Implementation of a Stopwatch (modulo 16) with the *Arty A7* board

Andrea Nicolai, Michele Guadagnini

Department of Physics and Astronomy
MANAGEMENT AND ANALYSIS OF PHYSICS DATASET (Mod. A)
Università degli Studi di Padova
via 8 Febbraio 1848, 2, 35122 Padova PD

1. Aim

This study aims to implement a stopwatch that displays seconds ($0 \mapsto 15$) with 4 LEDs of the *Arty A7* board. Moreover, three buttons of the *Arty A7* are used to develop the following functionalities:

- **START**: Enable counting;
- **STOP**: Disable counting, and freeze the system;
- **RESET**: Reset to zero.

In addition, three of the total four switches available take the following roles:

- **BACKWARDS**: if ON, then count backwards;
- **Double velocity**: Enable counting at double speed (0.5 sec);
- **Halve velocity**: Enable counting at half speed (2 sec).

2. Introduction

Digilent *Arty A7* Development Board is a ready-to-use development platform designed around the Artix-7 Field Programmable Gate Array (FPGA) from Xilinx. The *Arty A7* development board provides adaptability and a flexible processing platform. The *Arty A7* development board is fully compatible with the high-performance Vivado design suite.

October 13, 2021

The development board is bound to multiple sets of processing peripherals. The *Arty A7* development board features as a communication powerhouse, chock-full of UARTs, SPIs, I2Cs, and an Ethernet MAC. This development board also features a precise timekeeper with twelve 32-bit timers.

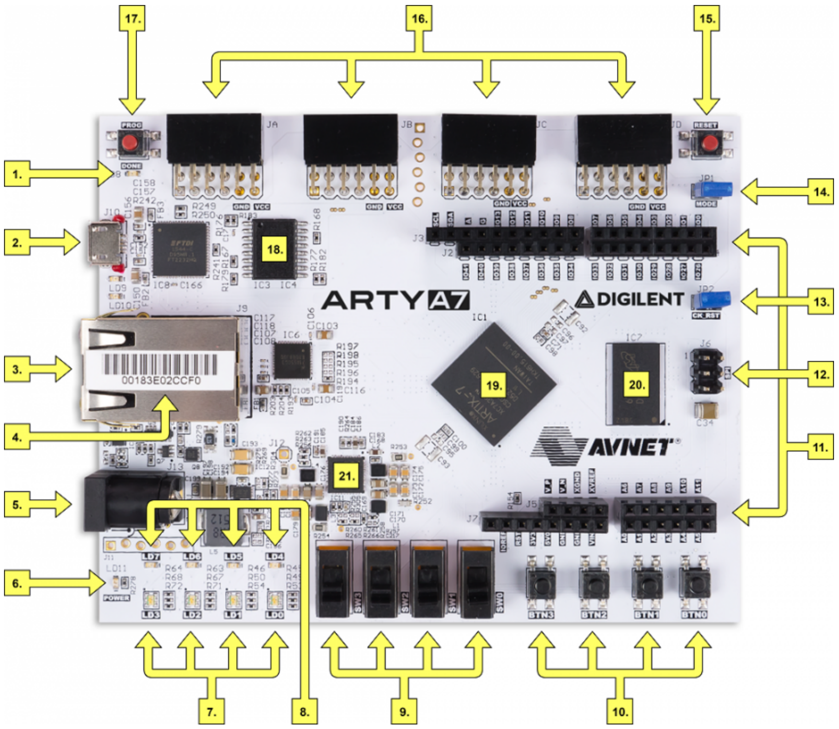

Figure 1: Schematic of the *Arty A7* development board.

