PmodHB5 Programmer's Reference Manual

Revision: December 15, 2011

www.digilentinc.com

1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Introduction

This document describes the programming interface to the PmodHB5 library that is included as part of the PmodLib library. It describes the capabilities of the PmodHB5 library and all the API functions used to access its features.

The purpose this library is to offer supporting functions for interfacing with the PmodHB5 hardware on either the Cerebot 32MX4 or 32MX7 microcontroller.

The primary use for this library is operating DC motors via the PmodHB5 utilizing pulse width modulation. See the Digilent PmodHB5 2A H-Bridge Reference Manual for additional operating and signaling information.

PmodHB5 Basic API Functions

uint8_t PmodHB5ChangeDirection(HBRIDGE *hBridge)

Parameters
    HBRIDGE *hBridge - HBRIDGE struct

Returns
    uint8_t - 1 if direction changed, 0 otherwise

The HBRIDGE.currentDirection and HBRIDGE.newDirection fields are compared, if they are different the duty cycle of the OC specified in HBRIDGE.ocChannel is set to 0%, if HBRIDGE.rpm == 0 then the direction bit is toggled, reversing the DC motor direction. Waiting for the DC motor RPM to slow to 0 prior to changing direction, prevents current from flowing back into the HB5, setting the duty cycle to 0% prevents short circuits in the hbridge when changing directions.

This function should be called repeatedly until returning 1 indicating that conditions for direction change have been met and the direction change applied. PmodHB5getQEncRPM should be called prior attempting direction change (except when RPM is known to be 0, for example when the module is first initialized) for a least one RPM polling period as specified in PmodHB5getQEncRPM to ensure an accurate RPM value. If hBridge->newDirection == hBridge->currentDirection no change will be made and a value of one will be returned.

void PmodHB5getQEncRPM(HBRIDGE *hBridge,uint32_t pollsPerSec,uint32_t pollDivisor)

Parameters

HBRIDGE *hBridge - HBRIDGE struct
uint32_t pollsPerSec - number of quadrature

Returns

rpm, quadrature encoding

The time period used for calculating RPM is pollPerSec/pollDivisor, Example: Calculate RPM every 200ms: calling PmodHB5getQEncRPM 1000 times per second, set pollsPerSec = 1000 and pollDivisor = 5, resulting in a polling period of 200ms, the current number of polls for this time period is stored in HBRIDGE.pollCount and is incremented every time this function is called. When a change in state is detected in quadrature position is detected, HBRIDGE.pulseCount is incremented. Once HBRIDGE.pollCount reaches the determined polling period of 200ms (pollCount == 200), RPM is calculated. Quadrature encoding is stored in HBRIDGE.quadPos every time this function is called.

void PmodHB5SetDCInitialDirection(HBRIDGE *hBridge)

Parameters

HBRIDGE *hBridge - HBRIDGE struct

Returns

Upon direction change, new direction is stored in hBridge->currentDirection

Sets the initial duty cycle to 0% and direction of the DC motor based on HBRIDGE.newDirection. This function should be used only when the RPM is known to be 0 (example: setup and initialization) otherwise the PmodHB5 could be damaged.

void PmodHB5SetDCPWMDutyCycle(uint16_t dutyCycle,uint8_t oc)

Parameters

uint16_t dutyCycle - OCx duty cycle as a percentage of PRx
uint8_t oc - OC channel

Returns

none

Sets the PWM duty cycle of the specified output comparator. Duty cycle calculated as percentage of PRx should be passed example OC2 50%: dutyCycle = PR2/2

PmodHB5 Additional Information

The PMODHB5_DIR enum denotes motor direction.

```
typedef enum
{
        PMOD_HB5_DIR_CW,    //Motor direction clockwise
        PMOD_HB5_DIR_CCW   //Motor direction counter clockwise
}PMODHB5_DIR;
```

The HBRIDGE struct represents a PmodHB5 module maintaining state for the following: hardware configuration, RPM, direction, current quadrature position.

```
typedef struct
{
        uint8_t quadPos;                    //Quadrature encoding
        uint16_t sensorAport;               //Port mask for sensor A
        uint16_t sensorAportBit;            //Bit mask for sensor A
        uint16_t sensorBport;               //Port mask for sensor B
        uint16_t sensorBportBit;            //Bit mask for sensorB
        uint16_t directionPort;             //Port mask for direction
        uint16_t directionPortBit;          //Bit mask for direction
        uint32_t pulseCount;                 //Quadrature encoding state changes/time period
                                             //detected
        uint32_t pollCount;                 //Current number of quadrature encoding state polls
                                            // for current polling period (used in RPM calc)
        uint16_t rpm;                       //Current motor RPM
        uint16_t prevRpm;                   //Previous motor RPM
        PMODHB5_DIR newDirection;           //New motor direction
        PMODHB5_DIR currentDirection;       //Current motor direction
        uint8_t ocChannel;                  //Output comparator channel, (1-5)
}HBRIDGE;
```

