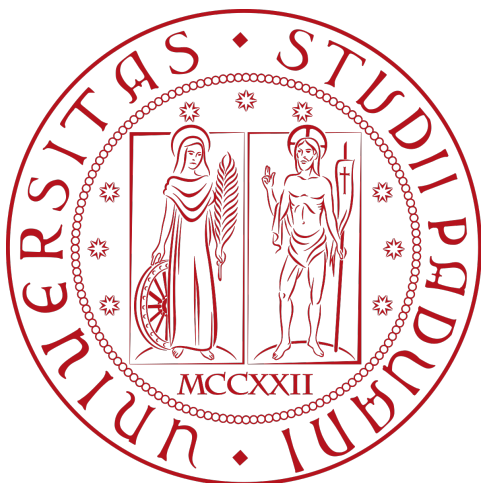




UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Tecnica e Gestione dei sistemi industriali

Corso di Laurea in Ingegneria Meccatronica

Tesi di Laurea

**Programmazione della scheda di sviluppo
dsPICDEM™ MCLV-2 da MATLAB
Simulink**

Relatore:

Ch. mo Prof. Fabio Tinazzi

Laureando:

Simone Gallato

Matricola: 2033466

Anno Accademico 2025-2026

Ringraziamenti

Desidero ringraziare innanzitutto il mio relatore, il Prof. Fabio Tinazzi, per la disponibilità e la guida fornite durante la stesura di questo elaborato. Ringrazio di cuore tutte le persone che mi sono state accanto in questo percorso: mia mamma, mio papà, i miei fratelli, nonna Laura, nonno Franco e i miei amici. Un ringraziamento speciale voglio dedicarlo anche a Maria, che mi è stata vicina emotivamente nelle fasi conclusive di questa triennale e nello sviluppo del progetto di tesi. Un piccolo ringraziamento anche a Luna, che ha dormito sulle mie gambe durante le lunghe sessioni di studio, talvolta riempiendomi di graffi. Questo primo percorso universitario è stato una vera sfida, a dir poco un'odissea, tanto da perdere la rotta più di una volta. Voglio quindi ringraziare me stesso per la dedizione e la perseveranza.

Sommario

Il presente lavoro di tesi affronta la programmazione della scheda di sviluppo dsPIC-DEM™ MCLV-2, equipaggiata con il microcontrollore ARM Cortex-M7 ATSA-ME70Q21B, adottando un approccio interamente Model-Based tramite l'ambiente MATLAB/Simulink. L'obiettivo principale è esplorare e validare metodologie di sviluppo che permettano di gestire la notevole complessità hardware di microcontrollori ad alte prestazioni, solitamente impiegati in contesti industriali, senza la necessità di intervenire manualmente sulla configurazione dei numerosi registri.

Per risolvere tale problematica, è stato implementato un flusso di generazione automatica del codice C/C++ sfruttando le toolchain software integrate di MathWorks (Simulink Coder, Embedded Coder) e di Microchip Technology (MPLAB Device Blocks for Simulink, compilatore XC32 e MPLAB X IDE). Dopo un'analisi dettagliata dell'ecosistema software e dell'architettura hardware – comprendente anche il sistema di programmazione e debug MPLAB ICD-4 – l'approccio è stato applicato concretamente sviluppando specifici modelli Simulink. Dapprima è stato realizzato un algoritmo per l'acquisizione di segnali analogici tramite periferica AFEC e la conseguente generazione di segnali PWM per la modulazione dell'intensità di un LED di debug. Successivamente, è stata configurata e testata un'interfaccia di comunicazione seriale asincrona (RS-232) bidirezionale tra la scheda e un PC host. Al fine di ottimizzare il trasferimento di un'elevata mole di dati senza gravare sui cicli della CPU, la comunicazione è stata supportata dall'approfondimento e dall'implementazione pratica del controller DMA (XDMAc) in modalità Ping-Pong.

I risultati dei test eseguiti in tempo reale dimostrano l'efficienza e la robustezza del codice generato automaticamente, confermando che il paradigma Model-Based consente di configurare le periferiche in modo rapido, intuitivo e con un rischio di errore umano ridotto al minimo. In conclusione, questo lavoro consolida a livello interdisciplinare le conoscenze teoriche sui microcontrollori e sugli azionamenti elettrici, fornendo un'architettura software validata e scalabile che costituirà una solida base di partenza per futuri sviluppi orientati alla prototipazione complessa

nel controllo dei motori.

Indice

1	Introduzione	1
2	Descrizione Software	3
2.1	Il flusso di programmazione	3
3	Descrizione Hardware	7
3.1	Scheda di sviluppo dsPICDEM™ MCLV-2 con 100-Pin Motor Control PIM (MA320203)	7
3.2	Microcontrollore ATSAME70Q21B	12
3.3	Programmatore/Debugger MPLAB ICD-4	13
4	DMA nelle famiglie SAM E70/S70/V70/V71 (approfondimento per il corso di "Microcontrollori e DSP")	17
4.1	Architettura generale di un DMAC	17
4.2	Descrizione XDMAC	19
5	Programmazione della scheda da MATLAB Simulink	25
5.1	Organizzazione e struttura di un codice Simulink per la programmazione della scheda	26
5.2	I passaggi per eseguire la programmazione della scheda	30
6	Comunicazione seriale con la scheda da MATLAB Simulink	33
6.1	La configurazione della periferica UART	34
6.2	Implementazione del lampeggiamento a frequenza modulabile di un led	35
6.3	Implementazione di ricezione e trasmissione seriale nel codice Main	38
6.4	Implementazione di trasmissione e ricezione da seriale nel codice Host	42
6.5	Test della comunicazione seriale	45
7	Conclusioni	47

Elenco delle figure

2.1	Processo di generazione automatica del codice.	4
2.2	Diagramma di flusso della programmazione.	6
3.1	Mappatura da PIM MA320203 a ATSAME70 (Parte 1).	8
3.2	Mappatura da PIM MA320203 a ATSAME70 (Parte 2).	8
3.3	Rappresentazione del 100-pin PIM Header (Fonte: <i>dsPICDEM™ MCLV-2 Development Board User's Guide</i>).	9
3.4	Impostazioni dei jumper (Fonte: <i>dsPICDEM™ MCLV-2 Development Board User's Guide</i>).	10
3.5	Schema a blocchi della scheda MCLV-2 (Fonte: <i>dsPICDEM™ MCLV-2 Development Board User's Guide</i>).	11
3.6	MPLAB ICD-4 logo (Fonte: <i>MPLAB® ICD-4 In-Circuit Debugger User's Guide</i>).	13
3.7	Schema a blocchi di MPLAB ICD-4 (Fonte: <i>MPLAB® ICD 4 In-Circuit Debugger User's Guide</i>).	15
3.8	Diagramma della connessione JTAG/SWD (Fonti: <i>Wurth Elektronik 490107670612S da www.mouser.it</i> , <i>dsPICDEM™ MCLV-2 Development Board User's Guide</i> e <i>ATSAME70 Motor Control PIM Design Documentation</i>).	16
3.9	Connessione ICD-4 con alimentazione esterna del target (Fonte: <i>MPLAB® ICD 4 In-Circuit Debugger User's Guide</i>).	16
4.1	Schema di trasferimento da periferica a memoria tramite DMA . . .	18
4.2	Schema di principio organizzazione con DMAC e periferica I/O (CODEC) (Fonte: <i>Introduzione alle applicazioni industriali di Microcontrollori e DSP</i> di Simone Buso)	18
4.3	Schema a blocchi di XDMAC (fonte: Microchip Technology Inc. <i>SAM E70/S70/V70/V71 Family Data Sheet</i>).	20
4.4	Strutture dei descrittori XDMAC (fonte: Microchip Technology Inc. <i>SAM E70/S70/V70/V71 Family Data Sheet</i>).	23

4.5	Gerarchia di trasferimento in memoria (fonte: Microchip Technology Inc. <i>SAM E70/S70/V70/V71 Family Data Sheet</i>).	23
4.6	Gerarchia di trasferimento verso una periferica (fonte: Microchip Technology Inc. <i>SAM E70/S70/V70/V71 Family Data Sheet</i>).)	24
5.1	Posizionamento dei led di debug nella scheda dsPICDEM™ MCLV-2 (fonte: Microchip Technology Inc., <i>dsPICDEM™ MCLV-2 Development Board User's Guide</i>).	25
5.2	Blocchi Simulink di configurazione.	26
5.3	Modello Simulink dell'acquisizione della tensione del potenziometro, tramite periferica AFEC	28
5.4	Modello Simulink per la Modulazione a larghezza di impulso del led D2, tramite periferica PWM	28
5.5	Parte della sezione "Main" del blocco PWM SAMx7 di MATLAB Simulink (blocco del toolbox PLAB Device Blocks for Simulink: dsPIC, PIC32 e SAM mcu). Si può notare l'indicazione " <i>PWM2max0 = 60000 for 0.002</i> ".	29
5.6	Schermata di MPLAB X IDE per la configurazione generale.	31
5.7	Schermata di MPLAB X IDE per la selezione del compilatore.	31
6.1	MCP2200 USB TO RS232 DEMO BOARD (Fonte: https://www.microchip.com/en-us/development-tool/mcp2200ev-vcv).	33
6.2	Configurazione UART0 per ricezione e trasmissione, tramite blocco "UART Configuration"	35
6.3	Blocchi Simulink: "Data Store Memory" per la creazione delle variabili globali e Digital Output Write.	36
6.4	Codice Simulink per il lampeggiamento a frequenza modulabile da 1 Hz a 10 Hz del led di debug D17.	37
6.5	Lampeggiamento a frequenza variabile di Debug Led D17, con duty cycle fisso al 50%.	38
6.6	Codice Simulink per la ricezione seriale da Host.	39
6.7	Configurazione dei blocchi "Rate Transition".	40
6.8	Codice Simulink per la trasmissione seriale all'host.	41
6.9	Interfaccia del codice host con la quale ci si è interfacciati alla scheda.	43
6.10	Codice per la trasmissione dei dati verso la scheda, tramite porta seriale del PC di host.	44
6.11	Codice per la ricezione del vettore di dati dalla scheda, tramite porta seriale del PC di host.	45

6.12	Test della comunicazione, visualizzato dal blocco "Scope" di Simu- link da codice host.	46
6.13	Codice Simulink per il disegno dell'onda quadra modulabile nello blocco "Scope" del codice host.	46

Capitolo 1

Introduzione

Il presente lavoro di tesi si pone come obiettivo principale la **programmazione di una scheda di sviluppo dsPICDEM™ MCLV-2 mediante un approccio Model-Based**, partendo da un modello implementato in **ambiente MATLAB/Simulink**. Quest'ultimo viene sottoposto a un processo di generazione automatica del codice C/C++ attraverso l'impiego di toolchain software dedicate, tra cui figurano soluzioni proprietarie di **Microchip Technology Inc.** La trattazione si articola inizialmente in una disamina dell'ecosistema software e dell'architettura hardware impiegati, e prosegue successivamente con l'esposizione di un modello Simulink per il lampeggiamento di due LED di debug. L'ultimo capitolo è invece dedicato all'esposizione della configurazione della comunicazione seriale asincrona con la scheda dsPICDEM™ MCLV-2. Il progetto consolida e applica concretamente le metodologie di sviluppo acquisite nel corso di "**Laboratorio di Azionamenti Elettrici**". A completamento del lavoro, l'elaborato include un focus specifico dedicato a una delle tematiche affrontate nell'insegnamento di "**Microcontrollori/DSP**", creando così un forte legame interdisciplinare tra le due materie.

Capitolo 2

Descrizione Software

2.1 Il flusso di programmazione

Entrando un po' più nel dettaglio, l'implementazione di questo progetto si basa sul pacchetto (toolbox) per MATLAB Simulink "MPLAB Device Blocks for Simulink: dsPIC, PIC32 e SAM mcu", ovvero un supporto hardware per Simulink creato per la generazione automatica di codice in C, per sistemi embedded e per la prototipazione rapida nelle famiglie di microcontrollori e DSP prodotte da Microchip Technology: dsPIC (16 bit), PIC32 (32 bit-MIPS) e SAM (32 bit-ARM).

Questa libreria fornisce solo funzioni di tipo model-based, di conseguenza è necessaria l'integrazione con librerie adibite alla vera e propria scrittura automatica di codice in linguaggio C (o C++): Embedded Coder [1], MATLAB Coder [2] e Simulink Coder [3]. In particolare:

- **MATLAB Coder** è un generatore di codice C/C++ a partire da un codice MATLAB, di conseguenza si occupa di tradurre tanto il codice MATLAB presente nel modello Simulink quanto quello presente nelle MATLAB functions;
- **Simulink Coder** invece è un generatore di codice C e C++ a partire da modelli Simulink a tempo discreto, di conseguenza si occupa di tradurre un'architettura grafica nella tipica struttura software (come loop, funzioni, inizializzazioni, ecc.);
- **Embedded Coder** infine si integra con MATLAB e Simulink Coder per generare codice C/C++ compatto e veloce, specifico per sistemi embedded, adatto all'architettura di destinazione.

In questo caso la scheda MCLV-2 utilizzata ospita un microcontrollore SAME70 con architettura ARM Cortex-M7, che dovrà essere impostato come target in fase di scrittura automatica del codice.

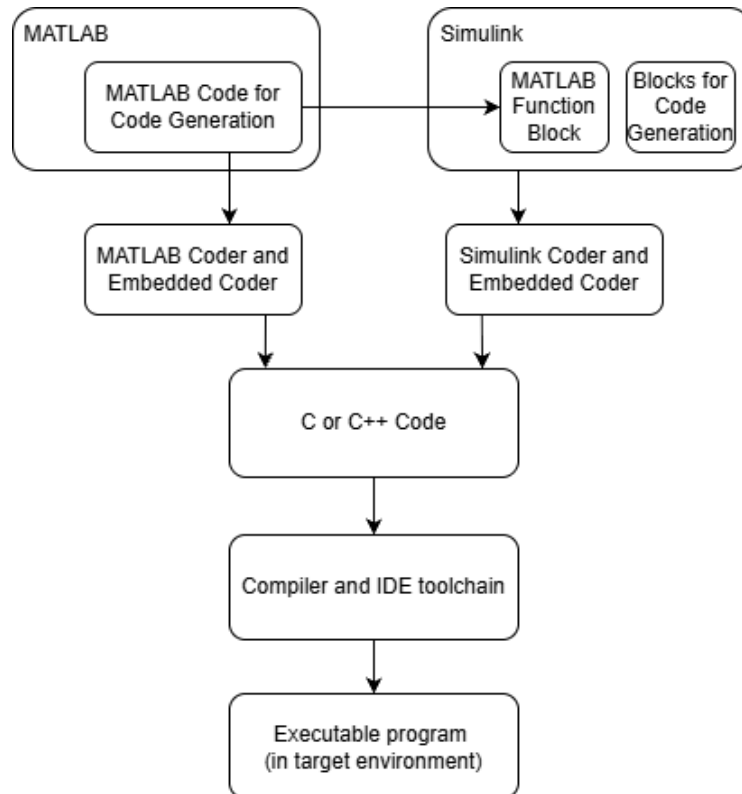


Figura 2.1: Processo di generazione automatica del codice.

A seguito della generazione automatica del codice C/C++ avviene la fase di compilazione, ovvero la traduzione in istruzioni in codice binario che devono essere caricate nella memoria non volatile del microcontrollore/DSP. Esistono vari tipi di compilatori, tutti quanti scaricabili dal sito della Microchip, specifici delle architetture adottabili per la scheda di sviluppo MCLV-2:

- **XC-DSC** se il target è un DSP dsPIC;
- **XC32** se il target è un MCU: PIC32 o SAM.

Per questo progetto di tesi, siccome la scheda di sviluppo ospita un microcontrollore ATSAME70Q21B (con architettura ARM Cortex-M7 a 32 bit), è necessario installare un compilatore di tipo XC32. L'ultimo elemento da installare è l'ambiente di sviluppo integrato MPLAB X IDE, in grado di riconoscere automaticamente il compilatore che si intende utilizzare (in questo caso XC32). Nel caso in cui MPLAB X non riesca a trovarlo, è sufficiente andare ad inserirlo manualmente nella sezione Embedded Tools delle impostazioni. MPLAB X IDE possiede diverse

feature che in questo progetto di tesi non sono state utilizzate, come l'editor di testo, strumenti di debug ecc.; tuttavia, è bastato come software di interfaccia per comunicare con la scheda. L'installazione di MPLAB X IDE è automaticamente accompagnata dall'installazione dell'IPE (Integrated Programming Environment) MPLAB IPE, il quale serve a caricare il file compilato all'interno della memoria di sola lettura del microcontrollore. Quando si preme "Make and Program Device Main Project", MPLAB X IDE attiva inizialmente MPLAB XC COMPILER per la fase di compilazione del codice in C/C++, in seguito attiva MPLAB IPE, che carica il codice nella Flash del microcontrollore attraverso un particolare tool hardware.

