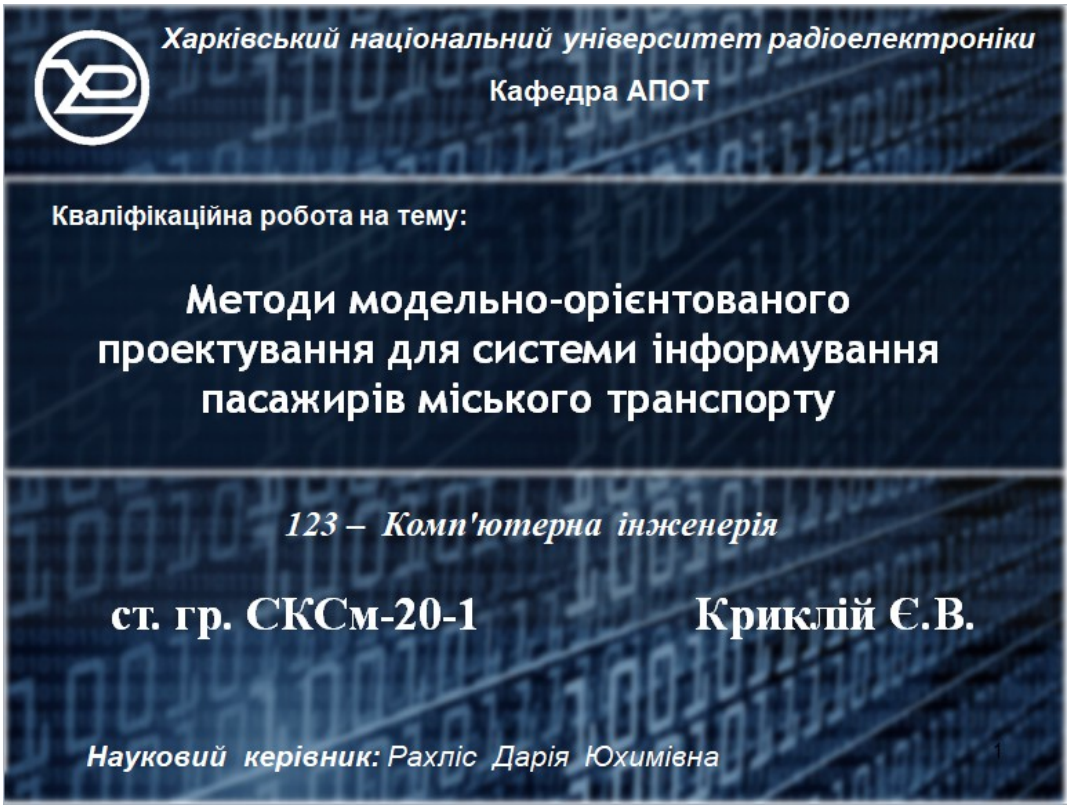
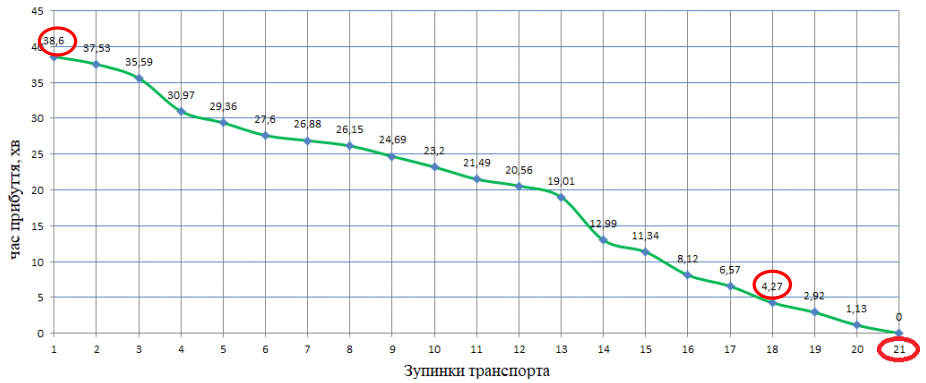
ДОДАТОК А.
Графічна частина атестаційної роботи



Результати моделювання СІП

Графік, побудований в результаті моделювання в системі Simulink, підтверджує загальну тенденцію: зменшення часу очікування транспорту при його наближенні до зупинки № 21.



