

MQCO: Iontrapping experimental control software

Peter Maunz

January 19, 2014

1 Overview

The Trapped Ion Quantum Control System (TIQCS) consists of FPGA controller hardware and a python computer frontend. On the FPGA (Opal Kelly XEM-6010) an enhanced van Neumann machine is executed which allows the user to control the experimental procedure in a powerful assembler like language.

2 FPGA firmware

The FPGA firmware executes a *Pulse Program*. The Pulse Program can be at most 4096 words long where each command and variable take 1 word of memory. In addition the Pulse Program has access to the FPGA internal memory in which variables are stored. Via simple commands there is also access to the on-board SDRAM (128 MByte). This memory can be written to and read from the host computer. In addition there are two pipes implemented between the host computer and the FPGA Pulse Program. Values can be read from and written to these pipes.

Counters to count incoming pulses are implemented in a gates fashion. The Pulse Program sets the gates for the 24 different counters, the counting is then done in parallel to the execution of the Pulse Program. When the gate is lowered, the accumulated counter value (24 bit) is transferred to the computer via the outgoing pipe without the intervention of the Pulse Program. The counter value can also be read from the Pulse Program.

There are 8 timestamping channels implemented.

2.1 Pulse Program Basics

The Pulse Program uses ‘#’ as the rest of line comment character. Exceptions are the following:

```
#insert <filename>
```

is used to insert the contents of one file at the position of the insert command.

#define *<constant>*

is used to define constants. Constants have to be used for the *<channel>* arguments of the DDS commands. In general if a command takes two arguments, the first argument has to be a constant.

var *<name>* *<value>*, [*<type>*], [*<unit>*], [*<encoding>*]

is used to declare variables. For the Pulse Program only the *<name>* and *<value>* arguments are used. The additional arguments are interpreted by the graphical front-end to enable the control of the variable values. The *<type>* defines what the intended use of the variable. The *<unit>* describes its default dimension and unit. The *<encoding>* determines how a variable value which on the front-end side can be a floating point value with physical quantity is converted into the 32 bit binary value used on the FPGA. The possible types are

parameter The variable will be used as a parameter variable.

exitcode The variable will be used as an exitcode transmitted to the computer at the end of program execution.

trigger The variable will be used for trigger values.

mask The variable will be used as a bitmask to mark significant bits for a shutter value.

shutter *<mask variable name>* The variable will be used as a shutter variable together with the bitmask *<mask variable name>*.

Possible encodings are

AD9912_FRQ The frequency value will be converted into the coarse frequency word for the AD9912 (32 most significant bits)

AD9912_FRQFINE The frequency value will be converted into the fine frequency word for the AD9912 (16 least significant bits)

AD9912_PHASE The value in degree will be converted to the AD9912 phase word

CURRENT

VOLTAGE

TIME The time duration value will be converted into clock cycles.

Independent of the definition, variables with dimension time will always be converted to clock cycles.

$\langle label \rangle$:

Is the declaration of a label. Jump command can set the next command to a label.

2.2 Basic commands

The van Neumann machine uses two main internal registers. The W register is used to hold values while the INDF register holds the address of a variable. Basic command are available to load and store values from the memory into these registers. There are also commands to manipulate the W register.

NOP

No operation

END

End execution.

LDWR $\langle variable \rangle$

load value from variable into W register.

LDWI

load value from the address pointed to by INDF register into W register.

STWR $\langle variable \rangle$

store value in W register into variable.

STWI

store value from W register into address pointed to by INDF register.

LDINDF $\langle variable \rangle$

load the contents of $variable$ into the INDF register.

ANDW $\langle variable \rangle$

$W = W \& variable$.

ADDW $\langle variable \rangle$

$W = W + variable$.

INC $\langle variable \rangle$

$W = variable + 1$.

DEC $\langle variable \rangle$

$W = variable - 1$.

CLRW

$W = 0$.

ORW $\langle variable \rangle$
Bitwise or. $W = W | variable$.

2.3 Comparison and jumps

Jumps can redirect the program flow to a $\langle label \rangle$. Depending on the jump command, a jump is conditioned either on the value of the W register or on the value of an internal compare bit. The internal compare bit is set by a comparison command.

CMP $\langle variable \rangle$
Set W to 0 if $W \leq variable$.

CMPEQUAL $\langle variable \rangle$
compare W and $\langle variable \rangle$ and set the internal compare bit to true if $W = \langle variable \rangle$.

CMPGE $\langle variable \rangle$
compare W and $\langle variable \rangle$ and set the internal compare bit to true if $W \geq variable$.

CMPL $\langle variable \rangle$
compare W and $\langle variable \rangle$ and set the internal compare bit to true if $W \leq variable$.

CMPGREATER $\langle variable \rangle$
compare W and $\langle variable \rangle$ and set the internal compare bit to true if $W > variable$.

JMP $\langle label \rangle$
Jump to $label$.

JMPZ $\langle label \rangle$
Jump to $label$ if $W = 0$.

JMPNZ $\langle label \rangle$
Jump to $label$ if $W \neq 0$.

JMPCMP $label$
Jump to $label$ if the internal compare bit is set.

JMPNCMP $label$
Jump to $label$ if the internal compare bit is not set.

2.4 Shutter and counter-gate subsystem

There are 32 shutter channels (digital output bits) and 32 trigger channels (trigger digital output bits). Both are digital output bits with the difference that the trigger bits will be reset by the firmware after one clock cycle, while the shutter bits have to be set by

the Pulse Program. In addition there are 32 bits of counter gates. These are used to gate the 24 counter and 8 timestamping channels.

Shutter, trigger and counter gates are buffered in the Pulse Program. The commands setting the values only set an internal buffer. One of the two ‘UPDATE’ commands is then used to apply all the values of shutters, triggers and counter gates simultaneously.

For setting the shutter bits an internal mask is used. In this way it is possible to define a mask of bits that will be changed.

SHUTTERMASK *<variable>*

Set internal register shutter_mask to *variable*.

ASYNCSHUTTER *< variable>*

Update internal shutter register, bits set in shutter_mask are updated with the bits from *variable*.

COUTERMASK *< variable>*

Set the internal register with gate signals for the 24 counters and 8 timestampers. Bits 23:0 gate counters 23:0, bits 31:24 gate timestamping on channels 7:0. The 8 external input channels are used repeatedly. External input channel 0 is routed to counter channels 0, 8, and 15. The counts from channels 0 – 15 are transmitted to the computer after the counter is gated low. Channels 16 – 24 are only accessible from the Pulse Program. The timestamping channels are also operated by a gate. On the enabling edge of the gate the current values of a global counter (running at 50 MHz) is transferred to the computer. Each detection event will then trigger the transmission of the current counter value. On counter overrun a overrun marker is written to the host computer. In this way, the total time of the gate is not limited.

