# **PSC driver usage**

## **Build requirments:**

* libevent2-dev (>= 2.0.19)
* libfftw3-dev (>= 3.3)
* fftw is optional. In configure/CONFIG\_SITE set USE\_FFTW=NO to disable.
* If have errors when installing the libraries,

dpkg: warning: 'ldconfig' not found in PATH or not executable

dpkg: warning: 'start-stop-daemon' not found in PATH or not executable

dpkg: error: 2 expected programs not found in PATH or not executable

Add the following two lines to /etc/sudoers

Defaults env\_reset

Defaults secure\_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

## **In start script:**

**createPSC**( "sessionname", "hostname", 8765, 1 )

// extern "C" void createPSC( const char\* name,

const char\* host,

int port,

int timeout)

**dbLoadRecords**( "db/psc-ctrl.db", "P=TST-MAG{XX}, NAME=sessionname" )

Where "hostname" is the host or IP address of the PSC and 8765 is the port number to connect. The file 'psc-ctrl.db' contains standard records for connection and driver status information.

The fourth argument of createPSC is used to enable the recv data timeout. When this argument is 0 no timeout is set; when 1 a timeout of PSCInactivityTime seconds is used. If no data is received within this time the connection is closed.

The driver will also always timeout if there is data in the socket buffer for more than PSCInactivityTime seconds. This should only occur for sustained data rates in excess of what the server can accept, or if the server stops accepting data.

## **Message format**

The driver sends and receives variable length messages (aka. blocks) with an 8 bytes header followed a variable length body. The 'bodylen' field in the header gives the number of bytes in the body.

struct header

{

char A; /\* must be 'P' \*/

char B; /\* must be 'S' \*/

uint16\_t msgid;

uint32\_t bodylen; /\* in bytes \*/

};

msgid: 1 – signle write; other – waveform

**typedef** **struct** \_srvRxDatas

{

**char** idA;

**char** idB;

**short** MsgId;

Xuint32 bodyLen;

**Int** data[360];

} srvRxDatas, \*pSRV\_RX\_DATA;

**#define** pSrvRxDdrOffAddr ( (pSRV\_RX\_DATA) (DDR3\_PSCREG\_BASE) )

## **Receive**

To receive a block as an array (waveform). The block is assumed to be an array of 2's complement signed 16-bit integers. It is stored as floating point numbers to allow for scaling.

record( waveform, "$(P)wfin-I")

{

Field ( DTYP, "PSC Block I16 In" )

field ( INP , "@$(NAME) $(msgid) $(offset=0) $(step=0)" )

field ( SCAN, "I/O Intr" )

field ( FTVL, "DOUBLE" )

field ( NELM, "10" )

}

The record will be processed whenever a block with message id '$(msgid)' is received. On reception up to 20 bytes (2\*10) are stored in the record and the NORD field is set accordingly.

* Use DTYP "PSC Block I32 In" to treat as 32-bit integers.
* DTYP "PSC Block I8 In" may also be used if FTVL is CHAR or UCHAR.

The '$(offset)' and '$(step)' parameters (in bytes) allow a waveform to be extracted from a subset of a message. The offset is the first byte which will be part of the first element in the waveform. The step is the number of bytes between the first bytes of the first and second elements. The number of bytes in each element is determined by the DTYP.

To receive a block as a group of individual registers.

record ( longin, "$(P)regX-I" )

{

field ( DTYP, "PSC Reg" )

field ( INP, "@$(NAME) $(msgid) $(offset)" )

field ( SCAN, "I/O Intr" )

}

On reception of message id '$(msgid)' this record is processed. If the length of the message is less than '$(offset)+4' then the record will go into alarm (INVALID). Otherwise the bytes '$(offset)' through '$(offset)+3' are read, interpreted as a 32-bit integer, and stored in the VAL field.

* In addition to 'longin', the record types 'bi', 'mbbi', 'mbbiDirect', and 'ai' may also be used.

For the 'mbbi\*' record types the NOBT and SHFT fields are used to allow several records to operate on the same offset. For the bi record type, the MASK field may be used.

