**LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY**

**-LIGO-**

**CALIFORNIA INSTITUTE OF TECHNOLOGY**

**MASSACHUSETTS INSTITUTE OF TECHNOLOGY**

|  |  |  |
| --- | --- | --- |
| Document TypeTest Procedure | DCC Number **T1500440**-v1 | August 13, 2015 |
| **Hardware Watchdog PSoC Schematic and Code Changes** |
| B. Abbott, R. Abbott |

Distribution of this draft:

This is an internal working note of the LIGO Laboratory

 **California Institute of Technology Massachusetts Institute of Technology**

 **LIGO Project – MS 18-33 LIGO Project – MS 20B-145**

 **Pasadena, CA 91125 Cambridge, MA 01239**

 Phone (626) 395-2129 Phone (617) 253-4824

 Fax (626) 304-9834 Fax (617) 253-7014

 E-mail: info@ligo.caltech.edu E-mail: info@ligo.mit.edu

 <http://www.ligo.caltech.edu/>

**Overview:**

The Cypress PSoC Creator 3.1 file for the 2014 (v1) iteration of the Hardware Watchdog Chassis (D1300642-v1) as found to be very confusing to understand and troubleshoot, which would make it unwieldy to maintain and upgrade, if needed. We set about to simplify the PSoC Top Design schematic, and comment and simplify the main.c code. This document shows the reasons for the changes, and the steps that were taken. After these changes were made, the new code was fully tested on a watchdog chassis to ensure that no functional changes occurred.

**Process:**

This is the schematic as it exists in the DCC as the “TopDesign.cysch” in a PSoC Creator 3.1 project file. We found it very difficult to follow and understand.

Before:



Pins were eliminated that were never used in the code, and were not wired on the hardware board at all. They were likely vestigal pins from older iterations of the design. The remaining pins were either MUX selector bits, Error output bits, the comparator output pin, the PD input signal or the Remote\_Reset, and Reset\_rqst\_in which are both signals coming from the Hardware board, and are “OR”ed together in code, rendering the schematic “OR” symbol superfluous:



With those pins gone, it left logic gates, D-flip-flops, and a counter that all did nothing. I eliminated them, along with their individual clocks:



This left a pretty clean schematic that could be labeled and rearranged from this:



To this:



2. The code:

The original code was completely un-commented (except in the copy-and-pasted DMA\_Config function). This made it extremely difficult to follow. There also was some vestigial code that turned on or off things that were otherwise not being used. I went through and commented the code, and removed useless code. Here is the original code:

/\* File Name: main.c

\* working code Hwardware Watchdog 2014

\*

\* Description:

\* Sets up a complete hardware watchdog system

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

#include <device.h>

#include <project.h>

#include <math.h>

#define REQUEST\_PER\_BURST 1u

#define BYTES\_PER\_BURST 1u

#define UPPER\_SRC\_ADDRESS CYDEV\_PERIPH\_BASE

#define UPPER\_DEST\_ADDRESS CYDEV\_PERIPH\_BASE

#define LED\_TRUE 1

#define LED\_FALSE 0

#define PD\_TRUE 1

#define PD\_FALSE 0

void DMA\_Config(void);

void PD\_RMS1(void);

void Pin\_Init(void);

void Zero\_Init(void);

void main()

{

 int i, j,k,l,m, u, v, Vout, RMS\_Scale;

 int N=63, Nx, Ny, Nz;

 float mean, max[6], min[6], s[6], rms\_ratio;

 float PD\_IN[6][63];

 float out[6][63], LED\_IN[6], x[6], kf;

 float PD\_rms[6];

 int rms\_cal[6];

 int timing\_length, rms\_length, minimum\_length;

 int timing\_count, rms\_count, which\_count;

 float timing\_constant, rms\_constant;

 int8 Vpd\_rms\_ref=30;

 float Vled\_ref=0.5;

 int16 output[6][63];

 timing\_constant=1.0;

 rms\_constant=1.0;

 rms\_ratio=0.55;

 X0: Zero\_Init();

 CyDelay(2000);

 u=0;

 v=0;

 l=0;

 kf=0;

 for (i=0; i<6; i++)

 {

 rms\_cal[i]=0;

 }

 PD\_ER\_Write(1);

 CyDelay(2000);

 PD\_ER\_Write(0);

 LED\_ER\_Write(1);

 CyDelay(2000);

 LED\_ER\_Write(0);

 PD\_ER\_Write(1);

 CyDelay(2000);

 PD\_ER\_Write(0);

 LED\_ER\_Write(1);

 CyDelay(2000);

 LED\_ER\_Write(0);

 X1: if(!Remote\_Reset\_Read())

 goto xyz1;

 else

 goto xyz;

 xyz1: Nx=0;

 while(!Remote\_Reset\_Read())

 {

 Nx=Nx+1;

 CyDelay(20);

 if(Nx>1000)

 {

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 goto xyz;

 }

 }

 if((Nx>50)&(Nx<255))

 goto read\_setting;

 if((Nx>300)&(Nx<525))

 goto rms\_writing;

 if((Nx>575)&(Nx<775))

 goto timing\_writing;

 goto xyz;

 read\_setting:

 rms\_length=rms\_constant\*100;

 timing\_length=timing\_constant\*200;

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(1000);

 if(rms\_length>timing\_length)

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 for(i=0; i<timing\_length; i++)

 {

 CyDelay(20);