TRIGGER *<variable>*

Set internal trigger register.

UPDATE *<variable>*

Update shutters, counter gates, triggers and start the delay counter with the value in *<variable>*. The delay counter runs in the background while the following commands of the Pulse Program are executed. It is necessary to wait for the expiration of the counter with the ‘WAIT’ command before the next ‘UPDATE’.

UPDATEINDF

As update, use the value pointed to by the INDF register for the wait time.

WAIT

wait until the delay counter expires. Waits until the counter is expired. If no counter is running it will not wait. If the counter expired before reaching this command, execution continues.

LDCOUNT $\langle counterchannel \rangle$
load the last counter value from $\langle counterchannel \rangle$ (needs to be ‘define’d)
into W register.

LDTDCCOUNT
load the value from the global timestamping counter into W.

2.5 Pipe from and to host computer

The pipe to and from the computer allow efficient control of the Pulse Program and efficient transmission of results to the host computer. The pipes use 32 bit words. In the case of the pipe to the computer, the most significant 8 bits of the value are a marker for the 24 data bits.

The following marker values are used:

0xffffffff end of experiment marker

0xffffxxxx exitcode marker with 16 bit exitcode

0xff000000 timestamping overflow marker

0xffffxxxx scan parameter with 12 bit address, is followed by additional
scanparameter word.

0x1nxxxxxx 24 bit count result from channel n (4 bit)

0x2nxxxxxx 24 bit timestamp result channel n (4 bit)

0x3nxxxxxx 24 bit timestamp gate start channel n (4 bit)

0x4xxxxxxx other return

The following commands are used to interact with these pipes. Two consecutive commands accessing pipes need to be separated by commands that do not access pipes (e.g. ‘NOP’).

WRITEPIPE
write the value in W into the pipe to the host computer.

READPIPE
read a value from the pipe from the host computer into the W register.
If there is no new data in the pipe, the last value in the pipe is used.

READPIPEINDF
Read the value from the pipe from the host computer in the INDF
register.

WRITEPIPEINDF
Write the value from the INDF register into the pipe to the host com-
puter.

JMPPIPEAVAIL *<label>*

Jump to *<label>* if the pipe from the host computer has data.

JMPPIPEEMPTY *<label>*

Jump to *<label>* if the pipe from the host computer is empty.

2.6 Memory access

It is possible to use the 128 MByte of on-board SDRAM of the Opal Kelly module. However, while the FPGA block ram allows for random access in one clock cycle, the SDRAM has longer read and write latencies. As variables can be used for fast random access values, the use of the external memory is motivated by the need for large amount of memory. Usually, this memory is accessed in large junks.

The memory framework thus uses a FIFO between SDRAM and Pulse Program. Consecutive values can be read from the FIFO in one clock cycle. Repositioning the memory pointer to a new (non-consecutive) values will clear the FIFO and start reading at the new location. There is a delay before the new data is available.

SETRAMADDR *<variable>*

Set the memory pointer to the address in *variable*.

RAMREADINDF

Read one value from the RAM to the INDF register.

RAMREAD

Read one value from the RAM to the W register.

JMPRAMVALID *<label>*

Jump to *label* if the RAM FIFO is valid.

JMPRAMINVALID *<label>*

Jump to *label* if the RAM FIFO is invalid.

2.7 Direct digital synthesizers

The subsystem for the direct digital synthesizer also uses a FIFO to allow for parallel data transmission to all DDS chips. The commands in the Pulse Program execute in one clock cycle. However, after execution, the command is not yet written to the DDS. For the AD9912 it takes approximately $2\mu\text{s}$ to transmit one value (Phase, Frequency or Amplitude).

DDSFREQ *<channel>*, *<variable>*

write frequency (32 most significant bits) from variable to DDS channel. *<channel>* has to be a define. The value is sent to the DDS in the background. It is only updated after an `io_update`.

DDSFREQFINE *<channel>*, *<variable>*

write frequency (16 least significant bits) from variable to DDS channel.

$\langle channel \rangle$ has to be a define. The value is sent to the DDS in the background. It is only updated after an `io_update`.

DDSAMP $\langle channel \rangle$, $\langle variable \rangle$

write amplitude from variable to DDS channel. The value is sent to the DDS in the background and takes effect without `io_update`.

DDSPHS $\langle channel \rangle$, $\langle variable \rangle$

write phase from variable to DDS channel. $\langle channel \rangle$ has to be a define. The value is sent to the DDS in the background. It is only updated after an `io_update`.

WAITDDSWRITEDONE

Wait for the DDS writes to complete.

2.8 Deprecated commands

These commands are only present for backwards compatibility and are not recommended to be used.

`DDSCHN`, `SHUTTER`, `COUNT`, `COUNT1`, `COUNTBOTH`, `DELAY`, `STWR1`, `LDWR1`, `JMPZ1` ,
`JMPNZ1`, `CLRw1`, `[CMP1`

2.9 Common idioms

Here I will list and describe common idioms used in Pulse Programs.

The idiom for generating a digital pulse pattern is the following:

```
1 TRIGGER triggervariable           # write the inteernal trigger buffer
2 SHUTTERMASK maskvariable          # write the shutter mask
3 ASYNCSHUTTER shuttervariable      # update the shutter bits enabled in mask
4 COUNTERMASK countergates          # set the value for counter gates
5 WAIT                              # wait for the last counter to expire
6 UPDATE timevariable               # update trigger, shutter and countermask
7                                   # and start the counter
8 TRIGGER triggervariable2          # write the inteernal trigger buffer
9 SHUTTERMASK maskvariable2         # write the shutter mask
10 ASYNCSHUTTER shuttervariable2    # update the shutter bits enabled in mask
11 COUNTERMASK countergates2        # set the value for counter gates
12 WAIT                              # wait for the last counter to expire
13 UPDATE timevariable2             # update trigger, shutter and countermask
                                   # and start the counter
```

The execution continues from line 6 to line 7 without waiting. Lines 8 to 11 do not affect the output of the FPGA. In line 12 we wait for the last counter of duration `timevariable` to expire. Then we continue to line 13.

