Degree Course Systems Engineering
Option Infotronics

Diploma 2010

Yannick Salamin

USB-SPI Translator

Professor Christophe Bianchi
Expert Lixin Ran

Lieu, date de la remise du rapport
The original report was written by the student and has not been corrected. It may therefore contain inaccuracies or errors.
<table>
<thead>
<tr>
<th>Titre / Titel</th>
<th>USB-SPI translator</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work involves a transparent communication between a standard USB port and multiple standard SPI ports. Although it is a small sub-system of a steering phased array antenna, it can also be used in all other similar applications.</td>
<td></td>
</tr>
<tr>
<td>The USB interface has become the major communication port for computers. However, USB is still not widely used in MCU world, in which the main serial communication protocols are RS232, I²C and SPI. Therefore it is still inconvenient for the interconnection between a computer and an MCU-based hardware. In this work, a Future technology's USB Serial UART interface chip, i.e., FT232R, and an MCU (TI's MSP430 or Atmel's ATmega 88) with a standard SPI port will be utilized to realize a USB-SPI translator, which could provide the hardware connection and transparent data transmission functions.</td>
<td></td>
</tr>
<tr>
<td>The work consists of following part:</td>
<td></td>
</tr>
<tr>
<td>Design, fabricate and debug a small hardware to realize the electrical connection between a standard USB port and multiple standard SPI ports;</td>
<td></td>
</tr>
<tr>
<td>Write a simple firmware code for the MCU connected with the FT232R to realize the translation between a UART protocol to SPI protocol;</td>
<td></td>
</tr>
<tr>
<td>Write a demo program to realize the two-way transparent data transmission between computer and MCUs by calling the driver provided by Future Technology.</td>
<td></td>
</tr>
</tbody>
</table>

| Delais / Termine |
| Attribution du theme / Ausgabe des Auftrags: | Remise du rapport / Abgabe des Schlussberichts: |
| 22.02.2010 | 12.11.2010, 12:00 |
| Remise du rapport intermediaire / Zwischenbericht: | Exposition publique / Ausstellung Diplomarbeiten: |
| 07.05.2010, 17:00 | |
| Defense intermediaire / Zwischenverteidigung: | Defense orale / Mündliche Verteidigung: |
| 21.05.2010 | Semaine 47 / Woche 47 |

1 Par sa signature, l'étudiant-e s'engage à respecter strictement la directive et le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition. Durch seine Unterschrift verpflichtet sich der Student, die Richtlinie einzuhalten sowie die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.
Objectives

The project involves the design of a communication interface between a computer and 37 serial interfaced D/A Converters to update their outputs and also process feedback information from the DACs outputs.

Methods / Experiences / Results

A USB controller (FT245R) is used for the USB communication between the computer and the processing unit (ATmega88). The user is able to send the DACs update information toward the microcontroller which sends the digital values to the DACs. A CPLD device is used to select multiple channels of 37 DACs. The MCU can sample the DACs output voltage via its built-in A/D converter and then send the information to the computer in order to be displayed for the user.

Two hardware boards are designed and fabricated. The main board, USB-SPI Translator which provides the power and data processing functions, is a general USB to SPI communication interface board. The other board is in charge of the DACs channels selection. After the realization of the hardware boards, firmware for the data processing unit (MCU) and the CPLD are written.

Some experiments about the whole system demonstrate that the DACs are able to be updated individually with a desired voltage. The precision of the DACs outputs is about one decimal. The MCU can get a 10-bits digital value of the DACs output. The digitalized DACs outputs are transmitted to the computer and displayed for the user.
Table of Contents

Acronyms .............................................................................................................................................. 5

1 Introduction ......................................................................................................................................... 6

2 Objectives ......................................................................................................................................... 8

3 Requirements ..................................................................................................................................... 9

4 Global System Architecture ........................................................................................................... 10

5 Hardware .......................................................................................................................................... 11
   5.1 Power .......................................................................................................................................... 11
      5.1.1 Power Source ....................................................................................................................... 11
      5.1.2 Power Management ............................................................................................................... 13
      5.1.3 Battery recharge management ............................................................................................ 17
      5.1.4 Switch ON/OFF ................................................................................................................... 18
      5.1.5 Provided Power Levels ........................................................................................................ 20
   5.2 Data Processing .......................................................................................................................... 21
      5.2.1 USB ...................................................................................................................................... 21
      5.2.2 MCU .................................................................................................................................... 22
   5.3 Inputs protection ........................................................................................................................... 25
   5.4 Chips selection ............................................................................................................................. 26
      5.4.1 CPLD .................................................................................................................................... 27
   5.5 Feedback channel selection ........................................................................................................ 27
   5.6 Detailed System Architecture ..................................................................................................... 28
      5.6.1 Hardware boards .................................................................................................................. 28
   5.7 PCB Design ................................................................................................................................ 29

6 Software .......................................................................................................................................... 31
   6.1 User interface ............................................................................................................................... 31
      6.1.1 GUI ...................................................................................................................................... 31
      6.1.2 Data processing ...................................................................................................................... 33
   6.2 USB-SPI Translator ...................................................................................................................... 36
      6.2.1 Startup .................................................................................................................................. 36
      6.2.2 Read USB data ..................................................................................................................... 38
      6.2.3 Received data analysis ......................................................................................................... 38
      6.2.4 DAC update ......................................................................................................................... 40
      6.2.5 DAC feedback processing .................................................................................................... 41
      6.2.6 Write USB data .................................................................................................................... 42
      6.2.7 Battery charging management ............................................................................................. 42
   6.3 CS Dispatch .................................................................................................................................. 44
      6.3.1 Firmware .............................................................................................................................. 44
      6.3.2 Simulation ............................................................................................................................. 45
      6.3.3 Synthesis ............................................................................................................................... 47

7 Experimental results ........................................................................................................................ 48
   7.1 Power management ...................................................................................................................... 48
<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Voltage control beam steering antenna board</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Solution block schema</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>DC source board</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>General view of the system</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>Global System Architecture</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Total system consumption and life time calculations</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>Battery comparative array</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>Used battery</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>Battery control</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>Step-up 5[V]</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>Power selector</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>Power management simulation schematic</td>
<td>16</td>
</tr>
<tr>
<td>13</td>
<td>Power management simulation result</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>Battery charger</td>
<td>17</td>
</tr>
<tr>
<td>15</td>
<td>Diode characteristic curve</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>RS Flip flop</td>
<td>19</td>
</tr>
<tr>
<td>17</td>
<td>Switch ON/OFF</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>MCU button</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>Power levels</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>USB Controller</td>
<td>21</td>
</tr>
<tr>
<td>21</td>
<td>MCU pin connection</td>
<td>22</td>
</tr>
<tr>
<td>22</td>
<td>SPI link for multiple slave</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>LC filter</td>
<td>24</td>
</tr>
<tr>
<td>24</td>
<td>2.5[V] Voltage reference</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>Programming device</td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td>Programming connector</td>
<td>25</td>
</tr>
<tr>
<td>27</td>
<td>Input protection circuit</td>
<td>25</td>
</tr>
<tr>
<td>28</td>
<td>Ideas comparative array</td>
<td>26</td>
</tr>
<tr>
<td>29</td>
<td>Advantage / Disadvantage table</td>
<td>26</td>
</tr>
<tr>
<td>30</td>
<td>Data transfer signals</td>
<td>27</td>
</tr>
<tr>
<td>31</td>
<td>CPLD clock signal</td>
<td>27</td>
</tr>
<tr>
<td>32</td>
<td>CPLD push button</td>
<td>27</td>
</tr>
<tr>
<td>33</td>
<td>Chips selection pin (sample)</td>
<td>27</td>
</tr>
<tr>
<td>34</td>
<td>Multiplexer hierarchy</td>
<td>27</td>
</tr>
<tr>
<td>35</td>
<td>Detailed System Architecture</td>
<td>28</td>
</tr>
<tr>
<td>36</td>
<td>Electrical connections between the two boards</td>
<td>29</td>
</tr>
<tr>
<td>37</td>
<td>USB-SPI Translator board</td>
<td>30</td>
</tr>
<tr>
<td>38</td>
<td>CS Dispatch board (Front)</td>
<td>30</td>
</tr>
<tr>
<td>39</td>
<td>CS Dispatch board (back)</td>
<td>30</td>
</tr>
<tr>
<td>40</td>
<td>USB-SPI Translator HW system</td>
<td>30</td>
</tr>
<tr>
<td>41</td>
<td>GUI</td>
<td>31</td>
</tr>
<tr>
<td>42</td>
<td>Connected device</td>
<td>32</td>
</tr>
<tr>
<td>43</td>
<td>Transmission types</td>
<td>32</td>
</tr>
<tr>
<td>44</td>
<td>GUI type in boxes (sample)</td>
<td>32</td>
</tr>
<tr>
<td>45</td>
<td>Connected device</td>
<td>33</td>
</tr>
<tr>
<td>46</td>
<td>Transmission types</td>
<td>34</td>
</tr>
<tr>
<td>47</td>
<td>GUI type in boxes (sample)</td>
<td>34</td>
</tr>
<tr>
<td>48</td>
<td>Connected device</td>
<td>35</td>
</tr>
<tr>
<td>49</td>
<td>Transmission types</td>
<td>35</td>
</tr>
<tr>
<td>50</td>
<td>GUI type in boxes (sample)</td>
<td>35</td>
</tr>
<tr>
<td>51</td>
<td>Connected device</td>
<td>36</td>
</tr>
<tr>
<td>52</td>
<td>Transmission types</td>
<td>36</td>
</tr>
<tr>
<td>53</td>
<td>GUI type in boxes (sample)</td>
<td>36</td>
</tr>
<tr>
<td>54</td>
<td>Connected device</td>
<td>37</td>
</tr>
<tr>
<td>55</td>
<td>Transmission types</td>
<td>37</td>
</tr>
<tr>
<td>56</td>
<td>GUI type in boxes (sample)</td>
<td>37</td>
</tr>
<tr>
<td>57</td>
<td>Connected device</td>
<td>38</td>
</tr>
<tr>
<td>58</td>
<td>Transmission types</td>
<td>38</td>
</tr>
<tr>
<td>59</td>
<td>GUI type in boxes (sample)</td>
<td>38</td>
</tr>
<tr>
<td>60</td>
<td>Connected device</td>
<td>39</td>
</tr>
<tr>
<td>61</td>
<td>Transmission types</td>
<td>39</td>
</tr>
<tr>
<td>62</td>
<td>GUI type in boxes (sample)</td>
<td>39</td>
</tr>
<tr>
<td>63</td>
<td>Connected device</td>
<td>40</td>
</tr>
<tr>
<td>64</td>
<td>Transmission types</td>
<td>40</td>
</tr>
<tr>
<td>65</td>
<td>GUI type in boxes (sample)</td>
<td>40</td>
</tr>
<tr>
<td>66</td>
<td>Connected device</td>
<td>41</td>
</tr>
<tr>
<td>67</td>
<td>Transmission types</td>
<td>41</td>
</tr>
<tr>
<td>68</td>
<td>GUI type in boxes (sample)</td>
<td>41</td>
</tr>
<tr>
<td>69</td>
<td>Connected device</td>
<td>42</td>
</tr>
<tr>
<td>70</td>
<td>Transmission types</td>
<td>42</td>
</tr>
<tr>
<td>71</td>
<td>GUI type in boxes (sample)</td>
<td>42</td>
</tr>
<tr>
<td>72</td>
<td>Connected device</td>
<td>43</td>
</tr>
<tr>
<td>73</td>
<td>Transmission types</td>
<td>43</td>
</tr>
<tr>
<td>74</td>
<td>GUI type in boxes (sample)</td>
<td>43</td>
</tr>
<tr>
<td>75</td>
<td>Connected device</td>
<td>44</td>
</tr>
<tr>
<td>76</td>
<td>Transmission types</td>
<td>44</td>
</tr>
<tr>
<td>77</td>
<td>GUI type in boxes (sample)</td>
<td>44</td>
</tr>
<tr>
<td>78</td>
<td>Connected device</td>
<td>45</td>
</tr>
<tr>
<td>79</td>
<td>Transmission types</td>
<td>45</td>
</tr>
<tr>
<td>80</td>
<td>GUI type in boxes (sample)</td>
<td>45</td>
</tr>
<tr>
<td>81</td>
<td>Connected device</td>
<td>46</td>
</tr>
<tr>
<td>82</td>
<td>Transmission types</td>
<td>46</td>
</tr>
<tr>
<td>83</td>
<td>GUI type in boxes (sample)</td>
<td>46</td>
</tr>
<tr>
<td>84</td>
<td>Connected device</td>
<td>47</td>
</tr>
<tr>
<td>85</td>
<td>Transmission types</td>
<td>47</td>
</tr>
<tr>
<td>86</td>
<td>GUI type in boxes (sample)</td>
<td>47</td>
</tr>
<tr>
<td>87</td>
<td>Connected device</td>
<td>48</td>
</tr>
<tr>
<td>88</td>
<td>Transmission types</td>
<td>48</td>
</tr>
<tr>
<td>89</td>
<td>GUI type in boxes (sample)</td>
<td>48</td>
</tr>
<tr>
<td>90</td>
<td>Connected device</td>
<td>49</td>
</tr>
<tr>
<td>91</td>
<td>Transmission types</td>
<td>49</td>
</tr>
<tr>
<td>92</td>
<td>GUI type in boxes (sample)</td>
<td>49</td>
</tr>
<tr>
<td>93</td>
<td>Connected device</td>
<td>50</td>
</tr>
<tr>
<td>94</td>
<td>Transmission types</td>
<td>50</td>
</tr>
<tr>
<td>95</td>
<td>GUI type in boxes (sample)</td>
<td>50</td>
</tr>
<tr>
<td>96</td>
<td>Connected device</td>
<td>51</td>
</tr>
<tr>
<td>97</td>
<td>Transmission types</td>
<td>51</td>
</tr>
<tr>
<td>98</td>
<td>GUI type in boxes (sample)</td>
<td>51</td>
</tr>
<tr>
<td>99</td>
<td>Connected device</td>
<td>52</td>
</tr>
<tr>
<td>100</td>
<td>Transmission types</td>
<td>52</td>
</tr>
<tr>
<td>101</td>
<td>GUI type in boxes (sample)</td>
<td>52</td>
</tr>
<tr>
<td>102</td>
<td>Connected device</td>
<td>53</td>
</tr>
<tr>
<td>103</td>
<td>Transmission types</td>
<td>53</td>
</tr>
<tr>
<td>104</td>
<td>GUI type in boxes (sample)</td>
<td>53</td>
</tr>
<tr>
<td>105</td>
<td>Connected device</td>
<td>54</td>
</tr>
<tr>
<td>106</td>
<td>Transmission types</td>
<td>54</td>
</tr>
<tr>
<td>107</td>
<td>GUI type in boxes (sample)</td>
<td>54</td>
</tr>
<tr>
<td>108</td>
<td>Connected device</td>
<td>55</td>
</tr>
<tr>
<td>109</td>
<td>Transmission types</td>
<td>55</td>
</tr>
<tr>
<td>110</td>
<td>GUI type in boxes (sample)</td>
<td>55</td>
</tr>
<tr>
<td>111</td>
<td>Connected device</td>
<td>56</td>
</tr>
<tr>
<td>112</td>
<td>Transmission types</td>
<td>56</td>
</tr>
<tr>
<td>113</td>
<td>GUI type in boxes (sample)</td>
<td>56</td>
</tr>
</tbody>
</table>
Figure 45: GUI incoming data display ................................................................. 33
Figure 46: USB data packet ............................................................................. 33
Figure 47: Data format ..................................................................................... 34
Figure 48: Sequence diagram of transmission ......................................................... 34
Figure 49: USB data read timing diagram ............................................................. 38
Figure 50: DAC update time diagram ................................................................. 40
Figure 51: DAC data feedback time diagram ......................................................... 41
Figure 52: USB data write time diagram ............................................................... 42
Figure 53: CS Dispatch TOP level ....................................................................... 44
Figure 54: Chips selection simulation ................................................................. 46
Figure 55: FB channel selection simulation ......................................................... 46
Figure 56: Battery output current (a) with external power source, (b) without external power source ................................................................................. 48
Figure 57: Power consumption by external supply (a) switched off, (b) switch on ........................................................................................................ 49
Figure 58: Power consumption by battery supply (a) switch off, (b) switch on ........................................................................................................ 49
Figure 59: Battery management (a) start of recharging signal, (b) battery charging current ......................................................................................... 50
Figure 60: Battery stops recharged ..................................................................... 51
Figure 61: Same voltage transmission (a) GUI, (b) 6.5[V] data transmission ........................................................................................................ 52
Figure 62: Different voltage transmission for each DAC (a) GUI, (b) all data transmission, (c) DAC 6 data transmission ........................................................................ 53
Figure 63: Single voltage transmission (a) GUI, (b) all data transmission, (c) DAC 12 data transmission ........................................................................ 53
Figure 64: Few voltages transmission (4 DACs) (a) GUI, (b) all data transmission, (b) DAC 31 data transmission ........................................................................ 54
Figure 65: Global test setup ................................................................................. 55
Figure 66: DACs feedback connector ................................................................. 56
Figure 67: DACs feedback test ......................................................................... 56
Figure 68: Beam steering experiment setup ......................................................... 57
Figure 69: Beam steering experiment results ...................................................... 58
Acronyms

DAC  Digital to Analog Converter
ADC  Analog to Digital Converter
SPI  Serial Peripheral Interface
FIFO First In First Out
CPLD Complex Programmable Logic Device
IC   Integrated Circuit
JTAG Joint Test Action Group
ISP  In-System Programmable
GUI  Graphical User Interface
MCU  Micro-Controller Unit
1 Introduction

Nowadays the most used communication interface for computer is USB. However, USB is still not widely used in embedded systems, in which serial communication interfaces as RS232 or SPI are mainly used. Therefore interconnection between computers and embedded systems is still an inconvenience and needs some translation interface in order to connect the equipment’s together.

The AEM group of the Zhejiang University is a laboratory specialized in applied electromagnetic research. Recently they focused their research on metamaterial directive antenna. In this way, they developed a DC voltage control beam steering metamaterial antenna. The main function of this antenna is to tune the beam azimuth by DC voltage. Like phased array antenna, the antenna consists of a voltage-controlled metamaterial slab (beam steering antenna) of 36 phase shifters and a patch antenna. By applying a specific voltage on a phase shifter, a corresponding phase will result. The final direction of the antenna is determined by the 36 phases together.

The final aim is to control the applied voltage on the beam steering antenna by a computer, and thereby be able to control the direction of the whole antenna in real-time. In order to realize it, some intermediary interface shall process the incoming computer information and apply the right voltage on each phase shifter.

Our solution is to have a first control system (Control board) to fulfill the communication between the computer (PC) and a power source system (DC source board). The DC source board shall perform the digital to analog conversion of the received user information in order to provide and keep the correct voltage on each phase shifter of the beam steering antenna. By placing the patch antenna behind the regulated beam steering antenna, the DC voltage control beam steering metamaterial antenna is completed.
The DC source board has already been done. To provide the specific voltage for each phase shifter, thirty-seven digital-to-analog converters are used. The chosen communication interface for the DACs is the SPI protocol, a common and easy to use serial communication interface. Each DAC is able to receive digital information through the SPI interface, converts it into analog signal, which reflects the digital information, and keep the result on the output until new digital information is received. Each DAC possesses an amplifier to increase the output voltage into a range of 2-10V approximately.

As mentioned above, an additional interface is needed to fulfill the communication between the computer and the DC source board. Due to the communication interfaces are different, the control system shall work as protocol translator between USB and SPI in order to transmit the digital information coming from the computer toward the DC source board.

This paper present the realization of the control system, a communication interface between a computer and thirty-seven serial interfaced (SPI) digital-to-analog converters (DACs) incorporate in the DC source board.
2 Objectives

The existing DC source board is only able to communicate through SPI interfaces. Therefore, in order to send information to the DACs from a computer, a translation interface is needed to handle the computer data and send it toward the DACs.

The final objective consists of a user cable to enter a desired direction angle into a graphical interface, on a computer, and control the antenna’s direction. Due to the relation between applied voltage, phase shifter and final direction isn’t known yet, some experimental tests are needed. In order to realize these tests, the user shall also be able to control the output voltage of each DAC in order to regulate each single phase shifter of the voltage –controlled board (beam steering antenna) and thereby determine the right relation between voltage, phase and direction. Once the relation established, the user should only enter the angle and each single DAC voltage shall be calculate by the MCU incorporate in the control system.

The main purpose of the project is to realize a USB-SPI translator system which provides both hardware connection and transparent data transmission functions. In order to accomplish it, several tasks are required:

- First, a small hardware is to be designed, fabricated and debugged which realize the electrical connection between a standard USB port and multiple standard SPI interfaces.

- Once the hardware finished, a small firmware for the data unit processing (MCU) connected with a USB controller has to be written. The system shall work as translator between USB and SPI protocols.

- Write a graphical interface for the user, which realizes the bidirectional communication with the hardware.
3 Requirements

In projects it’s essential to clearly and completely define the functions and duties of the product. The system has to meet certain requirements; some of them are defined by the user [1], others are necessary to obtain a well working system.

The USB-SPI Translator system shall:

**Functional specifications**
- process data between a USB interface and 37 SPI interfaces
- update the DACs output voltage on the DC source board
- implement a FT245R USB controller and a ATmega88 microcontroller which are communicating in parallel operation

**Interface specifications**
- have a maximal size of 16.5mm by 3.5mm and shall be able to be fixed to the digital-to-analog board
- be a low power consumption system
- provide a SPI master interface able to communicate with 37 SPI slave devices

**GUI specifications**
- allow the user to choose between different data transmission types
- allow the user to type in the desired information to be transmit
- transmit the information typed in by the user by using the provided FTDI driver
- receive information by using the provided FTDI driver
- display the received information

**Convenient specifications**
- provide a battery able to supply the data processing unit
- process feedback information of the DACs output voltage

Further, some additional specifications might be defined in order to obtain a complete well working product.
4 Global System Architecture

As result of the analysis of the objectives and the requirements, a first block diagram describing the global system architecture, is shown in Figure 5. A more specific and detailed system architecture is depicted at the end of section 5 Hardware.

![Figure 5: Global System Architecture](image-url)
5 Hardware

In this section the hardware of the USB-SPI Translator system is discussed. Each function is described and the selection of the components is explained. For the experimental version, most of the components used are chosen from the lab if available and if the requirements are met, for convenience. If the experiment work out well the final version can integrate more specific devices as mentioned in this paper.

This discussion allows us to represent the USB-SPI Translator system in a more detailed system architecture (see section 5.5 Detailed System Architecture).

The different schematics resulting from the discussion are included in Appendix sections B Schematic. Also all the components used are listed in Appendix sections D Components list.

5.1 Power

5.1.1 Power Source

For embedded systems, in which the mobility and the self-sufficiency are important, it’s very useful to use a battery. On the contrary, if the system stay at the same place and the maintenance shall be very seldom, it’s a better way to use an external power supply.

In this application the system doesn’t need especially to be self-powered, but due to the USB-SPI Translator system is in charge for carrying information between equipment, a self-powered system can be used during power cuts in order to save the information. In this way, the system shall be able to switch the power source from external power link to a battery able to supply the data processing unit of the system, when the external power source is no longer available.

In order to fulfill this function, the system shall provide a DC connector with both signals, 5[V] and GND, and a battery able to supply the system, or at least the data processing unit.

External link

In order to supply the whole system by external power, one standard DC connector is needed. As external supplier, the digital-to-analog converter board is able to provide 5[V] power which is perfect. A standard ISP 2 pins connector is used to draw the power from the external supply to the USB-SPI Translator system.

Power pack

To choose a battery for the system, some calculation about the needed battery capacity and its life time have to be done.

First, a table with the different potential components is defined. With the help of the given manufacturer datasheets, each component consumption, for idle and active states, is defined and the approximate system consumption can be calculated. The results are shown in Figure 6.
Comment:

*All the calculations are theoretical and thereby the obtained results might differ with the reality. This should be taken into consideration when choosing the right battery for the system.*

Once the system consumption known and the life time defined, the battery capacity can be calculate with the formula as follow:

$$\text{Capacity} \ [\text{mAh}] = \text{Total system consumption} \times \text{life time}$$ (1)

A comparative table, Figure 7, showing different batteries gives an idea of what kind of battery the system needs. It allows us to understand the relation between the size of the battery and the capacity of it. The table points out clearly that the capacity depends strongly on the size.

<table>
<thead>
<tr>
<th>Fabricant</th>
<th>Model</th>
<th>Technology</th>
<th>Capacity [mAh]</th>
<th>Voltage [V]</th>
<th>Size [mm]</th>
<th>Connection</th>
<th>Item</th>
<th>Total Voltage [V]</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARTA</td>
<td>55615604940</td>
<td>NiMH</td>
<td>150</td>
<td>4.8</td>
<td>23x15x14</td>
<td>PCB pin</td>
<td>1</td>
<td>4.8</td>
</tr>
<tr>
<td>VARTA</td>
<td>55625603059</td>
<td>NiMH</td>
<td>250</td>
<td>3.6</td>
<td>25x20x26</td>
<td>PCB pin</td>
<td>1</td>
<td>3.6</td>
</tr>
<tr>
<td>VARTA</td>
<td>55625602059</td>
<td>NiMH</td>
<td>250</td>
<td>2.4</td>
<td>25x20x27</td>
<td>PCB pin</td>
<td>2</td>
<td>4.8</td>
</tr>
<tr>
<td>VARTA</td>
<td>55750504012</td>
<td>NiMH</td>
<td>500</td>
<td>4.8</td>
<td>70x50x9</td>
<td>wire</td>
<td>1</td>
<td>4.8</td>
</tr>
<tr>
<td>Pololu</td>
<td>1005</td>
<td>NiMH</td>
<td>700</td>
<td>4.8</td>
<td>46x41x11</td>
<td>wire</td>
<td>1</td>
<td>4.8</td>
</tr>
<tr>
<td>VARTA</td>
<td>56445201012</td>
<td>Li-Ion</td>
<td>595</td>
<td>3.7</td>
<td>42x34x5</td>
<td>wire</td>
<td>1</td>
<td>3.7</td>
</tr>
<tr>
<td>VARTA</td>
<td>56427201012</td>
<td>Li-Ion</td>
<td>1400</td>
<td>4.2</td>
<td>60x40x5</td>
<td>wire</td>
<td>1</td>
<td>4.2</td>
</tr>
<tr>
<td>VARTA</td>
<td>56622201013</td>
<td>Li-Ion</td>
<td>2200</td>
<td>3.7</td>
<td>65x18</td>
<td>wire</td>
<td>1</td>
<td>3.7</td>
</tr>
<tr>
<td>ENIX ENERGIES</td>
<td>800042</td>
<td>Li-Ion</td>
<td>2600</td>
<td>3.75</td>
<td>55x45x18</td>
<td>wire</td>
<td>1</td>
<td>3.75</td>
</tr>
<tr>
<td>DMS Technologies</td>
<td>7140-0085</td>
<td>NiCd</td>
<td>700</td>
<td>4.8</td>
<td>50x50x15</td>
<td>wire</td>
<td>1</td>
<td>4.8</td>
</tr>
<tr>
<td>Panasonic</td>
<td>AAA ZR03/2BP</td>
<td>Nickel Ox.</td>
<td>850</td>
<td>1.5</td>
<td>45x10</td>
<td>support</td>
<td>3</td>
<td>4.5</td>
</tr>
<tr>
<td>VARTA</td>
<td>56422302016</td>
<td>Lithium</td>
<td>2600</td>
<td>3.7</td>
<td>65x35x10</td>
<td>wire</td>
<td>1</td>
<td>3.7</td>
</tr>
</tbody>
</table>

Figure 6: Total system consumption and life time calculations

As the size of the hardware is limited, the size of the battery is determinant. Another significant point is the technology used. By using Li-Ion technology, the charge of the battery should be accurate and a battery controller is needed in order to regulate the charge. On the other hand, NiMH batteries can be
easily charged with small current flow without external regulation. The second choice is more convenient for the system, due to the gain of size is significant.

