VIA UNIVERSITY COLLEGE

2015

Project Report – SEPI4

189576 FELIX DANIEL ABAIDOO
189613 LUBOMIR KLUCKA
189614 PAWEL KRAMARZ

SUPERVISORS:
ERLAND KETIL LARSEN
IB HAVN
LARS BECH SØRENSEN

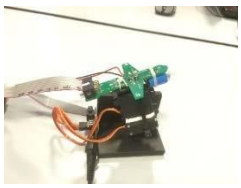
Contents

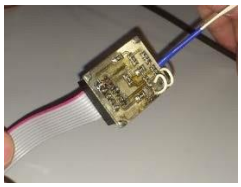
List of Figures

Abstract

The main purpose of this project is to design, implement and test a real-time system to control an aircraft simulator, which is based on the MicroC OS-II. The system runs on ATmega2560 microcontroller and STK600 board connected to LCD board and other given components. In order to develop this simulated airplane system application some important steps were considered. Starting with analysis of requirements, designing, implementing and testing it iteratively.

1. Introduction





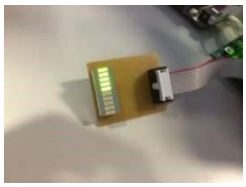


Figure 1 - Components

To make this aircraft simulator, the group was given a set of components. Because of the possibility to use those components for many different kinds of functions, analysis have to be made to determine the exact functionalities of the system. After that, design, implement and test the real-time application, make sure that it is working properly - that it is schedulable, protect resources with semaphores or mutexes and consider threats of doing this (f.e. deadlocks, priority inversion).

Devices used:

- AtMega2560 microcontroller
- Atmel STK600 development board
- LCD Display with 4 buttons
- 2 servo motors
- Aircraft PCB with accelerometer, temperature sensor and LEDs
- LED bar
- R-2R D/A Converter

2. Analysis

2.1 System requirements

Functional requirements

1. The pilot must be able to control the thrust of the plane with the encoder
2. The pilot must be able to turn and tilt the plane to desired direction
3. The pilot must be able to see engine temp, speed, height on LCD display
4. The pilot must be able to see the speed on the led bar
5. The pilot must be able to select different modes of LCD display by buttons on the LCD

Non-functional requirements

1. The system should be reliable

2. The system should be robust
3. The system should be maintainable
4. The system should be schedulable
5. The system should be able to synchronize the servo and the airplane
6. The system must signal warnings and malfunctions

2.2 Use-case diagram

In this UseCase diagram the user is called Pilot. The Use-case diagram below shows the functionalities of the system.

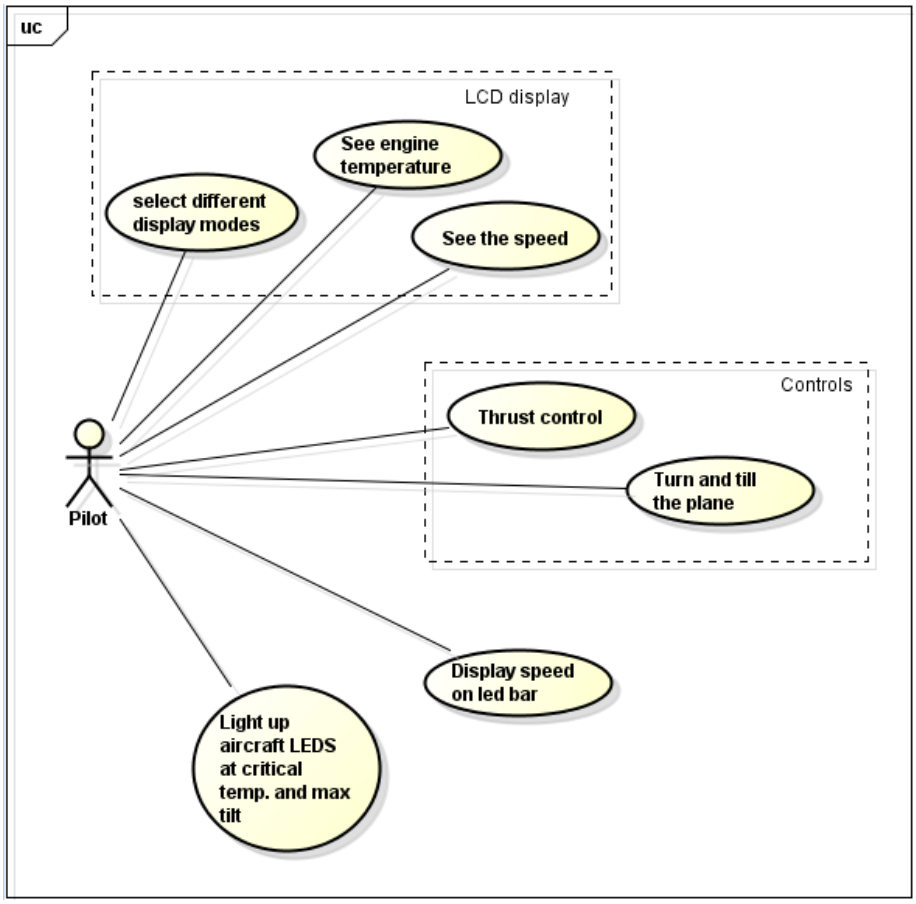


Figure 2 - Use-case diagram

2.3 Use-case description

Each of the use-case have a description – this can be seen in the example in Figure 3, describing the ‚Turn and tilt the plane‘functionality. Other use-case descriptions can be seen in Appendices.

Turn and till the plane / UseCase Description

ITEM	VALUE
UseCase	Turn and till the plane
Summary	Pilot can control servos in order to turn or/and tilt the plane.
Actor	Pilot
Precondition	
Postcondition	
Base Sequence	While button 1 on STK600 is pressed, the plane is tilting down. While button 2 on STK600 is pressed, the plane is tilting up. While button 3 on STK600 is pressed, the plane is turning left. While button 4 on STK600 is pressed, the plane is turning right.
Branch Sequence	
Exception Sequence	There are set boundaries for servos.
Sub UseCase	
Note	

Figure 3 - Use-case description

2.4 Constraints

LCD
The LCD-chip needs some time to process the incoming data, and the crystals also need time to react. From experience we know that the refreshing frequency for LCD displays should not exceed the 200Hz.

