**NATIONAL AVIATION UNIVERSITY.**

**INSTITUTE OF INFORMATION DIAGNOSTIC SYSTEMS**

**DEPARTMENT OF AVIATION COMPUTER-INTEGRATED COMPLEXES**



**Term project**

By discipline

Methods of mathematical modelling

Done by:

students of IIDS 533

Illya Tretyakov, Michael Panarin

Kyiv 2017

1. **Introduction**

After creation of wind generator, it’s necessary to measure generator`s parameters, such as rotation speed and the force which is produced by the rotor.

For these aims measuring devices set and software were created.

This paper contains short devices description, software manual and user’s guide.

1. **Devices description**

Measuring devices set contains following elements:

* 1. Tachometer **Uni-t UT372**



**Specification:**

|  |  |
| --- | --- |
| Measurement: | 10 to 99.999 RPM |
| Range selection | 10 to 99 RPM, 100 to 999 RPM, 1000 to 9999 RPM, 10000 to 99999 RPM |
| Accuracy | 0.04% ±2dgt |
| Display | 99999 |
| Target distance | 50 mm to 200 mm |
| Start/Hold: | Start and Stop measurement, Data Hold |
| Max./min./ave/zero: | Maximum value, minimum value, average value, zeroing |
| Sleep Mode | Around 15 minutes |
| Low battery display | 4.8 V |
| USB interface | Yes |

**General characteristics:**

|  |  |
| --- | --- |
| Power: | 1.5V batteries (AA) x 4 |
| LCD size | 54 x 41 mm |
| Product size | 182 x 56 x 34 mm |
| Product weight | 100 g |

* 1. Force transducer **HBM C9C**



**Опис:**

Компактний та економічно вигідний датчик сили типу C9C надійно вимірює силу стискування. Датчик сили дозволяє вирішити завдання як динамічних, так і статичних вимірювань. Завдяки високій фундаментальній частоті, C9C також підходить для дуже швидких вимірювань. Його зварний дизайн з нержавіючої сталі робить датчик сили надзвичайно міцним.

* 1. Індикаторна панель для датчика тиску **HBM WE2107**

WE2107 weighing indicators are an amplifier for connection to commercially available strain gage load cells or scales. The load cell signal is amplified and digitally converted; all further processing steps then follow digitally in a microprocessor.

WE2107 - це підсилювач. Сигнал навантажувача посилюється і перетворюється цифровим способом; всі подальші кроки обробки слідують цифровим чином у мікропроцесорі.

WE2107 має:

* З'єднання навантажувальних елементів: до шести з навантаженням 350 або мінімум 58
* • один COM-порт для послідовного зв'язку з ПК / ПЛК (RS232 або RS485)
* 2 цифрових контрольних входи
* 4 цифрових виходів (перемикачі лімітного значення або управління заповненням / дозуванням)
* аналоговий вихід (4 ... 20 мА)
* дві функціональні клавіші (визначені користувачем)

**Додаткові можливості:**

* Точність до 6000е (0,8 мкВ / е)
* Функції меню можна вимкнути / увімкнути
* Вибір фільтра
* Максимальне регулювання потужності, часткове регулювання потужності, калібрування мВ / В
* Лінеаризація діапазону зважування
* 4 граничних значення перемикачів з гістерезисом
  1. Мультиметр **UT61b** (два)



|  |  |
| --- | --- |
| Тип | цифровой |
| Выбор диапазона | автоматический выбор диапазона |
| Разрядность | 3999 |
| Постоянное напряжение | до 750 В |
| Переменное напряжение | до 750 В |
| Сопротивление | до 40 МОм |
| Емкость | до 4000 мкФ |
| Частота | до 10 МГц |
| Постоянный ток | до 10 А |
| Переменный ток | до 10 А |
| Температура | Есть |
| Звуковая прозвонка цепи | Есть |
| Подключение к ПК | USB |
| Тестирование диодов | Есть |
| Аналоговая гистограмма | Есть |

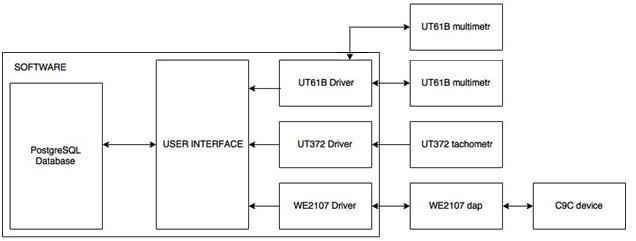
Схема блоку, яка показує взаємозв'язки між функціональними блоками наступної системи, показана на рис.1.

