ALGORITHM FOR IoT-SENSOR DESIGN AND MAINTENANCE SERVICE

Mila Ilieva-Obretenova, Elena Blagoeva, Boiko Karkov

University of Mining and Geology "St. Ivan Rilski", 1700 Sofia; mila.ilieva@mgu.bg; elena.blagoeva@mgu.bg; boiko.karkov@mgu.bg

ABSTRACT. Internet of Things (IoT) is the extension of Internet to physical devices and everyday objects. Embedded with electronics, Internet connectivity, and other forms of hardware (such as sensors), these devices can communicate and interact with others over the Internet, and they can be remotely monitored and controlled. The paper offers an algorithm for IoT-sensor design and Maintenance Service with a high level of detail. It is intended for students and junior designers. The hypothesis is the following: unambiguous steps are defined by the usage of appropriate methods. The methodology includes a theory for: semiconductor devices, Wheatstone bridge, analogue-to-digital conversion with successive approximation, finite elements method, unified modelling language (UML) and programming language Verilog. The algorithm is illustrated by an example for a pressure sensor. The result is an algorithm in two parts: simulation design and service design. The simulation design includes hardware design and simulation elements calculation. The service design includes definitions of Managed Objects classes with names, attributes, operations and methods. With the chosen detail level, the algorithm is a good base for designing a system with two sensors (e.g. a pressure sensor and a temperature sensor) and for experiments in specific working environment (e.g. underground mine).

Keywords: IoT-sensor, pressure sensor, temperature sensor, simulation, service

АЛГОРИТЪМ ЗА ПРОЕКТИРАНЕ НА IoT -СЕНЗОР И ОБСЛУЖВАНЕ НА ПОДДЪРЖАНЕТО

Мила Илиева-Обретенова, Елена Благоева, Бойко Кърков
Минно-геоложки университет "Св. Иван Рилски", 1700 София

РЕЗЮМЕ. IoT (Internet of Things) е разширението на интернет към физически устройства и обекти за ежедневна употреба. Снабдени с електроника, интернет свързаност и други форми на хардуер (като сензори), тези устройства могат да комуникират и да взаимодействат с други чрез интернет и могат да бъдат дистанционно наблюдавани и контролирани. Предлага се алгоритъм за проектиране на IoT-сензор и обслужване на поддържането с висока степен на детайлизация. Предназначен е за студенти и младши проектанти. Хипотезата е следната: Еднозначност на стъпките се постига чрез използване на подходящи методи и средства. Методологията за проектиране включва теории за: полупроводникови елементи, Уитстонов мост, аналогово-цифрово преобразуване с последователна апроксимация, метод на крайните елементи, унифициран език за моделиране (UML) и език за програмиране Verilog. Алгоритъмът се илюстрира с пример за сензор за налягане. Резултатът се състои от алгоритъм в две части: проектиране на симулация и проектиране на обслужване. Проектирането на симулацията включва: проектиране на хардуер и изчисляване на елементите на симулацията. Проектирането на обслужването съдържа класове управлявани обекти с имена, атрибути, операции и методи. С избраната степен на детайлизация алгоритъмът представлява добра основа за проектиране на система с два сензора (например сензор за налягане и сензор за температура) и за експерименти в специфична работна среда (например подземен рудник).

Ключови думи: IoT-сензор, сензор за налягане, сензор за температура, симулация, обслужване

Introduction

IoT (Internet of Things) is the extension of Internet to physical devices and everyday objects. Embedded with electronics, Internet connectivity, and other forms of hardware (such as sensors), these devices can communicate and interact with others over the Internet, and they can be remotely monitored and controlled. The recent developments are represented in articles with low level of detail (Miller, 2018) or represent concepts of very high level (Ruh, 2018; Perera et al., 2018). This article offers an algorithm for IoT-sensor design and Maintenance Service with high level of detail. It is intended for students and junior designers. Unambiguous steps are defined by the usage of appropriate methods.

