Nucleo Motor Control

Firmware Reference for Nucleos Assigned to Motor Control

Michigan Mars Rover Team

Updated 4/22/2020 11:32 AM

Introduction

Many Nucleos on the MRover 2020 rover (codenamed Lucy) are flashed with firmware that configures them exclusively for motor control. This document describes the behavior of only those Nucleos.

These Nucleos:

- Receive commands via I^2C bus
- Output PWM signals to control the motor
- Sample quadrature signals for relative encoders
- Communicate with specific SPI devices to monitor the absolute encoders

Overview

These Nucleos meet their functionality by internally splitting themselves into 6 distinct controller units, termed “channels”, labeled 0, 1, 2, 3, 4, and 5. Each channel is independent.

Each channel’s interfaces are exposed on the pins of the Nucleo. Each channel’s interfaces have different pin assignments.

Each channel has a set of registers (listed below), to dictate its behavior. Inputs from commands and various data modify the values in these registers. The channel continuously uses the values in these registers to produce an output signal for the motor controller.

Interfaces

Each channel has 4 effective interfaces:

- An I^2C interface to receive commands
  - Operates at up to 400 kHz (I^2C standard – fast mode)
  - Handles a standard set of I^2C transactions
- A PWM interface to control a motor
  - PWM pulse width and period is calculated based channel registers
- A quadrature signal interface to sample a relative encoder*
  - Samples the A and B channels at 8 MHz
  - Adds/Subtracts encoder counts to the quadEnc register upon observing relative encoder feedback
- A Receive-Only Master SPI interface to communicate with an absolute encoder*
  - Samples the SPI channel on command
  - Expects a 16 bit left-aligned integer representing an angle on MISO during communication
  - Overwrites the spiEnc register upon receiving absolute encoder data

* Only channels 0,1, and 2 have a quadrature signal and SPI interfaces

Registers

Each channel has its own set of registers that dictate the channel's behavior. These registers are not persistent (will not be saved under loss of power)

Register	Type	Value	Starting Value	Units
controlMode	8-bit value	0xFF – internal logic uses closed-loop control 0x00 – internal logic uses open-loop control	0	
dir	8-bit value	0x01 – PWM direction pin is set 0x00 – PWM direction pin is reset	0	
openSetpoint	16-bit unsigned int	This register is an input for open-loop control	0	µS
closedSetpoint	32-bit int	This register is an input for closed-loop control	0	encoder counts
FF	32-bit float	This register is an input for closed-loop control	0	
K_P	32-bit float	This register is an input for closed-loop control	0	
K_I	32-bit float	This register is an input for closed-loop control	0	
K_D	32-bit float	This register is an input for closed-loop control	0	
quadEnc	32-bit integer	This register is incremented/decremented based on quadrature interface feedback	0	encoder counts
spiEnc	16-bit unsigned int	This register is overwritten based on SPI interface data	0	encoder counts
pwmMode	8-bit value	0xFF – PWM interface operates in lock antiphase 0x00 – PWM interface operates in sign magnitude	0	
pwmMin	16-bit unsigned int	This register sets the PWM interface minimum pulse width	0	µS
pwmMax	16-bit unsigned int	This register sets the PWM interface maximum pulse width	65535	µS
pwmPeriod	16-bit unsigned int	This register sets the PWM interface period	65535	µS

In closed-loop control mode, the Nucleo will use output from a standard PID loop. The error for the PID loop is calculated by closedSetpoint – quadEnc. The PWM pulse width and PWM direction will automatically be adjusted for lock antiphase or sign magnitude PWM modes, based on pwmMode.

In open-loop control mode, the Nucleo will output openSetpoint as the PWM pulse width and dir for the PWM direction. The user must make necessary adjustments to openSetpoint and the dir values based on whether the motor requires lock anti-phase or sign magnitude PWM signals.

The Nucleo will automatically turn off the PWM interface if it hasn't received any I^2C transactions for 1000 ms.

The PWM interface signals of channels 0, 1, and 2 are generated by the *Nucleo F303RE* Advanced Timer "TIM1". Channels 3, 4, and 5 are generated by "TIM8". PWM interface signals generated by the same timer must have the same period. Therefore, the true PWM interface period of channels on the same timer will be set to the maximum pwmPeriod among all the channels on that timer.