ДОДАТОК Б

Лістинг Б.1 – GPS parser to parse GPGGA and GPRMC sentences received from GPS module

```
/* GPS parser to parse GPGGA and GPRMC sentences received from GPS module.
 * It is based on a generic parser that parses data from a Start character
 * to an End Character.
 * Message format considered here is:
 * StartChar|Data|CheckChar|Checksum|EndChar
 * The 'Data' field is separated by a parameter called 'separator'
 * The checksum is calculated for the Data field, i.e., for all characters
 * between Start Character and End Character*/

#include <Arduino.h>
#include "gps_parser.h"
#include <string.h>

uint8_T* dataBuffer = NULL;              // Declare the buffer

extern "C" void initialSetup(uint8_T bufferSize)
{
    dataBuffer = (uint8_T*)calloc(bufferSize,sizeof(uint8_T));
       // Allocate memory to the buffer

    if(NULL == dataBuffer){
// Check if memory is allocated to the buffer
        Serial.println("Failed to allocate memory to the buffer");
        printf("Memory not allocated");
        exit(0);
    }
}

extern "C" void releaseBuffer(){
    free(dataBuffer);               // Free the allocated memory
    dataBuffer = NULL;
}

void parseData(uint8_T *rawStream
        , uint32_T *time
        , float *latitude
        , float *longitude
        , uint8_T *noSat
        , double *hdop
        , double *alti
        , double *msl
        , double *speed
        , double *tckAngle
        , uint32_T *date
        , uint8_T bufferSize
        , uint8_T separator
        , int8_T *statusOut){

    uint8_T *commaPtr = NULL;                     // Pointer to present comma
    uint8_T *nextCommaPtr = NULL;                 // Pointer to next comma
    uint8_T *lastCommaPtr =  NULL;               // Pointer to last comma
```

```
    uint8_T *placeholder = NULL;                    // Temporary value holder
    int lengthField = 0;                           // Length of the temporary holder

    int isGPGGA = 0;                // Flag to check if the sentence is GPGGA
    int isGPRMC = 0;                // Flag to check if the sentence is GPRMC

    int commaCount = 0;            // Counter for the number of commas encountered

    *statusOut = 0;

    isGPGGA = strncmp(rawStream, "GPGGA", 5);    // Is sentence GPGGA?
    isGPRMC = strncmp(rawStream, "GPRMC", 5);   // Is sentence GPRMC?

    int state = -1;                    // State variable

    // State 1: GPGGA sentence
    // State 2: GPRMC sentence

    if (!isGPGGA)                 // GPGGA state
        state = 1;
    else if (!isGPRMC)            // GPRMC state
        state = 2;
    else
        state = -1;                                    // Default state

    if (!isGPGGA | !isGPRMC) {

        lastCommaPtr = rawStream;        // Store the pointer to the last comma
        while(lastCommaPtr != NULL){     // Continue until the end of the sentence
            commaPtr = strchr(lastCommaPtr + 1, separator);
// Pointer to the present comma (+1 to avoid the present comma being accounted for)
            nextCommaPtr = strchr(commaPtr + 1, separator);
// Pointer to the next comma
            lengthField = (nextCommaPtr - 1)- (commaPtr + 1) + 1;
// Length of the temporaty placeholder buffer

            placeholder = (uint8_T*)calloc(lengthField, sizeof(uint8_T));
// Allocate memory to the temporary buffer

            if(lengthField > 0 & NULL == placeholder){
 // Check if memory is allocated to the buffer
// The placeholder can be NULL if there are no characters between two commas

                *statusOut = -4;
                exit(0);
            }

            for(int i = 0; i < lengthField; i++){
 // Add the elements to the temporary placeholder buffer
                placeholder[i] = commaPtr[i+1];
            }
            placeholder[lengthField] = '\0';   // Append a null character at the end

            switch(state){
                case 1:
 //-------------------GPGGA state-----------------------
                    switch(commaCount){
// Assign the output elements depending on the number of commas read
                        case 0:
                            *time = (uint32_T)strtol(placeholder, NULL, 10); // TIME
                            break;
```

```
                        case 1:
                            *latitude = atof(placeholder);        // LATITUDE
                            lastCommaPtr = commaPtr;
