Technical University of Crete

Production Engineering & Management Dept.

Study and development of an interactive information platform for a prototype hybrid hydrogen-electric vehicle

Anastasios K. Petrou

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Production Engineering.

Department of Production Engineering & Management.

Laboratory of Intelligent Systems & Robotics IS&RL

CHANIA 2012

This page was intentionally left blank.

Study and development of an interactive information platform for a prototype hybrid hydrogen-electric vehicle

Master Thesis of

Anastasios K. Petrou

apetrou@isc.tuc.gr, petroutassos@gmail.com
DPEM, Technical University of Crete, Greece

Supervising Committee

Master's Advisor

Dr. Nikolaos Tsourveloudis
nikost@dpem.tuc.gr
Professor, Intelligent Systems & Robotics Lab, Machine Tools Lab.
DPEM, Technical University of Crete, Greece

Dr. Ioannis Nikolos
jnikolo@dpem.tuc.gr
Assistant Professor, Intelligent Systems & Robotics Lab.
DPEM, Technical University of Crete, Greece

Dr. Efstratios Ioannidis
efioan@dpem.tuc.gr
Assistant Professor, CAM Lab.
DPEM, Technical University of Crete, Greece

Abstract

In this thesis an interactive multi-threading driver information system is presented. The software was entirely developed using VB.Net and C programming language. The application was initially developed to support the prototype urban vehicle TUC Eco Racer and its participation in the international competition Shell Eco Marathon.

The developed system includes software interface and real time communication with 1) a hydrogen fuel cell module and 2) a microcontroller equipped with a number of sensors used for data acquisition and management. ,These data/sensor readings include: 1) Hydrogen temperature and ambient temperature, 2) Voltage and current produced by the fuel cells, 3) pressure, concentration of hydrogen and fuel consumption, 4) Concentration of oxygen, 5) Instantaneous velocity and acceleration of the vehicle, and 6) Inclination of the track. The above information is recorded and displayed in a graphical user interface, enabling the driver to access and control various functionalities and subsystems through a touch screen. At the same time a vehicle tracking functionality is provided. Further, a sound warning system for the pedestrians and the cyclists has developed.

This page was intentionally left blank

Μελέτη και ανάπτυξη συστήματος διαδραστικής πληροφόρησης για πρωτότυπο υβριδικό όχημα υδρογόνου-ηλεκτρισμού

Μεταπτυχιακή Διατριβή του

Πέτρου Κ. Αναστάσιου

apetrou@isc.tuc.gr, petroutassos@gmail.com
ΜΠΔ, Πολυτεχνείο Κρήτης, Ελλάδα

Εξεταστική Επιτροπή

Επιβλέπων

Δρ. Νικόλαος Τσουρβελούδης
nikost@dpem.tuc.gr
Καθηγητής, Εργαστήριο Ευφυών Συστημάτων & Ρομποτικής,
Διατμηματικό Εργαστήριο Εργαλειομηχανών.
ΜΠΔ, Πολυτεχνείο Κρήτης, Ελλάδα

Δρ. Ιωάννης Νικολός
jnikolo@dpem.tuc.gr
Επίκουρος Καθηγητής, Εργαστήριο Ευφυών Συστημάτων & Ρομποτικής.
ΜΠΔ, Πολυτεχνείο Κρήτης, Ελλάδα

Δρ. Ευστράτιος Ιωαννίδης
efioan@dpem.tuc.gr
Επίκουρος Καθηγητής, Εργαστήριο Βιομηχανικής Παραγωγής με τη Βοήθεια Η/Υ (CAM).
ΜΠΔ, Πολυτεχνείο Κρήτης, Ελλάδα

Περίληψη

Στην εργασία προτείνεται πολυνηματικό διαδραστικό σύστημα πληροφόρησης οδηγού εξ' ολοκλήρου ανεπτυγμένο με χρήση γλώσσας VB.Net και C. Η εφαρμογή αναπτύχθηκε για την υποστήριξη του πρωτότυπου ερευνητικού οχήματος πόλης TUC Eco Racer και τη συμμετοχή του στο διεθνή διαγωνισμό οικονομίας καυσίμου Shell Eco Marathon (http://www.tucer.tuc.gr/).

Το σύστημα που αναπτύχθηκε περιλαμβάνει λογισμικό διασύνδεσης και επικοινωνίας σε πραγματικό χρόνο, των κυψελών καυσίμου υδρογόνου με μικροελεγκτή και αισθητήρες, για την απόκτηση και διαχείριση πληροφορίας, όπως: 1) Θερμοκρασία κυψελών υδρογόνου και θερμοκρασία περιβάλλοντος, 2) Τάση και ένταση ρεύματος που παράγει το σύστημα κυψελών, 3) Πίεση, συγκέντρωση και κατανάλωση καυσίμου υδρογόνου, 4) Συγκέντρωση οξυγόνου, 5) Στιγμιαία ταχύτητα και επιτάχυνση του οχήματος, καθώς και την 6) Κλίση του οδοστρώματος. Οι ανωτέρω πληροφορίες καταγράφονται και απεικονίζονται σε ειδικά σχεδιασμένα παράθυρα επικοινωνίας για την άμεση ενημέρωση του χρήστη-οδηγού παρέχοντας τη δυνατότητα ενεργοποίησης ή απενεργοποίησης των επιμέρους λειτουργιών και υποσυστημάτων στον οδηγό μέσω οθόνης αφής. Παράλληλα παρέχεται η δυνατότητα εντοπισμού και απεικόνισης της θέσης του οχήματος, καθώς επίσης, έχει αναπτυχθεί και διασυνδεθεί σύστημα ηχητικής ειδοποίησης που στοχεύει στο να γίνει αντιληπτή η διέλευση του οχήματος απο τους πεζούς και τους δικυκλιστές.

To my parents Kostas, Pagona

To Lila

This page was intentionally left blank.

Contents

Award

Part of this work and especially the Pedestrian Sound Warning Functionality helped TUCer Team to win the first safety award at the Shell Eco Marathon 2011 that conducted in the EuroSpeedway Lausitz track in Germany.

OFF-TRACK AWARD

ADAC Safety Award

28/05/2011

This went to the Technical University of Crete for the second year running.

Not only were judges impressed with the way the team's workshop was set up, but their vehicle met, and in some aspects exceeded, safety requirements. "They have used their experiences from last year and developed a unique concept in pedestrian warning systems," commented one of the judges. "The whole team ethos is about developing and exhibiting safety." Very close runners-up were the Warsaw University of Technology and the Technical University of Sofia.

Figure a: Snapshot of the Shell Eco Marathon site for the ADAC Safety Award 2011.

Figure b: TUCer team celebrating the winning of the ADAC Safety Award with ER11 vehicle.

Declaration

The work in this thesis is original and no portion of the work referred to here has been submitted in support of an application for another degree or qualification of this or any other university or institution of learning.

Signed: Date:

Anastasios K. Petrou

Acknowledgments

Ας μου επιτραπεί το κομμάτι των ευχαριστιών, σε αντίθεση με την υπόλοιπη εργασία, να γραφθεί στα Ελληνικά, στο πρώτο ενικό και σε πιο ‘χαλαρή’ γλώσσα.

Αρχικά λοιπόν, θα ήθελα να εκφράσω τις ιδιαίτερες ευχαριστίες μου στον κ. Νικόλαο Τσουρβελούδη, τον επιβλέποντα καθηγητή, που με τις γνώσεις του, τις συμβουλές του αλλά πάνω από όλα με τη θέληση και την πάντα ευχάριστη διάθεσή του βοήθησε ώστε να έχουμε μία άψογη συνεργασία και όλη η εργασία να κυλήσει ομαλά και σε εύλογο χρονικό διάστημα.

Τους επίκουρους καθηγητές και μέλη της εξεταστικής επιτροπής κ. Ιωάννη Νικολό και κ. Ευστράτιο Ιωαννίδη για την άψογη συνεννόηση και συνεργασία.

Τον κ. Σάββα Πιπερίδη, ΕΤΕΠ του Εργαστηρίου Ευφυών Συστημάτων & Ρομποτικής για την άψογη συνεργασία μας τα δύο χρόνια που ήμουν μέλος της ερευνητικής ομάδας TUCER. Η συμβολή του ήταν καθοριστικής σημασίας ειδικά σε περιπτώσεις όπου όλα ‘πήγαιναν στραβά’.

Τον κ. Πολυχρόνη Σπανουδάκη, υποψήφιο διδάκτωρα του τμήματος Μηχανικών Παραγωγής & Διοίκησης και επικεφαλή της ερευνητικής ομάδας TUCER για τη φανταστική ευκαιρία που μου έδωσε και την εμπιστοσύνη που μου έδειξε ώστε να αποτελέσω μέλος της ομάδας για δύο χρόνια, 2010 και 2011.

Τον κ. Ιωάννη Τσινάρη, εργαστηριακό συνεργάτη του Διατμηματικού Εργαστηρίου Εργαλειομηχανών και μέλος της ομάδας TUCER για την άψογη συνεργασία.

Το ‘Συνάδελφο’! και φίλο κ. Δημήτριο Ευσταθίου για την άψογη συνεργασία σε όλη τη διάρκεια του μεταπτυχιακού, για τις άκρως επιστημονικές συζητήσεις μας και για όλα αυτά τα βράδια που ξενυχτήσαμε στο εργαστήριο και στο γραφείο δουλεύοντας.

Τους φίλους και μέλη της ομάδας TUCER, κ. Θάνο Τζανάκη και κ. Ιωάννη Στρατηγό για την άψογη συνεργασία και παρέα αυτά τα δύο χρόνια.

Πάνω από όλα θα ήθελα να ευχαριστήσω τους γονείς μου, Κώστα και Παγώνα για τη συμπαράσταση, τη στήριξη και την ενθάρρυνση τους όλο αυτό το διάστημα. Τέλος θα ήθελα να ευχαριστήσω τη Λίλα Τακμάκη για τη ψυχολογική υποστήριξη και την ατελείωτη υπομονή της σε όλο αυτό το διάστημα μέχρι την ολοκλήρωση του μεταπτυχιακού διπλώματος.

Ευχαριστώ!

This page was intentionally left blank.

CHAPTER 1

The Platform

1.1 The concept

The idea of developing the Interactive Information Platform was born in the Intelligent Systems & Robotics Laboratory of Technical University of Crete for the support of the TUC Eco Racer 11 (ER11) and 12 (ER12) prototype urban vehicles (http://www.tucer.tuc.gr/) and their participation in the annual European competition Shell Eco Marathon (http://www.shell.com/home/content/ecomarathon/). The ER11 and ER12 (Figure 1) vehicles are prototypes hybrid hydrogen-electric vehicles and they are an achievement of the TUC Eco Racing team which designs and constructs prototype, low consumption urban vehicles since 2008.

The concept was simple and very interesting (Figure 2): The development of a system that will provide information to the driver and allow him to interact with the hardware devices and the whole system in real time. For the achievement of this goal there was the need to equip the ER11 and ER12 with a variety of several hardware devices. And for the fitting interaction and functionality of all of these hardware components a custom software application developed from scratch.

This chapter presents and describes in detail the platform, it's functionality, the hardware components consists of and the custom developed software.

Figure 1.1: The ER12 prototype urban vehicle.

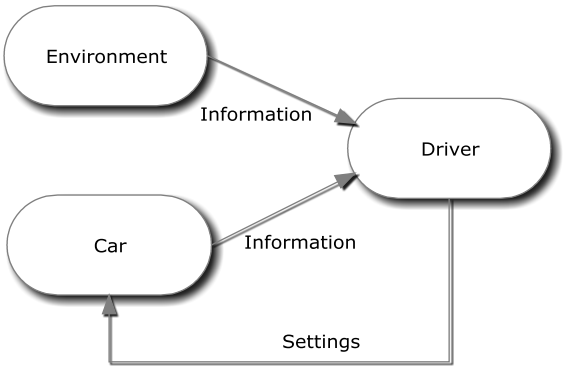

Figure 1.2: The concept of the Interactive Information Platform.

1.2 The hardware

The interactive information platform consists of several hardware components. These are:

- ***A Nexa™ Power Module from Ballard Company.***
  The Nexa™ power module is a small, low maintenance and fully automated fuel cell system designed to be integrated into products for portable and back-up power markets. The Nexa™ system provides up to 1200 watts of unregulated DC power at a nominal output voltage of 26 VDC. Using hydrogen fuel, the Nexa™ module is extremely quiet and produces zero harmful emissions.

- ***A BSZ-PG 1200 DC-DC converter.***

  An especially designed, for the Nexa Fuel Cell system, DC-DC converter the "BSZ-PG 1200" developed by the ISLE Company used. The "BSZ-PG 1200" has to handle the battery-management and to control the fuel cell system and complies with requirements such as: High efficiency also at light load, protection of battery and fuel cell, microprocessor controlled operation for optimum use of battery and fuel cell, fully automatic operation, compact design, lightweight construction etc.

- ***A VIA Epia-P830 Pico-ITX embedded board.***
  Single board computer with VIA Nano 1.2 GHz processor equipped with a 2GB SO-DIMM DDR 3 and with a 16GB solid state drive.

- ***An Arduino microcontroller.***
  Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software.

- ***An AccuStar® Electronic Clinometer module.***

- ***An ADXL203 dual-axis accelerometer from Analog Devices.***

- ***A reed switch.***

- ***A 7 inches VGA TFT USB touch screen.***
- ***A pair of USB speakers.***

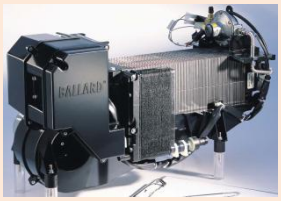	Nexa™ Power Module
	BSZ-PG 1200 DC-DC converter
	VIA Epia-P830 Pico-ITX embedded board
	Arduino microcontroller
	AccuStar® Electronic Clinometer
	ADXL203 dual-axis accelerometer
	Reed switch
	7 inches VGA TFT USB touch screen
	USB speakers

Table 1.1: The hardware devices used in the Interactive Information Platform.

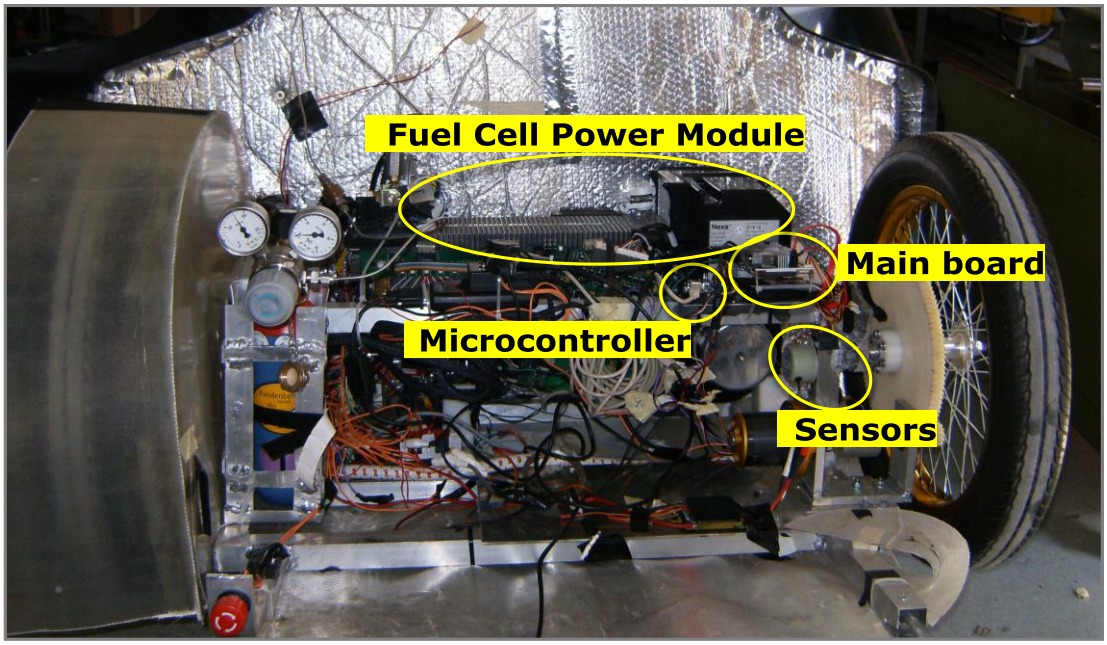

Figure 1.3: The set up of the hardware devices.

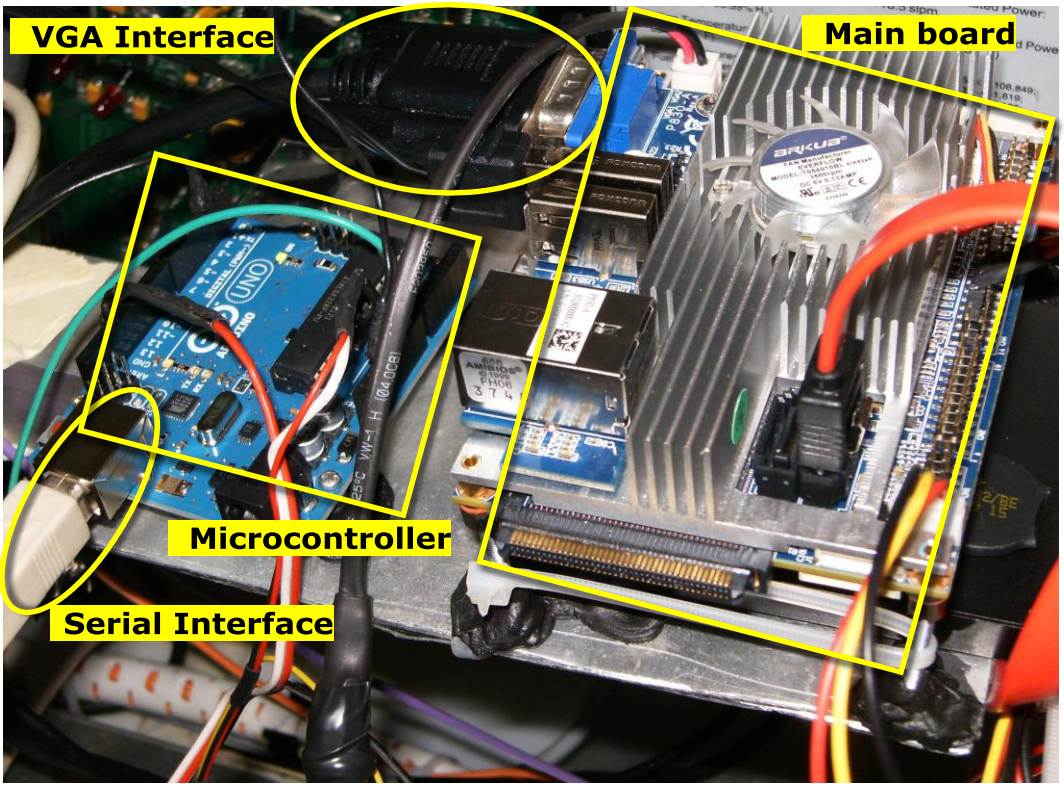

Figure 1.4: The computer unit and the microcontroller.

