

Document Name

D-TACQ Solutions Ltd

AO32CPCI Software User Guide

Prepared By: Peter Milne

Date: 20100110

First issue: 20090316

Table of Contents

| | | |
|-------|--|----|
| 1 | Introduction..... | 6 |
| 1.1 | Intended Audience..... | 6 |
| 1.2 | Scope..... | 6 |
| 1.3 | Glossary..... | 6 |
| 1.4 | References..... | 6 |
| 1.5 | Notation..... | 6 |
| 2 | Operating Mode Summary..... | 7 |
| 2.1 | Modes..... | 7 |
| 2.1.1 | RIM : Registered Immediate Mode..... | 7 |
| 2.1.2 | AWG: Arbitrary Waveform Generator..... | 7 |
| 2.1.3 | LLC: Low Latency Mode..... | 7 |
| 2.2 | Immediate or Triggered..... | 7 |
| 2.3 | Clocking Options..... | 7 |
| 3 | Installation..... | 8 |
| 3.1 | Device Driver Package..... | 8 |
| 3.2 | ACQ196 Host..... | 8 |
| 3.2.1 | Firmware Base Rev..... | 8 |
| 3.2.2 | AO32CPCI Device driver and support:..... | 8 |
| 3.2.3 | Self Test..... | 9 |
| 3.3 | x86 Host..... | 11 |
| 4 | Interface..... | 12 |
| 4.1 | Device Nodes..... | 12 |
| 4.1.1 | Slot Based Addressing..... | 12 |
| 4.1.2 | Root Nodes..... | 12 |
| 4.1.3 | Raw Nodes..... | 12 |
| 4.1.4 | Data Nodes..... | 13 |
| 4.1.5 | Control Nodes..... | 14 |
| 4.1.6 | Signal Nodes..... | 14 |
| 4.2 | Commands..... | 15 |
| 4.3 | Command Details..... | 15 |
| 4.3.1 | MODE..... | 15 |
| 4.3.2 | AO_CLK, DO_CLK..... | 15 |
| 4.3.3 | AO_TRG, DO_TRG..... | 16 |
| 4.3.4 | AWG Mode Control..... | 16 |
| 4.3.5 | Signal Control and monitoring..... | 16 |
| 4.4 | Generate Analog Waveforms..... | 16 |
| 4.5 | Generate Digital Waveforms..... | 17 |
| 4.5.1 | Short cut..... | 17 |
| 4.5.2 | Turbo Dup mode..... | 17 |
| 4.6 | Web Pages..... | 17 |
| 4.7 | EPICS Device Support..... | 17 |
| 5 | Usage Guide..... | 18 |
| 5.1 | Immediate Operation - DC Values..... | 18 |
| 5.1.1 | AO:..... | 18 |
| 5.1.2 | DO:..... | 18 |
| 5.2 | AWG Operation..... | 18 |

- 5.3 Low Latency Control Mode18
- 5.4 Diagnostic : check external clock/trigger state.....19
- 5.5 Example: Channel Ident wire test:.....20
- 5.6 Example: AWG, triggered with concurrent AI capture.....20
- 5.7 Maintenance: FPGA firmware upgrade.....20
- 6 Appendix: Build the Device Driver.....21
 - 6.1 Xscale Target.....21
 - 6.2 x86 target.....21
 - 6.2.1 Important Note for x86 device driver.....22
- 7 Module Parameters.....23
 - 7.1 Parameters for diagnostics.....23

Copyright and Attribution.

Document created using OpenOffice.Org www.openoffice.org.

This document and D-TACQ Software comprising platform Linux port, Linux kernel modules and most applications are released under GNU GPL/FDL:

Document:

Copyright (c) 2009 Peter Milne, D-TACQ Solutions Ltd.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2, with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

Software:

Copyright (C) 2009 Peter Milne, D-TACQ Solutions Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

1 Introduction

AO32CPCI is a 6U CPCI peripheral mode device with 32 x 16 bit Simultaneous Analog Outputs and optionally 64 Simultaneous Digital Outputs. The AO, DO functions may be operated together or separately, in registered or clocked [AWG] modes. The card features internal, external clock and trigger sources.

AO32CPCI operates under the control of a system slot card which is required to master data transfers on the backplane. Two system slot types are supported

- ACQ196CPCI, using bus master DMA to run AWG up to 1MHz
- x86, using programmed IO to run AWG up to 300kHz.

The device driver uses simple, scriptable ascii-coded “knobs” to control all operating parameters. AWG data is binary coded for performance reasons. AWG may be loaded on a channel by channel basis.

