UNIVERSITÀ DI PISA

PROGETTAZIONE CIRCUITI DIGITALI

AA 2011/2012

Gestione di una libreria di forme d'onda a dente di sega su Altera-DE2 board e visualizzazione VGA

Docente: Roberto Saletti

Autore: Enrico Molinari

Indice

---

1) Presentazione del progetto “libreria di forme d’onda a dente di sega”

1.1) Obiettivo del progetto e ambiente di lavoro

Lo scopo del mio progetto è stato di programmare un sistema FPGA in grado di visualizzare su un monitor CRT una piccola libreria di forme d’onda a dente di sega a varia frequenza, per le quali fosse possibile modulare fase e ampiezza.

Al fine di conseguire tale obiettivo ho utilizzato la scheda DE2 della Altera, equipaggiata con un elemento programmabile FPGA chiamato “Cyclone II”, modello EP2C35F672C6, e con una serie di dispositivi periferici (periferici rispetto all’elemento programmabile su cui programmare la rete logica implementante l’intelligenza del sistema), come memorie, convertitori analogico -> digitale e digitale -> analogico, tastiere, ecc…

Per realizzare i blocchi della rete logica da programmare in hardware, via canale USB-blaster, sull’elemento programmabile FPGA, ho utilizzato l’ambiente di sviluppo Quartus II, versione 8.1, sempre della Altera. Ho descritto le funzioni logiche implementanti i vari blocchi della rete per mezzo del codice Verilog di descrizione dell’hardware.

Ho usato, oltre chiaramente all’FPGA la cui programmazione realizza l’intelligenza che pilota il resto del sistema, alcuni dispositivi periferici, utilizzati appunto dal Cyclone II, una volta programmato, durante il suo funzionamento. I dispositivi, montati sulla DE2, che ho utilizzato sono:

- 7 commutatori switch (SW[6]…SW[0]) e i 4 pushbuttons (KEY[3]…KEY[0]) in qualità di primo stadio per l’acquisizione di dati provenienti dall’esterno, ossia per l’acquisizione delle impostazioni settate dall’utente della scheda ai fini della modulazione di frequenza, fase e ampiezza del dente di sega da visualizzare
- L’oscillatore quarzato a 50 MHz come sorgente dalla quale trarre il segnale di clock per la sincronizzazione di tutti i blocchi del sistema programmato
- La memoria SRAM IS61LV25616 della ISSI a 512 Mbytes, organizzati secondo una struttura a 256 K locazioni, ciascuna di 16 bits (2 bytes), per l’immagazzinamento dei dati portatori dell’informazione relativa alle varie forme d’onda visualizzabili
- Il convertitore video digitale->analogico a 10 bits, a 3 canali paralleli (3 DACs a 10 bits), modello ADV7123 della Analog Devices, per il pilotaggio di un monitor a raggi catodici secondo lo standard analogico XGA 1024x768@60Hz
- Il connettore VGA D-SUB a 16 PINs, pilotato sia dai 3 DACs di cui sopra (i 3 colori primari) che direttamente dall’FPGA (sincronismo orizzontale e verticale), da collegare al monitor CRT
- Il canale USB-blaster attraverso il quale i programming files (la “mappa dei fusibili”, solo per rendere l’idea) generati dall’ambiente di sviluppo dopo il compilamento del codice

sorgente verilog ed il successivo “placing & routing” del codice oggetto, vengono programmati sul Cyclone II

Riporto qui di seguito una figura descrittiva dell’ambiente complessivo nel quale mi sono trovato ad operare.

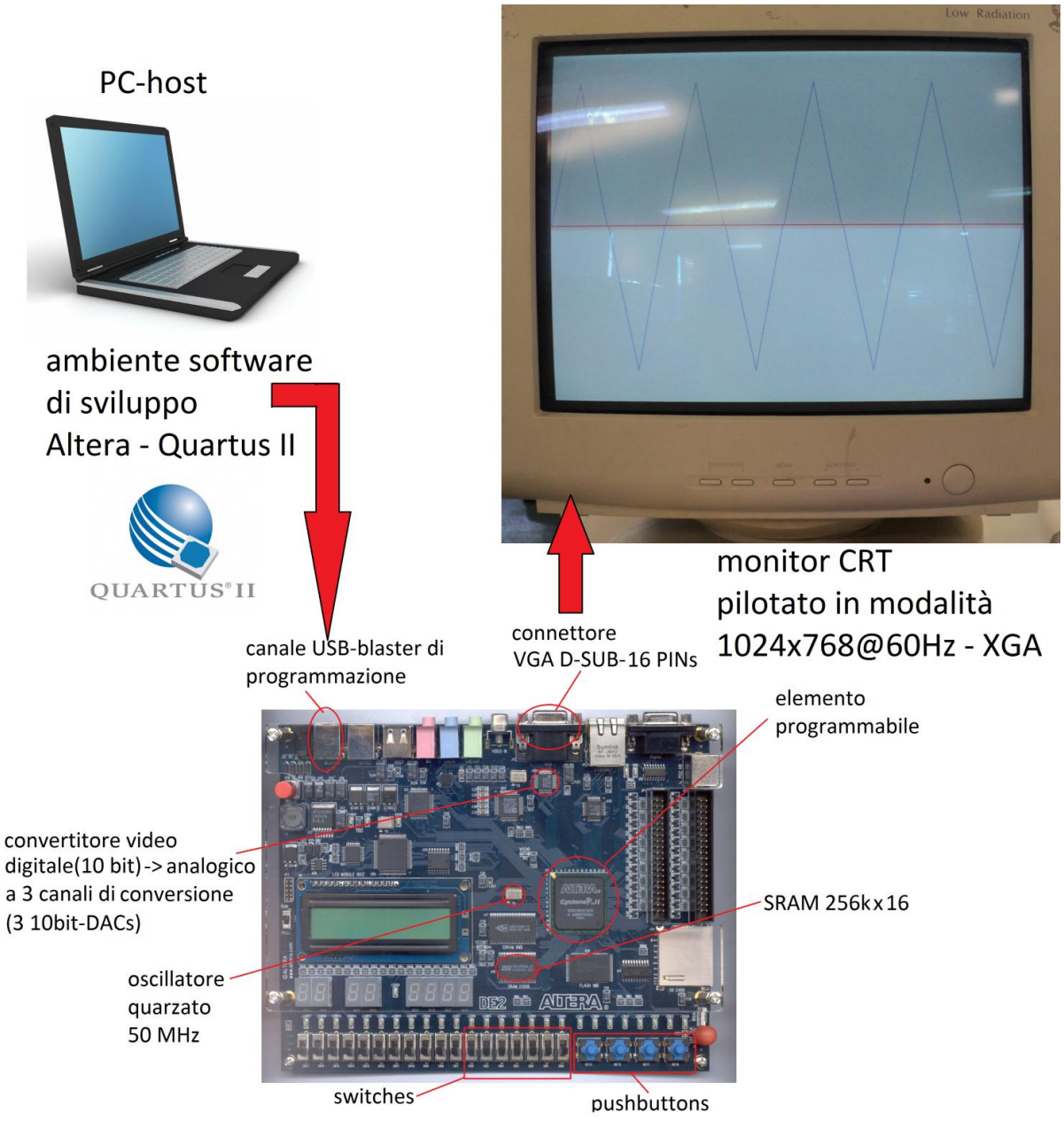


Fig. 1

L’ambiente complessivo nel quale mi sono trovato ad operare: personal computer host sul quale gira il software Altera-Quartus II di sviluppo per FPGA, la scheda DE2 della Altera, un monitor CRT per la visualizzazione delle forme d’onda richieste dall’utente

Qui in basso invece riporto lo schematic rappresentativo del top-level file “DENTE_SEGA_system.bdf” (Block Diagram File), raffigurante i blocchi logici e le interconnessioni implementanti la rete logica, puramente combinatoria, da programmare sull’FPGA. Il funzionamento di questa rete e la sua interazione con le periferiche montate sulla scheda DE2, oltre che con il terminale video, determinano la visualizzazione delle forme d’onda a dente di sega della libreria.

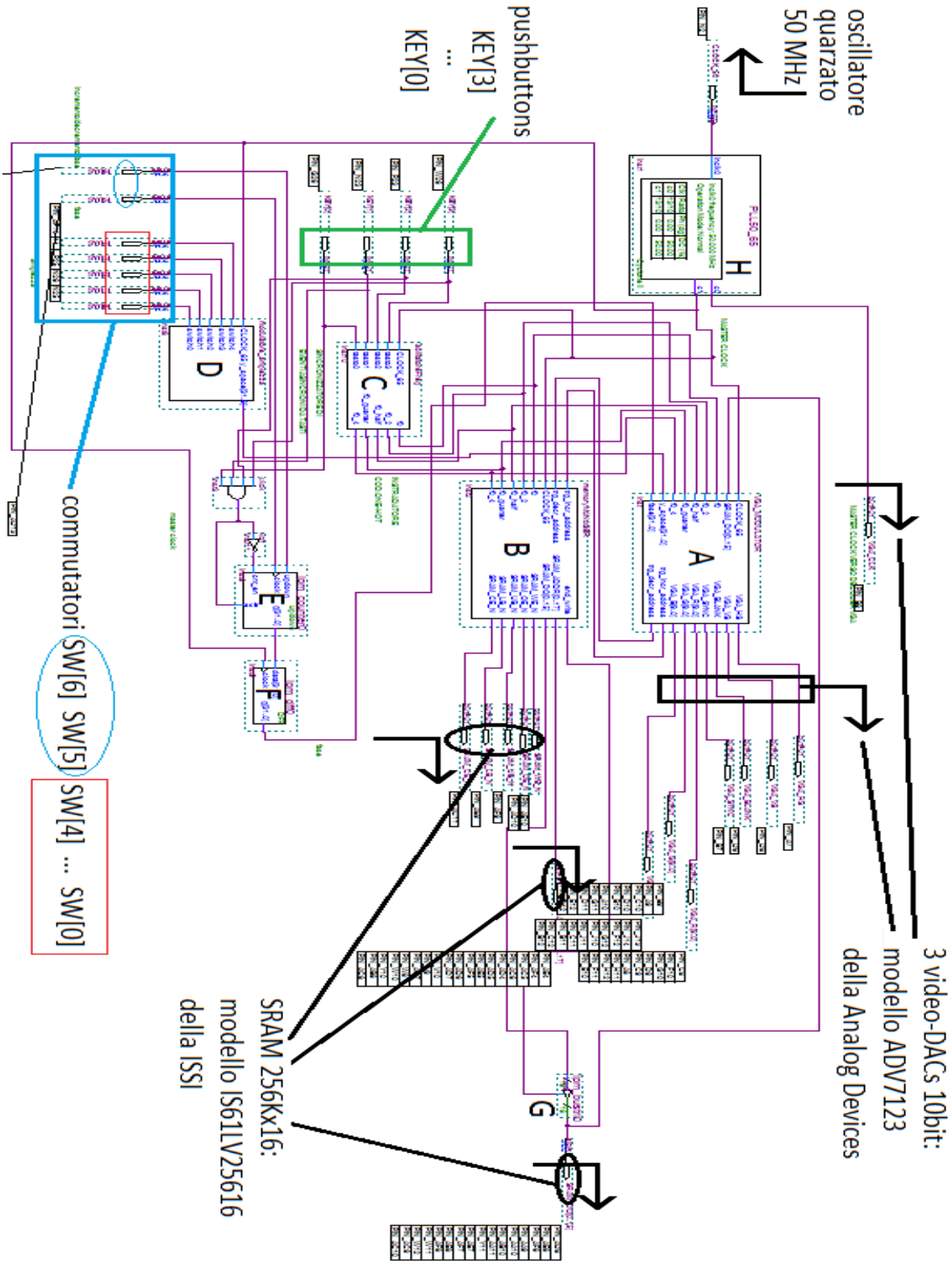


Fig. 2

Schematic rappresentativo del top-level file “DENTE_SEGA_system.bdf” raffigurante i blocchi logici e le interconnessioni implementanti l’intelligenza da programmare sull’FPGA, che piloterà i dispositivi periferici (sia il terminale video che i dispositivi a bordo della scheda) ai fini della visualizzazione del dente di sega richiesto dall’utente

1.2) Breve descrizione dei blocchi e funzionamento del sistema

L’utente, premendo un qualunque pushbutton (KEY[3]…KEY[0]), può richiedere la visualizzazione di una particolare frequenza del dente di sega. Premendo KEY[0] si richiede la visualizzazione del dente di sega “f0_4” avente un periodo pari a circa un quarto della larghezza dello schermo, premendo invece KEY[1] si richiede la visualizzazione del dente di sega “f0_2” avente un periodo pari a circa metà della larghezza dello schermo, premendo KEY[2] si richiede la visualizzazione del dente di sega “f0_half” avente semi-periodo pari alla larghezza dello schermo, mentre premendo KEY[3] si richiede la visualizzazione del dente di sega “f0_quarter” avente un quarto di periodo pari alla larghezza dello schermo (di fatto una rampa). Se l’utente non preme alcun pushbutton oppure imposta una combinazione qualunque diversa da quelle descritte sopra, viene visualizzato il dente di sega di default “f0”, avente periodo pari alla larghezza dello schermo. Il blocco C serve al sistema ad acquisire l’informazione circa la frequenza richiesta dall’utente e ad imporre ai blocchi A e B, per mezzo dell’accensione del corretto segnale di enable, fra i 5 disponibili in uscita (gli altri 4 rimarranno spenti), il giusto comportamento da seguire, al fine della corretta visualizzazione su monitor.

L’utente, commutando SW[6], impone se la fase della forma d’onda da visualizzare deve incrementare oppure decrementare lungo l’asse, mentre commutando SW[5], ad ogni switching DOWN -> UP (allontanamento della levetta del commutatore dal bordo della scheda), l’utente imposta l’entità, il modulo, in termini di numero di pixels, di tale sfasamento. Lo stato DOWN oppure UP di SW[6] impone al blocco E, che è un contatore, il senso del conteggio, ovvero se deve eseguire un up-counting oppure un down-counting, mentre le commutazioni DOWN -> UP di SW[5] costituiscono gli eventi contati da E, di volta in volta incrementanti oppure decrementanti il numero memorizzato, all’uscita del contatore E, in corrispondenza dell’evento precedente. Il blocco E eroga un numero, un valore, il quale attraversando il “macro” flip-flop D positive edge triggered, ossia il blocco F, per essere sincronizzato con il clock di sistema, va in ingresso al blocco A, così che questo possa visualizzare lo sfasamento, in termini di pixels, rappresentato proprio da quel valore “contato” da E.

L’utente, commutando opportunamente i 5 commutatori SW[4]…SW[0], può richiedere il progressivo incremento oppure il decremento dell’ampiezza del dente di sega f0_4. Il dato rappresentativo dell’informazione circa l’entità dello “stiramento”, lungo l’asse delle ordinate, richiesto dall’utente, viene erogato dal blocco D, che funge appunto da selezionatore (abilitatore) di ampiezza, e inviato in ingresso al blocco A per la visualizzazione.

Il blocco B serve alla scrittura in memoria SRAM, montata sulla scheda DE2, dell'informazione relativa alle forme d'onda delle 5 frequenze di dente di sega che devono costituire la libreria. B accede (indirizzandole) alle locazioni della SRAM, scrive, attraverso il buffer/bus tri-state G, reso logicamente trasparente, gli opportuni dati della forma d'onda appena selezionata dall'utente mediante i 4 pushbuttons, dopodiché alla fine della scrittura dei dati della forma d'onda richiesta manda in alta impedenza G. Su sollecitazione del blocco A, B accede alle locazioni, prima accedute per la scrittura, anche in fase di lettura, per consentire al blocco A di poter valutare in corrispondenza di ciascun fronte positivo del clock di sistema, su uno dei suoi ingressi, il dato memorizzato di volta in volta nella giusta locazione, al fine della visualizzazione della forma d'onda alla quale quei dati, puntati in SRAM da B, si riferiscono.

Il blocco A implementa la funzionalità responsabile del corretto pilotaggio del convertitore digitale -> analogico e del connettore D-SUB VGA. A, per mezzo di 2 suoi segnali di uscita, entranti nel blocco B, impone a quest'ultimo l'indirizzamento, di volta in volta, cioè ad ogni fronte in salita del clock di sistema, della giusta locazione SRAM, oltre al senso, crescente o decrescente, con il quale B, che gestisce direttamente la memoria, deve scorrere le locazioni di quest'ultima. Questo duplice verso di scorrimento della SRAM, da parte di B, richiedibile da A in fase di lettura dei dati, ossia in fase di stampa su monitor, è utile per la visualizzazione di semionde positive e negative, risparmiando metà del numero di locazioni SRAM che altrimenti sarebbero state necessarie se non avessi previsto tale sistema di indirizzamento. In tal modo B punta sempre la corretta locazione SRAM, pertanto A, su uno dei suoi ingresso, quello cioè che lo mette in comunicazione con la SRAM, trova sempre, ad ogni fronte positivo del clock di sistema, il giusto dato. Questo dato viene elaborato da A, così da visualizzare correttamente il dente di sega richiesto dall'utente. A pertanto deve generare ad ogni colpo di clock le 3 parole digitali che, una volta decodificate in analogico dai 3 DACs della Analog Devices, costituiranno i segnali in corrente relativi ai 3 colori primari RGB necessari alla corretta colorazione di ciascun pixel dello schermo. Inoltre A impone, direttamente sul connettore VGA, ossia direttamente sul monitor CRT, i segnali di sincronizzazione responsabili della deflessione orizzontale e verticale del pennello elettronico all'interno del tubo a raggi catodici.

Il blocco H è un PLL che accetta in ingresso il clock a 50 MHz proveniente da uno dei 2 oscillatori al quarzo montati sulla scheda DE2 e genera, in uscita, un clock alla frequenza desiderata (non tutti i rapporti di conversione sono possibili), che nel mio caso è di 65 MHz (rapporto 13/10 realizzabile), essendo tale frequenza di clock necessaria in ingresso al convertitore digitale -> analogico: ho pertanto piegato tutta la tempistica di sincronismo dell'intero sistema programmato a questa necessità del DAC video.

Qui di seguito riporto un diagramma di flusso descrivente il livello fondamentale di funzionamento della rete logica che ho programmato sul Cyclone II, ovvero quella parte di logica che gestisce la visualizzazione della frequenza di dente di sega selezionata dall'utente.

π = valutazione dello stato dei tasti KEYs ad ogni fronte positivo del clock di sistema, all'atto dell'attivazione di always@(posedge clock65MHz) del blocco A e di quello B

α = processo destinato a durata molto poco, nell'ordine di una decina di µs; a fine scrittura inizia la fase di lettura dati da SRAM, durante la quale solo il processo β continua ad essere iterato

β = loop infinito, visualizzazione ad oltranza del dente di sega selezionato

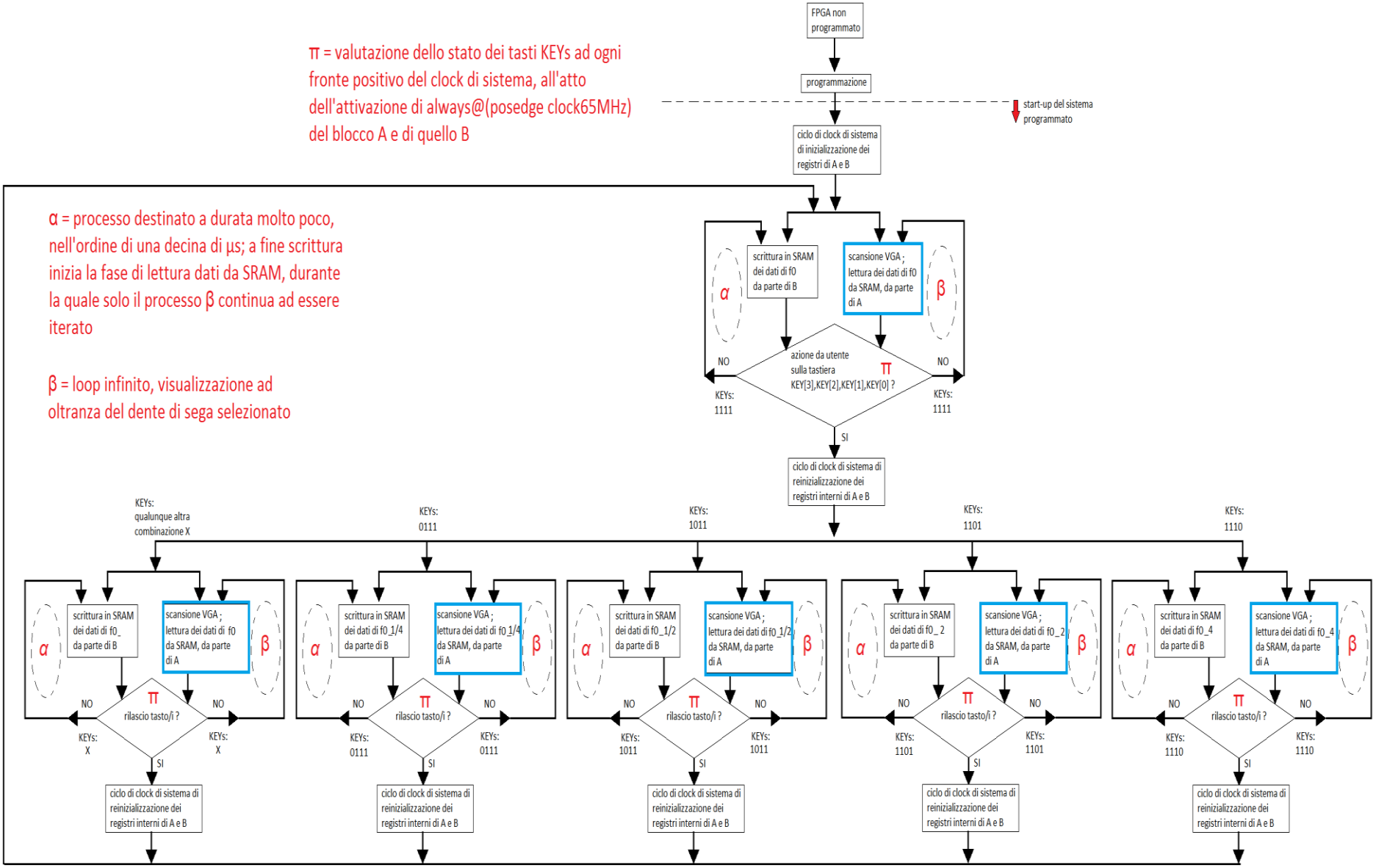
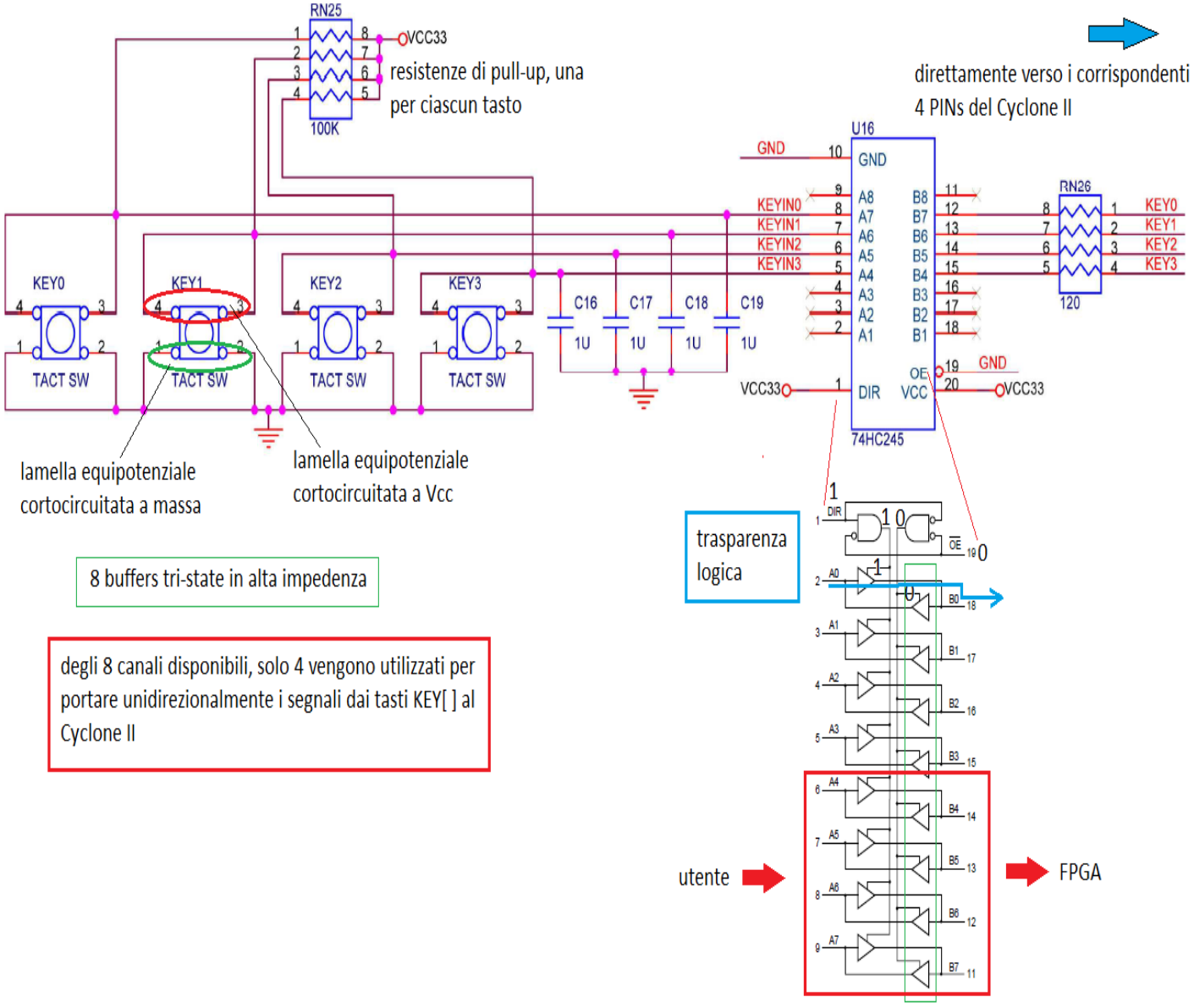


Fig. 3

Diagramma di flusso descrivente il livello fondamentale di funzionamento della rete logica, ovvero quella parte di logica che gestisce la visualizzazione della frequenza di dente di sega selezionata dall'utente

2) Selezione della frequenza del dente di sega

La scheda DE2 dispone di una tastiera formata da 4 tasti di tipo "pushbutton", che io ho utilizzato per consentire all'utente la scelta della frequenza di dente di sega che desidera visualizzare sul monitor CRT.



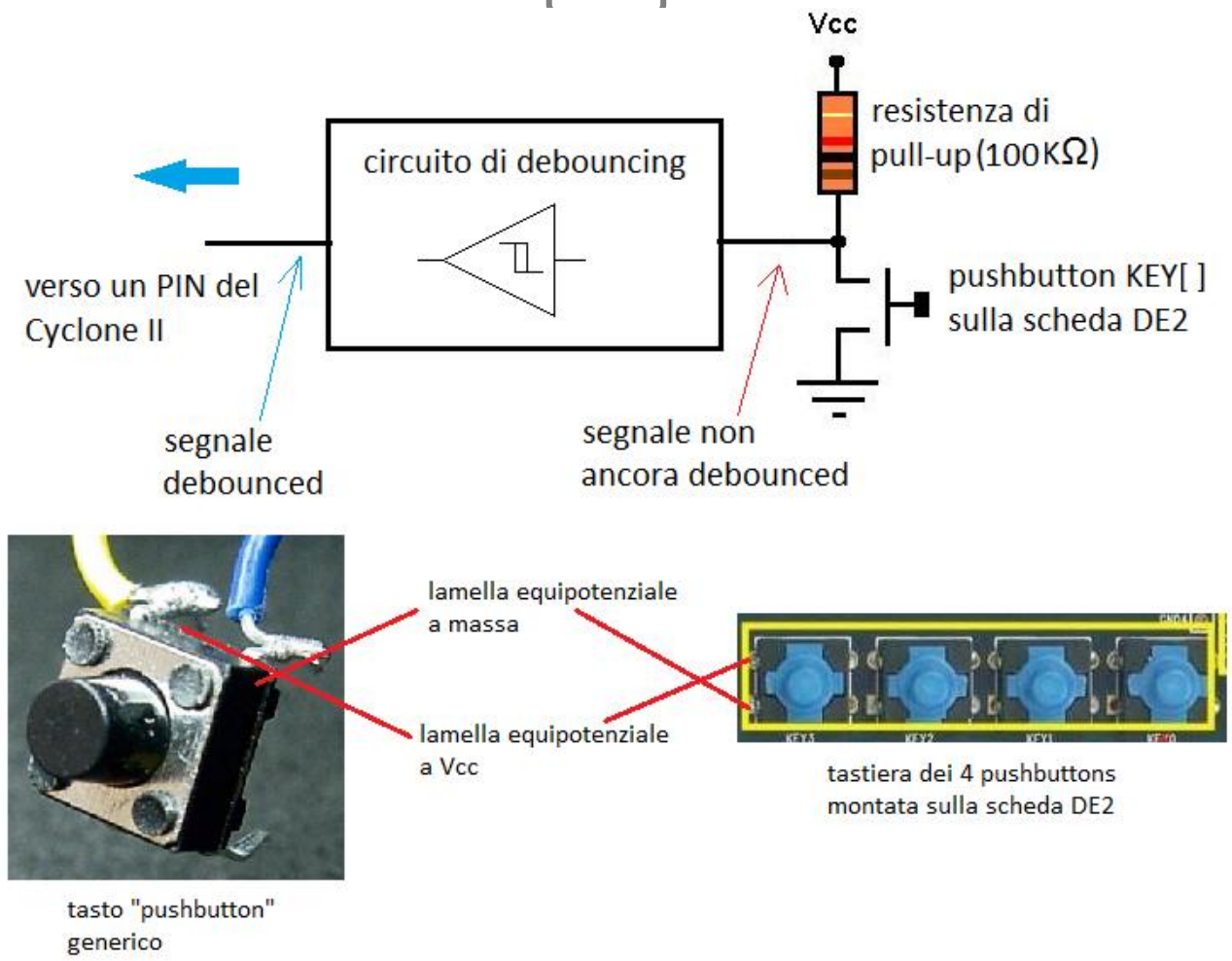


Fig. 4

Componentistica costituente il primo stadio di acquisizione dati per il selezionatore di frequenze di dente di sega

La pressione di ciascun pushbutton da parte dell'utente provoca la cortocircuitazione fra la lamella metallica equipotenziale 1-2 collegata a massa e l'altra lamella equipotenziale 3-4 collegata, tramite una resistenza di pull-up di valore molto elevato (100KΩ), all'alimentazione Vcc = 3,3 V. Infatti prima della pressione del tasto di fatto non scorre corrente lungo la resistenza di pull-up, quindi non vi è caduta di tensione ai suoi terminali, quindi la lamella collegata al circuito di debouncing è di fatto cortocircuitata a Vcc. In seguito alla pressione del tasto, si crea il corto fra le 2 lamelle, la corrente può attraversare il corto Vcc -> GND appena resosi disponibile, si viene a creare una forte caduta di tensione ai terminali della resistenza di pull-up (il suo valore elevato sventa il pericolo di conflitto logico 1 vs 0 sulla lamella 3-4 all'atto della pressione del tasto), pertanto sulla lamella 3-4 va ad insistere il valore logico basso (0 V). Il segnale in tensione costituente il potenziale della lamella 3-4 passa attraverso una squadra RC e successivamente attraverso uno dei 4 canali utilizzati (sugli 8 disponibili) di bufferaggio tri-state messi a disposizione dal tranceiver 74HC245, della Philips Semiconductors, così da ricevere un'azione di filtraggio passa-basso e dunque di debouncing, cioè di stabilizzazione del proprio transitorio 1 -> 0, affetto (come del resto il transitorio 0 -> 1 di rilascio del tasto) dal noto problema del ringing. Ciascuno

dei 4 canali di acquisizione dati da tastiera di pushbuttons va poi ad insistere direttamente sul corrispondente PIN del Cyclone II.

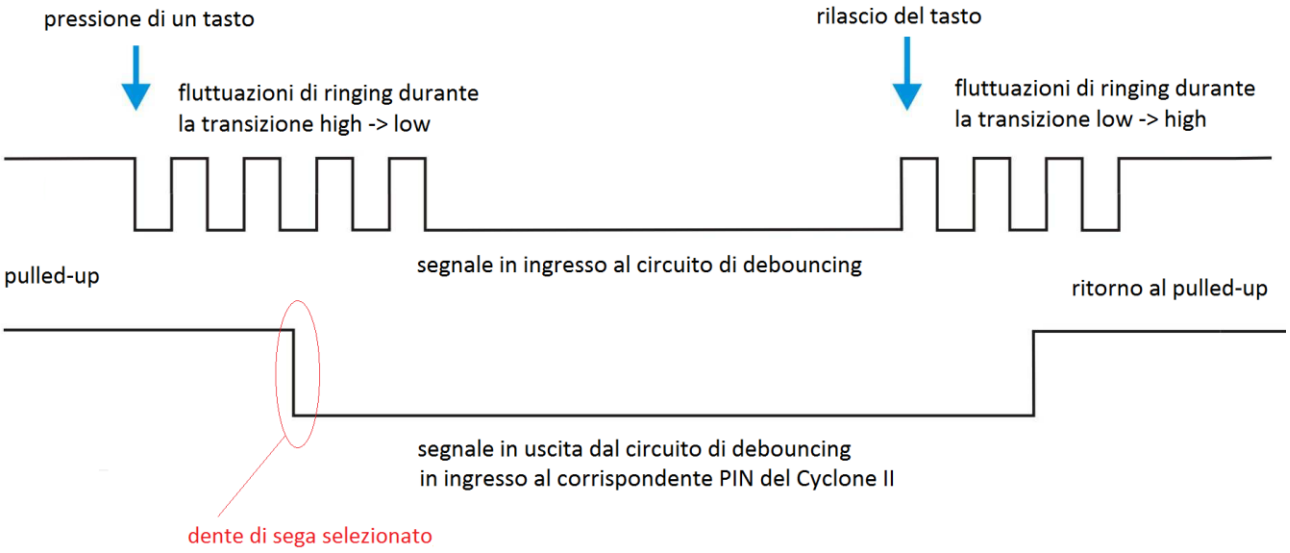


Fig. 5

Segnale in ingresso al circuito di debouncing e corrispondente segnale debounced in ingresso all'elemento programmabile FPGA

Dallo schematic del file "DENTE_SEGA_system.bdf" estraiamo la parte coinvolta nell'acquisizione e successiva gestione dei comandi di selezione delle frequenze di dente di sega.

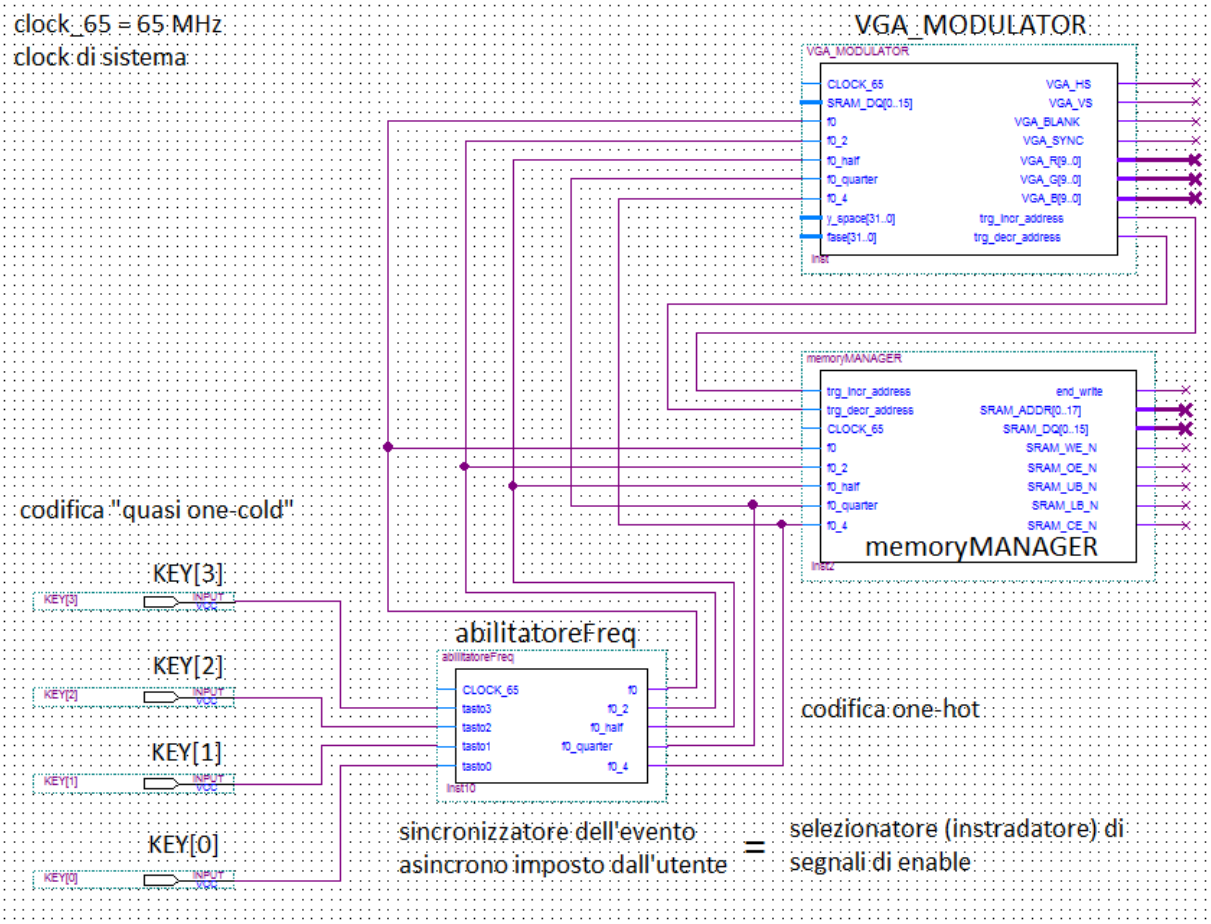


Fig. 6

Porzione dello schematic del file “DENTE_SEGA_system.bdf” coinvolta nell’acquisizione e successiva gestione dei comandi di selezione delle frequenze di dente di sega

La situazione di default è quella che vede i 4 PINs di ingresso al blocco abilitatoreFreq, cioè il blocco adibito alla selezione della frequenza da visualizzare, tirati su a Vcc (pulled-up), situazione che si verifica quando l’utente non tiene premuto alcun tasto. La forma d’onda che viene visualizzata su monitor è quella di default f0, avente periodo pari all’intera larghezza dello schermo. L’utente può arbitrariamente premere uno qualunque dei 4 pushbuttons per richiedere la visualizzazione della frequenza corrispondente. Tale pressione è ovviamente asincrona, cioè non allineata temporalmente con il fronte positivo del clock di sistema, il quale sincronizza tutti i blocchi della rete da programmare sul Cyclone II, pertanto l’invio del segnale basso KEY[x] (x = 0,1,2,3) direttamente ai blocchi VGA_MODULATOR e memoryMANAGER costituirebbe un’evidente violazione delle regole del progetto sincrono-statico. Per evitare pertanto sicure violazioni dei tempi di setup e di hold dei flip-flop interni sia al blocco VGA_MODULATOR che a quello memoryMANAGER, ho utilizzato la rete abilitatoreFreq, positive edge triggered, in modo da sincronizzare, con il fronte positivo del clock di sistema, l’evento di abilitazione di ciascuna frequenza percepito da VGA_MODULATOR e da memoryMANAGER, i quali appunto leggono tale informazione ad ogni fronte positivo del clock di sistema.

Il blocco abilitatoreFreq è un instradatore/selezionatore di segnali di enable che riceve in ingresso una certa combinazione di livelli logici KEY[3]…KEY[0] e accende, in corrispondenza di ciascuna stringa di livelli logici (stringa di 4 bits) un solo segnale di enable. La codifica degli ingressi potremmo definirla “quasi one-cold”, nel senso che di regola sono tutti 1, oppure tutti 1 tranne uno, che è appunto il bit “freddo” (0), corrispondente al tasto premuto, oppure altre combinazioni di 1 e 0 che però selezionano la stessa frequenza associata alla stringa di 4 livelli logici alti, ossia la frequenza di default f0. Il controllo dello stato della stringa KEY[3]…KEY[0] viene eseguito da abilitatoreFreq ad ogni fronte positivo del clock di sistema, come si evince dal costrutto always@ del codice verilog che realizza la funzionalità del blocco abilitatoreFreq, che riporto qui di seguito, così l’eventuale aggiornamento della frequenza abilitata viene eseguito in modo perfettamente sincrono. La codifica delle uscite di abilitatoreFreq è perfettamente “one-hot”, poiché soltanto uno fra i 4 segnali di enable è acceso (1), ovvero può essere visualizzata soltanto una frequenza alla volta.

```
always@(CLOCK_65)

begin
if( (tasto0 == 1) && (tasto1 == 1) && (tasto2 == 1) && (tasto3 == 1) )
 // i 4 pushbottons => alzati, non premuti, i 4 pins serviti sono a Vcc
 // cioè sono ingressi pulled-up => situazione di default
 begin
  REG_f0 <= 1;
  REG_f0_2 <= 0; // abilitato il dente di sega avente periodo = larghezza schermo
  REG_f0_half <= 0;
  REG_f0_quarter <= 0;
  REG_f0_4 <= 0;
 end
else if( (tasto0 == 0) && (tasto1 == 1) && (tasto2 == 1) && (tasto3 == 1) )
 begin
  REG_f0 <= 0;
  REG_f0_2 <= 0; // abilitato il dente di sega avente periodo = larghezza schermo / 4
  REG_f0_half <= 0;
  REG_f0_quarter <= 0;
  REG_f0_4 <= 1;
 end
else if( (tasto0 == 1) && (tasto1 == 0) && (tasto2 == 1) && (tasto3 == 1) )
 begin
  REG_f0 <= 0;
  REG_f0_2 <= 1; // abilitato il dente di sega avente periodo = larghezza schermo / 2
  REG_f0_half <= 0;
  REG_f0_quarter <= 0;
  REG_f0_4 <= 0;
 end
else if( (tasto0 == 1) && (tasto1 == 1) && (tasto2 == 0) && (tasto3 == 1) )
 begin
  REG_f0 <= 0;
  REG_f0_2 <= 0; // abilitato il dente di sega avente periodo = 2*larghezza schermo
  REG_f0_half <= 1;
  REG_f0_quarter <= 0;
  REG_f0_4 <= 0;
 end
else if( (tasto0 == 1) && (tasto1 == 1) && (tasto2 == 1) && (tasto3 == 0) )
 begin
  REG_f0 <= 0;
  REG_f0_2 <= 0; // abilitato il dente di sega avente periodo = 4*larghezza schermo
  REG_f0_half <= 0; // larghezza schermo = un quarto del dente di sega
  REG_f0_quarter <= 1;
  REG_f0_4 <= 0;
 end
else // qualunque altro stato logico dei 4 ingressi, dei 4 tasti KEY[] => abilita il dente di default
```

Fig. 7

Costrutto always@ del codice verilog che descrive la funzionalità selezionatrice del blocco abilitatoreFreq

Qui di seguito riporto ciò che viene visualizzato su monitor CRT in corrispondenza della pressione dei 4 pushbuttons.

