

# DT100 Interface Control Document

Prepared by: Peter Milne, D-TACQ Solutions Ltd

Date: 30 April 2004

The Interface Control Document ICD is a black box description of the DT100 data acquisition system and is the primary description of all external wire and program interfaces.

[Table of Contents](#)

[Table of Commands](#)

[Start of Document](#)

## Change History

Rev	Date	Author	Description
1	01 Aug 99	Pgm	D-TACQ internal doc
2	07 Aug 99	Pgm	First issue
	05 Sep 99	Pgm	1: add test power output to transition module, 2: modify GATED_CONTINUOUS state diagram to show instantaneous transition from ARM to RUN
3			
4	06 Sep 99	Pgm	Various updates
5	08 Apr 00	Pgm	Add device, remote interface defs
6	14 May 00	Pgm	Add sample rate, streaming opts
7	20 May 00	Pgm	Sigcmd responses prefixed ACQ32
8	19 Jun 00	Pgm	Adjust commands, add examples
9	09 Oct 00	Pgm	Add bigdump dma, continuous pre+post, TRIGGERED
10	02 Jan 01	Pgm	Add seek commands
11	25 Jan 01	Pgm	Add Enhanced, CPCI functionality
12	25 May 01	Pgm	Add Low Latency Functionality
13	30 May 01	Pgm	Doc data upload techniques
14	02 June 01	Pgm	reserveAO
15	24 Aug 01	Pgm	diags, thresholds, userLed
16	06 Sep 01	Pgm	remove unwanted commands
17, 18	08 Sep 01	Pgm	add getVoltsRange, getAvailableChannels
19, 20	20 Sep 01	Pgm	multi board sync, addition data triggers
21	11/05/02	Pgm	reformatted
22	27/05/03	Pgm	Updates in progress (htstream etc)
23	30/04/04	Pgm	Note on voltage coding
24	06/06/04	Pgm	Add stream detail, multi-frame streaming def.
25	01/10/04	Pgm	Correct setSyncRoute table as per EB suggestion

## Table of Contents

<u>1 Scope Of Document</u>	<u>5</u>
<u>1.1 Nomenclature</u>	<u>5</u>
<u>2 Physical</u>	<u>5</u>
<u>2.1 Industrial PC version</u>	<u>5</u>
<u>2.2 Compact PCI version</u>	<u>5</u>
<u>3 Electrical</u>	<u>5</u>
<u>3.1 Mains Power</u>	<u>5</u>
<u>3.2 Network</u>	<u>5</u>
<u>3.3 Host Computer</u>	<u>5</u>
<u>3.4 Control Lines</u>	<u>6</u>
3.4.1 Industrial PC.....	6
3.4.2 Compact PCI.....	6
<u>3.5 Signal</u>	<u>6</u>
3.5.1 Signal Socket Labeling.....	7
3.5.2 Signal Socket Wiring.....	7
<u>3.6 Slot Assignment:</u>	<u>8</u>
3.6.1 Industrial PC System.....	8
3.6.2 Compact PCI System.....	8
<u>3.7 Termination Labelling</u>	<u>8</u>
3.7.1 Industrial PC System.....	8
3.7.2 Compact PCI System.....	8
<u>3.8 Clock and Trigger Routing</u>	<u>9</u>
3.8.1 Industrial PC - Clock and trigger daisy chain.....	9
3.8.2 Compact PCI - Clock and trigger routing.....	9
<u>3.9 Signal Conditioning Front End</u>	<u>9</u>
3.9.1 Industrial PC.....	9
3.9.2 Compact PCI .....	9
<u>4 Software Interface</u>	<u>10</u>
<u>4.1 Introduction</u>	<u>10</u>
<u>4.2 Software Deliverables</u>	<u>10</u>
<u>4.3 Modes</u>	<u>11</u>
<u>4.4 GATED TRANSIENT State Diagram</u>	<u>12</u>
<u>4.5 GATED CONTINUOUS State Diagram</u>	<u>13</u>
<u>4.6 SOFT TRANSIENT State Diagram</u>	<u>14</u>
<u>4.7 Device Level Interface</u>	<u>15</u>
4.7.1 Master Nodes.....	15
4.7.2 Data Nodes.....	16
4.7.3 Status Devices.....	17
4.7.4 Special Nodes.....	17
4.7.5 Command Set.....	17
<u>4.8 Diagnostics</u>	<u>26</u>
<u>4.9 API Level Interface</u>	<u>27</u>
<u>4.10 Remote Interface</u>	<u>28</u>
4.10.1 Remote Commands.....	29
<u>4.11 Remote Client</u>	<u>32</u>
<u>4.12 Lan Networking</u>	<u>32</u>
<u>4.13 Subrate Streaming</u>	<u>33</u>
4.13.1 Initiating subrate streaming.....	33
4.13.2 Single Frame (SF) Structure.....	34

4.13.3 Multi Frame Structure.....	34
<u>4.14 Binary Data Format</u>	<u>36</u>
4.14.1 Channel Device.....	36
4.14.2 Cross Section Device.....	36
<u>4.15 Accessing Data</u>	<u>36</u>
<u>4.16 Read from Device</u>	<u>37</u>
<u>4.17 Programmed read from device</u>	<u>37</u>
<u>4.18 Bigdump DMA</u>	<u>37</u>
4.18.1 Raw Channel Order.....	39
<u>4.19 Dynamic User Mapped DMA - DUMDMA</u>	<u>39</u>
<u>4.20 Typical timing comparison for data upload methods</u>	<u>40</u>
<u>5 Enhanced Software Operation</u>	<u>42</u>
<u>5.1 Summary</u>	<u>42</u>
<u>5.2 State Table</u>	<u>42</u>
<u>5.3 Commands</u>	<u>43</u>
5.3.1 Command Listing.....	43
<u>5.4 Use of Existing Modes</u>	<u>45</u>
5.4.1 Example - .....	46
<u>5.5 Loading data for Output Waveform Generation</u>	<u>47</u>
5.5.1 Analog Output AO.....	47
5.5.2 Digital Output DO.....	47
<u>6 Appendix: Command Interface Signal Conditioning SC-1</u>	<u>48</u>
<u>7 Appendix: example remote scripts</u>	<u>50</u>
<u>7.1 Simple initialization and capture</u>	<u>50</u>
<u>7.2 Simple initialization for Gated Continuous</u>	<u>50</u>
<u>7.3 Polling Status</u>	<u>51</u>
<u>7.4 Reading transient data</u>	<u>51</u>
<u>7.5 Streaming Data</u>	<u>51</u>
<u>8 Appendix: Digital Input Routing Commands for Acq32CPCI</u>	<u>52</u>
<u>9 Appendix: ACQ16PCI Differences</u>	<u>53</u>
<u>9.1 ACQ16PCI Channel Mask Hardware Assist.</u>	<u>53</u>
<u>10 Appendix: Low Latency Functionality</u>	<u>53</u>
<u>10.1 Overview</u>	<u>53</u>
<u>10.2 References</u>	<u>54</u>
<u>10.3 Description of Operation of Example</u>	<u>54</u>
10.3.1 External Signals:.....	54
10.3.2 Files:.....	54
10.3.3 Operation .....	54
<u>10.4 Examples</u>	<u>55</u>
<u>11 Appendix: High Throughput Streaming</u>	<u>57</u>
<u>12 Appendix: Examples of Multiple Board Synchronisation</u>	<u>58</u>
<u>12.1 Within a Chassis</u>	<u>58</u>
<u>12.2 Between Chassis</u>	<u>58</u>
<u>13 Appendix: Driver ioctl() and globals parameters.</u>	<u>59</u>
<u>13.1 Driver Load time parameters.</u>	<u>59</u>
<u>13.2 Driver ioctl().</u>	<u>59</u>

A table of commands can be found at the end of the document. [Table of Commands](#)

## 1 Scope Of Document

Defines all customer interfaces to the DT100 Acquisition System.

### 1.1 Nomenclature

Physical signals e.g. pins for wiring are numbered from 1. However the software interface follows the 'C' convention of numbering from 0.

Thus "channel == 0" in software refers to the physical channel AI01.

## 2 Physical

### 2.1 Industrial PC version

The Industrial PC version of dt100 is housed in a 19 inch industrial chassis.

( *W, H, D [mm]* ) 430 \* 170 \* 450

All cabling - power, control. signal is connected to the rear of the chassis.

### 2.2 Compact PCI version

The Compact PCI version is typically housed in a 19" wide, 8U high chassis.

Any PICMG 2.0 rev 3 chassis to customer specification may be used.

## 3 Electrical

### 3.1 Mains Power

110V/240V, 250W. Standard three pin IEC socket.

### 3.2 Network

100 Base T, RJ-45 connector.

### 3.3 Host Computer

- IA32 processor running Linux.
- Main Memory: Minimum 128 MB DRAM
- Disk storage: (where required) 4GB or better , Diskless network boot option.

## 3.4 Control Lines

### 3.4.1 Industrial PC

D-TACQ recommends use of ACQ32TERM02/ACQ16TERM01 transition modules; using these modules, control signal is as follows.

Control Signals CLOCK and GATE applied to 9 way D-type socket (Female), wiring as follows:

A control socket is provided for each board. Use of the control lines is user configurable - the boards may be operated independently or boards may be slaved from a master. Master boards control slave boards via a cross board ribbon cable inside the chassis. The default configuration is for a single board to master all the others in the chassis. In this case, only the master control port is active.

<i>Pin</i>	<i>Signal</i>	<i>Pin</i>	<i>Signal</i>
1	+5V (do not use)		
2	GND	6	CLOCK (active on falling edge)
3	GND	7	GATE (active (ON) low)
4	GND	8	reserved
5	GND	9	reserved

The 5 volt lines are intended to power D-TACQ active termination - generally these should not be used unless possibly to power low power opto isolation. The +5V and (reserved) pins should be left unconnected.

The GATE signal may also be used as a TRIGGER in appropriate modes - the TRIGGER is active on the falling edge, and the trigger pulse must be wider than one sample period.

Please Note: If connecting directly to the front panel RJ45 connector in the absence of TERM0x module, please follow pinout given in ACQxx Installation Guide.

### 3.4.2 Compact PCI

The Compact PCI version allows a full set of clocks and triggers to be specified, with front and rear panel options. The front panel signals are on single pin LEMO connectors, and are opto isolated.

Backplane routed, PXI-compatible clock and trigger signals are used for inter-board synchronisation.

External access to the backplane routed signals is available via rear transition module TM.

## 3.5 Signal

The signal connector format shall be 37 Way D-type sockets (Female), 16 channels per socket.

### 3.5.1 Signal Socket Labeling

<i>Label</i>	<i>Meaning</i>
AI0116	Analog Inputs for channels 01 .. 16
AI1732	Analog Inputs for channels 17 .. 32
AI3348	Analog Inputs for channels 33 .. 48
AI4964	Analog Inputs for channels 49 .. 64
AI6580	Analog Inputs for channels 65 .. 80
AI8196	Analog Inputs for channels 81 .. 96

### 3.5.2 Signal Socket Wiring

<i>Pin</i>	<i>Signal</i>	<i>Pin</i>	<i>Signal</i>
01	In01+		
02	In02+	20	In01-
03	In03+	21	In02-
04	In04+	22	In03-
05	In05+	23	In04-
06	In06+	24	In05-
07	In07+	25	In06-
08	In08+	26	In07-
09	In09+	27	In08-
10	In10+	28	In09-
11	In11+	29	In10-
12	In12+	30	In11-
13	In13+	31	In12-
14	In14+	32	In13-
15	In15+	33	In14-
16	In16+	34	In15-
17	GND	35	In16-
18	GND	36	GND
19	GND	37	GND

Optional on ACQ32TERM01:

The +/- 15 V Aux Power Output shall provide power for customer supplied signal conditioning. The power is supplied via instrument grade high frequency DC-DC convertors. Maximum current is 375 mA per rail per pair of 37 Way D-type connectors. The power supplies have internal foldback current limits and are short circuit rated. Two LEDs to indicate "Voltage OK" on each rail shall be supplied with each transition module.

