



INSTITUTO SUPERIOR TÉCNICO

MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE  
COMPUTADORES

SISTEMAS ELECTRÓNICOS DE PROCESSAMENTO DE  
SINAL

NCO

&

**Transmissor BPSK**

|                                      |           |
|--------------------------------------|-----------|
| Maria Margarida Dias dos Reis        | n.º 73099 |
| David Gonçalo C. C. de Deus Oliveira | n.º 73722 |
| Nuno Miguel Rodrigues Machado        | n.º 74236 |

Grupo n.º 5 de segunda-feira das 15h30 - 18h30

Lisboa, 17 de Abril de 2015

# Índice

|          |                                       |          |
|----------|---------------------------------------|----------|
| <b>1</b> | <b>Introdução</b>                     | <b>1</b> |
| <b>2</b> | <b>Projecto #1 - NCO</b>              | <b>1</b> |
| <b>3</b> | <b>Projecto #2 - Transmissor BPSK</b> | <b>6</b> |
| <b>4</b> | <b>Conclusões</b>                     | <b>7</b> |

# 1 Introdução

Com este trabalho laboratorial o objectivo é a familiarização com o sistema de desenvolvimento de *software* e *kit* de processamento digital de sinal DSK TMS320C6713. O processador em causa é de 32 *bits*, com um relógio de 225 MHz, sendo capaz de fazer o *fetching* e execução de 8 instruções por ciclo de relógio. Relativamente ao *software*, a ferramenta utilizada para programar o DSK é o CCS v5.5.

Na primeira fase do projecto pretende-se implementar um oscilador numericamente controlado (NCO) e de seguida um transmissor *binary phase-shift keying* (BPSK).

## 2 Projecto #1 - NCO

Um oscilador numericamente controlado permite gerar uma frequência instantânea proporcional ao sinal de entrada. É um gerador digital de sinal que cria uma representação síncrona, discreta no tempo e discreta em amplitude de uma forma de onda.

As características do NCO são apresentadas na seguinte tabela.

Tabela 1: Características do NCO.

| Parâmetro                | Símbolo   | Valor  | Descrição   |
|--------------------------|-----------|--------|---|
| frequência de amostragem | $f_s$     | 16 kHz |   |
| frequência mínima        | $f_{min}$ | 2 kHz  | frequência a que a amplitude do sinal de entrada é mínima |
| frequência máxima        | $f_{max}$ | 6 kHz  | frequência a que a amplitude do sinal de entrada é máxima |

### 2.1

Pretende-se primeiramente desenvolver um oscilador de relaxação utilizando uma variável inteira com sinal de 16 *bits* e a circularidade da representação em complemento para dois. Na figura abaixo encontra-se uma representação do sinal a obter.

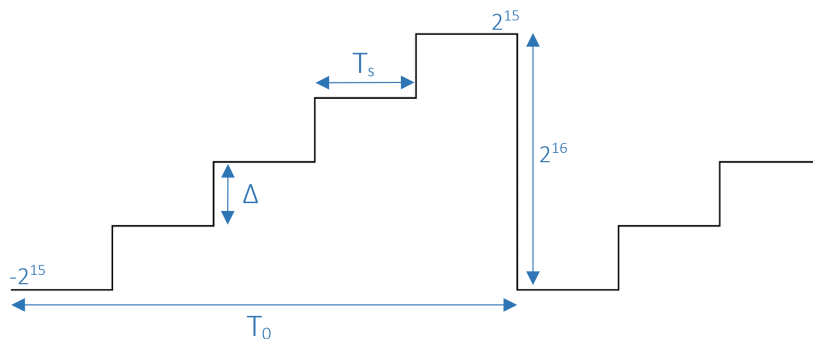


Figura 1: Esquema do oscilador de relaxação.

