Laboratorio 2:
Realizzazione di un sistema integrato con accelerometro, porta seriale e controller VGA

Sistemi Digitali Integrati

Professor Massimo Ruo Roch

Gruppo 2:

Calabrese Deborah	241408
Lanieri Valerio	234526
Mazzia Claudia	235208

18 novembre 2018

Sommario

L'obiettivo di questo laboratorio é realizzare un sistema composito per il controllo della posizione di un'immagine su di uno schermo utilizzando una scheda Nucleo F401RE, un modulo di sensori mems IKS01A1 ed una scheda FPGA Cyclone II della Altera.

Indice

1 Introduzione **2**

2 Microcontrollore **2**
2.1 Accelerometro . . . 3
2.1.1 Matrice di rotazione e quaternioni . . . 4

3 Scheda FPGA **5**
3.1 Modulo seriale . . . 5
3.1.1 Datapath . . . 6
3.1.2 Unità di controllo . . . 10
3.1.3 Timing . . . 13
3.1.4 Analisi del timing e validazione del componente . . . 16
3.2 Modulo VGA . . . 17
3.2.1 Datapath . . . 18
3.2.2 Unità di controllo e timing . . . 20
3.2.3 Analisi del timing e validazione del componente . . . 25

4 Test **27**
4.1 Test del microcontrollore . . . 27
4.2 Test del componente di gestione della seriale . . . 28
4.3 Test del componente di controllo della VGA . . . 29
4.4 Test complessivo . . . 31

5 Conclusione **31**

6 Appendice: testbench **32**

1 Introduzione

Per svolgere questo laboratorio dobbiamo sfruttare le componenti a disposizione per ricavare uno spostamento attraverso l'accelerometro posto sul modulo di sensori IKS01A1, quindi inviarlo via seriale alla scheda FPGA, elaborare i dati ricevuti e infine aggiornare la posizione sullo schermo della figura stampata. Tale figura, nel nostro caso, é un quadrato rosso, inizialmente di dimensioni 10 pixel x 10 pixel e posto orizzontalmente fra il pixel 315 e il pixel 325, verticalmente fra il pixel 235 e il pixel 245.

Abbiamo dunque programmato la scheda Nucleo per impostare correttamente il modulo accelerometro, acquisire letture di accelerazione sui tre assi ed integrarle due volte, in modo da ottenere i valori di spostamento, per poi inviarli tramite linea seriale alla scheda FPGA. Questa, a sua volta, é stata programmata per accettare i dati di posizione x, y e z attraverso una comunicazione seriale e controllare uno schermo dotato di ingresso VGA. Tale controllo sostanzialmente aggiorna la posizione del quadrato rosso, rispettivamente alzandolo o abbassandolo lungo l'asse z, spostandolo a destra o sinistra lungo l'asse x e ingrandendolo o rimpicciolendolo, ossia rappresentando uno spostamento lungo l'asse y.

2 Microcontrollore

Il microcontrollore a disposizione è una scheda NUCLEO-F401RE della STmicroelectronics a cui e stato aggiunto un modulo di sensori MEMs IKS01A1. Ai fini del laboratorio l'unico sensore utilizzato è l'LSM6DS0, che include un accelerometro e un giroscopio triassiali. Tramite l'utilizzo della piattaforma online Mbed

abbiamo creato un programma che raccoglie i dati dai suddetti accelerometro e giroscopio, li elabora trasformandoli in coordinate spaziali e li invia alla FPGA. Per settare e leggere i registri del nostro sensore abbiamo utilizzato il protocollo seriale I2C, settando il relativo periferico del microcontrollore mediante delle funzioni incluse nelle librerie fornite dalla piattaforma Mbed. In particolare abbiamo sfruttato:

- **Classe I2C**: grazie alla funzione I2C presente in questa classe abbiamo impostato il microcontrollore come master. Di conseguenza, il nostro sensore sarà il suo slave, utilizzando il pin PB_9 come SDA e il pin PB_8 come SCL. Utilizzando infine la funzione "Write" effettuiamo lo scambio dati tra microcontrollore e sensore.

Per quanto riguarda invece lo scambio dati tra microcontrollore ed FPGA è stata utilizzata l'interfaccia seriale RS232. Anche in questo caso sono state utilizzate le funzioni incluse nelle librerie rese disponibili da Mbed:

- **Classe Serial**: grazie alla funzione Serial presente in questa classe abbiamo settato l'apposito periferico del microcontrollore per effettuare la comunicazione seriale utilizzando il protocollo RS232. In particolare abbiamo settato il pin di trasmissione (PB_6), il pin di ricezione (PA_10) ed il baud rate desiderato, ovvero 9600. Mediante la funzione printf abbiamo poi effettuato la trasmissione dei dati verso l'FPGA.

Il nostro sensore memorizza ognuna delle tre accelerazioni lineari e delle tre velocità angolari su due registri da un byte ciascuno. Abbiamo dunque creato una funzione, che abbiamo chiamato mult_read, in cui effettuiamo la lettura dei due registri di output. Da notare che abbiamo acquisito i due byte come variabili di tipo char per poi fare il casting da char a int16_t in modo tale da fornire in uscita una variabile a 16 bit in CA2. Tale operazione viene ovviamente eseguita per tutte e tre le accelerazioni e tutte e tre le velocità.

2.1 Accelerometro

Tramite la funzione da noi creata e chiamata setup_accelerometro definiamo il comportamento dell'accelerometro e del giroscopio settandone gli appositi registri:

- **CTRL_REG1_G**: settiamo tale registro di controllo del giroscopio a 0x68 in modo tale che l' accelerometro e il giroscopio campionino ogni 119 Hz e che il giroscopio abbia un fondo scala pari a 500 dps. E' necessario fare ciò poiché da datasheet ci viene detto che quando si vogliono utilizzare contemporaneamente accelerometro e giroscopio l'output data rate (ODR) dell'accelerometro deve essere settato, nell'apposito registro (CRTL_REG6_XL), come se fosse in power down.

- **CTRL_REG7_XL**: settiamo questo registro di controllo dell'accelerometro a 0x85 in modo tale da attivare la modalità alta risoluzione per l'accelerometro e il relativo filtro passa basso, impostando una frequenza di taglio pari a 2.38 Hz. Da notare che tale valore é stato scelto in modo da rispettare la relazione $f_{cutoff} = \frac{ODR}{50}$. Così facendo le letture dell'accelerometro sono più precise ed inoltre si eliminano quasi completamente i contributi dell'accelerazione gravitazionale sui tre assi, evitando il problema di deriva del quadrato lungo l'asse z.

Da notare che i registri che hanno già di default i valori che noi desideriamo non sono presenti nella funzione setup_accelerometro. Riportiamo comunque di seguito quali sono tali registri e come incidono sul comportamento del nostro sensore:

- **CTRL_REG6_XL**: setta l'accelerometro in power down per poterlo utilizzare contemporaneamente al giroscopio e setta il fondo scala dell'accelerometro a 2g.

- **CTRL_REG_4**: abilita le uscite del giroscopio.

- **CTRL_REG5_XL**: abilita le uscite dell'accelerometro.

- **CTRL_REG_8**: attiva la modalità big endian per l'immagazzinamento dei dati in memoria.
- **FIFO_CTRL**: abilita la modalità bypass per la memoria FIFO. Così facendo leggiamo direttamente i valori dai registri di uscita del sensore senza che questi vengano memorizzati in una coda.

Ciò che a noi interessa è in realtà la posizione del nostro sensore. Abbiamo quindi dovuto effettuare l'integrazione dei valori forniti dall'accelerometro e lo abbiamo fatto utilizzando il metodo dei Trapezi nella funzione da noi chiamata trapezi_pos. Questo metodo introduce un'ulteriore fonte di errore nel calcolo della posizione poiché fornisce un valore approssimato dell'integrale da calcolare.

Come detto precedentemente, grazie all'utilizzo del filtro passa basso relativo all'accelerometro, e grazie alla modalità alta risoluzione, eliminiamo il contributo dovuto all'accelerazione gravitazionale sulle nostre letture.

Prima di far ciò abbiamo però provato a implementare un algoritmo che avesse lo stesso effetto del suddetto filtro ma sfruttando i quaternioni e le matrici di rotazione. Purtroppo, i risultati ottenuti con questi algoritmi non erano quelli sperati, nel senso che il quadrato derivava comunque lungo l'asse z. Pertanto, abbiamo deciso di utilizzare direttamente il filtro già presente nel nostro sensore. A scopo informativo, riportiamo di seguito i passaggi effettuati per ottenere i quaternioni e la relativa matrice, oltre che quelli effettuati per ricavare la matrice di rotazione a partire dagli angoli di Eulero forniti dal giroscopio.

2.1.1 Matrice di rotazione e quaternioni

Gli angoli che ci vengono forniti dalle letture del giroscopio sono i cosiddetti angoli di Eulero. Tali angoli descrivono l'orientamento della terna degli Assi Corpo, ovvero la terna solidale al sensore rispetto alla terna degli Assi Verticali Locali, cioè la terna che segue il sensore nel suo movimento rispetto al punto iniziale di riposo mantenendo invariato il suo orientamento. Ora dobbiamo riportare gli Assi Verticali Locali sugli Assi Corpo. Per fare ciò, effettuiamo tre rotazioni successive, indicate appunto dagli angoli di Eulero, intorno agli assi di una terna, che a sua volta é allineata con gli assi Verticali Locali. Per far ciò è importante rispettare l'ordine delle rotazioni, che dovrà sempre essere prima attorno all'asse z (ψ), poi attorno all'asse y (θ) ed infine attorno all'asse x (ϕ). Si ottiene così la seguente matrice di rotazione:

$$\begin{pmatrix} \cos\psi\cos\theta & \cos\theta\cos\psi & -\sin\theta \\ \cos\psi\sin\phi\sin\theta - \cos\phi\sin\psi & \cos\phi\cos\psi + \phi\sin\psi\sin\theta & \cos\theta\sin\phi \\ \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta & \cos\phi\sin\psi\sin\theta - \cos\psi\sin\phi & \cos\phi\cos\theta \end{pmatrix} \tag{1}$$

Per ogni orientamento degli Assi Corpo la precedente sequenza di rotazioni ammette una sola combinazione degli angoli di Eulero, tranne che nel caso $\theta = \pm 90°$. In tale caso infatti si hanno i punti di singolarità che rendono possibili infinite combinazioni degli angoli di rollio e di imbardata (rispettivamente, rotazione attorno all'asse x e z). Per tale motivo non è possibile utilizzare una semplice matrice di rotazione ma occorre ricorrere all'utilizzo dei quaternioni.

Non riporteremo la trattazione teorica di tutte le proprietà dei quaternioni ma ci limiteremo ad illustrare le operazioni eseguite. Le relazioni che forniscono le componenti del quaternione a partire dagli angoli di Eulero sono le seguenti:

$$q0 = \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \tag{2}$$

$$q1 = \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\psi}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\psi}{2} \tag{3}$$

$$q2 = \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} \tag{4}$$

$$q3 = \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\psi}{2} - \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\psi}{2} \tag{5}$$

Quando il quaternione q:

$$q = q0 + q1i + q2j + q3k \tag{6}$$

descrive l'assetto del sensore, rappresentando la rotazione che allinea gli Assi Verticali Locali agli Assi Corpo, la matrice che segue costituisce la matrice di trasformazione in termini del quaternione :

$$\begin{pmatrix} q0^2 + q1^2 - q2^2 - q3^2 & 2(q1q2 + q0q3) & 2(q1q3 - q0q2) \\ 2(q1q2 - q0q3) & q0^2 - q1^2 + q2^2 - q3^2 & 2(q2q3 + q0q1) \\ 2(q1q3 + q0q2) & 2(q2q3 - q0q1) & q0^2 - q1^2 - q2^2 + q3^2 \end{pmatrix} \tag{7}$$

Questa è la matrice da noi utilizzata e moltiplicata per il vettore delle accelerazioni fornite dall'accelerometro. Al vettore così ottenuto abbiamo poi sottratto il vettore formato dalle componenti dell'accelerazione gravitazionale rispetto ai tre assi, ovvero 0 $\frac{m}{s^2}$ rispetto x e y e -9.8 $\frac{m}{s^2}$ rispetto all'asse z.

