POLITECNICO DI TORINO

Corso di Laurea Specialistica
in Ingegneria Elettronica

Tesi di Laurea Specialistica

Sviluppo di una piattaforma tollerante ai guasti a ridondanza multipla per applicazioni ferroviarie

Relatore

Prof. Leonardo Reyneri

Co-relatori aziendali

Stefano Serri
Giovanni Sica

Candidato
Sara Tronci

1. Abstract

Nel corso degli anni l'affidabilità di un sistema ha acquisito maggiore peso durante le fasi della sua progettazione. L'analisi di affidabilità prevista non mira solamente a stabilire per quanto tempo il dispositivo rispetterà le specifiche di progetto, ma analizza i guasti a cui è soggetto in modo da ridurli: si studia il contesto, i tipi di guasti che si possono verificare, la probabilità che si verifichino e si fa una valutazione del danno che questo guasto può provocare a persone o cose. Solitamente questa analisi è supportata da simulazioni del sistema che tengono conto delle variabili dell'ambiente circostante, ci sono però contesti in cui il livello di sicurezza fornito deve essere molto alto, e c'è bisogno di affidarsi a un modello reale. Lo scopo della tesi è stato quindi l'implementazione di una piattaforma di test che consenta di analizzare i possibili rischi di guasto in modo da generare una serie di requisiti utili per progettare un'apparecchiatura elettronica con un elevato livello di robustezza. L'idea alla base è un "Sistema Ridondante Fault Tolerance" costituito da tre moduli equivalenti tra loro che svolgono le stesse funzioni. In caso di guasto il sistema è in grado di garantire un uscita certa e continuare a essere operativo, segnalando la presenza del guasto. La piattaforma é stata pensata per essere utilizzata in abito ferroviario, un contesto in cui prevenire un malfunzionamento o arrivare a prevederlo prima che si verifichi con un'accurata diagnostica comporta evitare che si verificano incidenti di elevata entità . Un possibile studio oggetto della tesi è la trasmissione di comandi tra locomotive in un treno con trazione distribuita e comandi di frenata. Le due locomotive comunicano tra loro tramite un ponte radio. Si è cercato di realizzare un dimostratore in grado di trasferire controlli e comandi tra due moduli sicure per questo lavoro di analisi il ponte radio è stato sostituito da una comunicazione cablata.

2. Introduzione

L'argomento della tesi prende spunto da un'applicazione ferroviaria in cui è utilizzato un sistema ridondante:

sono presenti all'interno dell'unità due CPU che lavorano in modalità auto-backup. Le CPU elaborano gli stessi dati comunicano a tra loro e in caso di guasto del master la seconda CPU prende il controllo.

L'unità in questione è presente ad esempio sulla locomotiva e replicata nei vagoni, e gestisce ad i segnali di controllo sulla trazione, rileva i valori di trasduttori di pressione etc.

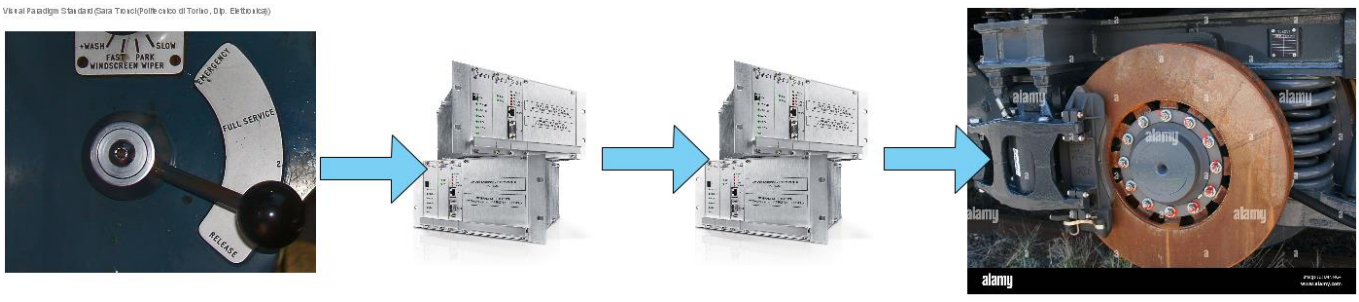
Il passo avanti è quello di migliorare questo tipo di soluzione incrementando il numero di CPU da due a tre.

In questo modo in presenza di un'incongruenza tra le due CPU ci sarà sempre un'uscita certa il sistema segnalerà il fault e potrà continuare a funzione in attesa di manutenzione.

L'obbiettivo della tesi è stata analizzare e implementare un dispositivo di questo genere sviluppando insieme ad esso una piattaforma di generazione dei guasti.

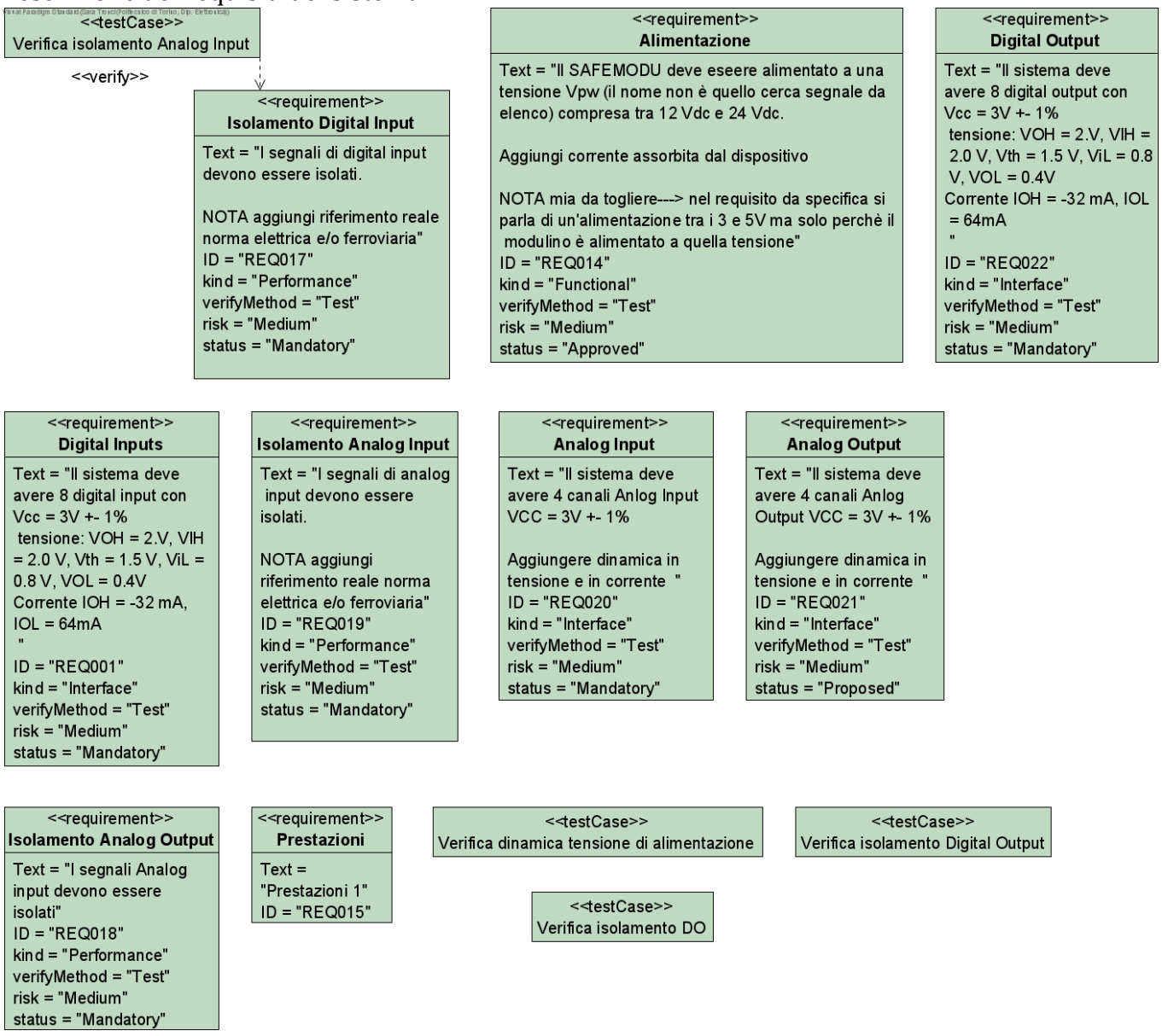
La tesi si articola in 5 macro capitoli analoghi alle 5 fasi di sviluppo di un progetto, con la libertà di poter relativamente trascurare i vincoli di TEMPO e COSTO:

1. AVVIO DEL PROGETTO --> Prima di iniziare a implementare l'architettura è stata effettuata un'analisi dei tipi di guasti che si possono verificare utilizzando degli automi a stati e i diagrammi temporali dei segnali.
2. PIANIFICAZIONE --> Partendo dalle analisi e dalle specifiche del dispositivo, mostrate nel capitolo successivo, sono stati proposti un diagramma a blocchi e una possibile architettura del dispositivo. Il risulta di questo confronto è stato stillare un "piano di progetto" che ha portato alla fase successiva
3. ESECUZIONE --> Gli elementi dell'architettura proposta sono stati implementati. Sono stati fatti gli schemi elettrici, dimensionati i componenti, sviluppato parte dell'applicativo che hanno portato alla realizzazione fisica del dispositivo.
4. MONITORAGGIO E REVISIONE ---> Questa fase prevede l'insieme di tutti quei test che non mirano solo a verificare che il dispositivo funzioni correttamente ma anche la sua affidabilità. In maniera più completa è la fase in cui si verifica che i requisiti del progetto siano stati coperti. Trattandosi di una tesi e non di un prodotto da vendere, ci si è soffermati di più sulla copertura delle funzioni importanti.
5. CONCLUSIONE DEL PROGETTO --> l'ultimo capitolo è dedicato a una breve analisi sui problemi riscontrati, su cosa potrebbe essere migliorato e su quali potrebbero essere l'evoluzione del progetto.



3. Requisiti di Sistema

Descrizione dei requisiti del sistema



3.1. Digital Inputs

Il sistema deve avere 8 digital input con Vcc = 3V +- 1%

- tensione: VOH = 2.V, VIH = 2.0 V, Vth = 1.5 V, ViL = 0.8 V, VOL = 0.4V
- Corrente IOH = -32 mA, IOL = 64mA

Text: Il sistema deve avere 8 digital input con Vcc = 3V +- 1%
tensione: VOH = 2.V, VIH = 2.0 V, Vth = 1.5 V, ViL = 0.8 V, VOL = 0.4V
Corrente IOH = -32 mA, IOL = 64mA

3.2. Alimentazione

Il SAFEMODU deve eseere alimentato a una tensione Vpw (il nome non è quello cerca segnale da elenco) compresa tra 12 Vdc e 24 Vdc.

Aggiungi corrente assorbita dal dispositivo

NOTA mia da togliere---> nel requisito da specifica si parla di un'alimentazione tra i 3 e 5V ma solo perchè il modulino è alimentato a quella tensione

Text: Il SAFEMODU deve eseere alimentato a una tensione Vpw (il nome non è quello cerca segnale da elenco) compresa tra 12 Vdc e 24 Vdc.

Aggiungi corrente assorbita dal dispositivo

NOTA mia da togliere---> nel requisito da specifica si parla di un'alimentazione tra i 3 e 5V ma solo perchè il modulino è alimentato a quella tensione

3.3. Prestazioni

Prestazioni 1

Text: Prestazioni 1

3.4. Isolamento Digital Input

I segnali di digital input devono essere isolati.

NOTA aggiungi riferimento reale norma elettrica e/o ferroviaria

Text: I segnali di digital input devono essere isolati.

NOTA aggiungi riferimento reale norma elettrica e/o ferroviaria

3.5. Isolamento Analog Output

I segnali Analog input devono essere isolati

Text: I segnali Analog input devono essere isolati

3.6. Isolamento Analog Input

I segnali di analog input devono essere isolati.

NOTA aggiungi riferimento reale norma elettrica e/o ferroviaria

Text: I segnali di analog input devono essere isolati.

NOTA aggiungi riferimento reale norma elettrica e/o ferroviaria

3.7. Analog Input

Il sistema deve avere 4 canali Anlog Input VCC = 3V +- 1%

Aggiungere dinamica in tensione e in corrente

Text: Il sistema deve avere 4 canali Anlog Input VCC = 3V +- 1%

Aggiungere dinamica in tensione e in corrente

3.8. Analog Output

Il sistema deve avere 4 canali Anlog Output VCC = 3V +- 1%

Aggiungere dinamica in tensione e in corrente
Text: Il sistema deve avere 4 canali Anlog Output VCC = 3V +- 1%

Aggiungere dinamica in tensione e in corrente

3.9. Digital Output

Il sistema deve avere 8 digital output con Vcc = 3V +- 1%

- tensione: VOH = 2.V, VIH = 2.0 V, Vth = 1.5 V, ViL = 0.8 V, VOL = 0.4V
- Corrente IOH = -32 mA, IOL = 64mA

Text: Il sistema deve avere 8 digital output con Vcc = 3V +- 1%
tensione: VOH = 2.V, VIH = 2.0 V, Vth = 1.5 V, ViL = 0.8 V, VOL = 0.4V
Corrente IOH = -32 mA, IOL = 64mA

4. Caso di Studio

Per analizzare i possibili guasti che si possono presentare dello sviluppo del progetto di SAFEMODU di cui si è parlato nell'introduzione, si è pensato di progettare una piattaforma di analisi dei guasti che va a sostituire il disposivo in questione.

Nei paragrafi successivi sono descritte due impostazioni differenti del sistema:
1) L' Applicazione guida - opzione proposta che costituiva il sistema iniziale in cui la piattaforma di analisi dei guasti è un Dimostratore Multipiattaforma. In questo contesto si ipotizza di progettare il SAFEMODU con un tripla rindondaza di tre microcontrollori basati su architteture differenti.
2) L'Applicazione guida - opzione implementata, che costituisce il sistema come è stato realmente implementato durante il corso della della tesi e in cui la piattaforma di analisi dei guasti è un Dimostratore Monopiattaforma
In questo sistema i microcontrollori hanno la stessa architettura ma funzionano a frequenze di clock differenti. Quindi i segnali di ingresso e di controllo di ciascun uC saranno ritardate rispetto agli altri due.
Questa appena citata è una delle motivazioni principali principali per cui si propagano le commutazioni spurie dovute ad esempio a un corto circuito in ingresso.

Segue la specifica dei requisiti a cui ho fatto riferimento:

5. 1 Reference standards.

EN 50155	CENELEC - Electronic equipment used on rolling stock	Nov. 1995
EN 50121-1	CENELEC - Electromagnetic compatibility - part 1 general	Feb. 1996
EN 50121-2-3	CENELEC - Electromagnetic compatibility - part 2-3 rolling stock	

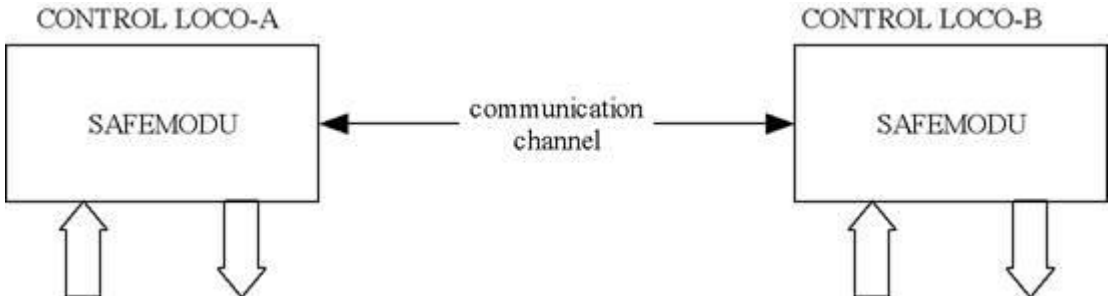
6. 2 Scope.

The scope of this document is to make a list of requirements in order to design an electronic equipment with an high level of safety integrity able to transfer command and controls between 2 equipment.
A typical example can be the transmission of commands between locomotive in a train with distributed traction and brake controls.
The scope of this job is to make a demonstrator able to tranfer command and controls between 2 safe modules.

The context of the system is the transmission of command and control between 2 locos that are connected with a radio link.

For this job we will replace the radio link with a wired communication (CAN / ETHERNET / RS485).

In the picture below is summarized the context of the system.



7. 3 Interface requirements for SAFEMODU.

The SAFEMODU must be designed in order to fullfil the following requirements :

- The SAFEMODU must have 8 input LVTTL, each input can be used a safe input (depending on the application)
- The input of SAFEMODU must be insulated (using digital isolators).
- The SAFEMODU must have 8 output LVTTL, each output can be used as safe output (depending on the application)
- The output of SAFEMODU must be insulated (using digital isolator).
- The SAFEMODU must have 4 analog input channels (fs 3.3V) internally the module must have one or more A/D converter to read these signals.
- The SAFEMODU must have 4 analog output channels (fs 3.3V) internally the module have one or more D/A converter to generate these signals.
- The supply power of the SAFEMODU must be 3.3V or 5V.
- The local power must be insulated respect the internal power for safety requirements (EN50129).
- The SAFEMODUE must have an output (LVTTL) to signal the FAIL STATE of the module.
- The FAIL STATE output must generate a life signal 50Hz when the SAFEMODU is working well (the concept of working well depends from the internal diagnostic of the SAFEMODU).
- The SAFEMODU must have 1/2 CAN or RS485 channels for remote communication.
- The communication channel need to be insulated.

8. 4 Functional requirements

In the example project we need to make a design of a SAFEMODU able to transfer to a SAFEMODU partner the traction and braking commands.

The I/O don't of the module must be transferred to the partner module with the safety integrity specified; the signal don't need complex processed locally by the SAFEMODU.

The local process of the SAFEMODU is restricted to the control of congruence of the data and the integrity of the complete module to achieve the level of diagnosticability requested for the SIL level specified.

8.1. 4.1 Digital input.

In the table below list of the digital input signals of SAFEMODU :

DI	DESCRIPTION
1..0	Traction/Brake (generated only by LOCO-MASTER) 00 à coasting 01 à traction 10 à braking 11 à coasting
3..2	LOCO MODE 00 à not allowed 01 à LOCO-MASTER 10 à LOCO-SLAVE 11 à not allowed
5..4	MOVEMENT DIRECTION (generated only by LOCO-MASTER) 00 à not allowed 01 à Cab.Enabled.A 10 à Cab.Enabled.B 11 à not allowed
6	
7	SAFETY BRAKE 0 or 1 steady state requires a safety brake application 50Hz SAFETY BRAKE not requested

8.2. 4.2 Digital output.

In the table below list of the digital output signals of SAFEMODU :

DO	DESCRIPTION
1..0	Traction/Brake repetion signals 00 à coasting 01 à traction 10 à braking 11 à coasting
3..2	LOCO MODE repetition signals 00 à not allowed 01 à LOCO-MASTER 10 à LOCO-SLAVE 11 à not allowed

5..4	MOVEMENT DIRECTION repetition signals on loco slave 00 à not allowed 01 à Cab.Enabled.A 10 à Cab.Enabled.B 11 à not allowed
6	Not used
7	SAFETY BRAKE REQUEST repetition signal 0 or 1 steady state requires a safety brake application 50Hz SAFETY BRAKE not requested

8.3. 4.3 Analog input.

In the table below the list of the analog input signals of SAFEMODU :

AI	DESCRIPTION
0	BrakeDemand generated only by loco master
1	TractionDemand generated only by loco master
2	EDbrake electrodynamics brake by the LOCO which is installed the SAFEMODU
3	BrakeApplied by the LOCO which is installed the SAFEMODU

Each analogue input signal is considered valid when is in the range 0,4..2,9V outside this range the input signal need to be considered faulty.

8.4. 4.4 Analog output.

In the table below list of the analog output signals of SAFEMODU :

AO	DESCRIPTION
0	BrakeDemand repetition signal of the LOCO MASTER
1	TractionDemand repetition signal of the LOCO MASTER
2	EDbrake electrodynamics brake of the LOCO partner
3	BrakeApplied by the LOCO partner

Each analogue output signal is considered valid when is in the range 0,4..2,9V outside this range the output signal need to be considered faulty.

The scope of the SAFEMODU is to transfer every 100ms a stream of data trough the communication channel from LOCO-MASTER and LOCO-SLAVE and viceversa allowing the repetition of the control signals between the 2 locos.

Each SAFEMODU must read the digital input and analog input safely packing the data and trasmit over the safe channels.

Each SAFEMODU must get from the network the remote frame from the loco partner and must repeat the signal in a safe way.

Locally the SAFEMODU must run the diagnostic of the input signals generating an ALERT to the communication partner and locally when the input are not congruent.

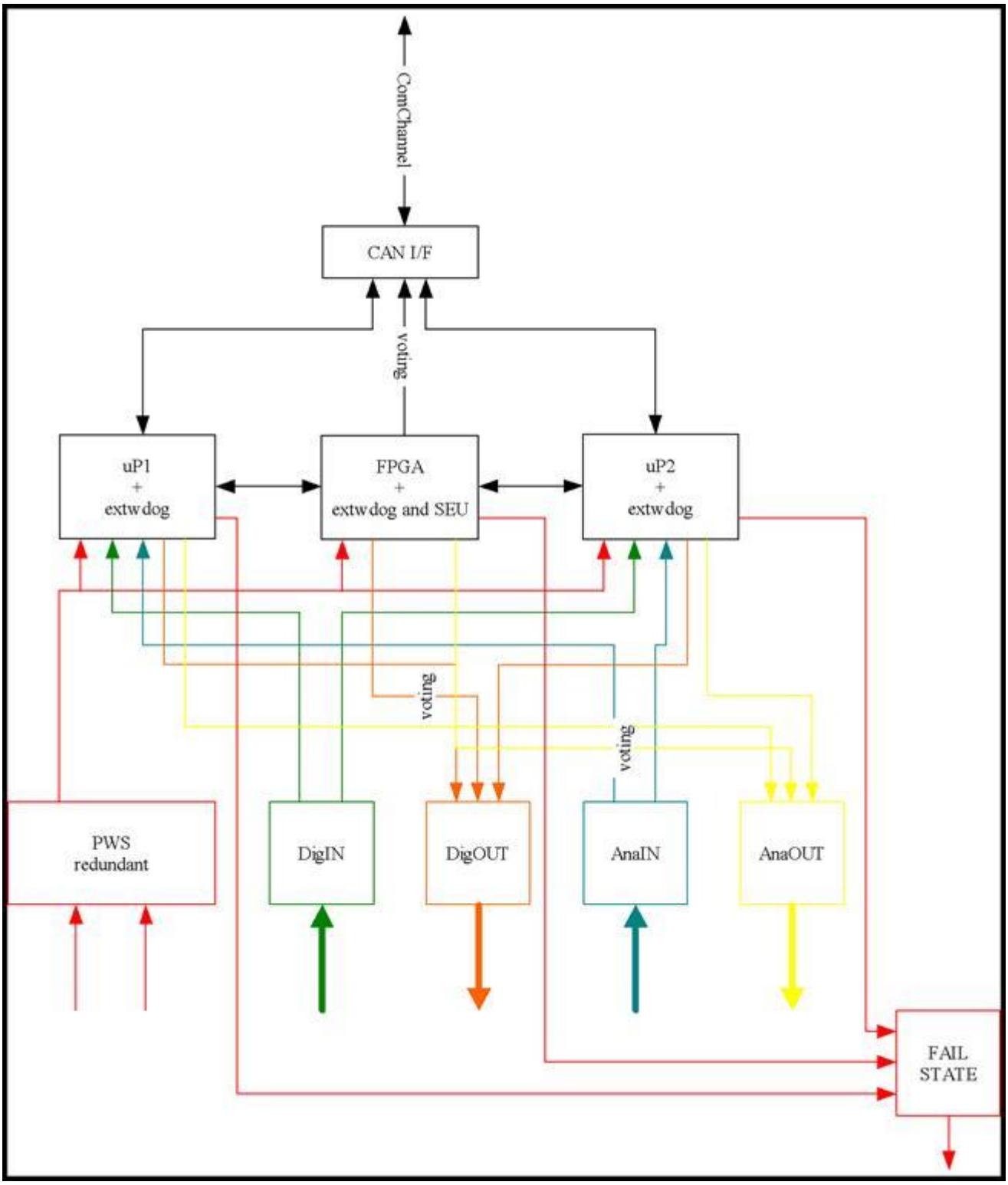
Locally the SAFEMODY must repeat the signals received from the network frame generating a local alert if the input data are not congruent.

According to the protocol implementation between the LOCO, is requested a robust control able to guarantee a safe discard of the not recoverable frame and an auto-correction of the frame when it's possible.

A frame must be auto-corrected by the receiver in case of 1bit flip.

A frame must be discarded by the receiver in case of 2 or more bit flip.

In the picture below a block scheme of the SAFEMODU.



9. 5 Safety requirements

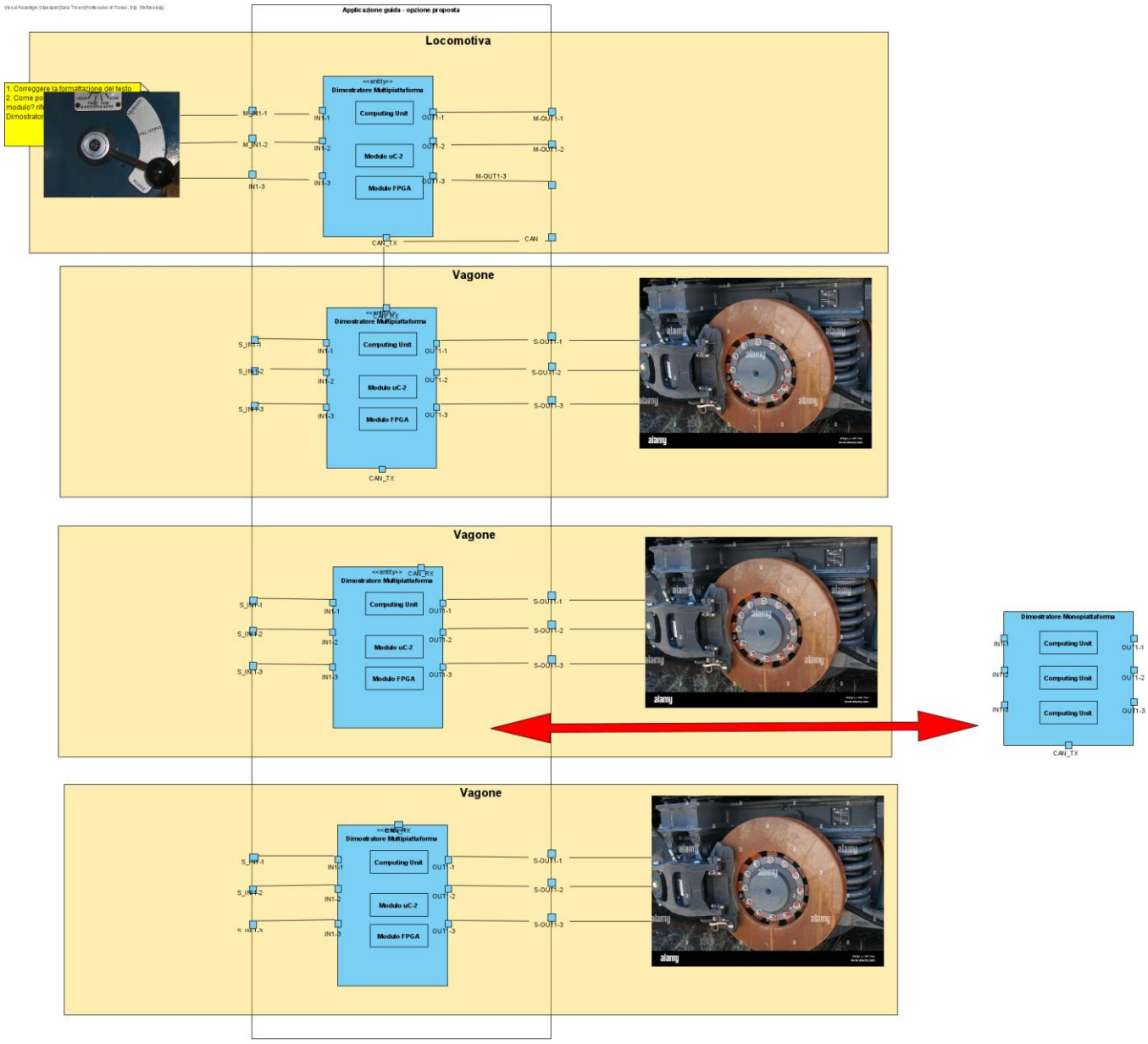
The SAFEMODU must be designed according the guide lines of EN50126 (overall), EN50129 and IEC61508 for HW and EN50128 for SW and logic elements.

The safety communication channel has to be complaint with EN50159.

The minimum safety integrity level of the module must be qualitative SIL3 and with THR < 10e-7 .

The insensibility to the hazard must be HFT=1 excluding the communication channel between the module.

Regarding the communication channel between the SAFEMODU, it can think to the extension by defining a 2nd backup channel. The diagnosticability of each parts units must achieve the requirements to fullfil the SIL requirem



9.1. Modulo uC-2

Modulo generico a microcontrollore, non ridondato, con alimentazione a 5V/12V (TBD) con:

1. 8 IN digitali LV-TTL 3.3V
2. 8 out digitali LV-TTL 3.3V
3. 4 in analogici 0-3.3V
4. 2 out analogici o PWM 0-3.3V con estensione di altri 2 pin (non connessi) per future estensioni
5. 2 porte UART
6. 2 porte CAN

Il modulo è costituito dal microprocessore MSP430.
Il clock e WDT sono interni al microprocessore utilizzato.

9.1.1. VAL
Pin di alimentazione del Computing Unit
Tensione compresa fra 6V e 20V

9.1.2. DIN0
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

9.1.3. DIN1
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

9.1.4. DIN2
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

9.1.5. AIN0
Pin d'ingresso analogico del Computing Unit LV-TTL 3.3V.

9.1.6. AIN1
Pin d'ingresso analogico del Computing Unit LV-TTL 3.3V.

9.1.7. DOUT0
Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

9.1.8. DOUT1
Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

9.1.9. DOUT2
Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

9.1.10. AO0/PWM0
Pin d'uscita del segnale analogico o PWM del Computing Unit LV-TTL 3.3V.

9.1.11. AO1/PWM1
Pin d'uscita del segnale analogico o PWM del Computing Unit LV-TTL 3.3V.

9.1.12. AIN2
Pin d'ingresso analogico del Computing Unit LV-TTL 3.3V.

9.1.13. DIN3
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

9.1.14. DIN4
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

9.1.15. DIN5
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

9.1.16. DIN6
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

9.1.17. DIN7
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

9.1.18. AIN3
Pin d'ingresso analogico del Computing Unit LV-TTL 3.3V.

9.1.19. DOUT3

Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

9.1.20. DOUT4

Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

9.1.21. DOUT5

Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

9.1.22. DOUT6

Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

9.1.23. DOUT7

Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

9.1.24. AO2/PWM2

Pin d'uscita del segnale analogico o PWM del Computing Unit LV-TTL 3.3V. Questo pin al momento è di estensione, non risulta connesso.

9.1.25. AO3/PWM3

Pin d'uscita del segnale analogico o PWM del Computing Unit LV-TTL 3.3V. Questo pin al momento è di estensione, non risulta connesso.

9.1.26. UART_A

connection uart pin.

9.1.27. UART0_

connection uart pin.

9.2. Modulo FPGA

Modulo generico a microcontrollore o FPGA, non ridondato, con alimentazione a 5V/12V (TBD) con:

1. 8 IN digitali LV-TTL 3.3V
2. 8 out digitali LV-TTL 3.3V
3. 4 in analogici 0-3.3V
4. 2 out analogici o PWM 0-3.3V con estensione di altri 2 pin (non connessi) per future estensioni
5. 2 porte UART
6. 2 porte CAN

Il processore utilizzato è un FPGA Altera MAX10 o Cyclone-IV + VHDL + ADC124S021

DAC da usare:

- MCP4922

WDT esterno al microprocessore/FPGA:

- TPS3813K33

Partitore di tensione su VAL con jumper su uno degli AIN

9.2.1. VAL

Pin di alimentazione del Computing Unit
Tensione compresa fra 6V e 20V

9.2.2. DIN0

jfgkjdfsks

9.2.3. UART

connection uart pin.

9.2.4. UART1

connection uart pin.

9.3. Dimostratore Multipiattaforma

Modulo principale, dotato di alimentazione a 12V/24V (verifica il valore dell'alimentazione) con:

1. 8 segnali Digital Input
2. 8 segnali Digital Output
3. 4 segnali Analog Input
4. 4 segnali Analog Output
5. 2 porte UART
6. 2 porta CAN

Questo modulo integra le seguenti componenti:

1. Un Computing Unitche utilizza il microcontrollore STM32F072VB.
2. Un Modulo uC-2 che utilizza un microcontrollore della famiglia MSP430
3. Un Modulo FPGA.
4. Un DIN replicator-1 in base al numero di ingressi digitali, per evitare il cortocircuito del segnale in ingresso oppure un DIN replicator in base al numero di ingressi digitali, per evitare il cortocircuito del segnale ingresso, con pin dedicato.
5. AIN replicator per la protezione degli ingressi analogici.
6. OUT voter-1OUT voter

9.3.1. VAL-1

Pin di alimentazione del Dimostratore Multipiattaforma. La tensione è compresa fra 6V e 20V, dedicato al Modulo uC o al Modulo FPGA.

9.3.2. VAL-2

Pin di alimentazione del Dimostratore Multipiattaforma. La tensione è compresa fra 6V e 20V, dedicato al Modulo uC o al Modulo FPGA.

9.3.3. VAL-3

Pin di alimentazione del Dimostratore Multipiattaforma. La tensione è compresa fra 6V e 20V, dedicato al Modulo uC o al Modulo FPGA.

9.3.4. IN0-1

Pin d'ingresso digitale del Dimostratore Multipiattaforma

9.3.5. IN1-1

Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.6. IN1-2

Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.7. IN1-3

Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.8. AIN0-1

Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.9. AIN0-2
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.10. AIN0-3
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.11. OUT1-1
Pin del segnale digitale d'uscita del Dimostratore Multipiattaforma.

9.3.12. OUT1-2
Pin del segnale digitale d'uscita del Dimostratore Multipiattaforma.

9.3.13. OUT1-3
Pin del segnale digitale d'uscita del Dimostratore Multipiattaforma.

9.3.14. VAL-4
Pin di alimentazione del Dimostratore Multipiattaforma. La tensione è compresa fra 6V e 20V, che fornisce l'alimentazione all'OUT voter-1.

9.3.15. IN2-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.16. IN2-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma, connesso al pin (model element not found) del (model element not found).

9.3.17. IN2-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.18. IN3-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.19. IN3-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.20. IN3-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.21. IN4-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.22. IN4-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.23. IN4-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.24. IN5-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.25. IN5-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.26. IN5-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.27. IN6-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.28. IN6-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.29. IN6-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.30. IN7-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.31. IN7-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.32. IN7-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.3.33. IN0-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma

9.3.34. AIN1-1
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.35. AIN1-2
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.36. AIN1-3
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.37. AIN2-1
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.38. AIN2-2
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.39. AIN2-3
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.40. AIN3-1
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.41. AIN3-2
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.3.42. AIN3-3
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.4. Applicazione guida - opzione proposta

L'Applicazione guida - opzione proposta, costituita da due Dimostratore Multipiattaforma uno usato come Master e l'altro come Slave e che comunicano tra li loro tramite rete CAN.

9.4.1. M_IN1-1
Prima replica del segnale IN1 inviata al Dimostratore Multipiattaforma Master.

9.4.2. M_IN1-2
Seconda replica del segnale IN1 inviata al Dimostratore Multipiattaforma Master.

9.4.3. IN1-3

Terza replica del segnale IN1 inviata al Dimostratore Multipiattaforma Master.

9.4.4. S-OUT1-1

Prima replica del segnale d'uscita OUT1 del Dimostratore MultipiattaformaSlave.

9.4.5. S-OUT1-2

Seconda replica del segnale d'uscita OUT1 del Dimostratore Multipiattaforma Slave.

9.4.6. S-OUT1-3

Terza replica del segnale d'uscita OUT1 del Dimostratore MultipiattaformaSlave.

9.4.7. S_IN1-1

Prima replica del segnale IN1 inviata al Dimostratore Multipiattaforma Slave.

9.4.8. S_INI1-2

Seconda replica del segnale IN1 inviata al Dimostratore Multipiattaforma Slave.

9.4.9. S_INI1-3

Seconda replica del segnale IN1 inviata al Dimostratore Multipiattaforma Slave.

9.4.10. M-OUT1-1

Prima replica del segnale d'uscita OUT1 del Dimostratore Multipiattaforma Master.

9.4.11. M-OUT1-2

Seconda replica del segnale d'uscita OUT1 del Dimostratore Multipiattaforma Master.

9.5. Dimostratore Monopiattaforma

Modulo principale, dotato di alimentazione a 12V/24V (verifica il valore dell'alimentazione) con:

1. 8 segnali Digital Input
2. 8 segnali Digital Output
3. 4 segnali Analog Input
4. 4 segnali Analog Output
5. 2 porte UART
6. 2 porta CAN

Questo modulo integra le seguenti componenti:

1. Tre Computing Unit identici che utilizzano il microcontrollore STM32F072VB.
2. Un DIN replicator-1 in base al numero di ingressi digitali, per evitare il cortocircuito del segnale in ingresso oppure un DIN replicator in base al numero di ingressi digitali, per evitare il cortocircuito del segnale ingresso, con pin dedicato.
3. AIN replicator per la protezione degli ingressi analogici.
4. OUT voter-1OUT voter

9.5.1. IN1-1

Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.2. IN1-2

Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.3. IN1-3

Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.4. VAL-1
Pin di alimentazione del Dimostratore Multipiattaforma. La tensione è compresa fra 6V e 20V, dedicato al Modulo uC o al Modulo FPGA.

9.5.5. VAL-2
Pin di alimentazione del Dimostratore Multipiattaforma. La tensione è compresa fra 6V e 20V, dedicato al Modulo uC o al Modulo FPGA.

9.5.6. VAL-3
Pin di alimentazione del Dimostratore Multipiattaforma. La tensione è compresa fra 6V e 20V, dedicato al Modulo uC o al Modulo FPGA.

9.5.7. IN0-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma

9.5.8. AIN0-1
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.9. AIN0-2
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.10. AIN0-3
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.11. VAL-4
Pin di alimentazione del Dimostratore Multipiattaforma. La tensione è compresa fra 6V e 20V, che fornisce l'alimentazione all'OUT voter-1.

9.5.12. IN2-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.13. IN2-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma, connesso al pin (model element not found) del (model element not found).

9.5.14. IN2-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.15. IN3-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.16. IN3-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.17. IN3-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.18. IN4-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.19. IN4-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.20. IN4-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.21. IN5-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.22. IN5-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.23. IN5-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.24. IN6-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.25. IN6-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.26. IN6-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.27. IN7-1
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.28. IN7-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.29. IN7-3
Pin d'ingresso digitale del Dimostratore Multipiattaforma.

9.5.30. IN0-2
Pin d'ingresso digitale del Dimostratore Multipiattaforma

9.5.31. AIN1-1
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.32. AIN1-2
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.33. AIN1-3
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.34. AIN2-1
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.35. AIN2-2
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.36. AIN2-3
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.37. AIN3-1
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.38. AIN3-2
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.39. AIN3-3
Pin d'ingresso analogico del Dimostratore Multipiattaforma.

9.5.40. OUT1-1

Pin d'uscita digitale del Dimostratore Monopiattaforma.

9.5.41. OUT1-2

Pin d'uscita digitale del Dimostratore Monopiattaforma.

9.5.42. OUT1-3

Pin d'uscita digitale del Dimostratore Monopiattaforma.

10. Funzioni Preliminari del Dimostratore

In questpria fase del progetto si è deciso "quali" funzioni devono essere eseguite dal l Dimostratore Multipiattaforma e "chi" ha l'autorizzazione ad attivarle. In breve si tratta di elencare casi d'uso e attori, facenti parte del sistema, e descriverli accuratamente per capire cosa implentare.

Il seguente diagramma mostra un anteprima dei vari casi d'uso previsti dal Dimostratore Multipiattaforma. Per maggiori dettagli vedere i singoli casi d'uso.

10.1. Actors

Name	Description
CONFIGURATOR	Attore abilitato alla configurazione dei dei registri di sistema e all'abilitazione dei moduli dell'unità.
SUPERVISOR	Attore in grado di accedere a tutti i casi d'uso previsti dal Dimostratore.
FAULT SOURCE	Attore che rappresenta la sorgente di guasto o di un possibile malfuzionamento del sistema.

REMOTE ANALOG ACTUATOR	Attore che rappresenta un sensore che genera un uscita analogica, connesso da remoto all'unità.
LOCAL ANALOGSENSOR	Attore che rappresenta un sensore analogico presente all'interno dell'unità.
REMOTE BINARY ACTUATOR	Attore che rappresenta un attuatore con uscita digitale connesso da remoto al sistema.
LOCAL BINARY SENSOR	Attore che rappresenta un sensore genera un'uscita digitale presente all'interno dell'unità.

10.2. evidenzia guasto alimentazione

Verifica che dato un ingresso in continua compreso tra i 12Vdc e i 36Vdc il Voltage Regulator interno alla scheda generi una tensioni di'uscita VCC. Verificare inoltre che questa tensione sia propagata a tutti i componenti attivi presenti nella scheda.

10.3. evidenzia guasto memoria

Confronto tra i dati memorizzati nei registri dei tre Computing Unit, verificando se sono congruenti.

10.4. evidenzia guasto processore

Confronto tra i segnali di uscita di dati e controllo dei tre microcontrollori sono concordi e in caso contrario segnalare la presenza di un fault.

10.5. evidenzia guasto uscite

Analisi dei valori dei segnali d'uscita verificando se l'uscita reale corrisponde all'uscita attesa.

10.6. Configura

insieme di tutte le funzioni necessarie per settare i registri interni dell'unità dopo l'accensione.

10.7. Evidenzia Guasto

Qualunque tipo di guasto su una delle tre repliche di:

1. ingressi
2. processore
3. memoria
4. alimentazione
5. uscite

deve essere evidenziato inviando un segnale al SUPERVISOR

10.8. Verifica FOFS

Il SUPERVISOR invia un comando (via CAN).
Il sistema inietta un guasto a turno su una delle tre repliche di:

- ciascun ingresso (ad es simulando uno stuck-at 0/1)
- ciascun processore (ad es. mandandolo in halt)
- un'uscita (ad es invertendo il valore di uscita)

Il sistema verifica che:

1. venga evidenziato il guasto
2. l'uscita del sistema non vari

10.8.1. Verifica Funzionalita Identificazione Guasto DIN

1. .... accendo il (model element not found)

10.9. Sync FSA

Sincronizzazione della macchina a stati.

10.10. Receive ANALOG

Il LOCAL ANALOGSENSOR e il REMOTE ANALOG ACTUATOR ricevono un segnale analogico virtualmente unico sui tre ingressi M_IN1-1, M_IN1-2, IN1-3, ma con potenziali ritardi inter-copia e/o singoli errori.

10.11. Send ANALOG

LOCAL BINARY SENSOR e il REMOTE ANALOG ACTUATOR inviano tre repliche di un segnale analogico virtualmente unico sui tre ingressi M_IN1-1, M_IN1-2, IN1-3, ma con potenziali ritardi inter-copia e/o singoli errori.

Il sistema considera come valore quello identificato per maggioranza (due su tre)

Tale valore viene inviato alle tre uscite digitali S-OUT1-1, S-OUT1-2, S-OUT1-3 al REMOTE BINARY ACTUATOR, tramite il solo CAN BUS.

La comunicazione deve essere protetta da uno dei Casi di guasto

10.12. Receive DIGITAL

Il LOCAL BINARY SENSOR riceve un segnale digitale rtualmente unico sui tre ingressi M_IN1-1, M_IN1-2, IN1-3, ma con potenziali ritardi inter-copia e/o singoli errori.

Il sistema considera come valore quello identificato per maggioranza (due su tre)

Tale valore viene inviato alle tre uscite digitali S-OUT1-1, S-OUT1-2, S-OUT1-3 al REMOTE BINARY ACTUATOR, tramite il solo CAN BUS.

La comunicazione deve essere protetta da uno dei Casi di guasto

10.13. Compare Clocks

Confronto temporale tra i differenti CLK dei Computing Unit M1, M2 ed M3.

10.14. Casi di guasto

La comunicazione deve avvenire in modo affidabile anche in caso di uno solo dei seguenti guasti:

1. rottura di uno degli M_IN1-1, M_IN1-2, IN1-3
2. rottura di uno degli S-OUT1-1, S-OUT1-2, S-OUT1-3

3. rottura di uno dei due Computing Unit o del Modulo FPGA anche di entrambi i Dimostratore Multipiattaforma
4. errore in una locazione di uno dei due Computing Unit o del Modulo FPGA anche di entrambi i Dimostratore Multipiattaforma
5. rottura di uno dei due segnali del CAN BUS
6. mancanza di una delle alimentazioni di uno dei due Computing Unit o del Modulo FPGA anche di entrambi i Dimostratore Multipiattaforma

10.15. Send DIGITAL

LOCAL BINARY SENSOR invia tre repliche di un segnale virtualmente unico sui tre ingressi M_IN1-1, M_IN1-2, IN1-3, ma con potenziali ritardi inter-copia e/o singoli errori.

Il sistema considera come valore quello identificato per maggioranza (due su tre)

Tale valore viene inviato alle tre uscite digitali S-OUT1-1, S-OUT1-2, S-OUT1-3 al REMOTE BINARY ACTUATOR, tramite il solo CAN BUS.

La comunicazione deve essere protetta da uno dei Casi di guasto

11. Tolleranza ai Guasti in Ingresso

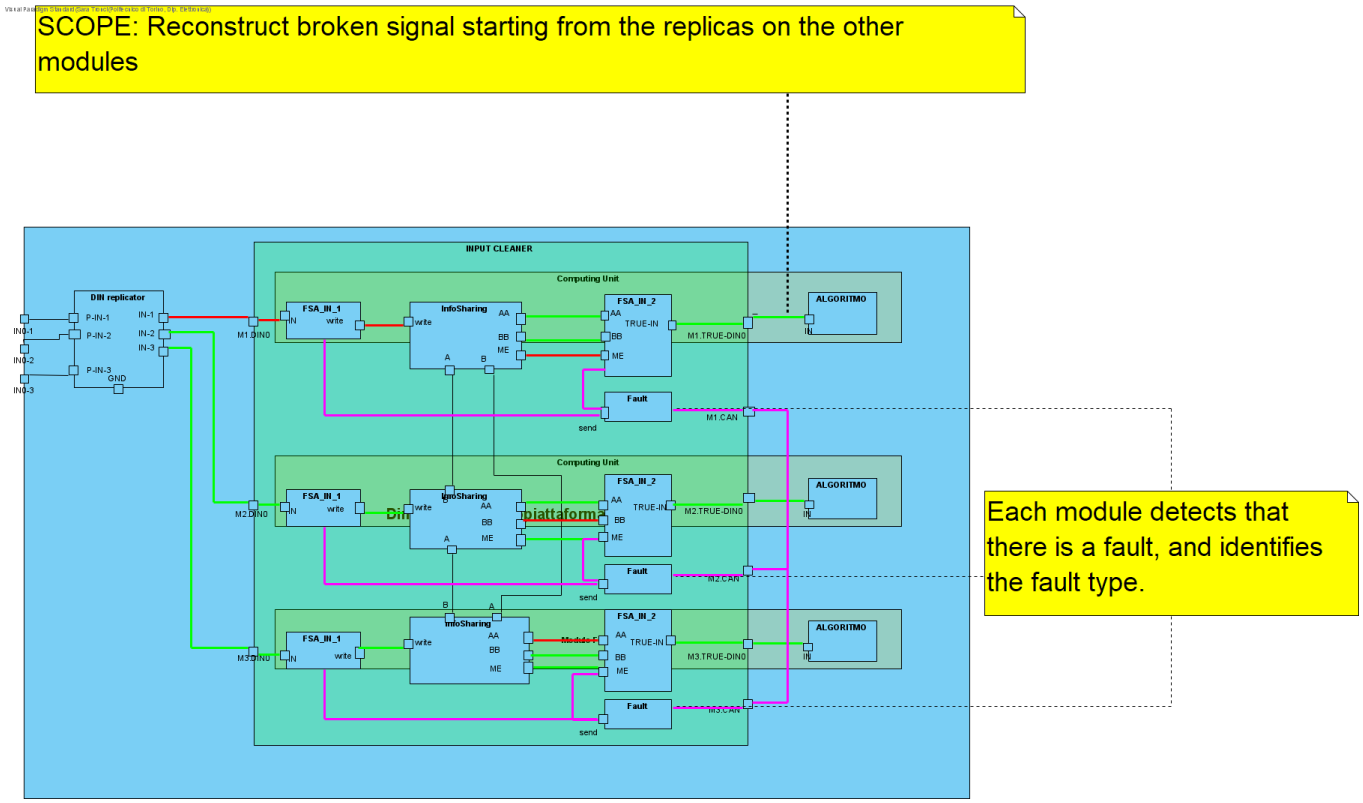
Il seguente diagramma dei componenti illustra come deve funzionare da un punto di vista logico il Dimostratore Monopiattaforma e come interagiscono tra loro i tre moduli Computing Unit nell'analisi di un possibile fault che si può verificare sui Digital Input in ingresso.

Per semplificare l'analisi, è mostrato ciò che succede a un singolo Digital Input in ingresso.

Il segnale viene triplicato è propagato ai tre moduli Computing Unit. Ognuno di essi riceve tramite comunicazione su rete CAN anche il valore che hanno ricevuto in ingresso gli altri due moduli, ed effettua un confronto.

In questo modo il segnale risulta filtrato prima di essere processato all'interno dei moduli e in caso di incongruenze il fault è segnalato.

Di seguito è mostrato nel dettaglio l'attività svolta dai singoli blocchi.



11.1. InfoSharing

11.1.1. write

Segnale di ingresso dell'InfoSharing, corrispondente al segnale DIN filtrato proveniente dalFSA_IN_1.

11.2. FSA_IN_1

Macchina a stati, in terna al Computing Unit, che riceve in ingresso il segnale digitale e lo filtra da eventuali bouncing prima di propagarlo al modulo InfoSharing.

Per maggiori dettagli vedere l'automa a stati relativo e il diagrammma temporale presente nei capitoli successivi

11.2.1. IN

Segnale d'ingresso digitale del FSA_IN_1.

11.2.2. write

Segnale d'uscita digitale del FSA_IN_1.

11.3. Computing Unit

Modulo generico a microcontrollore, non ridondato, con alimentazione a 12V/24V con:

1. 8 IN digitali LV-TTL 3.3V
2. 8 out digitali LV-TTL 3.3V
3. 4 in analogici 0-3.3V
4. 2 out analogici o PWM 0-3.3V con estensione di altri 2 pin (non connessi) per future estensioni
5. 2 porte UART
6. 2 porte CAN

Il modulo ha un clock e WDT interno al microprocessore utilizzato il STM32F072 (32-bit Cortex M0).

11.3.1. VAL

Pin di alimentazione del Computing Unit
Tensione compresa fra 6V e 20V

11.3.2. DIN0

Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

11.3.3. DIN1

Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

11.3.4. DIN2

Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

11.3.5. AIN0

Pin d'ingresso analogico del Computing Unit LV-TTL 3.3V.

11.3.6. AIN1

Pin d'ingresso analogico del Computing Unit LV-TTL 3.3V.

11.3.7. DOUT0

Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

11.3.8. DOUT1

Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

11.3.9. DOUT2

Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

11.3.10. AO0/PWM0

Pin d'uscita del segnale analogico o PWM del Computing Unit LV-TTL 3.3V.

11.3.11. AO1/PWM1

Pin d'uscita del segnale analogico o PWM del Computing Unit LV-TTL 3.3V.

11.3.12. AIN2

Pin d'ingresso analogico del Computing Unit LV-TTL 3.3V.

11.3.13. DIN3

Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

11.3.14. DIN4
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

11.3.15. DIN5
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

11.3.16. DIN6
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

11.3.17. DIN7
Pin d'ingresso digitale del Computing Unit LV-TTL 3.3V.

11.3.18. AIN3
Pin d'ingresso analogico del Computing Unit LV-TTL 3.3V.

11.3.19. DOUT3
Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

11.3.20. DOUT4
Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

11.3.21. DOUT5
Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

11.3.22. DOUT6
Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

11.3.23. DOUT7
Pin d'uscita digitale del Computing Unit LV-TTL 3.3V.

11.3.24. AO2/PWM2
Pin d'uscita del segnale analogico o PWM del Computing Unit LV-TTL 3.3V. Questo pin al momento è di estensione, non risulta connesso.

11.3.25. AO3/PWM3
Pin d'uscita del segnale analogico o PWM del Computing Unit LV-TTL 3.3V. Questo pin al momento è di estensione, non risulta connesso.

11.3.26. UART_A
connection uart pin.

11.3.27. UART_B
connection uart pin.

11.4. DIN replicator
Il modulo DIN replicator possiede tre ingressi e tre uscite. Il modulo protegge il segnale in ingresso da corto circuiti ed eventuali sovracorrenti.

- Il pin P-IN-1 é connesso tramite RES al al pin IN-1, e tramite il Diodo Z a GND.
- Il pin P-IN-2 é connesso tramite RES al al pin IN-2, e tramite il Diodo Z a GND.
- Il pin P-IN-3 é connesso tramite RES al al pin IN-3, e tramite il Diodo Z a GND.

11.4.1. P-IN-1

Pin d'ingresso digitale dell' DIN replicator, connesso al pin IN2-1 del Dimostratore Multipiattaforma.

11.4.2. IN-3

Pin dell'uscita digitale del DIN replicator.

11.4.3. IN-2

Pin dell'uscita digitale del DIN replicator.

11.4.4. IN-1

Pin dell'uscita digitale del DIN replicator.

11.4.5. P-IN-2

Pin d'ingresso digitale dell' DIN replicator, connesso al pin IN2-2 del Dimostratore Multipiattaforma.

11.4.6. P-IN-3

Pin d'ingresso digitale del DIN replicator, connesso al pin IN2-3 del Dimostratore Multipiattaforma.

11.4.7. GND

Pin del DIN replicator relativo alla massa di riferimento connesso al pin GND del Dimostratore Multipiattaforma.

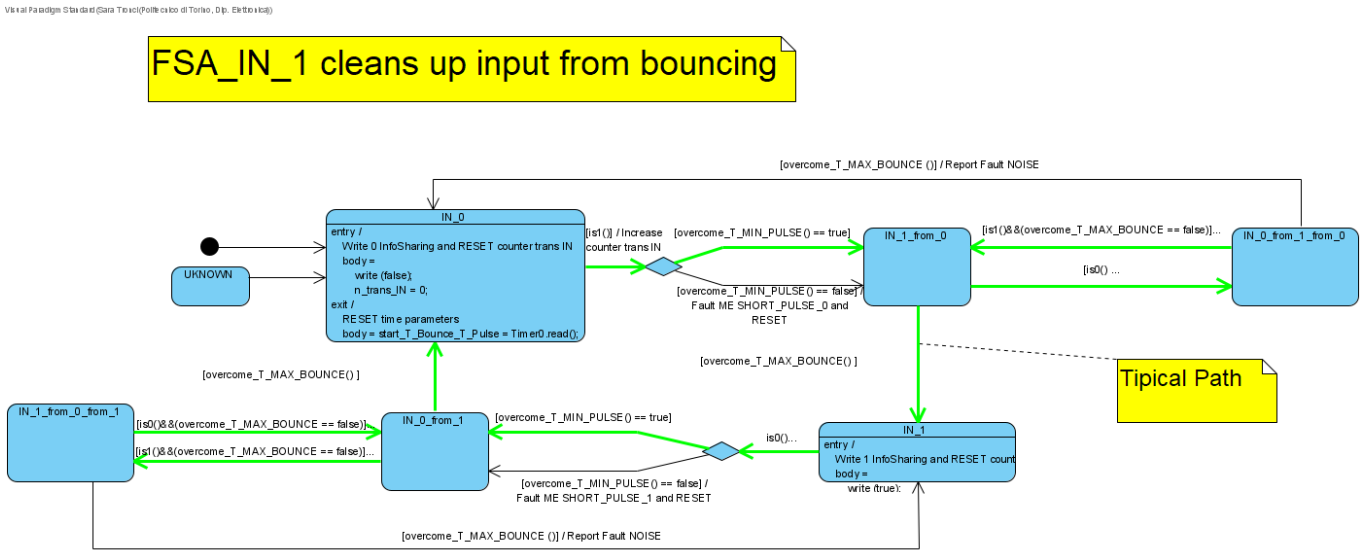
11.4.8. A

Pin d'ingresso della (model element not found) del (model element not found).

11.4.9. B

Pin d'iuscita della RES del (model element not found).

