

# POLITECNICO DI TORINO

---



## ESERCITAZIONE DI LABORATORIO DI SISTEMI DIGITALI INTEGRATI **Laboratorio 2**

*Progetto di laboratorio di Sistemi  
Digitali Integrati*

**Data:**  
**22/01/2016**

**Partecipanti:**  
*Marco Coletta 231067*  
*Francesco Condemi 231127*  
*Nicolangelo Piscitelli 231005*

# Indice

<b>1</b>	<b>UART</b>	<b>2</b>
1.1	Scopo esercitazione . . . . .	2
1.2	Strumentazione utilizzata . . . . .	2
1.3	Specifiche . . . . .	3
1.4	Progetto . . . . .	3
1.4.1	Trasmettitore . . . . .	3
1.5	Simulazione Modelsim . . . . .	7
1.6	Test in laboratorio . . . . .	8
1.6.1	Ricevitore . . . . .	9
1.7	Simulazione Modelsim . . . . .	15
1.8	Test in laboratorio . . . . .	20
1.8.1	Test Trasmettitore e Ricevitore . . . . .	21
<b>2</b>	<b>VGA Controller</b>	<b>22</b>
2.1	Scopo esercitazione . . . . .	22
2.2	Strumentazione utilizzata . . . . .	22
2.3	Specifiche . . . . .	23
2.4	Progetto . . . . .	23
2.4.1	FIFO controller . . . . .	23
2.5	Simulazione Modelsim . . . . .	29
2.5.1	VGAsincr . . . . .	31
2.6	Simulazione Modelsim . . . . .	39
2.7	Test in laboratorio . . . . .	46
<b>3</b>	<b>Memory Controller</b>	<b>48</b>
3.1	Specifiche . . . . .	48
3.2	Progetto . . . . .	48
3.2.1	Arbitro . . . . .	49
3.2.2	Memory Interface . . . . .	54
3.3	Analisi dei tempi . . . . .	57
3.4	Simulazione Modelsim . . . . .	58
3.5	Test in laboratorio . . . . .	59
<b>4</b>	<b>Test Finale</b>	<b>61</b>
4.1	Test Bench . . . . .	62

# **Capitolo 1**

## **UART**

### **1.1 Scopo esercitazione**

Scopo di questa esercitazione è la realizzazione di un modulo UART composto da un trasmittore-ricevitore avente *bit rate* di 9600 bit/s da implementare sulla scheda "Altera DE2".

### **1.2 Strumentazione utilizzata**

- Altera DE2
- Oscilloscopio digitale Tektronix TDS 210
- Sonde compensate
- Computer

## 1.3 Specifiche

Le specifiche assegnate sono:

- Standard RS 232
- Bit rate pari a 9600 bit/s
- 1 start bit
- 1 stop bit

## 1.4 Progetto

Partendo dalle specifiche si è realizzato il *data path* del sistema. Inizialmente il trasmettitore e il ricevitore si sono considerati progetti indipendenti per poi unirli in un secondo momento. Il primo riceve in ingresso un frame composto da dieci bit (otto di dato, uno di start e uno di stop) e invia in uscita un singolo bit con *bit rate* pari a 9600 bit/s.

Il ricevitore, invece, ha il ruolo di acquisire i dati, rilevare lo start bit e quindi riconoscere l' inizio di una trasmissione, valutare il carattere ricevuto e parallelizzare l'uscita su 8 bit. Per evitare campionamenti errati dovuti ad eventuali *glitch* si è deciso di far lavorare il ricevitore ad una frequenza sedici volte maggiore quella del trasmettitore.

### 1.4.1 Trasmettitore

#### Data Path

In figura 1.1 è rappresentato il *data path* progettato per il "Trasmettitore".

Esso è composto da uno *shift register* che prende in ingresso un dato da dieci bit e lo restituisce in uscita un bit per volta, e da due contatori, di cui il primo modulo 2605 utilizzato per determinare quando abilitare lo *shifter* ad inviare un dato in uscita ed il secondo modulo 11 utilizzato per determinare quando l'intera operazione è terminata. Tutto il sistema opera ad una frequenza di clock di 25 MHz.

Per calcolare il terminal count del primo contatore si è usata la seguente formula:

$$n_{clock} = \frac{25MHz}{9600Hz} = 2605 \quad (1.1)$$

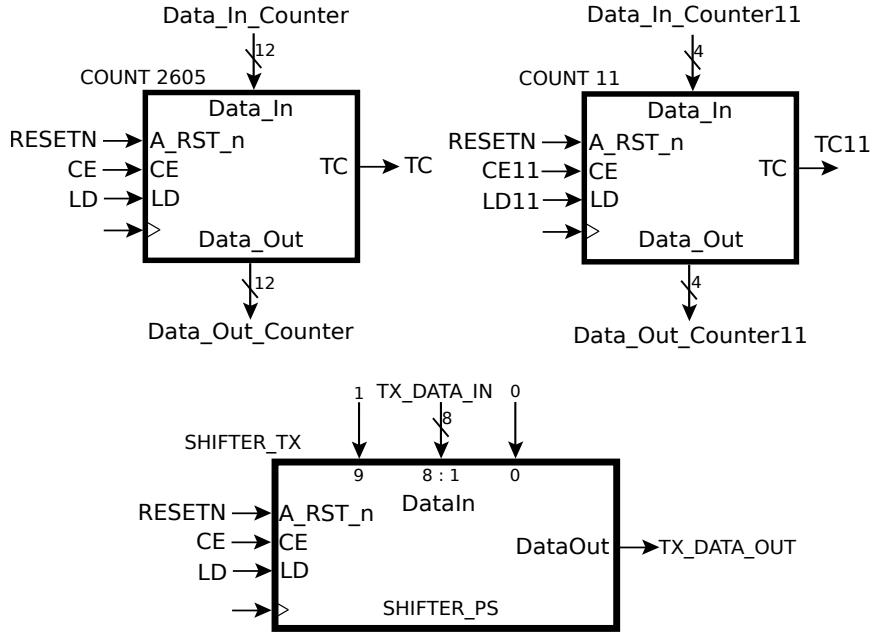
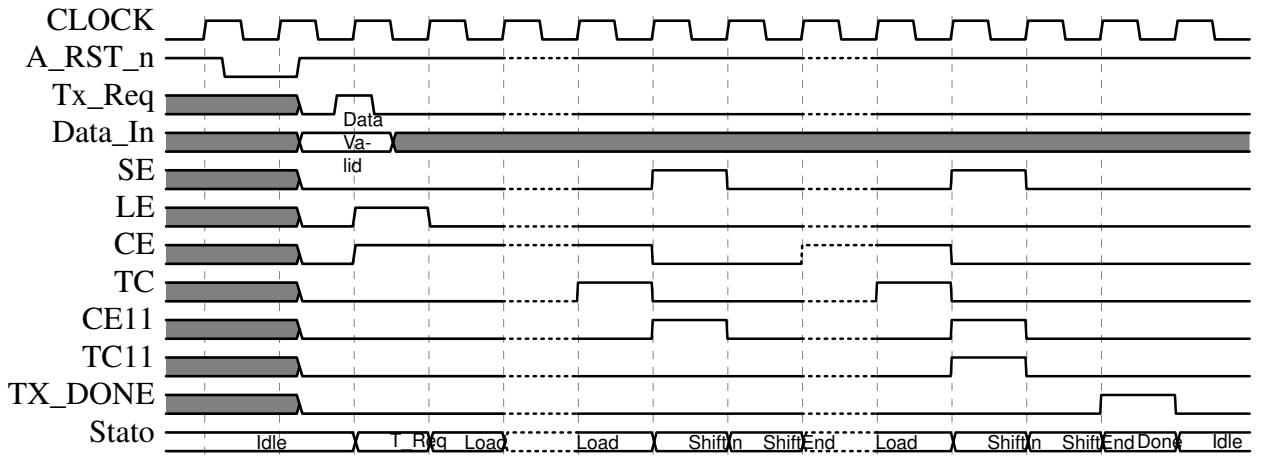


Figura 1.1: Trasmettitore Data Path

### Timing Diagram



### FSM

In figura 1.2 è rappresentata la *finite state machine* del "Trasmettitore".

La macchina lascia lo stato di "Idle" quando viene campionata una richiesta di invio dati tramite il segnale "TX\_Req". Essa finisce quindi in una coppia di stati che fanno sì che tale dato venga caricato nello *shift register* e contemporaneamente venga abilitato a contare il "COUNT2605". Si entra quindi in un *loop* che fa sì che ogni volta che viene asserito il "TC" di tale contatore possa essere portato in un'uscita un bit e quindi incrementato di una unità il dato di uscita del "COUNT11". Tale artefatto viene utilizzato per far sì che i singoli bit escano dalla macchina con una frequenza di 9600 Hz nonostante essa lavori a 25 MHz.

Quando viene asserito "TC11" vuol dire che tutti i dieci bit sono stati portati in uscita

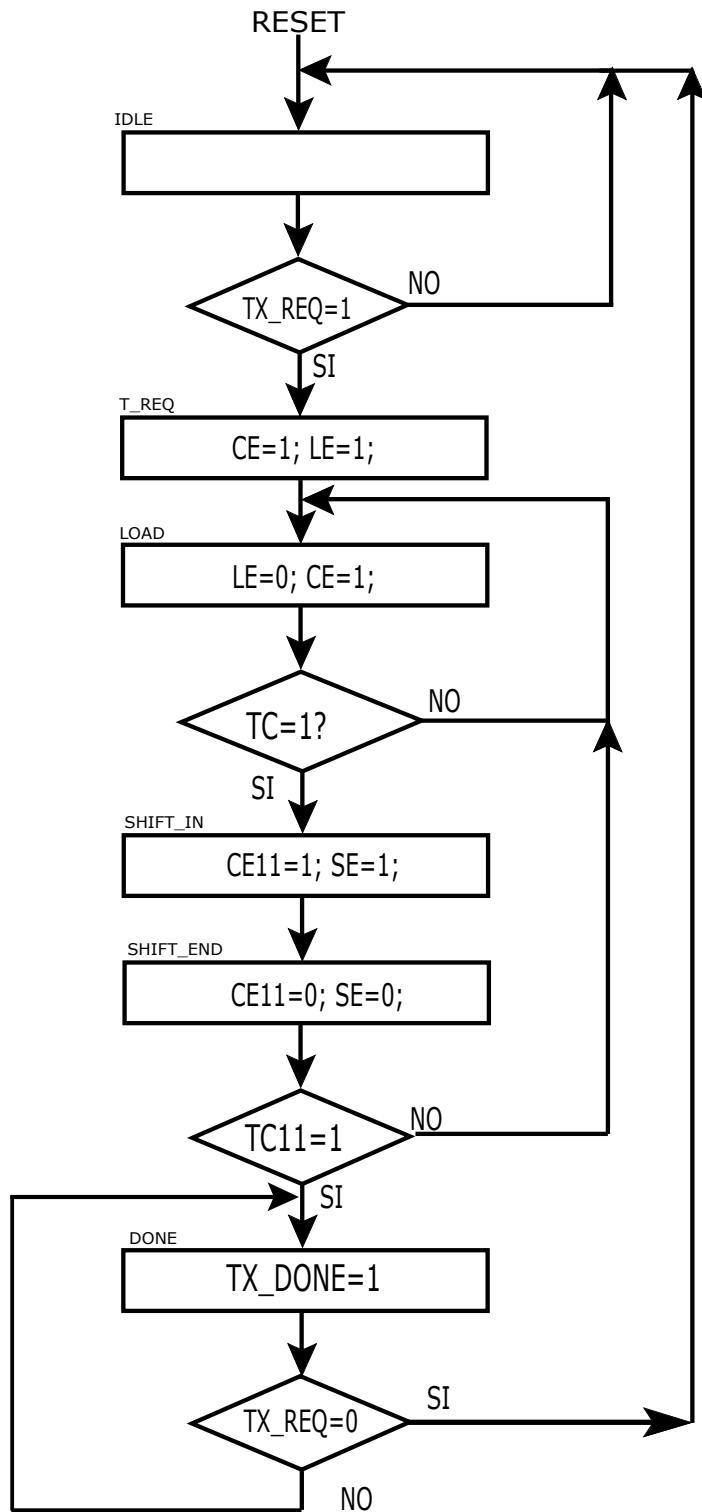


Figura 1.2: Trasmettitore Finite State Machine

e quindi la macchina finisce nell'ultimo stato di "Done" nel quale viene asserito il segnale "TX\_DONE" che segnala il completamento dell'operazione.

## 1.5 Simulazione Modelsim

Di seguito viene riportata la simulazione mediante il software modelsim:

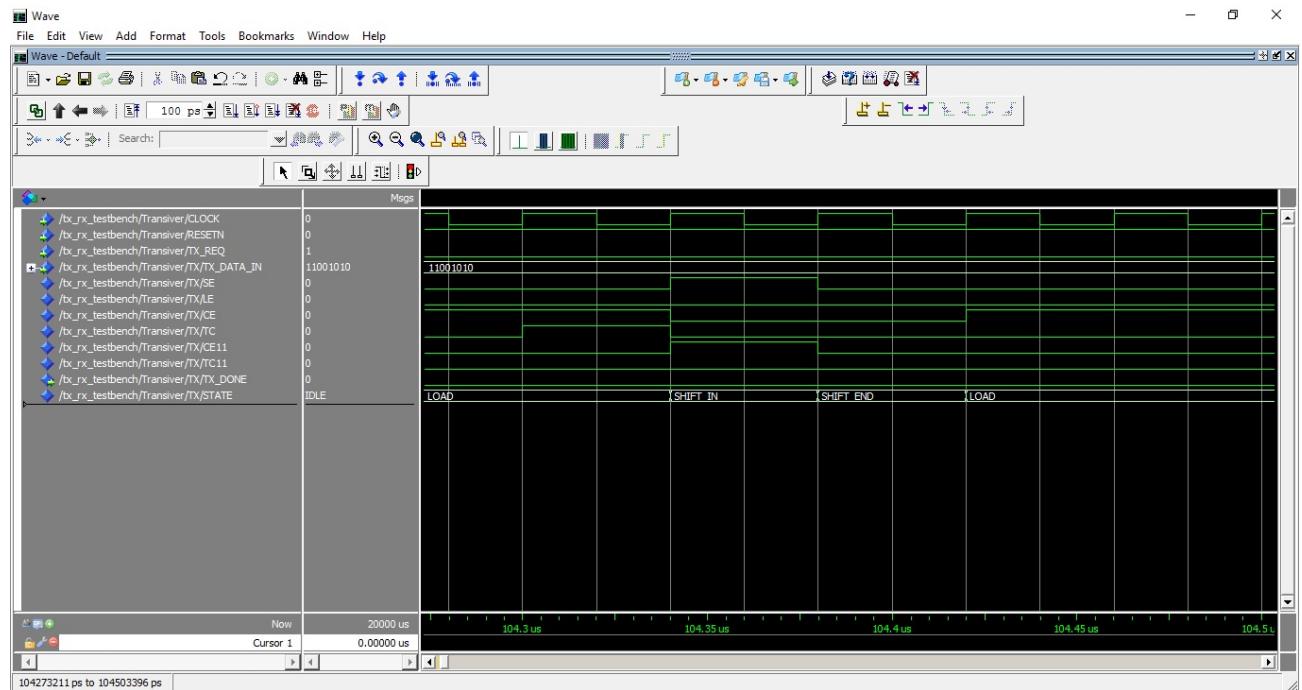


Figura 1.3: Simulazione Trasmettitore

## 1.6 Test in laboratorio

Il test in laboratorio è stato svolto apportando una piccola modifica al progetto ed inserendo nella entity superiore degli switch e dei led che potessero essere utili nelle varie prove. In questo caso lo switch zero funziona da reset mentre lo switch 1 serve per la richiesta di trasmissione.

Gli switch da 2 a 9 servono per dare dei dati. L'invio dei dati è stato dapprima controllato mediante oscilloscopio:

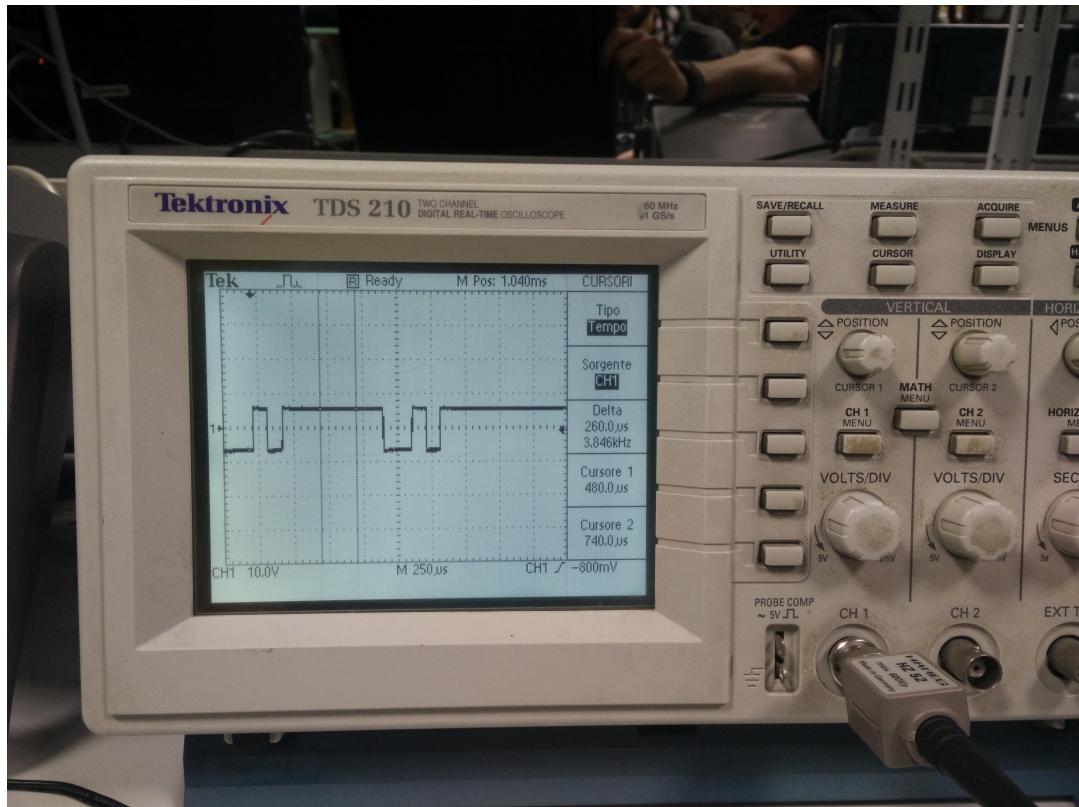


Figura 1.4: Dati inviati visualizzati sull'oscilloscopio

## 1.6.1 Ricevitore

### Data Path

In figura 1.5 viene mostrato il *data path* del "Ricevitore".

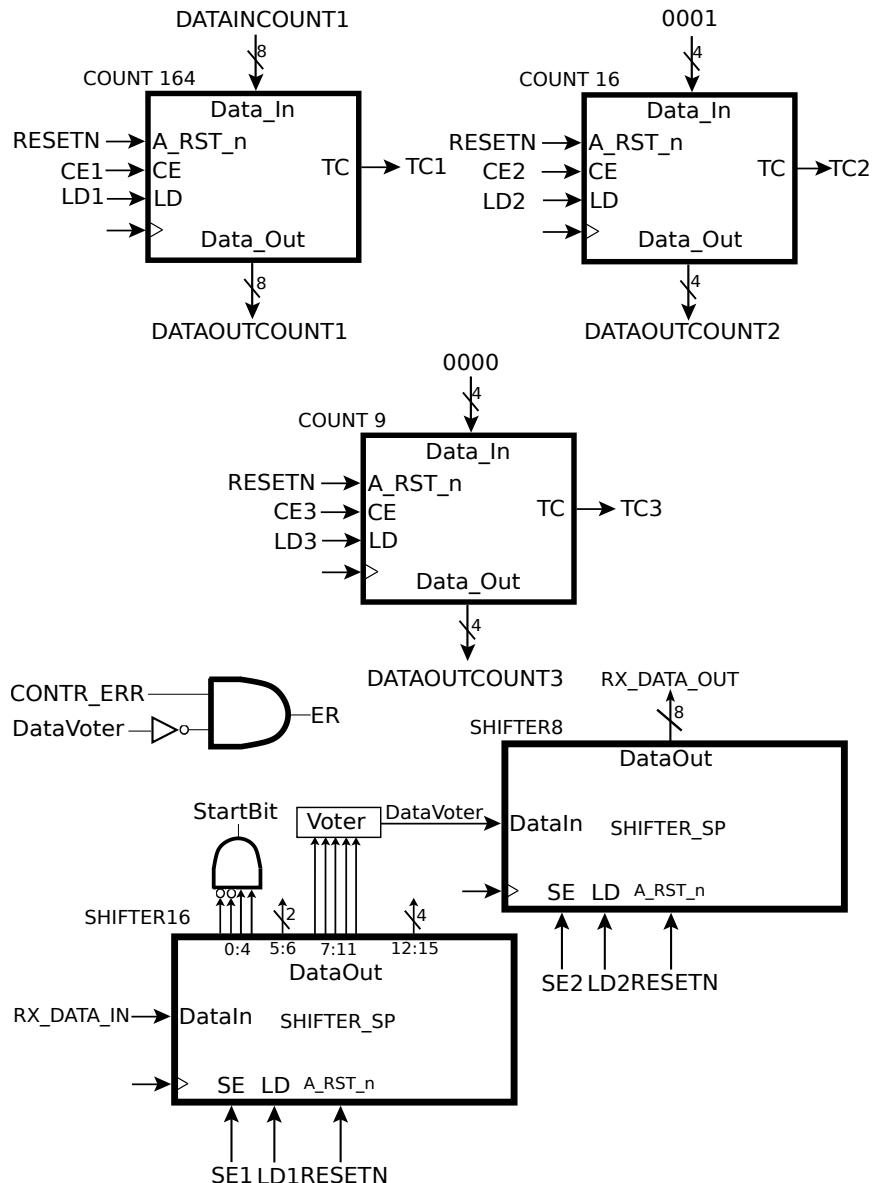


Figura 1.5: Ricevitore Data Path

Come anticipato nei paragrafi precedenti, per diminuire la probabilità che il ricevitore campioni dei dati errati, esso lavora ad una frequenza sedici volte maggiore di quella del trasmettitore. In tale modo esso campiona sedici volte ogni bit che riceve in ingresso. Un contatore modulo 16 permette di individuare quando è terminata la trasmissione di un bit e quando è cominciata la successiva.

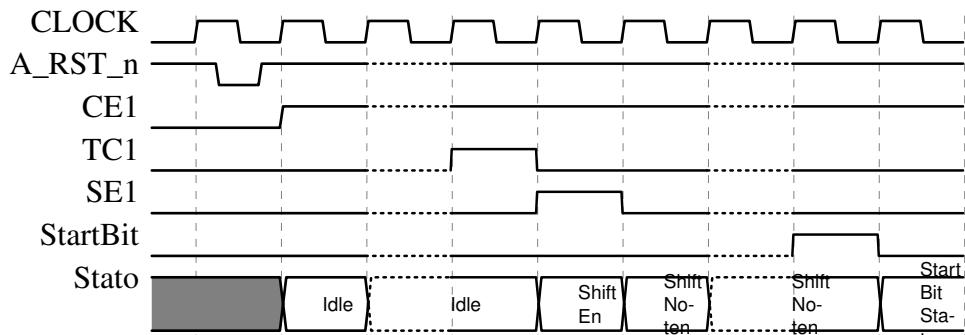
In maniera analoga a quanto fatto per il "Trasmettitore" un contatore modulo 164 permette a tale macchina di lavorare a 153600 Hz (sedici volte 9600 Hz), nonostante essa lavori ad una frequenza di clock di 25MHz.

Un contatore modulo 9 infine permette di individuare quando è stato ricevuto anche l'ultimo bit di dato e quindi quando è possibile portare in uscita il dato parallelizzato. Il primo *Shift Register*, per il motivo sopra citato, è in grado di memorizzare sedici bit. I primi quattro, di cui i primi due negati, entrato in una porta "and" per individuare lo start bit. Infatti secondo il protocollo RS232 il bit di stop deve essere un '1' e quello di start uno '0'.