For records which receive data TSE=-2 requests that the driver assign a timestamp. The default is to assign the reception time (read after the header is parsed, before body), or by includeing 'info("TimeFromBlock","<offset")' in the record instance specifies that an 8 bytes (4 bytes posix sec, 4 bytes nanoseconds) be read from the data block.

## **Send**

Each PSC socket maintains a send buffer which is flushed on command

record ( bo, "$(P)Send-Cmd" )

{

field ( DTYP, "PSC Ctrl Send All" )

field ( DESC, "Send pending to PSC" )

field ( OUT , "@$(NAME)" )

field ( ZNAM, "Send" )

field ( ONAM, "Send" )

}

To send an array as a block include a record like:

record ( waveform, "$(P)wfout-SP" )

{

field ( DTYP, "PSC Block I16 Out" )

field ( INP , "@$(NAME) $(msgid)" )

field ( FTVL, "DOUBLE" )

field ( NELM, "10" )

field ( FLNK, "$(P)Send-Cmd" )

}

Processing this record will cause the array data to be converted to 16-bit integers, formatted into a message, and added to the send buffer. If the record is FLNK'd to "$(P)Send-Cmd" then the message will be send immediately.

If the PSC socket is not connected then the record goes into INVALID alarm.

* Use DTYP "PSC Block I32 Out" to send as 32-bit integers.
* DTYP "PSC Block I8 Out" may also be used if FTVL is CHAR or UCHAR.

To send a group of individual register values as a block. First add an entry to the start script.

setPSCSendBlockSize ( "sessionname", $(msgid), $(blocksize) )

Then create several records.

record(longout, "$(P)regX-I")

{

Field ( DTYP, "PSC Reg")

field ( OUT, "@$(NAME) $(msgid) $(offset)")

field ( FLNK, "$(P)Q:$(offset)-Cmd\_")

}

In addition to 'longout', the record types 'bo', 'mbbo', 'mbboDirect', and 'ao' may also be used.

This will update a scratch copy of the register block. Also add a record:

record(longout, "$(P)Q:$(offset)-Cmd\_")

{

Field ( DTYP, "PSC Ctrl Send")

field ( OUT , "@$(NAME)")

field ( VAL , "$(msgid)")

field ( FLNK, "$(P)Send-Cmd")

}

When processed, this will build a message using the current contents of the scratch copy, and add it to the send buffer.

## **Single Register Sends**

DTYPs "PSC Single \*" Send a message with a body including a 32-bit address, and a 32-bit value. So the body length includes the 4 bytes of address.

DTYPs "PSC Single U32" is supported for bo, mbbo, and mbboDirect recordtypes, "PSC Single I32" is supported for longout and ao recordtypes, and "PSC Single F32" is supported for the ao recordtype.

## **Global and Statistics**

Several DTYPs are available to give socket global information as well as event counters (both global and per msgid).

A **bi** record with DTYP "PSC Ctrl Connected" gives an indication of current TCP socket connection status. Supports "I/O Intr".

A **stringin** record with "PSC Ctrl Message" gives the last message string from the driver. Used to give socket and driver level error messages. Supports "I/O Intr".

A **bo** record with "PSC Ctrl Send All" flushes the driver transmit buffer to the socket buffer.

A **longout** with "PSC Ctrl Send" adds the current content of a message block to the driver's transmit buffer.

A **longin** with ""PSC Unknown Msg Count" or "PSC Conn Count" gives the current # of unknown messages (no records listening) received, or the number of successful connections.

A **longin** with "PSC Block Msg Count" gives the number of messages received with a given message id.

## **Driver Configuration Variables**

**PSCDebug** - Controls error and debug messages.

* 0 - Print as little as possible
* 1 - Print errors (default)
* 2 - Print info and messages sent
* 5 - Full debug info

**PSCMaxSendBuffer** - Max send buffer size

* Defaults to 1MB

**PSCInactivityTime** - Timeout when nothing send/received for some seconds.

Closes the connection and starts the reconnect timer.