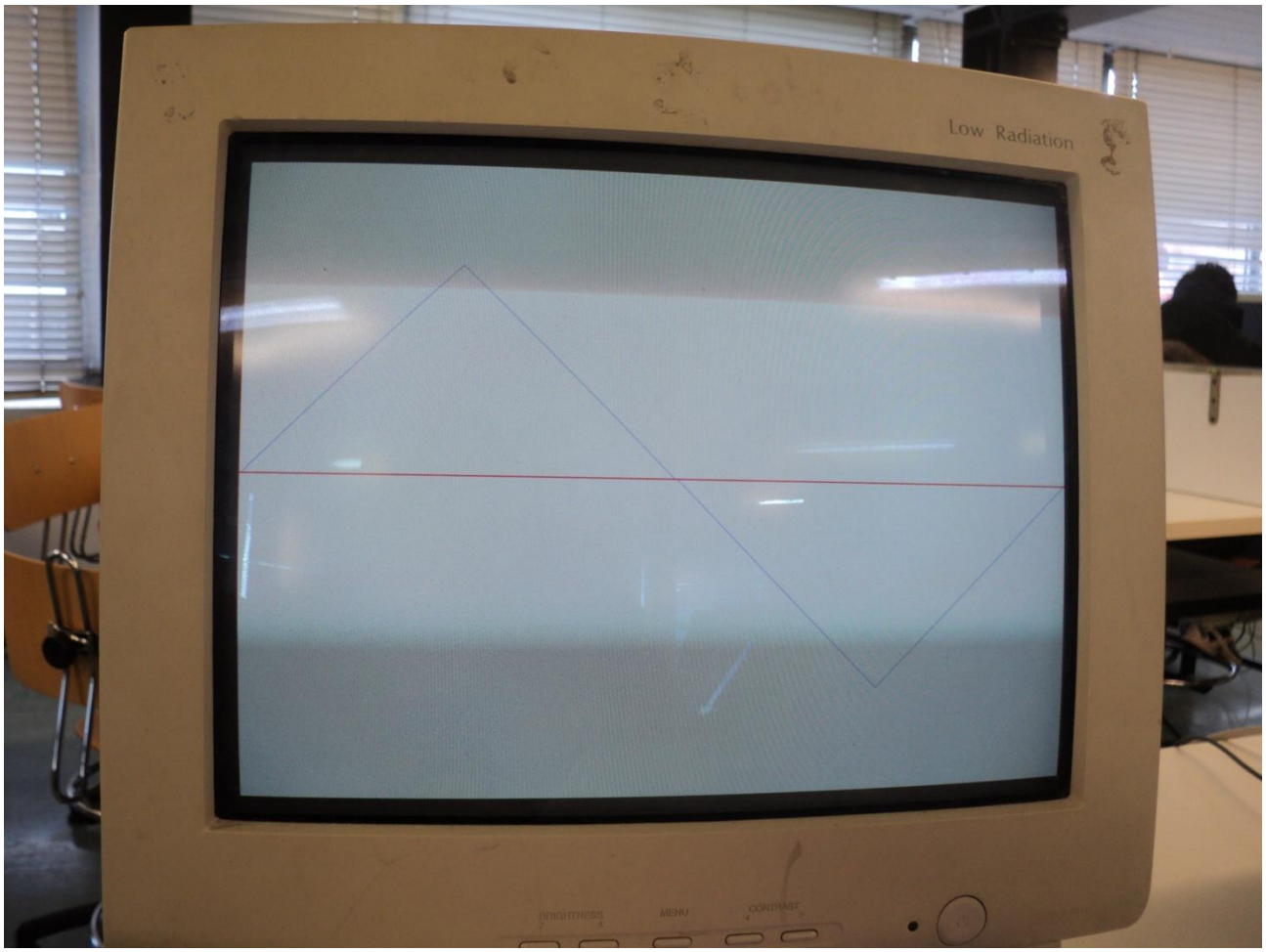


Fig. 8

Frequenza di default “f0”, visualizzata in corrispondenza di assenza di pressione sui 4 pushbuttons oppure in caso di pressione di almeno 2 pushbuttons fra i 4 disponibili

Fig. 9

Frequenza quadrupla “f0_4”, visualizzata in corrispondenza della pressione del pushbutton KEY[0]

Fig. 10

Frequenza doppia “f0_2”, visualizzata in corrispondenza della pressione del pushbutton KEY[1]

Fig. 11

Frequenza dimezzata “f0_half”, visualizzata in corrispondenza della pressione del pushbutton KEY[2]

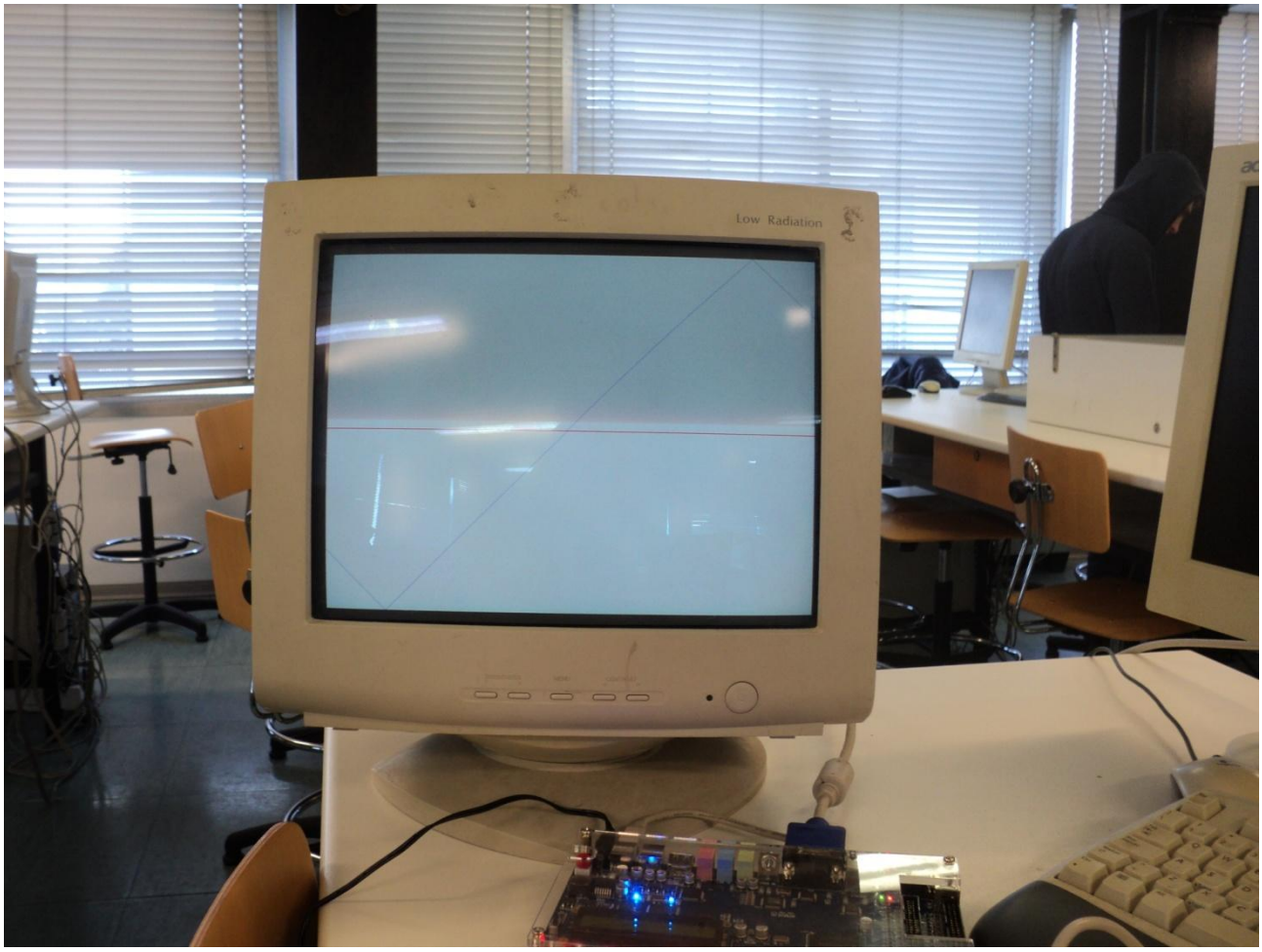


Fig. 12

Frequenza “f0_quarter”, visualizzata in corrispondenza della pressione del pushbutton KEY[3]

3) Modulazione della fase delle forme d'onda a dente di sega

La scheda DE2 dispone di 18 commutatori (switches) che possono essere utilizzati dall'utente per inserire valori logici. Al contrario dei 4 tasti di tipo "push-button", i 18 switches forniscono segnali di tensione che non vengono trattati da opportuni circuiti di debouncing. Il mancato filtraggio delle fluttuazioni di ringing che affliggono ciascuno dei 18 segnali, all'atto di una commutazione, impone un uso poco frequente di questi commutatori: è bene, infatti, usarli per inserire dei valori logici (o per meglio dire, delle combinazioni di valori logici) che determinano uno stato di funzionamento abbastanza prolungato del nostro FPGA, così che il ringing dei segnali erogati dagli switches e direttamente inviati ai corrispondenti PINs del Cyclone II risulti attivo per un periodo di tempo trascurabile rispetto al lasso di tempo durante il quale ciascun commutatore fornisce al Cyclone II una valore stabile di tensione. Tale valore è pari a 0 V (valore logico basso) nel caso in cui il commutatore sia rivolto verso il bordo della scheda (stato "DOWN" dello switch), pari invece a 3.3 V (valore logico alto) nel caso in cui il commutatore sia rivolto nel verso opposto (stato "UP").

Fig. 13

Tastiera dei 18 switches montata sulla scheda DE2 della Altera

Per realizzare sullo schermo del monitor CRT, per una qualunque frequenza (e per una qualunque ampiezza, nel caso del dente di sega f0_4) appartenente alla libreria del dente di sega, uno sfasamento che fosse deciso arbitrariamente dall'utente della scheda, ho deciso di utilizzare i commutatori SW[6] e SW[5] della tastiera di switches, che chiaramente costituiscono il primo stadio di acquisizione dati relativamente alla fase modulata dall'utente, più delle risorse programmate sull'FPGA dai programming files, compilati dall'assembler implementato in Quartus II. Tali risorse sono:

1) Una porta AND4
2) Un inverter
3) Un contatore a 32 bit
4) Un registro costituito da 32 flip-flop D positive edge-triggered messi in parallelo

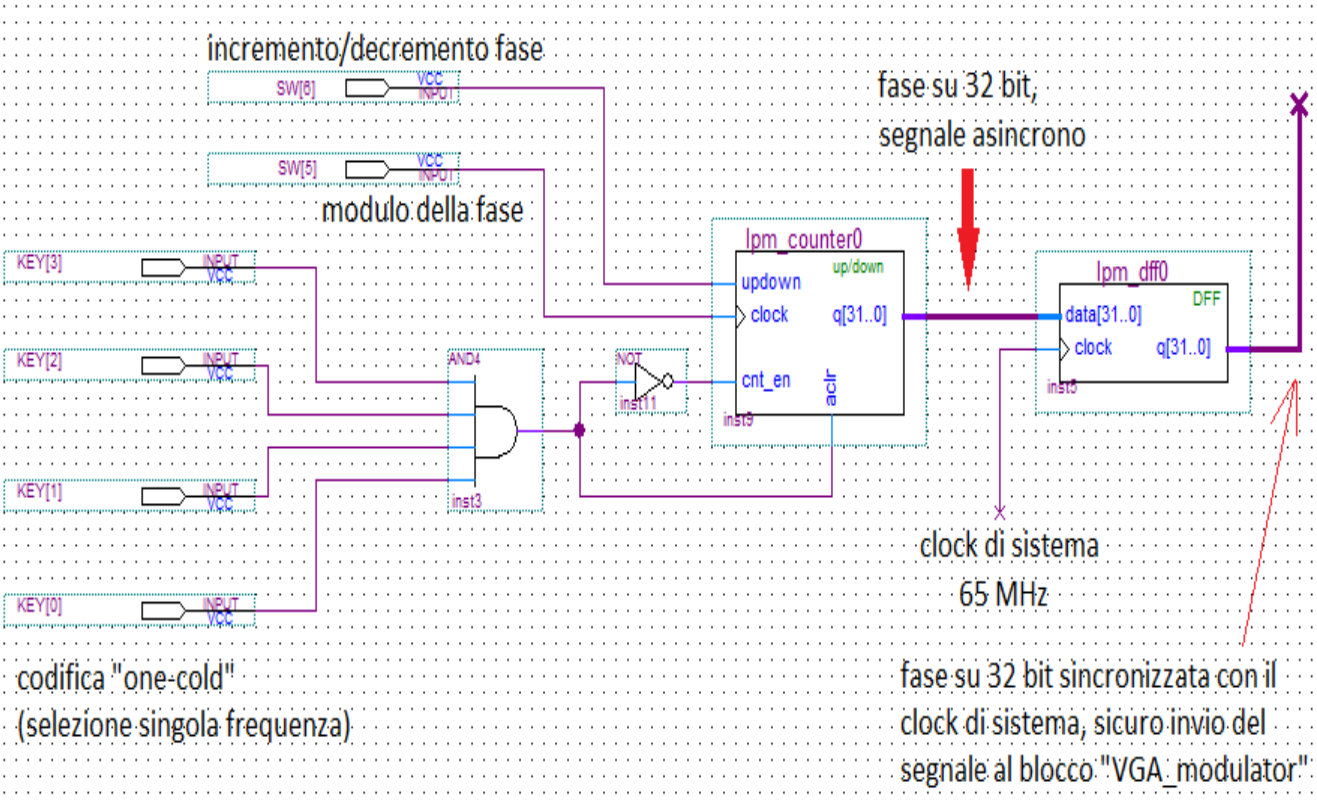


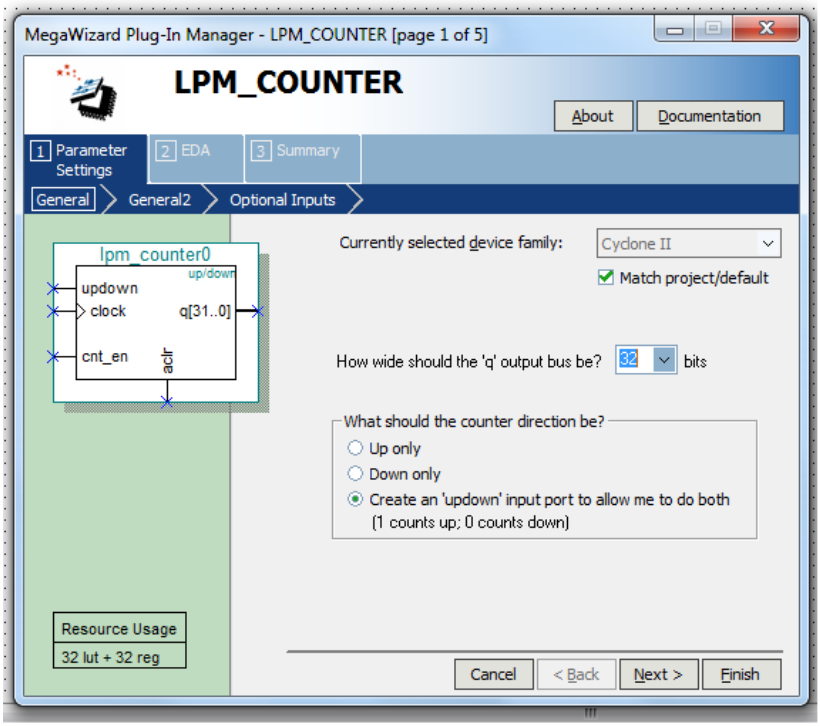
Fig. 14

Parte dello schematic del top-level file “DENTE_SEGA_system.bdf” finalizzata all’implementazione della funzione di modulazione di fase delle forme d’onda

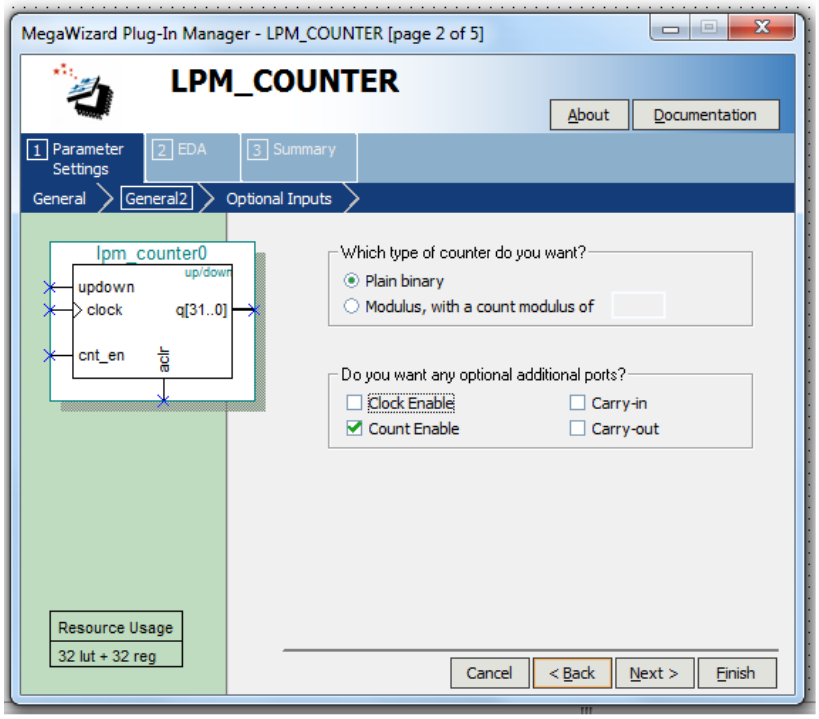
Se non premiamo alcun pushbutton, ovvero se siamo nello stato di default (frequenza di default su monitor), allora la porta AND4 prende in ingresso 4 livelli logici alti, fornisce un valore logico alto, il quale va a insistere sull’ingresso “aclr” (ingresso di clear asincrono dell’uscita q) del contatore e, in modo complementato, su quello “cnt_en” (ingresso di abilitazione al conteggio). Entrambi gli ingressi del contatore sono attivi alti, quindi il contatore è disabilitato a contare, pertanto se arrivasse un fronte positivo sul suo ingresso di clock (cioè un evento da contare) non si avrebbe alcun incremento o decremento di q, e mantiene sull’uscita q il valore logico 0, distribuito sui 32 bits disponibili. Nel momento in cui viene premuto almeno uno dei 4 tasti, ovvero nel momento in cui viene selezionata dall’utente una certa frequenza non di default, oppure la stessa frequenza di default se vengono premuti, ad esempio, contemporaneamente 2 tasti, l’uscita della AND4 va a 0, l’ingresso di clearing (dell’uscita q) asincrono viene disabilitato, q pertanto non è più costretta ad assumere costantemente il valore 0, in altre parole non è più “congelata” a 0, inoltre cnt_en è diventato 1, quindi il contatore è abilitato a contare in modo progressivo, ossia a incrementare il valore di q all’arrivo di un fronte positivo del suo segnale di clock, ovvero all’atto dello spostamento DOWN -> UP della levetta di SW[5], se il segnale updown è 1, cioè se la levetta SW[6] è in stato UP. Invece se il segnale updown è 0, cioè se la levetta SW[6] è in stato DOWN, allora il contatore è abilitato a contare in modo regressivo, ossia a decrementare il valore di q all’arrivo di

un fronte positivo del clock, ovvero sempre all'atto dello spostamento DOWN -> UP della levetta di SW[5].

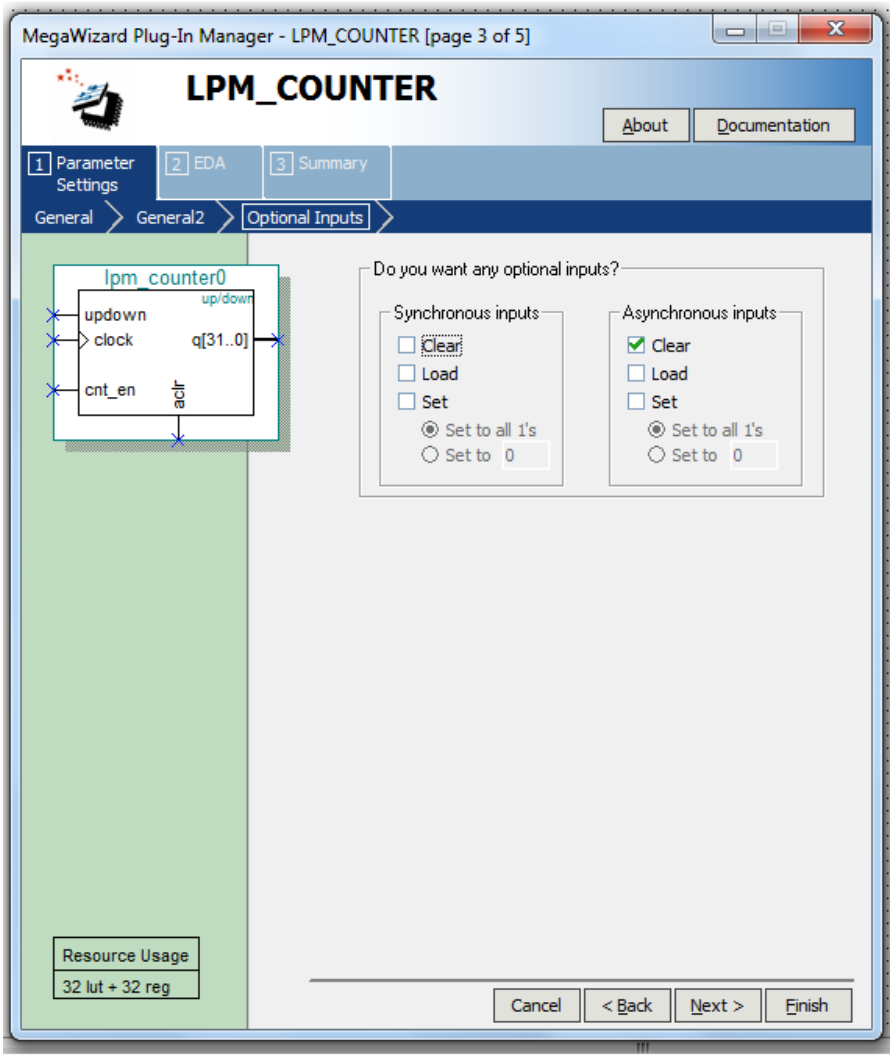
Ho deciso di realizzare il contatore in questione utilizzando la relativa mega-wizard messa a disposizione dall'ambiente di sviluppo Quartus II. Riporto qui di seguito le fasi salienti del percorso software guidato, durante le quali viene chiesto al progettista di impostare caratteristiche del contatore binario quali la larghezza del bus dati q di uscita, l'up-counting, il down-counting, ingressi sincroni/asincroni ecc…



1)



2)

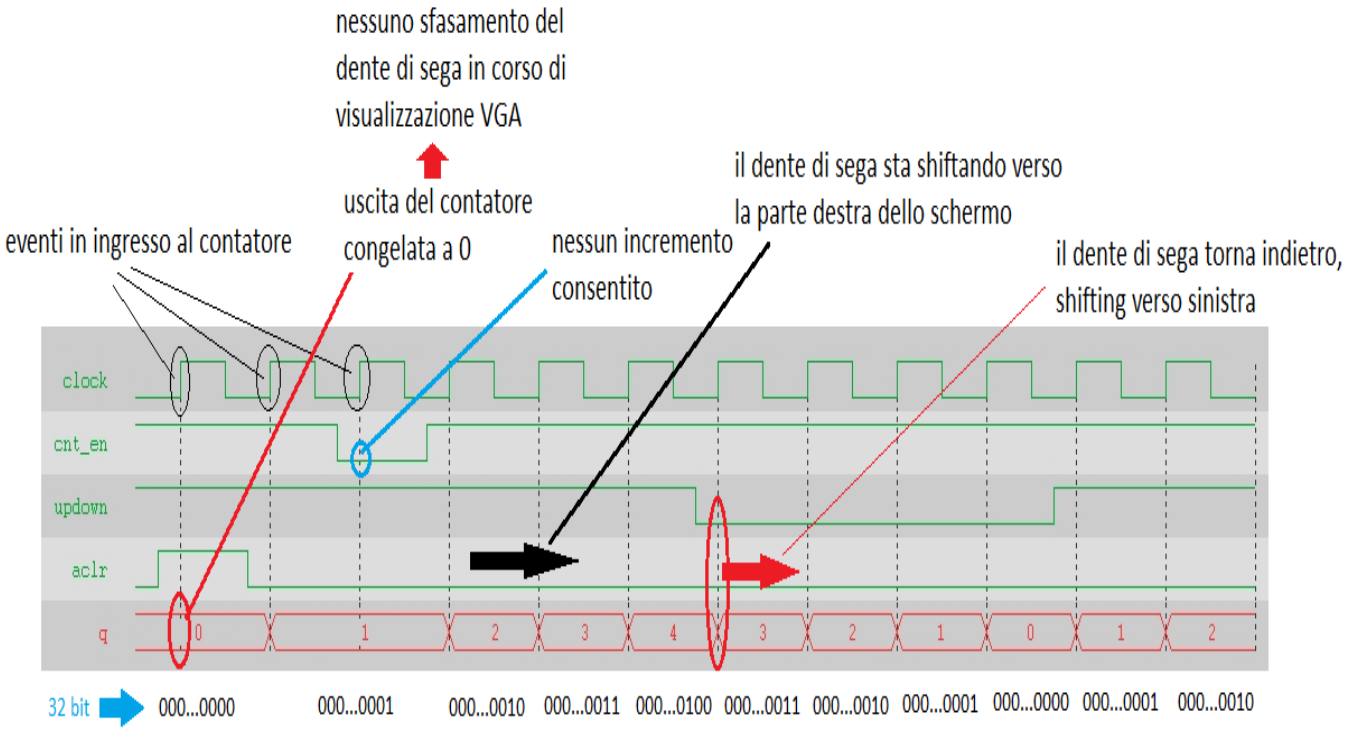


3)

Fig. 15

Mega-wizard messa a disposizione dall'ambiente di sviluppo Quartus II per la realizzazione di un contatore

Il seguente diagramma di forme d'onda spiega chiaramente il funzionamento del contatore.



clock = evento asincrono, segnale modulato dall'utente mediante toggling dello switch SW[5]
updown = evento asincrono, valore impostato dall'utente mediante toggling dello switch SW[6]
cnt_en = attivo alto, up/down counting abilitato al fronte positivo del clock (SW[5]) quando è tenuto premuto almeno un tasto fra i 4 utilizzati, cioè quando è selezionata una frequenza non di default (uscita della AND4 = 0)
aclr = segnale asincrono di clearing, impone q = 0 quando è attivo (alto), lascia libero q di incrementare/decrementare quando è tenuto premuto almeno un tasto fra i 4 utilizzati
q = segnale asincrono, fase del dente di sega espressa su 32 bit, coefficiente moltiplicativo di 20 pixels

Fig. 16

Funzionamento del contatore

Appena viene visualizzata su monitor una qualunque forma d'onda della libreria del dente di sega, questa appare con fase di default, ossia con fase nulla; in altre parole è nullo, in termini di numero di pixels, lo sfasamento dei pixels "cardine"visualizzati rispetto ai pixels "cardine" di un dente di sega, di stessa frequenza, che potremmo indicare come "forma d'onda di riferimento". Se adesso impostiamo UP il commutatore SW[6] e iniziamo a togglare l'adiacente SW[5], noteremo che ogni volta che la levetta viene spostata in direzione DOWN -> UP (evento "positive edge" del clock del contatore), il dente di sega in corso di visualizzazione riceve un phase-shifting verso destra di 20 pixels, ossia la sua fase, inizialmente nulla, incrementa di 20 pixels. Questo perché ad ogni spostamento DOWN -> UP di SW[5], cioè ad ogni fronte positivo del segnale di clock in ingresso al contatore, la sua uscita q incrementa di 1, e tale incremento si trasmette al numero intero (32 bits)

q[31..0] che costituisce l'uscita del "macro" flip-flop D positive edge-triggered, anch'esso realizzato con una mega-wizard. Quest'ultimo numero intero è trasmesso, mediante un bus largo 32 bits, all'ingresso "fase[31..0]" del blocco VGA_MODULATOR. L'ingresso "fase" rappresenta il multiplo intero φ (fase = 0, 1, 2, 3, … qualunque intero esprimibile su 32 bits), della quantità pari a 20 pixels, di cui si sposta, verso destra, un qualunque dente di sega della libreria, rispetto alla posizione di default (fase = 0). La granularità pari a 20 pixels dell'incremento/decremento di fase è stata decisa osservando un compromesso fra accuratezza dello sfasamento visualizzato e tempi di attesa, ossia numero di commutazioni di SW[5], necessari affinchè si potesse apprezzare uno spostamento sull'asse rosso. Un fattore moltiplicativo 1, in luogo di 20, avrebbe aumentato l'accuratezza della modulazione di fase, ma avrebbe comportato un numero di commutazioni di SW[5] eccessivo al fine di un apprezzabile sfasamento, pari infatti esattamente al numero di pixels di cui si desidera shiftare la forma d'onda. Se ad un certo punto si desidera far tornare indietro, lungo l'asse, il dente di sega, è sufficiente commutare lo switch SW[6] dallo stato UP a quello DOWN, così da abilitare la funzionalità di down-counting del contatore binario. Se adesso continuiamo a togglare SW[5], osserveremo che ad ogni commutazione DOWN -> UP di SW[5] si avrà un decremento di fase pari a 20 pixels. Potremo pertanto riportare il dente di sega a fase nulla oppure proseguire ancora con il down-counting e shiftare la forma d'onda corrente sulle fasi negative. Riporto qui in basso un frammento di codice verilog implementante la funzionalità di phase-shifting del blocco VGA_MODULATOR.

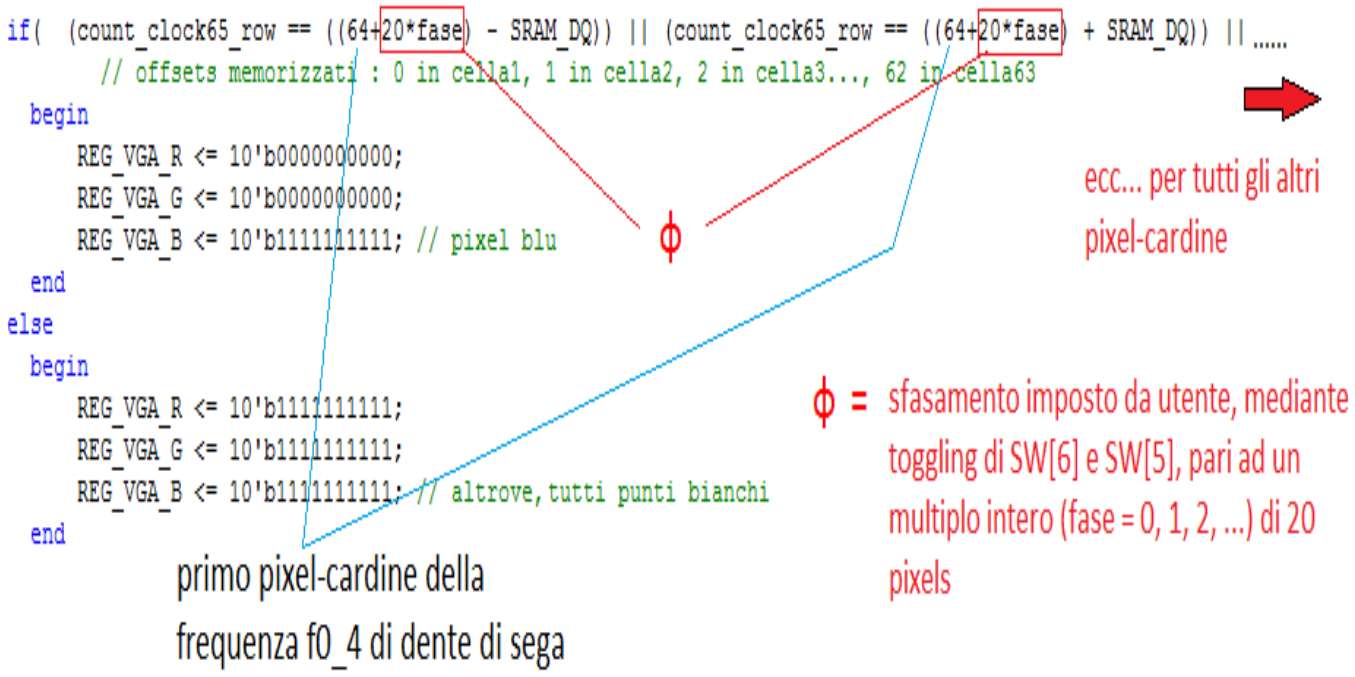


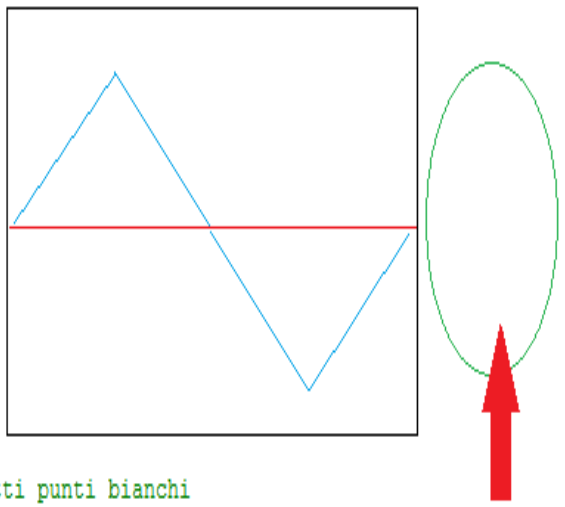
Fig. 17

Descrizione in verilog, interna al blocco VGA_MODULATOR, della funzionalità di sfasamento

Il motivo che mi ha spinto ad utilizzare il macro flip-flop D è legato al tentativo di non violare le regole del progetto sincrono-statico. Infatti ogni volta che l'utente toggla SW[5] dallo stato DOWN a quello UP, si ha l'incremento di q, ossia l'incremento della fase. L'azione di toggling di SW[5], da parte dell'utente, è ovviamente asincrona, assolutamente arbitraria, dunque scorrelata dal segnale di sincronismo "clock di sistema". Il fronte positivo del clock in ingresso al contatore è pertanto un evento asincrono, pertanto anche l'update del valore di q, ossia l'incremento della fase, è un evento assolutamente asincrono. Inviare al sistema, nella fattispecie in ingresso al blocco VGA_MODULATOR, un segnale asincrono, peraltro dal cui valore dipende la corretta visualizzazione del dente di sega corrente, è un'aperta violazione delle regole del progetto sincrono-statico. Per evitare altrimenti sicure violazioni dei tempi di setup e di hold dei flip-flop interni, in questo caso, al blocco VGA_MODULATOR, ho utilizzato la struttura "lpm_dff0"a 32 flip-flop D positive edge-triggered, così da sincronizzare (ovvero allineare temporalmente), con il fronte positivo del clock di sistema, l'evento di incremento/decremento della fase percepito dal blocco VGA_MODULATOR, che deve appunto leggere tale informazione ad ogni fronte positivo del clock di sistema per visualizzare correttamente il dente di sega con lo sfasamento richiesto dall'utente.