Com recurso à Figura 1 pode-se deduzir que

$$f_0 = \frac{\Delta}{2^{16}} \times f_s \leftrightarrow \Delta = \frac{f_0}{f_s} \times 2^{16}. \quad (2.1)$$

Existe uma variável de estado da rampa que a cada  $T_s$ , período de amostragem, é incrementada de  $\Delta$ , como se pode ver na Figura 1. A variável de estado da rampa é de 16 *bits* com representação em  $Q_{15}$  e, sabendo que o maior número positivo que se pode representar em  $Q_{15}$  é  $2^{15} - 1 = 32767$  e o menor número negativo que se pode representar é  $-(2^{15} - 1) = -32767$ , a variável de estado começa com o valor -32767 e vai até um máximo de 32767. Quando é atingido o valor máximo, 32767, entra em efeito a circularidade da representação em complemento para dois e, assim, a variável de estado não atinge o valor de  $2^{15}$ , “dando a volta” para -32767.

Relativamente à variável  $\Delta$  esta encontra-se também representada em  $Q_{15}$ . O NCO tem como característica uma frequência  $f_0$  que varia entre 2 kHz e 6 kHz. Estes valores são controlados a partir da amplitude do sinal de entrada. Quando esta for mínima, a frequência  $f_0$  é de 2 kHz e quando for máxima, a frequência  $f_0$  é de 6 kHz. Com estas especificações pode-se calcular três valores de  $\Delta$  com recurso à equação (2.1), para a frequência mínima, a frequência média e a frequência máxima.

Tabela 2: Valores de  $\Delta$  para as três frequências especificadas.

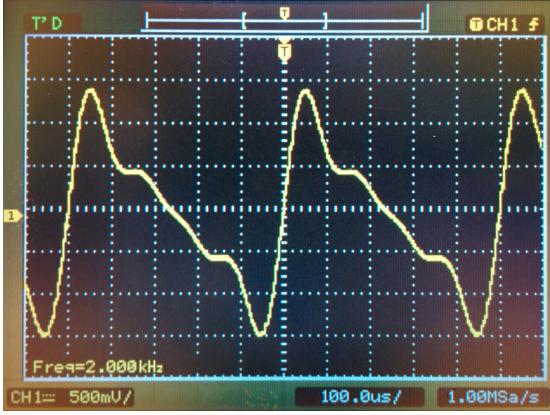
| $f_0$ | $\Delta$ |
|-------|----------|
| 2 kHz | 8192     |
| 4 kHz | 16384    |
| 6 kHz | 24576    |

Em código, a variável de estado da rampa é **status** e a variável que representa os incrementos é **delta**. No código abaixo está a criação da rampa para um frequência de 4 kHz.

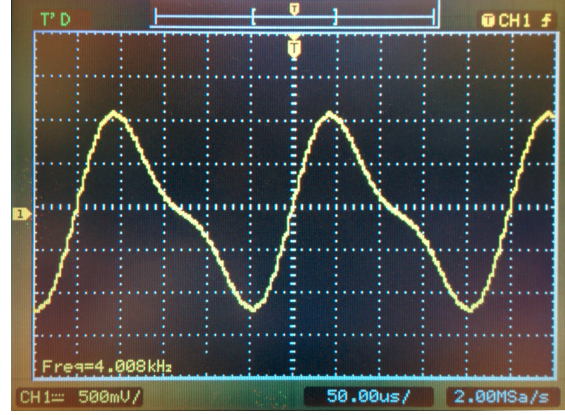
```

1 void main() {
2
3     short delta = 16384;
4     short status = -32767;
5
6     while(1){
7         ...
8         //criacao da rampa
9         status = status + delta;
10        ...
11    }
12 }
```

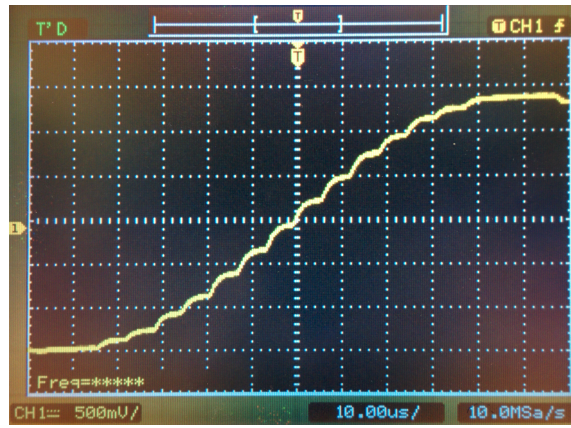
Nas figuras da próxima página pode-se ver o sinal obtido experimentalmente que representa a rampa para dois valores diferentes de frequência  $f_0$ .



