;APP01.DAP

;Application 1: Sampling Three Inputs

;from Apps2000.pdf

; resets board

RESET

; Start input procedure with

; IDEFINE <name> <# input pipes for procedure

IDEFINE A 3

SET IPIPE0 S2 ; input proc. SET IPIPE<#> <input ID>

; in this case Singleended, input 2

SET IPIPE1 S5 ; input proc, SingleEnded, input 5

SET IPIPE2 D0 ; input proc, Differential, input 0

TIME 10000 ; set sampling time to 10,000 microseconds.

; with 3 pins, inputs are sampled every 30K us

END ; end input procedure

PDEFINE B ; start processing procedure with PDEFINE <name>

BPRINT ; transfer binary data from all input channels

;to binary pipe $BINOUT (direct to PC)

END ; end of processing procedure

To start acquisition, send START command: START <procedure names>

START A, B

To stop acquisition, send STOP command:

STOP

//APP02.DAP

// Application 2: Sampling 5000 values with hardware triggering

// from Apps2000.pdf

RESET // reset

IDEFINE A 1 // define input procedure A, one channel inpput

SET IPIPE0 S0 // input process, single ended, pin 0

TIME 20 // time interval of 20 microseconds between samples (50 kHz rate)

COUNT 100 // acquire 100 samples -> 2x10^6 us = 2 ms

HTRIGGER ONESHOT // use external triggering - start acquisition when ext. trigger goes from low to high.

END

PDEFINE B

BPRINT

END

START A, B

MATLAB: Set 1

Example set 1 demonstrates the use of a Data Acquisition Processor with MATLAB. The set contains three MATLAB files, init.m, data.m and stop.m, and one DAPL file data.dap.

1. init.m –shows how to initialize communication with the DAP from MATLAB environment.
2. data.m –shows how to read data values from input channels into matrices.
3. stop.m –shows how to terminate communication with the DAP.
4. data.dap–shows how to configure the DAP.

File INIT.M

This file shows an example on how to initializes communication with the Data Acquisition Processor.

%% File INIT.M %% % Open text and binary handles to ACCEL device

texthandle = dapopen('\\.\dap0\$sysin', 'write')

if texthandle == -1

error('Error opening DAP text handle')

end

binaryhandle = dapopen('\\.\dap0\$binout', 'read')

if binaryhandle == -1

error('Error opening DAP binary handle')

end

% Send a command to reset the DAP

dappstr(texthandle, 'RESET');

% Flush old DAP data in binary pipe

dapflshi(binaryhandle);

File DATA.M

This file shows how to configure the Data Acquisition Processor with the DAPL file data.dap, and reads data from the Data Acquisition Processor into matrices in MATLAB environment.

%% File DATA.M %%

% Configure DAP using DAPL command file DATA.DAP

cnfg = dapcnfig(texthandle, 'data.dap');

if cnfg < 1  % Print error message

error('Error configuring DAP')

end

% Get DAP data % Read data from DAP

channel0 = dapgetm(binaryhandle,[200,1],‘int16’)

channel1 = dapgetm(binaryhandle,[200,1],‘int16’)

File STOP.M

This file shows how to terminate communication and closes all handles to the Data Acquisition Processor.

%% File STOP.M %%

% Stop DAP communication by sending STOP command to the DAP

dappstr(texthandle,'STOP');

% Close text and binary handles to ACCEL device

textclose = dapclose(texthandle)

if textclose == -1

error('Error closing DAP text handle')

end

binclose = dapclose(binaryhandle)

if binclose==-1

error('Error closing DAP binary handle')

end

DAPL Command File: DATA.DAP

This file shows the DAPL configuration for the Data Acquisition Processor.

;; DAPL file, DATA.DAP ;;

reset

; Sample analog input, s0

; Sample analog input, s1

; Time of 5 msec ; Take 1000 samples

idef a 2

set ip0 s0

set ip1 s1

time 5000

count 1000

end

; Send data through binary communication pipe

pdef b  merge(ip0, ip1, $binout) end

start a, b

Running the Examples

To run the example at the MATLAB command window, type the following in the

MATLAB command window in sequence:

init

data

stop

***Conventional communication handles***

There are actually four predefined communication pipes that you should know about. These scripts try to

open or close a handle to each channel automatically, making the channels available if you need them,

doing no harm if you don't need them. If you use these scripts, you will soon come to expect that these

handles are always available and you won't think about them much. All of the other examples use these.

|  |  |  |
| --- | --- | --- |
| Handle variable | Associated  communication pipe | Purpose |
| **hTextToDap** | $SysIn | Commands going to DAPL system on DAP |
| **hTextFromDap** | $SysOut | Messages returned to host system from DAP |
| **hBinToDap** | $BinIn | Precomputed signals downloaded to DAP |
| **hBinFromDap** | $BinOut | Sample values delivered to host from DAP |

**Application 46 — Sending Data to a Data Acquisition Processor**

A Data Acquisition Processor supports high-speed binary data transfer to and from a host PC. Receiving data from the PC is similar to sending data to the PC. This application provides two examples illustrating PC to Data Acquisition Processor binary transfer. The first example shows how to transfer previously logged data to a Data Acquisition Processor for analog output. The second example illustrates the use of a Data Acquisition Processor for postprocessing.

A Data Acquisition Processor reads data from the PC through the $BININ communication pipe. Any task can use $BININ as its input pipe. The following DAPL commands illustrate reading data from the PC and sending the data to output channel pipe 0.

RESET

PDEF A

COPY ($BININ, OPIPE0)

END

ODEF B 1

SET OPIPE0 A0

TIME 10000

END

START A, B

A COPY command reads data from $BININ and sends the data to output channel pipe 0. An output procedure defines output channel pipe 0 for analog output on A0 at 100 Hz.

