Universidade Federal do Rio Grande do Sul
Instituto de Informática da UFRGS
Definição do Projeto 2 – Sistemas Digitais

Nome: Luccas da Silva Lima e Matheus Almeida Silva
Matrícula: 324683 e 316326

Problema a ser resolvido:

Iniciamos com um número *x* qualquer e aplicamos a seguinte regra para atualizar o valor de *x*:

$$x := \begin{cases} x/2 & \text{se } x \text{ par} \\ 3x + 1 & \text{se } x \text{ ímpar} \end{cases}$$

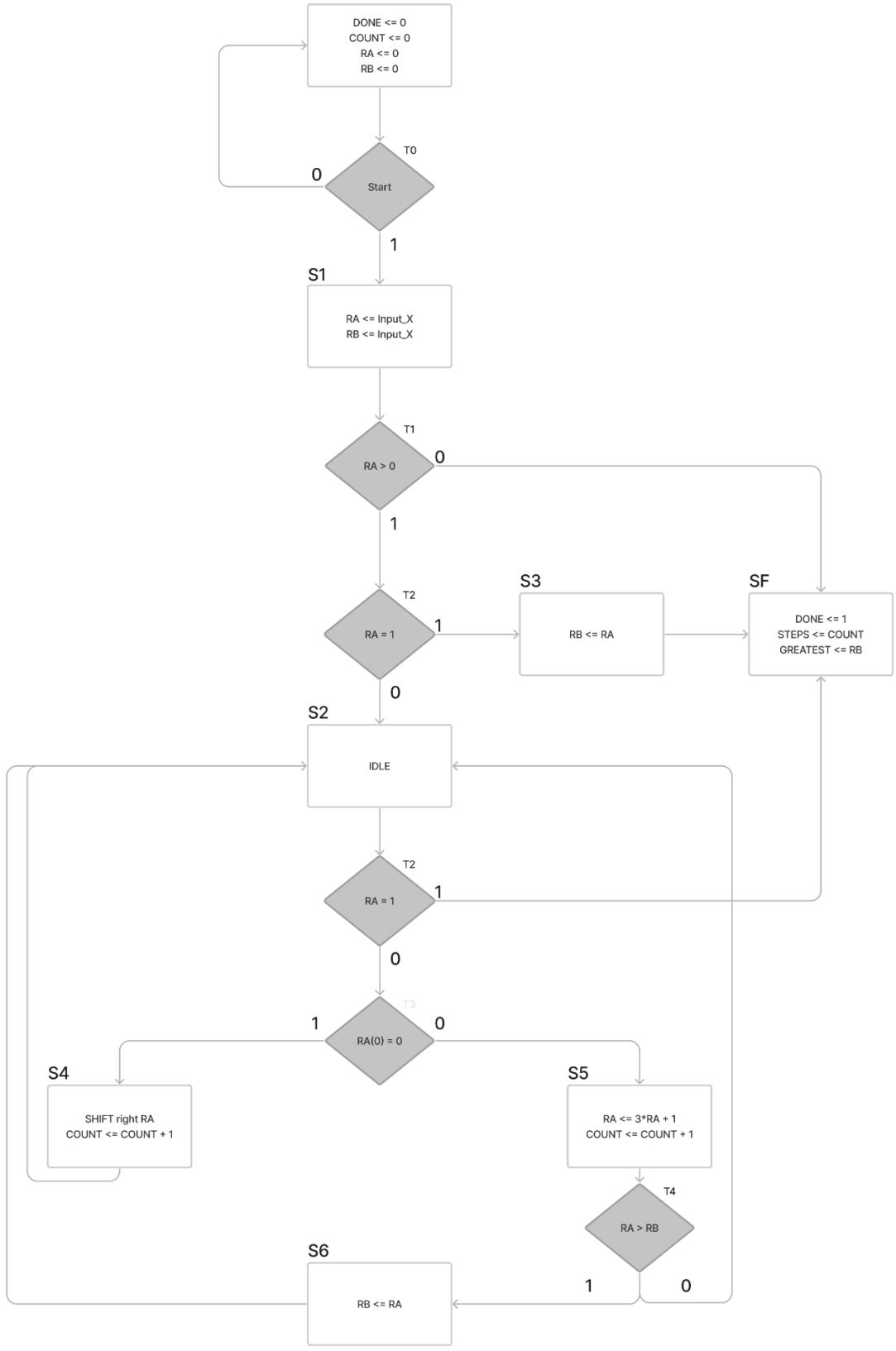
A **Conjectura de Collatz** diz que, para qualquer número ***x* > 0** inicial, chegaremos sempre ao valor 1 depois de uma quantidade finita de passos. A questão em aberto é que não se sabe se isso é verdade. Este é um problema em aberto na matemática e descrito como um dos mais difíceis de ser resolvido, apesar de sua definição extremamente simples. Neste trabalho, esse problema será implementado em um sistema digital, partindo de um *x* inicial com 8 bit, e pede-se que determine a quantidade de passos necessários para chegar em 1 e qual o maior valor encontrado no processo. Por exemplo, partindo do valor 3, temos:

3 (ímpar) → 10 (par) → 5 (ímpar) → 16 (par) → 8 (par) → 4 (par) → 2 (par) → 1 (par)

Neste caso, a resposta será 7 passos e 16 como maior valor. Uma vez que o maior valor pode crescer muito além do valor inicial de *x*, o controle interno de *x* (e do maior valor) deverá ser feito utilizando dois bytes (16 bits). A contagem de passos pode ser realizada utilizando um único byte.

