



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Progetto di Sistemi Elettronici per l'Internet Of Things

Implementazione lettura/ scrittura con tastierino e display LCD interfacciati su FPGA, con comunicazione SPI e invio dati su Thingspeak

Docente:
Prof. Emiliano Sisinni

Studenti:
Stefano Molari, Matricola N° 727197
Daniil Kretinin, Matricola N° 729256

Indice

1.	Introduzione	1
1.1.	Hardware utilizzato.....	1
1.2.	FPGA.....	2
1.3.	VHDL	3
2.	Programmazione FPGA.....	4
2.1.	Interfacciamento dei dispositivi.....	4
2.2.	Assegnamento pin.....	5
2.3.	Controller del tastierino numerico	6
2.4.	Controller del display	7
2.5.	SPI.....	8
2.6.	Macchina a stati dell'SPI.....	10
2.7.	Main_fsm.....	11
2.8.	Visione complessiva dei VHDL	12
3.	Simulazione con ModelSim	13
3.1.	Testbench complessivo del sistema.....	13
4.	Programmazione HPS con ThingSpeak	14
4.1.	Firmware in C	14
4.2.	Cross-compilazione.....	15
4.3.	Connessione dell'HPS.....	15
4.4.	Test della comunicazione SPI	17
4.5.	ThingSpeak.....	17
5.	Risultati	19
5.1.	Specifiche rispettate	19
5.2.	Sviluppi futuri	20

1. Introduzione

1.1. Hardware utilizzato

Nel progetto viene utilizzata la scheda DE10-NANO (Figura 1), costituita principalmente da:

- Parte FPGA (Field Programmable Gated Array), una Altera Cyclone V;
 - Parte HPS (Hard Processor System) montato a bordo della scheda come System On Chip.

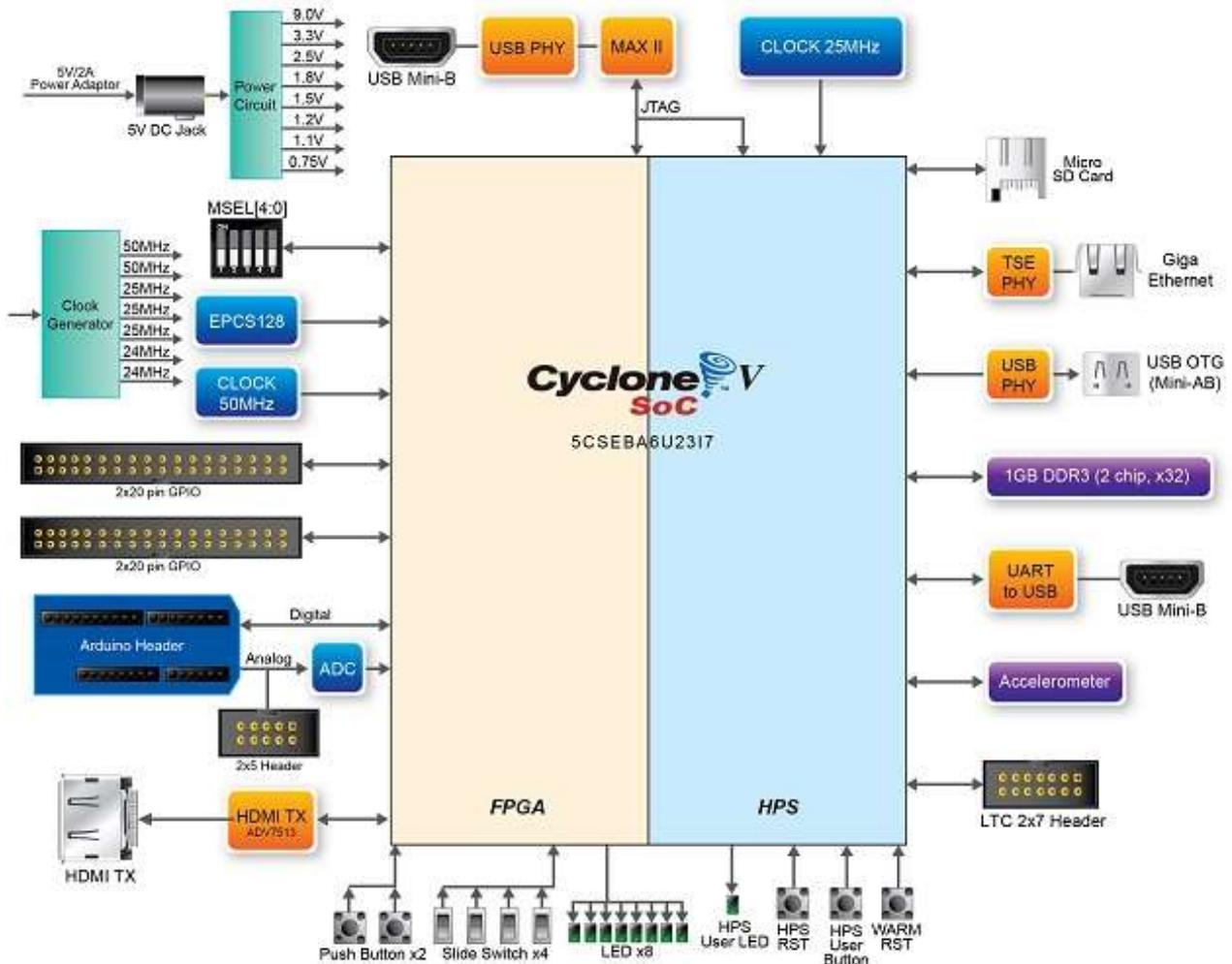


Figura 1 - Diagramma a blocchi della DE10-NANO Cyclone 5

L'obiettivo del progetto è l'implementazione di uno schermo LCD (Liquid Crystal Display) 16x2 che mostri a video i dati digitati da un tastierino di 10 cifre (da 0 a 9): entrambi i dispositivi sono collegati in locale all'FPGA. Quest'ultima passa i dati all'HPS interfacciandosi con una comunicazione SPI (Serial Peripheral Interface) per poi inoltrarli in cloud alla piattaforma Thingspeak.

1.2. FPGA

Le schede FPGA (Field Programmable Gate Array) sono dispositivi logici programmabili, nel senso che le funzioni logiche vengono realizzate per mezzo della chiusura/apertura degli switch comandati elettricamente: in questo modo le operazioni si possono eseguire in parallelo (mentre nei comuni µC si eseguono in sequenza).

Il concetto di programmabilità è illustrato anche con un confronto tra una semplice matrice di percorsi non programmata (Figura 2) e una matrice opportunamente programmata (Figura 3) per svolgere specifiche funzioni (tra cui multiplexing, buffering, ecc.).