DAPview Plus can transfer data from a file to the Data Acquisition Processor using the Data file option in the Control menu. Programs created with programming languages also can transfer data to a Data Acquisition Processor.

A Data Acquisition Processor also can process data from the PC and send the results back to the PC. The following example illustrates how a Data Acquisition Processor with an on-board DSP chip accelerates FFT processing of PC data.

RESET

PDEF A

FFT (5, 8, 1, $BININ, $BINOUT)

END

START A

HTRIGGER (for output updating)

Specify the operating mode of the output hardware trigger signal.

HTRIGGER <type>

Parameters

<type> A keyword: ONESHOT | GATED | OFF

Description

Including the HTRIGGER in an output updating configuration determines the manner that output updating responds to the external hardware output triggering signal. The <type> keyword specifies the operating mode and must be one of the following:

OFF

The default when the HTRIGGER command is omitted from the output configuration. The DAP does not respond to the external hardware signal. Updating can begin when sufficient data are available in the output buffers.

ONESHOT

Updating can begin when the hardware output triggering signal goes active. Once the trigger level is detected and updating is underway, the DAP no longer responds to the external hardware signal.

GATED

Update clocking proceeds while the hardware trigger signal is at the active level, and is suspended while the hardware trigger signal is at the inactive level.

The hardware trigger signal is described in the Data Acquisition Processor hardware documentation. It is not recognized until the OUTPUTWAIT condition is satisfied.

Example :

HTRIGGER GATED

Specify that the hardware trigger allows output updates to be delivered when the external trigger level is active, and suspends updates when the external trigger level is inactive.

See Also: CLCLOCKING, CLOCK, UPDATE, OUTPUTWAIT, HTRIGGER for input

**OUTPUTWAIT**

Delay output updating until sufficient data are present in output channel pipes.

**OUTPUTWAIT** <preload>

**Parameters**

<preload>

Number of samples that must be available before updating starts.

WORD CONSTANT | LONG CONSTANT

**Description**

The OUTPUTWAIT command causes an output-updating configuration to wait until <preload> samples have

arrived from a data supplier before beginning to deliver output updates. The scheduling of tasks is unpredictable,

and particularly so when a system configuration first starts to run. A delay in starting the output updates allows

collecting enough data in memory to sustain operation despite temporary delays. Running out of data would cause

the hardware processes to terminate prematurely with an *underflow condition.*

The value of <preload> must not be zero and must be a multiple of the number of output channel pipes. After

this number of samples (or more) is received into the output channel pipe, the OUTPUTWAIT condition is satisfied,

and the output updating task is scheduled for execution at high priority.

When the HTRIGGER command specifies an external hardware triggering mode other than OFF, the OUTPUTWAIT

condition must be satisfied first before the output updating will respond to the external trigger signal level.

If a CYCLE command is present in the output-updating configuration, it overrides the OUTPUTWAIT specification.

Satisfying the CYCLE command is equivalent to having an unlimited number of values available in memory.

For multiple-DAP systems generating output signals in a master-slave configuration, the OUTPUTWAIT value for

each slave board must be satisfied before it can respond to the output clocking signals from the master board.

Conflicts would occur at initial startup or at the beginning of output bursts. In some rare cases, you might need to

increase the OUTPUTWAIT value on the master board, to give the slave boards sufficient time to generate enough

data.

If you do not specify an OUTPUTWAIT value, the DAPL system computes a default number equivalent to 100

milliseconds (1/10 second) of output updates. Most applications should use the default OUTPUTWAIT value.

Warning: Setting <preload> less than its default value may result in output underflow and premature shutdown.

**Example**

OUTPUTWAIT 1000

Wait for 1000 values to arrive in the output channel pipe before starting to update the outputs.

**See Also**

COUNT for output, CYCLE, UPDATE, HTRIGGER for output

**EDIT**

Modify input configurations and output configurations.

**EDIT** <proc\_name> <proc\_command>

**ED** <proc\_name> <proc\_command>

Modify com pipes.

**EDIT** <cpipe\_name> <cpipe\_parameters>

**ED** <cpipe\_name> <cpipe\_parameters>

**Parameters**

<proc\_name>

The name of the input or output configuration to change.

<proc\_command>

A configuration command as it would appear in an input configuration or output configuration.

<cpipe\_name>

The name of the communications pipe to change.

<cpipe\_parameters>

Communications pipe configuration parameters.

**Description**

EDIT modifies an input configuration, output configuration, or communications pipe.

For input or output configurations, the <proc\_command> has the same form as a configuration command in the

original input or output configuration. For input configurations, <proc\_command> can be a SET, TIME or COUNT

command. For output configurations, <proc\_command> can be a SET, TIME, COUNT, CYCLE or OUTPUTWAIT

command. See the command references pages for information about these individual command types.

For a communications pipe, <cpipe\_parameters> can be BLOCKING = <num> or

WIDTH=LONG|WORD|BYTE|FLOAT.

The EDIT command can only change a configuration or com pipe when it is inactive. In particular, the data type of

a communications pipe can be changed only when the pipe is empty.

**Examples**

EDIT INPR SET IP4 S3 10

Change input pipe 4 of configuration INPR to input S3 with a gain of 10.

**Chapter 19 DAPL Commands: EDIT 211**

EDIT A TIME 1000

Change the update time of output configuration A to 1 millisecond per update.

EDIT $BinOut WIDTH=LONG

Change the width of $BinOut to long for sending long values to the PC.

**See Also**

ERASE, IDEFINE, ODEFINE, RESET