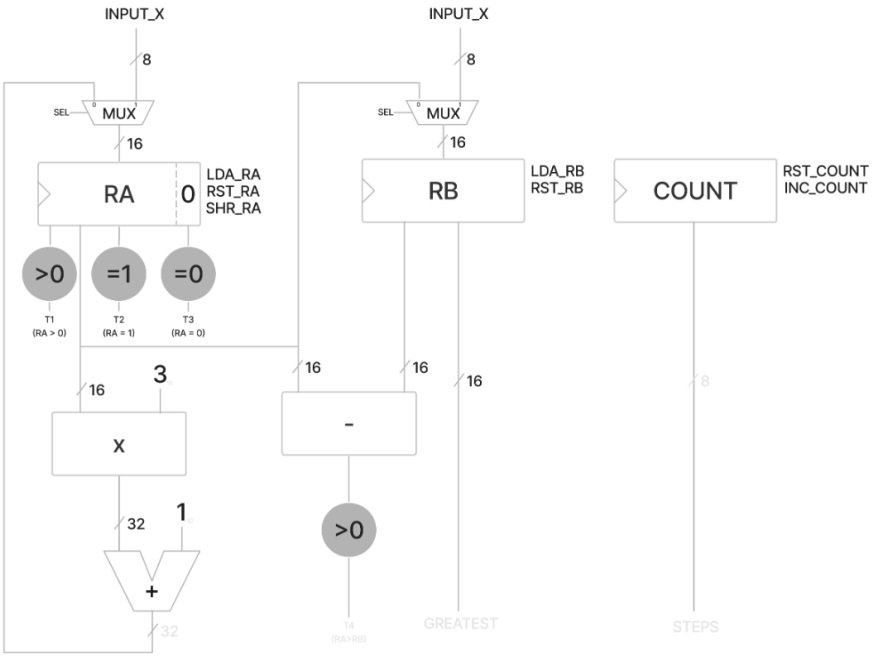
FICHA DE RESPOSTA (a ser preenchia e entregue)
Passo 1 ao Passo 5

Passo 1: Descreva o fluxograma ASM do algoritmo para hardware (2 pts)

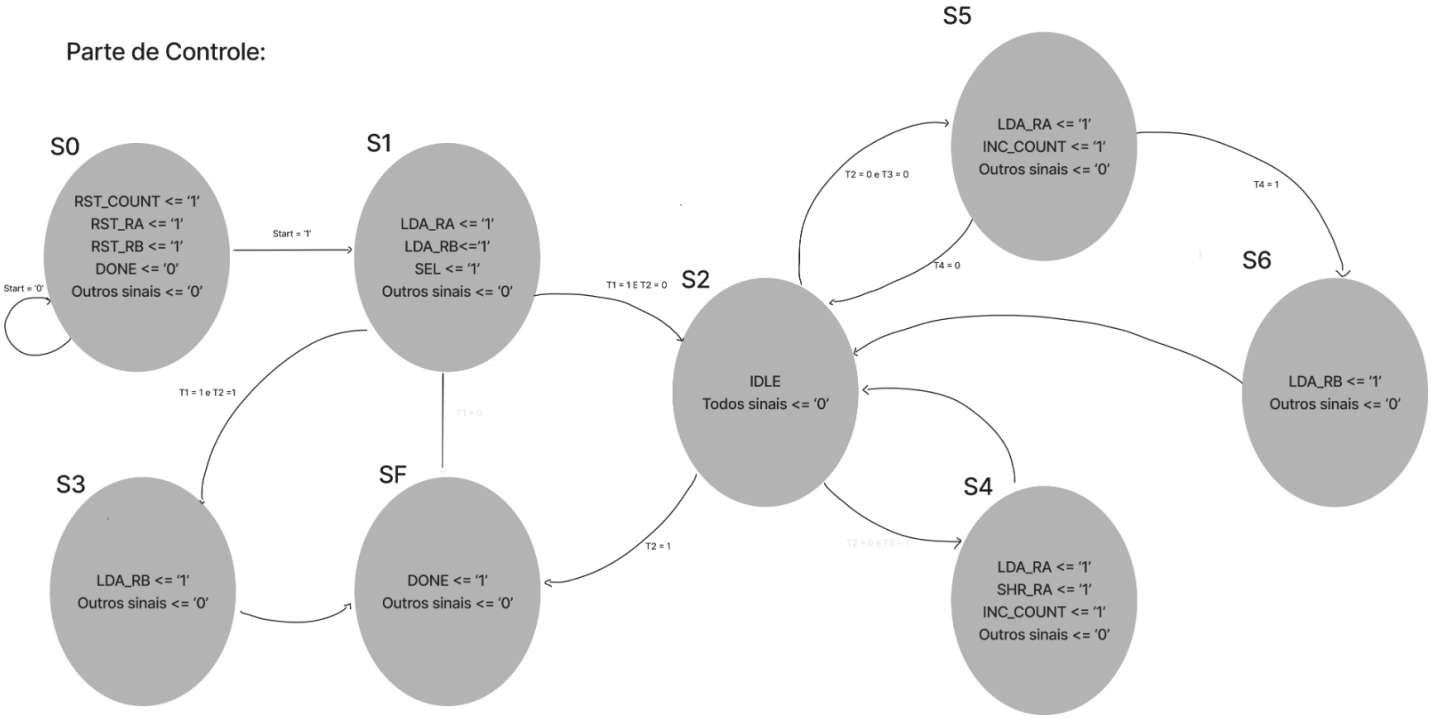


Passo 2: Desenhe o diagrama de blocos da Parte Operativa (PO) e a FSM da Parte de Controle (PC) extraídos do fluxograma ASM (2 pts)