1.1 Intended Audience

AO32CPCI end users.

1.2 Scope

Describes the application interface to the AO32CPCI device driver.

1.3 Glossary

- *AO* : Analog Output - eg AO01 .. AO32
- *DO* : Digital Output - eg DO64
- *AWG* : Arbitrary Waveform Generator
- *CPCI*: Compact PCI

1.4 References

1. AO32CPCI installation Guide
2. 2GUG

1.5 Notation

- **command** : indicates name of a program (command)
- `preformatted text` : literal input or output from terminal session.
- *Defined Term* : some term or acronym specific to this domain (perhaps referenced in the glossary)

2 Operating Mode Summary

2.1 Modes

AO32CPCI offers three operating modes:

2.1.1 RIM : Registered Immediate Mode

AO32CPCI provides a discrete register for each channel. The device driver maps the registers to users-space file nodes, and also makes an ascii-to binary translation.

So a simple user script can write ascii scalar to any channel, providing a very simple way to set DC and slow outputs to discrete channels.

2.1.2 AWG: Arbitrary Waveform Generator

The device driver provides a file-per-channel interface. Application code writes a binary waveform time-series to each channel node and then triggers the card. AO32CPCI plays out the waveform simultaneously on each channel. The AWG plays “canned waveforms” - first the application loads the waveform, then the waveform is replayed, either once or in a loop. The number of samples in the waveform is controlled by buffer lengths in the kernel device driver. The device driver has to transform the channelized data contained in the channel files to a multiplexed format used by the AO32CPCI, this will cause a delay on arming the board. During waveform playback, the device driver is responsible for feeding a FIFO buffer on the AO32CPCI; a low-tide interrupt signals the driver to refill the FIFO.

Currently on ACQ196CPCI host only, it's possible to load very long AWG patterns to memory in multiplexed format, allowing long waveforms with zero start delay. A similar feature could be implemented for x86.

2.1.3 LLC: Low Latency Mode

This is used by high performance control systems, where the host computer writes a channel slice to the AO32CPC and this is immediately sent to all channels.

2.2 Immediate or Triggered

All modes are qualified by an Immediate or Triggered control.

- Immediate: the value is written to the DAC right away
- Triggered: the DAC output value is only updated on external trigger.

2.3 Clocking Options.

AWG modes are clocked. There is a choice of internal or external clock. The internal clock is 33MHz/N where N is an integer divide set by the user. The divider can also be used with external clock. AO32CPCI can route clock and trigger signals via the front panel or via PXI-compatible signaling lines in the backplane. One AO32CPC can send master clock, trigger signals to another.

3 Installation

3.1 Device Driver Package

This document is valid for this revision or later:

Source Code:

```
ao32cpci_linux_drv-201101100929-src.tgz
```

Device driver precompiled for ACQ196:

```
ao32cpci_linux_drv-200905120850-xscale.tgz
```

3.2 ACQ196 Host

3.2.1 Firmware Base Rev

Please that ACQ196 is running this revision or later :

```
acq2xx-2.6.21-acqX00-154.2077.2984-200903152203
```

3.2.2 AO32CPCI Device driver and support:

Check that there is no previous release stored on ACQ196

```
ssh root@IP: rm /bigffs/ao32cpci*
```

Should return blank.

```
scp ao32cpci_linux_drv-DATECODE.tgz root@IP:/bigffs
```

```
ssh root@IP:
```

```
cd /bigffs;ln -s ao32cpci_linux_drv-DATECODE.tgz ao32_linux_drv.tgz
```

Check that this function is present in /ffs/rc.local.funs

```
load_ao32cpci()
{
    (cd /;tar xvzf /bigffs/ao32_linux_drv.tgz
    if [ -f /bigffs/ao32cpci.defaults ];then
        source /bigffs/ao32cpci.defaults
    fi
    load.ao32cpci)
}
```

Uncheck this option in /ffs/rc.local.options:

```
AO32CPCI=YES
```

3.2.3 Self Test

Loopback AO0132 to AI0132, DO0031 to AI3364, DO3263 to AI6596.

Ideally, set up dt100rc to monitor the ACQ196CPCI (Control Page).