Come già detto fare ciò non risolve i problemi di deriva del quadrato lungo l'asse z poiché bisognerebbe implementare anche il filtro di Kalman. Esso corregge gli errori di assetto e di bias del giroscopio. Inoltre, anche ulteriori accorgimenti sarebbero necessari, come per esempio l'utilizzo dei magnetometri. Resici conto dell'onerosità di tale progetto, e considerato il poco tempo a nostra disposizione, abbiamo accantonato l'idea di progettare tutto ciò e abbiamo invece utilizzato il filtro dell'accelerometro già presente nel nostro sensore. Dunque, il nostro progetto di fatto non sfrutta né la matrice di rotazione né i quaternioni.

3 Scheda FPGA

La scheda FPGA a disposizione (modello Cyclone II della Altera) é stata programmata utilizzando l'IDE dedicato Quartus II. Il progetto in VHDL é stato opportunamente testato su Modelsim e rifinito fino ad ottenere risultati esattamente identici alle nostre aspettative. Infine, dopo aver caricato il progetto, abbiamo usato l'oscilloscopio a disposizione per sincerarci che i segnali inviati fossero effettivamente uguali a quelli simulati, e dunque conformi al nostro progetto. Nelle prossime sezioni descriveremo come abbiamo progettato e realizzato i vari componenti.

3.1 Modulo seriale

Per effettuare il trasferimento dati da microcontrollore a FPGA è stata utilizzata l'interfaccia seriale RS-232. E' dunque stato necessario realizzare il blocco di ricezione rispettandone il protocollo. Tale protocollo prevede che venga trasferito un carattere alla volta. Ogni carattere è formato da:

- 1 bit di start;
- 7-8 bit di dato;
- 0-1 bit di parità;
- 1-1.5-2 bit di stop;

Noi abbiamo scelto di utilizzare 8 bit di dato, 0 bit di parità e 1 bit di stop per un totale di 10 bit.

Il protocollo prevede che la linea stia a livello logico 1, cioè a livello basso di tensione, quando non vi è trasferimento dati e prevede che l'inizio di una trasmissione sia caratterizzato da uno 0 logico (definito start bit), cioè una tensione alta. La fine di una trasmissione, invece, é caratterizzata da un 1 logico (stop bit), ossia la linea deve essere riportata al livello basso di tensione.

Ulteriore precisazione, per quanto riguarda il pacchetto di dati inviati, è che viene sempre mandato prima l'LSB e solo alla fine l'MSB. Bisogna quindi far attenzione a questo dettaglio quando si ricostruisce il dato ricevuto. Il carattere può, inoltre, essere trasferito a diversi baudrate, che nel caso della RS-232 coincide con il bps. Pertanto bisogna sincronizzare il blocco trasmettitore, ovvero il microcontrollore, con il blocco ricevitore, ovvero l' FPGA, cioè bisogna farli lavorare allo stesso baudrate affinché la comunicazione avvenga con successo.

Da specifiche di progetto ci è stato inoltre richiesto di effettuare un controllo sul frame inviato in modo tale da aggiornare i valori della VGA solo se tutti e tre i nostri dati sono arrivati correttamente. Il controllo viene fatto inviando dal microcontrollore prima due byte di start (0xff), poi i dati ed infine due byte di stop (0x00). Solo una volta verificato che i byte di start e stop sono stati ricevuti correttamente aggiorniamo i registri, la cui uscita viene inviata al modulo di controllo VGA per aggiornare la posizione a schermo del quadrato.

3.1.1 Datapath

In questa sezione riportiamo il datapath e descriviamo il funzionamento degli elementi che lo compongono. Per rendere il datapath più facilmente leggibile lo suddividiamo in tre blocchi che spiegheremo singolarmente.

Blocco generatore dei clock

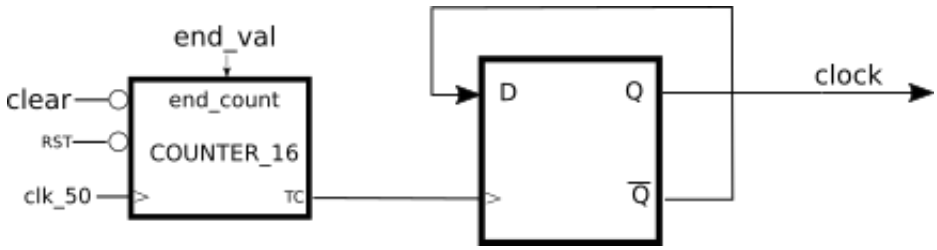

Figura 1: Componente che genera uart_clock e mem_clock

Abbiamo utilizzato due contatori su 16 bit e due D flip flop per generare i clock di cui abbiamo necessitato per sincronizzare il ricevitore con il trasmettitore e per sincronizzare lo scambio dati fra lo shift register, che ricostruisce il carattere, e il buffer in cui memorizziamo i dati. La scelta di un contatore su 16 bit è dovuta al fatto che in questo modo saremmo stati liberi di utilizzare qualunque baudrate semplicemente variando il suo ingresso (end_val). Tuttavia, la nostra scelta è ricaduta necessariamente su un baudrate pari a 9600 per far fronte alle specifiche di progetto.

I blocchi generatori di clock sono:

- **UART CLOCK GEN**: genera il clock della seriale. Esso utilizza l'uscita Terminal Counter del contatore come clock del D flip flop. In tal modo all'uscita del flip flop (Q) avremo il nostro uart_clock. Poichè vogliamo che l'uart_clock sia un'onda quadra con duty cycle del 50%, imponiamo che il contatore abbia TC_{freq} = baudrate*2. Poiché $TC_{freq} = \frac{f_{proc}}{end_val}$+1, otteniamo che il valore da assegnare ad end_val è pari a $\frac{f_{proc}}{TCfreq}$-1. Dunque nel nostro caso end_val = 2603.

- **MEM CLOCK GEN**: alla fine di ogni trasmissione asseriamo il clear dello shift register. Per non perdere i dati contenuti al suo interno utilizziamo un clock a frequenza doppia rispetto all'Uart_clock per sincronizzare lo scambio dati tra shift register e registri del blocco buffer frame, presente nello schematico successivo. Abbiamo chiamato tale clock mem_clock. Per generarlo assegniamo ad end_val un valore pari a 1301, in accordo con le formule sopra citate, con l'unica differenza che in questo caso il baud rate va moltiplicato per 4.

Blocco principale

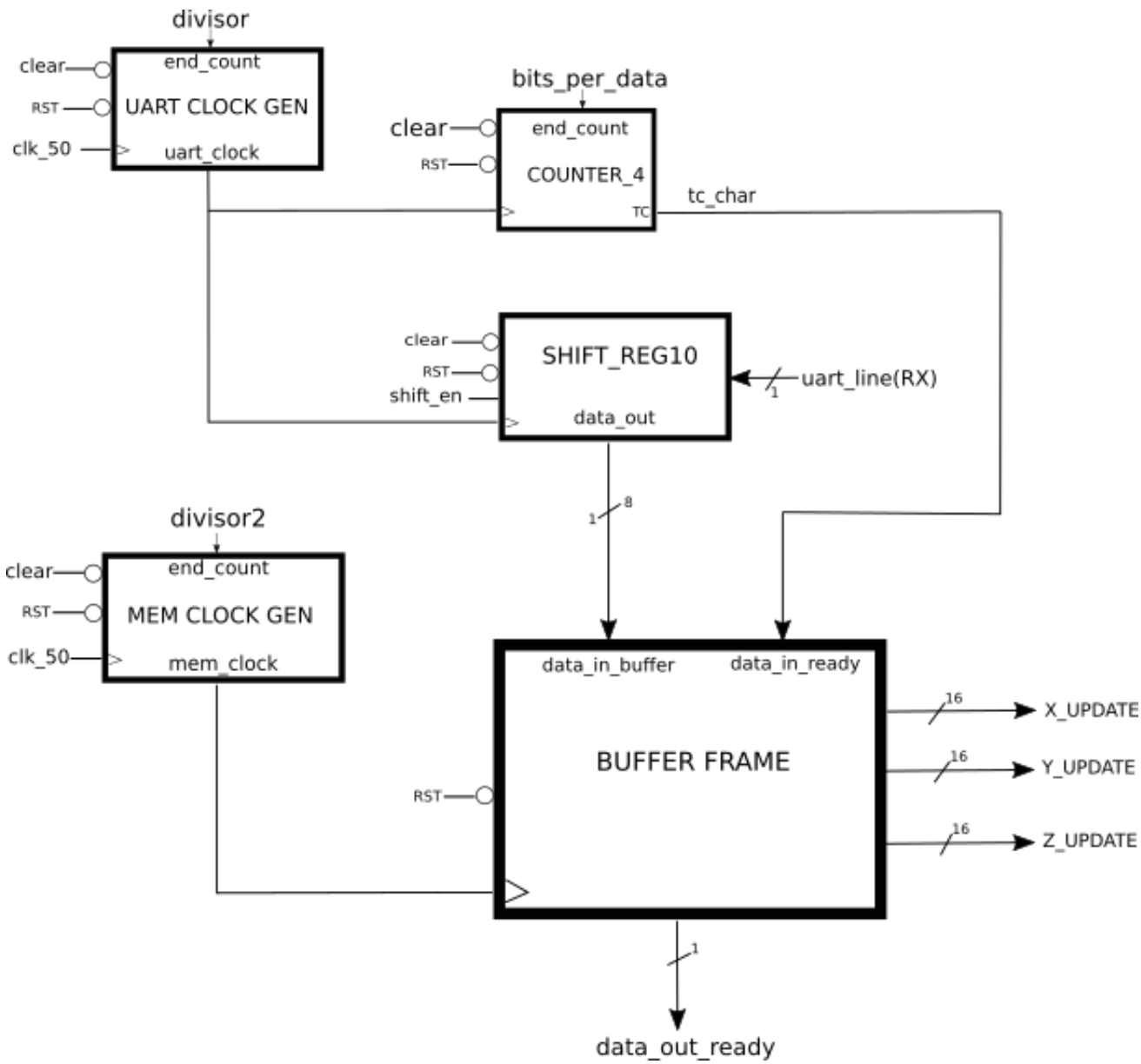

Figura 2: Blocco ricevitore della seriale

- **UART CLOCK GEN e MEM CLOCK GEN**: il primo blocco genera il clock del COUNTER4, dello shift register e della UART FSM, mentre il secondo blocco genera il clock del buffer frame e della

relativa unità di controllo.

- **COUNTER_4**: questo contatore conta fino a nove, ossia il valore di bits_per_data, in modo tale da sapere quando sono arrivati tutti i bit del carattere. Una volta finito di contare, questo blocco invia il segnale di Terminal Counter (tc_char) all'unità di controllo denominata UART FSM.
- **shift_reg10**: è un registro SIPO (Serial In Parallel Out) su 10 bit, il cui ingresso è la linea seriale. Esso effettua uno shift a destra in modo tale da avere in uscita il dato ricostruito correttamente, ovvero con l'MSB nella posizione di peso 2^8 e LSB nella posizione di peso 2^1.
- **buffer frame**: è il blocco in cui effettuiamo il controllo sui byte di start e stop, memorizziamo i sei byte di dato e mandiamo in uscita i tre dati su 16 bit. I segnali di dato sono denominati x_update, y_update e z_update. La trattazione approfondita di questo blocco è riportata di seguito.