Methodology

The design methodology includes the application of the theory for: semiconductor devices (Lienig, 2017), Wheatstone bridge (Ekelof, 2001), analogue-to-digital conversion with successive approximation (Baker, 2010), finite element method (Logan, 2011), Unified Modelling Language – UML (Fowler, 2004) and programming language Verilog (Bergeron, 2012). The algorithm is illustrated with a pressure sensor.

Results

Simulation design

Hardware design. 1. Block-diagram design: Fig.1 shows a block-diagram of a monitoring system for the liquid level. It contains a sensor, an amplifier, an analogue-to-digital converter (ADC), a microcontroller, a RF-transmitter, a gateway, a power supply and a clock.

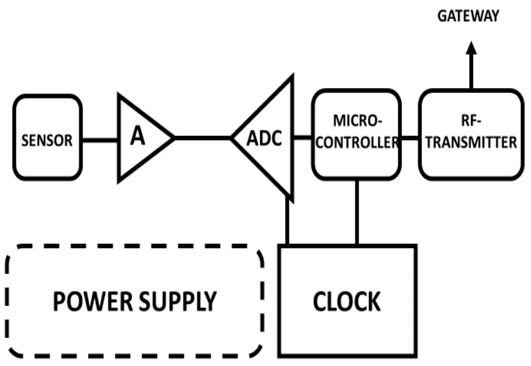

Fig. 1. Block-diagram of a monitoring system for the liquid level

2. Measure scheme for sensor resistance: Fig. 2 shows a measure scheme for sensor resistance with a Wheatstone bridge and an amplifier.

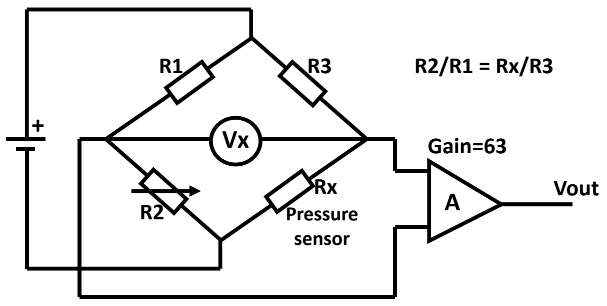

Fig. 2. Measure scheme for sensor resistance

3. Detailed block-diagram of ADC: Fig.3 shows a detailed block-diagram of ADC with successive approximation, where SAR is the Successive Approximation Register, Vref is the Reference Voltage, DAC is the Digital-to-analogue converter, Vin is the Input Voltage, S/H is the sample and hold circuit and EOC is the End of Conversion.

The operation of ADC with successive approximation is the following: SAR is initialised so that the most significant bit (MSB) is equal to a digital 1 (Vref). This code is fed into the DAC, it is converted to the analogue equivalent and is fed into the comparator circuit for comparison with the sampled input voltage Vin. If this analogue voltage exceeds Vin, the comparator causes the SAR to reset this bit (turns it to 0). Otherwise, the bit is left as 1. Then the next bit is set to 1 and the same test is done. The binary search continues until every bit in the SAR has been tested. The resulting code is the digital approximation of the sampled input voltage and is finally the output by the SAR at the end of the conversion. Fig.4 shows the operation of ADC with successive approximation for the binary digit 00000001.