1.3 Operating principle and functionality

The operating principle of the platform is as follows (Figure 1.5): The software application executed on Epia board communicates with the Nexa power module and with the arduino microcontroller. Nexa power module provides information like fuel cell's temperature, voltage, current and watt, hydrogen's cumulative consumption and pressure and purge cell voltage. The arduino microcontroller is responsible for the sensors data acquisition and provides the vehicle's velocity (reed switch), vehicle's horizontal acceleration (ADXL203 dual-axis accelerometer) and track's inclination (AccuStar electronic clinometer). The user has the ability to interfere, to activate or deactivate any of the supported functionalities the interactive information platform provides.

The communication between the hardware devices and the software is bidirectional and an asynchronous protocol is used (see Appendix A for a brief introduction to serial communication). That means that the interface doesn't include a clock line. Instead, each device provides its own clock to use as a timing reference.

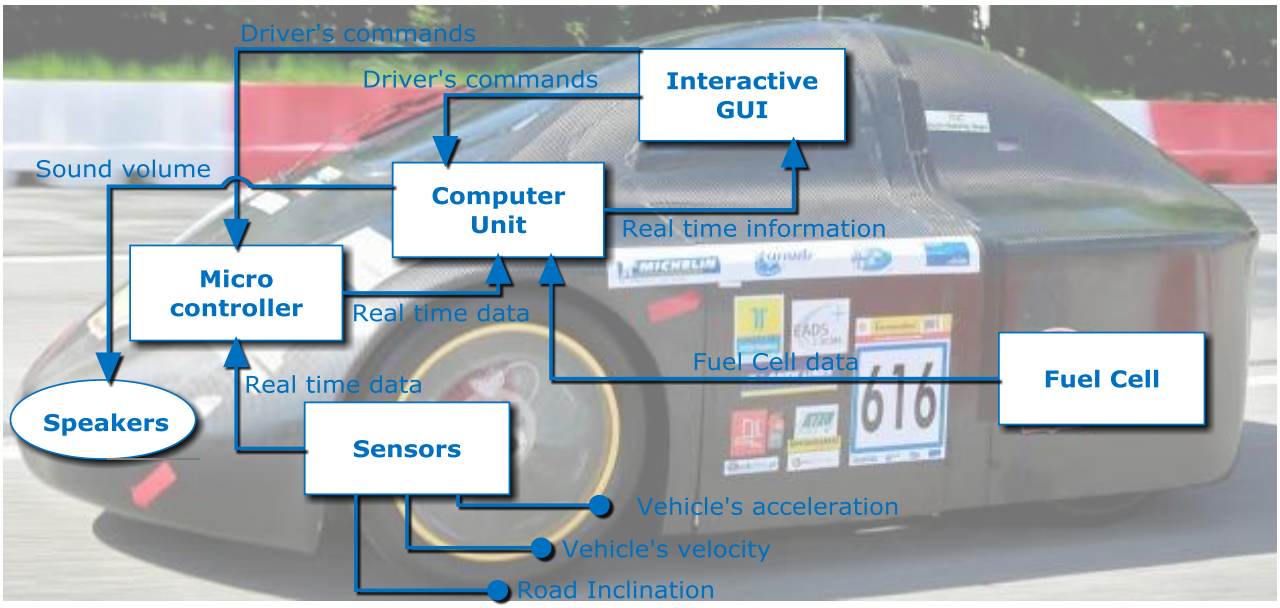

Figure 1.5: Flows of the Interactive Information Platform.

Communication with the microcontroller is achieved using commands in a string format but communication with the Fuel Cell is achieved by receiving bytes in a hexadecimal format.

The functionality that the platform provides comprises:

- Visualization functionality of the vital information that retrieved from the FC module and the microcontroller.
- Data recording functionality of the retrieved data for later study and processing.
- Sound warning functionality for the warning of the pedestrians and the cyclists.

- Vehicle tracking functionality for the visualization of the vehicles position on the track during the race.
- Support for an energy management functionality.
- The driver has the ability to activate or deactivate any of the functionalities described above and he is able to interact with the platform.

1.4 The Software

The software was especially studied and developed as a part of the interactive information platform described above. It was developed from scratch in VB.Net which means it is a Windows OS native application. It provides a friendly and functional graphical user interface (GUI) to allow the user constantly being informed about system's status and allow him to interact with the hardware devices and control the whole system in general.

The user has the ability to activate or deactivate any of the supported functionalities, to set or reset wanted variables (such as fuel volume, timer etc.), to load extra track-maps and information files for the Vehicle Tracking functionality using the system's settings, to load a custom sound file for the Pedestrian Sound Warning functionality, to select the location where the data will be recorded, to make the necessary serial port settings and several other options.

Due to the need the graphical user interface has to remain responsive, the need of multiple calculations and the simultaneous communication with two different hardware devices the software developed is a multithreading application. And because of the interactivity the application offers, this software is also an event-driven application.

When the software launches a welcome window appears and it provides a brief information text about the interactive information platform (Figure 1.6). The user is being guided through the necessary steps to make the required settings for the hardware devices and the software itself.

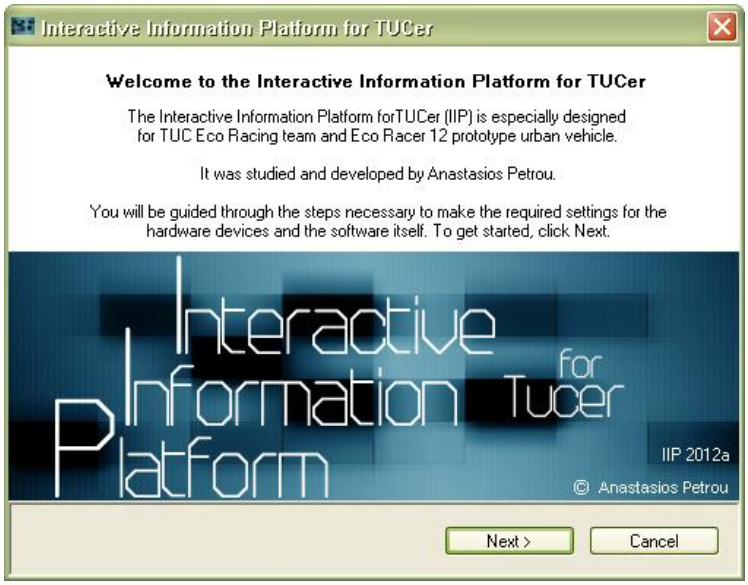

Figure 1.6: The welcome window of the software.

By clicking “Next” the license agreement of the software will appear and the user will be prompted to accept it (Figure 1.7).

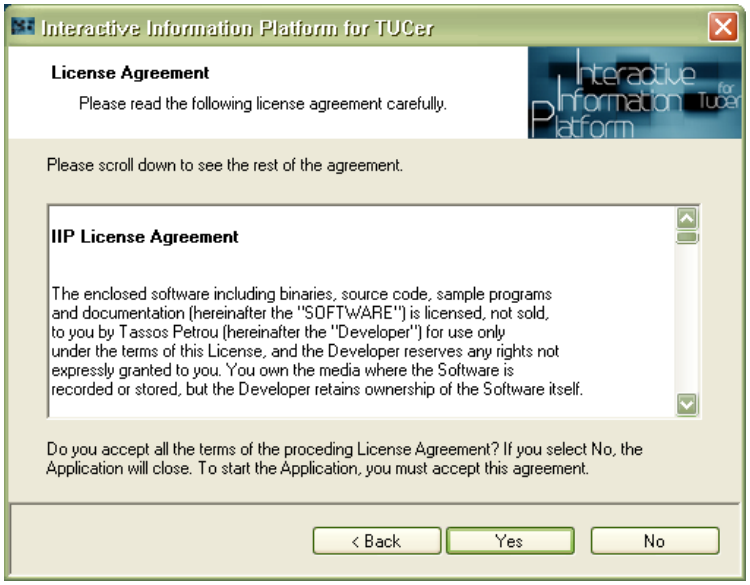

Figure 1.7: The license agreement window.

The user has to accept the license agreement by clicking “Yes” and then a window for the software’s folder destination path will appear (Figure 1.8). This folder will be the folder where the files of the recorded data using the data recording functionality, the images and information files for the vehicle tracking functionality and the sound files for the pedestrian sound warning functionality will be saved. User has the ability to select the destination path in which this folder will be created (Figure 1.9).

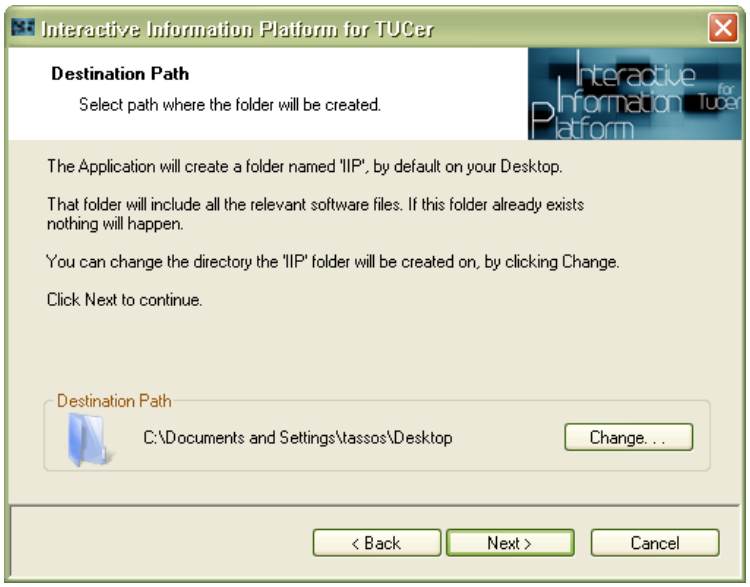

Figure 1.8: Window for setting the software’s folder path.

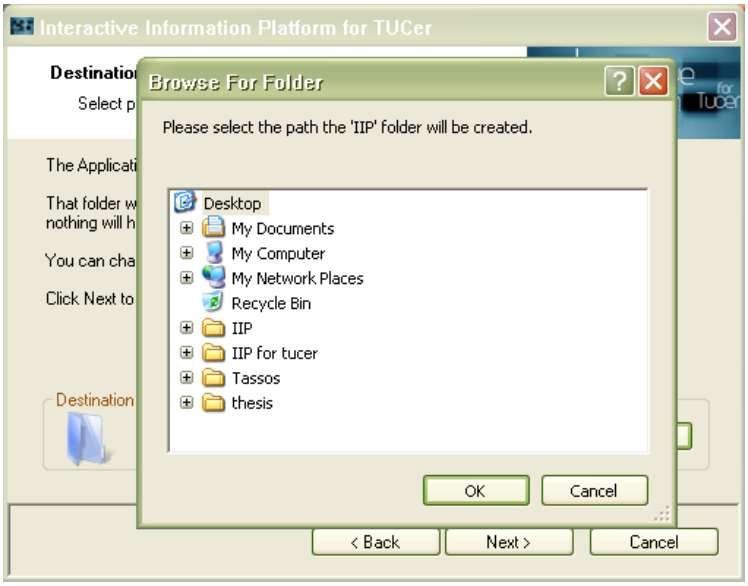

Figure 1.9: Browsing window for choosing the software's folder path.

By clicking "Next" a folder containing all the relevant software files will be created in the desired location and then a window for making the necessary serial ports settings for the hardware devices will appear (Figure 1.10).

Then the user is able to choose between two options. By clicking "Default Settings" the user chooses to make use of the application's functionality where the appropriate, for each hardware device, serial port is recognized on the fly. If the application doesn't recognize any of the hardware devises that are connected a warning message will appear (Figure 1.11).

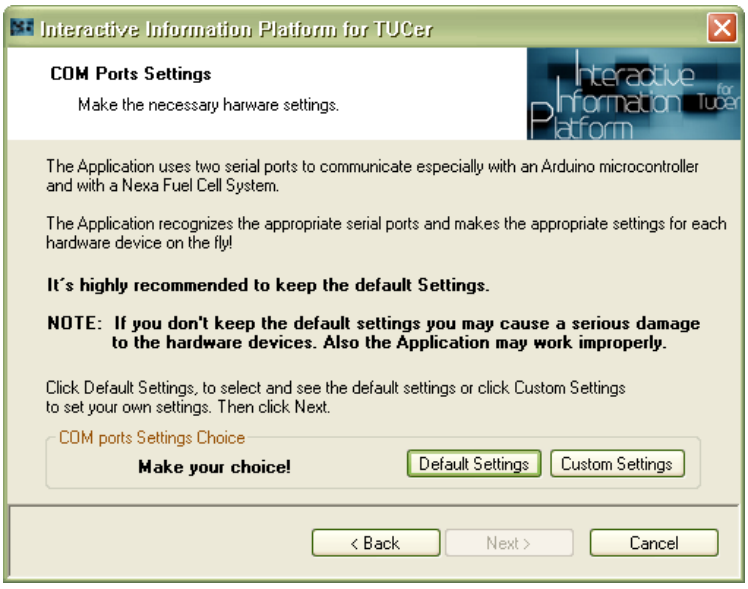

Figure 1.10: Window for the serial ports settings.

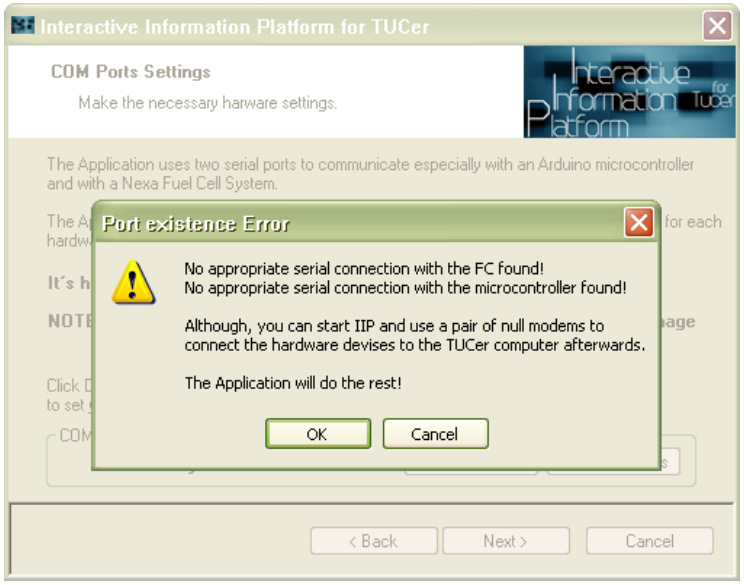

Figure 1.11: Warning message for the user's information.

By clicking "Custom Settings" the user has the ability to choose by his own the serial port for each hardware device and make the appropriate settings for it. A warning message will pop up to ensure the custom settings choice (Figure 1.12) and then a window for choosing the serial ports' settings will appear (Figure 1.13).

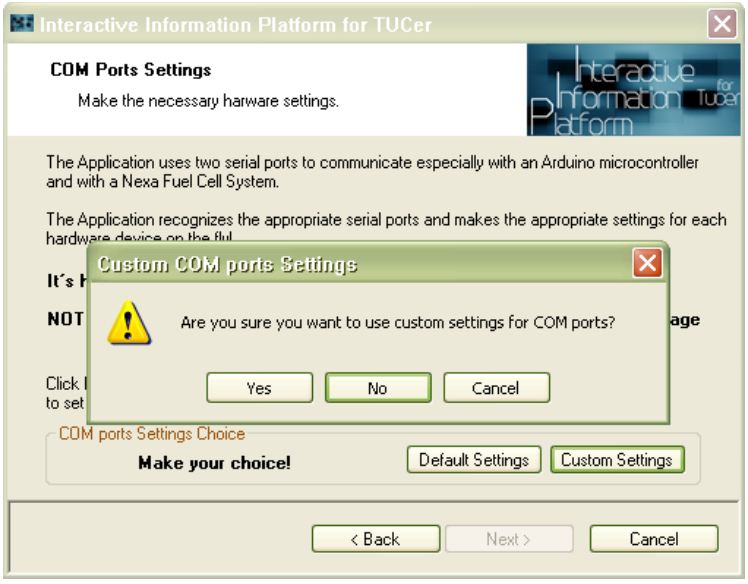

Figure 1.12: Warning message to ensure custom settings choise.

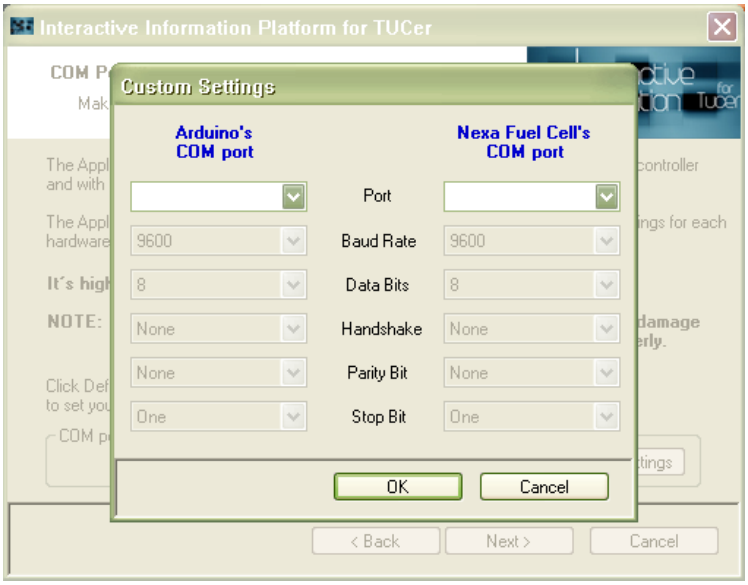

Figure 1.13: Window for the custom settings of the serial ports.

If the application doesn't obtain enough free serial ports from the system a warning message will pop up (Figure 1.14) and then a window for choosing the serial port's settings will appear. In this case a radio button is provided giving to the user the ability to choose the hardware device the serial port will belong to (Figure 1.15).

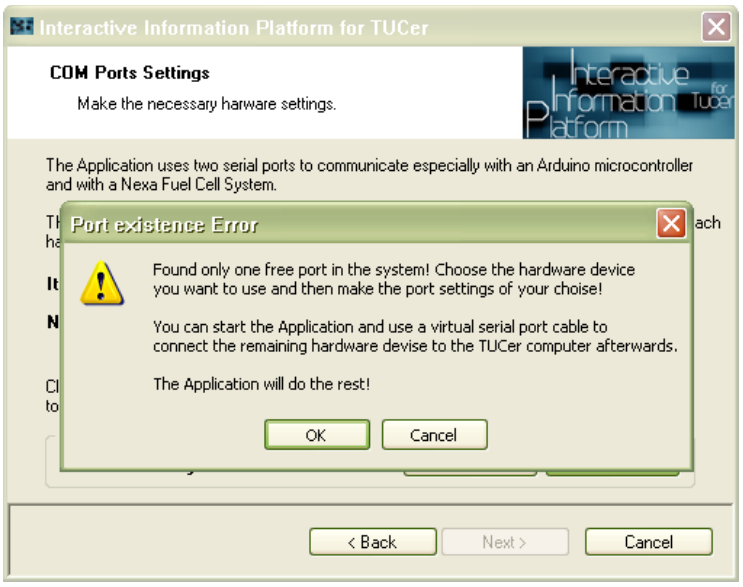

Figure 1.14: Warning message to inform the user that not enough serial ports found.

