VGA Controller

Fabio Callegari, Leonardo Vannoli

Settembre 2020

Sommario

In questo lavoro ci si è occupati di sviluppare un **VGA Controller** per lo standard VGA 60Hz 640x480. Utilizzando il linguaggio di descrizione hardware **Verilog** è stato scritto il codice del VGA Controller. Lo scopo del codice è quello di generare su schermo 8 barre colorate verticali. Per le operazioni di testing del codice sono stati usati l simulatori Verilog **Icarus Verilog** e **ModelSim**, e **GTKWave** come tool di analisi per il debbugging del codice Verilog. Infine, con l'aiuto di un VGA Simulator online, è stata stampata a schermo l'immagine generata dal VGA Controller.

1 VGA Controller

Lo standard VGA (Video Graphic Array) è uno standard analogico relativo a display per computer introdotto sul mercato nel 1987 da IBM [1]. Per guidare il display, lo standard VGA definisce due segnali di temporizzazione digitali, **Horizontal Sync Signal** e **Vertical Sync Signal**, e tre segnali analogici R,G,B.
Questi segnali sono schematizzabili come mostrato in Fig 1

Dalla Fig (1) possiamo notare che il segnale Horizontal Sync (Hsynq) passa da 0 a 1 dopo un tempo chiamato di **Sync Pulse**. Dopo aver atteso un intervallo di tempo suddiviso in **Back Porch**, **Active Video** e **Front Porch**, il segnale torna a 0 per il Sync Pulse successivo. L'Active Video è l'area di segnale sincronizzata temporalmente con la trasmissione dei canali R,G,B da visualizzare. Quindi l'intervallo di **Horizontal Blanking** deve terminare con la fine della porzione di segnale alto dedicata al Back Porch e ricominciare con l'innesco del Front Porch.
L'andamento temporale del Vertical Sync (Vsynq) segue la stessa logica dell'Hsynq ma su una scala di tempi più lunga in quanto deve contenere un'intera schermata.
Tutto ciò avviene ogni $\frac{1}{Hz\ Standard}$ ossia il tempo in cui lo display fa un refresh della schermata per mostrare i possibili nuovi contenuti.

1.1 Lo standard VGA 640x480 60Hz

Lo standard preso in esame per il nostro lavoro è lo stardard VGA 60Hz 640x480.
Questo standard possiede un Pixel Clock di $25MHz$ e quindi un tempo per singolo pixel che vale $\frac{1}{25MHz}$. Tenendo presente questo facile fattore di conversione, è possibile tabellare i

