ISOTDAQ 2017 lab book

Tutors and their labs

1. VMEbus programming – Markus Joos
2. NIM – Francesca Pastore & Andrea Negri
3. NIM & scintillator – Kostas Kordas
4. Muon DAQ – Enrico Pasqualucci & Luca Galli
5. FPGA basics – Petr Zejdl
6. MicroTCA – Joschka Lingemann & Alessandro Thea
7. LabView DAQ – Cristovao Barreto
8. ADC basics for TDAQ – Manoel Barros Marin
9. Networking for Data Acquisition Systems – Silvia Fressard-Batraneanu & Fabrice Le Goff
10. Micro controllers – Mauricio Feo
11. Storage systems – Christophe Haen & Tommaso Colombo
12. Control of DAQ systems – Giovanna Lehmann-Miotto & Roland Sipos
13. FPGA & LabVIEW – Maha Moatemri (National Instruments)

The main page of the 2017 ISOTDAQ School is `https://indico.cern.ch/event/557251/`

For up-to-date information on times and places, please see `https://indico.cern.ch/event/557251/timetable/`

Local organizers

- Andrea Borga, Electronics Department, **emergency phone +31 (0)630 553462**
- Olya Igonkina, ATLAS group
- Jos Vermeulen, ATLAS group
- Ruud Kluit, Electronics Department
- Peter Janswijer, Electronics Department

Local support

- Joan Berger, Secretariat
- Kees Huyser, Communication, editor

Advisory committee

- Gergely Barnaföldi (Wigner Research Centre, Budapest)
- Markus Joos (CERN)
- Krzysztof Korcyl (Polish Academy of Sciences)
- Kostas Kordas (Aristotle University of Thessaloniki)
- Livio Mapelli (CERN)
- Niko Neufeld (CERN)
- Enrico Pasqualucci (INFN Roma)
- Francesca Pastore (Royal Holloway - University of London)
- Hannes Sakulin (CERN)
- Christoph Schwick (CERN)
- Andre Sznajder (Universidade do Estado do Rio de Janeiro)
- Gokhan Unel (University of California - Irvine)

Contents

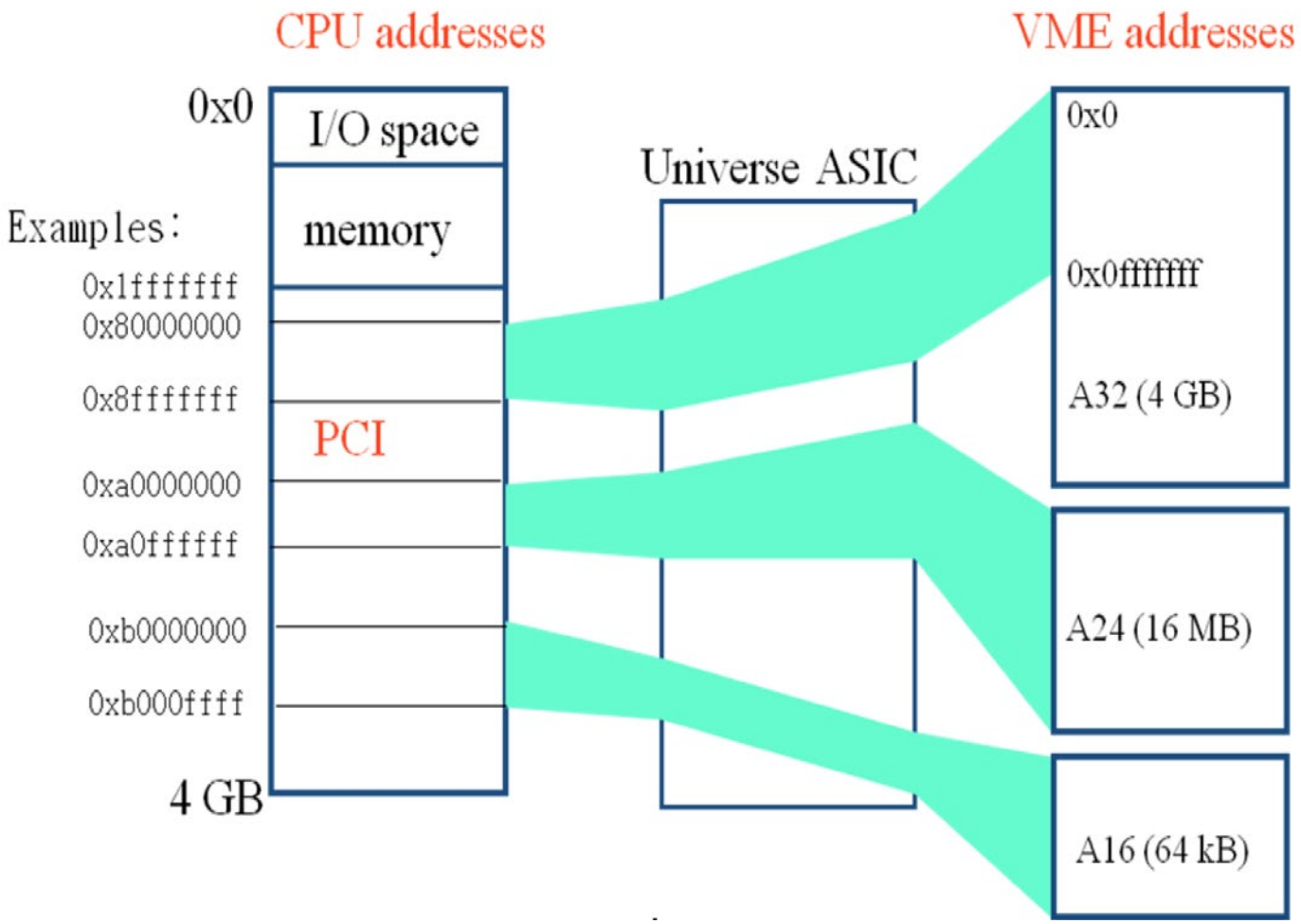
Exercise 1

VMEbus programming

Introduction

For the moment forget what you (hopefully) have learnt about the VMEbus protocol and the details of the H/W. For this exercise you have to look at a VMEbus slave as if it was a piece of memory in your PC. The purpose of this exercise is to demonstrate that in some respects there is little difference between internal and external memory; as far as the programming is concerned. The exercise also shows the differences between the two types of memory.

What is important to understand is that the VMEbus memory has to be mapped into the (virtual) address space of a user process before it can be accessed. This ties 3 busses together: CPU, PCI and VMEbus as shown in the picture below.



The first part of the exercise is to figure out how to create the appropriate mappings for the type of VMEbus access that you have to do. Then you actually transfer the data. This is done in single cycle mode which means that the CPU controls the data transfer.

In the second part of the exercise you will perform block transfers (DMA). This requires a different programming technique since it is not the CPU that moves the data but an external device (a DMA controller). Such DMA controllers are not VMEbus specific. You find them everywhere (e.g. in Network interfaces, disk controllers, USB devices, etc.)

Before you start you should be able 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?

1. On the VMEbus single board computer log on with the DAQ school account (daqschool / g0ldenhorn).
2. Run `source setup` and then change directory to `exercise1/groupX`
3. Open the file `solution.cpp` with an editor of your choice (`vi`, `nedit`).
4. Add the missing code to `solution.cpp` to execute the VMEbus cycles listed below:
    1. Write `0x12345678` to address `0x08000000` in A32 / D32 mode. Use the "safe" cycles
    2. Read the data back from address `0x08000000` and compare it
    3. Write `0x87654321` to address `0x08000004` in A32 / D32 mode. Use the "fast" cycles
    4. Read the data back from address `0x08000004` and compare it
    5. 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.
    6. Read the data back from `0x08001000` in A32 / D64 / MBLT mode and compare it
5. Run `make` to compile the application
6. Run `solution` and catch the VMEbus transfers with the VMEtro VBT325 analyser

Good practice

- Check all error codes
- Do not forget to undo all initialization steps (return memory, close libraries) before you exit from an application

Exercise 2

The Trigger

Introduction

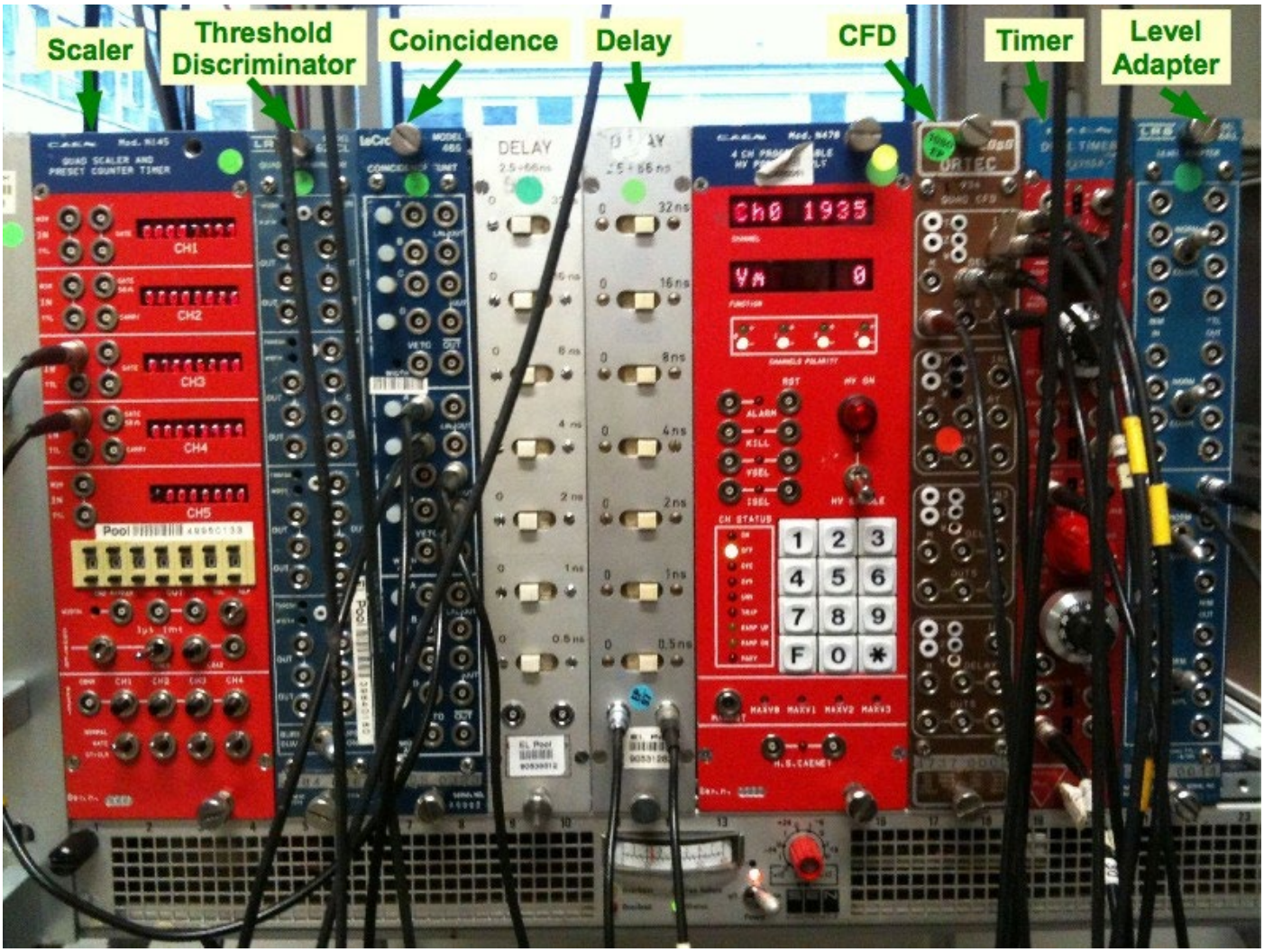


Figure 1. NIM modules.

This is a basic exercise based on the trigger lecture. It introduces all the elements and concepts needed in exercise 3 and 4. The available NIM modules are showed in Fig. 1. The exercise is composed of 4 parts. At each step, look at the corresponding schema and follow the instructions.

A trigger is given by the transition of a signal from the logical 0 to 1. Before setting up any trigger system, you must have decided the levels corresponding to these logical levels and all the components of the system need to be coherently configured.

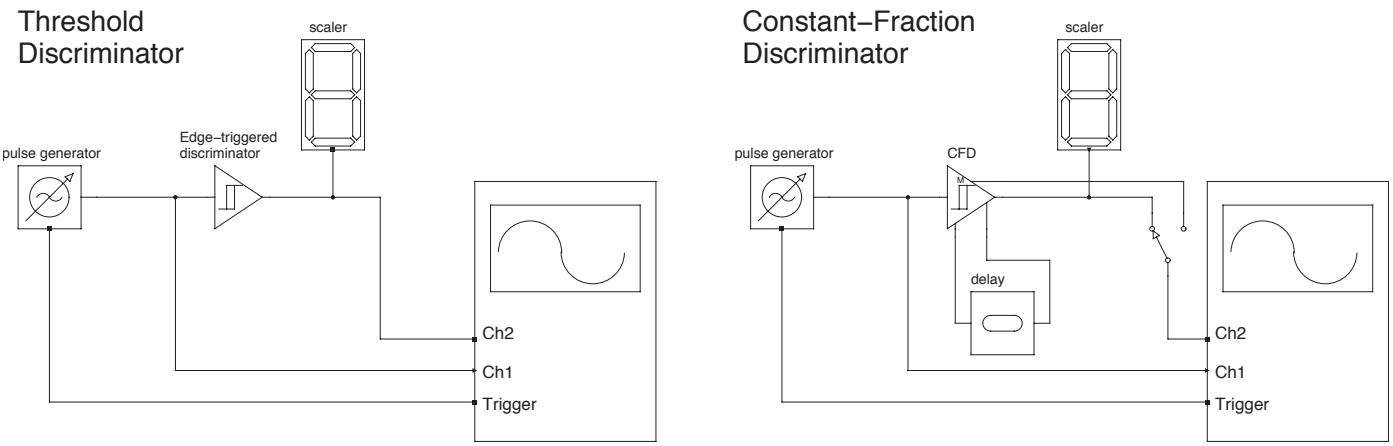


Figure 2. Scheme of threshold and constant fraction discriminators.

Part 1a: Threshold Discriminator

The **Signal Generator** is pre-configured to provide a triangular pulse with a period of 300 µs.

Look at the signal (Channel Output) with the oscilloscope (CH1), using the Trigger Output of the generator as oscilloscope trigger (EXT).

The Trigger Output is TTL signal.

How do you expect it?

Why do we use it?

Now try to characterize the signal:

Leading edge time	
Trailing edge time	
Width	

Using the LEMO cables, try to implement the schema shown in the left part of Fig. 2, i.e.:

- Split the generator output signal: connect the two parts to the input of the **Threshold Discriminator** and to the oscilloscope.
- Connect one output signal of the discriminator to the scaler module and a second output to the oscilloscope (CH2).

We have set-up a simple trigger system: you have a digital answer based on the amplitude of a signal. Reproduce the oscilloscope display shown in Fig. 3 and observe the amplitude of both the signal and its corresponding trigger.

Can you modify their amplitude?

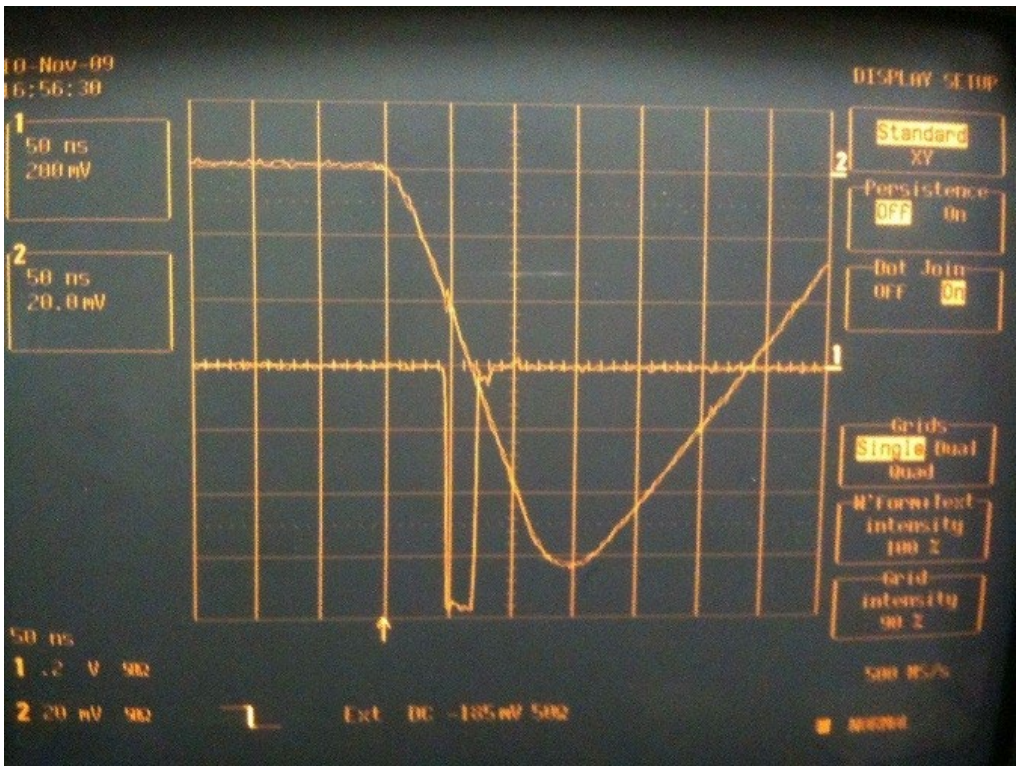


Figure 3. Input signal and threshold discriminator output.

The threshold set on the discriminator can be measured with a **Voltmeter** (x10 output) and changed with a screwdriver. Change the threshold value: observe the behaviour of the discriminated signal on the scope and its rate on the scaler.

Can you relate them to the threshold values?

In real experiments, how is the best threshold value found?

Part 1b: Threshold Discriminator, the jitter

Using the above set-up, set the discriminator threshold to 60 mV 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 (-100, -150, -200, -250 mV) and fill up Table 1 with your numbers.

Input signal amplitude (mV)	Threshold D (ns)	CFD (ns)
100		
150		
200		
250		

Table 1. Measured delays on the discriminated signal with respect to reference.

Part 2: Constant Fraction Discriminator (CFD)

Now use the **Constant Fraction Discriminator** to make a trigger from the generator signal and implement the layout shown in the right diagram of Fig. 2. Using the Voltmeter and the screwdriver, set these CFD parameters:

threshold (T):	60 mV	Measure with Voltmeter (x10 output)
walk (Z):	2 mV	Measure with Voltmeter (x10 output)
delay (D):	80 ns	Set with delay module + 2x10 ns cables

Connect the CFD monitor output (M) to the scope CH2 and reproduce Fig. 4.

Figure 4. Input signal and CFD monitor output.

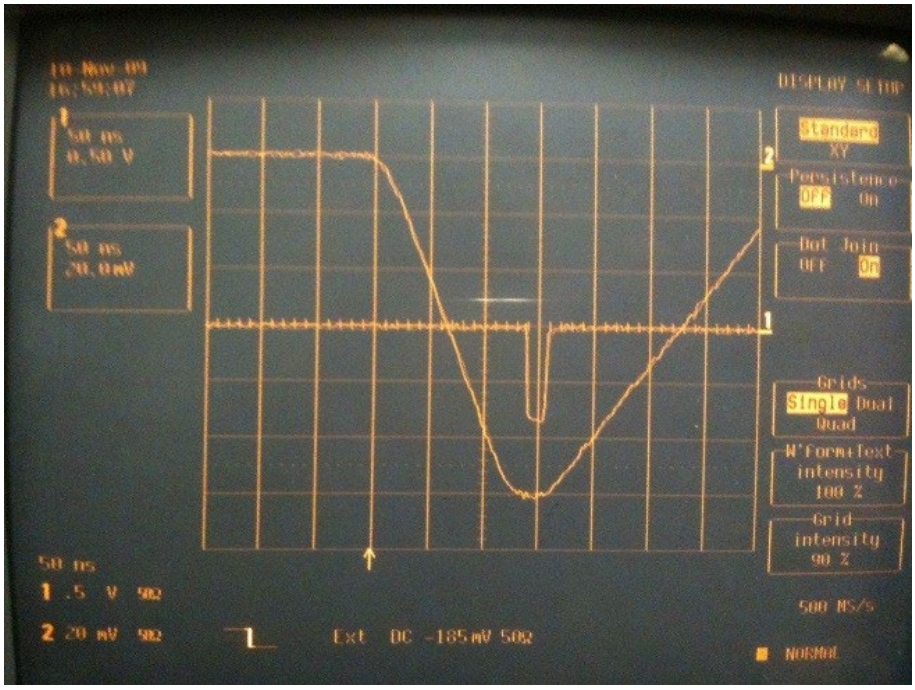


Figure 5. Input signal and CFD output.

***Can you recognize the* CFD technique?**

Which is the effect of varying the value of the delay D?

Connect now the CFD output to the scope (CH2) and change the amplitude of the input signal.

What happens 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 (-100, -150, -200, -250 mV). Fill up Table 1 with your numbers. Compare the results with the previous measurements.

Can you see the advantage?

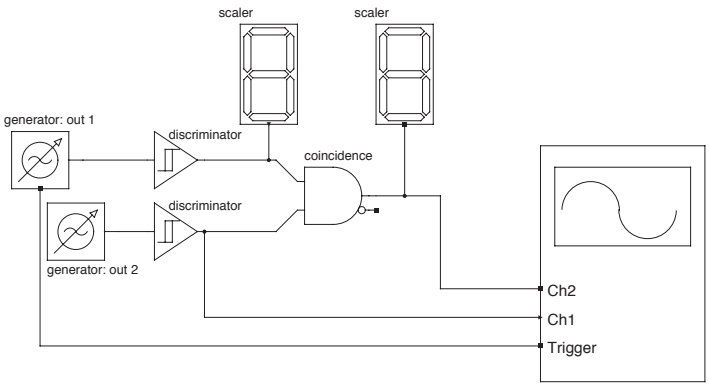
Can you make the CFD behave like a normal threshold discriminator?

Which configuration parameter has to be modified?

Part 3: Making a timing coincidence

We try now to simulate the coincidence of two different trigger signals, in a simplified way.

For that, use an additional output of the signal generator, which is configured to generate a triangular pulse similar to the first one. Use two threshold discriminator units to discriminate both signals, as described in Fig. 6.



Ffigure 6. Coincidence layout

We have now two independent trigger signals with similar characteristics. Look at them in the scope.

Which parameters are important when making a coincidence?

Use one unit of the **Coincidence Module**, which is able to generate the logical AND of its input signals. The module has two outputs: OUT and LIN-OUT.

Can you guess the timing behaviour of the AND output?

When are you expecting the AND output to rise?

The **Scaler Module** is a simple and useful tool in a trigger system: it allows you to simply count the triggers and verify if your system is behaving correctly. Then use the scaler to measure the counting rate of your coincidence and try to answer these questions:

Can you count any trigger? How can you recover the coincidence rate?

After your adjustments, which is the width of the coincidence signal?

Can you explain the different behaviour of the OUT and the LIN-OUT signals?

Which is better to use in a real trigger system?

How can you save the trigger efficiency if one of the signals have large jitter?

Which is the drawback?

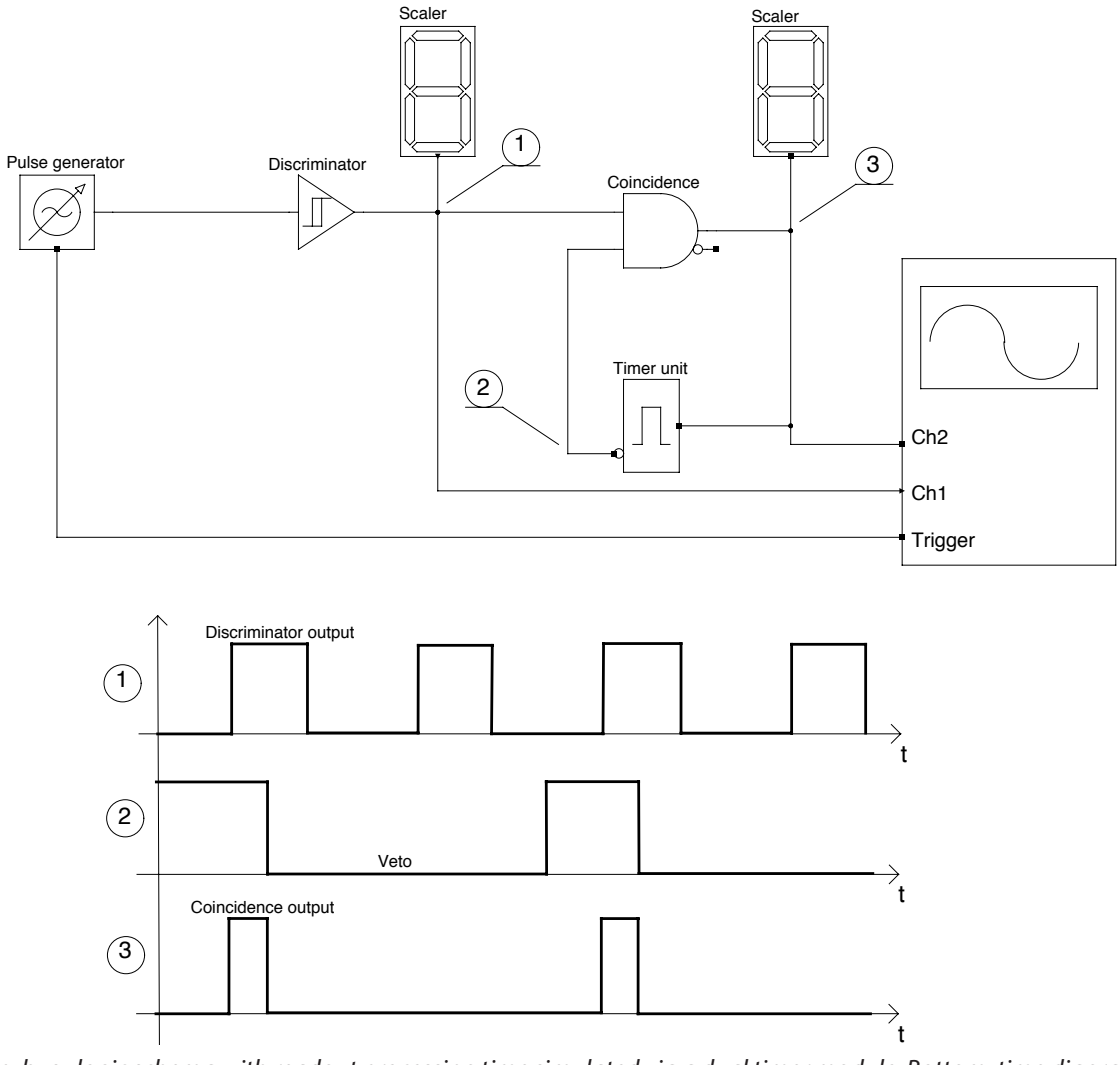


Figure 7. Top: busy logic schema with readout processing time simulated via a dual timer module. Bottom: time diagram of signals at the discriminator output (1), after the veto (2) and at the coincidence output (3).

Part 4: Trigger veto and dead-time

A busy logic can be implemented using the coincidence module and a **Dual-Timer Module** which simulates a readout system with a fixed processing time (readout dead-time).

Configure one stage of a dual timer module to generate signals with 10 ms width. Then implement the busy logic in a second stage of the coincidence unit as shown in Fig. 7:

- one input of the coincidence unit is the trigger signal;
- to simulate the start of the readout, and so the trigger ACCEPT signal sent to the readout system, use the output of the busy coincidence to drive the timer module (START);
- use the output of the timer as the VETO of the busy coincidence: this is the BUSY signal sent back to the trigger system;
- connect the trigger signals before and after the busy logic to the scaler and check the correct logic.

You can easily make a rate measurement configuring the Scaler to work with a time gate of 1s with a GT+CLR configuration.

Compare the trigger ACCEPT rate and the readout rate (after the BUSY) on the scalers.

How do they relate with the timer module setting?

Can you reproduce the numbers using the LIN-OUT of the coincidence unit?

Alternatively you can make an AND between the trigger and the output of the timer (with inverted logic) and not as a veto.

Where is the difference in the logic? Which risk are we taking?

Can you explain the behaviours observed disabling either one or the other input of the coincidence unit?

Appendix: the Constant Fraction Discriminator

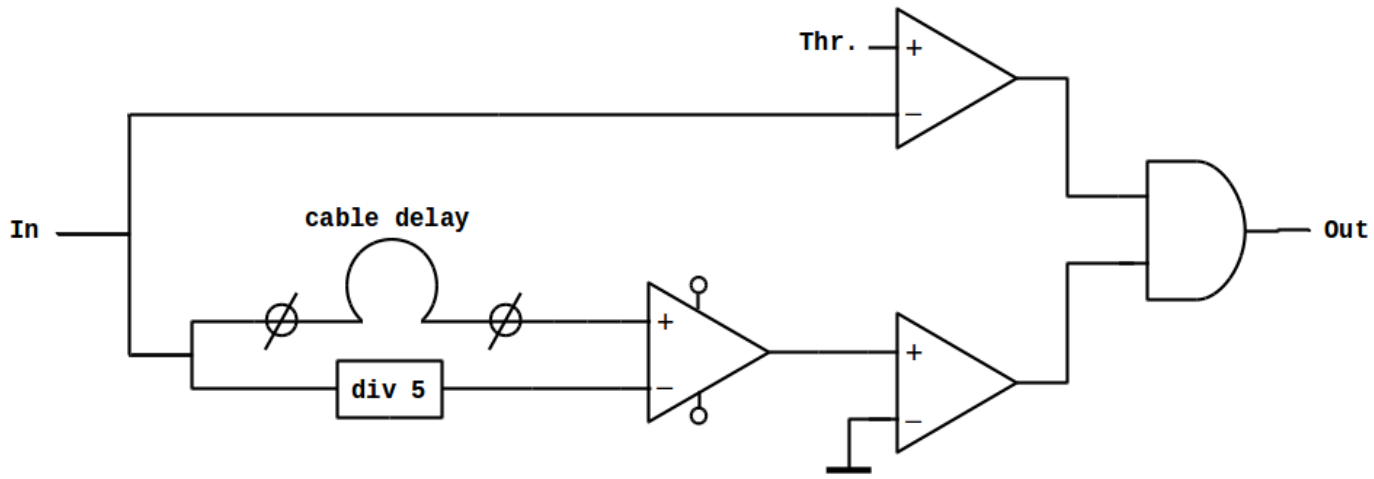


Figure 8. CFD function diagram.

