Real-Time Hands-on Tool for Teaching Three-Phase Motor Control

Guide for Setting Up ADC and SPI Between Raspberry Pi 5 and TI LaunchPad

Hardware

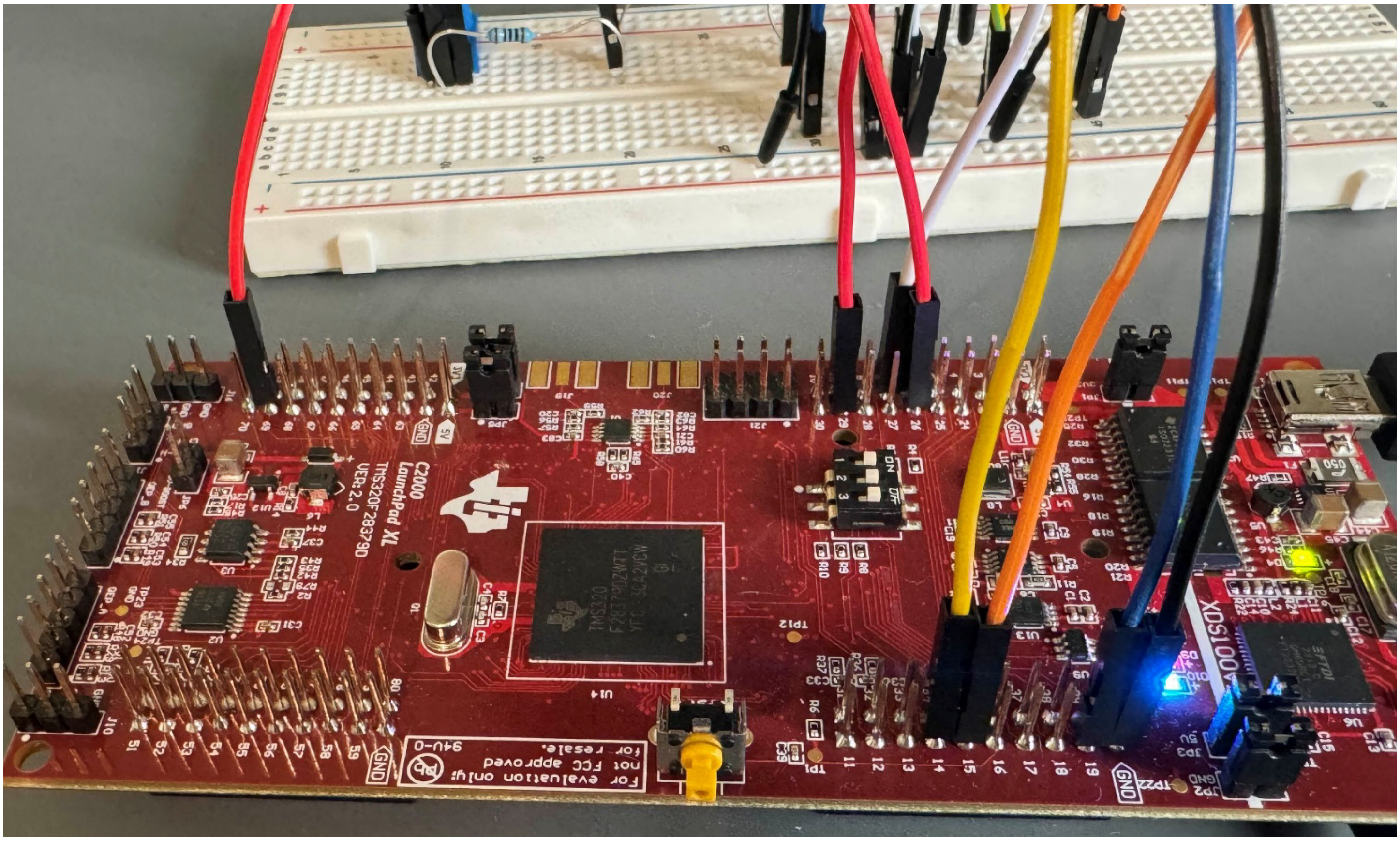
Connections between Raspberry Pi 5 (RPi) and LaunchPad (MCU) are as follows.

