**FLOWCODE**

A picture containing chart

Description automatically generated

**C CODE**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*

//\*\* Source name: C:\Users\madar\Documents\drone\_flowcode\_v2.fcfx

//\*\* Title:

//\*\* Description:

//\*\* Device: PIC.16F.16F877A

//\*\*

//\*\* Generated by: Flowcode v8.2.2.15

//\*\* Date: Friday, June 25, 2021 15:11:05

//\*\* Users: 1

//\*\* Registered to:

//\*\* License key: >= 2

//\*\*

//\*\* https://www.matrixtsl.com

//\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define MX\_PIC

#define MX\_CAL\_PIC

#define MX\_CLK\_SPEED 19660800

#define FCP\_NULL Unconnected\_Port

#include "Arduino.h"

#include "config.h"

#include "def.h"

#include "types.h"

#include "MultiWii.h"

#include "Alarms.h"

#include <xc.h>

#include <math.h>

//Chip Configuration Settings

\_\_PROG\_CONFIG(0x1, 0x3F72);

void initializeSoftPWM(void);

#define FCVsz\_MOTOR 5

#define FCVsz\_RCCOMMAND 6

#define FCVsz\_SERV 6

#define FCVsz\_AMPERES 64

#define FCV\_FALSE (0)

#define FCV\_TRUE (1)

MX\_GLOBAL MX\_UINT8 FCV\_MOTOR[FCVsz\_MOTOR];

MX\_GLOBAL MX\_UINT8 FCV\_ICR3;

MX\_GLOBAL MX\_UINT8 FCV\_PASSTHRU\_MODE;

MX\_GLOBAL MX\_UINT8 FCV\_ICR1;

MX\_GLOBAL MX\_UINT8 FCV\_SERVO\_1\_PINMODE;

MX\_GLOBAL MX\_UINT8 FCV\_PROMICRO = (0x1);

MX\_GLOBAL MX\_UINT8 FCV\_RCCOMMAND[FCVsz\_RCCOMMAND];

MX\_GLOBAL MX\_UINT8 FCV\_SERVO;

MX\_GLOBAL MX\_UINT8 FCV\_FLAPPERONS;

MX\_GLOBAL MX\_UINT8 FCV\_PRI\_SERVO\_FROM = (0x1);

MX\_GLOBAL MX\_UINT8 FCV\_MEGA = (0x1);

MX\_GLOBAL MX\_UINT8 FCV\_OCR1A = (0x0);

MX\_GLOBAL MX\_UINT8 FCV\_TCCR1B;

MX\_GLOBAL MX\_UINT8 FCV\_SERV[FCVsz\_SERV];

MX\_GLOBAL MX\_UINT8 FCV\_TCCR1A;

MX\_GLOBAL MX\_UINT8 FCV\_NUMBER\_MOTOR = (0x1);

MX\_GLOBAL MX\_UINT8 FCV\_WGM13;

MX\_GLOBAL MX\_UINT8 FCV\_CS10;

MX\_GLOBAL MX\_UINT8 FCV\_TC4H;

MX\_GLOBAL MX\_UINT8 FCV\_WGM12;

MX\_GLOBAL MX\_UINT8 FCV\_WGM11;

MX\_GLOBAL MX\_UINT8 FCV\_LOG\_VALUES;

MX\_GLOBAL MX\_UINT8 FCV\_PRI\_SERVO\_TO = (0x1);

MX\_GLOBAL MX\_UINT8 FCV\_ATOMICPWM\_PIN6\_HIGHSTATE = (0x1);

MX\_GLOBAL MX\_UINT8 FCV\_EXT\_MOTOR\_RANGE = (0x1);

MX\_GLOBAL MX\_UINT8 FCV\_TCCR3B;

MX\_GLOBAL MX\_UINT8 FCV\_OCR4C;

MX\_GLOBAL MX\_UINT8 FCV\_HWPWM6 = (0x1);

MX\_GLOBAL MX\_UINT8 FCV\_ATOMICPWM\_PIN5\_HIGHSTATE = (0x1);

MX\_GLOBAL MX\_UINT8 FCV\_CS31;

MX\_GLOBAL MX\_UINT8 FCV\_TCCR3A;

MX\_GLOBAL MX\_UINT8 FCV\_WGM33;

MX\_GLOBAL MX\_UINT8 FCV\_CS30;

MX\_GLOBAL MX\_UINT8 FCV\_WGM32;

MX\_GLOBAL MX\_UINT8 FCV\_WGM31;

MX\_GLOBAL MX\_UINT16 FCV\_WGM30;

MX\_GLOBAL MX\_UINT8 FCV\_PROMINI = (0x1);

MX\_GLOBAL MX\_UINT8 FCV\_AMPERES[FCVsz\_AMPERES];

#if (NUMBER\_MOTOR > 4)

//for HEX Y6 and HEX6/HEX6X/HEX6H flat for promini

volatile uint8\_t atomicPWM\_PIN5\_lowState;

volatile uint8\_t atomicPWM\_PIN5\_highState;

volatile uint8\_t atomicPWM\_PIN6\_lowState;

volatile uint8\_t atomicPWM\_PIN6\_highState;

#endif

#if (NUMBER\_MOTOR > 6)

