ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

PEA3411 – Introdução à Automação e Tópicos Embarcados para Sistemas Elétricos

Prof. Eduardo Pellini

Christian P. Braga	11805850
Fernando C. Chaim	11200608
Thiago M. Curto	11833671

Algoritmo de Deglitch e Debounce de Entradas Digitais
S07

São Paulo

2023

1. Explicação do algoritmo

A) Debouncing

O Código é composto por dois arquivos diferentes: '*debounce_and_register_events.m*' e '*Debounce.m*'.

O Primeiro, '*debounce_and_register_events.m*' consiste em uma função que recebe três parâmetros - Os dados vetorizados da saída digital do relé comum (parâmetro "data", o número mínimo de bits que o código passará a reconhecer como uma mudança real no nível do sinal (parâmetro "duration"), e o período de amostragem dos dados coletados pelo relé - e realiza o debounce dos arquivos, bem como o registro dos eventos.

Antes de apresentar como a função funciona, é relevante discutir o que representa o parâmetro "duration". Esse parâmetro é calculado na função '*Debounce.m*' e corresponde ao arredondamento em número inteiro superior da razão entre o tempo de debounce (que é definido pelo usuário) pela taxa de amostragem, resultando na sensibilidade do algoritmo em relação aos 'bounces' do sinal coletado. Ou seja, se há uma mudança nas sequência de bits coletados, em que o sinal muda de valor por um período maior que "duration bits", o algoritmo passa a considerar que isso não se trata de um 'bounce', mas sim de um dado que é real.

Uma explicação detalhada do da função '*debounce_and_register_events.m*' está explicitada logo abaixo:

"De início, o código extrai o tamanho do vetor de saída digital - *'data'* e cria os vetores *'buffer'* e *'debounced_data'* do mesmo tamanho do vetor de Dados Digitais, porém estes são compostos apenas por zeros.

Após isso, através da função *for* são realizadas iterações sucessivas de 1 até o tamanho máximo desse vetor e nessas iterações, é realizada uma comparação entre o valor do vetor data e o valor do vetor buffer na posição i. Caso os valores sejam diferentes, a variável counter é acrescida de uma unidade e caso o contrário, ela é zerada. Assim, surgem três opções:

a) **Se** o resultado do **contador for maior ou igual que a duração** definida inicialmente na função, o vetor *'debounced_data'* armazenará na posição i, o valor de data também na posição i;
b) **Se** o resultado do **contador for igual a zero**, o vetor *'debounced_data'* também armazenará na posição i, o valor de data(i);

c) Por último, **se nenhuma dessas opções** for contemplada, *'debounced_data'* armazenará em i, o valor anterior de *'debounced_data'*, isto é, *'debounced_data(i-1)'*.

Ao final dessa sequência de condições (elseifs), registra-se na posição i-ésima do buffer o valor de data em i.

Para encerrar o laço for, os eventos devem ser registrados: Diante disso, **se i for maior que 1** e o i-ésimo termo de *debounced_data* for diferente do i-ésimo termo - 1 desse mesmo vetor, o **tipo do evento (event_type)** será registrado como debounced_data de i, o **tempo do evento (event_time)** é registrado como **(i - 1) multiplicado pelo Período de Amostragem** e o registro final do evento: **event_log** será um vetor que contém event_log (já existente), event_time e event_type. Dessa forma, será possível ao final de i-ésimas iterações compor um vetor de event_log com várias posições. Nesse sentido, é avaliado se ocorreu uma mudança de valor entre dois valores de debounced_data seguidos e se houve, isso **caracteriza um evento** que deve ser **registrado pelo algoritmo**. Com isso, o laço for é encerrado e consequentemente o arquivo.**"**

Além do arquivo '*debounce_and_register_events.m*', o processo de debouncing conta com o arquivo '*Debounce.m*', que funciona basicamente como um arquivo principal (main) que realiza os pedidos ao usuário, bem como a apresentação dos resultados para ele.

