FFTC SDK User Manual

Project	PowerFFT SDK Enhancement
Title	FFTC SDK User Manual
Reference	
Client Reference	
Author(s)	Nicholas Kopp
Date	March 15, 2012

REVISION HISTORY

Date	Changes Made	Issue	Initials
August 06, 2009	First draft	1.0	Nko
December 09, 2010	Reflect changes made for NLR	2.0	Nko
December 22, 2010	Further update.	2.1	Nko
May 09, 2011	Add links to on-line docs	2.2	Nko
March 15, 2012	Update contact info, minor changes	2.3	Nko

Hybrid DSP Systems
Koningin Emmalaan 3
2635HH Den Hoorn ZH
Netherlands

Tel: +31 (0) 15 8700817

Email: info@hybriddsp.nl
Web: www.hybriddsp.nl
Web: http://www.hybriddsp.com/Products/ESAFFTCSDK.aspx

References

[REF 1] FTAB Architectural design document FFTC-ASD-ADD-001
[REF 2] FFTC API Requirements Document – FFTC-ASD-RS-002
[REF 3] SoW PowerFFT SDK Enhancement TEC-EDP/2008.22/MS Issue 1, Rev. 3

1 Background

The FFTC is a space qualified processor for high performance, low power Fourier transform processing. The FFTC SDK is a software development kit (SDK) for the FFTC and the standard control FPGA firmware and it is described in this guide. It is intended for developers who are working with products based on this processor such as the FTAB from Astrium or the OPDP from NLR. An understanding of the basic FFTC architecture is a requirement.

The following resources may be useful when working with the SDK.

Resource	Description	Location
FFTC / FTAB SDK API Documentation (also included in installer)	Detailed API reference.	*http://www.hybriddsp.nl/esa/doc150312/ftabsde.chm*
Email address for questions / support		support@hybriddsp.nl

2 Introduction

2.1 Overview

The SDK is a set of programmers' libraries and applications for assisting in the development of algorithms for products based on the ESA FFTC space qualified processor. Currently the FTAB board developed by Astrium and the OPDP board from NLR are supported but the base classes can be extended for other products built around the FFTC.

The libraries make algorithm development and testing faster and easier. The learning curve for new users is reduced. In addition to the libraries there are graphical based applications (GUIs), also built on top of the libraries. The GUIs permit easy demonstration of the FFTC and simple testing of algorithms in a visual manner.

2.2 Benefits

Some of the benefits of the SDK are described below:

- **Easy to use** – The FFTC libraries are written for the Microsoft .NET framework. This is an easy to use run-time that is standard with Windows Vista and 7 and a free download for Windows XP. .NET allows developers to use a wide range of languages including C++, Visual Basic and C#.
- **Flexible** – By use of Mono the same binaries can be used also under Linux.
- **Powerful** – By using the libraries from an established framework and language, highly complex applications can be developed that integrate compile and run-time sections into one application.
- **Low cost** – Compilers and software development environments are available free of charge (e.g. Visual Studio Express and Sharp Develop).
- **Encourages development –** The SDK includes a simulator and performance profiler that allows algorithms to be run and benchmarked without FFTC based hardware.

2.3 Features

A software development kit or SDK is an often used phrase; however its meaning is broad, ranging from simply installing a set of documents to providing a complete graphical based user environment. The ESA FFTC SDK comprises the following:

- **Algorithm Library** – A programmers library (DLL) consisting of a set of classes for describing an algorithm and ultimately producing it in a form that can be used with the hardware.

- **FFTC Simulator Library** – A programmers library (DLL) consisting of a simulator and performance profiler for the FFTC chip and control firmware.

Generally the developer will not need to access this library directly, rather access is via the target board simulator.

- **FFTC Utils Library** – A DLL featuring a number of utilities that may be useful to developers. These include the specific data formats used by the board and various data generators for testing purposes.

2.4 Installing the SDK

The SDK comes with a Microsoft installer. Double click the installer and follow the steps. The software is installed under Program Files/ESA/FFTC.

2.4.1 Third Party Libraries

FFTW library (libfftw3f-3.dll) must be either in the executing directory, in the system32 directory or on search path. It is included by default with the installer.

If working with an actual board then the appropriate drivers, software and hardware **must** be installed on the host computer.

If you wish to compile the unit test projects then NUnit must be installed. This is a free unit testing framework. Ensure that these projects reference nunit.framework.dll.

2.4.2 Native Libraries

It is necessary to copy the native library esafftcsimwin32.dll to either the directory you are running the executable from, the Windows System32 directory or somewhere else on the search path.

2.4.3 32-bit and 64-bit

By default the target is 32-bit since this is compatible with both 32-bit and 64-bit OS. However you wish to target 64-bit because your application requires this then remember you will need 64-bit versions of esafftcsimwin32.dll and libfftw3f-3.dll. Failure to use these will result in a bad image format exception. Another advantage 64-bit is the performance of the simulator which can improve by up to 20%.

