Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática
Laboratório de Sistemas Digitais

Bounce na Comutação de Contactos Mecânicos – Problema e Possíveis Soluções

1. Introdução

A comutação de contactos mecânicos presentes em interruptores, botões de pressão, relés e outros dispositivos similares apresenta *bounce*, isto é, a ocorrências de múltiplas transições espúrias (indesejadas) quando se realiza uma comutação OFF->ON ou ON->OFF. A figura 1 apresenta um exemplo de *bounce* capturado com um osciloscópio durante uma comutação ON->OFF. No exemplo apresentado, em vez de ocorrer apenas uma transição, ocorrem sete transições. De notar que o número de transições e a largura dos impulsos (*glitches*) produzidos não é em geral fixo, nem previsível, mas normalmente ocorrem numa janela temporal de dezenas a centenas de micro-segundos. Em algumas aplicações estas múltiplas transições não são problemáticas. No entanto, há casos (e.g. quando se pretende usar este sinal como entrada de relógio, i.e. *clock* de um circuito, ou como *enable* de componentes com sinais de *clock* "rápidos") em que estas transições possuem um efeito indesejado (e.g. provocando múltiplos incrementos incorretos de um contador ou transições incorretas de uma máquina de estados finitos).

No caso dos *kits* DE2-115 usados nas aulas práticas, o problema do *bouncing* pode manifestar-se quando se usam os interruptores (SW) ou os botões de pressão (KEY) como dispositivos de entrada do sistema. Apesar do problema se manifestar muito mais nos interruptores do que nos botões de pressão, a verdade é que o circuito colocado pelo fabricante do *kit* nas entradas da FPGA ligadas aos botões de pressão, com a intenção de eliminar o *bouncing*, reduz o problema, mas não o resolve completamente. Assim, é necessário tomar as devidas precauções quer no caso dos interruptores, quer no caso dos botões.

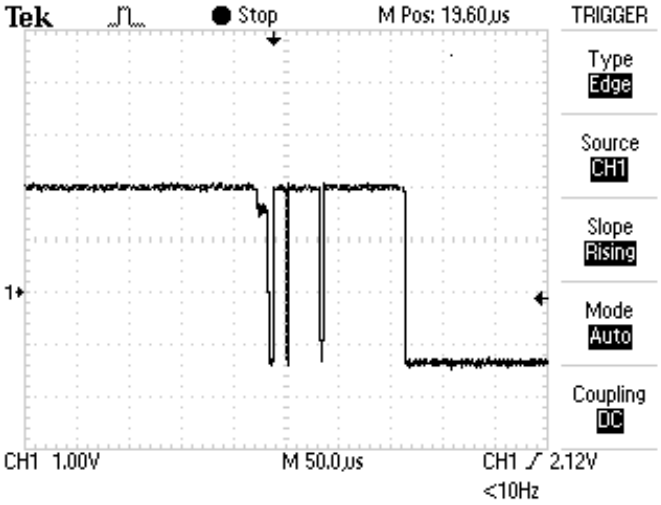

Figura 1 – Exemplo de *bounce* na transição de ON->OFF.

Em circuitos que operem a frequências do sinal de relógio muito baixas (<100Hz) basta amostrar a este ritmo as entradas provenientes de contactos mecânicos para que o *bounce* não se manifeste. No entanto, noutras situações são necessários circuitos mais elaborados, como os descritos nas secções seguintes.

2. Circuito Elementar de *Debounce*

Existem várias formas de reduzir a probabilidade da ocorrência de *bounce*, a maior parte baseadas em circuitos que se adicionam entre a saída do contacto mecânico e a entrada do sistema, para produzir um impulso "limpo" (livre de *bounce*). A figura 2 ilustra uma possível arquitetura de um circuito de *debounce* baseado num *shift register*. Os sinais de *reset* dos *flip-flops* que constituem o *shift-register* são assíncronos e ativos a '1'. A entrada *dirtyIn_n* destina-se a ligar ao contacto de entrada (que apresenta *bounce*). A saída *cleanOut* disponibiliza um impulso filtrado (*debounced*). Além destes dois portos, existe ainda um sinal de entrada de *clock* de referência responsável por efetuar o deslocamento da informação no *shift register*.

