0.1 Architettura

Sulla board che abbiamo in dotazione per risparmiare sul numero di segnali necessari per pilotare 8 digit, la Digiland ha provveduto a collegare i catodi in comune a tutte le digit tra di loro e collegare gli anodi di ogni led della digit tutti ad un anodo comune per ognidigit, come riportato in fig 1.

Pertanto per abilitare una digit si bisognerà innanzitutto alimentare opportunamente l'anodo e poi pilotando i catodi opportunamente verrà mostrato sul display il valore. Tale soluzione così fatta presenta però un problema di fondo, alimentando tutte le digit si mostrerà su di esse la stessa cifra. Pertanto si deve provvedere a realizzare una soluzione più ingegnosa, in particolare, sfruttando la persistenza dell immagini sulla retina e la velocità a cui può funzionare la board si riesce a mostrare valori diversi su ogni digit. In particolare ci saranno due componenti che molto velocemente attivano una sola digit alla volta e pilotano i catodi per mostrare il valore che si vuole mostrare su quella digit.

La seguente descrizione si riferisce all'architettura in grado di pilotare soltanto 4 digit, le modifiche necessarie per poterne pilotare 8 sono descritte nell'apposito paragrafo.

L'architettura del componente è mostrata in fig.2. I segnali in ingresso saranno:

- *clk* - segnale di clock per la tempificazione;
- *reset* - segnale di reset, per resettare il valore del display quando è alto (tramite eventuale pressione di un pulsante);
- *values* - segnale di 16 bit per detereminare il valore da visualizzare sul display;
- *dots* - segnale di 4 bit per l'abilitazione dei punti decimali sul display;
- *enable_digit* - segnale di 4 bit per l'abilitazione degli 4 anodi corrispondenti alle 4 cifre sul display (logica 1-attivo);

I segnali in uscita saranno invece:

- *anodes* - bus per l'abilitazione delle 4 digit della batteria di display (0-attivo);
- *cathodes* - bus per pilotare i dei segmenti di ogni cifra (0-attivo);

Il componente è stato realizzato tramite la composizione dei seguenti componenti:

- *clock_divisor* - divisore di frequenza per il clock, necessario perchè se utilizzaimo una frequenza elevata vedremmo lo stesso valore per ogni digit;
- *counter_mod2n* - contatore modulo 2^n, con n=2, per la selezione della cifra da attivare in base al valore di conteggio;
- *anodes_manager* - componente per la selezione degli anodi delle cifre da attivare, formato da un decoder 1-4;
- *cathod_manager* - componente per la selezione dei segmenti da attivare per ogni cifra, formato da un multiplexer 4-1 e un nibble selector/cathod coder.

L'implementazione completa è consultabile qui: display_7_segmenti.vhd.

0.2 Clock divisor e contatore modulo 2^n

0.2.1 Contatore modulo 2^n

Il contatore è un componente che conta il numero di impulsi applicati in ingresso (sul fronte di salita del clock). Oltre al *clock*, in ingresso c'è un segnale di abilitazione (1-attivo) e un segnale di *reset_n* per resettare il conteggio. In uscita, il segnale *counter* riporta il valore di conteggio corrente, mentre *counter_hit* diventa 1 solamente se il valore del conteggio è costituito da tutti 1 (valore massimo). L'implementazione, effettuata tramite descrizione behavioural, è consultabile qui: counter_mod2n.vhd.

0.2.2 Clock divisor

Prima di utilizzarlo, il segnale di clock in ingresso al componente viene filtrato tramite un clock divisor, che si occupa di filtrare i fronti del clock ad una frequenza *clock_frequency_in* per averli ad una frequenza più bassa *clock_frequency_out*. Il funzionamento di tale componente è del tutto analogo a quello di un contatore modulo 2^n, dove *clock_frequency_out* non è altro che il *counter_hit*, ossia un valore che diventa alto solamente quando il contatore ha raggiunto il suo valore massimo (calcolabile come *clock_frequency_in*/*clock_frequency_out* - 1). Ovviamente sappiamo che il clock generato con questo metodo non è esente da problemi di sincronizzazione e di scue, ma per i nostri scopi e per il fatto che avevamo bisogno di frequenze minori di 5 Mhz, tale scelta risulta più adatta .

L'implementazione, effettuata tramite descrizione behavioural, è consultabile qui: clock_divisor.vhd.