                            commaPtr = strchr(lastCommaPtr + 1, separator);
 // Account for the direction of Latitude
// If 'S' then the value is negative
// If 'N' then positive
                            commaCount++;
                           if (*(commaPtr + 1) == 83 ){ // If the character is an 'S'
                                (*latitude) = (*latitude)*(-1);    // Multiply by -1
                            }
                            break;

                        case 3:
                            *longitude = atof(placeholder);         // LONGITUDE
                            lastCommaPtr = commaPtr;
                            commaPtr = strchr(lastCommaPtr + 1, separator);
// Account for the direction of the longitude
 // If 'W' the value is negative
 // If 'E' the value is positive
                            commaCount++;
                            if (*(commaPtr + 1) == 87 ){  // If the character is 'W'
                                (*longitude) = (*longitude)*(-1);   // Multiply by -1
                            }
                            break;

                        case 5:

                            if (*(commaPtr + 1) == '1'){
                                *statusOut = 1;
                            }
                            else{
                                *time = 0;
                                *latitude = 0;
                                *longitude = 0;
                            }
                            break;

                        case 6:
                            *noSat = (uint8_T)strtoul(placeholder, NULL, 10);
// NUMBER OF SATELLITES
                            break;

                        case 7:
                            *hdop = strtod(placeholder, NULL);
// HORIZONTAL DILUTION OF POSITION
                            break;

                        case 8:
                            *alti = strtod(placeholder, NULL);   // ALTITUDE
                            break;

                        case 10:
                            *msl = strtod(placeholder, NULL); // MEAN SEA LEVEL
                            break;

                        default:
                            break;
                    }
                    break;

                case 2:
```

```
//------------------GPRMC state-----------------------
                    switch(commaCount){
// Assign the output elements depending on the number of commas read
                        case 0:                                 // TIME
                            *time = (uint32_T)strtol(placeholder, NULL, 10);
                            break;

                        case 1:
                            if (*(commaPtr + 1) == 'A'){
                                *statusOut = 1;
                            }
                            break;

                        case 2:
                            *latitude = atof(placeholder);          // LATITUDE
                            lastCommaPtr = commaPtr;
                            commaPtr = strchr(lastCommaPtr + 1, separator);
// Account for the direction of Latitude
// If 'S' then the value is negative
// If 'N' then positive
                            commaCount++;
                            if (*(commaPtr + 1) == 83 ){   // If the character is 'S'
                                (*latitude) = (*latitude)*(-1);   // Multiply by -1
                            }
                            break;

                        case 4:
                            *longitude = atof(placeholder);       // LONGITUDE
                            lastCommaPtr = commaPtr;
                            commaPtr = strchr(lastCommaPtr + 1, separator);
// Account for the direction of Longitude
// If 'W' then the value is negative
// If 'E' then positive
                            commaCount++;
                            if (*(commaPtr + 1) == 87 ){   // If the character is 'W'
                                (*longitude) = (*longitude)*(-1);   // Multiply by -1
                            }
                            break;

                        case 6:
                            *speed = strtod(placeholder, NULL);        // SPEED
                            break;

                        case 7:
                            *tckAngle = strtod(placeholder, NULL);  // TRACK ANGLE
