming_byte;

// defines the sampling period (in ms), for a sampling frequency of 50 Hz
int sampling_period = pow(10,3)/50;

// initializes the system's state, controlling data transmission (0 not transmiting, and 1 for transmiting)
bool system_state = 0;

// initializes the variable to store the time instant associated to the last sample
unsigned long previous_millis = 0L;

// creates the variable for accessing the number of milliseconds passed since the arduino board began
// runnig the current program
unsigned long current_millis;

 // instant that marks the beggining of a specific level
unsigned long t0_level;

// the setup routine runs once when you press reset:
void setup() {
  // configures the digital pin as output, being assigned the operating voltage when set to HIGH state,
  // or 0V when set to LOW state
  pinMode(digitalpin, OUTPUT);
  // initalize serial communication at 115200 bits per second:
  Serial.begin(115200);
}

// the loop routine runs over and over again forever:
void loop() {
  if (Serial.available()>0){
    incoming_byte = Serial.read(); // read the incoming byte

    switch(incoming_byte){
      case 'S': // character for starting streaming
      system_state = 1;

      // assigns the operating voltage since the HIGH state was define as the initial state
      digitalWrite(digitalpin, digitalpin_state);
      break;
```

```
      case 'E': // character for ending streaming
      system_state = 0;
      previous_millis = 0L; // resets
      break;
      }
  }

  if (system_state) // streams sampled and quantitized data
  {
    current_millis = millis();
    t0_level = current_millis;

    // run until not being within the time period of each level
    while (current_millis - t0_level <= interval_duty50){

      // sampling and quantization will be performed
      if (current_millis - previous_millis >= sampling_period){

        Serial.print(current_millis);
        Serial.print(",");

        Serial.println(analogRead(sensorPin_0));

        previous_millis = current_millis;// assigns the time instant associated to the last sample
      }

      current_millis = millis();
    }

    // define the next level
    digitalpin_state = !digitalpin_state;
    digitalWrite(digitalpin, digitalpin_state);
  }
}
```

Anexo II – Código para o exercício 2

```
int sensorPin_0 = A0; // selects the analog pin (A0) for transmiting the square wave, with amplitude [0; Vcc] V
int sensorPin_1 = A1; // selects the analog pin (A1) for transmiting the square wave, with amplitude [0; Vcc/2] V

bool digitalpin_state = HIGH;  // initializes the state of the digital pin
const byte digitalpin = 2;     // selects the digital pin (D2)

// Set the wave frequency
int frequency = 1; // In Hz

// duty cycle 50%, corresponding to half of the wave period (in ms)
int interval_duty50 = pow(10,3)/(frequency*2.0);

// creates the variable for storing the char coming from the serial port
char incoming_byte;

// defines the sampling period (in ms), for a sampling frequency of 50 Hz
int sampling_period = pow(10,3)/50;

// initializes the system's state, controlling data transmission (0 not transmiting, and 1 for transmiting)
bool system_state = 0;

// initializes the variable to store the time instant associated to the last sample
unsigned long previous_millis = 0L;

// creates the variable for accessing the number of milliseconds passed since the arduino board began
// runnig the current program
unsigned long current_millis;

// instant that marks the beggining of a specific level
unsigned long t0_level;

// the setup routine runs once when you press reset:
void setup() {
  // configures the digital pin as output, being assigned the operating voltage when set to HIGH state,
  // or 0V when set to LOW state
  pinMode(digitalpin, OUTPUT);

  // initialize serial communication at 115200 bits per second:
  Serial.begin(115200);
}

// the loop routine runs over and over again forever:
void loop() {
  if (Serial.available()>0){
    incoming_byte = Serial.read(); // read the incoming byte

    switch(incoming_byte){
      case 'S': // character for starting streaming
      system_state = 1;

      // assigns the operating voltage since the HIGH state was define as the initial state
      digitalWrite(digitalpin, digitalpin_state);
      break;

      case 'E': // character for ending streaming
      system_state = 0;
      previous_millis = 0L;  // resets
      break;
```

```
      case 'F': // character to define the frequency wave
      frequency = Serial.parseInt(); // frequency wave in Hz

      // duty cycle 50%, corresponding to half of the wave period (in ms)
      interval_duty50 = pow(10,3)/(frequency*2.0);

      previous_millis = 0L; // resets
      }
  }

  if (system_state) // streams sampled and quantitized data
  {
    current_millis = millis();
    t0_level = current_millis;