12. Evidenzia Guasto Ingresso - State Machine

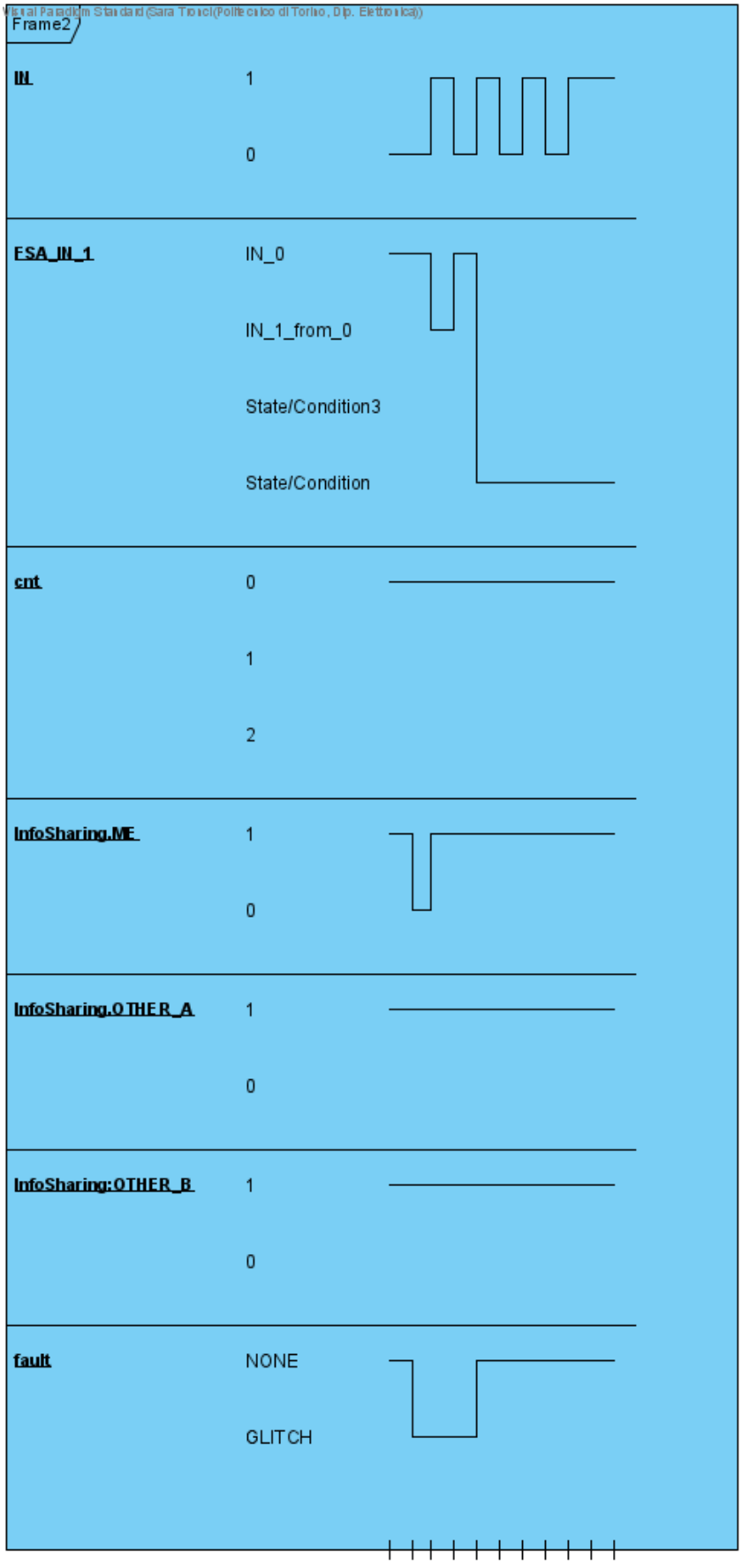


12.1. States

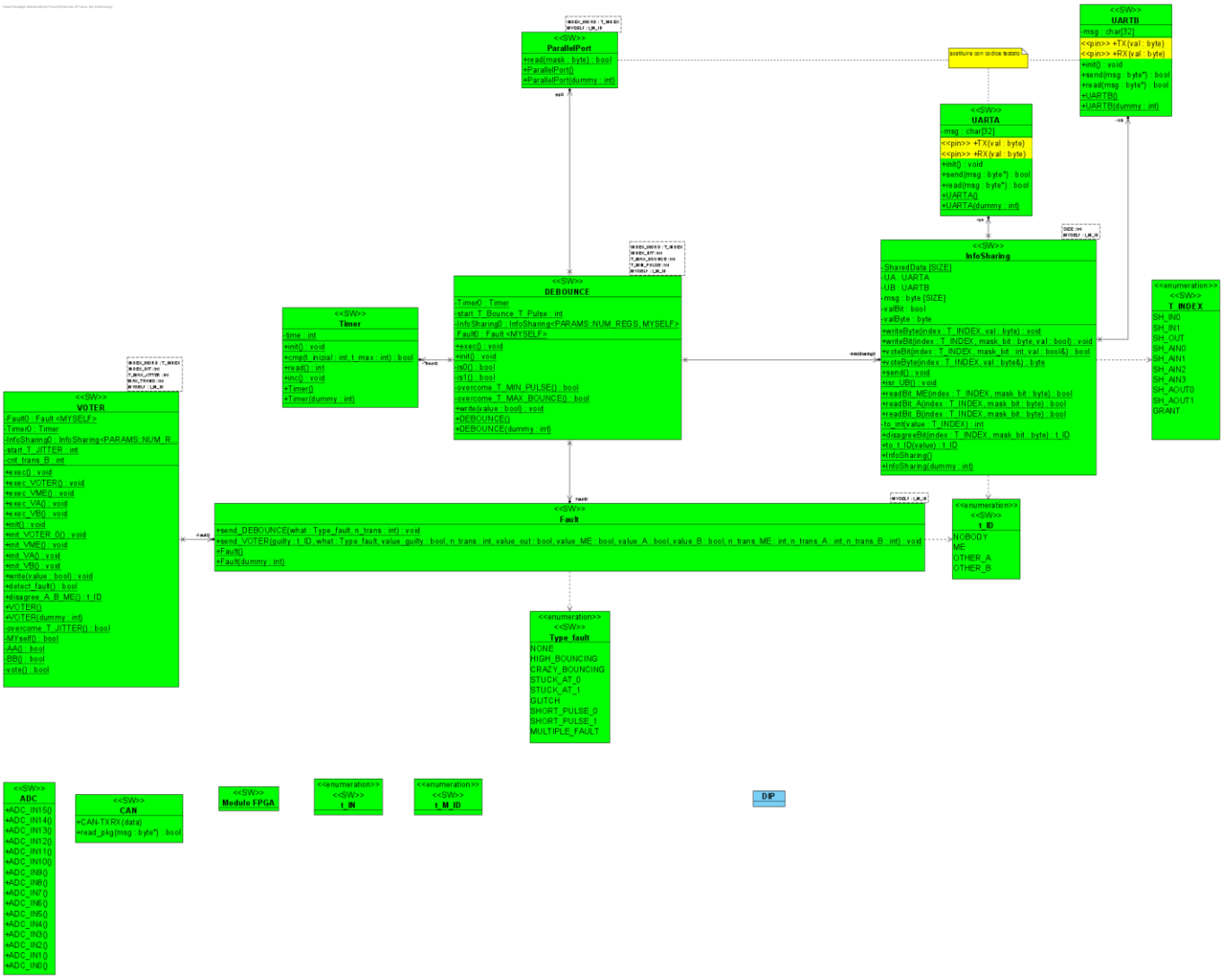
Name	Description
UKNOWN	
IN_1_from_0_from_1	The input IN making a transition from 0 to 1 and the third last value of IN was 1. I leave the state if IN occurs a transition to 1 within the allowed T_MAX_BOUNCE, if it doesn't into T_MAX_BOUNCE, you leave the state, to go to IN_0 but you send a fault.
IN_0_from_1	The input IN has just made a transition from 1 to 0. You exit of this state, if IN == 0, or if you exceeded the T_MAX_BOUNCE.
IN_0_from_1_from_0	The input IN making a transition from 1 to 0 and the third last value of IN was 0. I leave the state if IN occurs a transition to 1 within the allowed T_MAX_BOUNCE, if it doesn't into T_MAX_BOUNCE, you leave the state, to go to IN_0 but you send a fault.
IN_1_from_0	The input IN has just made a transition from 0 to 1. You exit of this state, if IN == 0, or if you exceeded the T_MAX_BOUNCE.
IN_1	In this state the input bit INDEX_BIT of the DEBOUNCE is 1. voteBit(index: T_INDEX, mask_bit: int, val: bool&): bool compares the values of the same bits DEBOUNCE of INDEX_BIT between the three cores. Leave the state if there are at least two cores that have the same value or if the INDEX_BIT makes a transaction at 0.
IN_0	In this state the input bit INDEX_BIT of the DEBOUNCE is 0.

	voteBit(index: T_INDEX, mask_bit: int, val: bool&): bool compares the values of the same bits DEBOUNCE of INDEX_BIT between the three cores. Leave the state if there are at least two cores that have the same value or if the INDEX_BIT makes a transaction at 1. You leave IN_0 if IN make at transaction at 1 and you go to state IN_1_from_0 You leave IN_0 if IN make a transaction at 1, or

13. Evidenzia Guasto Ingresso - Timing



14. Input Cleaner



14.1. t_IN

14.1.1. Enumeration Literals

Name	Description
IN0	
IN1	
IN2	
IN3	
IN4	
IN5	
IN6	
IN7	
AIN0	

AIN1	
AIN2	
AIN3	

14.2. Modulo FPGA

14.2.1. DOUT

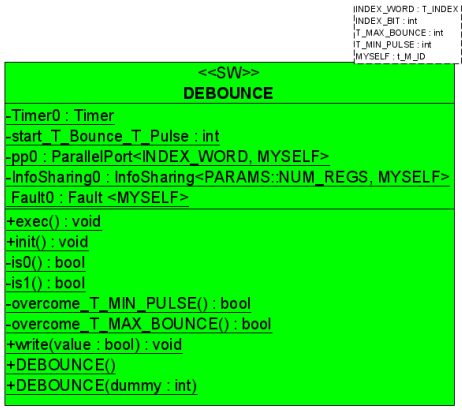
Digital signals maching Modulo FPGA's pins DOUT0, DOUT1, DOUT2, DOUT3, DOUT4, DOUT5, DOUT6 and DOUT7.

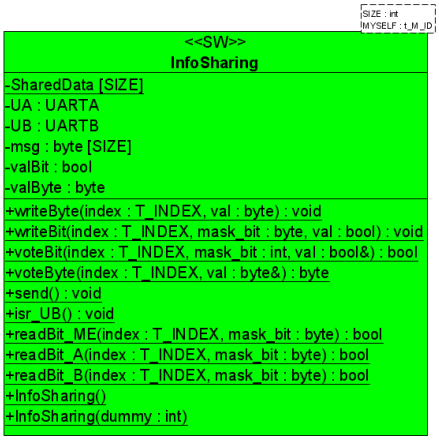
14.3. t_M_ID

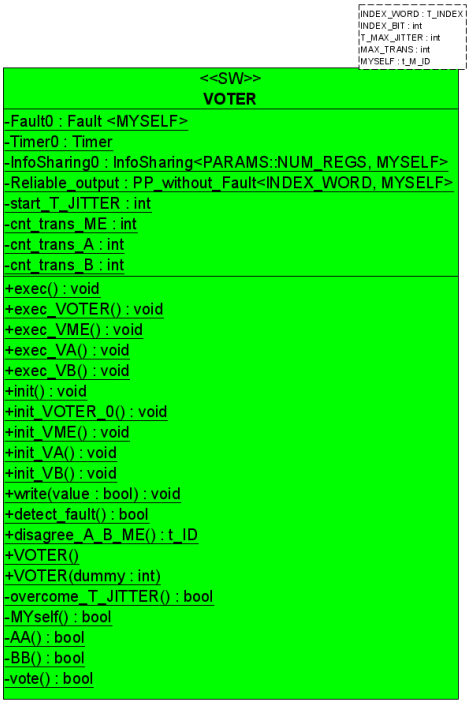
14.3.1. Enumeration Literals

Name	Description
M1	
M2	
M3	

15. Input Cleaner - SW section







15.1. VOTER

La FSA_IN_2 è una automa, che descrive come il microcontrollore ME, interagisce con gli altri due OTHER_A e OTHER_B in funzione delle transizioni del bit d'ingresso scritto InfoSharing0: InfoSharing.

Si considera quindi l'ingresso della FSA_IN_2, libero da bouncing.

La FSA, ma la macchina è in grado di gestire evventuali fault:

1

FSA_2 è un automa mirato all'acquisizione di un unico segnale elettromeccanico disponibile in tre copie provenienti da altrettanti trasduttori distinti, precedentemente ripuliti da rimbalzi. L'automa valuta affidabilmente il valore del segnale anche nel caso che al massimo uno dei trasduttori (o catene di acquisizione) soffra di un qualsivoglia guasto.

La FSA_IN_2 è in grado di segnalare eventuali errori.

15.1.1. Attributes

Signature	Description

-dummy1 : t_ID	
-dummy2 : Type_fault	
-Fault0 : Fault	
-Timer0 : Timer	
-InfoSharing0 : InfoSharing	
-Reliable_output : PP_without_Fault	
-start_T_JITTER : int	
-cnt_trans_ME : int	
-cnt_trans_A : int	Variable to count the number of transactions sent by the INDEX_BIT input bits of OTHER_A.
-cnt_trans_B : int	Variable to count the number of transactions sent by the INDEX_BIT input bits of OTHER_B.
-Reporter0 : Reporter	

15.1.2. detect_fault

individua l'errore, esaminando se sono tutti d'accordo
restituisce 1 se è stato rilevato un fault, e 0 in caso contrario

15.1.3. disagree_A_B_ME

Funzione che esamina i valori di A B e ME e restituisce NOBODY se sono tutti d'accordo, se invece c'è un disaccordo, restituisce il t_ID di chi è in disaccordo.

15.1.4. MYself

It returns the value of the input bit position INDEX_BIT of the (model element not found) microcontroller, which is carrying out the data processing.

15.1.5. AA

It returns the value of the input bit position INDEX_BIT of the OTHER_A microcontroller, which was read by InfoSharing0: InfoSharing.

15.1.6. BB

It returns the value of the input bit position INDEX_BIT of the OTHER_B microcontroller, which was read by InfoSharing0: InfoSharing.

15.1.7. vote

It returns the voting bits value of the three microcontroller on the bit input SH_IN0 to INDEX_BIT position.

15.2. DEBOUNCE

15.2.1. Attributes

Signature	Description
-Timer0 : Timer	

-start_T_Bounce_T_Pulse : int	Start time of bouncing signal, and time of pulse
-pp0 : ParallelPort	
-InfoSharing0 : InfoSharing	
Fault0 : Fault	
-n_trans_IN : int	valore delle transizioni effettuate dall'ingresso tra una scrittura su infoSharing e l'altra
-Reporter0 : Reporter	

15.3. InputCleaner8

La classe InputCleaner8 iceve in ingresso 8 inputs digitali on/off.
Libera singolarmente gli input dal bounce. e memorizza internamente il dato.
bouncing -> una serie di transizioni del segnale digitale da on a off, e viceversa per un tempo inferiore a T_MAX_BOUNCE.

Trasmette i valori debounciati di ciascun ingresso, tramite UART, a due inputcleaner8 gemelli , interni a due moduli esterni che ricevono copia virtualmente identica, degli stessi 8 input.

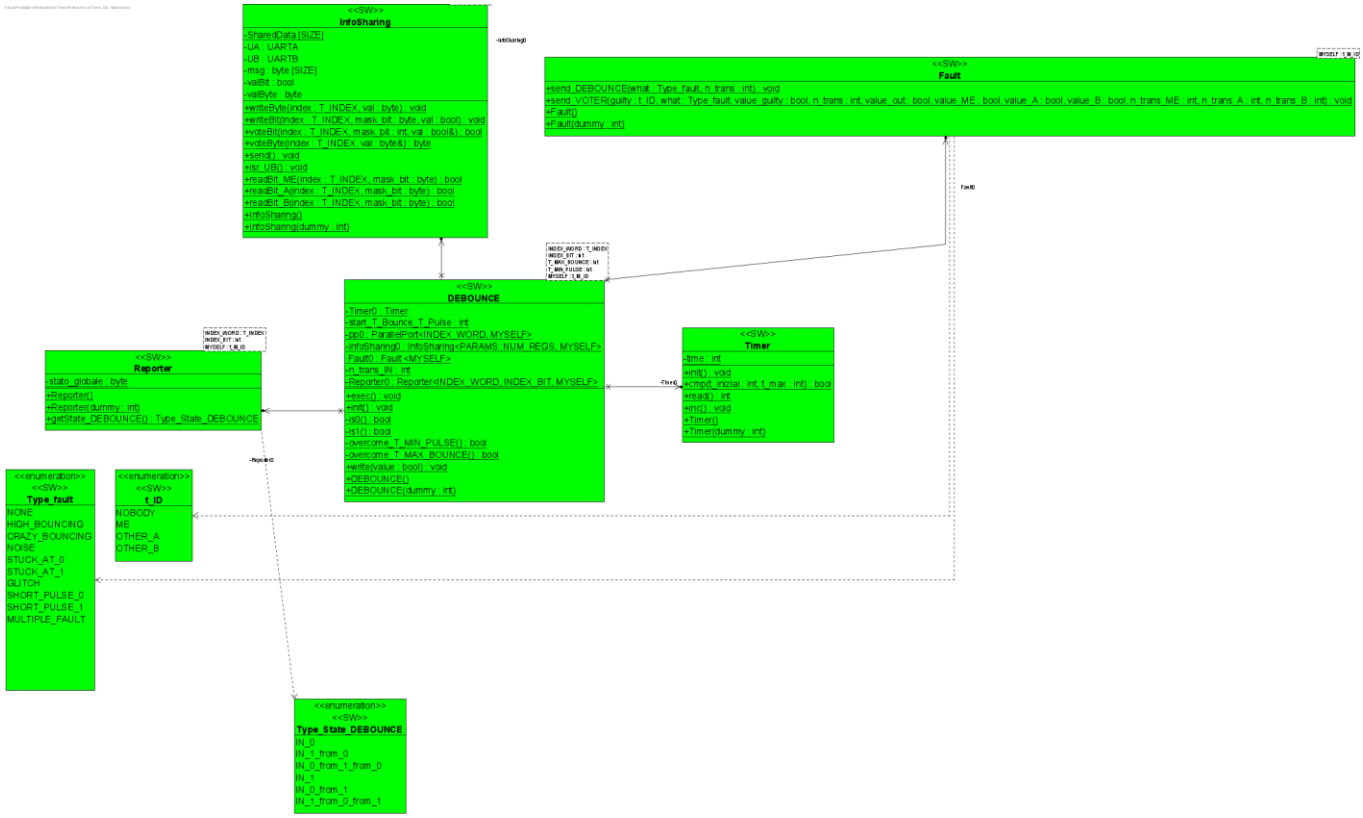
Riceve copia degli 8 input dai due moduli di cui sopra.
Per ogni input avviene una votazione, basata sulla maggioranza, che dettermina il valore da mandare in uscita, che è considerato il valore corretto.
Inoltre esamina se se si sono verificati alcuni tipi di fault.

15.3.1. Attributes

Signature	Description
D0 : DEBOUNCE	
D1 : DEBOUNCE	
D2 : DEBOUNCE	
D3 : DEBOUNCE	
D4 : DEBOUNCE	
D5 : DEBOUNCE	
D6 : DEBOUNCE	
D7 : DEBOUNCE	
-V0 : VOTER	
-V1 : VOTER	
-V2 : VOTER	
-V3 : VOTER	
-V4 : VOTER	

-V5 : VOTER	
-V6 : VOTER	
-V7 : VOTER	
-fault0 : Fault	

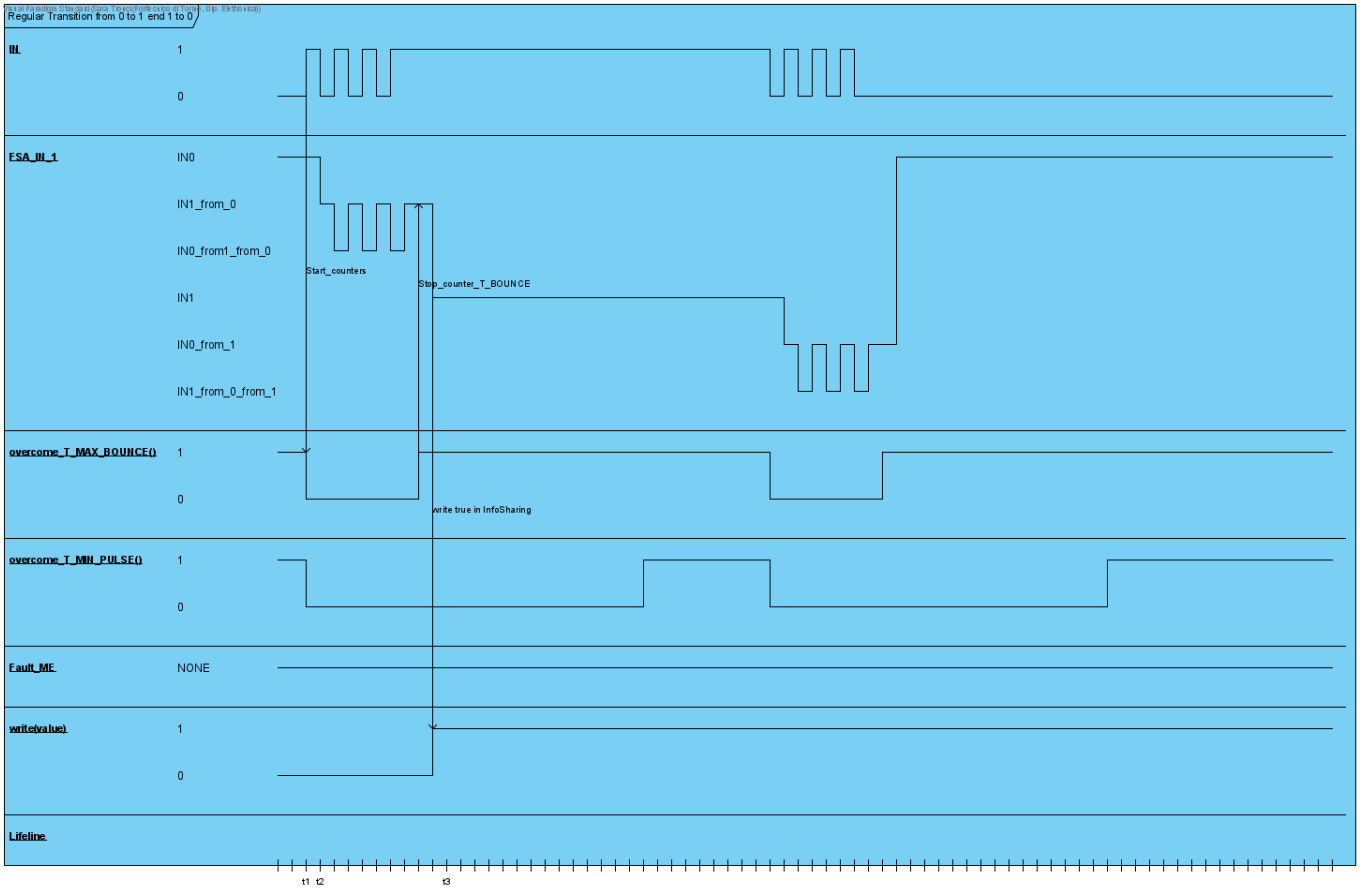
16. DEBOUNCE FSA_IN_1



17. Diagramma temporale FSA_IN_1 - esempio 1

Il seguente diagramma temporale mostra un esempio del funzionamento FSA_IN_1 tramite un diagramma temporale.

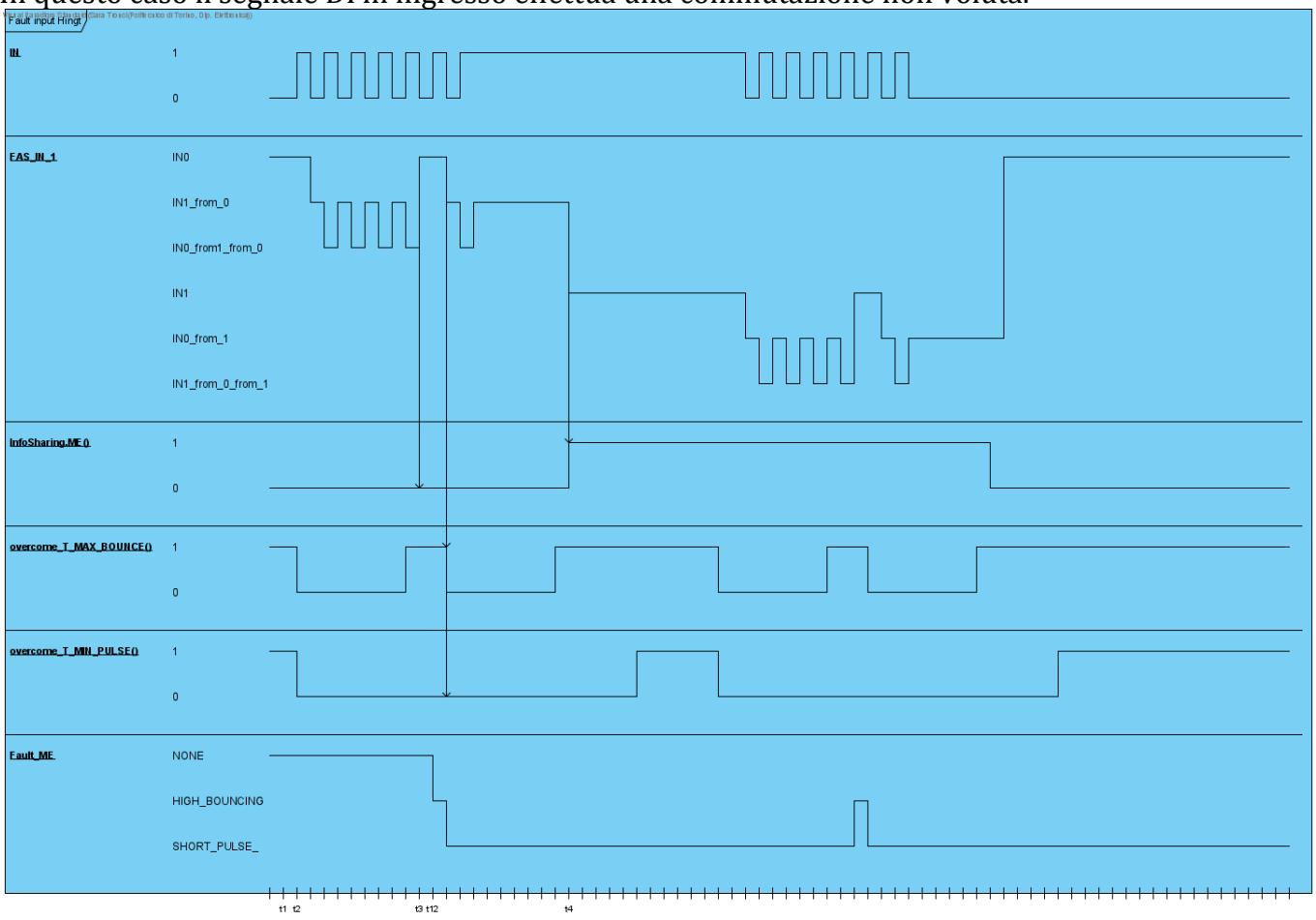
In questo caso il segnale DI in ingresso effettua una commutazione da 1 a 0 e poi da 0 a 1 senza fault.



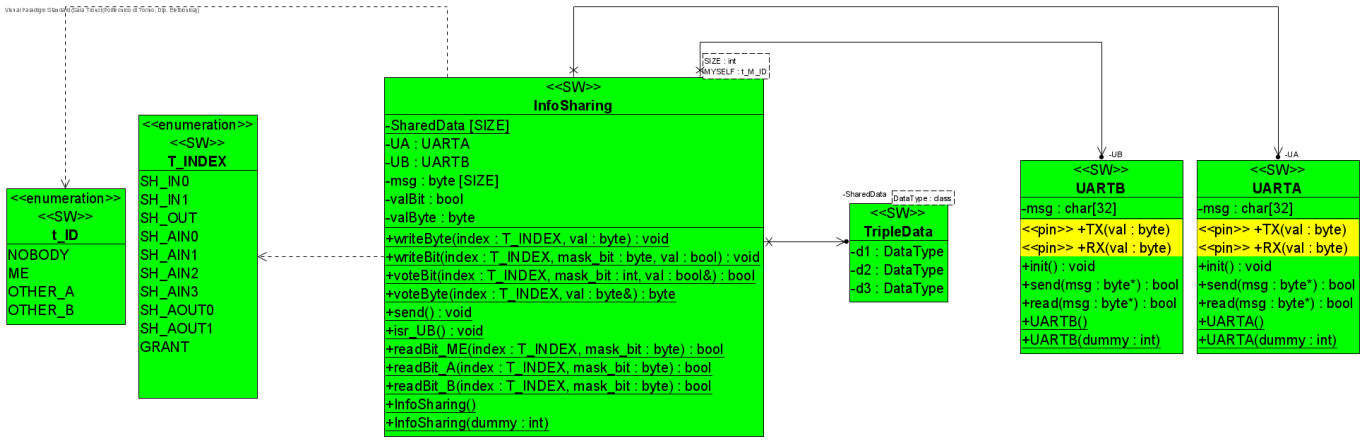
18. Diagramma Temporale FSA_IN_1 - Esempio 2

Il seguente diagramma temporale mostra un esempio del funzionamento FSA_IN_1 tramite un diagramma temporale.

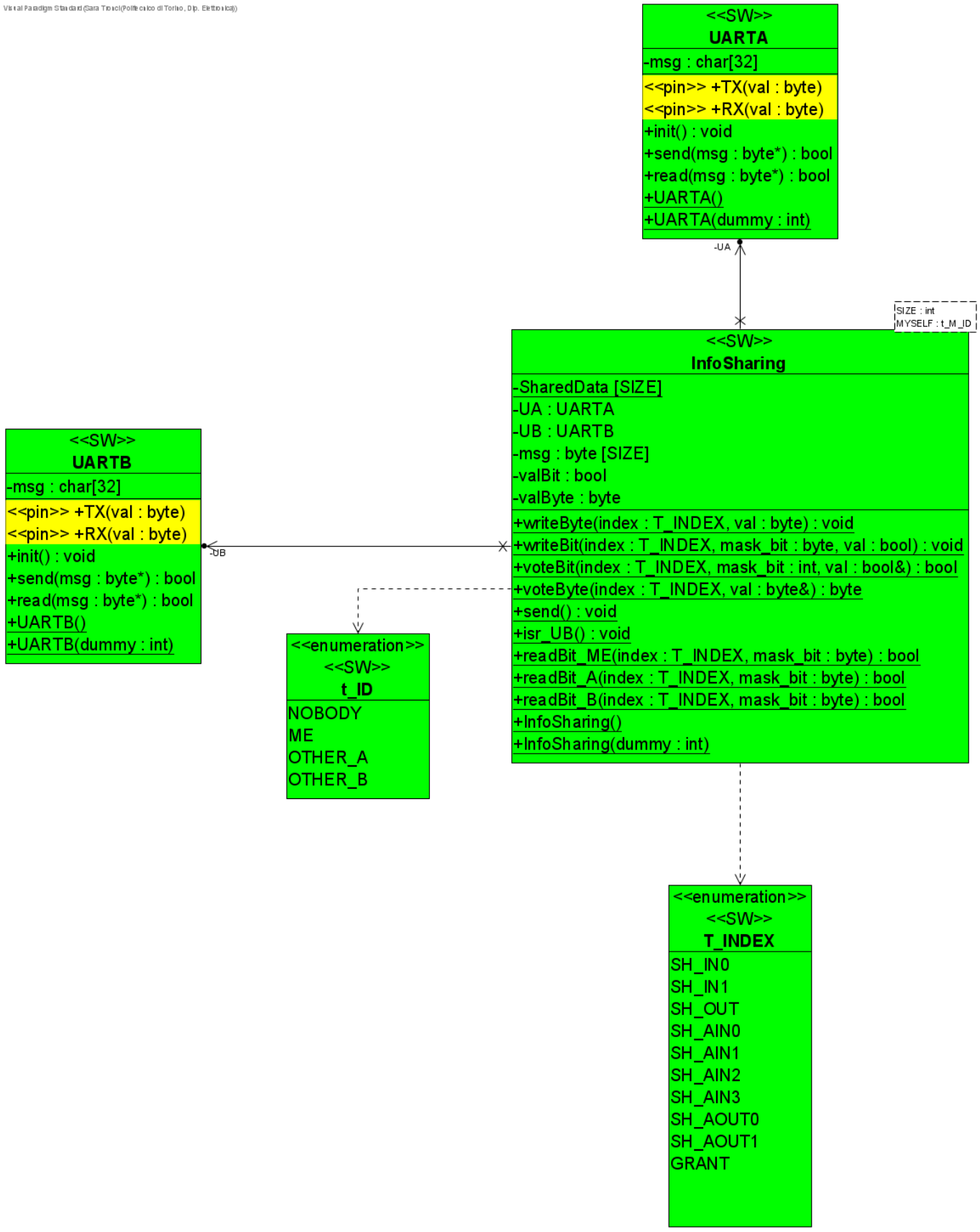
In questo caso il segnale DI in ingresso effettua una commutazione non voluta.



19. Descrizione di InfoSharing



20. InfoSharing



20.1. UARTB

20.1.1. Attributes

Signature	Description
-msg : char[32]	

20.2. UARTA

20.2.1. Attributes

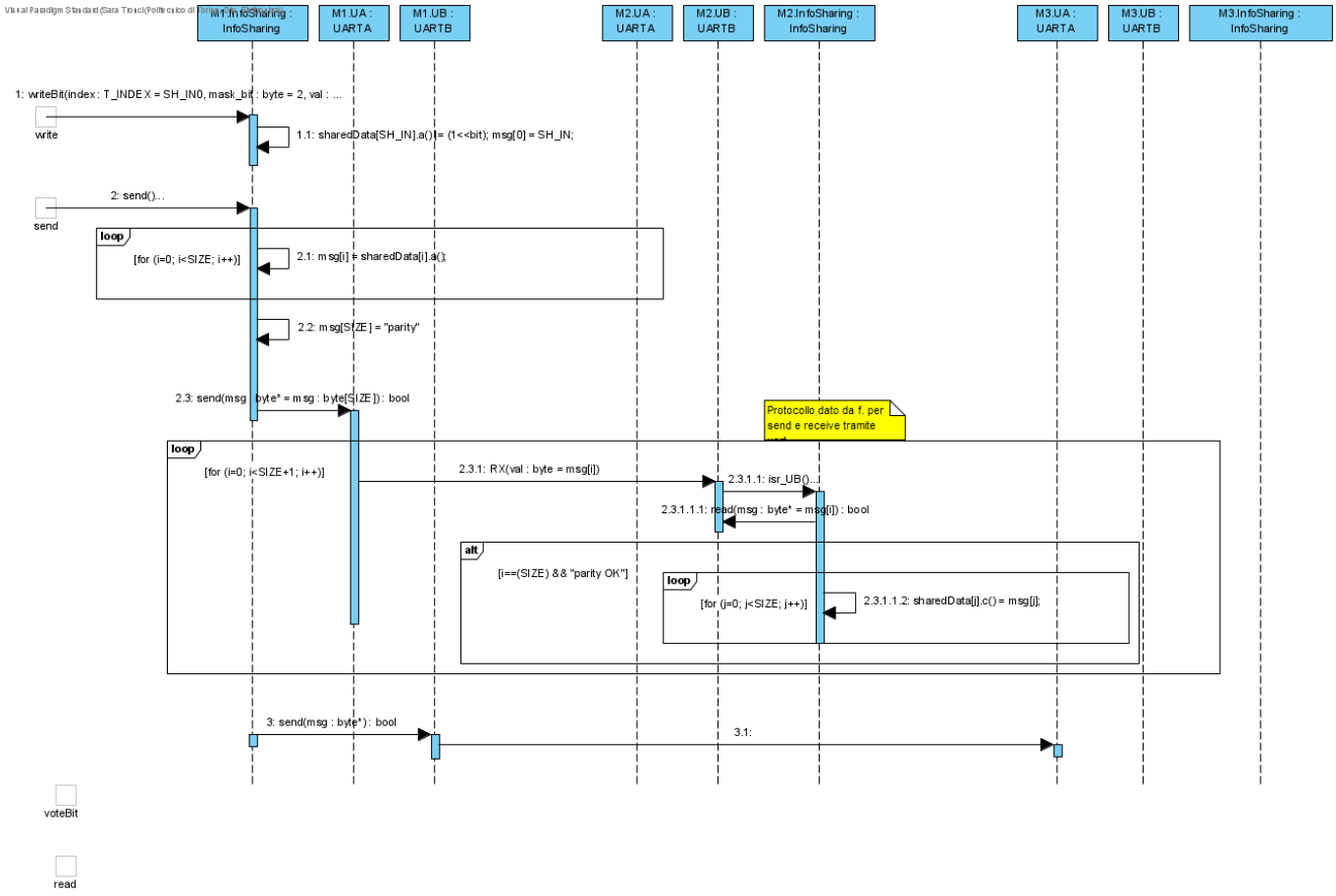
Signature	Description
-msg : char[32]	

20.3. T_INDEX

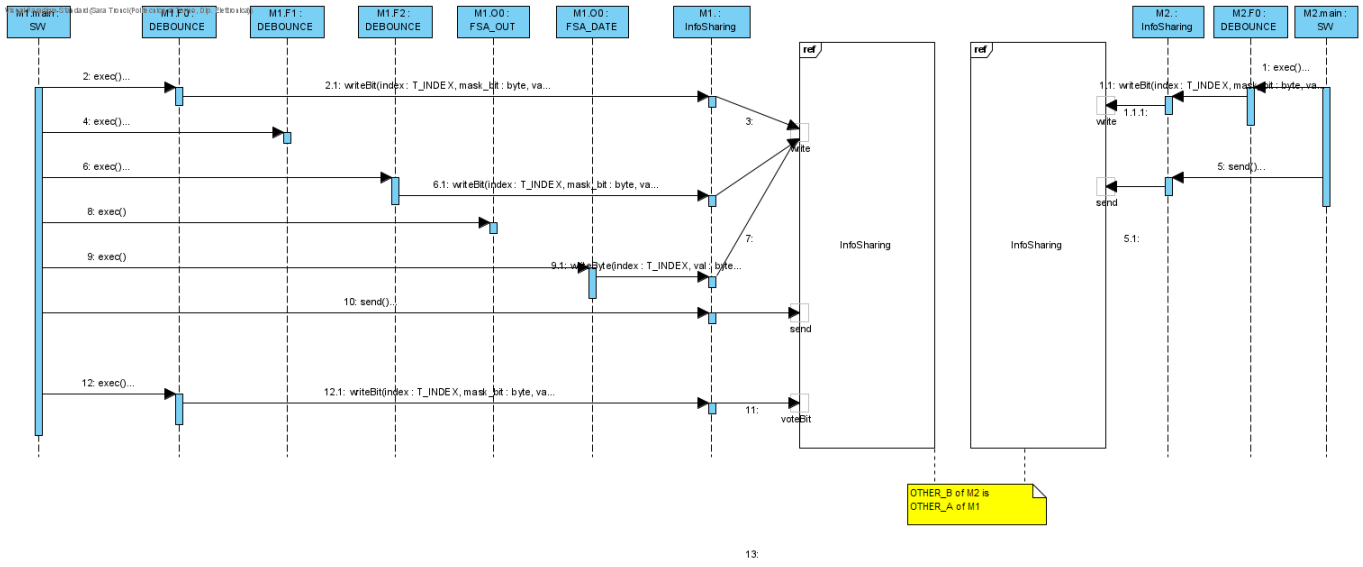
20.3.1. Enumeration Literals

Name	Description
SH_IN0	Specify the digital input bit.
SH_IN1	Specify the digital input bit.
SH_OUT	Specify the digital output bit.
SH_AIN0	Specify the signal analog input.
SH_AIN1	Specify the signal analog input.
SH_AIN2	Specify the signal analog input.
SH_AIN3	Specify the signal analog input.
SH_AOUT0	Specify the signal analog output.
SH_AOUT1	Specify the signal analog input.
GRANT	

21. InfoSharing



22. InfoSharing Usage

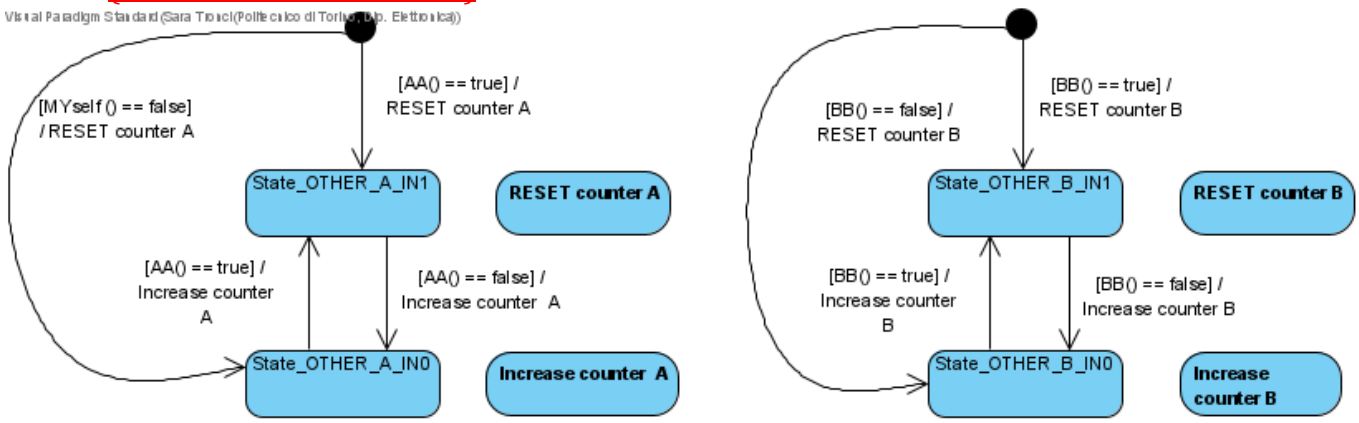


23. Substate ME_at_1 OTHER_A and OTHER_B

Substate diagram of the (model element not found), which contains two automatons, one of the OTHER_A, one of OTHER_B, that work in parallel.

The input signals INDEX_BIT of OTHER_A and OTHER_B and are free from bouncing because outputs of FSA_IN_1.

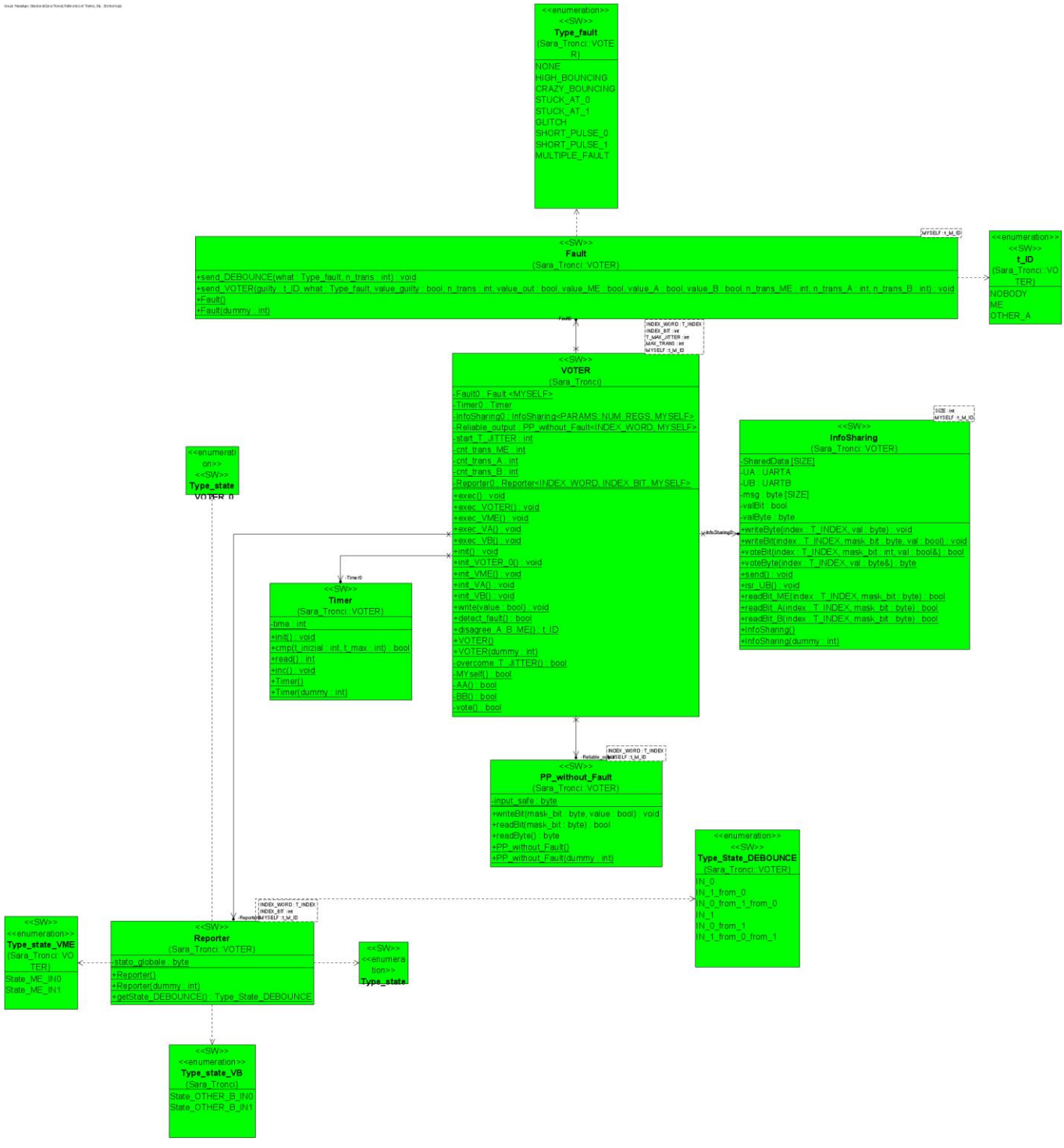
Whenever input of OTHER_A and OTHER_B make a transition from 0 to 1 and conversely, the machine increment their respective counter cnt_trans_A: int and cnt_trans_B: int until one exit from father state (model element not found).



23.1. States

Name	Description
State_OTHER_B_IN0	OTHER_B input INDEX_BIT is at 0. You leave this state when INDEX_BIT go at 1.
State_OTHER_B_IN1	OTHER_B input INDEX_BIT is at 1. You leave this state when INDEX_BIT go at 0.
State_OTHER_A_IN0	OTHER_A input INDEX_BIT is at 0. You leave this state when INDEX_BIT go at 1.
State_OTHER_A_IN1	OTHER_A input INDEX_BIT is at 1. You leave this state when INDEX_BIT go at 0.

24. VOTER FSA_IN_2



24.1. Type_State_DEBOUNCE

24.1.1. Enumeration Literals

Name	Description
IN_0	
IN_1_from_0	

IN_0_from_1_from_0	
IN_1	
IN_0_from_1	
IN_1_from_0_from_1	

24.2. Type_state_VME

24.2.1. Enumeration Literals

Name	Description
State_ME_IN0	
State_ME_IN1	

24.3. Type_state_VA

24.3.1. Enumeration Literals

Name	Description
State_OTHER_A_IN0	
State_OTHER_A_IN1	

24.4. Type_state_VB

24.4.1. Enumeration Literals

Name	Description
State_OTHER_B_IN0	
State_OTHER_B_IN1	

24.5. Type_state_VOTER_0

24.5.1. Enumeration Literals

Name	Description
OUT_0	
OUT_1	
VOTING	

24.6. Reporter

24.6.1. Attributes

Signature	Description
-stato_globale : byte	Formato V0 V0 VME VA VB D D D V0 riservato alla FSA principale del VOTER VME riservato alla FSA contatore degli stati ME VA riservato alla FSA contatore degli stati A VB riservato alla FSA contatore degli stati B D riservato aalla FSA principale di DEBOUNCE

24.6.2. getState_DEBOUNCE

Maschera gli stati che riguardano DEBOUNCE

24.7. Type_fault

La classe enumerativa

24.7.1. Enumeration Literals

Name	Description
NONE	
HIGH_BOUNCING	Caso in cui c'è un errore ma non sono riuscita a identificarlo nella funzione detect_fault()
CRAZY_BOUNCING	Caso in cui c'è un errore ma non sono riuscita a identificarlo nella funzione detect_fault()
NOISE	questo è il caso in cui ho ad esempio un'interferenza in ingresso. Il segnale fa una transizione, ma all'interno di T_MAX_BOUNCE ritorna al vqlore che aveva all'inizio. #CercaUnNomeMigliore
STUCK_AT_0	Indicates that ME has an input SH_IN0 stack at 0.
STUCK_AT_1	Indicates that ME has an input SH_IN0 stack at 1.
GLITCH	Indico in generale una generale una transizione spuria, il tipo di glitch si distingue in base al numero di transizioni che si verificano.
SHORT_PULSE_0	

SHORT_PULSE_1	
MULTIPLE_FAULT	Il guasto si è verificato su più di una componente... non ha senso identificarlo come procedo? Fornisco comunque un uscita OUT ma non è attendibile.

24.8. t_ID

24.8.1. Enumeration Literals

Name	Description
NOBODY	Più di una componente è rotta
ME	
OTHER_A	
OTHER_B	

24.9. PP_without_Fault

24.9.1. Attributes

Signature	Description
-input_safe : byte	Variable contains a input byte free by faults.

24.9.2. writeBit

Write value on the bit value of the bit position variable input_safe

24.10. Fault

24.10.1. send_DEBOUNCE

Sends a message that contains the Type_fault detected.

Devo mandare anche l'identificativo MYSELF quindi non serve che glielo passi come parametro

24.10.2. send_VOTER

Sends a message that contains the Type_fault detected.

24.11. InfoSharing

This is a communication infrastructure to help sharing the info among processors in a triple-redundant set of processors.

The systems is supposed to the made of three processors, called respectively:

- ME, that is, the processor on which the actual instance of the InfoSharing is running;

- OTHER_A; that is, the processor connected to ME via the (model element not found) peripheral;
- OTHER_B; that is, the processor connected to ME via the UB: UARTB peripheral;

This class stores three replicas of a set of critical data (SIZE bytes) into attribute SharedData, built using a TripleData structure.

The three replicas of each byte built inside each element of SharedData are as follows:

1. a(): DataType& stores the data of ME
2. b(): DataType& stores the virtually identical data periodically received from OTHER_A
3. c(): DataType& stores the virtually identical data periodically received from OTHER_B

This class has operations to:

1. write data of ME into shared storege, by means of writeByte(index: T_INDEX, val: byte): void;
2. read data from either of three processors Computing Unit;
3. vote data to return the most likely value of each bit (respectively, byte) by means of voteBit(index: T_INDEX, mask_bit: int, val: bool&): bool (respectively, voteByte(index: T_INDEX, val: byte&): byte);
4. send ME's data to both OTHER_A and OTHER_B, by means of send(): void.

24.11.1. Attributes

Signature	Description
-SharedData[SIZE]	
-UA : UARTA	
-UB : UARTB	
-msg : byte[SIZE]	
-valBit : bool	
-valByte : byte	

24.11.2. writeByte

Writes the value val into the index-th location of ME's replica in SharedData

24.11.3. writeBit

Writes the value val into the index-th location of ME's replica in SharedData

24.11.4. voteBit

Votes the three local copies of ME, OTHER_A, OTHER_B in bit mask_bit of location index.
Returns the most likely value (two out of three) into val.
Returns true if all three copies match; false otherwise.

24.11.5. send

Sends ME's replica of SharedData to OTHER_A (via (model element not found)) and to OTHER_B (via UB: UARTB)

24.12. Timer

Timer class contains all information related to the Computing Unit timer functions.

24.12.1. Attributes

Signature	Description
-time : int	

24.12.2. init

Configures and starts the Timer.

24.12.3. cmp

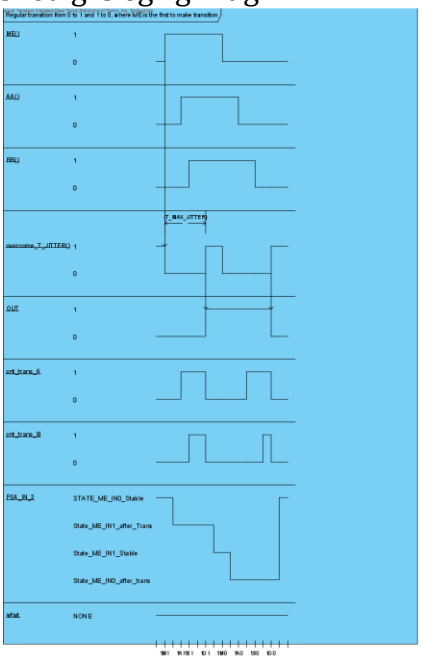
Compares the difference between the current value of time provided by the Timer and the t_inizial with the t_max, considering also overflow. It returning 1 if greater than or equal, 0 if different.

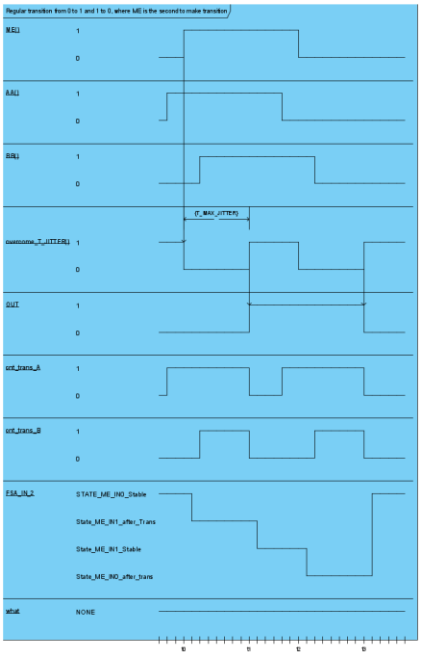
24.12.4. read

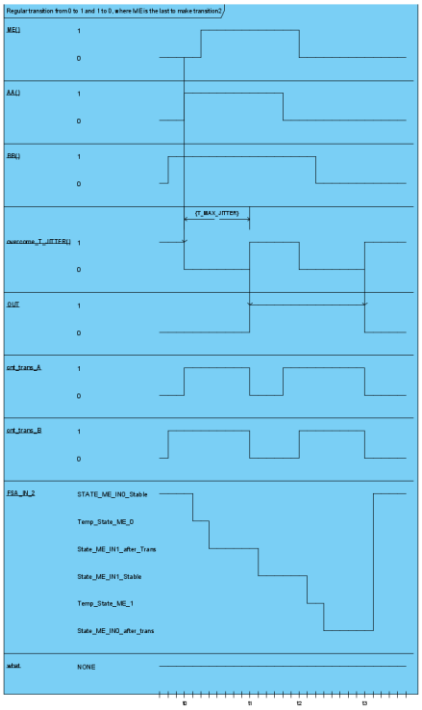
It provides the actual time measured by the Timer

25. FSA_IN_2 Timing Diagram

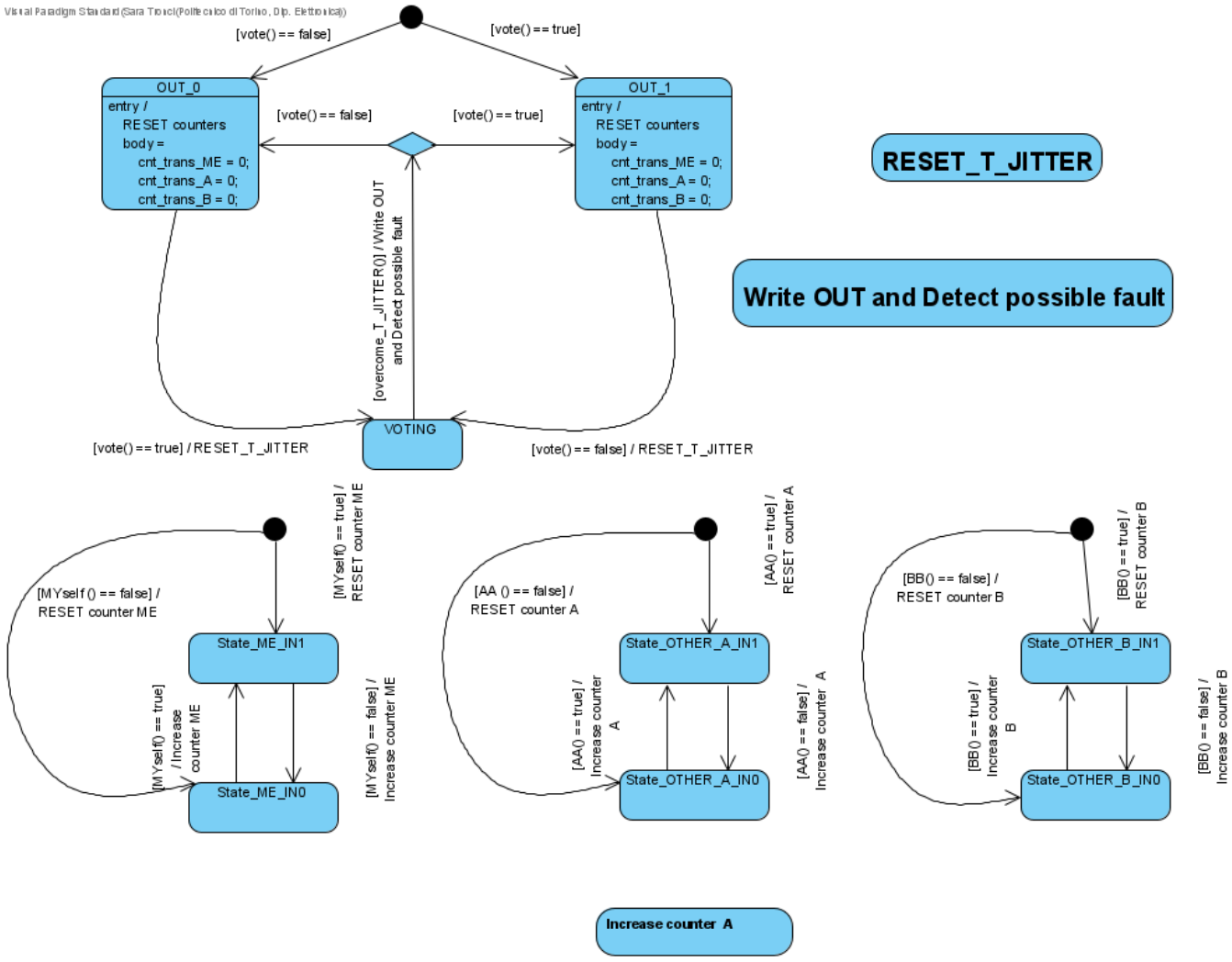
s<ed gzs egz gzr dg







26. FSA_IN_2 State Machine Diagram_out_based



26.1. States

Name	Description
VOTING	
OUT_1	l'uscita è a 0
OUT_0	l'uscita è a 0
State_ME_IN0	OTHER_A input INDEX_BIT is at 0. You leave this state when INDEX_BIT go at 1.
State_ME_IN1	OTHER_A input INDEX_BIT is at 1. You leave this state when INDEX_BIT go at 0.
State_OTHER_B_IN0	OTHER_B input INDEX_BIT is at 0. You leave this state when INDEX_BIT go at 1.