Per visualizzare l'esecuzione del codice nella scheda, ci si serve di un altro codice Simulink che, questa volta, viene eseguito interamente nell'ambiente MATLAB Simulink. Tale codice permette di comunicare con la scheda solo se si permette a Simulink di interagire con le periferiche (USB) del computer; per fare ciò, è necessario installare un ulteriore toolbox per Simulink, chiamato: "**Motor Control Blockset**". In particolare, sono sufficienti i blocchi presenti nella sezione "**Protection and Diagnostics**" del toolbox.

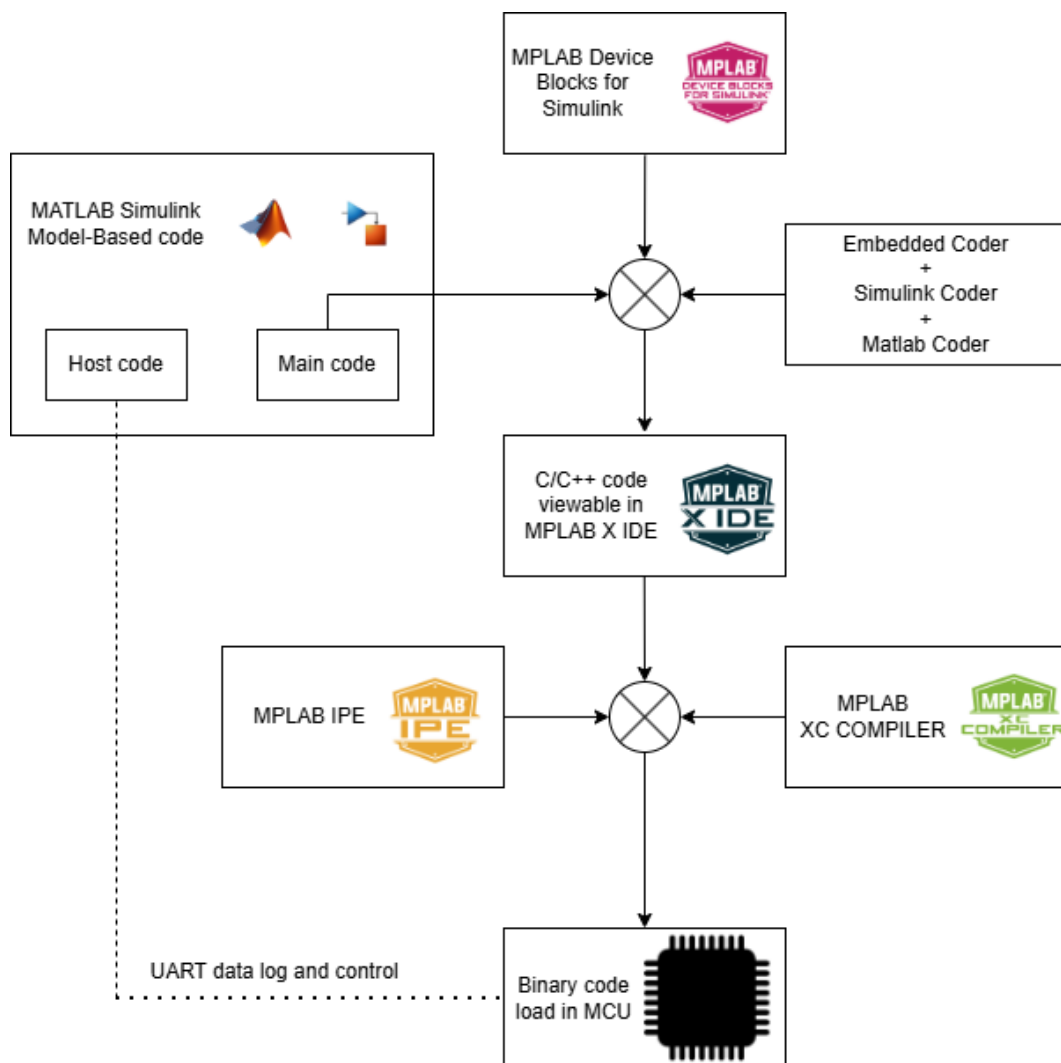


Figura 2.2: Diagramma di flusso della programmazione.

Capitolo 3

Descrizione Hardware

3.1 Scheda di sviluppo dsPICDEM™ MCLV-2 con 100-Pin Motor Control PIM (MA320203)

La scheda di sviluppo **dsPICDEM™ MCLV-2** consiste in un modulo hardware progettato per il controllo di motori a corrente continua senza spazzole (BLDC) o motori sincroni a magneti permanenti (PMSM) in modalità con o senza sensori (sensored/sensorless) [4]. La caratteristica principale di questa scheda è la modularità, ovvero la capacità di ospitare i controllori digitali di segnale (DSC) e i microcontrollori (MCU) della Microchip, come dsPIC33F, PIC24F, dsPIC33E e PIC24E; inoltre, è in grado di ospitare microcontrollori di tipo **SAM**. Tale modularità è garantita da un **PIM (Plug-In Module)**, avente le seguenti funzionalità:

1. **Adattamento del Pinout**, ovvero la mappatura dei pin del microcontrollore verso i pin standard del connettore a 100 pin del PIM;
2. **Componenti di supporto al microcontrollore**, come: il generatore di clock, la gestione dell'alimentazione ed il circuito di reset (è possibile resettare l'esecuzione del microcontrollore da MCLV-2);
3. **Interfacce di programmazione e debug**, ovvero dei connettori dedicati a strumenti di debug;
4. **Gestione dei segnali analogici**, utili per ricevere i segnali di feedback, filtrati o amplificati dalla scheda madre tramite la **Matrix Board**, e per la configurazione degli amplificatori operazionali nel caso in cui si utilizzino quelli esterni (già presenti nella MCLV-2, tramite il connettore **External Op Amp**) o quelli interni disponibili nei dispositivi dsPIC;

5. Utilizzo indicatori e I/O locali a MCLV-2, come i due LED di debug (DBG LED1 e DBG LED2).

Per la programmazione della scheda è di vitale importanza avere sottomano la mappatura risultante dell'adattamento del Pinout del PIM; in particolare, per questo progetto di tesi è stato utilizzato un microcontrollore **ATSAME70Q21B**, di conseguenza è montato un adattatore **ATSAME70 100-Pin Motor Control PIM, MA320203** [5]. Si riporta sotto la relativa mappatura dei pin PIN/MCU, che deriva dal datasheet ufficiale.

Pin PIM	Nome Pin	Funzionalità MCLV2	Pin MCU SAME70	Numero Pin MCU
1	DBG LED2	Debug LED 2	PA24 LED	56
2	VDD	VDD	VDD	5, 30, 43, 80, 72, 96, 134, 143
3	PWM1H3	PWM Output 3H-	PA13 PWMO_WH	42
13	MCLR	Device Master Clear	NRST	83
15	VSS	VSS	VSS	44, 61, 95, 115, 135, 138
16	VDD	VDD	VDD	5, 30, 43, 80, 72, 96, 134, 143
18	FAULT	DC bus Current Fault (active-low)	PD09 FAULT PWM	110
19	TX	UART Transmit	PA02 PFCFLT_PFCEN	93
23	PIM IMOTOR SUM	DC bus current signal	PA20 AFEO CH9_VDC_ISHUNT	22
26	PGC	Device programming clock	PB07 SWCLK	89
27	PGD	Device programming data line	PB06 SWDIO	79
29	PIM REC NEUTR	Motor neutral line voltage	PB03 AFEO CH2 REC NEUTR	31
30	AVDD	Analog supply	AVDD	5, 30, 43, 80, 72, 96, 134, 143
31	AVSS	Analog supply	AVSS	44, 61, 95, 115, 135, 138
32	PIM POT	Potentiometer signal	PB00 AFEO CH10 PIM POT	21
35	PIM_VBUS	DC bus voltage (downscaled)	PA18 AFEO CH7 VBUS; PC15 AFEC1 CH2 VBUS	24, 18
36	VSS	VSS	VSS	44, 61, 95, 115, 135, 138
37	VDD	VDD	VDD	5, 30, 43, 80, 72, 96, 134, 143
41	PIM MONITOR 1	Hall/Current/Voltage feedback	PD30 AFEO CHO_Uph	34
42	PIM MONITOR 2	Hall/Current/Voltage feedback	PA17 AFEO_CH6_Vph	25
43	PIM MONITOR 3	Hall/Current/Voltage feedback	PA21 AFEO CH1 IBUS & PA19 AFEO CHB_IBUS	32, 33
45	VSS	VSS	VSS	44, 61, 95, 115, 135, 138
46	VDD	VDD	VDD	5, 30, 43, 80, 72, 96, 134, 143
47	HALLB	Hall sensor/QEI input	PD10_HALLB & PA01_TIOBO	101, 99
48	HALLC	Hall sensor/QEI input	PD11 HALLC & PA16 TIOZERO	98, 45
49	RX	UART Receive	PA09 URXDO	75
50	TX	UART Transmit	PA10 UTXDO	66
58	PIM_FLT_OUT2	General I/O	1 AFE1 CH6 PFCVACZC_IA; PB02 AFEO CH5 PFCVACZ	14, 26
59	PIM_FLT_OUT1	General I/O	PB01 AFE1 CHO_IPFC_IB; PE04_AFEO CH4_IPFC_IB	20, 27

Figura 3.1: Mappatura da PIM MA320203 a ATSAME70 (Parte 1).

Pin PIM	Nome Pin	Funzionalità MCLV2	Pin MCU SAME70	Numero Pin MCU
60	DBG LED1	Debug LED 1	PC23_LED	127
62	VDD	VDD	VDD	5, 30, 43, 80, 72, 96, 134, 143
65	VSS	VSS	VSS	44, 61, 95, 115, 135, 138
68	LIN CS	LIN Chip Select signal	PE02 BTN	7
75	VSS	VSS	VSS	44, 61, 95, 115, 135, 138
78	CAN RX	CAN Receive	PD00 PFC PWM	1
80	HALLA	Hall sensor/QEI input	PD13 HALLA & PA00 TIOAO	88, 102
83	BTN 1	Push-button - S2 input	PC03 BTN1	40
84	BTN 2	Push-button S3 input	PC01 BTN2	38
86	VDD	VDD	VDD	5, 30, 43, 80, 72, 96, 134, 143
87	CAN RX	CAN Receive	PD28 CAN RX	71
88	CAN TX	CAN Transmit	PD12 CAN TX	92
93	PWM1L1	PWM Output-1L	PD24 PWMCO_UL	55
94	PWM1H1	PWM Output-1H	PA11 PWMCO_UH	64
98	PWM1L2	PWM Output-2L	PD25 PWMCO_VL	52
99	PWM1H2	PWM Output-2H	PA12 PWMCO_VH	68
100	PWM1L3	PWM Output-3L	PD26 PWMCO_WL	53

Figura 3.2: Mappatura da PIM MA320203 a ATSAME70 (Parte 2).

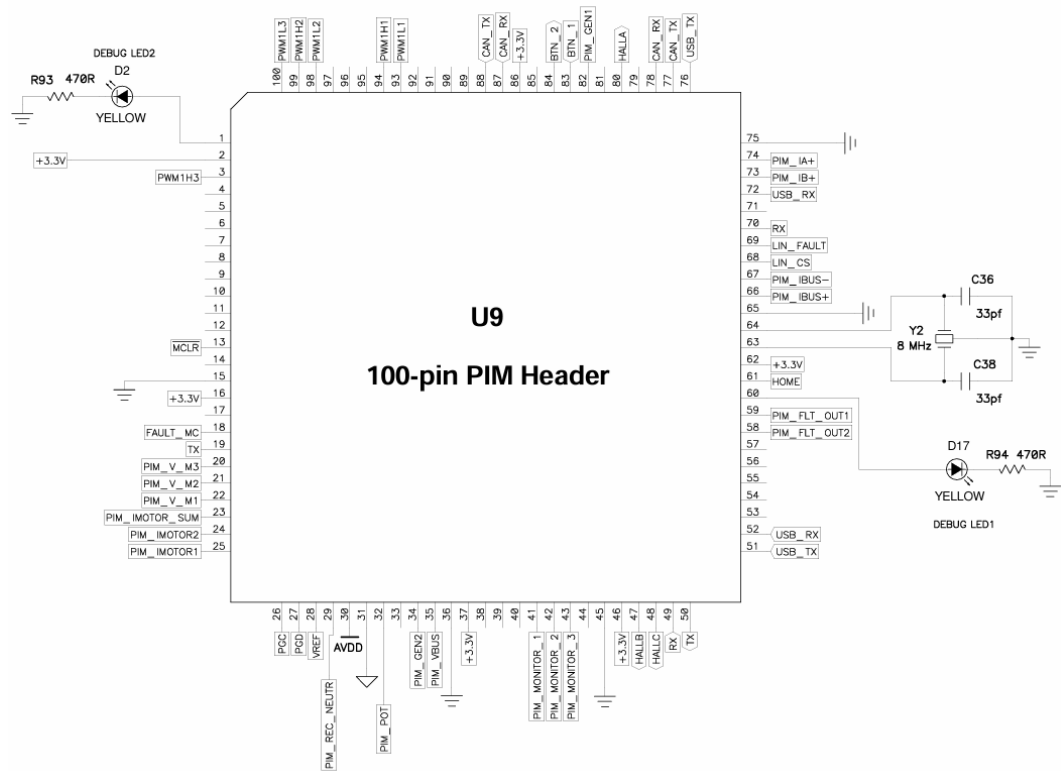


Figura 3.3: Rappresentazione del 100-pin PIM Header (Fonte: *dsPICDEM™ MCLV-2 Development Board User's Guide*).

I principali elementi che compongono la scheda di sviluppo **dsPICDEM MCLV-2** sono:

1. Un **Inverter trifase**, avente una potenza nominale di 48V/15A;
2. Interfacce per **sensori ad effetto Hall** ed **encoder in quadratura**; inoltre è presente un feedback per la tensione di fase utile al controllo sensorless e **misure di corrente** tramite resistori di shunt;
3. **Porte di comunicazione**: CAN, LIN, USB e RS-232 per l'interfacciamento con l'esterno e **porte di programmazione** ICSP e RJ11 dedicate a programmatori Microchip;
4. Il sistema **PIM** a 100 pin e la **Matrix Board** per la configurazione di amplificatori operazionali esterni al microcontrollore adottato;
5. **Controlli manuali** tramite: un potenziometro da 10k Ω , due pulsanti utente configurabili e un pulsante di reset della routine in esecuzione sul microcontrollore;
6. **Indicatori LED** dedicati a: debug, allo stato delle uscite PWM e alla segnalazione dell'alimentazione;
7. Un **ingresso flessibile** adatto ad alimentazioni dai 24 V ai 48 V e i regolatori di tensione interni per l'alimentazione dei vari dispositivi presenti nella scheda.

Un'ulteriore funzionalità della scheda di sviluppo dsPICDEM MCLV-2 consiste nella possibilità di configurare fisicamente alcune impostazioni attraverso dei jumper. In particolare, per questo progetto di tesi è importante che siano posizionati nei pin 7 e 8 dei jumper JP4 e JP5 per **assegnare la periferica UART alle linee di trasmissione e ricezione**^[4].

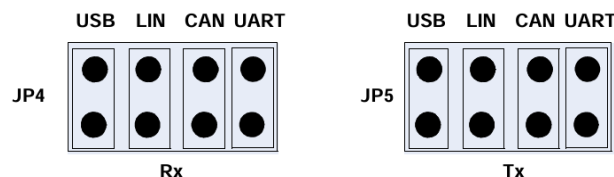


Figura 3.4: Impostazioni dei jumper (Fonte: *dsPICDEM™ MCLV-2 Development Board User's Guide*).