O circuito apresentado na figura 2 pode ser usado juntamente com os botões de pressão do *kit* DE2-115, pelo que se assume que a entrada *dirtyIn_n* está em lógica negativa (ativa baixa). O impulso filtrado ("limpo") produzido na saída *cleanOut* está em lógica positiva. O mesmo tipo de circuito pode ser usado com os interruptores do *kit* DE2-115 se os *flip-flops* possuírem um sinal de *reset* ativo a '0'.

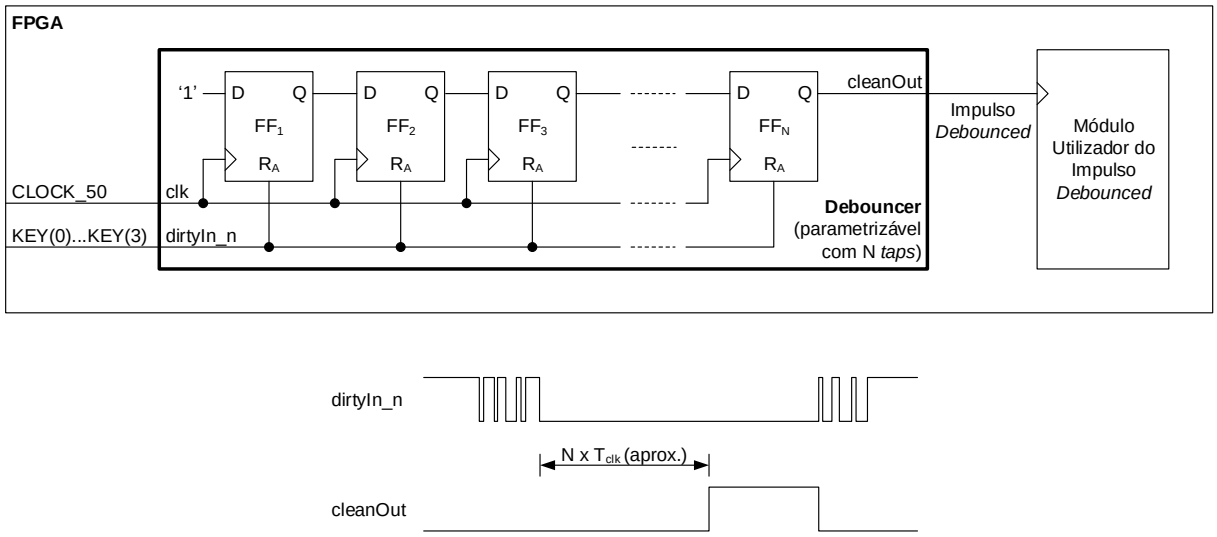

Figura 2 – Arquitetura de um circuito de *debounce* baseado num *shift register* e utilização no contexto de um projeto para o *kit* DE2-115.

Tarefas a realizar

- Analise o módulo "*Debouncer*" apresentado na figura 2 e modele-o em VHDL.
- Utilize o módulo "*Debouncer*" no contexto de um projeto dos guiões 4 ou 5 de LSDig onde observou a ocorrência de *bounce*.
- Teste o sistema resultante para vários valores de N (e.g. 20, 200 e 2000) e avalie o seu comportamento.

3. Implementação Alternativa

No seguinte endereço é disponibilizada uma arquitetura de um circuito de *debounce* potencialmente mais "poupado" em termos de recursos de implementação:
https://www.eewiki.net/display/LOGIC/Debounce+Logic+Circuit+%28with+VHDL+example%29

Por conveniência é replicada neste documento a arquitetura proposta (figura 3). Nesta arquitetura o número de bits do contador pode ser parametrizável. Interprete o circuito apresentado, modele-o em VHDL e use-o no contexto de um projeto.