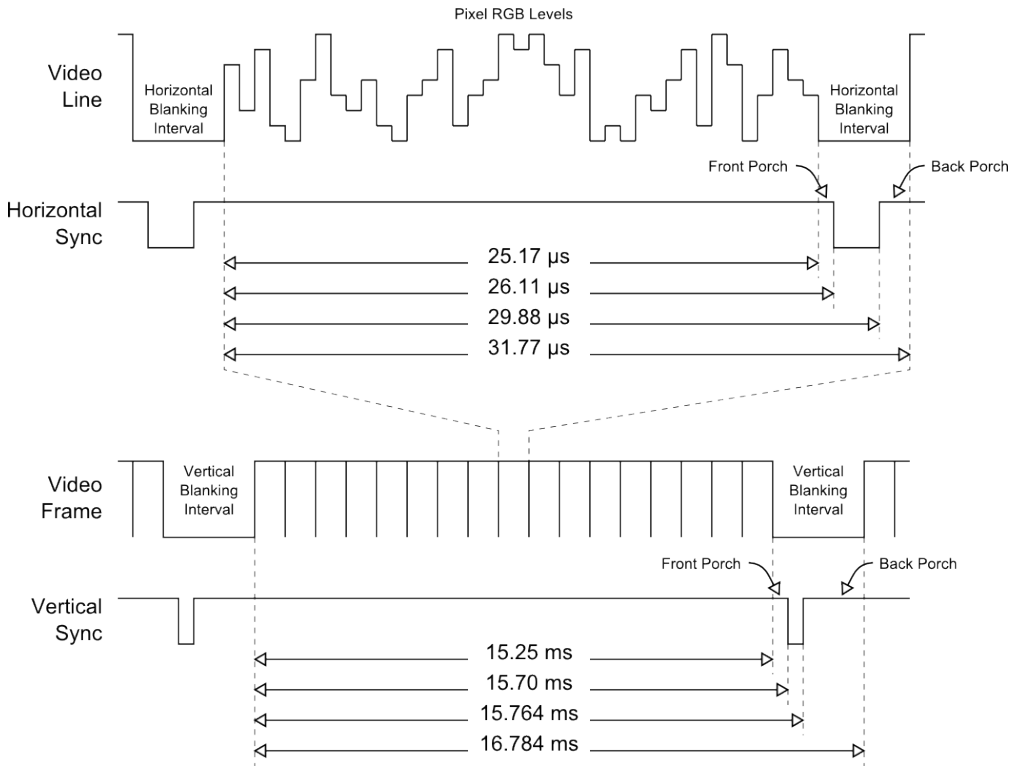


Figura 1: Video Line with RGB Levels Horizontal Sync signal, Video Frame and Vertical Sync signal

tempi di durata dei segnali dello standard in esame in termini di pixel (per l'Horizontal) e linee di pixel (per il Vertical). Questi valori sono riportati nella Tab 1 [3].

Tabella 1: Tempi dei segnali per lo standard VGA 640x480 60Hz in pixel units

Format	**Pixel Clock (MHz)**	**Horizontal** (pixel)				**Vertical** (lines)			
		Active Video	**Front Porch**	**Sync Pulse**	**Back Porch**	**Active Video**	**Front Porch**	**Sync Pulse**	**Back Porch**
640x480 60Hz	25	640	16	96	48	480	11	2	31

Dai valori riportati in Tab 1 osserviamo che, in pixel, lo standard VGA 640x480 60Hz si compone di 800x524 pixel in totale. Per sviluppare un firmware che permetta di mostrare a schermo un'immagine bisogno quindi tenere in considerazione i pixel in "eccesso" non corrispondono alla parte di **Active Video**. Il segnale analogico RGB va quindi acceso solo in presenza della porzione di pixel corrispondenti all'Active Video e non fuori da tale area.

2 La logica del firmware

Una volta studiato il problema, si è passati alla scrittura del firmware. Il firmware è stato suddiviso in moduli così da facilitarne il debbugging.

Per prima cosa si sono studiate le specifiche della scheda FPGA su cui sarebbe stato flashato il firmware. La scheda in questione è la scheda FPGA Atlera DE2-115

2.1 Caratteristiche scheda FPGA

La scheda FPGA per cui è stato sviluppato il codice è la Cyclone® IV EP4CE115. La FPGA è montata su una board Altera DE2-115 Terasic. La FPGA è pensata per un uso educativo, motivo per il quale possiede una abbondanza di interfacce che possano accomodare le più svariate applicazioni [2].
Importante, ai fini del nostro lavoro, la presenza di una connettore VGA Out.
Inoltre è importante tenere in considerazione che la board possiede un clock a 50MHz. Essendo il clock dello standard VGA a 25MHz, una parte del lavoro del codice risiede nella divisione del clock in entrata.

2.2 Divisione del Clock

Per compiere la divisione del clock da 50MHz presente sulla board e portarlo a 25MHz, è stato scritto il modulo `clock50_divider` (1).
Questo modulo prende in input il `clk` con variabile wire. Ogni volta che il clock (a 50 MHz) è sul fronte positivo (`posedge`), la variabile di output `divided_clk` viene invertita. Questo avviene ogni $20\,ns$, quindi il periodo del nuovo clock durerà $40\,ns$, ossia 25MHz.

