03-06-2009

Wireless Terminal for Fuel Cells Report

Authors:
Jeppe Bundgaard, Teddy Roskvist and Carsten Schwartz (AU9109)
Version 0.9

History of this Document

Rev.	Date (dd mm yyyy)	Changed by;	Description of changes (including affected section numbers)	Reviewed by; Date;	Approved by; Date;
0.5	18 05 2009	CSN	Initial disposition of report		
0.6	18 05 2009	CSN	Check up on function calls etc.		
0.7	24 05 2009	CSN	Draft of conclusion		
0.8	25 05 2009	JBC, CSN	Draft of implementation plus minor corrections		
0.9	02 06 2009	CSN	Added that touch screen has been enabled for the menu		

Template rev. R1.0

Contents

1. Introduction

1.1 Purpose

This report shows that the authors are capable of utilizing Windows Embedded in a practical application.

This document gives an overview of the work on the Wireless Terminal for Fuel Cells, from here called WTFCU, through:

- The requirements
- The design
- and the final implementation

The implementation differs from the design due to time constraints, this is described in the conclusion.

1.2 Reading Guidelines

This document is a report based on the project proposal described in [1]

2. Requirement Specification

The initial requirements from the project are described in this section:

A fuel cell has several interesting characteristics for monitoring, each requirement is a description on what the device must be able to monitor.

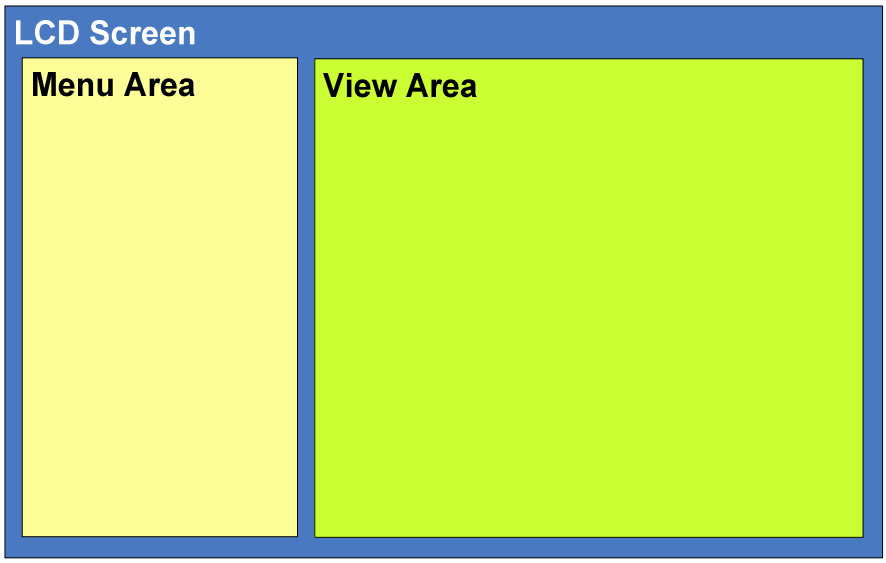
Req. ID.Rev	Description	Purpose
1.1	The device must monitor and display current operational state live	To enable the user to observe live data to ensure accepted operational behaviour
2.1	The device must display statistics on the operational state of the device	The enable the user to depict trends or errors in operational behaviour
3.1	The device must display current stack usage of the real time software operating it	The enable developers of Fuel Cell Controller software to optimize the memory foot print of the software
4.1	The device must be configurable	To enable the user to setup communications with one or more Fuel Cells

3. Overall Design

This section describes the initial design of the WTFCU.

3.1 Graphical Overview

The LCD screen of the wireless terminal is split into a menu area and a view area as shown below.



The LCD screen contains a menu are and a view area. Each menu item will make a specific view appear as the active view.