LIS331 – Accelerometer
From the datasheet we can read that the display is driven by SPI interface, which is clocked inside of the driver with the frequency of µC/128. In our case it is 63.5kHz. We assume we receive 20 bytes each time from the LIS331. The fastest update rate while reading from the accelerometer is 400Hz.

Servo motors
Servos are controlled by sending an electrical pulse of variable width, or pulse width modulation (PWM), through the control wire. There is a minimum pulse, a maximum pulse and a repetition rate. A servo motor can usually only turn 90° in either direction for a total of 180° movement.

Led bar
Led bar has only 10 LEDs, therefore the simulation of speed on it can be limited. Because of this, when the last LED would light up, this would be a maximum speed.

3. Design

3.1 Tasks

Requirements for the system were divided into 3 tasks. All tasks are periodic - timer controlled tasks are used, because it is easier to determine their exact period than with OSTimeDelay tasks and prove their schedulability.

It was also an option to use a sporadic task if, for example, some kind of emergency button were to be impleneted, but in the analysis part it was decided not to use this kind of function.

1. Task
   - Hard real-time task
   - Task is reading data from accelerometer and temperature sensor on the aircraft, encoder value from LCD display board and reading if any button was pressed both on STK600 or LCD display board
2. Task
   - Hard real-time task
   - Task is used for controlling the servo motors with buttons on STK600, lighting up the LEDs on aircraft in case of maximum tilt or high temperature, displaying speed on ledbar and switching between LCD display modes
3. Task
   - Soft real-time task
   - Task is used for displaying data on the LCD. It displays the x and y values from accelerometer, temperature and speed.

3.2 Task diagram

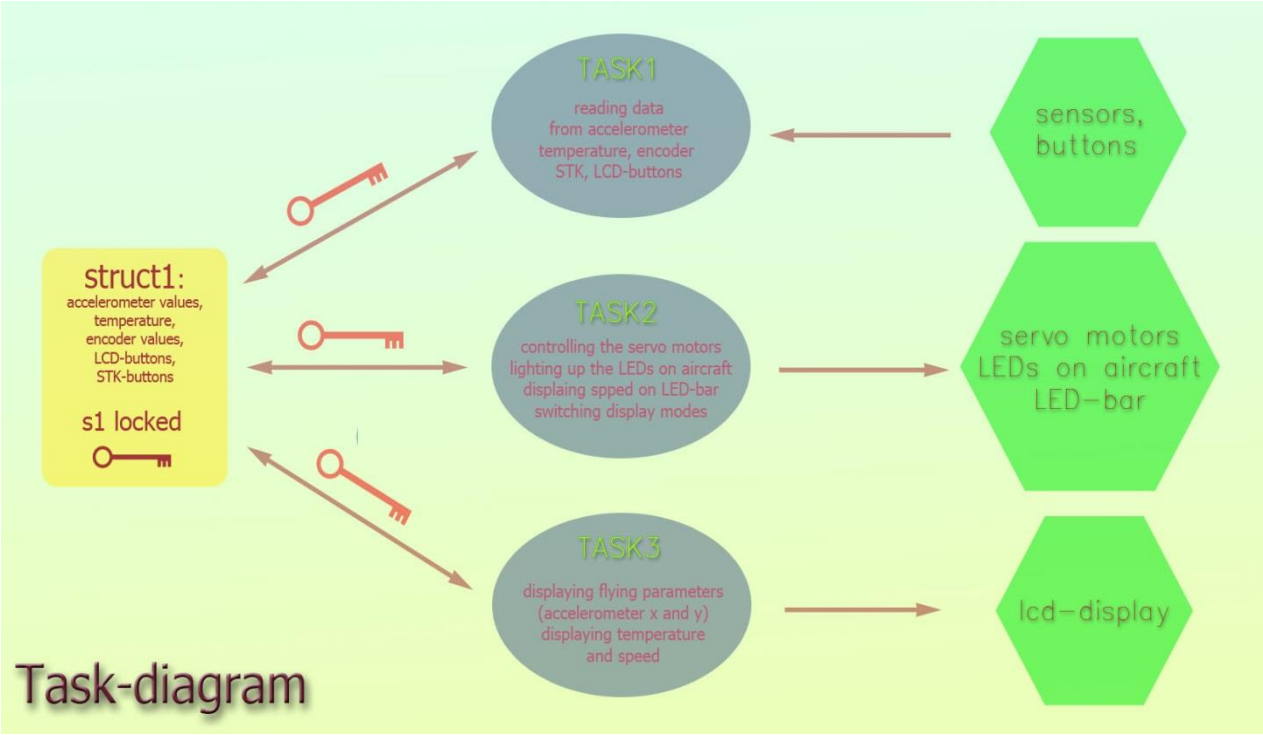


Figure 4 - Task diagram

Task diagram above shows how the system is supposed to work. All resource values are in one struct – *struct1*, protected by semaphore *s1*. Each task needs access to the resource and therefore has to acquire semaphore before it can get to the resource. Diagram also shows what hardware resources each task has access to.

3.3 Scheduling strategy and piano-roll

Although MicroC/OS-II supports also dynamic priorities, but due to lack of experience it was chosen to use fixed priorities scheduling – FPS. FPS is also widely used in practice.

Period and deadline is defined for each task together with guessed computation time, which is expected to be even lower in the real measurements. With these values, they can be seen in Figure 5, it can be calculated if the system is schedulable.

	period [ms]	exec. time [ms]	deadline [ms]
Task 1	300	50	150
Task 2	500	100	250
Task 3	1000	300	500

Figure 5 - Table of tasks

In the Cyclic executive approach the table represents the major cycle. By splitting the execution into smaller procedures (minor cycles) the exact execution time of each of them can be obtained. Finally a timeline has to be created. The advantage of this approach is, that all code shares the common address space, so the data does not be protected by semaphores.

