The pulse sequencer

Aka Pauls Box

Philipp Schindler

September 4, 2008

Outline

Outline

A few definitions

Datapoints, Cycles and Scans

- Cycle: An simple experimental cycle consisting of Preparation, manipulation, detection
- Datapoint: One datapoint consists of several repeated cycles (typically 50-100) with the same parameters
- Scan: One scan consists of several datapoints with a single varied parameters

A few definitions

Datapoints, Cycles and Scans

- Cycle: An simple experimental cycle consisting of Preparation, manipulation, detection
- Datapoint: One datapoint consists of several repeated cycles (typically 50-100) with the same parameters
- Scan: One scan consists of several datapoints with a single varied parameters

Synchronous and Asynchronous signals

- Synchronous: Deterministcally switched in one experiment cycle
- Asynchronous: Switched between two experiments (may be varied in a scan)

Functionality

The Box is responsible for all synchronous signals in the experiment.

Timing and program control control

- Minimum time step: 10ns
- Allows simple control flow techniques
  - infinite loops
  - finite loops
  - conditional jumps (do something if trigger is high)

Functionality

Digital outputs

- Synchronous exactly timed digital outputs
- Minimum time step: 10ns

Functionality

Digital outputs

- Synchronous exactly timed digital outputs
- Minimum time step: 10ns

Radio frequency outputs

- Frequency from 1 .. 300 MHz
- Switching time: several 100ns
- Pulse shaping possible
- Phase coherent switching
- Up to 16 RF outputs

Functionality

Digital outputs

- Synchronous exactly timed digital outputs
- Minimum time step: 10ns

Radio frequency outputs

- Frequency from 1 .. 300 MHz
- Switching time: several 100ns
- Pulse shaping possible
- Phase coherent switching
- Up to 16 RF outputs

Trigger inputs

- 8 digital trigger inputs for program flow control

Versions of the Box

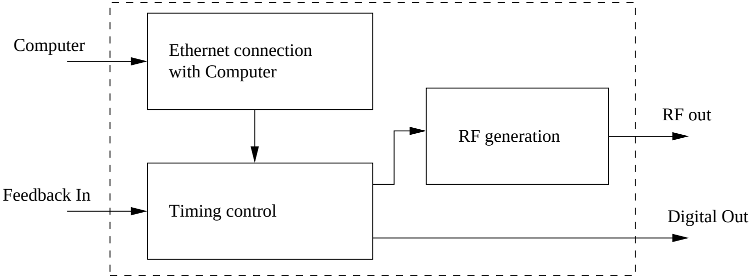
Hardware versions

- "Main Board" (Sequencer): Same version for all installs (Rev. C.)
- Breakout board, Synthesizer, Variable Gain amplifier:
  - New version (Autumn 2008)
  - Different versions (homebuild vs. Evaluation boards)

Software Versions

- sequencer and sequencer2
- Complete rewrite of the software for the new DDS boards (autumn 2008)
- Faster, cleaner code. Almost compatible with old software

Block Diagram



Programmable Pulse Generator

Communication with computer

Communication

- Communication with the control computer is realized over a standard network protocol.
- No additional drivers for the computer are necessary
- The Box needs a IP address within a small subnet (192.168.0.220..255)
- Communication is done over a specific protocol (pulse transfer protocol PTP)
- PTP core of the box saves program into memory

Program flow control

- A simple homebrew "microprocessor" is the heart of the box (Pulse control processor PCP).
- PCP Fetches instructions from memory and executes it.
- Possible instruction classes:
  - Pulse: Set the digital output to value X
  - jump: Program flow control (jump if trigger, ...)
  - wait: Halt processor for a certain amount of clock cycles
  - start/stop processor

Radio frequency pulse generation

- With the help of direct digital sysnthesizer
- Phase coherent switching
- Arbitrary pulse shapes possible
- More explanations later

