Presentation of Real-Time TFR-data

Anders Karlsson
2014

Thesis work in Computer Science, 15 credits
IngOnline

Examiner: Jonas Boustedt
Supervisor: Carina Pettersson
Presentation of Real-Time TFR-data

by

Anders Karlsson

Faculty of Engineering and Sustainable Development

University of Gävle

S-801 76 Gävle, Sweden

Email:

anders@akme.se

Abstract

In a high-voltage direct current system (HVDC), a process is continuously recording data (e.g., voltage sampling). To access this data, the operator must first set triggers, then wait for the transient-fault-recording (TFR) to complete, and finally open the recorded file for analysis. A more refined solution is required. It should allow the operator to select, add and watch the measuring points in real-time. Thus, the purpose of this thesis work is to read, process and present the samples continuously, as series in a graphical chart. Programming will be done by using iterative and incremental development. The result of this thesis work has been an executable application with a graphical user interface (GUI), able to showing the content in the buffers as a graph in a chart.

Keywords: ABB, HVDC, TFR, C#, .NET, Visual studio, real-time.
Preface

In the late autumn of 2013 I contacted Jonny Eriksson at ABB with the hope to do a thesis work in embedded systems. This resulted in that I visited him at the HVDC dept. in Ludvika a few weeks later. Now, six moth later, my thesis work is completed and it has been done at HVDC in Ludvika where Jonny has been my attendant. Thank you for believing in me and hope I proved you right.

I am extremely grateful that I got the opportunity to do this journey. I have met people from all around the world and it has made a big impression on me to experience this.

At ABB I have been entrusted to develop and evaluate the potential for real-time monitoring of TFR data. I have in doing so learned many new things and had the opportunity to work with possibly some of the world’s brightest minds.

I would also like to give special thanks to Lars Ungerdaahl who has been my sounding board and my unwritten mentor at ABB.

Last, but not least, I want to mention my family. I could not have done this without you.

Thanks!

Falun, May 26, 2014

Anders Karlsson
Table of Contents

Abstract ............................................................................................................................. iii
Preface ............................................................................................................................... iv
Terminology ........................................................................................................ vii

1 Introduction .................................................................................................................... 1

2 Background ...................................................................................................................... 1
  2.1 Problem justification ................................................................................................. 2
  2.2 Purpose ...................................................................................................................... 3
  2.3 Scope ......................................................................................................................... 3

3 Method .............................................................................................................................. 3
  3.1 Solution ....................................................................................................................... 3
  3.2 Environment ............................................................................................................... 3

4 Implementation ............................................................................................................... 4
  4.1 TFR buffers and structures ....................................................................................... 4
    4.1.1 The MAIN program and its objects ........................................................................ 5
    4.1.2 TFR structs ........................................................................................................... 5
      4.1.2.1 AnaInfo2 .......................................................................................................... 5
      4.1.2.2 AnaSamp2 ...................................................................................................... 5
      4.1.2.3 TFR2 ............................................................................................................... 5
      4.1.2.4 MainDS20 ...................................................................................................... 6
  4.2 The Syncfusion tool .................................................................................................... 6
    4.2.1 Configuring the chart ........................................................................................... 6
      4.2.1.1 Axis ............................................................................................................... 7
      4.2.1.2 Zoom functions .............................................................................................. 7
      4.2.1.3 Legend ............................................................................................................ 7
    4.2.2 Configuring series ................................................................................................. 7
      4.2.2.1 Adding series to the chart .............................................................................. 7
      4.2.2.2 Adding points to series ................................................................................. 8
    4.2.3 Presenting series in the chart ............................................................................... 8
  4.3 Presentation .................................................................................................................. 8
    4.3.1 A ToolStripMenu ............................................................................................... 9
      4.3.1.1 File ............................................................................................................... 9
      4.3.1.2 Status .......................................................................................................... 9
      4.3.1.3 Series ......................................................................................................... 9
      4.3.1.4 Refresh rate ................................................................................................. 9
      4.3.1.5 Buffer and Pre buffer .................................................................................. 9
    4.3.2 The chart ............................................................................................................... 9
    4.3.3 Startup ................................................................................................................ 10
    4.3.4 Adding the series .............................................................................................. 10
    4.3.5 Creating an instance .......................................................................................... 10
  4.4 Business Logic ............................................................................................................ 10
    4.4.1 Get TFR data .................................................................................................... 10
    4.4.2 Adding timestamp ............................................................................................ 11
    4.4.3 Finishing the class buffer ................................................................................ 11
  4.5 Data access ................................................................................................................ 12
    4.5.1 Accessing the data ........................................................................................... 12
    4.5.2 INtime access ................................................................................................. 12
    4.5.3 Marshaling ..................................................................................................... 13
Terminology

Buffer
A memory area to contain data that can be read from and written to.