An idiom to skip timesteps is realized in the following example. The example is for a step of waiting without changing any external settings.


```

1  QubitWait: NOP
2      LDWR QubitWaitTime      # load QubitWaitTime
3      JMPZ QubitAnalyze      # if time is 0 jump to next step
4      WAIT                    # otherwise wait for last timer to expire
5      UPDATE QubitWaitTime    # and update and set the new timer
6                               duration
7  QubitAnalyze: NOP

```

The jump commands can be used to condition the program execution on the measured results. In the following example for a ion cooling step, we are checking for the ion fluorescence. If we see enough counts, we continue, otherwise we repeat the cooling step. If the ion is not detected after a maximal number of steps we stop program execution.

```

1      LDWR MaxInitRepeat
2      STWR initRemaining
3  cool: NOP
4      SHUTTERMASK CoolingOnMask
5                               # set the mask of shutter to be changed for
5      ASYNCSHUTTER CoolingOn  # set the shutter values
6      COUNTERMASK CheckIonCounters
7                               # set the counter gates
7      WAIT                    # wait for the last counter to expire
8      UPDATE CoolingTime      # update all values and start the cooling
9                               counter
9      COUNTERMASK Null        # close all counter gates, leave cooling on
10     WAIT                    # wait for end of cooling interval
11     LDCOUNT PMTChannel      # load the counter values (PMTChannel
12     CMP                    PresenceThreshold    # was defined as the number of the counter)
12                               # if counts greater than threshold W=W
13                               else W=0
13     JMPNZ pump              # if ion detected go on in the sequence
14     LDWR MaxInitRepeat      # Load the maximum number of repetitions
15     JMPZ pump               #if MaxInitRepeat=0 disable the checking
16     DEC initRemaining        for an ion
17     STWR initRemaining      # Decrease the number of cooling loops left
18     JMPNZ cool              #Store the result
19     LDWR IonLeftExitCode    # Retry if initRemaining is > 0
20     WRITEPIPE              # Ion left, load the exitcode
21     END                    # write the exitcode to the computer
22  pump: NOP                  # End program execution
                               # Here comes the next step

```

3 Interfacing the Pulse Program with the frontend

Currently the front-end uses two ways for scanning values between different experimental points. In the first case the FPGA is able to control all values that have to be changed during a scan. I will call this an *internal scan*, in the other case some external equipment controlled by the host computer has to be changed between experimental points (*external scan*).

3.1 Internal scan

Because no host intervention is necessary for the internal scan, the whole scan can be executed with under Pulse Program control while the communication of results and scan values is done using the pipes. A typical Pulse Program would use the following idiom:

```
1  scanloop: NOP
2      JMPPPIPEEMPTY endlabel      # if no additional data is available in the
3      READPIPEINDF                # pipe we are done with the scan
4      NOP                        # read the address of the variable to be
5      WRITEPIPEINDF              # changed from the pipe
6      NOP                        # write the address back to the computer
7      READPIPE                  # as marker between different points
8      NOP                        # read the new variable value from the pipe
9      WRITEPIPE                 # write it back to the computer
10     NOP
11     STWI                      # store the new value in the variable
12     LDWR experiments          # load the number of experiments to do for
13     STWR experimentsleft      # this value
14     STWR experimentsleft      # store it in the loop parameter
15     experimentloop: NOP
16     here goes everything that makes a single experiment.
17     DEC experimentsleft      # decrease the number of experiments left
18     STWR experimentsleft     # to do, result is in W
19     STWR experimentsleft     # store the result
20     JMPNZ experimentloop     # if experimentsleft<0 jump to
21     JMP scanloop            # experimentloop
22     JMP scanloop            # jump to scanloop
23 endlabel:
24     LDWR myexitcode          # load exitcode into W register
25     WRITEPIPE                # write exitcode to pipe
26     END                      # end execution
```

3.2 External scan

For an external scan the FPGA only has to execute the experiments to be averaged into one result point. The Pulse Program is restarted for every point, however the Pulse Program is not overwritten for each point. Thus one needs to take care to not overwrite variables that will be needed in the next program run.

A minimal program could look like this:

```
1  #define COOLDDS 0
2  var startupMask      1, mask
3  var startup          1, shutter startupMask
4  var startupTime      1, parameter, ms
5  var coolingOnMask     1, mask
6  var coolingOn         1, shutter coolingOnMask
7  var coolingCounter    1, counter
8  var coolingOffMask    1, mask
9  var coolingOff        0, shutter coolingOffMask
10 var coolingOffCounter 0, counter
11 var coolingTime       10, parameter, ms
12 var experiments      350, parameter
13 var experimentsleft  350
14 var epsilon          500, parameter, ns
15 var endLabel 0 xffffff
16
17     SHUTTERMASK startupMask
18     ASYNCSHUTTER startup
19     UPDATE startupTime
20     LDWR experiments
21     STWR experimentsleft
22 cooling: NOP
23     SHUTTERMASK coolingOnMask
24     ASYNCSHUTTER coolingOn
25     COUNTERMASK coolingCounter
26     WAIT                                     # for end of startup or last
27     UPDATE coolingTime
28
29     SHUTTERMASK coolingOffMask
30     ASYNCSHUTTER coolingOff
31     COUNTERMASK coolingOffCounter
32     WAIT
33     UPDATE epsilon
34
35     DEC experimentsleft
36     STWR experimentsleft
37     JMPNZ cooling
```

38
39
40
41
42

4 Front-end

The front-end is used to interact with the Pulse Program. It repares the data to be written and analyzes the results from the Pulse Program. The main window (figure 1) has controls for instruments while the Pulse Program is *not* running. The windows for the DDS, Shutters, Triggers and external instruments are all in separate dockwidgets. These dockwidgets can be re-arranged, torn out into an independent window or closed. Once closed they can be re-open via the View menu or the context menu of any dockwidget.

In addition to the Main window the program has non-modal windows for the Pulse Program configuration (wrench icon), the voltage control (Voltage icon) and FPGA settings (gear icon). Furthermore, there is a "dedicated counters" window available from the graph icon. The separate windows are described in detail below.