Parte Operativa:



Parte de Controle:



Passo 3: Descreve o VHDL do PC-PO no ISE ou Vivado, sintetize e simule (4 pts)

VHDL:

```
----------------------------------------------------------------------------------
-- Projeto 2 - Conjectura De Collatz
-- Alunos:  Luccas da Silva Lima  - 324683
--          Matheus Almeida Silva - 316326
-- VHDL do PCPO da Conjectura de Collatz
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Collatz_VHDL is
   Port ( start : in STD_LOGIC;
          rst : in STD_LOGIC;
          clk : in  STD_LOGIC;
          inputX : in STD_LOGIC_VECTOR (7 downto 0);
          done : out STD_LOGIC;
          greatest : out STD_LOGIC_VECTOR (15 downto 0);
          steps : out STD_LOGIC_VECTOR (7 downto 0));
end Collatz_VHDL;

architecture Behavioral of Collatz_VHDL is
   --MUX
   signal SEL: STD_LOGIC := '0';
   signal outputMuxA: STD_LOGIC_VECTOR (15 downto 0);
   signal outputMuxB: STD_LOGIC_VECTOR (15 downto 0);
   --RA
   signal LDA_RA: STD_LOGIC := '0';
   signal RST_RA: STD_LOGIC := '0';
   signal SHR_RA: STD_LOGIC := '0';
   signal outputRA: STD_LOGIC_VECTOR (15 downto 0);
   --RB
   signal LDA_RB: STD_LOGIC := '0';
   signal RST_RB: STD_LOGIC := '0';
   signal outputRB: STD_LOGIC_VECTOR (15 downto 0);
   --COUNT
   signal INC_COUNT: STD_LOGIC := '0';
   signal RST_COUNT: STD_LOGIC := '0';
   signal outputCOUNT: STD_LOGIC_VECTOR (7 downto 0);
   --TESTS
   signal outputT1: STD_LOGIC := '0';
   signal outputT2: STD_LOGIC := '0';
   signal outputT3: STD_LOGIC := '0';
   signal outputT4: STD_LOGIC := '0';
   -- Estados
```

```
type state_type is (S0, S1, S2, S3, S4, S5, S6, SF, S1T, S2T, S5T);
signal state, nextState : state_type;

begin
-- PARTE OPERATIVA
-- PROCESS MUX
MUX : process(SEL, inputX, outputRA)
variable inputMUXA : STD_LOGIC_VECTOR(31 downto 0);
begin
   if  (SEL = '1') then
      outputMuxA(15 downto 8) <= "00000000";
      outputMuxA(7 downto 0) <= inputX;
      outputMuxB(15 downto 8) <= "00000000";
      outputMuxB(7 downto 0) <= inputX;
   else
      inputMuxA := STD_LOGIC_VECTOR(unsigned(outputRA) * 3 + 1);
      outputMUXA <= inputMUXA(15 downto 0);
      outputMuxB <= outputRA;
   end if;
end process;
-- PROCESS RA
RA : process(clk, rst)
begin
   if rst = '1' then
         outputRA <= "0000000000000000";
   elsif rising_edge(clk) then
      if (LDA_RA = '1') then
         if (SHR_RA = '1') then
            outputRA <= STD_LOGIC_VECTOR(shift_right(unsigned(outputRA),1));
         else
            outputRA <= outputMuxA;
         end if;
      elsif (RST_RA = '1') then
         outputRA <= "0000000000000000";
      else
         outputRA <= outputRA;
      end if;
   end if;
end process;
-- PROCESS RB
RB : process(clk, rst)
begin
   if rst = '1' then
         outputRB <= "0000000000000000";
   elsif rising_edge(clk) then
      if (LDA_RB = '1') then
         outputRB <= outputMuxB;
```

```
      elsif (RST_RA = '1') then
         outputRB <= "0000000000000000";
      else
         outputRB <= outputRB;
      end if;
   end if;
end process;
-- PROCESS COUNT
COUNT : process(clk, rst)
begin
   if rst = '1' then
         outputCOUNT <= "00000000";
   elsif rising_edge(clk) then
      if (INC_COUNT = '1') then
         outputCOUNT <= STD_LOGIC_VECTOR(unsigned(outputCOUNT) + 1);
      elsif (RST_COUNT = '1') then
         outputCOUNT <= "00000000";
      end if;
   end if;
end process;
-- PROCESS TESTS
TESTS : process(outputRA, outputRB)
begin
   if (outputRA > "0000000000000000") then
      outputT1 <= '1';
   else
      outputT1 <= '0';
   end if;
   if (outputRA = "0000000000000001") then
      outputT2 <= '1';
   else
      OutputT2 <= '0';
   end if;
   if (outputRA(0) = '0') then
      outputT3 <= '1';
   else
      outputT3 <= '0';
   end if;
   if (outputRA > outputRB) then
      outputT4 <= '1';
   else
      outputT4 <= '0';
   end if;
end process;

--PARTE CONTROLE
--PROCESS STATES REGISTERS
```

```
RStates: process(rst, clk)
begin
   if rst = '1' then
      state <= S0;
   elsif rising_edge(clk) then
      state <= nextState;
   end if;
end process;

--PROCESS FSM
FSM : process(state, nextState, start, outputT1, outputT2, outputT3, outputT4)
begin
   LDA_RA <= '0';
   RST_RA <= '0';
   SHR_RA <= '0';
   LDA_RB <= '0';
   RST_RB <= '0';
   RST_COUNT <= '0';
   INC_COUNT <= '0';
   SEL <= '0';
   DONE <= '0';

   case state is
      when S0=>
         RST_RA <= '1';
         RST_RB <= '1';
         RST_COUNT <= '1';
         if (start = '1') then
            nextState <= S1;
         else
            nextState <= S0;
         end if;
      when S1=>
         LDA_RA <= '1';
         LDA_RB <= '1';
         RST_RA <= '0';
         RST_RB <= '0';
         RST_COUNT <= '0';
         SEL <= '1';
         nextState <= S1T;
      when S1T=>
         if (outputT1 = '1') then
            if (outputT2 = '1') then
               nextState <= S3;
            else
               nextState <= S2;
            end if;
```

```
    else
      nextState <= SF;
    end if;
  when S2=>
    LDA_RA <= '0';
    RST_RA <= '0';
    SHR_RA <= '0';
    LDA_RB <= '0';
    RST_RB <= '0';
    RST_COUNT <= '0';
    INC_COUNT <= '0';
    SEL <= '0';
    DONE <= '0';
    nextState <= S2T;
  when S2T=>
    if (outputT2 = '1') then
      nextState <= SF;
    else
      if (outputT3 = '1') then
        nextState <= S4;
      else
        nextState <= S5;
      end if;
    end if;
  when S3=>
    LDA_RA <= '0';
    LDA_RB <= '1';
    SEL <= '0';
    nextState <= SF;
  when S4=>
    LDA_RA <= '1';
    SHR_RA <= '1';
    INC_COUNT <= '1';
    nextState <= S2;
  when S5=>
    LDA_RA <= '1';
    INC_COUNT <= '1';
    nextState <= S5T;
  when S5T=>
    if (outputT4 = '1') then
      nextState <= S6;
    else
      nextState <= S2;
    end if;
  when S6=>
    LDA_RA <= '0';
    LDA_RB <= '1';
```

```
            INC_COUNT <= '0';
            nextState <= S2;
         when SF=>
            LDA_RB <= '0';
            DONE <= '1';
            nextState <= SF;
         when others  =>
            nextState <= S0;
      end case;
   end process;
   greatest <= outputRB;
   steps <= outputCOUNT;
end Behavioral;
```