 }

 LED\_ER\_Write(0);

 for(i=0; i<(rms\_length-timing\_length+1); i++)

 {

 CyDelay(20);

 }

 PD\_ER\_Write(0);

 CyDelay(1000);

 for (i=0; i<5; i++)

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 }

 goto xyz;

 }

 else

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 for(i=0; i<rms\_length; i++)

 {

 CyDelay(20);

 }

 PD\_ER\_Write(0);

 for(i=0; i<(timing\_length-rms\_length+1); i++)

 {

 CyDelay(20);

 }

 LED\_ER\_Write(0);

 CyDelay(1000);

 for (i=0; i<5; i++)

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 }

 goto xyz;

 }

 rms\_writing: PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(1010);

 Ny=0;

 while(!Remote\_Reset\_Read())

 {

 Ny=Ny+1;

 CyDelay(20);

 if(Ny>350)

 goto xyz;

 }

 if((Ny<30))

 goto xyz;

 else

 {

 rms\_constant=0.01\*Ny;

 for (i=0; i<10; i++)

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 }

 goto xyz;

 }

 timing\_writing: PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(1010);

 Nz=0;

 while(!Remote\_Reset\_Read())

 {

 Nz=Nz+1;

 CyDelay(20);

 if(Nz>350)

 goto xyz;

 }

 if(Nz<18)

 goto xyz;

 else

 {

 timing\_constant=0.005\*Nz;

 for (i=0; i<15; i++)

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 }

 goto xyz;

 }

 xyz: for (j=0; j<N; j++)

 {

 for (i=0; i<6; i++)

 {

 CyDelay(1);

 LED\_IN1\_Sel\_Write(0);

 LED\_IN2\_Sel\_Write(0);

 LED\_IN3\_Sel\_Write(0);

 LED\_IN4\_Sel\_Write(0);

 LED\_IN5\_Sel\_Write(0);

 LED\_IN6\_Sel\_Write(0);

 PD\_IN1\_Sel\_Write(0);

 PD\_IN2\_Sel\_Write(0);

 PD\_IN3\_Sel\_Write(0);

 PD\_IN4\_Sel\_Write(0);

 PD\_IN5\_Sel\_Write(0);

 PD\_IN6\_Sel\_Write(0);

 CyDelay(1);

 if(i==0)

 PD\_IN1\_Sel\_Write(1);

 if(i==1)

 PD\_IN2\_Sel\_Write(1);

 if(i==2)

 PD\_IN3\_Sel\_Write(1);

 if(i==3)

 PD\_IN4\_Sel\_Write(1);

 if(i==4)

 PD\_IN5\_Sel\_Write(1);

 if(i==5)

 PD\_IN6\_Sel\_Write(1);

 CyDelay(1);

 ADC\_DelSig\_IRQ\_Start();

 ADC\_DelSig\_Start();

 ADC\_DelSig\_StartConvert();

 Filter\_Start();

 if(ADC\_DelSig\_IsEndConversion(ADC\_DelSig\_WAIT\_FOR\_RESULT))

 {

 out[i][j]=ADC\_DelSig\_GetResult16();

 }

 CyDelay(1);

 }

 }

 for (i=0; i<6; i++)

 {

 PD\_rms[i]=0.0;

 mean=0.0;

 for(j=1; j<N; j++)

 {

 mean=mean+out[i][j];

 }

 mean=mean/(N-1);

 for(j=1; j<N; j++)

 {

 PD\_rms[i]=PD\_rms[i]+(out[i][j]-mean)\*(out[i][j]-mean);

 }

 PD\_rms[i]=PD\_rms[i]/(N-1);

 PD\_rms[i]=sqrt(PD\_rms[i]);

 rms\_cal[i]=PD\_rms[i];

 PD\_rms[i]=PD\_rms[i]/rms\_ratio;

 PD\_rms[i]=PD\_rms[i]/rms\_constant;

 }

 VDAC\_Start();

 if (PD\_rms[0]>(0.2\*Vpd\_rms\_ref))

 {

 rms\_cal[0]=0.69\*rms\_cal[0]+0.57\*kf;

 kf=rms\_cal[0];

 VDAC\_SetValue(rms\_cal[0]);

 CyDelay(10);

 }

 else

 {

 VDAC\_SetValue(3);

 CyDelay(10);

 }

 m=0;

 for (i=0; i<6; i++)

 {

 if (PD\_rms[i]>Vpd\_rms\_ref)

 {

 m=1;

 }

 else

 m=m;

 }

 if(m==1)

 {

 PD\_ER\_Write(1);

 CyDelay(1);

 }

 else

 {

 PD\_ER\_Write(0);

 CyDelay(1);

 }

 X2: for (i=0; i<6; i++)

 {

 Pin\_Init();

 DAC\_Start();

 Comp1\_Start();

 CyDelay(1);

 if(i==0)

 LED\_IN1\_Sel\_Write(1);

 if(i==1)

 LED\_IN2\_Sel\_Write(1);

 if(i==2)

 LED\_IN3\_Sel\_Write(1);

 if(i==3)

 LED\_IN4\_Sel\_Write(1);

 if(i==4)

 LED\_IN5\_Sel\_Write(1);

 if(i==5)

 LED\_IN6\_Sel\_Write(1);

 CyDelay(1);

 if (Comp1\_GetCompare())

 {

 k=1;

 }

 else

 {

 k=k;

 }

 }

 if (k==1)

 {

 if(l==0)

 {

 LED\_ER\_Write(0);