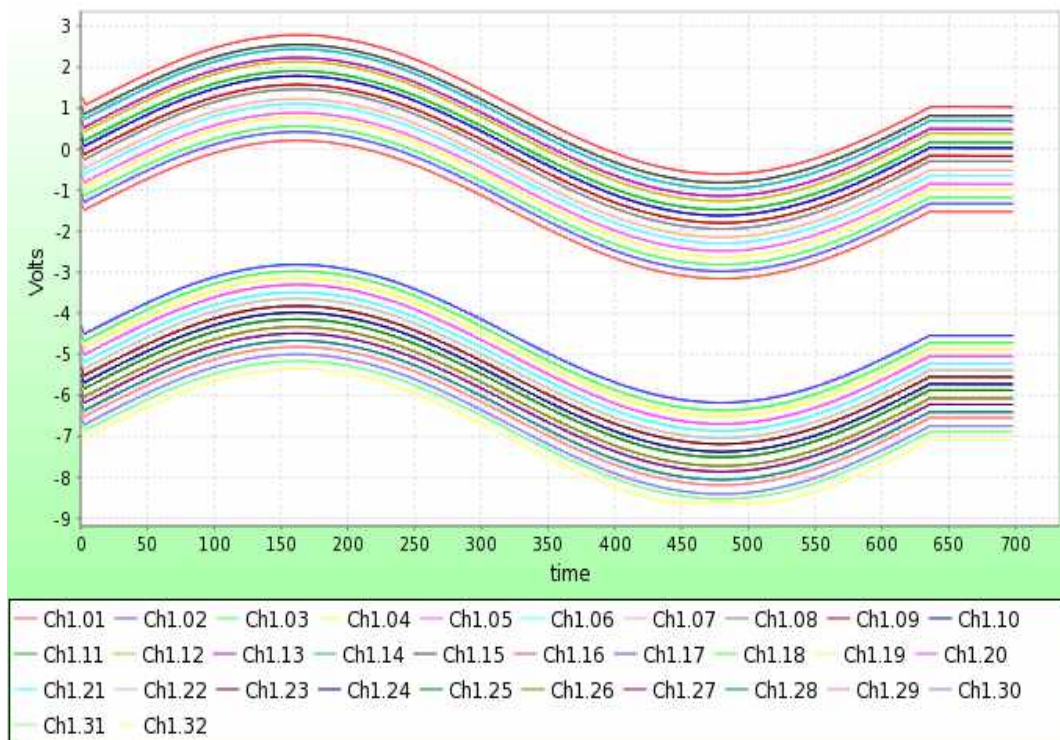
Optionally, set one Firefox browser window to the lower AO32.SLOT page, enable refresh.

```
cd /usr/local/CARE; ./ao32-self-test
```

Set a second Firefox browser window to the DIO.cgi page.

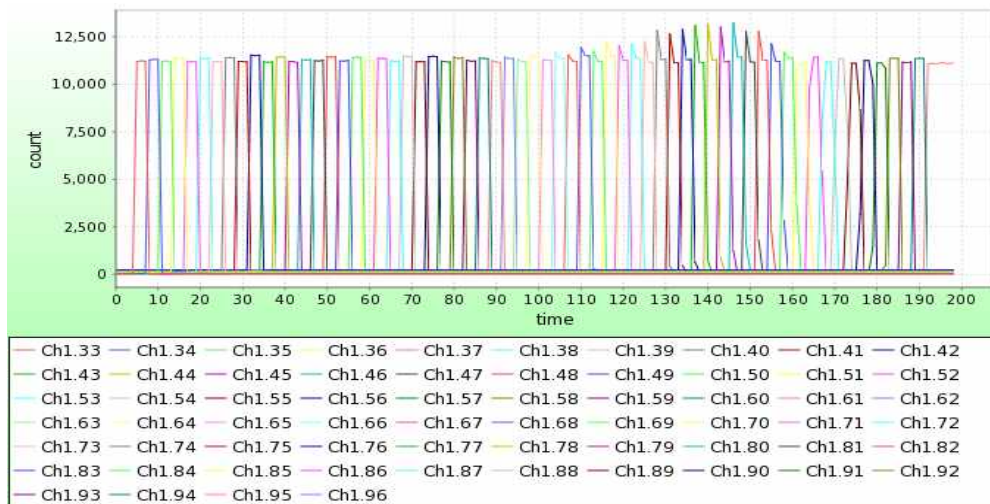
Press the DOWN ARROW on D3 to trigger the capture.

In dt100rc, use View After - AO16-32 show mirrored sine waves, while AO33-96 show a walking bit pattern test.