PmodHB5 Use Example

The following example operates a single PmodHB5 using a timer to generate pulse width modulation to drive the enable pin at 50% duty cycle. Pressing BTN1 or BTN2 will change motor direction. When a motor direction change has been requested, duty cycle is set to 0%, once the RPM drops to zero, the direction bit is changed and duty cycle set to 50%.

```
/* ------------------------------------------------------------- */
/*              PIC32 Configuration Settings              */
/* ------------------------------------------------------------- */
#pragma config FPLLMUL = MUL_20, FPLLIDIV = DIV_2, FPLLODIV = DIV_1
#pragma config FWDTEN = OFF
#pragma config POSCMOD = HS, FNOSC = PRIPLL
#pragma config FPBDIV = DIV_2

#include <stdint.h>
#include <plib.h>
#include <pmodlib.h>

#define SYSTEM_CLOCK (80000000L)                    //System clock speed (8 MHz Crystal/ FPLLIDIV * FPLLMUL / FPLLODIV)
#define PB_DIV              2                       //Peripheral bus divisor
#define PRESCALE 256                                //Timer Prescale
#define PB_CLOCK (SYSTEM_CLOCK/PB_DIV)              //Peripheral bus clock
#define TOGGLES_PER_SEC  6000                       //Timer 1 toggles per second
#define T1_TICK (SYSTEM_CLOCK/PB_DIV/PRESCALE/TOGGLES_PER_SEC)

uint8_t directionChangeComplete = 1;
HBRIDGE hBridge;

void initHB5()
{
        PORTSetPinsDigitalOut(IOPORT_D, BIT_7); //HB5 Direction
        PORTSetPinsDigitalOut(IOPORT_D, BIT_1); //HB5 Enable
        PORTSetPinsDigitalIn(IOPORT_D, BIT_9);   //HB5 Sensor A
        PORTSetPinsDigitalIn(IOPORT_C, BIT_1);   //HB5 Sensor B

        hBridge.sensorAport = IOPORT_D;
        hBridge.sensorAportBit = BIT_9;
        hBridge.sensorBport = IOPORT_C;
        hBridge.sensorBportBit = BIT_1;
        hBridge.directionPort = IOPORT_D;
        hBridge.directionPortBit = BIT_7;
        hBridge.currentDirection = PMOD_HB5_DIR_CW;
        hBridge.newDirection = PMOD_HB5_DIR_CW;
        hBridge.ocChannel = 2;

        //Set initial motor direction
        PmodHB5SetDCInitialDirection(&hBridge);

        //PWM for motor
         OpenOC2(OC_ON | OC_TIMER2_SRC | OC_PWM_FAULT_PIN_DISABLE, 0, 0);
         OpenTimer2(T2_ON | T2_PS_1_256, SYSTEM_CLOCK/PB_DIV/PRESCALE/(TOGGLES_PER_SEC/6));

        //Timer for checking RPM and direction change
        OpenTimer1(T1_ON | T1_SOURCE_INT | T1_PS_1_256, T1_TICK);
         ConfigIntTimer1(T1_INT_ON | T1_INT_PRIOR_2);

        INTEnableSystemMultiVectoredInt();

}
```

```
void getQuadEncodingChangeDir()
{
        //Get the quadrature encoding and calculate RPM
        PmodHB5getQEncRPM(&hBridge,TOGGLES_PER_SEC,5);

        //Change direction, directionChangeComplete is set to 1 if it is complete, 0 otherwise
        directionChangeComplete = PmodHB5ChangeDirection(&hBridge);
}

int main(void)
{
        uint8_t portBits = 0;
        PORTSetPinsDigitalIn(IOPORT_A, BIT_6|BIT_7); //Enable digital IO from BTN1 and BTN2
        initHB5();

        while(1)
        {
                portBits = PORTReadBits(IOPORT_A,BIT_6|BIT_7); //Read BTN1 and BTN2 bits
                //Change motor direction on BTN1 or BTN2 press
                if(portBits & BIT_6)
                {
                        hBridge.newDirection = PMOD_HB5_DIR_CW;
                }
                else if(portBits & BIT_7)
                {
                        hBridge.newDirection = PMOD_HB5_DIR_CCW;
                }
                //Set duty cycle to 50% if the direction change has completed
                if(directionChangeComplete)
                {
                         PmodHB5SetDCPWMDutyCycle(PR2/2,2);
                }
        }

}

void __ISR(_TIMER_1_VECTOR, ipl2) Timer1Handler(void)
{
        getQuadEncodingChangeDir();
        INTClearFlag(INT_T1);

}
```