Indice

1.Introduzione

Nel progetto presentato di seguito si è deciso di implementare su scheda FPGA Altera Cycolne II EP2C20F484C7N un'applicazione che si interfacci con una tastiera ps2 (Chicony) ed uno schermo VGA. L'applicazione consiste nel muovere un 'serpente', formato da 4 quadrati da 10x10px colorati di rosso, tramite la pressione delle frecce della tastiera e mostrarlo a video insieme ad un secondo oggetto (un quadrato singolo 10x10px colorato di verde) che si muove nello schermo in modo autonomo seguendo le quattro direzioni diagonali e cambiando direzione ad ogni contatto con un bordo dello schermo. Quando il serpente colpisce con la 'testa' (ovvero il suo primo quadrato) il quadrato verde verrà assegnato un punto al giocatore. I punti verranno mostrati nei display 7 segmenti presenti sulla scheda.

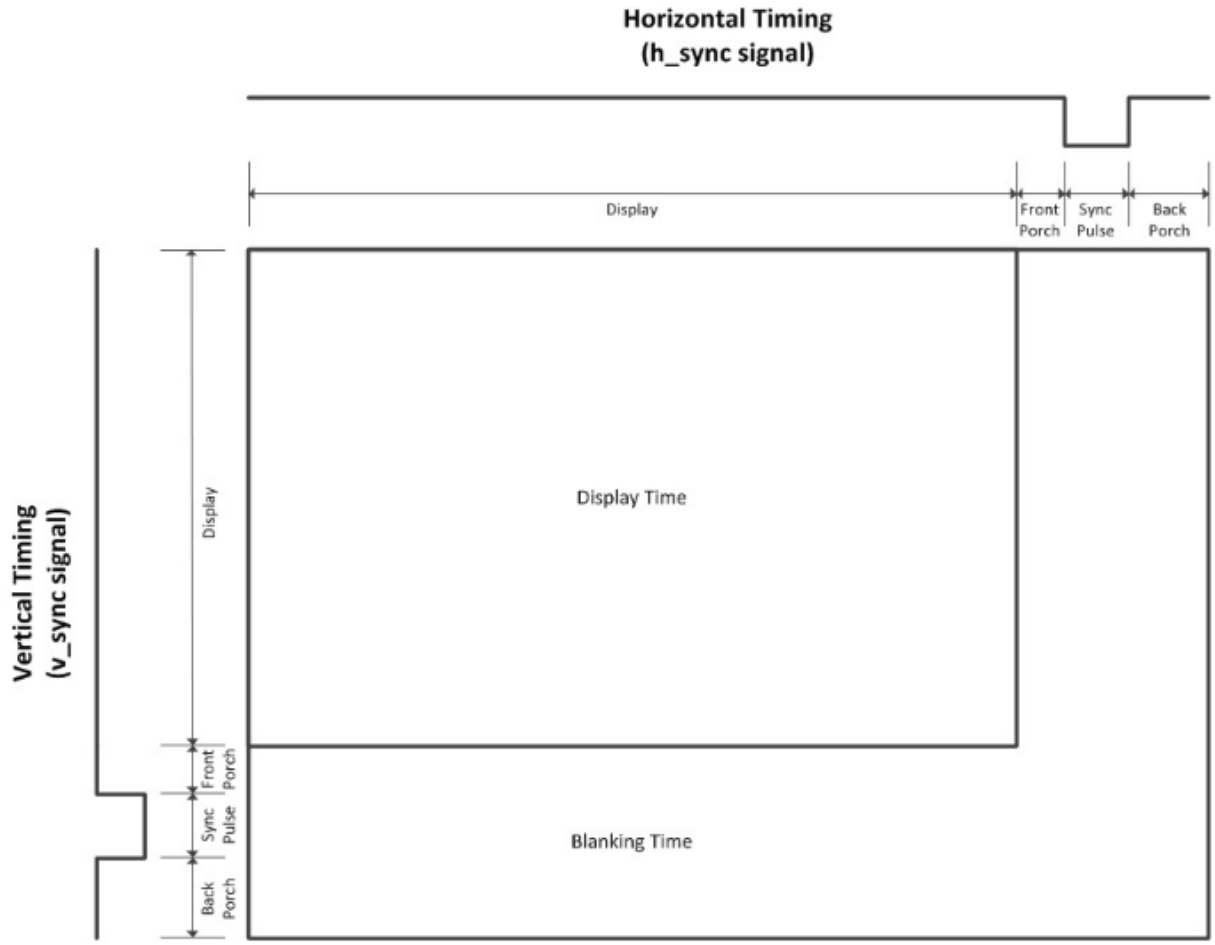
2. Breve cenno sui protocolli utilizzati per le interfacce

2.1 VGA

VGA è un'interfaccia standard per il controllo di monitor analogici. La connessione prevede 15 pin di cui 5 sono a massa, 4 sono dedicati a funzioni varie, 1 non è connesso ed i restanti 5, quelli di nostro interesse, sono 3 per il controllo dei colori R (Red), G (Green), B (Blue) entrambi analogici, verranno connessi all'uscita di un DAC, e 2 sono h_sync e v_sync. Per poter controllare questi segnali serve a monte del monitor un controller programmabile che generi le corrette forme d'onda necessarie per interfacciarsi al monitor. In particolare serve che al controller della VGA sia fornito il pixel clock il quale a sua volta userà per generare le forme d'onda h_sync e v_sync in modo corretto e ovviamente i segnali RGB. Questo pixel clock dipende dalla scelta di risoluzione secondo la seguente tabella:

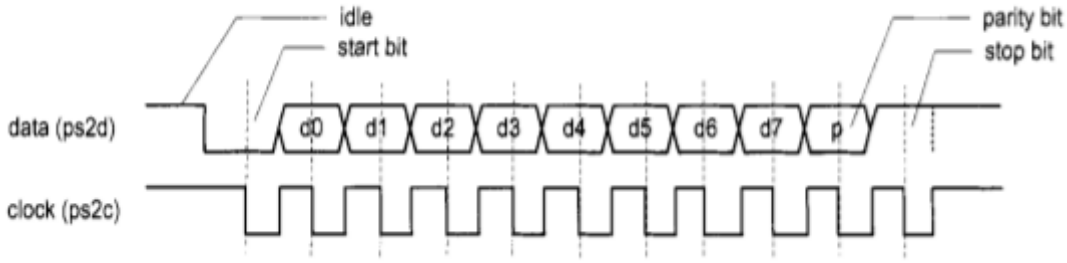
Resolution (pixels)	**Refresh Rate (Hz)**	**Pixel Clock (MHz)**	**Horizontal (pixel clocks)**				**Vertical (rows)**				**h_sync Polarity**	**v_sync Polarity**
			Display	**Front Porch**	**Sync Pulse**	**Back Porch**	**Display**	**Front Porch**	**Sync Pulse**	**Back Porch**		
640x350	70	25.175	640	16	96	48	350	37	2	60	p	n
640x350	85	31.5	640	32	64	96	350	32	3	60	p	n
640x400	70	25.175	640	16	96	48	400	12	2	35	n	p
640x400	85	31.5	640	32	64	96	400	1	3	41	n	p
640x480	60	25.175	640	16	96	48	480	10	2	33	n	n
640x480	73	31.5	640	24	40	128	480	9	2	29	n	n
640x480	75	31.5	640	16	64	120	480	1	3	16	n	n
640x480	85	36	640	56	56	80	480	1	3	25	n	n
…	…	…	…	…	…	…	…	…	…	…	…	…

Per comodità, dato che nella scheda scelta per il progetto è già fornito un Clock a 50MHz, si è scelta una risoluzione di 640x480px così che il pixel clock sia di circa 25MHz, metà della frequenza del clock a nostra disposizione e quindi facile da generare.
Affinché tutto funzioni correttamente i segnali di sincronizzazione devono seguire il seguente andamento temporale:



2.2 PS2

La porta PS2 introdotta da IBM è una periferica largamente utilizzata per la comunicazione tra personal computer e tastiera e mouse prima dell'avvento del protocollo USB. Prevede la comunicazione attraverso due cavi: uno per il flusso di dati l'altro per l'invio da parte della periferica del clock. Il dato è inviato in pacchetti da 11-bit dei quali: 1 start bit, 8 data bit, 1 bit di parità e 1 bit di stop. La comunicazione è di tipo seriale, poiché viene inviato un bit alla volta, sincrona, infatti entrambi i partecipanti alla comunicazione (host e periferica) sono sincronizzati sullo stesso clock, e bidirezionale, cioè ad inviare dati può essere sia la periferica sia l'host per settare alcuni parametri (nel nostro caso la comunicazione sarà unidirezionale). Il diagramma temporale è il seguente:



3. Struttura delle Entity

Le entity che compongono il progetto sono le seguenti:

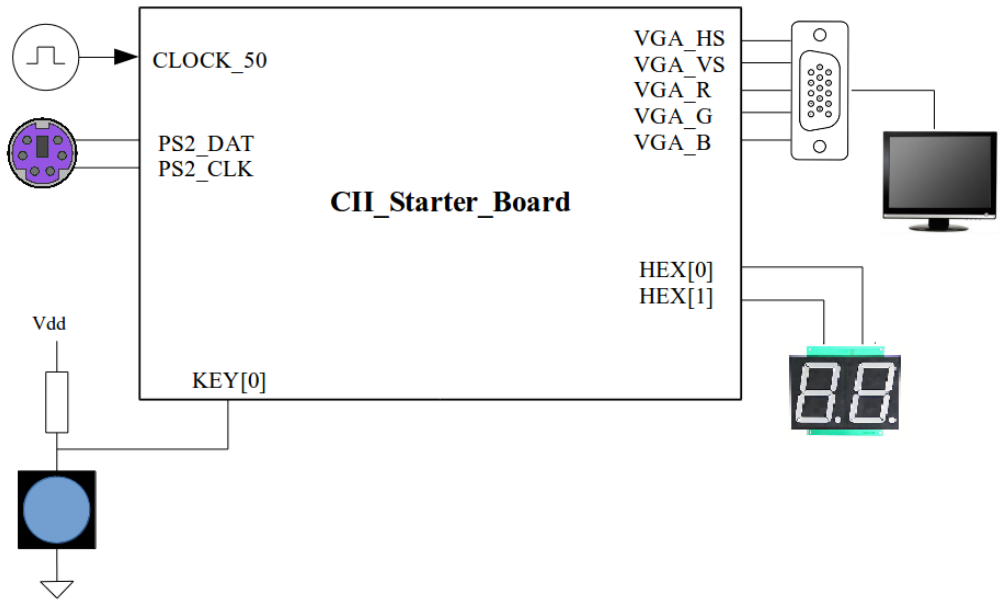
- *CII_Starter_Board*
- *FSM_position_per_snake*
- *FSM_position_apple*
- *PS2_port_receiver*
- *adjust_button*
- *digit_to_7seg*
- *dec_BCD*

3.1 CII_Starter_Board

Top_level_entity, gestisce il controllo della periferica VGA quindi la generazione di h_sync, v_sync, R, G e B, e del pixel clock. Inoltre viene gestito il contatto tra la testa del serpente ed il quadrato che si muove in modo indipendente e l'assegnamento dei punti. Viene anche regolata la velocità del serpente.