State_OTHER_B_IN1	OTHER_B input INDEX_BIT is at 1. You leave this state when INDEX_BIT go at 0.
State_OTHER_A_IN0	OTHER_A input INDEX_BIT is at 0. You leave this state when INDEX_BIT go at 1.
State_OTHER_A_IN1	OTHER_A input INDEX_BIT is at 1. You leave this state when INDEX_BIT go at 0.

26.2. Write OUT and Detect possible fault

26.2.1. SELECT Parameters of A

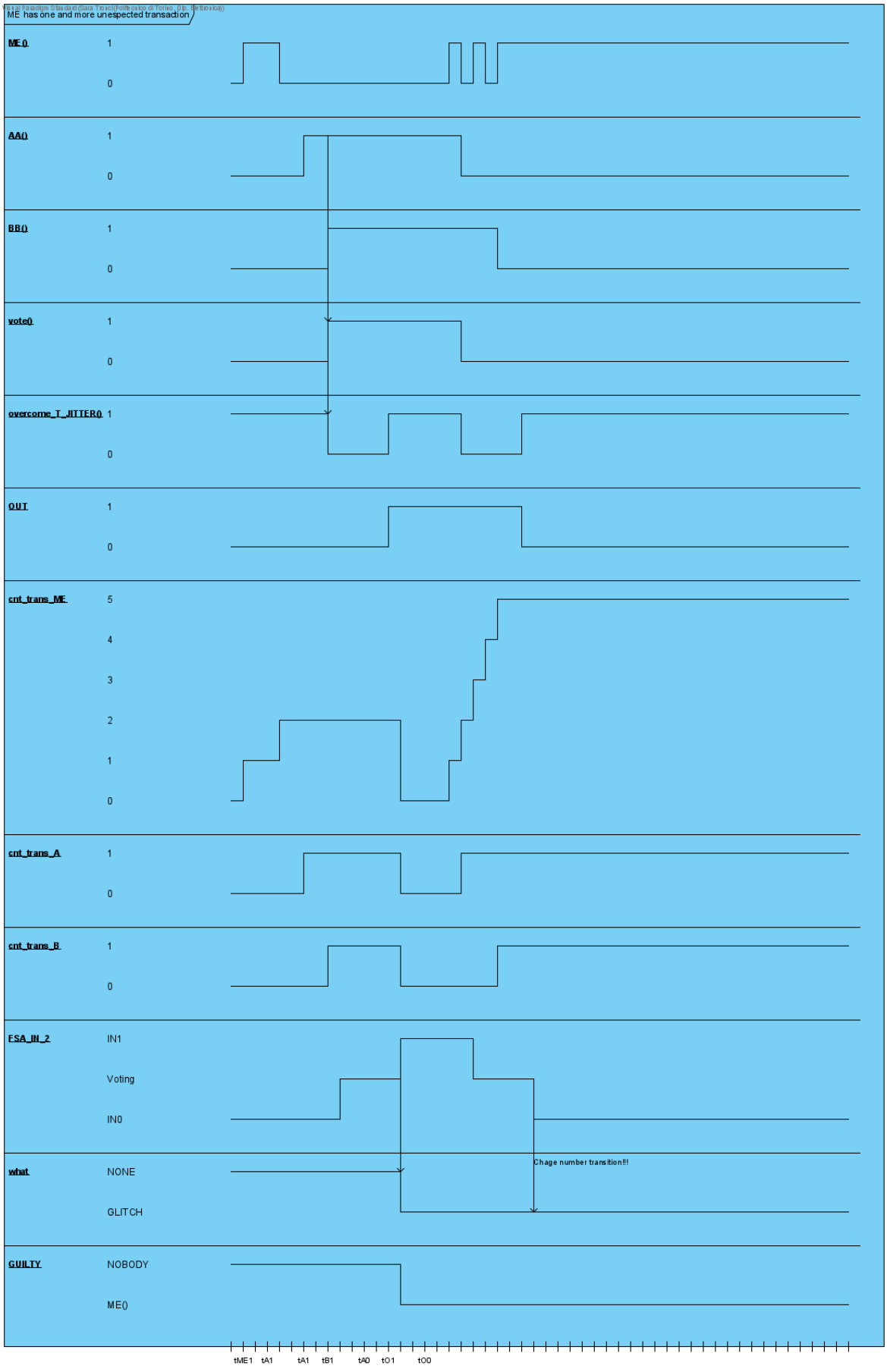
counter = cnt_trans_A;
value_guilty = A();
guilty = OTHER_A;

26.3. RESET_T_JITTER

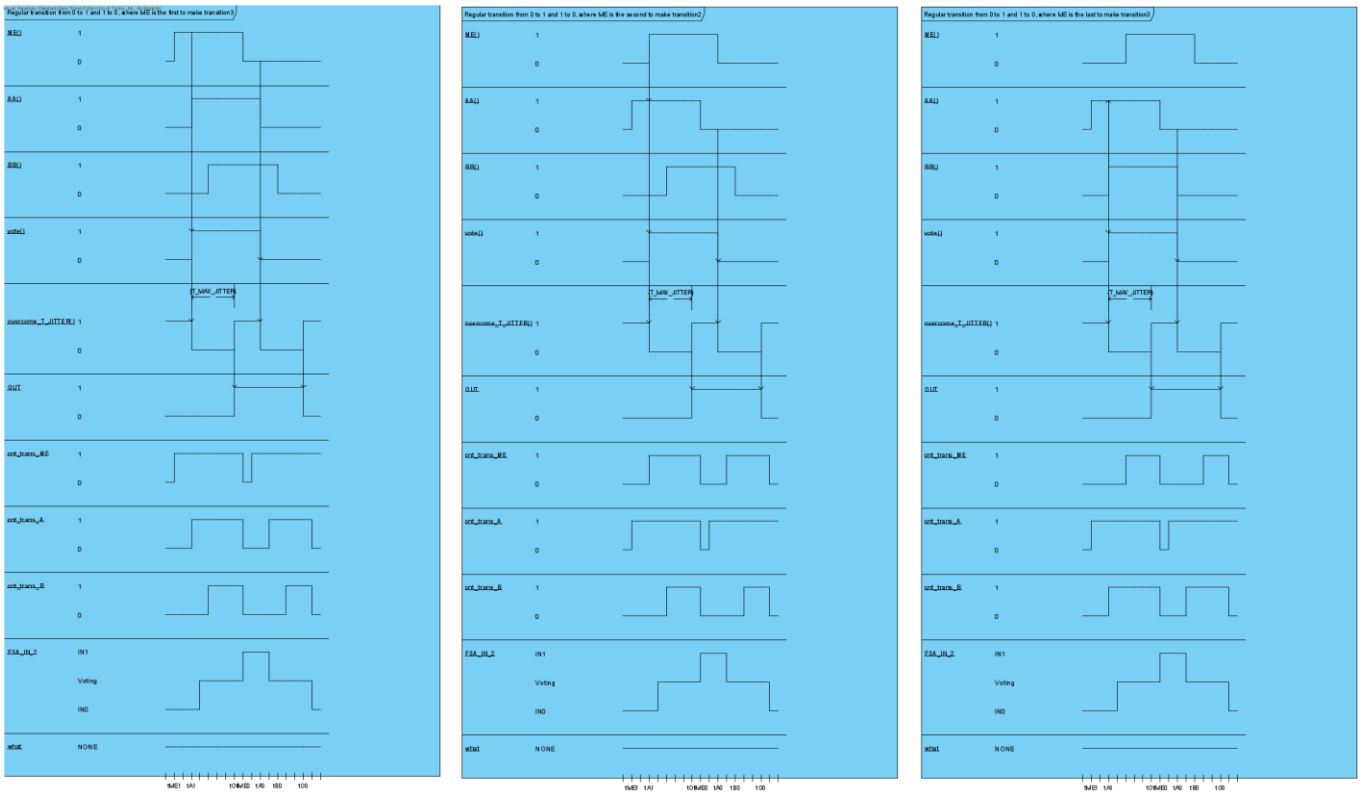
In questo caso resetto solo T_JITTER

27. out_based FSA_IN_2 Timing Diagram ME broke down

Vengono esaminati tutti i casi in cui ME una serie di transizioni non prevviste, durente T_JITTER, e gli altri due ingressi non hanno transizioni

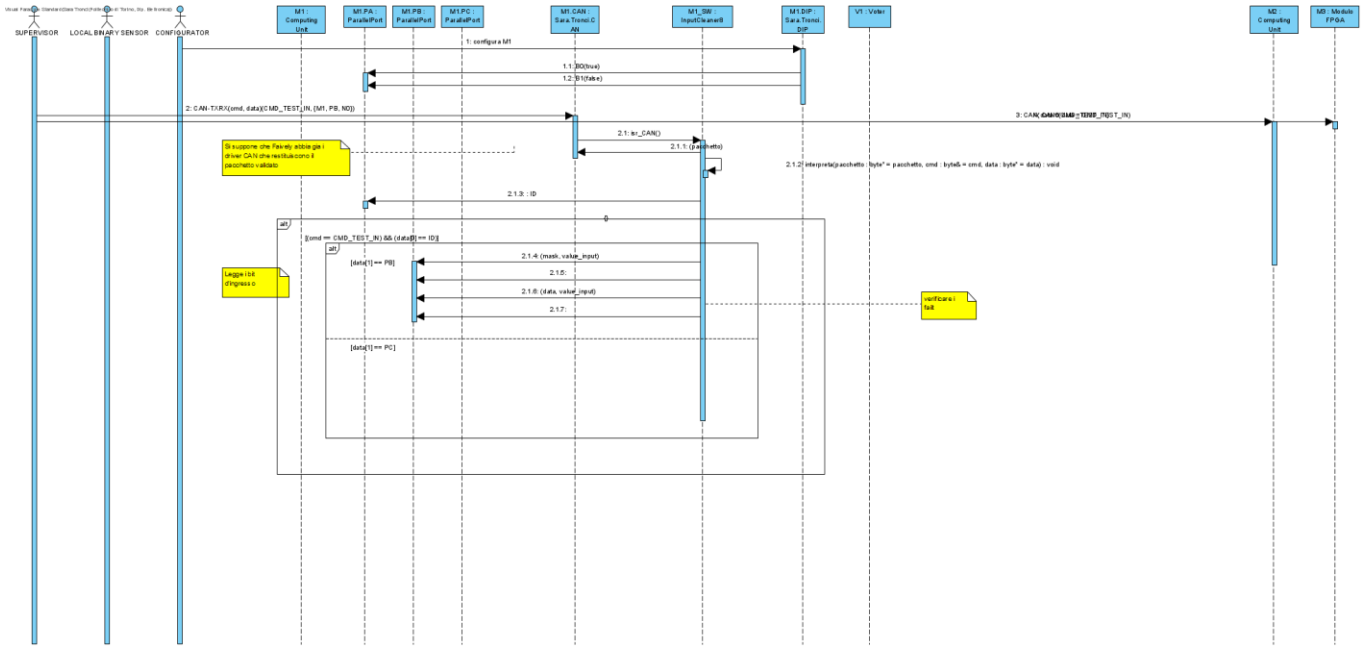

ME has one and more unespected transaction
ME()
AA()
BB()
vote()
overcome_T_JITTER()
OUT
cnt_trans_ME
cnt_trans_A
cnt_trans_B
FSA_IN_2
IN1
Voting
IN0
what
NONE
GLITCH
Chage number transition!!!
GUILTY
NOBODY
ME()
tME1 tA1 tA1 tB1 tA0 tO1 tO0

28. out_based FSA_IN_2 Timing diagram

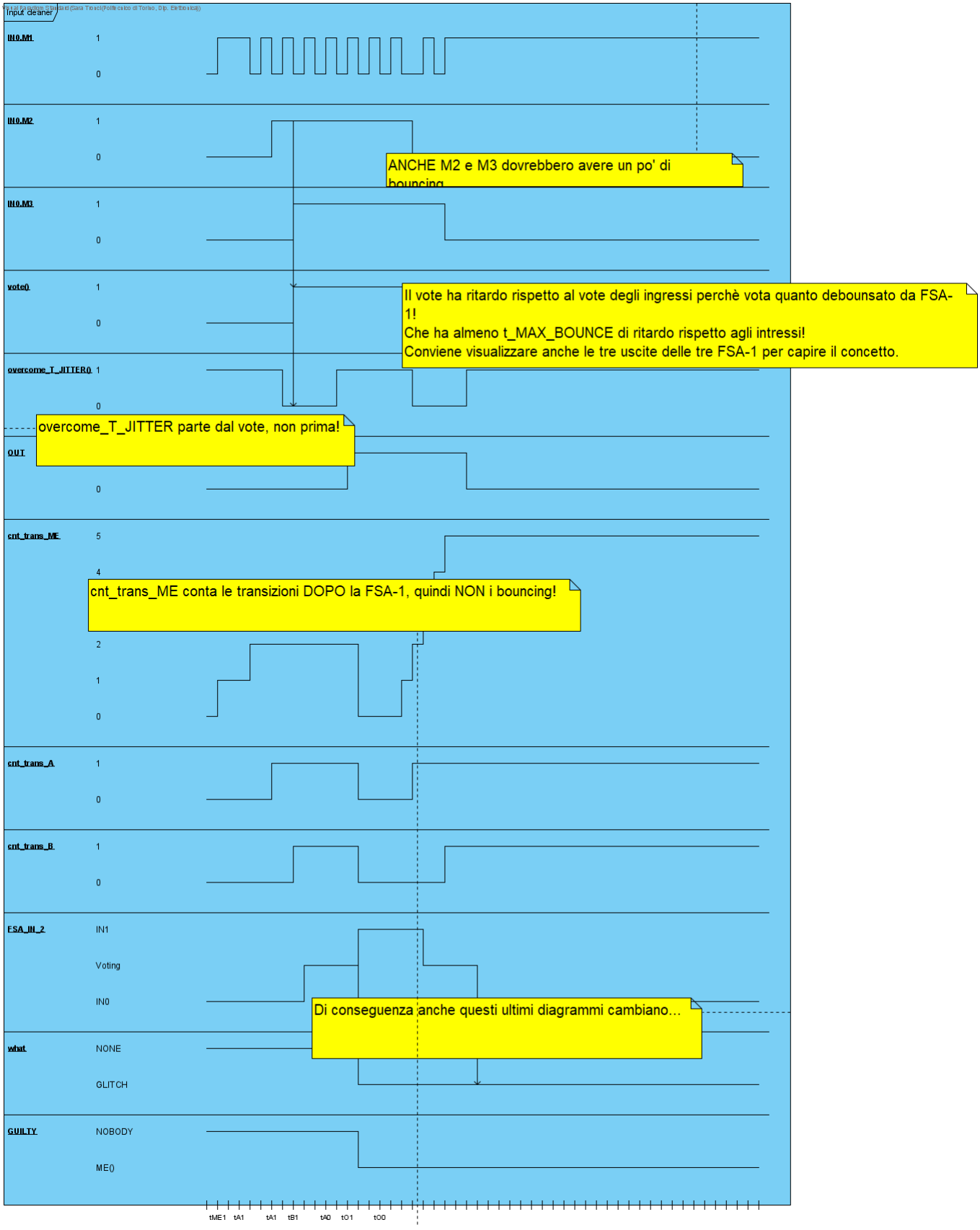


29. Verifica Funzionalita Identificazione Guasto DIN

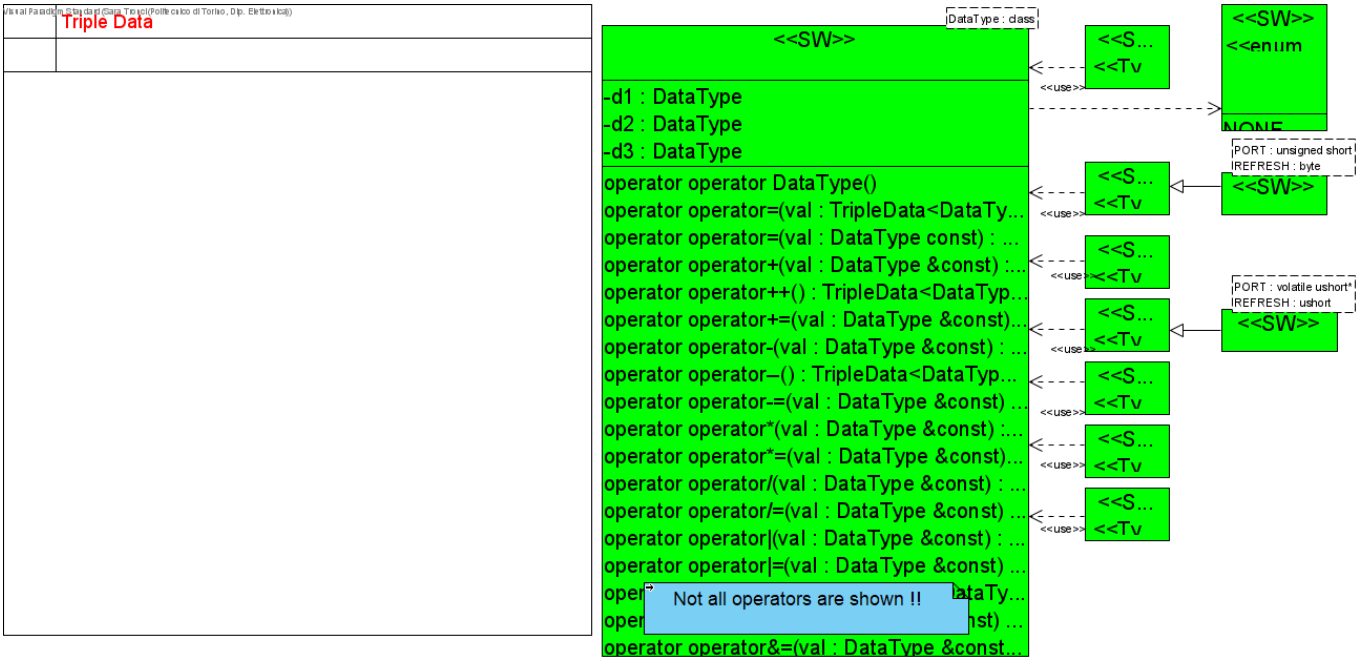
1. .... accendo il (model element not found)



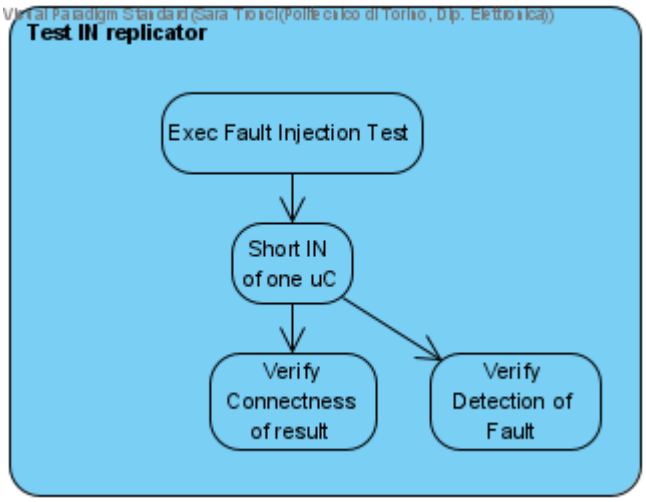
30. Global


Input cleaner
IN0.M1
IN0.M2
IN0.M3
vote0
overcome_T_JITTER0
OUT
cnt_trans_ME
cnt_trans_A
cnt_trans_B
FSA_IN_2
IN1
Voting
IN0
what
NONE
GLITCH
GUILTY
NOBODY
ME0
ANCHE M2 e M3 dovrebbero avere un po' di bouncing
Il vote ha ritardo rispetto al vote degli ingressi perchè vota quanto debounsato da FSA-1!
Che ha almeno t_MAX_BOUNCE di ritardo rispetto agli intressi!
Conviene visualizzare anche le tre uscite delle tre FSA-1 per capire il concetto.
overcome_T_JITTER parte dal vote, non prima!
cnt_trans_ME conta le transizioni DOPO la FSA-1, quindi NON i bouncing!
Di conseguenza anche questi ultimi diagrammi cambiano...
tME1 tA1 tA1 tB1 tA0 tO1 tO0

31. Triple Data



32. Test Steps



32.1. Test IN replicator

Algoritmo del test che può essere effettuato sul DIN replicator, per rilevare eventuali malfunzionamenti.

32.1.1. Short IN of one uC

Induce un corto circuito in un pin d'uscita del DIN replicator, che è connesso a un pin d'ingresso del Computing Unit.

32.1.2. Verify Connectness of result

Verifica che i risultati siano coerenti con quello che ci si aspetta

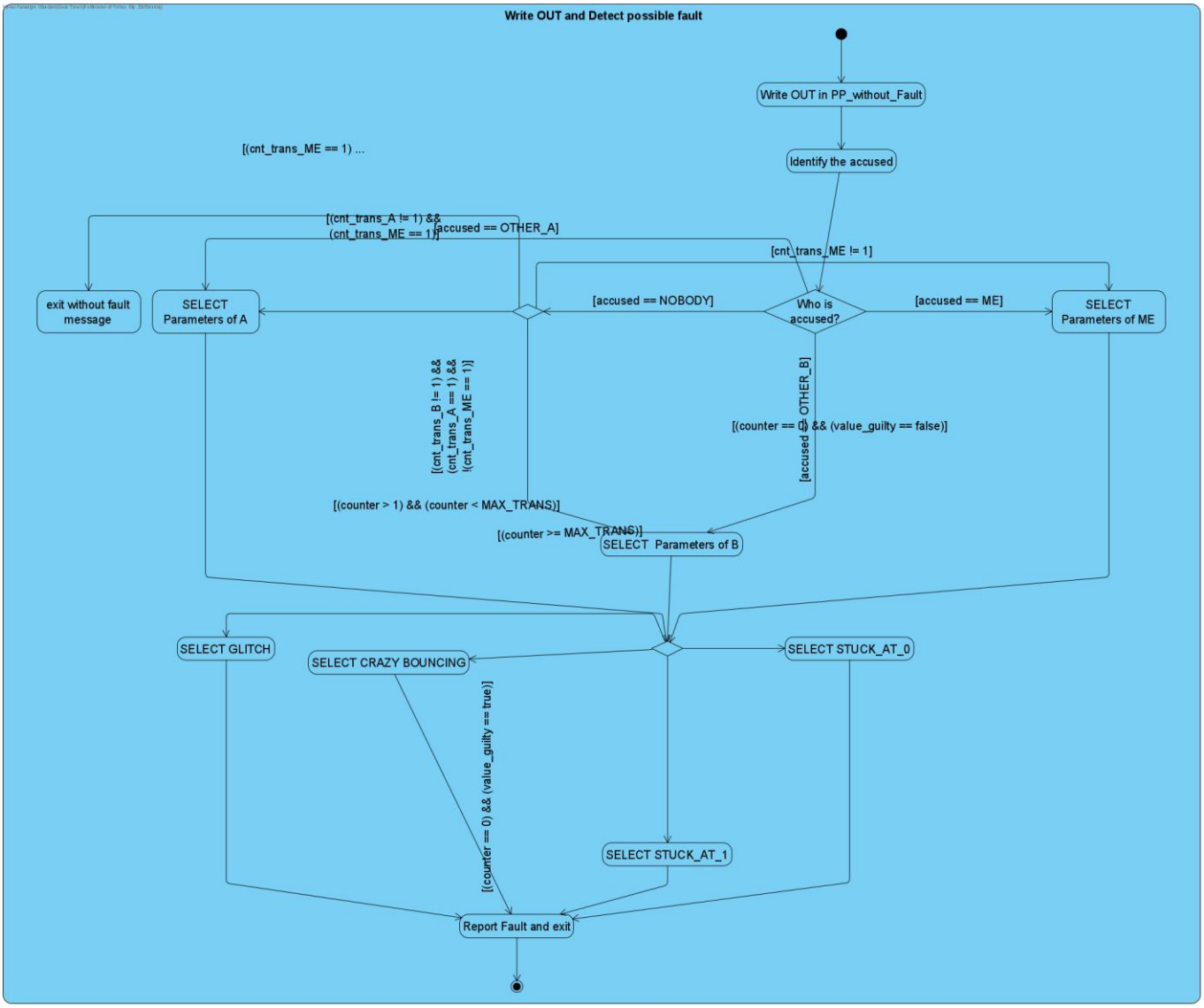
32.1.3. Verify Detection of Fault

Verifica il rilevamento del fault.

32.1.4. Exec Fault Injection Test

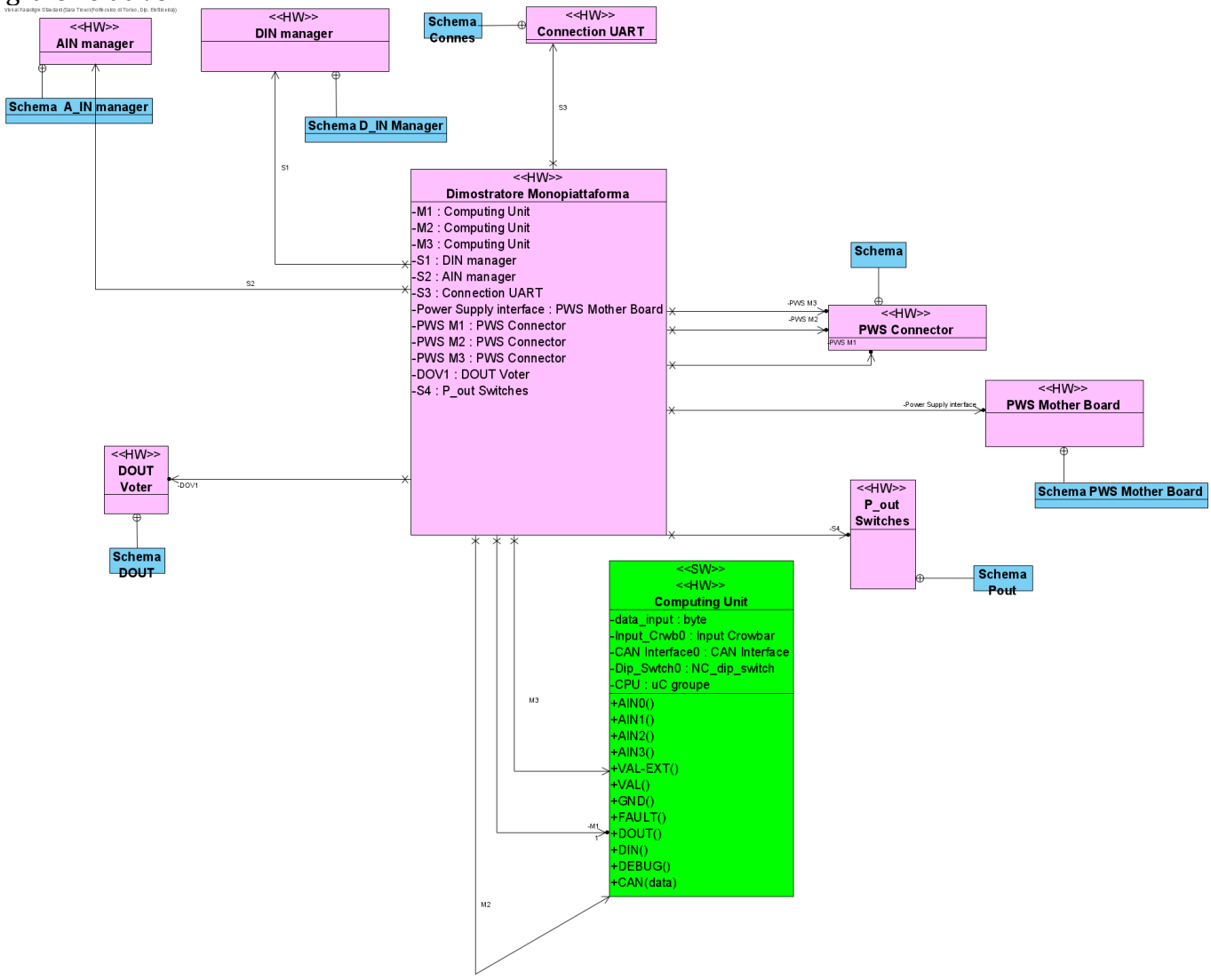
Configurazione di eventuale parametri per l'esecuzione del test in cui viene provocato un fault, forzando a un certo valore gli ingressi del dispositivo.

33. Write OUT and Detect possible fault Activity Diagram



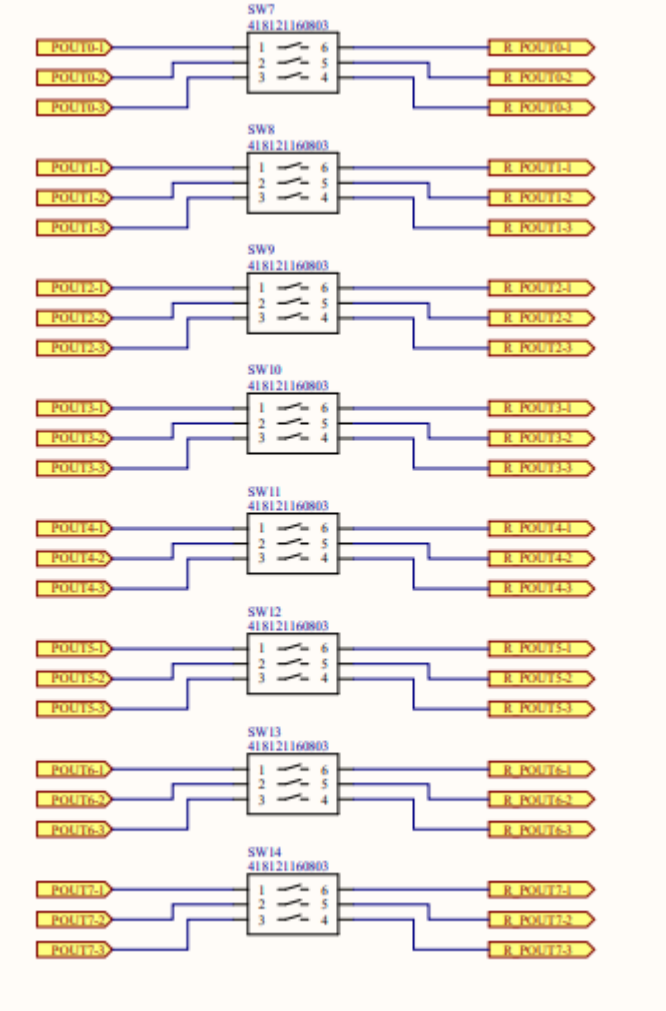
34. Descrizione del Dimostratore Monopiattaforma

Il seguente diagramma delle classi descrive le classi Hardware e Software appartenenti al Dimostratore Monopiattaforma, Le classi che figurano incluse in altre classi sono discusse nel paragrafo relativo.



34.1. Schema Pout Switches

SEgue schema elettrico del modulo. I DIP switch consentono all'utente di abilitare e disabilitare manualmente le uscite di potenza.

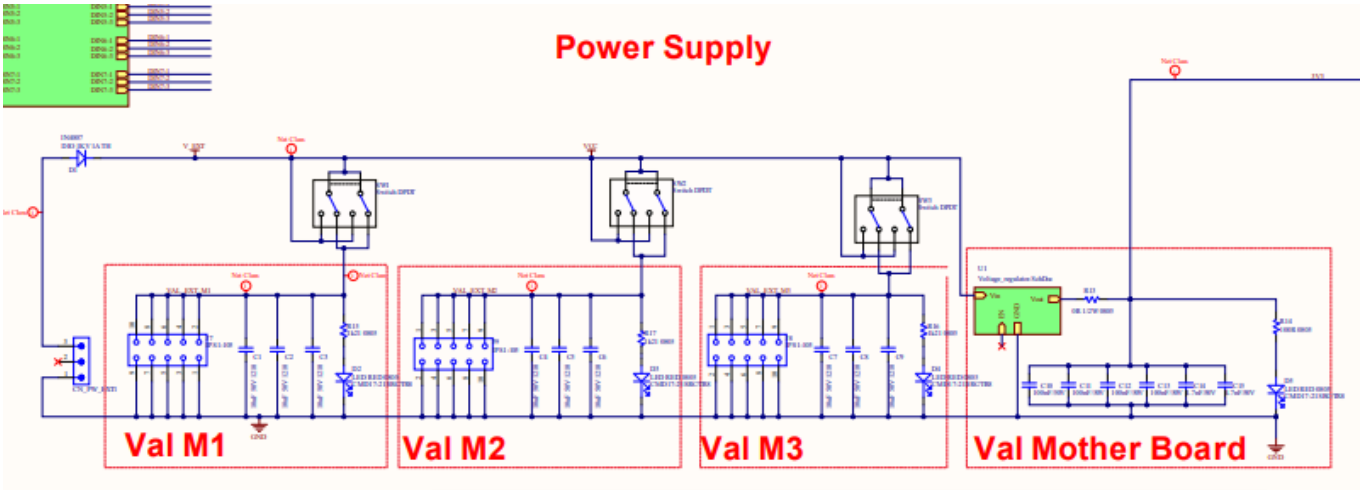


34.2. Schema Connessioni PWS

Schema elettrico relativo all'implementazione del PWS Connector.

Quenta interfaccia è utilizzata quanto i moduli Computing Unit, realizzati come schede piggy back sono connesse alla scheda medre. In questo contesto ricevono la tensione di alimentazione 3.3V nominali generata dal Voltage Regulator presente sulla scheda madre.

Nel circuito è presente anche un'interuttore switch che consente all'utente di sconnettere o attivare l'alimentazione dalla scheda piggy back, anche se essa è connessa alla scheda madre.



34.3. Schema DOUT

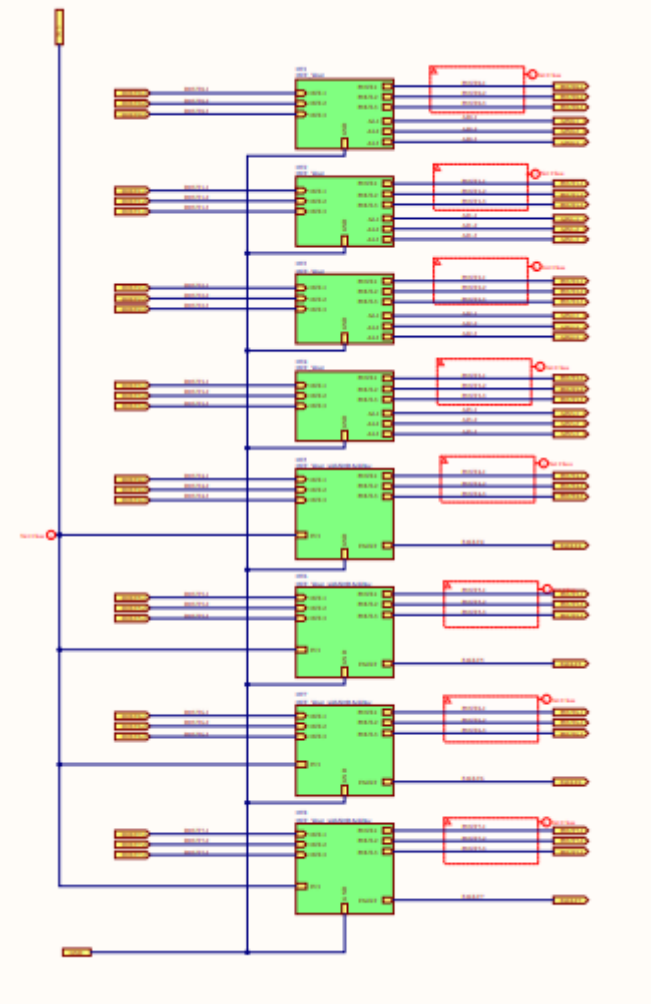
Seguono gli schemi elettrici relativi alla classe DOUT Voter .

COme si vede nella prima figura sono stati utilizzati due moduli differenti:

1. OUT voter modulo elementare, riservato alle prima 4 triplette di DOUT (DOUT0, DOUT1, DOUT2, DOUT3 provenienti dai tre microcontrollori).
2. OUT VOTEr with Xor modulo elentare, riservato alle ultime quatro triplette (DOU4, DOUT5, DOUT6, DOUT7).

Nel due moduli la detteminazione del valore dell'uscita è implementata allo stesso modo, tramite l'utilizzo di due MOS BUK9875_100A per ogni uscita, coi quali si ottiene una porta XOR A tre ingressi.

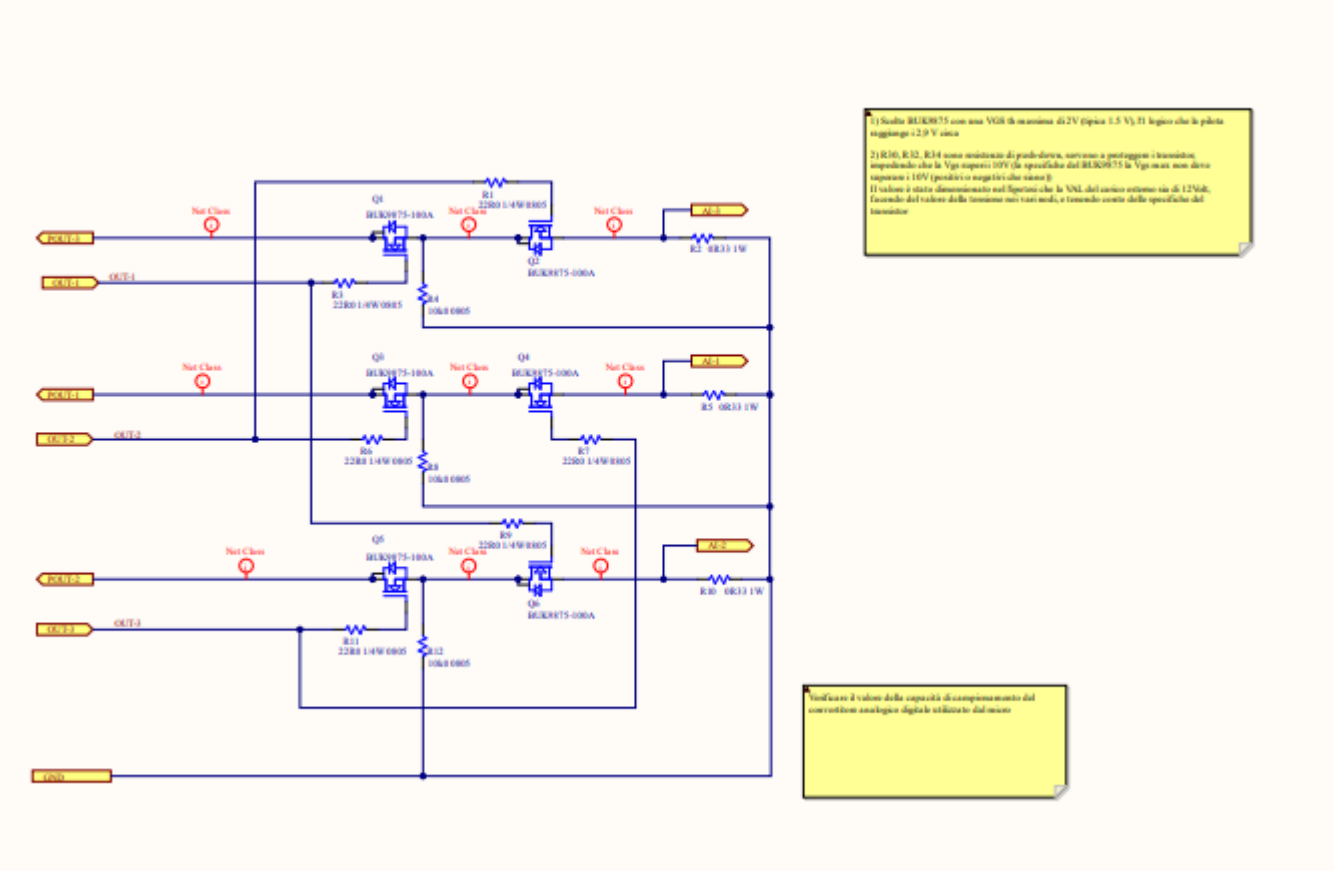
I due blocchi elementari differiscono per il modo con cui viene rilevato il caso di Fault.



IN figura è mostrato lo schema elettrico del modulo elementare OUT VOTER. Per ogni esgnale d'uscita è rilevato il segnale della corrente AI, che si propagano ai 4 ingressi AIN del microcontrollore stesso. Come si è letto nel paragrafo elativo al AIN manager, in questo caso devo essere abilitati i DIP switch

attivare il collegato dei segnali AIN inteni verso il Computing Unit.

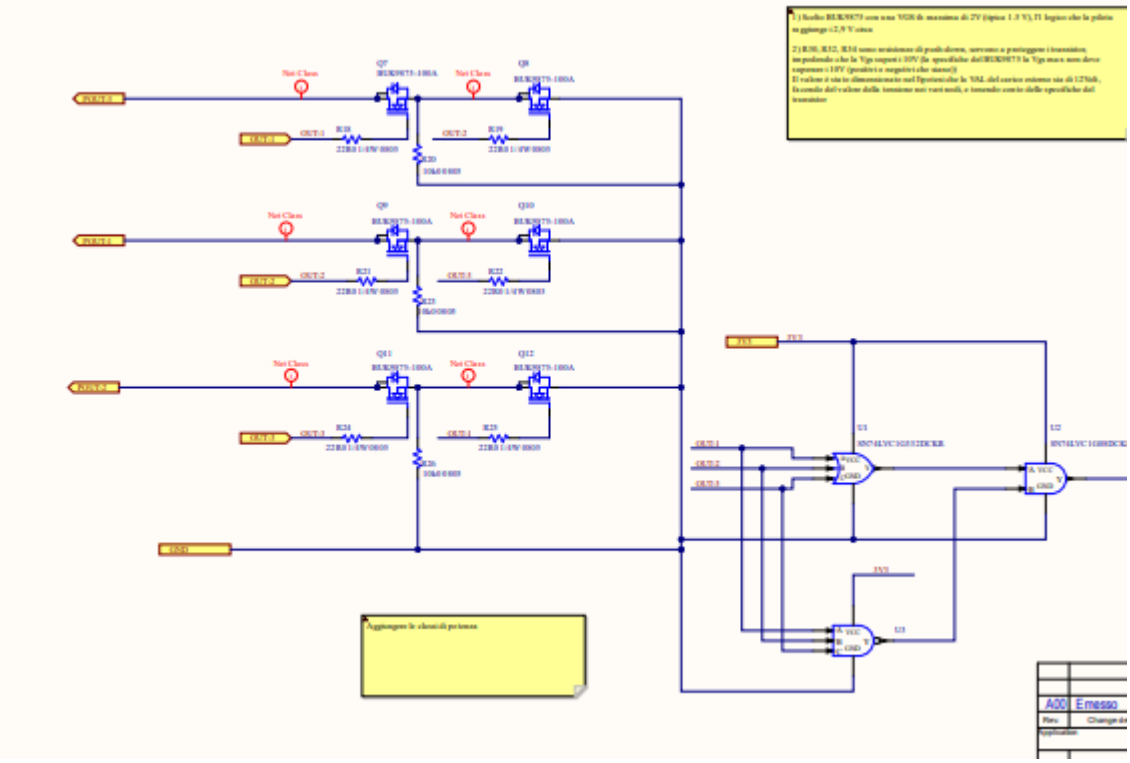
In base al valore delle correnti il Computing Unit asserisce o meno il segnale di Fault relativo a quel DOUT,



La terza figura mostra lo schema elettrico del secondo blocco elementare il DOUT Voter with XOR.

In questo modulo il rilevamento del fault avviene tramite l'implementazione della funzione logica relativa tramite tre porte logiche integrate.

Fault = (OUT1 OR OUT2 OR OUT3) AND ((OUT1 AND OUT2 AND OUT3))

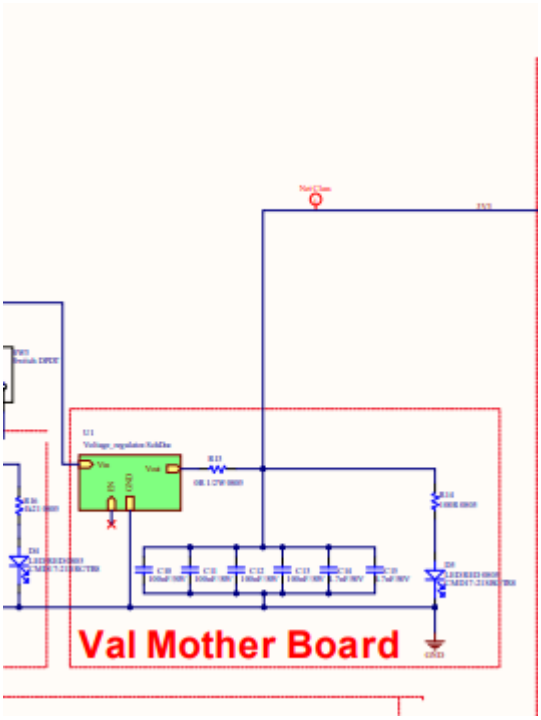


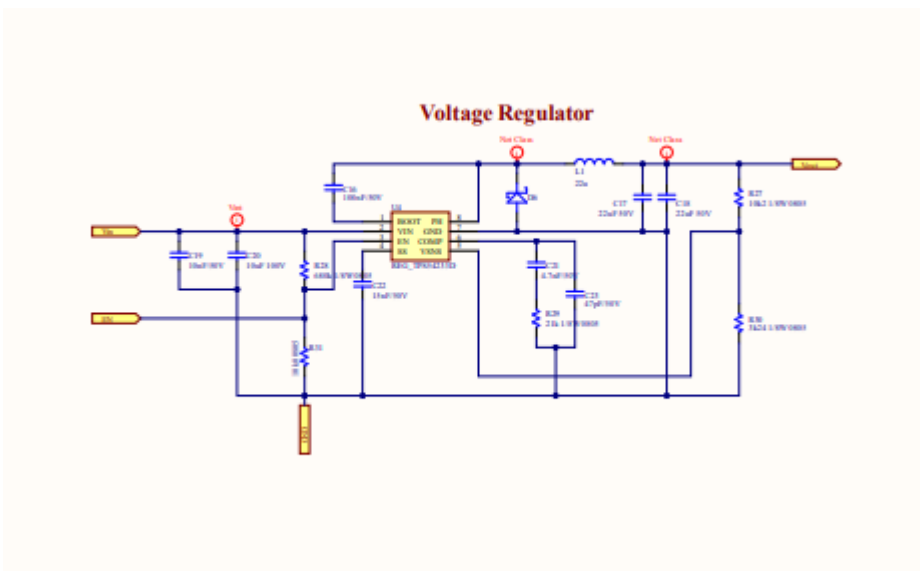
34.4. Schema PWS Mother Board

Segue lo schema elettrico relativo al PWS Mother Board.

La classe riceve in ingresso un alimentazione compresa tra i 12V e i 36V in continua e fornisce un uscita di 3.3V nominale con una tolleranza del 3%.

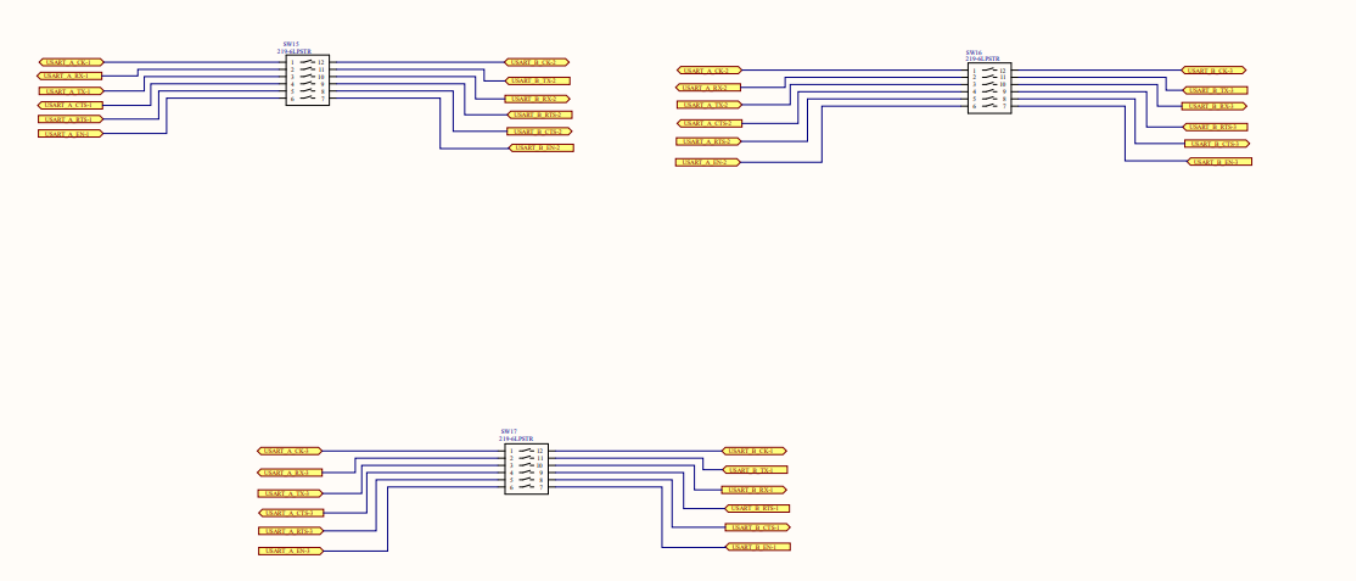
La classe del Voltage Regulator implementata è la stessa utilizzata per alimentare singolarmente il Computing Unit.





34.5. Schema Connessione UART

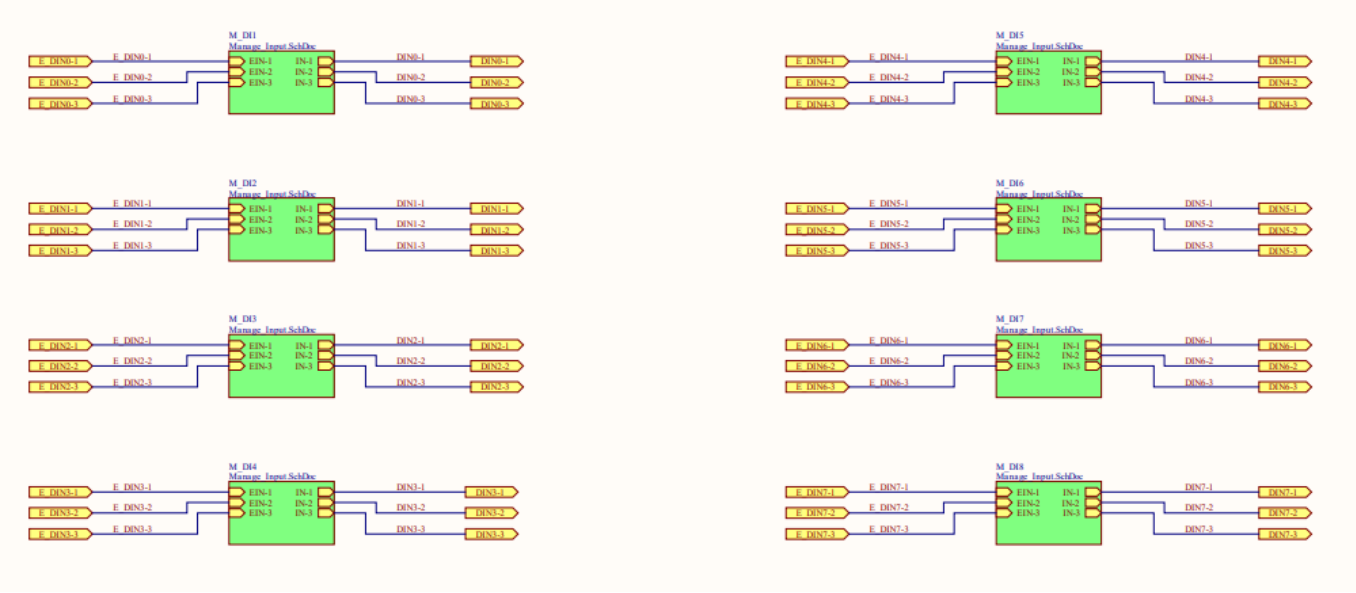
Segue lo schema elettrico del Connection UART che consente di attivare manualmente i collegamenti della comunicazione seriale tra un microcontrollore e gli altri due.

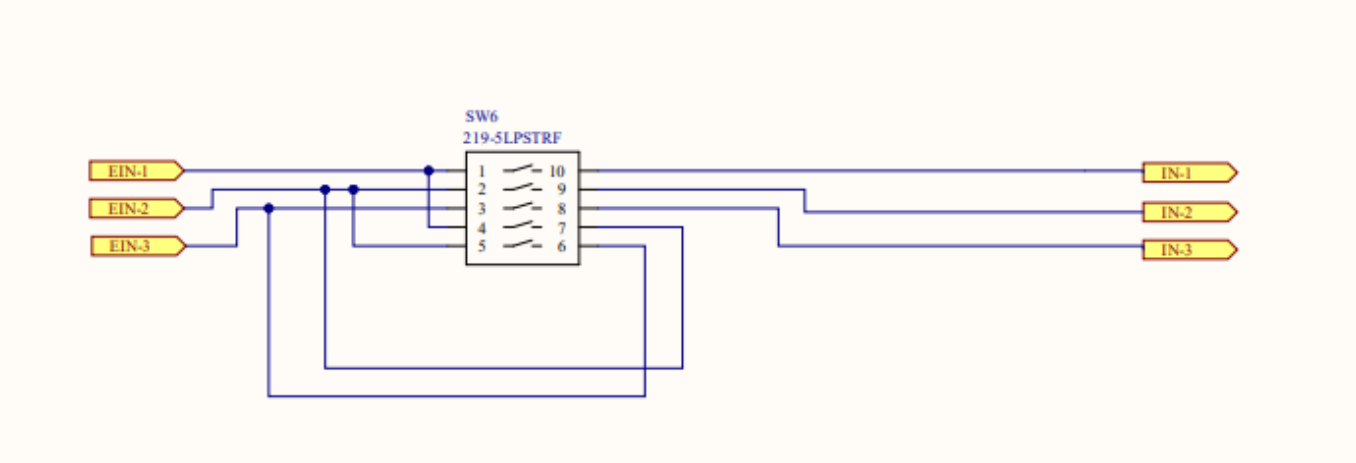


34.6. Schema D_IN Manager

Segue lo schema elettrico del DIN manager, con particolare sul singolo blocco costituito da un Dip Switch che riceve in ingresso i segnali DIN triplicati da propagare verso i tre moduli.

Come si nota dallo schema attivando manualmente le posizioni del DIP è possibile generare un corto tra i segnali, che implica la simulazione di un possibile fault.

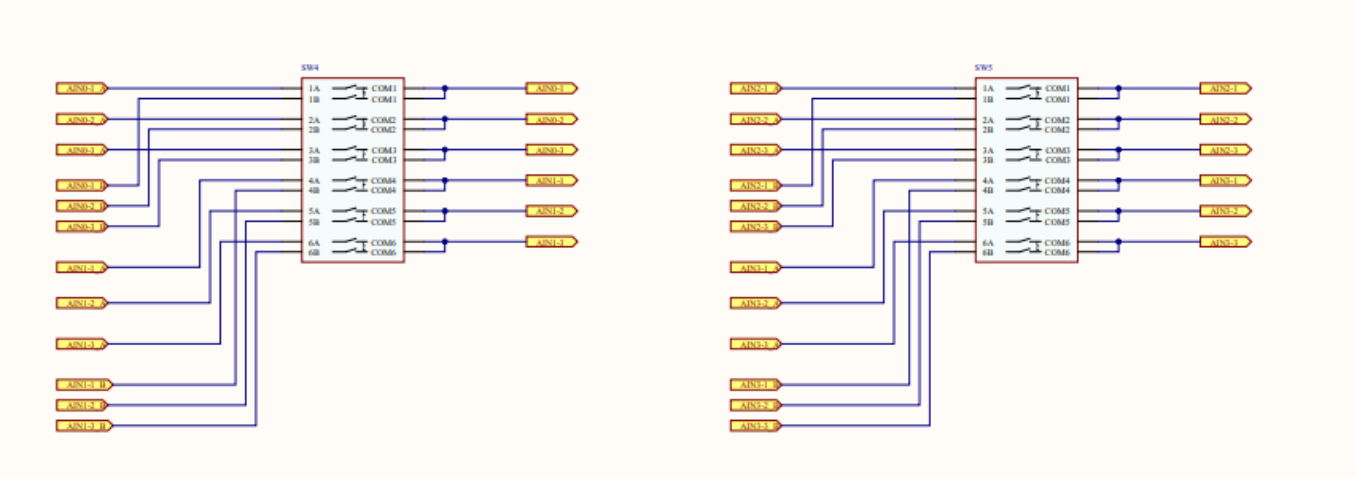




34.7. Schema A_IN manager

In figura è mostrato lo schema elettrico del AIN manager.

i Moduli DP switch consentono di selezionare se propagare al Computing Unit, il segnale analogico proveniente dall'esterno o il segnale analogico interno uscente dal DOUT Voter .



34.8. PWS Connector

Classe relativa alla parte di circuito che propaga l'alimentazione ai moduli Computing Unit M1, M2 e M3 .

Questa classe consente di propagare la VCC da 3.3V nominali, generata dal Voltage Regulator presente sulla scheda MAdre, quando le schede sono connesse ad essa.

L'alimentazione sulle schede piggy Back, può essere disabilitata con l'attivazioni dei tre interruttori a leva, uno per ogni unità.

