Overview

IC/DAQ + Electronics project intro

Hardware communication overview

PyVisa

SAGI 2024: IC/DAQ + Electronics Project Intro, Communicating w/ Hardware in Python

Sam Condon and Logan Foote

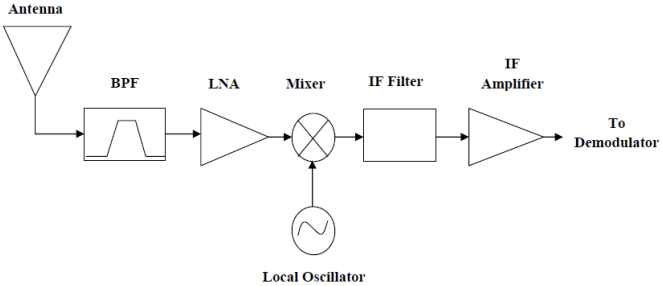
July 29th, 2024

Overview

1. IC/DAQ + Electronics project intro.
2. Hardware communication overview.
3. PyVisa.
4. Connecting to an oscilloscope and function generator demonstration.

IC/DAQ + Electronics Project Intro

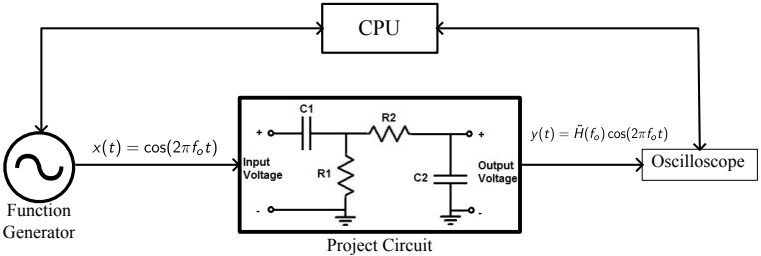
1. Recall the Heterodyne receiver concept from the Radio Astronomy lectures:



2. Essential elements of all radio receivers are high quality filters and mixers.
3. In the IC/DAQ + electronics project we will build a bandpass filter then use Python, a function generator, and an oscilloscope, to run automated frequency response measurements.

IC/DAQ + Electronics Project Intro

1. **Measurement schematic:**



2. The bandpass filter we will construct is a **linear time-invariant system**. Thus, sinusoidal signals are **eigenfunctions**.

$$x(t) = \cos(2\pi f_o t) \rightarrow y(t) = \tilde{H}(f_o)\cos(2\pi f_o t) \tag{1}$$

3. We will use Python to control the function generator and record the response of the circuit measured by the oscilloscope for hundreds (thousands) of individual frequencies.

IC/DAQ + Electronics Project Intro

1. **Siglent SDG2082x Arbitrary Waveform Generator:**

2. Dual channel, 120 MHz bandwidth, 20 Vpp.
3. LAN (ethernet) interface for communication.
4. Provides several built-in waveforms and programmable arbitrary waveforms.

IC/DAQ + Electronics Project Intro

1. **KKMoon FY6600 Arbitrary Waveform Generator:**

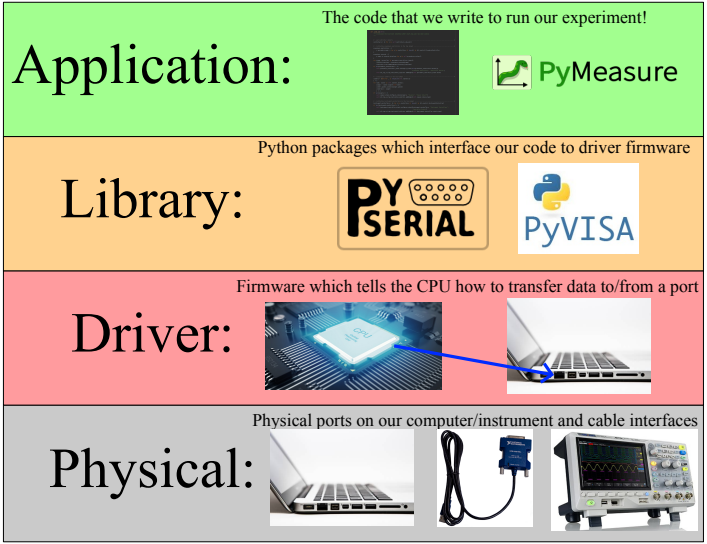
2. Smaller bandwidth, but plenty sufficient for this project.
3. USB port interface for serial communication.
4. Also provides several built-in and programmable waveform options.

IC/DAQ + Electronics Project Intro

1. **Siglent SDS1104X-E Oscilloscope**

2. 4-channels, two with 1 GHz ADCs
3. Can sample and store up to 14 Mpts
4. LAN (ethernet) interface for communication.

Hardware Communication "Stack"



Hardware Communication "Stack"

Physical and Driver levels:

1. Together, the physical and driver levels of the stack comprise what we call a "hardware communication protocol."
2. There are several types of hardware communication protocols including: USB, Rs232, Rs485, GPIB, and TCPIP.
3. We will cover Rs232/Rs485, GPIB, and TCPIP in this lesson.