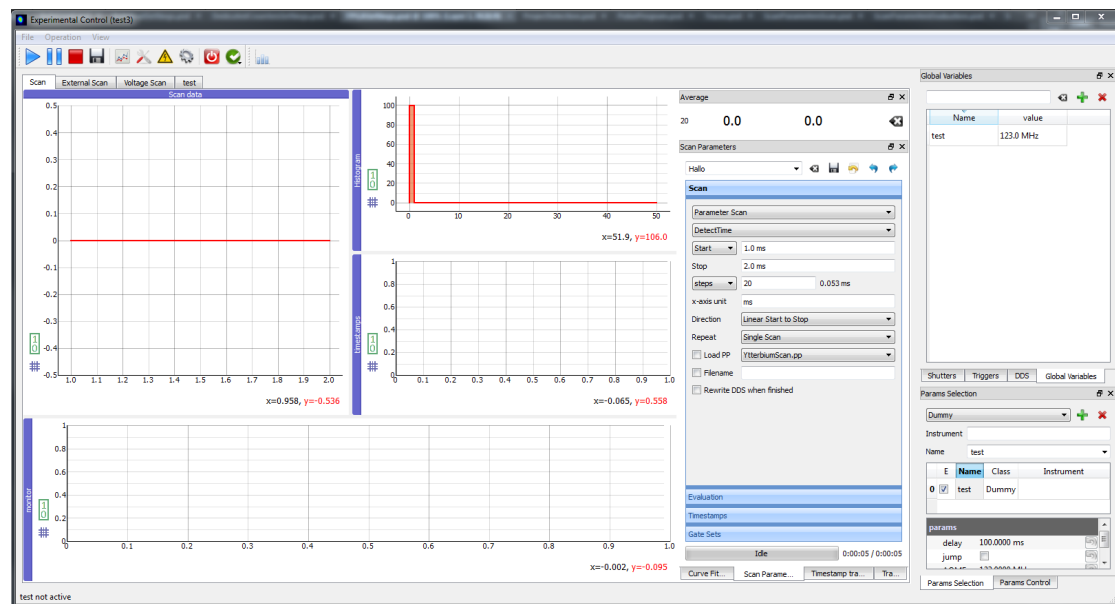


Figure 1: Main window.

4.1 Project Selection

The control program defines a "Base directory". The folders in the Base directory are considered Projects. Each Project has dedicated gui settings and data directories. The

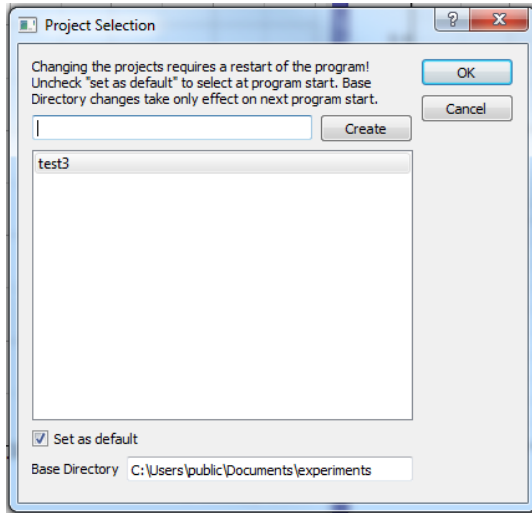


Figure 2: Project Selection. Folders in the base directory are considered projects. All configuration data and result data is saved in the project folder.

configuration data of the control program is saved in the folder .gui-config in the Project directory. It is recommended to create a folder "config" for the pulse program files and other configuration files. Data is saved by default in a directory structure consisting of year, month and day directories. This window is shown on first startup of the control program.

If "Set as default" is checked the control program will start with the default project. In this case the project can be changed by selecting File → Project.

5 FPGA settings

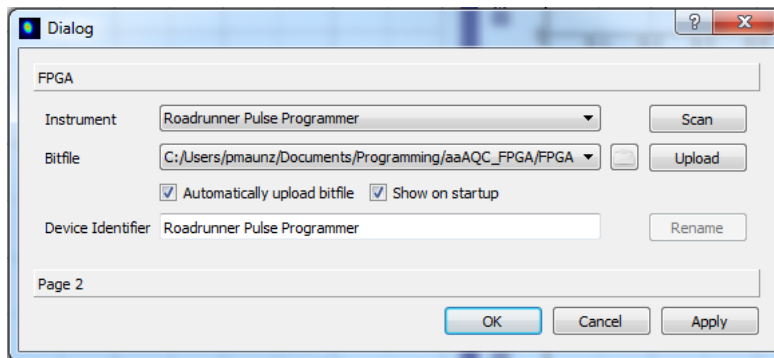


Figure 3: FPGA settings window.

The FPGA settings window (figure3) is shown on start-up. If "Automatically upload bitfile" is checked, the bitfile will be uploaded on "OK". The bitfile can also be uploaded manually. In the case the Instrument selection is empty, please make sure previous processes have been closed (and orphans have been closed). Disconnecting and re-connecting

the USB cable should help too. After fixing the problem press "Scan".

5.1 Plot display

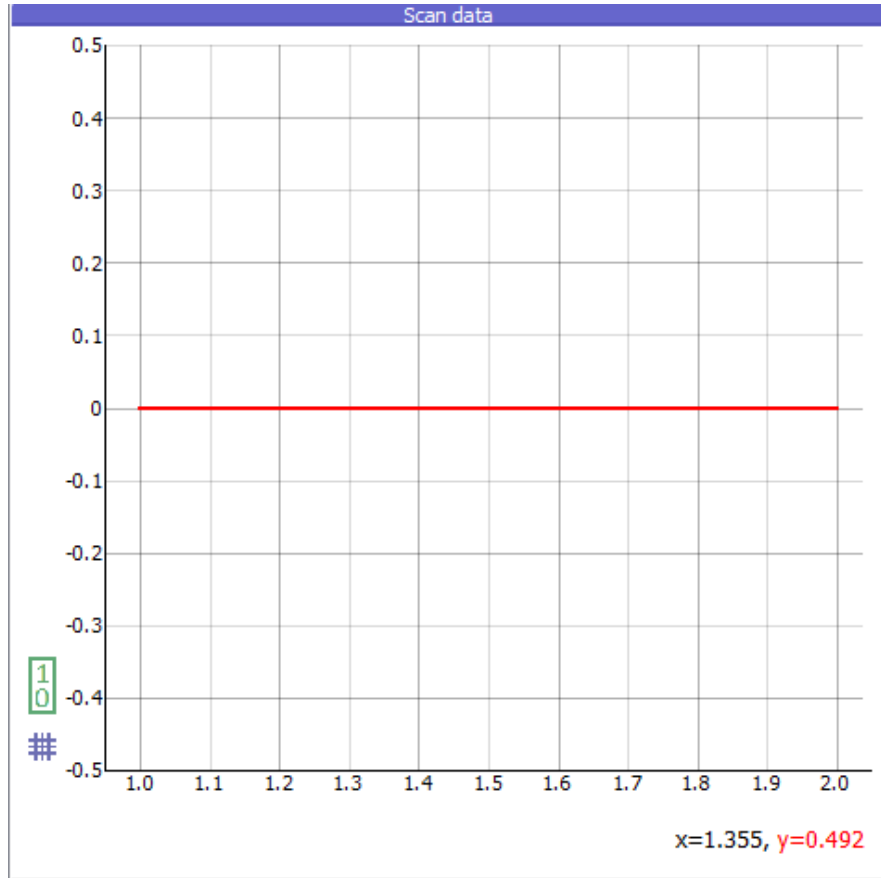


Figure 4: Plot widget. The A button autoscales the axis, the grid button toggles the grid lines and the "1 0" button scales the vertical axis to a unity range.

