



Politecnico di Torino
Team ICARUS PoliTO



Relazione del lavoro svolto in team

Progettazione e realizzazione in laboratorio

IT Area

Luca Pittalis

28 aprile 2025

Indice

1	Introduzione	1
2	Ingresso nel team e primi progetti	3
3	Il Progetto del Tubo di Pitot	5
3.1	Introduzione	5
3.2	Il sensore	5
3.2.1	Specifiche	7
3.2.2	Incertezze di misura	7
3.3	Il circuito	9
3.4	Il codice	11
3.4.1	Struttura di base	11
3.4.2	Inizio della conversione	11
3.4.3	Ricezione dei dati	11
3.4.4	Calcolo della pressione e della temperatura	14
3.4.5	Calcolo della velocità	14
3.4.6	Incertezza di misura della velocità	15
3.4.7	Pacchetto completo	15
4	Conclusione	16
4.1	Resoconto dell'esperienza	16
4.2	Ringraziamenti	17

CAPITOLO 1

Introduzione

Il team Icarus PoliTO consta di 3 progetti in ambito aerospaziale: RA, ACC e APA.

Record Aircraft (RA), nato nel 2017, è un aeromodello il cui obiettivo è parte dell'attuale sviluppo industriale legato alle nuove fonti di energia: questo UAV (unmanned aerial vehicle) si basa sull'energia solare che alimenta il motore elettrico e gli altri sistemi di bordo.

Air Cargo Challenge (ACC) è un UAV cargo capace di portare a termine una specifica missione. Alcuni parametri richiesti per la competizione includono il tipo di payload da trasportare, la velocità con cui questi vengono trasportati e la loro efficienza.

Advanced Propulsion Aircraft (APA) è un nuovo progetto, ancora in via di sviluppo, nato dall'unione dei campi dell'aeronautica e della propulsione a razzo. L'obiettivo è sviluppare un UAV capace di distinguersi per la velocità massima che è in grado di raggiungere, grazie a un motore a razzo appositamente progettato.

Personalmente, ho avuto la possibilità di lavorare come membro dell'area IT, il cui scopo è realizzare tutti i sistemi elettronici e informatici per portare a termine le missioni dei 3 progetti, e più nello specifico nell'area Firmware, operante nella programmazione a basso livello al fine di mettere in comunicazione l'elettronica e i sistemi di controllo delle nostre missioni. Il team è propenso a utilizzare design custom ovunque sia possibile, inclusi l'elettronica, i computer di volo automatici e i sistemi di acquisizione dati.



Figura 1.1: Render di RA



Figura 1.2: ACC in fase di atterraggio



Figura 1.3: Render di Dart (progetto precedente alla nascita di APA, a cui è stato poi riadattato)

CAPITOLO 2

Ingresso nel team e primi progetti

All'inizio dell'anno accademico 2024-2025 sono entrato in Icarus come membro dell'area IT. Per accedervi sono state necessarie ricerche personali e le conoscenze acquisite nei corsi di "Algoritmi e strutture dati", "Calcolatori elettronici" e "Elettromagnetismo e Teoria dei circuiti" frequentati l'anno precedente. Nei primi incontri settimanali, i membri più anziani del team hanno tenuto diverse lezioni, riguardanti vari ambiti fondamentali per il corretto sviluppo del custom firmware di Icarus.

In particolare, gli argomenti considerati hanno riguardato:

- Utilizzo di Git & GitHub per la gestione delle repository di codice condiviso
- Stabilimento delle convenzioni di codice allo scopo di avere una scrittura in C pulita e ben leggibile
- Utilizzo di CubeMX per la configurazione di microcontrollori STM32
- Comprensione generale dell'utilizzo dei protocolli UART e I2C, utilizzati per la comunicazione tra microcontrollori e periferiche

Dopo qualche sessione introduttiva sulla programmazione dei microcontrollori, abbiamo lavorato a diversi progetti di Rebasing per il controllo e la pulizia del codice di diversi progetti, e in particolare io mi sono dedicato alla verifica di Hypo, una scheda per comunicazioni wireless con protocollo LoRa atta ad accendere un fuso, così da consentire di innestare la reazione per la partenza di un booster allo stato solido da distanza di sicurezza.

In seguito alla verifica, la scheda è stata attivamente utilizzata in uno static fire per il progetto Dart, eseguito con successo il 16/12/2024. Nello stesso test sono anche state provate, per conto dell'Agenzia Spaziale Europea, delle piastrelle di regolite simil-lunare, al fine di verificare il potenziale impatto di un getto a razzo per l'atterraggio sulla superficie del satellite.