As result of the formula (1), the capacity of the battery should be between 250mAh and 700mAh. As well, the table above contains a battery that meets the different discussed requirements. The small size of the selected battery was a major feature for the choice.

<table>
<thead>
<tr>
<th>Fabricant</th>
<th>Model</th>
<th>Technology</th>
<th>Capacity [mAh]</th>
<th>Voltage [V]</th>
<th>Size [mm]</th>
<th>Connection</th>
<th>Item</th>
<th>Total Voltage [V]</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARTA</td>
<td>55625603059</td>
<td>NiMH</td>
<td>250</td>
<td>3.6</td>
<td>25x20x26</td>
<td>PCB pin</td>
<td>1</td>
<td>3.6</td>
</tr>
</tbody>
</table>

The battery chosen for the experimental version differs with the selected one\(^1\); the characteristics are shown below in Figure 8.

<table>
<thead>
<tr>
<th>Fabricant</th>
<th>Model</th>
<th>Technology</th>
<th>Capacity [mAh]</th>
<th>Voltage [V]</th>
<th>Size [mm]</th>
<th>Connection</th>
<th>Item</th>
<th>Total Voltage [V]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZL</td>
<td></td>
<td>NiMH</td>
<td>40</td>
<td>3.6</td>
<td>17x13x16</td>
<td>PCB pin</td>
<td>1</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Figure 8: Used battery

5.1.2 Power Management

The system has two different potential power sources, and especially one of these power sources is a battery that needs to be managed. Follow the power management circuit, which fulfill the power source selection and the battery management, is described.

Battery control

As the battery is normally always connected to the system, a P-MOSFET transistor (U1A) is used to control the battery utilization. MOSFET are voltage-driven transistors. The theoretical relation that controls the P-MOSFET conductibility is as follow:

\[
V_{GS} = V_{Gind} - V_{Source} \leq 0[V] \Rightarrow Drive! \\
V_{GS} = V_{Grind} - V_{Source} > 0[V] \Rightarrow Blocked!
\]  

Figure 9 shows the use of the transistor (U1A) as switch. The battery is connected to the source of the P-MOSFET and the external power link (Vdc) is connected to the grind in order to control the transistor. When the external power is available (5V), the P-MOSFET is open and no current is drawn from the battery. On the contrary, if no external power is available (0V), the P-MOSFET drives and the battery will supply the system.

\(^1\) For the experimental version a smaller battery available in the lab has been chosen for convenience.
**Step-up**

When the battery is being used the voltage of this one drop. In order to have a constant output voltage of the battery power source when it’s being used, a voltage converter IC chip is used to step up the voltage to a constant 5[V].

The *ON Semiconductor NCP140 [5]* step-up DC/DC converter is chosen for this application\(^2\). The *NCP1410* device is a boost voltage switching converter IC specially designed for battery operated hand-held electronic. The operating input voltage range of the IC, for a 5[V] output, is between 1.8[V] and 5[V] and the output current drawn from the battery can be up to 200mA at 2.2V input voltage.

\(^2\) The step-up was chosen because it was available in the lab and met the requirement.
The output voltage of the converter is determined by the external feedback network comprised of R5 and R6 (refer to Figure 10). Below is described the calculation for a 5[V] output with the given relationship:

\[ V_{out} = V_{ref} \left( 1 + \frac{R5}{R6} \right) = 5[V] \]  

(3)

Select R5 = 100KΩ

\[ R6 = R5 \left( \frac{V_{out}}{V_{ref}} - 1 \right) = 100K \left( \frac{5}{1.19} - 1 \right) = 333KΩ \]

The besides used components for a well working step-up are chosen according to the datasheet given by the manufacturer [5].

**Power selector**

Due to the system can be supplied by two different power sources, some electronic is used to select the power source which has to supply the system. If the external power source is available, the system shall be supplied by this one and not use the battery.

A P-MOSFET transistor (U1B) is used to select the power source (see Figure 11). For the conductibility of the P-MOSFET, refer to the relation (2) on page 13. The external power source is connected to the drain of the P-MOSFET and the step-up output (battery power source) to the grind. The source pin of the P-MOSFET drives the selected power source toward the system.

When external power is available the first P-MOSFET transistor (U1A), which controls the battery utilization, doesn’t drive and the step-up circuit output is null (0[V]). With the grind voltage low, null, the second P-MOSFET transistor (U1B) drives and the external power source supplies the system.

On the other hand, if the external source isn’t available the first P-MOSFET (U1A) drives the battery current through the step-up which apply 5[V] on the grind. Thus, the second P-MOSFET (U1B) doesn’t drives and the battery supplies the system through a standard diode. The diode is used to avoid a feedback voltage on the step-up output when the external source supplies the system.

![Figure 11: Power selector](image-url)

Figure 11: Power selector
Power management simulation

A simulation of the power management has been realized with the *OrCAD* software *PSpice*. For the simulation the schematic is simplified as shown in Figure 12.

![Power management simulation schematic](image)

**Figure 12 : Power management simulation schematic**

![Power management simulation result](image)

**Figure 13 : Power management simulation result**

If external supply (*Vdc*) is available, the simulation shows that the first MOSFET (yellow signal) doesn’t drives. As the first MOSFET monitor the second MOSFET, the external power source supply the system (red signal) (1). When the external source is no longer available, the first MOSFET drives and consequently block the second MOSFET. The battery supplies the system through the diode (2). The output voltage (*Vsysenm*) is reduced due to the power drop on the diode. If no power sources are available, the output voltage is null (3).
5.1.3 Battery recharge management

In order to avoid the necessity to recharge the integrated battery by another special system, the USB-SPI Translator system shall perform the recharge of the battery.

The battery source should be connected to an analog pin from MCU in order to convert the battery level with the built in analog-to-digital converter. The obtained digital result is used by the MCU to control the charging of the battery. The battery should be recharged only if the voltage level is under a specified low threshold and should stop charging when the voltage reached the specified high threshold. The MCU controls the charge of the battery by controlling the conductivity of an N-MOSFET (U2A in Figure 14) with the signal “charge”. The theoretical relation that controls the N-MOSFET conductivity is as follows:

\[ V_{GS} = V_{Gind} - V_{Source} = 0[V] \rightarrow \text{Open!} \]

\[ V_{GS} = V_{Gind} - V_{Source} > 0[V] \rightarrow \text{Drive!} \] (4)

The N-MOSFET drives when the “charge” signal on the grind is high and will consequently connect the ground to the diode. If the control signal is low, the N-MOSFET is open and the external source is connected to the diode.

The purpose of the diode\(^3\) is to avoid that the N-MOSFET (U2A) draws current from the battery when the external source isn’t available. The diode only drives if the voltage on the anode side, external power source, is higher than the voltage on the cathode side, battery side. The current drawn by the battery is exponential of the voltage different between anode and cathode as shown in Figure 15.

\(^3\) The diodes are standard diode chosen from the lab.
Figure 15: Diode characteristic curve

The diode characteristic curve above shows clearly that more the battery is charged, less current is driven by the diode due to the voltage different between anode and cathode become smaller. This characteristic is interesting by the fact that it is better for the battery to reduce the charging current above 80% of charge capacity. When the external power source isn’t available or the signal “charge” is high (no charging), the voltage different between anode and cathode is negative, thus no current is driven by the diode and the battery is protected.

The external source is directly connected to the diode as shown in Figure 14. The current provided by the external source is unknown, thereby a resistance (R8) is used to decrease the charging current. A small charging current scale is determined, after what the resistant can be calculated as follows:

Select $I_{\text{charge}} = 5\text{mA}$

$$R8 = \frac{V_{\text{ext}}}{I_{\text{charge}}} = \frac{5}{0.005} = 1k\Omega$$

5.1.4 Switch ON/OFF

The user shall be able to turn on and off the system by using a push button. When the system is shutdown, the minimal electronic shall be supplied in order to avoid a waste of current, especially when the battery is supplying the system.

When the USB-SPI Translator system is switched off, a minimal electronic should be supplied only to allow the user to switch on the whole system. To control the supply of the system behind the switch ON/OFF circuit, a P-MOSFET transistor (U7B in Figure 17) is used. For the conductivity of the P-MOSFET, refer to the relation (2) page 13.

As shown in Figure 17, the grind of the P-MOSFET controls the conductivity of the transistor, so when the system is switched on, the grind should be low, null. Like this the transistor drives and the whole system is supplied by the selected power source. If the system is turned off, the grind should be high, in this case the transistor becomes open and by consequent the system is turned-off.

In order to control the P-MOSFET transistor, a RS-Flip-flop is used. A NAND RS-Flip-flop produces a low level on the output by applying a low level on the Set input. By applying a low level on the Reset input, a high level result on the output. If both inputs are high level, the output is kept unchanged. The logical connection and the true table of the NAND RS flip flop are shown in Figure 16.
The CD4011B chip from *Fairchild [6]* manufacturer is chosen to fulfill the NAND RS flip flop. The device is a CMOS technology integrated circuits consisting of four 2-Input NAND gates.

As mentioned above, the user should be able to turn on the system by using a push button (SW_ON). As shown in Figure 17, by pushing the button, the user connects the ground to the Set input of the RS flip flop. As consequent, the RS flip flop output is pulled down and the N-MOSFET transistor drives the selected power source (Vsel) forward the system. A pull-up resistance on each input keeps the output state unchanged and thereby the system stays supplied.

---

4 The CD4011B chip was chosen for its availability in the lab.
The push button should be used for the switch on and the shutdown for a well-working system. In order to fulfill this requirement, the push button is connected through an N-MOSFET transistor (U7A) working as interrupter, to the MCU (Figure 18). The pull-up resistance (R9) on the set input of the RS flips flop keeps the transistor open. When the user push the button, the grind of the P-MOSFET is connected to the ground and allows the transistor to drive. It allows the MCU to be aware of the push button state. The microcontroller shall pull down the “ON/OFF” external signal in order to shut down the system, when the user push the button more than a defined time. The “ON/OFF” signal is connected to the reset input of the RS flip flop, applying a low level on the reset input will produce a high level on the output of the RS flip flop, and by the way shut the system down.

![Figure 18: MCU button](image)

### 5.1.5 Provided Power Levels

The USB-SPI Translator system use different voltage level for different components and therefor the system shall transform the source power into the different needed power level.

The provided voltage level by the selected source power is 5[V], voltage regulators are used to regulate the voltage level. For each voltage level, a specific voltage regulator is chosen.

![Figure 19: Power levels](image)

Figure 19 shows the different voltage level used by the different functionalities. It results that the system should integrate two voltage regulators.
Only the USB controller is supplied with the full power level. The reason is that the minimal well-working voltage limit given by the manufacturer for the FT245R [4] device is 4[V].

In order to have a low power consumption system, a 3.3[V] voltage level is chosen for the different data processing unit, the ATmega88 [2] microcontroller and the XC95144XL CPLD [3] (refer to section 5.3 Chips selection). The MICREL MIC5205-3.3BM [7] voltage regulator from manufacturer is chosen in order to provide a 3.3[V] power level.

The digital-to-analog converters used on the DACs board are using a reference voltage level of 2.5V. Thereby the analog-to-digital converter integrate in the MCU should have the same reference voltage level. This way the feedback information can be accurate. The same voltage reference chip as used by the DACs board is chosen, the BB TI manufacturer REF3025 [8] voltage reference chip.

5.2 Data Processing

To handle all the outgoing and incoming signals, the USB-SPI Translator system needs a processing unit to manage it. The main task of the processing unit is to translate communication protocols for bidirectional data transmission between standard USB interface and standard SPI interface.

5.2.1 USB

The main data processing unit is the microcontroller, but in order to successfully communicate with a computer through USB protocol, a USB controller is used. The controller used shall work as translator between a standard USB communication link and a FIFO communication interface able to communicate with the MCU by parallel operations.

The FTDI FT245R [3] USB controller chip chosen (refer to section 3 Functional specification) is a USB to parallel FIFO interface for bidirectional data transfer by rates up to 1Mbyte/second. In Figure 20 the USB controller is designed according to the given datasheet [3].

5 The MIC5205-3.3BM chip was chosen for its availability in the lab and its already use in the DC source board
Power configuration

The *FT245R* chip supports bus powered or self-powered configurations. Because we have a separate power supply, the chip is self-powered. As the USB controller device is self-powered, it shouldn’t draw current down the USB bus when the USB host is powered down. In order to comply with this requirement above, the bus power (Vbus) is used to control the RESET # pin of the *FT245R* device as shown in Figure 20. Like this, when the USB host is powered the RESET# signal is high and the device is functional. When the USB host is shut down, the signal on RESET# is low and the device is turned off until the host switches on again.

5.2.2 MCU

The microcontroller is the central control unit and thereby is connected with the whole system, by signals and data lines. The chosen microcontroller is the *ATmega88* from *Atmel* [2] (refer to section 3 Functional specification). The ATmega88 device is a low power CMOS 8-bit microcontroller based on AVR CPU architecture. The hardware of the used MCU features (I/O, SPI, A/D Converter) are described in this section. Figure 21 shows the MCU pin connections.

![Figure 21: MCU pin connection](image)

USB Controller to MCU connection

The main task of the MCU is to process the data between a USB host (Computer) and 37 SPI interfaces (DACs). As described above in section 5.2.1 USB, an intermediary device is used for the communication with the computer.
As the USB controller communication interface with the microcontroller is an 8-bit parallel FIFO interface, two 8-bits ports from MCU are used for the bidirectional data transfers with the intermediary device. One 8-bits port, the PORTD (PD) is used to transfer the 8-bits data. The another 8-bits port, PORTB (PB) from MCU, use four bits to monitor the RXF# (PB6) and TXE# (PB7) status bits and generate the RD# (PB0) and WR (PB1) strobe signals to the device when required. The RXF# and RD# signal are used to read the data on the device. The TXE# and WR signals are used to write the data into the USB controller.

**I/O Signals**

Like mentioned before, the MCU is the central control unit, therefore one of its duty is to control the different external signals. Some of these signal are used to control the system, others are used to control external status information for the user.

Light emitting diodes (LED) are used as status information. One LED shall inform the user of the working state of the system, ON/OFF. Another LED shall inform the user of a low battery level, and if external power source available, battery recharging status.

**SPI interface**

The chosen MCU provide an SPI interface. This feature is used by the system to communicate with the digital-to-analog converters on the DC source board. The SPI interface provided by the USB-SPI Translator board has to work as master device and be able to communicate with 37 SPI slave interfaces.

![SPI link for multiple slave](image)

Figure 22 shows the connection between one master SPI interface and several slave SPI interfaces. The data links stay the same for each slave devices. A slave select signal is used to enable the slave device that has to exchange the data information with the master device. Due to the DACs used are 10-bits devices, two SPI transmissions are necessary to update a DAC. In order to synchronize the data transfer a additional signal (FS_N) is used.

Because the chosen MCU has not enough I/O pins to provide 37 slaves select signals, a sub-system is used to perform the slave devices selection. This slave selection sub-system is described in section 5.3 Chips selection.
**Analog-to-digital converter**

The chosen microcontroller integrates a 10-bit analog-to-digital converter (ADC), which means analog signal are able to be process by the MCU. The built in ADC is used to convert the feedback information of the digital-to-analog converter output. After data processing by the MCU, the user can be aware of the different DAC voltage level in real-time. Another use of the provided ADC is to convert the battery voltage level in order to control its charge.

In order to have an accuracy A/D converter, the analog voltage supply for the ADC (AVcc pin) should be connected to the digital supply voltage (3V3) via an LC filter as shown in Figure 23.

![Figure 23 : LC filter](image)

The voltage reference used by the digital-to-analog converter on the DC source board is 2.5[V]. As the conversion result depends highly on the voltage reference used, the same reference voltage should be used by the built in ADC as the DC source board. Figure 24 shows the voltage regulator used for a precise voltage reference of 2.5[V]. The device used is the same used by the DACs board (refer to section 5.1.4 Power levels).

![Figure 24 : 2.5[V] Voltage reference](image)

The ATmega88 microcontroller provides two permanent analog inputs. They are used for the DACs feedback (ADC6) and the battery feedback (ADC7). Due to the microcontroller can’t provide 37 analog inputs for every DACs output, a sub-system is used to select the DAC which has to be converted. The DAC feedback sub-system is described in section 5.4 Feedback channel selection.

Because 2.5[V] is used as reference voltage, a voltage divider is used for the battery signal in order to match the highest battery output voltage with the reference voltage. This is necessary because the converting input voltage range is between 0[V] and Vref.

\[
V_{out} = V_{ref} = V_{bat_{max}} \times \frac{R16}{R16 + R15} = 2.5[V]
\]

By selecting R15=**100[kΩ]**, R16 resulting is **200[kΩ]**.
**MCU programming interface**

To be able to write the firmware into the microcontroller, a programming interface is needed. A multi device programming interface, called ISPExpert, is used as programming device for the MCU. The ISPExpert device is able to work with Joint Test Action Group (JTAG) and In-System Programming (ISP) programming protocols.

The second choice, ISP protocol, uses the SPI interface to write into the target device, for that reason this option is chosen in order to avoid waste of pin uses, as the MCU I/O pins are limited. Figure 26 shows the electrical connection for the programming connector.

![Programming device](image)

**5.3 Inputs protection**

The USB-SPI Translator system communicates with external systems and is able to receive information from them. For this reason a protection for the MCU should be implemented. A zener diode with a serial resistance is used as protection circuit against high voltage signal on the MCU.

![Input protection circuit](image)

As shown in Figure 27, each external input signal is connected in series to a resistance and to a zener diode in parallel. The breakdown voltage for the zener diode is chosen according the highest potential signal voltage level. When the analog signal is higher than the breakdown voltage of the zener diode, the diode will drive and the MCU pin will see 3.6[V]. The excess of voltage is absorbed by the resistance. Like this the MCU is protected from external high voltage level. The formula (6) below shows the relation for the serial resistance (R18):

\[
\frac{U_{\text{emax}} - U_z}{I_{z\text{max}}} < R_s
\]  

(6)

According to the datasheet of the zener diode [9], Izmax is calculated as follow:

\[
I_{z\text{max}} = \frac{P_{\text{max}}}{U_z} = 139[\text{mA}].
\]

By using the obtained result in the formula (6), Rs can be determined (Uemax = 25[V], Uz = 3.6[V]). The resistant value should be above 157[Ω], therefore the selected value for Rs (R18) is 200[Ω].
5.4 Chips selection

In order to send new digital information to the digital-to-analog converters, the CS pin of the DAC chip needs to be pulled down firstly. The MCU already chosen (ATmega88) own 23 programmable I/O pin and actually the system needs 37 slave select as mention above in section 5.2 2 MCU (SPI interface). Therefore a specific sub-system, described in this sub-section, shall manage the chips selection.

Several solutions could be used to perform the chips selection. In order to choose the best solution for the application, an analysis about different ideas is depicted below.

A first array including four potential solutions shows in Figure 28 the difference between the ideas. After what Figure 29 show the advantages and disadvantages for each idea.

<table>
<thead>
<tr>
<th>Idea</th>
<th>Workable</th>
<th>HW</th>
<th>SW difficult</th>
<th>Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bigger MCU</td>
<td>YES</td>
<td>Little size extension</td>
<td>output selection</td>
<td>75uA &lt; I &lt; 400uA</td>
</tr>
<tr>
<td>Second MCU</td>
<td>YES</td>
<td>Double size extension</td>
<td>CS divided selection</td>
<td>500uA</td>
</tr>
<tr>
<td>MCU-CPLD</td>
<td>YES</td>
<td>Extra chip extension</td>
<td>MCU-CPLD communication</td>
<td>&lt; 50mA</td>
</tr>
<tr>
<td>De-Multiplexer</td>
<td>YES</td>
<td>Some extra chips extension</td>
<td>address selection</td>
<td>&lt; 300uA</td>
</tr>
</tbody>
</table>

Figure 28 : Ideas comparative array

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
</table>
| Bigger MCU | • Only one chip used  
• No synchronization with another chip  
• Low power  
• Easy connecting link  

| Second MCU | Data sharing  
|-------------|-------------|
| Multi task  | • Divided chip selection  
|             | • Synchronization between both MCU  |
| MCU – FPGA  | Synchronization between MCU and FPGA  
|             | Extra debug interface used  |
| De-multiplexer | Some extra chips used  
|               | Complicated connecting link  |

Figure 29 : Advantage / Disadvantage table

Although the analysis shows clearly that using a bigger MCU for the application is the best solution, the specification requires the use of the ATmega88 MCU. Consequently the second best solution, which is
the use of a CPLD for the chips selection, is kept. The chosen CPLD for the application is the \textit{XC95144XL} device from \textit{Xilinx} \footnote{The CPLD chosen was available in the lab and was already used by it} [3].

### 5.4.1 CPLD

To use a CPLD for the chips selection, the MCU and the CPLD has to communicate together. SPI protocol is used as communication interface between the MCU and the CPLD device. Figure 30 shows the SPI data signals used for the communication.

The CPLD need an external clock to work. The SPI clock from MCU is used as clock signal for the device. As shown in Figure 31 only one clock reference is used. In this way, the CPLD is at any time synchronous to the MCU and the communication become easier.

A push button is added to the system for the CPLD. The button pulls up the general set/reset (GSR) pin when it’s pushed, in this way the button can be used as reset button for the CPLD.

Due to the CPLD is used to select the DACs, each DAC chip selection signal has to be connected to the CPLD as output. In order to enable the DAC read the new digital information to convert, the corresponding CS\textsubscript{N} pin is pulled down.

### 5.5 Feedback channel selection

As the USB-SPI Translator system conveys digital information toward the digital-to analog converters, it might be interesting to have a feedback of each DAC’s output voltage.

The MCU shall read the output of the DACs, digitalize it with its integrated analog-to-digital converter and then send the information back to the computer. As mentioned above, a sub-system is needed to select and thereby connect the DAC output signal to the analog MCU pin in order to convert the output voltage.

Because the DACs output voltages are analog signals, analog multiplexers are used to connect the selected DAC output to the MCU. As there are 37 DACs, at least one 37 to 1 multiplexer is needed. As such big analog multiplexer doesn’t exist on the market; several smaller multiplexer should be used. By using five 8 to 1 multiplexers, least of multiplexer input pins are wasted ($5\times8=40$ pins, $40-37=3$ pins unused). As shown in Figure 34, a sixth

---

\footnote{The CPLD chosen was available in the lab and was already used by it}
A multiplexer is needed to select the right output of the five multiplexers.

The Texas Instrument SN74HC4851 [10] 8-channel analog multiplexer is chosen to fulfill the feedback channel selection task.

The CPLD is used to control the multiplexer’s inputs (channel selection). An address of six bits (sel1...6) represents the DAC output to select. The first three bits select the feedback channel (FB_N1…37) and the last three bits select the right output of the five multiplexer which are selecting the feedback channel.

5.6 Detailed System Architecture

USB-SPI Translator

![Detailed System Architecture](image)

USB-SPI Translator is divided in four sub-systems (green-brown) and several functional blocks (gray). The red links stand for electricity, blue links for data transmission, brown for signals and black links for external connections.

5.6.1 Hardware boards

The USB-SPI Translator system is divided into two electronic boards. The first one, called USB-SPI Translator, provide the Power and Data processing units. This board is a general USB to SPI communication interface board that could be used for any different applications involving USB to SPI
protocol communication. A detailed and complete specification document describing the *USB-SPI Translator* board is included in Appendix section A *USB-SPI Translator Board Specification*.

The second board, *CS Dispatch* which is more specific for this application, fulfills the *Chips selection* and the *Feedback channel selection* sub-systems.

Both boards, *USB-SPI Translator* and *CS Dispatch*, need to communicate together. The detailed electrical connection for data transmissions between the two boards and the connector for the DC source board is shown in Figure 36.

![Electrical connections between the two boards](image)

**Figure 36** : Electrical connections between the two boards

### 5.7 PCB Design

In this sub-section the PCB designs are briefly described. The different PCB layouts are included in Appendix section C *PCB layouts*.

For convenience both PCBs size should be chosen precisely in order to be integrated to the DC source board. The different connectors are placed precisely in order to avoid complicate cables connection between the different boards.
The resulting hardware boards:

Figure 37 : USB-SPI Translator board

Figure 38 : CS Dispatch board (Top)

Figure 39 : CS Dispatch board (Bottom)

Figure 40 : USB-SPI Translator HW system
6 Software

This section describes the different firmware written for the different processing unit of the USB-SPI Translator system.

The different source codes are included in Appendix section E \textit{Source code}.

6.1 User interface

As the requirement specifies that the user shall be able to type in the information applied on the DC source board, an application providing a graphical user interface and a bi-directional USB data communication, has to be written. This sub-section will describe the \textit{USBFT245R} source code which is included in appendix section E \textit{Source code}.

The programming language used for the graphical user interface and the data transfer using the FTDI driver is C++ and the software used is \textit{Microsoft Visual Studio 2005}.

In order to realize the bidirectional communication between the computer and the FT245R USB controller implemented by the USB-SPI Translator system, a provided \textit{FTDI} driver should be used. \textit{FTDI} provides two types of driver, Virtual COM Port (VCP) drivers and direct (D2XX) drivers. The D2XX driver type is chosen for this application, it allows a direct access to a USB device via a DLL interface. For the bidirectional communication the application software can access the USB device through a series of DLL function calls given by \textit{FTDI} [3].

A software example given by \textit{FTDI} is used for the structure of the application. The application is divided in two sections, the graphical interface for the user and the bidirectional communication; both are implemented in the class \textit{Form1}. When the application is launched the startup routine (\textit{tWinMain}) is called. The startup routine will first create and enter in a single-threaded process. Then a new object of the class \textit{Form1} (graphical interface) is created, which calls the constructor of the class \textit{Form1}. The constructor class calls the \textit{InitializeComponent()} function that initialize the graphical interface described in Section 6.1.1 GUI.

