Adaptive DFE™ Application Note

Formal Status
March 19, 2013

About this Document

Topics discussed include the following:

Revision History

March 19, 2013, Formal Status

This release does not include any technical changes.

April 19, 2011, Formal Status

First release of the document.

Overview

The bit-error-rate-based (BER-based) Adaptive DFE (ADFE) Application (the App) uses the IDT RapidIO Switch API software package (the API) to implement BER-based ADFE.

BER-based ADFE has two operating modes: Monitoring and Adjusting. When in Monitoring mode, the App checks the BER on a lane, or on multiple lanes, periodically to ensure that the BER has not exceeded the required maximum. A count of the number of bit errors is maintained. The count is decremented at a rate configured to match the target BER for the link. If the count exceeds the configured maximum, then the App uses the API's "adjust" functionality to change the DFE settings for the lane to improve the signal quality.

"Adjusting" searches for DFE coefficients that improve signal quality. The search algorithm adjusts each configured coefficient sequentially. Each time a coefficient is changed, the BER is checked to determine if it has improved. The coefficients are adjusted by a programmable amount (Delta), first adding and then subtracting. Each time all coefficients are tried and no better values are found, Delta is reduced by 1 for all coefficients. Adjustment completes and returns to Monitoring if one of the following occurs: a BER of 0 is measured, all Delta values are 0, or the programmed number of Adjustment attempts is completed. If the BER exceeds a programmed rate, then Adjustment aborts and no further Monitoring or Adjustment is performed.

The App manages a configured set of devices, ports, and lanes. The App supports hot swap of devices, and hot swap of components to the monitored links.

The App is intended to operate in a standard Linux environment that incorporates RapidIO support. The App uses standard Linux facilities for operation, such as:

- sysfs for register access to RapidIO devices
- System logging facility for debug and status information
- File system for configuration information

For specific application usage examples, see ADFE Application Configuration File Recommendations.

ADFE Application Installation/Execution Instructions

The ADFE Application has the following directories and contents:

- SRIO_API – Contains the license file, makefile.linux, and three subdirectories:
  - inc: Include/definitions files for the API
  - src: Source code files for the API
  - doc: Documentation for the API
- ADFE_APP – Contains the ADFE application source and definitions file, license file, as well as a makefile to create the application. Note that the application depends on linking to the SRIO_API library.
- ADFE_Docs – This document, and the *Signal Quality Optimization Application Note*.
- ADFE_Scripts – Example configuration file

To install the application:

1. Copy the contents of the SRIO_API, ADFE_APP, and ADFE_Scripts directories to the target Linux system.
2. Install the SRIO API library by performing the following steps:
   a. “cd” to the SRIO_API directory
   b. “make” the API
   c. “make install” to copy the API to the standard library directory for Linux
3. Compile the ADFE Application by performing the following steps:
   a. “cd” to the ADFE_APP directory
   b. “make” the application
4. Create a configuration file for you system:
   a. Copy the example.conf file from the ADFE_Scripts directory to the /etc directory
   b. Rename the example.conf file in the /etc directory to “adfe.conf”
   c. Modify the “adfe.conf” file to reflect the ADFE monitoring/adjustment parameters required for your system.
5. Execute the ADFE Application
   a. “cd” to the ADFE_APP directory
   b. enter “./adfed” without the quotation marks.
   c. enter “tail -f /var/log/debug” to determine if there are any syntax errors in the adfe.conf file, and to check that the ADFE Application is executing correctly.

ADFE Application Configuration File Recommendations

The ADFE Application configuration file must contain definitions for all RapidIO switches with ports/lanes that must be monitored and adjusted. These switches may appear in the system after the ADFE Application has been started. The ADFE main process periodically checks for the appearance of any devices in the configuration file, and starts an ADFE worker process for that device.

Similarly, the configuration file must define all ports and lanes that must be monitored and adjusted. The ADFE worker process for each switch periodically checks the configured ports for transitions from PORT_UNINIT to PORT_OK, and starts monitoring when this happens. When a port transitions from PORT_OK to PORT_UNINIT during monitoring, monitoring is stopped. If the port transitions back to PORT_OK, then monitoring is restarted. However, if the port transitions to PORT_UNINIT during adjustment, then adjustment and monitoring are permanently stopped for that port.