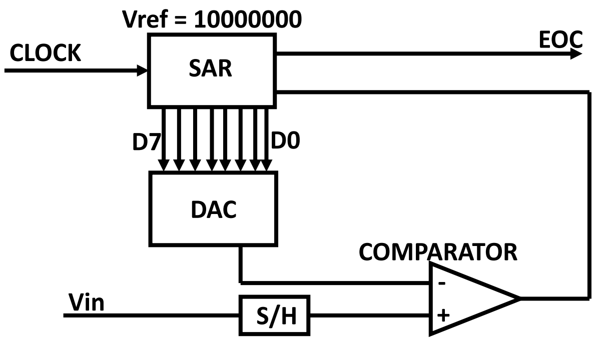

Fig. 3. Detailed block-diagram of ADC

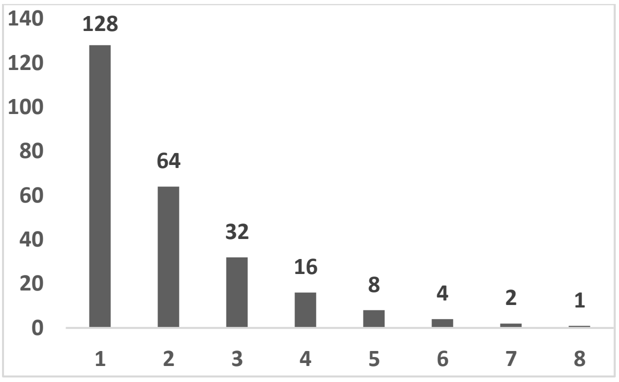

Fig. 4. Operation of ADC with successive approximation

4. System improvement: Fig.5 shows a system improvement with a group of sensors, which could be connected to one multiplexor (MUX), so they could monitor a group of parameters, characterising a certain territory – a mining territory, an environment territory, etc.

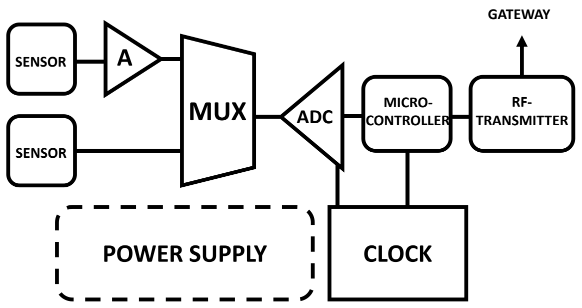

Fig. 5. System improvement: System with a group of sensors

Simulation elements calculation. 1. Sensor choice: A nonlinear resistor is chosen with resistance dependent on the applied pressure: R=f(P). 2. Calculation of dependence pressure-voltage: The voltage is measured as a function of pressure with the Wheatstone bridge: U=f(P). The minimum output voltage of the sensor is 0,1mV=1.10^{-4}V. The maximum output voltage of the sensor is 35mV=$3{,}5.10^{-2}$V. The maximum applied pressure is 350,25kPa. The pressure for 0,1mV in kPa is:

$$P = \frac{350.25x0.1}{35} = 1kPa \qquad (1)$$

3. Amplification calculation for the voltage after sensor: ADC works in the range from 0 to 2,2V. Therefore, the voltage after the sensor needs amplification A_U:

$$A_U = \frac{2.2}{3.5x10^{-2}} = 63 \qquad (2)$$

4. Calculation of successive approximation step of ADC: For the conversion of the voltage into a digital signal an 8-bit ADC is used. The possible counts are:

$$C = 2^8 = 256 \qquad (3)$$

The active counts are:

$$C_1 = 2^8 - 1 = 255 \text{ (From 0 to 255)} \qquad (4)$$

The approximation step is calculated from the ADC range and the possible counts are:

$$V = \frac{2.2}{256} = 8.59375.10^{-3} \text{ V / count} \qquad (5)$$

Table 1 shows visualization of voltage distribution in Excel.

Table 1: *Voltage Distribution in Excel*

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	Voltage,V	Pressure, kPa	Pressure, mm
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0.008594	0	0
0	0	0	0	0	0	1	0	0.017188	0	0
0	0	0	0	0	0	1	1	0.025781	0	0
0	0	0	0	0	1	0	0	0.034375	0	0
0	0	0	0	0	1	0	1	0.042969	0	0
0	0	0	0	0	1	1	0	0.051563	0	0
0	0	0	0	0	1	1	1	0.060156	0	0
0	0	0	0	1	0	0	0	0.06875	0	0
0	0	0	0	1	0	0	1	0.077344	0	0
0	0	0	0	1	0	1	0	0.085938	0	0
0	0	0	0	1	0	1	1	0.094531	0	0
0	0	0	0	1	1	0	0	0.103125	0	0
0	0	0	0	1	1	0	1	0.111719	0	0
0	0	0	0	1	1	1	0	0.120313	0	0
0	0	0	0	1	1	1	1	0.128906	0	0
0	0	0	1	0	0	0	0	0.1375	1.45	148
0	0	0	1	0	0	0	1	0.146094	2.9	296
0	0	0	1	0	0	1	0	0.154688	4.35	444
0	0	0	1	0	0	1	1	0.163281	5.8	592
0	0	0	1	0	1	0	0	0.171875	7.25	740
0	0	0	1	0	1	0	1	0.180469	8.7	888