No	**Description**	**No**	**Description**	**No**	**Description**
1	FPGA programming DONE LED	8	User RGB lights	15	chipKIT processor reset
2	Shared USB JTAG / UART port	9	User slide switches	16	Pmod connectors
3	Ethernet connector	10	User push buttons	17	FPGA programming reset button
4	MAC address sticker	11	Arduino/chipKIT shield connectors	18	SPI flash memory
5	Power jack for optional power supply	12	Arduino/chipKIT shield SPI connector	19	Artix FPGA
6	Power good LED	13	chipKIT processor reset jumper	20	Micron DDR3 memory
7	User LEDs	14	FPGA programming mode	21	Dialog Semiconductor DA9062 power supply

3. Design of the stopwatch

To understand the working of the stopwatch, primarily one needs to have knowledge about the clock cycles and the trigger. The *Arty A7* board has inbuilt 100 MHz oscillator. Therefore the clock frequency is 100 MHz, means to say that every clock cycle lasts up to 10 ns.

The first step to do in the code implementation is to define the variables that will be used by the program. Six *STD_LOGIC* variables are needed, one for each button *START*, *STOP*, *RESET*, one for the *CLOCK* signal, one *BACK_TRIG* that denotes whether the counting will be operating incrementally or decrementally, one *HALF_SPEED* and *DOUBLE_SPEED* that either double or halve the counting velocity, and finally one of dimension 4 to store the state of each LED. The variables definition is reported in Listing 1.

The counter is implemented as a *Finite State Machine* (FSM) with four possible states: *s_idle*, *s_counting_forward*, *s_counting_backwards*, *s_paused*. By pressing the buttons, it is possible to change the state of the stopwatch.

The number of elapsed seconds is stored in binary format in the signal *seconds*; in this way we have a direct correspondence between the single bit values and the LED states.

The program essentially is divided in two main processes: *p_count_forward* and *p_count_backwards*. The code related to **p_count_forward** is the one

```
entity counter is
    Port ( start : in STD_LOGIC;
           stop : in STD_LOGIC;
           rst : in STD_LOGIC;
           clk : in STD_LOGIC;
           back_trig: in STD_LOGIC;
           half_speed: in STD_LOGIC;
           double_speed: in STD_LOGIC;
           led_out : out STD_LOGIC_VECTOR (3 downto 0));
end counter;
```

Listing 1: Definition of the entity *counter* that contains the needed variables.

listed in 2: here there are reported instructions to be executed in the counting-forward state.

In the *forward* state, counting is done by increasing the integer variable *cnt* at each *rising edge* of the clock. Since the clock of the board is 100 MHz, we define another integer variable, *WTIME*, that stores the number of clock cycles that represent a second (10^8 cycles). If the program is in the state *s_counting_forward*, when *cnt* reaches this value, the signal *seconds* is increased by 1. After 16" *seconds*, it reaches overflow and the program is stopped and reset. On the contrary, if the machine is in state *s_counting_backwards*, the *cnt* variable is initialized to $WTIME$, which might take different values according to the input related to the velocity of counting, and then is decreased up to the null value. In such a way, every time *cnt* reaches 0, the seconds decrease in steps of 1, starting from 15 and ending with 0.

Note in the appendix 4 as the **s_counting_backwards** differs from the *s_counting_forward* by essentially considering the counting going forward and backwards, and by exchanging the roles of the two integers $WTIME$ and *cnt*. Moreover, according to the switches designed for taking into account the velocity, $WTIME$ is either halved or doubled to increase or decrease the speed.

Finally, the logical variables defined in the entity *counter* are linked to the physical buttons, LEDs or port in the schematic (.xdc) file. In particular, the *start* variable is linked to the physical button *BTN0*, *stop* to *BTN1* and *reset* to *BTN2*. Switches *back_trig*, *half_speed*, *double_speed* are, respectively $sw[0]$, $sw[1]$, $sw[2]$. The seconds are displayed on the LEDs from *LD0* (LSB)

```
when s_counting_forward =>

  if double_speed = '1' and half_speed = '0' then
    WTIME := WTIME_INIT/2;
  elsif double_speed = '0' and half_speed = '1' then
    WTIME := WTIME_INIT*2;
  else
    WTIME := WTIME_INIT;
  end if;

  if cnt < WTIME then
     if stop = '1' then
        state_fsm <= s_paused;
     end if;
  elsif cnt = WTIME or cnt > WTIME then
     if seconds = b"1111" then --overflow
        state_fsm <= s_idle;
        seconds <= (others => '0');
        cnt := 0;
     else seconds <= seconds + 1;
          if forw = '1' then
             state_fsm <= s_counting_forward;
          elsif forw = '0' then
             state_fsm <= s_counting_backward;
          end if;
          cnt := 0;
     end if;
  end if;
  cnt := cnt + 1 ;
```

Listing 2: Set of instructions of the state *s_counting_forward*.

to *LD3* (MSB).

