SystemView
BY ELANIX®

A Guide
To The
Real Time DSP Architect

RTDA

EAGLEWARE
Electronic Design Automation Software
ELANIX
Table of Contents

1.0 INTRODUCTION ........................................................................................................ 5

2.0 INSTALLATION INSTRUCTIONS ........................................................................ 7

   2.1 INSTALLATION ................................................................. 7
   2.2 SOFTWARE REQUIREMENTS ............................................. 7
   2.3 HARDWARE REQUIREMENTS ............................................. 7

3.0 OVERVIEW ............................................................................................................... 8

   3.1 OPERATIONAL CONCEPT .................................................. 8
   3.2 OVERVIEW ................................................................. 8
   3.3 CODE COMPOSER STUDIO INTERFACE ......................... 9

4.0 CODE COMPOSER STUDIO WITH SYSTEMVIEW ........................................... 10

   4.1 OVERVIEW ............................................................... 10
   4.2 CCS WORKSPACE ....................................................... 11
   4.3 DATA EXCHANGE ........................................................ 12
   4.4 GUIDELINES FOR CREATING A CCS PROGRAM .......... 13
       4.4.1 Writing CCS Programs using RTDX ......................... 14
       4.4.2 Command Channel ................................................. 14
       4.4.3 Status Channel ..................................................... 14
       4.4.4 Input Channel ........................................................ 14
       4.4.5 Output Channel .................................................... 15
       4.4.6 Initialization .......................................................... 15
   4.5 CREATING A SYSTEMVIEW SIMULATION ..................... 16
   4.6 RUNNING CCS AND SYSTEMVIEW SIMULTANEOUSLY .... 20

APPENDIX: EXAMPLE FILES ....................................................................................... 21

Real Time DSP Architect
1.0 Introduction

The Real Time DSP Architect (RTDA) by Elanix is a comprehensive software module that works within SystemView to provide a design flow, from bit-true system design to implementation in a target DSP such as the Texas Instruments™ C5000/C6000 DSP platforms. The design flow is accomplished using a seamless interface between the SystemView RTDA module and the Texas Instruments Code Composer Studio™ (CCS), which is an Integrated Development Environment (IDE).

SystemView utilizes the CCS open architecture to create an efficient interface. The SystemView RTDA can simplify the development process of code validation, debugging time, and design optimization, which can reduce or eliminate problems for programmers that design and test real-time embedded signal processing applications.

The SystemView interface to CCS consists of real-time source, general, and sink token blocks that are part of the SystemView DSP library. These blocks provide real-time access to the application running on a target DSP, such as the Texas Instruments Evaluation Module (EVM) or the DSP Starter Kit (DSK), through the Real Time Data Exchange™ (RTDX) software.

SystemView and the RTDA software allows the user to fully integrate the real-time software prototype into the DSP design, by acquiring data from, and sending data to, the target DSP. This supports the hardware-in-the-loop simulations using a target DSP such as the TI C5x/C6x platforms, to validate the design.
Key features of the RTDA software include:

- Hardware-in-the-loop simulation using Texas Instruments TMS320C54x or C6x DSK or EVM board

- Support for multiple RTDA tokens in a SystemView simulation, each communicating with separate DSP board, or, on single processor board using virtual multi-tasking.

- Integrated Design Flow from Bit-True System Design to target DSP Implementation.

- Close interaction between SystemView and the DSP platform, using the Code Composer Studio Debugger, to reduce code validation and debugging time of the target DSP.

- Data Acquisition support, allowing SystemView to acquire real-time data from the target DSP, in addition to transferring data from SystemView to the target DSP for real time processing and analysis.
2.0 Installation Instructions

2.1 Installation
The Real Time DSP Architect (RTDA) is automatically installed with the latest version of SystemView.

2.2 Software Requirements
To use the RTDA, the following software is available from the manufacturer, and must be installed on the PC and running under Windows 95/98, 2000 or Windows NT. NOTE: Later versions of Texas Instruments Code Composer Studio may not support Windows 95, and the user should refer to the Texas Instrument documentation for hardware interface requirements.