(a)



(b)



(c)

Figura 2: Oscilador de relaxação para  $f_0 = 2$  kHz (a), oscilador de relaxação para  $f_0 = 4$  kHz (b) e pormenor da rampa criada (c).

Como se pode ver na Figura 2(c), por comparação com o esperado teoricamente da Figura 1, o oscilador de relaxação implementado funciona de acordo com o previsto.

## 2.2

De forma a criar a *look-up-table* (LUT) com 32 valores positivos de meio período da função seno, é necessário começar por determinar esses valores, para que, posteriormente, os mesmos sejam convertidos para o formato mais preciso de representação,  $Q_{15}$ , uma vez que se encontram no intervalo  $[-1, 1]$ . Assim, tendo em conta que meio período da função seno é  $\pi$ , podemos calcular os valores da seguinte maneira:

$$a_k = \sin\left(\frac{\pi}{32}k\right), k = 0, 1, \dots, 32. \quad (2.2)$$

Os 32 valores determinados são então convertidos para o formato  $Q_{15}$ , recorrendo a:

$$a_{k_{15}} = \text{round}\left(a_k \times 2^{15}\right), \quad (2.3)$$

sendo assim criada a LUT pretendida.

Apresenta-se de seguida o excerto de código onde é declarada a LUT com os valores de meio período da função seno, no vector `sine` que tem 33 posições, cada uma de 16 *bits*.

```

1  ...
2  //LUT do seno
3  short sine[33] =
    {0,3212,6393,9512,12540,15447,18205,20788,23170,25330,27246,28899,30274,31357,
    32138,32610,32767,32610,32138,31357,30274,28899,27246,25330,23170,20788,18205,
    15447,12540,9512,6393,3212,0};
4  ...

```

se calhar  
explicar  
porquê 33  
valores e  
nao os 32  
pedidos -  
david

## 2.3

Para que se possa aceder aos valores da função seno, é utilizada a variável de estado do oscilador, `status`, como índice da LUT. Apenas 5 *bits* da variável são utilizados para endereçar a LUT, sendo criada a variável `i`, tal como especificado na Figura 3, onde a variável de estado do oscilador é representada por `x`.

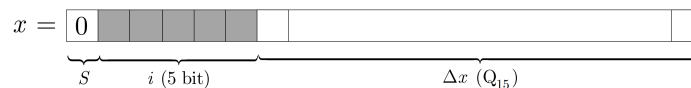


Figura 3: Representação da variável do estado do oscilador.

De modo a aceder aos 5 *bits* pretendidos da variável de estado, é realizado um deslocamento de 10 *bits* para a direita, sendo, de seguida, utilizada a função lógica AND com a máscara 31 (5 *bits* menos significativos com o valor lógico 1). É apresentado o excerto de código que realiza o procedimento especificado.

```

1  ...
2  //indexar a LUT e obter os valores do seno
3  i = status >> 10;
4  i = i & 31;
5  y1 = sine[i];
6  ...

```

## 2.4 e 2.5

Foram criadas duas variáveis com o objectivo de controlar a amplitude e frequência do sinal sinusoidal. A variável `delta` representa o controlo da frequência e a variável `amp` representa o controlo da amplitude. O código que permite implementar este controlo é apresentado de seguida.

```

1 void main(){
2  ...
3  //variavel de controlo de frequencia
4  short delta = 0
5  //variavel de controlo da amplitude: define um ganho de 1/2
6  short amp = 16384;

