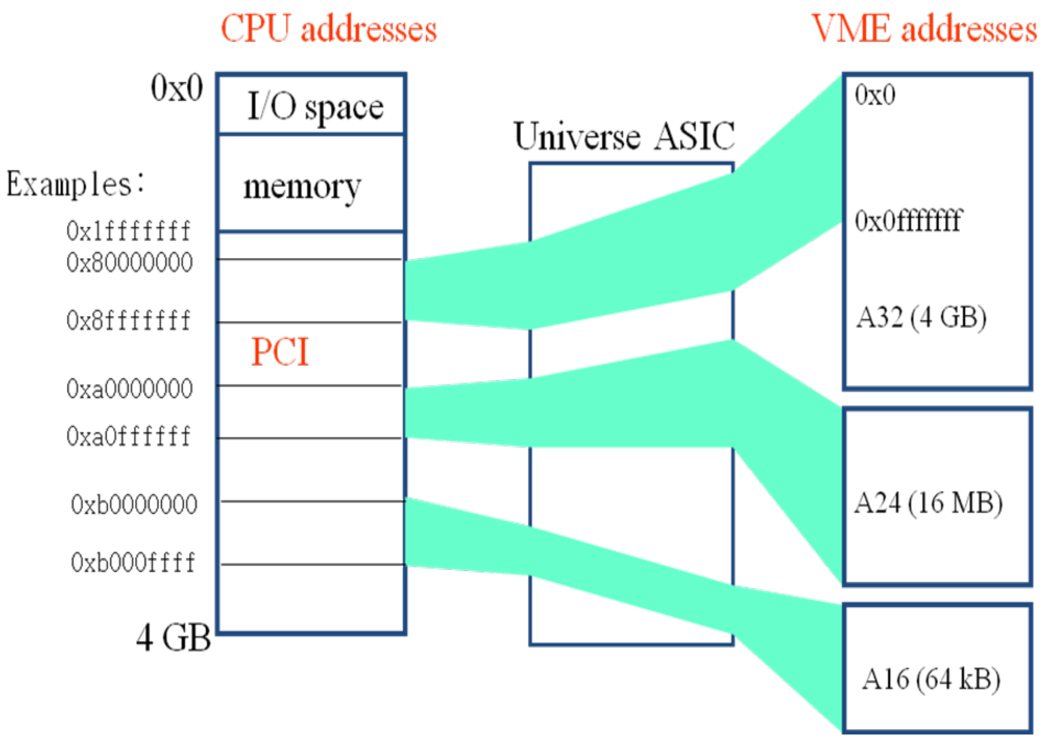
VMEbus Programming

Exercise 1

Introduction

This exercise will permit you to use the VMEbus slave as if it was a piece of memory in your PC. This will demonstrate that from the programming point of view there is very little difference between internal and external memory. The differences between the two types of memory are also emphasized.

An important aspect is that the VMEbus memory has to be mapped into the (virtual) address space of a user process before it can be accessed. This connects three busses together: CPU, PCI and VMEbus as shown in the picture below.



Exercise1, Figure 1: Mapping of the VME address space into the PC memory space.

Outline

The first part of the exercise will start with creating the appropriate mapping specific to the VMEbus access that will be used. Once this is completed, one can initiate data transfers which will be done in single cycle mode, meaning that the CPU controls the data transfer.

The second part concerns block transfers via a direct memory access (DMA) controller. This requires a different programming technique since it is not the CPU that moves the data but the DMA controller. Such DMA controllers are not VMEbus specific, as they can be found in many hardware implementations, such as network interfaces, disk controllers, USB devices, etc.

Pre-requirements:
Before starting you should try to answer these questions:

- What does the acronym A24D32 mean?
- What is endianness and how do you deal with it?
- What are the advantages of block transfers

Work plan:

1. On the work station (pcdaqschool(1/2)) log on with the DAQ school account (**daqSchool / g0ldenhorn**).
2. Execute "**ssh –Y tds-sbc-0(1/2)"** and log in with the same password
3. Run "**source setup**" and then change directory to exercise1/student
4. Copy the file "skeleton.cpp" to "solution.cpp" and start an editor session (vi, nedit) for "solution.cpp".
5. Add the missing code to "solution.cpp" to execute the VMEbus cycles listed below:
   a) Write 0x12345678 to address 0x08000000 in A32 / D32 mode. Use the "safe" cycles.
   b) Read the data back from address 0x08000000 and compare it.
   c) Write 0x87654321 to address 0x08000004 in A32 / D32 mode. Use the "fast" cycles.
   d) Read the data back from address 0x08000004 and compare it.
   e) Write a block of 1 KB to address 0x08001000 in A32 / D32 / BLT mode. You have to prepare the data in a cmem_rcc buffer.
   f) Read the data back from 0x08001000 in A32 / D64 / MBLT mode and compare it.
6. Run "**make**" to compile the application.
7. Run "**solution**" and catch the VMEbus transfers on the display module.

Additionally, if you have extra time you may try the following:

1. Go back to the API of the vme_rcc library and add additional transfer modes (e.g. D8 and D16 cycles).
2. Play with the "cctscope" and "scanvme" utilities to familiarize yourself more with the VMEbus interface H/W .

Good practice:
Do not forget to undo all initialization steps (return memory, close libraries) before you exit from an application.

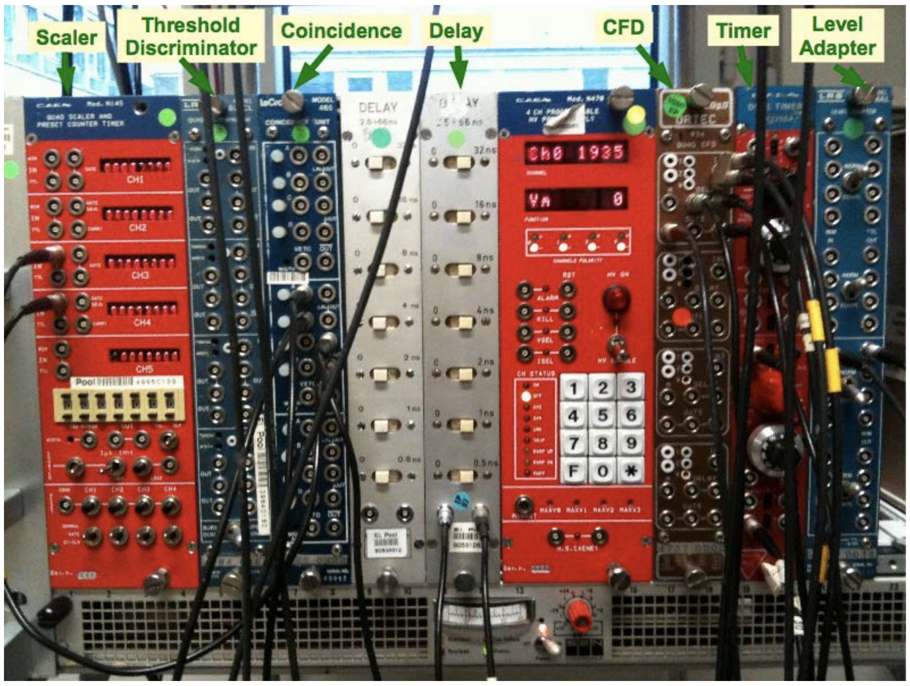
The Trigger

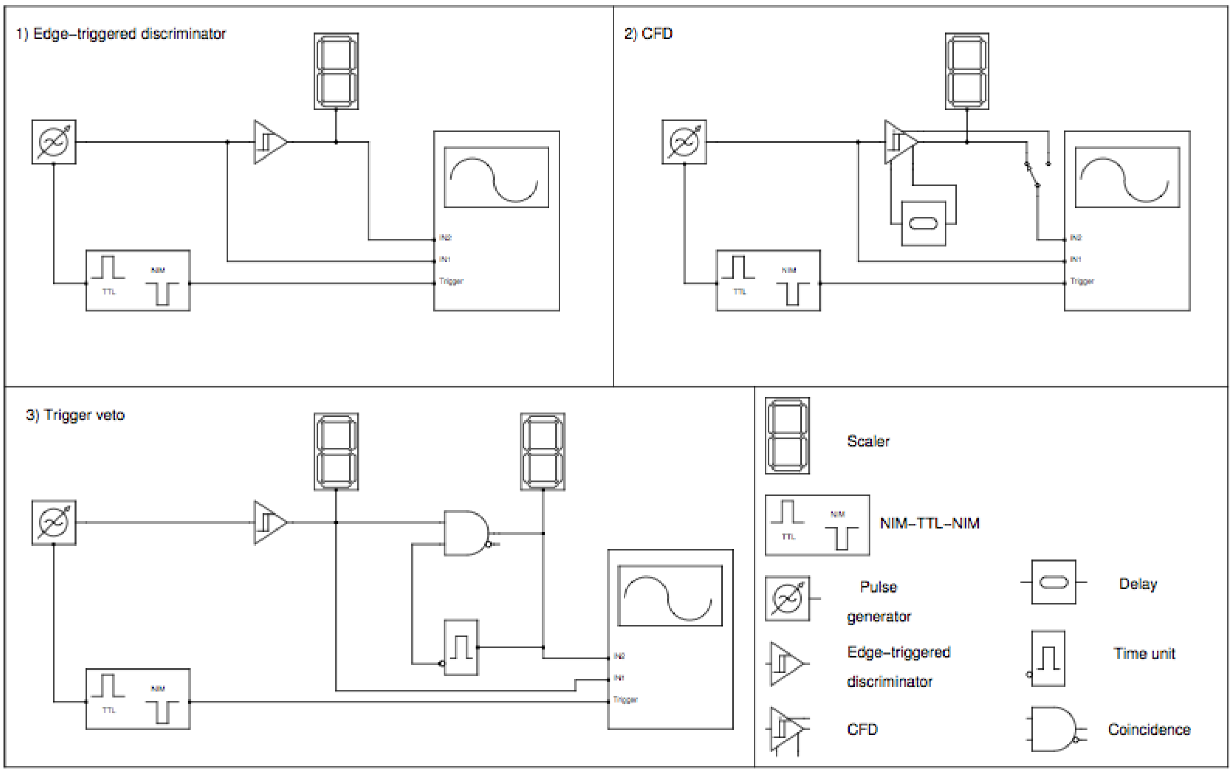
Exercise 2

Introduction:
This exercise will introduce the students to basic trigger systems implemented in NIM logic.

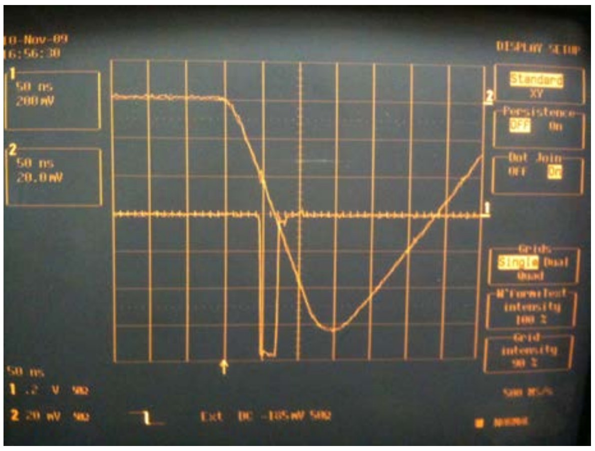
Pre-requirements:
The introduction of the trigger lecture is essential.

Work plan:



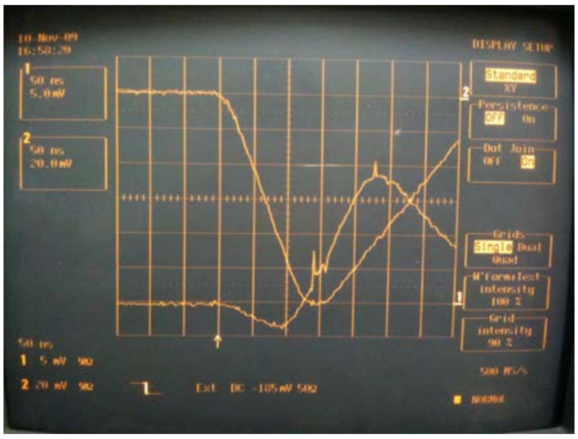


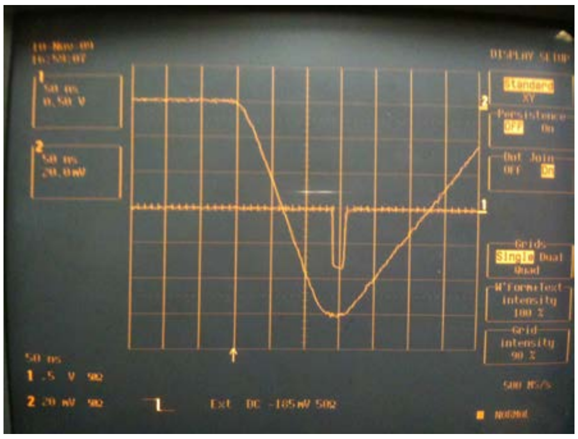
- **Part 1a: Threshold Discriminator**
  - The signal generator will be pre-configured to provide a triangular pulse (T=300us, leading=100ns, trailing=200ns, width=200ns, offset=0, amplitude=-100mV)
  - First the students should look the signal (MAIN OUT) in the oscilloscope (CH1), using the SYNC OUT of the generator as a oscilloscope trigger (EXT)
    - The SYNCOUT is TTL signal. Transform it into a NIM signal using the dedicated level-adapter module
  - Split the generator output signal and connect one branch to the input of the threshold discriminator. The other branch should be properly terminated on the oscilloscope side (1MΩ)
  - Connect one output signal of the discriminator to the scaler module and a second output to the oscilloscope (CH2)
  - Check the threshold on the discriminator with a Voltmeter (x10 output)
  - Change the threshold with a screw driver and
    - observe the behavior of the output signal on the scope
    - observe the rate on the scaler * Can you relate them to the threshold values?


Threshold Discriminator Output

- **Part 1b: Threshold Discriminator - Jitter**
  - Using the above setup, set the threshold to 30mV and change the amplitude of the input signal. Which is the effect on the discriminated signal? How does it affect a timing measurement?
  - Measure the discriminated signal delay with respect to the reference as a function of the amplitude of the input signal (-50, -100, -150, -200 mV). Fill up Table 1 with your numbers.

- **Part 2: Constant Fraction Discriminator**
  - Use the previous signal as input of the constant fraction discriminator.
  - Connect to the oscilloscope the input signal (CH1)
  - Setup the CFD parameters:
    - threshold (T) --> 27 mV - Measure with Voltmeter (x10 output)
    - walk (Z) --> 2mV - Measure with Voltmeter
    - delay (D) --> 80 ns - Set with delay module + 2x10ns cables
  - Connect the monitor output (M) of the CFD to the oscilloscope (CH2). Can you recognize the CFD technique? Which is the effect of varying D?
  - Connect to the oscilloscope the discriminated output of the CFD (CH2)
  - Change the amplitude of the input signal. What happen to the output of the discriminator?
  - Measure the discriminated signal delay with respect to the reference as a function of the amplitude of the input signal (-50, -100, -150, -200 mV). Fill up Table 1 with your numbers. Compare the results with the previous measurements.


CF Discriminator Monitor Output


CF Discriminator Output

- **Optional**

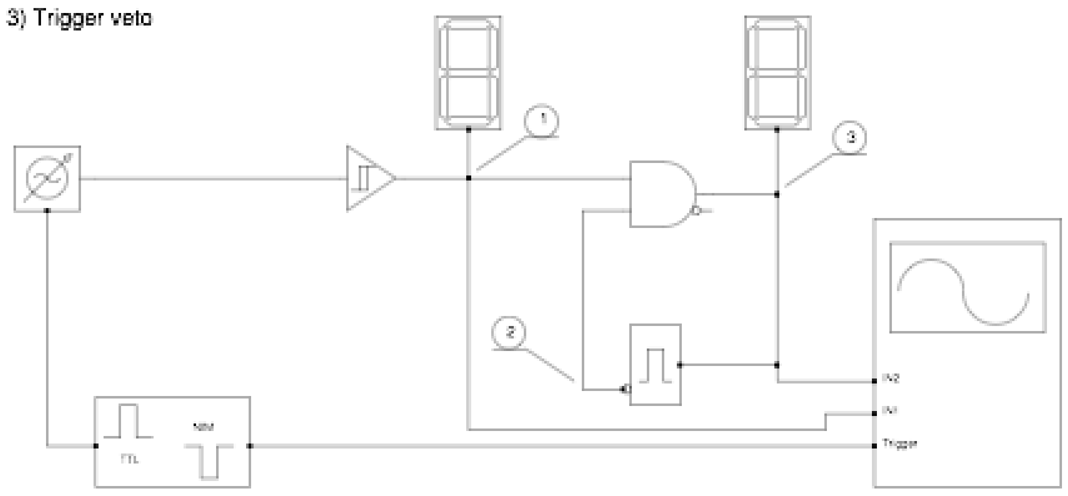
Can you make the CFD behave like a normal threshold discriminator? Which configuration parameters have to be touched?

Discriminators Results:

Table 1: Delay on the discriminated signal with respect to reference

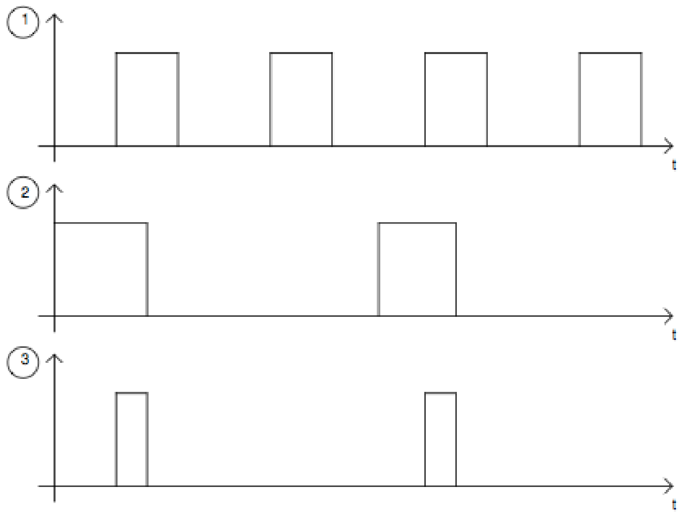
input signal Amplitude (mV)	Threshold D (ns)	Constant Fraction D (ns)
50		
100		
150		
200		

- **Part 3: Trigger veto or dead-time**
  - Configure one stage of a dual timer module to generate signals with 10ms width
  - Connect the output of discriminator and the negated output (OUTbar) of the timer (the "veto") to a coincidence unit.
  - Connect another output of the discriminator and one output of the coincidence to two scaler ports
  - The output of the coincidence has to drive the timer module (START)
  - Compare the counting rates of the scalers. How do they relate with the timer setting?



Timing diagram:

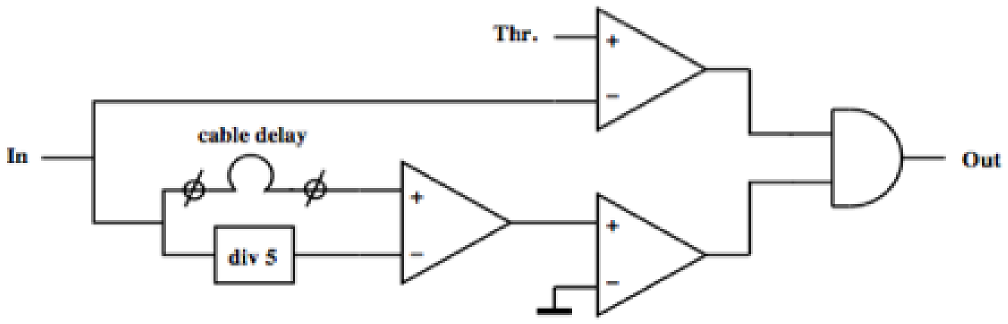
1. Discriminator output
2. Veto
3. Coincidence output



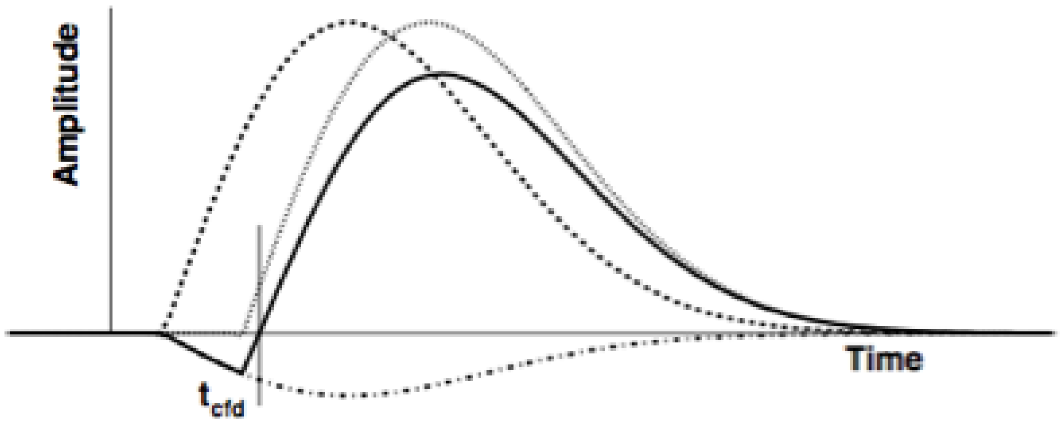
- **Optional**

Can you explain the behaviors observed disconnection either one or the other input of the coincidence unit?

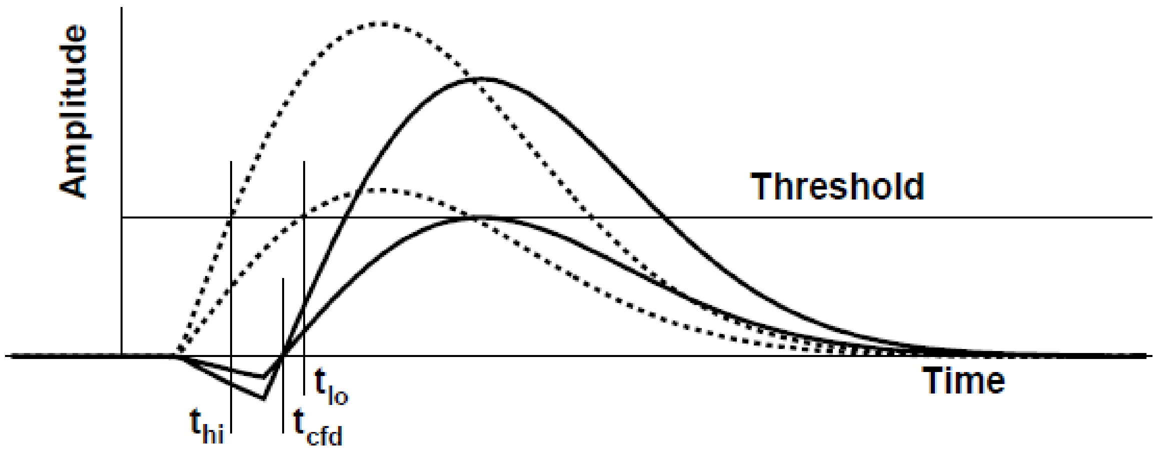
Constant Fraction Discriminator:



The above schema shows the functional diagram of a CFD. The input signal is treated in two different discrimination branches, whose results are then merged by the final AND gate. The top branch is a standard threshold discriminator, where the input signal is compared against a (configurable) threshold *Thr*.
The bottom branch implements instead the constant fraction technique. Technically, the input signal is split: one copy is delayed, while the other is attenuated by a factor 5. The two copies are then subtracted and the final result is compared with a threshold of (close to) zero. In fact, the zero-crossing time of the resulting signal in nearly independent from the input signal leading edge steepness (i.e. the source of time jitter in a standard threshold discriminator).



The above figure shows in detail the signals in the bottom branch of the CFD. The input pulse (dashed curve) is delayed (dotted) and added to an attenuated inverted pulse (dash-dot) yielding a bipolar pulse (solid curve). The output of the bottom branch fires when the bipolar pulse changes polarity which is indicated by time t_{cfd}. From a practical point of view, a small threshold, as close as possible, is actually used in the final comparator of the bottom branch. This is needed to avoid fake signals possibly caused by the noise. Such a small threshold il normally called *walk* (Z).
In order to complete the CFD description, the merging of the top and bottom branch signals has to be considered, with the help of the following figure.



In the top branch, the threshold discriminator fires at time t_{hi}, that depends on pulse leading edge characteristics. The bottom branch instead fires at a time t_{cfd}, as discussed above, which is almost constant. Due to the delay introduce in the bottom branch, normally $t_{cfd} > t_{hi}$. Therefore, the overall CFD, defined as the signal generated by the final AND gate, will fires at t_{cfd}, achieving both our requirements:

3. Only select signal above a given amplitude *Thr*
4. Provide an output trigger whose timing is independent from input signal amplitude

As can be seen in the above figure, the CFD operating principle is not retained for all the possible combinations of configured delay, threshold and input signal amplitude. As the top branch timing depends on the signal amplitude, a small enough signal can make it fire at a time $t_{lo} > t_{cfd}$. In this case the CFD will behave like a normal threshold discriminator, as the output AND gate will be driven by t_{lo}.