34.9. Connection UART

La classe Connection UART consente all'utente di attivare manualmente, tramite DIP Switch la comunicazione seriale tr a i Computing Unit M1, M2, M3.

Se attivata ogni modulo comunica con gli altri due in ricezione e in trasmissione secondo protocollo UART.

34.10. Computing Unit

Classe relativa al componente Computing Unit, che contiene il modulo uC.
Contiene tutte le funzioni SW che consentono il funzionamento del modulo.
La classe possiede i seguenti attributi HW di seguito elencate:

1. Input Crobar
2. Analog Input Crowbar
3. CAN INterface
4. OUtput CRowbar
5. Debug Interface
6. uC group
7. SWD Interface

Maggiori dettagli sono presenti nelle classi relative

34.10.1. Attributes

Signature	Description
-data_input : byte	Variabile che contiene il valore degli ingressi digitali
-Data Output : byte	
-Input_Crwb0 : Input Crowbar	Istanza relativa alla classe Input Crowbar, dedicata alla protezione dei Digital Input dalle sovratensioni.
-CAN Interface0 : CAN Interface	
-Outup Crowbar0 : Output Crowbar	
-DEBUG : DEBUG Interface	
-Analog Iput Crowbar 0 : Analog Input Crowbar	
-SW Interface 0 : SWD Interface	
-Dip_Swtch0 : NC_dip_switch	
-CPU : uC groupe	
-J1 : IPT1-115-BOTTOM	
-J2 : IPT-110-BOTTOM	
-CAM M : DB9 M	
-CAN F : DB9 F	
-CN_PW_EXT : CONN MALE 3 POS	

-CN_PW : IPT-105-BOTTOM	
-DATA compare : InfoSharing	

34.10.2. AIN0
Segnale di Analog Input del Computing Unit.

34.10.3. AIN1
Segnale di Analog Input del Computing Unit.

34.10.4. AIN2
Segnale di Analog Input del Computing Unit.

34.10.5. AIN3
Segnale di Analog Input del Computing Unit.

34.10.6. VAL-EXT
Pin positivo connesso alla alimentazione esterna del Computing Unit.Il pin va collegato a un tensione continua (tra il pin e il GND della scheda) compresa ta i 12Vdc e i 36 VDC.

34.10.7. VAL
Pin positivo connesso alla alimentazione VCC 3.3V del Computing Unit. Consente di alimentare la scheda quando è inseita sulla scheda madre e l'alimentatore è abilitato tramite switch.

34.10.8. GND
Pin di riferimento GND.

34.10.9. FAULT
Segnale digital Output attivo alto in presenza di un fault

34.10.10. DOUT
Digital signals maching Computing Unit's pins DOUT0, DOUT1, DOUT2, DOUT3, DOUT4, DOUT5, DOUT6 and DOUT7.

34.10.11. DIN
Digital signals input maching Computing Unit's pins DIN0, DIN1, DIN2, DIN3, DIN4, DIN6, DIN5 e DIN7.

34.10.12. DEBUG
Insieme dei segnali utilizzati per il download del SW all'interno del Computing Unit. vedi maggioni dettagli nel diagramma dei componenti relativo.

34.10.13. CAN
Insieme di segnali relativi all'interfaccia CAN utilizzata dal Computing Unit.

34.11. P_out Switches
Classe relativa a i segnali d'uscita in potenza.

34.12. DOUT Voter
Classe relativa al Il DOUT Voter .

La classe implementata riceve in ingresso 8 DIgital output da ogni microcontrollore e restituisce sempre tre repliche del segnale ma dopo avere eseguito una votazione interna.

Dati tre bit, dato che il valore può essere o '0' o '1' anche in presenza di un fault ci saranno sempre almeno due bit concordi, e questo rende l'uscita certa, che viene poi triplicata.

Nel caso in cui uno dei segnali non è concorde con gli altri due viene asserito il segnale di presenza di Fault.

Per maggiori informazioni su come funziona internamente il modulo vedere la parte relativa allo schema elettrico.

34.13. AIN manager

La classe AIN manager ha la funzione di propagare ai tre moduli Computing Unit i segnali AIN interno, provenienti dal DOUT Voter , oppure i 4 AIN provvenienti dall'esterno.

Questa funzione è implementata via HW con l'ausilio di due moduli DIP switch, come si vede dallo schema elettrico.

34.14. PWS Mother Board

La classe del PWS Mother Board implementa il modulo che consente di alimentare dall'esterno la scheda madre.

Il modulo implementato utilizza la stessa confifurazione del Voltage Regulator presente all'interno del Computing Unit.

Il modulo riceve un'alimentazione esterna compresa tra i 12Vdc e i 36Vdc ed eroga un uscita nominale di 3.3V necessaria per alimentare i moduli attivi presenti all'interno della scheda, e fornire un riferimento di tensione.

34.15. DIN manager

Modulo che consente triplicare gli ingressi e simulare dei cortocircuiti tra loro in tutte le combinazioni.

Questo modulo implementa il component DIN replicator-1.

34.16. Dimostratore Monopiattaforma

Classe relativa al Dimostratore Monopiattaforma.

La classe è di tipo HW e contiene al suo interno tre moduli Computing Unit, e una serie di moduli di tipo HW utilizzati principalmente per simulare i guasti e garantire un'uscita certa e la segnalazione del FAULT se presente:

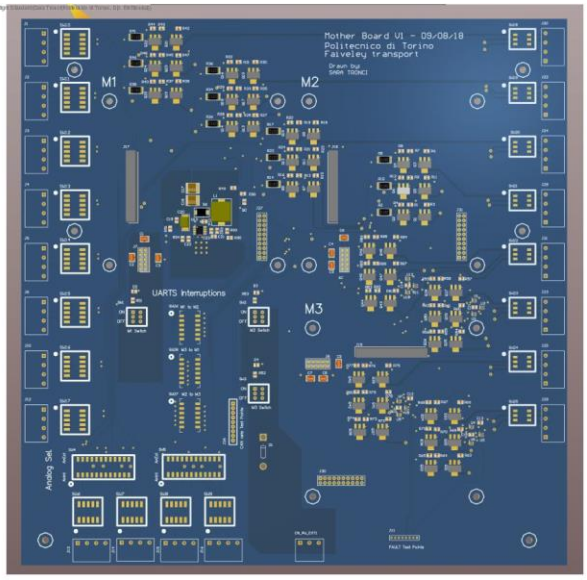
1.

34.16.1. Attributes

Signature	Description
-M1 : Computing Unit	Modulo di Computing Unit identificato come M1 e contrassegnato dall'ID '01' .
-M2 : Computing Unit	Modulo di Computing Unit identificato come M2 e contrassegnato dall'ID '10' .

-M3 : Computing Unit	Modulo di Computing Unit identificato come M2 e contrassegnato dall'ID '11' .
-S1 : DIN manager	Oggetto della classe DIN manager che riceve in ingresso per un totale di 24 segnali accorparti a tre. Ogni trio è un DI.
-S2 : AIN manager	Oggetto della classe AIN manager. Il modulo implementato riceve in gresso i 4 segnali analogici AIN provenienti dall'esterno, con le loro triplice (3x4 AIN).
-S3 : Connection UART	L'oggetto alla classe Connection UART. Gestisce per ogni Computing Unit i segnali relativi alla due porte di comunicazione seriale USART.
-Power Supply interface : PWS Mother Board	Oggetto della classe PWS Mother Board, da connettere all'alimentazione esterna in continua (da12Vdc a 24Vdc) relativa al Dimostratore Multipiattaforma.
-PWS M1 : PWS Connector	Oggetto della classe PWS Connector, utilizzato per fornire l'alimentazione al modulo M1 Computing Unit, nel momento in cui è collegato al Dimostratore Monopiattaforma.
-PWS M2 : PWS Connector	Oggetto della classe PWS Connector, utilizzato per fornire l'alimentazione al modulo M2 Computing Unit, nel momento in cui è collegato al Dimostratore Monopiattaforma.
-PWS M3 : PWS Connector	Oggetto della classe PWS Connector, utilizzato per fornire l'alimentazione al modulo M3 Computing Unit, nel momento in cui è collegato al Dimostratore Monopiattaforma.
-DOV1 : DOUT Voter	Istanza della classe DOUT Voter riceve in ingresso le tre repliche, una per degli 8 DO proventi dai tre Computing Unit e restituisce le repliche dei 4 segnali analogici e le 4 segnali dedicati ai fault relativi ai DO4, DO5, DO6, DO7. Per dettagli sul funzionamento vedere la classe relativa.
-S4 : P_out Switches	Istanza relativa alla classe P_out Switches. Vedi classe citata.

35. FOTO PCB scheda madre



Modello 3D Scheda Madre Dimostratore Multipiattaforma - Lato Componenti

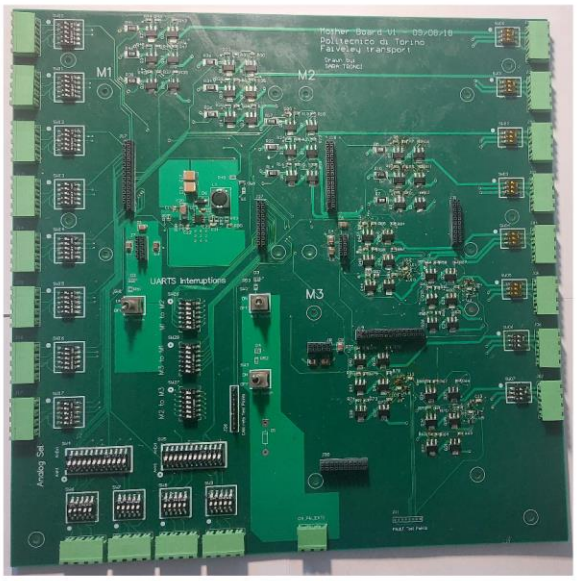


Foto Scheda Madre Dimostratore Multipiattaforma - Lato Componenti

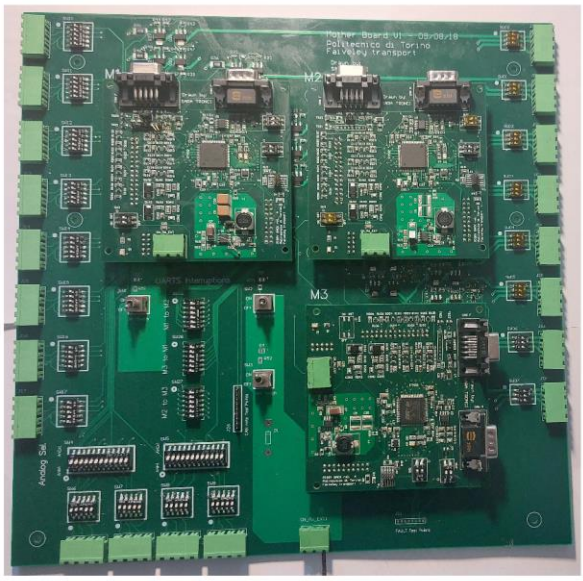
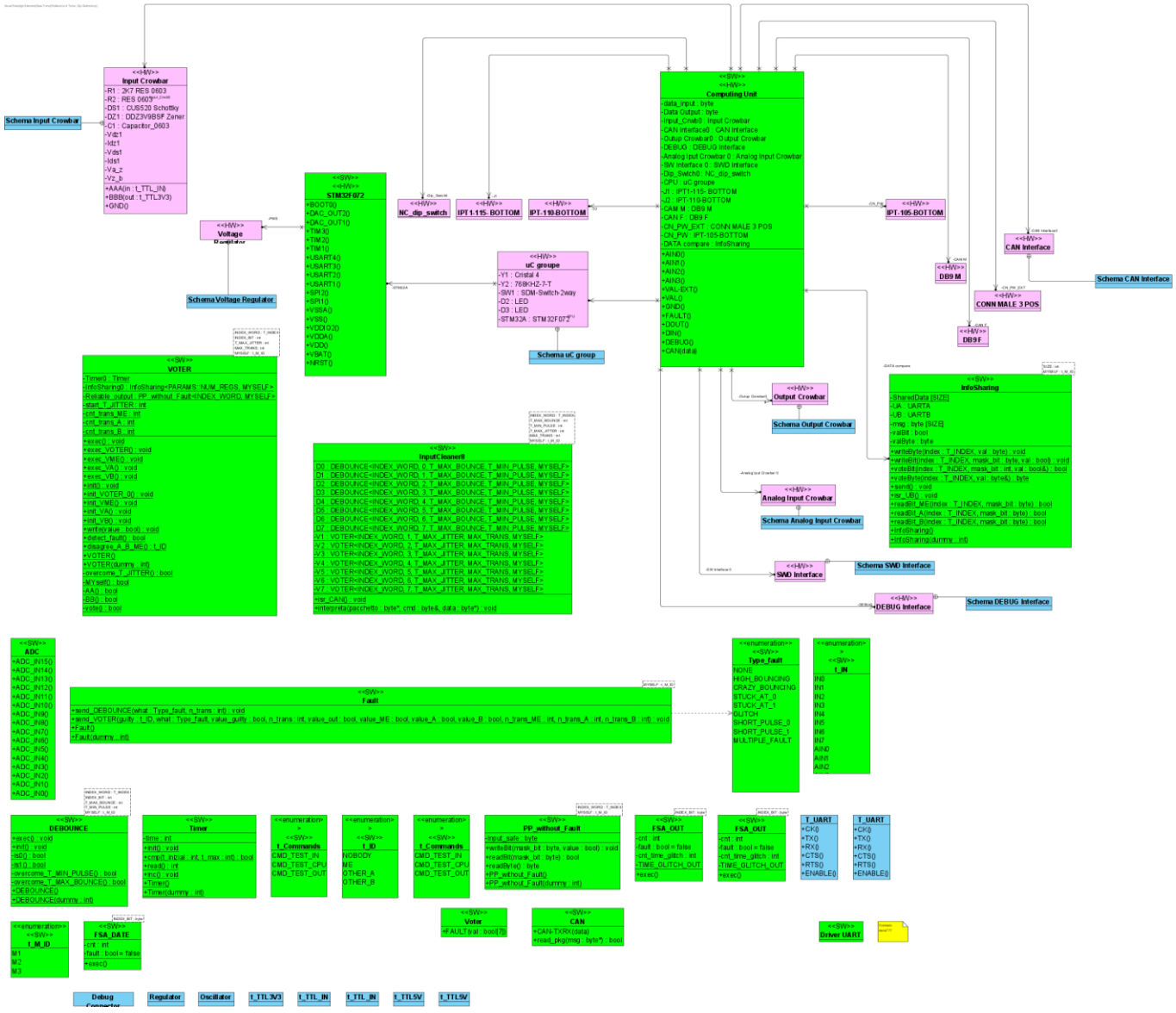


Foto Scheda Madre Dimostratore Monopiattaforma con 3 Computing Unit installate

36. Descrizione del Computing Unit

Il seguente diagramma delle classi illustra com'è costituita la classe del Computing Unit.
Questa unità è la parte elaborativa del Dimostratore Monopiattaforma e contiene sia classi Hardware che classi Software.



36.1. Output Crowbar

Classe relativa allOutput Crowbar, che ha la funzione di proteggere gli Output Digitali dalle sovratensioni

36.2. T_UART

Tipo che contiene i 5 fili standard della urta più l'enable

36.3. t_TTL_IN

Parameters (tagged values) to define logic levels of 3.3V TTL signals.

36.4. t_TTL5V

Parameters (tagged values) to define logic levels of 5V TTL signals.

36.5. IPT-105-BOTTOM

Classe del connettore di interfaccia della tensione di alimentazione da VCC da 3.3V nominali con cui la scheda piggy back riceve l'alimentazione dalla scheda madre.

36.6. t_TTL3V3

Parameters (tagged values) to define logic levels of 3.3V TTL signals.

36.7. CONN MALE 3 POS

Classe relativa al connettore da tre posizioni a cui connettere l'alimentazione esterna quando vi è necessità di utilizzare il Computing Unit senza che sia montato sul Dimostratore Monopiattaforma. In questo caso l'alimentazione da connettere è compresa tra i 12Vdc e i 36Vdc.

36.8. DB9 F

Classe relativa a uno dei due connettori DB9 utilizzato per la comunicazione tramite rete CAN di due Computing Unit quando non sono connessi alla scheda madre.

36.9. DB9 M

Classe relativa a uno dei due connettori DB9 utilizzato per la comunicazione tramite rete CAN di due Computing Unit quando non sono connessi alla scheda madre.

36.10. IPT-110-BOTTOM

Classe relativa al connettore di interfaccia sulla scheda madre attraverso il quale vengono propagati i segnali relativi alle due interfacce seriali USART A e USART B e ai due segnali della rete can.

36.11. IPT1-115- BOTTOM

Classe relativa la connettore a cui sono collegati gli 8 DIN, gli 8 DOUT e i 4 segnali AIN.
Questo è il connettore di interfaccia dati sulla scheda madre.

36.12. CAN Interface

Classe relativa all'interfaccia della rete CAN.

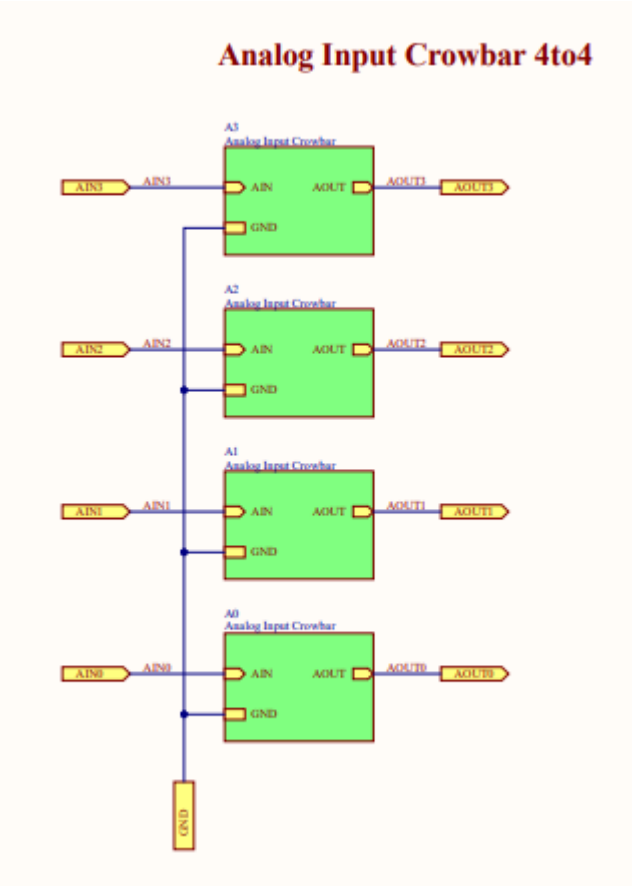
36.13. NC_dip_switch

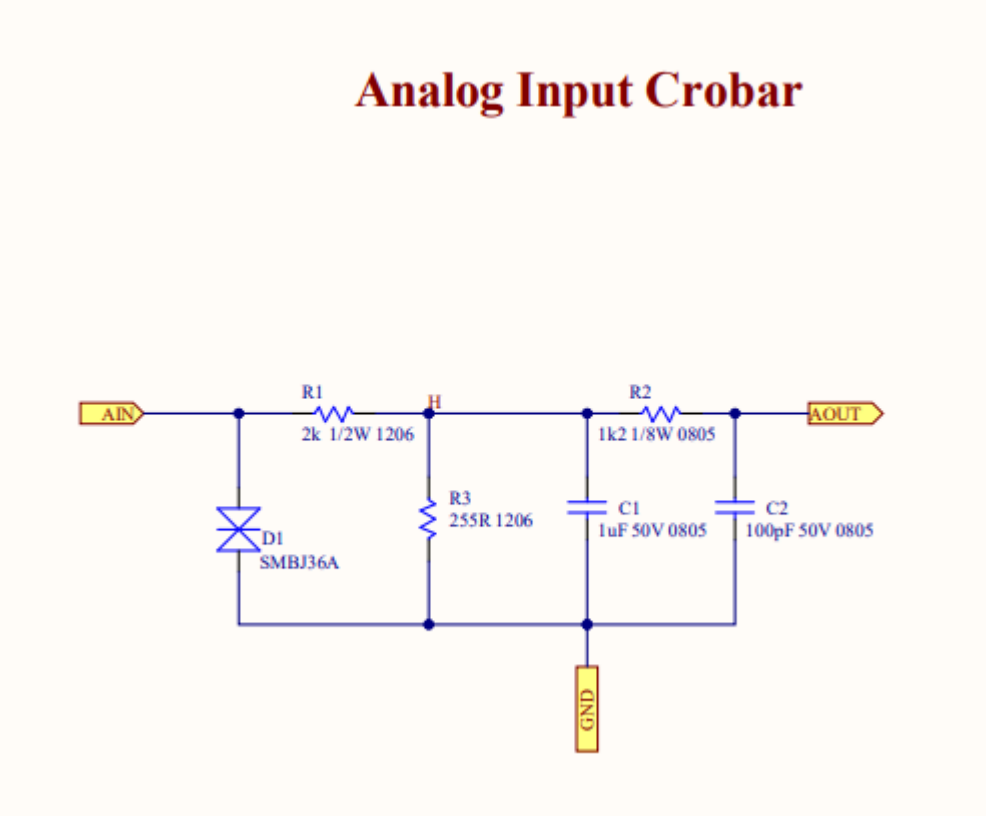
Classe relativa al componente DIP switch CT2192MST-ND utilizzato per effettuare il reset del circuito e per abilitare il uC group.

Ciò avviene tramite attivazione dei due interruttori DIP relativi tramite azione esterna dell'utente.

36.14. Schema Input Crowbar

Di seguito lo schema del modulo input Crowbar utilizzato per proteggere il segnale dalle sovratensioni.





36.15. Input Crowbar

Classe relativa all'Input Crowbar. Questo modulo elementare riceve in ingresso un segnale digitale. Il modulo protegge gli ingressi del STM32F072 a cui è collegato dalle sovratemsioni e dalle sovracorrenti.

36.15.1. Attributes

Signature	Description
-R1 : 2K7 RES 0603	
-R2 : RES 0603	
-DS1 : CUS520 Schottky	
-DZ1 : DDZ3V9BSF Zener	
-C1 : Capacitor_0603	
-Vdz1	Voltage drop between DZ1: DDZ3V9BSF Zener Anode and Cathode.
-Idz1	Current flowing in DZ1: DDZ3V9BSF Zener.
-Vds1	Voltage drop between DS1: CUS520 Schottky DZ1: DDZ3V9BSF Zener Anode and Cathode.
-Ids1	Current flowing in DS1: CUS520 Schottky.
-Va_z	Voltage drop in R1: 2K7 RES 0603.
-Vz_b	Voltage drop in R2: RES 0603.

36.16. DEBUG Interface

Classe relativa all'interfaccia per il download dell'applicativo all'interno del Computing Unit

36.17. Analog Input Crowbar

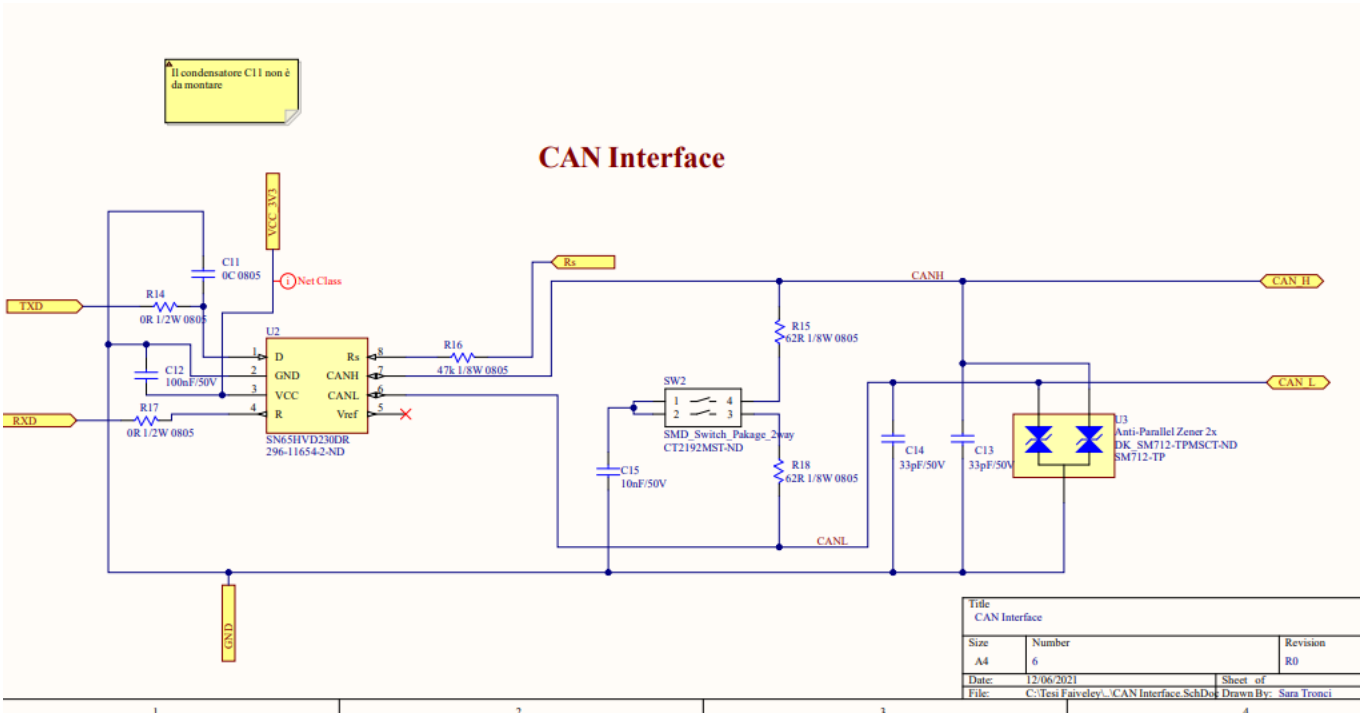
Classe che protegge da sovratensioni gli ingressi analogici del Computing Unit.

36.18. SWD Interface

Classe relativa al modulo di interfaccia per il debug dell'applicativo all'interno del Computing Unit.

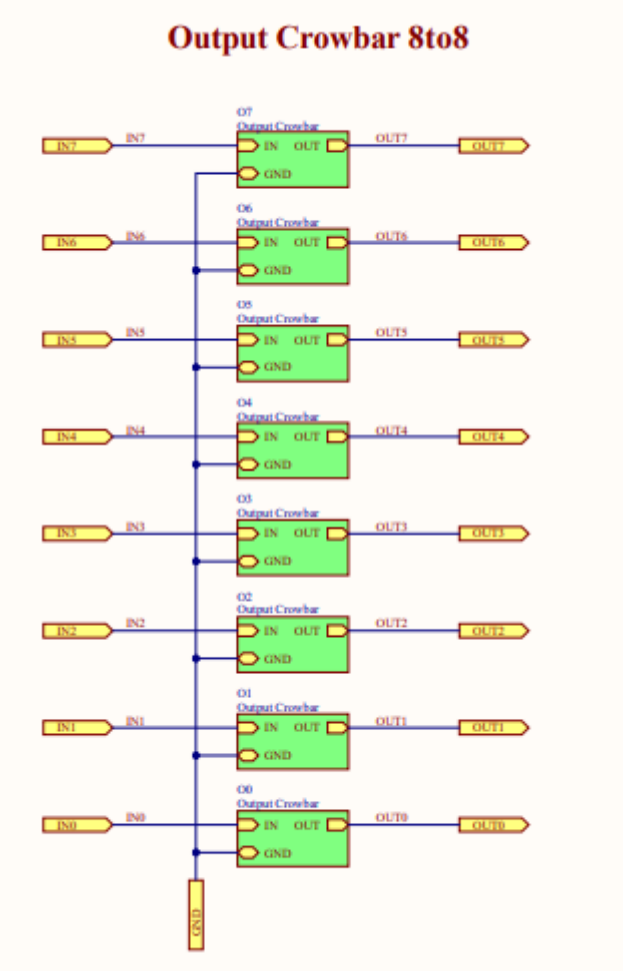
36.19. Schema CAN Interface

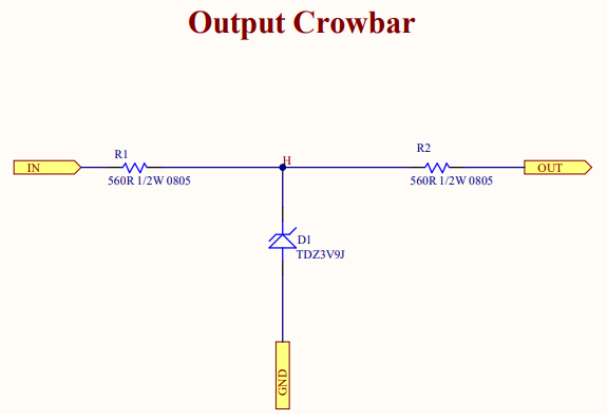
Segue Schema elettrico dell'interfaccia della rete CAN.



36.20. Schema Output Crowbar

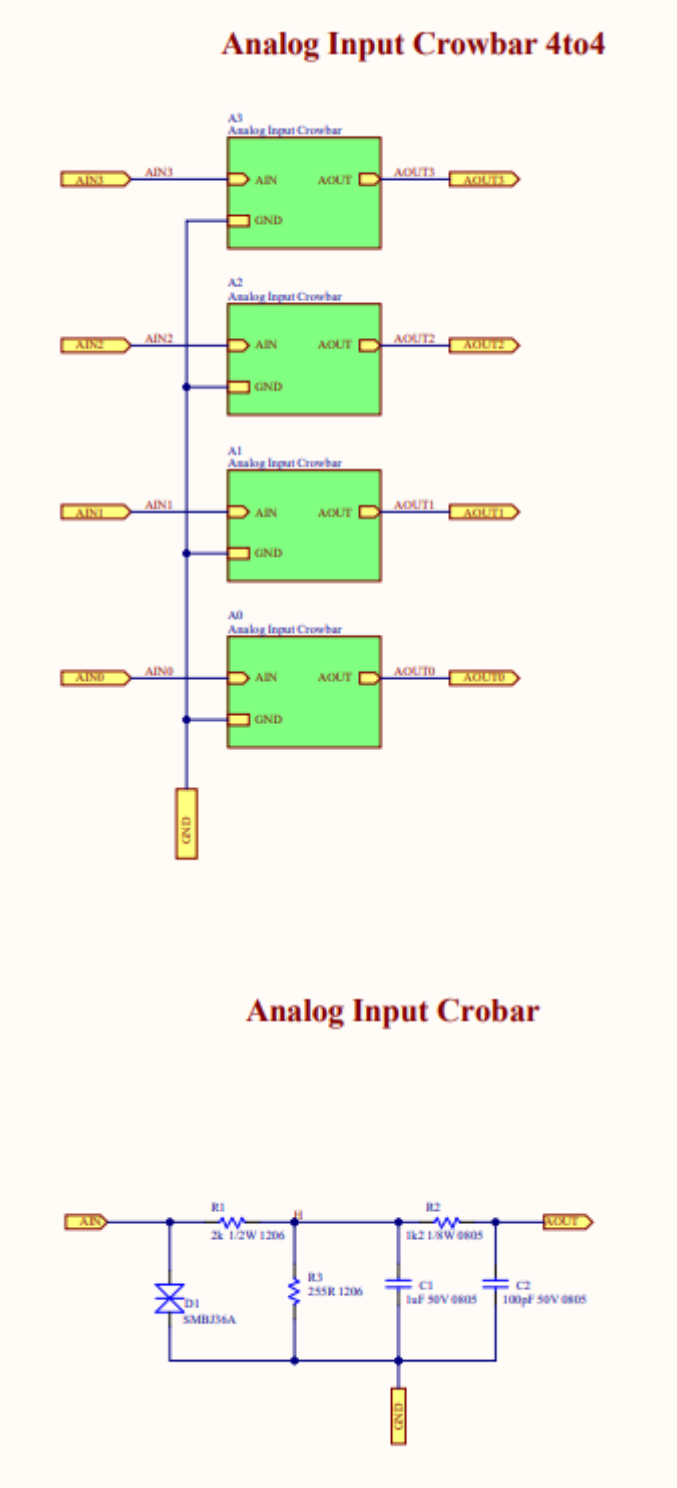
Schema elettrico dell' Output Crowbar.
Inserito zener e restistenze per la protezione dalle sovratensioni.





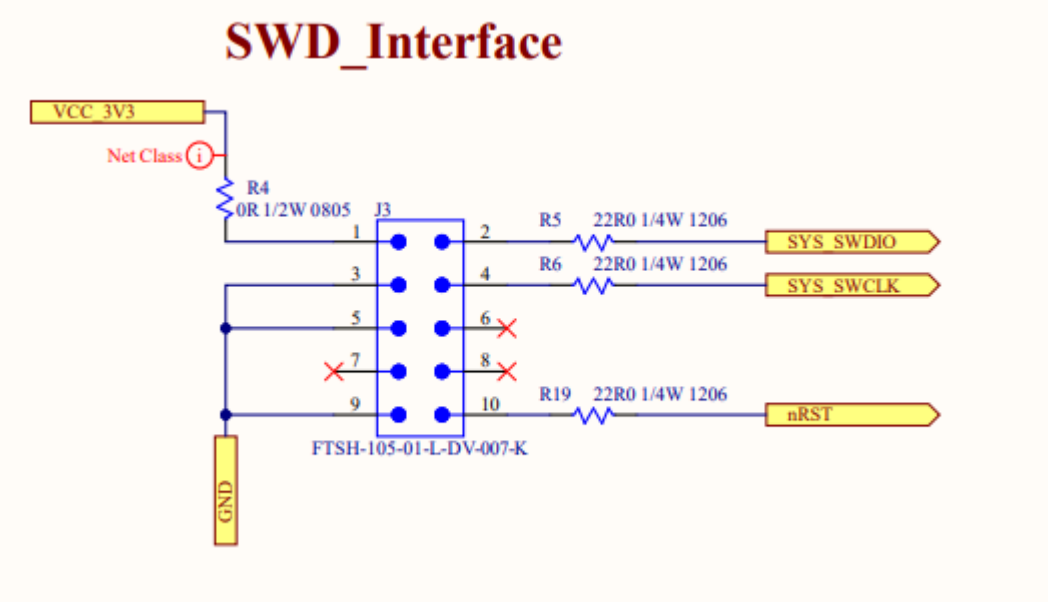
36.21. Schema Analog Input Crowbar

Di seguito lo schema relativo all'Analog Input Crowbar. Inserito transil bidirezione e filtri RC per la protezione sovratensioni e sovracorrenti sugli ingressi analogici.



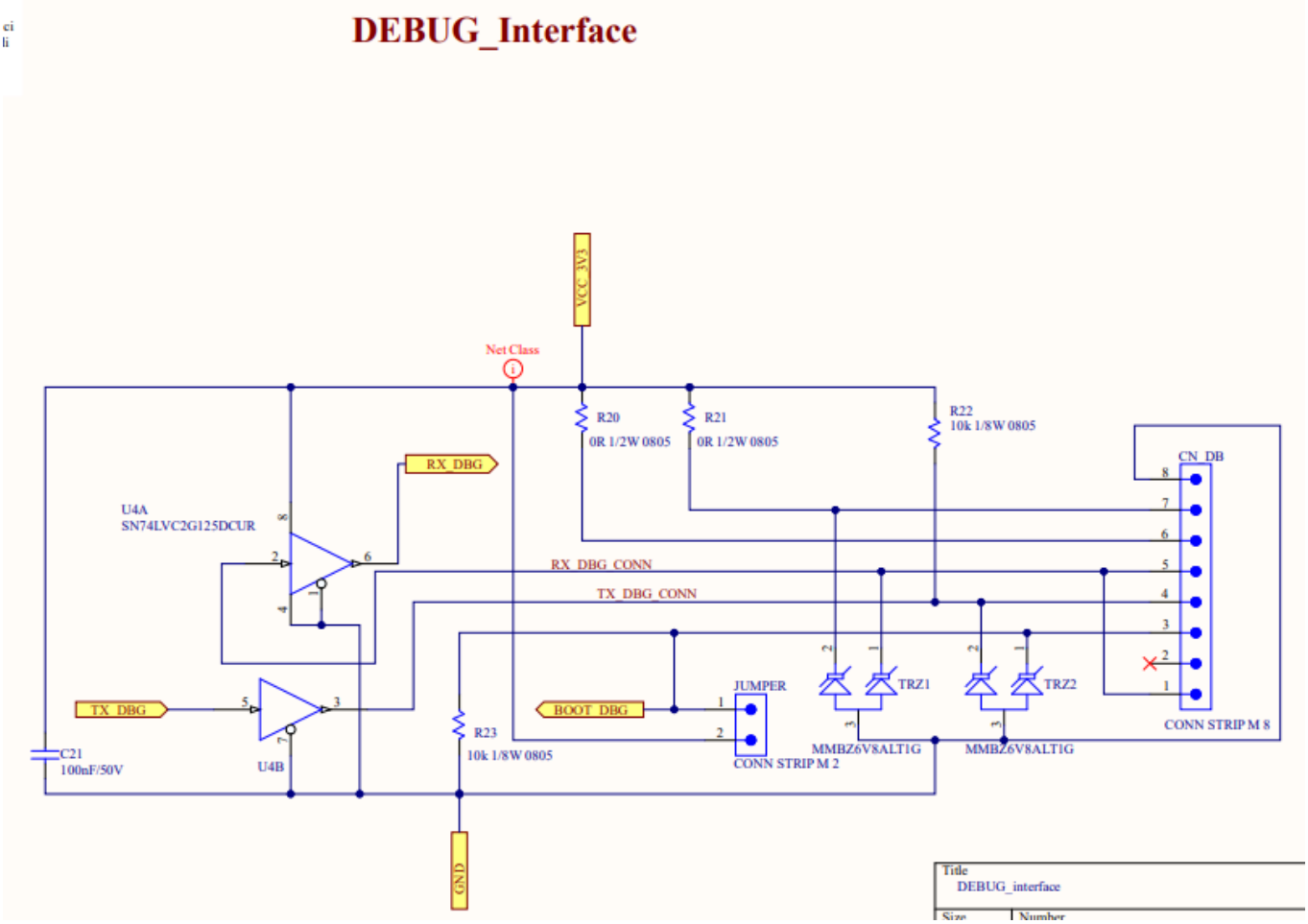
36.22. Schema SWD Interface

segue schema elettrico dell' interfaccia per il debut del software.



36.23. Schema DEBUG Interface

Segue schema elettrico dell'interfaccia di download dell'applicativo, tramite apposito TOOL



36.24. CAN

36.24.1. CAN-TXRX

Signature: CAN-TXRX(data)

36.24.2. read_pkg

Signature: read_pkg(msg : byte) : bool

36.25. Output Crowbar

Classe relativa allOutput Crowbar, che ha la funzione di proteggere gli Output Digitali dalle sovratensioni

36.26. DEBUG Interface

Classe relativa all'interfaccia per il download dell'applicativo all'interno del Computing Unit

36.27. Analog Input Crowbar

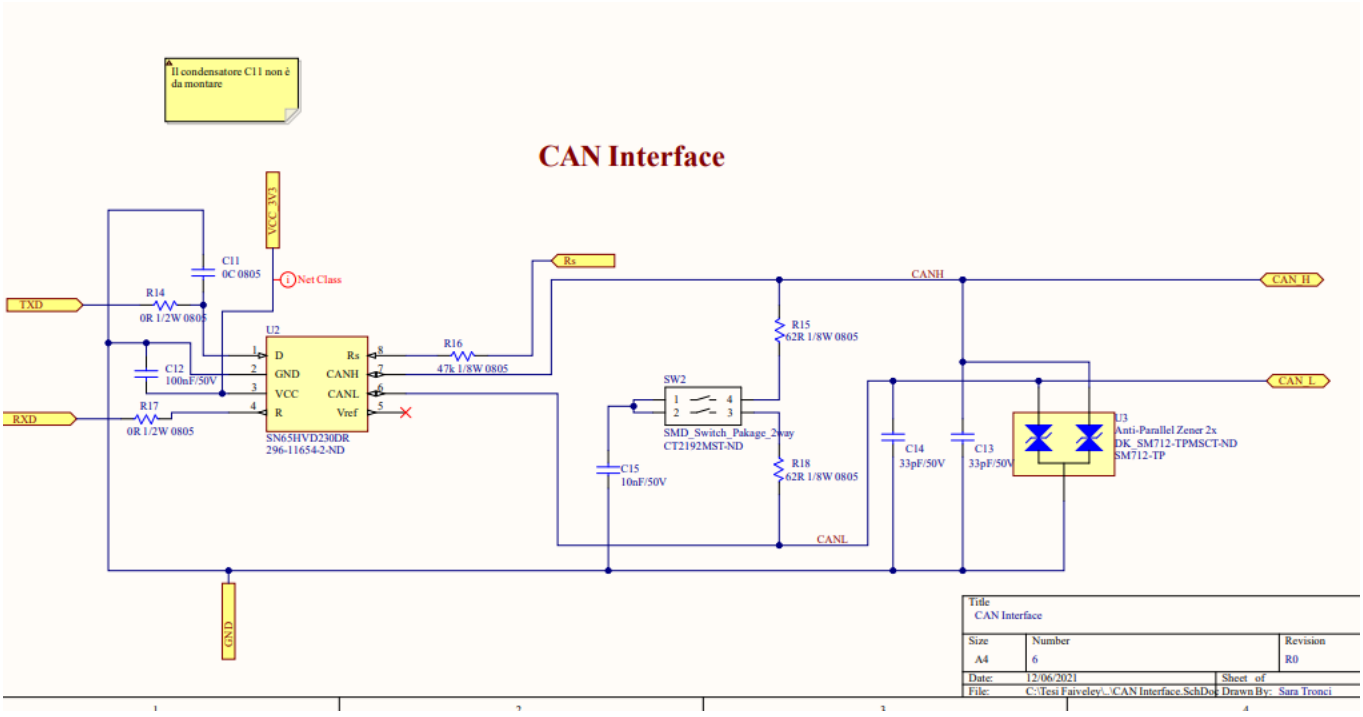
Classe che protegge da sovratensioni gli ingressi analogici del Computing Unit.

36.28. SWD Interface

Classe relativa al modulo di interfaccia per il debug dell'applicativo all'interno del Computing Unit.

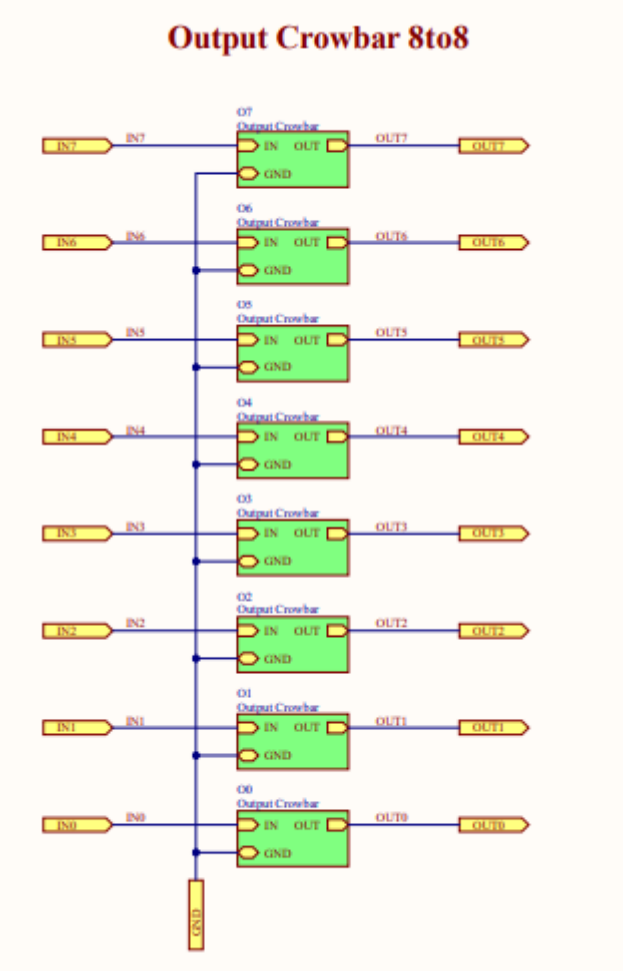
36.29. Schema CAN Interface

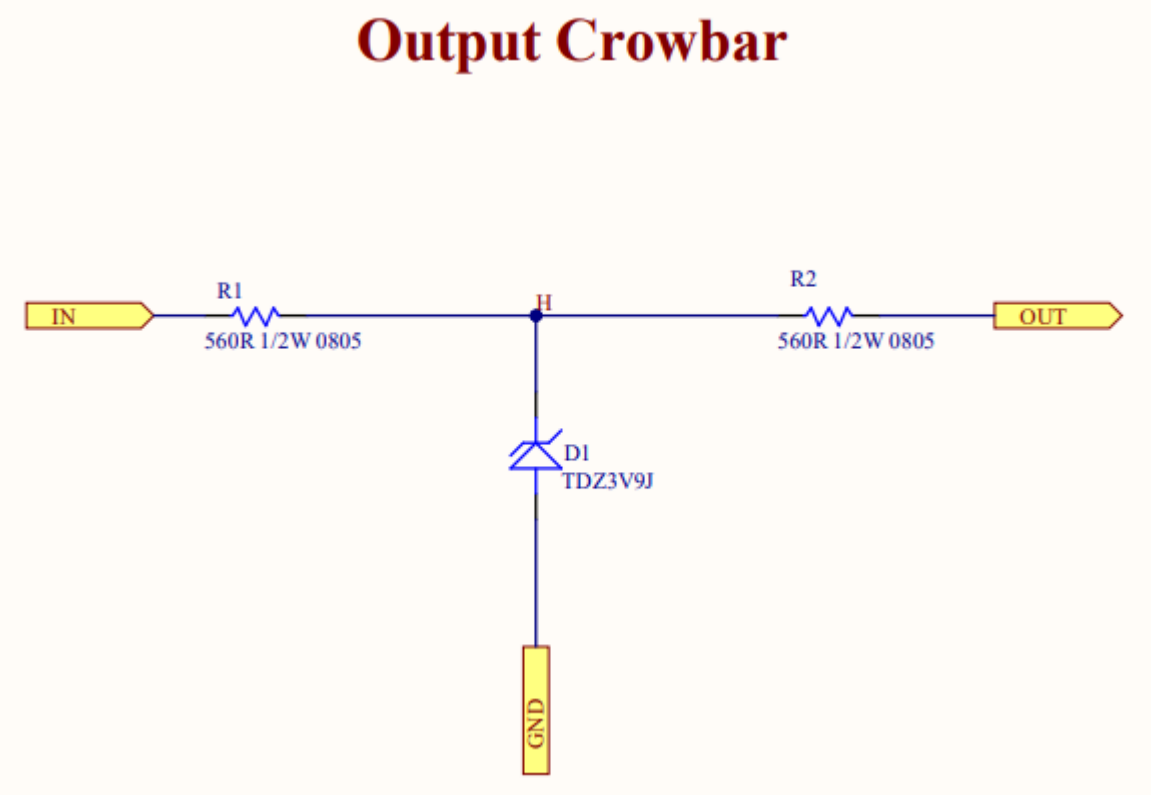
Segue Schema elettrico dell'interfaccia della rete CAN.



36.30. Schema Output Crowbar

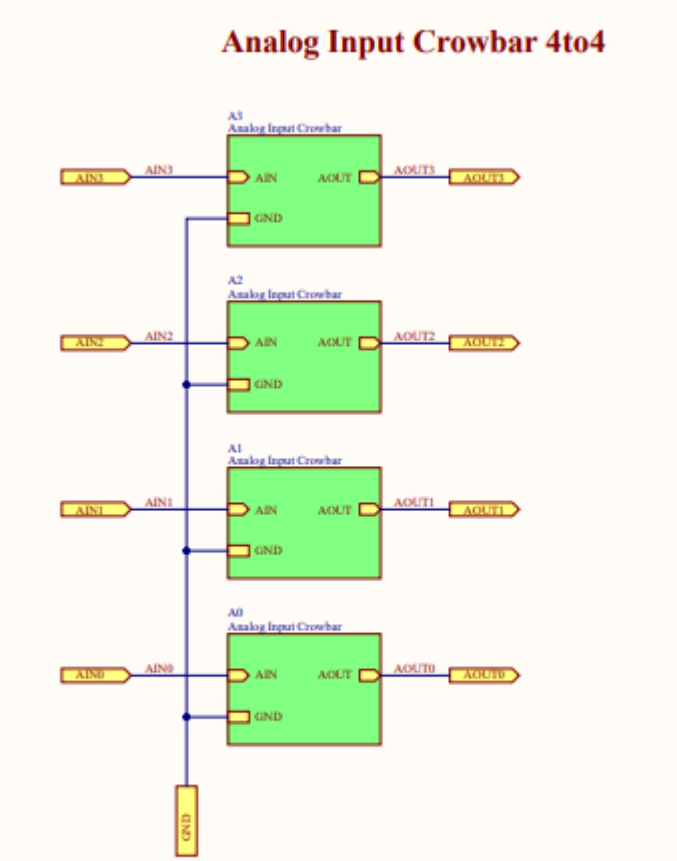
Schema elettrico dell' Output Crowbar.
Inserito zener e restistenze per la protezione dalle sovratensioni.

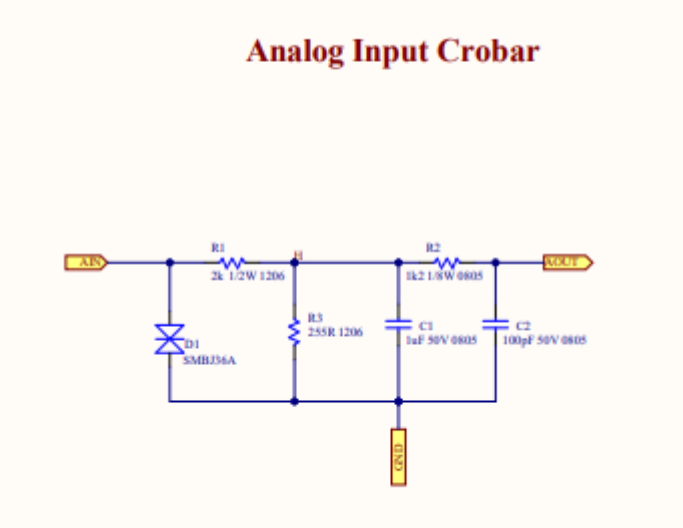




36.31. Schema Analog Input Crowbar

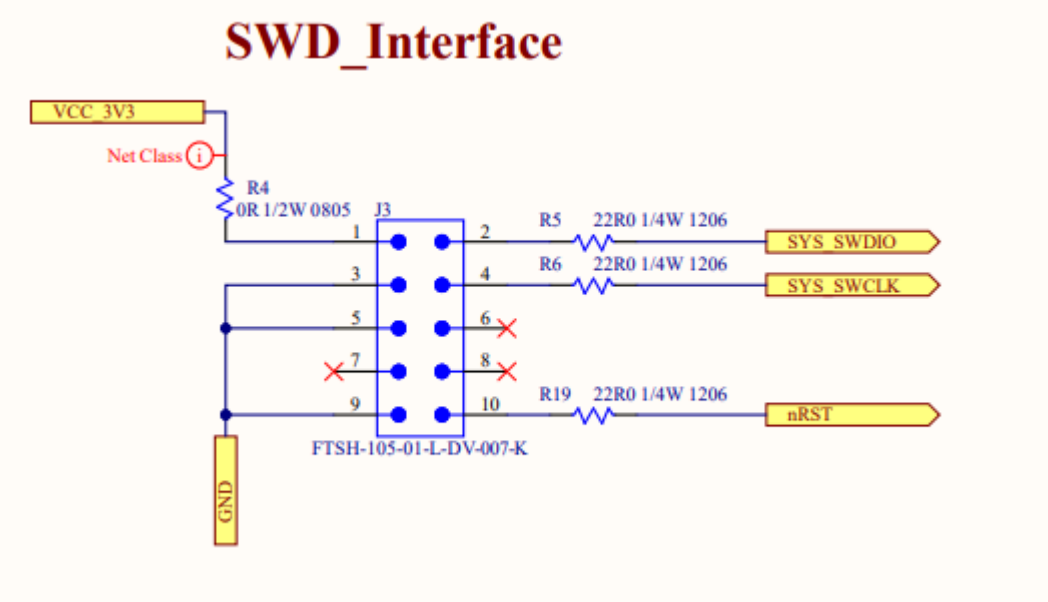
Di seguito lo schema relativo all'Analog Input Crowbar. Inserito transil bidirezione e filtri RC per la protezione sovratensioni e sovracorrenti sugli ingressi analogici.





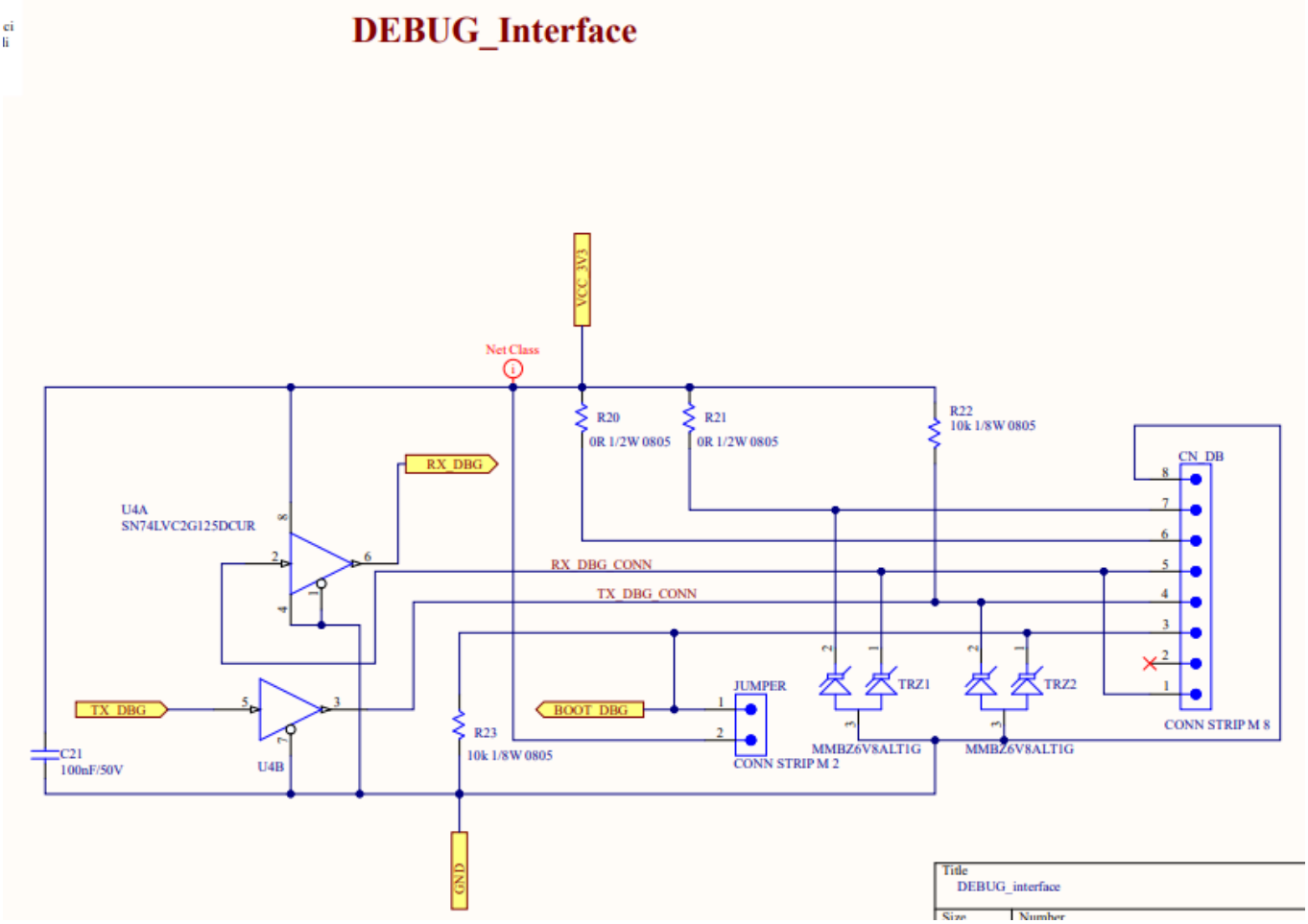
36.32. Schema SWD Interface

segue schema elettrico dell' interfaccia per il debut del software.



