

ALS Controls Expansion / Superbend I/O card step 2 revised 8/29/99

the following was developed by Alan Biocca, Mike Chin, and Bill Brown
results from 7/30 meeting and subsequent work

Hardware Requirements

format: Industry Pack

4 16 bit analog input channels
 analog characteristics ... (see ILC-2 rqmnts)
2 16 bit analog output channels
 analog characteristics ... (see ILC-2)
2 bytes digital i/o
 with bitwise input/output selectability
 drive / input characteristics ... (see ILC-2)
product serial number
 stored in onboard serial mem
ADC, DAC calibration info
 stored in onboard serial mem
control output during power-on
 details ... tbd. depends on board space. (chin) tbd

serial eeprom for calibration coefficients
 no write-enable jumper required
 protocol sufficiently difficult
 RT system will not have write protocol code
 reading and writing will be handled by host CPU software
 no room in XILINX for help

control output behavior during reboot
 Mike to see if he has room to add jumper tbd

interrupt behavior
 insuff room for averaging. tbd??

readout behavior
 tbd. reading once may produce bad value. read twice
 and compare.

IP id header
 differentiable from first boards.
 different board model number or revision.
 no serial number required since it will be in flash mem

binary i/o changes tbd...
 was two bytes
 might change to 1.5 bytes
 review usage and transition module
 this might free up the bits needed for eeprom

transition module tbd ...
 should nail this down as it interacts some with ip

calibration and test plan

labview based calibration system

writes mx+b equivalent coefficients
writes segmented line fit coefficients (possibly, not req initially)

format of eeprom memory (proposed)

16-64K serial eeprom <size tbd> tbd ...
treat as 'disk files' of ascii data

first byte of memory is **'format indicator byte'**

 this constant reflects the format used
 and will change if we need to change format in the future
 chosen to be unusual value
 not found in unprogrammed memory or ascii files
 value <tbd> (in 128..255 range)

file format

file name
 is [#\$%*+-.:_0-9a-zA-Z]+
 terminate with <tbd>
 code above 128
zero or more lines of ascii
 zero or more ascii characters, 0 .. 127
 EOL line separator
 use single character
 use unix style <LF> linefeed
 each line of ascii will typically be
 tag <space> value <EOL>
 eg
 rev 02a1
 serial 1-137
EOF mark
 terminates the file
 not found in ascii data, not in 0..127
 use <tbd>
 two EOF marks in a row signifies end of media
 adding a file requires erasing second EOF mark
 and terminating with two EOF marks
 to maintain end of media

content of eeprom memory (proposed)

product rev and serial number (ascii, in file 'id')
 rev model-revno
 serial run-board
 date production-date

calibration data (file 'cal')

 date cal-date

 ai0m slope value analog inputs
 ai0b offset value
 ...
 ai3m
 ai3b
 ao10m val analog output slope
 ao0b

ao1m
ao1b
OR
ai 0 32767 voltage
ai 0 -32767 voltage
ai 1 32767 voltage
ai 1 -32767 voltage
OR
ai 0 mincount minvolts maxcount maxvolts
ai 1 ...
ai 2
ai 3
ao 0
ao 1

extended calibration data (file 'ecal')

ai 0 count voltage
ai 0 count voltage
...
ai 1 count voltage
...
ao 0 count voltage
...
ao 1 count voltage
...

reading eeprom memory

serial from eeprom to xilinx
serial to parallel in xilinx?
readout parallel via bus?
suggestion - software to copy out entire contents into ram at startup
read routines to read from ram image

writing eeprom memory

writing program sits in labview on pc
serializes to bus
bus to xilinx
xilinx to eeprom part
xilinx contains little support for writing
to help protect memory from erasure
consider a write-enable jumper?
writing functions (direct to eeprom memory)
format
writes EOF throughout memory
delete files (name)
overwrites first byte of name with EOF
effectively setting end of media there
deletes named file and those after
write EOFs over files??
open file for write (name)
finds end of media
writes file name, end marker for name
can have only one open write file at a time
write data to open file
checks characters for validity
converts end of line

```

    writes to file
close
    terminate write
    write dual EOF
        in case write ended inside old file?
eof
```