6.1.1 GUI

The graphical interface (Figure 41) allowing the user to enter the desired information to be transmitted and displaying the received information shall provide following features:

- Display the connected device
- Types of transmission choice
- Data type in boxes
- Display the received data

As mentioned above, the initialization of the graphical interface is called by the constructor of the class \textit{Form1}. The initialization function instances a new object for each graphic form used and load the window. Follows, the different features provided by the graphical user interface are described.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{Figure_41.png}
\caption{GUI}
\end{figure}
Connected device information

It’s useful to know the name and the serial number of the plugged in devices. Thereby the wanted target device can clearly be chosen if several USB devices are plugged. Like the Figure 42 shows, a combo box display the selected device and radio buttons allow the user to choose the type of information to display the device.

When the application is launched the available devices are listed into the combo box. The user can manually update the device list by changing the display type. When a new device is added to the list this one is initialized by the host USB device (computer).

If a device is unplugged and plugged again, the user should update the list manually to be sure of the fully capability of the device to communicate with the computer. The same if the device is reset.

Types of transmission

Different types of transmission are needed for a well-working product. The user should be able to transmit the desired DACs output voltage of the DC source board but also transmit the final antenna angle.

As the Figure 43 shows, three different types of transmission are available for the user. For experimental tests the user can directly control the output voltage of the DACs. The user can choose between applying the same output voltage on each DAC or different output voltages for each DAC. The final purpose is to control directly the direction of the antenna by entering the angle on the computer. In order to be able to use this provided function, first the relation between the final angle, the phase shifter and the applied voltage has to be determined by experimental tests. Once the relation is known, the application will calculate the applied output voltage for each DAC matching the desired antenna angle.

Data type in

The user shall be able to type in the information to be sent. Text boxes are used to get the entered information by the user.

Like the Figure 44 shows, by selecting the transmission type the corresponding text boxes are enabled and the user can type in the information to be send. If the user chooses to update different voltage on each DAC, different voltage transmission type, a text box for each DAC is available. The user can update the desired DACs, just by type in the value in the DACs corresponding text boxes (V1…37). The entered value shall be with an accuracy of one decimal place (format X.X). If a box is empty when the information is sent, a no change value is sent (9.9) to inform the MCU that the DAC do not need to be updated.
Some calculations about the beam steering antenna (voltage-controlled) permitted to define the applicable values. The voltage range allowed to be applied on the phase shifter, consequently on the DACs output is between 0[V] and 5[V]. The angle change limitation of the antenna is 60[°], from -30[°] to +30[°]. By the calculations, each phase shifter voltage for each angle has been calculated. No function is available to calculate the DACs voltage for an angle. Therefore the results are included into the software as table (Angles/DACs voltage). When the user wants to update a new angle, the USB data packet is generated with the new DACs values corresponding to the angle.

Once the information to be send entered, the user can transmit the data forward the USB-SPI Translator board by pressing a provided Send button. The data transmission will be explained in section 6.1.2 Data transfer.

**Received data display**

Due to the communication with the USB-SPI Translator board is bi-directional the user shall be able to see the received data. A list box as shown in Figure 45, displays the incoming data from the USB-SPI Translator board. Every time new information is received, the list box is cleared and the new incoming data is displayed.

### 6.1.2 Data processing

The USB data transmission involves two tasks, sending and reading the data on the USB bus. By pressing the Send button the sending routine (buttonSend) is launched and the transmission is started.

**USB data generating**

Before any data can be sent, the data has to match some defined format in order to be correctly read by the receiver.

The chosen transmission type is determined and the typed in values are added to a buffer (cBuff). Before the read data is added to the buffer, the data value is checked. When the entered value isn’t in the authorized range, an information message informs the user of it. If the read box is empty a no change value (9.9) is added instead into the send buffer. With all the data, the software generates a USB data packet. Each USB data packet generated matches a format depicted in Figure 46.

<table>
<thead>
<tr>
<th>Header</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD</td>
<td>ToT</td>
</tr>
<tr>
<td>2 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>

**CMD**
- 10 (USB_SPI translator board)
- 01 (SPI data transmission)

**ToT**
- 00 (Single voltage update)
- 0C (Same voltage update)
- C0 (Direct angle update)

**Data1 to Data n**
- Data Bytes to process

*Figure 46 : USB data packet*
Composed of a header and a data section, the USB data packets have different transmission purposes. The header indicates the type of transmission and the data section represents the new DACs values.

The voltage values (data) are transmitted in ASCII (char) type and consequently each voltage value is represented by three char bytes. The relation between numerical and ASCII (char) value is shown in Figure 47 by an example:

<table>
<thead>
<tr>
<th>Type</th>
<th>Entered information</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerical</td>
<td>2.6</td>
<td>2</td>
<td>.</td>
<td>6</td>
</tr>
<tr>
<td>Ascii (char)</td>
<td>0x32</td>
<td>0x2E</td>
<td>0x36</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 47 : Data format**

**Data transmission**

For the data transmission two main routines are used, the *btnSend_Click()* function for the sending routine and the *ReadingProc()* function for the reading routine.

The send routine launched by the *Send* button is in charge for several tasks listed below:

- Get the user data information
- Generate a USB data packet with the data
- Create and start the reader thread
- Set the transmission parameters
- Write the data on the USB bus
- Display the incoming data

The read routine is launched by a separated specific reader thread. The reader thread is created and started by the send routine. Once the read routine started, the function wait (thread sleeping) for a notification event. A notification event is set when a character has been received and will unblock the read routine.

A sequence diagram showing a data transmission is shown in Figure 48.

**Figure 48 : Sequence diagram of transmission**

The Figure 48 shows that the two first above listed tasks are executed by the send routine when the reader thread doesn’t exist yet. When the USB data packet has been generated the send routine checks that a USB device is selected and ready to communicate. If no device is found, an advertisement message will inform the user.
In case one device is ready, the `startThread()` function is called by the send routine. The sub-routine creates the reader thread that execute the `readerProc()` function. After the thread was created and started it is put asleep for 1000ms allowing the send routine to continue.

In order to have a correct USB communication, a couple of settings have to be set first. The event notification condition is set up in order to set an event when a character has been received by the device. The speed transmission (baud rate) has to be set and the chosen baud rate for the bidirectional communication is 9600[bps].

When the transmission parameters are set, the provided DLL function `FT_Write()` is used to write on the USB bus. The `FT_Write()` function takes four parameters as follow:

- `ftHandle` : Status of the device.
- `IpBuffer` : Pointer to the buffer that contains the data to be written to the device.
- `dwBytesToWrite` : Number of bytes to write to the device.
- `IpdwBytesWritten` : Pointer pointing to the number of bytes written to the device.

If all parameters correct, the DLL function send the data to the USB controller device.

One second after the reader thread was created, the thread wakes up and launches the `readerProc()` function. The reader routine check if an event has occurred (data received). If not, the function blocks on the event and the send routine becomes active.

When some data is received a notification event is set and the reader thread is unblocked. The number of received bytes is found out by the `FT_GetQueueStatus()` function. The provided DLL function takes two parameters as follow:

- `ftHandle` : Status of the device.
- `IpdwAmountInRxQueue` : Pointer to the number of bytes in the receive queue.

A loop reads one by one every received byte. The read byte is converted to a string character and stored into a receive buffer (`buff`). The provided DLL function `FT_Read()` is used to read the incoming bytes. Four parameters are taken by the reading function:

- `ftHandle` : Status of the device.
- `IpBuffer` : Pointer to the buffer that receives the data from the device.
- `dwBytesToRead` : Number of bytes to be read from the device.
- `IpdwBytesReturned` : Pointer pointing to the number of bytes read from the device.

When all the received bytes are read and stored into the `buff` buffer, the reader routine will block on the event until some next data is received.

**Data displaying**

Due to the reader routine is a separated thread, the graphic objects aren’t available from the readingProc() function and the display of the data should be done by a sub-routine. The sub-routine reads the `buff` buffer and displays the data in the list box.
6.2 USB-SPI Translator

The processing unit for the USB-SPI Translator board is the ATmega88 microcontroller. This sub-section describes the embedded firmware written for the MCU. A detailed analyze of the needed software permitted to describe the firmware by flowcharts which have helped to write the source code. The resulting flowcharts of the analysis are included in appendix section D *MCU Firmware Flowcharts*.

As central control unit, the MCU has to communicate with the computer, the CS Dispatch board and the DC source board. The MCU communicates with the computer through the USB controller (FT245R) by FIFO parallel operation. For the CS Dispatch board and the DC source board the communication interface used is SPI protocol.

The MCU integrated in the USB-SPI Translator board is in charge for the following tasks:

- Startup
- Read USB data
- Received data analysis
- DAC update
- DAC feedback processing
- Write USB data
- Battery charging management

When the MCU gets supplied, the *main()* function of the program is launched. It calls the startup routine which is the *init()* method. All the register and I/O of the MCU are set at their initial values. After that the program enters in an infinite loop of data processing. The endless loop processes all of the listed tasks above but Startup. The loop checks if a flag for a task is set, if yes the software will execute the corresponding task. The flags are set by interrupt sub-routine (ISR) or by the tasks themself in order to execute another task. Follows, the different MCU tasks are discussed and depicted.

6.2.1 Startup

Like mentioned above the startup task is the initialization of the USB-SPI Translator board. The *init()* function is the first routine called by the switch on of the MCU. The status register (SREG), I/Os and all used MCU peripherals are initialized by setting the registers to their initial values.

*I/O*

All connected pins are either input or output signals. In order to control or read correctly a pin both, direction and the initial status, have to be set.

*TIMERS*

The ATmega88 microcontroller provides two 8-bits Timers (*TIMER0* and *TIMER2*) and one 16-bit Timer (*TIMER1*). *TIMER0* and *TIMER1* are used to execute respectively 1ms and 1s interrupt-sub routine.
The two timers are initialized as follows:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>TIMER0</th>
<th>TIMER1</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-bits timer [bits]</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Output timing [ms]</td>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>Clock prescaler</td>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>Output compare value (A register)</td>
<td>250</td>
<td>15625</td>
</tr>
<tr>
<td>Interrupt enable on output match</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

*Interrupt*

The ATmega88 device provides several different interrupt sources with each a separate program vector in the program memory space. When an interrupt occurs, the corresponding interrupt sub-routine (ISR) is triggered if the interrupt source is enabled. In addition to the two internal timers interrupt sources, the system uses two external interrupt sources. External interrupt occurs if one of the enabled pin toggles. Below the two external sources:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>RXF#</th>
<th>Button</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>PB6</td>
<td>PC2</td>
</tr>
<tr>
<td>Usage</td>
<td>USB Read</td>
<td>Shutdown</td>
</tr>
</tbody>
</table>

*SPI*

As mentioned above, the provided SPI interface is used as communication interface. The SPI interface of the MCU is initialized as follow:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>SPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation type</td>
<td>Master</td>
</tr>
<tr>
<td>Transmission speed</td>
<td>clk/2</td>
</tr>
<tr>
<td>2x speed enable</td>
<td>YES</td>
</tr>
<tr>
<td>SPI data mode</td>
<td>3</td>
</tr>
<tr>
<td>Data order</td>
<td>MSB first</td>
</tr>
<tr>
<td>End of transmission interrupt</td>
<td>YES</td>
</tr>
</tbody>
</table>

*ADC*

In order to have accurate analog-to-digital conversions, the built in ADC has to be initialized as follow:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>ADC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog source</td>
<td>ADC6 (feedback)</td>
</tr>
<tr>
<td>Voltage reference source</td>
<td>Extern (2.5[V])</td>
</tr>
<tr>
<td>Clock prescaler</td>
<td>8</td>
</tr>
<tr>
<td>End of transmission interrupt</td>
<td>YES</td>
</tr>
</tbody>
</table>
When the initialization is finished the data processing can start. Every time the microcontroller is reset, the initialization is executed firstly.

### 6.2.2 Read USB data

When the USB controller has data to transmit to the MCU the USB device pulls down the RXF# pin. As mentioned above the RXF# pin is an interrupt source and launch the corresponding ISR (`PCINT0_vect`) when the pin toggles. The `PCINT0_vect` ISR reads byte per byte the data from USB controller and stores the data into the read buffer (`tUSBRD`). The read function was written according to the time diagram [3] in Figure 49, representing a read cycle for one byte.

![Figure 49: USB data read timing diagram](image)

By each reading the MCU reads 1 byte of data on the port D and store it into the read buffer of type `char` (8 bits). As the voltage value send by the user are composed of two digits and a comma, one voltage value takes three bytes in the buffer. Before the received values are able to be used, the software has to transform the received value over three `char` into one value of type `float` that can be used further. The new translate value is stored into a new data buffer (`tDACWR`). The follow relations are used to reconstruct the received value into a `float` variable:

```plaintext
data = (float)tUSBRD[1] - 48;
```

To transform a char number into a numerical (int, float) value, the char value is subtract by 48 (ASCII). For the second relation, the value is divided by ten due to it’s the decimal digit. The transformation relation is shown in Figure 47 page 34 by an example.

When the data has been read, translate and stored into the `tDACWR` buffer, the `fUSBRD` flag is set.

### 6.2.3 Received data analysis

When the MCU has received data from the USB controller, the `fUSBRD` flag is set and thereby the analysis of the received USB data is executed.
The first byte of the received buffer (tUSBRD), which stands for the USB data header, is analyzed. As described in section 6.1.2 Data processing, the USB data header is composed of two parts, CMD (2 bits) and ToT (6 bits). The two CMD bits define if the transmitted data is configuration information for the USB-SPI Translator board or new information for the DC source board. The ToT bits represent the type of transmission the user chose.

If the USB data packet received is for a DC source board update, the type of transmission is determined (ToT). Knowing the type of transmission, the corresponding sub-routine is called in order to update the DACs with the received data. Each transmission type is processed differently and thereby a sub-routine for each type is implemented. However to write a new value on a DAC the sub-routine (SPI_DAC_WR) stays the same (see section 6.2.4 DAC writing). As explained further, to update a DAC, two parameters are used.

- Index : Number of DAC to update
- Data : Digital voltage value to update

Those two parameters are determined by the different transmissions sub-routines described below. As the entered information by the user is a numerical voltage value, the software has to determine the corresponding digital value to write to the DAC device. The New_DAC_Value() function returns the digital information that has to be written to the DAC. The relation to determine the digital input value for the digital-to-analog converters is as follow:

$$DAC = \frac{data \times 2^n}{2^Vref} + V_{DAC\ offset} \ ; \ n = 10, V_{ref} = 2.5[V], V_{DAC\ offset} = 3.6[V], Ampli = 5$$

$$(7)$$

Same Voltage Process

To update all the DACs with the same voltage, all the chips selection are pulled down. Thereby all the DACs read the SPI data line and are updated at the same time and only one data transmission is necessary. To select all the DACs, the chips selection value to send to the CPLD is $\text{index} = 50$ (refer to section 6.2.4 DAC update).

Single Voltage Process

When the user chose the different voltage transmission, he can chose to update every DAC, just a few DACs or only one. The sub-routine checks for each DAC if it has to be updated by the corresponding values. As explained in section 6.1.1 GUI, when the user chose a different voltage transmission, the empty left input boxes are set with a no change value. The sub-routine run through the new data buffer (tDACWR) and checks each value. When the initial value is matched, nothing is done and the next value is checked by incrementing the buffer index. If the value checked isn’t a no change value, the DAC update function (SPI_DAC_WR) is called. The parameters to update the DAC are:

- Index : Buffer index.
- Data : Data matching the buffer index.

Each time a DAC is to update, the SPI_DAC_WR routine is called with the correct parameters.
**Direct Angle Process**

When the user enters a direct angle in order to update the antenna direction, the DACs output voltages are calculated by the user interface application and send to the MCU. The direct angle process is similar to the different voltage process by the exception that every DAC is updated with a new value. Thereby no value check for a non-change value has to be performed.

- **Index**: Buffer index.
- **Data**: Data matching the buffer index.

The `SPI_DAC_WR` routine is called with the correct parameters for each DAC.

### 6.2.4 DAC update

In order to write a new value to a digital-to-analog converter device, the `SPI_DAC_WR()` function is called. The function takes two parameters as follow:

- **Index**: Number of DAC to update
- **Data**: Digital voltage value to update

As mentioned in section 6.2.3 Received data analysis, the `SPI_DAC_WR()` function is called by the transmission type sub-routine in order to update the DACs. Figure 50 represent a DAC update cycle.

![Figure 50: DAC update time diagram](image)

To write a new value to the DAC, first the chip selection number (Index) has to be transmitted to the CPLD. As shows Figure 50, the first SPI transmission is the index value to the CPLD. This way, the CPLD can select the DAC that has to read the new value. When the DAC is selected the data can be sent to the DAC. Due to the DACs are 10-bits converters, two SPI transmissions (8-bits) are necessary to update a new value. The data format of the 16-bits transmission is represented below:

<table>
<thead>
<tr>
<th>D15</th>
<th>D14</th>
<th>D13</th>
<th>D12</th>
<th>D11</th>
<th>D10</th>
<th>D9</th>
<th>D8</th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>SPD</td>
<td>PWR</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **SPD**: Speed control bit (0 – Normal mode)
- **PWR**: Power control bit (0 – Normal operation)
When the device has been enabled with its chip selection (CS_N) set low, a falling edge of FS_N starts shifting the data bit by bit on the falling edges of SCLK, starting with the MSB, into the internal register of the DAC device.

In order to inform the CPLD that the DACs update is finished, a 0x00 index transmission is performed. Thereby the CPLD disables all the DAC from reading by pulling up all the CS_N signals (see section 6.3.1 Firmware).

### 6.2.5 DAC feedback processing

The built in analog-to-digital converter is used to convert the output voltage of the DACs. A DAC feedback update happened when the fFB flag is set. The TIMER1 ISR set the fFB flag by the set frequency. The frequency can be controlled by one second precision as the TIMER1 ISR is executed every second.

When the fFB flag is set, the FB_Process() is started and a new DAC data feedback update is executed.

Figure 51 shows a DAC data feedback cycle. First the CPLD is informed of a feedback update by transmitting the index = 0x64. The CPLD selects the feedback data source (DAC) to convert (see section 6.3 CS Dispatch). By strobbing the CS signal, the CPLD increment the feedback data source (DAC). For each feedback data source (DAC) an analog-to-digital conversion is done and the obtained digital value of the DAC output voltage is stored into a buffer (tFBData). Like the DACs, the built in ADC is a 10-bits converter; thereby the conversion results are using two 8-bits buffer inputs.

When all the DACs have been converted, a USB data packet has to be generated with the stored data into the tFBData buffer. Each digital result (ADC) has to be translated into a numerical value (format X.X) according to the following relation in order to send the information back to the computer:

\[
V_{DAC} = \frac{ADC \times V_{ref} \times AMPLI}{1024} - V_{offset} \quad ; \quad V_{ref} = 2.5[V], AMPLI = 5, V_{offset} = 3.5[V] \tag{8}
\]

To generate the USB data packed, each DAC output value has to fit with the data format described in section 6.1.2 Data processing (see Figure 47). To generate the data packet, which are stored into the USB write buffer (tUSBWR), the numerical value has to be translated into char values (3 bytes) like described above in the Read USB data sub-section. Below are shown the used relation:
To have a complete and ready USB data packet to be sent, the USB data header has to be set in the first position of the USB write buffer ($t_{USBWR}$).

### 6.2.6 Write USB data

To write the generated USB data packet contained into the $t_{USBWR}$ buffer, the $USB_{WR}_{\text{Data}}_{\text{Process}}()$ function is used. The USB data write routine has been written according to the USB data write cycle represented in Figure 52.

![Figure 52: USB data write time diagram](image)

Data can only be written into the USB controller when the $TXE#$ signal is low. When the $WR$ signal goes from high to low, the data byte set on the port D is written into the USB device. When a writing cycle is over the next byte is set on the port D for the next write cycle. When the MCU has finish to write the data into the USB controller, the port D has to be set back as input in order to be able to read future incoming USB data packet.

### 6.2.7 Battery charging management

As the MCU is used to control the recharge of the battery, the built in AD converter is used to have a feedback information of the battery charge. Every five second the $fBAT$ flag is set by the $TIMER1$ ISR and the battery management routine is executed ($Battery_{Process}$).

The battery source is connected to the MCU through the analog ADC7 pin. Before the conversion can starts the battery source channel (ADC7) has to be selected. To have a more accurate conversion the battery shouldn’t be recharging during the conversion, thereby the charge signal is set high. Then a conversion of the battery voltage level is performed by the AD converter. In order to check the battery level, the conversion result has to be translated into a numerical format as follow:
As explained in section 5.2.2 MCU (ADC), the battery is connected to the analog input through a voltage divider. Before the numerical battery level can be checked the real battery level has to be calculated as follow:

\[ V_{FB\text{bat}} = \frac{ADC \cdot V_{ref}}{1024} \quad ; \quad V_{ref} = 2.5[V] \]  

\[ V_{bat} = V_{FB\text{bat}} \cdot \frac{RBAT2}{RBAT1} \quad ; \quad RBAT1 = 300, \quad RBAT2 = 200 \]  

If the battery level is beneath the low threshold, the charge signal is set low in order to recharge the battery. While the battery is recharging or if no external source is available, in this case has to be recharged, the battery LED is lighting. When the battery level exceeds the high threshold, the charge signal is set high and the battery stops recharging and the battery LED is switch off. If the battery level is between the thresholds a variable with the last recharging status is checked and if the battery was recharging, the charge signal is set back low. If the battery wasn’t recharging the charge signal is kept high.
6.3 CS Dispatch

The CS Dispatch board uses a complex programmable logic device (CPLD) as processing unit. The development software *HDL Designer* from Mentor Graphics is used. HDL Designer is a project manager based on VHDL or Verilog languages that can automatically generate VHDL text from graphs (Block diagrams, state diagrams). The firmware is written in VHDL.

The three tasks the CPLD has to fulfill are:

- Read MCU data
- Chips selection
- Feedback channel selection

When the MCU transmit some information to the CPLD, the device analyzes the received data and then executes the request.

6.3.1 Firmware

As HDL Designer allows graphics design, the top level of the firmware is described with block diagrams (module). Figure 53 shows the TOP level architecture.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>Signal</td>
<td>1</td>
<td>Input</td>
<td>Clock signal</td>
</tr>
<tr>
<td>DIN</td>
<td>Signal</td>
<td>1</td>
<td>Input</td>
<td>SPI data input (MOSI)</td>
</tr>
<tr>
<td>CS</td>
<td>Signal</td>
<td>1</td>
<td>Input</td>
<td>Chip selection (CPLD) for SPI transmission</td>
</tr>
<tr>
<td>DOUT</td>
<td>Signal</td>
<td>1</td>
<td>Output</td>
<td>SPI data output (MISO)</td>
</tr>
<tr>
<td>CS_N</td>
<td>Bus</td>
<td>37</td>
<td>Output</td>
<td>DAC selection</td>
</tr>
<tr>
<td>FB_Addr</td>
<td>Bus</td>
<td>6</td>
<td>Output</td>
<td>Feedback channel selection</td>
</tr>
<tr>
<td>dbus0</td>
<td>Bus</td>
<td>6</td>
<td>Intern</td>
<td>Numerical chip to select</td>
</tr>
<tr>
<td>FB</td>
<td>Signal</td>
<td>1</td>
<td>Intern</td>
<td>Start a feedback update signal</td>
</tr>
</tbody>
</table>
The buses and signals used by the CS Dispatch TOP level are described above. The implemented modules are described in the following sub-sections.

**MCU communication (SPI)**

By pulling down the CS signal the MCU start a SPI transmission with the CPLD. The SPI module is in charge to read the incoming 8-bits data from MCU.

On each falling edge of CLK (clock) the DIN pin is read and the value stored into an internal 8-bits bus. With CS going high the transmission is over and the SPI module determines the request to execute. If the received data is **0x64** (100), an impulsion on the FB signal executes the feedback channel selection module. If not, the received data stand for the DAC to select and therefore the data is transmitted to the chips selection module through the internal dbus0 bus.

**Chips selection**

By sending a numerical number between 0 and 50, the MCU wants the CPLD to select the DAC by pulling down the corresponding CS_N[1…37] signal matching the transmitted value. A switch/case structure is used to select the DAC signal. The received value/DAC relations are represented in the table beneath.

<table>
<thead>
<tr>
<th>MCU data</th>
<th>Numerical value</th>
<th>DAC to select</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>0x01</td>
<td>1</td>
<td>CS_N1</td>
</tr>
<tr>
<td>0x02</td>
<td>2</td>
<td>CS_N2</td>
</tr>
<tr>
<td>0x03</td>
<td>3</td>
<td>CS_N3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x23</td>
<td>35</td>
<td>CS_N35</td>
</tr>
<tr>
<td>0x24</td>
<td>36</td>
<td>CS_N36</td>
</tr>
<tr>
<td>0x25</td>
<td>37</td>
<td>CS_N37</td>
</tr>
<tr>
<td>0x32</td>
<td>50</td>
<td>All CS_N[1…37]</td>
</tr>
<tr>
<td>0xFF</td>
<td>255</td>
<td>FB channel selection</td>
</tr>
</tbody>
</table>

**FB channel selection**

If the value 0xFF is received the SPI module generates an impulsion on the FB signal that starts a DAC output feedback update. The FB channel selection module is in charge to select the DAC that the output voltage has to be connected to the analog input of the MCU in order to be converter. To fulfill the selection the FB_Addr bus represents the address of the DAC to be select. The first DAC to be update is the device number one. By pulling down and high the CS signal the MCU increment the selected DAC (see Figure 51).

### 6.3.2 Simulation

Mentor Graphics provides software able to simulate hardware description language like VHDL. The simulation software called ModelSim XE is used to simulate the written VHDL code.

A test bench is implemented in order to generate the input signals and to get the output signal and buses. The table below shows the input signals generated by the test bench for the simulation.
<table>
<thead>
<tr>
<th>Input</th>
<th>Generated signal</th>
<th>Frequency/Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>Clock</td>
<td>500Hz</td>
<td>SPI clock signal from MCU</td>
</tr>
<tr>
<td>DIN</td>
<td>Pulses</td>
<td>0x09, 0x1C, 0x14, 0x00, 0x64</td>
<td>Chips selection (DAC update) Feedback channel selection</td>
</tr>
<tr>
<td>CS</td>
<td>Pulses</td>
<td>≤ 8 clock pulses, 37 pulses</td>
<td>SPI chip selection (CPLD) Incremental signal for FB update</td>
</tr>
</tbody>
</table>

Two simulations are done, one testing the *Chips selection* function and one testing the *Feedback channel selection*.

**Figure 54**: Chips selection simulation

Figure 54 shows the simulation result for the Chips selection module. The simulation shows clearly that after received the data (white rectangle) the corresponding DAC is selected (white dashed rectangle).

