**UNIVERSIDADE DE SÃO PAULO**

ESCOLA DE ENGENHARIA DE SÃO CARLOS

**SEL0384 – Laboratório de Sistemas Digitais I**

Prof. Dr. Maximiliam Luppe

Bárbara Fernandes Madera - nº: 11915032

Johnny Caselato Guimarães - nº: 11915481

**PRÁTICA Nº9**

**Dispositivos de Lógica Programável tipo FPGA**

**Contador Binário Síncrono**

**SÃO CARLOS**

**2023**

**1. Objetivo**

O propósito deste relatório é apresentar o processo de implementação de um contador binário síncrono com reset assíncrono utilizando a linguagem de descrição de hardware VHDL no kit Mercurio® IV (Cyclone® IV EP4CE30F23). O contador será projetado com um barramento de dados de tamanho parametrizável através de uma estrutura de arquitetura genérica, podendo ser sobrescrita para outras aplicações.

**2. Introdução**

Contadores são componentes fundamentais em aplicações digitais, utilizados para monitorar eventos ou sequências. Eles podem ser classificados como síncronos, que operam com um sinal de clock comum, e assíncronos, que não dependem de um clock compartilhado, permitindo transições independentes entre os flip-flops. Essas duas categorias de contadores desempenham papéis essenciais em circuitos digitais, proporcionando flexibilidade para atender a diferentes aplicações.

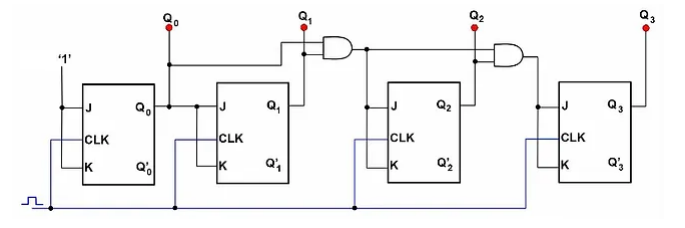


Figura 1 - Exemplo de contador binário síncrono implementado.

Os contadores síncronos garantem transições simultâneas e previsíveis entre flip-flops, nesta atividade, do tipo JK, sendo ideais para aplicações que exigem precisão de tempo. Já os contadores assíncronos, embora mais simples, podem apresentar problemas de temporização devido a variações nos atrasos de propagação.

**3. Equipamentos Necessários para Prática:**

● Kit Mercurio® IV

● Software Quartus II Web Edition

**4. Implementação e Resultados**

* Código VHDL:

Nesta atividade foram utilizados 4 arquivos VHDL: MercurioIV\_counter, counter, jkff e MercurioIV\_decod. Os códigos correspondentes ao decodificador e ao *flip-flop* JK não serão apresentados aqui pois já haviam sido disponibilizados ou implementados em outras atividades.

* + MercurioIV\_counter

No código de “MercurioIV\_counter” é feita a interface entre os elementos do hardware no kit com os parâmetros e componentes do software para a operação do contador, através dos botões e do decodificador para o display de 7 segmentos.

**-- Projeto Contador**

**-- Autores:**

**-- Bárbara Fernandes Madera - nº: 11915032**

**-- Johnny Caselato Guimarães - nº: 11915481**

**-- Professor: Maximiliam Luppe**

**entity MercurioIV\_counter is**

**port(**

**KEY : in bit\_vector(11 downto 0);**

**DISP0\_D : out bit\_vector(7 downto 0)**

**);**

**end MercurioIV\_counter;**

**architecture top of MercurioIV\_counter is**

**--Declaração de sinais**

**signal q\_out : bit\_vector(3 downto 0);**

**begin**

**--Instanciação do componente do contador**

**counter\_0 : work.counter**

**generic map(n => 4)**

**port map(clk => KEY(0), clr => KEY(1), q => q\_out);**

**--Instanciação do componente do decodificador para o display**

**disp\_0 : work.MercurioIV\_decod**

**port map(hexa => q\_out, segments => DISP0\_D(6 downto 0));**

**end top;**

* + counter

No código de “counter” é implementada lógica do contador propriamente dito, composto por um agregado de FF-JK 's, gerados a partir da parametrização. Além disso, são usados dois sinais, A e B para realizar a ligação entre os elementos lógicos (portas AND), e outros dois sinais *and\_out* e *result* para computar as saídas.