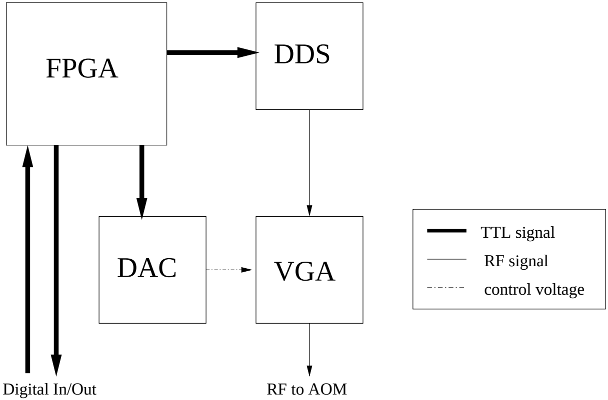
Digital output system

Digital output system

- 16 digital outputs freely available on
- 3.3 LVTTL standard
- Should be 5V TTL compatible

Outline

Block Diagram



What is an FPGA

Field Programmable Gate Array

- Reconfigurable Logic device
- Contains
  - Logical Units (LUTs)
  - Memory blocks (RAM)
  - Mulitpliers, PLL
- Device used in the sequencer has more than 12,000 LUTs
- Has only volatile memory. →Has to be reprogrammed at every power up.
- Dedicated non volatile memory for programming the device.

How to program an FPGA

Programming options

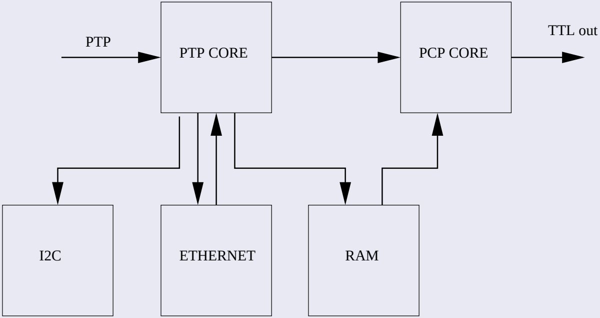
- Two Different Possibilities:
  - JTAG: Standard interface used for almost all microprocessors
  - Active Serial Programming: Only with the dedicated non volatile memory from the FPGA vendor.
- Astive serial programming is used for non volatikle programming
- Programming software can be downloaded from http://www.altera.com

Debugging the FPGA

- With the JTAG interface it is possible to debug the FPGA
- The synthesis / programmer software has the option to add a logic analyser to the FPGA
- This logic analyzer transfers data to the PC via the JTAG interface

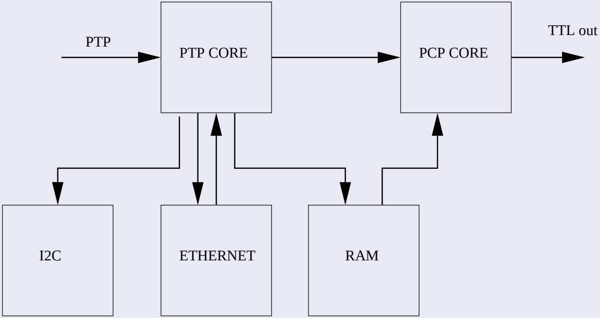
FPGA firmware

Firmware overview



FPGA firmware

Firmware overview

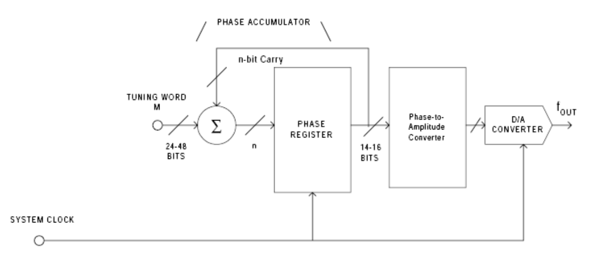


Firmware overview

- PTP Core: Handles data transfer
- RAM: On board memory fr sequence
- PCP Core: Timing and output control