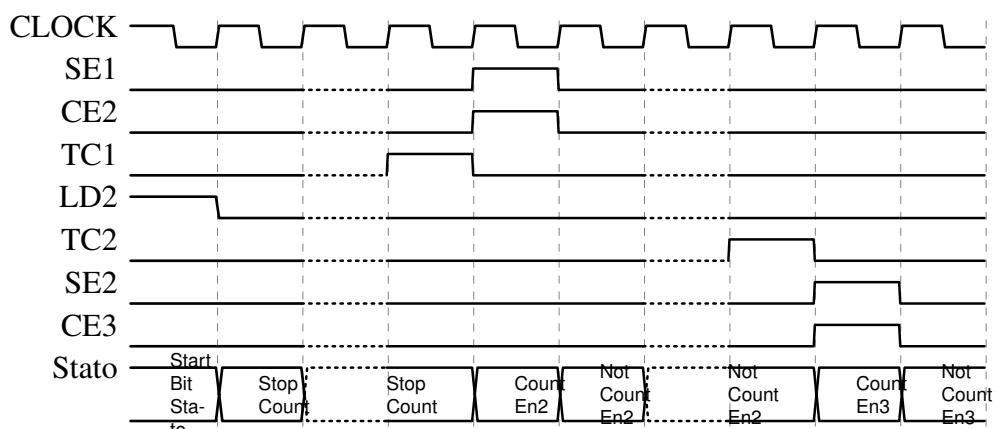
I bit dal settimo all'undicesimo vanno agli ingressi di un circuito combinatorio denominato "Voter" che porta l'uscita a '1' se in ingresso ci sono un numero maggiore di '1' e viceversa. Tale segnale è l'ingresso del secondo *shifter* che in fine parallelizzerà i dati portandoli in uscita.

## Timing Diagram

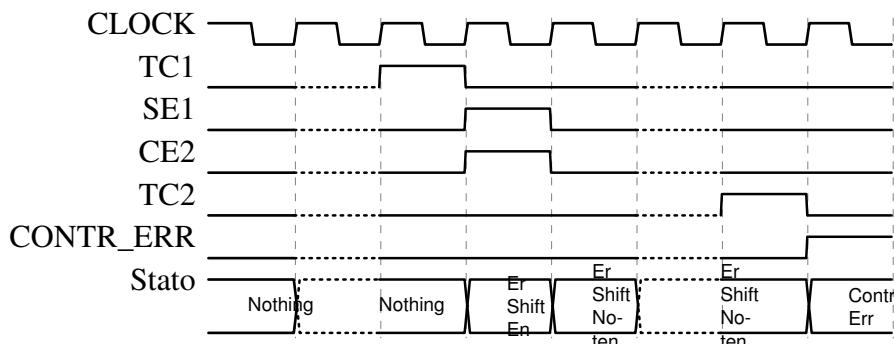
Il seguente *timing diagram* riporta la transizione degli stati dall'"Idle" fino al "Start-Bit\_State".



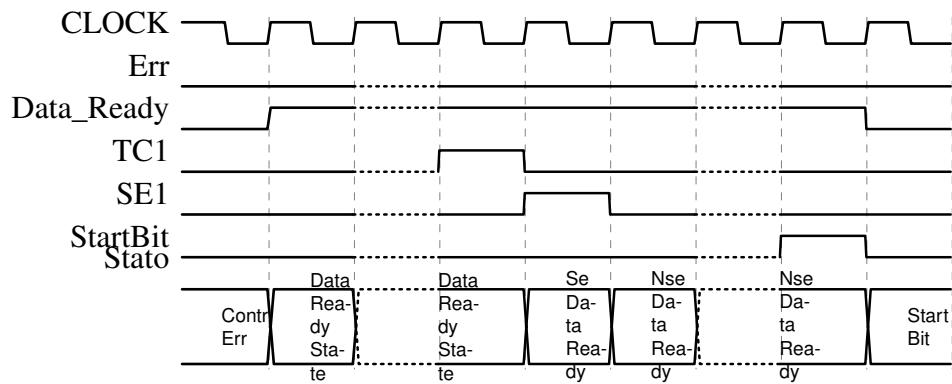
Il seguente *timing diagram* riporta la transizione degli stati dallo "StartBit\_State" fino a "NotCountEn3"



Il seguente *timing diagram* riporta la transizione degli stati dal "Nothing" fino a "ContrErr"



Il seguente *timing diagram* riporta la transizione degli stati da "ContrErr" fino alla rilevazione di un nuovo "StartBit". In particolar modo viene presentata la situazione in cui la ricezione dei dati non ha presentato errore.



## FSM

In figura 1.6 viene mostrato la FSM del "Ricevitore".

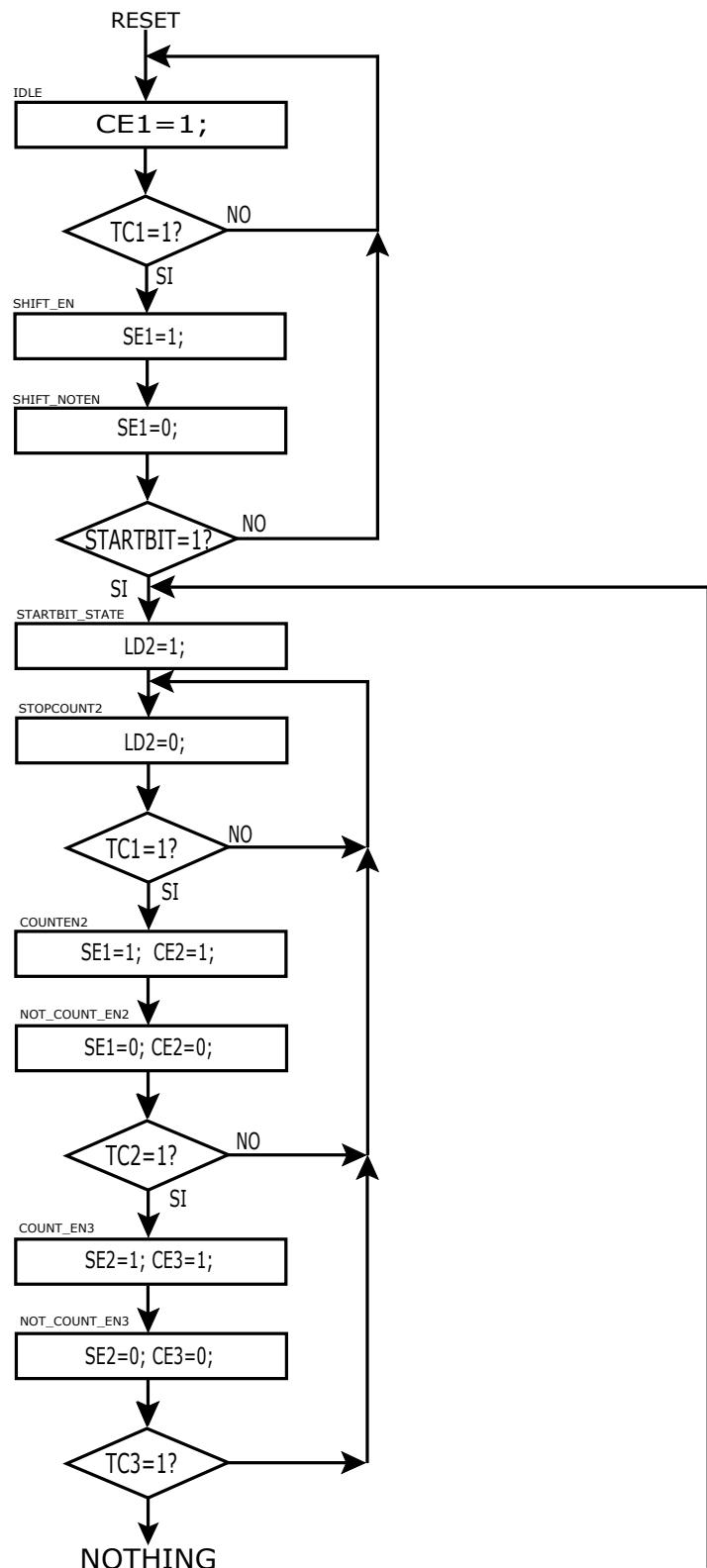


Figura 1.6: Ricevitore Finite State Machine

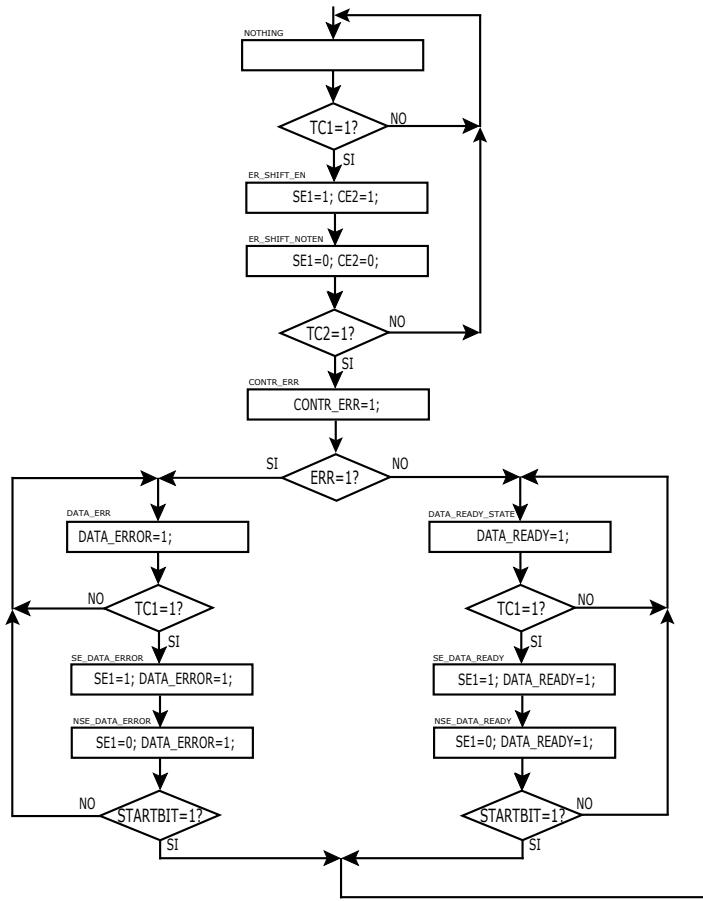


Figura 1.7: Ricevitore Finite State Machine continua

Tale macchina è programmata per far sì che il "COUNT164" sia sempre attivo in maniera tale da acquisire continuamente dati alla frequenza di 153600 Hz.

Quando viene rilevato lo "StartBit" viene abilitato a contare anche il "COUNT16", che solo per la prima volta parte dal valore 1, in quanto, per questo caso, sono già stati prelevati due campionati.

Quando si sta campionando lo stop bit viene abilitato il segnale "CONTR\_ERR", che fa sì che il segnale ER venga asserito se tale bit fosse pari '0' in contrasto con quanto stabilito dal protocollo.

Se "ER" è pari ad un uno viene asserito il segnale di errore "DATA\_ERROR", in caso contrario quello di dato valido "DATA\_VALID".

La macchina riparte quando viene rilevato un nuovo start bit.

## 1.7 Simulazione Modelsim

La simulazione con Modelsim del ricevitore è stata suddivisa in più parti per evidenziare gli stati principali e l'evoluzione della macchina a stati:

Di seguito la simulazione di quanto mostrato nel primo timing diagram:

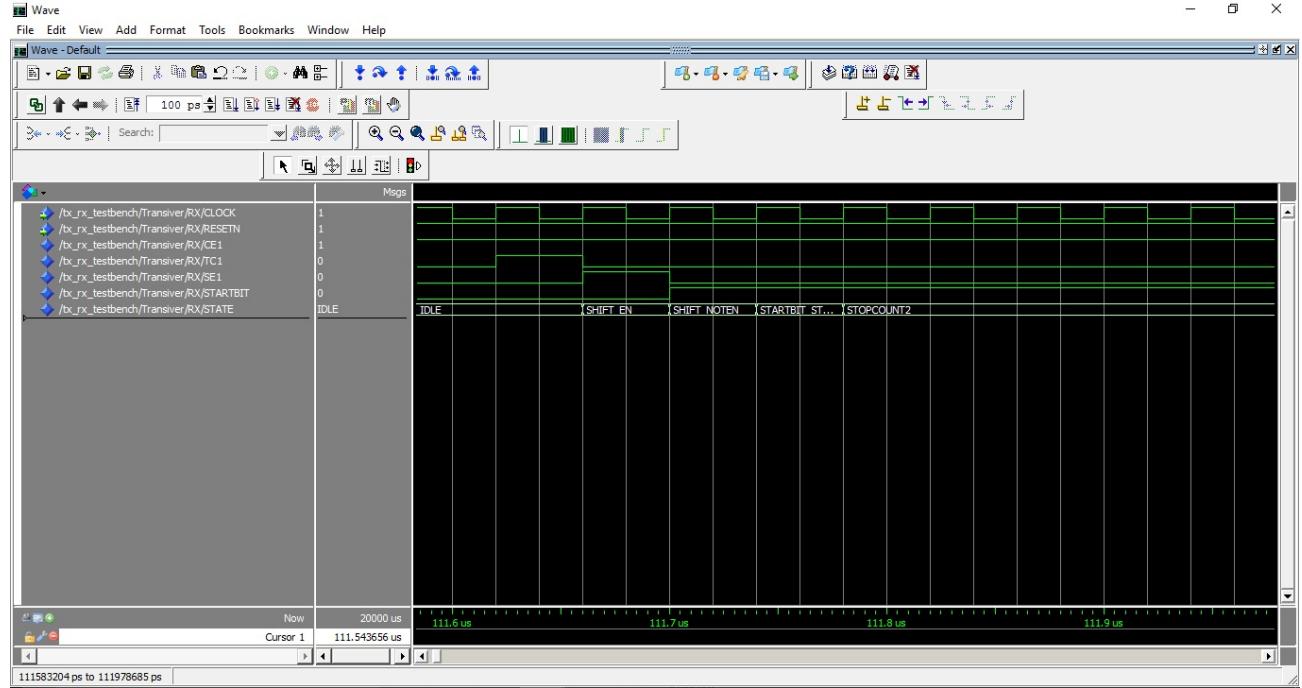


Figura 1.8: Evoluzione degli stati da IDLE a Startbitstate

Di seguito le tre simulazioni attinenti al secondo timing diagram:

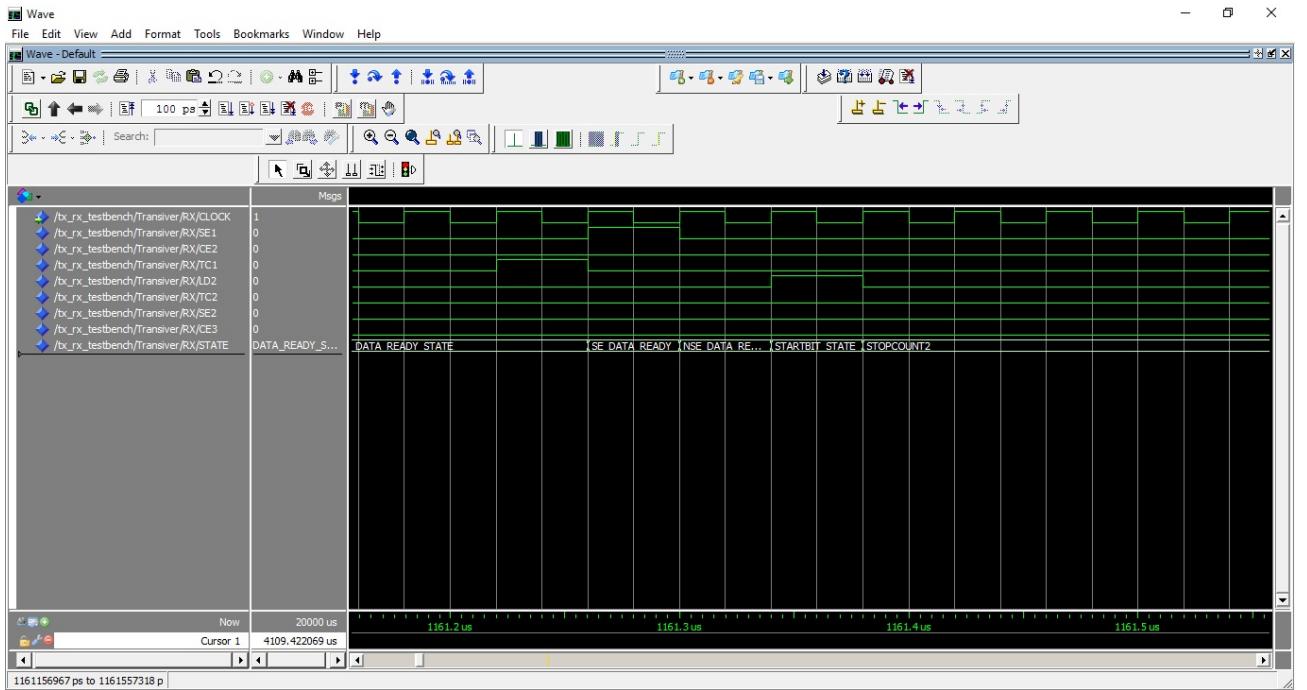


Figura 1.9: Evoluzione degli stati da startbitstate a stopcount2

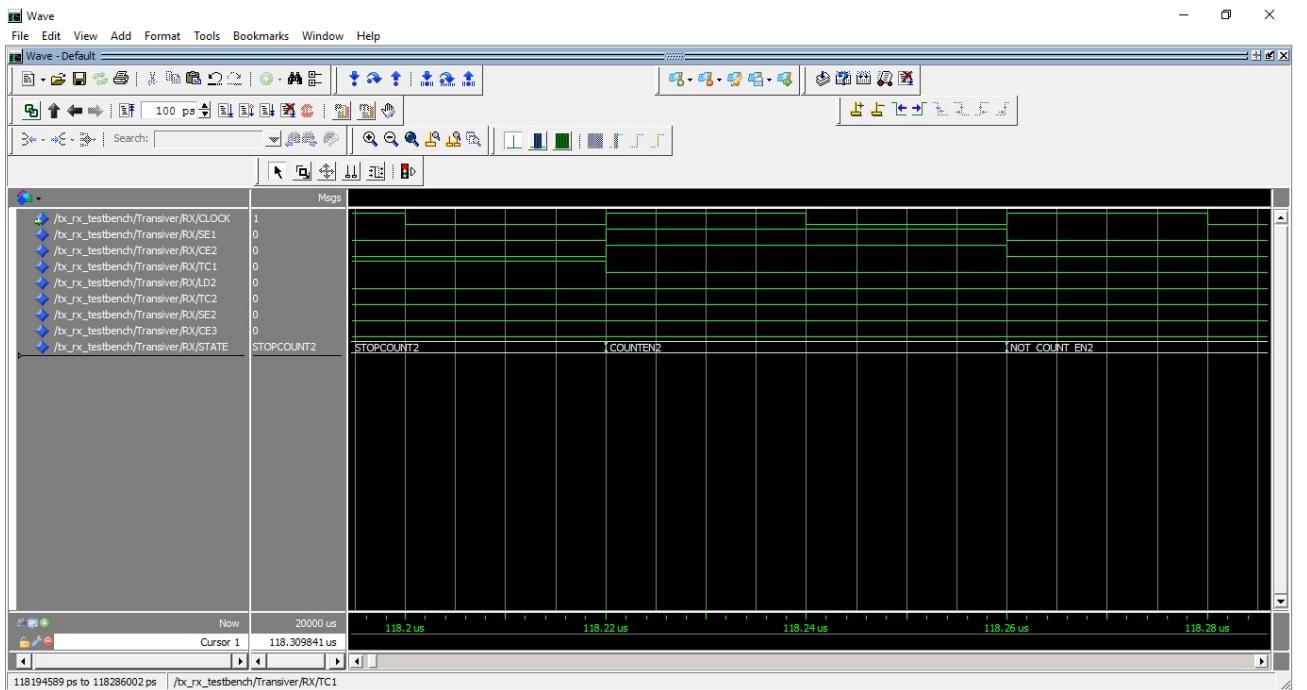


Figura 1.10: Evoluzione degli stati da stopcount2 a notcounten2

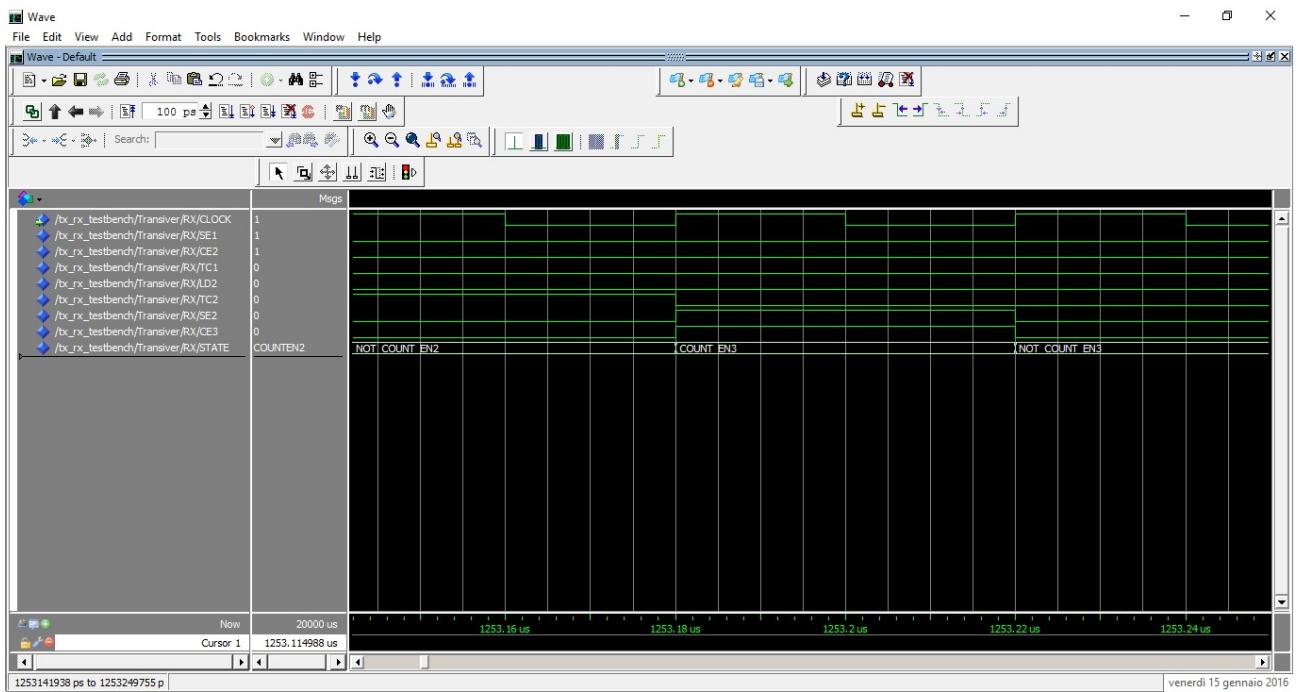


Figura 1.11: Evoluzione degli stati da notcounten2 a notcounten3

Di seguito le simulazioni con Modelsim riferite ai passaggi illustrati nel terzo timing diagram:

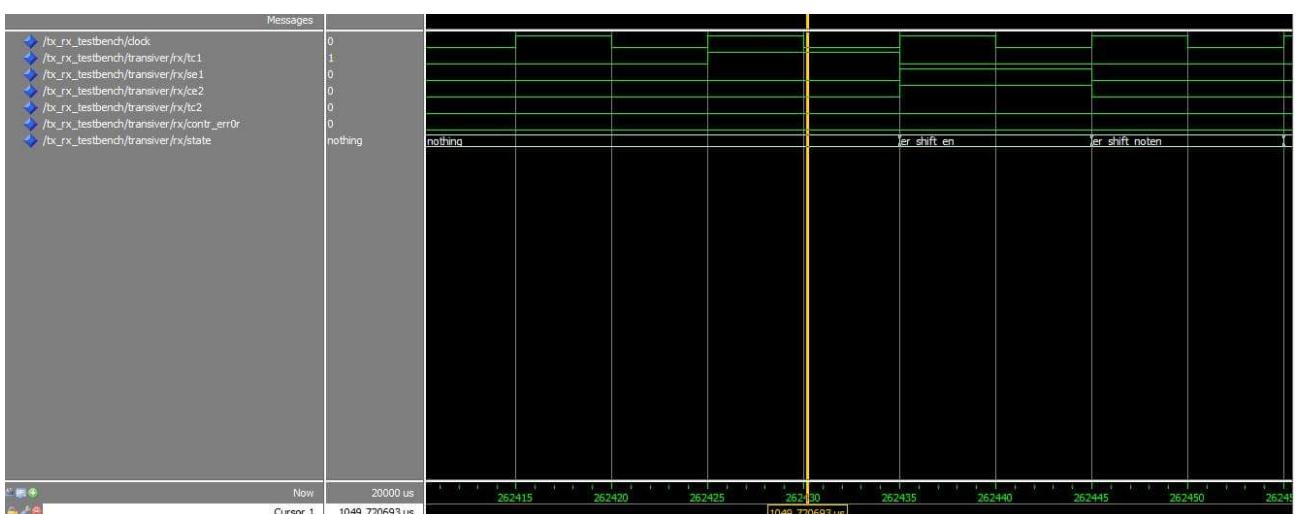


Figura 1.12: Evoluzione degli stati da nothing a ershiftnoten

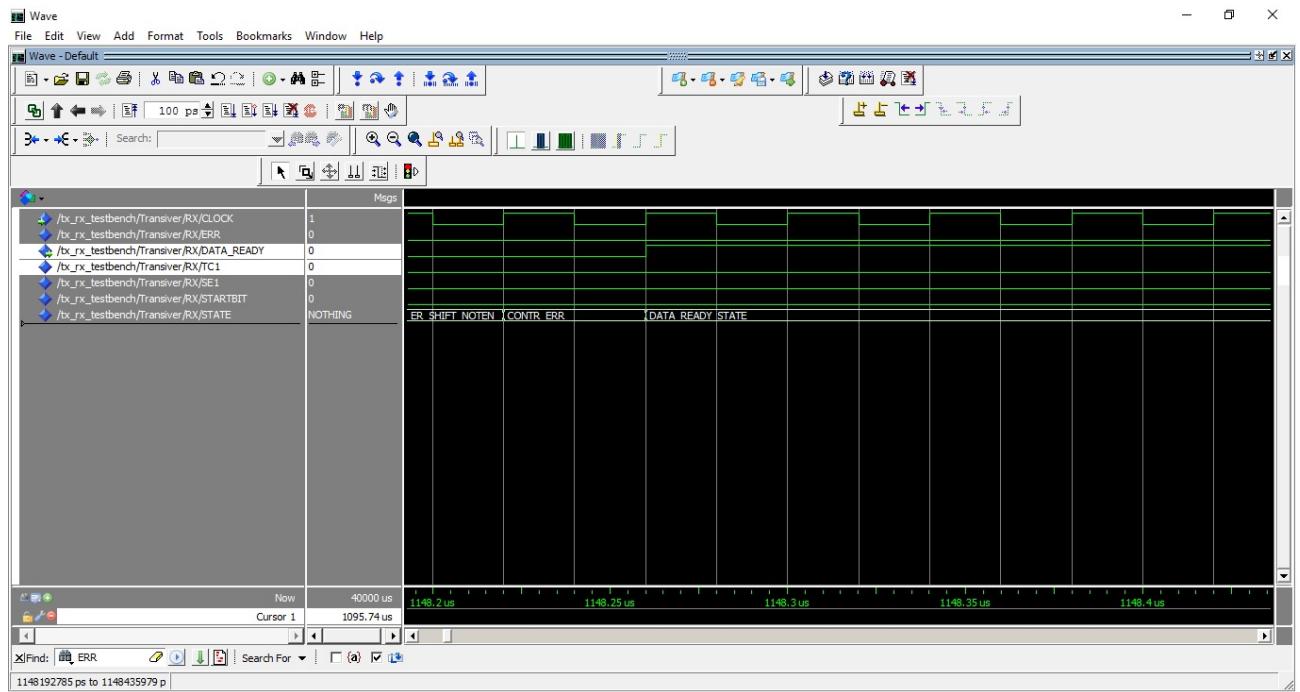


Figura 1.13: Evoluzione degli stati da ershiftnoten a contrerr

Le simulazioni sottostanti si riferiscono al quarto timing diagram del ricevitore:

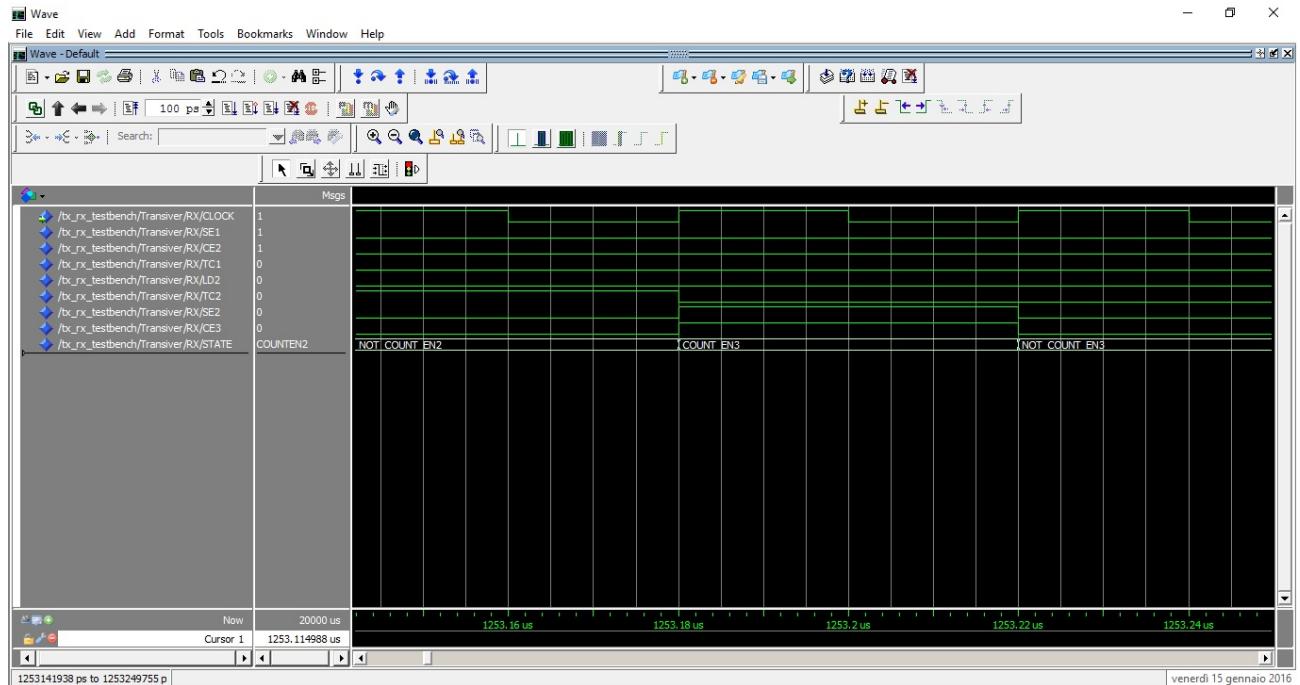


Figura 1.14: Evoluzione degli stati da notcounten2 a notcounten3

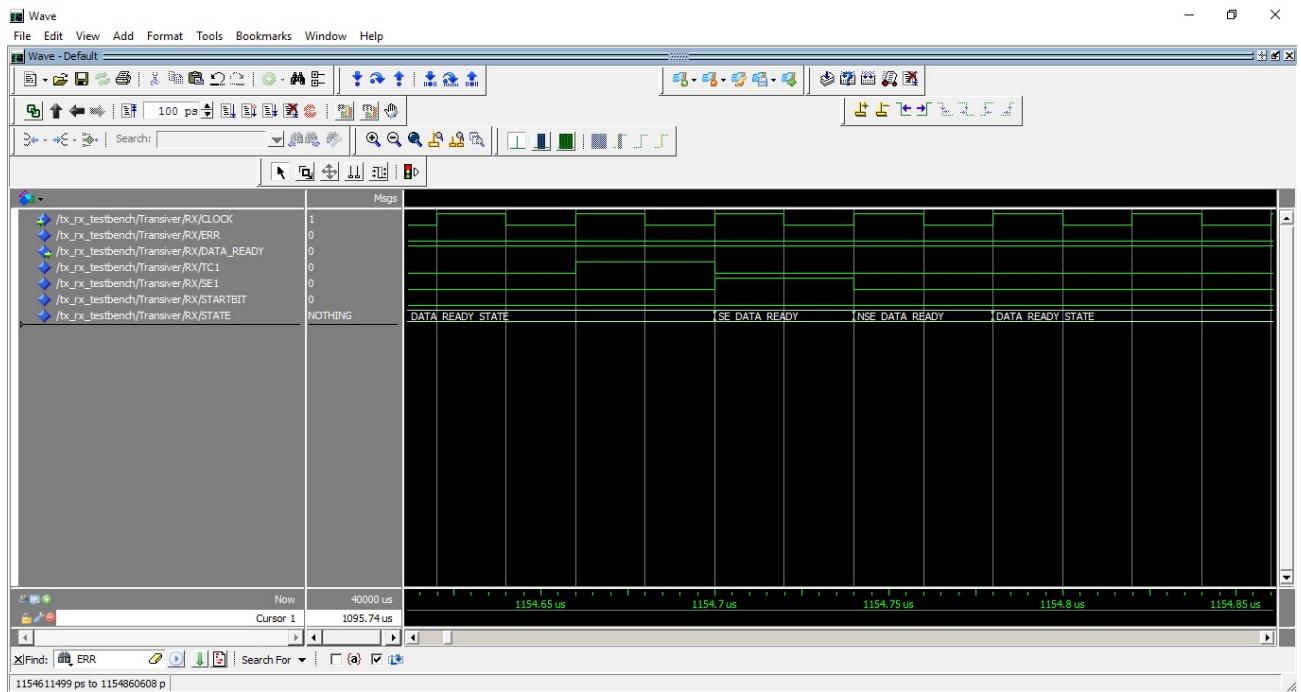


Figura 1.15: Evoluzione degli stati da datareadystate a nsedataready

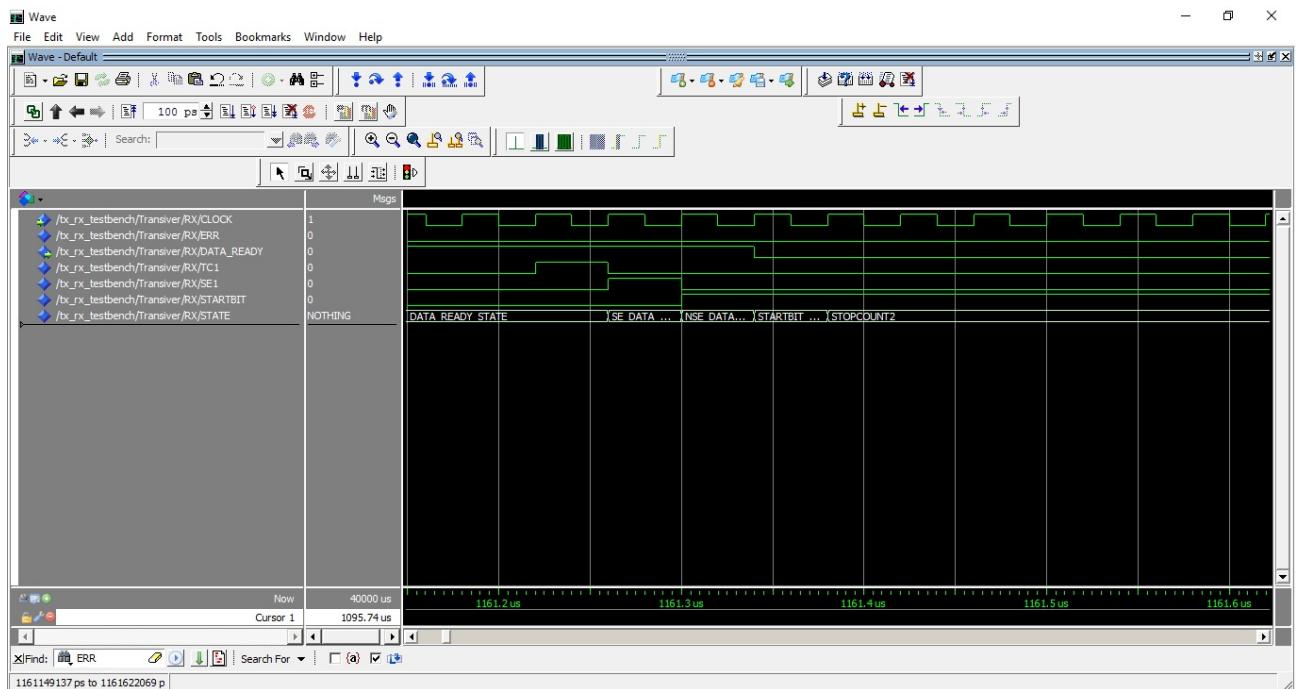


Figura 1.16: Evoluzione degli stati da nsedataready a startbit

## 1.8 Test in laboratorio

Per testare il ricevitore sono state aggiunte delle uscite alla entity in modo tale da poter visualizzare i dati attraverso dei led. Successivamente si è collegata la scheda al computer mediante cavo seriale ed è stata inviata una lettera mediante il software Terminal. In seguito viene riportata una foto del carattere inviato mediante terminale e lo stesso ricevuto sulla scheda. I led rossi sono stati collegati in modo tale da avere il bit più significativo a destra. Si nota infatti che inviando la lettera M da pc si riceve il rispettivo codice ascii 01001101.

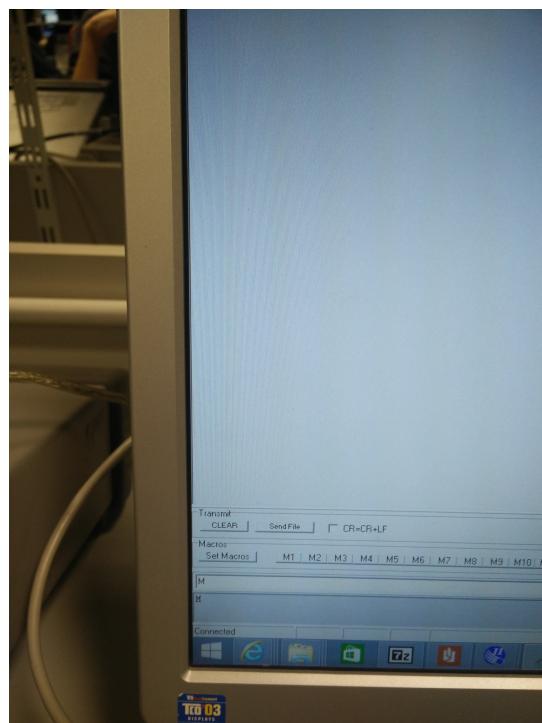


Figura 1.17: Lettera inviata mediante software Terminal

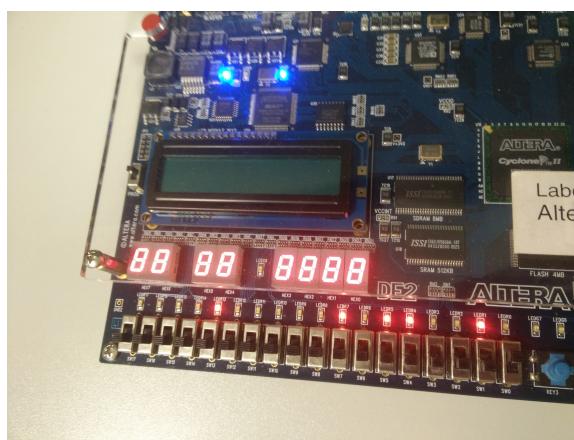


Figura 1.18: Dati ricevuti dalla DE2 mediante linea seriale

### 1.8.1 Test Trasmettitore e Ricevitore

Si è svolto poi un test in cui trasmettitore e ricevitore sono collegati insieme e si è interposto tra data\_ready del ricevitore e tx\_req del trasmettitore un blocchetto che genera un impulso. Questo affinché il trasmettitore invii un solo dato.

Per un test aggiuntivo è stato negato il terzo bit così da inviare da terminale una lettera e riceverela in minuscolo:

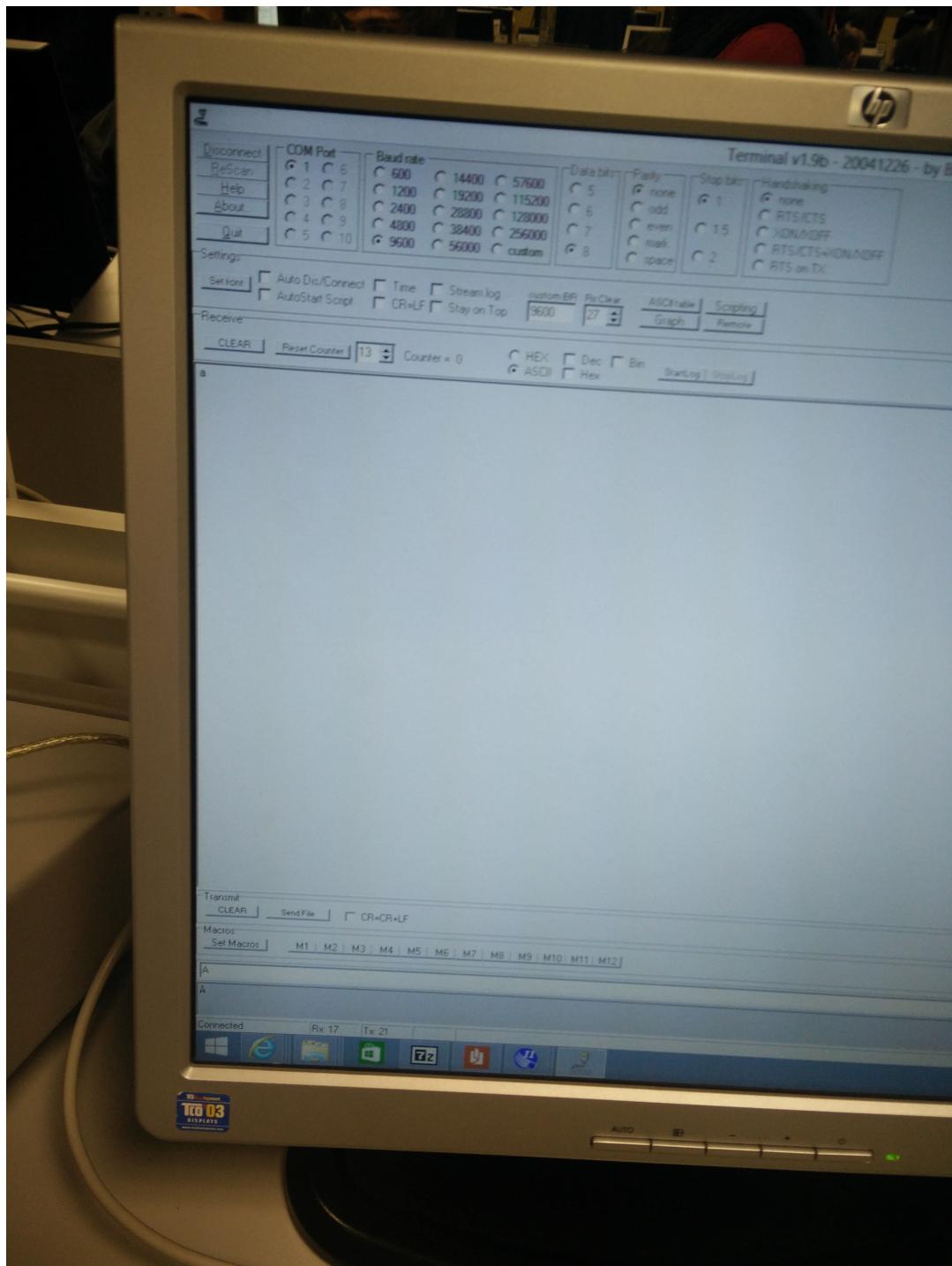


Figura 1.19: Dato inviato e poi ricevuto mediante seriale

# **Capitolo 2**

## **VGA Controller**

### **2.1 Scopo esercitazione**

Lo scopo di questa esercitazione è la realizzazione di un VGA controller implementato su scheda Altera DE2.

### **2.2 Strumentazione utilizzata**

- Altera DE2
- Oscilloscopio digitale Tektronix TDS 210
- Sonde compensate
- Computer
- Cavo seriale
- Cavo VGA
- Display

## 2.3 Specifiche

Le specifiche seguite sono:

- Standard VGA
- Risoluzione 640x480 a 60Hz
- Frequenza di lavoro 25MHz
- 30 bit di dato (10 R, 10 G, 10 B)

## 2.4 Progetto

Partendo dalle specifiche si è ricavato il *data path* del sistema. Si è scelto di dividere il VGA controller in due parti: FIFO controller per la gestione dei dati ed VGAsincr per la generazione dei segnali di sincronizzazione.

Il sistema ha in ingresso 30 bit di dato che servono per i diversi colori ma nella specifica applicazione ne sono stati utilizzati solo alcuni: si è scelto di utilizzare 3 bit per il rosso, 3 bit per il verde e 2 bit per il blu.

Il sistema si interfaccia con lo schermo mediante dei segnali HSYNC E VSYNC ed un triplo DAC a dieci bit che genera i livelli di tensione per i diversi colori.

Mentre dalla parte opposta si interfaccia con una sorgente dalla quale preleva i dati.

### 2.4.1 FIFO controller

Il FIFO controller è la parte che gestisce il flusso dati provenienti dall'esterno e diretti verso il DAC. Dato che i due flussi di dati hanno velocità differenti e che l'aggiornamento dello schermo non può essere bloccato, si è inserita una memoria FIFO.

In particolare è stata utilizzata una FIFO con 9 registri, perché è stato stimato in nove cicli di clock il ritardo massimo che può avvenire fra l'asserimento del "Req\_VGA" all'uscita dei relativi segnali "VGA\_r", "VGA\_b" e "VGA\_g". Durante il progetto per rendere possibile un flusso costante in ingresso si è deciso di utilizzare uno stadio di pipe. Questo stadio rende possibile il mantenimento di una banda di 25MHz quando il VGA controller richiede i dati. Altrimenti essa sarebbe risultata dimezzata a causa delle operazioni che impiega il FIFO controller per scambiare i dati.

Per il progetto si è cercato di utilizzare quanto più componenti genericci in modo tale da poter ampliare il parallelismo dei dati e il numero di registri a piacimento.

Durante la stesura del timing si è reso necessario modificare il component controllore in modo tale da avere il terminal count non ad N-1 ma ad N-2 in modo tale da permettere alla FSM di reagire in modo adeguato senza l'aggiunta di stati superflui.

## Data Path

Nel *data path* seguente sono illustrati tutti i componenti che sono stati utilizzati per realizzare il FIFO controller. In particolare sulla sinistra è illustrata una topview del componente con i relativi ingressi ed uscite mentre sulla destra sono presenti i componenti al suo interno.

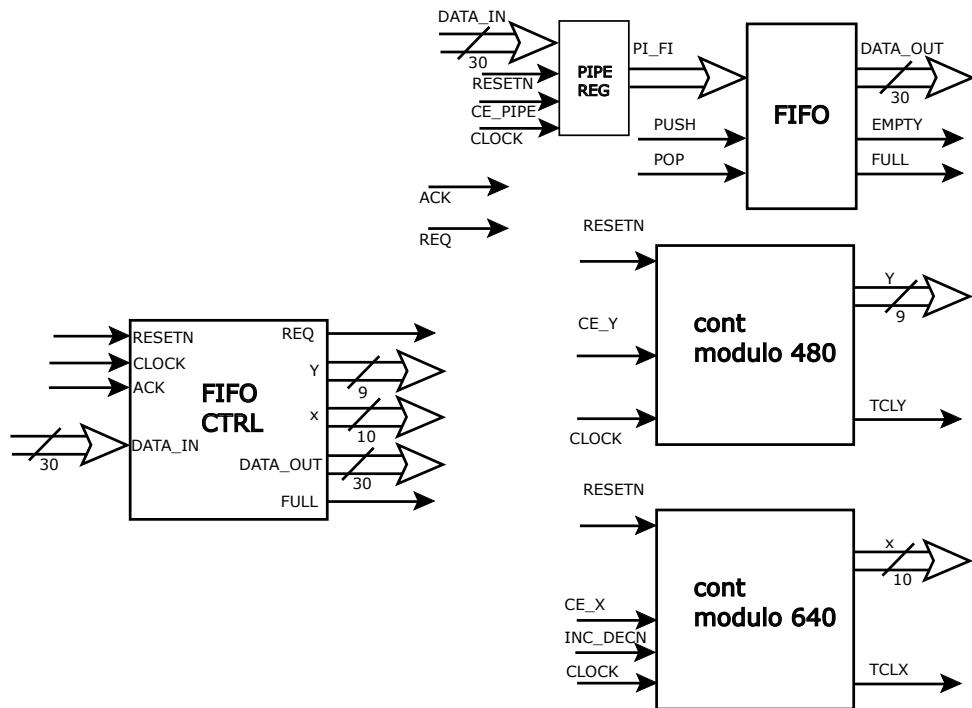


Figura 2.1: Data Path FIFO controller

## Timing Diagram

Di seguito vengono illustrati i timing principali riguardanti il funzionamento del FIFO controller. Si è diviso il timing in tre parti principali:

- fase iniziale di riempimento della FIFO
- richiesta dati dal VGA controller
- raggiungimento del terminal count delle colonne

**Timing: fase iniziale di riempimento della FIFO** In questo timing sono illustrati gli stati in cui la FSM si trova nella prima fase di carica della FIFO.