Circular buffer
A fixed size buffer where new data is continuously written. When the writers pointer reaches the buffers end, it jumps to the beginning where it continues writing and the previous data will be overwritten.

Abbreviations and acronyms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HVDC</td>
<td>High-Voltage Direct Current</td>
</tr>
<tr>
<td>HVAC</td>
<td>High-Voltage Alternating Current</td>
</tr>
<tr>
<td>TFR</td>
<td>Transient Fault Recorder</td>
</tr>
</tbody>
</table>

Data structures [8][10]

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit</td>
<td>The smallest element of information, 0 or 1.</td>
</tr>
<tr>
<td>byte</td>
<td>Contains eight bits. Can represent $2^8 = 256$ values.</td>
</tr>
<tr>
<td>word</td>
<td>A word is two bytes, or 16 bits. A nword is 16n bits.</td>
</tr>
<tr>
<td>[D</td>
<td>Q]word</td>
</tr>
<tr>
<td>bool</td>
<td>A boolean value represents either true or false.</td>
</tr>
<tr>
<td>char</td>
<td>A char is one byte and represents a character, meaning a letter, number or a symbol.</td>
</tr>
<tr>
<td>string</td>
<td>A collection of Chars.</td>
</tr>
<tr>
<td>int</td>
<td>An integer is a non fractional number of 32 bits. Represents $2^{32} = 4\times10^9$ values, from $-2\times10^9$ to $+2\times10^9$.</td>
</tr>
<tr>
<td>uint</td>
<td>An unsigned integer, contains only positive values. From 0 to $4\times10^9$.</td>
</tr>
<tr>
<td>float</td>
<td>Is a 32 bit number that can represents fractional numbers.</td>
</tr>
<tr>
<td>double</td>
<td>Like float but consists of 64 bits, that improves the precision compared to float.</td>
</tr>
</tbody>
</table>

Data types in different languages [9][10]

<table>
<thead>
<tr>
<th>C++</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>byte</td>
</tr>
<tr>
<td>dword</td>
<td>uint</td>
</tr>
</tbody>
</table>
1 Introduction

In a high-voltage direct current system (HVDC) a process is continuously recording data. In the event of an error this data is flushed down to file for an engineer to analyze, like Figure 1. The transient-fault-recorder (TFR)-data is only accessible after the error occurred since that is when the data is committed to disc.

![Figure 1: A recorded TFR file opened for analyzing.](image1)

The scope for this thesis has been to implement an application with functionality for monitoring the TFR-data in real-time.

2 Background

Since dawn of mankind humans have been in search of more comfortable ways to live their lives. Our curiosity has make us explore our surroundings and its possibilities. We took control over fire, invented the wheel, extracting energy from the rivers flowing water, the blowing wind, the sun, and by cleaving nuclei. [1]

However, we rarely want to use the energy where it's produced, meaning we need to transport it from the power plants to our cities (see Figure 2). To do this efficiently we need to convert the energy to a form that can be transported with low loss in which electricity is just that. [2]

When transporting energy over long distance there is less loss using HVDC than high-voltage alternating current (HVAC). [3]
ABB in Ludvika, Sweden, develops equipment for HVDC that can transmit high power energy over long distance. Inside an HVDC station there are many stands with control equipment. Each stand has a main-computer, which is connected to different kinds of input/output (I/O)-units, where each stand has it's own hardware configuration.

The main-computer is running a Windows Embedded OS alongside a real-time OS called INtime [4]. Each TFR measuring point is configured in INtime, and has its own memory allocation. This memory is designed as a circular buffer to which the sampled data is written, typically 10 seconds of data sampled at 1kHz with a pre-fault of 5 seconds and a post-fault of 5 seconds.

### 2.1 Problem justification

For each measuring point data is recorded continuously to the circular buffer. A trigger is set to detect if an error occurs. As we can see in Figure 3, this is triggered and the data continues to write to the buffer until it reaches the pre-fault point, meaning that data before the trigger occurred is still stored in the buffer. Now the buffer will be flushed to disk and the engineers can analyze the data.

When the engineers are adjusting the system with changes and updates they want to see the effect of these changes. To do this they must manually set a trigger to create such TFR-file that can be analyzed.

![Diagram](image.png)

*Figure 3: Present approach to analyze TFR-data.*

This procedure is time consuming and cumbersome. That is why a real-time application for viewing the TFR-data is desired.

