Fed9UVmeDevice Documentation

Gareth Rogers

Contents

The Fed9U Namespace

The Front End Driver (FED) is a 9U 400mm VME64x card designed for reading out the CMS silicon tracker signals transmitted by the APV25 analogue pipeline ASICs. The signals are transmitted to each FED via 96 optical fibres at a total input rate corresponding to 3 Gbytes/s. The FED digitizes the signals and processes the data digitally by applying algorithms for pedestal and common mode noise subtraction. The FED home page can be found here [1].

The Fed9U software controls the FED registers and the data readout from the FED buffers. It is used to configure the run parameters and readout the events, providing format checking of events. It is also capable of monitoring the temperature of the hottest regions of the FED PCB and the voltages supplied to the FED components.

All Fed9U software is contained within the Fed9U namespace. It is structured into two layers, Fed9UUtils and the Fed9UDevice each with its own header and dynamically linked library. Fed9UUtils is a collection of Utility classes that perform tasks that are used through out the Fed9U namespace. Fed9UDevice contains the libraries and header file for accessing the FED hardware and is dependent upon Fed9UUtils. There is also a group of standalone executables that can be used to perform a variety of testing and debugging of a FED. They allow the FED to be installed as a stand alone piece of hardware, independent from any other component of the Tracker readout chain.

The Fed9UVmeDevice Class

Overview

The aim of Fed9UVmeDevice was to provide a simple interface to the FED registers, which provided a set of methods that can be used to configure and control the FED with the minimum amount of effort on a Users part.

Fed9UVmeDevice accesses the FED through the class Fed9UVmeBase. This provides access to the FED registers that is similar in layout to the FED firmware. Fed9UVmeDevice provides an abstraction from this layout to an interface which is not constrained by the FED firmware structure.

The Abstract Base Class (ABC) Fed9UABC, provided by the Fed9UUtils library, is the base class from which Fed9UVmeDevice is derived. It provides the interface which is used by Fed9UVmeDevice for accessing the FED. The inherited interface provides the User to access the individual FED register at their lowest level of granularity. Fed9UVmeDevice extends on this interface to provide methods for the configuration and validation of the whole FED in just a few method calls.

Using Fed9UVmeDevice

A comprehensive list and description of all the methods provided by Fed9UVmeDevice class (as well as all the classes in the Fed9U namespace) are available on the FED savannah project site [2]. It can also be found by searching for fed9u on the CERN savannah home page. The following section does not assume any knowledge of the Fed9UVmeDevice methods or Fed9U namespace classes, but it is recommended that the reader familiarise themselves with the Fed9U namespace, especially the Fed9VmeDevice class.

Instantiating a Fed9UVmeDevice Object

There is only one public Fed9UVmeDevice constructor provided, both the copy and assignment constructors are declared private and are not implemented. The constructor takes a single argument, a constant reference to a Fed9UDescription object, of which it makes a local copy. This class is also derived from Fed9UABC and it is used to store the FED settings in software. Fed9UVmeDevice is continually updating its local Fed9UDescription and so the User must update their copy in order to ensure it reflects the current FED values.

A default constructed Fed9UDescription contains a set of valid settings for the FED, which would allow the User to immediately start using a FED to take data in the default data acquisition (DAQ) mode. There are two settings that are unique to a FED which the User must ensure are configured correctly before instantiating the Fed9UVmeDevice object. These are the slot number and the location of the Fed9UHalAddress table. The slot number represents the slot in the crate that contains the relevant FED. The Fed9UHalAddress table contains the addresses of the FED registers and is required if Fed9UVmeBase is to function correctly, see the Fed9UVmeBase documentation for more details. If the pc is controlling multiple FED crates then additionally the crate number must be correctly configured, however in a single crate environment the default value is fine.

Setting up a FED

A FED can be safely used to readout sensible data with a Fed9UVmeDevice object constructed from a default constructed description, it is recommended that the User configure the description object first to ensure that the FED runs as desired.