//for OCTO on promini

volatile uint8\_t atomicPWM\_PINA2\_lowState;

volatile uint8\_t atomicPWM\_PINA2\_highState;

volatile uint8\_t atomicPWM\_PIN12\_lowState;

volatile uint8\_t atomicPWM\_PIN12\_highState;

#endif

#else

#if (NUMBER\_MOTOR > 4)

//for HEX Y6 and HEX6/HEX6X/HEX6H and for Promicro

volatile uint16\_t atomicPWM\_PIN5\_lowState;

volatile uint16\_t atomicPWM\_PIN5\_highState;

volatile uint16\_t atomicPWM\_PIN6\_lowState;

volatile uint16\_t atomicPWM\_PIN6\_highState;

#endif

#if (NUMBER\_MOTOR > 6)

//for OCTO on Promicro

volatile uint16\_t atomicPWM\_PINA2\_lowState;

volatile uint16\_t atomicPWM\_PINA2\_highState;

volatile uint16\_t atomicPWM\_PIN12\_lowState;

volatile uint16\_t atomicPWM\_PIN12\_highState;

#endif

#endif

#if defined(SERVO)

#if defined(HW\_PWM\_SERVOS)

// hw servo pwm does not need atomicServo[]

#elif defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6))

#if defined(AIRPLANE) || defined(HELICOPTER)

// To prevent motor to start at reset. atomicServo[7]=5 or 249 if reversed servo

volatile uint8\_t atomicServo[8] = {125,125,125,125,125,125,125,5};

#else

volatile uint8\_t atomicServo[8] = {125,125,125,125,125,125,125,125};

#endif

#else

#if defined(AIRPLANE)|| defined(HELICOPTER)

// To prevent motor to start at reset. atomicServo[7]=5 or 249 if reversed servo

volatile uint16\_t atomicServo[8] = {8000,8000,8000,8000,8000,8000,8000,320};

#else

volatile uint16\_t atomicServo[8] = {8000,8000,8000,8000,8000,8000,8000,8000};

#endif

#endif

#endif

void initializeSoftPWM(void);

#if defined(SERVO)

void initializeServo();

#endif

#if defined(PROMINI)

#if defined(NRF24\_RX)

uint8\_t PWM\_PIN[8] = {9,6,5,3,A2,12,10,11}; //use pin 6 and 5 instead of 10 and 11 - only valid for quad!

//uint8\_t PWM\_PIN[8] = {3,5,6,7,A2,12,7,11}; //use pin 6 and 5 instead of 10 and 11 - only valid for quad!

#else

uint8\_t PWM\_PIN[8] = {9,10,11,3,6,5,A2,12}; //for a quad+: rear,right,left,front

#endif

#endif

#if defined(PROMICRO)

#if !defined(HWPWM6)

#if defined(TEENSY20)

uint8\_t PWM\_PIN[8] = {14,15,9,12,22,18,16,17}; //for a quad+: rear,right,left,front

#elif defined(A32U4\_4\_HW\_PWM\_SERVOS)

uint8\_t PWM\_PIN[8] = {6,9,10,11,5,13,SW\_PWM\_P3,SW\_PWM\_P4}; //

#else

uint8\_t PWM\_PIN[8] = {9,10,5,6,4,A2,SW\_PWM\_P3,SW\_PWM\_P4}; //for a quad+: rear,right,left,front

#endif

#else

#if defined(TEENSY20)

uint8\_t PWM\_PIN[8] = {14,15,9,12,4,10,16,17}; //for a quad+: rear,right,left,front

#elif defined(A32U4\_4\_HW\_PWM\_SERVOS)

uint8\_t PWM\_PIN[8] = {6,9,10,11,5,13,SW\_PWM\_P3,SW\_PWM\_P4}; //

#else

uint8\_t PWM\_PIN[8] = {9,10,5,6,11,13,SW\_PWM\_P3,SW\_PWM\_P4}; //for a quad+: rear,right,left,front

#endif

#endif

#endif

#if defined(MEGA)

uint8\_t PWM\_PIN[8] = {3,5,6,2,7,8,9,10}; //for a quad+: rear,right,left,front //+ for y6: 7:under right 8:under left

#endif

#if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6)) || (defined(MEGA) && defined(MEGA\_HW\_PWM\_SERVOS))

#if defined(NRF24\_RX)

volatile uint8\_t atomicPWM\_PIN5\_lowState;

volatile uint8\_t atomicPWM\_PIN5\_highState;

volatile uint8\_t atomicPWM\_PIN6\_lowState;

volatile uint8\_t atomicPWM\_PIN6\_highState;

#endif

void initOutput() {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* mark all PWM pins as Output \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for(uint8\_t i=0;i<NUMBER\_MOTOR;i++) {

pinMode(PWM\_PIN[i],OUTPUT);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Specific PWM Timers & Registers for the MEGA's \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#if defined(MEGA)

#if (NUMBER\_MOTOR > 0)

// init 16bit timer 3

TCCR3A |= (1<<WGM31); // phase correct mode

TCCR3A &= ~(1<<WGM30);

TCCR3B |= (1<<WGM33);

TCCR3B &= ~(1<<CS31); // no prescaler

ICR3 |= 0x3FFF; // TOP to 16383;