Cyclic executive approach has also disadvantages. It is difficult to construct the code and to resize it or implement new functionality without to redesign. For our purpose the cyclic executive is not the best choice to use, because the system should be easily extensible.

Because system runs on FPS, deadline monotonic scheduling was used – deadline is smaller than period, as can be seen in Table 1. Priorities are set according to DMS strategy – smaller the deadline, higher the priority. This means that Task 1 has the highest priority and Task 3 the lowest. Rate monotonic scheduling with *deadline = period* was also an option but in general, it must be possible for a task to define a deadline smaller than it's period.

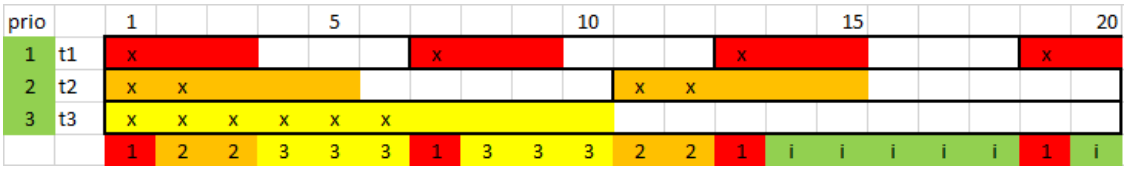


Figure 6 - piano roll

In the picture above can be seen 3 tasks. They are ordered by their priorities – task 1 has the highest priority = 1. The colors that fill out the cells show deadlines each task meet, or in case of task 3 which is soft task, should meet. Letter ‚x' inside the colored cells means expected execution time for each task. Period of the task can be seen at the point where each task starts again.

In the last line, it is shown the order in which the tasks run – number of task that is running together with its color. The ‚i' stands for idle – no task runs.

4. Implementation / Test

4.1 Oscilloscope measurements

The diagrams show the real utilization of the microcontroller each for every task.

The trigger was set in this case on the falling edge of the signal. The voltage level defines the task number as showed below:

- Task1 is defined by 2.15V
- Task2 is defined by 1.85V
- Task3 is defined by 1.55V

The Voltage is to be find in the right bottom corner. The delta sign (Δ) matches the execution time in case of measuring of execution time or it matches the time of period in case of measuring the period.

Results:

The Task 1 is ready after 350µs work, while Task2 needs 315µs. The longest time that the microcontroller uses to finish is the Task3 with 21.4ms. The reason for this long execution time of Task3 is with the high probability the wait command for the SPI buffer inside of the driver in the LCD.c file:

```
static uint8_t lcd_waitbusy(void) {
register uint8_t c;

while ((c=lcd_read(0)) & (1<<LCD_BUSY)) {
      ;       /* wait until busy flag is cleared */
}
delay(2);     /* the address counter is updated 4us after the busy flag is cleared */

/* now read the address counter */
return (lcd_read(0));  // return address counter
}
```

Looking on the time line diagrams we can find two other tasks working periodically. One is the timer task and the other is the start task (OS_TASK_START). The timer task works every 20ms and because it has the highest priority it preempts the other task. To achieve the best utilization one should consider the frequency of the timer task.

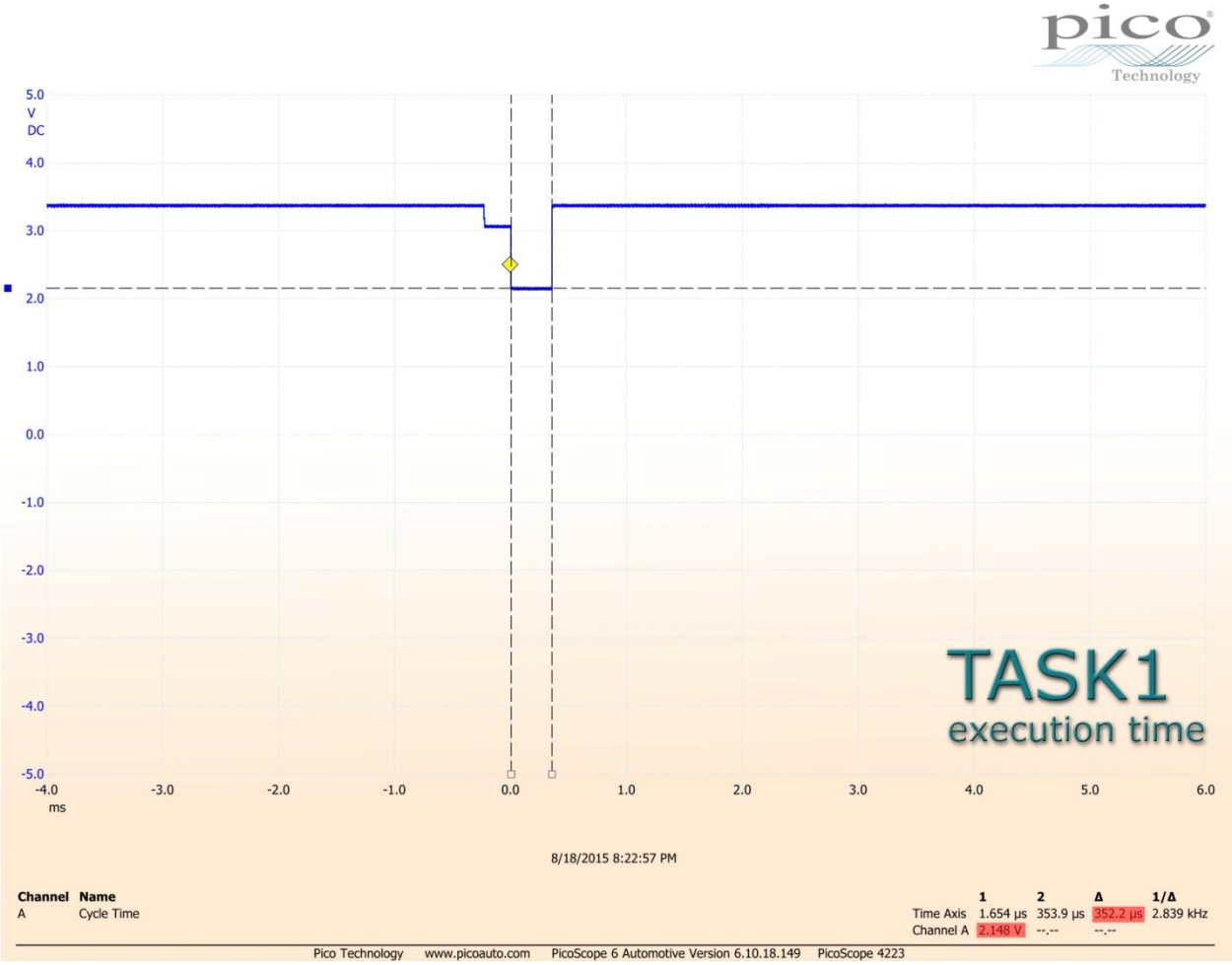


Figure 7 - Task 1 execution time

Please refer to the appendices for all diagrams.