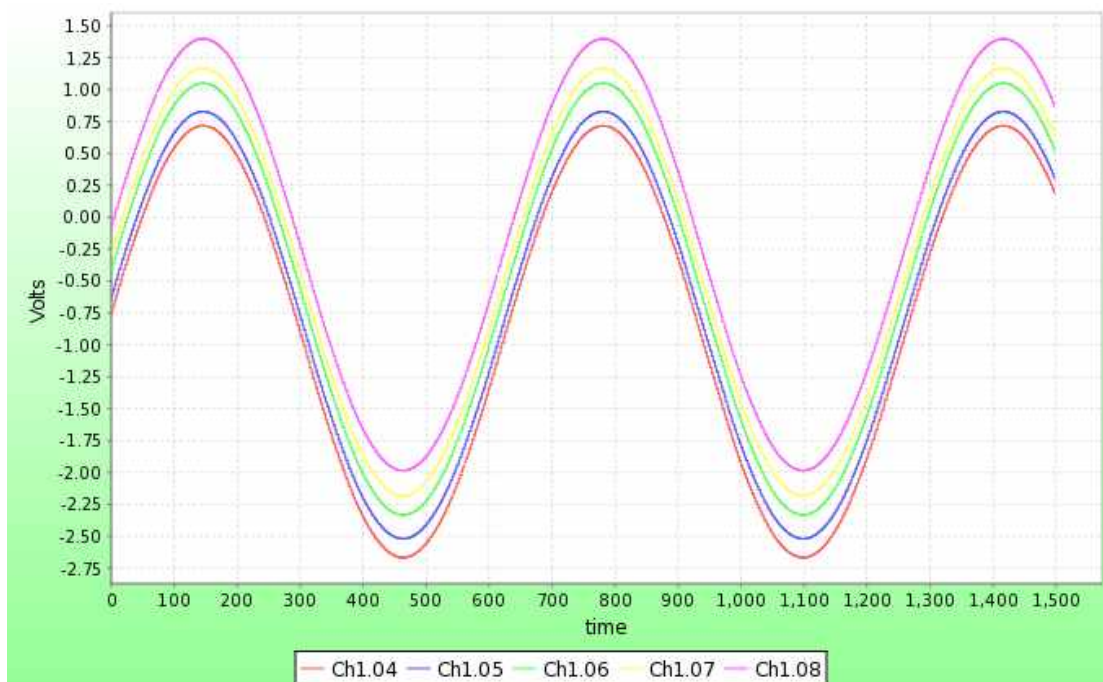


ao-self-test: AO0132 on AI0132. Note final value is maintained.

ao-self-test : DO00643 on AI3364, AI6596 - "Walking Bit Test", somewhat undersampled, hence catching the transient overshoot is a matter of chance



AWG switched to repeat mode - smooth repetition of the single cycle waveform.



3.3 x86 Host

You will probably have to build the Linux host driver from sources to create a kernel object module that is compatible with your host OS.

Please see Appendix [6.2] for Details

A self-test script is provided for x86:

```
./CARE/x86-ao32-self-test
```

Contents of this file as follows. It's recommended to use this as a basis for your own AWG control.

```
#!/bin/bash
# self test for AO32CPCI, hosted from x86

SLOT=${SLOT:-8}
NP=${NP:-512}
MODE=${MODE:-M_AWGI}
AO_CLKSRC=${AO_CLKSRC:-S_INTERNAL}
AO_CLKDIV=${AO_CLKDIV:-990}

WAVEFORM=AO.sin.512

DATAROOT=/dev/ao32cpci/data/ao32cpci.${SLOT}/

let ch=1
while [ $ch -le 32 ]; do
    CHX=$(printf "f.%02d" $ch)
    cp $WAVEFORM $DATAROOT/$CHX
    let ch="$ch+1"
done

set.ao32 $SLOT AO_CLK $AO_CLKSRC $AO_CLKDIV
set.ao32 $SLOT AO_MODE $MODE
set.ao32.data $SLOT commit 0x2
```

This script was tested successfully at D-TACQ up to CLKDIV=110

(33MHz/110 = 300kHz).

The maximum buffer size is set by Linux kernel memory allocation at 4MB

4MB/64bytes = 64K samples.

Longer waveform lengths would be possible, but this requires the driver to be rewritten to support multiple buffers.

To start on a hard trigger -eg front panel LEMO:

```
MODE=M_AWGT

set.ao32 $SLOT AO_TRG S_LEMO_CLK_OPTO
```

4 Interface

4.1 Device Nodes

The AO32CPCI Linux device driver presents an interface comprising a number of device nodes (virtual files). Access to most nodes is via simple scriptable ASCII strings, although for efficiency *AWG* data is binary.

4.1.1 Slot Based Addressing

By default, per-card nodes are numbered by *CPCI* slot.

An alternative resource-order addressing is possible, however slot-numbering is generally more useful.

Examples below assume a card in Slot 3. The device driver supports up to 8 cards.