3.1.1 Schema connessioni interfacce

Le periferiche sono connesse alla scheda e ai segnali della entity nel seguente modo:



3.1.2 Codice VHDL

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.numeric_std.all;


entity CII_Starter_Board is
port(
                ----------      Clock Input                     ----------
                --CLOCK_24  : in std_logic_vector(1 downto 0); --      24 MHz
                --CLOCK_27  : in std_logic_vector(1 downto 0); --      27 MHz
                CLOCK_50  : in std_logic;                                       --      50 MHz
                --EXT_CLOCK : in std_logic;                                     --      External Clock

                ----------      Push Button                     ----------
                KEY : in std_logic_vector(3 downto 0);      --      Pushbutton[3:0]

                ----------      DPDT Switch                     ----------
                -- SW : in std_logic_vector(9 downto 0);     --      Toggle Switch[9:0]

                ----------      7-SEG Displa            ----------
                HEX0 : out std_logic_vector(6 downto 0);  --      Seven Segment Digit 0
                HEX1 : out std_logic_vector(6 downto 0);  --      Seven Segment Digit 1
                --HEX2 : out std_logic_vector(6 downto 0);  --      Seven Segment Digit 2
                --HEX3 : out std_logic_vector(6 downto 0)  --      Seven Segment Digit 3

                ----------      LED             ----------
                --LEDG : out std_logic_vector(6 downto 0);          --      LED Green[7:0]
                --LEDR : out std_logic_vector(6 downto 0);--      LED Red[9:0]

                ----------      UART    ----------
                --UART_TXD : out std_logic;                         --      UART Transmitter
                --UART_RXD : out std_logic;                         --      UART Receiver

                ----------/     SDRAM Interface         ----------
                --DRAM_DQ    : inout std_logic_vector(15 downto 0);--      SDRAM Data bus 16 Bits
                --DRAM_ADDR  : out std_logic_vector(11 downto 0);           --      SDRAM Address bus 12 Bits
                --DRAM_LDQM  : out std_logic;                               --      SDRAM Low-byte Data Mask
                --DRAM_UDQM  : out std_logic;                               -       SDRAM High-byte Data Mask
                --DRAM_WE_N  : out std_logic;                               --      SDRAM Write Enable
                --DRAM_CAS_N : out std_logic;                               --      SDRAM Column Address Strobe
                --DRAM_RAS_N : out std_logic;                               --      SDRAM Row Address Strobe
                --DRAM_CS_N  : out std_logic;                               --      SDRAM Chip Select
                --DRAM_BA_0  : out std_logic;                               --      SDRAM Bank Address 0
                --DRAM_BA_1  : out std_logic;                               --      SDRAM Bank Address 0
                --DRAM_CLK   : out std_logic;                               --      SDRAM Clock
                --DRAM_CKE   : out std_logic;                               --      SDRAM Clock Enable

                ----------      Flash Interface         ----------
                --FL_DQ    : inout std_logic_vector(7 downto 0);   --      FLASH Data bus 8 Bits
                --FL_ADDR  : out std_logic_vector(21 downto 0);     --      FLASH Address bus 22 Bits
                --FL_WE_N  : out std_logic;                         --      FLASH Write Enable
                --FL_RST_N : out std_logic;                         --      FLASH Reset
                --FL_OE_N  : out std_logic;                         --      FLASH Output Enable
                --FL_CE_N  : out std_logic;                         --      FLASH Chip Enable

                ----------      SRAM Interface          ----------
                --SRAM_DQ   : inout std_logic_vector(15 downto 0);          --SRAM Data bus 16 Bits
                --SRAM_ADDR : out std_logic_vector(17 downto 0);            --SRAM Address bus 18 Bits
                --SRAM_UB_N : out std_logic;                                --      SRAM High-byte Data Mask
                --SRAM_LB_N : out std_logic;                                --      SRAM Low-byte Data Mask
                --SRAM_WE_N : out std_logic;                                --      SRAM Write Enable
                --SRAM_CE_N : out std_logic;                                --      SRAM Chip Enable
                --SRAM_OE_N : out std_logic;                                --      SRAM Output Enable

                ----------      SD_Card Interface       ----------
                --SD_DAT  : inout std_logic;                                --      SD Card Data
                --SD_DAT3 : inout std_logic;                                --      SD Card Data 3
                --SD_CMD  : inout std_logic;                                --      SD Card Command Signal
                --SD_CLK  : inout std_logic;                        --      SD Card Clock

                ----------      USB JTAG link           ----------
                --TDI : in std_logic;                               -- CPLD -> FPGA (data in)
```

```
		--TCK : in std_logic;			-- CPLD -> FPGA (clk)
		--TCS : in std_logic;			-- CPLD -> FPGA (CS)
		--TDO : out std_logic;				-- FPGA -> CPLD (data out)

		--------------------	I2C		----------------------------
		--I2C_SDAT : inout std_logic;		--	I2C Data
		--I2C_SCLK : out std_logic;		--	I2C Clock

		--------------------	PS2		----------------------------
		PS2_DAT : in std_logic;				--	PS2 Data
		PS2_CLK : in std_logic;				--	PS2 Clock

		--------------------	VGA		----------------------------
		VGA_HS : out std_logic;				--	VGA H_SYNC
		VGA_VS : out std_logic;				--	VGA V_SYNC
		VGA_R  : out std_logic_vector(3 downto 0);	--	VGA Red[3:0]
		VGA_G  : out std_logic_vector(3 downto 0);	--	VGA Green[3:0]
		VGA_B  : out std_logic_vector(3 downto 0)	--	VGA Blue[3:0]

		----------------	Audio CODEC		-------------------------
		--AUD_ADCLRCK : out std_logic;			--	Audio CODEC ADC LR Clock
		--AUD_ADCDAT  : in std_logic;			--	Audio CODEC ADC Data
		--AUD_DACLRCK : out std_logic;			--	Audio CODEC DAC LR Clock
		--AUD_DACDAT  : out std_logic;			--	Audio CODEC DAC Data
		--AUD_BCLK    : in std_logic;			--	Audio CODEC Bit-Stream Clock
		--AUD_XCK     : out std_logic;			--	Audio CODEC Chip Clock

		--------------------	GPIO	----------------------------
		--GPIO_0 : inout std_logic_vector(35 downto 0);	--	GPIO Connection 0
		--GPIO_1 : inout std_logic_vector(35 downto 0)	--	GPIO Connection 1
	);
end CII_Starter_Board;



architecture A of CII_Starter_Board is
signal X,nX,Y,nY,position_X1,position_Y1,position_X2,position_Y2,position_X3,position_Y3,position_X4,position_Y4,apple_X,apple_Y:
unsigned(9 downto 0);
signal u_point,d_point: unsigned(3 downto 0);
signal cont,ncont: unsigned(2 downto 0);
signal point,npoint: unsigned(5 downto 0);
signal CK_25, nCK_25, TICK_ROW, TICK_END_DISPLAY, TICK_VEL, PULSE_VEL, RES, RES_point, PULSE_point, CK, res_keyboard:
std_logic;
signal data_keyboard: std_logic_vector(7 downto 0);

component FSM_position_per_snake is
	port(	RES: in std_logic;
		CK: in std_logic;
		keyboard: in std_logic_vector(7 downto 0);
		out_X1: out unsigned(9 downto 0);
		out_Y1: out unsigned(9 downto 0);
		out_X2: out unsigned(9 downto 0);
		out_Y2: out unsigned(9 downto 0);
		out_X3: out unsigned(9 downto 0);
		out_Y3: out unsigned(9 downto 0);
		out_X4: out unsigned(9 downto 0);
		out_Y4: out unsigned(9 downto 0);
		TICK_REFRESH: in std_logic
	);
end component;

component FSM_position_apple is
	port(	RES: in std_logic;
		CK: in std_logic;
		out_X: out unsigned(9 downto 0);
		out_Y: out unsigned(9 downto 0);
		TICK_REFRESH: in std_logic
		);
end component;

component PS2_port_reciver is
	port (
		clk, reset: in std_logic;
		ps2d, ps2c: in std_logic;
		rx_en: in std_logic;
```

```
                        rx_done_tick: out std_logic;
                        dout: out std_logic_vector(7 downto 0)
                );
end component;

component adjust_button is
   port( clk    : in std_logic;
         reset  : in std_logic;
         button : in  std_logic;
         pulse  : out std_logic
                );
end component;

component digit_to_7seg is
                port( digit : in unsigned(3 downto 0);
                                        seg   : out std_logic_vector(6 downto 0) );
end component;

component  unsigned2digits is
   port( n : in unsigned(5 downto 0);
         u : out unsigned(3 downto 0);
         d : out unsigned(3 downto 0) );
end component;


begin
                buf0: RES<=not(KEY(0));
                buf_CK: CK<=CLOCK_50;
                --================================--
                --REGOLAZIONE VELOCITA' SERPENTE--
                --================================--
                --creo un contatore che conta ogni volta che X e Y hanno finito lo schermo
                --ogni 8 volte che in CII_Starter_Board finisco di disegnare una schermata mando TICK_VEL='1'
                --TICK_VEL starebbe alto finchè non arriva un altro TICK_END_DISPLAY quindi lo rendo tramite adjust_button un solo impulso di
clock
                --a questo punto lo do in ingresso alla macchina a stati del serpente che mi va a fare da enable e aggiorna la posizione

                reg_cont_vel: process(RES,CK)
                                begin
                                                if RES='1' then
                                                                cont<=to_unsigned(0,3);
                                                elsif CK'event and CK='1' then
                                                                cont<=ncont;
                                                end if;
                end process;

                rc_vel: ncont<=cont+1 when TICK_END_DISPLAY='1' else cont;

                rc_TICK_vel: TICK_VEL<='1' when cont=to_unsigned(1,3) else '0';

                tick_vel_2_pulse: adjust_button port map( clk=> CK,
                                reset=> RES,
                                button=> TICK_VEL,
                                pulse=> PULSE_VEL);

                d1_keyboard: PS2_port_reciver port map( clk=> CK,
                                                                                                reset=> RES,
                                                                                                rx_en=> '1',
                                                                                                ps2c=> PS2_CLK,
                                                                                                ps2d=> PS2_DAT,
                                                                                               dout=> data_keyboard);

                d0_snake: FSM_position_per_snake port map( RES=>RES,
                                                                                                                CK=> CK,
                                                                                                                keyboard=> data_keyboard,
                                                                                                                TICK_REFRESH=> PULSE_VEL,
                                                                                                                out_X1 => position_X1,
                                                                                                                out_Y1 => position_Y1,
                                                                                                                out_X2 => position_X2,
                                                                                                                out_Y2 => position_Y2,
                                                                                                                out_X3 => position_X3,
                                                                                                                out_Y3 => position_Y3,
                                                                                                                out_X4 => position_X4,
                                                                                                                out_Y4 => position_Y4);

                d2_apple: FSM_position_apple port map(RES=> RES,
```

```
                                                CK=> CK,
                                                out_X=> apple_X,
                                                out_Y=> apple_Y,
                                                TICK_REFRESH=> TICK_END_DISPLAY);