All plots are displayed using the pyqtgraph library. A typical window is shown in figure 4. The scales can be changed dynamically using the mouse. The middle mouse button can be used for panning, the mouse wheel for zooming in and out. If the wheel is used left of the vertical axis (or below the horizontal axis) the vertical axis, (horizontal axis) is zoomed while the other axis remains in autoscale. The "1 0" button on the left sets the vertical axis to a unity scale. The grid symbol toggles the grid display. The current cursor position is shown on the bottom right. The precision of this display can be increased by first zoomin in.

6 Dedicated Counters

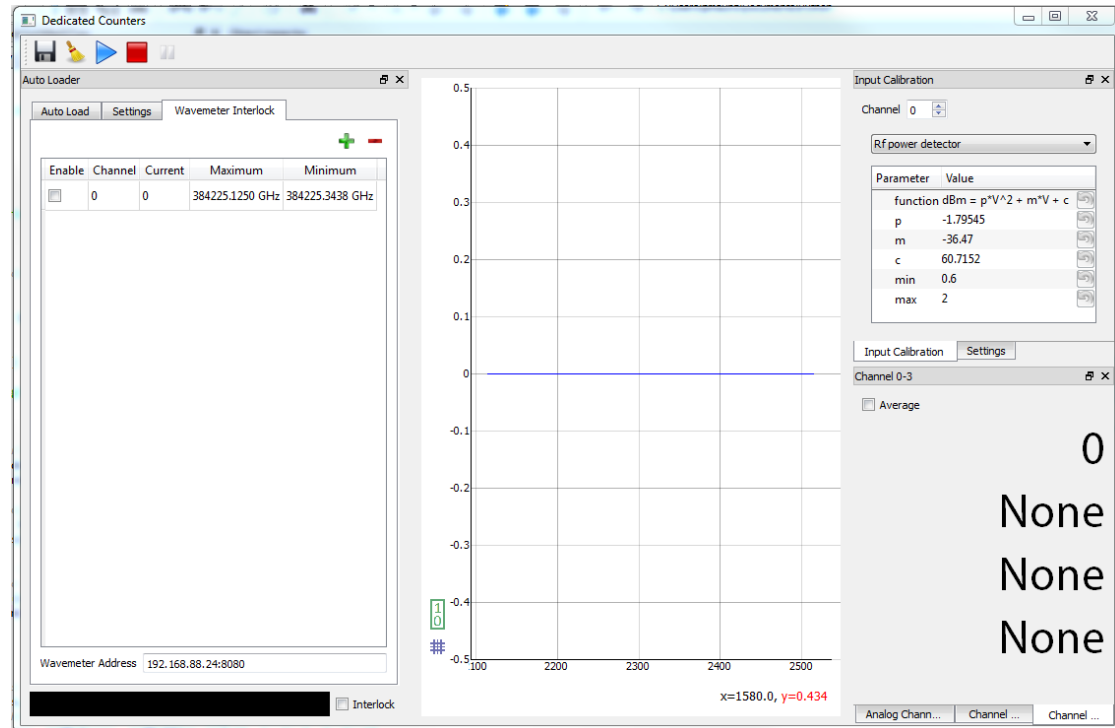


Figure 5: Dedicated counters window. The left dock widget controls the Auto-loading feature. The right dock widgets display the last counter values and analog input values as well as the counter control and input calibration widgets. Dock widgets can be re-arranged, torn out or closed if not needed.

In addition to the Pulse Program, the FPGA has dedicated counters for all 8 digital input lines and 4 analog inputs. These counters are *not* synchronized with the execution of the Pulse Program, nor do they stop during the Pulse Program execution. The analog input channels can be calibrated to voltage or any function can be used to map the values to different quantities. Currently there is a conversion function for reading the Mini-Circuits rf-power detectors. Others can be added easily.

The dedicated counters and the Auto-load feature building upon it are controlled from the "dedicated counters" window (figure 5).

The counter channels can be selected from the

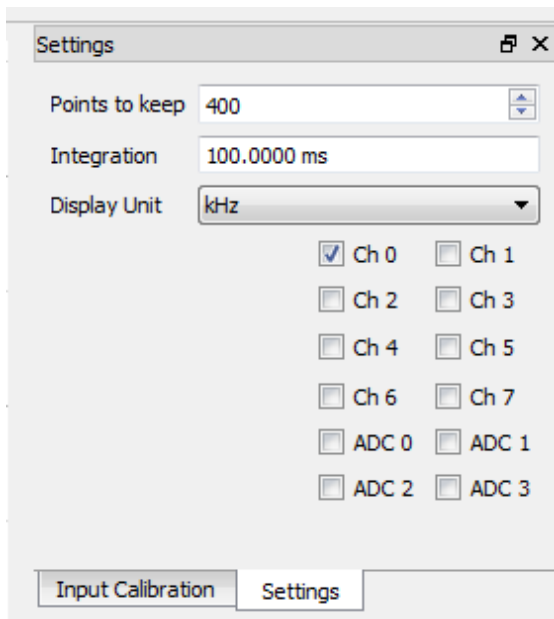


Figure 6: Counter control widget. The 8 digital and 4 analog input channels can be selected independently. The integration time (to be entered with unit) determines the integration time per point. The minimal recommended value is 10 ms.

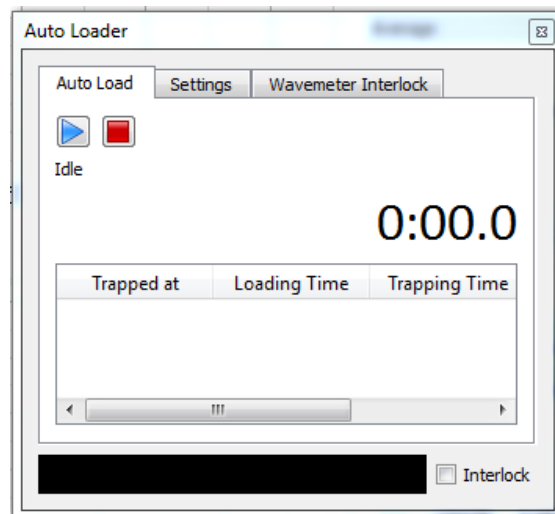


Figure 7: Main window.