2.3 Horizontal Counter

Il modulo `horizontal_counter` (2) si occupa di tenere conto del pixel in cui ci si trova ad ogni clock. Prende in input il clock a 25MHz e restituisce in output il numero del pixel in cui mi trovo attraverso la variabile `H_Count_Value`. Il valore di questa variabile viene aggiornato ad ogni clock fino al raggiungimento del ultimo pixel sulla riga orizzontale. Quando questa variabile raggiunge il valore 799, viene riazzerata in modo che il contatore per la riga successiva ricominci da zero. Contemporaneamente viene modificato da 0 ad 1 il valore della variabile `enable_V_Counter` che verrà trasmessa in output ad un modulo successivo (il modulo `vertical_Counter`) per innescare il contatore associato all'altra coordinata video.

Arrivato al valore `799` la variabile `enable_V_Counter` viene portata da 0 ad 1 così che al clock successivo la variabile `H_Count_Value` ritorni a 0 e il loop di aggiornamento ricominci. Il modulo restituisce anche la variabile `enable_V_Counter` così da poterla passare al modulo `vertical_counter`.

2.4 Vertical Counter

Dopo aver scorso su tutti i pixel presenti su ogni riga attraverso `horizontal_counter`, il firmware deve passare alla riga successiva per poi ripetere questa operazione fino a quando non si arriva al numero totale di righe. Per fare questo, il modulo `vertical_counter` (3) sempre prendere in input il clock a 25MHz, insieme alla variabile `enable_V_Counter`. Quando `enable_V_Counter` è pari ad 1, la variabile `V_Count_Value` inizialmente instanziata

a zero, viene incrementata, operando così il passaggio da una riga all'altra. Quest'ultima variabile viene mandata in output per il modulo `top`.

2.5 Top module

All'apice di tutta la catena è stato creato il modulo `top` (4). Questo modulo si occupa di coordinare le chiamate a tutti gli altri moduli sopracitati.
Questo modulo riceve in input il clock a 50MHz. Porta in output tutte le variabili che serviranno al controller per il suo funzionamento:

- `Hsynq`, variabile che memorizza il valore in binario del segnale di sync orizzontale
- `Vsynq`, variabile che memorizza il valore in binario del segnale di sync verticale
- `Red, Gree, Blue`: queste tre variabili binarie a tre bit (per sfruttare la profondità di colore) si occupano di memorizzare l'informazione del colore della VGA.

Dopo aver chiamato i tre moduli all'interno del modulo top, si ottengono in output i valori corrispondenti alla riga e alla colonna in cui ci si trova ad un determinato istante.
La variabile `Hsynq` viene quindi portata ad 1 quando il valore di `H_Count_Value>96`, ossia dopo che l'intervallo di Horizontal Synq si è concluso.
Allo stesso modo, la variabile `Vsynq` viene quindi portata ad 1 quando il valore di `V_Count_Value>2`, ossia dopo che l'intervallo di Vertical Synq si è concluso.
Infine, attraverso l'utilizzo di operatori booleani `&&` e `||`, vengono poratate ad 1 le tre variabili di colore in modo da ottenere 8 bande colorate: Black, Red, Magenta, Blue, Cyan, Green, Yellow e White. Da tenere in considerazione il fatto che la zona di Active Video sarà presente da `H_Count_Value>143` (Hsynq + Back Porch) e fino a `H_Count_Value<783` (Front Porch), mentre contemporaneamente `31<V_Count_Value<513` (per i medesimi motivi di `H_Count_Value`).

2.6 Testbench module

Per testare il nostro firmware, è stato creato il modulo `testbench` (5). All'interno del testbench, dopo aver creato un clock a 50MHz (`always #10 clk = ~clk;`), viene chiamato il modulo `top` il quale ritorna le due variabili di sync e le variabili di colore. Viene quindi istanziato un delay di 16.8 ms dopo il quale si conclude la simulazione.
Sono state aggiunte due ulteriori funzioni al modulo `testbench`: la prima stampa su file i risultati della simulazione così da poter dare il file output.txt prodotto in lettura ad un simulatore di VGA online; La seconda crea un file results.vcd leggibile con il programma GTKWave [4] di cui parleremo in seguito. Quest'ultimo file è servito, in fase di scrittura del firmware, per operazioni di debbugging.