```

```

7  short yf = 0;
8  ...
9  while(1){
10     if(intflag != FALSE){
11         ...
12         //obtencao do valor para a frequencia
13         delta = 16384 + (inbuf>>2);
14         ...
15         //controlo da amplitude e frequencia
16         yf = (y1*delta<<1)>>16);
17         y = (yf*amp<<1)>>16);
18
19         if(status < 0)
20             y = -y;
21
22         AIC_buffer.channel[LEFT] = y;
23     }
24 }
25 }

```

Analisando a Tabela 2 verifica-se que o valor de **delta** oscila com uma amplitude de 8192 em torno de 16384,  $\Delta_0$ . Ou seja,  $f_0$  tem uma frequência central em 4 kHz, oscilando com uma amplitude de 2 kHz. O incremento do oscilador é obtido de acordo com a seguinte equação, onde  $x$  é a amplitude do sinal de entrada:

$$\Delta = \Delta_0 + kx. \quad (2.4)$$

Com esta conclusão, teve de se garantir que o valor da amplitude do sinal de entrada não ultrapassa 8192, mantendo a relação entre cada amostra. Optou-se por dividir o valor de cada amostra por 4,  $k = 1/4$ , pois a amplitude máxima é de 32767, o equivalente a um *shift* de 2 *bits* para a direita.

Em baixo está o código referente ao cálculo para obter o valor de **delta**, sendo que todas as variáveis definidas neste excerto são de 16 *bits*, **short**, em formato  $Q_{15}$ .

```

1  ...
2  //obtencao do valor para a frequencia
3  delta = 16384 + (inbuf>>2);
4  ...

```

Tendo o valor de **delta**, é simples obter a amplitude de cada amostra do sinal de saída, multiplicando **delta** por **y1**, valor obtido da LUT referente à questão 2.2. Em baixo está representado um excerto do código que demonstra a obtenção da amplitude do sinal de saída. Todas as variáveis são de 16 *bits*, tendo **y1** e **delta** o formato de  $Q_{15}$ , como também **yf**. Isto deve-se ao facto de o formato do resultado da multiplicação com duas variáveis em  $Q_{15}$  ser  $Q_{30}$  com replicação do *bit* de sinal. Assim, é necessário efectuar um *shift* para a esquerda para remover o *bit* de sinal replicado, resultando num formato final de  $Q_{31}$ , para 32 bits. Para se poder armazenar numa variável de 16 *bits*, no formato  $Q_{15}$ , é necessário efectuar um *shift* de 16 posições para a direita, permitindo armazenar os 16 *bits* mais

significativos do resultado de 32 *bits*.

O código apresentado de seguida demonstra a explicação referida.

```
1 ...  
2 //controlo da amplitude e frequencia  
3 yf = (y1*delta<<1)>>16);  
4 ...
```

Para o controlo da amplitude do sinal de saída, multiplica-se o resultado final obtido anteriormente por uma constante de 16 *bits* em formato  $Q_{15}$ . Está representado um excerto de código que demonstra a alteração da amplitude do sinal de saída. Neste caso todas as variáveis são também de 16 *bits*, tendo *yf* e *amp* o formato de  $Q_{15}$ , como também *y*. Para armazenar a variável *y* em  $Q_{15}$  recorre-se à mesma lógica explicada anteriormente de fazer 15 *shifts* para a direita ao resultado da multiplicação.

O código apresentado de seguida demonstra a explicação referida.

```
1 ...  
2 //controlo da amplitude e frequencia  
3 y = (y_f*amp<<1)>>16);  
4 ...
```

## 2.6

teddy

## 2.7

Pretende-se agora melhorar a qualidade do oscilador sinusoidal utilizando interpolação linear. Esta interpolação é feita lendo dois valores consecutivos,  $y_1$  e  $y_2$ , da LUT do seno e depois obter o valor sinusoidal interpolado com recurso à seguinte equação

$$y = y_1 + (y_2 - y_1) \quad (2.5)$$

Na Figura 3, onde se encontra representada a variável de estado da rampa, pode-se ver que os 10 *bits* menos significativos

margarida

## 2.8

margarida

## 3 Projecto #2 - Transmissor BPSK

Neste projecto, pretende-se implementar o codificador de um transmissor BPSK (Binary Phase-shift Keying), representado na Figura X.