Detector and Trigger: Scintillators, trigger logic and input to readout modules (ADC & TDC)

Exercise 3

Introduction
This exercise consists in building the trigger logic and the input signals to the VMEbus readout modules for a detector using the experience with NIM electronics acquired in exercise #2. The detector comprises two scintillation counters detecting cosmic rays (muons). A schematic diagram of a scintillation counter is shown in
Figure 1. When a charged particle traverses the scintillator, it excites the atoms of the scintillator material and causes light (photons) to be emitted. Through a light guide the photons are transmitted directly or indirectly via multiple reflections to the surface of a photomultiplier (PM), the photocathode, where the photons are converted to electrons. The PM multiplies the electrons resulting in a current signal that is used as an input to an electronics system. The PM is shielded by an iron and mu metal tube against magnetic fields (of the Earth). The scintillator and light guide are wrapped in black tape to avoid interference with external light. The scintillation counter setup is shown in Figure 2. The NIM modules used to build the trigger and the input to the readout system and provide the high voltage is shown in Figure 3.

Outline:
The aim of the exercise is to get an understanding of the detector and trigger logic used in Exercise 4. The signals from two scintillation counters are analyzed using an oscilloscope and transformed into logic NIM signals that allow building a trigger based on a coincidence between the signals. The coincidence rate i.e. the rate of cosmic muons is counted using a scaler and the charge content of the scintillator signals is measured on the oscilloscope. In addition the inputs to the readout modules (QDC and TDC) are set up.
A schematic diagram of the full trigger and readout electronics is shown in Figure 4.

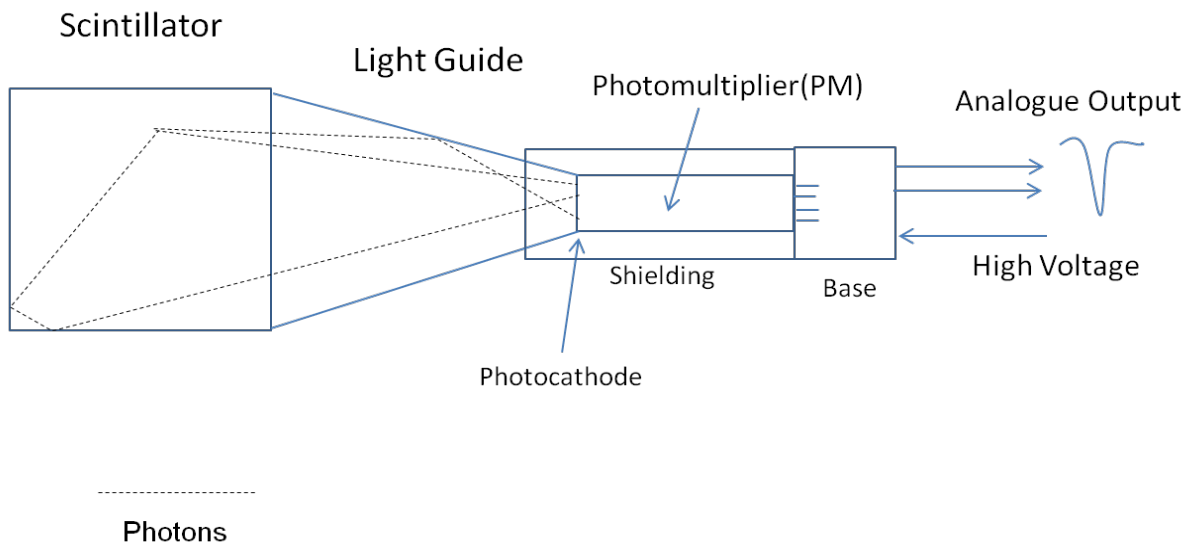


Figure 1. Schematic diagram of a scintillation counter.

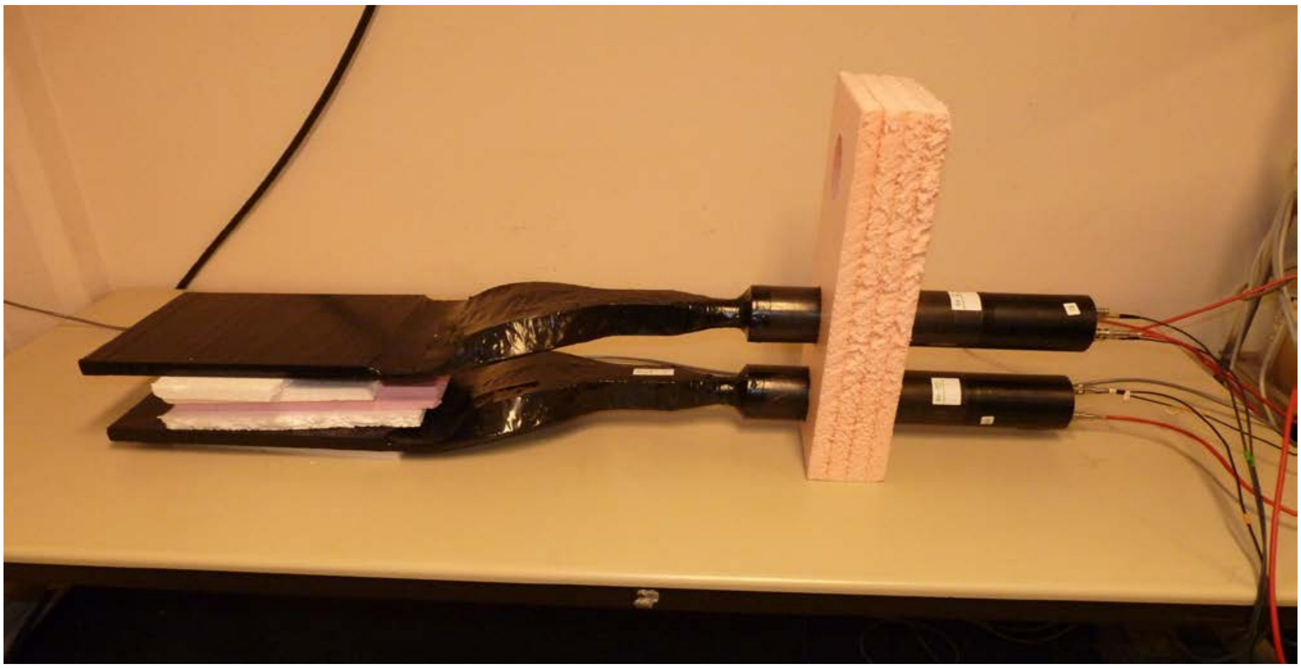


Figure 2. Scintillation counter setup

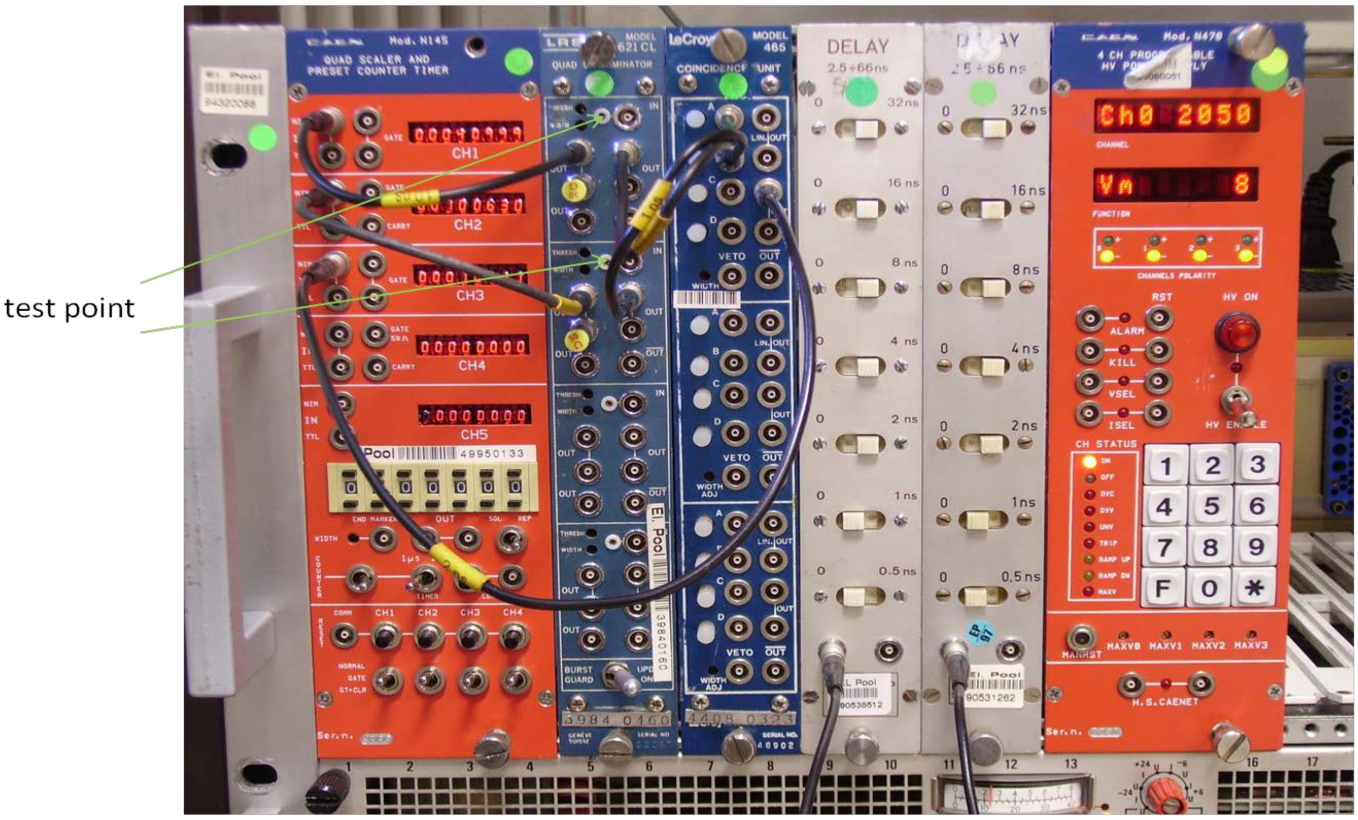


Figure 3. NIM trigger electronics. From left to right: scaler (counter), discriminator, coincidence unit, delay modules and high voltage power supply.

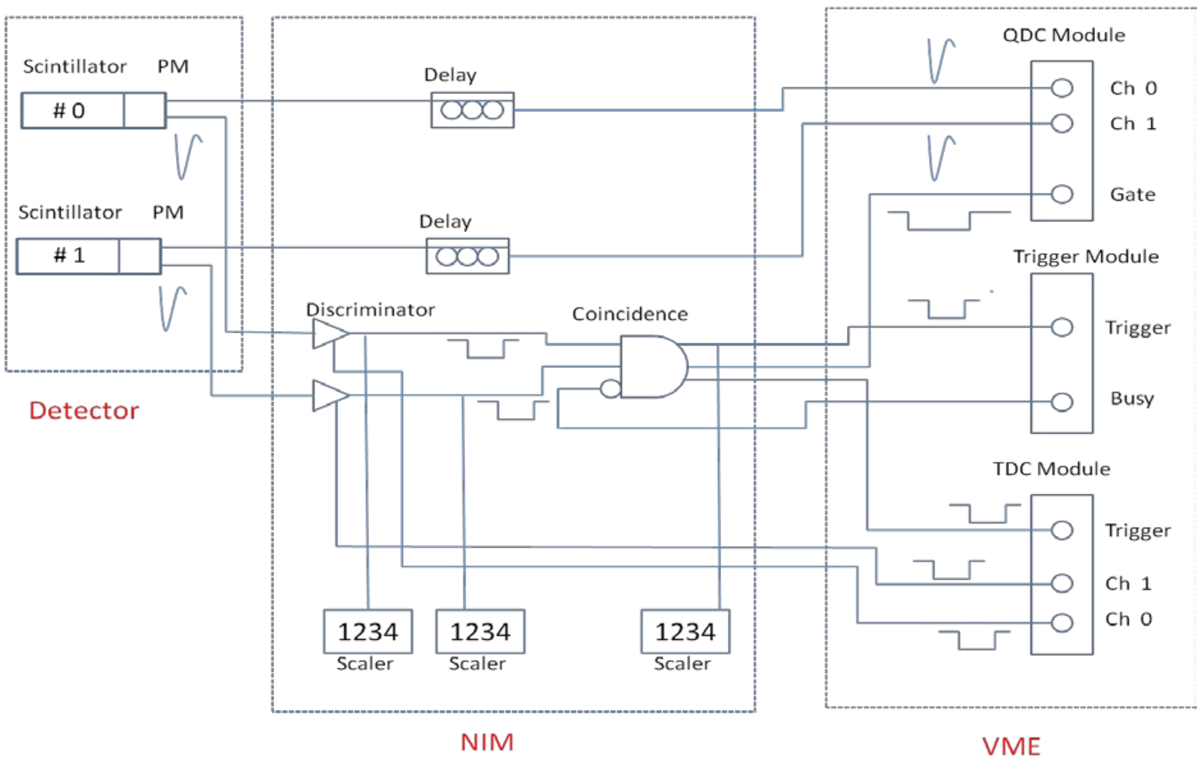


Figure 4. Diagram of the electronics for the detector, trigger and readout of the scintillator counter setup.

Work plan:

Note: whenever there are two parallel outputs from a (NIM) module one needs to make sure that they are both cabled, i.e. either terminated with 50 Ohm or connected to another unit. This ensures that the pulses have the correct NIM voltage levels: 0 and -0.8 Volts.

1. Install the scintillation counters close to each other with maximum overlap between the scintillator areas.
2. Check that the scintillator photomultiplier bases are connected to the N470 NIM high voltage supply.
3. Switch ON the NIM crate.
4. Connect an output from scintillator 0 (the upper one) to an oscilloscope (10ns LEMO), terminate the other output with 50 Ohm.
5. Set the nominal high voltage on scintillator 0 using channel 0 of the N470 HV supply. The voltage is marked on the label glued onto the base. Refer to 0 at the end of this exercise for a short guide to using the N470 HV supply.
6. Look at the signal on the oscilloscope (volts/div ~ 50 mV, time/div ~ 20ns). What is the maximum voltage of the signal?
7. Connect the cable to the input of the first channel of the discriminator.
8. Connect an output to the oscilloscope (0.5 Volts, 50 ns) and adjust the pulse width to around 100 ns using a small screwdriver.(terminate the other output with 50 Ohm), see Figure 3.
9. Connect the output to the first channel of the NIM scaler (N415) using a short LEMO cable (1ns).
10. Set the discriminator threshold to 50 mV: adjust the voltage on the test point using a DC voltmeter and a small screwdriver, see Figure 3. The voltage is 10 times the threshold value

i.e. the voltage should be around 0.5 Volts. This step may require teamwork.

11. What is the scaler rate?
12. Vary the threshold around 50 mV and check the variations in scaler rate.
13. Repeat points 4 to 11 above for scintillator #1(the lower one), connecting this scintillator in addition to the one already connected.
14. Given the scaler rates measured above, what is the probability of random (unphysical) coincidences between pulses from the two scintillators?
15. Connect an output from each of the two discriminator channels to the oscilloscope and check that they have a timing overlap i.e. are coincident.
16. Connect the cables from the discriminators to the first inputs of the coincidence unit (LeCroy 465) using short LEMO cables (1ns).
17. Connect an output from the coincidence unit to a scaler input. What is the rate? Given that the rate of cosmic muons is about 100 per second per square meter, does the rate make sense?
18. Connect an output of the coincidence unit to channel 1 of the oscilloscope.
19. Connect the (other) analogue output from scintillator 0 to a delay unit (LEMO 10ns) and the output of the delay unit to channel 2 of the oscilloscope.
20. Using channel 1 as a trigger, observe the analogue signal on channel 2. Channel 2 will then show the scintillator signals for the cosmic muons. Assuming that the signal is triangular, what is the charge of the signal? See Figure 5.
21. Adjust the delay unit such that the analogue signal falls within the NIM pulse from the coincidence unit: inputs to the charge to digital converter (QDC) in Exercise 4 are now ready (analogue signal and gate).
22. Repeat point 21 for scintillator 1.
23. Connect a cable from the first discriminator to channel 2 of the oscilloscope and check the timing with respect to the output from the coincidence (channel 1). The signal from the discriminator should precede the coincidence. Similarly for the second discriminator. The inputs to the time to digital converter (TDC) in Exercise 4 are now prepared (trigger and timing signals).
24. The signals from the discriminators are sometimes about twice as long as expected. What could the reason be?

Appendix 1: Short User's Guide to the CAEN N470 High Voltage Supply

This is a short list of the most common operations for the N470 High Voltage Supply used in Exercises 3 and 4. The manual can be found at http://www.caen.it/nuclear/product.php?mod=N470#

To select a channel: F0*(channel number)* e.g. F0*0*
To set the High Voltage on the selected channel: F1*(type value)* e.g. F1*2000*
To read the voltage on the selected channel: F6*
To read the current on the selected channel: F7*
To turn the selected channel ON: F10*

Notes:
The maximum voltage on the channels has been set to around 2300 Volts (on the potentiometers). These can be checked via F13*. The current limits have been set to 2mA (via F2).

Appendix 2: Charge of scintillation counter current pulse

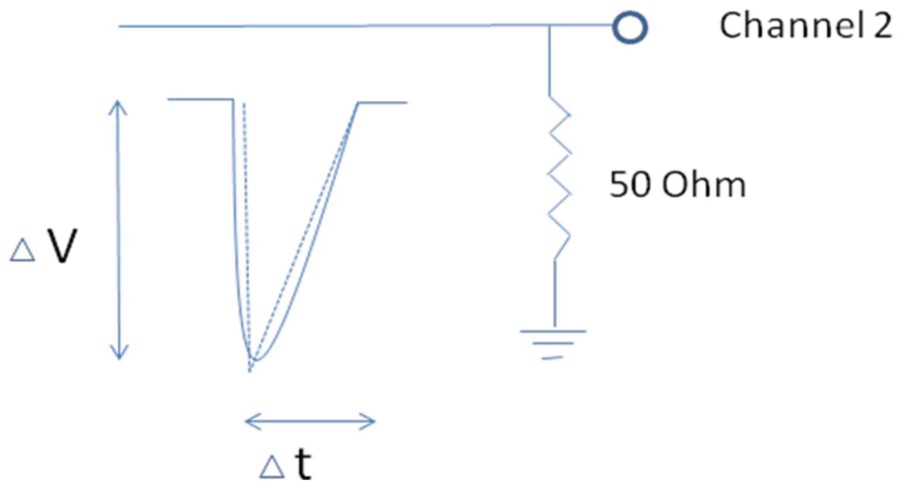


Figure 5. Input to the oscilloscope from a scintillation counter.

A small physics experiment: detector, trigger and data acquisition

Exercise 4

Introduction

This exercise comprises all the components of a typical experiment in high energy physics: beam, detector, trigger and data acquisition. The “beam” is provided by cosmic rays(muons) and the detector consists of a pair of scintillation counters, see Figure 2 in Exercise #3. The trigger logic, built in NIM electronics, forms a coincidence between the signals from the scintillation counters which indicates that a muon has traversed the detector, see Figure 3 in exercise #3. A data acquisition system based on VMEbus is used to record the pulse heights from the scintillation counters and measure the time of flight of the muon. The VMEbus crate is shown in Figure 6 and the VMEbus modules shortly described in 0, 0 and 0. The overall run control and monitoring is provided via software running on a (Linux) desktop PC as briefly described in 0.

Outline

This exercise is a continuation of exercise # 3. First, standalone programs are executed to give an understanding of the QDC and TDC VMEbus modules. A full DAQ system is then run on a multi-processor configuration, with the readout application on the VMEbus processor and the run control, GUI and infrastructure on a desktop Linux PC. Event rates and dumps are examined. An event monitoring program produces histograms of the QDC and TDC channel data which allow to ompute the charges of the input signals to the QDC and the speed of the cosmic muons.

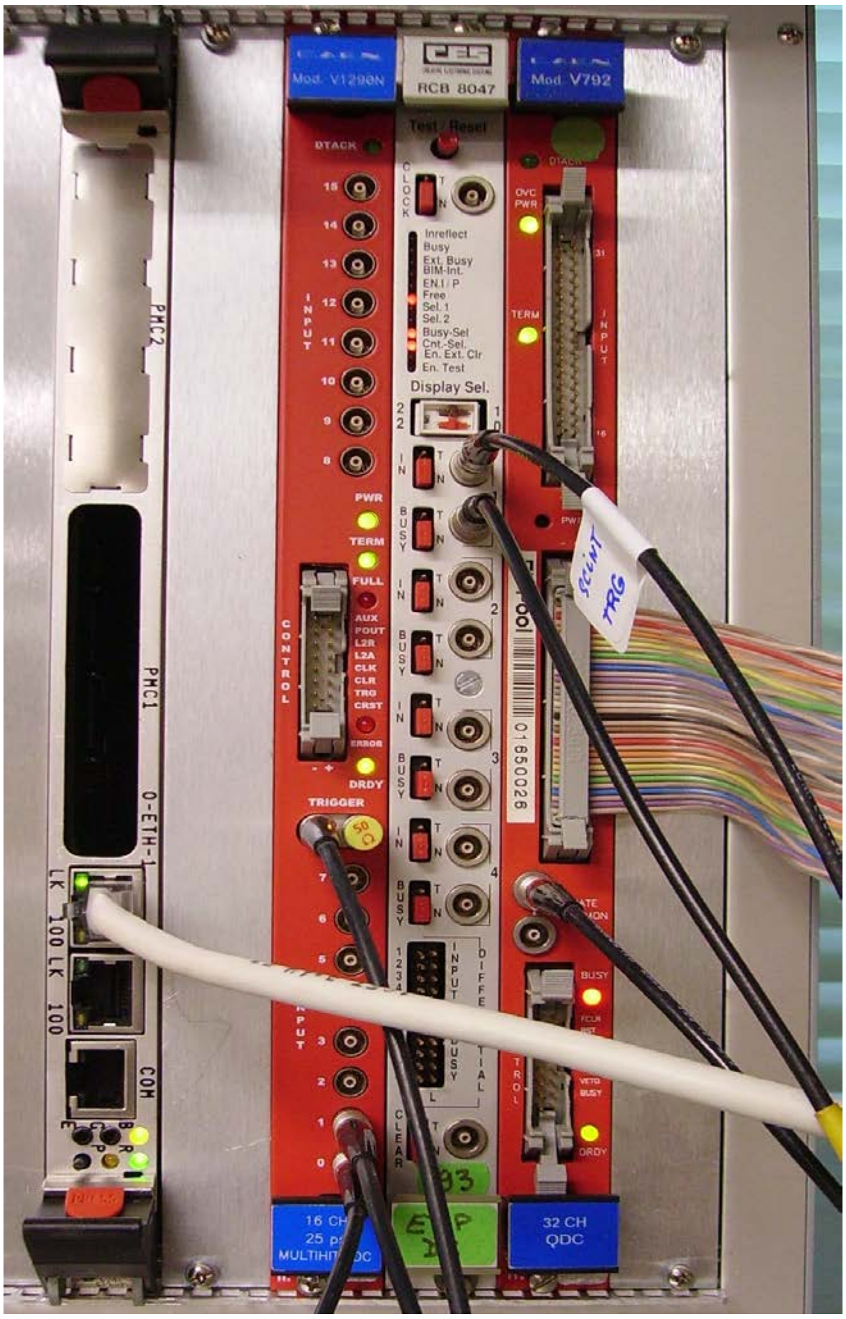


Figure 6. VMEbus data acquisition system: SBC (Single Board Computer), TDC(Time to Digital Converter, Trigger Module (CORBO), QDC(Charge to Digital Converter)

Workplan

- verify that the detector is working i.e. the scaler counts for scintillator 0, scintillator 1 and the coincidence are counting such that the TDC and QDC receive signals (note for the tutor: if the coincidences are not counting, remove the CORBO busy from the trigger coincidence by pushing the button).