TCCR3A |= \_BV(COM3C1); // connect pin 3 to timer 3 channel C

#endif

#if (NUMBER\_MOTOR > 1)

TCCR3A |= \_BV(COM3A1); // connect pin 5 to timer 3 channel A

#endif

#if (NUMBER\_MOTOR > 2)

// init 16bit timer 4

TCCR4A |= (1<<WGM41); // phase correct mode

TCCR4A &= ~(1<<WGM40);

TCCR4B |= (1<<WGM43);

TCCR4B &= ~(1<<CS41); // no prescaler

ICR4 |= 0x3FFF; // TOP to 16383;

TCCR4A |= \_BV(COM4A1); // connect pin 6 to timer 4 channel A

#endif

#if (NUMBER\_MOTOR > 3)

TCCR3A |= \_BV(COM3B1); // connect pin 2 to timer 3 channel B

#endif

#if (NUMBER\_MOTOR > 4)

TCCR4A |= \_BV(COM4B1); // connect pin 7 to timer 4 channel B

TCCR4A |= \_BV(COM4C1); // connect pin 8 to timer 4 channel C

#endif

#if (NUMBER\_MOTOR > 6)

// timer 2 is a 8bit timer so we cant change its range

TCCR2A |= \_BV(COM2B1); // connect pin 9 to timer 2 channel B

TCCR2A |= \_BV(COM2A1); // connect pin 10 to timer 2 channel A

#endif

#endif

#if defined(PROMICRO)

#if defined(EXT\_MOTOR\_64KHZ) || defined(EXT\_MOTOR\_32KHZ) || defined(EXT\_MOTOR\_16KHZ) || defined(EXT\_MOTOR\_8KHZ)

TCCR1A = (1<<WGM11);

TCCR1B = (1<<WGM13) | (1<<WGM12) | (1<<CS10);

TCCR3A = (1<<WGM31);

TCCR3B = (1<<WGM33) | (1<<WGM32) | (1<<CS30);

#if defined(EXT\_MOTOR\_64KHZ)

ICR1 = 0x00FF; // TOP to 255;

ICR3 = 0x00FF; // TOP to 255;

TC4H = 0x00;

OCR4C = 0xFF; // phase and frequency correct mode & top to 255

TCCR4B = (1<<CS40); // prescaler to 1

#elif defined(EXT\_MOTOR\_32KHZ)

ICR1 = 0x01FF; // TOP to 511;

ICR3 = 0x01FF; // TOP to 511;

TC4H = 0x01;

OCR4C = 0xFF; // phase and frequency correct mode & top to 511

TCCR4B = (1<<CS40); // prescaler to 1

#elif defined(EXT\_MOTOR\_16KHZ)

ICR1 = 0x03FF; // TOP to 1023;

ICR3 = 0x03FF; // TOP to 1023;

TC4H = 0x03;

OCR4C = 0xFF; // phase and frequency correct mode & top to 1023

TCCR4B = (1<<CS40); // prescaler to 1

#elif defined(EXT\_MOTOR\_8KHZ)

ICR1 = 0x07FF; // TOP to 2046;

ICR3 = 0x07FF; // TOP to 2046;

TC4H = 0x3;

OCR4C = 0xFF; // phase and frequency correct mode

TCCR4B = (1<<CS41); // prescaler to 2

#endif

TCCR1A |= \_BV(COM1A1); // connect pin 9 to timer 1 channel A

TCCR1A |= \_BV(COM1B1); // connect pin 10 to timer 1 channel B

TCCR3A |= \_BV(COM3A1); // connect pin 5 to timer 3 channel A

TCCR4D = 0;

TCCR4C |= (1<<COM4D1)|(1<<PWM4D); // connect pin 6 to timer 4 channel D

#else

#if (NUMBER\_MOTOR > 0) && ( !defined(A32U4\_4\_HW\_PWM\_SERVOS) )

TCCR1A |= (1<<WGM11); // phase correct mode & no prescaler

TCCR1A &= ~(1<<WGM10);

TCCR1B &= ~(1<<WGM12) & ~(1<<CS11) & ~(1<<CS12);

TCCR1B |= (1<<WGM13) | (1<<CS10);

ICR1 |= 0x3FFF; // TOP to 16383;

TCCR1A |= \_BV(COM1A1); // connect pin 9 to timer 1 channel A

#endif

#if (NUMBER\_MOTOR > 1)

TCCR1A |= \_BV(COM1B1); // connect pin 10 to timer 1 channel B

#endif

#if (NUMBER\_MOTOR > 2)

#if !defined(HWPWM6) // timer 4A

TCCR4E |= (1<<ENHC4); // enhanced pwm mode

TCCR4B &= ~(1<<CS41); TCCR4B |= (1<<CS42)|(1<<CS40); // prescaler to 16

TCCR4D |= (1<<WGM40); TC4H = 0x3; OCR4C = 0xFF; // phase and frequency correct mode & top to 1023 but with enhanced pwm mode we have 2047

TCCR4A |= (1<<COM4A0)|(1<<PWM4A); // connect pin 5 to timer 4 channel A

#else // timer 3A

TCCR3A |= (1<<WGM31); // phase correct mode & no prescaler

TCCR3A &= ~(1<<WGM30);