The CFD functional diagram is showed in Fig. 8. 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).

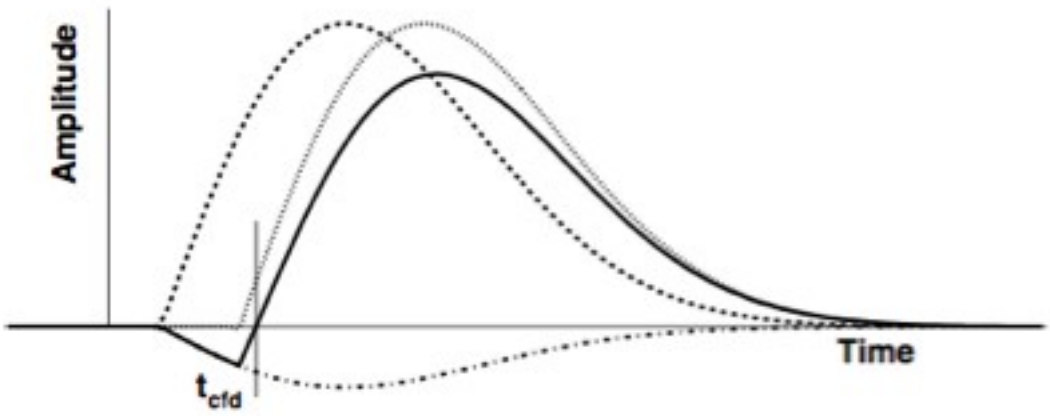


Figure 9. CFD function diagram.

Fig. 9 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 is normally called walk (Z).

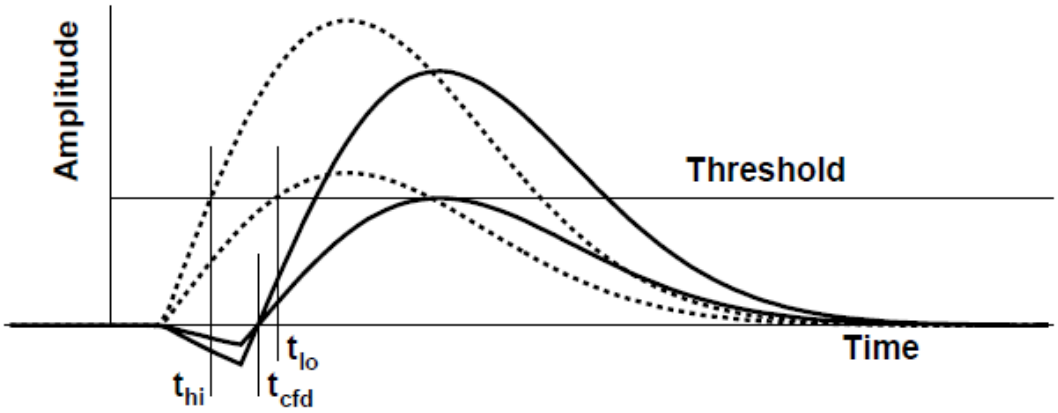


Figure 10. CFD function diagram.

In order to complete the CFD description, the merging of the top and bottom branch signals has to be considered, with the help of Fig. 10.

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:

- only select signal above a given amplitude *Thr*;
- 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}.

Exercise 3

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

Introduction

This exercise consists in building the trigger logic and the input signals to the VMEbus readout modules for a detector (exercise #4) 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 inFigure 1. Schematic diagram of a scintillation counter.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. Scintillation counter setup2.

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. NIM trigger electronics. From left to right: scaler (counter), discriminator, coincidence unit, delay modules and high voltage power supply.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 to build 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. Diagram of the electronics for the detector, trigger and readout of the scintillator counter setup.4.

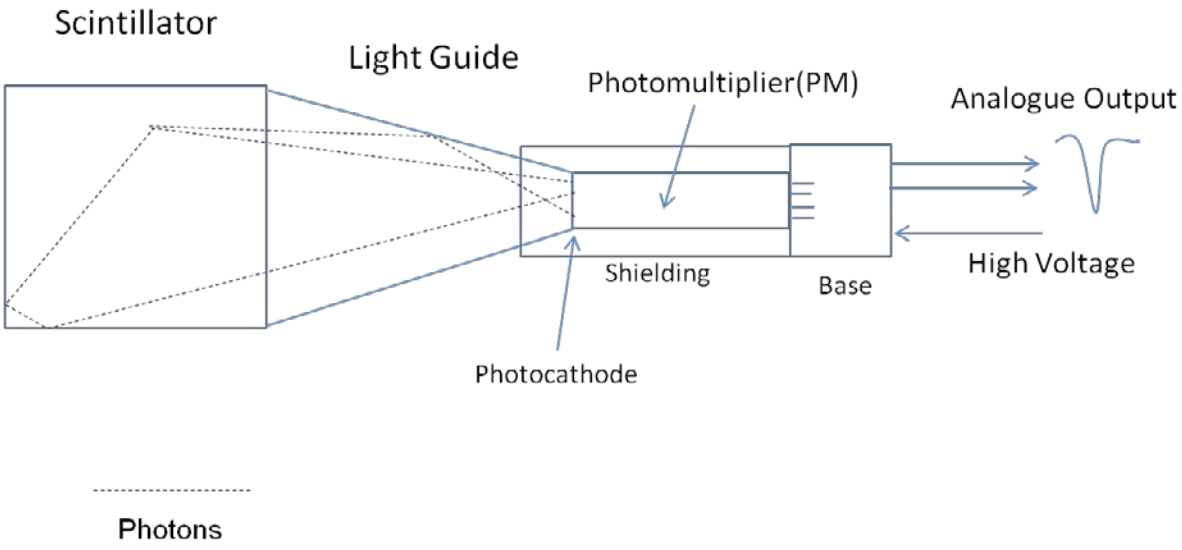


Figure 1. Schematic diagram of a scintillation counter.

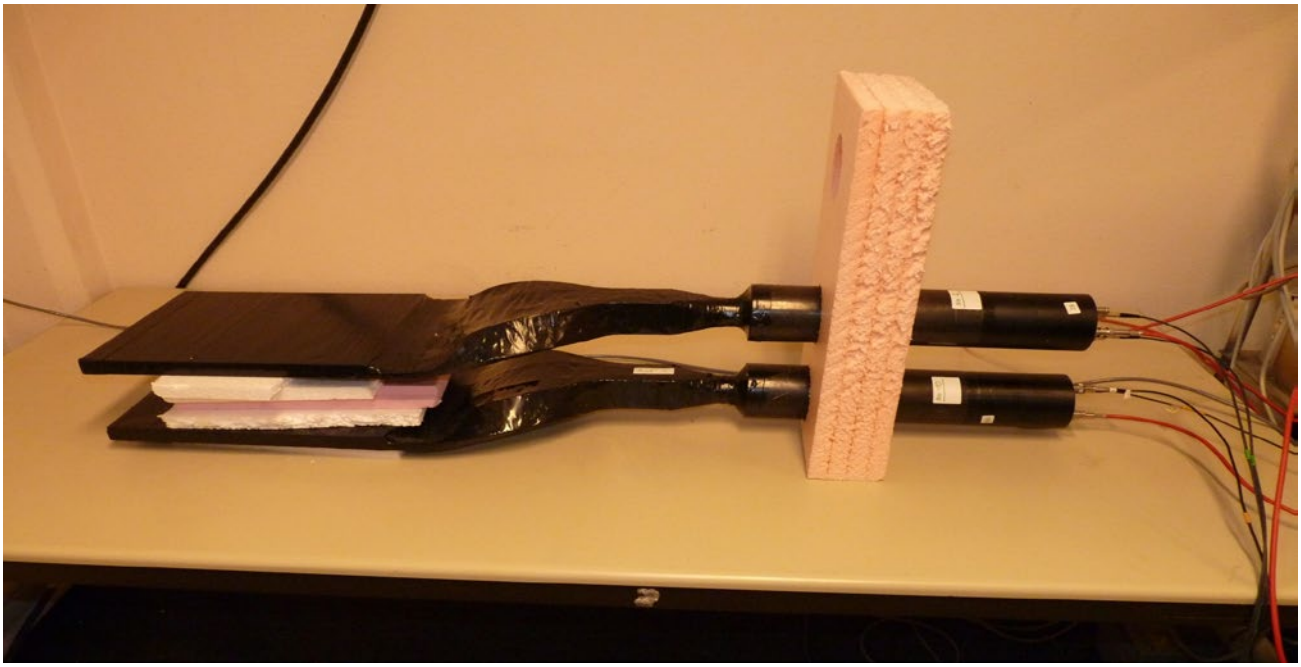


Figure 2. Scintillation counter setup

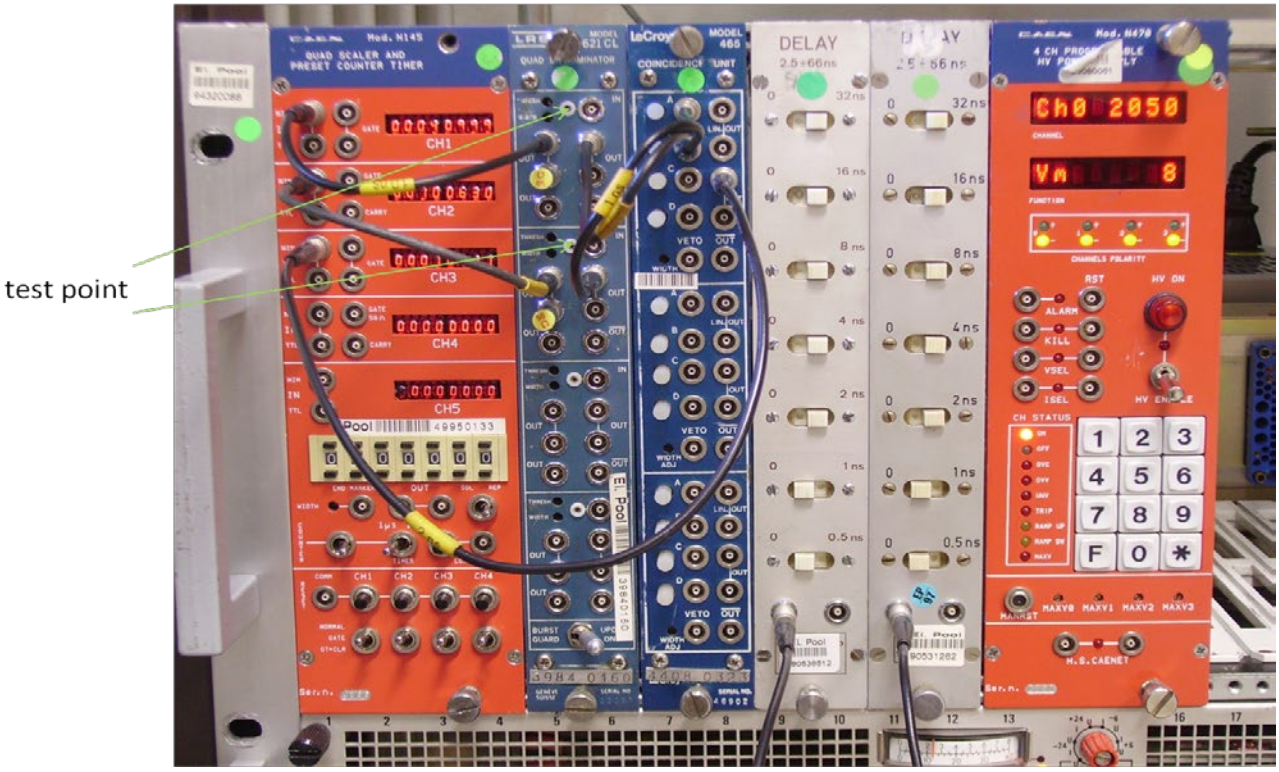


Figure 3. NIM trigger electronics. From left to right: scaler (counter), discriminator, coincidence unit, delay modules and high voltage power supply.

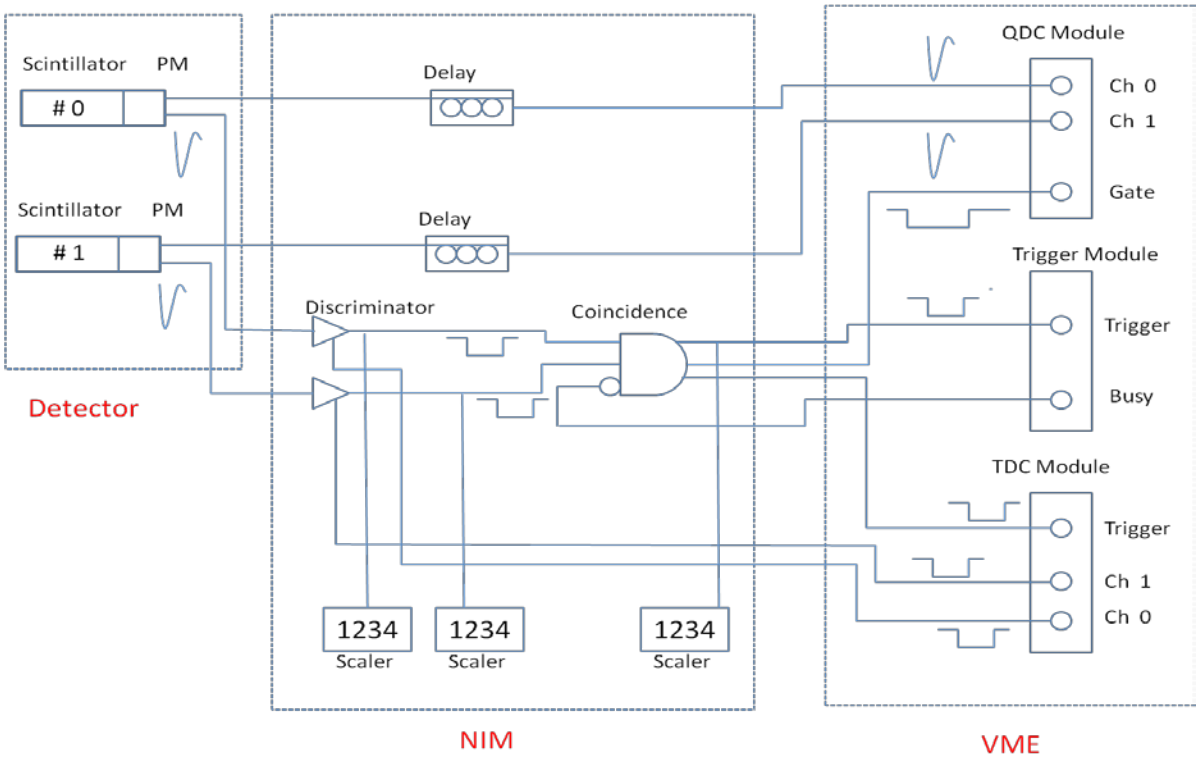


Figure 4. Diagram of the electronics for the detector, trigger and readout of the scintillator counter setup.

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 Appendix 1 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. NIM trigger electronics. From left to right: scaler (counter), discriminator, coincidence unit, delay modules and high voltage power supply.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. NIM trigger electronics. From left to right: scaler (counter), discriminator, coincidence unit, delay modules and high voltage power supply.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. Input to the oscilloscope from a scintillation5. **Note down the charge. You will need it again in exercise 4**
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

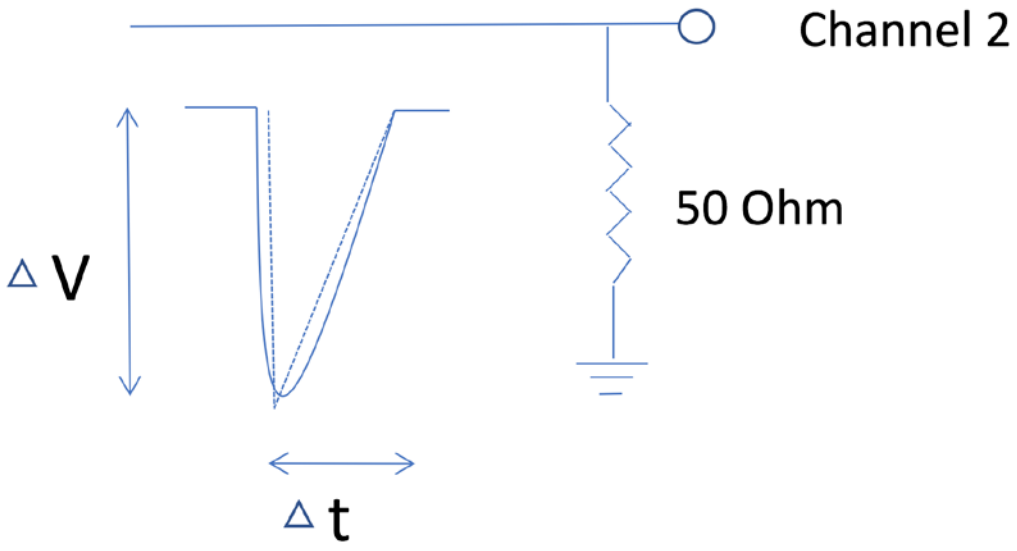


Figure 5. Input to the oscilloscope from a scintillation

Exercise 4

A small physics experiment: detector, trigger and data acquisition.

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. VMEbus data acquisition system: SBC (Single Board Computer), TDC (Time to Digital Converter, Trigger Module (CORBO), QDC (Charge to Digital Converter) and the VMEbus modules shortly described in Appendix 1, Appendix 2 and Appendix 3. The overall run control and monitoring is provided via software running on a (Linux) single board computer (SBC).

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, run control, GUI and infrastructure on a VMEbus SBC. Event rates and dumps are examined. An event monitoring program produces histograms of the QDC and TDC channel data which allow to compute the charges of the input signals to the QDC and the speed of the cosmic muons.

Work plan

- 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).
- Login to the SBC as user `daqschool`, password `g0ldenhorn`
- Start a Terminal window from the toolbar
- Go to TDAQ directory: `cd ~/TDAQ` and run the command source `./setup_RCDTDAQ.sh` to define the environment
- Run the program `v1290scope` which is a low-level test and debug program for the CAEN V1290 TDC

1. Run command: `v1290scope` (Use defaults for the command parameters).
2. VMEbus base address = `0x4000000`
3. Dump the registers (option 2). Is data ready? (bit DREADY in the status register). What are the values of the match window width and the window offset? See Appendix 1.
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 Appendix 1. Exit from the program by choosing menu entry 0.

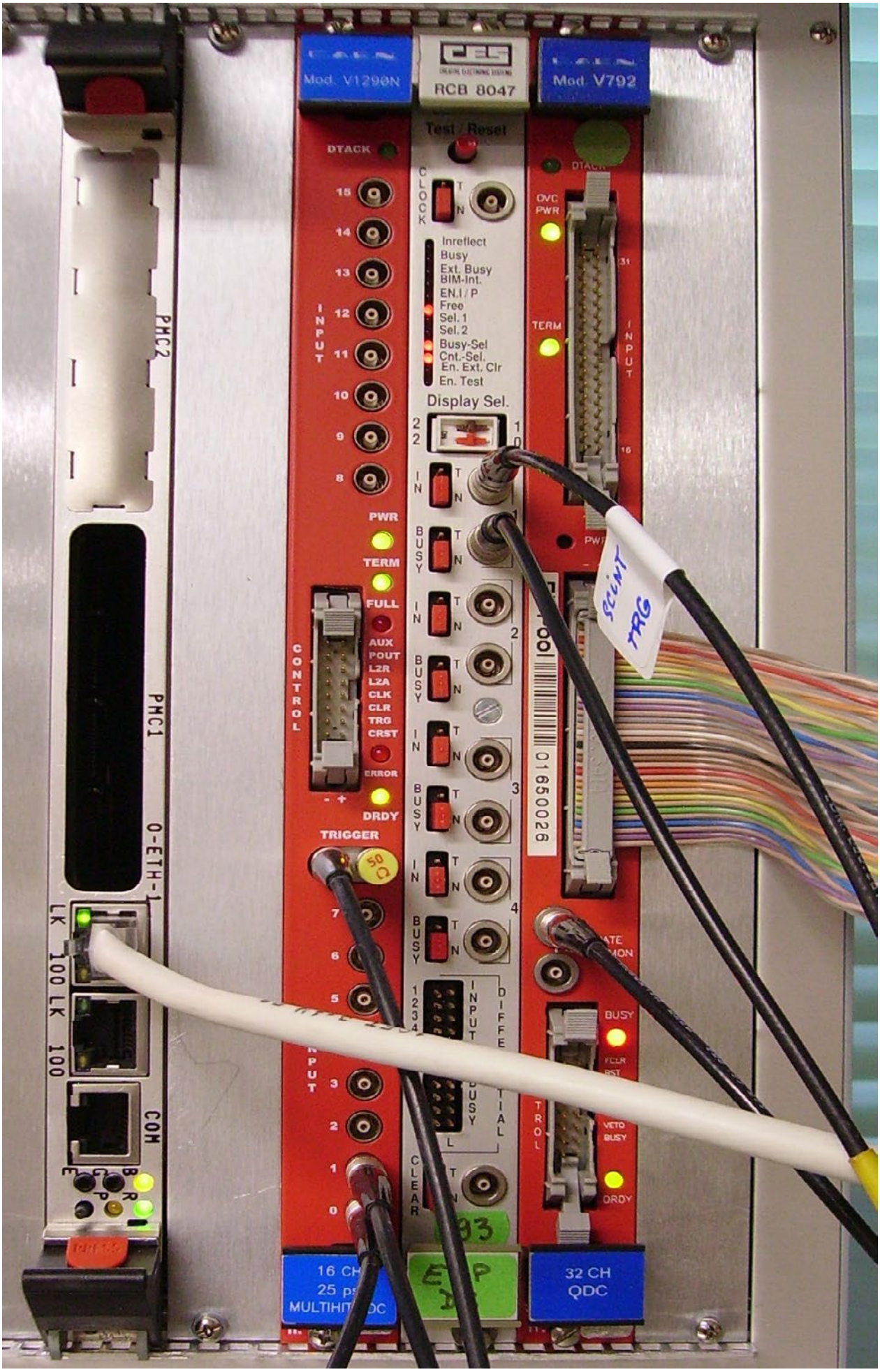


Figure 6. VMEbus data acquisition system: SBC (Single Board Computer), TDC (Time to Digital Converter, Trigger Module (CORBO), QDC (Charge to Digital Converter)

Run the program v792scope which is a low-level test and debug program for the CAEN V792 QDC

1. `v792scope`
2. VMEbus base address = `0x0`
3. dump the registers (option 2). Is data ready? Check also the LED on the module.
4. read an event (option 5). How many words are read? Which channels have data and which are pedestal (empty) values?

We now run the full DAQ system

1. Start the DAQ system: `./setup_RCDTDAQ.sh` start. 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. This is a somewhat long procedure and should result in a message 'OK!'.
2. Now start a GUI display: `./start_Igui.sh`. The "folders" in the infrastructure panel should be green! You may need help from the tutor here ...
3. We now go through the run states in order to start a run. But first please obtain a 'Control' access by selecting the 'Control' radio button in the top menu 'Access Control'. The initialize button should become active. Now, click on INITIALIZE and then wait for the RCDApp (in RCDSegment) to reach the INITIAL state. The readout application is now loaded on the VMEbus processor.
4. Click the CONFIG button followed by OK on the "Remember to ..." dialog box. This configures the VMEbus modules, the CORBO, QDC and TDC.
5. 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.
6. Click START in the control panel (on the left)
7. 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 VMEbus 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).

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 `~/ISOTDAQ /ROSMonitor/src/sc_monitor.cc` (the parts which are specific to the DAQ school are marked with ***)

1. Open another terminal window.
2. `cd ~/TDAQ`
3. `source ./setup_RCDTDAQ.sh` to define the environment.
4. Run the event monitoring task: `./event_dump.sh –e –1`

(–1 means to run forever. If you want only one event, please change it to 1). Once you have seen the raw data output of the even_dump you can terminate this application with `Ctrl+C`.

5. The first nine words of the data constitute an Event (ROD) header. The following words are the data from the QDC and the TDC. Do you recognize the data?
6. On the terminal start the monitoring program by executing monitor. This program monitors data, like the event_dump program, publishing measurements to the histogramming service.
7. In the GUI click on the OH button (Online Histogram). Click on Histogram Repository, part_RCDTDAQ, RCDMonitor. Double click on the histograms to view them.

Alternatively, using a new terminal execute `source ./setup_RCDTDAQ.sh` followed by `./start_ohp.sh`. This is online histogramming presenter. In the panel Histograms (on the left) select SCMonitor to view TDC histograms or RCDMonitor to view also the QDC histograms that we are producing.

8. 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?
9. The charge that you find in the histogram is not the charge delivered from the PMT to the QDC. What is the reason for that and how can we measure the proper charge?
10. The monitoring of the statistics can be reset by stopping and starting the monitoring program (`Ctrl+C` to terminate). This restarts the monitoring program described in point 6.
11. Display the histograms of the QDC channels. Record the pedestal values.

12. Using the formula shown in Appendix 2, compute the mean charges of the signals from the scintillators. Do they agree with the results obtained in exercise #3?

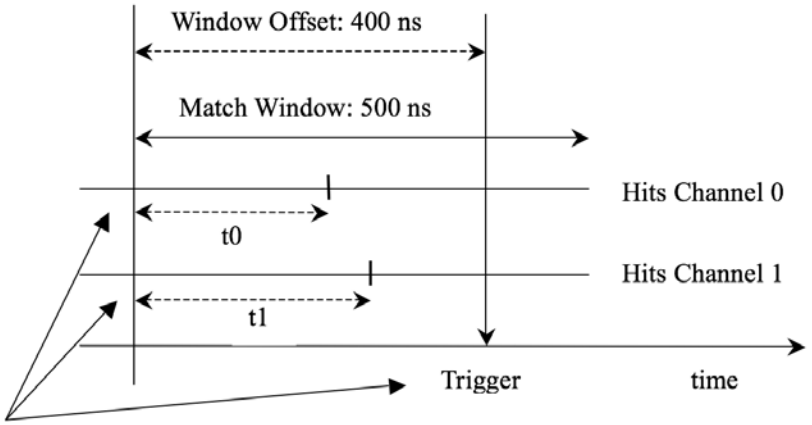
We now want to measure the time of flight of the muons between the two scintillators.

1. In the histogram for the data from the TDC we already get a Δ-t. This value, however, is not the time of flight of the muon. Why? How can we modify the set-up in such a way that we can correct the Δ-t for errors and measure the actual time of flight?
2. Restart the monitor program from the IGUI per point 10 above. Record the new mean value of the Δ-t histogram.
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 of exercise #3 and seen in the picture of the VMEbus crate, Figure 6. VMEbus data acquisition system: SBC (Single Board Computer), TDC (Time to Digital Converter, Trigger Module (CORBO), QDC (Charge to Digital Converter). A trigger match window is then defined by a window offset with respect to 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 25ps stored in the memory of the module.

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`



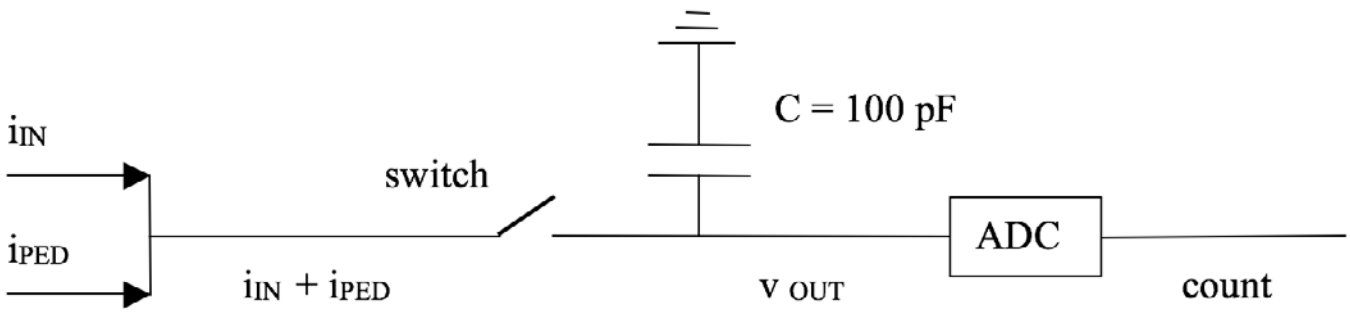
Input signals to the TDC

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

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:

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

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 to block further triggers until the readout code is terminated.

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

Exercise 5

FPGA programming

Introduction

In a lot of digital designs (DAQ, Trigger,…) the FPGAs are used. The aim of this exercise is to show you a way to logic design 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 (fig. 2)

The initial design is loaded into the board.

You will follow the example to understand the design flow. Four exercises are proposed to modify the original design functionality.

Figure 1: development kit

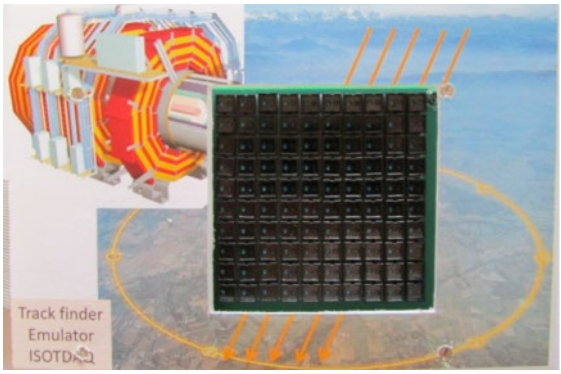


Figure 2: detector

Quick Start

1. Programs used are: QUARTUS (FPGA tool), ModelSim (simulator), LabView
2. Ask the tutor if you have question(s) or problem(s)

Exercise (example)

When you switch on the kit, the initial design is loaded into the FPGA.

On the LabView window, you can see the progression of the marker on the detector.

At the same time, you can see on the two 7-segments LED (the right ones on ALTERA kit) the column and the line number over which the maker is positioned.

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:

1. A green rectangle which is used to transmit the information to the computer via the RS232 connection to display the trace on LabView.
2. A blue rectangle in which the design generates the clock and the logic to control the detector (see Appendix A for detailed functionality).
3. 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 one 7-segment LED blinks (the third for the right side).

Click on 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 QUARTUS Processing->Start Compilation.

The design is compiled for the chosen component (Cyclone II).

The compiler executes multiple tasks:

- logic optimization
- generates a binary file used to program the FPGA (memory array),
- extracts the timing between each logic elements used for the timing analyses
- generate an output VHDL file used for the simulation

Simulation

When the compilation is finished, you can check the design with a simulator. To do this you will use ModelSim.

Check in the "Project" TAB if there is a file marked with a bleu "?", if YES, compile it (right-clic on it, Compile-> compile selected)

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 emulates a trace starting from the top left and finishing at bottom right describing a straight line on the detector.