Рис.1. Блок-схема встановлених пристроїв.

1. **ПРОГРАМНЕ ЗАБЕЗБЕЧЕННЯ**
   1. **БАЗА ПРОГРАМИ**

Це програмне забезпечення було створено на Python 3.6 для драйверів пристроїв з додатковими бібліотеками: **pywinusb, pyserial**.

І C# 7.1 для інтерфейсу, що використовує пакунок **windows forms**.

* 1. **PYTHON 3.6**

**Python** — інтерпретована об'єктно-орієнтована мова програмування високого рівня з динамічною типізацією.Розроблена в 1990 році Гвідо ван Россумом. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується декілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована

**Чому python?  
Читабельність**

Python дуже нагадує англійську мову, використовуючи слова "не" та "в". Також допомагає дуже суворі правила пунктуації Python.

Крім того, Python має набір правил, відомих як PEP 8, які розповідають кожному розробнику Python, як форматувати свій код. Це означає, що ви завжди знаєте, де слід розміщувати нові рядки, і що ще важливіше, що майже будь-який інший сценарій Python, який ви виберете, незалежно від того, написано новим чи досвідченим професіоналом, буде виглядати дуже схожим і таким же простим для читання.

* + 1. **pywinusb**

**Вступ**

Ця бібліотека є простим звязком USB / HID для користувача.

**Переваги**

* 100% python code
* Високошвидкісна обробка подій HID (використання подій)
  + 1. **pyserial**

**Вступ**

Цей модуль інкапсулює доступ до послідовного порту. Він забезпечує зворотні версії Python для Windows, OSX, Linux, BSD (можливо, будь-яка система, сумісна з POSIX) та IronPython. Модуль, названий "serial", автоматично вибирає відповідний бекенд.

Він випущений за ліцензією на безкоштовне програмне забезпечення.

**Особливості**

* Інтерфейс на тому ж класі на всіх підтримуваних платформах.
* Доступ до налаштувань порту за допомогою властивостей Python.
* Підтримка різних розмірів байтів, стоп-бітів, паритету та регулювання потоку за допомогою RTS / CTS та / або Xon / Xoff.
* Порт налаштований для бінарної передачі. Немає видалення байтів NULL, переклад CR-LF і т. Д. (Які багато разів увімкнено для POSIX.) Це робить цей модуль універсально корисним.
  + 1. **C# TODO:**

**4.2 МОДУЛІ**

**TODO**

Fig.2. Software block scheme

Всі модулі можуть бути використані як окремі або повністю в іншій програмі.

Вони були розроблені для визову інтерфейсом, але можуть також використовуватися будь-якою іншою підпрограмою.

**4.2.1. serial\_read.py**

Модуль створено для підключення до пристрою "we2107" і зчитує дані з нього. Для роботи потрібна бібліотека pySerial

**class we2107** –клас для об'єктів пристрою

**we2107.\_\_init\_\_** – конструктор з необхідними параметрами

**we2107.read\_data** – функція, яка повертає серійні дані з конкретного COM-порту як рядок

**we2107.\_\_del\_\_** – Деструктор класу, щоб переконатися, що з'єднання закрито належним чином

**\_\_name\_\_ == ‘\_\_main\_\_’ –** основна функція, яку запускаєтся інтерфейсом

**4.2.2. tacho\_read**

**class ut372device** – клас для об'єктів тахометру

**ut372device.\_\_init\_\_** – конструктор з необхідними параметрами

**ut372device. sample\_handler\_count (data)** – Викидає сміття та збирає повний пакет даних в глобальний список

*data –* об'єкт даних Дані з пристрою.

