



Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

RTEMS su RPi per applicazioni “aerospace-like”

Relatore: Prof. Domenico Giorgio Sorrenti

Co-relatore: Prof. Pietro Braione

Tutor aziendale: Ing. Fabrizio Bernardini

Relazione della prova finale di:

Clark Ezpeleta

Matricola 832972

Anno Accademico 2020-2021

Indice

Introduzione	1
1 Tecnologie	3
1.1 RTEMS	3
1.2 Raspberry Pi	5
1.3 Eclipse	6
2 Porting di RTEMS su RPi	7
3 Integrazione hardware e software	8
4 Attività sperimentale	9
4.1 Obiettivi	9
4.2 Test set-up	9
4.3 Test application software	11
4.4 Risultati finali	12
5 Conclusioni	13

Introduzione

Il lavoro che ho svolto ha due obiettivi principali: il porting di RTEMS su Raspberry Pi e la creazione di applicativi RTEMS per testare il corretto funzionamento delle interfacce GPIO, UART, I2C, SPI e l'utilizzo degli interrupts tramite le api di RTEMS.

Per poter svolgere il porting ho dato una lettura al RTEMS User Manual[1], per comprendere meglio i concetti di RSB(RTEMS Source Builder) e BSP (board support packages), ed ho utilizzato la guida fornita da Giorgio Basile [2], membro della BIS-Italia, che raccoglie tutti i passaggi esposti sul blog di Alan Tech per il porting della versione 4.11.

Purtroppo i passaggi illustrati sul blog non sono totalmente corretti, poichè sono per il porting della versione di RTEMS precedente a quella che ho utilizzato, cioè la 5.1 che è la più recente e stabile. Dopo aver effettuato correttamente il porting ho creato una guida in italiano che raggruppa tutti i passaggi effettuati integrando le correzioni necessarie, ed è stata corretta anche la guida di Giorgio Basile [3]. Dopo aver settato l'ambiente di lavoro settato ho iniziato a creare gli applicativi RTEMS da eseguire sulla Raspberry Pi.

Per poter creare gli applicativi RTEMS ho dovuto familiarizzare con il linguaggio C, leggere l'RTEMS Classi API Guide [4] ed analizzare i codici sorgente di esempio trovati nel git repository di asuol[5] per poter comprendere l'utilizzo delle API.

Tutto il lavoro è stato eseguito in modalità "smart-working", per questo motivo mi è stata spedita dalla BIS-Italia la Raspberry Pi 3B+ per poter effettuare il porting, e dalla Microchip dei componenti aggiuntivi utili per gli applicativi RTEMS di test dell'interfaccia I2C e SPI.

Oltre all'attività software ho dovuto eseguire una piccola attività hardware, cioè creare dei circuiti saldando i vari componenti e utilizzando la breadboard in modo da poterli collegare alla Raspberry Pi, per fare ciò ho seguito gli schemi elettrici che mi ha inviato Fabrizio Bernardini e ho letto i datasheet dei componenti dell'interfaccia I2C [6] [7] e SPI [8] [9] in modo da comprendere il loro funzionamento e poter creare i driver.

Premesso tutto ciò ho suddiviso la mia relazione nei seguenti capitoli:

- **Capitolo 1** - in questo capitolo vengono descritte le principali tecnologie utilizzate durante il mio lavoro. Innanzitutto viene descritto RTEMS che è il sistema operativo su cui si basa tutta l'attività, dopodichè viene descritta

la Raspberry Pi su cui verranno eseguiti gli applicativi RTEMS, ed infine viene descritto Eclipse che è l'IDE utilizzato per creare gli applicativi.

- **Capitolo 2** - in questo capitolo viene descritta tutta l'attività di porting e la definizione della toolchain per poter realizzare applicativi RTEMS. Vengono esposte tutte le problematiche rilevate e le loro soluzioni.
- **Capitolo 3** - in questo capitolo vengono descritti le api che RTEMS ha a disposizione, la loro struttura e in che modo li ho testati e per quali motivi si è scelto di testarli.
- **Capitolo 4** - in questo capitolo viene descritta l'attività software che si basa sulla creazione degli applicativi RTEMS per testare il corretto funzionamento delle api di RTEMS. Viene esposto anche una descrizione dei componenti aggiuntivi offerti da Microchip di cui ho creato i driver per poterli utilizzare con RTEMS.
- **Capitolo 5** - in questo capitolo viene descritto ciò che si è raggiunto e la possibile estensione del mio lavoro.