        --=====================================================--
        --CONTROLLO CONTATTI SERPENTE-MELA E ASSEGNAMENTO PUNTI--
        --=====================================================--
        --se la testa del serpente tocca la mela --> +1
```

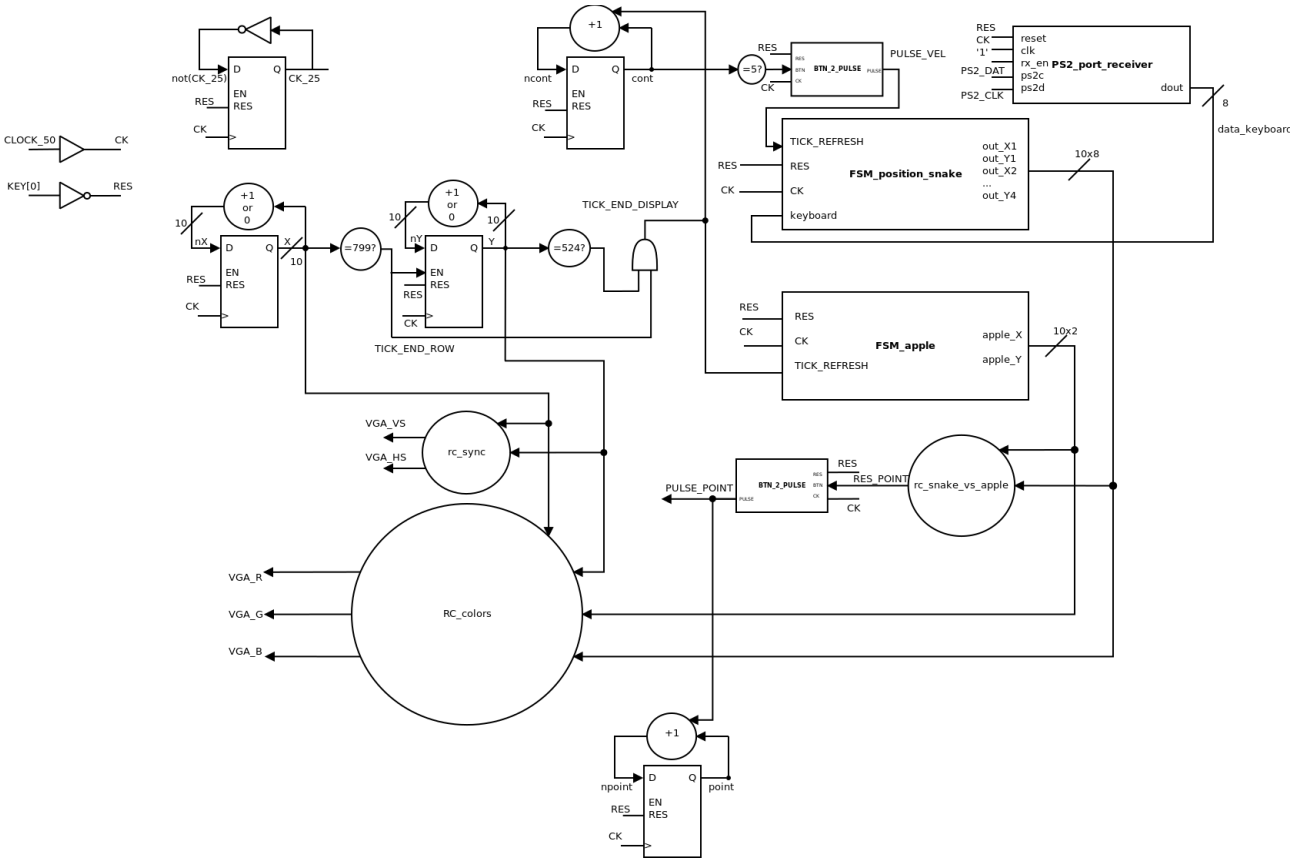
Per i contatti tra il serpente ed i quadrato si è deciso di considerarli tali quando le due posizioni X e le due Y di entrambi (testa e quadrato solitario) si trovano i un intorno di ±5 px. Si ricorda inoltre che le position X e Y sia della testa del serpente sia della mela sono riferite al centro, quando vengono visualizzate a video sono colorati 5 px prima e dopo in entrambe le direzioni in modo da fare un quadrato 10x10.

```
        rc_snake_vs_apple: process
(position_X1,position_X2,position_X3,position_X4,position_Y1,position_Y2,position_Y3,position_Y4,apple_X,apple_Y, point,npoint,RES_point)
                begin
                        if ((position_X1>apple_X-6 and position_X1<apple_X+6 and position_Y1>apple_Y-6 and
position_Y1<apple_Y+6)) then
                                RES_point<='1';
                        else
                                RES_point<='0';
                        end if;
                end process;

        d7: adjust_button port map( clk=> CK,
                                                                                reset=> RES,
                                                                                button=> RES_point,
                                                                                pulse=> PULSE_point);

        rc_point: npoint<=point+1 when PULSE_point='1' else point;

        reg_point: process(RES,CK)
                begin
                        if RES='1' then
                                point<=to_unsigned(0,6);
                        elsif CK'event and CK='1' then
                                point<=npoint;
                        end if;
        end process;

        --VISUALIZZO PUNTI NEI DISPLAY 7 SEG
        d3: unsigned2digits port map (n=> point,
                                        u=> u_point,
                                        d=> d_point);

        d4: digit_to_7seg port map (digit=> u_point,
                                        seg=> HEX0);

        d5: digit_to_7seg port map (digit=> d_point,
                                        seg=> HEX1);

        --==============================--
        --PARTE PER COLORAZIONE DISPLAY--
        --==============================--

        --Creazione CLOCK 25MHz, pixel_clock
        reg_Cloock: process(RES,CK)
                begin
                        if RES='1' then
                                CK_25<='0';
                        elsif CK'event and CK='1' then
                                CK_25<=nCK_25;
                        end if;
                end process;

        rc_clock: nCK_25<=not(CK_25);
```

Per sapere quando mettere i segnali h_sync e v_sync bassi o alti e per sapere quando colorare il display mi servono due contatori che mi indichino in che parte dello schermo mi trovo (X,Y). TICK_ROW mi indica quando ho finito di considerare una riga, TICK_END_DISPLAY mi indica quando ho finito di considerare una schermata.

```
	--contatore per coordinate X(0-->799) e Y(0-->524)
	reg_X: process(RES,CK_25)
		begin
			if RES='1' then
				X<=to_unsigned(0,10);
			elsif CK_25'event and CK_25='1' then
				X<=nX;
			end if;
		end process;

	rc_regX: nX<=to_unsigned(0,10) when X=799 else X+1;

	--creazione TICK di fine riga
	rc_TICK: TICK_ROW<='1' when X=799 else '0';

	reg_Y: process(RES,CK_25)
		begin
			if RES='1' then
				Y<=to_unsigned(0,10);
			elsif CK_25'event and CK_25='1' then
				if TICK_ROW='1' then
					Y<=nY;
				end if;
			end if;
		end process;

	rc_regY: nY<=to_unsigned(0,10) when Y=524 else Y+1;

	rc_TICK_END_DISPLAY: TICK_END_DISPLAY<='1' when Y=524 and X=799 else '0';


	rc_colors:
process(X,Y,position_X1,position_X2,position_X3,position_X4,position_Y1,position_Y2,position_Y3,position_Y4,apple_X,apple_Y)
		begin
			if (	(X>position_X1-5 and X<position_X1+5 and Y>position_Y1-5 and Y<position_Y1+5) or
				(X>position_X2-5 and X<position_X2+5 and Y>position_Y2-5 and Y<position_Y2+5) or
				(X>position_X3-5 and X<position_X3+5 and Y>position_Y3-5 and Y<position_Y3+5) or
				(X>position_X4-5 and X<position_X4+5 and Y>position_Y4-5 and Y<position_Y4+5)) then
				VGA_R<=std_logic_vector(to_unsigned(15,4));
				VGA_B<=std_logic_vector(to_unsigned(0,4));
				VGA_G<=std_logic_vector(to_unsigned(0,4));		--COLORAZIONE SERPENTE
			elsif (X>(apple_X-5) and X<(apple_X+5) and Y>(apple_Y-5) and Y<(apple_Y+5)) then
				VGA_R<=std_logic_vector(to_unsigned(0,4));
				VGA_B<=std_logic_vector(to_unsigned(0,4));
				VGA_G<=std_logic_vector(to_unsigned(15,4));		--COLORAZIONE MELA
			else
				VGA_R<=std_logic_vector(to_unsigned(0,4));
				VGA_B<=std_logic_vector(to_unsigned(0,4));
				VGA_G<=std_logic_vector(to_unsigned(0,4));
			end if;
		end process;