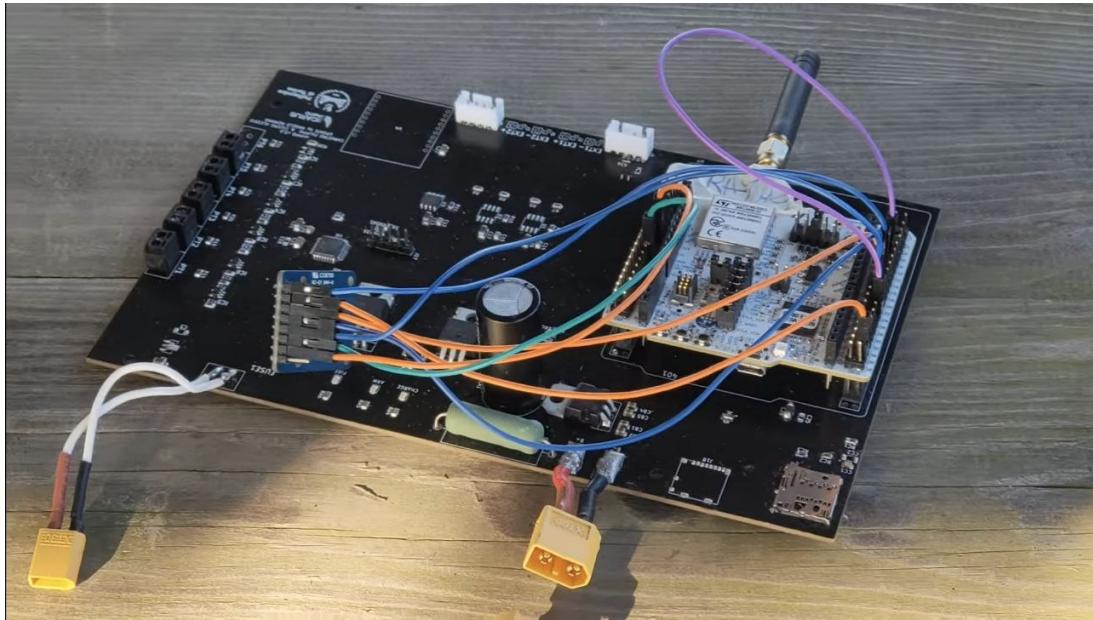


Figura 2.1: Hyppo Board



Figura 2.2: Vista laterale del booster e delle telecamere termiche dell'ESA durante il test statico

CAPITOLO 3

Il Progetto del Tubo di Pitot

3.1 Introduzione

Il progetto relativo al tubo di pitot è il primo lavoro a cui ho avuto la possibilità di dedicarmi partendo da zero. L'obiettivo principale di questo sensore è quello di prendere una temperatura (in °C) e una pressione differenziale (in psi, poi tradotti in Pascal) al fine di calcolare la velocità di un velivolo in un fluido laminare.

3.2 Il sensore

Il sensore trasduttore utilizzato è stato l'MS4525DO di TE Connectivity, e la sua connessione è avvenuta tramite un connettore Nano-Fit della Molex alla board di Air Telemetry V2 (custom board prodotta da Icarus, con alimentazione 5V da USB-A), a sua volta collegata a una nucleo board (modello NUCLEO-F401RE) tramite ST-Link per le operazioni di flash e debug.



Figura 3.1: Dettaglio del tubo di pitot (sinistra) e collegamento di Nucleo e Air Telemetry V2 (destra)

Dal codice seriale del trasduttore, ossia 4525D 5AI 001D 2209 0022 18293, è possibile avere le seguenti informazioni riguardanti il sensore:

- Alimentazione: 5V
- Tipo di Output: A (digital counts from 1638 to 14746 = 10%–90%)
- Interface: I2C
- Intervallo di pressione: ±1 psi (differenziale)
- Pressione minima misurabile $P_{\min} = -1$ psi
- Pressione massima misurabile $P_{\max} = +1$ psi

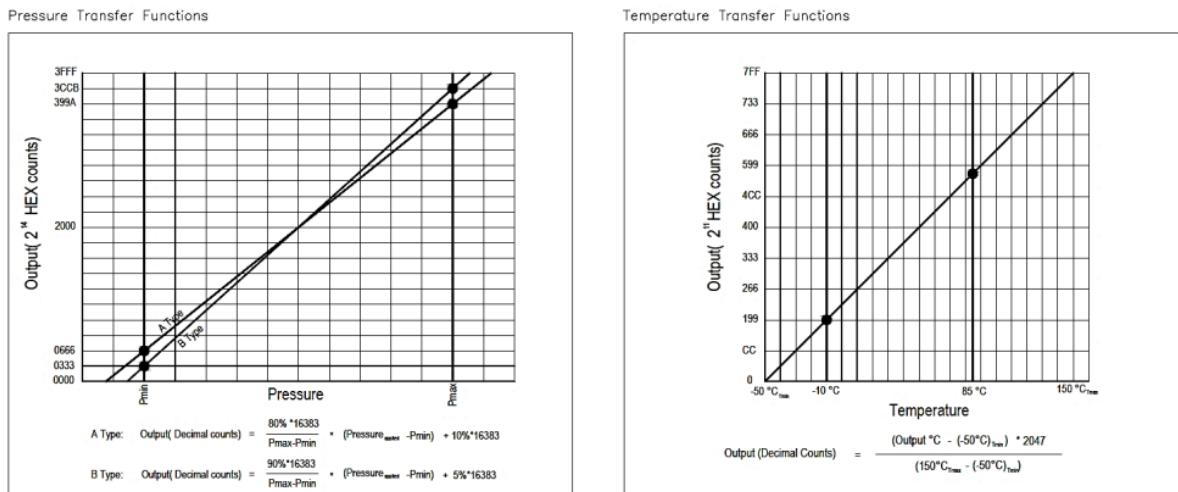
Il tipo di Output è rilevante per il calcolo della pressione differenziale, in quanto il digital count è un valore numerico intero riconducibile, tramite formula di documentazione, direttamente al valore fisico corrispondente. Tra i due metodi possibili (A e B), il sensore utilizzato tiene conto del tipo A, la cui risoluzione è leggermente inferiore, ma con margini di sicurezza decisamente maggiori.