3 Algorithm Library

The heart of the SDK is the algorithm library. The base classes are contained within ESA.Dataflow.dll and ESA.FFTC.dll. The latter contains the classes required to create valid instructions for the FFTC chip and standard control firmware.

3.1 Background

A board based on the FFTC such as the FTAB receives control vectors to instruct it as to what to do. The NLR OPDP receives macro images comprising of loops and one or more control vectors. Both control vectors and macro images derive from program images. Seen from the hardware control vectors and maco images are simply binary instructions comprising of one or more commands. A command describes a single data flow operation such as transferring data from the input (P0) to the first memory bank (P1).

This library is designed to be non-specific to the boards and instead specific to the FFTC processor and the standard control firmware. This firmware makes full use of the FFTC and provides additional features such as memory loop controllers.

The algorithm library is comprised of multiple classes that when used together can produce control vectors. The main class – **FFTCModule** - describes the standard FFTC architecture – essentially 7 controllers. These 7 are:

1. Input – P0
2. Memory bank 1 – P1
3. Memory bank 2 – P2
4. Memory bank 3 – P3
5. Memory bank 4 – P4
6. Output – P5
7. FFT Core – P6

The input controller is a data source. The output is a data sink. The memories can be either, dependent on whether they are being written to or read from. The FFT core is a process.

By use of additional classes such as **Block**, **Command** and **ControlVector** we can easily build control vectors by use of any .NET language. We write a program or script that upon execution creates a control vector in binary, xml or object form.

Blocks can essentially be thought of as vector variables. However unlike in a regular computer program we must say where the data it represents will be located. This can be on any sink or source controller (therefore not the FFT core which is a process). Blocks are therefore a convenient way of describing data.

Commands as stated earlier represent an operation such as transferring data from P0 to P1. Command objects are then added to ControlVector objects. Once

complete the control vector can be serialized to file as binary or xml or used later in the application as an object.

In C# a very simple example is shown below:

```
// Create an instance of FFTCModule
FFTCModule fftc = new FFTCModule();

// Create Blocks on the desired controllers
FFTCBlock bP0 = fftc.P0.CreateBlock(32, 64);
FFTCBlock bP1 = fftc.P1.CreateBlock(32, 64, 256, 128);

// Perform an operation on the Blocks resulting in a Command
Command c = bP1.Write(bP0);

// Add Command to Control Vector
ControlVector cv = new ControlVector();
cv.Add(c);

// Write to binary file
cv.Save("myCV.bin");
```

In this application we begin by creating an instance of the FFTCModule called fftc. This object contains a representation of the FFTC control firmware. Accessible as properties are the seven main controllers P0-P6. We can create a block by accessing these controllers on the fftc object then using the CreateBlock method. The type of block returned from this method is dependent on the type of controller, however all blocks inherit from the base block type 'Block' and for an FFTC module all blocks inherit from FFTCBlock. In this case we have made a block on P0 with width 32 samples and height 64 samples. On P1 we create a block of the same dimensions and also specify the location as x = 256 and y = 128.

With a block declared on P0 and a block on P1 a simple operation can be performed that will result in a Command for transferring data. Calling the Write method on the destination block with the source block as the argument results in this command. The Write method of the block is actually a short hand for FFTCModule.Write(Block, Block). Every Block created has a pointer to its parent Controller and every Controller has a pointer to its parent FFTCModule. More complex functions such as an FFT require calling methods of the FFTCModule directly.

A ControlVector is then instantiated and the Add method adds the Command. Finally we decide to create a binary file from the control vector by calling the Write method.

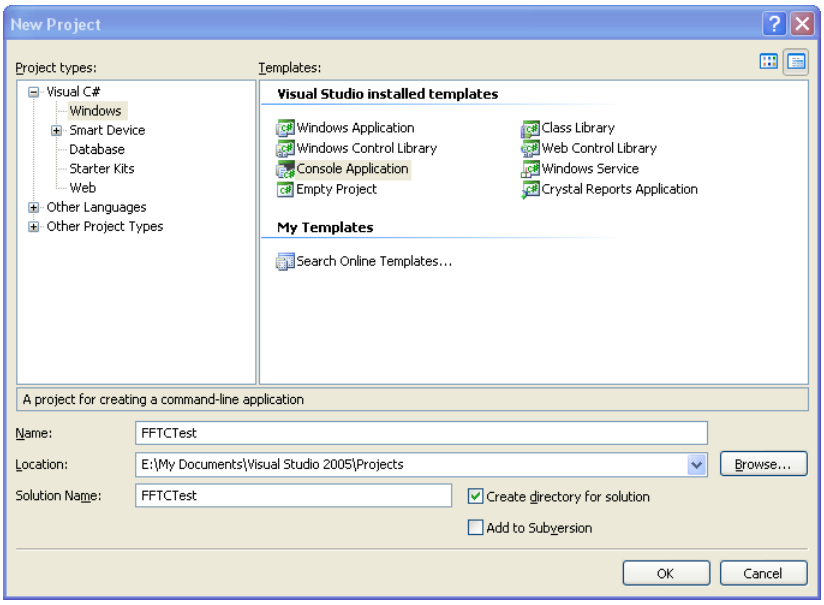
For detailed documentation see **TBD** *[http://www.hybriddsp.nl/esa/doc280609/]*