- open a window on PC pcdaqschool1 (or pcdaqschool2) and login as user daqSchool on the VMEbus processor tds-sbc-01(or tds-sbc-02): ssh daqSchool@tds-sbc-01(or tdb-sbc-02), pw=g0ldenhorn

- source ./setup4a or ./setuo4b (depending on which set-up you ar working) to define the environment
- run the program **v1290scope** which is a low-level test and debug program for the CAEN V1290 TDC
  1. cd DAQ/DataFlow/rcd_v1290/i686-slc4-gcc34-opt
  2. ./v1290scope Use defaults for the command parameters.
  3. dump the registers (option 2). Are data ready? (bit DREADY in the status register). What are the values of the match window width and the window offset? See 0.
  4. configure the TDC (option 3)
  5. read an event (option 5). The event has a format as shown in the CAEN manual pages: Output Buffer Register. How many words are read? (Check in the global trailer).What are the values of the TDC measurements (in ns). Do they make sense? See 0. Exit from the program (type0).
- run the program **v792scope** which is a low-level test and debug program for the CAEN V792 QDC
  1. cd DAQ/DataFlow/rcd_v792/i686-slc4-gcc34-opt
  2. ./v792scope
  3. VMEbus base address = 0
  4. dump the registers (option 2). Are data ready? Check also the LED on the module
  5. read an event (option 4). How many words are read? Which channels have data and which are pedestal(empty) values?
- we now run the full DAQ system
  1. Open a window on pcdaqschool1 (or pcdaqschool2) and login as user daqSchool: ssh –X daqSchool@pcdaqschool2, pw = g0ldenhorn.
  2. Source ./setup4a or ./setup4b to define the environment
  3. Start the DAQ system: setup_daq -p part_Scintillator -d part_daqSchool(the exact command line is printed in the output from the setup script, cut and paste from there ..). This script will read the configuration database and start a number of processes on the server: run control, GUI and a number of infrastructure SW components as sketched in 0. This is a somewhat long procedure and should result in a GUI display as shown in Figure 7. The "wheels" in the infrastructure panel should be green! You may need help from the tutor here ...
  4. We now go through the run states in order to start a run. Click on BOOT and then INITIALIZE. The readout application is now loaded on the VME processor.
  5. Click CONFIG and OK on "remember ..". This configures the VMEbus modules, the CORBO, QDC and TDC.

6. If you don't see the DFPanel tab close to the top of the GUI, click LOAD Panels and load the first panel: DFPanel should now appear in the bar above the Run Control panel.

7. Click START.

8. Data taking should now start. Click on the DFPanel and the L1 button to display the event rate. Is it what you would expect after exercise # 3? Check also the LEDs on the VME modules (the event rate is computed by the Information Service(IS) which periodically sends a command to the Readout Application to obtain the rate which is then retrieved by the GUI).

9. Click on the ED button at the top to produce an Event Dump. Expand "part_Scintillator" and "ReadoutApplication"(the lower one). Click on" Scintillator". Click the black right arrow in the panel above to dump an event. The first nine words constitute an Event (ROD) header. The following words are the data from the QDC and the TDC. Do you recognize the data?

- Event Monitoring

  This part demonstrates event monitoring. An event monitoring program obtains a sample of events from the readout application and analyses them, in this example by producing histograms of the values from the QDC channels as well as the time difference between the two TDC values. The histograms can then be viewed via the GUI. The code for the monitoring program can be found in /DAQ/DataFlow/ROSMonitor/src/EventMonitorMain.cc (the parts which are specific to the DAQ school are marked with *******)

1. Open another window and login as user daqSchool on pcdaqschool1 (or pcdaqschool2), ssh –X daqSchool@pcdaqschool2, pw = g0ldenhorn

2. Source ./setup4a or ./setup4b to define the environment

3. cd DAQ/DataFlow/ROSMonitor/i686-slc4-gcc34-opt

4. Run the event monitoring task: ./emon_task -p part_Scintillator (or part_Scintillator1) -t ReadoutApplication -e 1000 -v2 (1000 events in debug mode). The exact command line is shown in the output from the setup script: cut and paste from there ..).

5. events are now being monitored with debug information. At the end the histograms are stored such that they can be viewed from the GUI.

6. In the GUI click on the OH button (Online Histogram). Click on Histogram Repository, part_Scintillator, ScintMon. Double click on the histograms to view them.

7. Record the mean values of the QDC histograms and the mean value of the time difference histogram. The time histogram is not centered around zero. Why?

8. The pedestal values of the QDC channels are now measured. Remove the inputs to the QDC channels by unplugging the LEMO cables on the delay units.

9. Run the monitoring program again as described in point 4.

10. Display the histograms of the QDC channels. Record the pedestal values.

11. Using the formula shown in 0, compute the mean charges of the signals from the scintillators. Do they agree with the results obtained in exercise #3?

- We now want to measure the time of flight of the muons between the two scintillators.

1. Increase the distance between the scintillators by about 30 cm. This is done by moving the scintillation counter to the upper hole in the support AND turning it 180 degrees. What is the event rate?

2. Run the monitoring program again with 200 events. Record the mean value of the time histogram viewed from the GUI.

3. What is the difference with respect to the value measured before? Compute the speed of the cosmic muons.

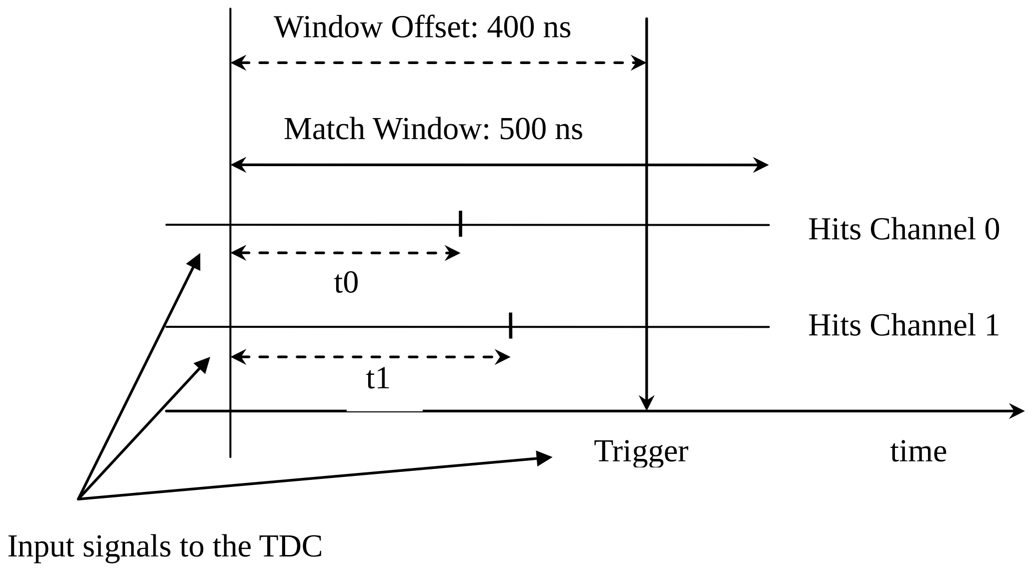
Appendix 1: TDC CAEN V1290 VMEbus module

The TDC is operated in *trigger matching* mode. This means that the TDC measures the time of arrival of the hits on a channel within a *match window*. The TDC receives a trigger and the channel signals as shown in the diagram of the complete setup, Figure 4 and seen in the picture of the VME crate, Figure 6. A trigger match window is then defined by a window offset wrt the trigger and a match window size as shown in the figure below. The hits occurring on channel 0 and channel 1 within the match window are recorded by the TDC and the values in units of 25 ps stored in the memory of the module.

The format of an *event* containing the data corresponding to a trigger is shown on a separate page.

The module is shown in the photo of the VMEbus crate and the manual for the module can be found at

http://www.caen.it/nuclear/product.php?mod=V1290N



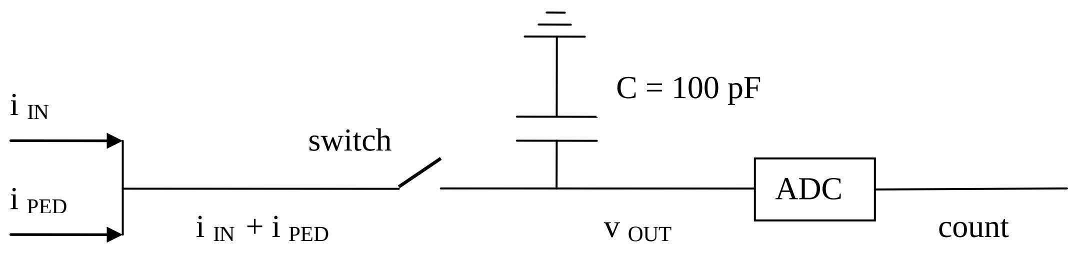
Appendix 2: QDC CAEN V792 VMEbus module

This page explains briefly how to calculate the charge of the input signal to the QDC from the data readout from the module over VMEbus. The module is shown in Figure 6.

The manual for the module can be found at

http://www.caen.it/nuclear/product.php?mod=V792

The circuitry of a channel is shown schematically, below.



The switch is closed as long as the gate input signal is present. The input current is the sum of i_{IN}, the current input to the module via the front panel (from the scintillator), and i_{PED}, a bias (or pedestal) current which is generated internally. The bias current allows to handle input signals with small positive voltage components. When the switch is closed during the time of the gate signal, the input current charges the capacitor C. When the switch is opened again, the voltage across C, v_{OUT}, is converted by an ADC and stored in the memory of the module. The ADC has the property that **one count = 1 mV**.

We now have for the charge of the capacitor:

$Q = C * v_{OUT} = 100\ (pF) * count\ (mV) = 0.1 * count\ (pC)$

To compute the charge in the signal input to the channel, corresponding to i_{IN}, we have to correct for the pedestal value:

$\mathbf{Q_{IN} = 0.1 * (count - count_{PED})\ (pC)}$

count = channel data with input signal present

$count_{PED}$ = channel data with input signal removed ($i_{IN} = 0$)

Appendix 3: CES RCB 8047 CORBO VMEbus trigger module

When a NIM signal is sent to a channel on the CORBO, a bit is set in a status register and an interrupt on VMEbus is generated, optionally.

The DAQ process on the VMEbus processor can then execute the code to readout the data from the QDC and TDC modules. In addition, the CORBO generates a busy signal which allows blocking further triggers until the readout code is terminated.

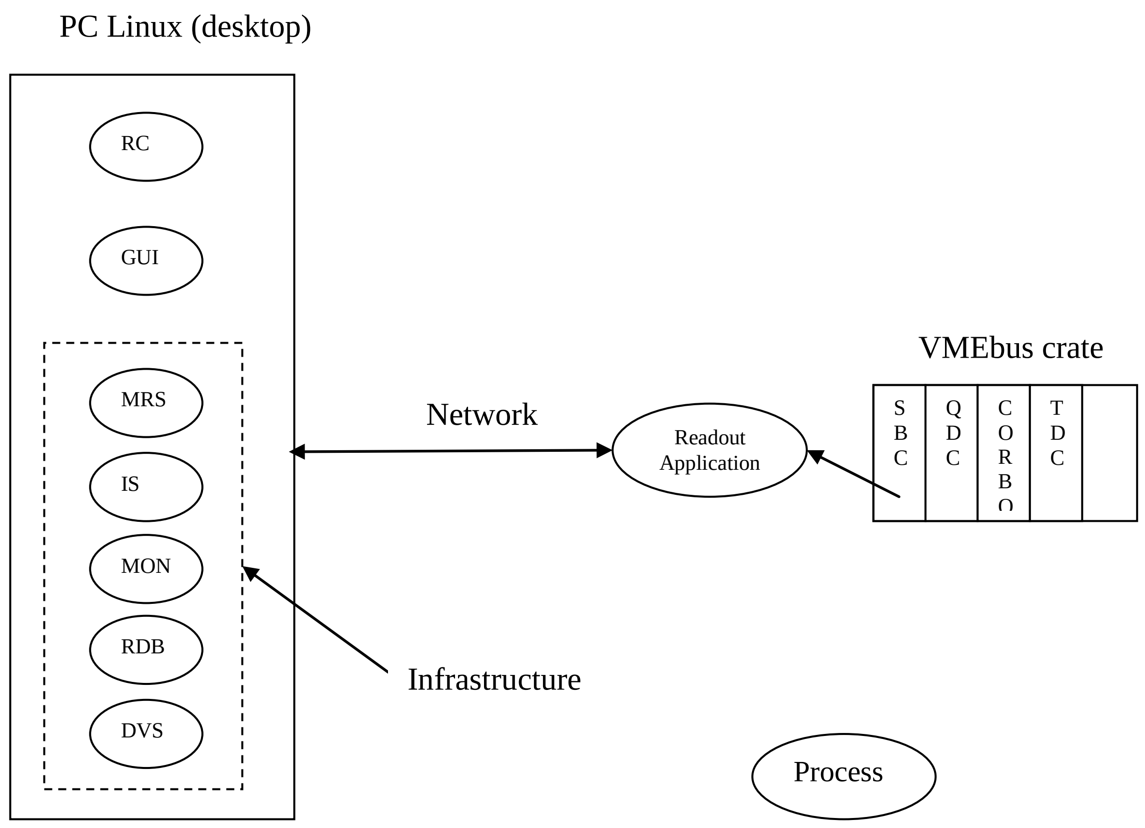
The CORBO module is shown in Figure 6.

Appendix 4: DAQ Processes

The DAQ system is started by executing the script setup_daq:

```
setup_daq part_Scintillator
```

where part_Scintillator is the name of the partition which describes the configuration of the system, hardware and software, via a configuration database. The figure below sketches the processes in the system. They communicate via IPC: a network based Inter Process Communication system



RC: run control

GUI: Graphical User Interface

MRS: Message Reporting Service

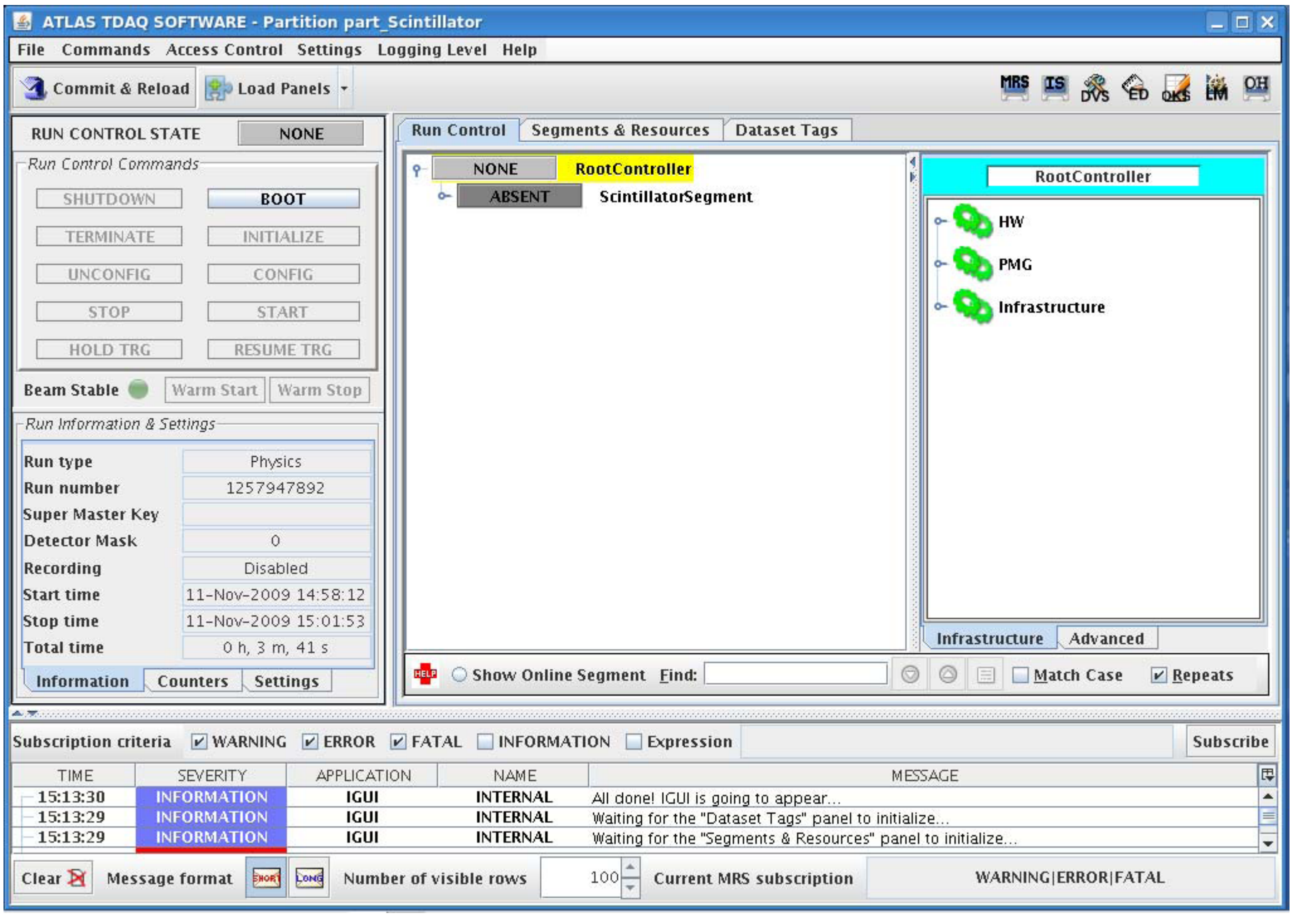
IS: Information Service

MON: Monitoring Service

RDB: Remote Database Access

DVS: Diagnostic and Verification System

Appendix 5: DAQ GUI



RUN CONTROL STATE: NONE

Run type	Physics
Run number	1257947892
Super Master Key	
Detector Mask	0
Recording	Disabled
Start time	11-Nov-2009 14:58:12
Stop time	11-Nov-2009 15:01:53
Total time	0 h, 3 m, 41 s

Subscription criteria: [✓ WARNING] [✓ ERROR] [✓ FATAL] INFORMATION Expression

TIME	SEVERITY	APPLICATION	NAME	MESSAGE
15:13:30	INFORMATION	IGUI	INTERNAL	All done! IGUI is going to appear...
15:13:29	INFORMATION	IGUI	INTERNAL	Waiting for the "Dataset Tags" panel to initialize...
15:13:29	INFORMATION	IGUI	INTERNAL	Waiting for the "Segments & Resources" panel to initialize...

Number of visible rows: 100
Current MRS subscription: WARNING|ERROR|FATAL

Figure 7. DAQ GUI after startup

FPGA Programming

Exercise 5

INTRODUCTION:

In a lot of digital designs (DAQ, Trigger, …) the FPGAs are used. The aim of this exercise is to show you a simple way to design logic in a FPGA. You will learn all the steps from the idea to the test of the design.

In this exercise you will:

-discover how we can do parallel applications

-program a FPGA from the design up to the implementation and the test

The boards used are ALTERA development kit (Figure 1) based on a small FPGA (CYCLONE) with multiple additional interface components like audio CODEC, switches, button, seven-segments display, LEDs, ….
and a home-made board (named detector in the following pages) connected to the development kit with a flat cable (figure 2)

The initial design is loaded into the board.
You will follow this example to understand the design flow. Three exercises are proposed to modify to original design functionality.

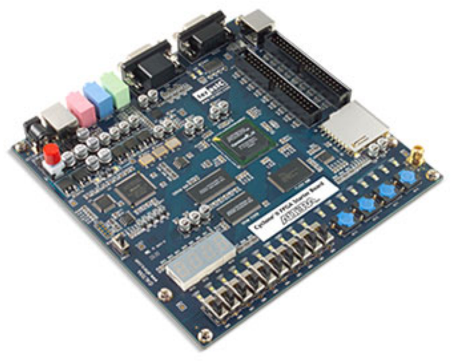


Figure 8: development kit

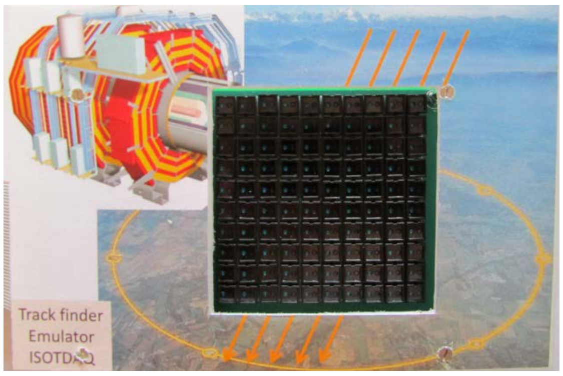


Figure 2: detector

QUICK START:

1) Start the PC, the instructor will give you the user name and password (student—Poland2012).
2) Programs used are: QUARTUS (FPGA tool), ModelSim (simulator), LabView.
3) Ask the tutor in case of problem.

EXERCISE (example)

When you switch on the kit, the initial design is loaded into the FPGA.
On the LabView window, you can see when the marker is passed over the detector (create a trace).
At the same time, you can see in the 7-segments display (Altera kit) the column and the line number over which the maker is positioned.
Ask the tutor for a demonstration.

DESIGN ENTRY

The design file is named “CII_Starter_Default.bdf” (for all exercises you should work with the same design file).
The design is divided in three parts:

a) A green rectangle which is used to transmit the information to the computer via the RS232 connection to display the trace on LabView.
b) A blue rectangle which generate the clock and the logic to control the detector (see Appendix A for detailed functionality).
c) A red rectangle, which contains the logic to detect the trace. You will change the logic in this rectangle in the following exercises.

The idea of all exercises is to detect a trace. As soon as the trace is detected the first 7-segment display blinks.
Click on the key0 (Altera kit) to stop the blinking. Now generate another trace.
Spend some time to understand how this design works.
Do you understand it?

COMPILATION
This design is the entry of your logic, it should be compiled now; go to *Processing->Start Compilation*.
The design is compiled for the chosen component (Cyclone II).
The compiler executes multiple tasks: logic optimization generates the binary file to program the FPGA (memory array), extracts the timing between each logic elements (used for the simulation).

SIMULATION
After this, you can check the design with a simulator. To do this you will use ModelSim.
In the “Transcript” tab, type ‘source sim.tcl’, ENTER. The simulator opens the waveform, loads the signals, and starts the simulation.
At the end, stimuli and results are displayed in the wave window.
This simulation generates a trace starting from the top left and finishing at bottom right describing a straight line.
(line0 :column0; line1 :column1;….; see figure 3)
Remember where the signal OK goes to “TRUE”.

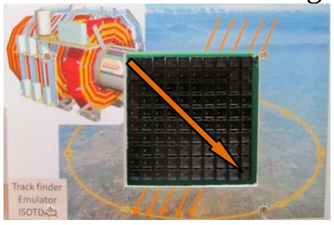


Figure 3: straight line

When you finished with the simulator type ‘quit –sim ‘ ENTER in the “Transcript” tab.

PROGRAM THE KIT

To download the design on the board, (QUARTUS program) go to on *Tools->Programmer* (Check that the Hardware is USB-Blaster, if not ask the instructor).
One file is shown in the window: it is your design. Click on Start .The programmer takes some seconds. At the end, a message appears to inform you that the programming is completed (or not successful: in this case usually the board is switched OFF, or the cable is not well connected).

TEST

Now, you are ready to do the other exercises yourself.

Good Luck!

EXERCISE I

The exercise above uses the graphic to describe the design. In this exercise, we want to do the same with a text design entry (VHDL).

In the QUARTUS design entry (file “CII_Starter_Default.bdf”), delete the line between inst134 and JKFF inst132 and connect the output ‘result’ of “track1”box to the JKFF inst132 with a line.

-Compile the design

-Simulate the design

Go to ModelSim, compile the file marked with a ? in the “Project” tab (click on the file to compile –   Menu Compile-> Compile selected)

Type “quit –sim“ in the “Transcript” tab.

Type “source sim.tcl “ in the “Transcript” tab.

Find out the difference with the previous result (check where the signal OK goes to “TRUE”).

Can you explain the difference? Can you modify the file “track1.vhd” to have the same result as in the previous exercise?

-Download the design

-Test the design

EXERCISE II

In this exercise we want to detect a curved trace.

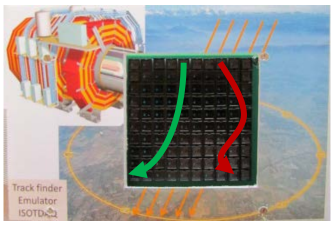
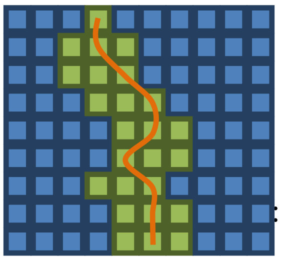


Figure 4



: example of trace expected.

In the QUARTUS design entry (file “CII_Starter_Default.bdf”), delete the line between output ‘result’ of “track1” box to the JKFF inst132, and connect the output of the box “trck_fnd01” to JKFF inst132.

The “trck_fnd01” box logic detects only a straight trace. Compile the design and do a simulation (see below for simulation).

The exercise consists to modify the “trck_fnd01” box logic to detect any curve trace as in figure 4. The trace should start at any pixel in the first line and goes to next line passing to a pixel adjacent to the pixel of the previous line and so forth (figure 5).

-Compile the design

-Simulate the design

Go to ModelSim, compile the file marked with a ? in the “Project” tab (click on the

file to compile – Menu Compile-> Compile selected)
To simulate, type “quit –sim’ ENTER in “Transcript” tab to exist any running simulation.
Type ‘source sim2.tcl’ ENTER in “Transcript” tab to simulate in this exercise.
A signal OK becomes true if the logic detects the expected trace.