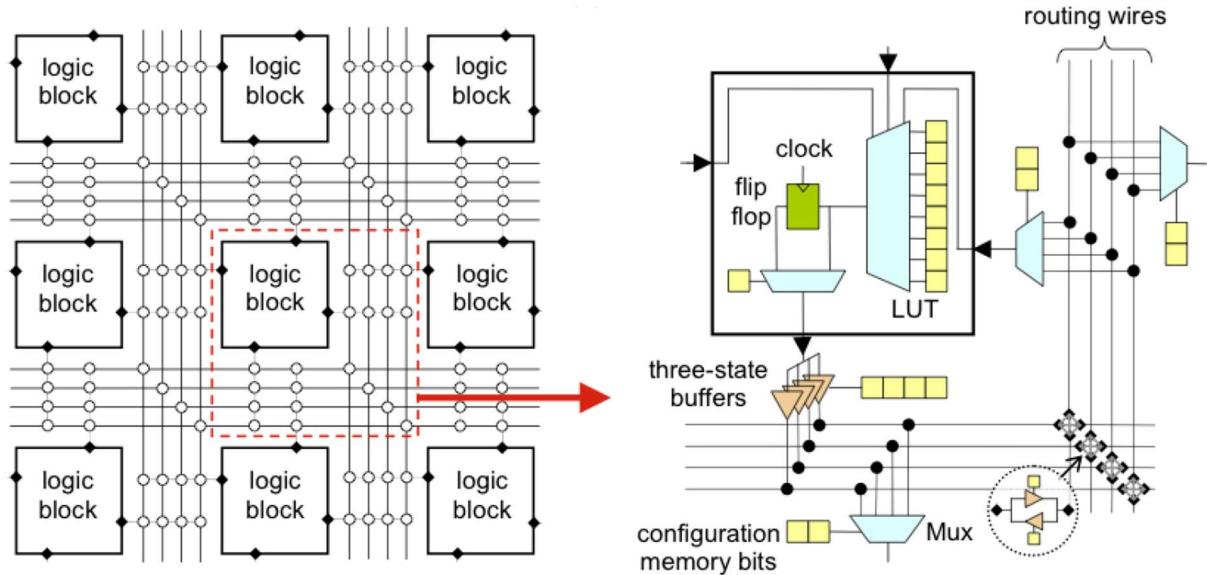


Figura 2 - Matrice non ancora programmata

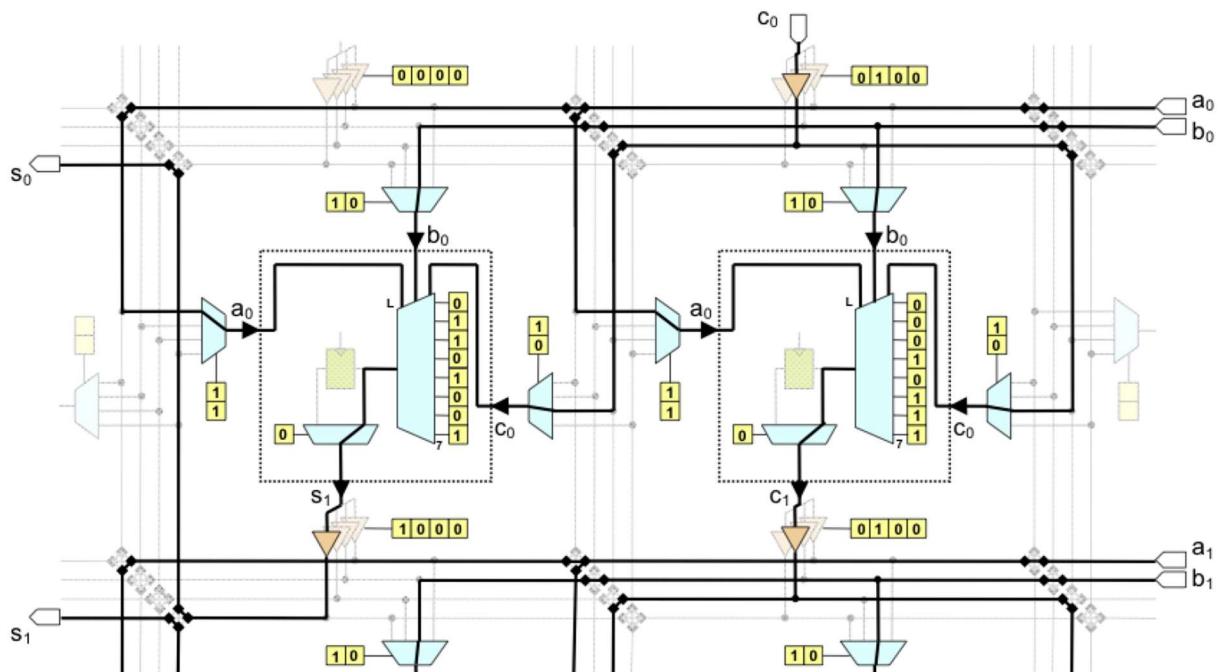


Figura 3 - Matrice programmata elettricamente

1.3. VHDL

Il linguaggio utilizzato per la descrizione hardware è il VHDL (Very High Speed Integrated Circuits Hardware Description Language): si tratta di definire dettagliatamente il comportamento e la struttura di sistemi elettronici complessi, sia a livello di logica combinatoria che sequenziale.

Questo linguaggio viene poi convertito in circuito tramite Quartus Prime che fa da sintetizzatore: in questo modo è possibile descrivere e simulare circuiti integrati e sistemi digitali, facilitando il processo di progettazione e verifica.

In Figura 4 sono raffigurati gli step per passare da un file VHDL a un circuito sintetizzato, mentre in Figura 5 viene raffigurato l'insieme dei tool e i collegamenti per le FPGA Altera.

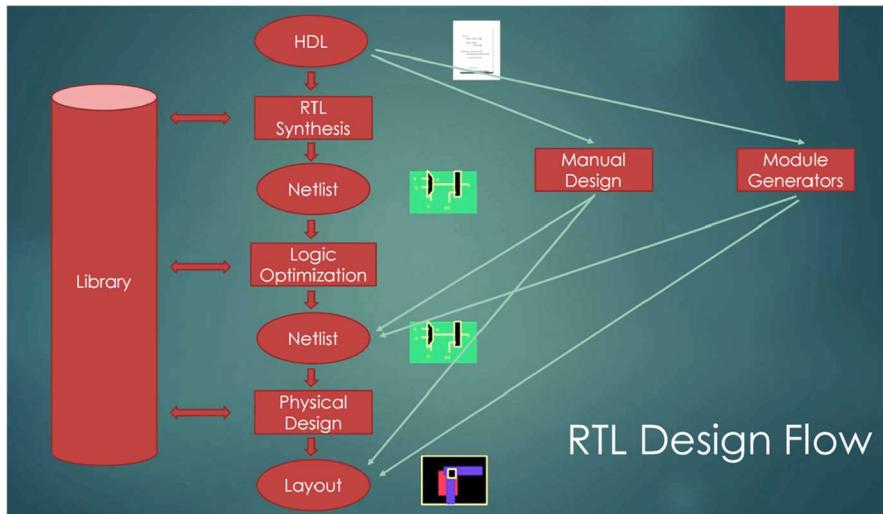


Figura 4 - Register Transfer Level

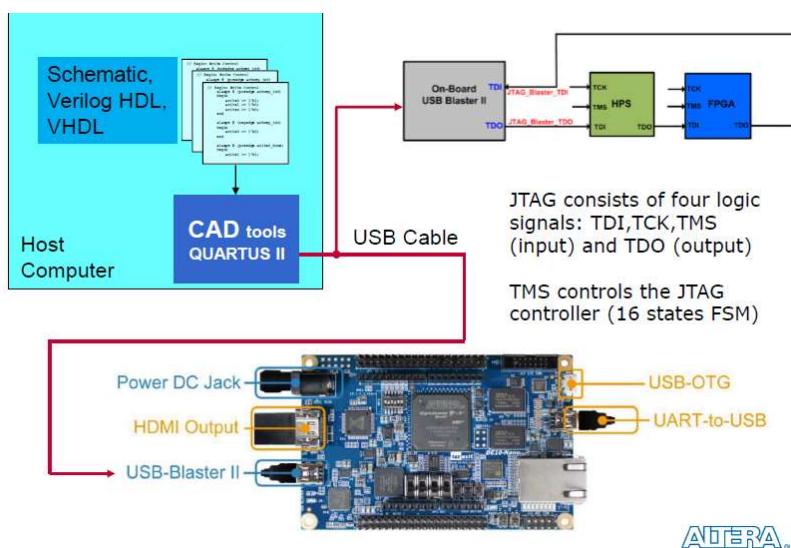


Figura 5 - Altera Quartus II CAD Tools

2. Programmazione FPGA

2.1. Interfacciamento dei dispositivi

Prima di dettagliare i singoli moduli, mostriamo una panoramica di com'è strutturato il progetto nell'insieme: in Figura 6 viene riportato lo schema a blocchi.

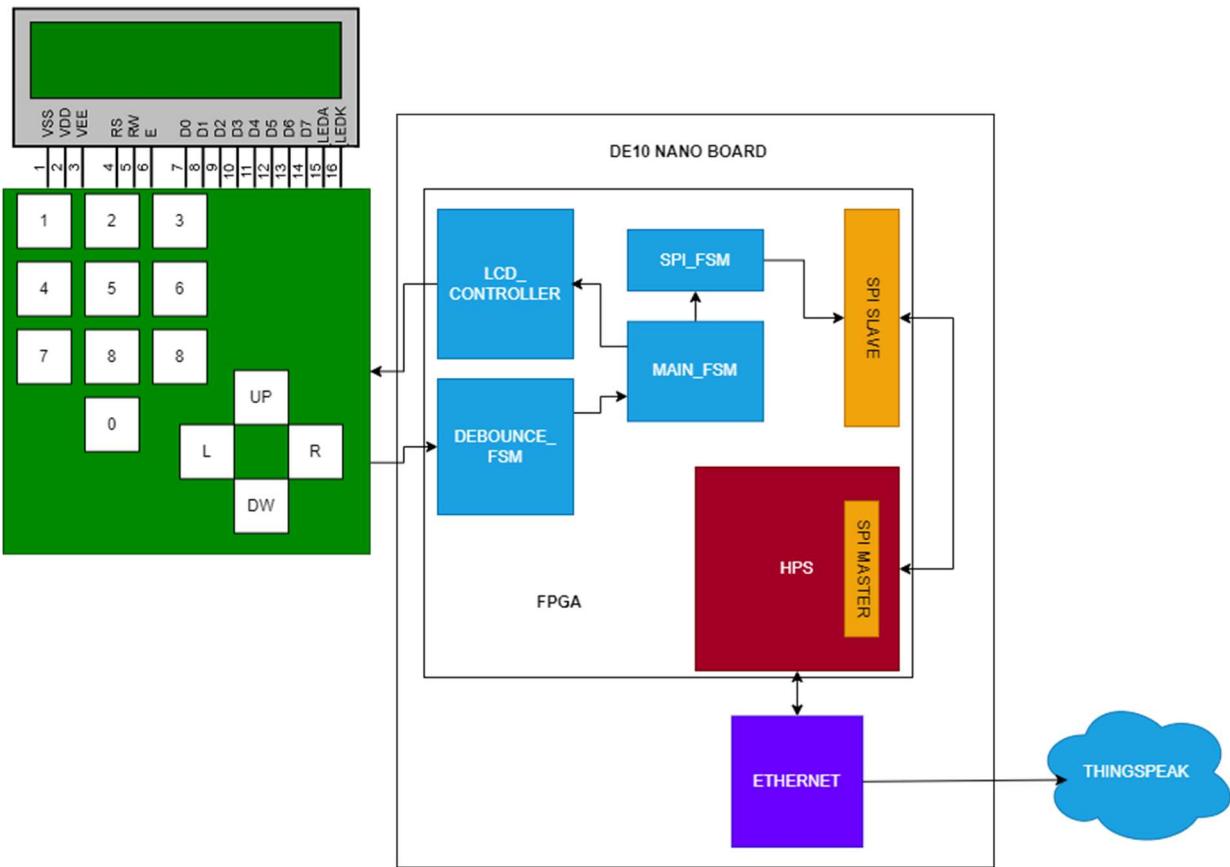


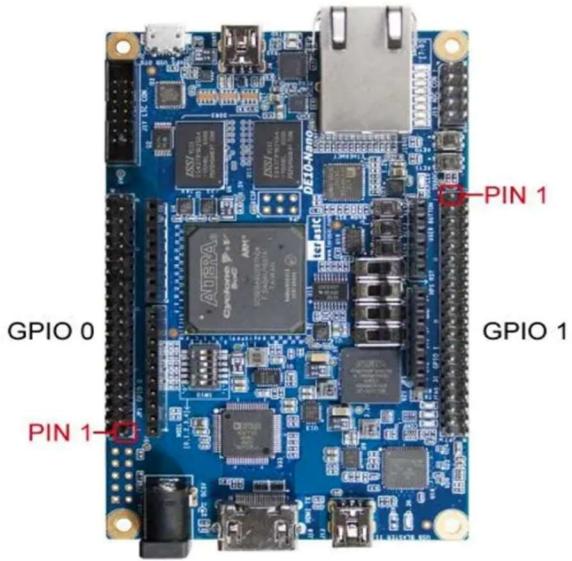
Figura 6 - Schema delle interconnessioni tra blocchi

Il sistema è composto dalla scheda DE10 nano e da una custom PCB. In dettaglio la prima ha al suo interno l'FPGA con l'HPS e il circuito per la gestione di Ethernet; la seconda invece alloggia un display LCD 16x2 e dei pulsanti numerici e di comando.