TCCR3B &= ~(1<<WGM32) & ~(1<<CS31) & ~(1<<CS32);

TCCR3B |= (1<<WGM33) | (1<<CS30);

ICR3 |= 0x3FFF; // TOP to 16383;

TCCR3A |= \_BV(COM3A1); // connect pin 5 to timer 3 channel A

#endif

#endif

#if (NUMBER\_MOTOR > 3) || ( (NUMBER\_MOTOR > 0) && defined(A32U4\_4\_HW\_PWM\_SERVOS) )

#if defined(HWPWM6)

TCCR4E |= (1<<ENHC4); // enhanced pwm mode

TCCR4B &= ~(1<<CS41); TCCR4B |= (1<<CS42)|(1<<CS40); // prescaler to 16

TCCR4D |= (1<<WGM40); TC4H = 0x3; OCR4C = 0xFF; // phase and frequency correct mode & top to 1023 but with enhanced pwm mode we have 2047

#endif

TCCR4C |= (1<<COM4D1)|(1<<PWM4D); // connect pin 6 to timer 4 channel D

#endif

#if (NUMBER\_MOTOR > 4)

#if defined(HWPWM6)

TCCR1A |= \_BV(COM1C1); // connect pin 11 to timer 1 channel C

TCCR4A |= (1<<COM4A1)|(1<<PWM4A); // connect pin 13 to timer 4 channel A

#else

initializeSoftPWM();

#endif

#endif

#if (NUMBER\_MOTOR > 6)

#if defined(HWPWM6)

initializeSoftPWM();

#endif

#endif

#endif

#endif

#if defined(SERVO)

#if defined(PRI\_SERVO\_FROM) && defined(SEC\_SERVO\_FROM)

#if PRI\_SERVO\_FROM < SEC\_SERVO\_FROM

#define SERVO\_START PRI\_SERVO\_FROM

#else

#define SERVO\_START SEC\_SERVO\_FROM

#endif

#else

#if defined(PRI\_SERVO\_FROM)

#define SERVO\_START PRI\_SERVO\_FROM

#endif

#if defined(SEC\_SERVO\_FROM)

#define SERVO\_START SEC\_SERVO\_FROM

#endif

#endif

#if defined(PRI\_SERVO\_TO) && defined(SEC\_SERVO\_TO)

#if PRI\_SERVO\_TO > SEC\_SERVO\_TO

#define SERVO\_END PRI\_SERVO\_TO

#else

#define SERVO\_END SEC\_SERVO\_TO

#endif

#else

#if defined(PRI\_SERVO\_TO)

#define SERVO\_END PRI\_SERVO\_TO

#endif

#if defined(SEC\_SERVO\_TO)

#define SERVO\_END SEC\_SERVO\_TO

#endif

#endif

#endif

void writeServos() {

#if defined(SERVO)

#if defined(PRI\_SERVO\_FROM) && !defined(HW\_PWM\_SERVOS) // write primary servos

for(uint8\_t i = (PRI\_SERVO\_FROM-1); i < PRI\_SERVO\_TO; i++){

#if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6)) || (defined(MEGA) && defined(MEGA\_HW\_PWM\_SERVOS))

atomicServo[i] = (servo[i]-1000)>>2;

#else

atomicServo[i] = (servo[i]-1000)<<4;

#endif

}

#endif

#if defined(SEC\_SERVO\_FROM) && !defined(HW\_PWM\_SERVOS) // write secundary servos

#if (defined(SERVO\_TILT)|| defined(SERVO\_MIX\_TILT)) && defined(MMSERVOGIMBAL)

// Moving Average Servo Gimbal by Magnetron1

static int16\_t mediaMobileServoGimbalADC[3][MMSERVOGIMBALVECTORLENGHT];

static int32\_t mediaMobileServoGimbalADCSum[3];

static uint8\_t mediaMobileServoGimbalIDX;

uint8\_t axis;

mediaMobileServoGimbalIDX = ++mediaMobileServoGimbalIDX % MMSERVOGIMBALVECTORLENGHT;

for (axis=(SEC\_SERVO\_FROM-1); axis < SEC\_SERVO\_TO; axis++) {

mediaMobileServoGimbalADCSum[axis] -= mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX];

mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX] = servo[axis];

mediaMobileServoGimbalADCSum[axis] += mediaMobileServoGimbalADC[axis][mediaMobileServoGimbalIDX];

#if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6))

atomicServo[axis] = (mediaMobileServoGimbalADCSum[axis] / MMSERVOGIMBALVECTORLENGHT - 1000)>>2;

#else

atomicServo[axis] = (mediaMobileServoGimbalADCSum[axis] / MMSERVOGIMBALVECTORLENGHT - 1000)<<4;

#endif

}

#else

for(uint8\_t i = (SEC\_SERVO\_FROM-1); i < SEC\_SERVO\_TO; i++){

#if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6)) || (defined(MEGA) && defined(MEGA\_HW\_PWM\_SERVOS))

atomicServo[i] = (servo[i]-1000)>>2;

#else

atomicServo[i] = (servo[i]-1000)<<4;

#endif

}

#endif

#endif

// write HW PWM servos for the mega

#if defined(MEGA) && defined(MEGA\_HW\_PWM\_SERVOS)