3.2 Using Visual Studio 2010

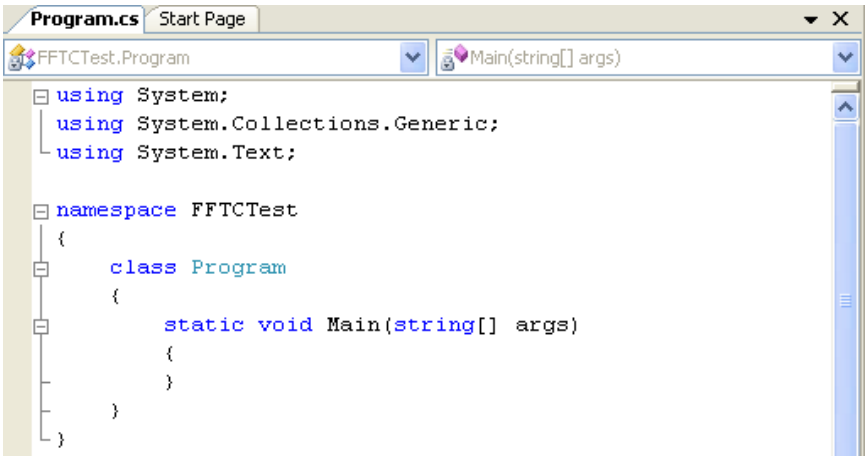
The SDK libraries have been compiled for Microsoft .NET 3.5 and 32-bit. To use the libraries Visual Studio 2010 is the SDE of preference. If these are not available then the free Express version or Sharp Develop will also suffice. Below is a simple walk

through for getting started with a C# project (note that any .NET language can be used).

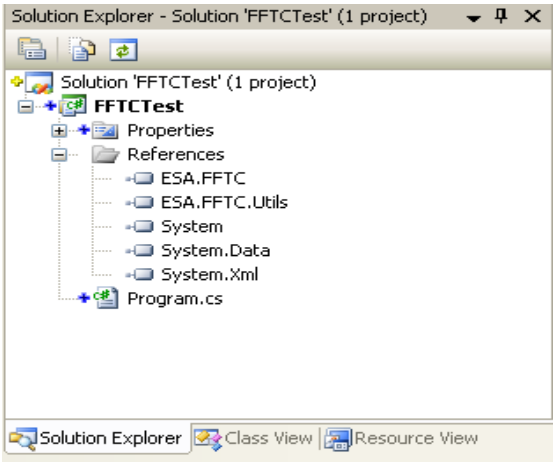
1. Create a new project by clicking File :: New Project… Select 'Console Application' and name it FFTCTest.



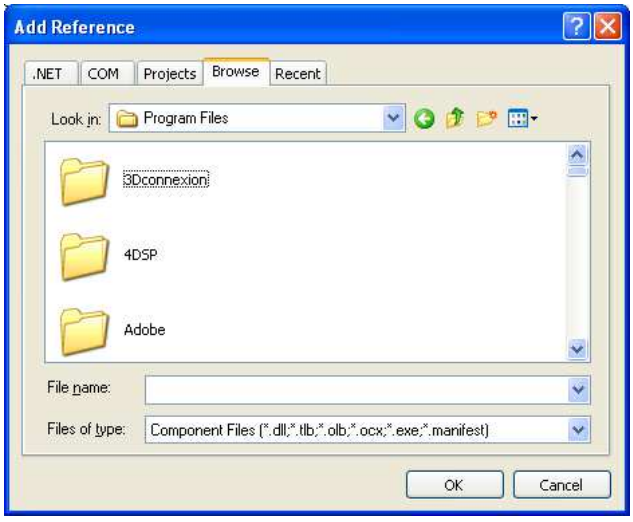
2. You'll see something like this:



3. Next add references to the necessary DLLs. Go to the solution explorer and right click References, Add Reference…



Select the Browse tab and select the ESA.FFTC.Utils, ESA.FFTC and ESA.Dataflow DLLs (the latter is not shown in the screen shot above but you must add it). The DLLs are originally to be found in the SDK directory under Program Files if you ran the installer.