4.1.2 Root Nodes

| | |
|-----------------------------------|-------------------------------|
| /dev/ao32cpci/ | master root node |
| /dev/ao32cpci/ctrl/ao32cpci.SLOT/ | control nodes for <i>SLOT</i> |
| /dev/ao32cpci/data/ao32cpci.SLOT/ | data nodes for <i>SLOT</i> |
| /dev/ao32cpci/raw | raw nodes (maintenance only) |

4.1.3 Raw Nodes

| | |
|-----------------------------------|--------------------------|
| /dev/ao32cpci/raw/ao32.3.regs | DEBUG: map to regs space |
| /dev/ao32cpci/raw/ao32.3.fifo | DEBUG: map to FIFO |
| /dev/ao32cpci/raw/ao32.3.flash | DEBUG: map to FLASH |
| /dev/ao32cpci/raw/ao32.3.map | DEBUG: monitor map file |
| /dev/ao32cpci/raw/ao32.3.spiflash | FPGA image in flash |

4.1.4 Data Nodes

| | |
|--|---------------------------|
| /dev/ao32cpci/data/ao32cpci.3 | Data for Slot 3 |
| Registered Modes {AO,DO}_MODE = M_{RIM,RTU} | |
| /dev/ao32cpci/data/ao32cpci.3/32 | DC, AO channel 32 (T) |
| /dev/ao32cpci/data/ao32cpci.3/31 | DC, AO channel 32 (T) |
| | |
| /dev/ao32cpci/data/ao32cpci.3/02 | DC, AO channel 32 (T) |
| /dev/ao32cpci/data/ao32cpci.3/01 | DC, AO channel 32 (T) |
| /dev/ao32cpci/data/ao32cpci.3/D064 | Immediate D064 (B) |
| | |
| AWG Modes {AO,DO}_MODE = M_{AWGI, AWGT} | |
| /dev/ao32cpci/data/ao32cpci.3/f.32 | AWG, AO channel 32 (B) |
| /dev/ao32cpci/data/ao32cpci.3/f.31 | AWG, AO channel 31 (B) |
| .. | |
| /dev/ao32cpci/data/ao32cpci.3/f.02 | AWG, AO channel 02 (B) |
| /dev/ao32cpci/data/ao32cpci.3/f.01 | AWG, AO channel 01 (B) |
| /dev/ao32cpci/data/ao32cpci.3/f.D064 | AWG D064 (B) |
| | |
| /dev/ao32cpci/data/ao32cpci.3/histo | diagnostic fifo histogram |
| /dev/ao32cpci/data/ao32cpci.3/commit | AWG Control Word (T) |
| | |
| LLC Mode {AO,DO}_MODE = M_{LLI, LLC} | |
| /dev/ao32cpci/data/ao32cpci.3/XXLLC | raw LLC write AO, DO (B) |
| | |
| Diagnostics | |
| /dev/ao32cpci/data/ao32cpci.3/dump | diagnostic regs dump |

4.1.5 Control Nodes

/dev/ao32cpci/ctrl/ao32cpci.SLOT/

| | |
|-------------|----------------------------------|
| AO_CLK | AO CLK Definition |
| AO_ICLK_MAS | AO CLK Master Definition. |
| AO_MODE | AO Operating Mode |
| AO_TRG | AO Trigger definition |
| DO0 | DO01..DO32 immediate value ASCII |
| DO1 | DO33..DO64 immediate value ASCII |
| DO_CLK | DO CLK Definition |
| DO_MODE | DO Operating Mode |
| DO_TRG | DO Trigger Definition. |

4.1.6 Signal Nodes

/dev/ao32cpci/ctrl/ao32cpci.SLOT/

| | IO | Read Values | Write Values |
|---------------------|----|---|---|
| SIG_LEMO_CLK_DIRECT | I | H, L | - |
| SIG_LEMO_CLK_OPTO | I | H, L | - |
| SIG_LEMO_TRG_DIRECT | I | H, L | - |
| SIG_LEMO_TRG_OPTO | I | H, L | - |
| SIG_PXI_0 | I | H, L | - |
| SIG_PXI_1 | IO | H, L, 1, 0 | -, 1, 0 |
| SIG_PXI_3 | I | H, L | - |
| SIG_PXI_4 | IO | H, L, 1, 0 | -, 1, 0 |
| | | H:: Input High L:: Input Low 1:: Output 1 0:: Output 0 | - Set Input 1 Set Output 1 0 Set Output 0 |