#if (PRI\_SERVO\_FROM == 1 || SEC\_SERVO\_FROM == 1)

OCR5C = servo[0];

#endif

#if (PRI\_SERVO\_FROM <= 2 && PRI\_SERVO\_TO >= 2) || (SEC\_SERVO\_FROM <= 2 && SEC\_SERVO\_TO >= 2)

OCR5B = servo[1];

#endif

#if (PRI\_SERVO\_FROM <= 3 && PRI\_SERVO\_TO >= 3) || (SEC\_SERVO\_FROM <= 3 && SEC\_SERVO\_TO >= 3)

OCR5A = servo[2];

#endif

#if (PRI\_SERVO\_FROM <= 4 && PRI\_SERVO\_TO >= 4) || (SEC\_SERVO\_FROM <= 4 && SEC\_SERVO\_TO >= 4)

OCR1A = servo[3];

#endif

#if (PRI\_SERVO\_FROM <= 5 && PRI\_SERVO\_TO >= 5) || (SEC\_SERVO\_FROM <= 5 && SEC\_SERVO\_TO >= 5)

OCR1B = servo[4];

#endif

#if (PRI\_SERVO\_FROM <= 6 && PRI\_SERVO\_TO >= 6) || (SEC\_SERVO\_FROM <= 6 && SEC\_SERVO\_TO >= 6)

OCR4A = servo[5];

#endif

#if (PRI\_SERVO\_FROM <= 7 && PRI\_SERVO\_TO >= 7) || (SEC\_SERVO\_FROM <= 7 && SEC\_SERVO\_TO >= 7)

OCR4B = servo[6];

#endif

#if (PRI\_SERVO\_FROM <= 8 && PRI\_SERVO\_TO >= 8) || (SEC\_SERVO\_FROM <= 8 && SEC\_SERVO\_TO >= 8)

OCR4C = servo[7];

#endif

#endif

// write HW PWM servos for the promicro

#if defined(PROMICRO) && defined(A32U4\_4\_HW\_PWM\_SERVOS)

#if (PRI\_SERVO\_FROM <= 7 && PRI\_SERVO\_TO >= 7)

OCR1A = servo[6];// Pin 9

#endif

#if (PRI\_SERVO\_FROM <= 5 && PRI\_SERVO\_TO >= 5)

OCR1B = servo[4];// Pin 10

#endif

#if (PRI\_SERVO\_FROM <= 6 && PRI\_SERVO\_TO >= 6)

OCR3A = servo[5];// Pin 5

#endif

#if (PRI\_SERVO\_FROM <= 4 && PRI\_SERVO\_TO >= 4)

OCR1C = servo[3];// Pin 11

#endif

#endif

#endif

}

#if defined(SERVO\_1\_HIGH) && !defined(A32U4\_4\_HW\_PWM\_SERVOS)

#define SERVO\_PULSE(PIN\_HIGH,ACT\_STATE,SERVO\_NUM,LAST\_PIN\_LOW) \

}else if(state == ACT\_STATE){ \

LAST\_PIN\_LOW; \

PIN\_HIGH; \

SERVO\_CHANNEL+=SERVO\_1K\_US; \

state++; \

}else if(state == ACT\_STATE+1){ \

SERVO\_CHANNEL+=atomicServo[SERVO\_NUM]; \

state++; \

ISR(SERVO\_ISR) {

static uint8\_t state = 0; // indicates the current state of the chain

if(state == 0){

SERVO\_1\_HIGH; // set servo 1's pin high

SERVO\_CHANNEL+=SERVO\_1K\_US; // wait 1000us

state++; // count up the state

}else if(state==1){

SERVO\_CHANNEL+=atomicServo[SERVO\_1\_ARR\_POS]; // load the servo's value (0-1000us)

state++; // count up the state

#if defined(SERVO\_2\_HIGH)

SERVO\_PULSE(SERVO\_2\_HIGH,2,SERVO\_2\_ARR\_POS,SERVO\_1\_LOW); // the same here

#endif

#if defined(SERVO\_3\_HIGH)

SERVO\_PULSE(SERVO\_3\_HIGH,4,SERVO\_3\_ARR\_POS,SERVO\_2\_LOW);

#endif

#if defined(SERVO\_4\_HIGH)

SERVO\_PULSE(SERVO\_4\_HIGH,6,SERVO\_4\_ARR\_POS,SERVO\_3\_LOW);

#endif

#if defined(SERVO\_5\_HIGH)

SERVO\_PULSE(SERVO\_5\_HIGH,8,SERVO\_5\_ARR\_POS,SERVO\_4\_LOW);

#endif

#if defined(SERVO\_6\_HIGH)

SERVO\_PULSE(SERVO\_6\_HIGH,10,SERVO\_6\_ARR\_POS,SERVO\_5\_LOW);

#endif

#if defined(SERVO\_7\_HIGH)

SERVO\_PULSE(SERVO\_7\_HIGH,12,SERVO\_7\_ARR\_POS,SERVO\_6\_LOW);

#endif

#if defined(SERVO\_8\_HIGH)

SERVO\_PULSE(SERVO\_8\_HIGH,14,SERVO\_8\_ARR\_POS,SERVO\_7\_LOW);

#endif

}else{

LAST\_LOW;