**-- Projeto Contador**

**-- Autores:**

**-- Bárbara Fernandes Madera - nº: 11915032**

**-- Johnny Caselato Guimarães - nº: 11915481**

**-- Professor: Maximiliam Luppe**

**entity counter is**

**generic(**

**n : integer := 2**

**);**

**port(**

**clk, clr : in bit;**

**q : out bit\_vector(n-1 downto 0)**

**);**

**end counter;**

**architecture rtl of counter is**

**--Declaração de sinais**

**component jk\_ff**

**port(**

**jk\_clk, jk\_clr, j, k : in bit;**

**jk\_q : buffer bit**

**);**

**end component;**

**signal and\_out : bit\_vector(n downto 0);**

**signal and\_A : bit\_vector(n downto 0);**

**signal and\_B : bit\_vector(n downto 0);**

**signal result : bit\_vector(n downto 0);**

**begin**

**and\_A(0) <= '1';**

**and\_B(0) <= '1';**

**gera\_jk : for i in 0 to n-1 generate**

**and\_out(i) <= and\_A(i) and and\_B(i);**

**jk\_0 : jk\_ff port map(**

**jk\_clk => clk,**

**jk\_clr => clr,**

**j => and\_out(i),**

**k => and\_out(i),**

**jk\_q => and\_B(i+1)**

**);**

**and\_A(i+1) <= and\_out(i);**

**result(i) <= and\_B(i+1);**

**end generate gera\_jk;**

**q <= result(n-1 downto 0);**

**end rtl;**

* Circuito RTL:

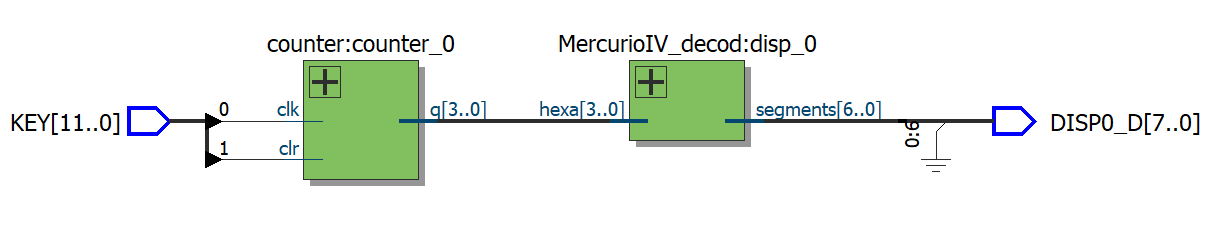


Figura 2 - Visualização RTL geral.

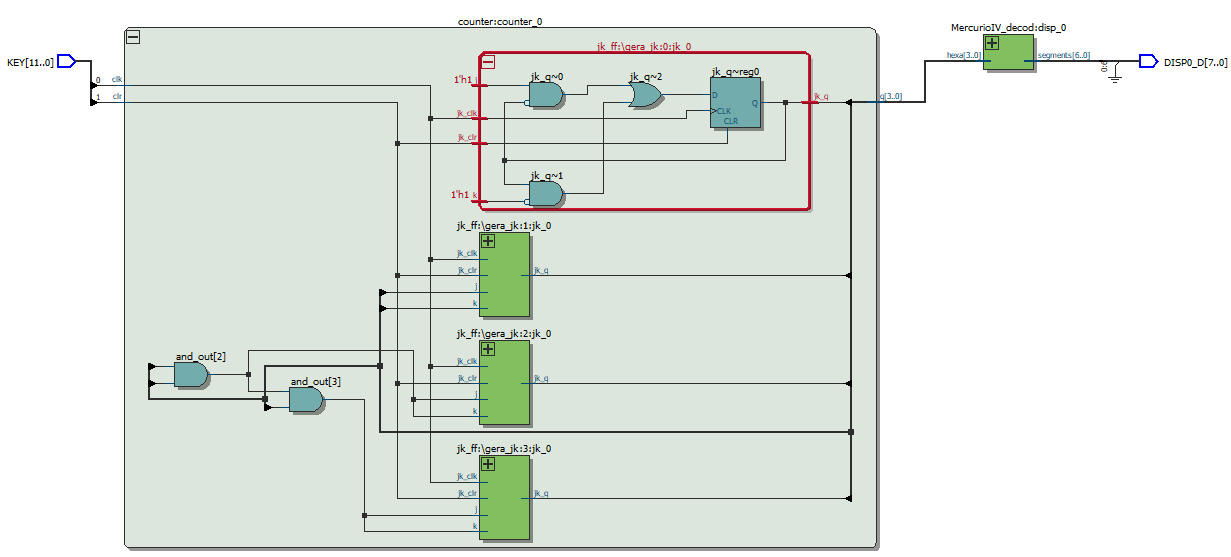


Figura 3 - Visualização RTL expandida.

* Número de células lógicas utilizadas:

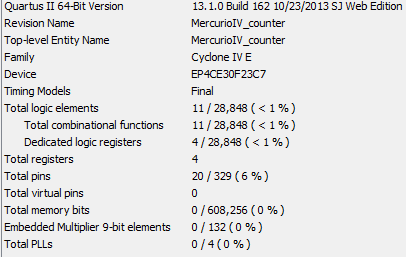


Figura 4 - Resumo dos resultados da compilação.

**5. Conclusão**

Na implementação prática desta atividade, observou-se que os resultados obtidos no kit foram inconsistentes devido à necessidade de um *debouncer* nos botões. Ao tocar em um botão para gerar um pulso de clock, vários sinais eram computados simultaneamente, resultando em comportamento imprevisível no contador. Essa inconsistência ressalta a importância de considerar e mitigar fenômenos como o bouncing nos dispositivos de entrada, garantindo assim a estabilidade e confiabilidade das operações em circuitos digitais.