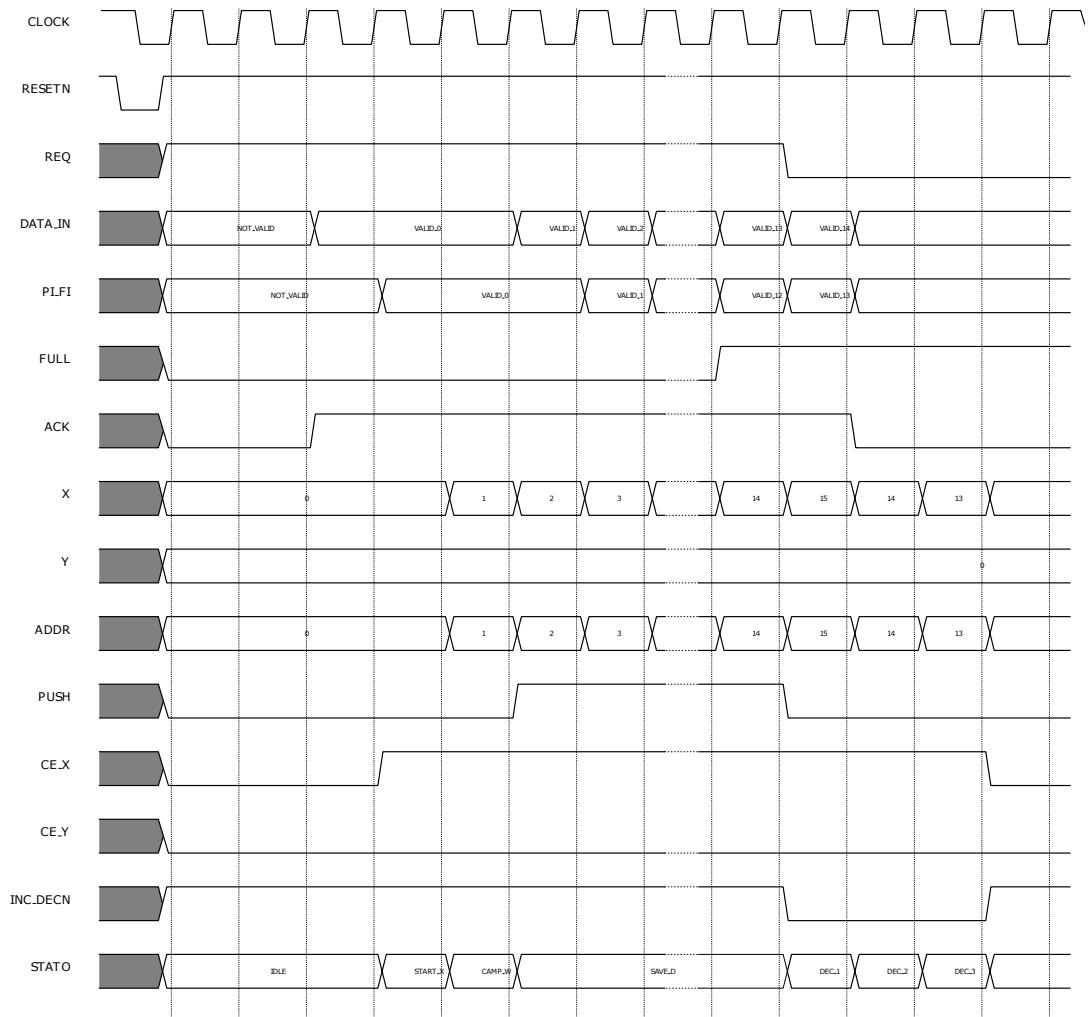


Figura 2.2: Timing diagram: fase di caricamento della FIFO dopo il reset

**Timing: richiesta dati dal VGA controller** Il seguente timing diagram illustra gli stati necessari alla FSM per caricare i dati nella FIFO quando il VGA controller incomincia a richiederli.

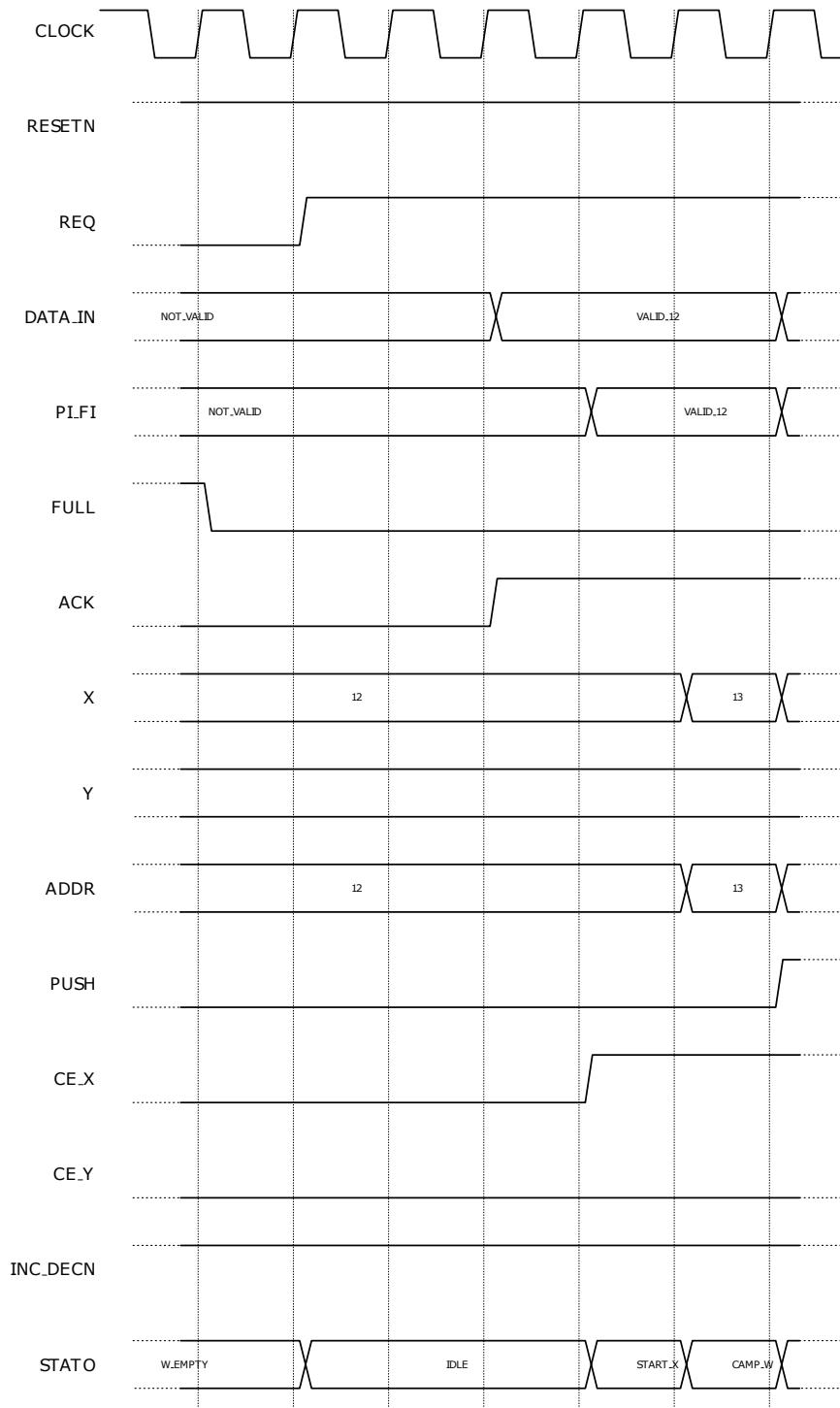


Figura 2.3: Timing diagram: inizio della richiesta dei dati da parte del VGA controller

**Timing:raggiungimento del terminal count delle colonne** Il timing seguente illustra gli stati in cui la FSM rileva un terminal count delle x e provvede ad incrementare il contatore delle y.

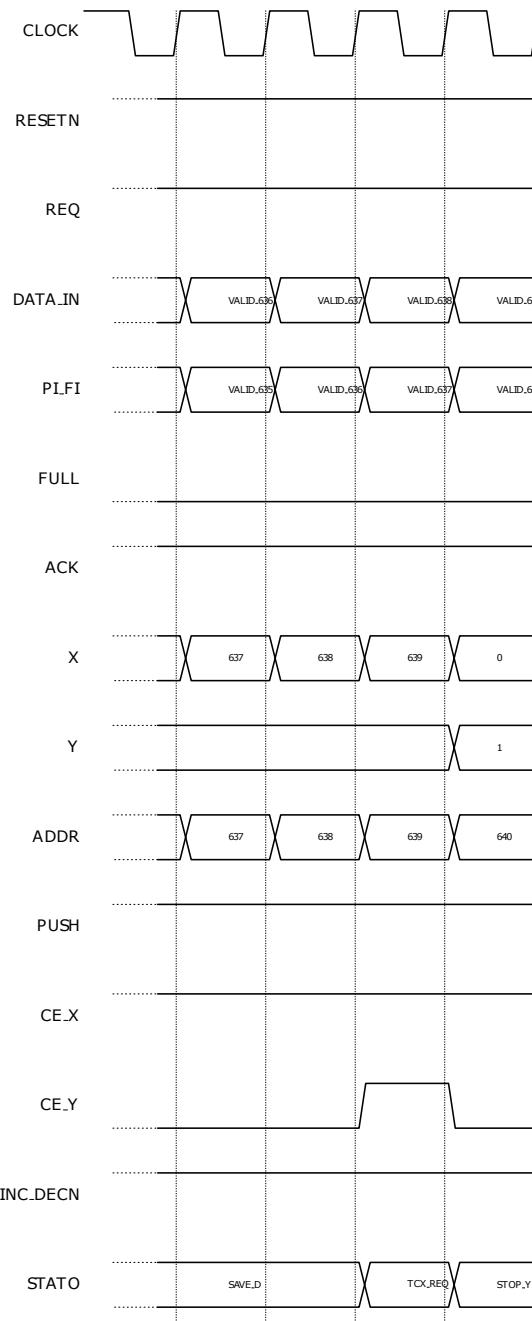


Figura 2.4: Timing diagram: traggimento del terminal count del contatore delle x ed incremento del contatore delle y

## FSM

Di seguito è illustrata la FSM del FIFO controller. Tutti i segnali presenti nel timing e non citati nei vari stati sono al livello di default.

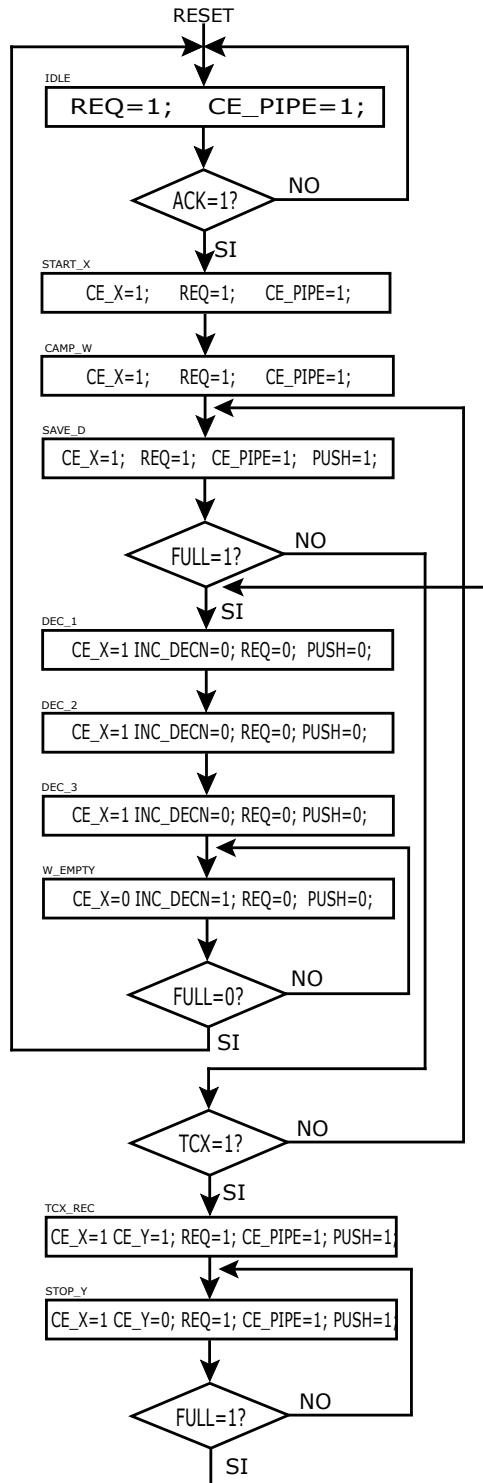


Figura 2.5: FSM del FIFO controller

## 2.5 Simulazione Modelsim

Per controllare la correttezza della FSM si è simulato il blocco FIFO controller in modo tale da capire se i timing diagram e il progetto fossero corretti ed in particolare che la sequenza degli stati fosse rispettata.

A tal proposito si è divisa la simulazione in tre parti come è stato fatto precedentemente per il timing:

Di seguito la simulazione della fase di caricamento della FIFO nei colpi di clock successivi al reset:

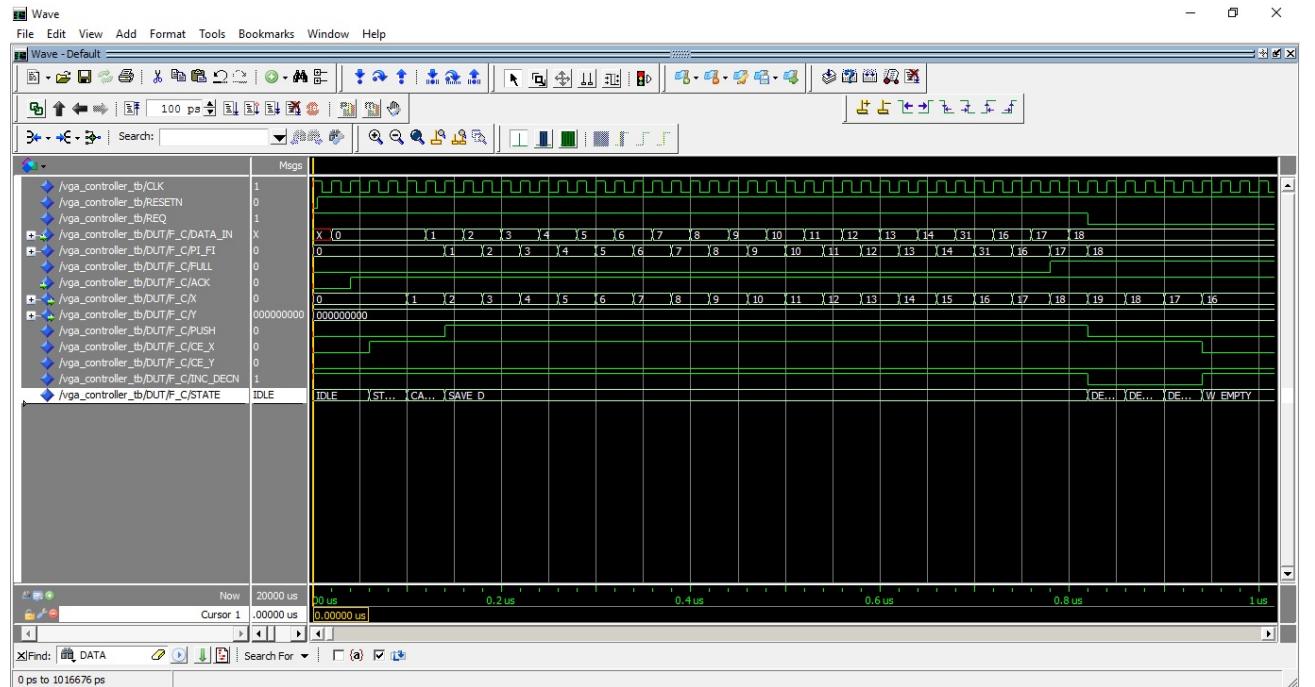


Figura 2.6: Evoluzione degli stati da IDLE a DEC\_3

La seguente simulazione documenta i colpi di clock che intercorrono tra quando il VGA sincr richiede dei dati a quando il FIFO controller riesce ad andare a regime e ad acquisire dalla memoria un dato ad ogni colpo di clock:

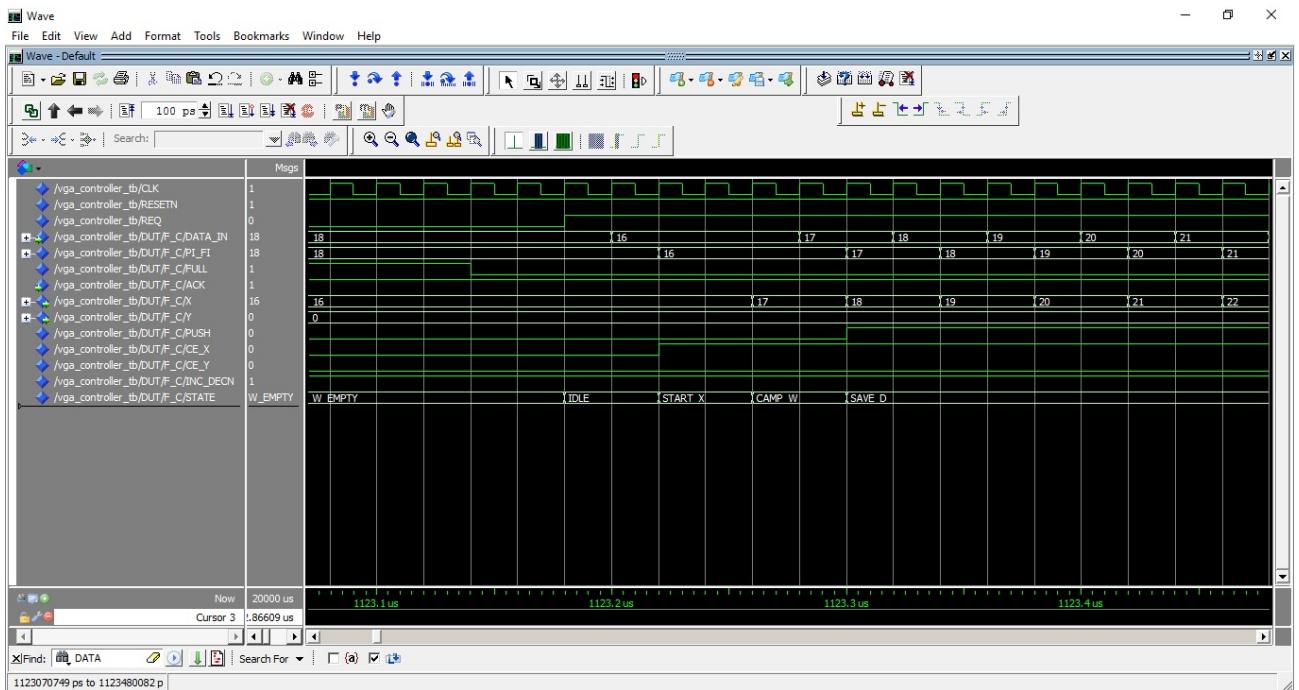


Figura 2.7: Evoluzione degli stati da W\_empty a camp\_w

La seguente simulazione illustra il completamento di una riga dello schermo ed il seguente incremento del contatore delle y:

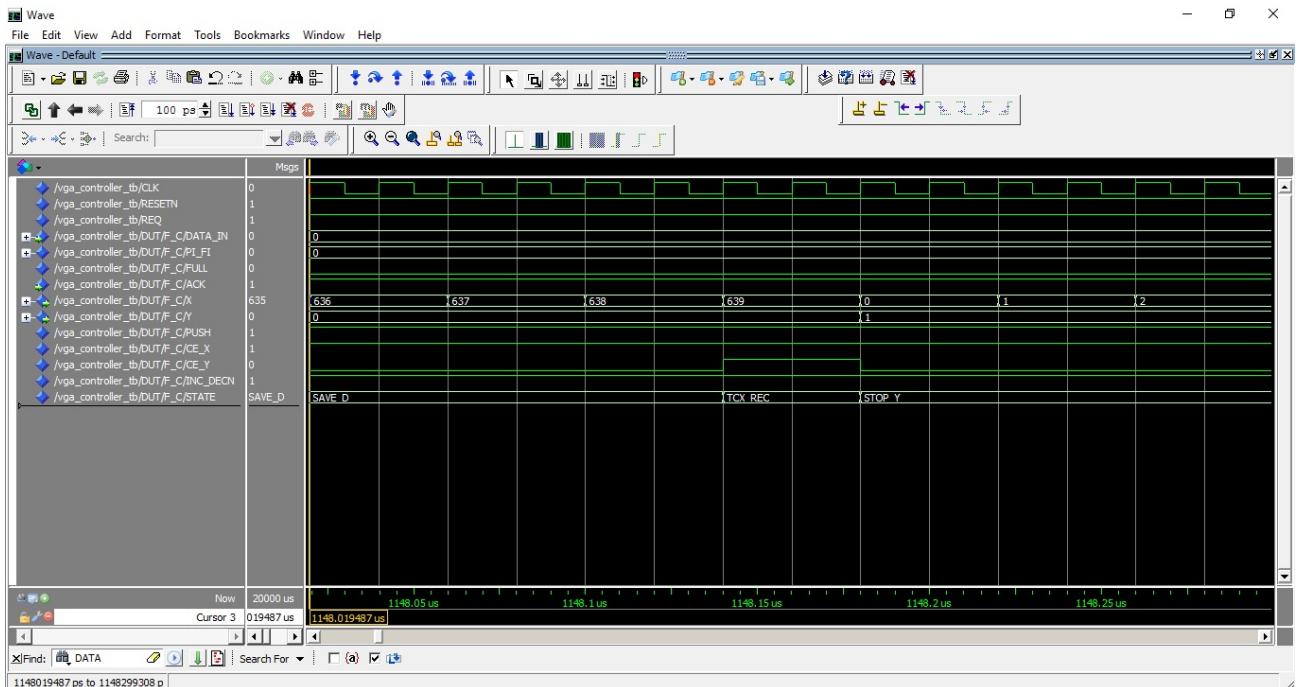


Figura 2.8: Evoluzione degli stati da save\_d a stop\_y

### 2.5.1 VGAsincr

Tale blocco deve garantire il giusto sincronismo orizzontale e verticale per il pilotaggio del monitor (VGA\_HS e VGA\_VS) e la generazione di segnali necessari sia al DAC (VGA\_CLOCK, VGA\_SYNC, VGA\_BLANK), che al FIFO\_CONTROLLER (POP).

Le specifiche per la sincronizzazione impongono il rispetto di determinati intervalli di tempo. Un impulso attivo-basso di durata  $3.8 \mu s$  (syncA) è applicato all'input di sincronizzazione orizzontale del monitor (HSYNC). Il suo significato è la fine di una riga e l'inizio della successiva. Segue un periodo di durata  $3.8 \mu s$  detto back porch (b) in cui HSYNC deve essere portato a '1' e un periodo (c) di 640  $3.8 \mu s$  in cui vengono pilotati i pixel di ogni riga. Durante tale intervallo il segnale VGA\_BLANK deve essere negato. Infine l'HSYNC deve rimanere a livello logico alto per  $0.6 \mu s$  (d). Il timing verticale (VSYNC) è il medesimo ma i periodi hanno durata di intere linee (ogni linea dura  $31.9 \mu s$ ) ma l'impulso iniziale indica la fine di un frame e l'inizio del successivo.

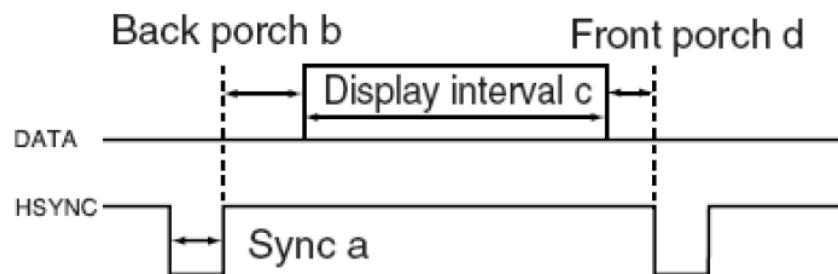


Figura 2.9: VGA timing specification

VGA mode		Horizontal Timing Spec				
Configuration	Resolution(HxV)	a(us)	b(us)	c(us)	d(us)	Pixel clock(Mhz)
VGA(60Hz)	640x480	3.8	1.9	25.4	0.6	25 (640/c)

Figura 2.10: Horizontal specification

VGA mode		Vertical Timing Spec			
Configuration	Resolution (HxV)	a(lines)	b(lines)	c(lines)	d(lines)
VGA(60Hz)	640x480	2	33	480	10

Figura 2.11: Vertical specification

## Data Path

Il *data path* è costituito da due contatori, uno che conta ogni 40 ns e l'altro che conta le linee (ogni linea dura 31.9  $\mu$ s. Entrambi possiedono vari terminal count in modo da poter differenziare i vari intervalli presenti nelle specifiche di timing. Per trasformare i tempi da aspettare in colpi di clock si è usata la seguente formula:

$$n_{clock} = \frac{\text{intervallo}}{40\text{ns}} \quad (2.1)$$

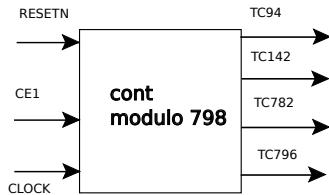


Figura 2.12: Contatore modulo 798

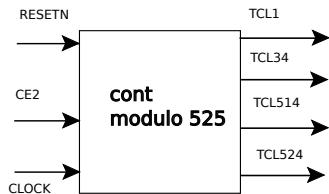


Figura 2.13: Contatore linee modulo 525

## Timing Diagram

Di seguito verranno riportati i timing diagram riguardanti la sincronizzazione orizzontale e verticale. Per quest'ultima il diagramma è stato diviso in tre macroblocchi per evidenziare i diversi intervalli.