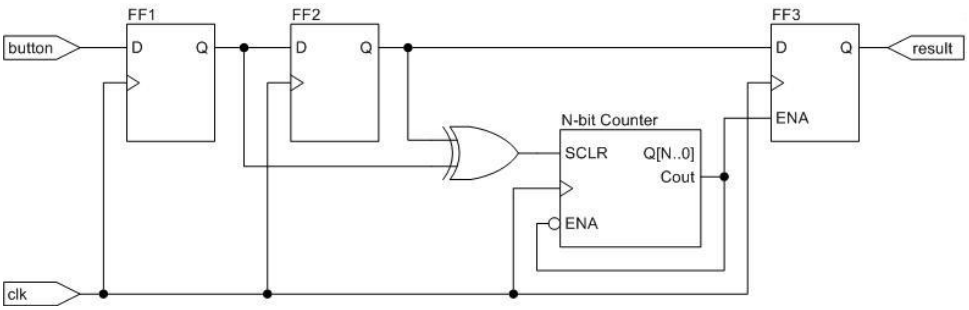

Figura 3 - Arquitetura alternativa para o circuito de *debounce* (a saída C_{out} transita para '1' quando o contador atinge o final de contagem; a porta lógica XOR é usada para detetar transições na entrada).

4. Implementação Configurável

Os circuitos anteriores filtram o *bounce* que pode ocorrer na entrada, produzindo na saída um impulso "limpo", atrasado relativamente à transição de entrada e com uma duração que depende do tempo que a entrada permanece ativa. No entanto, por vezes é necessário produzir um impulso livre de *bounce* mas com uma duração fixa, bem definida (tipicamente um período do sinal de relógio de referência) e que portanto não dependa da duração da ativação da entrada. Neste caso a saída do circuito de *debounce* pode também ser usada como *enable* (em vez de *clock*), permitindo utilizar o mesmo sinal de relógio em todos os componentes, mas cada um com o seu sinal de *enable* individual [1].

O código VHDL da figura 5 descreve um d*ebouncer* parametrizável, cujo comportamento temporal é ilustrado na figura 4. A operação deste *debouncer* pode ser descrita sucintamente da seguinte forma:

- A polaridade do sinal de entrada e do impulso de saída são parametrizáveis/configuráveis estaticamente aquando da instanciação do módulo.
- Sempre que a entrada **`dirtyIn`** é ativada, a saída é inativada e é iniciado um contador com o valor **`kHzClkFreq * mSecMinInWidth`**.
- Se a entrada **`dirtyIn`** permanecer ativa, após terem decorrido **`mSecMinInWidth`** mili-segundos, a saída é ativada durante um ciclo de relógio (T_{refClk}).
- Decorrido este tempo (T_{refClk}), a saída do *debouncer* regressa ao estado inativo e assim permanece até que a entrada **`dirtyIn`** seja novamente reativada, durante **`mSecMinInWidth`** mili-segundos.

O tempo **`mSecMinInWidth`** destina-se precisamente a filtrar as transições espúrias da entrada (*bounce*).

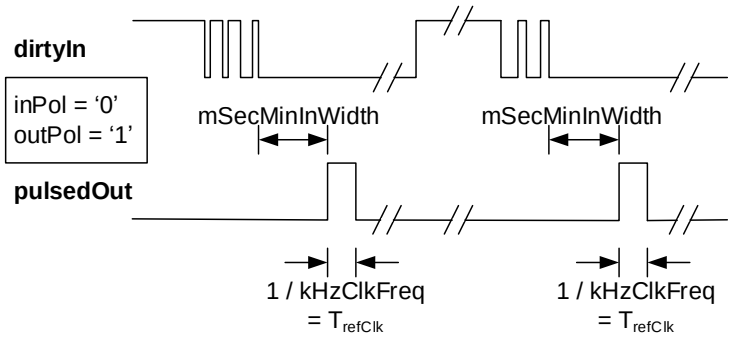

Figura 4 – Comportamento temporal do circuito de *debounce* configurável/parametrizável (assumindo que a entrada é ativa baixa e a saída é ativa alta).