1 Tecnologie

1.1 RTEMS



RTEMS sta per Real-Time Executive MultiProcessor System ed è un sistema operativo real-time (RTOS) general purpose open source (licenza GPL 2.0 modificata) gestito da OAR Corporation. Il suo sviluppo iniziò nella fine degli anni 80 utilizzando il linguaggio Ada e C, e venne usato inizialmente per scopi militari, invece le prime versioni utilizzabili sono state rese disponibili su server ftp nel 1994.

Attualmente viene utilizzato in molti settori tra cui quello aereo spaziale, infatti è stato utilizzato in alcune missioni spaziali ad esempio sul MRO (Mars Reconnaissance Orbiter) è in esecuzione un applicativo RTEMS che controlla l'Electra UHT Transceiver (EUT) che serve per la comunicazione tra Marte e la Terra .

E' da notare che RTEMS è stato validato dall'ESA, European Space Agency, ciò vuol dire che sono stati scritti dei programmi che riproducono gli scenari critici (ad esempio la gestione di molti device oppure la gestione e l'esecuzione concorrente dei task), e sono stati eseguiti sul sistema operativo per poter verificare il suo corretto funzionamento in quei casi. La validazione viene tutt'ora fatta poiché è un sistema operativo che è in continua evoluzione e avrà più funzionalità con il passare del tempo.

RTEMS può essere visto come un insieme di direttive raggruppate in una serie di manager, che si occupano di varie funzionalità tra cui il controllo e la sincronizzazione dei task e processori, la gestione della memoria e la mutua esclusione. Invece la gestione dello scheduling, dispatching e object management sono forniti dal executive core.

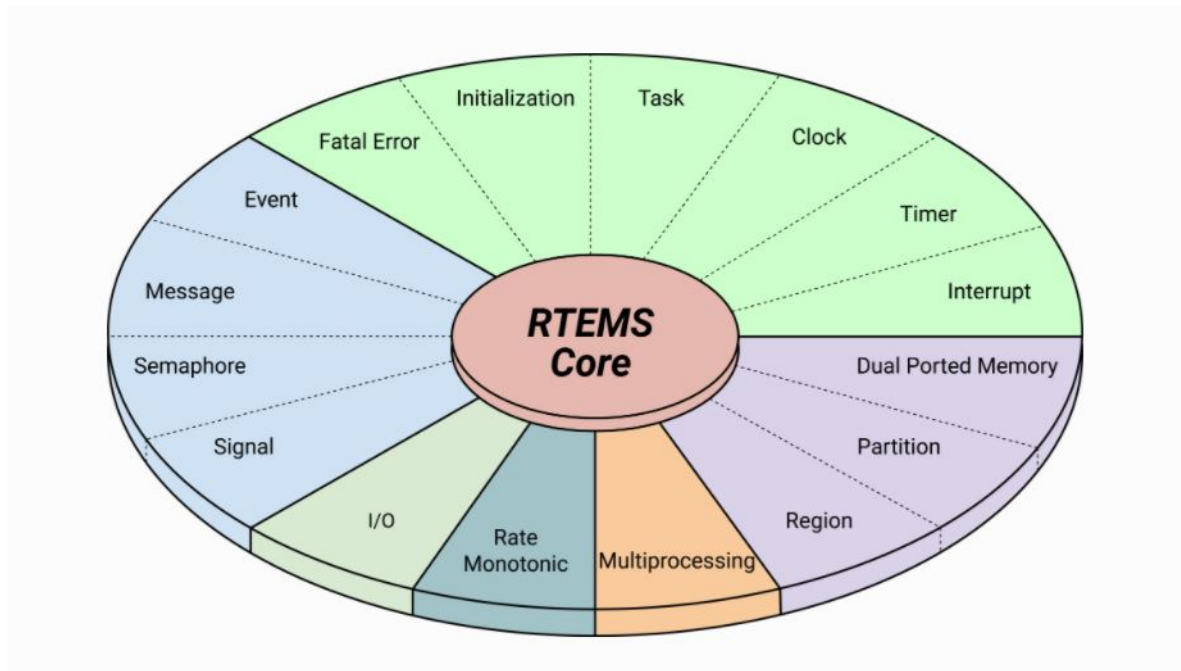


Figura 1.1: Architettura RTEMS

Utilizzando i managers di RTEMS lo sviluppatore può concentrarsi al solo sviluppo dell'applicativo e ciò riduce notevolmente il tempo di sviluppo.