Blocco buffer frame

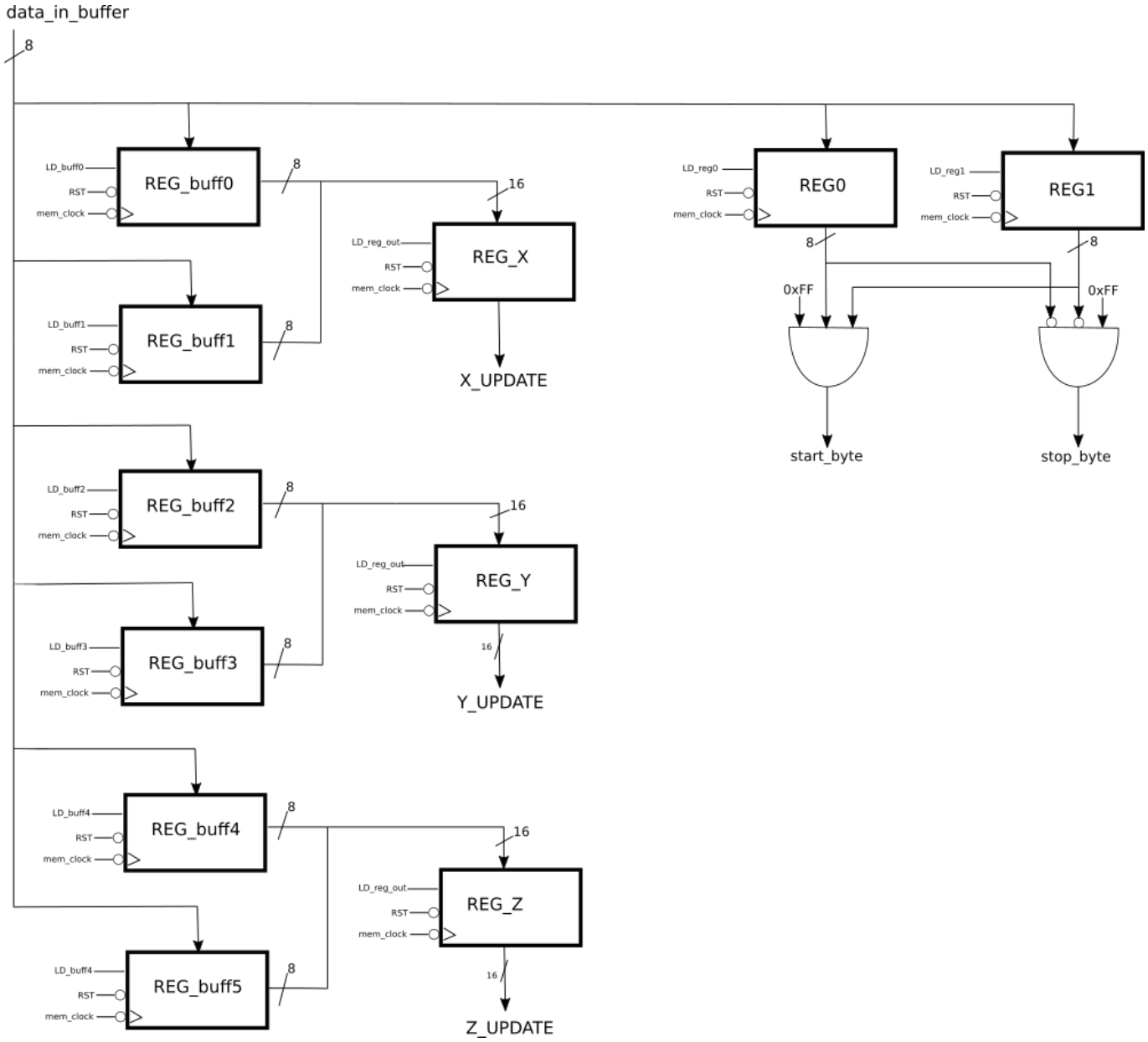

Figura 3: BUFFER FRAME: blocco in cui si memorizzano i dati in ingresso dalla seriale

- **REG0 e REG1**: in questi registri memorizziamo i primi e gli ultimi due byte che inviamo dal microcontrollore in modo da poter fare il controllo sul frame. Se il controllo risulta essere positivo allora la ricezione è avvenuta correttamente e i dati vengono inviati al blocco VGA. Tale controllo é svolto tramite le due porte AND.
- **REG_buffN**: utilizziamo tali registri per memorizzare gli 8 bit di dato in arrivo dallo shift register.
- **REG_X, REG_Y, REG_Z**: sono i registri su 16 bit in cui memorizziamo i dati che inviamo alla VGA.

3.1.2 Unità di controllo

Poiché il blocco ricevitore della seriale e il blocco buffer frame utilizzano clock diversi per temporizzare le loro operazioni, abbiamo utilizzato due control unit. Le ASM chart vanno interpretate nel seguente modo: ogni rettangolo contiene i segnali che vengono impostati a '1'. Se il segnale, durante un dato stato, é basso, non appare nel riquadro facente riferimento allo stato.

Unità di controllo del blocco ricevitore
L'unità di controllo relativa al blocco ricevitore della seriale utilizza il clock del processore per temporizzare le operazioni, come si può vedere dal rispettivo timing (riportato nell'apposita sezione).

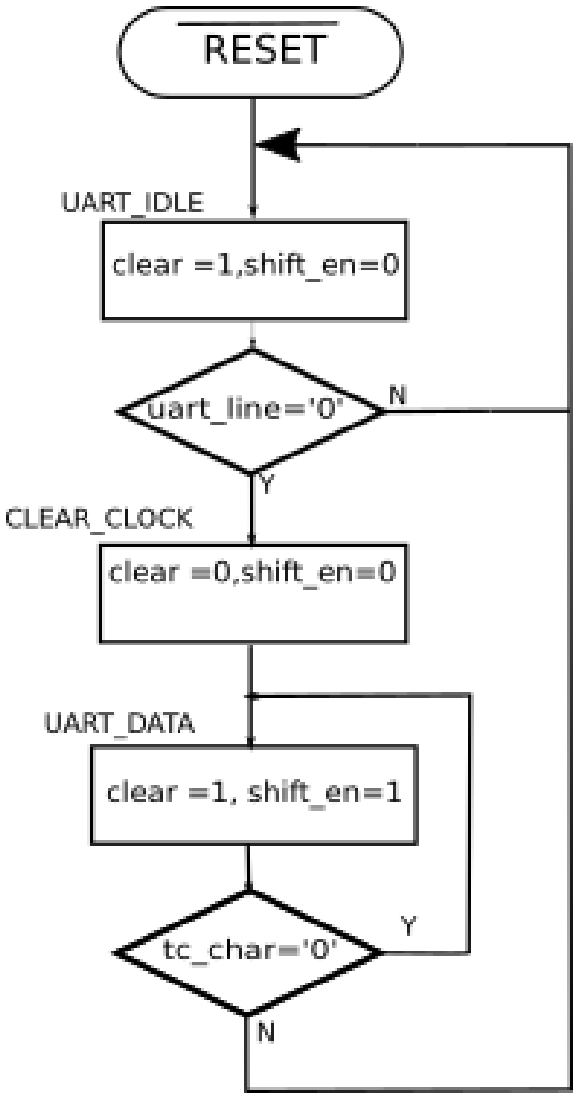

Figura 4: ASM chart dell'unità di controllo del blocco ricevitore della seriale

- **UART_IDLE** : stato in cui ci limitiamo ad aspettare l'inizio di una nuova trasmissione.
- **CLEAR_CLOCK**: stato in cui resettiamo tutti i contatori in modo tale da sincronizzare il clock del blocco ricevitore con il trasmettitore, ovvero il microcontrollore.

- **UART_DATA**: stato in cui abilitiamo lo shift register a shiftare i dati che sta ricevendo in ingresso dalla linea seriale. Restiamo in tale stato finchè non è arrivato un intero carattere per poi tornare nello stato di idle. Per fare questo controllo ci basta attendere che tc_char sia pari a 1.

Come si può notare da tale unità di controllo, e più ancora dal relativo timing, visto che i caratteri vengono inviati di seguito, l'uscita utile dello shift register, ovvero quella in cui il carattere è stato ricostruito con i bit nella posizione di peso corretto, resta memorizzata solo per mezzo colpo di clock dell'uart_clock. Tale uscita viene poi azzerata durante lo stato successivo (CLEAR_CLOCK).

Unità di controllo del BUFFFER FRAME

L'unità di controllo relativa al buffer frame utilizza il clock da noi chiamato mem_clock per temporizzare le operazioni e, in particolare, è attiva sul fronte di salita mentre, come si può notare dal relativo timing, i componenti del buffer frame rispondono al fronte di discesa del mem_clock. Tale scelta è stata fatta poiché se anche i componenti rispondessero al fronte di salita del mem_clock perderemmo il valore in uscita dallo shift register in quanto, come precedentemente spiegato, tale valore si presenta in uscita solo per mezzo colpo dello uart_clock.

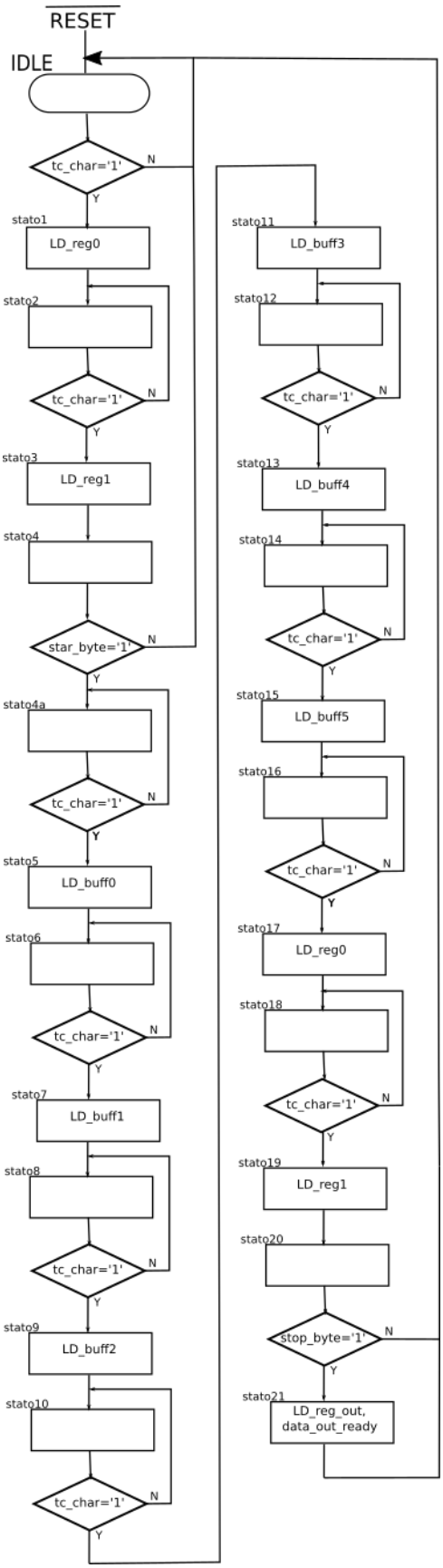

Figura 5: ASM chart dell'unità di controllo del frame buffer

- **IDLE**: in questo stato aspettiamo che sia arrivato un intero carattere.
- **stato1 e stato2**: nello stato1 abilitiamo il load del REG0 mentre nello stato2 lo disabilitiamo e aspettiamo che sia arrivato un altro carattere. Così facendo salviamo il primo byte di start in arrivo dal microcontrollore e attendiamo l'arrivo del secondo.
- **stato3 e stato4**: nello stato3 abilitiamo il load del REG1 mentre nello stato4 lo disabilitiamo. Così facendo salviamo il secondo byte di start in arrivo dal microcontrollore e controlliamo che sia il primo che il secondo byte siano arrivati correttamente. Se il controllo risulta essere positivo allora precediamo attendendo l'arrivo di un nuovo carattere, ovvero andiamo nello stato4a. In caso contrario, torniamo nello stato di idle.
- **Da stato5 a stato16**: in questi stati non facciamo altro che memorizzare, nei relativi registri, i dati effettivamente utili in arrivo dal microcontrollore.
- **Da stato17 a stato20**: ripetiamo le operazioni fatte negli stati da 1 a 4 con la differenza che in questo caso effettuiamo il controllo sui byte di stop. Se tale controllo è positivo andiamo nello stato 21, altrimenti torniamo nello stato di idle.
- **stato21**: in questo stato effettuiamo il caricamento dei registri a 16 bit REG_x, REG_Y e REG_Z le cui uscite vengono mandate al blocco VGA, in modo da aggiornare la posizione del quadrato sullo schermo. In questo stato, inoltre, asseriamo il segnale data_out_ready. Tale segnale viene utilizzato come segnale di load dei registri di aggiornamento posizione del blocco VGA.

Per quanto riguarda il segnale data_out_ready servirebbe un'ulteriore unità di controllo, poiché tale segnale deve essere asserito per un unico colpo di clock del processore a 50 MHz, e non per un colpo di mem_clock. Per evitare di complicare ulteriormente il modulo seriale abbiamo invece preferito utilizzare un process per svolgere tale operazione.