So, is it possible to build a solution that can present the TFR-data in real-time?
2.2 **Purpose**
The purpose of this thesis work will be to determine if it is possible to connect to the INtime-buffers and present the TFR-data in an application in real-time.

To do this we must get access to the buffers in INtime, read chosen buffers, process the data, and send the result to the output stream.

2.3 **Scope**
In this thesis work the result will be limited to connect to one (1) buffer in INtime, process the data, and display it on screen as a graph in a chart, presented in an executable application.

3 **Method**
The task of this thesis work will be to create an application that can connect to the TFR-buffers and present the data as series in a graph.

Programming will be done by using iterative and incremental development [7], meaning that while creating the parts of the program, writing and testing will be done parallel during the process. The benefits of this is that errors is detected early and can be corrected early in the process.

3.1 **Solution**
The application will be divided into three distinct layers, each with their specific task, *Presentation*, *Business logic* and *Data access*. [11]

3.2 **Environment**
The development will be done using Microsofts Visual Studio 2010 in the programming language C# (see sharp) under the Windows .NET-environment. Since the main-computer is running Windows, the purpose will be to create an executable application for viewing the TFR-data under the Windows OS.

ABB has bought a .NET component from *Syncfusion* that can present the data in series in a chart, which easily interacts with the user who can zoom an scroll within the chart. This component will be used in the presentation layer. [5]
4 Implementation
First we need to explore the TFR structures in INtime and get information about the structures. A struct describes the content of the data in the buffers.

Then we need to learn the Syncfusion functions and test its functionality to know how to create, insert and update charts and series.

The program will be divided into three layers (see Figure 4), Presentation, Business Logic and Data access, where each layer has its own interface and can individually be upgraded or even replaced without doing anything to the surrounding layers. To maintain this structure, the Presentation layer will always communicate with the Business Logic layer, and must never talk directly to Data access.

- Graphical User Interface (GUI)
  - Interacting with the user
  - A List (collection) contains all added series
  - Each serie contains an instance of Business Logic
  - Contains a thread which updates the chart periodically and iterates through the List.

- Prepare data
  - Get data from Data Access
  - Compute buffer contents
  - Add time stamp
  - Merge data to instance buffer

- Collect data
  - Connect to INtime
  - Read data in the buffers

Figure 4: Application architectural overview.

4.1 TFR buffers and structures
In INtime there is an explorer tool (see Figure 5) that opens a window, showing ongoing processes in a tree view. Each process has its own threads and objects and can be viewed by expanding the dedicated tree view for the chosen object.

Figure 5: Screen dump showing the INtime-explorer.
For this thesis we have been assigned an own process called *MAIN*. How the process is created and installed under INtime is beyond the scope of this thesis.

### 4.1.1 The MAIN program and its objects

When expanding the *MAIN* process it reveals a number of objects. The ones we are interested in are named **AnaInfo2**, **AnaSamp21**, **AnaSamp22**, **DigInfo2**, **DigSamp21**, **DigSamp22**, **TFR2** and **MainDS20**.

By clicking on any of those we can see the current content in them, the raw data, where each value is a *DWORD* and that can represent anything from bool values to numbers to strings. To be able to understand the contents we need the *structs* for each objects.

### 4.1.2 TFR structs

There are four structs that describes the content of the objects in the *MAIN* process. Just four? Yes, the *AnaSamp2* only consists float-numbers, and *DigSamp2* only bool-values. We are now able to decode the contents in the objects.

A simple program was created that reads data from the buffers in INtime and presents the selected objects content in a list. This program later became the basis of the *Data Access layer* described on page 12.

Because of the scope of this thesis, we omit the digital sections.

#### 4.1.2.1 AnaInfo2

This object contains information about the analog sample objects such as id and name. There are twelve channels available but only four are in use.

#### 4.1.2.2 AnaSamp2

The two ring buffers, *AnaSamp21* and *AnaSamp22*, each contains 484'800 bytes. This data consists of 12 channels and each channel contains 10'100 samples each of float numbers.

\[
\text{channels} \times \text{bufferSize} \times \text{sizeOf(float)} = 12 \times 10\,100 \times 4 = 484\,400
\]

#### 4.1.2.3 TFR2

Here we can find information about the sample time, the buffer size to which buffer samples are written, the pointer to where in the buffer the data is written, and much more. What we need to know to be able to get data from the buffers is:

- **CurrentSamplePeriodTime** Length of each samples in *µs*
- **SampleIndex** Buffer size
- **RecordIndex** Writing pointer

Beyond the scope of this thesis work, but worth mentioning, is that data is recorded to one (1) buffer at the time, either *AnaSamp21* or *AnaSamp22*. This is controlled by a variable in the *TFR2* structure. If a
trigger occur, the RecordIndex will continue writing data to the present buffer until it reaches a pre-/post fault-samples variable. The RecordIndex will then switch to the next buffer and the previous buffer can later be written to disk without interfering with the writing of new real-time data. This will also affect the DigSamp21 and DigSamp22, and vice versa.

