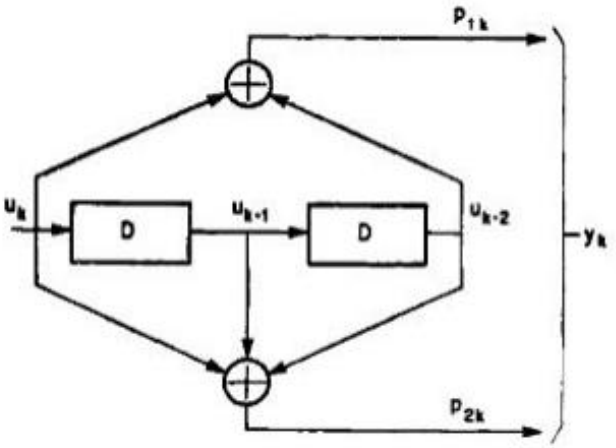
Prova finale di reti logiche
a.a. 2021 / 2022

A cura di Diego Corna e Filippo Corna

1 INTRODUZIONE

Il progetto consiste nella realizzazione di un modulo HW descritto in VHDL, che si interfacci con la memoria secondo la seguente specifica.

In modulo riceve un flusso continuo di parole da 8 bit e restituisce in uscita una sequenza continua di parole ognuna da 8 bit, codificate in questo modo: ogni parola di ingresso viene serializzata, producendo un flusso continuo da 1 bit, a cui viene applicato il codice convoluzionale ½ che codifica ogni bit con 2 bit, secondo lo schema in figura. Così viene generato un flusso continuo Y.



Codificatore convoluzionale con tasso di trasmissione $\frac{1}{2}$.

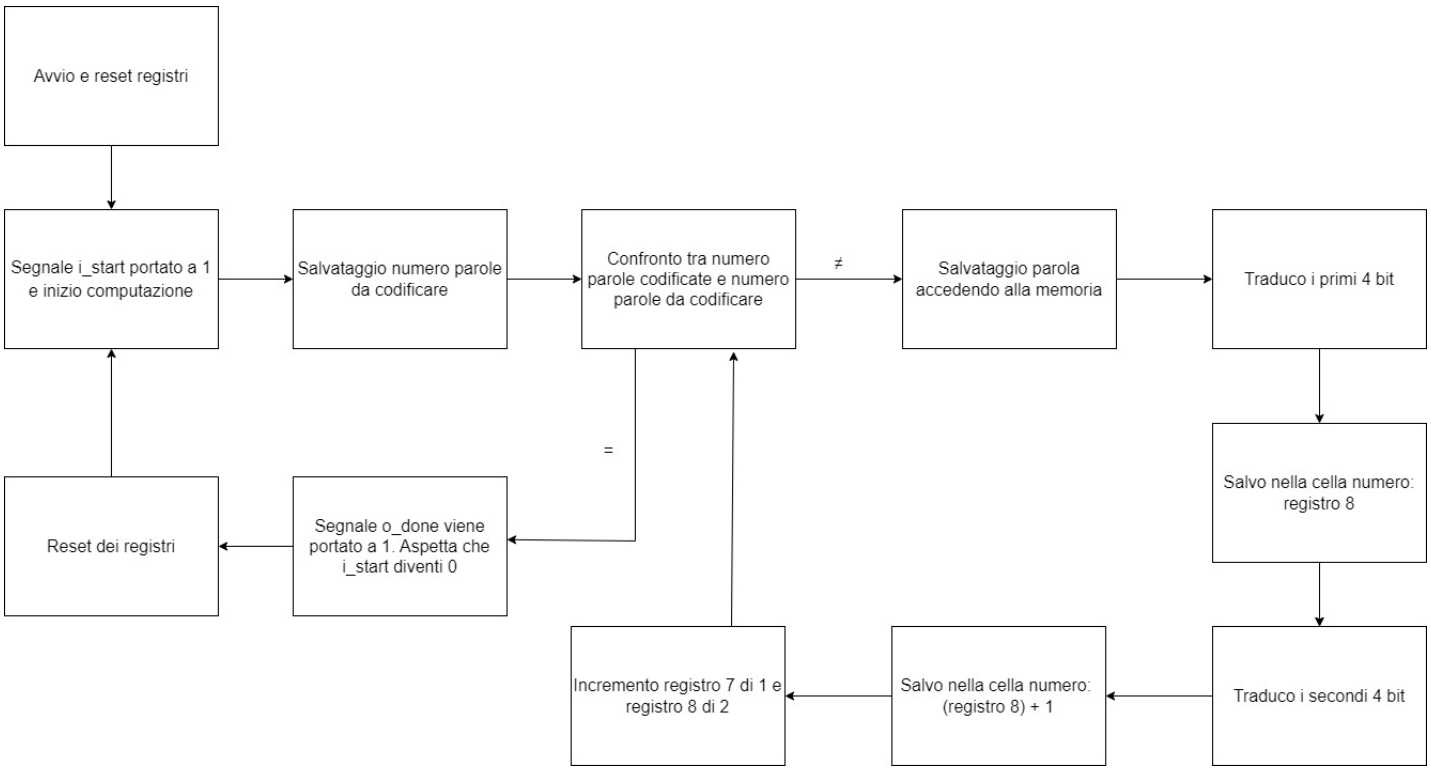
Secondo la notazione in figura, il flusso Y è ottenuto come concatenamento alternato dei due bit di uscita. Il bit Uk genera i bit p1k e p2k, che sono concatenati per generare il flusso yk da 1 bit. La sequenza finale di uscita è la parallelizzazione, su 8 bit, del flusso continuo yk.