(The tutor will give you some explanations on the results and the signals shown in the waveform)

Remember where the signal OK goes to "TRUE".

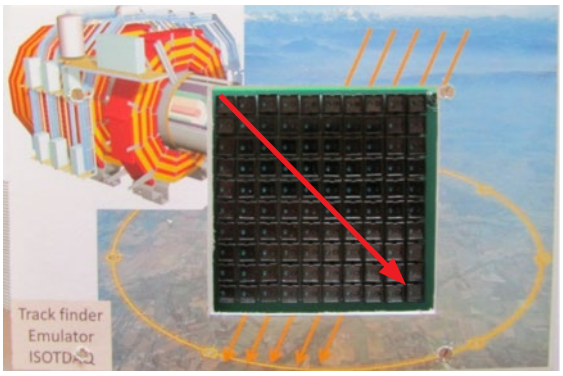


Figure 3: straight line

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 tutor).

One file is shown in the window: it is your design. Click on Start .The programmer takes few 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

Draw a straight line from top left to bottom right to see if the design works well!

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

Good Luck!

Exercise 5.1

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 inst_graph and JKFF inst_result and connect the output 'result' of "track1" box to the JKFF inst_result with a line.

- Compile the design
- Simulate the design

Go to ModelSim:

Compile the file marked with a ? in the "Project" tab (select the file to be compiled – 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 5.2

In this exercise we want to detect a curved trace.

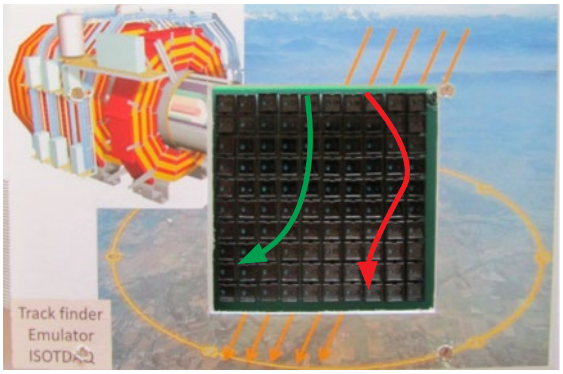


Figure 4

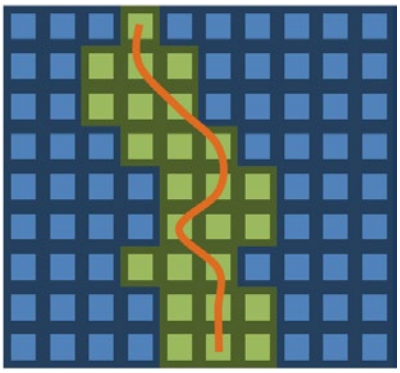


Figure 5: example of trace expected.

In the QUARTUS design entry (file "CII_Starter_Default.bdf"), delete the line between output 'result' of "track1" box to the JKFF inst_result, and connect the output of the "trck_fnd01" box to JKFF inst_result.

The "trck_fnd01" box logic detects only a straight trace. Compile the design and do a simulation:

- Compile the design (QUARTUS)
- 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 start the simulator.

A signal OK becomes true if the logic detects the expected trace (here a straight trace).

In this exercise, you will examine the implementation of the design in the FPGA and see how we can change the results (max. frequency …)

1. In QUARTUS open TimeQuest (Tools -> TimeQuest timing Analyser)

-double click on Report Fmax Summary ("Tasks" window)

You can see the maximum frequency of each clocks implemented in the design

(Note the max frequency that "scan_clk" can reach)

2. Go back to QUARTUS,

Open the partition window (Assignments -> Design partitions window)

Right-click on the partition named "trck_fnd01:instzigzag" (Locate-> Locate in Chip Planner)

Now, you will specify the place where your logic will be implemented:

There is a blue rectangle in the Chip planner (named "trck_fnd01:instzigzag").

Place it where you want (not at the place where the logic is actually implemented) to implement the logic at the next compilation.

Compile the design (Quartus), and execute the TimeQuest (see point 1). Normally the maximum frequency will change.

This give you an idea of the importance of the place of you logic or how to reserve a place if you work in a team (each person will have a reserved place to implement his logic).

NB: For your information, for each clock of the design, the frequency to reach has to be specified in a constraint file.

Exercise 5.3

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 going to a pixel adjacent to the pixel of the first line and so forth (figure 5).

To help you, you have to change code in the "mask_build" entity (beginning of the "trck_fnd01.vhd").

- 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.

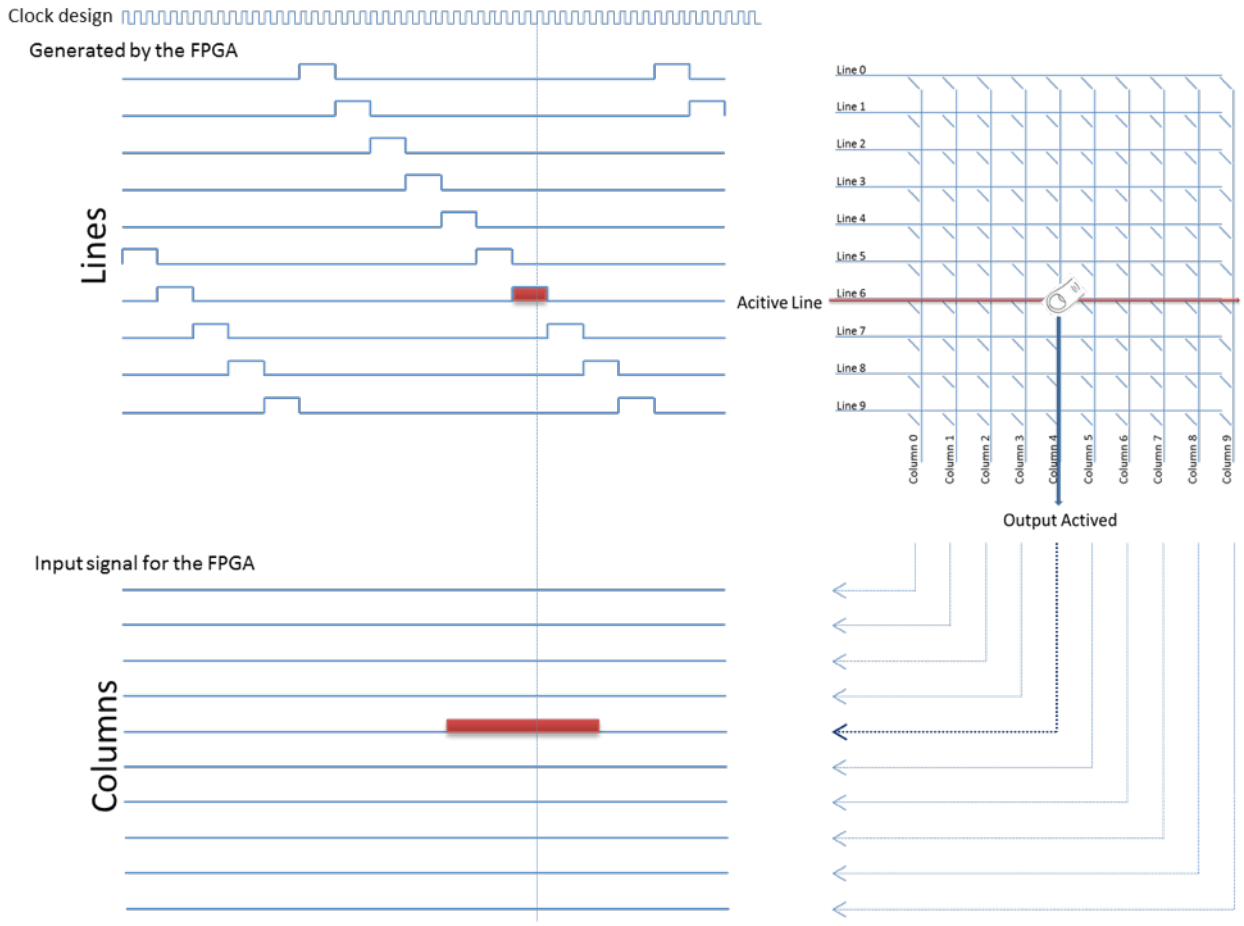
- Download the design
- Test the design

Exercise 5.4

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 5.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. Each line is activated one after the other (0, 1, 2,… 8, 9, 0, 1, …) . 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).

Exercise 6

Micro TCA

Overview

In this exercise you will …

- explore the Micro-TCA technology
- learn about the PCI (express) bus
- write data acquisition software to sample music and display the wave forms

Introduction

In this exercise you will work with modular electronics based on the rather new Micro-TCA standard. TCA stands for Telecommunications Computing Architecture, an architecture that is used in Telco industry to provide high-bandwidth, high-availability solutions. Both Micro-TCA and its bigger brother Advanced TCA (ATCA) will be used in the upgrades of the LHC experiments. Like VME or Compact PCI, Micro-TCA defines a rack—called *shelf* in TCA speak—and boards, called Advanced Mezzanine Cards or AMCs. Typical shelves have space for 12 AMCs but we will work with a slightly smaller version. A *Micro-TCA Carrier Hub* (MCH) performs management functions, such as monitoring temperatures and regulating fan speed to provide the necessary cooling. A Micro-TCA shelf may contain a second MCH for redundancy (but we will work with only one). The backplane contains high-speed serial links that are suitable for transferring data at rates of 10 Gb/s or more using various protocols. Typically the backplanes have a single or dual-star layout with all high speed-links going from each AMC to the MCH slot(s). The MCH then contains a switch for the desired protocol—in our case PCI Express (PCIe). Other backplane layouts exist with high-bandwidth links between neighboring AMC slots.

The test setup

We are using a small ELMA Micro-TCA shelf containing:

- a built-in power module and a built-in fan
- a backplane with star and mesh connections
- an MCH by NAT
- an AMC containing a Processor running Linux
- an I/O AMC (AMC-ADIO24) providing digital and analog IO
- optionally, an AMC that can generate a programmable load on the crate

Figure 1. The exercise setup

We will be working directly on the processor AMC. Keyboard, mouse and screen are directly connected to this card which runs a standard Scientific Linux CERN (SLC) distribution. We will use the network port of this card to communicate with management port of the MCH.

Get to know the setup

Log into the processor AMC:

User: student (ask your tutor for the pwd)

Connect to the MCH by telnet

You can see details of the processes in the MCH by connecting to it with telnet (telnet 137.128.63.19). Among many other options, here, you can enable debugging messages. For example you can show all IMPI messages passing by, by setting `imsg_dbg` to `INFO` level.

Try it. (use command `h` to display help). Keep the window open while doing the next two steps to see actions the MCH is doing.

Explore the system using NATView

- `cd /ISOTDAQ/MCH/natview-version` (you must start natview from here)
- `java -jar natview.jar`
- Connect to the NAT MCH at IP 137.139.63.19

You can use NATView to browse the different Field Replaceable Units (FRUs) in the system. You can get information about the units, their voltages and temperatures as well as the valid operating ranges for all these quantities. You can also modify the non-critical, critical and non-recoverable thresholds for sensor readings.

Try plotting the fan speed of the cooling unit and the current and the temperature of a sensor on the Load AMC over time. Lower the upper thresholds of the temperature sensor to 50, 55, 60 degrees. *Hint*: you can enable auto-update of the plots by right clicking on the item in the tree AND tuning on global auto-update. Then turn on all load groups of the load AMC as shown in the next section. See how the MCH reacts.

IPMI

NATView communicates with the MCH through IPMI (Intelligent Platform Management Interface) commands. The MCH either answers to the IPMI commands itself or it forwards the request to a FRU using a dedicated I2C (Inter-Integrated Circuit, often pronounced I-squared-C) link.

You can also directly use the IPMI protocol to talk to a card in the system. For example, we can program the load of the load board (produced at CERN) through IPMI. For this we use the program `ipmitool` with the following syntax:

```
ipmitool -I lan -H <ip_address> -U admin -P admin -T 0x82 -t <AMC_address>

            -b 7 raw 44 7 0  0 <group_number> <action> 0 15
```

Where:

<ip_address> is your MCH address

<AMC_address> is the target AMC address (Slot1 = 0x72, Slot2 = 0x74, Slot 3= 0x76 ....)

<group_number> is the LED/Load group number from 4 to 11

<action> if 0xff group ON, if 0x00 group OFF

- Try switching on all the load-groups of the AMC
- See the reaction on the temperature of the AMC
- When a non-critical threshold is reached the MCH should increase the fan-speed of the crate (see plot)
- When the critical threshold is reached, the MCH may shut down the AMC or the system (if programmed to do so)
- If the MCH has not shut down the AMC, automatically, you should now turn off the load groups quickly to avoid that the system overheats (the fan is not powerful enough in this crate)

Explore the Backplane

NATView also contains a backplane viewer that can visualize the channels on the backplane. Try it.

AMCs usually have 21 ports, MCHs up to 84 ports. An MCH connects to multiple connectors and is composed of a number of printed circuit boards stacked on top of each other. The boards are called the tongues of an MCH.

Ports are grouped into fabrics. MCHs usually provide switches for a certain fabrics.

In our test setup, the MCH contains

- a Gigabit Ethernet Switch on Fabric A going to ports 0 and 1 of each AMC.
- a PCI-express Gen3 Switch on fabrics D-G supporting up to 4 lanes, going to ports 4-7 of each AMC.

Figure 2 shows the backplane of our test shelf, Figure 3 shows a typical backplane of a larger shelf with 12 AMCs and redundant MCHs.

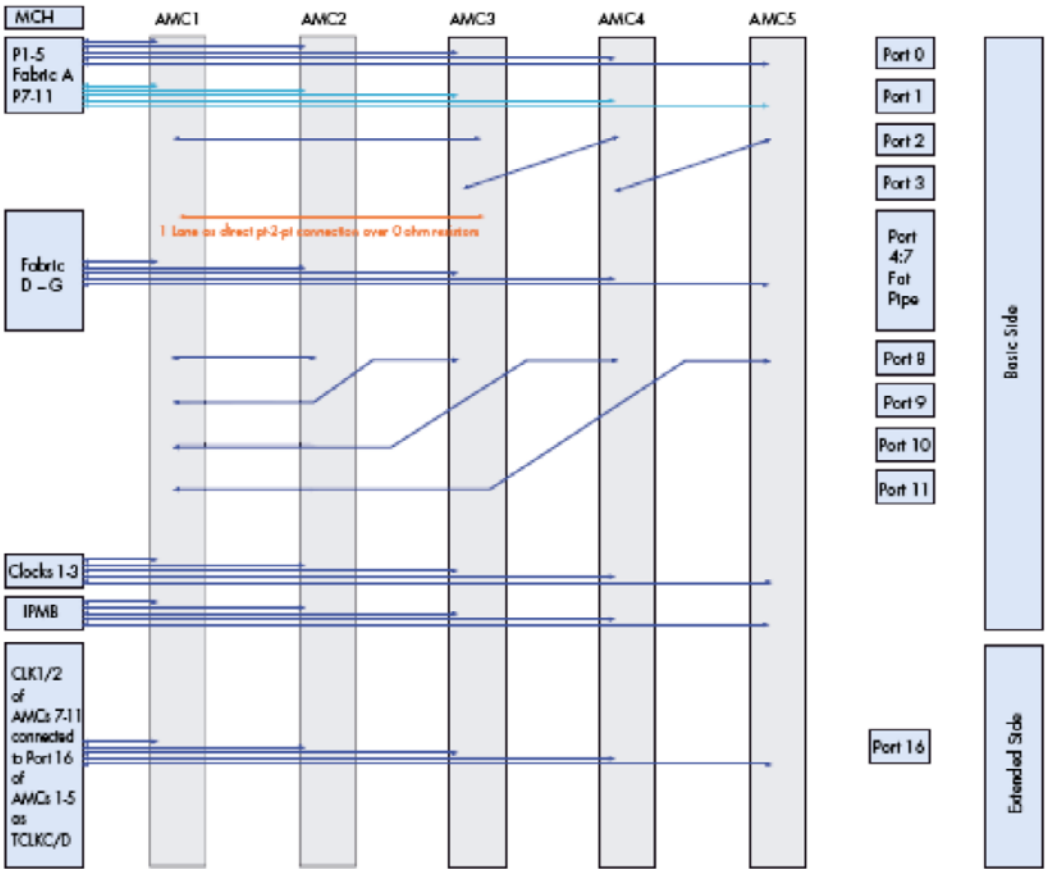


Figure 2. Backplane of the ELMA blue eco shelf used in the exercise.

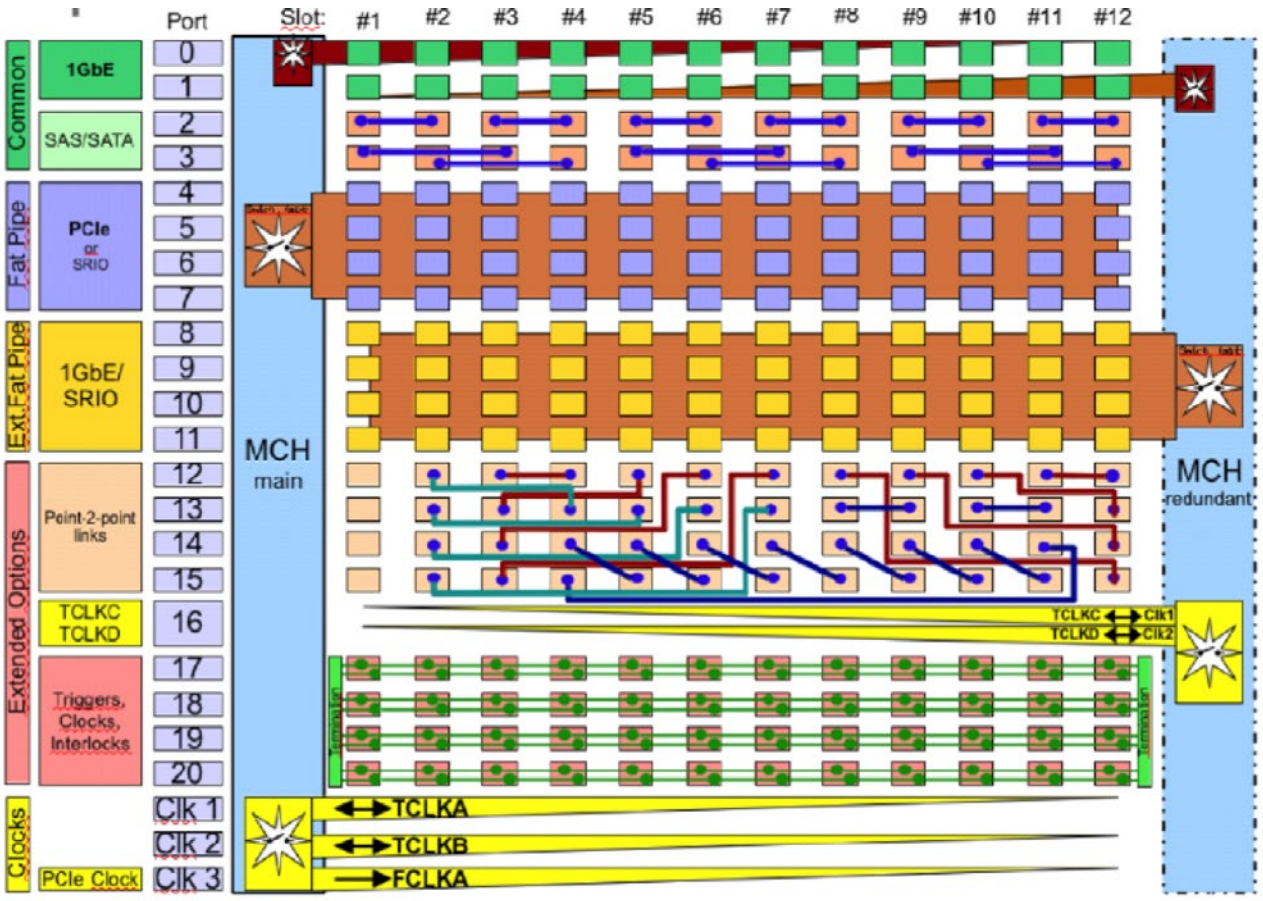


Figure 3. Backplane of a typical larger Micro-TCA crate with 12 AMCs. (not used in the exercise)

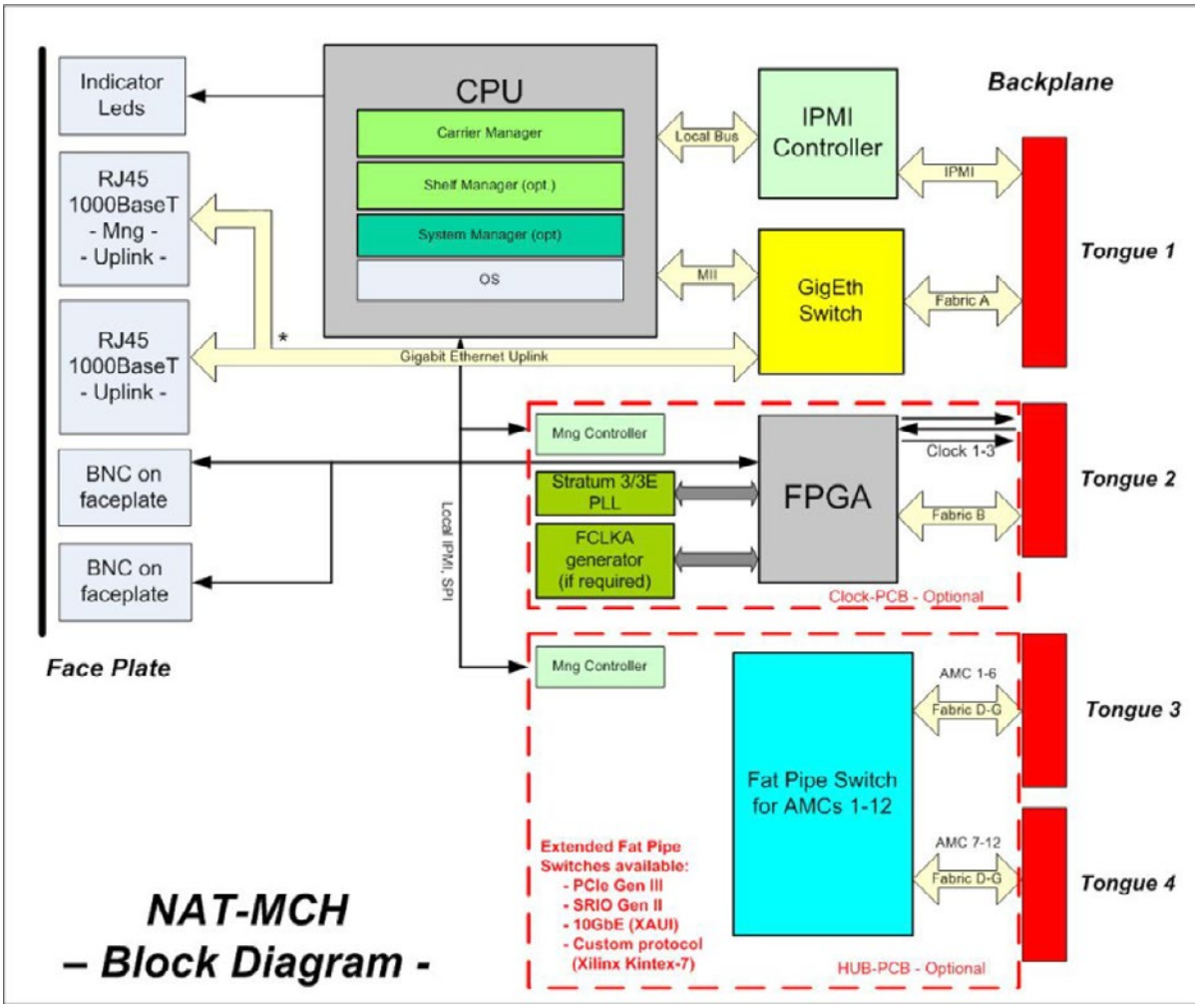


Figure 4. Block Diagram of the NAT MCH.

Have a look at the backplane using the backplane viewer. (You should see something similar to Figure 5). What ports and pipes are in use?

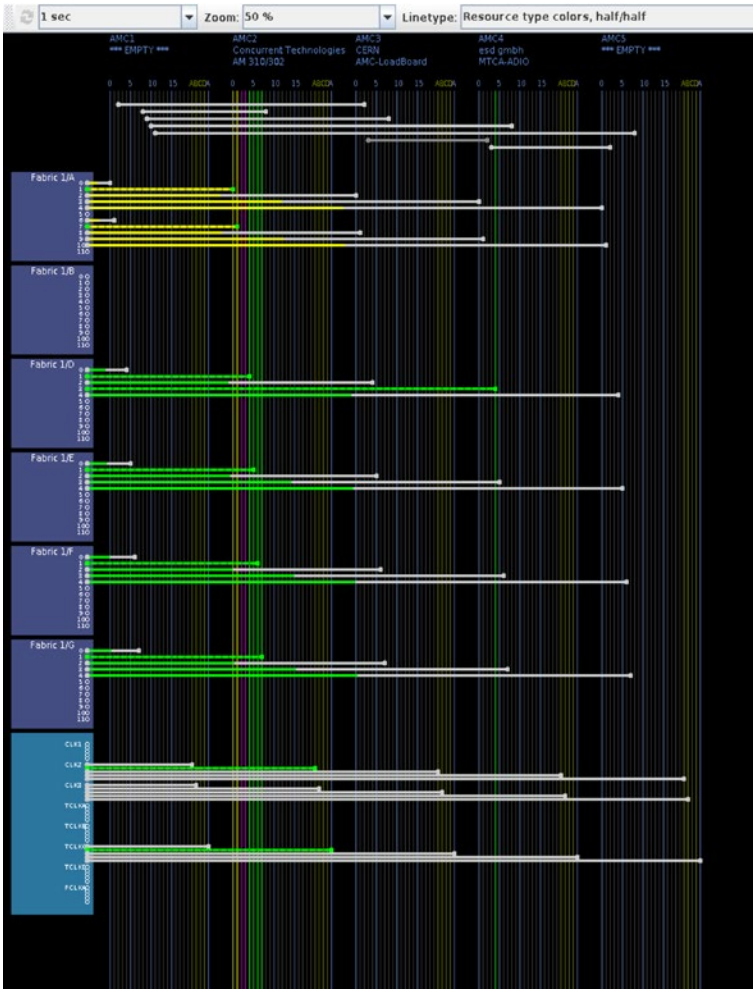


Figure 5. Backplane viewer

PCI Express

The MCH in our test system provides a PCIe switch. Cards in the shelf may use it to communicate with each other. In our system, the Processor AMC communicates with the IO AMC via PCIe. Unlike its predecessor, PCI, PCI express is a serial link. Up to 32 serial links may be combined to form a link. Table 1 shows the speeds in Giga-transfers (GT) per second per lane. Physically, PCIe links are point-to-point, as opposed to a bus topology used in PCI. Data are transferred in packets (like in Ethernet) with data integrity checks, re-transmissions and flow control. PCIe switches are used to connect multiple devices to a single controller (called root complex in PCIe). Despite all these differences in the lower link layers, PCI-express is software compatible to PCI.

	per lane	per lane
Gen-1	2.5 GT/s	250 MB/s
Gen-2	5 GT/s	500 MB/s
Gen-3	8 GT/s	85 MB/s
Gen-4	16 GT/s	970 MB/s

Table 1. Speed of PCI-express

To discover the bus structure and devices, you can use the `lspci` tool.

For example, try: `lspci -tv` to display the bus in a tree structure or try options –`v` and `-vv` to display detailed information about the devices.

Try to locate the IO AMC. Find its bus address and its base address.

Hot Plugging (demo by your tutor)

In a Micro-TCA system, AMCs may be hot-plugged (i.e. exchanged without switching the shelf off). We will try hot-plugging the IO AMC card. Since the IO AMC is a PCI device connected to the Processor AMC, we have let the Processor AMC know.

Together with the tutor, try these steps (you need to be root on the machine):

- show the PCI devices with `lspci`
- pull gently on the black lever
- wait till the blue light is on
- pull the card out by its lever
- repeat `lspci` (did anything change?)
- push the card back in
- wait till the blue light is on
- push the black lever in

```
echo 1 > /sys/bus/pci/devices/[address]/remove
```

- (`address` is the address of this device: *PCI bridge: PLX Technology, Inc. PEX 8111 PCI Express-to-PCI Bridge*)
- repeat lspci (did anything change?)

```
echo 1 > /sys/bus/pci/rescan
```

- repeat lspci (did anything change?)

Build your own digital scope

Now let's get our hands dirty and do some programming. You will use the Analog-to-Digital converter on the IO/AMC to repeatedly sample an analog input channel and to display the waveform of the sampled signal. The A/D converter continuously samples its input at a programmable frequency. The acquired data may either be polled or transferred to host memory by Direst Memory Access (DMA).

- First have a look at the provided example program `adio_scope.cpp` (in `/home/student/amc_adio/src`). See how the program maps the address space of the IO AMC into the Processor AMC's memory space and how it then addresses the IO AMC's registers by simple read and write operations to a data structure. Run the program (from `/home/student/amc_adio/bin`) and play with it. You can recompile it by running make in `/home/student/amc_adio`.
- The scope should sample Analog Input 0 at 44.1 kHz. Have a look at the documentation (Hardware Manual) of the IO AMC and find out how to set up the ADC to sample at this frequency.

1. First set up the timestamp counter to provide timestamps in microseconds (register DIVMODE)
2. Then find out how fast you can poll a register of the IO AMC. Is polling fast enough to do a scope working at 44.1 kHz?
3. Now set up the IO AMC to sample Analog input 1 at 44.1 kHz You will need to set up registers FGENAB and ADCMODE.

- After setting up the card, your program should acquire a few hundred samples from the ADC.
- Your program should produce a file with lines of two columns, containing a timestamp in microseconds and the voltage in volts (separated by a space). You can start from the routine `acquire_shot()` which already provides parts of the solution.
- A ROOT program to plot you file is available. So you don't need to worry about producing the graphics. The program will let you choose the file name to plot.

```
root –l

root [0] .x /home/student/gui_cint.cpp
```

- As an input signal, you can connect the provided head-phone jack to your smart phone and play some music (you'll have to turn up the volume). Or you download a function generator onto your phone – for example RADONSOFT Signal Generator. (No smart phone in your group? Ask you tutor.)