3.1.3 Timing

Riportiamo di seguito i timing separati delle due unità di controllo per renderne più veloce e comprensibile la lettura. Per facilitarne la consultazione ricordiamo che:

- mem_clock ha una frequenza doppia rispetto a uart_clock;
- il segnale di controllo relativo allo shift register (shift_enable) risponde all'uart_clock ed é pertanto riportato solamente nel timing del blocco ricevitore, nonostante i suoi effetti si vedano nel timing dell'unità di controllo del buffer frame;
- il segnale data_out_ready rimane alto per un solo colpo di clock a 50 MHz, pertanto nel timing dell'unità di controllo del buffer frame é rappresentato come un impulso molto piccolo, dato che il mem_clock (relativo a questo timing) ha una frequenza molto minore;
- il segnale LD_reg_out diventa don't care dopo essere stato asserito poiché lo campioniamo solo quando data_out_ready é alto;
- nel timing dell'unità di controllo del blocco ricevitore potrebbe sembrare che il segnale cnt_16 a volte rimanga sempre a 0 prima di passare a 2603. Ovviamente non é così: esso viene incrementato di 1 ad ogni colpo del clock a 50 MHz, dunque prima di passare a 2603 esso vale 2602, anche se nel timing tale valore non viene riportato;

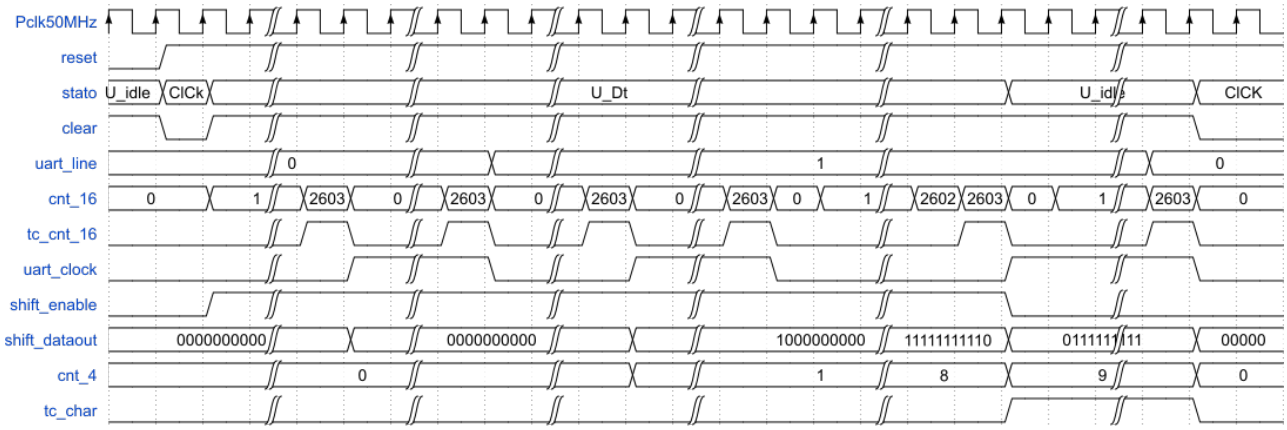

Figura 6: Timing dell'unità di controllo del blocco ricevitore

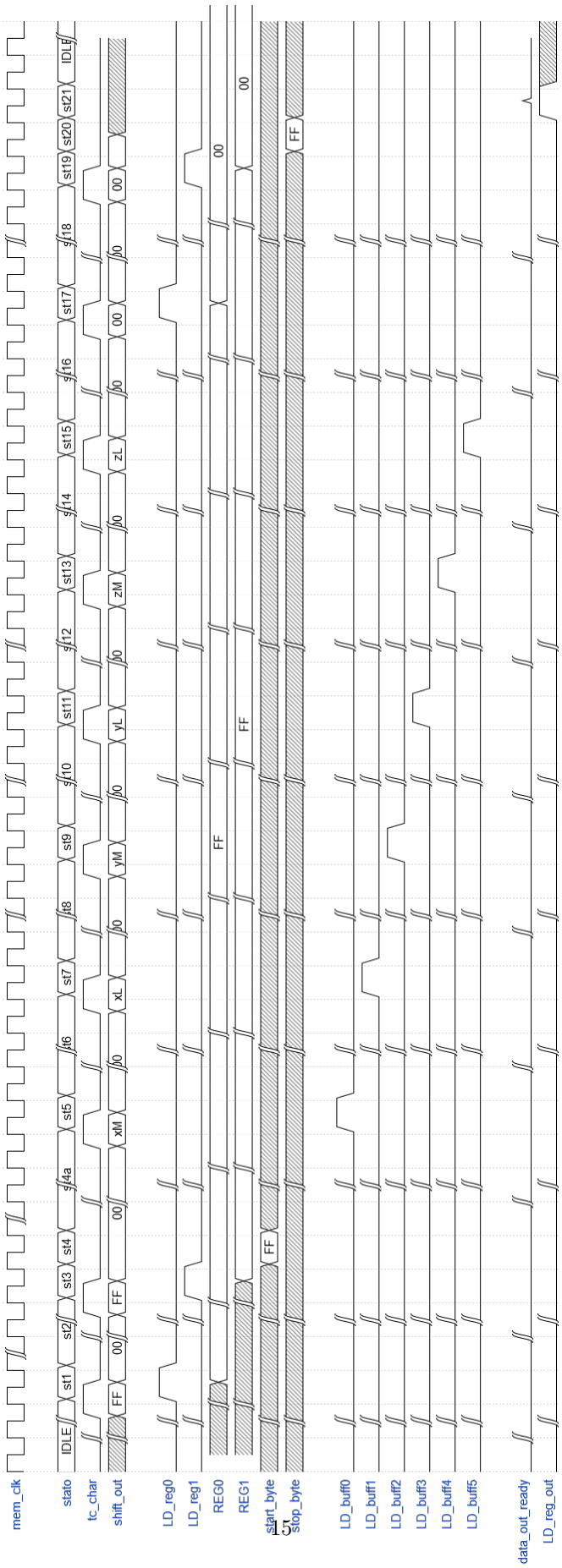

Figura 7: Timing dell'unità di controllo del buffer frame

3.1.4 Analisi del timing e validazione del componente

Dato il grande numero di colpi di clock necessario per esaminare il comportamento del componente seriale nella sua interezza, riportiamo solo alcune sezioni della simulazione del blocco ricevitore.

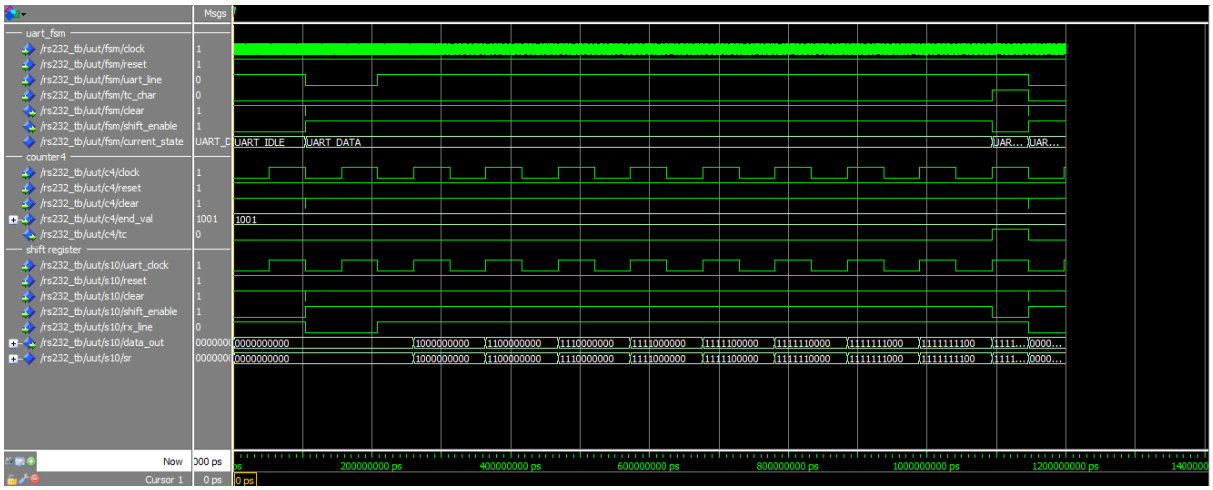

Figura 8: Inizio della trasmissione seriale. Il segnale data_out dello shift register varia sul fronte di salita dell'uart_clock in base al bit ricevuto di volta in volta dalla linea seriale

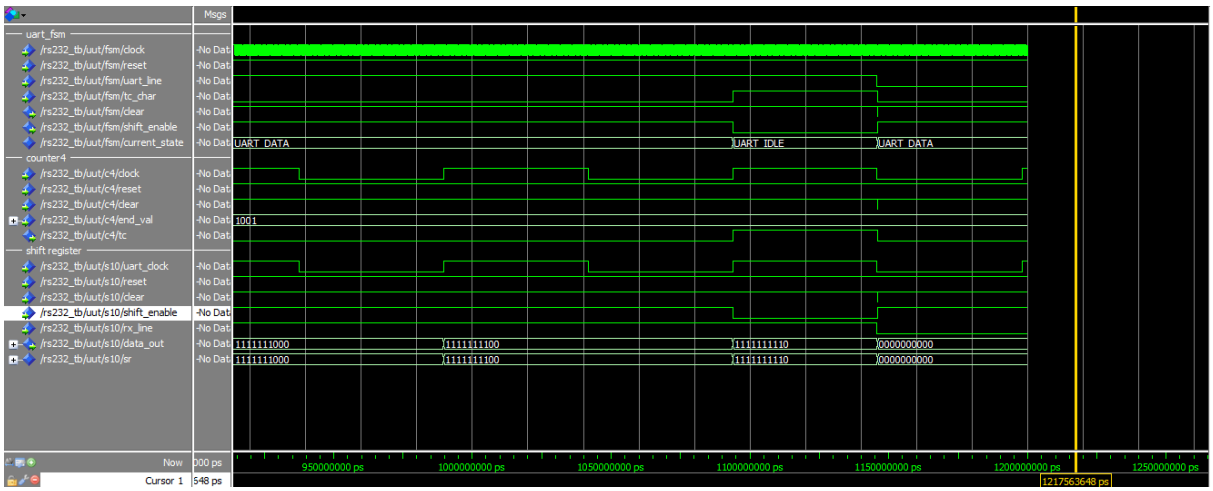

Figura 9: Fine di una trasmissione ed inizio della trasmissione successiva

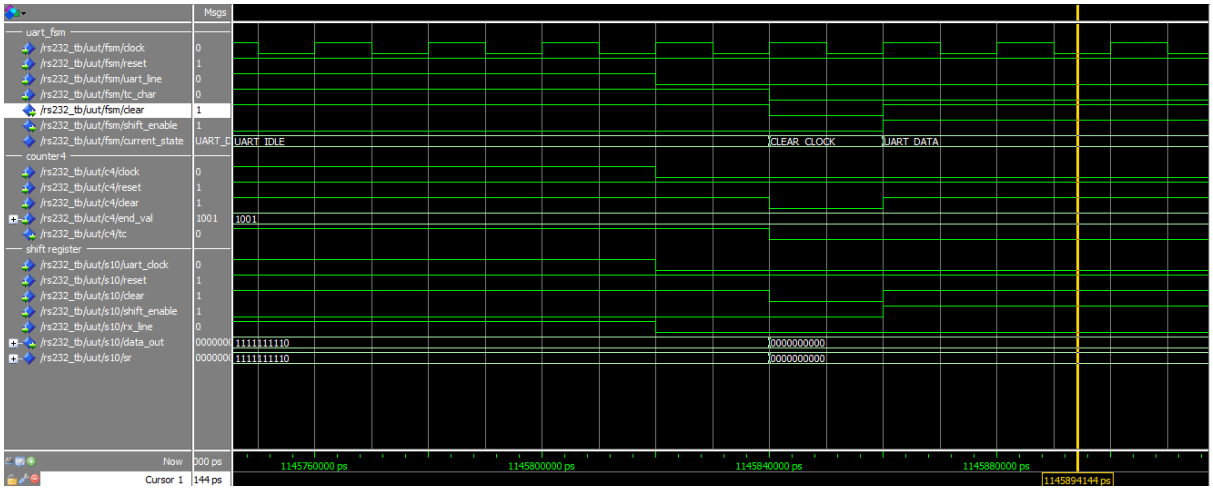

Figura 10: Zoom sullo stato clear_clock: dopo che abbiamo ricevuto il primo "start" byte il segnale di clear viene asserito e i contatori utilizzati per gestire lo shift register e lo shift register stesso vengono resettati

3.2 Modulo VGA

Per comandare uno schermo con controllore VGA é necessario inviare dei segnali che rispettino delle tempistiche ben precise, stabilite in base alla frequenza di refresh dello schermo. In particolare, dato che la consegna di questo laboratorio specifica come obiettivo l'ottenimento di un refresh a 60 Hz, vengono imposti un segnale di clock a 25 MHz per gestire i segnali e il seguente timing da rispettare:

	H Sync	Back Porch	Video Out	Front Porch
Durata	3.8 μs	1.9 μs	25.4 μs	0.6 μs
Colpi di clock	95	47.5	*635*	15
H Sync	BASSO	ALTO	ALTO	ALTO
RGB out	BASSO	BASSO	SEGNALE	BASSO

Tabella 1: Timing dei segnali di controllo VGA, scansione orizzontale

	V Sync	Back Porch	Video Out	Front Porch
Righe	2	33	480	10
V Sync	BASSO	ALTO	ALTO	ALTO
RGB out	BASSO	BASSO	SEGNALE	BASSO

Tabella 2: Timing dei segnali di controllo VGA, scansione verticale

Come si può vedere, si possono inviare segnali per accendere i led e fare effettivamente apparire qualcosa sullo schermo solo quando sia la scansione orizzontale che quella verticale si trovano nella regione "Video Out". E' necessario fare diverse considerazioni. Innanzitutto, facendo trascorrere solo 25.4 μs per scansionare l'area orizzontale disponibile non riusciamo a coprire tutti i 640 pixel nominali ma solo 635. Questo problema deriva dal fatto che il clock considerato per rispettare le tempistiche di cui sopra ha una frequenza leggermente più alta di 25 MHz (25.2 MHz circa).