<table>
<thead>
<tr>
<th>Product</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemView by Elanix</td>
<td>Eagleware-Elanix</td>
</tr>
<tr>
<td>RTDA Library</td>
<td>Eagleware-Elanix</td>
</tr>
<tr>
<td>DSP Library</td>
<td>Eagleware-Elanix</td>
</tr>
<tr>
<td>Code Composer Studio v.1.2 or later.</td>
<td>Texas Instruments</td>
</tr>
</tbody>
</table>

2.3 Hardware Requirements
To use the real-time signal generation, data acquisition and analysis capabilities of Real Time DSP Architect, you must connect a target DSP such as the Texas Instruments C5000 or C6000, mounted on a test bed such as the TI EVM/DSK to a PC. The configuration of the PC should be a Pentium 400 MHz or faster, having a minimum of 128MB of RAM (256MB recommended), with at least 100MB of hard disk space available.
3.0 Overview

3.1 Operational Concept

The RTDA software extends the SystemView environment to provide a design flow from bit-true system design to implementation in DSP platforms like the TI TMS320C5000 or TMS320C6000. This design flow is achieved through an interface between the SystemView design environment and the Code Composer Studio development environment for real-time analysis, debugging, validation and optimization.

3.2 Overview

This section provides an overview of the RTDA, and it explains the interface with CCS. The design flow shown in Figure 1, illustrates the relationship of the SystemView and CCS interface over the RTDX data path, and the hardware connection between the PC and the DSP platform. SystemView provides a plug-in for communication between SystemView, Code Composer Studio, and the target DSP. A plug-in is a CCS component that can access CCS properties, invoke methods, and detect events. The RTDA link provides the capability for real-time analysis that helps reduce the debugging time, and validate DSP simulations.

The Real-Time DSP Architect for SystemView v5.0 contains new and updated features compared with that of SystemView v4.5. RTDA now supports multiple RTDA tokens for a simulation. Each token is given a unique identifier and can be applied to a single DSP board, or a specific board and/or task number, if more than one DSP is configured on the user’s system. For RTDA tokens with more than one input or output, each of the ports can be configured independently for rate and RTDX data transfer.
3.3 Code Composer Studio Interface

Texas Instruments offers an Integrated Development Environment (IDE) for DSP software development, called Code Composer Studio (CCS). The software is available for either the C5000 or C6000 DSP products. The interface between SystemView and the CCS uses RTDA source, sink, and general tokens that are an integral part of the SystemView DSP library. The DSP library has specific RTDA tokens designed to support development of applicable DSP products. RTDA supports multiple tokens and multiple target processors per simulation. The tokens provide real-time access to the application running on the test bed through the Real Time Data Exchange (RTDX), which is a component of Code Composer Studio.
4.0 Code Composer Studio with SystemView

4.1 Overview
The following is a description of how a selected target DSP is integrated into a SystemView simulation, using a Code Composer Studio target “C” program, and RTDA.

The DSP bit-true library contains a collection of tokens, specifically designed for digital signal processing. RTDA extends the SystemView environment to support the design and validation of the target DSP. RTDA uses three token groups that support the DSP validation:

Source Tokens. One-way data transfer; Source tokens have outputs only, and provide time-sampled data to the SystemView simulation. A SystemView RTDA Source token provides real-time data acquisition from the target DSP. This allows the designer to utilize real time data coming from the target DSP as a source signal in SystemView.

Sink Tokens. One-way data transfer; Sink tokens have inputs only, and accept data from the SystemView simulation. A SystemView RTDA Sink token provides a signal generation interface to the target DSP. It allows the user to send signal data generated by the SystemView design to the target DSP for real-time processing.

General Tokens. Data flows in both directions, from SystemView to Code Composer Studio, and back to SystemView. The DSP target program processes the data from SystemView and transfers the results back to the SystemView simulation. A SystemView RTDA General token allows the user to replace a portion of the SystemView design with a link to the target DSP while running the program under development, for hardware-in-the-loop simulation.
The TI Debug interface provides a close interaction between SystemView, the debugging component of Code Composer Studio, and the target DSP. This allows the user to set break points in the CCS debugging component before executing the system in SystemView. The application running on the target DSP is automatically executed and will stop when it reaches a break point.