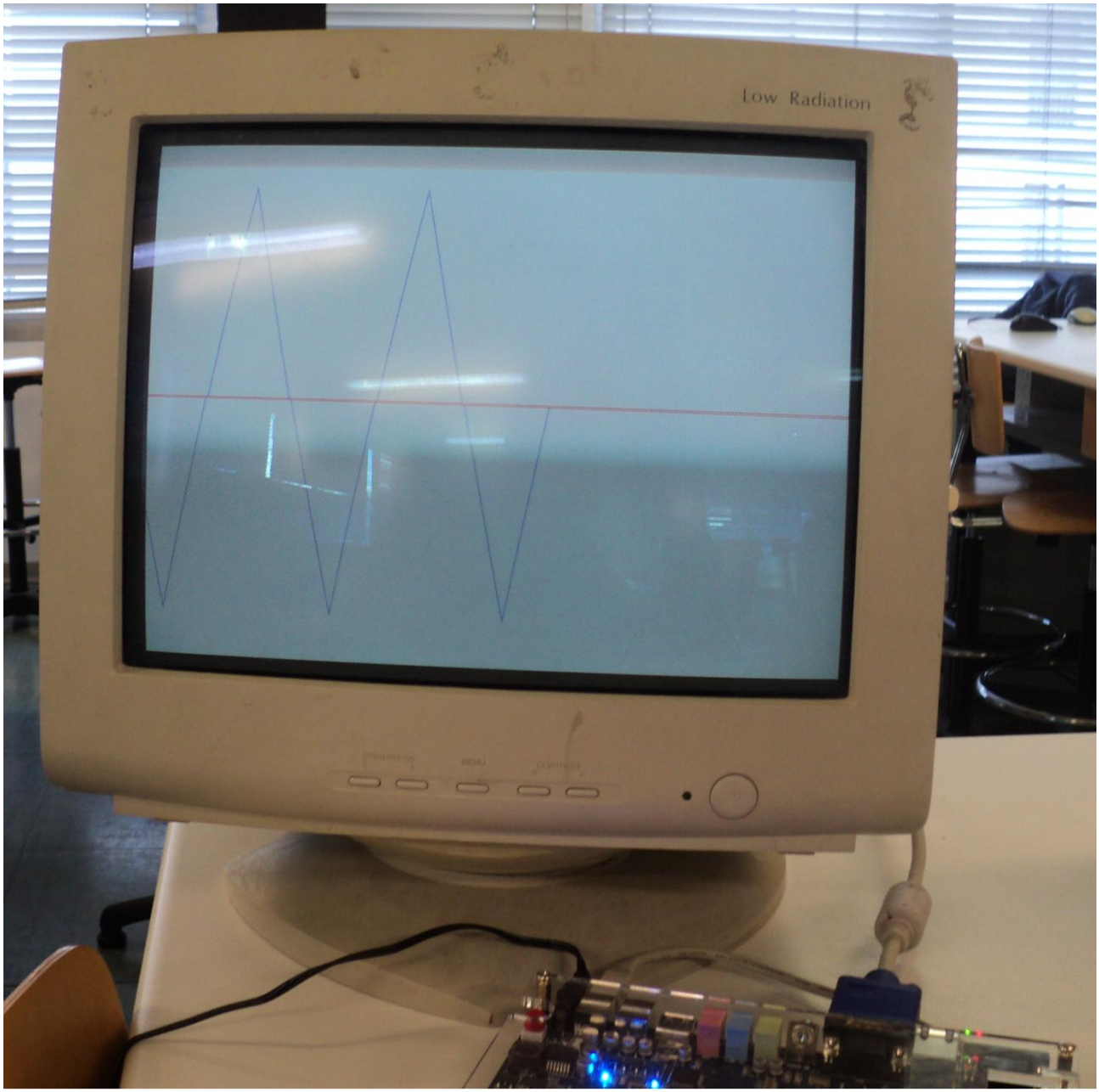
Incrementando o decrementando la fase di un qualunque dente di sega della libreria è possibile notare come questo, in procinto di traslarsi lungo l'asse verso destra o verso sinistra, tenda ad uscire dallo schermo, lasciandolo, alla fine, completamento bianco (sfasamento >= 1024 pixels). Ho tentato di rimediare a questo effetto visualizzativo spurio modificando il codice verilog, descrivente il blocco VGA_MODULATOR, dalla forma 1 a quella 2:

```
if( (count_clock65_row == ((64+20*fase) - SRAM_DQ)) || (count_clock65_row == ((64+20*fase) + SRAM_DQ))
|| (count_clock65_row == ((320+20*fase) - SRAM_DQ)) || (count_clock65_row == ((320+20*fase) + SRAM_DQ))
|| (count_clock65_row == ((576+20*fase) - SRAM_DQ)) || (count_clock65_row == ((576+20*fase) + SRAM_DQ))
|| (count_clock65_row == ((832+20*fase) - SRAM_DQ)) || (count_clock65_row == ((832+20*fase) + SRAM_DQ)) )

 begin

        REG_VGA_R <= 10'b0000000000;
        REG_VGA_G <= 10'b0000000000;
        REG_VGA_B <= 10'b1111111111; // pixel blu
 end

else

 begin
        REG_VGA_R <= 10'b1111111111;
        REG_VGA_G <= 10'b1111111111;
        REG_VGA_B <= 10'b1111111111; // altrove, tutti punti bianchi
 end
```

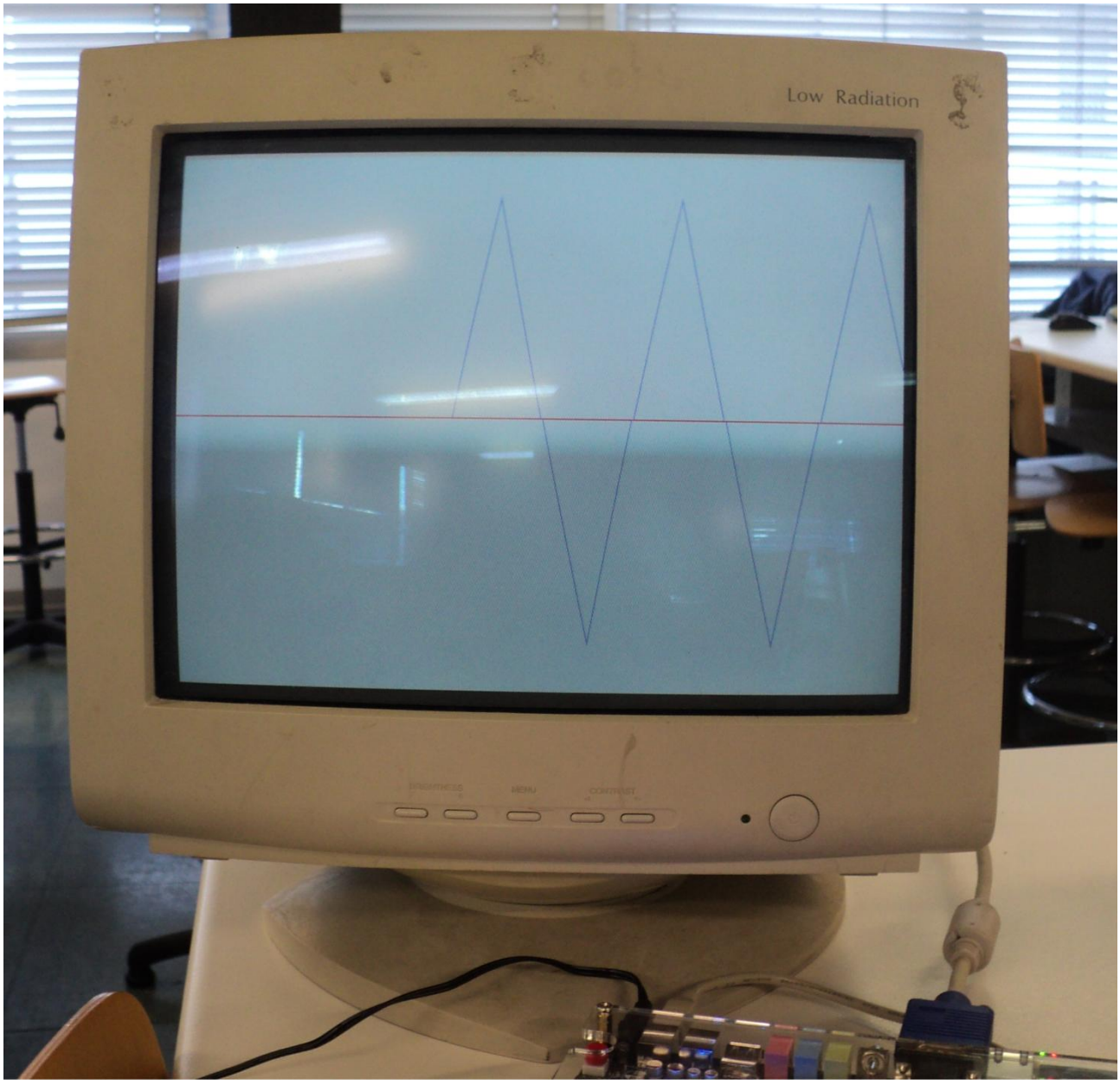


nessun prolungamento della forma d'onda nel caso di decremento della fase della stessa

1)

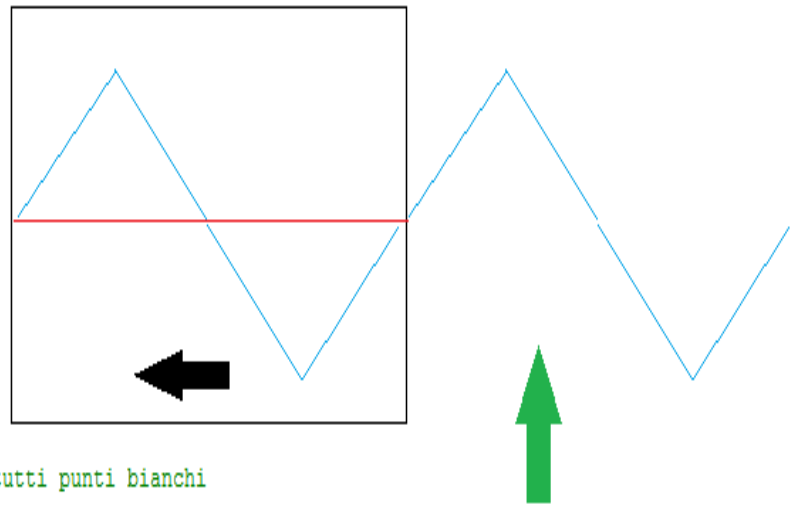


1)



1)

```
if( (count_clock65_row == ((64+20*fase) - SRAM_DQ)) || (count_clock65_row == ((64+20*fase) + SRAM_DQ))
|| (count_clock65_row == ((320+20*fase) - SRAM_DQ)) || (count_clock65_row == ((320+20*fase) + SRAM_DQ))
|| (count_clock65_row == ((576+20*fase) - SRAM_DQ)) || (count_clock65_row == ((576+20*fase) + SRAM_DQ))
|| (count_clock65_row == ((832+20*fase) - SRAM_DQ)) || (count_clock65_row == ((832+20*fase) + SRAM_DQ))
|| (count_clock65_row == 2*((64+20*fase) - SRAM_DQ)) || (count_clock65_row == 2*((64+20*fase) + SRAM_DQ)) // <=
|| (count_clock65_row == 2*((320+20*fase) - SRAM_DQ)) || (count_clock65_row == 2*((320+20*fase) + SRAM_DQ)) //
|| (count_clock65_row == 2*((576+20*fase) - SRAM_DQ)) || (count_clock65_row == 2*((576+20*fase) + SRAM_DQ)) //
|| (count_clock65_row == 2*((832+20*fase) - SRAM_DQ)) || (count_clock65_row == 2*((832+20*fase) + SRAM_DQ)) ) //

begin

        REG_VGA_R <= 10'b0000000000;
        REG_VGA_G <= 10'b0000000000;
        REG_VGA_B <= 10'b1111111111; // pixel blu
end

else

begin
        REG_VGA_R <= 10'b1111111111;
        REG_VGA_G <= 10'b1111111111;
        REG_VGA_B <= 10'b1111111111; // altrove, tutti punti bianchi
end
```



il periodo di dente di sega uscente alla sinistra dello schermo (decremento fase) verrebbe sostituito dal periodo successivo

2)

Fig. 18

Possibile soluzione al problema dell'uscita dallo schermo della forma d'onda modulata in fase

Tuttavia ho preferito non implementare questa soluzione, lasciando irrisolto il problema dell'uscita dallo schermo del dente di sega modulato in fase, poiché il timing analyzer di Quartus II, nel caso di scelta della soluzione 2, mi ha segnalato un critical warning, che poi (ho verificato) avrebbe compromesso la corretta visualizzazione del dente di sega avente periodo pari ad un quarto dello schermo (pressione del pushbutton KEY[0]). Il critical warning in questione segnalava la presenza di un percorso hardware, una volta svolta l'operazione di "place e route" da parte del fitter e compilati i relativi programming files dall'assembler, lungo il quale, una volta effettivamente programmato l'FPGA, si sarebbe verificato uno slack temporale negativo, di circa 87 ns. Si tratta del solito problema di violazione della giusta tempistica (tsu, tco, th) di campionamento, da parte dei flip-flop D, relativa ad un path registro -> logica –> registro all'interno della nostra rete, programmata su Cyclone II.

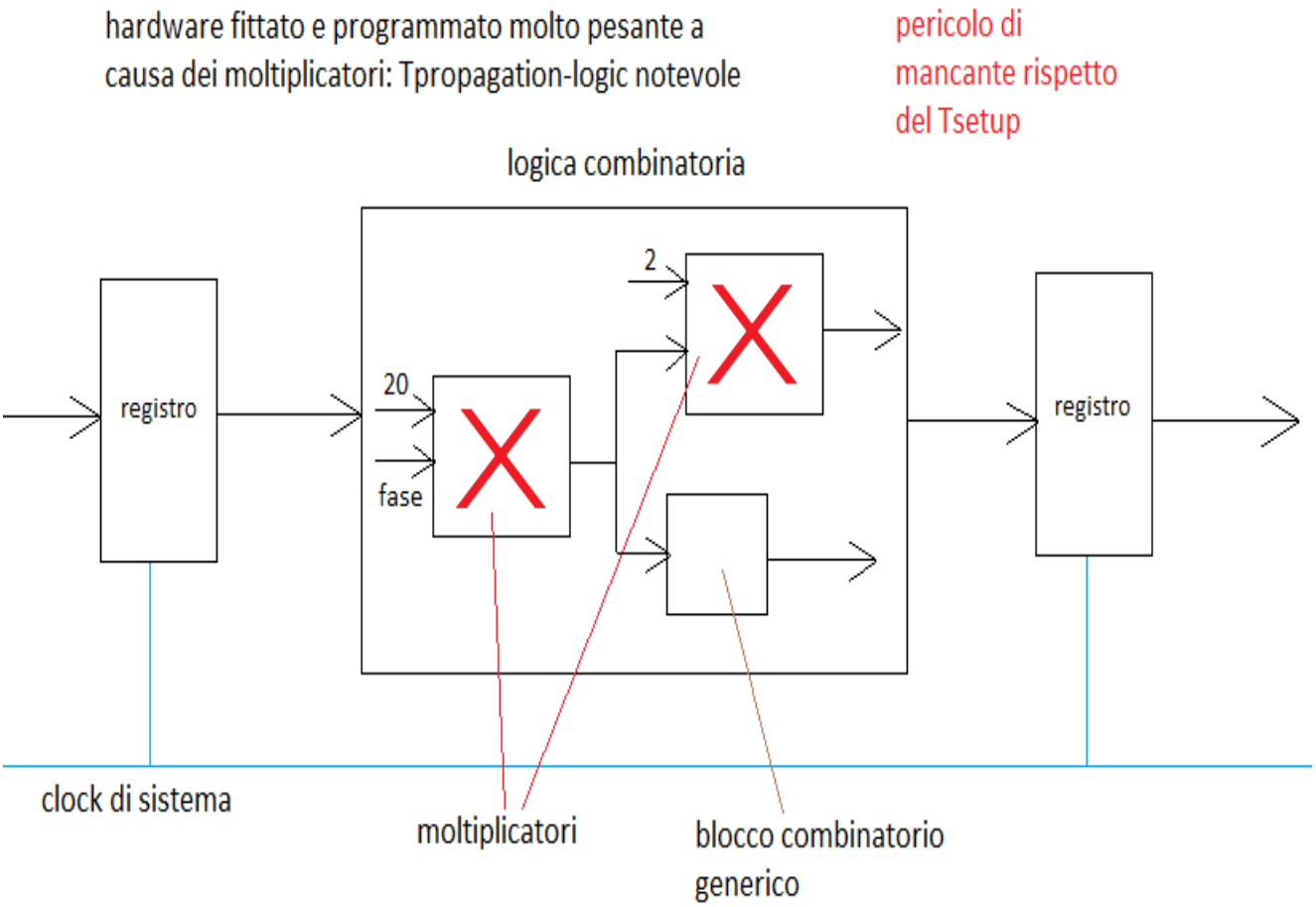


Fig. 19

Rappresentazione generica (concettuale) del path critico programmato in hardware su FPGA, nel caso di adottamento della strategia 2 di figura 18

Nel caso in cui io avessi deliberatamente trascurato la segnalazione del critical warning da parte del timing analyzer, ovvero nel caso io avessi adottato la soluzione 2 al problema della fuoriuscita da schermo del dente di sega modulato in fase, avrei permesso il fitting e successivamente la programmazione, il mapping, su FPGA, di un path che può essere rappresentato concettualmente come mostrato in figura 19. Al primo fronte positivo del clock di sistema, il primo flip-flop D fornisce in ingresso alla rete combinatoria, contenente i moltiplicatori in cascata, dopo un certo tco, un segnale stabile che alimenta la logica predisposta al calcolo dei prodotti determinanti il valore da confrontare con il contenuto del registro “count_clock65_row” (si tenga presente il codice verilog precedente). Avendo 2 moltiplicatori in cascata, il tempo T-propagation logic necessario al risultato dell’operazione per manifestarsi stabile all’uscita della logica, interposta fra i 2 registri, è molto grande, essendo il moltiplicatore una rete combinatoria molto complessa. Quindi appena arriva il secondo fronte positivo del clock, il segnale in uscita dalla logica e in ingresso al secondo flip-flop D è ben lungi dall’essere stabile da almeno un lasso di tempo pari al t-setup tipico del flip-flop D: il timing analyzer mi segnala che ci vorrebbero circa almeno altri 87 ns,

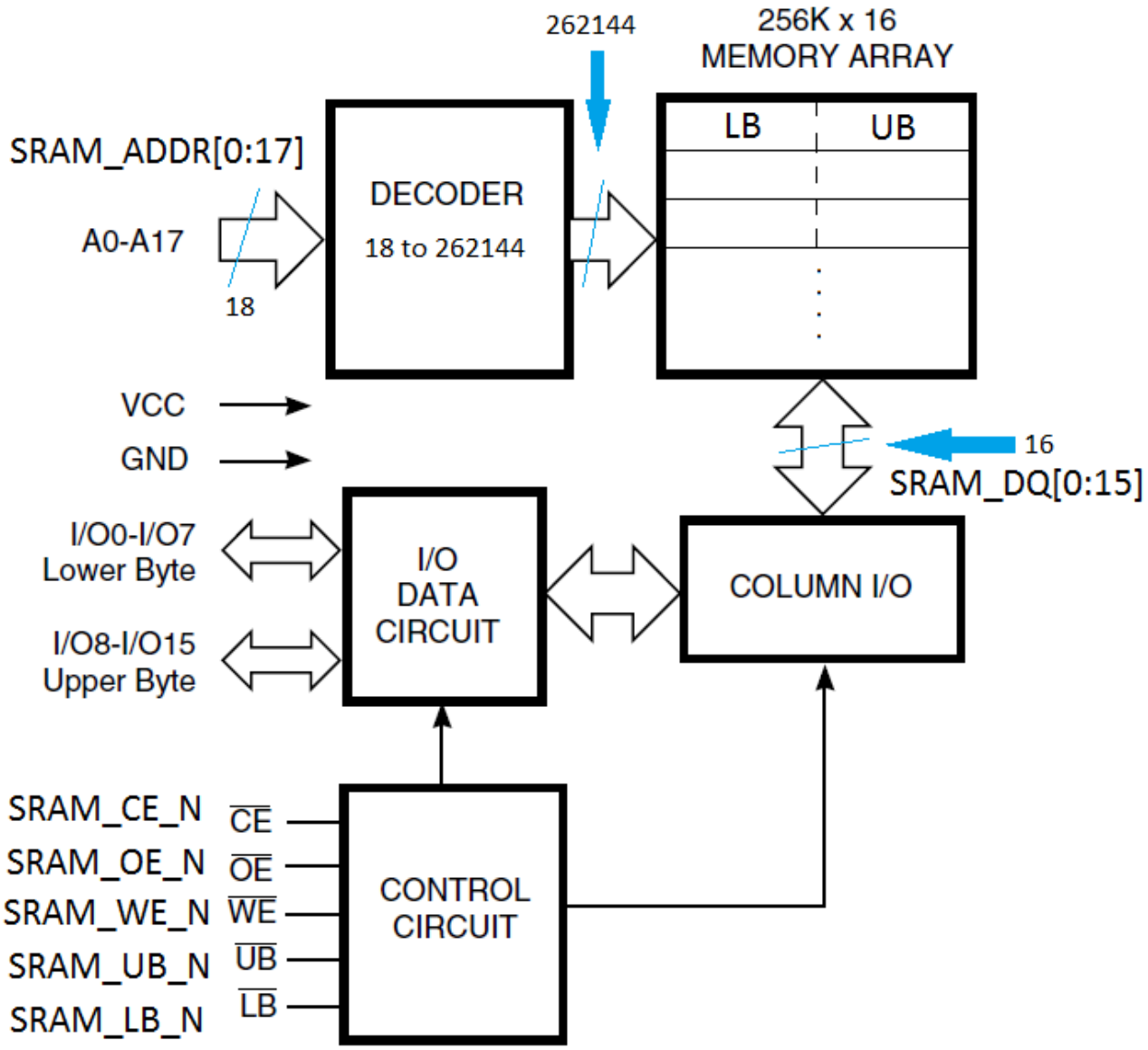
prima dell'arrivo del secondo fronte positivo del clock, affinchè il t-setup in ingresso al secondo registro possa essere rispettato: in altre parole io richiedo una frequenza di funzionamento pari a 65 MHz, ma per consentire il buon campionamento dei dati da parte dei registri lungo quel critical path, sarebbe necessaria una frequenza di clock molto più bassa (circa 18 MHz). Ho provato anche a sostituire i prodotti con delle somme equivalenti (2*x = x +x), così da spingere la catena software analizzatore/compilatore/fitter/assembler a generare, in ultima istanza, dei programming files che prevedessero la programmazione di sommatori, ossia componenti hardware meno complessi dei moltiplicatori: lo slack temporale è diminuito, rimanendo tuttavia negativo (critical warning persistente), e quindi ho preferito non implementare neppure quest'ultima soluzione.

Da notare che se durante la modulazione in fase di una certa frequenza, appartenete alla libreria del dente di sega, l'utente, mediante tastiera dei 4 pushbuttons, seleziona un'altra frequenza disponibile, la fase con la quale viene visualizzata immediatamente dopo il nuovo dente di sega è quella di default, ovvero fase = 0.

4) Gestione della memoria SRAM

4.1) Struttura della SRAM

Sulla scheda DE2 sono montate 2 memorie RAM: una SDRAM da 8 Mbytes ed una SRAM da 512 Kbytes. Ho optato per la seconda, sia per la semplicità delle temporizzazioni da imporre ai segnali di controllo per l'accesso alle sue locazioni, sia per il modesto utilizzo, in termini di numero di celle sulle quali scrivere l'informazione, a me necessario. La memoria SRAM IS61LV25616 della ISSI consta di 512 Kbytes distribuiti in locazioni ciascuna di 2 bytes, pertanto abbiamo 256K locazioni, ciascuna formata da 16 bits: un lower-byte "LB" più significativo (bit0, bit1, …, bit7) ed un upper-byte "UB" meno significativo (bit8, bit9, …, bit15). I segnali di controllo per l'accesso asincrono alla SRAM sono (tutti attivi bassi): CE (chip-enable), OE (output-enable), WE (write-enable), UB (upper_byte-enable), LB (lower_byte-enable).



IO data circuit: struttura a transceiver di larghezza pari a 16 bits, per ciascuna delle 256K locazioni

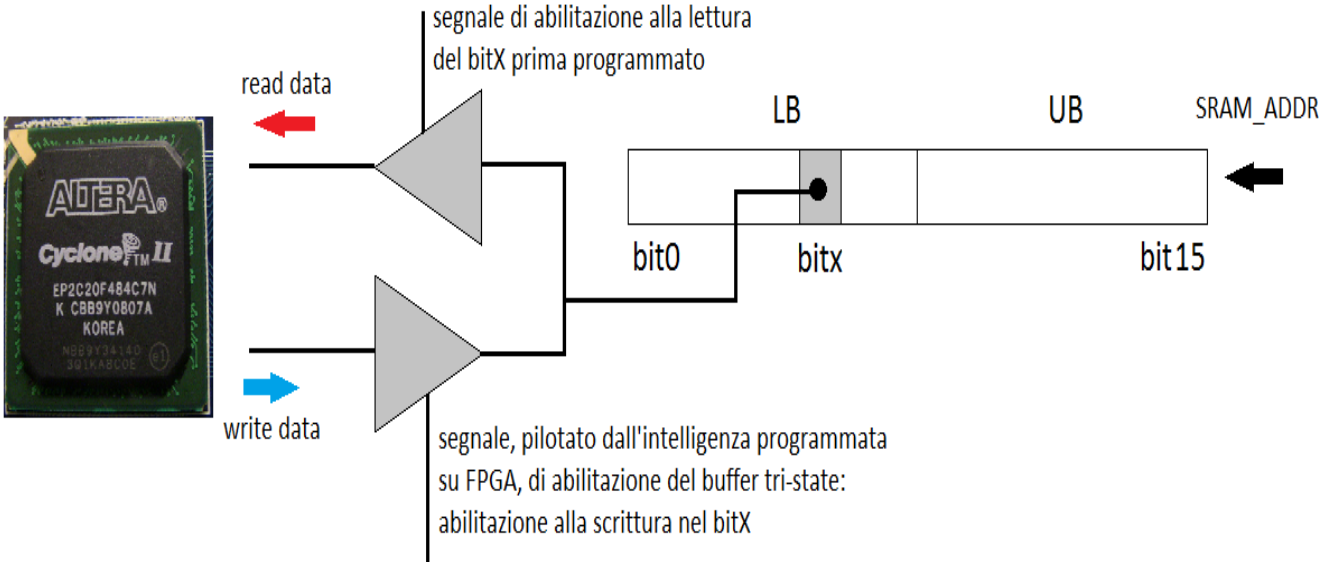


Fig. 20

Schema funzionale a blocchi rappresentativo della SRAM IS61LV25616 della ISSI

4.2) Fase di scrittura dati: rappresentazione delle forme d'onda in memoria

Supponiamo di voler visualizzare su monitor la frequenza di default della libreria, ovvero il dente di sega f0 avente periodo pari alla larghezza dello schermo. Ho deciso di sfruttare, in qualità di dato rappresentativo della forma d'onda del dente di sega, la simmetria che tale funzione presenta rispetto all'ascissa dei suoi punti di picco. Nella fattispecie abbiamo 2 punti di picco, il picco della semionda positiva e quello della semionda negativa. Si osserva facilmente come fissata una qualunque riga dello schermo contenente almeno un punto della semionda positiva, si possano individuare 2 punti (blu) appartenenti alla forma d'onda del dente di sega da visualizzare. La coordinata di uno dei 2 punti può essere espressa come l'ascissa, in termini di pixels, del pixel-cardine della semionda positiva, ossia 256 (il pixel-cardine è il 256°esimo pixel della riga), decurtata di una quantità, sempre in termini di pixels, che potremmo definire "offset" relativo a quella riga del dente di sega. La posizione dell'altro punto può essere espressa come l'ascissa, sempre in termini di pixels, del pixel-cardine della semionda positiva, ossia 256, incrementata dell'offset relativo a quella stessa riga. Per i punti blu appartenenti alla semionda negativa vale lo stesso ragionamento, dove però l'offset è riferito al pixel-cardine della semionda negativa appunto, che è il 768° pixel della riga.

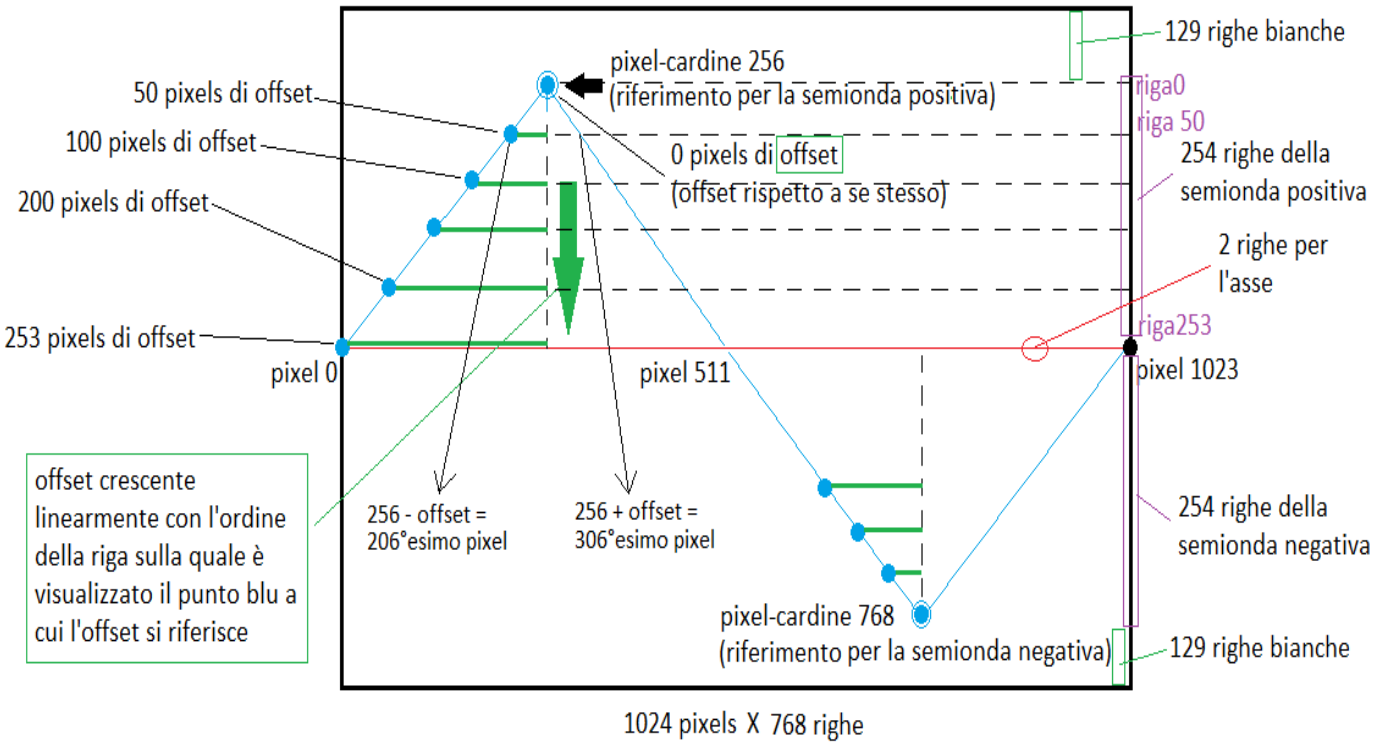


Fig. 21

Sfruttamento della simmetria del dente di sega ai fini della rappresentazione dell'informazione

Risulta chiaro che, rappresentando così l'informazione relativa alla forma d'onda del particolare dente di sega selezionato (ma questo modo di ragionare si estende a tutte le frequenze della nostra libreria), abbiamo bisogno di accedere alle prime 254 celle di memoria SRAM e in ciascuna di queste memorizzare l'offset costituente la distanza dei 2 punti blu rispetto al pixel-cardine. Non c'è bisogno di scrivere in 254 x 2 = 508 celle SRAM, ossia in 254 locazioni per memorizzare i 254 offsets relativi alla semionda positiva e nelle rimanenti 254 locazioni per memorizzare i 254 offsets della semionda negativa, dal momento che è sufficiente riferire i primi 254 offsets al pixel-cardine 768 per rappresentare l'informazione relativa alla semionda negativa del dente di sega.

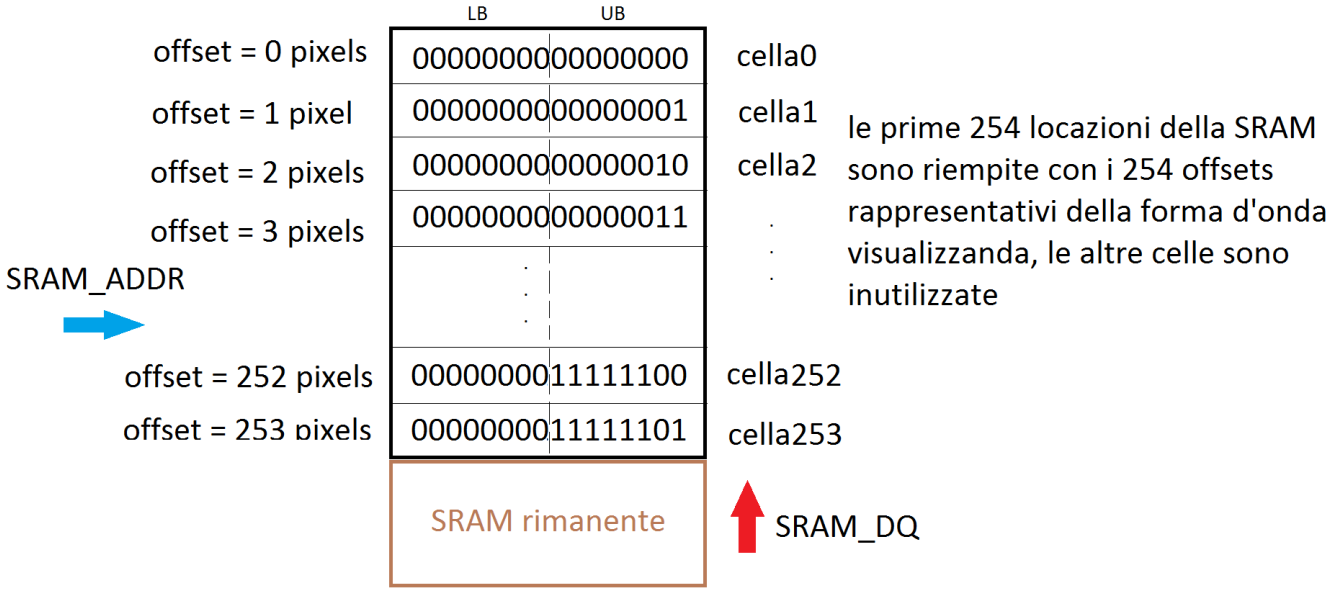


Fig. 22

Rappresentazione concettuale della collocazione degli offsets nella SRAM

La SRAM è un componente hardware presente sulla scheda DE2, ma esterno all'elemento programmabile FPGA, pertanto è stato necessario implementare, mediante una descrizione funzionale in codice verilog, un blocco, che ho chiamato memoryMANAGER, il quale gestisce la scrittura in SRAM degli offsets rappresentativi dell'informazione relativa a ciascuna frequenza di libreria, nonché il successivo accesso ai fini della lettura dei dati precedentemenți scritti. Il blocco memoryMANAGER, una volta programmato sul Cyclone II, costituirà la parte di logica programmata che si prenderà cura, ogni volta che l'utente selezionerà una particolare frequenza

da visualizzare, di scrivere in SRAM, nelle prime N locazioni necessarie per quel particolare dente di sega da visualizzare (N = 254 per la frequenza di default, N = 63 per la frequenza selezionata dalla pressione del pushbutton KEY[0] ecc…), gli N offsets che saranno successivamente sfruttati dal blocco VGA_MODULATOR per la corretta visualizzazione del dente di sega sul monitor. Riporto qui di seguito la parte del top-level file "DENTE_SEGA_system.bdf" rappresentante il blocco di gestione della SRAM, durante la fase di scrittura degli offsets, nel caso in cui l'utente non prema alcun pushbutton oppure nel caso in cui ne tenga premuti almeno 2, fra i 4 disponibili.

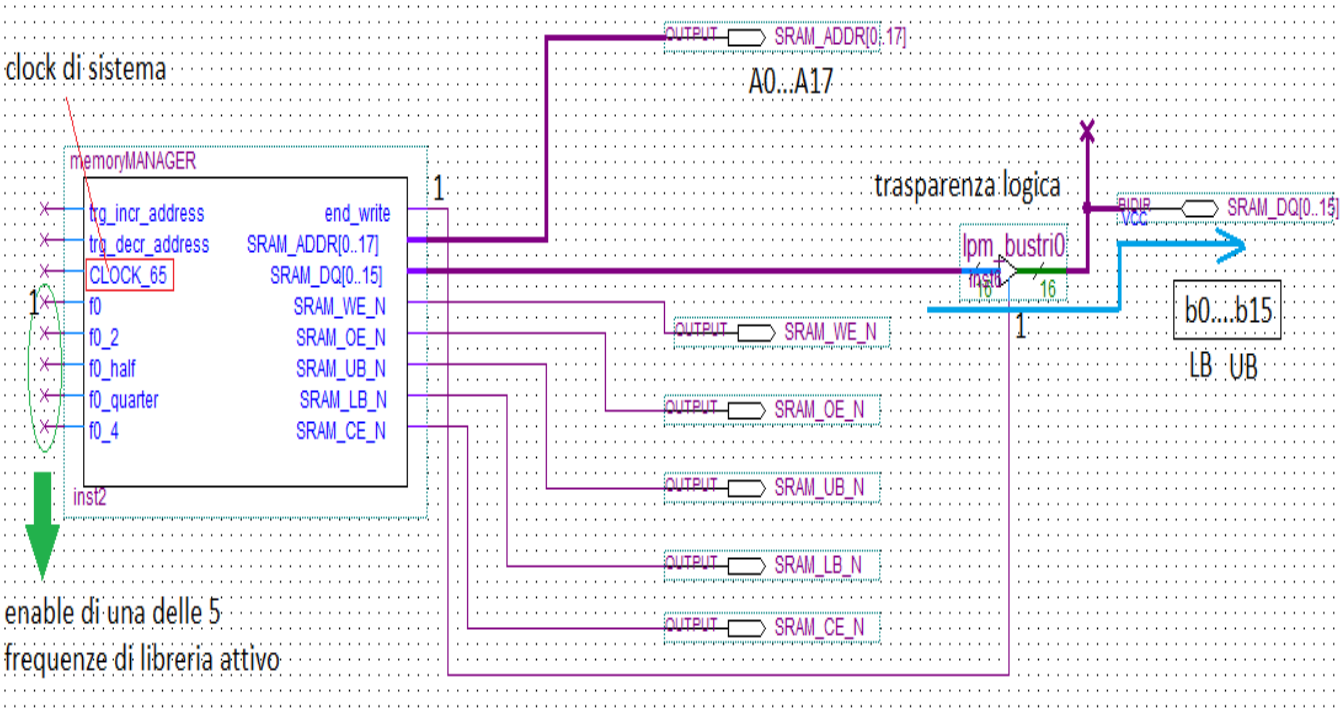


Fig. 23

Parte di schematic del top-level file "DENTE_SEGA_system.bdf" rappresentante il blocco memoryMANAGER di gestione della SRAM (frequenza f0 di default abilitata, fase di scrittura degli offsets)

4.3) Fase di scrittura dati: temporizzazioni dei segnali di controllo della SRAM

Il costruttore (ISSI) della SRAM, per mezzo del relativo datasheet, ci propone 4 modalità di temporizzazione dei segnali di controllo della SRAM, affinchè l'elemento FPGA programmato dall'utente possa generare dei segnali di controllo che, applicati ai piedini della memoria montata sulla scheda, garantiscano un corretto e ben funzionante (affidabile) accesso in scrittura della SRAM. Ho scelto il quarto tipo di write-cycle, del quale riporto il diagramma temporale tratto proprio dal datasheet della SRAM IS61LV25616.

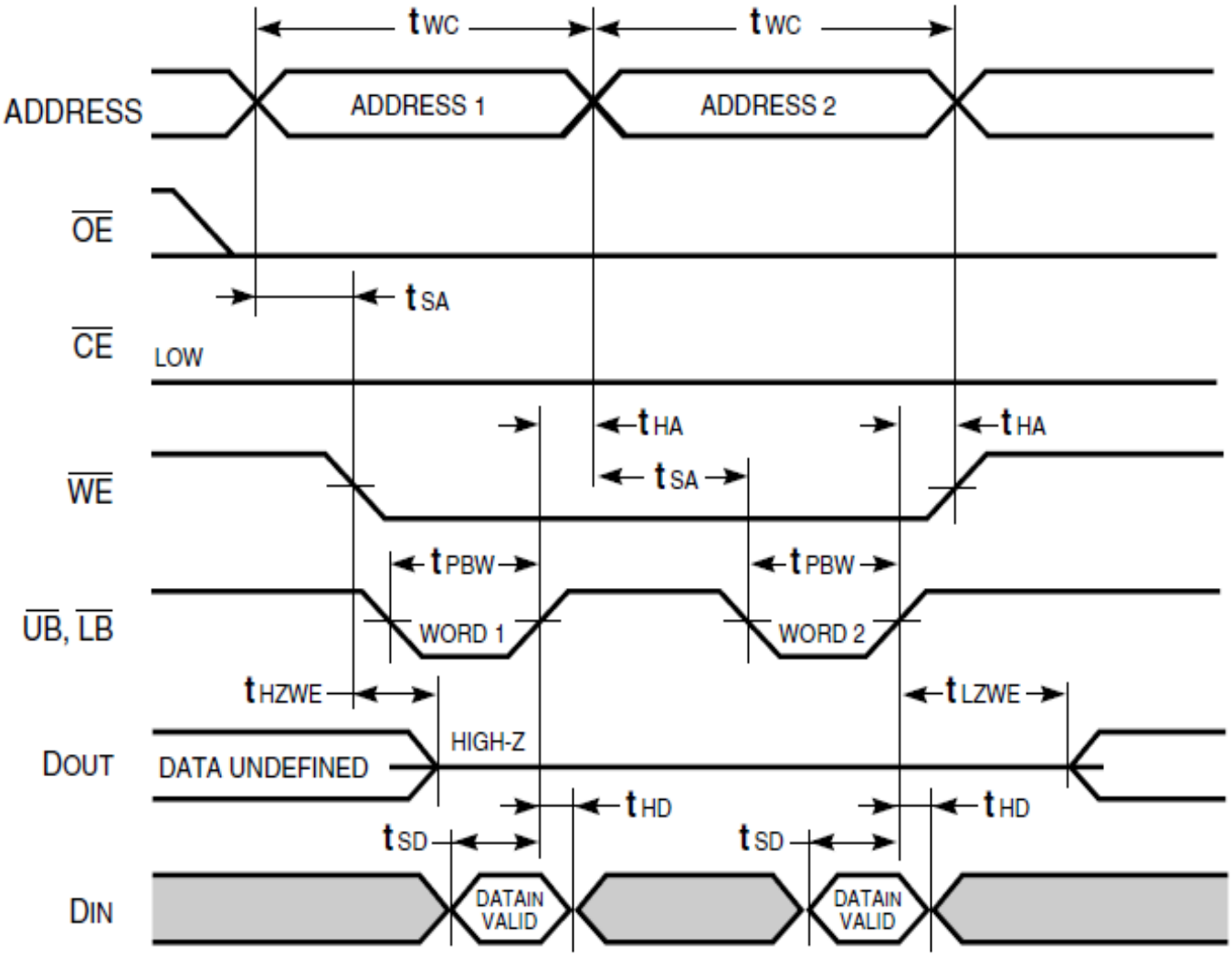


Fig. 24

Temporizzazioni dei segnali di controllo della SRAM, generabili dall'elemento programmabile Cyclone II, proposte dal costruttore della memoria (uno dei 4 write-cycles proposti nel datasheet) durante la fase di scrittura dati

Il write-cycle N°4, definito anche "LB/UB controlled" dallo stesso costruttore della SRAM IS61LV25616 , è caratterizzato dal fatto che i segnali CE e OE possono essere tenuti sempre attivi durante la scrittura degli offsets, ovvero bassi, dunque sono segnali "statici" che non devono essere commutati frequentemente, e ciò non fa che semplificare la scrittura del codice verilog che implementa funzionalmente la generazione dei segnali di controllo della SRAM. Anche WE può essere tenuto basso, cioè attivo, per molti cicli di indirizzamento e scrittura, mentre gli unici segnali da modulare con un certa frequenza sono UB e LB: questi devono essere tenuti alti durante la commutazione (l'updating, l'aggiornamento) dei segnali di ADDRESS e di DATA-IN, poi devono essere abbassati, cioè attivati, allo scadere del tempo di setup di ADDRESS e di DATA-IN, per poi essere rialzati, cioè ridisattivati, allo scadere del tempo necessario alla scrittura degli stessi ADDRESS e di DATA-IN. Tuttavia il datasheet della SRAM non specifica il numero, nemmeno come

ordine di grandezza, di cicli di indirizzamento e scrittura durante i quali è possibile tenere continuativamente basso WE, motivo per cui ho preferito adottare una temporizzazione leggermente diversa dal write-cycle N°4 suggerito dal costruttore: l'unica differenza infatti sta nella modulazione di WE, che accompagna quella di LB e UB, come si può vedere nella figura seguente.