Il modulo da implementare deve leggere la sequenza da codificare da una memoria con indirizzamento al Byte in cui è memorizzato.
Il numero di parole da codificare è memorizzato nell'indirizzo 0 e parole del flusso di ingresso vengono salvate a partire della cella 1 di memoria.
Il flusso di uscita deve essere memorizzato a partire dall'indirizzo 1000 e la dimensione massima della sequenza di ingresso è 255 byte.

L'elaborazione partirà quando un segnale di ingresso START viene portato a 1. Questo segnale rimarrà alto finche il segnale di DONE non verrà portato alto, al termine della computazione. DONE deve rimanere a 1 finché START non è riportato a 0. START non può essere riportato alto finché DONE non torna a 0 e eventualmente si riparte con la codifica. Si possono quindi codificare più flussi uno in seguito all'altro senza dover

“resettare” il modulo con il segnale RESET. Il modulo deve essere progettato considerando che prima della prima codifica verrà
Il RESET viene sempre dato al modulo, mentre in caso di una seconda elaborazione, non serve aspettare il RESET, ma semplicemente la terminazione dell’elaborazione.

2 SCHEMA FUNZIONAMENTO

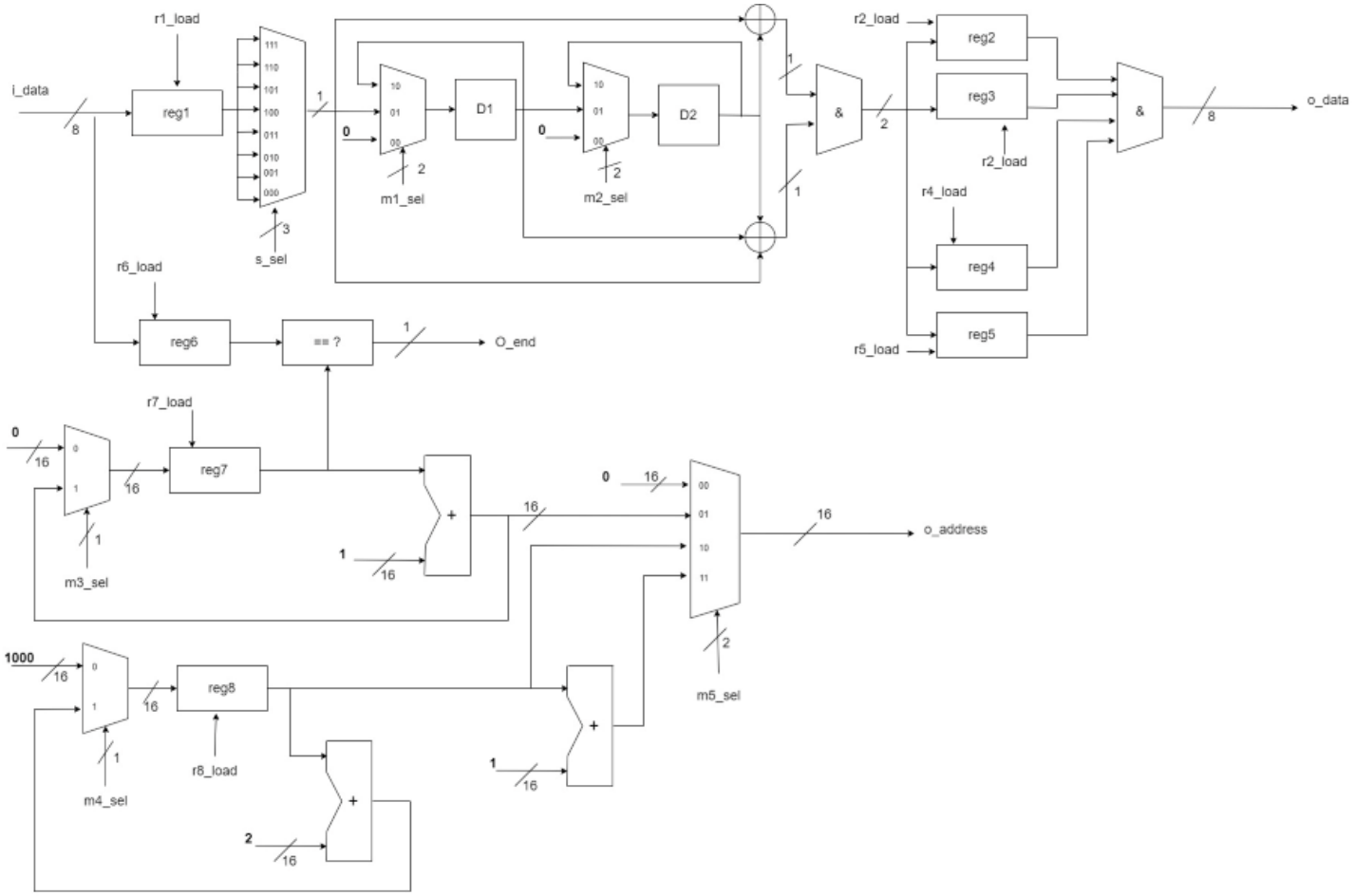


La traduzione della parola avviene in questo modo:

Si fa passare un bit alla volta e vengono generati due bit nei segnali PK1 e PK2. La concatenazione di questi segnali viene poi salvata nei registri 2, 3 ,4, 5; il primo bit della parola da codificare genererà 2 bit che sono salvati nel registro 2, il secondo bit genererà 2 bit che saranno salvati nel registro 3 e così via. Successivamente, la concatenazione di queste quattro coppie di bit comporrà una parola di 8 bit. Questa procedura viene ripetuta con i secondi 4 bit della parola da codificare e si otterrà un’altra parola da 8 bit.

3 ARCHITETTURA

3.1 DATAPATH



Reg1: Registro dove si salva la parola da codificare in 8 bit, pilotato dal segnale r1_load. E' collegato ad un MUX m_sel che seleziona singolarmente i bit della parola che poi verranno elaborati secondo la specifica.

Msel: MUX che seleziona attraverso il segnale m_sel (3 bit) i singoli bit da codificare della parola registrata in r1.

Reg2 – Reg5: Registri in cui vengono salvate le coppie di bit codificate secondo la specifica, che vengono in seguito concatenate per comporre gli 8 bit di o_data.