	--RC PER LA GENERAZIONE DEI SEGNALI DI SINCRONISMO PER VGA
	rc_sync: process(X,Y)
		begin
			if (X>655 and X<752) then
				VGA_HS<='0';
			else
				VGA_HS<='1';
			end if;
			if (Y>489 and Y<492) then
				VGA_VS<='0';
			else
				VGA_VS<='1';
			end if;
		end process;


end A;
```

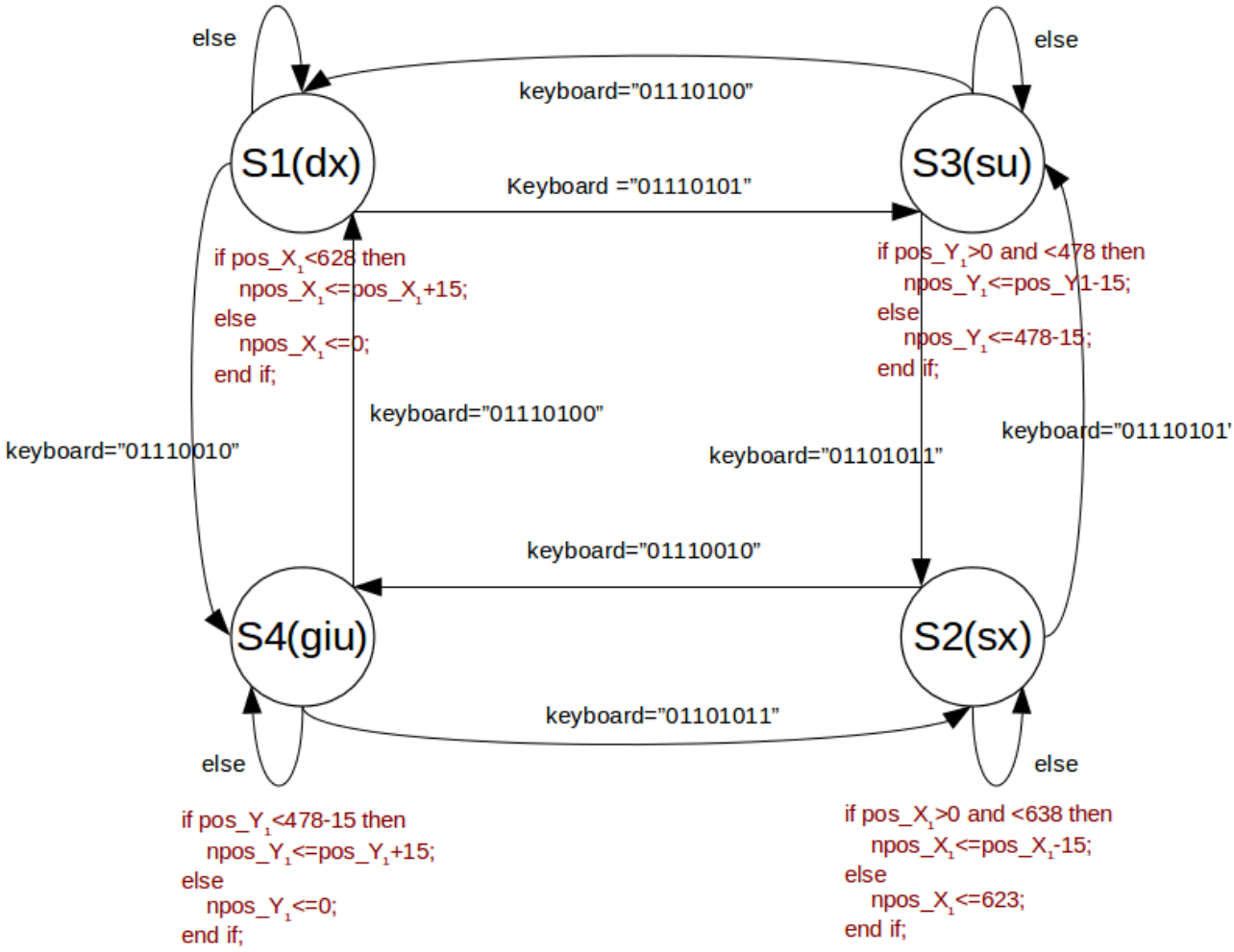
3.1.3 Schema RTL



3.2 FSM_postion_per_snake

In questa entity viene regolato il movimento del serpente che dipende dalla sua posizione attuale, dalla sua direzione (stato della macchina a stati) e dai tasti premuti dalla tastiera. Ha come uscite le posizioni X e Y dei singoli quadrati che compongono il serpente.

3.2.1 Schema macchina a stati



Gli stati definiscono le quattro diverse direzioni di spostamento:

- S1 → destra;
- S2 → sinistra;
- S3 → alto;
- S4 → basso.

La pressione di una freccia nella tastiera fa si che nel segnale keyboard sia presente un codice che identifica quale freccia è stata premuta, questo cambierà o meno lo stato della macchina a stati e quindi il movimento del serpente. Se in uno stato si raggiunge la fine dello schermo il serpente ricompare dall'altro lato. Le posizioni X e Y dei restanti 3 quadrati seguono a mo di scia il quadrato antecedente (ad es. npos_X2<=pos_X1; npos_Y2<=pos_Y1; e così via per tutti gli altri).

3.2.2 Codice VHDL

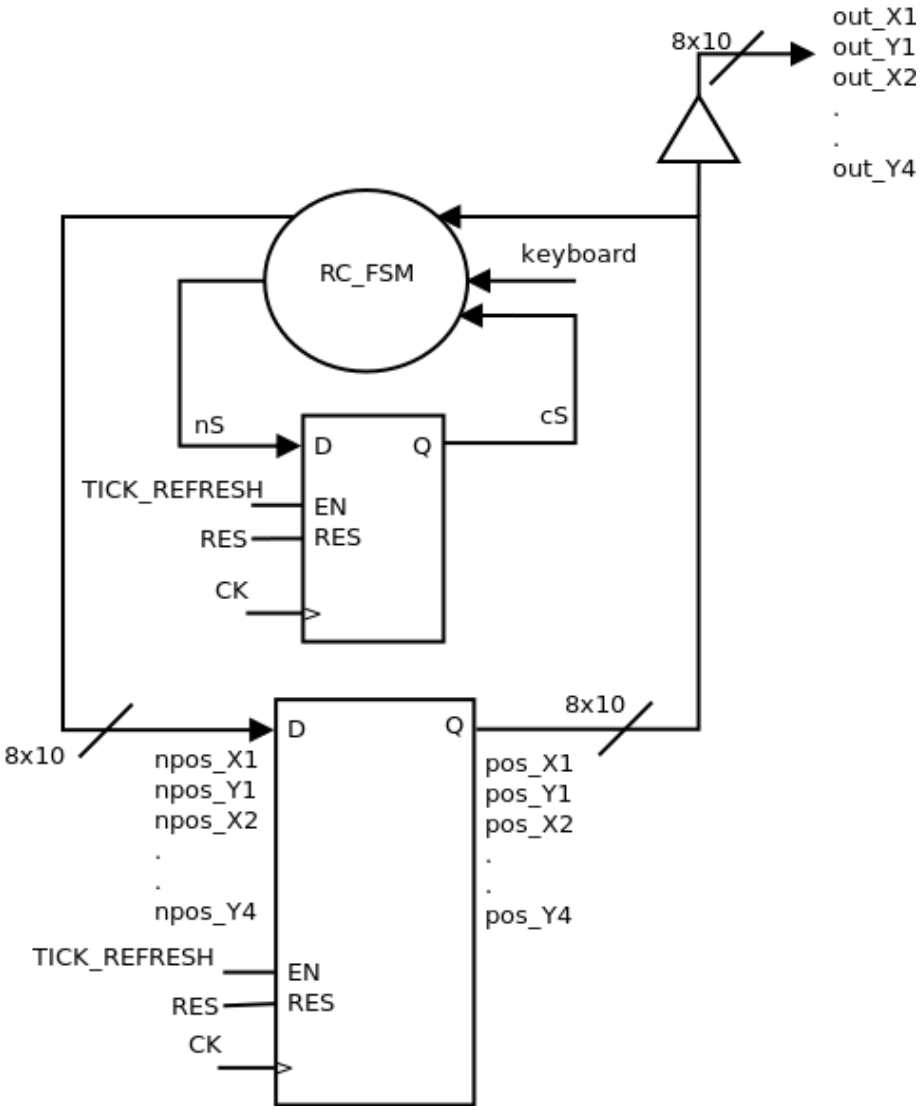
```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.numeric_std.all;

entity FSM_position_per_snake is
port(       RES: in std_logic;
            CK: in std_logic;
            keyboard: in std_logic_vector(7 downto 0);
            out_X1: out unsigned(9 downto 0);
```

```
		out_Y1: out unsigned(9 downto 0);
		out_X2: out unsigned(9 downto 0);
		out_Y2: out unsigned(9 downto 0);
		out_X3: out unsigned(9 downto 0);
		out_Y3: out unsigned(9 downto 0);
		out_X4: out unsigned(9 downto 0);
		out_Y4: out unsigned(9 downto 0);
		TICK_REFRESH: in std_logic
	);
end FSM_position_per_snake;

architecture A of FSM_position_per_snake is
type STATO is (S1,S2,S3,S4);
signal nS,cS: STATO;
signal npos_X1, npos_Y1, pos_X1, pos_Y1, npos_X2, npos_Y2, pos_X2, pos_Y2,
				npos_X3, npos_Y3, pos_X3, pos_Y3, npos_X4, npos_Y4, pos_X4, pos_Y4: unsigned(9 downto 0);

begin
	reg_FSM: process(RES,CK)
		begin
			if RES='1' then
				pos_X1<=to_unsigned(50,10);
				pos_Y1<=to_unsigned(5,10);
				pos_X2<=to_unsigned(35,10);
				pos_Y2<=to_unsigned(5,10);
				pos_X3<=to_unsigned(20,10);
				pos_Y3<=to_unsigned(5,10);
				pos_X4<=to_unsigned(5,10);
				pos_Y4<=to_unsigned(5,10);
				cS<=S1;
			elsif CK'event and CK='1' then
				if TICK_REFRESH='1' then
					cS<=nS;
					pos_X1<=npos_X1;
					pos_Y1<=npos_Y1;
					pos_X2<=npos_X2;
					pos_Y2<=npos_Y2;
					pos_X3<=npos_X3;
					pos_Y3<=npos_Y3;
					pos_X4<=npos_X4;
					pos_Y4<=npos_Y4;
				end if;
			end if;
		end process;

	rc_FSM:	process(cS, pos_X1, pos_Y1, pos_X2, pos_Y2, pos_X3, pos_Y3, pos_X4, pos_Y4, keyboard)
		begin
			case cS is
				when S1=>  npos_Y1<=pos_Y1;
							npos_X2<=pos_X1;
							npos_Y2<=pos_Y1;
							npos_X3<=pos_X2;
							npos_Y3<=pos_Y2;
							npos_X4<=pos_X3;
							npos_Y4<=pos_Y3;
							if pos_X1<(638-10) then                                --direzione destra
								npos_X1<=pos_X1+15;
							else
								npos_X1<=to_unsigned(0,10);

							end if;
							if keyboard="01110101" then --premo su
									nS<=S3;
							elsif keyboard="01110010" then   --premo giu
									nS<=S4;
							else
									nS<=S1;
							end if;

				when S2=>  npos_Y1<=pos_Y1;
							npos_X2<=pos_X1;
							npos_Y2<=pos_Y1;
							npos_X3<=pos_X2;
							npos_Y3<=pos_Y2;
							npos_X4<=pos_X3;
							npos_Y4<=pos_Y3;
```