Although RTDA is compatible with any TI DSP board that can be used with Code Composer Studio, it is helpful to describe some DSP units used for development that are available through TI resellers. The DSP Starter Kit, (DSK) is a Texas Instrument entry-level product that has a target DSP mounted to a circuit board. The circuit has been configured for connection through the parallel port of a PC, through the JTAG interface of the DSP chip. The acronym JTAG refers to a standard chip I/O interface that is based on the Joint Test Action Group (JTAG) test standard, as adopted by the IEEE.

4.2 CCS Workspace

A CCS workspace contains configuration information related to a CCS project, the CCS environment and the currently loaded GEL file. Breakpoints used in debugging a CCS project are also saved in the workspace.

Version 4.5 required that a single Code Composer Studio workspace, with the name “SvuRTDA.wks”, be defined for use with RTDA tokens. The requirement, which was necessary for loading the SystemView plug-in OCX, has now been removed, since SystemView now communicates with CCS through the TI COM interface. When SystemView launches Code Composer Studio, the user can specify whether a workspace should be loaded. The workspace options are indicated within the RTDA token parameter settings.

The default workspace used when CCS is launched should contain a GEL file that describes how certain functions, such as a startup routine, are to
be performed. GEL is an acronym for General Extension Language, and is
an interpretive language that is similar to C. In Code Composer Studio 2.0,
a GEL file can be established using the CCS setup program. Note that
using an incorrect GEL file may cause CCS launch problems.

4.3 Data Exchange
After the “C” target program is written, it is compiled and linked within
the CCS IDE. To finish the simulation loop, a supporting SystemView
simulation is created and launched with RTDA resident in the IDE. The
target program may be debugged in the Code Composer Studio IDE using
one or more break points, or using single step operation. Communication
between Code Composer Studio and the target DSP is through the Real
Time Data Exchange (RTDX) communication protocol.

An RTDA DSP target program operates on an input frame of data per call,
where a processing frame is defined in units of time. The frame time is
determined by the number of input samples divided by the sample rate, or
equivalently, the number of samples multiplied by the sample spacing. An
RTDA token can have multiple inputs, each with its own rate, with the
restriction that the frame time be the same for each input. There is a
processing delay equal to the processing frame time, in which the input
samples are collected.

The number of input samples transferred to the target DSP during a
processing iteration is referred to as the “Buffer Size” parameter. A
performance advantage is realized for block transfers, and the user should
specify as large a buffer size as practical, however, the longer the block of
data, the longer the induced processing delay. For example, a
programmed 1024 sample FFT on the EVM/DSK would have an ideal
Input Vector Length equal to the FFT block size of 1024. The processing
delay will be equal to 1024 samples.

For the RTDA General Token, there is also a “Buffer Size” for each of the
output channels, which represents the amount of processed data that is sent
back to SystemView from Code Composer Studio. The output buffer size
determines the output rate, where output rate is equal to the number of output samples divided by the frame time.

An RTDA Source Token, like all other SystemView source tokens, operates at the system sample rate. The processing frame time is therefore completely determined by the output buffer size. Because the output rate is fixed, there is a requirement that all outputs from an RTDA source have the same buffer size.

4.4 Guidelines for Creating a CCS Program

This section describes guidelines for creating a program that will run in CCS, and accept and/or generate data from a SystemView simulation. In addition to the DSP algorithm, the user must supply an RTDX interface “C” code routine, which sets up communication channels between the target DSP and SystemView. Data is transmitted to SystemView using General and Sink tokens, or transmitted from SystemView to the target DSP using General and Source tokens.

The RTDX channels contain data, which are either command information sent to the target DSP from the SystemView simulation, sampled data generated by either SystemView or the DSP, or status information reporting back from the DSP. Specific naming conventions must be followed for selection of the channels, so that the names specified by the RTDA token match names used by the target interface code. It is required to structure the code so that the RTDX communication and the users DSP algorithm processing are implemented in a specific order.

The appendix contains some CCS code examples and supporting simulations, for Source, General and Sink tokens, located in the TI subfolder installed under the SystemView Examples folder.
4.4.1 Writing CCS Programs using RTDX

When writing a CCS program using RTDX, the user must create, enable, and read/write each of the RTDX channels defined by the corresponding SystemView RTDA token. The DSP target program must read all data written on an RTDX channel sent from SystemView.