0.3 Anodes manager

L'obiettivo di tale componente è la gestione degli anodi relativi alle cifre del display. Dal momento che i singoli anodi relativi a ciascuna delle 4 cifre del display sono 0-attivi (la Nexys4 DDR utilizza i transistor per pilotare abbastanza corrente nel punto di anodo comune, le abilitazioni dell'anodo sono invertite), l'anodes manager dovrà attivare uno solo dei 4 diverso anodi mantenendo basso uno solo dei 4 bit relativi agli anodi, utilizzando in ingresso il valore fornito dal contatore. Il componente dovrà inoltre tenere conto del segnale enable in ingresso, che permette di attivare e disattivare manualmente i singoli anodi.

Per realizzare l'anodes manager si è utilizzato un decoder 2-4: ricevuto in ingresso il valore del contatore, il decoder alza solo uno dei 4 bit relativi agli anodi. Tali uscite sono poi messe in AND con il segnale di enable per pilotare soltanto le digit che si è degiso di abilitare. Infine, i bit vengono invertiti per rispettare la logica 0-attiva. L'implementazione completa, effettuata tramite descrizione data-flow, è consultabile qui: anodes_manager.vhd.

E' interessante notare come, nell'implementazione del decoder, oltre alle 4 possibili combinazioni di ingressi è stato aggiunto il caso "others", che genera in uscita tutti bit alti (che verranno poi negati successivamente). Tale tecnica serve ad evitare il fault masking, poiché avendo tutti 1 in uscita (caso non previsto dal normale funzionamento di un decoder) posso riconoscere subito la presenza di comportamenti imprevisti nel componente, che verrano poi manifestati all'esterno mostrando su tutte le digit lo stesso valore anche quando i valori dovrebbero essere diversi, perchè

tutti gli anodi sono abilitati nello stesso istante. Una soluzione duale consisterebbe nello spegnere tutte le digit, basta sostituire x"F" con x"0".

```
architecture dataflow of anodes_manager is
  signal anodes_swhitching : STD_LOGIC_VECTOR (3 downto 0) := (others =>
     '0');
  begin
    anodes <= not anodes_swhitching OR not enable_digit;
    with select_digit select anodes_swhitching <=
      x"1"    when "00",
      x"2"    when "01",
      x"4"    when "10",
      x"8"    when "11",
      x"F"    when others;
end dataflow;
```

Codice Componente 1: Implementazione data-flow dell'anodes manager.

0.4 Cathodes manager

Il cathodes manager permette di gestire i catodi associati ad ogni segmento omologo di ogni cifra del display a 7 segmenti. Per accendere il giusto segmento è necessario che il catodo corrispondente sia posto a 0, poichè i catodi sono pilotati da segnali 0-attivi. Il componente prende in ingresso:

- il bus *counter*, uscita del contatore (come l'anodes manager), che serve per segliere quale nibble mostrare sulla digit;
- il bus *values* (16 bit) per determinare il valore di ogni cifra e dunque i segmenti da accendere;
- il bus *dots* (4 bit) per determinare quali dei 4 punti decimali accendere.

In uscita abbiamo un bus ad 8 bit *cathodes* che indica la configurazione dei catodi relativi alla cifra attiva in quel momento e all'eventuale punto da accendere.

Per realizzare tale componente, si è utilizzata un'implementazione di tipo behavioural. In particolare, abbiamo due *process*:

- *digit_switching* - in base al valore di *select_digit* (contatore), si occupa di settare i bit del bus interno *nibble*[1] corrispondenti alla rappresentazione del valore che si vuole mostrare su quella digit ;
- *decoder* - in base alla *digit* presente nel *nibble*, imposta *cathodes_for_digit* al valore necessario per accendere i segmenti nel modo corretto per rappresentare il valore richiesto. Tali valori sono espressi come costanti e ricavabili dal reference manual.

[1] Un nibble è una stringa di 4 bit.

Infine, per determinare l'accensione dei dots, si utilizza un *multiplexer 4-1* generico. Il valore di cathodes è dunque determinato come segue: si calcola dapprima la parte dei dots, selezionando solo quelli relativi alle cifre selezionate e poi negandoli (per logica 0-attiva), e infine concatena aggiunge *cathodes_for_digit*.

```
    digit_switching: process (select_digit,values)
    begin
      case select_digit is
        when "00" => nibble <= digit_0;
        when "01" => nibble <= digit_1;
        when "10" => nibble <= digit_2;
        when "11" => nibble <= digit_3;
        when others => nibble <= (others => '0');
      end case;
    end process;
    decoder : process (nibble)
    begin
      case nibble is
        when "0000" => cathodes_for_digit <= zero;
        when "0001" => cathodes_for_digit <= one;
        when "0010" => cathodes_for_digit <= two;
        [...]
           end case;
  end process;

cathodes <= not dots(to_integer(unsigned(select_digit))) &
   cathodes_for_digit;
```

Codice Componente 2: Determinazione del valore di cathods..