2.7 Simulatori Icarus e ModelSim

Icarus è un simulatore Verilog che opera come compilatore di codici sorgente scritti in Verilog [5]. Permette, una volta eseguita la compilazione, di generare un file intermedio chiamato

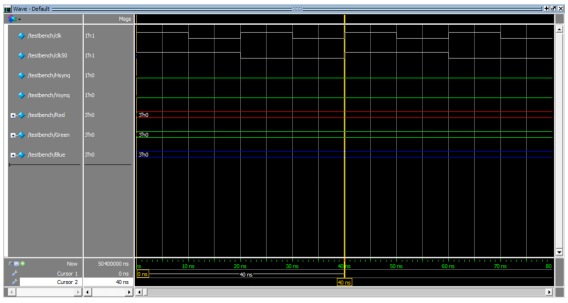
vpp assembly, eseguibile tramite comando "`vpp`".
Una volta scritto il nostro firmware, questo è stato compilato usando Icarus e quindi la simulazione lanciata tramite il comando `vpp`. Alla fine, la simulazione ci ha restituito i file output.txt e results.vcd. ModelSim è un ambiente di simulazione multilinguaggio, per la simulazione di linguaggi di descrizione hadware quali VHDL e Verilog (tra gli altri). La simulazione viene eseguita utilizzando una Graphic User Interface (GUI) [7].

3 Risultati

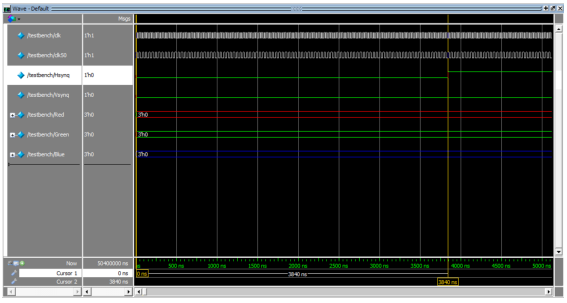
Di seguito sono mostrati i risultati ottenuti dalla simulazione del VGA Controller. I grafici sono stati ottenuti utilizzando GTKWave. Questo tool è un analizzatore e visualizzatore di forme d'onda. Fornisce un metodo per visualizzare i risultati della simulazione sia per segnali analogici che segnali digitali e consente varie operazioni di ricerca e manipolazioni temporali, dando inoltre la possibilità di salvare risultati parziali (ovvero "segnali di interesse") estratti da un dump di simulazione completo.

3.1 Studio dei segnali con GTKWave

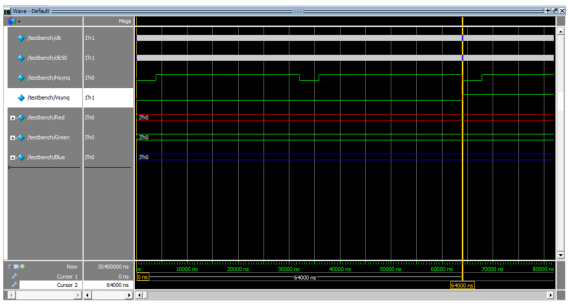
Per prima cosa sono stati studiati i due segnali di clock, quello a 50MHz (clock della board) e quello a 25MHz (compatibile con lo standard VGA scelto) Fig. 2(a). Inoltre sono stati studiati i tempi di arrivo e di durata degli altri segnali. In Fig. 2(b) viene mostrato il tempo di attivazione dell' HSync, dopo aver atteso 3840 ns, ossia dopo i 96 pixel di durata del SyncPulse, mentre in Fig. 2(c) quello di arrivo del primo segnale di Vsynq, dopo aver atteso 64000 ns, ossia dopo 2 linee orizzontali complete. In Fig. 2(d) viene invece mostrato il tempo del primo fronte di salita del segnale Red. Questo segnale arriva non all'inizio della display area, ma dopo 80 pixel dall'inizio di quest'ultima, poiché è stato scelto di mostrare a display per prima una colonna nera, dove tutti e tre i segnali di colore sono assenti.
Vengono infine mostrati i tempi di durata di un segnale Hsynq, comprensivo di Back Porch, Active Video e Front Porch (Fig. 2(e)) ed i tempi di durata del Full Frame, ossia di una schermata completa (Fig. 2(e)).