    // run until not being within the time period of each level
    while (current_millis - t0_level <= interval_duty50){

      // sampling and quantization will be performed
      if (current_millis - previous_millis >= sampling_period){

        Serial.print(current_millis);
        Serial.print(",");

        Serial.print(analogRead(sensorPin_0));
        Serial.print(",");

        Serial.println(analogRead(sensorPin_1));

        previous_millis = current_millis;// assigns the time instant associated to the last sample
      }

      current_millis = millis();
    }

    // define the next level
    digitalpin_state = !digitalpin_state;
    digitalWrite(digitalpin, digitalpin_state);
  }
}
```

Anexo III – Código para o exercício 4

```
int TS = 1;           // Sampling period
const byte TW = 100;  // Wave period
const byte a0 = 5;    // LSB in (Least significant bit) represented by a0 digital pin
const byte NBITS = 3; // Number of bits

byte data = 0;        // Output value
byte mask = 1;        // Mask for bit-wise operation
byte pin;         // variable for accessing the pins which represent a given bit(3 pins: D5, D6 e D7)
byte maxCode = 1; // initializes the variable which will store the maximum level

int sensorPin = A0; // selects the analog pin (A0) for transmiting the square wave

// creates the variable for accessing the number of milliseconds passed since the arduino board began
// runnig the current program
unsigned long current_millis;

// initializes the variable to store the time instant associated to the last sample
unsigned long previous_millis = 0L;

// instant that marks the beggining of a specific level
unsigned long t0_level;

char incoming_byte; // creates the variable for storing the char coming from the serial port

// initializes the system's state, controlling data transmission (0 not transmiting, and 1 for transmiting)
bool system_state = 0;

// the setup routine runs once when you press reset:
void setup()
{
    // initialize serial communication at 115200 bits per second:
    Serial.begin(115200);

    // cycle for defining the digital pins that we will represent the NBITS
    for (pin = a0; pin < (a0+NBITS); pin++) {

        // configures the digital pin as output, being assigned the operating voltage when set to HIGH state,
        // or 0V when set to LOW state
        pinMode(pin, OUTPUT);

        // updates the number of levels (equivalent to 2^N=8 for N=3)
        maxCode *= 2;
    }
  maxCode = maxCode - 1; // maximum level to be assigned (7 for N=3)
}

// the loop routine runs over and over again forever:
void loop()
{
  if (Serial.available()>0){
    incoming_byte = Serial.read(); // read the incoming byte

    switch(incoming_byte){
      case 'S': // character for starting streaming
      system_state = 1;
      break;

      case 'E': // character for ending streaming
      system_state = 0;
      previous_millis = 0L;
      break;
```

```
      case 'T': // character to define the wave period (in ms)
      TS = Serial.parseInt();
      break;
    }
  }

  if (system_state) // streams sampled and quantitized data
  {
    pin = a0; // set the pin representing the LSB for initial analysis

// for a given data value, it is evaluated the combination of bits required. For this, the data is masked
// with each of the bit positions (starts at bit a0, with the mask 001, performing left-shift operation (<<=) for
// accessing the subsequent bits) that we want to control, evaluating if the corresponding pin needs to be active
// for producing the desired output. When the condition mask > 0 is verified, all bit positions have been evaluated
    for (mask=001; mask>0; mask<<=1) { // Iterate through bit mask
      if (data & mask){
        digitalWrite(pin,HIGH); // activates the digital pin corresponding to a given bit
      }
      else{
        digitalWrite(pin,LOW); // inactivates the digital pin corresponding to a given bit
      }

      pin++; // increments the digital pin for the next mask being applied
    }

    current_millis = millis();
    t0_level = current_millis;

// for the time period of a certain voltage level, the sampling and quantization will be perfomed.
// Note that there exist maxCode + 1 number of levels (8, being 0 to maxCode = 7, for NBITS = 3). For
// a wave with period TW, each level should have a time period of TW/(maxCode + 1)

    while (current_millis - t0_level <= TW/(maxCode + 1)){

      if (current_millis - previous_millis >= TS){
        Serial.print(current_millis);
        Serial.print(",");
        Serial.println(analogRead(sensorPin));
        previous_millis = current_millis;   // assigns the time instant associated to the last sample
      }

      current_millis = millis();
    }

// if the data still haven't reached its maximum level value (7, for NBITS = 3), increment by 1.
// Otherwise, reset to 0, acquiring a new wave

    (data == maxCode ? data = 0 : data++);
  }
}
```