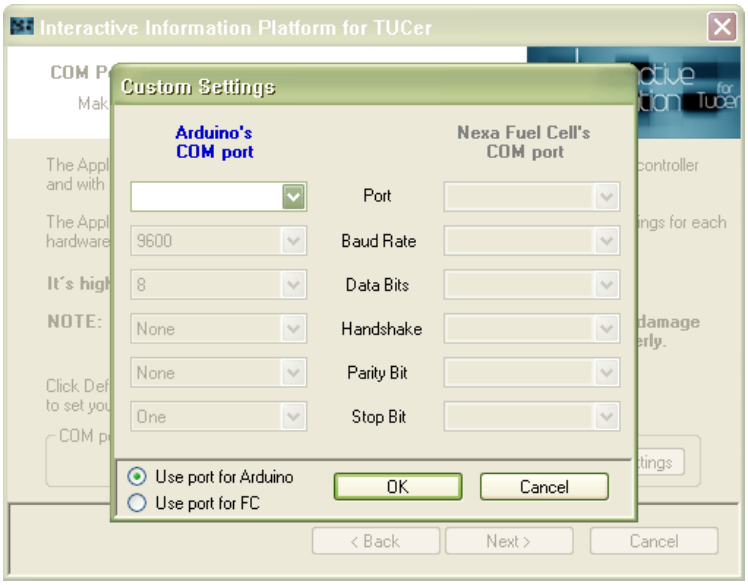

Figure 1.15: Window for the custom settings of the serial ports. A radio button is provided for the choice of the hardware device the serial port will belong to.

After the user makes his choice the “Next” button will be enabled and a message will appear on the parent window in the field “Com ports Settings Choice” informs user about his choice (Figure 1.16, 1.17).

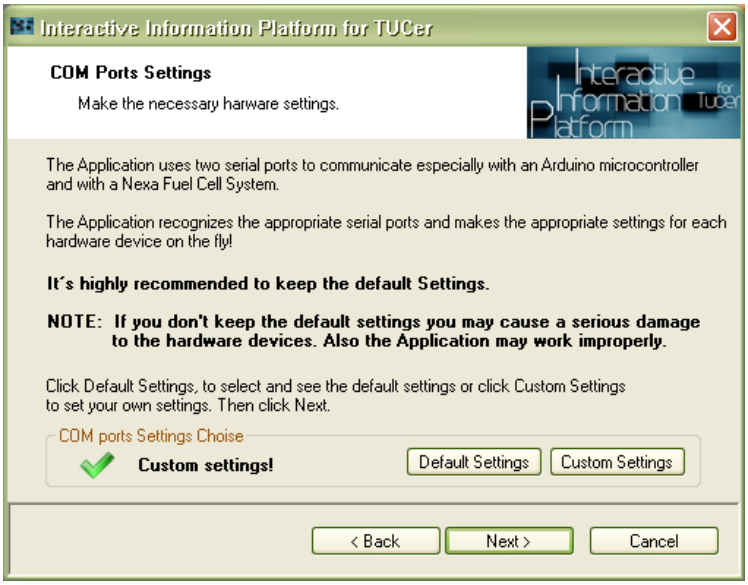

Figure 1.16: Message appearing after user’s choice for the port settings.

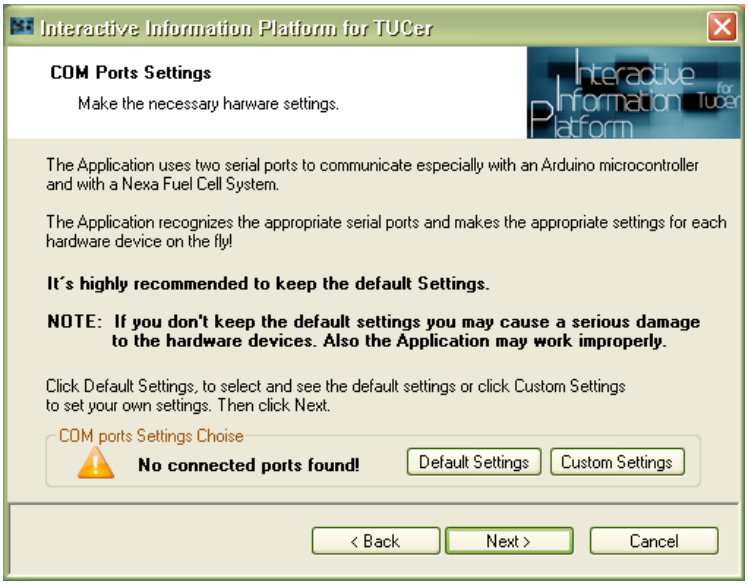

Figure 1.17: Warning message appearing to inform the user.

By clicking “Next” the Interactive graphical user interface will appear (Figure 1.18).

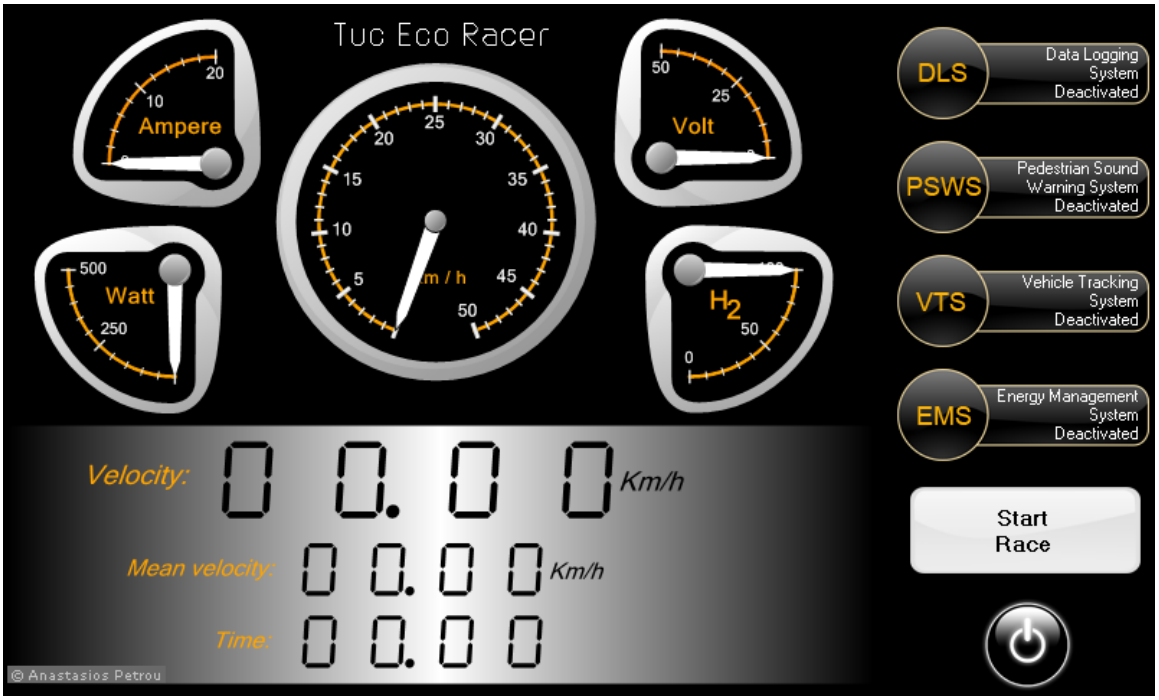

Figure 1.18: The interactive graphical user interface.

Figure 1.19: The inside of the TUCer 12 vehicle with the Interactive Information Platform and the touch screen.

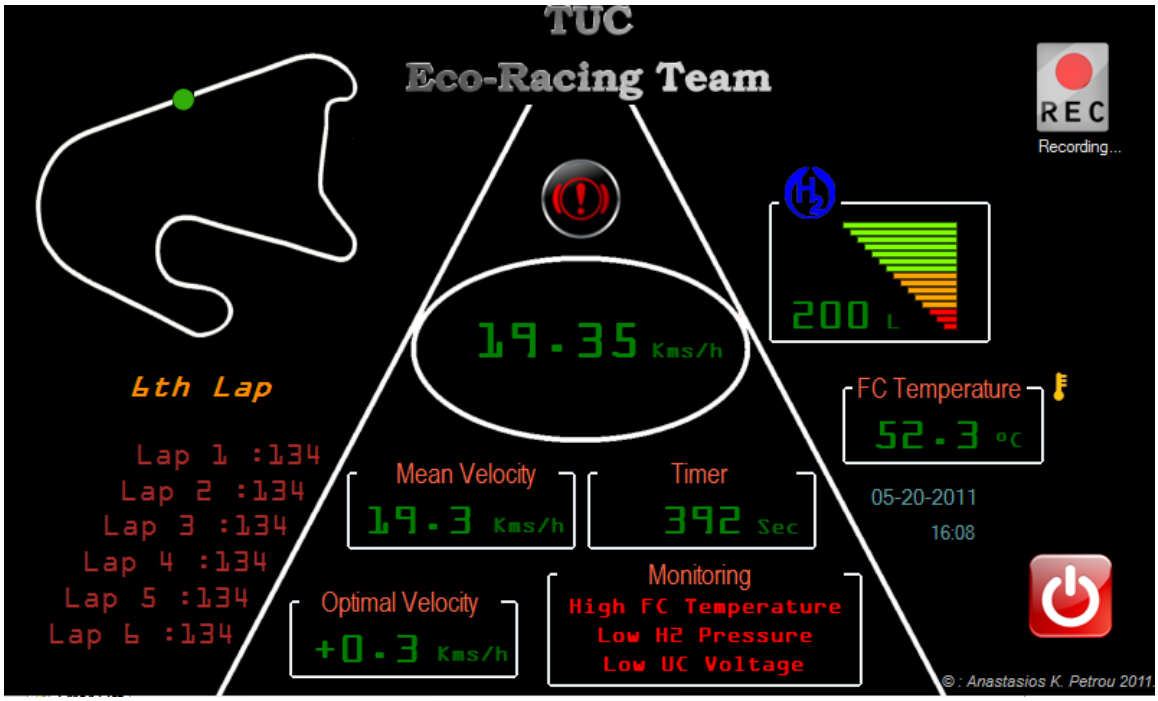

Figure 1.20: The first version (beta) of the interactive graphical interface.

1.4.1 The interactive graphical user interface

A friendly and functional interactive graphical user interface (GUI) provides the visualization functionality of the system's vital information and also provides the ability to the user to react with the system. It consists of several gauges and legible indicators. In the main area of the screen the user-driver can see the values of the measured data and in the right side of the screen the control buttons for the user-driver are provided. Figure 1.21 shows the interactive graphical user interface and its functionality. The description of the GUI is as follows:

1. Gauge for displaying hydrogen's fuel volume. By double pushing it the user has the ability to set fuel tank's capacity.

2. Gauge for displaying fuel cell stack voltage.

3. Gauge for displaying vehicle's current velocity.

4. Gauge for displaying fuel cell stack current.

5. Gauge for displaying fuel cell stack watt.

6. Warning indicator. It appears to inform driver for a warning situation.

7. Message displaying type of warning. Warning may be: Low Voltage, Low H2 volume, Low H2 pressure, High H2 pressure or the combination of any of the previous warnings.

8. Numeric indicator for displaying the vehicle's current velocity.

9. Numeric indicator for displaying the vehicle's mean velocity. It displays only if the "Start Race" button is pushed.

10. Numeric indicator for displaying the race time. It displays only if the "Start Race" button is pushed.

11. Track map for displaying vehicle's position during the race. It is visible only if the vehicle tracking system is activated.

12. Label displaying the lap of the race. It is visible only if the vehicle tracking system is activated.

13. Touch button for the activation/deactivation of the data recording functionality.

14. Touch button for the activation/deactivation of the pedestrian sound warning functionality.

15. Touch button for the activation/deactivation of the vehicle tracking functionality.

16. Touch button for the activation/deactivation of the energy management system.

17. Touch button has to be pushed when the race starts. By pushing this button indicators for displaying the vehicle's mean velocity and the race time become active.

18. Touch button for the termination of the application.

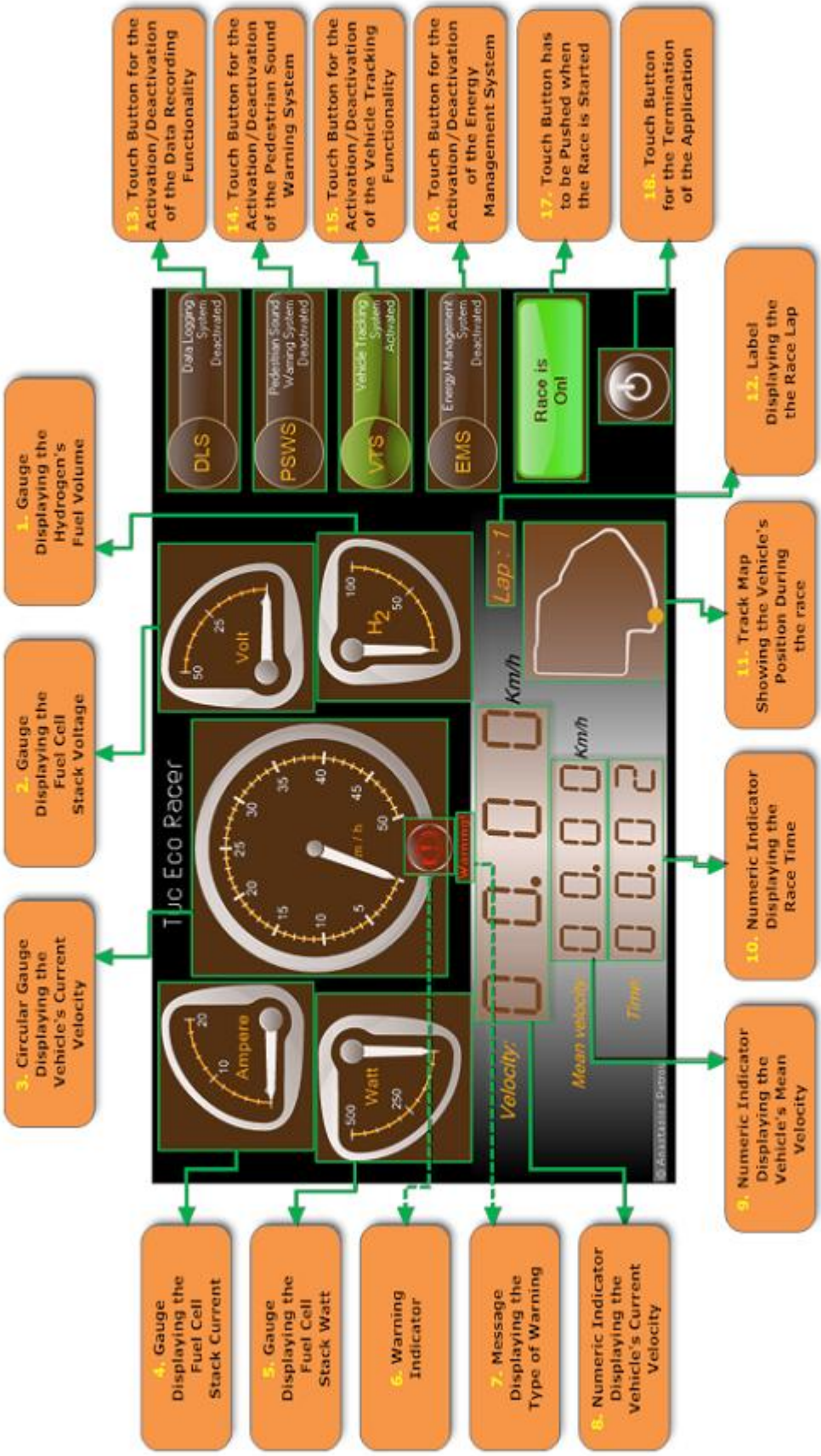

Figure 1.21: The interactive graphical user interface in detail.

1.4.2 Threads and events

As mentioned before the developed application is a multithreading and also an event driven application. This subchapter introduces the threads and the events of the software application.

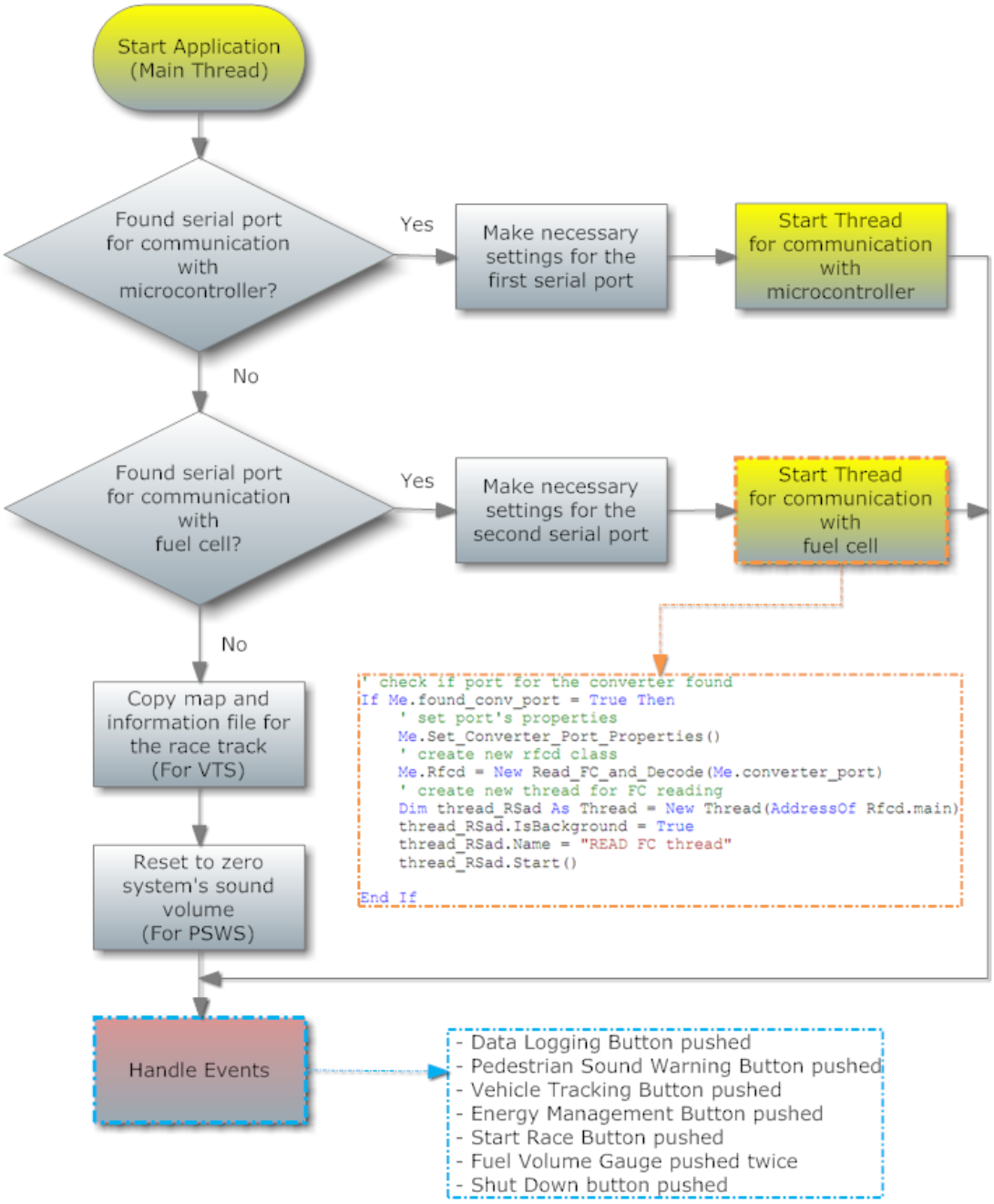

Figure 1.22: Flowchart of the application's main thread.