                            break;

                        case 8:
                            *date = (uint32_T)strtol(placeholder, NULL, 10);  // DATE
                            break;

                        default:
                            break;
                    }
                    break;
                default:
                    break;
            }
            free(placeholder);                              // Free the pointer
            placeholder = NULL;

            commaCount++;
```

```
            lastCommaPtr = commaPtr;
        }
    }
}

uint8_T fetchChecksum(uint8_T *pointToCheckChar, uint8_T endChar, int8_T *status){

    uint8_T *pointToEndChar = NULL;               // Location to the end character
    uint8_T *checksumString = NULL;              // Checksum ASCII values
    uint8_T checksum = 0;                       // Value to be returned
    uint8_T checksumLength = 0;            // Length between checkChar and endChar

    pointToEndChar = strchr(pointToCheckChar, endChar);

    checksumLength = (pointToEndChar - 1) - (pointToCheckChar + 1) + 1;
/* The checksum string starts one after the Check Character and ends
     * one character before the End Character.
     * Additional 1 for the '\0' character.*/

    checksumString = (uint8_T*)calloc(checksumLength+1, sizeof(uint8_T));  //
Allocate the memory to store checksum ASCII characters

    if (checksumString == NULL){                  // If memory not allocated
        *status = -3;                            // Indicate on the status output
        printf("Memory not allocated");       // Throw an error
        exit(0);                                 // Exit the program
    }

    for(int i = 0; i < checksumLength; i++){ // Loop over length of checksum received

        checksumString[i] = pointToCheckChar[i+1];
   /*  Append to the checksum array.
         * The +1 on RHS is because the characters after checkChar until
         * EndChar is the checksum value*/
    }

    checksumString[checksumLength] = '\0';
                // Append NULL charcater at the end of the array
    checksum = (uint8_T)strtoul(checksumString, NULL, 16);
// Convert the received ASCII checksum HEX value to a uint8

    free(checksumString);         // Free the allocated memory
    checksumString = NULL;
    return checksum;              // Return the calculated checksum
}

extern "C" void readData(uint8_T *inData
        , uint8_T dataLength
        , const uint8_T startChar
        , const uint8_T endChar
        , const uint8_T separator
        , const uint8_T checkChar
        , uint8_T *retString
        , const uint8_T bufferSize
        , int8_T *status){

    uint8_T presentChar = 0;                            // Present character
    uint8_T checksum = 0;                               // Checksum value
    uint8_T *pointToCheckChar = NULL;            // Pointer to the check character
    static bool storeData = false;        // Flag to start storing to an array
    static bool storeChecksum = false;   // Flag to start storing checksum
    static int index = 0;              // Index of the Data field
```

```
    static uint8_T parity = 0;         // Parity to be calculated

    *status = 0;                            // Reset status at every time step

    for(int i = 0; i < dataLength; i++){
// For each of the element of the string received

        presentChar = inData[i];
// Present character read from the input string

        if (index == bufferSize)          // If the buffer is overfull
        {
            memset(dataBuffer,0, bufferSize);      // Reset the buffer
            parity = 0;                           // Reset the parity
            index = 0;                     // Reset the index

            storeData = false;           // Disable storing Data Buffer
            storeChecksum = false;      // Disable storing checksum

            *status = -2;             // Indicate overfull on the status flag

            continue;                 // Go to waiting for start character state
        }

        if (!storeData){                // Waiting for start character

            if(presentChar == startChar){     // If start character is located

                storeData = true;
// Enable flag to store the values in the Data Buffer
                memset(dataBuffer, 0, bufferSize);        // Reset the buffer