**Figure 55**: FB channel selection simulation
The simulation in Figure 55 shows that by receiving the value 0x64 from MCU, the CPLD generate a pulse on the FB signal and thereby start a FB channel selection. For each falling edge of CS, when flag is high (DAC FB update), the feedback channel is incremented.

### 6.3.3 Synthesis

This sub-section describes how Xilinx ISE Design is used for the synthesis and the upload of the CS Dispatch firmware into the CPLD.

When the VHDL code is uploaded into the CPLD device, the pins have to match with the input and output signals. If nothing is done, Xilinx allocate the inputs and outputs randomly to the pins. Precision Synthesis is a software allowing to set synthesis constraint, like pin allocation or timing constraints. The following pin allocation is set for the synthesis according to the schematic (see Appendix section A Schematic):

<table>
<thead>
<tr>
<th>Input / Output</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>22</td>
</tr>
<tr>
<td>DIN</td>
<td>6</td>
</tr>
<tr>
<td>CS</td>
<td>8</td>
</tr>
<tr>
<td>DOUT</td>
<td>7</td>
</tr>
<tr>
<td>CS_N [0:36]</td>
<td>35,36,37,39,40,41,42,43,49,50,52,53,54,55,56,58,59,60,61,63,64,65,66,67,68,70,71,72,73,74,76,77,78,79,80,81,82</td>
</tr>
<tr>
<td>FB_Addr [0:5]</td>
<td>11,12,13,14,15,16</td>
</tr>
</tbody>
</table>

The *.ucf file generated, containing all the synthesis rules, is included to the Xilinx ISE project in order to run the synthesis with the correct pin allocation.
7 Experimental results

In this section the realized experimental tests are described. First the USB-SPI Translator system is tested. Finally experimental tests about the whole system, including the DC voltage control beam steering metamaterial antenna, are done to prove the well-working of the product.

For the USB-SPI Translator board, hardware and software have been tested.

7.1 Power management

7.1.1 Power selector

In order to test the power selector circuitry, the battery outgoing current flow is measured.

As Figure 56 (a) shows, the current drawn from the battery is null when external power supply is available. However if no external supply is available, Figure 56 (b) shows that current is drawn from the battery in order to supply the system. The test demonstrates that the power selection works and that the battery is able to supply at least the data processing unit (MCU).

7.1.2 Power consumption

To have a precise idea of the power consumption, two kinds of measurements are done, with external supply and by battery supply. For the tests the whole USB-SPI Translator system is connected.
External supply

![Image](72x522 to 539x697)

(a)          (b)

Figure 57: Power consumption by external supply (a) switched off, (b) switch on

\[ I_{\text{switch off}} = 132[\mu A] \quad I_{\text{switch on}} = 104.4[\text{mA}] \]

Battery supply

![Image](72x246 to 539x421)

(a)          (b)

Figure 58: Power consumption by battery supply (a) switch off, (b) switch on

\[ I_{\text{switch off}} = 143[\mu A] \quad I_{\text{switch on}} = 10.98[\text{mA}] \]

Figure 57 and Figure 58 (a) show that even when the system is shut down a small current is drawn. The user should be able to switch on the system by a push button. In order to fulfill the requirement a small electronic is needed to switch on the system and consequently a small current is drawn.

Figure 58 (b) shows that the battery capacity is too small to supply the whole system, but enough to supply the data processing unit (MCU) and keep safe the data.
7.1.3 Inputs protection

In order to test the input protection system, an external voltage generator is used to apply the different voltage on the input connector. The voltage range tested is from 0[V] to 25[V]. The results of the experiment are presented in the table below:

<table>
<thead>
<tr>
<th>Applied voltage [V]</th>
<th>MCU Voltage [V]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Data</td>
</tr>
<tr>
<td>0.51</td>
<td>0.51</td>
<td></td>
</tr>
<tr>
<td>1.10</td>
<td>1.10</td>
<td></td>
</tr>
<tr>
<td>1.65</td>
<td>1.65</td>
<td></td>
</tr>
<tr>
<td>2.05</td>
<td>2.05</td>
<td></td>
</tr>
<tr>
<td>2.37</td>
<td>2.37</td>
<td></td>
</tr>
<tr>
<td>2.50</td>
<td>2.49</td>
<td></td>
</tr>
<tr>
<td>2.90</td>
<td>2.88</td>
<td>No use</td>
</tr>
<tr>
<td>3.30</td>
<td>3.12</td>
<td></td>
</tr>
<tr>
<td>3.60</td>
<td>3.29</td>
<td></td>
</tr>
<tr>
<td>5.03</td>
<td>3.83</td>
<td></td>
</tr>
<tr>
<td>10.02</td>
<td>4.20</td>
<td></td>
</tr>
<tr>
<td>15.01</td>
<td>4.42</td>
<td></td>
</tr>
<tr>
<td>25.07</td>
<td>4.51</td>
<td>Over voltage</td>
</tr>
</tbody>
</table>

The experiment shows that under 3[V] the diode doesn’t drive and the applied voltage is directly applied on the MCU. Above 3[V] the diode starts to drive and consequently a voltage drop appears on the resistance. For voltages over 5[V] the diode drives fully and the voltage is low down by the resistance.

The experiment shows that the inputs protection work and that the MCU is protected.

7.1.4 Battery management

By measuring the battery voltage level and the charge signal from MCU, the battery management system can be tested. For the tests, the low threshold is set at 3[V] and the high threshold at 3.6[V].

Figure 59 : Battery management (a) start of recharging signal, (b) battery charging current
\[ I_{\text{charge}} = -865[\mu\text{A}] \]

Figure 59 shows the charge signal (yellow) and the battery voltage level (bleu). When the battery level is converted by the MCU and the voltage level is under the low threshold, the charge signal is set low and the battery is being recharged. Figure 59 (b) shows the battery current. If the current is positive, the battery is supplying the system. In the opposite, if the current is negative, the battery is recharged. When the battery is recharged the second LED is lighting (see zoom in Figure 59 (b)).

![Figure 59: Charge signal and battery voltage level](image)

Figure 60: Battery stops recharged

Figure 60 shows the battery reaching the high threshold and consequently the charge signal (yellow) is set high and the battery stop recharging.
7.2 Data transmissions

For the data transmission, the three types of transmission are tested. For the following data transmission pictures, yellow signals represent the SPI clock and the bleu signals the SPI data signal. As the DACs have an offset of 3.5[V], the output voltage differs with the entered user information. Therefore the offset isn’t applied for the data transmissions tests, in order to verify that the data sent correspond to the typed in value. For the global system experiment (see section 7.3 USB-SPI Translator system) the offset is set back.

7.2.1 Same voltage

Index: 0x50 => All DACs; Data: 0x0428 => new value: 0x10A =>

\[ V_{DAC} = \frac{0x10A + 2 \times V_{ref} \times 15}{1024} = 6.5[V] \]

For the same voltage transmission, only one transmission is needed. As Figure 61 shows, the index number sent to the CPLD in order to select all the DACs is \textbf{0x50}. The new value transmit to the DACs is \textbf{0x10A} which correspond to \textbf{6.5[V]}.

Figure 61 : Same voltage transmission (a) GUI, (b) 6.5[V] data transmission
### 7.2.2 Different voltage

**Complete DACs update**

![Image of Complete DACs update]

Figure 62: Different voltage transmission for each DAC (a) GUI, (b) all data transmission, (c) DAC 6 data transmission

Index: 0x06 => **DAC 6**  ;  Data: 0x0104 => new value: 0x41 => \( V_{DAC} = \frac{0x41 \times 2 \times V_{ref} \times 5}{1024} = 1.6[V] \)

**Single DAC update**

![Image of Single DAC update]

Figure 63: Single voltage transmission (a) GUI, (b) all data transmission, (c) DAC 12 data transmission

Index: 0x0C => **DAC 12**  ;  Data: 0x023C => new value: 0x8F => \( V_{DAC} = \frac{0x8F \times 2 \times V_{ref} \times 5}{1024} = 3.5[V] \)
Few DACs update

![Image](image1.png)

(a) Few voltages transmission (4 DACs) (a) GUI, (b) all data transmission, (b) DAC 31 data transmission

Index: 0x1F => DAC 31; Data: 0x026C => new value: 0x9B = \( V_{DAC} = \frac{0x9B \times 2 + V_{ref} \times 5}{1024} = 3.8[V] \)

7.2.3 Direct angle

![Image](image2.png)

(a) Direct angle transmission (-21°) (a) GUI, (b) all data transmission, (c) DAC 24 data transmission

Index: 0x17 => DAC 24; Data: 0x02C0 => new value: 0xB0 = \( V_{DAC} = \frac{0xB0 \times 2 + V_{ref} \times 5}{1024} = 4.3[V] \)

The direct angle transmission is more specifically tested in the section 7.3 USB-SPI Translator system.
7.3 USB-SPI Translator system

In order to test the whole system, and at the same time the Chips selection (CPLD) function, the USB-SPI system (USB-SPI Translator and CS Dispatch boards) with the DC source board (DACs) is tested in a global experiment.

7.3.1 Tests setup

![Global test setup](image)

7.3.2 DACs update

The *chips selection* function is successfully tested by using the *different voltage* type transmission. In order to completely test the USB-SPI Translator system, the *direct angle* transmission type is used. By that type of transmission every DACs should be updated with a specific value.

Two *direct angle* transmissions are tested, by measuring the DACs output the results are shown below:

*Angle -21 °*

<table>
<thead>
<tr>
<th>DAC</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theory</td>
<td>6.8</td>
<td>6.6</td>
<td>6.4</td>
<td>6.2</td>
<td>6.0</td>
<td>5.6</td>
<td>5.2</td>
<td>4.8</td>
<td>4.2</td>
<td>8.1</td>
<td>7.7</td>
<td>7.3</td>
<td>7.1</td>
<td>6.8</td>
<td>6.6</td>
<td>6.4</td>
<td>6.2</td>
<td>6.0</td>
</tr>
<tr>
<td>Measure</td>
<td>6.8</td>
<td>6.6</td>
<td>6.4</td>
<td>6.2</td>
<td>6.0</td>
<td>5.6</td>
<td>5.3</td>
<td>4.9</td>
<td>4.3</td>
<td>8.0</td>
<td>7.6</td>
<td>7.3</td>
<td>7.1</td>
<td>6.8</td>
<td>6.6</td>
<td>6.4</td>
<td>6.3</td>
<td>5.9</td>
</tr>
</tbody>
</table>
The experiment shows that the DACs are successfully updated.

### 7.3.3 DACs Feedback

As the DC source board has not anticipated connectors for the DACs feedback, a few wires are sold to the DACs output as shown in Figure 66. Five DACs output connected to the CS Dispatch board are enough to test the feedback processing. DAC 9, 10, 11, 12, 13 are connected.

![Figure 66: DACs feedback connector](image)

To test the feedback function, different values are sent to the connected DACs. The follow relation is used to check the feedback value:

\[
V_{\text{applied}} = V_{\text{feedback}} \times \text{Ampli} - V_{\text{offset}}
\]

\[
r_9 = 1.4 \times 5 - 3.5 = 3.5[V]
\]

\[
r_{10} = 1.3 \times 5 - 3.5 = 3.0[V]
\]

\[
r_{11} = 1.3 \times 5 - 3.5 = 3.0[V]
\]

\[
r_{12} = 0.8 \times 5 - 3.5 = 0.5[V]
\]

\[
r_{13} = 0.9 \times 5 - 3.5 = 1.0[V]
\]

The obtained results show a lack of precision, however, the feedback process converts the DACs output, generate a USB data packet and sent it to the user interface which displays it. Some random value appears due to the feedback pins are not connected. The value -52.8 stand for no data received. Because only five DACs are connected, not all DACs converted values are sent to the user interface.
7.4 Beam steering experiment

7.4.1 Experiment setup

The beam steering experiment is set up according to the block diagram represented in Figure 2.

With the user interface, the desired angle to test is set by applying the correct voltages on the antenna (Beam steering antenna). The source signal coming from the patch antenna is deflected by the beam steering antenna in the good direction. A receiver antenna detects the source signal and stores the gain of the signal into a computer during the measurement. By rotating the antenna by 180 degrees, the gain detected by the receiver antenna varies. The maximal obtained gain represents the source antenna direction angle. The results are described in section 7.4.2 Result.

7.4.2 Result

As mentioned above, to measure the beam angle of the source antenna, the antenna is rotated about 180 degrees while transmitting. For each rotation (1 degree steps), a computer stores the gain value detected by the receiver antenna measured by a vector network analyzer. MatLab is used to represent the obtained gain values by the corresponding beam angle. The highest gain represents the beam angle of the source
antenna. If the set angle match with the obtained result, the transmission direction of the source antenna is correct.

![Beam steering experiment results](image)

**Figure 69: Beam steering experiment results**

Five different angles (angle set: 0, 10, -10, 20, -30) have been tested. Figure 69 shows the five curves for the five tested angle. The five maximum gain values are pointed out with arrows. Angles 0, 10 and -10 have strong main lobe and small side lobes, and the positions of the maximum gain (main lobe) are matched well with the ones we set. The side lobes of angles 20 and -30 are little larger, but we can still recognize the main lobes with matching position. Therefore, the experiment is successful.
8 Conclusion

According to the sections 2 Objectives and 3 Requirements, a conclusion about the USB-SPI Translator system describes the status of the system and the improvements for future works.

8.1 USB-SPI Translator status

The USB-SPI Translator system is composed of two boards, USB-SPI Translator and CS Dispatch, and the graphical user interface application. For the graphical user interface, a setup file allows to install the GUI application on computers. The hardware has a final size of 16.7[mm] by 3.6[mm] and possesses connectors to connect the 37 DACs.

Power management

The USB-SPI Translator system is capable of selecting the power source in charge to supply the system. The first used source, if available, is the external power source. However if no external power source is available anymore, the system is self-powered by the battery. When the board is self-powered, with the experimental battery, only the data procession unit (MCU) can be supplied. The power consumption by external supply is up to 104[mA] in active operation.

By low battery level detection the system is able to recharge the battery when the external power source is available.

DACs update

The USB-SPI Translator system is able to control the DC source board by using the USB-SPI Translator application (user interface). Once the software installed (USB-SPI_Translator_setup.exe) on the computer, the user has the choice between controlling the output voltage of each DAC and controlling the resulting angle of the voltage-controlled antenna.

The user can update every single DACs output voltage by choosing the different voltage transmission. By entering a value into the corresponding DACs input box, the user can update one, a few or every DACs. If the same voltage transmission is chosen, the user can update all the DACs with the same voltage at the same time.

As final purpose of the application, the user can control the resulting angle of the antenna by choosing the direct angle transmission. For each angle the system updates the 37 DACs output voltage in order to control the antenna direction.

An USB data packet is generated with the new voltage values according to the entered information by the user. Using the provided DLL function by FTDI, the USB data packet generated is sent to the USB controller (FT245R).

By receiving data from the computer, the USB controller starts a communication with the MCU (Atmega88) in order to transmit the data to the MCU by parallel operation. The received data is analyzed by the processing unit to identify the data transmission type. Each DAC is updated with the corresponding received information from the user interface. The DAC going to be updated is selected by the CPLD,
which has received the information from MCU. With the DAC selected by the CPLD, the MCU working as SPI master device sends the new digital values to the DACs (SPI slave devices).

**DACs feedback processing**

The MCU starts the feedback processing by informing the CPLD of a new DACs feedback processing. The CPLD selects the first DAC and the MCU execute the conversion of the first DAC output voltage. By pulling down and high the CS signal the next DAC is selected by the CPLD and then converted by the MCU. With all the DACs feedback information, the MCU generates an USB data packet that is sent to the USB Controller.

When data is available into the USB controller, the USB host (PC) reads the data from USB controller device and store it into a buffer. Every time new information is read from USB controller, the buffer is cleared and the new data are set in. By each end of transmission, the list box is updated with the received buffer values.

### 8.2 Improvements

**External Clock**

Currently the internal clock of the MCU is used for the data processing unit. By using an external clock generator, a more precise clock could be used to increase the analog-to-digital conversions precision. External clock generator working at high frequency would allow faster operation and thereby have faster updates of the DACs.

**Battery capacity**

The currently implemented battery is able to supply the data processing unit. By increasing the battery capacity the system would be better supplied and the self-powered life time increased.

**Configuration**

The USB data packet format allows different type of transmission. Beside the DACs update and DAC feedback transmissions, system configuration packet transmission could be implemented. By this function the user might chose to enable/disable the DAC feedback function, control the power operation mode of the system (active/sleep) and many other features.

**Wireless Communication**

By now a USB cable connect the computer with the USB-SPI Translator board. In order to avoid the excess of wire and the diminution of mobility, an RF communication could be implemented for the computer to USB-SPI Translator board communication.
9 References

RD [1] Diploma work specification document
Appendix A – Board specification

USB-SPI Translator
USB-SPI Translator

[ELECTRONIC BOARD SPECIFICATION]
# TABLE OF CONTENTS

1 REFERENCES .................................................................................................................. 3

2 TERMS, DEFINITIONS AND ABBREVIATED TERMS .................................................. 3

2.1 ABBREVIATED TERMS ............................................................................................ 3

2.2 DEFINITION ............................................................................................................. 3

3 INTRODUCTION ............................................................................................................ 3

4 FUNCTIONAL SPECIFICATION .................................................................................. 4

4.1 DATA TRANSFER .................................................................................................... 4

  RQ4005: USB IF number .............................................................................................. 4
  RQ4010: Data management ......................................................................................... 4
  RQ4015: SPI IF number ............................................................................................. 4
  RQ4020: USB to SPI .................................................................................................... 4
  RQ4025: MCU - USB to SPI ....................................................................................... 4
  RQ4030: USB received data packet ............................................................................ 4
  RQ4035: SPI channel selection .................................................................................. 5
  RQ4040: SPI to USB .................................................................................................... 5
  RQ4045: MCU - SPI to USB ....................................................................................... 5
  RQ4050: USB transmitted data packet ....................................................................... 6
  RQ4055: MCU ........................................................................................................... 6

4.2 FUNCTIONAL MODES .............................................................................................. 6

  RQ4060: system modes ............................................................................................... 6
  RQ4065: Modes transition ......................................................................................... 7
  RQ4070: “OFF” mode ................................................................................................ 7
  RQ4075: “INITIALISATION” mode ............................................................................. 7
  RQ4080: “OPERATIONAL” mode ............................................................................. 7
  RQ4085: “STAND-BY” mode .................................................................................... 7
  RQ4090: external wakeup signal .............................................................................. 7

4.3 COMMAND AND HOUSEKEEPING ....................................................................... 7

  RQ4095: Board switch on ......................................................................................... 7
  RQ4100: Board reset .................................................................................................. 7
  RQ4105: Status information ...................................................................................... 8
  RQ4110: “POWER ON” information ...................................................................... 8
  RQ4115: “TRANSMITTING/RECEIVING DATA” information ................................... 8
  RQ4120: debug support ............................................................................................ 8

5 INTERFACE .................................................................................................................... 8

5.1 MECHANICAL ........................................................................................................... 8

  RQ5005: Board size .................................................................................................. 8
  RQ5010: Board fixing ................................................................................................ 8

5.2 ELECTRICAL ............................................................................................................ 8

  5.2.1 POWER ............................................................................................................. 8

    RQ5015: Power plug .............................................................................................. 8
    RQ5020: Power pack ............................................................................................ 8
    RQ5025: Power source ......................................................................................... 8
    RQ5030: Voltages .................................................................................................. 8
    RQ5035: 3.3V voltage ............................................................................................ 9
    RQ5040: Operational mode ................................................................................... 9
    RQ5045: Standby mode ....................................................................................... 9

  5.2.2 USB ................................................................................................................. 9

    RQ5050: USB connector ..................................................................................... 9
    RQ5055: USB signals ........................................................................................... 9
    RQ5060: USB performance ............................................................................... 9

  5.2.1 SPI ................................................................................................................. 9

    RQ5065: SPI connector ..................................................................................... 9
    RQ5070: SPI signals ............................................................................................ 9
    RQ5075: SPI master device ............................................................................... 10
    RQ5080: SPI slave devices .............................................................................. 10

  5.2.2 Debug ............................................................................................................ 10

    RQ5085: Debug interface ................................................................................... 10
1 REFERENCES

RD [1] Diploma work specification document

2 TERMS, DEFINITIONS AND ABBREVIATED TERMS

2.1 Abbreviated terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART</td>
<td>Universal asynchronous receiver transmitter</td>
</tr>
<tr>
<td>FTDI</td>
<td>Future Technology Devices International</td>
</tr>
<tr>
<td>SPI</td>
<td>Serial Peripheral Interface</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller</td>
</tr>
<tr>
<td>LED</td>
<td>Lighting emitting diode</td>
</tr>
</tbody>
</table>

2.2 Definition

FT245R FT245R is an USB controller which translates the USB data to serial UART interface.

3 INTRODUCTION

This document describes the specification of a USB to SPI electronic card. The specification defines what the card shall do. It will be used to design, fabricate and debug the electronic card which will be used for the USB-SPI Translator.

The purpose of this card is to translate USB data into standard SPI data and SPI data into USB data in order to manage the communication between a computer and a sub-system of a steering phased array antenna. (refer to RD [1]). As we need 37 chip select, the CS signal will be perform by an external board “CS_Dispatch_Board”. This external board shall perform the chip selection signal for each DAC chips.
4 Functional Specification

4.1 Data transfer

**RQ4005: USB IF number**
The card shall provide one USB link.

**RQ4010: Data management**
The card shall implement a USB controller FT245R and one MCU ATmega88 for data management purpose. Communication between these two components will be managed by parallel operation.

**RQ4015: SPI IF number**
The card shall provide two standard SPI links.

**RQ4020: USB to SPI**
The card shall translate received USB data packets into SPI data packets as presented in the figure below:

![USB to SPI data transmission](image1)

**Figure 1: USB to SPI data transmission**

**RQ4025: MCU - USB to SPI**
Each received USB data packet is translated into one SPI packet for one SPI channel.

**RQ4030: USB received data packet**
The card shall manage the USB data packet coming from the computer. These packets have the following format.

<table>
<thead>
<tr>
<th>Header</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD</td>
<td>SPI CS</td>
</tr>
</tbody>
</table>

2 bits 6 bits 0-1022 Bytes

![SPI data transmission packet](image2)

**Figure 2: SPI data transmission packet**

CMD : 01 (SPI data transmission)
**SPI CS** : SPI channel selection  
*Data1 to Data n*: Data Bytes to process

### USB_SPI translator board configuration

<table>
<thead>
<tr>
<th>Header</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD</td>
<td>SPI CS</td>
</tr>
<tr>
<td>2 bits</td>
<td>6 bits</td>
</tr>
<tr>
<td></td>
<td>Data 1 to Data n</td>
</tr>
<tr>
<td></td>
<td>0-1022 Bytes</td>
</tr>
</tbody>
</table>

**Figure 3**: USB_SPI translator board configuration packet

**CMD** : 10 (USB_SPI translator board)  
**SPI CS** : SPI channel selection  
*Data1 to Data n*: USB_SPI translator configuration data

**RQ4035: SPI channel selection**

The card shall select the SPI channel by using the header of the received USB data packet. The selected SPI channel shall be forward to the **CS Dispach Board**.

**RQ4040: SPI to USB**

The card shall translate the received SPI data packets into USB data packets according to the following figure:

**RQ4045: MCU - SPI to USB**

The MCU shall be able to receive data from different SPI channel and to generate for each SPI channel an USB data packet that will be send to the computer.
**RQ4050: USB transmitted data packet**

The card shall generate USB data packets according to the following formats.

**SPI data transmission**

<table>
<thead>
<tr>
<th>Header</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>SPI CN</td>
</tr>
</tbody>
</table>

| 2 bits | 6 bits | 0-1022 Bytes |

*Figure 5: SPI data transmission packet*

- **Source**: 01 (SPI data transmission)
- **SPI CN**: SPI channel number
- **Data1 to Data n**: Data Bytes from SPI port

**USB_SPI translator housekeeping**

<table>
<thead>
<tr>
<th>Header</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Type</td>
</tr>
</tbody>
</table>

| 2 bits | 6 bits | 0-1022 Bytes |

*Figure 6: USB_SPI translator housekeeping packet*

- **Source**: 10 (USB_SPI translator board)
- **Type**: 000001 (status) 000010 (…)
- **Data1 to Data n**: USB_SPI translator housekeeping data

**RQ4055: MCU**

The card shall implement a MCU which provide master/slave SPI interfaces.

### 4.2 Functional modes

**RQ4060: system modes**

The card shall provide the following modes:

- "INITIALISATION", "OPERATIONAL", "STAND-BY", "OFF" modes
**RQ4065: Modes transition**

**Figure 7: Modes transition diagram**

RQ4070: “OFF” mode
In this mode the whole system shall be turned off.

RQ4075: “INITIALISATION” mode
“INITIALISATION” is a switching mode from “OFF” to “OPERATIONAL” mode. This mode is automatically called after a switch on of the board.

RQ4080: “OPERATIONAL” mode
In this mode the card shall be fully operational.

RQ4085: “STAND-BY” mode
This mode will suspend the USB, so in this mode the FT245R chip and the MCU gone in suspend mode. The card will switch to the “OPERATIONAL” mode after a wake-up signal.

RQ4090: external wakeup signal
The card shall be able to wakeup according to the activity of USB and SPI interfaces. This will put the card in the “OPERATIONAL” mode (refer to RD [4])

### 4.3 Command and housekeeping

**RQ4095: Board switch on**
The switch on of the board shall be realized with a push button.

**RQ4100: Board reset**
The card shall provide a push button in order to reset the board and to allow the initialization of all the functions.
RQ4105: Status information
The card shall provide the following status information by means of LEDs:
“POWER ON”, “TRANSMITTING/RECEIVING DATA”, modes status.

RQ4110: “POWER ON” information
“POWER ON” status information indicates if the card is power supplied or not. A green LED is lighting when the card is powered.

RQ4115: “TRANSMITTING/RECEIVING DATA” information
“TRANSMITTING/RECEIVING DATA” status information indicates when data is being transmitted or received by the device. A green LED is blinking when the card perform data transmission or receiving.

RQ4120: debug support
The card shall provide an interface for debug support.

5 INTERFACE

5.1 Mechanical

RQ5005: Board size
The dimensions of the board shall be.

Maximum Length: 60 mm
Maximum Width: 30 mm
Maximum Height: 20 mm

RQ5010: Board fixing
The card shall be able to be fixed to other board.

5.2 Electrical

5.2.1 POWER

RQ5015: Power plug
The card shall provide a connector which provide the power supply composed of GND and 5[V] signal.

RQ5020: Power pack
The card shall own a battery which shall be able to supply the whole board. The board shall charge the battery using the external power supply.

RQ5025: Power source
The card shall be able to select which power supply to use.