Le due formule citate sopra per il calcolo di pressione differenziale e temperatura (dalle cui inverse si trova la forma utilizzata per la traduzione) sono le seguenti:

$$\text{Output pressione (decimal counts)} = \frac{0.8 \cdot 16383}{P_{\max} - P_{\min}} \cdot (P_{\text{applied}} - P_{\min}) + 0.1 \cdot 16383$$

$$\text{Output temperatura (decimal counts)} = \frac{(\text{Output } ^\circ\text{C}) - (-50^\circ\text{C})_{\min}) \cdot 2047}{(150^\circ\text{C}_{\max} - (-50^\circ\text{C})_{\min})}$$

PRESSURE AND TEMPERATURE TRANSFER FUNCTION



Sensor Output at Significant Percentages

% of Count	Output Type A (psi)	Output Type B (psi)	Digital Counts (decimal)	Digital Counts (hex)
0	$P_{\min} - (P_{\max} - P_{\min}) * 10 / 80$	$P_{\min} - (P_{\max} - P_{\min}) * 5 / 90$	0	0X0000
5		P_{\min}	819	0X0333
10	P_{\min}		1638	0X0666
50			8192	0X2000
90	P_{\max}		14746	0X399A
95		P_{\max}	15563	0X3CCB
100	$P_{\max} + (P_{\max} - P_{\min}) * 10 / 80$	$P_{\max} + (P_{\max} - P_{\min}) * 5 / 90$	16383	0X3FFF

Temperature Output vs Counts

OUTPUT (°C)	Digital Count (decimal)	Digital Counts (hex)
-50	0	0X0000
0	511	0X01FF
10	614	0X0266
25	767	0X02FF
50	1023	0X03FF
85	1381	0X0565
150	2047	0X07FF

Figura 3.2: Grafici relativi a pressione e temperatura

3.2.1 Specifiche

PERFORMANCE SPECIFICATIONS

Supply Voltage¹: 5.0V or 3.3 V_{DC}

Reference Temperature: 25°C (unless otherwise specified)

PARAMETERS	MIN	TYP	MAX	UNITS	NOTES
Accuracy	-0.25		0.25	% SPAN	2
Total Error Band	-1		1	% SPAN	3,7
Burst pressure		SEE TABLE 1			
Common mode pressure		NOT TO EXCEED 300 PSI			
Load Resistance (R_L)	10			kΩ	
Long term stability (offset & span)		±0.5		% SPAN	
Compensated Temperature	-10		85	°C	
Operating Temperature	-25		105	°C	
Weight	1.43		2.02	grams	
Update time		0.5		ms	6
Start time to data ready			8.4	ms	6
Solder temperature		250°C MAX 5 SEC.			

Figura 3.3: Specifiche delle prestazioni

Il tempo di delay relativo alla conversione dell'ADC è stato considerato con la seguente relazione:

$$\text{Start time to data ready} + \text{Update time} + \text{Tolleranza} = 8.4 \text{ ms} + 0.5 \text{ ms} + 1.1 \text{ ms} = 10 \text{ ms}$$

Inoltre, le temperature di utilizzo negli scenari indicati sono tali da non uscire mai dall'intervallo ottimale di compensazione, con valori sempre compresi tra -10°C e 85°C.

Con la verifica del funzionamento della pressione differenziale, è stata denotata anche l'eventuale presenza di un offset, individuato in un valore circa pari a -150 Pa, compensato da un valore positivo applicato ad ogni lettura da parte del sensore. Inizialmente il progetto avrebbe previsto di avere una fase di bilanciamento a terra, prima della partenza, ma questo avrebbe provocato dei problemi nel caso di riavvio del sistema durante il volo, motivo per cui l'idea è stata scartata in favore della calibrazione con un valore fisso, più che sufficiente per questo tipo di misurazione.

Infine, allo scopo di avere la registrazione di un ΔP positivo e non negativo all'aumentare della pressione sul sensore, è opportuno posizionare i due tubi di gomma correttamente, invertendone il verso nel caso in cui questi registrino un aumento negativo della pressione a una qualsiasi influenza esterna (banalmente anche il semplice soffio di una persona).

3.2.2 Incertezze di misura

Come visibile dalle specifiche relative alle prestazioni, l'incertezza di misura è pari all'1% dello span.

Per quanto riguarda la pressione differenziale, l'intervallo è facilmente visibile dal codice seriale del trasduttore, con valori che vanno da -1 psi a 1 psi.

Per quanto riguarda la temperatura, è possibile recuperare l'intervallo dalla tabella riportata nella tabella inferiore della Figura 3.2, dove si possono notare valori che vanno da -50°C a 150°C.

Ricapitolando, quindi, i valori dello span sono i seguenti:

- Pressione: $2 \text{ psi} \approx 13790 \text{ Pa}$
- Temperatura: 200°C

È possibile calcolare le incertezze di misura in questo modo:

- Pressione: $\delta(\Delta P) = 1\% \cdot 2 \text{ psi} = 0.02 \text{ psi} \approx 137.9 \text{ Pa}$
- Temperatura: $\delta T = 1\% \cdot 200^\circ\text{C} = 2^\circ\text{C}$

Come si può notare, il valore dell'incertezza di misura della pressione differenziale è del tutto simile a quello dell'offset di 150 Pa, delimitando quindi una variazione in linea con i valori attesi dallo strumento.