### 3.6 Slot Assignment:

#### 3.6.1 Industrial PC System

The following slot assignment is necessary so that board to channel alignment is maintained regardless of the number of boards installed, and so that the interrupt line assigned to the acq32 boards does not clash with any other hardware.

PCI slots viewed from left (host CPU) to right

<i>Slot</i>	<i>Function</i>
1	host
2	network
3	ACQ32 #2
4	ACQ32 #1
5	-
6	ACQ32 #4
7	ACQ32 #3

\*\* This slot location is valid for ROBO-XXX (AMD K6-II) systems only. Later systems have a more logical slot assignment.

\*\* WORKTODO: define slot location on Celeron systems.

#### 3.6.2 Compact PCI System

Slot placement in the CompactPCI system is free form, other than the obvious restriction that the System Host computer has to go in the system slot. D-TACQ specify racks with the more normal System Slot on the right layout.

A logical slot position scheme is also available for CompactPCI, but requires keying to the specific type of Host board.

### 3.7 Termination Labelling

#### 3.7.1 Industrial PC System

081/	065/	113/	097/	017/	001/	049/	033/
096	080	128	112	032	016	064	048
Board 3		Board 4		Board 1		Board 2	

#### 3.7.2 Compact PCI System

The Compact PCI system is more free-form and cable labelling format is left to the customer requirement.

Compact PCI boards can be located by software either in resource order or in slot order (recommended).

## 3.8 Clock and Trigger Routing

### 3.8.1 Industrial PC - Clock and trigger daisy chain

For the normal case of a single master, connect the short RJ45 terminated cable from the termination module of Board 1 to the RJ45 on Board1.

The external clock and trigger should be connected to the 9 Way D-Type on Board 1.

Connect the clock daisy chain ribbon cable (16W ribbon Header) between all boards in the chassis.

Numerous other independent clock and trigger combinations are possible.  
Please contact D-TACQ for details.

### 3.8.2 Compact PCI - Clock and trigger routing

The Compact PCI system offers 6 separate input lines, these may be switched under software control to group of

- Mezzanine Input - MI
- Mezzanine Output - MO
- Rear IO - J5 and J3
- Backplane - PXI signals.

## 3.9 Signal Conditioning Front End

### 3.9.1 Industrial PC

The front end circuit comprises a high impedance buffer amplifier.

### 3.9.2 Compact PCI

The Compact PCI system uses a flexible mezzanine architecture to allow application specific signal conditioning functions to be applied. The minimal mezzanine comprises a straight through connection to give the same functionality as the Industrial PC system, other mezzanines provide filtering and input overload protection. Please contact D-TACQ for details.

## 4 Software Interface

### 4.1 Introduction

Application software may interface with the dt100 at any of three levels:

1. Device Level Interface
2. API Level
3. Remote Level

The Customer control software may interact with a number of independent DT100 system boxes, each with the same interface, distinguished by network node name. In other words, each DT100 provides a number of channels, indexed from zero. Each system box can be independently configured, triggered and queried, as required by customer.

The remote control facility allows the customer the possibility of arming multiple BSU's to run concurrently, however for true coherent sampling, each BSU must be supplied with the same hardware clock and trigger signals.

Data may be obtained from the dt100 either locally or remotely from logical data devices or by using the remote control facility.. The data may be obtained either as

1. Transient data post capture
2. Subrate Streaming data during capture - this has to be at a lower data rate, the exact rate depends on the power of the CPU's and the bandwidth of the data paths. Data may be streamed either to disk and/or to a remote network client.
3. HTSTREAM – high throughput streaming via PCI bus.
4. LOWLATENCY – low latency control oriented streaming via PCI bus.

The intention of the streaming mode is to allow a subrate view of the data as it is being captured, with the possibility of viewing the full rate data after captured has terminated.

### 4.2 Software Deliverables

The PC host shall run Linux. All host side software is supplied as source code under GPL.

The host side software is built from sources as part of Acceptance.

The embedded ARM firmware, and the embedded FPGA firmware is supplied as object code.

## 4.3 Modes

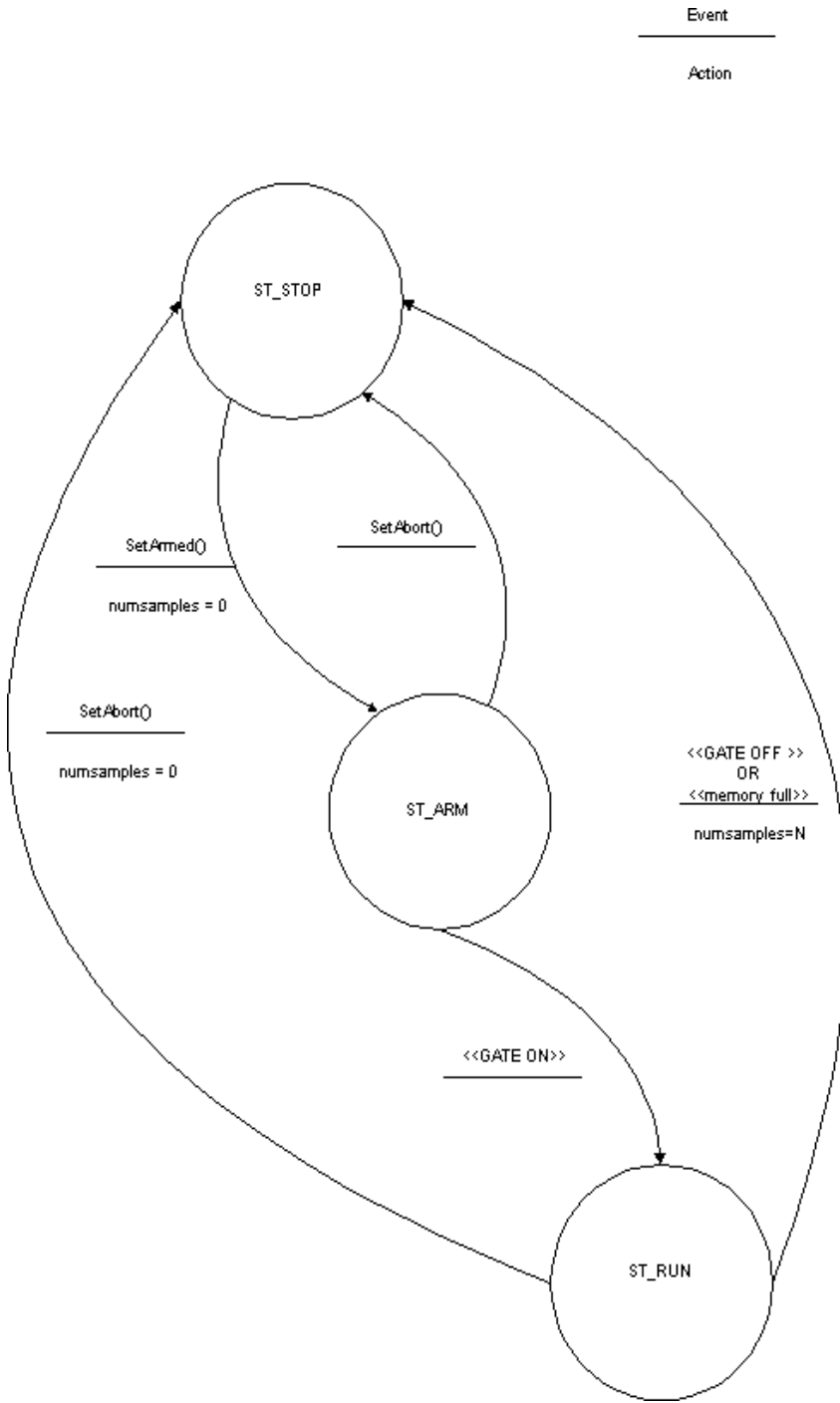
The system can be configured in the following modes:

<i>Mode</i>	<i>Start Condition</i>	<i>Parameter</i>	<i>Stop Condition</i>
GATED_TRANSIENT*	SetArm() then GATE_ON	NUM_SAMPLES	transient capture, terminated by GATE_OFF OR NUM_SAMPLES captured OR end of sample memory
GATED_CONTINUOUS	SetArm() then GATE_ON	PRE_SAMPLES, POST_SAMPLES	continuous capture to circular buffer terminated by GATE_OFF + POST_SAMPLES
TRIGGERED_CONTINUOUS	SetArm()	PRE_SAMPLES, POST_SAMPLES	Continuous capture to circular buffer terminated by TRIGGER + POST_SAMPLES
SOFT_TRANSIENT**	SetArm()	NUM_SAMPLES	NUM_SAMPLES
SOFT_CONTINUOUS	SetArm()		for use with sub rate streaming.
*default			
** used for test			

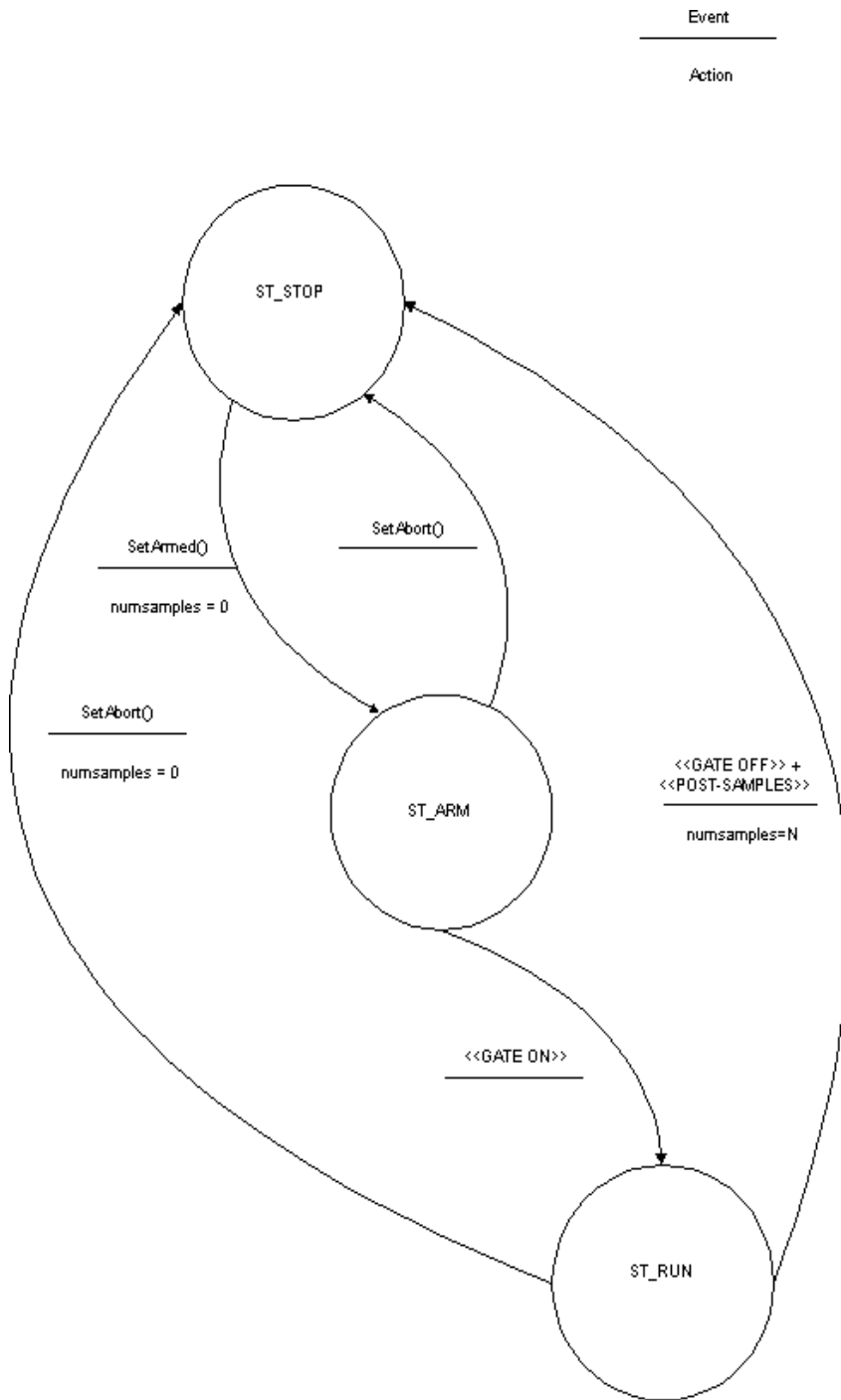
Acq32 defaults to using External Clock. However, if Internal Clock is selected prior to setting the Mode, the Internal Clock shall be used instead. This applies to all Modes. In addition, capture can be aborted at any time from software with the setAbort command. During GATED\_TRANSIENT operation, subrate data may be streamed off the dt100 system.