Vengono elaborati inizialmente i primi 4 bit della parola (codificati uno alla volta e la coppia di bit risultante viene salvata nel registro corretto) e poi gli ultimi 4 (analogamente) e si ottiene una parola codificata in 16 bit divisa in due parti da 8 bit, salvate a partire dall'indirizzo di memoria 1000.

Reg6: Registro dove si salva il numero di parole da computare, pilotato dal segnale r6_load. E' collegato a un comparatore che lo confronta con il contenuto del reg7.

Reg7: Registro in cui salvo il numero di parole computate, pilotato dal segnale r7_load. Ciò che salvo che salvo è selezionato dal MUX m3.

M3: MUX che controlla con il segnale m3_sel quale indirizzo salvare nel reg7:

0: 0

1: contenuto del reg7 incrementato di 1. Con il flusso delle parole da codificare, il contenuto di reg7 cresce continuamente di 1.

Reg8: Registro in cui salvo l'indirizzo della memoria a 16 bit su cui scrivere il risultato, pilotato dal segnale r8_load. L'indirizzo che salvo è selezionato dal MUX m4 e parte da 1000.

M4: MUX che controlla con il segnale m4_sel quale indirizzo salvare nel reg8:

0: 1000

1: contenuto di reg8 incrementato di 2. Viene incrementato di 2 perché il risultato è diviso in due parti da 8 bit.

M5: MUX che controlla il valore di o_address attraverso il segnale m5_sel:

00: 0

01: contenuto di reg7 + 1

10: contenuto di reg8

11: contenuto di reg8 + 2

D1: flipflop D come descritto nella specifica: collegato a una delle due porte XOR e al MUX m1.

M1: MUX che controlla il segnale di ingresso del flipflop D1 col segnale m1_sel:

00: D1

01: il bit selezionato da Msel

10: 0

D2: flipflop D come descritto nella specifica: collegato alle due porte XOR e al MUX m2.

M2: MUX che controlla il segnale di ingresso del flipflop D1 col segnale m1_sel:

00: D2

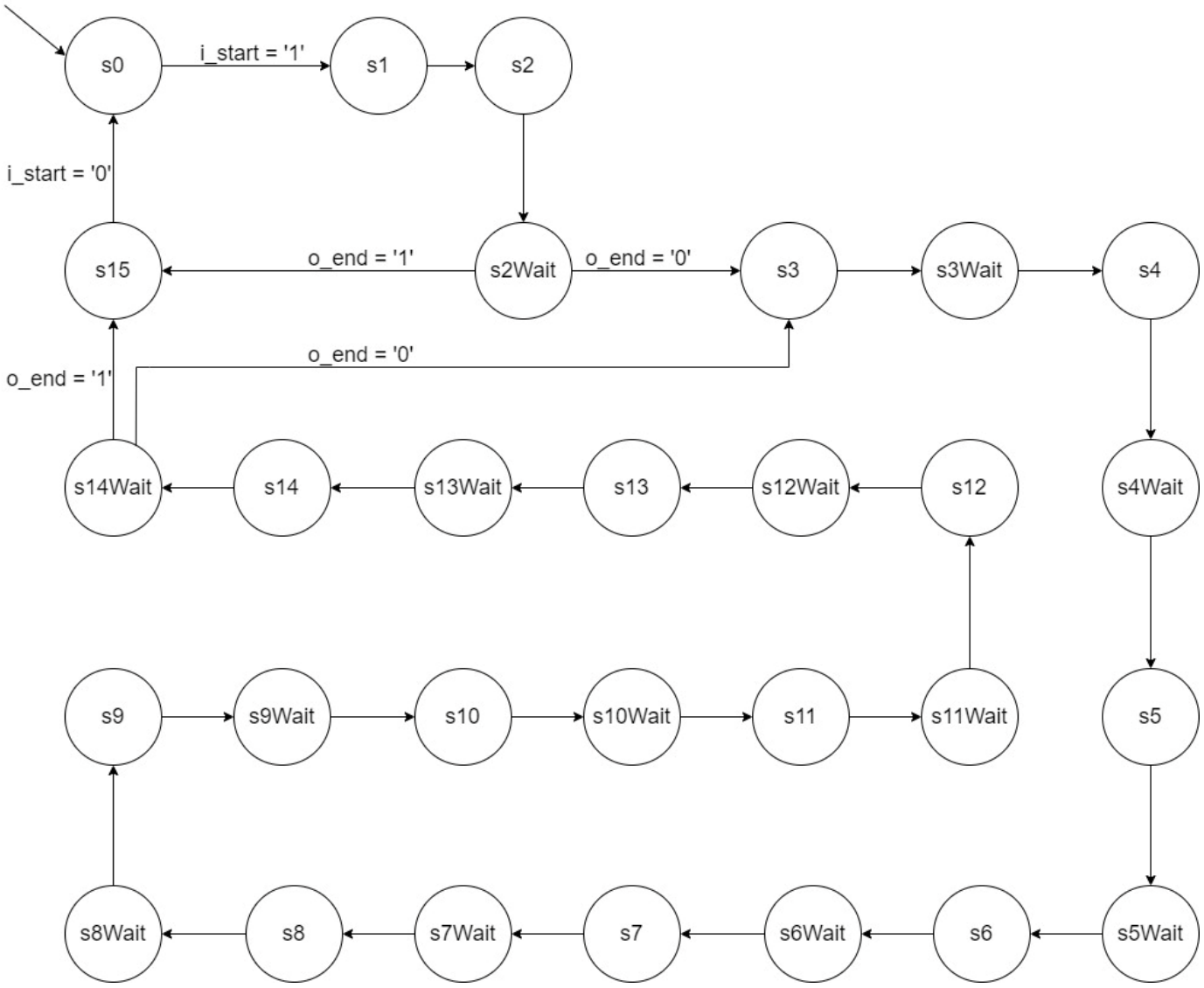
01: D1

10: 0

M1 e M2 servono per garantire un corretto reset dei flipflop D1 D2.

Comparatore: confronta il numero di parole trovate con il numero di parole da tradurre. Se è uguale porta o_end a 1.

3.2 MACCHINA A STATI



3.2.1 Valori di default:

r1_load = r2_load = r3_load = r4_load = r5_load = r6_load = r7_load = r8_load = 0

s_sel = 000

m1_sel = m2_sel = 10

m3_sel = m4_sel = 1

m5_sel = 00

o_en = o_we = o_done = 0.

3.2.2 STATI:

-**S0**: r7_load = r8_load = 1, m1_sel = m2_sel = 00, m3_sel = m4_sel = 00

-**S1**: o_en = 1, m5_sel = 00

-**S2**: r6_load = 1

-**S3**: o_en = 1, m5_sel = 01

-**S4**: r1_load = 1

-**S5**: s_sel = 111, m1_sel = m2_sel = 01, r2_load = 1

-**S6**: s_sel = 110, m1_sel = m2_sel = 01, r3_load = 1

-**S7**: s_sel = 101, m1_sel = m2_sel = 01, r4_load = 1

-**S8**: s_sel = 100, m1_sel = m2_sel = 01, r5_load = 1

-**S9**: o_en = 1, o_we = 1, m5_sel = 10

-**S10**: s_sel = 011, m1_sel = m2_sel = 01, r2_load = 1

-**S11**: s_sel = 010, m1_sel = m2_sel = 01, r3_load = 1

-**S12**: s_sel = 001, m1_sel = m2_sel = 01, r4_load = 1

-**S13**: s_sel = 000, m1_sel = m2_sel = 01, r5_load = 1

-**S14**: o_en = 1, o_we = 1, m5_sel = 11, r7_load = 1, r8_load = 1

-**S15**: o_done = 1

3.2.3 Descrizione stati:

S0: stato iniziale che viene utilizzato anche come stato di reset. In questo stato tutti i componenti del datapath vengono inizializzati. I registri 7 e 8 vengono inizializzati, rispettivamente, a 0 e 1000 e i flip flop a 0. Se il segnale start è a 1 si passa allo stato S1 altrimenti si rimane in S0.