5. ADC scale calculation: The ADC scale is calculated from the possible counts and the ADC range:

$$C_V = \frac{256}{2.2} = 116 \text{ counts / V} \qquad (6)$$

6. Calculation of Wheatstone bridge offset in V: We suppose that the Wheatstone bridge is not ideally balanced and prior to the pressure application the voltmeter reads 2mV=2.10^{-3}V. The voltage after the amplifier is:

$$V_{offset} = 2x10^{-3}x63 = 126x10^{-3}V = 0.126V \qquad (7)$$

7. Calculation of Wheatstone bridge offset in counts:

$$\text{offset} = 0.126x116 = 14.6 =\sim 15 \text{ counts} \qquad (8)$$

8. Calculation of active counts of the ADC reporting the pressure:

$$C_A = 255 - 15 = 240 \text{ counts} \qquad (9)$$

9. Calculation of the sensor slope in counts/kPa:

$$S_1 = \frac{240}{350.25} = 0.68522 \text{ counts/kPa} \qquad (10)$$

10. Calculating of the sensor slope in V/kPa:

$$S_2 = 0.68522x8.59375x10^{-3} = 5.88865x10^{-3} \text{ V/kPa} = \\ = 5.88865x10^{6} \text{ V/Pa} \qquad (11)$$

11. Defining the voltage function: The voltage function is linear from the type:

$$y = ax + b, \tag{12}$$

Where y is the output voltage V_{out} in V, as is the slope S_2 in V/kPa, x is the pressure P in kPa and b is offset V_{offset} in V.

$$V_{out} = S_2 x P + V_{offset} \tag{13}$$

12. Calculation of the dimension on X axe in kPa/count:

$$m= \frac{350.25}{240} = 1.45 \text{ kPa/count} \tag{14}$$

13. Calculation of the dimension on X axis in mm/count: For practical aims let's assume that the fluid height is directly proportional to the pressure:

$$P=\rho gh, \tag{15}$$

Where P is the pressure in kPa, ρ is the fluid density (for water ρ = 1kg/l), g is the standard gravity 9,8m/s^2 и h is the fluid height over the sensor in meters. Then

$$\text{Factor} = \frac{P}{\rho x g} = \frac{1.45}{1x9.8} = 0.147959 \text{ m/count} = 148 \text{ mm/count} \tag{16}$$

Fig.6 shows the dimensions: dimension on X axis in kPa/count and dimension on X axis in mm/count.

The figure shows that 1kPa corresponds approximately to 100 mm.

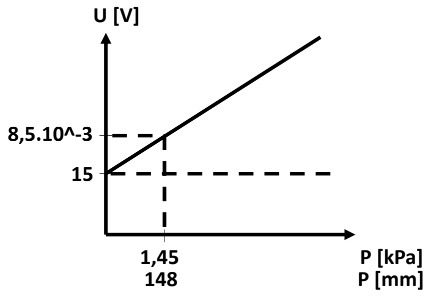

Fig. 6. Dimensions on X axis

Checking: For 117 counts: (117-15)x1.45 = 147.9kPa, $\frac{147.9}{9.8}$ =15.09 m, (117-15)x148 = 15096 mm;

for 219 counts: (219-15)x1.45=295.8kPa, $\frac{295.8}{9.8}$ = 30.184m, (219-15)x148=30192mm. The check shows a difference in the calculations of about 1cm.

14. Definition of the function for fluid height in mm:

$$h = (\text{ncounts} - \text{offset}) * \text{factor}, \tag{17}$$

where ncounts is the number of counts of ADC, corresponding to the analogue value of the sensor voltage.

Service Design

UML diagram synthesis. Fig.7 shows a UML diagram for Managed Objects classes for Maintenance Service.

