CESR-CLEO HV Interlock Software

All software is being developed in the HV module of the CVS
HVInterlockServer.cxx defines a class to represent the Interlock board. The following files contain all of the necessary definitions, etc.:

HVInterlockServer.h
HVInterlockServer.hxx
HVInterlockServer.cxx

*List of defines contained in **HVServerInterlock.h***

```
#define TOCLEAR    (unsigned short) 0x1
#define TOSET      (unsigned short) 0x2
#define INHIBIT    (unsigned short) 0x4
#define FAIL       (unsigned short) 0x8
#define HVEN       (unsigned short) 0x10

#define TO_MASK    (unsigned short) 0xc
#define TOC_SHIFT  0
#define TOS_SHIFT  1
#define INH_SHIFT  2
#define FAIL_SHIFT 3
#define HVEN_SHIFT 4

#define INTERLOCK_ID  (int)46
#define ZERO 0x0
#define CLK_RATE (float) 16000000.0
```

Define a struct called HVInterlockCSR

```
typedef struct
{
 unsigned short INTERLOCK;  /* 0x0 CESR-CLEO HV Interlock Base Address */
 unsigned short ID;         /* 0x2 CESR-CLEO HV Interlock ID */
 unsigned short CSR;        /* 0x4 Clears counter; CSR */
 unsigned short TMO;        /* 0x8 Watchdog Timeout */
}  HVInterlockCSR;
```

List of public accessor functions

unsigned short **getID**();

boolean **compareID**();

void **reset**();

unsigned short **getCSR**();

void **clearCSR**();

int **getTimeoutClear**();

int **getTimeoutClear**();

void **setTimeoutClear**();

void **clearTimeoutClear**();

int **getTimeoutSet**();

void **setTimeoutSet**();

void **clearTimeoutSet**();

int **getInhibit**();

void **setInhibit**();

void **clearInhibit**();

int **getFail**();

int **getHVEnable**();

unsigned short **getTimeoutRaw**();

void **setTimeoutRaw(*unsigned short ValueInHex***);

float **getTimeoutInSeconds**();

void **setTimeoutInSeconds**(*float ValueInSeconds);*

int **hitwd**();

Description of public accessor functions

unsigned short **getID**():

Input Parameters: None
Returns: Board ID (in decimal – should be 46)

Description: This reads address 0 of the board which returns the board ID. This value is set by LPM_CONSTANT inside the Altera chip.

***unsigned int* compareID*()*:**

Input Parameters: None
Returns: 1 if ID's match
0 if ID's do not match

Description: The ID on the board is a constant value set by LPM_CONSTANT inside the Altera chip. This value is currently set to 46. This function is meant to be a check to make sure that the board we're talking to is really a CESR-CLEO HV Interlock VME Board.

void **reset**():

Input Parameters: None
Returns: Nothing

Description: This function is used to recover from a FAIL condition after a timeout. It calls setTimeoutClear() followed by clearCSR(). This will clear the FAIL, reset the watchdog, and start the timer counting up again.

unsigned short **getCSR**():

Input Parameters: None
Returns: Current value of CSR in hex.

Description: Reads CSR.
Example: If getCSR() returns 0x18, that means
HVEnable = 1

FAIL = 1
All other bits are zero.

void **clearCSR**():

Input Parameters: None
Returns: Nothing

Description: Writes the value 0x0 to the CSR. This will clear the TOS, TOC, and INHIBIT bits and allow the counter to start counting up again. (It cannot clear the FAIL, nor does it touch the value of HVENABLE).

int **getTimeoutClear**():

Input Parameters: None
Returns: Value of Timeout Clear bit
1 if Timeout Clear is active
0 if Timeout Clear is not active

Description: Reads the CSR and masks off the Timeout Clear bit.

void **setTimeoutClear**():

Input Parameters: None
Returns: Nothing

Description: Writes D[2..0] = 2 to CSR. Sets Timeout Clear bit. If Timeout Set bit is already set, this function will clear the TOS bit and set the TOC bit.

void **clearTimeoutClear**():

Input Parameters: None
Returns: Nothing

Description: Writes to CSR. Clears Timeout Clear bit without changing any of the other bit values in the CSR.

int **getTimeoutSet**():

Input Parameters: None
Returns: Value of Timeout Set bit
1 if Timeout Set is active
0 if Timeout Set is not active

Description: Reads the CSR and masks off the Timeout Set bit.

void **setTimeoutSet**():

Input Parameters: None
Returns: Nothing

Description: Writes D[2..0] = 1 to CSR. Sets Timeout Set bit. If Timeout Clear bit is set, this function will clear the TOC bit and set the TOS bit.

void **clearTimeoutSet**():

Input Parameters: None
Returns: Nothing

Description: Writes to CSR. Clears Timeout Set bit without changing any of the other bit values in the CSR.

int **getInhibit**():

Input Parameters: None
Returns: Value of Inhibit bit from CSR
1 if INHIBIT relay is open
0 if INHIBIT relay is closed

Description: Reads the CSR and masks off the INHIBIT bit.

void **setInhibit**():

Input Parameters: None
Returns: Nothing

Description: Sets the INHIBIT bit, provided FAIL is not active. When FAIL is active, the INHIBIT bit is automatically cleared.

void **clearInhibit**():

Input Parameters: None
Returns: Nothing

Description: Clears the INHIBIT bit without affecting any of the other CSR values.

int **getFail**():

Input Parameters:	None
Returns:	1 if FAIL is active (CESR HVEnable controls CLEO HV) 0 if FAIL is inactive (CLEO controls its own HV)
Description:	Reads the CSR and masks off the FAIL bit and returns it.

***int* getHVEnable**():

Input Parameters: None
Returns: Status of CESR HV ENABLE

Description: Reads the CSR and masks off the HVEnable bit; returns it.

unsigned short **getTimeoutRaw**():

Input Parameters: None
Returns: Value of watchdog timeout register in hex

Description: Reads A = 3 and returns the value found there. This is the value that the watchdog timer will count up to. Once this value is reached, the watchdog will time out and FAIL will be asserted.

void **setTimeoutRaw(*unsigned short ValueInHex*)**:

Input Parameters: Value of watchdog timeout register in hex
Returns: Nothing

Description: Writes the number *ValueInHex* to A = 3.

float **getTimeoutInSeconds**():

Input Parameters: None
Returns: Value of watchdog timeout register converted to seconds.

Description: Reads A = 3, converts the value found there from hex to seconds using the value of the SYSTEM CLOCK that clocks the watchdog timer. This is the value that the watchdog timer will count up to. Once this value is reached, the watchdog will time out and FAIL will be asserted.

void **setTimeoutInSeconds**(*float ValueInSeconds*):

Input Parameters: The value the timer will count up to in seconds
Returns: Nothing

Description: Converts *ValueInSeconds* to the corresponding value in hex. Then, it writes the hex value to A = 3. This is the

value that the watchdog timer will count up to (in seconds).

int **hitwd**():

Input Parameters:	None
Returns:	Status of CESR HV ENABLE
Description:	Hits watchdog timer by reading CSR.