Testbench:

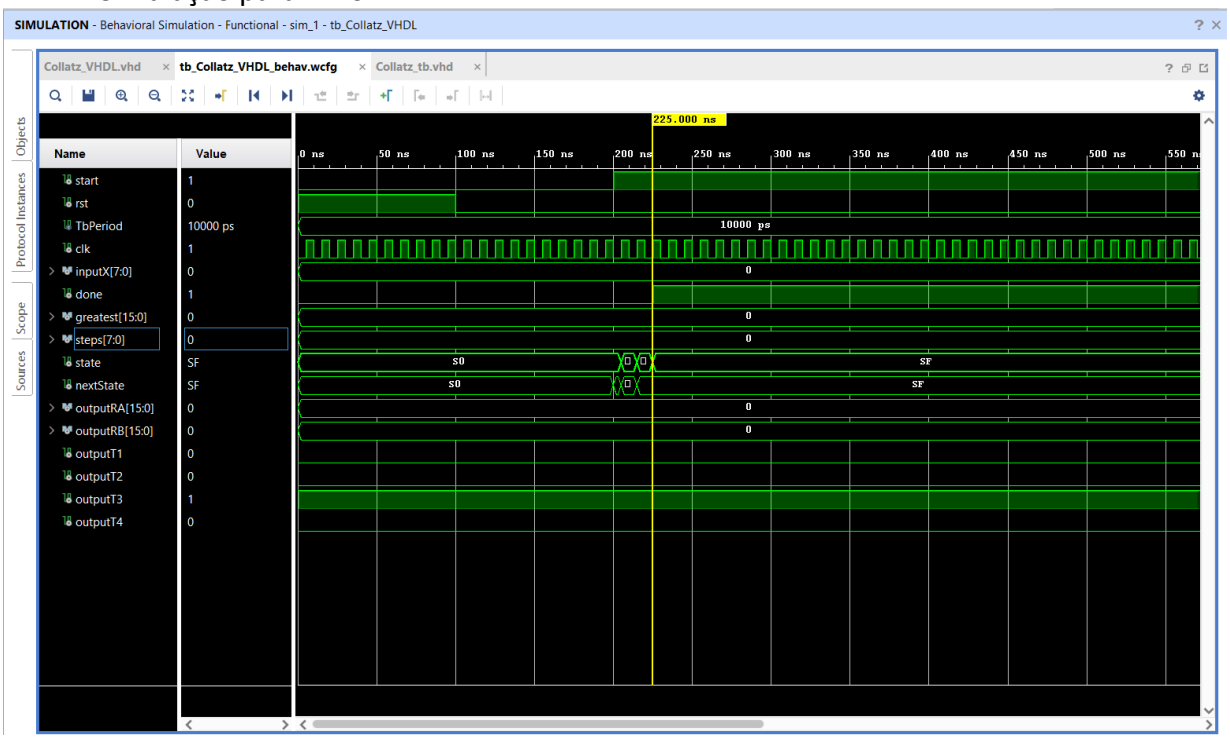
```
library ieee;
use ieee.std_logic_1164.all;

entity tb_Collatz_VHDL is
end tb_Collatz_VHDL;

architecture tb of tb_Collatz_VHDL is

   component Collatz_VHDL
      port (start    : in std_logic;
            rst      : in std_logic;
            clk      : in std_logic;
            inputX   : in std_logic_vector (7 downto 0);
            done     : out std_logic;
            greatest : out std_logic_vector (15 downto 0);
            steps    : out std_logic_vector (7 downto 0));
   end component;

   signal start    : std_logic;
   signal rst      : std_logic;
   signal clk      : std_logic;
   signal inputX   : std_logic_vector (7 downto 0);
   signal done     : std_logic;
   signal greatest : std_logic_vector (15 downto 0);
   signal steps    : std_logic_vector (7 downto 0);

   constant TbPeriod : time := 10 ns; -- EDIT Put right period here
   signal TbClock : std_logic := '0';
   signal TbSimEnded : std_logic := '0';

begin
```

```
    dut : Collatz_VHDL
    port map (start    => start,
              rst      => rst,
              clk      => clk,
              inputX   => inputX,
              done     => done,
              greatest => greatest,
              steps    => steps);

    -- Clock generation
    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';

    -- EDIT: Check that clk is really your main clock signal
    clk <= TbClock;

    stimuli : process
    begin
      -- EDIT Adapt initialization as needed
      start <= '0';
      inputX <= (others => '0');

      -- Reset generation
      -- EDIT: Check that rst is really your reset signal
      rst <= '1';
      wait for 100 ns;
      rst <= '0';
      wait for 100 ns;

      -- EDIT Add stimuli here
      start <= '1';
      --inputX <= "00000000"; --0
      --inputX <= "00000001"; --1
      inputX <= "00000011"; --3
      --inputX <= "00000100"; --4
      --inputX <= "10000001"; --129
      --inputX <= "11111110"; --254
      --inputX <= "11111111"; --255
      wait for 450 * TbPeriod;

      -- Stop the clock and hence terminate the simulation
      TbSimEnded <= '1';
      wait;
    end process;

end tb;
```