1.4.2.1 Main thread

The main thread of the application is responsible for the next processes:

- Checking for the existence of appropriate serial ports for each hardware device.
- If appropriate serial ports were found, the necessary settings for them are carried out.
- Two threads, a thread for the communication with the microcontroller and a thread for the communication with the fuel cell module are created.
- The relevant files (track pictures and information files) for the vehicle tracking functionality are copied to the application's folder.
- System's sound volume turned off for the initialization of the pedestrian sound warning functionality.
- Handling of the events.

Figure 1.22 shows the flowchart for the main thread.

1.4.2.2 Data acquisition from the fuel cell

Because of the continuous experimentation with the energy management system and the electronics components setup of the TUC Eco Racer 12 vehicle two different ways for the information acquisition from the fuel cell module were developed. The first way is the direct communication with the fuel cell module and the second way is the communication through the dc-dc converter.

1.4.2.2.1 Direct communication with the fuel cell module

A serial port is used for the communication between the fuel cell module and the computer unit. The fuel cell provides information such as Fuel Cell Stack Temperature, Fuel Cell Stack Voltage, Fuel Cell Stack Current, Hydrogen Pressure, Hydrogen Concentration, Cumulative Hydrogen Consumption, Oxygen Concentration, Ambient Temperature and Purge Cell Voltage.

The serial port interface uses full duplex communication, a pair of wires for transmission, and a pair of wires for reception. The full duplex communication allows asynchronous data transmission without needing to handle bus contention. The differential voltage levels used by the serial port are defined by the RS-485 standard. The features of the serial port communication follow:

1. Communication is asynchronous at 9600 baud, with the fuel cell module sending a data stream to the serial buffer approximately once every 200 ms.
2. SLIP (Serial Line Internet Protocol, Internet RFC 1055) is used to encode and decode the messages sent between devices. The SLIP code uses a one-byte tag (0xC0) at the beginning and at the end of each message. Three other special characters called "escape

characters," 0xDB, 0xDC, and 0xDD are required to handle cases where 0xC0 must occur in the middle of the message.

3. The message from the fuel cell will always include a 40 bytes segment at the beginning of the message that includes all relevant operating data. Up to an additional 100 bytes may be added for diagnostic and fault code retrieval purposes to the end of the message. These bytes should be considered unused bytes except for the purposes of computing the checksum at the end of the message.

4. In addition to the varying length of the message that accounts for the diagnostic transmission, additional bytes are required to handle the transmission of the "escape characters".

5. A check sum is computed over the entire message and displayed as the last byte at the end of the message. The check sum is computed as a simple summation of the message bytes. Overflow bits are discarded. The Check Sum does not include the Tags or any "escape characters."

6. Each character is sent containing 1 start bit, 8 data bits, no parity bit, and 1 stop bit.

7. The format for the message from the fuel cell is given below:

Tag	Status	Fail Code	Warning Bitmap	Last Command Acknowledge	Stack Temperature	Stack Voltage	Stack Current

H2 Pressure	H2 Concentration	H2 Cumulative Consumption	Oxygen concentration	Ambient Temperature	Purge Cell Voltage

Additional diagnostic and fault code bytes (0 to 100 extra bytes)	Checksum	Tag

8. Information in the message header and footer (the 2 Tags, the Status, the Fail Code, the Warning Bitmap, the Last Command Acknowledge, and the Check Sum) are sent as single bytes.

9. The fuel cell has the following Status Codes:

0x00 = Standby

0x01 = Start up

0x02 = Normal Operation

0x03 = Warning

0x04 = Normal Shut Down

0x05 = Failure Shut Down

0x06 = Non Restartable

10. The fuel cell has the following Fail Codes:

0x00 = Normal Operation

0x01 = High Fuel Cell Stack Temperature

0x02 = Low Fuel Cell Stack Voltage

0x03 = High Fuel Cell Stack Current

0x04 = Low Cell Voltage

0x05 = Low Fuel Pressure

0x06 = Fuel Leak Detected

0x07 = Low Oxygen Concentration

0x08 = Low Ambient Temperature

0x09 = Low Purge Cell Voltage

0x0A = Low Battery Voltage

0x0B = Startup Time Expired

0x0C = Self Test Fault

0x0D = General Software Fault

0x0E = Spurious Interrupt Fault

11. The fuel cell has the following Warning Bitmap Codes:

0x00 = No Warnings

0x01 = High Fuel Cell Stack Temperature Warning

0x02 = Low Fuel Cell Stack Voltage Warning

0x04 = High Fuel Cell Stack Current Warning

0x08 = Low Fuel Pressure Warning

0x10 = Fuel Leak Warning

0x20 = Low Oxygen Concentration Warning

0x40 = Low Purge Cell Voltage Warning

These warning codes are designed so that more than one warning can be issued at one time. The bitmap is a combination of the warnings present. The warning codes are combined with "OR" logic to form a single byte. For example, to send Low Fuel Cell Stack Voltage and Low Fuel Pressure Warnings simultaneously, the code 0x0A would be sent.

12. The Last Command Acknowledge is a repetition of the last command received.

13. The data (Fuel Cell Stack Temperature, Voltage, Current, Hydrogen Pressure, Hydrogen Concentration, and Cumulative Hydrogen Consumption, Oxygen Concentration) are sent as floating point numbers using the following 4 byte format as follows:

Sign (1 bit)	Exponent (8 bits)	Mantissa (23 bits)

The 4 bytes are arranged in the following fashion:

Sign (1 bit) + Exponent (7 MSB's)	Exponent (LSB) + Mantissa (7 MSB's)	Mantissa (8 bits)	Mantissa (8 LSB's)
Fourth byte	Third byte	Second byte	First byte

The mantissa and the exponent are arranged so that the Most Significant Bit (MSB) is on the left and the Least Significant Bit (LSB) is on the right. To convert this format into a decimal number, the following formula is used:

Where:

Sign is either 1 or 0

Exponent is 8 bits (0 to 255)

Mantissa is 23 bits

14. The engineering units for the data are as follows:

Data	Unit
Fuel Cell Stack Temperature	^{o} C
Fuel Cell Stack Voltage	Volt
Fuel Cell Stack Current	Ampere
Hydrogen Pressure	Bar (gauge)
Hydrogen Concentration	ppm
Cumulative Hydrogen Consumption	Standard liters
Oxygen Concentration	Percent (%)
Ambient Temperature	^{o} C
Purge Cell Voltage	Volt

1.4.2.2.2 Slip decoding

In Normal Mode, the fuel cell system transmits a 40-character status message followed by a 1 byte checksum. If the status data contains the SLIP End Character 0xC0 or the SLIP Escape Character 0xDB then each occurrence of these characters is encoded as a two-byte escape

sequence consisting of the Escape Character 0xDB followed by the Escape-Esc Character 0xDC or the Escape-End Character 0xDD, as appropriate. Hence an encoded SLIP message is transmitted by the fuel cell as a character stream that is a minimum of 43 bytes (0xC0, 40 status bytes, 1 byte checksum and 0xC0) and a maximum of 84 bytes. In reality, the 84-byte message will never be observed since the values 0xC0 and 0xDB will never appear in the status message status code, failure code, warning bit map, and acknowledgement fields.

As an aside, the fuel cell module has a Diagnostic Mode in which an extended status message is transmitted. A receiver that knows only the structure of the basic Normal Mode message can still correctly process a Diagnostic Mode message without knowing its complete structure since the first 40 bytes of the diagnostic Mode message are the same as the Normal Mode message and the last byte is always the checksum over the entire message. Thus, the receiver should compute the checksum over any valid message it gets, regardless of length, and compare it to the last byte in the message to determine the message's validity. Then the receiver can decide whether to make use of the first 40 bytes or the extended message, as appropriate.

In summary, the invocation of the SLIP decode routine by the receiver should not depend on or be triggered by the receipt of any specific of characters. Instead, the SLIP decode routine should be called whenever a 0xC0 character is received. The receive algorithm can be implemented in one of two ways. Receive Algorithm B is a simplification of Algorithm A.

Receive Algorithm A

1. When any character arrives on the serial interface, put the character into the serial receive buffer.

2. If the received character is 0xC0, call the SLIP decode routine, passing it all the data currently in the receive buffer.

3. If the SLIP decode routine returns 0, then the decode operation failed. This will happen if the receive buffer contains only the 0xC0 character or if there was noise on the serial line and a byte-stuffed SLIP character was dropped from the message.

4. If the SLIP decodes routine returns a non-zero value, N, then the first N-1 characters constitute the fuel cell status message and the Nth character is the 8-bit checksum over the preceding N-1 characters. Compute the checksum over the first N-1 characters and compare it to the Nth octet. If they match, then the (N-1)-char status messages has been received intact. If not, discard the message.

Receive Algorithm B

1. When any character arrives on the serial interface, examine it to see if it is the SLIP End Character, 0xC0. If not, put the character in the receive buffer. If so and the receive buffer is empty, continue. Otherwise, invoke the SLIP decode routine on the contents of the receive buffer.

2. If the SLIP decode routine returns 0, then the decode operation failed. This will only happen if there was noise on the serial line and a byte-stuffed SLIP character was dropped from the message.

3. If the SLIP decode routine returns a non-zero value, N, then the first N-1 characters constitute the Nexa status message and the Nth character is the 8-bit checksum over the preceding N-1 characters. Compute the checksum over the first N-1 characters and compare it to the Nth octet. If they match, then the (N-1)-char status messages has been received intact. If not, discard the message.

Strictly speaking, a packet that conforms to the SLIP protocol need only have the trailing 0xC0. It is standard practice, however, to prefix a SLIP-encoded message with a leading 0xC0. The purpose of this is to 'flush' a partially-received message from the receiver's buffer, i.e. a message whose tail (including its trailing 0xC0) was corrupted or truncated due to noise on the serial line. The leading 0xC0 of the next message will cause the partially-received data of the previous message to be flushed out of the receive buffer and be passed to the SLIP decode routine. The SLIP decode operation may or may not succeed.

Regardless, integrity of a fuel cell's message is protected by its checksum. If the checksum byte has been damaged or dropped then the checksum calculation that follows the SLIP decode will detect the damaged message.

Note that it is crucial that a character beginning and ending with 0xC0 not be passed to the SLIP decode routine. If a 0xC0 appears in the receive data passed to this routine it must only be at the end of the buffer. The serial receive algorithm outlined above guarantees that this is the case. Under normal circumstances where there is no noise on the serial line, the receive interface will get both the leading 0xC0 and the trailing 0xC0.

In Receive Algorithm A above, the leading 0xC0 will be passed to the SLIP decode routine by itself. Since it is preceded by no data, the SLIP decode routine will return 0, indicating that a valid SLIP message has not been received. When the trailing 0xC0 is received, the SLIP-encoded message and the trailing 0xC0 will be passed to the SLIP decode routine and the message will be properly decoded. Therefore, it is typical that the SLIP decode routine will be called twice for each status message transmitted by the fuel cell module, the first time for the leading 0xC0 and the second time for the status message and the trailing 0xC0.

In Receive Algorithm B the SLIP decode routine is invoked only once per status message since the 0xC0 character is not added to the receive buffer and the decode routine is called only if the receive buffer is not empty.

Although, because the process of the constant sampling of the fuel cell's serial port is a very laborious process for the CPU and because there is no need of knowing the diagnostic message that may be transmitted, Receive Algorithm C was developed. According to this algorithm the serial reading took place every two seconds.

Receive Algorithm C

1. Sleep thread for two seconds.

2. Discard any characters are on the input buffer.

3. Check the character that arrives on the input buffer. If the character is the 0xC0 character, discard it and check next character. If next character is the 0xC0 character, discard it and put the characters follow into the serial receive buffer. If not, put character into serial buffer and put the characters follow into the serial receive buffer too.

4. When the serial receive buffer fills up with 42 characters check the 41th character (CheckSum byte). If this character is equal to 40 then the message has been received intact. So, the decode method is called. If not, there was noise on the serial line or one or more characters were dropped from the message. So, discard message and go to step 2.

5. Go to step 1.

The implementation of the Receive Algorithm C presented on Table 1.2.

```
    ' private function for reading values from FC
    Private Function readFC() As Byte()

        Dim readingOk As Boolean = False
        Dim receivedata(42) As Byte

        Try
            'check if com port is open. if not open it
            If converter_port.IsOpen = False Then
                converter_port.Open()
            End If
        Catch ex As Exception
        End Try

        ' try to discard input buffer
        Try
            converter_port.DiscardInBuffer()
        Catch ex As Exception
        End Try
        ' Read from FC
        Try
            While readingOk = False
                ' Read byte from input buffer
                Dim bt As Byte = converter_port.ReadByte

                If bt = &HC0 Then ' Check if byte is the 0xC0 byte

                    receivedata(0) = bt

                    Dim bt2 As Byte = converter_port.ReadByte
                    ' Read byte from input buffer
                    ' Check if next byte is the 0xC0 byte
                    If bt2 = &HC0 Then
                        ' save byte into receive buffer
                        receivedata(0) = bt2
                        While converter_port.BytesToRead < 42
                            ' do nothing until the input buffer
                            ' fills up with 42 bytes
                        End While
                        'save the 42 bytes into the receive buffer
                        converter_port.Read(receivedata, 1, 42)
                        readingOk = True
                    Else
                        ' save byte into receive buffer
                        receivedata(1) = bt2
                        While converter_port.BytesToRead < 41
                            ' do nothing until the input buffer
                            ' fills up with 41 bytes
                        End While
                        'save the 41 bytes into the receive buffer
                        converter_port.Read(receivedata, 2, 41)
                        readingOk = True
                    End If
                End If

            End While
        Catch ex As Exception
        End Try

        Return receivedata
```

```
End Function
```

Table 1.2: Receive Algorithm C.

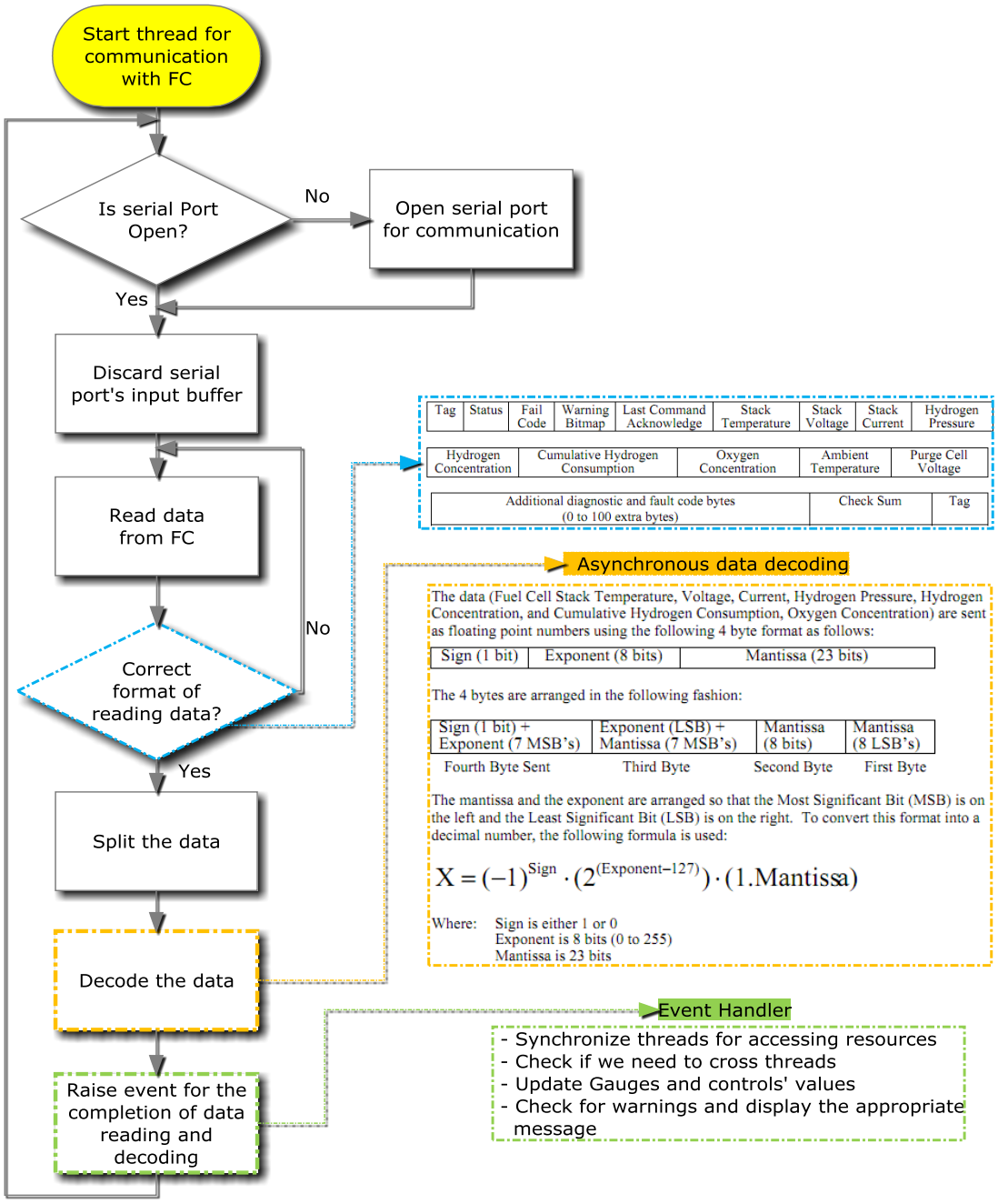

Figure 1.23: Thread for the communication with the FC.

The processes of the data acquisition from the FC and the decoding of the retrieved data are executed in a different thread inside the application. The flow chart for this thread is shown in Figure 1.23. The thread's functionality for the communication and the decoding of the fuel cell is:

- Checking if serial port is open. If not, open it.
- Discard serial port's input buffer.
- Read data from fuel cell.
- Check if received data have the right format.
- Split data.
- Decode data.
- Raise the event for completion of the process.
- Synchronize threads for accessing resources.
- Check if there is a need for crossing threads.
- Update Gauges and controls' values.
- Check for warnings and display the appropriate message.

1.4.2.3. Communication through the dc-dc converter

A serial port is used for the communication between the dc-dc converter and the computer unit. The dc-dc converter provides information such as Fuel Cell Stack Temperature, Fuel Cell Stack Voltage, Fuel Cell Stack Current, Hydrogen Pressure, Hydrogen Concentration, Cumulative Hydrogen Consumption, Oxygen Concentration, Ambient Temperature and Purge Cell Voltage etc.

