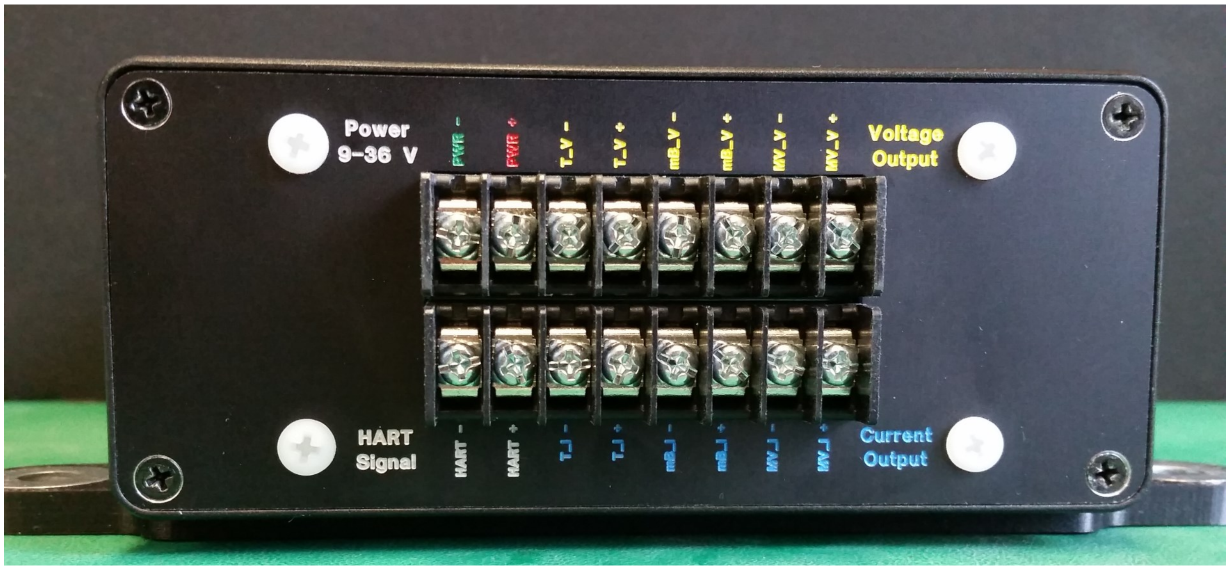
Water Vapor Monitor System Quick Start Guide

NASA Glenn Research Center

Larry Greer - 8/17/2018

Setup and Operating Instructions

Attach the power supply to the (PWR- and PWR+) terminals. Due to an unforeseen change in the circuit power distribution, the input range **IS NOT THE SAME** as the the range indicated on the panel. The correct input range is **9 - 30 volts** (the panel labelling will be changed on subsequent units). The top row of the terminal connector breaks out the isolated DAC voltage outputs for the water vapor (MV_V^+, MV_V^-), pressure (mB_V^+, mB_V^-) and temperature (T_V^+, T_V^-) sensors. The bottom row of the terminal connector breaks out the isolated DAC current outputs for the water vapor (MV_I^+, MV_I^-), pressure (mB_I^+, mB_I^-) and temperature (T_I^+, T_I^-) sensors. The V^+ and I^+ terminals are the outputs for the corresponding sensors and the V^- and I^- terminals are the returns. The voltage and current outputs both track the corresponding sensor values as shown in table 1. Lastly, an external HART signal (500mVpp, 1200Hz to 2200Hz) can be applied to the $HART^-$ and $HART^+$ terminals.



Terminal connector blocks for power, DAC voltage & current outputs and HART signal

		Operating Conditions			
DAC Outputs		Normal Operation	$Sensor_{Value} < Min_{Value}$	$Sensor_{Value} > Max_{Value}$	Lost WVSS-II Communication
Water Vapor	I_{out}	$\left(16.0_{mA}\frac{ppmv_{MeasuredValue} - ppmv_{MinValue}}{ppmv_{MaxValue} - ppmv_{MinValue}}\right) + 4.0_{mA}$	2_{mA}	2_{mA}	2_{mA}
	V_{out}	$5.0_{volts}\frac{ppmv_{MeasuredValue} - ppmv_{MinValue}}{ppmv_{MaxValue} - ppmv_{MinValue}}$	0_V	5_V	5.5_V
Pressure	I_{out}	$\left(16.0_{mA}\frac{mbar_{MeasuredValue} - mbar_{MinValue}}{mbar_{MaxValue} - mbar_{MinValue}}\right) + 4.0_{mA}$	2_{mA}	2_{mA}	2_{mA}
	V_{out}	$5.0_{volts}\frac{mbar_{MeasuredValue} - mbar_{MinValue}}{mbar_{MaxValue} - mbar_{MinValue}}$	0_V	5_V	5.5_V
Temperature	I_{out}	$\left(16.0_{mA}\frac{{}^\circ C_{MeasuredValue} - {}^\circ C_{MinValue}}{{}^\circ C_{MaxValue} - {}^\circ C_{MinValue}}\right) + 4.0_{mA}$	2_{mA}	2_{mA}	2_{mA}
	V_{out}	$5.0_{volts}\frac{{}^\circ C_{MeasuredValue} - {}^\circ C_{MinValue}}{{}^\circ C_{MaxValue} - {}^\circ C_{MinValue}}$	0_V	5_V	5.5_V

$ppmv_{MaxValue}$ = 40,000 ppmv	$mbar_{MaxValue}$ = 1016 mbar	$°C_{MaxValue}$ = 40 °
$ppmv_{MinValue}$ = 50 ppmv	$mbar_{MinValue}$ = 150 mbar	$°C_{MinValue}$ = -15 °C

Table 1 - DAC voltage & current output matrix

The water vapor monitor system accepts data from the WVSS-II unit in the format specified by table 2, which possesses a total of 74 bytes of ACSII data plus a carriage return and line feed for a grand total of 76 bytes per data packet. The data is space separated with left-justified padding to keep the packet size constant.

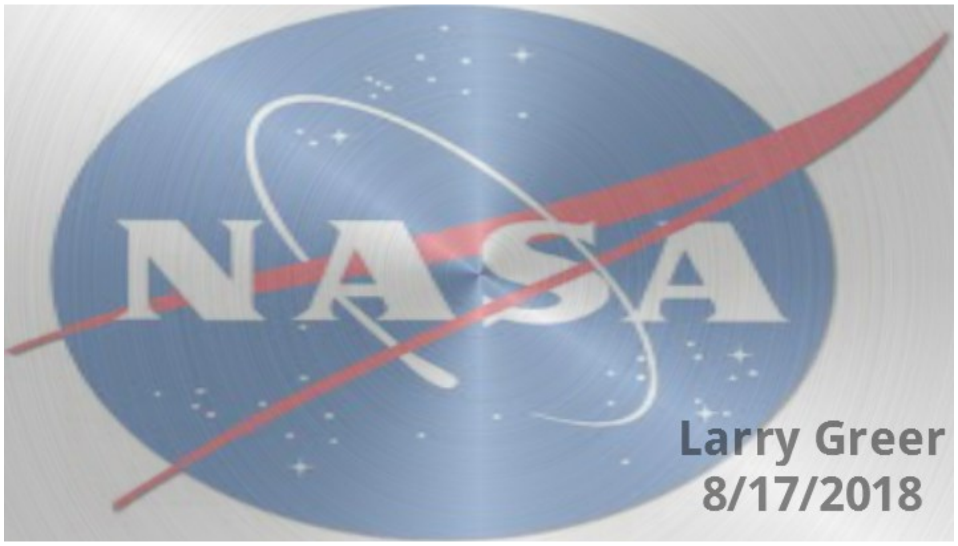
Water Vapor Measurement: WWWWW.WW (WV Concentration in ppmv)
Pressure: PPPP.P (Pressure inside sample cell in mbar)
Temperature: ±TT.TT (Temperature of sample cell in degrees C)
PP2F: XXXXXX (PP2F in digital counts)
LasSigPow: XXXXXX (power of laser signal in digital counts)
Peak Index: XXX (index of peak in digital counts)
Null Value: XXX (null value in digital counts)
MidPt: XX.XX (midpoint in mAmp)
Adj MidPt: XX.XX (adjusted midpoint in mAmp, if peak tracking turned on)
PT: X.XXXXXX (unitless Pressure/Temperature correction factor, PT)
BLCF: X.XXXXXX (unitless Beers Law correction factor for non-linearity)

Table 2 - WVSS-II ASCII Data Packet

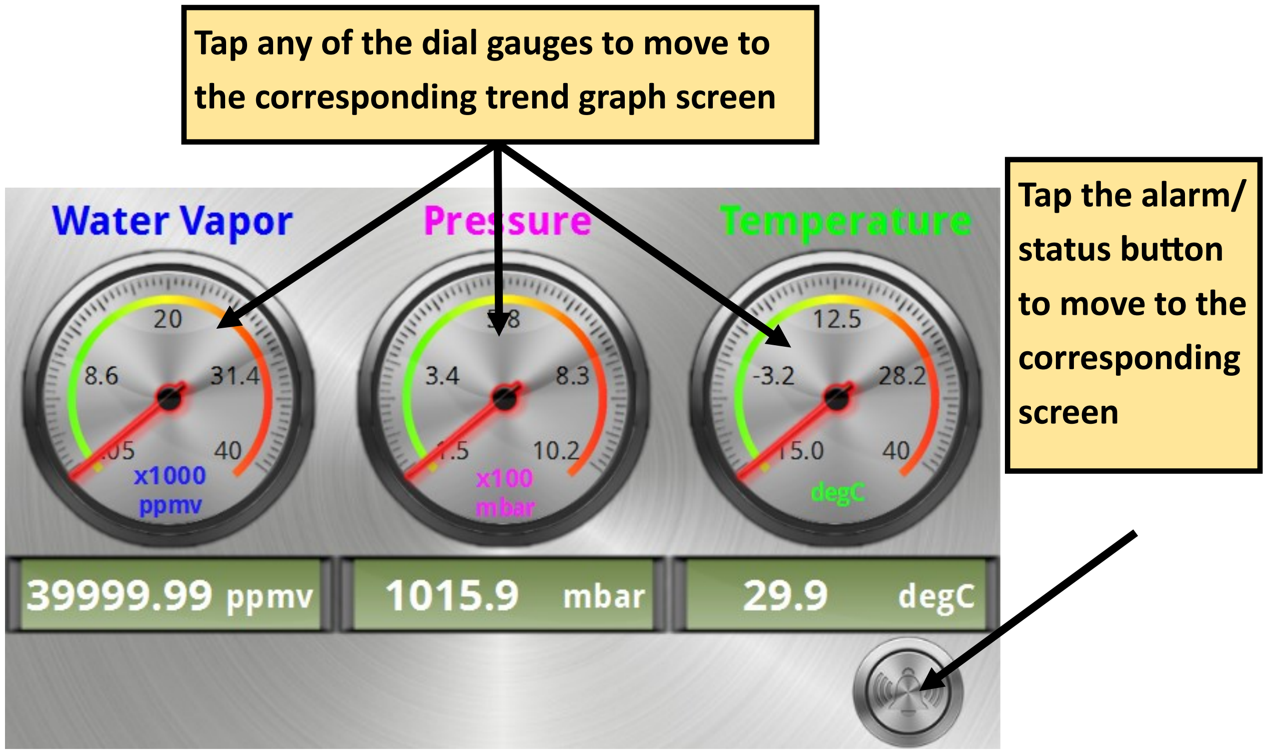
The PC communications port is a pass-through to allow the WVSS-II data to be channeled to a separate computer running the application software for the WVSS-II.

Isolated serial Connectors for WVSS-II and PC

Touch Screen Operation

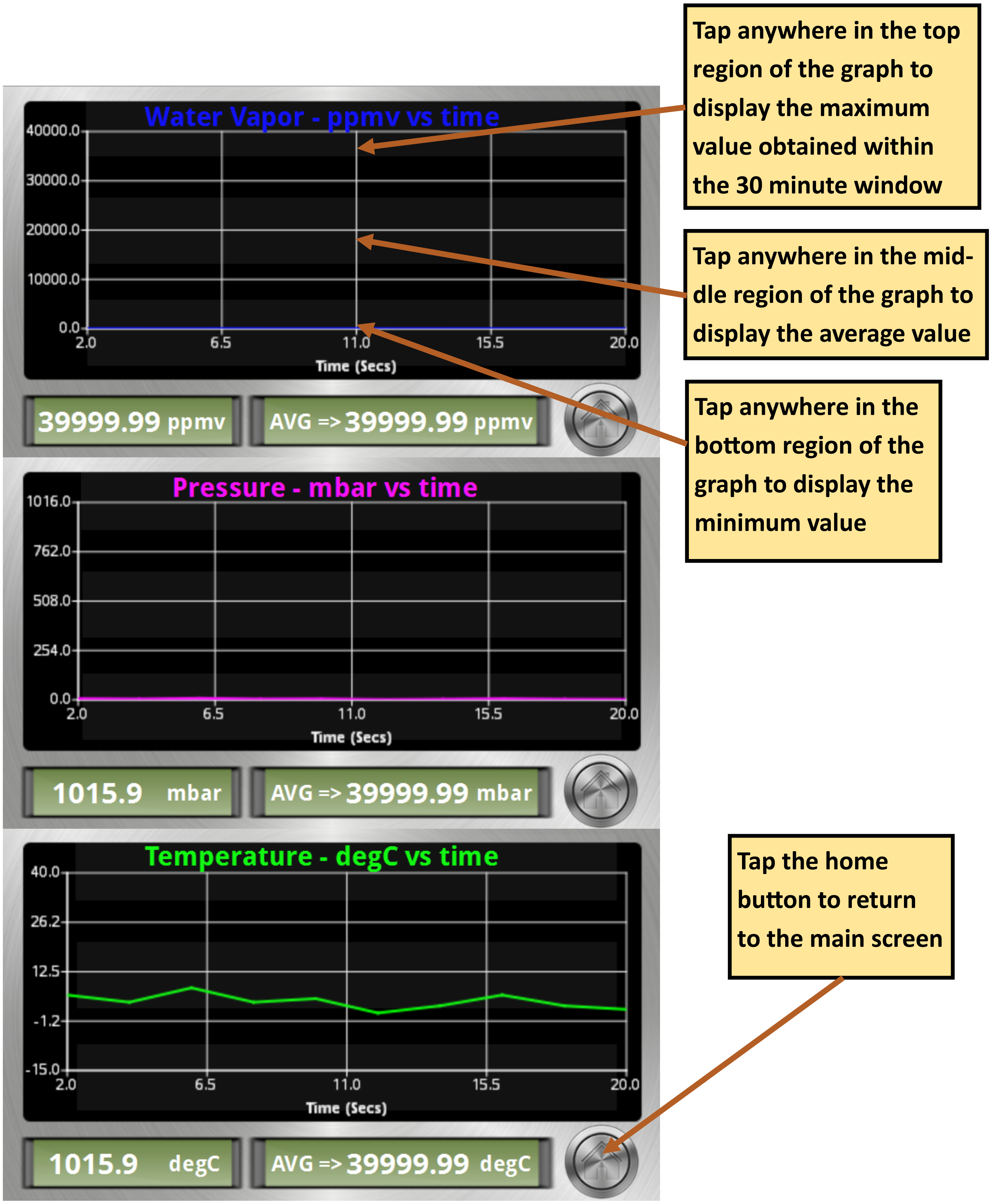


When power is applied to the unit, the Panel Pilot display takes a few seconds to boot-up. Then it displays an initial screen which stays on for approximately 3 seconds. Afterward, it transitions to the main screen.


Tap any of the dial gauges to move to the corresponding trend graph screen
Tap the alarm/ status button to move to the corresponding screen
Water Vapor
Pressure
Temperature
x1000 ppmv
x100 mbar
degC
39999.99 ppmv
1015.9 mbar
29.9 degC

The main screen displays the water vapor concentration, air pressure and temperature on both the analog dials and the digital displays. If the unit loses communication with the WVSS-II sensor, all digital displays show “No Data” and flash red. Any individual display will also flash red when the boundaries for its corresponding measurement are exceeded. Furthermore, if a minimum boundary is violated the analog needle is “pegged” at the low end. The needle is “pegged” and the high end if a maximum boundary is violated. The user can navigate to all other available screens from this primary screen.

The trend graph screens display up to 5 minutes of water vapor, pressure and temperature data vs time captured at 3 second intervals. Thirty minutes of data is buffered for each of the measurements for calculating statistics: minimum value, maximum value and average value. The default statistic shown is the average value, but tapping the top, middle or bottom sections of any trend graph will display the other statistical values.