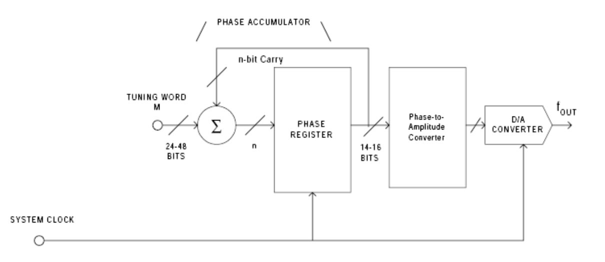
Direct Digital Synthesis

Overview of direct digital synthesis



Direct Digital Synthesis

Overview of direct digital synthesis

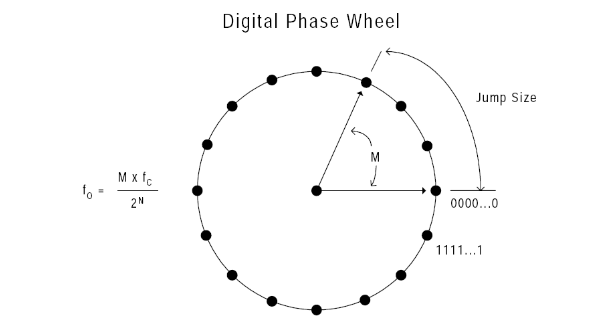


Principle

- Phase register keeps track of current phase.
- Current phase is converted to an amplitude.
- Digital Amplitude is converted to an analog signal.

Direct Digital Synthesis

The phase wheel



The phase wheel

- One cycle in the phase wheel corresponds to 2π phase difference.
- Bigger step size corresponds to higher frequency.

Direct Digital Synthesis

How is the frequency calculated

- Every clock cycle the phase register is incremented by a certain amount
- The increment n_{inc} determines the frequency

$$f_{out} = f_{clock} \frac{n_{inc}}{2^{32}}$$

- The frequency resolution depends only on the reference frequency

$$\delta f_{min} = f_{clock} \frac{1}{2^{32}} = 800\,\mathrm{MHz}\,\frac{1}{2^{32}} \approx 0.1\,\mathrm{Hz}$$

- The maximum output frequency is determined by the sampling theorem:

$$f_{max} \approx 0.4\, f_{clock}$$

Phase Coherent Switching

What is phase coherent switching?

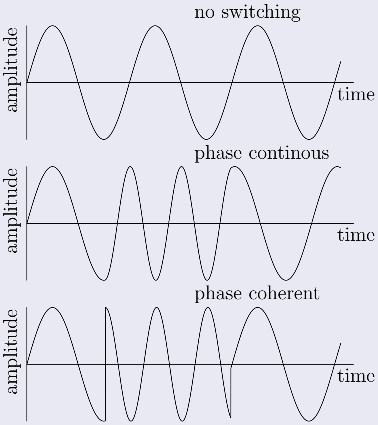


Figure: Different phase switching methods

Phase Coherent Switching

Realization of phase coherent switching

- For every different frequency used in one cycle a seperate phase register is needed
- The DDS has only one phase register
- The FPGA includes 16 independent phase registers

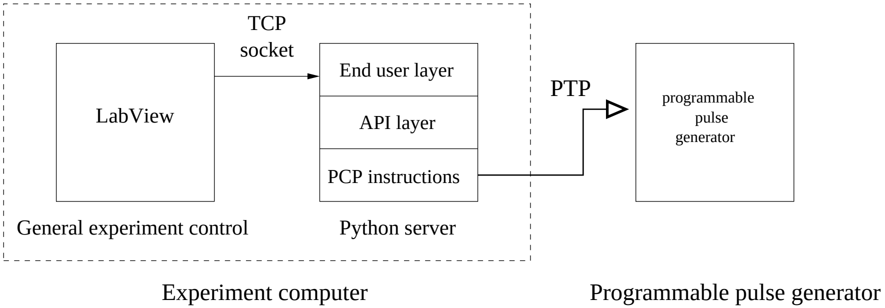
Making pulse shaping work

Generating pulse shapes

- Pulse shapes are generated with an variable gain amplifier (VGA)
- This VGA is controlled by a digital to analog converter (DAC)
- The DAC is controlled directly by the sequencer