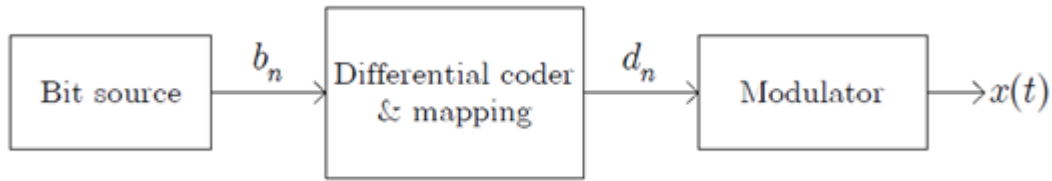


Figura 4: Esquema de um transmissor BPSK.

O codificador tem como entrada os bits  $b_n$ , que se trata de uma sequência que vai alternando entre o valor lógico '0' e o valor lógico '1' ( $b_n = 1, 0, 1, 0, \dots$ ) e como saída os valores  $d_n$ , que podem ser  $-1$  ou  $+1$ . O codificador realiza a operação  $c_n = c_{n-1} \oplus b_n$ , considerando  $c_0 = '0'$ , para que depois possam ser determinados a sequência  $d_n$ , de acordo com o mapeamento:

$$c_n = '0' \rightarrow d_n = +1; c_n = '1' \rightarrow d_n = -1; \quad (3.1)$$

Assim, o sinal modulado BPSK é dado por:

$$s(t) = \sin(2\pi f_0 t + \pi c_n) = d_n \sin(2\pi f_0 t) \quad (3.2)$$

Ou, usando tempo discreto:

$$s_n = s(nT_s) = d_n \sin(2\pi f_0 T_s n) \quad (3.3)$$

É utilizada uma frequência de amostragem,  $f_s = 1/T_s$ , de  $16 \text{ kHz}$  e uma frequência da portadora,  $f_0$ , de  $4 \text{ kHz}$ . A taxa de bits é de  $f_b = 1 \text{ kbps}$ , por isso por cada bit, há 4 períodos da portadora.

### 3.1

Os bits da sequência  $b_n$  são implementados recorrendo a um contador, uma vez que a cada 16 amostras do sinal de entrada, é necessário que haja uma alteração do bit seguinte da sequência  $b_n$ , passando de '0' para '1' ou vice-versa. Para que seja realizada essa alteração, é utilizada a função XOR do bit  $b_n$  anterior com '1'. É assim apresentado o excerto de código que implementa a sequência de bits  $b_n$ .

```

1  ...
2  if(b_i>15){
3      b_i=0;
4      b_n=(b_n^1);
5      ...
6  }
7  b_i++;

```

Foi criada outra solução para a obtenção da sequência  $b_n$  com objectivo de não utilizar a instrução condicional, `if`. Seguiu-se o conselho do enunciado de usar um contador que quando ocorre o *overflow* gera um novo *bit* alternado. Está representado de seguida o excerto de código:

```

1  ...
2  b_i++;

```

```
3  b=(b_i&16)>>4; // mascara para obter o bit de overflow
4  b_n=(b_n^b); // xor para alternar o bit bn
5  b_i=(b_i&15); // obtencao dos 4 bits menos significativos
6  ...
```

Como se pode observar, a ocorrência de *overflow* acontece quando a variável, `b_i` que é incrementada a cada ciclo, atinja o valor 16. Para detectar esta ocorrência aplica-se a máscara 16 (o quinto *bit* menos significativo com o valor lógico 1) e efectua-se um *shift* 4 posições para a direita. De seguida, aplica-se a função XOR com o resultado anterior com a variável `b_n` de forma a obter uma sequência  $b_n$  que varia alternadamente entre '0' e '1' ou vice-versa a uma taxa de  $f_b = 1 \text{ kbps}$ .

## 3.2

teddy

## 3.3

margarida

## 4 Conclusões