A 9-pin RS232 direct link is used. The baud rate is 38400 baud with one stop bit, none parity bit, one start bit and 8 bit data length. The protocol for retrieving data out of dc-dc converter (Figure 1.24) is as follows. Master indicates the computer unit and slave indicates the dc-dc converter:

- Master starts data transfer by sending first byte.
- Slave responds within 10ms.
- Master has to wait for slave reply until next byte is sent.
- Master monitors time out and correctness of slave values.
- After data reading the values are scaled according to their scaling factors.

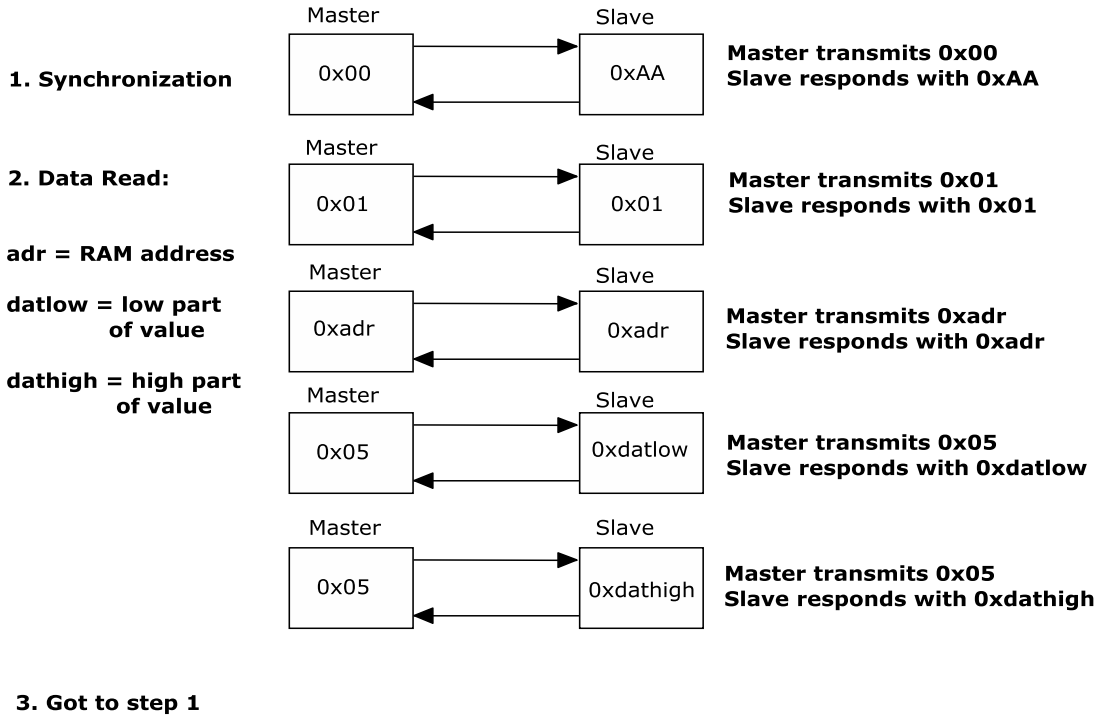

Figure 1.24: Communication protocol for the dc-dc converter.

Table 1.3 shows the addresses and the scaling factors of the dc-dc converter and the code implementation for the communication with the dc-dc converter is shown on Table 1.4.

Adress	Variable	Explanation	Scaling Factor
0x3F	nexa_warnings	Warnings from Nexa	
0x3F.0	stack_temp_warn		
0x3F.1	stack_volt_warn		
0x3F.2	stack_current_warn		
0x3F.3	fuel_press_warn		
0x3F.4	fuel_leak_warn		
0x3F.5	low_ox_conc_warn		
0x3F.6	low_purge_cell_volt_warn		
0x40	dcdc_error_flags		
0x40.0	Gesamterror	Summation error bit	
0x40.1	Temp_error	Over temperature	
0x40.2	V_bat_min_error	Battery under voltage	
0x40.3	V_bat_max_error	Battery over voltage	
0x40.4	V_nexa_min_error	Nexa under voltage	
0x40.5	V_nexa_max_error	Nexa over voltage	
0x40.6	I_max_error	Over current	

0x41.7	Unit_12V	1 = 12 Volt-unit 0 = 24 Volt- unit	
0x42	V_Nexa	Nexa output voltage	210 = 60 V
0x44	I_Battery	Battery current	210 = 71 A (Unit_12V=0); 210 = 142 A (Unit_12V=1)
0x46	V_Battery	Battery voltage (! Scale according to bit "Unit_12V")	210 = 36,71 V (Unit_12V=0); 210 = 18,355 V (Unit_12V=1)
0x49	Regime	0 - Start 1 - Standby 2 – Power_Start 3 – Mainloading_Current 4 - Mainloading _Voltage 5 – Timeloading 6 – Shut down 7 - Error	
0x4A	V_ch_min	Minimum battery voltage	210 = 36,71 V (Unit_12V=0); 210 = 18,355 V (Unit_12V=1)
0x4C	V_ch_max	Maximum battery voltage	210 = 36,71 V (Unit_12V=0); 210 = 18,355 V (Unit_12V=1)
0x4E	T_reload	Reload time	210 = 85,25
0x50	I_ch_max	Maximum battery current	210 = 71 A (Unit_12V=0); 210 = 142 A (Unit_12V=1)
0x52	I_change	Current for changing to phase reload	210 = 71 A (Unit_12V=0); 210 = 142 A (Unit_12V=1)
0xA0-0xC4	nexa_rs485_buffer[40]		
0xA0	Status		
0xA1	Failcode		
0xA2	Warning Bitmap		
0xA3	Last Command Acknowl.		
0xA4	Stack Temperature		
0xA8	Stack Voltage		

0xAC	Stack Current		
0xB0	Hydrogen Pressure		
0xB4	Hydrogen Concentration		
0xB8	Cumul. Hydrogen Consumption		
0xBC	Oxygen Concentration		
0xC0	Ambient Temperature		
0xC4	Purge Cell Voltage		

Table 1.3: Addresses and scaling factors of the dc-dc converter

```
 Dim sync() As Byte = {&H0, &H1}
Dim H2pressureadr() As Byte = {&HB8, &HBA}
Dim answer As Byte() = {&H0}
Dim five As Byte() = {&H5}
 Dim pressure() As Byte = {&H0, &H0, &H0, &H0}
'… … …
'''''''''''''''''''' Synchronization
SerialPort1.Write(sync, 0, 1)
SerialPort1.Read(answer, 0, 1)
SerialPort1.Write(sync, 1, 1)
SerialPort1.Read(answer, 0, 1)

'''''''''''''''''''' Read two first bytes
SerialPort1.Write(H2pressureadr, 0, 1)
SerialPort1.Read(answer, 0, 1)
SerialPort1.Write(five, 0, 1)
SerialPort1.Read(pressure, 3, 1)
SerialPort1.Write(five, 0, 1)
SerialPort1.Read(pressure, 2, 1)
  '… … …
```

Table 1.4: Part of the implemented code for the communication with the dc-dc converter.

1.4.2.4. Communication with the microcontroller

A serial port is used for the communication between the microcontroller and the computer unit. The microcontroller provides information such as track inclination, vehicle's current velocity, vehicle's current acceleration and the vehicle's traveled distance.

The serial port interface uses full duplex communication, a pair of wires for transmission, and a pair of wires for reception. The full duplex communication allows asynchronous data transmission without needing to handle bus contention at 9600 baud rate. For the communication between the microcontroller and the software application a custom code was developed at a microcontroller level (Table 1.5).

```
// 07/05/2012
// Petrou Tassos
// v.1.3
```

```
const int accelerationPin = 1; // pin for accelerometer
const int inclinationPin = 2; //  pin for inclinometer
int accelerationValue ; // x-axis acceleration value
int inclinationValue ; // road inclination value
double distance ; // vehicle's traveled distance
double velocity ; // vehicle's velocity
unsigned long oldtime ; // time for the completion of the last wheel roatation
unsigned int totalRotationCounter ; // number of wheel rotations
boolean oneRotation ;

void setup()
{
  Serial.begin(9600);
  Serial.flush();
  attachInterrupt (0,rotation,RISING);
  velocity = 0;
  oldtime = 0;
  totalRotationCounter = 0;
  distance = 0;
  oneRotation = false ;
}

void loop()
{
  int serialRead = 0;
  accelerationValue = analogRead(accelerationPin);
  inclinationValue = analogRead(inclinationPin);

  if (oneRotation)
  {
    velocity = (2 * 3.14 * 0.27 * 3.6 * 1000 / (millis()-oldtime));
    distance = 2 * 3.14 * 0.27 *  totalRotationCounter;
    oldtime = millis();
    oneRotation = false;
  }
// if no wheel rotation has completed in 4 seconds then set velocity equal to zero
  if ((millis() - oldtime) > 4000)  {
    velocity = 0 ;
  }

  if (Serial.available() > 0)
  {
// deduct ascii value of '0' to find numeric value of sent number
    serialRead = Serial.read() - '0' ;
     if (serialRead == 0) // Reset totalRotationCounter
    {
      totalRotationCounter = 0;
      distance = 0;
      Serial.println(distance); // send distance
      Serial.flush();
    }
    if (serialRead == 1) // send acceleration value
```

```
    {
      Serial.println(accelerationValue);
      Serial.flush();
    }
    else if (serialRead == 2) // send inclination value
    {
      Serial.println(inclinationValue);
      Serial.flush();
    }
     else if (serialRead == 3) // send velocity
    {
      Serial.println(velocity,2);
      Serial.flush();
    }
     else if (serialRead == 4)
    {
      Serial.println(distance,2); // send distance
      Serial.flush();
    }

    else
    {
    }
  }
}
void rotation()
{
  totalRotationCounter++;
  oneRotation = true;
}
```

Table 1.5: Implemented code at the microcontroller level.

The process for the communication between the microcontroller and the software application is executed in a different thread. The thread's functionality is as follows:

- Checking if serial port is open. If not, then thread opens it.
- Discard serial port's input buffer.
- Synchronize threads. It sends to the serial port the appropriate commands for reading data.
- Reading the microcontroller's response.
- Synchronize threads for accessing variables.
- Checking if Pedestrian Sound Warning System is activated.
- If PSWS is activated a thread for the PSWS functionality is created.
- Checks if there is a need to cross threads, update gauges and controls' values.

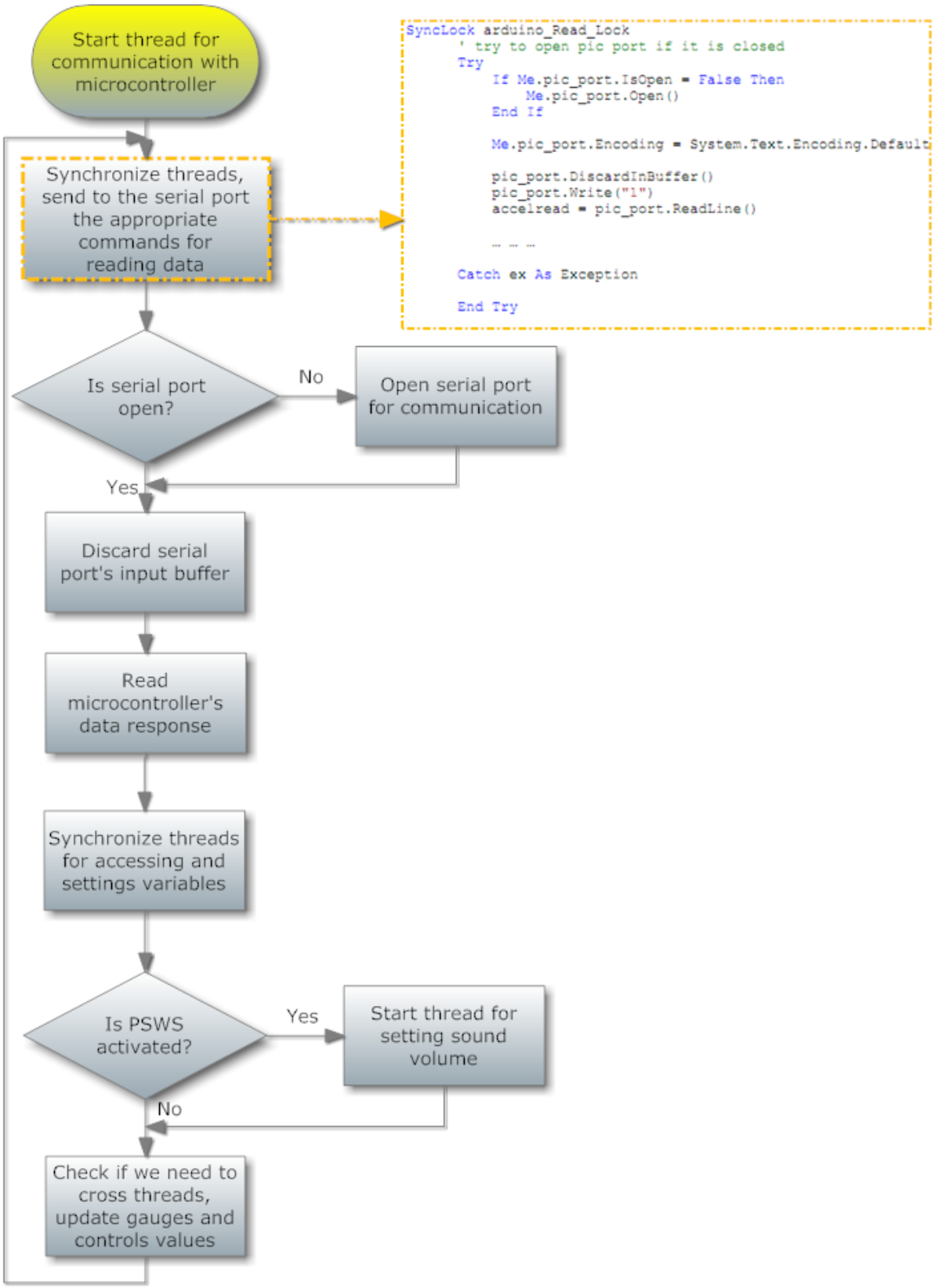

Figure 1.25: Flowchart for the communication with the microcontroller.

1.4.2.5 Data recording functionality

The interactive graphical user interface provides the ability to the user-driver to activate-deactivate the data recording functionality, by touching the corresponding button. This functionality is executed by the software application in a different thread. Figure 1.26 shows the flowchart for the data logging functionality push button event and for the functionality thread.

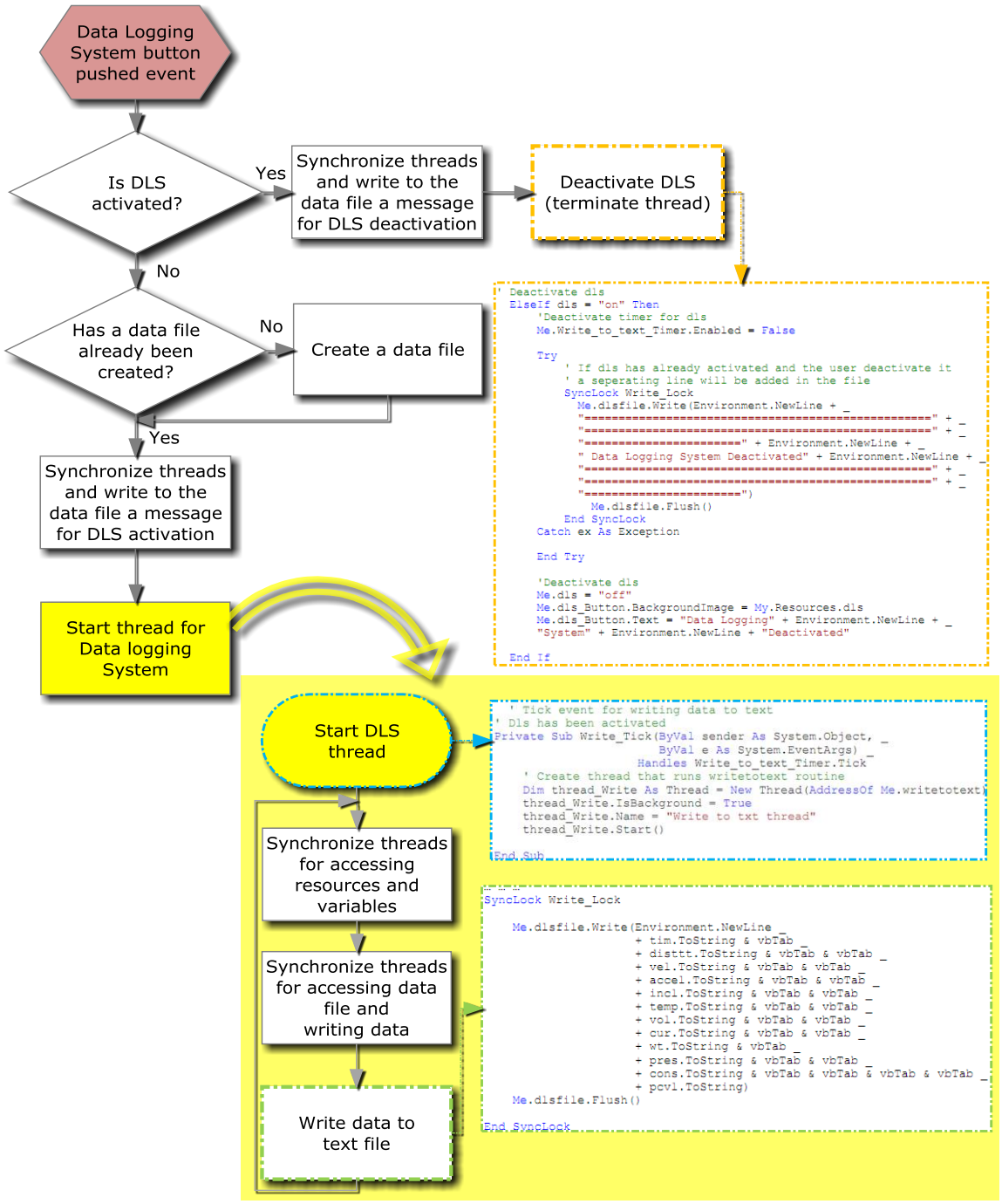

Figure 1.26: Flowchart for the data logging thread.

When the user-driver pushes the data logging button the sequence of the processes that executed is as follows:

- Checking for the state of the data recording functionality. If data recording is activated then terminate thread. If not, continue.
- Checking for the existence of a data recording file. If a data file hasn't already been created, then the thread creates one.
- Synchronize threads. Write a DLS activation message to the data file.
- Start thread for the data recording functionality.

The code for the data logging functionality push button event is shown in Table 1.6.