Design di RTEMS :



Figura 1.2: Struttura applicativa RTEMS

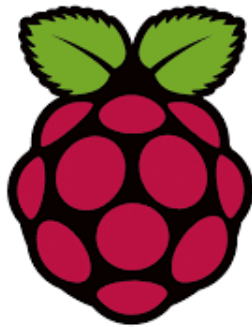
Da come si può notare RTEMS Executive è un intermediario tra il codice dell'applicativo e il target hardware, invece le dipendenze hardware con altri device sono localizzati nel livello "device drivers". Il RTEMS I/O manager incorpora queste dipendenze hardware nel sistema mentre allo stesso momento fornisce all'applicazione code l'accesso ad esse. Queste dipendenze hardware

sono isolate in specifiche BSP, Board Support Packages, per questo motivo il porting di un applicativo RTMES su altri processori è semplice poichè basterebbe selezionare la BSP del microprocessore su cui si vuol eseguire l'applicativo e compilare con le sue librerie.

In questo modo durante lo sviluppo di un applicativo real-time si ha la totale indipendenza dall'architettura dei microprocessori.

E' disponibile il porting di RTEMS su molte architetture CPU tra cui ARM, MIPS, LEON,ERC32 e i PowerPC; durante il mio lavoro ho trattato il porting su architettura ARM utilizzando una Raspberry Pi 3B+.

1.2 Raspberry Pi



Raspberry Pi è una serie di computer a scheda singola sviluppata da Raspberry Pi Foundation in collaborazione con la Broadcom, in Inghilterra.

Originariamente è stato usato per insegnare le basi dell'informatica nelle scuole e nei paesi in via di sviluppo, ma attualmente grazie al suo basso prezzo viene usato anche nel settore della robotica, domotica e in molti altri. Al momento sono state rilasciate quattro generazioni di Raspberry Pi, e tutti i modelli hanno un Broadcom SoC (System on Chip) con processore ARM integrato e on-chip GPU (Graphics Processing Unit).

Durante il mio lavoro viene usata la **Raspberry Pi 3B+** che utilizza il Broadcom BCM2837B0 SoC [10] con processore Cortex-A53 (ARMv8) 64-bit.

1.3 Eclipse

Eclipse è un IDE open source rilasciato con licenza EPL (Eclipse Public License) creato da IBM usato principalmente per la programmazione in Java ma grazie a vari plugin può essere usato per altri linguaggi di programmazione come C, C++, COBOL, Python e molti altri.

L'ambiente di sviluppo Eclipse include l'Eclipse Java development tools per Java, e l'Eclipse CDT per C/C++.

Per utilizzare RTEMS gcc cross compiler su Eclipse C, bisogna installare il plugin di RTEMS e settare le variabili di ambiente.

2 Porting di RTEMS su RPi

L'attività di porting è la prima fase del lavoro e consiste in:

1. Installazione della tool-suite sul computer host, dove vengono creati gli applicativi RTEMS.
2. Verifica del corretto funzionamento della tool-suite, eseguendo i programmi di test su Raspberry Pi
3. Configurazione del IDE Eclipse C, per la creazione di eseguibili RTEMS
4. Provare a compilare e buildare i programmi sia da terminale, e sia da Eclipse C

Per svolgere tutta la procedura mi sono munito di un computer con sistema operativo Ubuntu, una scheda Raspberry Pi 3B+, una microSD, e convertitore TTL USB.

RTEMS in sé è complesso per questo il team di RTEMS ci fornisce "l'ecosistema di RTEMS" che è una collezione di strumenti, packages, codici sorgente e documentazione, utile per definire come sviluppare, mantenere e usare RTEMS. Durante il porting ho utilizzato due importanti strumenti che fanno parte dell'ecosistema RTEMS, e sono :

- RTEMS RSB = l'RTEMS Source Builder è tool molto utile per compilare e buildare i moduli di RTEMS e delle BSP
- BSP raspberrypi = la Build Support Package è il codice di supporto, che contiene le librerie di RTEMS per una specifica scheda (ad esempio la BSP raspberrypi viene usata per la Raspberry Pi 3B+)

Dopo aver effettuato correttamente tutti i passaggi per il porting, ho creato una guida che li espone, in questo modo che qualunque utente voglia approcciarsi ad RTEMS su RPi per la prima volta riesca ad eseguirlo senza dover cercare tra le tante fonti sparse su internet.