References (.pdf files available in /ISOTDAQ/doc)

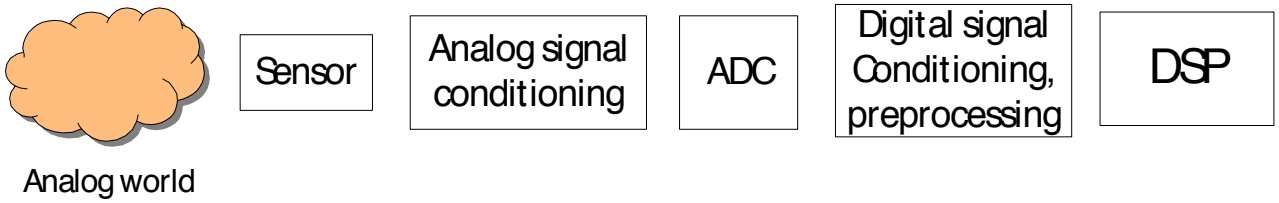
- Short intro to uTCA: `MicroTCA_ShortOverview.pdf`
- http://en.wikipedia.org/wiki/PCI_Express `PCIExpressWikipedia.pdf`
- More info about PCIe: http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1 `PCIExpress_Xilly-bus[1-3].pdf`
- Info about the shelf: `ELMA_BlueEco_Shelf.pdf`
- MCH docs: `NAT-MCH_datasheet.pdf, NATMCH_UsersManual_V123.pdf`
- Manual for the IO AMC: `HardwareManual_IO_AMC.pdf`

Exercise 7

LabView Programming

Introduction

The following block diagram shows the main components of an electronic measurement system:



In most cases the function of the components are the same. The sensor converts the physical value (temperature, pressure, acceleration…) to an electrical signal. After the conditioning of the signal and converting it to the digital space the system process and store the digital values. The numbers and the physical types of the measured signals can be different and processing algorithm as well. If we could handle all types of signals we would be able to solve any measuring problem. Additionally if it was also true for the actuators, we would be able to handle all controlling problems. This is the goal of the modular electronic instrumentation.

National Instruments has been developing measurement equipment since the '80s. They provide general, programmable interface modules: for analog/digital inputs/outputs (programmable ADC/DAC modules with programmable signal conditioning), for standard communication interfaces (Ethernet, CAN/LIN, PROFIBUS, PCI…). They provide a development environment to communicate with these modules with a PC. The software environment uses a graphical programming language.

During this laboratory we are using the following equipment. The modules place in the NI Compact DAQ chassis (cDAQ-9178). This chassis can communicate with the PC (LAN or USB) and control the placed-in modules. The analog input module (NI 9211) contains 24 bit delta-sigma ADCs and 4 differential channels. This module can handle the thermo-couple sensors. We are using an analog input and a digital output module as well.

The chassis contains the following, programmable component: four 32 bit general purpose counters, clock generators, FIFO for analog/digital inputs (with 127 depth per slot), FIFO for digital outputs (2047 samples), clock generators (base clock: 20 MHz, 10 MHz, 100 kHz, divisors: 1 to 16), digital trigger circuit.

By the graphical programming environment we can program and control the modules. We can visualize our acquired data and do calculations. The data processing with this cDAQ chassis runs on the processor of our desktop PC.

Try to answer the following questions:

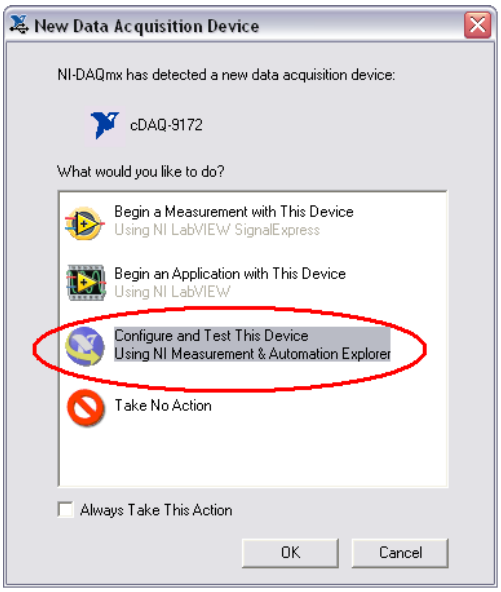
- If this chassis contains only some counters and FIFOs why does it cost approx 1000 €?
- What is the advantage of this modularity, developing a measurement system?
- What would be the problem is we would like to use this system as a control system or as a real-time signal processing application? How would you solve this problem?
- For which application would you choose this configuration? Why?

Exercise 7.1: Take a Basic Measurement with CompactDAQ

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-9178) 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.



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

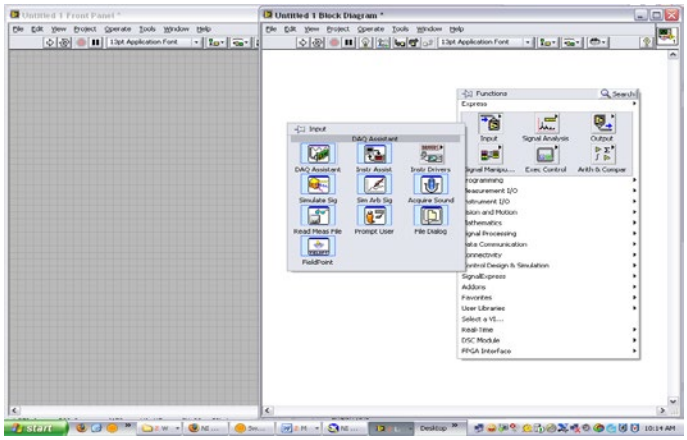
Note: NI Measurement & Automation Explorer is a configuration utility for all National Instruments hardware.

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-9178 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-9178 and click on Self-Test.

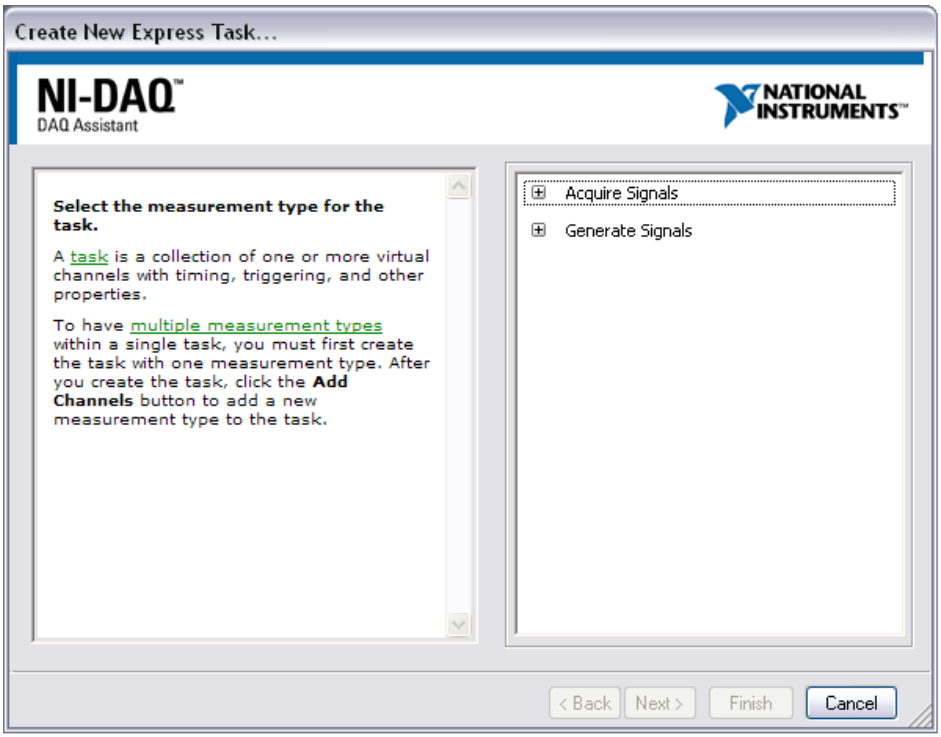
8. The device passes the self-test, which means it has initialized properly and is ready to be used in your LabVIEW application.

Program LabVIEW Application

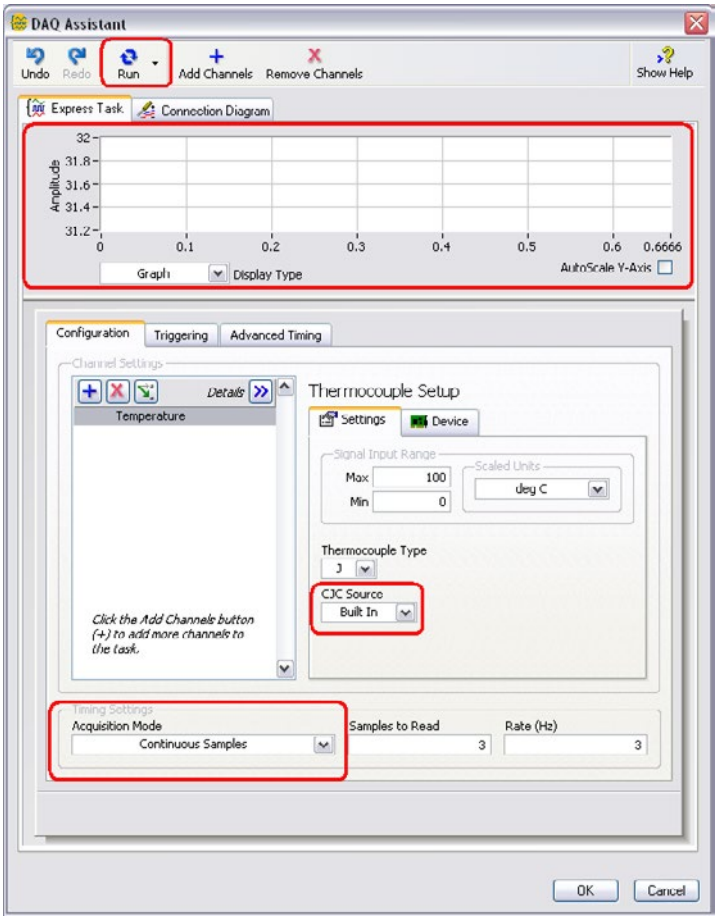
9. Create a new VI from the Project Explorer. Right click on the Exercises folder and select **New» VI**. Once opened, Save the VI in the Exercise folder under the name "1-Basic Measurement.vi."
10. Press <Ctrl +T> to tile front panel and block diagram windows.
11. Pull up the Functions Palette by right-clicking 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.



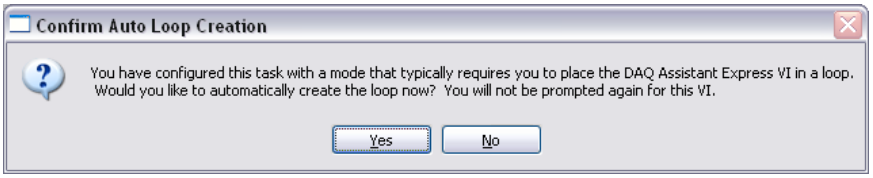
13. The Create New Express Task… window then appears:



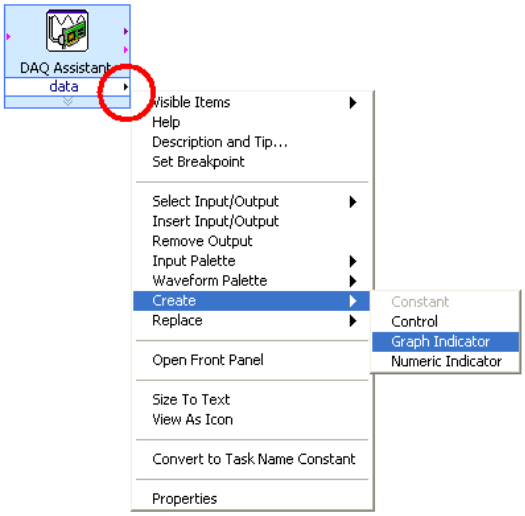
14. To configure a temperature measurement application with a thermocouple, click on **Acquire Signals» Analog Input» Temperature» Thermocouple**. Click the + sign next to the cDAQ1Mod1 (NI 9211), highlight channel ai0, and click Finish. This adds a physical channel to your measurement task.
15. Change 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.



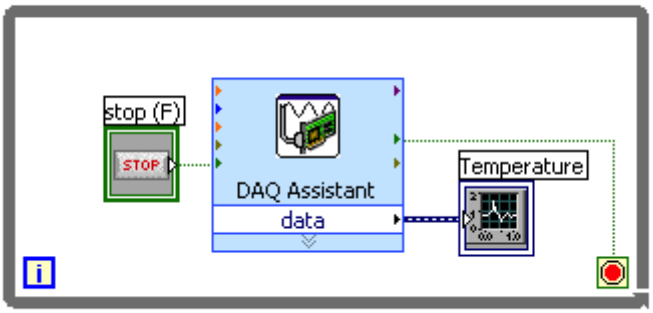
16. Click Stop and then click OK to close the Express block configuration window to return to the LabVIEW block diagram.
17. LabVIEW automatically creates the code for this measurement task. Click Yes to automatically create a While Loop.



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.



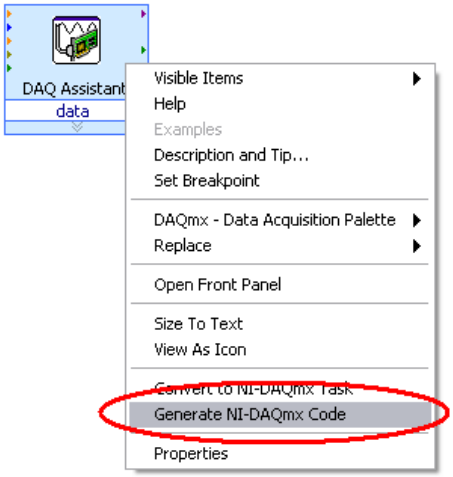
19. Notice that a graph indicator is placed on the front panel.
20. Your block diagram should now look like the figure below. The while loop automatically adds a stop button to your front panel that allows you to stop the execution of the loop.



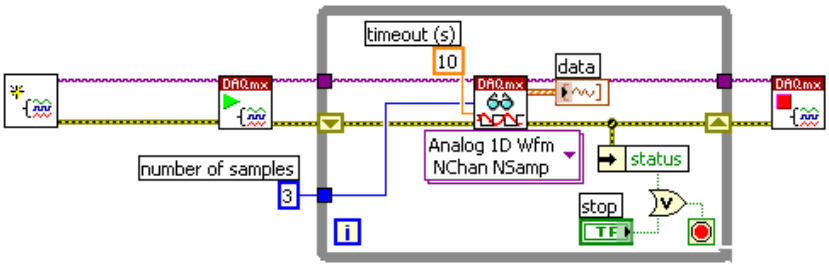
Additional Steps

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.

21. 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.
22. 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."

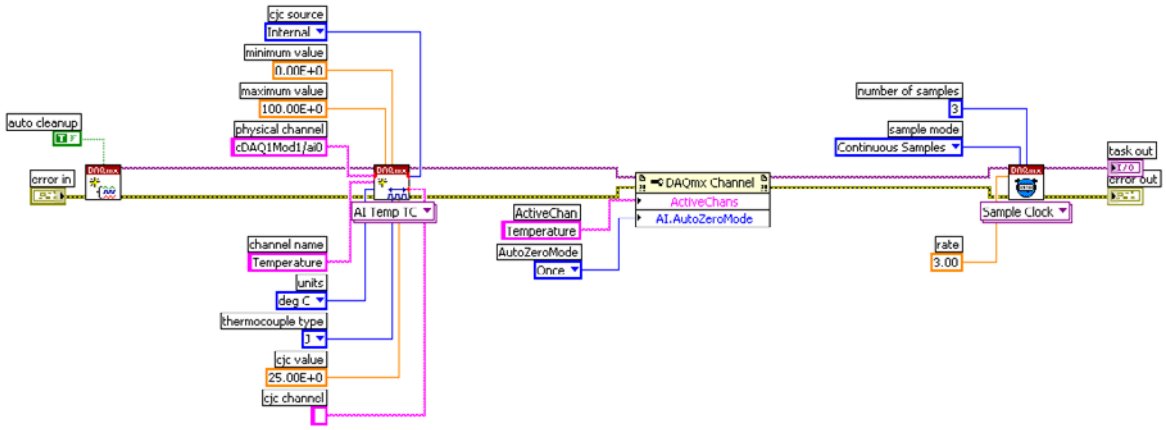


Your block diagram should now appear something like this:



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

23. 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.
24. DAQmx Read.vi reads data based on the parameters it receives from the currently untitled VI on the far left.
25. Double-click on the untitled VI and open that VI's block diagram (code shown below).



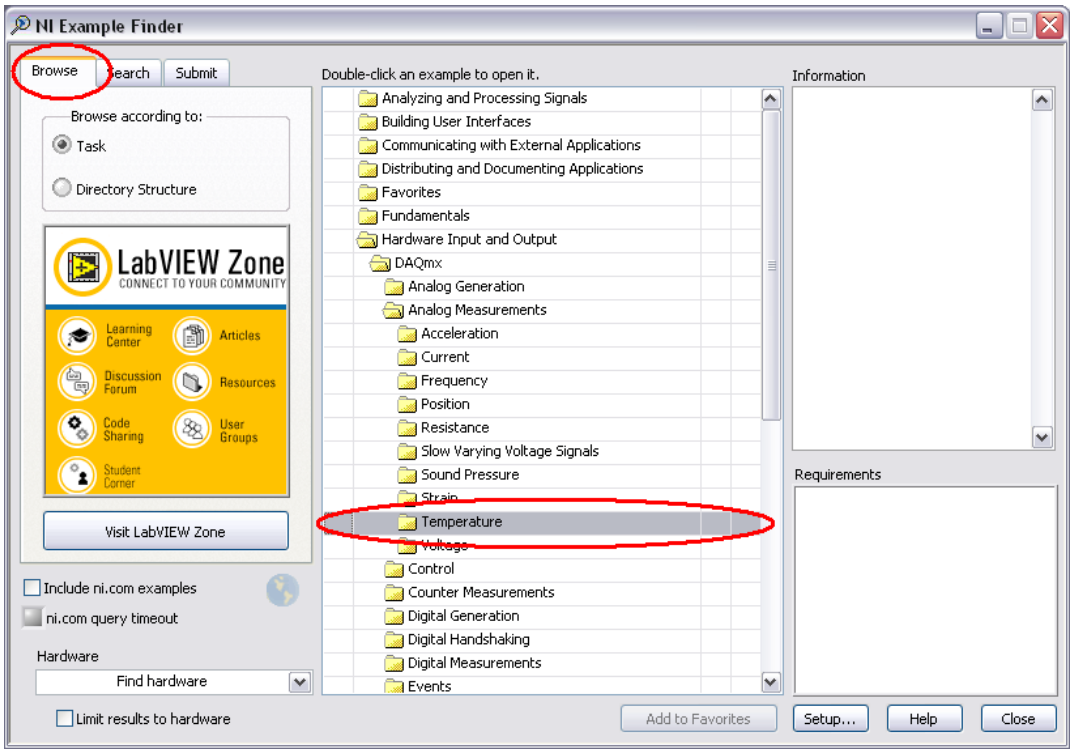
All the parameters that are wired as inputs to the different DAQmx setup VIs reflect the setting you originally configured in the DAQ Assistant Express VI.

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 your application.

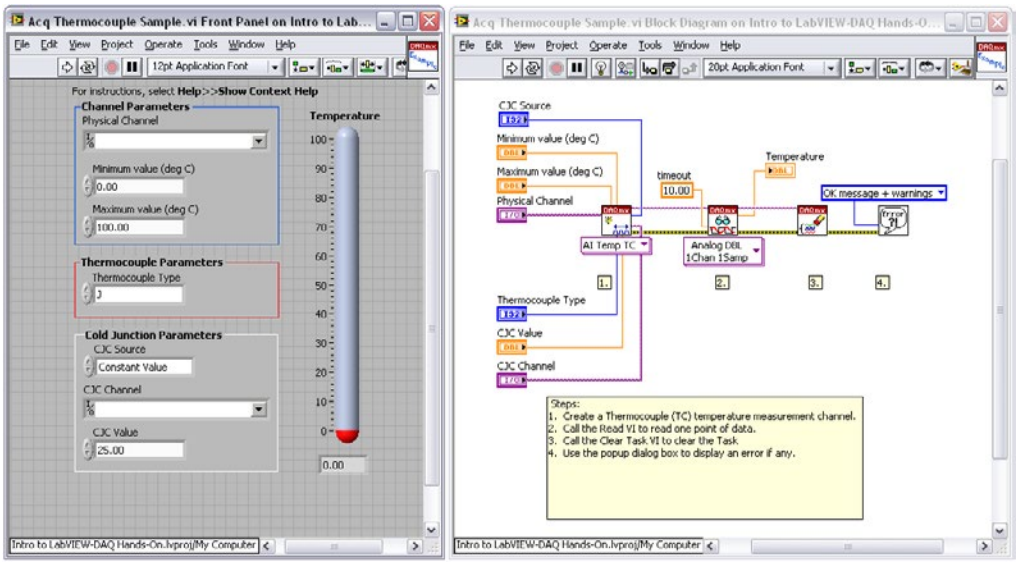
Using the LabVIEW Example Finder

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

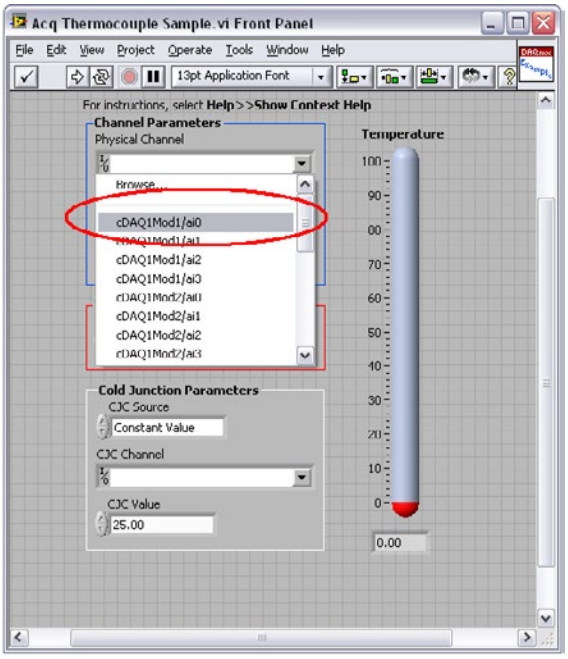
26. Open the LabVIEW Example Finder to find DAQ examples that use DAQmx standard VIs. Go to **Help**» **Find Examples…** to launch the LabVIEW Example Finder.
27. Browse to the DAQmx Analog Measurements folder from the Browse tab at **Hardware Input and Output» DAQmx» Analog Measurements**.



28. The following VI will then appear:



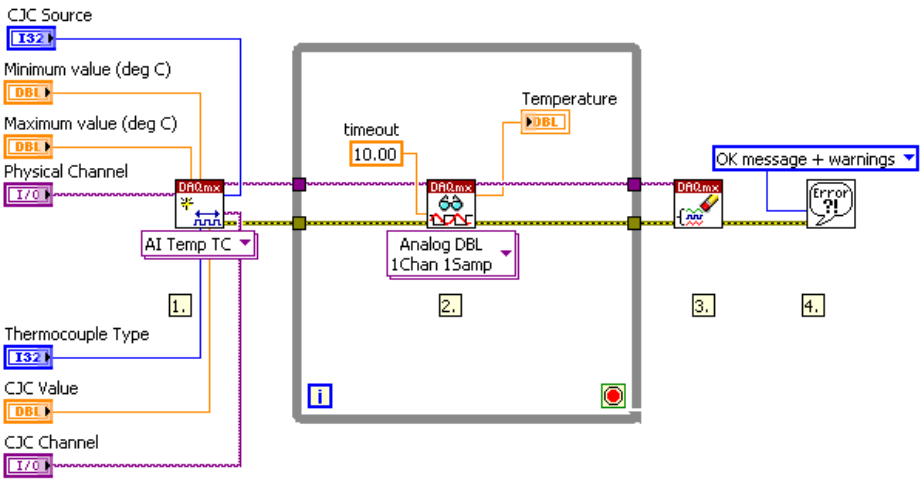
29. Set the Physical Channel to match the CompactDAQ chassis channel and run the application. Expand the physical channel control from the Front Panel and select cDAQ1Mod1/ai0.



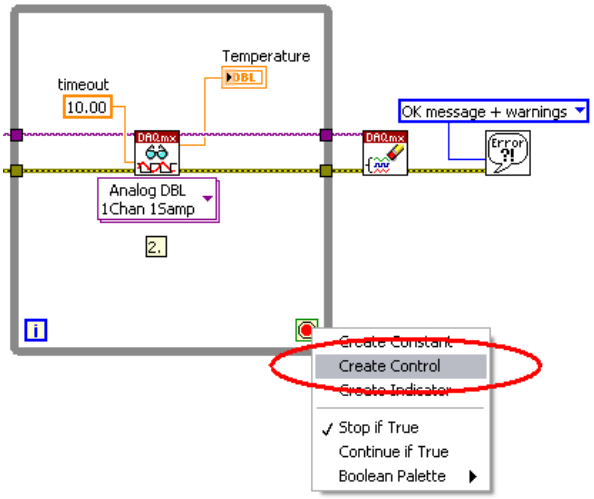
Press the Run button several times while holding and releasing the thermocouple on the CompactDAQ chassis and observe the value change on the front panel.

30. 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.

31. 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.

32. Run the VI. Acq Thermocouple Sample.vi now runs continuously.
33. 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.

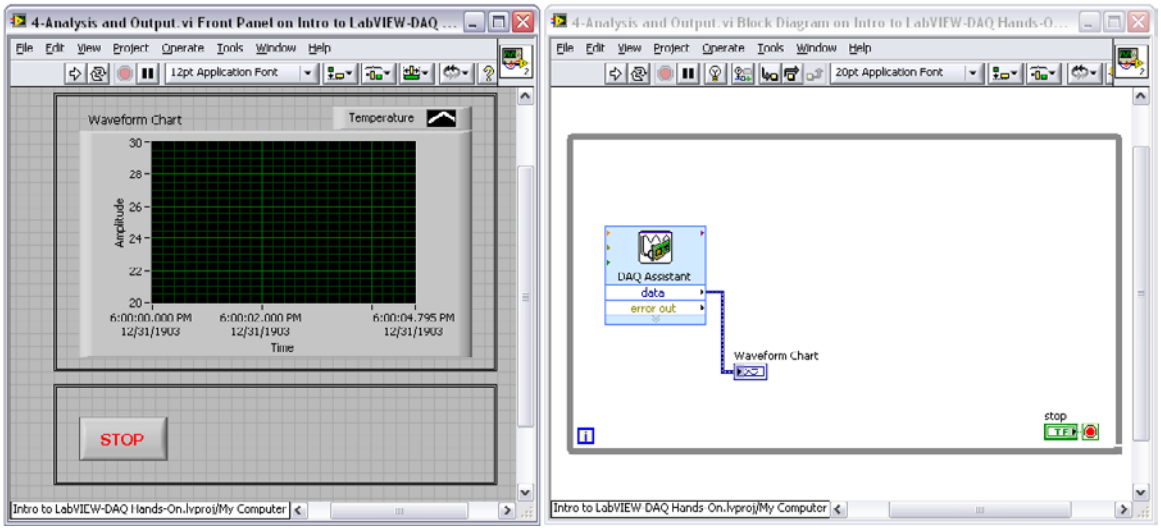
Exercise 7.2: Add Analysis and Digital Output to the DAQ Application

Set up Hardware

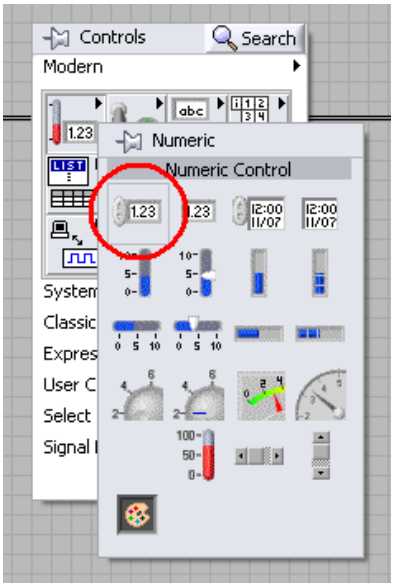
1. Confirm that the CompactDAQ chassis is powered on and connected to the PC via the USB cable. If not, or if it is not behaving as expected, repeat steps #1-8 from Exercise #1

LabVIEW Application – Compare signal to user-defined alarm

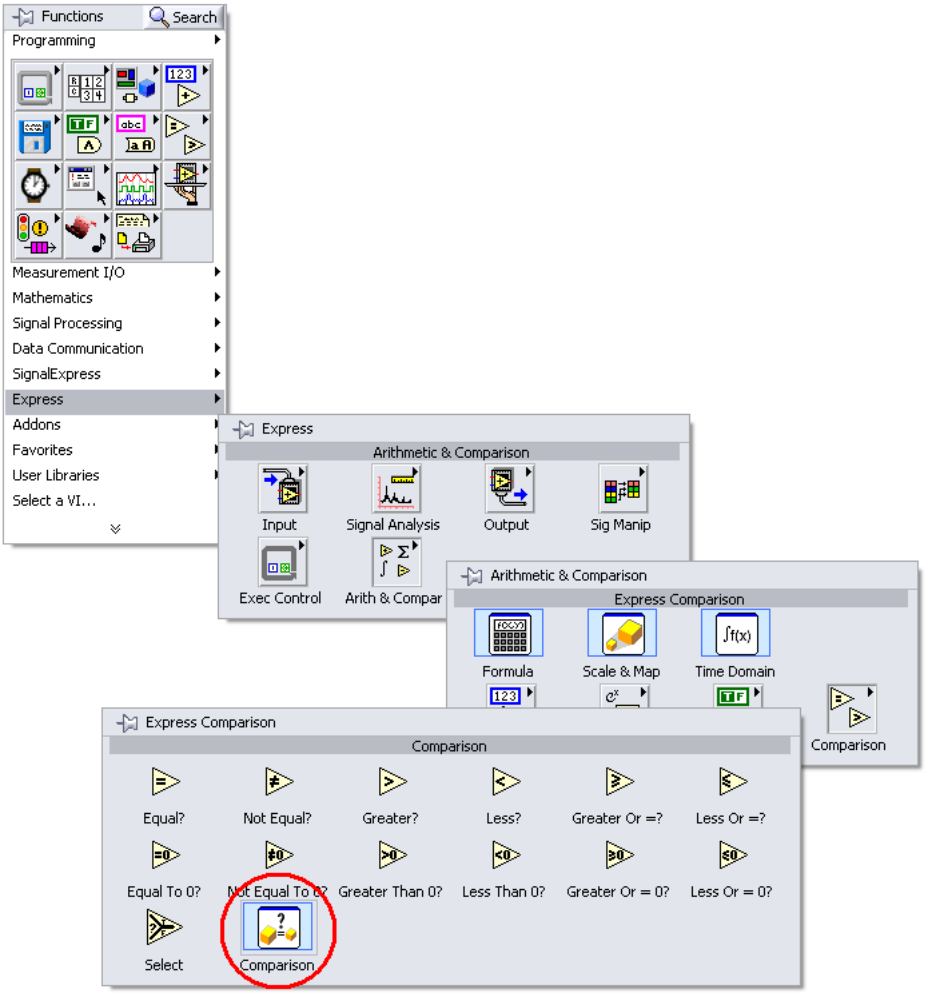
2. Exercise 2 is functionally the same as the end result of Exercise 1. You can open Exercise 1 to synchronize with the illustrations in this section. Open 1-Analysis and Output.vi from the Exercises folder in the Project explorer. The VI will appear like the image below, with additional space on the block diagram to add functionality:



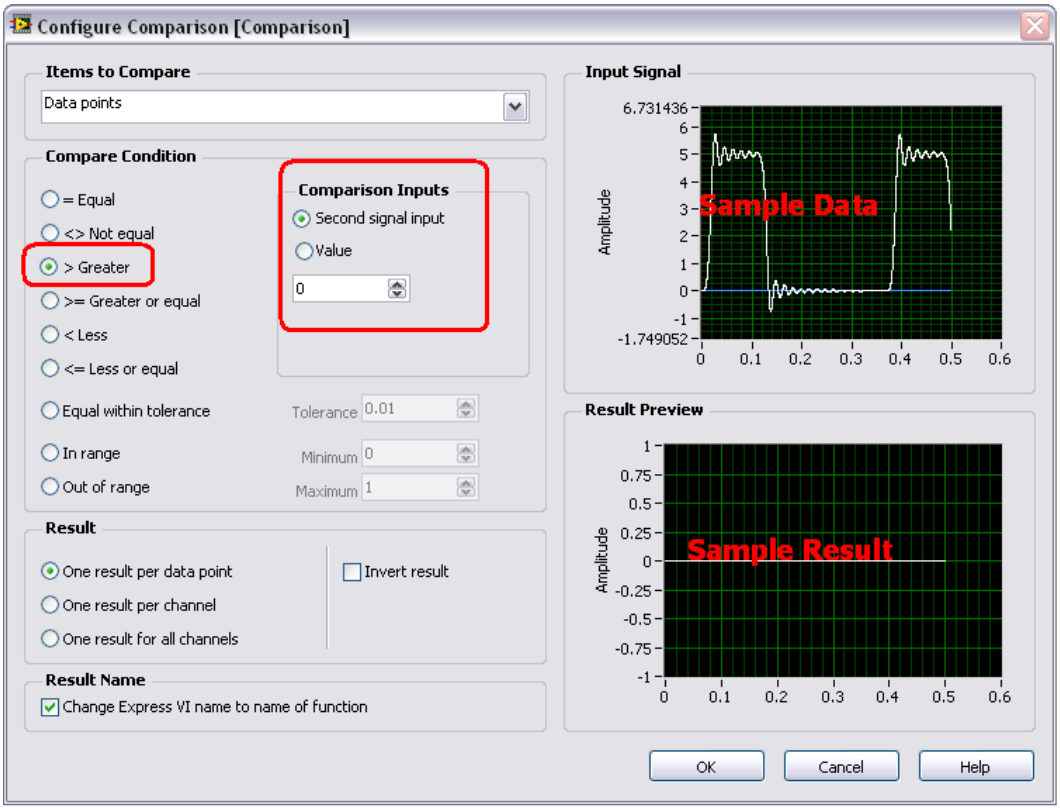
3. Create an alarm that signals if acquired temperature goes above a user-defined level. On the front panel, right-click to open the Controls palette Programming» Numeric and place a numeric control on the front panel.