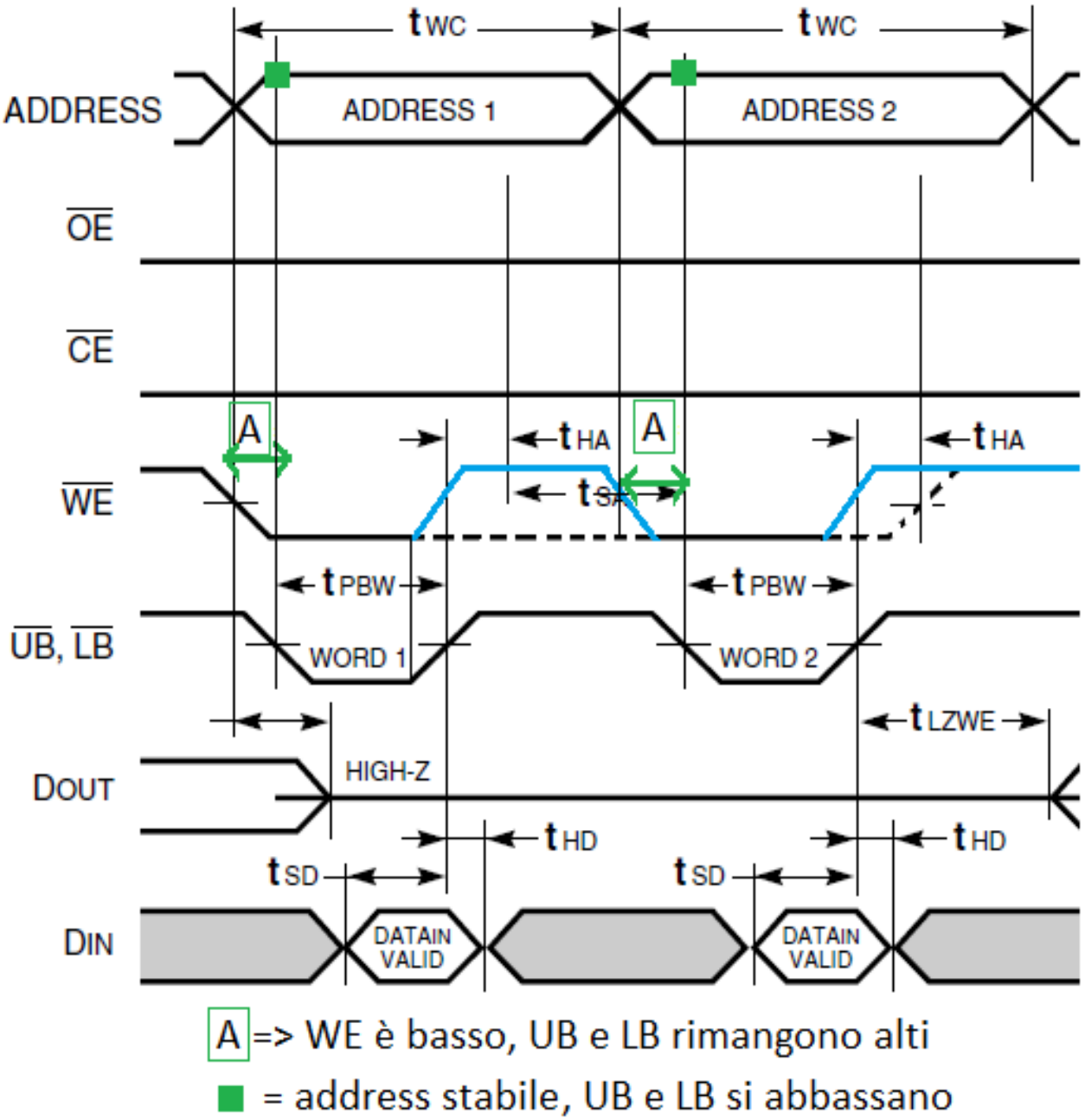


Fig. 25

Temporizzazioni in fase di scrittura dei segnali di controllo della SRAM da me adottate, leggermente modificate rispetto a quelle consigliate dal costruttore e riportate nel datasheet sotto il nome di "write-cycle N°4"

Riporto qui di seguito la porzione di un file.vwf contenente gli istanti iniziali di una simulazione timing. Notiamo che allo start-up del sistema, diciamo subito dopo che abbiamo lanciato la programmazione via usb-blaster del Cyclone II (quindi appena dopo che i programming files generati dall'assembler di Quartus II abbiano "impressionato" l'hardware programmabile del Cyclone II con la "mappa dei fusibili", composta dall'azione di "place e route" da parte del fitter di Quartus II), abbiamo che l'FPGA, o meglio quella sua parte che implementa il blocco memoryMANAGER, punta l'ultima locazione della SRAM 256K x 16, ovvero SRAM_ADDR = 18'b111111111111111111, dove SRAM_ADDR è il registro di 18 bits che contiene l'indirizzo della locazione di memoria a cui si desidera accedere, mentre il registro di 16 bits contenente il valore DQ da scrivere in SRAM, all'indirizzo puntato, contiene il massimo valore esprimibile su 16 bits, ossia SRAM_DQ = 16'b1111111111111111. Si noti altresì l'overflow sia di SRAM_ADDR che di SRAM_DQ, propedeutico alla scrittura del valore 0, cioè dell'offset nullo (corrispondente ai 2 punti di picco della frequenza di default), presso la prima (A0…A17 = 0…0) delle 254 locazioni necessarie alla rappresentazione dei 254 offsets caratterizzanti la frequenza di default.

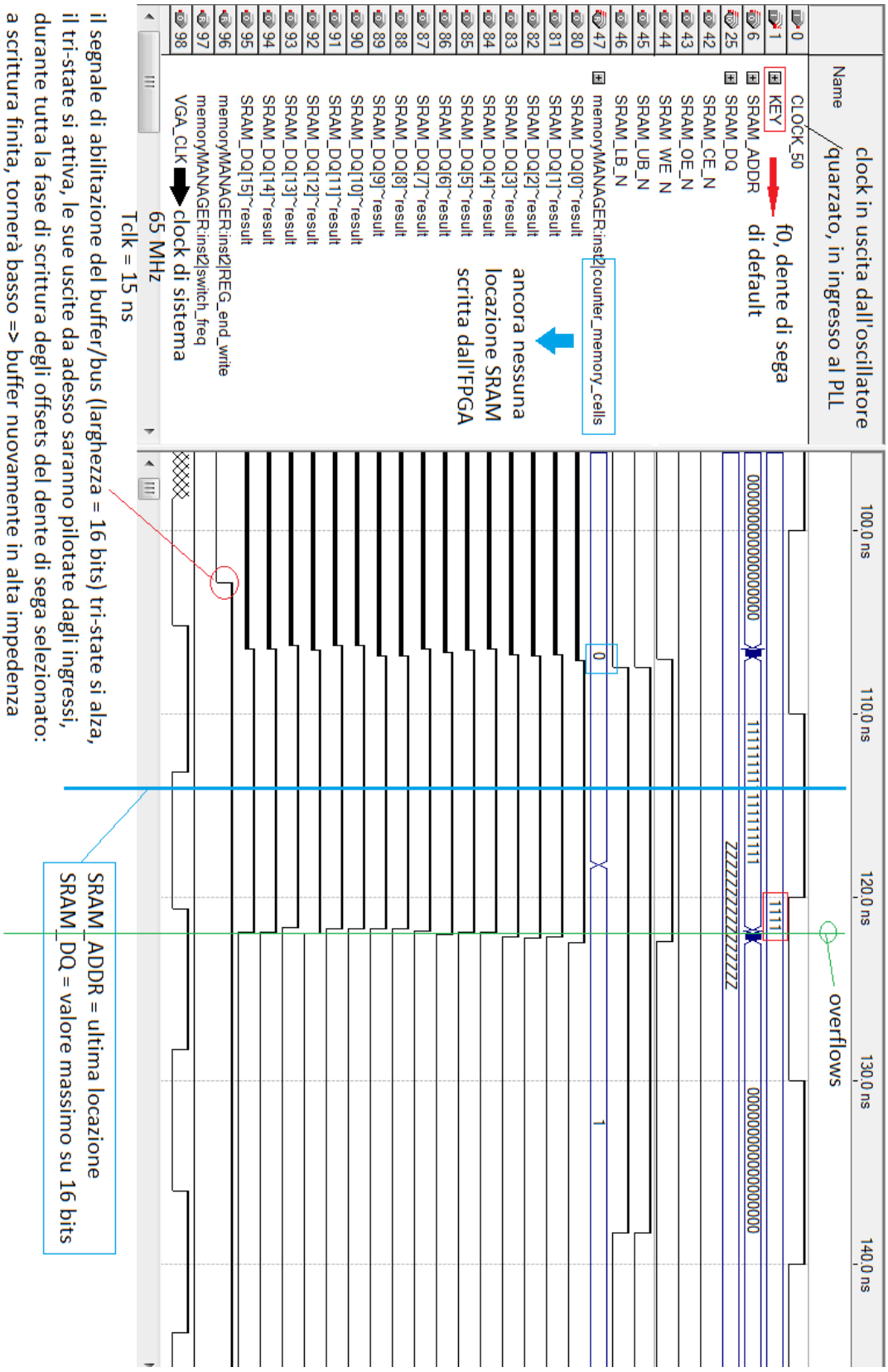


Fig. 26

Simulazione timing raffigurante il funzionamento del Cyclone II negli istanti appena successivi alla sua programmazione via canale usb-blaster

Riporto alcuni frammenti di codice verilog implementanti la funzionalità del memoryMANAGER allo start-up del sistema e durante la fase di scrittura degli offsets, dettagliatamente spiegati.

```
always @(posedge CLOCK_65)
begin
if( (f0 == 1) && (f0_2 == 0) && (f0_half == 0) && (f0_quarter == 0) && (f0_4 == 0) )
// scrittura e lettura del dente di sega di default

begin // f0

if(start == 0)
 begin
  REG_end_write <= 1'b1;
  REG_SRAM_WE_N <= 1'b1;
  REG_SRAM_UB_N <= 1'b1;
  REG_SRAM_LB_N <= 1'b1;
  REG_SRAM_OE_N <= 1'b0;
  REG_SRAM_CE_N <= 1'b0;
  REG_SRAM_ADDR <= 18'b111111111111111111;
  REG_SRAM_DQ <= 16'b1111111111111111;
  //---------------------registri che assegnano valore alle uscite del blocco
  counter_memory_cells <= 0;
  control <= 0;
  index <= 0;
  index_PosNegWF <= 0;
  count_read <= 0;
  switch_freq <= 0;
  rr <= 0;
  //--------------------registri interni di servizio
  start <= 1;
 end

else // if(start == 1)

 begin // start

 start <= 1;

if(switch_freq == 1)
 begin
  REG_end_write <= 1;
  switch_freq <= 0;
  REG_SRAM_ADDR <= 18'b111111111111111111; // sfrutterò l'overflow
  REG_SRAM_DQ <= 16'b1111111111111111; // al prossimo incremento avrò: puntamento prima cella e primo dato = 0
  REG_SRAM_UB_N <= 1; REG_SRAM_LB_N <= 1; REG_SRAM_WE_N <= 1; REG_SRAM_OE_N <= 0; REG_SRAM_CE_N <= 0;
  counter_memory_cells <= 0; // scorrimento celle: 1 -> 254
  control <= 0;
  index <= 0;
  index_PosNegWF <= 0; // semionda positiva/negativa
  count_read <= 0; // conteggio celle: 1,2,...,254 / 254,253,...,1
  rr <= 0;
 end




else // if (switch_freq == 0)
 begin /// switch_freq

  switch_freq <= 0;

 if(REG_end_write == 1)
  begin
    index <= 0;
    index_PosNegWF <= 0;
    count_read <= 0;
    rr <= 0;
```

l'instradatore di segnali di enable (abilitatoreFreq) ha settato il segnale di abilitazione della frequenza di default

primo colpo di clock di funzionamento dell'FPGA programmato, siamo al primo ingresso dentro l'if che gestisce la scrittura in SRAM e lettura da SRAM su VGA del dente di sega di default

inizializzazione di tutti i registri

per mezzo di assegnamenti continui "assign"

contatore del numero di locazioni SRAM in cui il Cyclone II ha scritto fino a quel momento, per adesso la scrittura non è ancora iniziata, nessuna cella SRAM ancora scritta

al secondo colpo di clock si entra qui

questo bit è basso quando l'FPGA si occupa di scrivere e leggere in/da SRAM gli offsets relativi al dente di sega di default; assume invece valore alto quando l'FPGA si occupa di scrivere e leggere in/da SRAM gli offsets relativi a tutte le altre frequenze del dente di sega

codice che gestisce solo il risettaggio, il ripristino, dei valori dei registri di memoryMANAGER all'atto del rilascio di uno dei 4 pushbuttons, cioè all'atto del ritorno alla frequenza di default da una qualunque delle altre frequenze di dente di sega: end_write da 0 (stavamo visualizzando una certa frequenza del dente di sega) ripassa a 1 (memoryMANAGER deve scrivere i nuovi offsets, cioè i 254 offsets del dente di default), switch_freq passa da 1 a 0, SRAM_ADDR e SRAM_DQ, che avevano valori "casuali" dipendenti dall'istante in cui abbiamo rilasciato il pushbutton, cioè dall'istante in cui abbiamo interrotto la visualizzazione su VGA, cioè la lettura da SRAM, devono essere ripristinati per previsto prossimo overflow

buffer/bus tri-state trasparente, il Cyclone II deve accedere in scrittura alle celle della SRAM

Fig. 27

Parte del codice del blocco memoryMANAGER implementante la funzionalità della gestione della SRAM durante gli istanti di start-up dell'FPGA programmato

```
if (counter_memory_cells <= 254)
 begin //254
  REG_end_write <= 1; // mantenimento di end_write ad 1
  REG_SRAM_OE_N <= 1'b0;
  REG_SRAM_CE_N <= 1'b0;

if (control == 0 && SRAM_UB_N == 1) // Tclock65_primo-periodo
 begin // preparazione alla memorizzazione nelle prime 254 celle SRAM

  REG_SRAM_WE_N <= 0;     //1->0 AMBIGUITà DEL DATASHEET SRAM CIRCA IL NUMERO
                          // DI WRITE-CYCLES DURANTE I QUALI WE_N PUò ESSERE TENUTO BASSO, ATTIVO
  REG_SRAM_UB_N <= 1'b1; // scrittura ancora disabilitata
  REG_SRAM_LB_N <= 1'b1;

  control <= 1; //0->1 // control <= control + 1; => sommatore = tanto hardware!!!
  REG_SRAM_ADDR <= REG_SRAM_ADDR + 1; // prima volta: puntamento della prima cella SRAM: address = 000..00
  REG_SRAM_DQ <= REG_SRAM_DQ + 1; // prima volta: datoIO = 0..0

  // circa 15ns a disposizione (T_setup molto lungo) per i segnali SRAM_ADDR e SRAM_DQ per regimare

 end

else if(control == 1 && REG_SRAM_UB_N == 1) // Tclock65_secondo-periodo : WE_N = 0 = scrittura stabilmente attiva,
 begin // segnali di address e dataIN ormai stabili...posso abbassare LB e UB, cioè attivare la scrittura

  REG_SRAM_WE_N <= 0; // mantenimento di WE_N a 0
  REG_SRAM_UB_N <= 1'b0; // memorizzazione, in cella pre-acceduta, di dato pre-formato
  REG_SRAM_LB_N <= 1'b0;
  control <= 0; //1->0 // control <= control + 1; => RISPARMIAMO HARDWARE, qui che possiamo!!
  counter_memory_cells <= counter_memory_cells;
  REG_SRAM_ADDR <= REG_SRAM_ADDR;
  REG_SRAM_DQ <= REG_SRAM_DQ; // mantenimento di dato e address

 end

else if(control == 0 && REG_SRAM_UB_N == 0) // Tclock65_terzo-periodo : alzo WE_N, UB_N e LB_N => disabilitazione della scrittura
 begin

  REG_SRAM_WE_N <= 1'b1; //0->1

  REG_SRAM_UB_N <= 1'b1;
  REG_SRAM_LB_N <= 1'b1;

  control <= 0; // control rimane = 0
  counter_memory_cells <= counter_memory_cells;
  REG_SRAM_ADDR <= REG_SRAM_ADDR;
  REG_SRAM_DQ <= REG_SRAM_DQ; // mantenimento di dato e address

 end

else // if(control == 1 && SRAM_UB_N == 0)

 begin
  REG_SRAM_WE_N <= 1'b1;
  REG_SRAM_UB_N <= 1'b1;
  REG_SRAM_LB_N <= 1'b1;
  counter_memory_cells <= counter_memory_cells;
 end

end //254

 else // if(counter_memory_cells >= 255) // puntamento alla 255esima cella SRAM (il puntatore ha strabordato),
  begin                                 // la scrittura deve cessare

   REG_SRAM_UB_N <= 1'b1;
   REG_SRAM_LB_N <= 1'b1;
   REG_SRAM_WE_N <= 1'b1;
   REG_SRAM_OE_N <= 1'b1;
   REG_SRAM_CE_N <= 1'b1;

  end
end // fine scrittura dei 254 offsets nelle 254 celle
```

buffer/bus tri-state in trasparenza in->out per tutta la durata della scrittura degli offsets

CE e OE abilitati per tutta la durata della scrittura

al fronte in salita del primo ciclo di clock (primo dei 3), abbasso WE, commuto (aggiorno) address e dato, LB e UB ancora alti, disattivati, cioè ancora non scrivo, per garantire un congruo Tsetup dei segnali address e dato

vi si entra al fronte positivo del secondo ciclo di clock, alla fine del Tsetup di address e dato, UB e LB si abbassano, inizia l'azione di scrittura; UB e LB mantenuti bassi fino al prossimo positive edge del clock, così da garantire un Twrite per address e dato molto lungo (15 ns)

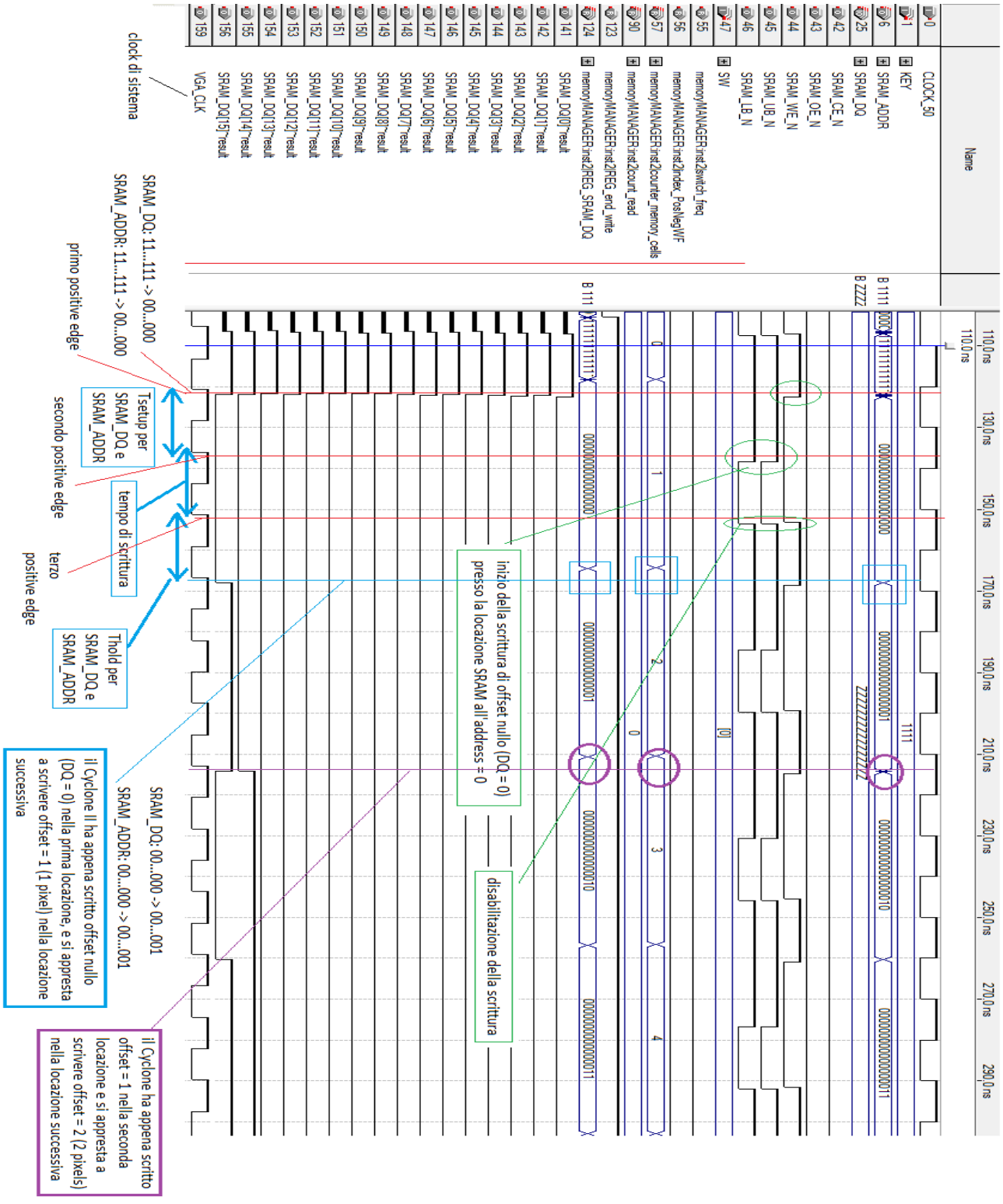
vi entro al fronte in salita del terzo e ultimo ciclo di clock, WE, LB e UB si alzano, scrittura disabilitata fino a successivo positive edge del clock (ritorno al primo dei 3 rami if)

caso che non capiterà mai, tuttavia il compilatore di Quartus II non lo sa questo, quindi se non lo mettessi sarebbe portato a credere che ci sia un caso che effettivamente può capitare ma il cui trattamento (valore da assegnare ai registri in quel caso) non è stato coperto nel codice, dunque il compilatore inferisce dei latches aggiuntivi per conservare i valori memorizzati al clock positive edge precedente

Fig. 28

Parte del codice del blocco memoryMANAGER implementante la funzionalità di generazione dei segnali di controllo e di accesso alla SRAM, durante i 3 cicli di clock di sistema coinvolti nella scrittura di un certo dato DQ (un particolare offset) in una certa locazione SRAM.

Quest'ultimo frammento di codice genera i segnali di cui riporto, qui sotto, una simulazione timing.



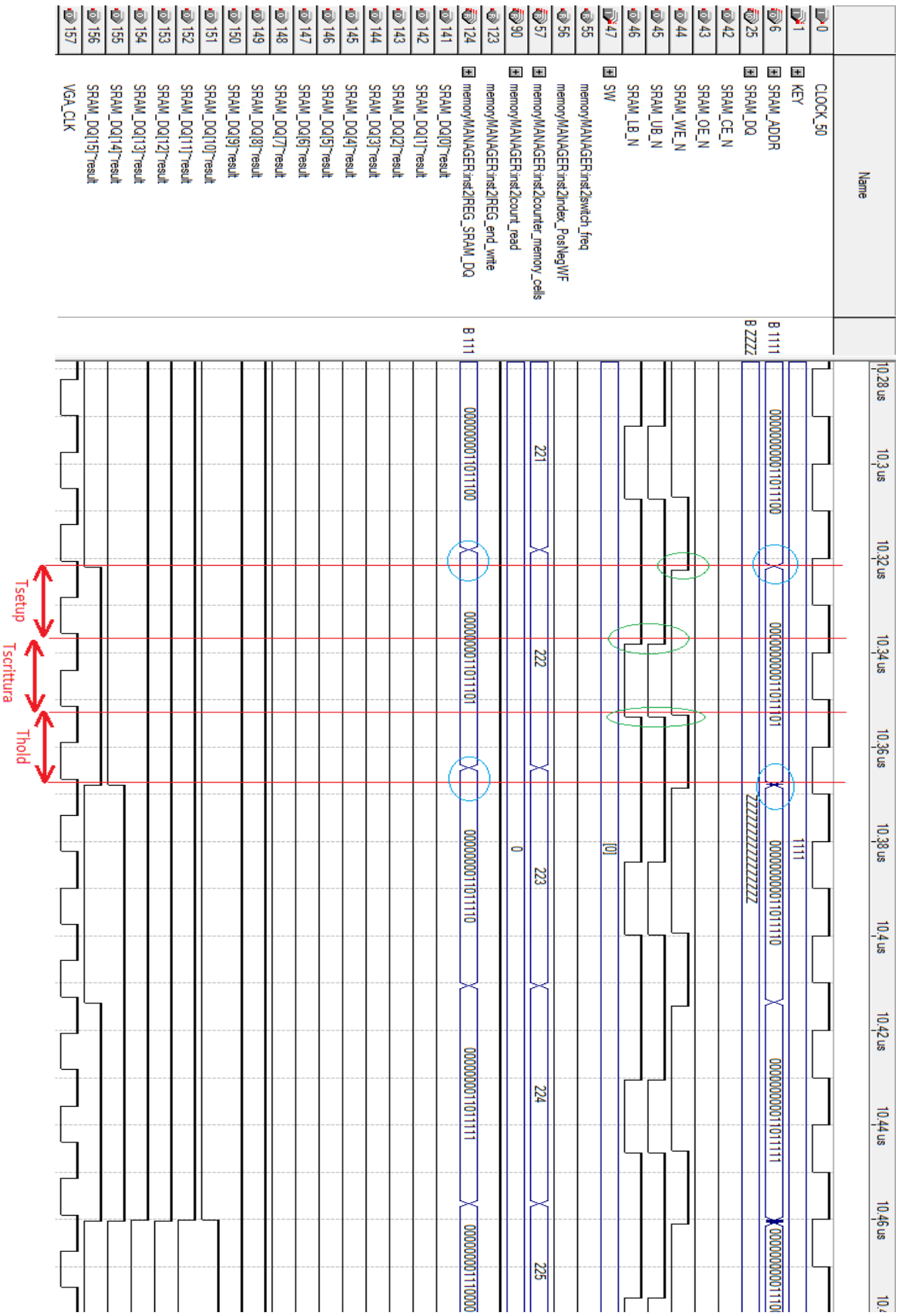


Fig. 29

Simulazione timing raffigurante i segnali di controllo generati dal Cyclone II durante l'accesso in scrittura alle locazioni SRAM

Qui di seguito riporto gli istanti della simulazione timing nei quali l'FPGA è in procinto di concludere la scrittura, in SRAM, degli offsets del dente di sega f0 di default. Dopo aver terminato la scrittura dell'ultimo offset, pari a 253 (253 pixels), si ha un periodo di clock di sistema (15 ns) durante il quale l'elemento programmato deve mutare la propria funzionalità da fase di scrittura a fase di lettura, pertanto end_write si abbassa, portando il buffer/bus tri-state (largo 16 bits) in alta impedenza, puntando nuovamente l'ultima locazione della SRAM e disabilitando, cioè alzando, tutti i segnali di controllo della stessa, come CE, OE, WE, LB e UB.

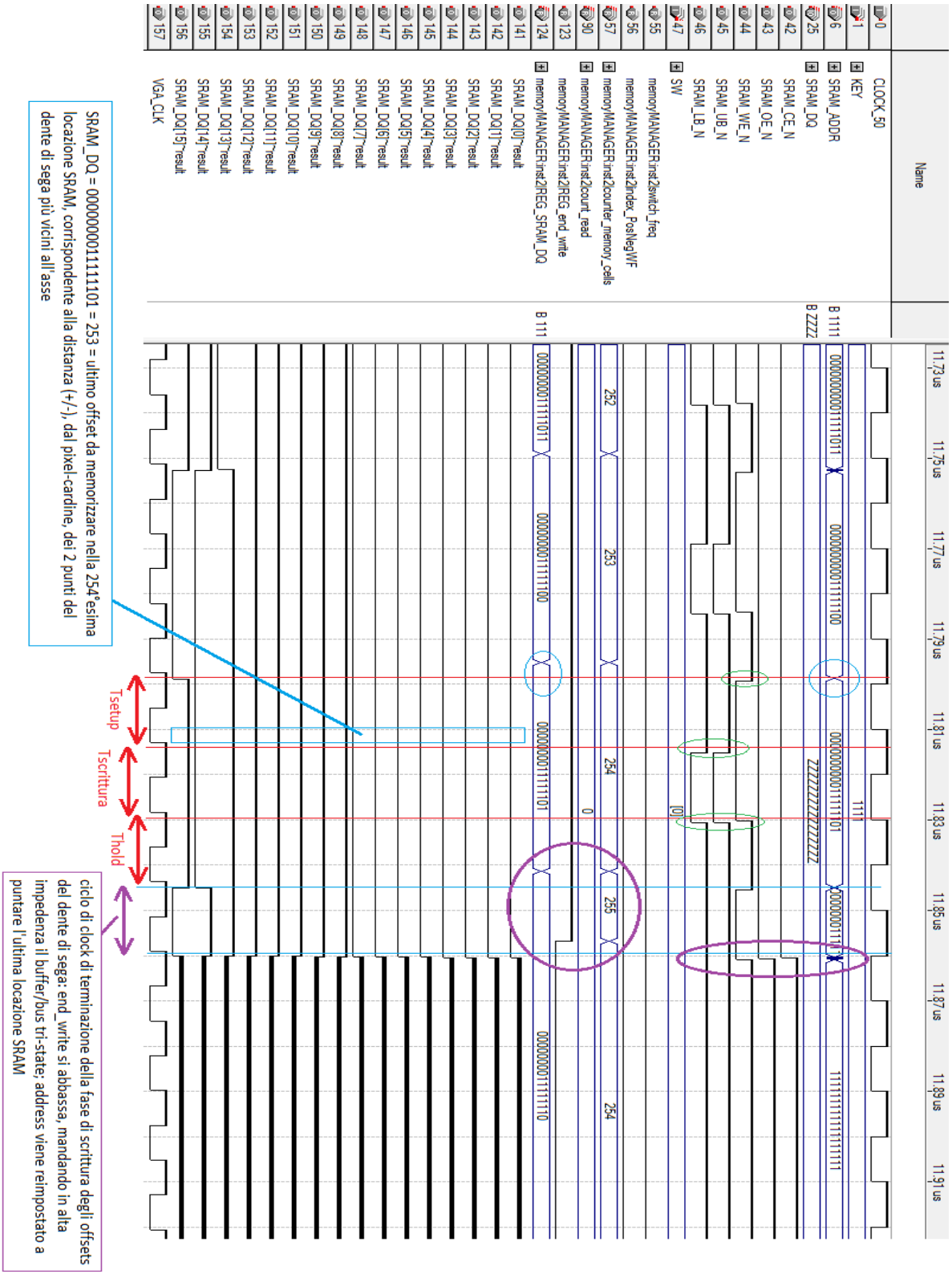


Fig. 30

Simulazione timing raffigurante i segnali di controllo generati dal Cyclone II durante gli ultimi cicli di accesso in scrittura alla SRAM

Gli ultimi nanosecondi che precedono l'impostazione in alta impedenza del buffer/bus tri-state, e la conseguente conclusione della scrittura in SRAM da parte dell'FPGA, vedono un comportamento dei segnali di controllo implementato dal seguente frammento di codice, sempre del blocco memoryMANAGER.

```
...
 begin

  if(counter_memory_cells == 255)
   begin

    REG_end_write <= 0;
    counter_memory_cells <= counter_memory_cells - 1;
    REG_SRAM_ADDR <= 18'b111111111111111111;
    REG_SRAM_DQ <= REG_SRAM_DQ;
    REG_SRAM_WE_N <= 1'b1;
    REG_SRAM_UB_N <= 1'b1;
    REG_SRAM_LB_N <= 1'b1;
    REG_SRAM_OE_N <= 1'b1;
    REG_SRAM_CE_N <= 1'b1;
    control <= 0;

   end // sfrutterò l'overflow
  else
    counter_memory_cells <= counter_memory_cells;

 end
...
```

per riportare il contatore di locazioni SRAM già scritte al valore che deve avere alla fine della memorizzazione dei dati relativi alla frequenza di default, ossia 254

Fig. 31

Frammento di codice del blocco memoryMANAGER implementante la conclusione della funzionalità di scrittura in SRAM da parte del Cyclone II

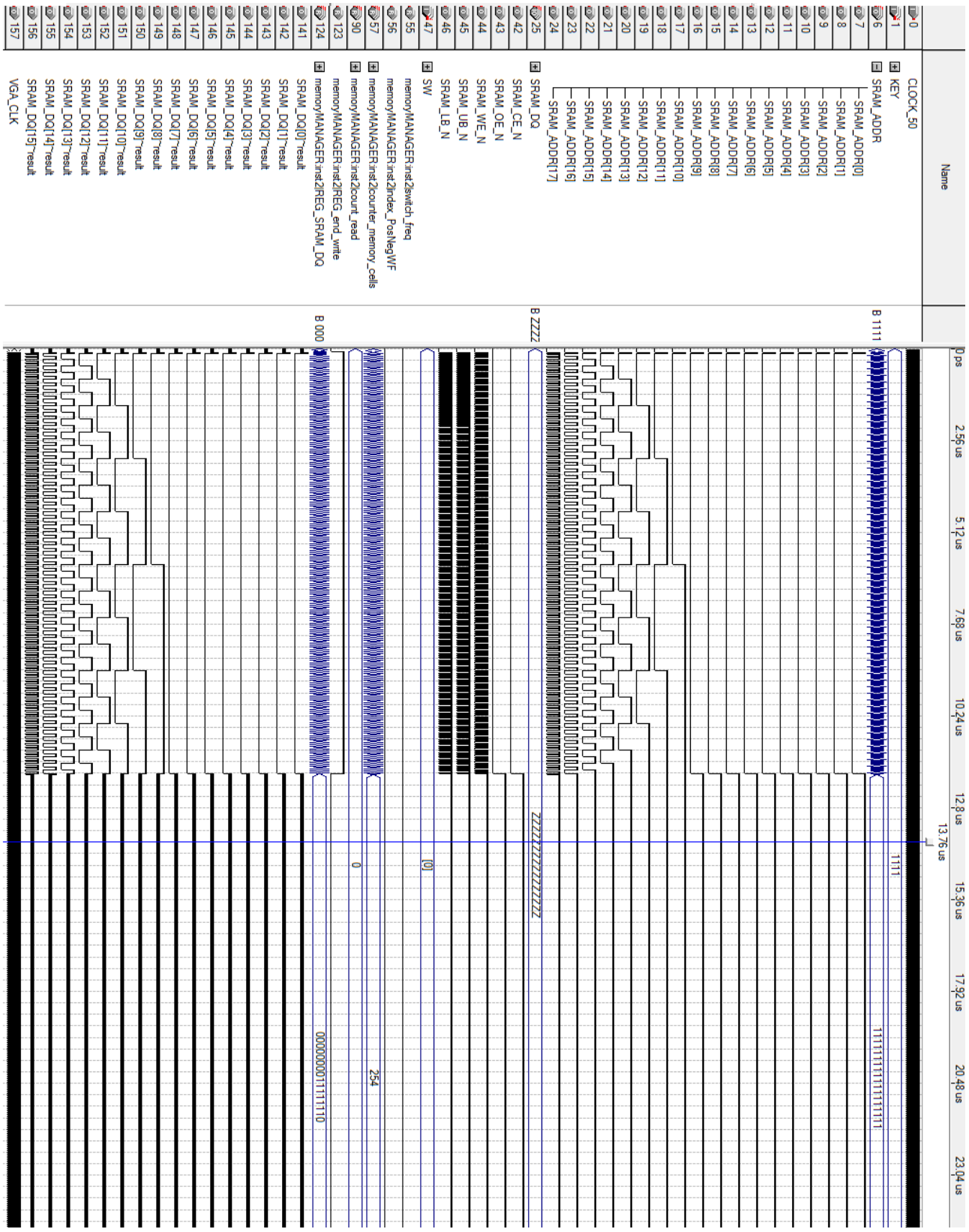


Fig. 32

Simulazione timing raffigurante i segnali di controllo generati dal Cyclone II durante tutta la fase di accesso in scrittura alla SRAM

All’atto della pressione di un pushbutton, ad esempio KEY[2], il modulo programmato (parte dell’FPGA) di pilotaggio della porta VGA, ovvero il blocco VGA_MODULATOR, è impegnato a generare gli opportuni segnali di sincronismo, ad alimentare il convertitore digitale -> analogico con le opportune stringhe binarie RGB e a richiedere al blocco memoryMANAGER, settando opportunamente i bits di controllo (con i quali VGA_MODULATOR controlla memoryMANAGER) “trg_incr_address” e “trg_decr_address”, di puntare la locazione successiva oppure quella precedente, per consentire a VGA_MODULATOR di leggere l’offset SRAM_DQ lì memorizzato e tradurre tale informazione nella giusta colorazione dei pixels. Pertanto, all’atto della pressione del pushbutton, i registri interni al modulo memoryMANAGER hanno valori assolutamente aleatori: casuali in quanto l’azione di pressione, ad esempio del pushbutton KEY[2] da parte dell’utente, è assolutamente asincrona ed eseguibile in un istante arbitrariamente scelto dall’utente stesso, oltre ad essere non conoscibili da parte dello stesso utente, viste le velocità dei segnali in gioco. Quindi, nel momento in cui l’utente preme KEY[2] per richiedere la visualizzazione su monitor della frequenza “f0_half” del dente di sega (larghezza dello schermo = metà periodo), è necessario che il codice di memoryMANAGER ne implementi il ritorno ad una situazione di “appena avvenuta programmazione via usb-blaster”, ovvero i registri devono essere reinizializzati, affinchè memoryMANAGER possa ricominciare da capo la propria routine di scrittura degli offsets del dente di sega appena selezionato, indipendentemente dal punto nel quale è stato interrotto, nella fase di lettura si presume (utente umano), a causa della pressione del tasto.

Riporto qui di seguito alcuni frammenti di codice significativi.

```
else if( (f0 == 0) && (f0_2 == 0) && (f0_half == 1) && (f0_quarter == 0) && (f0_4 == 0) )
 begin // f0/2
  start <= 1;
if(switch_freq == 0)
 begin
  REG_end_write <= 1;
  switch_freq <= 1;
  REG_SRAM_ADDR <= 18'b111111111111111111; // sfrutterò l'overflow
  REG_SRAM_DQ <= 16'b1111111111111111; // al prossimo incremento avrò: puntamento prima cella e primo dato = 0
  REG_SRAM_UB_N <= 1; REG_SRAM_LB_N <= 1; REG_SRAM_WE_N <= 1; REG_SRAM_OE_N <= 0; REG_SRAM_CE_N <= 0;
  counter_memory_cells <= 0; // scorrimento celle: 1 -> 383
  control <= 0;
  index_PosNegWF <= 0;
  index <= 0;
  count_read <= 0; // conteggio celle: 1,2,...,383 / 383,382,...,1
  rr <= 0;
 end

else // if (switch_freq == 1)
 begin // switch_freq
   switch_freq <= 1;
 if(REG_end_write == 1)
  begin
   index_PosNegWF <= 0;
   index <= 0;
   count_read <= 0;
   rr <= 0;
if(control == 0 && REG_SRAM_UB_N == 1) // al posedge del primo dei 3 cicli di clock di scrittura,
  begin                                // quando incremento SRAM_ADDR e SRAM_DQ
  REG_end_write <= 1; // mantenimento di end_write ad 1
  counter_memory_cells <= counter_memory_cells + 1;
  REG_SRAM_OE_N <= 1'b0;
  REG_SRAM_CE_N <= 1'b0; // mantenimento di CE e OE a 0
  control <= 1;
  end
else // agli altri 2 posedge
 begin
  if(counter_memory_cells == 384)
   begin
    REG_end_write <= 0;
    counter_memory_cells <= counter_memory_cells - 1; // counter_memory_cells = 383, non 384
    REG_SRAM_ADDR <= 18'b111111111111111111;          // 383 = effettivo numero di celle scritte
    REG_SRAM_DQ <= REG_SRAM_DQ;
    REG_SRAM_WE_N <= 1'b1;
    REG_SRAM_UB_N <= 1'b1;
    REG_SRAM_LB_N <= 1'b1;
    REG_SRAM_OE_N <= 1'b1;
    REG_SRAM_CE_N <= 1'b1;
    control <= 0;
   end // sfrutterò l'overflow
  else
    counter_memory_cells <= counter_memory_cells;
 end
```

entro qui al primo fronte positivo del clock successivo all'evento asincrono "pressione di KEY[2]"

end_write va portato da 0 a 1 per rendere nuovamente pilotabili le 16 uscite del tri-state, cioè per poter scrivere i 383 offsets SRAM_DQ;
switch_freq va portato da 0 a 1 per poter gestire l'eventuale rilascio di KEY[2] e quindi la ri-visualizzazione del dente di sega di default

al secondo fronte in salita del clock

alla fine della scrittura dei 383 offsets

Fig. 33

Frammento di codice implementante il trattamento del caso di switching dalla frequenza di default f0 a quella f0_half appena selezionata, oltre a quello di fine scrittura dei 383 offsets di cui consta il dente di sega f0_half

```
if (counter_memory_cells <= 383)
 begin //383
  REG_end_write <= 1; // mantenimento di end_write ad 1
  REG_SRAM_OE_N <= 1'b0;
  REG_SRAM_CE_N <= 1'b0;

 if (control == 0 && REG_SRAM_UB_N == 1) // Tclock65_primo-periodo
  begin // preparazione alla memorizzazione nelle prime 383 celle SRAM

   REG_SRAM_WE_N <= 0; //1->0 // AMBIGUITà DEL DATASHEET SRAM CIRCA IL NUMERO (alto quanto????)
                       // DI WRITE-CYCLES DURANTE I QUALI WE_N PUò ESSERE TENUTO BASSO, ATTIVO
   REG_SRAM_UB_N <= 1'b1; // scrittura ancora disabilitata
   REG_SRAM_LB_N <= 1'b1;
   control <= 1; //0->1
   REG_SRAM_ADDR <= REG_SRAM_ADDR + 1; // prima volta: puntamento della prima cella SRAM: address = 000..00
   REG_SRAM_DQ <= REG_SRAM_DQ + 1; // prima volta: datoIO = 0..0

   // circa 15ns a disposizione (T_setup molto lungo) per i segnali SRAM_ADDR e SRAM_DQ per regimare
  end
else if(control == 1 && REG_SRAM_UB_N == 1) // control = 1 && SRAM_UB_N == 1 => Tclock65_secondo-periodo : WE_N = 0 = scrittura stabilmente attiva,
 begin // segnali di address e dataIN ormai stabili...posso abbassare LB e UB, cioè attivare la scrittura
  REG_SRAM_WE_N <= 0; // mantenimento di WE_N a 0
  REG_SRAM_UB_N <= 1'b0; // memorizzazione, in cella pre-acceduta, di dato pre-formato
  REG_SRAM_LB_N <= 1'b0;
  control <= 0; //1->0
  counter_memory_cells <= counter_memory_cells;
  REG_SRAM_ADDR <= REG_SRAM_ADDR;
  REG_SRAM_DQ <= REG_SRAM_DQ; // mantenimento di dato e address
 end
else if(control == 0 && REG_SRAM_UB_N == 0) // Tclock65_terzo-periodo : alzo WE_N, UB_N e LB_N => disabilitazione della scrittura
 begin // 15ns = tempo di Hold per address e dataIN molto lungo
  REG_SRAM_WE_N <= 1'b1; //0->1

  REG_SRAM_UB_N <= 1'b1;
  REG_SRAM_LB_N <= 1'b1;

  control <= 0; // control rimane = 0
  counter_memory_cells <= counter_memory_cells;
  REG_SRAM_ADDR <= REG_SRAM_ADDR;
  REG_SRAM_DQ <= REG_SRAM_DQ; // mantenimento di dato e address
 end
 else
  begin
   REG_SRAM_WE_N <= 1'b1;
   REG_SRAM_UB_N <= 1'b1;
   REG_SRAM_LB_N <= 1'b1;
   counter_memory_cells <= counter_memory_cells;
  end
end //383
 else // if(counter_memory_cells >= 384)
  begin
   REG_SRAM_UB_N <= 1'b1;
   REG_SRAM_LB_N <= 1'b1;
   REG_SRAM_WE_N <= 1'b1;
   REG_SRAM_OE_N <= 1'b1;
   REG_SRAM_CE_N <= 1'b1;
  end
end // fine scrittura dei 383 offsets nelle 383 celle
```

Tsetup di address e dato

Tscrittura dato in cella_address

Thold di dato e address

Fig. 34

Parte del codice del blocco memoryMANAGER implementante la funzionalità di generazione dei segnali di controllo e di accesso alla SRAM, durante i 3 cicli di clock coinvolti nella scrittura di un certo dato DQ offset, del dente di sega f0_half, in una certa locazione SRAM

Di seguito riporto la porzione del file.vwf in cui si vede il comportamento dei segnali di controllo della SRAM, pilotati dal Cyclone II, nel caso in cui dopo aver richiesto la visualizzazione su monitor della frequenza di default f0, l'utente richieda di vedere il dente di sega f0_half.