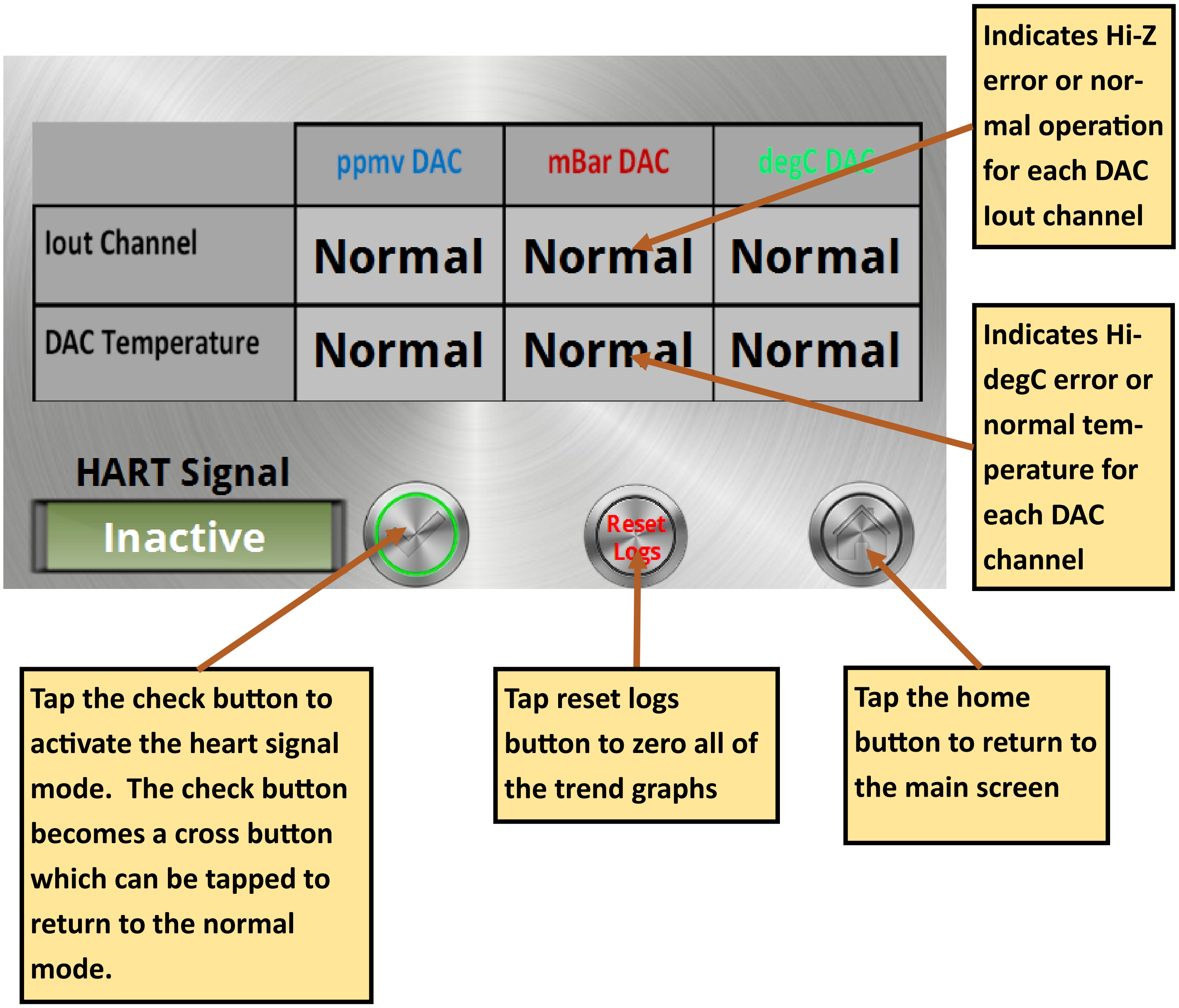


Tap anywhere in the top region of the graph to display the maximum value obtained within the 30 minute window

Tap anywhere in the middle region of the graph to display the average value

Tap anywhere in the bottom region of the graph to display the minimum value

Tap the home button to return to the main screen



	ppmv DAC	mBar DAC	degC DAC
Iout Channel	Normal	Normal	Normal
DAC Temperature	Normal	Normal	Normal

HART Signal

Inactive

Indicates Hi-Z error or normal operation for each DAC Iout channel

Indicates Hi-degC error or normal temperature for each DAC channel

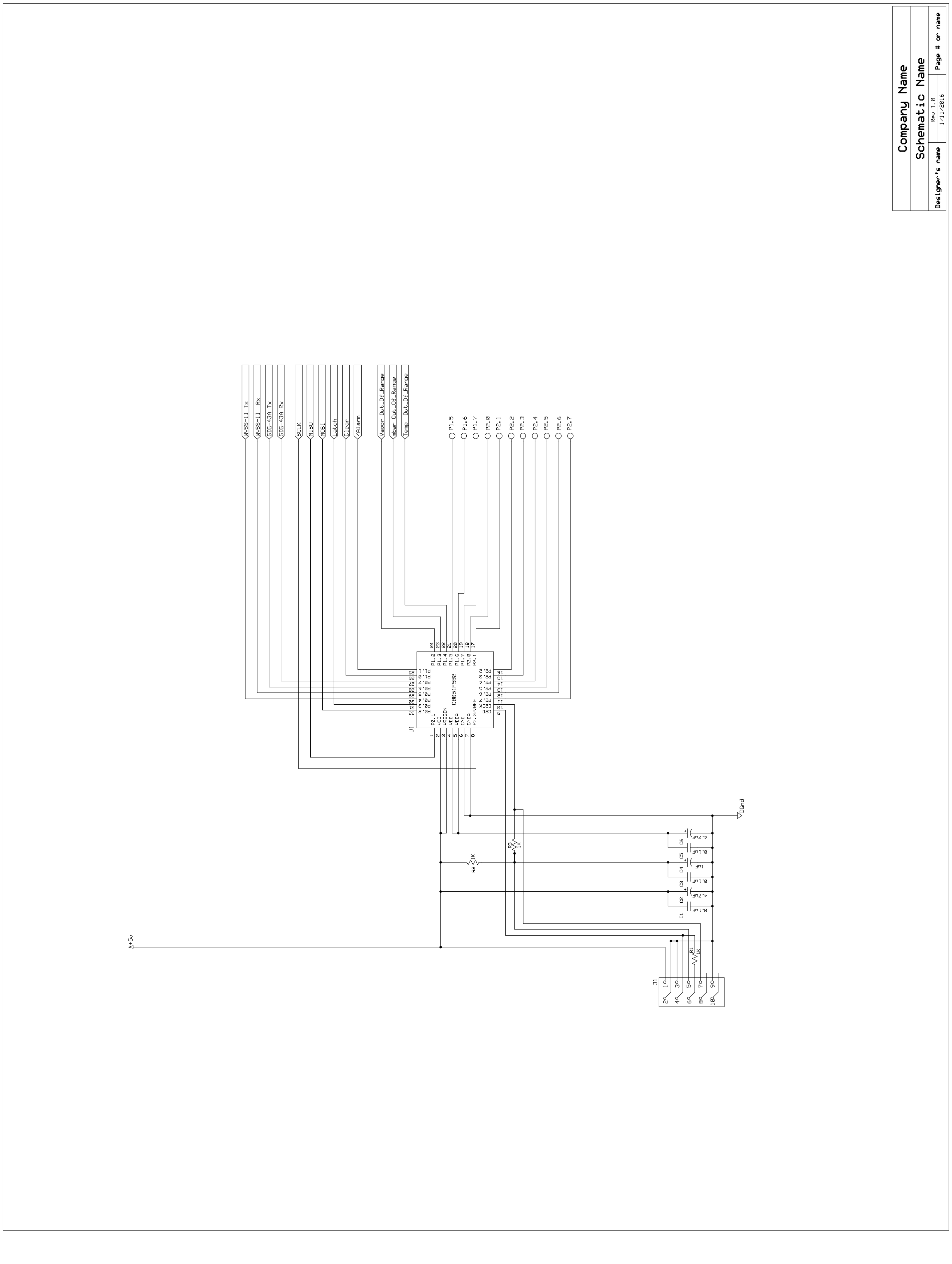
Tap the check button to activate the heart signal mode. The check button becomes a cross button which can be tapped to return to the normal mode.

Tap reset logs button to zero all of the trend graphs

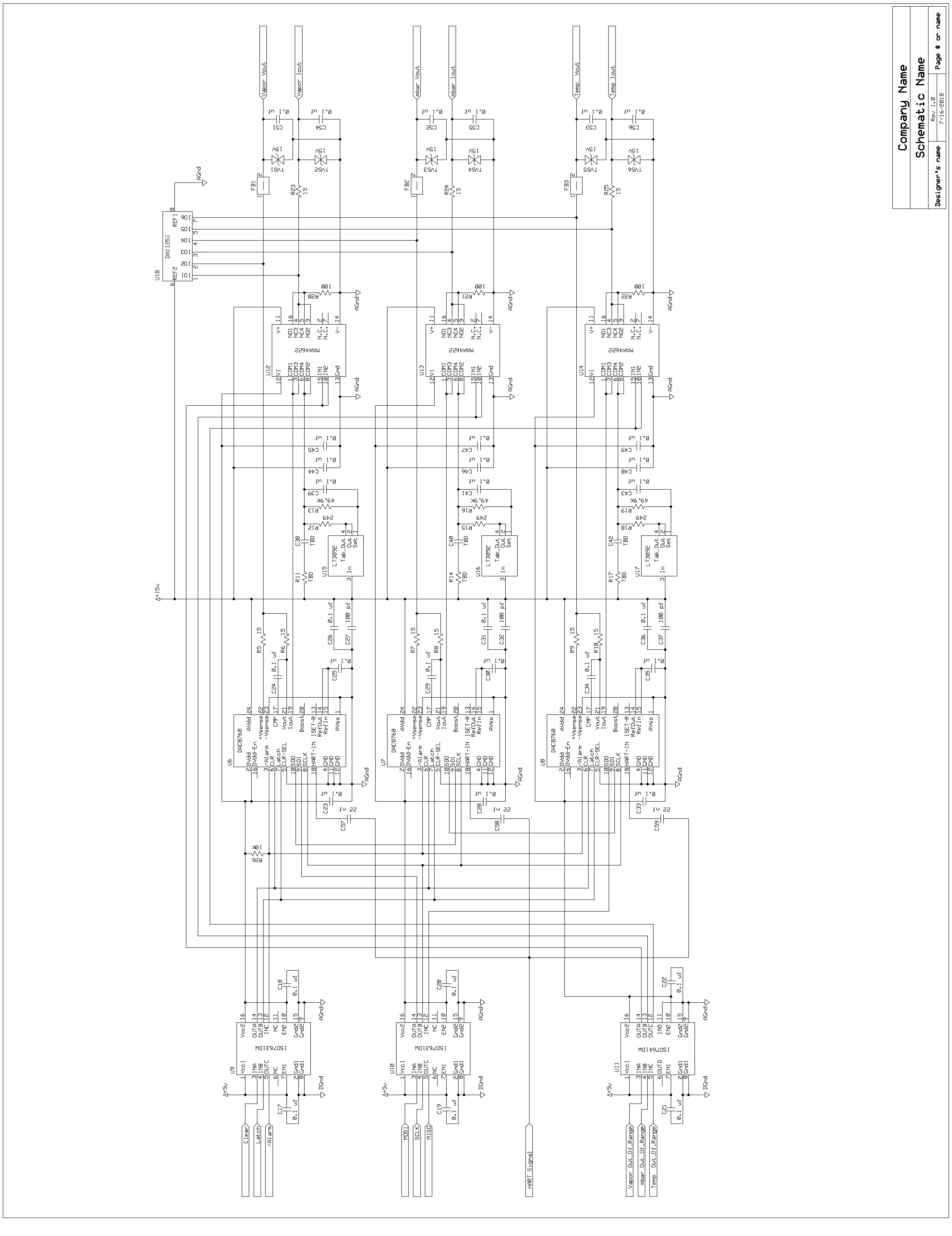
Tap the home button to return to the main screen

The alarm/status screen displays the operating conditions for each of the DAC channels. If any of the current outputs "Iout" are open or have to high a resistance, a "Hi-Z" message is displayed in the corresponding box on the table. If any of the DAC channels exceed their normal operating temperature, a "Hi-degC" message is displayed in the corresponding box on the table. Also the heart signal mode can be activated or de-activated by using the check/cross button. The current heart signal operation mode is also displayed on the screen as either active or inactive. Furthermore, the trend graphs can be reset to a zero start time and empty buffers by tapping the "reset logs" button. The button temporarily turns green to indicate that the request was processed. Lastly, the user can return to the main screen by tapping the home button which exists on all of the ancillary screens.