BLOCK DIAGRAM

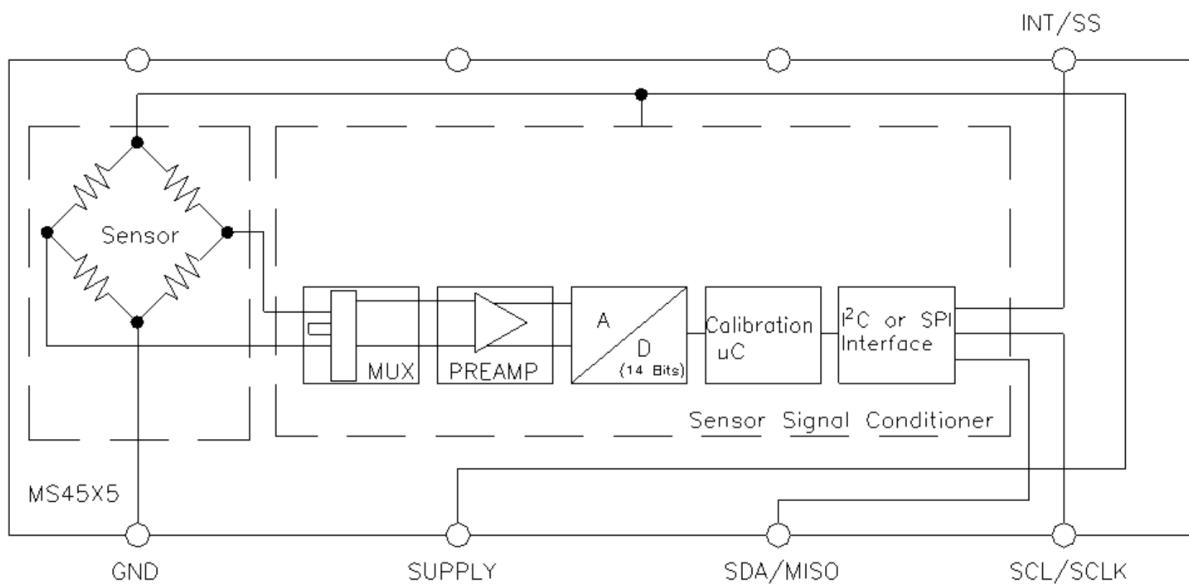


Figura 3.4: Schematica interna del sensore

3.3 Il circuito

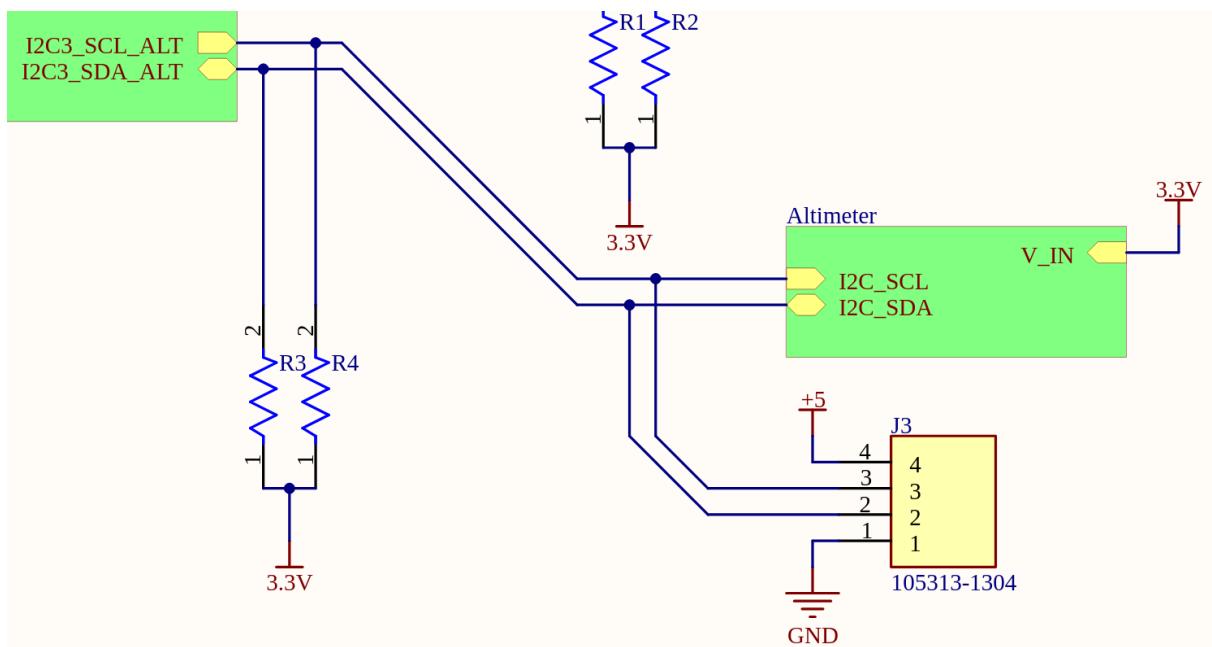


Figura 3.5: Dettaglio del circuito schematico di Air Telemetry V2, il pitot è il modulo giallo

Come si può notare, l'handler I2C3 è condiviso tra altimetro (altra periferica della board) e pitot, fattore di cui si dovrà tenere conto nella gestione del flusso dei dati.