Va detto, per essere rigorosi, che la simulazione in questione si riferisce al caso in cui l'evento asincrono e arbitrario da parte dell'utente, ossia la pressione del pushbutton KEY[2], avvenga in un istante nel quale la visualizzazione della forma d'onda di default f0, già completamente scritta in SRAM, non è ancora cominciata sul monitor: in quegli istanti, infatti, il blocco VGA_MODULATOR sta ancora pilotando, sui terminali di ingresso del DAC RGB-VGA, il segnale di sincronismo orizzontale VGA_HS associato alla prima, delle 6 righe, che devono essere generate durante l'impulso attivo basso del segnale di sincronismo verticale VGA_VS. In altre parole nell'istante in cui il vettore KEY (uno script di input per il simulatore) passa da 1111 a 1011, un utente umano non avrebbe ancora visto, su monitor, la forma d'onda f0, benché questa avesse i suoi 254 offsets già pronti in SRAM. Ho optato per questa simulazione dal momento che per una simulazione timing di 5 – 10 ms il simulatore può impiegare anche alcuni minuti, pertanto la simulazione di una situazione tangibile, visibile, da parte di un utente umano, che pertanto si sarebbe dovuta prolungare per un lasso di tempo almeno dell'ordine del secondo, avrebbe impiegato diverse ore.

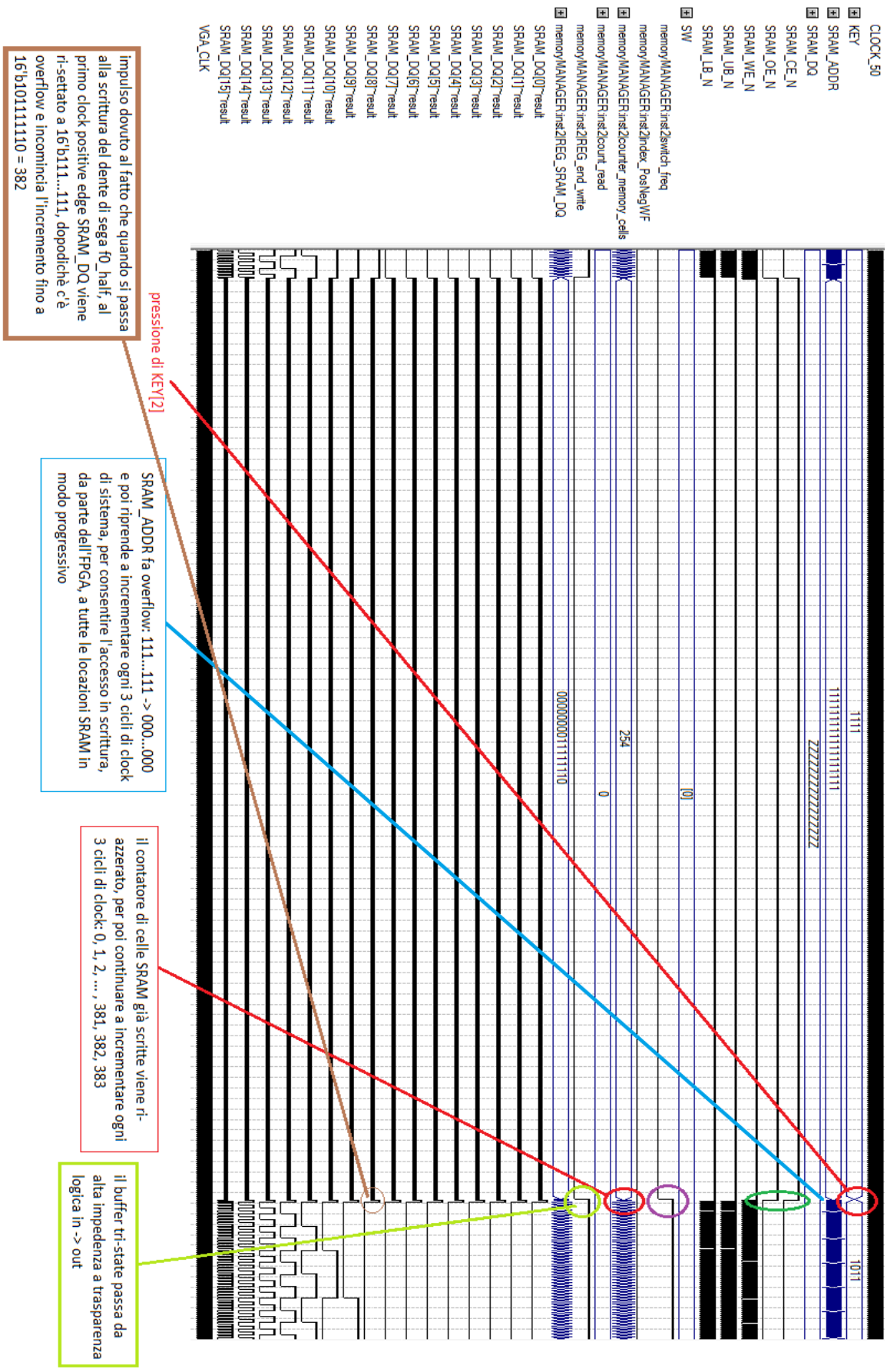

CLOCK_50
KEY
SRAM_ADDR
SRAM_DQ
SRAM_CE_N
SRAM_OE_N
SRAM_WE_N
SRAM_UB_N
SRAM_LB_N
SW
memoryMANAGER:inst2|switch_freq
memoryMANAGER:inst2|index_PosNegWF
memoryMANAGER:inst2|counter_memory_cells
memoryMANAGER:inst2|count_read
memoryMANAGER:inst2|REG_end_write
memoryMANAGER:inst2|REG_SRAM_DQ
SRAM_DQ[0]"result
SRAM_DQ[1]"result
SRAM_DQ[2]"result
SRAM_DQ[3]"result
SRAM_DQ[4]"result
SRAM_DQ[5]"result
SRAM_DQ[6]"result
SRAM_DQ[7]"result
SRAM_DQ[8]"result
SRAM_DQ[9]"result
SRAM_DQ[10]"result
SRAM_DQ[11]"result
SRAM_DQ[12]"result
SRAM_DQ[13]"result
SRAM_DQ[14]"result
SRAM_DQ[15]"result
VGA_CLK
1111
1111111111111111
ZZZZZZZZZZZZZZZZ
[0]
254
0
0000000011111110
1011
pressione di KEY[2]
impulso dovuto al fatto che quando si passa alla scrittura del dente di sega f0_half, al primo clock positive edge SRAM_DQ viene ri-settato a 16'b111...111, dopodichè c'è overflow e incomincia l'incremento fino a 16'b1011111110 = 382
SRAM_ADDR fa overflow: 111...111 -> 000...000 e poi riprende a incrementare ogni 3 cicli di clock di sistema, per consentire l'accesso in scrittura, da parte dell'FPGA, a tutte le locazioni SRAM in modo progressivo
il contatore di celle SRAM già scritte viene ri-azzerato, per poi continuare a incrementare ogni 3 cicli di clock: 0, 1, 2, ... , 381, 382, 383
il buffer tri-state passa da alta impedenza a trasparenza logica in -> out

Fig. 35

Simulazione timing raffigurante il passaggio fra memorizzazione in SRAM dei 254 offsets relativi alla frequenza di default e la richiesta, da parte dell'utente, di scrivere in SRAM i 383 offsets relativi al dente di sega f0_half

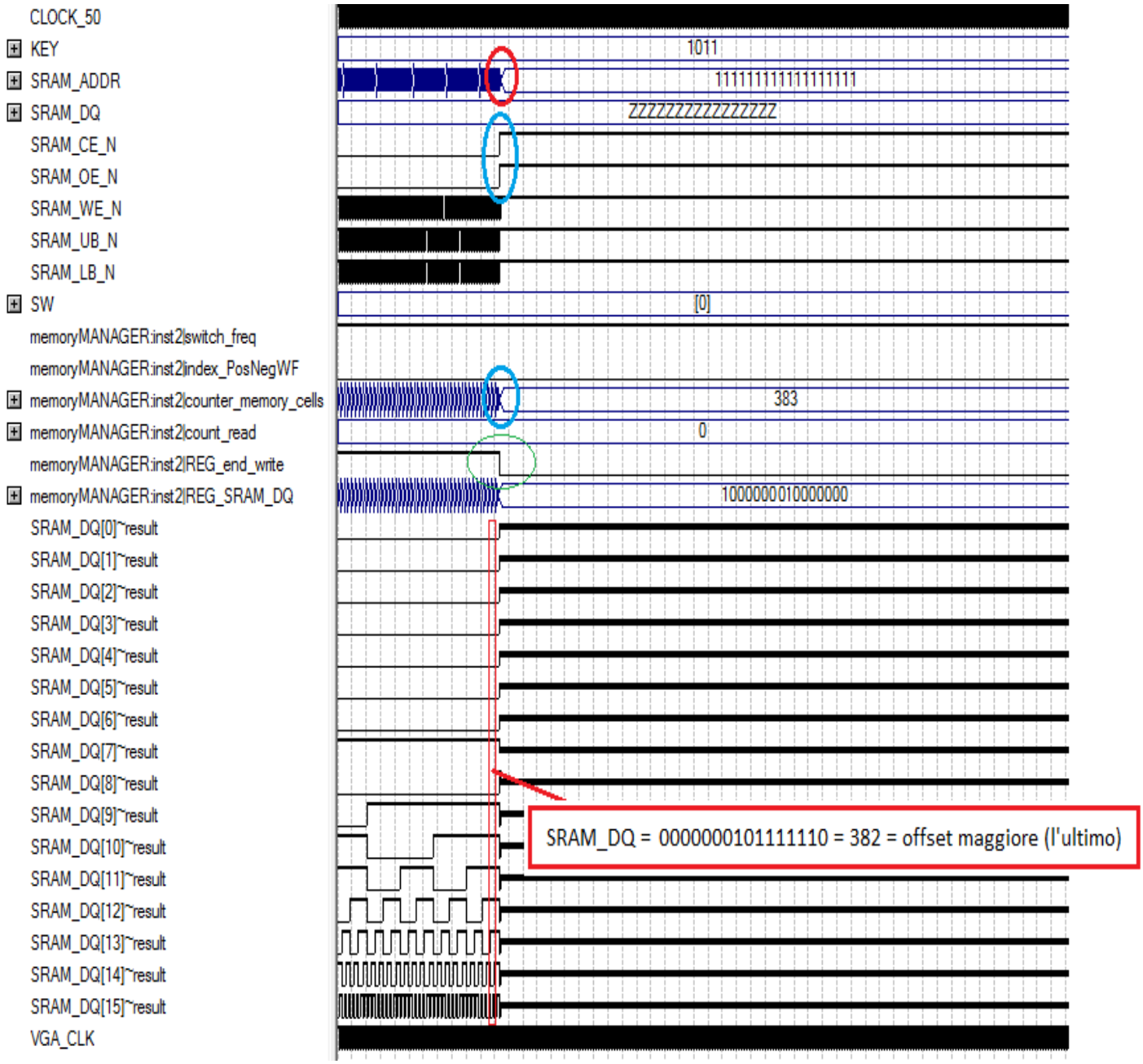


Fig. 36

Simulazione timing raffigurante gli istanti conclusivi della scrittura in SRAM dei 383 offsets relativi al dente di sega f0_half

4.4) Fase di lettura dati: interazione fra VGA_MODULATOR e memoryMANAGER

Il blocco VGA_MODULATOR, in buona sostanza, genera i segnali di sincronismo orizzontale e verticale da inviare rispettivamente sui PINs 13 e 14 del connettore D-SUB VGA, e modula le 3 stringhe binarie ciascuna delle quali, composta da 10 bits, costituisce l'informazione digitale relativa all'intensità di uno dei 3 colori di base (rosso, verde, blu): ciascuna stringa, data in ingresso al DAC-RGB, è convertita nel corrispondente valore analogico di corrente da inviare al connettore VGA che si trova a bordo della scheda. Il blocco VGA_MODULATOR scandisce pertanto il monitor CRT secondo lo standard XGA e quindi conosce la posizione, ad ogni ciclo di clock di sistema, del fascio elettronico proiettato sullo schermo, grazie ai suoi registri interni count_y e count_x. VGA_MODULATOR, all'atto della selezione, da parte dell'utente, della frequenza del dente di sega da visualizzare, modula alcuni parametri come l'ordine della riga su monitor di inizio e fine della porzione di frame contenente la forma d'onda da visualizzare, ossia modula l'ordine della riga in corrispondenza della quale finisce la fascia bianca superiore e quella da cui comincia la fascia bianca inferiore; modula, inoltre, il valore dei pixels-cardine rispetto ai quali riferire gli offsets. Tuttavia questi ultimi non sono informazioni note al VGA_MODULATOR, pertanto questo deve stimolare opportunamente il memoryMANAGER affinchè glie le comunichi. Il memoryMANAGER ha infatti accesso alla SRAM, può puntare (indirizzare) qualunque sua locazione sia in scrittura che in lettura, quindi una volta che tutti gli offsets, relativi al dente di sega desiderato, sono stati memorizzati in SRAM, il memoryMANAGER deve scorrere la porzione di SRAM nel giusto modo, accedere alla giusta locazione, indirizzandola, così da mettere a disposizione del VGA_MODULATOR il dato SRAM_DQ, su 16 bits, in questa contenuto: dato che, letto in ingresso dal VGA_MODULATOR, consente allo stesso di calcolare, per mezzo del confronto il cui codice riporto qui di seguito, presso quali pixels stampare su monitor il bianco e presso quali stampare il blu, ossia la forma d'onda.

```
else if((count_y >= 164) && (count_y <= 417))
  begin
        //dente di sega : semi-onda positiva
      ...
    if( (count_clock65_row == (256 - SRAM_DQ)) || (count_clock65_row == (256 + SRAM_DQ)) )
      begin // offset => pixel cardine: 256. man mano che scandisco la SRAM, l'offset aumenta
            // offsets memorizzati : 0 in cella1, 1 in cella2, 2 in cella3..., 253 in cella 25
            // la distanza fra i 2 punti blu (lungo la linea bianca) si allarga
        REG_VGA_R <= 10'b0000000000;
        REG_VGA_G <= 10'b0000000000;
        REG_VGA_B <= 10'b1111111111; // pixel blu
      end
    else
      begin
        REG_VGA_R <= 10'b1111111111;
        REG_VGA_G <= 10'b1111111111;
        REG_VGA_B <= 10'b1111111111; // altrove, sulla linea, tutti punti bianchi
      end

      ...
```

Fig. 37

Frammento di codice del blocco VGA_MODULATOR implementante i confronti necessari a calcolare presso quali pixels stampare, su monitor, il bianco e presso quali stampare il blu, ossia la forma d'onda

Il memoryMANAGER ha accesso agli offsets SRAM_DQ che servono al VGA_MODULATOR durante la scansione dello schermo, quindi è il memoryMANAGER che glie li deve fornire, indirizzandoli mediante opportuna generazione dei segnali SRAM_ADDR. Tuttavia il memoryMANAGER non conosce, istante per istante, la posizione del raggio catodico, ossia quali sono i valori (numeri interi) memorizzati nei registri count_x e count_y, che sono appunto interni al VGA_MODULATOR, pertanto è stato necessario realizzare un interfacciamento diretto VGA_MODULATOR -> memoryMANAGER per mezzo di 2 bits, "trg_incr_address" e "trg_decr_address" ("sollecitazione, triggering, all'incremento di indirizzo" e "sollecitazione al decremento di indirizzo"), che sono opportunamente pilotati da VGA_MODULATOR e letti, rilevati, da memoryMANAGER. In tal contesto il memoryMANAGER è lo "slave" che legge, ad ogni colpo di clock di sistema, l'informazione che il VGA_MODULATOR, il "master", gli ha comunicato il periodo di clock precedente. Questa informazione consiste nella coppia di bits sopra citata e assume una codifica "one-hot", dunque ad ogni colpo di clock il memoryMANAGER leggerà solo uno dei 2 bits a valore logico alto, mentre l'altro sarà sempre a valore logico basso. Il memoryMANAGER legge l'informazione proveniente dal VGA_MODULATOR, inoltre legge il valore logico di un suo registro interno, un suo bit, che ho chiamato "index", e solo adesso è in grado di prendere una decisione, ovvero se incrementare il registro contenente l'indirizzo, su 18 bits, della locazione SRAM da puntare, se invece decrementarlo, quindi se scorrere la parte di SRAM contenente gli offsets in modo progressivo o regressivo, oppure se mantenere, durante il periodo di clock corrente, puntata la stessa locazione del ciclo di clock precedente, oppure ancora se disabilitare o abilitare la lettura dell'offset all'interno della locazione attualmente puntata. memoryMANAGER setta opportunamente, inoltre, il proprio bit "index", affinchè al ciclo di clock successivo possa prendere autonomamente la decisione giusta: tale situazione è utile in corrispondenza di discontinuità del comportamento del VGA_MODULATOR, ad esempio sul bordo sinistro e destro di ciascuna riga in fase di scansione.

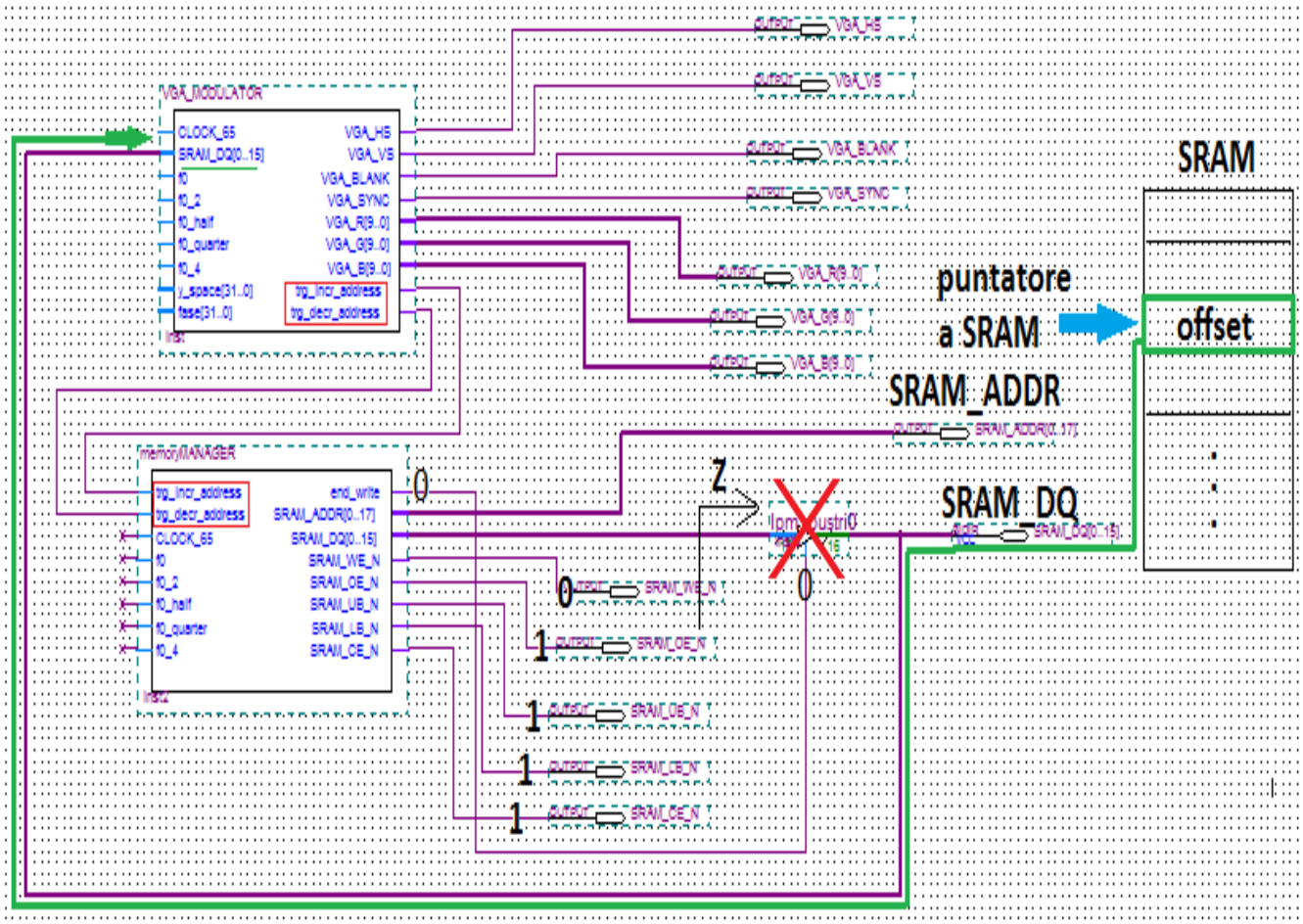


Fig. 38

Parte del top-level file “DENTE_SEGA_system.bdf” raffigurante l’interazione fra VGA_MODULATOR e memoryMANAGER durante la fase di lettura degli offsets relativi ad una qualunque frequenza del dente di sega selezionata

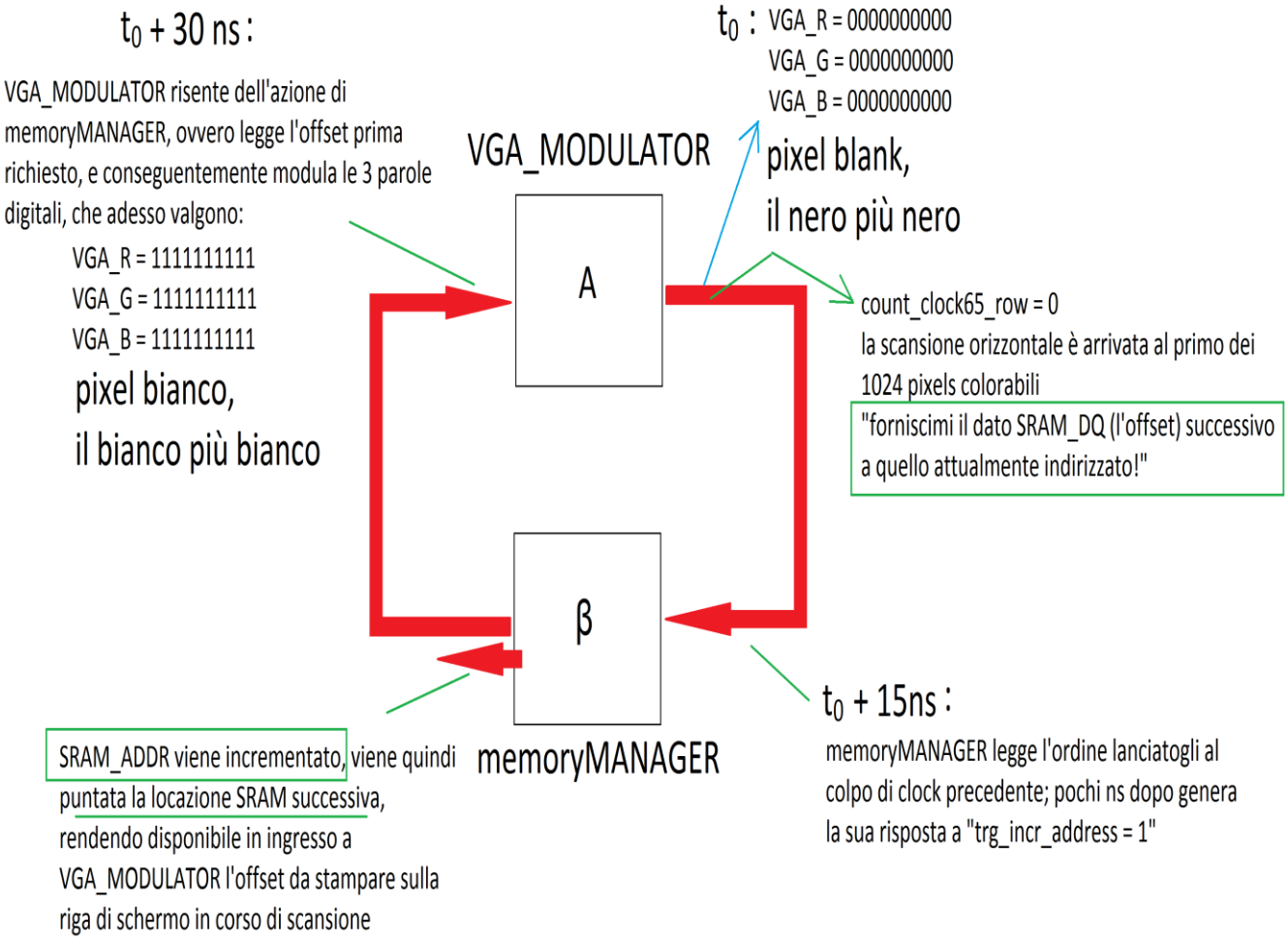


Fig. 39

Modellizzazione, secondo il noto schema funzionale a blocchi della teoria della retroazione, dell'interazione fra VGA_MODULATOR e memoryMANAGER

Riporto qui di seguito una descrizione illustrativa della dinamica dell'aggiornamento dei registri "trg_incr_address" , "trg_decr_address" e "index", durante la scansione del monitor, che regola l'interazione fra VGA_MODULATOR e memoryMANAGER nella fase di lettura degli offsets. È stato preso, a titolo di esempio, il caso di dente di sega f0 di default.

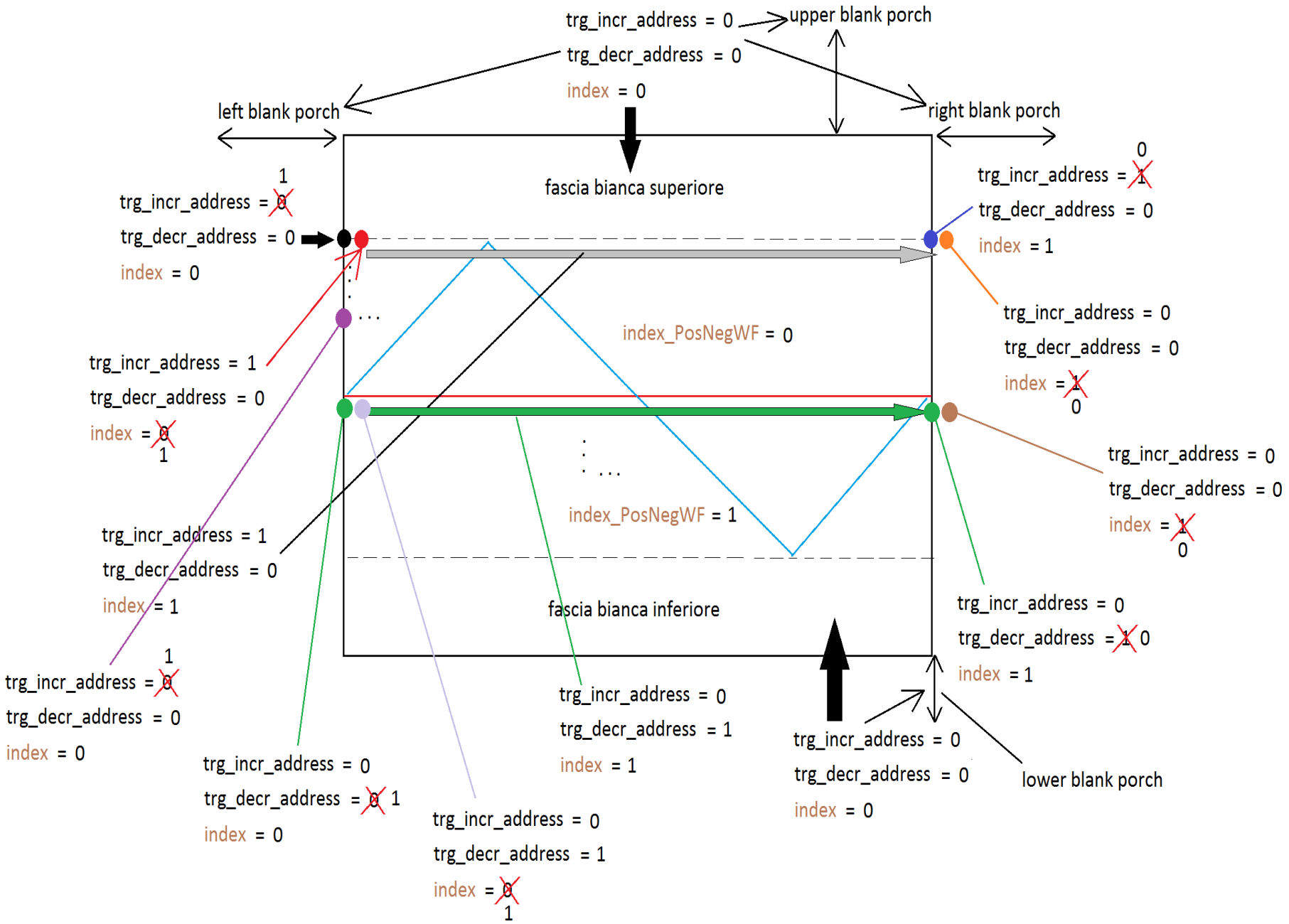

trg_incr_address = 0
trg_decr_address = 0
index = 0
upper blank porch
left blank porch
right blank porch
fascia bianca superiore
trg_incr_address = 1 0
trg_decr_address = 0
index = 1
trg_incr_address = 0
trg_decr_address = 0
index = 1 0
trg_incr_address = 1 0
trg_decr_address = 0
index = 0
index_PosNegWF = 0
trg_incr_address = 1
trg_decr_address = 0
index = 0 1
trg_incr_address = 0
trg_decr_address = 0
index = 1 0
trg_incr_address = 1
trg_decr_address = 0
index = 1
index_PosNegWF = 1
fascia bianca inferiore
trg_incr_address = 1 0
trg_decr_address = 0
index = 0
trg_incr_address = 0
trg_decr_address = 1 0
index = 1
trg_incr_address = 0
trg_decr_address = 0 1
index = 0
trg_incr_address = 0
trg_decr_address = 1
index = 1
trg_incr_address = 0
trg_decr_address = 1
index = 0 1
trg_incr_address = 0
trg_decr_address = 0
index = 0
lower blank porch

Fig. 40

Dinamica dell'aggiornamento dei registri "trg_incr_address" , "trg_decr_address" e "index", durante la scansione del monitor da parte del fascio elettronico, in fase di lettura degli offsets

Qui di seguito mostro i frammenti di codice verilog del blocco memoryMANAGER implementanti il comportamento dello stesso durante la fase di lettura degli offsets, e quindi di visualizzazione su monitor della forma d'onda selezionata.

```
else // if(end_write == 0)
  begin // read

 if(index_PosNegWF == 0)

   begin // positive WF

     if(trg_incr_address == 0 && index == 0)
      begin
     // lettura celle non ancora iniziata, scansione VGA ancora fuori dal sow-frame
         count_read <= count_read;
         index <= 1'b0;
         index_PosNegWF <= 0;
         REG_SRAM_ADDR <= REG_SRAM_ADDR;
         REG_SRAM_UB_N <= 1'b1;
         REG_SRAM_LB_N <= 1'b1;
         REG_SRAM_CE_N <= 1'b1;
         REG_SRAM_OE_N <= 1'b1; // lettura disabilitata
         rr <= 0;
      end

    else if(trg_incr_address == 1 && index == 0) // 0->1
      begin

      index_PosNegWF <= 0;

       if(rr == 0)
        begin
         index <= 1'b0;
         count_read <= count_read + 1;
         REG_SRAM_ADDR <= REG_SRAM_ADDR + 1;
         REG_SRAM_CE_N <= 1'b0; // chip=memoria SRAM abilitata
         REG_SRAM_OE_N <= 1'b1; // ma lettura ancora disabilitata
         REG_SRAM_UB_N <= 1'b1;
         REG_SRAM_LB_N <= 1'b1;
         rr <= 1'b1; // 0->1
        end
       else // if(rr == 1)
        begin
         REG_SRAM_ADDR <= REG_SRAM_ADDR;
         count_read <= count_read;
         REG_SRAM_CE_N <= 1'b0;
         REG_SRAM_OE_N <= 1'b0;
         REG_SRAM_UB_N <= 1'b0;
         REG_SRAM_LB_N <= 1'b0; //
         index <= 1'b1; // index: 0->1
         rr <= 1'b0; // 1->0
        end

      end
```

all'ultimo fronte in salita del clock, in fase di scrittura, memoryMANAGER ha togglato il bit "end_write" (tri-state in alta impedenza), quindi al primo clock positive edge successivo alla commutazione del registro end_write, si entra qui

finchè il VGA_MODULATOR scandisce le righe della fascia bianca superiore, mantiene a 0 il flag "trg_incr_address", mentre memoryMANAGER mantiene a 0 il suo registro interno "index", quindi a ogni fronte positivo del clock, mentre VGA_MODULATOR scandisce nella fascia bianca superiore, si entra qui, e memoryMANAGER continua a puntare l'ultima locazione SRAM

vi entriamo anche quando VGA_MODULATOR scandisce il "left blank porch", il "right blank porch", l'impulso attivo basso sia del sincronismo orizzontale che di quello verticale, oltre all'asse rosso

primo periodo di clock di lettura: aggiorno/incremento address e abilito il chip della SRAM

Tsetup per l'address = 15 ns

secondo (e ultimo) periodo di clock di lettura: SRAM_ADDR ormai stabile, posso abilitare la lettura del dato

dente di sega, alla fine della scansione del "left blank porch", cioè al primo pixel della prima riga del dente di sega (pallino nero), VGA_MODULATOR toggla il flag "trg_incr_address" da 0 a 1, quindi al successivo colpo di clock (scansione del secondo pixel) memoryMANAGER rileva la commutazione del flag ed entriamo lì: incremento dell'address, dunque overflow 111..11 -> 000..00, quindi memoryMANAGER punta la prima locazione, consentendone la lettura del dato (offset nullo) al VGA_MODULATOR, che lo tradurrà nel corretto posizionamento dei pixels blu

```
else if(trg_incr_address == 1 && index == 1)
  begin
   count_read <= count_read;
   rr <= 1'b0;
   index <= 1'b1; // mantenimento di index ad 1
   index_PosNegWF <= 0;
   REG_SRAM_ADDR <= REG_SRAM_ADDR; // mantenimento indirizzo
   REG_SRAM_UB_N <= 1'b0;
   REG_SRAM_LB_N <= 1'b0;
   REG_SRAM_CE_N <= 1'b0;
   REG_SRAM_OE_N <= 1'b0;
   // dato DQ mantenuto in lettura durante tutti i 1024 pixels
  end
else if(trg_incr_address == 0 && index == 1) // 1->0
  begin
   REG_SRAM_UB_N <= 1'b1;
   REG_SRAM_LB_N <= 1'b1;
   REG_SRAM_CE_N <= 1'b1;
   REG_SRAM_OE_N <= 1'b1;
   index <= 1'b0; // index: 1->0
   rr <= 1'b0;

   if(count_read == 254)
    begin
     index_PosNegWF <= 1'b1;
     REG_SRAM_ADDR <= REG_SRAM_ADDR + 1;
     count_read <= count_read + 1; // count_read = 255;
    end

   else
    begin
     index_PosNegWF <= 1'b0;
     count_read <= count_read;
     REG_SRAM_ADDR <= REG_SRAM_ADDR;
    end

  end

end // positive WF
```

durante tutta la scansione di una riga del dente di sega (della semionda positiva), il VGA_MODULATOR mantiene a 1 il flag "trg_incr_address", così memoryMANAGER, ad ogni colpo di clock, dal secondo pixel al 1024°esimo pixel, rileva lo stato del flag ed entra lì

appena VGA_MODULATOR arriva a scandire il 1024°esimo pixel della riga del dente di sega, appena dopo il colpo di clock corrispondente, commuta lo stato del flag da 1 a 0, così al fronte positivo del clock successivo, memoryMANAGER rileva il cambiamento avvenuto su quel suo bit di ingresso 15 ns prima ed entrì qui: disabilita la lettura della cella attualmente puntata

se il registro interno a memoryMANAGER, che conta il numero di locazioni SRAM accedute in lettura fino a quel momento, è arrivato a 254, allora il registro che determina il verso di scorrimento del puntatore alle locazioni SRAM viene commutato da 0 a 1 e SRAM_ADDR viene incrementato, così che al primo decremento (semionda negativa, scorrimento regressivo delle locazioni) di SRAM_ADDR abbiamo la lettura del massimo offset (=253)

se la scansione del semi-frame sopra l'asse non è ancora terminato

Fig. 41

Frammento di codice verilog del blocco memoryMANAGER implementante il comportamento dello stesso durante la fase di lettura degli offsets e di visualizzazione della semionda positiva

```
else // if(index_PosNegWF == 1)
  begin // negative WF
    if(trg_decr_address == 0 && index == 0)
     begin
      count_read <= count_read;
      rr <= 1'b0;
      index <= 1'b0; // mantenimento di index a 0
      index_PosNegWF <= 1;
      REG_SRAM_ADDR <= REG_SRAM_ADDR;
      REG_SRAM_UB_N <= 1'b1;
      REG_SRAM_LB_N <= 1'b1;
      REG_SRAM_CE_N <= 1'b1;
      REG_SRAM_OE_N <= 1'b1; // lettura disabilitata
     end

    else if(trg_decr_address == 1 && index == 0) // 0->1
    // scansione SRAM celle a rovescio: 254 -> 1 (semi-onda negativa)
     begin
     index_PosNegWF <= 1;
      if(rr == 0)
       begin
        index <= 1'b0;
        count_read <= count_read - 1;
        REG_SRAM_ADDR <= REG_SRAM_ADDR - 1;
        REG_SRAM_CE_N <= 1'b0; // chip=memoria SRAM abilitata
        REG_SRAM_UB_N <= 1'b1; // ma lettura ancora disabilitat
        REG_SRAM_LB_N <= 1'b1;
        REG_SRAM_OE_N <= 1'b1;
        rr <= 1'b1; // 0->1
       end
      else // if(rr == 1)
       begin
        REG_SRAM_ADDR <= REG_SRAM_ADDR;
        count_read <= count_read;
        REG_SRAM_CE_N <= 1'b0;
        REG_SRAM_OE_N <= 1'b0;
        REG_SRAM_UB_N <= 1'b0;
        REG_SRAM_LB_N <= 1'b0; //
        index <= 1'b1; // index: 0->1
        rr <= 1'b0; // 1->0
       end
    end
```

VGA_MODULATOR scandisce il "left blank porch", oppure il "right blank porch", oppure la fascia bianca inferiore, oppure l'impulso attivo basso del segnale di sincronismo orizzontale

al primo pixel della prima riga del dente di sega sotto l'asse, VGA_MODULATOR commuta il flag "trg_decr_address" da 0 a 1, così al successivo colpo di clock (scansione secondo pixel) memoryMANAGER rileva 1 sul suo ingresso ed entriamo lì: primo ciclo di clock per decrementare/aggiornare l'address e, dopo un Tsetup = 15 ns, abilitazione alla lettura da parte del VGA_MODULATOR

```
else if(trg_decr_address == 1 && index == 1)
 begin
  count_read <= count_read;
  rr <= 1'b0;
  index <= 1'b1; // mantenimento di index ad 1
  index_PosNegWF <= 1;
  REG_SRAM_ADDR <= REG_SRAM_ADDR; // mantenimento address corrente
  REG_SRAM_UB_N <= 1'b0;
  REG_SRAM_LB_N <= 1'b0;
  REG_SRAM_CE_N <= 1'b0;
  REG_SRAM_OE_N <= 1'b0; // mantenimeto lettura DQ per tutta la sow-row
 end
else if(trg_decr_address == 0 && index == 1) // 1->0
 begin
  REG_SRAM_UB_N <= 1'b1;
  REG_SRAM_LB_N <= 1'b1;
  REG_SRAM_CE_N <= 1'b1;
  REG_SRAM_OE_N <= 1'b1;
  index <= 1'b0; // index: 1->0
  rr <= 1'b0;

  if(count_read == 1)
   begin
    index_PosNegWF <= 1'b0;
    REG_SRAM_ADDR <= 18'b111111111111111111;
    count_read <= count_read - 1;
   end
  else
   begin
    index_PosNegWF <= 1'b1;
    count_read <= count_read;
    REG_SRAM_ADDR <= REG_SRAM_ADDR;
   end

 end

 end // negative WF

end // read
```

durante la scansione della riga da parte di VGA_MODULATOR, memoryMANAGER punta la solita locazione SRAM, così che VGA_MODULATOR abbia sempre a disposizione, in ingresso, l'offset contenuto in quella locazione

VGA_MODULATOR sta scandendo l'ultimo pixel della riga, ri-commuta il flag per comunicare al memoryMANAGER che non ha più bisogno dell'offset contenuto in quella locazione ancora puntata: disabilitazione della lettura in vista di un prossimo aggiornamento/decremento del puntatore a SRAM

se lo scorrimento a ritroso delle locazioni contenenti gli offsets è arrivato a conclusione, ossia all'offset nullo (alla prima locazione), cioè se la visualizzazione del semi-frame relativo alla semionda negativa è terminato, allora memoryMANAGER reimposta lo scorrimento progressivo delle locazioni SRAM, in vista di una prossima semionda positiva (nuovo frame da visualizzare), oltre a puntare nuovamente l'ultima locazione, in vista di un prossimo incremento, ossia overflow (si ritorna alla prima locazione)

Fig. 42

Frammento di codice verilog del blocco memoryMANAGER implementante il comportamento dello stesso durante la fase di lettura degli offsets e di visualizzazione della semionda negativa

5) Modulazione di ampiezza

5.1) Obbiettivo del sistema

Il primo stadio dell'acquisizione dati, relativamente alla modulazione di ampiezza del dente di sega f0_4, è costituito dai primi 5 switches, partendo da destra, disponibili sulla scheda DE2 (SW[4] … SW[0]). Supponiamo che all'atto della programmazione del Cyclone II tutti gli switches siano nello stato DOWN, e che l'utente prema il pushbutton KEY[0] al fine di visualizzare il dente di sega avente periodo pari ad un quarto dello schermo (frequenza quadrupla "f0_4"). Supponiamo adesso che l'utente desideri, progressivamente da destra verso sinistra, commutare uno dopo l'altro i 5 switches di cui sopra, lasciando nello stato UP gli switches precedenti ad ogni nuova commutazione DOWN -> UP. Questa progressiva commutazione DOWN -> UP dei 5 SW[] deve tradursi in una altrettanto progressiva modulazione crescente dell'ampiezza del dente di sega f0_4. Con tutti e 5 gli SW[] in stato DOWN abbiamo un dente di sega avente ampiezza pari a 63 righe; se l'utente solleva SW[0] l'ampiezza della forma d'onda, ossia il numero di righe dal punto di picco all'asse rosso, deve raddoppiare. Se l'utente, successivamente, solleva anche SW[1], allora l'ampiezza deve incrementare di altre 63 righe, portandosi ad un valore triplo rispetto a quello di default, ecc… Quando, infine, l'utente solleva SW[4] (gli altri 4 switches sono sollevati), l'ampiezza del dente di sega deve assumere un valore sestuplo rispetto a quello di default. Inoltre se l'utente inizia a commutare regressivamente, in modo UP-> DOWN, gli SW[] finora sollevati progressivamente, sul monitor deve essere visualizzato il progressivo decremento del dente di sega: la ragione di questa regressione geometrica deve essere pari a quella della precedente progressione, ovvero 63 righe ad ogni commutazione.