The functionality of the data recording thread is as follows:

- Synchronize threads for accessing resources and variables.
- Synchronize threads for accessing and writing to the data file.
- Write data to the file.

```
' Click event for data logging system
    Private Sub dls_Button_Click(ByVal sender As System.Object, _
                                 ByVal e As System.EventArgs) _
                                 Handles dls_Button.Click

        ' Activate dls
        If dls = "off" Then

            Me.dls = "on"
            Me.dls_Button.BackgroundImage = My.Resources.dlsa2
            Me.dls_Button.Text = "Data Logging" + Environment.NewLine + _
            "System" + Environment.NewLine + "Activated"

            ' Check if data file is not already created
            If Me.dlsfilename Is Nothing Then
                ' Data file is not created. It will be now!

                ' Current date and time
                Dim datetime As String = Format(Now, "dd-MM-yyyy  h,mm,ss")
                ' Set name of file that keeps the recording data
                Me.dlsfilename = Me.datafolderpath + "\data (" + datetime +
                                                        ")" + ".log"
                ' Try

                ' Create file that keeps the recording data
                Me.dlsfile = New System.IO.StreamWriter(Me.dlsfilename)
                ' Write the first line to the data file
                Me.dlsfile.Write("Time" & vbTab + "Distance" & vbTab & vbTab + "Velocity" _
                           & vbTab & vbTab + "Acceleration" & vbTab + "Inclination" _
                           & vbTab + "FC Temperature" & vbTab + "FC Voltage" & vbTab +
"FC_Current" _
                           & vbTab + "Watt" & vbTab + "H2_Pressure" & vbTab +
"H2_Cumulative_Consumption" _
                           & vbTab + "Purge Cell Voltage" + vbNewLine)
                Me.dlsfile.Flush()

                'Catch ex As Exception

                'End Try

            Else
                ' Data file is already created. Do nothing

            End If

            Try
                ' If dls has already activated and the user deactivate it
                ' a seperating line will be added in the file
                SyncLock Write_Lock
```

```
                    Me.dlsfile.Write(Environment.NewLine + _
"=========================================================================" +
"=========================================================================" + _
                                    "======================" + Environment.NewLine + _
                                    " Data Logging System Activated" + Environment.NewLine +
"=========================================================================" + _
"=========================================================================" + _
                                    "======================")
                    Me.dlsfile.Flush()
                End SyncLock
            Catch ex As Exception

            End Try

            'Activate timer for dls
            Me.Write_to_text_Timer.Enabled = True

            ' Deactivate dls
        ElseIf dls = "on" Then
            'Deactivate timer for dls
            Me.Write_to_text_Timer.Enabled = False

            Try
                ' If dls has already activated and the user deactivate it
                ' a seperating line will be added in the file
                SyncLock Write_Lock
                    Me.dlsfile.Write(Environment.NewLine + _
"=========================================================================" + _
"=========================================================================" + _
                                    "======================" + Environment.NewLine +
                                    " Data Logging System Deactivated" + Environment.NewLine + _
"=========================================================================" + _
"=========================================================================" +
                                    "======================")
                    Me.dlsfile.Flush()
                End SyncLock
            Catch ex As Exception

            End Try

            'Deactivate dls
            Me.dls = "off"
            Me.dls_Button.BackgroundImage = My.Resources.dls
            Me.dls_Button.Text = "Data Logging" + Environment.NewLine + _
            "System" + Environment.NewLine + "Deactivated"

        End If

    End Sub
```

Table 1.6: Implemented code for the data logging functionality.

1.4.2.6. Setting fuel's volume functionality

The interactive graphical user interface provides the ability to the user- driver to set the fuel tank's maximum volume, by touching twice the gauge displaying the fuel's volume. This

functionality is provided because there is no other way for the application to know the fuel's volume at the time that application starts. Figure 1.27 shows the flowchart for the fuel gauge double push button event.

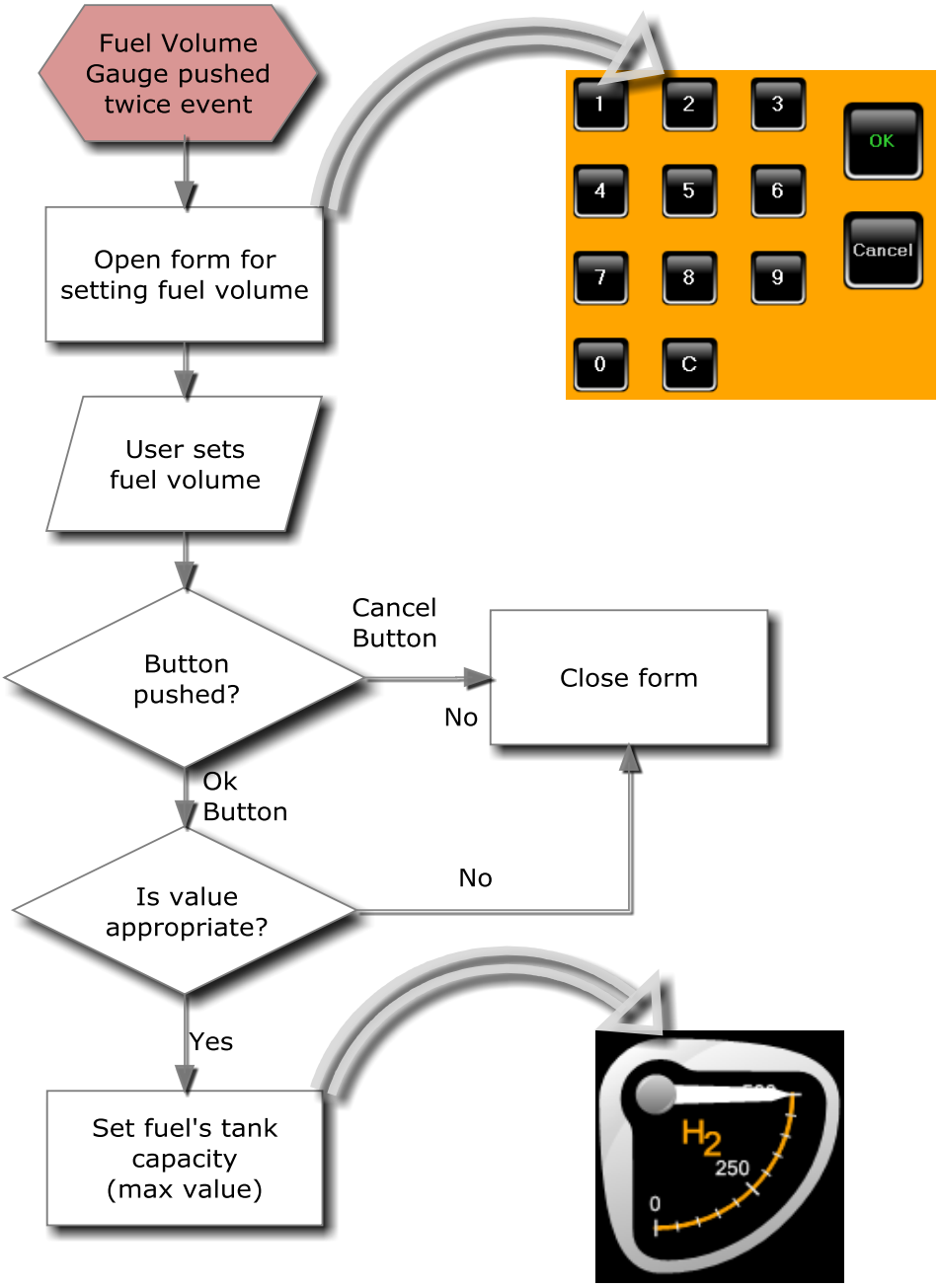

Figure 1.27: Flowchart for the fuel gauge double push event.

1.4.2.7 Vehicle tracking system functionality

The interactive graphical user interface also provides the ability to the user- driver to activate-deactivate the vehicle tracking functionality by touching the corresponding button. The functionality of the vehicle tracking is executed in a different thread in the application also. Figure 1.28 shows the flowchart for the vehicle tracking functionality push button event and Table 1.8 shows the code that updates vehicle's position.

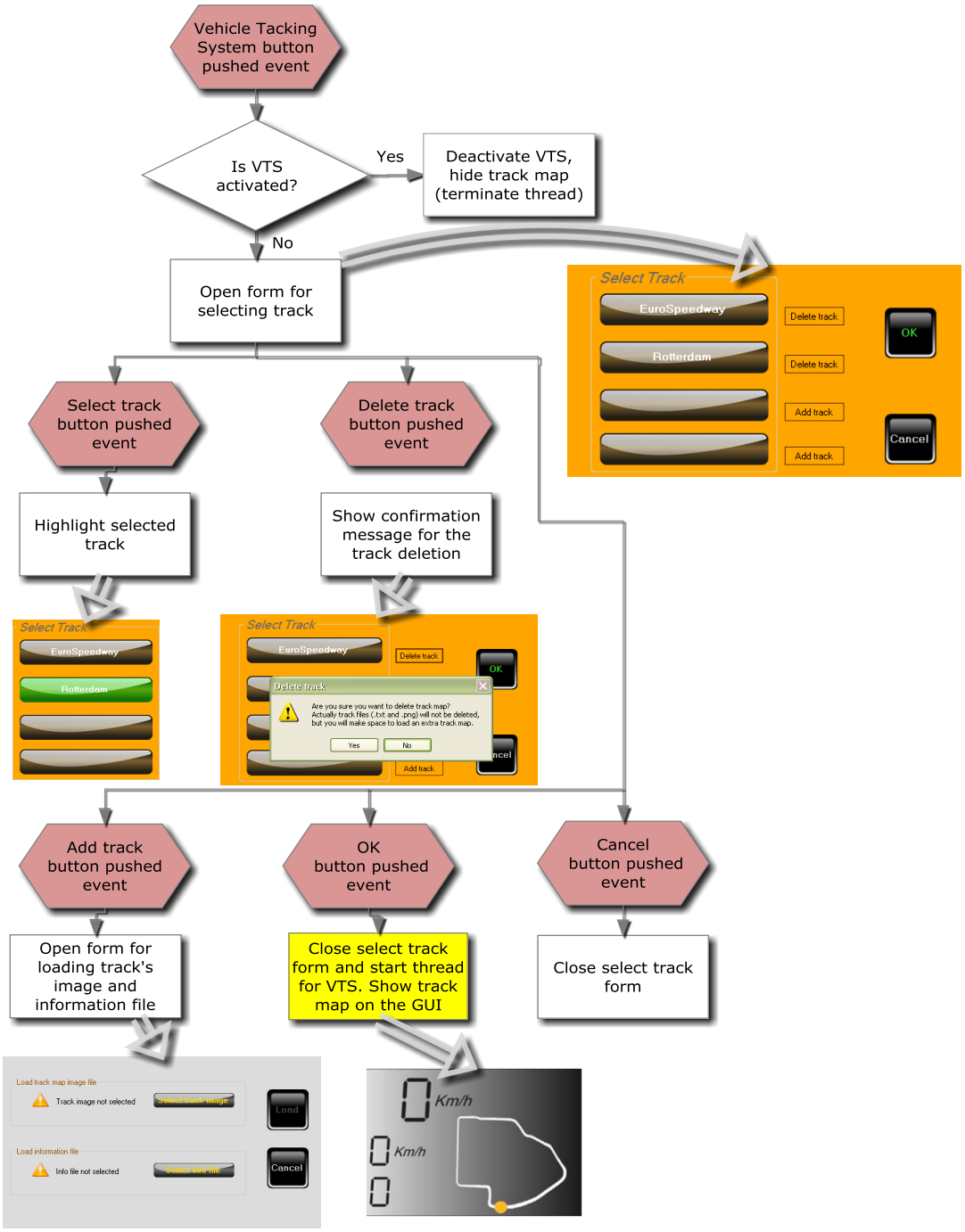

Figure 1.28: Flowchart for the VTS button pushed event.

```
'
' Routine to show vehicles position
'
Private Sub CalculatePosition()
    ' show dot image and give it the start mark coordinates
    Dim position As Integer = 0
    Dim dist As Double = 0
    Dim tracklength As Integer = 0
    Dim lapno As Integer

    ' synchronize length for getting track's length
    SyncLock coordinate_Lock
        tracklength = coordinate(10, 0)
    End SyncLock

    SyncLock lap_Lock
        lapno = Me.lap
    End SyncLock

    SyncLock distance_Lock
        ' calculate distance
        dist = Me.distance - ((lapno - 1) * tracklength)
    End SyncLock


    SyncLock coordinate_Lock

        For k As Integer = 0 To 9 ' i have 11 elements into array
            If dist > coordinate(k, 0) And dist < coordinate(k + 1, 0)
            Then

                position = k
                Me.update_position(coordinate(k, 1), coordinate(k, 2))
            End If
        Next

        If dist > tracklength Then
            SyncLock lap_Lock
                Me.lap = Me.lap + 1
                'Me.update_position(coordinate(0, 1), coordinate(0, 2))
            End SyncLock
            Me.update_position(coordinate(0, 1), coordinate(0, 2))
        End If

    End SyncLock
End Sub
'
' Routine to update vehicles position when vts is activated
'
Private Sub update_position(ByVal xcoor As Integer, _
                           ByVal ycoor As Integer)

    ' See if we need to cross threads
    ' If so, have the UI thread call this method for us
    If Me.InvokeRequired Then
        Try
            Me.Invoke(New update position delegate(AddressOf update position),
                      New Object() {xcoor, ycoor})

        Catch ex As Exception

        End Try
    Else
        Me.GaugeContainer2.Images("Image2").Visible = True
        Me.GaugeContainer2.Images("Image2").Location.X = xcoor
        Me.GaugeContainer2.Images("Image2").Location.Y = ycoor

    End If
End Sub
```

Table 1.7: Implemented code for the updating of the vehicle's position.

This page was intentionally left blank.

CHAPTER 2

The Sound Warning System

2.1 Sound warning functionality

The pedestrian sound warning functionality is an innovative idea that was born in Intelligent Systems & Robotics Laboratory (IS&RL) of Technical University of Crete considering the European and international rules of adding sounds in electric cars for the warning of the pedestrians and the bicyclists. The specific functionality helped Technical University of Crete Eco Racing team to win the ADAC safety award in the annual competition Shell Eco Marathon the year 2011.

The interactive graphical user interface provides the ability to the user- driver to activate-deactivate the sound warning functionality, by touching the corresponding button. This functionality is also executed by the software application in a different thread. Figure 2.1 shows the flowchart for the push button event and Figure 2.2 shows the flowchart for the thread that executes the sound warning functionality.

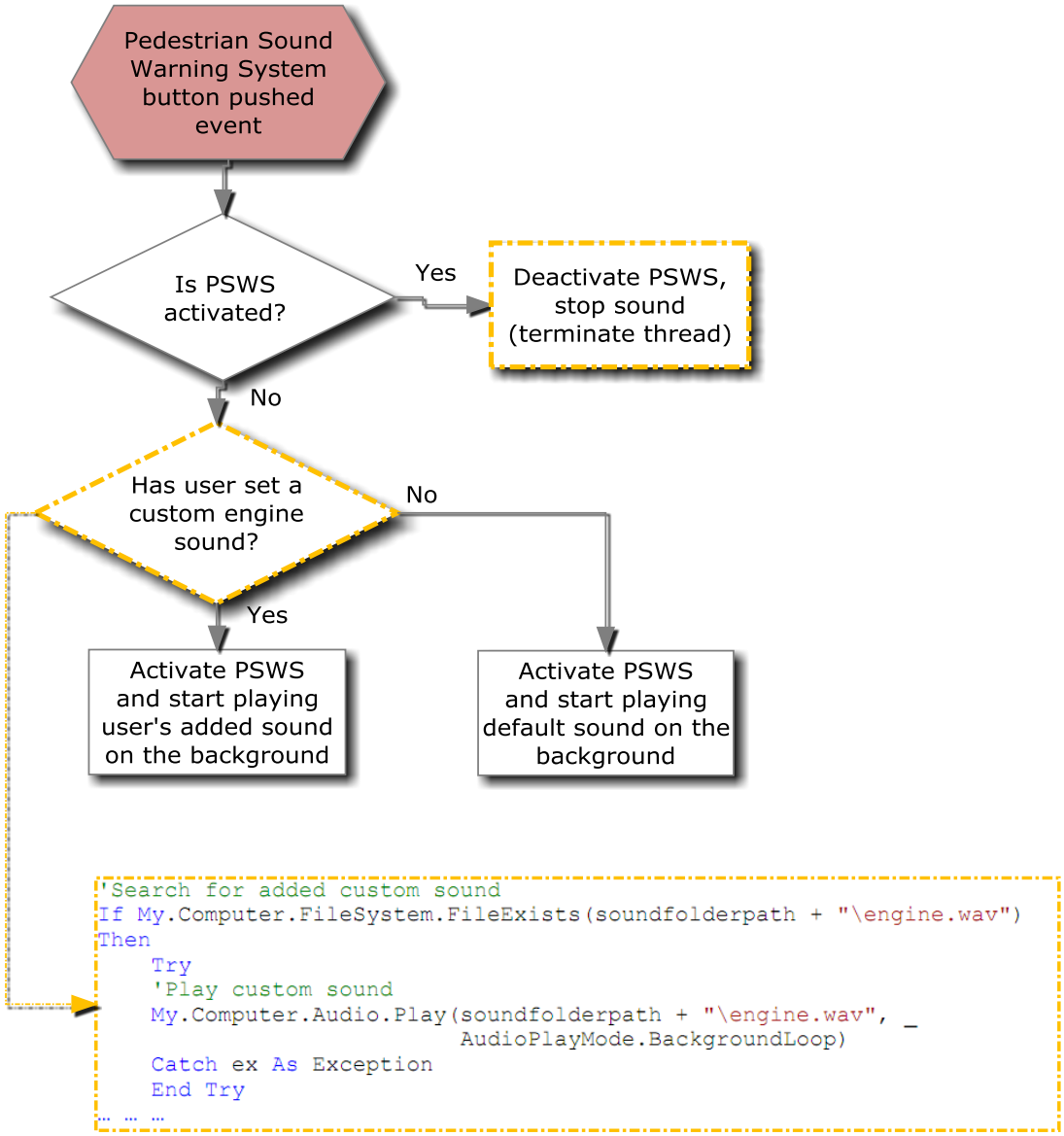

Figure 2.1: Flowchart for the PSWS button pushed event.

When the user-driver pushes the pedestrian sound warning button the sequence of the processes that executed is as follows:

- Checking for the state of the sound warning functionality. If sound warning is activated then terminate thread. If not, continue.
- Checking for the existence of a custom added sound.
- Start thread for the sound warning functionality playing the appropriate sound.