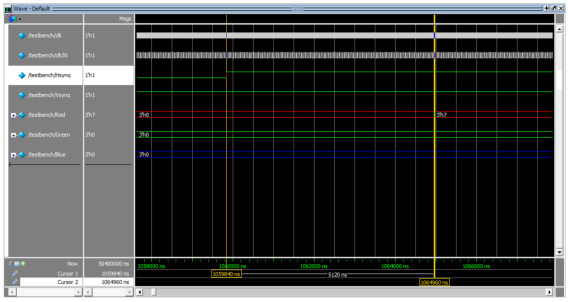
(a) Segnali di clock a 50MHz (sopra) e 25MHz (sotto)



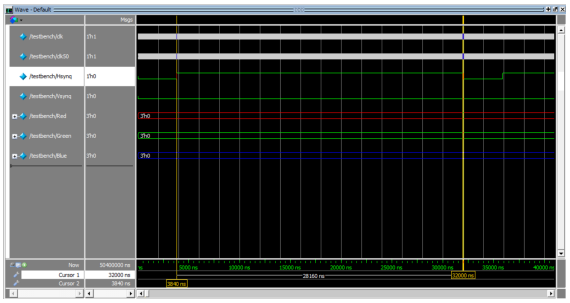
(b) Fronte di salita del segnale HSynq



(c) Primo fronte di salita del segnale Vsynq



(d) Arrivo del segnale Red nella zona di Active Video



(e) Tra le due barre gialle viene mostrata la durata di un segnale HSync

(f) La barra gialla a destra Segnali presenti in un intero frame. In basso sono evidenziati i tempi di durata

Figura 2: Analisi dei segnali

3.2 Display del segnale VGA

Infine, attraverso l'uso di un simulatore VGA online [6], è stato possibile mostrare a schermo il risultato del VGA Controller. In Fig (3) viene mostrato l'output del VGA Controller, ossia 8 bande verticali colorate. La cornice grigia è stata inserita in fase di post-produzione per permettere la visione anche dell'ultima colonna bianca. Le colonne si distanziano tra loro di 80 pixel.