### Horizontal Timing : sincronismo orizzontale

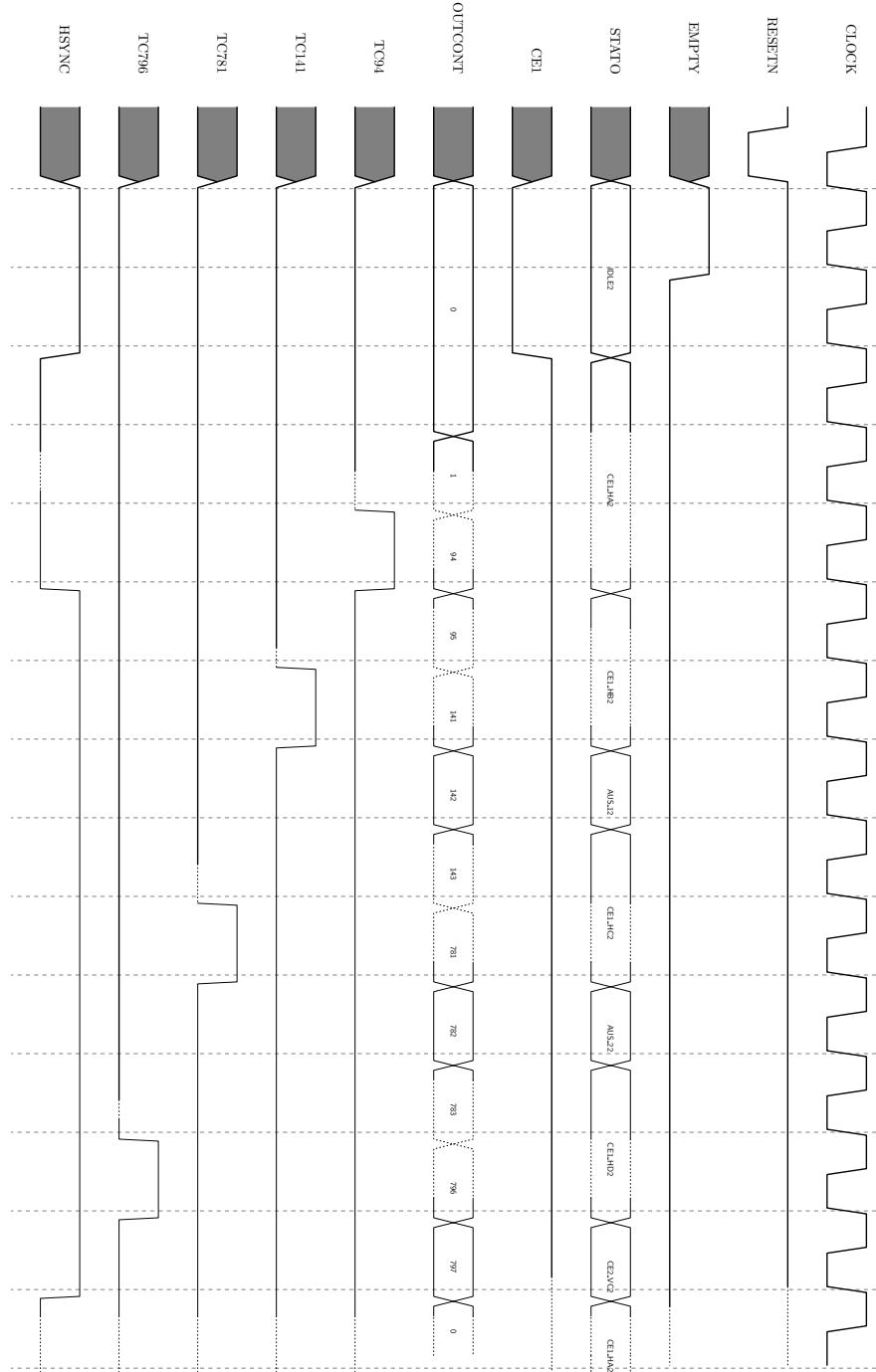


Figura 2.14: Timing sincronismo orizzontale

## Vertical Timing

**Vertical timing: syncA e back porch b** La figura sottostante illustra il timing dureante gli intervalli synca e back porch b.

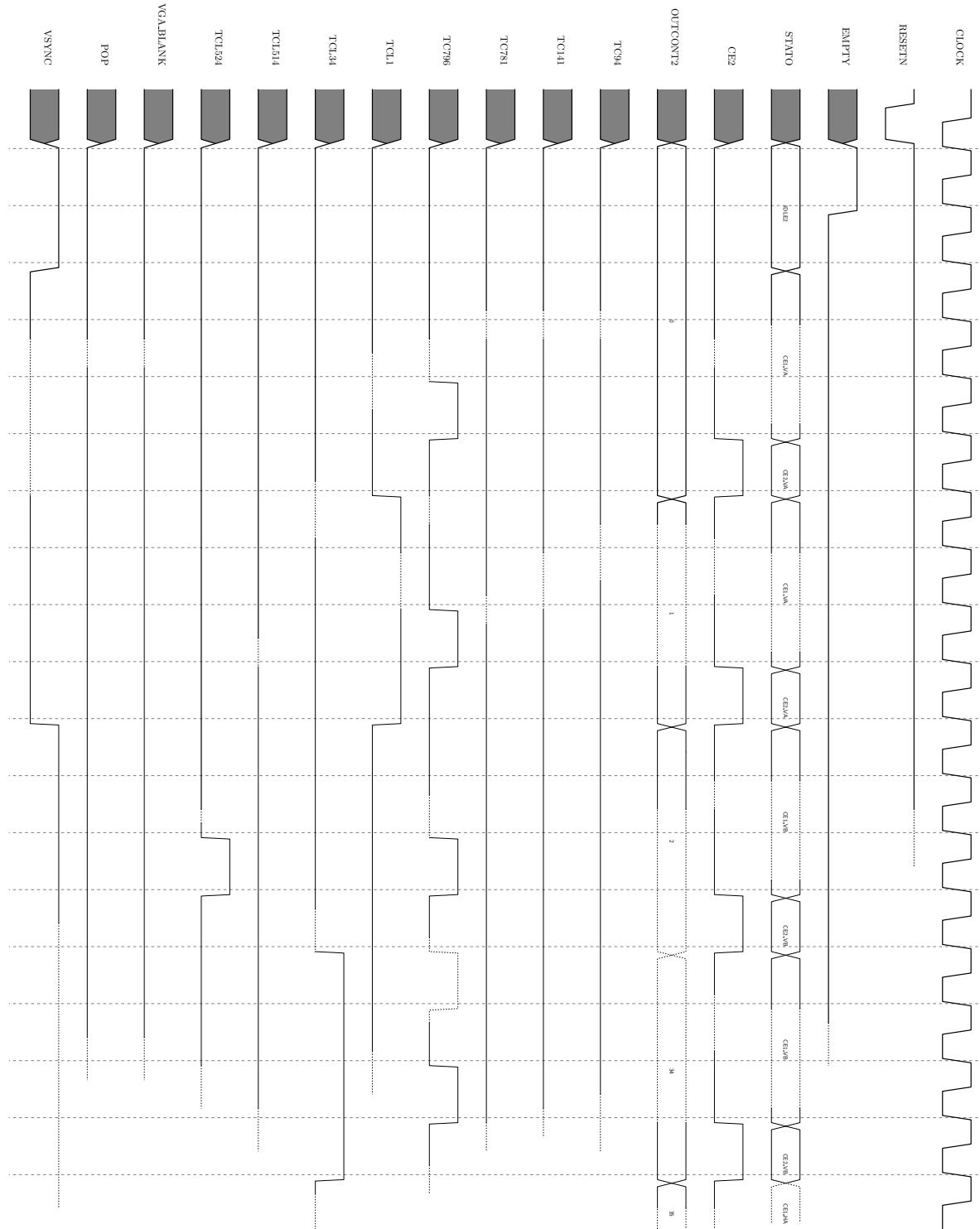


Figura 2.15: Timing sincronismo verticale

**Vertical timing: interval c.** Tale blocco illustra il cambiamento dei segnali coinvolti durante l'intervallo c. Esso si ripete 480 volte, corrispondenti al numero di righe da scansionare.

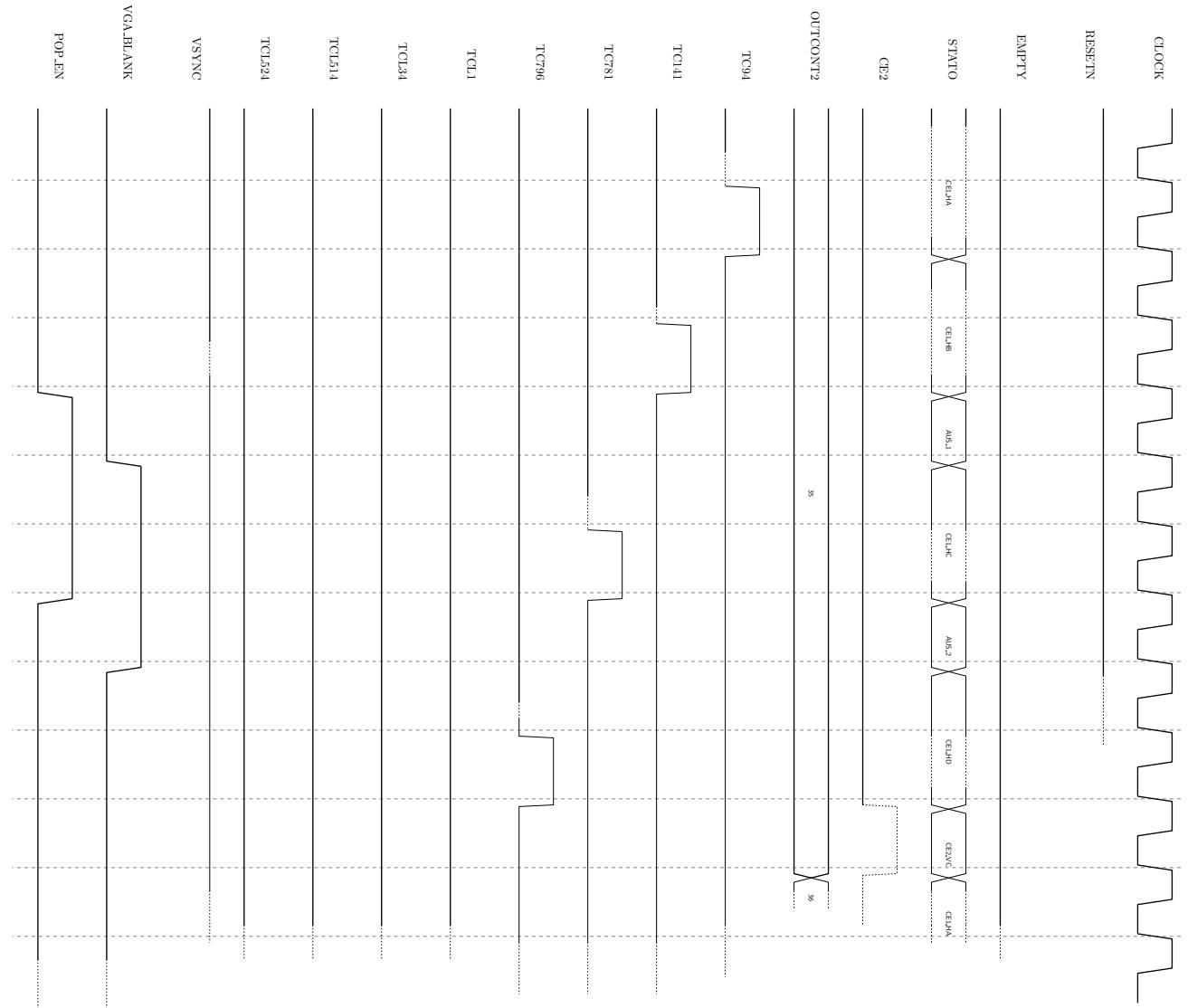


Figura 2.16: Timing sincronismo verticale

**Vertical timing: front porch d** Il diagramma sottostante illustra il comportamento della macchina durante il front porch d e la sua ripartenza dal sync a.

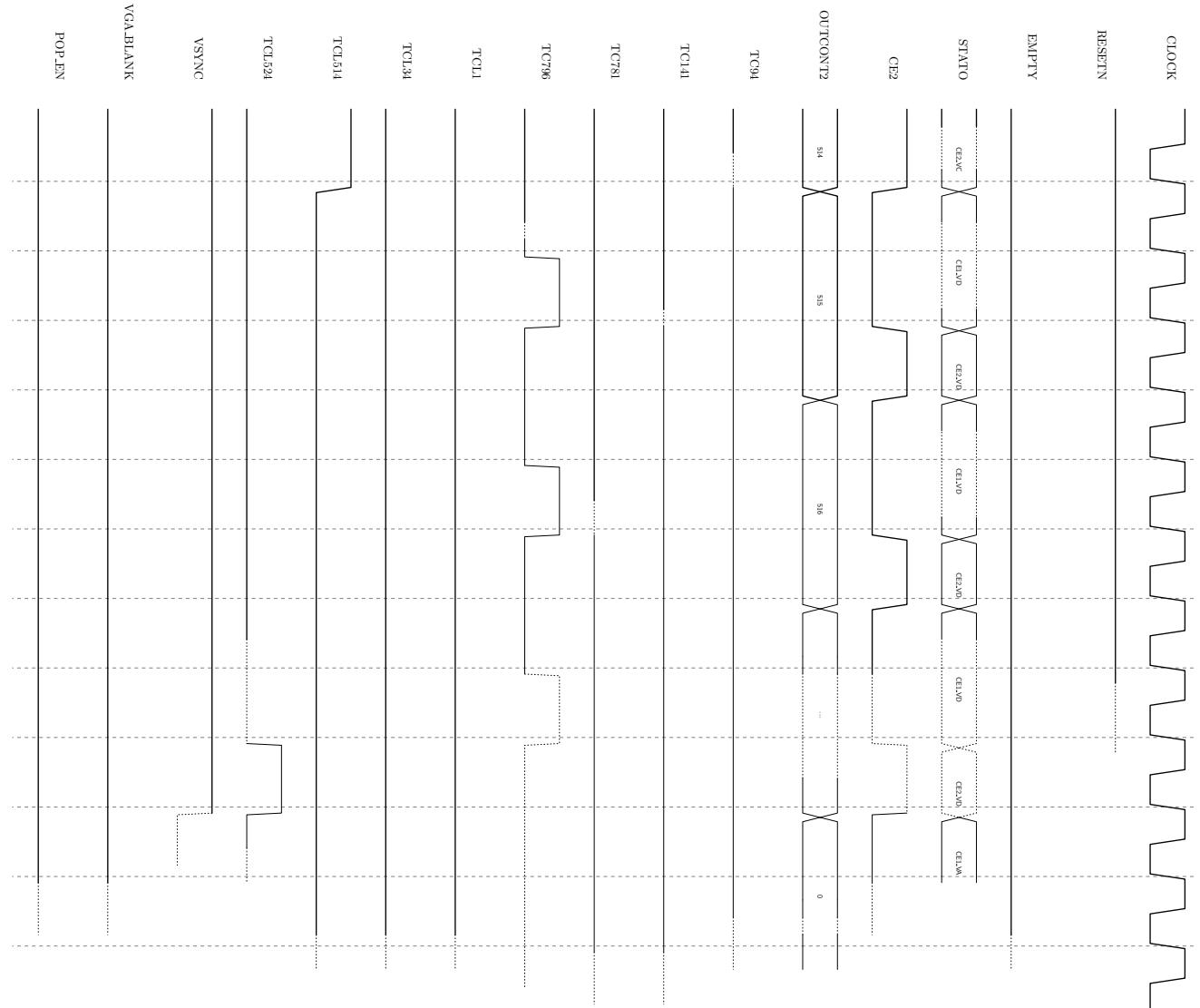


Figura 2.17: Timing sincronismo verticale

## FSM

Si è deciso di implementare due FSM separate per controllare il sincronismo verticale e il sincronismo orizzontale. La prima controlla VSYNC, i segnali da inviare al DAC e abilita il contatore di linee(CE2), la seconda HSYNC e il contatore a 25 MHz (CE1).

**Horizontal FSM** Di seguito la FSM relativa al sincronismo orizzontale:

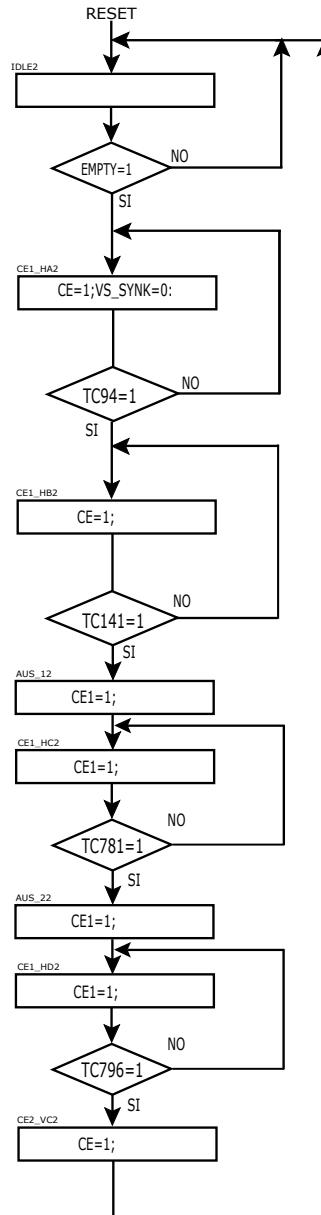


Figura 2.18: Horizontal FSM

**Vertical FSM** Di seguito la FSM relativa al sincronismo verticale:

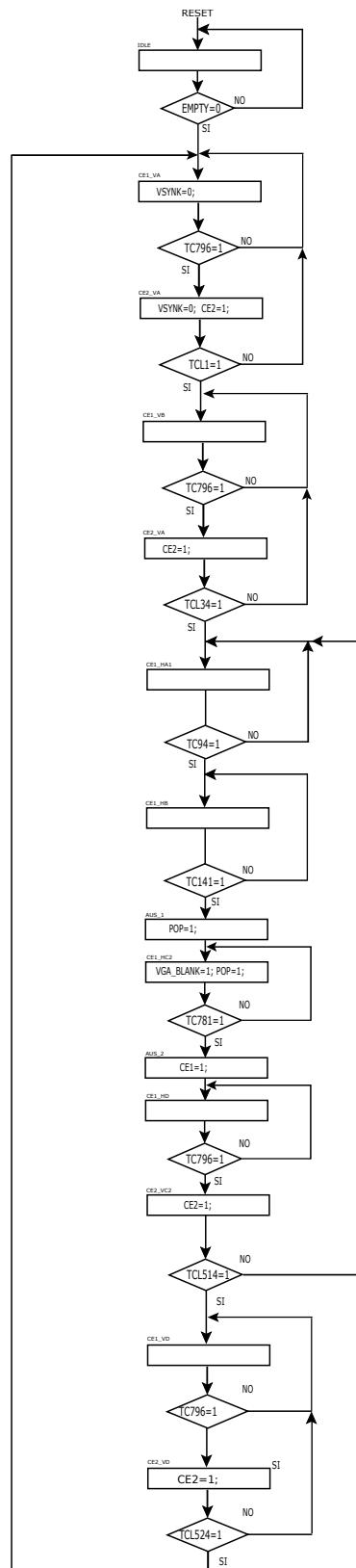


Figura 2.19: Vertical FSM

## 2.6 Simulazione Modelsim

Poichè bisogna rispettare dei tempi imposti dallo standard VGA si sono fatte diverse simulazioni sulle due FSM in modo da quantificare i tempi e controllare che l'evoluzione degli stati fosse giusta.

A tal proposito si espongono in seguito le simulazioni relative prima alla gestione del sincronismo orizzontale e dopo quello verticale.

Le seguenti tre simulazioni riguardano gli stati coperti dalla FSM del sincronismo orizzontale:

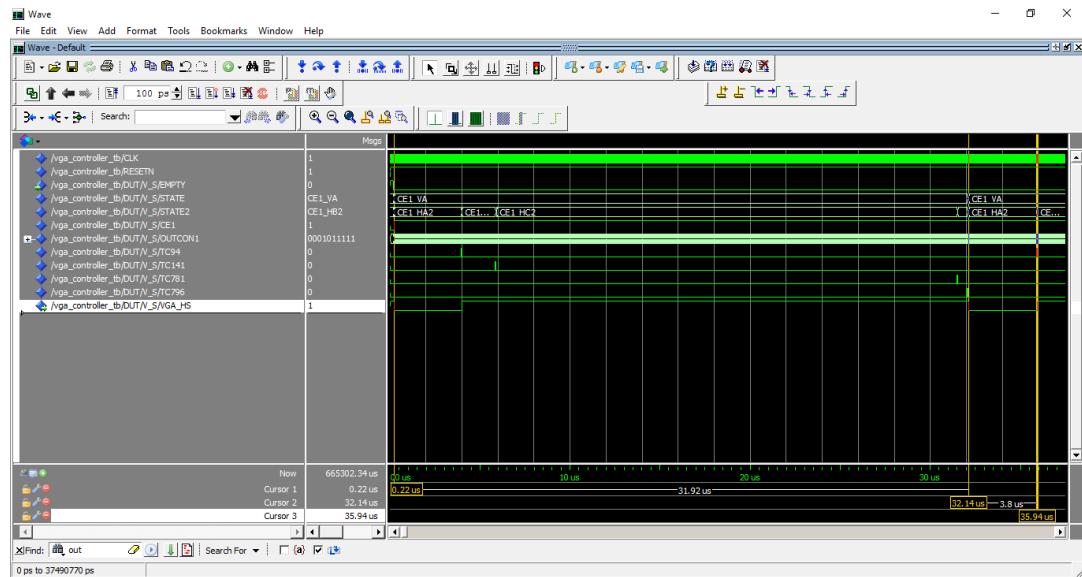


Figura 2.20: Misura dei tempi del sincronismo orizzontale

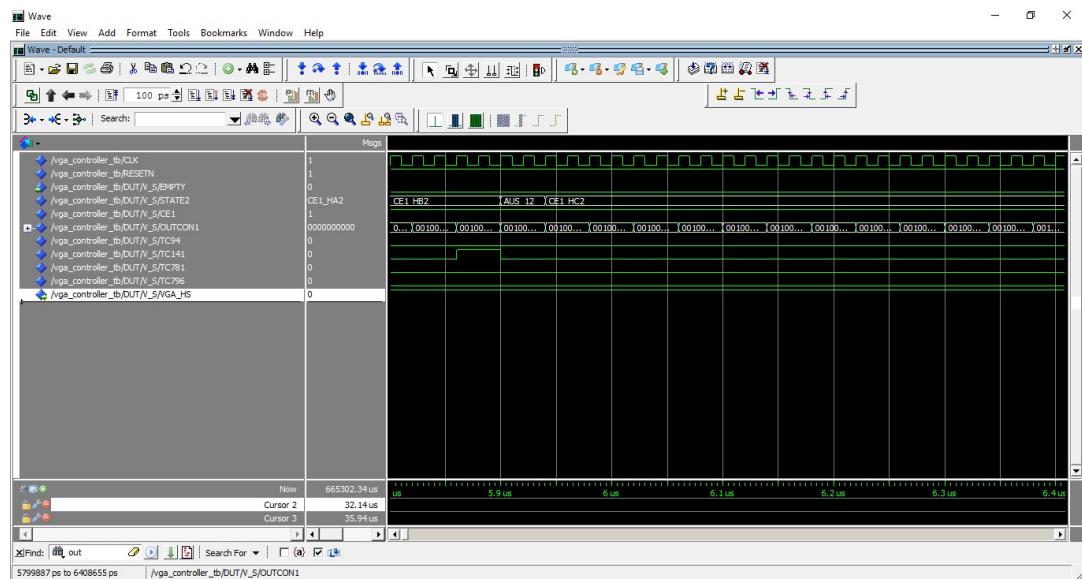


Figura 2.21: Evoluzione degli stati da ce1\_hb2 a ce1\_mc2

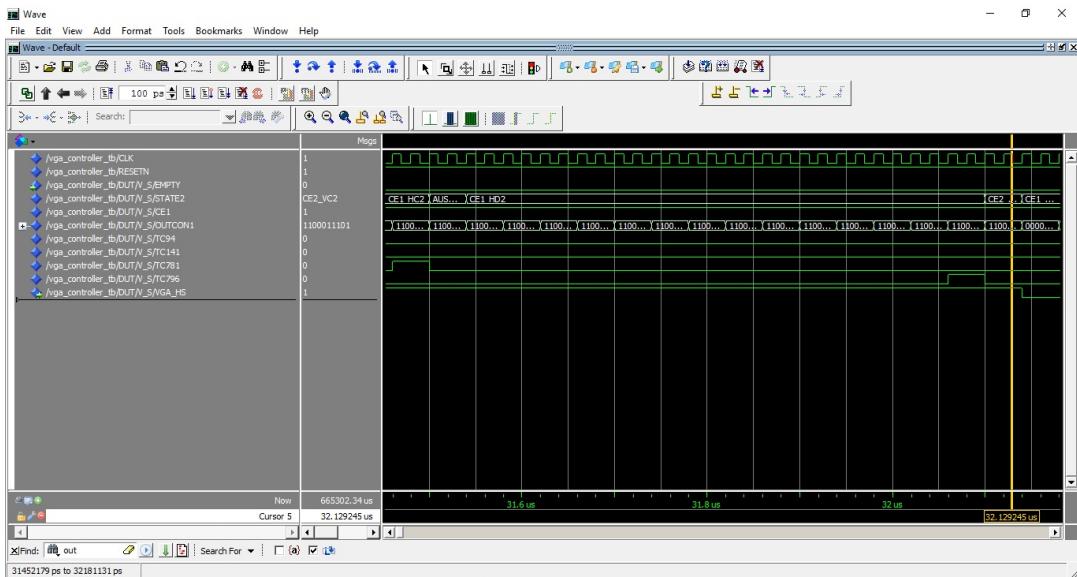


Figura 2.22: Evoluzione degli stati da ce1\_hc2 a ce2\_vc2

Le simulazioni sottostanti invece sono riferite alla FSM che genera il sincronismo verticale. Dato che ingloba il sincronismo orizzontale è stata suddivisa in più parti: Di seguito è riportata la simulazione con la misura di una ripetizione del sincronismo verticale