I segnali prodotti dai pulsanti vengono gestiti dal blocco "DEBOUNCE_FSM", il quale agisce da anti-rimbalzo e fornisce l'informazione di quale tasto sia stato premuto al blocco "MAIN_FSM". Questo blocco elabora i segnali in ingresso e produce i segnali necessari al blocco "LCD_CONTROLLER" per poter gestire la scrittura dei caratteri su LCD o la gestione dei comandi impartiti dall'utente. Inoltre, il blocco "MAIN_FSM" si occupa di inviare al blocco "SPI_FSM" l'informazione di quale tasto sia stato premuto, il quale gestisce la comunicazione SPI tra FPGA (slave) e HPS (master). L'informazione viene infine processata dall'HPS, il quale invia tramite il controller ethernet le richieste di scrittura al servizio IOT Thingspeak.

2.2. Assegnamento pin

Per una denominazione più contestualizzata nel progetto e una più immediata comprensione, i pin GPIO della DE10-Nano Cyclone 5 sono stato rimappati con i nomi riportati in Figura 7. La mappatura è inoltre documentata in un file di testo “Pin Assignmente DE10-NANO”, allegato nella cartella del progetto.



GPIO 0 (JP1)				GPIO 1 (JP7)			
RIGHT	GPIO_0[0]	1	GPIO_0[1]	PIN_E8	DB7	GPIO_1[0]	1
<u>UP</u>	GPIO_0[2]	3	GPIO_0[3]	PIN_D11	DB6	GPIO_1[2]	3
<u>BT9</u>	GPIO_0[4]	5	GPIO_0[5]	PIN_AH13	DB5	GPIO_1[4]	5
<u>BT6</u>	GPIO_0[6]	7	GPIO_0[7]	PIN_AH14	DB4	GPIO_1[6]	7
<u>BT3</u>	GPIO_0[8]	9	GPIO_0[9]	PIN_AH3	DB3	GPIO_1[8]	9
5V	5V	11	GND	Gnd	5V	11	11
<u>BT0</u>	GPIO_0[10]	13	GPIO_0[11]	PIN_AG14	DB2	GPIO_1[10]	13
PIN_AE23	GPIO_0[12]	15	GPIO_0[13]	PIN_AE6	PIN_AH24	GPIO_1[12]	15
PIN_AD23	GPIO_0[14]	17	GPIO_0[15]	PIN_AE24	PIN_AG23	GPIO_1[14]	17
PIN_D12	GPIO_0[16]	19	GPIO_0[17]	PIN_AD20	PIN_AG24	GPIO_1[16]	19
PIN_C12	GPIO_0[18]	21	GPIO_0[19]	PIN_AD17	PIN_AH21	GPIO_1[18]	21
PIN_AC23	GPIO_0[20]	23	GPIO_0[21]	PIN_AC22	BT1	GPIO_1[20]	23
PIN_Y19	GPIO_0[22]	25	GPIO_0[23]	PIN_AB23	BT2	GPIO_1[22]	25
PIN_AA19	GPIO_0[24]	27	GPIO_0[25]	PIN_W11	BT4	GPIO_1[24]	27
3V3	3.3V	29	GND	Gnd	3.3V	29	29
<u>DB1</u>	GPIO_0[26]	31	GPIO_0[27]	PIN_W14	BT5	GPIO_1[26]	31
<u>DB0</u>	GPIO_0[28]	33	GPIO_0[29]	PIN_Y17	BT7	GPIO_1[28]	33
<u>E</u>	GPIO_0[30]	35	GPIO_0[31]	SCK	BT8	GPIO_1[30]	35
<u>RW</u>	GPIO_0[32]	37	GPIO_0[33]	MOSI	LEFT	GPIO_1[32]	37
<u>RS</u>	GPIO_0[34]	39	GPIO_0[35]	MISO	DOWN	GPIO_1[34]	39

Figura 7 - DE10_Nano Cyclone V - Pin Assignment GPIO

2.3. Controller del tastierino numerico

L'utente può interagire con il sistema tramite un tastierino di 10 cifre, disposte su una Printed Circuit Board come indicato in Figura 8.

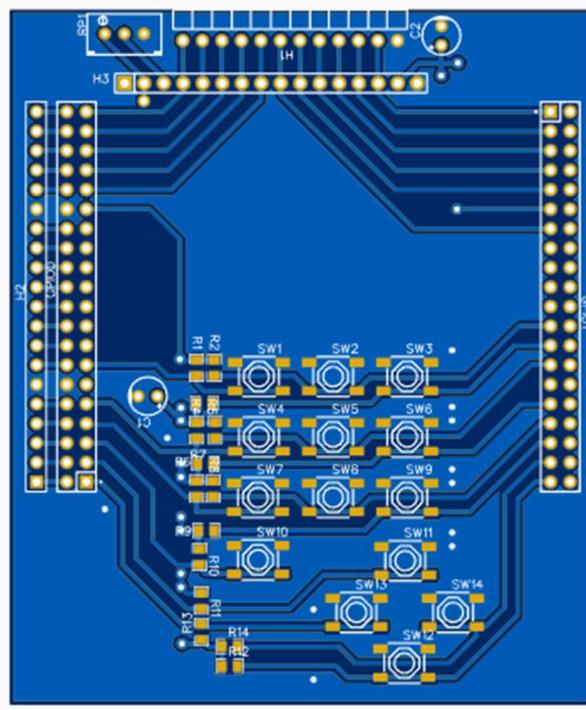


Figura 8 - Layout del PCB

Ogni pulsante fa da segnale di input che caratterizzerà il vettore “buttons_signals” di dimensione 14: i primi 10 rappresentano i dati e gli ultimi 4 rappresentano le istruzioni (up, down, left, right). Soltanto i primi 10 sono stampabili e saranno quindi inviati al cloud.

Il controller istanzia a sua volta un componente “debounce” che serve per garantire la stabilità dell’uscita: una volta premuto un pulsante, per 100 ms (pausa anti-rimbalzo) l’uscita rimarrà indipendente da ogni altra azione di un qualsiasi pulsante (come in Figura 9).

Il controller ha due uscite: il flag “output” segnala se un pulsante è stato premuto, l’uscita “btn_num” (32 bit) che indica con un numero intero da 0 a 9 quale pulsante è stato premuto.

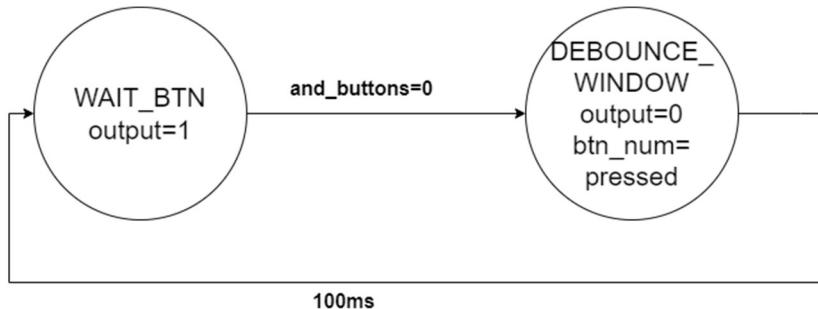


Figura 9 - Pausa anti-rimbalzo di 100ms dopo l'avvenuta pressione del pulsante

2.4. Controller del display

Il modulo “debounce_fsm” riceve in ingresso “lcd_bus” (10 bit) e “lcd_enable”, oltre agli ingressi ausiliari di clock e di reset. L’ingresso vettoriale a 10 bit contiene a sua volta:

- 1 bit “rs” per indicare se si tratta di istruzione (0) o di dato (1);
- 1 bit “rw” per indicare se l’operazione è di scrittura (0) o di lettura (1);
- 8 bit “lcd_data” che codificano il dato vero e proprio.

Le uscite del controller sono: “rs”, “rw” (lasciato sempre a 1), “e” (per aggiornare o meno il display coi nuovi dati) e lcd_data (8 bit) per mandare il dato al display. Gli 8 bit di “lcd_data” codificano un numero intero da 0 a 9 (formato ASCII, Figura 10), perciò i primi 4 bit codificano la colonna e rimangono costantemente a “0011”, mentre i 4 bit meno significativi codificano la riga e assumeranno valori da “0000” a “1001”.

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0 a P ^ F							- 9 E x p					
xxxx0001	(2)		!	1 A Q a q						.	A T 6 ä q					
xxxx0010	(3)		"	2 B R b r						「 4 W x F 0						
xxxx0011	(4)		#	3 C S c s						」 W T E e w						
xxxx0100	(5)		\$	4 D T d t						、 I T T M 2						
xxxx0101	(6)		%	5 E U e u						・ O T J C Ü						
xxxx0110	(7)		&	6 F U f v						ヲ K N E P S						
xxxx0111	(8)		'	7 G W g w						ア F X L g n						
xxxx1000	(1)		(8 H X h x						イ O N V J X						
xxxx1001	(2))	9 I Y i y						カ T J B - Y						

Figura 10 - Tabella ASCII dei caratteri di interesse

Il controller implementa una sua macchina a 4 stati (**Power_up**, **Initialize**, **Ready**, **Send**), come mostrato in Figura 11.

- Al segnale di “reset_n=0” riparte dallo stato **Power_up** con attesa di 50 ms.
- Nello stato **Initialize**, avviene la configurazione vera e propria del display:
 - ◆ numero delle linee del display, settato in modalità 2-lines;
 - ◆ numero di pixels per ogni carattere, settato con 5x8 “dots”;
 - ◆ operazione di “clear” dello schermo, impostata inizializzando il vettore “lcd_bus” con “00_0011_0000”;
 - ◆ entry mode, impostato su “increment”.

Tra le precedenti operazioni c’è un’attesa di $50\mu s \div 2ms$, per un totale di circa 7ms.