**ut372device. \_positioning\_count** – приватна функція, яка аналізує вихідні дані та форми

**ut372device.connect –** Функція, яка підключається до пристрою

**ut372device.** **\_send\_init\_packet –** приватна функція, яка надсилає пакет ініціалізації на пристрій

**ut372device.** **receive\_package –** Функція, яка очікує на отриману подію пакету та обробляє відключення пристрою

**\_\_name\_\_ == ‘\_\_main\_\_’ –** основна функція, яку запускаєтся інтерфейсом

**4.2.3. power\_calc**

**class PowerCalc –** клас для розрахунку потужності

**PowerCalc.\_\_init\_\_** - конструктор функція з необхідними параметрами

**PowerCalc.frequency** – функція обчислення частоти

**PowerCalc.conv\_force** – функція перетворення сили

**PowerCalc.torque** – функція для розрахунку крутного моменту

**PowerCalc.power** – функція для розрахунку потужності

**\_\_name\_\_ == ‘\_\_main\_\_’ –** основна функція, яку запускаєтся інтерфейсом

**4.2.4. ut61b**

**class UT61B –** клас для підключення UT61b

**UT61B.\_\_init\_\_** - конструктор функція з необхідними параметрами

**UT61B.read\_raw\_data** – функція, яка зчитує пакет даних з пристрою ut61b

**UT61B.is\_data\_valid** – перевіряє правильність даних

**UT61B.get\_meas** – мапить отримані дані в рядковий словник

**UT61B.\_\_del\_\_** - Деструктор для правильного закриття з'єднання

**\_\_name\_\_ == ‘\_\_main\_\_’ –** основна функція, яку запускаєтся інтерфейсом

**4.2.5. Interface TODO**

**5. USER GUIDE**

**5.1 Setting up devices**

First of all, it’s necessary to set up the devices.

For UNI-T UT372:

1. Turn on the device.
2. Long press **R/C** button, until setup menu will appear.
3. Turn USB mode on by pressing **M/M/A** button. The result should look like “USB 1” in the following menu.
4. Turn back to the measurement mode by pressing **ON/OFF** button.

For UNI-T UT61b:

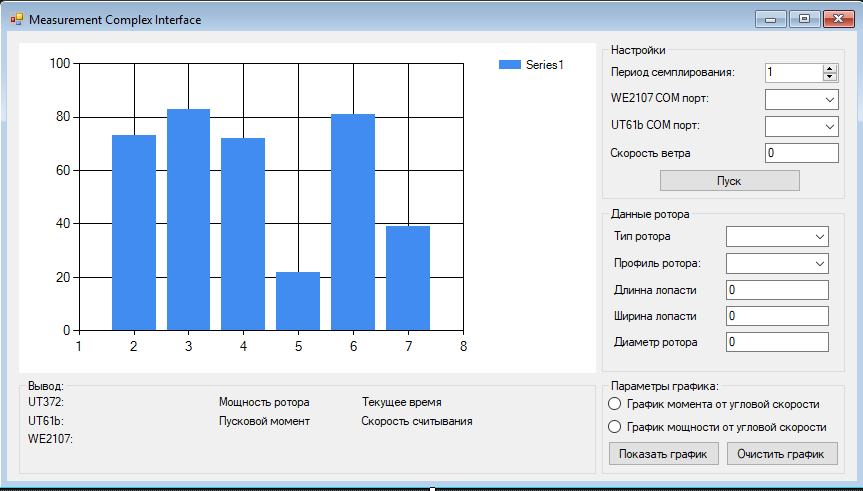
1. Turn on the device.
2. Long press **«rel ∆»** button, computer symbol shows up.

For HBM WE2107:

1. Ensure that load cell (C9C) is connected to the device.
2. Connect WE2107 via COM port (or use COM-to-USB converter cable).
3. Check up the device settings:
   1. Press ENTER and UP buttons simultaneously to call menu.
   2. Press NEXT button 3 times for **UART1** to appear on the screen, press ENTER.
   3. Addr is not necessary to set up, press NEXT and ENTER when bAudr appears. For correct working of the software it’s necessary to set baud rate to 2400 by clicking NEXT button.
   4. After setting the baud rate press CE to set PAIRITY which should be EVEN. Go back using CE button.
   5. Press CE button 2 times to exit settings menu.