3 Integrazione hardware e software

Effettuato il porting sono passato all'attività software, cioè la creazione di RTEMS executive per testare il funzionamento delle RTEMS Classic API che sono le api per l'utilizzo delle interfacce.

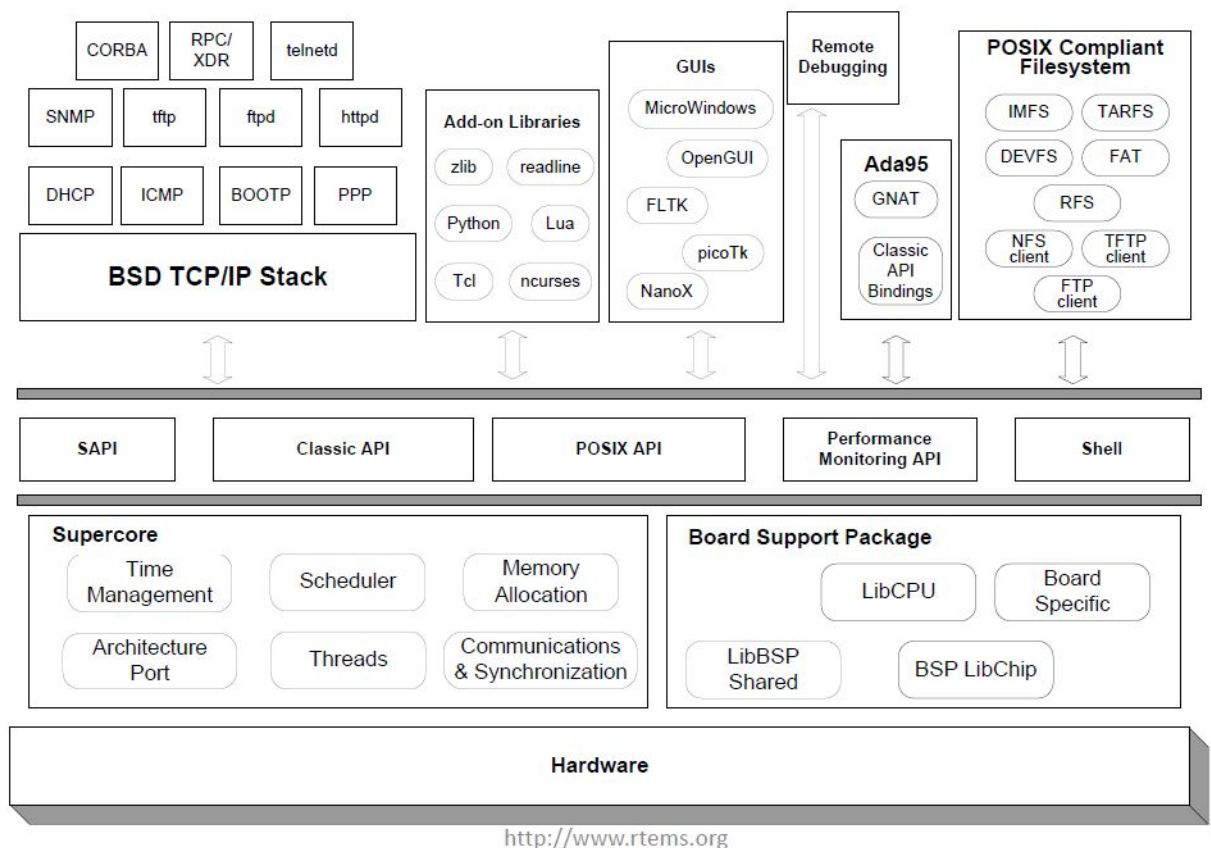


Figura 3.1: Architettura dettagliata applicativo RTEMS

—
TODO
—

4 Attività sperimentale

4.1 Obiettivi

Gli obiettivi dell'attività sperimentale sono:

TODO

4.2 Test set-up

Ogni programma di test creato implica un circuito elettrico differente costruito con jumper, breadboard, resistenze, condensatori, bottoni, led e schede aggiuntive mandate dalla Microchip su cui ho dovuto saldare cavi e componenti per poterli utilizzare e collegare alla Raspberry Pi.