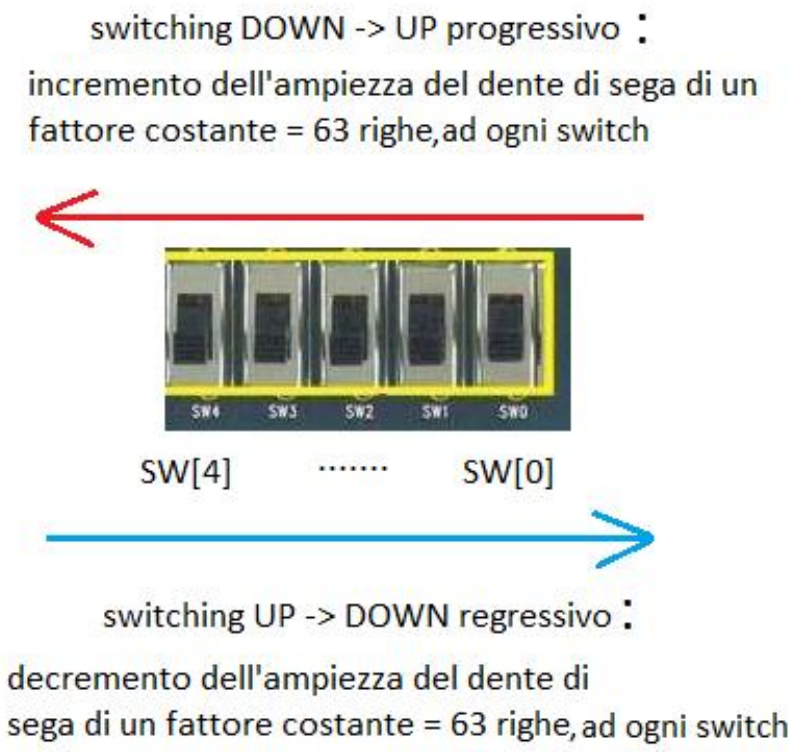


Fig. 43

Azioni da parte dell'utente che comportano, su monitor, un progressivo incremento e decremento dell'ampiezza del dente di sega f0_4

Di seguito mostro un diagramma di flusso descrivente la funzionalità, su richiesta dell'utente, di "stiramento" e "compressione" del modulo del dente di sega f0_4 da parte del blocco VGA_MODULATOR. Questo diagramma di flusso può essere pensato come una parte di quello principale riportato in figura 3, da innestare all'interno del blocco di perimetro blu facente parte del ramo (quello più a destra) che tratta la selezione, mediante pressione di KEY[0], del dente di sega f0_4. Nel diagramma di flusso sottostante i blocchi Ω rappresentano funzionalmente l'azione di reinizializzazione dei registri interni a VGA_MODULATOR: il funzionamento del sistema passa attraverso i blocchi Ω ogni qual volta l'utente commuta DOWN -> UP uno switch durante la successione progressiva di commutazioni atta all'espansione del modulo del dente di sega oppure durante la successione regressiva delle commutazioni, finalizzata alla ricompressione della forma d'onda. I blocchi V invece rappresentano funzionalmente tutte quelle operazioni finalizzate alla stampa su schermo del dente di sega corrente, implementate da VGA_MODULATOR, sia a livello di suoi registri interni, sia a livello di pilotaggio di segnali verso memoryMANAGER e verso i dispositivi periferici, ovvero fuori dall'FPGA, come i 3 video-DACs o come il connettore VGA D-SUB. In corrispondenza del fronte di salita del clock di sistema, entrando pertanto nella struttura if else if controllata dal segnale y_space[31:0] (controllo eseguito all'attivazione dell'always@ interno al codice di VGA_MODULATOR), viene valutata la condizione espressa dai blocchi romboidali, che appunto sono blocchi condizionali del tipo "lo stato dei 5 switches SW[4]…SW[0] è cambiato?" (si intende rispetto al colpo di clock precedente): se la risposta è negativa, cioè se l'utente vuole visualizzare la stessa ampiezza di prima, allora VGA_MODULATOR procede, collaborando con memoryMANAGER, alla visualizzazione della stessa forma d'onda, se invece la risposta è affermativa si procede ad una reinizializzazione dei registri di VGA_MODULATOR, in uno dei 2 sensi possibili, ossia incremento o decremento dell'ampiezza (solo lo stato di tutti gli SW[] in condizione DOWN e quello di tutti gli SW[] in condizione UP hanno, ovviamente, un unico senso possibile di evoluzione).

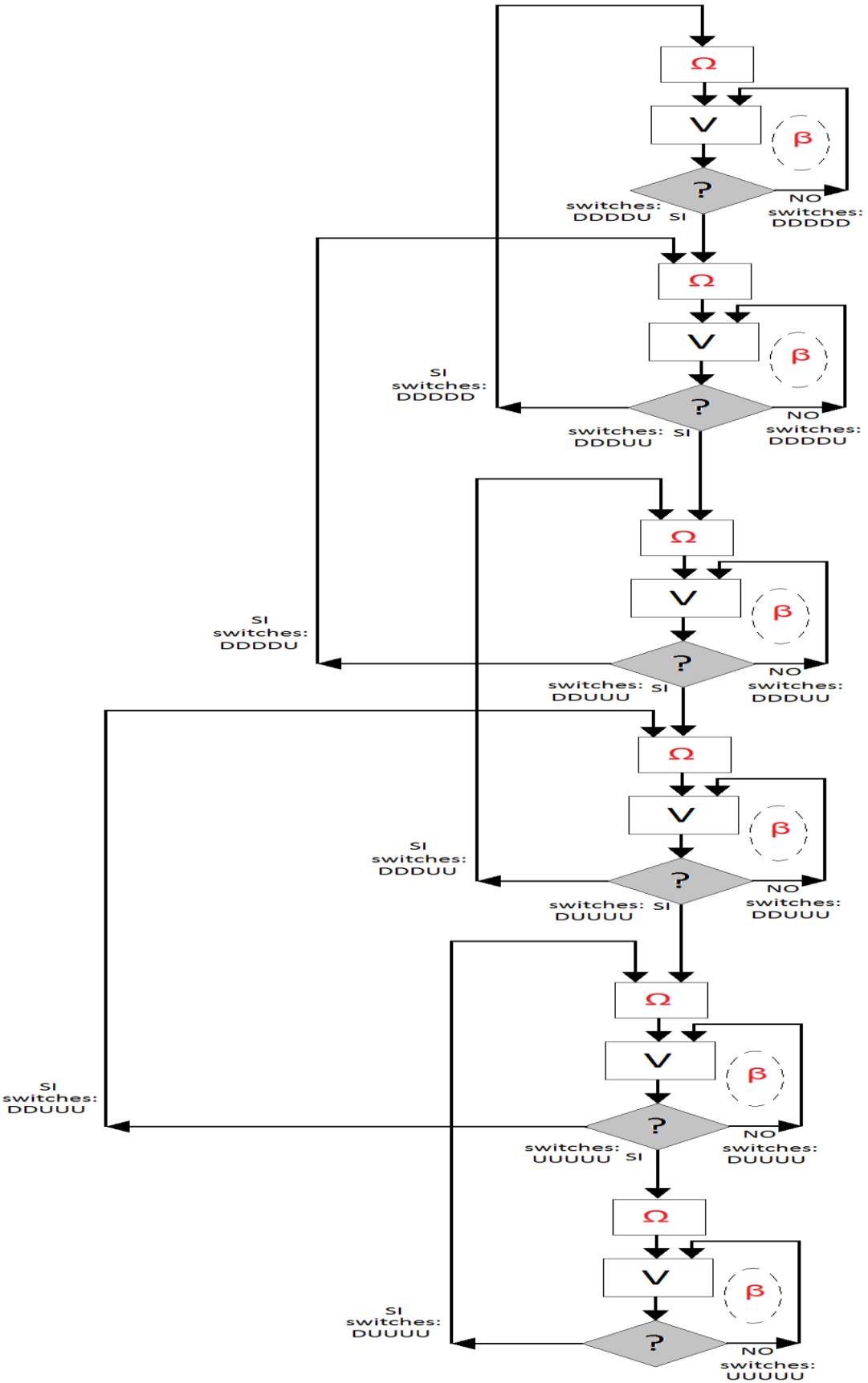


Fig. 44

Diagramma di flusso descrivente la funzionalità di incremento e decremento dell'ampiezza del dente di sega f0_4 da parte del blocco VGA_MODULATOR

5.2) Realizzazione logica

Per rendere possibile tutto ciò ho implementato un blocco all'interno della rete da programmare sul Cyclone II, che ho chiamato "modulatore_ampiezza", che prende in ingresso i PINs dell'FPGA provenienti dai primi 5 switches SW[4] … SW[0], oltre al segnale di clock di sistema, e ad ogni fronte positivo del clock rende disponibile in uscita, ovvero in ingresso al blocco VGA_MODULATOR, il segnale y_space[31:0], il cui valore (un intero) è appunto memorizzato su un registro a 32 bits.

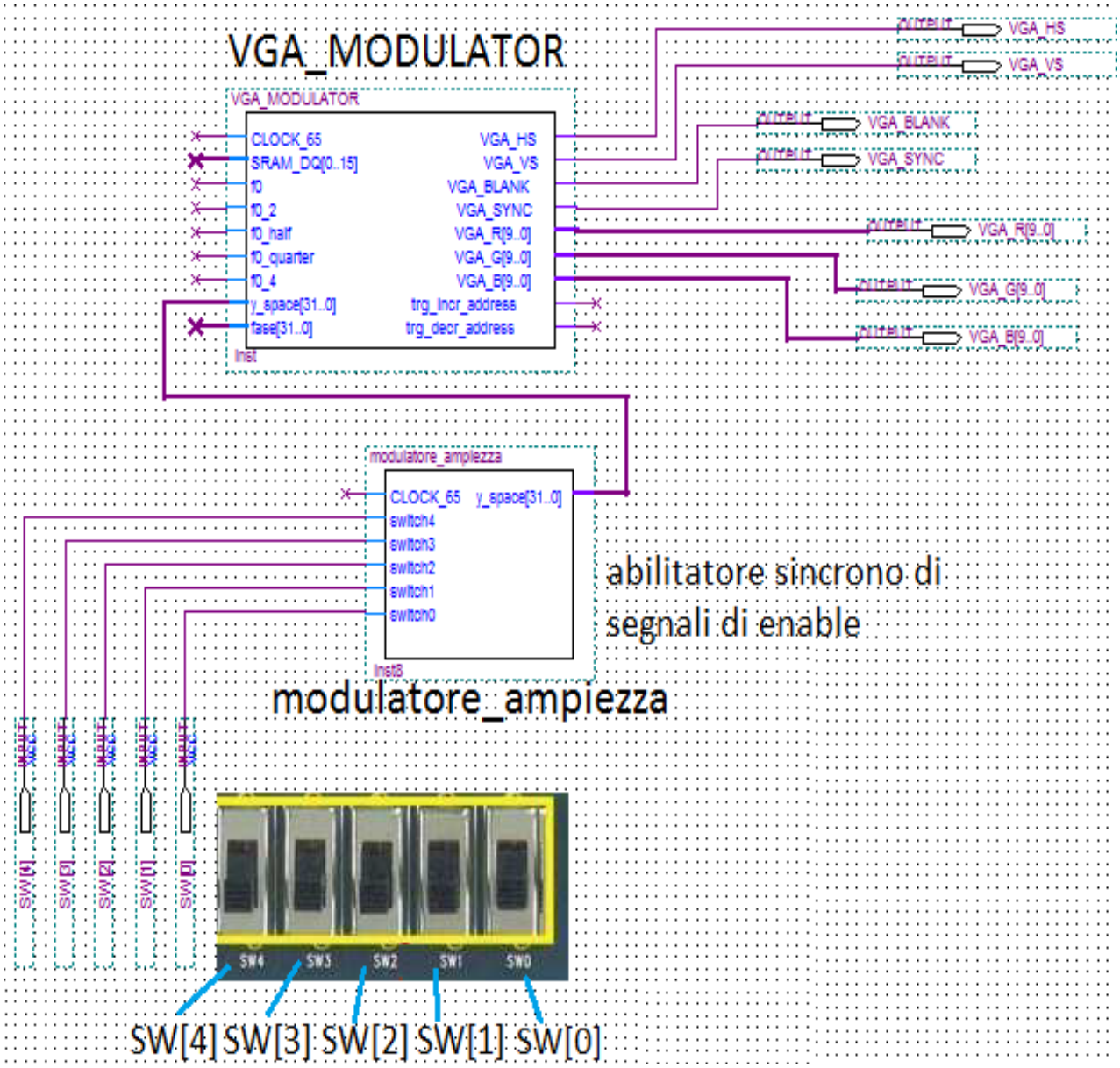


Fig. 45

Porzione dello schematic del top-level file "DENTE_SEGA_system.bdf" raffigurante il decodificatore di abilitazione di ampiezza che impone, ad ogni colpo di clock di sistema, un certo valore all'ingresso y_space[31:0] del blocco VGA_MODULATOR

Qui di seguito riporto il codice verilog che descrive la funzionalità selezionatrice dell'abilitatore di ampiezza.

```
module modulatore_ampiezza (CLOCK_65, switch4, switch3, switch2, switch1, switch0, y_space);
// sincronizzatore di segnali (eventi) asincroni provenienti dai primi 5 SW[]
// modulatore di ampiezza

input CLOCK_65, switch4, switch3, switch2, switch1, switch0;

output [31:0] y_space; // 32 bit, un integer

integer REG_y_space;

assign y_space = REG_y_space;

always@(CLOCK_65)

begin
// SW[] DOWN => pin del cycloneII messo a GND => situazione di default
// SW[] UP => pin del cycloneII messo a Vcc

if( (switch4 == 0) && (switch3 == 0) && (switch2 == 0) && (switch1 == 0) && (switch0 == 0) )
  REG_y_space <= 0; // ampiezza di default = 63 righe // ampiezza A
else if( (switch4 == 0) && (switch3 == 0) && (switch2 == 0) && (switch1 == 0) && (switch0 == 1) )
  REG_y_space <= 1; // ampiezza = 63 * 2 = 126 righe // ampiezza 2A
else if( (switch4 == 0) && (switch3 == 0) && (switch2 == 0) && (switch1 == 1) && (switch0 == 1) )
  REG_y_space <= 2; // ampiezza = 63 * 3 = 189 righe // ampiezza 3A
else if( (switch4 == 0) && (switch3 == 0) && (switch2 == 1) && (switch1 == 1) && (switch0 == 1) )
  REG_y_space <= 3; // ampiezza = 63 * 4 = 252 righe // ampiezza 4A
else if( (switch4 == 0) && (switch3 == 1) && (switch2 == 1) && (switch1 == 1) && (switch0 == 1) )
  REG_y_space <= 4; // ampiezza = 63 * 5 = 315 righe // ampiezza 5A
else if( (switch4 == 1) && (switch3 == 1) && (switch2 == 1) && (switch1 == 1) && (switch0 == 1) )
  REG_y_space <= 5; // ampiezza = 63 * 6 = 378 righe // ampiezza 6A
else
  REG_y_space <= 0;

end

endmodule
```

Fig. 46

Descrizione in codice verilog del blocco "modulatore_ampiezza"

Questo blocco non fa altro che rilevare ad ogni fronte in salita del clock di sistema l'azione asincrona dell'utente, che commuta gli switches come spiegato sopra, quindi valuta il valore da assegnare all'uscita y_space in base all'azione asincrona di switching appena rivelata (cioè in base al ramo if…else if in cui siamo entrati all'atto dell'attivazione dell'always@ interno al codice sopra riportato), dopodiché l'intero y_space è reso disponibile all'ingresso di VGA_MODULATOR.

Il concetto che sta alla base del funzionamento di questa modulazione di ampiezza è che il blocco modulatore_ampiezza deve fornire, sincronamente, un numero intero (0,1,2,3,4,5) pari al numero di righe consecutive lungo le quali ciascuno dei 63 offsets, del dente di sega f0_4, deve essere riportato, visualizzato, in aggiunta alla singola riga sulla quale, per default, quel particolare offset determina la distanza dei punti blu dai pixels-cardine. Se l'utente, ad esempio, commuta DOWN -> UP SW[0], allora modulatore_ampiezza fornisce l'intero "1" a VGA_MODULATOR, quindi quest'ultimo capisce, al ciclo di clock successivo, che ciascun dato SRAM_DQ[0:15] a lui in ingresso, puntato dal memoryMANAGER, cioè ciascun offset, deve essere visualizzato su monitor non una sola volta, ciascuno cioè sulla propria riga di competenza (63 offsets -> 63 righe => ampiezza di default), bensì su una riga in più, ossia anche sulla riga successiva. Pertanto il VGA_MODULATOR dovrà terminare la scansione della fascia bianca superiore, ossia iniziare la scansione della semi-onda positiva, 63 righe prima, più in alto cioè, nel frame, rispetto al caso di default, così da fare in modo che il dente di sega f0_4 ad ampiezza doppia (63 + 63 righe) intersechi l'asse rosso negli stessi punti di prima, volendo noi una modulazione di ampiezza che non modifichi fase e frequenza in modo spurio. Se l'utente commuta DOWN -> UP anche SW[1], dopo SW[0], allora modulatore_ampiezza fornisce l'intero "2" a VGA_MODULATOR, quindi quest'ultimo capisce che ciascun dato SRAM_DQ[0:15] a lui in ingresso, puntato dal memoryMANAGER, cioè ciascun offset, deve essere visualizzato su monitor non una sola volta, non 2 volte (cioè una volta aggiuntiva), bensì 3 volte, ossia 2 volte aggiuntive. Ciascun offset pertanto deve determinare la distanza fra 2 punti blu, rispetto ad un certo pixel-cardine, su una certa riga, che possiamo chiamare "riga di competenza", e sulle 2 righe consecutive, così da ingannare la vista dell'utente e creare una sorta di continuità lineare che concorre alla gradevole, e in questo caso corretta, visualizzazione della forma d'onda modulata in ampiezza. Questo dente di sega avrà dunque un'ampiezza tripla rispetto a quella di default (63 + 2*(63) righe). Affinchè le intersezioni con l'asse risultino invariate rispetto a quelle fra l'asse stesso ed il dente di sega f0_4 di default, è necessario che VGA_MODULATOR termini la scansione della fascia bianca superiore 2*63 righe prima, nel frame, rispetto al caso di ampiezza di default. Se l'utente commuta DOWN -> UP SW[4], dopo aver commutato in ordine cronologico SW[0], SW[1], SW[2] e SW[3], allora modulatore_ampiezza fornisce l'intero "5" a VGA_MODULATOR, quindi quest'ultimo capisce che ciascun offset SRAM_DQ[0:15] deve essere visualizzato su monitor ben 6 volte, ossia 5 volte aggiuntive. Ciascun offset deve determinare la distanza fra 2 punti blu, rispetto ad un certo pixel-cardine, su una certa riga, che possiamo chiamare "riga di competenza", e sulle 5 righe successive, così da ingannare la vista dell'utente e creare l'effetto di continuità lineare di cui sopra. Questo dente di sega avrà pertanto un'ampiezza sestupla rispetto a quella di default (63 + 5*(63) righe). Affinchè le intersezioni con l'asse risultino invariate, è necessario che VGA_MODULATOR termini la scansione della fascia bianca superiore 5*63 righe prima, nel frame, rispetto al caso di ampiezza di

default. Riporto qui di seguito una figura qualitativamente rappresentativa dell'effetto di continuità lineare che ho implementato ai fini della modulazione di ampiezza del dente di sega f0_4.

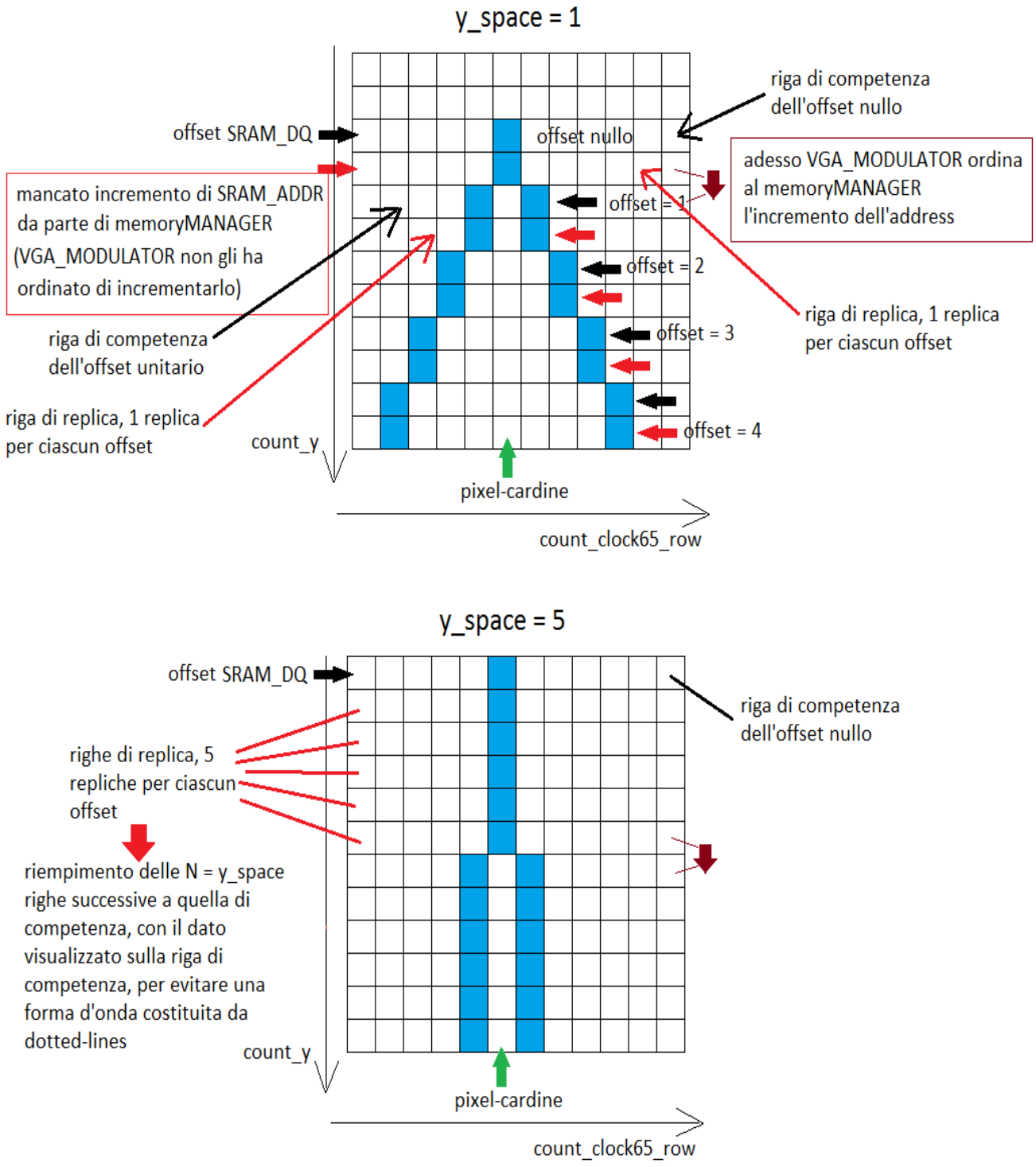


Fig. 47

Metodo grafico implementato da VGA_MODULATOR per creare l'effetto di continuità lineare delle forme d'onda modulate in ampiezza: in alto è riportata la situazione nel caso di SW[0] in stato UP, in basso invece quella in cui tutti e 5 gli SW[] lo sono (rispettivamente ampiezza doppia e sestupla di quella di default)

Qui in basso riporto un frammento significativo di codice verilog, descrivente la funzionalità di visualizzazione, da parte di VGA_MODULATOR, del dente di sega f0_4, nel quale è presente la funzione di raddoppio (SW[0] in stato UP) dell'ampiezza, da 63 a 126 righe.

```
else if(y_space == 1) // SW[0] è UP, gli altri 4 sono DOWN
begin // A = 63 * 2 = 126 righe

switchVGA <= 1;

if(ampiezza == 1 || ampiezza == 3)

begin
 REG_trg_incr_address <= 0;
 REG_trg_decr_address <= 0;
 count_x <= 0;
 count_y <= 0;
 REG_VGA_HS <= 0; REG_VGA_VS <= 0;
 count_clock65_row <= 0;
 count_row_sow <= 1;
 ampiezza <= 2;
 a <= 2;
end

else // if (ampiezza == 2)

 begin // ampiezza2
   ampiezza <= 2;

   if((count_y >= 0) && (count_y <= 5))
    begin
     REG_VGA_VS <= 0; // vertical active low sync pulse
     REG_VGA_R <= 10'b0000000000;
     REG_VGA_G <= 10'b0000000000;
     REG_VGA_B <= 10'b0000000000;
     REG_trg_incr_address <= 0;
     REG_trg_decr_address <= 0;
     a <= 2;
    end
   else // if((count_y >= 6) && (count_y <= 805))
     REG_VGA_VS <= 1;

 count_x <= count_x + 1;

if((count_x >= 0) && (count_x <= 136))
  begin
    REG_VGA_HS <= 0; // active low pulse
    REG_VGA_R <= 10'b0000000000;
    REG_VGA_G <= 10'b0000000000;
    REG_VGA_B <= 10'b0000000000;
    REG_trg_incr_address <= 0;
    REG_trg_decr_address <= 0;
    count_clock65_row <= 0;
    count_row_sow <= count_row_sow;
    a <= a;
  end
```

vi entro al fronte in salita del clock di sistema, VGA_MODULATOR valuta l'ingresso y_space e capisce che l'utente ha richiesto il raddoppio dell'ampiezza, se al clock precedente avevamo "ampiezza = 1", oppure che l'utente ha richiesto il decremento dell'ampiezza di un terzo, se al clock precedente si aveva "ampiezza = 3" : trattamento delle commutazioni progressive DOWN->UP e regressive UP->DOWN

registro interno a VGA_MODULATOR con il quale gestiamo la bidirezionalità delle commutazioni successive: modulazione crescente o decrescente dell'ampiezza

al primo fronte in salita del clock successivo alla richiesta di cambiamento di ampiezza, i registri vanno tutti reinizializzati

al successivo fronte positivo del clock si entra qui

impulso basso del sincronismo verticale

inizia così la banda di guardia "upper blank porch"

impulso basso del sincronismo orizzontale

```
else if((count_x >= 137) && (count_x <= 296))
  begin
   REG_VGA_HS <= 1;
   REG_VGA_R <= 10'b0000000000; // left blank porch
   REG_VGA_G <= 10'b0000000000;
   REG_VGA_B <= 10'b0000000000;
   REG_trg_incr_address <= 0;
   REG_trg_decr_address <= 0;
   count_clock65_row <= 0;
   count_row_sow <= count_row_sow;
   a <= a;
  end
else if((count_x >= 297) && (count_x <= 1320))
   begin // c => la stampa su monitor dipende da count_y corrente
     REG_VGA_HS <= 1;
       if((count_y >= 0) && (count_y <= 5))
        begin // REG_VGA_VS active-low pulse
          REG_VGA_R <= 10'b0000000000;
          REG_VGA_G <= 10'b0000000000;
          REG_VGA_B <= 10'b0000000000;
        end
       else if((count_y >= 6) && (count_y <= 34))
        begin // upper blank porch
          REG_VGA_R <= 10'b0000000000;
          REG_VGA_G <= 10'b0000000000;
          REG_VGA_B <= 10'b0000000000;
          REG_trg_incr_address <= 0;
          REG_trg_decr_address <= 0;
          count_clock65_row <= 0;
          count_row_sow <= 1;
          a <= 2;
        end
      // y = 35 => prima riga NON blank (interamente bianca)
       else if((count_y >= 35) && (count_y <= 291))
        begin // fascia bianca sopra il dente di sega
          REG_VGA_R <= 10'b1111111111;
          REG_VGA_G <= 10'b1111111111;
          REG_VGA_B <= 10'b1111111111;
          REG_trg_incr_address <= 0;
          REG_trg_decr_address <= 0;
          count_clock65_row <= 0;
          count_row_sow <= 1;
          a <= 2;
        end
```

banda di guardia "left blank porch"

i 1024 pixels colorabili, se count_y punta dentro alla parte attiva del frame

impulso basso del sincronismo verticale

banda di guardia "upper blank porch"

fascia bianca superiore

```
else if((count_y >= 292) && (count_y <= 417))
 begin //dente di sega : semi-onda positiva

  REG_trg_decr_address <= 0;

  if(count_clock65_row == 0)
   begin
    if(a == 2)
    begin
     REG_trg_incr_address <= 1; // REG_trg_incr_address: 0->1
     a <= 1;
    end
    else // if(a == 1)
    begin
     REG_trg_incr_address <= 0; // REG_trg_incr_address: 0->0 => nessun incremento dell'address => ri-sta
     a <= 2; // alla prossima riga, al suo primo pixel, ordina di incrementare l'address
    end
   count_clock65_row <= count_clock65_row + 1;
   count_row_sow <= count_row_sow; // mantenimento di count_row_sow
  end
 else if(count_clock65_row == 1023)
  begin
   REG_trg_incr_address <= 0; // REG_trg_incr_address: 1/0->0
   count_clock65_row <= 0; // count_clock65_row: 1023->0
   count_row_sow <= count_row_sow + 1;
   a <= a;
  end
 else // count_clock65_row = 1, 2, .., 1022
  begin
   count_clock65_row <= count_clock65_row + 1;
   REG_trg_incr_address <= REG_trg_incr_address; // mantenimento di REG_trg_incr_address ad 1 o 0
   count_row_sow <= count_row_sow; // mantenimento di count_row_sow
   a <= a; // mantenimento di a (2 o 1) durante la scansione della riga
  end
```

prima riga del dente di sega ad ampiezza doppia: VGA_MODULATOR ha anticipato la fine della fascia bianca superiore di N = y_space = 1 volte il modulo, espresso in numero di righe, del dente di sega f0_4 di default, cioè di 1*63 = 63 righe

al primo fronte in salita di clock durante la scansione della parte "c" di una riga, associata ad un certo offset

VGA_MODULATOR chiede al memoryMANAGER di incrementare SRAM_ADDR

riga precedente l'asse

scansione della riga di competenza dell'offset richiesto da VGA_MODULATOR, che sarà puntato dal prossimo ciclo di clock

così al primo fronte in salita del clock, durante la scansione della parte "c" della riga successiva, si entra nell'altro ramo if, quindi VGA_MODULATOR NON richiede l'incremento di SRAM_ADDR: riga aggiuntiva => visualizzazione dello stesso offset

così al primo fronte in salita del clock, durante la scansione della parte "c" della riga ancora successiva, VGA_MODULATOR riprende a chiedere l'incremento di SRAM_ADDR al memoryMANAGER

Fig. 48

Frammento significativo del codice verilog descrivente la funzionalità di visualizzazione del dente di sega f0_4 ad ampiezza doppia (SW[0] in stato UP)

Riporto, qui di seguito, delle foto raffiguranti i risultati su monitor delle modulazioni in ampiezza richiedibili, da utente, mediante i 5 commutatori SW[4]…SW[0].

Fig. 49

Ampiezza di default della frequenza “f0_4” del dente di sega (KEYs: DDDDD)

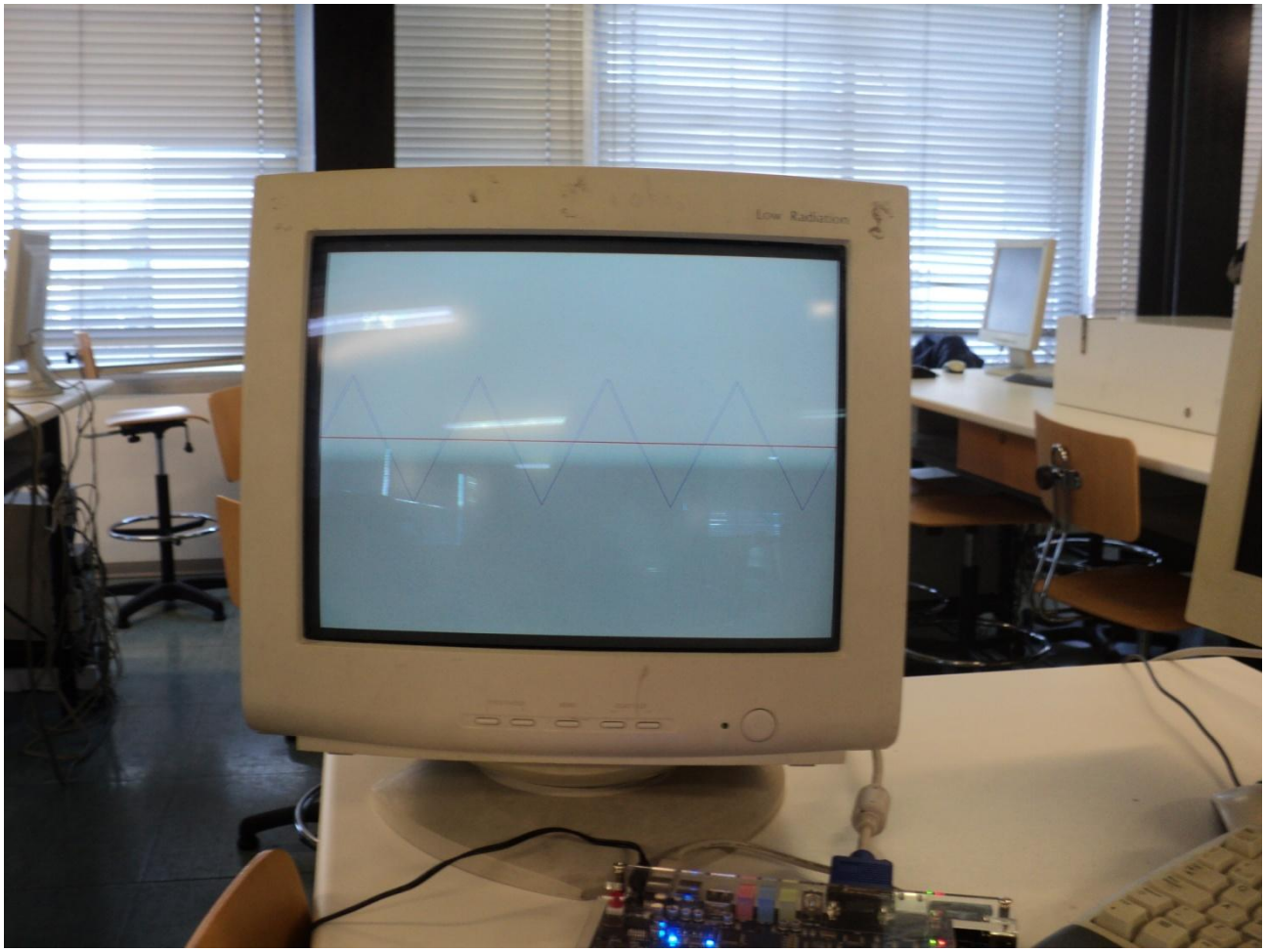


Fig. 50

Ampiezza doppia della frequenza “f0_4” del dente di sega (KEYs: DDDDU)

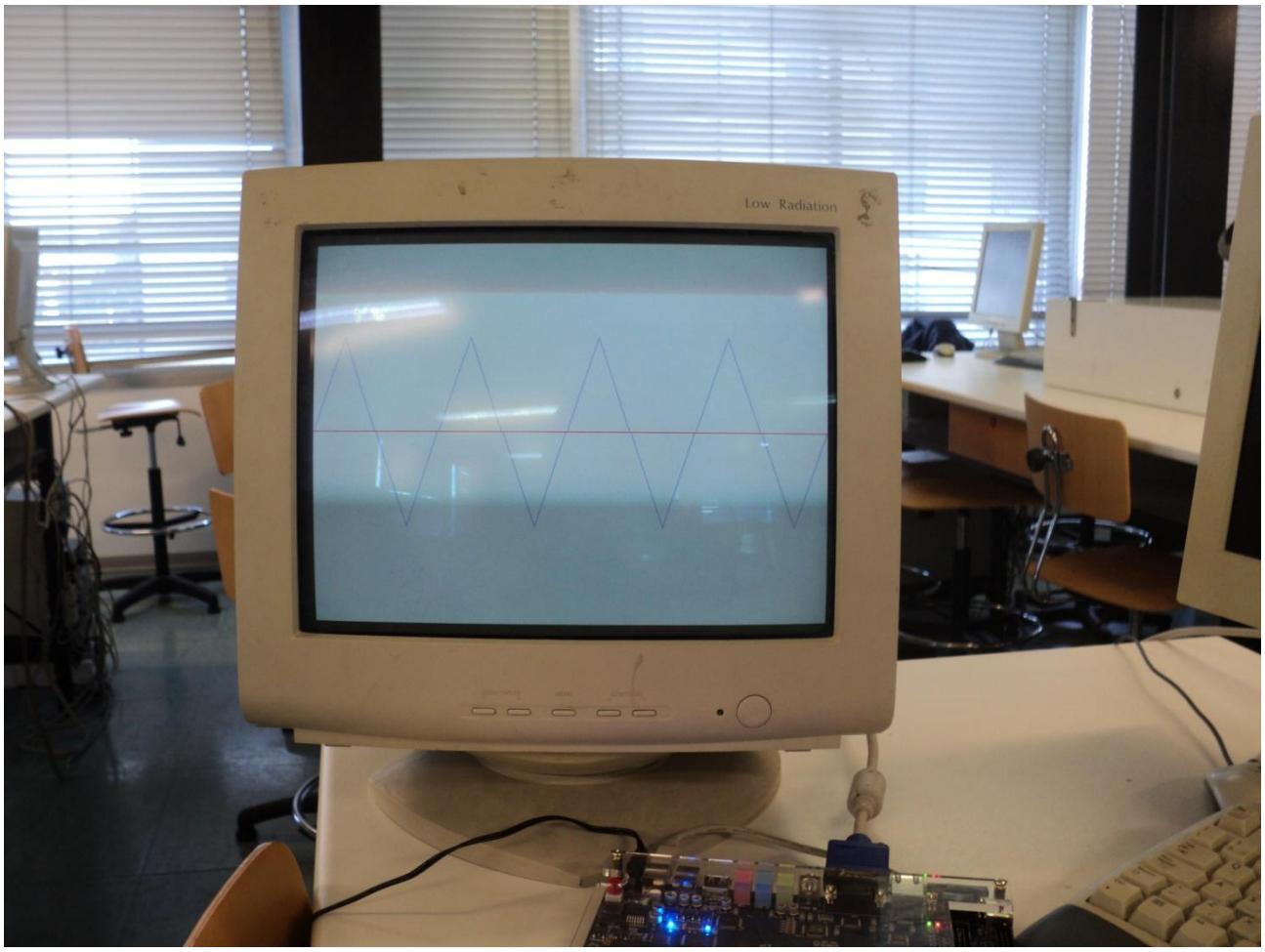


Fig. 51

Ampiezza tripla della frequenza “f0_4” del dente di sega (KEYs: DDDUU)

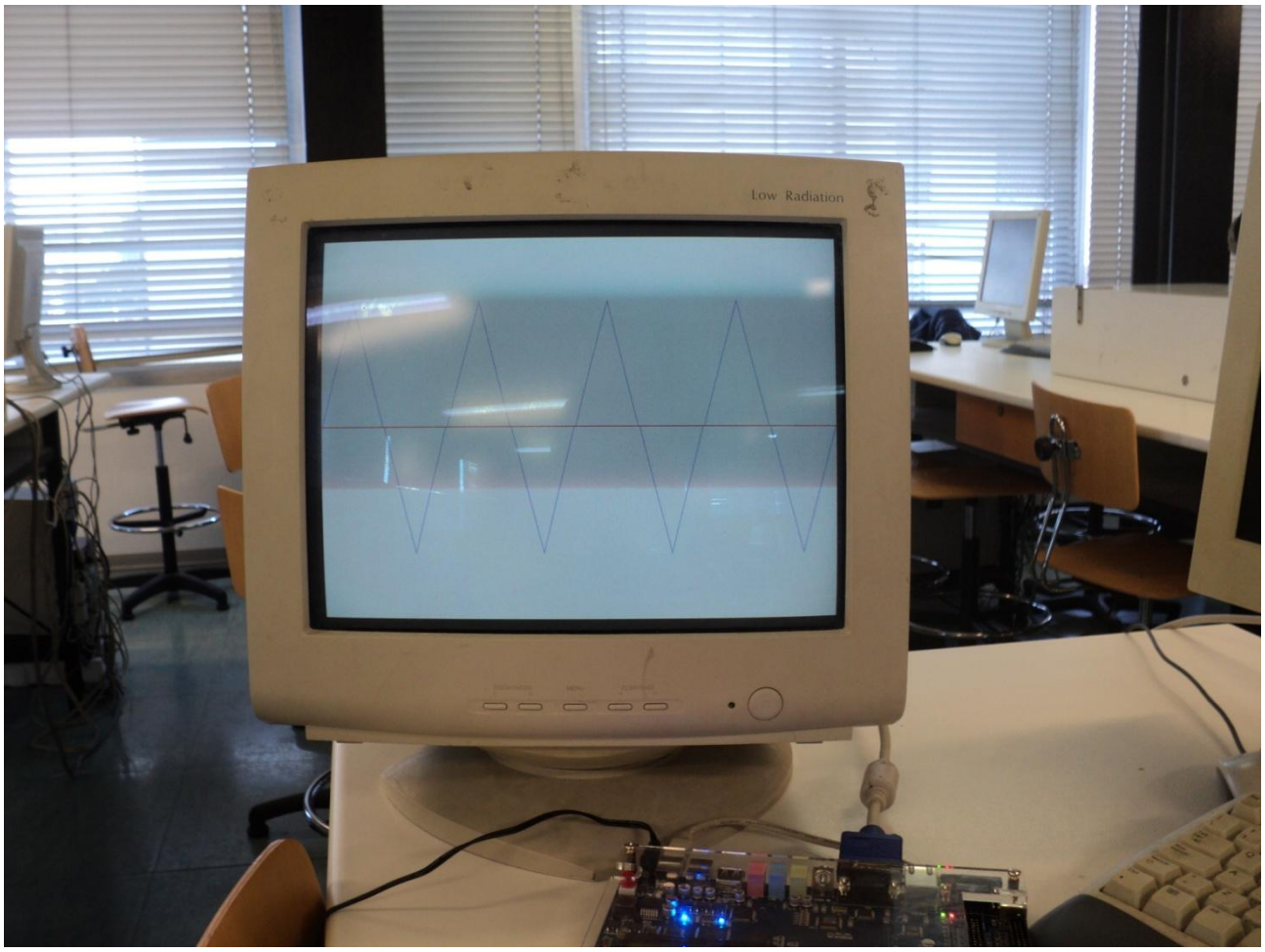


Fig. 52

Ampiezza quadrupla della frequenza “f0_4” del dente di sega (KEYs: DDUUU)