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"
5. 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.



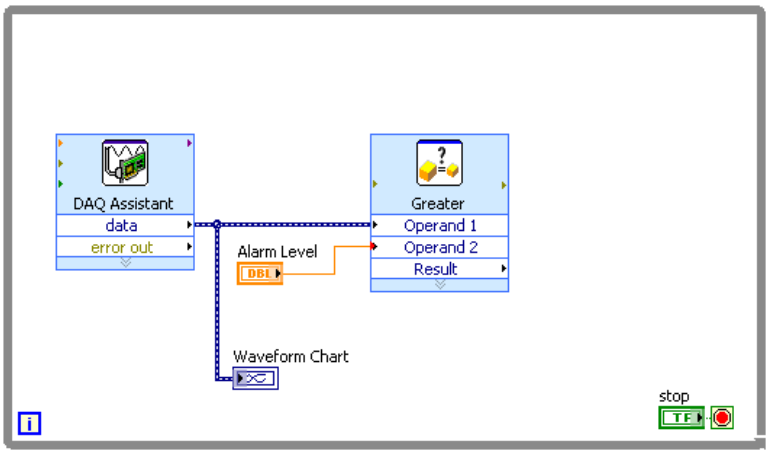
6. Once placed on the block diagram, the Comparison Express VI's configuration dialog will appear.



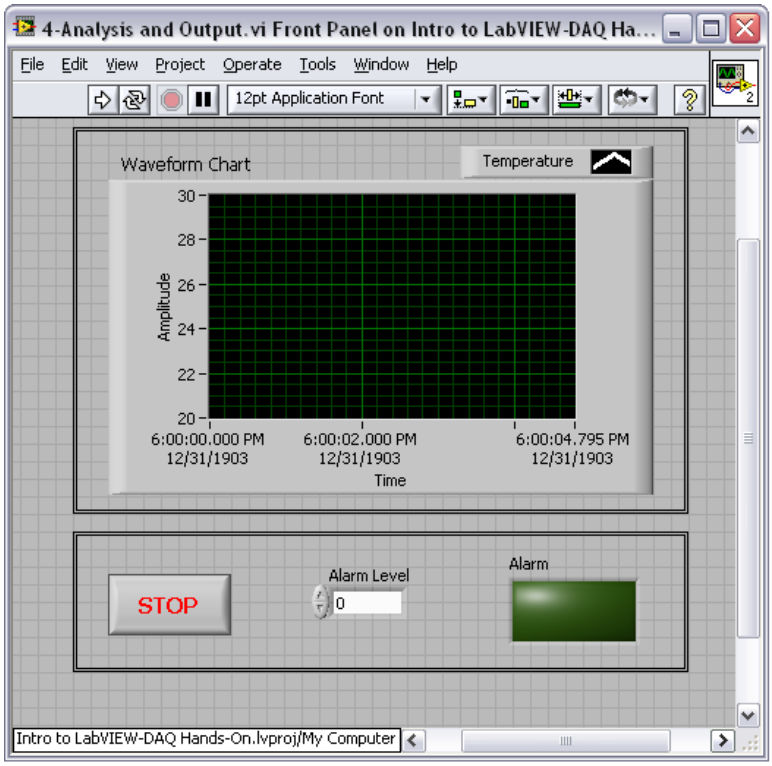
Select "> Greater" in the Compare Condition section and "Second signal input" from the Comparison Inputs section then click OK.

7. 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

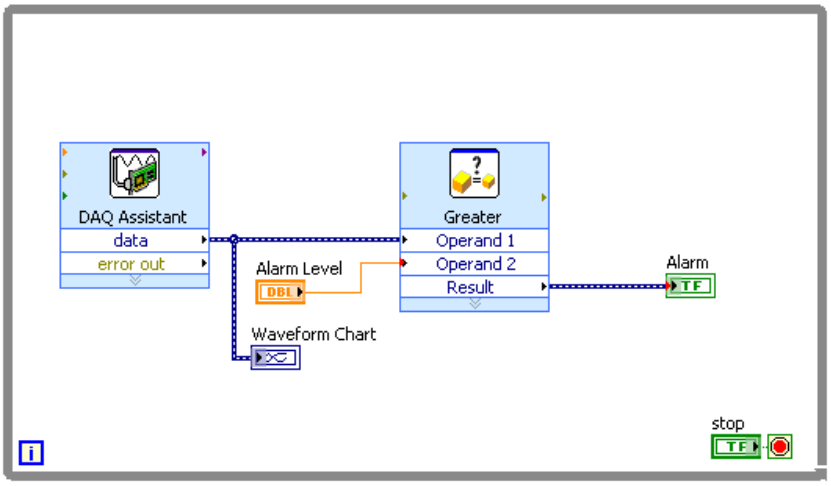
on the Comparison Express VI. Perform the same hover, drag and connect to wire the Alarm Level control and the Operand 2 input on the Comparison Express VI. Your block diagram should now look like this:



8. 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:



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

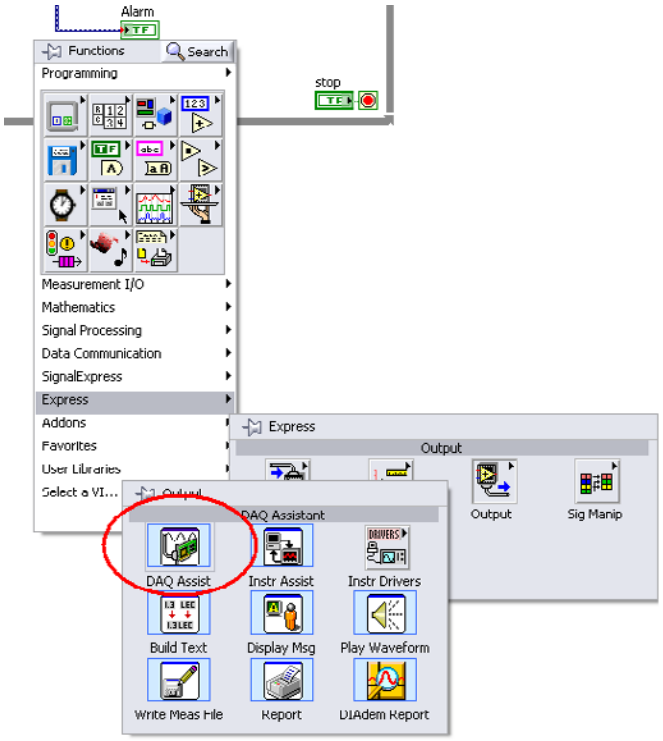


9. Run the application. Press the Run button and then change the Alarm Level control to some level above the current ac-

quired 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.

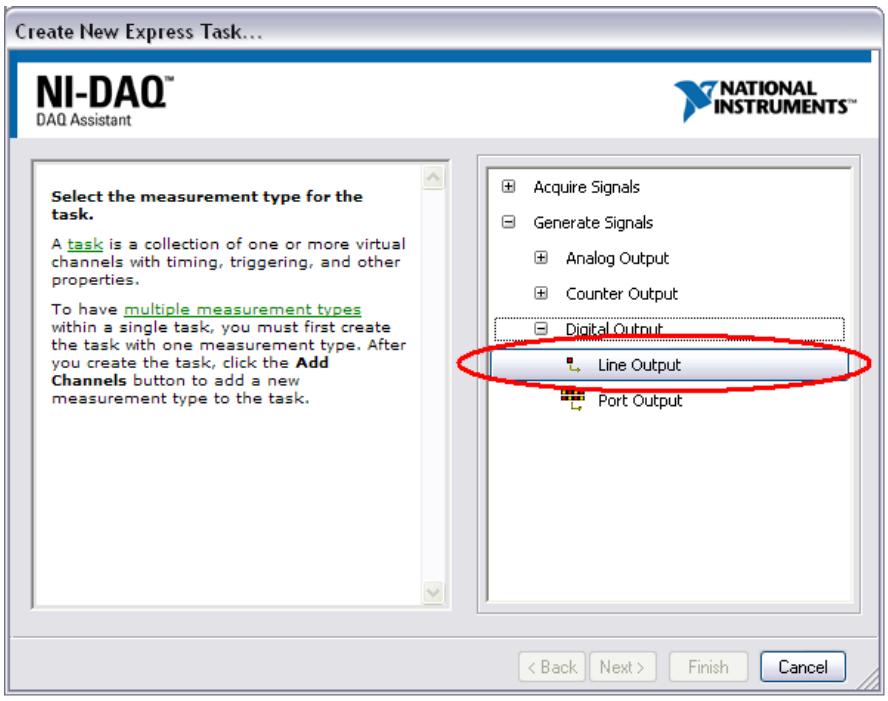
Output Alarm to CompactDAQ Chassis

10. Use another DAQ Assistant Express VI to output Alarm's status to the CompactDAQ's 9474 module. Open the Functions palette on the block diagram and find the DAQ Assistant Express VI at Functions» Express» Output.

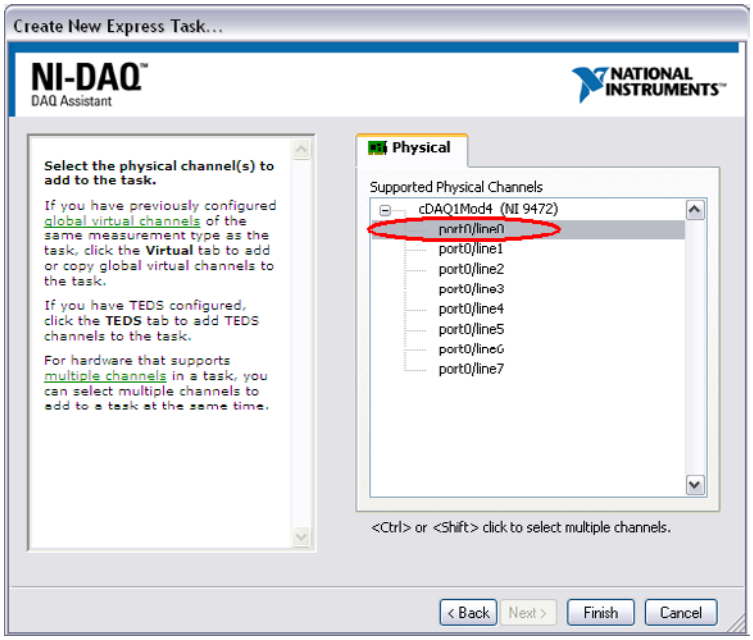


<picture of palette w/ DA circled>

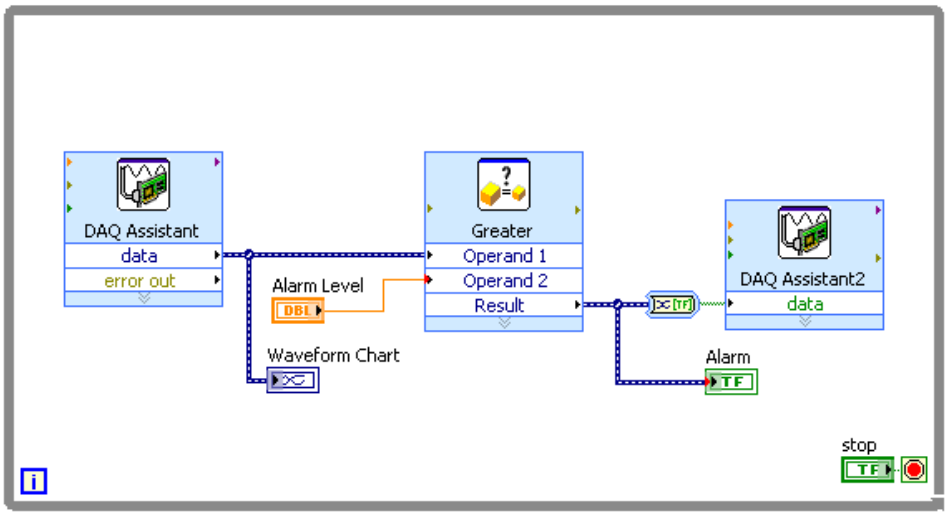
11. Select Generate Signals» Digital Output» Line Output from the Create New Express Task… window.



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



13. Press OK in the DAQ Assistant window that appears, since all of its settings are correct for the application.
14. 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:

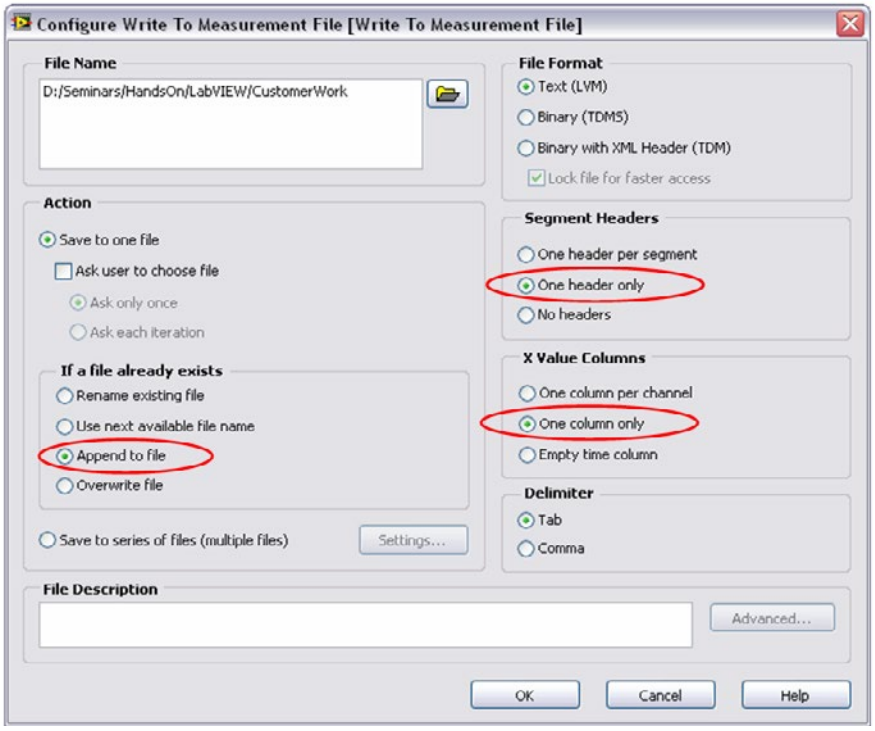


15. Run the VI. Press the Run button. Notice that the LED bank on the CompactDAQ 9474 module turns on and off to match Alarm's value on the front panel.
16. Save and close the VI.

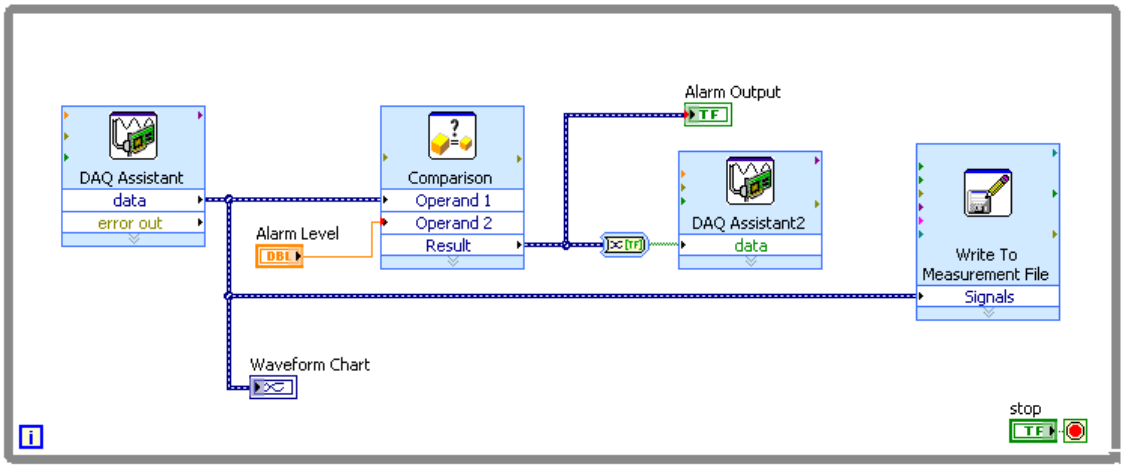
Exercise 7.3: Writing Data to File with LabVIEW

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.



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.

Exercise 7.4: Generate, Acquire, Analyze and Display

Generate a sine waveform using the analog output module. Try to check the signal with an oscilloscope. 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. Try to play with the Nyquist-Shannon sampling theorem.

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.

Exercise 8

ADC basics for TDAQ

Concepts of this lab

Figure 1 below shows the generic signal flow of a Data AcQusition (DAQ) system to perform Analog to Digital Conversions (ADC).

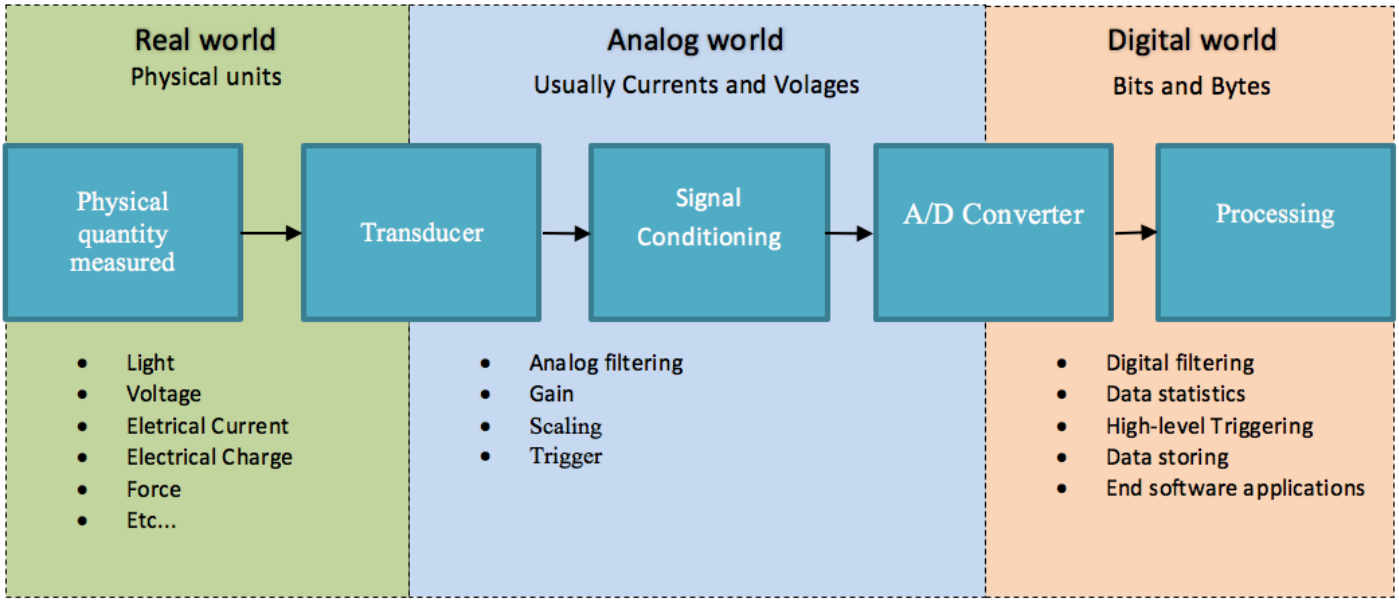


Figure 1- Basic ADC DAQ chain

The scope of this laboratory experience is to understand and experiment with the following components of a Data AcQuisition system:

1. Triggers:
    1. External triggers: accelerator like type of triggers
    2. Triggering on the signal: astroparticle like type of triggers
2. Analog to Digital Converter (ADC): we will try to understand fundamental parameters that come into play when measuring with ADC; as for example its resolution, speed, bandwidth, acquisition window, etc.

Lab setup

Figure 2 below depicts the setup for Lab8.

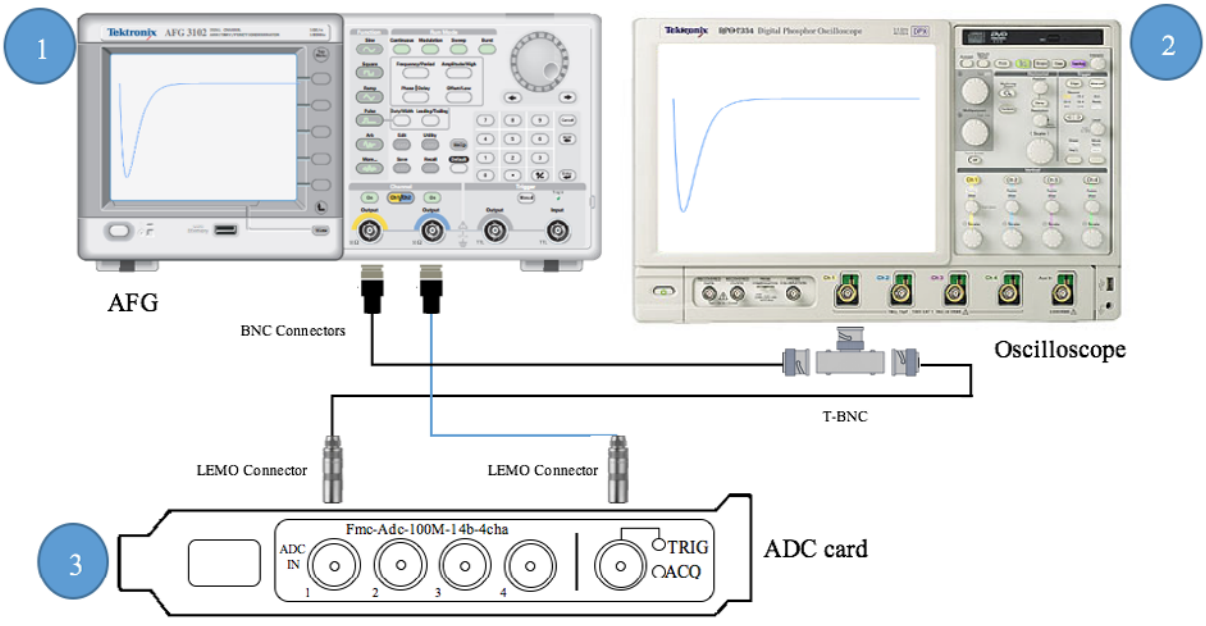


Figure 2 - Equipment setup

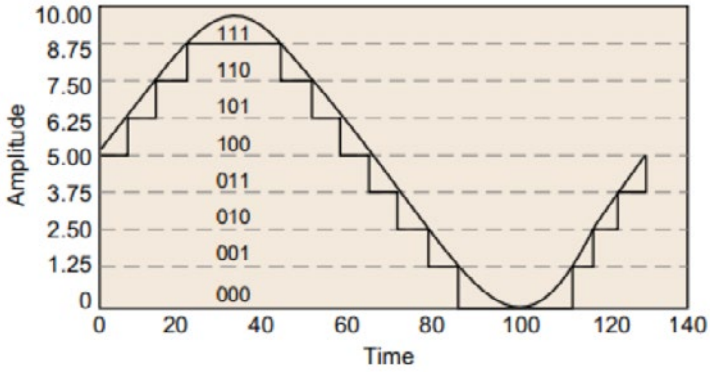
1. Tektronix AFG3252: Arbitrary Function Generator, is the **input** of our system
2. Tektronix DPO70404C: Oscilloscope, it the **monitor** of our system (to cross-check that the signals fed to the system are truly what intended)
3. SPEC+FMC ADC card plus host PC: this is the **core ADC DAQ**
    1. FPGA Mezzanine Card (FMC) ADC: `http://www.ohwr.org/projects/fmc-adc-100m14b4cha`
    2. Simple PCI Express Carrier (SPEC) card: `http://www.ohwr.org/projects/spec/wiki`
    3. Linux based host PC

Introduction to Analog to Digital Conversion (ADC)

An Analog to Digital Converter (also known as "digitizer") is a device which converts the level of an analogue signal into an integer number which is closest to the real value of the signal in terms of ratio. There is a limit for the number of choices of this integer which is determined by the number of bits used for the digitization.

Example: We have a voltage signal whose range is [0 – 10] Volt, and we have a digitizer which has just 3 bits. We can have 8 different integer numbers with 3 bits. Each of these numbers will correspond to a voltage. See table below.

Voltage	Digitizer bits	Integer
0.00 – 1.25	000	0
1.25 – 2.50	001	1
2.50 – 3.75	010	2
3.75 – 5.00	011	3
5.00 – 6.25	100	4
6.25 – 7.50	101	5
7.50 – 8.75	110	6
8.75 – 10.00	111	7



This kind of conversion is performed at regular intervals (**samples**), and the repetition (frequency, Hz) is called **sampling frequency** (normally expressed in samples per seconds). Thus, digitization is not only performed in the signal domain, but also in the time domain.

Also note that to improve the precision of the measurement one should:

- Increase the number of bits, so each corresponding range is small with respect to the signal to be sampled
- Sampling should be done at the proper frequency (see Nyquist frequency); and even more importantly, a precise sampling clock should be used, such that the deviation between consecutive samples is kept as constant as possible.

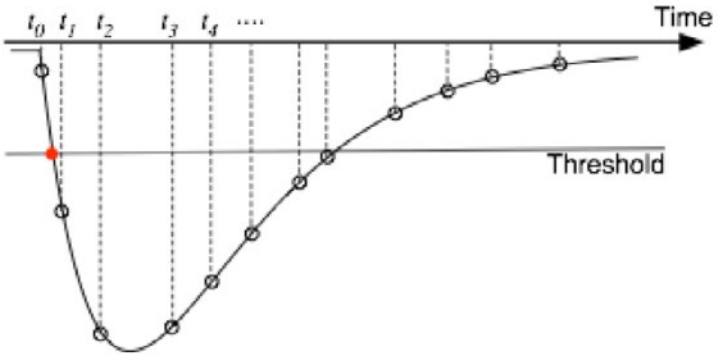
If you are interested in more details about ADC parameters, please check:
`http://www.analog.com/en/analog-to-digital-converters/products/index.html`

If are interested in understanding more in detail the importance of clock stability (jitter), please check:
`http://anlage.umd.edu/Microwave%20Measurements%20for%20Personal%20Web%20Site/Tek%20Intro%20to%20Jitter%2061W_18897_1.pdf`

The ADC card we will be using has 14 bit voltage resolution and a typical conversion time of 10 ns (100 MS/s). In this exercise, we will operate the ADC in order to understand and push its limits.

The Measurement

Qualify in the best possible way a set of signals mimicking real experimental equipment. The tutors will provide some pre-cooked ones (sine waves, positive pulses, scintillator like pulses) but you can try to think of your real world input.



In general terms, a digitization process is characterized by the following:

- Signal range: how much the signal can vary to be correctly interpreted by the device (voltage)
- Sampling frequency: the rate at which it can convert an analogue value to a discrete signal (bits)
- Latency: How long does the device takes to finalize the acquisition process in the chain (time)
- Noises: physical quantities responsible for the signal deterioration (e.g. added noise, intrinsic noise, quantization errors, etc.)