4.2 Commands

set.ao32 *SLOT KNOB VALUE*

get.ao32 *SLOT KNOB*

set.ao32.data *SLOT KNOB VALUE*

get.ao32.data *SLOT KNOB*

4.3 Command Details

4.3.1 MODE

set.ao32 *SLOT AO_MODE MODE*

set.ao32 *SLOT DO_MODE MODE*

where *MODE* is one of

- *M_RIM* Registered Immediate Mode
- *M_RTU* Registered Triggered Update
- *M_AWGI* AWG Immediate
- *M_AWGT* AWG Triggered
- *M_LLI* Low Latency Immediate
- *M_LLC* Low Latency Clocked

4.3.2 AO_CLK, DO_CLK

set.ao32 *SLOT AO_CLK SRC CLKDIV EDGE*

set.ao32 *SLOT DO_CLK SRC CLKDIV EDGE*

where *SRC* is one of

- *S_INTERNAL* # 33.333 MHz Clock.
- *S_PXI_0*
- *S_PXI_1*
- *S_LEMO_CLK_DIRECT*
- *S_LEMO_CLK_OPTO*
- *S_PXI_3*
- *S_PXI_4*
- *S_LEMO_TRG_DIRECT*
- *S_LEMO_TRG_OPTO*

EDGE = { *FALLING*, *RISING* }

CLKDIV = 1..65000

4.3.3 AO_TRG, DO_TRG

set.ao32 *SLOT* AO_TRG SRC EDGE

set.ao32 *SLOT* DO_TRG SRC EDGE

where *SRC* is one of

- S_PXI_0
- S_PXI_1
- S_LEMO_CLK_DIRECT
- S_LEMO_CLK_OPTO
- S_PXI_3
- S_PXI_4
- S_LEMO_TRG_DIRECT
- S_LEMO_TRG_OPTO

EDGE = { FALLING, RISING }

4.3.4 AWG Mode Control

set.ao32.data *SLOT* commit COMMIT_WORD

COMMIT_WORD is a bit-wise OR of

- | | | |
|------------------|------|---------------------------------|
| • COMMIT_DC | 0x01 | set DC output |
| • COMMIT_AWG | 0x02 | initiate AWG output on trigger |
| • COMMIT_ONESHOT | 0x20 | oneshot AWG action when enabled |

4.3.5 Signal Control and monitoring

The state of the signal lines can be monitored, and for output-capable lines, set from their respective signal nodes

4.4 Generate Analog Waveforms

Set up clock and trigger conditions

Set AO_MODE to *AWGI* or *AWGT*

If using trigger, define the trigger using **set.ao32** *SLOT* AO_TRG SRC EDGE

Write a pre-cooked binary pattern to /data/ao32cpci.*SLOT*/f.CC

or, on ACQ196, for simple sine waves, **funge**n is a convenient data source.

Commit the data :

set.ao32.data *SLOT* commit 0x2 ;# continuous

set.ao32.data *SLOT* commit 0x22 ;# one-shot

Note: the commit process also includes a “transform process” that converts the channelized data to raw (multiplexed) form. As the waveform grows longer, this may become to time consuming. It should also be possible to load the waveform in multiplexed format, if required; this would be somewhat faster.

4.5 Generate Digital Waveforms

```
csv2do64 -s SLOT dX=def-file1 dY=def-file2 ....
```

Where

dX : d0 .. d63

def-file is an ascii definition for one bit

Each line in def-file is a pair: *START-CLOCK, LENGTH*

An extreme example can be found in /usr/local/CARE/do64-walking-bit-test

Note that the waveforms generated in this way can easily be very long, and may overrun the default driver buffer allocation. The buffer allocation may be extended - see 7 for details.

4.5.1 Short cut

If all the files happen to be named d01, d02 ... etc, the following option works:

```
csv2do64 -s SLOT d*
```

4.5.2 Turbo Dup mode

If the output data set from csv2do64 is very long but sparse (long runs of the same value, the following option gives a considerable speed up:

```
AO32CPCI_TDUP=1 csv2do64 -s SLOT d*
```

4.6 Web Pages

- shows CTRL parameters (Read Only)
- shows diagnostic
- immediate control of DO (To Do).

4.7 EPICS Device Support

Device support for EPICS is provided. AO32CPCI is intended to be used under control of the embedded

- Immediate PV's for AO, DO
HOSTNAME_AO32_slot_AOcc
- AWG PV's
HOSTNAME_AO32_slot_CHcc

5 Usage Guide

5.1 Immediate Operation - DC Values

5.1.1 AO:

Set *AO_MODE* to *M_RIM*

Set `/dev/ao32cpci/data/ao32cpci.SLOT/CH` to the ASCII encoding of the raw binary value required.