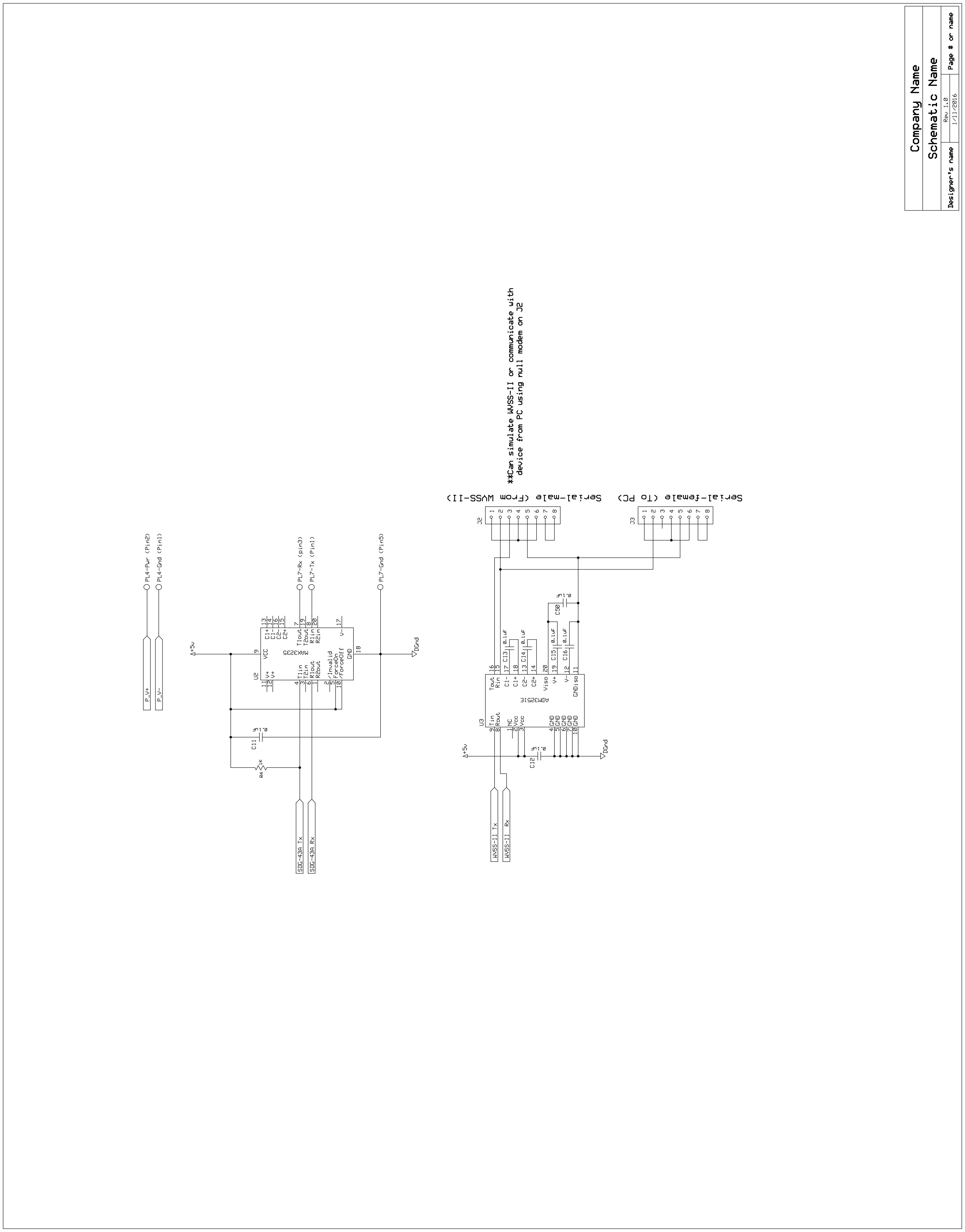
Microcontroller Schematic



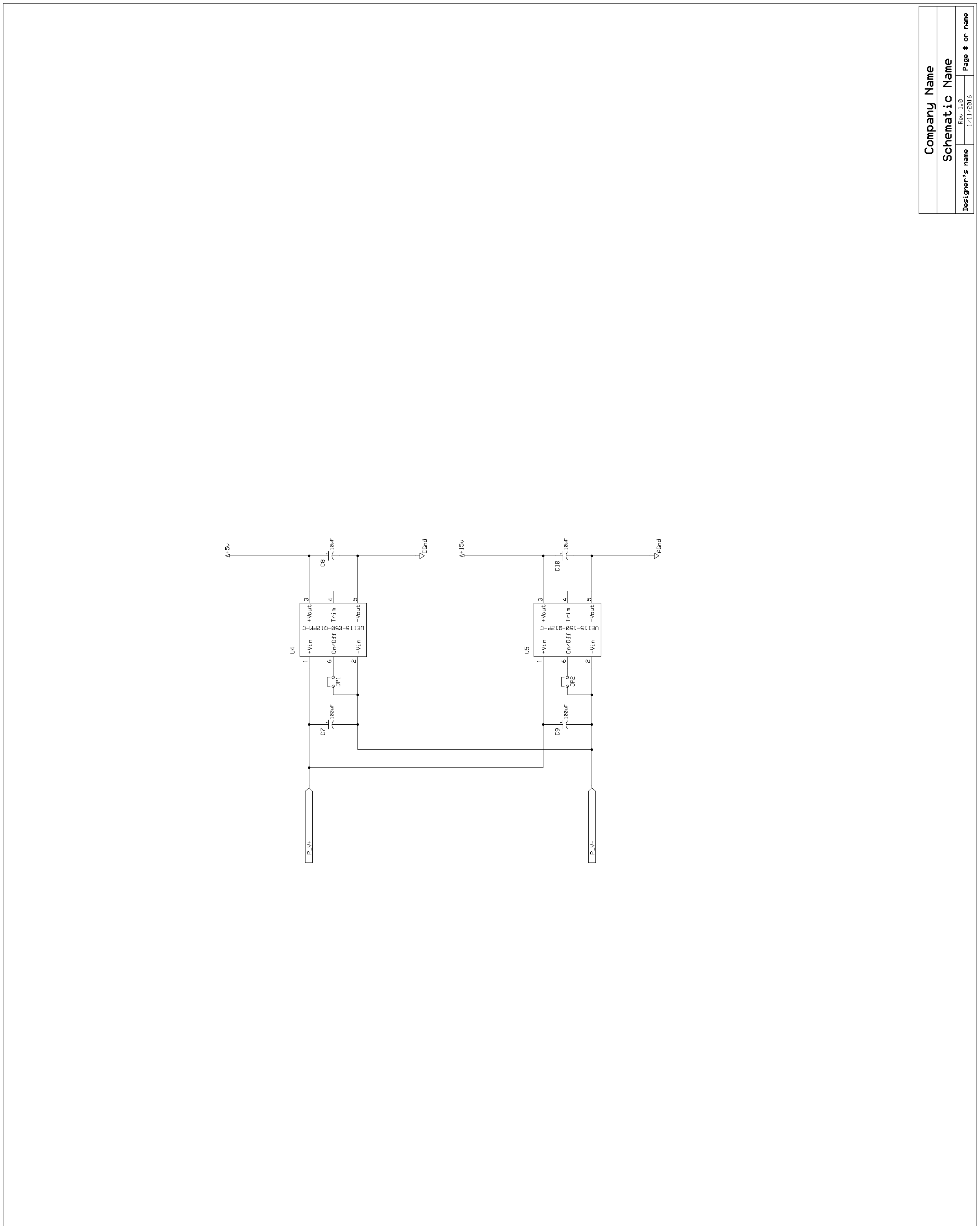
Isolated Voltage/Current DAC Schematic



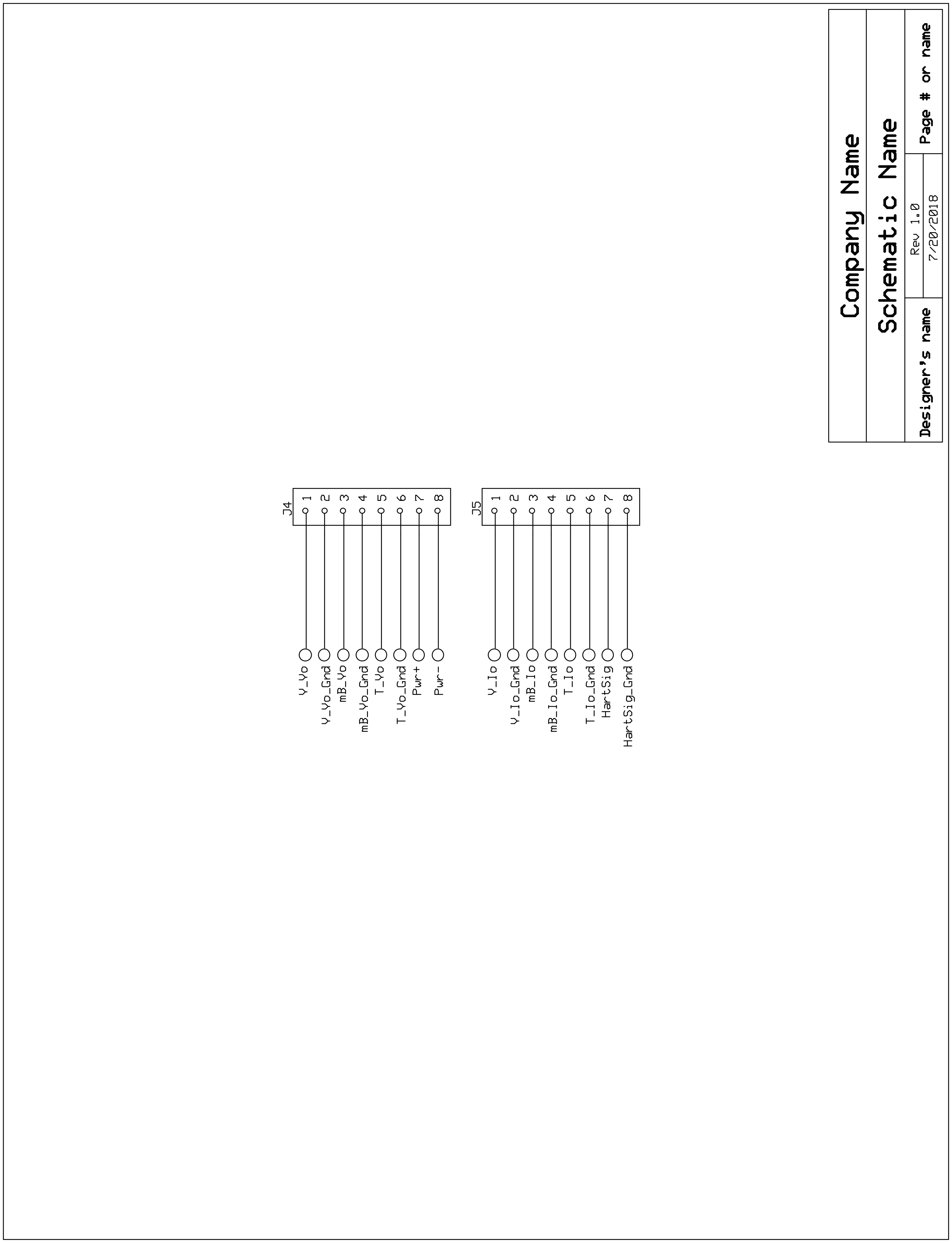
RS-232 Transceiver Schematic



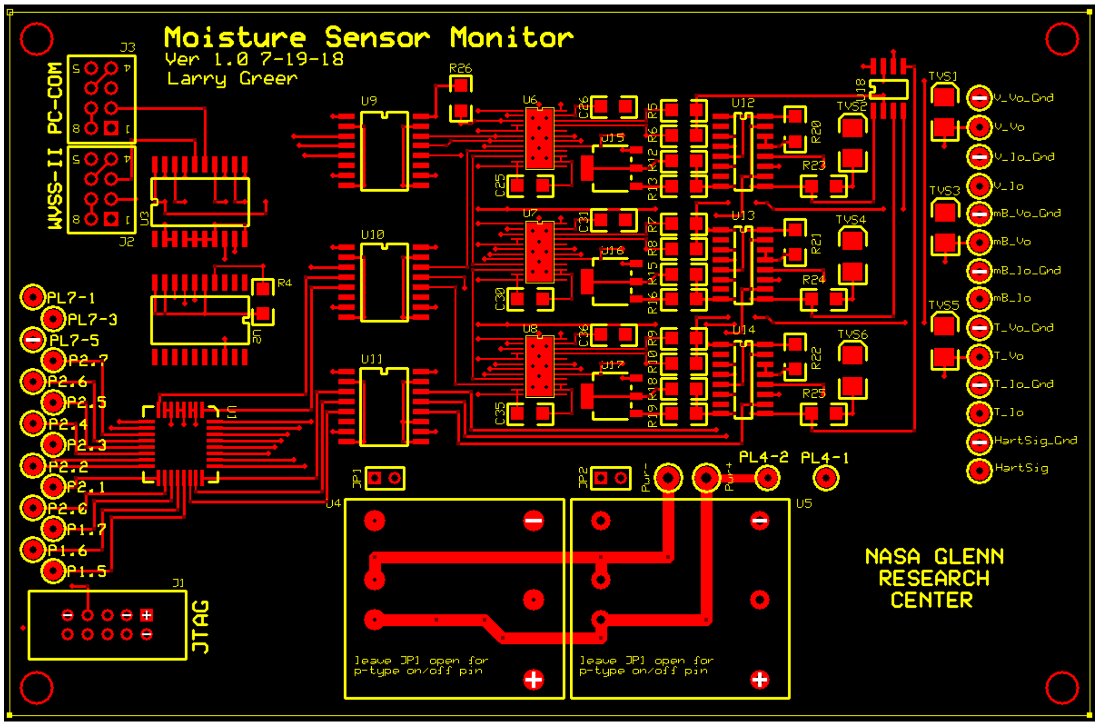
Power Supply Schematic



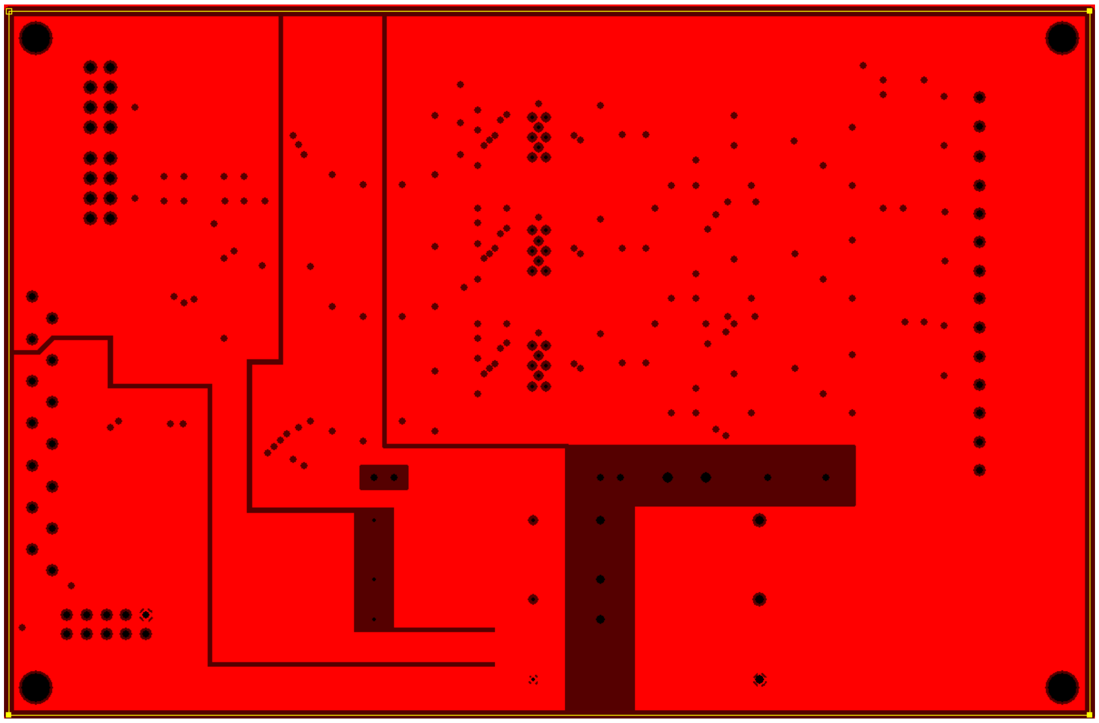
Output Connector Schematic



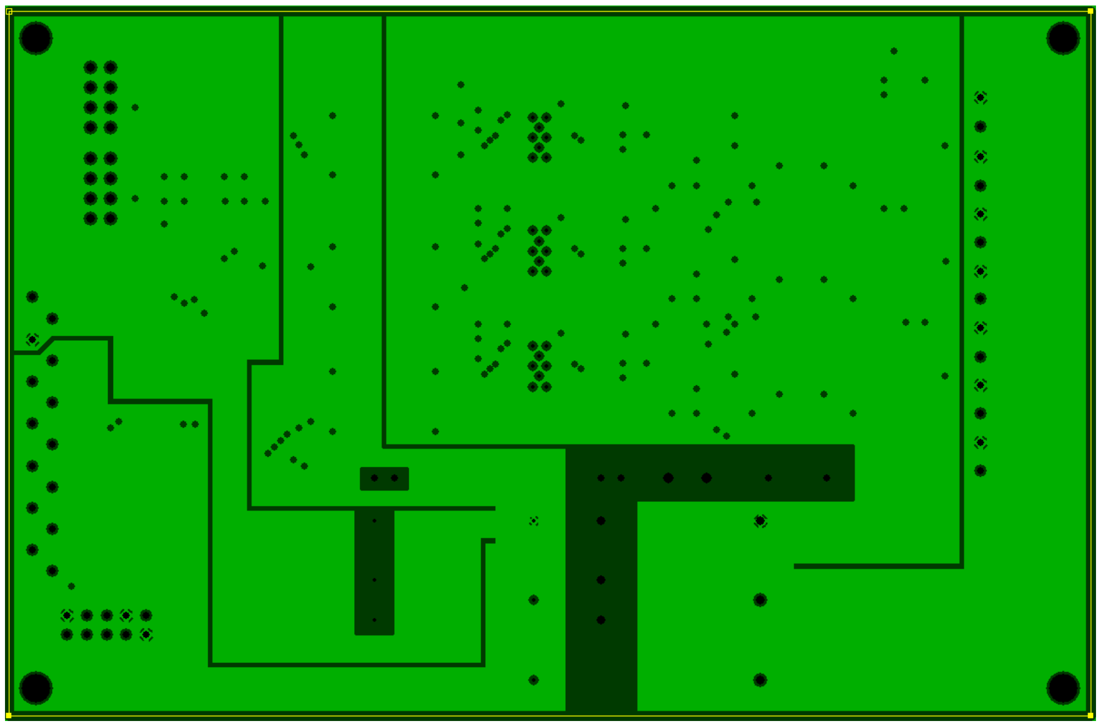
Moisture Sensor Monitor PCB - Top Plane



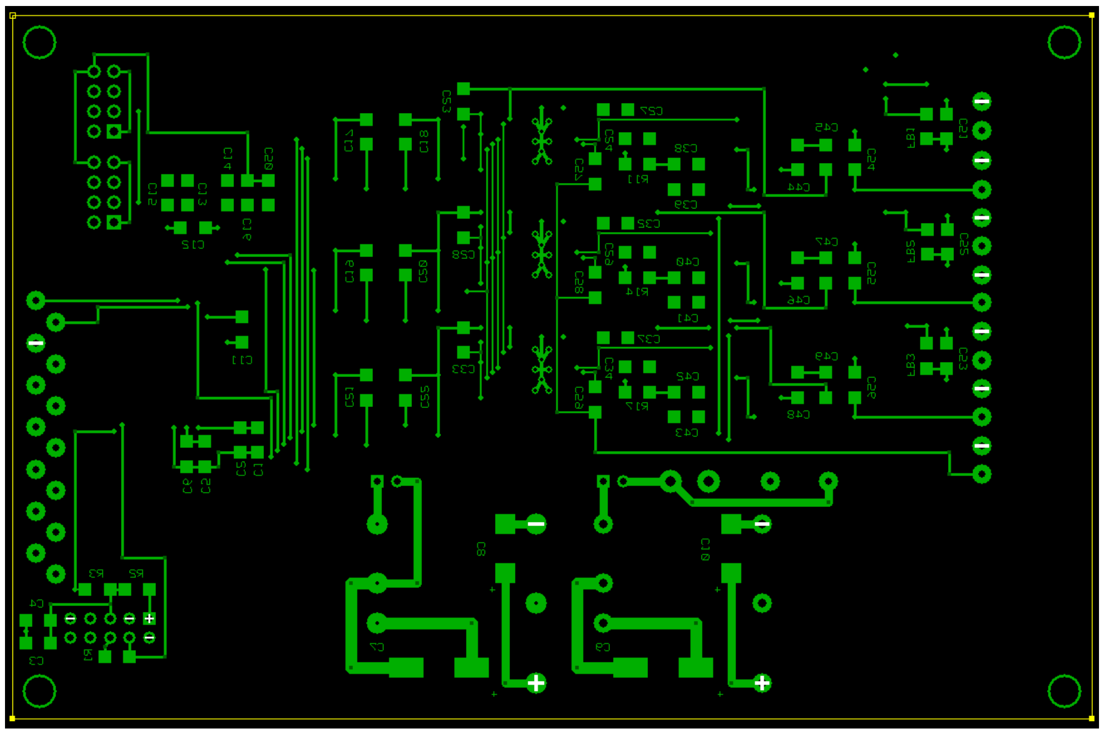
Moisture Sensor Monitor PCB - Power Plane