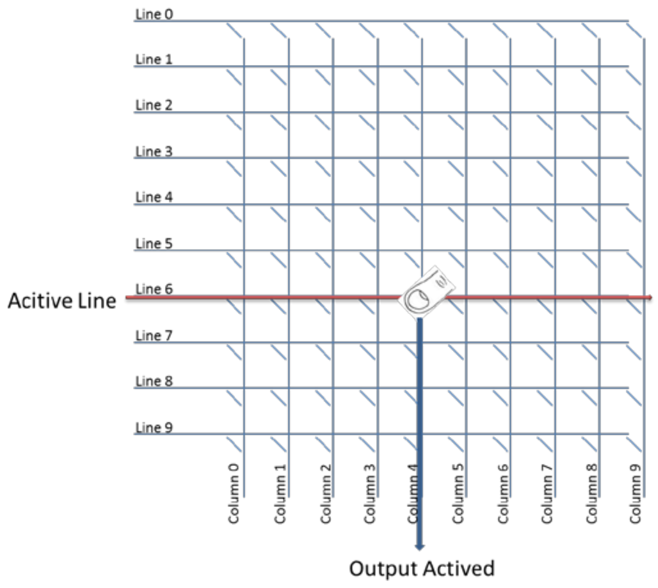
-Download the design
-Test the design

EXERCISE III

If you have time, you can modify the previous file to detect only the curve trace on right or left (not in zigzag like the red trace in figure 4).

APPENDIX A

The detector is a matrix of 10 lines and 10 columns (100 pixels). Only one line is activated at a time.



When a line is activated the result of each column indicates if the marker is over a pixel in the corresponding line. Each line is activated one after the other (0, 1, 2… 8,9,0,1, etc). Each line is activated during 4 clocks cycles. The detection logic checks the result (if pixel is masked by the marker) only during the third clock cycle (signal “check” in the design).

Data Acquisition with a PCI module

Exercise 6

Introduction

In this exercise you will

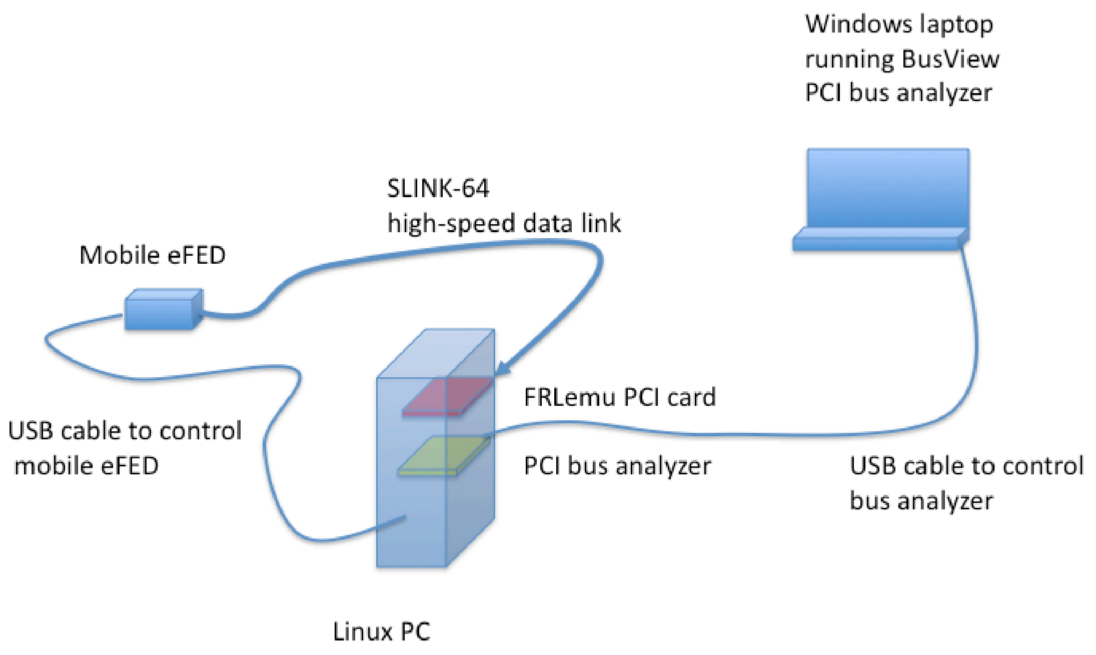
- learn about the basics of the PCI bus,
- perform read and write accesses and watch them with a bus analyzer,
- do some benchmarking of read and write accesses and see the overhead of a software library,
- write a simple data acquisition software that reads events form a high-speed data link (SLINK-64) and decodes fields in the headers and trailers and verifies the length of the event
- benchmark your solution
- look at an implementation using Direct Memory Access (DMA) and watch the PCI bus activities with the bus analyzer

Pre-requirements

You have to read the introduction to the PCI Bus (Appendix B) before beginning the exercise

Work plan

Description of the Exercise setup:


Windows laptop running BusView PCI bus analyzer
SLINK-64 high-speed data link
Mobile eFED
FRLemu PCI card
USB cable to control mobile eFED
PCI bus analyzer
USB cable to control bus analyzer
Linux PC

The SLINK-64 is the data link used to read out data from the CMS detector at CERN. It transfers data words of 64 bit width plus one control bit per data word. The design speed is 50 MHz – corresponding to 400 MB/s bandwidth. Data are transferred using the Low Voltage Differential Signal (LVDS) standard. A multiplexing ratio of 8 to 1 is used on the cable. This means that data are actually transferred over multiple serial LVDS links. The data record for event fragments is explained in Appendix A.

Exercise 1: PCI read and write access
In this exercise you will look at single read and write accesses on the PCI bus. You will perform read/write accesses to the FRL Emulator PCI card but you will only use a single register of the card. You will capture the activity on the PCI bus using the PCI bus analyzer card.

1) Start the PCI Bus analyzer (Windows laptop)

- Double-click on the VMETRO BusView icon on the Desktop. Click Ok.
- In the "PCI Current Setup" window you can select the trigger conditions for the PCI analyzer
  In the line PCI 0:
    - Set the column C/BE to xxxxx1xx to trigger on Memory accesses, only
    - Set the column #FRAME to 0 and #IRDY to 1 to trigger on the start of a cycle
    - All other columns should be 'x' (or multiple 'x')

  Set the trigger position to 25% (second of the red/green buttons in the toolbar)
- Click the flash icon in the toolbar to arm the trigger

2) Start the PCI_interactive.exe program to generate read and write accesses

- Log into the Linux PC (User name: `dgigi`, pwd: `dogigi`)
- `cd ~/proDir/TriDAS/daq/fedbuilder/frl_emu`
- `bin/linux/x86_slc4/Ex1_PCIInteractive.exe`

The program will prompt you to do read or write accesses.

3) Perform an access

4) The PCI Bus analyzer will display a window "PCI Clock mode Trace on Tracer …"
To get a graphic view of the access, click on the "Open new waveform window" button (third from the right) in the toolbar.

- Study the signals during the access.
- Close the waveform window and the textual Trace window.

To perform another access:

- Click the flash button in the toolbar of the PCI analyzer to arm the trigger.
- Go back to 3).

Exercise 2: Benchmark read and write access using different libraries
In this exercise, you will measure the average time for PCI read and for PCI write accesses when these are invoked using a software library. You will learn how much overhead the software library adds.
We will use the Hardware Access Library (HAL) developed for CMS. This library provides a convenient way of defining registers and bit fields inside registers, which can be addressed by giving their name. [If you are interested you may have a look at the definition of the address table in one of the include files of this exercise.]

On the Linux PC,

- `cd ~/proDir/TriDAS/daq/fedbuilder/frl_emu`

You will need to modify the source file
`src/common/Ex2_BenchRW_HAL.cc`

using your favorite editor.
The source files contain comments that will tell you how to use the jal::Timer() library to time your PCI accesses. They also provide examples of a read and a write access.

Your program should give the average time in microseconds (averaged over 10000 reads or writes) for

1. a PCI read access
2. a PCI write access

Run `make` to build the executable.

The executable will be located in `bin/linux/x86_slc4`
Run your executable.

- How do the times you measured compare to the durations of the PCI accesses on the bus?
- How much overhead is added by the library?

Exercise 3: Read data from a digital readout link (SLINK-64)

In this exercise, you will generate data in the mobile eFED, transmit it to the FRL Emulator PCI card over the SLINK-64 and read it into the PC memory by doing individual PCI accesses. Each event starts with a 64bit header word, followed by a number of 64bit data words and finally ends with a 64bit trailer. In order to differentiate between control words (header/trailer) and data words, a 65th bit is transferred over the link. Appendix A explains the data format of the event data in more detail.

1) On the Linux PC, modify the source file
```
src/common/Ex3_FRLemu_DAQ.cc
```

Then compile / link it by running
```
make
```

and start the executable in `bin/linux/x86_slc4`

The program already prints the received data to the screen (in hex or binary format).
Add some code to count the length of the received events (the length is counted in 64bit words including the header and the trailer word) and compare it with the ‘Evt_lgth’ counter in the event trailer.

2) In another terminal on the Linux PC, start the Control Panel of the mobile eFED
```
cd ~/usb_tester
labview usb_tool.vi &
```

In the window that pops up, click on the right arrow button in the tool bar, to start the LabView VI. In the next Window, click Ok. Select either loop mode or single trigger mode and click Configure, then Start. In single trigger mode you may produce single events by clicking the Trigger button. In loop mode the mobile eFED will send generate data until it receives back-pressure from the PCI card. In loop mode you may also use the Pause button to temporarily stop sending of data.
Exercise 4: Benchmark your readout
Modify your program from Exercise 3 to read as fast as possible and time the reading. You will need to turn off printing of the events.

You may start from Exercise 3 or from
`src/common/Ex4_BenchFRLemu_DAQ.cc`

For this test you should use Loop mode of the mobile eFED. How many MB / second can you read? Compute the throughput in MB/s (Megabytes per second) for

1. all data read from PCI and
2. for the event data (header+payload+trailer), only.

Note: 1) will be higher than 2) since we are reading more than only the data

- How does the throughput for all data read from PCI compare to the benchmark in Exercise 2?
- How could you improve throughput?

Exercise 5: DMA data transfer

In this exercise you will use an application, which transfers data from the SLINK receiver card to the PC memory using Direct Memory Access (DMA). In this case the FRLemu PCI card acts as a bus master and writes the data to the PC's memory. Data are transferred in "burst mode". This means that at the beginning of the access, the address is transferred, and then many data words are transferred in the same cycle.

On the Linux PC, start the fedkit-test-merge application.

- `cd ~/fedkit3`
- `./fedkit-test-merge –J 2 –L 1 –n 1000000000000`

On the LabView console of the FED Emulator, start event generation in loop mode.

On the PCI Bus analyzer, arm the trigger. It should immediately display some traces. Look at the waveforms. You will see direct memory access cycles and single read/write operations.

Stop the fedkit-test-merge application by pressing Ctrl-C.

It will display the bandwidth achieved. How does this compare to the bandwidth you measured in Exercise 4?

Exercise 6 (If you still have time):
Repeat exercise 2 using a simpler library (pci-access). The functions in pci-access directly call a kernel driver to perform the PCI read/write accesses.

You may start from:
`src/common/Ex2_BenchRW_PCIAccess.cc`

How much faster is the pci-access library?

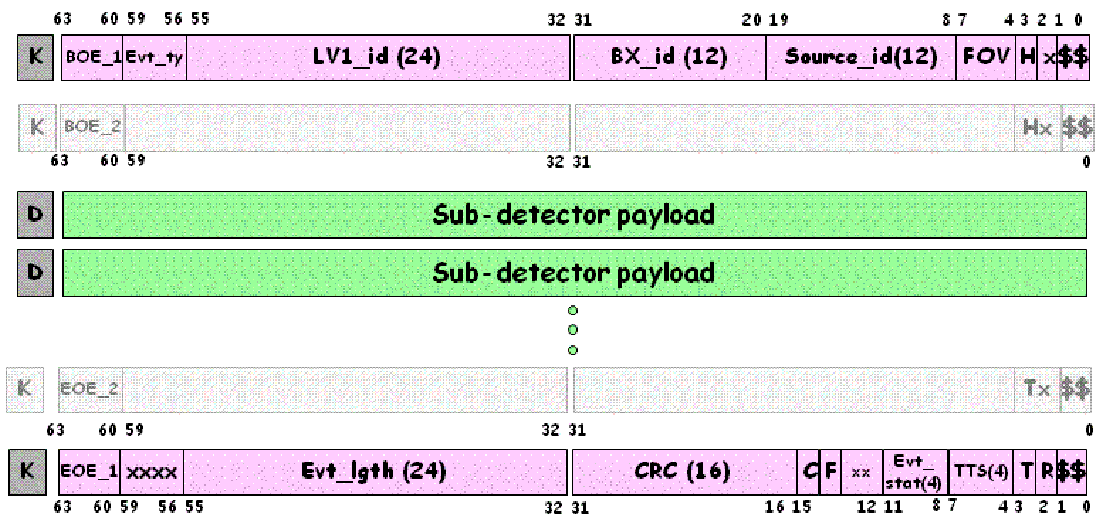
Appendix A: SLINK-64 data format

<u>Fields needed in the exercise:</u>

- K/D: the 65th bit is ‘1’ for control words (K) and ‘0’ for data words (‘D’)
- BOE_n: Identifier for the beginning of an event fragment (BEO_1 = hex 5)
- EOE_n: Identifier for the end of an event fragment (EOE_1 = hex A)
- Evt_lgth: The length of the event fragment counted in 64-bit words including header and trailer
- CRC: Cyclic Redundancy Code of the event fragment including header and trailer (The CRC field in the trailer has to be set to 0 when calculating the CRC)

Other fields:

- Evt_ty: Event type identifier (see notes below)
- LV1_id: The level-1 event number generated by the TTC system.
- BX_id: The bunch crossing number. Reset on every LHC orbit
- Source_id: Unambiguously identify the data source (FED/DCC).
- FOV: Version identifier of the common FED encapsulation (header + trailer).
- H: when set to '0', the current header word is the last one. When set to '1', another header word is following.
- C : when set to '1', the FRL has detected a transmission error over the s-link cable. - F : when set to '1', the FED_ID given by the FED is not the one expected by the FRL.
- Evt_stat: Event fragment status information
- TTS: Current values of the TTS bits
- T: when set to '0', the current trailer word is the last one. When set to '1', another trailer word is following
- R : when set to '1', the CRC value has been modified by the S-link sender card. The FED/DCC must set this bit to '0'.
- x: Indicates a reserved bit. The FED/DCC must set this bit to '0'.
- $: Indicates a bit used by the S-LINK64 hardware. The FED/DCC must set this bit to '0'.



Appendix B: Introduction to PCI

The PCI bus was introduced in the PC in 1992. Since then, it evolved in throughput with PCIx and PCIe.
But let’s focus on PCI. It is a synchronous parallel bus of 32 or 64 bit with a frequency up to 66 MHz (usually it is 33 or 66 MHz).
Bus: multiple elements can be attached to the same electrical signals.
Synchronous: each signal is latched on the clock rising edge.
Parallel: up to 64 bit can be transferred on each clock rising edge.

A PCI bus is composed of at least two elements (Master/Initiator and Slave/Target) with a maximum of 8 elements up to 33 MHz clock and a maximum of 4 up to 66MHz. One element can be Master or slave at different times.

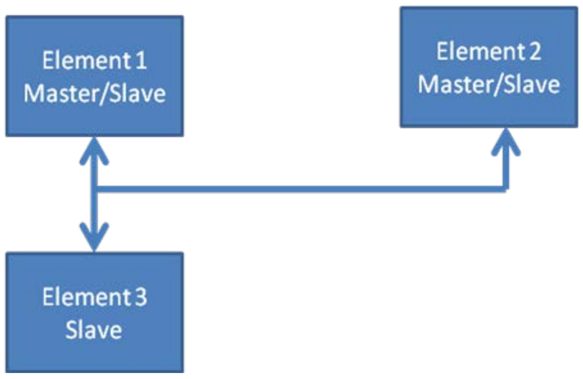


Figure 1: Topology example

The bus is composed of a clock, some control signals and 32 or 64 bits of data.
Clock is a free running clock up to 33 or 66 MHz provided by the system.
Data bits are 32 or 64 bit (data can be read or written by byte).

Control signals: (all signals are active low)

Managed by the Master/Initiator:

CBE(Command Byte Enable – 4 bits):
at address phase, it specifies the command (see appendix C)
at data phase it informs about the byte involved in the transfer (see appendix D)
FRAME:
This signal goes low at the beginning of the address phase of the transfer. When this signal goes low, CBE defines the command and the 32bit (64) of data contain the address (element base address : upper bits; internal address : lower bit).
This signal will go high at the clock before the last data.
IRDY : (Initiator ReaDY):

- when low indicates in a write access when the data are available on the bus.
- when low indicates in a read access when the master is ready to take data.

Managed by the Slave/Target:

TRDY: (Target ReaDY):

    - when low indicates in write access when the slave is ready to catch data
    - when low indicates in read access that the data on the bus is valid (provided by the slave)

DEVSEL: (DEVice SELect):

when low indicates to the master that a slave element is responding to the access (read or write). The slave has maximum 3 clocks to goes low after FFRAME goes low. After that the master aborts the access (BUS error).

STOP:

when low indicates that the slave is not able to catch data (write access) or to provide data (read access).

Managed by Master and slave:

PAR and PAR64 :

these two parity signals are driven by the element providing the address/data on the bus.

Other signals exist, but will not be presented here: INTA,INTB,INTC,INTD, LOCK, ….

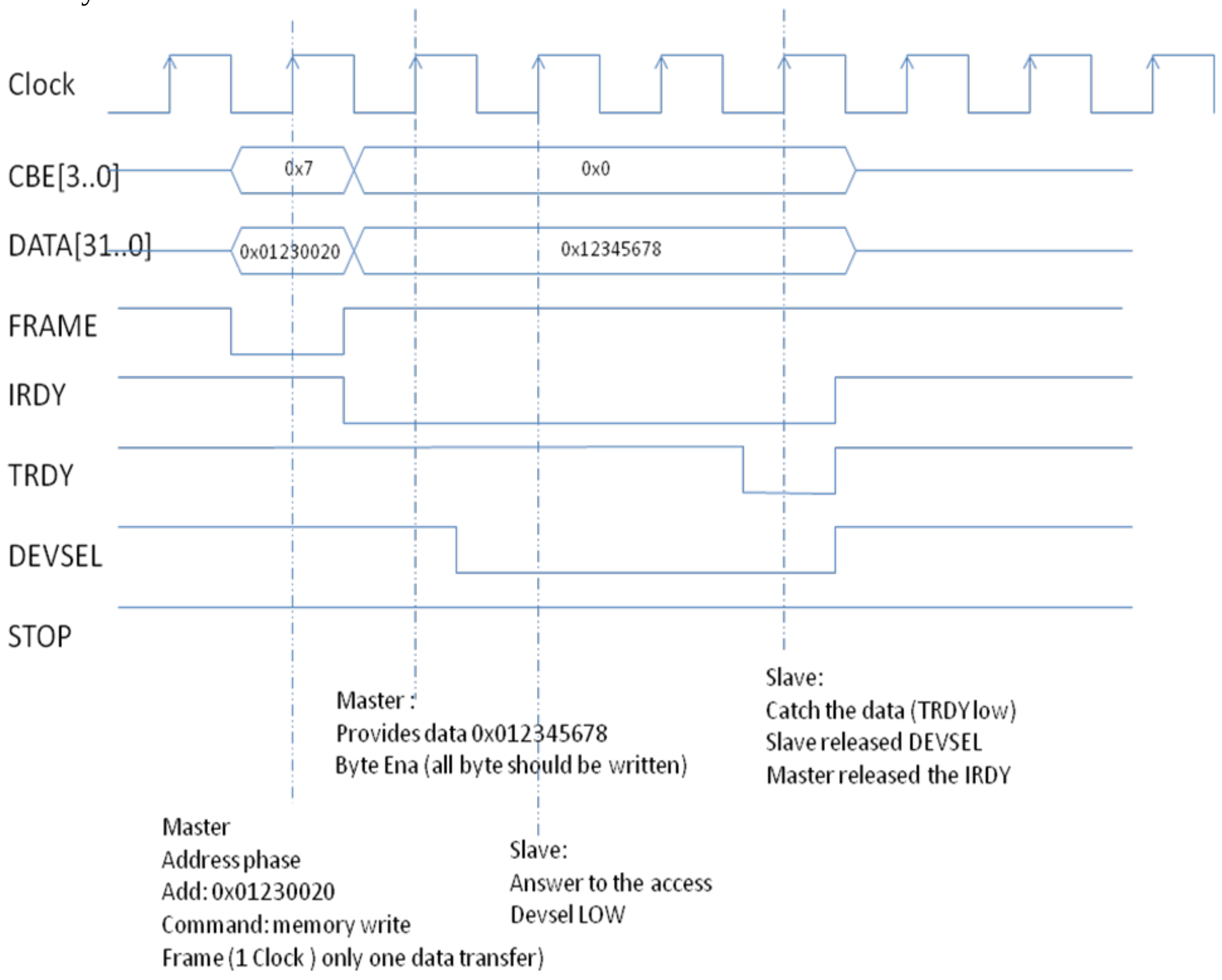
IDSEL: (ID SELect)

this signal is use by the system Master to select each element during the configuration (BASe address setup, ….).

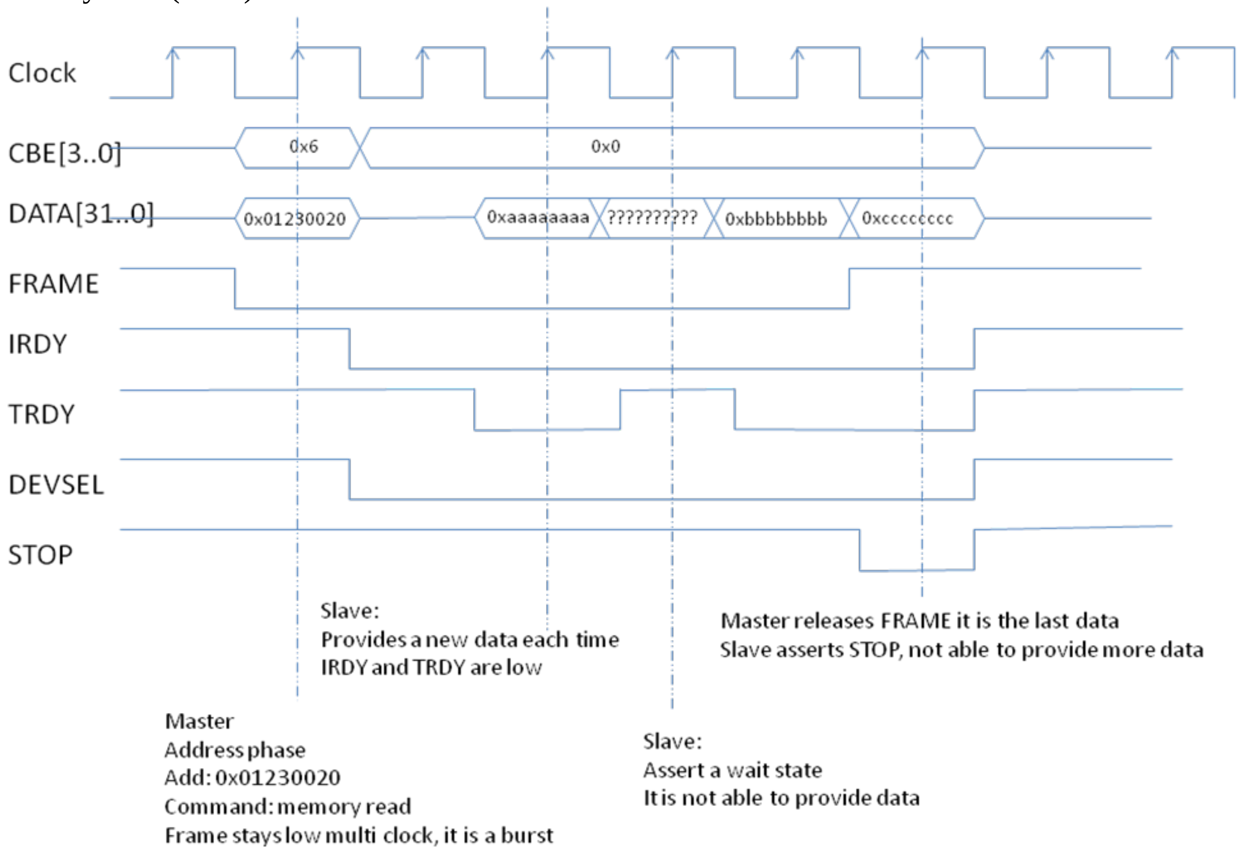
There are separate lines used for bus arbitration which are not discussed, here. Arbitration is the process used to decide which device may act as a bus master for the next transfer(s).