Outline

The Software



How does a pulse program look like

API layer

```
# Doppler cooling
set_ttl("doppler", 1)
wait(doppler_time)
set_ttl("doppler", 0)
# Generate coherent pulse
switch_on_dds(frequency, phase, amplitude)
wait(rf_time)
switch_off_dds()
# Detection
set_ttl("detection", 1)
wait(detection_time)
set_ttl("detection", 0)
```

How does a pulse program look like

API layer

```
# Doppler cooling
set_ttl("doppler", 1)
wait(doppler_time)
set_ttl("doppler", 0)
# Generate coherent pulse
switch_on_dds(frequency, phase, amplitude)
wait(rf_time)
switch_off_dds()
# Detection
set_ttl("detection", 1)
wait(detection_time)
set_ttl("detection", 0)
```

End user layer

```
DopplerCooling()
pulse_729(theta, phi, "carrier")
Detection()
```

What does the compiler do

Compiler overview

- Decode command string from LabView

What does the compiler do

Compiler overview

- Decode command string from LabView
- Convert end user program to API program

What does the compiler do

Compiler overview

- Decode command string from LabView
- Convert end user program to API program
- Convert API program to machine code

What does the compiler do

Compiler overview

- Decode command string from LabView
- Convert end user program to API program
- Convert API program to machine code
- Send machine code to sequencer

What does the compiler do

Compiler overview

- Decode command string from LabView
- Convert end user program to API program
- Convert API program to machine code
- Send machine code to sequencer
- Start sequencer

Communication with LabView

LabView communication

- Communication is realized over a TCP connection

Communication with LabView

LabView communication

- Communication is realized over a TCP connection
- Communication with LabView is handled in 3 steps:
  - LabView sends command string to server

Communication with LabView

LabView communication

- Communication is realized over a TCP connection
- Communication with LabView is handled in 3 steps:
  - LabView sends command string to server
  - Server returns to LabView that he started the compilation

Communication with LabView

LabView communication

- Communication is realized over a TCP connection
- Communication with LabView is handled in 3 steps:
  - LabView sends command string to server
  - Server returns to LabView that he started the compilation
  - Server returns results of compilation to LabView (errors, PMT events, compilation time)

Communication with LabView

The command string

- Example command string:

```
NAME,test_ttl.py;TRIGGER,NONE;FLOAT,duration,3.4;
```

- Objects contained in the command string:
  - name of the sequence
  - trigger option
  - sequence variable definitions
  - transitions (frequency, amplitude, shape, ...)
  - initial value of TTL channels

Communication with LabView

Sequence synchronization

- Server reports compilation success to LabView

Communication with LabView

Sequence synchronization

- Server reports compilation success to LabView
- LabView waits until a dedicates TTL output on the Box is set high (QFP Trigger)

Communication with LabView

Sequence synchronization

- Server reports compilation success to LabView
- LabView waits until a dedicates TTL output on the Box is set high (QFP Trigger)
- LabView triggers the Box (Box Trigger)
- QFP Trigger toggles to low

Communication with LabView

Sequence synchronization

- Server reports compilation success to LabView
- LabView waits until a dedicates TTL output on the Box is set high (QFP Trigger)
- LabView triggers the Box (Box Trigger)
- QFP Trigger toggles to low
- After the sequence is finished QFP Trigger is toggled to high again.