The best measurement is achieved by understanding and controlling those parameters. **The designers (so YOU) have to decide how to setup your TDAQ:** choose a trigger type, define the acquisition windows, find the signal in the window, maximize the scaling for increasing the accuracy of the measurement, etc.

Architecture of a Linux Device Driver for the PCIe Card

Before running the DAQ system, an overview on the communication between the software and the hardware layers and an introduction to the role of device drivers, is fundamental to understand. Figure 3 shows a simplified diagram of the Layers of a Linux Operating System (OS).

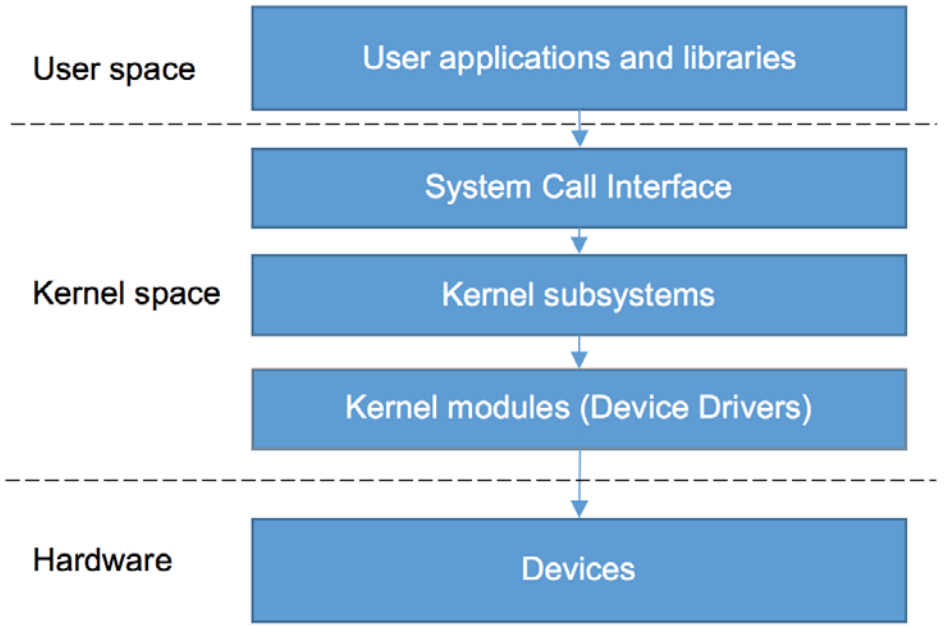


Figure 3- Layers of a Linux OS

The basic control is on the hardware peripheral itself. The lowest level software for this system resides in the kernel of the OS as a "device driver." There are certain control/status registers on the ADC card. These registers can be accessed just like a regular memory access in a C program.

As seen earlier in this document, in our context, the ADC DAQ hardware used is composed of two electronic boards: the SPEC carrier board and the FMC ADC module. The first board is attached directly to a PCIe slot connector on the PC, and is the host for the ADC module. The SPEC card acts as a bridge for the electrical signals of the ADC FMC to be interfaced and converted to PCIe. For the scope of this lab, the "bridging" is done by some "black box" electronics. Thus from a software perspective, one kernel module is instantiated to use the SPEC card, another one for the FMC ADC module.

For simple sensors and devices vendors provide information about their operation and use; which can be seen as a map of internal registers and/or procedures required to perform operations or change their current states (more about in Exercise 12 "DAQ Online Software"). For the sake of simplicity and reusability of software code, software engineers created the concept of frameworks. Among others, this concept was used at CERN for creating a flexible interface for the development of input and output drivers. The target is for very-high-bandwidth devices, with high-definition time stamps, and a uniform metadata representation. Such framework is named the ZIO (with Z standing for "The Ultimate I/O" Framework).

In short, the way ZIO provides to talk to our hardware is through two different channels, one for control and one for data. Figure 4 shows the two data flows.

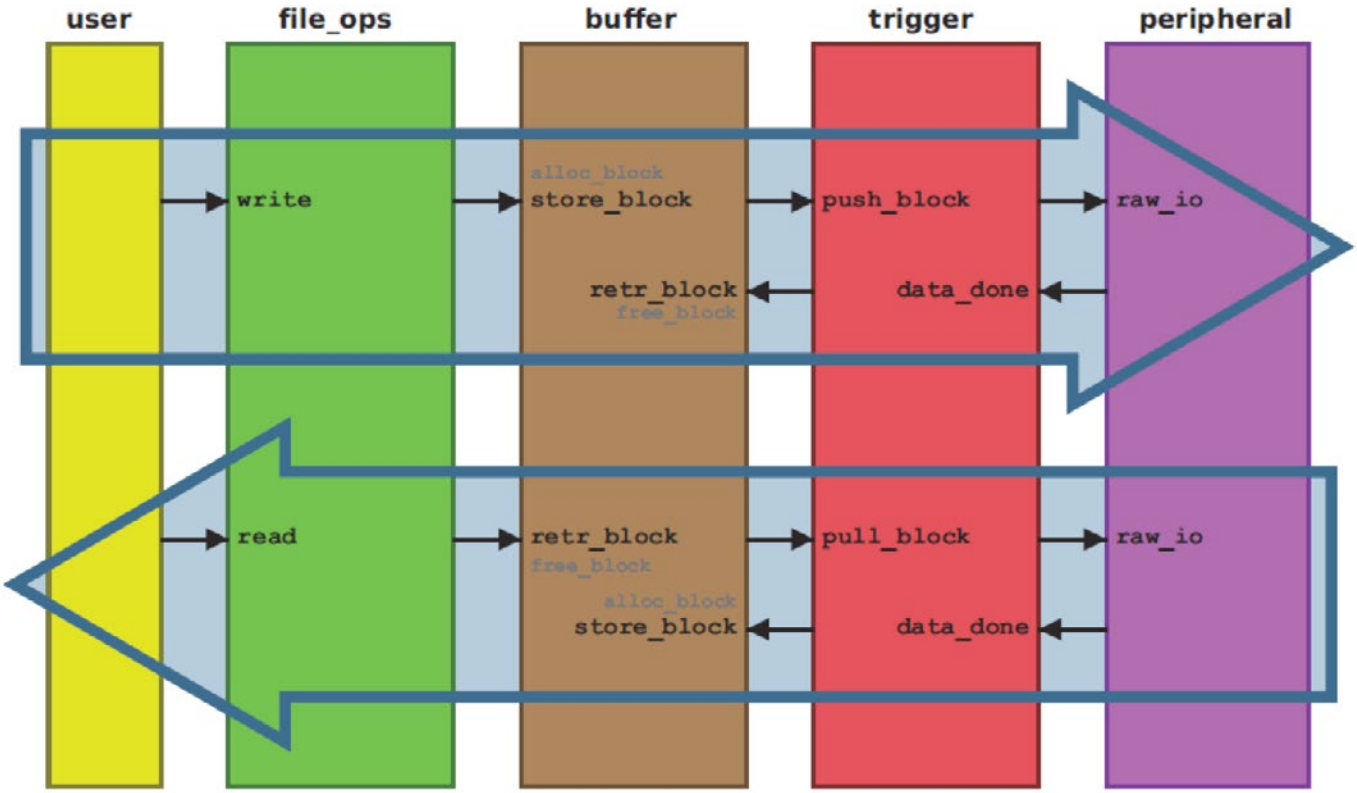


Figure 4 - ZIO data pipeline

When designing a piece of low-level software to communicate to an acquisition device, it is important to choose the way data streams should be passed back and forth the hardware device. To illustrate this they can be qualified in three types: **Polling mode, Interrupt based, and DMA (Direct Memory Access) transfers**.

- **Polling mode:** the CPU checks the state of the device's registers every time it can, this has the advantage of providing a fast and low latency communication and data transfer between them, but at the cost of high CPU load.
- **Interrupt based:** this type of transfers eases the CPU load time as it only have to care about the hardware when it tells the CPU to do so; it has the advantage of a lower impact on CPU loads and fast transfer, but with variable latency.
- **DMA transfers:** relies on a shared memory area the CPU provides to the DMA controller to work on the transfer with the hardware device, this frees completely the CPU from controlling the hardware transfer representing a true concurrent system. Highest of all transfer rates allied to an acceptable latency.

Operating the setup

1. Construct/confirm the experimental setup according to the sketch you find at the beginning of this document. Check that both outputs of the AFG are switched OFF.
   The signal generated by the Channel-1 (**source input**) of the AFG unit is supplied to the oscilloscope, and to the Channel-1 input of the ADC card. Use a T-BNC to split the signal at the AFG output. Also check if the Channel-2 output (**trigger input**) of the AFG is connected to the Trigger input of the ADC card. Again you can monitor the trigger on the scope by spitting with a T-BNC on the AFG output.
2. Using the AFG, set Channel-1 to generate a sine waveform with a frequency of 1 MHz and amplitude of 1 Vpp. Completed this operation, set Channel-2 to generate a pulse waveform with a frequency of 1 KHz and amplitude of 4 V. Don't turn them ON yet, and check if they have not an amplitude above 5 Vpp.
3. Login to the PC and reset the lab8 directories, so all the work/changes done by the previous group are removed and a fresh copy of the files are installed
4. In order to load the drivers, open the `~/adc directory` and execute the script for loading the drivers:

   ```
   $ cd ~/adc

   $ ./load_drivers.sh
   ```

   You need the root password in order to execute the command above. Ask your tutor for it.
   Spying the content of our script shows the correct order for loading the different kernel modules and set the user permission to talk with the ADC board. It is also possible to check the current kernel modules loaded by issuing the command lsmod.

5. Now it is time to test our ADC, turn ON only Channel-1 of the AFG and check if the signal is correctly displayed by the oscilloscope. Run the acquisition program which will subsequently show an acquisition plot:

```
$ cd Trigger_ext
$ ./fald-acq -a 1000 -b 0 -n 1 -l 1 -g 1 –r 10 -e -X 0100
```

Where acq-program has the following parameters:

```
--before|-b <num> number of pre samples
--after|-a <num> n. of post samples (default: 16)
--nshots|-n <num> number of trigger shots
--delay|-d <num> delay sample after trigger
--under-sample|-u|-D <num> pick 1 sample every <num>
--external|-e use external trigger
--threshold|-t <num> internal trigger threshold
--channel|-c <num> channel used as trigger (1..4)
--range|-r <num> channel input range: 100(100mv) 1(1v) 10(10v)
--negative-edge internal trigger is falling edge
--loop|-l <num> number of loop before exiting
--graph|-g <chnum> plot the desired channel
--X11|-X Gnuplot will use X connection
```

Did the program made the acquisition?

6. Turn ON Channel-2 of the AFG. Does it run now?
7. Now that we know that the ADC is working properly we can go to some real time data analysis. Bearing this in mind, there is a small program called `V_t_continous.C` which uses the ROOT framework functionalities and runs on top of `fald–acq`. To run `V_t_continous.C` issue the command:

```
$ root V_t_continous.C
```

8. Try now to modify Channel-1 frequency in steps of Hz while checking if the acquired waveform remains true to it. Check also if the measured amplitude is the same as set on the AFG or, say, twice its value. **Make sure you are changing the frequency NOT the amplitude.**
9. One issue you may get while changing the frequency is the 'non-stopping' graph, meaning it is continuously sweeping horizontally. What causes this?
Since we are using Channel-2 of the AFG as an external trigger, its triggering frequency dictates how, or at which point in time, the acquisition of Channel-1 signal starts. In practice: if the frequency of our sine wave is not a harmonic of Channel-2 pulse, i.e. not an integer multiple, the ADC doesn't capture the signal at the same phase.

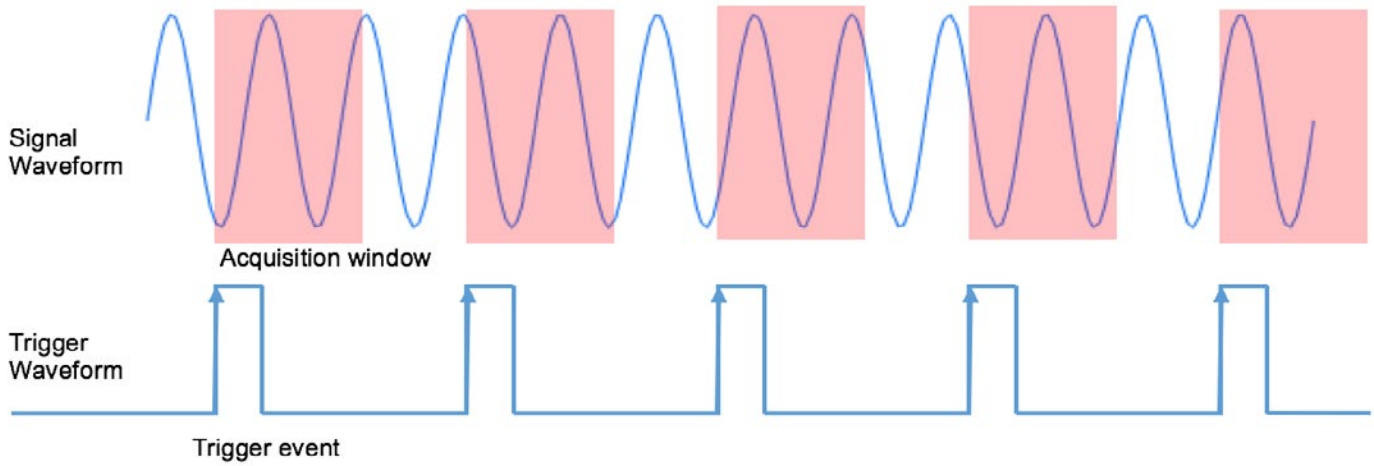


Figure 5- Signal frequency non multiple of Trigger frequency

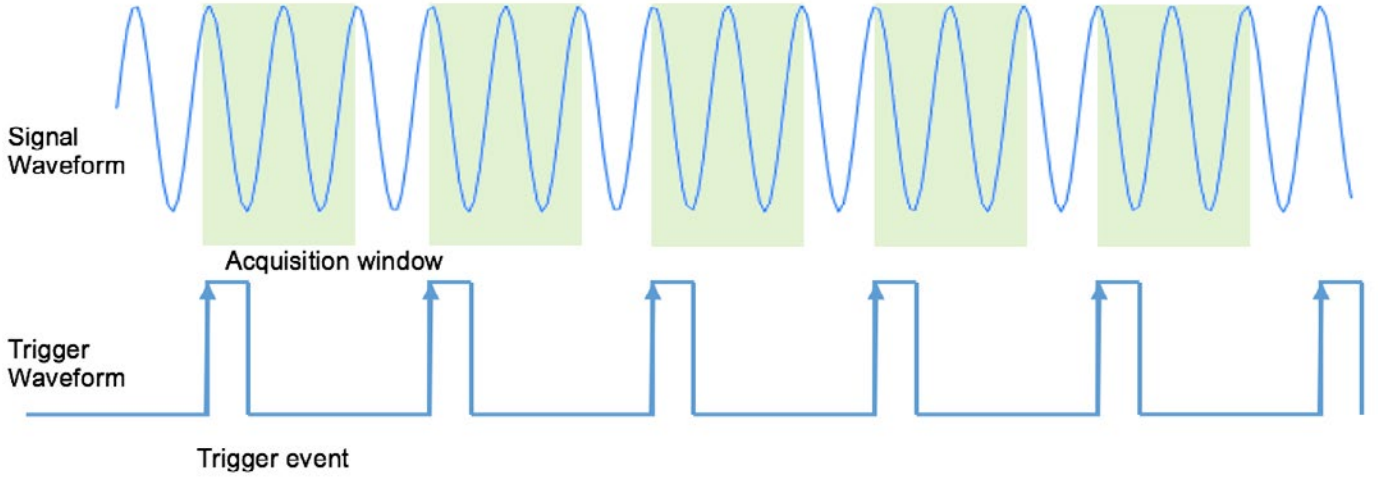


Figure 6 - Signal frequency multiple of Trigger frequency

10. Going further, let's push the frequency of our signal to the limits of the ADC capabilities. Recalling that our ADC specifications say that the default sampling rate is 100 Million Samples/s try to increase the signal frequency in steps of MHz. Please note: it is recommended to decrease the number of samples in our acquisition window; otherwise it would become hard to analyze the signal in a cloud of points.
    For this, open the V_t_continous.C file with your favourite editor and change the number of post samples (-a parameter) to 100 or less, it is located on the line which calls fald-acq.
    Run root V_t_continous.C again.
    Can you see the relation between the acquisition window and the signal speed?

11. You can see that the closer you get to 100 MHz the worst the acquisition signal looks like. Can you define the maximum AFG signal frequency where our sine wave keeps its shape in a single acquisition shot (i.e. you can still see a sine wave with the same frequency as the original signal)?
    This value is called the **Nyquist frequency**. The **Nyquist theorem** states that: *the minimum sampling rate of an acquisition device must be at least two times the maximum frequency of the original signal, otherwise information would be lost in the process.* See Figure 7 below: the original signal is in blue, the sampling points are pointed by the black arrows and the acquired data is represented in orange. When this criteria is not respected an effect called aliasing appears.

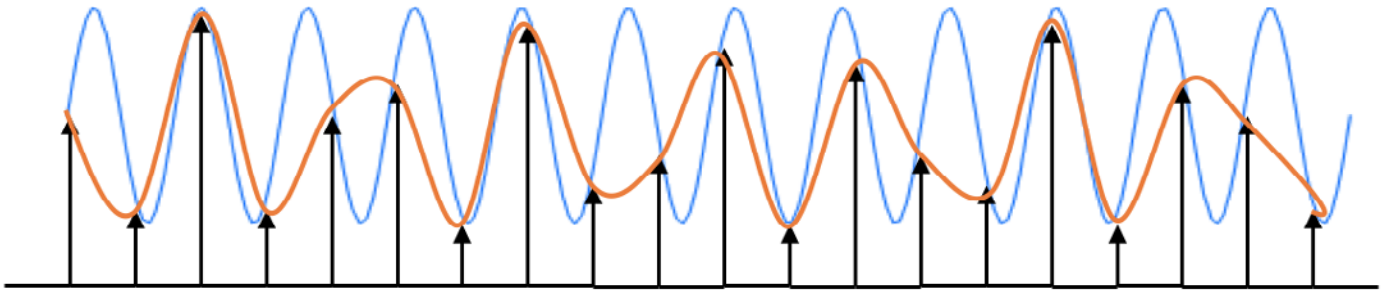


Figure 7 - Aliasing effect

12. The maximum Nyquist frequency is computed on a signal ideally composed of a single sinusoid. In fact any real function of a complex signal is mathematically described as the sum of a series of trigonometric functions in the **frequency domain** (called *signal spectrum*) instead of time. **Fourier series and transforms** base their analyses on this concept. The spectral representation of a sine wave is rather simple: it is only a single line centred around the frequency it coverts from the time domain. The spectral content is more complex for different waveforms signals such as square, saw-tooth and other more complex signals.
    In order to check how our acquisition systems behaves with complex signals go back to the frequency of 1 MHz but change the type from sine-wave to square signal. Increase the frequency to values below Nyquist criteria.

Ask the tutor to present a real time Fast Fourier Transform (FFT) and **ask further questions!**

Acknowledgements

Andrea Borga (andrea.borga@nikhef.nl, initiator and tutor of this lab in ISOTDAQ2015)
Cairo Caplan (cairo.caplan@cern.ch, initiatior and lab assistance in ISOTDAQ2015)
Diogenes Gimenez (diogenes.gimenez@usp.br, initiator and lab assistance in ISOTDAQ2015)

Exercise 9

Networking for Data Acquisition Systems

The purpose of this laboratory is to introduce you to the field of networking and data networks for High Energy Physics experiments.

I. Introduction

A computer network or data network is a telecommunications network that allows computers to exchange data. In computer networks, networked computing devices pass data to each other along data connections. The connections (network links) between nodes are established using either cable media or wireless media. The largest and best-known computer network is the Internet.

A switch is a networking device used to connect many devices (e.g. computers) together on a Local Area Network (LAN). A router is a networking device used to connect Local Area Networks (LANs) together. A network administrator has limited control over the traffic between devices on the same LAN and a much higher degree of control over the traffic between devices found in different LANs. To avoid this limitation one will usually group devices in LANs by their function, user group or security restrictions then will use routers or advanced switches with routing capability to connect these LANs together.

To ease the network design and improve network efficiency, management and security, most switches today support a logical grouping of physical ports known as a Virtual LAN (VLAN). If 2 devices are connected to the same VLAN they behave as being in the same LAN, and if 2 devices are connected to different VLANs, even if they are connected to the same physical switch, they behave as being connected to different LANs.

Most data acquisition systems have a computer network used to build and filter the collision data recorded by the detector. All the computers in the DAQ network are interconnected using a set of standard switches and one (for small DAQ systems) or several (for more complex DAQ systems) core networking equipment such as advanced switches or routers. The DAQ networks today make extensive use of the VLAN and routing technology to logically group and control the packet flows generated by the readout, collector, filtering or storage DAQ equipment.

To monitor a computer network, protocols such as SNMP (Simple Network Management Protocol), sFlow or NetFlow can be used to gather statistics from networking devices. SNMP is one of the most simple and widely used protocols for network monitoring and configuration. SNMP exposes management data in the form of variables on the managed systems, which in turn describe the system parameters. These variables can then be queried (and sometimes set) by management or monitoring applications. In a typical SNMP use-case an administrative computer has the task of monitoring the traffic load of a group of hosts or devices on a computer network. Each managed system (also called a slave) executes a software component called an agent, which reports information via SNMP to the managing systems (also called masters).

The information gathered via the SNMP protocol (ex: number of MB/s going in or out from one port, discarded or erroneous packet rates, interface speed etc.) can be stored in a standard database (like Oracle, MSSQL, MySQL etc.) or in RRD (Round Robin Database) files. To create, store and retrieve data from an RRD file the rrdtool application and library can be used.

In order to test network performance and troubleshoot potential problems, the iperf/iperf3 utility is used extensively. It allows the generation of different types of traffic flows with different characteristics and therefore allows the identification of performance tuning actions needed to improve the network behavior. This process is particularly important in data acquisition networks due to their demanding requirements in term of performance and their specific traffic flows. In some situations, the traffic also needs to be classified and prioritized using Quality of Service features so as, for example, to avoid that critical control messages are lost in different congestion points inside the network.

II. Objectives

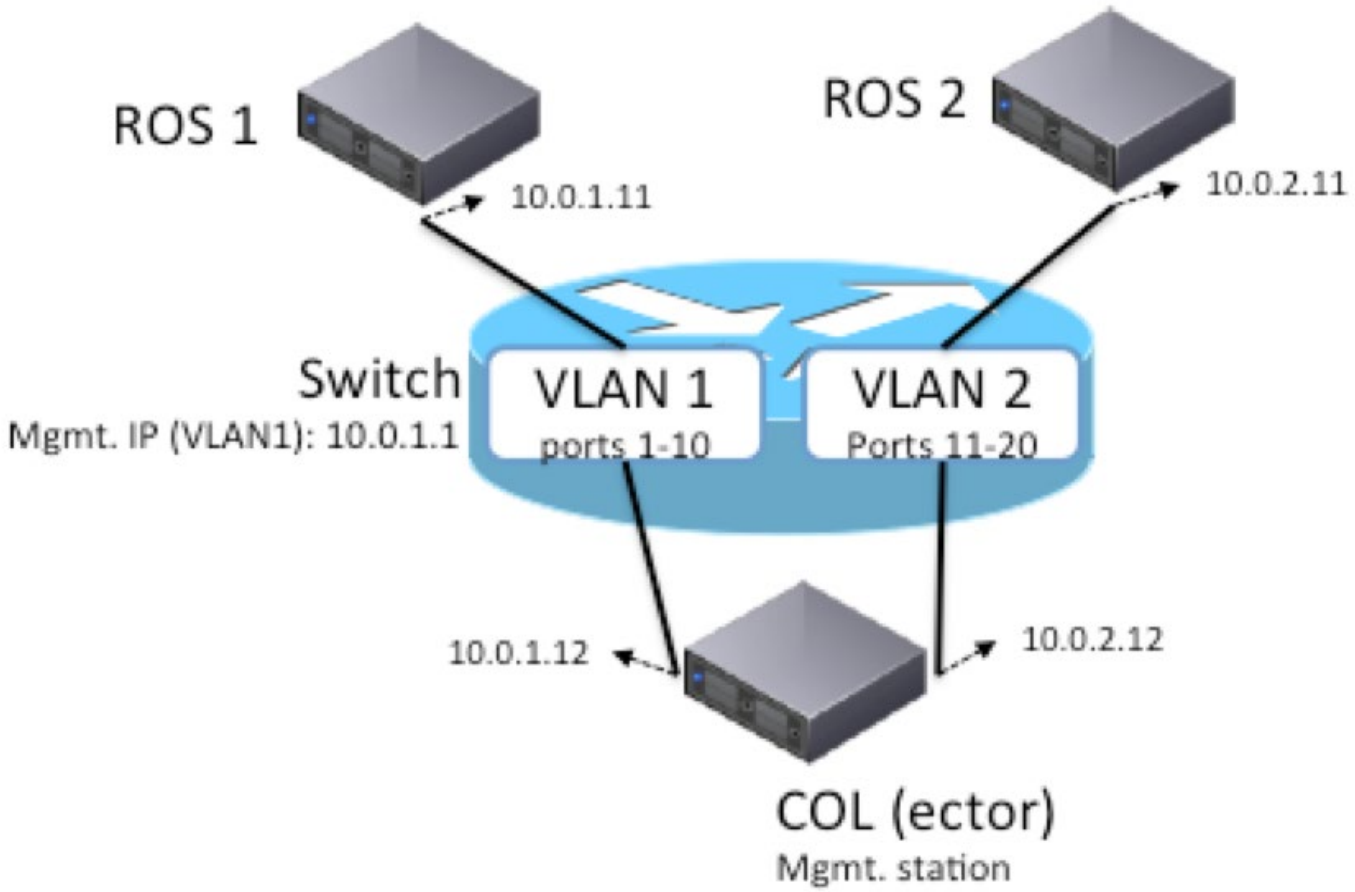
To simulate a very basic DAQ network we will use 2 computers (ROS1 and ROS2) as readout systems, a switch with management capabilities as the DAQ network and another computer (COL) to act as a data builder/collector.

In the first part you will configure the DAQ network (i.e. the switch) to be able to manage and monitor it remotely. Then you will configure 2 VLANs and map the existing ports to one or the other accordingly to the setup described in the following section.

In the second part, you'll configure a set of monitoring scripts to match the existing configuration and use them to monitor the traffic load on the connected ports. With the help of a network analyzer tool called Wireshark, you'll have a look at different types of traffic (ICMP, SNMP, Telnet) and try to identify the underlying protocols and Ethernet frame and IP packet composition.

In the third part of the lab, you will use the iperf3 utility to generate different types of traffic flows between hosts and analyze their performance characteristics. The last exercise will demonstrate how Quality of Service can be used to prioritize traffic.

III. Setup



IV. Step-by-step Guideline

1. Basic and VLAN Configuration

Power up the switch. Connect the cables between the computers and the switch. Check the connectivity light. (At the beginning, it doesn't matter what port numbers you choose to use for connecting the PCs);

Login to the COL PC (user: student, password: student);

Connect the serial cable (DB9-DB9) + serial-to-USB converter + USB cable from the serial port on the switch to an USB port on the COL computer;

Start the "screen" application to connect to the switch via the serial console, enter the privileged mode (enable) , display and analyze its default configuration:

```
[student@col] $ sudo screen /dev/ttyUSB0
sw-daq> enable
sw-daq# show running-config
```

Configure a management IP address on the switch:

```
sw-daq# configure terminal
sw-daq(config)# vlan 1
sw-daq(vlan-1)# ip address 10.0.1.1 255.255.255.0
```

Have a look at the configuration again (**show running config**). What changed from the previous version? How many VLANs are present in the configuration?

Now the switch can be managed remotely. Try to ping and then telnet the switch from the COL computer:

```
[student@col] $ ping 10.0.1.1
[student@col] $ telnet 10.0.1.1
```

Configure a second VLAN on the switch to match the setup described in the diagram in Section III:

```
sw-daq (config)# vlan 2
sw-daq (vlan-2)# name Vlan2
sw-daq (vlan-2)# untagged 11-20
```

Connect the cables to obtain the setup described in the diagram in Section III. Check that the COL – ROS1 and COL-ROS2 computer pairs can still communicate with the help of the ping command. Move the ROS2 cable to port 5. Test the communication between COL and ROS2 using ping. Can you explain what happened? At the end, move the ROS2 cable back to the original port.

2. Network Monitoring and Traffic Analysis

Request some simple information from the switch via SNMP using `snmpget`:

```
[student@col] $ snmpget –v 2c  –c public 10.0.1.1 sysDescr.     #switch description
[student@col] $ snmpget –v 2c  –c public 10.0.1.1 ifInOctets.1  #input traffic for port 1
```

SNMP uses two representations for OIDs: numerical and human-readable. You can use `snmptranslate` to translate between the two:

```
[student@col ]$ snmptranslate -On IF-MIB::ifInOctets.1
                .1.3.6.1.2.1.2.2.1.10.1
[student@col ~]$ snmptranslate .1.3.6.1.2.1.2.2.1.16.3
IF-MIB::ifOutOctets.3         #output traffic for port 3
```

Have a look at the following files located in the swmon folder and understand the role of each:

```
[student@col] $  cd ~/swmon
[student@col swmon]$ ls *.sh
switch_graph.sh  switch_stats_create.sh  switch_stats.sh
```

Run switch_stats_create.sh and look for the newly created `switch_stats.rrd` file:

```
[student@col swmon] $  ./switch_stats_create.sh
[student@col swmon]$ ls *.rrd
```

Verify and, if needed, modify the switch IP address used in `switch_stats.sh`. Then run the script to start polling the switch:

```
[student@col swmon]$ ./switch_stats.sh &
```

Open another console and run `switch_graph.sh` to start generating traffic plots:

```
[student@col swmon]$ ./switch_graph.sh &
```

Open the URL `file:///home/student/swmon/index.html` in a Web browser. This Web page displays images containing input and output traffic for the first four ports on the switch.

Modify `switch_stats_create.sh, switch_stats.sh, switch_graph.sh, index.html` to start monitoring only on the active ports. Try to understand what generates the traffic shown in the plots and identify the direction (in/out). Execute additional commands (ping,telnet) to generate more traffic and observe the changes in traffic.