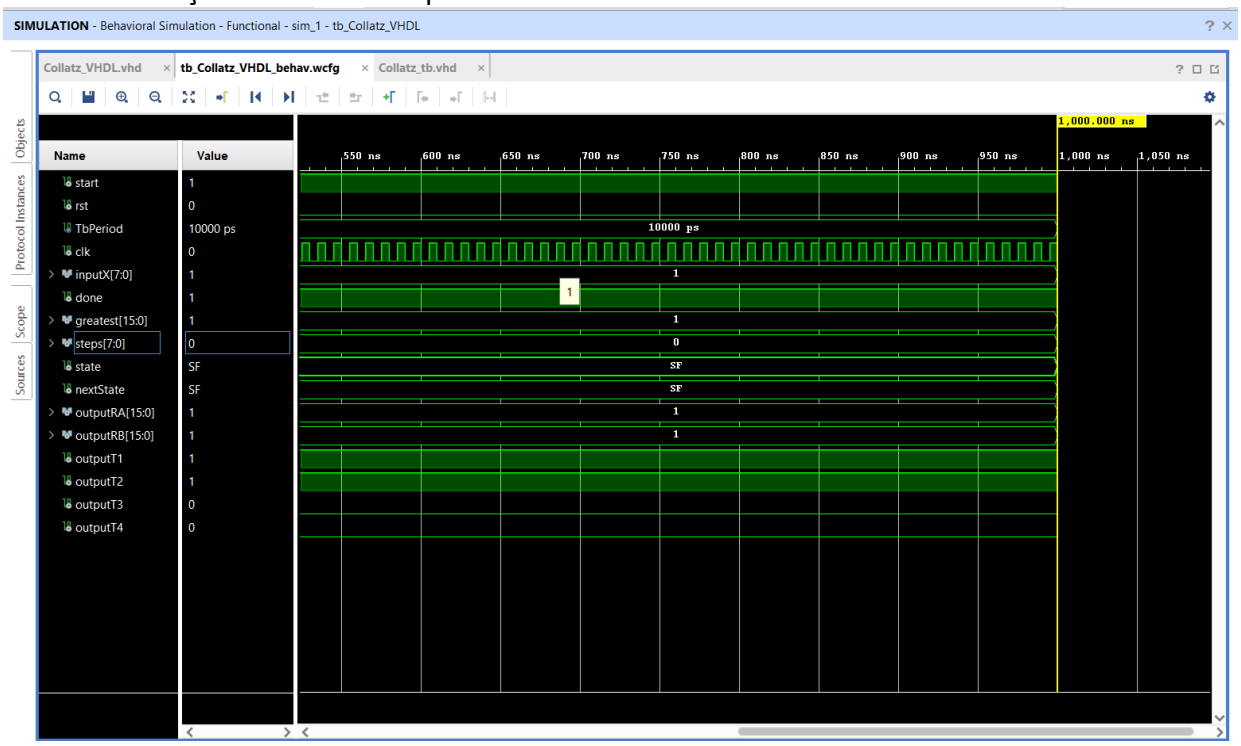
```
configuration cfg_tb_Collatz_VHDL of tb_Collatz_VHDL is
   for tb
   end for;
end cfg_tb_Collatz_VHDL;
```