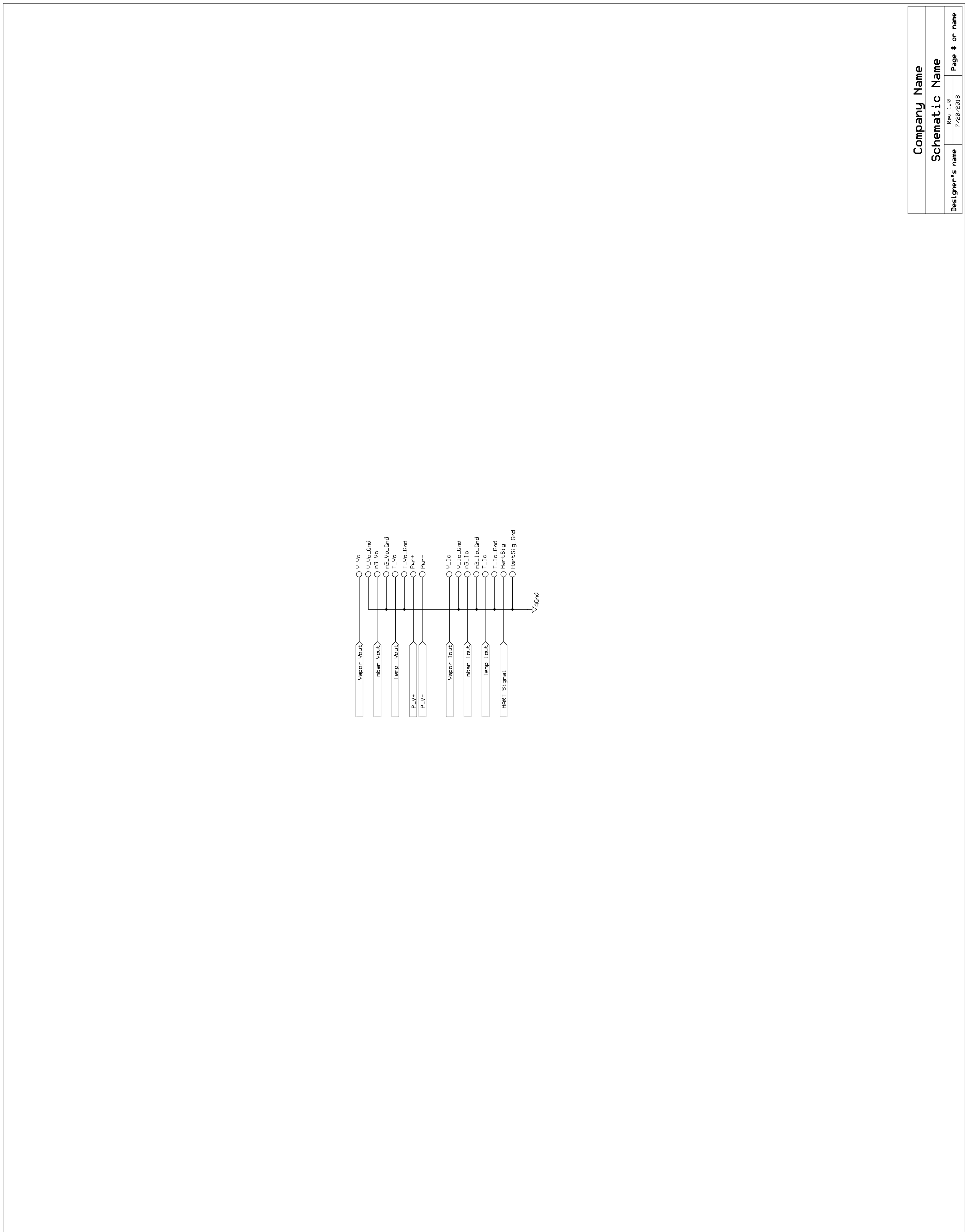
Moisture Sensor Monitor PCB - Ground Plane



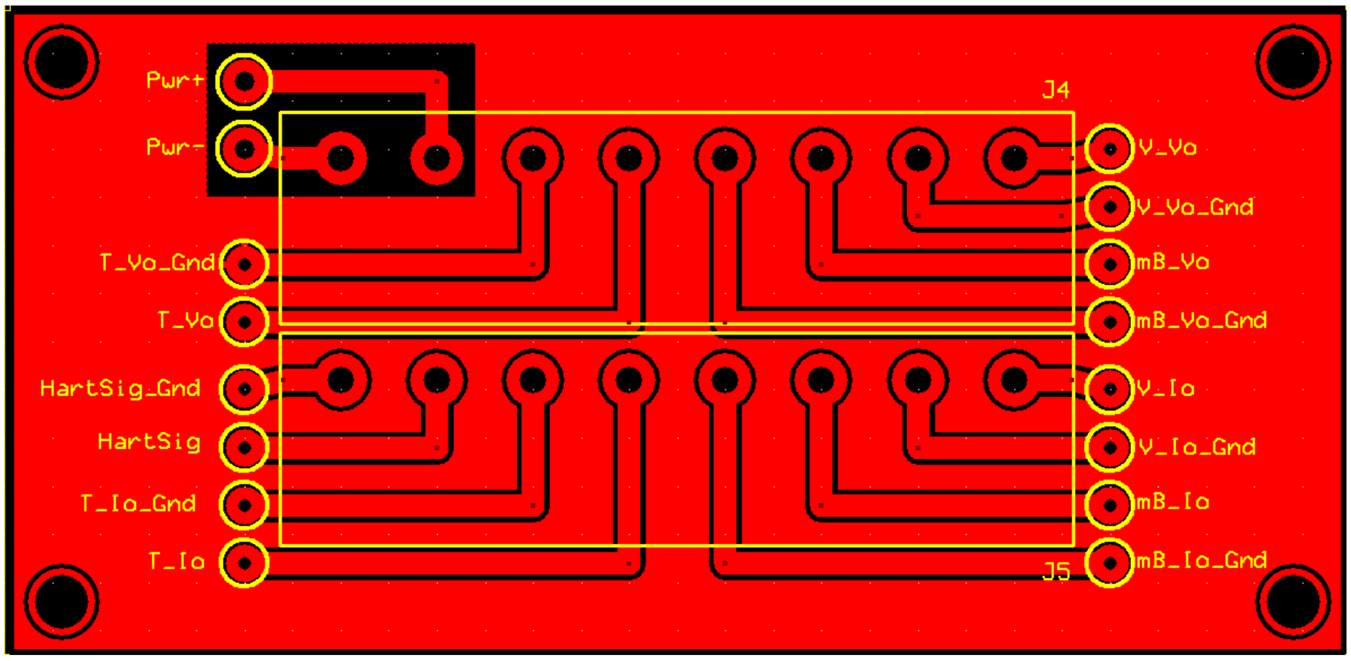
Moisture Sensor Monitor PCB - Bottom Plane