The simplest way to configure a FED using a Fed9UVmeDevice object is with the initialisation methods provided by Fed9UVmeDevice. They can be used to setup up the whole FED using a few method calls. Individual settings on the FED can be tweaked using the Fed9UVmeDevice methods provided to access the individual registers if required.

There are three different initialisation methods provided for configuring the FED init, initStrips and initAuxiliaryChips. init is called when there are major changes require to the run parameters, or a new Fed9UDescription object is loaded. It will setup all the FPGA parameters and the TTCrx device. The initStrips is only needed in processed raw or zero suppressed data acquisition modes. It sets up the strip parameters pedestals, cluster thresholds and the valid strip settings. initAuxiliaryChips sets up the temperature and voltage monitoring parameters. It will be needed only be need infrequently.

In the event that the FED is incorrectly configured it is possible to return the FED settings to their hardware default values by performing a FED reset. It will clear all the FED registers, except the TTCrx chip, the temperature monitors, pedestals, disabled strips, high and low cluster thresholds and the clock source. There are also FE and BE FPGA soft resets which clear the FPGA logic, but do not affect any of the register settings. The TTCrx, temperature monitors and voltage monitor all have there own hard resets which clear all the settings. The clock source is unaffected by resets.

Data Readout

There are two different ways that the FED can respond to incoming data. It can either respond by detecting incoming triggers or by detecting incoming frames, this is scope mode and frame finding mode respectively. When in scope mode no data processing is performed and both the APV tick mark and the frame are present in the event data. When in frame finding mode only the frame containing the strip data are present event data. There are three different frame finding modes virgin raw data mode where no data processing is performed; processed raw data mode where strip reordering and pedestal subtraction is performed; and zero suppression mode where strip reordering, pedestal subtraction, common mode median subtraction and cluster finding is performed. Scope mode, virgin raw, processed raw and zero suppressed data modes make up the four data acquisition (DAQ) modes available on the FED.

Once the FED has been properly configured in the desired DAQ mode event readout can be performed. When in scope the FED will respond to triggers rather than frames and no optical input is required on the FE FPGAs. In the frame finding modes the FE FPGAs must be driven as the FED looks for the frames instead.

The author has no experience with the S-Link features, so will only describe readout via the VME. When reading via the VME unread events will be stored in the QDR memory until it has been told the VME event buffer is empty. At this point the next event is sent from the QDR to the VME event buffer. An event can be checked for using the method hasEvent(), which has three different return values, no event present, event present, and last fragment of an event. A User will typically only be concerned whether or not an event is present, in which case the return can be treated as Boolean where a false represents no event and a true an event. If there is an event present the method getCompleteEvent should be called. This must be provided with a suitable large buffer to store the event and gives the User the option of using a block transfer if their hardware supports it. Once read the data should be passed to the Fed9UEvent class, which can check the event format for errors. After getCompleteEvent has completed the FED will prepare the next event to be readout. The number of events that have been received and are left to be readout can be monitored using the method getBeEventCounterStatus.

Monitoring the FED

The FED also contains two hardware monitoring devices. There is a temperature monitors present on each FPGA and a voltage monitoring device.

There are ten temperature monitoring devices on the FED, one for each FPGA. There are two temperatures measured, the FPGA temperature and the chip temperature. Each of these can have an independent maximum temperature limit, which when exceeded will be flagged in a status register on the device. There is also a critical temperature, which if either temperature exceeds power will be cut to the FED to prevent damage to the components. This is a last resort as the FED will have to be switched on at the FED and cannot be performed via software.

There is one voltage monitoring device which monitors 2.5V, 3.3V, 5V, 12V, its core voltage and the supply voltage. Each of these can have lower and upper limits set. If the voltage is outside of this limit then it will be flagged in a status register on the device. The temperature also has upper and lower limits, which will be flagged if the temperature goes out of bounds. It is possible for the voltage monitor to measure an external temperature as well as the internal chip temperature however in the current FED design this ability is not used.