4.2 Proof that system is schedulable

After measurement of real computation time, which was much smaller than expected computation time, it was decided to make periods and deadlines of tasks smaller to make the program run faster. Simple utilization test and worst execution time calculations for each task were made to make sure that the system will be schedulable and the hard-realtime tasks always meet their deadline in Figure 9 with formula in Figure 8.

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

Figure 8 - Formula for worst case execution time calculation

	period [ms]	exec. time [ms]	deadline	prio	utilization
T1	40	0,35	30	1	0,01
T2	60	0,31	50	2	0,01
T3	300	21,4	200	3	0,07
Utilization:	8,53 < 78%				
Response time:					
W1	0,35		0,35		
Worst case of task 1 is	0,35 [ms]				
W2.0	0,31		0,31		
W2.1	0,31 + [0,31/40]*0,35		0,66		
W2.2	0,31 + [0,66/40]*0,35		0,66		
Worst case of task 2 is	0,66 [ms]				
W3.0	21,4		21,4		
W3.1	21,4 + [21,4/40]*0,35 + [21,4/60]*0,31		22,06		
W3.2	21,4 + [22,06/40]*0,35 + [22,06/60]*0,31		22,06		
Worst case of task 3	22,06 [ms]				

Figure 9 - Utilization and RTA calculations

In top of Figure 9 is a table of tasks which defines their periods, computation times, deadlines priorities and utilization. Utilization is a simple test for FPS that needs to be done to prove that system is schedulable. Utilization is calculated by formula *computation time / period*. From theory, we know that in case of 3 tasks, their combined utilization must be less the 78%. Combined utilization in this case is 8,53% < 78%.

To be even more precise, worst case execution time calculations can be seen under utilization in Figure 6. It is calculated by adding together computation time plus maximum interference by higher priority tasks. Highest priority task cannot be interrupted by any other task, therefore it's worst case execution time is it's computation time.

Worst case execution times of all task are smaller than their deadlines, therefore they will always meet their deadlines.

4.3 Unit test

Before use of these devices some test are necessary to perform. Board switches, the aircraft and the encoder were tested with the Unit Test routine. Here are the results from the test:

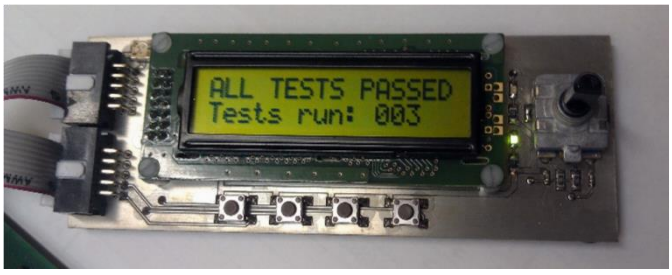


Figure 10 - LCD display test result

The picture above shows that the test is performed and successful

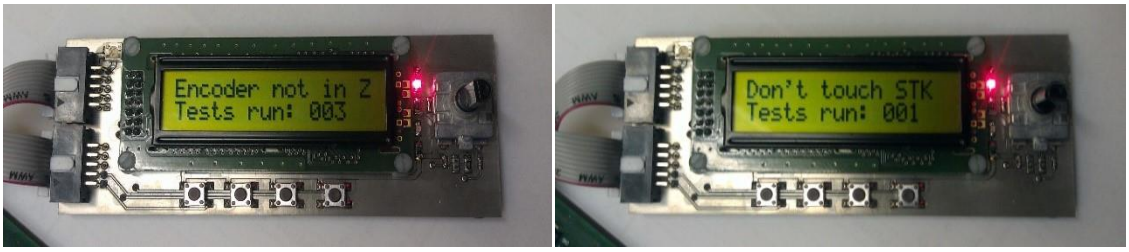


Figure 11 - LCD shows result

The pictures above show that the test fails when one of the test compontents fails.

The unit test is called from within the AppTaskStart function by `runUnitTest()`. The function `*all_tests()` returns a pointer to the result-value. In case of passing all tests, a green LED is lightning or a red LED in case of error. When the result is different than zero, an error has occurred and the pointer points to a string describing the kind of error. Finally the code will be stopped by an infinity loop.

```
uint8_t result = runUnitTest();
if (result == 0)
{
        while(1) {
                ;                       // in case of error stop the program.
        }
}
```

Testing of other hardware like the LCD-Display, however possible, is not implemented into the code. The HD44780U of LCD-Display has the feature to read the display memory to know what is actual displaying on it. Thus, it was not used in our code, because of time leak. The LED-bar has a HEF4894 as a driver, which cause impossible to test it.

5. Discussion

The aim of the project was to use the hardware given to us for the required system and to apply the knowledge acquired in class. The outcome is a simulator which simulates an aircraft in real time while using task scheduling so that the program would be schedulable and tasks meet deadlines.

Semaphore was used to protect resources. Semaphores can lead to deadlock, but that is not possible in this system, because of use of only one resource and one semaphore. That wouldn't be the case if we have used more advanced design with 2 resources and 2 tasks working with both of the resources.

Priority Inversion is neither possible since every task has to wait for the resource in order to do essential work. If, for example, task 2 did not use the resource, it could run before task 1 which would be waiting for task 3 to release the resource.