Menu Item	View	Description
Operational Status	Operational Status View	A view that shows live values of: - Active State - Stack Voltage - Stack Current - Stack Power - Stack Temperature - Bridge Voltage - DC Line Voltage Being updated live
Device Status	Device Status View	A view that shows live values of: - Cathode PWM - Cathode RPM - Main Status - H2 Pressure
Fast Statistics	Stack Statistics View (Fast)	A view that shows graphs with live values of: - Stack Voltage

		- Stack Current - Stack Temperature - Bridge Voltage - DC Line Voltage Update fast (Every scond)
Slow Statistics	Stack Statistics View (Slow)	A view that shows graphs with live values of: - Stack Voltage - Stack Current - Stack Temperature - Bridge Voltage - DC Line Voltage Update Slow (Every 15 minutes)
Memory Usage	Memory Usage View	A view that show the live max value of the stacks for the software task in the fuel cell.
Options/Setup	Options/Setup View	A view to setup: - The wireless communication - Sampling Timing - ??? Etc…
Last Error	Error View	A view that is shown as top view with an error description automatically when an error event is received from the transmission manager. Can be invoked from the menu.

Statistics views will contain as many data as there are pixels in the graphical views, this will be defined during implementation.

The fields that are greyed out are low priority views, and will be implemented if time permits.

3.2 Architecture

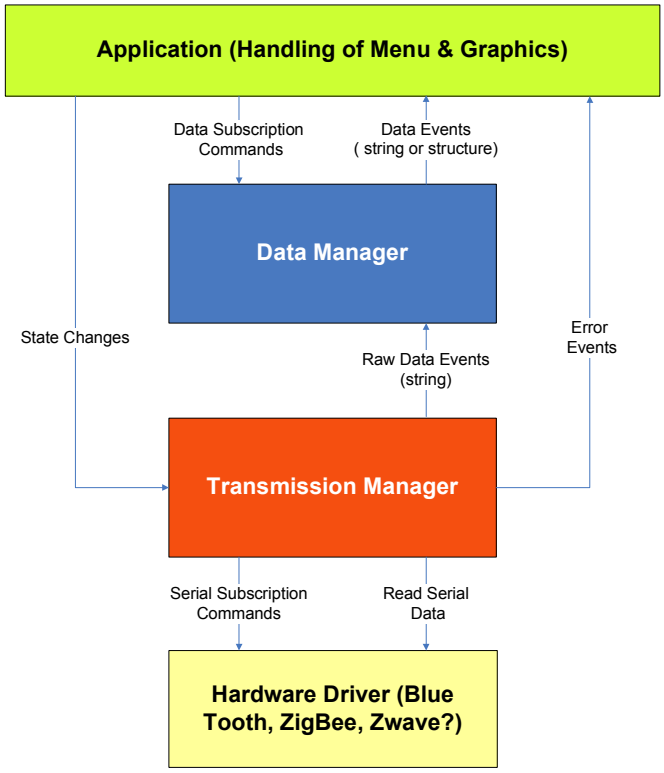
The WTFCU software is layered in modules to encapsulate functionality:

1. Application
    a. Changes the active view via user input through the menu
    b. Handles update of data in the active view and the graphics in this view
    c. Executes view/state change commands

2. Data Handler
    a. Converts raw data to well formatted data
    b. Stores well formatted data and statistics
    c. Sends data events based on subscription commands

3. Transmission Manager
    a. Executes data selection commands
    b. Creates raw data by parsing serial stream data
    c. Sends raw data events
    d. Validates the connection and sends Error events if necessary

4. Hardware Driver
    a. Creates wireless serial connection to a Fuel Cell
    b. Returns wireless serial data
    c. Sends wireless serial commands



4. Interfaces and Classes

The interfaces between the modules are described in this chapter as well as a short view of the class encapsulation of the software.

4.1 Interfaces

ID.Revision	**1.1**
Name	View Change - ChangeViewState()
Purpose	To enable the user to change the active view interactively
Functionality	1. Function calls invoked from the menu shows the active view as the top view 2. When a view is shown all data values are greyed out to indicate that they are old data. The values are shown in black after the first update to the active view
Modules	This interface is invoked by the Application when the user selects specific menu items.

ID.Revision	**2.1**
Name	State Change - ChangeDataStream()
Purpose	To change the content of the received serial data stream
Functionality	1. Function calls sends commands over the wireless connection that orders the Fuel Cell to broadcast a specific data set 2. Once sent the content of the data from the Fuel Cell will be as the command specifies.
Modules	The Application invokes this that is a method in the Transmission Manager in connection with Data Subscription

ID.Revision	**3.1**
Name	Data Subscription – SetupDataSubScription()
Purpose	To order data events for a specific view
Functionality	1. Function calls setup a filter in the Data Manager 2. The filter lets formatted data pass as events to the Application after transformation of raw data
Modules	Invoked by the application as a method on the Application. This function is called from within ChangeViewState(), if the data subscription differs from the serial data set, that is defined by the current state.