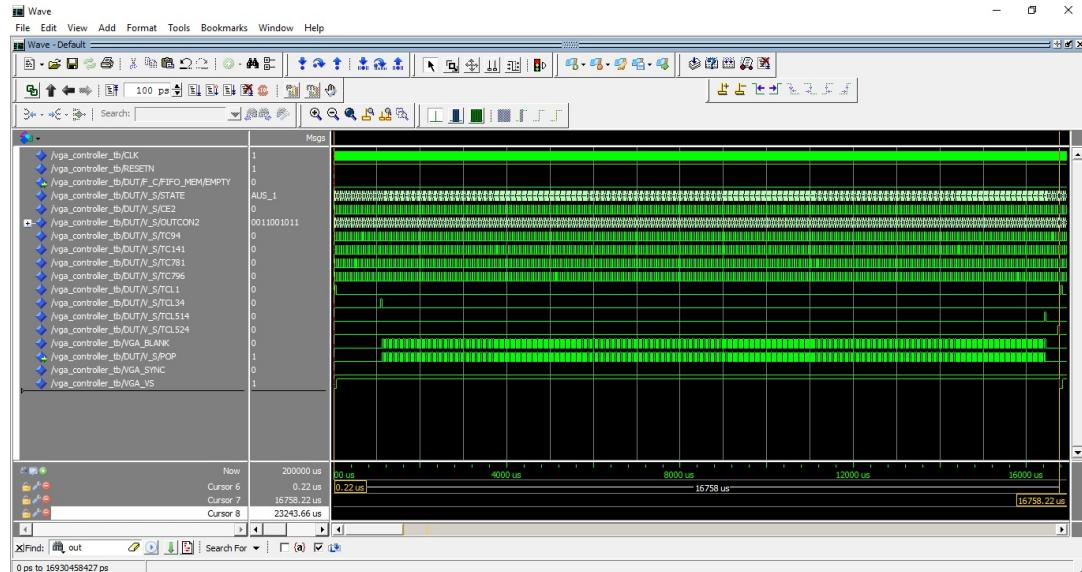


Figura 2.23: Misura della durata del sincronismo verticale

La simulazione sottostante documenta la durata del SYNC A

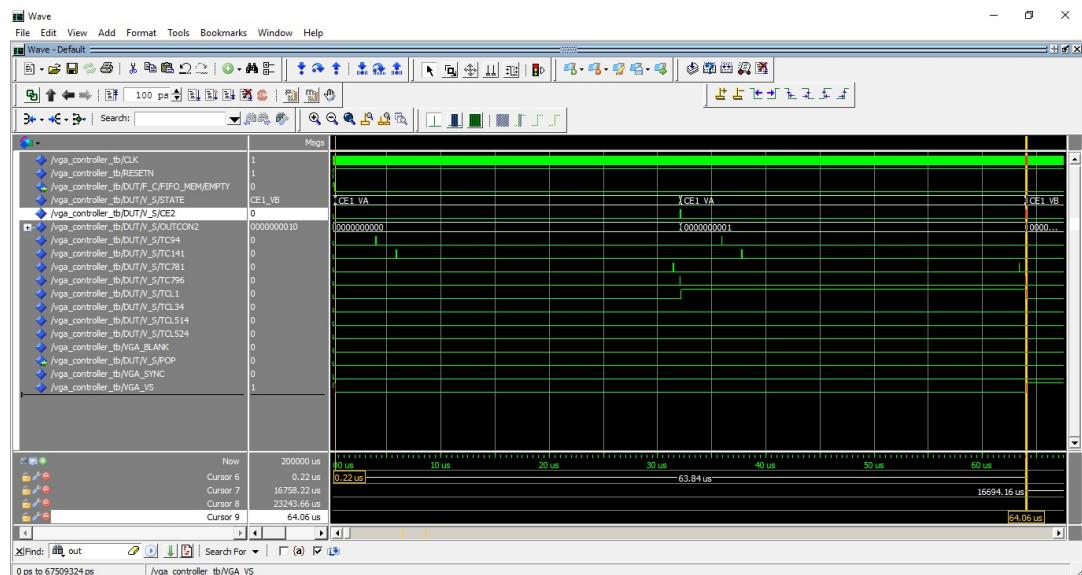


Figura 2.24: Misura della durata del SYNC A

Per un controllo più facile di seguito sono riportati degli screen della simulazione che ricalcano le sequenze riportate nel timing diagram con gli stessi segnali e gli stessi stati:

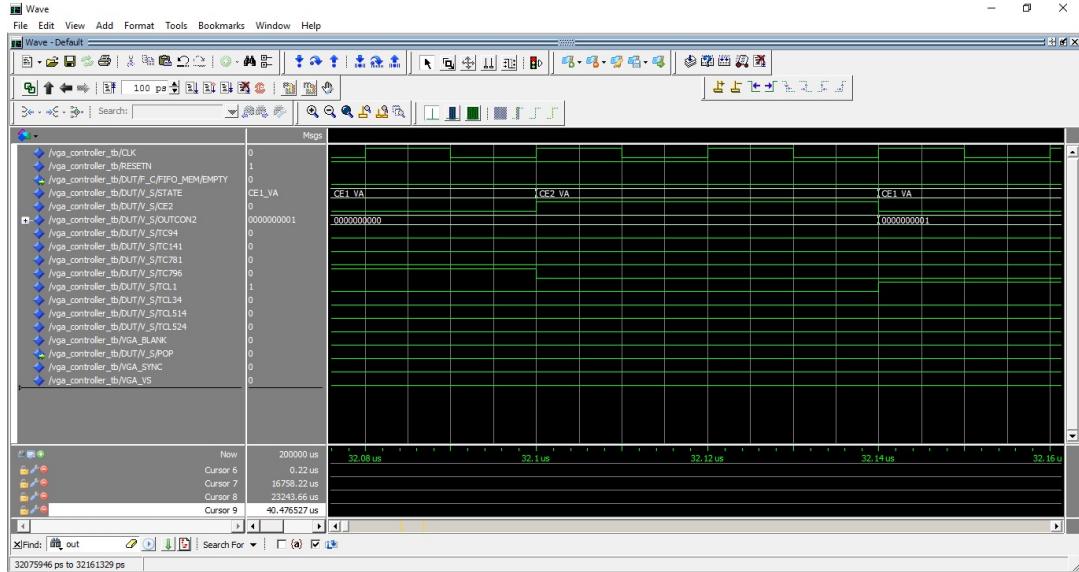


Figura 2.25: Evoluzione degli stati da ce1\_va -> ce2\_va -> ce1\_va

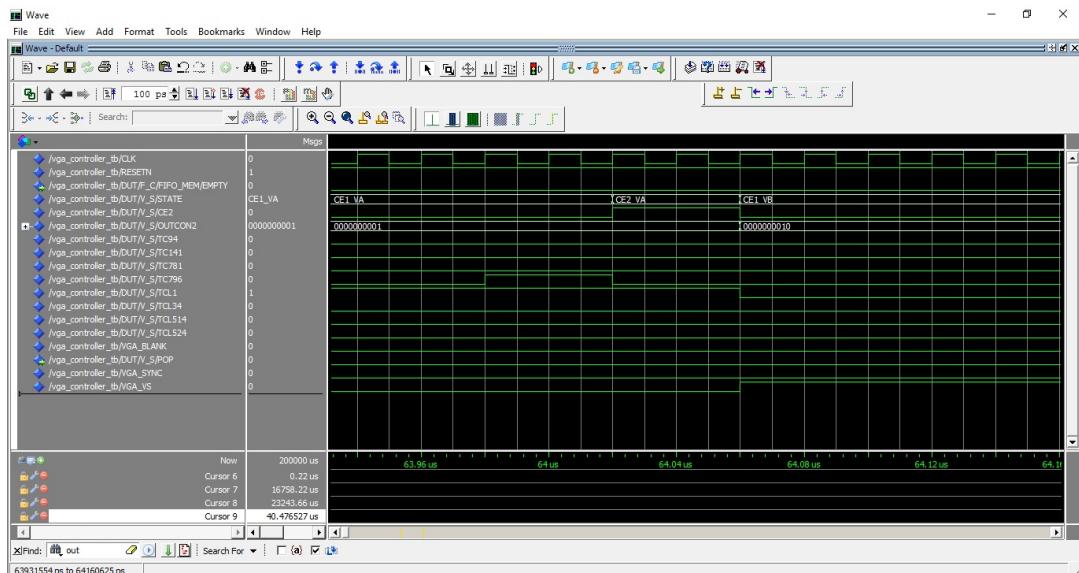


Figura 2.26: Evoluzione degli stati da ce1\_va -> ce2\_va -> ce1\_vb

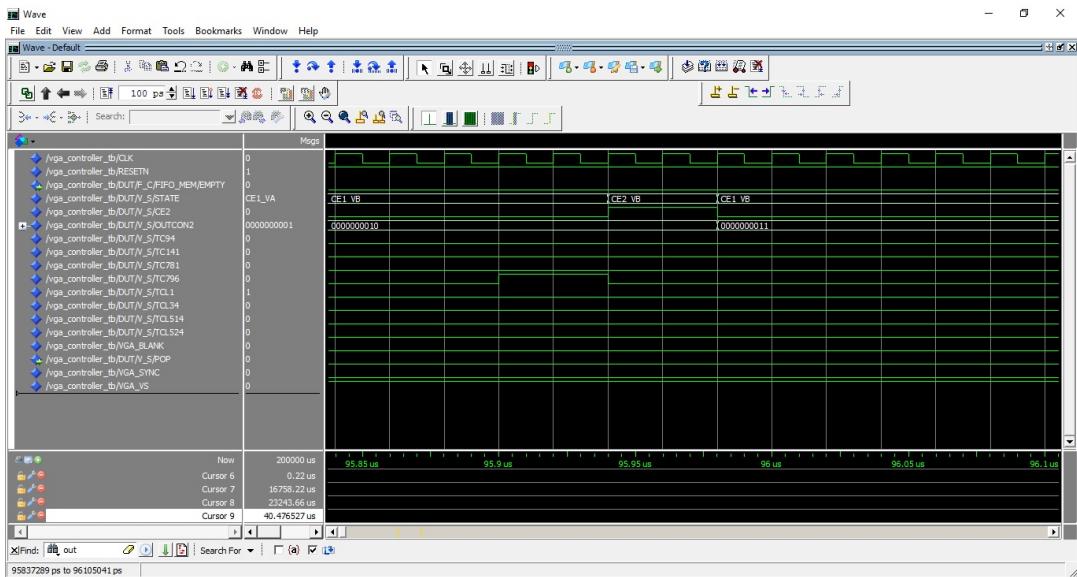


Figura 2.27: Evoluzione degli stati da ce1\_vb -> ce2\_vb -> ce1\_vb

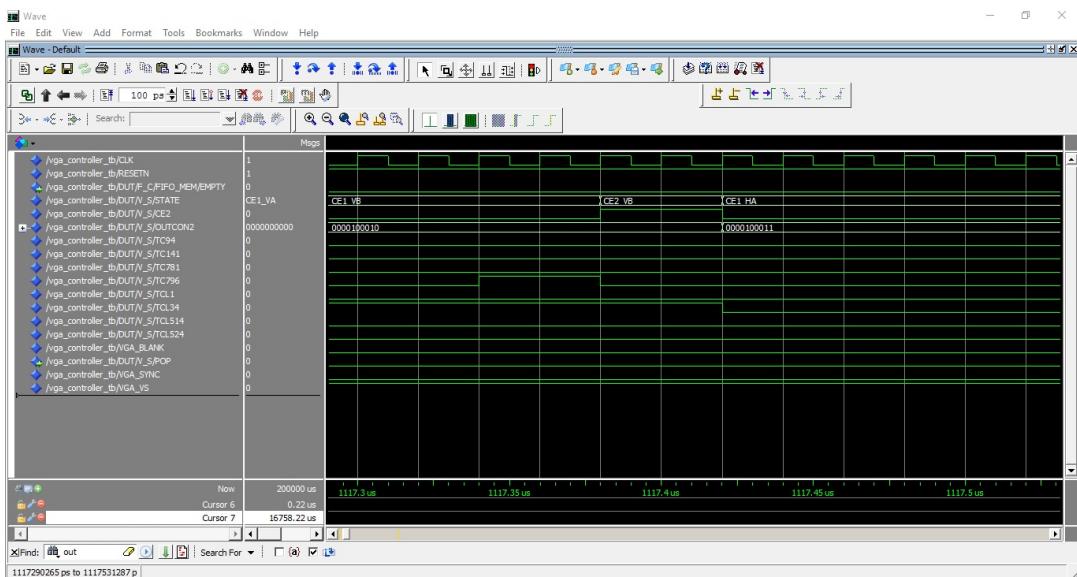


Figura 2.28: Evoluzione degli stati da ce1\_vb -> ce2\_vb -> ce1\_vb

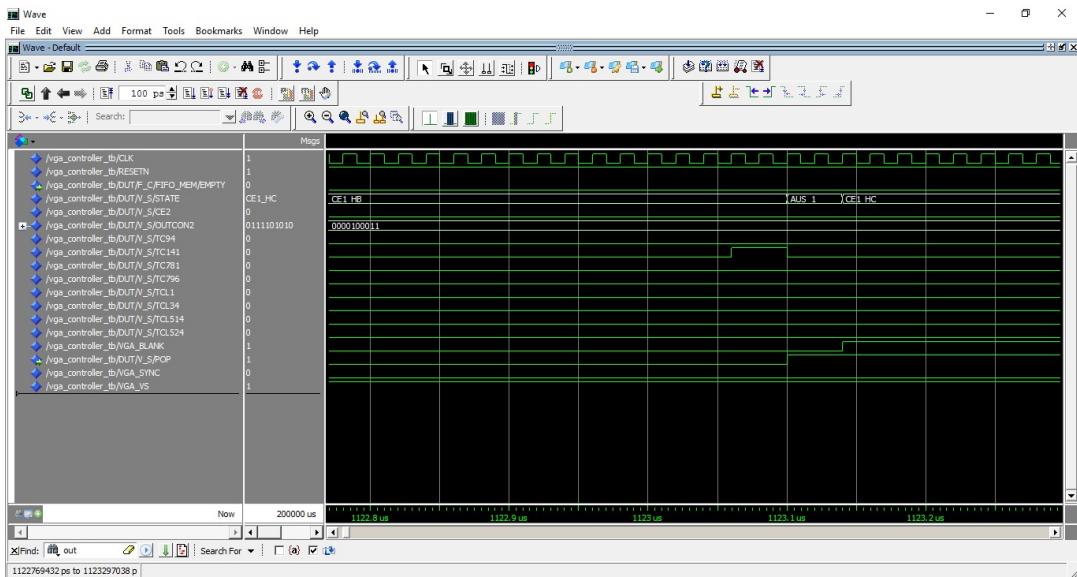


Figura 2.29: Evoluzione degli stati da ce1\_hb -> aus\_1 -> ce1\_hd

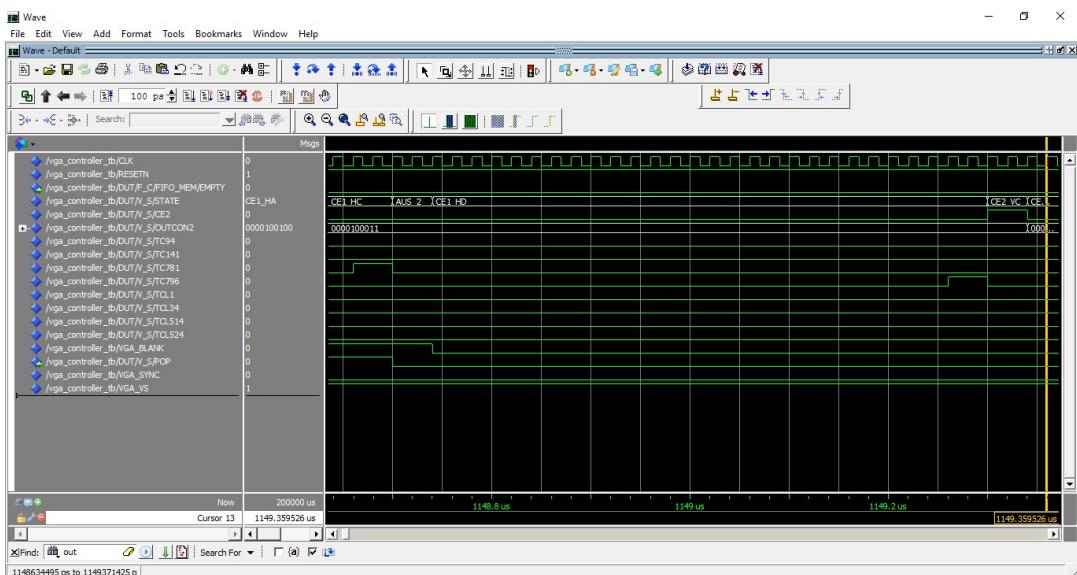


Figura 2.30: Evoluzione degli stati da ce1\_hd -> aus\_2 -> ce1\_hd -> ce2\_vc -> ce1\_ha

Di seguito la simulazione relativa al front porch d:

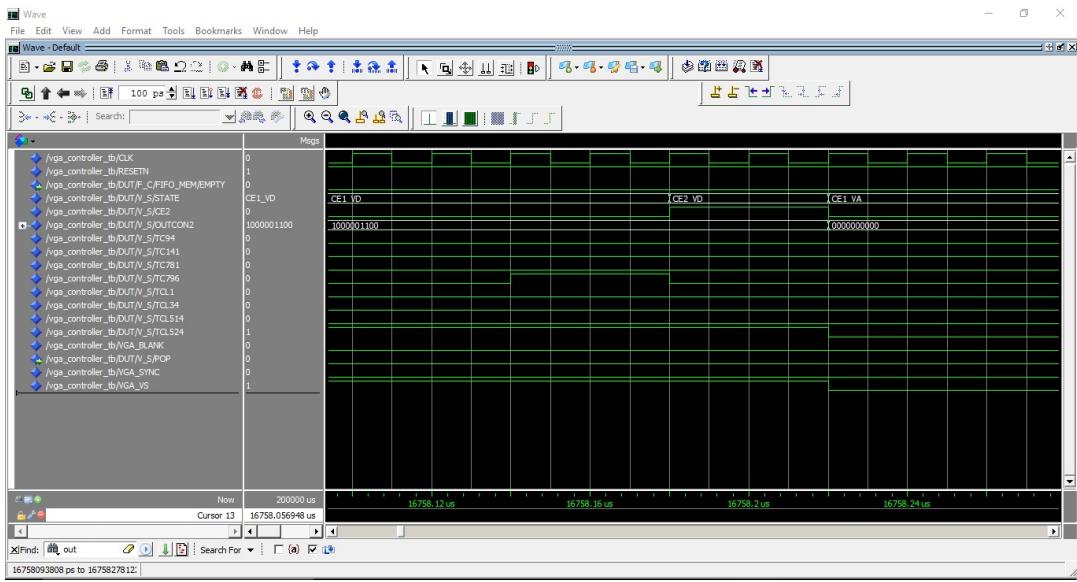


Figura 2.31: Evoluzione degli stati da ce1\_vd -> ce2\_vd -> ce1\_va

## 2.7 Test in laboratorio

Per testare il VGA controller è stata creata una entity che contiene FIFO controller e VGAsincr. Tra gli ingressi di questa entity sono inclusi anche degli switch mediante i quali si modificano i colori da inviare allo schermo.