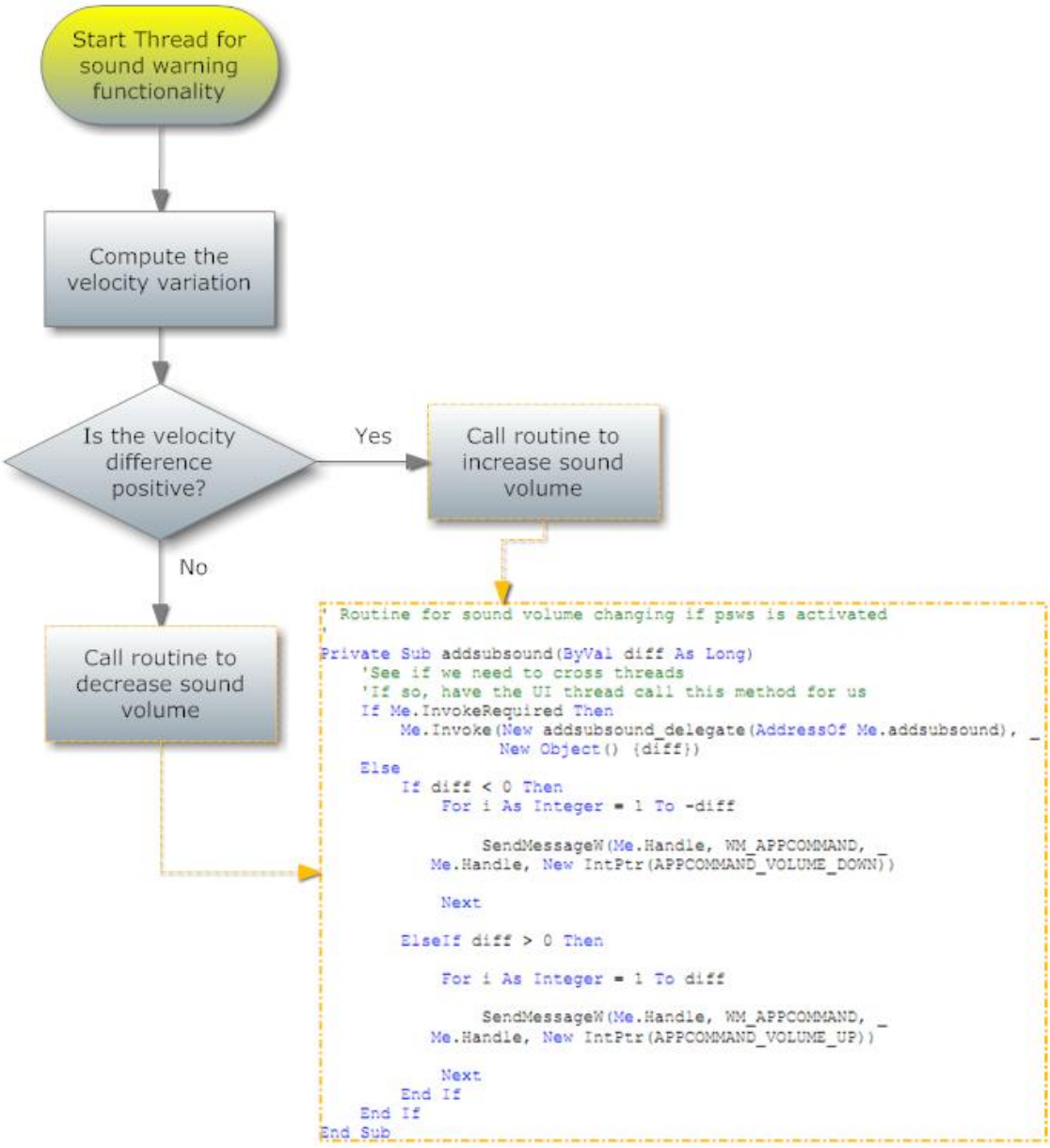

Figure 2.2: Flowchart for the thread that executes the sound warning functionality.

```
Private Sub check psws system(ByVal vel As Double)

        Dim psws_check As String = "0"
        Dim subb As Double = 0
        Dim check previous psws velocity As Double = 0

        SyncLock psws_Lock
            psws_check = Me.psws
        End SyncLock

        If psws_check = "on" Then

            'SyncLock velocity_Lock
            check_previous_psws_velocity = Me.psws_previous_velocity
            Me.psws_previous_velocity = vel
            'End SyncLock

            subb = Math.Round(vel - check_previous_psws_velocity)

            If subb > 0 Then
```

```
                Dim thread_sound As Thread = New Thread(AddressOf Me.addsubsound)
                thread_sound.IsBackground = True
                thread_sound.Name = "Change sound volume thread"
                thread_sound.Start(subb)

            ElseIf subb < 0 And check_previous_psws_velocity > 25 Then
                If vel <= 25 Then
                    Dim thread_sound As Thread = New Thread(AddressOf Me.addsubsound)
                    thread_sound.IsBackground = True
                    thread_sound.Name = "Change sound volume thread"
                    thread_sound.Start(vel - 25)
                Else
                    ' do nothing
                End If

            ElseIf subb < 0 And check_previous_psws_velocity <= 25 Then
                Dim thread_sound As Thread = New Thread(AddressOf Me.addsubsound)
                thread_sound.IsBackground = True
                thread_sound.Name = "Change sound volume thread"
                thread_sound.Start(subb)

            End If
        End If
    End Sub
```

Table2.1: Implemented code for the start of PSWS thread.

The software first samples the odometer response through the microcontroller and then computes the vehicle's current velocity. Then according to the vehicle's current velocity the software increase or decrease the volume of the sound that produced by the speakers.

This page was intentionally left blank.

Bibliography

[1] Michael Halvorson, "*Microsoft Visual Basic 2008 Step by Step"*, Microsoft.
[2] James Foxal, "*Teach Yourself Visual Basic 2008 in 24 Hours: Complete Starter Kit"*, Sams.
[3] Bill Sempf, "*Visual Basic 2008 For Dummies*", For Dummies.
[4] Evangelos Petroutsos, "*Mastering Microsoft Visual Basic 2008*", Wiley.
[5] Christian Gross, "*Beginning VB 2008, From Novice to Professional*", Apress.
[6] StudyVB.com, "*Visual Basic 2008 9.0 .NET Ebook* ", Online Book. Available at: http://www.studyvb.com/.
[7] Home & Learn, "*Visual Basic .NET Programming for Beginners*", Online book. Available at: http://www.homeandlearn.co.uk/net/vbNet.html
[8] Jan Axelson, "*Serial Port Complete 2nd Edition*", Lakeview Research.
[9] Joe Pardue, "*Virtual Serial Port Cookbook*", Smiley Micros.
[10] Innovatik.dk, "*Serial Com Port Communication*", Online Book. Available at: http://www.innovatic.dk/knowledg/SerialCOM/SerialCOM.htm.
[11] Alan Dennis, "*.NET Multithreading"* Manning Publications Co.
[12] Jim Beveridge, Robert Wiener, "*Multithreading Applications in Win 32"*, Addison-Wesley.
[13] Joe Duffy, "*Concurrent Programming on Windows*", Addison-Wesley.
[14] Gaston Hillar, "*C# 2008 and 2005 Threaded Programming: Beginner's Guide*", Packt.
[15] Tobin Titus , Fabio Claudio Ferracchiati , Srinivasa Sivakumar , Kourosh Ardestani , Tejaswi Redkar , Sandra Gopikrishna, "*Visual Basic .NET Threading Handbook*", Wrox.
[16] Joseph Albahari, "*Threading in C#*", Online Book, Available at: http://www.albahari.com/threading/.
[17] ooPIC.com, "*ooPIC Manual & Technical Specifications*", www.oopic.com.
[18] Massimo Banzi, "*Getting Started with Arduino*", O'Reilly.
[19] Simon Monk, "*30 Arduino Projects for the Evil Genius*", Mc Graw Hill.
[20] Michael McRoberts, "*Beginning Arduino*", Apress.
[21] Arduino Main Site, www.arduino.cc
[22] Heliocentris, "*Nexa™ Power Module User's Manual*". Available at: http://faculty.stut.edu.tw/~wcchang/MAN5100078.pdf
[23] Future Electronics, "AccuStar® *Electronic Clinometer Data Sheet*". Available at: http://www1.futureelectronics.com/doc/MEASUREMENT%20SPECIALTIES/02110102-000.pdf
[24] Analog Devices, "*ADXL203 dual-axis accelerometer Data Sheet*". Available at: http://www.analog.com/static/imported-files/data_sheets/ADXL103_203.pdf
[25] Isle, "*Fuel Cell Converter BSZ-PG 1200 Technical Description*". Available at: http://www.isle-ilmenau.de/english/gmbh/produkte/bsz/Te_De_BSZ_PG_02a.pdf
[26] Via Embedded, "*VIA EPIA-P700-10 Pico-ITX Data Sheet*". Available at: http://www.viaembedded.com/en/products/boards/690/1/EPIA-P700%20%28EOL%29.html
[27] Via Embedded, "*VIA EPIA-P830 Pico-ITX Data Sheet*". Available At: http://www.viaembedded.com/en/products/boards/1310/1/EPIA-P830.html
[28] CARTFT, "*CTF700 V2 - VGA 7" TFT - Touchscreen USB Data Sheet*". Available at: http://www.cartft.com/catalog/il/541
[29] ARC Electronics, RS232 *Data Interface. "A Tutorial on Data Interface and cables*". Available at: http://www.arcelect.com/rs232.htm
[30] Volunteers and Editors at Wikibooks.org, "*Wikibooks Serial Programming*". Available at: http://upload.wikimedia.org/wikipedia/commons/1/1f/Serial_Programming.pdf.

[31] Wikibooks, "*Serial Programming/RS-232 Connections*". Online book. Available at: http://en.wikibooks.org/wiki/Serial_Programming:RS-232_Connections.
[32] Jeremy Blum, "*Serial Communication and Processing: Tutorial 6 for Arduino*". Available at: http://www.jeremyblum.com/2011/02/07/arduino-tutorial-6-serial-communication-and-processing/.
[33] Society of Robots, "*Microcontroller Uart Tutorial*". Available at: http://www.societyofrobots.com/microcontroller_uart.shtml.
[34] Microcontroller Board, "*Pic Serial Communication Tutorial*". Available at: http://www.microcontrollerboard.com/pic_serial_communication.html.
[35] Wikipedia, "*Serial Communication*", Online Article. Available at: http://en.wikipedia.org/wiki/Serial_communication.
[36] Wikipedia, "*Serial Port*", Online Article. Available at: http://en.wikipedia.org/wiki/Serial_port#Pinouts.
[37] Wikipedia, "*Serial Cable*", Online Article. Available at: http://en.wikipedia.org/wiki/Serial_cable.
[38] Wikipedia, "*RS-232*", Online Article. Available at: http://en.wikipedia.org/wiki/RS-232.
[39] Tronixstuff, "*Arduino timing methods*", Online Tutorial. Available at: http://tronixstuff.wordpress.com/2011/06/22/tutorial-arduino-timing-methods-with-millis/.
[40] Gonium, "*Handling External Interrupts with Arduino*", Online Article. Available at: http://gonium.net/md/2006/12/20/handling-external-interrupts-with-arduino/.
[41] MSDN, "*Memory Leak Detection and Isolation*". Available at: http://msdn.microsoft.com/en-us/library/x98tx3cf%28v=vs.90%29.aspx.
[42] Wikibooks, "*Visual Basic/Effective Programming*", Online Book. Available at: http://en.wikibooks.org/wiki/Visual_Basic/Effective_Programming.
[43] Adam Wehmann, "*Visual Basic Array Tutorial*", Online book. Available at: http://patorjk.com/programming/tutorials/vbarrays.htm.
[44] Dundas, "*Dundas gauge for Windows Forms*", Online Documentation. Available at: http://support2.dundas.com/OnlineDocumentation/WinGauge2005/webframe.html.
[45] MSDN, "*Managed threading Best Practices*", Online Article. Available at: http://msdn.microsoft.com/en-us/library/1c9txz50%28v=vs.90%29.aspx.
[46] MSDN, *"How to: Make Thread-Safe Calls to Windows Forms Controls*", Online Article. Available at: http://msdn.microsoft.com/en-us/library/ms171728%28v=vs.90%29.aspx.
[47] MSDN, "*How to: Declare Events That Avoid Blocking*", Online Article. Available at: http://msdn.microsoft.com/en-us/library/wf33s4w7%28v=vs.90%29.aspx.
[48] MSDN, "*Thread Synchronization (C# and Visual Basic)"*, Online Article. Available at: http://msdn.microsoft.com/en-us/library/ms173179%28v=vs.100%29.aspx#Y247.
[49] MSDN, "*Managed and Unmanaged Threading*", Online Article. Available at: http://msdn.microsoft.com/en-us/library/5s8ee185%28v=vs.90%29.aspx.
[50] Tedd Pattison, "*Thread Synchronization*", MSDN Magazine Article. Available at: http://msdn.microsoft.com/en-us/magazine/cc163929.aspx.
[51] MSDN, "*Thread Synchronization"*, Online Article. Available at: http://msdn.microsoft.com/en-us/library/dsw9f9ts%28v=vs.90%29.aspx.
[52] Fabrice Marguerie, "*How to detect and avoid memory and resources leaks in .NET applications*", Online Article. Available at: http://madgeek.com/Articles/Leaks/Leaks.en.html.
[53] Ian Griffith*, "Doing Work Without Threads*", Online article. Available at: http://www.interact-sw.co.uk/iangblog/2004/09/23/threadless.

[54] Ian Griffith, "*How To Stop a Thread in .NET (and Why Thread.Abort is Evil)*", Online Article. Available at: http://www.interact-sw.co.uk/iangblog/2004/11/12/cancellation.
[55] Diranieh.com, "*Multithreading Basics*", Online Article. Available at: http://www.diranieh.com/NETThreading/MultithreadingBasics.htm.
[56] Dan Mabbutt, "*Disposing Objects*", Online Article. Available at: http://visualbasic.about.com/od/usingvbnet/a/disposeobj.htm.
[57] Dan Mabbutt, "*Coding New Instances of Objects*", Online Article. Available at: http://visualbasic.about.com/od/usingvbnet/a/newinst.htm.
[58]Ed Ball, "*Thread Safe Disposable Objects*", Online Article. Available at: http://code.logos.com/blog/2008/03/threadsafe_disposable_objects.html.
[59] Juval Lowy, "*Asynchronous Calls in .Net*", Online Article. Available at: http://www.code-magazine.com/Article.aspx?quickid=0305071.
[60] Suprotim Agarwal, "*Programmatically Increase, Decrease and Mute the Volume*", Online Article. Available at: http://www.dotnetcurry.com/ShowArticle.aspx?ID=431.
[61] John Spano, "*Delegates in .NET*", Online Article. Available at: http://www.developerfusion.com/article/5251/delegates-in-vbnet/.
[62] MSDN*," Interoperating with Unmanaged Code*", Online Article. Available at: http://msdn.microsoft.com/en-us/library/sd10k43k%28v=VS.85%29.aspx.
[63] MSDN, "*An Overview of Managed/Unmanaged Code Interoperability*". Available at: http://msdn.microsoft.com/en-us/library/ms973872.aspx.
[64] MSDN, "*How to: Declare Events That Conserve Memory Use*". Available at: http://msdn.microsoft.com/en-us/library/yt1k2w4e%28v=vs.90%29.aspx.
[65] Ted Pattison, "*Asynchronous Method Execution Using Delegates*", MSDN Magazine Article. Available at: http://msdn.microsoft.com/en-us/magazine/cc164036.aspx.
[66] Richard Grimes," *.NET Delegates: Making Asynchronous Method Calls in the .NET Environment*", MSDN Magazine Article. Available at: http://msdn.microsoft.com/en-us/magazine/cc301332.aspx.
[67] Ted Pattison, "*Asynchronous Method Execution Using Delegates*", MSDN Magazine Article. Available at: http://msdn.microsoft.com/el-gr/magazine/cc164036%28en-us%29.aspx.
[68] MSDN*,"Processes, Threads, and Apartments*", Online Article. Available at: http://msdn.microsoft.com/en-us/library/ms693344.
[69] Ted Pattison, "*Creating and Managing Secondary Threads*", MSDN Magazine Article. Available at: http://msdn.microsoft.com/en-us/magazine/cc188722.aspx.
[70] David Carmona, "*Programming the Thread Pool in the .NET Framework*", MSDN Online Article. Available at: http://msdn.microsoft.com/en-us/library/ms973903.aspx.
[71] Microsoft Corporation, "*Developing Applications for an Auto PC*", MSDN Online Article. Available at: http://msdn.microsoft.com/en-us/library/ms834174.aspx#apcapps_topic2.
[72] Robert Burns, "*Multithreaded Programming with Visual Basic .NET*", MSDN Online Article. Availablea at: http://msdn.microsoft.com/en-us/library/aa289496%28v=vs.71%29.aspx.
[73] Alex Calvo*," Comparing the Timer Classes in the .NET Framework Class Library*", MSDN Magazine Online Article. Available at: http://msdn.microsoft.com/en-us/magazine/cc164015.aspx.
[74] MSDN, "*Managed Threading Best Practices*", MSDN Online Article. Available at: http://msdn.microsoft.com/en-us/library/1c9txz50%28v=VS.90%29.aspx.
[75] MSDN, "*How to: Make Thread-Safe Calls to Windows Forms Controls*", MSDN Online Article. Available at: http://msdn.microsoft.com/en-us/library/ms171728%28v=VS.90%29.aspx.

[76] MSDN, "*Calling Synchronous Methods Asynchronously*", MSDN Online Article. Available at: http://msdn.microsoft.com/en-us/library/2e08f6yc%28v=VS.90%29.aspx.
[77] MSDN, "*Multithreading in Visual Basic*", MSDN Online Article. Available at: http://msdn.microsoft.com/en-us/library/eed6swsx%28v=VS.90%29.aspx.

Author: Anastasios K. Petrou (K. = Konstantinos)

E-mail: petroutassos@gmail.com , apetrou@isc.tuc.gr

This page was intentionally left blank.

Appendix A

Serial communication

The software developed in this master thesis is primarily based on serial communication between a computer unit, a fuel cell unit and an object oriented programmable intergraded circuit (OOpic microcontroller). So, this chapter is an introduction to serial communication and a clarification to the concepts that rule serial communication.

Serial Communication

In computer science, serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or a computer bus (*Figure A.1*). This is in contrast to parallel communication, where several bits are sent as a whole, on a link with several parallel channels (*Figure A.2*). Serial communication is used for long distance communications and for computer networking, where the cabling cost and synchronization difficulties make parallel communication impractical.

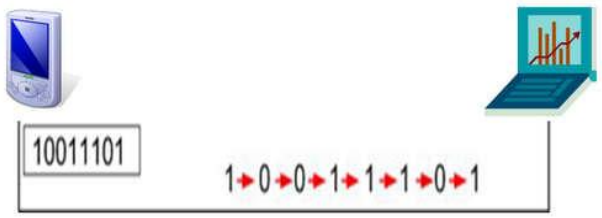

Figure A.1: Serial Communication.

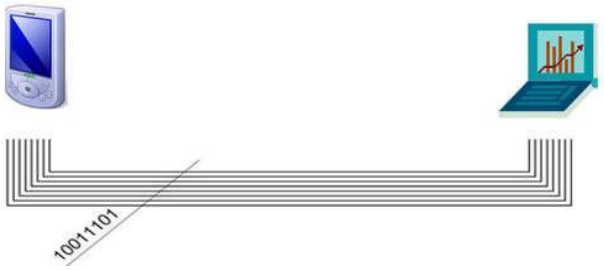

Figure A.2: Parallel Communication.

Serial computer buses or serial cables are becoming more common even at shorter distances, as improved signal integrity and transmission speeds in newer serial technologies have begun to outweigh the parallel bus's advantage of simplicity (no need for serialization and deserialization) and to outstrip its disadvantages (clock skew, interconnect density).