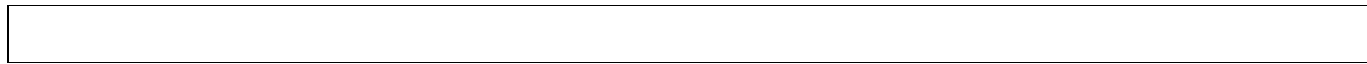
The DT100 system box behaves as a state machine, as described in sections 4.4, 4.5 and 4.6.

### 4.4 GATED\_TRANSIENT State Diagram

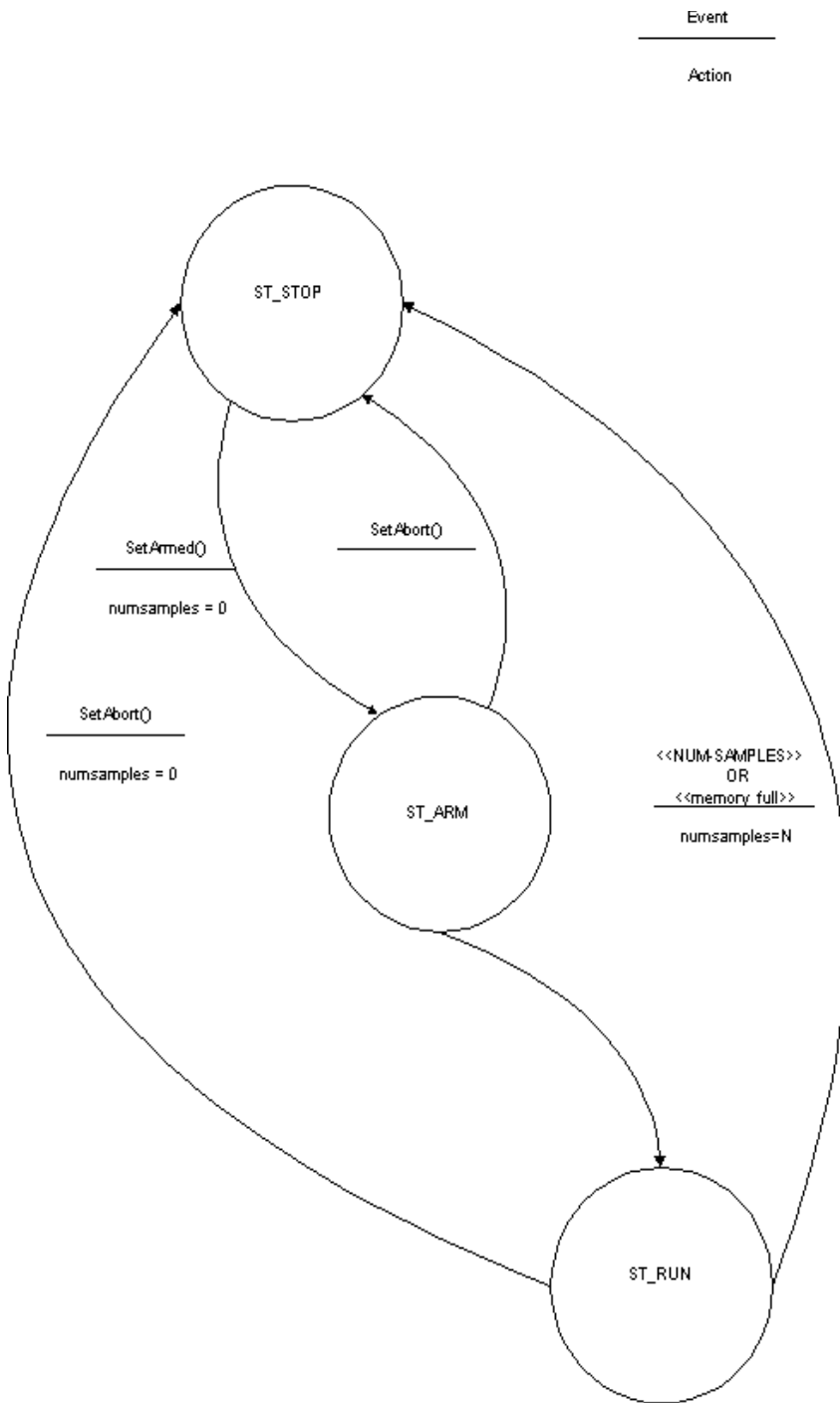


### 4.5 GATED\_CONTINUOUS State Diagram





### 4.6 SOFT\_TRANSIENT State Diagram



## 4.7 Device Level Interface

The acq32 device driver is loaded at boot time, the system auto detects the ACQ32 boards available and configures logical device nodes accordingly. The device level interface is designed to work with regular text utilities such as **cat** and **echo**

**eg**

```

;# output the data from board 1, channel 01
cat /dev/acq32/acq32.1.01

;# set mode on all boards
echo setModeGatedContinuous 10 20 >/dev/acq32/acq32.1.m.all

```

Each command elicits a response string from the device - a utility **acqcmd** is provided to complete the write and read interface that this implied; it is strongly recommended that **acqcmd** be used in any scripting - scripts that do not check for responses will prove to be unreliable.

The device nodes can be used in the following ways:

- Send them ascii commands and queries (the "Hayes Modem" approach)
- read() the data from data nodes
- use mmap() to for more efficient memory mapped access
- use special ioctl() calls. Use of ioctls only used when the functionality cannot be more easily achieved using the first three methods.

### 4.7.1 Master Nodes

Acquisition is configured via Master Nodes. Master nodes may be responsible only for themselves or may control a number of slave modes simultaneously.

NB: for slave control, it is assumed that the inter board clock and trigger sync signals is correctly connected - this is outside the scope of software control.

```

<b> : board number - { 1 .. 7 }

<c> : channel - { 01 .. 32, XX }

{slaves} : n1 [ n2 [ n3 ... ] ] { nX : { 1..7 } }

```

Master devices are specified as:

```

/dev/acq32/acq32.<b>.m{slaves}

```

**eg**

```

/dev/acq32/acq32.3.m3 - board 3, responsible for itself only

```

**/dev/acq32/acq32.1.m1234** - board 1, responsible for itself and slaves 2,3,4

Most common combinations of master, slave are available at boot time. It is possible to create any valid combination from the Linux shell.

A generic master all device, valid for any board combination is supplied: **/dev/acq32/acq32.m.all**.

The device driver for the Master Device supports multiple concurrent open paths - so that multiple processes are able to monitor the acquisition process.

## 4.7.2 Data Nodes

Data devices provide access to the data in a channel

**/dev/acq32/acq32.<b>.<c>**

e.g.

**/dev/acq32/acq32.1.01** ;# board 1 channel 1

**/dev/acq32/acq32.4.32** ;# board 4 channel 32

in addition, a cross section device returns the data for all unmasked channels:

**/dev/acq32/acq32.<b>.XX**

eg

**/dev/acq32/acq32.1.XX**

Output data nodes are provided for the function generator capability, two channel AO and 8 bit DO:

**/dev/acq32/acq32.<b>.AOi** - write a 32 bit value for immediate output by DACs

**/dev/acq32/acq32.<b>.AOf** - write a block of 32 bit values for function generation.

**/dev/acq32/acq32.<b>.DOi** - write a 32 bit value for immediate output at the digital outputs

format of the 32 bit DO is **XX {31:8}, OutputValue { 7:0 }**

**/dev/acq32/acq32.<b>,DOf** - write a block of 32 bit words,

where the word definition is **ClocksToSkip { 31:8 }, OutputValue { 7:0 }**

NB: The output from the function generation modes only appear following definition of the mode, arming and trigger.

### 4.7.3 Status Devices

A status device is available for every board.

**`/dev/acq32/acq32.<b>.stat`**

Applications can block on change of state. Output of a line of text occurs on state change

Format:

```
{jiffies} {delta-jiffies} {state} {OK|ERROR}
```

```
$cat /dev/acq32/acq32.2.stat
00038565 0000 0 ST_STOP OK
00040765 2200 2 ST_RUN OK
00040824 0059 0 ST_STOP OK
00046290 5466 2 ST_RUN OK
00046569 0279 0 ST_STOP OK
```

### 4.7.4 Special Nodes

A number of special nodes are used for system maintenance and debug - eg in-system programming.

**`/dev/acq32/acq32.<b>.host`** : used with `mmap()` to access host memory buffer after `bigdump`

**`/dev/acq32/acq32.<b>.dmabuf`** : not for application use

**`/dev/acq32/acq32.<b>.raw`** : not for application use

**`/dev/acq32/acq32.<b>.flash`** : used for in system flash programming

**`/dev/acq32/acq32.<b>.test`** : not for application use

### 4.7.5 Command Set

The Logical devices all accept ascii command strings in a similar manner to a modem.

The commands may be echoed onto the device from the command line, or sent via a program as required. A each command is responded to by the device. A utility, **`acqcmd`** is provided to simplify this procedure. User applications should wait for the response before sending the next command.