- Nello stato **Ready**, se il display è abilitato con “lcd_enable=1”, vengono inoltrati alle uscite rs, rw, lcd_data i rispettivi valori contenuti nel vettore lcd_bus.
- Lo stato **Send** temporizza l’invio dei dati al display, alternando 50 μs di semi-ciclo positivo (“e = 1”) con 50 μs di quello negativo (“e = 0”).

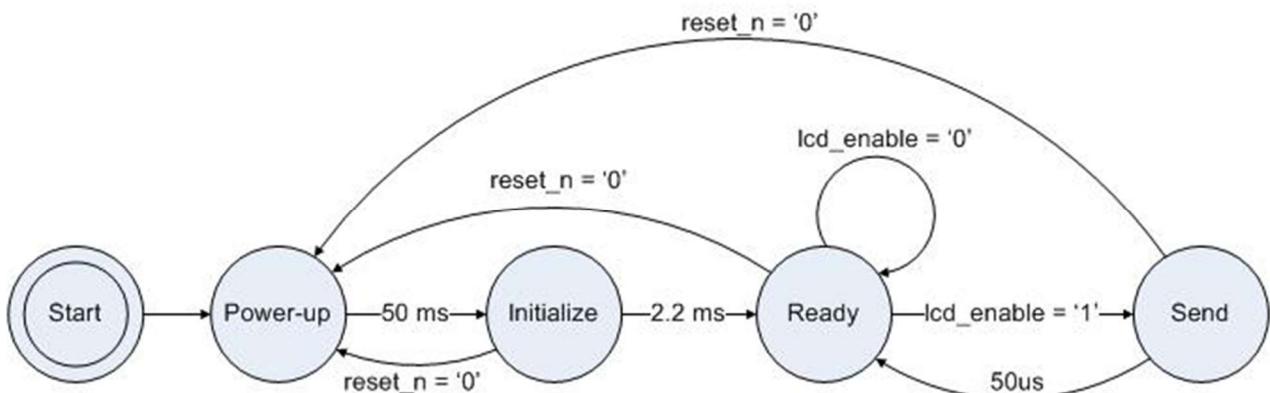


Figura 11 - Macchina a stati del controller LCD

2.5. SPI

Come già anticipato in precedenza, l’FPGA e l’HPS comunicano tramite SPI: il dispositivo principale si definisce Master (in questo caso è l’HPS) e decide se avviare una comunicazione e con chi. Gli altri dispositivi sono gli Slave e possono comunicare solo col Master, non tra di loro. Come mostrato in Figura 12, tutti i dispositivi hanno 4 linee:

- **SS** (Slave Select), per selezionare univocamente l’unità Slave con cui scambiare dati.

- **SCK**, che è il clock per sincronizzare la comunicazione. Ogni bit viene inviato/ricevuto al fronte di clock. In generale non coincide con la linea di clock del sistema.
- **MOSI** (Master Out-Slave In), per inviare dati allo Slave.
- **MISO** (Master In-Slave Out), per ricevere dati dallo Slave.

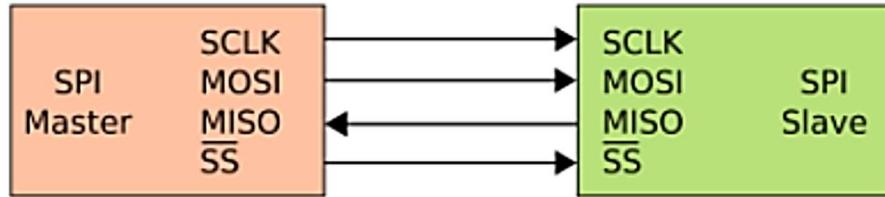


Figura 12 - Linee per la comunicazione SPI

In questo caso l'FPGA svolge il ruolo di unico Slave della comunicazione, quindi basta che il Master invii il segnale SCK (che fa da clock dell'SPI) e la sequenza di riconoscimento, lasciando inalterata la linea SS.

Quando non ci sono dati da inviare da Master a Slave, il clock viene impostato a zero; quindi, lo Slave non può inviare i dati. Qualora si volessero inviare dati (MISO) da FPGA a HPS, il Master deve necessariamente inviare dati (MOSI), anche se non rilevanti, come ad esempio una sequenza "F22F" esadecimale mostrata in Figura 13.



Figura 13 - sequenza "dummy" in uscita al Master, riportata dall'analizzatore logico

L'SPI dispone di 4 diverse modalità di funzionamento: nel progetto si è scelta la "Mode 1" (Figura 14), in cui dove il dato viene letto dal Master (MISO) sul fronte di salita e viene scritto dal Master (MOSI) sul fronte di discesa.

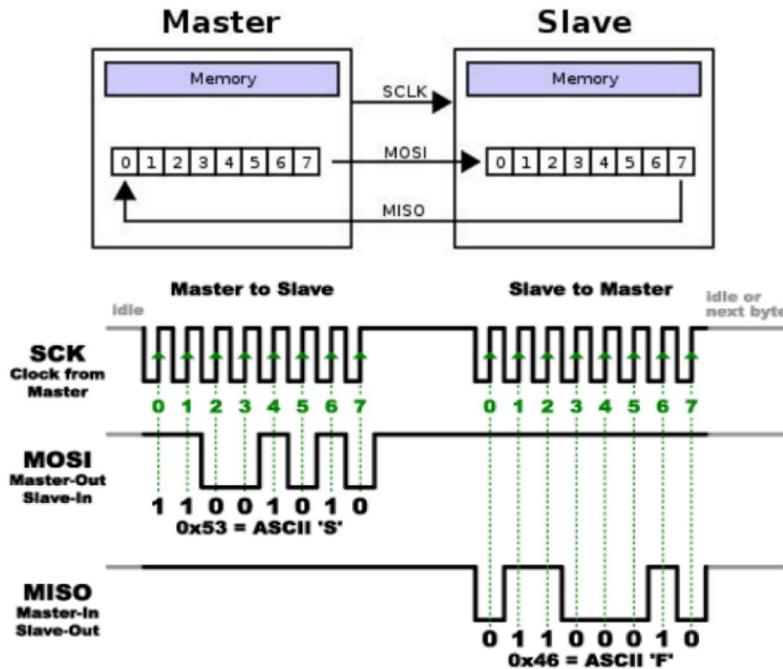


Figura 14 - Modalità 1 della comunicazione SPI

Dato che il clock dell'FPGA e il clock della comunicazione SPI viaggiano a due frequenze diverse, si rende necessaria una sincronizzazione tra i due.

Come rappresentato in Figura 15, si tratta di buffering tramite registro a 6 bit, di cui viene eseguito il controllo dei primi 2 bit e degli ultimi 2 bit:

- Se i primi due bit sono “00” e gli ultimi due sono “11”, allora è avvenuto un fronte di salita;
- Se, viceversa, i primi due bit sono “11” e gli ultimi due sono “00”, allora si tratta di un fronte di discesa.

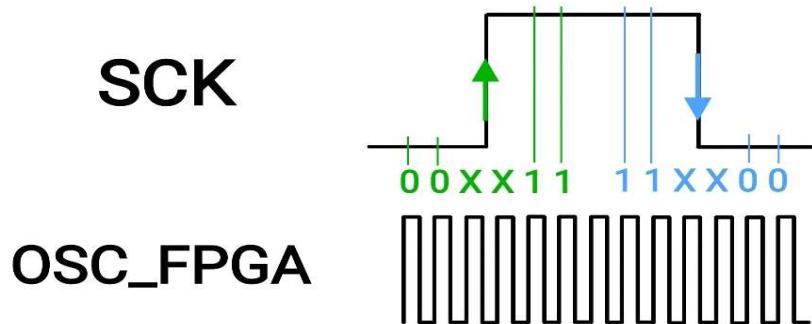


Figura 15 - Concetto di sincronizzazione a 6 bit

2.6. Macchina a stati dell'SPI

La comunicazione SPI è gestita con una macchina a due stati: ***wait_rqst***, ***send***. Una volta istanziata l'entità “*spi*”, si entra nel process in cui si aggiorna lo stato ad ogni colpo di clock (in caso di pulsante reset premuto si forza lo stato a *wait_rqst*).

Lo stato **wait_rqst** si occupa di cosa assegnare alla variabile “current_msg” basandosi sulla richiesta della comunicazione spi (condizione gestita dall’apposito flag, Figura 16):

- se l’FPGA non richiede di inviare dati all’HPS il dato MISO dovrà contenere una sequenza dummy (in questo caso si imposta “F22F” esadecimale);
- se, viceversa, l’FPGA attiva la richiesta di comunicazione allora il dato MISO da inviare conterrà il numero (da 0 a 9) del pulsante digitato.

Nello stato **send** viene passato sulla linea MISO il contenuto della variabile “current_msg”: l’assegnamento avviene quando sono disponibili tutti e 16 i bit (condizione verificata tramite flag “scrivi”).

Dall’HPS all’FPGA viene comunque mandato indietro un dato sulla linea MOSI (nel nostro è sempre lo stesso dato dummy perché non vogliamo leggere nulla dall’HPS) ogni volta che c’è una comunicazione tra le due parti, oltre al segnale SCK.