3.1. Simulation

In Figure 2 it is possible to appreciate the behaviour of a Stopwatch counter. In particular, the finite state machine phases of counting forward, idle, and paused are highlighted. Moreover, one should take into account that the overall speed of the process is increased to represent faster all these functionalities.

On the other hand, in Fig. 3 one can note as the Stopwatch can work either in the forward direction or in the backward one. Initially, once having started from 15, the counting occurs in the reverse order. Once the proper trigger has been switched, however, the seconds start to be incremented.

Finally, in Fig. 4, the Stopwatch experiences different velocities, as one can clearly see from the *LED* signal. Initially, the velocity is speeded up by a factor 2, whereas later on it is halved. Finally, it goes back to its initial value, in such a way that one can appreciate the three different regimes.

4. Conclusions

In this work, the implementation of a Stopwatch with the *Arty A7* is investigated. In particular, we were able to develop the functionality of START, STOP, and RESET directly from the board buttons and to check the correctness of the system's behaviour watching the LEDs blinking. Moreover, this object was further implemented by either increasing or decreasing the velocity and making it able to run incrementally or decrementally.

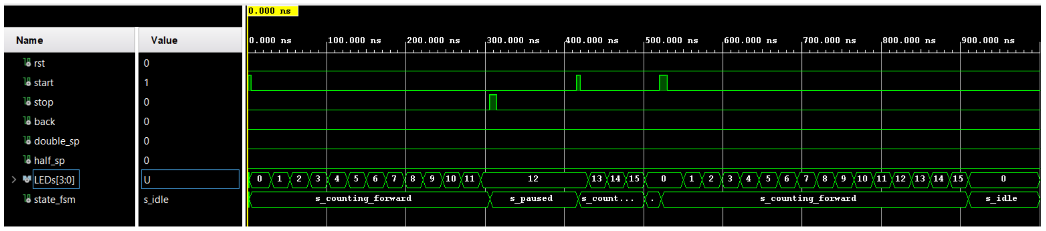

Figure 2: Stopwatch working simulation - 1. Simulation is sped up for a clearer view.

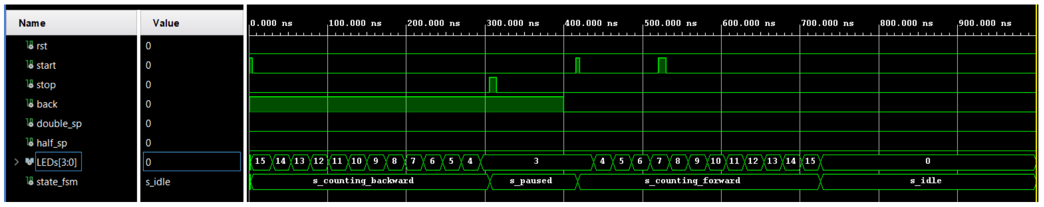

Figure 3: Stopwatch working simulation - 2. Here the counter works first in the reverse order, and then in the correct one.

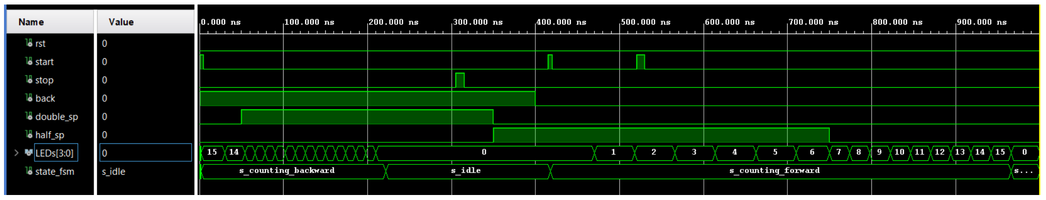

Figure 4: Stopwatch working simulation - 3. Here different speeds are used.