Error Handling

Basic error handling

- LabView displays error message from server in left upper corner.

Sequencer2 error handling

- More flexible error handling possible with the sequencer2
- Distinguish between different error classes
- Log errors to different files

Outline

Software

Where to get the software

- From webpage: http://pulse-sequencer.sf.net
- From mercurial repository on anna

Prerequisites

- Python 2.4 or higher (2.5 recommended) from www.python.org
- Mercurial if you want to use the latest repository version from www.selenic.com/mercurial/ (Use TortoiseHG)
- A python compatible text editor (Not Notepad !!!)

Which version of the server to use?

Which version?

- If possible use the sequencer2 software
- Use the old software only if you are using the old DDS/breakout board

Migrating to sequencer2

- Communication for LabView is identical
- Syntax of Pseudo XML files is identical
- Syntax of commands is similar
- Include file handling has changed fundamentally

Installing the server

Installing the software

- `hg clone [path_to_anna]/home/calcium40/ControlPrograms/sequencer/sequencer2`
- use TortoiseHG instead
- Configuration file located in: config/sequencer2.ini

Configuring the server

Configuring the server

- Configuration file located in: config/sequencer2.ini

Parameter Name	Value
box_ip_address	See PTP manual
DIO_configuration_file	Your hardware configuration file
file sequence_dir	The directory of your sequence files
files include_dir	The directory of your include files
nonet	False
reference_frequency	Your DDS reference frequency

Testing the Box without QFP

Test communication to box

- "No PTP reply received" → problem with the network
- Use wireshark to check network traffic www.wireshark.org

Hardware testing

- Hardware testing framework available
- Test the Bus cable and bus connectivity to the FPGA on the DDS board
- Test TTL output system
- Test the DDS
- Check README file of sequencer2

Testing the Box with QFP

Prerequisites

- Check QFP hardware configuration file location
- Check channel number of QFP Trigger (PB_TRIG)
- Check trigger input of Box Trigger
- Good Luck

Common Mistakes

Common errors

- Box trigger channel incorrect
- QFP trigger channel incorrect
- No connection between server and box (no PTP reply received)
- LabView uses comma (,) instead of dot (.) as a decimal seperator

Outline

A Python primer

A simple example:

```
if some_boolean == True:
    print "hello␣True␣world"
else:
    print "hello␣False␣world"
print "Hello␣to␣all␣worlds"
```

A few more words:

- Variables may be used without defining them beforehand
- Python is a dynamic language. Variable types may change during runtime
- Python uses indentation instead of brackets for identifying blocks

Data types

```
# An integer:
X = 12

# A floating point number:
Y = 32.123

# Convert an integer to a floating point:
float_X = float(X)

# Convert a floating point to an integer
int_Y = int(Y)

# A string:
text = "Hello␣world"
text ='"Hello␣world'

# Convert an object to a string:
str_Y = str(Y)

#A list of integers:
list1 = [32, 65, 76, 45]

#Lists may have different datatypes as items
list2 = [234.45, "test", 54]
```

A Python primer

For loops

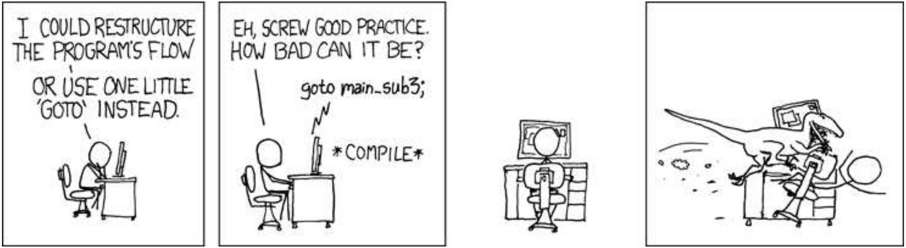
```
for index in range(100):
    print index

example_list = [1, 2, 3, 4, 6]
for item in example_list:
    print item
```