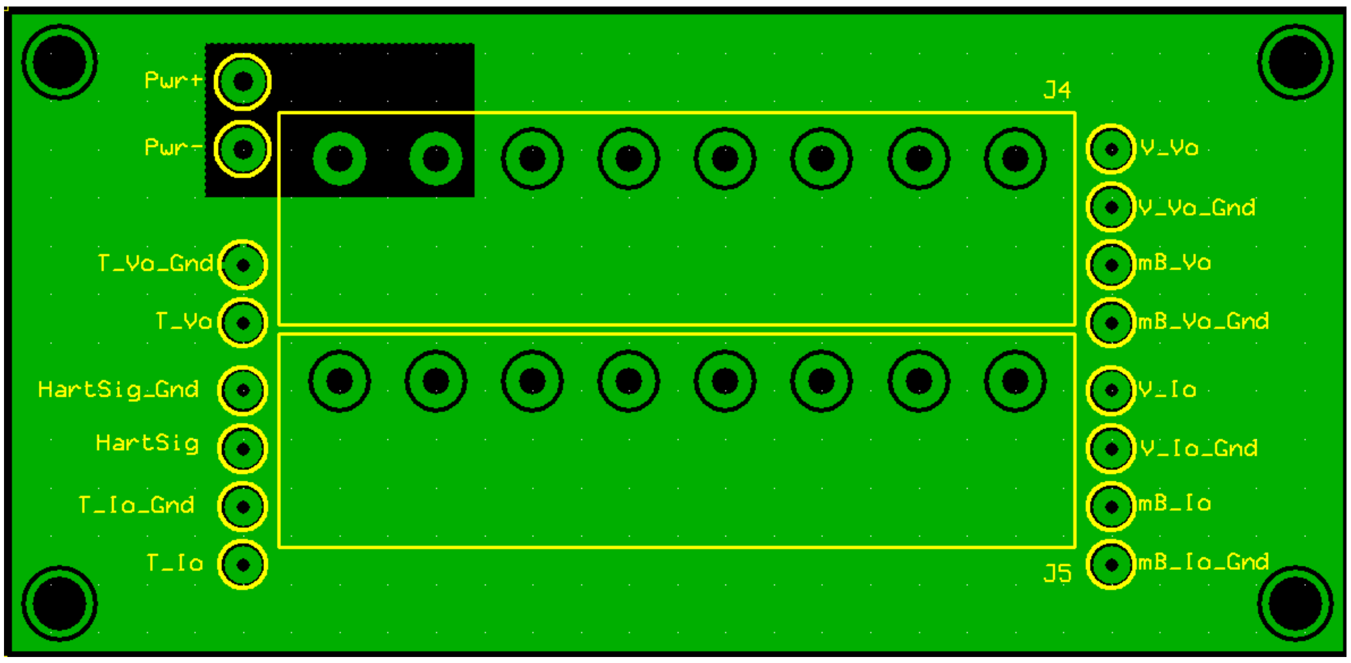
Moisture Monitor Connector Schematic



Moisture Monitor Connector PCB - Top Plane



Moisture Monitor Connector PCB - Bottom Plane



Moisture Sensor Monitor Source Code

```
////////////////////////////////////
//  Generated Initialization File  //
////////////////////////////////////
//#include "compiler_defs.h"
#include "compiler_defs.h"
#include "C8051F580_defs.h"
#include <intrins.h>
#include <stdio.h>
#include <math.h>

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------
// Microcontroller paramters
#define SYSCLK                          48000000                            // SYSCLK frequency in Hz

// Communication packet parameters
#define Serial_Header                   '&'                                 // Serial packet headers
#define CR_Terminator                   0x0D                                // Serial terminator #1 = CR
#define LF_Terminator                   0x0A                                // Serial terminator #2 = LF
#define ValidPacketSize                 74                                  // Size of valid packet from WVSS_II
#define SerialBufSize                   200                                 // Serial input buffer size
#define PacketBufSize0                  80
#define PacketBufSize1                  80

// Individual DAC8760 read/write variables
#define ppmv_SPI_Wr                     DAC8760_SPI_Frame_Wr[2]             // Water vapor SPI Write Frame
#define mBar_SPI_Wr                     DAC8760_SPI_Frame_Wr[1]             // Pressure SPI Write Frame
#define degC_SPI_Wr                     DAC8760_SPI_Frame_Wr[0]             // Temperature SPI Write Frame
#define ppmv_SPI_Rd                     DAC8760_SPI_Frame_Rd[2]             // Water vapor SPI Read Frame
#define mBar_SPI_Rd                     DAC8760_SPI_Frame_Rd[1]             // Pressure SPI Read Frame
#define degC_SPI_Rd                     DAC8760_SPI_Frame_Rd[0]             // Temperature SPI Read Frame

// DAC8760 register write addresses
#define NOP_Adr                         0x0
#define Wr_DAC_Data_Adr                 0x1
#define Rd_Register_Adr                 0x2
#define Wr_Control_Adr                  0x55
#define Wr_Reset_Adr                    0x56
#define Wr_Config_Adr                   0x57
#define Wr_DAC_GainCal_Adr              0x58
#define Wr_DAC_ZeroCal_Adr              0x59
#define WatchDog_Tmr_Rst_Adr            0x95

// DAC8760 register read addresses
#define Rd_Status_Reg                   0x0
#define Rd_DAC_Data_Reg                 0x1
#define Rd_Control_Reg                  0x2
#define Rd_Config_Reg                   0xB
#define Rd_DAC_GainCal_Reg              0x13
#define Rd_DAC_ZeroCal_Reg              0x17

// DAC8760 control register over voltage enable status
#define OVR_Enabled                     0x4000
#define ppmv_DAC_OVR_Enabled            ((ppmv_DAC.Control_Reg_Val & OVR_Enabled)!=0)
#define mBar_DAC_OVR_Enabled            ((mBar_DAC.Control_Reg_Val & OVR_Enabled)!=0)
#define degC_DAC_OVR_Enabled            ((degC_DAC.Control_Reg_Val & OVR_Enabled)!=0)

// DAC8760 configuration register Hart enable status
#define Hart_Enabled                    0x10
#define ppmv_DAC_Hart_Enabled           ((ppmv_DAC.Config_Reg_Val & Hart_Enabled)!=0)
#define mBar_DAC_Hart_Enabled           ((mBar_DAC.Config_Reg_Val & Hart_Enabled)!=0)
#define degC_DAC_Hart_Enabled           ((degC_DAC.Config_Reg_Val & Hart_Enabled)!=0)

// DAC8760 configuration register Iout enable status (**Range set for 4mA to 20mA => 0x0200 or disabled => 0x0000)
#define Iout_Enabled                    0x0200
#define ppmv_DAC_Iout_Enabled           ((ppmv_DAC.Config_Reg_Val & Iout_Enabled)!=0)
#define mBar_DAC_Iout_Enabled           ((mBar_DAC.Config_Reg_Val & Iout_Enabled)!=0)
#define degC_DAC_Iout_Enabled           ((degC_DAC.Config_Reg_Val & Iout_Enabled)!=0)

// DAC8760 configuration register alteration indicator (**Detect changes in Iout enable and Hart enable)
#define ppmv_ConfigRegAltered           (((ppmv_DAC.Config_Reg_Val & 0x0310)^ppmv_ConfigRegValue)!=0)
#define mBar_ConfigRegAltered           (((mBar_DAC.Config_Reg_Val & 0x0310)^mBar_ConfigRegValue)!=0)
#define degC_ConfigRegAltered           (((degC_DAC.Config_Reg_Val & 0x0310)^degC_ConfigRegValue)!=0)

// DAC8760 status register bits
#define CRC_FLT                         0x10
#define WD_FLT                          0x8
#define I_FLT                           0x4
#define SR_ON                           0x2
#define T_FLT                           0x1

// DAC8760 Iout and Temperature status shortcuts
#define ppmv_DAC_Iout_Error             ((ppmv_DAC.Status_Reg_Val & I_FLT)!=0)
#define mBar_DAC_Iout_Error             ((mBar_DAC.Status_Reg_Val & I_FLT)!=0)
#define degC_DAC_Iout_Error             ((degC_DAC.Status_Reg_Val & I_FLT)!=0)
#define ppmv_DAC_Temp_Error             ((ppmv_DAC.Status_Reg_Val & T_FLT)!=0)
#define mBar_DAC_Temp_Error             ((mBar_DAC.Status_Reg_Val & T_FLT)!=0)
#define degC_DAC_Temp_Error             ((degC_DAC.Status_Reg_Val & T_FLT)!=0)

// WVSS-II min/max output values
#define min_ppmv                        50.0
#define max_ppmv                        40000.0
#define min_mBar                        150.0
#define max_mBar                        1016.0
#define min_degC                        (-15.0)
#define max_degC                        40.0

// SDG_43A commands
#define ReqHartOn                       ((SDG_43A_Cmd & 0x01)!=0)

// System Status Flags
#define SerialTimeOutFlag               0x80
```

```
#define HartSignalActiveFlag                    0x40
#define ppmvDAC_IoutErrorFlag                   0x20
#define mBarDAC_IoutErrorFlag                   0x10
#define degCDAC_IoutErrorFlag                   0x8
#define ppmvDAC_TempErrorFlag                   0x4
#define mBarDAC_TempErrorFlag                   0x2
#define degCDAC_TempErrorFlag                   0x1

// Data packet status
#define InvalidData                             0
#define ValidData                               1
#define MaxInvalidRxPackets                     200

// Timer wait values
#define _500_msec_wait                          5000                    // number of timer0 cycles for 500msec wait time (1 cycle = 100usec)
#define _4000_msec_wait                         40000                   // number of timer0 cycles for 4000msec wait time (1 cycle = 100usec)

//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------
void Timer_Init(void);
void UART_Init(void);
void SPI_Init(void);
void Port_IO_Init(void);
void Oscillator_Init(void);
void Interrupts_Init(void);

unsigned char Serial_In_WVSS_II(void);
unsigned char Serial_In_SDG_43A(void);
//void Serial_Out_WVSS_II(unsigned char Serial_Data);
void Serial_Out_SDG_43A(unsigned char Serial_Data);
unsigned char Get_WVSS_II_Data(void);
unsigned char Get_SDG_43A_Data(void);
void DAC8760_Init(void);
void DAC8760_Data_Transfer(unsigned char Address, unsigned short Vapor_DAC_RegData, unsigned short Pressure_DAC_RegData,unsigned short Temperature_DAC_RegData);
void DAC8760_Register_Read(void);

//-----------------------------------------------------------------------------
// Global Variables
//-----------------------------------------------------------------------------
union SPI_Fmt
  {
  unsigned char byte[3];
  struct{unsigned char Address; unsigned short RegData;}SPI_Frame;
  };

struct Rd_Reg_Fmt
  {
  unsigned short Status_Reg_Val;
  unsigned short DAC_Data_Reg_Val;
  unsigned short Control_Reg_Val;
  unsigned short Config_Reg_Val;
  unsigned short DAC_GainCal_Reg_Val;
  unsigned short DAC_ZeroCal_Reg_Val;
  };

// RS232 variables
data unsigned char RcvChar;
data bit PendingXmit0=0,Serial_BufEmpty0=1,Serial_BufFull0=0;
data bit PendingXmit1=0,Serial_BufEmpty1=1,Serial_BufFull1=0;
data unsigned short Serial_BufTail0=0,Serial_BufHead0=0,Serial_BufCount0=0;
data unsigned short Serial_BufTail1=0,Serial_BufHead1=0,Serial_BufCount1=0;
data unsigned short WVSS_II_TimeOut_Period=_4000_msec_wait,SDG_43A_Xmit_Period=_500_msec_wait;
xdata unsigned char  PacketIndex0=0,HeaderNum0=0;
xdata unsigned char  PacketIndex1=0,HeaderNum1=0;
xdata unsigned char PacketBuf0[PacketBufSize0],ValidPacket0[PacketBufSize0];
xdata unsigned char PacketBuf1[PacketBufSize1],ValidPacket1[PacketBufSize1];
xdata unsigned char ValidIndex,ValidPacketSize1;
xdata unsigned char Serial_RcvBuf0[SerialBufSize];
xdata unsigned char Serial_RcvBuf1[SerialBufSize];

// DAC8760 variables
union SPI_Fmt DAC8760_SPI_Frame_Wr[3], DAC8760_SPI_Frame_Rd[3];
unsigned char DAC_Idx,Byte_Idx;
xdata unsigned char DAC_Status=0;
xdata struct Rd_Reg_Fmt ppmv_DAC,mBar_DAC,degC_DAC;

// WVSS-II variables
xdata float WaterVapor,Pressure,Temperature;
xdata unsigned short ppmv_DAC_DataValue,mBar_DAC_DataValue,degC_DAC_DataValue;
xdata unsigned short ppmv_ConfigRegValue,mBar_ConfigRegValue,degC_ConfigRegValue;

// SDG_43A variables
unsigned char SDG_43A_Cmd=0;

// external variables
extern xdata unsigned char String[],StringPtr;

// Port Pin definitions
sbit Latch = P0^3;
sbit Clear = P1^0;
sbit not_Alarm = P1^1;
sbit ppmv_OOR = P1^2;
sbit mBar_OOR = P1^3;
sbit degC_OOR = P1^4;

// Status bits
data bit SPI_Done=0;

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void)
{
// initialize microcontroller
Port_IO_Init();
Oscillator_Init();
Timer_Init();
```

```
UART_Init();
SPI_Init();
Interrupts_Init();
SFRPAGE = ACTIVE_PAGE;                                                                          // Enable watchdog
PCA0MD |= 0x40;

// initialize DAC8760's
DAC8760_Init();                                                                                 // For all daisy-chained DACs, set output enable, daisy-chain enable and voltage range for 0 to 5 volts
DAC8760_Data_Transfer(Wr_Config_Adr,0x0300,0x0300,0x0300);                                      // For all daisy-chained DACs, set Iout range to 4mA to 20mA and set dual output enable (Vout and Iout)
DAC8760_Register_Read();                                                                        // Read all DAC registers in daisy-chain

// main processing loop
while (1)                                                                                       // Spin forever
 {

  // handle WVSS_II serial data
  if (Get_WVSS_II_Data()==ValidData)                                                            // Read commands from WVSS_II serial port, load valid command values
   {
   if ((sscanf(ValidPacket0,"%f %f %f",&WaterVapor,&Pressure,&Temperature))==3)                 // Scan command string for all WVSS_II data outputs and send to SDG_43A if valid
    {
    WVSS_II_TimeOut_Period = _4000_msec_wait;                                                   // Reset serial timeout clock
    DAC_Status &= ~SerialTimeOutFlag;                                                           // Clear DAC status serial time-out bit
    if ((WaterVapor>=min_ppmv)&&(WaterVapor<=max_ppmv))                                         // Check if water vapor concentration is in range
     {
     ppmv_OOR = 0;                                                                              // In range: Clear over-range bit (Use IDAC output)
     ppmv_ConfigRegValue = ppmv_DAC.Config_Reg_Val | Iout_Enabled;                              // In range: Set Iout range for 4ma to 20ma
     ppmv_DAC_DataValue = (unsigned short)(65535*((WaterVapor-min_ppmv)/(max_ppmv-min_ppmv)));  // In range: Scale water vapor concentration for VDAC/IDAC
     }
    else
     {
     ppmv_OOR = 1;                                                                              // Out of range: Set over-range bit (**Switches in 2mA instead of normal IDAC output)
     ppmv_ConfigRegValue = ppmv_DAC.Config_Reg_Val & ~Iout_Enabled;                             // Out of range: Set Iout range to disabled
     if (WaterVapor<min_ppmv)
      ppmv_DAC_DataValue = 0;                                                                   // Out of range<min: Set VDAC/IDAC output = 0 (**VDAC=0 volts, IDAC=4mA, but overide switches in 2mA)
     else
      ppmv_DAC_DataValue = 0xFFFF;                                                              // Out of range>max: Set VDAC/IDAC output = full scale (**VDAC=5 volts, IDAC=20mA, but overide switches in 2mA)
     }
   if ((Pressure>=min_mBar)&&(Pressure<=max_mBar))                                              // Check if Air pressure is in range
     {
     mBar_OOR = 0;                                                                              // In range: Clear over-range bit (Use IDAC output)
     mBar_ConfigRegValue = mBar_DAC.Config_Reg_Val | Iout_Enabled;                              // In range: Set Iout range for 4ma to 20ma
     mBar_DAC_DataValue = (unsigned short)(65535*((Pressure-min_mBar)/(max_mBar-min_mBar)));    // In range: Scale air pressure for VDAC/IDAC
     }
   else
    {
    mBar_OOR = 1;                                                                               // Out of range: Set over-range bit (**Switches in 2mA instead of normal IDAC output)
    mBar_ConfigRegValue = mBar_DAC.Config_Reg_Val & ~Iout_Enabled;                              // Out of range: Set Iout range to disabled
    if (Pressure<min_mBar)
      mBar_DAC_DataValue = 0;                                                                   // Out of range: Set VDAC/IDAC output = 0 (**VDAC=0 volts, IDAC=4mA, but overide switches in 2mA)
    else
      mBar_DAC_DataValue = 0xFFFF;                                                              // Out of range>max: Set VDAC/IDAC output = full scale (**VDAC=5 volts, IDAC=20mA, but overide switches in 2mA)
    }
   if ((Temperature>=min_degC)&&(Temperature<=max_degC))                                        // Check if Air temperature is in range
    {
    degC_OOR = 0;                                                                               // In range: Clear over-range bit (Use IDAC output)
    degC_ConfigRegValue = degC_DAC.Config_Reg_Val | Iout_Enabled;                               // In range: Set Iout range for 4ma to 20ma
    degC_DAC_DataValue = (unsigned short)(65535*((Temperature-min_degC)/(max_degC-min_degC)));  // In range: Scale air temperature for VDAC/IDAC
    }
   else
    {
    degC_OOR = 1;                                                                               // Out of range: Set over-range bit (**Switches in 2mA instead of normal IDAC output)
    degC_ConfigRegValue = degC_DAC.Config_Reg_Val & ~Iout_Enabled;                              // Out of range: Set Iout range to disabled
    if (Temperature<min_degC)
      degC_DAC_DataValue = 0;                                                                   // Out of range: Set VDAC/IDAC output = 0 (**VDAC=0 volts, IDAC=4mA, but overide switches in 2mA)
    else
      degC_DAC_DataValue = 0xFFFF;                                                              // Out of range>max: Set VDAC/IDAC output = full scale (**VDAC=5 volts, IDAC=20mA, but overide switches in 2mA)
    }
   if (ppmv_DAC_OVR_Enabled || mBar_DAC_OVR_Enabled  || degC_DAC_OVR_Enabled)                   // if any over-voltage bits are set, send new control register value to clear bits
    DAC8760_Data_Transfer(Wr_Control_Adr,0x1008,0x1008,0x1008);
   if (ppmv_ConfigRegAltered || mBar_ConfigRegAltered  || degC_ConfigRegAltered)                // if any Iout range bits or Hart bits have been changed, send new configuration register values
    DAC8760_Data_Transfer(Wr_Config_Adr,ppmv_ConfigRegValue,mBar_ConfigRegValue,degC_ConfigRegValue);
   DAC8760_Data_Transfer(Wr_DAC_Data_Adr,ppmv_DAC_DataValue,mBar_DAC_DataValue,degC_DAC_DataValue);  // Send data to daisy-chained DAC8760's
   }
 }

  // handle WVSS_II serial timeout protocol
  if (WVSS_II_TimeOut_Period==0)
                                                                                                // Set over-range conditions if no valid serial data is acquired within timeout period
   {
   ppmv_OOR = 1;                                                                                // Set over-range bits for each DAC (**Switches in 2mA instead of normal IDAC output)
   mBar_OOR = 1;
   degC_OOR = 1;
   WaterVapor = 0;                                                                              // Set water vapor,pressure and temperature values = 0
   Pressure = 0;
   Temperature = 0;
   ppmv_ConfigRegValue = ppmv_DAC.Config_Reg_Val & ~Iout_Enabled;                               // Set Iout range to disabled
   mBar_ConfigRegValue = mBar_DAC.Config_Reg_Val & ~Iout_Enabled;
   degC_ConfigRegValue = degC_DAC.Config_Reg_Val & ~Iout_Enabled;
   DAC_Status |= SerialTimeOutFlag;                                                             // Set DAC status serial time-out bit
   if (!ppmv_DAC_OVR_Enabled || !mBar_DAC_OVR_Enabled  || !degC_DAC_OVR_Enabled)                // if any over-voltage bits are cleared, send new control register values to set bits
    DAC8760_Data_Transfer(Wr_Control_Adr,0x5008,0x5008,0x5008);
   if (ppmv_DAC_Iout_Enabled || mBar_DAC_Iout_Enabled  || degC_DAC_Iout_Enabled)                // if any Iout range bits are set for 4mA to 20mA, send new configuration register values to disable Iout
    DAC8760_Data_Transfer(Wr_Config_Adr,ppmv_ConfigRegValue,mBar_ConfigRegValue,degC_ConfigRegValue);
   DAC8760_Data_Transfer(Wr_DAC_Data_Adr,0xFFFF,0xFFFF,0xFFFF);                                 // Set VDAC/IDAC data values to full-scale  (**VDAC=>5volts, IDAC=>20mA)
   }

  // read DAC8760 registers
  DAC8760_Register_Read();                                                                      // Read all DAC registers in daisy-chain
  if (ppmv_DAC_Hart_Enabled && mBar_DAC_Hart_Enabled && degC_DAC_Hart_Enabled)                  // If all DACs indicate Hart Signal is enabled, set corresponding DAC status flag
    DAC_Status |= HartSignalActiveFlag;
  else
    DAC_Status &= ~HartSignalActiveFlag;

  // handle DAC8760 alarms
  if (not_Alarm==0)                                                                             // Check for any alarm conditions
    {
    if (ppmv_DAC_Iout_Error)                                                                    // Set/Reset Iout Error for water vapor DAC
      DAC_Status |= ppmvDAC_IoutErrorFlag;
    else
      DAC_Status &= ~ppmvDAC_IoutErrorFlag;
    if (mBar_DAC_Iout_Error)                                                            // Set/Reset Iout Error for air pressure DAC
      DAC_Status |= mBarDAC_IoutErrorFlag;
    else
      DAC_Status &= ~mBarDAC_IoutErrorFlag;
    if (degC_DAC_Iout_Error)                                                            // Set/Reset Iout Error for air temperature DAC
      DAC_Status |= degCDAC_IoutErrorFlag;
    else
      DAC_Status &= ~degCDAC_IoutErrorFlag;
    if (ppmv_DAC_Temp_Error)                                                            // Set/Reset Temperature Error for water vapor DAC
     DAC_Status |= ppmvDAC_TempErrorFlag;
    else
      DAC_Status &= ~ppmvDAC_TempErrorFlag;
    if (mBar_DAC_Temp_Error)                                                            // Set/Reset Temperature Error for air pressure DAC
      DAC_Status |= mBarDAC_TempErrorFlag;
    else
      DAC_Status &= ~mBarDAC_TempErrorFlag;
    if (degC_DAC_Temp_Error)                                                            // Set/Reset Temperature Error for air temperature DAC
      DAC_Status |= degCDAC_TempErrorFlag;
    else
      DAC_Status &= ~degCDAC_TempErrorFlag;
                    }
  else DAC_Status &= ~0x3F;                                                             // Clear error flags if not_Alarm bit is set

  // handle SDG_43A serial received data
  if (Get_SDG_43A_Data()==ValidData)                                                    // Read commands from SDG_43A serial port, load valid command values
   {
   if (ReqHartOn)                                                                       // If requested Hart signal input and any DACs indicate Hart Signal is disabled, send enable all Hart inputs
     {
     ppmv_ConfigRegValue = ppmv_DAC.Config_Reg_Val | Hart_Enabled;
     mBar_ConfigRegValue = mBar_DAC.Config_Reg_Val | Hart_Enabled;
     degC_ConfigRegValue = degC_DAC.Config_Reg_Val | Hart_Enabled;
     if (!ppmv_DAC_Hart_Enabled || !mBar_DAC_Hart_Enabled || !degC_DAC_Hart_Enabled)
       DAC8760_Data_Transfer(Wr_Config_Adr,ppmv_ConfigRegValue,mBar_ConfigRegValue,degC_ConfigRegValue);
     }
   else                                                                                 // If requested Normal signal input and any DACs indicate Hart Signal is enabled, send disable all Hart inpust
     {
     ppmv_ConfigRegValue = ppmv_DAC.Config_Reg_Val & ~Hart_Enabled;
     mBar_ConfigRegValue = mBar_DAC.Config_Reg_Val & ~Hart_Enabled;
     degC_ConfigRegValue = degC_DAC.Config_Reg_Val & ~Hart_Enabled;
     if (ppmv_DAC_Hart_Enabled || mBar_DAC_Hart_Enabled || degC_DAC_Hart_Enabled)
       DAC8760_Data_Transfer(Wr_Config_Adr,ppmv_ConfigRegValue,mBar_ConfigRegValue,degC_ConfigRegValue);
     }
   }

  // perform periodic SDG_43A serial transmissions
  if (SDG_43A_Xmit_Period==0)                                                           // Transmit data packet to SDG_43A at desired period
    {
    SDG_43A_Xmit_Period=_500_msec_wait;
    printf("\nrx,%7.2f,%4.1f,%4.2f",WaterVapor,Pressure,Temperature);                   // Format and send packet to SDG_43A => "rx,wwwww.ww,ppp.p,tt.tt,xxxxx,xxxxx,dddCRLF"
    printf(",%-1bu",DAC_Status);
    for (ValidIndex=0; ValidIndex<StringPtr; ValidIndex++) Serial_Out_SDG_43A(String[ValidIndex]);
    Serial_Out_SDG_43A(CR_Terminator);
    Serial_Out_SDG_43A(LF_Terminator);
    }

  SFRPAGE = ACTIVE_PAGE;                                                                // Reset watchdog timer
  PCA0CPH5 = 0x0;
  }

}

//-----------------------------------------------------------------------------
// Supporting Functions
//-----------------------------------------------------------------------------
//
// Initialize Timer0 and Timer1
void Timer_Init()
{
TCON = 0x50;                    // Enable timer0 and timer1
TMOD = 0x22;                    // Timer0 and Timer1 = Mode 2: 8-bit Counter/Timer with Auto-Reload
CKCON = 0x02;                   // Timer0 and Timer1 clocks = Sysclk/48
TL0 = 0x9C;
TH0 = 0x9C;                     // Set Timer0 auto-reload value for 100usec interrupts
TH1 = 0xE6;                     // Set Timer1 auto-reload value for UART1 baud rate of 19230 bps
}

//Initialize UART0 and UART1
void UART_Init()
{
SCON0 = 0x10;                   //UART1 reception enabled
SFRPAGE = CONFIG_PAGE;          //SMOD0 = default = 0x0C = No Parity, 8-bit data, 1 stop bit
SBRLL0 = 0x3C;                  //UART0 Baud rate set for 9600 bps
SBRLH0 = 0xF6;
SBCON0 = 0x43;                  //UART0 Baud Rate Generator enabled, Baud Rate Prescaler = 1
SFRPAGE = ACTIVE2_PAGE;
SCON1 = 0x50;                   //UART1 reception enabled
SFRPAGE = ACTIVE_PAGE;
}

//Initialize SPI0
void SPI_Init()
{
SPI0CFG = 0x40;                 // SPI0 Mode: Enable master mode. Operate as a master.
SPI0CN = 0x01;                  // Enable SPI0
SPI0CKR = 0x02;                 // Set SPI0 clock = 8MHz
}

//Initialize Port IO
void Port_IO_Init()
{
// P0.0  -  SCK  (SPI0), Push-Pull,  Digital
// P0.1  -  MISO (SPI0), Open-Drain, Digital
// P0.2  -  MOSI (SPI0), Push-Pull,  Digital
// P0.3  -  Latch,       Push-Pull,  Digital
// P0.4  -  UART0_TX,    Push-Pull,  Digital
// P0.5  -  UART0_RX,    Open-Drain, Digital
// P0.6  -  UART1_TX,    Push-Pull,  Digital
// P0.7  -  UART1_RX,    Open-Drain, Digital
//
// P1.0  -  Clear,       Push-Pull,  Digital
// P1.1  -  not_Alarm,   Open-Drain, Digital
// P1.2  -  ppmv_OOR,    Push-Pull,  Digital
// P1.3  -  mBar_OOR,    Push-Pull,  Digital
// P1.4  -  degC_OOR,    Push-Pull,  Digital
// P1.5  -  Unassigned,  Open-Drain, Digital
// P1.6  -  Unassigned,  Open-Drain, Digital
// P1.7  -  Unassigned,  Open-Drain, Digital
//
// P2.0  -  Unassigned,  Open-Drain, Digital
// P2.1  -  Unassigned,  Open-Drain, Digital
// P2.2  -  Unassigned,  Open-Drain, Digital
// P2.3  -  Unassigned,  Open-Drain, Digital
// P2.4  -  Unassigned,  Open-Drain, Digital
// P2.5  -  Unassigned,  Open-Drain, Digital
// P2.6  -  Unassigned,  Open-Drain, Digital
// P2.7  -  Unassigned,  Open-Drain, Digital
//
// P3.0  -  Unassigned,  Open-Drain, Digital
//
SFRPAGE = ACTIVE_PAGE;
PCA0MD &= ~0x40;                        // disable watchdog
PCA0MD = 0x04;                          // Set PCA0 clock source to timer0 overflow 100usec interval
PCA0CPL5 =0x26;                         // Set watchdog updata value to 1000msec
SFRPAGE = CONFIG_PAGE;
P0MDOUT = 0x5D;                         // P0.0, P0.2 - P0.4, P0.6 => Outputs are push-pull, all others are open drain
P1MDOUT = 0x1D;                         // P1.0, P1.2 - P1.4 => Outputs are push-pull, all others are open drain
P0SKIP = 0x08;                          // Skip P0.3
P1SKIP = 0x1F;                          // Skip P1.0 - P1.4
XBR0 = 0x05;                            // SPI0 I/O routed to port pins, UART0 TX0, RX0 routed to Port pins P0.4 and P0.5
XBR2 = 0x42;                            // Crossbar enabled, UART1 TX0, RX0 routed to Port pins
SFRPAGE = ACTIVE_PAGE;
}

//Initialize clock oscillator
void Oscillator_Init()
{
int i = 0;
SFRPAGE = CONFIG_PAGE;
OSCICN = 0xC7;                          // Oscillator enabled in normal mode and disabled in suspend mode, SYSCLK derived from internal oscillator/1
CLKMUL = 0x92;                          // Clock multiplier enabled, Clock multiplier Output scaled by a factor of 2/4 (1/2), Clock multiplier input = internal oscillator x 4
for (i = 0; i < 20; i++);               // Wait 5us for clock multiplier initialization