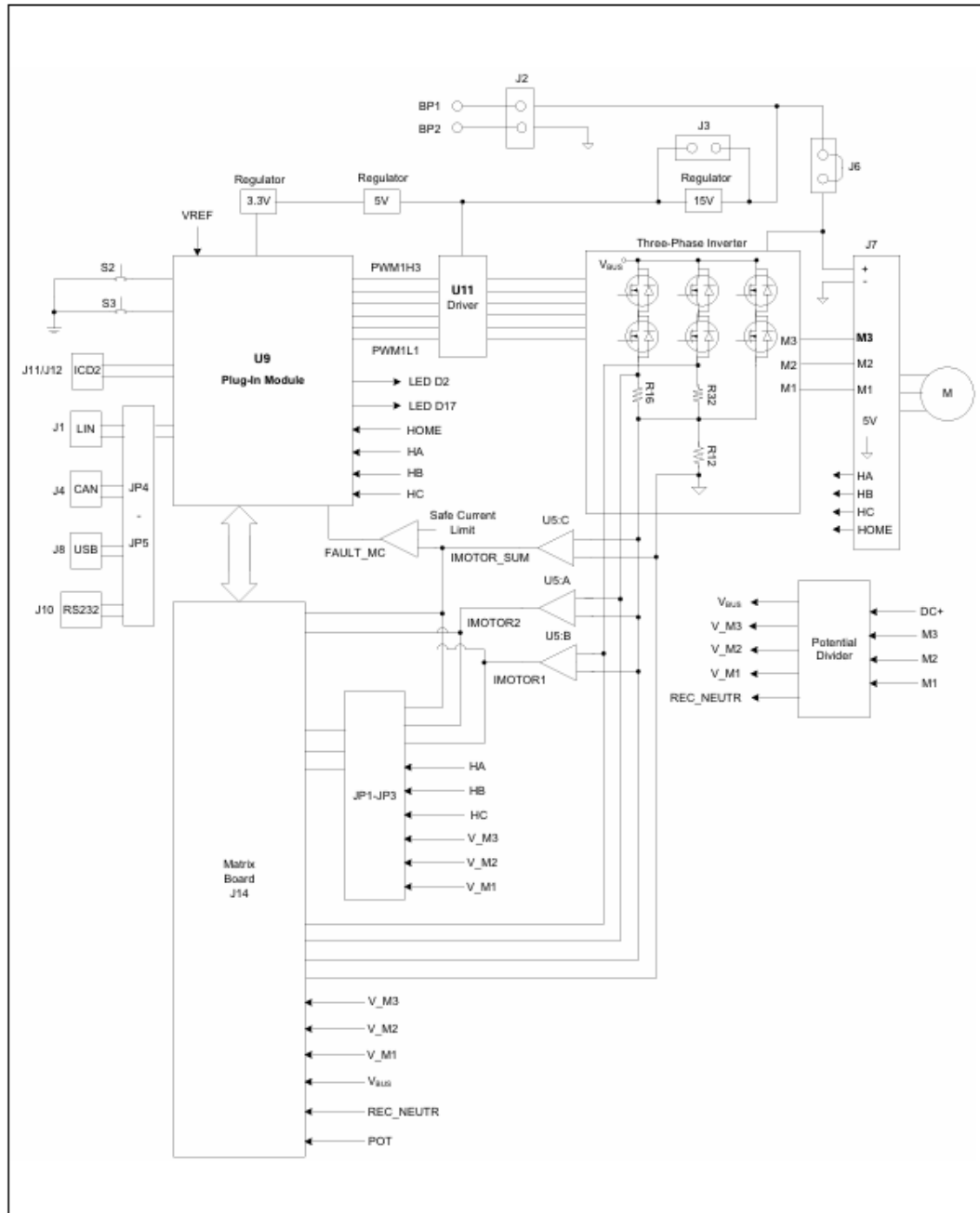


Figura 3.5: Schema a blocchi della scheda MCLV-2 (Fonte: *dsPICDEM™ MCLV-2 Development Board User's Guide*).

3.2 Microcontrollore ATSAME70Q21B

Come già accennato, il microcontrollore che alloggia nella scheda di sviluppo è un **ATSAME70Q21B**, appartenente alla serie SAM E70 basata su core ARM Cortex-M7 ad alte prestazioni. Di seguito si riportano le principali caratteristiche estratte dal datasheet ufficiale:

1. **Processore** ARM Cortex-M7 operante a una frequenza massima di **300 MHz**;
2. **FPU** per singola e doppia precisione;
3. Dispone di una **I-Cache da 16 Kb per le istruzioni** e una **D-Cache da 16 Kbyte per i dati** con un ECC (Error Code Correction) per eventuali corruzioni singole dei bit;
4. Presenta un'**Unità di Protezione della Memoria (MPU)** adibita all'assegnazione di regole d'accesso alla memoria tramite la definizione di aree critiche, oppure la definizione di "aree a sola lettura";
5. Include un set di istruzioni **DSP**;
6. Possiede: una **Flash da 2 MB**, una **Multi-port SRAM da 386 Kb** (permette l'accesso multiplo di: CU, DMA (XDMAC) e controller USB), una **ROM da 16 KB** e una **Backup RAM (BRAM)** adibita al salvataggio temporaneo di dati critici quando l'alimentazione viene a mancare;
7. Per la conversione analogico-digitale, e viceversa, possiede: due **ADC a 12 bit (AFEC: Analog Front-End) con campionamento massimo da 1.7 Msps** e un **DAC a 12 bit** con due canali e **1 Msps** per canale;
8. Possiede: **quattro moduli TC a 16 bit** con modalità: **Input Capture, Waveform Generation e PWM**; inoltre possiede **due PWM a 4 canali e 16 bit con uscite complementari** (PWMHx e PWMLx, con un Dead Time Generation hardware) utili al controllo dei motori tramite i convertitori DC-DC e l'inverter;
9. Infine sono presenti periferiche adibite alla connettività con l'esterno, in particolare: **Ethernet, USB** con DMA dedicato, **CAN-FD** e un'**interfaccia per sensori di immagine**. Inoltre presenta diverse interfacce per la **Comunicazione Seriale**: tre **USART**, tre interfacce per **I2C**, una per **QSPI** e cinque periferiche **UART a 2 fili**.

Un'altra caratteristica dell'ATSAME70Q21B consiste nell'essere un **modello "Q"**, quindi dispone di **144 pin con capacità di interrupt esterno** e resistore di terminazione on-die (per sopprimere ai disturbi di riflessione dovuti alle alte frequenze) [6].

3.3 Programmatore/Debugger MPLAB ICD-4

Un altro elemento fondamentale per l'attuazione di questo progetto di tesi è il **programmatore/debugger**, in particolare è stato utilizzato **MPLAB ICD-4**. MPLAB ICD-4 è uno strumento di programmazione e debug, sviluppato per i microcontrollori della Microchip e con architettura ARM Cortex-M7, il cui ruolo consiste nel caricare il codice in memoria e testarne l'esecuzione (debug) in tempo reale sull'hardware. Per questo progetto di tesi non è stato necessario utilizzare gli strumenti adibiti al debugging di MPLAB ICD-4, tuttavia è importante spiegare come avviene la programmazione tramite esso, così come è riportato nel datasheet [7].



Figura 3.6: MPLAB ICD-4 logo (Fonte: *MPLAB®ICD-4 In-Circuit Debugger User's Guide*).

Tutto parte da MPLAB X IDE, la quale trasferisce il codice nella ROM del microcontrollore tramite un protocollo di comunicazione; in particolare, per i Cortex permette di scegliere tra i protocolli **JTAG** e **SWD**. Poi, tramite una **Debugger Adapter Board AC1022015**, avviene la connessione tra MPLAB ICD-4 e il microcontrollore tramite l'interfaccia JTAG/SWD presente nel modulo plug-in **100-Pin Motor Control PIM (MA320203)**.

Siccome nel ATSAME70 non è presente nativamente un bootloader che carichi il firmware all'interno della sua memoria Flash, è presente un'interfaccia di programmazione JTAG/SWD; in particolare è possibile selezionare uno dei due protocolli da MPLAB X IDE. MPLAB ICD-4 funziona grazie a un microcontrollore ATSAME70 (esattamente lo stesso che alloggia nella scheda MCLV-2) e a un **FP-GA che permette di selezionare diversi protocolli di programmazione** (le architetture native Microchip utilizzano ICSP, In-Circuit Serial Programming). La comunicazione con l'esterno avviene tramite un'uscita RJ-45 alla quale è possibile collegare un cavo RJ-45 o RJ-11 (quest'ultimo **non supporta il protocollo JTAG/SWD**, quindi con esso non si riesce a programmare la scheda). Il cavo RJ-45 poi deve essere connesso all'adattatore AC1022015 e da lì connesso al PIM tramite

un cavo a nastro da 10-Pin [7].

Il **flusso di programmazione operativo** di MPLAB ICD-4 ha le seguenti fasi[7]:

1. Fase di **Erase All Before Program**: la memoria di sola lettura del microcontrollore target viene completamente cancellata per poi essere riprogrammata;
2. Fase di **Write**: l'immagine del firmware, che precedentemente è stata scaricata nella memoria SRAM dell'ICD-4, viene inviata serialmente nella Flash del SAME70, tramite l'interfaccia SWD/JTAG. Nel caso in cui si intenda avviare una sessione di debug-in-circuit, il programmatore inietta in memoria una porzione di codice detta "debug executive"; tale codice programma il microcontrollore in modo che comunichi in un certo modo con il debugger;
3. Fase di **Verify**: in questa fase viene assicurata la bontà dell'operazione appena eseguita, i dati appena scritti vengono letti indietro dal microcontrollore e confrontati con l'immagine del firmware in memoria;
4. Fase di **Release/Run**: una volta terminato il flusso di programmazione, il programmatore disabilita internamente i registri di in-circuit debug della scheda e rilascia il segnale di reset. Da questo momento in poi il microcontrollore inizia l'esecuzione autonoma e ciclica del firmware in modalità Release.

La comunicazione tra ICD-4 e PC avviene tramite un'interfaccia USB 2.0 High-Speed (480 Mbps), mentre l'alimentazione del target può avvenire in due modi[7]:

- **Alimentazione attraverso il target (metodo raccomandato e utilizzato in questo progetto di tesi)**: in questo modo l'ICD-4 campiona semplicemente la tensione del target tramite un apposito pin;
- **Alimentazione dal programmatore**: ovvero che, fornendo al programmatore un'alimentazione esterna da 9V DC, fornisce direttamente alimentazione al target.

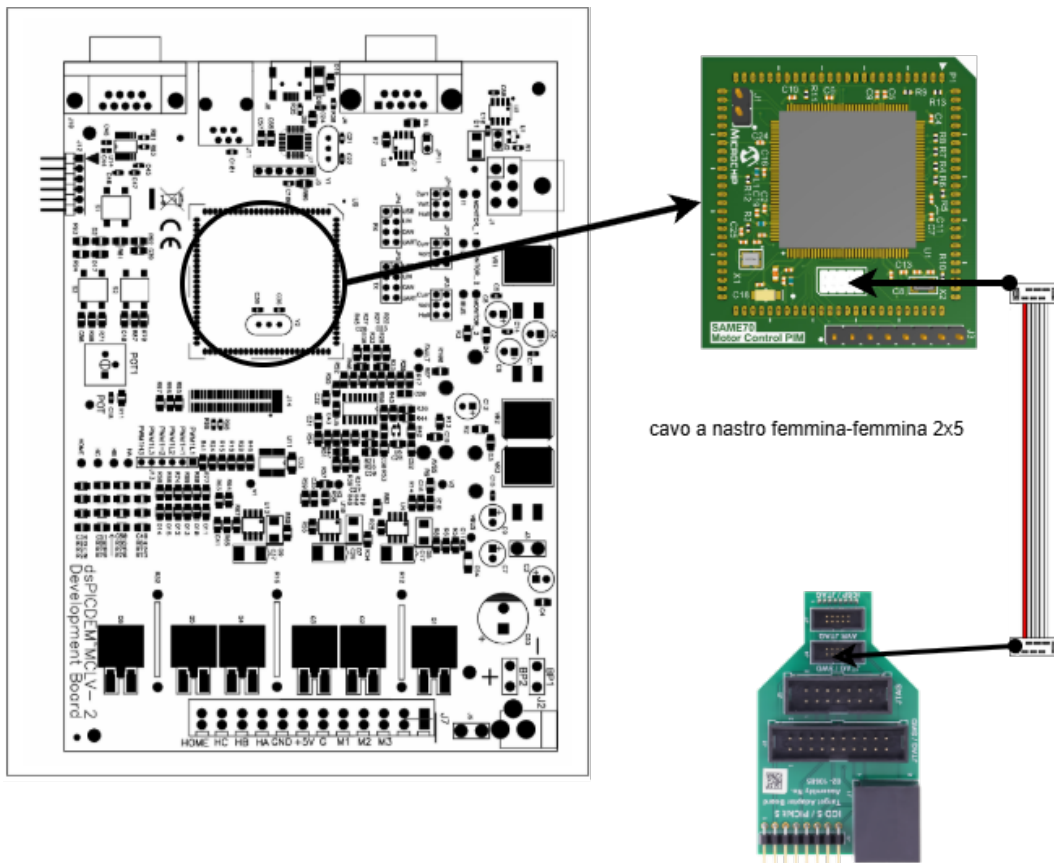


Figura 3.8: Diagramma della connessione JTAG/SWD (Fonti: *Wurth Elektronik 490107670612S* da www.mouser.it, *dsPICDEM™ MCLV-2 Development Board User's Guide* e *ATSAME70 Motor Control PIM Design Documentation*).

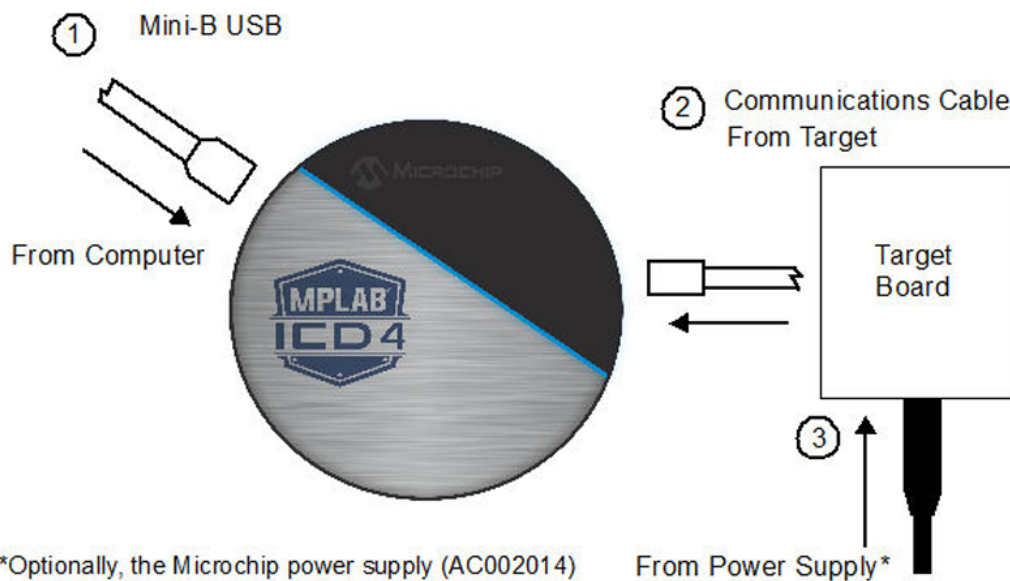


Figura 3.9: Connessione ICD-4 con alimentazione esterna del target (Fonte: *MPLAB® ICD 4 In-Circuit Debugger User's Guide*).

Capitolo 4

DMA nelle famiglie SAM E70/S70/V70/V71 (approfondimento per il corso di "Microcontrollori e DSP")

In questo capitolo si esegue un approfondimento sul DMA, in quanto è stato adottato nel codice test per attuare la trasmissione seriale dei dati in uscita dall'algoritmo caricato nel microcontrollore. Sia il Main Code che l'Host Code vengono descritti dettagliatamente nei capitoli 5 e 6.

4.1 Architettura generale di un DMAC

In generale un **DMA**, acronimo di **Direct Memory Access**, consiste in una tecnica hardware adibita ai trasferimenti di dati tra altre periferiche e memoria (oppure tra aree di memoria diverse), coinvolgendo il meno possibile la CPU. L'implementazione di tale tecnica avviene tramite il **DMAC (Direct Access Memory Controller)**, che consiste in un'**unità di controllo** adibita allo svolgimento di questa potente funzione di supporto.