Inizialmente si sono visualizzati i segnali di sincronismo mediante l'oscilloscopio:

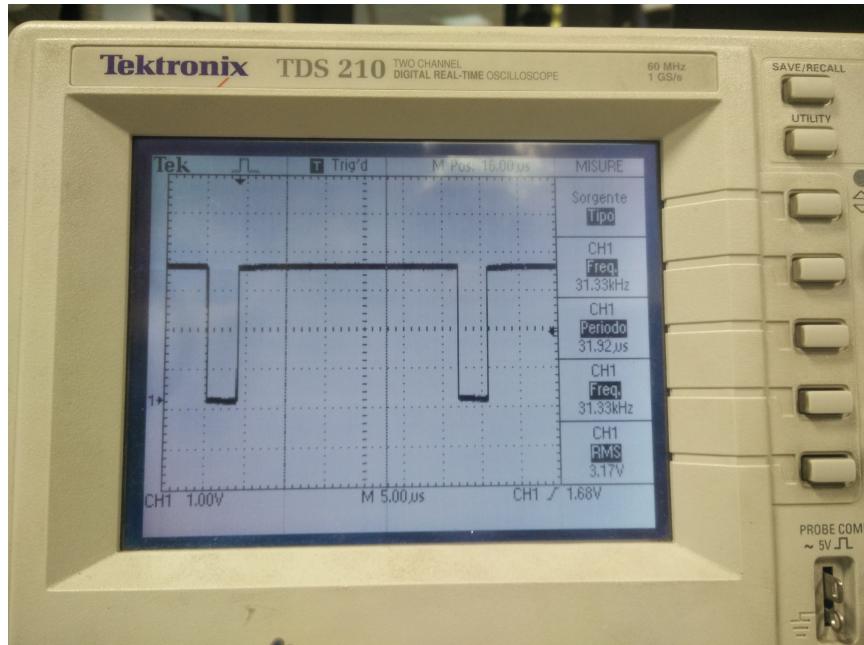


Figura 2.32: Sincronismo orizzontale mediante oscilloscopio

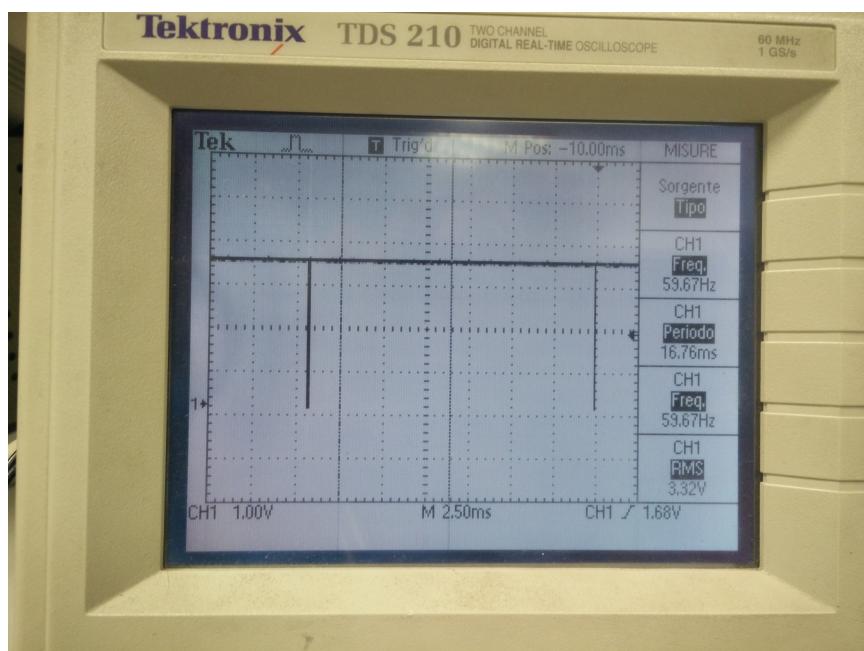


Figura 2.33: Sincronismo orizzontale mediante oscilloscopio

Successivamente poi si è collegato lo schermo alla scheda DE2 e si sono variati gli switch in modo tale da variare i bit più significativi dei colori ottenendo la variazione del colore dello schermo:

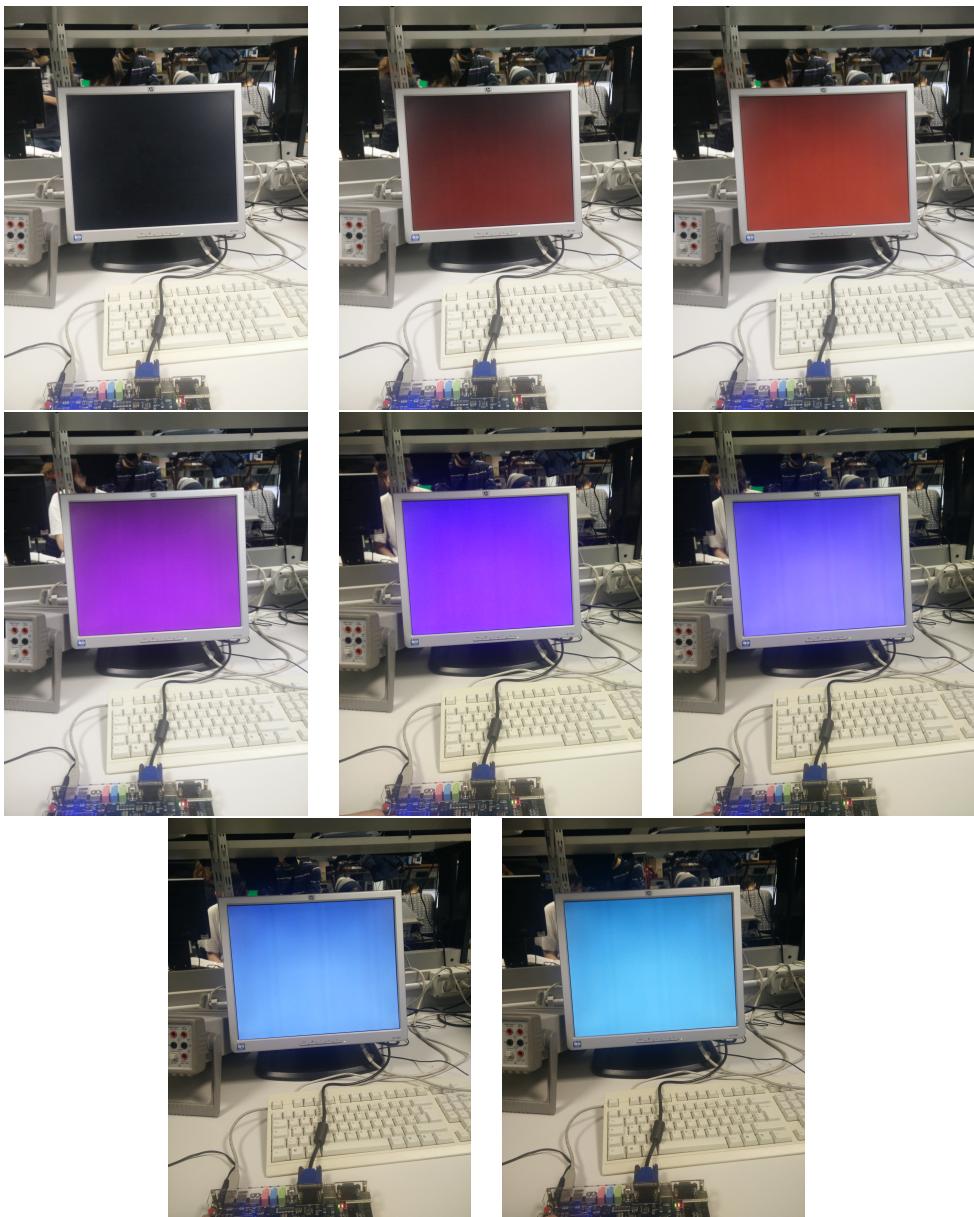


Figura 2.34: Colori visualizzati sul display variando gli switch

# Capitolo 3

## Memory Controller

### 3.1 Specifiche

La macchina denominata "Memory Controller" ha il compito di ricevere le richieste provenienti sia dall' UART che dal "VGA Controller" e di interfacciarsi con la SRAM presente sulla scheda "Altera DE2".

In caso di richieste contemporanee un arbitro deve far prevalere quella proveniente dal "VGA Controller" ed, inoltre, un algoritmo deve convertire le coordinate X ed Y dell'immagine che si vuol rappresentare in un codice binario che può essere identificato come *Address* dalla memoria.

In figura 3.1 è rappresentata una *black box* della macchina in questione.

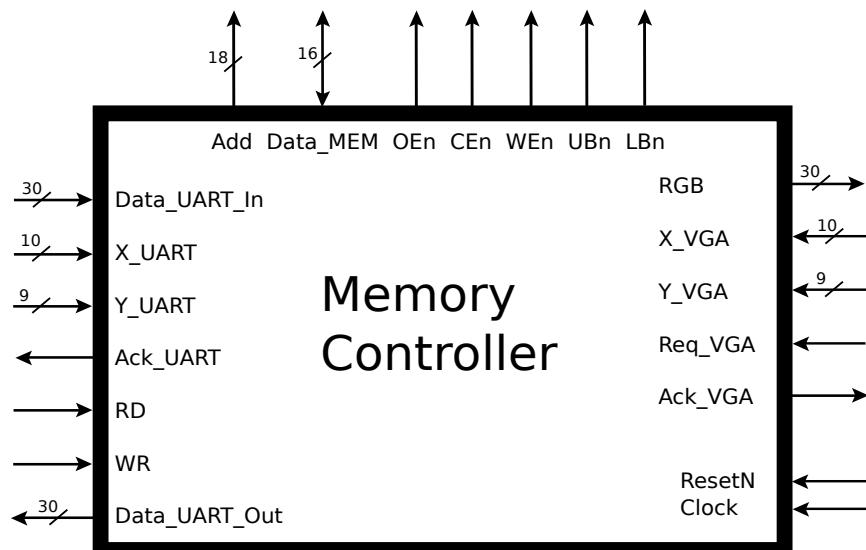


Figura 3.1: Memory Controller Black Box

### 3.2 Progetto

La macchina è stata suddivisa in due macchine a stati più semplici.

La prima è stata denominata "Arbitro" ed ha il compito di prelevare tutti i segnali di

ingresso del "Memory Controller" e gestire l'arbitraggio delle richieste di UART e "VGA Controller".

La seconda, invece, è stata denominata "Memory Interface" in quanto ha il compito di interfacciarsi direttamente con la memoria. Essa preleva i valori di X e Y in uscita dall'"Arbitro" e li converte in indirizzo da inviare alla memoria, oltre agli altri segnali a seconda se sia stata richiesta una lettura o una scrittura in memoria.

### 3.2.1 Arbitro

In figura 3.2 viene mostrata la *black box* dell'"Arbitro".

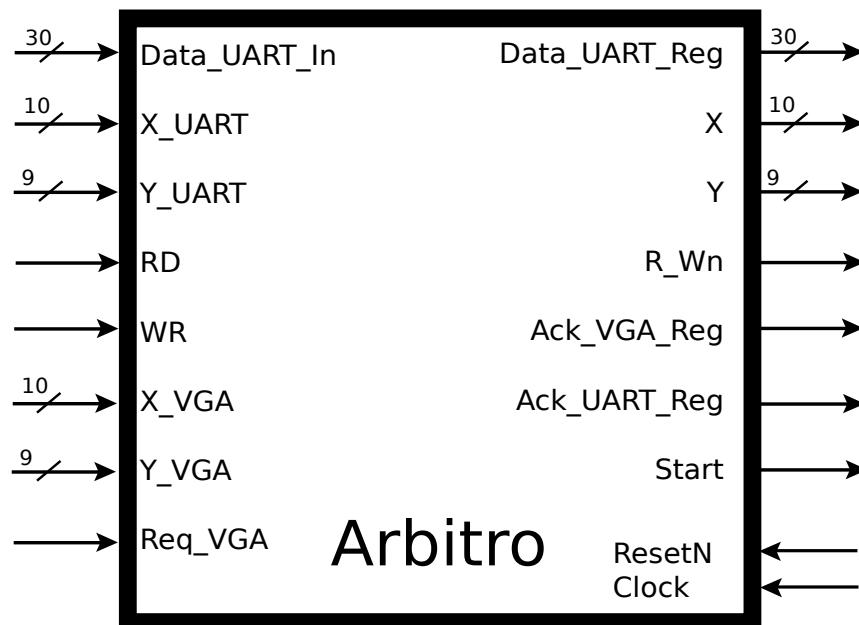


Figura 3.2: Arbitro Black Box

Essa preleva tutti i segnali di ingresso del "Memory Controller" e come uscite fornisce i segnali di X e Y che, a seconda delle richieste, sono uguali o a quelli provenienti dall'UART o dal "VGA Controller", il segnale "Data\_UART\_Reg" che verrà poi campionato da un registro nel "Memory Interface" per avere un livello di *pipe* ed i segnali "Ack\_UART\_Reg" ed "Ack\_VGA\_Reg" che vengono asseriti quando l'arbitro ha fatto prevalere la relativa richiesta e che, per il medesimo motivo del "Data\_UART\_Reg", verranno successivamente campionati da due registri.

### Data Path

In figura 3.3 è rappresentato il *data path* della macchina a stati denominata "arbitro". Affinché i dati vengano interpretati correttamente tutti i segnali di ingresso vengono temporizzati da un registro ciascuno, denominati "Reg1", "Reg2" fino a "Reg9". I segnali "RD" e "WR", che indicano rispettivamente una richiesta di lettura e di scrittura da parte dell'UART, vanno in una porta "or" per generare il segnale "Req\_UART",

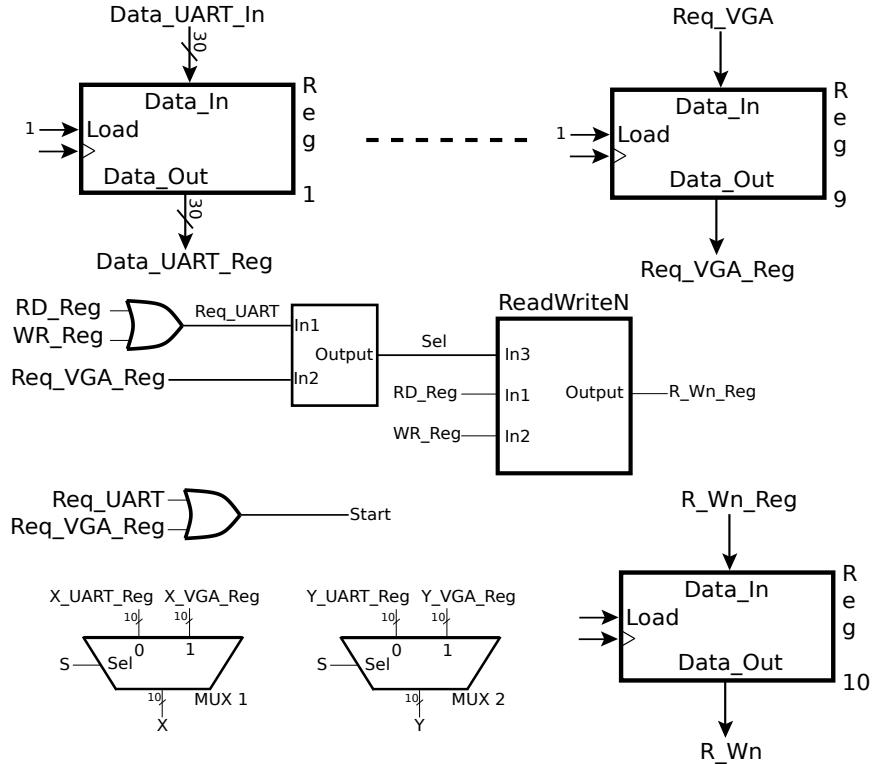


Figura 3.3: Arbitro Data Path

che a sua volta va in una seconda porta "or" con "Req\_VGA\_Reg" per definire il segnale di "Start" utilizzato per far uscire la macchina a stati dallo stato di "Idle", ovvero di inattività.

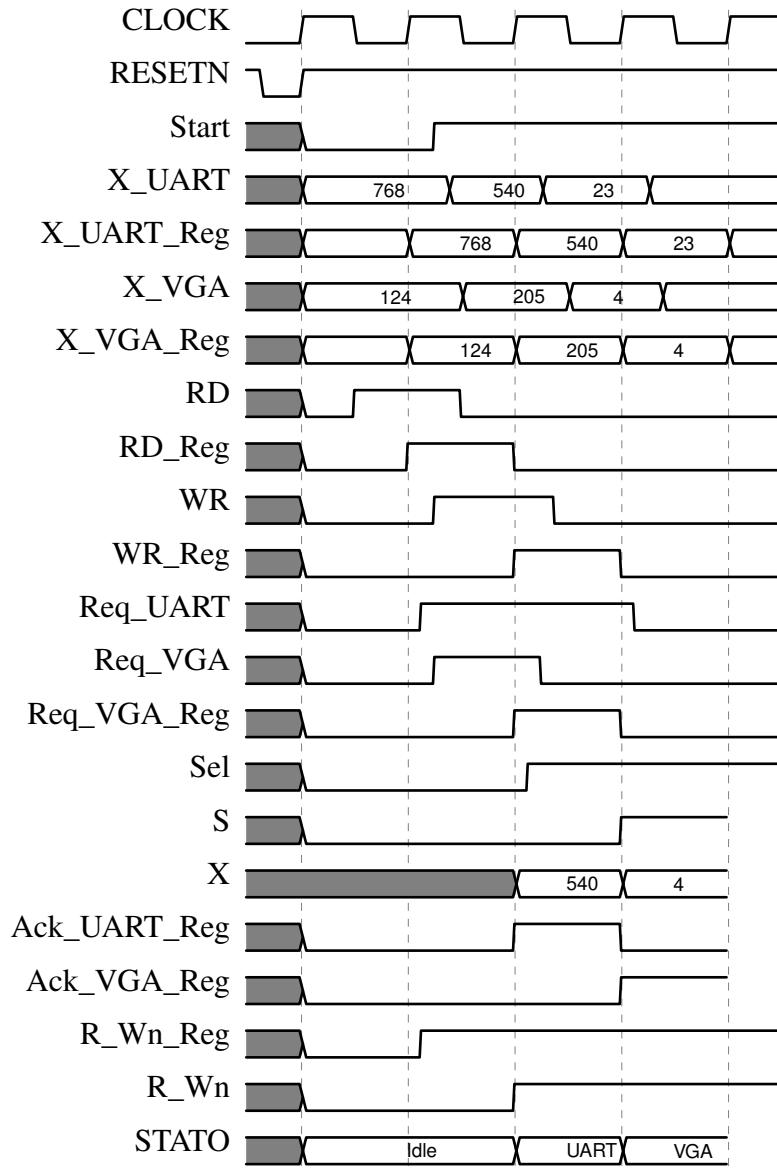
I due segnali di richiesta vengono portati agli ingressi di un blocco combinatorio, la cui uscita vale '0' in caso di assenza di richiesta o richiesta da parte dall'UART, mentre vale '1' in caso di richiesta del VGA o di richiesta contemporanea. Dall'analisi della tabella di verità si riscontra che "Output" = "In2", segnale che prende il nome di "Sel". In base al valore di tale segnale la *control unit* stabilisce il valore di "S" utilizzato come selettore dei due multiplexer e quindi se portare in uscita le coordinate provenienti dall'UART o dal "VGA Controller".

Inoltre, tale segnale viene anche portato all'ingresso di un secondo circuito combinatorio insieme ai segnali "RD\_Reg" e "WR\_Reg" e denominato "ReadWriteN", la cui uscita vale '0' quando è stata richiesta una scrittura ed '1' quando invece è stata richiesta una lettura. In figura 3.4 viene mostrata la tabella di verità di tale circuito.

In1	In2	In3	Out
-	-	1	1
1	0	0	1
0	1	0	0
0	0	0	1
1	1	0	1

Figura 3.4: ReadWriteN Truth Table

### Timing Diagram



## FSM

In figura 3.5 viene rappresentata la *finite state machine* dell'"Arbitro".

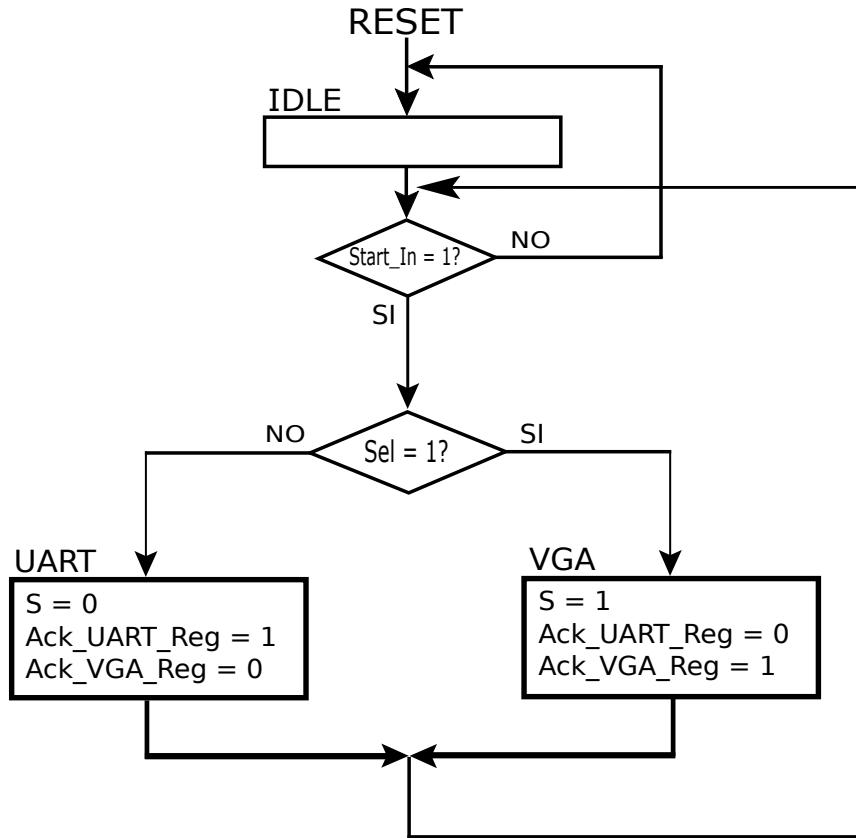


Figura 3.5: Arbitro Finite State Machine

Essa consta di soli tre stati: uno di inattività, denominato "Idle", e due che prendono il nome della richiesta che in quel ciclo di clock è stata servita. Al termine di questi due stati la macchina finisce nello stato di "Idle" solo se non vi è stata un'altra richiesta.

Il corretto funzionamento dell'arbitro è stato controllato mediante simulazione Modelsim:

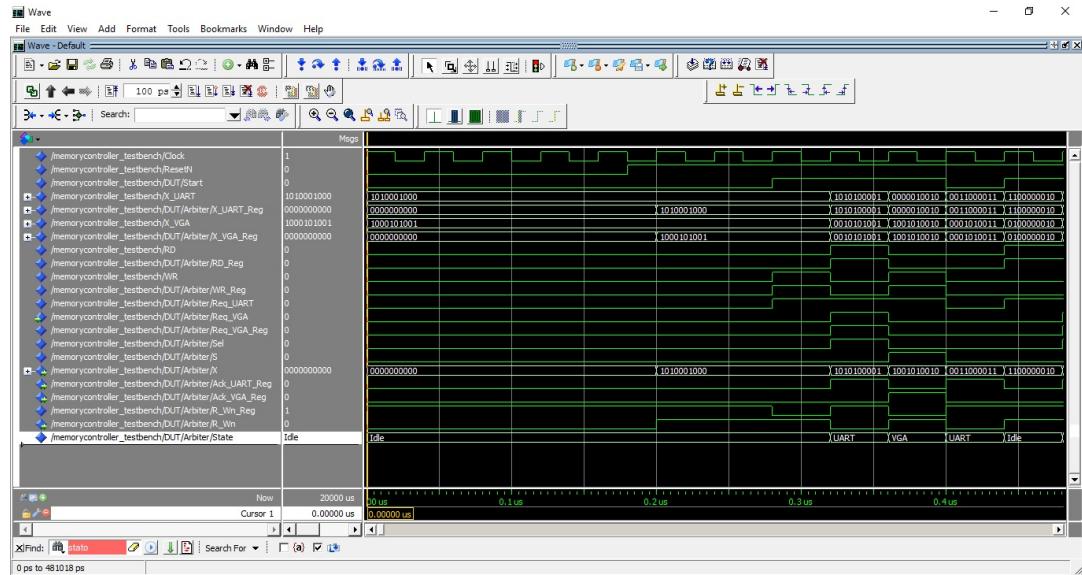


Figura 3.6: Simulazione con Modelsim della FSM relativa all'arbitro

### 3.2.2 Memory Interface

In figura 3.7 è raffigurata la *black box* del "Memory Interface".

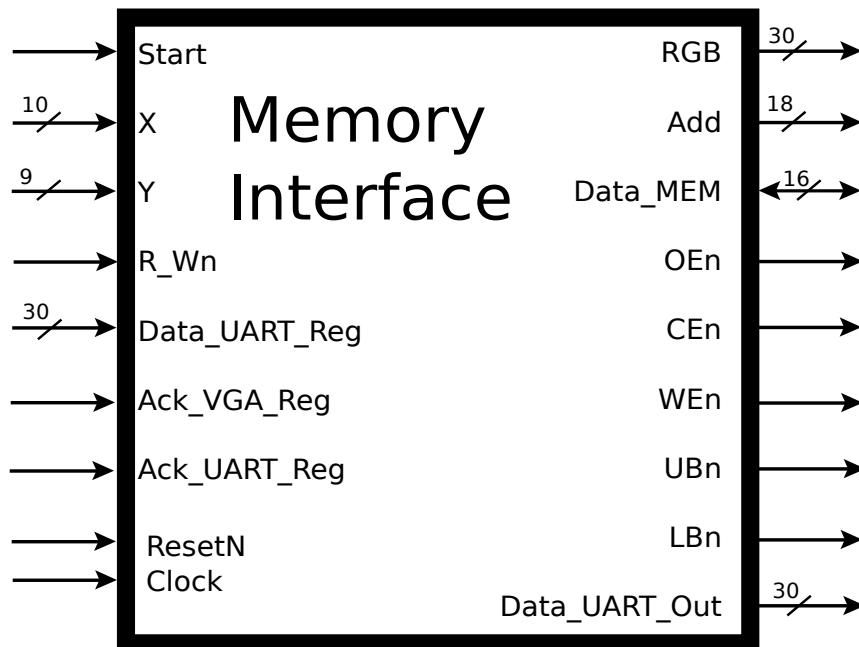


Figura 3.7: Memory Interface Black Box

Essa ha come ingressi tutte le uscite dell'"Arbitro" ed in uscita porta tutti i segnali da inviare alla memoria più i segnali "RGB" ed "Data\_UART\_Out", i quali vengono inviati rispettivamente a VGA e UART e rappresentano i segnali prelevati dalla memoria quando questa è in modalità lettura, ma estesi a trenta bit.

#### Data Path

La SRAM presente sulla scheda "Altera DE2" ha dimensioni di 256K x 16. Per poter salvare i valori di RGB per ogni pixel dobbiamo ridurre tale segnale da trenta ad otto bit, salvando quindi in ogni *word* i dati relativi a due pixel.

Per semplificare l'algoritmo che calcola "Add" a partire da "X" ed "Y", "X" viene shiftato di una posizione a destra (ovvero diviso per due) e le sue uscite divengono i nove bit meno significativi dell'"Add", mentre il segnale "Y" i nove più significativi. L'LSB di "X" viene, invece, utilizzato dalla *control unit* per stabilire se il valore di "Data\_UART\_In" debba essere salvato nel byte superiore od inferiore della *word* nella memoria.