RQ5030: Voltages
The card shall be able to provide 5V and 3.3V voltage.
**RQ5035: 3.3V voltage**  
The USB controller IC shall provide the 3.3V.

**RQ5040: Operational mode**  
In operational mode the maximum current shall be 100mA.

**RQ5045: Standby mode**  
In standby mode the maximum current shall be 500uA.

### 5.2.2 USB

**RQ5050: USB connector**  
A standard USB port shall be able to connect the card to a computer (Type: “A” receptacle)

**RQ5055: USB signals**  
The USB interface shall fulfill the USB1.1 / USB2.0 standards.

**RQ5060: USB performance**  
The USB interface shall fulfill the full-speed transactions.

### 5.2.1 SPI

**RQ5065: SPI connector**  
The card shall provide one standard SIP connector with 4 pins which handle the clock and the data lines for the SPI interface. Separately one SIP connector with 6 pins shall be used to transmit the selected DAC to the CS_Dispach.Board.

- **Type 1**: 4 pins Male Header SIP connector
- **Type 2**: 6 pins Male Header SIP connector

**RQ5070: SPI signals**  
The SPI interface shall fulfill master/slave communication mode with the following connection,

4 pins Male Header SIP connector

- **Pin1**: SCLK, clock signal
- **Pin2**: DIN, MOSI (Master Output Slave Input)
- **Pin3**: DOUT, MISO (Master Input Slave Output)
- **Pin4**: /FS_N, Frame synchronization
6 pins Male Header SIP connector

- Pin1: SCLK, clock signal
- Pin2: DIN, MOSI (Master Output Slave Input)
- Pin3: DOUT, MISO (Master Input Slave Output)
- Pin4: CS, chip select
- Pin5: Feedback, analogue signal

**RQ5075: SPI master device**
The card shall provide the master device who initiates the data packet.

**RQ5080: SPI slave devices**
Multiple slave devices are allowed with individual slave select (chip select).

### 5.2.2 Debug

**RQ5085: Debug interface**
The card shall provide an interface for debug and programming the MCU.

- 1 pin to the MCU
Appendix B - Schematics

USB-SPI Translator
Input

Vdc
Vbat
charge

Output

Vsel

Battery control

Power selector

Battery charger

Vdc
charge

Step-up - 5V

\[ V_{out(Step-up)} = 1.19 \times (1 + \frac{R5}{R6}) \]
INPUT

Decoupling

Vbat divider for ADC

Digital input protection

OUTPUT

SCLK
MOSI
CS
FS_N
ON/OFF
charge
LEDP
LEDB
RD
WR
I/O

RESET
D0
D1
D2
D3
D4
D5
D6
D7

LC filter

Voltage regulator - 2.5V

Voltage regulator - 3.3V

I2C

Decoupling

Vbat divider for ADC

Digital input protection

INPUT

Decoupling

Vbat divider for ADC

Digital input protection

OUTPUT

SCLK
MOSI
CS
FS_N
ON/OFF
charge
LEDP
LEDB
RD
WR
I/O

RESET
D0
D1
D2
D3
D4
D5
D6
D7

LC filter

Voltage regulator - 2.5V

Voltage regulator - 3.3V

I2C
CS Dispatch
Voltage regulator - 3.3V

Inputs
- Vcc
- VCC3.3

Outputs
- Vin
- EN
- GND
- BYP
- Vout
- VCC3.3

Components:
- U1: SPX3819-3.3
- C1: 1uF

Connectors:
- Pin 1: Vin
- Pin 2: EN
- Pin 3: GND
- Pin 4: BYP
- Pin 5: Vout

Title: CS Dispatch Board

Date: Monday, August 02, 2010

Sheet 2 of 4
Appendix C – PCB Layouts

USB-SPI Translator
CS Dispatch
Appendix D – Bill of Materials

USB-SPI Translator
## Bill Of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>C1, C10, C11</td>
<td>4.7uF</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>C2, C07, C8, C9, C12, C14</td>
<td>100nF</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>C3</td>
<td>10uF</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>C4</td>
<td>56nF</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>C5</td>
<td>180nF</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>C6</td>
<td>150pF</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>C7</td>
<td>47uF</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>C13, C16</td>
<td>1uF</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>C15</td>
<td>0.47uF</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>D1, D2</td>
<td>LED</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>D3, D4</td>
<td>3.6V</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>J1, J2</td>
<td>CON2</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>J3, J6</td>
<td>CON6</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>J4</td>
<td>CON8</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>J5</td>
<td>CON4</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>J7</td>
<td>JUMPER</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>J8</td>
<td>CON3</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>L1</td>
<td>22uH</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>L2</td>
<td>10uH</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>R1, R2</td>
<td>330</td>
</tr>
<tr>
<td>21</td>
<td>7</td>
<td>R3, R5, R7, R9, R10, R11, R15</td>
<td>100k</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>R4</td>
<td>560k</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>R6</td>
<td>330k</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>R8</td>
<td>1k</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>R12</td>
<td>4.7k</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>R13, R14</td>
<td>10k</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>R16</td>
<td>200k</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>R17, R18</td>
<td>200</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>SW1</td>
<td>PUSHBUTTON</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>U1, U7</td>
<td>IRF7306</td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>U2</td>
<td>IRF7307</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>U3</td>
<td>NCP1410</td>
</tr>
<tr>
<td>33</td>
<td>2</td>
<td>U4, U5</td>
<td>Diode</td>
</tr>
<tr>
<td>34</td>
<td>1</td>
<td>U6</td>
<td>CD4011B</td>
</tr>
<tr>
<td>35</td>
<td>1</td>
<td>U8</td>
<td>FT245RL</td>
</tr>
<tr>
<td>36</td>
<td>1</td>
<td>U9</td>
<td>ATmega88</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>U10</td>
<td>SPX3819−3.3</td>
</tr>
<tr>
<td>38</td>
<td>1</td>
<td>U11</td>
<td>REF3025</td>
</tr>
</tbody>
</table>
CS Dispatch
<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>C1</td>
<td>1uF</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>C2, C3, C4, C5, C6, C7, C8</td>
<td>0.1uF</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>C9, C10, C11, C12, C13, C14</td>
<td>32pF</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>D1, D2</td>
<td>LED</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>J1, J4</td>
<td>CON6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>J2, J3, J5</td>
<td>CON39</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>R1, R2</td>
<td>330</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>R3</td>
<td>1k</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>SW1</td>
<td>PUSHBUTTON</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>U1</td>
<td>SPX3819–3.3</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>U2</td>
<td>XC95144XL</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>U3, U4, U5, U6, U7, U8</td>
<td>SN74HC4851</td>
</tr>
</tbody>
</table>
Appendix E – Source code

USB-SPI Translator
Global definitions

const int n = 10;

const int REF = 2.048V;

char[MAX_DAC*3] tUSBRD;

char[MAX_DAC*2] tFBData;

char[MAX_DAC*3] tUSBWR;

int countReset = 0;

int FBIndex;

fUSBRD = 0;

fUSBWR = 0;

fSPIWR = 0;

FFB = 0;

FBAT = 0;
Main function

Start main()

Init()

fUSBRD = 1

YES → USB_RD_Data_Process()

NO

fFB = 1

YES → FB_Process()

NO → FB_Data_Process()

fUSBWR = 1

YES → USB_WR_Data_Process()

NO

fBAT = 1

YES → Battery_Process()

NO
USB Read Interrupt Sub Routine

ISR_USB_RD

- PB6(RXF#) pin falling edge interrupt, when RXF#'s low, data available in the FIFO

int count = 0;
tRead = Null;

PCICR = 0x00

- Pin Change Interrupt disable

RXF# low

- Check if there is still data to read

YES

Set RD# low

- Fetch next (if available) data byte from the receive buffer

Delay(20ns<t<50ns)

tUSBRD[i] = PORTB

Set RD# high

- Set RD# back high to be ready for a new read

Count++;

Delay(t>50ns)

fUSBRD = 1

Count > 37*3

YES

NO

Count > 37*3

NO

YES
SPI_ISR

SPI_ISR

fSPIWR = 0;

END
TIMER0_ISR

countReset++;

END
TIMER1 ISR

TIMER1_ISR

ff8 = 1;

END
ADC_ISR

PB2 = 0;

tADCResults[FBIndex] = ADCL;

tADCResults[FBIndex+1] = ADCH;

PB2 = 1;

FBIndex +=2;

fADCComplete = 1;

FBIndex > 72

YES

fFBRD = 0;

NO

FBIndex = 0;

END
Button ISR

1. **Button_ISR**
2. **TCCR0B = clock select**
3. **countReset > 3000**
4. **ON/OFF = 0**
5. **PCIF1 = 0**
6. **WDRF = 1**

- **PC2(button) pin raising edge interrupt**
- **Clock Select Bit, Start Timer0**
- **Shutdown the board**
- **Clear Pin Change Interrupt Flag 1**

System Reset
USB RD Data Process

Method for process the received USB data

char header = NULL;

header = tRead[0] && 0x03;

Read the first received byte with a mask to read only the first two bit (CMD)

Header = 0x10

YES

Data process

NO

Header = 0x01

YES

header = tRead[0] && 0xFC;

Read the SPI selection bits

Header = 0x00

NO

YES

Single_Voltage_Process()

NO

Header = 0x0C

YES

Same_Voltage_Process()

NO

Header = 0xCD

YES

Angle_Process()

YES

fUSBRD = 0;

PCICR = 0x01

Pin Change Interrupt Enable

END
USB WR Data Process

1. `int i = 0;`

2. `DDRD = 0xFF;` Set PORTD as output

3. `TXE# = 0;`
   - NO
   - YES
     - `WR = 1;`

4. `PORTD = uUSBWR[i];` Set the byte to transmit on PORTD

5. `WR = 0;` Write data byte on PORTD into transmit FIFO buffer (FT245R)

6. `i++;`

7. `i < MAX_DAC` if YES, go to step 1; if NO, go to step 8

8. `fUSBWR = 0;`

9. `DDRD = 0x00;` Set PORTD as input

END
Single Voltage Process

```c
char index = 1;
int i = 1;
float data = -1;

data = (float)tRead[i] - 48;
data += ((float)(tRead[i+2] - 48))/10;

if (data > -1) {
    index = (i/3)-1;
    SPI_DAC_WR(index, data);
}

i = i + 3;

if (index > MAX_DAC) {
    END
}
```

Check which DAC shall be updated and call the SPI_Send function

Input:
- tRead[]
- MAX_DAC

Output:
- SPI_DAC_WR

Function:
- SPI_DAC_WR(index, data)
Same Voltage Process

```
float data = -1;
char index = 1;

data = (float)tRead[1] - 48;

SPI_DAC_WR(index, data);

index++;

if (index > MAX_DAC)
    END
```
```c
float data = -1;
float tData = -1;
char index = 1;

tData = (float)tRead[1] - 48;
tData += ((float)(tRead[3] - 48))/10;

data = angle2VoltageFormule(index, tData);
SPI_DAC_WR(index, data);
index++;

if (index > MAX_DAC) END
```
SPI_DAC_WR

SPI_DAC_WR

int newDACValue = -1;

SPI_CPLD_WR(index);

newDACValue = New_DAC_Value(data);

FS_N = 0;

SPDR = (char)(newDACValue >> 8);

F_SPIWRComplete = 0

YES

SPDR = (char)(newDACValue);

F_SPIWRComplete = 0

YES

FS_N = 1;

END
SPI CPLD WR

SPI_CPLD_WR

PB2 = 0;

SPDR = index;

// Possibility to read

NO

SPIWRComplete = 0

YES

PB2 = 1;

END
New DAC Value

```
return (data*(2^n))/(2*REF);
```

END
FB Process

```
FB_Process

TCCR1B &= (0<<CS10) | (0<<CS11) | (0<<CS12);

Data = 0xFF;

SPI_CPLD_WR(data);

if SPIWR

if ISPIWR = 0

if fADCComplete = 0;

if ADCSRA |= (1<<ADSC);

if fADCComplete = 1

if FFBRD = 1

if fADCComplete = 1

Stop TIMER1

Start feedback process on the CPLD

Start AD conversion

END
```
FB Data Process

Float data = -1;
int tData = -1;
int i = 1;

tData = (int)(tFB[i]<<8);
tData += (byte)(tFB[i+1]);

Data = (tData*2.048)/1024;

tUSBWR[i] = ((int)(data)) + 48;
tUSBWR[i+1] = ' ';
data = (int)((data*10)-(((int)(data))*10));
tUSBWR[i+2] = ((char)data) + 48;

tUSBWR[0] = 0x40;

fUSBWR = 1;

fFB = 0;

TCCR1B &= (0<<CS10) | (1<<CS11) | (1<<CS12);

END
float batCode = -1;
float batVol = -1;

PORTC |= (1<<PC3);
Shut the MOSFET, stop charging battery during test

fADC = 1;

ADCSRA |= (1<<ADSC);
Start AD conversion

while(fADC == 1);

batCode = (unsigned char)ADCL;
batCode += (int)(ADCH<<8);
Read ADC result

batVol = (((batCode*REF)/N)*RBAT2)/RBAT1;
Calculate the analogue voltage

batVol < BAT_THRESHOLD_LOW
NO

PORTC |= (1<<PC3);
PORTC &= 0xF7;
YES

batVol > BAT_THRESHOLD_HIGH
NO

chargeBat == 1
NO

PORTC &= 0xFE;
YES

fBAT = 0;
END
FILENAME : USBSPITranslator.c
AUTHOR : Salamin Yannick <yannick.salamin@gmail.com>

FUNCTION : Realizing the bidirectional translation between USB FIFO
            interface and SPI protocol. DACs feedback processing and
            battery management

REVISION : 1.0 (version for ATmega88)

#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>

/* Global definitions & declarations */

/* Constants */
#define N 1024
#define REF 2.5
#define MAX_DAC 37
#define RBAT1 200
#define RBAT2 300
#define BAT_THRESHOLD_LOW 2.2
#define BAT_THRESHOLD_HIGH 3.6

/* Arrays */
char tUSBRD[37*3];
char tUSBWR[37*3];
char tFBData[37*2];
float tDACWR[37];

/* Variables */
int countReset = 0;
int FBIndex = 0;
int countBAT = 0;
int countFB = 0;
int chargeBat = 0;

/* Flags */
int fUSBRD = 0;
int fUSBWR = 0;
int fSPIWR = 0;
int fFB = 0;
int fFBRD = 0;
int fADC = 0;
int fBAT = 0;

/* Functions */
void USB_RD_Data_Process();
void FB_Process();
void FB_Data_Process();
void USB_WR_Data_Process();
void Single_Voltage_Process();
void Same_Voltage_Process();
void Angle_Process();
void SPI_DAC_WR(char index, int data);
void SPI_CPLD_WR(char index);
int New_DAC_Value(float data);
void Battery_Process();

/**************************************************************************/
/* FUNCTION     : init() */
/* INPUT        : - */
/* OUTPUT       : - */
/* COMMENTS     : Peripherals initialization */
/**************************************************************************/
void init(void)
{
	/*---------------------------------------------------------*/
	/* Status register initialisation */
	/*---------------------------------------------------------*/
	SREG = 0x00; // Reset status register

	/*---------------------------------------------------------*/
	/* Power management initialisation : Power down */
	/*---------------------------------------------------------*/
	SMCR = 0x04; // Select "Power down" mode without enabling to enter in sleep mode

	/*---------------------------------------------------------*/
	/* I/O initialisation */
	/*---------------------------------------------------------*/
	/* PORTB, SPI */
	DDRB = 0x2F; // Set PB0(RD),PB1(WR),PB2(CS), PB3(MOSI) and PB5(SCLK) as output
	// Set PB4(MISO_P),PB6(RXF) and PB7(TXE) as input

	//DDRＢ = (1<<PB2);

	PORTB = (1<<PB0) | (1<<PB2); // Set initial pins value: RD and CS high
	//PORTB = 0x05;

	/* PORTC*/
	DDRC = 0xFB; // Set PC0(LEDB),PC1(LEDP),PC3(charge), PC4(FS_N) and PC5(ON/OFF) as output
	// Set PC2(button) as input

	// Set initial pins value: LEDP,FS_N,ON/OFF high

	PORTC = (1<<PC1) | (1<<PC3) | (1<<PC4) | (1<<PC5);

	/* PORTD, USB FIFO */
	DDRD = 0x00; // Set PORTD as input

	/*---------------------------------------------------------*/
	/* Timer0 initialisation: 1ms */
	/*---------------------------------------------------------*/
	TCCR0A = 0x02; // Set CTC operation mode
	TCCR0B = 0xC0; // Prescaler: 8, TIMER0 disable
	OCR0A = 0x7D; // A output compare value: 250
	TIMSK0 = 0x02;
	TIFR0 = 0x02; // Enable A output match interrupt
/*-----------------------------------------------*/
/* Timer1 initialisation: 1s */
/*-----------------------------------------------*/
TCCR1A = 0x00; // Set CTC operation mode
TCCR1B = 0x0B; // Set prescaler 64
TCCR1C = 0xC0; // A output compare value: 15625
OCR1AH = 0x3D; // A output compare value: 15625
OCR1AL = 0x09; // A output compare value: 15625
TIMSK1 = 0x02; // Enable A output match interrupt

/*-----------------------------------------------*/
/* Pin Change Interrupt init: PCINT6(RXF#), PCINT10(Button) */
/*-----------------------------------------------*/
PCICR = 0x03; // Enable PCIE0 and PCIE1 interrupt
PCMSK0 = 0x40; // Enable PCINT6 interrupt
PCMSK1 = 0x04; // Enable PCINT10 interrupt

/*-----------------------------------------------*/
/* SPI initialisation: Master */
/*-----------------------------------------------*/
SPCR = 0xD8; // Set SPI enable and as master
SPSR |= (1<<SPI2X); // Enable double SPI speed, clk/2

/*-----------------------------------------------*/
/* ADC initialisation: */
/*-----------------------------------------------*/
ADMUX = 0x07; // Set AVref(2.5V) and select ADC6(feedback)
ADCSRA = 0x8C; // SetADC prescaler: 16 and enable interrupt

/*-----------------------------------------------*/
/ * FUNCTION : main() */
/ * INPUT : - */
/ * OUTPUT : - */
/ * COMMENTS : Main function */
/ *-----------------------------------------------*/

int main(void)
{
init(); // Initialisation function call
sei(); // Enable global interrupt

while(1)
{
  if(fUSBRD == 1) // Check if usb received data wait to be process
  {
    USB_RD_Data_Process(); // Call usb received data process function
  }

  if(fFB == 1) // Check if feedback should be updated
  {
    FB_Process(); // Call feedback update process function
    FB_Data_Process(); // Call feedback data process function
    fFB = 0;
  }
}
if(fUSBWR == 1) // Check if usb data need to be send
{
    USB_WR_Data_Process(); // Call usb write data function
}

if(fBAT == 1) // Check if battery need to be tested
{
    Battery_Process(); // Call battery test funtion
}

return 0;

/********************************************************************************
/* FUNCTION     : USB_RD_Data_Process() */
/* INPUT        : - */
/* OUTPUT       : -     */
/* COMMENTS     : Process the received usb data */
/********************************************************************************
void USB_RD_Data_Process()
{
    char header;
    int i = 1;

    header = tUSBRD[0];

    if((header&0x03) == 0x02)
    {
        switch(header&0x0FC)
        {
            case 0x00:
                Single_Voltage_Process(); // Call single voltage update function
                break;

            case 0x0C:
                Same_Voltage_Process(); // Call all the same voltage update function
                break;

            case 0xC0:
                Angle_Process(); // Call angle process funtion
                break;

            default:
                break;
        }
    }
    else if(header == 0x01)
    {
    }
    else
    {
    }

for(i=0;i<MAX_DAC*3;i++)
{
    tUSBRD[i] = 0x00; // Reset usb read buffer
}
```c
fUSBRD = 0;  // Clear USB read flag
PCICR = 0x03; // Enable pin change interrupt on PB6
}

/***************************************************************************/
/* FUNCTION     : USB_WR_Data_Process() */
/* INPUT        : - */
/* OUTPUT       : - */
/* COMMENTS     : Write data into the USB controller */
/***************************************************************************/
void USB_WR_Data_Process()
{
  int i = 0;
  DDRD = 0xFF; // Set PORTD as output

  for(i=0; i<37; i++)
  {
    if(((PINB&0x80) == 0x00)) // Check that USB controller is able to receive data
    {
      PORTB |= (1<<PB1); // Pull up WR
      PORTD = tUSBWR[i]; // Prepare PORTD with data to write
      PORTB = (0<<PB1); // Pull down WR, write data in USB controller
      fUSBWR = 0; // Reset USB write flag
    }
    else
    {
      fUSBWR = 1;
    }
  }
  TCCR1B = 0x0B; // Start Timer 1
  DDRD = 0x00; // Set PORTD as input
}

/***************************************************************************/
/* FUNCTION     : Single_Voltage_Process() */
/* INPUT        : - */
/* OUTPUT       : - */
/* COMMENTS     : Update single DAC voltage */
/***************************************************************************/
void Single_Voltage_Process()
{
  char index = 1;
  int tDigitalDAC[37];

  do
  {
    if(tDACWR[index-1]>-1) // Check if data to update
    {
      tDigitalDAC[index-1] = New_DAC_Value(tDACWR[index-1]); // Calculate new digital value to update
    }
    else
    {
    }
```
tDigitalDAC[index-1] = -1;
}

index++;
// Update index from increment value to determine which DAC to update
while(index<(MAX_DAC+1)); // Check all 37 DAC

index = 1;
do {
    if(tDigitalDAC[index-1]>-1) // Check if data to update
    {
        SPI_DAC_WR(index,tDigitalDAC[index-1]); // Call DAC write function with the index and value to update
    }
    index++;
    // Update index from increment value to determine which DAC to update
}while(index<(MAX_DAC+1)); // Check all 37 DAC

SPI_CPLD_WR(0x00); // Reset the chips selection

/********************************************************************************/
/* FUNCTION     : Same_Voltage_Process() */
/* INPUT        : - */
/* OUTPUT       : - */
/* COMMENTS     : Update DAC with the same voltage */
*********************************************************************************/
void Same_Voltage_Process()
{
    char index = 50; // Index value to update all the DACs
    SPI_DAC_WR(index,New_DAC_Value(tDACWR[0])); // Call DAC write function with the index and the digital value to update
    SPI_CPLD_WR(0x00); // Reset the chips selection
}

/********************************************************************************/
/* FUNCTION     : Angle_Process() */
/* INPUT        : - */
/* OUTPUT       : - */
/* COMMENTS     : Update DAC with voltage corresponding to the angle */
*********************************************************************************/
void Angle_Process()
{
    char index = 0;
    int tDigitalDAC[37];

do {
    tDigitalDAC[index] = New_DAC_Value(tDACWR[index]); // Calculate new digital value to update
    index++;
    // Update index from increment value to

determine which DAC to update

}while(index<MAX_DAC);  // Check all 37 DAC

index = 2;

do
{
  SPI_DAC_WR(index,tDigitalDAC[index-1]);
  index++;
}while(index<MAX_DAC+1);  // Update all 37 DAC

SPI_CPLD_WR(0x00);  // Reset the chips selection
}

/********************************************************************************
/* FUNCTION     : SPI_DAC_WR() */
/* INPUT        : char, float */
/* OUTPUT       : - */
/* COMMENTS     : Write digital voltage into SPI channel */
********************************************************************************/

void SPI_DAC_WR(char index,int data)
{
  PORTB |= (1<<PB2);

  SPI_CPLD_WR(index);  // Call cpld write function to select the DAC

  PORTC = 0xEA;  // Pull down FS_N, indicate new value

  SPCR |= (1<<SPIE);  // Enable SPI interrupt
  fSPIWR = 1;  // Set SPI writing flag
  SPDR = (char)(data>>8);  // Write new DAC value
  while(fSPIWR == 0);  // Wait for SPI transfert complete

  SPCR |= (1<<SPIE);  // Enable SPI interrupt
  fSPIWR = 1;  // Set SPI writing flag
  SPDR = (char)(data);  // Write new DAC value
  while(fSPIWR == 0);  // Wait for SPI transfert complete

  PORTC |= (1<<PC4);  // Pull up FS_N

}

/********************************************************************************
/* FUNCTION     : SPI_CPLD_WR(char index) */
/* INPUT        : char */
/* OUTPUT       : - */
/* COMMENTS     : Send index, DAC to select, to the CPLD through SPI channel */
********************************************************************************/

void SPI_CPLD_WR(char index)
{
  PORTB = (0<<PB2);  // Pull down CS, select CPLD to write SPI

  SPCR |= (1<<SPIE);  // Enable SPI interrupt
  fSPIWR = 1;  // Set SPI writing flag
  SPDR = index;  // Write index, DAC to select, to the CPLD
while(fSPIWR == 0); // Wait for SPI transfert complete

PORTB |= (1<<PB2); // Pull up CS, CPLD transfert finished

/********************************************************************************
* FUNCTION : FB_Process()                                               */
* INPUT     : -                                                       */
* OUTPUT    : -                                                       */
* COMMENTS  : Update the feedback values                              */
********************************************************************************/

void FB_Process()
{
    char index;
    FBIndex = 0;
    fFBRD = 1;

    TCCR1B &= (0<<CS10) | (0<<CS11) | (0<<CS12); // Stop Timer1
    ADMUX = 0x06; // Select ADC6(feedback)

    index = 0x64; // Set feedback update index
    SPI_CPLD_WR(index); // Inform CPLD of feedback process

    do
    {
        fADC = 1; // Clear ADC complete flag
        ADCSRA |= (1<<ADSC); // Start AD conversion
        while(fADC == 1); // Wait for AD conversion complete

        tFBData[FBIndex] = ADCL; // Store AD conversion results
        tFBData[FBIndex+1] = ADCH;

        PORTB = (0<<PB2); // Pull down CS, CPLD increment mux
        address
        PORTB |= (1<<PB2); // Pull up CS

        FBIndex += 2; // Increment index

        if(FBIndex > MAX_DAC*2) // Check if all DAC were read
        {
            fFBRD = 0; // Clear feedback flag
            FBIndex = 0; // Reset feedback index
        }
    }while(fFBRD == 1); // Read all 37 DAC