 }

 else

 { LED\_ER\_Write(1); }

 }

 else

 {

 LED\_ER\_Write(0);

 }

 X3: if((m==1) || (k==1))

 {

 l=l+1;

 m=0;

 k=0;

 }

 else

 {

 l=0;

 Count\_CLR\_WR\_Write(0);

 Counter\_ER\_Stop();

 }

 X4: if(l>(687\*timing\_constant))

 {

 HEPI\_ERROR\_Write(1);

 goto X5;

 }

 else

 {

 HEPI\_ERROR\_Write(0);

 goto X1;

 }

 X5: if (l>(1330\*timing\_constant))

 { SUS\_ERROR\_Write(1);

 l=3000;

 }

 else

 {

 SUS\_ERROR\_Write(0);

 }

 X6: for (j=0; j<N; j++)

 {

 for (i=0; i<6; i++)

 {

 CyDelay(1);

 LED\_IN1\_Sel\_Write(0);

 LED\_IN2\_Sel\_Write(0);

 LED\_IN3\_Sel\_Write(0);

 LED\_IN4\_Sel\_Write(0);

 LED\_IN5\_Sel\_Write(0);

 LED\_IN6\_Sel\_Write(0);

 PD\_IN1\_Sel\_Write(0);

 PD\_IN2\_Sel\_Write(0);

 PD\_IN3\_Sel\_Write(0);

 PD\_IN4\_Sel\_Write(0);

 PD\_IN5\_Sel\_Write(0);

 PD\_IN6\_Sel\_Write(0);

 CyDelay(1);

 if(i==0)

 PD\_IN1\_Sel\_Write(1);

 if(i==1)

 PD\_IN2\_Sel\_Write(1);

 if(i==2)

 PD\_IN3\_Sel\_Write(1);

 if(i==3)

 PD\_IN4\_Sel\_Write(1);

 if(i==4)

 PD\_IN5\_Sel\_Write(1);

 if(i==5)

 PD\_IN6\_Sel\_Write(1);

 CyDelay(1);

 ADC\_DelSig\_IRQ\_Start();

 ADC\_DelSig\_Start();

 ADC\_DelSig\_StartConvert();

 Filter\_Start();

 DMA\_Config();

 CYGlobalIntEnable;

 if(ADC\_DelSig\_IsEndConversion(ADC\_DelSig\_WAIT\_FOR\_RESULT))

 {

 out[i][j]=ADC\_DelSig\_GetResult16();

 }

 CyDelay(1);

 }

 }

 for (i=0; i<6; i++)

 {

 PD\_rms[i]=0.0;

 mean=0.0;

 for(j=1; j<N; j++)

 {

 mean=mean+out[i][j];

 }

 mean=mean/(N-1);

 for(j=1; j<N; j++)

 {

 PD\_rms[i]=PD\_rms[i]+(out[i][j]-mean)\*(out[i][j]-mean);

 }

 PD\_rms[i]=PD\_rms[i]/(N-1);

 PD\_rms[i]=sqrt(PD\_rms[i]);

 rms\_cal[i]=PD\_rms[i];

 PD\_rms[i]=PD\_rms[i]/rms\_ratio;

 PD\_rms[i]=PD\_rms[i]/rms\_constant;

 }

 VDAC\_Start();

 if (PD\_rms[0]>(0.2\*Vpd\_rms\_ref))

 {

 rms\_cal[0]=0.69\*rms\_cal[0]+0.57\*kf;

 kf=rms\_cal[0];

 VDAC\_SetValue(rms\_cal[0]);

 CyDelay(10);

 }

 else

 {

 VDAC\_SetValue(3);

 CyDelay(10);

 }

 m=0;

 for (i=0; i<6; i++)

 {

 if (PD\_rms[i]>Vpd\_rms\_ref)

 {

 m=1;

 }

 else

 m=m;

 }

 for (i=0; i<6; i++)

 {

 CyDelay(1);

 Pin\_Init();

 CyDelay(5);

 if(i==0)

 LED\_IN1\_Sel\_Write(1);

 if(i==1)

 LED\_IN2\_Sel\_Write(1);

 if(i==2)

 LED\_IN3\_Sel\_Write(1);

 if(i==3)

 LED\_IN4\_Sel\_Write(1);

 if(i==4)

 LED\_IN5\_Sel\_Write(1);

 if(i==5)

 LED\_IN6\_Sel\_Write(1);

 DAC\_Start();

 Comp1\_Start();

 CyDelay(1);

 if (LED\_info\_Read())

 if (Comp1\_GetCompare())

 k=1;

 else

 k=k;

 }

 if((m==1) || (k==1))

 {

 l=l+1;

 }

 else

 {

 l=l;

 }

 if (m==0)

 PD\_ER\_Write(0);

 if (m==1)

 PD\_ER\_Write(1);

 if (k==0)

 LED\_ER\_Write(0);

 if (k==1)

 LED\_ER\_Write(1);

 if(((!Remote\_Reset\_Read()) || Reset\_rqst\_in\_Read()) & (m==0) & (k==0))

 {

 CyDelay(2000);

 l=0;

 goto X0;

 }

 else

 {

 m=0;

 k=0;

 Reset\_rqst\_CLR\_WR\_Write(1);

 Reset\_rqst\_CLR\_WR\_Write(0);

 goto X5;

 }

 }

 void Pin\_Init(void)

 {

 CyDelay(10);

 LED\_IN1\_Sel\_Write(0);