ID.Revision	**4.1**
Name	Data Events – Fire_[event type]_Event()
Purpose	To send data that is requested via subscription to the

	Application
Functionality	1. Events are broadcasted via function calls 2. Data is sent as a string or a structure 3. The event is handled by the application to update data in the active view
Modules	Invoked by the Data Manager as response to receival and storing of formatted Raw Data, only invoked if the Data is part of the current data subscription. Handled by the Application to update the active view.

ID.Revision	**5.1**
Name	Raw Data Events – FireRawData()
Purpose	To send raw data for formatting and storage to the Data Manager
Functionality	1. Events are broadcasted via function calls 2. Data is sent as a string 1. The event is handled by the Data Manager and stored as a formatted string for values and in a buffer for statistical data
Modules	Invoked by the Transmission manager when raw data is parsed from serial data.

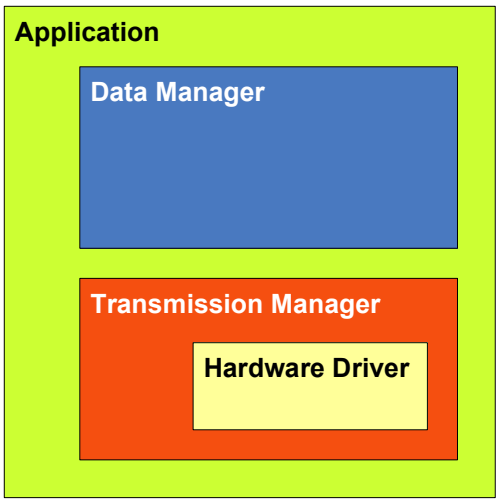
ID.Revision	**6.1**
Name	Read Serial Data – PollDataThread()
Purpose	To read the data the Fuel Cell transmits on the wireless serial connection.
Functionality	1. Data is read via a function call 2. The buffered data is parsed by the transmission manager and passed on as Raw Data 3. If the data is missing or corrupted an error event is sent to the application 4. Implemented as a free running Thread
Modules	Setup and started by Transmission Manager .

ID.Revision	**7.1**
Name	Error Events – FireError()
Purpose	To inform the user that data is corrupt.
Functionality	1. Error events are sent through a function call 2. The Application handles the event by showing an error view 3. The user can fix the problem by setting up the device or correction the error in some other way
Modules	Invoked by the Transmission manager when: 1. Data is corrupt 2. The Wireless connection times out Handled by the Application to present the user with the Error

4.2 Classes

The class names encapsulation in the design is described / shown below

Class Encapsualtion

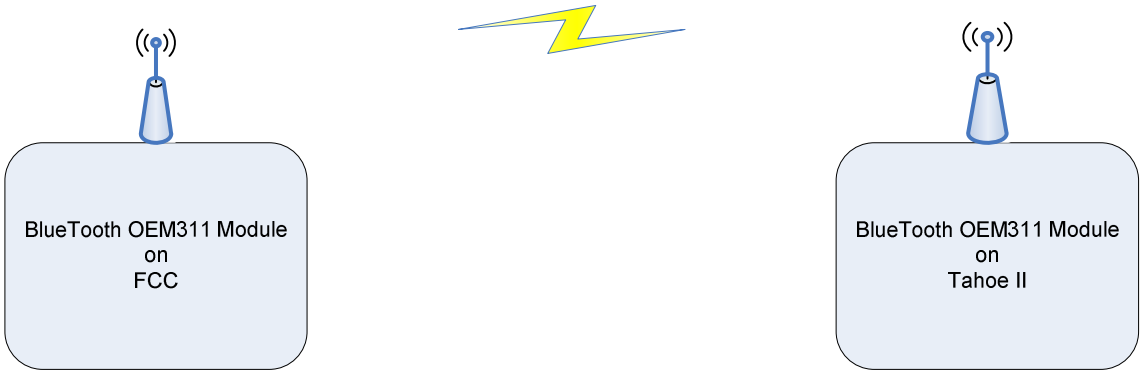


1. The Application creates instances of the Data Manager and the Transmission manager
2. The transmission manager creates and instance of the Hardware Driver

5. Implementation

5.1 Bluetooth

The idea was to have a direct replacement of the RS232 cable, to do that we used two OEM modules from BlueConnect with the Serial Port Profile (SSP) on board.