6. Conclusion

The general concept of the semester project was to create a system which controls the plane simulator, operating on its data and displaying information on lcd display and leds in a real-time operating system. Working on this project gives experience with real time systems, task scheduling, timers and semaphores. Also knowledge on how to use oscilloscope to measure

computation time of tasks was acquired. Not without difficulties, it was managed to a create fully functional system which is implementing all the requirements.

7. References

[B & W]: Alan Burns and Andy Wellings, “Real-Time Systems” and Programming Languages, 2009, ISBN 978-0-321-41745-9

Jean J. Labrosse , MicroC/OS-II: The Real Time Kernel

8. Appendices

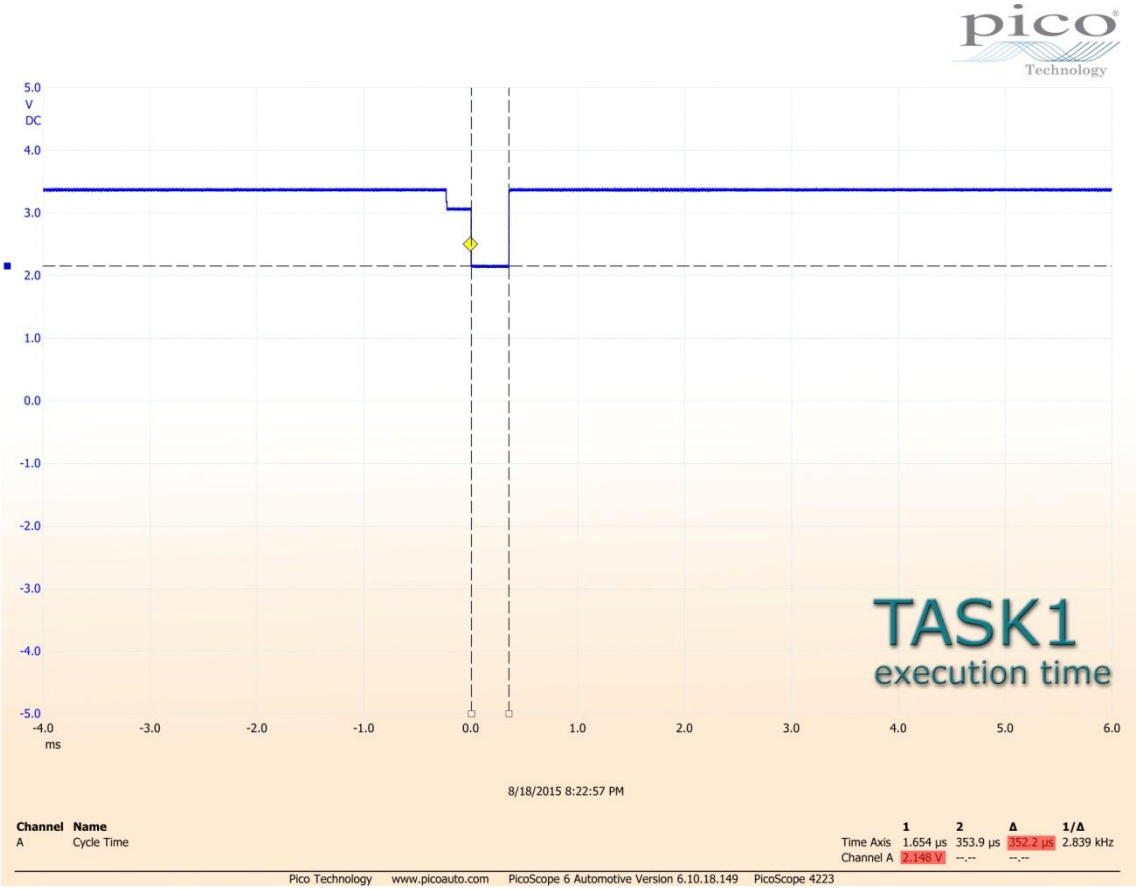
Control speed of engine motor 2015/08/14

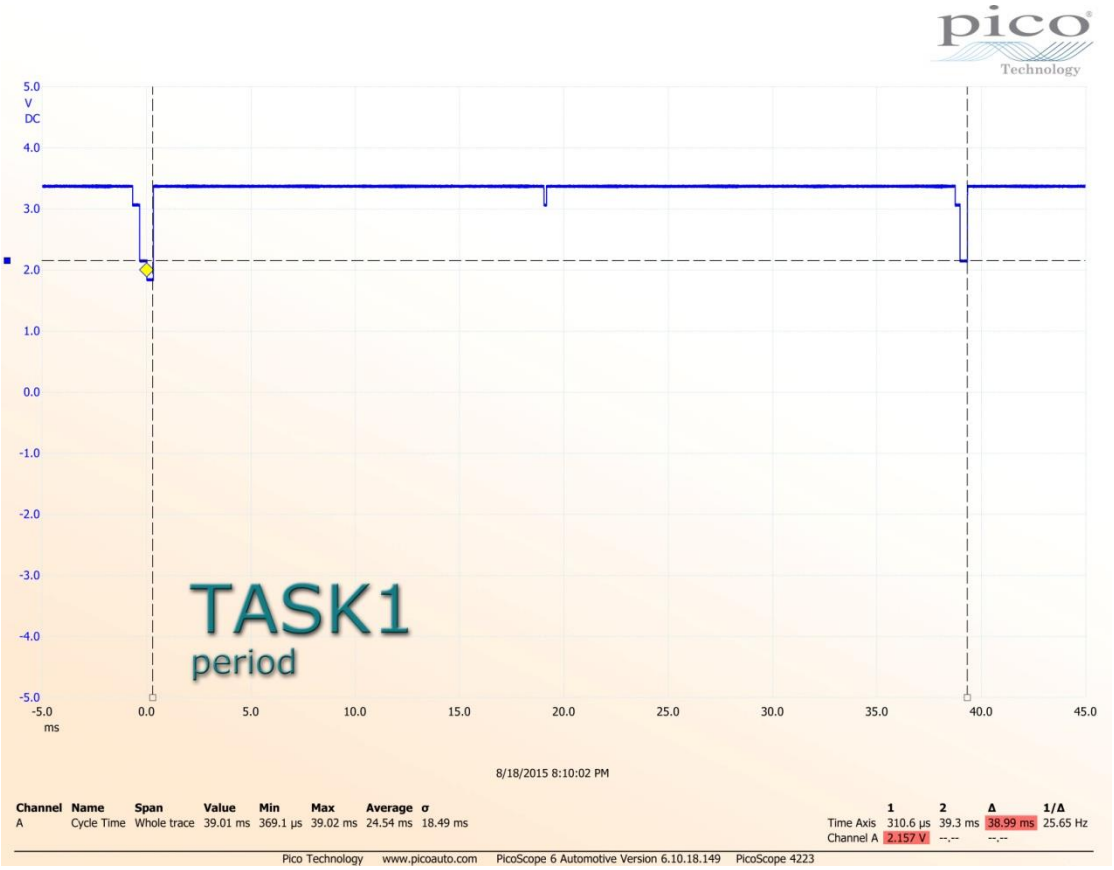
ITEM	VALUE
UseCase	Control speed of engine motor
Summary	User can control speed of the small motor with encoder
Actor	Pilot
Precondition	Motor must be turned on.
Postcondition	Pilot will be able to see speed of the motor on LCD.
Base Sequence	1. When encoder is turned to some direction, speed of the motor will be increasing. Then, if it is turned in other direction, speed will decrease.
Branch Sequence	
Exception Sequence	
Sub UseCase	
Note	