#if defined(SERVO\_RFR\_300HZ)

#if defined(SERVO\_3\_HIGH) // if there are 3 or more servos we dont need to slow it down

SERVO\_CHANNEL+=(SERVO\_1K\_US>>3); // 0 would be better but it causes bad jitter

state=0;

#else

SERVO\_CHANNEL+=SERVO\_1K\_US;

if(state<4){

state+=2;

}else{

state=0;

}

#endif

#endif

#if defined(SERVO\_RFR\_160HZ)

#if defined(SERVO\_4\_HIGH)

SERVO\_CHANNEL+=(SERVO\_1K\_US>>3);

state=0;

#else // if there are less then 4 servos we need to slow it to not go over ~170Hz (the highest working refresh rate for analog servos)

SERVO\_CHANNEL+=SERVO\_1K\_US;

if(state<8){

state+=2;

}else{

state=0;

}

#endif

#endif

#if defined(SERVO\_RFR\_50HZ) // to have ~ 50Hz for all servos

SERVO\_CHANNEL+=SERVO\_1K\_US;

if(state<30){

state+=2;

}else{

state=0;

}

#endif

}

}

#endif

#endif

#if defined(SERVO)

void initializeServo() {

#if !defined(HW\_PWM\_SERVOS)

// do pins init

#if (PRI\_SERVO\_FROM == 1) || (SEC\_SERVO\_FROM == 1)

SERVO\_1\_PINMODE;

#endif

#if (PRI\_SERVO\_FROM <= 2 && PRI\_SERVO\_TO >= 2) || (SEC\_SERVO\_FROM <= 2 && SEC\_SERVO\_TO >= 2)

SERVO\_2\_PINMODE;

#endif

#if (PRI\_SERVO\_FROM <= 3 && PRI\_SERVO\_TO >= 3) || (SEC\_SERVO\_FROM <= 3 && SEC\_SERVO\_TO >= 3)

SERVO\_3\_PINMODE;

#endif

#if (PRI\_SERVO\_FROM <= 4 && PRI\_SERVO\_TO >= 4) || (SEC\_SERVO\_FROM <= 4 && SEC\_SERVO\_TO >= 4)

SERVO\_4\_PINMODE;

#endif

#if (PRI\_SERVO\_FROM <= 5 && PRI\_SERVO\_TO >= 5) || (SEC\_SERVO\_FROM <= 5 && SEC\_SERVO\_TO >= 5)

SERVO\_5\_PINMODE;

#endif

#if (PRI\_SERVO\_FROM <= 6 && PRI\_SERVO\_TO >= 6) || (SEC\_SERVO\_FROM <= 6 && SEC\_SERVO\_TO >= 6)

SERVO\_6\_PINMODE;

#endif

#if (PRI\_SERVO\_FROM <= 7 && PRI\_SERVO\_TO >= 7) || (SEC\_SERVO\_FROM <= 7 && SEC\_SERVO\_TO >= 7)

SERVO\_7\_PINMODE;

#endif

#if (PRI\_SERVO\_FROM <= 8 && PRI\_SERVO\_TO >= 8) || (SEC\_SERVO\_FROM <= 8 && SEC\_SERVO\_TO >= 8)

SERVO\_8\_PINMODE;

#endif

#endif

#if defined(SERVO\_1\_HIGH)

#if defined(PROMINI) || (defined(PROMICRO) && defined(HWPWM6)) // uses timer 0 Comperator A (8 bit)

TCCR0A = 0; // normal counting mode

TIMSK0 |= (1<<OCIE0A); // Enable CTC interrupt

#define SERVO\_ISR TIMER0\_COMPA\_vect

#define SERVO\_CHANNEL OCR0A

#define SERVO\_1K\_US 250

#endif

#if (defined(PROMICRO) && !defined(HWPWM6)) // uses timer 3 Comperator A (11 bit)

TCCR3A &= ~(1<<WGM30) & ~(1<<WGM31); //normal counting & no prescaler

TCCR3B &= ~(1<<WGM32) & ~(1<<CS31) & ~(1<<CS32) & ~(1<<WGM33);

TCCR3B |= (1<<CS30);

TIMSK3 |= (1<<OCIE3A); // Enable CTC interrupt

#define SERVO\_ISR TIMER3\_COMPA\_vect

#define SERVO\_CHANNEL OCR3A

#define SERVO\_1K\_US 16000

#endif

#if defined(MEGA) // uses timer 5 Comperator A (11 bit)

TCCR5A &= ~(1<<WGM50) & ~(1<<WGM51); //normal counting & no prescaler

TCCR5B &= ~(1<<WGM52) & ~(1<<CS51) & ~(1<<CS52) & ~(1<<WGM53);

TCCR5B |= (1<<CS50);

TIMSK5 |= (1<<OCIE5A); // Enable CTC interrupt

#define SERVO\_ISR TIMER5\_COMPA\_vect

#define SERVO\_CHANNEL OCR5A

#define SERVO\_1K\_US 16000

#endif

#endif

#if defined(MEGA) && defined(MEGA\_HW\_PWM\_SERVOS)

#if defined(SERVO\_RFR\_RATE)

#if (SERVO\_RFR\_RATE < 20)

#define SERVO\_RFR\_RATE 20

#endif

#if (SERVO\_RFR\_RATE > 400)