Some access examples:

Memory write:



Memory read (burst):



APPENDIX C:

Command : (only CBE 3 to 0 specify the command)

0x0000:	Interrupt Acknowledge
0x0001:	Special Cycle
0x0010:	IO read
0x0011:	IO write
0x0100:	Reserved
0x0101:	Reserved
0x0110:	Memory read
0x0111:	Memory write
0x1000:	Reserved
0x1001:	Reserved
0x1010:	Configure read
0x1011:	Configure Write
0x1100:	Memory Read Multiple
0x1101:	DUAL address Cycle
0x1110:	Memory read Line
0x1111:	Memory write and invalidate

APPENDIX D:

Byte Enable (active low)
CBE0 when low indicates that the bit 7..0 are concerned by the access
CBE1 when low indicates that the bit 15..8 are concerned by the access

The Byte enable is set at the address phase and valid for all data of the access (can’t change during the access.

LabView Programming

Exercise 7

Introduction:
This exercise will introduce the LabView programming language.

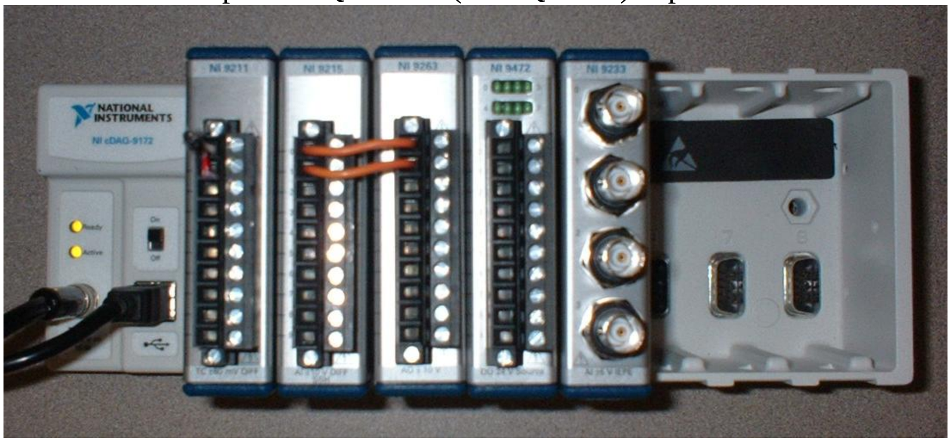
Work Plan:

<u>Exercise 1: Take a Basic Measurement with CompactDAQ</u>

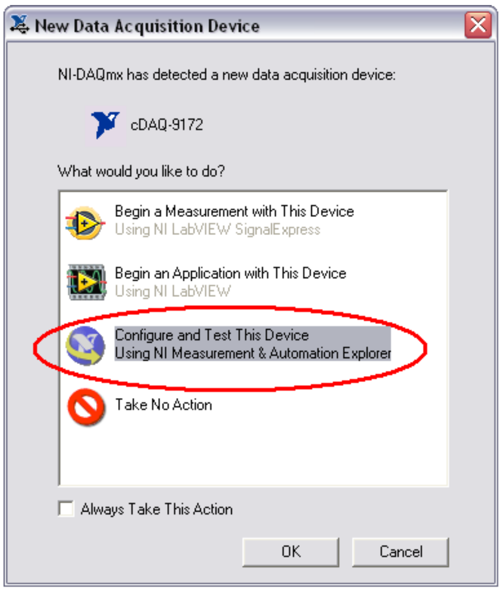
The purpose of this exercise is to use LabVIEW and NI CompactDAQ to quickly set up a program to acquire temperature data.

Set up the Hardware

1. Make sure that the NI CompactDAQ chassis (cDAQ-9172) is powered on.



2. Connect the chassis to the PC using the USB cable.
3. The NI-DAQmx driver installed on the PC automatically detects the chassis and brings up the following window.



New Data Acquisition Device

NI-DAQmx has detected a new data acquisition device:

cDAQ-9172

What would you like to do?

- Begin a Measurement with This Device
  Using NI LabVIEW SignalExpress
- Begin an Application with This Device
  Using NI LabVIEW
- Configure and Test This Device
  Using NI Measurement & Automation Explorer
- Take No Action

☐ Always Take This Action

OK | Cancel

4. Click on Configure and Test This Device Using NI Measurement & Automation Explorer.

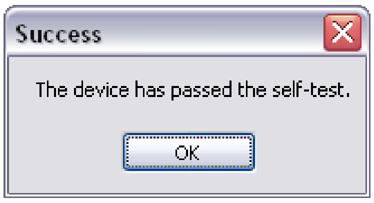
Note: NI Measurement & Automation Explorer is a configuration utility for all National Instruments hardware.

5. The Devices and Interfaces section under My System shows all the National Instruments devices installed and configured on your PC. The NI-DAQmx Devices folder shows all the NI-DAQmx

compatible devices. By default, the NI CompactDAQ chassis NI cDAQ-9172 shows up with the name “cDAQ1”.

6. This section of MAX also shows the installed modules as well as empty slots in the CompactDAQ chassis.

7. Right-click on NI cDAQ-9172 and click on Self-Test.


Success

The device has passed the self-test.

8. The device passes the self test, which means it has initialized properly and is ready to be used in your LabVIEW application.

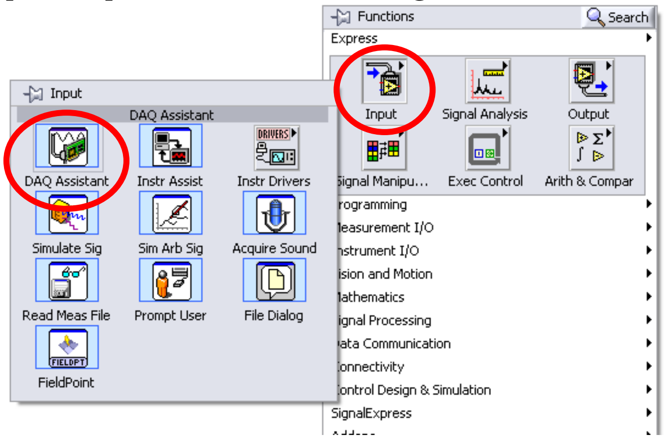
Program LabVIEW Application

9. Create a new VI from the Project Explorer. Right click on the Exercises folder and select **New» VI**. Once opened, Save the VI in the Exercise folder under the name “1-Basic Measurement.vi.”

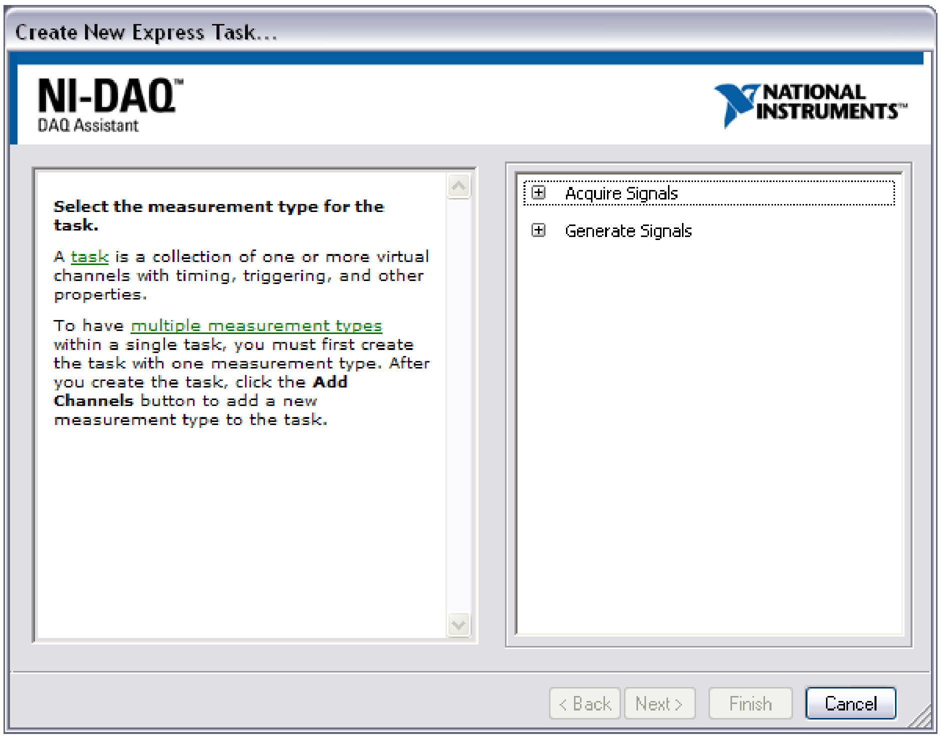
10. Press <Ctrl +T> to tile front panel and block diagram windows.

11. Pull up the Functions Palette by right-clicking on the white space on the LabVIEW block diagram window.

12. Move your mouse over the **Express» Input** palette, and click the DAQ Assistant Express VI. Left-click on the empty space to place it on the block diagram.



13. The Create New Express Task… window then appears:


Create New Express Task...

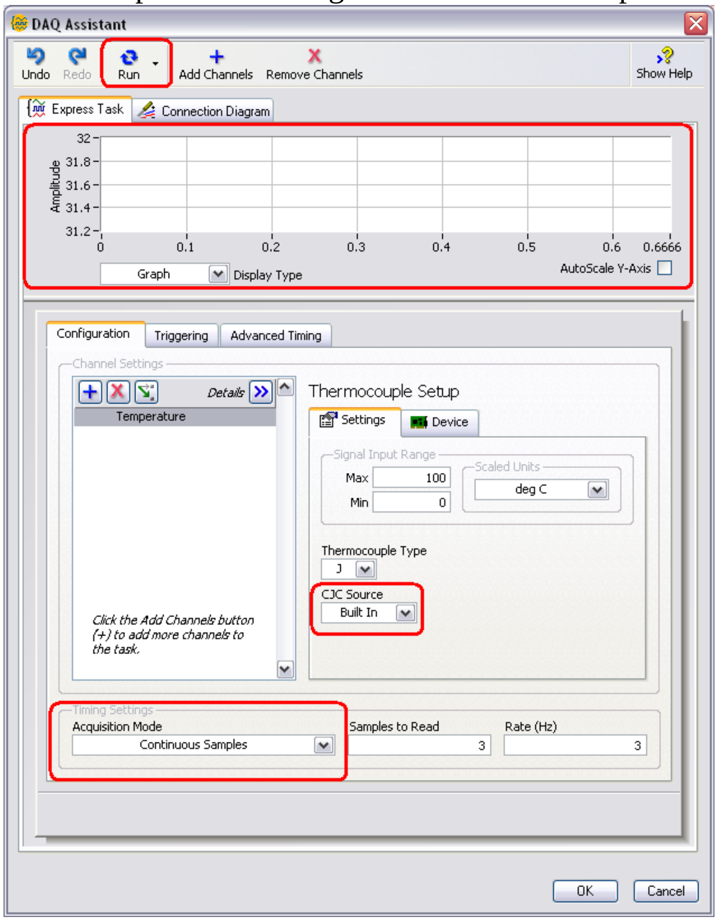
NI-DAQ™ DAQ Assistant

Select the measurement type for the task.

A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

14. To configure a temperature measurement application with a thermocouple, click on **Acquire Signals» Analog Input» Temperature» Thermocouple**. Click the + sign next to the cDAQ1Mod1 (NI 9211), highlight channel ai0, and click Finish. This adds a physical channel to your measurement task.
15. Change the CJC Source to Built In and Acquisition Mode to Continuous Samples. Click the Run button. You will see the temperature readings from the thermocouple in test panel window.



Signal Input Range: Max 100, Min 0

Scaled Units: deg C

Thermocouple Type: J

CJC Source: Built In

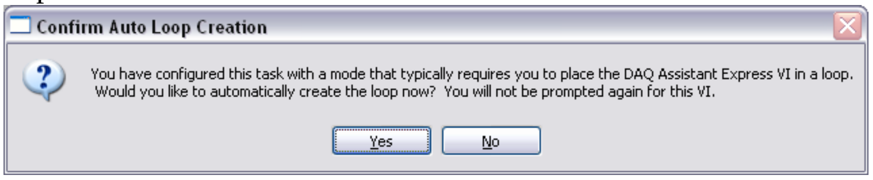
Timing Settings

Acquisition Mode: Continuous Samples

Samples to Read: 3

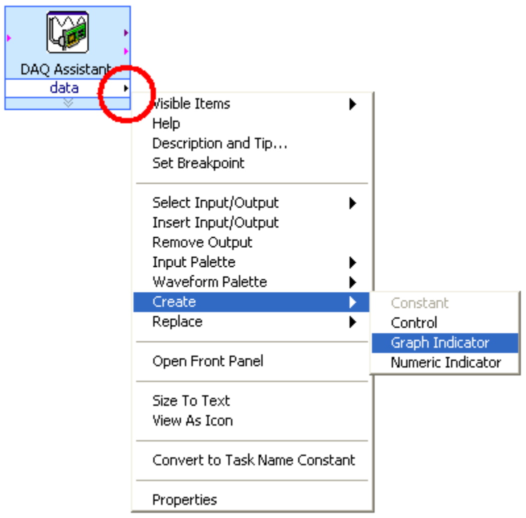
Rate (Hz): 3

16. Click Stop and then click OK to close the Express block configuration window to return to the LabVIEW block diagram.
17. LabVIEW automatically creates the code for this measurement task. Click Yes to automatically create a While Loop.

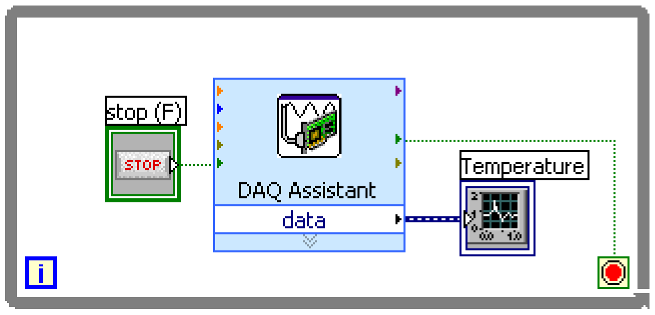

Confirm Auto Loop Creation

You have configured this task with a mode that typically requires you to place the DAQ Assistant Express VI in a loop. Would you like to automatically create the loop now? You will not be prompted again for this VI.

18. Right-click the data terminal output on the right side of the DAQ Assistant Express VI and select **Create» Graph** Indicator. Rename "Waveform Graph" to Temperature.

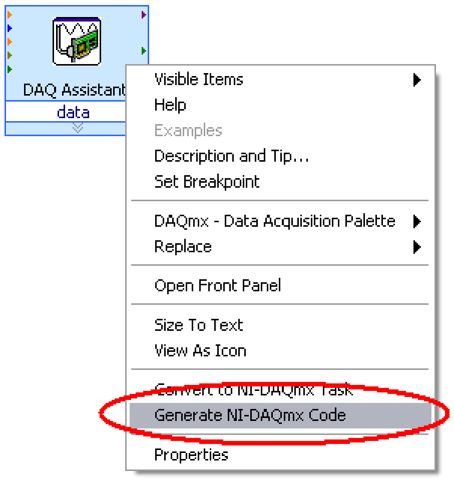


19. Notice that a graph indicator is placed on the front panel.
20. Your block diagram should now look like the figure below. The while loop automatically adds a stop button to your front panel that allows you to stop the execution of the loop.

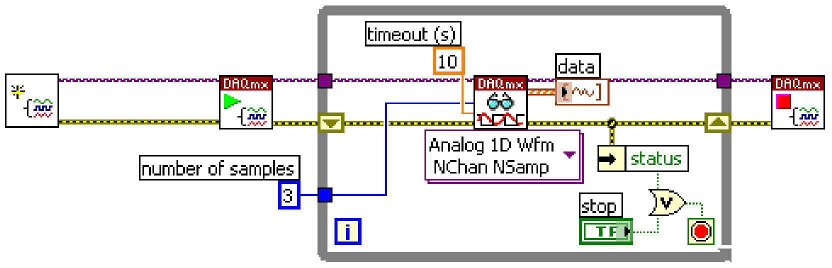


Additional Steps Express VIs make creating basic applications very easy. Their configuration dialogs allow you to set parameter and customize inputs and outputs based on your application requirements. However, to optimize your DAQ application's performance and allow for greater control you should use standard DAQmx driver VIs. Right Click on block diagram Functions» Measurement I/O Palette» NI-DAQmx.
20. Before you generate DAQmx code you need to remove all the code that was automatically created by the Express VI. Right click on the while loop and select "Remove While Loop." Then click on the Stop button control, and press the Delete key to remove the Stop button. Repeat actions for Temperature Graph as well as any additional wires that may remain. You can press <Control + B> to remove all unconnected wires from a block diagram.
21. Convert Express VI code to standard VIs. While not all Express VIs can be automatically

converted to standard VIs, the DAQ Assistant can. This will allow for greater application control and customization. Right-click on the DAQ Assistant Express VI you created in this exercise and select “Generate NI-DAQmx Code.”



Your block diagram should now appear something like this:

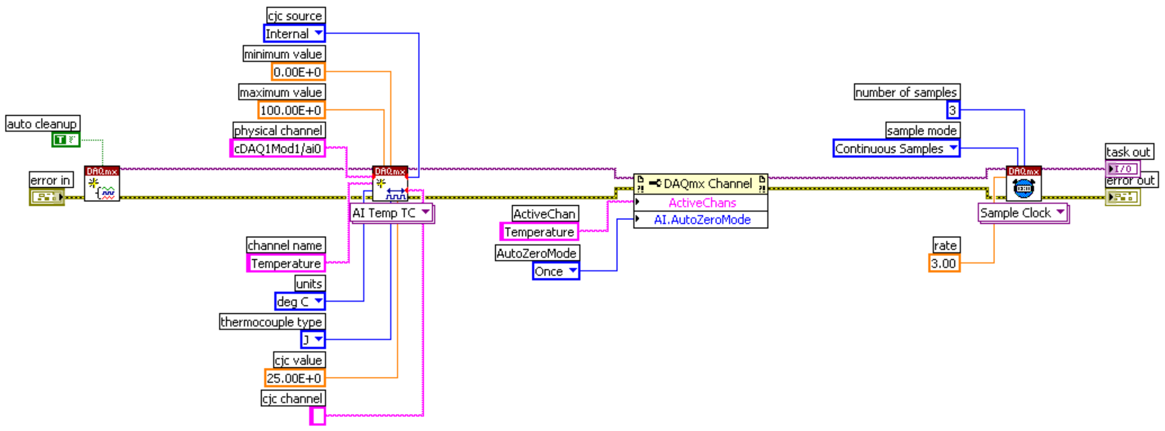


The Express VI has been replaced by two VIs. We’ll examine their functionality in the following steps.

22. Open Context Help by clicking on the Context Help icon on the upper right corner of the block diagram. Hover your cursor over each VI and examine their descriptions and wiring diagram.

23. DAQmx Read.vi reads data based on the parameters it receives from the currently untitled VI on the far left.

24. Double-click on the untitled VI and open that VI’s block diagram (code shown below).



All the parameters that are wired as inputs to the different DAQmx setup VIs reflect the setting you originally configured in the DAQ Assistant Express VI.

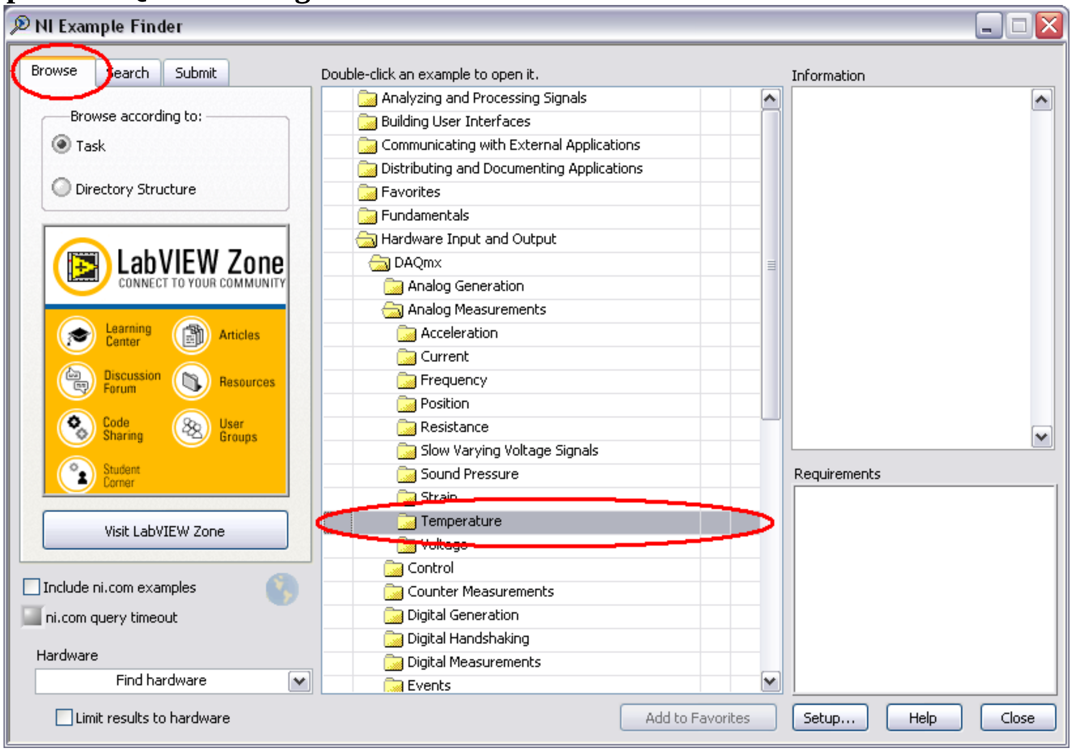
Note: By moving these parameter and setup VIs onto the block diagram, you can now programmatically change their values without having to stop your application and open the Express VI configuration dialog, saving development time and possibly optimizing performance by eliminating unnecessary settings depending on you application.

Using the LabVIEW Example Finder

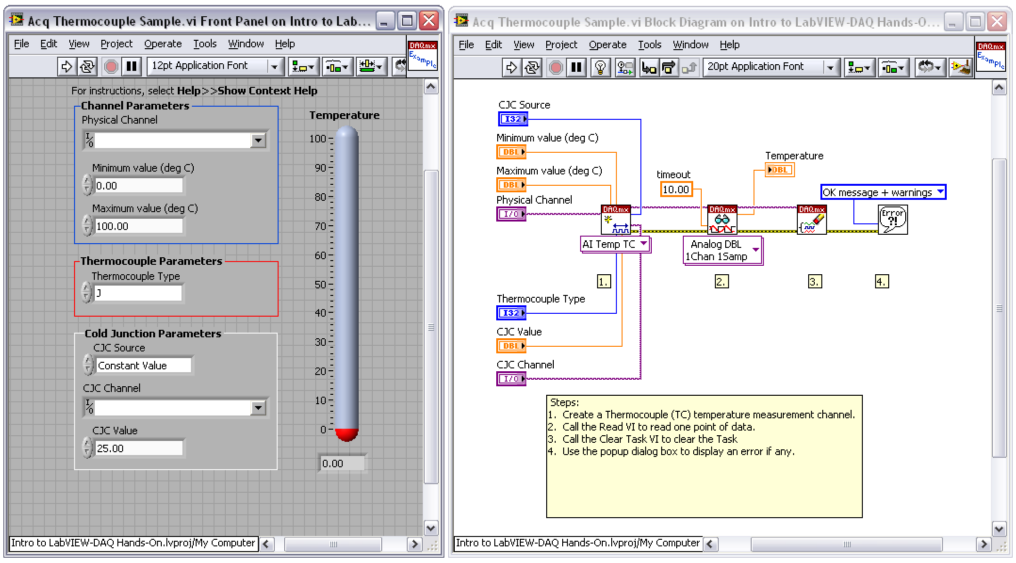
The LabVIEW Example Finder provides hundreds of example application to use as reference or as the starting point for your application.

25. Open the LabVIEW Example Finder to find DAQ examples that use DAQmx standard VIs. Go to **Help» Find Examples…** to launch the LabVIEW Example Finder.

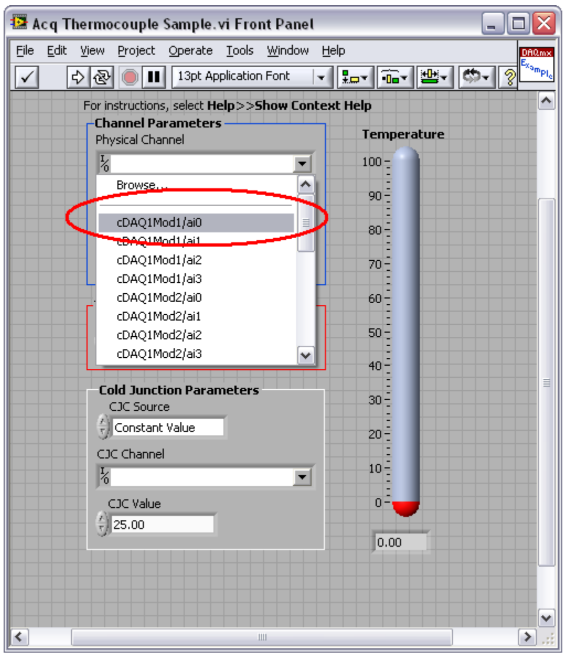
26. Browse to the DAQmx Analog Measurements folder from the Browse tab at **Hardware Input and Output» DAQmx» Analog Measurements**.