Appendix

The full code of the stopwatch implementation is reported below.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity counter is
```

```
    Port ( start : in STD_LOGIC;
           stop : in STD_LOGIC;
           rst : in STD_LOGIC;
           clk : in STD_LOGIC;
           back_trig: in STD_LOGIC;
           half_speed: in STD_LOGIC;
           double_speed: in STD_LOGIC;
           led_out : out STD_LOGIC_VECTOR (3 downto 0));
end counter;

architecture Behavioral of counter is

type state is (s_idle, s_paused, s_counting_forward, s_counting_backward);
signal state_fsm: state := s_idle;

signal seconds: unsigned(3 downto 0);
signal forw: STD_LOGIC;

begin

p_count: process(clk, rst, start, stop) is
variable WTIME_INIT: integer := 100000000; --100mHz is 100*1e6
variable WTIME: integer;
variable cnt: integer;

begin

  if rst = '1' then
     seconds <= (others => '0');
     cnt := 0;
     state_fsm <= s_idle;

  elsif rising_edge(clk) then
     case state_fsm is
         when s_idle =>
         --if counting forwards, then set start from b"0000"
           if start = '1' and forw = '1' then
              state_fsm <= s_counting_forward;
              WTIME := WTIME_INIT;
              cnt := 0;
```

```
--if counting backwards, then set start from b"1111"
  elsif start = '1' and forw = '0' then
     state_fsm <= s_counting_backward;
     seconds <= (others => '1');
     WTIME := WTIME_INIT;
     cnt := WTIME;

  end if;

when s_counting_forward =>

  if double_speed = '1' and half_speed = '0' then
    WTIME := WTIME_INIT/2;
  elsif double_speed = '0' and half_speed = '1' then
    WTIME := WTIME_INIT*2;
  else
    WTIME := WTIME_INIT;
  end if;

  if cnt < WTIME then
     if stop = '1' then
        state_fsm <= s_paused;
     end if;
  elsif cnt = WTIME or cnt > WTIME then
     if seconds = b"1111" then --overflow
        state_fsm <= s_idle;
        seconds <= (others => '0');
        cnt := 0;
     else seconds <= seconds + 1;
          if forw = '1' then
             state_fsm <= s_counting_forward;
          elsif forw = '0' then
             state_fsm <= s_counting_backward;
          end if;
          cnt := 0;
     end if;
  end if;
  cnt := cnt + 1 ;
```

```
      when s_counting_backward =>
        if double_speed = '1' and half_speed = '0' then
          WTIME := WTIME_INIT/2;
        elsif double_speed = '0' and half_speed = '1' then
          WTIME := WTIME_INIT*2;
        else
          WTIME := WTIME_INIT;
        end if;

        if cnt > 0 then
           if stop = '1' then
              state_fsm <= s_paused;
           end if;
        elsif cnt = 0 then
           if seconds = b"0000" then --underflow
              state_fsm <= s_idle;
              seconds <= (others => '0');
              cnt := WTIME;
           else seconds <= seconds - 1;
                if forw = '1' then
                   state_fsm <= s_counting_forward;
                elsif forw = '0' then
                   state_fsm <= s_counting_backward;
                end if;
                cnt := WTIME;
           end if;
        end if;
        cnt := cnt - 1 ;


      when s_paused =>
        if start = '1' and forw = '1' then
           state_fsm <= s_counting_forward;
        elsif start = '1' and forw = '0' then
           state_fsm <= s_counting_backward;
        end if;

      end case;
  end if;
```

```
end process;

led_out <= std_logic_vector(seconds);
forw <= NOT(back_trig);

end Behavioral;
```