                parity = 0;                               // Reset the parity
                continue;               // Go to the next character in the for loop
            }
        }
        else{
            /* If another start character is received
             * (as an unexpected behaviour)*/
            if(presentChar == startChar){

                // Reset the buffer
                memset(dataBuffer,0, bufferSize);
                // Reset the parity
                parity = 0;
                // Reset the index
                index = 0;
                // Continue to the next character in the for loop
                continue;
            }

            /* If the character is the check character, indicate that the
             * next characters will be checksum values*/
            else if(presentChar == checkChar){

                // Flag to indicate checksum is to be read
                storeChecksum = true;
            }

            /* If the character is the end character then:
             * 1. Stop storing in the Data Buffer
             * 2. Fetch the checksum value from the string
             * 3. Reset the Data buffer and Flags*/
```

```
            else if(presentChar == endChar){

                // Append to the Data Buffer
                dataBuffer[index] = presentChar;
                // Pointer to the check character
                pointToCheckChar = strchr(dataBuffer, checkChar);

                // Fetch the checksum value received from the sentence
                checksum = fetchChecksum(pointToCheckChar, endChar, *status);

                // If calculated parity and checksum matches
                if (parity == checksum){

                    // Reset the string to be returned
                    memset(retString, 0, bufferSize);

                    /* Copy the characters between startChar and checkChar
                     * to the output */
                    strncpy(retString, dataBuffer,
                            (pointToCheckChar - dataBuffer));
                    retString[(pointToCheckChar - dataBuffer) +1] = '\0';

                    *status = 1;

                }
                else{

                    // Reset the output
                    memset(retString, 0, bufferSize);
                }

                // Reset the buffer
                memset(dataBuffer, 0, bufferSize);
                // Reset the parity
                parity = 0;
                // Reset the index
                index = 0;

                // Disable storing of the Data Buffer
                storeData = false;
                // Disable storing checksum
                storeChecksum = false;

                continue;
            }

            /* If the character is between startChar and checkChar
             * calculate the parity*/
            if (!storeChecksum){
                parity = parity^presentChar;
            }

            /* Append the present character to the DataBuffer
             * Appends the characters between startChar and endChar
             * excluding the two*/
            dataBuffer[index++] = presentChar;
        }
    }
}
```

```
#ifndef _GPS_PARSER_H_
#define _GPS_PARSER_H_
#include "rtwtypes.h"

#ifdef __cplusplus
extern "C" {
#endif

    // Read the data from the input port and process
    void readData(uint8_T* buffer
            , uint8_T dataLength
            , const uint8_T startChar
            , const uint8_T endChar
            , const uint8_T separator
            , const uint8_T checkChar
            , uint8_T *yout
            , const uint8_T bufferSize
            , int8_T *status);

    // Allocate the buffer
    void initialSetup(uint8_T bufferSize);

    // Free the buffer
    void releaseBuffer();
    // Find the checksum from the NMEA string
    uint8_T fetchChecksum(uint8_T *pointToCheckChar, uint8_T endChar, int8_T
*status);

    void parseData(uint8_T *rawStream
            , uint32_T *time
            , float *latitude
            , float *longitude
            , uint8_T *noSat
            , double *hdop
            , double *alti
            , double *msl
            , double *speed
            , double *tckAngle
            , uint32_T *date
            , uint8_T bufferSize
            , uint8_T separator
            , int8_T *status);

#ifdef __cplusplus
}
#endif
#endif //_GPS_PARSER_H_
```