4.1.2.4 MainDS20
In the MainDS20 object we find information about the number of analog and digital channels, and which the start channel is for each of the objects.

4.2 The Syncfusion tool
When creating a window form application in visual studio, there is a toolbox (see Figure 6) with a few functions that can be added to the empty form by drag’n’drop. In the toolbox, under its own section, Syncfusion has several functions that can be added to the windows form.

![Figure 6: The toolbox showing Syncfusion components.](image)

4.2.1 Configuring the chart
When the chart, default named chartControl1, is added to the form, there are possibilities to change the layout and style (see Figure 7).

![Figure 7: Properties with easy editing for the chartControl1.](image)
We want a white background with black text, and we also want to change the header name to “Real-Time TFR Data”.

```csharp
ChartControl chartControl1 = new ChartControl();
chartControl1.Text = "Real-Time TFR Data";
```

### 4.2.1 Axis

When configuring the axis, we use DateTime format for the X-axis named “Time”, while the Y-axis is a double, which is default, named “Voltage”.

```csharp
chartControl1.PrimaryXAxis.DateTimeFormat = "HH:mm:ss";
chartControl1.PrimaryXAxis.Title = "Time";
chartControl1.PrimaryXAxis.ValueType = Syncfusion.Windows.Forms.Chart.ChartValueType.DateTime;
chartControl1.PrimaryYAxis.Title = "Voltage";
```

### 4.2.1.2 Zoom functions

When zoom is enabled the user can simply click on the chart and select a specific area on it with the mouse pointer. When the mouse key is released, Syncfusion will zoom into that area.

There are also key functions that can be enabled and then the user can zoom in and out using the [+]- and [-]- keys on the keyboard. The [Esc] key will return to 1:1 scale. While zoomed in, the arrow keys can be used to navigate in the window.

```csharp
chartControl1.EnableXZooming = true;
chartControl1.EnableYZooming = true;
chartControl1.KeyZoom = true;
```

### 4.2.1.3 Legend

When several series are added to the chart they will be shown with different colors. To keep track of each serie, the color can be described with a name in a legend.

```csharp
chartControl1.Legend = true;
```

### 4.2.2 Configuring series

In Syncfusion, a serie can be configured by choosing line width, line type that can be a smooth or a straight line, adding a shadow, viewing in 3D-style, etc. Since we will plot several hundred of points per series, we choose to minimize the complexity and use as little effects as possible.

```csharp
ChartSeries serie = new ChartSeries();
serie.Style.Border.Width = 2F;
serie.Style.DisplayShadow = false;
serie.Style.DrawTextShape = false;
serie.Text = "Serie1";
serie.Type = ChartSeriesType.Line;
```

#### 4.2.2.1 Adding series to the chart

When the series is configured, we use the `chartControl1` and add the `serie` to it.

```csharp
chartControl1.Series.Add(serie);
```
4.2.2.2 Adding points to series
Series are simply a line drawn between a collection of points, where a point consists of an X and a Y-value.

We use the `series` and add the points to it, shown in Figure 8.

\[
\text{serie.Points.Add}(x_0, y_0); \\
\text{serie.Points.Add}(x_1, y_1); \\
\text{serie.Points.Add}(x_2, y_2); \\
\ldots
\]

4.2.3 Presenting series in the chart
By keeping all series in a list we can add them to the chart by using a `foreach` loop. The function `UpdateSerie()` refers to `Adding points to series` in section 4.2.2.2 above.

\[
\text{foreach (Serie s in listOfSerie)} \\
\quad \{ \\
\quad \quad \text{s.UpdateSerie();} \\
\quad \}
\]

When all series have been updated they are automatically updated in the chart. To show only a specific section on the chart, we can add a `min`- and a `max`-value to the charts X-axis.

\[
\text{chartControl1.PrimaryXAxis.Range.Min} = \text{DateTime.Now.AddSeconds(-2).ToOADate();} \\
\text{chartControl1.PrimaryXAxis.Range.Max} = \text{DateTime.Now.ToOADate();}
\]

This would show a section over the two past seconds from now regardless to the time span in the added series.