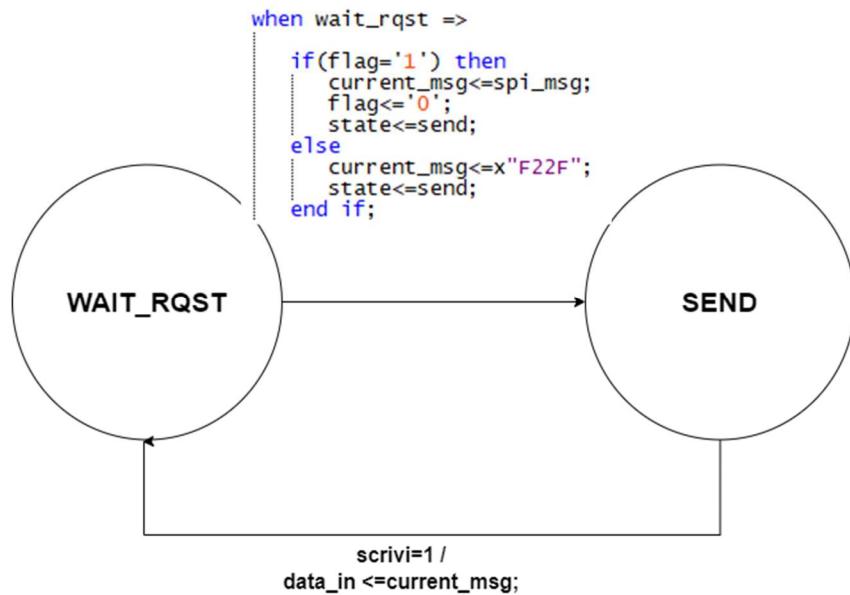


Figura 16 - Macchina a stati che gestisce l’SPI

2.7. Main_fsm

In questa macchina a 8 stati sostanzialmente avviene la distinzione del contenuto di “lcd_bus” e, a seconda che sia un dato un dato oppure un comando, si aggiorna o meno il contenuto di “spi_msg”. Come si può notare in Figura 17, soltanto i dati (caratteri effettivamente stampabili) portano alla condizione di aggiornare il messaggio, che poi sarà l’input della macchina *spi_fsm* e verrà mandato dall’FPGA all’HPS.

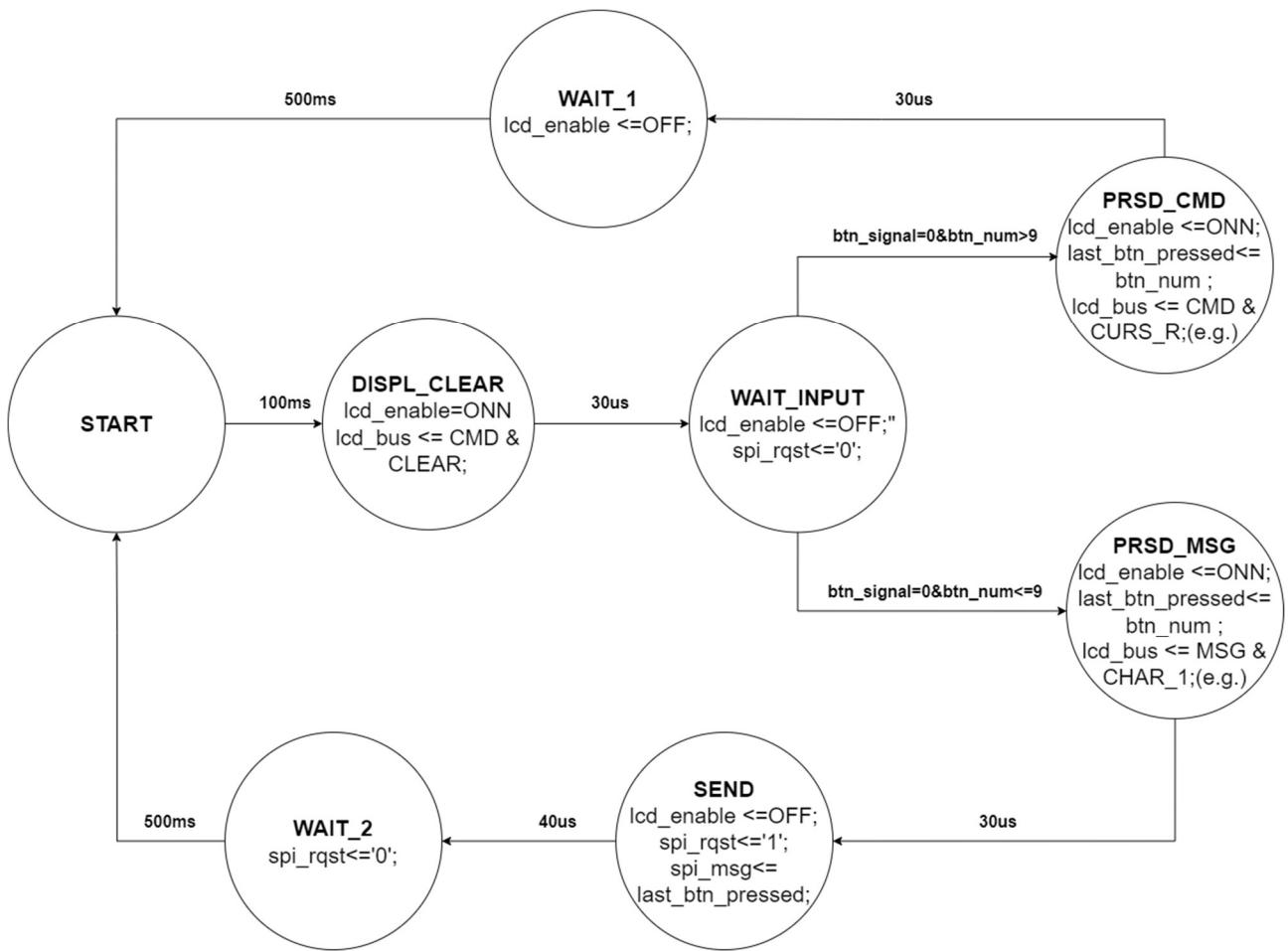


Figura 17 - Diagramma della macchina a stati "main_fsm"

2.8. Visione complessiva dei VHDL

L'insieme dei moduli VHDL descritti in precedenza è strutturato come in Figura 18, ottenuta grazie al tool “RTL Viewer” presente in Quartus.

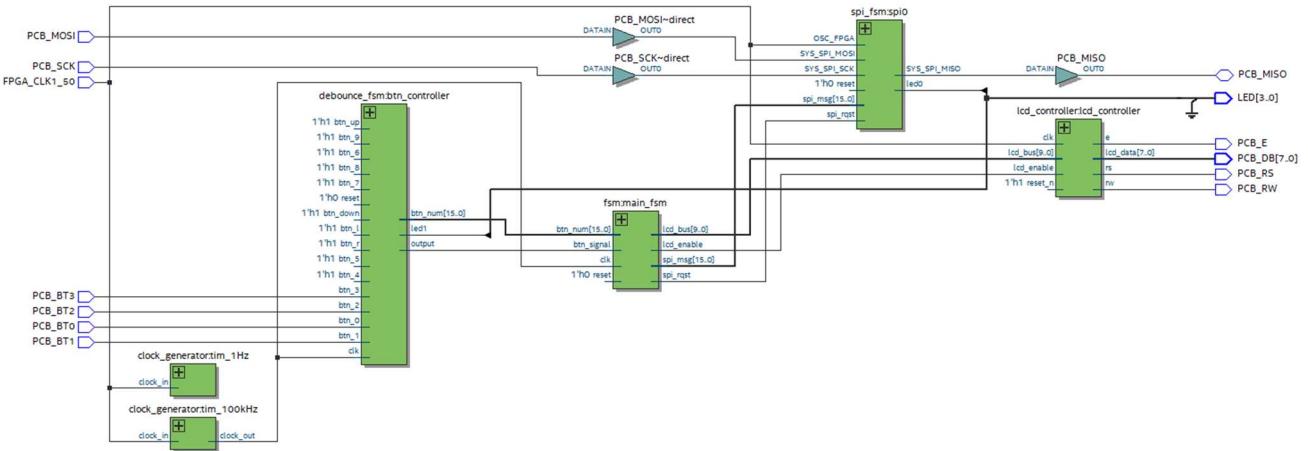


Figura 18 - Struttura complessiva dei moduli, ottenuta con RTL viewer

3. Simulazione con ModelSim

3.1. Testbench complessivo del sistema

Il sistema viene simulato con Modelsim, in modo da verificare la funzionalità dei vari blocchi che lo compongono. In Figura 19 si riporta la versione finale del testbench, che include già i segnali di tutti i moduli coinvolti, tra cui:

1. Ingressi di *lcd_controller* ;
2. Uscite di *lcd_controller* ;
3. Uscite dei moduli che generano la versione modificata del clock (*clock_generator*);
4. Uscite del controller dei pulsanti;
5. Ingressi del controller dei pulsanti;
6. Uscite del *main_fsm* che riguardano la gestione dell'spi.