I componenti aggiuntivi sono :

- MCP3425 SOT23-6 Evaluation Board [6]: un evaluation board, con su l'ADC (Analog Digital Converter) MCP3425, controllato tramite protocollo I2C
- MSOP-10 and MSOP-8 Evaluation Board [9]: un evaluation board che serve per il collegamento del componente MCP4822 alla Raspberry Pi
- MCP4822 [8]: un DAC (Digital Analog Converter) controllato tramite protocollo SPI

Il componente MPC3425 saldato sul eval board è un ADC da 16 bit che ha come input due tensioni V_{IN}^+ e V_{IN}^- e come output una tensione pari a $V_{IN}^+ - V_{IN}^- \times PGA$. Il componente ha la seguente configurazione predefinita:

- Programmable Gain Amplifier(PGA) = 1
- Continuous Conversion
- Programmable Data Rate = 12 bit

Durante il lavoro viene utilizzata questa configurazione, nel caso si volesse cambiare la configurazione del componente, si può eseguire una scrittura di due byte dove il primo byte rappresenta l'indirizzo di periferica e il bit R/W (nel nostro caso è 11010000) invece il secondo contiene i bit di configurazione

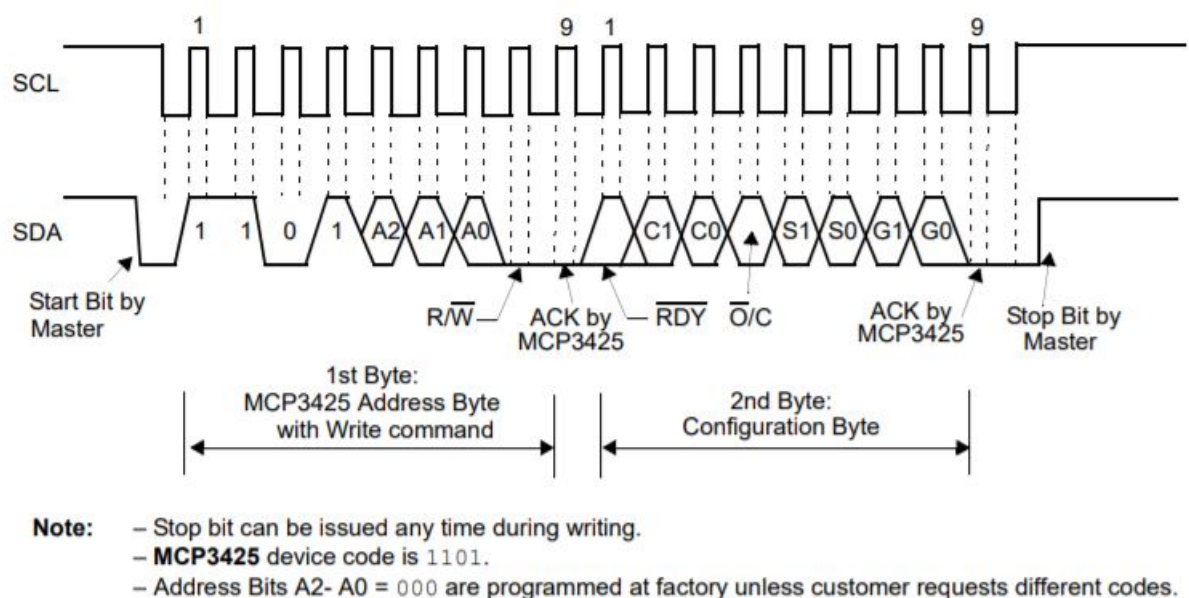


Figura 4.1: Diagramma temporale scrittura su MCP3425

La lettura del valore in uscita dal DAC deve essere eseguita nel seguente modo:

1. mandare sul bus I2C l'indirizzo di periferica con R/W pari a 1
2. lettura dei due byte
3. conversione dei byte per avere il valore effettivo della tensione

—
TODO : DESCRIZIONE COMPONENTE MCP4822

—
TODO: DESCRIZIONE CIRCUITI

4.3 Test application software

La struttura principale che tutti i programmi seguono è la seguente:

- `init.c` = è il file sorgente contenente la funzione `Init` che è il punto di partenza dell'eseguibile. E' come se fosse il metodo `main` in Java.
- `init.h` = è un file header che contiene tutte le direttive RTEMS per configurare il sistema
- `task_helper.c` = è il file sorgente contenente la definizione dei task, le funzioni per la manipolazione delle variabili globali, le funzioni aggiuntive
- `task_helper.h` = è un file header che contiene tutte le dichiarazioni delle funzioni definite in `task_helper.c` che si vogliono usare in `init.c`, ad esempio la definizione dei task.