The value is updated immediately.

5.1.2 DO;

Set *DO_MODE* to *M_RIM*

Set `/dev/ao32cpci/ctrl/DO1, ctrl/DO2` to the ASCII pattern required

- pattern is a string of 32 [01] characters.

Alternatively, if a binary interface is required, write 8 bytes to

`/dev/ao32cpci/data/ao32cpci.SLOT/DO64`

5.2 AWG Operation

On ACQ196 please see example `/usr/local/CARE/ ao32-self-test`

On x86 hosts please see example `./CARE/x86-ao32-self-test`

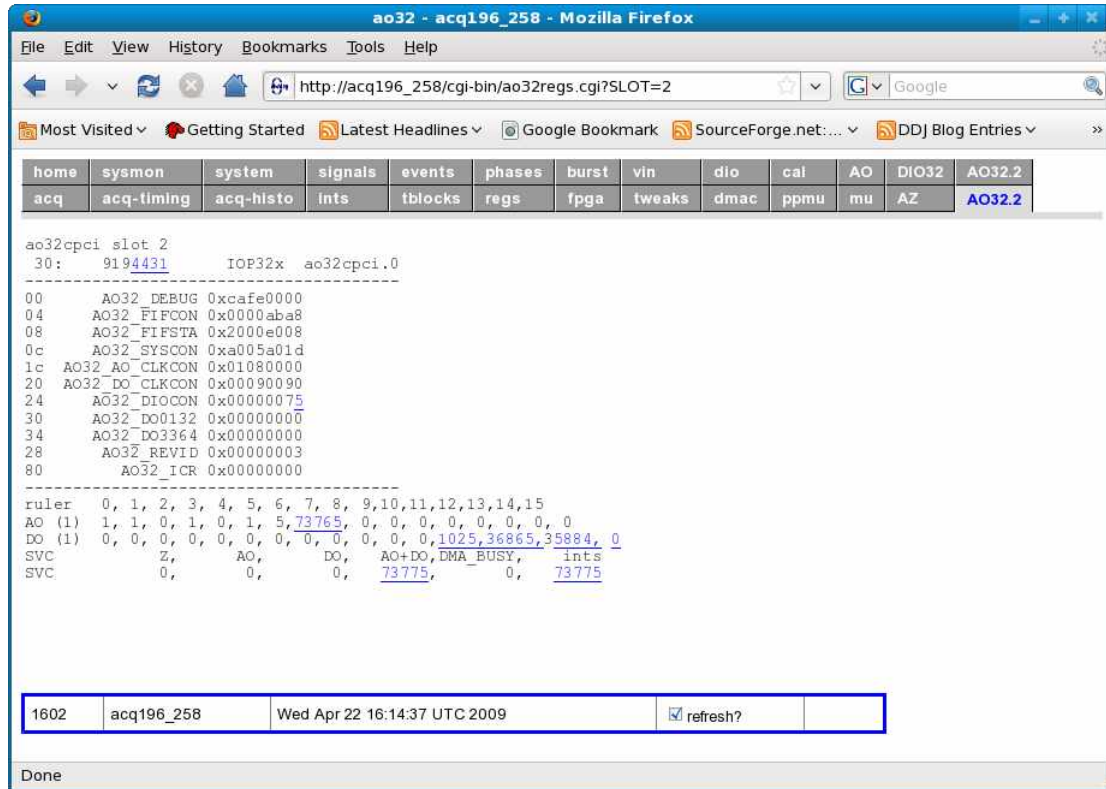
5.3 Low Latency Control Mode

This provides an efficient way to drive all the outputs in one write.

The primary intention is for use in PCS work with the LLC module, however applications such as EPICS may make use of it by writing a 64-byte AO vector + an 8 byte DO vector to the XXLLC node, with *AO_MODE* and *DO_MODE* both set to *M_LLI* or *M_LLC*. The data is sent by DMA immediately on write.

5.4 Diagnostic : check external clock/trigger state

The 8 DI and 2 DIOs are mapped as follows



Point browser to the ao32 regs page. Check the REFRESH box
 Observe register 24 AO32DIOCON - the bottom 8 bits show the current state of the signaling lines.