#define SERVO\_RFR\_RATE 400

#endif

#else

#if defined(SERVO\_RFR\_50HZ)

#define SERVO\_RFR\_RATE 50

#elif defined(SERVO\_RFR\_160HZ)

#define SERVO\_RFR\_RATE 160

#elif defined(SERVO\_RFR\_300HZ)

#define SERVO\_RFR\_RATE 300

#endif

#endif

#define SERVO\_TOP\_VAL (uint16\_t)(1000000L / SERVO\_RFR\_RATE)

// init Timer 5, 1 and 4 of the mega for hw PWM

TIMSK5 &= ~(1<<OCIE5A); // Disable software PWM

#if (PRI\_SERVO\_TO >= 1) || (SEC\_SERVO\_TO >= 1)

TCCR5A |= (1<<WGM51); // phase correct mode & prescaler to 8 = 1us resolution

TCCR5A &= ~(1<<WGM50);

TCCR5B &= ~(1<<WGM52) & ~(1<<CS50) & ~(1<<CS52);

TCCR5B |= (1<<WGM53) | (1<<CS51);

ICR5 = SERVO\_TOP\_VAL;

#if (PRI\_SERVO\_FROM == 1 || SEC\_SERVO\_FROM == 1)

pinMode(44,OUTPUT);

TCCR5A |= (1<<COM5C1); // pin 44

#endif

#if (PRI\_SERVO\_FROM <= 2 && PRI\_SERVO\_TO >= 2) || (SEC\_SERVO\_FROM <= 2 && SEC\_SERVO\_TO >= 2)

pinMode(45,OUTPUT);

TCCR5A |= (1<<COM5B1); // pin 45

#endif

#if (PRI\_SERVO\_FROM <= 3 && PRI\_SERVO\_TO >= 3) || (SEC\_SERVO\_FROM <= 3 && SEC\_SERVO\_TO >= 3)

pinMode(46,OUTPUT);

TCCR5A |= (1<<COM5A1); // pin 46

#endif

#endif

#if (PRI\_SERVO\_TO >= 4) || (SEC\_SERVO\_TO >= 4)

TCCR1A |= (1<<WGM11); // phase correct mode & prescaler to 8

TCCR1A &= ~(1<<WGM10);

TCCR1B &= ~(1<<WGM12) & ~(1<<CS10) & ~(1<<CS12);

TCCR1B |= (1<<WGM13) | (1<<CS11);

ICR1 = SERVO\_TOP\_VAL;

#if (PRI\_SERVO\_FROM <= 4 && PRI\_SERVO\_TO >= 4) || (SEC\_SERVO\_FROM <= 4 && SEC\_SERVO\_TO >= 4)

pinMode(11, OUTPUT);

TCCR1A |= (1<<COM1A1); // pin 11

#endif

#if (PRI\_SERVO\_FROM <= 5 && PRI\_SERVO\_TO >= 5) || (SEC\_SERVO\_FROM <= 5 && SEC\_SERVO\_TO >= 5)

pinMode(12,OUTPUT);

TCCR1A |= (1<<COM1B1); // pin 12

#endif

#endif

#if (PRI\_SERVO\_TO >= 6) || (SEC\_SERVO\_TO >= 6)

// init 16bit timer 4

TCCR4A |= (1<<WGM41); // phase correct mode

TCCR4A &= ~(1<<WGM40);

TCCR4B &= ~(1<<WGM42) & ~(1<<CS40) & ~(1<<CS42);

TCCR4B |= (1<<WGM43) | (1<<CS41);

ICR4 = SERVO\_TOP\_VAL;

#if (PRI\_SERVO\_FROM <= 6 && PRI\_SERVO\_TO >= 6) || (SEC\_SERVO\_FROM <= 6 && SEC\_SERVO\_TO >= 6)

pinMode(6,OUTPUT);

TCCR4A |= \_BV(COM4A1); // connect pin 6 to timer 4 channel A

#endif

#if (PRI\_SERVO\_FROM <= 7 && PRI\_SERVO\_TO >= 7) || (SEC\_SERVO\_FROM <= 7 && SEC\_SERVO\_TO >= 7)

pinMode(7,OUTPUT);

TCCR4A |= \_BV(COM4B1); // connect pin 7 to timer 4 channel B

#endif

#if (PRI\_SERVO\_FROM <= 8 && PRI\_SERVO\_TO >= 8) || (SEC\_SERVO\_FROM <= 8 && SEC\_SERVO\_TO >= 8)

#if defined(AIRPLANE) || defined(HELICOPTER)

servo[7] = MINCOMMAND; // Trhottle at minimum for airplane and heli

OCR4C = MINCOMMAND;

#endif

pinMode(8,OUTPUT);

TCCR4A |= \_BV(COM4C1); // connect pin 8 to timer 4 channel C

#endif

#endif

#endif // mega hw pwm

#if defined(PROMICRO) && defined(A32U4\_4\_HW\_PWM\_SERVOS)

// atm. always initialize 4 servos to pins 9, 10, 11, 5

TIMSK1 &= ~(1<<OCIE1A) & ~(1<<OCIE1B) & ~(1<<OCIE1C);