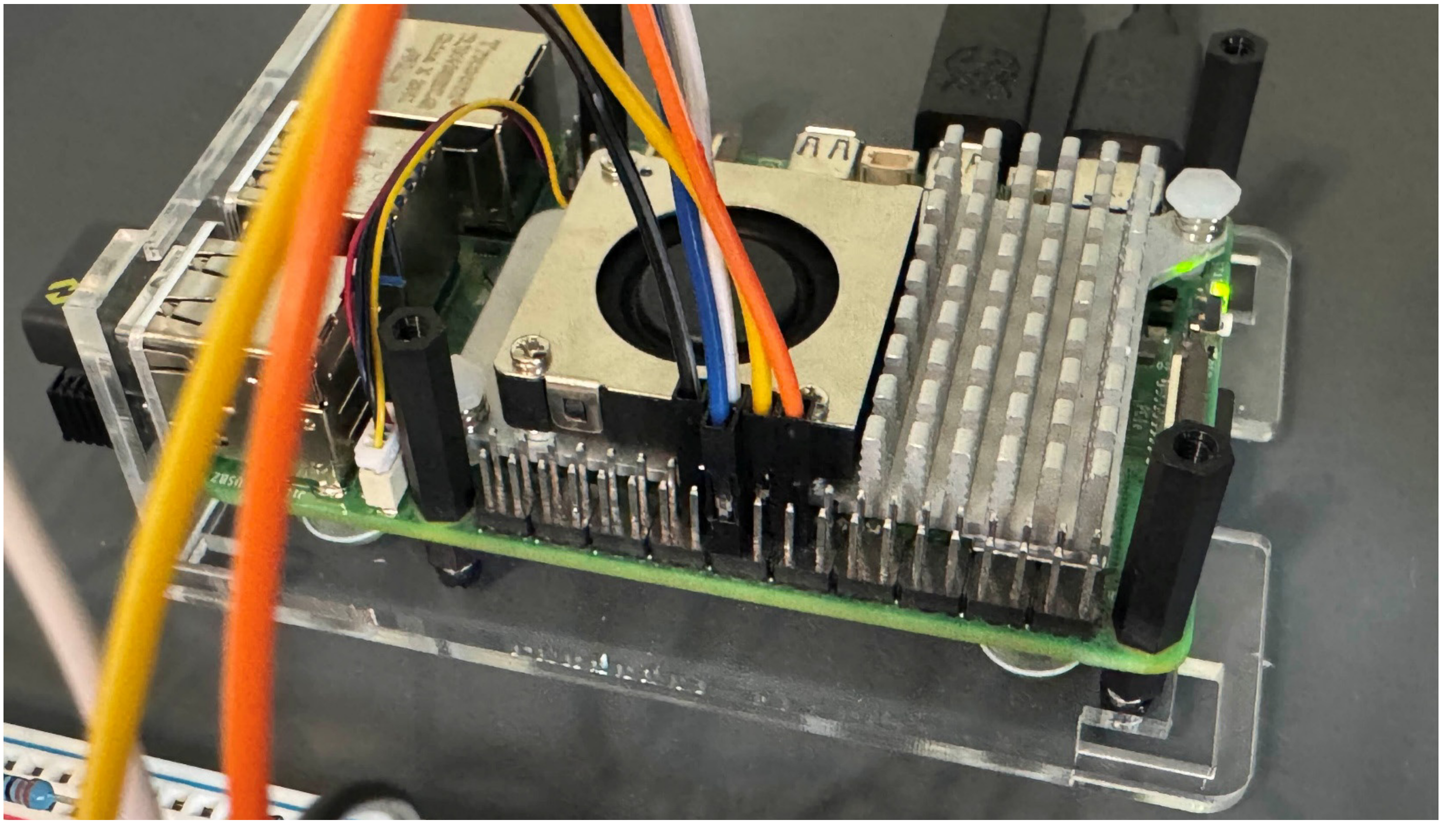
Notes:

1) Colours in the **Signal Name** column indicate the **colour for the jumper wires** used by yours truly.
2) See https://pinout.xyz/ for a **Raspberry Pi 5 pinout guide**.
3) For ADCA, though **ADCINA0 and ADCINA1** exist, they **share pins with DACA and DACB**, respectively. Thus, for this project, we used ADCINA2 and so on.
4) On the RPi, we chose **CE2 for SPI 1** for pinout simplicity.