O código apresentado na figura 5 pode ser encapsulado num módulo VHDL e usado de forma análoga aos *debouncers* anteriores. O módulo possui os seguintes parâmetros genéricos que permite a sua adaptação e utilização em diferentes sistemas:

- **`kHzClkFreq`** – frequência do sinal de *clock* de referência (em kHz).
- **`mSecMinInWidth`** – tempo mínimo (em mili-segundos) que a entrada deve estar estável (e ativa) para que seja gerado um impulso na saída, com a duração de um período do *clock* de referência (T_{refClk}).
- **`inPolority`** – polaridade da entrada ('0' para entrada ativa baixa; '1' para entrada ativa alta).
- **`outPolarity`** – polaridade da saída ('0' para saída ativa baixa; '1' para saída ativa alta).

No exemplo mostrado na figura 4, **`inPol`** = '0' e **`outPol`** = '1' (estando a entrada adaptada aos botões de pressão do *kit* DE2-115). Para uma entrada controlada por um botão ou interruptor, **`mSecMinInWidth`** pode estar tipicamente compreendido entre 50 e 200 (milissegundos). Caso o *clock* de referência seja proveniente do oscilador de 50 MHz do *kit* DE2-115, **`clkFrekHz`** deve ser 50000 (kHz).

[1] A utilização de um único sinal de *clock* em todo o sistema e de múltiplos sinais de *enable* (tantos quantos os necessários e com as fases adequadas) será a abordagem recomendada em todos os projetos de LSDig.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity DebounceUnit is
    generic(kHzClkFreq     : positive := 50000;
            mSecMinInWidth : positive := 100;
            inPolarity     : std_logic := '0';
            outPolarity    : std_logic := '1');
    port(refClk    : in  std_logic;
         dirtyIn   : in  std_logic;
         pulsedOut : out std_logic);
end DebounceUnit;

architecture Behavioral of DebounceUnit is

    constant MIN_IN_WIDTH_CYCLES : positive := mSecMinInWidth * kHzClkFreq;
    subtype TCounter is natural range 0 to MIN_IN_WIDTH_CYCLES;

    signal s_debounceCnt : TCounter := 0;
    signal s_dirtyIn, s_previousIn, s_pulsedOut : std_logic;

begin
    in_sync_proc : process(refClk)
    begin
        if (rising_edge(refClk)) then
            if (inPolarity = '1') then
                s_dirtyIn <= dirtyIn;
            else
                s_dirtyIn <= not dirtyIn;
            end if;

            s_previousIn <= s_dirtyIn;
        end if;
    end process;

    count_proc : process(refClk)
    begin
        if (rising_edge(refClk)) then
            if ((s_dirtyIn = '0') or
                (s_debounceCnt > MIN_IN_WIDTH_CYCLES)) then
                s_debounceCnt <= 0;
                s_pulsedOut   <= '0';

            elsif (s_dirtyIn = '1') then
                if (s_previousIn = '0') then
                    s_debounceCnt <= MIN_IN_WIDTH_CYCLES;
                    s_pulsedOut   <= '0';
                else
                    if (s_debounceCnt >= 1) then
                        s_debounceCnt <= s_debounceCnt - 1;
                    end if;

                    if (s_debounceCnt = 1) then
                        s_pulsedOut <= '1';
                    else
                        s_pulsedOut <= '0';
                    end if;
               end if;
            end if;
        end if;
    end process;

    pulsedOut <= s_pulsedOut when (outPolarity = '1') else
                 not s_pulsedOut;
end Behavioral;
```

Figura 5 – Código fonte completo do módulo de *debounce* configurável.