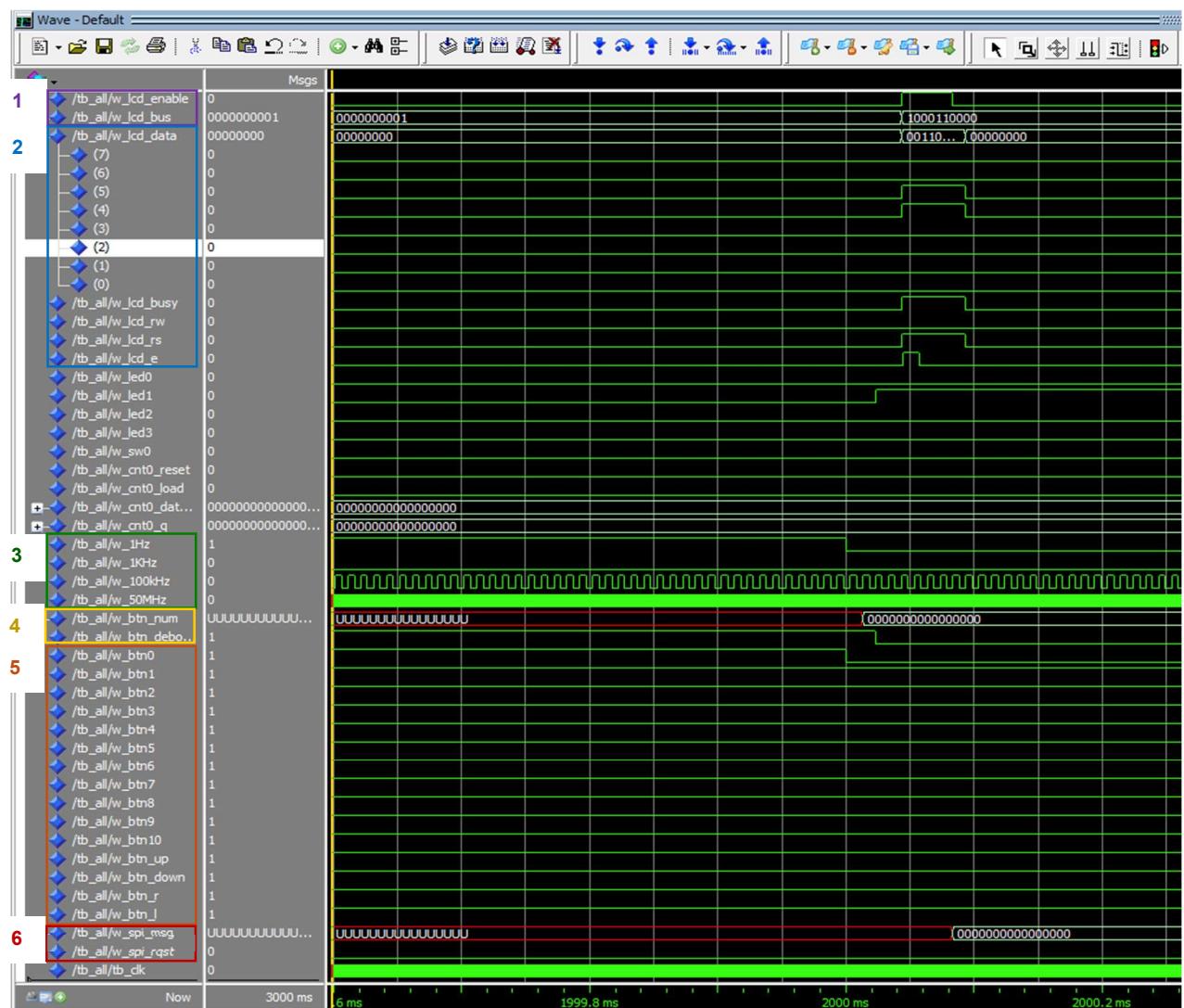


Figura 19 - Forme d'onda del testbench implementato in Modelsim

4. Programmazione HPS con ThingSpeak

4.1. Firmware in C

La comunicazione lato Master è gestita dal programma “main_Thingspeak”, che si appoggia alla funzione `SPIM_WriteReadTxRxData16()` di Terasic per comunicare a 16 bit con l’SPI.

Il programma richiama la libreria `curl/curl.h` (che sopporta il protocollo https) per il trasferimento degli URL lato Thingspeak.

In Figura 20 è mostrata in particolare la parte di codice che si occupa della trasmissione dati utilizzando la funzione `SPIM_WriteReadTxRxData16()` per assegnare il valore ricavato dall’FPGA in una variabile “temp” (che sta sull’HPS). Inoltre, nello stesso ciclo while il contenuto di “temp” viene inoltrato direttamente sul primo field della piattaforma ThingSpeak e sarà quindi visualizzabile sull’interfaccia del canale.

```
66 //check if SLAVE has sent back F22F or the BUTTON DATA
67 if(received == 0x0000 ||received == 0x0001||received == 0x0002||received == 0x0003||received == 0x0004||
68 >     switch(received){//conversion from 16 bit to int, in base of the API %d...
101     // url per la scrittura di dati del canale thingspeak
102     sprintf(bufferurl, "https://api.thingspeak.com/update?api_key=OP8H82Y91IYI1FYB&field1=%d",to_send);
103     dl_curl_easy_setopt(curl, CURLOPT_URL, bufferurl);
104     res = dl_curl_easy_perform(curl);
105
106     // Verifica se la richiesta è stata eseguita con successo
107     if (res != CURLE_OK) {
108         fprintf(stderr, "curl_easy_perform() failed: %s\n",
109                 dl_curl_easy_strerror(res));
110     } else {
111         printf("\r");
112         fflush(stdout);
113     }
114     to_send=15;// setting a not pressed char to not reenter the loop
115
116     sleep(15); // time interval to refresh ThingSpeak
117 }
118 sleep(0.1); //time interval between 2 data send of MOSI
119 }
120
121 dl_curl_easy_cleanup(curl);
122 curl_unload(lib_handle);
123
124 SPIHW_Close();
```

Figura 20 - Spezzone di codice C che si occupa della trasmissione / ricezione dei dati su 16 bit

Al di fuori del ciclo while si imposta tramite funzione “sleep” l’intervallo di tempo da attendere prima di aggiornare il dato seriale che l’HPS dovrà leggere: in questo caso, dopo che si è premuto un pulsante apparirà un numero nella stampa seriale e vi rimarrà per 15 s, dopodiché la seriale verrà aggiornata col nuovo valore (ad esempio la sequenza dummy se non viene più premuto alcun tasto) e la temporizzazione torna ad essere di 100 ms.

4.2. Cross-compilazione

Si procede compilando il programma per il sistema operativo della DE10-Nano. Il compilatore si trova sulla macchina virtuale Linux (che fa da Host) e il sistema operativo da compilare è sulla DE10_Nano (che fa da Target).

Il compiler e il linker stanno su un sistema operativo diverso da quello della DE10-Nano, perciò l'operazione si chiama cross-compilazione. La macchina che esegue il cross-compilatore è detta host, mentre quella su cui si eseguirà il file ottenuto dal processo è detta target.

Per la compilazione serve il makefile, che si trova nella directory contenente i file sorgente con estensione .h (header) e .c (codice); il makefile contiene la variabile "target", che specifica il nome del file di output generato, e il compilatore (qui usiamo l'arm-gcc cross-compiler).

Una volta impostata la directory, per eseguire il makefile basta scrivere "make" sulla riga di comando di Linux (Figura 21). Se non ci sono errori durante la compilazione, si genera un file di output con il nome specificato dalla variabile "target".

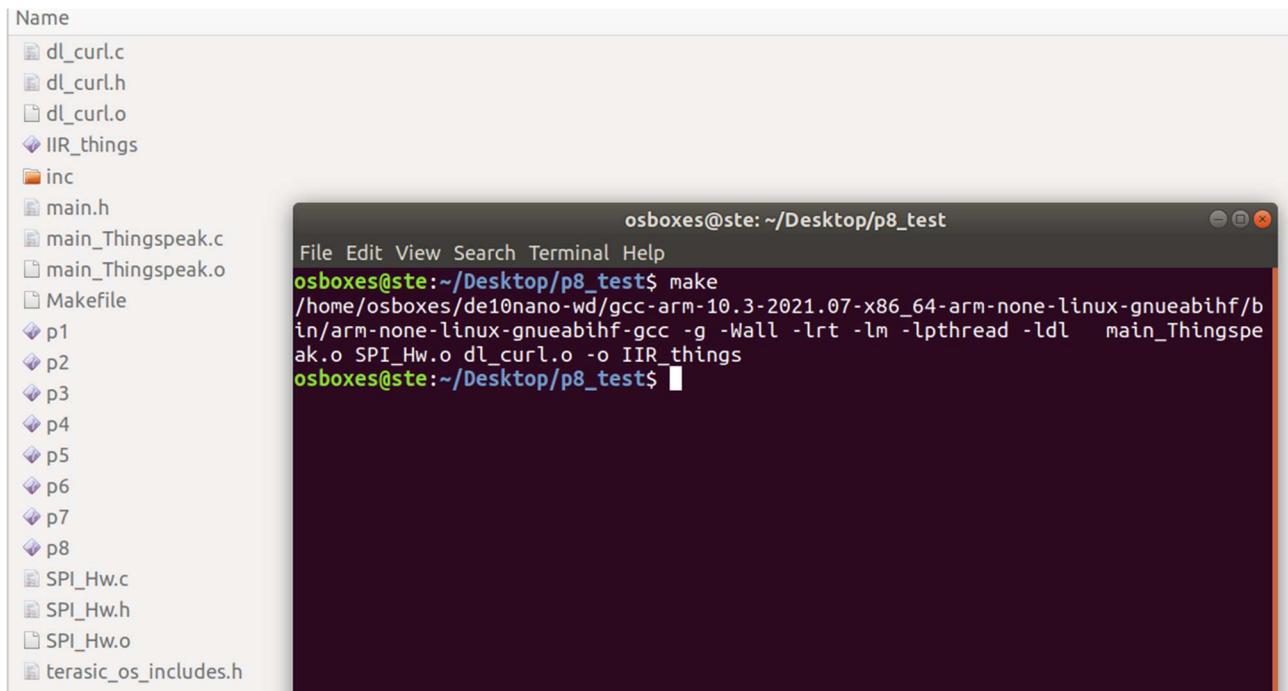


Figura 21 - Esecuzione del makefile

4.3. Connessione dell'HPS

Terminata questa fase, si recupera l'indirizzo IP assegnato alla FPGA. Per farlo, si avvia il software MobaXterm e si collega la scheda al PC tramite cavo Ethernet e mini-USB. Si crea quindi una connessione seriale con baud rate impostato a 115200 bps, specificando la porta USB a cui è connessa la scheda (in questo caso è la COM6). Successivamente, eseguendo

il comando *ifconfig*, verrà visualizzato l'indirizzo IP: in questo caso è 192.168.1.122, come si osserva in Figura 22.

```
root@socfpga:~# ifconfig
eth0      Link encap:Ethernet HWaddr 1a:9f:2b:bb:6a:33
          inet addr:192.168.1.122 Bcast:192.168.1.255 Mask:255.255.255.0
          inet6 addr: fe80::189f:2bff:febb:6a33/64 Scope:Link
             UP BROADCAST RUNNING MULTICAST DYNAMIC MTU:1500 Metric:1
             RX packets:2345 errors:0 dropped:0 overruns:0 frame:0
             TX packets:497 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1000
             RX bytes:211916 (206.9 KiB) TX bytes:95345 (93.1 KiB)
             Interrupt:34 Base address:0x8000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
             UP LOOPBACK RUNNING MTU:65536 Metric:1
             RX packets:172 errors:0 dropped:0 overruns:0 frame:0
             TX packets:172 errors:0 dropped:0 overruns:0 carrier:0
             collisions:0 txqueuelen:1
             RX bytes:13180 (12.8 KiB) TX bytes:13180 (12.8 KiB)
```

Figura 22 - Ottenimento indirizzo IP

Una volta ottenuto l'indirizzo, è possibile stabilire una connessione SSH compilando i campi richiesti:

- Remote host: Indirizzo IP della FPGA
- Username: root

Dopo l'accesso, si verrà reindirizzati alla directory /home/root: qui si carica il file di output generato dal makefile, assicurandosi di assegnargli i permessi di esecuzione (Figura 23).