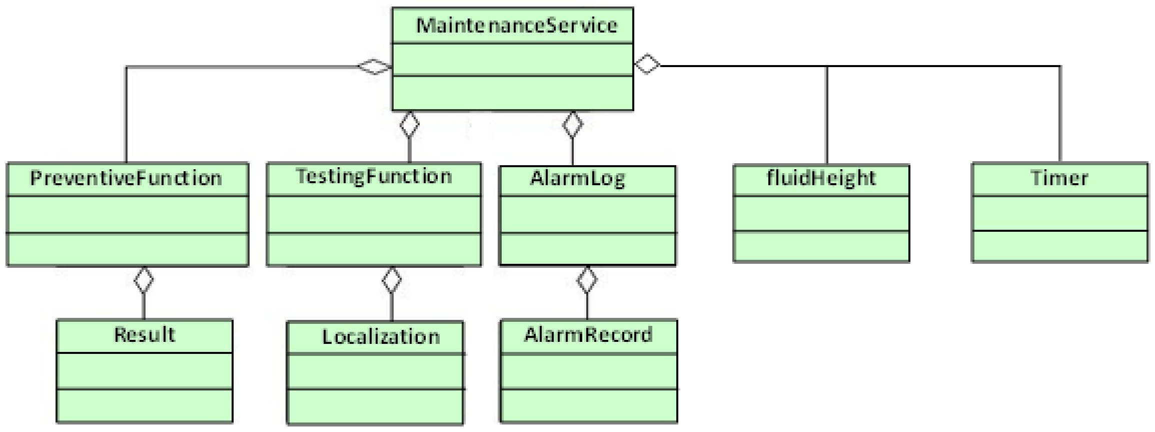

Fig. 7: UML diagram of Managed Objects classes for Maintenance Service

Definitions of Managed Objects (MO) classes with names, attributes, operations and methods.

1. MO **MaintenanceService** represents the information for maintenance service, which helps to monitor the process in normal working conditions and to avoid failures.
2. MO **fluidHeight** represents a function for calculation of sensor constants. A) Attributes: Attribute **gravity** represents gravitational field. Attribute **density_of_water** represents one of the physical properties of water – density. Attribute **ADCrange** shows the maximum voltage of ADC. Attribute **ADCbits** represents the number of ADC bits. Attribute **pressureMaximum** represents the maximum pressure applied on the sensor. Attribute **sensorOutput_min** shows the minimum output voltage of sensor. Attribute **sensorOutput_max** shows the maximum output voltage of sensor. Attribute **WBridgeIntercept** shows the initial voltage in Wheatstone bridge. Attribute **ncounts** represents the number of counts reached by certain pressure. B) Operations: Operation **MinPressure** calculates the minimum pressure measured by the sensor. Operation **Amplification** calculates the necessary amplification so that the sensor range corresponds to ADC range. Operation **ADCcounts** calculates the number of ADC counts. Operation **ADCfactor** calculates the ADC step in V/count. Operation **ADCscale** calculates the ADC scale in counts/V. Operation **InterceptVoltageAfterAmplifier** calculates the initial voltage after amplifier. Operation **offset** calculates the initial voltage, reported from ADC. Operation **ActiveADCcounts** calculates

the ADC counts with pressure. Operation **Slope1** calculates the slope in counts/kPa. Operation **Slope2** calculates the slope in V/kPa. Operation **PressureScale** calculates the dimension on X axis in kPa/count. Operation **Factor** calculates the dimension on X axis in mm/count. Operation **read_ncounts** reads the ADC counts by certain pressure. Operation **fluidHeight** calculates the liquid height over the sensor.

3. MO **Timer** represents a timer, which awakes the processor periodically for fluid height measurement. A) Attributes: Attribute **bHaveLastRead** is a Boolean variable, which shows the processor is awakened. Attribute **BigPeriod** shows the working period of the timer, i.e. 50000000ns=50000us. Attribute **PeriodsNumber** represents the number of periods „asleep-awakened“ of the processor for the timer`s period. B) Operations: Operation **UartStdOutnit** initialises the timer`s turn off. Operation **SysTick_Config** calculates the duration of one period of the timer.