Use `wireshark` to have a detailed look at the network traffic. Identify the type of traffic that is currently flowing thought the network (ARP, ICMP, SNMP, Telnet). Open certain packets, analyze their structure and try to identify important fields such as source/destination MAC address, source/destination IP address, Layer 4 protocols, etc.

```
[student@col]$ sudo su –
[root@col] # wireshark
```

3. Network Performance Testing and Tuning

On the COL computer, start an iperf3 server, binding it to one of its IP addresses:

```
[student@col]$ iperf3 -s –B 10.0.1.12
```

On the ROS1 node, start an iperf3 client which connects to the previous server. This will generate a TCP network flow between the two

hosts running close to link speed (1Gb/s). Analyze the command output: why is the reported bandwidth lower than the link speed?

```
[student@ros1l]$ iperf3 -c 10.0.1.12
```

On the ROS1 node, start an iperf3 client and connection but this time using UDP. Analyze the command output: do you notice any differences in comparison to the TCP example? Do you notice any discarded packets reported?

```
[student@ros1]$ iperf3 -c 10.0.1.12 —u —b 1G
```

On the ROS1 node, start an iperf3 client and connection using TCP traffic but lowering the TCP window size. Analyze the command output: do you notice any differences in comparison to the initial TCP example?

```
[student@ros1]$ iperf3 -c 10.0.1.12 —w 16k
```

On the ROS 1 node, start an iperf3 client and connection using TCP traffic but lowering the MTU size window size. Analyze the command output: do you notice any differences in comparison to the initial TCP example?

```
[student@ros1]$ iperf3 -c 10.0.1.12 —M 750
```

4. Traffic prioritization (Quality of Service)

First of all, you'll need to change the network setup so that all three nodes (ROS1, ROS2, COL) are part of the 10.0.1.0/24 network, associated to VLAN1. For this, you'll first need to connect to the ROS2 node using the management network and modify the data IP address:

```
[student@col]$ ssh <ros2_mgmt_ip_address>
[student@ros2]$ ifconfig       #identify the name of the data interface (in the format eth<X>)
[student@ros2]$ sudo su —
[root@ros2]# vim /etc/sysconfig/network-scripts/ifcfg-eth<X>
[root@ros2]# ifdown eth<X>;ifup eth<X>     #reinitialize the interface to take the new IP address
[root@ros2]#exit
[student@ros2]$ ping 10.0.1.11             #test that ROS1 can be reached from ROS2
```

Why doesn't the last ping work? Imagine the changes that need to be done so that the new setup has all three hosts connected using a single physical link.

You'll then configure the switch to define which traffic flow has priority based on an IP field called ToS (Type Of Service). Two ToS values will be matched to corresponding priority queues. Also QoS prioritization based on ToS needs to be enabled:

```
sw-daq# conf t
sw-daq(config)# qos dscp-map 000001 priority 1
sw-daq(config)# qos dscp-map 000111 priority 7
sw-daq(config)# qos type-of-service ip-precedence
```

Using iperf3, you'll then generate two TCP traffic flows with different ToS values and running at maximum speed (~1Gbit/s). The traffic sources are two different hosts (ROS1 and ROS2) but the destination is the same (COL) which means that both flows (total of 2Gbit/s) will compete on a single 1Gbit/s port. Based on the configuration, you'll observe which traffic flow has priority.

On the COL node, start two iperf3 servers using different ports:

```
[student@col]$   iperf3 -s —B 10.0.1.12 —p 5001 &
[student@col]$   iperf3 -s —B 10.0.1.12 —p 5002 &
```

On the ROS1 node, start an iperf3 client generating TCP traffic with low priority(1) for 30 seconds. In parallel, on the ROS2 node, start an iperf3 client generating TCP traffic with high priority(7) for 10 seconds. Observe what happens when the two sessions run in parallel:

```
[student@ros1]$ iperf3 -c 10.0.1.12  -t 30 -p 5001 -S 8  # this is low priority (1)
[student@ros2]$ iperf3 -c 10.0.1.12  -t 10 -p 5002 -S 56  # this is high priority (7)
```

On the switch side, you can also observe discarded packets on the specific output queues of the destination port (the port where COL is connected):

```
sw-daq# conf t
sw-daq(config)# qos watch-queue <destination_port> out
sw-daq#exit
sw-daq# show interface queues <destination_port>
```

VI. Questions[1]

- What is a host?
- What is an IP address?
- What is a MAC address?
- What is the difference between a Layer2 Switch and a Router?
- What is the purpose of the ARP protocol?
- What happens if you have a static entry in the ARP cache and the NIC for that target computer is changed?
- If IP determines that the packet that it is currently processing is destined for a remote subnet, where does IP send the packet?
- How could you find the physical address of the Ethernet card installed on your computer?
- What is the purpose of the TTL field in the IP frame?
- You are the network administrator of a Class C network. Your network consists of 100 computers. Your ISP assigns the address 137.138.111.0/24 to your network. Your network requires 10 subnets with at least 10 hosts per subnet. Which subnet mask should you configure to meet this requirement?
- What is the dotted decimal notation of subnet masks for the following IP addresses?

  192.168.10.1/23

  5.5.5.5/16

  203.40.21.58/27

  9.2.3.1/9
- What is the prefix notation of the following subnet masks?

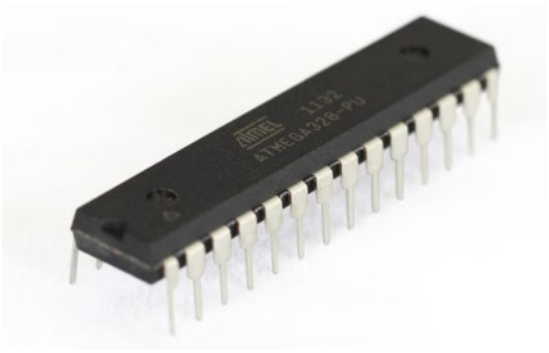
  255.255.0.0

  255.248.0.0

  255.255.255.255
- IP Fragmentation. Using wireshark to start a new capture. Ping another host using packet size=2900. Stop the capture and view the captured frames. What do you notice?

1 You will not be able to answer the questions based on the material from the ISOTDAQ school only. You will need to dig for information yourself (Google is your friend). This is just a list of things you should look for and understand, if you are interested in networking.

Exercise 10
Microcontrollers



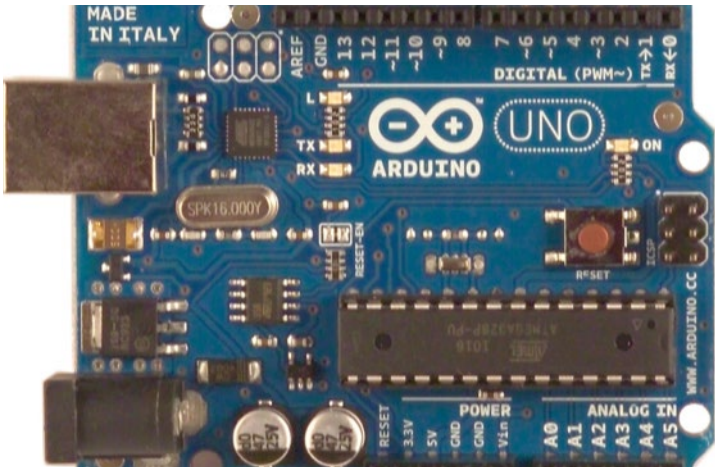
Introduction

Microcontrollers are small computers with all it's components (CPU, memories, peripherals) integrated on the same chip. Apart from their capability of processing data, they are low power, usually inexpensive devices that easily interfaces with sensors and actuators, making them perfect to use in embedded systems.

On this lab we are going to learn the basics about microcontrollers, how to use and program it, as well as common applications and explore the most relevant peripherals through the hands-on exercises and a simple project.

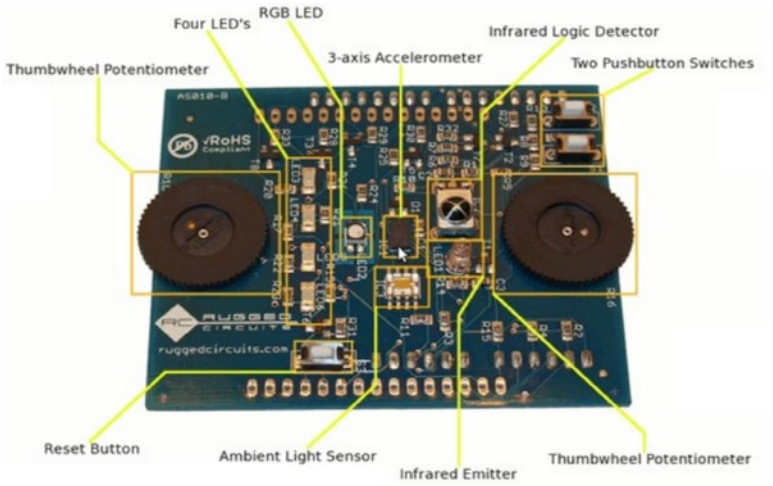
Arduino

Arduino is an open-source electronic prototyping platform based on easy-to-use software and hardware. In short, it is the most popular microcontroller development board, with a lot of "shields" (extension boards that you pile on top of the Arduino) to add functionalities and libraries to use the most common sensors and devices used with microcontrollers.



Gadget Shield

Together with the Arduino, we are going to use the Gadget Shield, which contains 4 LEDs, 2 push-buttons, 2 thumbwheel potentiometers, one RGB LED, one accelerometer, one ambient light sensor and an Infrared receiver and emitter.



Before you start

The Arduino software and drivers, as well as all the examples of the exercises below are going to be available for download at rivello.me/isotdaq/

Setting up the Arduino

The Arduino official homepage provides a very simple quick start guide. Basically you just have to download the software (the tutor may also provide it), plug the board on the USB port and install it. For further details, visit: http://arduino.cc/en/Guide/HomePage

Once you open the Arduino IDE and the arduino board is plugged, you have to select the board and COM port you're using under the Tools tab.

Exercise and Projects

We are going to do a few basic exercises to learn the basic functions of Arduino and later, using the hardware provided by the tutor, we are going to implement a simple project with the functions we have learned so far.

HANDS ON

Exercise 10.1: Blinking a LED

The basic structure of an Arduino program is very simple: It must contain a setup() and a loop() function. Open the Arduino software, start a new program and write the following structure:

```
void setup() {
...
}
void loop() {
...
}
```

The setup() function is executed once every time the Arduino is powered on. As the name suggests, it should be used for setting up your application, like initializing classes and variables, declaring pin modes, etc.

The loop() function keeps being executed in loop ad eternum. This is where the main logic of your program should be.

To get started, lets build a program to blink a LED. There is already a Led on every Arduino attached to pin 13. You should first of all declare the mode (output or input) of the pin to be used with the following function:

```
pinMode( [pin_number] , [OUTPUT/INPUT] );
```

In our case, the pin_number is 13 and the mode is OUTPUT. So to blink the LED we can write a HIGH and then a LOW signal to the pin 13, adding a delay between each command.

```
digitalWrite( [pin_number], [HIGH/LOW] );
delay( [milisseconds] );
```

Now try to make yourself a program to blink the LED on pin 13 once each second.

Exercise 10.2: Reading the state of a Push-Button

The same way we can declare a pin as output and write a state (HIGH or LOW) to it, we can also declare one as INPUT and read its state with the following function:

```
boolean_variable = digitalRead( pinNumber );
```

It returns TRUE or FALSE (HIGH or LOW). Let's use it to read the status of the Push-Button S1, which is on pin 12 of the Gadget Shield. (Disconnect the Arduino from the power before inserting/removing the shield)

Lets use the code from exercise 1 and make the pressing of the button change the blinking pattern. Stopping it, or maybe speeding it up? It's up to you.

Exercise 10.3: Serial communication

In this exercise, we're going to use the Arduino Serial library to send and receive characters from the PC using the Serial Monitor tool of the Arduino IDE. First thing to do is to initialize the Serial with the following command:

```
Serial.begin( 9600 );  // (9600 is the baudrate).
```

As it only needs to be executed once, it should be on the setup() function. Now there are 3 more important functions to learn. The available() returns whether there is a character available to be read:

```
boolean_variable = Serial.available();
```

The read() reads into a variable a single character from the serial buffer, and the println() works like in C, printing on the serial port a string. It also converts numbers into characters. The Serial library is very handy and there are more functions. For reference visit: http://arduino.cc/en/Reference/Serial

```
inByte = Serial.read(); // reads into inByte a character from the buffer.
Serial.println("Hello World");  // Writes a string to the serial port.
```

Now lets do a program to control the Led on pin 13 from the serial port and send a message whenever the button on pin 12 is pressed. We actually want to detect whenever a change in the state of the button occurs. For this reason we need 2 variables. One to store the current value and one to store the value from the last read. When it transitiones from HIGH to LOW, the button was pressed. When from LOW to HIGH, it was released. See the algorithm below and try to implement with the above given functions:

```
button_before = button_now;
button_now = digitalRead(12) ;
if ( before was HIGH and now is LOW) {
        Send a message saying: ("Button pressed!");
}
```

Now open the Serial Monitor (Tools -> Serial Monitor), press the button and see if the message arrives. Do the same for when the button is released.

What about controlling the LED on pin 13 from the Serial Monitor? Try yourself and ask the tutor if you need help.

Exercise 10.4: Reading an analog input

The Arduino UNO has 6 analog inputs. Reading one of them is as easy as reading a digital pin. It returns an integer value ranging from 0 to 1023. The scale varies from 0V to a reference voltage, which is by default 5V (but can be changed). In short: 0 -> 0V; 512 -> 2.5V; 1023 -> 5V. To read an analog pin, uses the following function:

```
integer_variable = analogRead( pin_number );
```

Now try the following code (just copy and paste into a new arduino file):

```
int analog_value0, analog_value1;
void setup() {
  Serial.begin(9600);
}
void loop() {
  analog_value0 = analogRead(0);
  analog_value1 = analogRead(1);
  Serial.print( analog_value0 );
  Serial.print("  ");
  Serial.println( analog_value1 );
  delay(200);
}
```

Open the Serial monitor and see the results and how they change with the thumbwheels on the gadget shield.

Analog inputs 2, 3 and 4 are the accelerometer Z, Y and X axis respectively. Input 5 is the light sensor.

Exercise 10.5: Using PWM as an analog output

In a similar way to digitalWrite(), Arduino can also write analog values to some of it's pins. It's not actually an analog signal, but a Pulse Width Modulated (PWM) signal that can emulate pretty well an analog signal. On this exercise we are going to control the dim of a LED through one of the analog inputs. Note that the Arduino UNO or Duemilanove only supports PWM on pins 3, 5, 6, 9, 10 and 11, therefore, we should use the LED on pin 11 of the gadget shield.

Use the following function:

```
analogWrite( pin, value); // value ranges from 0 to 255.
```

Note that analogRead() returns a range from 0 to 1023, while analogWrite only up to 255. In this case we have to map a range to another and fortunately Arduino have already a function for that:

```
value_mapped = map( value, 0, 1023, 0, 255 );
```

Now try creating a program that reads an analog input, maps it and writes to a LED.

Note: The 3 colors of the RGB LED on the gadget shield are on PWM pins, therefore you can create a whole gamma of colors by playing with the analogWrite function.

Exercise 10.6: Read an One Wire or I2C sensor

In addition to the sensors we have studied, there are also the digital sensors. These sensors, instead of having an analog output that we should convert to digital, they already provides the data in a digital form, following a certain protocol. The protocols varies from sensor to sensor, manufacturer to manufacturer. Some of the most popular ones are the I2C, SPI, OneWire.

In these cases, we only need to care about interpreting correctly the protocol. Each protocol have it's characteristics, pros and cons. Some uses multiple wires, some implements two-way communication.

In our case today we are going to read a temperature sensor that uses an One-Wire protocol, in which you notifies that you want a reading by sending an request signal and then waiting the sensor to respond. It uses the same via to send and receive the data, thus the name OneWire.

As the communication protocol itself is not the focus of our lab, we are going to use a library with already made functions to read such sensor. The tutor is going to assist you on this exercise. Ask him for the library and examples.

Exercise 10.7: Interrupt

Imagine a situation where we would like to keep reading our PIR Sensor from exercise 3 to sound an alarm as soon as it detects movement. But also imagine that for whatever reason our microcontroller is processing data or controlling other devices. How can we check the PIR sensor constantly to respond in time the alert if the processor is stuck with other tasks?

The solution is Interrupt. To explain how does it works, we are going to recreate the example above and then work on the solution.

Create a program that is constantly processing data (keeps flashing a LED at 0,5Hz) and also monitor an IR sensor (pin 2 from gadget shield). You can use the exercise 10 example labeled as "Wrong Way".

Try sound the alarm with this program, then program the Arduino using the example labeled as "Right Way" and compare the results.

Make a program yourself to generate an interrupt from the PIR Sensor while continually reading a temperature sensor.

CHEAT SHEET

Structure

```
void setup() {}
void loop() {}
```

Digital I/O

```
pinMode( pin# , [INPUT/OUTPUT] );
digitalWrite( pin# , [HIGH/LOW] );
var = digitalRead( pin# );      // returns boolean
```

Analog I/O

```
var = analogRead( pin# );       // returns integer from 0 to 1023.
analogWrite( pin# , value);     // PWM. Value = 0 to 255.
```

Time

```
Var = millis();                 // Milliseconds since boot. Returns unsigned long.
Var =  micros();                // Microseconds since boot. Returns unsigned long.
delay( ms );                    // ms = number of milliseconds to dely.
delayMicroseconds( us);
```

Serial

```
Serial.begin(9600);             // Initialize serial communication at 9600bps.
int Serial.available();         //Return number of bytes available for read.
Var = Serial.read();            // Read a byte from the receiver buffer.
Serial.Print(value);            // Prints a value to the serial port.
```

Exercise 11

Configure and evaluate a storage setup

Overview

The aim of this lab is to provide an overview about how to configure and evaluate a storage setup.

Objectives

- partition storage units
- setup raid systems
- performance measurements
- evaluate different raid strategies

Tools

dd

The linux `dd` tool allows you to make copies of files at a block level. Its basic syntax is :

```
dd if=/path/to/input/file of=/path/to/output/file bs=X count=Y
```

Where `X` is the block size of the individual transfers, and `Y` the amount of blocks you want to copy. We recommend 32M as X value.

The `seek` option allows you to skip a certain amount of blocks at the start of the output.

The `skip` option allows you to skip a certain amount of blocks at the start of the input.

The `oflag` is used to set particular flags that are used on the output stream. In our case the 'direct' option is useful. It forces the operating system to not use the write behind cache on the stream.

fdisk and sfdisk

Be very very careful. These tools can very easily wipe out the entire operating system if used on the wrong disk. Make sure you are only working on `/dev/isotdaq/XXX`

The `fdisk` tool is used to manipulate the partition table of a disk. The tool has an interactive shell and the important commands for the following exercises are:

```
n: create a new partition
d: erase a partition
w: write the new partition table to disk
p: show the current partition layout
h: help
```

The `sfdisk` tool allows you to dump the partition table of a disk into a file using the `–d` option, and then to apply the same schema to another disk using the redirect operator (<).

For example:

```
sfdisk –d /dev/isotdaq/disk1 > file   //to dump the partition table into file
sfdisk /dev/isotdaq/disk2 < file      //to read a partition table from a file
```

mdadm

The `mdadm` tool is used to manipulate the Linux software raid devices.

Its main running modes are 'Create' and 'Manage'. In order to create a new raid, the 'Create' mode is obviously to be used. You need to provide it with information about the raid level you want to create, and on which device it will reside.

Create a raid array:

```
mdadm --create=md<x>  --level=<x> --raid-devices=<N> <device1> <device2> … <deviceN>
```

Stop a raid array:

```
mdadm --misc --stop /dev/md/md<x>
```

fio

`Fio` is an advanced tool for characterising IO devices. It can be used to simulate different IO loads and profiles and evaluate disk performances. In our case we will use it to measure iops at a fixed block size. The following syntax will be enough for all of the exercises:

```
fio --rw=<opt1> --bs=<opt2> --runtime=<opt3> --filename=<opt4> --direct=1 --ioengine=libaio
--name=isotdaq

opt1: randread or randwrite
opt2: 4096
opt3: 60
opt4: /dev/isotdaq/disk<X> or /dev/md/md<Y>
```

Exercise 11.1: Determine the raw throughput of a single disk

For this exercise use dd to measure the throughput of one of the hard disks in the machine for read and write performance.

For write performance use `/dev/zero` as input file and `/dev/isotdaq/disk<N>` as output.

For read performance use `/dev/isotdaq/disk<N>` as input file and /dev/null as output.

Use the `seek` and `skip` options of dd to measure the performance at the end of the disk too.

Use the `count` option to write/read only 1GB of data.

Use `oflag=direct` during writing.

Hint: If you use an I/O block size of 32M the end of the disk should be around 7000

What is the read/write throughput of the disk in MB/s?

The `oflag=direct` ***option circumvents the operating system cache for the disk. Why is this important for this measurement?***

Which disk corresponds to which physical disk inside the enclosure?

Optional: Usually, for disk based storage, the write throughput is the same as the read throughput. Do you have any idea why it is different in this case?

Exercise 11.2: Determine the IOPS of a single disk

For this exercise use the fio tool to measure the random read and write Input Outputs Per Second (IOPS) of a single disk.

Good values for the parameters are a run time of 60s and IO size of 4096. For reading use `--rw=randread`. For writing use `--rw=randwrite`.

What are the values for random reading and random writing for these disks?

Why are these important values?

Why are the reading and writing values different?

Using the result from Ex.11.1: Calculate the IOPS of the throughput measurement. Why are the values for random IO so much smaller?

Exercise 11.3: Partitioning the disks

For the purpose of this exercise, we will create 4 partitions on each disk. They will later be used to host different raid types.

Create 4 partitions on `/dev/isotdaq/disk1` using `fdisk`. The partitions should be of type 'primary', and 2 Gb each.

After this you can either use `fdisk` to create the same partitions on the other 3 disks or use `sfdisk` to dump the layout of the first disk and import it to the other three disks.

Make sure that `/dev/isotdaq/disk<0-3>part<1-4>` ***exist***

Exercise 11.4: Creating the raid arrays

You will now create 4 different kinds of raid sets to measure their different properties. Use `mdadm` to create the following raids:

```
Raid0 on disk1part1, disk2part1, disk3part1, disk4part1
```

```
Raid1 on disk1part2, disk2part2
Raid5 on disk1part3, disk2part3, disk3part3, disk4part3
Raid6 on disk1part4, disk2part4, disk3part4, disk4part4
```

Hint: To create a raid set of a particular kind use

```
mdadm --create md<x>  --level=<x> --raid-devices=<N> <device1> <device2> … <deviceN>
```

Raid levels are 0, 1, 5 and 6. For easier recognition you can use the same number <x> for the raid level and the device name You can create and initialize multiple arrays in parallel.

Use `cat /proc/mdstat` ***to follow the initialisation of the raid arrays.***

Why do raid1, raid5 and raid6 need initialisation and raid0 does not?

Explain the different sizes of the finished raid sets.

Exercise 11.5: Performance Measurements

In this exercise you are going to explore the different performance values of the different raid types. Use `dd` and `fio` like in exercise 11.1 and 11.2 to determine the throughput and IOPS of the four raid sets you created earlier. For the throughput you can skip the measurement for the end of the device (Why?).

Remember to use `/dev/md/md<x>` for your measurements and not `/dev/isotdaq/`...

What values did you expect for the different raid sets?

Is what you got coherent with what you expected?

Which raid would you use for a data acquisition system?

What would you use for a normal file system/database?

Exercise 11.6: Failures

Remove disk1 and check if you can still access all your raid sets. Repeat the performance measurements for the raid sets that still work.

Explain why the performance of all raid sets has deteriorated.

Would you still choose the same raid type for your data as in exercise 11.5?

Why does raid 6 exist?

Exercise 12

DAQ Online Software

Outline

The aim of this exercise is to develop a control system for a DAQ system, which acquires health monitoring data (CPU, Memory usage, etc.) from a set of hosts. The system is composed of different independent sensors that simulate our data acquisition applications. The control system steers the behavior of those applications, individually and, in the final configuration, through a central controller.

Students will develop the system relying on the control and configuration capabilities provided by a simplified version of a real DAQ system.

Students 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 framework in a DAQ system.

Sensors

The Sensors are the data acquisition applications of our simple DAQ system. The main goal of a Sensor is to read and publish machine health metrics (CPU, Memory usage, etc.). Sensors can differ in what metrics they read and how they are implemented. Since we want a uniform way to control potentially different sensors, we define a simple Finite State Machine (FSM):

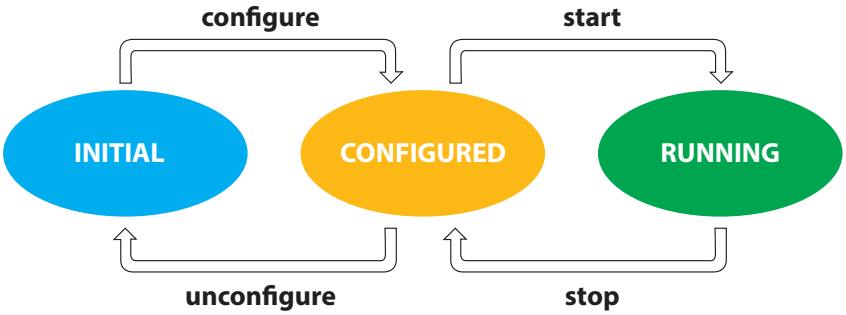
All the applications in the system must behave according to this FSM!

The FSM defines two main concepts:

1. Transition commands

   `configure, unconfigure, start, stop`
2. States

   `INITIAL, CONFIGURED, RUNNING`



Sensors are publishing the following data:

1. **hostname:** Indicates the name of the machine the sensor is running on.

   `Published in every state.`
2. **runtime:** Increasing counter measuring the time (in seconds) since last start.

   `Published when the application is in the RUNNING state, this value is updated every second.`
3. **type:** indicates the type of the published information (e.g. "CPUUsage" or "MemoryUsage").

   `Published when the application is in the CONFIGURED or RUNNING states.`
4. **value:** application-provided data (e.g. idle CPU percentage or free memory).

   `Published when the application is in the RUNNING state.`

All applications publish their FSM state (i.e. INITIAL, CONFIGURED, RUNNING). **Additionally, there is an ERROR state if the application raises an Exception, which signals that the application has a problem that prevents it from accepting FSM commands.**

2. DAQ Python framework

For this exercise a simplified version of the DAQ system was developed in the Python programming language. This system consists of a master server and applications publishing information to it. The applications need to implement a common interface to expose a REST API to the master server.

A web interface running on the master server is also provided, showing an overview of all applications, information published

and allows sending commands to each application. The master server also exposes a REST API, allowing each of the application to retrieve data from and send commands to other applications.

2.1 HTTP REST API

All communication between the applications and master server is done over HTTP. Both, the master server and the application run a web server and expose GET/POST endpoints that can be called. For applications, the communication is hidden behind the *DAQInterface* and is not exposed to users. They only need to implement this interface.

When an application is started the following HTTP requests happen in the background:

1. The application subscribes itself to the master server calling http://master.server/subscribe and submits the parameters hostname, port and name. Now the master server knows how to make requests to this application.
2. Before the website http://master.server/ is visited a request is sent to all subscribed applications on http://hostname:port/hostname, http://hostname:port/value, http://hostname:port/state, …
3. The HTTP response of each of this calls is a string and is displayed on the website.
4. A state transition on the application can be triggered by calling http://hostname:port/<configure>|<start>|<stop>|<unconfigure>

A controller contains names of all sub-applications and can make calls to the master server specifying the sub-application and state transition like http://master.server:port/name/<configure>|<start>|<stop>|<unconfigure>

and the master server will execute a state transition on the application with this name.

All this HTTP requests are hidden behind method calls on Python classes implementing the *DAQInterface*.

hello_monitor.py application

The hello_monitor.py application is the simplest possible monitoring application. It always publishes the value "Hello World" of type "Greeting".

All applications can be run as a regular python application, e.g.:

```
$ python hello_monitor.py <name>

# Application started with ID: 25818

# Visit http://localhost:36500 for an overview of all applications.
```

To uniquely identify each application a <name> needs to be assigned to the application (e.g. `python hello_monitor.py hello_monitor`).

The state and data published can be viewed by visit the mentioned URL (Fig. 1).

Applications

Name	Hostname	State	Runtime	Type	Value
cpu_monitor	pb-d-12801414-154-213-cern.ch	INITIAL configure	0 s	—	—

Figure 1: Web view

It is possible to trigger state changes from the web interface.

The *HelloMonitor* class inside *hello_world.py* implements the *DAQInterface*. Having all applications implement the same interface unifies the way we control them.

The DAQInterface expects you to provide the following methods:

5. `__init__()`: Is called on startup and allows for parsing of passed arguments.
6. `configure()`, unconfigure(), start(), stop(): This code is run if a state change is triggered on the application. If it does not raise an Exception the state change is executed.
7. `value()`: Returns the data to be published.

Exercise 12.0: Change the value returned by the hello_monitor.py application

Change the code in `hello_monitor.py` to return "Hello ISOTDAQ!".

Note: If there are no syntax errors the application will be automatically restarted after saving the file inside the editor.

Exercise 12.1: Fix the CPU sensor according to the sensor rules

Compare the behavior of the CPU sensor with the Memory sensor, and make sure that it complies with the finite state machine rules defined in the Outline.

Start both sensors in different terminals and send commands to them to spot the different behavior in the web interface.

```
$ python cpu_monitor.py <name>
```

```
$ python memory_monitor.py <name>
```

Find the bugs and happy fixing!

3. Controller

The second part of this exercise focuses on the role of the Controller application. The distributed monitoring system 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 an entity to manage the control flow. This is the role of the Controller, an application that receives FSM commands and forwards them to a set of children applications. The Controller application must also check the proper execution of the FSM transitions, deal with common problems, etc.

The Controller implements the same interface as every other application and is able to receive commands.

Exercise 12.2: Start a Controller application

Start the CPU monitor:

```
$ python cpu_monitor.py cpu_monitor
```

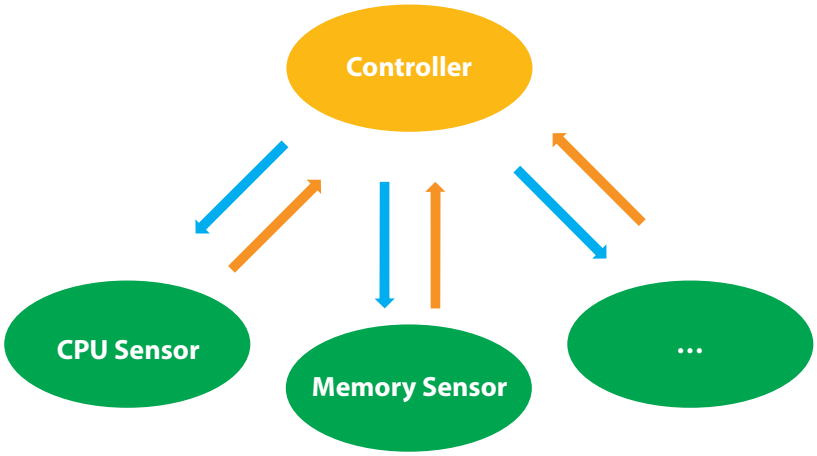
Start the memory monitor:

```
$ python memory_monitor.py memory_monitor
```

Start the controller by passing the names of the applications you want to attach to:

```
$ python controller.py <name> cpu_monitor memory_monitor
```

Now use the web interface to send commands to the controller and observe the propagation of commands and state changes.