Siccome il segnale "Data\_MEM" è una uscita bidirezionale, essa deve essere gestita da un buffer tristate in cui indicheremo con "Data\_MEM\_Out" il dato in uscita dal "Memory Interface" e quindi quello che si vuole andare a scrivere in memoria, mentre con "Data\_MEM\_In" quello in ingresso, ovvero quello letto da memoria.

"Data\_MEM\_Out" viene ottenuto prelevando i bit 29, 28, 27, 19, 18, 17, 9 e 8 del segnale "Data\_UART\_In" ovvero i bit più significativi del rosso (R), verde (G) e blu

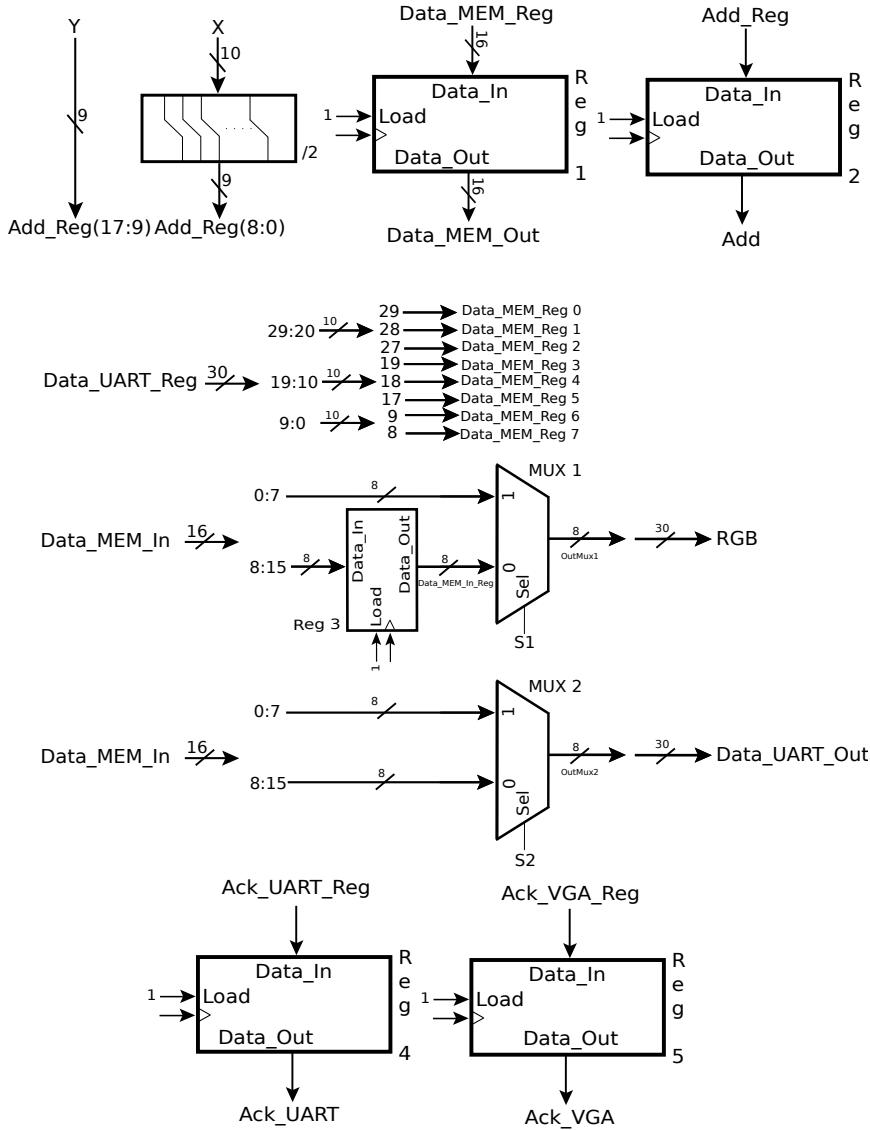


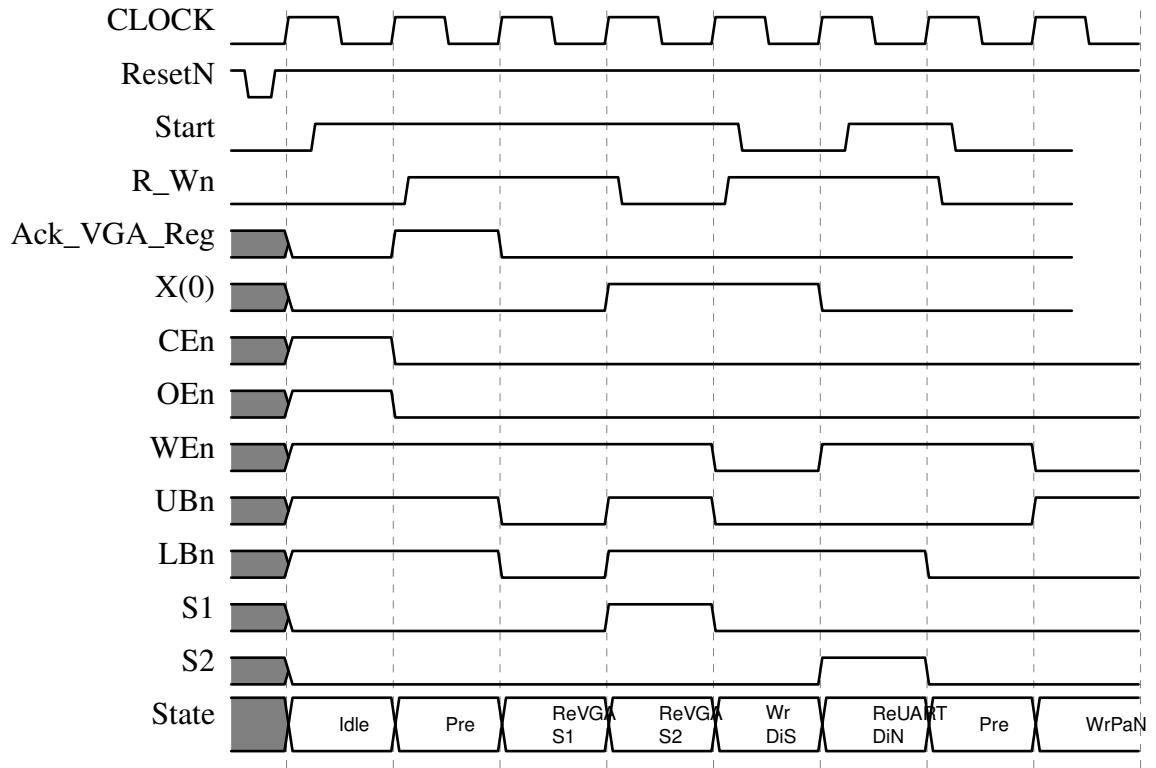
Figura 3.8: Memory Interface Data Path

(B).

"Data\_MEM\_In", invece, opportunamente riportato su trenta bit, viene portato in uscita sia come RGB sia come "Data\_UART\_Out", ma con modalità diverse. Infatti se è stata la VGA a richiedere la lettura, in uscita si deve avere il contenuto sia dell'*upper* byte che del *lower*, in due colpi di clock consecutivi. Per tale motivo, mentre il *lower* byte viene inviato subito al "MUX1" l'*upper* passa prima per un registro, in modo da essere ritardato di un ciclo di clock. Invece se la lettura è stata richiesta dall'UART si desidera leggere il valore di un solo pixel, pertanto i due byte di una *word* vengono inviati al "MUX2" il cui selettore viene indicato dalla *control unit* in base al valore del LSB di "X\_UART".

I due segnali di *acknowledge* prima di essere portati in uscita, vengono fatti passare per un registro in maniera tale da uscire contemporaneamente con i relativi valori di "RGB" e "Data\_UART\_Out".

## Timing Diagram



## FSM

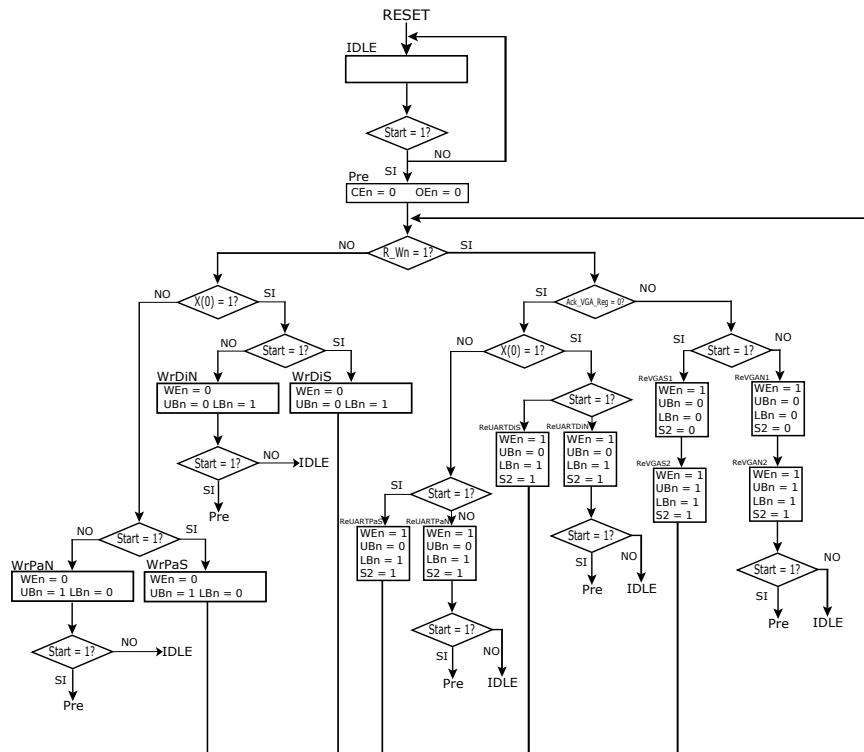


Figura 3.9: Memory Interface Finite State Machine

Lo "Start" che fa uscire la macchina a stati dallo stato di "Idle" è il medesimo dell'"Arbitro". Ma in questo caso la macchina finisce in uno stato di "Pre" dove vengono generate le uscite dell'"Arbitro" che faranno evolvere il "Memory Controller". Per il nome assegnato agli stati è stato utilizzato il seguente criterio

- Wr/Rd: Write/Read a seconda se la macchina vuole leggere o scrivere in memoria;
- UART/VGA: a seconda se la macchina sta servendo l'UART o la VGA;
- Di/Pa: Dispari/Pari a seconda se si vuole scrivere o leggere un byte contenuto nell'*lower* o *upper* byte. Si riferisce al valore di "X";
- N/S: No/Si a seconda se lo stato è succeduto al campionamento di una "Start" pari a '1' o pari a '0'.

Dopo lo stato "Pre" la macchina può evolvere o nel caso in cui si debba effettuare una scrittura o nel caso in cui si debba effettuare una lettura.

Nel secondo caso gli stati si dividono a seconda se tale richiesta è stata effettuata dalla UART o dal "VGA Controller". Mentre nel primo caso occorre ancora discernere il caso in cui si vuole leggere l'*upper* o il *lower* byte, nel secondo si susseguono due stati in quanto nel primo viene riportato in uscita l'*upper* byte e nel secondo il *lower*. Nel caso in cui si voglia eseguire una scrittura, occorre discernere, in maniera analoga, solo il caso in cui tale scrittura debba essere fatta sull'*upper* o sul *lower* byte. Infine, risulta evidente che tutti gli stati sono stati sdoppiati a seconda se nello stato precedente era stato campionato uno "Start" alto o basso. Ciò, perché, nel primo caso la macchina può lavorare in sequenza e quindi gestire subito dopo una nuova richiesta, nel secondo invece, non essendo stata pervenuta una nuova richiesta deve ritornare nello stato di "Pre".

### 3.3 Analisi dei tempi

Da quanto scritto qui sopra si evince che da quando viene perpetrata una richiesta a quando sono disponibili le relative uscite sono necessari tre cicli di clock. In particolare modo si ha:

- Ciclo di clock #1: Acquisizione della richiesta;
- Ciclo di clock #2: Campionamento delle coordinate e del "Data\_UART\_In" e arbitraggio dei dati;
- Ciclo di clock #3: Asserimento dell'*acknowledge* e relativi segnali da fornire alla memoria.

Ma essendo la macchina dotata di uno stadio di *pipe* essa è in grado di acquisire una richiesta ogni ciclo di clock, così come è in grado di fornire le relative uscite.

## 3.4 Simulazione Modelsim

Il corretto funzionamento del "Memory Controller" è stato verificato mediante simulazione ModelSim:

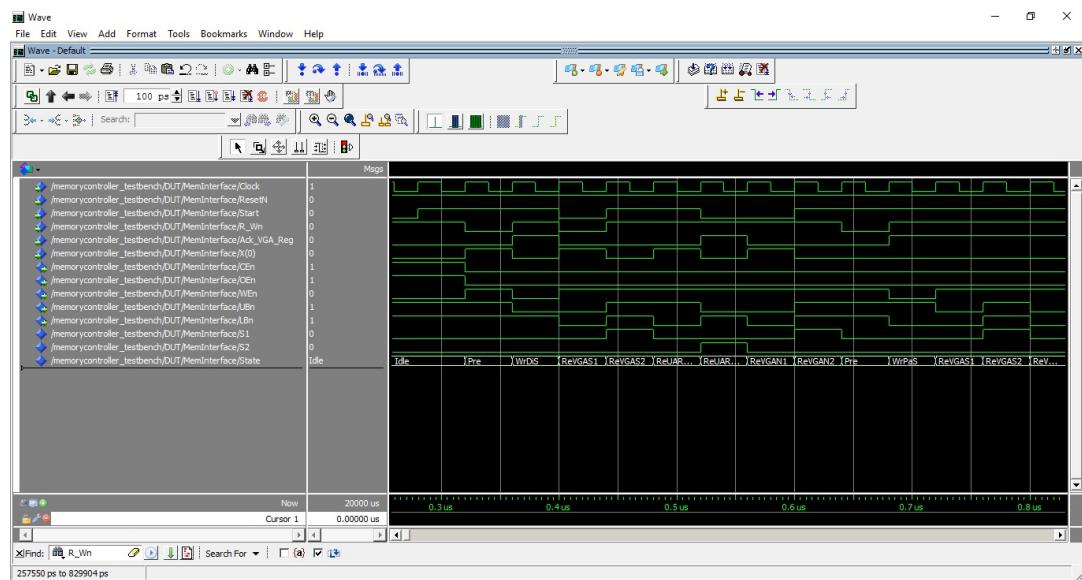


Figura 3.10: Simulazione con Modelsim di più letture e scritture in sequenza

## 3.5 Test in laboratorio

Per testare il "Memory Controller" sulla scheda DE2 è stata progettata una macchina a stati denominata "Test Machine", la quale in una prima fase carica la SRAM ed in un secondo momento mostra, attraverso dei LED, il valore contenuto nella locazione di memoria puntata da un indirizzo scelto dall'utente<sup>1</sup>. Tale macchina è costituita da un blocco composto da tre contatori. Il primo è un contatore su 8 bit la cui uscita viene poi "allargata" su 30 bit ed entra nel "Memory Controller" come "Data\_UART\_In". Il secondo è invece su 5 bit e viene indirizzato verso i cinque bit meno significativi dell'ingresso "X\_UART", mentre i *most significant* sono mantenuti a '0'. Infine, in modo analogo, l'ultimo contatore è di 3 bit e va a puntare i tre bit meno significativi di "Y\_UART". Dopo aver ricevuto il segnale di "start" tale macchina asserisce un segnale che viene riconosciuto come richiesta di lettura da parte del "Memory Controller" e fa partire i contatori in maniera tale che nella locazione di memoria puntata da X e Y viene scritto un valore dato dalla seguente formula:

$$\text{Data\_UART\_In} = X + 32Y$$

Caricato il programma su scheda sono stati effettuati i seguenti passi:

1. Deassерimento del segnale di "reset" ponendo in posizione '1' lo switch 0.
2. Invio del segnale di "start" ponendo in posizione '1' lo switch 1.
3. Attendere la fine dell'operazione di scrittura in memoria, segnalata dall'accensione del LED verde 0.
4. Deasserimento del segnale di "start" ed invio della richiesta di lettura da memoria ponendo in posizione '1' lo switch 2.
5. Grazie agli switch dal numero 3 al numero 10 si impostano i valori di X e Y con cui puntare la memoria.

Scelti i valori di X e Y ci aspettiamo che il valore mostrato dai LED sia pari a quello fornito dall'equazione prima citata. In particolar modo se il valore di X è pari al dato corretto sarà mostrato sul *lower byte*, in caso contrario sull'*upper*. Nell'immagine 3.11 si nota che il valore di Y è pari a "011", ovvero 3, mentre quello di X è stato impostato a "00010", ovvero 2. Partendo dalla sinistra il secondo LED acceso indica l'"Ack\_VGA" mentre degli altri sedici i primi otto rappresentano il *lower byte* e gli altri otto l'*upper*.

Il *lower byte* è risultato essere pari "01100010", ovvero 98, mentre l'*upper* è pari a "01100010", ovvero 99.

Avendo impostato un valore di X pari, ci attendevamo sul *lower byte* un valore pari a  $2 + 3 \cdot 32$ , ovvero 98. Il fatto che l'*upper byte* sia di un'unità maggiore dell'*lower* è una riprova del corretto funzionamento del "Memory Controller".

---

<sup>1</sup>Per rendere possibile ciò è stata effettuata una piccola modifica al "Memory Controller", in quanto quello originale poneva in uscita verso la VGA un solo byte per colpo di clock, mentre in questa situazione vogliamo leggere contemporaneamente l'intera word

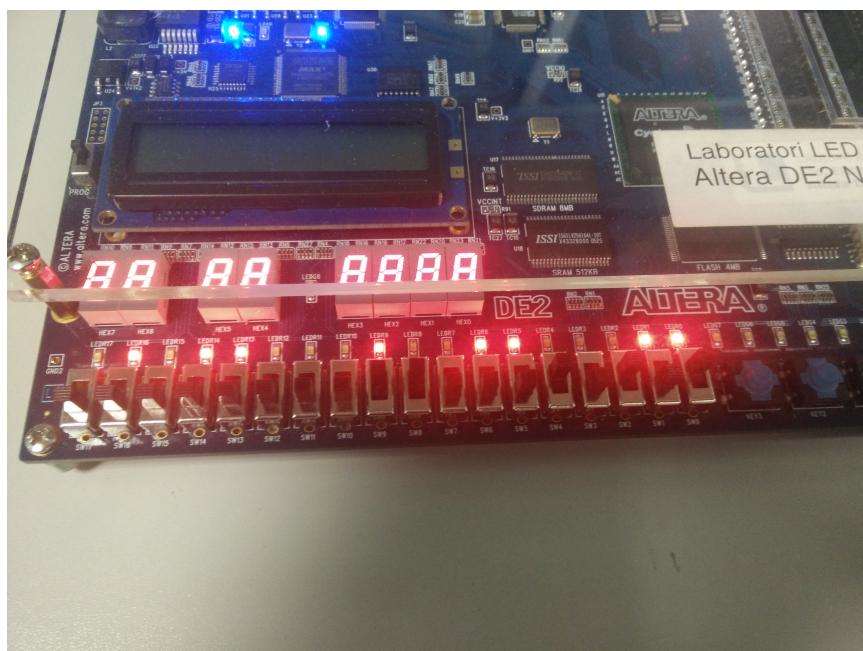


Figura 3.11: Risultato del test sul memory controller

# Capitolo 4

## Test Finale

Una volta testati singolarmente tutti i blocchetti si sono uniti in una macro entity comprendente "Memory Controller" e "VGA Controller" in modo tale da leggere dei dati inseriti opportunamente in memoria mediante un blocchetto apposito e visualizzarli su display.

Il risultato finale è quello presentato nella seguente foto:



Figura 4.1: Risultato finale

## 4.1 Test Bench

In quest'ultima sezione viene presentato il *test bench* della macchina che unisce "VGA Controller" e "Memory Controller".

Nella prima fase viene caricata la memoria in maniera del tutto uguale a quanto fatto per il "Memory Controller". In figura 4.2 è stato evidenziato che nella locazione di memoria puntata da "X" 189 e "Y" 0 sia stato memorizzato il dato "101000000 1110000000 010000000" (per una lettura più facile sono stati separati i tre segnali dei tre diversi colori).

In figura 4.3 viene, invece rappresentata la seconda fase dell'operazione. Il "VGA Controller" chiede al "Memory Controller" il dato memorizzato nell'indirizzo puntato dai segnali "X" 189 e "Y" 0, e quest'ultimo, secondo il *timing* analizzato nella sezione 4.3, lo porta in uscita sul segnale "RGB".

In figura 4.4, infine, con una latenza pari a quattro cicli di clock da quando il "VGA Controller" ha richiesto questo specifico dato, esso viene inviato al DAC attraverso i segnali "VGA\_r" ("10100000", "VGA\_g" ("1110000000") e "VGA\_b" ("0100000000").

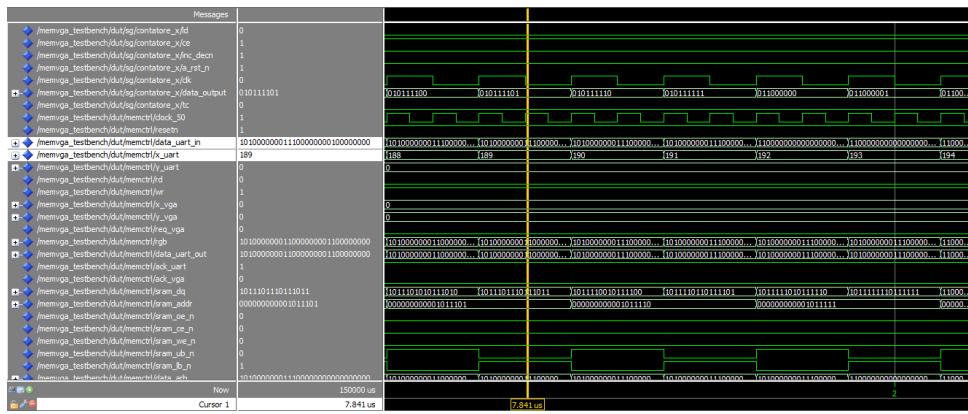


Figura 4.2: Scrittura dati in memoria

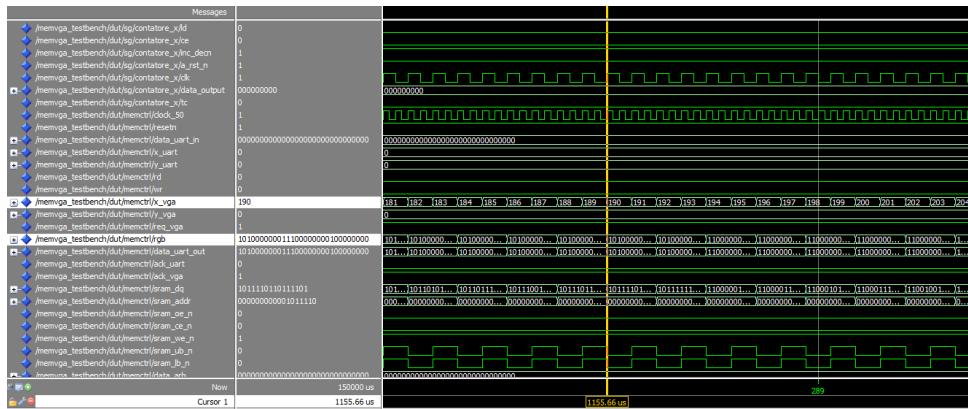


Figura 4.3: Lettura dati da memoria

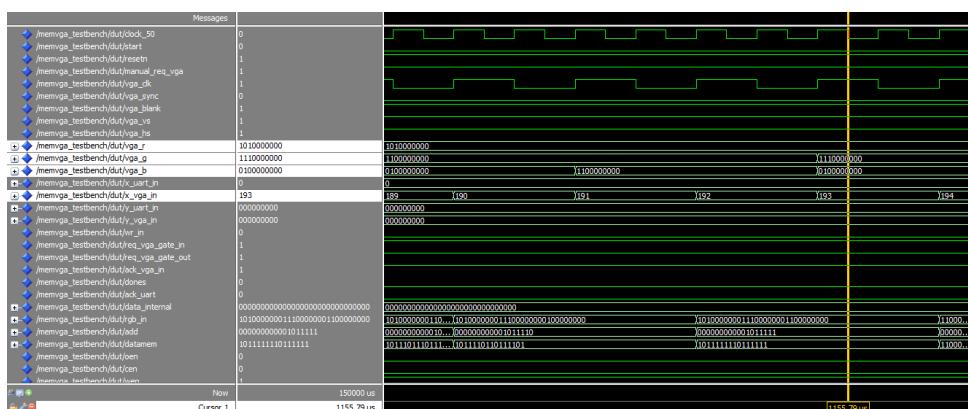


Figura 4.4: Uscita dati verso il DAC