The modules are setup through some AT commands when the module is in AT mode see [2].

General setup:
Both modules are setup to match the FCC interface:
19200 baud, 8 data bits, no parity, 1 stop bit, No flow control.

The FCC module is setup to be a discoverable Server and to auto accept a specific MAC when discovered (the MAC of the Tahoe II).

The Tahoe II module Is setup to be a discoverable Server and Client with auto search and auto connect to a specific MAC (the FCC).

When both modules are powered on the Tahoe will discover the FCC and connect to it, when the connection is established the module react as if it was a hard wired RS232 interface.

When this setup is performed the Tahoe board together with the FCC act as they are hardwired when they are powered up.
If another device wants to connect to the FCC (e.g. PDA) it is also possible, but the device have to connect to the FCC and used the default password when connecting.

5.2 Files

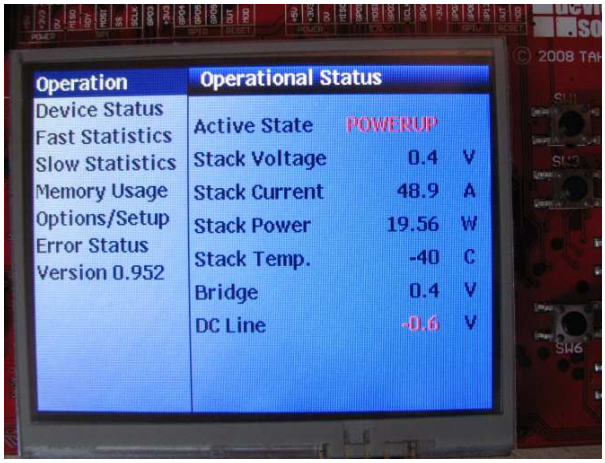
The design is implemented as following the guidelines in the design with one exception. A hardware driver class file was created initially but since the *SerialPort* class is a part of the micro framework, an instance of *SerialPort* was used in the *TransmissionManager Class*. The files we in the project are:

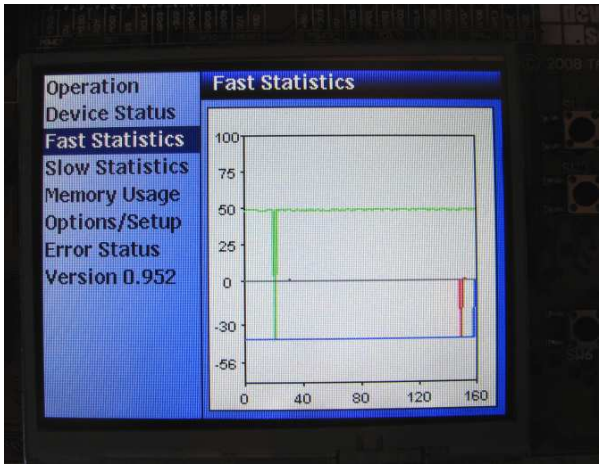
- *WTFCU.cs*, This is the application that has instances of buttonproviders, view classes, a Transmission Manager and a Datamanager.
- *GPIOButtonInputProvider.cs*, This class provides handling of buttons
- *HighlightableListBoxItem.cs*, A listbox item that can be highlighted used for the list box that implements the menu
- *HighlightableTextListBoxItem.cs*, inherited from the class above to display text in the menu
- *DataListBox.cs*, a cutom listbox for display data as [description value unit] or [description] used for *Operation*, *Device Status* and *Error Status* views
- *DataListBoxItem.cs*, a custom list box item for the class above
- *LineGraph.cs*, a custom class based in the image class to display graphs for fast and slow statistics (slow statistics ended up being every 10 seconds)
- *DataManager.cs*, Implements data management and formatting
- *TransmissionManager.cs*, implements serial communication and raw data transfer

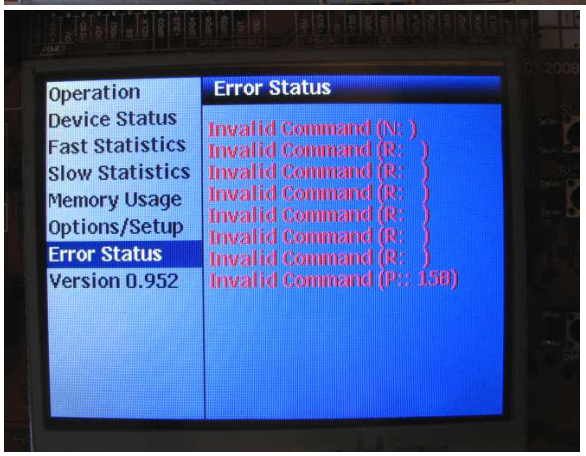
Visual studio Project files, Resource files etc. are not listed here.