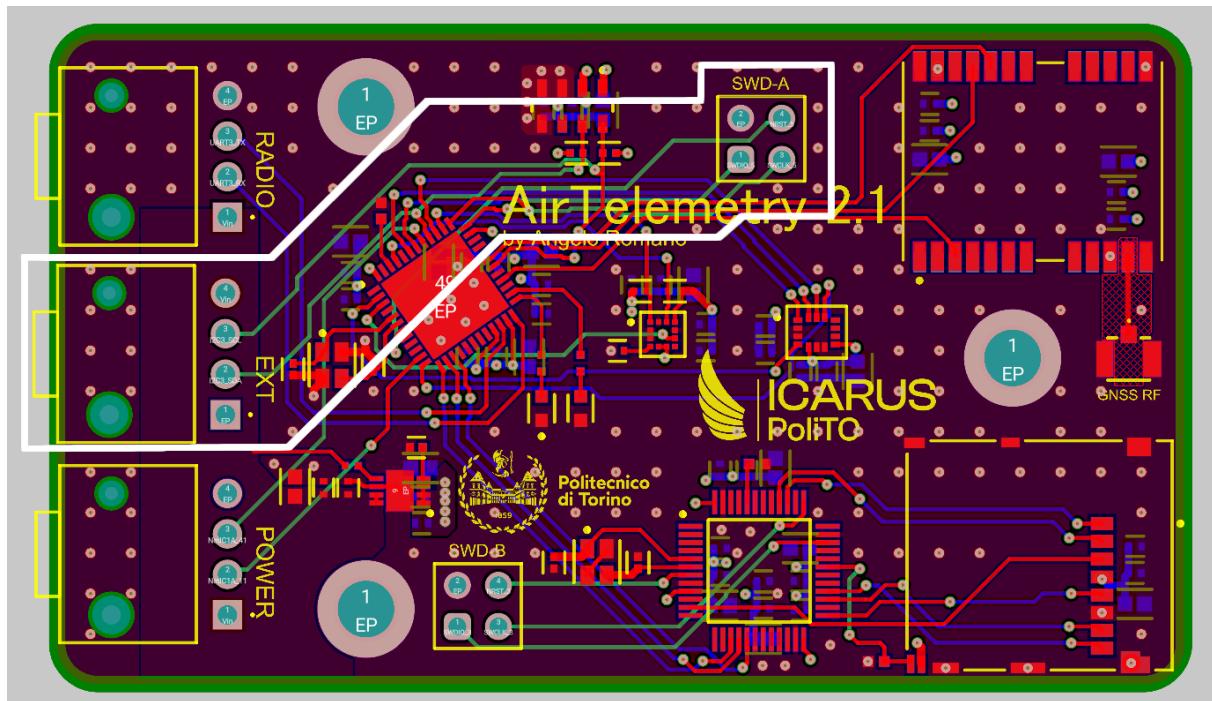


Figura 3.6: Air Telemetry V2 con evidenziata in bianco la sezione relativa al Pitot

Andando più nello specifico, di seguito i collegamenti per i PIN del connettore J3 e dell'SWD-A.

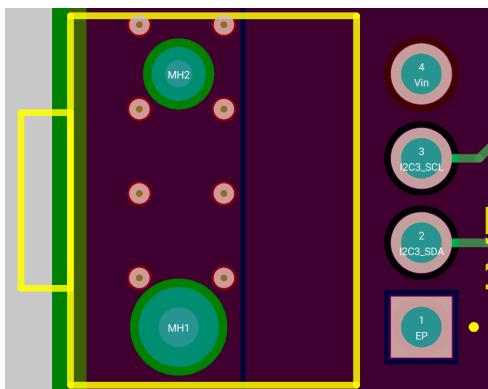


Figura 3.7: Configurazione dei pin per J3 (connettore Nano-Fit)

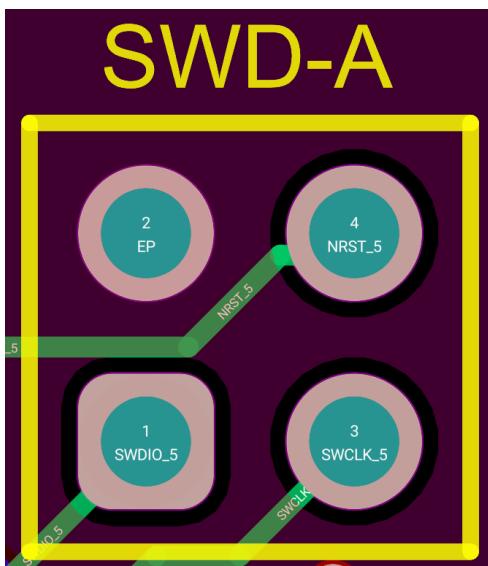


Figura 3.8: Configurazione dei pin per SWD-A

Configurazione:

- Vin: collegamento a 5V
- I2C3_SCL: collegamento al system clock
- I2C3_SDA: collegamento ai dati
- EP: collegamento a ground

Nota bene: rispetto alle altre due JST, in questo caso 5V e ground sono in posizioni opposte.

Configurazione:

- SWDIO_5: collegamento a data
- EP: collegamento a ground
- SWCLK_5: collegamento al clock
- NRST_5: collegamento al reset attivo basso (negated reset)

Nota bene: SWD-A non è utilizzato solamente dal Pitot, e i suoi pin si collegano infatti direttamente alla nucleo board.