```
                                        if pos_X1>0 and pos_X1<638 then                 --direzione sx
                                                npos_X1<=pos_X1-15;
                                        else
                                                npos_X1<=to_unsigned(638-15,10);

                                        end if;
                                        if keyboard="01110101" then --premo su
                                                nS<=S3;
                                        elsif keyboard="01110010" then   --premo giu
                                                nS<=S4;
                                        else
                                                nS<=S2;
                                        end if;

                when S3=>  npos_X1<=pos_X1;
                                        npos_X2<=pos_X1;
                                        npos_Y2<=pos_Y1;
                                        npos_X3<=pos_X2;
                                        npos_Y3<=pos_Y2;
                                        npos_X4<=pos_X3;
                                        npos_Y4<=pos_Y3;
                                        if pos_Y1>0 and pos_Y1<478 then                 --direzione alto
                                                npos_Y1<=pos_Y1-15;
                                        else
                                                npos_Y1<=to_unsigned(478-15,10);

                                        end if;
                                        if keyboard="01110100" then --premo dx
                                                nS<=S1;
                                        elsif keyboard="01101011" then --premo sx
                                                nS<=S2;
                                        else
                                                nS<=S3;
                                        end if;


                when S4=>  npos_X1<=pos_X1;
                                        npos_X2<=pos_X1;
                                        npos_Y2<=pos_Y1;
                                        npos_X3<=pos_X2;
                                        npos_Y3<=pos_Y2;
                                        npos_X4<=pos_X3;
                                        npos_Y4<=pos_Y3;
                                        if pos_Y1<478-15 then                   --direzione basso
                                                npos_Y1<=pos_Y1+15;
                                        else
                                                npos_Y1<=to_unsigned(0,10);

                                        end if;
                                        if keyboard="01110100" then --premo dx
                                                nS<=S1;
                                        elsif keyboard="01101011" then --premo sx
                                                nS<=S2;
                                        else
                                                nS<=S4;
                                        end if;

                when others => nS<=S1;
                                                npos_X1<=pos_X1;
                                          npos_Y1<=pos_Y1;
                                          npos_X2<=pos_X2;
                                          npos_Y2<=pos_Y2;
                                          npos_X3<=pos_X3;
                                          npos_Y3<=pos_Y3;
                                          npos_X4<=pos_X4;
                                          npos_Y4<=pos_Y4;
                end case;

end process;

buf1_x: out_X1<=pos_X1;
buf1_y: out_Y1<=pos_Y1;
buf2_x: out_X2<=pos_X2;
buf2_y: out_Y2<=pos_Y2;
buf3_x: out_X3<=pos_X3;
buf3_y: out_Y3<=pos_Y3;
```

```
  buf4_x: out_X4<=pos_X4;
  buf4_y: out_Y4<=pos_Y4;
end A;
```

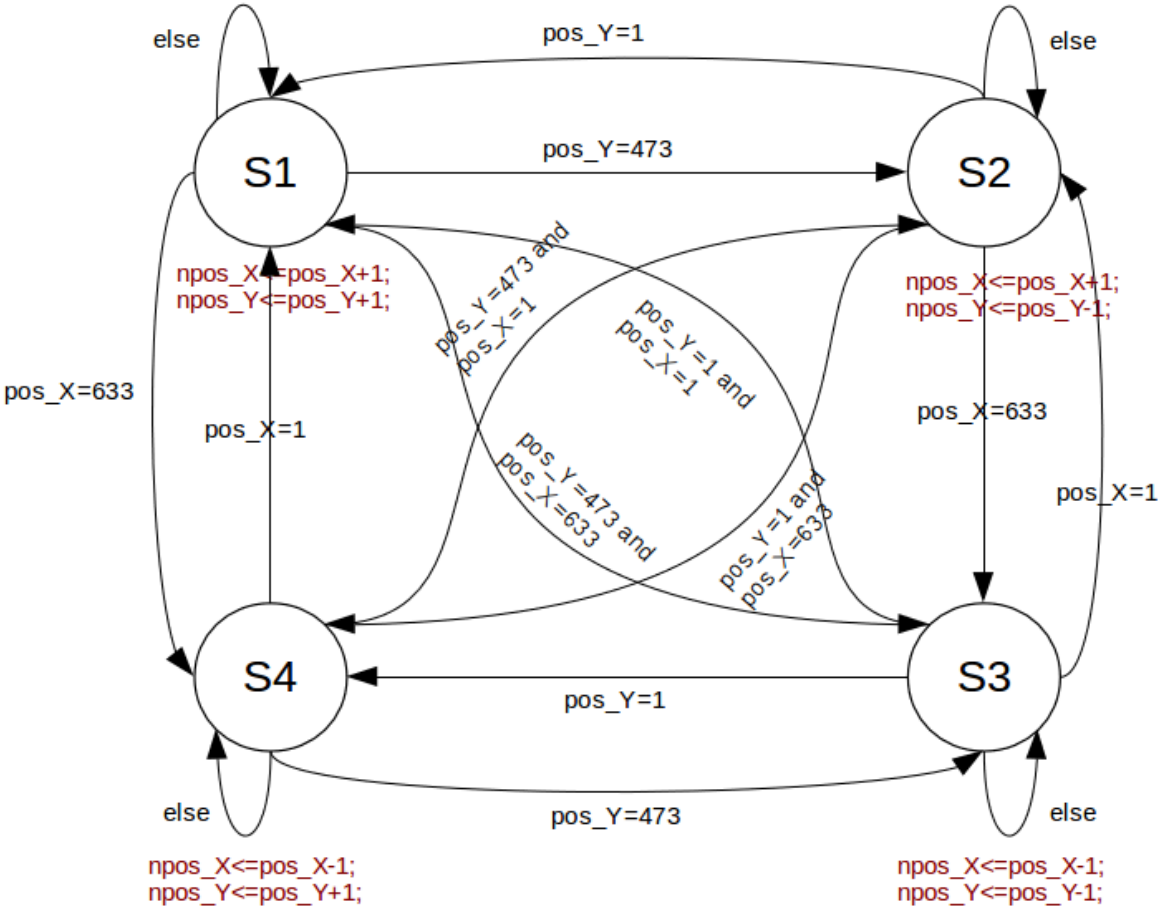
3.2.3 Schema RTL



3.3 FSM_postion_apple

Qui si regola il movimento e la posizione del quadrato che si muove in modo indipendente. Ha come uscite le posizioni attuali X e Y del quadrato.

3.3.1 Schema Macchina a stati



Gli stati definiscono le quattro diverse direzioni di spostamento:

- S1 → basso-destra;
- S2 → alto-destra;
- S3 → alto-sinistra;
- S4 → basso-sinistra.

Le condizioni di cambiamento di stato sono gli impatti del quadrato contro una delle pareti o uno spigolo. A tal fine si ricorda che la parte di schermo visualizzabile è X(0..639) e Y(0..479) e che il quadrato ha lato 10px (pos_X+5, pos_X-5; pos_Y+5, pos_Y-5).

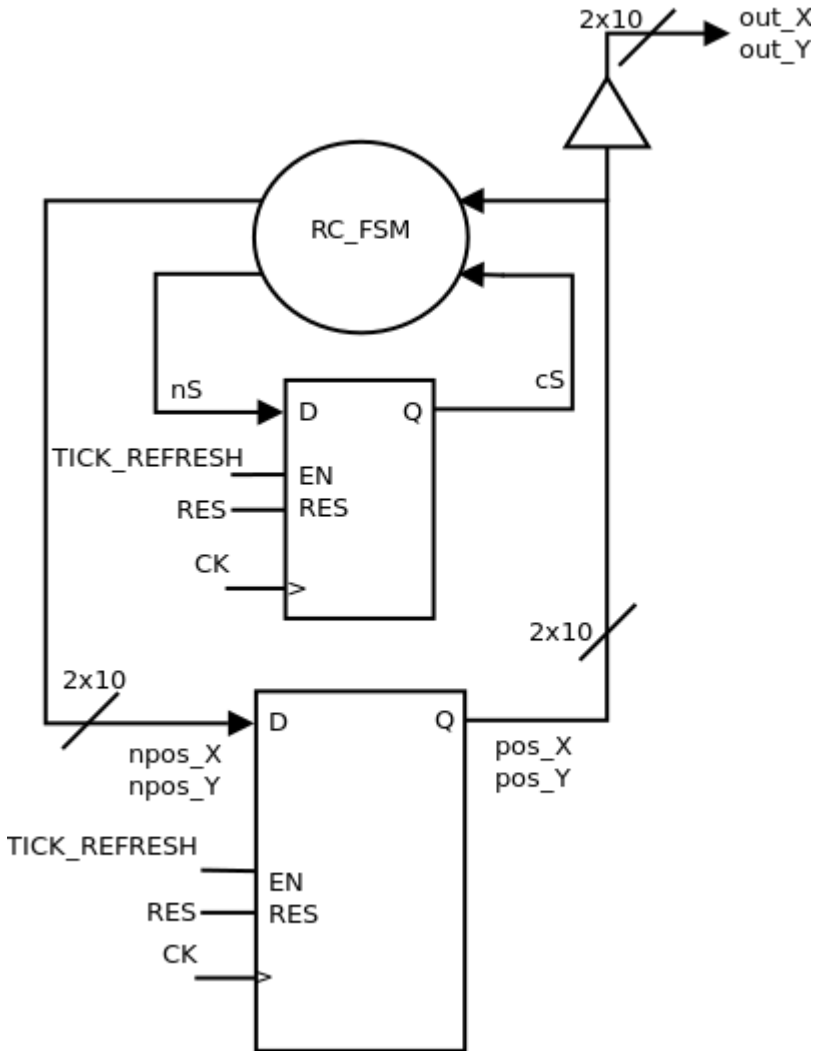
3.3.2 Codice VHDL

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.numeric_std.all;

entity FSM_position_apple is
port(           RES: in std_logic;
                CK: in std_logic;
                out_X: out unsigned(9 downto 0);
                out_Y: out unsigned(9 downto 0);
                TICK_REFRESH: in std_logic
               );
end FSM_position_apple;

architecture A of FSM_position_apple is
type STATO is (S1,S2,S3,S4);
```

```
signal nS,cS: STATO;
signal npos_X, npos_Y, pos_X, pos_Y: unsigned(9 downto 0);

begin
	reg_FSM: process(RES,CK)
		begin
			if RES='1' then
				pos_Y<=to_unsigned(473,10);
				pos_X<=to_unsigned(316,10);
				cS<=S1;
			elsif CK'event and CK='1' then
				if TICK_REFRESH='1' then
					cS<=nS;
					pos_X<=npos_X;
					pos_Y<=npos_Y;
				end if;
			end if;
		end process;


	rc_FSM:	process(cS, pos_X, pos_Y)
		begin
			case cS is
				when S1=>  npos_X<=pos_X+1;
					npos_Y<=pos_Y+1;		--direzione basso-destra
					if pos_Y>=478-5 then  --urto parete in basso
						nS<=S2;
					elsif pos_X>=638-5 and pos_Y>=478-5 then --urto spigolo basso destra
						nS<=S3;
					elsif pos_X>=638-5 then   -- urto parete destra
						nS<=S4;
					else
						nS<=S1;
					end if;

				when S2=>  npos_X<=pos_X+1;
					npos_Y<=pos_Y-1;		--direzione alto-destra
					if pos_Y<=1 then  --urto parete alta
						nS<=S1;
					elsif pos_X>=638-5 and pos_Y<=1 then   -- urto spigolo alto destra
						nS<=S4;
					elsif pos_X>=638-5 then    --urto parete destra
						nS<=S3;
					else
						nS<=S2;
					end if;

				when S3=>  npos_X<=pos_X-1;
					npos_Y<=pos_Y-1;		--direzione alto-sinistra
					if pos_Y<=1 then
						nS<=S4;
					elsif pos_X<=1 and pos_Y<=1 then
						nS<=S1;
					elsif pos_X<=1 then
						nS<=S2;
					else
						nS<=S3;
					end if;

				when S4=>  npos_X<=pos_X-1;
					npos_Y<=pos_Y+1;		--direzione basso-sinistra
					if pos_Y>=478-5 then
						nS<=S3;
					elsif pos_X<=1 and pos_Y>=478-5 then
						nS<=S2;
					elsif pos_X<=1 then
						nS<=S1;
					else
						nS<=S4;
					end if;

				when others => nS<=S1;
						npos_X<=pos_X;
						npos_Y<=pos_Y;
				end case;
```

```
  end process;

  buf0: out_X<=pos_X;
  buf1: out_Y<=pos_Y;

end A;
```