CLKMUL |= 0xC0;                         // Initialize clock multiplier
while ((CLKMUL & 0x20) == 0);           // Wait for clock multiplier ready bit = 1 (PLL is locked)
CLKSEL = 0x02;                          // SYSCLK derived from the Clock Multiplier
SFRPAGE = ACTIVE_PAGE;
}

//Initialize interrupts
void Interrupts_Init()
{
EIE2 = 0x08;                            // Enable UART1 interrupt
IE = 0xD2;                              // Globally enable all interrupts, enable interrupt requests generated by SPI0, enable UART0 interrupt, enable Timer0 interrupt
}

// ****************************************************************************************************
//  Function Name: Serial_In_WVSS_II
//        Purpose: Return current character from WVSS_II serial buffer
//          Input: Serial_RcvBuf0 (**Global) - UART0 serial buffer
//                 Serial_BufHead0 (**Global) - Pointer to current character in UART0 serial buffer
//                 Serial_BufTail0 (**Global) - Pointer to last character in UART0 serial buffer
//                 Serial_BufCount0 (**Global) - Number of characters in UART0 serial buffer
//                 SerialBufSize (**Global) - Serial buffer size
//                 Serial_BufEmpty0 (**Global) -  Flag indicator for empty UART0 serial buffer
//         Output: return = character
// ****************************************************************************************************
unsigned char Serial_In_WVSS_II(void)
{
unsigned char RcvChar;
ES0 = 0;                                                                 // disable UART0 interrupts -(all pages)
if (!Serial_BufEmpty0)
  {
  RcvChar = Serial_RcvBuf0[Serial_BufHead0++];                           // store current character in temporary variable and increment buffer head to next position
  if (Serial_BufCount0!= 0) --Serial_BufCount0;                          // update buffer count after removing current character from buffer
  if (Serial_BufHead0==SerialBufSize) Serial_BufHead0 = 0;               // check for end of buffer
  if (Serial_BufHead0==Serial_BufTail0) Serial_BufEmpty0 = 1;            // check for empty buffer
  Serial_BufFull0 = 0;
  }
else RcvChar = 0;
ES0 = 1;                                                                 // enable UART0 interrupts -(all pages)
return(RcvChar);                                                         // return character
}

// ****************************************************************************************************
//  Function Name: Serial_In_SDG_43A
//        Purpose: Return current character from SDG_43A serial buffer
//          Input: Serial_RcvBuf1 (**Global) - UART1 serial buffer
//                 Serial_BufHead1 (**Global) - Pointer to current character in UART1 serial buffer
//                 Serial_BufTai1 (**Global) - Pointer to last character in UART1 serial buffer
//                 Serial_BufCount1 (**Global) - Number of characters in UART1 serial buffer
//                 SerialBufSize (**Global) - Serial buffer size
//                 Serial_BufEmpty1 (**Global) -  Flag indicator for empty UART1 serial buffer
//         Output: return = character
// ****************************************************************************************************
unsigned char Serial_In_SDG_43A(void)
{
unsigned char RcvChar;
EIE2 &= ~0x08;                                                           // disable UART1 interrupts -(all pages)
if (!Serial_BufEmpty1)
  {
  RcvChar = Serial_RcvBuf1[Serial_BufHead1++];                           // store character in temporary variable
  if (Serial_BufCount1!= 0) --Serial_BufCount1;
  if (Serial_BufHead1==SerialBufSize) Serial_BufHead1 = 0;               // check for end of buffer
  if (Serial_BufHead1==Serial_BufTail1) Serial_BufEmpty1 = 1;            // check for empty buffer
  Serial_BufFull0 = 0;
  }
else RcvChar = 0;
EIE2 |= 0x08;                                                            // enable UART1 interrupts -(all pages)
```

```
return(RcvChar);                                                                // return character
}

// **************************************************************************************************
//  Function Name: Serial_Out_WVSS_II
//        Purpose: Transmit character on serial0 line
//          Input: Serial_Data - data to transmit
//                 PendingXmit0 (**Global) - Flag indicator for incomplete UART0 serial transmission
//         Output: none
// **************************************************************************************************
/*void Serial_Out_WVSS_II(unsigned char Serial_Data)
{
while (PendingXmit0);                                                           // wait for no pending-transmissions
PendingXmit0 = 1;                                                               // set pending-transmission flag
SFRPAGE = ACTIVE_PAGE;
SBUF0 = Serial_Data;                                                            // load character into serial hardware buffer to initiate serial transmission -(all pages)
}*/

// **************************************************************************************************
//  Function Name: Serial_Out_SDG_43A
//        Purpose: Transmit character on serial1 line
//          Input: Serial_Data - data to transmit
//                 PendingXmit1 (**Global) - Flag indicator for incomplete UART1 serial transmission
//         Output: none
// **************************************************************************************************
void Serial_Out_SDG_43A(unsigned char Serial_Data)
{
while (PendingXmit1);                                                           // wait for no pending-transmissions
PendingXmit1 = 1;                                                               // set pending-transmission flag
SFRPAGE = ACTIVE2_PAGE;
SBUF1 = Serial_Data;                                                            // load character into serial hardware buffer to initiate serial transmission -(all pages)
SFRPAGE = ACTIVE_PAGE;
}

// ******************************************************************
//  Function Name: Get_WVSS_II_Data
//        Purpose: Get WVSS_II data from UART0
//          Input: Serial_In_WVSS_II
//         Output: Packet and status
//
// ******************************************************************
unsigned char Get_WVSS_II_Data(void)
{
unsigned char SerialDataStatus=InvalidData;
unsigned char CurChar,index;
while (!Serial_BufEmpty0 && (SerialDataStatus==InvalidData))                                                // Loop until serial0 buffer is empty or valid packet obtained
 {
 CurChar = Serial_In_WVSS_II();                                                                             // Get current character from serial0 buffer
 switch (HeaderNum0)
  {
  case 0: if (CurChar==CR_Terminator)                                                                       // Check for carriage return
           HeaderNum0++;
                                                                                                            // yes - begin search for line-feed
          else
           PacketBuf0[PacketIndex0++] = CurChar;                                                            // no - load current character into packet buffer
          break;
  case 1: if ((CurChar==LF_Terminator)&&(PacketIndex0==ValidPacketSize))                                    // Check for line-feed and valid number of characters in packet
             {
             for (index=0; index<PacketIndex0; index++) ValidPacket0[index] = PacketBuf0[index];            // If conditions are met, load packet buffer into valid packet string
             ValidPacket0[PacketIndex0] = '\0';                                                             // Concatenate EOF to valid packet string
             SerialDataStatus = ValidData;
             }
           PacketIndex0 = 0;
           HeaderNum0 = 0;                                                                                  // Reset Get_WVSS_II_Data state machine index
           break;
 default: HeaderNum0 = 0;                                                                                   // For all fault conditions, reset Get_WVSS_II_Data state machine index
           break;
  }
 }
return(SerialDataStatus);                                                                                   // Return packet status - valid or invalid
}

// ******************************************************************
//  Function Name: Get_SDG_43A_Data
//        Purpose: Get SDG_43A command packet from UART
//          Input: Serial_In_SDG_43A
//         Output: Packet and status
//
// ******************************************************************
unsigned char Get_SDG_43A_Data(void)
{
unsigned char SerialDataStatus=InvalidData;
unsigned char CurChar,index;
while (!Serial_BufEmpty1 && (SerialDataStatus==InvalidData))                                                // Loop until serial1 buffer is empty or valid packet obtained
 {
 CurChar = Serial_In_SDG_43A();                                                                             // Get current character from serial1 buffer
 switch (HeaderNum1)
  {
  case 0: if (CurChar==Serial_Header)                                                                       // Check for packet header
             {
             HeaderNum1++;                                                                                  // yes - begin scan for valid packet, reset packet index
             PacketIndex1=0;
             }
           break;
  case 1: if (CurChar==CR_Terminator)                                                                       // Check for carriage return
             HeaderNum1++;                                                                                  // yes - begin search for line-feed
           else
             PacketBuf1[PacketIndex1++] = CurChar;                                                          // no - load current character into packet buffer
           break;
  case 2: if (CurChar==LF_Terminator)                                                                       // Check for line-feed
             {
             for (index=0; index<PacketIndex1; index++) ValidPacket1[index] = PacketBuf1[index];            // If line-feed found, load packet buffer into valid packet string
             ValidPacket1[PacketIndex1] = '\0';                                                             // Concatenate EOF to valid packet string
             ValidPacketSize1 = PacketIndex1;                                                               // Load packet size into valid packet size variable
             if ((sscanf(ValidPacket1,"%bu",&SDG_43A_Cmd))==1)                                              // Scan SDG_43A string for command byte and set valid data flag if present
               SerialDataStatus = ValidData;                                                                // Indicate capture of valid packet
             }
           HeaderNum1 = 0;                                                                                  // Reset Get_SDG_43A_Data state machine index
           break;
 default: HeaderNum1 = 0;                                                                                   // For all fault conditions, reset Get_SDG_43A_Data state machine index
           break;
```

```
    }
  }
return(SerialDataStatus);                                                                                   // Return packet status - valid or invalid
}

// ******************************************************************
//  Function Name: DAC8760_Init
//        Purpose: Initialize the three daisy-chained DAC8760's
//          Input: none
//         Output: none
//
// ******************************************************************
void DAC8760_Init(void)
{
unsigned char index;
Latch = 0;                                                              // Set Latch line to its nominal low state
Clear = 1;                                                              // Clear DAC by pulling clear line high
for (index=0; index<4; index++) _nop_();                                // Wait 1us
Clear = 0;
                        // Return clear line to its nominal low state
ppmv_OOR = 1;                                                           // Set all over-range lines high
mBar_OOR = 1;
degC_OOR = 1;
ppmv_SPI_Wr.SPI_Frame.Address = Wr_Control_Adr;                         // Initialize SPI byte stream to set control register of all three daisy-chained DACs
ppmv_SPI_Wr.SPI_Frame.RegData = 0x1008;                                 // for output enable, daisy-chain enable and voltage range for 0 to 5 volts.
mBar_SPI_Wr.SPI_Frame.Address = Wr_Control_Adr;
mBar_SPI_Wr.SPI_Frame.RegData = 0x1008;
degC_SPI_Wr.SPI_Frame.Address = Wr_Control_Adr;
degC_SPI_Wr.SPI_Frame.RegData = 0x1008;
for (DAC_Idx=0; DAC_Idx<3; DAC_Idx++)                                   // Output SPI data to daisy-chained DACs
  {
  for (Byte_Idx=0; Byte_Idx<3; Byte_Idx++)
    {
    SFRPAGE   = ACTIVE_PAGE;
    SPI0DAT = DAC8760_SPI_Frame_Wr[DAC_Idx].byte[Byte_Idx];             // Individually transmit 3 bytes to each DAC in daisy-chain (**8-bit address, 16-bit register value)
    SPI_Done = 0;
    while (!SPI_Done);                                                  // Wait for SPI byte transfer to complete
    }
  Latch = 1;                                                            // Latch each 24-bit DAC command individually during initialization
  for (index=0; index<4; index++) _nop_();                              // Wait 1us
  Latch = 0;                                                            // Set Latch line to its nominal low state
  }
}

// ******************************************************************************
//  Function Name: DAC8760_Data_Transfer
//        Purpose: Put Data in the requested Address of all
//                 three daisy-chained DAC8760's
//          Input: Address - Register address of DAC8760
//                 Vapor_DAC_RegData - Vapor DAC Register Data
//                 Pressure_DAC_RegData - Pressure DAC Register Data
//                 Temperature_DAC_RegData - Temperature DAC Register Data
//         Output: DAC8760_SPI_Frame_Rd - SPI data output for all DAC8760's
//
// ******************************************************************************
void DAC8760_Data_Transfer(unsigned char Address, unsigned short Vapor_DAC_RegData, unsigned short Pressure_DAC_RegData,unsigned short Temperature_DAC_RegData)
{
unsigned char index;
ppmv_SPI_Wr.SPI_Frame.Address = Address;                                // Load address and register data into SPI byte stream for all three daisy-chained DACs
ppmv_SPI_Wr.SPI_Frame.RegData = Vapor_DAC_RegData;
mBar_SPI_Wr.SPI_Frame.Address = Address;
mBar_SPI_Wr.SPI_Frame.RegData = Pressure_DAC_RegData;
degC_SPI_Wr.SPI_Frame.Address = Address;
degC_SPI_Wr.SPI_Frame.RegData = Temperature_DAC_RegData;
for (DAC_Idx=0; DAC_Idx<3; DAC_Idx++)                                   // Output SPI data to daisy-chained DACs
  {
  for (Byte_Idx=0; Byte_Idx<3; Byte_Idx++)
    {
    SFRPAGE   = ACTIVE_PAGE;
    SPI0DAT = DAC8760_SPI_Frame_Wr[DAC_Idx].byte[Byte_Idx];             // Individually transmit 3 bytes to each DAC in daisy-chain (**8-bit address, 16-bit register value)
    SPI_Done = 0;
    while (!SPI_Done);                                                  // Wait for SPI byte transfer to complete
    }
  }
Latch = 1;                                                              // Latch after all 72-bits in daisy-chain are loaded
for (index=0; index<4; index++) _nop_();                                // Wait 1us
Latch = 0;                                                              // Set Latch line to its nominal low state
}

// ******************************************************************************************************************
//  Function Name: DAC8760_Register_Read
//        Purpose: Put Data in the requested Address of all
//                 three daisy-chained DAC8760's
//          Input: none
//         Output: ppmv_DAC - Status, DAC Data, Control, Configuration, DAC Gain Calibration, DAC Zero Calibration
//                 mBar_DAC - Status, DAC Data, Control, Configuration, DAC Gain Calibration, DAC Zero Calibration
//                 degC_DAC - Status, DAC Data, Control, Configuration, DAC Gain Calibration, DAC Zero Calibration
//
// ******************************************************************************************************************
void DAC8760_Register_Read(void)
{
DAC8760_Data_Transfer(Rd_Register_Adr,Rd_Status_Reg,Rd_Status_Reg,Rd_Status_Reg);                       // Request Status Register read from daisy-chained DAC8760's
DAC8760_Data_Transfer(NOP_Adr,0x0,0x0,0x0);                                                             // Send NOP to daisy-chained DAC8760's
ppmv_DAC.Status_Reg_Val = ppmv_SPI_Rd.SPI_Frame.RegData;                                                // Retrieve all three DAC8760 Status Register values
mBar_DAC.Status_Reg_Val = mBar_SPI_Rd.SPI_Frame.RegData;
degC_DAC.Status_Reg_Val = degC_SPI_Rd.SPI_Frame.RegData;
DAC8760_Data_Transfer(Rd_Register_Adr,Rd_DAC_Data_Reg,Rd_DAC_Data_Reg,Rd_DAC_Data_Reg);                 // Request DAC Data Register read from daisy-chained DAC8760's
DAC8760_Data_Transfer(NOP_Adr,0x0,0x0,0x0);                                                             // Send NOP to daisy-chained DAC8760's
ppmv_DAC.DAC_Data_Reg_Val = ppmv_SPI_Rd.SPI_Frame.RegData;                                              // Retrieve all three DAC8760 DAC Data Register values
mBar_DAC.DAC_Data_Reg_Val = mBar_SPI_Rd.SPI_Frame.RegData;
degC_DAC.DAC_Data_Reg_Val = degC_SPI_Rd.SPI_Frame.RegData;
DAC8760_Data_Transfer(Rd_Register_Adr,Rd_Control_Reg,Rd_Control_Reg,Rd_Control_Reg);                    // Request Control Register read from daisy-chained DAC8760's
DAC8760_Data_Transfer(NOP_Adr,0x0,0x0,0x0);                                                             // Send NOP to daisy-chained DAC8760's
ppmv_DAC.Control_Reg_Val = ppmv_SPI_Rd.SPI_Frame.RegData;                                               // Retrieve all three DAC8760 Control Register values
mBar_DAC.Control_Reg_Val = mBar_SPI_Rd.SPI_Frame.RegData;
degC_DAC.Control_Reg_Val = degC_SPI_Rd.SPI_Frame.RegData;
DAC8760_Data_Transfer(Rd_Register_Adr,Rd_Config_Reg,Rd_Config_Reg,Rd_Config_Reg);                       // Request Configuration Register read from daisy-chained DAC8760's
DAC8760_Data_Transfer(NOP_Adr,0x0,0x0,0x0);                                                             // Send NOP to daisy-chained DAC8760's
ppmv_DAC.Config_Reg_Val = ppmv_SPI_Rd.SPI_Frame.RegData;                                                // Retrieve all three DAC8760 Configuration Register values
mBar_DAC.Config_Reg_Val = mBar_SPI_Rd.SPI_Frame.RegData;
degC_DAC.Config_Reg_Val = degC_SPI_Rd.SPI_Frame.RegData;
```

```
DAC8760_Data_Transfer(Rd_Register_Adr,Rd_DAC_GainCal_Reg,Rd_DAC_GainCal_Reg,Rd_DAC_GainCal_Reg);                    // Request DAC Gain Calibration Register read from daisy-chained DAC8760's
DAC8760_Data_Transfer(NOP_Adr,0x0,0x0,0x0);                                                                          // Send NOP to daisy-chained DAC8760's
ppmv_DAC.DAC_GainCal_Reg_Val = ppmv_SPI_Rd.SPI_Frame.RegData;                                                        // Retrieve all three DAC8760 DAC Gain Calibration Register values
mBar_DAC.DAC_GainCal_Reg_Val = mBar_SPI_Rd.SPI_Frame.RegData;
degC_DAC.DAC_GainCal_Reg_Val = degC_SPI_Rd.SPI_Frame.RegData;
DAC8760_Data_Transfer(Rd_Register_Adr,Rd_DAC_ZeroCal_Reg,Rd_DAC_ZeroCal_Reg,Rd_DAC_ZeroCal_Reg);                    // Request DAC Zero Calibration Register read from daisy-chained DAC8760's
DAC8760_Data_Transfer(NOP_Adr,0x0,0x0,0x0);                                                                          // Send NOP to daisy-chained DAC8760's
ppmv_DAC.DAC_ZeroCal_Reg_Val = ppmv_SPI_Rd.SPI_Frame.RegData;                                                        // Retrieve all three DAC8760 DAC Zero Calibration Register values
mBar_DAC.DAC_ZeroCal_Reg_Val = mBar_SPI_Rd.SPI_Frame.RegData;
degC_DAC.DAC_ZeroCal_Reg_Val = degC_SPI_Rd.SPI_Frame.RegData;
}

//-----------------------------------------------------------------------------
// Interrupt Service Routines
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Timer0_ISR
//-----------------------------------------------------------------------------
//
void Timer0_ISR (void) interrupt 1
{
if (SDG_43A_Xmit_Period != 0) --SDG_43A_Xmit_Period;                        // decrement delay time for SDG_43A transmit period
if (WVSS_II_TimeOut_Period != 0) --WVSS_II_TimeOut_Period;                  // decrement delay time for WVSS_II time-out period
}

//-----------------------------------------------------------------------------
// UART0_ISR
//-----------------------------------------------------------------------------
//
void UART0_ISR (void) interrupt 4
{
if (RI0)                                                                     // handle pending serial reception interrupt
  {
  Serial_RcvBuf0[Serial_BufTail0++] = SBUF0;                                 // load character from receiver buffer into command buffer
  if (Serial_BufCount0<SerialBufSize) ++Serial_BufCount0;                    // increment buffer count if not full
  if (Serial_BufTail0==SerialBufSize) Serial_BufTail0 = 0;                   // check for end of buffer
  if (Serial_BufFull0==1) Serial_BufHead0 = Serial_BufTail0;                 // if buffer full, overwrite oldest data by moving the head with the tail
  if (Serial_BufTail0==Serial_BufHead0) Serial_BufFull0 = 1;                 // if buffer tail moves into buffer head, buffer is full
  Serial_BufEmpty0 = 0;                                                      // indicate serial buffer not empty
  RI0 = 0;                                                                   // clear receive interrupt flag
  }
if (TI0)                                                                     // handle pending serial transmission interrupt
  {
  PendingXmit0 = 0;                                                          // clear pending transmission flag
  TI0 = 0;                                                                   // clear transmit interrupt flag
  }
}

//-----------------------------------------------------------------------------
// UART0_ISR
//-----------------------------------------------------------------------------
//
void UART1_ISR (void) interrupt 18
{
if (RI1)                                                                     // handle pending serial reception interrupt
  {
  Serial_RcvBuf1[Serial_BufTail1++] = SBUF1;                                 // load character from receiver buffer into command buffer
  if (Serial_BufCount1<SerialBufSize) ++Serial_BufCount1;                    // increment buffer count if not full
  if (Serial_BufTail1==SerialBufSize) Serial_BufTail1 = 0;                   // check for end of buffer
  if (Serial_BufFull1==1) Serial_BufHead1 = Serial_BufTail1;                 // if buffer full, overwrite oldest data by moving the head with the tail
  if (Serial_BufTail1==Serial_BufHead1) Serial_BufFull1 = 1;                 // if buffer tail moves into buffer head, buffer is full
  Serial_BufEmpty1 = 0;                                                      // indicate serial buffer not empty
  RI1 = 0;                                                                   // clear receive interrupt flag
  }
if (TI1)                                                                     // handle pending serial transmission interrupt
  {
  PendingXmit1 = 0;                                                          // clear pending transmission flag
  TI1 = 0;                                                                   // clear transmit interrupt flag
  }
}

//-----------------------------------------------------------------------------
// SPI0_ISR
//-----------------------------------------------------------------------------
//
void SPI0_ISR (void) interrupt 6
{
SPIF = 0;                                                                    // clear SPI0 interrupt flag
SPI_Done = 1;                                                                // indicate SPI transfer is complete
DAC8760_SPI_Frame_Rd[DAC_Idx].byte[Byte_Idx] = SPI0DAT;                      // load SPI0 output data
}
```

```
//
//  This file is part of the Franklin Software 8051 V6 Compiler package
//  Copyright (c) Franklin Software, Inc., 1991-1997
//
//  File Name:  PUTCHAR.C
//    Version:  2.01
//   Modified:  17.04.97 - SWB Changed the function of be NON reentrant
//              02.04.97 - SWB Created for C51 V6
//      Usage:  For ProView simply add this file into your project
//              directory and to your project list in the usual
//              manner.
//
//              For DOS users compile this file with C51, like this:
//              C51 CALLOC.C [memory_model]
//
//              Link the file with l51 into your app. like this:
//              L51 <.OBJ_list>, CALLOC.OBJ [L51_controls]
//
//  NOTE:  It is NOT recommended that this modified file be incorporated
//         into your standard Franklin Software library files.
//
//#include <reg51.h>
#define NumColumns                                          100

xdata unsigned char String[NumColumns],StringPtr=0;

int putchar( const char character )
  {
   if (character=='\n') StringPtr = 0;
   if ((character!='\n')&&(StringPtr<NumColumns)) String[StringPtr++]=character;
   return character;
  }
```