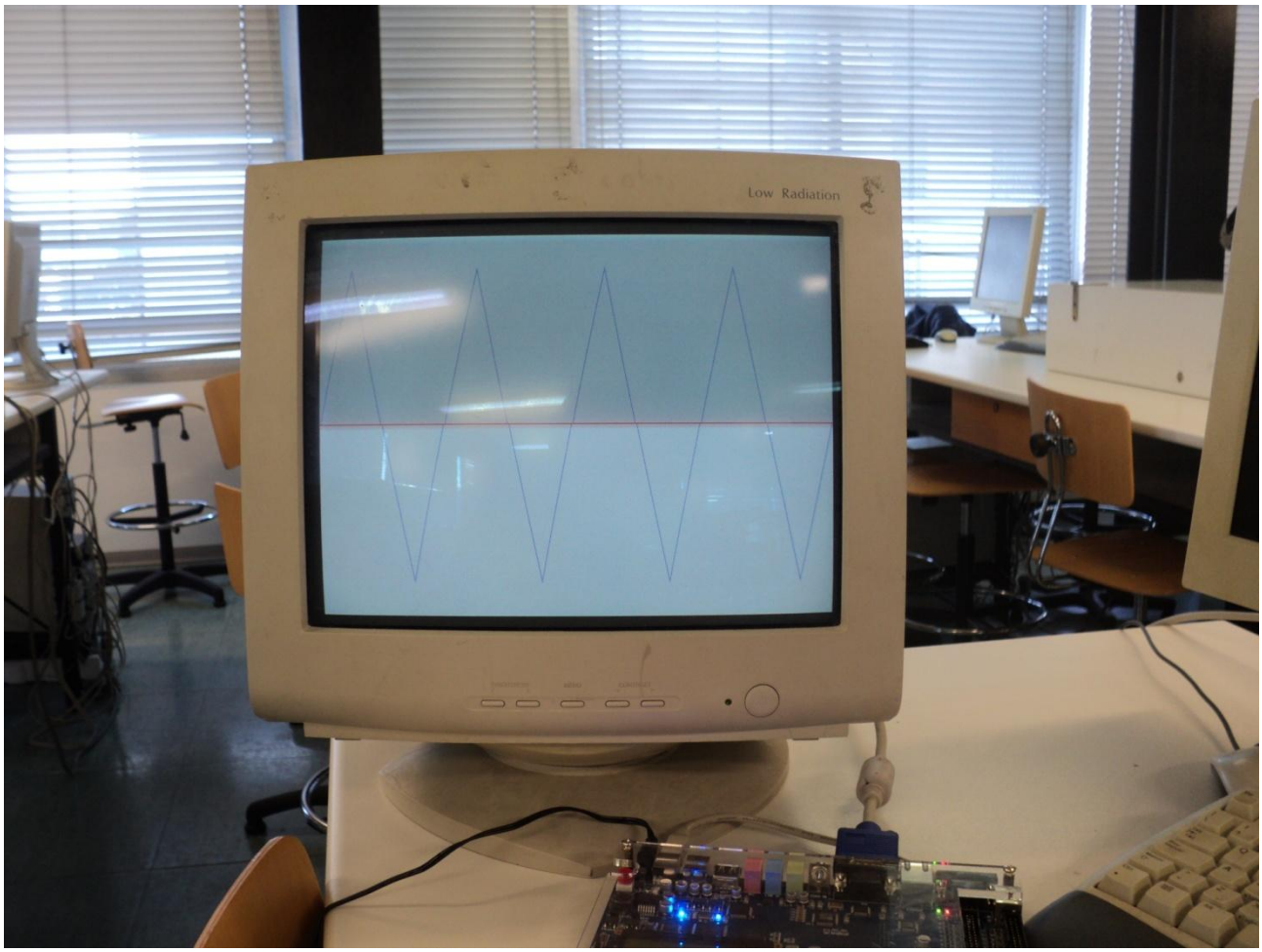


Fig. 53

Ampiezza quintupla della frequenza “f0_4” del dente di sega (KEYs: DUUUU)

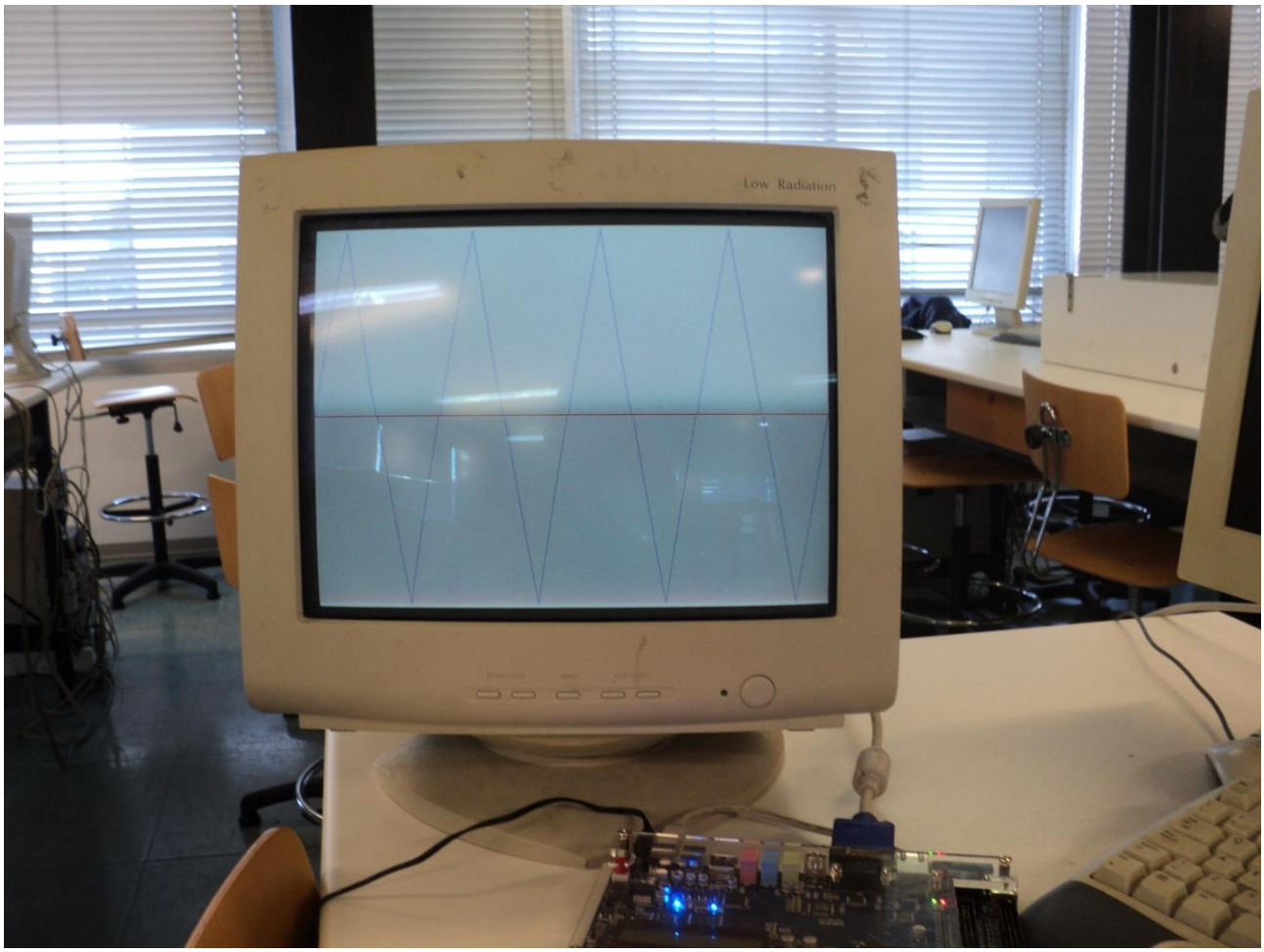


Fig. 54

Ampiezza sestupla della frequenza “f0_4” del dente di sega (KEYs: UUUUU)

6) Interfacciamento con monitor CRT

6.1) Standard VGA

Il monitor CRT (Cathod Ray Tube, tubo a raggi catodici), necessario alla visualizzazione delle forme d'onda appartenenti alla libreria del dente di sega, è stato pilotato in conformità allo standard analogico noto sotto l'acronimo di "VGA" (Video Graphics Array), introdotto sul mercato nel 1987 dalla IBM. Lo standard VGA, nato per pilotare monitors e televisori a tubo catodico, è stato adottato anche dagli attuali monitors a cristalli liquidi (LCD, Liquid Cristal Display) per una questione di compatibilità, nonostante il loro principio di funzionamento sia del tutto diverso da quello dei CRT. La modalità grafica da me scelta, ovvero la risoluzione VGA sulla quale ho adattato il codice verilog implementante la generazione dei segnali, di sincronismo e di dati, erogati dal blocco VGA_MODULATOR, è stata quella 1024x768. Tale modalità video, nota anche come standard XGA, prevede un frame visibile formato da 768 righe, scandite dal pennello elettronico da sinistra verso destra, ciascuna delle quali è costituita da 1024 pixels visibili, ovvero colorabili. La frequenza di refresh è di 60Hz, pertanto il pennello elettronico aggiorna, rinfresca, la videata corrente ben 60 volte al secondo, ovvero scandisce in modo lessicografico (sinistra -> destra, prima riga -> ultima riga) 60 frames al secondo. Affinché il monitor funzioni è necessario inviargli non solo i segnali analogici portatori dell'informazione relativa alla tonalità dei 3 colori primari, ma anche dei segnali di controllo, uno di sincronismo orizzontale e l'altro di sincronismo verticale, con temporizzazioni ben precise che variano in funzione della risoluzione VGA scelta. Il segnale di sincronismo orizzontale fornisce un impulso basso per indicare quando il pennello elettronico deve andare a capo, ovvero per indicare la fine della scansione di una riga e l'imminente inizio della scansione della successiva (il pennello ritorna così all'estremità sinistra del monitor). Il segnale di sincronismo verticale fornisce un impulso basso per indicare la fine della scansione di un frame e l'imminente inizio della scansione del successivo; in altri termini dà un impulso basso quando il pennello elettronico deve ritornare nella parte alta dello schermo. Il pennello elettronico, la cui deflessione elettrostatica, orizzontale e verticale, è modulata proprio dai 2 segnali di sincronismo da inviare al monitor, costa di 3 flussi elettronici distinti, ciascuno dei quali, pilotato dal corrispondente segnale in corrente (corrente arrivata al monitor attraverso il relativo PIN di ingresso della porta VGA montata sul retro del monitor stesso), è inviato su una particolare regione della superficie assegnata a ciascun pixel. Tale regione è ricoperta da un particolare fosforo che emette la corrispondente radiazione ottica primaria (rossa, verde o blu) se colpito dal flusso elettronico. Modulando la corrispondente corrente è possibile determinare l'intensità del flusso elettronico, così da modulare l'intensità ottica emessa dal fosforo colpito. Questa intensità luminosa va da un minimo di 0 IRE (pixel massimamente nero), quando la corrente è nulla, ossia è nullo il flusso elettronico incidente sul fosforo (quest'ultimo, per nulla eccitato, non emana alcun fotone), ad un massimo di 143 IRE, ossia la tonalità più viva, più brillante, del colore preso in considerazione; se a questo colore, caratterizzato dalla sua massima tonalità, vi mescoliamo gli altri 2 colori primari, anche loro caratterizzati dalla massima tonalità di 143 IRE, otteniamo un pixel massimamente bianco (lo sfondo delle forme d'onda a dente di sega è stato realizzato con questo tipo di bianco).

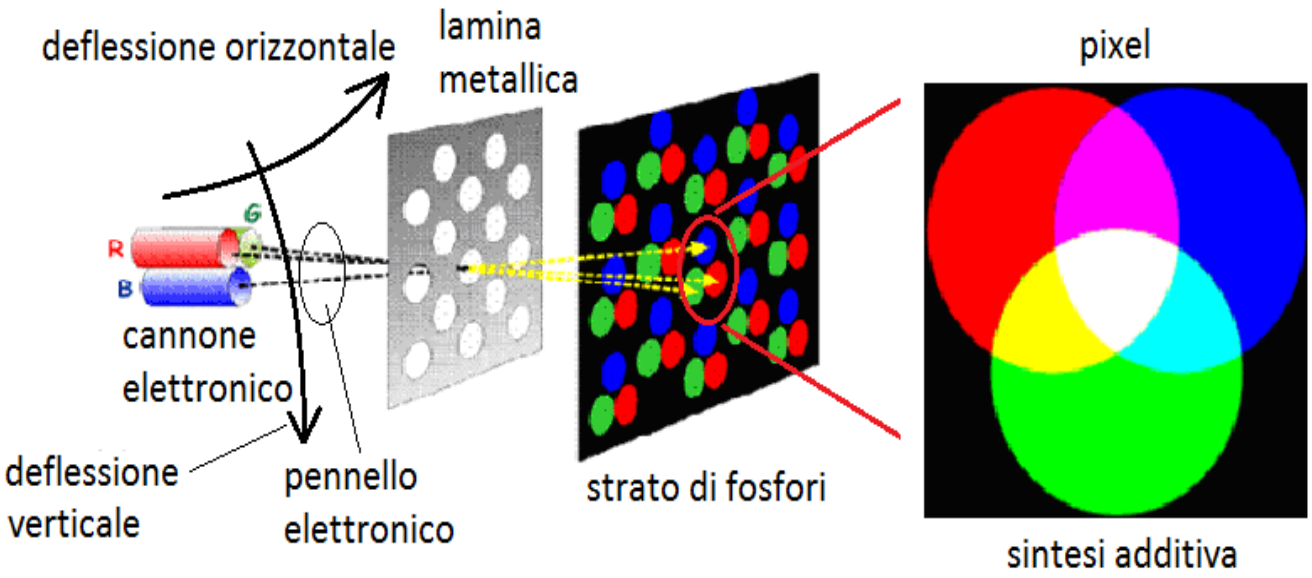
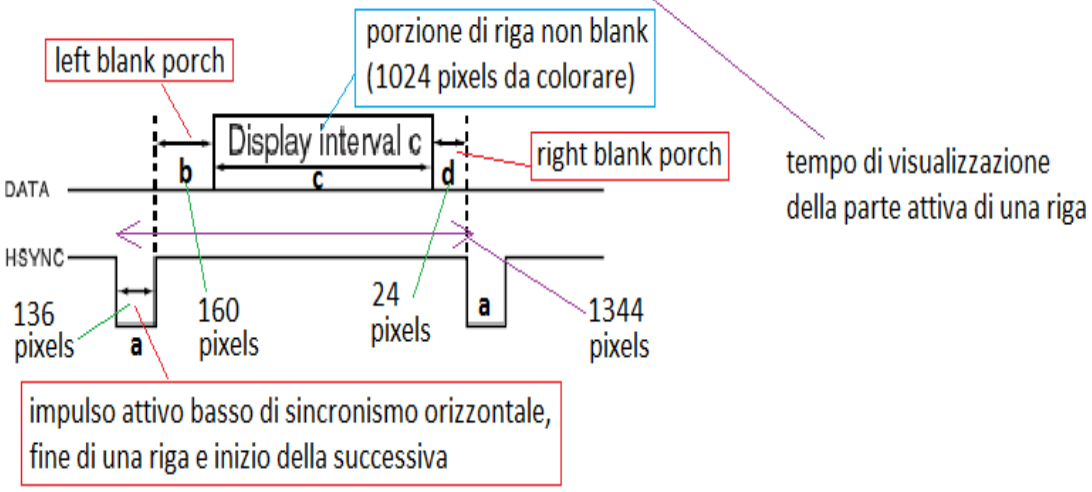


Fig. 55

Schema raffigurante il principio alla base della scansione del monitor (array di pixels) da parte del pennello elettronico

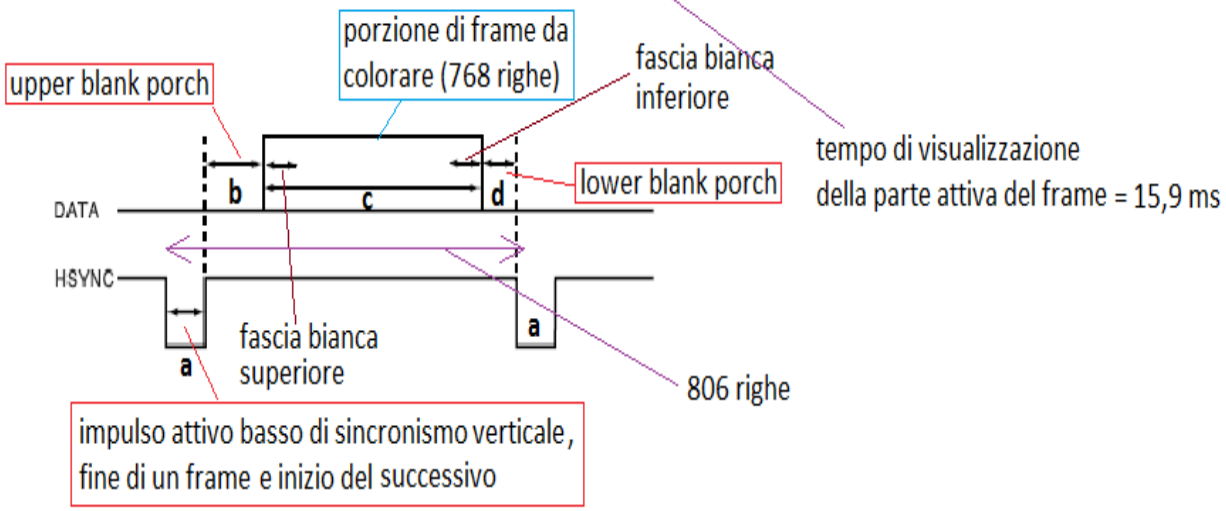
Occorre notare che il pennello elettronico deve scandire un'area più ampia di quella effettivamente visualizzata sul monitor e contenente l'immagine della forma d'onda (detta "area attiva"), poichè si devono rispettare delle bande di guardia prestabilite dalla VESA (Video Electronics Standards Association). L'area attiva misura in pixels 1024x768, aggiungendo però le bande di guardia, che ho chiamato "left blank porch", "right blank porch", "upper blank porch" e "lower blank porch", l'area effettivamente scandita misura 1344x806. Quando il pennello elettronico entra nell'area attiva del monitor i 3 segnali analogici informativi, relativi cioè ai 3 colori primari, devono portare l'informazione circa la tonalità (IRE) del colore dell'immagine, quando però il pennello scandisce le 4 bande di guardia deve essere visualizzato il "colore" blank, se così può essere definito, ossia il nero più scuro possibile, cioè ci deve essere assenza di colore (R=0, G=0, B=0). Avendo scelto la modalità 1024x768@60Hz, ho ottenuto che il tempo necessario per scandire completamente una schermata, comprendendo sia l'area attiva che le 4 bande di guardia, è di circa 16,7 ms (infatti 16,7 ms x 60 = 1s) e che la frequenza di pixel corrispondente (la frequenza di pixel, detta anche "dot rate", è legata alla risoluzione e alla frequenza di refresh dello schermo) è di 65 MHz, ovvero in un secondo il pennello elettronico scandisce, lungo l'array video bidimensionale, 65 milioni di pixels. Qui di seguito riporto le caratteristiche temporali dei segnali di sincronismo orizzontale e verticale da inviare in ingresso al terminale CRT, in conformità allo standard analogico XGA.

tipo di standard VGA	specifiche temporali del sincronismo orizzontale (µs)				pixel-clock
XGA(60Hz) 1024x768	a = 2,1	b = 2,5	c = 15,8	d = 0,4	65MHz



tempo di scansione dell'intera riga (parte attiva + banda di guardia) = 20,7 µs

tipo di standard VGA	specifiche temporali del sincronismo verticale (N°righe)				pixel-clock
XGA(60Hz) 1024x768	a = 6	b = 29	c = 768	d = 3	65MHz



tempo di scansione dell'intero frame (regione attiva + bande di guardia) = 16,7ms

Fig. 56

Segnali di sincronismo orizzontale e verticale da inviare al monitor CRT, in conformità allo standard analogico XGA

6.2) Componentistica on-board per interfacciamento XGA

Il monitor CRT ha bisogno di ricevere in ingresso, sulla porta VGA montata sul suo pannello posteriore, ciascuno sul PIN di ricezione opportuno, 3 segnali analogici costituenti i dati, cioè l'informazione RGB, oltre ad un segnale di sincronismo orizzontale e verticale per pilotare la deflessione del pennello elettronico. Pertanto la scheda DE2 dispone di un connettore VGA D-SUB a 16 PINs per consentire l'invio, al terminale video CRT, dei segnali di output sopra citati. Questi segnali devono essere generati, pertanto è necessario prevedere una logica, un'intelligenza, che eroghi i segnali di sincronismo VGA_HS e VGA_VS, da inviare ai PINs 13 e 14 del connettore VGA, e i segnali analogici informativi RGB, che chiameremo IOR, IOG e IOB (3 segnali in corrente, in uscita dalla scheda ed entranti, per mezzo di 3 PINs del connettore D-SUB VGA, nel monitor CRT), da inviare ai PINs 1, 2 e 3 del suddetto connettore. Il modulo VGA_MODULATOR costituisce la parte dell'elemento programmabile FPGA che si occupa, una volta fornita l'alimentazione alla scheda e programmato via usb-blaster lo stesso Cyclone II, di generare i segnali che sincronismo che vanno ad insistere direttamente sui PINs 13 e 14 del connettore di output VGA. VGA_MODULATOR, inoltre, genera ad ogni colpo di clock di sistema delle stringhe di bits, che chiameremo VGA_R[9:0], VGA_G[9:0] e VGA_B[9:0] che, una volta decodificate da un convertitore digitale -> analogico, il DAC ADV7123 della Analog Devices, costituiscono i segnali analogici IOR,G,B portatori dell'informazione relativa alla tonalità dei 3 colori primari.

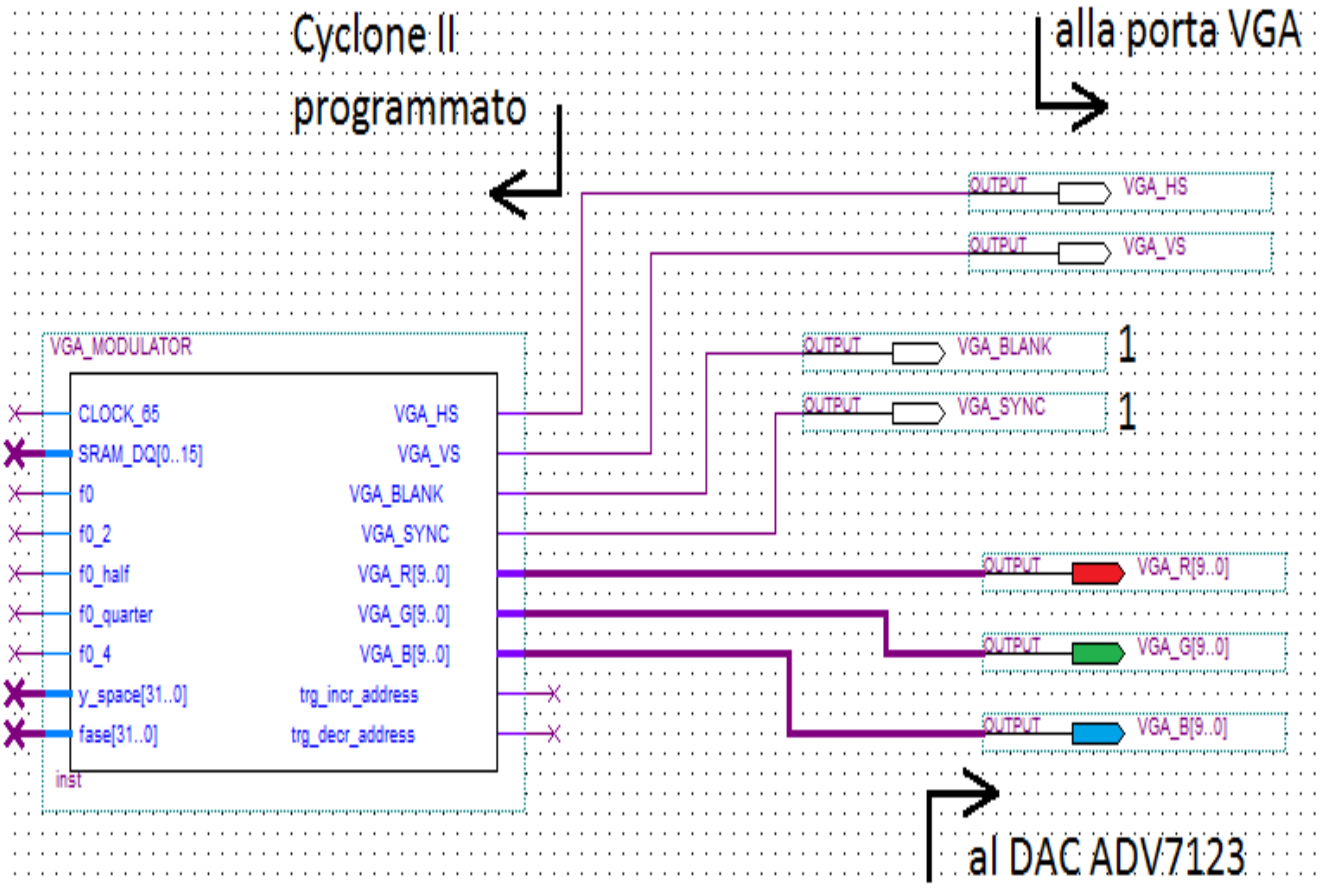


Fig. 57

Porzione dello schematic del top-level file “DENTE_SEGA_system.bdf” raffigurante il blocco VGA_MODULATOR

La frequenza del clock di sistema è stata scelta pari a 65 MHz proprio per rispettare la frequenza di clock che deve essere portata al DAC ADV7123 per consentirgli di sincronizzare correttamente la modulazione, nel tempo, delle 3 correnti di uscita RGB, pixel dopo pixel, con la scansione XGA dello schermo da parte del pennello elettronico, che simultaneamente avviene all’interno del monitor CRT. Il segnale di clock VGA_CLK a 65 MHz, ovvero il dot rate adatto allo standard XGA, è stata ottenuto prevedendo la programmazione di uno dei 4 anelli ad aggancio di fase (PLL, Phase Lock Loop) già pronti in hardware, e quindi ottimizzati, all’interno del Cyclone II. Tale PLL accetta in ingresso il clock prodotto dall’oscillatore quarzato a 50 MHz, presente sulla scheda, ed eroga, inviandolo a tutte le parti del sistema che necessitano di un segnale di sincronismo, un clock alla frequenza di 65 MHz. Anche in questo caso il progettista è assistito, nell’ambiente di sviluppo Quartus II, da una mega-wizard.

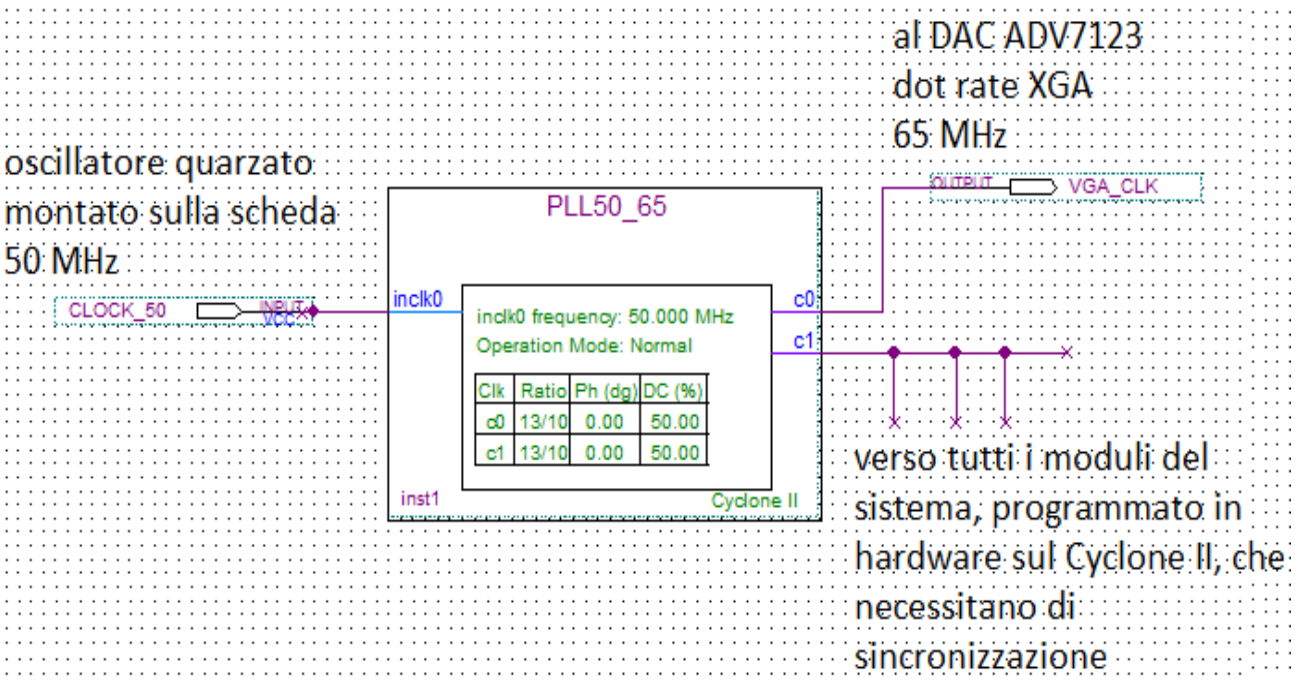


Fig. 58

PLL che eroga la frequenza di clock desiderata a partire da quella quarzata disponibile sulla scheda

Mediante la macro-wizard dedicata al PLL ho optato per una separazione in 2 canali (c0 e c1) dell’uscita di clock a 65 MHz, poiché con un’unica uscita l’ambiente di sviluppo mi segnalava, mediante un warning (benché non critico), che sulla forma d’onda del clock di sistema, quello cioè

in uscita dal PLL (CLOCK_65), si sarebbero potuti presentare dei jitters temporali che avrebbero comportato un'aleatorietà della forma d'onda stessa (una deviazione dalla forma d'onda "ideale") e quindi della tempistica, ad esempio, con la quale si presenta il fronte in salita del clock, con tutti i problemi di violazione delle regole del progetto sincrono-statico che ne conseguono. Questo a causa del fatto che (ce lo dice il suddetto warning) quell'unica uscita, che serve il segnale di clock di sistema a tutti i blocchi (VGA_MODULATOR, memoryMANAGER, ecc…) della rete da programmare sul Cyclone II, non solo va ad insistere su parti programmate dell'FPGA, ossia interne all'elemento programmato, ma va anche verso un PIN esterno al Cyclone II, ovvero il PIN di ingresso "VGA_CLK" del DAC RGB montato sulla scheda DE2. Questo PIN, abbastanza rumoroso, senz'altro molto più rumoroso degli ingressi "CLOCK_65" dei vari blocchi programmati all'interno del Cyclone II, potrebbe comportare delle fluttuazioni stocastiche dei fronti del clock di sistema, essendo il PIN "VGA_CLK" collegato agli ingressi del clock di sistema dei vari blocchi della mia rete programmata sull'FPGA, e ciò potrebbe causare gravi malfunzionamenti, legati al mancato rispetto dei tempi di setup e di hold dei flip-flop della rete. Il warning suggeriva pertanto di prevedere un canale di routing dedicato per l'ingresso VGA_CLK del DAC, pertanto ho pensato di separare il canale di routing "c0", dedicato all'instradamento del dot-rate (pixel-rate) clock verso il PIN VGA_CLK, dal canale "c1", dedicato al servizio dei blocchi della rete che necessitano del sincronismo di sistema. Tuttavia anche con questo provvedimento ho notato la permanenza del suddetto warning.

Il convertitore ADV7123 è necessario alla decodifica delle stringhe binarie generate dalla logica programmata nel blocco VGA_MODULATOR, ad ogni ciclo di clock, in valori di tensione, e quindi, grazie alle resistenze viste verso i PINs 1, 2 e 3 del connettore D-SUB VGA, in valori di corrente (IOR. IOG, IOB), che vanno poi a modulare le intensità dei rispettivi flussi del pennello elettronico nel monitor CRT. Abbiamo 3 canali di conversione digitale -> analogico, ossia 3 DACs, in ingresso a ciascuno dei quali viene presentata ormai stabile, trascorso un lasso di tempo, successivo al fronte in salita del clock VGA_CLK, pari al tco (= "clock_to_output" delay time) caratteristico dei data-registers, una parola binaria formata da 10 bits. Pertanto il set discreto di valori analogici della corrente in uscita da ciascun DAC consta di 1024 livelli, facenti parte di quella scala discreta nota sotto il nome di "Gray scale": la parola 0000000000 viene tradotta da ciascuno dei 3 DACs nel valore nullo della corrispondente corrente di uscita (di ingresso per il monitor CRT), e quindi comporterà un valore della tonalità ottica del corrispondente colore primario pari a 0 IRE (assenza di quel colore sul pixel), mentre la parola 1111111111 viene tradotta nel valore massimo della corrispondente corrente, e quindi comporterà un valore della tonalità ottica del corrispondente colore primario pari a 143 IRE (massima brillantezza di quel colore sul pixel). Riporto qui di seguito lo schema funzionale del dispositivo.

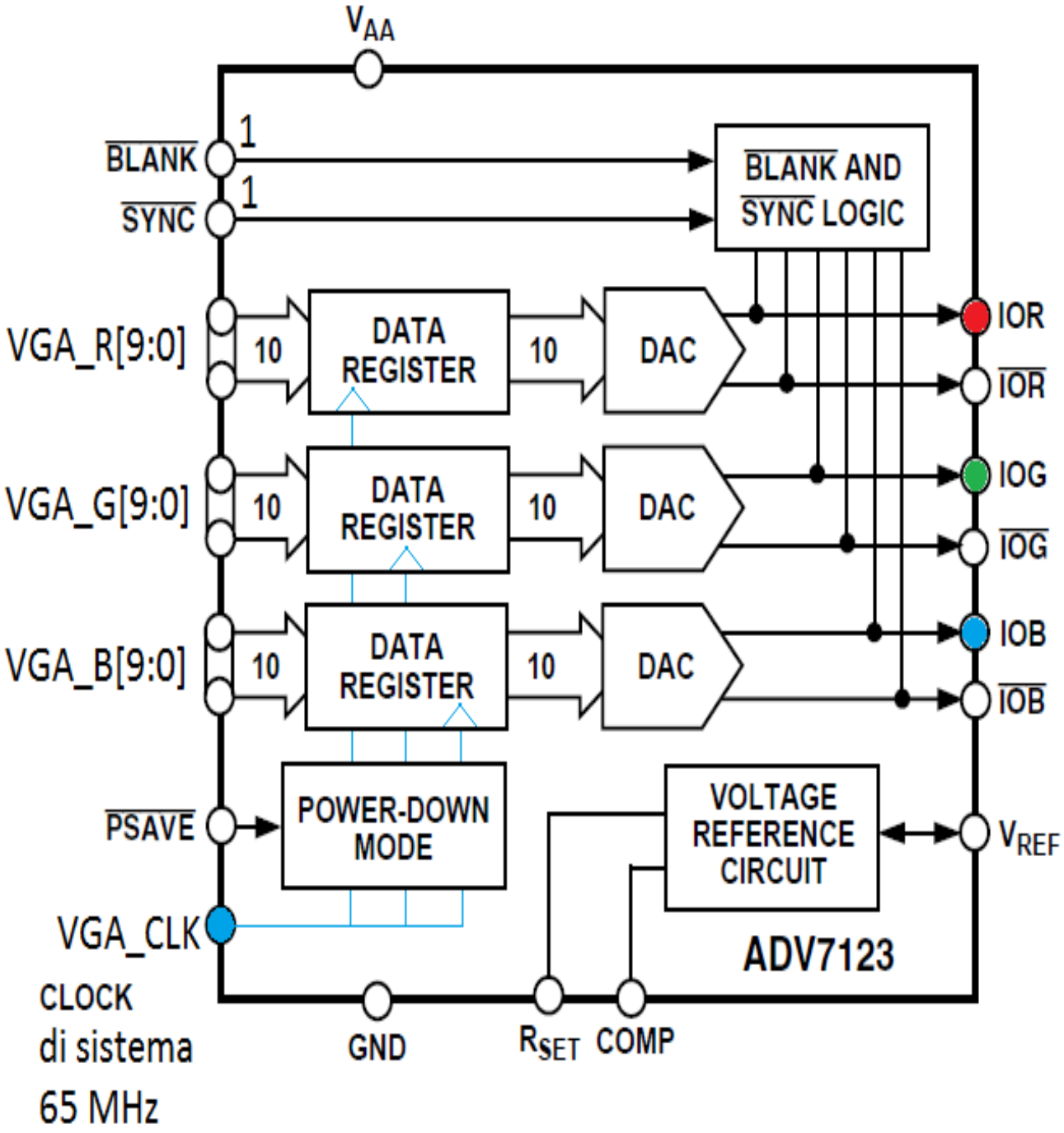


Fig. 59

Schema funzionale del DAC ADV7123 della Analog Devices

Riporto anche, nella figura seguente, lo schematic circuitale relativo a tutto l'apparato VGA montato sopra la scheda DE2, comprensivo cioè del convertitore ADV7123 e del connettore D-SUB VGA.

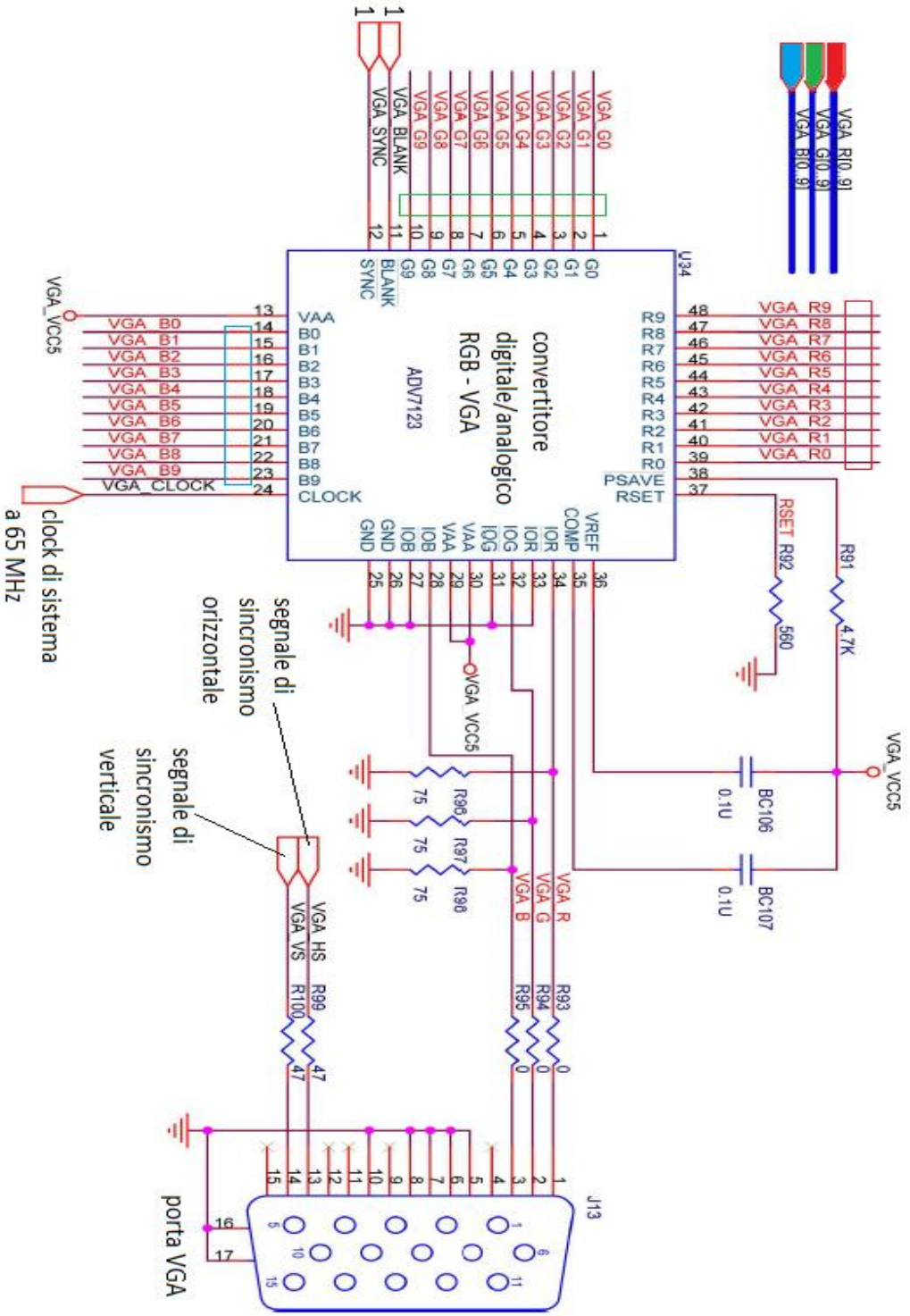


Fig. 60

Schematic circuitale dell'intero apparato VGA montato sopra la scheda DE2, comprensivo cioè del convertitore ADV7123 e del connettore D-SUB VGA

6.3) Temporizzazioni di segnali RGB da FPGA per controllo XGA del monitor

Nell'immagine seguente riporto un diagramma temporale qualitativo raffigurante le forme d'onda coinvolte nel funzionamento del DAC VGA; i tempi riportati sono relativi al caso di tensione di alimentazione del DAC pari a 5 V (cioè l'alimentazione a cui è sottoposto il dispositivo sula scheda DE2). I segnali di controllo SYNC e BLANK, per mezzo del codice verilog implementante la funzionalità del VGA_MODULATOR di generazione dei segnali, di controllo e di dati, in ingresso al DAC, li ho sempre tenuti alti, ovvero disabilitati, dal momento che per spegnere le correnti RGB, durante i lassi di tempo nei quali il pennello elettronico deve scandire le 4 bande di guardia VESA, ho optato per assegnare direttamente ai registri VGA_R[9:0], VGA_G[9:0] e VGA_B[9:0], interni a VGA_MODULATOR, il valore 10'b0000000000.