4.3 Presentation
The presentation layer is where the program interacts with the user. Besides presenting the results, the user can configure the program, add series, zoom in the chart, etc.

What we need to be able to control the application is to `start/stop` the update, add series to the chart, change the size of the presented data (2 - 30 seconds), update rate (0.1 – 2.0 seconds), and exit the application.

Zooming and controlling the chart is already implemented via Syncfusion.
4.3.1 **A ToolStripMenu**
Control functions can be added to the menubar at the top of the form where they don't distract the layout while still can contain all necessary functions.

The functions for the program is

- **File**
  - Exit
- **Status**: Running/Stopped
  - Start/Stop
- **Series**
  - Add
    - [Available objects in *AnaInfo2*]
- **Refresh rate**
  - [Refresh rate in milliseconds 100 – 2000]
- **Buffer**
  - Pre buffer
    - [0 – 4 seconds]
  - [2 – 30 seconds]

4.3.1.1 **File**
Exit [Alt+F4] is closing the application.

4.3.1.2 **Status**
The status is showing if the chart is in *running* or *stopped* mode.
Start/stop [Space bar] will toggle status mode.

4.3.1.3 **Series**
The series menu will dynamically load its content at startup, and data will be loaded from the TFR-objects.

4.3.1.4 **Refresh rate**
This defines how often the chart will be updated. These values are stored in an array and the menu is dynamically updated according to the content in the array.

4.3.1.5 **Buffer and Pre buffer**
The buffer will determine how much of the series data will be viewed in the chart. The pre buffer lets the data to be collected and aggregated before viewed.

Viewed chart = from *(Buffer + Pre buffer)* to *(Pre buffer)*

Values are contained in arrays and menu will dynamically update.

4.3.2 **The chart**
When the application starts, the chart is initiated but contains no series.
4.3.3 **Startup**
At startup, the presentation layer will communicate down in the chain to **Business Logic** to get information about available TFR buffers, and present them in the menu under Series. At the same time, a thread is created to update the chart automatically using the Refresh rate value.

4.3.4 **Adding the series**
When the user adds one of the series to the chart, an instance of that series will be added to a list. This list contains each and one of the series. The list will automatically iterate by the **Refresh rate thread** resulting in that each series in the chart will be updated.

4.3.5 **Creating an instance**
In the constructor of the series, all parameters that is needed for this series is given its initial values, such as **id** and **name**. The serie then creates an instance of **Business Logic** which purpose is to serve this series with data.

4.4 **Business Logic**
The Business Logic layer will be an instance of each series. Its purpose is to get TFR data from the **Data access layer**, add new data to the **Business Logic**’s instance buffer, which is bigger than the TFR ring buffer. It will also add timestamps to each value, and prepare the data buffer to be sent to the **Presentation layer**.

When a new instance is initialized, a thread is created that will perform this update repeatedly throughout the life of the instance. Default update time is set to 200 ms.

4.4.1 **Get TFR data**
In the TFR structure there is a pointer that keeps track of where new data is written, meaning that the data before that pointer is old and probably already read. We need to know from what point we last read data, and from where new, unread, data is written. Then we can collect the data between those two points (see Figure 9).

If the write pointer is less than our last read pointer, it means that the end of the circular buffer was reached and that data was continued to be written at the beginning of the buffer.

**Figure 9:** New data exists between the Read- to the Write pointer.

**Figure 10:** The new unread data continues from the buffer end to its beginning.
So, in this case we read to the end of the buffer and then we reset our reading counter to zero and we can continue reading until we reach the write pointer (see Figure 10).

```c
if (readingIndex >= bufferSize) {
    readingIndex = 0;
}
```

Now we can add these new data to our local buffer.

### 4.4.2 Adding timestamp

To be able to add the data to the series we need to add a timestamp to each data value. This is done by getting current time \((\text{now})\), that represents the last read value (now) in the buffer, then we calculate a timestamp for each one of the previous data values until the first value is calculated (see Figure 11).

In the TFR structure there is a value that represents how often a sample is captured per microsecond \((\mu s)\). We also know the buffer size and we can calculate the time between all samples.

- \(\text{currentTime}\) — timestamp when last sample was stored.
- \(\text{firstSample}\) — timestamp for the first sample.
- \(\text{timePerSample}\) — How long each sample is in milliseconds.
- \(\text{samplesPerMsec}\) — Samples per millisecond.
- \(\text{bufferSize}\) — Number of samples in the buffer.

\[
\text{timePerMsec} = \frac{\text{samplesCapturedEachMilliSecond}}{1000}
\]

\[
\text{timePerSample} = \frac{\text{bufferSize}}{\text{samplesPerMsec}}
\]

\[
\text{firstSample} = \text{currentTime} - \text{bufferSize} \cdot \text{timePerSample}
\]

By iterating through the buffer we can add date time values to each sample.

