SYSTEM\_THREAD\_ENTRY

(**PRIMERA** **VERSION 180719**)

(**SEGUNDA** **VERSION 190719**)

**#include** <system\_thread.h>

**#define** FREC\_CLK\_TIMER (120000000U) // Frecuency of the timer = 240 MHz

uint32\_t frec=200;

uint16\_t u16PwmPercent=30, u16ADC\_Data;

int16\_t error, ie, ie\_n1=0, u;

uint16\_t rpm\_sp=500,rpm=0, PwmPorcent;

uint8\_t kp=4, ki=4;

uint32\_t counts = 0, counts1 = 0, counts\_avg[4]={0};

ioport\_level\_t Pin,Led1=1, Led2=1;

**static** uint32\_t frec\_read = 0;

uint16\_t data[2] = {0};

**void** **control\_motor**(**void**);

**void** **control\_motor**(**void**){

g\_adc0.p\_api->read(g\_adc0.p\_ctrl, *ADC\_REG\_CHANNEL\_0*, &u16ADC\_Data);

rpm\_sp= (uint16\_t)((u16ADC\_Data \* 506)/245 + 294); // Convert data from ADC(13-245) to rpm (320-800)

rpm = (uint16\_t)(4 \* frec\_read); // conversion of frec to rpm (frec\*60/15)

error = (int16\_t)(rpm\_sp - rpm);

//ie = (int16\_t)((ki \* error) + ie\_n1); // For Gains greater than 1

ie = (int16\_t)((error/ki) + ie\_n1); // For Gains smaller than 1

ie\_n1 = ie;

ie = ie / 100;

**if**(ie > 100) ie = 100;

**if**(ie < 0) ie = 0;

//u = (int16\_t)((kp \* error) + ie); // For Gains greater than 1

u = (int16\_t)((error/kp) + ie); // For Gains smaller than 1

**if**(u > 100) u = 100;

**if**(u < 65) u = 65;

PwmPorcent = (uint16\_t) (100 - u); // Inverse logic

g\_timer2.p\_api->dutyCycleSet(g\_timer2.p\_ctrl, PwmPorcent, *TIMER\_PWM\_UNIT\_PERCENT*, 0); //Send the result of the Control to the Motor Driver

data[0] = u;//;

data[1] = rpm;

}

**void** **system\_thread\_entry**(**void**)

{

g\_timer0.p\_api->open(g\_timer0.p\_ctrl, g\_timer0.p\_cfg); //Timer 0 to measure the period of the sensor

g\_timer2.p\_api->open(g\_timer2.p\_ctrl, g\_timer2.p\_cfg); //Timer 2 to generate the PWM at 10 KHz

g\_external\_irq4.p\_api->open(g\_external\_irq4.p\_ctrl, g\_external\_irq4.p\_cfg); //Int 1 to read the counts of timer 1

g\_timer0.p\_api->start(g\_timer0.p\_ctrl);

g\_timer2.p\_api->start(g\_timer2.p\_ctrl);

g\_adc0.p\_api->open(g\_adc0.p\_ctrl, g\_adc0.p\_cfg); (**CAMBIO DE LUGAR ANTES DEL ADC READ**)

g\_adc0.p\_api->scanCfg(g\_adc0.p\_ctrl, g\_adc0.p\_channel\_cfg);

g\_adc0.p\_api->scanStart(g\_adc0.p\_ctrl);

**while**(1)

{

//g\_ioport.p\_api->pinWrite(IOPORT\_PORT\_01\_PIN\_14, IOPORT\_LEVEL\_HIGH); //Pin used to check algorithm time

//Pin = !Pin; g\_ioport.p\_api->pinWrite(IOPORT\_PORT\_01\_PIN\_14, Pin); //Pin used to check the sample time

control\_motor();

g\_timer2.p\_api->periodSet(g\_timer2.p\_ctrl, frec, *TIMER\_UNIT\_FREQUENCY\_HZ*); //used to change the frec manually

//g\_timer2.p\_api->dutyCycleSet(g\_timer2.p\_ctrl, u16PwmPercent, TIMER\_PWM\_UNIT\_PERCENT, 0); //used to change the dutycycle manually

//g\_ioport.p\_api->pinWrite(IOPORT\_PORT\_01\_PIN\_14, IOPORT\_LEVEL\_LOW); //Pin used to check algorithm time

tx\_queue\_flush(&Message\_Queue);

tx\_queue\_send(&Message\_Queue, data, TX\_NO\_WAIT);

tx\_thread\_sleep(10);

}

}

**void** **external\_irq4\_callback**(external\_irq\_callback\_args\_t \*p\_args)

{

SSP\_PARAMETER\_NOT\_USED(p\_args);

g\_timer0.p\_api->counterGet(g\_timer0.p\_ctrl, &counts); // read the period clocks

g\_timer0.p\_api->reset(g\_timer0.p\_ctrl); // restart the Timer for a new period measurement

counts\_avg[0] = (counts + counts\_avg[1] + counts\_avg[2] + counts\_avg[3])/4; // Counts average of the last 4 measurements

frec\_read = FREC\_CLK\_TIMER/counts\_avg[0]; // conversion of time to frec

counts\_avg[3] = counts\_avg[2];

counts\_avg[2] = counts\_avg[1];

counts\_avg[1] = counts;

}