Лістинг Б.2 – Listing for sender the data from GPS module to IoT ThingSpeaks

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include "ThingSpeak.h"
#include <ESP8266WiFi.h>

static const int RXPin = 12, TXPin = 13;
static const uint32_t GPSBaud = 9600;
```

```
// repace your wifi username and password
const char* ssid     = "Ehor";
const char* password = "*********";
unsigned long myChannelNumber = 1281267;
const char * myWriteAPIKey = "AME6XG0SJ7LJU0SA";

// The TinyGPS++ object
TinyGPSPlus gps;
WiFiClient  client;

// The serial connection to the GPS device
SoftwareSerial ss(RXPin, TXPin);

void setup()
{
  Serial.begin(115200);
  ss.begin(GPSBaud);
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  { delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  Serial.print("Netmask: ");
  Serial.println(WiFi.subnetMask());
  Serial.print("Gateway: ");
  Serial.println(WiFi.gatewayIP());
  ThingSpeak.begin(client);
  }
void loop()
{
   // This sketch displays information every time a new sentence is correctly
encoded.
  while (ss.available() > 0)
    if (gps.encode(ss.read()))
      displayInfo();
  if (millis() > 5000 && gps.charsProcessed() < 10)
  {
    Serial.println(F("No GPS detected: check wiring."));
    while(true);
  }
}
void displayInfo()
{
 // Serial.print(F("Location: "));
  if (gps.location.isValid())
  {
    double latitude = (gps.location.lat());
    double longitude = (gps.location.lng());
    String latbuf;
    latbuf += (String(latitude, 6));
    Serial.println(latbuf);
    String lonbuf;
    lonbuf += (String(longitude, 6));
    Serial.println(lonbuf);
    ThingSpeak.setField(1, latbuf);
    ThingSpeak.setField(2, lonbuf);
    ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
```

```
    delay(20000);
  }
  else
  {
    Serial.print(F("INVALID"));
  }
  Serial.print(F("  Date/Time: "));
  if (gps.date.isValid())
  {
    Serial.print(gps.date.month());
    Serial.print(F("/"));
    Serial.print(gps.date.day());
    Serial.print(F("/"));
    Serial.print(gps.date.year());
  }
  else
  {
    Serial.print(F("INVALID"));
  }

  Serial.print(F(" "));
  if (gps.time.isValid())
  {
    if (gps.time.hour() < 10) Serial.print(F("0"));
    Serial.print(gps.time.hour());
    Serial.print(F(":"));
    if (gps.time.minute() < 10) Serial.print(F("0"));
    Serial.print(gps.time.minute());
    Serial.print(F(":"));
    if (gps.time.second() < 10) Serial.print(F("0"));
    Serial.print(gps.time.second());
    Serial.print(F("."));
    if (gps.time.centisecond() < 10) Serial.print(F("0"));
    Serial.print(gps.time.centisecond());
  }
  else
  {
    Serial.print(F("INVALID"));
  }

  Serial.println();
}
```

Лістинг Б.3 – Listing for receiver of the data from IoT ThingSpeaks to LCD display of the Arduino UNO through Ethernet

```
#include <Ethernet.h>
#include <LiquidCrystal.h>
#include "ThingSpeak.h"
#define SECRET_MAC {0xDA, 0xDB, 0xDC, 0xDD, 0xDE, 0xDF}
byte mac[] = SECRET_MAC;
// with the arduino pin number it is connected to
const int rs = 2, en = 3, d4 = 4, d5 = 5, d6 = 6, d7 = 7;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
// Set the static IP address to use if the DHCP fails to assign
IPAddress ip(192, 168, 88, 239);
EthernetClient client;
//---------Channel Details---------//
unsigned long counterChannelNumber = 1282832;            // Channel ID
const char * myCounterReadAPIKey = "IMHMPHI1XS062OMS"; // Read API Key
```

```
const int FieldNumber1 = 1;  // The field you wish to read
const int FieldNumber2 = 2;  // The field you wish to read
//------------------------------//
//function to convert data to degree//
float toRadians(const float degree)
{
    float one_deg = (M_PI) / 180;
    return (one_deg * degree);
}
 //distance using Haversine Formula
float  distance(float  Latitude,  float  Longitude,  float  stop_lat,  float
stop_long)
{
    // Convert the latitudes and longitudes from degree to radians
    Latitude = toRadians(Latitude);
    Longitude = toRadians(Longitude);
    stop_lat = toRadians(stop_lat);
    stop_long = toRadians(stop_long);
      // Haversine Formula
    float dlong = stop_long - Longitude;
    float dlat = stop_lat - Latitude;
        float ans = pow(sin(dlat / 2), 2) + cos(Latitude) * cos(stop_lat) *
pow(sin(dlong / 2), 2);
    ans = 2 * asin(sqrt(ans));
      // Radius of Earth in Kilometers, R = 6371
    // Use R = 3956 for miles
    float R = 6371;
    // Calculate the result
    ans = ans * R;
    return ans;
}
void setup()
{ lcd.begin(16, 2);
  lcd.print("Arriving time:");
  lcd.setCursor (0,1);
  lcd.blink();
  delay(2000);
  Ethernet.init(10);  // Most Arduino Ethernet hardware
  Serial.begin(115200);  //Initialize serial
   // start the Ethernet connection:
  Serial.println("Initialize Ethernet with DHCP:");
  if (Ethernet.begin(mac) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // Check for Ethernet hardware present
    if (Ethernet.hardwareStatus() == EthernetNoHardware) {
          Serial.println("Ethernet  shield  was  not  found.   Sorry,  can't  run
without hardware. :(");
      while (true) {
        delay(1); // do nothing, no point running without Ethernet hardware
      }
    }
    if (Ethernet.linkStatus() == LinkOFF) {
      Serial.println("Ethernet cable is not connected.");
    }
  } else {
    Serial.print("  DHCP assigned IP ");
    Serial.println(Ethernet.localIP());
  }
  // give the Ethernet shield a second to initialize:
  delay(1000);
    ThingSpeak.begin(client);  // Initialize ThingSpeak
}
void loop()
{
```