27. The following VI will then appear:


Acq Thermocouple Sample.vi Front Panel on Intro to Lab...
Acq Thermocouple Sample.vi Block Diagram on Intro to LabVIEW-DAQ Hands-O...
For instructions, select Help>>Show Context Help
Channel Parameters
Physical Channel
Minimum value (deg C)
0.00
Maximum value (deg C)
100.00
Thermocouple Parameters
Thermocouple Type
J
Cold Junction Parameters
CJC Source
Constant Value
CJC Channel
CJC Value
25.00
Temperature
0.00
CJC Source
Minimum value (deg C)
Maximum value (deg C)
Physical Channel
timeout
10.00
Temperature
OK message + warnings
AI Temp TC
Analog DBL 1Chan 1Samp
Thermocouple Type
CJC Value
CJC Channel
Steps:
1. Create a Thermocouple (TC) temperature measurement channel.
2. Call the Read VI to read one point of data.
3. Call the Clear Task VI to clear the Task
4. Use the popup dialog box to display an error if any.
Intro to LabVIEW-DAQ Hands-On.lvproj/My Computer

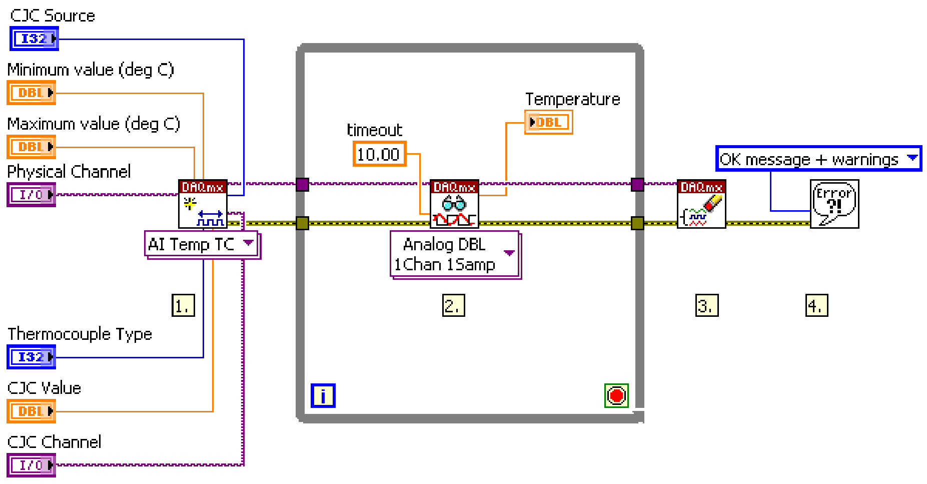
28. Set the Physical Channel to match the CompactDAQ chassis channel and run the application. Expand the physical channel control from the Front Panel and select cDAQ1Mod1/ai0.



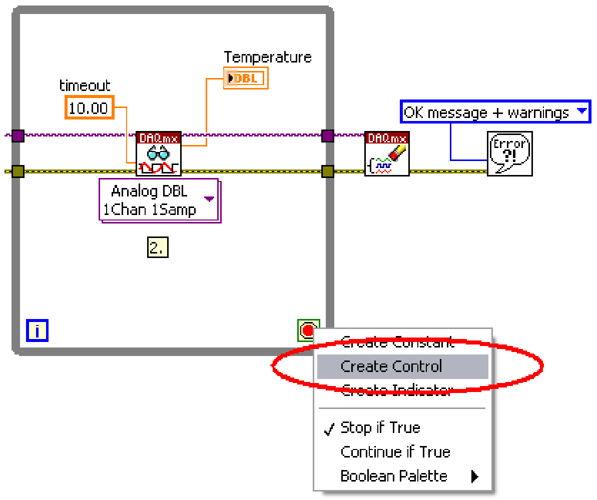
Press the Run button several times while holding and releasing the thermocouple on the CompactDAQ chassis and observe the value change on the front panel.

29. Open the block diagram and examine the code. This VI only uses standard VIs instead of Express VIs, which allows much more customization of inputs and run-time configuration. Acq Thermocouple Sample.vi has no while loop to allow for continuous execution, and the remaining steps of this exercise will focus on adding that functionality.

30. Add a while loop and Stop button to Acq Thermocouple Sample.vi. Right-click on the block diagram to bring up the Functions palette. Find the While Loop on the Programming» Structures palette and drag a while loop over the DAQmx Read.vi. You may need to spread the VIs across the block diagram so that there is room. You can create additional space by holding the Control key and dragging a box on the block diagram or front panel.



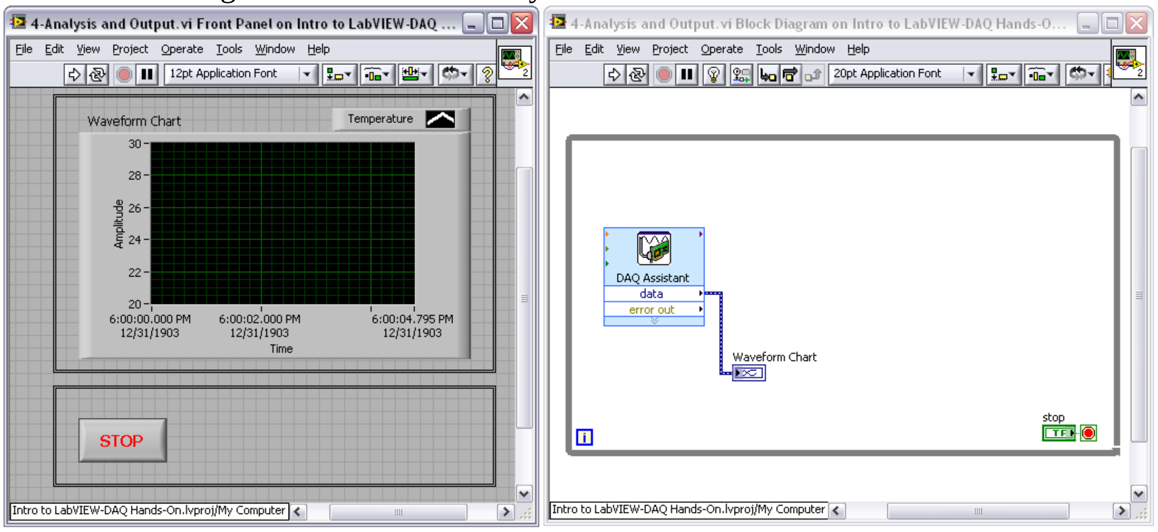
Right click on the While Loop's Conditional terminal and select "Create Control." This automatically wires a Stop button to the terminal.



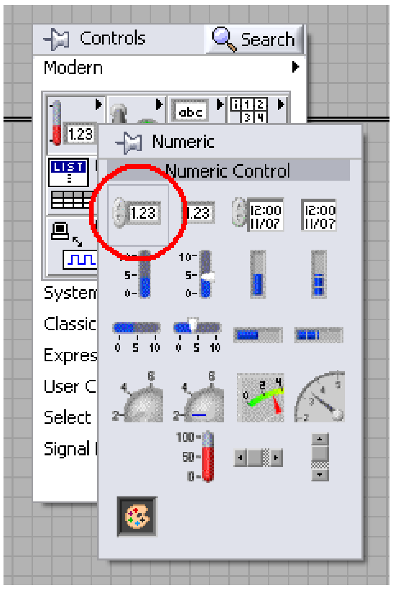
Notice that the Stop button has appeared on the front panel.
31. Run the VI. Acq Thermocouple Sample.vi now runs continuously.
32. Save the customized example VI to the Project. Go to **File» Save As…**, select **Copy» Substitute Copy for Original** and name the VI "Thermocouple Customized Example.vi." Save this VI. This allows for further development without overwriting the LabVIEW example.
End of Exercise 1

Exercise 2: Add Analysis and Digital Output to the DAQ Application

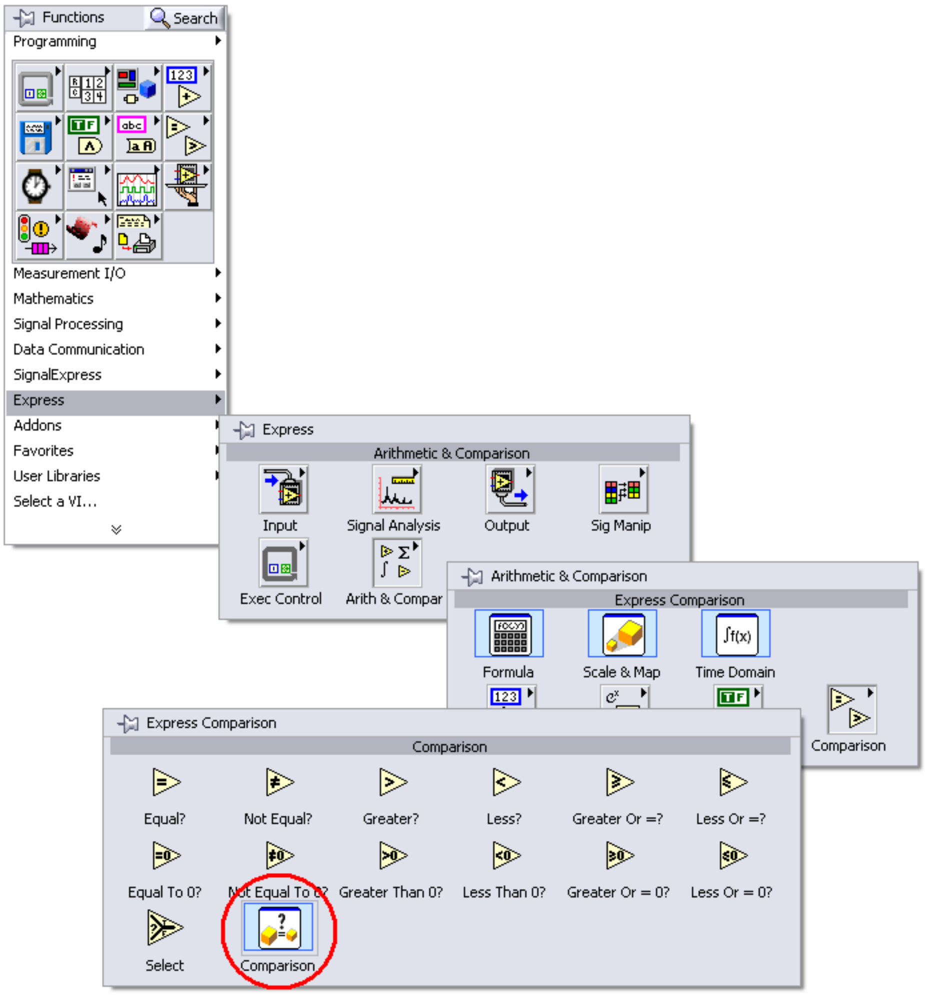
Set up Hardware
1. Confirm that the CompactDAQ chassis is powered on and connected to the PC via the USB cable. If not, or if it is not behaving as expected, repeat steps #1-8 from Exercise #1
LabVIEW Application – Compare signal to user-defined alarm
2. Exercise 2 is functionally the same as the end result of Exercise 1. You can open Exercise 1 to synchronize with the illustrations in this section.. Open 1-Analysis and Output.vi from the Exercises folder in the Project explorer. The VI will appear like the image below, with additional space on the block diagram to add functionality:


4-Analysis and Output.vi Front Panel on Intro to LabVIEW-DAQ ...
File Edit View Project Operate Tools Window Help
12pt Application Font
Waveform Chart
Temperature
30
28
26
24
22
20
Amplitude
6:00:00.000 PM 12/31/1903
6:00:02.000 PM 12/31/1903
6:00:04.795 PM 12/31/1903
Time
STOP
Intro to LabVIEW-DAQ Hands-On.lvproj/My Computer
4-Analysis and Output.vi Block Diagram on Intro to LabVIEW-DAQ Hands-O...
20pt Application Font
DAQ Assistant
data
error out
Waveform Chart
stop
TF

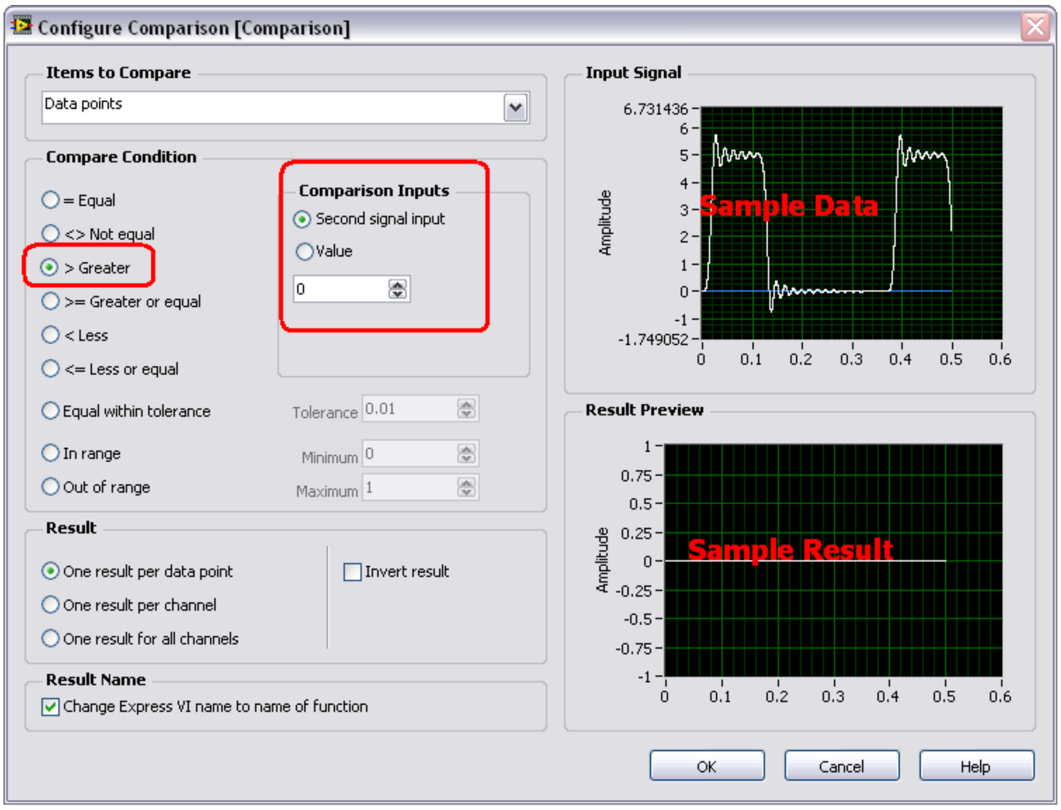
3. Create an alarm that signals if acquired temperature goes above a user-defined level. On the front panel, right-click to open the Controls palette Programming» Numeric and place a numeric control on the front panel.



4. Change the numeric control's name to "Alarm Level." Double-click on the control's label and replace the generic text with "Alarm Level"
6. Use the Comparison Express VI to compare the acquired temperature signal with the Alarm Level control. Switch to the block diagram, right-click on an empty space and open the Functions palette. Place the Comparison Express VI on the block diagram from Functions» Express» Arithmetic & Comparison» Comparison.



7. Once placed on the block diagram, the Comparison Express VI's configuration dialog will appear.


Configure Comparison [Comparison]

Items to Compare: Data points

Compare Condition
- = Equal
- <> Not equal
- [✓ > Greater]
- >= Greater or equal
- < Less
- <= Less or equal
- Equal within tolerance — Tolerance 0.01
- In range — Minimum 0
- Out of range — Maximum 1

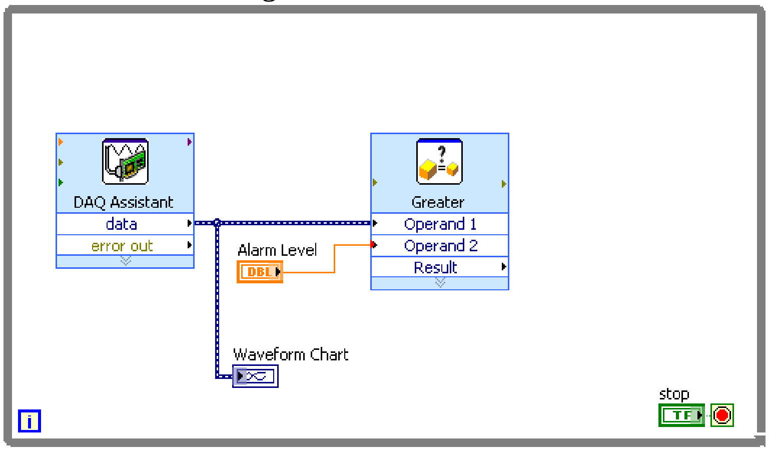
Comparison Inputs
- [✓ Second signal input]
- Value 0

Result
- [✓ One result per data point]
- One result per channel
- One result for all channels
- Invert result

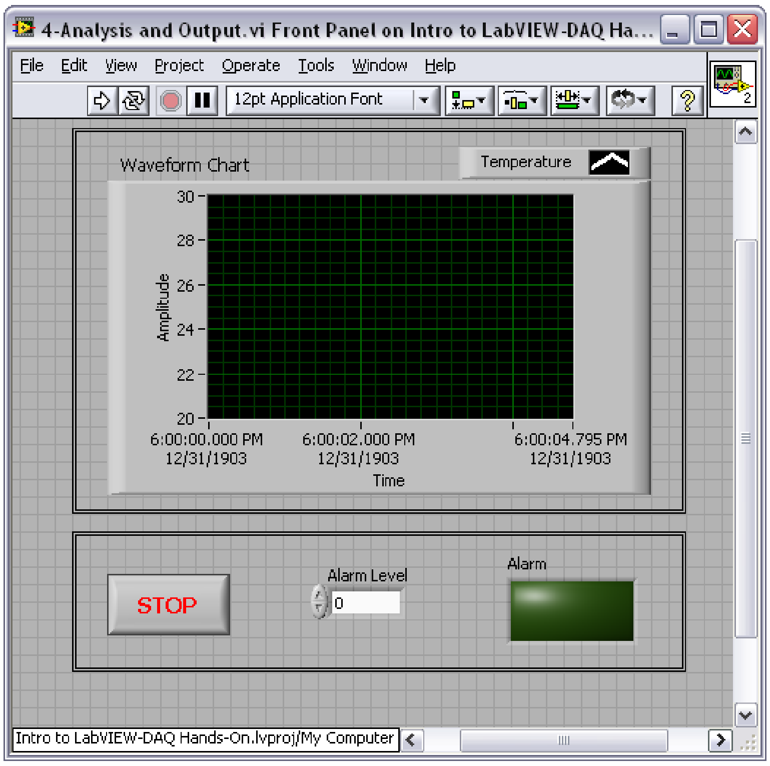
Result Name
- [✓ Change Express VI name to name of function]

Select "> Greater" in the Compare Condition section and "Second signal input" from the Comparison Inputs section then click OK.

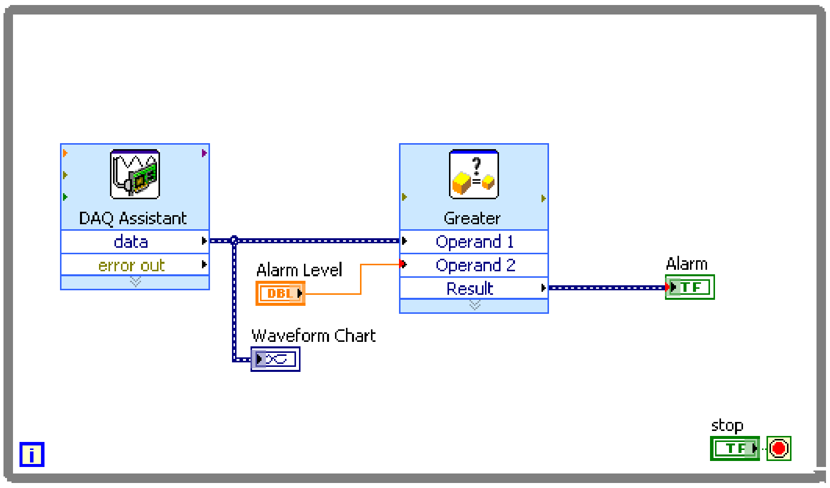
8. Connect the acquired temperature data and Alarm Level inputs to the Comparison Express VI. Hover over the output of the DAQ Assistant until the spool icon appears on your cursor, then left-click and drag you mouse to the Operand 1 input on the Comparison Express VI. Perform the same hover, drag and connect to wire the Alarm Level control and the Operand 2 input on the Comparison Express VI. Your block diagram should now look like this:



9. Display the result of the Comparison Express VI on the front panel. On the front panel, right click, open the Controls palette and add a Square LED indicator. The square LED is found at **Controls» Modern» Boolean**. Resize the Square LED so that it is easier to see and rename it "Alarm." Your front panel should look like this:


4-Analysis and Output.vi Front Panel on Intro to LabVIEW-DAQ Ha...
File Edit View Project Operate Tools Window Help
12pt Application Font
Waveform Chart
Temperature
30
28
26
24
22
20
Amplitude
6:00:00.000 PM 12/31/1903
6:00:02.000 PM 12/31/1903
6:00:04.795 PM 12/31/1903
Time
STOP
Alarm Level
0
Alarm
Intro to LabVIEW-DAQ Hands-On.lvproj/My Computer

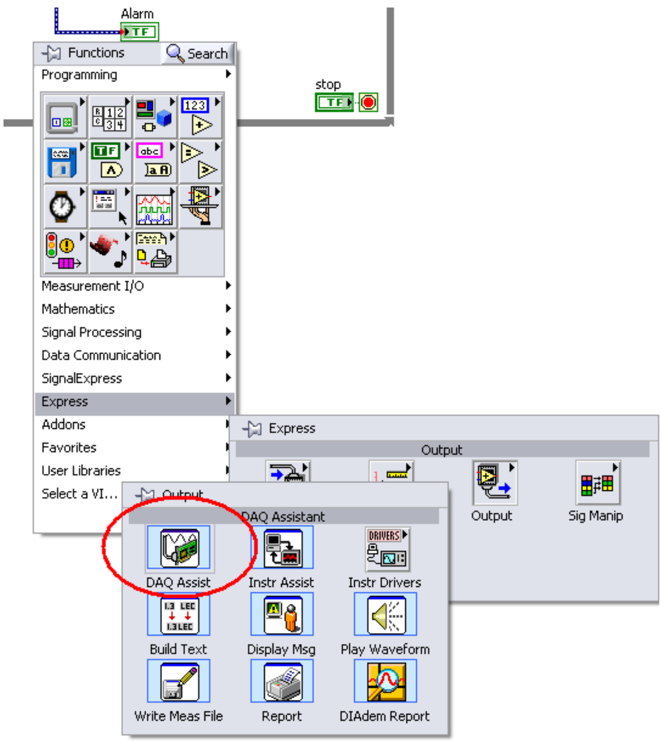
On the block diagram, wire the output of the Comparison Express VI to the input of the Alarm indicator's terminal.



10. Run the application. Press the Run button and then change the Alarm Level control to some level above the current acquired temperature signal. Hold the thermocouple until the temperature exceeds the Alarm Level value. The Alarm LED turns on when the acquired temperature signal goes above the level set on the front panel.

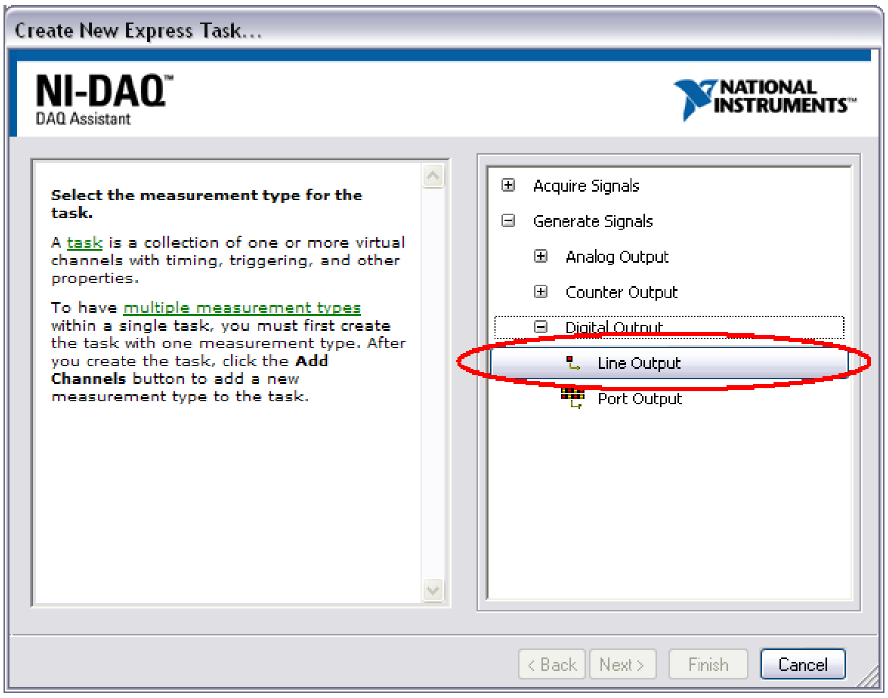
Output Alarm to CompactDAQ Chassis

11. Use another DAQ Assistant Express VI to output Alarm's status to the CompactDAQ's 9472 module. Open the Functions palette on the block diagram and find the DAQ Assistant Express VI at Functions» Express» Output.