*In order to be sure that the command has been accepted, user applications must parse the response string for errors. Even with correct syntax, a command may not succeed, and there only reliable way to determine success is to evaluate the return string.*

## 4.7.5.1 acqcmd setMode

Command	<b>acqcmd setMode &lt;MODE&gt; &lt;nsamples&gt;</b>  <b>MODE : { SOFT_TRANSIENT, GATED_TRANSIENT, GATED_CONTINUOUS }</b>
Description	Set acquisition mode and length
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.2 acqcmd setModeGatedContinuous

Command	<b>acqcmd setModeGatedContinuous &lt;pre-samples&gt; &lt;post-samples&gt;</b>
Description	Set GatedContinuous mode and lengths
Master/Slave ?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.3 acqcmd setModeTriggeredContinuous

Command	<b>acqcmd setModeTriggeredContinuous &lt;pre-samples&gt; &lt;post-samples&gt;</b>
Description	Set TriggeredContinuous mode and lengths
Master/Slave ?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.4 acqcmd getMode

Command	<b>acqcmd getMode</b>
Description	Returns currently selected mode
Master/Slave ?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.5 acqcmd getNumChannels

Command	<b>acqcmd getNumChannels</b>
Description	Query number of channels available

Master/Slave?	M/S
OK Response	ACQ32:<nchannels>
ERROR Response	ACQ32: ERROR error string

#### 4.7.5.6 getNumSamples

Command	<b>acqcmd getNumSamples</b>
Description	Query number of samples available
Master/Slave?	M/S
OK Response	ACQ32:<nsamples>
ERROR Response	ACQ32: ERROR error string

#### 4.7.5.7 acqcmd getState

Command	<b>acqcmd getState</b>
Description	Query board state
Master/Slave?	M/S
OK Response	ACQ32: <state>
ERROR Response	ACQ32: ERROR error string

#### 4.7.5.8 acqcmd format

Command	<b>acqcmd format &lt;format-type&gt;</b>  <b>&lt;format_type&gt; :</b>  <b>{ bin [default], hex, dec, volts [lines] }</b>  <b>Please Note: this command is not implemented.</b>
Description	Select data format
Master/Slave?	M/S
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.9 acqcmd seek

Command	<b>acqcmd seek &lt;whence&gt; &lt;samples&gt;</b>  <b>&lt;whence&gt; : { start   end   current   trig }</b>  <b>&lt;samples&gt; : offset to move, in samples</b>
Description	Cause a seek on data read  NB: this command is valid only for the duration of a path - ie a path to the device should be obtained using open() or fopen(), R+W mode, the command must be sent and the data obtained using the same path handle.
Master/Slave?	M/S
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.10 acqcmd setInternalClock

Command	<b>acqcmd setInternalClock &lt;rate&gt; [DOx]</b>  Rate is integer hertz. 0 => implies use external clock  Optional - output clock on line DOx { DO0.. DO5]
Description	Forces use of internal clock.
Master/Slave?	M/S
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.11 acqcmd setExternalClock

Command	<b>acqcmd setExternalClock &lt;DIx&gt; [div DOx]</b>  Accept incoming clock on line DIx {DI0..DI5}  Optional - output clock divided by <div> on line DOx { DO0.. DO5]
Description	Forces use of external clock.
Master/Slave?	M/S
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.12 acqcmd getInternalClock

Command	<b>acqcmd getInternalClock</b> Rate is integer hertz
Description	Returns actual internal clock setting
Master/Slave ?	M/S
OK Response	ACQ32: <rate>
ERROR Response	ACQ32: ERROR error string

## 4.7.5.13 acqcmd setSampleTagging

Command	<b>acqcmd setSampleTagging &lt;on&gt;</b> <on> : { 0 = turn off, 1 = turn on }
Description	Embeds sample tag in data stream..  The structure of the sample tag is defined in XXX
Master/Slave?	M/S
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.14 acqcmd bigdump

Command	<b>acqcmd bigdump</b>
Description	Dumps contents of sample memory to host dmabuffer  outputs dump length and time
Master/Slave ?	M/S
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string
Notes	<b>bigdump</b> data is delivered to host memory in physical channel order; this has issues for client apps only if they access the data via memory mapping. [Section 4.18.1]

Please note that bigdump data is delivered to host memory in physical channel order; this has issues for client apps only if they access the data via memory mapping.

## 4.7.5.15 acqcmd setChannelMask

Command	<b>acqcmd setChannelMask &lt;channel-mask&gt;</b>  <b>&lt;channel-mask&gt;: [1 0]{maxchannels}</b>  <b>1: enable, 0: disable (mask)</b>
Description	Set the channel mask for the device  Please NOTE: channels must always be masked in pairs  ie: if channel 1 is enabled, channel 2 is as well.  If the length of the string is shorter than the number of channels, then the rest are masked.
Master/Slave?	M
OK Response	ACQ32: 11111111000000000000000000000000 (for example)
ERROR Response	ACQ32: ERROR error string
Notes	Channel Masking in ACQ16PCI is hardware assisted [Section

## 4.7.5.16 acqcmd getChannelMask

Command	<b>acqcmd getChannelMask</b>
Description	Returns the channel mask for the device
Master/Slave?	M
OK Response	ACQ32: 11111111000000000000000000000000 (for example)
ERROR Response	ACQ32: ERROR error string

## 4.7.5.17 acqcmd getMode

Command	<b>acqcmd getMode</b>
Description	Returns the current Mode if set
Master/Slave?	M
OK Respons	ACQ32: MODE (blank if MODEs not operational)
ERROR Response	ACQ32: ERROR error string

## 4.7.5.18 acqcmd setCal

Command	<b>acqcmd setCal &lt;n&gt;</b>  select calibration set <n> { 0..7}
Description	
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.19 acqcmd reserveAO

Command	<b>acqcmd reserveAO &lt;n-samples&gt;</b>  reserve space for AO storage
Description	
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.20 acqcmd setUserLed

Command	<b>acqcmd setUserLed &lt;led-num&gt; &lt;mode&gt;</b>  <led-num>: { 3 or 4 } (matches artwork on front panel)  <mode> : OFF, ON, FLASH
Description	Compact PCI only - user control of Leds 3 & 4 (not on PCI)
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.21 acqcmd getVoltsRange

Command	<b>acqcmd getVoltsRange</b>  outputs voltage ranges for AI, AO (if fitted)
Description	
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.22 acqcmd getAvailableChannels

Command	<b>acqcmd getAvailableChannels</b>  outputs number of channels for AI, AO (if fitted)
Description	
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.23 acqcmd setChannelThreshold

Command	<p><b>acqcmd setChannelThreshold &lt;channel&gt; &lt;tr-above&gt; &lt;tr-below&gt;</b></p> <p>Use in conjunction with BILEVEL detector (see next)</p> <p>&lt;channel&gt; : {1..32,X} - channel to detect trigger on, X := ALL</p> <p>&lt;tr-above&gt; : trigger if value is above this number (bits)</p> <p>&lt;tr-below&gt; : trigger if value is below this number (bits)</p>
Command 2	<p>tip: for negative numbers try</p> <p>acqcmd -- setChannelThreshold 12 5 -5</p>
Command 3	<p><b>acqcmd setChannelThreshold &lt;channel&gt; 0 &lt;rising-th&gt;</b></p> <p>Use in conjunction with EDGE detector (see next)</p> <p>&lt;rising-th&gt; : threshold in bits</p>
Notes	<p><b>acqcmd setChannelThreshold &lt;channel&gt; &lt;falling-th&gt; 0</b></p> <p>Use in conjunction with EDGE detector (see next)</p>
Pre-Requisite	<p>&lt;falling-th&gt; : threshold in bits</p> <p>In addition, it is possible to specify the thresholds in Volts eg:</p> <p>acqcmd -- setChannelThreshold 5 3.14V -2.76V</p> <p>An once-off execution of the following commands is required.</p> <p>getAvailableChannels, getVoltsRange.</p>
Description	Set threshold trigger value channel (s)
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.24 acqcmd selectThresholdDetector

Command	<b>acqcmd selectThresholdDetector [det]</b>  <b>acqcmd selectThresholdDistributor [distributor]</b>  <b>det: [BILEVEL], BILEVEL_MULTI, EDGE, EDGE_MULTI</b>
Description	Selects threshold detection. Needs to be set before every setArm  the second form is deprecated.
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.7.5.25 acqcmd clearThresholds

Command	<b>acqcmd clearThresholds</b>
Description	Clear all threshold trigger values
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.8 Diagnostics

NB: do not attempt to run these commands when not in idle state.

## 4.8.0.1 acqcmd getFwrev

Command	<b>acqcmd getFwrev</b>
Description	Outputs current firmware revision, build time, build number
Master/Slave?	M/S
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

### 4.8.0.2 acqcmd getConfig

Command	<b>acqcmd getConfig</b>
Description	Outputs current LCA configuration, showing board flavour, and extended memory status
Master/Slave ?	M/S
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

### 4.8.0.3 acqcmd getCalinfo

Command	<b>acqcmd getCalinfo</b>
Description	Outputs information about the calibration set loaded on the board
Master/Slave ?	M/S
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

### 4.8.0.4 acqcmd getCaptureStats

Command	<b>acqcmd getCaptureStats</b>
Description	Outputs interesting information about the previous capture
Master/Slave ?	M/S
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 4.9 API Level Interface

A linkable library is provided to aid the development of applications running locally on the dt100 system.

Comprises a number of 'C' language functions with the following interface:

```
int dt100_GetNumChannels()
// returns number of channels available in rack {32,64,96}

enum STATE { ST_STOP, ST_ARM, ST_RUN };

enum STATE dt100_GetState();
// returns current state

int dt100_SetChannelMask( short* channels, int maxchannels );
// <channels>: a user supplied array indicating channels to use
// <channels>[0] == 1 means "enable channel 0
// <channels>[0] == 0 means "disable channel 0
```

```

// <maxchannels>: number of elements { 32, 64, 96 }
// returns 0 or negative error code

int dt100_GetMemoryDepth( int channel );
// <channel> selects channel in range { 0..GetNumChannels()-1}
// returns depth of memory in samples for <channel>,
// Function of available memory depth on the board where
// <channel> is located and the current channel mask
// Eg - first board has 64MB memory:
// <channel>,      enabled channels,      dt100_GetMemoryDepth (0)
// 0              32, 0 is enabled,      1M
// 0              8, 0 is enabled,      4M
// 0              32, 0 NOT enabled,     0

enum MODE {
M_GATED_TRANSIENT,
M_GATED_CONTINUOUS,
M_SOFT_TRANSIENT
};

int dt100_SetMode( MODE mode, int samples );
// <mode>          <samples>
// M_GATED_TRANSIENT - is ignored
// M_GATED_CONTINUOUS - POST-SAMPLES
// M_SOFT_TRANSIENT - NUM-SAMPLES { 0=>fill memory }

int dt100_SetArmed(); // request transition to ST_ARM
int dt100_SetAbort(); // request transition to ST_STOP

int dt100_GetNumSamples()
// returns number of samples available
// this is the number of clocks that have been captured.
// the total amount of data available is
// GetNumSamples * enabled channels

int dt100_GetChannelData(
    unsigned short* buf, // caller's buffer
    int chan,           // channel number
    int n0,             // the first sample to transfer
    int ninc,           // increment (stride)
    int nmax            // the max # samples to copy
);
// returns # transferred samples or negative error
// in GATED_CONTINUOUS mode,
// n0 == 0 represents the point of trigger (GATE going OFF)
// examples
// <n0>, <ninc>, <nmax>
// 0, 1, 100 get first 100 samples from trigger
// -100, 1, 200 get 200 samples spanning trigger
// -100, 1, 100 get 100 samples before trigger
// 0, -1, 100 get 100 samples before trigger,
// in reverse order ???!
```

## 4.10 Remote Interface

A server daemon runs on the dt100 box, listening for socket connections on port 0xd100. The server daemon is multi threaded, and is able to accept multiple non conflicting sockets. The server makes a

single line status report on opening the socket connection.

Each socket represents a conversation with a logical device, so all of the protocol defined in [Device Level Interface](#) still applies. In addition a number of remote level commands are added for session control and to enable device selection

It is highly recommended that separate sockets be opened concurrently for control (to master device) and for data (from data device). For streaming data, the data socket should be discarded after use, and a new socket reopened when streaming is restarted.