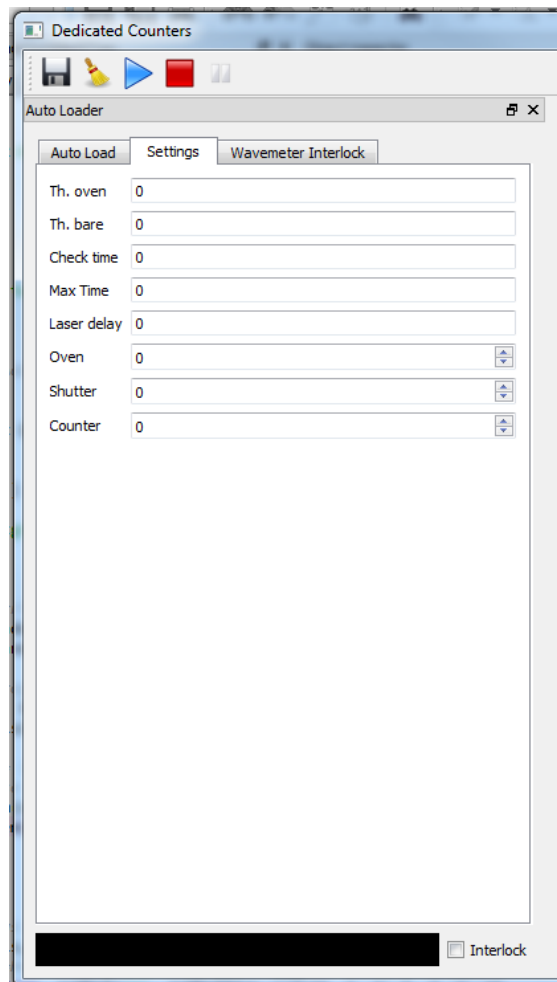


Figure 8: Main window.

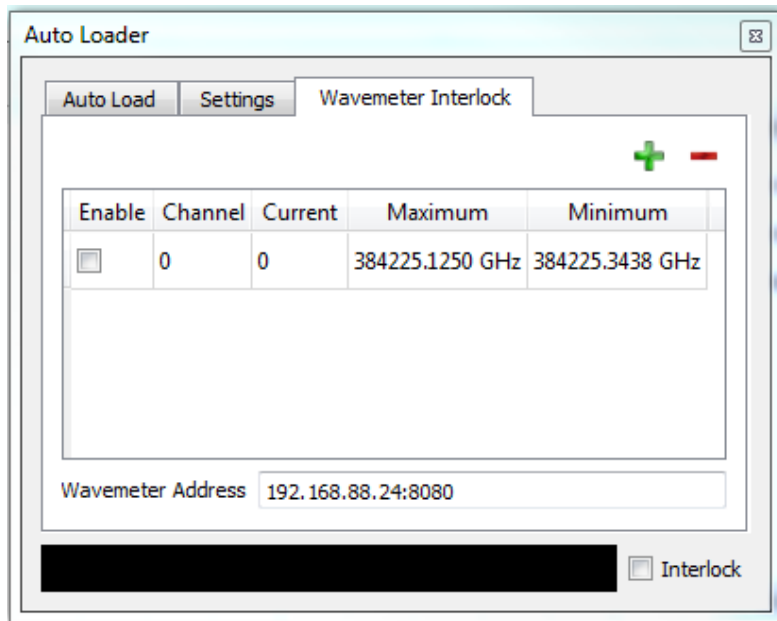


Figure 9: Main window.

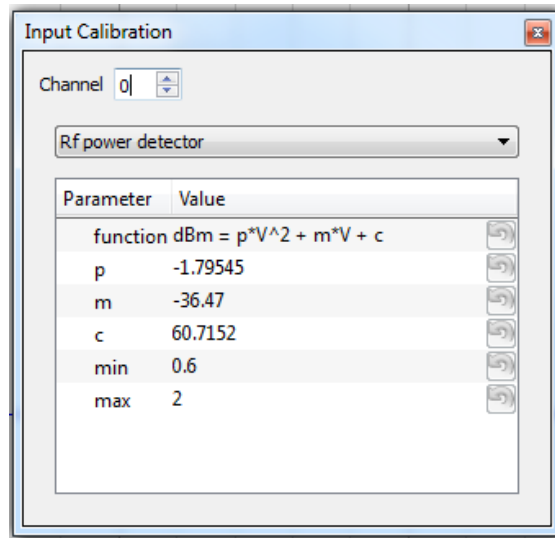


Figure 10: Main window.

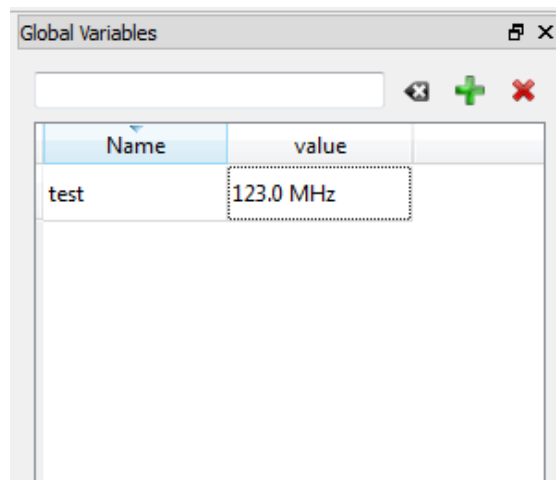


Figure 11: Main window.

DDS

Channel

Frequency	Phase	Amplitude
0.0000 MHz	0.0000 rad	0
0.0000 MHz	0.0000 rad	0
0.0000 MHz	0.0000 rad	0
0.0000 MHz	0.0000 rad	0
0.0000 MHz	0.0000 rad	0

☐ Apply directly
 ☐ Write on startup

Write All

Reset

Apply

Figure 12: Main window.

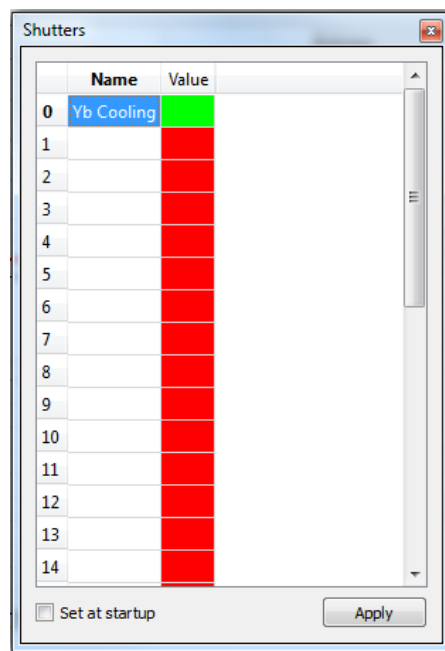


Figure 13: Main window.