 LED\_IN2\_Sel\_Write(0);

 LED\_IN3\_Sel\_Write(0);

 LED\_IN4\_Sel\_Write(0);

 LED\_IN5\_Sel\_Write(0);

 LED\_IN6\_Sel\_Write(0);

 PD\_IN1\_Sel\_Write(0);

 PD\_IN2\_Sel\_Write(0);

 PD\_IN3\_Sel\_Write(0);

 PD\_IN4\_Sel\_Write(0);

 PD\_IN5\_Sel\_Write(0);

 PD\_IN6\_Sel\_Write(0);

 CyDelay(1);

 }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name: DMA\_Config

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Summary:

\* Initializes and sets up DMA for use

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void DMA\_Config(void)

{

 /\* Declare variable to hold the handle for DMA channel \*/

 uint8 channelHandle;

 /\* Declare DMA Transaction Descriptor for memory transfer into

 \* High Pass Filter Channel.

 \*/

 uint8 tdChanA;

 /\* Configure the DMA to Transfer the data in 1 burst with individual trigger

 \* for each burst.

 \*/

 channelHandle = DMA\_DmaInitialize(BYTES\_PER\_BURST, REQUEST\_PER\_BURST,

 HI16(UPPER\_SRC\_ADDRESS), HI16(UPPER\_DEST\_ADDRESS));

 tdChanA = CyDmaTdAllocate();

 CyDmaTdSetConfiguration(tdChanA, 1, DMA\_INVALID\_TD, 0);

 /\* Set the source address as ADC\_DelSig and the destination as

 \* Filter Channel A. \*/

 CyDmaTdSetAddress(tdChanA, LO16((uint32)ADC\_DelSig\_DEC\_SAMP\_PTR), LO16((uint32)Filter\_STAGEAH\_PTR));

 /\* Set tdChanA to be the initial TD associated with channelHandle \*/

 CyDmaChSetInitialTd(channelHandle, tdChanA);

 /\* Enable the DMA channel represented by channelHandle and preserve the TD \*/

 CyDmaChEnable(channelHandle, 1);

}

 void Zero\_Init(void)

 {

 HEPI\_ERROR\_Write(0);

 SUS\_ERROR\_Write(0);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 Comp\_CLK\_Start();

 CLK\_Start();

 Reset\_CLK\_Start();

 TC\_CLK\_Start();

 TIME\_IN\_Start();

 Count\_CLR\_WR\_Write(0);

 TC\_Ctr\_WR\_Write(1);

 Comp\_Ctr\_WR\_Write(1);

 Reset\_rqst\_Ctr\_WR\_Write(1);

 TC\_Ctr\_WR\_Write(0);

 Comp\_Ctr\_WR\_Write(0);

 Reset\_rqst\_Ctr\_WR\_Write(0);

 Reset\_rqst\_CLR\_WR\_Write(1);

 TC\_CLR\_WR\_Write(1);

 Comp\_CLR\_WR\_Write(1);

 CyDelay(1);

 Reset\_rqst\_CLR\_WR\_Write(0);

 Comp\_CLR\_WR\_Write(0);

 TC\_CLR\_WR\_Write(0);

 }

Here is the new, commented and cleaned code:

/\* File Name: main.c

\* working code for Hardware Watchdog

\*E1500315 (Set version number on line 42)

\* Description:

\* Sets up a complete hardware watchdog system

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

#include <device.h>

#include <project.h>

#include <math.h>

#define REQUEST\_PER\_BURST 1u

#define BYTES\_PER\_BURST 1u

#define UPPER\_SRC\_ADDRESS CYDEV\_PERIPH\_BASE

#define UPPER\_DEST\_ADDRESS CYDEV\_PERIPH\_BASE

#define LED\_TRUE 1

#define LED\_FALSE 0

#define PD\_TRUE 1

#define PD\_FALSE 0

void DMA\_Config(void);

void PD\_RMS1(void);

void Pin\_Init(void);

void Zero\_Init(void);

void main()

{

 int i, j,k,l,m, u, v, version;

 int N=63, Nx, Ny, Nz;

 float mean, max[6], min[6], s[6], rms\_ratio;

 float PD\_IN[6][63];

 float out[6][63], LED\_IN[6], x[6], kf;

 float PD\_rms[6];

 int rms\_cal[6];

 int timing\_length, rms\_length;

 float timing\_constant, rms\_constant;

 int8 Vpd\_rms\_ref=30;

 float Vled\_ref=0.5;

 int16 output[6][63];

 timing\_constant=1.0;

 rms\_constant=1.0;

 rms\_ratio=0.55;

 version=2; //: Enter new code version number here after any revisions://

 LCD\_Start();

 X0: Zero\_Init(); //:Calls the "Zero Init" function on line 644 to set all the output

 selector bits to "0"://

 CyDelay(2000); //:Pause for 2 seconds://

 u=0;

 v=0;

 l=0;

 kf=0;

 for (i=0; i<6; i++)//:Initializes all of the rms calculation variables to

"0"://

 {

 rms\_cal[i]=0;

 }

 for(i=0; i<version; i++)//: Flashes the front panel "LED" and "PD" lights to indicate

 the number of current code version://

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(1000);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(1000);

 }

 X1: if(!Remote\_Reset\_Read()) //:If, upon power up, the control room is

sending a "Reset" request, goto xyz1://

 goto xyz1;

 else//: Otherwise, go to the main code, xyz://

 goto xyz;

 xyz1: Nx=0;

 while(!Remote\_Reset\_Read()) //:Count the duration of the control room's

 "Reset" request://

 {

 Nx=Nx+1;