36.33. Schema DEBUG Interface

Segue schema elettrico dell'interfaccia di download dell'applicativo, tramite apposito TOOL



36.34. uC groupe

36.34.1. Y1

Signature: -Y1 : Cristal 4
Istanza relativa alla classe

36.34.2. Y2

Signature: -Y2 : ABS07-32.768KHZ-7-T

36.34.3. SW1

Signature: -SW1 : SDM-Switch-2way

36.34.4. D2

Signature: -D2 : LED

36.34.5. D3

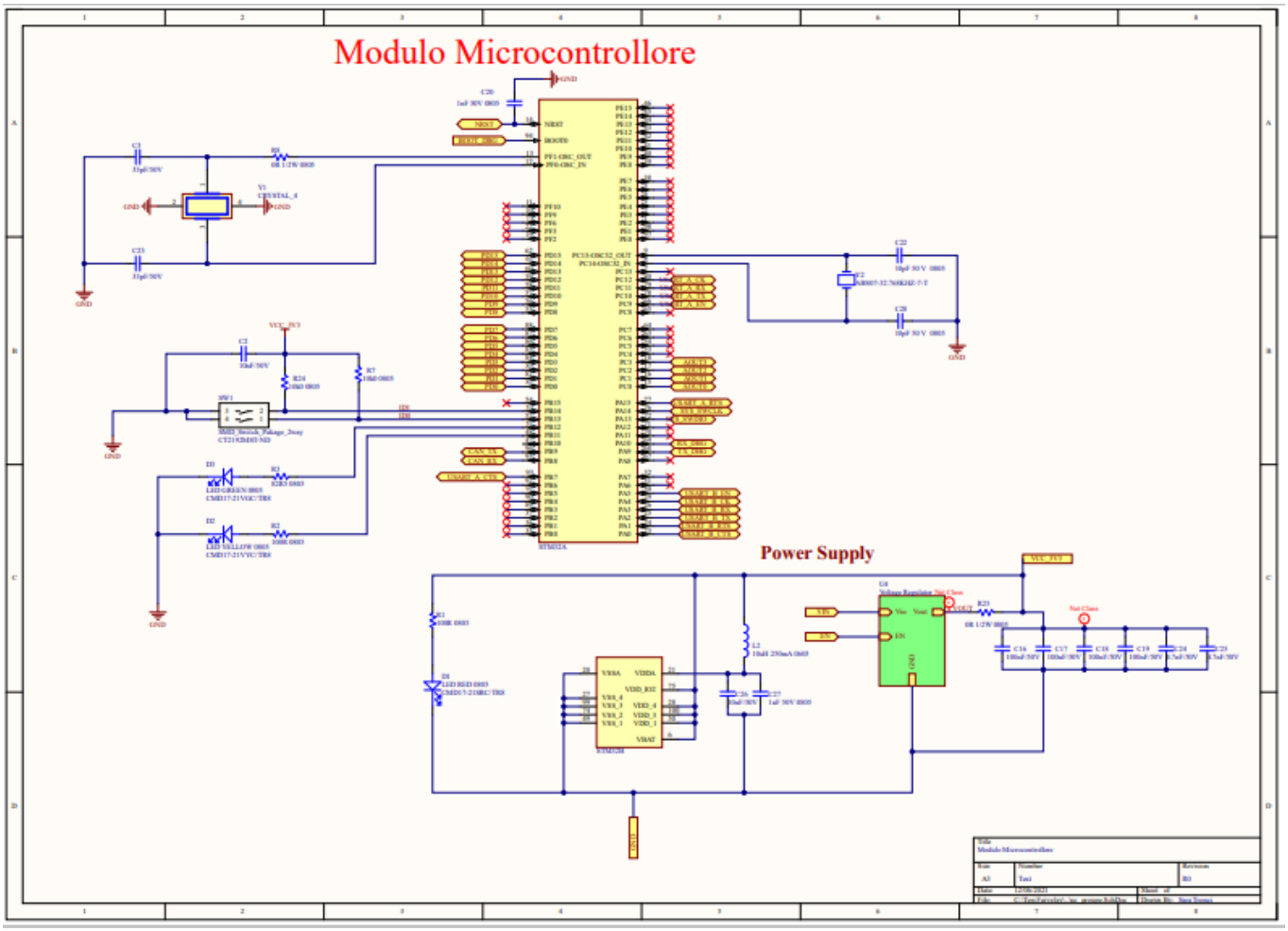
Signature: -D3 : LED

36.34.6. STM32A

Signature: -STM32A : STM32F072

36.35. Schema uC group

Di seguito lo schema relativo al uC groupe che contiene il microcontrollore STM32
Sono stati aggiunti due oscillatori per poter generare in caso di necessità due segnale di clock più precisi di quelli interni.
Il dip switch è necessario per impostare l'ID del Computing Unit quando è montato sul Dimostratore Monopiattaforma.
Questo modulo contiene anche il Voltage Regulator che consente di alimentare il Computing Unit quando non è assemblato al Dimostratore Monopiattaforma.



36.36. Voltage Regulator

Classe Relativa al Voltage Regulator.

Riceve in ingresso un alimentazione Vin compresa tra i 12Vdc e i 36 Vdc e fornisce un uscita di 3.3 Vdc con una tolleranza del 1%.

36.37. STM32F072

36.37.1. PA

Signature: +PA : ParallelPort

36.37.2. PB

Signature: +PB : ParallelPort

36.37.3. PC

Signature: +PC : ParallelPort

36.37.4. PD

Signature: +PD : ParallelPort

36.37.5. PE

Signature: +PE : ParallelPort

36.37.6. PF

Signature: +PF : ParallelPort

36.37.7. adc
Signature: adc : ADC

36.37.8. PWS
Signature: -PWS : Voltage Regulator

36.37.9. BOOT0
Signature: BOOT0()

36.37.10. DAC_OUT2
Signature: DAC_OUT2()

36.37.11. DAC_OUT1
Signature: DAC_OUT1()

36.37.12. TIM3
Signature: TIM3()

36.37.13. TIM2
Signature: TIM2()

36.37.14. TIM1
Signature: TIM1()

36.37.15. USART4
Signature: USART4()

36.37.16. USART3
Signature: USART3()

36.37.17. USART2
Signature: USART2()

36.37.18. USART1
Signature: USART1()

36.37.19. SPI2
Signature: SPI2()

36.37.20. SPI1
Signature: SPI1()

36.37.21. VSSA
Signature: VSSA()

36.37.22. VSS
Signature: VSS()

36.37.23. VDDIO2
Signature: VDDIO2()

36.37.24. VDDA
Signature: VDDA()

36.37.25. VDD
Signature: VDD()

36.37.26. VBAT

Signature: VBAT()

36.37.27. NRST

Signature: NRST()

36.38. Schema Voltage Regulator

Di seguito è mostrato lo schema elettrico del Voltage Regulator,

La configurazione implementata è quella di un regolatore buck-boost con tensione di uscita positiva (alias flyback a uscita positiva), che opera con un duty cycle del 50% fornisce una tensione di uscita di 3,3V con un errore del 1%. Questo regolatore ha la capacità di ridurre (buck) o aumentare (boost) la tensione di uscita per mantenerla costante, dosando il duty cycle in base alle fluttuazioni della tensione in ingresso.

36.39. uC groupe

36.39.1. Attributes

Signature	Description
-Y1 : Cristal 4	Istanza relativa alla classe
-Y2 : ABS07-32.768KHZ-7-T	
-SW1 : SDM-Switch-2way	
-D2 : LED	
-D3 : LED	
-STM32A : STM32F072	

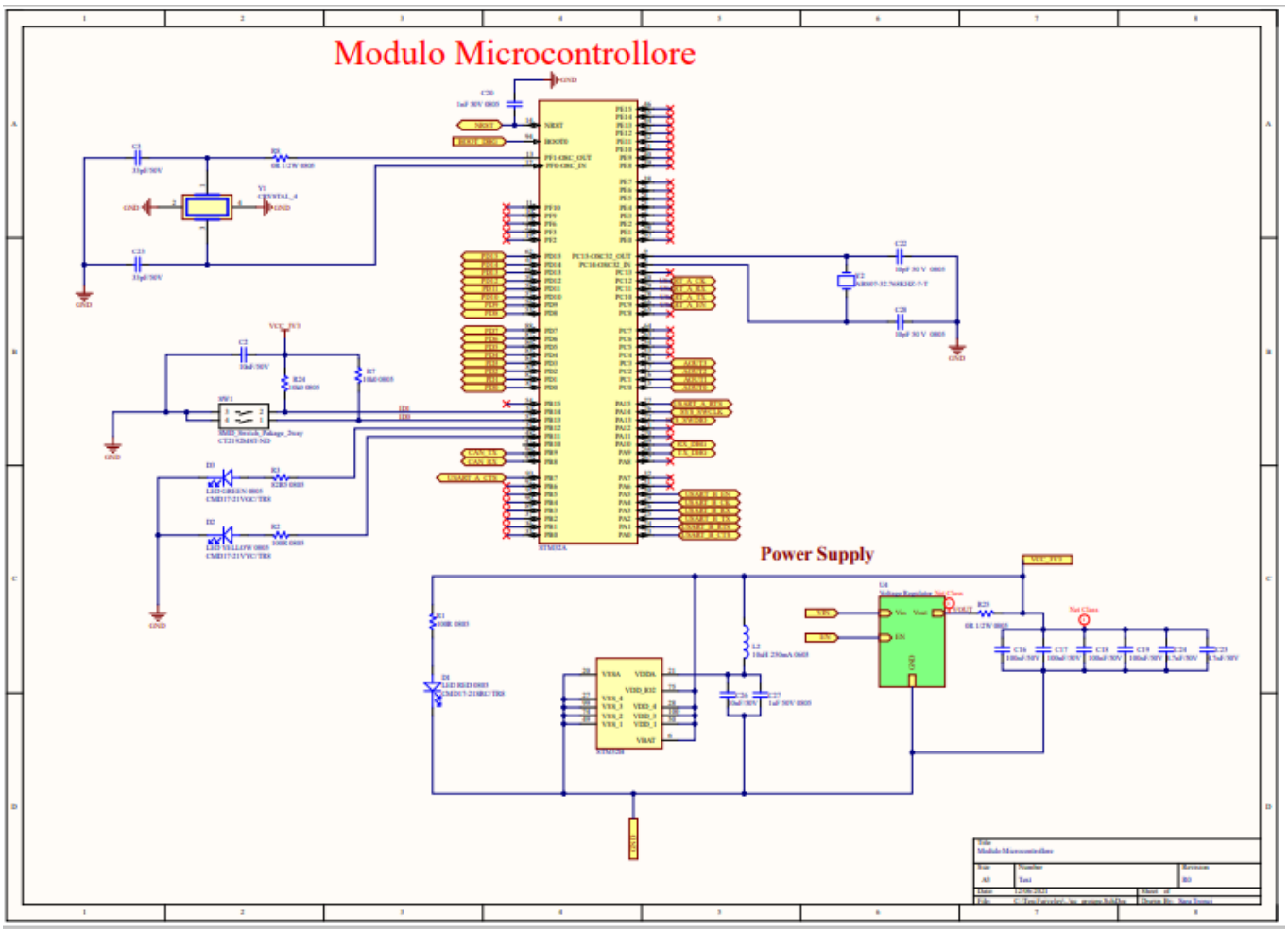
36.40. Schema uC group

Di seguito lo schema relativo al uC groupe che contiene il microcontrollore STM32

Sono stati aggiunti due oscillatori per poter generare in caso di necessità due segnale di clock più precisi di quelli interni.

Il dip switch è necessario per impostare l'ID del Computing Unit quando è montato sul Dimostratore Monopiattaforma.

Questo modulo contiene anche il Voltage Regulator che consente di alimentare il Computing Unit quando non è assemblato al Dimostratore Monopiattaforma.



36.41. Voltage Regulator

Classe Relativa al Voltage Regulator.

Riceve in ingresso un alimentazione Vin compresa tra i 12Vdc e i 36 Vdc e fornisce un uscita di 3.3 Vdc con una tolleranza del 1%.

36.42. STM32F072

36.42.1. Attributes

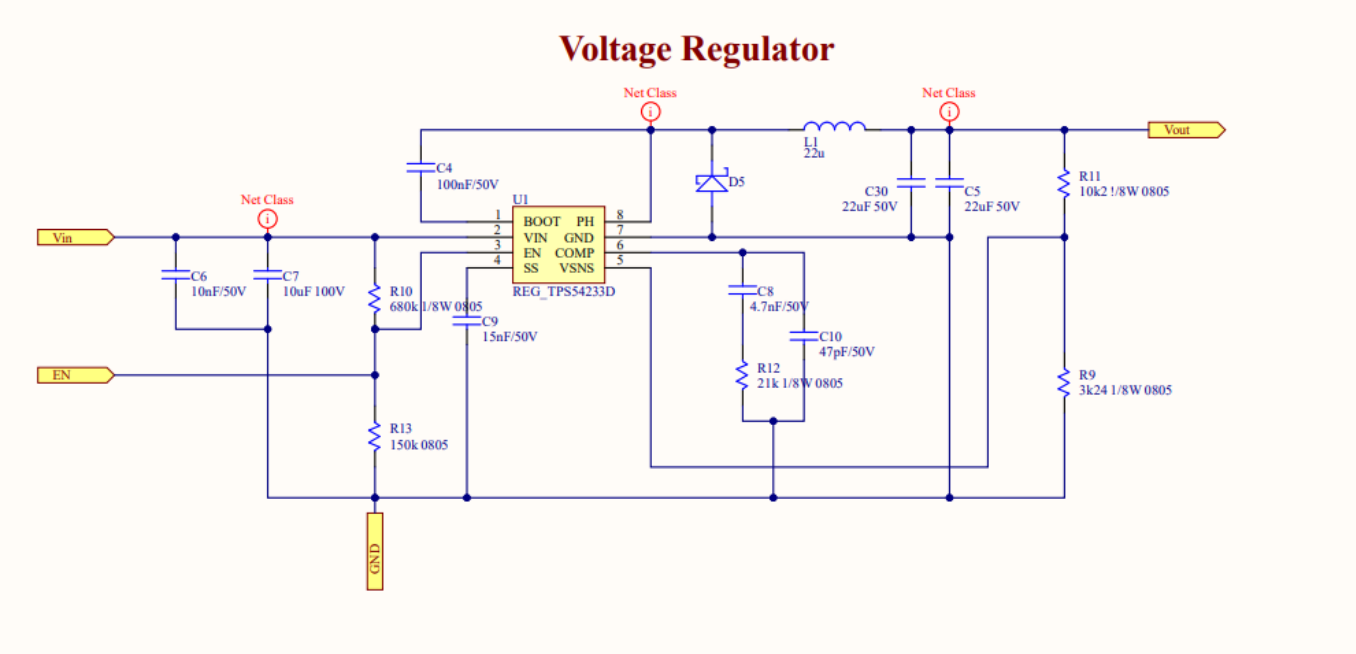
Signature	Description
+PA : ParallelPort	
+PB : ParallelPort	

+PC : ParallelPort	
+PD : ParallelPort	
+PE : ParallelPort	
+PF : ParallelPort	
adc : ADC	
-PWS : Voltage Regulator	

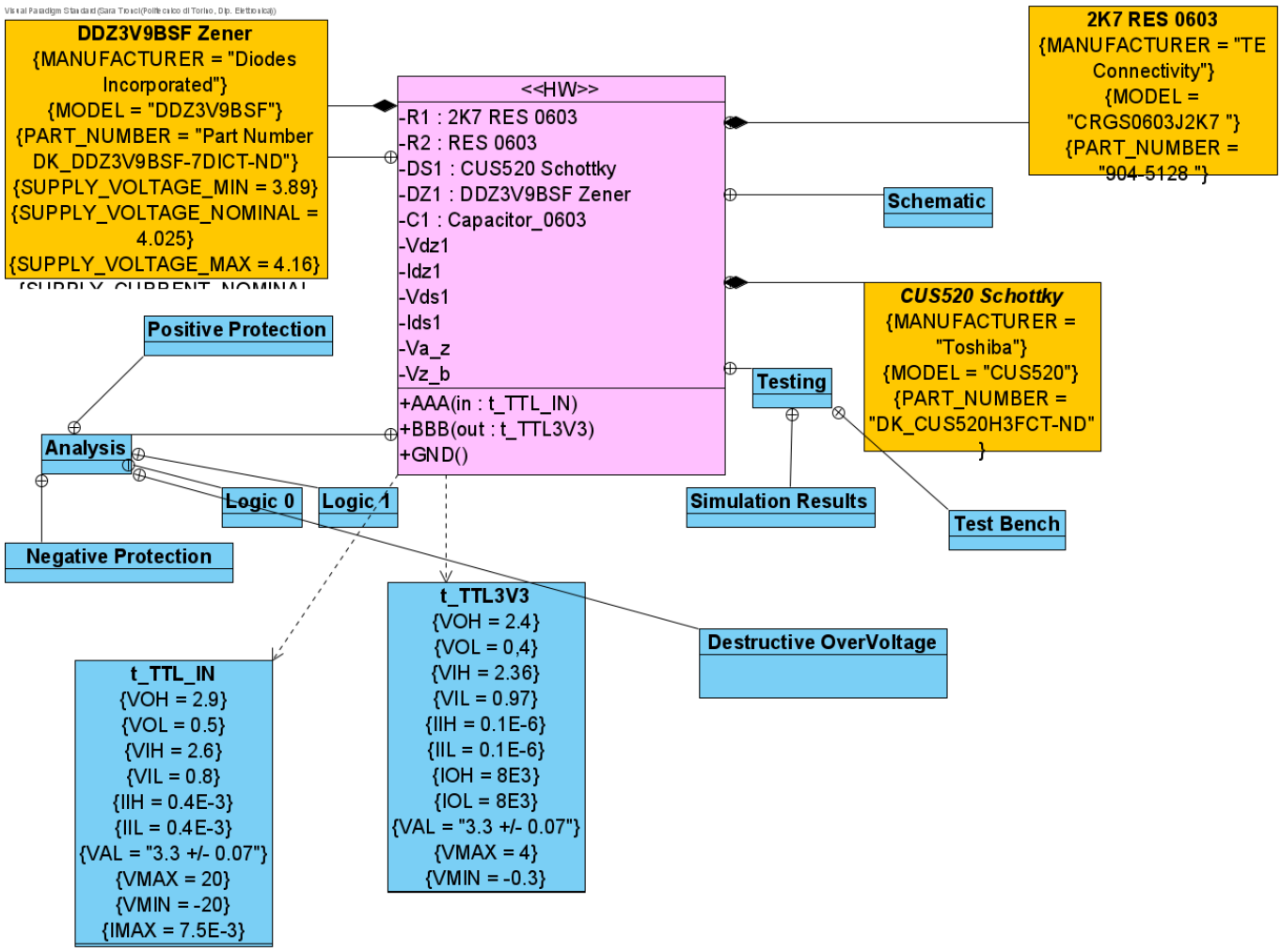
36.43. Schema Voltage Regulator

Di seguito è mostrato lo schema elettrico del Voltage Regulator,

La configurazione implementata è quella di un regolatore buck-boost con tensione di uscita positiva (alias flyback a uscita positiva), che opera con un duty cycle del 50% fornisce una tensione di uscita di 3,3V con un errore del 1%. Questo regolatore ha la capacità di ridurre (buck) o aumentare (boost) la tensione di uscita per mantenerla costante, dosando il duty cycle in base alle fluttuazioni della tensione in ingresso.



37. Input Crowbar



37.1. Negative Protection

In this paragraph, it's analyzed Input Crowbar behavior, when the input is Vmin (t_TTL_IN) <= V(A) <= 0 V,and in this analysis it's defined the diodo type using for negative protection and a constraint on R1.

Before proceeding you have to do some considerations:

- -0,1uA <= I(B) <= +0,1uA
- There aren't constraints on I(A).
- Since the input to the crowbar is a DC voltage, we can neglect capacitor C.
- DS1: CUS520 Schottky is in forward zone, and DZ1: DDZ3V9BSF Zener is in reverse zone.

Since for negative voltage in input Vmin (t_TTL_IN) <= V(A) <= 0 V then must have V(B) => Vmin (t_TTL3V3).

According to Kirchhoff's voltage law V(B) = - (Vds1 + Vz-b). Since I(B) is very low, then I(B) R2 < 10 uV is negligible. Therefore approximately V(B) = - Vds1.

checking the value of the current flowing across the diode under these conditions:

you write according Kirchoff's current law Ids1 = - Ia - Ib + Idz1 , you can see in DDZ3V9BSF Zener datasheet that current reverse Idz1 = 10 uA, and -0,1 uA <= Ib <= 0,1 uA , they are both negligible.

In the worst case Ids1 = - Ia = Imax(t_TTL_IN) = 7,5 mA , and according to CUS520 Schottky datasheet at Ids1 = 10mA it corresponds Vds1 = 0,3 V, then the specification is verify.

Always imposing V(B) => Vmin (t_TTL3V3) and you using Kirchoff's law's, you get it V(B) = V(A) - Va-z = V(A) - IA*R1 => Vmin (t_TTL3V3),
since R1 => (Vmin (t_TTL3V3) - V(A)) /(- I(A)).
If you don't have constraints on I(A), you evaluate as I(A) = Idz1 - Ids1 + Ib , therefore according to specifications R1min = (-0,3V - 0V)/(10uA - 10mA + 10 mA) = 3 Kohm.
The constraint is R1 <= 3 Kohm.
If you choose R1 = 2,7 Kohm, there you obtain -7,5 mA <= I(A) <= +7,5 mA for the negative protection.

Finally checking if the constraints on power are respected:

- R1: 2K7 RES 0603 -> Va-z = V(A) -V(B). You ca verify that the maximum voltage value that drops on R1 is Va-z = -20 + 0,3 = 19,7, since Imax = -7,5, therefore the power on R1 P_R1 = (-20V)*(-7,5mA) = 147, 45 mW <= 250 W according to 2K7 RES 0603 datasheet.
- DZ1: DDZ3V9BSF Zener -> Vdz1 = 0,3 V, and idz1 10uA uA in the worst case, the power on DZ1: DDZ3V9BSF Zener is P_dz1 = 3uW < 0,5 W according to DDZ3V9BSF Zener datasheet.
- DS1: CUS520 Schottky -> Vds1 = -0,3 and Ids1 =

.

37.2. Logic 0

This is the usually operating point for zero level. When voltage input 0 <= V(A) <= VIL (t_TTL_IN), you must have V(B) <= VIL (t_TTL3V3).
In this analysis there are following boundary condition, and approximations :

- The circuit works in DC, you can omit C1: Capacitor_0603, it's to have an open circuit.
- -0,1uA <= I(B) <= +0,1uA
- There are constraints on I(A).
- DS1: CUS520 Schottky and DZ1: DDZ3V9BSF Zener are both in reverse zone.

According to Kirchoff's laws V()

37.3. CUS520 Schottky

Class of CUS520 Schottky, for more details see the documentation.

37.4. DDZ3V9BSF Zener

Class of DDZ3V9BSF Zener, for more details see the documentation.

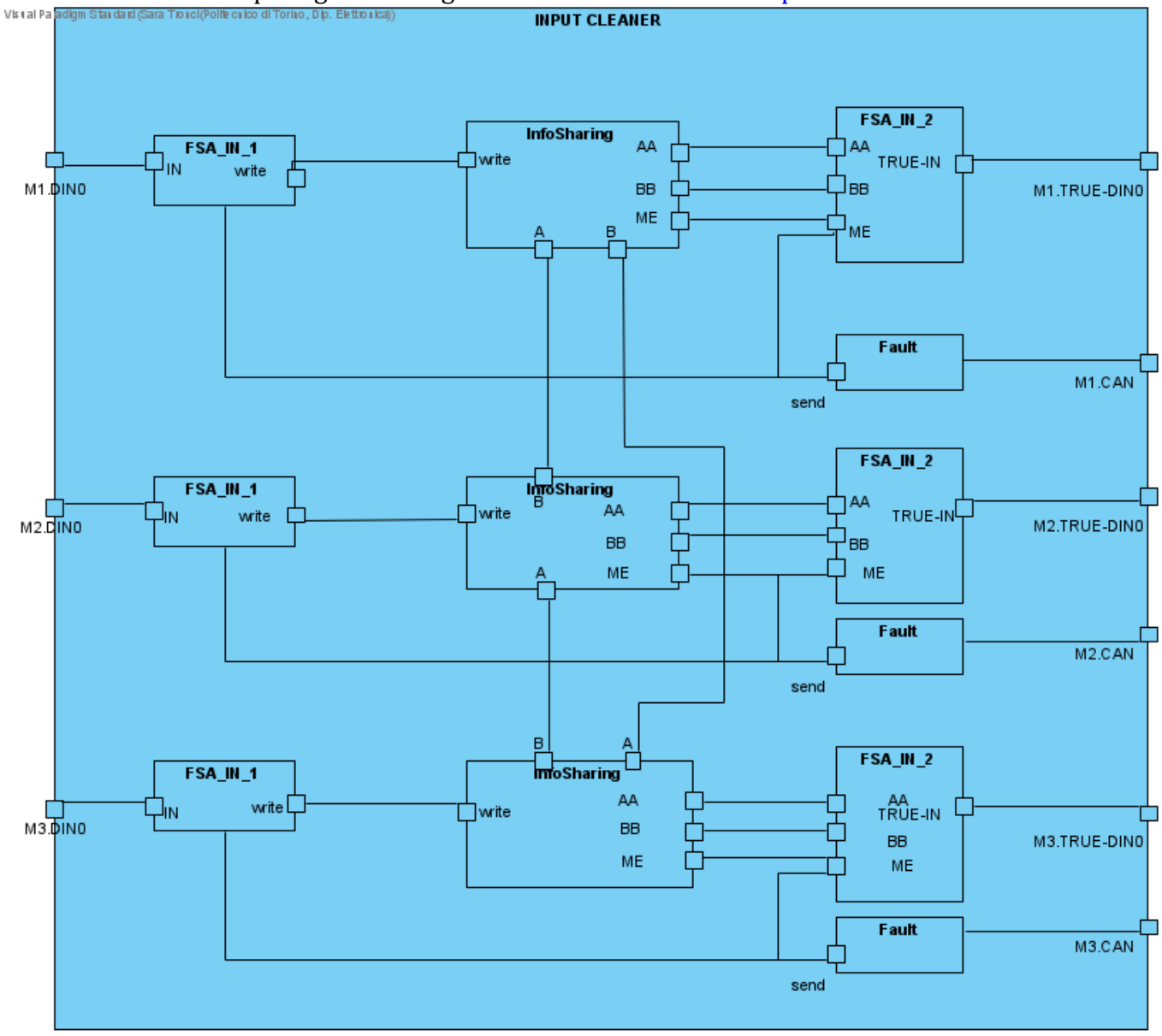
37.5. 2K7 RES 0603

Class of 2K7 RES 0603, for more details see the documentation.

38. Input Cleaner

l seguente diagramma dei componenti descrive com'è dovrebbe essere implementato il modulo INPUT CLEANER.

Il modulo è utilizzato per ogni DI in ingresso al Dimostratore Monopiattaforma.



38.1. INPUT CLEANER

Il seguente modulo, come anticipato nel paragrafo iniziale ha la funzione di filtrare i segnali digitali in ingresso al Dimostratore Monopiattaforma, e di propagare via rete CAN l'informazione relativa al FAULT.

L'INPUT CLEANER è costituito dai seguenti moduli di molteplicità pari a tre, sono uno per ogni Computing Unit presente all'interno del Dimostratore Multipiattaforma:

1. FSA_IN_1 --> Macchina a stati che riceve in ingresso il segnale DI e lo filtra da eventuali bouncing
2. InfoSharing --> MAcchina a stati che riceve in ingresso il segnale filtra e lo confronta con quello degli altri moduli Computing Unit

3. FSA_IN_2 --> Macchina a stati che stabilisce il valore d'uscita confrontanto i tre valori ingresso
4. Fault --> Modulo che rileva se è presente un fault sui segnali d'ingresso.

38.1.1. M1.DIN0

Segnale Digital Input DIN0 ingresso al INPUT CLEANER destinato al Computing Unit M1 e proveniente da un sistema esterno.

38.1.2. M1.CAN

Segnale di fault trasmesso via CAN dal Computing Unit M1.

38.1.3. M1.TRUE-DIN0

Segnale d'uscita del INPUT CLEANER, corrispondente al segnale DIN0 filtrato e corretto che verrà processato dal Computing Unit M1.

38.1.4. M2.DIN0

Segnale Digital Input DIN0 ingresso al INPUT CLEANER destinato al Computing Unit M2 e proveniente da un sistema esterno

38.1.5. M3.DIN0

Segnale Digital Input DIN0 ingresso al INPUT CLEANER destinato al Computing Unit M3 e proveniente da un sistema esterno.

38.1.6. M2.CAN

Segnale di fault trasmesso via CAN dal Computing Unit M1.

38.1.7. M2.TRUE-DIN0

Segnale d'uscita del INPUT CLEANER, corrispondente al segnale DIN0 filtrato e corretto che verrà processato dal Computing Unit M2.

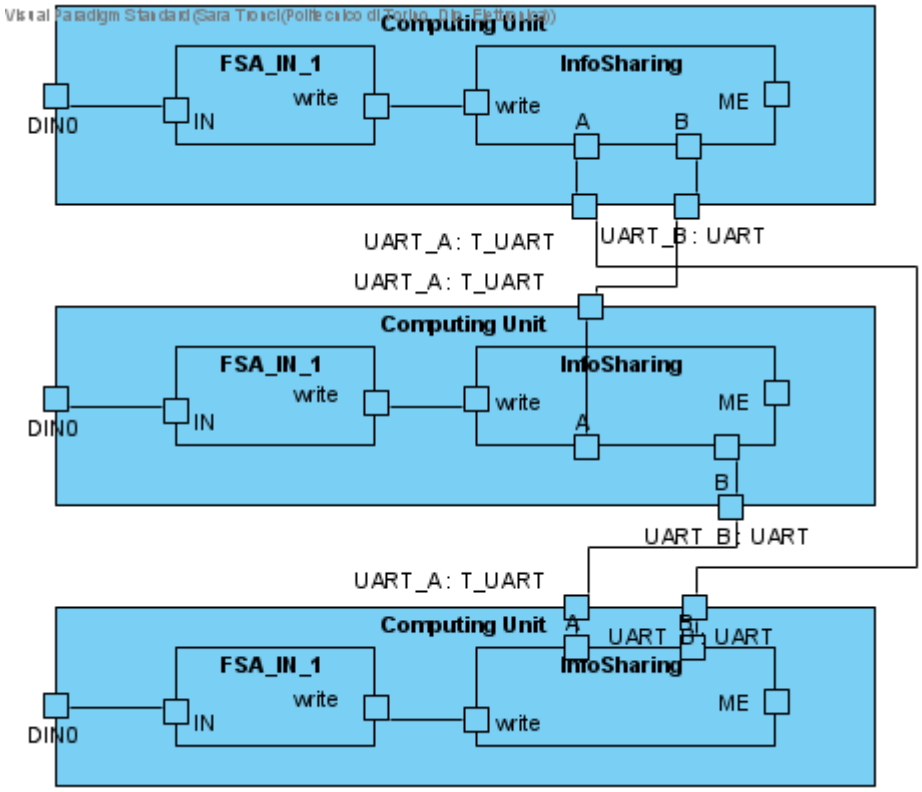
38.1.8. M3.CAN

Segnale di fault trasmesso via CAN dal Computing Unit M2.

38.1.9. M3.TRUE-DIN0

Segnale d'uscita del INPUT CLEANER, corrispondente al segnale DIN0 filtrato e corretto che verrà processato dal Computing Unit M2.

39. Input Cleaner - Physical

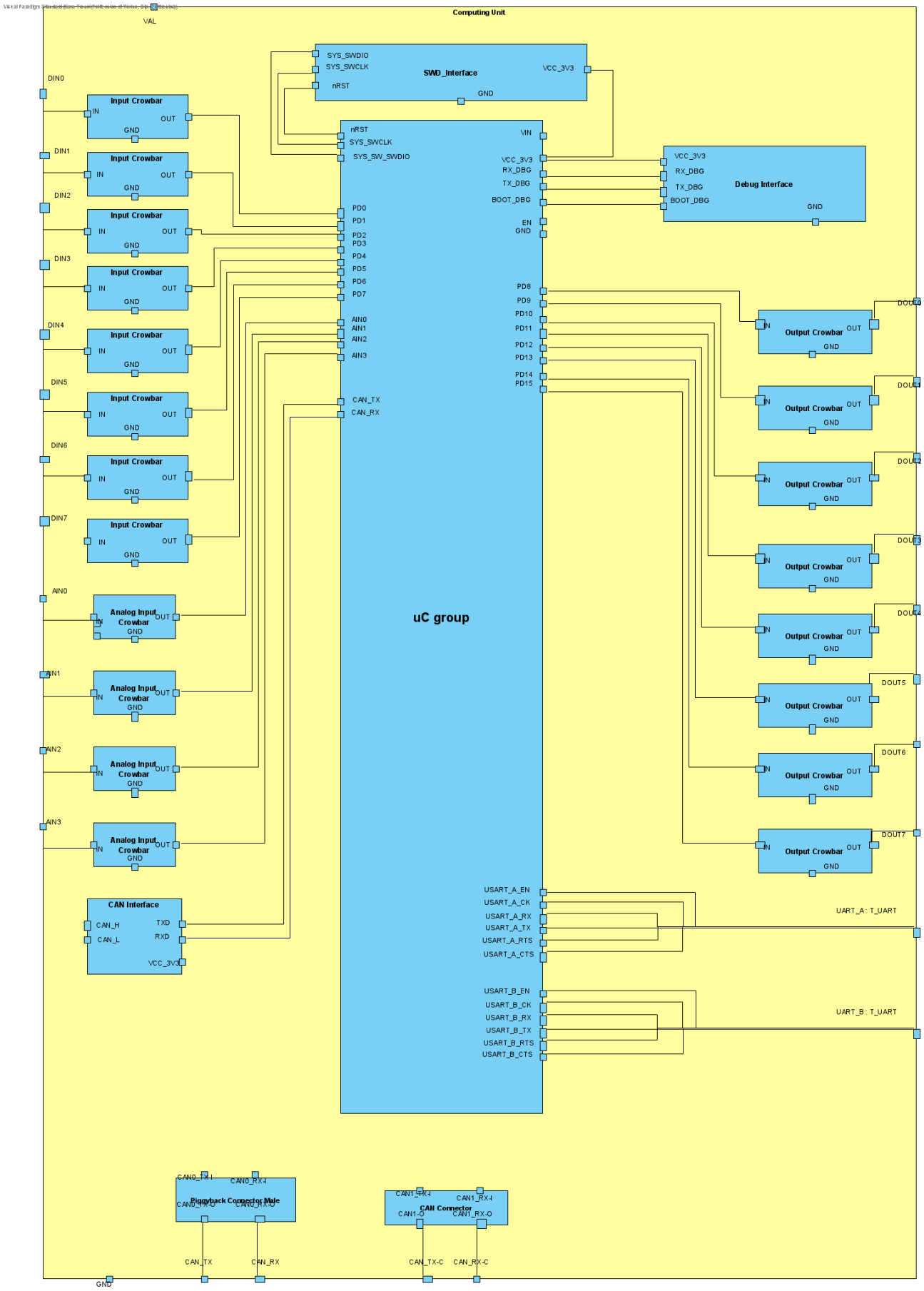


40. Modulo uC

Il Computing Unitmostra l'architettura della scheda piggy back realizzata, che utilizza il microcontrollore STM32F072VB.

Allinterno del modulo sono presenti:

1. Il modulo STM32F072VB parte di controllo della scheda, include il STM32F072VB e i componenti che ne consentono l'attivazione:
   1.1. Il Voltage Regulator che fornisce al uC un alimentazione nominale di 3V3 con un errore dell'1%
   1.2. il modulo Oscillators, contenente due oscillatori esterni al quarzo per impostare ad esempio la frequenza della UART, o comunque per ottenere a seconda dei casi una frequenza di CLK più precisa, rispetto all'oscillatore interno al microcontrollore.
   1.3.


Computing Unit
VAL
SWD_Interface
SYS_SWDIO
SYS_SWCLK
nRST
VCC_3V3
GND
uC group
Debug Interface
RX_DBG
TX_DBG
BOOT_DBG
Input Crowbar
Analog Input Crowbar
Output Crowbar
CAN Interface
CAN_H
CAN_L
TXD
RXD
DIN0
DIN1
DIN2
DIN3
DIN4
DIN5
DIN6
DIN7
AIN0
AIN1
AIN2
AIN3
DOUT0
DOUT1
DOUT2
DOUT3
DOUT4
DOUT5
DOUT6
DOUT7
USART_A_EN
USART_A_CK
USART_A_RX
USART_A_TX
USART_A_RTS
USART_A_CTS
USART_B_EN
USART_B_CK
USART_B_RX
USART_B_TX
USART_B_RTS
USART_B_CTS
UART_A: T_UART
UART_B: T_UART
Piggyback Connector Male
CAN Connector
CAN_TX
CAN_RX
CAN_TX-C
CAN_RX-C

40.1. uC group

Il modulo uC group costituisce la parte di controllo del Computing Unit. AL suo interno è presente:

- un microprocessore STM32F072 che gestisce le funzioni del modulo
- Un Voltage Regulator che alimenta il STM32F072 e tutti i moduli attivi all'interno del Computing Unit
- Un Modulo Oscillators contenete due oscillatori al quarzo

40.1.1. PD0

Pin per il segnale DI0 del modulo uC group, connesso al pin PD0 del STM32F072VB.

40.1.2. PD1

Pin per il segnale Digital Input DI1 del modulo uC group, connesso al pin PD1 del STM32F072VB, impostato come DI1

40.1.3. PD2

Pin per il segnale Digital Input DI2 del modulo uC group, connesso al pin PD2 del STM32F072VB.

40.1.4. PD3

Pin per il segnale Digital Input DI3 del modulo uC group, connesso al pin PD3 del STM32F072VB.

40.1.5. PD4

Pin per il segnale Digital Input DI4 del modulo uC group, connesso al pin PD4 del STM32F072VB.

40.1.6. PD5

Pin per il segnale Digital Input DI5 del modulo uC group, connesso al pin PD5 del STM32F072VB.

40.1.7. PD6

Pin per il segnale Digital Input DI6 del modulo uC group, connesso al pin PD6 del STM32F072VB.

40.1.8. PD7

Pin per il segnale Digital Input DI7del modulo uC group, connesso al pin PD7 del STM32F072VB.

40.1.9. AIN0

Analog input AI0 del modulo uC group, connesso al pin PC0 del STM32F072VB.

40.1.10. AIN1

Pin per il segnale Analog Input AIN1 del modulo uC group, connesso al pin PC1 del STM32F072VB.

40.1.11. AIN2

Pin per il segnale Analog Input AIN2 del modulo uC group, connesso al pin PC2 del STM32F072VB.

40.1.12. AIN3

Pin per il segnale Analog Input AIN3 del modulo uC group, connesso al pin PC3 del STM32F072VB.

40.1.13. PD8

Pin per il segnale di tipo Digital Output del modulo uC group, connesso al pin PD8 del STM32F072VB.

40.1.14. PD9

Pin per il segnale di tipo Digital Output del modulo uC group, connesso al pin PD9 del STM32F072VB.

40.1.15. PD10

Pin per il segnale di tipo Digital Output del modulo uC group, connesso al pin PD10 del STM32F072VB.

40.1.16. PD11

Pin per il segnale di tipo Digital Output del modulo uC group, connesso al pin PD11 del STM32F072VB.

40.1.17. PD12

Pin per il segnale di tipo Digital Output del modulo uC group, connesso al pin PD12 del STM32F072VB.

40.1.18. PD13

Pin per il segnale di tipo Digital Output del modulo uC group, connesso al pin PD13 del STM32F072VB.

40.1.19. PD14

Pin per il segnale di tipo Digital Output del modulo uC group, connesso al pin PD14 del STM32F072VB.

40.1.20. PD15

Pin per il segnale di tipo Digital Output del modulo uC group, connesso al pin PD15 del STM32F072VB.

40.1.21. CAN_TX

Pin del uC group dedicato al segnale di trasmissione della rete CAN. IL pin è connesso internamente al connesso al pin PB9 del STM32F072VB.

Il pin è collegato al pinTXD del modulo CAN Interface.

40.1.22. CAN_RX

Pin del uC group dedicato al segnale di ricezione della rete CAN. IL pin è connesso internamente al connesso al pin PB8 del STM32F072VB.

Il pin è collegato al pin RXD del modulo CAN Interface.

40.1.23. nRST

Pin dedicato segnale di reset del del uC group. Il segnale è attivo basso.

40.1.24. SYS_SWCLK

Pin dedicato al segnale di clock utilizzato per il debug del uC group.

40.1.25. SYS_SW_SWDIO

Pin dedicato al segnale di Data Input usato durante il debug del uC group

40.1.26. VIN

Pin per il segnale Digital Input del modulo uC group, connesso al pin PD0 del STM32F072VB.

40.1.27. VCC_3V3

Pin per il segnale Digital Input del modulo uC group, connesso al pin PD1 del STM32F072VB, impostato come DI1

40.1.28. EN

Pin dedicato al segnale che abilita l'alimentazione al uC group,

40.1.29. GND

Pin per il segnale Digital Input del modulo uC group, connesso al pin PD2 del STM32F072VB.

40.1.30. USART_A_EN

Pin del DO della UARTA che assume il valore 1 se la UARTA è abilitata, '0' se disabilita.

40.1.31. USART_A_CK

Pin collegato al segnale di clock del canale di comunicazione seriale UARTA.

40.1.32. USART_A_RX

Pin collegato al segnale di ricezione RX del canale di comunicazione seriale UARTA.

40.1.33. USART_A_TX

Pin collegato al segnale di trasmissione del canale di comunicazione seriale UARTA.

40.1.34. USART_A_RTS

Pin collegato al segnale RTS del canale di comunicazione seriale UARTA.

40.1.35. USART_A_CTS

Pin collegato al segnale CTS del canale di comunicazione seriale UARTA.

40.1.36. USART_B_EN

Pin del DO della UARTB che assume il valore 1 se la UARTB è abilitata, '0' se disabilita.

40.1.37. USART_B_CK

Pin collegato al segnale di clock del canale di comunicazione seriale UARTB.

40.1.38. USART_B_RX

Pin collegato al segnale di trasmissione RX del canale di comunicazione seriale UARTB
.

40.1.39. USART_B_TX

Pin collegato al segnale di trasmissione TX del canale di comunicazione seriale UARTB.

40.1.40. USART_B_RTS

Pin collegato al segnale RTS del canale di comunicazione seriale UARTB.

40.1.41. USART_B_CTS

Pin collegato al segnale CTS del canale di comunicazione seriale UARTB.

40.1.42. RX_DBG

Pin del uC group dedicato al segnale RX per l'interfaccia di download del del SW applicativo.

40.1.43. TX_DBG

Pin del uC group dedicato al segnale TX per l'interfaccia di download del del SW applicativo

40.1.44. BOOT_DBG

pin dedicato al segnale di attivazione della programmazione

40.2. Output Crowbar

Il Modulo Output Crowbar protegge il Digital Output del Computing Unit dalle sovracorrenti.

40.2.1. OUT

Pin dell'Output Crowbar che propaga il segnale Digital Output filtrato.

40.2.2. IN

Pin a cui è connesso un Digital Output del STM32F072VB.

40.2.3. GND

Pin di riferimento comune del Modulo Output Crowbar.

40.3. Analog Input Crowbar

Il Modulo Analog Input Crowbar protegge il segnale Analogico in ingresso al Computing Unit da sovratensioni e sovracorrenti.

40.3.1. GND

Pin di riferimento comune del Modulo Analog Input Crowbar.

40.3.2. OUT

Pin d'uscita del Modulo Analog Input Crowbar, che propaga il segnale filtrato e prottetto. Il pin si connette a uno dei pin del STM32F072VB (PC0- PC1 - PC2 - PC3) configurati come Analog Input.

40.3.3. IN

Pin del Modulo Analog Input Crowbar a cui è connesso il segnale Analog Input esterno del Computing Unit.

40.3.4. IN

Pin del Modulo Analog Input Crowbar a cui è connesso il segnale Analog Input esterno del Computing Unit.

40.4. Input Crowbar

Il Modulo Input Crowbar protegge il segnale Digitale in ingresso al Computing Unit da sovratensioni e sovracorrenti.

40.4.1. OUT

Pin d'uscita del Modulo Input Crowbar, che propaga il segnale filtrato e prottetto. Il pin si connette a uno dei pin del STM32F072VB (PD0 - PD1 - PD2 - PD3 - PD4 - PD5 - PD6 - PD7) configurati come Digital Input.

40.4.2. IN

Pin del Modulo Input Crowbar a cui è connesso il segnaleDigital Input esterno del Computing Unit.

40.4.3. GND

Pin di riferimento comune del Modulo Input Crowbar.

40.5. SWD_Interface

Interfaccia di debug del Computing Unit. Con un apposito tool attraverso questa interfaccia è possibile effettuare il debug del Computing Unit.

Per maggiori informazioni sull'interfaccia di debug si veda il datasheet del STM32F072.

40.5.1. VCC_3V3

Pin del SWD_Interface dedicato al riferimento positivo dell'alimentazione di 3.3V nominali. Il pin è connesso al pin VCC_3V3 del uC group

40.5.2. SYS_SWDIO

Pin dedicato al segnale Data Input per il debug del Computing Unit.

40.5.3. SYS_SWCLK

Pin dedicato al segnale di CLK per il debug del Computing Unit.

40.5.4. nRST

Pin dedicato al segnale di reset del Computing Unit. Il segnale è attivo basso.

40.5.5. GND

Pin del SWD_Interface dedicato al riferimento comune (GND)dell'alimentazione. Il pin è connesso al pin GND del uC group

40.6. Debug Interface

Interfaccia di programmazione del Computing Unit. Consente, tramite l'apposito tool connesso al PC di effettuare il Download dell'applicativo nel Computing Unit.

40.6.1. VCC_3V3

Pin del Modulo Debug Interface connesso al pin di alimentazione VCC_3V3 del uC group.

40.6.2. RX_DBG

Pin dedicato al segnale di ricezione del Debug Interface.

40.6.3. TX_DBG

Pin dedicato al segnale di trasmissione del Debug Interface.

40.6.4. GND

Pin del Debug Interface dedicato al riferimento GND.

40.7. CAN Interface

Modulo di interfaccia della rete rete CAN.

40.7.1. TXD

Pin dedicato al segnale di trasmissione della CAN e connesso al pin CAN_TX del Modulo uC group.

40.7.2. RXD

Pin dedicato al segnale di ricezione della rete CAN e connesso al pin (model element not found) del Modulo uC group.

40.7.3. VCC_3V3

Pin dedicato all'alimentazione in ingresso al Modulo CAN Interface e connesso al pin VCC_3V3 del uC group (che è internamente collegato al Voltage Regulator)

40.7.4. CAN_L

Pin del CAN Interface connesso al segnale CAN_L

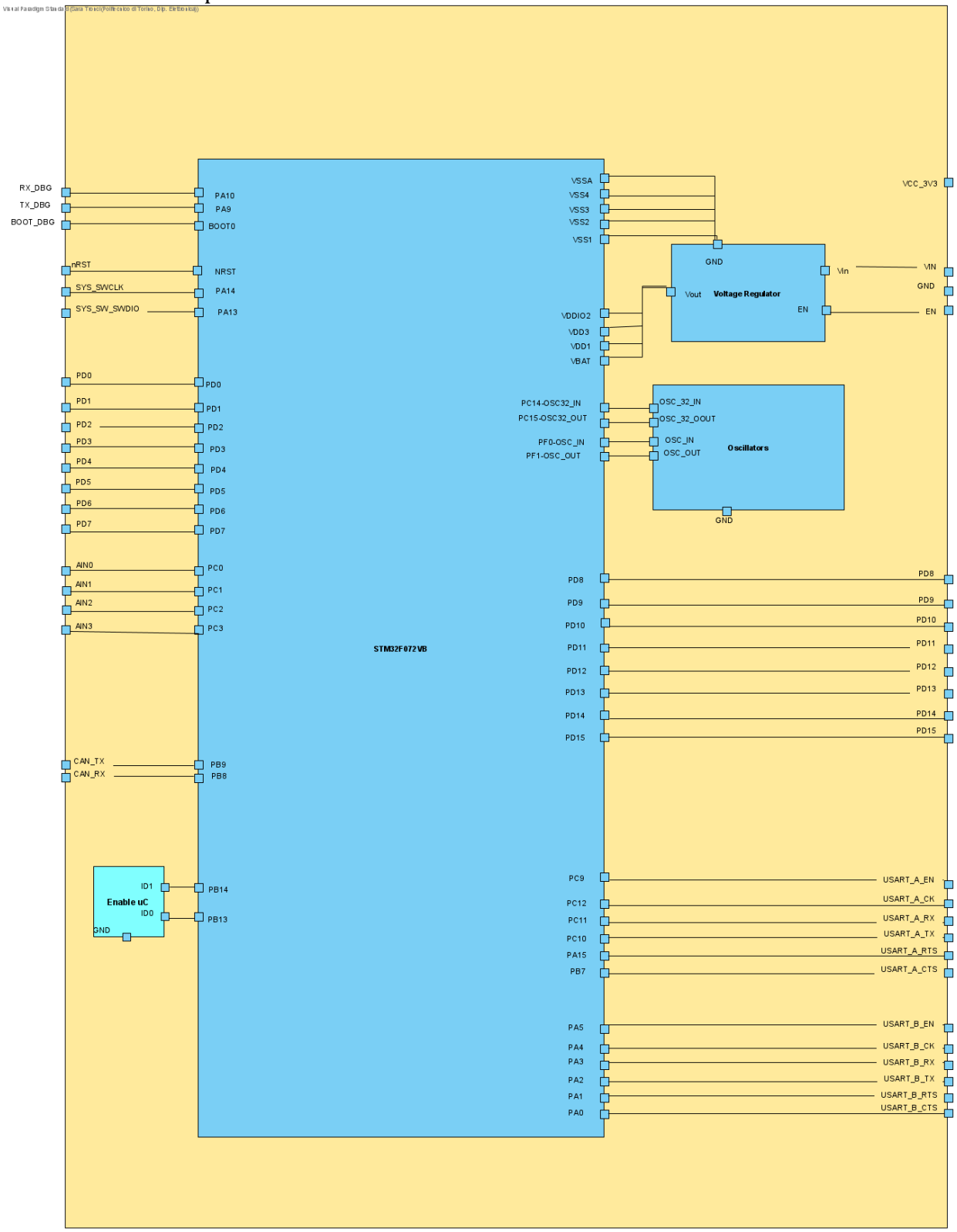
40.7.5. CAN_H

Pin del CAN Interface connesso al segnale CAN_H

41. uC group

Regolatore di tensione che riceve ingresso una tensione compresa tra i 12V e i 36V e fornisce un uscita di 3.3V nominali +/- 1%.

il modulo è utilizzato per alimentare il STM32F072VB e il CAN Interface.



41.1. Oscillators

Modulo contenente due oscillatori al quarzo.

41.2. Voltage Regulator

Regolatore di tensione, che adatta la tensione in ingrasso, compresa tra i 6 e i 20 Volt, a quella che deve alimentare il STM32F072VB, e deve essere di 3.3 Volt.

Inserire schema qui o nel diagramma delle classi relativo?

41.2.1. EN

Pin dedicato al segnale di abilitazione del Voltage regulator

41.3. Enable uC

All'interno del Dimostratore Monopiattaforma sono presenti tre Computing Unit.
ll modulo Enable uC serve a impostare l'ID di ciascun Computing Unit, spostando gli interrutori del DIP switch:

- ID1 = OFF ; ID0 = ON ==> ID = "01"
- ID1 = ON ; ID0 = ON ==> ID = "01"
- ID1 = OFF ; ID0 = ==> ID = "01"
- ID1 = OFF ; ID0 = ON ==> ID = "01"

41.4. STM32F072VB

DESCRIZIONE SPECIFICHE MICRO

41.4.1. PD0

General-purpose I/O pin.
Alternate functions: SPI2_NSS, I2S2_WS, CAN_RX.
Il pin è impostato come Digital Input segnale pull up a media velocità

41.4.2. NRST

Pin dedicato al segnale di Reset del STM32F072VB

41.4.3. PA3

General-purpose I/O pin.
Alternate functions: USART2_RX,TIM2_CH4, TIM15_CH2, TSC_G1_IO4 ADC_IN3, COMP2_INP
Per l'applicativo in uso il pin è impostato come segnale USART2_RX.

41.4.4. PA4

General-purpose I/O pin.
Alternate functions: SPI1_NSS, I2S1_WS, TIM14_CH1, TSC_G2_IO1, USART2_CK COMP1_INM4, MP2_INM4, ADC_IN4, DAC_OUT1
Per l'applicativo in uso il pin è impostato come segnale USART2_CK.

41.4.5. PA2

General-purpose I/O pin.
Alternate functions: USART2_TX,

COMP2_OUT, TIM2_CH3, TIM15_CH1, TSC_G1_IO3 ADC_IN2, COMP2_INM6, WKUP4.
Per l'applicativo in uso il pin è impostato come segnale USART2_TX.

41.4.6. PA5

General-purpose I/O pin.
Alternate functions: SPI1_SCK, I2S1_CK, CEC, TIM2_CH1_ETR, TSC_G2_IO2, COMP1_INM5, COMP2_INM5, ADC_IN5, DAC_OUT2
Per l'applicativo in uso il pin è impostato come segnale di Digital Output e viene commutato a '1' se la USART è abilitata, a '0' se risulta disabilitata.

41.4.7. PA6

General-purpose I/O pin.
Alternate functions: SPI1_MISO, I2S1_MCK, TIM3_CH1, TIM1_BKIN, TIM16_CH1, COMP1_OUT, TSC_G2_IO3, EVENTOUT, USART3_CTS, ADC_IN6

41.4.8. PA12

General-purpose I/O pin.
Alternate functions: CAN_TX, USART1_RTS, TIM1_ETR, COMP2_OUT, TSC_G4_IO4, EVENTOUT, USB_DP

41.4.9. PA13

General-purpose I/O pin.
Alternate functions: IR_OUT, SWDIO, USB_NOE
Il pin è impostato come SWDIO, segnale di data input output utilizzato per il debug

41.4.10. PB1

General-purpose I/O pin.
Alternate functions: TIM3_CH4, USART3_RTS, TIM14_CH1, TIM1_CH3N, TSC_G3_IO3, ADC_IN9

41.4.11. PA9

General-purpose I/O pin.
Alternate functions: USART1_TX, TIM1_CH2, TIM15_BKIN, TSC_G4_IO1
il pin è abilitato come USART1_TX in quanto l'interfaccia di dowload del SW utilizza l'interfaccia seriale della UART_1

41.4.12. PA7

General-purpose I/O pin.
Alternate functions: SPI1_MOSI, I2S1_SD, TIM3_CH2, TIM14_CH1, TIM1_CH1N, TIM17_CH1, COMP2_OUT, TSC_G2_IO4, EVENTOUT, ADC_IN7

41.4.13. PA8

General-purpose I/O pin.
Alternate functions: USART1_CK, TIM1_CH1, EVENTOUT, MCO, CRS_SYNC

41.4.14. PA0

General-purpose I/O pin.
Alternate functions: USART2_CTS, TIM2_CH1_ETR, COMP1_OUT, SC_G1_IO1, USART4_TX, RTC_ TAMP2, WKUP1, ADC_IN0, COMP1_INM6
Per l'applicativo in uso il pin è impostato come segnale USART2_CTS.

41.4.15. PA1

General-purpose I/O pin.
Alternate functions: USART2_RTS, TIM2_CH2, TIM15_CH1N, TSC_G1_IO2, USART4_RX, EVENTOUT, C_IN1, OMP1_INP
Per l'applicativo in uso il pin è impostato come segnale USART2_RTS.

41.4.16. PA10

General-purpose I/O pin.
Alternate functions: USART1_RX, TIM1_CH3, TIM17_BKIN, TSC_G4_IO2
il pin è abilitato come USART1_RX in quanto l'interfaccia di dowload del SW utilizza la ricezione seriale della UART

41.4.17. PA14

General-purpose I/O pin.
Alternate functions: USART2_TX, SWCLK
Il pin è impostato come USART2_TX

41.4.18. PA15

General-purpose I/O pin.
Alternate functions: SPI1_NSS, I2S1_WS, T25USART2_RX, USART4_RTS, TIM2_CH1_ETR, EVENTOUT
Per l'applicativo in uso il pin è impostato come segnale USART4_RTS.

41.4.19. PB0

General-purpose I/O pin.
Alternate functions: TIM3_CH3, TIM1_CH2N, TSC_G3_IO2, EVENTOUT, USART3_CK
ADC_IN8

41.4.20. PF9

General-purpose I/O pin.
Alternate functions: TIM15_CH1

41.4.21. PF6

General-purpose I/O pin. No alternate functions.

41.4.22. PF3

General-purpose I/O pin.
Alternate functions: EVENTOUT

41.4.23. PF2

General-purpose I/O pin.
Alternate functions: EVENTOUT, WKUP8

41.4.24. PF10

General-purpose I/O pin.
Alternate functions: TIM15_CH2

41.4.25. PE9

General-purpose I/O pin.
Alternate functions: TIM1_CH1

41.4.26. PE8

General-purpose I/O pin.
Alternate functions: TIM1_CH1N

41.4.27. PE7

General-purpose I/O pin.
Alternate functions: TIM1_ETR

41.4.28. PE6

General-purpose I/O pin.
Alternate functions: TIM3_CH4 WKUP3, RTC_TAMP3

41.4.29. PE5
General-purpose I/O pin.
Alternate functions: TSC_G7_IO4, TIM3_CH3

41.4.30. PE4
General-purpose I/O pin.
Alternate functions: TSC_G7_IO3, TIM3_CH2

41.4.31. PE3
General-purpose I/O pin.
Alternate functions: TSC_G7_IO2, TIM3_CH1

41.4.32. PE2
General-purpose I/O pin.
Alternate functions: TSC_G7_IO1, TIM3_ETR

41.4.33. PE15
General-purpose I/O pin.
Alternate functions: SPI1_MOSI, I2S1_SD, TIM1_BKIN

41.4.34. PE14
General-purpose I/O pin.
Alternate functions: SPI1_MISO, I2S1_MCK, TIM1_CH4

41.4.35. PE13
General-purpose I/O pin.
Alternate functions: SPI1_SCK, I2S1_CK, TIM1_CH3

41.4.36. PE12
General-purpose I/O pin.
Alternate functions: SPI1_NSS, I2S1_WS, TIM1_CH3N

41.4.37. PE11
General-purpose I/O pin.
Alternate functions: TIM1_CH2

41.4.38. PE10
General-purpose I/O pin.
Alternate functions: TIM1_CH2N

41.4.39. PE1
General-purpose I/O pin.
Alternate functions: EVENTOUT, TIM17_CH1

41.4.40. PE0
General-purpose I/O pin.
Alternate functions: EVENTOUT, TIM16_CH1

41.4.41. PD9
General-purpose I/O pin.
Alternate functions: USART3_RX

41.4.42. PD8
General-purpose I/O pin.
Alternate functions: USART3_TX

41.4.43. PD7

General-purpose I/O pin.
Alternate functions: USART2_CK
Il pin è impostato come Digital Input segnale pull up a media velocità

41.4.44. PD6

General-purpose I/O pin.
Alternate functions: USART2_RX
Il pin è impostato come Digital Input segnale pull up a media velocità

41.4.45. PD5

General-purpose I/O pin.
Alternate functions: USART2_TX
Il pin è impostato come Digital Input segnale pull up a media velocità

41.4.46. PD4

General-purpose I/O pin.
Alternate functions: SPI2_MOSI, I2S2_SD, USART2_RTS
Il pin è impostato come Digital Input segnale pull up a media velocità

41.4.47. PD3

General-purpose I/O pin.
Alternate functions: SPI2_MISO, I2S2_MCK, USART2_CTS
Il pin è impostato come Digital Input segnale pull up a media velocità

41.4.48. PD2

General-purpose I/O pin.
Alternate functions: USART3_RTS, TIM3_ETR
Il pin è impostato come Digital Input segnale pull up a media velocità

41.4.49. PD15

General-purpose I/O pin.
Alternate functions: TSC_G8_IO4, CRS_SYNC

41.4.50. PD14

General-purpose I/O pin.
Alternate functions: TSC_G8_IO3

41.4.51. PD13

General-purpose I/O pin.
Alternate functions: TSC_G8_IO2

41.4.52. PD12

General-purpose I/O pin.
Alternate functions: USART3_RTS, TSC_G8_IO1

41.4.53. PD11

General-purpose I/O pin.
Alternate functions: USART3_CTS

41.4.54. PD10

General-purpose I/O pin.
Alternate functions: USART3_CK

41.4.55. PC9

General-purpose I/O pin.