La funzione del DMAC consiste nel sostituirsi alla CPU nel controllo dei **bus di dati e indirizzi**, attraverso degli opportuni **deviatori** che impediscono l'utilizzo simultaneo dei bus. In linea di principio, il funzionamento della trasmissione diretta tra periferica I/O e memoria è il seguente[8]:

1. La periferica I/O invia una **Transfer Request** al DMAC;

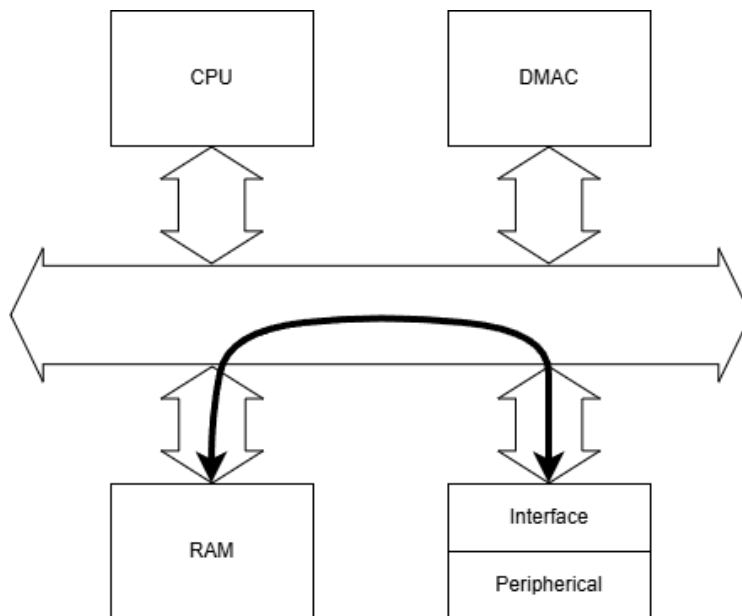


Figura 4.1: Schema di trasferimento da periferica a memoria tramite DMA

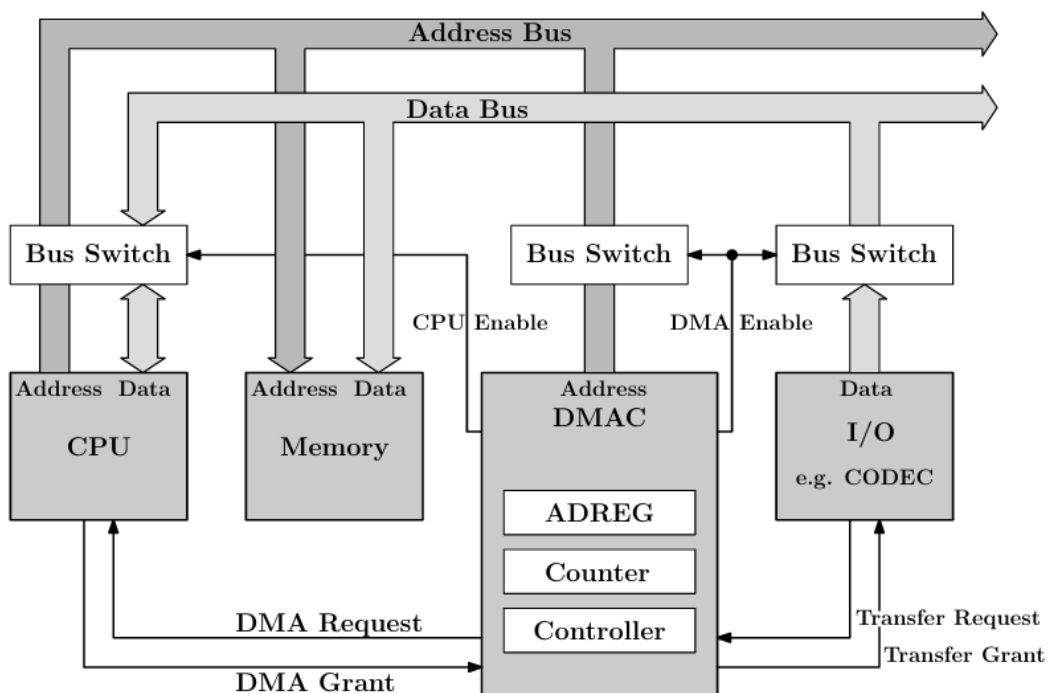


Figura 4.2: Schema di principio organizzazione con DMAC e periferica I/O (CODEC) (Fonte: *Introduzione alle applicazioni industriali di Microcontrollori e DSP* di Simone Buso)

2. Il DMAC invia alla CPU una **DMA Request**, richiedendo quindi il controllo dei bus per un trasferimento dati;
3. A un certo punto la CPU concede il trasferimento autonomo di dati emettendo un **DMA Grant**, quindi il DMAC abilita i flag per la commutazione dei deviatori (Bus Switch) e si impossessa del controllo dei bus;
4. Infine il DMAC imposta l'indirizzo di memoria sul bus degli indirizzi (precedentemente salvato in ADREG, Address Register) e autorizza la periferica I/O a leggere o scrivere il dato in memoria, tramite un segnale di **Transfer Grant**.

In genere, il trasferimento di dati tra periferica e memoria può avvenire con due principali modalità:

- in modalità **burst**, ovvero dove la fase di *Transfer Grant* viene ripetuta tante volte quanti sono i dati contenuti in un certo blocco dati, quindi il **trasferimento avviene per blocchi**;
- in modalità **cycle stealing**, invece, il DMAC trasferisce un **numero di dati sufficiente a non precludere alla CPU l'accesso ai bus di dati e indirizzi**, quindi si hanno *Transfer Grant* intermittenti.

4.2 Descrizione XDMAC

Tutti i microcontrollori delle famiglie SAM E70/S70/V70/V71, che comprendono ATSAME70Q21B1, implementano questa tecnica hardware tramite il controller DMAC: **XDMAC**. In questi microcontrollori, l'XDMAC è connesso ai vari client attraverso una fitta rete di bus (12-layer Bus Matrix AHB, Advanced High-performance Bus) e permette la **configurazione indipendente di 24 canali** (ogni linea di trasmissione/ricezione è identificata da un numero da 0 a 51)[6]. Nel complesso, l'unità di controllo DMA, XDMAC, è composta fondamentalmente dai seguenti elementi[6]:

- **Periferica Sorgente**, ovvero il dispositivo client (o slave) mappato in memoria sulla rete di interconnessione, da cui l'XDMAC svolge **l'operazione di lettura dei dati** non appena la periferica attiva la richiesta di trasferimento (*transfer request*);
- **Periferica di Destinazione**, cioè il dispositivo client, sempre mappato in memoria sulla rete di interconnessione, su cui l'XDMAC svolge **l'operazione di scrittura dei dati**, successivamente alla richiesta di trasferimento;

- **Canale logico (Channel)**, ovvero il movimento di dati tra la periferica sorgente e quella di destinazione. Esso è composto da uno **stack FIFO**, utile al disaccoppiamento delle operazioni di lettura e scrittura, e da due **FSM** (Finite State Machines: **Source** e **Destination**), che si occupano indipendentemente delle operazioni di lettura e scrittura con i client;
- **Tipo di trasferimento**, che solitamente è sincronizzato via hardware se è attivato dalla richiesta di trasferimento della periferica, oppure auto-attivato nel caso del trasferimento tra aree di memoria. Tale impostazione viene decisa tramite la scrittura degli **Status Registers** e **Configuration Registers**, all'interno dell'interfaccia **APB (Advanced Peripheral Bus Interface)** direttamente connessa al processore in fase di configurazione.

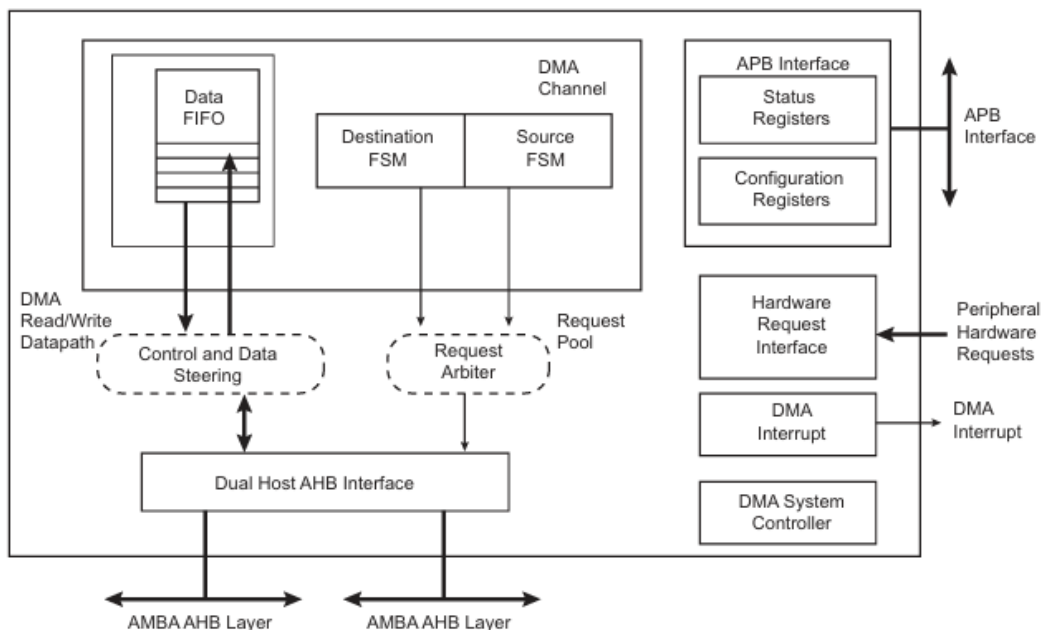


Figura 4.3: Schema a blocchi di XDMAC (fonte: Microchip Technology Inc. *SAM E70/S70/V70/V71 Family Data Sheet*).

Il DMA Channel è connesso poi al Bus Matrix tramite la **Dual Host AHB Interface**; infatti l'XDMAC possiede due interfacce AHB indipendenti (AHB_IF0 e AHB_IF1), permettendo così di leggere e scrivere dati simultaneamente. Poiché l'XDMAC è composto da 24 canali che possono richiedere insieme un trasferimento di dati, è presente un **Request Arbiter** programmabile, con il quale è possibile stabilire il grado di priorità dei vari canali in fase di configurazione.

Un'altra importante caratteristica dell'XDMAC consiste nella **gerarchia di trasferimento dei dati in memoria o dalla memoria**; infatti esso adotta una strategia

per gestire autonomamente (senza l'intervento della CPU, o almeno, il meno possibile) il **trasferimento di un'elevata mole di dati**, come, ad esempio, il trasferimento in memoria dei dati letti da un sensore d'immagine tramite la periferica ISI (Image Sensor Interface). La struttura base che viene adottata per ogni trasferimento è la seguente[6]:

- **XDMAC Host Transfer**: consiste nel livello gerarchico più alto ed è composto da una **lista concatenata di blocchi (Linked List)**. Esso rappresenta l'intera operazione di trasferimento, come, ad esempio, lo spostamento dei dati dalla memoria alla periferica UART, dove i singoli blocchi sono un vettore di dati da trasmettere.
- **XDMAC Block**: come detto prima, i blocchi compongono una lista concatenata (monodirezionale), dove ogni singolo blocco contiene l'indirizzo di memoria di quello successivo. Di per sé i blocchi sono solo un'astrazione di come vengono organizzati i dati in memoria, infatti sono il risultato delle istruzioni scritte precedentemente dal programmatore e caricate in RAM. Queste istruzioni sono dette **Descrittori** e consistono in un vettore di elementi (**Array** o **Struct**) che può avere **quattro forme differenti** a seconda della complessità:
 1. **View 0**: descrittore base, che permette di modificare l'indirizzo di origine o destinazione (MBR_TA);
 2. **View 1**: descrittore con doppio indirizzo, Source Address (MBR_SA) e Destination Address (MBR_DA), utile per i trasferimenti tra aree di memoria;
 3. **View 2**: descrittore che permette anche il cambio di configurazione del canale XDMAC (MBR_CFG), come la dimensione del dato da trasmettere o quella del microblocco;
 4. **View 3**: descrittore con controllo dell'intero blocco (MBR_BC) e tre registri stride (MBR_DS, MBR_SUS, MBR_DUS), utili ad impostare salti di memoria personalizzati.

Ogni volta che viene terminata la trasmissione di un blocco di dati, l'XDMAC estrae l'indirizzo di memoria del descrittore successivo da quello precedente, legge dalla RAM il prossimo descrittore ed aggiorna i registri di configurazione per la trasmissione del nuovo blocco. **La trasmissione dati inizia con la lettura dell'indirizzo del primo descrittore, che risiede in XDMAC_CNDAx** (Next Descriptor Address Register, sovrascritto dalla

CPU in fase di configurazione iniziale), mentre è **conclusa quando l'XDMAC legge 0 nel puntatore al descrittore successivo (terminatore della trasmissione)**.

- **XDMAC Microblock:** il microblocco invece consiste in un **raggruppamento programmabile di dati** utile a organizzare un intero blocco in memoria, in quanto permette di aggiornare dinamicamente e in modo condizionale gli indirizzi di memoria alla fine di una determinata quantità di dati scambiati. Tali impostazioni vengono salvate nei **registri di stride**.
- **XDMAC Chunk/Burst:** in questo livello avviene la **scomposizione del microblocco in una quantità di dati fissa e prestabilita**, così da attuare la trasmissione fisica del dato attraverso uno dei due layer AHB (AHB_IF0 o AHB_IF1). Questa trasmissione può avvenire con due meccanismi differenti a seconda delle esigenze:
 1. **Memory Burst**, ovvero una trasmissione dati **verso la memoria**. Tale meccanismo è necessario a rendere il più efficiente possibile il processo di lettura/scrittura della memoria tramite XDMAC, infatti i dati del microblocco vengono trasferiti con vari Burst (raffiche);
 2. **Peripheral Chunk**, ovvero una trasmissione dati **verso una periferica**. Questo meccanismo consiste nel suddividere il microblocco in Chunk e di regolarne il trasferimento con un segnale di sincronismo (*Transfer Request*) proveniente dalla periferica di destinazione (o partenza). Questo metodo risulta di vitale importanza, in quanto l'XDMAC deve essere il più possibile sincronizzato con il ciclo di lavoro delle periferiche per poter attuare una trasmissione efficiente dei dati.

Per definire la dimensione di un singolo Burst/Chunk è necessario innanzitutto stabilire la dimensione dell'XDMAC Microblock, attraverso il campo UBLLEN del registro XDMAC_CUBC (Channel Microblock Control Register). In seguito, si indica nel registro XDMAC_CC (Channel Configuration Register) quanti dati raggruppare per ogni singola operazione sul bus compilando i campi: CSIZE per impostare la dimensione del Chunk e MBSIZE per impostare la dimensione del Burst.

Channel Next Descriptor	Offset	Structure member	Name
View 0 Structure	DSCR_ADDR+0x00	Next Descriptor Address Member	MBR_NDA
	DSCR_ADDR+0x04	Microblock Control Member	MBR_UBC
	DSCR_ADDR+0x08	Transfer Address Member	MBR_TA
View 1 Structure	DSCR_ADDR+0x00	Next Descriptor Address Member	MBR_NDA
	DSCR_ADDR+0x04	Microblock Control Member	MBR_UBC
	DSCR_ADDR+0x08	Source Address Member	MBR_SA
	DSCR_ADDR+0x0C	Destination Address Member	MBR_DA
View 2 Structure	DSCR_ADDR+0x00	Next Descriptor Address Member	MBR_NDA
	DSCR_ADDR+0x04	Microblock Control Member	MBR_UBC
	DSCR_ADDR+0x08	Source Address Member	MBR_SA
	DSCR_ADDR+0x0C	Destination Address Member	MBR_DA
	DSCR_ADDR+0x10	Configuration Register	MBR_CFG
View 3 Structure	DSCR_ADDR+0x00	Next Descriptor Address Member	MBR_NDA
	DSCR_ADDR+0x04	Microblock Control Member	MBR_UBC
	DSCR_ADDR+0x08	Source Address Member	MBR_SA
	DSCR_ADDR+0x0C	Destination Address Member	MBR_DA
	DSCR_ADDR+0x10	Configuration Member	MBR_CFG
	DSCR_ADDR+0x14	Block Control Member	MBR_BC
	DSCR_ADDR+0x18	Data Stride Member	MBR_DS
	DSCR_ADDR+0x1C	Source Microblock Stride Member	MBR_SUS
	DSCR_ADDR+0x20	Destination Microblock Stride Member	MBR_DUS

Figura 4.4: Strutture dei descrittori XDMAC (fonte: Microchip Technology Inc. *SAM E70/S70/V70/V71 Family Data Sheet*).