IDT recommends that only ports/lanes whose electrical characteristics fall between the *RapidIO Specification (Rev. 2.1)* Medium and Long reach channel characteristics for 5.0 and 6.25 Gbaud operation should be monitored. Monitoring parameters of these channels should be configured to check that the bit rate does not exceed required channel bit error rates. It is possible to monitor for bit error rates of up to 1 bit error in 10^9 bits. On 6.25 Gbaud lanes, this corresponds to one bit error every 160 msec. Note that RapidIO compliant channels must operate at bit error rates of one error in 10^{12} bits, which at 6.25 Gbaud corresponds to one bit error in 160 seconds.

The monitored BER is specified using a number of samples, the sample period, and a leak rate. For example, a BER of one error in 10^{12} bits can be specified using a sample period of 160 seconds, and a sample count of 1. It can also be specified as 160 samples, where each sample takes 1 second to collect. The leak rate in both cases is 1 – this is the allowed number of bit errors that can be seen in any one period, as defined by the sample period and number of samples. The sample period is the same for all lanes on a device.

If more samples are taken for shorter periods, it allows the same sample period to support different BERs. For example, suppose one link is known to be noisy, so a 10^9 BER is tolerable. A different, higher performance link on the same switch must support a BER of 10^{12}. In this case, a sample period of 2 seconds could be used. The noisy link would use a sample count of 1, and a leak rate of 13. The higher performance link would use a sample count of 160, and a leak rate of 1.

Note that sample periods, sample counts, and leak rates must be set so that the hardware BER counter does not overflow. The maximum value of the hardware BER counter for 1x ports is 255, while for 2x and 4x ports the hardware BER counter value is 15. In the example above, if both the noisy and higher performance links were 1x ports, the sample period could be set to 4 seconds. The sample count for the noisy link is set to 1, and the leak rate to 25. If the noisy port was a 2x port, it is not possible for the high BER to be detected with these settings, since the leak rate would always exceed the maximum BER that could be detected.

It is statistically possible for the BER to be 10^{12}, but to receive more than one bit error in 160 seconds on a 6.25 Gbaud lane. For this reason, monitoring uses a “leaky bucket” approach to control the degree of statistical certainty (confidence interval) that determines the target BER has been exceeded. Note that greater certainty requires additional time, meaning that the system will run for a longer period with a BER that exceeds the target rate. For example, suppose the target BER is 1 in 10^{12}. The BER leak parameter should be set to 1 in order to discard one bit error every 160 seconds. If the BER count ever reaches or exceeds 3 bit errors, then it is reasonably certain that the BER has exceeded 1 in 10^{12}.

When monitoring determines that the BER exceeded the target rate, the DFE coefficients can be adjusted. IDT recommends that only DFE coefficients 2, 3, and 4 should be adjusted by a maximum value of 1. The expectation is only small amounts of adjustment are required to achieve the target bit error rate. The risk of link re-initialization increases when larger changes in DFE parameters are attempted, For this reason, IDT recommends that coefficients 0 and 1 are never adjusted, as these can have large impacts on BER even with small changes.

It is also possible for other software to adjust the DFE coefficients and the link partners transmission parameters when monitoring fails. The Applications configuration file must not specify adjustment parameters for lanes that are adjusted by other software.

Command Line Options

The App is invoked using the “adfed” command which has following parameters:

- help (or –h) – Displays help information for the command line
- no-daemon – Disables execution as a daemon, and forces operations as a regular user mode process
- debug – Enables sending debug information to the system log. By default, filtering is set to enable messages with syslog priority of LOG_INFO or higher.
- time (or –t) <poll_time> – sysfs polling period in seconds. This parameter defines how often the App scans for the presence of specified devices. The default setting for this parameter is 1 sec.
- pidfile <file_name> – Overrides default name for PID file used by daemon. The default PID file name is “/var/run/adfed.pid”.

Environment Variables

The environmental variables include the following:

- ADFE_CONFIG_FILE – Overrides full path and file name for the application configuration file. By default, the App uses “/etc/adfe.conf” as its configuration file.
- RIO_SYSFS_PATH – Overrides path to device directory in sysfs. This is used to perform access to device registers using sysfs “config” file in the device directory. By default, this path is set to “/sys/bus/rapidio/devices/”.