**5.2 Using executable file**

**MeasureComplexInterface.exe –** executable file which launches the interface software in project folder.



Execution process:

1. Open the program, set up the parameters in Settings block.
2. Set up parameters in Rotor Data block.
3. Optionally you can select between radio buttons which allows you to see the chart for chosen values.
4. Outputs are shown inside Output block.

Data is writing then in the database.

**5.3 Using Sources**

**5.3.1 Python installation**

Before you start, you will need Python on your computer, but you may not need to download it.

First of all check that you don't already have Python installed by entering **python** in a command line window. If you see a response from a Python interpreter it will include a version number in its initial display. Generally any recent version will do, as Python makes every attempt to maintain backwards compatibility.

**For Windows**: the most stable Windows downloads are available from the Python for Windows page.

**5.3.2 pywinusb installation**

The most convenient way of installing is using easy\_install or pip

Run “pip install pywinusb” command in command line

**5.3.3 pySerial installation**

pySerial can be installed from PyPI using:

pip install pyserial

or:

easy\_install -U pyserial

in command line

**5.3.4 pyodbc installation**

To install, use pip or setuptools, run

pip install pyodbc

in command line

**5.3.5 Getting software sources**

Cloning the github repository is a valid option: <https://github.com/PanarinM/com_slash_usb/tree/interface> to get sources for further usage.

**5.3.6 Starting the program**

* Navigate to the sources folder via “CD” command in command line
* Connect devices
* Run “python reader.py” command to start the program
* Follow instructions from 5.1.2

**6. Code listing**

**6.1 reader.py**

*from serial\_read import we2107*

*from pyodbc import Error*

*from tacho\_read import ut372device, DeviceIsNotConnected*

*from tacho\_acess\_db import AccessConnect*

*import serial.tools.list\_ports*

*from serial import SerialException*

*from os import listdir, path*

*from sys import exit*

*def \_\_db\_choose():*

*"""*

*Lists \*.accdb Access database files in the current folder. Then asks user to choose one*

*:return: String with path to the DB*

*"""*

*counter = 0*

*dbfiles = []*

*for file in listdir('.'):*

*if file.endswith('.accdb'):*

*counter += 1*

*print('{}. {}'.format(counter, file))*

*dbfiles.append(file)*

*if counter == 0:*

*print('No Accsess DB files found')*

*input()*

*exit()*

*while True:*

*userinput = int(input('Choose the DB file: '))*

*if userinput > counter or userinput < 0:*

*print('Wrong input!')*

*continue*

*else:*

*return path.realpath('.\\{}'.format(dbfiles[userinput-1]))*

*def \_\_comchoose():*

*"""*

*Lists availible COM ports. Then asks user to choose one*

*:return: String with com port*

*"""*

*comlist = serial.tools.list\_ports.comports()*

*for com in comlist:*

*print('{}. {} {}'.format(comlist.index(com) + 1, com.device, com.description))*

*if len(comlist) == 0:*

*print('No COM devices found')*

*input()*

*exit()*

*while True:*

*userinp = int(input('Input COM number: '))*

*if userinp > len(comlist) or userinp < 0:*

*print('Wrong input')*

*exit()*

*else:*

*return comlist[userinp-1].device*

*if \_\_name\_\_ == "\_\_main\_\_":*

*comnumb = str(\_\_comchoose())*

*pathtodb = \_\_db\_choose()*

*try:*

*db = AccessConnect(pathtodb, 'table')*

*except Error:*

*print('Error occured during the database connection. Check if Microsoft Access Driver is installed and availible')*

*input()*

*exit()*

*try:*

*pressure = we2107(comnumb)*

*except SerialException:*

*print('Wrong COM port for pressure device')*

*input()*

*exit()*

*try:*

*tacho = ut372device()*

*print('Tachometer device was found by product ID = {} and vendor ID = {}'.format(tacho.product\_id,*