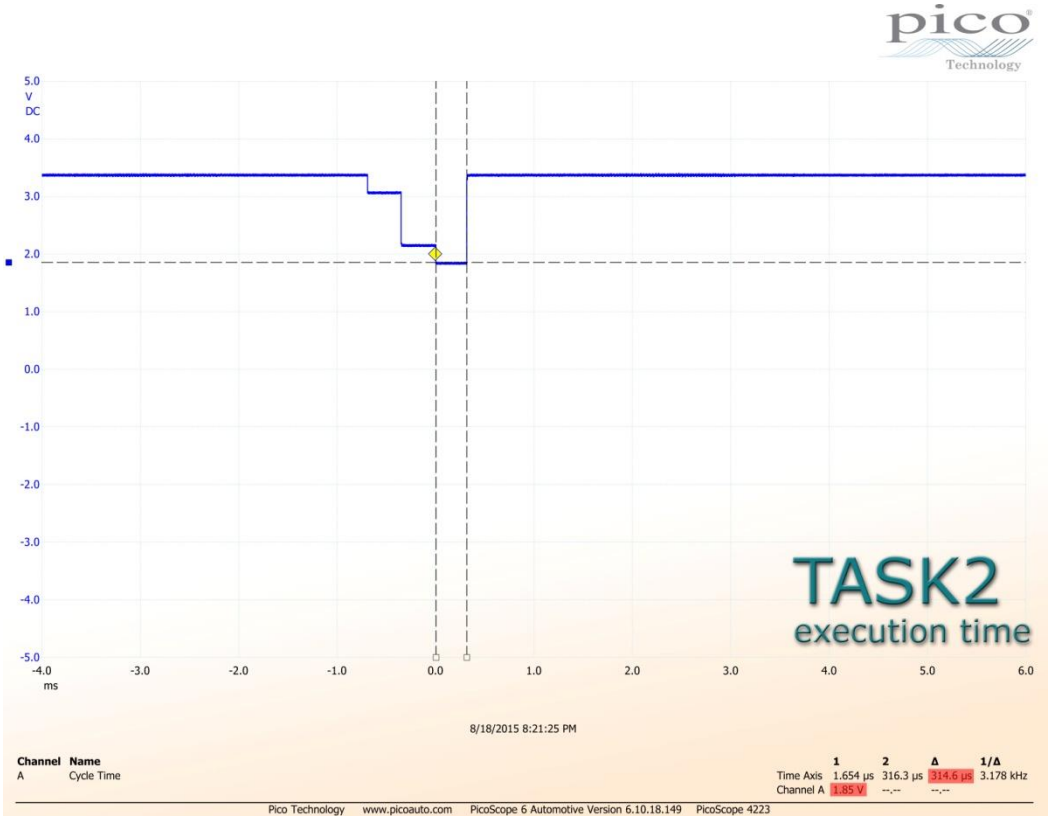
ITEM	VALUE
UseCase	select different display modes
Summary	switch between display modes
Actor	Pilot
Precondition	
Postcondition	
Base Sequence	The pilot has possibilities to switch between 4 different screens: turns on/off the display, speed, accelerometer data, temperature value.
Branch Sequence	
Exception Sequence	
Sub UseCase	
Note	