Hardware Communication Protocols

Rs232 and Rs485:

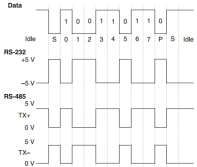


Figure: **Left:** Typical serial interface cable to transmit Rs232 or Rs485 signals. **Center:** Voltage signals for Rs232 and Rs485. **Right:** Rs232 port on the back of an old computer.

1. For "serial" transmission of data between a computer port and an instrument.
2. Good for short range, point-to-point, low-speed, wired communication.
3. Key terminology to understand: 1) "baud rate" 2) "data bits" 3) "parity" 4) "stop bits".

Hardware Communication Protocols

Rs232 and Rs485:

1. **Baud Rate:** Bits per second on the transmission line.
2. **Data Bits:** Number of bits between a start bit and stop bit.
3. **Parity:** Error checking mechanism. Can be "even", "odd", or "none."
4. **Stop Bits:** Number of bits to transmit to indicate the end of a byte.

Hardware Communication Protocols

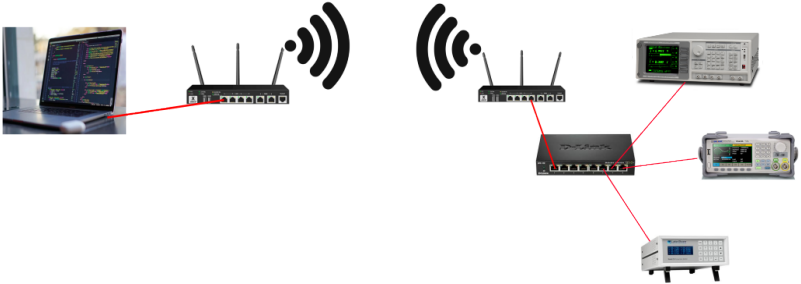
GPIB:

1. Short-range 8-bit parallel bus with bi-directional data transfer and addressing.
2. Also known as IEEE-488.
3. Allows us to make several high-speed parallel connections to multiple instruments with a single USB to GPIB adapter.

Hardware Communication Protocols

TCPIP:

1. Suite of communication protocols used to interconnect network devices on the internet.
2. Can use a network switch to connect multiple instruments to the same network to control our experiments remotely.



PyVisa

1. To communicate with hardware in python, we recommend starting with the PyVisa package.

PyVISA

2. PyVisa provides a unified framework for communication over several protocols including:
   1. Serial (Rs232/485)
   2. TCPIP
   3. USB
   4. GPIB
   5. PXI
   6. VXI

PyVisa

1. Connecting to hardware is very straightforward with PyVisa:

```
>>> import pyvisa
>>> rm = pyvisa.ResourceManager()
>>> rm.list_resources()
('ASRL1::INSTR', 'ASRL2::INSTR', 'GPIB0::12::INSTR')
>>> inst = rm.open_resource('GPIB0::12::INSTR')
>>> print(inst.query('*IDN?'))
```

2. Above, we import the package, open the **resource manager**, call the list_resources method to view the available hardware, connect to the GPIB instrument with address 12, then print out the instrument ID.

PyVisa

1. List of resource names from the official PyVisa documentation:

Interface	Syntax
ENET-Serial INSTR	ASRL[0]::host address::serial port::INSTR
GPIB INSTR	GPIB[board]::primary address[::secondary address][::INSTR]
GPIB INTFC	GPIB[board]::INTFC
PXI BACKPLANE	PXI[interface]::chassis number::BACKPLANE
PXI INSTR	PXI[bus]::device[::function][::INSTR]
PXI INSTR	PXI[interface]::bus-device[.function][::INSTR]
PXI INSTR	PXI[interface]::CHASSISchassis number::SLOTslot number[::FUNCfunction][::INSTR]
PXI MEMACC	PXI[interface]::MEMACC
Remote NI-VISA	visa://host address[:server port]/remote resource
Serial INSTR	ASRLboard[::INSTR]
TCPIP INSTR	TCPIP[board]::host address[::LAN device name][::INSTR]
TCPIP SOCKET	TCPIP[board]::host address::port::SOCKET
USB INSTR	USB[board]::manufacturer ID::model code::serial number[::USB interface number][::I
USB RAW	USB[board]::manufacturer ID::model code::serial number[::USB interface number]::R
VXI BACKPLANE	VXI[board][::VXI logical address]::BACKPLANE
VXI INSTR	VXI[board]::VXI logical address[::INSTR]
VXI MEMACC	VXI[board]::MEMACC
VXI SERVANT	VXI[board]::SERVANT

PyVisa

Basic summary of the steps for instrument communication via PyVisa:

1. Identify the hardware communication protocol (USB, GPIB, TCPIP).
2. Write the appropriate resource name, open the resource using rm.open_resource().
3. Find the instruments programming guide. Sometimes this is a standalone document, but often it is just found as part of the instrument's manual. You should be able to find the appropriate documentation with a simple google search, otherwise you can email the manufacturer.
4. Write some basic functions to utilize the command and query syntax of the instrument.
5. Use the **write()** and **query()** PyVisa methods to send commands and receive data from the instrument.