Questo non pone un ostacolo considerevole perché é sufficiente allungare il ciclo di Video Out di 0.2 μs. Si tratta di un tempo sufficientemente breve per evitare che ci siano intoppi di sorta nella gestione dello schermo. In questo modo riusciamo a scansionare 640 pixel in orizzontale in altrettanti colpi di clock. Inoltre, i colpi di clock da far trascorrere durante il periodo di Back Porch sono nominalmente 47.5, ma data

la difficoltà intrinseca di calcolare mezzo colpo di clock abbiamo deciso di approssimare la cifra a 47. Ridurre questa fase di 0.5 μs, com'é facilmente intuibile, non pone nessun problema nel nostro caso.

Per quanto riguarda il timing della scansione verticale, il nostro compito é facilitato perché é sufficiente contare quante righe sono state già scansionate per individuare in che stato ci troviamo e gestirlo di conseguenza.

Per realizzare nella pratica quanto appena descritto abbiamo progettato in VHDL un modulo contenente due unità di controllo ed un datapath, contenente tutti gli elementi necessari per contare la riga e la colonna attuale, generare un clock a 25 MHz e, infine, accendere o spegnere i LED dello schermo in base alla posizione corrente del quadrato ed alla sua dimensione. Posizione e dimensione vengono aggiornate di volta in volta dai dati in arrivo dal modulo che gestisce la comunicazione seriale. In questo modo la scansione verticale e la scansione orizzontale sono gestite contemporaneamente.

3.2.1 Datapath

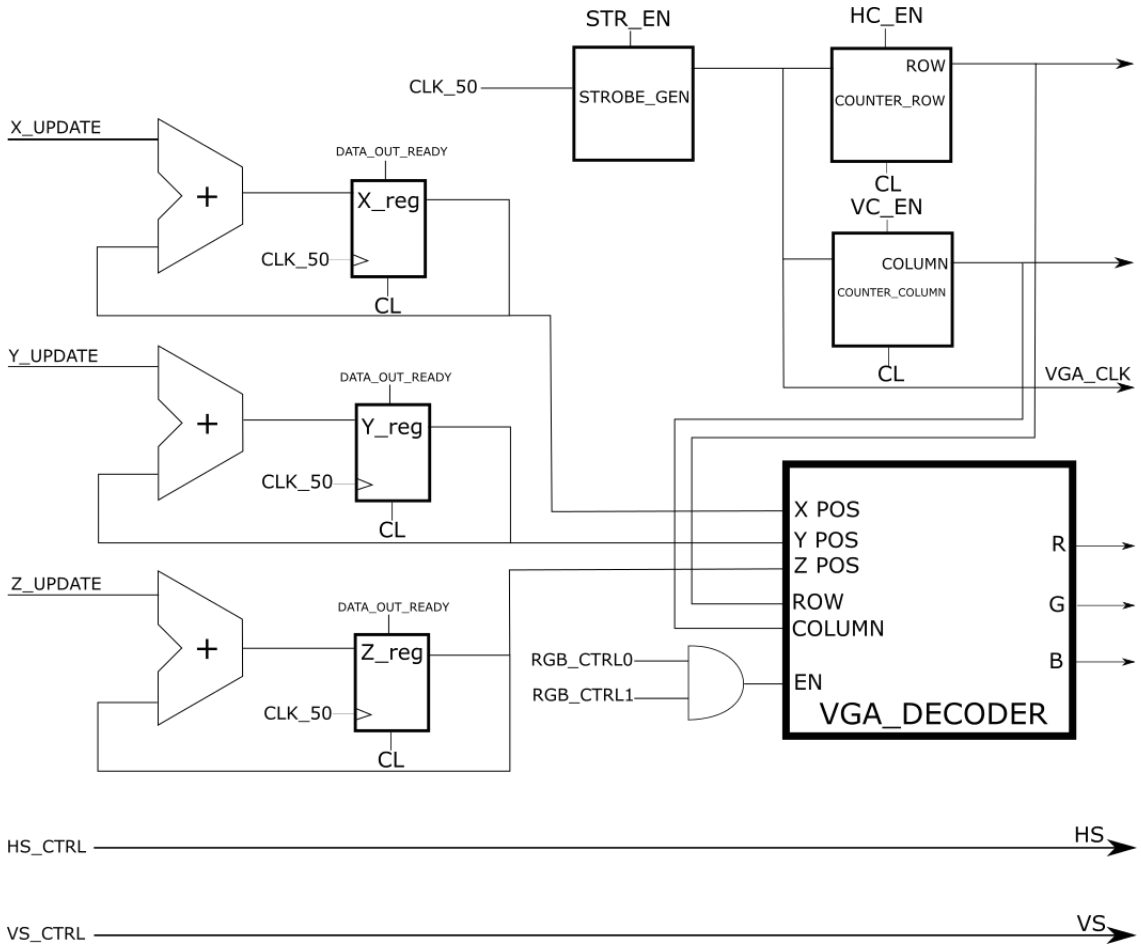

Figura 11: Datapath

In questa sezione descriveremo nel dettaglio tutti gli elementi contenuti nel Datapath e il loro funzionamento.

- **Sommatori**: tre sommatori utilizzati per aggiornare la posizione sui tre assi. Agli ingressi ricevono il dato precedente, proveniente da un registro posto a valle (X_reg, Y_reg o Z_reg, in base alla posizione), e lo spostamento lungo gli assi, in arrivo dal modulo che gestisce la lettura dalla linea seriale.

- **Registri (X_reg, Y_reg, Z_reg)**: tre registri utilizzati per contenere la spostamento dalla posizione iniziale, considerati i 3 assi. Il segnale di enable viene dato dal modulo che gestisce la lettura dalla linea seriale ed é denominato DATA_OUT_READY. Tale segnale si alza per un colpo di clock non appena i 3 dati sulla posizione sono validi, cosicché il registro possa campionare la posizione aggiornata solo quando sono arrivati dei dati validi dall'ingresso seriale, e comunque solo una volta.

- **STROBE_GEN**: questo modulo accetta in ingresso il clock a 50 MHz e fa uscire un clock avente una frequenza di 25 MHz. Tale segnale viene mandato ai contatori di colonna e di riga e anche in uscita, dato che é necessario inviarlo al controllore stesso della VGA (ossia, é connesso all'uscita VGA_CLK).

- **COUNTER_ROW e COUNTER_COLUMN**: due contatori, rispettivamente uno per contare la posizione orizzontale e uno per la posizione verticale. Questi contatori hanno i segnali di clear e di enable provenienti dalle unità di controllo. I contatori aumentano se il segnale di enable é asserito (HC_EN o VC_EN) e se il segnale di clock a 25 MHz si trova sul fronte di discesa. Si é scelto di aumentare il contatore sul fronte di discesa perché in questo modo, in fase di simulazione, l'aumento corrisponde in maniera precisa ad un intervallo di $\frac{1}{25MHz}$=40 ns, facilitandoci il lavoro in fase di testing e validazione delle tempistiche. Inoltre, i dati di riga e colonna vengono inviati alle unità di controllo.

- **VGA_DECODER**: il decoder VGA prende in ingresso le uscite dei contatori di riga e di colonna e le confronta con la posizione lungo gli assi x, y e z per decidere se accendere i LED o meno. Per semplicità, come già detto, abbiamo deciso di stampare a schermo un quadrato di colore rosso, quindi, durante gli intervalli giusti, questo blocchetto si limita ad asserire tutti i bit del segnale R. Inoltre, per sicurezza, abbiamo fatto in modo che questa unità venga abilitata da due segnali di enable provenienti dalle due unità di controllo. Tali segnali sono ovviamente alti quando ciascuna unità é nella fase VIDEO_ON, ossia quando é ammesso inviare segnali ai LED.

  La seguente condizione, posta all'interno di un process, fa in modo che il quadrato appaia alla posizione corrente e che abbia la dimensione appropriata, considerata la distanza dell'accelerometro dallo schermo:

$$COLONNA >= H_BEGIN + 315 + X + Y$$

$$COLONNA <= H_END - 315 + X - Y$$

$$RIGA >= V_BEGIN + 235 + Z + Y$$

$$RIGA <= V_END - 235 + Z - Y$$

  H_BEGIN, H_END, V_BEGIN e V_END sono delle costanti che contengono il valore che i contatori devono assumere per arrivare rispettivamente all'inizio e alla fine della fase di VIDEO ON, sia per la scansione orizzontale che per quella verticale. Per essere precisi, H_BEGIN é pari a 95+47=142, H_END é pari a 95+47+640=782, V_BEGIN é pari a 2+33=35 e V_END é pari a 2+33+480=515.

  In assenza di spostamento, ossia quando X, Y e Z sono pari a zero, sullo schermo comparirà un quadrato rosso di dimensione 10 pixel x 10 pixel fra le coordinate x 315 e 325 e fra le coordinate z 235 e 245. Se viene rilevato uno spostamento lungo l'asse x, X viene aggiornato di conseguenza, con il risultato di spostare a destra o a sinistra sia il limite sinistro che il limite destro entro il quale il colore rosso viene acceso all'interno di una riga. Lo stesso discorso va fatto per quanto riguarda l'asse z.

  Infine il dato di posizione lungo l'asse y, che é quello entrante nello schermo, aumenta o diminuisce sia il limite inferiore che quello superiore, sia per quanto riguarda l'asse x che per l'asse z. In questo modo, infatti, aumentiamo l'area totale occupata dal quadrato sullo schermo. Da notare che, essendo l'asse y dell'accelerometro diretto verso lo schermo, un aumento positivo di tale coordinata corrisponde ad un rimpicciolimento del quadrato, e viceversa.

I segnali in uscita dal datapath sono il dato contenuto nel contatore di righe e nel contatore di colonne, il segnale di clock a 25 MHz, i segnali di controllo dei cristalli liquidi R, G e B, e infine i segnali HS (Horizontal Sync) e VS (Vertical Sync). Mentre i segnali provenienti dai contatori vengono inviati alle unità di controllo, così da poter stabilire in che stato ci si trovi, il resto dei segnali appena descritti viene inviato direttamente alla VGA.

3.2.2 Unità di controllo e timing

Abbiamo deciso di realizzare due unità di controllo per questa sezione perché in questo modo possiamo controllare indipendentemente i segnali di sincronismo orizzontale e verticale e i segnali che abilitano il componente VGA_DECODER. In caso contrario, avremmo avuto bisogno di una unità di controllo a 5*4 stati, per un totale di 20 stati, per gestire efficacemente i segnali di controllo dell'uscita VGA e del datapath.

Infatti, sarebbe stato necessario considerare i quattro stati della scansione orizzontale, più uno per asserire il segnale di enable del contatore di righe, per ciascuno dei quattro stati che regolano la scansione verticale. Utilizzando due unità di controllo abbiamo potuto semplificare notevolmente il nostro progetto, dato che esse hanno rispettivamente 5 e 4 stati.

Di seguito descriviamo i singoli stati di entrambe le unità di controllo. Ricordiamo che le ASM chart vanno interpretate nel seguente modo: ogni rettangolo contiene i segnali che vengono impostati a '1'. Se il segnale, durante un dato stato, é basso, non appare nel riquadro facente riferimento allo stato.

Unità di controllo H

Preposta al regolamento dei segnali durante la scansione orizzontale. Il passaggio da uno stato a un altro é regolato dalla cifra indicata dal contatore di posizione orizzontale.