## 4.10.1 Remote Commands

### 4.10.1.1 dt100 getBoards

Command	<b>dt100 getBoards</b>
Description	Get the current configuration
Master/Slave?	na
OK Response	DT100: <boards>  <boards> : board configuration records, one line per board
ERROR Response	DT100: ERROR error string

### 4.10.1.2 dt100 open master

Command	<b>dt100 open master / dev/acq32/acq32.&lt;board&gt;.m&lt;slaves&gt;</b>  <board> : { 1,2,3,4 }  <slaves> : { bbb } eg 1234
Description	Open channel to selected device
Master/Slave?	M
OK Response	DT100:
ERROR Response	DT100: ERROR device in use

### 4.10.1.3 dt100 open slave

Command	<b>dt100 open slave /dev/acq32/acq32.&lt;board&gt;</b>  <board> : { 1,2,3,4 }
Description	Open channel to selected device. Valid for queries only
Master/Slave?	S
OK Response	DT100:
ERROR Response	DT100: ERROR

## 4.10.1.4 dt100 open data

Command	<b>dt100 open data / dev/acq32/acq32.&lt;board&gt;.&lt;channel&gt;</b>  <board> : { 1,2,3,4 }  <channel> : { nn, XX } eg 01.
Description	Open data channel to selected device
Master/Slave?	D
OK Response	DT100:
ERROR Response	DT100: ERROR device in use

## 4.10.1.5 dt100 read

Command	<b>dt100 read &lt;start&gt;,&lt;stop&gt;, &lt;stride&gt;</b>
Description	Open data channel to selected device
Master/Slave?	D
OK Response	DT100: <nn> bytes followed by  Stream of data in RAW BINARY format.  The data will be framed if setTagging has been set on
ERROR Response	DT100: ERROR device in use

## 4.10.1.6 dt100 stream

Command	<b>dt100 stream</b> [<stride>] [<mean>] [npairs] [burst]  <stride> : { 1..1000, default 1 (full rate) }  <mean> : { 0 = no-mean, 1= mean of stride samples }  <npairs> : number of channel pairs (from 1 <sup>st</sup> channel)  <burst> : non-zero: take blocks of data at full rate (experimental)
Description	Stream data taking every <stride> sample  { default: <stride>=1 (full rate) }
Master/Slave?	D
OK Response	Stream of data in selected format.  NB: restricted to BINARY only at this time.
ERROR Response	DT100: ERROR device in use
Notes	This is Subrate Streaming. Max data rate is restricted.  Sample Tagging must be enabled

## 4.10.1.7 bye

Command	<b>bye</b>
Description	Close any open channels and drop the socket
Master/Slave?	M/S/D
OK Response	DT100:
ERROR Response	DT100: ERROR

## 4.10.1.8 acqcmd

Command	<b>acqcmd</b> <device level interface command>
Description	All device level commands on selected device
Master/Slave?	M
OK Response	DT100:
ERROR Response	DT100: ERROR error string

## 4.11 Remote Client

An example remote gui client <dt100rc> is provided to allow remote configuration, control and viewing of data.

dt100rc is implemented as a Java 1.4 application and so will run unchanged on a number of platforms, including Windows and Linux.

An example remote console client <dt100console> is provided to allow manual or scripted test of the commands.

Library stubs are provided to enable development of further remote applications.

## 4.12 Lan Networking

4.4.1. Each BSU has a unique LAN IP address.

4.4.2. Each BSU shall supply the normal UNIX facilities eg rlogin, telnet, ftp, nfs client, nfs server.

4.4.3. Suitable user accounts to be set up as part of the supply.

2G boards are capable of running as LAN based nodes in their own right. The primary means of control shall be via the dt100d port (0xd100, 53504). Web based diagnostic and test scripting is available at port 80.

## 4.13 Subrate Streaming

Subrate Streaming is the process of delivering subrate data via the LAN connection, optionally while acquiring full rate data to local sample memory.

The Subrate streaming facility is enabled on LAN only via the dt100 stream command.

Subrate data is delivered as a raw binary cross-section

ie data is ordered by [sample][channel], channel order is the raw device order.

2G products provide a mapping command so that external clients are able to derive their own lookup table to make the physical to logical channel mapping.

The Subrate data is also framed by a simple 64 sample framing structure. The subrate data may be free running, or, on 2G it may be synchronized to an external repeating event. In the synchronized mode, a second layer framing structure applies to allow multi frame structure in a burst. Multi frames may be up to 256 frames long, for a maximum burst length of 16384 samples. Client software then has control of Burst Delay (samples) and Burst Length( frames).

### 4.13.1 Initiating subrate streaming.

For synchronized streaming, set up an event routing, and event definition.

Set up Burst Length, Burst Delay, set Sample Tagging on.

Invoke a continuous capture mode. (eg SOFT\_CONTINUOUS)

Start streaming with dt100 stream.

### 4.13.2 Single Frame (SF) Structure

The Single frame is a 64 sample frame, with the frame word defined as a “sample tag”

When Sample Tagging is enabled, a Sample Tag is prepended to each sample of data from the cross-section device.

The Sample tag is 16 bits wide to minimize the per sample data overhead and uses a 64 sample telemetry style frame to allow all necessary data to be encoded.

Information in the Sample Tag Frame is:

- Sample Number : n0 .. n47 : 48 bits - no possibility of overflow in any practical experiment, even with higher data rates.
- Trigger Bit : present in every tag - to identify exact moment of trigger
- DIO : 16 bits to show DIO state, 8 bits per sample - ie DIO is available at half sample rate.
- Subframe number : 6 bits, 0..63
- Reserved bits : x0..x15

The format of the Sample Tag takes a telemetry style frames structure:

<i>Sample</i>	<i>d15</i>	<i>d14</i>	<i>d13..d8</i>	<i>d7..d0</i>
0 n0	T	000000	0xfe	
1 n1	T	000001	0xed	
2 n2, 4, 6,	T	00xxx0	DIO 0..7	
3 n3, 5, 7	T	00xxx1	DIO 8..15	
47 n47	T	101111	DIO 0..7	
48 x0	T	110000	DIO 8..15	
49 x1	T		DIO 0..7	
63 x15	T	111111	DIO 8..15	

The advantage of this coding is that the overhead per frame is equivalent to one additional channel versus the overhead of four channels for a naïve encoding.

Sample Number is the FULL RATE sample number.

### 4.13.3 Multi Frame Structure

The Multi Frame structure builds another layer on the basic single frame by redefining the high data rate DIOx as follows:

<i>Sample</i>	<i>d15</i>	<i>d14</i>	<i>d13..d8</i>	<i>d7..d0</i>	<i>Description</i>
00	n00	T	000000	0xfe	SF0
01	n01	T	000001	0xed	SF1
02	n02	T	000010	0xf0	MF2
03	n03	T	000011	0x01	MF3
04	n04	T	000100	FNa	Frame Number {15:0}
05	n05	T	000101	FNb	
06	n06	T	000110	Ja	Jiffies {31:0}
07	n07	T	000111	Jb	
08	n08	T	001000	Jc	
09	n09	T	001001	Jd	
10	n10	T	001010	Esa 0xaa	Event Signature {31:0}
11	n11	T	001011	Esb 0x55	
12	n12	T	001100	ESc	
13	n13	T	001101	ESd	
14	n14	T	001110	ESOFa	Event 2 SampleOffset {24:0}
15	n15	T	001111	ESOFb	
16	n16	T	010000	ESOFc	
17	n17	T	010001	DIO	DIO {7:0}
18	n18	T	010010	EDIOa	Extended DIO {31:0}
19	n19	T	010011	EDIOb	
20	n20	T	010100	EDIOc	
21	n21	T	010101	EDIOD	
22	n22	T	010110	BLENa	Burst LENgth (frames){15:0}
23	n23	T	010111	BLENb	
24	n24	T	011000	BDELa	Burst DELay (samples){15:0}
25	n25	T	011001	BDELb	
26	n26	T	011010	OVER	Overrun count
27	n27	T	011011		
28	n28	T	011100	MFNa	MultiFrame num {31:0}
29	n29	T	011101	MFNb	
30	n30	T	011110	MFNb	
31	n31	T	011111	MFNb	
32	n32	T	100000	Filla	0xf1 {all evens to end}
33	n33	T	100001	Fillb	0x11 {all odds to end}
...					
47	n47	T	101111	Fillb	
48	x00	T	110000	Filla	x00 : E_TRIGGER_TOO_FAST
49	x01	T	110001	Fillb	x01
...					
63	x15	T	111111	Fillb	

## 4.14 Binary Data Format

### 4.14.1 Channel Device

When reading from a single channel device, the data is presented as a stream of raw 16 bit little endian data. This data is in two's complement format:

<b>Coding</b>	<b>ACQ32 AI Voltage</b>
0x8000	-10.000V
0x0000	0.000V
0x7FFF	+10.000V

By contrast the ACQ32 AO voltage coding is Offset Binary:

<b>Coding</b>	<b>ACQ32 AOVoltage</b>
0x0000	-10.000V
0x8000	0.000V
0xFFFF	+10.000V

ACQ16 boards present 14 bit data left justified in a 16 bit field.

<b>Coding</b>	<b>ACQ16 AI Voltage</b>
0x8000	-2.50V
0x0000	0.00V
0x7FFF	+2.50V

### 4.14.2 Cross Section Device

When reading from the cross-section device, the data for all unmasked channels is appears for each sample in turn.

## 4.15 Accessing Data

Data is available in channelised or cross section from from the data devices. Four methods are provided for accessing data

## 4.16 Read from Device

This is the default method for getting data off the device. It is a *host-pull* method, where the host computer pulls the data from acq32 memory. Because *host-pull* uses relatively inefficient single word bus accesses, the achievable data rate is nowhere near the pci bus bandwidth; however the method is the most simple to use and the speed is adequate for many applications.

In addition, formatted data may be obtained in this way. Please note that formatted data is only recommended for relatively short data sets.

example 5.2.1:

```
cat /dev/acq32/acq32.2.03 >mydata/board2/ch03
```

example 5.2.2:

```
acqcmd format volts
cat /dev/acq32/acq32.3.XX | more
```

## 4.17 Programmed read from device

The remote control and API read methods make read calls on the data devices.

Example 5.2.1:

```
dt100 read <start>,<stop>, <stride>
```

Example 5.2.2:

```
dt100_GetChannelData
```

## 4.18 Bigdump DMA

*Please note: bigdump is now deprecated in favour of the more flexible DUMDMA; but it will be retained as an operating mode as it does have the highest performance.*

A high performance "bigdump" DMA mode of operation is available, where the entire contents of Sample Memory are uploaded to host memory in a single high speed transfer - *acq32-push*. The samples can then be mapped into application space; Applications can reference the data as an array [NSAMPLES] [NCHANNELS].

The Linux kernel must be preconfigured to allocate a large buffer to DMA.

Typically the buffer is 64MB to allow the dump of one acq32 buffer at a time; One block per acq32 may be allocated to allow the data from multiple acq32's to be dumped to host memory concurrently.

After capture, the upload is initiated using the "bigdump" command.

Application code should then use **mmap() 2** to map memory from the device node / **dev/acq32/acq32.<b>.host**.

Alternatively, after a bigdump, **read() 2** calls on data devices acq32.N.cc and acq32.N.XX will copy data from the local buffer rather than from the acq32 buffer, thus achieving a performance boost as well.

Sample code is available to demonstrate this action.

While bigdump achieves the maximum possible data rate on the pci bus, there are a number of issues that have to be taken into account:

- the buffer has to be preallocated at boot time
- there is a 2sec latency before any data becomes available,
- operation with multiple boards has to either be sequential, or a discrete buffer performanceboard has to be allocated at boot time.
- the channelised data is not delivered in logical channel order, but in a raw channel order - channel data should not be read from the regular channel devices, but from special "L" device nodes.

### 4.18.1 Raw Channel Order

For ACQ32[C]PCI Only, physical channel order does not match logical channel order.