4. Back in the code add the following line under the other using statements. This imports the namespace from the DLLs just referenced.

```
using ESA.FFTC;
using ESA.Dataflow;
```

5. Enter the code shown earlier to result in.

```
static void Main(string[] args)
{
    // Create an instance of FFTCModule
    FFTCModule fftc = new FFTCModule();

    // Create Blocks on the desired controllers
    FFTCBlock bP0 = fftc.P0.CreateBlock(32, 64);
    FFTCBlock bP1 = fftc.P1.CreateBlock(32, 64, 256, 128);
```

```
        // Perform an operation on the Blocks resulting in a Command
        Command c = bP1.Write(bP0);

        // Add Command to Control Vector
        ControlVector cv = new FFTCControlVector();
        cv.Add( c );

        // Write to binary file
        cv.Save("myCV.bin");

        // Write to xml file
        cv.Serialize("myCV.xml");
    }
```

6. Run the application by clicking Debug :: Start Debugging (or F5 if default C# settings are used). In the bin\Debug directory of the project there will be two new files called 'myCV.bin' and 'myCV.xml'.

3.3 Key Classes of ESA.FFTC

3.3.1 FFTCModule

The FFTCModule is the key class of the algorithm libraries. It exposes a model of the seven controllers, references to the supported data formats and various methods for performing operations on blocks of defined data. See on-line documentation for further information.

3.3.1.1 Primary Public Properties

Name	Description
FFTCore	The FFT core FFTController .
Name	Gets or sets the module name.
P0	P0 InputController.
P1	P1 MemoryController.
P2	P2 MemoryController.
P3	P3 MemoryController.
P4	P4 MemoryController.
P5	P5 OutputController.
State	Gets the state of the module.

Of particular note here are the controllers P0 to P5. To declare a block of data call the CreateBlock method of the desired controller.

3.3.1.2 Primary Static Fields – Supported Data Formats

A data format can be specified by use of a static field on FFTCModule.

Name	Description
d16BIT_PAR	16-bit parallel.

d16BIT_PAR_SI	16-bit parallel sign inverted.
d16BIT_SEQ	16-bit sequential.
d16BIT_SEQ_SI	16-bit sequential sign inverted.
d32BIT_PAR	32-bit parallel.
d32BIT_PAR_SI	32-bit parallel sign inverted.
d32BIT_SEQ	32-bit sequential.
d32BIT_SEQ_SI	32-bit sequential sign inverted.
defaultFormat	Default data format of the module.
dHYBRID_LONG	Hybrid long.
dIEEE_PAR	IEEE-754 parallel.
dIEEE_SEQ	IEEE-754 sequential.

All data formats are static for the FFTCModule. Access them as FFTCModule.dIEEE_PAR for example. When creating a block this can be specified as part of one of the overloaded methods. If not specified then the default format for that controller is used.

3.3.1.3 Primary Public Methods

Name	Description
Add	Uses the ALU to add two Blocks together.
Conjugate	Performs a transparent conjugate multiply on Block b.
Double	Performs a double operation on BlockFFT b.
FFT	Overloaded. Performs an FFT.
IFFT	Performs an inverse FFT.
Multiply	Overloaded. Uses the ALU to multiply two Blocks together.
Negate	Performs a negate operation on BlockFFT b.
Reset	Creates a Command containing a reset operation.
ResetControllers	Resets all the Controllers
Square	Overloaded. Performs a square operation on Block b.
Subtract	Uses the ALU to subtract one Block from another, thus a – b.
Write	Overloaded. Creates a Command for transferring data from the Controller represented by Block src to that represented by Block dst.

An example of a more complex operation that uses both the FFT and ALU units of the core in one pass is shown below:

```
            // Create an instance of FFTCModule
FFTCModule fftc = new FFTCModule();

// Create Blocks on the desired controllers
FFTCBlock bP1 = fftc.P1.CreateBlock(32, 64);
FFTCBlock bP2 = fftc.P2.CreateBlock(32, 64);
FFTCBlock bP3 = fftc.P3.CreateBlock(32, 64);

// Perform an operation on the Blocks resulting in a Command
Command c1 = bP3.Write(fftc.FFT(bP1 * bP2));
Command c2 = bP3.Write(bP1 * fftc.IFFT(bP2));
```

3.3.2 Block and FFTCBlock

The Block class defines 2-dimensional data as being on a particular sink or source controller (P0 to P5). A block is created by calling the CreateBlock method of the desired controller. All blocks are derived from the abstract Block class. A block is either given an automatic name or a user specified name. It must be unique. On an FFTC module all blocks are actually FFTCBlocks which is derived from Block.