- **RESET:** il segnale di clear viene asserito unicamente in questo stato. Tale segnale é inviato ai registri x_reg, y_reg e z_reg e al contatore di posizione orizzontale COUNTER_COLUMN. Il segnale di enable del generatore di clock a 25 MHz (STR_EN) é basso, così come il segnale di sincronismo orizzontale HS. In questo stato si entra quando il segnale di reset viene asserito (assegnato all'interruttore 0 della scheda FPGA). Lo stato successivo é sempre H_SYNC, a meno che il reset non sia asserito.

- **H_SYNC:** durante questo stato il segnale di sincronismo orizzontale HS viene abbassato. I segnali di enable del generatore di clock a 25 MHz (STR_EN) e del contatore orizzontale (HC_EN) sono asseriti, così come nel resto degli stati successivi. Lo stato seguente é BACK_PORCH e vi si entra quando il contatore orizzontale ha raggiunto il valore di 95.

- **BACK_PORCH:** identico allo stato precedente, fuorché l'asserimento di HS, che rimane asserito per tutti gli stati rimanenti. Lo stato successivo é VIDEO_ON e vi si entra quando il contatore orizzontale ha raggiunto il valore di 142.

- **VIDEO_ON:** durante questo stato tutti i segnali di enable gestiti da questa unità di controllo sono asseriti, incluso uno dei due segnali di enable del decoder VGA (RGB_EN). Lo stato successivo é FRONT_PORCH e vi si entra quando il contatore orizzontale ha raggiunto il valore di 782.

- **FRONT_PORCH:** questo stato é identico allo stato di BACK_PORCH a livello di segnali, ossia il segnale RGB_EN viene negato, mentre HS, STR_EN e HC_EN sono asseriti. Lo stato successivo é UPDATE_ROW e vi si entra quando il contatore orizzontale ha raggiunto il valore massimo raggiungibile meno uno, ossia 796.

- **UPDATE_ROW:** durante questo stato viene asserito VC_EN per permettere al contatore verticale di aumentare di uno al prossimo scatto del contatore orizzontale, ossia quando il contatore orizzontale sarà reimpostato a 0. In questo modo garantiamo che un colpo del contatore verticale corrisponda più precisamente possibile all'intervallo di scansione di un'intera riga. Durante questo stato RGB_EN

rimane negato, ovviamente, mentre tutti gli altri segnali di controllo sono asseriti. Lo stato successivo é H_SYNC e vi si entra quando il contatore orizzontale ha fatto un singolo scatto.

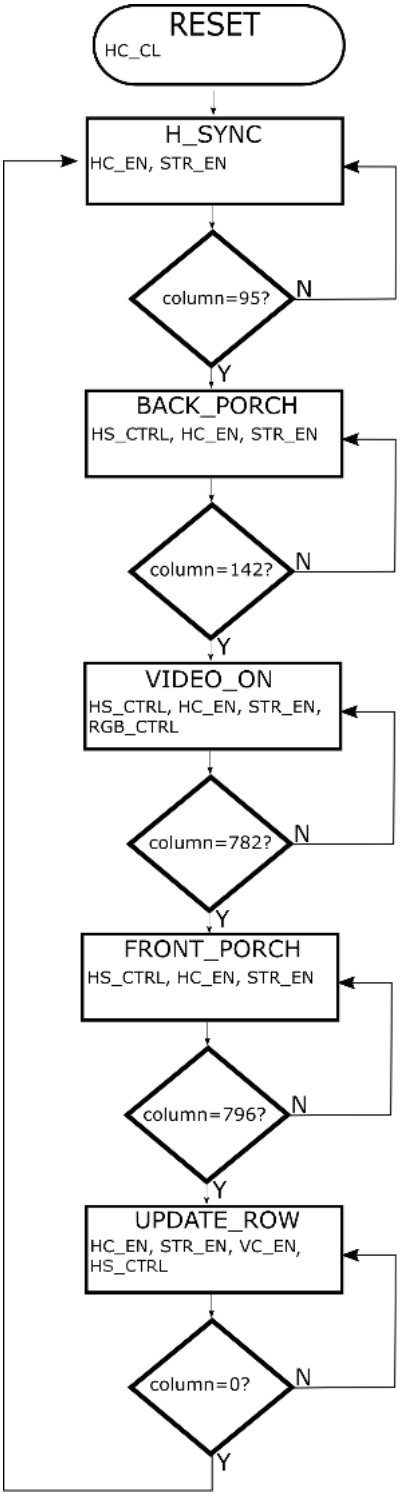

Figura 12: ASM chart della unità di Controllo per la scansione orizzontale

Unità di controllo V

Preposta al regolamento dei segnali lungo la scansione verticale. Anche in questo caso il passaggio da uno stato a un altro é regolato dalla cifra indicata dal contatore di posizione, in questo caso verticale.

Da notare che questa unità di controllo gestisce una quantità limitata di segnali rispetto all'altra, ossia solo HS_CTRL, VC_EN, VC_CL e RGB_CTRL. Tale scelta é stata fatta poiché la seconda unità di controllo é secondaria rispetto alla prima, dato che essa può cambiare stato solo se la prima funziona correttamente (infatti, il contatore di riga viene attivato dalla unità di controllo H). Pertanto abbiamo ritenuto che non valesse la pena renderla in grado di controllare altri segnali in concomitanza con la prima, come ad esempio quello di STR_EN.

- **RESET:** durante questo stato il segnale di sincronismo verticale VS e il segnale di enable del decoder VGA RGB_EN sono negati, mentre il segnale di clear del contatore verticale VC_CLEAR viene asserito. In questo stato si entra quando il segnale di reset viene asserito (assegnato all'interruttore 0 della scheda FPGA). Lo stato successivo é V_SYNC, a meno che il reset non sia asserito.

- **V_SYNC:** durante questo stato il segnale di sincronismo verticale VS viene abbassato. Lo stato successivo é BACK_PORCH e vi si entra quando il contatore verticale ha raggiunto il valore di 2.

- **BACK_PORCH:** identico allo stato precedente, fuorché l'asserimento di VS, che rimane asserito per tutti gli stati rimanenti. Lo stato successivo é VIDEO_ON e vi si entra quando il contatore orizzontale ha raggiunto il valore di 35.

- **VIDEO_ON:** durante questo stato il segnale di enable del decoder VGA (RGB_EN) viene asserito. Lo stato successivo é FRONT_PORCH e vi si entra quando il contatore verticale ha raggiunto il valore di 515.

- **FRONT_PORCH:** durante questo stato il segnale RGB_EN viene negato mentre VS rimane ovviamente asserito. Lo stato successivo é V_SYNC e vi si entra quando il contatore verticale si é resettato tornando a zero, ossia non appena ha raggiunto il valore massimo (525).

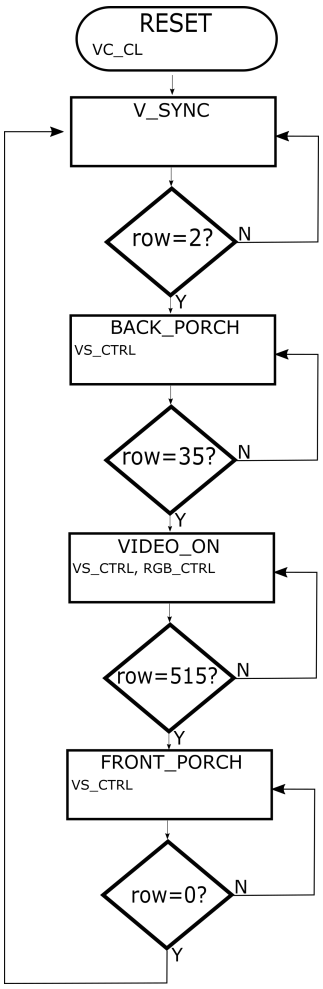

Figura 13: ASM chart della unità di Controllo per la scansione verticale

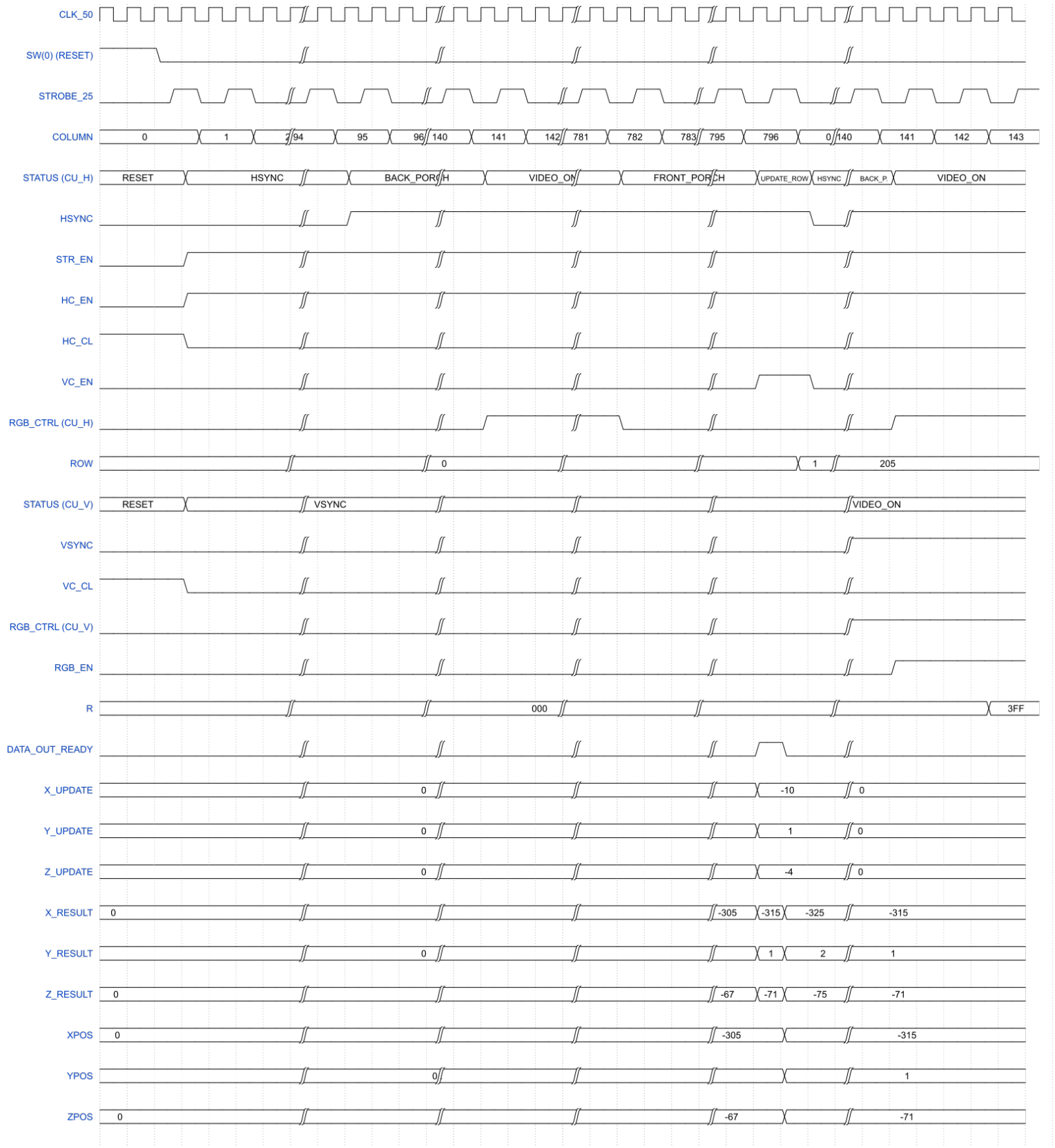

Figura 14: Timing Diagram del modulo VGA

3.2.3 Analisi del timing e validazione del componente

Di seguito riportiamo l'esito dei test del componente su Modelsim. In particolare, le tempistiche desiderate per la scansione orizzontale sono o perfettamente rispettate oppure appena differenti dal valore desiderato. Nello specifico:

- fra lo stato HSYNC e lo stato BACK_PORCH intercorrono 3.8 μs.
- fra BACK_PORCH e VIDEO_ON intercorrono 18.8 μs anziché 18.9. Questo é dovuto al fatto che non contiamo fino a 142.5 ma solo fino a 142. Come già detto, tuttavia un errore di 10 ns non é assolutamente influente ai fini della funzionalità di questo componente;
- fra VIDEO_ON e FRONT_PORCH intercorrono esattamente 25.6 secondi, come desiderato;
- fra FRONT_PORCH e UPDATE_ROW intercorrono 0.6 μs, come desiderato.