 CyDelay(20);

 if(Nx>1000) //:If the "Reset" Request duration is greater than 1000 loops of

the xyz1 code, flash error lights, and goto xyz://

 //:xyz is the main code area://

 {

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 goto xyz;

 } //:End the if(Nx>1000) loop://

 } //:End the while(!Remote\_Reset\_Read) loop://

 if((Nx>50)&(Nx<255)) //:If the Nx timer stopped between 50 and 255

loops, goto read\_setting subroutine://

 goto read\_setting;

 if((Nx>300)&(Nx<525)) //:If the Nx timer stopped between 300 and 525

loops, goto rms\_writingting subroutine://

 goto rms\_writing;

 if((Nx>575)&(Nx<775)) //:If the Nx timer stopped between 575 and 775

loops, goto timing\_writing subroutine://

 goto timing\_writing;

 goto xyz;

 read\_setting: //:This bit of code allows the control room to glean the settings of the

current timing length and rms length://

 //:By booting, and giving a Control Room "Reset" command that lasts

between 1 second (50\*20ms per loop) and ://

 //: 5.1 seconds (255 \* 20ms), you get a readback of the timing constant

and rms\_constant://

 rms\_length=rms\_constant\*100;

 timing\_length=timing\_constant\*200;

 PD\_ER\_Write(0); //:Initialize the PD\_ER and LED\_ER bits to zero://

 LED\_ER\_Write(0);

 CyDelay(1000);

 if(rms\_length>timing\_length)//:If the rms\_constant is bigger than twice

 the timing\_constant, turn on the PD\_ER and

 LED\_ER bits://

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 for(i=0; i<timing\_length; i++) //:Pause for 20ms\*timing\_length://

 {

 CyDelay(20);

 }

 LED\_ER\_Write(0); //:turn off the LED\_ER bit, then pause for 20ms times

the difference between the rms\_length and the

timing\_length://

 for(i=0; i<(rms\_length-timing\_length+1); i++)

 {

 CyDelay(20);

 }

 PD\_ER\_Write(0);//: Then turn off the PD\_ER bit://

 CyDelay(1000); //: Pause for a second://

 for (i=0; i<5; i++) //:Then blink the two bits on and off 5 times to

signify completion://

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 }

 goto xyz; //:go to the main code://

 }

 else//:If the rms\_constant is not bigger than twice the timing\_constant,

 turn on the PD\_ER and LED\_ER bits://

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 for(i=0; i<rms\_length; i++)//:Pause for 20ms\*rms\_length://

 {

 CyDelay(20);

 }

 PD\_ER\_Write(0);//: Turn off the PD\_ER bit (instead of the LED\_ER bit

 above)://

 for(i=0; i<(timing\_length-rms\_length+1); i++)//:pause for 20ms times

the difference between the timing\_length and the rms\_length ://

 {

 CyDelay(20);

 }

 LED\_ER\_Write(0);//: Then turn off the LED\_ER bit://

 CyDelay(1000); //: Pause for a second://

 for (i=0; i<5; i++)//:Then blink the two bits on and off 5 times to

signify completion://

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 }

 goto xyz; //:go to the main code://

 }

 rms\_writing: PD\_ER\_Write(0); //:Initialize the PD\_ER and LED\_ER bits to zero://

 LED\_ER\_Write(0);

 CyDelay(1010); //: Pause for just over a second, assumedly waiting for

another "Reset" command from the control room://

 Ny=0;

 while(!Remote\_Reset\_Read())//:Count the duration of the control room's

 second "Reset" request://

 {

 Ny=Ny+1;

 CyDelay(20);

 if(Ny>350)//:If the second "Reset" Request duration is greater than

350 loops of the rms\_writing code, goto the main code://

 goto xyz;

 }

 if((Ny<30))//:If the second "Reset" Request duration is less than 30

 loops of the rms\_writing code, goto the main code://

 goto xyz;

 else //: If it's between these two duration times, use its duration

to set the new rms\_constant value://

 {

 rms\_constant=0.01\*Ny; //:the new value of rms\_constant is the 0.01

times the "Reset" command duration divided by 20ms CyDelay://

 for (i=0; i<10; i++)//:Then blink the two bits on and off 10 times to

 signify completion://

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 }

 goto xyz;//:go to the main code://

 }

 timing\_writing: PD\_ER\_Write(0); //:Initialize the PD\_ER and LED\_ER bits to zero://

 LED\_ER\_Write(0);

 CyDelay(1010); //: Pause for just over a second, assumedly waiting

for another "Reset" command from the control room://

 Nz=0;

 while(!Remote\_Reset\_Read())//:Count the duration of the control room's

second "Reset" request://

 {

 Nz=Nz+1;

 CyDelay(20);

 if(Nz>350)//:If the second "Reset" Request duration is greater than

 350 loops of the rms\_writing code, goto the main code://

 goto xyz;

 }

 if(Nz<18)//:If the second "Reset" Request duration is less than 18

 loops of the rms\_writing code, goto the main code://

 goto xyz;

 else //: If it's between these two duration times, use its duration

to set the new timing\_constant value://

 {

 timing\_constant=0.005\*Nz;//:the new value of timing\_constant is the

 0.005 times the "Reset" command duration

 divided by 20ms CyDelay://

 for (i=0; i<15; i++)//:Then blink the two bits on and off 15 times to

 signify completion://

 {

 PD\_ER\_Write(1);