    ADMUX = 0x07; // Select ADC7(Battery)
}

/********************************************************************************
* FUNCTION : FB_Data_Process()                                          */
* INPUT     : -                                                       */
* OUTPUT    : -                                                       */
* COMMENTS  : process the FB data in order to send them to the computer  */
********************************************************************************/

void FB_Data_Process()
{
    float data = -1;
```c
long tData = -1;
int i = 1;
int index = 1;

do
{
    tData = 0;
data = 0;

    tData = (tFBData[index] << 8); // Read feedback data
tData += (tFBData[index - 1]);

    data = (tData * REF) / (N); // Calculate the analogue equivalent value
    // data = data * 5;
    // data = data - 3.5;

    tUSBWR[i] = ((int)(data) + 48); // Write the first digit in the write buffer
    tUSBWR[i + 1] = '.'; // Write the dot in the write buffer

    data = ((int)(data * 10) - ((int)(data) * 10)); // Format second digit

    tUSBWR[i + 2] = ((char)data) + 48; // Write second digit in the write buffer

    i += 3;
    index += 2;
}
while (i < MAX_DAC * 3 + 1);

fUSBWR = 1; // Set usb write flag
fFB = 0; // Clear feedback flag

}/**/  

int New_DAC_Value(float data)
{
    int temp = (data * (N)) / (2 * REF); // Calculate new digital value for DAC
    return temp; // return new digital value for DAC
}

void Battery_Process()
{/ */} */
int batCode = 0;
float batVol = 0;

PORTC |= (1<<PC3); // Shut the MOSFET, stop charging battery
during test

ADMUX = 0x07; // Select ADC7(Battery)

fADC = 1; // Clear ADC complete flag
ADCSRA |= (1<<ADSC); // Start AD conversion

while(ADC == 1); // Wait for AD conversion complete

batCode = (unsigned char)ADCL; // Read AD conversion results
batCode += (int)(ADCH<<8);

batVol = (((batCode*REF)/N)*RBAT2)/RBAT1; // Calculate the real analogue voltage

if(batVol < BAT_THRESHOLD_LOW) // Check if battery is full
{
    PORTC |= (1<<PC0); // Turn on the Battery LED
    PORTC &= 0xF7; // Start charging battery
    chargeBat = 1;
}
else if(batVol > BAT_THRESHOLD_HIGH) // Check if battery need to be charged
{
    PORTC &= 0xFE; // Turn off the Battery LED
    PORTC |= (1<<PC3); // Shut the MOSFET, stop charging battery
    chargeBat = 0;
}
else
{
    if(chargeBat == 1)
    {
        PORTC &= 0xF7; // Charge battery
    }
}

fBAT = 0; // Clear battery check flag
/* include <avr/io.h> */
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>

#define MAX_DAC 37
#define SHUTDOWN_TIME 3000
#define RESET_TIME 1000
#define BAT_TIME 100

extern int countReset;
extern int countFB;
extern int countBAT;
extern int fUSBRD;
extern int fSPIWR;
extern int fFB;
extern int fFBRD;
extern int fADC;
extern int fBAT;
extern char tUSBRD[37*3];
extern float tDACWR[37];

/* FUNCTION     : TIMER0_COMPA_vect */
/* INPUT        : - */
/* OUTPUT       : - */
/* COMMENTS     : ISR for Timer 0 */
/*==================================*/
SIGNAL(TIMER0_COMPA_vect)
{
    countReset++;
    // Increment board reset counter
}

/* FUNCTION     : TIMER1_COMPA_vect */
/* INPUT        : - */
/* OUTPUT       : - */
/* COMMENTS     : ISR for Timer 1 */
/*==================================*/
SIGNAL(TIMER1_COMPA_vect)
{
    if(countBAT > BAT_TIME)
    {
        countBAT = 0;
        fBAT = 1;
    }
    else
    {
        countBAT++;
    }
if(countFB > 5)
{
    fFB = 1;                       // Set feedback update flag
    countFB = 0;
}
else
{
    countFB++;
}

/******************************************************************************
/* FUNCTION     : PCINT0_vect                                     */
/* INPUT        : -                                              */
/* OUTPUT       : -                                               */
/* COMMENTS     : ISR for PB6(RXF#) level change                    */
/******************************************************************************
SIGNAL(PCINT0_vect)
{
    int i = 0;
    int index = 0;
    float data = -1;
    PCICR = 0x02;                                                      // Disable pin change interrupt for PB6

    do
    {
        if(((PINB&0x40) == 0x00))                                      // Check if PB6(RXF#) is low
        {
            PORTB = (1<<PB0);                                          // Pull down PB0(RD) in order to fetch next
            (if available) data byte from the receive buffer
            tUSBRD[i] = FIND;                                           // Read PORTB
            PORTB |= (1<<PB0);                                          // Pull up PB0(RD)
            i++;                                                      // Increment buffer index
        }
        else
        {
            break;                                                   // Break out the loop if PB6(RXF#) is high
        }
    }while(i<(MAX_DAC*3)+1);                                            // Read all 37 new DAC value

    i = 1;                                                              // Reset index for next buffer

    do
    {
        data = -1;

        data = (float)tUSBRD[i]-48;                                   // Read digit before the virgule
        data += ((float)(tUSBRD[i+2]-48))/10;                          // Read digit after the virgule

        if(data != 9.9)
        {
            data = data + 3.6;                                         // Add DACs voltage offset
            data = data/5;                                              // Divide the DACs amplification
            tDACWR[index] = data;                                       // Store the data
} else {
    tDACWR[index] = -1; // No charging value
}

i += 3; // Increment index to read next received value
index++; // Increment index for next value
}

while (index < MAX_DAC);

if (tUSBRD[0] != 0xFF) // Check if read buffer is empty
{
    fUSBRD = 1; // If read buffer is not empty, set USBRD flag
}

/*****************************************************************************/
/* FUNCTION     : PCINT1_vect
/* INPUT        : - */
/* OUTPUT       : - */
/* COMMENTS     : ISR for PC2(Button) level change */
/*****************************************************************************/
SIGNAL(PCINT1_vect)
{
    sei(); // Enable global interrupt

    if ((PINC & 0x04) == 0x04)
    {
        TCCR0B = 0xC2; // Start TIMER0
    }

    if (countReset > SHUTDOWN_TIME)
    {
        PORTC = (0 << PC5); // Shut down the board
    }
    else
    {
        if ((PINC & 0x04) == 0x00)
        {
            TCCR0B = 0xC0; // Stop TIMER0
            TCNT0 = 0x00; // Reset TIMER0
        }
    }
}

/*****************************************************************************/
/* FUNCTION     : ADC_vect
/* INPUT        : - */
/* OUTPUT       : - */
/* COMMENTS     : ISR for ADC conversion completed */
/*****************************************************************************/
SIGNAL(SPI_STC_vect)
{
    fSPIWR = 0; // Clear SPI writing flag
}

/*****************************************************************************/
SIGNAL(ADC_vect)
{
    fADC = 0; // Clear AD conversion flag
}
CS Dispatch
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.NUMERIC_STD.all;

ENTITY TOP IS
  PORT(
    CLK : IN  std_logic;
    CS  : IN  std_logic;
    DIN : IN  std_logic;
    CS_N : OUT unsigned (36 DOWNTO 0);
    DOUT : OUT std_logic;
    FB_Addr : OUT unsigned (5 DOWNTO 0)
  );

END TOP;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.NUMERIC_STD.all;

LIBRARY CS_Dispatch_lib;

ARCHITECTURE struct OF TOP IS

  -- Architecture declarations

  -- Internal signal declarations
  SIGNAL FB : std_logic := '0';
  SIGNAL dbus0 : unsigned(5 DOWNTO 0);

  -- Component Declarations

  COMPONENT FB_selection
    PORT ( 
      CS    : IN  std_logic;
      FB    : IN  std_logic;
      FB_Addr : OUT unsigned (5 DOWNTO 0)
    );
  END COMPONENT;

  COMPONENT SPI
PORT (
    CLK : IN  std_logic;
    CS  : IN  std_logic;
    DIN : IN  std_logic;
    DOUT: OUT std_logic;
    FB  : OUT std_logic;
    dbus0: OUT  unsigned (5 DOWNTO 0)
);
END COMPONENT;

COMPONENT chips_selection
PORT (
    dbus1 : IN  unsigned (5 DOWNTO 0);
    CS_N  : OUT unsigned (36 DOWNTO 0)
);
END COMPONENT;

-- Optional embedded configurations
-- pragma synthesis_off
FOR ALL : FB_selection USE ENTITY CS_Dispatch_lib.FB_selection;
FOR ALL : SPI USE ENTITY CS_Dispatch_lib.SPI;
FOR ALL : chips_selection USE ENTITY CS_Dispatch_lib.chips_selection;
-- pragma synthesis_on

BEGIN

-- Instance port mappings.
U_2 : FB_selection
    PORT MAP (
        CS  => CS,
        FB  => FB,
        FB_Addr => FB_Addr
    );
U_0 : SPI
    PORT MAP (
        CLK  => CLK,
        CS   => CS,
        DIN  => DIN,
        DOUT => DOUT,
        FB   => FB,
        dbus0 => dbus0
    );
U_1 : chips_selection
    PORT MAP (
        dbus1 => dbus0,
        CS_N  => CS_N
    );

END struct;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.NUMERIC_STD.all;

ENTITY SPI IS
PORT(
    CLK : IN std_logic;
    CS : IN std_logic;
    DIN : IN std_logic;
    DOUT : OUT std_logic;
    FB : OUT std_logic;
    dbus0 : OUT unsigned (5 DOWNTO 0)
);

-- Declarations
END SPI ;

ARCHITECTURE struct OF SPI IS

    signal counter_int : unsigned(7 DOWNTO 0);
    signal count : natural:=0;

BEGIN

    process(CLK)
    begin

        if CS = '0' then
            if count < 8 then
                if falling_edge(CLK) then
                    counter_int(?-count) <= DIN;
                    count <= count + 1;
                end if;
            end if;
        else
            if count < 8 then
                if counter_int = 100 then
                    FB <= '1',
                    '0' AFTER 4 us;
                    count <= 0;
                else
                    dbus0 <= RESIZE(counter_int,dbus0'length);
                    count <= 0;
                end if;
            end if;
        end if;
    end process;

END struct;
end if;
  count <= 0;                        -- Reset counter
end if;
end process;

END ARCHITECTURE struct;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.NUMERIC_STD.all;

ENTITY chips_selection IS
    PORT(
        dbus1 : IN unsigned (5 DOWNTO 0);
        CS_N : OUT unsigned (36 DOWNTO 0)
    );

    -- Declarations

END chips_selection ;

ARCHITECTURE struct OF chips_selection IS
BEGIN

    process(dbus1)
    begin
        -- chips selection, for the value 0 no DAC are selected,
        -- for the values between 1 and 37 the corresponding DAC
        -- is selected and for the value 50, every DAC is selected
        case TO_INTEGER(dbus1) is
            when 0 => CS_N <= (others => '1');
            when 1 => CS_N <= (0 => '0', others => '1');
            when 2 => CS_N <= (1 => '0', others => '1');
            when 3 => CS_N <= (2 => '0', others => '1');
            when 4 => CS_N <= (3 => '0', others => '1');
            when 5 => CS_N <= (4 => '0', others => '1');
            when 6 => CS_N <= (5 => '0', others => '1');
            when 7 => CS_N <= (6 => '0', others => '1');
            when 8 => CS_N <= (7 => '0', others => '1');
            when 9 => CS_N <= (8 => '0', others => '1');
            when 10 => CS_N <= (9 => '0', others => '1');
            when 11 => CS_N <= (10 => '0', others => '1');
            when 12 => CS_N <= (11 => '0', others => '1');
            when 13 => CS_N <= (12 => '0', others => '1');
            when 14 => CS_N <= (13 => '0', others => '1');
            when 15 => CS_N <= (14 => '0', others => '1');
            when 16 => CS_N <= (15 => '0', others => '1');
            when 17 => CS_N <= (16 => '0', others => '1');
            when 18 => CS_N <= (17 => '0', others => '1');
            when 19 => CS_N <= (18 => '0', others => '1');
            when 20 => CS_N <= (19 => '0', others => '1');
            when 21 => CS_N <= (20 => '0', others => '1');
            when 22 => CS_N <= (21 => '0', others => '1');
            when 23 => CS_N <= (22 => '0', others => '1');
        end case;
    end process;

-- VHDL Architecture CS_Dispatch_lib.chips_selection.struct
-- Created:
--          by - Yannick Salamin.UNKNOWN (AEM-1D2421F5C0D)
--          at - 19:56:34 09/25/2010
--
-- using Mentor Graphics HDL Designer(TM) 2005.3 (Build 74)
--
when 24 => CS_N <= (23 => '0', others => '1');
when 25 => CS_N <= (24 => '0', others => '1');
when 26 => CS_N <= (25 => '0', others => '1');
when 27 => CS_N <= (26 => '0', others => '1');
when 28 => CS_N <= (27 => '0', others => '1');
when 29 => CS_N <= (28 => '0', others => '1');
when 30 => CS_N <= (29 => '0', others => '1');
when 31 => CS_N <= (30 => '0', others => '1');
when 32 => CS_N <= (31 => '0', others => '1');
when 33 => CS_N <= (32 => '0', others => '1');
when 34 => CS_N <= (33 => '0', others => '1');
when 35 => CS_N <= (34 => '0', others => '1');
when 36 => CS_N <= (35 => '0', others => '1');
when 37 => CS_N <= (36 => '0', others => '1');
when others => CS_N <= (others => '1');
end case;
end process;

END ARCHITECTURE struct;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.NUMERIC_STD.all;

ENTITY FB_selection IS
PORT(
    CS : IN  std_logic;
    FB : IN  std_logic;
    FB_Addr : OUT unsigned (5 DOWNTO 0)
);

END FB_selection;

ARCHITECTURE struct OF FB_selection IS

signal address_int : unsigned(5 DOWNTO 0);
signal flag : std_ulogic;
signal count_int : unsigned(5 DOWNTO 0);

BEGIN

process(FB,CS)
begin
    if FB = '1' then
        FB_Addr <= (others => '0'); -- Reset counter
        count_int <= (others => '0'); -- Reset feedback channel
        flag <= '1'; -- Set flag for FB process on
    elsif falling_edge(CS) then
        if flag = '1' then
            if count_int < 37 then
                FB_Addr <= count_int; -- Update feedback channel
                count_int <= count_int + 1; -- Increment counter
            else
                count_int <= (others => '0'); -- Reset counter
                FB_Addr <= (others => '0'); -- Reset feedback channel
                flag <= '0'; -- Reset FB processing flag
            end if;
        end if;
    end if;
end process;

END ARCHITECTURE struct;
GUI
#include "stdafx.h"
#include "Form1.h"
#include <windows.h>
#include "USB.h"

using namespace USBFT245R;

int APIENTRY _tWinMain(HINSTANCE hInstance,
            HINSTANCE hPrevInstance,
            LPTSTR lpCmdLine,
            int nCmdShow)
{
    System::Threading::Thread::CurrentThread->ApartmentState = System::Threading::
    ApartmentState::STA;
    Application::Run(new Form1());
    return 0;
}

Form1::Form1()
{
    InitializeComponent(); // Call the component initialisation method
}

void Form1::Dispose(Boolean disposing)
{
    StopThread(); // Close the thread
    if (disposing && components)
    {
        components->Dispose();
    }
    __super::Dispose(disposing);
}

void Form1::InitializeComponent()
this->btnClose = (new System::Windows::Forms::Button());
this->btnSend = (new System::Windows::Forms::Button());
this->ListBox1 = (new System::Windows::Forms::ListBox());
this->comboBox1 = (new System::Windows::Forms::ComboBox());
this->radioNumber = (new System::Windows::Forms::RadioButton());
this->radioDescription = (new System::Windows::Forms::RadioButton());
this->radioSerial = (new System::Windows::Forms::RadioButton());
this->groupBox1 = (new System::Windows::Forms::GroupBox());
this->textBoxAngle = (new System::Windows::Forms::TextBox());
this->label1 = (new System::Windows::Forms::Label());
this->label2 = (new System::Windows::Forms::Label());
this->groupBox2 = (new System::Windows::Forms::GroupBox());
this->radioDiffV = (new System::Windows::Forms::RadioButton());
this->radioSameV = (new System::Windows::Forms::RadioButton());
this->radioDirect = (new System::Windows::Forms::RadioButton());
this->label3 = (new System::Windows::Forms::Label());
this->label4 = (new System::Windows::Forms::Label());
this->textBoxSameV = (new System::Windows::Forms::TextBox());
this->label5 = (new System::Windows::Forms::Label());
this->textBox3 = (new System::Windows::Forms::TextBox());
this->label6 = (new System::Windows::Forms::Label());
this->textBox4 = (new System::Windows::Forms::TextBox());
this->label7 = (new System::Windows::Forms::Label());
this->textBox5 = (new System::Windows::Forms::TextBox());
this->label8 = (new System::Windows::Forms::Label());
this->textBox6 = (new System::Windows::Forms::TextBox());
this->label9 = (new System::Windows::Forms::Label());
this->textBox7 = (new System::Windows::Forms::TextBox());
this->label10 = (new System::Windows::Forms::Label());
this->textBox8 = (new System::Windows::Forms::TextBox());
this->label11 = (new System::Windows::Forms::Label());
this->textBox9 = (new System::Windows::Forms::TextBox());
this->label12 = (new System::Windows::Forms::Label());
this->textBox10 = (new System::Windows::Forms::TextBox());
this->label13 = (new System::Windows::Forms::Label());
this->textBox11 = (new System::Windows::Forms::TextBox());
this->label14 = (new System::Windows::Forms::Label());
this->textBox12 = (new System::Windows::Forms::TextBox());
this->label15 = (new System::Windows::Forms::Label());
this->textBox13 = (new System::Windows::Forms::TextBox());
this->label16 = (new System::Windows::Forms::Label());
this->textBox14 = (new System::Windows::Forms::TextBox());
this->label17 = (new System::Windows::Forms::Label());
this->textBox15 = (new System::Windows::Forms::TextBox());
this->label18 = (new System::Windows::Forms::Label());
this->textBox16 = (new System::Windows::Forms::TextBox());
this->label19 = (new System::Windows::Forms::Label());
this->textBox17 = (new System::Windows::Forms::TextBox());
this->label20 = (new System::Windows::Forms::Label());
this->textBox18 = (new System::Windows::Forms::TextBox());
this->label21 = (new System::Windows::Forms::Label());
this->textBox19 = (new System::Windows::Forms::TextBox());
this->label22 = (new System::Windows::Forms::Label());
this->textBox20 = (new System::Windows::Forms::TextBox());
this->label23 = (new System::Windows::Forms::Label());
this->textBox21 = (new System::Windows::Forms::TextBox());
this->label24 = (new System::Windows::Forms::Label());
this->textBox22 = (new System::Windows::Forms::TextBox());
this->label28 = (new System::Windows::Forms::Label());
this.textBox26 = (new System.Windows.Forms.TextBox());
this.label29 = (new System.Windows.Forms.Label());
this.textBox27 = (new System.Windows.Forms.TextBox());
this.label30 = (new System.Windows.Forms.Label());
this.textBox28 = (new System.Windows.Forms.TextBox());
this.label31 = (new System.Windows.Forms.Label());
this.textBox29 = (new System.Windows.Forms.TextBox());
this.label32 = (new System.Windows.Forms.Label());
this.textBox30 = (new System.Windows.Forms.TextBox());
this.label33 = (new System.Windows.Forms.Label());
this.textBox31 = (new System.Windows.Forms.TextBox());
this.label34 = (new System.Windows.Forms.Label());
this.textBox32 = (new System.Windows.Forms.TextBox());
this.label35 = (new System.Windows.Forms.Label());
this.textBox33 = (new System.Windows.Forms.TextBox());
this.label36 = (new System.Windows.Forms.Label());
this.textBox34 = (new System.Windows.Forms.TextBox());
this.label37 = (new System.Windows.Forms.Label());
this.textBox35 = (new System.Windows.Forms.TextBox());
this.label38 = (new System.Windows.Forms.Label());
this.textBox36 = (new System.Windows.Forms.TextBox());
this.label39 = (new System.Windows.Forms.Label());
this.textBox37 = (new System.Windows.Forms.TextBox());
this.label40 = (new System.Windows.Forms.Label());
this.textBox38 = (new System.Windows.Forms.TextBox());
this.label41 = (new System.Windows.Forms.Label());
this.textBox39 = (new System.Windows.Forms.TextBox());
this.label42 = (new System.Windows.Forms.Label());
this.textBox40 = (new System.Windows.Forms.TextBox());
this.label43 = (new System.Windows.Forms.Label());
this.textBox41 = (new System.Windows.Forms.TextBox());
this.label44 = (new System.Windows.Forms.Label());
this.listBox1;

//
// btnClose
//
this.btnClose.Location = System.Drawing.Point(269, 549);
this.btnClose.Name = S"btnClose";
this.btnClose.Size = System.Drawing.Size(75, 23);
this.btnClose.TabIndex = 1;
this.btnClose.Text = S"Exit";
this.btnClose.Click += new System.EventHandler(this, &Form1::BtnClose_Click);

//
// btnSend
//
this->btnSend->Location = System::Drawing::Point(173, 549);
this->btnSend->Name = S"btnSend";
this->btnSend->Size = System::Drawing::Size(75, 23);
this->btnSend->TabIndex = 2;
this->btnSend->Text = S"Write";
this->btnSend->Click += new System::EventHandler(this, &Form1::BtnSend_Click);

//
// listBox1
//
this->listBox1->Location = System::Drawing::Point(217, 24);
this->listBox1->Name = S"listBox1";
this->listBox1->Size = System::Drawing::Size(116, 238);
this->listBox1->TabIndex = 3;
//
// comboBox1
//
this->comboBox1->Location = System::Drawing::Point(12, 24);
this->comboBox1->Name = S"comboBox1";
this->comboBox1->Size = System::Drawing::Size(156, 21);
this->comboBox1->TabIndex = 4;
this->comboBox1->SelectedIndexChanged += new System::EventHandler(this, &Form1::comboBox1_SelectedIndexChanged);
//
// radioNumber
//
this->radioNumber->Location = System::Drawing::Point(28, 76);
this->radioNumber->Name = S"radioNumber";
this->radioNumber->Size = System::Drawing::Size(104, 24);
this->radioNumber->TabIndex = 5;
this->radioNumber->Text = S"Number";
this->radioNumber->CheckedChanged += new System::EventHandler(this, &Form1::radioNumber_CheckedChanged);
//
// radioDescription
//
this->radioDescription->Location = System::Drawing::Point(28, 100);
this->radioDescription->Name = S"radioDescription";
this->radioDescription->Size = System::Drawing::Size(104, 24);
this->radioDescription->TabIndex = 6;
this->radioDescription->Text = S"Description";
this->radioDescription->CheckedChanged += new System::EventHandler(this, &Form1::radioDescription_CheckedChanged);
//
// radioSerial
//
this->radioSerial->Location = System::Drawing::Point(28, 124);
this->radioSerial->Name = S"radioSerial";
this->radioSerial->Size = System::Drawing::Size(104, 24);
this->radioSerial->TabIndex = 7;
this->radioSerial->Text = S"Serial";