S1: si legge dalla memoria nell'indirizzo 0 il numero di parole da codificare (memoria asincrona, quindi il numero verrà fornito dalla memoria nel ciclo successivo).

S2: si scrive nel registro 1 il numero richiesto alla memoria nello stato S1.

S2Wait: si valuta il valore di o_end fornito dal comparatore, se o_end = 0 si passa allo stato S3, se invece è o_end = 1 si passa a S15.

S3: si legge dalla memoria la parola da codificare.

S4: si salva la parola richiesta al ciclo precedente nel registro 1.

S5-S8: stati per il processo di codifica dei primi 4 bit della parola salvata nel registro 1.

S9: scrittura in memoria della concatenazione dei registri 2, 3, 4 e 5 nella cella di memoria al numero salvato nel registro 8.

S10-S13: stati per il processo di codifica dei secondi 4 bit della parola salvata nel registro 1.

S14: scrittura in memoria della concatenazione dei registri 2, 3, 4 e 5 nella cella di memoria al numero salvato nel registro 8 più 1.

S14wait: si valuta il valore di o_end fornito dal comparatore, se o_end = 0 si passa allo stato S3, se invece è o_end = 1 si passa a S15.

S15: la codifica termina e il segnale o_done viene posto a 1. Si passa allo stato S0 solo quando il segnale i_start sarà portato a 0. Questo è l'unico stato in cui o_done è a 1, in tutti gli altri è a 0.

SXwait: sono gli stati di attesa che fanno in modo che i segnali del DATAPATH assumano i valori corretti, così da permettere il giusto utilizzo di questi da parte degli altri stati.

4 RISULTATI SPERIMENTALI

4.1 REPORT

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	78	0	0	134600	0.06
LUT as Logic	78	0	0	134600	0.06
LUT as Memory	0	0	0	46200	0.00
Slice Registers	85	0	0	269200	0.03
Register as Flip Flop	85	0	0	269200	0.03
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Timing Report

Slack (MET) : 5.377ns (required time - arrival time)

4.2 SIMULAZIONI

I seguenti test vengono passati sia in Behavioral Simulation, sia in Post-Synthesis.

Test 1: verifica la condizione di esempio fornita dalle specifiche.

Test 2: sequenza di 6 parole e RESET asincrono.

Test 2bis: elaborazioni consecutive con RESET sulla prima.

Test 3: sequenza di lunghezza massima RAM(0) = ‘11111111’).

Test 3bis: sequenza di lunghezza nulla (RAM(0) = ‘00000000’).

Test 4: double processing sulla stessa RAM.

Test 5: codifica 3 flussi uno dopo l’altro.

5 CONCLUSIONI

Il modello realizzato soddisfa largamente le richieste proposte dalla specifica: non presenta Latch e il clock è inferiore a 100ns.

La specifica era molto chiara e non è stato difficile implementare il modulo, seguendo quanto spiegato a lezione e grazie anche ad un ottimo lavoro di squadra. Dopo poche difficoltà iniziali legate al funzionamento del programma utilizzato, Vivado, grazie ad una progettazione precisa ed efficace, non ci sono stati troppi problemi nella stesura del codice. Non abbiamo incontrato molti problemi nemmeno nel momento di test. Per questo, abbiamo potuto spendere le ultime ore dedicate al progetto per cercare di ottimizzare alcuni stati e controllare che non avessimo lasciato nulla in sospeso, e alla fine siamo stati soddisfatti del risultato ottenuto.

La realizzazione di questo progetto è stata molto interessante e ha permesso di affinare ciò che avevamo imparato nel corso anche da un punto di vista pratico. Poiché è stato svolto in due, è stato utile anche per migliorare la nostra capacità di lavorare in gruppo e la comunicazione, che è stata fondamentale per un corretto ed efficace svolgimento della prova.