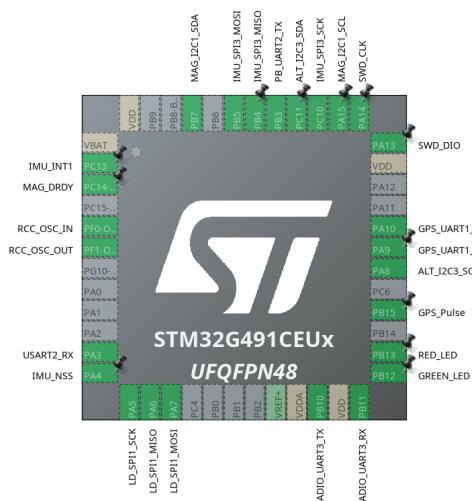


Figura 3.9: Schematica della nucleo board STM32 utilizzata

3.4 Il codice

3.4.1 Struttura di base

Il codice ha lo scopo primario di gestire il flusso dati in arrivo dal pitot, ricevendo i digital count e traducendoli in un valore fisico. Di seguito è indicata la sequenza delle fasi principali.

1. Inizio della conversione da parte dell'ADC
2. Ricezione dei dati tramite callback I2C (e partenza della richiesta successiva)
3. Calcolo di pressione differenziale e temperatura a partire dai digital count
4. Uso di pressione e di temperatura per il calcolo della velocità

3.4.2 Inizio della conversione

Dentro main.c, viene chiamata la seguente funzione all'inizializzazione:

```
1 uint8_t InitSystem(void)
```

Al suo interno è chiamata la seguente funzione relativa al pitot:

```
1 void MS4525D0_start_conv(&ms45525do);
```

Dentro la funzione di conversione, avviene la trasmissione del primo pacchetto (MS4525D0_INIT):

```
1 void MS4525D0_start_conv(MS4525D0_handler_TYPEDEF *handle) {
2
3     // First packet for initialization
4     handle->packetType = MS4525D0_INIT;
5
6     // Emptying the buffer for initialization
7     memset(conversionBuffer, 0, sizeof(conversionBuffer));
8
9     // Start ADC Conversion
10    conversionBuffer[0] = 0x1EU;
11    HAL_I2C_Master_Transmit(handle->I2C_handle, MS4525D0_I2C_ADDRESS, (uint8_t*)
12        conversionBuffer, 1, 1000);
}
```

3.4.3 Ricezione dei dati

La ricezione dei dati avviene dal main tramite la seguente funzione di interrupt:

```
1 void HAL_I2C_MemRxCpltCallback(I2C_HandleTypeDef *hi2c) {
2     // Altimeter
3     [...] // (code relative to the altimeter)
4
5     // MS4525D0
6     if(hi2c == ms45525do.I2C_handle && I2C_readFlag == I2C_MS4525D0){
7         ms45525do_data.timestamp = __HAL_TIM_GetCounter(&htim2);
8         if(MS4525D0_MemRxCpltCallback(&ms45525do, &ms45525do_data)){
9             // Log data
10        }
11    }
12 }
```

Come citato precedentemente, l'altimetro e il pitot hanno lo stesso handler, I2C3, e questo risulterebbe un problema in questo tipo di situazione. Per ovviare al possibile conflitto, alla trasmissione è associato un `DelayElapsedCallback` allo scopo di identificare, tramite `I2C_readFlag`, quale periferica sta trasmettendo in ogni possibile istante. La gestione avviene in questo modo:

```

1 void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim) {
2     if (htim == &htim2) {
3         // Altimeter
4         if (_HAL_TIM_GET_IT_SOURCE(alt.tim_handle, alt.tim_cc_it_source)) {
5             I2C_readFlag = I2C_ALT;
6             ALT_TIM_OC_DelayElapsedCallback(&alt);
7         }
8         // MS4525DO
9         if (_HAL_TIM_GET_IT_SOURCE(ms45525do.tim_handle, ms45525do.
10            tim_cc_it_source)) {
11             I2C_readFlag = I2C_MS4525DO;
12             MS4525DO_TIM_OC_DelayElapsedCallback(&ms45525do);
13         }
14     [...]
15 }
```

Sempre nella stessa funzione viene anche chiamato `MS4525DO_TIM_OC_DelayElapsedCallback`, che si occupa anche della ricezione e della lettura dei dati da `conversionBuffer`, prendendo dal primo e secondo byte il valore della pressione differenziale e dal terzo e quarto byte i valori della temperatura.

```

1 void MS4525DO_TIM_OC_DelayElapsedCallback(MS4525DO_handler_typedef *handle){
2     _HAL_TIM_CLEAR_IT(handle->tim_handle, handle->tim_cc_it_source);
3     HAL_I2C_Mem_Read_IT(handle->I2C_handle, MS4525DO_I2C_ADDRESS, 0x00U, 1, (
4         uint8_t *) conversionBuffer, 4);
}
```

Dopo la lettura è chiamato l'interrupt di `HAL_I2C_MemRxCpltCallback`, che, come nel già citato caso del `DelayElapsedCallback`, fa uso di `I2C_readFlag` per differenziare altimetro e pitot.