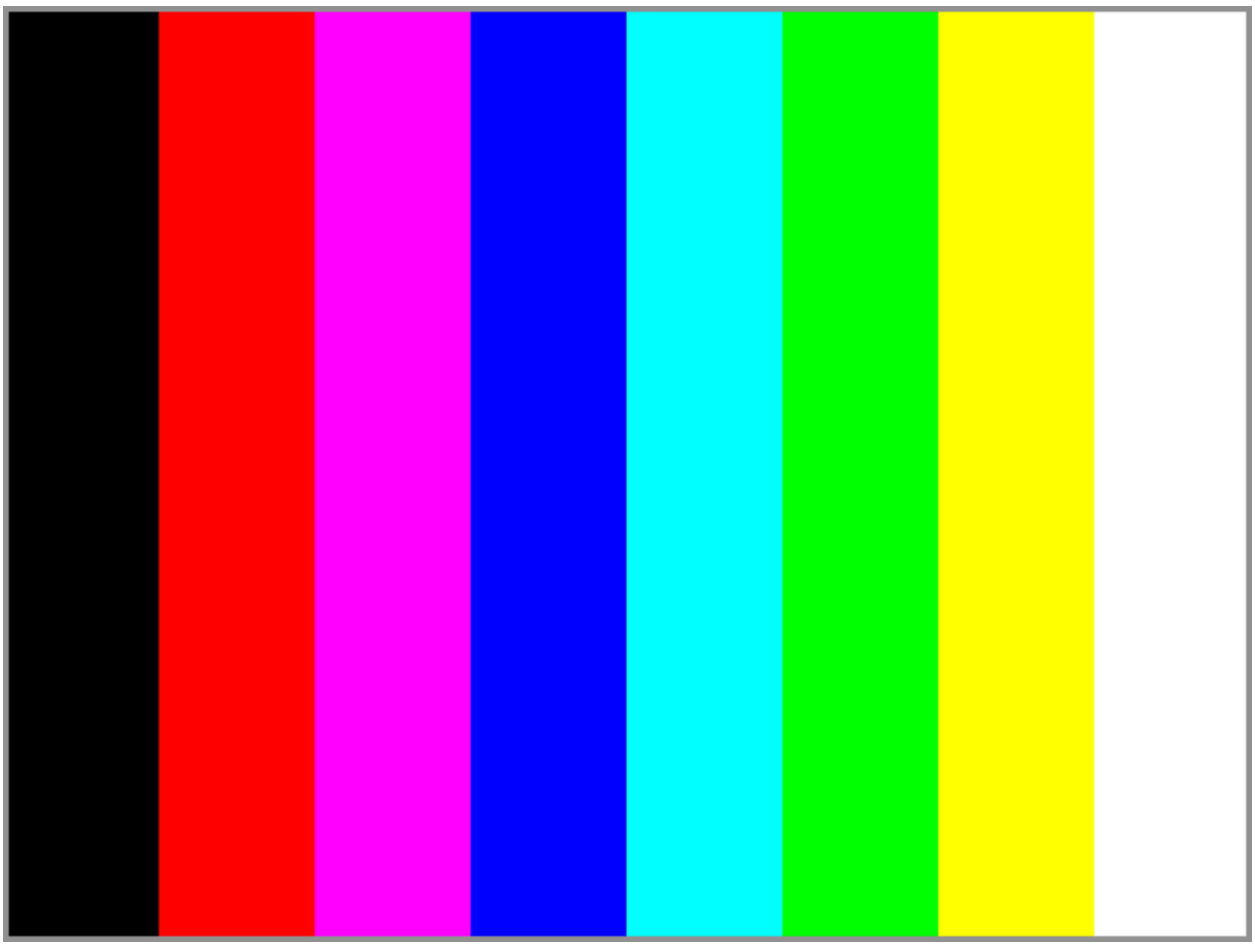


Figura 3: Ouput del VGA Controller. La cornice grigia è stata aggiunta solo delimitare l'immagine così da rendere visibile anche la colonna bianca.

Appendice

A Verilog Code

Code Listing 1: `module clock50_divider`

```
module clock50_divider (
  input wire clk, //50MHz
  output reg divided_clk = 0 //25 MHz
);
  always@ (posedge clk)
      divided_clk <= ~divided_clk; //flip the signal
endmodule
```

Code Listing 2: `module horizontal_counter`

```
module horizontal_counter(
  input clk_25MHz,
  output reg enable_V_Counter = 0,
  output reg [15:0] H_Count_Value = 0
);
  always@(posedge clk_25MHz) begin
    if (H_Count_Value <800) begin
      H_Count_Value<= H_Count_Value +1;
      enable_V_Counter <= 0; //disable Horizontal Counter
      if (H_Count_Value == 799)
        enable_V_Counter <= 1; // trigger V Counter
    end
    else begin
      H_Count_Value <= 1; //reset Horizontal Counter
      enable_V_Counter <= 0; // trigger V Counter
    end
  end

endmodule
```

Code Listing 3: `module vertical_counter`

```
module vertical_counter(
  input clk_25MHz,
  input enable_V_Counter,
  output reg [15:0] V_Count_Value = 1
);

 always@(posedge clk_25MHz) begin
   if(enable_V_Counter == 1'b1) begin
     if (V_Count_Value < 524)
       V_Count_Value<= V_Count_Value +1;
     else V_Count_Value <= 1; //reset Vertical Counter
```

```
    end
  end

endmodule
```