this->radioSerial->CheckedChanged += new System::EventHandler(this, &Form1::radioSerial_CheckedChanged);
//
// groupBox1
//
this->groupBox1->Location = System::Drawing::Point(12, 60);
this->groupBox1->Name = S"groupBox1";
this->groupBox1->Size = System::Drawing::Size(156, 96);
this->groupBox1->TabIndex = 8;
this->groupBox1->TabStop = false;
this->groupBox1->Text = S"List Devices By";
//
// textBoxAngle
//
this->textBoxAngle->Enabled = false;
this->textBoxAngle->Location = System::Drawing::Point(5, 281);
this->textBoxAngle->Name = S"textBoxAngle";
this->textBoxAngle->Size = System::Drawing::Size(76, 20);
this->textBoxAngle->TabIndex = 10;

//
// label1
//
this->label1->AutoSize = true;
this->label1->Location = System::Drawing::Point(9, 8);
this->label1->Name = S"label1";
this->label1->Size = System::Drawing::Size(41, 13);
this->label1->TabIndex = 11;
this->label1->Text = S"Device";

//
// label2
//
this->label2->AutoSize = true;
this->label2->Location = System::Drawing::Point(214, 8);
this->label2->Name = S"label2";
this->label2->Size = System::Drawing::Size(74, 13);
this->label2->TabIndex = 12;
this->label2->Text = S"Incoming data";

//
// groupBox2
//
this->groupBox2->Controls->Add(this->radioDiffV);
this->groupBox2->Controls->Add(this->radioSameV);
this->groupBox2->Controls->Add(this->radioDirect);
this->groupBox2->Location = System::Drawing::Point(12, 165);
this->groupBox2->Name = S"groupBox2";
this->groupBox2->Size = System::Drawing::Size(156, 97);
this->groupBox2->TabIndex = 13;
this->groupBox2->TabStop = false;
this->groupBox2->Text = S"Angle set type";

//
// radioDiffV
//
this->radioDiffV->AutoSize = true;
this->radioDiffV->Location = System::Drawing::Point(16, 65);
this->radioDiffV->Name = S"radioDiffV";
this->radioDiffV->Size = System::Drawing::Size(138, 17);
this->radioDiffV->TabIndex = 5;
this->radioDiffV->TabStop = true;
this->radioDiffV->Text = S"Different voltage for test";
this->radioDiffV->UseVisualStyleBackColor = true;
this->radioDiffV->CheckedChanged += new System::EventHandler(this, &Form1::radioDiffV_CheckedChanged);

//
// radioSameV
//
this->radioSameV->AutoSize = true;
this->radioSameV->Location = System::Drawing::Point(16, 42);
this->radioSameV->Name = S"radioSameV";
this->radioSameV->Size = System::Drawing::Size(125, 17);
this->radioSameV->TabIndex = 4;
this->radioSameV->TabStop = true;
this->radioSameV->Text = S"Same voltage for test";
this->radioSameV->UseVisualStyleBackColor = true;
this->radioSameV->CheckedChanged += new System::EventHandler(this, &Form1::radioSameV_CheckedChanged);

//
// radioDirect
//
this->radioDirect->AutoSize = true;
this->radioDirect->Location = System::Drawing::Point(16, 19);
this->radioDirect->Name = S"radioDirect";
this->radioDirect->Size = System::Drawing::Size(82, 17);
this->radioDirect->TabIndex = 3;
this->radioDirect->TabStop = true;
this->radioDirect->Text = S"Direct angle";
this->radioDirect->UseVisualStyleBackColor = true;
this->radioDirect->CheckedChanged += new System::EventHandler(this, &Form1::radioDirect_CheckedChanged);
//
// label3
//
this->label3->AutoSize = true;
this->label3->Location = System::Drawing::Point(2, 265);
this->label3->Name = S"label3";
this->label3->Size = System::Drawing::Size(34, 13);
this->label3->TabIndex = 14;
this->label3->Text = S"Angle";
//
// label4
//
this->label4->AutoSize = true;
this->label4->Location = System::Drawing::Point(2, 304);
this->label4->Name = S"label4";
this->label4->Size = System::Drawing::Size(72, 13);
this->label4->TabIndex = 16;
this->label4->Text = S"Same voltage";
//
// textBoxSameV
//
this->textBoxSameV->Location = System::Drawing::Point(5, 320);
this->textBoxSameV->Name = S"textBoxSameV";
this->textBoxSameV->Size = System::Drawing::Size(76, 20);
this->textBoxSameV->TabIndex = 15;
//
// label5
//
this->label5->AutoSize = true;
this->label5->Location = System::Drawing::Point(2, 365);
this->label5->Name = S"label5";
this->label5->Size = System::Drawing::Size(20, 13);
this->label5->TabIndex = 18;
this->label5->Text = S"V1";
//
// textBox3
//
this->textBox3->Location = System::Drawing::Point(21, 362);
this->textBox3->Name = S"textBox3";
this->textBox3->Size = System::Drawing::Size(57, 20);
this->textBox3->TabIndex = 17;
//
// label6
//
this->label6->AutoSize = true;
this->label6->Location = System::Drawing::Point(2, 391);
this->label6->Name = S"label6";
this->label6->Size = System::Drawing::Size(20, 13);
this->label6->TabIndex = 20;
this->label6->Text = S"V2";

//
// textBox4
//
this->textBox4->Location = System::Drawing::Point(21, 388);
this->textBox4->Name = S"textBox4";
this->textBox4->Size = System::Drawing::Size(57, 20);
this->textBox4->TabIndex = 19;

//
// label7
//
this->label7->AutoSize = true;
this->label7->Location = System::Drawing::Point(2, 417);
this->label7->Name = S"label7";
this->label7->Size = System::Drawing::Size(20, 13);
this->label7->TabIndex = 22;
this->label7->Text = S"V3";

//
// textBox5
//
this->textBox5->Location = System::Drawing::Point(21, 414);
this->textBox5->Name = S"textBox5";
this->textBox5->Size = System::Drawing::Size(57, 20);
this->textBox5->TabIndex = 21;

//
// label8
//
this->label8->AutoSize = true;
this->label8->Location = System::Drawing::Point(2, 443);
this->label8->Name = S"label8";
this->label8->Size = System::Drawing::Size(20, 13);
this->label8->TabIndex = 24;
this->label8->Text = S"V4";

//
// textBox6
//
this->textBox6->Location = System::Drawing::Point(21, 440);
this->textBox6->Name = S"textBox6";
this->textBox6->Size = System::Drawing::Size(57, 20);
this->textBox6->TabIndex = 23;

//
// label9
//
this->label9->AutoSize = true;
this->label9->Location = System::Drawing::Point(2, 469);
this->label9->Name = S"label9";
this->label9->Size = System::Drawing::Size(20, 13);
this->label9->TabIndex = 26;
this->label9->Text = S"V5";

//
// textBox7
//
this->textBox7->Location = System::Drawing::Point(21, 466);
this->textBox7->Name = S"textBox7";
this->textBox7->Size = System::Drawing::Size(57, 20);
this->textBox7->TabIndex = 25;

//
// label10
this->label10->AutoSize = true;
this->label10->Location = System::Drawing::Point(2, 495);
this->label10->Name = S"label10";
this->label10->Size = System::Drawing::Size(20, 13);
this->label10->TabIndex = 28;
this->label10->Text = S"V6";

this->textBox8->Location = System::Drawing::Point(21, 492);
this->textBox8->Name = S"textBox8";
this->textBox8->Size = System::Drawing::Size(57, 20);
this->textBox8->TabIndex = 27;

this->label11->AutoSize = true;
this->label11->Location = System::Drawing::Point(87, 339);
this->label11->Name = S"label11";
this->label11->Size = System::Drawing::Size(26, 13);
this->label11->TabIndex = 36;
this->label11->Text = S"V10";

this->textBox9->Location = System::Drawing::Point(112, 336);
this->textBox9->Name = S"textBox9";
this->textBox9->Size = System::Drawing::Size(57, 20);
this->textBox9->TabIndex = 35;

this->label12->AutoSize = true;
this->label12->Location = System::Drawing::Point(93, 313);
this->label12->Name = S"label12";
this->label12->Size = System::Drawing::Size(20, 13);
this->label12->TabIndex = 34;
this->label12->Text = S"V9";

this->textBox10->Location = System::Drawing::Point(112, 310);
this->textBox10->Name = S"textBox10";
this->textBox10->Size = System::Drawing::Size(57, 20);
this->textBox10->TabIndex = 33;

this->label13->AutoSize = true;
this->label13->Location = System::Drawing::Point(93, 287);
this->label13->Name = S"label13";
this->label13->Size = System::Drawing::Size(20, 13);
this->label13->TabIndex = 32;
this->label13->Text = S"V8";

this->textBox11->Location = System::Drawing::Point(112, 284);
textBox11->Name = S"textBox11";
textBox11->Size = System::Drawing::Size(57, 20);
textBox11->TabIndex = 31;

label14->AutoSize = true;
label14->Location = System::Drawing::Point(2, 520);
label14->Name = S"label14";
label14->Size = System::Drawing::Size(20, 13);
label14->TabIndex = 30;
label14->Text = S"V7";

textBox12->Location = System::Drawing::Point(21, 517);
textBox12->Name = S"textBox12";
textBox12->Size = System::Drawing::Size(57, 20);
textBox12->TabIndex = 29;

label15->AutoSize = true;
label15->Location = System::Drawing::Point(173, 339);
label15->Name = S"label15";
label15->Size = System::Drawing::Size(26, 13);
label15->TabIndex = 56;
label15->Text = S"V20";

textBox13->Location = System::Drawing::Point(200, 336);
textBox13->Name = S"textBox13";
textBox13->Size = System::Drawing::Size(57, 20);
textBox13->TabIndex = 55;

label16->AutoSize = true;
label16->Location = System::Drawing::Point(175, 313);
label16->Name = S"label16";
label16->Size = System::Drawing::Size(26, 13);
label16->TabIndex = 54;
label16->Text = S"V19";

textBox14->Location = System::Drawing::Point(200, 310);
textBox14->Name = S"textBox14";
textBox14->Size = System::Drawing::Size(57, 20);
textBox14->TabIndex = 53;

label17->AutoSize = true;
label17->Location = System::Drawing::Point(175, 287);
label17->Name = S"label17";
label17->Size = System::Drawing::Size(26, 13);
label17->TabIndex = 52;
this->label17->Text = S"V18";

//
// textBox15
//
this->textBox15->Location = System::Drawing::Point(200, 284);
this->textBox15->Name = S"textBox15";
this->textBox15->Size = System::Drawing::Size(57, 20);
this->textBox15->TabIndex = 51;

//
// label18
//
this->label18->AutoSize = true;
this->label18->Location = System::Drawing::Point(87, 520);
this->label18->Name = S"label18";
this->label18->Size = System::Drawing::Size(26, 13);
this->label18->TabIndex = 50;
this->label18->Text = S"V17";

//
// textBox16
//
this->textBox16->Location = System::Drawing::Point(112, 517);
this->textBox16->Name = S"textBox16";
this->textBox16->Size = System::Drawing::Size(57, 20);
this->textBox16->TabIndex = 49;

//
// label19
//
this->label19->AutoSize = true;
this->label19->Location = System::Drawing::Point(87, 495);
this->label19->Name = S"label19";
this->label19->Size = System::Drawing::Size(26, 13);
this->label19->TabIndex = 48;
this->label19->Text = S"V16";

//
// textBox17
//
this->textBox17->Location = System::Drawing::Point(112, 492);
this->textBox17->Name = S"textBox17";
this->textBox17->Size = System::Drawing::Size(57, 20);
this->textBox17->TabIndex = 47;

//
// label20
//
this->label20->AutoSize = true;
this->label20->Location = System::Drawing::Point(87, 469);
this->label20->Name = S"label20";
this->label20->Size = System::Drawing::Size(26, 13);
this->label20->TabIndex = 46;
this->label20->Text = S"V15";

//
// textBox18
//
this->textBox18->Location = System::Drawing::Point(112, 466);
this->textBox18->Name = S"textBox18";
this->textBox18->Size = System::Drawing::Size(57, 20);
this->textBox18->TabIndex = 45;

//
// label21
//
this->label21->AutoSize = true;
this->label21->Location = System::Drawing::Point(87, 443);
this->label21->Name = S"label21";
this->label21->Size = System::Drawing::Size(26, 13);
this->label21->TabIndex = 44;
this->label21->Text = S"V14";

//
// textBox19
//
this->textBox19->Location = System::Drawing::Point(112, 440);
this->textBox19->Name = S"textBox19";
this->textBox19->Size = System::Drawing::Size(57, 20);
this->textBox19->TabIndex = 43;

//
// label22
//
this->label22->AutoSize = true;
this->label22->Location = System::Drawing::Point(87, 417);
this->label22->Name = S"label22";
this->label22->Size = System::Drawing::Size(26, 13);
this->label22->TabIndex = 42;
this->label22->Text = S"V13";

//
// textBox20
//
this->textBox20->Location = System::Drawing::Point(112, 414);
this->textBox20->Name = S"textBox20";
this->textBox20->Size = System::Drawing::Size(57, 20);
this->textBox20->TabIndex = 41;

//
// label23
//
this->label23->AutoSize = true;
this->label23->Location = System::Drawing::Point(87, 391);
this->label23->Name = S"label23";
this->label23->Size = System::Drawing::Size(26, 13);
this->label23->TabIndex = 40;
this->label23->Text = S"V12";

//
// textBox21
//
this->textBox21->Location = System::Drawing::Point(112, 388);
this->textBox21->Name = S"textBox21";
this->textBox21->Size = System::Drawing::Size(57, 20);
this->textBox21->TabIndex = 39;

//
// label24
//
this->label24->AutoSize = true;
this->label24->Location = System::Drawing::Point(87, 365);
this->label24->Name = S"label24";
this->label24->Size = System::Drawing::Size(26, 13);
this->label24->TabIndex = 38;
this->label24->Text = S"V11";

//
// textBox22
//
this->textBox22->Location = System::Drawing::Point(112, 362);
this->textBox22->Name = S"textBox22";
this->textBox22->Size = System::Drawing::Size(57, 20);
this->textBox22->TabIndex = 37;
//
// label28
//
this->label28->AutoSize = true;
this->label28->Location = System::Drawing::Point(262, 520);
this->label28->Name = S"label28";
this->label28->Size = System::Drawing::Size(26, 13);
this->label28->TabIndex = 90;
this->label28->Text = S"V37";
//
// textBox26
//
this->textBox26->Location = System::Drawing::Point(287, 517);
this->textBox26->Name = S"textBox26";
this->textBox26->Size = System::Drawing::Size(57, 20);
this->textBox26->TabIndex = 89;
//
// label29
//
this->label29->AutoSize = true;
this->label29->Location = System::Drawing::Point(262, 495);
this->label29->Name = S"label29";
this->label29->Size = System::Drawing::Size(26, 13);
this->label29->TabIndex = 88;
this->label29->Text = S"V36";
//
// textBox27
//
this->textBox27->Location = System::Drawing::Point(287, 492);
this->textBox27->Name = S"textBox27";
this->textBox27->Size = System::Drawing::Size(57, 20);
this->textBox27->TabIndex = 87;
//
// label30
//
this->label30->AutoSize = true;
this->label30->Location = System::Drawing::Point(262, 469);
this->label30->Name = S"label30";
this->label30->Size = System::Drawing::Size(26, 13);
this->label30->TabIndex = 86;
this->label30->Text = S"V35";
//
// textBox28
//
this->textBox28->Location = System::Drawing::Point(287, 466);
this->textBox28->Name = S"textBox28";
this->textBox28->Size = System::Drawing::Size(57, 20);
this->textBox28->TabIndex = 85;
//
// label31
//
this->label31->AutoSize = true;
this->label31->Location = System::Drawing::Point(262, 443);
this->label31->Name = S"label31";
this->label31->Size = System::Drawing::Size(26, 13);
this->label31->TabIndex = 84;
this->label31->Text = S"V34";
this->textBox29->Location = System::Drawing::Point(287, 440);
this->textBox29->Name = S"textBox29";
this->textBox29->Size = System::Drawing::Size(57, 20);
this->textBox29->TabIndex = 83;

this->label32->AutoSize = true;
this->label32->Location = System::Drawing::Point(262, 417);
this->label32->Name = S"label32";
this->label32->Size = System::Drawing::Size(26, 13);
this->label32->TabIndex = 82;
this->label32->Text = S"V33";

this->textBox30->Location = System::Drawing::Point(287, 414);
this->textBox30->Name = S"textBox30";
this->textBox30->Size = System::Drawing::Size(57, 20);
this->textBox30->TabIndex = 81;

this->label33->AutoSize = true;
this->label33->Location = System::Drawing::Point(262, 391);
this->label33->Name = S"label33";
this->label33->Size = System::Drawing::Size(26, 13);
this->label33->TabIndex = 80;
this->label33->Text = S"V32";

this->textBox31->Location = System::Drawing::Point(287, 388);
this->textBox31->Name = S"textBox31";
this->textBox31->Size = System::Drawing::Size(57, 20);
this->textBox31->TabIndex = 79;

this->label34->AutoSize = true;
this->label34->Location = System::Drawing::Point(262, 365);
this->label34->Name = S"label34";
this->label34->Size = System::Drawing::Size(26, 13);
this->label34->TabIndex = 78;
this->label34->Text = S"V31";

this->textBox32->Location = System::Drawing::Point(287, 362);
this->textBox32->Name = S"textBox32";
this->textBox32->Size = System::Drawing::Size(57, 20);
this->textBox32->TabIndex = 77;

this->label35->AutoSize = true;
this->label35->Location = System::Drawing::Point(262, 339);
this->label35->Name = S"label35";
this->label35->Size = System::Drawing::Size(26, 13);
this->label35->TabIndex = 76;
this->label35->Text = S"V30";
//
// textBox33
//
this->textBox33->Location = System::Drawing::Point(287, 336);
this->textBox33->Name = S"textBox33";
this->textBox33->Size = System::Drawing::Size(57, 20);
this->textBox33->TabIndex = 75;
//
// label36
//
this->label36->AutoSize = true;
this->label36->Location = System::Drawing::Point(262, 313);
this->label36->Name = S"label36";
this->label36->Size = System::Drawing::Size(26, 13);
this->label36->TabIndex = 74;
this->label36->Text = S"V29";
//
// textBox34
//
this->textBox34->Location = System::Drawing::Point(287, 310);
this->textBox34->Name = S"textBox34";
this->textBox34->Size = System::Drawing::Size(57, 20);
this->textBox34->TabIndex = 73;
//
// label37
//
this->label37->AutoSize = true;
this->label37->Location = System::Drawing::Point(262, 287);
this->label37->Name = S"label37";
this->label37->Size = System::Drawing::Size(26, 13);
this->label37->TabIndex = 72;
this->label37->Text = S"V28";
//
// textBox35
//
this->textBox35->Location = System::Drawing::Point(287, 284);
this->textBox35->Name = S"textBox35";
this->textBox35->Size = System::Drawing::Size(57, 20);
this->textBox35->TabIndex = 71;
//
// label38
//
this->label38->AutoSize = true;
this->label38->Location = System::Drawing::Point(175, 520);
this->label38->Name = S"label38";
this->label38->Size = System::Drawing::Size(26, 13);
this->label38->TabIndex = 70;
this->label38->Text = S"V27";
//
// textBox36
//
this->textBox36->Location = System::Drawing::Point(200, 517);
this->textBox36->Name = S"textBox36";
this->textBox36->Size = System::Drawing::Size(57, 20);
this->textBox36->TabIndex = 69;
//
// label39
//
this->label39->AutoSize = true;
this->label39->Location = System::Drawing::Point(175, 495);
this->label39->Name = S"label39";
this->label39->Size = System::Drawing::Size(26, 13);
this->label39->TabIndex = 68;
this->label39->Text = S"V26";
//
// textBox37
//
this->textBox37->Location = System::Drawing::Point(200, 492);
this->textBox37->Name = S"textBox37";
this->textBox37->Size = System::Drawing::Size(57, 20);
this->textBox37->TabIndex = 67;
//
// label40
//
this->label40->AutoSize = true;
this->label40->Location = System::Drawing::Point(175, 469);
this->label40->Name = S"label40";
this->label40->Size = System::Drawing::Size(26, 13);
this->label40->TabIndex = 66;
this->label40->Text = S"V25";
//
// textBox38
//
this->textBox38->Location = System::Drawing::Point(200, 466);
this->textBox38->Name = S"textBox38";
this->textBox38->Size = System::Drawing::Size(57, 20);
this->textBox38->TabIndex = 65;
//
// label41
//
this->label41->AutoSize = true;
this->label41->Location = System::Drawing::Point(175, 443);
this->label41->Name = S"label41";
this->label41->Size = System::Drawing::Size(26, 13);
this->label41->TabIndex = 64;
this->label41->Text = S"V24";
//
// textBox39
//
this->textBox39->Location = System::Drawing::Point(200, 440);
this->textBox39->Name = S"textBox39";
this->textBox39->Size = System::Drawing::Size(57, 20);
this->textBox39->TabIndex = 63;
//
// label42
//
this->label42->AutoSize = true;
this->label42->Location = System::Drawing::Point(175, 417);
this->label42->Name = S"label42";
this->label42->Size = System::Drawing::Size(26, 13);
this->label42->TabIndex = 62;
this->label42->Text = S"V23";
this->textBox40->Location = System::Drawing::Point(200, 414);
this->textBox40->Name = S"textBox40";
this->textBox40->Size = System::Drawing::Size(57, 20);
this->textBox40->TabIndex = 61;

this->label43->AutoSize = true;
this->label43->Location = System::Drawing::Point(175, 391);
this->label43->Name = S"label43";
this->label43->Size = System::Drawing::Size(26, 13);
this->label43->TabIndex = 60;
this->label43->Text = S"V22";

this->textBox41->Location = System::Drawing::Point(200, 388);
this->textBox41->Name = S"textBox41";
this->textBox41->Size = System::Drawing::Size(57, 20);
this->textBox41->TabIndex = 59;

this->label44->AutoSize = true;
this->label44->Location = System::Drawing::Point(175, 365);
this->label44->Name = S"label44";
this->label44->Size = System::Drawing::Size(26, 13);
this->label44->TabIndex = 58;
this->label44->Text = S"V21";

this->textBox42->Location = System::Drawing::Point(200, 362);
this->textBox42->Name = S"textBox42";
this->textBox42->Size = System::Drawing::Size(57, 20);
this->textBox42->TabIndex = 57;

this->label25->AutoSize = true;
this->label25->Location = System::Drawing::Point(2, 346);
this->label25->Name = S"label25";
this->label25->Size = System::Drawing::Size(85, 13);
this->label25->TabIndex = 91;
this->label25->Text = S"Different voltage";

this->AutoScaleBaseSize = System::Drawing::Size(5, 13);
this->ClientSize = System::Drawing::Size(355, 584);
this->Controls->Add(this->label25);
this->Controls->Add(this->label28);
this->Controls->Add(this->textBox26);
this->Controls->Add(this->label29);
this->Controls->Add(this->textBox27);
this->Controls->Add(this->label30);
this->Controls->Add(this->textBox28);
```csharp
    Controls->Add(this->textBox7);
    Controls->Add(this->label8);
    Controls->Add(this->textBox6);
    Controls->Add(this->label7);
    Controls->Add(this->textBox5);
    Controls->Add(this->label6);
    Controls->Add(this->textBox4);
    Controls->Add(this->label5);
    Controls->Add(this->textBox3);
    Controls->Add(this->label4);
    Controls->Add(this->textBoxSameV);
    Controls->Add(this->groupBox2);
    Controls->Add(this->label2);
    Controls->Add(this->label1);
    Controls->Add(this->radioSerial);
    Controls->Add(this->radioDescription);
    Controls->Add(this->radioNumber);
    Controls->Add(this->groupBox1);
    Controls->Add(this->groupBox2);
    Controls->Add(this->label13);
    Controls->Add(this->groupBox2);
    Controls->Add(this->label12);
    Controls->Add(this->groupBox1);
    Controls->Add(this->label11);
    Controls->Add(this->groupBox2);
    Controls->Add(this->radioSerial);
    Controls->Add(this->radioDescription);
    Controls->Add(this->radioNumber);
    Controls->Add(this->comboBox1);
    Controls->Add(this->listBox1);
    Controls->Add(this->btnSend);
    Controls->Add(this->btnClose);
    Controls->Add(this->groupBox1);
    Controls->Add(this->textboxAngle);
    Location = System::Drawing::Point(56, 48);
    Name = S"Form1";
    Text = S"FTDI Loopback Test";
    Load += new System::EventHandler(this,&Form1::Form1_Load);
    groupBox2->ResumeLayout(false);
    PerformLayout();
    ResumeLayout(false);
    PerformLayout();