4. MO **PreventiveFunction** represents the programmes for testing the fluid height by normal working conditions.

A) Attributes: Attribute **LastRead** represents the value of the last read. Attribute **mmH2O** shows the value of recent reading. Attribute **permissible_delta** represents the allowed difference in mm, i.e. 500mm=50cm. B) Operations: Operation **delta** calculates the difference between the last and the recent reading. Operation **assign_lastRead** assigns the recent reading as the last reading.

5. MO **Result** represents the result after execution of a preventive function. A) Attributes: Attribute **ID** represents the notification number after the preventive function. B) Operations: Operation **printf** writes a notification after the preventive function.

6. MO **AlarmLog** represents log Alarms, containing the collected maintenance events. A) Attributes: Attribute **ID** represents the log number. Attribute **N** represents the log size, i.e. 1000 records. B) Operations: At this stage operations for the log are not defined.

7. MO **AlarmRecord** represents the Alarm reaction, if a failure occurs. A) Attributes: Attribute **ID** represents the alarm number. B) Operations: Operation **printf** writes an alarm in log by exceeding the allowed difference.

8. MO **TestingFunction** represents the testing programmes for detection of failure: 1.Calculation of the counts number by increasing or decreasing the liquid level over the allowed difference. 2. By reaching a crucial level [mm], i.e. 10000mm=10m, visualisation of the decreasing level speed starts. The crucial level in counts is 83. A) Attributes: Attribute **adc_dac_data** shows the liquid level in counts. Attribute **liquid_level** shows the liquid level in mm. Attribute **gravity** shows the earth gravity. Attribute **density_of_water** shows the water density. Attribute **steadyTime** shows the steady time for procedure execution, i.e. 750 000ns=750us. Attribute **leakingTime** shows the time for leaking simulation, i.e. 10000ns=10us. B) Operations: Operation **Pressure** calculates the pressure by the liquid level in mm and earth gravity. Operation **V$_{out}$** calculates the output voltage by slope, pressure and initial voltage. Operation **ncount** calculates the counts number by output voltage and ADC scale. Operation **liquid_level** calculates leaking 1% every 1us.

9. MO **Localisation** contains information about the counts by which a failure is reported. A) Attributes: Attribute **ID** represents the number of localisation. B) Operations: Operation

assign adc_compare assigns the next count smaller than the calculated.

Synthesis of operations` methods. 1. fluidHeight

```
Int fluidHeight (int ncounts) {
        /* model for pressure sensor:
           MinPressure = PressureMaximum * sensorOutput_min / sensorOutput_max
           Amplification=ADCrange/sensorOutput_max
           ADCcounts = 2^ADCbit – 1
           ADCfactor = ADCrange/2^ADCbit
           ADCscale = 2^ADCbit/ADCrange
           InterceptVoltageAfterAmplifier = WbridgeIntercept * Amplification
           offset = InterceptVoltageAfterAmplifier * ADCscale
           ActiveADCcounts = ADCcounts – offset
           Slope1 = ActiveADCcounts / pressureMaximum
           Slope2 = Slope1 * ADCfactor
           PressureScale = PressureMaximum / ActiveADCcounts
           Factor = PressureScale / Density_of_water * gravity
        */

        const int offset = 15;
        const int Factor = 148;

        return (ncounts – offset) * Factor;
}
```

2. Timer

```
Int main (void)

{       bHaveLastRead = 0;

       // UART init
        UartStdOutInit ();

        // set SysTick interrupt timer
        SysTick_Config (50000000/10000);

        // go to sleep
        while (1)
                PreventiveFunction ();
        return 0;
}
```

UART is a Universal Asynchronous Receiver-Transmitter. The period of the timer is 5µs.