Using the EPROM

The EPROM is a 2 kilobyte area of non-volatile memory that is split into four equal sized sections. It is possible to write protect one quarter, half or all of the EPROM memory. In the final system it is expected that the write protection level will be locked in the hardware giving the User access to only certain quadrants of memory. For now however a User must set the write only level after each power up, otherwise the EPROM memory cannot be accessed. Due to the current nature of the write only level and the lack of any definition of what the EPROM memory should be used for the User should be careful that they do not over write any important data that someone else may have written and also care should be taken not to write over the FED serial number, for which a specific set/get method exists in Fed9UVmeDevice. At present the only bounds checking that takes place is that the User is writing to a

valid area of memory (i.e. inside the 2KB region) there is no software warning if a User attempts to write to a write protected area. If a write protected area is written the write will appear to have succeeded there is no warning for either the hardware or software. The only mechanism that can be used to check is by reading back from that area of memory to ensure the written data is now present.

Fed9UVmeDeviceException

ICUtils::ICException provides a set of pre-processor macros which are used to create the frame work for an exception class derived from ICUtils::ICException, which can then be customised to the meet the exception class requirements. It is not provided as part of the Fed9U namespace, but is provided as a general utility in the ICUtils namespace.

Fed9UVmeDeviceException is an exception class used to describe the various error conditions that can occur during the use of Fed9UVmeDevice. It is derived from ICUtils::ICException, which is derived from std::exception. It contains an exception code list, which can be used to identify the type of error thrown. It also contains an error string that identifies where the error was thrown from and gives some information on the error conditions that generated the event.

It is expected that Fed9UVmeDevice will only throw exceptions of type Fed9UVmeDeviceException. This cannot be relied upon though it is possible that an included class might through an exception of some other type. Fed9UVmeDevice is capable of catching ICUtils::ICExceptions, which is what included classes are expected to throw (or classes derived from this). It is advised that any classes including Fed9UVmeDevice are prepared to handle other exception types.

There are basically two different errors that will be thrown from Fed9UVmeDevice. The first is that which has been re-thrown from an included class. The other is an out of bounds error from one of the methods setting a new value to the FED.

In the case of a re-thrown error the documentation for that class should be consulted. The most common type of error encountered in this case said something like “cannot write to address 010004 with 4 bytes of data”. This typically means that the base address has been incorrectly set and there is no FED in that slot.

Out of range errors are pretty much what all errors that are thrown from Fed9UVmeDevice come down to. Those from higher level methods, such as init, are usually just out of range errors thrown from method calls within the higher level method. The boundary condition that has been exceeded is given in the error print out along with the method it occurred in. There is one other type of error that may occur which is an inconsistency error. Certain settings should be the same across all FE FPGAs and so are checked for consistency. If a FE FPGA is determined to have an inconsistent value an error is thrown. The first attempt to remedy this should be to enter the correct value for that FE FPGA. If this is unsuccessful it has probably been caused by that FE FPGA not loading and can easily be fixed performing a FED reset, which will reload the firmware. This will reset all the FED register values.

Summary

Fed9UVmeDevice is a derived class of Fed9UABC and provides access to the FED registers. It has been designed to allow the User a simple and easy to use interface that is hopefully flexible enough to meet all a Users needs. It relies on a variety of support classes that range from exception handling to classes that correspond directly to devices on the FED. It has been integrated into the wider Fed9USoftware and serves as a base class for Fed9UDevice, which provides additional higher level methods.

References

[1] FED UK - http://www.te.rl.ac.uk/esdg/cms-fed/qa_web/

[2] Fed9U savannah project page - http://savannah.cern.ch/projects/fed9usoftware/

[3] Authors email address - ggr1@aber.ac.uk