4.4.2 Command Channel

The Command Channel is an RTDX input channel that provides command information from SystemView and supplies buffer size information. At each call to the target program, the command channel sends the token identifier along with the SystemView command, “initialize”, “run” or “finalize”. At initialization, the command channel also sends the number of samples for each input and output channel. The Command Channel may be initialized globally with the following macro:

```
RTDX_CreateInputChannel (ch_command);
```

4.4.3 Status Channel

The Status Channel is an RTDX output channel that provides status information back to SystemView. At the end of each target processing iteration, the status channel sends back a zero for successful operation, or an error code to indicate failure. The Status Channel may be initialized globally with the following macro:

```
RTDX_CreateOutputChannel (ch_status);
```

4.4.4 Input Channel

For an Input Channel, the user must create the channel names as referenced in the RTDA token. Keep in mind that here is a limit to 20 input and 20 output channels per RTDA token. The appendix source code shows the RTDX input channel structure that may be initialized globally by:

```
RTDX_CreateInputChannel (ch_in0);
RTDX_CreateInputChannel (ch_in1);
```
4.4.5 Output Channel

Structure the Output Channel names as listed in the RTDA token parameter dialog. Referring to the appendix, the source code example shows that the RTDX output channel structure may be initialized globally by the following macro:

```c
RTDX_CreateOutputChannel (ch_out0);
RTDX_CreateOutputChannel (ch_out1);
```

The general token also has output channels to SystemView that contain target DSP processed data. The general token example in the appendix shows how it processes the input data and assigns the outputs to two output channels.

4.4.6 Initialization

In the body of the main function, the user must include the target initialize function call shown, before any RTDX communication can take place.

```c
TARGET_INITIALIZE (); /* Enable RTDX interrupt */
```

Refer to the Code Composer Studio documentation for information and examples for creating RTDX applications.

The SystemView RTDA example code, listed in the Appendix, provides the user with an example of multiple RTDA tokens in a system, and how the target application can receive and distribute RTDA commands from SystemView. This example is also included in the SystemView installation.
4.5 Creating a SystemView Simulation

After completion of the target DSP program, the user must create a SystemView simulation containing one or more RTDA tokens. If a single processor board is installed on the user’s system, all RTDA tokens must point to the same DSP target executable program. The target program can be set up to base its processing on each particular RTDA token.

If more than one DSP board, or a board with more than one processor, is installed on the user’s system, then there can be a separate executable for each of the DSP processors. An RTDA token can be set to load an executable program to any of the processors available in the system.

To add an RTDA token to a SystemView simulation, toggle the Library button located at the left end of the tool bar to view the optional libraries. Next click/drag the DSP token into the screen area, then right click and select library to display the DSP library as shown in Figure 4.

![SystemView Bit-true DSP Library](image)

**Figure 4** SystemView Bit-true DSP Library
Next, select the token type, which can be Source, General or Sink. Once the token is selected, click the Parameters button on the right and it will launch the RTDA Parameter dialog box, as shown in Figure 5. The RTDA Token Dialog allows the user to specify information about the target DSP, how data is to be transferred to and from the target and how the data processed on the DSP target is to be used within the SystemView environment.

![Figure 5 TI General Token Parameter Dialog box.](image)

The first group of dialog controls contains information about the target, including specifying the board and task numbers and the path to the target executable. The token ID # is an identifier sent to the target at each processing iteration. The number should be unique for each RTDA token, as the target cannot distinguish between two with the same identifier. The CCS workspace controls allow setting the name of the workspace indicating whether SystemView should load it. Since only one workspace
can be loaded into CCS at a time, SystemView will load the first workspace encountered in an RTDA token.

The command channel group of controls allows the user to specify the names of the command and status channels used by the target program. The transfer type indicates the register size of the channel used to send command and status information. There is a check box to indicate using RTDA under version 4.5 or earlier of SystemView.

The RTDX configuration controls are to specify the size and number of main RTDX buffers used by Code Composer Studio. Refer to the CCS documentation for further information about these parameters.

The next two groups are used to specify parameters for the input and output channels of the target program. To edit any of the rows in the list box, double-click the desired row, or select the row and press the edit button. Data output from an RTDA token can be converted to a user-specified type, similar to many of the tokens in the DSP library.