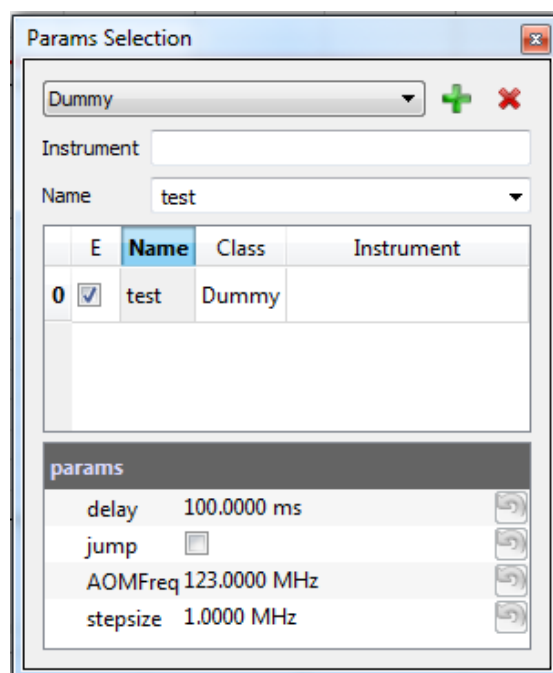


Figure 14: Main window.

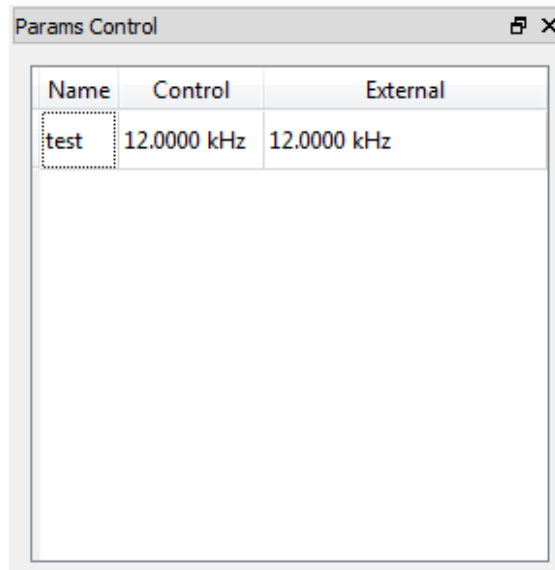


Figure 15: Main window.

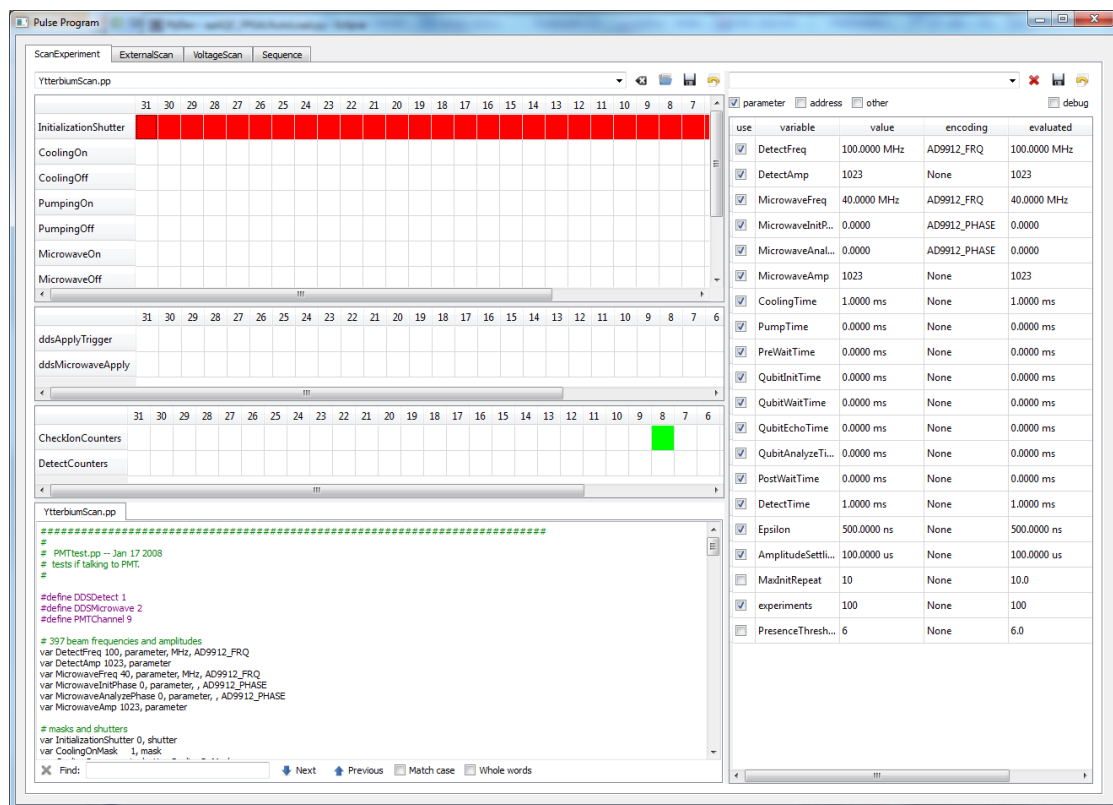


Figure 16: Main window.

6.1 Autoload

6.1.1 Settings

6.1.2 Interlock

6.2 Analog Inputs

7 Global Variables

8 Static settings

8.1 DDS

8.2 Shutters

8.3 Triggers

9 External Instruments

9.1 External Instrument Selection

9.2 External Instrument Control

10 Pulse Program

11 Scan configuration

11.1 Scan settings

11.2 Evaluation settings

11.3 Timestamp settings

11.4 Gate Sets settings

12 Curve fitting

13 Traces

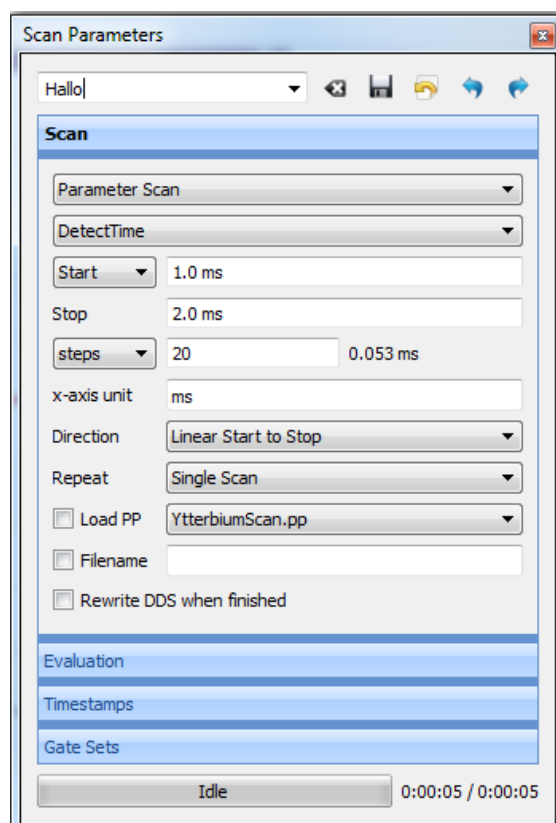


Figure 17: Main window.