### 4.4.3 Finishing the class buffer

Since we now have new data in a temporary buffer, we can merge that data onto our instance buffer. If the instance buffer is greater than the allowed maximum buffer size, we delete the first values from the buffer until the buffer size is within limits (see Figure 12).
```csharp
if (bufferData.Count > MAX_BUFFER_SIZE)
{
    int previous = 0;
    int start = bufferData.Count - MAX_BUFFER_SIZE;
    bufferData.RemoveRange(previous, start);
}
```

4.5 Data access

When each instance of Business Logic wants to access the TFR data, the Data access layer handles the communication with INtime. Since this is a static class, no instance is needed and all public functions can be reached directly by just calling `DataAccess`.

```csharp
bufferSize = DataAccess.GetBufferSize();
```

4.5.1 Accessing the data

The current configuration consists of 12 analog buffers and when accessing a specific channel from the TFR structure we need to read every twelfth data, starting with the index for the chosen channel.

To do this we take arguments for this function.

- `ref Buffer` lets us write into the callers buffer.
- `bufferSize` is the size of the TFR buffer in INtime.
- `numberOfChannels` is the number of channels in current TFR buffer.
- `channelIndex` selects the channel we want to read.

With these arguments, the data Access can insert the current TFR-channel's samples directly into the Business Logic’s buffer.

```csharp
DataAccess.GetTfrAnalogData(ref tfrBuffer, 
bufferSize, 
numberOfChannels, 
channelIndex)
```

4.5.2 INtime access

The Access Data layer connects to INtime through one of ABB's specially written interfaces which is implemented as a DLL-file.

```csharp
using ABB.NtxWrapper;
```

We can use `NtxOpenSharedMemory` which takes the arguments to the

---

1 A reference in C# is similar to a pointer in C/C++.
INtime process and its object. If opened successfully the buffer will contain a handle to the data.

    handle.NtxOpenSharedMemory(“local”, “MAIN”, “TFR2”)

To read the data we use Read which takes a buffer as a reference to store the read data, and the size of the dataStructure we want to read.

    int dataLength = Marshal.SizeOf(typeof([dataStructure]));
    byte[] readData = new byte[dataLength];
    handle.Read(ref readData, (uint)dataLength)

Both functions return a boolean value to confirm whether the connection and reading succeeded (true) or not (false).

4.5.3 Marshaling
Since the TFR structures in INtime is written in C++, which is unmanaged, and read in C#, which is managed, we need to convert the structures between those.[6][12]

To do this we use Microsofts InteropServices that provides a number of functions for converting data between managed and unmanaged structures.

4.5.3.1 Struct in C++
Here is the structure of the TfrAnalogInfo in the C++ language. CHAR_DEF is a macro that creates a fixed size array from the args.

    __BEGINSTRUCT( TFRANALOGINFO )
    DWORD m_ChannelNumber;
    CHAR_DEF( m_ChannelName, 40 )
    float m_DefaultFactor_A;
    int m_DefaultMinValue;
    __ENDSTRUCT( TFRANALOGINFO )

4.5.3.2 Struct in C#
First we need to specify the size each data segment in bytes. As we’ve seen in 4.1.1, all data is stored in DWORDs which is equal to 4 bytes.

Since there are fixed arrays instead of strings in C/C++, the strings are stated before variables are defined by using the MarshalAs function where we can define the string length.

After transforming the C++ struct to C# it looks like this.

    [StructLayout(LayoutKind.Sequential, Pack = 4)]
    public struct TfrAnalogInfo
    {
        public uint m_ChannelNumber;
        [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 40)]
        public string m_ChannelName;
        public float m_DefaultFactor_A;
        public int m_DefaultMinValue;
    }
5 **Result**

The result of this thesis work is an executable application with a graphical user interface (GUI), able of showing the content in the AnaSamp21 as a graph in a chart (see Figure 13).

![Figure 13: The final Real-Time TFR data application, showing the oscillating MS2 as a sine wave.](image)

5.1 **HiDraw**

This is outside the scope of this thesis work, but to get an understanding of how the TFR-buffers are created and configured I want to show the HiDraw application (see Figure 14).

HiDraw is a graphical code generating application where input points can be added to output units through filters and gates, etc., connected together with virtual wires, drawn with the mouse.