Exercise 12.3: Improve controller to handle a faulty application

The faulty sensor is an application simulating a failure in the processing of a command. This can happen for many different reasons in the real world. Students have to improve the Controller code in order to handle this type of failure.

Start the faulty application:

```
$ python faulty_memory_monitor.py faulty_memory_monitor
```

Attach it to a controller:

```
$ python controller.py <name> faulty_memory_monitor
```

When attempting a `configure` state change the `faulty_memory_monitor` may sometimes return "ERROR".

Improve the controller to manage the faulty application. Think about multiple ways to handle errors.

Exercise 12.4: Enhance the Controller's flexibility

The controller implements a strict FSM logic. If one controlled application changes state, the controller goes into the error state and will not clear the error even if all children applications go back into the same, consistent state.

Change the controller logic to clear the error state if all child applications are in a coherent state, corresponding to the state of the controller. To do this, the best way is to add a FSM (states ERROR, OK) in addition to the standard FSM (INITIAL, CONFIGURED, RUNNING).

Exercise 12.5: Add a root Controller and send commands to all sensors

A controller can be attached to another one, they implement the same interface as applications.

Run a root controller controlling all other controllers. Both groups work together on this exercise.

Exercise 13

Programming FPGAs with LabVIEW

Introduction to Field-programmable Gate Arrays

Field-programmable gate arrays (FPGAs) are reprogrammable silicon chips. FPGA chip adoption across all industries is driven by the fact that FPGAs combine the best parts of application-specific integrated circuits (ASICs) and processor-based systems. FPGAs provide hardware-timed speed and reliability, but they do not require high volumes to justify the large upfront expense of custom ASIC design.

Reprogrammable silicon also has the same flexibility of software running on a processor-based system, but it is not limited by the number of processing cores available. Unlike processors, FPGAs are truly parallel in nature, so different processing operations do not have to compete for the same resources. Each independent processing task is assigned to a dedicated section of the chip, and can function autonomously without any influence from other logic blocks. As a result, the performance of one part of the application is not affected when you add more processing.

Every FPGA chip is made up of a finite number of predefined resources with programmable interconnects to implement a reconfigurable digital circuit and I/O blocks to allow the circuit to access the outside world. There could be literally thousands to millions of these components inside a single chip.

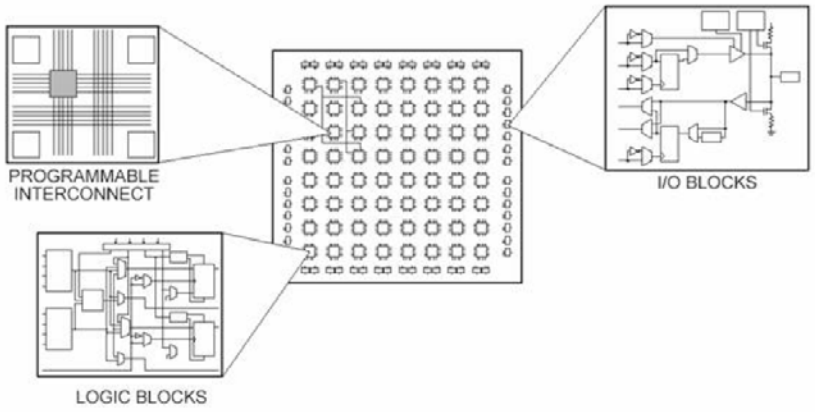


Figure 1. The Different Parts of an FPGA

FPGA resource specifications include the number of configurable logic blocks, number of fixed function logic blocks such as multipliers, and size of memory resources like embedded block RAM (Figure 1.). Of the many FPGA chip parts, these are typically the most important when selecting and comparing FPGAs for a particular application. Over the last decade a hybrid architecture, sometimes called a heterogeneous architecture, has emerged in which a microprocessor is paired with an FPGA that is then connected to I/O (Figure 2.). This approach takes advantage of the benefits that both these targets offer.

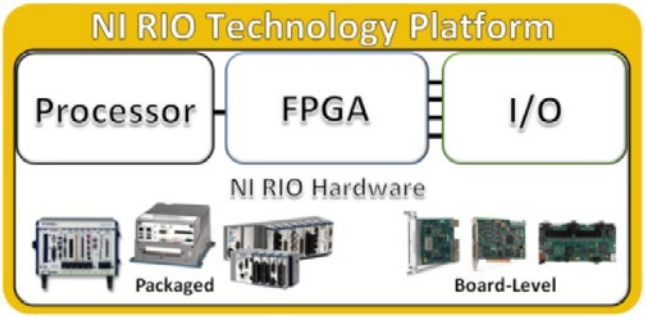


Figure 2. Processor – FPGA hybrid system architecture

National Instruments (NI) offers an entire family of Reconfigurable Input/Output (RIO) devices based on this ideal hybrid architecture coupling both a microprocessor and an FPGA, all that you can program in the NI LabVIEW graphical system design environment.

How to design?

Now that you know the building blocks of an FPGA chip, you may ask, "How do you configure all of these millions of components to build up the logic that you need to execute?"

The answer is that you define digital computing tasks in software using development tools and then compile them to a configuration file or bitstream that contains information on how the components should be configured and wired together. The challenge in the past with FPGA technology was that the low-level FPGA design tools could be used only by engineers with a deep understanding of digital hardware design. However, the rise of high-level synthesis (HLS) design tools, such a Lab-

VIEW FPGA changes the rules of FPGA programming and delivers new technologies that convert graphical block diagrams into digital hardware circuitry. The LabVIEW programming environment is distinctly suited for FPGA programming because it clearly represents parallelism and data flow, so users who are both experienced and inexperienced in traditional FPGA design processes can leverage FPGA technology. In addition, so that previous intellectual property (IP) is not lost, you can integrate existing VHDL code into your LabVIEW FPGA designs.

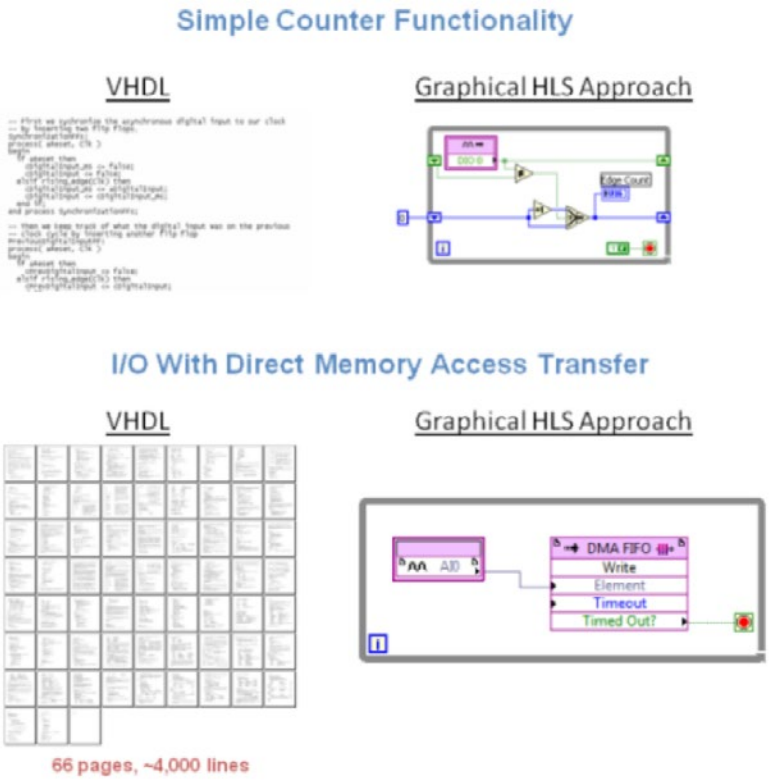


Figure 3. On the right is LabVIEW system design software, which is a high-level design tool for FPGAs in NI RIO hardware devices. It provides abstraction for the low-level complexity often found when creating and scaling VHDL designs.

To simulate and verify the behavior of your FPGA logic, LabVIEW offers features directly in the development environment. Without knowledge of the low-level HDL language, you can create test benches to exercise the logic of your design. In addition, the flexibility of LabVIEW allows more advanced users model the timing and logic of their designs by exporting to cycle-accurate logic simulators such as Xilinx ISim or Mentor Graphics ModelSim.

For this exercise NI provides myRIOs and host computers with the complete hardware-software setup required, including the LabVIEW design software.

What Is myRIO?

The myRIO embedded student design device was created for students to "do real-world engineering" in one semester. It features a 667 MHz dual-core ARM Cortex-A9 programmable processor and a customizable Xilinx FPGA that students can use to start developing systems and solve complicated design problems faster—all in a compact form factor. The myRIO device features the Zynq-7010 All Programmable system on a chip (SoC) to use the power of LabVIEW system design software both in a real-time (RT) application and on the FPGA level.

myRIO is a reconfigurable and reusable teaching tool that helps students learn a wide variety of engineering concepts as well as complete design projects. Using the real-time, FPGA and built-in WiFi capabilities along with onboard memory, students can deploy applications remotely and run them "headlessly" (without a remote computer connection). Three connectors (two myRIO expansion ports [MXP] and one miniSystems port [MSP] that is identical to the myDAQ connector) send and receive signals from sensors and circuitry that students need in their systems. Forty digital I/O lines overall with support for SPI, PWM out, quadrature encoder input, UART and I2C; eight single-ended analog inputs; two differential analog inputs; four single-ended analog outputs; and two ground-referenced analog outputs allow for connectivity to countless sensors and devices and programmatic control of systems. All of this functionality is built in and preconfigured in

the default FPGA functionality. The myRIO also has an accelerometer, an Audio input and an audio output, which will be used during this exercise.

Steps of the course Project

The following section guides you through an exercise in which a simple data acquisition system is implemented. The FPGA is programmed to receive sample data from the audio input, filter the audio, send the result to the audio input and forward the raw and filtered data to the host computer. The host is responsible for receiving the data from the FPGA and displaying the results.

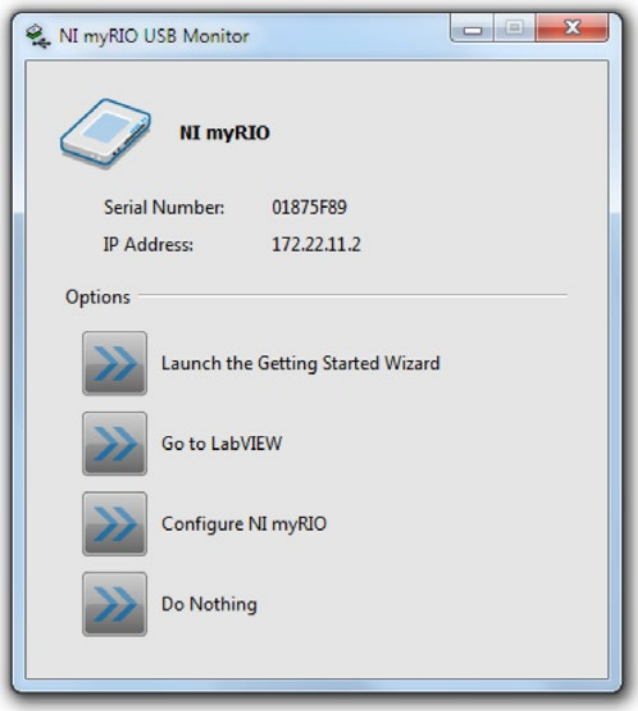
This example does not contain anything which could not be done with a host processor without an FPGA. The point is to show the complexity of the LabVIEW FPGA design process. It is up to the attendee to improve the project with a filtering function and accelerometer control in the FPGA code.

To get the most out of this exercise, make sure you ask questions to your instructor, who is a National Instruments professional. He or she will help you to do the LabVIEW program and also will explain the reason for each step taken in detail.

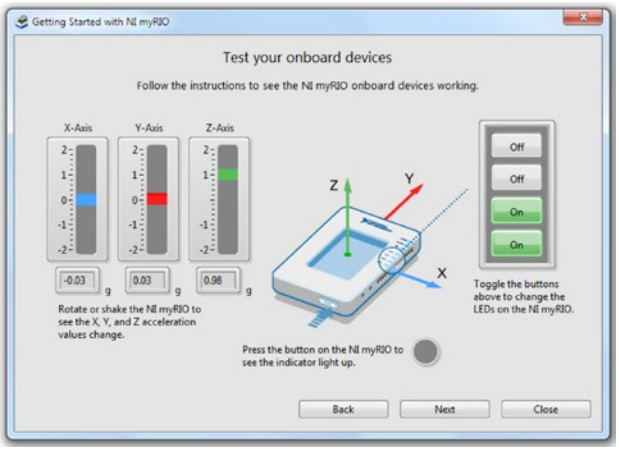
Connecting and Configuration the myRIO

Make sure you power the myRIO device using the power adapter that came with it. Plug the USB Type B end of the USB cable into the myRIO device. Connect the other end of the cable to your computer's USB port.

Without starting LabVIEW or NI MAX, if the device is powered, the OS should recognise the myRIO device and install and set up the drivers for it. Once this is complete, in the Windows OS, Windows should automatically launch the myRIO USB Monitor shown below.



Along with the serial number and IP address, you have four options to choose from when an myRIO device is detected: From the myRIO USB Monitor, select Launch the `Getting Started Wizard`. Select `Next` and the wizard connects to the myRIO unit, checks for software on it and prompts you to rename the device. If the myRIO unit does not have software installed on it, the wizard automatically installs the most up-to-date software. After that, a diagnostic window opens. With it, you can observe the values of the built-in three-axis accelerometer, test the functionality of the user-defined push button and toggle the four onboard LEDs.



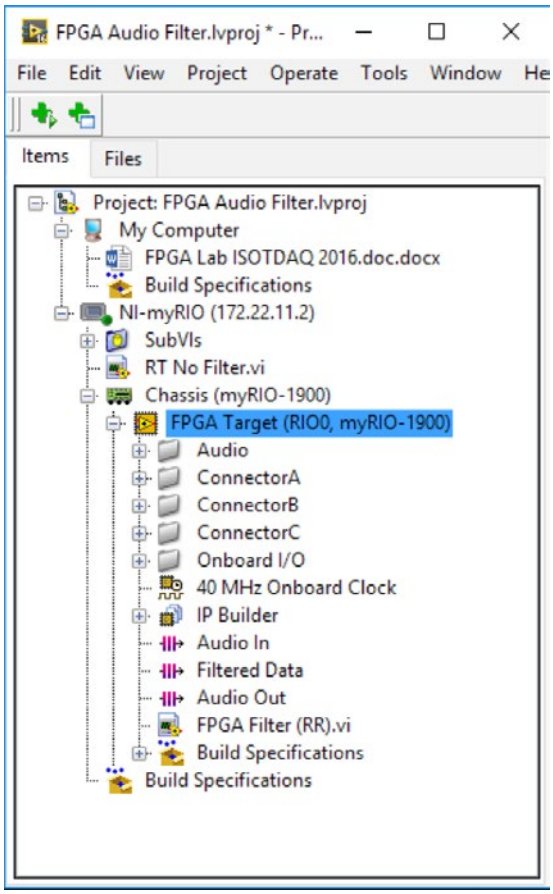
Ensure that you have an audio source (laptop, microphone, mp3 player, smart phone) connected to the myRIO's `audio input`. Also ensure that you have connected a loud speaker to the myRIO's `audio output`.

Creating the LabVIEW Project and Programming the FPGA

On the desktop open the folder called "`FPGA Demo ISOTDAQ`"

Double-click on the project "`FPGA Audio Filter`" to open the LabVIEW project

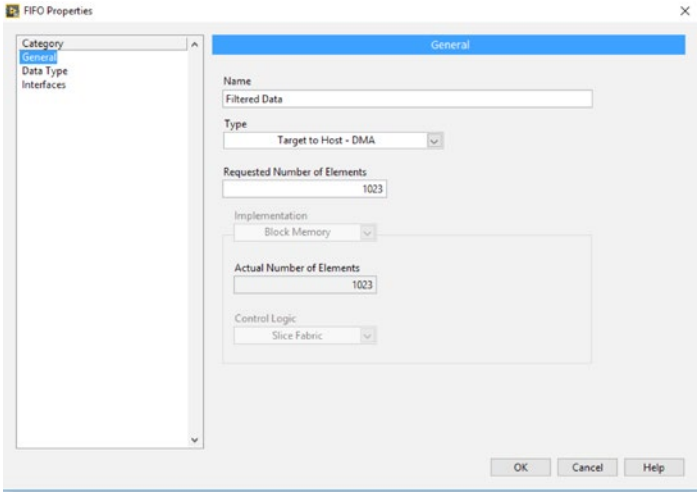
Expand the `myRIO` target and notice the new "chassis" tree. This tree appears on devices that contain a targetable FPGA. Logically, the FPGA target falls under the chassis tree because the FPGA is integrated within the myRIO chassis.



The FPGA's primary role is handling all of the I/O on the myRIO device, so you can find the I/O under this FPGA target as project-unique items. The hierarchy of the FPGA target is organized into folders so the user can tell where each I/O node is located on the physical device. The two MXP connectors, the one MSP connector and the onboard I/O all have unique folders. The folders are further subdivided into the I/O type (digital/analog) and the physical banks of I/O. You can drop these unique I/O items into an FPGA VI to read or write to that location. Any control or indicator on the front panel of the FPGA VI can be written to or read from, respectively, in the RT VI.

Back in the project window right click again on FPGA Target then select `New/FIFO`

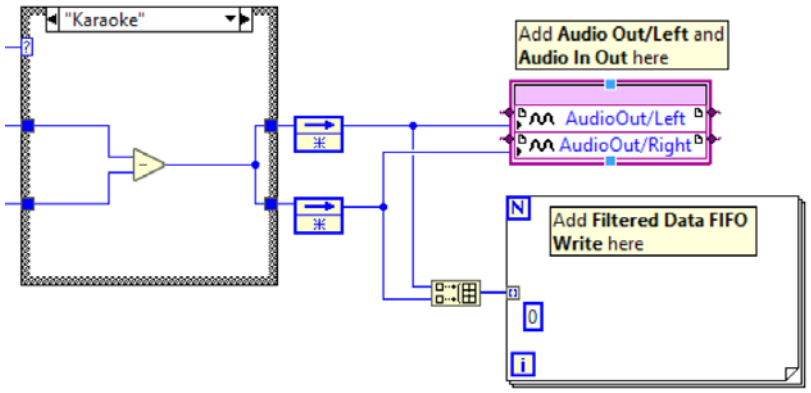
Enter "`Filtered Date`" as Name, select the `Type` to `Target to Host – DMA`, on the `Data Type` tab set type to I16, and finally click `OK`.



Open the VI "FPGA Filter (RR).vi" which you can find under the FPGA Target in the project and open its Block Diagram by clicking ctrl+E in the Front Panel window.

Back in the Project Explorer expand the FPGA Target >> Audio and drag AudioOut/Left to the Block Diagram where the it is instructed to add it in the comments.

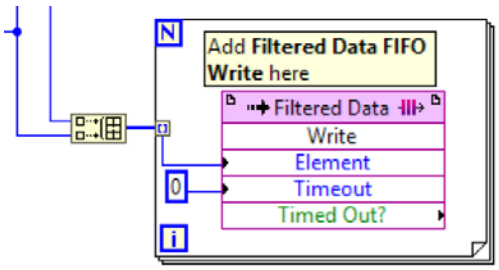
You can observe that the corresponding FPGA IO node appeared on the Block Diagram. Expand the FPGA IO node to add another output. Make sure the second output is AudioOut/right, if this is not the case, click on the output and choose Audio >> AudioOut/right. Now you should have the same FPGA IO node as the following screenshot. Complete the connections accordingly.



From the Project Explorer drag the Filtered Data to the Block Diagram.

You can observe that the DMA channel node appeared on the Block Diagram. Place it inside the for loop where it is indicated with a comment.

Connect the DMA Channel node according to the following screenshot. The 0 constant must be wired to the Timeout terminal of the DMA node.



The FIFO Data will be read in the Host VI so that the filtered data can be displayed. Go back to the project and open the RT No Filter.vi. Open the Block Diagram and double click on the subVI FPGA Filter (Read Filtered Audio).vi. Open the Block Diagram to see how the data from the Filtered Data FIFO are read.

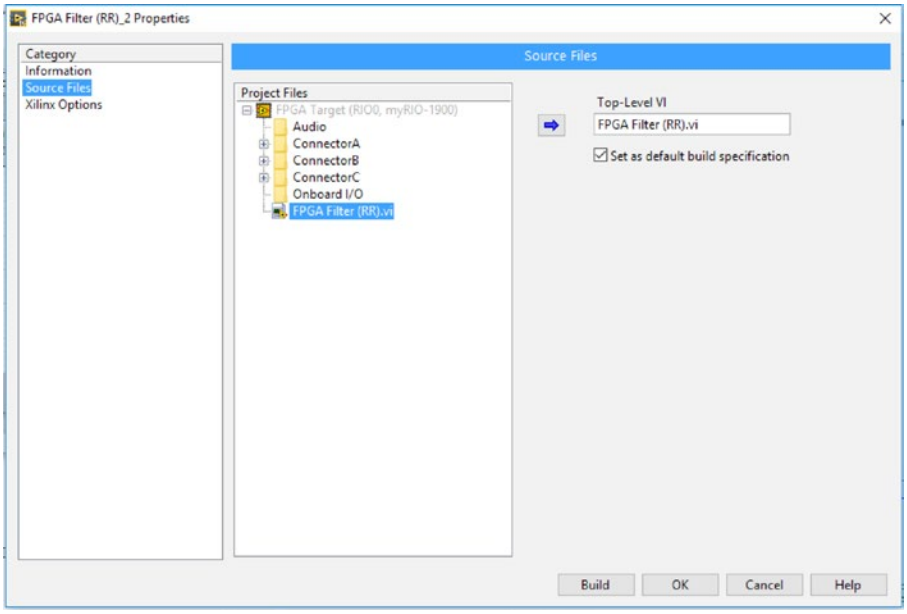
The Challenge

Using the knowledge you acquired, change the FPGA Filter RR.vi so that:

- You read the signal form the accelerometer of myRIO in 3 axis
- Depending on the position of myRIO, the audio data will be filtered. For example, if the myRIO is tilted more than 30° in the y-axis, only Bass tones are filtered.
- Be creative

Compiling the FPGA code and configuring the Host code

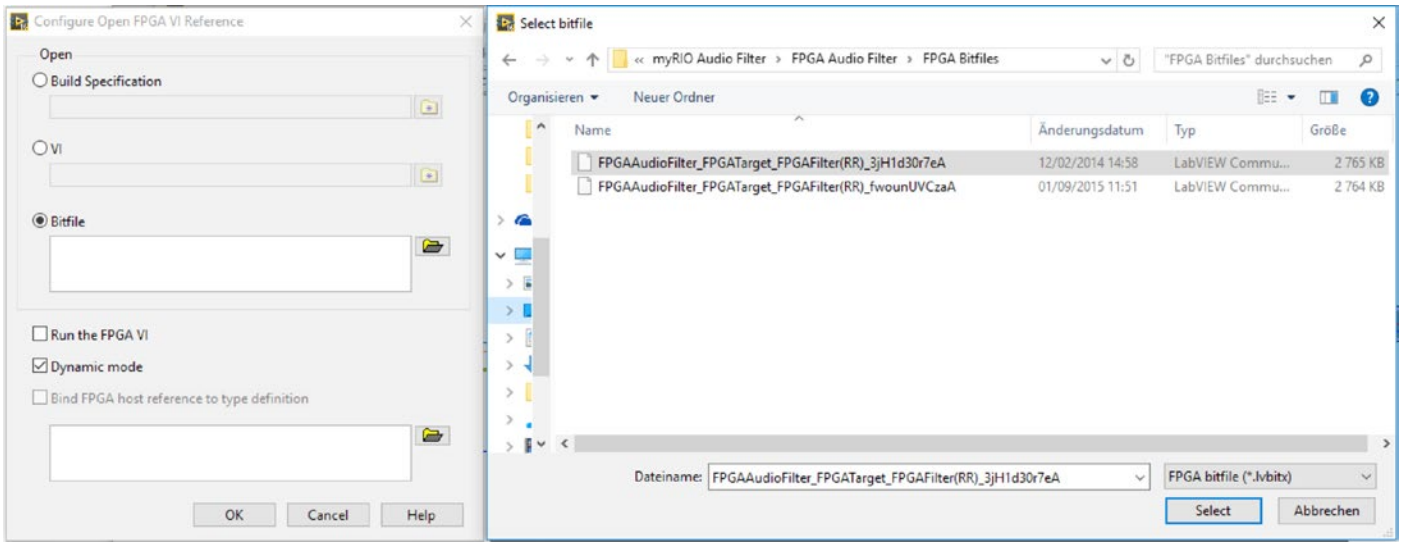
In order to be able to run the code on the FPGA it needs to be compiled. In the Project Explorer under the FPGA Target right click Build Specifications then select `New/Compilation`. On the `Source Files` page select the `FPGA Filter RR.vi` and on the Information page enable "`Run when loaded to FPGA`". Click `Build` and the FPGA code will be compiled and the bitfile made available for future use.



Now open the Host application `RT No Filter.vi`

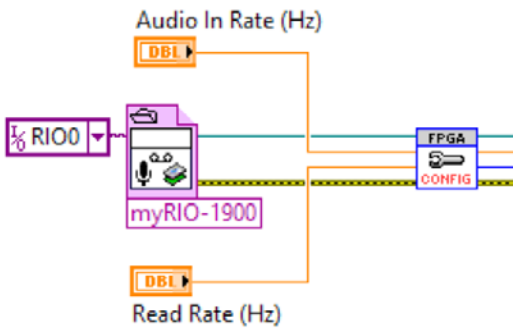
Open the Block Diagram and right-click to open the function palette. Under `FPGA Interface` choose `Open FPGA VI Reference` and place it before the SubVI `FPGA Filter (config).vi`.

Double click to configure the `Open FPGA VI Reference`. Select `Bitfile` and the Bitfile we've just created, click OK.



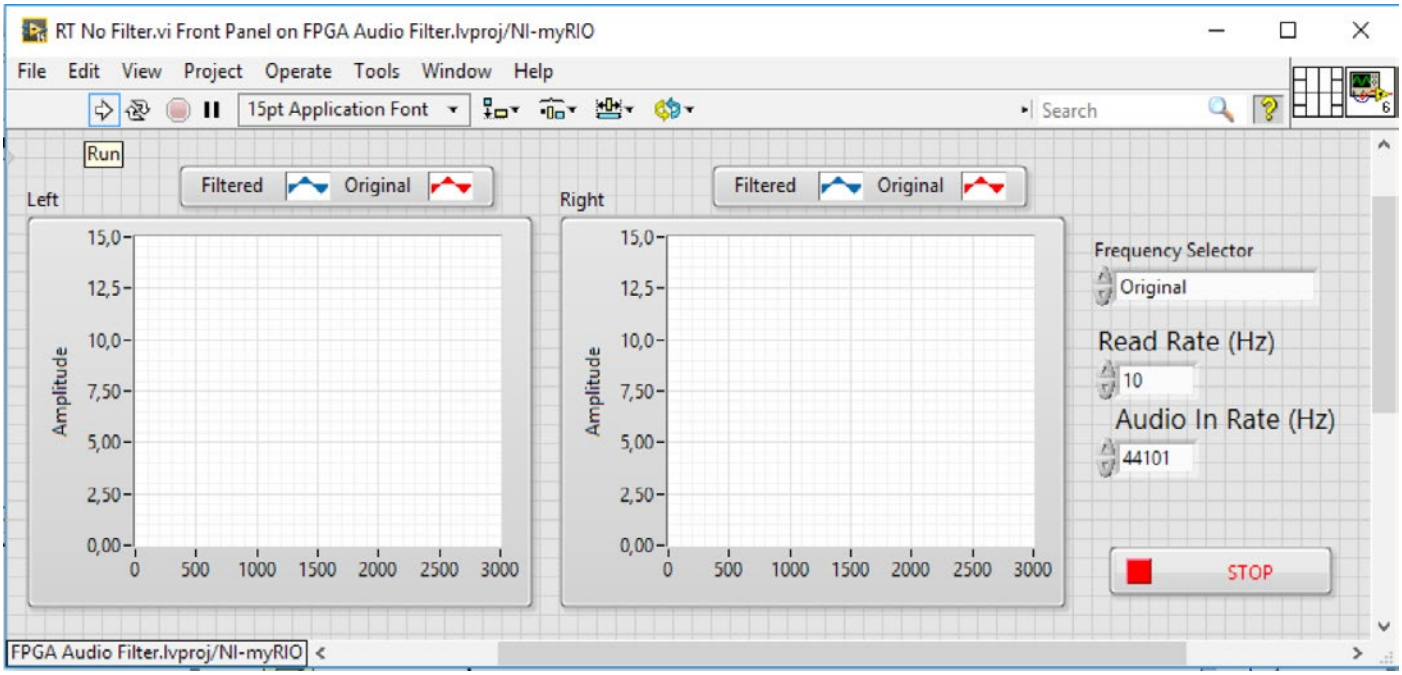
Right click the Resource name input terminal of the vi on the Block Diagram and select `Create >> Constant`. In the drop-down menu of the constant select the FPGA board we are using (`FPGA Target – RIO`).

Connect the output of the Open FPGA VI Refrence to the FPGA Filter (config).vi, as show in the following screenshot



If you did the challenge and configured the filtering depending on the accelerometer signals, change the Host VI accordingly, so that you replace the `Frequency Selector` with the appropriate code.

If you did everything correctly, you should be able to run the application now by clicking on the run icon of the Host VI.



For training documents and videos on this topic visit the following page on our website.
http://www.ni.com/academic/students/learn-rio/

Notes

Notes

Notes


< Centrum
Oosterringdijk (fietspad)
Diemen/IJburg >
ARCNL 110
AMOLF 104
Nikhef 105
CWI 123
125
SURFsara NLeSC 140
Ignis
Science Park
Aqua
Carolina Mac Gillavrylaan
P3
AUC 113
Science Park A'dam
UvA
Aer
205
UvA 904
Kruislaan
Terra
P7
NS Amsterdam Science Park
Station Science Park
tunnel 3,35 m