Simulações:

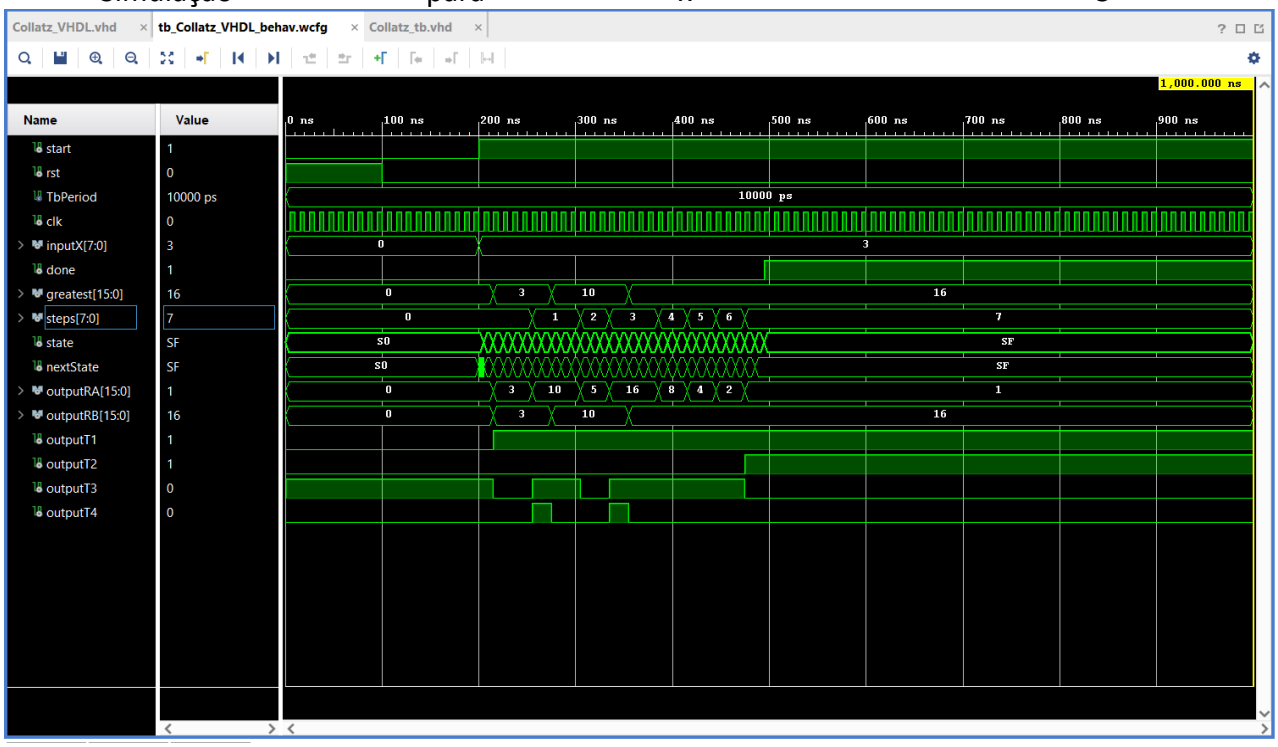
Simulação para x = 0



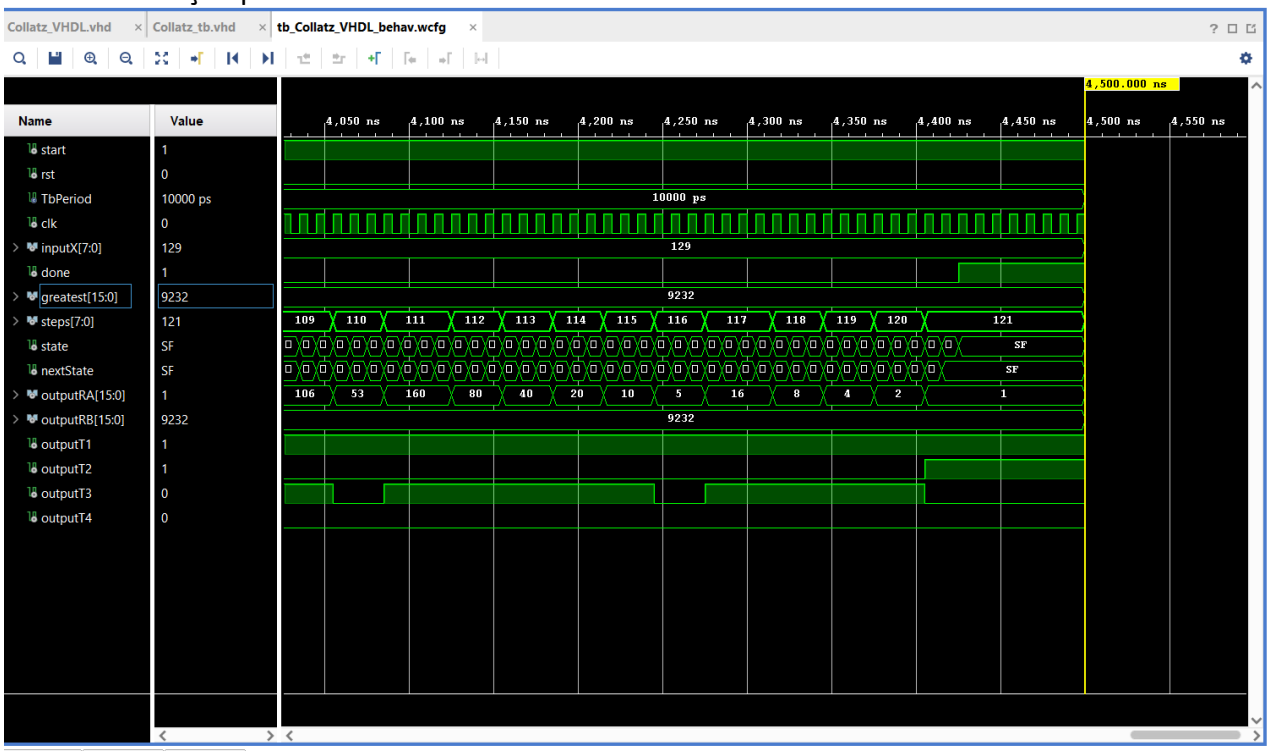
Simulação para x = 1



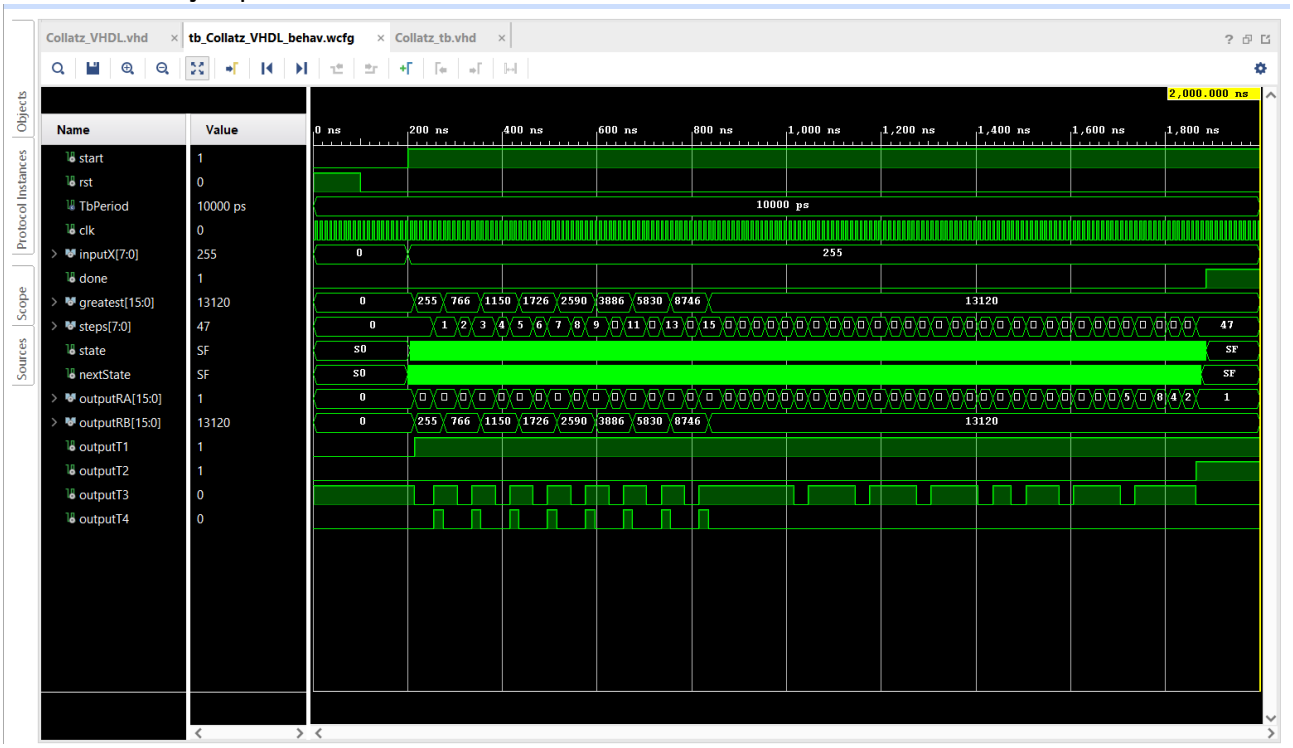
Simulação para x = 3



Simulação para x = 129



Simulação para x = 255



Passo 4: Descreva o algoritmo em C e sintetize no HLS do Vivado usando um conjunto de diretivas e justifica as escolhas (1 pt)

Algoritmo em C

- header:

```
#ifndef __COLLATZ_H__
#define __COLLATZ_H__

typedef unsigned char Uint8; //8bits
typedef unsigned short Uint16; //16bits

// Prototype of top level function for C-synthesis
void Collatz(Uint8 x, Uint16 greatest, Uint8 steps);

#endif
```

- função:

```
#include "ConjecturaDeCollatz.h"
#include <stdio.h>
#include <stdlib.h>

void Collatz(Uint8 x, Uint16 greatest, Uint8 steps){
```

```
        Uint16 aux = (Uint16)x;
        greatest = (Uint16)x;
        steps = (Uint8)0;
        if (aux > (Uint16)0){
                loop:while(aux != (Uint16)1){
                        if (aux % (Uint16)2 == 0){
                                aux = aux >> 1;
                        } else{
                                aux = (aux * (Uint16)3) + (Uint16)1;
                        }
                        if (aux > greatest){
                                greatest = aux;
                        }
                        steps = steps + (Uint8)1;
                }
        }
        printf("\n steps:%d \n greatest:%d \n\n",(int)steps,(int)greatest);
}
```