Operations can be performed on blocks by use of the Read, Write and arithmetical operators +, - and *. Depending on the location (controller) of the block some operations will not be valid. For example it is not possible to multiply a block on the P5 controller as this would entail a read operation, something impossible for an output (sink). If this is attempted an exception would be thrown upon running the application.

Therefore to multiply two blocks together using the FFTC's arithmetic logic unit the following can be written:

```
// Create an instance of FFTCModule
FFTCModule fftc = new FFTCModule();

// Create Blocks on the desired controllers
FFTCBlock bP1 = fftc.P1.CreateBlock(32, 64);
FFTCBlock bP2 = fftc.P2.CreateBlock(32, 64);
FFTCBlock bP3 = fftc.P3.CreateBlock(32, 64);

// Perform an operation on the Blocks resulting in a Command
Command c = bP3.Write(bP1 * bP2);
```

3.3.2.1 Primary Public Properties (FFTCBlock)

Name	Description
DataSamples	Same as TotalSamples.
Format	Gets the DataFormat of the Block.
Height	Gets the height in samples.
Location	Gets the Controller on which the Block was declared.
Name	Gets or sets the name of the Block. Must be unique.
NoSequences	Gets the number of sequences.
SequenceSize	Gets the sequence size.
TotalDataSamples	Gets the number of DataSamples multipled by Height.
TotalLoops	Gets the total number of loops.
TotalSamples	Gets the total number of samples in the block.
Width	Gets the width in samples.

3.3.2.2 Primary Public Methods (FFTCBlock)

Name	Description
operator -	a – b: Subtract Block b from Block a using the ALU.
operator *	a * b, 1 * b: Multiply Block a by b or b by 1 using the ALU. In latter case source A of the FFTC chip will be transparent.
operator +	a + b: Add Block a to Block b using the ALU.
Read	Overloaded. Generates a Command that reads data from the Controller defined by this

	Block to the Controller defined by dst Block. If reading from a block defined on P0 then there can be up to two destinations in the one command.
Write	Generates a Command that writes data from the Controller defined by Block src to the Controller defined by this Block.

3.3.3 Derived Blocks – FFTCBlockInput, FFTCBlockOutput, FFTCBlockMem

The FFTCBlock class is abstract. Blocks created by the CreateBlock method are specific to that type of controller. Therefore the input controller P0 returns a FFTCBlockInput type for example. This has some additional and overridden properties compared to the base FFTCBlock and Block classes.

3.3.3.1 FFTCBlockInput Primary Public Properties

Name	Description
DataSamples	SequenceSize minus TotalZeros.
LeadingZeros	Gets the number of leading zeros.
TotalDataSamples	DataSamples multipled by Height.
TotalZeros	Gets the total number of zeros.
TrailingZeros	Gets the number of trailing zeros.

3.3.3.2 FFTCBlockOutput Primary Public Properties

Name	Description
Scale	Gets the scale.

3.3.3.3 FFTCBlockMem Primary Public Properties

Name	Description
Direction	Gets or sets the read/write direction to horizontal or vertical.
h	Sets the addressing to horizontal mode for all subsequent commands and returns itself.
NoSequences	Gets the number of sequences.
SequenceSize	Gets the sequence size.
TotalHeight	Gets the total height of the defined Block taking into consideration loop controllers.
TotalLoops	Gets the total number of loops.
TotalSamples	Gets the total number of samples based on width, height and the loop controllers.
TotalWidth	Gets the total width of the defined Block taking into consideration loop controllers.
TotalZeros	Gets the total number of zeros.
TrailingZeros	Gets the number of trailing zeros.
v	Sets the addressing to vertical mode for all subsequent commands and returns itself.
VerticalReverse	Gets or sets the vertical reverse mode. Cannot use Y direction with horizontal addressing.
X	Gets the start position of the Block.
Y	Gets the start line of the Block.

3.3.3.4 FFTCBlockMem Primary Public Methods

Name	Description
addLoop	Adds a loop to the loops controller.
GetLoopAddressing	Gets the MemoryAddressing3Loop associated with this Block.
removeLoop	Removes the last loop added to the loop controller.

Since a memory bank can be read from or written to and that it supports vertical, horizontal and looped addressing schemes it is far more complex than the input and output blocks. Of particular interest are:

- h, v properties: Putting a .v or .h after a BlockMem object will alter the addressing direction from that point on to vertical or horizontal respectively. It is a short hand method for setting the Direction property.
- X, Y properties: The memory banks are arranged in a 2D format. They can be randomly accessed. It is therefore possible to set the x and y parameters of the corner of the 2D block nearest the origin.
- Loops: The loop controllers permit up to 3 nested loops to be added for reading and writing. The properties beginning with 'Total' all take into consideration the loop settings. Loops are added or removed by use of the addLoop and removeLoop properties. In addition a VerticalReverse flag can be enabled or disabled that will reverse reading or writing when using loops in the vertical direction.