L'implementazione completa è consultabile qui: cathodes_manager.vhd.

0.5 Display su Nexys 4

La board Nexys 4 DDR ha installato a bordo due batterie di 4 digit ciascuna, per un totale di 8 digit. La soluzione che abbiamo visto precedentemente però permette di controllare una sola batteria di 4 digit, e non potendo instanziare due componenti che controllano ciascuno 4 digit, è necessario apportare alcune modifiche agli elementi che costituiscono il componente che pilota i display a 7 segmenti.

In particolare, di seguito, vediamo alcune modifiche che abbiamo apportato ai componenti per poter pilotare tutte le digit che mette a disposizione la Nexys 4 DDR.

```
entity display_7_segments is PORT ( enable      : in STD_LOGIC;
                  clock       : in STD_LOGIC;
                  reset      : in STD_LOGIC;
                  values       : in STD_LOGIC_VECTOR  (31 downto 0);
                  dots       : in STD_LOGIC_VECTOR  (7 downto 0) ;
```

```
                    enable_digit    : in STD_LOGIC_VECTOR  (7 downto 0);
                    anodes          : out STD_LOGIC_VECTOR (7 downto 0);
                    cathodes        : out STD_LOGIC_VECTOR (7 downto 0)
                  );
end display_7_segments;

...

component counter_UpMod2n_Re_Sr is
        GENERIC (   n    : NATURAL :=3     );
          PORT (  enable       : in STD_LOGIC ;
            reset_n     : in STD_LOGIC;
            clock       : in STD_LOGIC;
            count_hit   : out STD_LOGIC;
            COUNTS      : out STD_LOGIC_VECTOR ((n-1) downto 0)         );
end component;
```

Codice Componente 3: display$_{7s}egments$

Nella top level entity del componente che permette di mostrare i valori sui display, il numero di segnali che indicano il la cifra da mostrare passano da 16 a 32, in quanto per ogni digit servono 4 bit per poter codificare i valori, in esadecimale, che possiamo mostrare su di essi. Poichè il numero di display da pilotare passa da 4 a 8, anche il numero di segnali che pilotano i punti , le digit e gli anodi aumentano. Inoltre il contatore non è più un contatore modulo 4 ma è un contatore modulo 8, in quanto deveno essere abilitate 8 digit.

Tali modifiche riguradano anche l'anodes_manager che deve pilotare 8 digit e non più 4, pertanto il decoder da un decoder 2:4 diventa un decoder 3:8 come si nota di seguito.

```
   with select_digit select anodes_switching <=
    x"01"     when "000",
    x"02"     when "001",
    x"04"     when "010",
    x"08"     when "011",
    x"10"  when "100",
    x"20"  when "101",
    x"40"  when "110",
    x"80"  when "111",
    (others => '0')    when others;
  end case;
```

Codice Componente 4: Abilitazione degli anodi

Per il cathodes_manager cambia il numero di nibble che dobbiamo gestire, infatti anche essi passano da 4 a 8, ognuna di esse è costituita da 4 bit di values, stringa del valore da mostrare sul display codificato in codifica binaria classica. In particolare partendo dal LSB ogni 4 bit di values codifica il valore da mostrare su una delle digit. Anche il process che si accupa di assegnare ai catodi la corretta codifica del valore viene estesa per poter gestire 8 digit e non più 4.

```
    alias digit_0 is values (3 downto 0);
  alias digit_1 is values (7 downto 4);
```

```
  alias digit_2 is values (11 downto 8);
  alias digit_3 is values (15 downto 12);
  alias digit_4 is values (19 downto 16);
  alias digit_5 is values (23 downto 20);
  alias digit_6 is values (27 downto 24);
  alias digit_7 is values (31 downto 28);

...

    digit_switching: process (select_digit,values)
    begin
      case select_digit is
        when "000" => nibble <= digit_0;
        when "001" => nibble <= digit_1;
        when "010" => nibble <= digit_2;
        when "011" => nibble <= digit_3;
        when "100" => nibble <= digit_4;
        when "101" => nibble <= digit_5;
        when "110" => nibble <= digit_6;
        when "111" => nibble <= digit_7;
        when others => nibble <= (others => '0');
      end case;
    end process;
```

Codice Componente 5: cathodes$_m$*anager.vhd*..