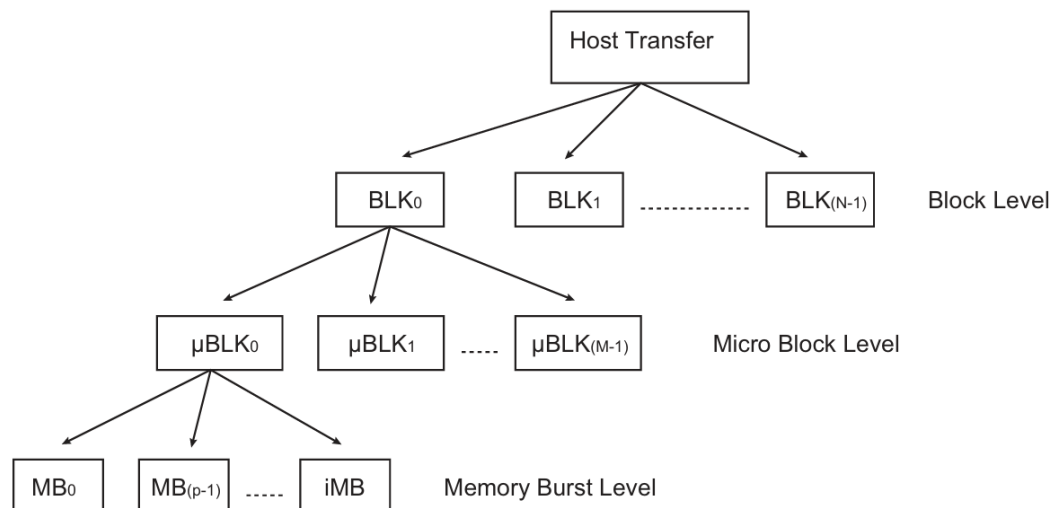


Figura 4.5: Gerarchia di trasferimento in memoria (fonte: Microchip Technology Inc. *SAM E70/S70/V70/V71 Family Data Sheet*).

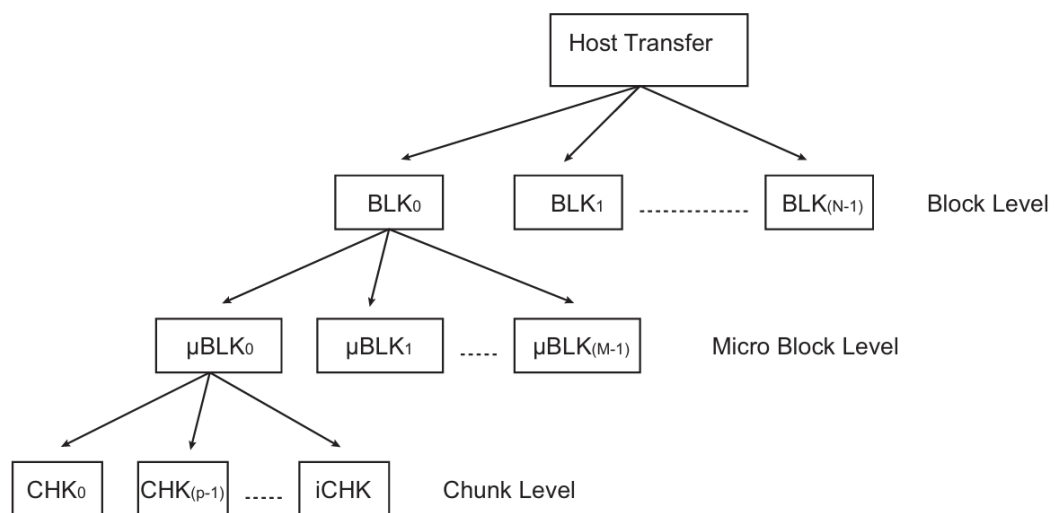


Figura 4.6: Gerarchia di trasferimento verso una periferica (fonte: Microchip Technology Inc. *SAM E70/S70/V70/V71 Family Data Sheet*.)

Programmazione della scheda da MATLAB Simulink

25

5.1 Organizzazione e struttura di un codice Simulink per la programmazione della scheda

Il paradigma di programmazione che è stato adottato per la scrittura di questo codice non si discosta molto dal metodo convenzionale di stesura di un codice Simulink.

L'unica differenza risiede nel

blocco di configurazione, solitamente non presente nei codici Simulink, dove sono posizionati: i vari blocchi di **configurazione delle periferiche adottate**, le **opzioni di compilazione** e il **Master**.

MCHP_MASTER
SAME70Q21B - LQFP144
300MHZ

Compiler-Version
XC32 - v4.60

UART 0 Config
230400 (-0.76%)
Bytes / Step: 1.14

L'impostazione del blocco "UART 0 Config" viene trattata nel capitolo 6 di questo progetto di tesi, mentre:

Figura 5.2: Blocchi Simulink di configurazione.

- Nel blocco **Microchip_MASTER** viene selezionato il modello **SAME70Q21B**, con case di tipo **LQFP144** (Low-profile Quad Flat Pack), da **300MHZ**;
- Nel blocco **Compiler-Version** è stato selezionato **XC32** (per microcontrollori a 32 bit) con l'ultima versione disponibile.

Per quanto riguarda le altre impostazioni dei due blocchi, sono state lasciate quelle di default.

Il primo algoritmo di test della scheda implementa la **modulazione dell'intensità luminosa del LED di debug D2**, tramite il **potenziometro POT**. Per fare ciò, si utilizzano: la periferica **AFEC (Analog Front-End Controller)** per acquisire il segnale del potenziometro e la periferica **PWMC (Pulse Width Modulation Controller)** per l'alimentazione di D2, con duty cycle ricavato dalla tensione ripartita in uscita dal potenziometro. La configurazione di entrambe le periferiche è estremamente semplificata dall'interfaccia Simulink, infatti risulta analoga alla configurazione di un qualsiasi blocco:

1. **Configurazione AFEC**: nella sezione *Analog Channels* sono stati selezionati:

- **AFE Controller 0** in "*Select ADC (AFEC) Controller*";
- **uint16** in "*Single-Ended conversion output are*" e "*Differential conversions output are*";
- in "*AFEC seq 0*" è stato selezionato **AD10/PB0** (canale 10 dell'ADC e pin PB0 del microcontrollore);
- il "*Block output sample time*" è stato impostato pari al periodo di modulazione (**Tpwm = 0.001 s**).

Nella sezione *Options* è stata selezionata la **modalità Free Run**, inoltre è stata impostata una *Input impedance* < 5 k Ω . Quest'ultima impostazione è molto importante, siccome dà un'indicazione al Master Block del tempo di campionamento da adottare. **Infatti con tale impedenza: Tsamp=400 ns e Tconv=213.33 ns**. L'impedenza di uscita del potenziometro POT risulta non superare mai i 2.5 k Ω , ciò si può dimostrare analizzando lo schema elettrico del potenziometro dal datasheet della scheda di sviluppo dsPICDEM™ MCLV-2[4].

2. **Configurazione PWM:** nella sezione "*Main*" del blocco, sono stati selezionati:

- **PWM Controller 0** in "*Select Pulse Width Modulation (PWM) Controller*";
- in "*Max period (s)*", un **periodo massimo di modulazione pari al doppio di quello che verrà impostato** ($2 \times T_{pwm} = 0.002$ s); in questo modo il master imporrà il **prescaler** e il **limite di conteggio del contatore (a 16 bit)** in maniera automatica. In particolare, sceglierà il prescaler in funzione di "*Max period*", così che in tale periodo il contatore termini il conteggio; poi calcola il limite del conteggio a partire dal periodo di modulazione Tpwm desiderato per la portante;
- per fare in modo che il segnale modulato raggiunga il LED di debug D2, bisogna selezionare il canale giusto. In "*PWMHI*" (PWM High channel 1) è stato scelto **PortA24/Pin[56]**.

Nella sezione *Initial Values* è stato inserito **Tpwm**, mentre in *Block Inputs* sono stati selezionati: **Independent Scalar** nella sottosezione *Block input format*, **a Floating Point input with a physical scaling in (s)** in *Period*, **a Uint raw point** in *Duty Cycle* e **Not a block input** su *Dead Time*.

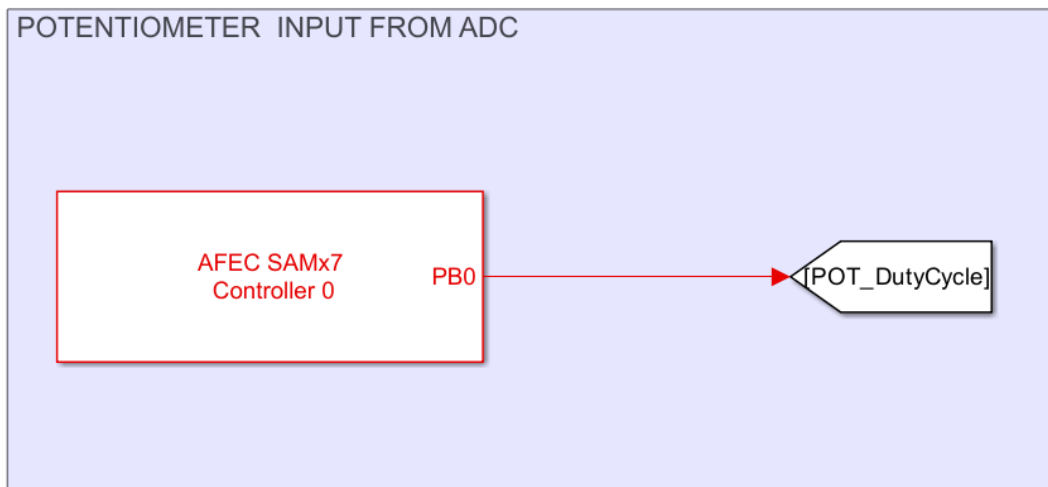


Figura 5.3: Modello Simulink dell'acquisizione della tensione del potenziometro, tramite periferica AFEC

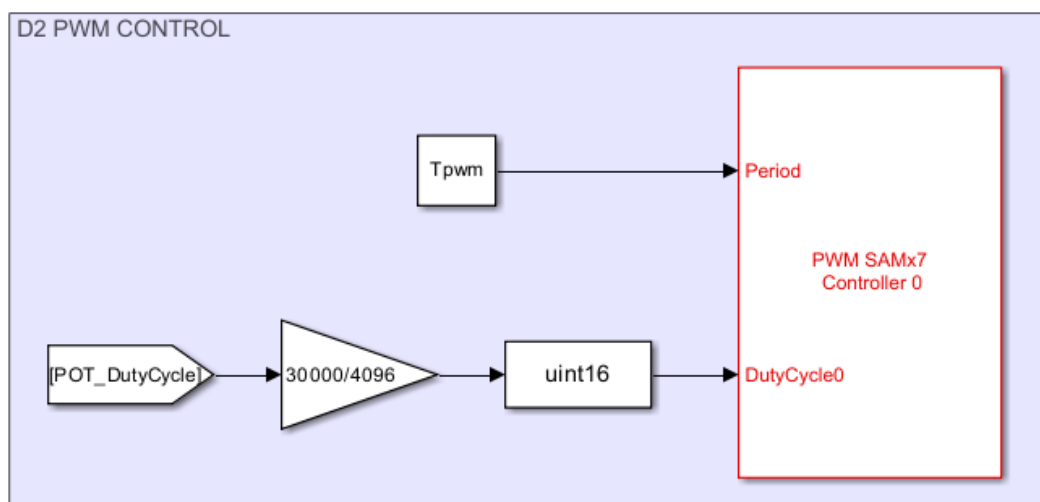


Figura 5.4: Modello Simulink per la Modulazione a larghezza di impulso del led D2, tramite periferica PWM

L'algoritmo prevede una fase iniziale in cui l'AFEC produce un valore intero a 12 bit senza segno sulla base del valore in tensione in uscita dal potenziometro e lo salva nella variabile **POT_DutyCycle**; successivamente il risultato dell'acquisizione viene **scalato con un blocco gain** e convertito in formato **uint16** (intero senza segno a 16 bit).

Il ridimensionamento del valore avviene con un fattore:

$$GAIN = \frac{30000}{4096} \approx 7.32 \quad (5.1)$$

In questo modo viene **adattato al range di conteggio del timer**, massimizzando la risoluzione del segnale PWM in uscita, verso il LED D2. Il valore 30000 deriva dal fatto che il Master, per garantire il periodo massimo impostato ($2 \times T_{pwm} = 0.002$), ha scelto un valore del prescaler in modo tale che il contatore arrivi a 59999 ($60000 - 1$) in 0.002 secondi. Quindi per generare una portante con periodo di 0.001 secondi (T_{pwm}), il contatore si deve resettare ogni volta che arriva a 29999 ($30000 - 1$). Ne risulta che il range di valori con il quale può avvenire l'evento di match comprende 30000 numeri uint16 e con il potenziometro si seleziona uno di quei valori.

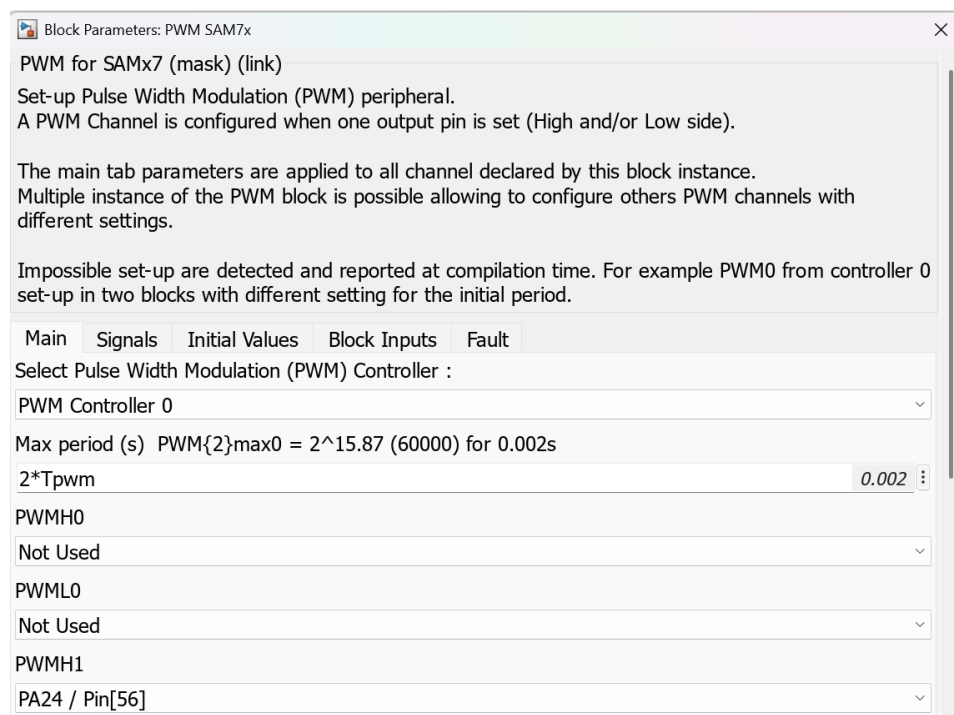


Figura 5.5: Parte della sezione "Main" del blocco PWM SAMx7 di MATLAB Simulink (blocco del toolbox PLAB Device Blocks for Simulink: dsPIC, PIC32 e SAM mcu). Si può notare l'indicazione " $PWM2max0 = 60000$ for 0.002".

5.2 I passaggi per eseguire la programmazione della scheda

Come è stato esposto nella sezione riservata al Programmatore/Debugger MPLAB ICD-4, dal software **MPLAB X IDE** è possibile eseguire la programmazione e il debugging completo della scheda a cui il programmatore è connesso. Di per sé, le istruzioni per eseguire il debugging sono esposte in dettaglio nel documento relativo, il cui URL è scritto nella bibliografia di questo elaborato di tesi[7]. Tuttavia, si intende esporre in maniera riassuntiva la procedura operativa adottata nei vari test del codice descritto nelle altre sezioni di questo capitolo:

1. Dall'interfaccia Simulink, sul **codice che si vuole caricare nella scheda**, espandere la tendina "**MICROCHIP**" e premere il pulsante "**Build Model**". In alternativa si può andare nella **sezione "MICROCHIP"** (tra "FORMAT" e "APPS"), premere il pulsante "**Embedded Coder**" e in seguito premere "**Build**";
2. Se la programmazione è andata a buon fine (altrimenti riferirsi alla diagnostica di Simulink), dovrebbe essere apparso un file **.X** nella directory in cui si trova il file Simulink appena compilato. A questo punto si deve **eseguire il software MPLAB X IDE** e aprire da esso la cartella dove si trova il progetto (.X) appena creato, nella sezione **Open Project** (in alto a sinistra della schermata, sotto "View" e "Navigate").
3. A questo punto si deve **connettere l'ICD-4 al PC** tramite una porta USB qualsiasi. Se la connessione è andata a buon fine, in basso a destra di MPLAB X IDE dovrebbe apparire il messaggio "**MPLAB ICD 4 is found**";
4. Andare in "**Project Properties**" (sempre in alto a sinistra) e replicare le impostazioni mostrate in figura 5.6. Se non appare il compilatore, premere la sezione in alto "**Tools > Options > Embedded > Build Tools**", per poi comunicare ad MPLAB X IDE la posizione dello stesso nella memoria del PC (come si mostra in figura 5.7);
5. Se ora si connette la scheda all'alimentazione e all'ICD-4 (attenendosi alla figura 3.8), si potrà premere il pulsante "**Make and Program Device Main Project**" e completare così la programmazione della scheda dsPICDEM™ MCLV-2.