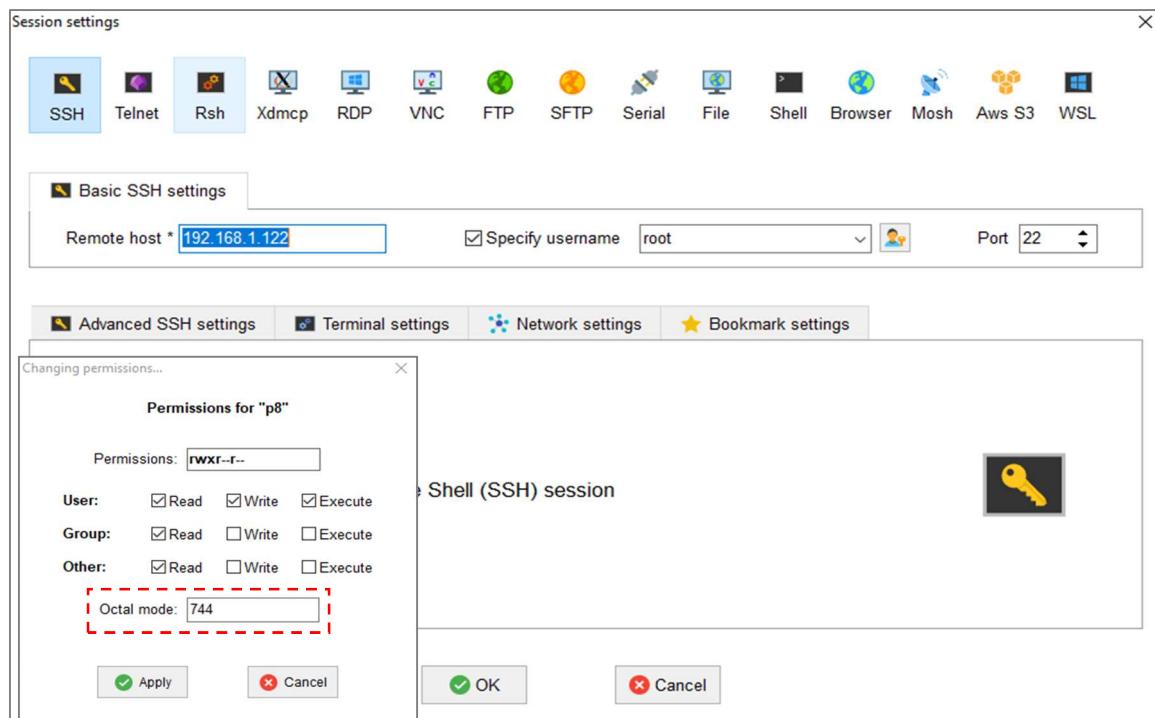
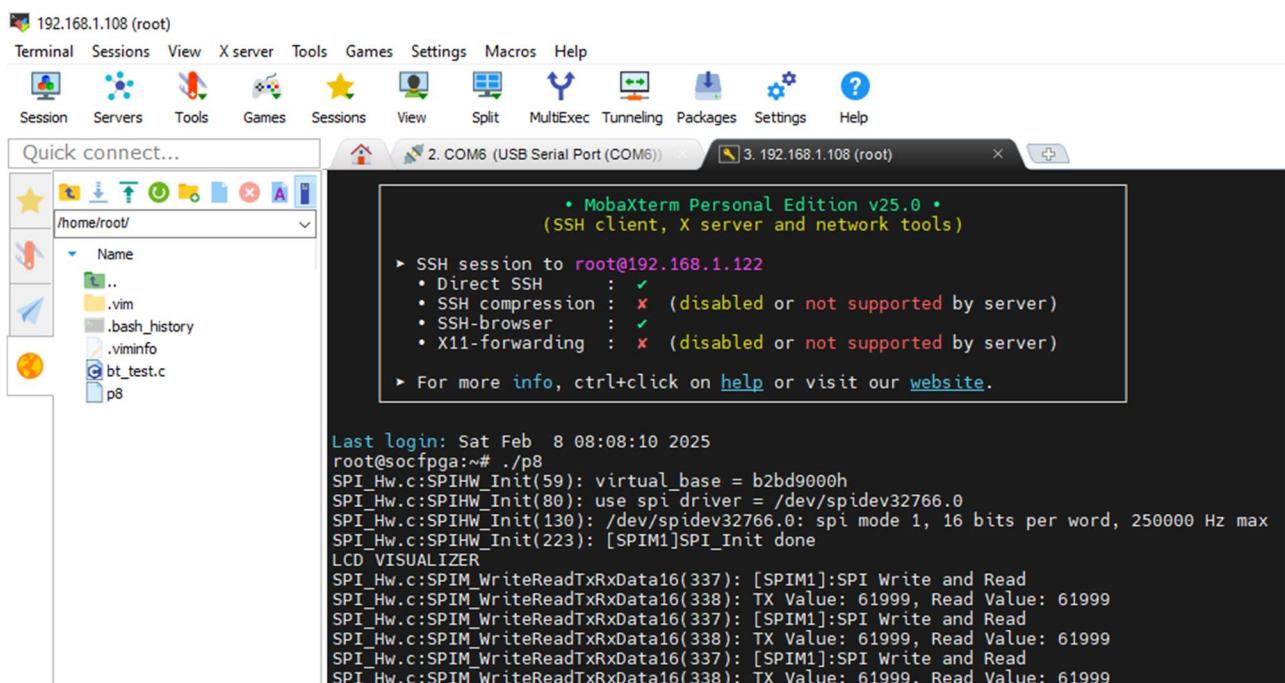


Figura 23 - Connessione SSH, directory e impostazione permessi

4.4. Test della comunicazione SPI

Una volta avviata la sessione SSH si può testare il funzionamento del programma tramite il comando `./NomeFile`: in questo caso si esegue il file `./p8()` che avvia semplicemente una comunicazione tra Master e Slave.

Terminata l'inizializzazione (“*SPI_Init done*”), il programma prosegue ciclicamente con l'aggiornamento di due variabili: TX Value e Read Value. La prima indica la sequenza dummy (corrispondente a “F22F” esadecimale) che il Master invia allo Slave, mentre la seconda indica il dato che arriva al Master: si chiude semplicemente un loopback in cui al Master viene tornato lo stesso valore che ha mandato allo Slave.



```
• MobaXterm Personal Edition v25.0 •
(SSH client, X server and network tools)

> SSH session to root@192.168.1.122
  • Direct SSH : ✓
  • SSH compression : ✘ (disabled or not supported by server)
  • SSH-browser : ✓
  • X11-forwarding : ✘ (disabled or not supported by server)

> For more info, ctrl+click on help or visit our website.

Last login: Sat Feb 8 08:08:10 2025
root@socfpga:~# ./p8
SPI_Hw.c:SPIHW_Init(59): virtual_base = b2bd9000h
SPI_Hw.c:SPIHW_Init(80): use spi driver = /dev/spidev32766.0
SPI_Hw.c:SPIHW_Init(130): /dev/spidev32766.0: spi mode 1, 16 bits per word, 250000 Hz max
SPI_Hw.c:SPIHW_Init(223): [SPIM1]SPI_Init done
LCD VISUALIZER
SPI_Hw.c:SPIM_WriteReadTxRxData16(337): [SPIM1]:SPI Write and Read
SPI_Hw.c:SPIM_WriteReadTxRxData16(338): TX Value: 61999, Read Value: 61999
SPI_Hw.c:SPIM_WriteReadTxRxData16(337): [SPIM1]:SPI Write and Read
SPI_Hw.c:SPIM_WriteReadTxRxData16(338): TX Value: 61999, Read Value: 61999
SPI_Hw.c:SPIM_WriteReadTxRxData16(337): [SPIM1]:SPI Write and Read
SPI_Hw.c:SPIM_WriteReadTxRxData16(338): TX Value: 61999, Read Value: 61999
```

Figura 24 - Test del programma “p8”, visualizzando il transito dei dati tramite connessione SSH

4.5. ThingSpeak

ThingSpeak è una piattaforma IoT in cloud per l'aggregazione e visualizzazione di dati provenienti da varie fonti in tempo reale. La piattaforma supporta nativamente l'elaborazione mediante script Matlab online o tramite applicazione.

ThingSpeak è suddiviso in canali, ognuno dei quali possiede le proprie API Key, cioè due stringhe alfanumeriche che permettono la scrittura e lettura dei vari campi all'interno del canale. Dentro ogni canale si creano due “field”, cioè i campi che contengono i dati.

In Figura 25 è riportato un esempio di field, denominato come “Number of button pressed”, il cui aggiornamento ogni 15 secondi e il suo contenuto è un numero intero che indica il pulsante premuto sul tastierino.

Number of button pressed Options

Name	Number of button pressec
Field	Field 1
Update Interval	15 second(s)
Units	Enter Measurement Units
Data Type	<input checked="" type="radio"/> Integer <input type="radio"/> Decimal <input type="radio"/> 1 (# of places)

Save
Cancel

Figura 25 - Impostazione del field di ThingSpeak

In seguito alla pressione del pulsante, sull'interfaccia del field “Number of button pressed” appare effettivamente il numero intero del pulsante digitato, ad esempio il “2” in Figura 26.