Pin Assignments

On the MRover 2020 rover, these correspond to a *Nucleo F303RE Development Board*

Interface		Channel	Pin	If Disconnected
I^2C	SCL	All	PB8	All channels will not respond to commands. It is imperative this interface is connected.
	SDA	All	PB9	
PWM	Signal	0	PC0	No internal effect on the channel.
		1	PC1	
		2	PC2	
		3	PC6	
		4	PC7	
		5	PC8	
	Dir	0	PC10	
		1	PC11	
		2	PC12	
		3	PC13	
		4	PC14	
		5	PC15	
	~Dir	0	PB10	
		1	PB11	
		2	PB12	
		3	PB13	
		4	PB14	
		5	PB15	
	GND	All	GND (any)	
Quadrature	A	0	PA0	The affected channel will not sense relative encoder input. Use of only open loop control on the channel is recommended.
		1	PA6	
		2	PA11	
	B	0	PA1	
		1	PA4	
		2	PA12	
SPI	CLK	0,1,2	PB3	All channels will not sense absolute encoder input.
	MISO	0,1,2	PB4	
	CS	0	PB	The affected channel will not sense absolute encoder input.
		1	PB	
		2	PB	

I²C Behavior

Each channel behaves as an I²C slave. The first three bits of the channel's address correspond to the Nucleo's designated number (+1) on the rover, while the last four bits of the address correspond to the channel's number on the Nucleo.

For example, channel 2 on Nucleo 3 would have the I²C address of 100 0010.

The Nucleo expects transactions to adhere to the custom format defined below. For each transaction, the master should follow these steps:

1. Issue a START condition
2. Send the channel's 7-bit address followed by a Write bit
3. Send one byte to indicate the desired transaction type (see "Code" column below)
4. Send the appropriate number of data bytes
    a. If no data bytes should be sent, skip this step
5. Issue a RESTART condition OR issue a STOP condition and then a START condition
6. Send the channel's 7-bit address followed by a Read bit
7. Accept the appropriate number of data bytes
    a. If no data bytes should be accepted, skip steps 5-7
8. Issue a NACK
9. Issue a STOP condition

These transactions are not compliant with the smbus standard. It is recommended to use the *open(), ioctl(), read(),* and *write()* functions defined in *linux/i2c-dev.h* to control the master's I²C interface.

Data bytes that are sent to the channel will overwrite the appropriate registers in that channel. Data bytes received from the channel will contain the current value of the appropriate register in that channel.

Transaction Type	Code	Registers Written To				Registers Read From
OPEN	0x10	3 Bytes				0 Bytes
		1 Byte	2 Bytes			
		dir	openSetpoint			
OPEN_PLUS	0x1F	3 Bytes				4 Bytes
		1 Byte	2 Bytes			
		dir	openSetpoint			quadEnc
CLOSED	0x20	8 Bytes				0 Bytes
		4 Bytes	4 Bytes			
		closedSetpoint	FF			
CLOSED_PLUS	0x2F	8 Bytes				4 Bytes
		4 Bytes	4 Bytes			
		closedSetpoint	FF			quadEnc
CONFIG_PWM	0x30	7 Bytes				0 Bytes
		1 Byte	2 Bytes	2 Bytes	2 Bytes	
		pwmMode	pwmMin	pwmMax	pwmPeriod	
CONFIG_K	0x3F	12 Bytes				0 Bytes
		4 Bytes	4 Bytes	4 Bytes		
		KP	KI	KD		
QUAD_ENC	0x40	0 Bytes				4 Bytes
						quadEnc
QUAD_ENC_ADJUST	0x4F	4 Bytes				0 Bytes
		quadEnc				
SPI_ENC	0x50	0 Bytes				2 Bytes
						spiEnc

Further Reading

Official Documents:

- From STMicroelectronics
  - STMF303xD/E Datasheet
  - RM0316 Reference Manual
  - UM1786 User Manual
- From CTRE
  - Talon SRX User's Guide
- From MPS
  - MagAlpha MA730 Datasheet