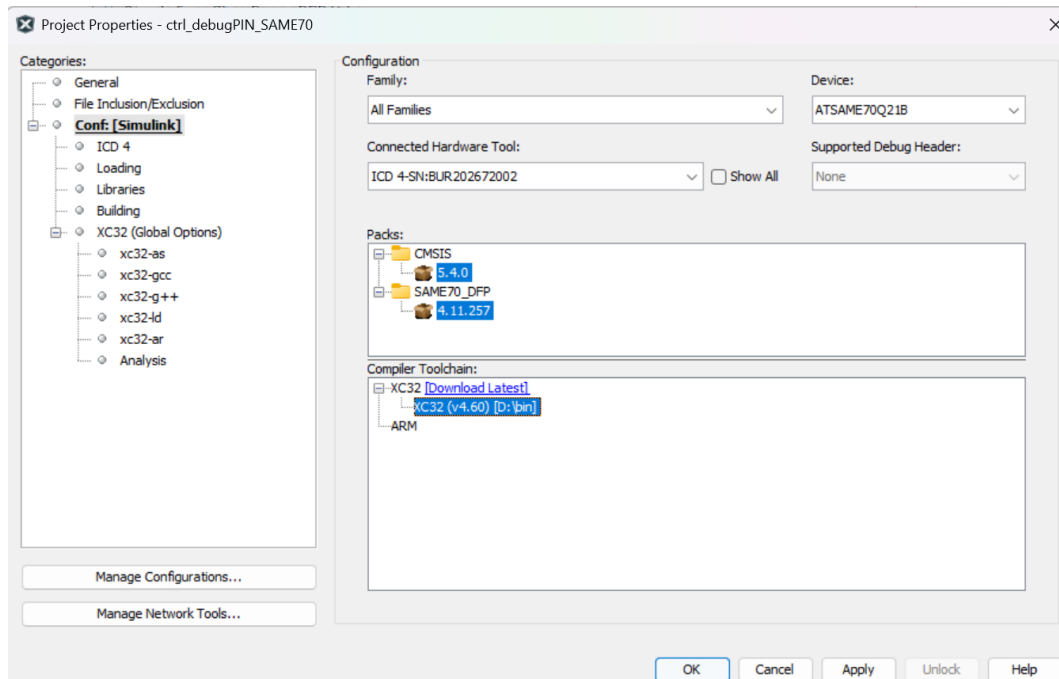


Figura 5.6: Schermata di MPLAB X IDE per la configurazione generale.

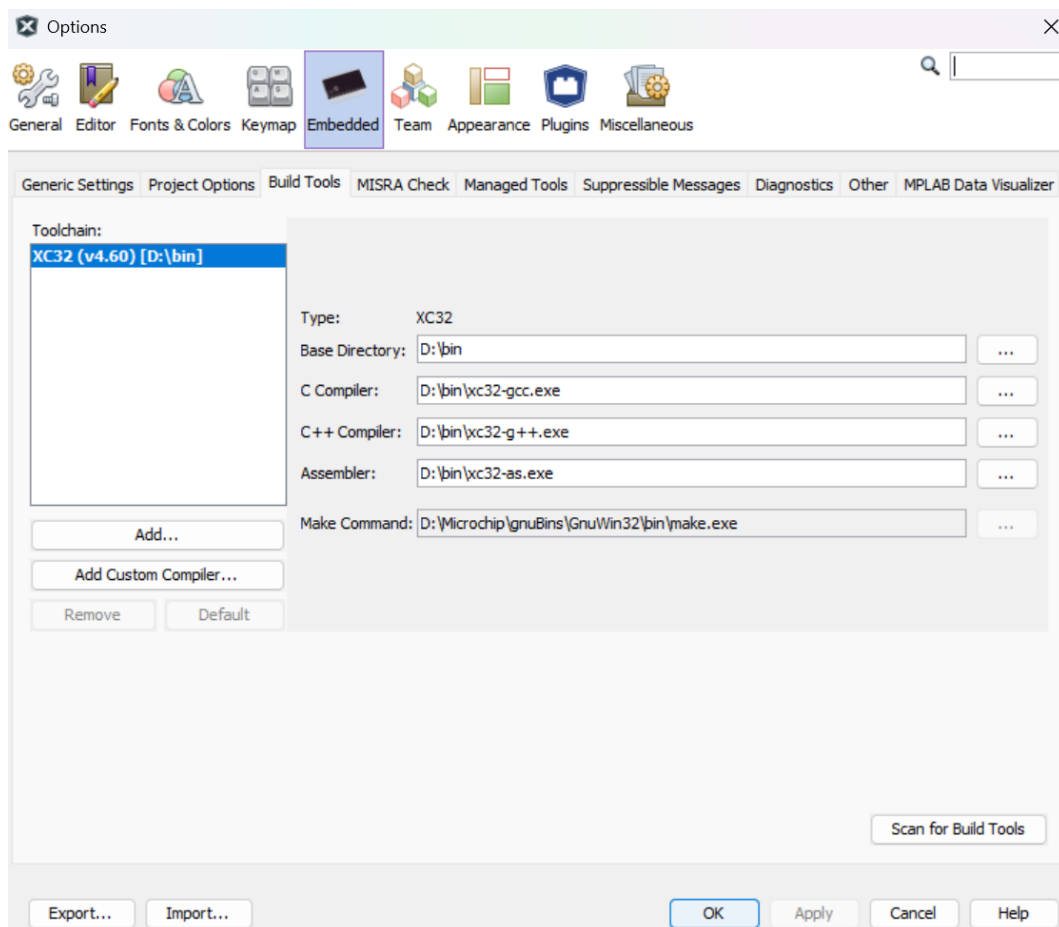


Figura 5.7: Schermata di MPLAB X IDE per la selezione del compilatore.

Capitolo 6

Comunicazione seriale con la scheda da MATLAB Simulink

In quest'ultimo capitolo del progetto di tesi si vuole spiegare com'è stata attuata la comunicazione seriale con la scheda dsPICDEM™ MCLV-2 da MATLAB Simulink, tramite un codice host.

I codici di test prevedono:

- la **trasmissione di una parola binaria da 8 bit verso la scheda**, che maschera: un numero da 0 a 127 e un flag di enable;
- la **ricezione di un vettore di 120 parole binarie da 8 bit dalla scheda**, di cui 118 contengono sempre (ciascuna): un numero da 0 a 127 e un flag di enable, mentre le altre servono a comunicare gli estremi del pacchetto al codice host.



Figura 6.1: MCP2200 USB TO RS232 DEMO BOARD (Fonte: <https://www.microchip.com/en-us/development-tool/mcp2200ev-vcp>).

Tra le porte di comunicazione presenti nella scheda dsPICDEM™ MCLV-2, è stata utilizzata la **porta RS-232** per la **comunicazione seriale asincrona con l'ester-**

no. Il collegamento ad una delle porte USB del PC è stato effettuato tramite un adattatore RS-232/mini_USB ed un cavo mini_USB/USB.

6.1 La configurazione della periferica UART

Come citato all'inizio della sezione 5.1 della tesi, tra i blocchi di configurazione utilizzabili vi è "UART Configuration" (presente nel toolbox MPLAB Device Blocks for Simulink). In tale blocco si selezionano i parametri fondamentali per la trasmissione seriale tramite periferica UART (Universal Asynchronous Receiver-Transmitter), che poi il master traduce in istruzioni di configurazione in fase di building del codice.

Innanzitutto, osservando la sezione "Main" del blocco di configurazione, è necessario selezionare quale periferica UART del microcontrollore utilizzare. In questo caso, siccome si vuole utilizzare la **porta RS-232** del dsPICDEM™ MCLV-2, è obbligatorio configurare ed utilizzare la periferica: **UART0 (PA9/Pin[75] per Rx e PA10/Pin[66] per Tx)**. A tal proposito è importante ricordarsi di **connettere i jumper 7-8 in JP4 e JP5 per fare in modo che le linee di trasmissione e ricezione siano connesse alla porta seriale RS-232**. Sempre in questa sezione, poi è stata selezionata una **Interrupt Priority pari a 3** (valore intermedio) ed un **Baud Rate di 230 400 [bit/s]**. Quest'ultimo valore è dovuto al fatto che i gruppi di 120 parole binarie vengono trasmessi con una cadenza di 6 ms; siccome poi ogni parola è da 8 bit con 1 bit di start ed 1 di stop, significa che devono essere trasmesse:

$$TxRate = \frac{120 \cdot 10}{0.006} \text{ bit/s} = 200\,000 \text{ bit/s}. \quad (6.1)$$

Si riportano in figura 6.2 le impostazioni adottate per questo codice (sono state tenute quelle di default per quanto riguarda le altre impostazioni, a parte quelle in "Main").

Per la ricezione viene implementato un semplice **buffer circolare**, in grado di ospitare al massimo 8 byte per volta, invece per la **trasmissione seriale si utilizza il DMA in modalità Ping-Pong**[6]. In particolare si nota che è stato impostato un **DMA Buffer size pari a 120 byte**, ciò significa che **ogni microblocco di dati associato è composto da 120 parole binarie da 8 bit (1 byte)**.

L'implementazione della modalità Ping-Pong prevede la **predisposizione di due buffer di dati in memoria (in questo caso, ognuno da 120 byte)**, riempiti ad ogni ciclo di trasferimento dalla CPU. Vengono inoltre preparati **due descrittori**

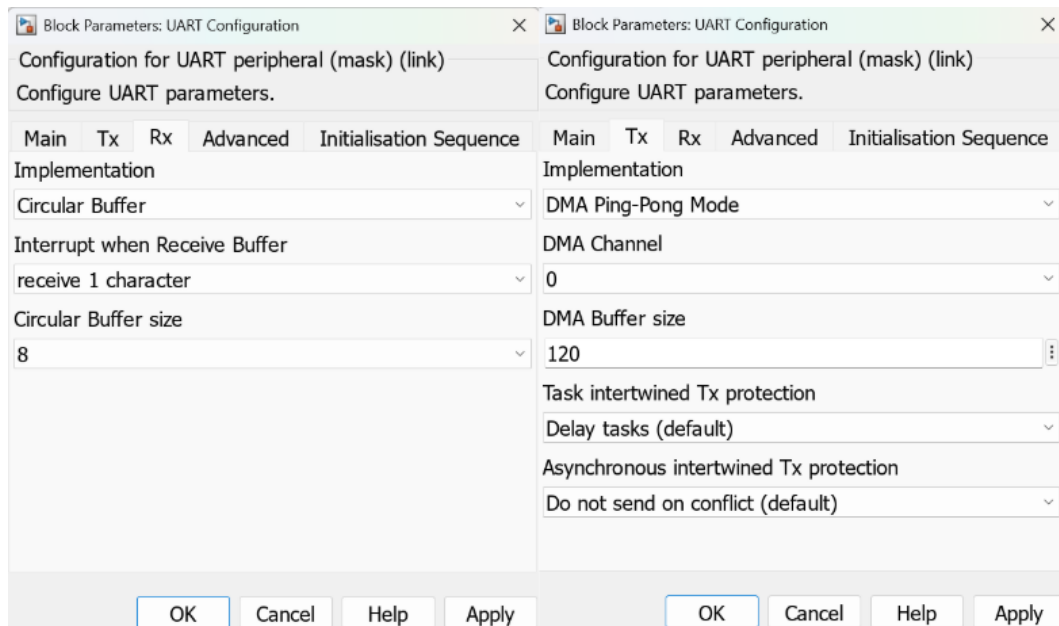


Figura 6.2: Configurazione UART0 per ricezione e trasmissione, tramite blocco "UART Configuration"

(**View 0 o 1**), nei quali il primo viene linkato al secondo ed il secondo al primo (tramite i campi MBR_NDA: Micro-Block Member of Next Descriptor Register). La modalità Ping-Pong prevede allora di trasferire alla periferica UART il contenuto del primo dei due buffer in memoria (**Microblocco Ping**) secondo le regole del primo descrittore; successivamente il DMA Controller legge il secondo descrittore dove si regola il trasferimento del secondo buffer in memoria (**Microblocco Pong**), mentre la CPU torna a riempire il primo buffer. Nel secondo descrittore, l'XDMAC trova in MBR_NDA l'indirizzo del primo descrittore ed il ciclo ricomincia. Ogni microblocco viene trasferito alla periferica UART0 **sotto forma di Chunk**; in questo modo **il trasferimento tra memoria e periferica è sincronizzato con la periferica stessa**. Inoltre, la CPU sa quando deve riscrivere il contenuto di uno dei due buffer grazie alla **IRQ (Interrupt Request) inviata dall'XDMAC subito dopo l'ultimo Chunk del Microblocco (Ping o Pong)**.

6.2 Implementazione del lampeggiamento a frequenza modulabile di un led

In questa sezione si spiega com'è stato implementato il codice che utilizza i valori numerici provenienti dalla ricezione seriale per **regolare ed abilitare il lampeggiamento del LED di debug D17** della scheda dsPICDEM™ MCLV-2. Come è stato

visto nella sezione precedente, dalla scheda vengono ricevuti due valori mascherati in una parola binaria da 8 bit. Tali valori vengono poi salvati in variabili globali denominate:

- **InputVal:** contenente una parola binaria a 7 bit, rappresentante di un **numero intero senza segno da 0 a 127**;
- **Enable:** contenente un **valore booleano, quindi 0 o 1**.

Per la rappresentazione di **variabili globali in MATLAB Simulink** (ovvero variabili che vivono al di fuori dello stream dell'algoritmo in elaborazione, quindi la posizione più bassa dello stack LIFO in RAM), si utilizza il blocco **Data Store Memory** della libreria Simulink/Signal Routing.

Per interagire con il LED di debug della scheda, è necessario in questo caso utilizzare il blocco **Digital Output Write** del toolbox MPLAB Device Blocks for Simulink. Tale blocco, una volta selezionata la **porta relativa al LED D17 (PC23)**, è in grado di accendere il LED a seconda del valore logico dell'ingresso.

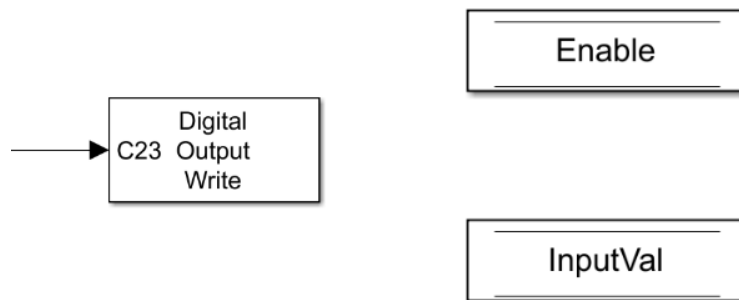


Figura 6.3: Blocchi Simulink: "Data Store Memory" per la creazione delle variabili globali e Digital Output Write.

Come si nota dalla figura 6.4, l'abilitazione del lampeggiamento viene attuata, appunto, tramite il valore logico contenuto nella variabile **Enable**: se impostato a **0** allora il **LED rimane acceso senza lampeggiare**; se invece viene impostato a **1**, esso **lampeggia ad una frequenza determinata dal valore contenuto in InputVal (intero senza segno da 0 a 127)**. Sempre osservando la figura 6.4, il circuito nel sottosistema condizionale "case{1}" serve a generare un **lampeggio ad una frequenza da 0.1 Hz a 1 Hz** al variare del contenuto di InputVal (che qui prende il nome di "Max Count [0 127]"). Il funzionamento di tale algoritmo si basa sul **contatore "counter"** (nel riquadro viola), che **incrementa di 1 con un periodo di: $T_s = 0.5 \cdot 10^{-5} s$** . Tramite il valore in ingresso si imposta il **limite del contatore**, secondo la seguente relazione lineare:

$$F_{out} = \frac{1 - 0.1}{127} \cdot InputVal + 0.1$$

$$CounterLimit = \frac{F_{out}}{T_s}$$

dove F_{out} è la frequenza di lampeggiamento (in Hz) selezionata per il LED.

La generazione dell'onda quadra a frequenza F_{out} variabile, con cui lampeggia il LED D17, può essere spiegata dalla figura 6.5. Si nota che **il duty cycle dell'onda quadra è stato impostato fisso al 50%** per fare in modo che il LED lampeggi uniformemente.