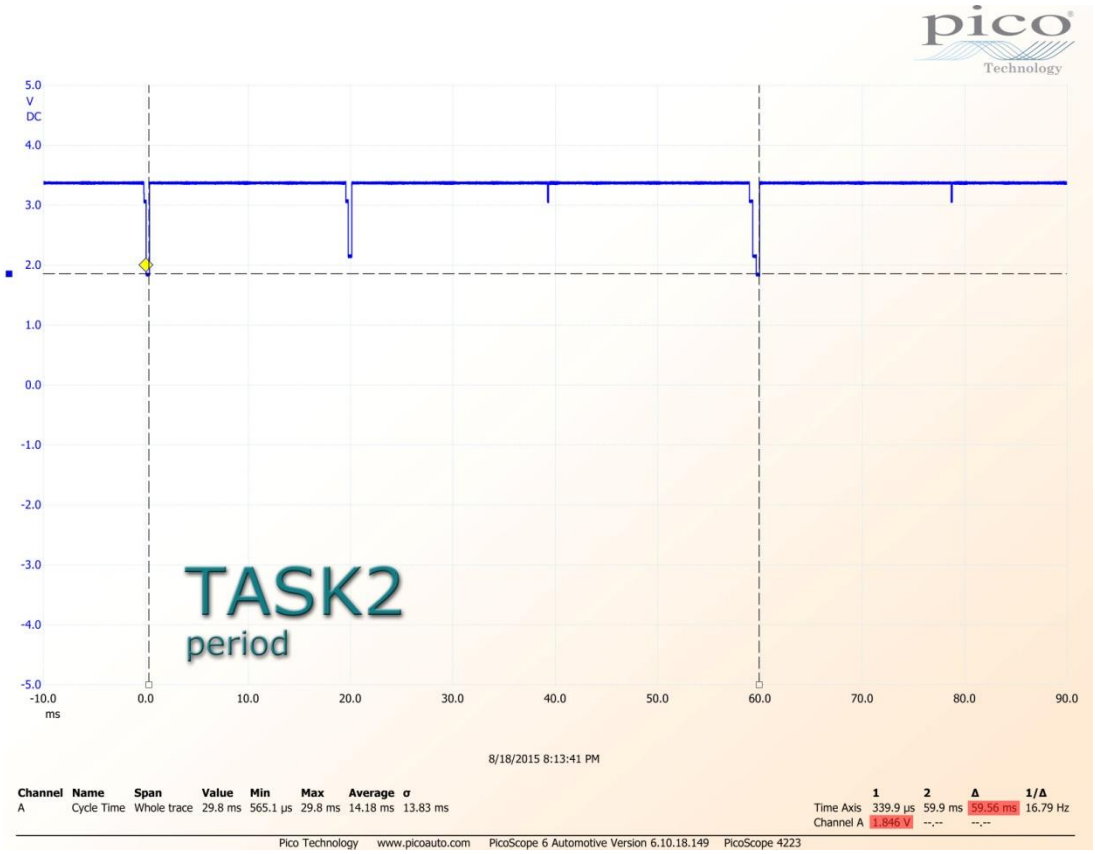
ITEM	VALUE
UseCase	Display speed on led bar
Summary	The speed of engine motor will be displayed on LED bar.
Actor	Pilot
Precondition	
Postcondition	
Base Sequence	With increasing speed, more leds will be lightened up on LED bar. With decreasing speed, less leds will be lightened up on LED bar.
Branch Sequence	
Exception Sequence	
Sub UseCase	
Note	

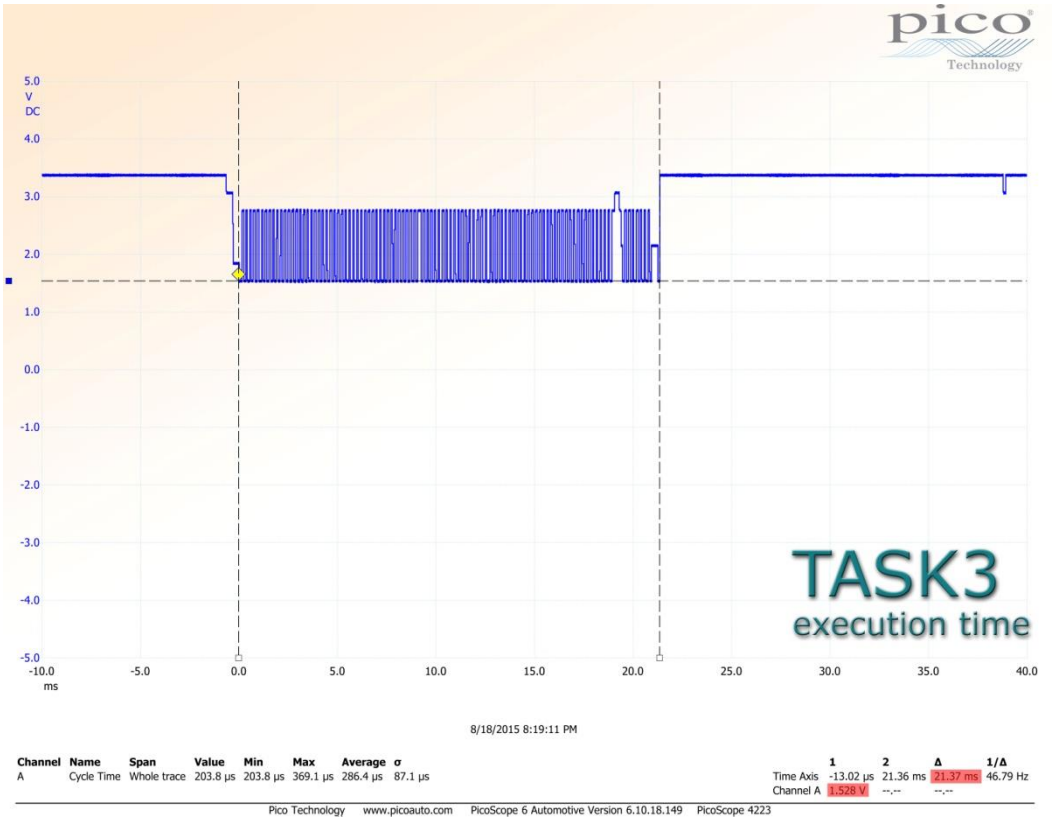
ITEM	VALUE
UseCase	Thrust control
Summary	Pilot can speed up/down the plane.
Actor	Pilot
Precondition	
Postcondition	Speed of the engine is controlled with the encoder.
Base Sequence	When the encoder is rotated the speed changes accordingly to the position of the encoder.
Branch Sequence	
Exception Sequence	There is a minimum and maximum speed.
Sub UseCase	
Note	