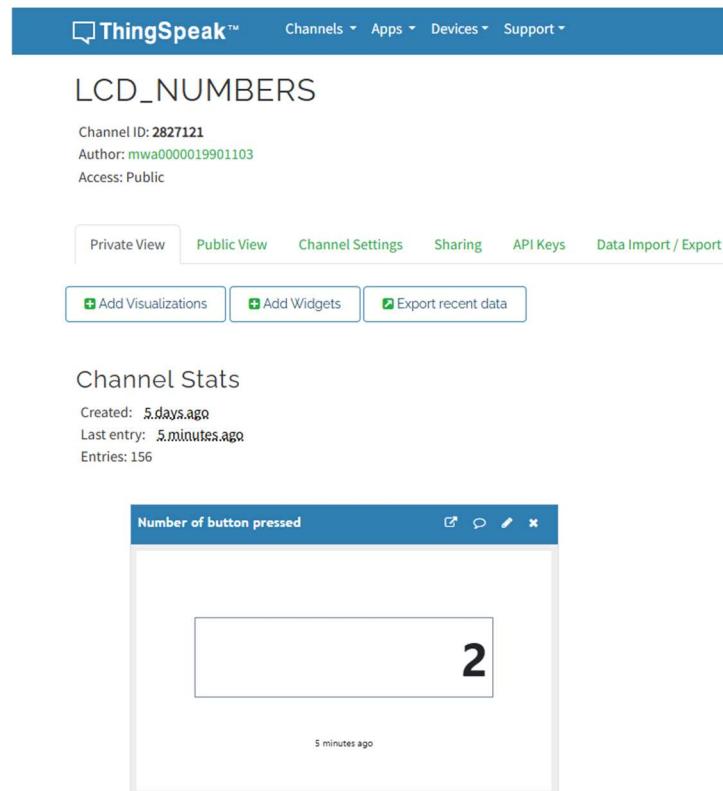


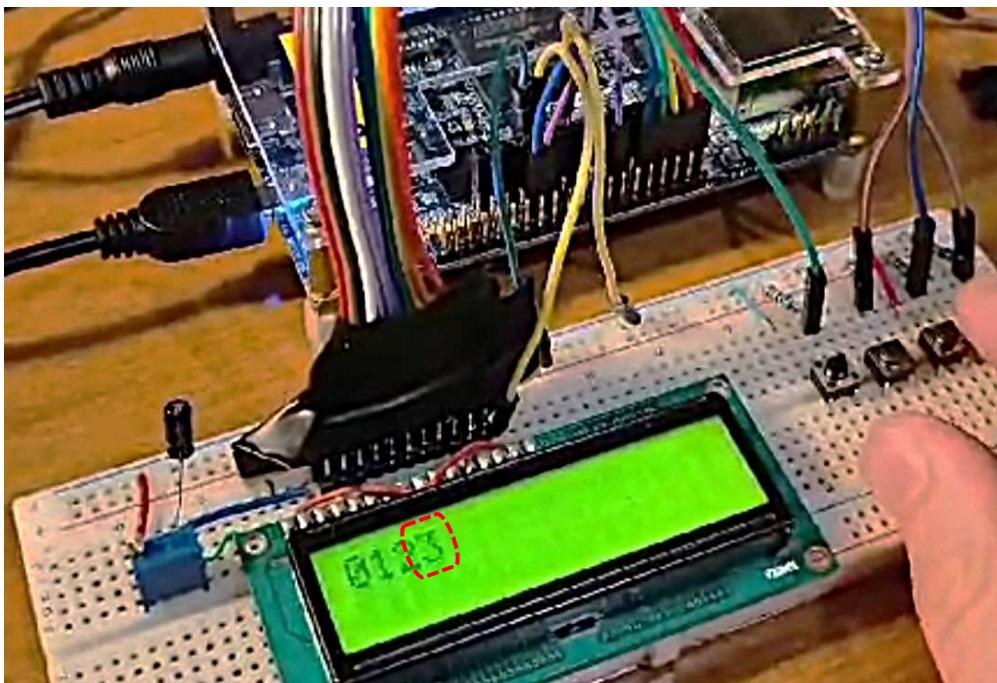
Figura 26 - Numero intero arrivato a destinazione sul cloud

5. Risultati

5.1. Specifiche rispettate

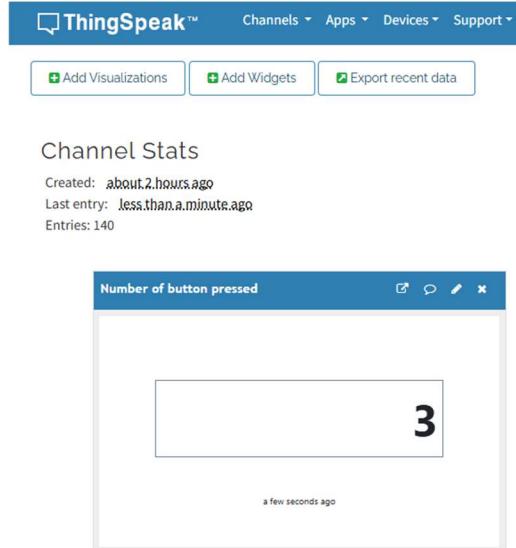
In conclusione, il progetto ha rispettato gli obiettivi posti all'inizio: premendo per esempio il numero “3” del tastierino si riportano i vari passi in ordine cronologico:

1. Il pulsante digitato sul tastierino viene rilevato e graficato sul display LCD 16x2



2. Il dato digitato viene correttamente inviato da FPGA a HPS sulla linea MISO, mentre sulla linea MOSI si avrà solamente il dummy (perché non serve leggere dal Master)

3. Premuto il pulsante, il dato viene spedito sul cloud e visualizzato correttamente sull'apposita interfaccia di ThingSpeak



5.2. Sviluppi futuri

Una possibile continuazione del progetto consiste nell'inserimento dei 4 pulsanti di comando (su, giù, destra, sinistra) che sono già fisicamente saldati sul PCB e la conseguente integrazione di questi comandi nella macchina a stati “*debounce_fsm*”. Inoltre, così si completerebbe anche l'altro branch della macchina a stati “*main_fsm*”, che gestisce il caso in cui il pulsante premuto non sia un numero da 0 a 9 (Figura 27).

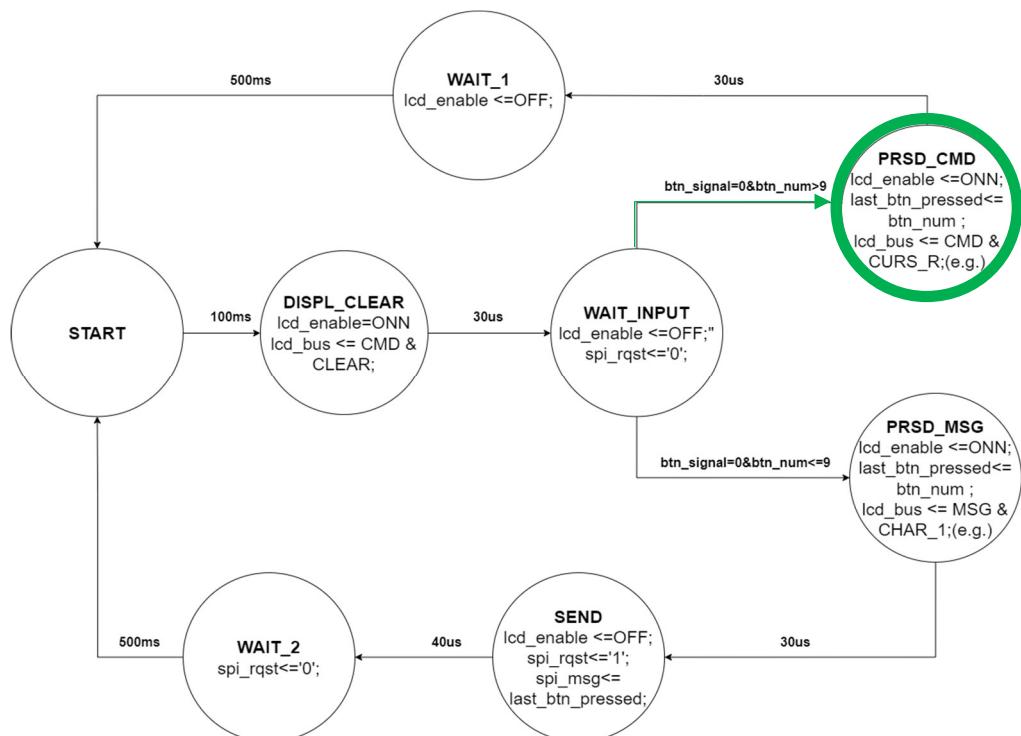


Figura 27 - Completamento del ramo "pulsanti CMD"

Avendo a disposizione 4 nuovi segnali di input è possibile assegnare nuovi comandi al controller dell'LCD: come si nota dall'inizializzazione del modulo “*fsm.vhd* ” in Figura 28, è possibile attivare anche altri comandi (es: lo shift del cursore), permettendo la sovrascrittura sulla stessa casella oppure la scrittura su una casella in generale, impostata tramite i pulsanti.

```
--https://www.electronicwings.com/sensors-modules/lcd-16x2-display-module

--commands
constant CLEAR:           charcode := x"01";    --Clear the display screen  1.64ms
constant CURS_SHR:         charcode := x"06";    --Shift the cursor right (e.g. data gets written in an inc
constant DISP_ON_CURS_OFF: charcode := x"0C";    --Display on, cursor off   40 us
constant DISP_ON_CURS_BLINK: charcode := x"0E";   --Display on, cursor blinking  40 us
constant CURS_HOME_1:      charcode := x"80";    --Force the cursor to the beginning of the 1st line 40 us
constant CURS_HOME_2:      charcode := x"C0";    --Force the cursor to the beginning of the 2nd line 40 us
constant CURS_L:            charcode := x"10";    --Shift cursor position to the left 40 us
constant CURS_R:            charcode := x"14";    --Shift cursor position to the right  40 us
constant DISP_L:             charcode := x"18";   --Shift entire display to the left  40 us
constant DISP_R:             charcode := x"1C";   --Shift entire display to the right 40 us

--constant xxx:  charcode := x"38";    --2 lines, 5x8 matrix, 8-bit mode  40 us
--constant xxx:  charcode := x"28";    --2 lines, 5x8 matrix,4-bit mode   40 us
--constant xxx:  charcode := x"30";    --1 line, 8-bit mode   40us
--constant xxx:  charcode := x"20";    --1 line, 4-bit mode   40us
```

Figura 28 - Comandi per l'inizializzazione dell'LCD 16x2