CSE 316
Microprocessors and Microcontrollers Sessional
Simplified Automated Quad-copter

Supervisors
Md. Abdus Sattar (Associate Professor, Dept. of CSE, BUET)
Muhammad Ali Nayeem (Lecturer, Dept. of CSE, BUET)
Abdus Salam Azad (Lecturer, Dept. of CSE, BUET)

Group Members
Shadman Saqib Eusuf (1205003)
Kazi Ashik Islam (1205007)

October 5, 2016

Abstract

In this report we have described the details about implementing our term project “Simplified Automated Quad-copter”. Automating 3D motion is a tough and rare task in our country. How we tried to attempt it, what difficulties we faced and how much we could ultimately implement are discussed in depth in this report.

Contents

List of Figures

1 Problem Specification

We needed to automate the motion of a quad-copter (3D motion). So, we had to set a predefined path of flight for it. To be more specific, our goal was to drive the quad-copter (flight control board of quad, to be more precise) using ATmega so that it could fly following a fixed route. Normally transmitter-receiver controlled quad-copters are quite common. Our aim was to automate this, so that the control would no longer be manual.

2 Equipments

- ATmega 32A – 3 pieces
- Lipo Battery
- Working Quad-copter (Including frame, motors, esc, flight control board etc.)
- Wires and Electrical Tape
- Breadboard 2 pieces (1 small & 1 large)
- USBasp AVR Programmer

3 Solution Overview

First of all, to automate the motion of quad we needed to know how it actually flies when controlled manually with transmitter-receiver. We found out that according to the input provided in the transmitter, the receiver generates pulses of certain widths. Different inputs in the transmitter correspond to pulses of different widths in the receiver. The receiver is generally connected to the flight control board of quad. Now our main idea was to replace this receiver with ATmega microcontroller.

In this regard, to control a quad copter, transmitter-receiver with at least 5 channels (namely Aileron, Elevetor, Throttle, Rudder and Auxiliary) is needed. Through all the channels the flight control board gets some pulses for moving the quad in different directions. So, we had to generate 5 pulses with our ATmega microcontrollers. In addition, the pulses would have to be synchronized (i.e. no phase difference among them) for giving the flight control board an illusion of a receiver as it is constantly checked; and thereby controlling it. If we generated 3-4 pulses with ATmega32A, we had