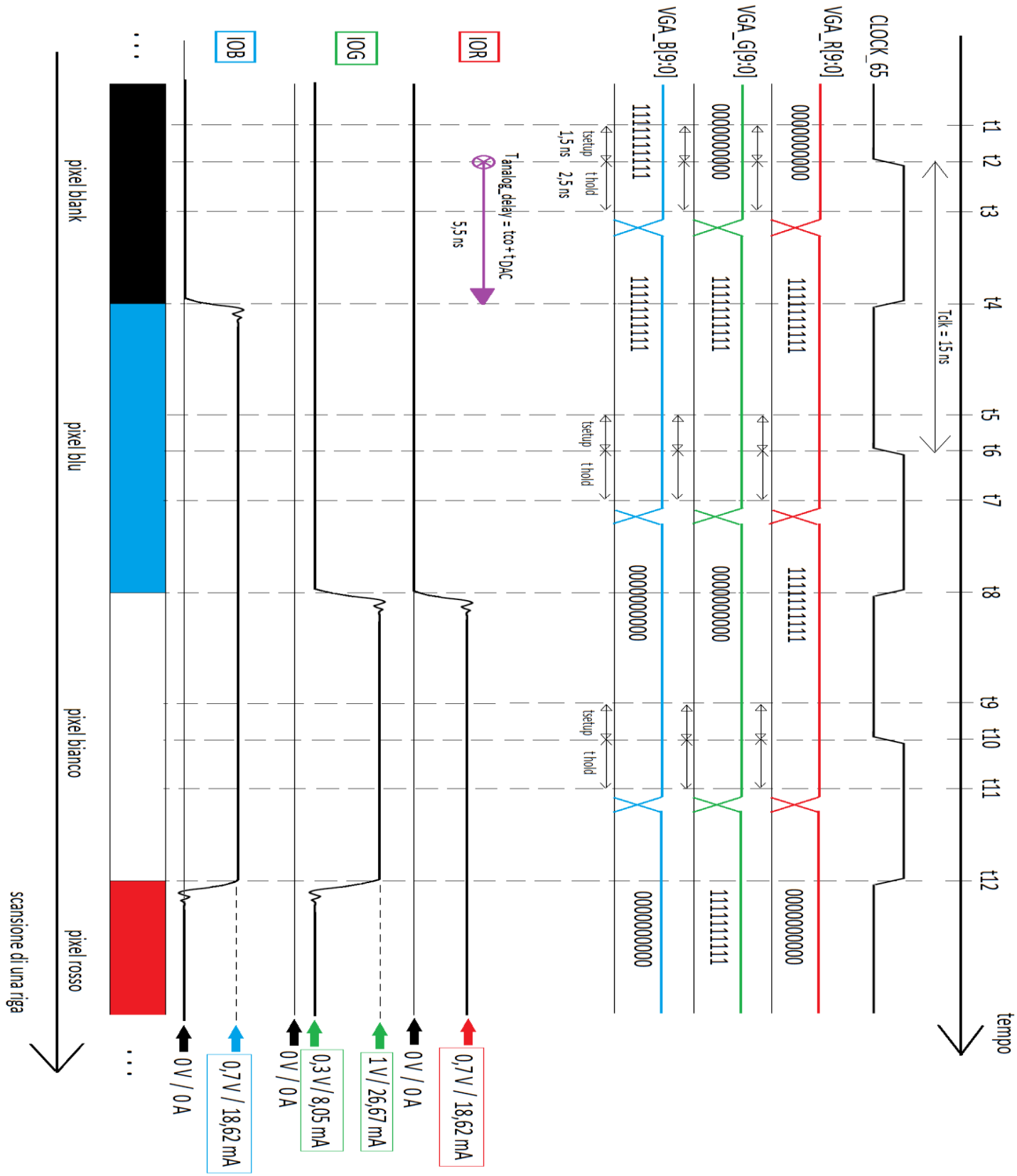


Fig. 61

Diagramma temporale qualitativo delle forme d’onda coinvolte nel funzionamento del DAC VGA

Nell'istante t2, presso il quale abbiamo un fronte in salita del clock di sistema a 65 MHz (periodo = 15 ns), si verificano 2 eventi concorrenti, simultanei, uno riguardante l'FPGA, o meglio, la sua parte chiamata VGA_MODULATOR, l'altro riguardante il dispositivo periferico a bordo della DE2 ("periferico" rispetto al Cyclone II) ADV7123, poiché entrambi sono sincronizzati dal clock di sistema, ovvero entrambi "valutano" lo stato dei loro ingressi, e conseguentemente aggiornano lo stato delle loro uscite, in corrispondenza del fronte positivo del clock di sistema: per l'ADV7123 si ha il caricamento delle 3 parole binarie, attualmente presenti sugli ingressi RGB dei 3 DACs, ai fini della loro conversione in analogico, mentre per il VGA_MODULATOR si ha l'aggiornamento delle suddette 3 parole binarie. Nell'istante t2, infatti, gli ingressi digitali RGB dei 3 DACs sono stabili (t_setup dei 3 data-registers interni all'ADV7123 rispettato), quindi il fronte positivo del clock di sistema triggera il campionamento, il latching, delle 3 parole digitali, da parte dei 3 data-registers interni all'ADV7123. Dopo un lasso di tempo pari a circa 5,5 ns, comprensivo sia del ritardo tco legato alla stabilizzazione delle 3 parole digitali appena campionate sulle 10 uscite di ciascuno dei 3 data-registers, sia del ritardo legato alla conversione digitale -> analogico dei 3 DACs in cascata ai 3 data-registers, si ha l'aggiornamento, l'adeguamento, delle 3 correnti RGB agli ingressi digitali VGA_R[9:0], VGA_G[9:0] e VGA_B[9:0] rivelati e campionati all'istante t2. Nella fattispecie la corrente IOB si alza da 0 a 18,62 mA, così da rendere blu il pixel all'interno della cui sezione circolare il pennello elettronico è in procinto di transitare. Simultaneamente nell'istante t2, tuttavia, grazie all'attivazione del costrutto always@ interno al codice verilog che descrive l'hardware del blocco VGA_MODULATOR (always@ sensibile al fronte in salita del clock di sistema), viene avviata anche una valutazione di 2 registri interni a VGA_MODULATOR, che ho chiamato count_x e count_y, il primo che ad ogni colpo di clock incrementa per consentire allo stesso VGA_MODULATOR di controllare la scansione orizzontale, il secondo che incrementa alla fine di ciascuna scansione orizzontale (quando cioè count_x = 1344) per controllare quella verticale. In base al risultato di questa duplice valutazione, attivata ad ogni colpo di clock, cioè ad ogni ingresso nell'always@ (quindi anche all'istante t2), in altre parole in base a dove si trova il pennello elettronico del monitor CRT, i valori di VGA_R[9:0], VGA_G[9:0] e VGA_B[9:0], che sono uscite per l'FPGA e ingressi per l'ADV7123, vengono aggiornati: 0000000000, 0000000000, 1111111111 -> 1111111111, 1111111111, 1111111111. In questo modo all'istante t6, cioè al successivo fronte in salita del clock di sistema, oltre ad avere il ri-aggiornamento delle 3 parole digitali, si ha anche l'inizio della conversione in analogico delle 3 stringhe di 1, la quale finirà 5,5 ns dopo, all'istante t8, con la salita verso i rispettivi valori massimi delle correnti IOR e IOG.

Qui di seguito riporto alcuni segmenti significativi di una simulazione timing raffigurante l'evoluzione, nel tempo, dei segnali di uscita prodotti dal blocco VGA_MODULATOR, oltre ad altri segnali relativi ad alcuni registri interni al blocco stesso.

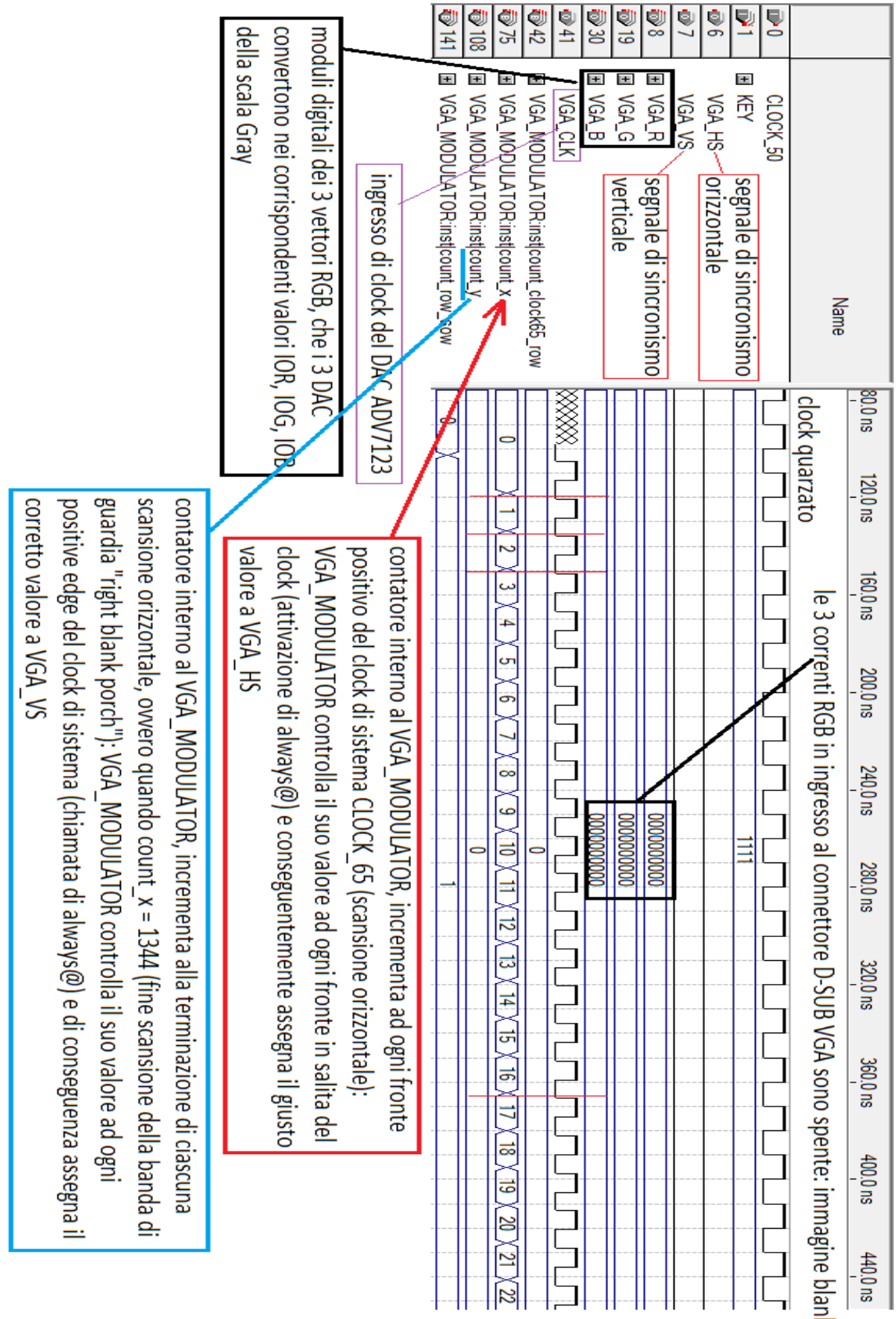


Fig. 62

Segmento di simulazione timing raffigurante il funzionamento del blocco VGA_MODULATOR allo start-up

I segnali di sincronismo VGA_HS e VGA_VS sono entrambi bassi, poichè allo start-up il VGA_MODULATOR deve generare sia l'impulso basso del sincronismo orizzontale (siamo all'inizio della prima riga) che l'impulso basso del sincronismo verticale (siamo all'inizio della visualizzazione del primo frame). Si noti, in riferimento alla figura precedente, il leggero ritardo che intercorre fra l'inizio dell'incremento (ossia aggiornamento) del segnale count_x, che regola la scansione orizzontale, e il fronte in salita del segnale VGA_CLK, ossia il segnale di sincronismo che dall'uscita c0 del PLL arriva fino all'ingresso di clock del DAC ADV7123. Il segnale di clock di sistema CLOCK_65, infatti, è ovviamente in leggero anticipo, e non in ritardo, rispetto all'istante di inizio della commutazione di count_x: il ritardo di VGA_CLK è dovuto alla elevata capacità vista dall'FPGA verso il percorso di routing che, uscendo dall'elemento programmabile, si dirama sulla scheda fino al PIN di clock del dispositivo periferico DAC-VGA.

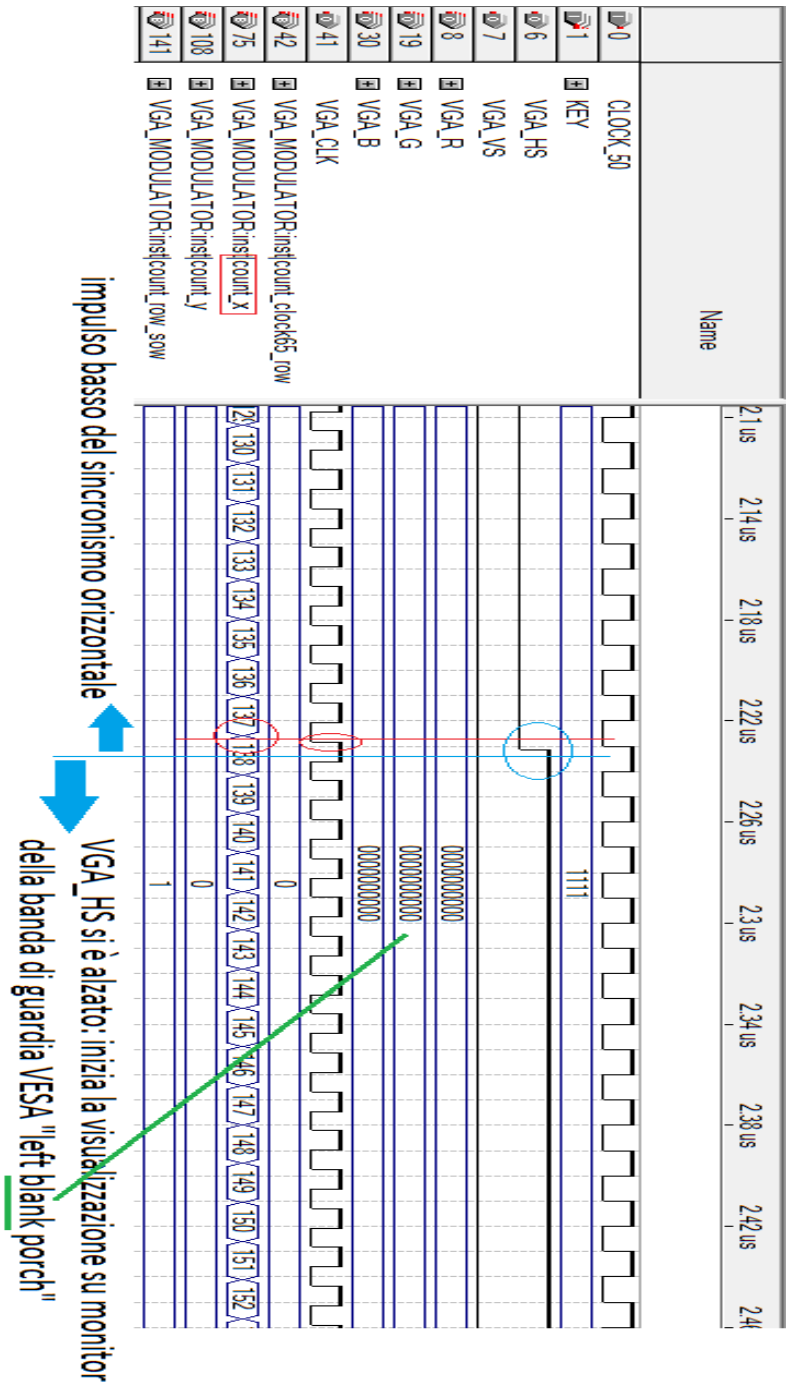


Fig. 63

Segmento di simulazione timing raffigurante il lasso di tempo, intorno ai 2,2 µs dallo start-up, nel quale si verifica l'innalzamento ad 1 di VGA_HS, da parte di VGA_MODULATOR (fine dell'impulso basso di sincronismo orizzontale)

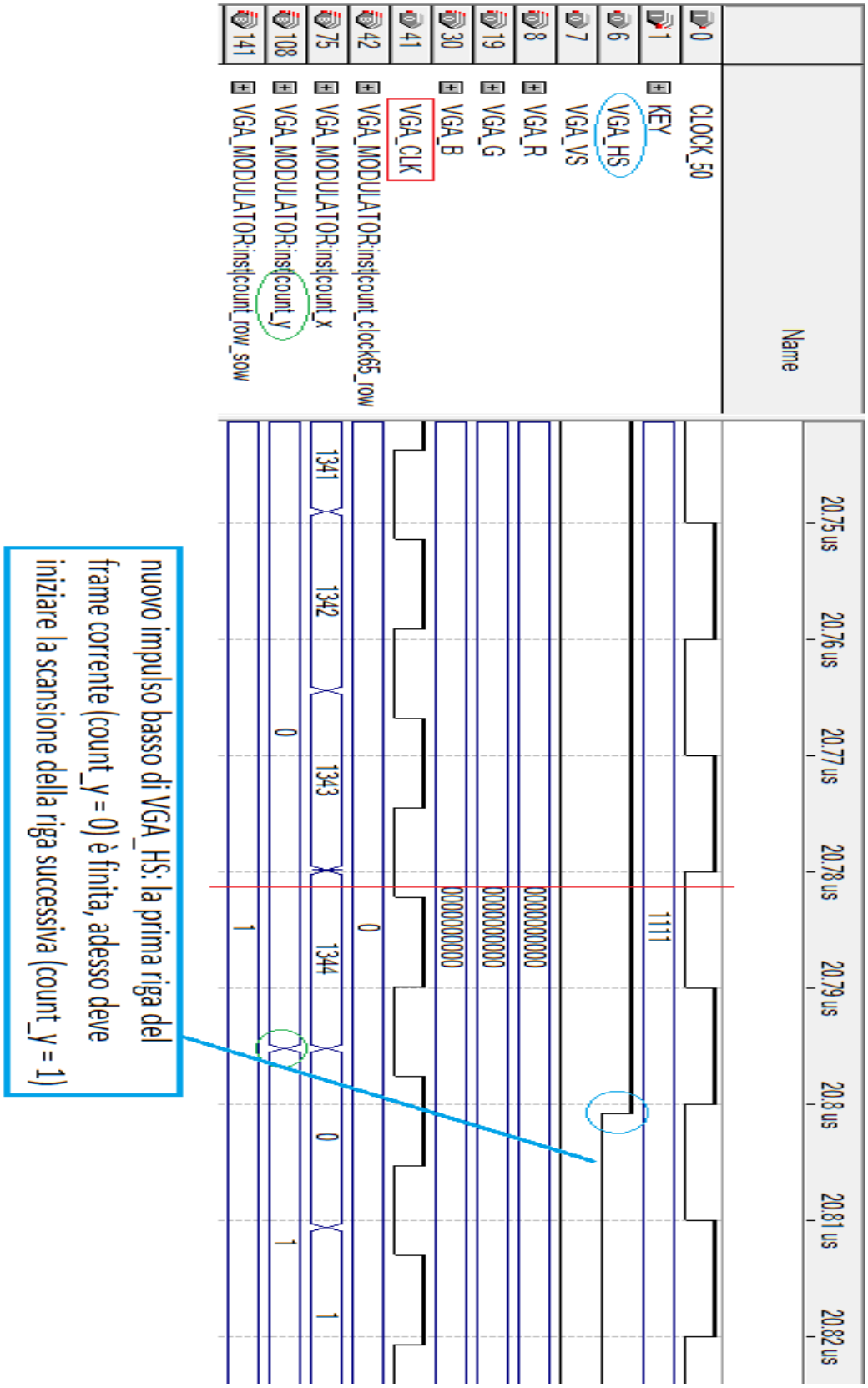


Fig. 64

Segmento di simulazione timing raffigurante il lasso di tempo, intorno ai 20,8 µs dallo start-up, nel quale si verifica il riabbassamento a 0 di VGA_HS, da parte di VGA_MODULATOR: siamo alla fine della banda di guardia “right blank porch”, cioè alla fine della scansione della prima riga “count_y = 0”, ed il successivo nuovo impulso basso del sincronismo orizzontale impone al pennello elettronico il veloce ritorno sul bordo sinistro del monitor per servire l’imminente inizio della scansione della seconda riga “count_y = 1”

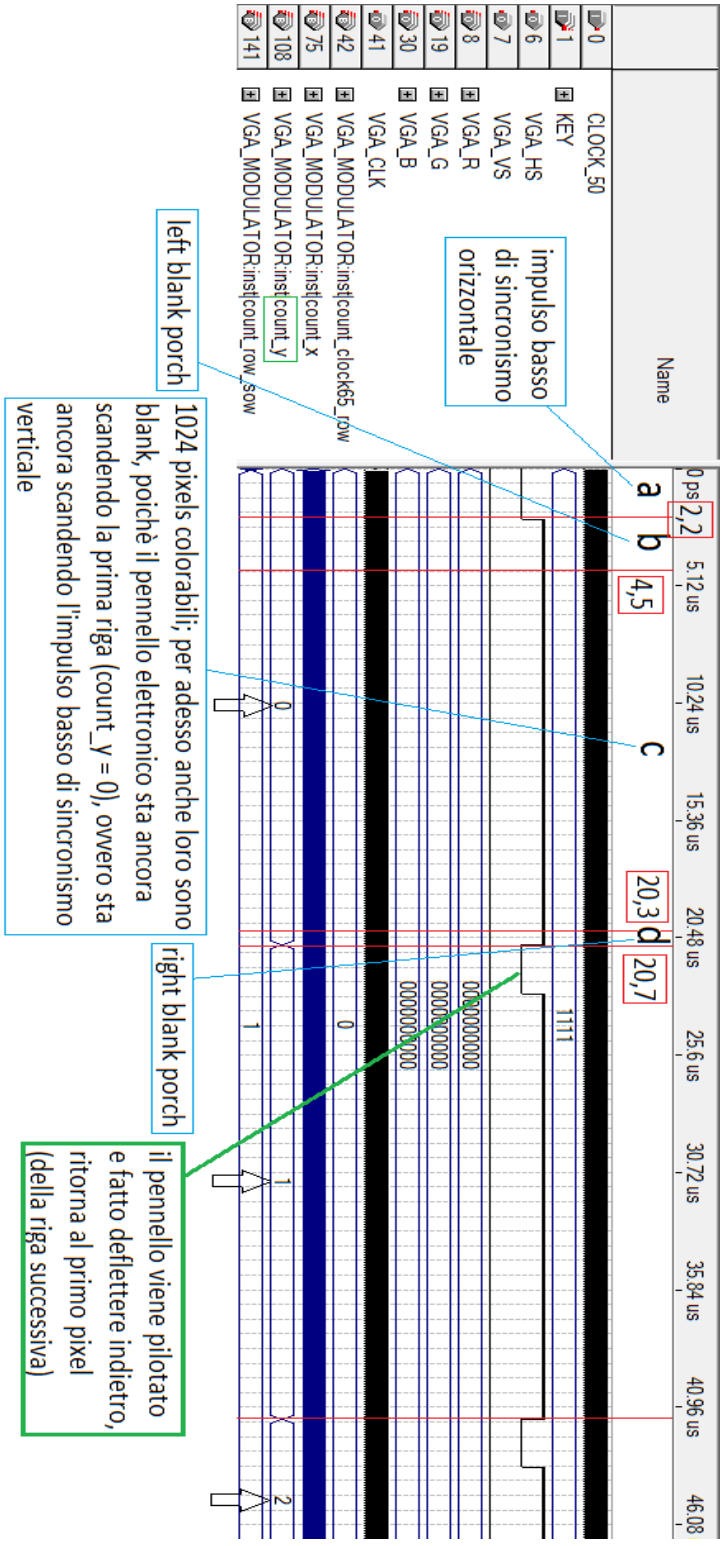


Fig. 65

Segmento di simulazione timing raffigurante il lasso di tempo durante il quale si verifica la scansione delle prime 2 righe (count_y = 0 e count_y = 1), nonché l'inizio della scansione della terza (count_y = 2)

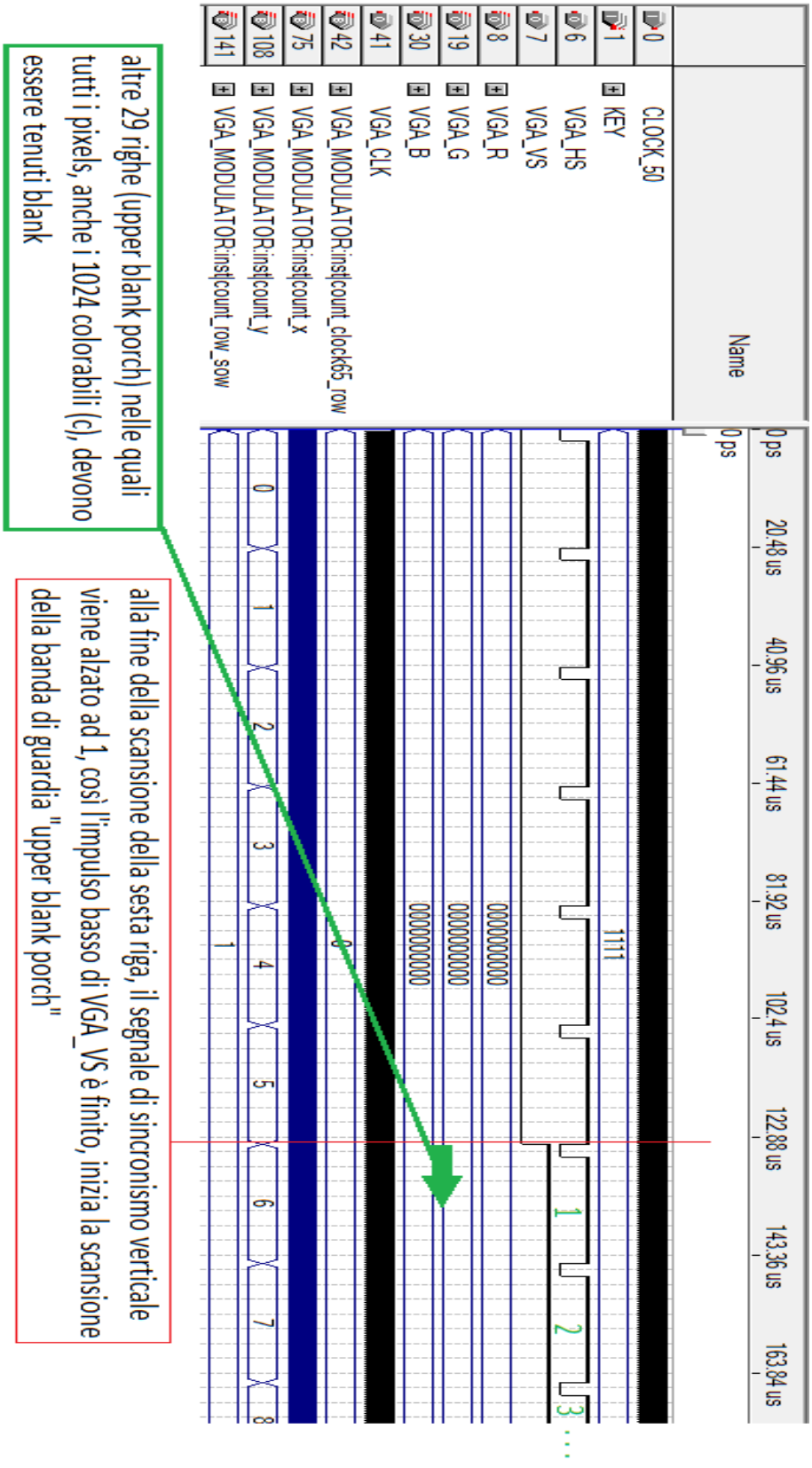


Fig. 66

Segmento di simulazione timing raffigurante l'innalzamento ad 1 del segnale di sincronismo verticale (inizio della scansione della prima delle 29 righe della banda di guardia VESA "upper blank porch")

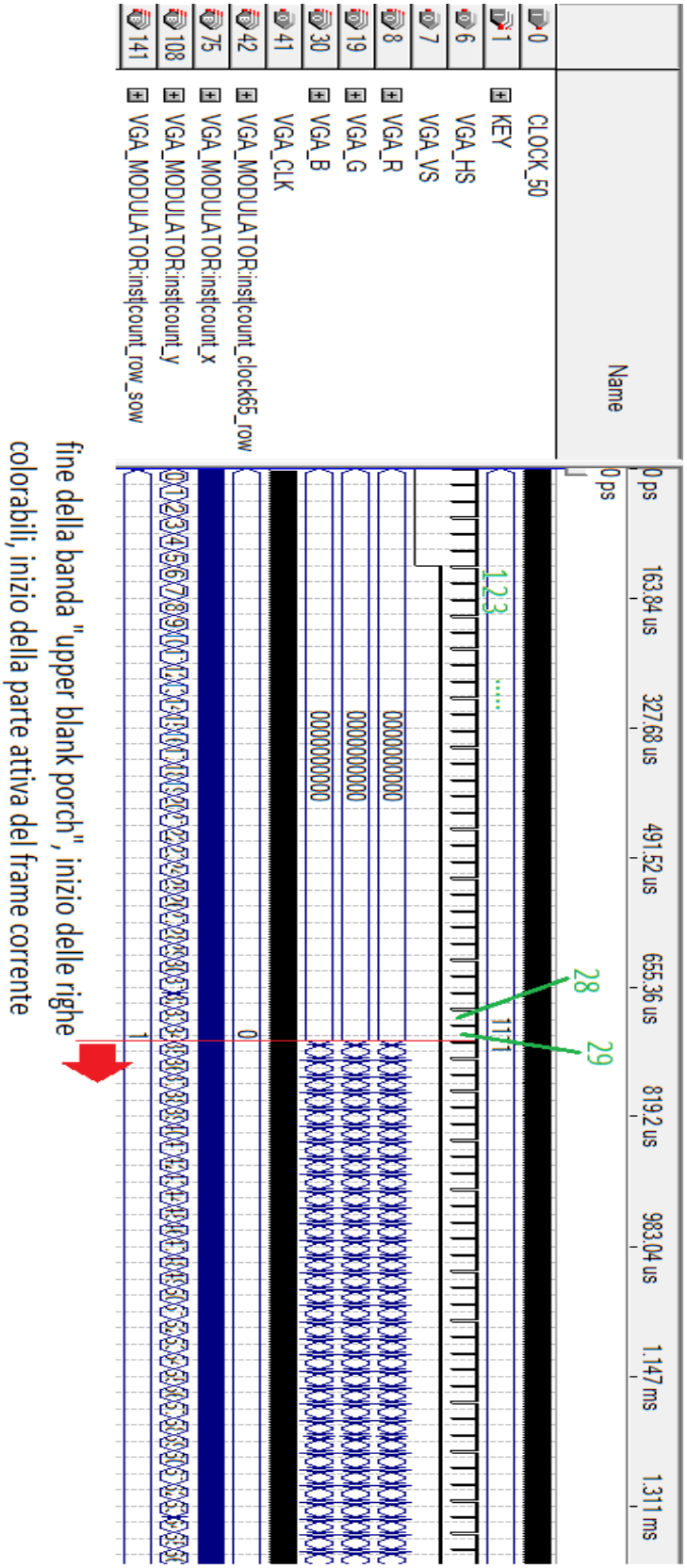


Fig. 67

Segmento di simulazione timing raffigurante la fine della scansione delle 29 righe della banda di guardia VESA “upper blank porch” e l’inizio della visualizzazione su monitor della parte attiva del frame corrente

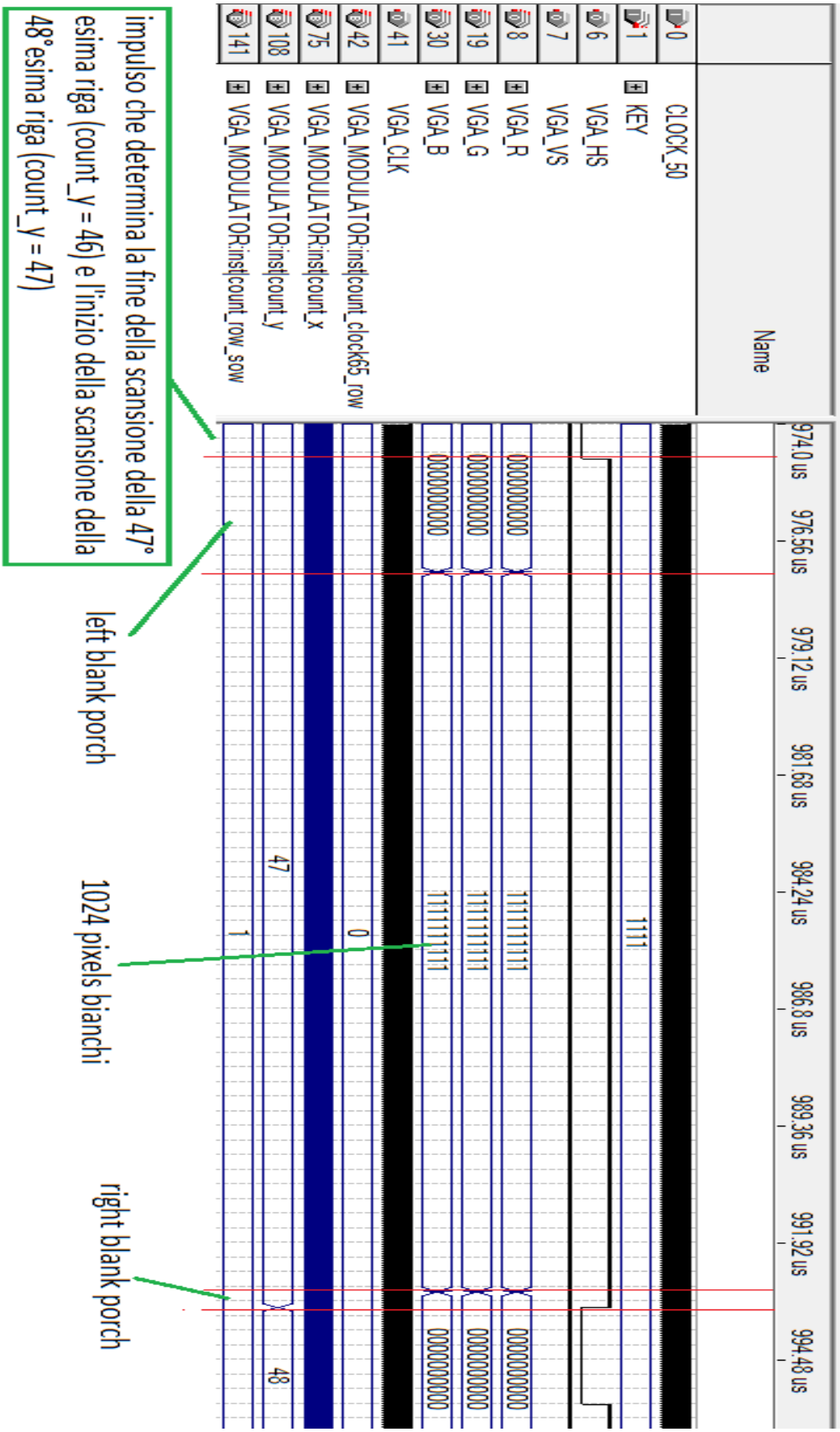


Fig. 68

Particolare di simulazione timing raffigurante la scansione della 48°esima riga, appartenente quindi alla fascia bianca superiore della frequenza di default f0 del dente di sega: si noti la modulazione digitale dei dati VGA_R[9:0], VGA_G[9:0] e VGA_B[9:0] con la quale si passa dai pixels blank (fasce scure laterali) a quelli bianchi, costituenti lo sfondo della forma d'onda

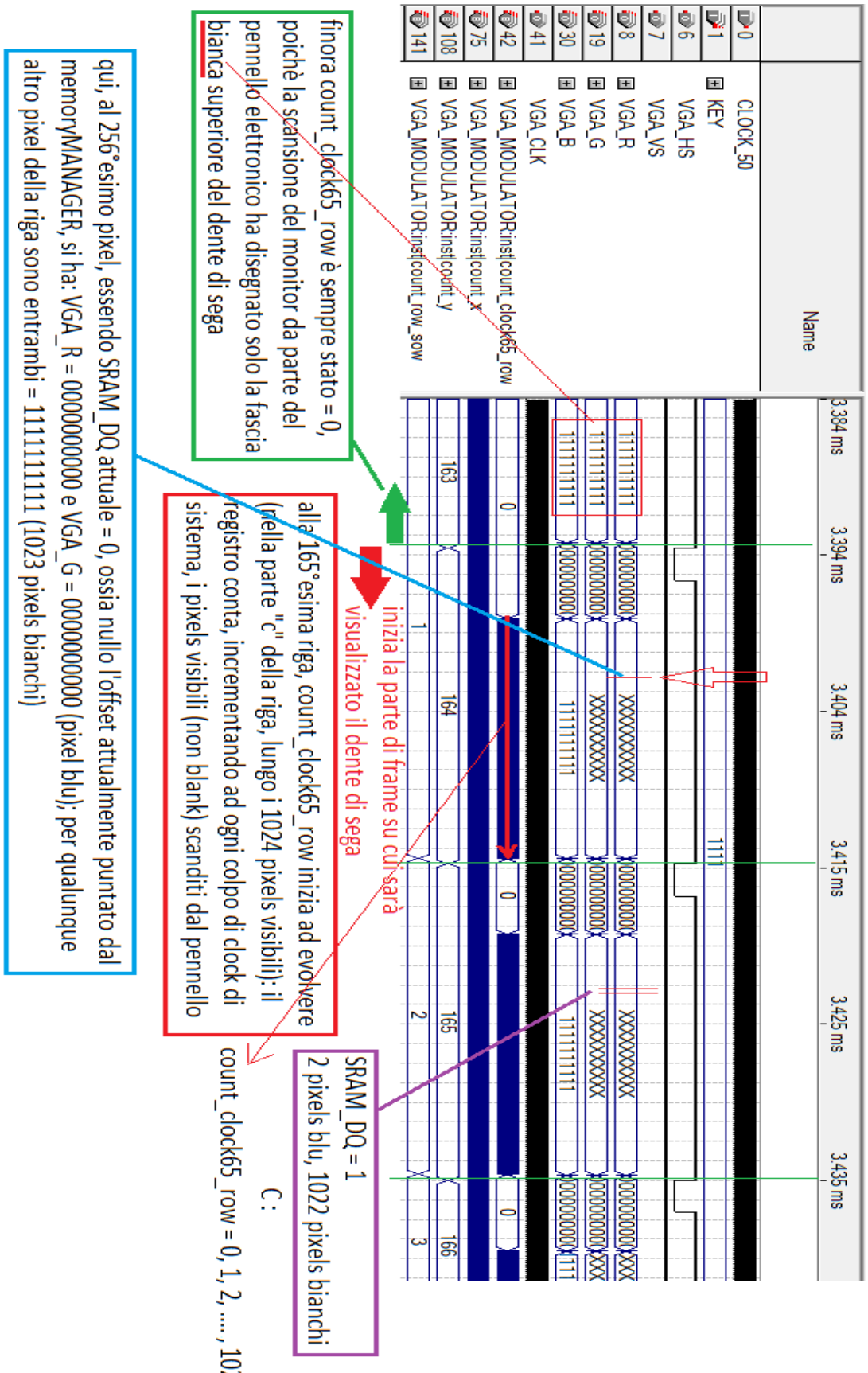


Fig. 69

Segmento di simulazione timing raffigurante la fine della scansione della fascia bianca superiore e l'inizio della parte del frame corrente nella quale sarà visualizzata la forma d'onda del dente di sega

Come si può evincere dalla figura precedente, il registro count_clock65_row è tenuto a 0 finchè la scansione riguarda la fascia bianca superiore, ossia finchè il pennello elettronico insiste sulla parte di schermo che non comprende alcun punto della forma d'onda visualizzanda. Appena il pennello entra nella parte di frame attivo che ospiterà il dente di sega, il registro count_clock65_row, nella parte "c" di ciascuna riga, inizia a incrementare ad ogni colpo di clock di sistema, così da contare il numero di pixels scanditi (count_clock65_row = 0, 1, 2, …, 1023), quindi colorati, fino a quel momento. Anche count_row_saw ("contatore delle righe del dente di sega") è un registro il cui contenuto inizia ad evolvere solo all'atto dell'ingresso del pennello elettronico all'interno della parte di frame contenente il dente di sega: il suo incremento contrassegna il numero di righe, costituenti l'immagine del dente di sega, scandite fino a quel momento.