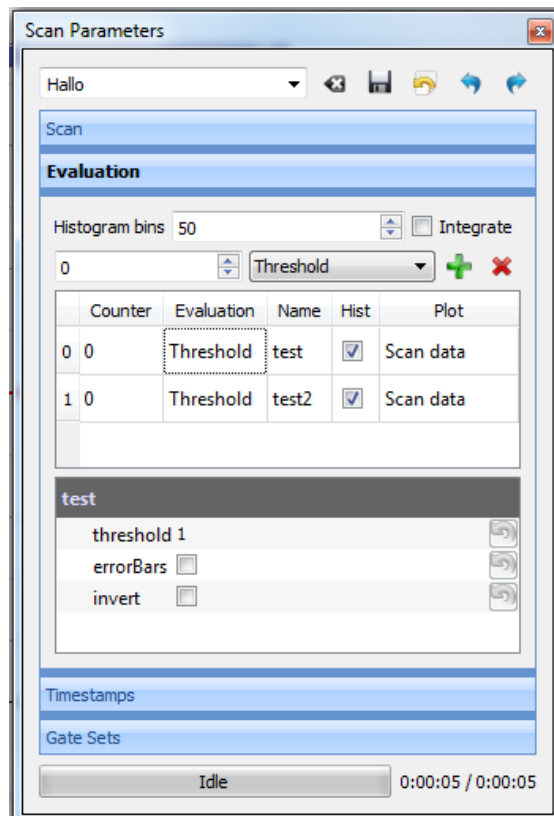


Figure 18: Main window.

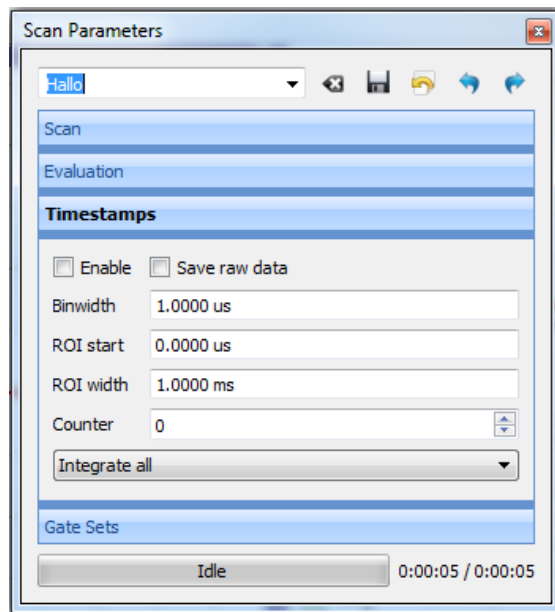


Figure 19: Main window.

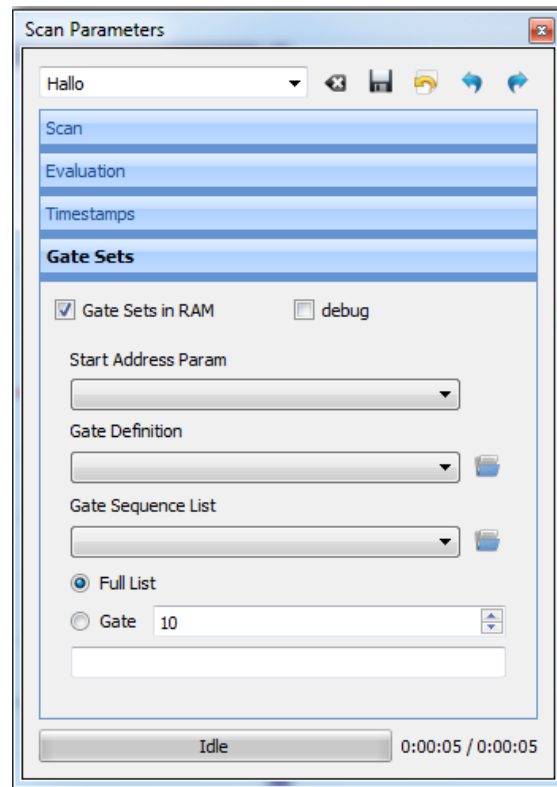


Figure 20: Main window.

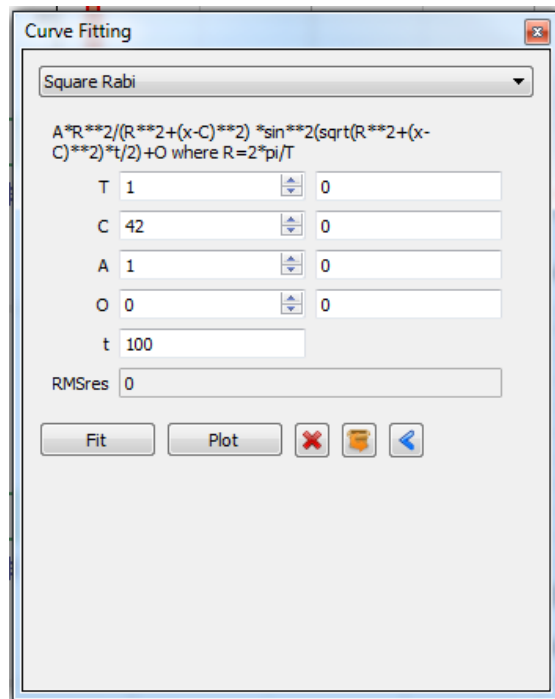


Figure 21: Main window.

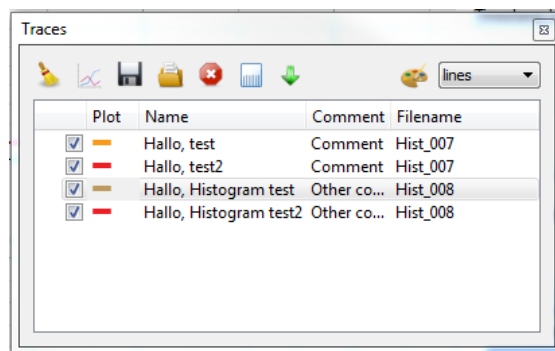


Figure 22: Main window.