3.3.4 FFTCBlockFFT

The result of an operation in the FFT core is an FFTCBlockFFT object. These cannot be explicitly created by the user. For example the operation BlockA * BlockB will result in an automatically generated block of type FFTCBlockFFT. This can then be consumed by a Write method on a conventional destination block (FFTCBlockMem or FFTCBlockOutput).

3.3.5 ControlVector and FFTCControlVector

Commands are added to ControlVector objects. On the FFTC module the type of control vector is FFTCControlVector which is derived from ControlVector. The key methods and properties of a ControlVector are shown below:

3.3.5.1 Primary Public Properties (FFTCControlVector)

Name	Description
Count	Total number of Commands
HasInput	Is there at least one command that uses P0?
HasOutput	Is there at least one command that uses P5?
HasPaddingWord	Returns true if there is a padding word in this ControlVector, else false. Padding words are required if the number of bits in complete control vectors is not a multiple of 64.
Length	Gets the length of the control vector in words including any padding

	word.
LengthExcludingPaddingWord	Gets the length of the control vector in words excluding any padding word.
Name	Gets or sets the name of the ControlVector.
Target	Gets the target module which the instance of the FFTCModule that created the first command added to the control vector. All subsequent commands added must also be of same instance.
Words	*Gets control vector as a list of 32-bit words. If length is not multiple of 64-bits then add a padding word.*

3.3.5.2 Primary Public Methods

Name	Description
Add	Adds a Command
Clear	Removes all Commands.
Deserialize	Overloaded. Deserializes the given XML file into the ControlVector object.
Copy	Makes a deep copy.
Equals	Determines whether the specified Object is equal to the current Object.
GetInputCount	Gets the number of commands featuring P0.
GetOutputCount	Gets the number of command featuring P5.
Load	Overloaded. Loads the control vector from the binary file specified.
Save	Overloaded. Writes the ControlVector to a binary file.
Serialize	Overloaded. Serializes the ControlVector to an XML file with name based on current Name.
SetInputFormats	Sets all inputs found in the ControlVector to the given DataFormat.
SetOutputFormats	Sets all outputs found in the ControlVector to the given DataFormat.

Of interest are:

- Deserialize, Serialize: These methods read from and write to an xml file. A snippet of the generated xml is shown below. Per ControlVector its name, length in words, type, target name, target type and target assembly (dll that contains the target type) are present as attributes. Per Command the length in words and command availability are shown. Each Command consists of one ControllerSettings element per controller and these contain either one or two SettingsLayout elements (P0 and P5 have two separate sections of the control vector layout). A SettingsLayout has name, length (in words) and command availability bit attributes. Each setting has its own element. The attributes W, SB and NB are the word, start-bit and number of bits respectively. These are present in the xml to permit the xml representation to be self-describing and thus allow sanity checks and applications independent of these libraries to be written.

```
<?xml version="1.0" encoding="utf-8"?>
<ControlVector Name="CV7" Length="48" Type="ESA.FFTC.FFTCControlVector"
Assembly="ESA.FFTC, Version=2.0.0.1, Culture=neutral, PublicKeyToken=null"
TargetName="FFTC9" TargetType="ESA.FFTC.FFTCModule" TargetTypeAssembly="ESA.FFTC,
Version=2.0.0.1, Culture=neutral, PublicKeyToken=null">
  <Command Length="12" CommandAvailability="25">
    <ControllerSettings ControllerName="P0" Count="2">
      <SettingsLayout Name="ITC" Length="1" CAB="0">
        <Dest_P0_2 W="0" SB="0" NB="3">0</Dest_P0_2>
        <Dest_P0_1 W="0" SB="8" NB="3">1</Dest_P0_1>
        <Format_P0 W="0" SB="16" NB="5">3</Format_P0>
```

```
</SettingsLayout>
```

- GetInputCount, GetOutputCount: These methods get the number of commands in the control vector that feature inputs or outputs.
- Load, Save: Loads and saves a binary control vector (effectively assembly and disassembly).
- Equals and not equals operators: Permits simple comparison of control vectors. ControlVectors do not have to be pointing to same object or have same name, but all settings must otherwise be identical.
- SetInputFormats, SetOutputFormats: Sets the data format of all inputs or outputs to a given format.

3.3.6 Control Vector Wizard

The class FFTCControlVectorWizard has a number of static methods for quickly creating control vectors for:

- Echo
- Write
- Read
- FFT 1D Long
- FFT 1D Short
- FFT 2D

All these methods return ControlVectorList objects. This is a simple wrapper for holding multiple control vectors. Only the long 1D FFT method actually returns more than one control vector (one for loading twiddle vectors, one for processing).