Diretivas

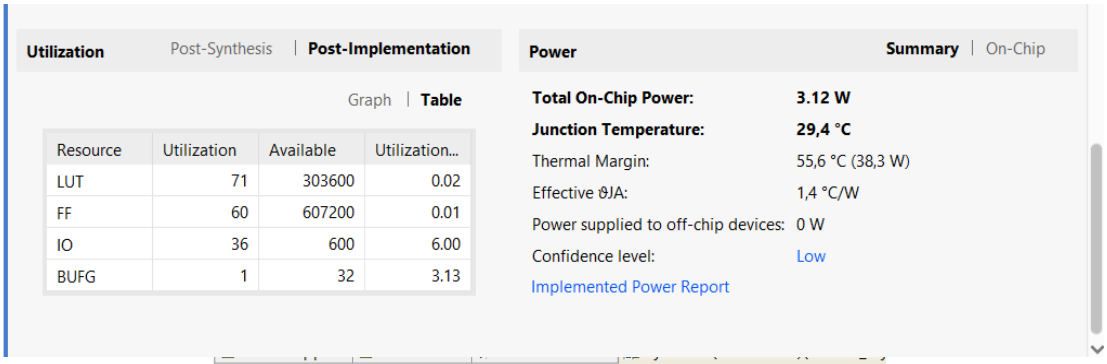
Pragma HLS pipeline: Permite a execução simultânea de operações. A escolha de utilizar Pipeline foi devida ao fato de que o pipeline de um loop permite que as operações do loop sejam implementadas de maneira concorrente.

Pragma HLS unroll: Com o unroll, é possível desenrolar loops para criar várias operações independentes em vez de uma única coleção de operações. No nosso trabalho, optamos por utilizar unroll no loop while para que o loop fosse totalmente desenrolado, para poder ser executado todo simultaneamente.

Passo 5: Tabela final comparativa (1 pt)

FPGA:xc7vx485t-ffg1157-1

Versão	PC-PO	HLS
# LUTs	71	85
# ffps	60	18
# DSP	0	0
# BRAM	0	0
# BUFG	1	0
Tclk	10	3
# cc	49	8



Utilization Post-Synthesis | **Post-Implementation**

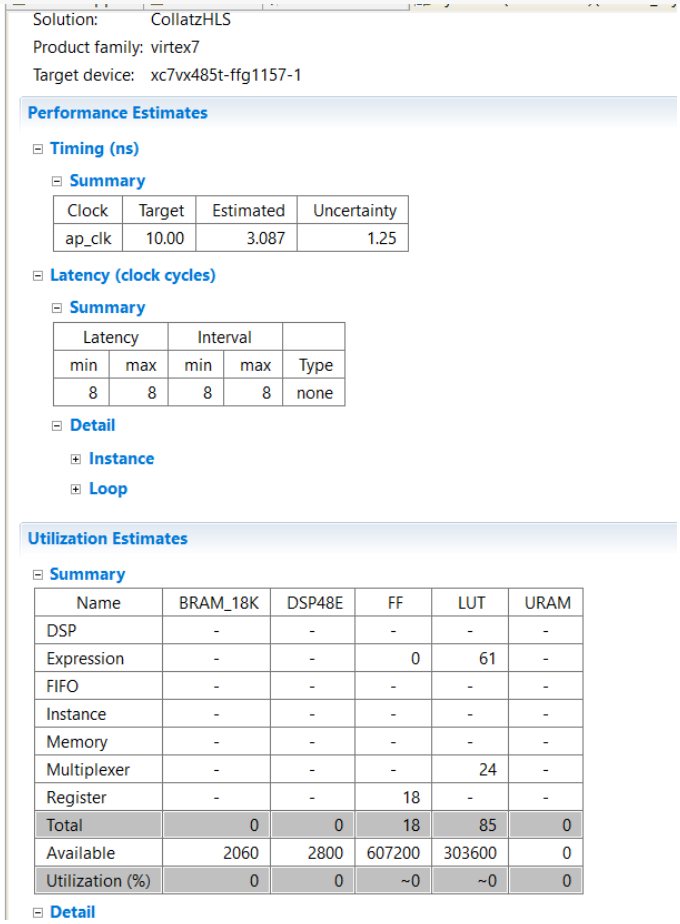
Graph | **Table**

Resource	Utilization	Available	Utilization...
LUT	71	303600	0.02
FF	60	607200	0.01
IO	36	600	6.00
BUFG	1	32	3.13

Power **Summary** | On-Chip

Total On-Chip Power:	**3.12 W**
Junction Temperature:	**29,4 °C**
Thermal Margin:	55,6 °C (38,3 W)
Effective ϑJA:	1,4 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report



Solution: CollatzHLS

Product family: virtex7

Target device: xc7vx485t-ffg1157-1

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	3.087	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
8	8	8	8	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	61	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	24	-
Register	-	-	18	-	-
Total	0	0	18	85	0
Available	2060	2800	607200	303600	0
Utilization (%)	0	0	~0	~0	0

Detail

Entrega desta ficha preenchida dia 13 de outubro de 2022
Apresentação com ppt ou similar em aula no dia 13 de outubro de 2022