Code Listing 4: `module top`

```
//`include "clock_divider.v"
`include "clock50_divider.v"
`include "horizontal_counter.v"
`include "vertical_counter.v"

module top(
  input clk,
  output Hsynq,
  output Vsynq,
  output [2:0] Red,
  output [2:0] Green,
  output [2:0] Blue,
  output clk_25M
);
  wire clk_25M;
  wire enable_V_Counter;
  wire [15:0] H_Count_Value;
  wire [15:0] V_Count_Value;
  //clock_divider VGA_Clock_gen (clk, clk_25M); //100MHz
  clock50_divider VGA_Clock_gen (clk, clk_25M);  //50MHz
  horizontal_counter VGA_Horiz(clk_25M, enable_V_Counter, H_Count_Value);
  vertical_counter VGA_Verti(clk_25M, enable_V_Counter, V_Count_Value);

  //outputs
  assign Hsynq = (H_Count_Value > 96)?1'b1:1'b0;
  assign Vsynq = (V_Count_Value > 2)?1'b1:1'b0;

  //colors
  //column 1 is Black
  //column 2 is Red
  //column 3 is Red+Blue (Magenta)
  //column 4 is Blue
  //column 5 is Blue+Green (Cyan)
  //column 6 is Green
  //column 7 is Green+Red (Yellow)
  //column 8 is Green+Red+Blue (White)

  assign Red = (((H_Count_Value <= 384 && H_Count_Value > 224) ||
  (H_Count_Value <= 784 && H_Count_Value > 624)) &&
  V_Count_Value < 515 && V_Count_Value > 33) ? 3'b111:3'b000;

```

```
  assign Blue = (((H_Count_Value <= 544 && H_Count_Value > 304) ||
  (H_Count_Value <= 784 && H_Count_Value > 704)) &&
  V_Count_Value < 515 && V_Count_Value > 33) ? 3'b111:3'b000;

  assign Green = (H_Count_Value <= 784 && H_Count_Value > 464 &&
  V_Count_Value < 515 && V_Count_Value > 33) ? 3'b111:3'b000;
endmodule
```

Code Listing 5: module testbench

```
//testbench module
`timescale 1ns/ 1ns

`include "top.v"

module testbench;
  reg clk = 1;
  reg clk50;
  wire Hsynq;
  wire Vsynq;
  wire [2:0] Red;
  wire [2:0] Green;
  wire [2:0] Blue;
  integer f;

  top UUT(clk,Hsynq,Vsynq,Red,Green,Blue,clk50);
  //always #5 clk = ~clk; //100MHz clock
  always #10 clk = ~clk; //50MHz clock

  initial
    begin
      f = $fopen("output.txt","w");
      #50400000;
      $fclose(f);
      $finish();
    end

  always @(posedge clk)
        begin
      $fwrite(f,"%0t␣ns:␣%b␣%b␣%b␣%b␣%b\n",$time,Hsynq,Vsynq,Red,Green,Blue);
        end

  initial
    begin
      $dumpfile("results.vcd");
      $dumpvars(0);
    end
endmodule
```

Riferimenti bibliografici

[1] **Standard VGA:** https://it.wikipedia.org/wiki/Video_Graphics_Array

[2] **FPGA Altera DE2-115:** https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=139&No=502&PartNo=1

[3] **VGA 640x480 Hz:** http://martin.hinner.info/vga/timing.html

[4] **GTKWave software:** http://gtkwave.sourceforge.net/

[5] **Icarus Verilog:** http://iverilog.icarus.com/

[6] **VGA simulator:** https://ericeastwood.com/lab/vga-simulator/

[7] **ModelSim:** https://www.intel.it/content/www/it/it/software/programmable/quartus-prime/model-sim.html