The parameters from the dialog can be saved and loaded from an .INI file. The RTDA token will read/write all parameters from the [data] section. Some of the parameter tag names are different from those of v4.5; however, old .INI files can still be read. The table below lists the parameters from the .INI file.
<table>
<thead>
<tr>
<th>Parameter Tag Name</th>
<th>Tag Value Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>TokenID</td>
<td>Number</td>
</tr>
<tr>
<td>ProcessorBoard</td>
<td>Number</td>
</tr>
<tr>
<td>ProcessorTask</td>
<td>Number</td>
</tr>
<tr>
<td>Executable</td>
<td>Character String</td>
</tr>
<tr>
<td>Workspace</td>
<td>Character String</td>
</tr>
<tr>
<td>LoadWorkspace</td>
<td>True or False</td>
</tr>
<tr>
<td>MainBufferSize</td>
<td>Number (minimum 1024)</td>
</tr>
<tr>
<td>NumberMainBuffers</td>
<td>Number (minimum 4)</td>
</tr>
<tr>
<td>CommandName</td>
<td>Character String</td>
</tr>
<tr>
<td>StatusName</td>
<td>Character String</td>
</tr>
<tr>
<td>RtdxType</td>
<td>int8, int16, int32</td>
</tr>
<tr>
<td>OldProtocol</td>
<td>True or False</td>
</tr>
<tr>
<td>NumberInputs</td>
<td>Number</td>
</tr>
<tr>
<td>InputName&lt;N&gt;</td>
<td>Character String</td>
</tr>
<tr>
<td>InputRtdxType&lt;N&gt;</td>
<td>int8, int16, int32, float, double</td>
</tr>
<tr>
<td>InputLength&lt;N&gt;</td>
<td>Number</td>
</tr>
<tr>
<td>NumberOutputs</td>
<td>Number</td>
</tr>
<tr>
<td>OutputName&lt;N&gt;</td>
<td>Character String</td>
</tr>
<tr>
<td>OutputRtdxType&lt;N&gt;</td>
<td>int8, int16, int32, float, double</td>
</tr>
<tr>
<td>OutputLength&lt;N&gt;</td>
<td>Number</td>
</tr>
<tr>
<td>OutputSvuType&lt;N&gt;</td>
<td>same, uint, int, fixed, float, double</td>
</tr>
<tr>
<td>OutputRegisterSize&lt;N&gt;</td>
<td>Number</td>
</tr>
<tr>
<td>OutputFractionSize&lt;N&gt;</td>
<td>Number</td>
</tr>
</tbody>
</table>

Table 1 RTDA Initialization File Parameters.
4.6 Running CCS and SystemView Simultaneously.

A SystemView RTDA simulation can be run either with or without Code Composer Studio debugging. When debugging is required, the user can set break points, or single step through the DSP code. The SystemView simulation will halt and wait for CCS to resume the data transfer to/from SystemView. When CCS debugging is not required, an integrated SystemView/CCS simulation takes the form of a standard SystemView simulation. The user sets the token parameters prior to the simulation run and SystemView then starts CCS IDE.

An RTDA simulation can be terminated using the standard SystemView stop button next to the green “go” button on the toolbar. The target program will be halted, and Code Composer will be left running.
APPENDIX: Example Files

```c
#include <stdio.h>                   /* fprintf() */
#include <rtdx.h>                    /* RTDX */
#include "target.h"                  /* TARGET_INITIALIZE() */

/* Source Token is a four sample wide symmetric pulse train */

int SourceInit(int *time)
{
    *time = 0;
    return 0;
}

int Source(int *time, int outlen, int *output)
{
    const int width = 4;
    static int pulse = 1;
    int i;
    for (i = 0; i < outlen; i++)
    {
        if (*time >= width)
        {
            output[i] = *time = 0;
            pulse = -pulse;
        }
        output[i] = pulse;
        (*time)++;
    }
    return 0;
}

/* General Token is a two sample average */

int GeneralInit(int *last)
{
    *last = 0;
    return 0;
}

int General(int *last, int *input, int outlen, int *output)
{
    int i;
    output[0] = *last + input[0];
    for (i = 1; i < outlen; i++)
    {
        output[i] = input[i - 1] + input[i];
    }
    *last = input[outlen - 1];
    return 0;
}
```