Primeiramente, o arquivo estabelece o valor do período de amostragem conforme é pedido no enunciado (4 μs), pede que o usuário faça o input do valor do tempo de debounce (que precisa ser maior que 10 μs) e calcula o parâmetro "duration" (cálculo e significado explicados no terceiro parágrafo). Após isso, ele realiza a leitura do arquivo de dados escolhido (no caso, utilizamos o arquivo "Data2.m" para fazer um estudo sobre os sinais e o arquivo "Data3.m" para testar, de fato, o algoritmo) e abre o arquivo '*debounce_and_register_events.m*' para realizar o debouncing.

Com os resultados do vetor com os dados após o debouncing e a matriz que apresenta o tipo do evento (se o sinal foi de 1 para 0 ou de 0 para 1) e o tempo em que este ocorreu, o arquivo gera um gráfico que apresenta os sinais originais (sem debounce) e os sinais após o debounce, bem como realiza impressões no prompt de comando do usuário que descrevem todos os eventos que ocorreram em uma linguagem mais entendível e o tempo em que estes ocorreram.

2. Resultados

A) Conjunto de dados 2

Primeiramente, realizamos a testagem do código para o conjunto de dados "Dados2.m", para um tempo de debounce de 11 µs e obtivemos os seguintes resultados:

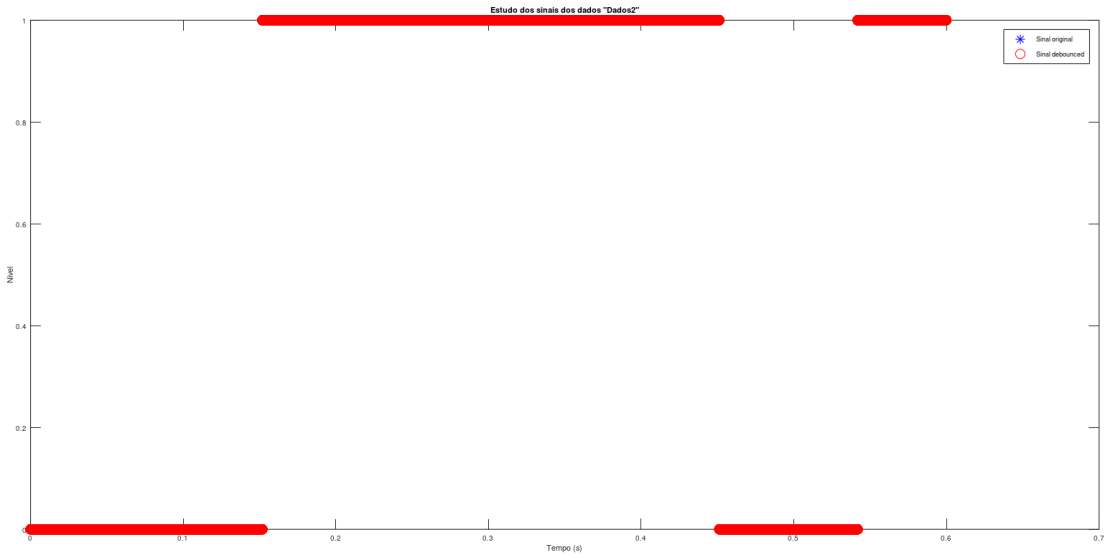

Figura 1 - Gráfico para a amostra de dados2

Dando um zoom em dois períodos em que ocorrem eventos pode-se ver o funcionamento do algoritmo:

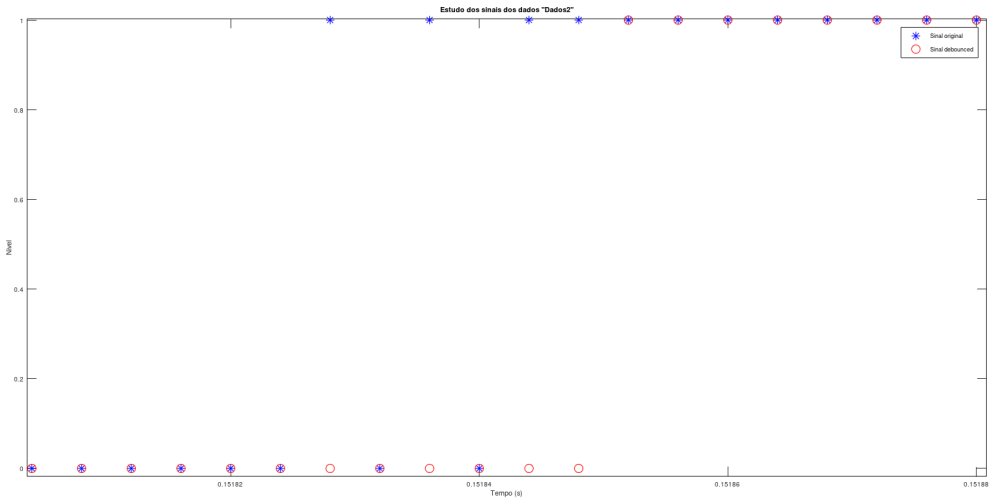

Figura 2 - Ampliação do primeiro evento

Como pode-se observar, o algoritmo inibiu dois bounces do sinal e só passou a considerar que o sinal havia mudado após 4 bits seguidos (número de bits maior que

duration).

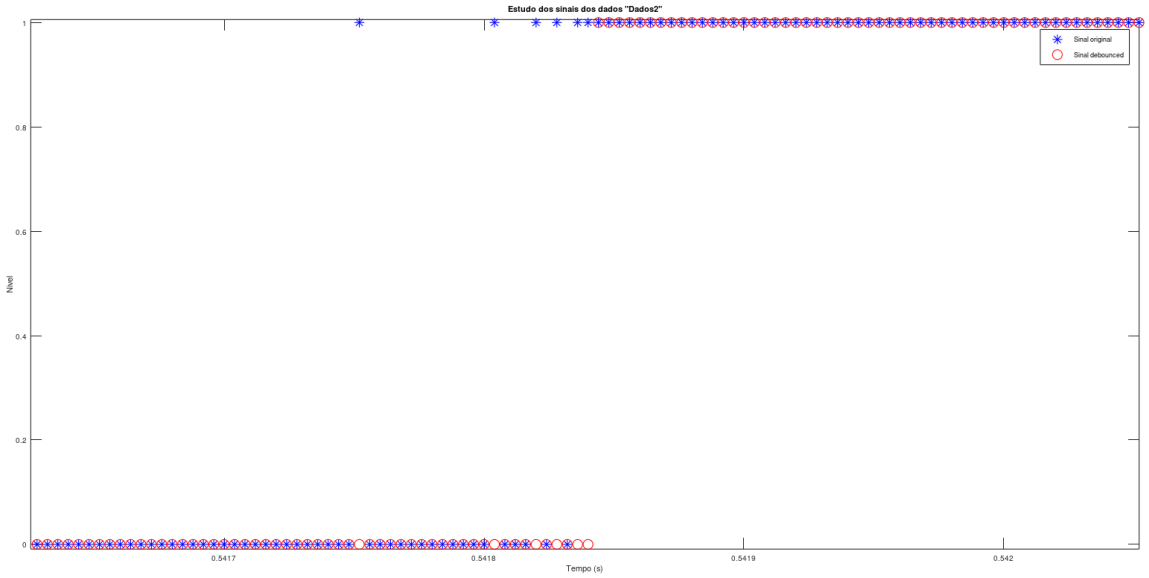

Figura 3 - Ampliação do terceiro evento

Como pode-se observar, o algoritmo inibiu dois bounces do sinal e só passou a considerar que o sinal havia mudado após 4 bits seguidos (número de bits maior que duration).

Ou seja, para o conjunto de dados 2, o debouncing foi um sucesso.

Além do gráfico, o algoritmo fez um registro de todos os eventos, evidenciando o tempo em que ocorreram e bem como o seu tipo.

```
Debouncing dos arquivos de dados 3: O tempo de ocorrência do evento 1 foi de: 0.151852 segundo
O nivel do sinal foi de 0 para 1


O tempo de ocorrência do evento 2 foi de: 0.451200 segundo
O nivel do sinal foi de 1 para 0


O tempo de ocorrência do evento 3 foi de: 0.541844 segundo
O nivel do sinal foi de 0 para 1
```

Figura 4 - Registro dos eventos para o conjunto de dados 2

B) Conjunto de dados 3

Após a testagem do código no conjunto de dados 2, realizamos a testagem do código para o conjunto de dados “Dados3.m”, para um tempo de debounce de 16 µs (para verificar se alterar o tempo de debounce alterava o duration) e obtivemos os seguintes resultados:

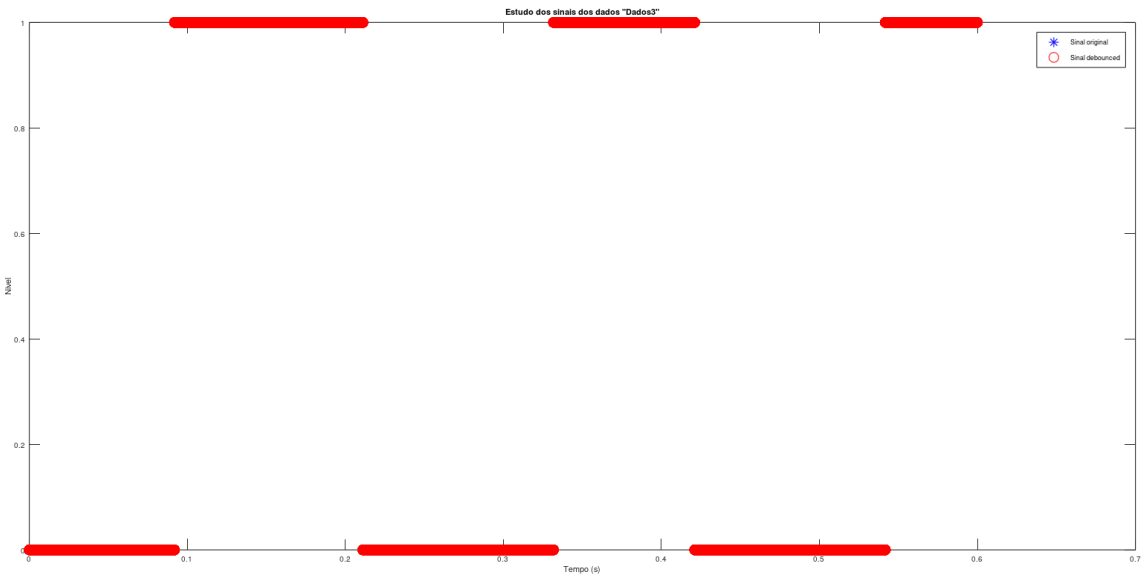

Figura 5 - Gráfico para a amostra de dados 3

Dando um zoom em dois períodos em que ocorrem eventos pode-se ver o funcionamento do algoritmo:

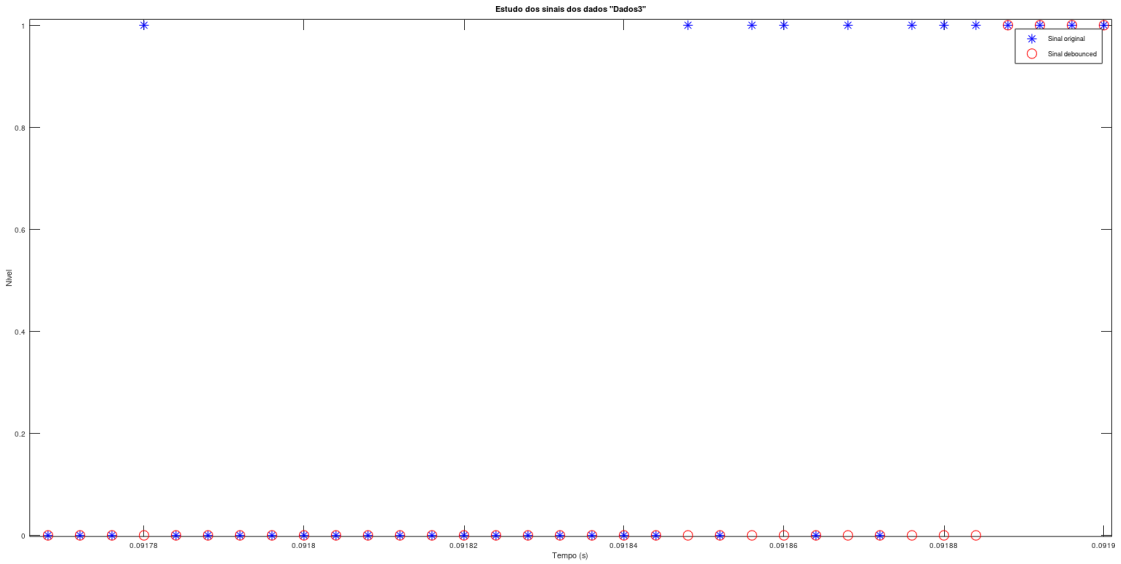

Figura 6 - Ampliação do primeiro evento

Como pode-se observar, o algoritmo inibiu cinco bounces do sinal e só passou a considerar que o sinal havia mudado após 5 bits seguidos (número de bits maior que duration, o que comprova que, alterar o tempo de debounce realmente altera o valor de duration).

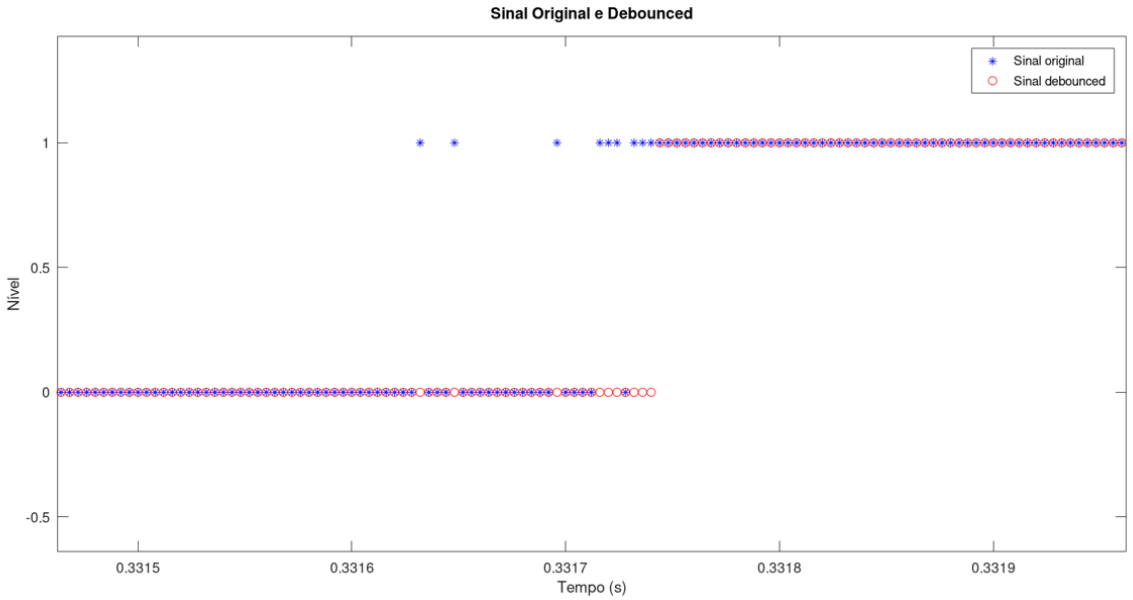

Figura 7 - Ampliação do terceiro evento

Como pode-se observar, o algoritmo inibiu sete bounces do sinal e só passou a considerar que o sinal havia mudado após 5 bits seguidos (número de bits maior que duration).

Ou seja, para o conjunto de dados 3, o debouncing foi um sucesso.

Além do gráfico, o algoritmo fez um registro de todos os eventos, evidenciando o tempo em que ocorreram e bem como o seu tipo.

```
Debouncing dos arquivos de dados 3: O tempo de ocorrência do evento 1 foi de: 0.091888 segundo
O nivel do sinal foi de 0 para 1


O tempo de ocorrência do evento 2 foi de: 0.211100 segundo
O nivel do sinal foi de 1 para 0


O tempo de ocorrência do evento 3 foi de: 0.331744 segundo
O nivel do sinal foi de 0 para 1


O tempo de ocorrência do evento 4 foi de: 0.421100 segundo
O nivel do sinal foi de 1 para 0


O tempo de ocorrência do evento 5 foi de: 0.541744 segundo
O nivel do sinal foi de 0 para 1
```

Figura 8 - Registro dos eventos para o conjunto de dados 3