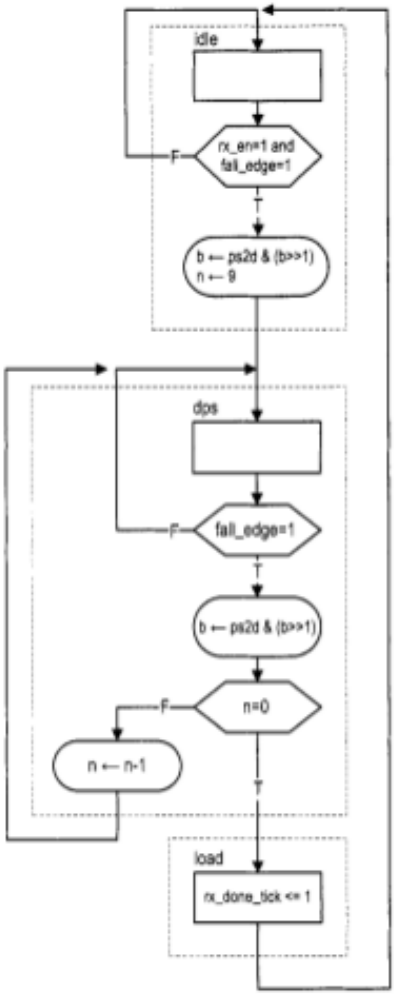
3.3.3 Schema RTL



3.4 PS2_port_receiver

Per questa parte si è fatto riferimento al documento FPGA Prototyping by VHDL Examples Xilinx Spartan 3 nel quale è illustrato il funzionamento della porta ps2 e un esempio di codice VHDL il quale è stato utilizzato per realizzare questa parte di progetto.
Questa entity controlla i dati che vengono ricevuti dalla tastiera e li mette a disposizione della top_level_entity.

3.4.1 Schema macchina a stati



I tre stati identificano tre diverse situazioni:

- idle: qui la macchina a stati se è abilitata la ricezione (rx_en='1') aspetta che venga rivelato un fronte di discesa su ps2c, per farlo viene utilizzato un filtro che fa in modo che un eventuale rumore non venga identificato come fronte di discesa. Non appena viene rilevato si mette il dato in ps2d (start bit) in uno shift register b;
- dps: ogni volta che viene rilevato un fronte di discesa si mette il dato in b e questo viene fatto shiftare di un bit, tutto ciò viene ripetuto 10 volte (dato + parità + stop bit);
- load: viene messo alto il segnale rx_done che segnala l'avvenuta ricezione e va direttamente in idle.

Il dato che ora è contenuto negli 8 bit centrali di b viene messo in dout e portato fuori dall'entity.

3.4.2 Codice VHDL

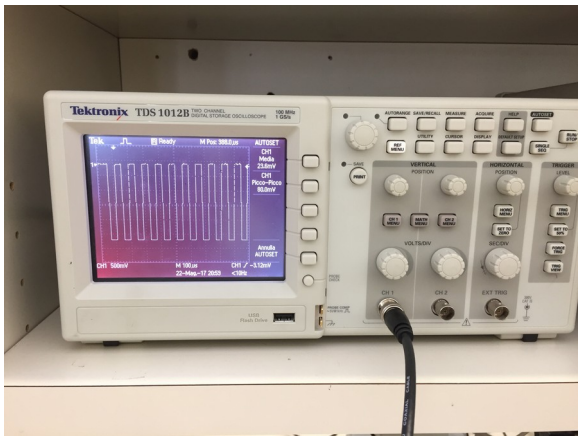
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PS2_port_reciver is
	port (
		clk, reset: in std_logic;
		ps2d, ps2c: in std_logic;
```

```
                rx_en: in std_logic;
                rx_done_tick: out std_logic;
                dout: out std_logic_vector(7 downto 0)
        );
        end PS2_port_reciver;

architecture A of PS2_port_reciver is
        type statetype is (idle, dps, load);
        signal state_reg, state_next: statetype;
        signal filter_reg, filter_next: std_logic_vector(7 downto 0);
        signal f_ps2c_reg, f_ps2c_next: std_logic;
        signal b_reg, b_next: std_logic_vector(10 downto 0);
        signal n_reg, n_next: unsigned(3 downto 0);
        signal fall_edge: std_logic;
begin
--=============================================================================
--filter and falling edge tick generation for ps2c
--=============================================================================
process(clk,reset)
begin
        if reset='1' then
                filter_reg <= (others=>'0');
                f_ps2c_reg <='0';
        elsif (clk'event and clk='1') then
                filter_reg <= filter_next;
                f_ps2c_reg <= f_ps2c_next;
        end if;
end process;

filter_next <= ps2c & filter_reg(7 downto 1);  --concatenazione

f_ps2c_next <= '1' when filter_reg="11111111" else
                '0' when filter_reg="00000000" else
                f_ps2c_reg;

fall_edge <= f_ps2c_reg and (not f_ps2c_next);  --filtro i fronti di discesa

--=============================================================================
--fsdm to extract the 8bit data
--=============================================================================

--Definizione dei registri
process(clk, reset)
begin
        if reset='1' then
                state_reg <= idle;
                n_reg <= (others => '0');
                b_reg <= (others => '0');
        elsif (clk'event and clk='1') then
                state_reg <= state_next;
                n_reg <= n_next;
                b_reg <= b_next;
        end if;
end process;

--Definizione delle logiche + Macchina a Stati
process(state_reg, n_reg, b_reg, fall_edge, rx_en, ps2d)
begin
        rx_done_tick <= '0';
        state_next <= state_reg;
        n_next <= n_reg;
        b_next <= b_reg;

        case state_reg is
                when idle =>
                        if fall_edge='1' and rx_en='1' then
                                b_next <= ps2d & b_reg(10 downto 1);        --concateno il primo bit del dato --> b funge da shift register
                                n_next <= "1001";                       --9
                                state_next <= dps;
                        end if;

                when dps =>
                        if fall_edge='1' then
                                b_next <= ps2d & b_reg(10 downto 1);        --ad ogni fall_edge (cioè clock del ps2) shifto di uno b e
inserisco nel LSB il bit in ps2d
                                        if n_reg = 0 then
```

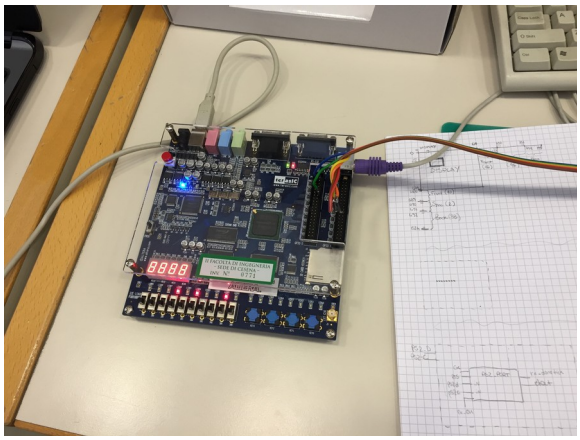
```
                                        state_next <= load;
                                    else
                                        n_next <= n_reg-1;          --lo faccio per 10 volte
                                    end if;
                            end if;

                    when load =>
                            state_next <= idle;
                            rx_done_tick <= '1';
            end case;
end process;