Microcontroller C8051F582 - U1

SILICON LABS

C8051F58x/F59x

Mixed Signal ISP Flash MCU Family

Analog Peripherals
- **12-Bit ADC**
  - Up to 200 ksps
  - Up to 32 external single-ended inputs
  - VREF from on-chip VREF, external pin or V_{DD}
  - Internal or external start of conversion source
  - Built-in temperature sensor
- **Three Comparators**
  - Programmable hysteresis and response time
  - Configurable as interrupt or reset source
  - Low current

On-Chip Debug
- On-chip debug circuitry facilitates full speed, non-intrusive in-system debug (no emulator required)
- Provides breakpoints, single stepping, inspect/modify memory and registers
- Superior performance to emulation systems using ICE-chips, target pods, and sockets
- Low cost, complete development kit

Supply Voltage 1.8 to 5.25 V
- Typical operating current: 15 mA at 50 MHz; Typical stop mode current: 230 µA

High-Speed 8051 µC Core
- Pipelined instruction architecture; executes 70% of instructions in 1 or 2 system clocks
- Up to 50 MIPS throughput with 50 MHz clock
- Expanded interrupt handler

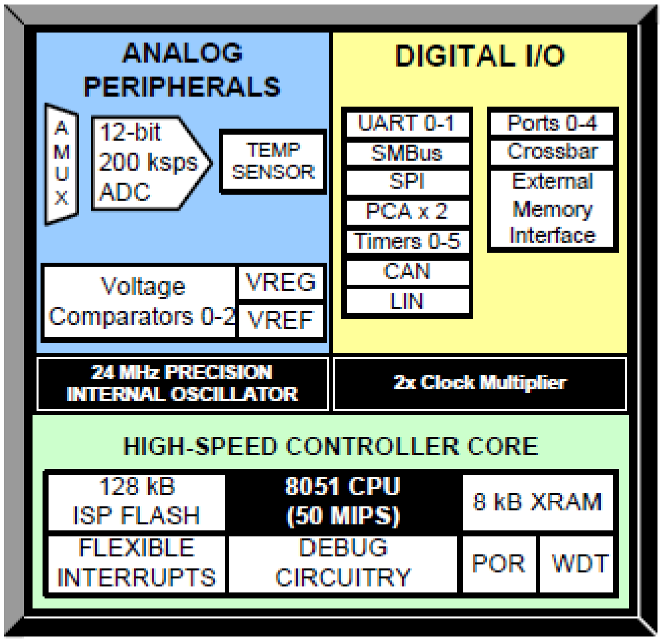
Automotive Qualified
- Temperature Range: –40 to +125 °C

Memory
- 8448 bytes internal data RAM (256 + 8192 XRAM)
- 128 or 96 kB Banked Flash; In-system programmable in 512-byte Sectors
- External 64 kB data memory interface programmable for multiplexed or non-multiplexed mode

Digital Peripherals
- 40, 33, or 25 Port I/O; All 5 V push-pull with high sink current
- CAN 2.0 Controller—no crystal required
- LIN 2.1 Controller (Master and Slave capable); no crystal required
- Two Hardware enhanced UARTs, SMBus™, and enhanced SPI™ serial ports
- Six general purpose 16-bit counter/timers
- Two 16-Bit programmable counter array (PCA) peripherals with six capture/compare modules each and enhanced PWM functionality

Clock Sources
- Internal 24 MHz with ±0.5% accuracy for CAN and master LIN operation.
- External oscillator: Crystal, RC, C, or clock (1 or 2 pin modes)
- Can switch between clock sources on-the-fly; useful in power saving modes

Packages
- 48-Pin QFP/QFN (C8051F580/1/4/5)
- 40-Pin QFN (C8051F588/9-F590/1)
- 32-Pin QFP/QFN (C8051F582/3/6/7)



RS-232 Transceiver MAX3235E - U2

19-1473; Rev 2; 8/04

MAXIM

±15kV ESD-Protected, 1µA, 250kbps, 3.3V/5V, Dual RS-232 Transceivers with Internal Capacitors

General Description

The MAX3233E/MAX3235E are EIA/TIA-232 and V.28/V.24 communications interfaces with automatic shutdown/wake-up features, high data-rate capabilities, and enhanced electrostatic discharge (ESD) protection. All transmitter outputs and receiver inputs are protected to ±15kV using IEC 1000-4-2 Air-Gap Discharge, to ±8kV using IEC 1000-4-2 Contact Discharge, and to ±15kV using the Human Body Model. The MAX3233E operates from a +3.3V supply; the MAX3235E operates from +5.0V.

All devices achieve a 1µA supply current using Maxim's revolutionary AutoShutdown Plus™ feature. These devices automatically enter a low-power shutdown mode when the following two conditions occur: either the RS-232 cable is disconnected or the transmitters of the connected peripherals are inactive, and the UART driving the transmitter inputs is inactive for more than 30 seconds. They turn on again when they sense a valid transition at any transmitter or receiver input. AutoShutdown Plus saves power without changes to the existing BIOS or operating system.

The MAX3233E/MAX3235E have internal dual charge pumps requiring no external capacitors. Both transceivers have a proprietary low-dropout transmitter output stage that enables true RS-232 performance from a +3.0V to +3.6V supply for the MAX3233E or a +4.5V to +5.5V supply for the MAX3235E. These devices are guaranteed to operate up to 250kbps. Both are available in space-saving 20-pin wide SO or plastic DIP packages.