Architectures

Examples of serial communication architectures are among others:

- Morse code telegraphy
- RS-232 (low-speed, implemented by serial ports)
- RS-422
- RS-423

- RS-485
- I²C
- SPI
- ARINC 818 Avionics Digital Video Bus
- Universal Serial Bus (moderate-speed, for connecting peripherals to computers)
- FireWire
- Ethernet
- Fibre Channel (high-speed, for connecting computers to mass storage devices)
- InfiniBand (very high speed, broadly comparable in scope to PCI)
- MIDI control of electronic musical instruments
- DMX512 control of theatrical lighting
- SDI-12 industrial sensor protocol
- Serial Attached SCSI
- Serial ATA
- SpaceWire Spacecraft communication network
- HyperTranspot
- PCI Express
- SONET and SDH (high speed telecommunication over optical fibers)
- T-1, E-1 and variants (high speed telecommunication over copper pairs)
- MIL-STD-1553A/B

Serial buses

A serial bus or serial cable is a cable that can be used to transfer information between two devices using serial communication. The form of connectors depends on the particular serial port used. A serial cable for connecting two data terminal equipments directly is known as a null modem cable (*Figure A.3*).

Figure A.3: Null modem cable.

The maximum working length of a cable varies depending on the characteristics of the transmitters and receivers, the baud rate on the cable, and the capacitance and resistance of the cable. The RS-232 standard states that a compliant port must provide defined signal characteristics for a capacitive load of 2500 pF. This does not correspond to a fixed length of cable since varying cables have different characteristics. Empirically tested combinations of bit rate, serial ports, cable type, and lengths may provide reliable communications, but generally RS-232 compatible ports are intended to be connected by at the most a few tens of meters of cable. Other serial communications standards are better adapted to drive hundreds or thousands of meters of cable.

Figure A.4: Serial Cable, typically used for RS-232 serial communication

RS-232 standard

RS-232 (Recommended Standard 232) is the traditional name for a series of standards for serial binary single-ended data and control signals connecting between a DTE (Data Terminal Equipment) and a DCE (Data Circuit-terminating Equipment). It is commonly used in computer serial ports. The standard defines the electrical characteristics and timing of signals, the meaning of signals, and the physical size and pin out of connectors. The current version of the standard is *TIA-232-F Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange,* issued in 1997.

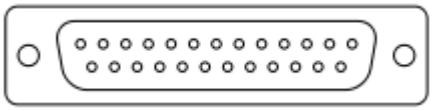

Figure A.5: A 25 pin connector as described in the RS-232 standard.

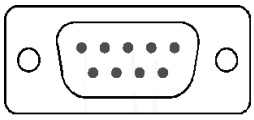

Figure A.6: A 9 pin connector for the RS-232 standard.

An RS-232 port was once a standard feature of a personal computer for connections to modems, printers, mice, data storage, un-interruptible power supplies, and other peripheral devices. However, the limited transmission speed, relatively large voltage swing, and large standard connectors motivated development of the universal serial bus which has displaced RS-232 from most of its peripheral interface roles. Many modern personal computers have no RS-232 ports and must use an external converter to connect to older peripherals. Some RS-232 devices are still found especially in industrial machines or scientific instruments.

Scope of the RS-232 standard

The Electronic Industries Association (EIA) standard RS-232-C defines:

- Electrical signal characteristics such as voltage levels, signaling rate, timing and slew-rate of signals, voltage withstand level, short-circuit behavior, and maximum load capacitance.
- Interface mechanical characteristics, pluggable connectors and pin identification.
- Functions of each circuit in the interface connector.
- Standard subsets of interface circuits for selected telecom applications.

The standard does not define such elements as the character encoding or the framing of characters, or error detection protocols. The standard does not define bit rates for transmission, except that it says it is intended for bit rates lower than 20,000 bits per second. Many modern devices support speeds of 115,200 bit/s and above. RS-232 makes no provision for power to peripheral devices.

Details of character format and transmission bit rate are controlled by the serial port hardware, often a single integrated circuit called a UART that converts data from parallel to asynchronous start-stop serial form. Details of voltage levels, slew rate, and short-circuit behavior are typically controlled by a line driver that converts from the UART's logic levels to RS-232 compatible signal levels, and a receiver that converts from RS-232 compatible signal levels to the UART's logic levels.

History of the RS-232 standard

RS-232 was first introduced in 1962. The original DTEs were electromechanical teletypewriters, and the original DCEs were usually modems. When electronic terminals began to be used, they were often designed to be interchangeable with teletypewriters, and so supported RS-232. The C revision of the standard was issued in 1969 in part to accommodate the electrical characteristics of these devices.

Since application to devices such as computers, printers, test instruments, and so on was not considered by the standard, designers implementing an RS-232 compatible interface on their equipment often interpreted the requirements idiosyncratically. Common problems were non-standard pin assignment of circuits on connectors, and incorrect or missing control signals. The lack of adherence to the standards produced a thriving industry of breakout boxes, patch boxes, test equipment, books, and other aids for the connection of disparate equipment. A common deviation from the standard was to drive the signals at a reduced voltage. Some manufacturers therefore built transmitters that supplied +5 V and -5 V and labeled them as "RS-232 compatible".

Later personal computers started to make use of the standard so that they could connect to existing equipment. For many years, an RS-232-compatible port was a standard feature for serial communications, such as modem connections, on many computers. It remained in widespread use into the late 1990s. In personal computer peripherals, it has largely been supplanted by other interface standards, such as USB. RS-232 is still used to connect older designs of peripherals, industrial equipment, console ports, and special purpose equipment, such as a cash drawer for a cash register.

The standard has been renamed several times during its history as the sponsoring organization changed its name, and has been variously known as EIA RS-232, EIA 232, and most recently as TIA 232. The standard continued to be revised and updated by the Electronic Industries Alliance and since 1988 by the Telecommunications Industry Association (TIA). Revision C was issued in a document dated August 1969. Revision D was issued in 1986. The current revision is *TIA-232-F Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange*, issued in 1997.

Limitations of the RS-232 standard

Because the application of RS-232 has extended far beyond the original purpose of interconnecting a terminal with a modem, successor standards have been developed to address the limitations. Some of the RS-232 standard's limitations are:

- The large voltage swings and requirement for positive and negative supplies increases power consumption of the interface and complicates power supply design. The voltage swing requirement also limits the upper speed of a compatible interface.
- Single-ended signaling referred to a common signal ground limits the noise immunity and transmission distance.
- Multi-drop connection among more than two devices is not defined. While multi-drop "work-arounds" have been devised, they have limitations in speed and compatibility.
- Asymmetrical definitions of the two ends of the link make the assignment of the role of a newly developed device problematic. The designer must decide on either a DTE-like or DCE-like interface to use and which connector pin assignments.
- The handshaking and control lines of the interface are intended for the setup and takedown of a dial-up communication circuit. In particular, the use of handshake lines for flow control is not reliably implemented in many devices.
- No method is specified for sending power to a device. While a small amount of current can be extracted from the DTR and RTS lines, this is only suitable for low power devices such as mice.
- The 25-way connector recommended in the standard is large compared to current practice.

RS-232 standard details

In RS-232, data is sent as a time-series of bits. Both synchronous and asynchronous transmissions are supported by the standard. In addition to the data circuits, the standard defines a number of control circuits used to manage the connection between the DTE and DCE. Each data or control circuit only operates in one direction that is, signaling from a DTE to the attached DCE or the reverse. Since transmit data and receive data are separate circuits, the interface can operate in a full duplex manner, supporting concurrent data flow in both directions. The standard does not define character framing within the data stream, or character encoding.

Voltage levels

The RS-232 standard defines the voltage levels that correspond to logical one and logical zero levels for the data transmission and the control signal lines. Valid signals are plus or minus 3 to 15 volts; the ±3 V range near zero volts is not a valid RS-232 level. The standard specifies a maximum open-circuit voltage of 25 volts: signal levels of ±5 V, ±10 V, ±12 V, and ±15 V are all

commonly seen depending on the power supplies available within a device. RS-232 drivers and receivers must be able to withstand indefinite short circuit to ground or to any voltage level up to ±25 volts. The slew rate, or how fast the signal changes between levels, is also controlled.

For data transmission lines (TxD, RxD) logic one is defined as a negative voltage, the signal condition is called marking, and has the functional significance. Logic zero is positive and the signal condition is termed spacing. Control signals are logically inverted with respect to what one sees on the data transmission lines. When one of these signals is active, the voltage on the line will be between +3 to +15 volts. The inactive state for these signals is the opposite voltage condition, between −3 and −15 volts. Examples of control lines include request to send (RTS), clear to send (CTS), data terminal ready (DTR), and data set ready (DSR).

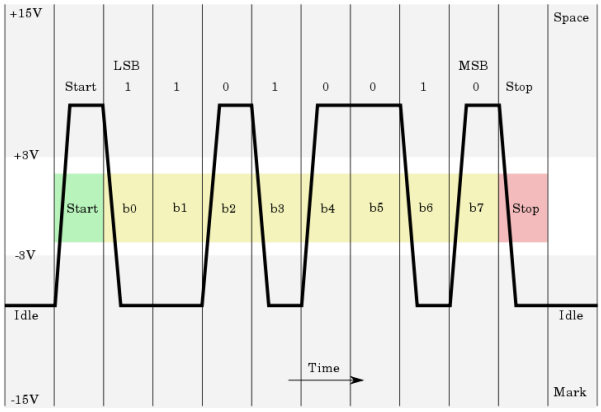

Figure A.7: Diagrammatic oscilloscope trace of voltage levels for an uppercase ASCII "K" character (0x4b) with 1 start bit, 8 data bits and 1 stop bit.

Because the voltage levels are higher than logic levels typically used by integrated circuits, special intervening driver circuits are required to translate logic levels. These also protect the device's internal circuitry from short circuits or transients that may appear on the RS-232 interface, and provide sufficient current to comply with the slew rate requirements for data transmission.

Because both ends of the RS-232 circuit depend on the ground pin being zero volts, problems will occur when connecting machinery and computers where the voltage between the ground pin on one end and the ground pin on the other end is not zero. This may also cause a hazardous ground loop. Use of a common ground limits RS-232 to applications with relatively short cables. If the two devices are far enough apart or on separate power systems, the local ground connections at either end of the cable will have differing voltages; this difference will reduce the noise margin of the signals. Balanced, differential, serial connections such as USB, RS-422 and RS-485 can tolerate larger ground voltage differences because of the differential signaling.

Unused interface signals terminated to ground will have an undefined logic state. Where it is necessary to permanently set a control signal to a defined state, it must be connected to a voltage source that asserts the logic 1 or logic 0 level. Some devices provide test voltages on their interface connectors for this purpose.

Connectors

RS-232 devices may be classified as Data Terminal Equipment (DTE) or Data Communication Equipment (DCE). This defines at each device which wires will be sending and receiving each signal. The standard recommended but did not make mandatory the D-subminiature 25 pin connector (Figure A.8). In general and according to the standard, terminals and computers have male connectors with DTE pin functions, and modems have female connectors with DCE pin functions. Other devices may have any combination of connector gender and pin definitions. Many terminals were manufactured with female terminals but were sold with a cable with male connectors at each end; the terminal with its cable satisfied the recommendations in the standard.

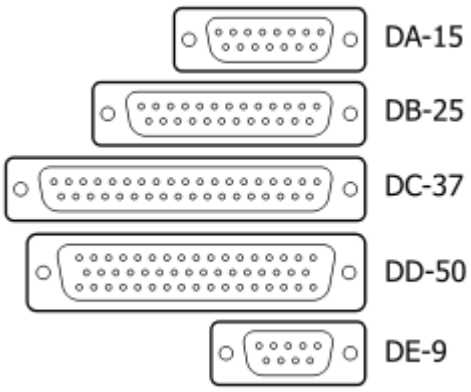

Figure A.8: DA, DB, DC, DD, and DE RS-232 standard sized connectors.

The standard specifies 20 different signal connections. Since most devices use only a few signals, smaller connectors can often be used.

Cables

The standard does not define a maximum cable length but instead defines the maximum capacitance that a compliant drive circuit must tolerate. A widely used rule of thumb indicates that cables more than 50 feet (15 m) long will have too much capacitance, unless special cables are used. By using low-capacitance cables, full speed communication can be maintained over larger distances up to about 1,000 feet (300 m). For longer distances, other signal standards are better suited to maintain high speed.

Conventions

For functional communication through a serial port interface, conventions of bit rate, character framing, communications protocol, character encoding, data compression, and error detection, not defined in RS-232 standard, must be agreed to by both sending and receiving equipment.

This page was intentionally left blank.

Appendix B

Multithreading

Since the software developed for the purposes of the specific thesis is a multithreading application, this brief chapter is an introduction to the concept and the idea of multithreading.

Introduction

Our brain allows us to multitask. For example, we can prepare dinner while talking on the telephone. However, this multitasking has limits, and we can do only two or three things simultaneously. But suppose we could put down the work, start another piece of work, then put that down, and then switch to the original work. By this way probably we could handle a few hundred tasks at the same time. This method called multitasking serialization.

Now suppose you and another person are preparing dinner in the kitchen, but you are not communicating with each other. In that case the likelihood that you will run into the other person is pretty high. So, there is a big difference between multitasking with a single brain and multitasking with multiple brains. Multitasking always has a cost, which is orchestration. And sometimes doing more multitasking is not going to speed things up. There is a limit to how many brains are required to run an efficient kitchen.

In general is more difficult to make code run efficiently in parallel. Race conditions and deadlocks are the two common problems in multitasking applications.

Clarifications

Program

A program is typically defined as a series of instructions that are related in some way. In .NET terms, a program can be defined as an assembly, or group of assemblies, that work together to accomplish a task. Assemblies are nothing more than a way of packaging instructions into maintainable elements. An assembly is generally housed in a dynamic link library (DLL) or an executable.

Closely related to programs are processes and threads. A program's execution occurs on one or more threads contained with a process. Threads allow the operating system to exert control over processes and the threads that execute within.

Process

A process gives a program a place to live, allowing access to memory and resources. A process is an operating system's object used to associate one or more paths of execution with required resources, such as memory that stores values manipulated by threads that exist within the process.

A process provides a level of isolation that keeps different applications from inadvertently interacting with each other. It is like cans with several different colors of paint. While each color of paint is in its own can it cannot mix with other paints. The can is similar to a process in that it keeps things in the can contained within and things outside of the can out. Every process

contains one or more threads. You can think of a thread as the moving part of the process. Without a thread interacting with elements within a process, nothing interesting will happen.

Thread

Threads are paths of execution. The threads perform the operations while the process provides the isolation. A single-threaded application has only one thread of execution.

When an executable is launched the operating system creates a process for the executable to run in. The operating system then loads the executable into the process's memory and looks for an entry point, a specially marked place to start carrying out the instructions contained within the executable. This entry point is like the front door of a building. Every building has one, and front doors are relatively easy to find. Generally speaking, it's impossible to get into a restaurant without going through the front door. Once the entry point is identified, a thread is created and associated with the process. The thread is started, executing the code located at the entry point. From that point on the thread follows the series of instructions. This first thread is referred to as the main thread of the process.

Multitasking

When an operating system supports execution of multiple concurrent processes it is said to be multitasking. The operating system used for the development of the specific project, Microsoft windows XP, is a multitasking operating system. There are two common forms of multitasking: preemptive and cooperative. Preemptive multitasking is the one we care about.

Preemptive multitasking

When more than one application is executing, there must be some means of determining whose turn it is to execute. This is generally referred to as scheduling. Scheduling involves an element in one of two states: currently executing and waiting to execute. Under modern operating systems scheduling is performed on a per-thread basis. This allows a single thread to be paused and then resumed. Only one thread can be executing at a given point in time. All other threads are waiting for their turn to execute. This allows the operating system to exert a high degree of control over applications by controlling the execution of their threads.

Time slicing

Time slicing is when an operating system can dictate for how much time a program is allowed to execute. Between the times of execution, the program is in a state of deep freeze and does nothing. The user is not aware of the time slices, because a time slice operates on the order of microseconds. Because time slicing is so fast, user think the program is running continuously.

Let's say that a program that runs two tasks, task 1 and task 2, is executed. The microprocessor is a single core, and thus when running two separate tasks, there will be two time slices (Figure B.1). In the figure, the entire processing cycle is represented as a pie, and each time slice is a slice of the pie.

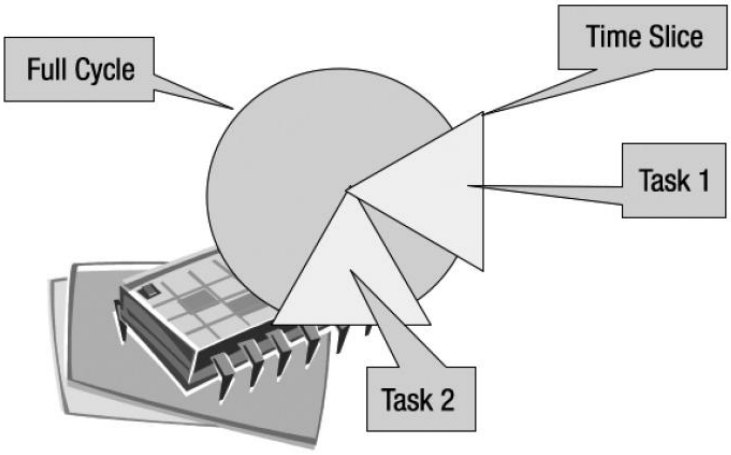

Figure B.1: Single core microprocessor running two tasks.

Task 1 and task 2 run in a serial manner, rather than concurrently. This is because the microprocessor is a single-task device made to look like a multitask device. A programmer must develop an application with multiple tasks, for a single-core microprocessor, when there is the need of application's background tasks not to affect the foreground tasks. The operating system will still allocate time slices in this case and use preemptive multitasking. For the reference, the VIA EPIA-P700 motherboard used for this project is a single 1 GHz core board.

Figure B.2 illustrates how the same application executes on a multiple-core microprocessor. The operating system, in a bid to make more efficient use of the microprocessor, has put one task on one core and another task on the other core. Now both tasks are running in parallel. In that case is possible that both tasks would want to manipulate the same piece of data at the same time. In a single-core microprocessor, that is not physically possible. However, with a single processor, it is still possible for one task to be interrupted mid-flight while accessing some data, and for the other task to step in and use inconsistent data.

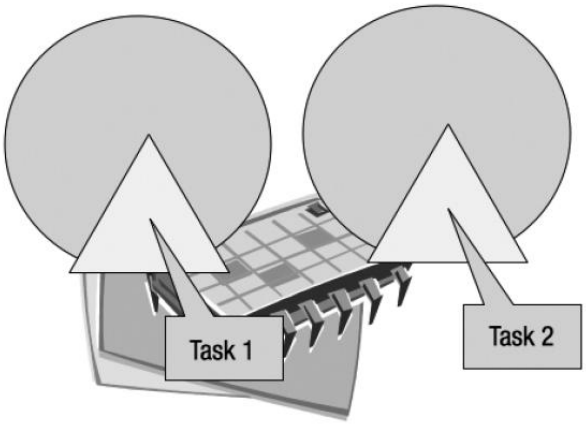

Figure B.2: Multiple-core microprocessor running two tasks.

The End!