--Sparo fuori gli 8 bit
dout <= b_reg(8 downto 1);          --Questi sono i dati
end A;
```

3.4.3 Estrazione codici frecce tastiera

(Questa parte è stata svolta in collaborazione con Devis Massari, collega al terzo anno di ingegneria elettronica per l'energia e l'informazione.)
E' stata creata una nuova top_level_entity per la prova del programma sopra descritto e l'estrazione dei codici che la tastiera invia per ogni tasto premuto. In questa entity si è portato l'ingresso da tastiera, PS2_CLK, su un pin GPIO della scheda e sui led il dato ricevuto, cioè dout della entity PS2_port_receiver.



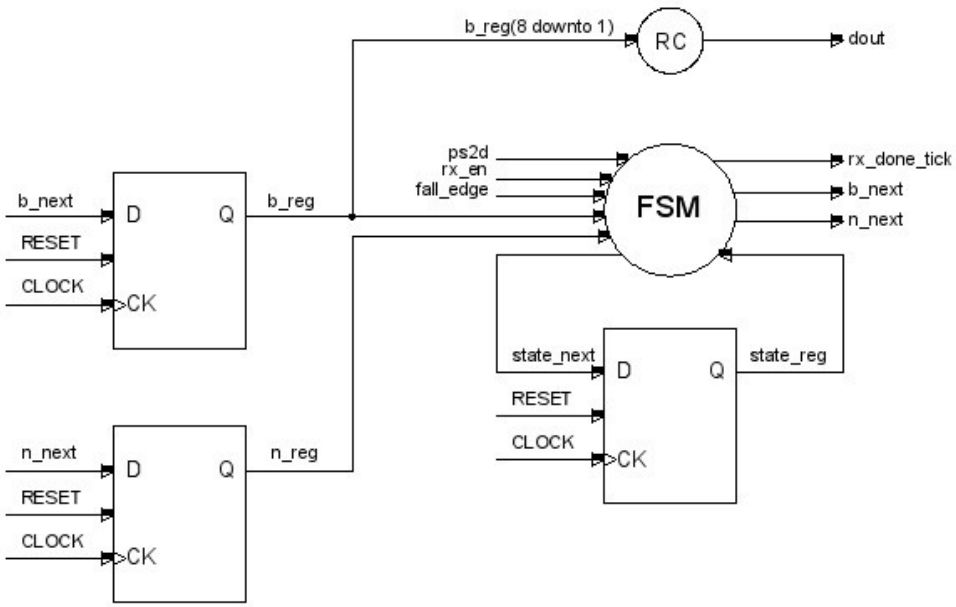
Clock da tastiera



Collegamento scheda FPGA

Per verificarne la validità è stato portato il pin GPIO ad un oscilloscopio e si è visualizzata la forma d'onda. I led rossi accesi rappresentano i bit a 1 del dato a 8 bit ricevuto dalla tastiera.

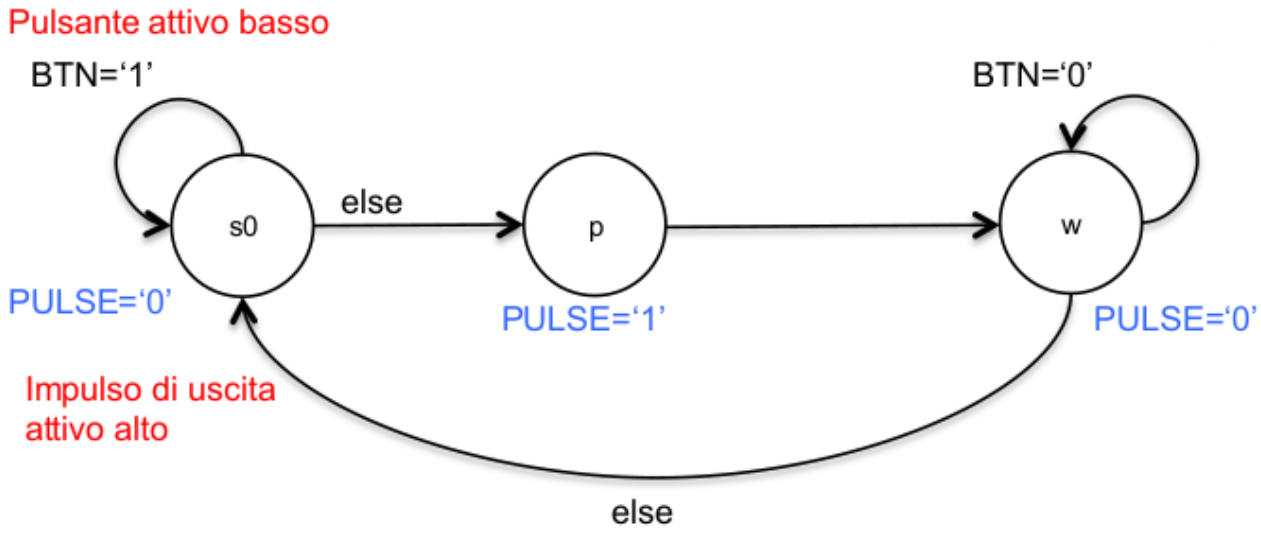
3.4.3 Schema RTL



3.5 Adjust_button

Questa entity è stata sviluppata durante le esercitazioni del corso con il professore. Converte la pressione di un pulsante o comunque lo stato prolungato di un segnale al valore logico alto, in un impulso di durata 1 Tclock.

3.5.1 Schema Macchina a stati

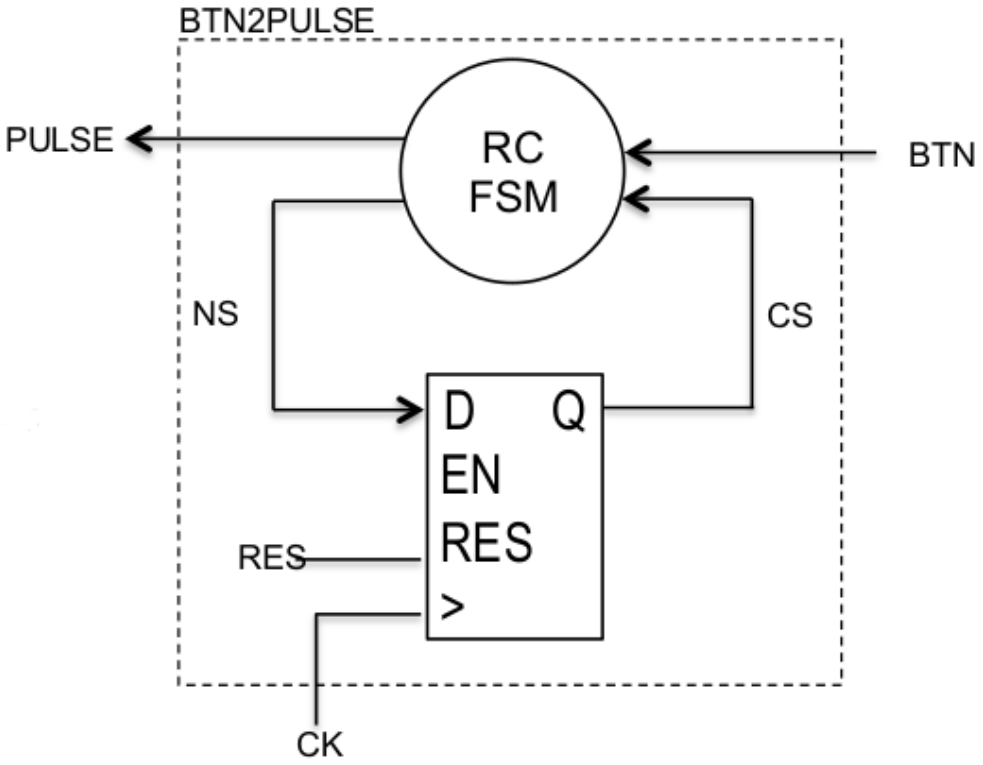


3.5.2 Codice VHDL

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.numeric_std.all;

entity adjust_button is
  port( clk    : in std_logic;
```

```
        reset  : in std_logic;
        button : in  std_logic;
        pulse  : out std_logic );
end adjust_button;

architecture A of adjust_button is
type stato is (s1, s2, s3, s4);
signal cs, ns : stato;
signal pulse_in : std_logic;

begin

  process(reset,clk)
  begin
    if reset='1' then
        cs <= s1;
        pulse <= '0';
    elsif clk'event and clk='1' then
        cs <= ns;
        pulse <= pulse_in;
    end if;
  end process;


  process(cs, button)
  begin
    case cs is
      when s1 =>
          pulse_in <= button;
          if button='1' then
            ns <= s2;
          else
            ns <= s1;
          end if;

      when s2 =>
          pulse_in <= '0';
          if button='1' then
            ns <= s2;
          else
            ns <= s3;
          end if;

      when s3 =>
          pulse_in <= button;
          if button='1' then
            ns <= s4;
          else
            ns <= s3;
          end if;

      when s4 =>
          pulse_in <= '0';
          if button='1' then
            ns <= s4;
          else
            ns <= s1;
          end if;

      when others =>
          pulse <= '0';
          ns <= s1;
    end case;
  end process;

end A;
```

3.5.3 Schema RTL



3.6 digit_to_7seg

Dato in ingresso un unsigned porta in uscita il suo codice in 7seg pronto per essere mostrato in un display 7segmenti.

3.6.1 Codice VHDL

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.numeric_std.all;

entity digit_to_7seg is
  port( digit : in unsigned(3 downto 0);
        seg   : out std_logic_vector(6 downto 0) );
end digit_to_7seg;

architecture A of digit_to_7seg is
begin

   process(digit)
   begin
      case digit is
         when to_unsigned(0,4) => seg <= "1000000";
         when to_unsigned(1,4) => seg <= "1111001";
         when to_unsigned(2,4) => seg <= "0100100";
         when to_unsigned(3,4) => seg <= "0110000";
         when to_unsigned(4,4) => seg <= "0011001";
         when to_unsigned(5,4) => seg <= "0010010";
         when to_unsigned(6,4) => seg <= "0000010";
         when to_unsigned(7,4) => seg <= "1111000";
         when to_unsigned(8,4) => seg <= "0000000";
         when to_unsigned(9,4) => seg <= "0010000";
         when others           => seg <= "0000110";
      end case;
```

```
  end process;