Real Time DSP Architect
int SinkInit(FILE **pfile, int *sum)
{
    *pfile = fopen("C:\RTDASink.txt", "w");
    if (*pfile)
    {
        *sum = 0;
        return 0;
    }
    else return 1;
}

int Sink(int *sum, int inlen, int *input)
{
    int i;
    for (i = 0; i < inlen; i++)
    {
        *sum += input[i];
    }
    return 0;
}

int SinkFinal(FILE *pfile, int sum)
{
    if (!pfile) return 0;
    if (fprintf(pfile, "sum=%d\n", sum) < 0)
    {
        fclose(pfile);
        return 2;
    }
    if (fclose(pfile)) return 3;
    else return 0;
}
/ Create RTDX channels /

RTDX_CreateInputChannel(ch_command);
RTDX_CreateOutputChannel(ch_status);
RTDX_CreateInputChannel(ch_input);
RTDX_CreateOutputChannel(ch_output);

#define MAX_BUFFER_SIZE 32

main()
{
    int command[MAX_BUFFER_SIZE], status, token_id, svu_flag;
    int input[MAX_BUFFER_SIZE], output[MAX_BUFFER_SIZE];
    int outlenSource, inlenGeneral, outlenGeneral, inlenSink;
    int time, last, sum;
    FILE *pfile;

    TARGET_INITIALIZE(); /* Enable RTDX */

    RTDX_enableInput(&ch_command);
    RTDX_enableOutput(&ch_status);
    RTDX_enableInput(&ch_input);
    RTDX_enableOutput(&ch_output);

    for (;;) /* Infinite message loop. SystemView will Halt the program */
    {
        while(!RTDX_read(&ch_command, command, 2 * sizeof(int)));

        token_id = command[0];
        svu_flag = command[1];
switch (token_id)
{
    case 1: /* Source Token - 1 output */
        if (svu_flag > 0) /* initialize */
            { 
            while(!RTDX_read(&ch_command, &outlenSource, sizeof(int)));
                if (outlenSource < MAX_BUFFER_SIZE)
                    {
                        status = SourceInit(&time);
                    }
                else status = -1;
            RTDX_write(&ch_status, &status, sizeof(int));
        }
    else if (!svu_flag) /* run */
        { 
            status = Source(&time, outlenSource, output);
            RTDX_write(&ch_status, &status, sizeof(int));
            if (!status)
                { 
                RTDX_write(&ch_output, output, outlenSource * sizeof(int));
            }
        }
    else /* finalize */
        {
            RTDX_write(&ch_status, &status, sizeof(int));
        }
    break;
    case 2: /* General Token - 1 input, 1 output */
        if (svu_flag > 0) /* initialize */
            { 
            while(!RTDX_read(&ch_command, command, 2 * sizeof(int)));
                inlenGeneral = command[0];
                outlenGeneral = command[1];
                if (inlenGeneral < MAX_BUFFER_SIZE)
                    { 
                    status = GeneralInit(&last);
                    }
                else status = -1;
            RTDX_write(&ch_status, &status, sizeof(int));
        }
    else if (!svu_flag) /* run */
        { 
            while(!RTDX_read(&ch_input, input, inlenGeneral * sizeof(int)));
            status = General(&last, input, outlenGeneral, output);
            RTDX_write(&ch_status, &status, sizeof(int));
            if (!status)
                { 
                }
case 3: /* Sink Token - 1 input */
    if (svu_flag > 0) /* initialize */
    {
        while(!RTDX_read(&ch_command, &inlenSink, sizeof(int)));
        if (inlenSink < MAX_BUFFER_SIZE)
            status = SinkInit(pfile, &sum);
        else status = -1;
    }
    else if (!svu_flag) /* run */
    {
        while(!RTDX_read(&ch_input, input, inlenSink * sizeof(int)));
        status = Sink(&sum, inlenSink, input);
    }
    else /* finalize */
    {
        status = SinkFinal(pfile, sum);
    }
    RTDX_write(&ch_status, &status, sizeof(int));
    break;
}