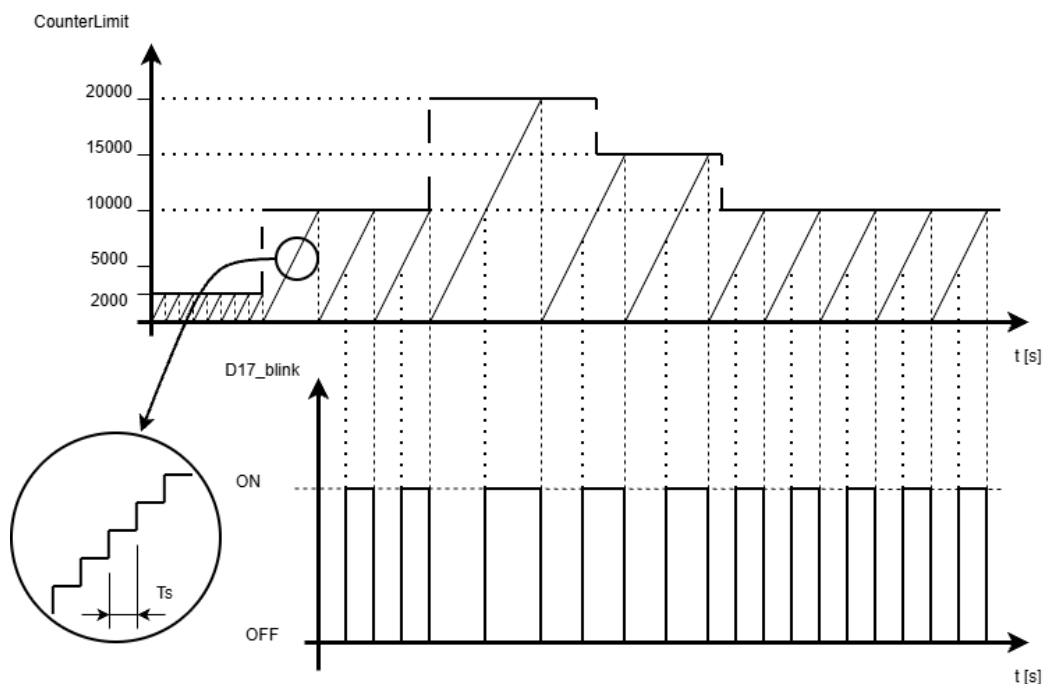


Figura 6.5: Lampeggiamento a frequenza variabile di Debug Led D17, con duty cycle fisso al 50%.

6.3 Implementazione di ricezione e trasmissione seriale nel codice Main

Innanzitutto si analizza l'**algoritmo di ricezione seriale**, che prevede il **rilevamento** e il **prelievo** di un nuovo dato (**parola binaria da 8 bit**) dalla periferica UART0. In seguito si estraggono da esso i valori di **Enable** e **InputVal**, per poi **sovrascrivere con essi il contenuto delle rispettive variabili globali (blocchi "Data Store Memory")** descritte nella sezione precedente.

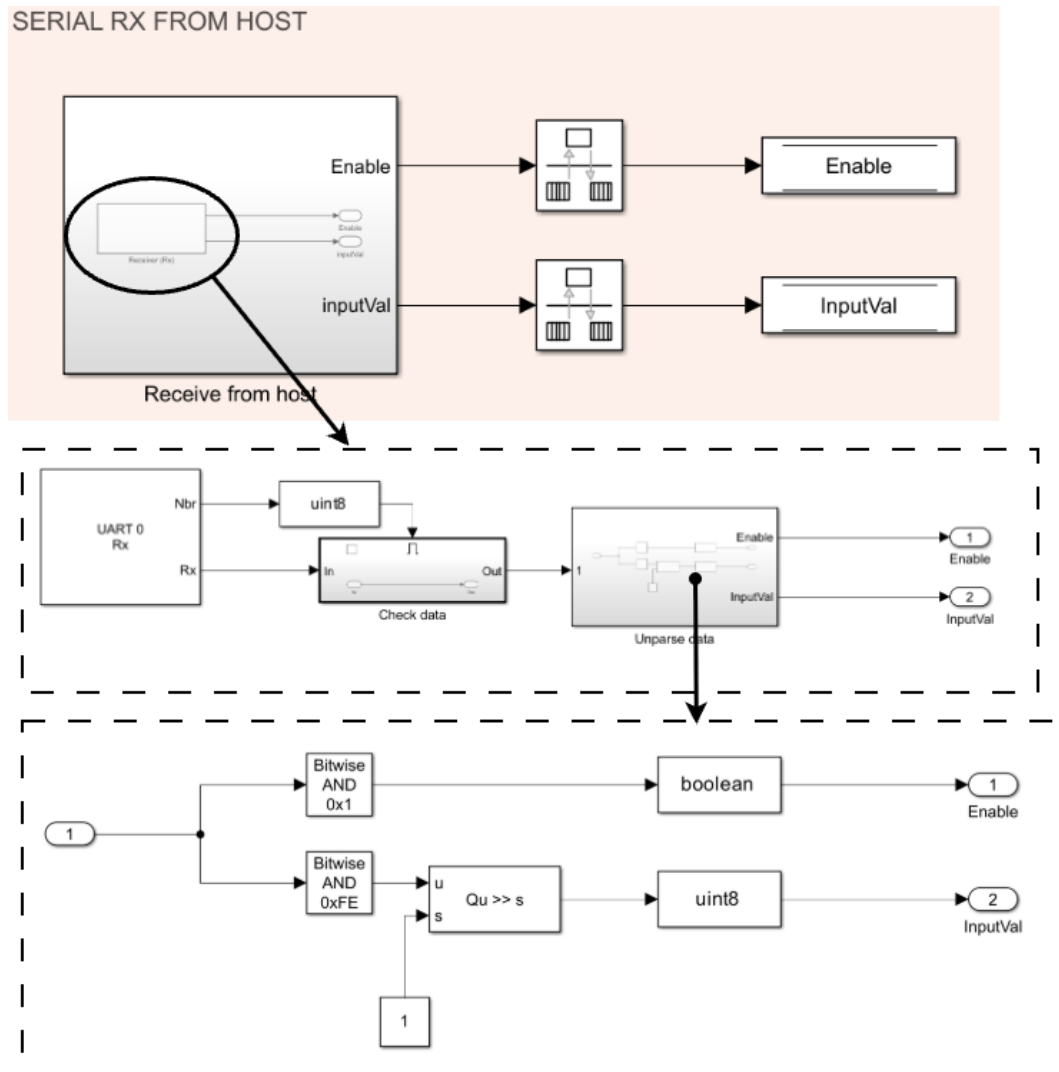


Figura 6.6: Codice Simulink per la ricezione seriale da Host.

Il codice Simulink rappresentato nella figura 6.6 si può suddividere in **tre parti fondamentali**:

1. Il **blocco UART0 Rx**, presente nel toolbox "MPLAB Device Blocks for Simulink", che permette di utilizzare una periferica UART per la ricezione, in questo caso l'UART0. La configurazione di questo blocco consiste principalmente nel selezionare: la periferica UART utilizzata, quanti byte deve ricevere a ciclo (in questo caso 1 byte) ed il tempo di campionamento. Quest'ultimo è stato selezionato pari a $T_{pwm} = 0.001\text{ s}$, siccome la periferica è chiamata a ricevere dati in maniera sporadica;
2. Il **sottosistema "Check data"**, ovvero un normalissimo subsystem il cui **codice interno viene eseguito un numero di volte pari agli impulsi di trigger**

e che, ogni volta che viene triggerato, **permette al dato "In" di passare in "Out"**. Per questa applicazione è fondamentale, siccome l'UART0 non è chiamata a ricevere dati in continuazione ma solo quando lo decide l'host;

3. Infine, il sottosistema **"Unparse data"** ha il compito di smascherare l'**Enable (bit meno significativo)** ed **InputVal (i 7 bit rimanenti più significativi)** dalla parola binaria in ingresso.

Dato che il resto del codice Simulink al di fuori di quello descritto in questa sezione (figura 6.6) opera ad un periodo di campionamento di $T_s = 5 \cdot 10^{-5} s$ e quindi ad una frequenza molto maggiore, **è necessario che ci siano dei blocchi "Rate Transition" nell'interfaccia del sottosistema**. Tali blocchi sono fondamentali in questo tipo di codici, perché permettono di **salvaguardare l'integrità ed il determinismo dei dati (evitando collisioni)** quando ci sono delle parti di codice, o dei blocchi, che operano a **frequenze diverse (Sample Time)**[9]. Si riporta in figura 6.7 la configurazione adottata per i blocchi Rate Transition utilizzati.

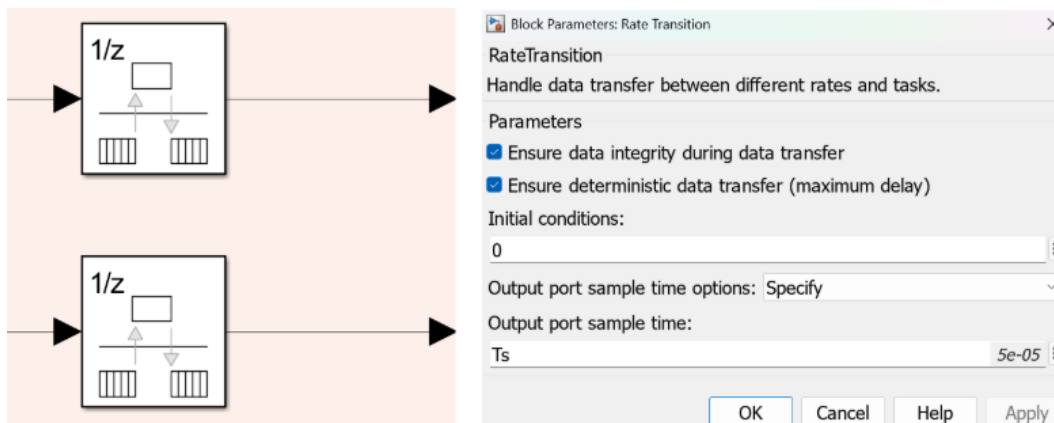


Figura 6.7: Configurazione dei blocchi "Rate Transition".

Per quanto riguarda invece l'**algoritmo di trasmissione**, esso ha i seguenti scopi: **mascherare i valori di POT_DutyCycle ed Enable in una parola binaria da 8 bit e creare un vettore di 120 parole binarie da spedire all'host**. L'obiettivo finale di questo codice consiste nel rendere visualizzabili alcuni parametri nel tempo, per dare un feedback al programmatore in fase di debug. In particolare si trasmettono: **la percentuale di intensità luminosa del LED di debug D2** (che corrisponde al duty cycle selezionato tramite il potenziometro) e **il valore logico del segnale Enable**, così come è stato ricevuto.

Il codice riportato in figura 6.8 risulta essere, in parte, la versione speculare del codice di ricezione (rappresentato dalla figura 6.6); tuttavia differisce molto nella parte in cui si genera il pacchetto da spedire.

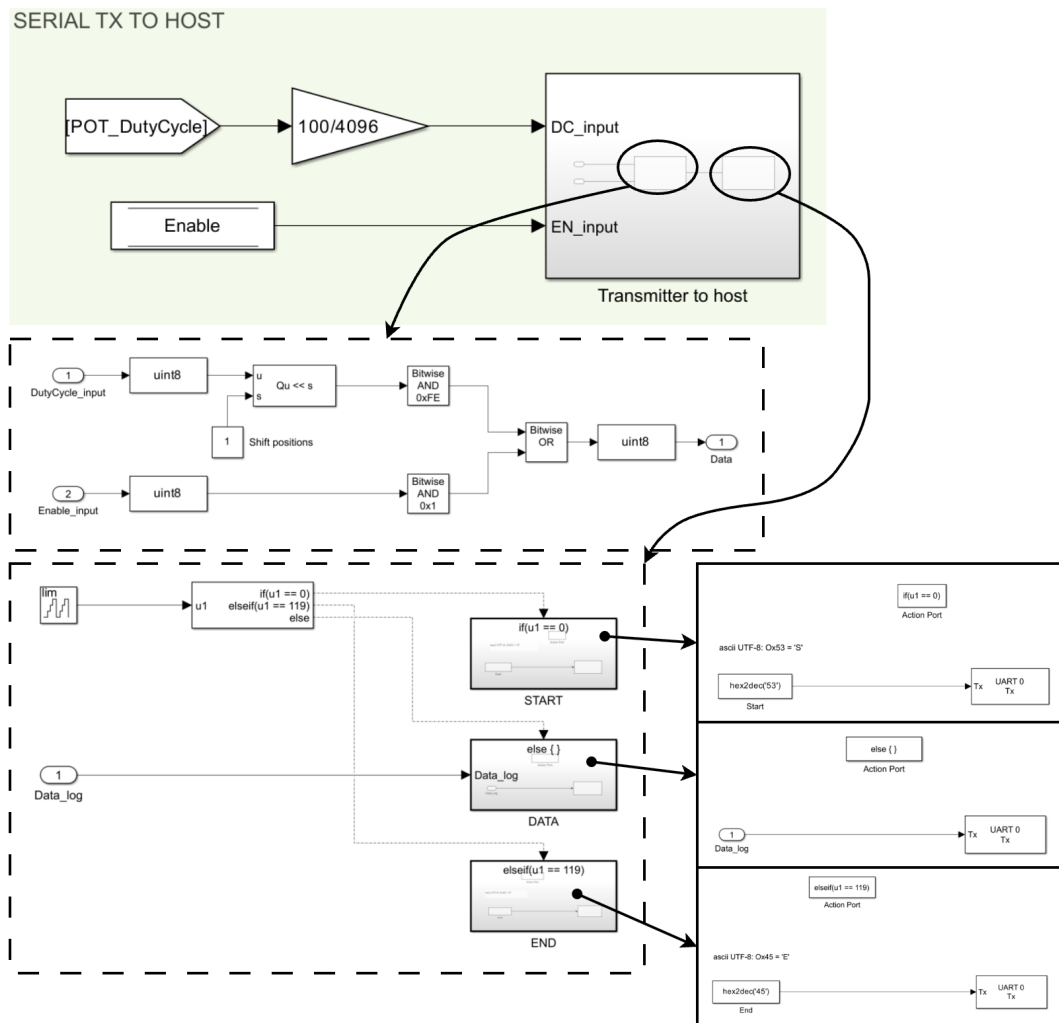


Figura 6.8: Codice Simulink per la trasmissione seriale all'host.

Sempre dalla figura 6.8, osservando il sottosistema più a valle, si nota che c'è un **contatore** la cui uscita è connessa ad un blocco condizionale in cui si discretizzano i casi in cui il conteggio è: 0, un numero qualsiasi, o 119. Il contatore, infatti, **conta da 0 a 119 per poi ricominciare**, con un **periodo di incremento pari a**: $T_s = 5 \cdot 10^{-5} s$; di conseguenza **termina un ciclo di conteggio in un tempo pari a**:

$$T_{cycle} = 120 \cdot T_s = [120 \cdot 5 \cdot 10^{-5}] s = 6 \cdot 10^{-3} s = 6 ms. \quad (6.2)$$

All'interno di ogni sottosistema condizionale (triggerato da una "Action Port") è presente un blocco "UART0 Tx" (sempre del toolbox: "MPLAB Device Blocks for Simulink") ed un dato in input. **Ogni volta che uno dei tre sottosistemi viene azionato, il dato in input al blocco "UART0 Tx" viene spedito all'host**; in questo modo l'host riceve ogni volta un **vettore di dati che ha un capo (Start: UTF-8 "S" = 0x53) e una coda (End: UTF-8 "E" = 0x45)**. Tra capo e coda vengono così spedite (verso l'host) **118 parole binarie da 8 bit** che contengono i valori: POT_DutyCycle ed Enable.

Il vettore di dati, così formato, è visualizzabile come nell'equazione sottostante.

$$DataVector = \begin{bmatrix} Ox53 = Ob01010011 \\ data_1 \\ \vdots \\ data_{i-1} \\ data_i \\ data_{i+1} \\ \vdots \\ data_{118} \\ Ox45 = Ob01000101 \end{bmatrix} \quad \text{dove } data_i = Ob d_{7,i} d_{6,i} d_{5,i} d_{4,i} d_{3,i} d_{2,i} d_{1,i} d_{0,i} \quad (6.3)$$

6.4 Implementazione di trasmissione e ricezione da seriale nel codice Host

Il codice host viene eseguito solo e unicamente nell'ambiente di MATLAB Simulink e non viene caricato nella scheda dsPICDEM™ MCLV-2; di conseguenza, **non utilizza i blocchi del toolbox MPLAB Device Blocks for Simulink**. Tuttavia, utilizza quelli del **toolbox "Motor Control Blockset/Protection and Diagnostics"**, ovvero:

1. **Host Serial Setup**, necessario alla configurazione della porta seriale del PC con cui ci si sta interfacciando;
2. **Host Serial Receive**, per attuare la ricezione da una delle porte seriali del PC;
3. **Host Serial Transmit**, per attuare la trasmissione tramite una delle porte seriali del PC.