    void Form1::ReadingProc()
    {
        ArrayList * palReadData;
        palReadData = new ArrayList();
        TimeSpan waitTime = TimeSpan(0, 0, 1); // 1 second timeout
        static char c = 0;
        char* pstr6 = &c;
        FT_STATUS status;
        int i = 0;
        String* s;
        float data;

        while(bContinue)
        {
            DWORD dwRead, dwRXBytes;
            Byte b;
            NumberFormatInfo* provider = new NumberFormatInfo();

            WaitForSingleObject(hEvent, -1);
            if(handle)
            {
                status = FT_GetQueueStatus(handle, &dwRead);
                if(status != FT_OK)
                {
                    MessageBox::Show("GError");
                }
            }
        }
    }
```
for(int i=0;i<37*3;i++)
{
    buf[i] = 0;
}

while(dwRead && bContinue)
{
    status = FT_Read(handle, &b, 1, &dwRXBytes);
    if(status != FT_OK)
    {
        MessageBox::Show("R Error");
        continue;
    }
    else
    {
        s = Convert::ToString(b);
        buf[i] = b;
        i++;
    }
    status = FT_GetQueueStatus(handle, &dwRead);
}

Thread::Sleep(0);

if(handle)
{
    FT_Close(handle);
}

void Form1::setVoltageTextBox(bool b)
{
    textBox3->Enabled = b;
    textBox4->Enabled = b;
    textBox5->Enabled = b;
    textBox6->Enabled = b;
    textBox7->Enabled = b;
    textBox8->Enabled = b;
    textBox9->Enabled = b;
    textBox10->Enabled = b;
    textBox11->Enabled = b;
    textBox12->Enabled = b;
    textBox13->Enabled = b;
    textBox14->Enabled = b;
    textBox15->Enabled = b;
    textBox16->Enabled = b;
    textBox17->Enabled = b;
    textBox18->Enabled = b;
    textBox19->Enabled = b;
    textBox20->Enabled = b;
    textBox21->Enabled = b;
    textBox22->Enabled = b;
textBox26->Enabled = b;
textBox27->Enabled = b;
textBox28->Enabled = b;
textBox29->Enabled = b;
textBox30->Enabled = b;
textBox31->Enabled = b;
textBox32->Enabled = b;
textBox33->Enabled = b;
textBox34->Enabled = b;
textBox35->Enabled = b;
textBox36->Enabled = b;
textBox37->Enabled = b;
textBox38->Enabled = b;
textBox39->Enabled = b;
textBox40->Enabled = b;
textBox41->Enabled = b;
textBox42->Enabled = b;
namespace USBFT245R
{
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::Threading;
    using namespace System::Text;
    using namespace System::Globalization;
    using namespace System::Runtime::InteropServices;
    using namespace std;

    const UInt32 FT_LIST_NUMBER_ONLY = 0x80000000;
    const UInt32 FT_LIST_BY_INDEX = 0x40000000;
    const UInt32 FT_LIST_ALL = 0x20000000;
    const UInt32 FT_OPEN_BY_SERIAL_NUMBER = 1;
    const UInt32 FT_OPEN_BY_DESCRIPTION = 2;

    const UInt32 FT_EVENT_RXCHAR = 1;
    const UInt32 FT_EVENT_MODEM_STATUS = 2;

    #define FT_PREFIX [DllImport("FTD2XX.dll")]

    enum {
        FT_OK,
        FT_INVALID_HANDLE,
        FT_DEVICE_NOT_FOUND,
        FT_DEVICE_NOT_OPENED,
        FT_IO_ERROR,
        FT_INSUFFICIENT_RESOURCES,
        FT_INVALID_PARAMETER,
        FT_INVALID_BAUD_RATE,
        FT_DEVICE_NOT_OPENED_FOR_ERASE,
        FT_DEVICE_NOT_OPENED_FOR_WRITE,
        FT_FAILED_TO_WRITE_DEVICE,
        FT_EEPROM_READ_FAILED,
        FT_EEPROM_WRITE_FAILED,
        FT_EEPROM_ERASE_FAILED,
        FT_EEPROM_NOT_PRESENT,
        FT_EEPROM_NOT_PROGRAMMED,
        FT_INVALID_ARGS,
        FT_NOT_SUPPORTED,
        FT_OTHER_ERROR
    };

typedef void * FT_HANDLE;
typedef unsigned long DWORD;
typedef unsigned long FT_STATUS;
typedef void * LPVOID;
typedef void * PVOID;
typedef DWORD * LPDWORD;
typedef DWORD ULONG;
typedef unsigned short USHORT;
typedef unsigned char UCHAR;
typedef unsigned short WORD;
typedef WORD * LPWORD;
typedef unsigned char UCHAR;
typedef UCHAR * PUCHAR;
typedef char CHAR;
typedef CHAR * PCHAR;
typedef ULONG FT_DEVICE;
typedef void HANDLE;
typedef int BOOL;
#define FALSE 0
#define TRUE 1

// as c++. net is a managed application and our ftd2xx.dll is unmanaged code you must declare the functions here explicitly
// to allow you to call them within the application. An include file and the .lib file simply wont work with c++.net. Its
// a similar problem in c#.
// I have only included these 4 functions in this to show you how to do this. For other functions that you require similar
// declerations will need to go here.

FT_PREFIX FT_STATUS FT_Open (int deviceNumber, FT_HANDLE * pHandle);
FT_PREFIX FT_STATUS FT_OpenEx (PVOID pArg1, DWORD Flags, FT_HANDLE * pHandle);
FT_PREFIX FT_STATUS FT_ListDevices (PVOID pArg1, PVOID pArg2, DWORD Flags);
FT_PREFIX FT_STATUS FT_ListDevices (UInt32 pvArg1, void * pvArg2, UInt32 dwFlags);  // FT_ListDevices by serial number or description by index only
FT_PREFIX FT_STATUS FT_Close (FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_Read (FT_HANDLE ftHandle, LPVOID lpBuffer, DWORD nBufferSize, LPDWORD lpBytesReturned);
FT_PREFIX FT_STATUS FT_Write (FT_HANDLE ftHandle, LPVOID lpBuffer, DWORD nBufferSize, LPDWORD lpBytesWritten);
FT_PREFIX FT_STATUS FT_SetBaudRate (FT_HANDLE ftHandle, ULONG BaudRate);
FT_PREFIX FT_STATUS FT_SetDivisor (FT_HANDLE ftHandle, USHORT Divisor);
FT_PREFIX FT_STATUS FT_SetDataCharacteristics (FT_HANDLE ftHandle, UCHAR WordLength, UCHAR StopBits, UCHAR Parity);
FT_PREFIX FT_STATUS FT_SetFlowControl (FT_HANDLE ftHandle, USHORT FlowControl, UCHAR XonChar, UCHAR XoffChar);
FT_PREFIX FT_STATUS FT_ResetDevice (FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_SetDtr (FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_ClrDtr (FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_SetRts (FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_ClrRts (FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_GetModemStatus (FT_HANDLE ftHandle, ULONG * pModemStatus);
FT_PREFIX FT_STATUS FT_SetChars (FT_HANDLE ftHandle, UCHAR EventChar, UCHAR EventCharEnabled, UCHAR ErrorChar, UCHAR ErrorCharEnabled);
FT_PREFIX FT_STATUS FT_Purge (FT_HANDLE ftHandle, ULONG Mask);
FT_PREFIX FT_STATUS FT_SetTimeouts (FT_HANDLE ftHandle, ULONG ReadTimeout, ULONG WriteTimeout);
FT_PREFIX FT_STATUS FT_GetQueueStatus (FT_HANDLE ftHandle, DWORD * dwRxBytes);
FT_PREFIX FT_STATUS FT_SetEventNotification (FT_HANDLE ftHandle, DWORD Mask, PVOID Param);
FT_PREFIX FT_STATUS FT_GetStatus (FT_HANDLE ftHandle, DWORD * dwRxBytes, DWORD * dwTxBYtes, DWORD * dwEvtDWord);
FT_PREFIX FT_STATUS FT_SetBreakOn (FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_SetBreakOff (FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_SetWaitMask(FT_HANDLE ftHandle, DWORD Mask);
FT_PREFIX FT_STATUS FT_WaitOnMask(FT_HANDLE ftHandle, DWORD *Mask);
FT_PREFIX FT_STATUS FT_GetEventStatus(FT_HANDLE ftHandle, DWORD *dwEventDWord);
FT_PREFIX FT_STATUS FT_ReadEE(FT_HANDLE ftHandle, DWORD dwWordOffset, LPWORD lpwValue);
FT PREFIX FT_STATUS FT_WriteEE(FT_HANDLE ftHandle, DWORD dwWordOffset, WORD wValue);
FT_PREFIX FT_STATUS FT_EraseEE(FT_HANDLE ftHandle);

// Missed out the programming stuff +++

FT_PREFIX FT_STATUS FT_EE_UASize(FT_HANDLE ftHandle, LPDWORD lpdwSize);
FT_PREFIX FT_STATUS FT_EE_UAWrite(FT_HANDLE ftHandle, UCHAR ucMask, UCHAR ucEnable);
FT_PREFIX FT_STATUS FT_EE_UARead(FT_HANDLE ftHandle, UCHAR ucMode);

FT_PREFIX FT_STATUS FT_SetLatencyTimer(FT_HANDLE ftHandle, UCHAR ucLatency);
FT_PREFIX FT_STATUS FT_GetLatencyTimer(FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_SetUSBParameters(FT_HANDLE ftHandle, ULONG ulInTransferSize, ULONG ulOutTransferSize);
FT_PREFIX FT_STATUS FT_GetDeviceInfo(FT_HANDLE ftHandle, FT_DEVICE *lpftDevice, LPDWORD lpdwID, PCHAR SerialNumber, PCHAR Description, LPVOID Dummy);
FT_PREFIX FT_STATUS FT_StopInTask(FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_RestartInTask(FT_HANDLE ftHandle);
FT_PREFIX FT_STATUS FT_SetResetPipeRetryCount(FT_HANDLE ftHandle, DWORD dwCount);
FT_PREFIX FT_STATUS FT_ResetPort(FT_HANDLE ftHandle);

// need these kernel functions for the Event Handling stuff
[DllImport("Kernel32.dll")] DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds);
[DllImport("Kernel32.dll")] HANDLE CreateEvent(void *pNULL, BOOL bManualReset, BOOL bInitialState, char *pcNULL);
[DllImport("Kernel32.dll")] BOOL SetEvent(HANDLE hEvent);

FT_HANDLE handle;
HANDLE hEvent;
HANDLE DEvent;
char buf[((37*3)+3)];

/// <summary>
/// Summary for Form1
/// 
/// WARNING: If you change the name of this class, you will need to change the
/// 'Resource File Name' property for the managed resource compiler tool
/// associated with all .resx files this class depends on. Otherwise,
/// the designers will not be able to interact properly with localized
/// resources associated with this form.
/// </summary>

public __delegate void addItemsEventHandler();

public __gc class Form1 : public System::Windows::Forms::Form
{
public:
    Form1(void);

public __event addItemsEventHandler* addItemsEvent;

protected:
    void Dispose(bool disposing);

private:
    System::Windows::Forms::Button* btnClose;
    System::Windows::Forms::Button* btnSend;
    System::Windows::Forms::Button* eventAdd;

    System::Threading::Thread* thrdReader;

    System::Windows::Forms::ListBox* listBox1;

    System::Windows::Forms::ComboBox* comboBox1;

    System::ComponentModel::Container* components;

    System::Windows::Forms::GroupBox* groupBox1;
    System::Windows::Forms::GroupBox* groupBox2;

    System::Windows::Forms::RadioButton* radioNumber;
    System::Windows::Forms::RadioButton* radioDescription;
    System::Windows::Forms::RadioButton* radioSerial;
    System::Windows::Forms::RadioButton* radioDirect;
    System::Windows::Forms::RadioButton* radioDiffV;
    System::Windows::Forms::RadioButton* radioSameV;

    System::Windows::Forms::Label* label1;
    System::Windows::Forms::Label* label2;
    System::Windows::Forms::Label* label3;
    System::Windows::Forms::Label* label4;
    System::Windows::Forms::Label* label5;
    System::Windows::Forms::Label* label6;
    System::Windows::Forms::Label* label7;
    System::Windows::Forms::Label* label8;
    System::Windows::Forms::Label* label9;
    System::Windows::Forms::Label* label10;
    System::Windows::Forms::Label* label11;
    System::Windows::Forms::Label* label12;
    System::Windows::Forms::Label* label13;
    System::Windows::Forms::Label* label14;
    System::Windows::Forms::Label* label15;
    System::Windows::Forms::Label* label16;
    System::Windows::Forms::Label* label17;
    System::Windows::Forms::Label* label18;
    System::Windows::Forms::Label* label19;
    System::Windows::Forms::Label* label20;
    System::Windows::Forms::Label* label21;
    System::Windows::Forms::Label* label22;
    System::Windows::Forms::Label* label23;
    System::Windows::Forms::Label* label24;
    System::Windows::Forms::Label* label25;
    System::Windows::Forms::Label* label26;
    System::Windows::Forms::Label* label27;
    System::Windows::Forms::Label* label28;
    System::Windows::Forms::Label* label29;
    System::Windows::Forms::Label* label30;
System::Windows::Forms::Label* label31;
System::Windows::Forms::Label* label32;
System::Windows::Forms::Label* label33;
System::Windows::Forms::Label* label34;
System::Windows::Forms::Label* label35;
System::Windows::Forms::Label* label36;
System::Windows::Forms::Label* label37;
System::Windows::Forms::Label* label38;
System::Windows::Forms::Label* label39;
System::Windows::Forms::Label* label40;
System::Windows::Forms::Label* label41;
System::Windows::Forms::Label* label42;
System::Windows::Forms::Label* label43;
System::Windows::Forms::Label* label44;

System::Windows::Forms::TextBox* textBoxAngle;
System::Windows::Forms::TextBox* textBoxSameV;
System::Windows::Forms::TextBox* textBox3;
System::Windows::Forms::TextBox* textBox4;
System::Windows::Forms::TextBox* textBox5;
System::Windows::Forms::TextBox* textBox6;
System::Windows::Forms::TextBox* textBox7;
System::Windows::Forms::TextBox* textBox8;
System::Windows::Forms::TextBox* textBox9;
System::Windows::Forms::TextBox* textBox10;
System::Windows::Forms::TextBox* textBox11;
System::Windows::Forms::TextBox* textBox12;
System::Windows::Forms::TextBox* textBox13;
System::Windows::Forms::TextBox* textBox14;
System::Windows::Forms::TextBox* textBox15;
System::Windows::Forms::TextBox* textBox16;
System::Windows::Forms::TextBox* textBox17;
System::Windows::Forms::TextBox* textBox18;
System::Windows::Forms::TextBox* textBox19;
System::Windows::Forms::TextBox* textBox20;
System::Windows::Forms::TextBox* textBox21;
System::Windows::Forms::TextBox* textBox22;
System::Windows::Forms::TextBox* textBox23;
System::Windows::Forms::TextBox* textBox24;
System::Windows::Forms::TextBox* textBox25;
System::Windows::Forms::TextBox* textBox26;
System::Windows::Forms::TextBox* textBox27;
System::Windows::Forms::TextBox* textBox28;
System::Windows::Forms::TextBox* textBox29;
System::Windows::Forms::TextBox* textBox30;
System::Windows::Forms::TextBox* textBox31;
System::Windows::Forms::TextBox* textBox32;
System::Windows::Forms::TextBox* textBox33;
System::Windows::Forms::TextBox* textBox34;
System::Windows::Forms::TextBox* textBox35;
System::Windows::Forms::TextBox* textBox36;
System::Windows::Forms::TextBox* textBox37;
System::Windows::Forms::TextBox* textBox38;
System::Windows::Forms::TextBox* textBox39;
System::Windows::Forms::TextBox* textBox40;
System::Windows::Forms::TextBox* textBox41;
System::Windows::Forms::TextBox* textBox42;

bool bContinue;
UInt32 dwOpenFlags;
ManualResetEvent* ev;
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
void InitializeComponent(void);
void ReadingProc();
void setVoltageTextBox(bool);

System::Void ButtonCancel_Click(System::Object * sender, System::EventArgs * e)
{
    Application::Exit();
}

System::Void OpenPort()
{
    FT_STATUS ftStatus = FT_OK;
    if(dwOpenFlags == FT_LIST_NUMBER_ONLY)
    {
        int iSelIndex = comboBox1->SelectedIndex;
        if(iSelIndex >= 0)
            ftStatus = FT_Open(iSelIndex, &handle);
    }
    else
    {
        String * str;
        char cBuf[64];
        str = comboBox1->GetItemText(comboBox1->SelectedItem);
        for(int i = 0; i < str->Length; i++)
        {
            cBuf[i] = str->Chars[i];
        }
        ftStatus = FT_OpenEx(cBuf, dwOpenFlags, &handle);
    }
}

System::Void ClosePort()
{
    if(handle)
    {
        FT_Close(handle);
        handle = NULL;
    }
}

System::Void btnSend_Click(System::Object * sender, System::EventArgs * e)
{
    FT_STATUS ftStatus = FT_OK;
    DWORD ret;
    int size = 0;
    char cBuf[150];
    String* s;
    float data1 = 0;
    float data2 = 0;
    float data3 = 0;
    char test1 = 0;
    char test2 = 0;
char test3 = 0;
int angle = 0;
int data = 1;
float newVolt = 0;
float data = 0;

// Get the direct angle value and generate the USB data packet
if (radioDirect->Checked == true) {
    float tAngle[61][36] = {
        {3.5, 3.2, 2.9, 2.6, 2.2, 1.7, 1.0, 0.1, 1.4, 1.7, 1.0},
        {1.5, 1.0, 0.7, 0.4, 0.3, 0.2, 0.1, 0.0, 0.1, 0.2, 0.3},
        {2.9, 2.6, 2.2, 1.7, 1.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6},
        {1.0, 0.1, 1.0, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9},
        {3.7, 3.4, 3.1, 2.8, 2.5, 2.2, 1.9, 1.6, 1.3, 1.0, 0.7},
        {4.2, 3.8, 3.6, 3.3, 3.1, 2.8, 2.5, 2.2, 1.9, 1.6, 1.3},
        {3.6, 3.3, 3.1, 2.8, 2.6, 2.3, 1.9, 1.6, 1.3, 1.0, 0.7},
        {3.3, 3.1, 2.8, 2.6, 2.3, 1.9, 1.6, 1.3, 1.0, 0.7, 0.4},
        {3.7, 3.5, 3.2, 3.0, 2.8, 2.6, 2.3, 1.9, 1.6, 1.3, 1.0},
        {3.3, 3.1, 2.9, 2.7, 2.4, 2.2, 1.9, 1.6, 1.3, 1.0, 0.7},
        {3.2, 3.0, 2.8, 2.5, 2.2, 1.9, 1.6, 1.3, 1.0, 0.7},
        {3.3, 3.1, 2.9, 2.7, 2.5, 2.2, 1.7, 1.4, 1.1, 0.8, 0.5},
        {3.1, 2.9, 2.7, 2.5, 2.2, 1.7, 1.4, 1.1, 0.8, 0.5, 0.2},
        {3.2, 2.9, 2.7, 2.5, 2.2, 1.7, 1.4, 1.1, 0.8, 0.5, 0.2},
        {3.0, 2.8, 2.6, 2.3, 2.0, 1.6, 1.1, 0.5, 0.4, 0.3, 0.2},
        {2.9, 2.7, 2.5, 2.2, 1.8, 1.4, 0.9, 0.2, 0.4, 0.3, 0.2},
        {3.1, 2.8, 2.6, 2.4, 2.1, 1.7, 1.3, 0.7, 0.4, 0.3, 0.2},
        {2.7, 2.4, 2.1, 1.8, 1.4, 0.9, 0.2, 0.4, 0.3, 0.2, 0.1},
        {2.7, 2.5, 2.2, 1.9, 1.5, 1.0, 0.5, 0.4, 0.3, 0.2, 0.1},
        {3.0, 2.8, 2.6, 2.3, 2.0, 1.6, 1.1, 0.5, 0.4, 0.3, 0.2},
        {2.5, 2.2, 1.9, 1.5, 1.0, 0.5, 0.4, 0.3, 0.2, 0.1},
        {2.9, 2.7, 2.5, 2.3, 2.0, 1.6, 1.1, 0.5, 0.4, 0.3, 0.2},
        {1.5, 1.1, 0.6, 0.0, 0.4, 0.3, 0.2, 0.1, 0.0, 0.0, 0.0},
        {1.4, 1.0, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0, 0.0, 0.0, 0.0},
        {2.8, 2.7, 2.5, 2.3, 2.0, 1.7, 1.4, 1.1, 0.8, 0.5, 0.2},
        {3.8, 3.6, 3.4, 3.2, 3.1, 3.0, 2.8, 2.7, 2.5, 2.3, 2.1},
        {0.8, 0.4, 0.3, 0.2, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}};
else if (radioDiffV->Checked == true)
{
    int i = 1;
cBuf[0] = 0x02;
    size = 0;
    size = textBox3->TextLength;
    if (size>0)
    {
        for(int i = 1; i<size+1; i++)
        {
            cBuf[i] = s->String::Chars[i-1];
        }
    }
}

else if (radioSameV->Checked == true)
{
    size = textBoxSameV->TextLength;
    s = textBoxSameV->Text;
    cBuf[0] = 0xC2;
    angle = Convert::ToInt32(s);
    angle += 31;

    for(int i = 0; i<36;i++)
    {
        newVolt = tAngle[angle-1][i];

        cBuf[index] = ((int)(newVolt)) + 48;
        cBuf[index+1] = '.';

        data = (int)((System::Math::Round(newVolt*10))-((int)(newVolt)*10));

        cBuf[index+2] = ((char)data) + 48;

        index += 3;
    }
    size = 37*3;
}

// Get the same voltage value and generate the USB data packet
else if (radioSameV->Checked == true)
{
    size = textBoxSameV->TextLength;
    s = textBoxSameV->Text;
    cBuf[0] = 0x0E;
    for(int i = 1; i<size+1;i++)
    {
        cBuf[i] = s->String::Chars[i-1];
    }
    size++;
}

// Get the different voltage value and generate the USB data packet
else if (radioDiffV->Checked == true)
{
    int i = 1;
cBuf[0] = 0x02;
    size = 0;
    size = textBox3->TextLength;
    if (size>0)
    {
        for(int i = 1; i<size+1; i++)
        {
            cBuf[i] = s->String::Chars[i-1];
        }
    }
}
s = textBox3->Text;
for(int j = 0; j<size; j++)
{
    cBuf[i] = s->String::Chars[j];
    i++;
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox4->TextLength;
if(size>0)
{
    s = textBox4->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox5->TextLength;
if(size>0)
{
    s = textBox5->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox6->TextLength;
if(size>0)
{
    s = textBox6->Text;
    for(int j = 0; j<size; j++)
    {
{ 
    cBuf[i] = s->String::Chars[j];
    i++;
}
} 
else 
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox7->TextLength;
if(size>0)
{
    s = textBox7->Text;
    for(int j = 0;j<size;j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else 
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox8->TextLength;
if(size>0)
{
    s = textBox8->Text;
    for(int j = 0;j<size;j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else 
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox12->TextLength;
if(size>0)
{
    s = textBox12->Text;
    for(int j = 0;j<size;j++)
    {
        cBuf[i] = s->String::Chars[j];
    }
}


```c
    i++; 
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox11->TextLength;
if(size>0)
{
    s = textBox11->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox10->TextLength;
if(size>0)
{
    s = textBox10->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox9->TextLength;
if(size>0)
{
    s = textBox9->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
```
else
{
cBuf[i] = '9';
cBuf[i+1] = '.';
cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox22->TextLength;
if(size>0)
{
s = textBox22->Text;
for(int j = 0;j<size;j++)
{
cBuf[i] = s->String::Chars[j];
i++;
}
}
else
{
cBuf[i] = '9';
cBuf[i+1] = '.';
cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox21->TextLength;
if(size>0)
{
s = textBox21->Text;
for(int j = 0;j<size;j++)
{
cBuf[i] = s->String::Chars[j];
i++;
}
}
else
{
cBuf[i] = '9';
cBuf[i+1] = '.';
cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox20->TextLength;
if(size>0)
{
s = textBox20->Text;
for(int j = 0;j<size;j++)
{
cBuf[i] = s->String::Chars[j];
i++;
}
}
else
{  cBuf[i] = '9';
  cBuf[i+1] = '.';
  cBuf[i+2] = '9';
  i+=3;
}

size = 0;
size = textBox19->TextLength;
if(size>0)
{
  s = textBox19->Text;
  for(int j = 0;j<size;j++)
  {
    cBuf[i] = s->String::Chars[j];
    i++;
  }
}
else
{
  cBuf[i] = '9';
  cBuf[i+1] = '.';
  cBuf[i+2] = '9';
  i+=3;
}

size = 0;
size = textBox18->TextLength;
if(size>0)
{
  s = textBox18->Text;
  for(int j = 0;j<size;j++)
  {
    cBuf[i] = s->String::Chars[j];
    i++;
  }
}
else
{
  cBuf[i] = '9';
  cBuf[i+1] = '.';
  cBuf[i+2] = '9';
  i+=3;
}

size = 0;
size = textBox17->TextLength;
if(size>0)
{
  s = textBox17->Text;
  for(int j = 0;j<size;j++)
  {
    cBuf[i] = s->String::Chars[j];
    i++;
  }
}
else
{
  cBuf[i] = '9';
cBuf[i+1] = '.';
cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox16->TextLength;
if(size>0)
{
    s = textBox16->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
cBuf[i+1] = '.';
cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox15->TextLength;
if(size>0)
{
    s = textBox15->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
cBuf[i+1] = '.';
cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox14->TextLength;
if(size>0)
{
    s = textBox14->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
cBuf[i+1] = '.';
cBuf[i+2] = '9';
}
size = 0;
size = textBox13->TextLength;
if(size>0)
{
    s = textBox13->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
cBuf[i+1] = '.';
cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox42->TextLength;
if(size>0)
{
    s = textBox42->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
cBuf[i+1] = '.';
cBuf[i+2] = '9';
i+=3;
}

size = 0;
size = textBox41->TextLength;
if(size>0)
{
    s = textBox41->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
cBuf[i+1] = '.';
cBuf[i+2] = '9';
i+=3;
size = 0;
size = textBox40->TextLength;
if(size>0)
{
    s = textBox40->Text;
    for(int j = 0;j<size;j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}
size = 0;
size = textBox39->TextLength;
if(size>0)
{
    s = textBox39->Text;
    for(int j = 0;j<size;j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}
size = 0;
size = textBox38->TextLength;
if(size>0)
{
    s = textBox38->Text;
    for(int j = 0;j<size;j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}
size = 0;
size = textBox37->TextLength;
if(size>0)
{
    s = textBox37->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox36->TextLength;
if(size>0)
{
    s = textBox36->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox35->TextLength;
if(size>0)
{
    s = textBox35->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox34->TextLength;
if(size>0)
{

s = textBox34->Text;
for(int j = 0; j<size; j++)
{
    cBuf[i] = s->String::Chars[j];
    i++;
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}
size = 0;
size = textBox33->TextLength;
if(size>0)
{
    s = textBox33->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}
size = 0;
size = textBox32->TextLength;
if(size>0)
{
    s = textBox32->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}
size = 0;
size = textBox31->TextLength;
if(size>0)
{
    s = textBox31->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}


```cpp

{  
    cBuf[i] = s->String::Chars[j];
    i++;
}

else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox30->TextLength;
if(size>0)
{
    s = textBox30->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox29->TextLength;
if(size>0)
{
    s = textBox29->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox28->TextLength;
if(size>0)
{
    s = textBox28->Text;
    for(int j = 0; j<size; j++)
    {
        cBuf[i] = s->String::Chars[j];
    }
}
```
i++;
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox27->TextLength;
if(size>0)
{
    s = textBox27->Text;
    for(int j = 0;j<size;j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = 0;
size = textBox26->TextLength;
if(size>0)
{
    s = textBox26->Text;
    for(int j = 0;j<size;j++)
    {
        cBuf[i] = s->String::Chars[j];
        i++;
    }
}
else
{
    cBuf[i] = '9';
    cBuf[i+1] = '.';
    cBuf[i+2] = '9';
    i+=3;
}

size = i;
}
else
{}
StartThread(); // Create and start the reader thread ftStatus = FT_SetEventNotification(handle, FT_EVENT_RXCHAR, hEvent); // Set the notification event ftStatus = FT_SetBaudRate(handle, 9600); // Set the baud rate, 9600 ftStatus = FT_Write(handle, cBuf, size, &ret); // Write into the USB controller
}
else {
    MessageBox::Show("Open Failed");
    return;
}

System::Threading::Thread::Sleep(1000);
WaitForSingleObject(DEvent, -1);

listBox1->Items->Clear(); // Clear the feedback display

// Update the display
for(int i = 1; i < 112; i += 3) {
    // Read the three byte standing for the voltage value
    test1 = buf[i];
    test2 = buf[i + 1];
    test3 = buf[i + 2];

    data1 = test1 - 48; // Read digit before the virgule
    data2 = (float)(test3 - 48) / 10; // Read digit after the virgule
    data3 = data1 + data2;

    s = Convert::ToString(data3); // Convert voltage to string type
    this->listBox1->Items->Add(s); // Add voltage value to the list box
    data1 = data2 = data3 = 0; // Reset data variable
}

System::Void FillComboBox(UInt32 dwDescFlags) {
    FT_STATUS ftStatus;
    UInt32 numDevs;
    void * p1;

    comboBox1->Items->Clear();

    p1 = (void*)&numDevs;
    ftStatus = FT_ListDevices(p1, NULL, FT_LIST_NUMBER_ONLY);

    if(ftStatus == 0) {
        char cBuf[64];
        if (ftStatus == 0) {
            UInt32 uDevNo;
            for(uDevNo = 0; uDevNo < numDevs; uDevNo++)
            {
                String * str;
                if(dwDescFlags == FT_LIST_NUMBER_ONLY) {
                    str = Convert::ToString(uDevNo);
                    
```
comboBox1->Items->Add(str);

else {
    ftStatus = FT_ListDevices(uDevNo, cBuf, dwDescFlags);
    str = Convert::ToString(cBuf);
    comboBox1->Items->Add(str);
}

if(comboBox1->Items->Count > 0) {
    comboBox1->SelectedIndex = 0;
}

System::Void StopThread()
{
    bContinue = false;
    if(hEvent) // let the thread come out of waiting for infinite time
        SetEvent(hEvent);
    if(thrdReader && thrdReader->IsAlive) { // stop it
        TimeSpan waitTime = TimeSpan(0, 0, 1); // 1 second timeout
        if(thrdReader->Join(waitTime) != true) {
            thrdReader->Abort(); // didnt stop the thread - take more drastic action
        }
    }
}

System::Void StartThread()
{
    // Create a reader thread here
    thrdReader = new Thread(new ThreadStart(&USBFT245R::Form1::ReadingProc));
    bContinue = true;
    thrdReader->Start();
    while (!thrdReader->IsAlive); // wait for the thread to start
    Thread::Sleep(1000);
}

System::Void Form1_Load(System::Object * sender, System::EventArgs * e)
{
    handle = NULL;
    hEvent = CreateEvent(NULL, FALSE, FALSE, "");
    radioDescription->Checked = true;
    radioDirect->Checked = true;
}

System::Void comboBox1_SelectedIndexChanged(System::Object * sender, System::EventArgs * e)
{
    ClosePort();
    OpenPort();
}

System::Void radioNumber_CheckedChanged(System::Object * sender, System::EventArgs * e)
{
    if(radioNumber->Checked)
    {
ClosePort();
dwOpenFlags = FT_LIST_NUMBER_ONLY;
FillComboBox(dwOpenFlags);
}

System::Void radioDescription_CheckedChanged(System::Object * sender, System::EventArgs * e)
{
    if(radioDescription->Checked) {
        ClosePort();
dwOpenFlags = FT_OPEN_BY_DESCRIPTION;
FillComboBox(FT_LIST_BY_INDEX | dwOpenFlags);
    }
}

System::Void radioSerial_CheckedChanged(System::Object * sender, System::EventArgs * e)
{
    if(radioSerial->Checked) {
        ClosePort();
dwOpenFlags = FT_OPEN_BY_SERIAL_NUMBER;
FillComboBox(FT_LIST_BY_INDEX | dwOpenFlags);
    }
}

System::Void radioDirect_CheckedChanged(System::Object* sender, System::EventArgs* e)
{
    if(radioDirect->Checked)
    {
        textBoxAngle->Enabled = true;
textBoxSameV->Enabled = false;
setVoltageTextBox(false);
    }
}

System::Void radioSameV_CheckedChanged(System::Object* sender, System::EventArgs* e)
{
    if(radioSameV->Checked)
    {
        textBoxAngle->Enabled = false;
textBoxSameV->Enabled = true;
setVoltageTextBox(false);
    }
}

System::Void radioDiffV_CheckedChanged(System::Object* sender, System::EventArgs* e)
{
    if(radioDiffV->Checked)
    {
        textBoxAngle->Enabled = false;
textBoxSameV->Enabled = false;
setVoltageTextBox(true);
    }
}