Alternate functions: TIM3_CH4

Per l'applicativo in uso il pin è impostato come segnale di Digital Output e viene commutato a '1' se la USART è abilitata, a '0' se risulta disabilitata.

41.4.56. PC8

General-purpose I/O pin.

Alternate functions: TIM3_CH3

41.4.57. PC7

General-purpose I/O pin.

Alternate functions: TIM3_CH2

41.4.58. PC6

General-purpose I/O pin.

Alternate functions: TIM3_CH1

41.4.59. PC5

General-purpose I/O pin.

Alternate functions: TSC_G3_IO1, USART3_RX, ADC_IN15, WKUP5

41.4.60. PC4

General-purpose I/O pin.

Alternate functions: EVENTOUT, USART3_TX, ADC_IN14

41.4.61. PC3

General-purpose I/O pin.

Alternate functions: SPI2_MOSI, I2S2_SD, EVENTOUT, ADC_IN13

Il pin è impostato come Analog Input per ulteriori info vedere la funzioneinit() dal diagramma delle classi

41.4.62. PC2

General-purpose I/O pin.

Alternate functions: SPI2_MISO, I2S2_MCK, EVENTOUT, ADC_IN12

Il pin è impostato come Analog Input per ulteriori info vedere la funzioneinit() dal diagramma delle classi

41.4.63. PC13

General-purpose I/O pin.

Alternate functions: WKUP2, RTC_TAMP1, RTC_TS, RTC_OUT

41.4.64. PC12

General-purpose I/O pin.

Alternate functions: USART3_CK, USART4_CK

Per l'applicativo in uso il pin è impostato come segnale USART4_CK.

41.4.65. PC11

General-purpose I/O pin.

Alternate functions: USART3_RX, USART4_RX

Per l'applicativo in uso il pin è impostato come segnale USART4_RX.

41.4.66. PC10

General-purpose I/O pin.

Alternate functions: USART3_TX, USART4_TX

Per l'applicativo in uso il pin è impostato come segnale USART4_TX.

41.4.67. PC1

General-purpose I/O pin.
Alternate functions: EVENTOUT, ADC_IN11
Il pin è impostato come Analog Input per ulteriori info vedere la funzioneinit() dal diagramma delle classi

41.4.68. PC0

General-purpose I/O pin.
Alternate functions: EVENTOUT, ADC_IN10
Il pin è impostato come Analog Input per ulteriori info vedere la funzioneinit() dal diagramma delle classi

41.4.69. PB9

General-purpose I/O pin.
Alternate functions: SPI2_NSS, I2S2_WS, I2C1_SDA, IR_OUT, TIM17_CH1, EVENTOUT, CAN_TX
il pin è inpostato come CAN_TX segnale di trasmissione della rete CAN.

41.4.70. PB8

General-purpose I/O pin.
Alternate functions: I2C1_SCL, CEC, TIM16_CH1, TSC_SYNC, CAN_RX

41.4.71. PB7

General-purpose I/O pin.
Alternate functions: I2C1_SDA, USART1_RX, USART4_CTS, TIM17_CH1N, TSC_G5_IO4
Per l'applicativo in uso il pin è impostato come segnale USART4_CTS.

41.4.72. PB6

General-purpose I/O pin.
Alternate functions: I2C1_SCL, USART1_TX, TIM16_CH1N, TSC_G5_I03

41.4.73. PB5

General-purpose I/O pin.
Alternate functions: SPI1_MOSI, I2S1_SD, I2C1_SMBA, TIM16_BKIN, TIM3_CH2, WKUP6

41.4.74. PB4

General-purpose I/O pin.
Alternate functions: SPI1_MISO, I2S1_MCK, TIM17_BKIN, TIM3_CH1, TSC_G5_IO2, EVENTOUT

41.4.75. PB3

General-purpose I/O pin.
Alternate functions: SPI1_SCK, I2S1_CK, TIM2_CH2, TSC_G5_IO1, EVENTOUT

41.4.76. PB2

General-purpose I/O pin.
Alternate functions: TSC_G3_IO4

41.4.77. PB15

General-purpose I/O pin.
Alternate functions: SPI2_MOSI, I2S2_SD, TIM1_CH3N, TIM15_CH1N, TIM15_CH2, WKUP7, RTC_REFIN

41.4.78. PB14

General-purpose I/O pin.

Alternate functions: SPI2_MISO, I2S2_MCK, I2C2_SDA, USART3_RTS, TIM1_CH2N, TIM15_CH1, SC_G6_IO4

41.4.79. PB13

General-purpose I/O pin.
Alternate functions: SPI2_SCK, I2S2_CK, I2C2_SCL, USART3_CTS, TIM1_CH1N, TSC_G6_IO3
il pin è inpostato come CAN_RX ricezione di trasmissione della rete CAN.

41.4.80. PB12

General-purpose I/O pin.
Alternate functions: TIM1_BKIN, TIM15_BKIN, SPI2_NSS, I2S2_WS, USART3_CK, TSC_G6_IO2, EVENTOUT

41.4.81. PB11

General-purpose I/O pin.
Alternate functions: USART3_RX, TIM2_CH4, EVENTOUT, TSC_G6_IO1, I2C2_SDA

41.4.82. PD1

Il pin è impostato come Digital Input segnale pull up a media velocità

41.4.83. PF1-OSC_OUT

General-purpose I/O pin.
Alternate functions: OSC_OUT

41.4.84. PF0-OSC_IN

General-purpose I/O pin.
Alternate functions: CRS_ SYNC, OSC_IN

41.4.85. PC15-OSC32_OUT

General-purpose I/O pin.
Alternate functions: OSC32_OUT

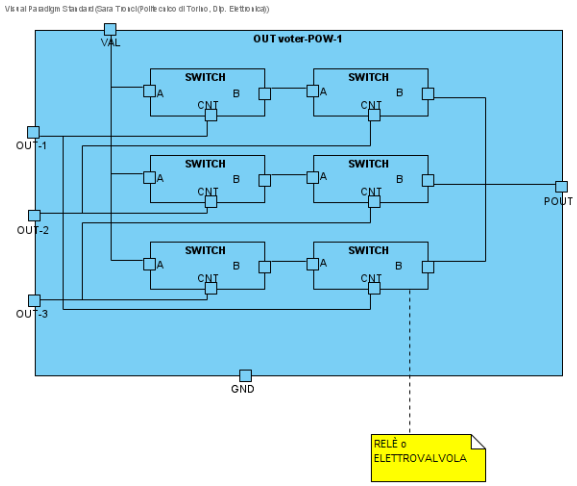
41.4.86. PC14-OSC32_IN

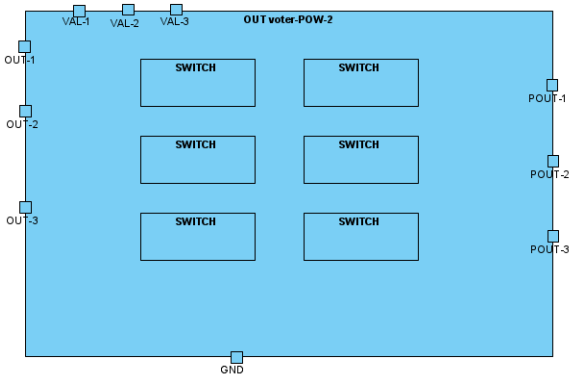
General-purpose I/O pin.
Alternate functions: OSC32_IN

41.4.87. BOOT0

Pin dedicato all'abilitazione della memoria di Boot del STM32F072VB

42. OUT voter-POW Component Diagram





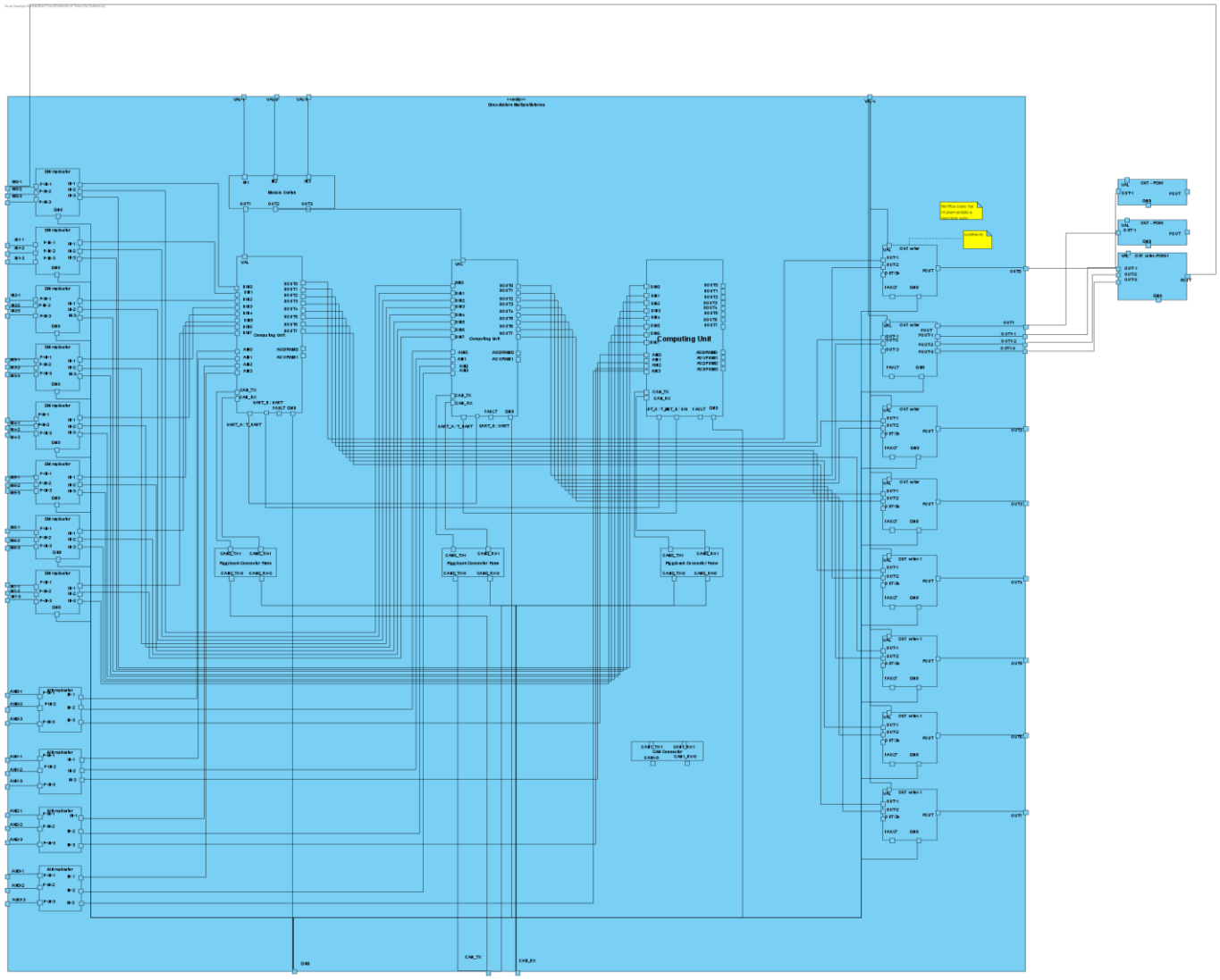
42.1. OUT voter-POW-1

Computes:

- POUT= (OUT1-1 * OUT1-2) + OUT1-2 * OUT1-3) + OUT1-1 * OUT1-3)
- FAULT = / [(OUT1-1* OUT1-2 * OUT1-3) + (/ OUT1-1* / OUT1-2 * / OUT1-3)]

43. Proposed Architecture

Il Dimostratore Multipiattaforma, è costituito da tre core (due Computing Unit, e un Modulo FPGA), che verificano in real-time, tramite tre algoritmi congruenti, che il sistema stia operando correttamente.



43.1. OUT voter

Computes:

- POUT= (OUT1-1 * OUT1-2) + (OUT1-2 * OUT1-3) + (OUT1-1 * OUT1-3)
- POUT-1= OUT-1
- ...
- FAULT = 0 if [(OUT1-1* OUT1-2 * OUT1-3) + (/ OUT1-1* / OUT1-2 * / OUT1-3)]
- FAULT = 1 if [(/ OUT1-1* OUT1-2 * OUT1-3) + (OUT1-1* / OUT1-2 * / OUT1-3)]
- FAULT = 2 if ...

43.2. OUT voter-1

Computes:

- POUT= (OUT1-1 * OUT1-2) + (OUT1-2 * OUT1-3) + (OUT1-1 * OUT1-3)
- FAULT = 0 if [(OUT1-1* OUT1-2 * OUT1-3) + (/ OUT1-1* / OUT1-2 * / OUT1-3)]
- FAULT = 1 if [(/ OUT1-1* OUT1-2 * OUT1-3) + (OUT1-1* / OUT1-2 * / OUT1-3)]
- FAULT = 2 if ...

43.2.1. FAULT

2-bit info che indica quale processore è difforme dagli altri; 0 se tutti concordi

43.3. AIN replicator

Modulo a tre ingrassi per ogni segnale che consente di propagare l'informazione triplicata e proteggere da un corto circuito e da eventuali sovracorrenti un'ingresso digitale del Computing Unit. Il circuito è analogo al DIN replicator.

43.4. Modulo Switch

Modulo che consente di abilitare l'alimentazione ai due Computing Unit e al Modulo FPGA. Ponendo su ON l'interruttore 1, si abilita il primo Computing Unit, ponendo su ON l'interrutore 2 il secondo Computing Unit, e infine a ON l'interrutore 3 il Modulo FPGA. Si può quindi decidere se tenere attivi tutti e tre i core, due o uno.

Il circuito è costituito da 3 RES connesse ciascuna nei prime tre ingressi del DIP Switch x4.

In alternativa al DIP si possono usare tre jumper, DA CONFERMARE

43.4.1. IN1

Pin d'ingresso del Modulo Switch.

43.4.2. OUT1

Pin d'uscita del Modulo Switch.

43.4.3. IN3

Pin d'ingresso del Modulo Switch.

43.4.4. OUT2

Pin d'uscita del Modulo Switch.

43.4.5. OUT3

Pin d'uscita del Modulo Switch.

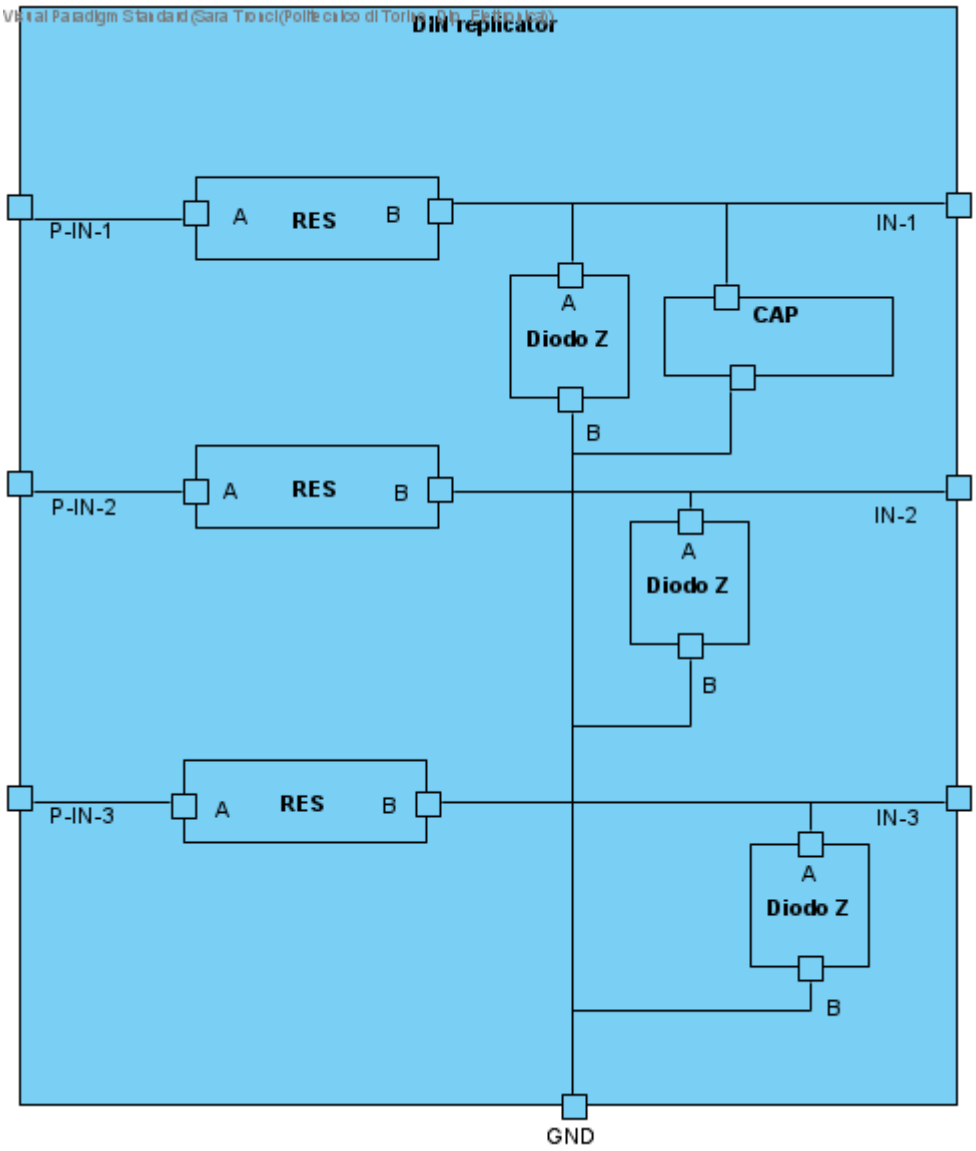
43.5. OUT - POW

Computes:

- POUT= (OUT1-1 * OUT1-2) + OUT1-2 * OUT1-3) + OUT1-1 * OUT1-3)
- FAULT = / [(OUT1-1* OUT1-2 * OUT1-3) + (/ OUT1-1* / OUT1-2 * / OUT1-3)]

44. IN replicator-3 Component Diagram

Descrizione del modulo utilizzato per proteggere gli ingressi digitali e analogici dell Computing Unit.



44.1. RES

Resistenza.

44.1.1. A

Pin d'ingresso della RES.

44.1.2. B

Pin d'uscita della RES.

44.2. Diodo Z

Diodo Zener del DIN replicator

44.2.1. A

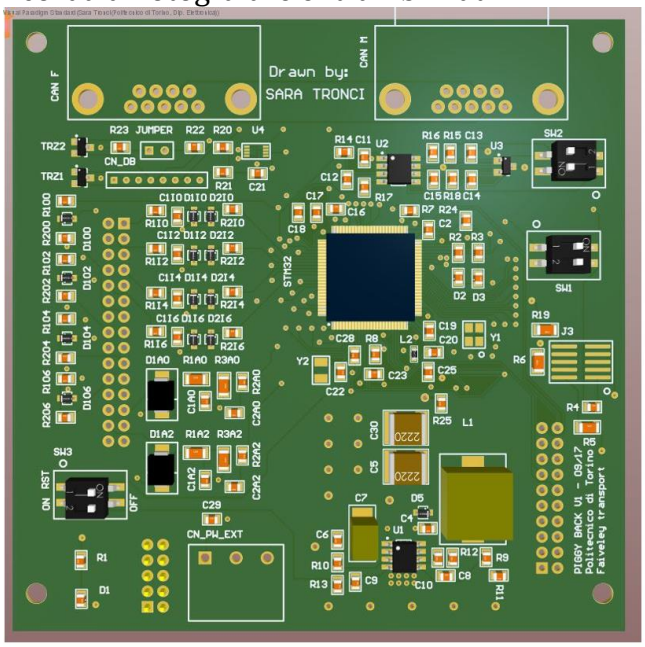
Pin d'ingresso del Diodo Z.

44.2.2. B

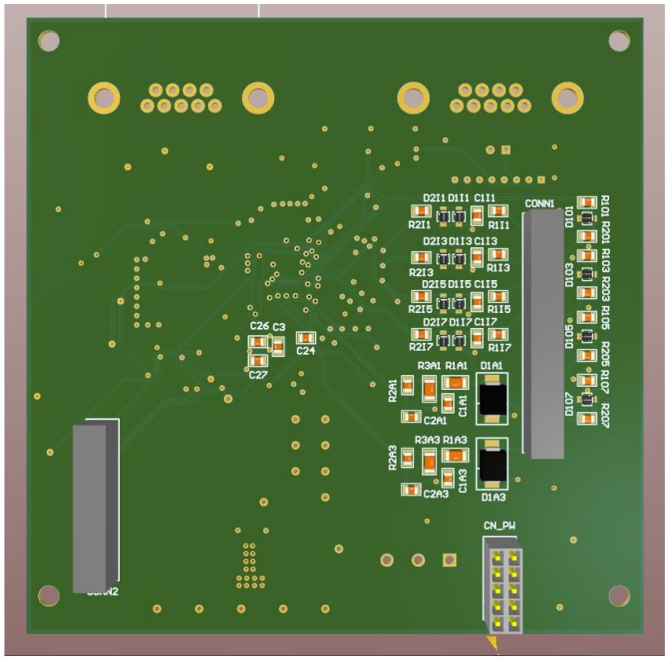
Pin d'uscita del Diodo Z del DIN replicator.

45. FOTO PCB scheda piggy back

ricorda di fotografare entrambi i lati



Modello 3D Computing Unit - Lato Componenti



Modello 3D Computing Unit - Lato Saldature

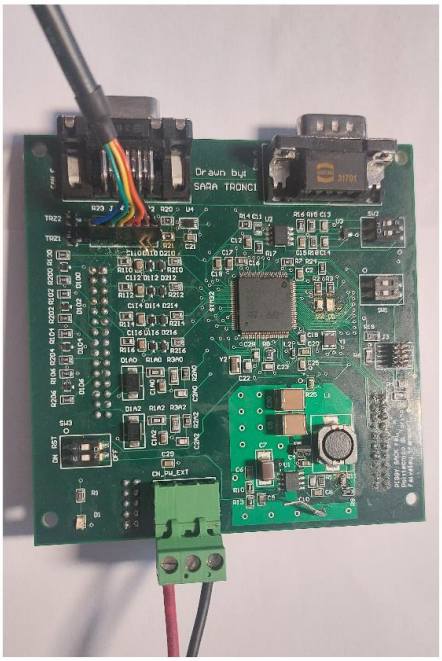


Foto Scheda Computing Unit - Lato Componenti con cavi di alimentazione e programmazione

46. Rigraziamenti

Ringrazio il relatore della tesi, il professor Leonardo Reyneri, che mi ha supportato in questo percorso, anche da un punto di vista umano, oltre che a darmi modo di accrescere le mie competenze.

Un grazie particolare va all'azienda Fayveley Transport, gruppo Wabtec, in cui lavoro attualmente e all'Ingegnere Roberto Tione che mi ha introdotta in questa realtà in cui lavoro da ormai cinque anni.

Ringrazio inoltre Giovanni Sica e Stefano Serri che sono stati i miei relatori di riferimento all'interno dell'azienda.

Un caloroso ringraziamento va anche a Patrizia e Stefania amiche ormai di lunga data con cui ho condiviso momenti di studio e di svago.

Appendici - Schemi elettrici e SW

47. TripleData Library

47.1. TripleData

This class is intended to offer support to simple redundancy management in C++.

It internally stores three replicas of a variable of a generic user-defined type DataType. For instance:

1. TripleData<byte> stores three replicas of an unsigned char, while
2. TripleData<float> stores three replicas of a float, while
3. TripleData<myType> stores three replicas of a user-defined type myType.

This class also overwrites all the most common algebraic and logic operators such as to be able to write a redundant algorithm exactly as if it were a normal C program.

For instance,

```
short a,b,c;
c = a + b;
```

stores in c the sum of a and b, while

```
TripleData<short> a,b,c;
c = a + b;
```

stores in the three internal replicas of c the sum of the three replicas of a and the corresponding three replicas of b.

This class also offers conversion operators to/from non-redundant data.

47.1.1. d1

Signature: -d1 : DataType

47.1.2. d2

Signature: -d2 : DataType

47.1.3. d3

Signature: -d3 : DataType

47.1.4. operator DataType

Signature: operator DataType()

Casts a redundant TripleData<DataType> variable a to a single DataType by voting.

```
TripleData<DataType> a;
DataType b;
b = (DataType) a;
```

If at most one replica of a is corrupted, voting will recover a correct value. If all three replicas are different from each other, the first (potentially corrupted) replica is returned.

The class' internal storage is NOT affected so, if the variable is corrupted, the casting operator will return a correct value, while the class is left corrupted.

To recover internal class' storage to a correct situation (namely, with all three replicas correct), one shall explicitly use the sync(): DataType operator.

Code Body: if (d1==d2) return d1;
else if (d1==d3) return d1;
else return d2;

47.1.5. operator=

Signature: operator=(val : TripleData) : TripleData

Overwrites the assignation operator a = b where both a and b are redundant TripleData<DataType>.

It copies a value into the class' internal storage, by triplicating the right-hand value into the class' internal replicas.

When using this operator, the right-hand variable is not verified to be correct; each of the three replicas of the right-hand value is copied as it is into the corresponding replica of the left-hand variable, for a faster execution. As a consequence, assigning a corrupted right-hand value to the left-hand variable, leaves the value corrupted.

To assign a correct left-hand value from a corrupted right-hand location, one shall use the following construct:

```
TripleData<DataType> a, b;
a = (DataType) b;
```

where type casting TripleData<DataType> b to (DataType) recovers a correct value from the three replicas (by voting). The correct value is then copied identically to the three replicas of a.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 = val.d1;
d2 = val.d2;
d3 = val.d3;
return *this; // Return a reference to myself

47.1.6. operator=

Signature: operator=(val : DataType) : TripleData

Overwrites the assignation operator a = b where a is a redundant TripleData<DataType> while b is a non-redundant DataType.

It copies a variable or a constant into the class' internal storage, by triplicating the right-hand value into the class' internal replicas.

Code Body: d1 = val;
d2 = val;
d3 = val;
return *this; // Return a reference to myself

47.1.7. operator+

Signature: operator+(val : DataType) : TripleData

Overwrites the algebraic operator **a + b** when a is redundant TripleData<DataType> while b is DataType.

It sums up each replica of the left-hand variable a to the right-hand variable (or constant) b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value is

summed up as it is to the right-hand variable, for a faster execution. As a consequence, summing up a corrupted left-hand value to a right-hand variable, leaves the value corrupted.

To sum up and correct a corrupted left-hand value to a right-hand variable (or constant), one shall use the following construct:

```
TripleData<DataType> a, b, c;
DataType b;
c = (DataType) a + b; // or, alternatively
c = (DataType) a + 2;
```

where type casting TripleData<DataType> a to (DataType) recovers a correct value from its three replicas (by voting). The correct value is then copied identically to the three replicas of c.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 + val;
result.d2 = d2 + val;
result.d3 = d3 + val;
return result;

47.1.8. operator+

Signature: operator+(val : TripleData) : TripleData

Overwrites the algebraic operator **a + b** when both a and b are redundant TripleData<DataType>.

It sums up each replica of the left-hand variable a to the corresponding replica of the right-hand variable b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is summed up as it is to the corresponding replica of the right-hand variable, for a faster execution. As a consequence, summing up a corrupted left-hand value to a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with sums is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but summing up the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To sum up and correct a corrupted left-hand value to a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b, c;
c = (DataType) a + (DataType) b;
```

where type casting TripleData<DataType> a and b to (DataType) recovers a correct value from the three replicas of both variables (by voting). The correct value is then copied identically to the three replicas of c.

Alternatively, for a faster execution, the following construct:

```
            TripleData<DataType> a, b, c;
            c = (DataType) a + b;
```

will only recover a but not b; the result might be potentially corrupted but recoverable, as at most one replica of the result will be corrupted.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;

```
result.d1 = d1 + val.d1;
result.d2 = d2 + val.d2;
result.d3 = d3 + val.d3;
return result;

// an alternative, apparently more efficient and more rubust version:
//result = (DataType) *this + (DataType) val;
//return result;
```

47.1.9. operator++

Signature: operator++() : TripleData

Overwrites the pre-increment operator ++a, when a is redundant TripleData<DataType>.

It individually increments each replica of the left-hand variable a and returns the value before incrementing as a redundant TripleData<DataType>. The left-hand operator is therefore modified.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value are incremented separately, for a faster execution. As a consequence, incrementing a corrupted left-hand value, leaves the value corrupted.

To have a correct value after incrementing, one shall explicitly use the sync(): DataType operator:

```
            TripleData<DataType> a;
            ++a;
            a.sync();
```

where sync(): DataType restores a correct value into a.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> temp;

```
temp = (*this);
d1++;
d2++;
d3++;
return temp;
```

47.1.10. operator++

Signature: operator++(dummy : int) : TripleData

Overwrites the post-increment operator **a++**, when a is redundant TripleData<DataType>.

It individually increments each replica of the left-hand variable a and returns the incremented value as a redundant TripleData<DataType>. The left-hand operator is therefore modified.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value are

incremented separately, for a faster execution. As a consequence, incrementing a corrupted left-hand value, leaves the value corrupted.

To have a correct value after incrementing, one shall explicitly use the sync(): DataType operator:

```
TripleData<DataType> a;
a++;
a.sync();
```

where sync(): DataType restores a correct value into a.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1++;
d2++;
d3++;
return *this; // Return a reference to myself

47.1.11. operator+=

Signature: operator+=(val : DataType) : TripleData

Overwrites the algebraic operator **a += b** when a is redundant TripleData<DataType> while b is DataType.

It sums up each replica of the left-hand variable a to the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, the left-hand variables is not verified to be correct; each of the three replicas of the left-hand value is summed up as it is to the right-hand variable, for a faster execution. As a consequence, summing up a corrupted left-hand value to a right-hand variable, leaves the value corrupted.

To sum up and correct a corrupted left-hand value to a right-hand variable, one shall use either of the following constructs:

```
TripleData<DataType> a;
DataType b;
a += b;
a.sync();
```

or

```
TripleData<DataType> a;
DataType b;
a = (DataType) a + b;
```

which recovers a before using its value.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 += val;
d2 += val;
d3 += val;
return *this; // Return a reference to myself

47.1.12. operator+=

Signature: operator+=(val : TripleData) : TripleData

Overwrites the algebraic operator **a += b** when both a and b are redundant TripleData<DataType>.

It sums up each replica of the left-hand variable a to the corresponding replica of the right-hand variable b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is summed up as it is to the corresponding replica of the right-hand variable, for a faster execution. As a consequence, summing up a corrupted left-hand value to a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with sums is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but summing up the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To sum up and correct a corrupted left-hand value to a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b;
a += (DataType) b;
```

where type casting TripleData<DataType> b to (DataType) recovers a correct value from the three replicas of b (by voting). The correct value is then summed identically to the three replicas of a.

If b is corrupted, its correct value is recovered, while if a is corrupted, it will remain corrupted. To have a fully correct value stored into a, one shall alternatively use either of the following constructs:

```
TripleData<DataType> a, b;
a += (DataType) b;
a.sync();
```

or

```
TripleData<DataType> a, b;
a = (DataType) a + (DataType) b;
```

which recover both a and b.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 += val.d1;
d2 += val.d2;
d3 += val.d3;
return *this; // Return a reference to myself

47.1.13. operator-

Signature: operator-(val : DataType) : TripleData

Overwrites the algebraic operator **a - b** when a is redundant TripleData<DataType> while b is DataType.

It subtracts each replica of the left-hand variable a from the right-hand variable (or constant) b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value is

subtracted as it is from the right-hand variable, for a faster execution. As a consequence, subtracting a corrupted left-hand value from a right-hand variable, leaves the value corrupted.

To subtract and correct a corrupted left-hand value from a right-hand variable (or constant), one shall use the following construct:

```
TripleData<DataType> a, b, c;
DataType b;
c = (DataType) a - b; // or, alternatively
c = (DataType) a - 2;
```

where type casting TripleData<DataType> a to (DataType) recovers a correct value from its three replicas (by voting). The correct value is then copied identically to the three replicas of c.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 - val;
result.d2 = d2 - val;
result.d3 = d3 - val;
return result;

47.1.14. operator-

Signature: operator-(val : TripleData) : TripleData

Overwrites the algebraic operator **a - b** when both a and b are redundant TripleData<DataType>.

It subtracts each replica of the left-hand variable a from the corresponding replica of the right-hand variable (or constant) b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is subtracted as it is from the corresponding replica of the right-hand variable, for a faster execution. As a consequence, subtracting a corrupted left-hand value from a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with subtraction is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but subtracting the first, second and third replicas of a from b, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To subtract and correct a corrupted left-hand value from a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b, c;
c = (DataType) a - (DataType) b;
```

where type casting TripleData<DataType> a and b to (DataType) recovers a correct value from the three replicas of both variables (by voting). The correct value is then copied identically to the three replicas of c.

Alternatively, for a faster execution, the following construct:

```
            TripleData<DataType> a, b, c;
            c = (DataType) a - b;
```

will only recover a but not b; the result might be potentially corrupted but recoverable, as at most one replica of the result will be corrupted.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 - val.d1;
result.d2 = d2 - val.d2;
result.d3 = d3 - val.d3;
return result;

47.1.15. operator--

Signature: operator--() : TripleData

Overwrites the pre-decrement operator --a, when a is redundant TripleData<DataType>.

It individually decrements each replica of the left-hand variable a and returns the value before decrementing as a redundant TripleData<DataType>. The left-hand operator is therefore modified.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value are decremented separately, for a faster execution. As a consequence, decrementing a corrupted left-hand value, leaves the value corrupted.

To have a correct value after decrementing, one shall explicitly use the sync(): DataType operator:

```
            TripleData<DataType> a;
            --a;
            a.sync();
```

where sync(): DataType restores a correct value into a.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1--;
d2--;
d3--;
return *this; // Return a reference to myself

47.1.16. operator--

Signature: operator--(dummy : int) : TripleData

Overwrites the post-decrement operator **a--**, when a is redundant TripleData<DataType>.

It individually decrements each replica of the left-hand variable a and returns the decremented value as a redundant TripleData<DataType>. The left-hand operator is therefore modified.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value are decremented separately, for a faster execution. As a consequence, decrementing a corrupted left-hand value, leaves the value corrupted.

To have a correct value after decrementing, one shall explicitly use the sync(): DataType operator:

```
            TripleData<DataType> a;
```

```
            a--;
            a.sync();
```

where sync(): DataType restores a correct value into a.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1--;
d2--;
d3--;
return *this; // Return a reference to myself

47.1.17. operator-=

Signature: operator-=(val : DataType) : TripleData

Overwrites the algebraic operator **a -= b** when a is redundant TripleData<DataType> while b is DataType.

It subtracts each replica of the left-hand variable a from the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, the left-hand variables is not verified to be correct; each of the three replicas of the left-hand value is subtracted as it is from the right-hand variable, for a faster execution. As a consequence, subtracting a corrupted left-hand value from a right-hand variable, leaves the value corrupted.

To subtract and correct a corrupted left-hand value from a right-hand variable, one shall use either of the following constructs:

```
            TripleData<DataType> a;
            DataType b;
            a -= b;
            a.sync();
```

or

```
            TripleData<DataType> a;
            DataType b;
            a = (DataType) a - b;
```

which recover a before using its value.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 -= val;
d2 -= val;
d3 -= val;
return *this; // Return a reference to myself

47.1.18. operator-=

Signature: operator-=(val : TripleData) : TripleData

Overwrites the algebraic operator **a -= b** when both a and b are redundant TripleData<DataType>.

It subtracts each replica of the left-hand variable a from the corresponding replica of the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of

the left-hand value is subtracted as it is from the corresponding replica of the right-hand variable, for a faster execution. As a consequence, subtracting a corrupted left-hand value from a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with subtractions is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but subtracting the first, second and third replicas of b from a, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To subtract and correct a corrupted left-hand value from a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b;
a -= (DataType) b;
```

where type casting TripleData<DataType> b to (DataType) recovers a correct value from the three replicas of b (by voting). The correct value is then subtracted identically from the three replicas of a.

If b is corrupted, its correct value is recovered, while if a is corrupted, it will remain corrupted. To have a fully correct value stored into a, one shall alternatively use either of the following constructs:

```
TripleData<DataType> a, b;
a -= (DataType) b;
a.sync();
```

or

```
TripleData<DataType> a, b;
a = (DataType) a - (DataType) b;
```

which recover both a and b.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 -= val.d1;
d2 -= val.d2;
d3 -= val.d3;
return *this; // Return a reference to myself

47.1.19. operator*

Signature: operator*(val : DataType) : TripleData

Overwrites the algebraic operator **a * b** when a is redundant TripleData<DataType> while b is DataType.

It multiplies each replica of the left-hand variable a by the right-hand variable (or constant) b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value is multiplied as it is by the right-hand variable, for a faster execution. As a consequence, multiplying a corrupted left-hand value by a right-hand variable, leaves the value corrupted.

To multiply and correct a corrupted left-hand value by a right-hand variable (or constant), one shall use the following construct:

```
TripleData<DataType> a, b, c;
```

```
DataType b;
c = (DataType) a * b; // or, alternatively
c = (DataType) a * 2;
```

where type casting TripleData<DataType> a to (DataType) recovers a correct value from its three replicas (by voting). The correct value is then copied identically to the three replicas of c.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 * val;
result.d2 = d2 * val;
result.d3 = d3 * val;
return result;

47.1.20. operator*

Signature: operator*(val : TripleData) : TripleData

Overwrites the algebraic operator **a * b** when both a and b are redundant TripleData<DataType>.

It multiplies each replica of the left-hand variable a by the corresponding replica of the right-hand variable b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is multiplied as it is by the corresponding replica of the right-hand variable, for a faster execution. As a consequence, multiplying a corrupted left-hand value by a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with multiplications is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but multiplying the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To multiply and correct a corrupted left-hand value by a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b, c;
c = (DataType) a * (DataType) b;
```

where type casting TripleData<DataType> a and b to (DataType) recovers a correct value from the three replicas of both variables (by voting). The correct value is then copied identically to the three replicas of c.

Alternatively, for a faster execution, the following construct:

```
TripleData<DataType> a, b, c;
c = (DataType) a * b;
```

will only recover a but not b; the result might be potentially corrupted but recoverable, as at most one replica of the result will be corrupted.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 * val.d1;
result.d2 = d2 * val.d2;
result.d3 = d3 * val.d3;
return result;

47.1.21. operator*=

Signature: operator*=(val : DataType) : TripleData

Overwrites the algebraic operator **a *= b** when a is redundant TripleData<DataType> while b is DataType.

It multiplies each replica of the left-hand variable a by the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, the left-hand variables is not verified to be correct; each of the three replicas of the left-hand value is multiplied as it is by the right-hand variable, for a faster execution. As a consequence, multiplying up a corrupted left-hand value by a right-hand variable, leaves the value corrupted.

To multiply and correct a corrupted left-hand value by a right-hand variable, one shall use either of the following constructs:

```
TripleData<DataType> a;
DataType b;
a *= b;
a.sync();
```

or

```
TripleData<DataType> a;
DataType b;
a = (DataType) a * b;
```

which recovers a before using its value.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 *= val;
d2 *= val;
d3 *= val;
return *this; // Return a reference to myself

47.1.22. operator*=

Signature: operator*=(val : TripleData) : TripleData

Overwrites the algebraic operator **a *= b** when both a and b are redundant TripleData<DataType>.

It multiplies each replica of the left-hand variable a by the corresponding replica of the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is multiplied as it is from the corresponding replica of the right-hand variable, for a faster execution. As a

consequence, multiplying a corrupted left-hand value by a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with multiplications is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but multiplying the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To multiply and correct a corrupted left-hand value by a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b;
a *= (DataType) b;
```

where type casting TripleData<DataType> b to (DataType) recovers a correct value from the three replicas of b (by voting). The correct value is then multiplied identically by the three replicas of a.

If b is corrupted, its correct value is recovered, while if a is corrupted, it will remain corrupted. To have a fully correct value stored into a, one shall alternatively use either of the following constructs:

```
TripleData<DataType> a, b;
a *= (DataType) b;
a.sync();
```

or

```
TripleData<DataType> a, b;
a = (DataType) a * (DataType) b;
```

which recover both a and b.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 *= val.d1;
d2 *= val.d2;
d3 *= val.d3;
return *this; // Return a reference to myself

47.1.23. operator/

Signature: operator/(val : DataType) : TripleData

Overwrites the algebraic operator **a / b** when a is redundant TripleData<DataType> while b is DataType.

It divides each replica of the left-hand variable a by the right-hand variable (or constant) b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value is divided as it is by the right-hand variable, for a faster execution. As a consequence, dividing a corrupted left-hand value by a right-hand variable, leaves the value corrupted.

To divide and correct a corrupted left-hand value by a right-hand variable (or constant), one shall use the following construct:

```
TripleData<DataType> a, b, c;
DataType b;
c = (DataType) a / b; // or, alternatively
```

```
        c = (DataType) a / 2;
```

where type casting TripleData<DataType> a to (DataType) recovers a correct value from its three replicas (by voting). The correct value is then copied identically to the three replicas of c.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 / val;
result.d2 = d2 / val;
result.d3 = d3 / val;
return result;

47.1.24. operator/

Signature: operator/(val : TripleData) : TripleData

Overwrites the algebraic operator **a / b** when both a and b are redundant TripleData<DataType>.

It divides each replica of the left-hand variable a by the corresponding replica of the right-hand variable b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is divided as it is by the corresponding replica of the right-hand variable, for a faster execution. As a consequence, dividing a corrupted left-hand value by a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with divisions is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but dividing the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To divide and correct a corrupted left-hand value by a corrupted right-hand variable, one shall use the following construct:

```
        TripleData<DataType> a, b, c;
        c = (DataType) a / (DataType) b;
```

where type casting TripleData<DataType> a and b to (DataType) recovers a correct value from the three replicas of both variables (by voting). The correct value is then copied identically to the three replicas of c.

Alternatively, for a faster execution, the following construct:

```
        TripleData<DataType> a, b, c;
        c = (DataType) a / b;
```

will only recover a but not b; the result might be potentially corrupted but recoverable, as at most one replica of the result will be corrupted.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 / val.d1;
result.d2 = d2 / val.d2;
result.d3 = d3 / val.d3;
return result;

47.1.25. operator/=

Signature: operator/=(val : DataType) : TripleData

Overwrites the algebraic operator **a /= b** when a is redundant TripleData<DataType> while b is DataType.

It divides each replica of the left-hand variable a by the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, the left-hand variables is not verified to be correct; each of the three replicas of the left-hand value is divided as it is by the right-hand variable, for a faster execution. As a consequence, dividing a corrupted left-hand value by a right-hand variable, leaves the value corrupted.

To divide and correct a corrupted left-hand value by a right-hand variable, one shall use either of the following constructs:

```
TripleData<DataType> a;
DataType b;
a /= b;
a.sync();
```

or

```
TripleData<DataType> a;
DataType b;
a = (DataType) a / b;
```

which recovers a before using its value.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 /= val;
d2 /= val;
d3 /= val;
return *this; // Return a reference to myself

47.1.26. operator/=

Signature: operator/=(val : TripleData) : TripleData

Overwrites the algebraic operator **a /= b** when both a and b are redundant TripleData<DataType>.

It divides each replica of the left-hand variable a by the corresponding replica of the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is divided as it is from the corresponding replica of the right-hand variable, for a faster execution. As a consequence, dividing a corrupted left-hand value by a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with divisions is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but dividing the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To divide and correct a corrupted left-hand value by a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b;
a /= (DataType) b;
```

where type casting TripleData<DataType> b to (DataType) recovers a correct value from the three replicas of b (by voting). The correct value is then divided identically by the three replicas of a.

If b is corrupted, its correct value is recovered, while if a is corrupted, it will remain corrupted. To have a fully correct value stored into a, one shall alternatively use either of the following constructs:

```
TripleData<DataType> a, b;
a /= (DataType) b;
a.sync();
```

or

```
TripleData<DataType> a, b;
a = (DataType) a / (DataType) b;
```

which recover both a and b.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 /= val.d1;
d2 /= val.d2;
d3 /= val.d3;
return *this; // Return a reference to myself

47.1.27. operator|

Signature: operator|(val : DataType) : TripleData

Overwrites the algebraic operator **a | b** when a is redundant TripleData<DataType> while b is DataType.

It OR-s each replica of the left-hand variable a with the right-hand variable (or constant) b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value is OR-ed as it is with the right-hand variable, for a faster execution. As a consequence, OR-ing a corrupted left-hand value with a right-hand variable, leaves the value corrupted.

To OR and correct a corrupted left-hand value with a right-hand variable (or constant), one shall use the following construct:

```
TripleData<DataType> a, b, c;
DataType b;
c = (DataType) a | b; // or, alternatively
c = (DataType) a | 0x20;
```

where type casting TripleData<DataType> a to (DataType) recovers a correct value from its three replicas (by voting). The correct value is then copied identically to the three replicas of c.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 | val;
result.d2 = d2 | val;
result.d3 = d3 | val;
return result;

47.1.28. operator|

Signature: operator|(val : TripleData) : TripleData

Overwrites the logic operator **a | b** when both a and b are redundant TripleData<DataType>.

It OR-s each replica of the left-hand variable a with the corresponding replica of the right-hand variable b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is OR-ed as it is with the corresponding replica of the right-hand variable, for a faster execution. As a consequence, OR-ing a corrupted left-hand value with a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with OR-s is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but OR-ing the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To AND and correct a corrupted left-hand value to a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b, c;
c = (DataType) a | (DataType) b;
```

where type casting TripleData<DataType> a and b to (DataType) recovers a correct value from the three replicas of both variables (by voting). The correct value is then copied identically to the three replicas of c.

Alternatively, for a faster execution, the following construct:

```
TripleData<DataType> a, b, c;
c = (DataType) a | b;
```

will only recover a but not b; the result might be potentially corrupted but recoverable, as at most one replica of the result will be corrupted.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 | val.d1;

result.d2 = d2 | val.d2;
result.d3 = d3 | val.d3;
return result;

47.1.29. operator|=

Signature: operator|=(val : DataType) : TripleData

Overwrites the logic operator **a |= b** when a is redundant TripleData<DataType> while b is DataType.

It OR-s each replica of the left-hand variable a with the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, the left-hand variables is not verified to be correct; each of the three replicas of the left-hand value is OR-ed as it is with the right-hand variable, for a faster execution. As a consequence, OR-ing a corrupted left-hand value with a right-hand variable, leaves the value corrupted.

To OR and correct a corrupted left-hand value to a right-hand variable, one shall use either of the following constructs:

```
TripleData<DataType> a;
DataType b;
a |= b;
a.sync();
```

or

```
TripleData<DataType> a;
DataType b;
a = (DataType) a | b;
```

which recovers a before using its value.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 |= val;
d2 |= val;
d3 |= val;
return *this; // Return a reference to myself

47.1.30. operator|=

Signature: operator|=(val : TripleData) : TripleData

Overwrites the logic operator **a |= b** when both a and b are redundant TripleData<DataType>.

It OR-s each replica of the left-hand variable a with the corresponding replica of the right-hand variable b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is OR-ed as it is with the corresponding replica of the right-hand variable, for a faster execution. As a consequence, OR-ing a corrupted left-hand value with a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with OR-s is more risky than with assignation operator = for the following reason: imagine that a has

the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but OR-ing the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To OR and correct a corrupted left-hand value with a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b;
a |= (DataType) b;
```

where type casting TripleData<DataType> b to (DataType) recovers a correct value from the three replicas of b (by voting). The correct value is then copied identically to the three replicas of c.

If b is corrupted, its correct value is recovered, while if a is corrupted, it will remain corrupted. To have a fully correct value stored into a, one shall alternatively use either of the following constructs:

```
TripleData<DataType> a, b;
a |= (DataType) b;
a.sync();
```

or

```
TripleData<DataType> a, b;
a = (DataType) a | (DataType) b;
```

which recover both a and b. Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 |= val.d1;
d2 |= val.d2;
d3 |= val.d3;
return *this; // Return a reference to myself

47.1.31. operator&

Signature: operator&(val : TripleData) : TripleData

Overwrites the logic operator **a & b** when both a and b are redundant TripleData<DataType>.

It AND-s each replica of the left-hand variable a with the corresponding replica of the right-hand variable b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is AND-ed as it is with the corresponding replica of the right-hand variable, for a faster execution. As a consequence, AND-ing a corrupted left-hand value with a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with AND-s is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but AND-ing the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second

replica (because of b's corruption), so the result will NOT be recoverable!

To AND and correct a corrupted left-hand value to a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b, c;
c = (DataType) a & (DataType) b;
```

where type casting TripleData<DataType> a and b to (DataType) recovers a correct value from the three replicas of both variables (by voting). The correct value is then copied identically to the three replicas of c.

Alternatively, for a faster execution, the following construct:

```
TripleData<DataType> a, b, c;
c = (DataType) a & b;
```

will only recover a but not b; the result might be potentially corrupted but recoverable, as at most one replica of the result will be corrupted.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 & val.d1;
result.d2 = d2 & val.d2;
result.d3 = d3 & val.d3;
return result;

47.1.32. operator&

Signature: operator&(val : DataType) : TripleData

Overwrites the algebraic operator **a & b** when a is redundant TripleData<DataType> while b is DataType.

It AND-s each replica of the left-hand variable a with the right-hand variable (or constant) b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value is AND-ed as it is with the right-hand variable, for a faster execution. As a consequence, AND-ing a corrupted left-hand value with a right-hand variable, leaves the value corrupted.

To AND and correct a corrupted left-hand value with a right-hand variable (or constant), one shall use the following construct:

```
TripleData<DataType> a, b, c;
DataType b;
c = (DataType) a & b; // or, alternatively
c = (DataType) a & 0x20;
```

where type casting TripleData<DataType> a to (DataType) recovers a correct value from its three replicas (by voting). The correct value is then copied identically to the three replicas of c.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 & val;
result.d2 = d2 & val;
result.d3 = d3 & val;

return result;

47.1.33. operator&=

Signature: operator&=(val : DataType) : TripleData

Overwrites the logic operator **a &= b** when a is redundant TripleData<DataType> while b is DataType.

It AND-s each replica of the left-hand variable a with the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, the left-hand variables is not verified to be correct; each of the three replicas of the left-hand value is AND-ed as it is with the right-hand variable, for a faster execution. As a consequence, AND-ing a corrupted left-hand value with a right-hand variable, leaves the value corrupted.

To AND and correct a corrupted left-hand value to a right-hand variable, one shall use either of the following constructs:

```
TripleData<DataType> a;
DataType b;
a &= b;
a.sync();
```

or

```
TripleData<DataType> a;
DataType b;
a = (DataType) a & b;
```

which recovers a before using its value.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 &= val;
d2 &= val;
d3 &= val;
return *this; // Return a reference to myself

47.1.34. operator&=

Signature: operator&=(val : TripleData) : TripleData

Overwrites the logic operator **a &= b** when both a and b are redundant TripleData<DataType>.

It AND-s each replica of the left-hand variable a with the corresponding replica of the right-hand variable b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is AND-ed as it is with the corresponding replica of the right-hand variable, for a faster execution. As a consequence, AND-ing a corrupted left-hand value with a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with AND-s is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b

can be individually recovered by voting, but AND-ing the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To AND and correct a corrupted left-hand value with a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b;
a &= (DataType) b;
```

where type casting TripleData<DataType> b to (DataType) recovers a correct value from the three replicas of b (by voting). The correct value is then copied identically to the three replicas of c.

If b is corrupted, its correct value is recovered, while if a is corrupted, it will remain corrupted. To have a fully correct value stored into a, one shall alternatively use either of the following constructs:

```
TripleData<DataType> a, b;
a &= (DataType) b;
a.sync();
```

or

```
TripleData<DataType> a, b;
a = (DataType) a & (DataType) b;
```

which recover both a and b. Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 &= val.d1;
d2 &= val.d2;
d3 &= val.d3;
return *this; // Return a reference to myself

47.1.35. operator^

Signature: operator^(val : DataType) : TripleData

Overwrites the algebraic operator **a ^ b** when a is redundant TripleData<DataType> while b is DataType.

It EXOR-s each replica of the left-hand variable a with the right-hand variable (or constant) b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value is EXOR-ed as it is with the right-hand variable, for a faster execution. As a consequence, EXOR-ing a corrupted left-hand value with a right-hand variable, leaves the value corrupted.

To EXOR and correct a corrupted left-hand value with a right-hand variable (or constant), one shall use the following construct:

```
TripleData<DataType> a, b, c;
DataType b;
c = (DataType) a ^ b; // or, alternatively
c = (DataType) a ^ 0x20;
```

where type casting TripleData<DataType> a to (DataType) recovers a correct value from its three replicas (by voting). The correct value is then copied identically to the three replicas of c.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 ^ val;
result.d2 = d2 ^ val;
result.d3 = d3 ^ val;
return result;

47.1.36. operator^

Signature: operator^(val : TripleData) : TripleData

Overwrites the logic operator **a ^ b** when both a and b are redundant TripleData<DataType>.

It EXOR-s each replica of the left-hand variable a with the corresponding replica of the right-hand variable b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is EXOR-ed as it is with the corresponding replica of the right-hand variable, for a faster execution. As a consequence, EXOR-ing a corrupted left-hand value with a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with EXOR-s is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but EXOR-ing the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted second replica (because of b's corruption), so the result will NOT be recoverable!

To EXOR and correct a corrupted left-hand value to a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b, c;
c = (DataType) a ^ (DataType) b;
```

where type casting TripleData<DataType> a and b to (DataType) recovers a correct value from the three replicas of both variables (by voting). The correct value is then copied identically to the three replicas of c.

Alternatively, for a faster execution, the following construct:

```
TripleData<DataType> a, b, c;
c = (DataType) a ^ b;
```

will only recover a but not b; the result might be potentially corrupted but recoverable, as at most one replica of the result will be corrupted.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 ^ val.d1;
result.d2 = d2 ^ val.d2;
result.d3 = d3 ^ val.d3;
return result;

47.1.37. operator^=

Signature: operator^=(val : DataType) : TripleData

Overwrites the logic operator **a ^= b** when a is redundant TripleData<DataType> while b is DataType.

It EXOR-s each replica of the left-hand variable a with the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, the left-hand variables is not verified to be correct; each of the three replicas of the left-hand value is EXOR-ed as it is with the right-hand variable, for a faster execution. As a consequence, EXOR-ing a corrupted left-hand value with a right-hand variable, leaves the value corrupted.

To EXOR and correct a corrupted left-hand value to a right-hand variable, one shall use either of the following constructs:

```
TripleData<DataType> a;
DataType b;
a ^= b;
a.sync();
```

or

```
TripleData<DataType> a;
DataType b;
a = (DataType) a ^ b;
```

which recovers a before using its value.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 ^= val;
d2 ^= val;
d3 ^= val;
return *this; // Return a reference to myself

47.1.38. operator^=

Signature: operator^=(val : TripleData) : TripleData

Overwrites the logic operator **a ^= b** when both a and b are redundant TripleData<DataType>.

It EXOR-s each replica of the left-hand variable a with the corresponding replica of the right-hand variable b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, neither the left-hand nor the right-hand variables are verified to be correct; each of the three replicas of the left-hand value is EXOR-ed as it is with the corresponding replica of the right-hand variable, for a faster execution. As a consequence, EXOR-ing a corrupted left-hand value with a corrupted right-hand variable, leaves the value corrupted.

Note that corruption propagation with EXOR-s is more risky than with assignation operator = for the following reason: imagine that a has the first replica corrupted, while the second and third replica are correct, while b only has its second replica corrupted. Both a and b can be individually recovered by voting, but EXOR-ing the first, second and third replicas of a and b together, will return a corrupted first replica (because of a's corruption) together with a corrupted

second replica (because of b's corruption), so the result will NOT be recoverable!

To EXOR and correct a corrupted left-hand value with a corrupted right-hand variable, one shall use the following construct:

```
TripleData<DataType> a, b;
a ^= (DataType) b;
```

where type casting TripleData<DataType> b to (DataType) recovers a correct value from the three replicas of b (by voting). The correct value is then copied identically to the three replicas of c.

If b is corrupted, its correct value is recovered, while if a is corrupted, it will remain corrupted. To have a fully correct value stored into a, one shall alternatively use either of the following constructs:

```
TripleData<DataType> a, b;
a ^= (DataType) b;
a.sync();
```

or

```
TripleData<DataType> a, b;
a = (DataType) a ^ (DataType) b;
```

which recover both a and b. Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 ^= val.d1;
d2 ^= val.d2;
d3 ^= val.d3;
return *this; // Return a reference to myself

47.1.39. operator~

Signature: operator~() : TripleData

Overwrites the logic prefix operator ~**a** when a is redundant TripleData<DataType>.

It complements each replica of the right-hand variable and returns the result as a redundant TripleData<DataType>. Variable a is not affected.

When using this operator, the right-hand variables is not verified to be correct; each of the three replicas of the right-hand value is complemented as it is, for a faster execution. As a consequence, complementing a corrupted right-hand value, returns a corrupted value.