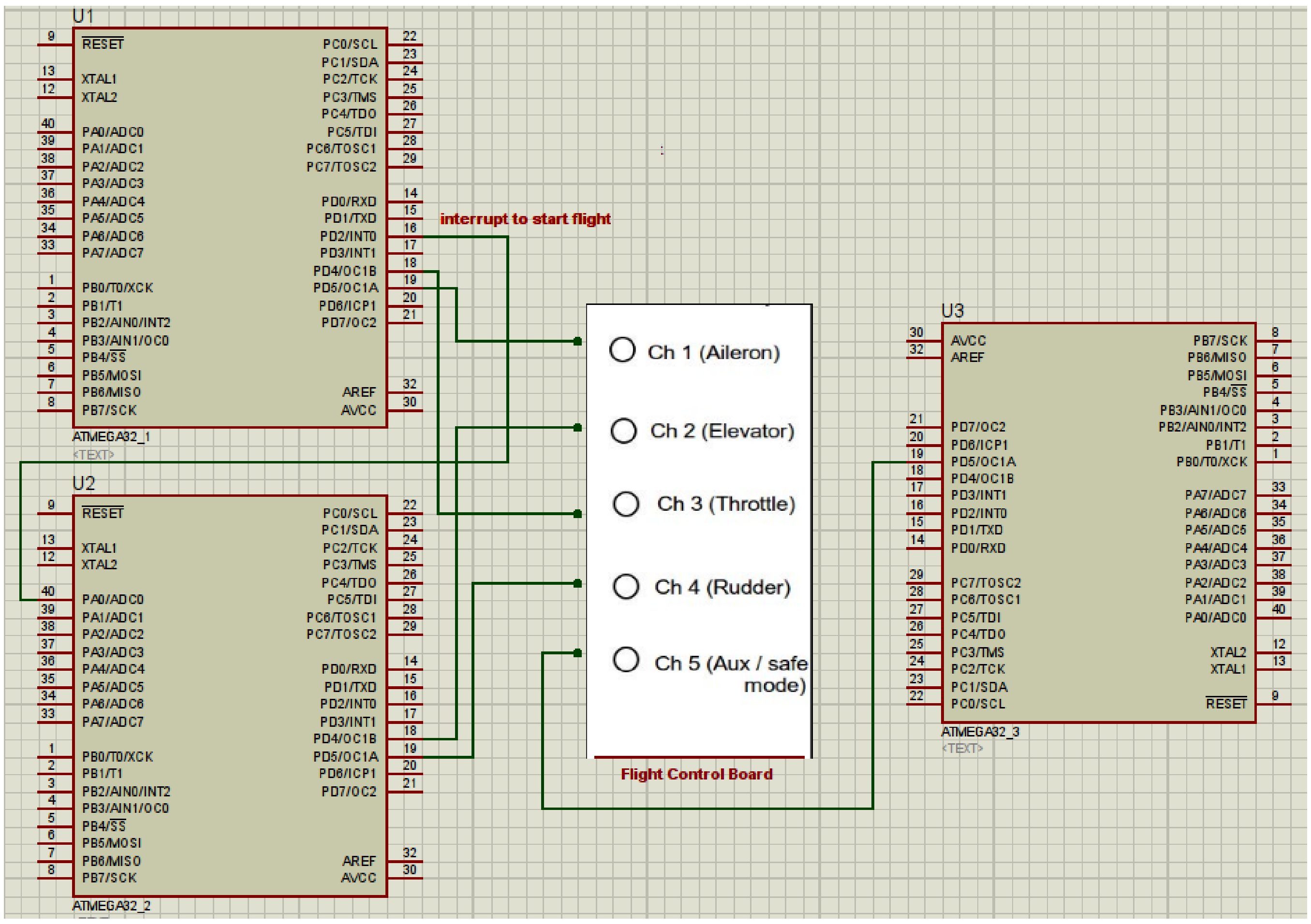

Figure 1: Circuit Diagram

been getting pulses with phase differences. So, we ultimately used 3 ATmega32A(s) for this synchronization and generated at most 2 pulses from each.

In the coding part, we generated 5 PWM signals using 3 ATmega32A(s) and controlled their respective duty cycles according to the different states of flight along our predefined path. Thus the speed of the bldc motors of quad got altered and direction of its motion could be controlled. Through this duty cycle modification we could also manage to achieve somewhat safe landing by means of trial and error.

4 Source Code

We had to use 3 atmega32 micro-controllers with different source codes. The source codes along with the circuit diagram are provided here.

4.1 Source atmega1.c

```c
/*
 * atmega1.c
 */

//period: 20000 micro -> ICR1
//timer 1 (A) -> PD5 (PIN 19 of ATMEGA32A) -> channel 1 (aileron
    )
//timer 1 (B) -> PD4 (PIN 18 of ATMEGA32A) -> channel 3 (
    throttle)

#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>

#define before_throttle 0
#define throttle_up 1
#define throttle_up_complete 2
#define throttle_down 3
#define throttle_down_complete 4
#define no_throttle 5

#define delay_throttle_up 22
#define delay_throttle_up_complete 29
#define delay_throttle_landing 64

#define aileron_width OCR1A
#define throttle_width OCR1B

uint8_t timer0_overflows, timer2_overflows;
volatile uint8_t cur_state, timer2_init_flag;

void TIMER1_INIT()
{
    ///clear OC1X on compare match (1010)
    ///no force on output compare (00)
    ///fast pwm WGM13:10 -> (1110)
    ///internal clock, no prescaler (001)

    TCCR1A = 0b10100010;
    TCCR1B = 0b00011001;

    ICR1 = 19999; ///period = ICR1+1 (as no prescaler)
    aileron_width = 1467; ///width = OCR1A + 1 (^^)
    throttle_width = 1138; /// (^^)
}

void TIMER2_INIT()
```

```c
{
    timer2_overflows = 0;
    TIMSK |= 0b01000000;     ///enable timer2 overflow interrupt
    TCCR2 = 0b00000110; ///internal clock, prescaler (/256)
    TCNT2 = 0;
}

ISR(TIMER2_OVF_vect)
{
    switch(cur_state)
    {
    case throttle_up:
        switch(timer2_overflows)
        {
        case delay_throttle_up: ///increment throttle for 23 overflows
            throttle_width = 1856;   /// 1863
            TIMSK &= 0b10111111;     /// disable timer2 overflow interrupt
            cur_state = throttle_up_complete;
            timer2_init_flag = 1;
            break;
        default:
            throttle_width += 31;
            timer2_overflows++;
        }
        break;

    case throttle_up_complete:
        switch(timer2_overflows)
        {
        case delay_throttle_up_complete: ///remain full throttle for 30 overflows
            TIMSK &= 0b10111111;     /// disable timer2 overflow interrupt
            cur_state = throttle_down;
            throttle_width = 1800;    /// 1814
            timer2_init_flag = 1;
            break;

        default:
            timer2_overflows++;
        }

        break;

    case throttle_down:
        switch(timer2_overflows)
        {
```

```c
        case delay_throttle_landing: ///remain in down throttle for 40 overflows
            TIMSK &= 0b10111111;     /// disable timer2 overflow interrupt
            cur_state = no_throttle;
            throttle_width = 1138;   ///bring throttle to 0
            break;

        default:
            timer2_overflows++;
        }

        break;

    }
}

ISR(INT0_vect)
{
    cur_state = throttle_up;
    aileron_width = 1452; //adjust aileron for flight

    TIMER2_INIT(); //gradually increment throttle
}


int main()
{
    DDRD = 0b00110000;  /// PD4-> throttle, PD5-> aileron for output
    timer2_init_flag = 0;

    GICR |= (1<<INT0);  ///enable INT0
    MCUCR |= (1<<ISC00) | (1<<ISC01); ///configure for rising edge

    TIMER1_INIT();
    sei();

    while(1){
        if((cur_state == throttle_up_complete || cur_state == throttle_down) && timer2_init_flag){
            timer2_init_flag = 0;
            TIMER2_INIT();
        }
    }
    return 0;
}
```