```
  int statusCode = 0;
      float   Longitude   =   ThingSpeak.readFloatField(counterChannelNumber,
FieldNumber1, myCounterReadAPIKey);
  // Check the status of the read operation to see if it was successful
  statusCode = ThingSpeak.getLastReadStatus();
  if(statusCode == 200){
     Serial.print("Longitude Track: ");
    Serial.println(Longitude);
  }
  else{Serial.println("Problem reading channel" );}
  delay(1000);
float     Latitude     =     ThingSpeak.readFloatField(counterChannelNumber,
FieldNumber2, myCounterReadAPIKey);
   // Check the status of the read operation to see if it was successful
  statusCode = ThingSpeak.getLastReadStatus();
  if(statusCode == 200){
    Serial.print("Latitude: ");
    Serial.println(Latitude);
  }
  else{ Serial.println("Problem reading channel" ); }
 delay(1000);
 //main//
float stop_long = 49.99614;
float stop_lat = 36.33915;
float S=distance(Latitude, Longitude, stop_lat, stop_long);
float T = ((S/15)*60);//minutes
if (statusCode == 200)
  { Serial.print("Distance: ");
    Serial.println(S);
    Serial.print("Arriving time: ");
    Serial.println(T);
   }
  else
  {  Serial.println("Unable to read channel / No internet connection");
  }
   delay (1000);
// lcd.clear ();
lcd.setCursor (0,1);
lcd.print(T);
lcd.setCursor (8,1);
lcd.print("minutes");
lcd.noBlink();
}
```

ДОДАТОК В.
Апробація результатів дослідження

МОДЕЛЬНО-ОРІЄНТОВАНЕ ПРОЕКТУВАННЯ СИСТЕМИ ІНФОРМУВАННЯ ПАСАЖИРІВ МІСЬКОГО ТРАНСПОРТУ

Криклій Є.В.
Науковий керівник – доц. Рахліс Д.Ю.
Харківський національний університет радіоелектроніки
(61166, Харків, пр. Науки, 14, каф. АПОТ, тел. (057) 702-13-26)
e-mail: yehor.kryklii@nure.ua, тел. +38(099)128-59-41

The main advantage of model-based design is to enable faster, more cost-effective development of dynamic systems, including control systems, signal processing, and communications systems. The model of the informational system for passengers of the public transport is proposed. The approach of model-oriented design was used to verify the correctness of the system which is operating in real time.

Вступ. На сьогоднішній день цифрові системи настільки складні, що традиційний процес їх перевірки вже мало придатний. Коли на пізніх стадіях проекту виявляються проблеми, то зазвичай для їх вирішення потрібні складні та витратні за часом зміни проекту, які призводять до дорогої модифікації апаратних засобів. Крім того складність алгоритмів не дає змогу перевірити його працездатність на всіх робочих режимах. Тому і необхідні нові методи, до яких і відноситься модельно-орієнтоване проектування (МОП) [1]. Цей підхід дає змогу перевірити коректність роботи системи на ранніх стадіях, коли ще можливо виправити помилки швидко та з мінімальними затратами.

Об'єктом дослідження є системи інформування пасажирів (СІП) громадського транспорту. Такі система працюють в реальному режимі часу та можуть мати різноманітні функції [2]. На сьогодні у світі існує багато прикладів реалізації таких систем інформування. Але головним недоліком таких систем є той факт, що вони запрограмовані на визначений набір функцій, який, найчастіше, не має цінності для пасажирів. Надлишкова інформація навпаки відволікає людину від головного, особливо, коли мова йде про людей похилого віку чи гостей міста. Також одним з недоліків готових рішень є те, що виробники даних пристрої зберігають в таємниці алгоритми, за допомогою яких працює їх пристрої, що ускладнює їх модернізацію та модифікацію. Для того, щоб позбутися цих недоліків було вирішено запропонувати пристрій, що може бути запрограмований за допомогою відкритого середовища розробки.