end A;
```

3.7 dec_BCD

Converte un numero binario da 6 bit (0..63) in due numeri che ne identifico unità e decine.

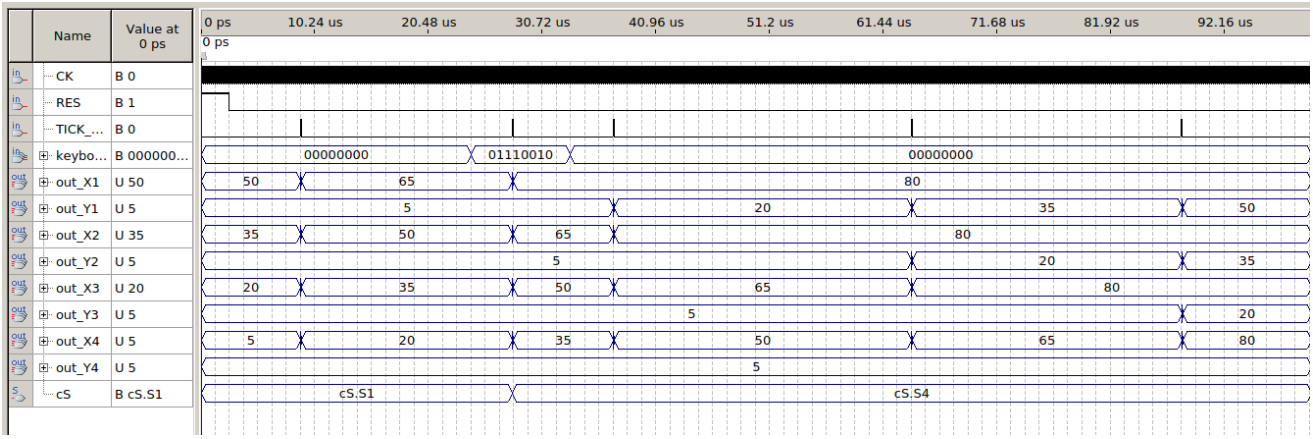
3.7.1 Codice VHDL

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.numeric_std.all;

entity unsigned2digits is
   port( n : in unsigned(5 downto 0);
         u : out unsigned(3 downto 0);
         d : out unsigned(3 downto 0) );
end unsigned2digits;


architecture A of unsigned2digits is
begin

   process(n)
   begin
      if n<10 then
         d <= to_unsigned(0,4);
         u <= n(3 downto 0);
      elsif n<20 then
         d <= to_unsigned(1,4);
         u <= resize(n-10,4);
      elsif n<30 then
         d <= to_unsigned(2,4);
         u <= resize(n-20,4);
      elsif n<40 then
         d <= to_unsigned(3,4);
         u <= resize(n-30,4);
      elsif n<50 then
         d <= to_unsigned(4,4);
         u <= resize(n-40,4);
      elsif n<60 then
         d <= to_unsigned(5,4);
         u <= resize(n-50,4);
      elsif n<70 then
         d <= to_unsigned(6,4);
         u <= resize(n-60,4);
      elsif n<80 then
         d <= to_unsigned(7,4);
         u <= resize(n-70,4);
      elsif n<90 then
         d <= to_unsigned(8,4);
         u <= resize(n-80,4);
      else
         d <= to_unsigned(9,4);
         u <= resize(n-90,4);
      end if;

   end process;

end A;
```

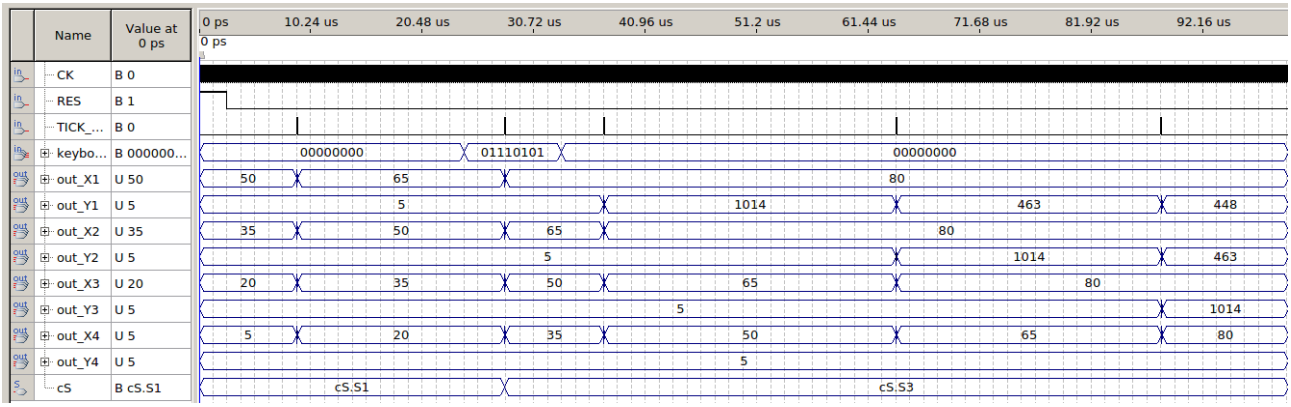
4 Compilation Summary e Simulazioni

Quartus II 32-bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	CII_Starter_Board
Top-level Entity Name	CII_Starter_Board
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	789 / 18,752 (4 %)
Total combinational functions	757 / 18,752 (4 %)
Dedicated logic registers	173 / 18,752 (< 1 %)
Total registers	173
Total pins	35 / 315 (11 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

4.1 Simulazione FSM_per_snake

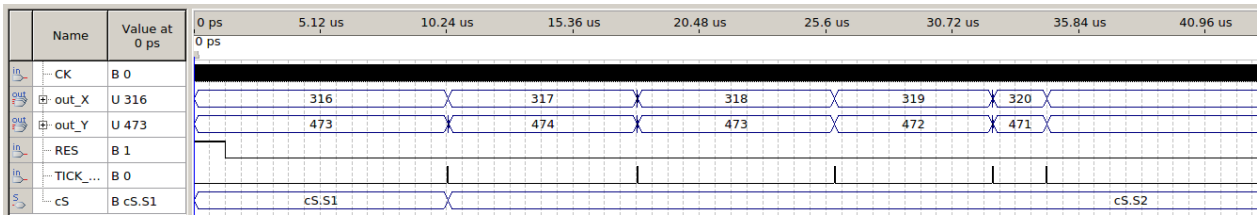


Si è messo nel segnale TICK_REFRESH un segnale che è alto per un periodo di clock simulando in questo modo l'uscita del adjust_button, mentre nel CK un segnale periodico con periodo 20ns (50MHz).
Da questa simulazione è possibile notare come la pressione del pulsante 'freccia giù' della tastiera, e quindi portare sul segnale keyboard di FSM_per_snake il codice "01110010", porti ad un cambio di stato della macchina a stati e le posizioni dei vari quadrati che compongono il serpente cambiano: in S1 salta di +15px la X1 mentre Y1 rimane costante (direzione destra), in S2 salta di +15px Y1 e X1 rimane costante (direzione basso). In entrambi i casi le X_i e Y_i (i=[2,3,4]) seguono i valori del quadrato antecedente ($X_i <= X_{i-1}$) in questo modo a video risulta l'andamento a scia del serpente.



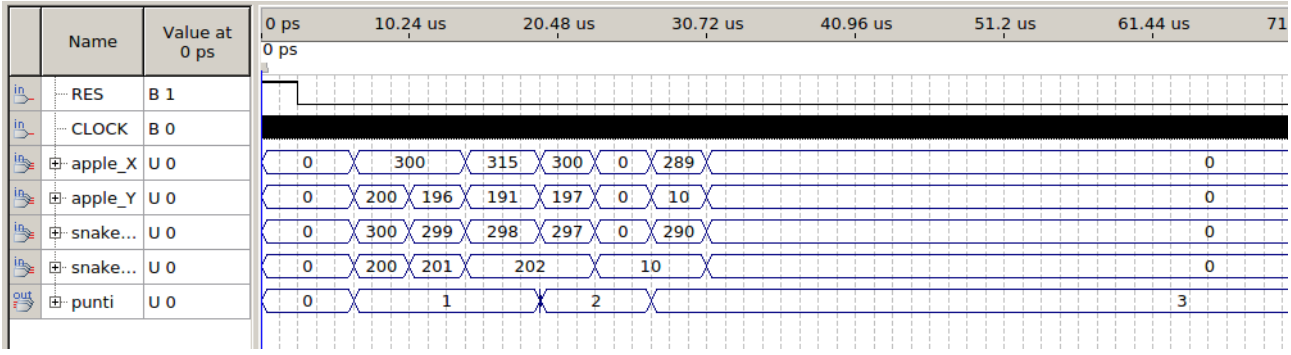
In questa seconda simulazione sempre di FSM_per_snake si è simulata la pressione del tasto 'freccia su', direzione alto che viene implementata con Y1<=Y1-15 e X1<=X1. Essendo il serpente nella posizione (65,5) al momento della pressione, viene decrementata di 15 la Y1 che, dato che è un unsigned da 10 bit, si porta al valore di 1014. Quando il registro riceve il segnale di enable (TICK_REFRESH), Y1 viene portato a 463 (fine schermo -5px per compensare la grandezza del quadrato.

4.2 Simulazione FSM_apple



Ad ogni impulso di TICK_REFRESH la posizione della mela viene aggiornata secondo lo stato in cui si trova. Nel caso specifico: S1 → out_X<=out_X+1 e out_Y<=out_Y+1, S2 → out_X<=out_X+1 e out_Y<=out_Y-1. Nel momento in cui out_Y è 473 e arriva TICK_REFRESH, la macchina a stati cambia stato, out_Y viene incrementato ancora secondo la logica di S1 raggiungendo così l'ultimo px disponibile in senso verticale per rendere visibile il quadrato e al TICK_REFRESH successivo segue la logica di S2.

4.3 Simulazione assegnamento punti



In questa parte di simulazione si è creato un progetto a parte contenente la sola parte dell'assegnamento dei punti che ha come ingresso le posizioni X e Y sia della mela sia della testa del serpente, oltre come sempre al clock ed al segnale di Reset. Dopo aver impostato un clock con periodo 20ns in accordo con quello fornito dalla scheda FPGA (50MHz), si sono assegnati valori a

apple_X, apple_Y, snake_X1 e snake_X2 in modo arbitrario per testare i diversi casi. Si può notare che se le posizioni apple_X con snake_X1 e apple_Y con snake_Y1 sono vicine in un intorno di ±5px viene assegnato un solo punto. Prima che venga assegnato un altro punto i due oggetti devono separarsi fino a non rispettare più la condizione descritta sopra, questo per evitare che nel momento in cui passano vicini vengano assegnati più punti in una volta sola.

5 Segnali di Sincronizzazione

I segnali che sincronizzano i vari blocchi e le varie entity sono:

- TICK_END_DISPLAY: attivo per il periodo di T_{CK_25}=40ns nel quale la rete che sta scorrendo tutta la lunghezza del display, ha completato tutte le righe e tutte le colonne, questo significa che sarà attivo ogni 1/60Hz=16.7ms. Questo segnale abilita il refresh della posizione della mela in questo modo la mela cambierà posizione solo una volta ogni scorrimento di schermata. Inoltre abilita il conteggio di cont che, arrivato a 5, attiva un secondo segnale di sincronizzazione TICK_VEL;
- TICK_VEL: attivo per il lasso di tempo in cui cont è uguale a 5 (starà ad un valore alto quindi fino al successivo TICK_END_DISPLAY) viene dato al blocco adjust_button in modo che ne esca un segnale (PULSE_VEL) attivo per un unico periodo di clock (20ns);
- PULSE_VEL: è l'ENABLE del registro della macchina a stati che controlla i movimenti del serpente. In questo modo il serpente aggiornerà la sua posizione una volta ogni 5 TICK_END_DISPLAY ed avrà una velocità quindi ragionevolmente controllabile da un essere umano. NOTA: se fosse stato messo TICK_VEL invece di PULSE_VEL la posizione veniva sempre aggiornata ogni 5 TICK_END_DISPLAY ma rimanendo l'enable alto per circa 16.7ms/20ns=8,35M di T_{CK} avrebbe aggiornato la posizione 8.35M di volte tra una schermata e la successiva.