Con il codice Simulink di interfaccia, visualizzabile in figura 6.9, si possono imporre: il **valore di Enable** tramite lo switch e il **valore di InputData** (con cui si modula la frequenza del LED D17, modificando il contenuto della variabile globale InputValue). Il blocco più importante è quello per eseguire la configurazione della

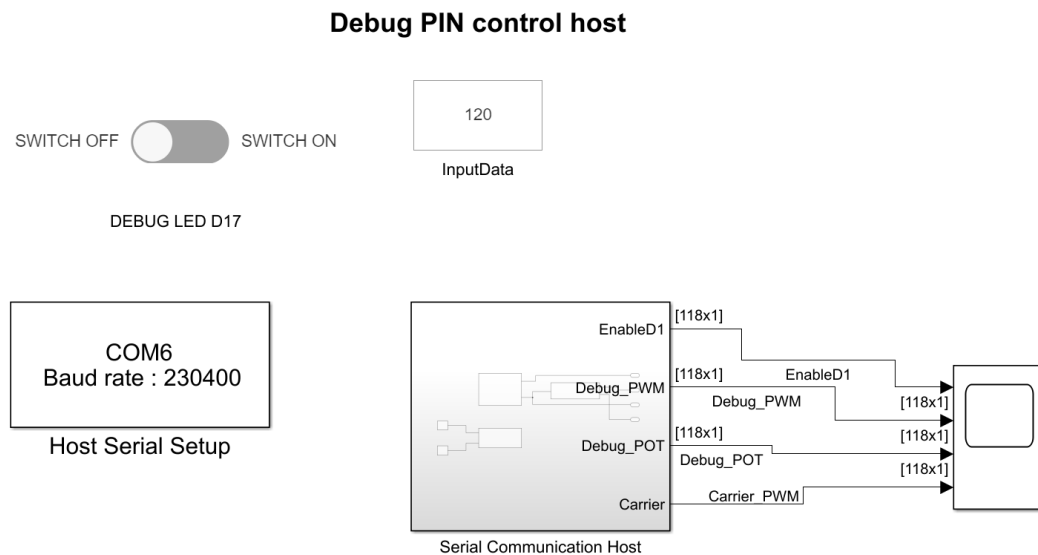


Figura 6.9: Interfaccia del codice host con la quale ci si è interfacciati alla scheda.

porta seriale, nel quale sono stati selezionati:

- la **porta seriale utilizzata** nel PC (in questo caso COM6);
- il **Baud Rate**, che **deve essere uguale identico a quello impostato nel codice caricato nella scheda**, in questo caso: 230400 bit/s ;
- e la **dimensione del singolo dato pari a 8 bit**.

Il **codice host di trasmissione** (via seriale), verso la scheda dsPICDEM™ MCLV-2, è riportato in figura 6.10. Tralasciando questa volta il blocco "Parse Information", che è esattamente lo stesso blocco riciclato dal codice main, si analizzano i sottosistemi: "**Change detector**" e "**Transmit block**". Il sottosistema "**Change detector**" si può dire anche "**Rilevatore di fronte**", e, come dice il nome, serve a rilevare se il

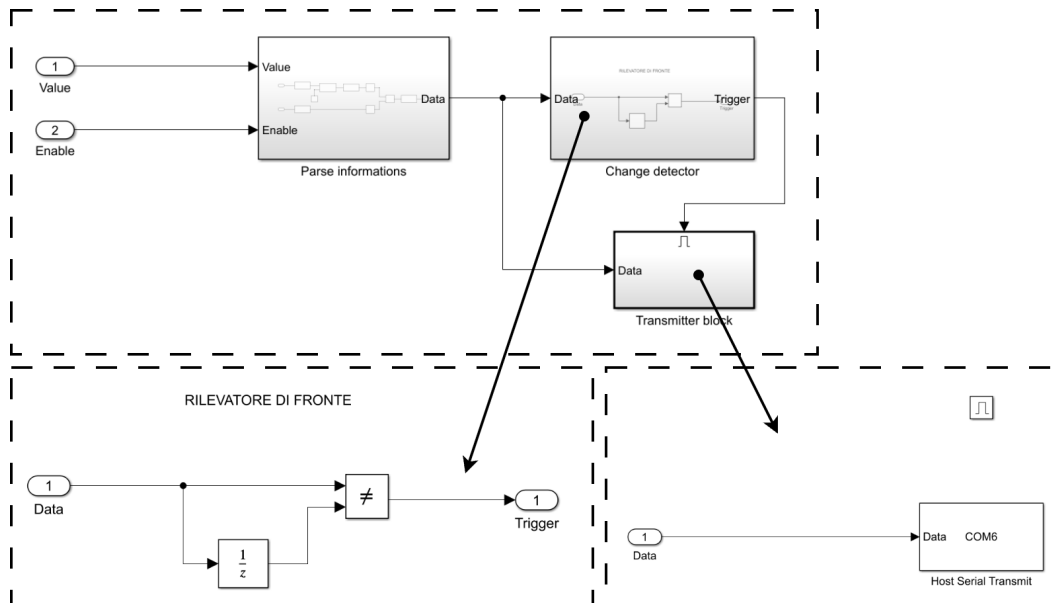


Figura 6.10: Codice per la trasmissione dei dati verso la scheda, tramite porta seriale del PC di host.

dato in ingresso "Data" ha subito dei cambiamenti. In questo caso, invia un **impulso di trigger** al blocco di trasmissione "**Transmitter block**", che appunto avvia la trasmissione del dato tramite porta seriale.

Analizzando ora il **codice host di ricezione**, riportato in figura 6.11, si evidenzia immediatamente la presenza di una **MATLAB Function** chiamata "**Unbuffer Data**" prima del solito blocco di smascheramento dei dati ("Unparse data"). Tale MATLAB Function serve a **inserire i 118 dati**, trasmessi ad ogni ciclo di $120 \cdot T_s$ dalla scheda (esclusi quindi quelli di "Start" e di "End"), **all'interno di una variabile vettoriale di MATLAB** chiamata "**Data**".

La parte più importante del sottosistema di ricezione del codice host è probabilmente la **configurazione del blocco "Host Serial Receive"**. Infatti con essa si impostano:

- La **dimensione del pacchetto di dati in ingresso**;
- Il **formato del singolo dato**;
- Il **carattere iniziatore del pacchetto** (in questo caso "S");
- Il **carattere terminatore del pacchetto** (in questo caso "E");
- Ed infine il "**Block sample time**", ovvero il tempo di campionamento del blocco, che è stato impostato **uguale al periodo di trasmissione del codice caricato nella scheda**.

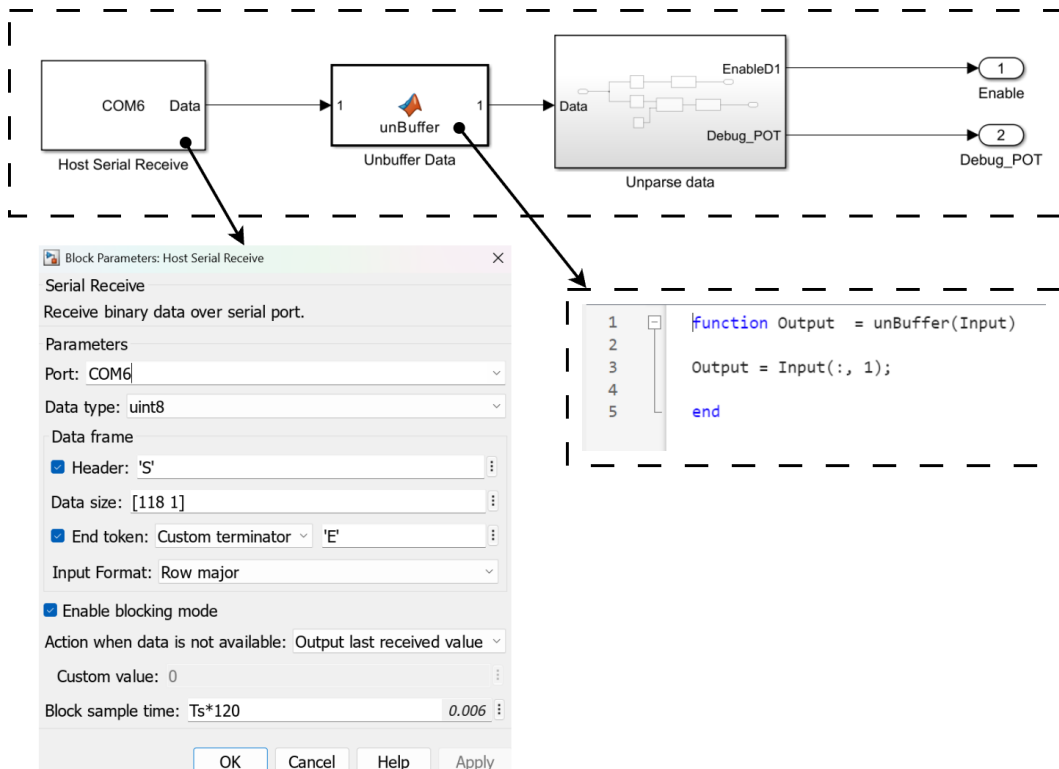


Figura 6.11: Codice per la ricezione del vettore di dati dalla scheda, tramite porta seriale del PC di host.

6.5 Test della comunicazione seriale

Una volta che il codice è stato convertito e caricato nella scheda, è possibile **premere il tasto Run del codice host** per visualizzare, tramite il blocco "Scope", le variazioni dei parametri trasmessi via seriale. In particolare, dalla figura 6.12, si possono vedere due andamenti:

- L'andamento **rosso** rappresenta il duty cycle selezionato con il potenziometro e quindi **l'intensità luminosa percentuale del LED di debug D2**;
- L'andamento **blu** rappresenta invece il disegno di un segnale PWM costruito con il duty cycle regolato con il potenziometro. Per costruire tale andamento è stato scritto un semplice codice Simulink, riportato nella figura 6.13, che simula la portante generata da un timer, messa a confronto con il duty cycle ricevuto.

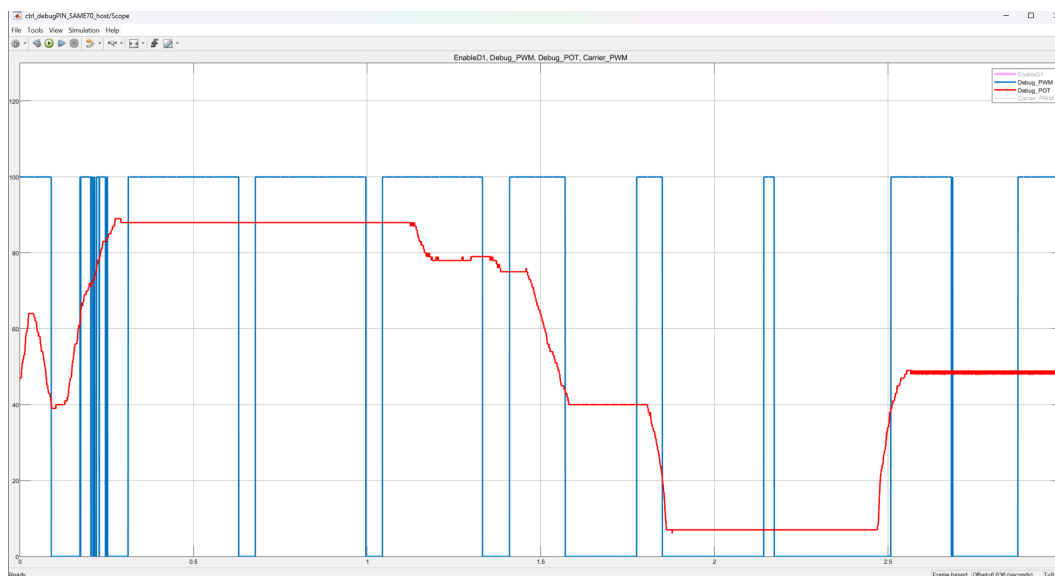


Figura 6.12: Test della comunicazione, visualizzato dal blocco "Scope" di Simulink da codice host.

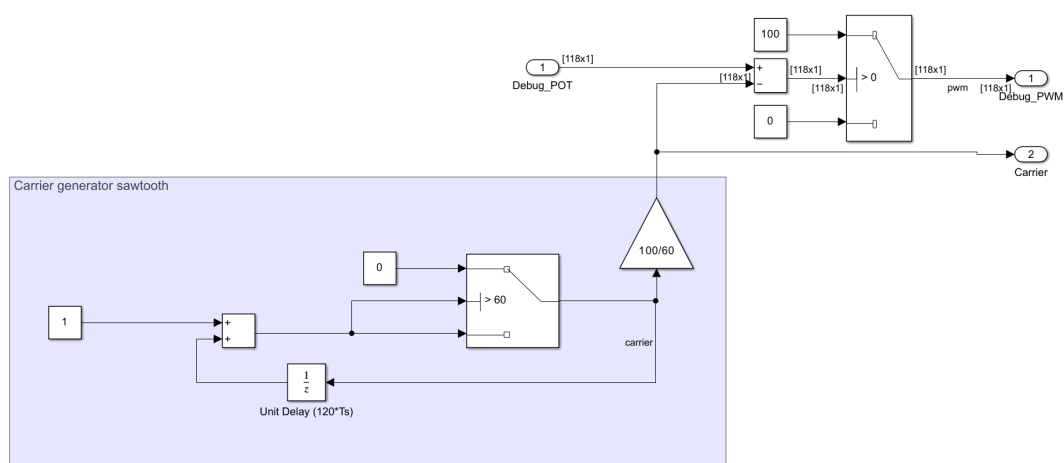


Figura 6.13: Codice Simulink per il disegno dell'onda quadra modulabile nello blocco "Scope" del codice host.

Capitolo 7

Conclusioni

In conclusione, è stata analizzata la notevole complessità hardware di microcontrollori potenti e sofisticati (come quelli delle famiglie SAM E70/S70/V70/V71), particolarmente adatti a contesti industriali che richiedono elevate prestazioni. Con questo elaborato di tesi si è applicato un concetto ormai consolidato nell'ambito industriale odierno: la programmazione Model-Based. Grazie a questo paradigma, è stato possibile configurare le periferiche tramite un'interfaccia grafica intuitiva, senza la necessità di intervenire manualmente sui numerosi registri. Ciò velocizza significativamente il processo di programmazione e permette di sviluppare algoritmi con un supporto software completo, riducendo al minimo i rischi di errore umano.

Il presente lavoro costituisce un solido punto di partenza per futuri sviluppi. Avendo consolidato i concetti di base per la programmazione e l'interfacciamento con la scheda dsPICDEM™ MCLV-2 tramite MATLAB Simulink, sarà possibile implementare progetti di maggiore complessità orientati alla prototipazione di azionamenti elettrici.

Bibliografia

- [1] MathWorks, *Code generation using Embedded Coder*. Disponibile su: <https://it.mathworks.com/help/ecoder/ug/generating-code-using-embedded-coder.html>.
- [2] MathWorks, *Description MATLAB Coder*. Disponibile su: https://it.mathworks.com/help/coder/index.html?s_tid=CRUX_topnav.
- [3] MathWorks, *Description Simulink Coder*. Disponibile su: <https://it.mathworks.com/help/rtw/>.
- [4] Microchip Technology Inc., *dsPICDEM™ MCLV-2 Development Board User's Guide*. Disponibile su: <https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/UserGuides/DS-52080a.pdf>.
- [5] Microchip Technology Inc., *ATSAME70 100-Pin Motor Control Plug-In Module Information Sheet*. Disponibile su: <https://www.microchip.com/en-us/development-tool/ma320203>.
- [6] Microchip Technology Inc., *SAM E70/S70/V70/V71 Family Data Sheet*. Disponibile su: <https://ww1.microchip.com/downloads/aemDocuments/documents/MCU32/ProductDocuments/DataSheets/SAM-E70-S70-V70-V71-Family-Data-Sheet-DS60001527.pdf>.
- [7] Microchip Technology Inc., *MPLAB® ICD 4 In-Circuit Debugger User's Guide*. Disponibile su: <https://ww1.microchip.com/downloads/en/DeviceDoc/50002596E.pdf>.
- [8] *Introduzione alle applicazioni industriali di Microcontrollori e DSP*, versione 2020, ISBN: 9788874889358, Simone Buso.
- [9] MathWorks: MATLAB Help Center (*Rate Transition*). Disponibile su: <https://it.mathworks.com/help/simulink/slref/ratetransition.html>.