4.2 Source atmega2.c

```c
/*
 * atmega2.c
 */

//period: 20000 micro -> ICR1
//timer 1 (A) -> PD5 (PIN 19 of ATMEGA32A) -> channel 4 (Rudder)
//timer 1 (B) -> PD4 (PIN 18 of ATMEGA32A) -> channel 2 (Elevator)

#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>

#define rudder_width OCR1A
#define elevator_width OCR1B

#define before_arm_state 0      // initial state
#define arm_state 1             // arming state
#define before_flight_state 2   // arm complete, wait for some time before flight
#define flight_state 3          // follow predefined route

#define delay_before_arm 200    // initial state time, approx 5s, increase it so that almost 12s delay is achieved
#define delay_armed 10          // arming time, approx 2.5s
#define delay_before_flight 29  // waiting time before flight approx 8s

uint16_t timer0_overflows, timer2_overflows;
volatile uint8_t cur_state, timer2_init_flag, flight_state_interrupt_flag;

void TIMER1_INIT()
{
    ///clear OC1X on compare match (1010)
    ///no force on output compare (00)
    ///fast pwm WGM13:10 -> (1110)
    ///internal clock, no prescaler (001)

    TCCR1A = 0b10100010;
    TCCR1B = 0b00011001;

    ICR1 = 19999; ///period = ICR1+1 (as no prescaler)
    rudder_width = 1491; ///width = OCR1A + 1 (^^) 1479
    elevator_width = 1479; /// (^^)
}

```

```c
void TIMER2_INIT()
{
    timer2_overflows = 0;
    TIMSK |= 0b01000000; ///enable timer2 overflow interrupt
    TCCR2 = 0b00000111; ///internal clock,  prescaler (/1024)
    TCNT2 = 0;
}


ISR(TIMER2_OVF_vect)
{

    switch(cur_state)
    {

    case before_arm_state:

        switch(timer2_overflows)
        {
        case delay_before_arm:
            rudder_width = 1131;     /// required for arming
            TIMSK &= 0b10111111;     /// disable timer2 overflow interrupt
            cur_state = arm_state;
            timer2_init_flag = 1;    /// in main timer2 may now be enabled again for next state
            break;

        default:
            timer2_overflows++;
        }
        break;

    case arm_state:

        switch(timer2_overflows)
        {
        case delay_armed:
            rudder_width = 1491;     /// neutral for waiting mode before flight 1479
            TIMSK &= 0b10111111;     /// disable timer2 overflow interrupt
            cur_state = before_flight_state;
            timer2_init_flag = 1;
            break;

        default:
            timer2_overflows++;
        }

        break;

    case before_flight_state:

        switch(timer2_overflows)
        {
        case delay_before_flight:

            TIMSK &= 0b10111111;    /// disable timer2 overflow interrupt
            PORTA |= 0b00000001;    ///send interrupt to atmega1
            cur_state = flight_state;
            elevator_width = 1522;     ///set elevator value

            PORTA &= 0b11111110;
            break;

        default:
            timer2_overflows++;
        }
        break;
    }
}


int main()
{
    DDRD = 0b00110000;  /// PD4-> elevator, PD5-> rudder for output
    DDRA = 0b00000001;  ///configure PA0 for output, used to send interrupt to atmega1 for initiating flight state

    cur_state = before_arm_state;
    timer2_init_flag = 1;

    TIMER1_INIT();
    sei();

    while(1){
        if(cur_state < flight_state && timer2_init_flag){
            timer2_init_flag = 0;
            TIMER2_INIT();
        }
    }
    return 0;
}
```