A few more words:

- For loops iterate over an iterable object.
- Iterable objects are:
  - Lists
  - Strings
  - Dictionaries

Python Gotchas

Always use good programming practices



Python Gotchas

Integer division

1 / 2 = 0

- The default division operator for two integer numbers is an integer.
- Be careful when defining your variables

Escape characters

```
filename = 'c:\newfile\file.dat'
```

- In strings the backslash (\) character is an escape character. This means that:
  - \n resembles a newline
  - \t resembles a tabulator
  - ... and a few more
- Filenames should be defined with shlash (/) instead of backslash:

```
filename = 'c:/newfile/file.dat'
```

- Be careful when defining your variables

Python Gotchas

Python uses referencing to variables

```
a = b = 3
a = 4
print a, b
# Prints 4, 3

a = [1, 2, 3]
b = a
a.append(4)
print b
# Prints [1, 2, 3, 4]
print a
# Prints [1, 2, 3, 4]
```

- In the integer case the second assignement of a is a **different** object.
- In the list case the operator **changes** the object, and both a and b refer to the same object.
- For creating a copy use the copy function

Further reading on python programming

More information

- Python website: http://www.python.org
- Software carpentry: http://www.swc.scipy.org/
- Dive into python: http://diveintopython.org/
- Python pitfalls: http://zephyrfalcon.org/labs/python_pitfalls.html

Helpful tools

- pylint static code checker: http://www.logilab.org/857
- Ipython interactive python shell http://ipython.scipy.org
- Python plugin for eclipse: http://pydev.sourceforge.net
- Komodo Edit: http://www.openkomodo.com

Creating a simple sequence file

Pseudo XML file structure

- Sequence files use a markup structure similar to HTML, XML
- Sequence files are **not** valid XML files
- Possible Markups:
  - <VARIABLES>Define sequence variables here
  - <TRANSITIONS> Define transition parameters here (Double pass AOM, ...)
  - <SEQUENCE> The python code for the sequence
  - More Markups for LabView use ...

Creating a simple sequence file

A simple example

```
# Define the sequence variablesxt
<VARIABLES>
det_time=self.set_variable("float","det_time",100000.000000,0.01,2e7)
</VARIABLES>
# The save form specifies which data will be saved and how, when a scan is p
# If this is omitted a standard form is used
<SAVE FORM>
  .dat   ;   %1.2f
  PMTcounts;   1;sum;          (1:N);          %1.0f
</SAVE FORM>
# Here the sequence can override program parameters. Syntax follows from "W
<PARAMS OVERRIDE>
AcquisitionMode fluorescence
DOasTTLword 1
Cycles 1
</PARAMS OVERRIDE>
# The sequence itself
<SEQUENCE>
ttl_pulse(["3", "5"],det_time)
</SEQUENCE>
# Some spooky LabView stuff
<AUTHORED BY LABVIEW>
1
</AUTHORED BY LABVIEW>
```

Declaring variables

Defining variables

```
# Define the sequence variablesxt
<VARIABLES>
#Syntax Example:
sequence_var = self.set_variable("variable_type", "variable_name", \
                                 default_val, min_val, max_val)

#More examples
float_var=self.set_variable("float","name␣for␣labview", 10, 0, 100.0)
int_var=self.set_variable("int","name␣for␣labview", 10, 0, 100)
bool_var=self.set_variable("bool","det_time")
</VARIABLES>

# Use the variables defined aboce direct in the python script
<SEQUENCE>
if bool_var:
    ttl_pulse(["3", "5"],det_time)
else:
    for item in range(int_var):
        ttl_pulse(["3", "5"],det_time)
</SEQUENCE>
```

- The variables block is analyzed by LabView and the variables are available in LAbView as well.