![Figure 14: The HiDraw worksheet showing the creation of our TFR generator.](image)

The analog channel 1, named as MS1, is attached to a counter that counts from 0 to 100'000 and then resets back to zero again. A three phase sine generator of 50 Hz is connected to channels MS2-4, as we can see in Figure 13 for the MS2 series.

From the visual worksheet, code will be generated which will be inserted in INtime.

- 14 -
5.2 Connect to INtime

The processes running in INtime can be viewed in an explorer window (see Figure 15) and each process can be collapsed to show its objects. By clicking on any object, the right screen will show information and a preview of the data in the objects buffer.

5.2.1 Show raw buffer data

By performing a right-click on any object in the INtime Explorer, a list of options will appear. By clicking *dump* at the bottom of the list a window will appear (see Figure 16) which shows a snap-shot of the content in the chosen buffer.

![Image](image1.png)

*Figure 15: INtime Explorer view, where the MAIN-process has been collapsed and the object AnaInfo2 is selected.*

![Image](image2.png)

*Figure 16: A snap-shot if the content in the AnaSamp21 buffer is shown as Dwords. The twelve channels can be seen, of which the first four channels contains TFR-data.*
5.2.2 Extracting the data

Initially we made a test program to connect to INtime and read the buffers of the process MAIN’s objects by hardcoding the buffer names (see Figure 15-21).

**TFR2**

The TFR2 buffer is the most comprehensive buffer (see Figure 19), and since we don't need all variables for the scope of this thesis work, we omit those unnecessary variables.

Required variables are:

- The **CurrentSamplePeriodTime** is showing that a new sample is written every 1’000 µs.
- At **SampleIndex** we can see that the size of each buffer is 10’101, but in the description in HiDraw, the size is 10’000...
- The **RecordIndex** is the writing pointer and counts up to **SampleIndex** and then resets to 0.

Figure 17: The menu containing the hardcoded object names.

Figure 18: The content of MainDS20 showing 12 analog channels.

Figure 19: Some of the content in the TFR2 buffer, though not all data is shown.
5.3 The menu

For the final application we put all the controllers in the menu of the form as seen in Figure 22.

The menu is applied by drag’n’drop in visual studio and then edited to enable dynamic updates with variable values e.g. the MAIN’s objects (see Figure 23-27).

Figure 20: AnalInfo2 contains information, like id and name of each channel.

Figure 21: The first values in AnaSamp21. This is showing the first channel MS1.

Figure 22: The menu is visible above the chart.

Figure 23: Open and Save are disabled, but Exit works.

Figure 24: Showing status, waiting for series. Start/stop can be clicked.

Figure 25: Dynamically updated from the AnalInfo2.
5.4 View the MS1

In the menu, when clicking on Series, Add and MS1, the chosen series will be added to the chart. Status will be changed to Running and as shown in Figure 28 we will see the name of the added series as a Legend in the charts top right corner.

![Image](image.png)

**Figure 28:** In the application we can see 2 seconds of the series MS1 running. The counter reached 100,000 and restarted at 0, continued counting.
5.5 Viewing the MS2, MS3 and MS4

If we *Add* the MS2 series, the chart will show an oscillating sine wave as shown in Figure 13.

Now we can add another series, MS3. We can see that the previous series is checked and those cannot be added a second time (see Figure 29).

By adding the *MS4* series we can, in the chart, view the oscillating three phase sine waves as created in the HiDraw application (see Figure 30).

![Figure 30: Two seconds of MS2-4 series shown in the chart.](image)

5.6 Zooming in the chart

As we can see in Figure 30 above, the two seconds is quite a long time and it's difficult to see any details.

If we place the mouse pointer, click and drag, we can see a rectangle over the series (see Figure 31).

![Figure 31: Using the mouse to zoom in to the selected range, shown as a blue rectangle.](image)

When releasing the mouse button the chart will zoom in to the selected area (see Figure 32).
By pressing the spacebar, or clicking the Status's Start/Stop, we can pause the chart from being continuously updated.

In the zoomed view we can scroll in the chart by either using the mouse wheel, arrow buttons, or drag the windows scrollbars.

To zoom out we simply use the [Esc]-button.
6 Discussion
Large organizations like ABB are very complex and to understand the systems you need to know who to ask. During the first five weeks on this theses work a lot of time and effort was spent on understanding who to ask and when.

6.1 My journey at ABB
In the first week, after I got equipment and access to servers and systems, I started to familiarize with the task and was able to begin to set goals for my next seven weeks at ABB.

6.1.1 Exploring Visual Studio and Syncfusion
Since my knowledge about C# was very limited I started with some small tests to get to know the language and the concept of .NET and the environment of Visual Studio 2010.