Most of the time, the device driver and bus interface can compensate for this, but for the special case of **bigdump** (and **htstream**), if data is viewed via direct host side memory mapping, this data appears in physical channel order, which is as follows:

```

index in bigdump physical memory [00] logical channel 00 acq32.N.01
index in bigdump physical memory [01] logical channel 01 acq32.N.02
index in bigdump physical memory [02] logical channel 08 acq32.N.09
index in bigdump physical memory [03] logical channel 09 acq32.N.10
index in bigdump physical memory [04] logical channel 16 acq32.N.17
index in bigdump physical memory [05] logical channel 17 acq32.N.18
index in bigdump physical memory [06] logical channel 24 acq32.N.25
index in bigdump physical memory [07] logical channel 25 acq32.N.26
index in bigdump physical memory [08] logical channel 02 acq32.N.03
index in bigdump physical memory [09] logical channel 03 acq32.N.04
index in bigdump physical memory [10] logical channel 10 acq32.N.11
index in bigdump physical memory [11] logical channel 11 acq32.N.12
index in bigdump physical memory [12] logical channel 18 acq32.N.19
index in bigdump physical memory [13] logical channel 19 acq32.N.20
index in bigdump physical memory [14] logical channel 26 acq32.N.27
index in bigdump physical memory [15] logical channel 27 acq32.N.28
index in bigdump physical memory [16] logical channel 04 acq32.N.05
index in bigdump physical memory [17] logical channel 05 acq32.N.06
index in bigdump physical memory [18] logical channel 12 acq32.N.13
index in bigdump physical memory [19] logical channel 13 acq32.N.14
index in bigdump physical memory [20] logical channel 20 acq32.N.21
index in bigdump physical memory [21] logical channel 21 acq32.N.22
index in bigdump physical memory [22] logical channel 28 acq32.N.29
index in bigdump physical memory [23] logical channel 29 acq32.N.30
index in bigdump physical memory [24] logical channel 06 acq32.N.07
index in bigdump physical memory [25] logical channel 07 acq32.N.08
index in bigdump physical memory [26] logical channel 14 acq32.N.15
index in bigdump physical memory [27] logical channel 15 acq32.N.16
index in bigdump physical memory [28] logical channel 22 acq32.N.23
index in bigdump physical memory [29] logical channel 23 acq32.N.24
index in bigdump physical memory [30] logical channel 30 acq32.N.31
index in bigdump physical memory [31] logical channel 31 acq32.N.32

```

### 4.19 Dynamic User Mapped DMA - DUMDMA

This is a high performance *acq32-push* method of uploading data. It is considerably faster than the *host-pull* PIO, while using less CPU cycles. It does have a lower throughput than **bigdump**, but it does not suffer from the disadvantages of **bigdump**, namely the pre-assignment of a large buffer and the initial latency waiting for data. The main disadvantage of this method is that a custom application must be written to get at the data.

The sequence of events in DUMDMA are as follows:

1. Application uses **mmap ( ) 2** on a channel device. The driver dynamically allocates kernel space memory and maps it into the application's address space. The kernel space memory is allocated in 128K maximum size chunks, but this is mapped to the application as a single linear buffer. The maximum buffer that may be allocated is 2 mega samples, 4MB.

2. Application makes an **ACQ32\_IOREAD\_LOCALBUF ioctl() 2**. The driver blocks the application, sends a data request packet to the acq32's incoming i2o message buffer, and waits for interrupts.
3. The acq32 marshals and DMA's the required data direct to the host computer buffers, sending an interrupt for each buffer.
4. On receipt of data, the driver can wake the application, which can then access the data directly in its own memory space.
5. When the application is finished with the data buffer, it calls `munmap()` 2, this makes the driver deallocate the buffer.

The driver has two blocking policies available:

1. block the application until ALL the data is available - this still offers a latency improvement over bigdump, since only the data for one channel has to be uploaded, as opposed to all the channels. The situation may be further improved by subranging and/or subsampling the data request.
2. The `ioctl()` can be set to be non-blocking. In this case, if the application attempts to access a page of data that has not yet been filled, the driver is in control of the demand paging mechanism of the kernel and forces the application to block until the data is available. This has the effect of making initial data ready for use with very low latency, while still continuing to fill the buffer in the background, while being totally transparent to the application.

A sample utility, `acq32fetch` has been provided to show the capability of DUMDMA. The program may be used directly in a system to output Value data to a file or pipe, but the best performance will be achieved by using this program as a base for numeric processing directly on the application mapped memory buffer.

## 4.20 Typical timing comparison for data upload methods

#1 DUMDMA

```
[dt100@disklessdtacq dt100]$ timex acq32fetch -o /dev/null 33 0x100000
user:0.00 s system:0.00 s elapsed:0:00.42 s CPU 0%
```

Time for 32 channels < 14s

#2 PIO Fetch

```
[dt100@disklessdtacq dt100]$ timex acq32fetch -o /dev/null 33 0x100000
user:0.00 s system:0.00 s elapsed:0:00.42 s CPU 0%
</acq32/acq32.2.01 of=/dev/null bs=1024 count=2048
2048+0 records in
2048+0 records out
user:0.41 s system:0.00 s elapsed:0:00.80 s CPU 50%
```

Time for 32 channels < 26s

#3 Bigdump

```
[dt100@disklessdtacq dt100]$ timex acqcmd -b 2 bigdump
ACQ32:BIGDUMP moved 0x04000000 bytes in 3443 msec OK
```

```
user:0.00 s system:0.00 s elapsed:0:03.45 s CPU 0%
```

```
[dt100@disklessdtacq dt100]$ timex dd if=/dev/acq32/acq32.2.01  
of=/dev/null b>  
2048+0 records in  
2048+0 records out  
user:0.01 s system:0.28 s elapsed:0:00.29 s CPU 100%
```

```
Time for 32 channels is 3.4 + 32 * 0.29 = < 13s
```

## 5 Enhanced Software Operation

(Also known as Generalize Phase Event Mode – GPEM)

The enhanced software allows finer grained control of operation than do the normal modes, while retaining full backwards compatibility.

In addition, where fitted on the board, Analog Output and Digital Output waveform Functions are enabled. Operation of the output waveforms mirrors that of the Analog Input Function.

### 5.1 Summary

Two concepts :- *Phase, Event* are introduced at a lower level than the existing Modes.

The existing Modes will continue to be supported, and can be thought of as fixed combinations of the lower level settings.

All captures proceed in two Phases:

- Phase 1 : capture to cyclic buffer of user defined length; zero length is permissible
- Phase 2 : follows on from Phase 1, one-shot capture to linear buffer of user defined length.

Each Phase can be independently in one of the three States - ST\_ARM, ST\_RUN, ST\_STOP.

The two Phases are bounded by three user-definable Events:

- Event 1 : after the system has been ARMED, this Event starts Phase 1 capture.
- Event 2: terminates Phase 1 and starts Phase 2.
- Event 3: terminates Phase 2.

The events are an OR-ing of a number of possible conditions, including a "true" value, making a non-event.

### 5.2 State Table

<i>Event</i>	<i>Phase 1</i>	<i>Phase 2</i>
( setArm)	ST_ARM	ST_ARM
Event 1	ST_RUN	ST_ARM
Event 2	ST_STOP	ST_RUN
Event 2	ST_STOP	ST_STOP

## 5.3 Commands

The following Device Level commands are proposed to allow user control:

<i>Modifier</i>	<i>Choices</i>	<i>Default</i>	<i>Description</i>
[f]	{ <b>AI, AO, DO</b> }	AI	Function
<p>	{ <b>P1, P2</b> }		Phase
<e>	{ <b>E1, E2</b> }		Event
<n>	{ 0..64M }		Number of samples
<d>	{ <b>DI</b> {0..5}		Digital Input Line

### 5.3.1 Command Listing

#### 5.3.1.1 acqcmd setPhase

Command	<b>acqcmd setPhase [f] &lt;p&gt; &lt;n&gt;</b>
Description	Sets the number of samples requested for Function [f], Phase <p>. The acq32 will satisfy the request up to the limit of memory and previous phase settings.  It is necessary to send BOTH commands setPhase 1 followed by setPhase 2; setPhase 1 clears both Phase 1 and Phase 2 settings, for the given Function
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

#### 5.3.1.2 acqcmd getPhase

Command	<b>acqcmd getPhase [f] &lt;p&gt; requested-samples</b>
Description	Gets the number of samples allocated for Phase <p>.
Master/Slave?	M
OK Response	ACQ32: <n>
ERROR Response	ACQ32: ERROR error string

Command	<b>acqcmd getPhase [f] &lt;p&gt; actual-samples</b>
Description	Gets the number of samples acquired so far for Phase <p>
Master/Slave?	M
OK Response	ACQ32: <n>
ERROR Response	ACQ32: ERROR error string

Command	<b>acqcmd getPhase [f] &lt;p&gt; state</b>
Description	Gets the current state of Phase <p>
Master/Slave?	M
OK Response	ACQ32: ST_ARM, ST_RUN, ST_STOP
ERROR Response	ACQ32: ERROR error string

### 5.3.1.3 acqcmd setEvent

Command	<b>acqcmd setEvent [f] &lt;e&gt; &lt;or-condition&gt;</b>
Description	<p>Defines the Event condition for Function [f], Event &lt;e&gt;. <b>&lt;or-condition&gt;</b> is one of the keywords below, together with parameters as required.</p> <p>Only one <b>&lt;or-condition&gt;</b> may be specified per command, however multiple or conditions may be accumulated over multiple commands. The list of or conditions can be cleared by using the <b>EV_NONE</b> condition.</p> <p><b>EV_TRUE</b> - the event fires immediately</p> <p><b>EV_SOFT</b> - the event is fired by software "fireEvent" command *</p> <p><b>EV_TRIGGER_RISING &lt;d&gt;</b> - the event is a rising edge of the external trigger signal</p> <p><b>EV_TRIGGER_FALLING &lt;d&gt;</b> - the event is a falling edge of the external trigger signal</p> <p><b>EV_NONE</b> - clears all conditions for this event</p> <p>future extensions include:</p> <p><b>EV_DATA_EXCEEDS &lt;channel&gt; &lt;value&gt;</b></p> <p><b>EV_DATA_BELOW &lt;channel&gt; &lt;value&gt;</b></p> <p>implicit in the definition for Event 3 is "Phase 2 samples reached".</p> <p>Multiple conditions for a Event may be specified either in a single command or in multiple separate commands</p> <p>* it is not necessary to specify <b>EV_SOFT</b> as it is always available</p>
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

### 5.3.1.4 acqcmd getEvent

Command	<b>acqcmd getEvent [f] &lt;e&gt;</b>
Description	Returns the condition set for Event <e>
Master/Slave?	M
OK Response	ACQ32: or-conditions
ERROR Response	ACQ32: ERROR error string

### 5.3.1.5 acqcmd fireEvent

Command	<b>acqcmd fireEvent [f] &lt;e&gt;</b>
Description	Fires the given event - triggering the event in the acquisition system if the capture is waiting on that event AND EV_SOFT has been selected for the event.
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

### 5.3.1.6 acqcmd setClock

Command	<b>acqcmd setClock [f] { &lt;d&gt;, INTERNAL_CLOCK }</b>
Description	Selects clock source for [f]  nb - the traditional command acqcmd setInternalClock <freq> must be specified first to set up the internal clock.  BEWARE: setInternalClock <freq> implicitly performs setClock AI INTERNAL_CLOCK, so for the case where INTERNAL_CLOCK is required, but not for AI, the clock frequency must be set first followed by the explicit AI external clock selection.
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR error string

## 5.4 Use of Existing Modes

The existing modes will still be available as before. They can be thought of as macros combining phases and events as follows:

<i>Mode</i>	<i>Event 1</i>	<i>Phase 1</i>	<i>Event 2</i>	<i>Phase 2</i>	<i>Event 3</i>
SOFT_TRANSIENT	EV_TRUE	0	EV_TRUE	Nsamples	(nsamples)
GATED_TRANSIENT	EV_TRUE	0	EV_TRIGGER_FALLING	Nsamples	(nsamples) OR EV_TRIGGER_RISING
GATED_CONTINUOUS	EV_TRIGGER_FALLING	Memory Length - POST_SAMPLES	EV_TRIGGER_RISING	POST_SAMPLES	(post_samples)
SOFT_CONTINUOUS	EV_TRUE	Memory Length	-	-	-

So it is possible to replicate the original Modes by using Phase and Event commands, along with many other possibilities.