In questa stessa istanza viene anche salvato il timestamp, ossia il tempo in microsecondi tra l'inizializzazione della board e la ricezione del pacchetto corrente.

```

1 void HAL_I2C_MemRxCpltCallback(I2C_HandleTypeDef *hi2c) {
2     // Altimeter
3     if (hi2c == alt.I2C_handle && I2C_readFlag == I2C_ALT) {
4         // Convert data, check if new data is available
5         if (ALT_RxCpltCallback(&alt, &altData)) {
6             // Log data
7         }
8     }
9
10    // MS4525DO
11    if(hi2c == ms45525do.I2C_handle && I2C_readFlag == I2C_MS4525DO){
12        ms45525do_data.timestamp = _HAL_TIM_GetCounter(&htim2);
13        if(MS4525DO_MemRxCpltCallback(&ms45525do, &ms45525do_data)){
14            // Log data
15        }
16    }
17 }
```

All'interno di `MS4525DO_MemRxCpltCallback` avvengono i seguenti passaggi:

1. Verifica la tipologia del pacchetto
2. Se il pacchetto è `MS4525DO_INIT`, prosegui al pacchetto successivo (che sarà `MS4525DO_DATA`)
3. Se il pacchetto è `MS4525DO_DATA`, verifica lo stato e se in `STATUS_GOOD` prosegui con i calcoli per pressione e temperatura
4. Imposta nuovamente il timer per tenere conto dei tempi di conversione dell'ADC
5. Segnala l'invio di un nuovo pacchetto
6. Assegna alla struttura `data` il valore attuale dello status

```

1 MS4525DO_status_t MS4525DO_MemRxCpltCallback(MS4525DO_handler_TYPEDEF *handle,
2   MS4525DO_data *data) {
3
4     MS4525DO_status_t status = STATUS_GOOD;
5
6     if(handle->packetType == MS4525DO_INIT) {
7       handle->packetType = MS4525DO_DATA;
8     } else {
9       // Evaluation of status as STATUS_FAULT
10      status = (conversionBuffer[0] >> 6) & STATUS_FAULT;
11
12      //If status is either STATUS_GOOD or STATUS_STALE_DATA, continue with
13      // calculations
14      if(status != STATUS_FAULT) {
15        MS4525DO_pressure_temperature_calculations(data);
16      }
17
18      memset(conversionBuffer, 0, sizeof(conversionBuffer));
19
20      if(handle->packetType == MS4525DO_DATA){
21        // Modifying it from zero so that we can actually receive data
22        conversionBuffer[0] = 0x54U;
23      }
24
25      // Setting the timer again to account for conversion time
26      __HAL_TIM_SET_COMPARE(handle->tim_handle, handle->tim_channel,
27      __HAL_TIM_GET_COMPARE(handle->tim_handle, handle->tim_channel) +
28      MS4525DO_CONVERSION_TIME);
29      HAL_I2C_Master_Transmit(handle->I2C_handle, MS4525DO_I2C_ADDRESS, (uint8_t*)
30      conversionBuffer, 1, 1000);
31
32      data->status = status;
33      return status;
34    }

```

3.4.4 Calcolo della pressione e della temperatura

All'interno di `MS4525D0_MemRxCpltCallback` viene chiamata la seguente funzione:

```
1 void MS4525D0_pressure_temperature_calculations(MS4525D0_data *data)
```

I cui passaggi sono i seguenti:

1. Estrazione di pressione e temperatura dai quattro byte di `conversionBuffer` (i primi due per la pressione e i rimanenti per la temperatura)
2. Calcolo per passare dai digital count ai valori effettivi di pressione e temperatura
3. Traduzione della pressione da psi a pascal
4. Memorizzazione dei valori in `data`

```
1 void MS4525D0_pressure_temperature_calculations(MS4525D0_data *data) {
2
3     // Extracting pressure and temperature counts from conversionBuffer
4     uint32_t pressureCount = (uint16_t)(conversionBuffer[0] & 0x3F) << 8 |
5         conversionBuffer[1];
6     uint32_t temperatureCount = (uint16_t)(conversionBuffer[2]) << 3 |
7         conversionBuffer[3] & 0xE0 >> 5;
8
9     // Calculation for Pressure [psi] and Temperature [Celsius] (formulas from
10    // pitot_datasheet, type A)
11    data->temperature = (float)(temperatureCount) * (MS4525D0_T_MAX -
12        MS4525D0_T_MIN) / T_CNT + MS4525D0_T_MIN;
13    float pressurePsi = ((float)(pressureCount) - OUTPUT_TYPE_MULTIPLIER_C *
14        P_CNT) * ((MS4525D0_P_MAX - MS4525D0_P_MIN) / (OUTPUT_TYPE_MULTIPLIER_D *
15        P_CNT)) + MS4525D0_P_MIN;
16
17     // Translation from psi to pascals and offset correction
18     data->pressure = pressurePsi * PSI_TO_PA_MULTIPLICATION_FACTOR +
19         MS4525D0_OFFSET;
20
21     [...] // Calculation for speed
22 }
```

3.4.5 Calcolo della velocità

Per il calcolo della velocità è stata utilizzata l'equazione di Bernoulli, nella seguente forma:

$$P_{\text{tot}} = P_{\text{stat}} + \frac{1}{2} \rho v^2$$

$$\Delta P = P_{\text{tot}} - P_{\text{stat}} = \frac{1}{2} \rho v^2$$

$$v = \sqrt{\frac{2\Delta P}{\rho}}$$

$$\rho = \frac{P_{\text{stat}}}{RT}$$

$$v = \sqrt{\frac{2\Delta P R T}{P_{\text{stat}}}}$$

E considerando i seguenti valori:

- Pressione statica $P_{\text{stat}} = 101325 \text{ Pa}$
- Pressione differenziale ΔP misurata dal sensore
- Temperatura assoluta T del fluido, calcolata come somma della temperatura in Celsius e 273.15 K
- Costante dei gas perfetti specifica per l'aria $R = 287.05 \frac{\text{J}}{\text{kg}\cdot\text{K}}$

Si arriva al calcolo finale per la velocità, sempre in `MS4525D0_pressure_temperature_calculations`.

```

1 // Calculation for speed [formula used: v = sqrt((2*PRT)/p)]
2     float kelvinTemperature = data->temperature + C_TO_KELVIN;
3     data->speed = sqrt((2 * data->pressure * AIR_CONSTANT * kelvinTemperature) /
        STATIC_PRESSURE);

```

Nota bene: la velocità qui calcolata è solamente relativa al pitot, ma questa andrebbe adattata sulla base dell'angolo d'attacco del velivolo, fattore considerato poi nei sistemi di controllo del volo, estranei a questa documentazione.

3.4.6 Incertezza di misura della velocità

La formula applicata per il calcolo dell'incertezza della velocità, trascurando l'incertezza di P_{stat} e di R e tenendo conto dei valori di $\delta(\Delta P)$ e δT calcolati precedentemente, è la seguente:

$$v = \sqrt{\frac{2\Delta PRT}{P_{\text{stat}}}}$$

$$\delta v = \sqrt{\left(\frac{\partial v}{\partial \Delta P} \delta(\Delta P)\right)^2 + \left(\frac{\partial v}{\partial T} \delta T\right)^2}$$

$$\delta v = \sqrt{\left(\frac{RT}{vP_{\text{stat}}} \delta(\Delta P)\right)^2 + \left(\frac{\Delta PR}{vP_{\text{stat}}} \delta T\right)^2}$$

Pertanto, in funzione dell'attuale velocità e pressione differenziale, è possibile determinare l'incertezza di misura. Inoltre, in termini macroscopici, si può osservare che l'incertezza è inversamente proporzionale alla velocità e direttamente proporzionale alla pressione differenziale.

3.4.7 Pacchetto completo

Una volta calcolata la pressione, la temperatura e la velocità, avendo anche tenuto traccia di status e timestamp, un pacchetto correttamente ricevuto presenterà la seguente forma:

```

    ▼ VARIABLES
        ▼ Local
            ▼ data = 0x20000424 <ms45525do_data>
                status = STATUS_GOOD
                temperature = 26.2090836
                pressure = 196.819824
                speed = 18.2711582
                timestamp = 2238720

```

Figura 3.10: Esempio di pacchetto generato da una perturbazione laminare

CAPITOLO 4

Conclusione

4.1 Resoconto dell'esperienza

Penso che questo progetto sia stato per me enormemente formativo, sia per l'esperienza sul campo che per le grandi conoscenze acquisite lungo il percorso. Lo studio del firmware e dei sistemi embedded presenta delle sfide talvolta di difficile risoluzione, spesso estranee a qualsiasi argomento trattato a lezione e dunque necessitanti di molto spirito di iniziativa, tenacia e pensiero logico di approccio ai problemi.

Trovo che lavorare con Icarus mi abbia dato una grande mano per quanto riguarda la scelta del mio percorso futuro, sia per la laurea magistrale che per una sempre più vicina carriera nel mondo del lavoro. Grazie al team studentesco ho avuto la possibilità di conciliare due delle mie più grandi passioni, quali l'informatica e l'aerospazio, in un ambiente che difficilmente sarei riuscito a trovare altrove.

Uno dei problemi più grandi all'interno di gruppi studenteschi di questo tipo è il continuo succedersi di membri sempre nuovi, i quali difficilmente, per via del termine degli studi, rimarranno tra le fila del team per più di qualche anno. Questo continuo turnover, pur apportando nuove energie e entusiasmo al team, rende tuttavia difficile tramandare le conoscenze acquisite nel tempo. Relazioni come questa si pongono come obiettivo principale quello di porre soluzione proprio a questo genere di difficoltà, catalogando il know-how e rendendolo disponibile a chiunque siglerà il futuro successo del team.

Così come altri hanno avuto la premura di insegnarmi e di rendermi ciò che sono, io stesso proverò al meglio delle mie possibilità a portare altri sotto la mia ala, preferibilmente a propulsori spenti, per consentirgli di imparare a camminare, a correre, e un giorno perfino a volare.

Grazie per l'attenzione.

4.2 Ringraziamenti

Oltre a tutti i membri di Icarus, senza i quali niente di questo sarebbe stato possibile, vorrei dedicare questo passaggio finale ad alcune persone in particolare.

In primis, ringrazio Ed e Mattia di AESA (oltre che già membri di Icarus) per avermi spinto ad entrare nel team, decisione che è senza alcun dubbio valsa il tempo e le energie dedicateci.

In secondo luogo, ringrazio Angelo e Samuele per l'impegno e la dedizione dimostrati nei miei confronti e verso gli altri membri del team, per averci trasmesso tutto ciò che sappiamo sui sistemi embedded e per le numerose mattinate e serate trascorse insieme al Toolbox.

And lastly, I'd like to thank Eren for helping me throughout the whole project, from the coding, the debugging, the troubleshooting, all the way to the soldering of the Nano-Fit and the various air compressor and bike-fan shenanigans. Memories were made.