Зміст дослідження. Підхід МОП заснований на застосуванні концепції «in-the-loop testing» має декілька варіантів в залежності від того, що виступає в якості об'єкта тестування. Якщо в якості об'єкта тестування виступає модель, то такий підхід носить назву «model-in-the-loop» (MiL), якщо програмне забезпечення − «software-in-the-loop». (SiL), прототип пристрою –

«processor-in-the-loop» (PiL), а якщо готовий пристрій – «hardware-in-the-loop» (HiL) [4]. При МОП систем реального часу, до яких відноситься система інформування пасажирів громадського транспорту, ядром розробки є програмна модель. Така модель дозволить тестувати і досліджувати алгоритм функціонування, реалізований в пристрої, в реальному часі.

Модель запропонованої СІП базується на мікроконтролері Arduino, а значить може бути перепрограмована в залежності від локації та потреб тієї чи іншої зупинки громадського транспорту. Принцип роботи системи полягає у тому, що на борт громадського транспорту встановлюється обладнання з GPS-приймачем. Цей приймач оброблює сигнали зі супутникових навігаційних систем. Данні про координати транспорту передаються до блока управління, який обчислює час його прибуття. Ця інформація виводиться на екран, що встановлений на зупинці громадського транспорту.

Для розробки, налагодження і тестування програмного забезпечення системи інформування пасажирів громадського транспорту застосовують такі засоби, як програмні та фізичні імітатори сигналів та інтерфейсів.

Тому, *мета дослідження* – розробка моделі системи інформування пасажирів громадського транспорту згідно з парадигмами модельно-орієнтованого проектування.

Висновки. Система інформування пасажирів призначена для підвищення якості транспортного обслуговування населення на основі оперативного інформування пасажирів про роботу громадського транспорту. *Наукова новизна* визначається використанням підходу модельно-орієнтованого проектування СІП громадського транспорту. Це дає змогу перевірити працездатність системи не на реальному об’єкті, а імітувати навколишнє середовище системи. Застосування підходу МОП забезпечує підвищення надійності створюваних програмних, апаратних та програмно-апаратних систем.

Список джерел:

1. Jones D. Model-Based Design of control systems: Simulate and test before committing to hardware / Doug Jones, Brian McKay // Industrial embedded systems. – Режим доступу: http://industrial.embedded-computing.com/articles/model-based-design-control-systems-simulate-test-committing-hardware/. – Дата доступу: 24.12.2020. – Загол. з екр.

2. Esfeh Ansari M. Waiting time and headway modelling for urban transit systems – a critical review and proposed approach / Mohammad Ansari Esfeh, S. C. Wirasinghe, Saeid Saidi, Lina Kattan // Transport Reviews. – 2020. – P. 1-23.

3. Журавлев С. С. Применение модельно-ориентированного проектирования к созданию АСУ ТП опасных промышленных объектов / С. С. Журавлев, С. В. Рудометов, В. В. Окольнишников, С. Р. Шакиров // Вестн. НГУ. Серия: Информационные технологии. − 2018. − Т. 16, № 4.− С. 56-67.

Нормоконтроль пройдено 21.12.21

 */Рахліс Д.Ю.*