4.3 Source atmega3.c

```c
/*
 * atmega3.c
 */

//period: 20000 micro -> ICR1
//timer 1 (A) -> PD5 (PIN 19 of ATMEGA32A) -> channel 5 (
    auxiliary)

#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>

#define aux_width OCR1A

void TIMER1_INIT()
{
    ///clear OC1X on compare match (1010)
    ///no force on output compare (00)
    ///fast pwm WGM13:10 -> (1110)
    ///internal clock, no prescaler (001)

    TCCR1A = 0b10100010;
    TCCR1B = 0b00011001;

    ///period = ICR1+1 (as no prescaler)
    ICR1 = 19999;

    ///width = OCR1A + 1 (^^) 1972 -> on
    aux_width = 1972;
}

int main()
{
    /// PD5 -> auxiliary (safe mode)
    DDRD = 0b00100000;

    TIMER1_INIT();
    sei();

    while(1);
    return 0;
}
```

5 Challenges We Faced During the Project

Throughout the lifetime of the project we had to deal with several difficulties.

- The first problem that we faced was, we couldn't find a quad-copter with our needed specifications in the market. We needed a copter whose signal receiver is replaceable, whereas in market copters, receiver is an integral part of the body.

  To solve this issue we had to look for custom made copters made by our senior students in ME department. Two of our friends in ME department helped us greatly in this matter. Ultimately we were able to find a working copter to our specifications but we lost valuable time in the meantime.

- We didn't know much about the steering system of the copter before. So we had to practise regularly with radio transmitter control to observe what kind of signal is sent to the Flight Control Board by the receiver.

- Theoretically a certain kind of motion is supposed to be triggered by changing signal in a single channel. For example, increasing pulse width of throttle signal is supposed to elevate the copter only. But as mentioned above, the copter is custom made. So, just to elevate the copter, we had to send a combination of signals in 3 channels, namely Throttle, Aileron and Elevator.

  Fortunately, there is a feature (Receiver Test) in the flight control board to see the pulse width of a received signal. We used this feature to discover the appropriate combination and applied a trial and error approach. Ultimately we were able to stabilize the motion to a satisfactory extent.

- The flight control board expects very precise duty cycle. So we tried and failed to generate PWM signals with timer 0 and timer 2.

  However, all the signals in 5 channels had same time period. So we were able to generate 2 PWM signals with timer 1. Ultimately, we had to use 3 ATmega32A(s) microcontrollers generating Aileron and Throttle signal with the first one, Elevator and Rudder signal with the second one and Auxiliary signal with the third one.

- Safety was very important factor in this project, as a copter out of control can be very dangerous. So we had to be very careful. We checked the whole circuit and double checked our code before every flight trial.

- The ATmega(s) may not always drive the flight control board deterministically. Four of our Atmega(s) were showing this non-determinism, they sometimes worked and sometimes not. We had to replace them with three new ones.

6 Modifications

The main idea behind our simple predefined path was to move the quad along all 3 axes. Theoretically we had to alter the pulses of 3 different channels for this purpose, one for moving in each axis. However, in practical scenario if we were trying to lift the quad with certain PWM signal in 1 channel we saw its motion was not purely upwards, rather it was going front-right as well. So, we modified out simple path to a simpler one and just tried to implement lift and move upwards, stay at a certain height for some time and then land. For this purpose only, we had to control pulses in 3 channels simultaneously. At last we managed to achieve this with a small to medium margin of error.

For safe landing, the idea of using timer turned out to be good enough and so, the need of altimeter was no longer very crucial. For this reason and the fact of non-deterministic interfacing between ATmega and flight control board we did/could not use pressure sensor as altimeter as proposed previously.

7 Acknowledgement

We are very grateful and thankful to everyone who helped us in this project. Among them Md. Raihan Ali Khan (Student ID# 1210002, Dept. of ME, BUET) and Shadman Tahmid (Student ID# 1210019, Dept.of ME, BUET) are specially mentionable. They helped us a lot with mechanical works (repair, replacement and modification of quad parts) of our project and overall management. We would also like to thank Bibekananda Datta (ME '11) who gave us an almost ready quad but unfortunately we could not use it later for unavoidable circumstances. We are really grateful Kowshik Parvej(ME '11) who gave us his ready quad without which this project may have had to be abandoned. Finally we would like to thank our families, friends and everyone else (whom we are unable to name here) who helped us in this project.