Configuration File Syntax

The application configuration file consists of parameter line(s) and one or more device parameter blocks. The configuration file format uses keywords, curly braces, and parameters. Keywords are entered as shown in the following syntax example. Note that keyword detection is case sensitive. Curly braces are used to clearly identify blocks of parameters for switches and ports. Parameters are identified by enclosing them in “<> and “>” symbols. Parameters are values that control the operation of the ADFE Application.

```
# Example Comment
switches = <numswitches>
/<DEVICE_NAME> {
  <BER_Time> <numPorts>
  port <portNum> <firstLane> <numLanes> {
    Mon <numSamps> <BERleak> <BERadj> <Adj <adjNumSamps> <adjAbortBER>
    <adjIter> <numCoeffs <<coeffNum> <Delta>>…>>…
  }…
}
```

For more information, see the example configuration file for compliant BER and example adjustment parameters provided as part of the application package.

The keyword “#” identifies the start of a comment. All text after a “#” is ignored by the Application. A “#” can occur anywhere in text that specifies parameters.

The keywords “switches” and “=” must occur as the first non-commented text within the configuration file. The <numswitches> parameter defines the number of switches/device parameter blocks that follow The number of device parameter blocks must be equal to the <numswitches> parameter.

A first line of a device parameter block begins with the "/" keyword, immediately followed by <DEVICE_NAME> and terminated by an opening bracket "{". The first character of the device name must appear immediately after the "/" keyword – no white space is allowed. The <DEVICE_NAME> must be specified according to sysfs format existing for selected access method. <DEVICE_NAME> must be formed in accordance with the RIO_SYSFS_PATH environment variable and can have several variants as shown below. Please note that the leading "/" will be truncated from the device name.

Example 1: Using RapidIO Access Only

RIO_SYSFS_PATH is not set. Default path is: /sys/bus/rapidio/devices/

Device names can be specified as: "00:s:1200", "00:s: 1400", "01:s:0200", etc. In this case, device names are in the format defined for the RapidIO subsystem.

Example 2: Using I2C Access Only

RIO_SYSFS_PATH must to be set to "/sys/bus/i2c/devices/i2c-0/"

Valid device names should be specified as: "0-006a" or any other device name existing in the I2C sysfs structure.

Example 3: Mixed RapidIO and I2C Access

In this case parameter blocks must specify full path to each device directory. To allow it:

RIO_SYSFS_PATH must to be set to "/" (root directory) and device names defined relatively to it:

"sys/bus/rapidio/devices/00:s:1200" for RapidIO, and

"sys/bus/i2c/devices/i2c-0/0-006a" for I2C.

Each device parameter block must be terminated with a "}".

The terms in the switch parameter block are defined as follows:

- <BER_Time> – This is the time in milliseconds to wait between checks of the error counters for all ports/lanes on this device. The minimum value is 160 milliseconds. This is the interval required to catch an individual bit error on a 6.25 Gbaud link with a BER of 1 in 10EXP-9. Since the BER counter can track up to 15 errors, IDT recommends setting this value to 1000 (one second) for most systems.
- <numPorts> – This is the number of ports that will be monitored on this switch. Ports are checked in the order that they are defined. If <numPorts> is 0 then this device should not be part of the device parameter block. There must be exactly <numPorts> port parameter blocks within each device parameter block.

The "port" keyword defines the start of a port parameter block:

- <portNum> – This is the port number of the port that will be Monitored by the App.
- <firstLane> – This is the first lane number to be monitored within the port number. This parameter allows lanes to be monitored individually.
- <numLanes> – This is the number of lanes on the port that will be Monitored by the App. There must be <numLanes> lane parameter blocks within the port parameter block. The lanes monitored are numbered sequentially after <firstLane>. An open curly brace must follow <numLanes>.

The "Mon" keyword identifies the start of each lane parameter block as follows:

- <numSamps> – This is the number of <BER_Time> periods that must pass before decrementing the error counter for this lane. <BER_Time> and <numSamps> determine the BER that is being checked. For more information, see ADFE Application Configuration File Recommendations.
- <BERleak> – The amount by which the error counter for this lane is reduced when <numSamps> periods has expired. This value determines the allowable BER for a lane/port.

- <BERadj> – If the error counter ever reaches this number of errors, then monitoring of this lane is concluded. Typically, this value will be 2 or more.

If the optional “adj” keyword is entered after the <BERadj> parameter, then Adjustment is attempted when monitoring detects a BER that exceeds the target BER.