5.3 GUI

The GUI has as depicted in the design been split up into a menu area and a view area. Below dumps of the GUI in *Operation*, *Fast Statistics* and *Error Status* state are shown:







The pictures are of the Tahoe II evaluation board that has been connected to a Fuel Cell Controller through Bluetooth. Data that not are refreshed are shown in red in the operational view. The error view is shown automatically when errors occur.

5.4 Class Overview

The classes as taken from Visual Studios Class diagram are displayed below:

Classes are in several categories:

- The Application – WTFCU, handling of buttons, state and graphics
- Text Based Views, handling of text based data
- Graphics Based views, handling of trends in text based data
- Data Management , Management and formatting of data
- Transmission Management, reading data and communication with the Fuel Cell

5.5 Test

The following has been tested

ID	Test Description	Expected Result	Status
1	Power up the Fuel Cell Controller, the Tahoe board and the attached BT modules	The blue led on the BT modules starts lighting blue after 5 seconds indicating correct pairing The yellow led will flash indicating data transfer	OK
2	Observe the first display of the application	The Operational view is shown and values are updated in the view	OK
3	Select the menu items for the implemented views (*Operation, Device Status, Fast Statistics, Slow Statistics and Error Status*).	The views appear and data is updated as expected	OK
4	Remove the cable to the Bluetooth device connected to the Tahoe board	Data is not being refreshed. Values in text views appear in red and graphs are static	OK
5	Reattach the cable to the Bluetooth device connected to the Tahoe board	Data is starts being refreshed. Values in text views appear in black and graphs are dynamic	OK
6	Remove the cable to the Bluetooth device connected to the Fuel Cell Controller	Data is not being refreshed. Values in text views appear in red and graphs are static	OK
7	Reattach the cable to the Bluetooth device connected to the Fuel Cell Controller	Data is starts being refreshed. Values in text views appear in black and graphs are dynamic	OK

6. Conclusion

The groups' initial decision with this report was to make what it perceived an ideal design, and implement what was possible with the given time frame.

The WTFCU implementation, utilizing the .Net Micro Framework running on a Tahoe II evaluation board, ended up with:

1. A set of Bluetooth units that automatically created at wireless virtual serial connection between the devices they connected to
2. A system that displays data as fast as it is transmitted from the Fuel Cell
3. Data is stored in a Data Manager Class that formats and stores data, and makes it available in a thread safe manner
4. A menu / view system based on custom menu and view classes, the menu is accessible via buttons or touch screen
5. A set of numeric data displays based on a custom data view classes
6. A set graphics displays based on a custom graph class

Requirements 3.1 (Stack Usage) and 4.1 (Configuration) was not implemented due to the time given for the work. 3.1 could be implemented in a second custom data view and 4.1 were beyond the scope since we had a limited set of Bluetooth units and one Fuel Cell to work with.

The implemented WTFCU showed us that it is possible to monitor a Hydrogen based Fuel Cell via windows embedded utilizing a Bluetooth connection.

There is an issue on serial performance, we are not quite confident with the Serial Port class offered by the .Net Micro Framework since single bytes are lost in a random pattern, although data is received errorless most of the time.

If the WTFCU should be perfected in the future, we recommend to:

1. *Implement a view for display stack usage to enable developer to optimize fuel cell memory usage.*
2. *Implement an Options Setup page for pairing of Com ports, Bluetooth Setup etc.*
3. *Implement a custom serial driver or a better utilization of the existing serial class*

7. APPENDICES

7.1 References

Reference Number	Description
1	Project Proposal: Wireless Service Terminal for Fuel Cells
2	Serial Port Adapter AT Command Specification v3.8.pdf

7.2 Word/Abbreviation List

Word/ Abbreviation:	Explanation:
WTFCU	Wireless Terminal for Fuel Cells