*tacho.vendor\_id))*

*except DeviceIsNotConnected:*

*print('Tachometer device was not found by specified product ID and vendor ID')*

*input()*

*exit()*

*print('------------------------------------------')*

*while True:*

*tacho\_data = tacho.receive\_package()*

*if tacho\_data[2] is None or tacho\_data[2] == 0:*

*continue*

*db.add\_record(tacho\_data[2], 'ut372', tacho\_data[1], tacho\_data[0])*

*c9c\_data = pressure.read\_data() if pressure.read\_data() < 3500 else print('corrupt packet')*

*if c9c\_data is not None:*

*db.add\_record(tacho\_data[2], 'c9c', c9c\_data, 'gram')*

*elif c9c\_data is None:*

*c9c\_data = 0*

*db.add\_record(tacho\_data[2], 'c9c', c9c\_data, 'gram')*

*output = """\rut372: {} {} | c9c: {} gram | time: {} """.format(tacho\_data[1], tacho\_data[0], c9c\_data, tacho\_data[2])*

*print(output, end='')*

**6.2 serial\_read.py**

*from serial import Serial, PARITY\_EVEN, SerialException*

*class we2107(object):*

*"""*

*class for pySerial c9c device, connected through we2107*

*"""*

*def \_\_init\_\_(self, comnumb):*

*self.ser = Serial()*

*self.ser.baudrate = 2400*

*self.ser.port = comnumb*

*self.ser.parity = PARITY\_EVEN*

*self.ser.stopbits = 1*

*self.ser.open()*

*def read\_data(self):*

*"""*

*Sends serial command to device and then reads the output data*

*:return: String with output data*

*"""*

*self.ser.write(b';S15;')*

*self.ser.write(b'MSV?;')*

*data = self.ser.readline()*

*return data[0]+data[1]\*255+data[2]*

*def \_\_del\_\_(self):*

*"""*

*convenient close of serial port*

*:return: None*

*"""*

*self.ser.close()*

*if \_\_name\_\_ == '\_\_main\_\_':*

*comnumb = input('Input COM port number: ')*

*try:*

*device = we2107(comnumb)*

*except SerialException:*

*print('Wrong COM port')*

*exit()*

*while True:*

*print(device.read\_data())*

**6.3 tacho\_access\_db.py**

*import pyodbc*

*from pyodbc import Error*

*class AccessConnect:*

*"""*

*main class for data manipulations in DB*

*"""*

*def \_\_init\_\_(self, database=r'C:\Users\Public\Documents\database.accdb', table=r'table'):*

*self.database = database*

*self.table = table*

*self.connstr = r'DRIVER={Microsoft Access Driver (\*.mdb, \*.accdb)};DBQ=' + self.database*

*self.cnxn = pyodbc.connect(self.connstr, autocommit=True)*

*self.cur = self.cnxn.cursor()*

*def \_\_del\_\_(self):*

*try:*

*self.cnxn.close()*

*except Error:*

*pass*

*def read\_record(self, sdate, devices):*

*"""*

*function that read a records for a specific date and time for a specific devices*

*:param sdate: datetime object. Date of the required data*

*:param devices: iteratable object of strings(list, set, dict, tuple, etc.). List of the devices*

*:return: list of tuples with data from DB*

*"""*

*strsql = "SELECT dev\_name,dev\_state,dev\_date FROM [{table}] where dev\_date > #{date}# " \*

*"and dev\_name IN ({device})".format(table=self.table, date=sdate, device=','.join(["'"+j+"'" for j in devices]))*

*self.cur.execute(strsql)*

*t = list(self.cur)*

*return t*

*def add\_record(self, sdate, device, value, data\_type):*

*"""*

*function that adds a record for a specific date and time for a specific device*

*:param sdate: datetime object. Date for the record*

*:param device: string object. Device name*

*:param value: float object. Data from devic*

*:param data\_type: string object. Dimension of the value*

*:return: None*

*"""*

*strsql = "insert into [{table}] (dev\_name, dev\_state, dev\_date, dev\_data\_type) values('{name}', {state}, " \*

*"#{sdate}#, '{data\_type}')".format(table=self.table, name=device, state=value, sdate=sdate, data\_type=data\_type)*