Commands

TTL pulse

```
ttl_pulse(device_key, duration)
```

RF pulse:

```
rf_pulse(theta, phi, ion, transition_param, address=0)
```

Bichro RF pulse

```
rf_bichro_pulse(theta, phi, ion, transition_param, transition2_param, address=0, address2=1)
```

Switch on DDS:

```
rf_on(frequency, amplitude, dds_address=0)
```

Pause:

```
seq_wait(wait_time)
```

TTL Pulses

Usage of the TTL pulse command

```
# Pulse two channels at the same time
ttl_pulse(["channel␣name1", "channel␣name2"], pulse_duration)

# Pulse a single channel
ttl_pulse("channel␣name", pulse_duration)
```

Using the is last statement

- The TTL channel names are defined in the LabView settings editor
  - Set the DIO type to PB for a non inverting channel
  - Set the DIO type to !PB for an inverting channel
- It is possible to create pulses with multiple channels at once.

Creating Frames

Using the is last statement

```
# Add the optional keyword is_last to not reset the start time
# Is last is True if it is omitted

# Create a pulse from time 0 to 100
ttl_pulse(["3", "5"],100,is_last=False)

# Create a pulse from time 50 to 130
ttl_pulse(["1", "4"],80, start_time=50)

#Create a pulse from 130 to 330
ttl_pulse(["3", "7"],200)
```

Transitions

The transition object

- Normally the transition data is transferred from LabView to the server.
- Transition data include:
  - Frequency and amplitude of RF pulse
  - Rabi frequencies for each ion
  - Pulse shape
- It is possible to define transitions directly in the sequence file.

RF Pulses

Generate RF pulses

```
# Generate an RF pulse
rf_pulse(theta, phi, ion, "transition_name")

# Example:
# Generate a pi pulse with phase 0 on ion one
# The transition is the one defined in LAbView as "carrier1"
rf_pulse(1,0,1,"carrier1")
```

Switch on a single DDS

rf_on(frequency, amplitude, dds_address=0)

Using an include file

Include files

- Include files define new commands based on the basic commands shown above
- Generate an include file for every part of your sequence.

```
#An example of a sequence with includes:
<SEQUENCE>
 DopplerCool()
 SidebandCool()
 rf_pulse(1,0,1,"carrier1")
 PMTDetection()
</SEQUENCE>
```

Creating include files

- Include files use only basic commands
- Include files may send a variable back to LabView

```
# Define a Python function with an optional parameter
def PMTDetection(pmt_detect_wait=2000):
    """Generates a PMT readout cycle
    @param pmt_detect_wait: Duration of readout cycle
    """
    # We need to send a return string to LabView
    previous_pm_counts = get_return_var("PM Count")
    if previous_pm_counts != None:
        new_pm_counts = previous_pm_counts + 2
    else:
        new_pm_counts = 2
    add_to_return_list("PM Count", new_pm_counts)
    # Generate the Pulses and wait 50 musecs
    PMT_trigger_length = 1
    ttl_pulse("PMT trigger", PMT_trigger_length, is_last=False)
    ttl_pulse("PMT trigger", PMT_trigger_length, start_time=pmt_detect
    seq_wait(50)
```

Firmware Development

Further firmware development

- Add a counter to the trigger inputs. → Conditional rotations.
- Use FPGA on DDS board to generate pulse shapes → Faster compilation times
- Add simple mathematical functions to the PCP core.
- .....

Further software development

- Add frequency chirped pulses (soon)
- Add Trigger commands to the end user layer

Currently developed (by Max)

- Control of the analog outputs of NI6711 output card
- Possibility to generate long voltage ramps

Read the doucmentation I

- innsbruck-doc for the pcp
  available at http://pulse-sequencer.sf.net
- Master's thesis of Paul Pham
  available at http://pulse-sequencer.sf.net
- A Technical Tutorial on Digital Signal Synthesis
  Analog Devices
  http://www.analog.com/UploadedFiles/Tutorials/450968421DDS_Tutorial_rev12-2-99.pdf