La posizione di riga viene incrementata al momento giusto e il segnale VGA_R assume il valore 3FF quando le coordinate puntano ad una zona dello schermo occupata dal quadrato. Infine, l'aggiornamento della posizione del quadrato avviene esattamente come desiderato, ossia all'arrivo di nuovi dati dalla linea seriale.

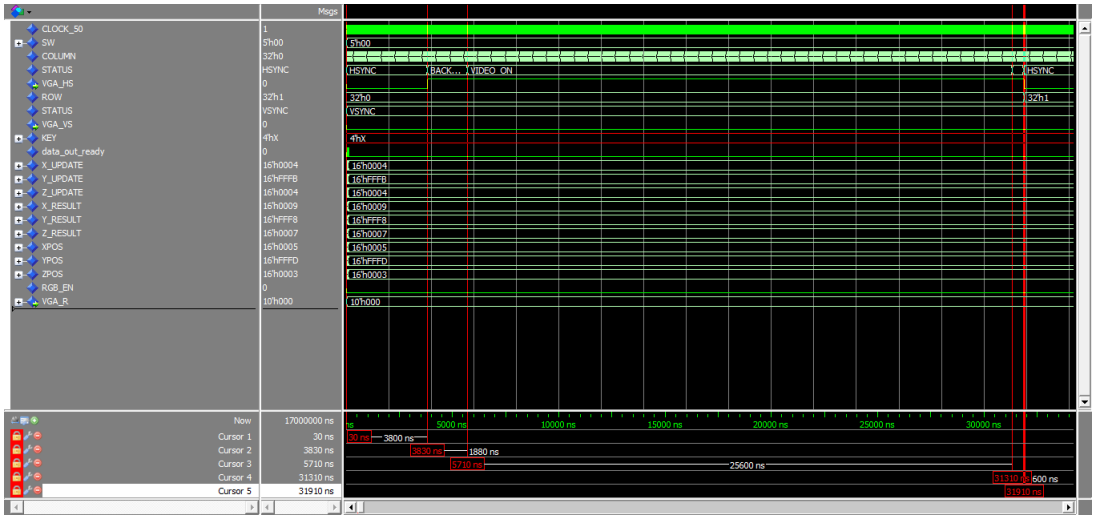

Figura 15: Timing dei segnali di controllo (scansione orizzontale)

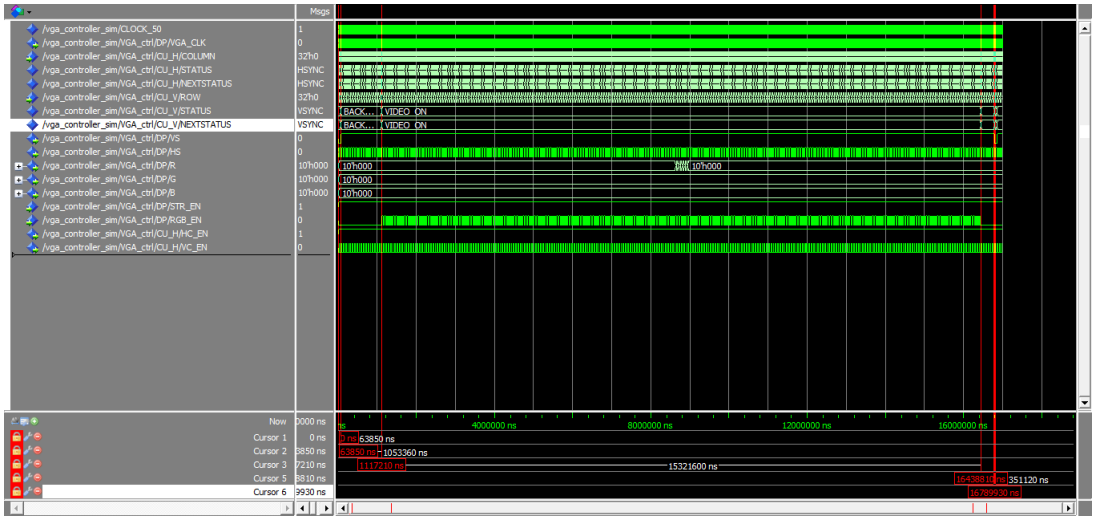

Figura 16: Timing dei segnali di controllo (scansione verticale)

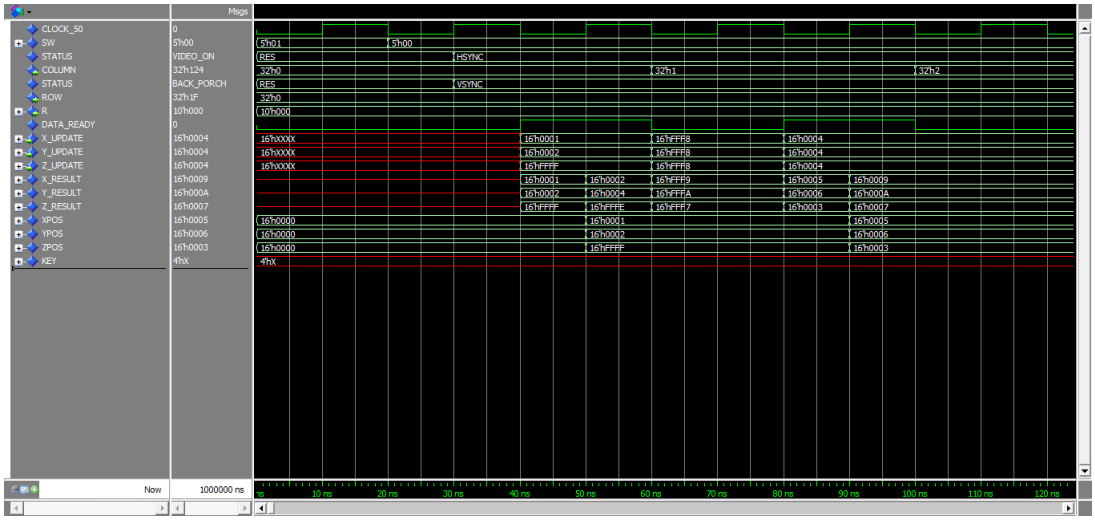

Figura 17: Aggiornamento della posizione del quadrato tramite dati in arrivo dalla linea seriale

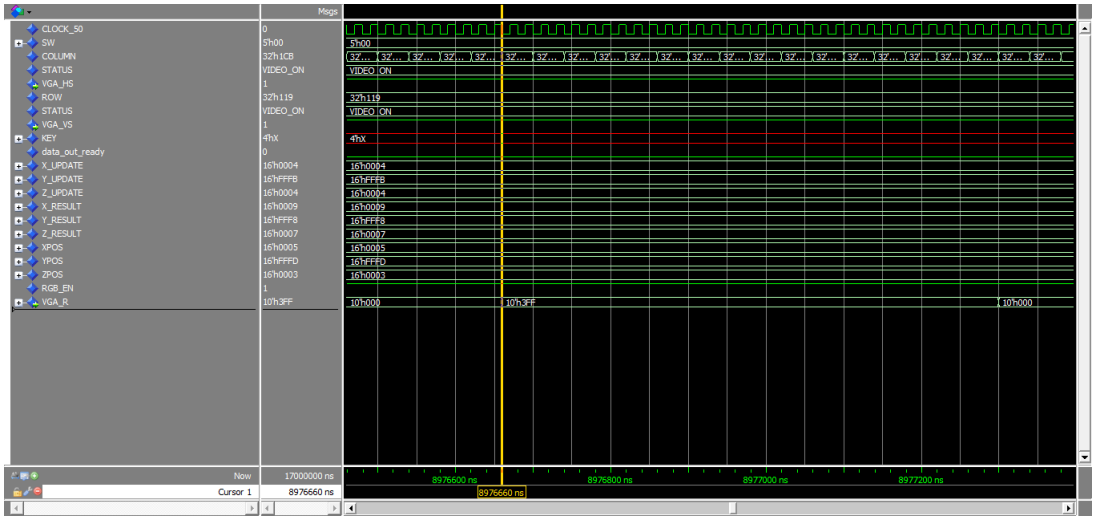

Figura 18: R viene settato a 3FF quando i contatori raggiungono delle coordinate dove il quadrato é presente

4 Test

4.1 Test del microcontrollore

Per sincerarci che il microcontrollore Nucleo F401RE rispettasse il nostro progetto abbiamo innanzitutto analizzato i segnali in uscita dai pin SCL e SDA tramite oscilloscopio. Le forme d'onda rilevate erano esattamente quelle che ci aspettavamo: SCL assume un aspetto simile a quello di un segnale di clock una volta ricevuta una transizione spuria da parte di SDA, mentre SDA, durante una trasmissione, commuta solo durante il fronte di discesa di SCL.

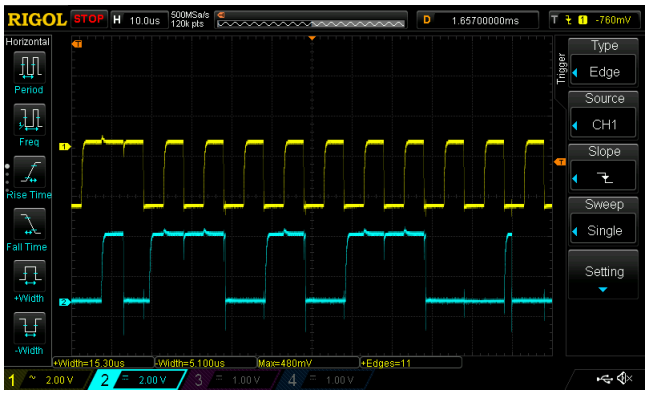

Figura 19: Segnali comunicazione via protocollo I2C. In alto SCL, in basso SDA

Infine abbiamo testato tramite Terminal il corretto invio dei dati tramite protocollo seriale. Anche questo test é stato positivo, nel senso che abbiamo sempre visualizzato i byte di inizio e di fine trasmissione, nonché valori ragionevoli per gli spostamenti lungo gli assi x, y e z, ossia fra 0 e $\pm$ 10 pixel. A tale riguardo, occorre tenere presente che il microcontrollore é stato programmato per inviare uno spostamento relativo e non assoluto, dunque ha senso che per spostamenti non estremamente impulsivi i dati inviati siano dell'ordine di grandezza di cui sopra.

4.2 Test del componente di gestione della seriale

Per testare questo componente abbiamo previsto innanzitutto che l'uscita GPIO_0(0) mostrasse il segnale ricevuto dal microcontrollore, così da poterci sincerare che i segnali inviati fossero uguali a quelli ricevuti dalla FPGA.

Abbiamo anche usato i LED LEDR(0) e LEDG(0) per ricevere informazioni sullo stato del trasferimento. In particolare, LEDR(0) é connesso all'output di stato del componente di gestione della seriale e si spegne qualora una trasmissione sia in corso. La frequenza di accensione e spegnimento può essere intesa, in termini estremamente lassi, come un indice della frequenza di comunicazione seriale. LEDG(0) invece segue il segnale di data_out_ready, ossia il segnale che regola l'aggiornamento della posizione del quadrato. Dato che questo segnale deve rimanere alto per un solo colpo di clock, ci aspettavamo di vederlo sempre spento. Se così non fosse, il LED LEDG(0) segnalerebbe un aggiornamento non corretto del dato di posizione.