 LED\_ER\_Write(1);

 CyDelay(500);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 CyDelay(500);

 }

 goto xyz;//:go to the main code://

 }

 //:Main Code://

 xyz: for (j=0; j<N; j++) //:Loop N times (N currently=63)://

 {

 for (i=0; i<6; i++) //:Loop 6 times, and set the value of the correct ADC PD Input

into the "out" array out[i][j]://

 {

 CyDelay(1);

 LED\_IN1\_Sel\_Write(0);

 LED\_IN2\_Sel\_Write(0);

 LED\_IN3\_Sel\_Write(0);

 LED\_IN4\_Sel\_Write(0);

 LED\_IN5\_Sel\_Write(0);

 LED\_IN6\_Sel\_Write(0);

 PD\_IN1\_Sel\_Write(0);

 PD\_IN2\_Sel\_Write(0);

 PD\_IN3\_Sel\_Write(0);

 PD\_IN4\_Sel\_Write(0);

 PD\_IN5\_Sel\_Write(0);

 PD\_IN6\_Sel\_Write(0);

 CyDelay(1);

 if(i==0)

 PD\_IN1\_Sel\_Write(1);

 if(i==1)

 PD\_IN2\_Sel\_Write(1);

 if(i==2)

 PD\_IN3\_Sel\_Write(1);

 if(i==3)

 PD\_IN4\_Sel\_Write(1);

 if(i==4)

 PD\_IN5\_Sel\_Write(1);

 if(i==5)

 PD\_IN6\_Sel\_Write(1);

 CyDelay(1);

 ADC\_DelSig\_IRQ\_Start();//:Enables interrupts at the end of conversion://

 ADC\_DelSig\_Start();//:Start the ADC://

 ADC\_DelSig\_StartConvert();//:Starts conversion://

 Filter\_Start();//:Starts the filter://

 DMA\_Config();//:Configure the DMA by calling the DMA\_Config function://

 CYGlobalIntEnable;//:Enable Global Interrupts://

 if(ADC\_DelSig\_IsEndConversion(ADC\_DelSig\_WAIT\_FOR\_RESULT))//:If the ADC is

 ready with a

 result://

 {

 out[i][j]=ADC\_DelSig\_GetResult16();//:Set the value of the correct ADC PD

Input into the "out" array

out[i][j]://

 }

 CyDelay(1);

 }

 }

 for (i=0; i<6; i++)//: initialize each of the 6 PD\_rms and mean://

 {

 PD\_rms[i]=0.0;

 mean=0.0;

 for(j=1; j<N; j++)//: calculate the mean value of all of the PD inputs://

 {

 mean=mean+out[i][j];

 }

 mean=mean/(N-1);

 for(j=1; j<N; j++)//: calculate the rms value of each of the PD inputs://

 {

 PD\_rms[i]=PD\_rms[i]+(out[i][j]-mean)\*(out[i][j]-mean);

 }

 PD\_rms[i]=PD\_rms[i]/(N-1);

 PD\_rms[i]=sqrt(PD\_rms[i]);

 rms\_cal[i]=PD\_rms[i];

 PD\_rms[i]=PD\_rms[i]/rms\_ratio;

 PD\_rms[i]=PD\_rms[i]/rms\_constant;

 }

 VDAC\_Start();

 if (PD\_rms[0]>(0.2\*Vpd\_rms\_ref)) //:If the rms voltage

of the first PD is above 6

(=0.2\*Vpd\_rms\_ref(currently 30))://

 {

rms\_cal[0]=0.69\*rms\_cal[0]+0.57\*kf;//:set the rms\_cal of

the first PD to 0.69 times its

old value, plus0.57\*kf (initially 0)://

 kf=rms\_cal[0];//: change kf to this new rms\_cal value

for the next go-round://

 VDAC\_SetValue(rms\_cal[0]); //:send the value of the first pd rms

calculation out to be monitored on the back

panel://

 CyDelay(10);

 }

 else

 {

 VDAC\_SetValue(3);//:If the rms voltage of the first PD

is not above 6, send out a DAC value

of 3 to the back panel://

 CyDelay(10);

 }

 m=0;

 for (i=0; i<6; i++)//:Check if any of the PD\_rms values are greater than the reference

voltage://

 {

 if (PD\_rms[i]>Vpd\_rms\_ref)

 {

 m=1;//:If any are greater, set the "m" error bit to one://

 }

 else

 m=m;//:otherwise, leave "m" alone (awkward coding)://

 }

 if(m==1)//:If the error bit is set, write out the PD \_ER bit://

 {

 PD\_ER\_Write(1);

 CyDelay(1);

 }

 else//:Otherwise, don't://

 {

 PD\_ER\_Write(0);

 CyDelay(1);

 }

 X2: for (i=0; i<6; i++)//:Loop 6 times, and compare the value of the correct ADC LED Input

 to the DAC value of 2640mV from "DAC"://

 {

 Pin\_Init();

 DAC\_Start();

 Comp1\_Start();

 CyDelay(1);

 if(i==0)

 LED\_IN1\_Sel\_Write(1);

 if(i==1)

 LED\_IN2\_Sel\_Write(1);

 if(i==2)

 LED\_IN3\_Sel\_Write(1);

 if(i==3)

 LED\_IN4\_Sel\_Write(1);

 if(i==4)