#### 5.4.1 Example -

Capture with soft start, pretrigger, trigger on edge and run to completion

Command	Comment
<b>acqcmd setPhase 1 32768</b>	Pre-trigger 32768 samples
<b>acqcmd setPhase 2 2650000000000000</b>	Post trigger ridiculous number
<b>acqcmd getPhase 2 requested-samples</b>  ACQ32: 999000	Find out the actual Post trigger length
<b>acqcmd setEvent 1 TRUE</b>	Straight thru on ARM
<b>acqcmd setEvent 2 EV_TRIGGER_FALLING</b>	This is "the trigger point"
<b>acqcmd setEvent 3 EV_NONE</b>	Use default "Phase 2 Samples reached"
<b>acqcmd setArm</b>	Arm and go ..
<b>acqcmd getPhase 1 state</b>  ACQ32: ST_RUN	Phase 1 is running ...
<b>acqcmd getPhase 2 state</b>  ACQ23: ST_ARM	Phase 2 is ready to go
<b>acqcmd getPhase 1 actual-samples</b>  ACQ32: 1024	Poll # samples ....

For further examples of use of GPPEM, please refer to the simple setup scripts in the distribution in ~dt100/GPEM.

In particular, please refer to GPPEM/reset for a reliable reset sequence.

## 5.5 Loading data for Output Waveform Generation

### 5.5.1 Analog Output AO

The analog output data comprises a vector of  $\langle n1 \rangle + \langle n2 \rangle$  32 bit samples.

Data format comprises 2 16 bit offset binary values, AO0, AO1. (for coding, see 4.14)

The values are merged into a single 32 bit word for loading.

The data should be written to the **acq32 . <b> .AOf** device prior to arming the board, but after declaring the phase definition (using **setPhase AO P1 <n1>**, **setPhase AO P2 <n2>** declarations ).

It is the application's responsibility to ensure that sufficient data is supplied.

Data beyond  $\langle n1 \rangle + \langle n2 \rangle$  shall be ignored.

### 5.5.2 Digital Output DO

The digital output data is supplied as a series of 32 bit binary values comprising a 24 bit clock count followed by an 8 bit output state value.

The data should be written to the **acq32 . <b> .DOf**, and the same considerations apply as per AO.

## 6 Appendix: Command Interface Signal Conditioning SC-1

The external IO signals for this unit are described in the Signal Conditioning Design Document.

A series of "acqcmd" commands are defined, for use with a previously opened Master channel.

### 6.0.0.1 acqcmd setDIO

Command	<b>acqcmd setDIO</b>
Description	Configure DIO
Master/Slave?	M
OK Response	ACQ32:  Example:  acqcmd setDIO 10-----  where 1 = Output High, 0 = Output Low, - = Input  ACQ32:
ERROR Response	ACQ32: ERROR device in use

### 6.0.0.2 acqcmd getDIO

Command	<b>acqcmd getDIO</b>
Description	monitor DIO status
Master/Slave?	M
OK Response	ACQ32: + DI_STATE DO_STATE AI_C AO_CT DO_CT  DI_STATE xxxxxxxxxxxxxxxxx : H, L status of DI lines (even if set as output)  DO_STATE : xxxxxxxxxxxxxxxxx : 1,0,- echo of setDIO  AI_C : x : Analog Input Clock status A - active, x - inactive  AO_CT: xx : Analog Output Clock/Trigger status  DO_CT: xx : Digital Output Clock/Trigger status  Example:  acqcmd getDIO  ACQ32: LHHHHHHHHHHHHHHHH -11111111111111 A xx xx
ERROR Response	ACQ32: ERROR device in use

## 6.0.0.3 acqcmd setChannel

Command	<b>acqcmd setChannel &lt;channel&gt; &lt;gain1&gt; &lt;gain2&gt; &lt;excitation&gt;</b>  <channel> : { 0..31}  <gain1> : { 1, 10, 100, 1000 }  <gain2> : { 1, 2, 4, 8 }  <excitation> : { 1, 2, 5, 10 }
Description	Configure analog channel
Master/Slave?	M
OK Response	ACQ32:
ERROR Response	ACQ32: ERROR device in use

The entire setup dialog may be stored on the client computer as a script and replayed directly to the server; NB - the client must

read each response before sending the next command - the console client is capable of doing this.

An example script might be: this is a comment

```
dt100 open master 1.m1
acqcmd setDIO 10-----10-----0000
acqcmd setChannelMask 111100000000000000000000000000001111
acqcmd setChannel 0 1 1 1
acqcmd setChannel 1 10 2 2
acqcmd setChannel 2 100 4 5
acqcmd setChannel 3 1000 10 10 ;# this one is gonna be NOISY!
acqcmd setChannel 28 1 1 1
acqcmd setChannel 29 1 1 1
acqcmd setChannel 30 1 1 1
acqcmd setChannel 31 1 1 1
```

## 7 Appendix: example remote scripts

### 7.1 Simple initialization and capture

NB: the script must run through a suitable filter that handshakes command transmit with receipt of previous response, such as the DtacqClient object.

```
# simpleinit.txt: simple initialisation
# java -cp dt100rc.jar DtacqClient target <simpleinit.txt
dt100 open master /dev/acq32/acq32.1.m1
# kick off an example capture
acqcmd setInternalClock 100000
acqcmd setChannelMask 11111111000000000000000000000000
# ruler 12345678901234567890123456789012
acqcmd setMode SOFT_TRANSIENT 8000
acqcmd setArm
# poll for completion
acqcmd getState
acqcmd getNumSamples
acqcmd getState
acqcmd getNumSamples
acqcmd getState
acqcmd getNumSamples
acqcmd getState
acqcmd getNumSamples
acqcmd getState
acqcmd getNumSamples
bye
```

### 7.2 Simple initialization for Gated Continuous

```
# continuous simple initialisation
# java -cp dt100rc.jar DtacqClient target <continuous.txt
dt100 open master /dev/acq32/acq32.1.m1
# kick off an example capture
acqcmd setInternalClock 100000
acqcmd setChannelMask 11111111000000000000000000000000
# ruler 12345678901234567890123456789012
acqcmd setMode GATED_CONTINUOUS 0x100000
acqcmd setArm
# poll for completion
acqcmd getState
acqcmd getNumSamples
acqcmd getState
acqcmd getNumSamples
acqcmd getState
acqcmd getNumSamples
acqcmd getState
acqcmd getNumSamples
acqcmd getState
acqcmd getNumSamples
bye
```

## 7.3 Polling Status

This is not really scriptable, but in pseudo code:

1. Open a channel to master device.
2. Loop with getState, getNumSamples until end state achieved.

Or block on read from status device.

## 7.4 Reading transient data

Not scriptable.

## 7.5 Streaming Data

Not Scriptable

## 8 Appendix: Digital Input Routing Commands for Acq32CPCI

Acq32CPCI has a very flexible routing strategy for the DI lines.

Nb - these commands have no function on acq32pci.

### 8.0.0.1 acqcmd setSyncRoute

Command	<b>acqcmd setSyncRoute &lt;d&gt; &lt;route&gt; [&lt;route&gt;]</b>
Description	Sets the route for a given signal <d> to be the OR mask of routes.  Appropriate routes are tabled below
Master/Slave?	M
OK Response	ACQ32: <n>
ERROR Response	ACQ32: ERROR error string

<i>Signal</i>	<i>Ports</i>
<b>DI0</b>	Any of { <b>MI0, MO0, J50, PXI_TRIG0, PXI_TRIG6</b> }  <b>MI0</b> Connects to front panel “CLK” LEMO and is the default external clock input.
<b>DI1</b>	Any of { <b>MI1, MO1, J51, PXI_TRIG1, PXI_TRIG7</b> }
<b>DI2</b>	Any of { <b>MI2, MO2, J52, PXI_TRIG2, PXI_STAR</b> }  <b>MI2</b> Connects to front panel “TRG” LEMO and is the default external trigger input.
<b>DI3</b>	Any of { <b>MI3, MO3, J53, PXI_TRIG3, PXI_CLK10</b> }
<b>DI4</b>	Any of { <b>MI4, MO4, J34, PXI_TRIG4</b> }
<b>DI5</b>	Any of { <b>MI5, MO5, J35, PXI_TRIG5</b> }

## 9 Appendix: ACQ16PCI Differences

ACQ16PCI works the same as ACQ32PCI, other than the following:

- Threshold triggering is available on ACQ16PCI.
- Channel Masking has Hardware Assist [Section 9.1]
- Data format is 14 bit left justified [Section 4.14]
- Enhanced Functionality (GPEM) [Section 5] is NOT SUPPORTED; It is strongly recommended that ACQ16PCI be run only in TRIGGERED\_CONTINUOUS mode.
- Subrate Streaming is NOT SUPPORTED [Section 4.10.1.6]

### 9.1 ACQ16PCI Channel Mask Hardware Assist.

Channel masking in ACQ32PCI is purely a software function.

ACQ16PCI hardware has the ability to reconfigure itself to achieve higher sample rates with lower channel counts using the same gross bandwidth. The hardware assist is switched in using the normal

setChannelMask command [Section 4.7.5.15], but for certain fixed patterns, the hardware can switch out channels to achieve higher data rates:

#	Command		Max Sample Rate *
		1234567890123456	
1	SetChannelMask	1100000000000000	2 x 10MSPS
2	SetChannelMask	1111000000000000	4 x 6MSPS
	SetChannelMask	1111110000000000	6 x 3MSPS \$
	SetChannelMask	1111111100000000	8 x 3MSPS
	SetChannelMask	1111111111000000	10 x2.5MSPS \$
	SetChannelMask	1111111111110000	12 x2.5MSPS
	SetChannelMask	1111111111111100	14 x1.8MSPS \$
	SetChannelMask	1111111111111111	16 x1.8MSPS
	SetChannelMask	xxxxxxxxxxxxxxxxxx	N x1.8MSPS \$
\$ with soft masking, may be useful to maximise use of available memory store.			
* depends on rating of convertor			

## 10 Appendix: Low Latency Functionality

### 10.1 Overview

The acq32 may be used in low latency control applications.

Acquired data can be delivered directly to host memory, giving a delay from clock to sample in memory sub 10usecs.

The ADC may be either clocked externally or soft clocked.

On board counters allow measurement of absolute time of sample, and process time.

Sampling commences on external trigger. Clock and trigger edges are software sleectable.

A decimation facility is available to subsample the data.

The external clock may be subdivided and used to provide both an onboard and off board clock facility

## 10.2 References

1. PCI bus protocol definition - acq32busprot.h
2. llcontrol control application -

## 10.3 Description of Operation of Example

### 10.3.1 External Signals:

- 1MHz clock on DI0
- AICLK on DI1
- TRIGGER on DI2

### 10.3.2 Files:

- llcontrol.c - example app top level
- llif.h - interface to device driver
- llprotocol.h - interface to acq32 bus level protocol

### 10.3.3 Operation

- Fills a TestDescription struct from command line args.
- calls mmapMbox() to map control mailboxes into application space.
- calls mmapDmaBuffer() to allocate and map a suitable DMA data buffer.
- calls runTest() to run the test.