To correct, then complement a corrupted right-hand value, one shall use the following construct:

```
TripleData<DataType> a;
~(a.sync());
```

which recovers a before complementing its value.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Instead, when the complemented value must casted to a (DataType):

```
TripleData<DataType> a;
DataType b;
b = ~a;
```

a faster way could be to first cast, then to complement:

```
TripleData<DataType> a;
DataType b;
b = ~((DataType) a);
```

Code Body: TripleData<DataType> result;
result.d1 = ~ d1;
result.d2 = ~ d2;
result.d3 = ~ d3;
return result;

47.1.40. operator>>

Signature: operator>>(val : int) : TripleData

Overwrites the logic operator **a >> b** when a is redundant TripleData<DataType>, while b is a byte (often, a constant). It right-shifts each replica of the left-hand variable a by the number of positions defined by the right-hand variable (or constant) b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value is right-shifted as it is by the value of the right-hand variable, for a faster execution. As a consequence, right-shifting a corrupted left-hand value, leaves the value corrupted.

To right-shift and correct a corrupted left-hand value, one shall use the following construct:

```
TripleData<DataType> a, b, c;
byte b;
c = (DataType) a >> b; // or, alternatively
c = (DataType) a >> 2;
```

where type casting TripleData<DataType> a to (DataType) recovers a correct value from its three replicas (by voting). The correct value is then copied identically to the three replicas of c.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 >> val;
result.d2 = d2 >> val;
result.d3 = d3 >> val;
return result;

47.1.41. operator<<

Signature: operator<<(val : int) : TripleData

Overwrites the logic operator **a << b** when a is redundant TripleData<DataType>, while b is a byte (often, a constant). It left-shifts each replica of the left-hand variable a by the number of positions defined by the right-hand variable (or constant) b and returns the result as a redundant TripleData<DataType>. Neither a nor b are affected.

When using this operator, the left-hand variable is not verified to be correct; each of the three replicas of the left-hand value is left-shifted as it is by the value of the right-hand variable, for a faster execution. As a consequence, left-shifting a corrupted left-hand value, leaves the value corrupted.

To left-shift and correct a corrupted left-hand value, one shall use the following construct:

```
            TripleData<DataType> a, b, c;
            byte b;
            c = (DataType) a << b; // or, alternatively
            c = (DataType) a << 2;
```

where type casting TripleData<DataType> a to (DataType) recovers a correct value from its three replicas (by voting). The correct value is then copied identically to the three replicas of c.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: TripleData<DataType> result;
result.d1 = d1 << val;
result.d2 = d2 << val;
result.d3 = d3 << val;
return result;

47.1.42. operator>>=

Signature: operator>>=(val : int) : TripleData

Overwrites the logic operator **a >>= b** when a is redundant TripleData<DataType> while b is byte.

It right-shifts each replica of the left-hand variable a by the number of positions identified by the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, the left-hand variables is not verified to be correct; each of the three replicas of the left-hand value is right-shifted as it is by the value of the right-hand variable, for a faster execution. As a consequence, left-shifting a corrupted left-hand value, leaves the value corrupted.

To right-shift and correct a corrupted left-hand value, one shall use either of the following constructs:

```
            TripleData<DataType> a;
            byte b;
            a >>= b;
            a.sync();
```

or

```
            TripleData<DataType> a;
            byte b;
            a = (DataType) a >> b;
```

which recovers a before using its value.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 >>= val;
d2 >>= val;
d3 >>= val;
return *this; // Return a reference to myself

47.1.43. operator<<=

Signature: operator<<=(val : int) : TripleData

Overwrites the logic operator **a >>= b** when a is redundant TripleData<DataType> while b is byte.

It left-shifts each replica of the left-hand variable a by the number of positions identified by the right-hand variable (or constant) b, stores the result back to a and returns the result as a redundant TripleData<DataType>. Variable a is affected, while b is not affected.

When using this operator, the left-hand variables is not verified to be correct; each of the three replicas of the left-hand value is left-shifted as it is by the value of the right-hand variable, for a faster execution. As a consequence, left-shifting a corrupted left-hand value, leaves the value corrupted.

To left-shift and correct a corrupted left-hand value, one shall use either of the following constructs:

```
TripleData<DataType> a;
byte b;
a <<= b;
a.sync();
```

or

```
TripleData<DataType> a;
byte b;
a = (DataType) a << b;
```

which recovers a before using its value.

Because of the longer execution time of voting, it is strongly suggested to use it only when really required.

Code Body: d1 <<= val;
d2 <<= val;
d3 <<= val;
return *this; // Return a reference to myself

47.1.44. operator==

Signature: operator==(val : DataType) : bool

Overwrites the logic operator **a == b** when **a** is redundant TripleData<DataType> while **b** is DataType.

It first votes **a** and compares the result against **b** and returns result as a bool.

Neither variable **a** nor **b** is affected.

Code Body: return ((DataType) *this == val);

47.1.45. operator==

Signature: operator==(val : TripleData) : bool

Overwrites the logic operator **a == b** when both **a** and **b** are redundant TripleData<DataType>.

It first votes **a** and **b** separately, then it compares voting result and returns comparison result as a bool.

Neither variable **a** nor **b** is affected.

Code Body: return ((DataType) *this == (DataType) val);

47.1.46. operator!=

Signature: operator!=(val : DataType) : bool

Overwrites the logic operator **a != b** when **a** is redundant TripleData<DataType> while **b** is DataType.

It first votes **a** and compares the result against **b** and returns result as a bool.

Neither variable **a** nor **b** is affected.

Code Body: return ((DataType) *this != val);

47.1.47. operator!=

Signature: operator!=(val : TripleData) : bool

Overwrites the logic operator **a == b** when both **a** and **b** are redundant TripleData<DataType>.

It first votes **a** and **b** separately, then it compares voting result and returns comparison result as a bool.

Neither variable **a** nor **b** is affected.

Code Body: return ((DataType) *this != (DataType) val);

47.1.48. sync

Signature: sync() : DataType

Attempts to recover the correct value of the data stored into the class, starting from its three replicas.

If the three replicas have not been corrupted, or at most one of them has been corrupted, the sync(): DataType operation will recover the correct value by majority voting, then restores this value into the corrupted replica (the other replicas are not touched).

If two or more replicas are corrupted, the error is unrecoverable and the sync(): DataType will return the value of the first replica.

This operation differs from only voting (as performed by the casting operator operator DataType()) as the latter only returns a potentially correct value without correcting the three replicas, while the former also recovers the data into the class.

Code Body:
```
if (d1 == d2) {
 if (d1 == d3) {
  return d1;
  }
 else {
  d3 = d1;
  return d1;
  }
} else {
 if (d1 == d3) {
  d2 = d1;
  return d1;
  }
 else if (d2 == d3) {
  d1 = d3;
  return d1;
  }
 else
  return d1;
 }
```

47.1.49. check

Signature: check() : bool

Returns true when a correct value can be recovered, by using either the type casting operator operator DataType() or the sync(): DataType operator.

In other words, it returns true when either the three replicas are identical or at most one replica is different from the other two.

When all the three replicas differ, it returns false.
There is a potential situation (usually with very low probability) that two errors have affected two different replicas in the same way; in this case, the check(): bool operator will return true, but the correct value CANNOT be recovered.

Code Body: if (d1 == d2) return true;
if (d1 == d3) return true;
if (d2 == d3) return true;
else return false;

47.1.50. vote

Signature: vote() : DataType, Code Body: return (DataType) (*this);

47.1.51. voteBit

Signature: voteBit(mask : DataType) : bool, Code Body: if ((d1&mask)==(d2&mask)) return (d1&mask);
else if ((d1&mask)==(d3&mask)) return (d1&mask);
else return (d2&mask);

47.1.52. agree

Signature: agree() : bool, Code Body: return check();

47.1.53. agreeBit

Signature: agreeBit(mask : DataType) : bool, Code Body: if (((d1&mask) == (d2&mask)) && ((d1&mask) == (d3&mask)))
 return true;
else
 return false;

47.1.54. disagree

Signature: disagree() : WHO, Code Body: if (agree()) return NONE;
if (d1==d2) return CC;
if (d1==d3) return BB;
return AA;

47.1.55. disagreeBit

Signature: disagreeBit(mask : DataType) : WHO, Code Body: if (agreeBit(mask)) return NONE;
if ((d1&mask)==(d2&mask)) return CC;
if ((d1&mask)==(d3&mask)) return BB;
return AA;

47.1.56. a

Signature: a() : DataType
Returns first replica of the data.
Code Body: return d1;

47.1.57. b

Signature: b() : DataType
Returns second replica of the data.
Code Body: return d2;

47.1.58. c

Signature: c() : DataType
Returns third replica of the data.

Code Body: return d3;

47.1.59. x

Signature: x(who : WHO) : DataType

Returns either:

- first replica of the data, if who==AA;
- second replica of the data, if who==BB;
- third replica of the data, if who==CC;
- first replica of the data, for any other value of who;

Code Body: switch (who){
case AA: return d1; break;
case BB: return d2; break;
case CC: return d3; break;
default: return d1; break;
}

47.1.60. set

Signature: set(mask : TripleData) : void

Sets to 1, in the three replicas of data stored in the class, all the bits which are set to 1 in the argument mask, while the other bits are not affected.

Exactly equivalent to the operator operator|=(val: TripleData&): TripleData&. In practice, the two forms:

```
TripleData<DataType> a, b;
a.set(b);
a |= b;
```

have identical effect. Despite identical, both are left for compatibility with old programs.

For more details and side effects of this operation, see documentation of operator|=(val: TripleData&): TripleData&.

Code Body: (*this) |= mask;

47.1.61. set

Signature: set(mask : DataType) : void

Sets to 1, in the three replicas of data stored in the class, all the bits which are set to 1 in the argument mask, while the other bits are not affected.

Exactly equivalent to the operator operator|=(val: DataType&): TripleData&. In practice, the two forms:

```
TripleData<DataType> a;
DataType b;
a.set(b);
a |= b;
```

have identical effect. Despite identical, both are left for compatibility with old programs.

For more details and side effects of this operation, see documentation of operator|=(val: DataType&): TripleData&.

Code Body: (*this) |= mask;

47.1.62. reset

Signature: reset(mask : TripleData) : void

Reets to 0, in the three replicas of data stored in the class, all the bits which are set to 1 in the argument mask, while the other bits are not affected.

Exactly equivalent to the operator operator&=(val: TripleData&): TripleData& associated with operator~(): TripleData&. In practice, the two forms:

```
TripleData<DataType> a, b;
a.reset(b);
a &= ~b;
```

have identical effect. Despite identical, both are left for compatibility with old programs.

For more details and side effects of this operation, see documentation of operator&=(val: TripleData&): TripleData& and operator~(): TripleData&.

Code Body: (*this) &= ~mask;

47.1.63. reset

Signature: reset(mask : DataType) : void

Sets to 0, in the three replicas of data stored in the class, all the bits which are set to 1 in the argument mask, while the other bits are not affected.

Exactly equivalent to the operator operator&=(val: DataType&): TripleData& associated with ~. In practice, the two forms:

```
TripleData<DataType> a;
DataType b;
a.reset(b);
a &= ~b;
```

have identical effect. Despite identical, both are left for compatibility with old programs.

For more details and side effects of this operation, see documentation of operator&=(val: DataType&): TripleData&.

Code Body: (*this) &= ~mask;

47.1.64. write

Signature: write(val : TripleData) : void

Writes the content of argument val into the three replicas of data stored in the class.

Exactly equivalent to the operator operator=(val: TripleData&): TripleData&. In practice, the two forms:

```
TripleData<DataType> a, b;
a.write(b);
a = b;
```

have identical effect. Despite identical, both are left for compatibility with old programs.

For more details and side effects of this operation, see documentation of operator=(val: TripleData&): TripleData&.

Code Body: (*this) =val;

47.1.65. write

Signature: write(val : DataType) : void

Writes the content of argument val into the three replicas of data stored in the class.

Exactly equivalent to the operator operator=(val: DataType): TripleData&. In practice, the two forms:

```
            TripleData<DataType> a;
            DataType b;
            a.write(b);
            a = b;
```

have identical effect. Despite identical, both are left for compatibility with old programs.

For more details and side effects of this operation, see documentation of operator=(val: DataType): TripleData&.

Code Body: (*this) =val;

47.1.66. write

Signature: write(data : TripleData, mask : TripleData) : void

Writes partially, in the three replicas of data stored in the class, the value of the argument data, but only in all the bits which are set to 1 in the argument mask, while the other bits are not affected.

In practice:

```
            TripleData<DataType> a, data, mask;
            a = 0x1357;
            data = 0xFF00;
            mask = 0x0FF0;
            a.write(data, mask);
```

leaves first nibble (1) and last nibble (7) unchanged, then writes F0 in the second and third nibble.

So, at the end, a will contain the value 0x1F07.

This operation is a combination of the operator=(val: TripleData&): TripleData&, operator&(val: TripleData&): TripleData&,, operator|(val: TripleData&): TripleData& and operator~(): TripleData&. For more details and side effects of this operation, see documentation of these operators..

Code Body: (*this) = (data & mask) | ((*this) & ~mask);

47.1.67. write

Signature: write(data : DataType, mask : DataType) : void

Writes partially, in the three replicas of data stored in the class, the value of the argument data, but only in all the bits which are set to 1 in the argument mask, while the other bits are not affected.

In practice:

```
            TripleData<DataType> a,;
            DataType data, mask;
            a = 0x1357;
            data = 0xFF00;
            mask = 0x0FF0;
            a.write(data, mask);
```

leaves first nibble (1) and last nibble (7) unchanged, then writes F0 in the second and third nibble.

So, at the end, a will contain the value 0x1F07.

This operation is a combination of the operator=(val: DataType): TripleData&, operator&(val: DataType&): TripleData&, operator|(val: DataType&): TripleData& and ~. For more details and side effects of these operations, see documentation of these operators.

Code Body: (*this) = (data & mask) | ((*this) & (DataType)(~mask));

47.1.68. test

Signature: test(data : TripleData) : bool

Returns true if and only if the bits which are set in data are set also in the data stored in the class.

In practice:

```
TripleData<DataType> a, data;
a = 0x1F07;
data = 0x1F00;
a.test(data);
```

will return true, as all the bits which are set in data are also set in a, while if:

```
a = 0x15D7;
data = 0xFF00;
a.test(data);
```

will return false, as some of the bits which are set in data are NOT set in a.

This operation is equivalent to test(data, data).

For more details and side effects of this operation, see documentation of these operators.

Code Body: return (*this).test(data, data);

47.1.69. test

Signature: test(data : DataType) : bool

Returns true if and only if the bits which are set in data are set also in the data stored in the class.

In practice:

```
TripleData<DataType> a, data;
a = 0x1F07;
data = 0x1F00;
a.test(data);
```

will return true, as all the bits which are set in data are also set in a, while if:

```
a = 0x15D7;
data = 0xFF00;
a.test(data);
```

will return false, as some of the bits which are set in data are NOT set in a.

This operation is equivalent to test(data, data).

For more details and side effects of this operation, see documentation of these operators.

Code Body: return (*this).test(data, data);

47.1.70. test

Signature: test(data : TripleData, mask : TripleData) : bool

Returns true if and only if ALL bits of data stored in the class, corresponding to bits set to 1 in argument mask are identical to data.

In practice:

```
TripleData<DataType> a, data, mask;
a = 0x1F07;
data = 0xFF00;
mask = 0x0FF0;
a.test(data, mask);
```

will return true, as all the bits in the second and third nibble (namely, all those bits where mask is 1) of a are identical to data, while if:

```
a = 0x15D7;
data = 0xFF00;
mask = 0x0FF0;
```

```
        a.test(data, mask);
```
will return false, as some of the bits in the second and third nibble (namely, all those bits where mask is 1) of a are NOT identical to data.

This operation is a combination of the operator=(val: TripleData&): TripleData&, operator&(val: TripleData&): TripleData&,, operator|(val: TripleData&): TripleData& and operator~(): TripleData&. For more details and side effects of this operation, see documentation of these operators.

Code Body: return (((*this) & mask) == (data & mask));

47.1.71. test

Signature: test(data : DataType, mask : DataType) : bool

Returns true if and only if ALL bits of data stored in the class, corresponding to bits set to 1 in argument mask are identical to data.

In practice:

```
        TripleData<DataType> a;
        DataType data, mask;
        a = 0x1F07;
        data = 0xFF00;
        mask = 0x0FF0;
        a.test(data, mask);
```
will return true, as all the bits in the second and third nibble (namely, all those bits where mask is 1) of a are identical to data, while if:

```
        a = 0x15D7;
        data = 0xFF00;
        mask = 0x0FF0;
        a.test(data, mask);
```
will return false, as some of the bits in the second and third nibble (namely, all those bits where mask is 1) of a are NOT identical to data.

This operation is a combination of the operator=(val: DataType): TripleData&, operator&(val: DataType&): TripleData&, operator|(val: DataType&): TripleData& and ~. For more details and side effects of these operations, see documentation of these operators.

Code Body: return (((*this) & mask) == (DataType)(data & mask));

47.1.72. testExact

Signature: testExact(data : TripleData) : bool

Returns true if and only if ALL bits of data stored in the class are identical to data.

In practice:

```
TripleData<DataType> a, data;
a = 0x1F07;
data = 0x1F07;
a.test(data);
```
will return true, as all the bits of a are identical to data, while if:

```
a = 0x15D7;
data = 0xFF00;
a.test(data);
```
will return false, as some of the bits of a are NOT identical to data.

This operation is a combination of the operator=(val: TripleData&): TripleData&, operator&(val: TripleData&): TripleData&,, operator|(val: TripleData&): TripleData& and

operator~(): TripleData&. For more details and side effects of this operation, see documentation of these operators.

Code Body: return ((*this) == data);

47.1.73. testExact

Signature: testExact(data : DataType) : bool

Returns true if and only if ALL bits of data stored in the class are identical to data.

In practice:

TripleData<DataType> a;
a = 0x1F07;
data = 0x1F07;
a.test(data);

will return true, as all the bits of a are identical to data, while if:

a = 0x15D7;
data = 0xFF00;
a.test(data);

will return false, as some of the bits of a are NOT identical to data.

This operation is a combination of the operator=(val: DataType): TripleData&, operator&(val: DataType&): TripleData&, operator|(val: DataType&): TripleData& and ~. For more details and side effects of these operations, see documentation of these operators.

Code Body: return ((*this) == data);

47.1.74. swap

Signature: swap(data : TripleData) : void, Code Body: DataType tmp;

```
tmp = d1;
d1 = data.d1;
data.d1 = tmp;
tmp = d2;
d2 = data.d2;
data.d2 = tmp;
tmp = d3;
d3 = data.d3;
data.d3 = tmp;
```

48. Input Cleaner - SW section

48.1. VOTER

La FSA_IN_2 è una automa, che descrive come il microcontrollore ME, interagisce con gli altri due OTHER_A e OTHER_B in funzione delle transizioni del bit d'ingresso scritto InfoSharing0: InfoSharing.

Si considera quindi l'ingresso della FSA_IN_2, libero da bouncing.

La FSA, ma la macchina è in grado di gestire evventuali fault:

1

FSA_2 è un automa mirato all'acquisizione di un unico segnale elettromeccanico disponibile in tre copie provenienti da altrettanti trasduttori distinti, precedentemente ripuliti da rimbalzi. L'automa valuta affidabilmente il valore del segnale anche nel caso che al massimo uno dei trasduttori (o catene di acquisizione) soffra di un qualsivoglia guasto.

La FSA_IN_2 è in grado di segnalare eventuali errori.

48.1.1. dummy1

Signature: -dummy1 : t_ID

48.1.2. dummy2

Signature: -dummy2 : Type_fault

48.1.3. Fault0

-Fault0 : Fault

48.1.4. Timer0

-Timer0 : Timer

48.1.5. InfoSharing0

-InfoSharing0 : InfoSharing

48.1.6. Reliable_output

-Reliable_output : PP_without_Fault

48.1.7. start_T_JITTER

-start_T_JITTER : int

48.1.8. cnt_trans_ME

-cnt_trans_ME : int

48.1.9. cnt_trans_A

-cnt_trans_A : int

Variable to count the number of transactions sent by the INDEX_BIT input bits of OTHER_A.

48.1.10. cnt_trans_B

-cnt_trans_B : int

Variable to count the number of transactions sent by the INDEX_BIT input bits of OTHER_B.

48.1.11. Reporter0

-Reporter0 : Reporter

48.1.12. exec

Signature: exec() : void, Code Body: exec_VME ();
exec_VA ();
exec_VB ();
exec_VOTER ();

48.1.13. exec_VOTER

Signature: exec_VOTER() : void, Code Body: Type_state_VOTER_0 state_VOTER_0 = Reporter0.getState_VOTER_0 ();
Type_state_VOTER_0 new_state_VOTER_0 = state_VOTER_0;

```
switch (state_VOTER_0)
{
        // TRANSITIONS
        case OUT_0:
                if ( vote () )
                {
                        new_state_VOTER_0 = VOTING;
                }
                break;
        case OUT_1:
```

```
                if ( vote () == false)
                {
                        new_state_VOTER_0 = VOTING;
                }
                break;

        case VOTING:
                if (overcome_T_JITTER () && vote ())
                {
                        new_state_VOTER_0 = OUT_1;
                }
                if(overcome_T_JITTER () && (vote () == false))
                {
                        new_state_VOTER_0 = OUT_0;
                }
                break;
}

if ( new_state_VOTER_0 != state_VOTER_0)
{
        switch (state_VOTER_0)
        {
                case OUT_0:
                case OUT_1:
                        start_T_JITTER = Timer0.read();
                        break;
                case VOTING:
                        if ( overcome_T_JITTER ())
                        {
                                write(vote());
                                detect_fault();

                        }
                        break;
        }
        if ((new_state_VOTER_0 == OUT_0) || (new_state_VOTER_0 == OUT_1))
        {
                // RESET counters
                cnt_trans_ME = 0;
                cnt_trans_A = 0;
                cnt_trans_B = 0;
        }
}
Reporter0.setState_VOTER_0 ( new_state_VOTER_0);
```

48.1.14. exec_VME

Signature: exec_VME() : void, Code Body: if (Reporter0.getState_VME () == State_ME_IN0)

```
{
        if (MYself () == true)
        {
                Reporter0.setState_VME (State_ME_IN1);
                cnt_trans_ME ++;
```

```
        }
}
else
{
        if (MYself () == false)
        {
                Reporter0.setState_VME (State_ME_IN0);
                cnt_trans_ME ++;
        }
}
```

48.1.15. exec_VA

Signature: exec_VA() : void, Code Body: if (Reporter0.getState_VA () == State_OTHER_A_IN0)

```
{
        if (AA () == true)
        {
                Reporter0.setState_VA (State_OTHER_A_IN1);
                cnt_trans_A ++;
        }
}
else
{
        if (AA () == false)
        {
                Reporter0.setState_VA (State_OTHER_A_IN0);
                cnt_trans_A ++;
        }
}
```

48.1.16. exec_VB

Signature: exec_VB() : void, Code Body: if (Reporter0.getState_VB () == State_OTHER_B_IN0)

```
{
        if (BB () == true)
        {
                Reporter0.setState_VB (State_OTHER_B_IN1);
                cnt_trans_B ++;
        }
}
else
{
        if (BB () == false)
        {
                Reporter0.setState_VB (State_OTHER_B_IN0);
                cnt_trans_B ++;
        }
}
```

48.1.17. init

Signature: init() : void, Code Body: init_VME ();

```
init_VA ();
init_VB ();
init_VOTER_0 ();
```

48.1.18. init_VOTER_0

Signature: init_VOTER_0() : void, Code Body: if (vote () == false)

```
{
        Reporter0.setState_VOTER_0 (OUT_0);
}
else
{
        Reporter0.setState_VOTER_0 (OUT_1);
}
cnt_trans_ME = 0;
cnt_trans_A = 0;
cnt_trans_B = 0;
```

48.1.19. init_VME

Signature: init_VME() : void, Code Body: if(MYself() == false)

```
{
        Reporter0.setState_VME(State_ME_IN0);
}
else
{
        Reporter0.setState_VME(State_ME_IN1);
}
cnt_trans_ME = 0;
```

48.1.20. init_VA

Signature: init_VA() : void, Code Body: if(AA () == false)

```
{
        Reporter0.setState_VA (State_OTHER_A_IN0);
}
else
{
        Reporter0.setState_VA (State_OTHER_A_IN1);
}
cnt_trans_A = 0;
```

48.1.21. init_VB

Signature: init_VB() : void, Code Body: if(BB() == false)

```
{
        Reporter0.setState_VB(State_OTHER_B_IN0);
}
else
{
        Reporter0.setState_VB(State_OTHER_B_IN1);
}
cnt_trans_B = 0;
```

48.1.22. write

Signature: write(value : bool) : void, Code Body: Reliable_output.writeBit (INDEX_BIT, value);

48.1.23. detect_fault

Signature: detect_fault() : bool
individua l'errore, esaminando se sono tutti d'accordo
restituisce 1 se è stato rilevato un fault, e 0 in caso contrario

Code Body: write (vote());

```
YYY::t_ID accused, guilty;
YYY::Type_fault name_fault;
bool value_guilty;
int counter;

accused = disagree_A_B_ME();
switch(accused)       // If there is'n an opposed
{
        case NOBODY:
                if (cnt_trans_ME != 1)
                {
                        counter = cnt_trans_ME;
                        value_guilty = MYself();
                        guilty = ME;
                }
                else if (cnt_trans_A != 1)
                {
                        counter = cnt_trans_A;
                        value_guilty = AA();
                        guilty = OTHER_A;
                }
                else if (cnt_trans_B != 1)
                {
                        counter = cnt_trans_B;
                        value_guilty = BB();
                        guilty = OTHER_B;
                }
                else
                        return false;
                break;
        case ME:
                guilty = ME;
                counter = cnt_trans_ME;
                value_guilty = MYself();
                break;
        case OTHER_A:
                guilty = OTHER_A;
                counter = cnt_trans_A;
                value_guilty = AA();
                break;
        case OTHER_B:
                guilty = OTHER_B;
                counter = cnt_trans_B;
                value_guilty = BB();
                break;
}

// Identifie the type of fault

if(counter == 0)              // there is a STUCK AT
```

```
{
        if(value_guilty == true)
        {
                name_fault = STUCK_AT_1;
        }
        if(value_guilty == false)
        {
                name_fault = STUCK_AT_0;
        }
}
if ((counter > 1) && (counter < MAX_TRANS))
{
        name_fault = GLITCH;
}
else if (counter >=  MAX_TRANS)
{
        name_fault = CRAZY_BOUNCING;
}
```

Fault0.send_VOTER (guilty, name_fault, value_guilty, counter, vote (), MYself (), AA (), BB (), cnt_trans_ME, cnt_trans_A, cnt_trans_B);
return true;

48.1.24. disagree_A_B_ME

Signature: disagree_A_B_ME() : t_ID

Funzione che esamina i valori di A B e ME e restituisce NOBODY se sono tutti d'accordo, se invece c'è un disaccordo, restituisce il t_ID di chi è in disaccordo.

Code Body: return InfoSharing0.disagreeBit(INDEX_WORD, 1<<INDEX_BIT);

48.1.25. VOTER

Signature: VOTER()

48.1.26. VOTER

Signature: VOTER(dummy : int)

48.1.27. overcome_T_JITTER

Signature: overcome_T_JITTER() : bool, Code Body: return Timer0.cmp(start_T_JITTER, T_MAX_JITTER);

48.1.28. MYself

Signature: MYself() : bool

It returns the value of the input bit position INDEX_BIT of the (model element not found) microcontroller, which is carrying out the data processing.

Code Body: return InfoSharing0.readBit_ME(INDEX_WORD, 1<<INDEX_BIT);

48.1.29. AA

Signature: AA() : bool

It returns the value of the input bit position INDEX_BIT of the OTHER_A microcontroller, which was read by InfoSharing0: InfoSharing.

Code Body: return InfoSharing0.readBit_A(INDEX_WORD, 1<<INDEX_BIT);

48.1.30. BB

Signature: BB() : bool

It returns the value of the input bit position INDEX_BIT of the OTHER_B microcontroller, which was read by InfoSharing0: InfoSharing.

Code Body: return InfoSharing0.readBit_B(INDEX_WORD, 1<<INDEX_BIT);

48.1.31. vote

Signature: vote() : bool

It returns the voting bits value of the three microcontroller on the bit input SH_IN0 to INDEX_BIT position.

Code Body: bool tmp;
InfoSharing0.voteBit(INDEX_WORD, 1<<INDEX_BIT, tmp);
return tmp;

48.2. DEBOUNCE

48.2.1. Timer0

-Timer0 : Timer

48.2.2. start_T_Bounce_T_Pulse

-start_T_Bounce_T_Pulse : int

Start time of bouncing signal, and time of pulse

48.2.3. pp0

-pp0 : ParallelPort

48.2.4. InfoSharing0

-InfoSharing0 : InfoSharing

48.2.5. Fault0

Fault0 : Fault

48.2.6. n_trans_IN

-n_trans_IN : int

valore delle transizioni effettuate dall'ingresso tra una scrittura su infoSharing e l'altra

48.2.7. Reporter0

-Reporter0 : Reporter

48.2.8. exec

Signature: exec() : void, Code Body: // Transitions state machine

```
Type_State_DEBOUNCE state_DEBOUNCE = Reporter0.getState_DEBOUNCE ();
Type_State_DEBOUNCE new_state_DEBOUNCE = state_DEBOUNCE;
switch(state_DEBOUNCE)
{
        case IN_0:
                if (is1 ())
                {
                        new_state_DEBOUNCE = IN_1_from_0;
                }
                break;

        case IN_1_from_0 :
                if (is0 () && (overcome_T_MAX_BOUNCE () == false))
                {
                        new_state_DEBOUNCE = IN_0_from_1_from_0;
```

```
                }
                if (overcome_T_MAX_BOUNCE())
                {
                        new_state_DEBOUNCE = IN_1;
                }
                break;

        case IN_0_from_1_from_0:
                if (is1 () && overcome_T_MAX_BOUNCE () == false)
                {
                        new_state_DEBOUNCE = IN_1_from_0;
                }
                if (overcome_T_MAX_BOUNCE ())
                {
                        new_state_DEBOUNCE = IN_0;
                }
                break;

        case IN_1:
                if (is0 ())
                {
                        new_state_DEBOUNCE = IN_0_from_1;
                }
                break;

        case IN_0_from_1 :
                if (is1 () && (overcome_T_MAX_BOUNCE() == false))
                {
                        new_state_DEBOUNCE = IN_1_from_0_from_1;
                }
                if (overcome_T_MAX_BOUNCE())
                {
                        new_state_DEBOUNCE = IN_0;
                }
                break;


        case IN_1_from_0_from_1:
                if (is0 () && (overcome_T_MAX_BOUNCE () == false))
                {
                        new_state_DEBOUNCE = IN_0_from_1;
                }
                if (overcome_T_MAX_BOUNCE ())
                {
                        new_state_DEBOUNCE = IN_1;
                }
                break;
}
// effect EXIT state


if (new_state_DEBOUNCE != state_DEBOUNCE)
```

```
{
	switch(state_DEBOUNCE)
	{

		case IN_0:
			//EXIT
			start_T_Bounce_T_Pulse = Timer0.read();

			// TRANSITIONS
			if (is1 ())
			{
				n_trans_IN ++;
				if (overcome_T_MIN_PULSE() == false)
				{
					Fault0.send_DEBOUNCE (SHORT_PULSE_0, n_trans_IN);
					start_T_Bounce_T_Pulse = Timer0.read();
				}
			}
			break;

		case IN_1_from_0:
			// EXIT
			// None effect

			// TRANSITIONS
			if (is0() && (overcome_T_MAX_BOUNCE () == false))
			{
				n_trans_IN ++;
			}
			// if (overcome_T_MAX_BOUNCE() ) causes just a change of state
			break;

		case IN_0_from_1_from_0:
			// EXIT
			// None effect

			// TRANSITIONS
			if (is1 () && (overcome_T_MAX_BOUNCE  () == false))
			{
				n_trans_IN ++;
			}
			if (overcome_T_MAX_BOUNCE())
			{
				Fault0.send_DEBOUNCE (NOISE, n_trans_IN);
			}
			break;

		case IN_1:
			//EXIT
			start_T_Bounce_T_Pulse = Timer0.read();

			// TRANSITIONS
```

```
                    if (is0 ())
                    {
                            n_trans_IN ++;
                            if (overcome_T_MIN_PULSE() == false)
                            {
                                    Fault0.send_DEBOUNCE (SHORT_PULSE_1, n_trans_IN);
                                    start_T_Bounce_T_Pulse = Timer0.read();
                            }
                    }
                    break;

            case IN_0_from_1:
                    // EXIT
                    // None effect

                    // TRANSITIONS
                    if (is1() && (overcome_T_MAX_BOUNCE () == false))
                    {
                            n_trans_IN ++;
                    }
                    // if (is0 () && overcome_T_MAX_BOUNCE() ) causes just a change of state
                    break;
            case IN_1_from_0_from_1:
                    // EXIT
                    // None effect

                    // TRANSITIONS
                    if (is0 () && (overcome_T_MAX_BOUNCE () == false))
                    {
                            n_trans_IN ++;
                    }
                    if (is1 () && overcome_T_MAX_BOUNCE())
                    {
                            Fault0.send_DEBOUNCE(NOISE, n_trans_IN);
                    }
                    break;


    }

    // effect ENTRY in state

    switch (new_state_DEBOUNCE)
    {
            case IN_0:
                    write (false);
                    n_trans_IN = 0;
                    break;
            case IN_1:
                    write (true);
                    n_trans_IN = 0;
                    break;
```

```
                default:
                        break;
        }
}
```

Reporter0.setState_DEBOUNCE (new_state_DEBOUNCE);

48.2.9. init

Signature: init() : void, Code Body: Reporter0.setState_DEBOUNCE (IN_0); // initial state

48.2.10. is0

Signature: is0() : bool, Code Body: return (pp0.read(1<<INDEX_BIT) == false);

48.2.11. is1

Signature: is1() : bool, Code Body: return (pp0.read(1<<INDEX_BIT) == true);

48.2.12. overcome_T_MIN_PULSE

Signature: overcome_T_MIN_PULSE() : bool, Code Body: return Timer0.cmp(start_T_Bounce_T_Pulse, T_MIN_PULSE);

48.2.13. overcome_T_MAX_BOUNCE

Signature: overcome_T_MAX_BOUNCE() : bool, Code Body: return Timer0.cmp(start_T_Bounce_T_Pulse, T_MAX_BOUNCE);

48.2.14. write

Signature: write(value : bool) : void, Code Body: InfoSharing0.writeBit(INDEX_WORD, (1<<INDEX_BIT), value);

48.2.15. DEBOUNCE

Signature: DEBOUNCE()

48.2.16. DEBOUNCE

Signature: DEBOUNCE(dummy : int)

48.3. InfoSharing

This is a communication infrastructure to help sharing the info among processors in a triple-redundant set of processors.

The systems is supposed to the made of three processors, called respectively:

- ME, that is, the processor on which the actual instance of the InfoSharing is running;
- OTHER_A; that is, the processor connected to ME via the (model element not found) peripheral;
- OTHER_B; that is, the processor connected to ME via the UB: UARTB peripheral;

This class stores three replicas of a set of critical data (SIZE bytes) into attribute SharedData, built using a TripleData structure.

The three replicas of each byte built inside each element of SharedData are as follows:

1. a(): DataType& stores the data of ME
2. b(): DataType& stores the virtually identical data periodically received from OTHER_A
3. c(): DataType& stores the virtually identical data periodically received from OTHER_B

This class has operations to:

1. write data of ME into shared storege, by means of writeByte(index: T_INDEX, val: byte): void;
2. read data from either of three processors Computing Unit;

3. vote data to return the most likely value of each bit (respectively, byte) by means of voteBit(index: T_INDEX, mask_bit: int, val: bool&): bool (respectively, voteByte(index: T_INDEX, val: byte&): byte);
4. send ME's data to both OTHER_A and OTHER_B, by means of send(): void.

48.3.1. SharedData

-SharedData[SIZE]

48.3.2. UA

Signature: -UA : UARTA

48.3.3. UB

Signature: -UB : UARTB

48.3.4. msg

Signature: -msg : byte[SIZE]

48.3.5. valBit

Signature: -valBit : bool

48.3.6. valByte

Signature: -valByte : byte

48.3.7. writeByte

Signature: writeByte(index : T_INDEX, val : byte) : void
Writes the value val into the index-th location of ME's replica in SharedData
Code Body: SharedData[to_int(index)].a() = val;

48.3.8. writeBit

Signature: writeBit(index : T_INDEX, mask_bit : byte, val : bool) : void
Writes the value val into the index-th location of ME's replica in SharedData
Code Body: if (val)

```
  SharedData[to_int(index)].a() |= mask_bit;
else
  SharedData[to_int(index)].a() &= ~ mask_bit;
```

48.3.9. voteBit

Signature: voteBit(index : T_INDEX, mask_bit : int, val : bool) : bool
Votes the three local copies of ME, OTHER_A, OTHER_B in bit mask_bit of location index.
Returns the most likely value (two out of three) into val.
Returns true if all three copies match; false otherwise.

48.3.10. voteByte

Signature: voteByte(index : T_INDEX, val : byte) : byte

48.3.11. send

Signature: send() : void
Sends ME's replica of SharedData to OTHER_A (via (model element not found)) and to OTHER_B (via UB: UARTB)

48.3.12. isr_UB

Signature: isr_UB() : void

48.3.13. readBit_ME

Signature: readBit_ME(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].a() & mask_bit);

48.3.14. readBit_A

Signature: readBit_A(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].b() & mask_bit);

48.3.15. readBit_B

Signature: readBit_B(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].c() & mask_bit);

48.3.16. to_int

Signature: to_int(value : T_INDEX) : int, Code Body: /*

```
switch (value)
{
        case SH_IN:       return  0;
        case SH_OUT:     return  1;
        case SH_AIN0:    return  2;
        case SH_AIN1:    return  3;
        case SH_AIN2:    return  4;
        case SH_AIN3:    return  5;
        case SH_AOUT0: return  6;
        case SH_AOUT1: return  7;
        case GRANT:      return  8;
        default :            return   0;
}
*/
return (int) value;
```

48.3.17. disagreeBit

Signature: disagreeBit(index : T_INDEX, mask_bit : byte) : t_ID, Code Body: return to_t_ID(SharedData[to_int(index)].disagreeBit (mask_bit));

48.3.18. to_t_ID

Signature: to_t_ID(value) : t_ID, Code Body: switch (value)

```
{
     case HardenedData::NONE:          return NOBODY;
     case HardenedData::AA:           return ME;
     case HardenedData::BB:return OTHER_A;
     case HardenedData::CC:return OTHER_B;
}
```

48.3.19. InfoSharing

Signature: InfoSharing()

48.3.20. InfoSharing

Signature: InfoSharing(dummy : int)

48.4. InputCleaner8

La classe InputCleaner8 iceve in ingresso 8 inputs digitali on/off.
Libera singolarmente gli input dal bounce. e memorizza internamente il dato.

bouncing -> una serie di transizioni del segnale digitale da on a off, e viceversa per un tempo inferiore a T_MAX_BOUNCE.

Trasmette i valori debounciati di ciascun ingresso, tramite UART, a due inputcleaner8 gemelli , interni a due moduli esterni che ricevono copia virtualmente identica, degli stessi 8 input.

Riceve copia degli 8 input dai due moduli di cui sopra.
Per ogni input avviene una votazione, basata sulla maggioranza, che dettermina il valore da mandare in uscita, che è considerato il valore corretto.
Inoltre esamina se se si sono verificati alcuni tipi di fault.

48.4.1. D0
D0 : DEBOUNCE

48.4.2. D1
D1 : DEBOUNCE

48.4.3. D2
D2 : DEBOUNCE

48.4.4. D3
D3 : DEBOUNCE

48.4.5. D4
D4 : DEBOUNCE

48.4.6. D5
D5 : DEBOUNCE

48.4.7. D6
D6 : DEBOUNCE

48.4.8. D7
D7 : DEBOUNCE

48.4.9. V0
-V0 : VOTER

48.4.10. V1
-V1 : VOTER

48.4.11. V2
-V2 : VOTER

48.4.12. V3
-V3 : VOTER

48.4.13. V4
-V4 : VOTER

48.4.14. V5
-V5 : VOTER

48.4.15. V6
-V6 : VOTER

48.4.16. V7

-V7 : VOTER

48.4.17. fault0

-fault0 : Fault

48.4.18. isr_CAN

Signature: isr_CAN() : void

48.4.19. interpreta

Signature: interpreta(pacchetto : byte, cmd : byte, data : byte) : void

48.4.20. exec

Signature: exec() : void, Code Body: // Run exec() of 8 DEBOUNCE

```
D0.exec();
D1.exec();
D2.exec();
D3.exec();
D4.exec();
D5.exec();
D6.exec();
D7.exec();

// Run exec() of 8 VOTER

V0.exec();
V1.exec();
V2.exec();
V3.exec();
V4.exec();
V5.exec();
V6.exec();
V7.exec();
```

48.4.21. init

Signature: init() : void, Code Body: // Run init() of 8 DEBOUNCE

```
D0.init();
D1.init();
D2.init();
D3.init();
D4.init();
D5.init();
D6.init();
D7.init();

// Run init() of 8 VOTER

V0.init();
V1.init();
V2.init();
V3.init();
V4.init();
```

V5.init();
V6.init();
V7.init();

49. DEBOUNCE FSA_IN_1

49.1. Reporter

49.1.1. stato_globale

-stato_globale : byte
Formato

V0 V0 VME VA VB D D D

V0 riservato alla FSA principale del VOTER
VME riservato alla FSA contatore degli stati ME
VA riservato alla FSA contatore degli stati A
VB riservato alla FSA contatore degli stati B

D riservato aalla FSA principale di DEBOUNCE

49.1.2. Reporter

Signature: Reporter()

49.1.3. Reporter

Signature: Reporter(dummy : int)

49.1.4. getState_DEBOUNCE

Signature: getState_DEBOUNCE() : Type_State_DEBOUNCE
Maschera gli stati che riguardano DEBOUNCE
Code Body: return ((ReporterX *) this) -> getState_DEBOUNCE(stato_globale); // non ho bisogno di shiftare in questo caso

49.2. Type_fault

La classe enumerativa

49.3. Fault

49.3.1. send_DEBOUNCE

Signature: send_DEBOUNCE(what : Type_fault, n_trans : int) : void
Sends a message that contains the Type_fault detected.

Devo mandare anche l'identificativo MYSELF quindi non serve che glielo passi come parametro
Code Body: // ricorda di mandare il my_name

49.3.2. send_VOTER

Signature: send_VOTER(guilty : t_ID, what : Type_fault, value_guilty : bool, n_trans : int, value_out : bool, value_ME : bool, value_A : bool, value_B : bool, n_trans_ME : int, n_trans_A : int, n_trans_B : int) : void
Sends a message that contains the Type_fault detected.

Code Body: // ricorda di mandare il my_name

49.3.3. Fault

Signature: Fault()

49.3.4. Fault

Signature: Fault(dummy : int)

49.4. InfoSharing

This is a communication infrastructure to help sharing the info among processors in a triple-redundant set of processors.

The systems is supposed to the made of three processors, called respectively:

- ME, that is, the processor on which the actual instance of the InfoSharing is running;
- OTHER_A; that is, the processor connected to ME via the (model element not found) peripheral;
- OTHER_B; that is, the processor connected to ME via the UB: UARTB peripheral;

This class stores three replicas of a set of critical data (SIZE bytes) into attribute SharedData, built using a TripleData structure.

The three replicas of each byte built inside each element of SharedData are as follows:

1. a(): DataType& stores the data of ME
2. b(): DataType& stores the virtually identical data periodically received from OTHER_A
3. c(): DataType& stores the virtually identical data periodically received from OTHER_B

This class has operations to:

1. write data of ME into shared storege, by means of writeByte(index: T_INDEX, val: byte): void;
2. read data from either of three processors Computing Unit;
3. vote data to return the most likely value of each bit (respectively, byte) by means of voteBit(index: T_INDEX, mask_bit: int, val: bool&): bool (respectively, voteByte(index: T_INDEX, val: byte&): byte);
4. send ME's data to both OTHER_A and OTHER_B, by means of send(): void.

49.4.1. SharedData

-SharedData[SIZE]

49.4.2. UA

Signature: -UA : UARTA

49.4.3. UB

Signature: -UB : UARTB

49.4.4. msg

Signature: -msg : byte[SIZE]

49.4.5. valBit

Signature: -valBit : bool

49.4.6. valByte

Signature: -valByte : byte

49.4.7. writeByte

Signature: writeByte(index : T_INDEX, val : byte) : void

Writes the value val into the index-th location of ME's replica in SharedData

Code Body: SharedData[to_int(index)].a() = val;

49.4.8. writeBit

Signature: writeBit(index : T_INDEX, mask_bit : byte, val : bool) : void
Writes the value val into the index-th location of ME's replica in SharedData
Code Body: if (val)

```
  SharedData[to_int(index)].a() |= mask_bit;
else
  SharedData[to_int(index)].a() &= ~ mask_bit;
```

49.4.9. voteBit

Signature: voteBit(index : T_INDEX, mask_bit : int, val : bool) : bool
Votes the three local copies of ME, OTHER_A, OTHER_B in bit mask_bit of location index.
Returns the most likely value (two out of three) into val.
Returns true if all three copies match; false otherwise.

49.4.10. voteByte

Signature: voteByte(index : T_INDEX, val : byte) : byte

49.4.11. send

Signature: send() : void
Sends ME's replica of SharedData to OTHER_A (via (model element not found)) and to OTHER_B (via UB: UARTB)

49.4.12. isr_UB

Signature: isr_UB() : void

49.4.13. readBit_ME

Signature: readBit_ME(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].a() & mask_bit);

49.4.14. readBit_A

Signature: readBit_A(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].b() & mask_bit);

49.4.15. readBit_B

Signature: readBit_B(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].c() & mask_bit);

49.4.16. to_int

Signature: to_int(value : T_INDEX) : int, Code Body: /*

```
switch (value)
{
        case SH_IN:       return  0;
        case SH_OUT:      return  1;
        case SH_AIN0:     return  2;
        case SH_AIN1:     return  3;
        case SH_AIN2:     return  4;
        case SH_AIN3:     return  5;
        case SH_AOUT0: return  6;
        case SH_AOUT1: return  7;
        case GRANT:       return  8;
        default :         return   0;
}
*/
return (int) value;
```

49.4.17. disagreeBit

Signature: disagreeBit(index : T_INDEX, mask_bit : byte) : t_ID, Code Body: return to_t_ID(SharedData[to_int(index)].disagreeBit (mask_bit));

49.4.18. to_t_ID

Signature: to_t_ID(value) : t_ID, Code Body: switch (value)

```
{
 case HardenedData::NONE:        return NOBODY;
 case HardenedData::AA:          return ME;
 case HardenedData::BB:return OTHER_A;
 case HardenedData::CC:return OTHER_B;
}
```

49.4.19. InfoSharing

Signature: InfoSharing()

49.4.20. InfoSharing

Signature: InfoSharing(dummy : int)

49.5. Timer

Timer class contains all information related to the Computing Unit timer functions.

49.5.1. time

-time : int

49.5.2. init

Signature: init() : void
Configures and starts the Timer.
Code Body: time = 0;

49.5.3. cmp

Signature: cmp(t_inizial : int, t_max : int) : bool
Compares the difference between the current value of time provided by the Timer and the t_inizial with the t_max, considering also overflow. It returning 1 if greater than or equal, 0 if different.

49.5.4. read

Signature: read() : int
It provides the actual time measured by the Timer
Code Body: return time;

49.5.5. inc

Signature: inc() : void, Code Body: time++;

49.5.6. Timer

Signature: Timer()

49.5.7. Timer

Signature: Timer(dummy : int)

49.6. DEBOUNCE

49.6.1. Timer0

-Timer0 : Timer

49.6.2. start_T_Bounce_T_Pulse

-start_T_Bounce_T_Pulse : int

Start time of bouncing signal, and time of pulse

49.6.3. pp0

-pp0 : ParallelPort

49.6.4. InfoSharing0

-InfoSharing0 : InfoSharing

49.6.5. Fault0

Fault0 : Fault

49.6.6. n_trans_IN

-n_trans_IN : int

valore delle transizioni effettuate dall'ingresso tra una scrittura su infoSharing e l'altra

49.6.7. Reporter0

-Reporter0 : Reporter

49.6.8. exec

Signature: exec() : void, Code Body: // Transitions state machine

```
Type_State_DEBOUNCE state_DEBOUNCE = Reporter0.getState_DEBOUNCE ();
Type_State_DEBOUNCE new_state_DEBOUNCE = state_DEBOUNCE;
switch(state_DEBOUNCE)
{
        case IN_0:
                if (is1 ())
                {
                        new_state_DEBOUNCE = IN_1_from_0;
                }
                break;

        case IN_1_from_0 :
                if (is0 () && (overcome_T_MAX_BOUNCE () == false))
                {
                        new_state_DEBOUNCE = IN_0_from_1_from_0;
                }
                if (overcome_T_MAX_BOUNCE())
                {
                        new_state_DEBOUNCE = IN_1;
                }
                break;

        case IN_0_from_1_from_0:
                if (is1 () && overcome_T_MAX_BOUNCE () == false)
                {
                        new_state_DEBOUNCE = IN_1_from_0;
                }
```

```
                if (overcome_T_MAX_BOUNCE ())
                {
                        new_state_DEBOUNCE = IN_0;
                }
                break;

        case IN_1:
                if (is0 ())
                {
                        new_state_DEBOUNCE = IN_0_from_1;
                }
                break;

        case IN_0_from_1 :
                if (is1 () && (overcome_T_MAX_BOUNCE() == false))
                {
                        new_state_DEBOUNCE = IN_1_from_0_from_1;
                }
                if (overcome_T_MAX_BOUNCE())
                {
                        new_state_DEBOUNCE = IN_0;
                }
                break;


        case IN_1_from_0_from_1:
                if (is0 () && (overcome_T_MAX_BOUNCE () == false))
                {
                        new_state_DEBOUNCE = IN_0_from_1;
                }
                if (overcome_T_MAX_BOUNCE ())
                {
                        new_state_DEBOUNCE = IN_1;
                }
                break;
}
// effect EXIT state


if (new_state_DEBOUNCE != state_DEBOUNCE)
{
        switch(state_DEBOUNCE)
        {

                case IN_0:
                        //EXIT
                        start_T_Bounce_T_Pulse = Timer0.read();

                        // TRANSITIONS
                        if (is1 ())
                        {
                                n_trans_IN ++;
```

```
                        if (overcome_T_MIN_PULSE() == false)
                        {
                                Fault0.send_DEBOUNCE (SHORT_PULSE_0, n_trans_IN);
                                start_T_Bounce_T_Pulse = Timer0.read();
                        }
                }
                break;

        case IN_1_from_0:
                // EXIT
                // None effect

                // TRANSITIONS
                if (is0() && (overcome_T_MAX_BOUNCE () == false))
                {
                        n_trans_IN ++;
                }
                // if (overcome_T_MAX_BOUNCE() ) causes just a change of state
                break;

        case IN_0_from_1_from_0:
                // EXIT
                // None effect

                // TRANSITIONS
                if (is1 () && (overcome_T_MAX_BOUNCE  () == false))
                {
                        n_trans_IN ++;
                }
                if (overcome_T_MAX_BOUNCE())
                {
                        Fault0.send_DEBOUNCE (NOISE, n_trans_IN);
                }
                break;

        case IN_1:
                //EXIT
                start_T_Bounce_T_Pulse = Timer0.read();

                // TRANSITIONS
                if (is0 ())
                {
                        n_trans_IN ++;
                        if (overcome_T_MIN_PULSE() == false)
                        {
                                Fault0.send_DEBOUNCE (SHORT_PULSE_1, n_trans_IN);
                                start_T_Bounce_T_Pulse = Timer0.read();
                        }
                }
                break;

        case IN_0_from_1:
```

```
                        // EXIT
                        // None effect

                        // TRANSITIONS
                        if (is1() && (overcome_T_MAX_BOUNCE () == false))
                        {
                                n_trans_IN ++;
                        }
                        // if (is0 () && overcome_T_MAX_BOUNCE() ) causes just a change of state
                        break;
                case IN_1_from_0_from_1:
                        // EXIT
                        // None effect

                        // TRANSITIONS
                        if (is0 () && (overcome_T_MAX_BOUNCE () == false))
                        {
                                n_trans_IN ++;
                        }
                        if (is1 () && overcome_T_MAX_BOUNCE())
                        {
                                Fault0.send_DEBOUNCE(NOISE, n_trans_IN);
                        }
                        break;


        }

        // effect ENTRY in state

        switch (new_state_DEBOUNCE)
        {
                case IN_0:
                        write (false);
                        n_trans_IN = 0;
                        break;
                case IN_1:
                        write (true);
                        n_trans_IN = 0;
                        break;
                default:
                        break;
        }
}

Reporter0.setState_DEBOUNCE (new_state_DEBOUNCE);
```

49.6.9. init

Signature: init() : void, Code Body: Reporter0.setState_DEBOUNCE (IN_0); // initial state

49.6.10. is0

Signature: is0() : bool, Code Body: return (pp0.read(1<<INDEX_BIT) == false);

49.6.11. is1

Signature: is1() : bool, Code Body: return (pp0.read(1<<INDEX_BIT) == true);

49.6.12. overcome_T_MIN_PULSE

Signature: overcome_T_MIN_PULSE() : bool, Code Body: return Timer0.cmp(start_T_Bounce_T_Pulse, T_MIN_PULSE);

49.6.13. overcome_T_MAX_BOUNCE

Signature: overcome_T_MAX_BOUNCE() : bool, Code Body: return Timer0.cmp(start_T_Bounce_T_Pulse, T_MAX_BOUNCE);

49.6.14. write

Signature: write(value : bool) : void, Code Body: InfoSharing0.writeBit(INDEX_WORD, (1<<INDEX_BIT), value);

49.6.15. DEBOUNCE

Signature: DEBOUNCE()

49.6.16. DEBOUNCE

Signature: DEBOUNCE(dummy : int)

50. VOTER FSA_IN_2

50.1. Reporter

50.1.1. stato_globale

-stato_globale : byte
Formato

V0 V0 VME VA VB D D D

V0 riservato alla FSA principale del VOTER
VME riservato alla FSA contatore degli stati ME
VA riservato alla FSA contatore degli stati A
VB riservato alla FSA contatore degli stati B

D riservato aalla FSA principale di DEBOUNCE

50.1.2. Reporter

Signature: Reporter()

50.1.3. Reporter

Signature: Reporter(dummy : int)

50.1.4. getState_DEBOUNCE

Signature: getState_DEBOUNCE() : Type_State_DEBOUNCE
Maschera gli stati che riguardano DEBOUNCE
Code Body: return ((ReporterX *) this) -> getState_DEBOUNCE(stato_globale); // non ho bisogno di shiftare in questo caso

50.2. Type_fault

La classe enumerativa

50.3. PP_without_Fault

50.3.1. input_safe

-input_safe : byte

Variable contains a input byte free by faults.

50.3.2. writeBit

Signature: writeBit(mask_bit : byte, value : bool) : void

Write value on the bit value of the bit position variable input_safe

Code Body: if (value)

```
  input_safe |= mask_bit;
else
  input_safe &= ~mask_bit;
```

50.3.3. readBit

Signature: readBit(mask_bit : byte) : bool, Code Body: return (input_safe & mask_bit);

50.3.4. readByte

Signature: readByte() : byte, Code Body: return input_safe;

50.3.5. PP_without_Fault

Signature: PP_without_Fault()

50.3.6. PP_without_Fault

Signature: PP_without_Fault(dummy : int)

50.4. Fault

50.4.1. send_DEBOUNCE

Signature: send_DEBOUNCE(what : Type_fault, n_trans : int) : void

Sends a message that contains the Type_fault detected.

Devo mandare anche l'identificativo MYSELF quindi non serve che glielo passi come parametro

Code Body: // ricorda di mandare il my_name

50.4.2. send_VOTER

Signature: send_VOTER(guilty : t_ID, what : Type_fault, value_guilty : bool, n_trans : int, value_out : bool, value_ME : bool, value_A : bool, value_B : bool, n_trans_ME : int, n_trans_A : int, n_trans_B : int) : void

Sends a message that contains the Type_fault detected.

Code Body: // ricorda di mandare il my_name

50.4.3. Fault

Signature: Fault()

50.4.4. Fault

Signature: Fault(dummy : int)

50.5. InfoSharing

This is a communication infrastructure to help sharing the info among processors in a triple-redundant set of processors.

The systems is supposed to the made of three processors, called respectively:

- ME, that is, the processor on which the actual instance of the InfoSharing is running;
- OTHER_A; that is, the processor connected to ME via the (model element not found) peripheral;
- OTHER_B; that is, the processor connected to ME via the UB: UARTB peripheral;

This class stores three replicas of a set of critical data (SIZE bytes) into attribute SharedData, built using a TripleData structure.

The three replicas of each byte built inside each element of SharedData are as follows:

1. a(): DataType& stores the data of ME
2. b(): DataType& stores the virtually identical data periodically received from OTHER_A
3. c(): DataType& stores the virtually identical data periodically received from OTHER_B

This class has operations to:

1. write data of ME into shared storege, by means of writeByte(index: T_INDEX, val: byte): void;
2. read data from either of three processors Computing Unit;
3. vote data to return the most likely value of each bit (respectively, byte) by means of voteBit(index: T_INDEX, mask_bit: int, val: bool&): bool (respectively, voteByte(index: T_INDEX, val: byte&): byte);
4. send ME's data to both OTHER_A and OTHER_B, by means of send(): void.