Il programma di test del UART si tratta di un semplice "hello world!" è stato il più semplice da creare, poichè durante il porting è stato usato un programma di esempio simile per visualizzare i log di sistema. Dal codice sorgente dei programmi di esempio, come quelli per il `ticker.img`, si può ipotizzare correttamente che per far stampare un messaggio su log è sufficiente definire la direttiva di sistema

```
#define CONFIGURE_APPLICATION_NEEDS_SIMPLE_CONSOLE_DRIVER
```

e usare le funzioni della libreria standard `stdio.h`

```
printf("log");  
fflush(stdout);
```

L'interfaccia GPIO è stata testata con tre programmi per coprire le seguenti situazioni :

- accensione e spegnimento di un led ad intervalli di 1 secondo.
- alla pressione di un bottone si ha l'accensione o spegnimento di un led.
- alla pressione di un bottone, viene avviato un task.

—

TODO : DESCRIZIONE DEL PRIMO PROGRAMMA

—

TODO: DESCRIZIONE DEL SECONDO PROGRAMMA, DESCRIVENDO
COME È STATA GESTITA LA SEZIONE CRITICA

—

TODO: DESCRIZIONE DEL TERZO PROGRAMMA, DESCRIVENDO LA GESTIONE DEI TASK CON INTERRUPT

—

L'interfaccia I2C è stata testata creando un programma che legge dal SDA l'output del componente ADC che manipolandolo come indicato dal datasheet mi dà come risultato la tensione in entrata al ADC. Poiché bisogna utilizzare un componente aggiuntivo, ho dovuto creare il driver associato.

—

TODO : DESCRIZIONE PROGRAMMA I2C E DEL DRIVER

—

Per testare l'interfaccia SPI è stato creato un programma che legge dal MOSI l'output del componente che dovrebbe rappresentare la tensione in uscita. Anche in questo caso si è dovuto creare il driver associato

—

TODO : DESCRIZIONE PROGRAMMA SPI E DEL DRIVER

—

4.4 Risultati finali

TODO

5 Conclusioni

TODO

Il lavoro svolto sarà utile alla BIS-Italia, fazione italiana della British Interplanetary Society società storica britannica di cui sono membro, che mi ha seguito durante lo stage ed integrerà ciò al progetto di creazione di una replica in scala 1:3 di ExoMars Rover che verrà utilizzato per divulgazione. Tutto il lavoro è stato svolto con l'aiuto dei membri della BIS-Italia e la collaborazione con Microchip.

Ringraziamenti

Bibliografia

- [1] RTEMS Team. *RTEMS User Manual*. URL: <https://docs.rtems.org/branches/master/user/index.html>.
- [2] Giorgio Basile. *RTEMS v4.11 on RaspberryPi*. URL: <https://gist.github.com/giorgiobasile/461d8b8c15d59c6c4445066af6a3124b>.
- [3] Giorgio Basile. *RTEMS v5 on RaspberryPi*. URL: <https://gist.github.com/giorgiobasile/1c1930a8a3ff8e36061cd7f4ef83da95>.
- [4] RTEMS Team. *RTEMS Classic API Guide*. URL: <https://docs.rtems.org/branches/master/c-user/index.html>.
- [5] asuol. *RTEMS rpi testing*. URL: https://github.com/asuol/RTEMS_rpi_testing.
- [6] Microchip. *MCP3425 SOT23-6 Evaluation Board User's Guide*. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/51787a.pdf>.
- [7] Microchip. *16-Bit Analog-to-Digital Converter with I2C Interface and On-Board Reference*. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/22072b.pdf>.
- [8] Microchip. *MCP4802/4812/4822 Data Sheet*. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/20002249B.pdf>.
- [9] Microchip. *10-Pin MSOP and 8-Pin MSOP Evaluation Board User's Guide*. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/50002569A.pdf>.
- [10] Broadcom. *BCM2835 ARM Peripherals*. URL: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf>.