- <adjNumSamps> – This is the number of <waitTime> periods required to determine the BER when adjusting. To relate this value to BER, see <numSamps>.
- <adjAbortBER> – If the measured BER meets or exceeds this value for adjNumSamps or fewer intervals, abort adjustment and monitoring.
- <numCoeffs> – The number of coefficients that will be Adjusted. numCoeffs value ranges from 1 to 5. There must be <numCoeff> pairs of <coeffNum> <Delta>.
- <coeffNum> – The number of the coefficient, from 0 to 4, which is to be Adjusted.
- <Delta> – The amount by which the coefficient can be adjusted. Valid values range from 1 to 4.

Application Design Description

The App consists of one main process and worker processes. It uses the gcclib as well as the IDT S-RIO API library – no other library files are necessary. The main process is responsible for initialization of the worker processes. There is one worker process for each active device specified in the configuration file. If one or more specified devices are not available (no corresponding sysfs config files) the main process periodically checks for their presence to create a worker process in case of a device insertion.

The main process performs the following steps on initialization:

1. Performs command line parameters parsing (if any)
2. Initializes the IDT Switch API
   - This includes binding default register read and write routines that use sysfs file access routines.
   - If the IDT Switch API initialization fails, the daemon creates a system log and kills itself.
   - If different read/write routines are required, they can be bound in at this time.
3. Initializes message logging using syslog facility
4. Reads the configuration file
   - If the configuration file does not exist, exit with error.
   - If there are syntax errors in the configuration file, skip malformed device parameter blocks.
   - Creates a list of devices and the configuration information for each device, including a list of ports and lanes to be managed
5. Transforms itself to a daemon process (with corresponding PID file)
6. Walks through the device list and creates a separate worker process for each active device
7. Monitors for new devices
   - Periodically checks to see if the specified managed device file now exists.
   - If the managed device file exists, starts the worker process for that device.
8. If the main process is terminated all child worker processes are terminated as well.
9. If an individual worker process exits with success status, its corresponding device is no longer monitored by the main process (assuming that this is a user controlled stop for that worker process).
10. If an individual worker process exits with error status, its corresponding device is returned for monitoring assuming that error exit was caused by device removal.
11. If all specified switches have their worker processes running, the main process pauses itself.

12. If the main process receives a SIGUSER1, the following occurs:
    a. The main process reports the status of which workers are active to the system log file with INFO priority level.
    b. The main process signals all active workers to display their status to the system log file

The worker process design is as follows:

1. Create an IDT Switch API device handle for the managed device.
   — If the device handle cannot be created, the worker process generates an entry in the syslog file and kills itself. It does not signal the daemon.
2. Allocate and initialize an IDT Switch API idt_sc_ber_mon_update_t data structure for each port/lane, based on the information in the configuration file parsed by the daemon.
3. Configure the BER counters for each port/lane.
   — If the BER counters cannot be configured, the worker process generates an entry in the syslog file, and exits with error status.
4. At the rate specified by the configuration file, the worker process performs the following:
   a. Reads the BER counters for a port
   b. Calls the IDT Switch API routine idt_sc_ber_mon_update with the idt_sc_ber_mon_update_t data structures for that port
   c. Creates syslog entries based on any state changes in the idt_sc_ber_mon_update_t data structure
      – If adjustment has started
      – If adjustment has completed
      – If adjustment has aborted
      – If monitoring has aborted
      – If an error code is returned
      – If the debug level requires information for each update.
      – If an error code has been returned, the worker process exits with error status.
5. If the daemon has been killed, it signals the worker processes that they should die gracefully.
   — The worker process gracefully aborts monitoring/adjustment by restoring the last known good DFE coefficients if adjustment was underway.
   — The worker process then exits with success status. The daemon is not signaled.
6. If a port is uninitialized, the worker will periodically check to see if the port becomes initialized.
   — If the port becomes initialized, monitoring starts.
   — If the port becomes uninitialized, monitoring/adjustment is aborted.
   — Both of the transitions above cause the worker process to display the state change in the syslog file.
7. If the worker process receives a SIGUSER1, the following occurs:
   a. The worker process displays the status of each lane to the system log file using an INFO priority level.
   b. The status of each lane consists of whether or not the port is initialized. If the port is initialized, the current state of the monitoring/adjustment algorithm is also displayed.