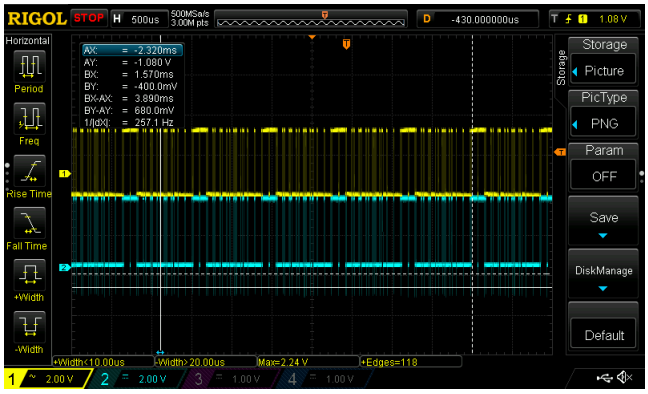

Figura 20: In alto: segnale in uscita dal microcontrollore, in basso: segnale in uscita dal pin GPIO_0(0), che rispecchia il segnale in ingresso della comunicazione seriale, ossia GPIO_1(13). Essendo i due segnali identici abbiamo potuto confermare la corretta ricezione. Da notare la presenza dei byte di start e di stop all'inizio e alla fine della trasmissione dei 48 bit di posizione

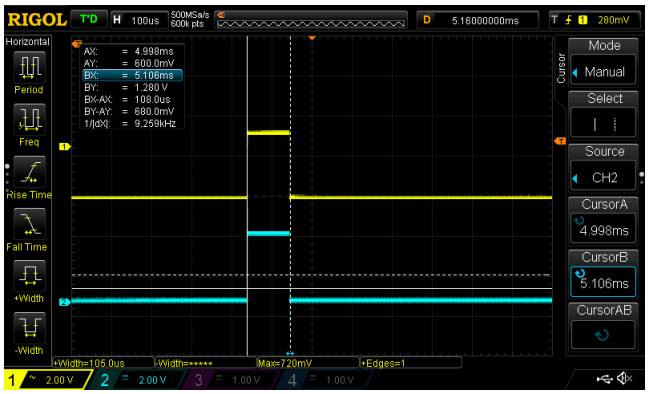

Figura 21: In questa immagine viene riportata la persistenza approssimativa di un bit durante il protocollo di comunicazione seriale. Tale intervallo é stato valutato in maniera approssimativa mediante i cursori e ammonta a circa 108 μs, denotando un baud rate vicino a quello atteso, ossia 9600

L'esito di tutti questi test é stato positivo. Da GPIO_0(0) abbiamo visto uscire una replica esatta di quanto viene ricevuto all'ingresso GPIO_1(13), ossia di quello che viene inviato tramite seriale dal microcontrollore. Dal momento in cui abbiamo connesso il microcontrollore alla FPGA il LED LEDR(0) ha cominciato

a lampeggiare, segnalando una trasmissione in corso. Inoltre, come ci aspettavamo, il LED LEDG(0) sembra sempre spento.

Come ultima nota, l'interruttore sw(1) é stato predisposto per selezionare, fra UART_RXD e GPIO_1(13), l'ingresso da connettere al componente di gestione della linea seriale. Questa scelta é stata fatta perché inizialmente non eravamo sicuri di dove sarebbe stato opportuno connettere l'uscita seriale del microcontrollore. Tale interruttore, inoltre, seleziona quale fra i due ingressi inviare al pin GPIO_0(0), così da poter essere in grado di analizzare il segnale in arrivo in entrambi i casi.

Tale interruttore si é rivelato poco utile in quanto il pin UART_RXD, facente parte dell'ingresso RS-232, non può essere utilizzato da solo per ricevere un segnale da una linea seriale. Dunque, per tutti i test abbiamo sempre tenuto sw(1) alto, selezionando come ingresso GPIO_1(13) e connettendo sempre l'uscita seriale del microcontrollore a tale pin.

4.3 Test del componente di controllo della VGA

In laboratorio il test del componente ha dato esito positivo, essendo in grado di disegnare un quadrato di 10 pixel per 10 al centro dello schermo. Per essere sicuri che tutto funzionasse a dovere, abbiamo fatto una lettura dei segnali di HSYNC e VSYNC tramite oscilloscopio e abbiamo ottenuto la forma d'onda e le tempistiche attese.

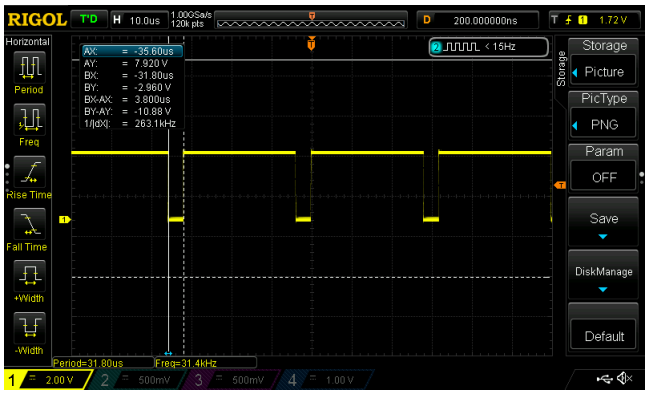

Figura 22: Segnale di sincronismo orizzontale (HSYNC)

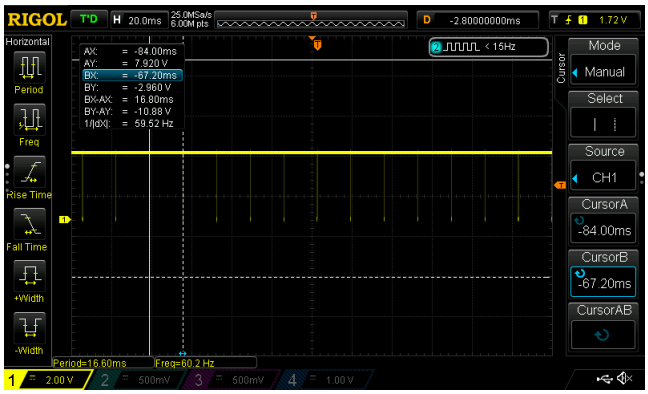

Figura 23: Segnale di sincronismo verticale (VSYNC)

Chiaramente nella fase iniziale dei test il quadrato rimaneva immobile al centro dello schermo, dato che il modulo di controllo VGA non era collegato ancora a un componente che determinasse l'aggiornamento dei

registri di posizione x, y e z. Per sincerarci che questa sezione del progetto funzionasse, abbiamo impostato i tasti KEY(0), KEY(1) e KEY(2) per forzare uno spostamento lungo i 3 assi.

Nello specifico, il tasto KEY(0) determina uno spostamento a destra, KEY(1) a sinistra, KEY(2) alternativamente in alto o in basso. Da notare che KEY(3) non é stato usato per il semplice fatto che la board che avevamo a disposizione aveva tale input non funzionante. Da notare inoltre che il tasto KEY(2) poteva essere usato anche per rimpicciolire o ingrandire il quadrato. I seguenti interruttori regolano le operazioni appena descritte:

- sw(2): se alzato attiva la modalità di spostamento tramite bottoni;
- sw(3): se basso impone che il tasto KEY(2) si occupi dell'aggiornamento della posizione z (altezza), se alto impone che tale tasto aggiorni la posizione y (dimensione);
- sw(4): se basso impone che il tasto KEY(2) determini un aumento del valore di posizione a cui si riferisce (ossia lungo z o lungo y) o una diminuzione se alto;

Da notare che la risoluzione di spostamento ammonta ad un singolo pixel. Per aggiornare la posizione a schermo abbiamo fatto entrare in ciascun sommatore il valore 1 o -1, in base alla pressione di un determinato tasto, come sopra.

Tali valori, nel caso dell'utilizzo di 16 bit e di rappresentazione in complemento a 2, in esadecimale si traducono in 0001 e FFFF. Al tempo stesso, quando un tasto viene premuto, viene asserito il segnale di enable dei registri posti per acquisire il dato di spostamento.

Infine, é stata prevista una procedura tale che la pressione di un tasto comporti un singolo spostamento prima che venga rilasciato. In caso contrario, infatti, dato un campionamento a 50 MHz, alla pressione di un tasto sarebbe impossibile sottrarre o sommare solo 1 al dato di posizione corrente ma si vedrebbe subito il quadrato sparire dallo schermo.

Questo test ha dato esito positivo. Il quadrato si sposta esattamente come dovrebbe, ossia:

- a destra di 1 pixel qualora il dato lungo l'asse x sia 1, a sinistra di un pixel qualora il dato lungo l'asse x sia -1;
- in alto di un pixel qualora il dato lungo l'asse z sia -1, in basso di un pixel qualora il dato lungo l'asse z sia 1;
- si allarga di due pixel sia lungo l'asse z che lungo l'asse x se lungo l'asse y arriva uno spostamento di -1, viceversa se lo spostamento in arrivo é pari a 1;

Riportiamo una foto presa dall'oscilloscopio che mostra il segnale R in uscita dal relativo pin VGA.

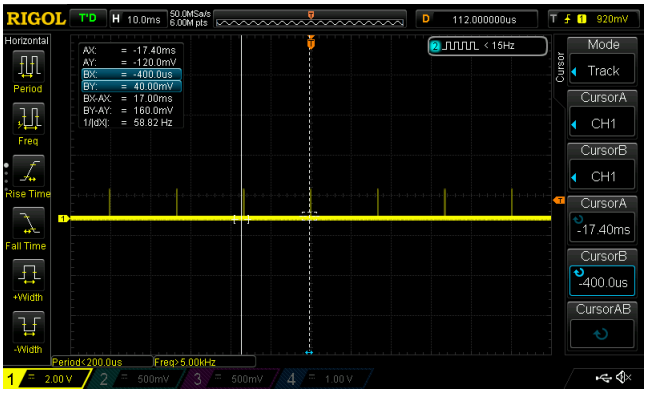

Figura 24: Segnale VGA per il controllo del colore rosso. Da notare che, in assenza di spostamento, la distanza fra una serie di valori alti e un'altra é di ca. 17 ms, ossia ca. 60 Hz, pari alla frequenza di refresh dello schermo.

4.4 Test complessivo

Connettendo tutte le componenti e facendo un test complessivo abbiamo ottenuto un risultato soddisfacente, nel senso che il quadrato viene rappresentato correttamente e si muove seguendo molto precisamente lo spostamento del microcontrollore. Inoltre, esso si ingrandisce o si rimpicciolisce rispettivamente se allontaniamo o avviciniamo il microcontrollore (ossia l'accelerometro) allo schermo.

Figura 25: Schermo all'inizio del test

Figura 26: Schermo dopo aver spostato il microcontrollore in avanti

Figura 27: Schermo dopo aver spostato il microcontrollore a sinistra, in basso e all'indietro

5 Conclusione

L'esito del nostro progetto é soddisfacente, nel senso che i componenti da noi impiegati svolgono pienamente il loro dovere, rispettivamente:

- l'accelerometro rileva un'accelerazione lungo i tre assi e la invia al microcontrollore, che la trasforma in uno spostamento relativo;

- il microcontrollore invia tale spostamento alla scheda FPGA tramite linea seriale;
- la scheda FPGA riceve il dato di spostamento, lo salva e infine aggiorna la posizione a schermo di un quadrato;
- infine, la scheda FPGA gestisce opportunamente i segnali VGA per poter stampare un'immagine a schermo;

Il nostro progetto é comunque passibile di miglioramento in vari sensi.

Essendo limitati negli spostamenti dai jumper connessi fra microcontrollore e FPGA, non é stato possibile determinare quanto adeguata fosse la sensibilità con cui il quadrato si sposta sullo schermo rispetto ad uno movimento dell'accelerometro, anche se durante i test ci é persa ottimale.

Nello stato attuale, il quadrato può uscire dallo schermo, ossia i valori di spostamento lungo gli assi possono assumere valori superiori al range massimo consentito data una risoluzione di 640x480 pixel. In tale senso, nel datapath del componente di controllo VGA si potrebbe pensare di inserire 3 blocchi in grado di far saturare il dato di spostamento qualora si superi il massimo consentito, così da poter sempre confinare il quadrato all'interno dello schermo.

Infine, attualmente, usiamo solo 4 dei 16 bit totali di dinamica disponibili per rappresentare lo spostamento. Questo é dovuto al fatto che si é scelto di rappresentare i dati provenienti dal microcontrollore come spostamenti relativi, dunque in generale non riceveremo mai valori che vadano rappresentati con un numero di bit pari a 16. Per evitare tale spreco si potrebbe pensare di riprogettare il modo in cui i dati inviati dal microcontrollore vengono usati per spostare il quadrato a schermo, oppure, più semplicemente, potremmo ridurre il parallelismo interno della FPGA per quanto riguarda tali segnali.

6 Appendice: testbench

Abbiamo realizzato due testbench. Il primo, chiamato VGA_sim.vhd, é stato utilizzato per testare il componente di controllo della VGA, mentre l'altro, chiamato RS232_TB, é stato sfruttato per testare il componente di gestione della linea seriale.

Nello specifico, il primo é stato utilizzato per testare la durata delle singole fasi di sincronismo orizzontale e verticale, nonché il corretto aggiornamento del dato di posizione del quadrato. Il secondo invece é stato sfruttato per sincerarci che, inviando un dato tramite protocollo seriale, esso venisse ricevuto e salvato correttamente nei tre registri contenenti gli spostamenti relativi rispetto ai 3 assi. Esso é anche stato usato per appurare che il segnale data_out_ready venisse asserito per un solo colpo di clock al termine di una trasmissione.