4 FFTC Simulator and Performance Profiler Library

The FFTC simulator and performance profiler are available in the ESA.FFTC.Simulator dll. The simulator can either be used directly or more commonly and as recommended via the wrapper of the target board (e.g. OPDP or FTAB). The performance profiler is accessed via this library directly. The output of the profiler is an xml file that can be used as input for the performance profiler viewing control or a third party custom tool.

The FFTC module simulator is derived from DataflowModuleSimulator abstract class.

The following is provided as reference only for the simulator.

4.1 FFTCModuleSimulator

4.1.1 Simulation

The basic pattern for using the simulator directly is to instantiate FFTCModuleSimulator, then set it with a control vector, then read or write data as the algorithm requires. It can be instantiated and used according to the following pattern.

```
FFTCModuleSimulator fftcsim = new FFTCModuleSimulator();
fftcsim.Set(myControlVector);
fftcsim.Write(dataBuffer);
byte[] resultBuffer = new byte[1024];
int bytesRead = fftcsim.Read(resultBuffer, 0, 1024);
```

4.1.1.1 Primary Public Properties relating to Simulator Functionality

Name	Description
ControlVectorQueueLength	Gets the length of the control vector queue in words.
ControlVectorQueueMaxLength	Gets the maximum size of the queue in words.
Status	Gets the status of the simulator.
FFTCPerformanceProfileSettings	Gets or sets the performance profiler settings
Input	Gets the P0 input FIFO. This is encapsulated by the Write method.
Output	Gets the P5 output FIFO. This is encapsulated by the Read method.
P0 to P6	The controller simulators. It is typically not required for a user to access these explicitly.

4.1.1.2 Primary Public Methods relating to Simulator Functionality

Name	Description
GetControllerIDs	Gets an array of the controller ids.
GetControllerNames	Gets an array of the controller names.
GetControllerState	Overloaded. Gets the state of the controller unit specified.
Read	Reads from the simulator into the specified buffer.
ReadBytes	Reads the number of bytes specified by length from the simulator.
ReadInt64	Reads an eight-byte unsigned integer from the simulator.

Reset	Resets the simulator.
Set	Sends a new control vector to the simulator.
Write	Overloaded. Writes to the simulator.
Flush	Flushes the input FIFO. It is important to call this method after a sequence of write operations.
CanSet	Tests whether the simulator is currently in a state to accept the given control vector.
RunPeformanceProfiler	Runs a performance profile for the given ControlVector.