50.5.1. SharedData

-SharedData[SIZE]

50.5.2. UA

Signature: -UA : UARTA

50.5.3. UB

Signature: -UB : UARTB

50.5.4. msg

Signature: -msg : byte[SIZE]

50.5.5. valBit

Signature: -valBit : bool

50.5.6. valByte

Signature: -valByte : byte

50.5.7. writeByte

Signature: writeByte(index : T_INDEX, val : byte) : void
Writes the value val into the index-th location of ME's replica in SharedData
Code Body: SharedData[to_int(index)].a() = val;

50.5.8. writeBit

Signature: writeBit(index : T_INDEX, mask_bit : byte, val : bool) : void
Writes the value val into the index-th location of ME's replica in SharedData
Code Body: if (val)
SharedData[to_int(index)].a() |= mask_bit;
else

SharedData[to_int(index)].a() &= ~ mask_bit;

50.5.9. voteBit

Signature: voteBit(index : T_INDEX, mask_bit : int, val : bool) : bool
Votes the three local copies of ME, OTHER_A, OTHER_B in bit mask_bit of location index.
Returns the most likely value (two out of three) into val.
Returns true if all three copies match; false otherwise.

50.5.10. voteByte

Signature: voteByte(index : T_INDEX, val : byte) : byte

50.5.11. send

Signature: send() : void
Sends ME's replica of SharedData to OTHER_A (via (model element not found)) and to OTHER_B (via UB: UARTB)

50.5.12. isr_UB

Signature: isr_UB() : void

50.5.13. readBit_ME

Signature: readBit_ME(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].a() & mask_bit);

50.5.14. readBit_A

Signature: readBit_A(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].b() & mask_bit);

50.5.15. readBit_B

Signature: readBit_B(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].c() & mask_bit);

50.5.16. to_int

Signature: to_int(value : T_INDEX) : int, Code Body: /*

```
switch (value)
{
        case SH_IN:       return  0;
        case SH_OUT:      return  1;
        case SH_AIN0:     return  2;
        case SH_AIN1:     return  3;
        case SH_AIN2:     return  4;
        case SH_AIN3:     return  5;
        case SH_AOUT0: return  6;
        case SH_AOUT1: return  7;
        case GRANT:       return  8;
        default :             return   0;
}
*/
return (int) value;
```

50.5.17. disagreeBit

Signature: disagreeBit(index : T_INDEX, mask_bit : byte) : t_ID, Code Body: return to_t_ID(SharedData[to_int(index)].disagreeBit (mask_bit));

50.5.18. to_t_ID

Signature: to_t_ID(value) : t_ID, Code Body: switch (value)

```
{
    case HardenedData::NONE:        return NOBODY;
    case HardenedData::AA:          return ME;
    case HardenedData::BB:return OTHER_A;
    case HardenedData::CC:return OTHER_B;
}
```

50.5.19. InfoSharing

Signature: InfoSharing()

50.5.20. InfoSharing

Signature: InfoSharing(dummy : int)

50.6. Timer

Timer class contains all information related to the Computing Unit timer functions.

50.6.1. time

-time : int

50.6.2. init

Signature: init() : void
Configures and starts the Timer.
Code Body: time = 0;

50.6.3. cmp

Signature: cmp(t_inizial : int, t_max : int) : bool
Compares the difference between the current value of time provided by the Timer and the t_inizial with the t_max, considering also overflow. It returning 1 if greater than or equal, 0 if different.

50.6.4. read

Signature: read() : int
It provides the actual time measured by the Timer
Code Body: return time;

50.6.5. inc

Signature: inc() : void, Code Body: time++;

50.6.6. Timer

Signature: Timer()

50.6.7. Timer

Signature: Timer(dummy : int)

50.7. VOTER

La FSA_IN_2 è una automa, che descrive come il microcontrollore ME, interagisce con gli altri due OTHER_A e OTHER_B in funzione delle transizioni del bit d'ingresso scritto InfoSharing0: InfoSharing.
Si considera quindi l'ingresso della FSA_IN_2, libero da bouncing.

La FSA, ma la macchina è in grado di gestire evventuali fault:
1

FSA_2 è un automa mirato all'acquisizione di un unico segnale elettromeccanico disponibile in tre copie provenienti da altrettanti trasduttori distinti, precedentemente ripuliti da rimbalzi. L'automa valuta affidabilmente il valore del segnale anche nel caso che al massimo uno dei trasduttori (o catene di acquisizione) soffra di un qualsivoglia guasto.
La FSA_IN_2 è in grado di segnalare eventuali errori.

50.7.1. dummy1
Signature: -dummy1 : t_ID

50.7.2. dummy2
Signature: -dummy2 : Type_fault

50.7.3. Fault0
-Fault0 : Fault

50.7.4. Timer0
-Timer0 : Timer

50.7.5. InfoSharing0
-InfoSharing0 : InfoSharing

50.7.6. Reliable_output
-Reliable_output : PP_without_Fault

50.7.7. start_T_JITTER
-start_T_JITTER : int

50.7.8. cnt_trans_ME
-cnt_trans_ME : int

50.7.9. cnt_trans_A
-cnt_trans_A : int
Variable to count the number of transactions sent by the INDEX_BIT input bits of OTHER_A.

50.7.10. cnt_trans_B
-cnt_trans_B : int
Variable to count the number of transactions sent by the INDEX_BIT input bits of OTHER_B.

50.7.11. Reporter0
-Reporter0 : Reporter

50.7.12. exec
Signature: exec() : void, Code Body: exec_VME ();
exec_VA ();
exec_VB ();
exec_VOTER ();

50.7.13. exec_VOTER
Signature: exec_VOTER() : void, Code Body: Type_state_VOTER_0 state_VOTER_0 = Reporter0.getState_VOTER_0 ();
Type_state_VOTER_0 new_state_VOTER_0 = state_VOTER_0;

switch (state_VOTER_0)

```
{
	// TRANSITIONS
	case OUT_0:
		if ( vote () )
		{
			new_state_VOTER_0 = VOTING;
		}
		break;
	case OUT_1:
		if ( vote () == false)
		{
			new_state_VOTER_0 = VOTING;
		}
		break;

	case VOTING:
		if (overcome_T_JITTER () && vote ())
		{
			new_state_VOTER_0 = OUT_1;
		}
		if(overcome_T_JITTER () && (vote () == false))
		{
			new_state_VOTER_0 = OUT_0;
		}
		break;
}

if ( new_state_VOTER_0 != state_VOTER_0)
{
	switch (state_VOTER_0)
	{
		case OUT_0:
		case OUT_1:
			start_T_JITTER = Timer0.read();
			break;
		case VOTING:
			if ( overcome_T_JITTER ())
			{
				write(vote());
				detect_fault();

			}
			break;
	}
	if ((new_state_VOTER_0 == OUT_0) || (new_state_VOTER_0 == OUT_1))
	{
		// RESET counters
		cnt_trans_ME = 0;
		cnt_trans_A = 0;
		cnt_trans_B = 0;
	}
}
```

```
Reporter0.setState_VOTER_0 ( new_state_VOTER_0);
```

50.7.14. exec_VME

Signature: exec_VME() : void, Code Body:

```
if (Reporter0.getState_VME () == State_ME_IN0)
{
	if (MYself () == true)
	{
		Reporter0.setState_VME (State_ME_IN1);
		cnt_trans_ME ++;
	}
}
else
{
	if (MYself () == false)
	{
		Reporter0.setState_VME (State_ME_IN0);
		cnt_trans_ME ++;
	}
}
```

50.7.15. exec_VA

Signature: exec_VA() : void, Code Body:

```
if (Reporter0.getState_VA () == State_OTHER_A_IN0)
{
	if (AA () == true)
	{
		Reporter0.setState_VA (State_OTHER_A_IN1);
		cnt_trans_A ++;
	}
}
else
{
	if (AA () == false)
	{
		Reporter0.setState_VA (State_OTHER_A_IN0);
		cnt_trans_A ++;
	}
}
```

50.7.16. exec_VB

Signature: exec_VB() : void, Code Body:

```
if (Reporter0.getState_VB () == State_OTHER_B_IN0)
{
	if (BB () == true)
	{
		Reporter0.setState_VB (State_OTHER_B_IN1);
		cnt_trans_B ++;
	}
}
else
{
	if (BB () == false)
	{
		Reporter0.setState_VB (State_OTHER_B_IN0);
```

```
                cnt_trans_B ++;
        }
}
```

50.7.17. init

Signature: init() : void, Code Body: init_VME ();
init_VA ();
init_VB ();
init_VOTER_0 ();

50.7.18. init_VOTER_0

Signature: init_VOTER_0() : void, Code Body: if (vote () == false)

```
{
        Reporter0.setState_VOTER_0 (OUT_0);
}
else
{
        Reporter0.setState_VOTER_0 (OUT_1);
}
cnt_trans_ME = 0;
cnt_trans_A = 0;
cnt_trans_B = 0;
```

50.7.19. init_VME

Signature: init_VME() : void, Code Body: if(MYself() == false)

```
{
        Reporter0.setState_VME(State_ME_IN0);
}
else
{
        Reporter0.setState_VME(State_ME_IN1);
}
cnt_trans_ME = 0;
```

50.7.20. init_VA

Signature: init_VA() : void, Code Body: if(AA () == false)

```
{
        Reporter0.setState_VA (State_OTHER_A_IN0);
}
else
{
        Reporter0.setState_VA (State_OTHER_A_IN1);
}
cnt_trans_A = 0;
```

50.7.21. init_VB

Signature: init_VB() : void, Code Body: if(BB() == false)

```
{
        Reporter0.setState_VB(State_OTHER_B_IN0);
}
else
{
        Reporter0.setState_VB(State_OTHER_B_IN1);
```

```
}
cnt_trans_B = 0;
```

50.7.22. write

Signature: write(value : bool) : void, Code Body: Reliable_output.writeBit (INDEX_BIT, value);

50.7.23. detect_fault

Signature: detect_fault() : bool
individua l'errore, esaminando se sono tutti d'accordo
restituisce 1 se è stato rilevato un fault, e 0 in caso contrario
Code Body: write (vote());

```
YYY::t_ID accused, guilty;
YYY::Type_fault name_fault;
bool value_guilty;
int counter;

accused = disagree_A_B_ME();
switch(accused)       // If there is'n an opposed
{
        case NOBODY:
                if (cnt_trans_ME != 1)
                {
                        counter = cnt_trans_ME;
                        value_guilty = MYself();
                        guilty = ME;
                }
                else if (cnt_trans_A != 1)
                {
                        counter = cnt_trans_A;
                        value_guilty = AA();
                        guilty = OTHER_A;
                }
                else if (cnt_trans_B != 1)
                {
                        counter = cnt_trans_B;
                        value_guilty = BB();
                        guilty = OTHER_B;
                }
                else
                        return false;
                break;
        case ME:
                guilty = ME;
                counter = cnt_trans_ME;
                value_guilty = MYself();
                break;
        case OTHER_A:
                guilty = OTHER_A;
                counter = cnt_trans_A;
                value_guilty = AA();
                break;
        case OTHER_B:
```

```
                guilty = OTHER_B;
                counter = cnt_trans_B;
                value_guilty = BB();
                break;
}

// Identifie the type of fault

if(counter == 0)            // there is a STUCK AT
{
        if(value_guilty == true)
        {
                name_fault = STUCK_AT_1;
        }
        if(value_guilty == false)
        {
                name_fault = STUCK_AT_0;
        }
}
if ((counter > 1) && (counter < MAX_TRANS))
{
        name_fault = GLITCH;
}
else if (counter >=  MAX_TRANS)
{
        name_fault = CRAZY_BOUNCING;
}

Fault0.send_VOTER (guilty, name_fault, value_guilty, counter, vote (), MYself (), AA (), BB (),
cnt_trans_ME, cnt_trans_A, cnt_trans_B );
return true;
```

50.7.24. disagree_A_B_ME

Signature: disagree_A_B_ME() : t_ID

Funzione che esamina i valori di A B e ME e restituisce NOBODY se sono tutti d'accordo, se invece c'è un disaccordo, restituisce il t_ID di chi è in disaccordo.

Code Body: return InfoSharing0.disagreeBit(INDEX_WORD, 1<<INDEX_BIT);

50.7.25. VOTER

Signature: VOTER()

50.7.26. VOTER

Signature: VOTER(dummy : int)

50.7.27. overcome_T_JITTER

Signature: overcome_T_JITTER() : bool, Code Body: return Timer0.cmp(start_T_JITTER, T_MAX_JITTER);

50.7.28. MYself

Signature: MYself() : bool

It returns the value of the input bit position INDEX_BIT of the (model element not found) microcontroller, which is carrying out the data processing.

Code Body: return InfoSharing0.readBit_ME(INDEX_WORD, 1<<INDEX_BIT);

50.7.29. AA

Signature: AA() : bool

It returns the value of the input bit position INDEX_BIT of the OTHER_A microcontroller, which was read by InfoSharing0: InfoSharing.

Code Body: return InfoSharing0.readBit_A(INDEX_WORD, 1<<INDEX_BIT);

50.7.30. BB

Signature: BB() : bool

It returns the value of the input bit position INDEX_BIT of the OTHER_B microcontroller, which was read by InfoSharing0: InfoSharing.

Code Body: return InfoSharing0.readBit_B(INDEX_WORD, 1<<INDEX_BIT);

50.7.31. vote

Signature: vote() : bool

It returns the voting bits value of the three microcontroller on the bit input SH_IN0 to INDEX_BIT position.

Code Body: bool tmp;
InfoSharing0.voteBit(INDEX_WORD, 1<<INDEX_BIT, tmp);
return tmp;

51. InfoSharing

51.1. UARTB

51.1.1. msg

Signature: -msg : char[32]

51.1.2. TX

Signature: TX(val : byte)

51.1.3. RX

Signature: RX(val : byte)

51.1.4. init

Signature: init() : void

51.1.5. send

Signature: send(msg : byte) : bool

51.1.6. read

Signature: read(msg : byte) : bool

51.1.7. UARTB

Signature: UARTB()

51.1.8. UARTB

Signature: UARTB(dummy : int)

51.2. UARTA

51.2.1. msg

Signature: -msg : char[32]

51.2.2. TX
Signature: TX(val : byte)

51.2.3. RX
Signature: RX(val : byte)

51.2.4. init
Signature: init() : void

51.2.5. send
Signature: send(msg : byte) : bool

51.2.6. read
Signature: read(msg : byte) : bool

51.2.7. UARTA
Signature: UARTA()

51.2.8. UARTA
Signature: UARTA(dummy : int)

51.3. InfoSharing

This is a communication infrastructure to help sharing the info among processors in a triple-redundant set of processors.

The systems is supposed to the made of three processors, called respectively:

- ME, that is, the processor on which the actual instance of the InfoSharing is running;
- OTHER_A; that is, the processor connected to ME via the (model element not found) peripheral;
- OTHER_B; that is, the processor connected to ME via the UB: UARTB peripheral;

This class stores three replicas of a set of critical data (SIZE bytes) into attribute SharedData, built using a TripleData structure.

The three replicas of each byte built inside each element of SharedData are as follows:

1. a(): DataType& stores the data of ME
2. b(): DataType& stores the virtually identical data periodically received from OTHER_A
3. c(): DataType& stores the virtually identical data periodically received from OTHER_B

This class has operations to:

1. write data of ME into shared storege, by means of writeByte(index: T_INDEX, val: byte): void;
2. read data from either of three processors Computing Unit;
3. vote data to return the most likely value of each bit (respectively, byte) by means of voteBit(index: T_INDEX, mask_bit: int, val: bool&): bool (respectively, voteByte(index: T_INDEX, val: byte&): byte);
4. send ME's data to both OTHER_A and OTHER_B, by means of send(): void.

51.3.1. SharedData
-SharedData[SIZE]

51.3.2. UA
Signature: -UA : UARTA

51.3.3. UB
Signature: -UB : UARTB

51.3.4. msg

Signature: -msg : byte[SIZE]

51.3.5. valBit

Signature: -valBit : bool

51.3.6. valByte

Signature: -valByte : byte

51.3.7. writeByte

Signature: writeByte(index : T_INDEX, val : byte) : void
Writes the value val into the index-th location of ME's replica in SharedData
Code Body: SharedData[to_int(index)].a() = val;

51.3.8. writeBit

Signature: writeBit(index : T_INDEX, mask_bit : byte, val : bool) : void
Writes the value val into the index-th location of ME's replica in SharedData
Code Body: if (val)
SharedData[to_int(index)].a() |= mask_bit;
else
SharedData[to_int(index)].a() &= ~ mask_bit;

51.3.9. voteBit

Signature: voteBit(index : T_INDEX, mask_bit : int, val : bool) : bool
Votes the three local copies of ME, OTHER_A, OTHER_B in bit mask_bit of location index.
Returns the most likely value (two out of three) into val.
Returns true if all three copies match; false otherwise.

51.3.10. voteByte

Signature: voteByte(index : T_INDEX, val : byte) : byte

51.3.11. send

Signature: send() : void
Sends ME's replica of SharedData to OTHER_A (via (model element not found)) and to OTHER_B (via UB: UARTB)

51.3.12. isr_UB

Signature: isr_UB() : void

51.3.13. readBit_ME

Signature: readBit_ME(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].a() & mask_bit);

51.3.14. readBit_A

Signature: readBit_A(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].b() & mask_bit);

51.3.15. readBit_B

Signature: readBit_B(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].c() & mask_bit);

51.3.16. to_int

Signature: to_int(value : T_INDEX) : int, Code Body: /*
switch (value)
{

```
        case SH_IN:       return  0;
        case SH_OUT:      return  1;
        case SH_AIN0:     return  2;
        case SH_AIN1:     return  3;
        case SH_AIN2:     return  4;
        case SH_AIN3:     return  5;
        case SH_AOUT0: return  6;
        case SH_AOUT1: return  7;
        case GRANT:       return  8;
        default :         return   0;
}
*/
return (int) value;
```

51.3.17. disagreeBit

Signature: disagreeBit(index : T_INDEX, mask_bit : byte) : t_ID, Code Body: return to_t_ID(SharedData[to_int(index)].disagreeBit (mask_bit));

51.3.18. to_t_ID

Signature: to_t_ID(value) : t_ID, Code Body: switch (value)

```
{
     case HardenedData::NONE:        return NOBODY;
     case HardenedData::AA:          return ME;
     case HardenedData::BB:return OTHER_A;
     case HardenedData::CC:return OTHER_B;
}
```

51.3.19. InfoSharing

Signature: InfoSharing()

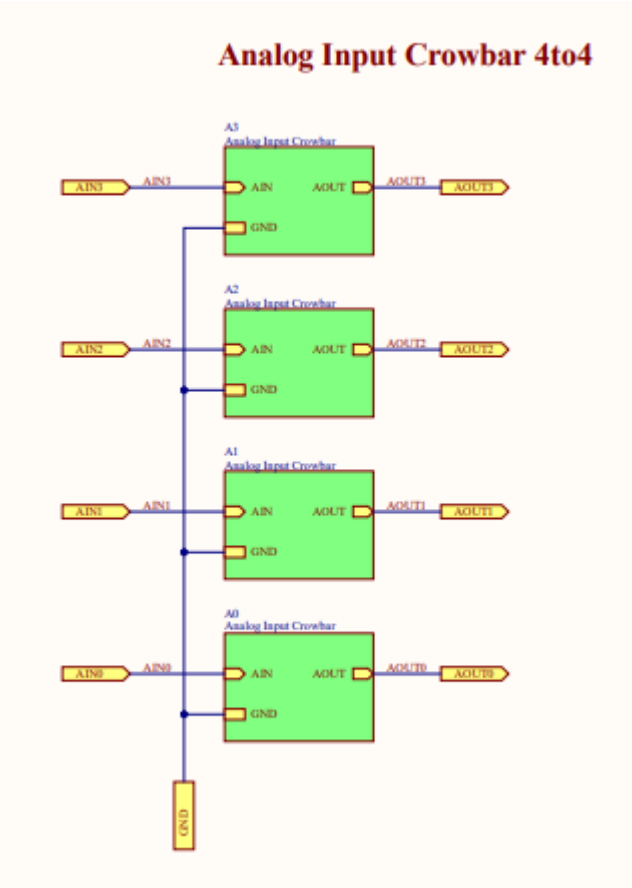
51.3.20. InfoSharing

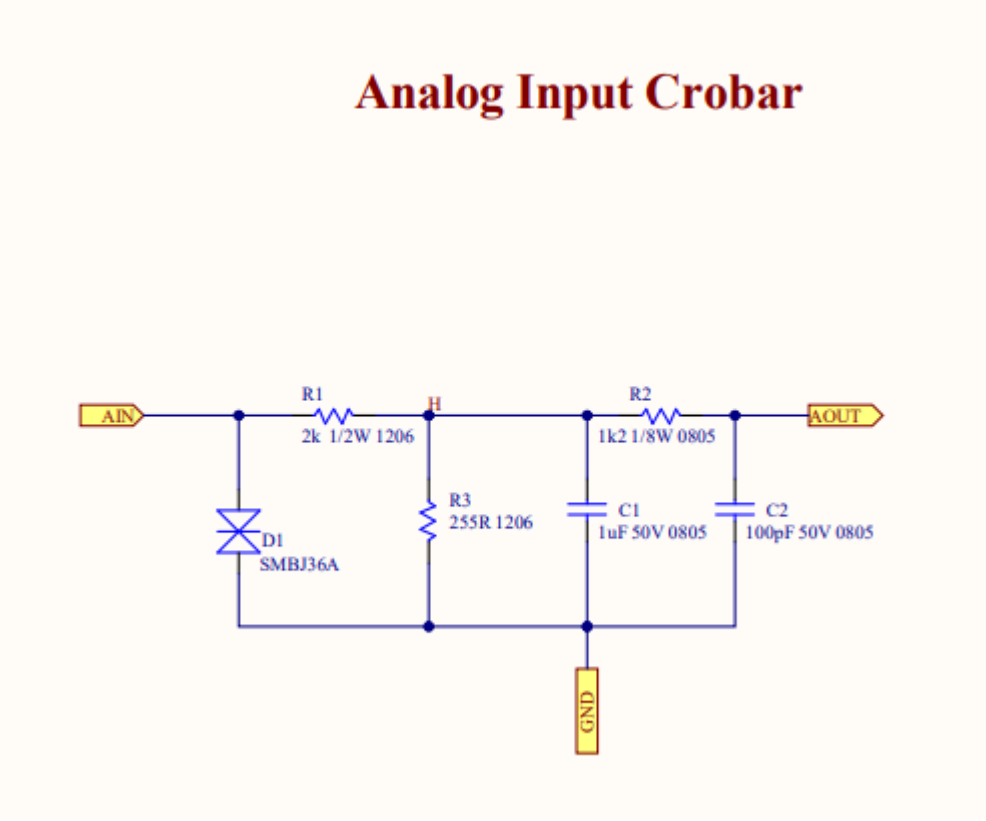
Signature: InfoSharing(dummy : int)

52. Descrizione del Computing Unit

52.1. Schema Input Crowbar

Di seguito lo schema del modulo input Crowbar utilizzato per proteggere il segnale dalle sovratensioni.


Analog Input Crowbar 4to4
A3
Analog Input Crowbar
AIN3
AIN3
AIN
AOUT
AOUT3
AOUT3
GND
A2
Analog Input Crowbar
AIN2
AIN2
AIN
AOUT
AOUT2
AOUT2
GND
A1
Analog Input Crowbar
AIN1
AIN1
AIN
AOUT
AOUT1
AOUT1
GND
A0
Analog Input Crowbar
AIN0
AIN0
AIN
AOUT
AOUT0
AOUT0
GND
GND


Analog Input Crobar
AIN
R1
2k 1/2W 1206
H
R2
1k2 1/8W 0805
AOUT
D1
SMBJ36A
R3
255R 1206
C1
1uF 50V 0805
C2
100pF 50V 0805
GND

52.2. Input Crowbar

Classe relativa all'Input Crowbar. Questo modulo elementare riceve in ingresso un segnale digitale. Il modulo protegge gli ingressi del STM32F072 a cui è collegato dalle sovratemsioni e dalle sovracorrenti.

52.2.1. R1

Signature: -R1 : 2K7 RES 0603

52.2.2. R2

Signature: -R2 : RES 0603

52.2.3. DS1

Signature: -DS1 : CUS520 Schottky

52.2.4. DZ1

Signature: -DZ1 : DDZ3V9BSF Zener

52.2.5. C1

Signature: -C1 : Capacitor_0603

52.2.6. Vdz1

Signature: -Vdz1
Voltage drop between DZ1: DDZ3V9BSF Zener Anode and Cathode.

52.2.7. Idz1

Signature: -Idz1
Current flowing in DZ1: DDZ3V9BSF Zener.

52.2.8. Vds1

Signature: -Vds1
Voltage drop between DS1: CUS520 Schottky DZ1: DDZ3V9BSF Zener Anode and Cathode.

52.2.9. Ids1

Signature: -Ids1
Current flowing in DS1: CUS520 Schottky.

52.2.10. Va_z

Signature: -Va_z
Voltage drop in R1: 2K7 RES 0603.

52.2.11. Vz_b

Signature: -Vz_b
Voltage drop in R2: RES 0603.

52.2.12. AAA

Signature: AAA(in : t_TTL_IN)

52.2.13. BBB

Signature: BBB(out : t_TTL3V3)

52.2.14. GND

Signature: GND()

52.3. IPT-105-BOTTOM

Classe del connettore di interfaccia della tensione di alimentazione da VCC da 3.3V nominali con cui la scheda piggy back riceve l'alimentazione dalla scheda madre.

52.4. CONN MALE 3 POS

Classe relativa al connettore da tre posizioni a cui connettere l'alimentazione esterna quando vi è necessità di utilizzare il Computing Unit senza che sia montato sul Dimostratore Monopiattaforma. In questo caso l'alimentazione da connettere è compresa tra i 12Vdc e i 36Vdc.

52.5. DB9 F

Classe relativa a uno dei due connettori DB9 utilizzato per la comunicazione tramite rete CAN di due Computing Unit quando non sono connessi alla scheda madre.

52.6. DB9 M

Classe relativa a uno dei due connettori DB9 utilizzato per la comunicazione tramite rete CAN di due Computing Unit quando non sono connessi alla scheda madre.

52.7. IPT-110-BOTTOM

Classe relativa al connettore di interfaccia sulla scheda madre attraverso il quale vengono propagati i segnali relativi alle due interfacce seriali USART A e USART B e ai due segnali della rete can.

52.8. IPT1-115- BOTTOM

Classe relativa la connettore a cui sono collegati gli 8 DIN, gli 8 DOUT e i 4 segnali AIN.
Questo è il connettore di interfaccia dati sulla scheda madre.

52.9. CAN Interface

Classe relativa all'interfaccia della rete CAN.

52.10. NC_dip_switch

Classe relativa al componente DIP switch CT2192MST-ND utilizzato per effettuare il reset del circuito e per abilitare il uC group.

Ciò avviene tramite attivazione dei due interruttori DIP relativi tramite azione esterna dell'utente.

52.11. Computing Unit

Classe relativa al componente Computing Unit, che contiene il modulo uC.
Contiene tutte le funzioni SW che consentono il funzionamento del modulo.
La classe possiede i seguenti attributi HW di seguito elencate:

1. Input Crobar
2. Analog Input Crowbar

3. CAN INterface
4. OUtput CRowbar
5. Debug Interface
6. uC group
7. SWD Interface

Maggiori dettagli sono presenti nelle classi relative

52.11.1. data_input

Signature: -data_input : byte
Variabile che contiene il valore degli ingressi digitali

52.11.2. Data Output

Signature: -Data Output : byte

52.11.3. Input_Crwb0

Signature: -Input_Crwb0 : Input Crowbar
Istanza relativa alla classe Input Crowbar, dedicata alla protezione dei Digital Input dalle sovratensioni.

52.11.4. CAN Interface0

Signature: -CAN Interface0 : CAN Interface

52.11.5. Outup Crowbar0

Signature: -Outup Crowbar0 : Output Crowbar

52.11.6. DEBUG

Signature: -DEBUG : DEBUG Interface

52.11.7. Analog Iput Crowbar 0

Signature: -Analog Iput Crowbar 0 : Analog Input Crowbar

52.11.8. SW Interface 0

Signature: -SW Interface 0 : SWD Interface

52.11.9. Dip_Swtch0

Signature: -Dip_Swtch0 : NC_dip_switch

52.11.10. CPU

Signature: -CPU : uC groupe

52.11.11. J1

Signature: -J1 : IPT1-115- BOTTOM

52.11.12. J2

Signature: -J2 : IPT-110-BOTTOM

52.11.13. CAM M

Signature: -CAM M : DB9 M

52.11.14. CAN F

Signature: -CAN F : DB9 F

52.11.15. CN_PW_EXT

Signature: -CN_PW_EXT : CONN MALE 3 POS

52.11.16. CN_PW

Signature: -CN_PW : IPT-105-BOTTOM

52.11.17. DATA compare

Signature: -DATA compare : InfoSharing

52.11.18. AIN0

Signature: AIN0()
Segnale di Analog Input del Computing Unit.

52.11.19. AIN1

Signature: AIN1()
Segnale di Analog Input del Computing Unit.

52.11.20. AIN2

Signature: AIN2()
Segnale di Analog Input del Computing Unit.

52.11.21. AIN3

Signature: AIN3()
Segnale di Analog Input del Computing Unit.

52.11.22. VAL-EXT

Signature: VAL-EXT()
Pin positivo connesso alla alimentazione esterna del Computing Unit.Il pin va collegato a un tensione continua (tra il pin e il GND della scheda) compresa ta i 12Vdc e i 36 VDC.

52.11.23. VAL

Signature: VAL()
Pin positivo connesso alla alimentazione VCC 3.3V del Computing Unit. Consente di alimentare la scheda quando è inseita sulla scheda madre e l'alimentatore è abilitato tramite switch.

52.11.24. GND

Signature: GND()
Pin di riferimento GND.

52.11.25. FAULT

Signature: FAULT()
Segnale digital Output attivo alto in presenza di un fault

52.11.26. DOUT

Signature: DOUT()
Digital signals maching Computing Unit's pins DOUT0, DOUT1, DOUT2, DOUT3, DOUT4, DOUT5, DOUT6 and DOUT7.

52.11.27. DIN

Signature: DIN()
Digital signals input maching Computing Unit's pins DIN0, DIN1, DIN2, DIN3, DIN4, DIN6, DIN5 e DIN7.

52.11.28. DEBUG

Signature: DEBUG()
Insieme dei segnali utilizzati per il download del SW all'interno del Computing Unit. vedi maggioni dettagli nel diagramma dei componenti relativo.

52.11.29. CAN

Signature: CAN(data)
Insieme di segnali relativi all'interfaccia CAN utilizzata dal Computing Unit.

52.12. DEBOUNCE

52.12.1. Timer0

-Timer0 : Timer

52.12.2. start_T_Bounce_T_Pulse

-start_T_Bounce_T_Pulse : int
Start time of bouncing signal, and time of pulse

52.12.3. pp0

-pp0 : ParallelPort

52.12.4. InfoSharing0

-InfoSharing0 : InfoSharing

52.12.5. Fault0

Fault0 : Fault

52.12.6. n_trans_IN

-n_trans_IN : int
valore delle transizioni effettuate dall'ingresso tra una scrittura su infoSharing e l'altra

52.12.7. Reporter0

-Reporter0 : Reporter

52.12.8. exec

Signature: exec() : void, Code Body: // Transitions state machine

```
Type_State_DEBOUNCE state_DEBOUNCE = Reporter0.getState_DEBOUNCE ();
Type_State_DEBOUNCE new_state_DEBOUNCE = state_DEBOUNCE;
switch(state_DEBOUNCE)
{
	case IN_0:
		if (is1 ())
		{
			new_state_DEBOUNCE = IN_1_from_0;
		}
		break;

	case IN_1_from_0 :
		if (is0 () && (overcome_T_MAX_BOUNCE () == false))
		{
			new_state_DEBOUNCE = IN_0_from_1_from_0;
		}
		if (overcome_T_MAX_BOUNCE())
		{
			new_state_DEBOUNCE = IN_1;
		}
		break;

	case IN_0_from_1_from_0:
```

```
                if (is1 () && overcome_T_MAX_BOUNCE () == false)
                {
                        new_state_DEBOUNCE = IN_1_from_0;
                }
                if (overcome_T_MAX_BOUNCE ())
                {
                        new_state_DEBOUNCE = IN_0;
                }
                break;

        case IN_1:
                if (is0 ())
                {
                        new_state_DEBOUNCE = IN_0_from_1;
                }
                break;

        case IN_0_from_1 :
                if (is1 () && (overcome_T_MAX_BOUNCE() == false))
                {
                        new_state_DEBOUNCE = IN_1_from_0_from_1;
                }
                if (overcome_T_MAX_BOUNCE())
                {
                        new_state_DEBOUNCE = IN_0;
                }
                break;


        case IN_1_from_0_from_1:
                if (is0 () && (overcome_T_MAX_BOUNCE () == false))
                {
                        new_state_DEBOUNCE = IN_0_from_1;
                }
                if (overcome_T_MAX_BOUNCE ())
                {
                        new_state_DEBOUNCE = IN_1;
                }
                break;
}
// effect EXIT state


if (new_state_DEBOUNCE != state_DEBOUNCE)
{
        switch(state_DEBOUNCE)
        {

                case IN_0:
                        //EXIT
                        start_T_Bounce_T_Pulse = Timer0.read();
```

```
                // TRANSITIONS
                if (is1 ())
                {
                        n_trans_IN ++;
                        if (overcome_T_MIN_PULSE() == false)
                        {
                                Fault0.send_DEBOUNCE (SHORT_PULSE_0, n_trans_IN);
                                start_T_Bounce_T_Pulse = Timer0.read();
                        }
                }
                break;

        case IN_1_from_0:
                // EXIT
                // None effect

                // TRANSITIONS
                if (is0() && (overcome_T_MAX_BOUNCE () == false))
                {
                        n_trans_IN ++;
                }
                // if (overcome_T_MAX_BOUNCE() ) causes just a change of state
                break;

        case IN_0_from_1_from_0:
                // EXIT
                // None effect

                // TRANSITIONS
                if (is1 () && (overcome_T_MAX_BOUNCE  () == false))
                {
                        n_trans_IN ++;
                }
                if (overcome_T_MAX_BOUNCE())
                {
                        Fault0.send_DEBOUNCE (NOISE, n_trans_IN);
                }
                break;

        case IN_1:
                //EXIT
                start_T_Bounce_T_Pulse = Timer0.read();

                // TRANSITIONS
                if (is0 ())
                {
                        n_trans_IN ++;
                        if (overcome_T_MIN_PULSE() == false)
                        {
                                Fault0.send_DEBOUNCE (SHORT_PULSE_1, n_trans_IN);
                                start_T_Bounce_T_Pulse = Timer0.read();
                        }
```

```
                }
                break;

        case IN_0_from_1:
                // EXIT
                // None effect

                // TRANSITIONS
                if (is1() && (overcome_T_MAX_BOUNCE () == false))
                {
                        n_trans_IN ++;
                }
                // if (is0 () && overcome_T_MAX_BOUNCE() ) causes just a change of state
                break;
        case IN_1_from_0_from_1:
                // EXIT
                // None effect

                // TRANSITIONS
                if (is0 () && (overcome_T_MAX_BOUNCE () == false))
                {
                        n_trans_IN ++;
                }
                if (is1 () && overcome_T_MAX_BOUNCE())
                {
                        Fault0.send_DEBOUNCE(NOISE, n_trans_IN);
                }
                break;


    }

    // effect ENTRY in state

    switch (new_state_DEBOUNCE)
    {
        case IN_0:
                write (false);
                n_trans_IN = 0;
                break;
        case IN_1:
                write (true);
                n_trans_IN = 0;
                break;
        default:
                break;
    }
}

Reporter0.setState_DEBOUNCE (new_state_DEBOUNCE);
```

52.12.9. init

Signature: init() : void, Code Body: Reporter0.setState_DEBOUNCE (IN_0); // initial state

52.12.10. is0

Signature: is0() : bool, Code Body: return (pp0.read(1<<INDEX_BIT) == false);

52.12.11. is1

Signature: is1() : bool, Code Body: return (pp0.read(1<<INDEX_BIT) == true);

52.12.12. overcome_T_MIN_PULSE

Signature: overcome_T_MIN_PULSE() : bool, Code Body: return Timer0.cmp(start_T_Bounce_T_Pulse, T_MIN_PULSE);

52.12.13. overcome_T_MAX_BOUNCE

Signature: overcome_T_MAX_BOUNCE() : bool, Code Body: return Timer0.cmp(start_T_Bounce_T_Pulse, T_MAX_BOUNCE);

52.12.14. write

Signature: write(value : bool) : void, Code Body: InfoSharing0.writeBit(INDEX_WORD, (1<<INDEX_BIT), value);

52.12.15. DEBOUNCE

Signature: DEBOUNCE()

52.12.16. DEBOUNCE

Signature: DEBOUNCE(dummy : int)

52.13. Timer

Timer class contains all information related to the Computing Unit timer functions.

52.13.1. time

-time : int

52.13.2. init

Signature: init() : void
Configures and starts the Timer.
Code Body: time = 0;

52.13.3. cmp

Signature: cmp(t_inizial : int, t_max : int) : bool
Compares the difference between the current value of time provided by the Timer and the t_inizial with the t_max, considering also overflow. It returning 1 if greater than or equal, 0 if different.

52.13.4. read

Signature: read() : int
It provides the actual time measured by the Timer
Code Body: return time;

52.13.5. inc

Signature: inc() : void, Code Body: time++;

52.13.6. Timer

Signature: Timer()

52.13.7. Timer

Signature: Timer(dummy : int)

52.14. ADC

52.14.1. ADC_IN15
Signature: ADC_IN15()

52.14.2. ADC_IN14
Signature: ADC_IN14()

52.14.3. ADC_IN13
Signature: ADC_IN13()

52.14.4. ADC_IN12
Signature: ADC_IN12()

52.14.5. ADC_IN11
Signature: ADC_IN11()

52.14.6. ADC_IN10
Signature: ADC_IN10()

52.14.7. ADC_IN9
Signature: ADC_IN9()

52.14.8. ADC_IN8
Signature: ADC_IN8()

52.14.9. ADC_IN7
Signature: ADC_IN7()

52.14.10. ADC_IN6
Signature: ADC_IN6()

52.14.11. ADC_IN5
Signature: ADC_IN5()

52.14.12. ADC_IN4
Signature: ADC_IN4()

52.14.13. ADC_IN3
Signature: ADC_IN3()

52.14.14. ADC_IN2
Signature: ADC_IN2()

52.14.15. ADC_IN1
Signature: ADC_IN1()

52.14.16. ADC_IN0
Signature: ADC_IN0()

52.15. FSA_DATE

52.15.1. cnt
Signature: -cnt : int
Variable to counter that count the numbr of times that repeats a loop.

52.15.2. fault
Signature: -fault : bool = false

52.15.3. exec
Signature: exec()

52.16. Fault

52.16.1. send_DEBOUNCE
Signature: send_DEBOUNCE(what : Type_fault, n_trans : int) : void
Sends a message that contains the Type_fault detected.

Devo mandare anche l'identificativo MYSELF quindi non serve che glielo passi come parametro
Code Body: // ricorda di mandare il my_name

52.16.2. send_VOTER
Signature: send_VOTER(guilty : t_ID, what : Type_fault, value_guilty : bool, n_trans : int, value_out : bool, value_ME : bool, value_A : bool, value_B : bool, n_trans_ME : int, n_trans_A : int, n_trans_B : int) : void
Sends a message that contains the Type_fault detected.
Code Body: // ricorda di mandare il my_name

52.16.3. Fault
Signature: Fault()

52.16.4. Fault
Signature: Fault(dummy : int)

52.17. VOTER

La FSA_IN_2 è una automa, che descrive come il microcontrollore ME, interagisce con gli altri due OTHER_A e OTHER_B in funzione delle transizioni del bit d'ingresso scritto InfoSharing0: InfoSharing.
Si considera quindi l'ingresso della FSA_IN_2, libero da bouncing.
La FSA, ma la macchina è in grado di gestire evventuali fault:
1

FSA_2 è un automa mirato all'acquisizione di un unico segnale elettromeccanico disponibile in tre copie provenienti da altrettanti trasduttori distinti, precedentemente ripuliti da rimbalzi. L'automa valuta affidabilmente il valore del segnale anche nel caso che al massimo uno dei trasduttori (o catene di acquisizione) soffra di un qualsivoglia guasto.
La FSA_IN_2 è in grado di segnalare eventuali errori.

52.17.1. dummy1
Signature: -dummy1 : t_ID

52.17.2. dummy2
Signature: -dummy2 : Type_fault

52.17.3. Fault0
-Fault0 : Fault

52.17.4. Timer0
-Timer0 : Timer

52.17.5. InfoSharing0

-InfoSharing0 : InfoSharing

52.17.6. Reliable_output

-Reliable_output : PP_without_Fault

52.17.7. start_T_JITTER

-start_T_JITTER : int

52.17.8. cnt_trans_ME

-cnt_trans_ME : int

52.17.9. cnt_trans_A

-cnt_trans_A : int

Variable to count the number of transactions sent by the INDEX_BIT input bits of OTHER_A.

52.17.10. cnt_trans_B

-cnt_trans_B : int

Variable to count the number of transactions sent by the INDEX_BIT input bits of OTHER_B.

52.17.11. Reporter0

-Reporter0 : Reporter

52.17.12. exec

Signature: exec() : void, Code Body: exec_VME ();
exec_VA ();
exec_VB ();
exec_VOTER ();

52.17.13. exec_VOTER

Signature: exec_VOTER() : void, Code Body: Type_state_VOTER_0 state_VOTER_0 = Reporter0.getState_VOTER_0 ();
Type_state_VOTER_0 new_state_VOTER_0 = state_VOTER_0;

```
switch (state_VOTER_0)
{
        // TRANSITIONS
        case OUT_0:
                if ( vote () )
                {
                        new_state_VOTER_0 = VOTING;
                }
                break;
        case OUT_1:
                if ( vote () == false)
                {
                        new_state_VOTER_0 = VOTING;
                }
                break;

        case VOTING:
                if (overcome_T_JITTER () && vote ())
                {
                        new_state_VOTER_0 = OUT_1;
                }
```

```
                if(overcome_T_JITTER () && (vote () == false))
                {
                        new_state_VOTER_0 = OUT_0;
                }
                break;
}

if ( new_state_VOTER_0 != state_VOTER_0)
{
        switch (state_VOTER_0)
        {
                case OUT_0:
                case OUT_1:
                        start_T_JITTER = Timer0.read();
                        break;
                case VOTING:
                        if ( overcome_T_JITTER ())
                        {
                                write(vote());
                                detect_fault();

                        }
                        break;
        }
        if ((new_state_VOTER_0 == OUT_0) || (new_state_VOTER_0 == OUT_1))
        {
                // RESET counters
                cnt_trans_ME = 0;
                cnt_trans_A = 0;
                cnt_trans_B = 0;
        }
}
Reporter0.setState_VOTER_0 ( new_state_VOTER_0);
```

52.17.14. exec_VME

Signature: exec_VME() : void, Code Body: if (Reporter0.getState_VME () == State_ME_IN0)

```
{
        if (MYself () == true)
        {
                Reporter0.setState_VME (State_ME_IN1);
                cnt_trans_ME ++;
        }
}
else
{
        if (MYself () == false)
        {
                Reporter0.setState_VME (State_ME_IN0);
                cnt_trans_ME ++;
        }
}
```

52.17.15. exec_VA

Signature: exec_VA() : void, Code Body: if (Reporter0.getState_VA () == State_OTHER_A_IN0)

```
{
	if (AA () == true)
	{
		Reporter0.setState_VA (State_OTHER_A_IN1);
		cnt_trans_A ++;
	}
}
else
{
	if (AA () == false)
	{
		Reporter0.setState_VA (State_OTHER_A_IN0);
		cnt_trans_A ++;
	}
}
```

52.17.16. exec_VB

Signature: exec_VB() : void, Code Body: if (Reporter0.getState_VB () == State_OTHER_B_IN0)

```
{
	if (BB () == true)
	{
		Reporter0.setState_VB (State_OTHER_B_IN1);
		cnt_trans_B ++;
	}
}
else
{
	if (BB () == false)
	{
		Reporter0.setState_VB (State_OTHER_B_IN0);
		cnt_trans_B ++;
	}
}
```

52.17.17. init

Signature: init() : void, Code Body: init_VME ();

```
init_VA ();
init_VB ();
init_VOTER_0 ();
```

52.17.18. init_VOTER_0

Signature: init_VOTER_0() : void, Code Body: if (vote () == false)

```
{
	Reporter0.setState_VOTER_0 (OUT_0);
}
else
{
	Reporter0.setState_VOTER_0 (OUT_1);
}
cnt_trans_ME = 0;
```

```
cnt_trans_A = 0;
cnt_trans_B = 0;
```

52.17.19. init_VME

Signature: init_VME() : void, Code Body: if(MYself() == false)

```
{
        Reporter0.setState_VME(State_ME_IN0);
}
else
{
        Reporter0.setState_VME(State_ME_IN1);
}
cnt_trans_ME = 0;
```

52.17.20. init_VA

Signature: init_VA() : void, Code Body: if(AA () == false)

```
{
        Reporter0.setState_VA (State_OTHER_A_IN0);
}
else
{
        Reporter0.setState_VA (State_OTHER_A_IN1);
}
cnt_trans_A = 0;
```

52.17.21. init_VB

Signature: init_VB() : void, Code Body: if(BB() == false)

```
{
        Reporter0.setState_VB(State_OTHER_B_IN0);
}
else
{
        Reporter0.setState_VB(State_OTHER_B_IN1);
}
cnt_trans_B = 0;
```

52.17.22. write

Signature: write(value : bool) : void, Code Body: Reliable_output.writeBit (INDEX_BIT, value);

52.17.23. detect_fault

Signature: detect_fault() : bool
individua l'errore, esaminando se sono tutti d'accordo
restituisce 1 se è stato rilevato un fault, e 0 in caso contrario
Code Body: write (vote());

```
YYY::t_ID accused, guilty;
YYY::Type_fault name_fault;
bool value_guilty;
int counter;

accused = disagree_A_B_ME();
switch(accused)        // If there is'n an opposed
{
```

```
	case NOBODY:
		if (cnt_trans_ME != 1)
		{
			counter = cnt_trans_ME;
			value_guilty = MYself();
			guilty = ME;
		}
		else if (cnt_trans_A != 1)
		{
			counter = cnt_trans_A;
			value_guilty = AA();
			guilty = OTHER_A;
		}
		else if (cnt_trans_B != 1)
		{
			counter = cnt_trans_B;
			value_guilty = BB();
			guilty = OTHER_B;
		}
		else
			return false;
		break;
	case ME:
		guilty = ME;
		counter = cnt_trans_ME;
		value_guilty = MYself();
		break;
	case OTHER_A:
		guilty = OTHER_A;
		counter = cnt_trans_A;
		value_guilty = AA();
		break;
	case OTHER_B:
		guilty = OTHER_B;
		counter = cnt_trans_B;
		value_guilty = BB();
		break;
}

// Identifie the type of fault

if(counter == 0)		// there is a STUCK AT
{
	if(value_guilty == true)
	{
		name_fault = STUCK_AT_1;
	}
	if(value_guilty == false)
	{
		name_fault = STUCK_AT_0;
	}
}
```

```
if ((counter > 1) && (counter < MAX_TRANS))
{
        name_fault = GLITCH;
}
else if (counter >=  MAX_TRANS)
{
        name_fault = CRAZY_BOUNCING;
}

Fault0.send_VOTER (guilty, name_fault, value_guilty, counter, vote (), MYself (), AA (), BB (),
cnt_trans_ME, cnt_trans_A, cnt_trans_B );
return true;
```

52.17.24. disagree_A_B_ME

Signature: disagree_A_B_ME() : t_ID

Funzione che esamina i valori di A B e ME e restituisce NOBODY se sono tutti d'accordo, se invece c'è un disaccordo, restituisce il t_ID di chi è in disaccordo.

Code Body: return InfoSharing0.disagreeBit(INDEX_WORD, 1<<INDEX_BIT);

52.17.25. VOTER

Signature: VOTER()

52.17.26. VOTER

Signature: VOTER(dummy : int)

52.17.27. overcome_T_JITTER

Signature: overcome_T_JITTER() : bool, Code Body: return Timer0.cmp(start_T_JITTER, T_MAX_JITTER);

52.17.28. MYself

Signature: MYself() : bool

It returns the value of the input bit position INDEX_BIT of the (model element not found) microcontroller, which is carrying out the data processing.

Code Body: return InfoSharing0.readBit_ME(INDEX_WORD, 1<<INDEX_BIT);

52.17.29. AA

Signature: AA() : bool

It returns the value of the input bit position INDEX_BIT of the OTHER_A microcontroller, which was read by InfoSharing0: InfoSharing.

Code Body: return InfoSharing0.readBit_A(INDEX_WORD, 1<<INDEX_BIT);

52.17.30. BB

Signature: BB() : bool

It returns the value of the input bit position INDEX_BIT of the OTHER_B microcontroller, which was read by InfoSharing0: InfoSharing.

Code Body: return InfoSharing0.readBit_B(INDEX_WORD, 1<<INDEX_BIT);

52.17.31. vote

Signature: vote() : bool

It returns the voting bits value of the three microcontroller on the bit input SH_IN0 to INDEX_BIT position.

Code Body:

```
bool tmp;
InfoSharing0.voteBit(INDEX_WORD, 1<<INDEX_BIT, tmp);
return tmp;
```

52.18. t_TTL3V3

Parameters (tagged values) to define logic levels of 3.3V TTL signals.

52.19. InfoSharing

This is a communication infrastructure to help sharing the info among processors in a triple-redundant set of processors.

The systems is supposed to the made of three processors, called respectively:

- ME, that is, the processor on which the actual instance of the InfoSharing is running;
- OTHER_A; that is, the processor connected to ME via the (model element not found) peripheral;
- OTHER_B; that is, the processor connected to ME via the UB: UARTB peripheral;

This class stores three replicas of a set of critical data (SIZE bytes) into attribute SharedData, built using a TripleData structure.

The three replicas of each byte built inside each element of SharedData are as follows:

1. a(): DataType& stores the data of ME
2. b(): DataType& stores the virtually identical data periodically received from OTHER_A
3. c(): DataType& stores the virtually identical data periodically received from OTHER_B

This class has operations to:

1. write data of ME into shared storege, by means of writeByte(index: T_INDEX, val: byte): void;
2. read data from either of three processors Computing Unit;
3. vote data to return the most likely value of each bit (respectively, byte) by means of voteBit(index: T_INDEX, mask_bit: int, val: bool&): bool (respectively, voteByte(index: T_INDEX, val: byte&): byte);
4. send ME's data to both OTHER_A and OTHER_B, by means of send(): void.

52.19.1. SharedData

-SharedData[SIZE]

52.19.2. UA

Signature: -UA : UARTA

52.19.3. UB

Signature: -UB : UARTB

52.19.4. msg

Signature: -msg : byte[SIZE]

52.19.5. valBit

Signature: -valBit : bool

52.19.6. valByte

Signature: -valByte : byte

52.19.7. writeByte

Signature: writeByte(index : T_INDEX, val : byte) : void
Writes the value val into the index-th location of ME's replica in SharedData
Code Body: SharedData[to_int(index)].a() = val;

52.19.8. writeBit

Signature: writeBit(index : T_INDEX, mask_bit : byte, val : bool) : void

Writes the value val into the index-th location of ME's replica in SharedData
Code Body: if (val)
SharedData[to_int(index)].a() |= mask_bit;
else
SharedData[to_int(index)].a() &= ~ mask_bit;

52.19.9. voteBit

Signature: voteBit(index : T_INDEX, mask_bit : int, val : bool) : bool
Votes the three local copies of ME, OTHER_A, OTHER_B in bit mask_bit of location index.
Returns the most likely value (two out of three) into val.
Returns true if all three copies match; false otherwise.

52.19.10. voteByte

Signature: voteByte(index : T_INDEX, val : byte) : byte

52.19.11. send

Signature: send() : void
Sends ME's replica of SharedData to OTHER_A (via (model element not found)) and to OTHER_B (via UB: UARTB)

52.19.12. isr_UB

Signature: isr_UB() : void

52.19.13. readBit_ME

Signature: readBit_ME(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].a() & mask_bit);

52.19.14. readBit_A

Signature: readBit_A(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].b() & mask_bit);

52.19.15. readBit_B

Signature: readBit_B(index : T_INDEX, mask_bit : byte) : bool, Code Body: return (SharedData[to_int(index)].c() & mask_bit);

52.19.16. to_int

Signature: to_int(value : T_INDEX) : int, Code Body: /*

```
switch (value)
{
        case SH_IN:       return  0;
        case SH_OUT:      return  1;
        case SH_AIN0:     return  2;
        case SH_AIN1:     return  3;
        case SH_AIN2:     return  4;
        case SH_AIN3:     return  5;
        case SH_AOUT0: return  6;
        case SH_AOUT1: return  7;
        case GRANT:       return  8;
        default :         return   0;
}
*/
return (int) value;
```

52.19.17. disagreeBit

Signature: disagreeBit(index : T_INDEX, mask_bit : byte) : t_ID, Code Body: return to_t_ID(SharedData[to_int(index)].disagreeBit (mask_bit));

52.19.18. to_t_ID

Signature: to_t_ID(value) : t_ID, Code Body: switch (value)

```
{
    case HardenedData::NONE:        return NOBODY;
    case HardenedData::AA:          return ME;
    case HardenedData::BB:return OTHER_A;
    case HardenedData::CC:return OTHER_B;
}
```

52.19.19. InfoSharing

Signature: InfoSharing()

52.19.20. InfoSharing

Signature: InfoSharing(dummy : int)

52.20. PP_without_Fault

52.20.1. input_safe

-input_safe : byte

Variable contains a input byte free by faults.

52.20.2. writeBit

Signature: writeBit(mask_bit : byte, value : bool) : void

Write value on the bit value of the bit position variable input_safe

Code Body: if (value)

```
  input_safe |= mask_bit;
else
  input_safe &= ~mask_bit;
```

52.20.3. readBit

Signature: readBit(mask_bit : byte) : bool, Code Body: return (input_safe & mask_bit);

52.20.4. readByte

Signature: readByte() : byte, Code Body: return input_safe;

52.20.5. PP_without_Fault

Signature: PP_without_Fault()

52.20.6. PP_without_Fault

Signature: PP_without_Fault(dummy : int)

52.21. Voter

52.21.1. FAULT

Signature: FAULT(val : bool)

52.22. Type_fault

La classe enumerativa

52.23. InputCleaner8

La classe InputCleaner8 iceve in ingresso 8 inputs digitali on/off.
Libera singolarmente gli input dal bounce. e memorizza internamente il dato.
bouncing -> una serie di transizioni del segnale digitale da on a off, e viceversa per un tempo inferiore a T_MAX_BOUNCE.

Trasmette i valori debounciati di ciascun ingresso, tramite UART, a due inputcleaner8 gemelli , interni a due moduli esterni che ricevono copia virtualmente identica, degli stessi 8 input.

Riceve copia degli 8 input dai due moduli di cui sopra.
Per ogni input avviene una votazione, basata sulla maggioranza, che dettermina il valore da mandare in uscita, che è considerato il valore corretto.
Inoltre esamina se se si sono verificati alcuni tipi di fault.

52.23.1. D0

D0 : DEBOUNCE

52.23.2. D1

D1 : DEBOUNCE

52.23.3. D2

D2 : DEBOUNCE

52.23.4. D3

D3 : DEBOUNCE

52.23.5. D4

D4 : DEBOUNCE

52.23.6. D5

D5 : DEBOUNCE

52.23.7. D6

D6 : DEBOUNCE

52.23.8. D7

D7 : DEBOUNCE

52.23.9. V0

-V0 : VOTER

52.23.10. V1

-V1 : VOTER

52.23.11. V2

-V2 : VOTER

52.23.12. V3

-V3 : VOTER

52.23.13. V4

-V4 : VOTER

52.23.14. V5

-V5 : VOTER

52.23.15. V6

-V6 : VOTER

52.23.16. V7

-V7 : VOTER

52.23.17. fault0

-fault0 : Fault

52.23.18. isr_CAN

Signature: isr_CAN() : void

52.23.19. interpreta

Signature: interpreta(pacchetto : byte, cmd : byte, data : byte) : void

52.23.20. exec

Signature: exec() : void, Code Body: // Run exec() of 8 DEBOUNCE

```
D0.exec();
D1.exec();
D2.exec();
D3.exec();
D4.exec();
D5.exec();
D6.exec();
D7.exec();

// Run exec() of 8 VOTER

V0.exec();
V1.exec();
V2.exec();
V3.exec();
V4.exec();
V5.exec();
V6.exec();
V7.exec();
```

52.23.21. init

Signature: init() : void, Code Body: // Run init() of 8 DEBOUNCE

```
D0.init();
D1.init();
D2.init();
D3.init();
D4.init();
D5.init();
D6.init();
D7.init();

// Run init() of 8 VOTER

V0.init();
V1.init();
```

```
V2.init();
V3.init();
V4.init();
V5.init();
V6.init();
V7.init();
```

52.24. t_TTL5V

Parameters (tagged values) to define logic levels of 5V TTL signals.

52.25. t_TTL_IN

Parameters (tagged values) to define logic levels of 3.3V TTL signals.

52.26. T_UART

Tipo che contiene i 5 fili standard della urta più l'enable

52.26.1. CK

Signature: CK()

52.26.2. TX

Signature: TX()

52.26.3. RX

Signature: RX()

52.26.4. CTS

Signature: CTS()

52.26.5. RTS

Signature: RTS()

52.26.6. ENABLE

Signature: ENABLE()

52.27. FSA_OUT

52.27.1. cnt

Signature: -cnt : int
Variable to counter that count the numbr of times that repeats a loop.

52.27.2. fault

Signature: -fault : bool = false
It asserts to have a fault.

52.27.3. cnt_time_glitch

Signature: -cnt_time_glitch : int
Used to mesasure as mask_bit time remains fixed at the same value.

52.27.4. TIME_GLITCH_OUT

Signature: -TIME_GLITCH_OUT

Ratio between the maximum time that bit of SH_OUT remains stable at the same value duration and the execution time loop to measure it.

52.27.5. exec

Signature: exec()