0.6 Approfondimento: Double display on-board

Osserviamo il funzionamento del display a 7 segmenti su board. Per poter testare il componente descritto sulla board Nexys 4 utilizzando tutte e 8 le digit, si è realizzata un'entità di più alto di livello denominata *DoubleDisplayOnBoard* che utilizza il componente descritto nel paragrafo precedente. L'idea che abbiamo seguito consiste nel testare il funzionamento del display mostrando su di esso il valore, in esadecimale, dato in input facendo commutare i 16 switch presenti sulla board. Il primo problema che abbiamo incontrato consiste nel fatto che per poter mostrare il valore su tutte le 8 digit, abbiamo bisogno di un bus con parallelismo di 32 bit, quindi con gli switch possiamo pilotare a massimo 4 digit. Per risolvere tale problema abbiamo deciso di seguire la seguente idea. Suddividere il bus values in ingresso al componente display_7_segments e collegare le due metà a dei registri di tipo d abilitati con parallelismo di 16 bit, in ingresso a tali registri abbiamo collegato i 16 switch e collegando gli enable di tali registri con due pulsanti sulla board, possiamo caricare la rappresentazione in binario del valore che vogliamo mostrare nei registri in modo tale che tale metà verrà sostenuta in ingresso al componente fino a quando non si provvederà ad un nuovo aggiornamento rieseguendo la procedura di load, premendo il pulsante.

```
[...]
architecture structural of display_onBoard is
  entity DoubleDisplayOnBoard is
```

```
      Port (  clock         : in  STD_LOGIC;
          values      : in  STD_LOGIC_VECTOR (15 downto 0);
          load_reg_0_3    : in  STD_LOGIC;
          load_reg_4_7    : in  STD_LOGIC;
          anodes      : out  STD_LOGIC_VECTOR (7 downto 0);
          cathodes      : out  STD_LOGIC_VECTOR (7 downto 0)
      );
  end DoubleDisplayOnBoard;
[...]

alias values_0_3 is values_int (15 downto 0) ;
alias values_4_7 is values_int (31 downto 16) ;

begin
  register_0_3: register_d_Re_Ar PORT MAP (   clock   => clock,
                          load   => load_reg_0_3,
                          reset   => reset,
                          d    => values,
                          q     => values_0_3
                );
  register_4_7: register_d_Re_Ar PORT MAP (   clock    => clock,
                          load  => load_reg_4_7,
                          reset    => reset,
                          d   => values,
                          q      => values_4_7
                );

[...]
```

Codice Componente 6: DoubleDisplayOnBoard.vhd

L'implementazione completa è consultabile qui: display_onBoard.vhd.

Inoltre, è stato opportunamente configurato il file Nexys4DDR_master.ucf per effettuare il mapping tra i physical della board e le interfaccia di ingresso/uscita del componente che abbiamo sintetizzato, un punto interessante è il collegamento dei segnali di load dei registri ai bottoni presenti sulla scheda. Ciò ci permette di poter scegliere in quale registro e di conseguenza su quale batteria di display mostrare il valore che stiamo dando in input con gli switch.

```
[...]
#bottone destro
  NET "load_reg_0_3"          LOC=P17 | IOSTANDARD=LVCMOS33; #
      IO_L12P_T1_MRCC_14
#bottone sinistro
  NET "load_reg_4_7"          LOC=M17 | IOSTANDARD=LVCMOS33; #
      IO_L10N_T1_D15_14
[..]
```

Codice Componente 7: DoubleDisplayOnBoard.vhd

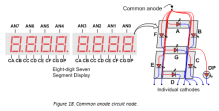

Figura 1: Display 7 segmente sulla Nexys 4

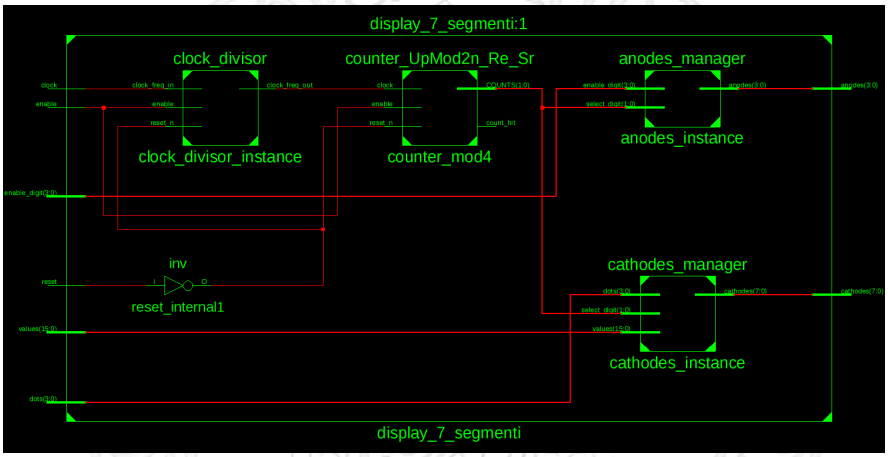

Figura 2: Schematico del latch RS abilitato.