 LED\_IN5\_Sel\_Write(1);

 if(i==5)

 LED\_IN6\_Sel\_Write(1);

 CyDelay(1);

 if (Comp1\_GetCompare())//:If any are less than the DAC value, set the "k" error bit to

one://

 {

 k=1;

 }

 else//:otherwise, leave "k" alone (awkward coding)://

 {

 k=k;

 }

 }

 if (k==1)//:If the error bit is set, and the error timer is not zero, write out

the LED \_ER bit://

 {

 if(l==0)

 {

 LED\_ER\_Write(0);

 }

 else

 { LED\_ER\_Write(1); }

 }

 else

 {

 LED\_ER\_Write(0);//:Otherwise, don't://

 }

 X3: if((m==1) || (k==1))//:If the PD rms error, OR the LED error have been thrown,

 start the "l" counter, and reset the "m" and "k"

 errors://

 {

 l=l+1;

 m=0;

 k=0;

 }

 else//:Otherwise, reset "l" to 0://

 {

 l=0;

 }

 X4: if(l>(687\*timing\_constant))//:if "l" has become bigger that 687 loops, times

 the timing\_constant (currently 1), throw the HEPI

 error. This takes about 20 min.)://

 {

 HEPI\_ERROR\_Write(1);

 goto X5;//:Then proceed to the next step://

 }

 else//:If "l" is less than 687 loops, reset the HEPI error, and go back to the

 very beginning://

 {

 HEPI\_ERROR\_Write(0);

 goto X1;

 }

 X5:

 if (l>(1330\*timing\_constant))//:if "l" has become bigger than 1330 loops of

the code below, times the timing\_constant

(currently 1),throw the SUS error, and set

"l" to 3000. This takes about 40 min.)://

 {

 SUS\_ERROR\_Write(1);

 l=3000;

 }

 else

 {

 SUS\_ERROR\_Write(0);

 }

 //:the following (almost identical) code runs during a HEPI fault or SUS fault://

 X6: for (j=0; j<N; j++)//:Loop N times (N currently=63)://

 {

 for (i=0; i<6; i++)//:Loop 6 times, and set the value of the correct ADC PD Input

into the "out" array out[i][j]://

 {

 CyDelay(1);

 LED\_IN1\_Sel\_Write(0);

 LED\_IN2\_Sel\_Write(0);

 LED\_IN3\_Sel\_Write(0);

 LED\_IN4\_Sel\_Write(0);

 LED\_IN5\_Sel\_Write(0);

 LED\_IN6\_Sel\_Write(0);

 PD\_IN1\_Sel\_Write(0);

 PD\_IN2\_Sel\_Write(0);

 PD\_IN3\_Sel\_Write(0);

 PD\_IN4\_Sel\_Write(0);

 PD\_IN5\_Sel\_Write(0);

 PD\_IN6\_Sel\_Write(0);

 CyDelay(1);

 if(i==0)

 PD\_IN1\_Sel\_Write(1);

 if(i==1)

 PD\_IN2\_Sel\_Write(1);

 if(i==2)

 PD\_IN3\_Sel\_Write(1);

 if(i==3)

 PD\_IN4\_Sel\_Write(1);

 if(i==4)

 PD\_IN5\_Sel\_Write(1);

 if(i==5)

 PD\_IN6\_Sel\_Write(1);

 CyDelay(1);

 ADC\_DelSig\_IRQ\_Start();//:Enables interrupts at the end of conversion://

 ADC\_DelSig\_Start();//:Start the ADC://

 ADC\_DelSig\_StartConvert();//:Starts conversion://

 Filter\_Start();//:Starts the filter://

 DMA\_Config//:Configure the DMA by calling the DMA\_Config function://

 CYGlobalIntEnable;//:Enable Global Interrupts://

 if(ADC\_DelSig\_IsEndConversion(ADC\_DelSig\_WAIT\_FOR\_RESULT)) //:If the ADC is

ready with a result://

 {

 out[i][j]=ADC\_DelSig\_GetResult16();//:Set the value of the correct ADC PD

Input into the "out" array

out[i][j]://

 }

 CyDelay(1);

 }

 }

 for (i=0; i<6; i++)//: initialize each of the 6 PD\_rms and mean://

 {

 PD\_rms[i]=0.0;

 mean=0.0;

 for(j=1; j<N; j++)//: calculate the mean value of all of the PD inputs://

 {

 mean=mean+out[i][j];

 }

 mean=mean/(N-1);

 for(j=1; j<N; j++)//: calculate the rms value of each of the PD inputs://

 {

 PD\_rms[i]=PD\_rms[i]+(out[i][j]-mean)\*(out[i][j]-mean);

 }

 PD\_rms[i]=PD\_rms[i]/(N-1);

 PD\_rms[i]=sqrt(PD\_rms[i]);

 rms\_cal[i]=PD\_rms[i];

 PD\_rms[i]=PD\_rms[i]/rms\_ratio;

 PD\_rms[i]=PD\_rms[i]/rms\_constant;

 }

 VDAC\_Start();

 if (PD\_rms[0]>(0.2\*Vpd\_rms\_ref))//:If the rms voltage of

the first PD is above

6(=0.2\*Vpd\_rms\_ref

(currently 30))://

 {

 rms\_cal[0]=0.69\*rms\_cal[0]+0.57\*kf;//:set the rms\_cal of

the first PD to 0.69 times

its old value, plus0.57\*kf

(set earlier, on line 299)://

 kf=rms\_cal[0];//:change kf to this new rms\_cal value

 for the next go-round://

 VDAC\_SetValue(rms\_cal[0]);//:send the value of the first pd rms

calculation out to be monitored on the back

panel://

 CyDelay(10);