*self.cnxn.execute(strsql)*

*self.cnxn.commit()*

**6.4 tacho\_read.py**

*from time import sleep*

*import pywinusb.hid as hid*

*from datetime import datetime*

*class DeviceIsNotConnected(Exception):*

*"""*

*exception derived class. Meant to be raised when device is disconnected.*

*"""*

*pass*

*class ut372device(object):*

*"""*

*class for tacho ut372 device objects*

*"""*

*def \_\_init\_\_(self, vendor\_id=0x1a86, product\_id=0xe008):*

*self.vendor\_id = vendor\_id*

*self.product\_id = product\_id*

*self.lst = []*

*self.data = 0, 0, 0*

*self.device = 'will further be overriden with device object'*

*self.marker = False*

*self.descriptor = {*

*'7;': '0',*

*'60': '1',*

*'5>': '2',*

*'7<': '3',*

*'65': '4',*

*'3=': '5',*

*'3?': '6',*

*'70': '7',*

*'7?': '8',*

*'7=': '9',*

*'?;': '0.',*

*'>0': '1.',*

*'=>': '2.',*

*'?<': '3.',*

*'>5': '4.',*

*';=': '5.',*

*';?': '6.',*

*'?0': '7.',*

*'??': '8.',*

*'?=': '9.',*

*'00': '00',*

*'78': 'conf',*

*'86': 'bit',*

*}*

*self.connect()*

*def sample\_handler\_count(self, data):*

*"""*

*Device data handler. Discards garbage and collects a full package into a global list*

*:param data: Data object. Data from device.*

*:return: None*

*"""*

*if not chr(data[2]) == '\x00':*

*if self.marker:*

*self.lst.append(chr(data[2]))*

*if len(self.lst) == 27:*

*self.data = self.\_positioning\_count()*

*if chr(data[2]) == '\n':*

*self.marker = True*

*def \_positioning\_count(self):*

*"""*

*Private function that parses data and forms output*

*:return: String of output data*

*"""*

*deciphered\_raw = []*

*for i in range(1, len(self.lst) - 6, 2):*

*try:*

*deciphered\_raw.append(self.descriptor[self.lst[i] + self.lst[i + 1]])*

*except KeyError:*

*self.lst.clear()*

*self.marker = False*

*return*

*count = deciphered\_raw[:5]*

*try:*

*count = int(''.join(list(reversed(count))))*

*mes\_unit = 'count'*

*except ValueError:*

*count = float(''.join(list(reversed(count))))*

*mes\_unit = 'rpm'*

*time = datetime.now()*

*time = time.strftime('%d-%m-%Y %H:%M:%S')*

*output = mes\_unit, count, time*

*self.lst.clear()*

*return output*

*def connect(self):*

*"""*

*Function that connects to a device*

*:return: None*

*"""*

*devfilter = hid.HidDeviceFilter(vendor\_id=self.vendor\_id, product\_id=self.product\_id)*

*hid\_device = devfilter.get\_devices()*

*try:*

*self.device = hid\_device[0]*

*except IndexError:*

*raise DeviceIsNotConnected('Device is not connected!')*

*self.device.open()*

*self.device.set\_raw\_data\_handler(self.sample\_handler\_count)*

*self.\_send\_init\_packet()*

*def \_send\_init\_packet(self):*

*"""*

*Private function that send initialization package to a device*

*:return: None*

*"""*

*report = self.device.find\_feature\_reports()*

*buffer = [0x00] \* 6*

*buffer[0] = 0x00*

*buffer[1] = 0x60*

*buffer[2] = 0x09*

*buffer[5] = 0x03*

*report[0].set\_raw\_data(buffer)*

*report[0].send()*

*def receive\_package(self):*

*"""*

*Function that waits for the package received event and handles device disconnection*

*:return: None*

*"""*

*if self.device.is\_plugged():*

*sleep(0.02)*

*return self.data*

*else:*

*raise DeviceIsNotConnected('Device was unplugged')*

*if \_\_name\_\_ == '\_\_main\_\_':*

*try:*

*ourdevice = ut372device()*

*except DeviceIsNotConnected:*

*print('Device is not connected!')*

*print(ut372device.receive\_package())*