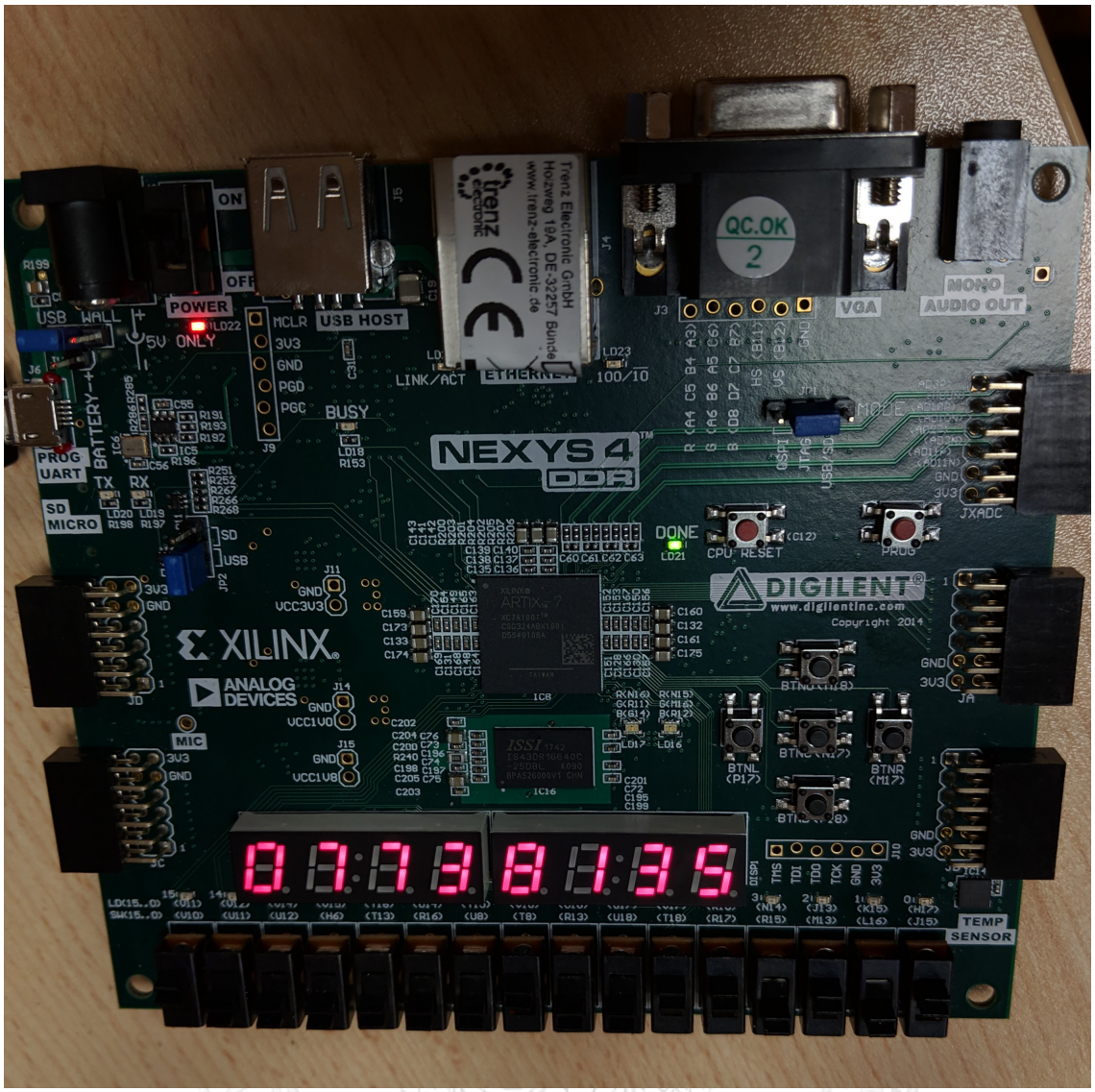

Figura 3: Display a 7 segmenti su board Nexys 4.