<picture of palette w/ DA circled>

12. Select Generate Signals» Digital Output» Line Output from the Create New Express Task… window.



Create New Express Task...

NI-DAQ™
DAQ Assistant

NATIONAL INSTRUMENTS™

Select the measurement type for the task.

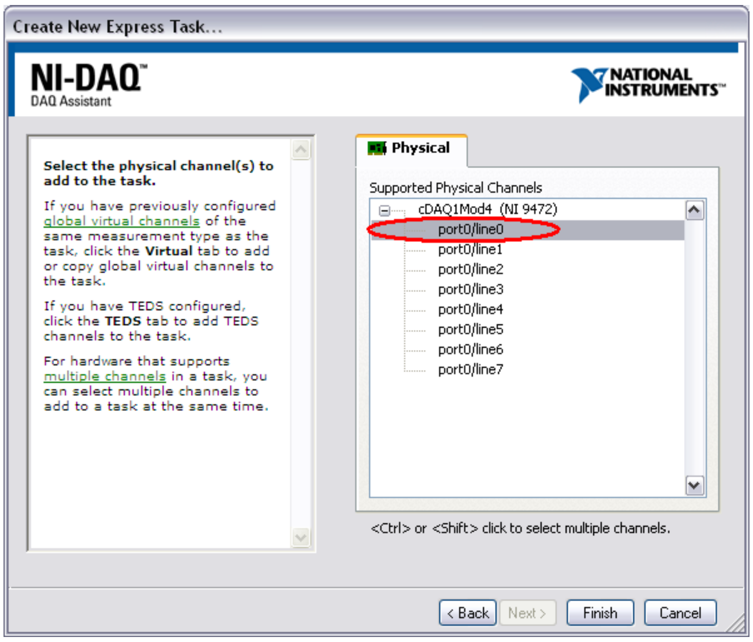
A task is a collection of one or more virtual channels with timing, triggering, and other properties.

To have multiple measurement types within a single task, you must first create the task with one measurement type. After you create the task, click the **Add Channels** button to add a new measurement type to the task.

- Acquire Signals
- Generate Signals
  - Analog Output
  - Counter Output
  - Digital Output
    - Line Output
    - Port Output

< Back | Next > | Finish | Cancel

13. Select the physical channel you want to use as output. Expand the + sign next to cDAQ1Mod4 in the following window and select port0/line0.


Create New Express Task...

NI-DAQ™ DAQ Assistant

Select the physical channel(s) to add to the task.

If you have previously configured global virtual channels of the same measurement type as the task, click the **Virtual** tab to add or copy global virtual channels to the task.

If you have TEDS configured, click the **TEDS** tab to add TEDS channels to the task.

For hardware that supports multiple channels in a task, you can select multiple channels to add to a task at the same time.

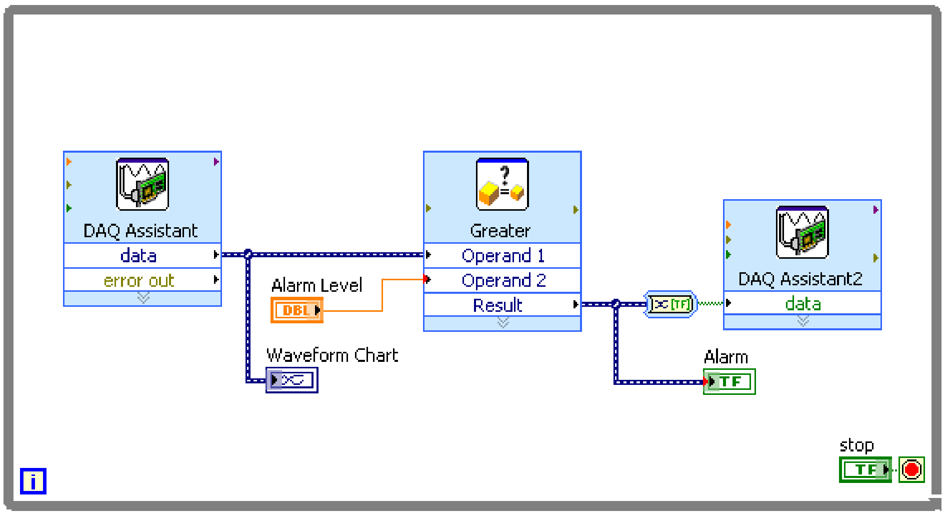
Physical

Supported Physical Channels

- cDAQ1Mod4 (NI 9472)
  - port0/line0
  - port0/line1
  - port0/line2
  - port0/line3
  - port0/line4
  - port0/line5
  - port0/line6
  - port0/line7

<Ctrl> or <Shift> click to select multiple channels.

14. Press OK in the DAQ Assistant window that appears, since all of its settings are correct for the application.
15. Create an additional wire that connects the Comparison Express VI's Result output to the **data** input on the new DAQ Assistant Express VI. A Convert from Dynamic Data function appears automatically. LabVIEW will always try to coerce unlike data types when two nodes are wired together. In this case, the output of the Compare Express VI is a Dynamic Data type, and the input of the DAQ Assistant is Boolean. LabVIEW placed the Convert from Dynamic Data node in between the two nodes so they could be connected. You can double-click the Convert from Dynamic Data to view its configuration. Your block diagram should now look like this:



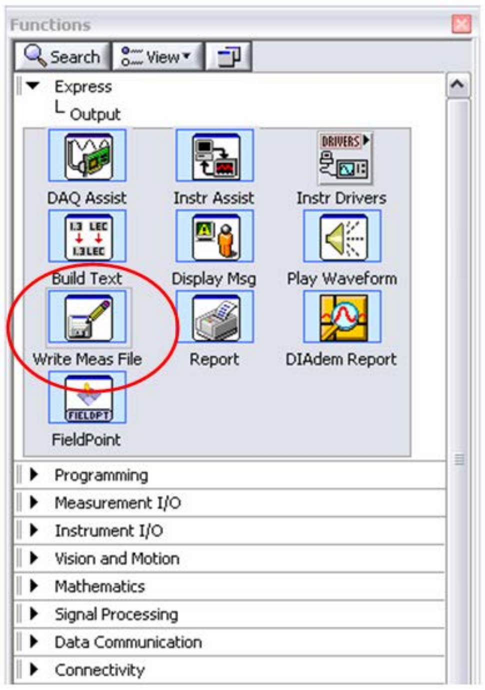
16. Run the VI. Press the Run button. Notice that the LED bank on the CompactDAQ 9472 module turns on and off to match Alarm's value on the front panel.
17. Save and close the VI.

End of Exercise 2

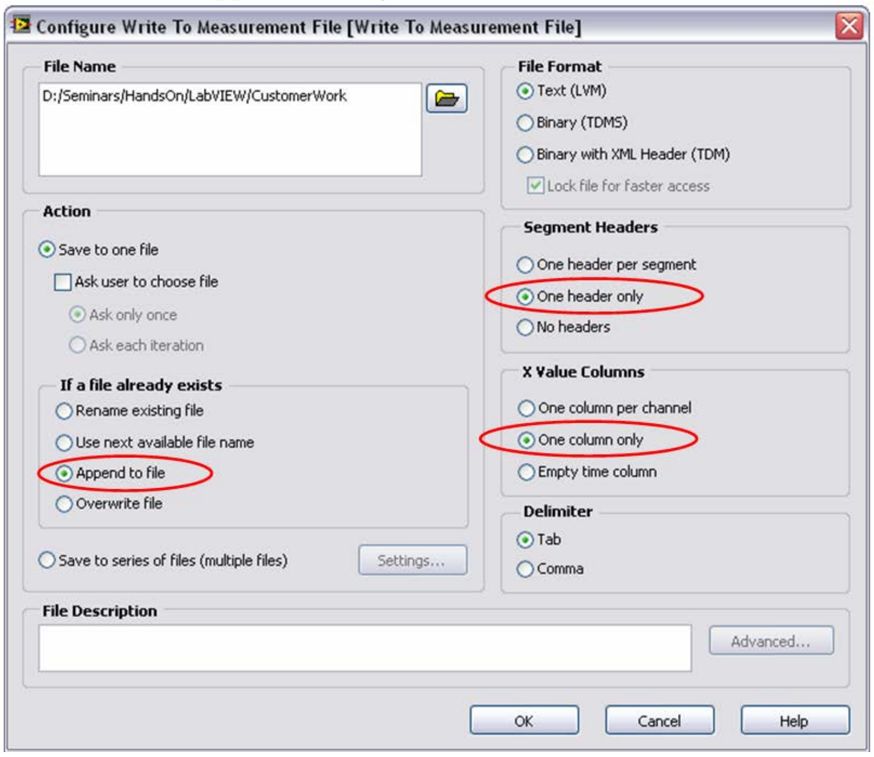
Exercise 3: Writing Data to File with LabVIEW

1. In the Exercise folder in the Project Explorer, open 2-Analysis and Output.vi. We will use the final program from the last exercise as the beginning of this exercise.
2. Right-click on the block diagram and select **Functions» Express» Output» Write to**

Measurement File and place it inside the While Loop on the block diagram.



3. A configuration window will appear. Configure the window as shown below and click OK.



Configure Write To Measurement File [Write To Measurement File]

File Name: D:/Seminars/HandsOn/LabVIEW/CustomerWork

File Format: [✓ Text (LVM)] Binary (TDMS) Binary with XML Header (TDM); [✓ Lock file for faster access]

Action: [✓ Save to one file]; Ask user to choose file; [✓ Ask only once] Ask each iteration

If a file already exists: Rename existing file; Use next available file name; [✓ Append to file]; Overwrite file

Save to series of files (multiple files) Settings...

Segment Headers: One header per segment; [✓ One header only]; No headers

X Value Columns: One column per channel; [✓ One column only]; Empty time column

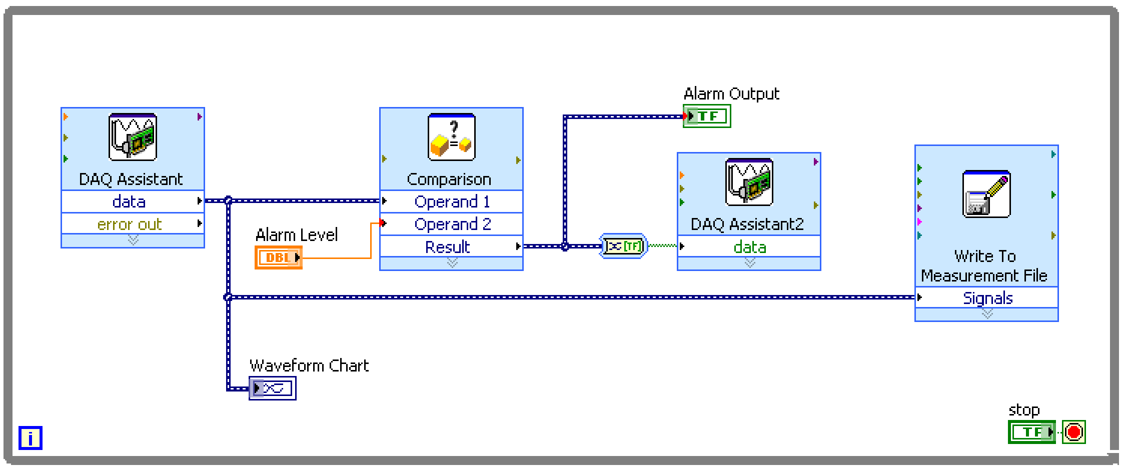
Delimiter: [✓ Tab] Comma

File Description

Advanced...

OK Cancel Help

4. Wire the output of the DAQ Assistant Express VI to the input of the Write to Measurement File Express VI.
5. Your block diagram should now resemble the following figure.



6. Save the VI by using the **File» Save As…** menu, select the **Copy» Open Additional Copy** and name it 3-Write to File.vi.
7. Run the VI momentarily and press STOP to stop the VI.
8. Your file will be created in the folder specified.
9. Open the file using Microsoft Office Excel or Notepad. Review the header and temperature data saved in the file.
10. Close the data file and the LabVIEW VI.

End of Exercise

Exercise 4: Generate, Acquire, Analyze and Display

Generate a sine waveform using the analog output module. Acquire the sine waveform using the analog input module. Perform the appropriate analysis on the acquired waveform to figure out the frequency of the acquired waveform. Finally display the acquired waveform and its frequency. This is a challenge exercise and step by step instructions are not provided, but rather the end goal is given. It is up to you to figure out how to come up with the program to achieve the given task.

End of Exercise

Network lab
Exercise 9

Introduction:

The lab purpose is to introduce you to the concept of event building in High Energy Physics experiments.

Outline:

I. Networking Introduction

Every data acquisition system has, at its core, a computer network to gather and filter collision data from the detector. Usually a DAQ network has one (for small DAQ systems) or a few (for more complex DAQ systems) core switches/routers and lots of pizza box switches in order to connect every computer in the network.

To simulate the core of a DAQ network we will use an HP Procurve switch. Switches map the Ethernet addresses of the nodes residing on each network segment and then allow only the necessary traffic to pass through the switch. When a packet is received by the switch, the switch examines the destination and source hardware addresses and compares them to a table of network segments and addresses. If the segments are the same, the packet is dropped ("filtered"); if the segments are different, then the packet is "forwarded" to the proper segment.

Switches help us connecting computers in the same area which are not too further apart in a LAN (local area network). However standard switches does not care about teams or applications. To be able to divide computers in groups by teams or applications we have to use VLANS. So actually having switches that supports VLANs is having a switched network that is logically segmented on an organizational basis, by functions, project teams, or applications rather than on a physical or geographical basis. What this is saying is that a VLAN is not defined by any physical restrains or needs, it can span an entire country or can be in the same floor in an office. VLANs are formed for administrative purposes and not geographical purposes.

To be able to monitor the networking devices in a network (ex: switches and routers) the most used protocol is SNMP (Simple Network Management Protocol). SNMP exposes management data in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried (and sometimes set) by managing applications. In typical SNMP use, one or more administrative computers have the task of monitoring or managing a group of hosts or devices on a computer network. Each managed system (also called Slave) executes, at all times, a software component called an *agent* (see below) which reports information via SNMP to the managing systems (also called Masters).

The information gathered via SNMP protocol (ex: number of MB/s going in or out from one port, errors, discards, interface speed etc) can be stored in a standard database (like Oracle, MSSQL, MySQL etc) or with the help of RRD files (Round Robin Database). In order to create and store data in an RRD file you will have to use rrdtool application. RRDtool refers to Round Robin Database tool. Round robin is a technique that works with a fixed amount of data, and a pointer to the current element. Think of a circle with some dots plotted on the edge. These dots are the places where data can be stored. Draw an arrow from the center of the circle to one of the dots; this is the pointer. When the current data is read or written, the pointer moves to the next element. As we are on a circle there is neither a beginning nor an end, you can go on and on and on. After a while, all the available places will be used and the process automatically reuses old locations. This way, the dataset will not grow in size and therefore requires no maintenance. RRDtool works with Round Robin Databases (RRDs).

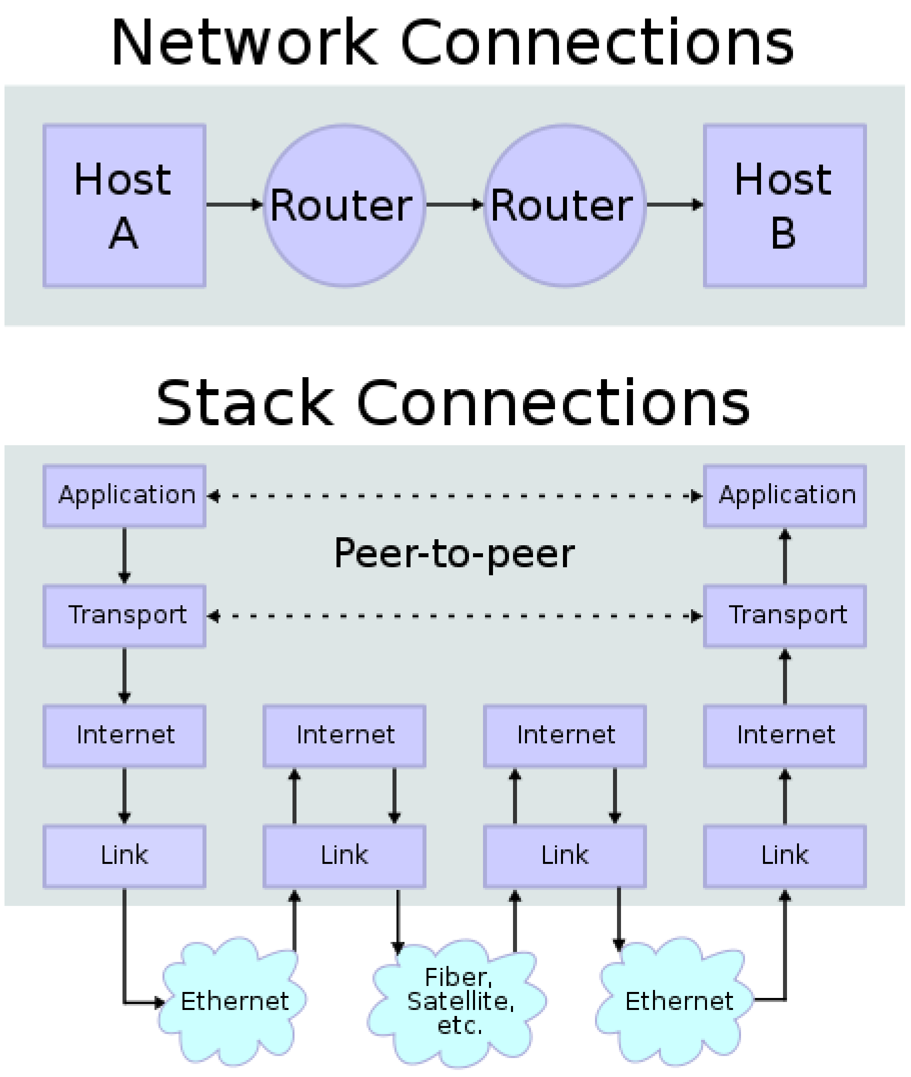


Illustration 1: Data transmission over a routed network

From a protocol point of view typically, the link protocol is E*thernet*, the Internet protocol is *IP*, the transport protocol are mainly *UDP* and *TCP*, and then an application protocol example is *http*. In order for data to be transmitted through the network, we encapsulate them into the protocol related to the layer we are using. This encapsulation consists in adding headers and footers to the data. Then, when a packet passes through the different layers, each layer read its related header and is able to process the following data.

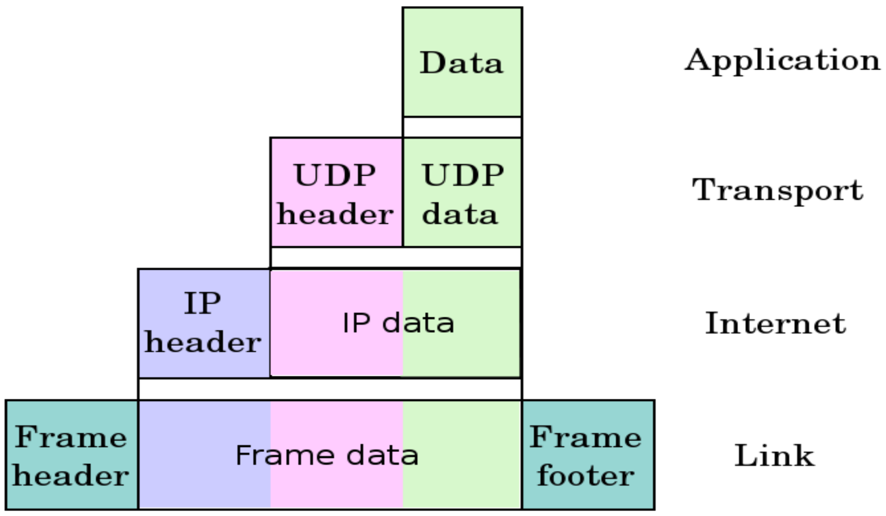


Illustration 2: Protocol stack

II. Event Building Introduction

Large experiments consist of a very complex detector, made of sub-detectors. Each sub-detector has a specific behavior and identifies different particles.

We want to get, for each collision, a picture of the complete detector. So each sub-detector is read-out by a device. The connection between this Readout device and the sub-detector is generally made of custom links which have to be resistant to radiations.

The readout devices perform a first data analysis before formatting them following networking standards, because we're now in a radiation safe area and standard devices are very efficient.

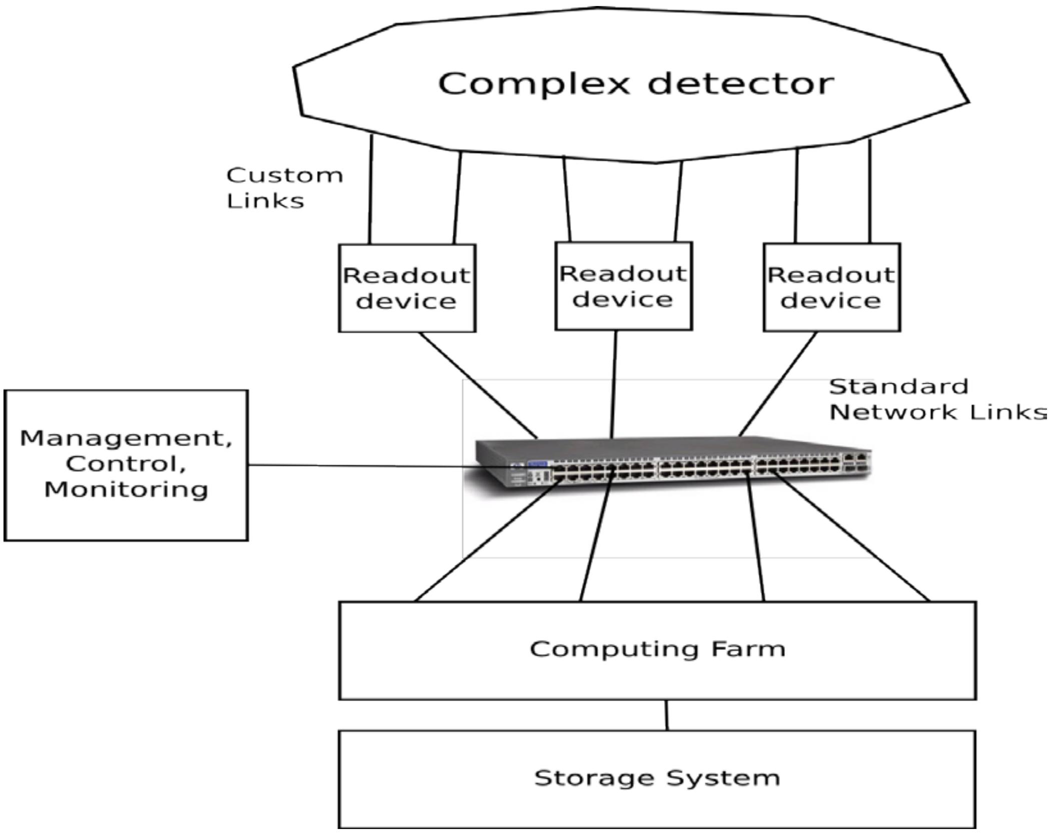
To perform further processing, HEP experiments relies on a computing farm, i.e. each single event will be processed by a single core, which will decide if the event is interesting or not. If the event is interesting, it will be written to a temporary storage system, else it will be discarded. This architecture means that there is no parallelism in the processing.

In order to perform the event building, a computing farm core needs to get the full picture of the detector, i.e. the information from all readout devices.

This is achieved mainly using 2 different “protocols” which are either **push** or **pull**. Pushing means that the readout devices will send their information, for an event i, to a single selected core. This core can be elected according to a round-robin rule. This is quite limited because this core can be busy. An improvement is possible using some back-pressure mechanisms. A core would advertise if it's available or busy.

Pulling consists for a core in requesting the event fragment to each readout device. Therefore only an available core can have made the request. It can be enhanced by requesting only a part of the event, analyzing it and if it looks interesting, requesting the full event.

III. Laboratory Objectives



In a first part you will have to configure your system, mainly the switch, so data can be transmitted from a data injector to the processing computers.

Then you have to set up a simple system to gather traffic information from the switch so that you will be able to monitor the traffic flow in you network.

In the end you will have to implement a simple event building software on the processing computers, which receives data from the network, decode it and then write them to a permanent storage system.