3. PreventiveFunction. The method for PreventiveFunction combines attributes and operations for the following objects: **PreventiveFunction, Result, AlarmLog и AlarmRecord**.

```
void PreventiveFunction (void) {
        int n_c = ncounts;
        int mmH2O = fluidHeight(n_c);
        printf (“ADC reports %d count, %d mmH2O\n”,
n_c, mmH2O);
        if (!bHaveLastRead) {
                bHaveLastRead=1;
                 lastRead=mmH2O;
        }
        else { int delta = mmH2O – lastRead;
                if ((delta < -500) II (delta > 500)) {
                        printf (“***\n*** Significant
        change in fluid level: %d mm, now %d
        mmH2O\n***\n”, delta, mmH2O);
                        lastRead=mmH2O;
                }
        }
}
```

4. TestingFunction. The method TestingFunction unifies the attributes and operations of the following objects: **TestingFunction** и **Localisation**.

```
module TestingFunction (adc_dac_data, adc_compare)

        input [7:0] adc_dac_data;
        output      adc_compare;

        real        liquid_level, pressure, Vout, ncount;

        initial begin
                liquid_level = 10000; //mm

                // hold steady for the first 750 us
                #750000;

                // start leaking 1% every 1us
                forever begin

                liquid_level = liquid_level * 0,99;
                #10000;
                end
        end // initial begin

        always @(liquid_level)
                begin
                  pressure=liquid_level*9,8;
                  Vout=pressure*5,88865.e-6+0,126
                  ncount=Vout*256/2,2
                end

        assign adc_compare = adc_dac_data < ncount;

endmodule // TestingFunction
```

It could be seen that the fluid level falls with 1 m for 10 μs.

Conclusion

The paper represents an algorithm for the design and service of an IoT-sensor. It contains two parts and meets the requirements of designers and service management. The contributions of paper are as follows:

1. The steps of an algorithm for design and simulation of IoT-sensor are defined with high level of detail.
2. The steps of an algorithm for Maintenance Service are defined. An object-oriented method is used. The Managed Objects (MO) classes, organised in hierarchy, are defined. The attribute, operations and methods of classes are defined.
3. In order to be verified, the algorithm is illustrated with a pressure sensor.
4. The integration of two areas is demonstrated– sensor design and service management – by the interaction of steps from both levels.
5. The steps for reusage are developed: within the frame of one level – for design of different kinds of sensors (for temperature, for humidity, for illumination, etc.) or for service in different functional areas (configuration, security, performance etc.); between the two levels – for sensor design and for service design.
6. A scheme for future work with two or more sensors and a multiplexor is proposed.

The future work could be considered in the following aspects:

1. Development of new functional elements – sensors for different environments;
2. Service modelling for the new elements and development of corresponding information models;
3. Modelling of sensor communications and considering sensors as network elements;
4. Integration of IoT with other platforms, i.e. Smart Grid.

References

Baker, J. 2010. *CMOS circuit Design, Layout, and Simulation, 3rd Edition.* Wiley-IEEE; 1208 p.

Bergeron, J. 2012. *Writing Testbenches: Functional Verification of HDL Models (2nd ed.).* Springer.

Ekelof, S. 2001. The Genesis of the Wheatstone Bridge. *Engineering Science and Education Journal, 10*, 1, 37–40.

Fowler, M. 2004. *UML Distilled: A Brief Guide to the Standard Object Modelling Language. 3rd ed.* Addison-Wesley Professional.

Lienig, J., H. Bruemmer. 2017. *Fundamentals of Electronic Systems Design*. Springer International Publishing. p. 1.

Logan, D. L. 2011. *A first course in the finite element method*. Cengage Learning.

Miller, J. 2018. *Proof-of-concept: The day after*. Mentor® A Siemens Business. www.mentor.com

Perera, C., Mahmoud Barhamgi, Supama De, Tim Baarslag, Massimo Vecchio, Kim-Kwang Raymond Choo. 2018. Designing the Sensing as a Service Ecosystem for the Internet of Things. – *Internet of Things Magazine, December 2018, 1,* 2.

Ruh, W. 2018. Drilling Deep into Digital Industrial Transformation will determine who survives and thrives. – *Internet of Things Magazine*, *September 2018*, *1*, 1.