The Syncfusion tool was installed by ABB at their Team Foundation Server (TFS), which I didn't have access to, so I downloaded and installed a 30-day trial version to be able to test it. I was soon able to create a real-time chart and configuring the layout and its functions.

6.1.2 Exploring end extracting data from INtime
After some weeks I got a MAIN-computer with the MAIN application installed in INtime for laboration purpose. It seemed to be more complicated to connect to INtime than I first predicted, and during this first week very little progress was made.

In the middle of the following week I was introduced to ABB's already existing, own developed tool for communicating with INtime. Now my work accelerated and I began building a testapplication for extracting data from INtime.

6.1.3 Managed and unmanaged code
To be able to decode the data in the buffers I needed to understand the structures. I suddenly faced a completely new problem when the INtime was running unmanaged code in C++, and the .NET ran managed code in C#. How to convert the structs between them?

One of the co-workers at ABB had a solution which solved the challenges very easy, and I was now able to read the content from INtime into my test application.

6.1.4 Transferring the test application to Syncfusion
Now, I was ready to implement my newfound knowledge of INtime communication to my previous real-time chart in Syncfusion. This did not work as I planned. The application hang, and though I implemented Try-Cache to almost every function, it wouldn't work. In retrospect, I think it had to to with the multiple threading in partly the refresh rate and partly the Syncfusion series update.

However, I created a new, blank solution in Visual Studio and started to build it up, part by part from the previous programs. The
result of this work is already presented under the title Result above.

6.2 **The purpose**
The purpose of this thesis work was to clarify opportunities, problems and issues of developing a real-time TFR data viewer.

6.2.1 **Opportunities**
To be able to select, add, and watch the measuring points in real-time is obviously a much more refined way of viewing the TFR without having to set triggers, wait for TFR recording to complete, and finally open the file for analyzing.

6.2.2 **View and performance**
The sample rate in our test was set to 1kHz, meaning that 2 seconds of data contains 2'000 points per series. If we wanted to see ten seconds, the 10'000 points significantly affected the charts update rate and the application became sluggish and slow. Imagine the performance when two or more series are added to the chart...

By setting unused chart functions explicitly to false, the application slightly gained performance.

On the other hand, when viewing more than two seconds of data, the 50 Hz sine wave will become a thick line on the screen, and no details will become viewable.

6.2.3 **Solution**
By viewing one second at a time, updated every second, would both give a good overview and great performance. But without any overhead, the user would easily miss seeing if an error occurred. To deal with this, one way could be to implement triggers that can alert the user in the event of an occur.

6.3 **Known errors**
There have occurred some errors that are still unresolved issues.

6.3.1 **Index out of range**
When running the application it sometimes happens that the program suddenly hangs, with at least one series added to the chart. A red cross replaces the chart, as we can see in Figure 33, and an error message is informing that an Index was out of range.

![Figure 33: Error screen showing a red cross and message icon.](image-url)
Since this doesn’t occur regularly I haven’t been able to solve this issue.

6.3.2 **Windows time inaccuracy**

Currently the timestamp is retrieved from the windows side, and not from INtime. Since the precision at 1kHz is equal to one millisecond it has come to our attention that the windows time is too inaccurate to be used for this purpose.

This results, due to this, that series can on one side be cascading, and on the other side consist a time gap (see Figure 34), due to merging different downloads at different times, with an inaccurate timestamp.

![Figure 34: A time gap in the series, due to Windows inaccurate time.](image)

To correct this I suggest that an implementation of the available atom-clock in INtime could be sent to serve a buffer in a process with an accurate timestamp that the application could use.
7  Conclusion
This thesis work has shown that it is possible to connect to the buffers in INtime and present the TFR-data to the output-stream for transparent display as series in a chart in real-time.

7.1  Future improvements
While I've been working on this project I have in the meantime found some issues that should be adjusted, as well as thoughts on how it might look and operate in the future.

7.1.1  Size and copying of data buffers
In the current code, the whole Business Logics buffer is transferred to the series at each update, containing 35'000 samples, and that is regardless of the timespan to be shown in the chart. This could be changed to only copying the data to be viewed, and that should lead to improved performance.

7.1.2  Hardcoded names to be dynamically updating
At the moment, the INtime process-names are hardcoded in the application. This should of course be a dynamic solution where the program itself locates available processes and objects.

7.1.3  Monitoring over network
It is possible to monitor several real-time parameters for viewing simultaneously. As seen in the Result, monitoring can be done directly on the main-computer, but it is more likely to implement only the data access layer on the main-computer and send the data over network for remote monitoring.