Work plan:

I. Network Configuration Guideline

1. Power up the switch. Connect cables between computers and switch. Check the connectivity light. (It doesn't matter what port numbers you chose to use for connecting the PCs);
2. Login to the network monitoring PC (NETMON) using *user: student, password: student;*
3. Connect the serial cable (DB9-DB9) from the serial port on the switch to the serial port on the NETMON computer;
4. Start "screen" application to connect to the switch:
   1. student> **screen /dev/ttyS0**
   2. sw-daqcluster-c1> **enable (user: admin, password: admin)**
   3. sw-daqcluster-c1# **show running-config**

5. Check that the switch has the ip address set to 10.128.2.2 (netmask 255.255.0.0). If the ip is not set please set it:
   1. sw-daqcluster-c1# **configure terminal**
   2. sw-daqcluster-c1(config)# **vlan 1**
   3. sw-daqcluster-c1(vlan-1)# **ip address 192.168.2.2 255.255.0.0**
6. Make sure that the NETMON PC can request information via SNMP from the switch. You can check that by having a look in the configuration file after the following lines:
   1. ip authorized managers IP NETMASK
   2. management-vlan 1
7. Have a look at the VLANs already configured on the switch (show running config). Are the connected ports all in the same vlan? Is it important to have all the ports in the same vlan? If necessary please make adjustments to where the cables are connected;
8. Try to ping the switch;
9. Try to request some simple information from the switch via SNMP:

   student> **ping 10.128.2.2**
10. Try to request some simple information from the switch via SNMP:

    student> **snmpget –v 2c –c public 10.128.2.2 sysDescr.0**
11. Use wireshark to have a detailed look at the network traffic (hint: use menu command on the switch);
12. Use wireshark to have again a detailed look at the network traffic.

II. Network Monitoring Guideline

1. From the console change directory to swmon

   student> **cd ~/swmon**
2. Have a look at the following files:

   switch_stats_create.sh, switch_stats.sh, switch_graph.sh.
3. Run switch_stats_create.sh and check for newly created switch_stats.rrd file:

   student> **./switch_stats_create.sh**
4. Run switch_stats.sh to start polling the switch:

   student> **./switch_stats.sh &**
5. Open another console and run switch_graph.sh in to start generating traffic plots.

   student> **./switch_graph.sh &**
6. Open index.html file in a browser (ex: firefox)
7. Modify **switch_stats_create.sh, switch_stats.sh, switch_graph.sh** to start monitoring only on the active ports (the one connected to the event building farm and the event injector)

Decoding for the SNMP OIDs related to traffic information

OID .1.3.6.1.2.1.2.2.1.10 =

.iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).interfaces(2).ifTable(2).ifEntry(1).ifInOctets(10)

OID .1.3.6.1.2.1.2.2.1.16 =

.iso(1).org(3).dod(6).internet(1).mgmt(2).mib-2(1).interfaces(2).ifTable(2).ifEntry(1).ifOutOctets(16)

II. Event Building Guideline

Our data acquisition system uses the network stack presented in Illustration 4. The transport layer implements MEP (Multi Event Packet). It is a simple protocol which stores several event fragments, from *1* to *m*.

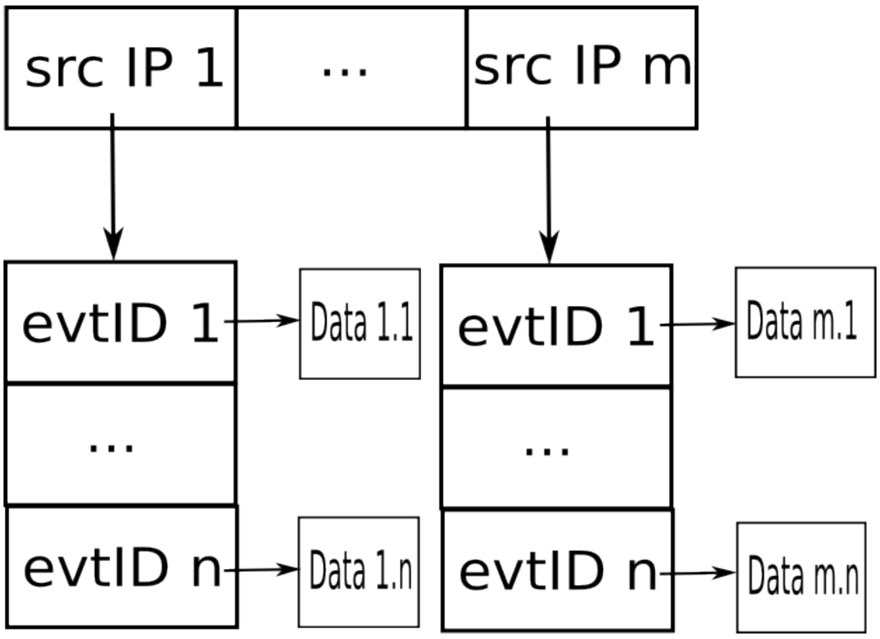
Different sources, from *1* to *n*, are transmitting these packets to the event-builder. It means that an event builder computer will receive *n* packets of *m* events and will have to write them to process them.

In order to store these data and to be ready to write them, or to discard them in case of problems, you'll need to store them in a data structure. An example of a solution is presented in Illustration 5.

The language used for this software is Python. You will find in annexes several information to help you for the implementation.

Event Building protocolIllustration 3: Event Building protocol description. Ethernet in grey, IP in yellow, MEP in green, fragment header in teal and raw data in white.

Bits 31:24	Bits 23:16	Bits 15:8	Bits 7:0
DA[7:0]	DA[15:8]	DA[23:16]	DA[31:24]
DA[39:32]	DA[47:40]	SA[7:0]	SA[15:8]
SA[23:16]	SA[31:24]	SA[39:32]	SA[47:40]
L/T[15:8]	L/T[7:0]	Version/ IHL	Type of Service
Total Length[15:8]	Total Length[7:0]	Identification[15:8]	Identification[7:0]
Flags/Fragment Offset[11:8]	Fragment Offset[7:0]	Time to Live	Protocol = 0xF2
Header Checksum[15:8]	Header Checksum[7:0]	IP-SA[31:24]	IP-SA[23:16]
IP-SA[15:8]	IP-SA[7:0]	IP-DA[31:24]	IP-DA[23:16]
IP-DA[15:8]	IP-DA[7:0]	L0ID / L1ID[7:0]	L0ID / L1ID[15:8]
L0ID / L1ID[23:16]	L0ID / L1ID[31:24]	Number of Events[7:0]	Number of Events[15:8]
Total Length[7:0]	Total Length[15:8]	Partition ID [7:0]	Partition ID [15:8]
Partition ID [23:16]	Partition ID [31:24]	EventID 1[7:0]	EventID 1[15:8]
Len 1[7:0]	Len 1[15:8]	Data1[7:0]	Data1[15:8]
Data1[23:16]	Data1[31:24]	…	…
…	…	Event ID2[7:0]	Event ID2[15:8]
Len 2[7:0]	Len 2[15:8]	Data2[7:0]	Data2[15:8]
Data2[23:16]	Data2[31:24]	…	…
…	…	…	…
…	…	…	…



Annex 1 (event building annex)

A. How to use a raw socket

```
import socket

# the public network interface
HOST = socket.gethostbyname(socket.gethostname())

# create a raw socket and bind it to the public interface
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_IP)
s.bind((HOST, 0))

# include IP headers
s.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)

# receive all packages
s.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

# receive a package
print s.recv(65565)

# disabled promiscuous mode
s.ioctl(socket.SIO_RCVALL, socket.RCVALL_OFF)
```

B. How to convert a 32-bit packed IP address to its standard dotted string

`inet_ntoa` (*packed_ip*)

Convert a 32-bit packed IP address (a string four characters in length) to its standard dotted-quad string representation (e.g. '123.45.67.89').

If the string passed to this function is not exactly 4 bytes in length, `socket.error` will be raised.

C. How to decode network frames

`struct.unpack(`*fmt*, *string*`)` ¶

Unpack the string (presumably packed by `pack(fmt, ...)`) according to the given format. The result is a tuple even if it contains exactly one item. The string must contain exactly the amount of data required by the format (`len(string)` must equal `calcsize(fmt)`).

Format	C Type	Python
x	Pad byte	No value
c	Char	String of length 1
b	Signed char	Integer
B	Unsigned char	Integer
?	_Bool	Bool
h	Short	Integer
H	Unsigned short	Integer
i	Int	Integer or long
l	Long	Integer
L	Unsigned long	Long
q	Long long	Long
Q	Unsigned long long	long
f	float	float
d	double	float
s	char[]	string
p	char[]	string
P	Void *	long

Unpacking data, you have to care about the endianness, or byte order.
They are mainly 2 types: big-endian and little-endian.
With the exemple of storing 0xABCD in memory, with increasing address from right to left. Using 8 bit atomic words:

Big-Endian	A	B	C	D
Little-Endian	D	C	B	A

Using 16 bit atomic words:

Big-Endian Little-Endian	AB CD	CD AB

Character	Byte order	Size and alignment
@	Native	Native
=	Native	Standard
<	Little-endian	Standard
>	Big-endian	Standard
!	Network = big-endian	Standard

D. How to use a dict

A *mapping* object maps hashable values to arbitrary objects. Mappings are mutable objects. There is currently only one standard mapping type, the *dictionary*. (For other containers see the built in list, set, and tuple classes, and the collections module.)

A dictionary's keys are *almost* arbitrary values. Values that are not hashable, that is, values containing lists, dictionaries or other mutable types (that are compared by value rather than by object identity) may not be used as keys. Numeric types used for keys obey the normal rules for numeric comparison: if two numbers compare equal (such as `1` and `1.0`) then they can be used interchangeably to index the same dictionary entry. (Note however, that since computers store floating-point numbers as approximations it is usually unwise to use them as dictionary keys.)

Dictionaries can be created by placing a comma-separated list of `key: value` pairs within braces, for example: `{'jack': 4098, 'sjoerd': 4127}` or `{4098: 'jack', 4127: 'sjoerd'}`, or by the dict constructor.

class `dict`([*arg*])¶

Return a new dictionary initialized from an optional positional argument or from a set of keyword arguments. If no arguments are given, return a new empty dictionary. If the positional argument *arg* is a mapping object, return a dictionary mapping the same keys to the same values as does the mapping object. Otherwise the positional argument must be a sequence, a container that supports iteration, or an iterator object. The elements of the argument must each also be of one of those kinds, and each must in turn contain exactly two objects. The first is used as a key in the new dictionary, and the second as the key's value. If a given key is seen more than once, the last value associated with it is retained in the new dictionary.

If keyword arguments are given, the keywords themselves with their associated values are added as items to the dictionary. If a key is specified both in the positional argument and as a keyword argument, the value associated with the keyword is retained in the dictionary. For example, these all return a dictionary equal to `{"one": 2, "two": 3}`:

1. `dict(one=2, two=3)`
    2. `dict({'one': 2, 'two': 3})`
    3. `dict(zip(('one', 'two'), (2, 3)))`
    4. `dict([['two', 3], ['one', 2]])`

The first example only works for keys that are valid Python identifiers; the others work with any valid keys.

These are the operations that dictionaries support (and therefore, custom mapping types should support too):

`len(d)`
  Return the number of items in the dictionary *d*.

`d[key]`

  Return the item of *d* with key *key*. Raises a KeyError if *key* is not in the map.

`d[key] = value`
  Set `d[key]` to *value*.

`del d[key]`
  Remove `d[key]` from *d*. Raises a KeyError if *key* is not in the map.

`key in d`

  Return `True` if *d* has a key *key*, else `False`.

`key not in d`

  Equivalent to `not key in d`.

`iter(d)`
  Return an iterator over the keys of the dictionary. This is a shortcut for `iterkeys()`.

`clear()`¶
  Remove all items from the dictionary.

`copy()`¶
  Return a shallow copy of the dictionary.

`fromkeys(`*seq*[, *value*])¶

  Create a new dictionary with keys from *seq* and values set to *value*.

  `fromkeys()` is a class method that returns a new dictionary. *value* defaults to `None`.

`get(`*key*[, *default*])¶
  Return the value for *key* if *key* is in the dictionary, else *default*. If *default* is not given, it defaults to `None`, so that this method never raises a KeyError.

`has_key(`*key*)¶
  Test for the presence of *key* in the dictionary. `has_key()` is deprecated in favor of `key in d`.

`items()`¶

Return a copy of the dictionary’s list of `(key, value)` pairs.

`iteritems()`¶

Return an iterator over the dictionary’s `(key, value)` pairs. See the note for dict.items().

Using `iteritems()` while adding or deleting entries in the dictionary may raise a RuntimeError or fail to iterate over all entries.

`iterkeys()`¶

Return an iterator over the dictionary’s keys. See the note for dict.items().

Using `iterkeys()` while adding or deleting entries in the dictionary may raise a RuntimeError or fail to iterate over all entries.

`itervalues()`¶

Return an iterator over the dictionary’s values. See the note for dict.items().

Using `itervalues()` while adding or deleting entries in the dictionary may raise a RuntimeError or fail to iterate over all entries.

`keys()`¶
Return a copy of the dictionary’s list of keys. See the note for dict.items().
`pop`(*key*[, *default*])¶

If *key* is in the dictionary, remove it and return its value, else return *default*. If *default* is not given and *key* is not in the dictionary, a KeyError is raised.

E. How to use binary files to store data in a flatten format

```
import os

outFd= os.open("test.bin", os.O_RDWR | os.O_CREAT)

os.write(outFd,"¥xFe¥xed¥xba¥xbe")

os.close(outFd)
```

```
hexdump test.bin
0000000 edfe beba
0000004
```

Another way:

```
dataFile = open("test.bin",  "w" )

dataFile.write("¥xFe¥xed¥xba¥xbe")

dataFile.close()
```

Annex 2 (network monitoring annex)

A. Introduction

The bash scripts can be used to retrieve various statistics from an SNMP compliant device (including networking switches). The main purpose is to be used in educational activities.

Every SNMP compliant switch can provide, via SNMP protocol, various information such as traffic, erros, transferred packets etc. All the values are available in real time. If the information is read from a switch periodically, and stored in a database, then various historical graphs can be plotted for future use.

B. Scripts

a. switch_stats_create.sh

Creates an RRD file for storing data gathered from the switch.

For this task the following applications/libraries must be installed:

- rrdtool application

b. switch_stats.sh

Starts interogating the switch every 5 seconds about traffic information.

Then stores the values in the rrd file.

For this task the following applications/libraries must be installed:

- rrdtool application
- net-snmp library

c. switch_graph.sh

Will create/update 5 png images using the rrd file used to store traffic info.

The images are stored in "./graphs" folder.

Each of the png images will display information related to one of the switch ports (input and output traffic in Bytes/s).

For this task the following applications/libraries must be installed:

- rrdtool application

d. index.html

A simple web page for displaying generated traffic plots.

C. Usage

First the "switch_stats_create.sh" script should be run. This will create a file named "switch_stats.rrd".

```
server > ./switch_stats_create.sh
```

The second step is to start (in background eventually) the switch_stas.sh and switch_graph.sh (doesn't matter the order):

```
server > ./switch_stats.sh

server > ./switch_graph.sh
```

The third step is to watch images created in ./graphs folder or to open the index.html file with a browser (IE, Firefox etc).

D. Links

http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol

http://oss.oetiker.ch/rrdtool/

http://tldp.org/LDP/abs/html/

http://www.freesoft.org/CIE/Topics/108.htm

Lab 11: Storage configuration and installation

TDAQ 2012, Cracow, Poland

Overview

The aim of this lab is to provide an overview about how to configure a storage setup of a data acquisition system.

Objectives

At the end of this lab, you will have to be able to:

- Configure the RAID configuration of your disks set
- Partition a storage unit
- Install a file system on your disks set
- Mount the device
- Distinguish among the different storage connection systems

Activities

The lab consist of two elements:

1. Hardware RAID configuration
2. Software RAID configuration

Evaluation

No evaluation is foreseen.

Materials

- Linux man pages
- *"The RAID book. A Storage System Technology Handbook", Paul Massiglia*
- *"Managing RAID on Linux", Derek Vadala*

Other Resources

Articles in scientific journals e.g., *"The CMS event builder and storage system", Journal of Physics: Conference Series Volume 219 Part 2, 2010 J. Phys.: Conf. Ser. 219 022038 doi:10.1088/1742-6596/219/2/022038*

1. Hardware RAID configuration

   a. Reboot the PC and enter the Setup Menu of the 3ware controller with 4 drives (press Alt+3 when RAID controller info appears).
   b. Create one unit of 4 drives with RAID5 configuration.
      - Leave the default values except for the RAID configuration.
      - You may need to delete the existing unit before proceeding.
      - Ignore the message about losing data and continue.
      - Write down the total capacity of the unit created.
      - Save the configuration.
      - Wait for the complete boot of the machine.
   c. Login to the machine with the root username and password.
   d. Open a terminal and check to which special device file /dev, associated to the physical unit, the unit has been assigned to in the operating system.
      - You can do it by looking for the occurrences of the keyword "sd" in the output of the dmesg linux command. You may use the command man dmesg to know what information it provides.
   e. Use parted to create one partition of 100% of the block device /dev/sdx. Parted is an interactive command.

```
$ parted /dev/sdb                 if "sdb" is the block device associated to your storage hardware
```

      You will first have to label the device. Type: mklabel msdos (in the parted prompt). You can then create the partition. Type: mkpart primary xfs 0 100% (in the parted prompt).
   f. Create an xfs file system on the /dev assigned to the unit created with inode size=512.
      - You will have to use the mkfs.xfs utility. Look at the man page to see how to set the options.
   g. Mount the device on the /array directory. In this way you are making the device created available to the file system. Use the mount command to assign the device /dev/sdx to the /array directory.
   h. Note down the occupancy (in human readable format) of the mounted point/directory. You will have to use the df linux command.

      Question: Why the capacity of /array is about 700 GB instead of 1TB?
   i. At this point the system has been installed and configured.

2. Software RAID configuration

The following steps will configure a RAID system via software. In order to do that we have to export the four units to the operating system so that they will be treated as independent partitions on which to create a RAID5 configuration.

a. Reboot the PC and enter the Setup Menu of the 3ware controller with 4 drives (press Alt+3 when RAID controller info appears).
b. Create four units of one disk each.
c. Login to the machine with the root username and password.
d. Open a terminal and check to which special devices file /dev, associated to the physical units, the units have been assigned to in the operating system.
   - You will see that this time there are four different devices /dev/sdx.
e. Use parted to create one partition of 1% of every block device /dev/sdx (look point 1.e).
f. Create a RAID5 configuration using mdadm command.
g. Check the rebuild status of the RAID (use both ways).

```
$ mdadm --detail /dev/md0          to check the status of the array

$ cat /proc/mdstat                 other way to check the status of the array
```

h. When the rebuild has finished, create an xfs file system on the unit created.
i. Mount the device on the /array directory.
j. Note down the occupancy (in human readable format) of the mounted point/directory.
k. Your RAID system is now ready to be used.
l. Copy a file that is at least three times bigger that the block size used in the RAID configuration.
m. Disconnect one disk from your pc and try to write and/or read into the /array. You can still do it because the RAID5 configuration makes your unit tolerant up to one disk failure.

```
$ mdadm --detail /dev/md0          to see which is the disk you have disconnected (for example /dev/sdc)
```

n. You can now reconnect the disk and rebuild the array.

```
$ mdadm -r /dev/md0 /dev/sdc1      to remove the "faulty" drive from the RAID set

$ mdadm -a /dev/md0 /dev/sdc1      to add a new drive from the RAID set and associate it to /dev/sdc1
```

   Wait for the rebuild to complete (mdadm --detail /dev/md0).

o. Try to disconnect two drives and write and/or read to /array. The system is broken even if you reconnect both drives. Try to remove and add them again. The rebuild process will not start because the system does not have enough information to rebuild the array. Your data is lost.

DAQ Online Software

Exercise 12

Introduction

Data Acquisition systems (DAQ) are large and heterogeneous infrastructures responsible for collecting, filtering and transferring experimental data from detectors to storage systems. DAQ systems typically rely on a large, distributed computing environment with thousands of software applications running concurrently, ranging from readout modules in VME creates to HLT processes in computer farms.

The Online Software is the software framework used by all DAQ applications that provides the means for controlling, configuring and monitoring the whole DAQ infrastructure.

This exercise presents to students the roles of the online software framework and introduces the concepts of controlling and configuring DAQ processes in such a distributed environment. This exercise will focus on the synchronization of DAQ processes, the start up and shutdown operations according to the finite state machine transitions and on the supervision of the whole system behavior.

Outline

Student will work on developing a distributed health monitoring application meant to collect information about machine health (CPU, Memory usage, etc.) from a set of hosts. The application is composed of a set of different independent sensors that simulate our data acquisition applications. The execution flow is managed by a central controller. Ad-hoc monitoring applications will be used to gather and aggregate data. Sensor applications have to work accordingly to a simple Finite State Machine. Students will develop the tool relying on the control and configuration capabilities provided by a simplified version of a real online system.

Student will learn about the most common situations in controlling DAQ applications in a distributed environment and how they can be addressed. They will also learn about the main capabilities provided by the online software system in a DAQ framework.

1. Step 1: State-aware application

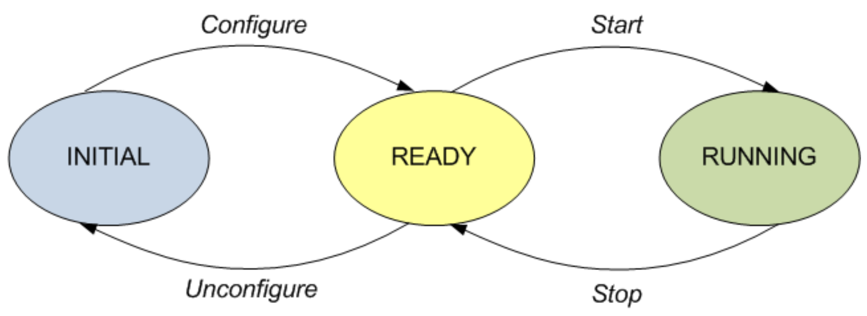


Fig.1 The sensor's Finite State Machine

The first part of the exercise focus on the role of a controlled application. More specifically, students have to work on one of the sensors of the distributed monitoring tool. This sensor has to behave according to the Finite State Machine presented in Fig.1. So, a sensor has to be able to receive command, to perform different tasks depending on the state it's in, and transit from one state to another in a proper way. The command will be sent by a command_sender application residing on a different machine, so the usage of distributed inter process communication technology will be explained to the students.

Brief description of expected behavior in the different states:

- INITIAL: simple error checks
- READY: the sensor reads the host name of the machine it's running on and make it available for publishing
- RUNNING: the sensor periodically reads the CPU load every 2 seconds and makes it available for publishing, together with the total running time

Students will start writing the application (in C++) from a pre-written skeleton and will code the missing pieces following the instructions. Once the sensor is ready, students have to integrate it in the simple online framework of the monitoring tool. At this stage the role of the Controlled generic interfaces will be explained and the application will be tested in a standalone configuration.

2. Step 2: Controller

The second part of this exercise focus on the role of the Controller.

The distributed monitoring application has to manage and gather information from a set of sensors running on different machines. All these sensors have to be properly configured and running at the same time to provide meaningful data, and this introduces the need for a Controller entity to manage the control flow. The main role of a Controller, as explained in Fig.2, is to forward commands to a set of children applications.   But it also has to check the proper execution of FSM transition, deal with common problems, etc.

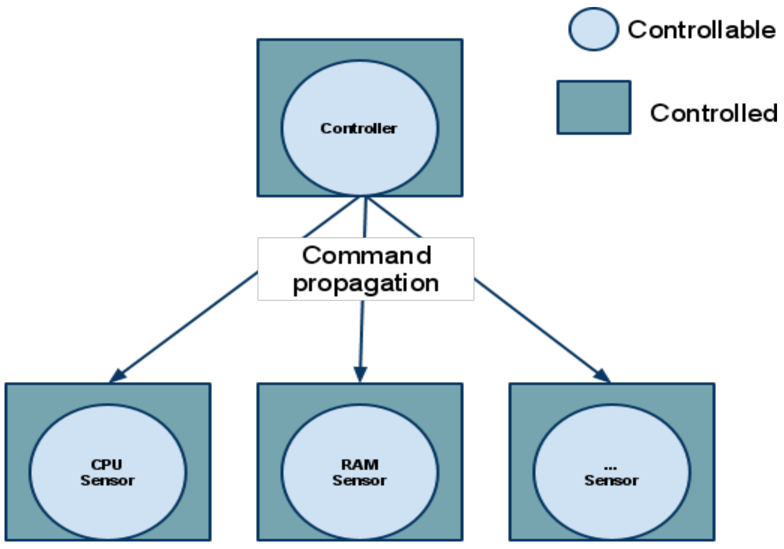


Fig.2 The main Controller

Starting from a controller skeleton, students will develop a custom controller to control the sensor applications. After that, the most common problematic situations will be presented to students and proper solutions will be investigated.