TI LaunchPad (LAUNCHXL-F28379D)				Raspberry Pi 5		
	Signal Name	**Datasheet Pin #**	**Physical Pin #**	**Signal Name**	**RPi Pin #**	
	Ground	J2_10	GND	Ground	25	
SPI A	MISO / POCI	J2_4 / GPIO59	14	MISO	21 / GPIO9	SPI 0
	MOSI / PICO	J2_5 / GPIO58	15	MOSI	19 / GPIO10	
	SCLK	J1_7 / GPIO60	7	SCLK	23 / GPIO11	
	CS / PTE	J2_9 / GPIO61	19	CE0	24 / GPIO8	
SPI B	MISO / POCI	J6_4 / GPIO64	54	MISO	35 / GPIO19	SPI 1
	MOSI / PICO	J6_5 / GPIO63	55	MOSI	38 / GPIO20	
	SCLK	J5_7 / GPIO65	47	SCLK	40 / GPIO21	
	CS / PTE	J6_9 / GPIO66	59	CE2	36 / GPIO16	
ADCA	ADCINA2	J3_9	29			
	ADCINA3	J3_6	26			
	ADCINA4	J7_9	69			
ADCB	ADCINB2	J3_8	28			
	ADCINB3	J3_5	25			
	ADCINB4	J7_8	68			

Photos of the connections for SPI A, SPI 0, and ADCA can be found below.





CCS Programming

For this project, we used Code Composer Studio (CCS) 20.1.1. The steps to make the project are as follows.

1) Open CCS.
2) In the top left corner, choose File then Import Project(s)…
3) Click the Browse… button located next to the Search directory box.
4) Open a similar example to build your program off of. You may find useful examples in
    a. C:\ti\C2000Ware_5_04_00_00\driverlib\f2837xd\examples\cpu1
    b. C:\ti\C2000Ware_5_04_00_00\training\device\f2837xd
5) Rename project, .c, and .sysconfig files as desired.
6) Change compilation mode from RAM (MCU will not remember the program the next time you plug it in) to FLASH (MCU will yes remember the next time you plug it in). To do this, right click on the project name, go to Build Configurations, and choose the desired method. Note that FLASH will take a significantly longer time to load than RAM.
7) In the .sysconfig file, the EPWM instance controls the sampling frequency. Create a new one, and in EPWM Time Base, the Time Base Period parameter controls the sampling frequency. The Time Base Period for a desired sampling frequency using an **up counter** can be computed as follows:

    $$T_{\text{Period}} = \frac{f_{\text{sysclk}}}{(N_{\text{div}})(N_{\text{div,HS}})(f_{\text{sample}})}$$

    For example, if we want a sampling frequency of $f_{\text{sample}} = 1.25\ \text{kHz}$ with a Time Base Clock divider of $N_{\text{div}} = 4$ and a High Speed Clock Divider of $N_{\text{div,HS}} = 1$, we can use the fact that $f_{\text{sysclk}} = 50\ \text{MHz}$ and compute as follows:

    $$T_{\text{Period}} = \frac{50 \times 10^6}{(4)(1)(1.25 \times 10^3)} = 10000$$

    This means I enter 10000 into the Time Base Period parameter.

    To view the sampling frequency, open the EPWM Action Qualifier menu, then ePWMxA Output Configuration, then ePWMxA Event Output Configuration, then change the **ePWMxA Time Base counter equals zero parameter** to **Toggle the output pins**. This will toggle the ePWMxA pin every period, meaning if you are measuring with a scope, the **sampling frequency will be double the frequency you measure**. For example, for EPWM2A, this signal uses J4_8 / GPIO2 which corresponds to physical pin 38.
8) We are now ready to use the ePWM for ADC.