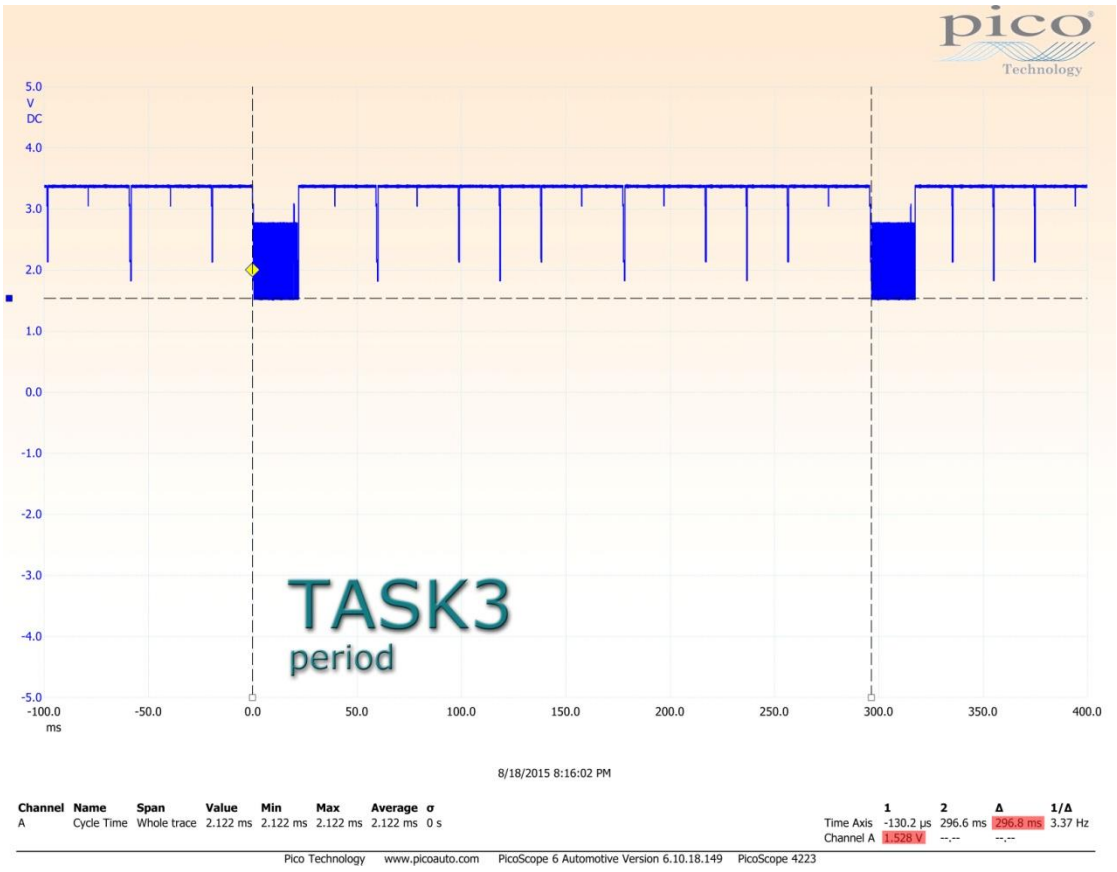

pico
Technology
5.0
V
DC
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
-4.0
-3.0
-2.0
-1.0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
ms
TASK1
execution time
8/18/2015 8:22:57 PM
Channel Name
A Cycle Time
1 2 Δ 1/Δ
Time Axis 1.654 µs 353.9 µs 352.2 µs 2.839 kHz
Channel A 2.148 V --.-- --.--
Pico Technology www.picoauto.com PicoScope 6 Automotive Version 6.10.18.149 PicoScope 4223


pico
Technology
5.0
V
DC
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
-5.0
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
40.0
45.0
ms
TASK1
period
8/18/2015 8:10:02 PM
Channel Name Span Value Min Max Average σ
A Cycle Time Whole trace 39.01 ms 369.1 µs 39.02 ms 24.54 ms 18.49 ms
1 2 Δ 1/Δ
Time Axis 310.6 µs 39.3 ms 38.99 ms 25.65 Hz
Channel A 2.157 V --.-- --.--
Pico Technology www.picoauto.com PicoScope 6 Automotive Version 6.10.18.149 PicoScope 4223


pico
Technology
5.0
V
DC
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
-4.0
-3.0
-2.0
-1.0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
ms
TASK2
execution time
8/18/2015 8:21:25 PM
Channel Name
A Cycle Time
1 2 Δ 1/Δ
Time Axis 1.654 µs 316.3 µs 314.6 µs 3.178 kHz
Channel A 1.85 V --.-- --.--
Pico Technology www.picoauto.com PicoScope 6 Automotive Version 6.10.18.149 PicoScope 4223


pico
Technology
5.0
V
DC
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
-10.0
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
ms
TASK2
period
8/18/2015 8:13:41 PM
Channel Name Span Value Min Max Average σ
A Cycle Time Whole trace 29.8 ms 565.1 µs 29.8 ms 14.18 ms 13.83 ms
1 2 Δ 1/Δ
Time Axis 339.9 µs 59.9 ms 59.56 ms 16.79 Hz
Channel A 1.846 V --.-- --.--
Pico Technology www.picoauto.com PicoScope 6 Automotive Version 6.10.18.149 PicoScope 4223


pico
Technology
5.0
V
DC
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
-10.0
ms
-5.0
0.0
5.0
10.0
15.0
20.0
25.0
30.0
35.0
40.0
TASK3
execution time
8/18/2015 8:19:11 PM
Channel Name Span Value Min Max Average σ
A Cycle Time Whole trace 203.8 µs 203.8 µs 369.1 µs 286.4 µs 87.1 µs
1 2 Δ 1/Δ
Time Axis -13.02 µs 21.36 ms 21.37 ms 46.79 Hz
Channel A 1.528 V --.-- --.--
Pico Technology www.picoauto.com PicoScope 6 Automotive Version 6.10.18.149 PicoScope 4223


pico
Technology
5.0
V
DC
4.0
3.0
2.0
1.0
0.0
-1.0
-2.0
-3.0
-4.0
-5.0
-100.0
ms
-50.0
0.0
50.0
100.0
150.0
200.0
250.0
300.0
350.0
400.0
TASK3
period
8/18/2015 8:16:02 PM
Channel Name Span Value Min Max Average σ
A Cycle Time Whole trace 2.122 ms 2.122 ms 2.122 ms 2.122 ms 0 s
1 2 Δ 1/Δ
Time Axis -130.2 µs 296.6 ms 296.8 ms 3.37 Hz
Channel A 1.528 V --.-- --.--
Pico Technology www.picoauto.com PicoScope 6 Automotive Version 6.10.18.149 PicoScope 4223