#### **Pseudo Code for Soft Clocking**

```
llSetAddr( target_address );  
while( !llCounterRunning() )  
    ;// poll for trigger  
  
for ( nsamples ){  
    llSetCmd( LLC_CSR_M_SOFTCLOCK );  
}
```

```

    llWaitDmaDone( );
    // process the data
}

```

### Pseudo Code for External Clocking

```

    llSetAddr( target_address );
    while( !llCounterRunning() )
    ; // poll for trigger

    for ( nsamples ){
    llWaitDmaDone( );
    // process the data
    }

```

## 10.4 Examples

D-TACQ make use of octave for data processing - an example octave script is provided in ~dt100/octave

### Example 1 - Soft Clocking with delay

```

cd octave;
;# do an initial capture
acqcmd setMode SOFT_TRANSIENT 1024
acqcmd setInternalClock 100000
acqcmd setArm
;# example of Soft Clocking
;# -b 2 : use board 2
;# -t : save timing data
;# -o /tmp/lldump : output data to file /tmp/lldump
;# -n 100 : capture 100 samples
;# -A : data appends to buffer (not overwrite, need to see stats)
;# A 100 : softclock mode, delay 100 external clocks between samples
[dt100@disklessdtacq octave]$ llcontrol -b 2 -t -o /tmp/lldump -n 100
-A A 100
[dt100@disklessdtacq octave]$ octave
GNU Octave, version 2.0.14 (i586-pc-linux-gnu).
Copyright (C) 1996, 1997, 1998, 1999 John W. Eaton.
This is free software with ABSOLUTELY NO WARRANTY.
For details, type `warranty'.

octave:1> llread
*** local user variables:
prot type rows cols name
====
wd matrix 100 32 chXX
wd scalar 1 1 fd
wd matrix 100 16 stats
wd scalar 1 1 statslen
wd matrix 100 1 tinst
wd matrix 100 1 tlatch
wd matrix 100 1 tproc
wd matrix 98 1 tsample
octave:2> tsample(1:5);#this is the number of ext clocks between

```

```
samples
ans =
102
102
102
102
101
```

Example 2 - External Clock

```
[dt100@disklessdtacq dt100]$ llcontrol -b 2 -t -o /tmp/lldump -n 100
-A Bii 10#
octave:3> llread
octave:4> tsample(1:5)
ans =
10
10
10
10
9
```

# Shows 10 external clocks between samples

Example 3 - External Clock, Positive edge trigger, decimation=5

```
[dt100@disklessdtacq dt100]$ llcontrol -b 2 -t -o /tmp/lldump -d 5 -n
100 -A B>
Polling for Counter Running
Polling for Counter Running
octave:5> llread
octave:6> tsample(1:5)
ans =
```

```
49
51
50
49
51
```

# shows 10\*5 external clocks between samples

## 11 Appendix: High Throughput Streaming

A feature for streaming data to host memory is available.

Applications for this feature include captures > 128MB and real time control, or logging to disk.

HT Streaming is best suited to applications with low channel count, since use is constrained by the data rate across the PCI bus.

On ACQ32, it is possible to stream all the data from 32 channels at 250kSPS. It should be possible to operate 4 x ACQ32PCI in one PCI segment at full data rate.

On ACQ16/16, it is possible to stream all the data from 16 channels at 2.4MSPS.

HT Streaming can make use of the Interrupt Mask ioctl; However interruption is not critical provided a big enough buffer is used.

Currently HT Streaming is to a single large buffer, bigdump type buffer recommended, with a recycle option for continuous operation.

For examples of use, and detail of operation, users are referred to the source code of the htstream application.

Please note that

- Channel Mask has no effect on HTSTREAM with ACQ32PCI; with ACQ16PCI, the Physical Channel Mask can be used to good effect to reduce data rates at reduced channel count.
- Data is delivered to host memory in raw Physical Channel Order [Section 4.18.1]
- HTSTREAMING can use a local status mode to reduce PCI polling. This is recommended.

## 12 Appendix: Examples of Multiple Board Synchronisation

### 12.1 Within a Chassis

Within a chassis, multiple board will have a DIO bus mechanism - a ribbon cable for PCI, PXI bussed lines for Compact PCI.

Example showing all boards in a chassis clocked from one shared InternalClock, and using threshold trigger

```
#!/bin/bash
# falling edge - set up a rising edge trigger on Channel 1

clock=${1:-2000000}
edge=${2:-falling}
level=${3:-2000}

echo edge_trigger $clock $edge $level
export ACQCMD_DEV=/dev/acq32/acq32.m.all

acqcmd setAbort
acqcmd setExternalClock DIO

acqcmd -b 1 setInternalClock $clock D00

acqcmd -b 1 clearThresholds
acqcmd -b 1 getAvailableChannels
acqcmd -b 1 getVoltsRange

case $edge in
    falling)
        acqcmd -b 1 setChannelThreshold 1 0 $level ;#falling edge
        ;;
    rising)
        acqcmd -b 1 setChannelThreshold 1 $level 0 ;#rising edge
        ;;
    *)
        echo please specify falling or rising
        exit;;
esac

acqcmd -b 1 selectThresholdDetector EDGE_MULTII

acqcmd setModeTriggeredContinuous 100000 10000
acqcmd setArm
```

### 12.2 Between Chassis

D-TACQ supplied an external clock and trigger sync module. Contact factory for details.

## 13 Appendix: Driver ioctls() and globals parameters.

While most operations are performed using commands, the driver may be further customised by the use of load time parameters. In addition, a limited number of ioctls() are defined for finer programmatic control.

### 13.1 Driver Load time parameters.

defined in acq32.h:

```
extern int acq32_use_interrupts; //set to 1 to use interrupts
extern int acq32_simulate; // set to 1 to enable simulation
extern int acq32_gate_hi; // set to 1 for gate active high
extern int acq32_extclock; // set to 0 for internal clock
extern int acq32_debug; // 0 => no debugs
extern int acq32_command_debug; // 0=> nodebugs
extern int acq32_busprot_debug; // 0=> no debugs
extern int acq32_path_buffer_size; // default path buffer size
extern int acq32_instrument_buf_len; // set length of inst buffer, if rpd
extern long acq32_big_buf_base; // PGM:TEMP base addr of big streambuf
extern long acq32_big_buf_len;
extern long acq32_big_buf_apportion; // slice up the bigbuf
extern int acq32_dumdma_to; // DUMDMA timeout - crank up if debugging
extern int acq32_fill_vma; // debug aid - fill vma on allocate
extern int acq32_max_channels; // override to set max channels on all brds
extern int acq32_double_tap; // arm-abort-arm fills cache extern int
acq32_debounce_clk; // enable clock debounce extern int acq32_enable_emtrig_bit;
```

### 13.2 Driver ioctls().

The acq32-drv driver module now features an ioctl() to selectively mask CPU interrupts. This is so that real time applications can run with all interrupts disabled, without a custom kernel or extra device drivers.

Ioctl definitions may be found in acq32ioctl.h.

The following ioctls() may be useful to application writers:

- ACQ32\_IOREAD\_LOCALBUF: used by DUMDMA
- ACQ32\_IOREAD\_GETPHYSICAL: used by DUMDMA
- ACQ32\_IOS\_INTS\_ENABLE
- ACQ32\_IOS\_INTS\_DISABLE

“Real Time Linux” with interrupts selectively disabled.

## Table Of Commands

4.7.5.1	<a href="#"><u>acqcmd setMode</u></a> .....	18
4.7.5.2	<a href="#"><u>acqcmd setModeGatedContinuous</u></a> .....	18
4.7.5.3	<a href="#"><u>acqcmd setModeTriggeredContinuous</u></a> ....	18
4.7.5.4	<a href="#"><u>acqcmd getMode</u></a> .....	18
4.7.5.5	<a href="#"><u>acqcmd getNumChannels</u></a> .....	18
4.7.5.6	<a href="#"><u>getNumSamples</u></a> .....	19
4.7.5.7	<a href="#"><u>acqcmd getState</u></a> .....	19
4.7.5.8	<a href="#"><u>acqcmd format</u></a> .....	19
4.7.5.9	<a href="#"><u>acqcmd seek</u></a> .....	20
4.7.5.10	<a href="#"><u>acqcmd setInternalClock</u></a> .....	20
4.7.5.11	<a href="#"><u>acqcmd setExternalClock</u></a> .....	20
4.7.5.12	<a href="#"><u>acqcmd getInternalClock</u></a> .....	21
4.7.5.13	<a href="#"><u>acqcmd setSampleTagging</u></a> .....	21
4.7.5.14	<a href="#"><u>acqcmd bigdump</u></a> .....	21
4.7.5.15	<a href="#"><u>acqcmd setChannelMask</u></a> .....	22
4.7.5.16	<a href="#"><u>acqcmd getChannelMask</u></a> .....	22
4.7.5.17	<a href="#"><u>acqcmd getMode</u></a> .....	22
4.7.5.18	<a href="#"><u>acqcmd setCal</u></a> .....	23
4.7.5.19	<a href="#"><u>acqcmd reserveAO</u></a> .....	23
4.7.5.20	<a href="#"><u>acqcmd setUserLed</u></a> .....	23
4.7.5.21	<a href="#"><u>acqcmd getVoltsRange</u></a> .....	24
4.7.5.22	<a href="#"><u>acqcmd getAvailableChannels</u></a> .....	24
4.7.5.23	<a href="#"><u>acqcmd setChannelThreshold</u></a> .....	25
4.7.5.24	<a href="#"><u>acqcmd selectThresholdDetector</u></a> .....	26
4.7.5.25	<a href="#"><u>acqcmd clearThresholds</u></a> .....	26
4.8.0.1	<a href="#"><u>acqcmd getFwrev</u></a> .....	26
4.8.0.2	<a href="#"><u>acqcmd getConfig</u></a> .....	27
4.8.0.3	<a href="#"><u>acqcmd getCalinfo</u></a> .....	27
4.8.0.4	<a href="#"><u>acqcmd getCaptureStats</u></a> .....	27
4.10.1.1	<a href="#"><u>dt100 getBoards</u></a> .....	29
4.10.1.2	<a href="#"><u>dt100 open master</u></a> .....	29
4.10.1.3	<a href="#"><u>dt100 open slave</u></a> .....	29
4.10.1.4	<a href="#"><u>dt100 open data</u></a> .....	30
4.10.1.5	<a href="#"><u>dt100 read</u></a> .....	30
4.10.1.6	<a href="#"><u>dt100 stream</u></a> .....	31
4.10.1.7	<a href="#"><u>bye</u></a> .....	31
4.10.1.8	<a href="#"><u>acqcmd</u></a> .....	31
5.3.1.1	<a href="#"><u>acqcmd setPhase</u></a> .....	43
5.3.1.2	<a href="#"><u>acqcmd getPhase</u></a> .....	43
5.3.1.3	<a href="#"><u>acqcmd setEvent</u></a> .....	44
5.3.1.4	<a href="#"><u>acqcmd getEvent</u></a> .....	45
5.3.1.5	<a href="#"><u>acqcmd fireEvent</u></a> .....	45
5.3.1.6	<a href="#"><u>acqcmd setClock</u></a> .....	45
6.0.0.1	<a href="#"><u>acqcmd setDIO</u></a> .....	48
6.0.0.2	<a href="#"><u>acqcmd getDIO</u></a> .....	48
6.0.0.3	<a href="#"><u>acqcmd setChannel</u></a> .....	49
8.0.0.1	<a href="#"><u>acqcmd setSyncRoute</u></a> .....	52

END of DOCUMENT