Below is a code sample. Note that a reference to nunit.framework is required if use of the Assert command is desired. Furthermore the use of reading and writing a long integer is purely for test purposes. Combined with data formats of IEEE parallel and no use of leading and trailing zeros or scaling factor the data will be transparently processed by the simulator and we should receive the same data out as we put in. A more efficient implementation would create the complete input buffer before writing this to the simulator in one operation. Also note that is it possible to wrap the Input and Output FIFOs (these are properties of the simulator) with standard .NET BinaryWriter or BinaryReader instances. The simulator library contains extension methods to these that permit the writing or reading of the supported complex data formats (CplxInt32, CplxInt16 and CplxFloat).

```
        [Test]
        public void FFTCSimIOSingleFFTCControlVectorSingleSample()
        {
            FFTCModule fftc = new FFTCModule();
            int w = 1024;
            int h = 1024;
            int loops = 3;

            // Create Blocks on the desired controllers
            FFTCBlock bP0 = fftc.P0.CreateBlock(FFTCModule.dIEEE_PAR, w, h);
            FFTCBlock bP1 = fftc.P1.CreateBlock(w, h);
            FFTCBlock bP2 = fftc.P2.CreateBlock(w, h);
            FFTCBlock bP5 = fftc.P5.CreateBlock(FFTCModule.dIEEE_PAR, w, h, 0);

            FFTCModuleSimulator fftcsim = new FFTCModuleSimulator();

            FFTCControlVector cv2 = new FFTCControlVector();
            cv2.Add(bP1.Write(bP0));
            cv2.Add(bP5.Write(bP1));

            DateTime st = DateTime.Now;
            for (int x = 0; x < loops; x++)
            {
                fftcsim.Set(cv2);
                for (long i = 0; i < w * h; i++)
                {
                    fftcsim.Write(i);
                }
                fftcsim.Flush();
                for (long i = 0; i < w * h; i++)
                {
                    long f = fftcsim.ReadInt64();
                    Assert.AreEqual(i, f);
                }
            }
            TimeSpan ts = DateTime.Now - st;
            fftcsim.Reset();
            Console.WriteLine(string.Format("Done in {0}s", ts.TotalSeconds));
            Console.WriteLine(string.Format("Samples/sec {0}", (long)(w * h) * loops /
ts.TotalSeconds));
        }
```

```
[Test]
public void FFTCSimP0LeadingAndTrailingZeros()
{
    // Some variables to make setting the data blocks easier
    // w = width, h = height, lz = leading zeros, tz = trailing zeros
    int w = 1024;
    int h = 768;
    int lz = 128;
    int tz = 64;

    // Instaniate the FFTCModule of the algorithm library
    FFTCModule fftc = new FFTCModule();

    // Create Blocks on the desired controllers
    FFTCBlockInput bP0 = fftc.P0.CreateBlock(FFTCModule.dIEEE_PAR, w, h, lz, tz);
    FFTCBlockMem bP1 = fftc.P1.CreateBlock(w, h);
    FFTCBlockOutput bP5 = fftc.P5.CreateBlock(FFTCModule.dIEEE_PAR, w, h, 0);

    // Make the control vector
    FFTCControlVector cv = new FFTCControlVector();
    cv.Add(bP1.Write(bP0));
    cv.Add(bP5.Write(bP1));

    // Instaniate the simulator
    FFTCModuleSimulator fftcsim = new FFTCModuleSimulator();

    // Set a counter for data generation and verification
    int ctr = 0;

    // Create binary readers and writers
    BinaryWriter bw = new BinaryWriter(fftcsim.Input);
    BinaryReader br = new BinaryReader(fftcsim.Output);

    // Set control vector on the control vector queue of the simulator
    fftcsim.Set(cv);

    // Note the start time for performance monitoring purposes.
    DateTime st = DateTime.Now;

    for (int j = 0; j < h; j++)
    {
        for (int i = 0; i < bP0.DataSamples; i++)
        {
            bw.Write(new CplxFloat(ctr++));
        }
    }
    bw.Flush();

    // Read per sequence lz zeros, then DataSamples of the
    //  ramp,then tz zeros. Repeat h times.
    // Perform asserts on data received (csUnit)
    ctr = 0;
    for (int c = 0; c < h; c++)
    {
        for (int j = 0; j < lz; j++)
        {
            CplxFloat f = br.ReadCplxFloat();
            Assert.AreEqual(CplxFloat.Zero, f);
        }

        for (int i = 0; i < bP0.DataSamples; i++)
        {
            CplxFloat f = br.ReadCplxFloat();
            Assert.AreEqual(new CplxFloat(ctr++), f);
        }

        for (int j = 0; j < tz; j++)
        {
            CplxFloat f = br.ReadCplxFloat();
```

```
                Assert.AreEqual(CplxFloat.Zero, f);
            }
        }

        TimeSpan ts = DateTime.Now - st;
        fftcsim.Reset();
        Console.WriteLine(string.Format("Done in {0}s", ts.TotalSeconds));
        Console.WriteLine(string.Format("Samples/sec {0}",
            w * h / ts.TotalSeconds));
    }
```

4.1.2 Performance Profiler

The simulator class FFTCModuleSimulator also contains the functionality to perform performance profiling; that is produce an estimate of the timings on each of the FFTC controllers for a given control vector. The FFTCModuleSimulator must not also be running a simulation at the same time as the profiling or an exception will be thrown. The steps taken to use the profiler are:

1. Create an instance of FFTCModuleSimulator
2. Set any relevant properties of the FFTCPeformanceProfileSettings property:
    a. Clock speeds for the various controllers (in Hz)
    b. Maximum expected speed for input and output (P0 and P5) as these are typically constrained by the speed of the external bus (e.g. SpaceWire). Specified in bytes per second.
    c. Set the independent IO flag – some external buses are cannot run true duplex therefore P0 and P5 cannot run at the same time. Set to false is this is the case.
3. Call the RunPerformanceProfile method passing in a control vector. This returns a ControlVectorTiming object containing the results. Results can be accessed either programmatically, via xml or via the PerformanceProfile graphical control (in the ESA.FFTC.Controls library).

5 FFTC Controls and Utilities

Many of the low level classes used by the SDK may also be useful to developers writing their own FFTC based applications. These utilities are split between graphical and non-graphical.

5.1 Graphical Controls

The graphical controls used to build the SDE are also available for users to create their own graphical applications. They are contained within the ESA.Dataflow.GUI library.

5.2 Non-graphical Components

These are contained in the ESA.FFTC.Utils library. They include:

- Data formats
- Data sources and sinks
- Data generators
- General numeric operations

5.2.1 Data formats

The FFTC natively only supports complex numbers. These are not supported as standard by programming languages so the ESA.FFTC.Utils library provides:

- Complex floating point
- Complex 16-bit integer
- Complex 32-bit integer
- Complex 8-bit integer

They permit standard arithmetic and logic operations.

5.2.2 Data generators

Data generators are included for creating twiddle vectors, sine, spikes and LFSR.

5.2.3 General numeric operations

The Numeric class contains a number of useful data conversion routines.