Put a square wave on the line you are interested in and check for activity, character corresponding to the bit should show in blue. eg in the picture above, there is activity on one of d0-d3, you would have to watch it to see which one ...

| Bit | Hex | Description | I/O |
|-----|------------|-------------------------------|--------------|
| 0 | 0x00000001 | PXI0 Signal | Input |
| 1 | 0x00000002 | PXI1 Signal | Input/Output |
| 2 | 0x00000004 | Front Panel Direct Clk | Input |
| 3 | 0x00000008 | Front Panel Opto-Coupled Clk | Input |
| 4 | 0x00000010 | PXI3 Signal | Input |
| 5 | 0x00000020 | PXI4 Signal | Input/Output |
| 6 | 0x00000040 | Front Panel Direct Trig | Input |
| 7 | 0x00000080 | Front Panel Opto-Coupled Trig | Input |

5.5 Example: Channel Ident wire test:

```
cat ao_id.sh
#!/bin/sh

# make a channel id ramp
# for debug:
# EXEC=echo ao_id.sh

SLOT=${1:-2}

let x=0
let ch=1

while [ $ch -le 32 ]
do
    CH=$(printf "%02d" $ch)
    $EXEC set.ao32.data $SLOT $CH $x
    let x="$x+100"
    let ch="$ch+1"
done

$EXEC set.ao32 $SLOT AO_MODE M_RIM
```

5.6 Example: AWG, triggered with concurrent AI capture

ACQ196 HOST ONLY

Connect:

AO0132 to AI0132

DO0132 to AI3364

DO3364 to AI6596

```
cd /usr/local/CARE/
./ao32-self-test
```

5.7 Maintenance: FPGA firmware upgrade.

This is performed remotely over Ethernet using ssh.

```
remote.update.ao32cpci IP SLOT1 [SLOT 2 ...]
```

6 Appendix: Build the Device Driver

6.1 Xscale Target

This target is supplied pre-built by D-TACQ, so generally there is no need for end-users to build it.

```
tar xvzf ao32cpci_linux_drv-200903251823-src.tgz
make ARCH=xscale
sudo make install
```

The target-side script `load.ao32cpci` is used to load, install the kernel device driver module and to create device nodes. The embedded Linux system will perform the system slot function, enumerating all *AO32CPCI* devices in the *CPCI* system. Both Linux and `load.ao32cpci` will support up to 8 devices, although we have only tested up to 2 devices at any one time.

6.2 x86 target

The device driver has been tested on i686 systems:

Centos 5 : kernel 2.6.18

Fedora 13: kernel 2.6.31.9

The driver is supplied as source code and should be built on-site against the exact kernel headers in use.

Other versions should work. The code may need some slight adjustment to compile cleanly – we welcome user patches and reports to extend the range of known-good kernels.

Unpack the source tarball:

```
tar xvzf ao32cpci_linux_drv-200903251823-src.tgz
make ARCH=x86
sudo make install
```

The `ao32cpci` device driver uses the `udev` hotplug system to create the corresponding device nodes.

It's possible that the PC boot up may be too fast for *AO32CPCI* initialisation.

In that case, delete `ao32cpci_drv.ko` from `/lib/modules/KERNEL/d-tacq/` and insert the driver manually after boot:

```
/sbin/insmod ao32cpci_drv.ko
```

The installed hot-plug script will be triggered on module load.

6.2.1 Important Note for x86 device driver

Only the RIM, AWG functionality is available, however data access is via Programmed IO from the x86, since there isn't a standard DMAC on x86 platforms. This means that AWG update uses more CPU cycles that would otherwise be the case; AWG has be tested to 300kHz, on system using a slow bus extender – embedded x86 hosts should do better.

7 Module Parameters

Driver settings may be modified using the defaults file.

eg

```
cat /bigffs/ao32cpci.defaults
export DEFAULTS="AO_awg_tblocks_count=2 DO_awg_tblocks_count=20
wave_max_samples=110000"
```

| Parameter | Default | Description |
|-----------------------------|---------|--|
| ao_resolution | 16 | #bits resolution (set 18 on -ER card) |
| ACQ196 Only: | | |
| AO_awg_tblocks_count* | 2 | Number of 6MB TBLOCKS allocated to combined AO AWG waveform. |
| DO_awg_tblocks_count | 1 | Number of 6MB TBLOCKS allocated to combined DO AWG waveform |
| wave_max_samples=110000* | 10240 | Maximum AO waveform length, per channel. |
| * both values needed for AO | | |
| | | |
| | | |

7.1 Parameters for diagnostics

| Parameter | Description |
|------------|--|
| AO_lotide | Interrupt threshold, AO |
| DO_lotide | Interrupt threshold, DO |
| epics_mode | Set when used by EPICS |
| OVERRUN | Error flag: Set during AWG operations if FIFO starved. |
| Version | Source revision indicator. |
| | |
| | |