TCCR1A |= (1<<WGM11); // phase correct mode & prescaler to 8

TCCR1A &= ~(1<<WGM10);

TCCR1B &= ~(1<<WGM12) & ~(1<<CS10) & ~(1<<CS12);

TCCR1B |= (1<<WGM13) | (1<<CS11);

pinMode(9,OUTPUT);

TCCR1A |= (1<<COM1A1); // pin 9

pinMode(10,OUTPUT);

TCCR1A |= (1<<COM1B1); // pin 10

pinMode(11,OUTPUT);

TCCR1A |= (1<<COM1C1); // pin 11

TCCR3A |= (1<<WGM31); // phase correct mode & prescaler to 8

TCCR3A &= ~(1<<WGM30);

TCCR3B &= ~(1<<WGM32) & ~(1<<CS30) & ~(1<<CS32);

TCCR3B |= (1<<WGM33) | (1<<CS31);

pinMode(5,OUTPUT);

TCCR3A |= (1<<COM3A1); // pin 5

#if defined(SERVO\_RFR\_RATE)

#if (SERVO\_RFR\_RATE < 50) || (SERVO\_RFR\_RATE > 400)

#error "\* invalid SERVO\_RFR\_RATE specified"

#endif

#define SERVO\_TOP\_VAL (uint16\_t)(1000000L / SERVO\_RFR\_RATE)

#elif defined(SERVO\_RFR\_50HZ)

#define SERVO\_TOP\_VAL 16700

#elif defined(SERVO\_RFR\_160HZ)

#define SERVO\_TOP\_VAL 6200

#elif defined(SERVO\_RFR\_300HZ)

#define SERVO\_TOP\_VAL 3300

#else

#error "\* must set SERVO\_RFR\_RATE or one of the fixed refresh rates of 50, 160 or 300 Hz"

#endif

#if defined(SERVO\_PIN5\_RFR\_RATE)

#if (SERVO\_PIN5\_RFR\_RATE < 50) || (SERVO\_PIN5\_RFR\_RATE > 400)

#error "\* invalid SERVO\_PIN5\_RFR\_RATE specified"

#endif

#define SERVO\_PIN5\_TOP\_VAL (uint16\_t)(1000000L / SERVO\_PIN5\_RFR\_RATE)

#else

#define SERVO\_PIN5\_TOP\_VAL SERVO\_TOP\_VAL

#endif

ICR1 = SERVO\_TOP\_VAL; // set TOP timer 1

ICR3 = SERVO\_PIN5\_TOP\_VAL; // set TOP timer 3

#endif // promicro hw pwm

}

#if (LOG\_VALUES >= 3) || defined(POWERMETER\_SOFT)

{

static uint32\_t lastRead = currentTime;

uint16\_t amp;

uint32\_t ampsum, ampus; // pseudo ampere \* microseconds

/\* true cubic function;

\* when divided by vbat\_max=126 (12.6V) for 3 cell battery this gives maximum value of ~ 500

\* when divided by no\_vbat=60 (6V) for 3 cell battery this gives maximum value of ~ 1000

\* \*/

static uint16\_t amperes[64] = { 0, 2, 6, 15, 30, 52, 82,123,

175,240,320,415,528,659,811,984,

1181,1402,1648,1923,2226,2559,2924,3322,

3755,4224,4730,5276,5861,6489,7160,7875,

8637 ,9446 ,10304,11213,12173,13187,14256,15381,

16564,17805,19108,20472,21900,23392,24951,26578,

28274,30041,31879,33792,35779,37843,39984,42205,

44507,46890,49358,51910,54549,57276,60093,63000};

if (analog.vbat > NO\_VBAT) { // by all means - must avoid division by zero

ampsum = 0;

for (i =0;i<NUMBER\_MOTOR;i++) {

amp = amperes[ ((motor[i] - 1000)>>4) ] / analog.vbat; // range mapped from [1000:2000] => [0:1000]; then break that up into 64 ranges; lookup amp

ampus = ( (currentTime-lastRead) \* (uint32\_t)amp \* (uint32\_t)conf.pint2ma ) / PLEVELDIVSOFT;

#if (LOG\_VALUES >= 3)

pMeter[i]+= ampus; // sum up over time the mapped ESC input

#endif

#if defined(POWERMETER\_SOFT)

ampsum += ampus; // total sum over all motors

#endif

}

#if defined(POWERMETER\_SOFT)

pMeter[PMOTOR\_SUM]+= ampsum / NUMBER\_MOTOR; // total sum over all motors

#endif

}

lastRead = currentTime;

}

#endif

}

HEX CODE

A picture containing text

Description automatically generated

A picture containing table

Description automatically generated

A picture containing text

Description automatically generated

Table

Description automatically generated with medium confidence

Table

Description automatically generated

Text

Description automatically generated with low confidence

Table

Description automatically generated

A picture containing table

Description automatically generated

Text

Description automatically generated with medium confidence

Table

Description automatically generated

**REFLECTION**

**Gimoros**

**I am really happy because we have learned a embedded devices and microprocessors. I also enjoyed making programs using flowcode. It really helps us to become more efficient in project planning and implementation. We are also very thankful because our professor is a master of his class and very hands on to make sure we learned a lot. I am very sure that this new skill that I learned will be used when I will be the one making it in the industry.**