 }

 else

 {

 VDAC\_SetValue(3);//:If the rms voltage of the first PD

is not above 6, send out a DAC value

of 3 to the back panel://

 CyDelay(10);

 }

 m=0;

 for (i=0; i<6; i++)//:Check if any of the PD\_rms values are greater than the reference

 voltage://

 {

 if (PD\_rms[i]>Vpd\_rms\_ref)//:If any are greater, set the "m" error bit to one://

 {

 m=1;

 }

 else

 m=m;//:Otherwise, don't://

 }

 for (i=0; i<6; i++)//:Loop 6 times, and compare the value of the correct ADC

 LED Input to the DAC value of 2640mV from "DAC"://

 {

 CyDelay(1);

 Pin\_Init();

 CyDelay(5);

 if(i==0)

 LED\_IN1\_Sel\_Write(1);

 if(i==1)

 LED\_IN2\_Sel\_Write(1);

 if(i==2)

 LED\_IN3\_Sel\_Write(1);

 if(i==3)

 LED\_IN4\_Sel\_Write(1);

 if(i==4)

 LED\_IN5\_Sel\_Write(1);

 if(i==5)

 LED\_IN6\_Sel\_Write(1);

 DAC\_Start();

 Comp1\_Start();

 CyDelay(1);

 if (LED\_info\_Read())

 if (Comp1\_GetCompare())//:If any are less than the DAC value, set the "k" error bit

to one://

 k=1;

 else

 k=k;//:Otherwise, don't do anything, right here.://

 }

 if((m==1) || (k==1))//:If the PD rms error, OR the LED error have been

 thrown, continue the "l" counter://

 {

 l=l+1;

 }

 else

 {

 l=l;//:Otherwise, don't do anything, right here.://

 }

 if (m==0)//:Then, by logic, k==1, right?://

 PD\_ER\_Write(0);

 if (m==1)

 PD\_ER\_Write(1);//:If the "m" error bit is set, write out the PD \_ER

 bit://

 if (k==0)

 LED\_ER\_Write(0);

 if (k==1)

 LED\_ER\_Write(1);//:I think you get this, by now.://

 if(((!Remote\_Reset\_Read()) || Reset\_rqst\_in\_Read()) & (m==0) & (k==0))

 { //:If there is a "Reset" request, and no lingering errors, go back to

 the very beginning (after sitting here for 2 seconds, of course.://

 CyDelay(2000);

 l=0;//:Reset the timer://

 goto X0;

 }

 else//:Otherwise, reset the errors, and go back to the "we're inside the

HEPI fault" loop://

 {

 m=0;

 k=0;

 goto X5;

 }

 }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name: Pin\_Init

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Summary:

\* Initializes Hardware MUX channels

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

 void Pin\_Init(void)

 {

 CyDelay(10);

 LED\_IN1\_Sel\_Write(0);

 LED\_IN2\_Sel\_Write(0);

 LED\_IN3\_Sel\_Write(0);

 LED\_IN4\_Sel\_Write(0);

 LED\_IN5\_Sel\_Write(0);

 LED\_IN6\_Sel\_Write(0);

 PD\_IN1\_Sel\_Write(0);

 PD\_IN2\_Sel\_Write(0);

 PD\_IN3\_Sel\_Write(0);

 PD\_IN4\_Sel\_Write(0);

 PD\_IN5\_Sel\_Write(0);

 PD\_IN6\_Sel\_Write(0);

 CyDelay(1);

 }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name: DMA\_Config

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Summary:

\* Initializes and sets up DMA for use

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void DMA\_Config(void)

{

 /\* Declare variable to hold the handle for DMA channel \*/

 uint8 channelHandle;

 /\* Declare DMA Transaction Descriptor for memory transfer into

 \* High Pass Filter Channel.

 \*/

 uint8 tdChanA;

 /\* Configure the DMA to Transfer the data in 1 burst with individual trigger

 \* for each burst.

 \*/

 channelHandle = DMA\_DmaInitialize(BYTES\_PER\_BURST, REQUEST\_PER\_BURST,

 HI16(UPPER\_SRC\_ADDRESS), HI16(UPPER\_DEST\_ADDRESS));

 tdChanA = CyDmaTdAllocate();

 CyDmaTdSetConfiguration(tdChanA, 1, DMA\_INVALID\_TD, 0);

 /\* Set the source address as ADC\_DelSig and the destination as

 \* Filter Channel A. \*/

 CyDmaTdSetAddress(tdChanA, LO16((uint32)ADC\_DelSig\_DEC\_SAMP\_PTR), LO16((uint32)Filter\_STAGEAH\_PTR));

 /\* Set tdChanA to be the initial TD associated with channelHandle \*/

 CyDmaChSetInitialTd(channelHandle, tdChanA);

 /\* Enable the DMA channel represented by channelHandle and preserve the TD \*/

 CyDmaChEnable(channelHandle, 1);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name: Zero\_Init

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Summary:

\* Initializes Output Variables upon startup

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

 void Zero\_Init(void)

 {

 HEPI\_ERROR\_Write(0);

 SUS\_ERROR\_Write(0);

 PD\_ER\_Write(0);

 LED\_ER\_Write(0);

 }