Applications

Subnotebook and Palmtop Computers
Cellular Phones
Battery-Powered Equipment
Handheld Equipment
Peripherals
Embedded Systems

Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX3233ECWP	0°C to +70°C	20 SO
MAX3233ECPP	0°C to +70°C	20 Plastic DIP
MAX3233EEWP	-40°C to +85°C	20 SO
MAX3233EEPP	-40°C to +85°C	20 Plastic DIP

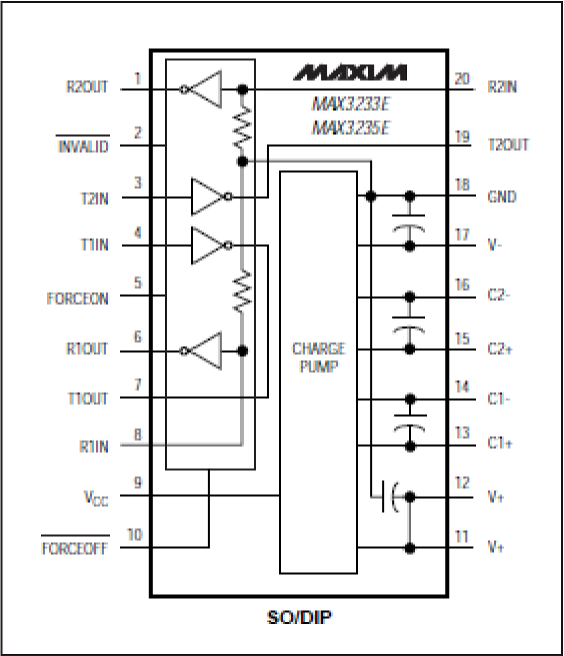
Ordering Information continued at end of data sheet.

AutoShutdown Plus is a trademark of Maxim Integrated Products, Inc.

Features

- ♦ ESD Protection for RS-232 I/O Pins
  - ±15kV—Human Body Model
  - ±8kV—IEC 1000-4-2, Contact Discharge
  - ±15kV—IEC 1000-4-2, Air-Gap Discharge
- ♦ Latchup Free
- ♦ 1µA Supply Current
- ♦ AutoShutdown Plus—1997 EDN Magazine Innovation of the Year
- ♦ Single-Supply Operation
  - +3.0V to +3.6V (MAX3233E)
  - +4.5V to +5.5V (MAX3235E)
- ♦ 250kbps Guaranteed Data Rate
- ♦ 6V/µs Guaranteed Slew Rate
- ♦ Meets EIA/TIA-232 Specifications Down to 3.0V (MAX3233E)
- ♦ Internal Charge-Pump Capacitors

Pin Configuration/ Functional Diagram



Typical Operating Circuit appears at end of data sheet.

For pricing, delivery, and ordering information, please contact Maxim/Dallas Direct! at 1-888-629-4642, or visit Maxim's website at www.maxim-ic.com.

Isolated RS-232 Transceiver ADM3251E - U3

ANALOG DEVICES

Isolated, Single-Channel RS-232 Line Driver/Receiver

Data Sheet **ADM3251E**

FEATURES

- 2.5 kV fully isolated (power and data) RS-232 transceiver
- *iso*Power integrated, isolated dc-to-dc converter
- 460 kbps data rate
- 1 Tx and 1 Rx
- Meets EIA/TIA-232E specifications
- ESD protection on R_{IN} and T_{OUT} pins
  - ±8 kV: contact discharge
  - ±15 kV: air gap discharge
- 0.1 μF charge pump capacitors
- High common-mode transient immunity: >25 kV/μs
- Safety and regulatory approvals
  - UL recognition
    - 2500 V rms for 1 minute per UL 1577
  - VDE Certificate of Conformity
    - DIN EN 60747-5-2 (VDE 0884 Teil 2): 2003-01
  - CSA Component Acceptance Notice #5A
- Operating temperature range: −40°C to +85°C
- Wide body, 20-lead SOIC package

APPLICATIONS

- High noise data communications
- Industrial communications
- General-purpose RS232 data links
- Industrial/telecommunications diagnostic ports
- Medical equipment

FUNCTIONAL BLOCK DIAGRAM

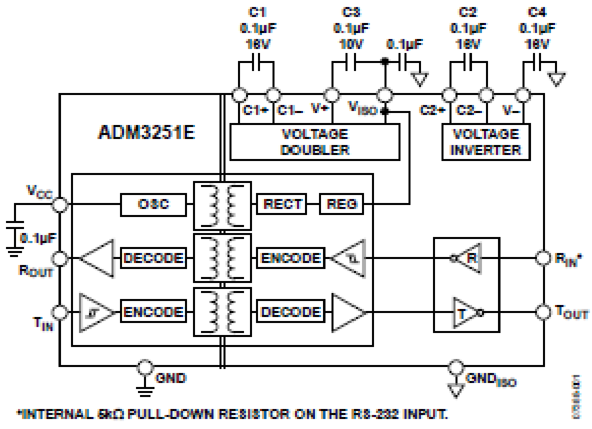


Figure 1.

GENERAL DESCRIPTION

The ADM3251E[1] is a high speed, 2.5 kV fully isolated, single-channel RS-232/V.28 transceiver device that operates from a single 5 V power supply. Due to the high ESD protection on the R_{IN} and T_{OUT} pins, the device is ideally suited for operation in electrically harsh environments or where RS-232 cables are frequently being plugged and unplugged.

The ADM3251E incorporates dual-channel digital isolators with *iso*Power™ integrated, isolated power. There is no requirement to use a separate isolated dc-to-dc converter. Chip-scale transformer *i*Coupler® technology from Analog Devices, Inc., is used both for the isolation of the logic signals as well as for the integrated dc-to-dc converter. The result is a total isolation solution.

The ADM3251E contains *iso*Power technology that uses high frequency switching elements to transfer power through the transformer. Special care must be taken during printed circuit board (PCB) layout to meet emissions standards. Refer to Application Note AN-0971, *Control of Radiated Emissions with isoPower Devices*, for details on board layout considerations.

The ADM3251E conforms to the EIA/TIA-232E and ITU-T V. 28 specifications and operates at data rates up to 460 kbps.

Four external 0.1 μF charge pump capacitors are used for the voltage doubler/inverter, permitting operation from a single 5 V supply.

The ADM3251E is available in a 20-lead, wide body SOIC package and is specified over the −40°C to +85°C temperature range.

[1] Protected by U.S. Patents 5,952,849; 6,873,065; and 7,075,329.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 ©2008–2013 Analog Devices, Inc. All rights reserved.
Technical Support www.analog.com

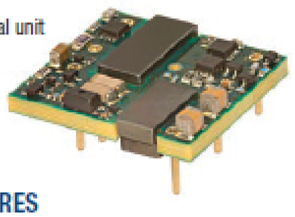
Isolated DC-DC converters UEI15-050-Q12PM-C, UEI15-150-Q12P-C- U4, U5

Murata Power Solutions

UEI15 Series

Isolated Wide Input Range 15-Watt DC-DC Converters

Typical unit



Featuring a full 15 Watt or greater output in one square inch of board area, the UEI series isolated DC/DC converter family offers efficient regulated DC power for printed circuit board mounting.

FEATURES

- Small footprint DC/DC converter, ideal for high current applications
- Industry standard 0.96" x 1.1" X 0.33" open frame package and pinout
- Wide range input voltages 9-36 and 18-75 Vdc
- Assembly and attachment for RoHS standards
- Isolation up to 2250 VDC (basic)
- Up to 15 Watts or greater total output power with overtemperature shutdown
- High efficiency synchronous rectifier forward topology
- Stable operation with no required external components
- Usable -40 to 85°C temperature range (with derating)
- Certified to UL 60950-1, CAN/CSA-C22.2 No. 60950-1, IEC60950-1, EN60950-1 safety approvals, 2nd edition
- Extensive self-protection shut down features

PRODUCT OVERVIEW

Wide range 4:1 inputs on the 0.96" x 1.1" x 0.33" converter are either 9 to 36 Volts DC (Q12 models) or 18 to 75 Volts DC (Q48 models), ideal for battery-powered and telecom equipment. The industry-standard pinout fits larger 1" x 2" converters. Fixed output voltages from 3.3 VDC to 15 VDC are regulated to within ±0.2% or less and may be trimmed within ±10% of nominal output. Applications include small instruments, area-limited microcontrollers, computer-based systems, data communications equipment, remote sensor systems, vehicle and portable electronics.

The UEI 15W series includes full magnetic and optical isolation up to 2250 Volts DC (basic insulation). For connection to digital systems, the outputs offer fast settling to current step loads and tolerance of higher capacitive loads. Excellent ripple and noise specifications assure compatibility to circuits using CPU's, ASIC's, programmable logic and FPGA's. For systems requiring controlled startup/shutdown, an external switch, transistor or digital logic may be used to activate the remote On/Off control.

A wealth of self-protection features avoid both converter and external circuit problems. These include input undervoltage lockout and overtemperature shutdown. The outputs current limit using the "hiccup" autorestart technique and the outputs may be short-circuited indefinitely. Additional features include output overvoltage and reverse conduction elimination.

The high efficiency offers minimal heat buildup and "no fan" operation.

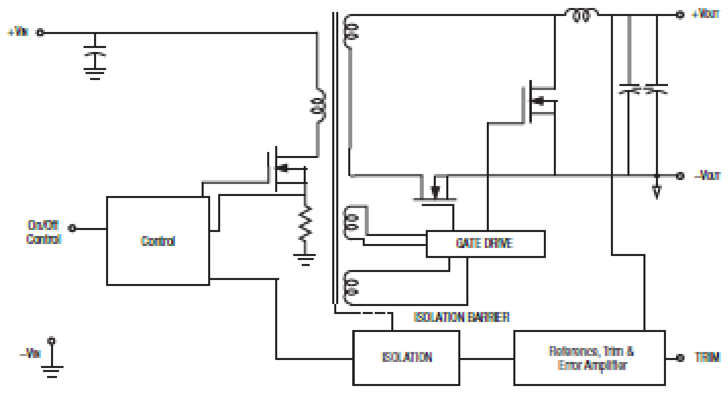


Figure 1. Simplified block diagram—3.3V and 5VOUT models only. Typical topology is shown.

RoHS COMPLIANT For full details go to www.murata-ps.com/rohs

www.murata-ps.com/support

16-Bit Programmable Voltage/Current DAC DAC8760 - U6 - U8

Product Folder | Order Now | Technical Documents | Tools & Software | Support & Community | Reference Design

TEXAS INSTRUMENTS

DACx760 Single-Channel, 12- and 16-Bit Programmable Current and Voltage Output Digital-to-Analog Converters for 4-mA to 20-mA Current Loop Applications

1 Features

- Current Output: 4 mA to 20 mA; 0 mA to 20 mA; 0 mA to 24 mA
- Voltage Output:
  - 0 V to 5 V; 0 V to 10 V; ±5 V; ±10 V
  - 0 V to 5.5 V; 0 V to 11 V; ±5.5 V; ±11 V (10% Over Range)
- ±0.1% FSR Total Unadjusted Error (TUE) Maximum
- DNL: ±1 LSB Maximum
- Simultaneous Voltage and Current Output
- Internal 5-V Reference (10 ppm/°C, Maximum)
- Internal 4.6-V Power-Supply Output
- Reliability Features:
  - CRC Check and Watchdog Timer
  - Thermal Alarm
  - Open Alarm, Short Current Limit
- Wide Temperature Range: –40°C to 125°C
- 6-mm × 6-mm 40-Pin VQFN and 24-Pin HTSSOP Packages

2 Applications

- 4-mA to 20-mA Current Loops
- Analog Output Modules
- Building Automation
- Environment Monitoring
- Programmable Logic Controllers (PLCs)
- Field Sensors and Process Transmitters

3 Description

The DAC7760 and DAC8760 are low-cost, precision, fully-integrated, 12-bit and 16-bit digital-to-analog converters (DACs) designed to meet the requirements of industrial process-control applications. These devices can be programmed as a current output with a range of 4 mA to 20 mA, 0 mA to 20 mA, or 0 mA to 24 mA; or as a voltage output with a range of 0 V to 5 V, 0 V to 10 V, ±5 V, or ±10 V with a 10% overrange (0 V to 5.5 V, 0 V to 11 V, ±5.5 V, or ±11 V). Both current and voltage outputs can be simultaneously enabled while being controlled by a single data register.

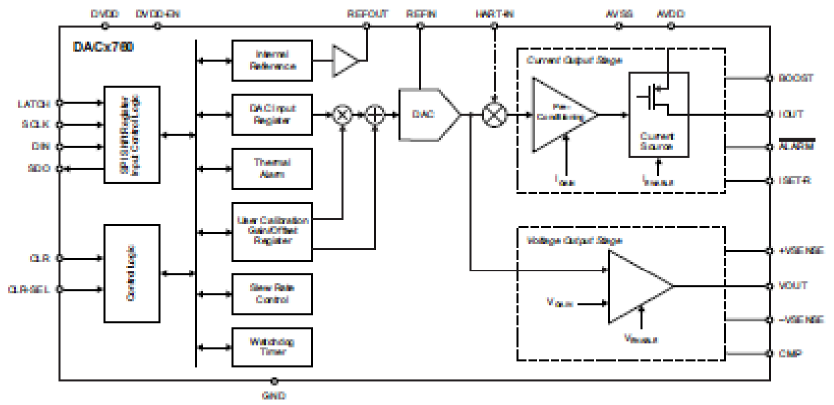
These devices include a power-on-reset function to ensure powering up in a known state (both IOUT and VOUT are disabled and in a Hi-Z state). The CLR and CLR-SEL pins set the voltage outputs to zero-scale or midscale, and the current output to the low end of the range, if the output is enabled. Zero and gain registers can be programmed to digitally calibrate the device in the end system. The output slew rate is also programmable by register. These devices can superimpose an external HART® signal on the current output and can operate with either a single 10-V to 36-V supply, or dual supplies of up to ±18 V. All versions are available in both 6-mm × 6-mm 40-pin VQFN and 24-pin HTSSOP packages.

Device Information[1]

PART NUMBER	PACKAGE	BODY SIZE (NOM)
DACx760	HTSSOP (24)	7.80 mm × 4.40 mm
	VQFN (40)	6.00 mm × 6.00 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Block Diagram



Quad-Channel Digital Isolators ISO7631DW, IS7641DW - U9 - U11

Product Folder | Sample & Buy | Technical Documents | Tools & Software | Support & Community

ISO76x1 Low-Power Triple and Quad-Channels Digital Isolators

1 Features

- Signaling Rate: 150 Mbps (M-Grade), 25 Mbps (C-Grade)
- Robust Design with Integrated Noise Filter (C-Grade)
- Low Power Consumption, Typical I_{CC} per Channel (3.3-V Supplies):
  - ISO7631FM: 2 mA at 10 Mbps
  - ISO7631FC: 1.5 mA at 10 Mbps
  - ISO7641FC: 1.3 mA at 10 Mbps
- Extremely-Low $I_{CC\_disable}$ (C-Grade)
- Low Propagation Delay: 7 ns Typical (M-Grade)
- Output Defaults to Low-State in Fail-Safe Mode
- Wide Temperature Range: –40°C to 125°C
- 50 KV/µs Transient Immunity, Typical
- Long Life With SiO_2 Isolation Barrier
- Operates From 2.7-V (M-Grade), 3.3-V and 5-V Supply and Logic Levels
- 2.7-V (M-Grade), 3.3-V and 5-V Level Translation
- Wide Body SOIC-16 Package
- Safety and Regulatory Approvals
  - 2500 V_{RMS} Isolation for 1 Minute per UL 1577
  - 4242 V_{PK} Basic Insulation per DIN V VDE V 0884-10 and DIN EN 61010-1
  - CSA Component Acceptance Notice 5A, IEC 60950-1 and IEC 61010-1 End Equipment Standards
  - CQC Certification per GB4943.1-2011
  - TUV 3000 V_{RMS} Reinforced Insulation according to EN/UL/CSA 60950-1 and EN/UL/CSA 61010-1

2 Applications

- Optocoupler Replacement in:
  - Industrial Fieldbus
    - Profibus
    - Modbus
    - DeviceNet™ Data Buses
  - Servo Control Interface
  - Motor Control
  - Power Supplies
  - Battery Packs

3 Description

The ISO7631F and ISO7641F devices provide galvanic isolation up to 4242 V_{PK} per VDE. The ISO7631F device has three channels, two of which operate in the forward direction and one which operates in the reverse direction. The ISO7641F device has 4 channels, three of which operate in the forward direction and one of which operates in the reverse direction. Suffix F indicates that output defaults to low-state in fail-safe conditions (see). M-Grade devices are high-speed isolators capable of up to150-Mbps data rates with fast propagation delays, whereas C-Grade devices are capable of up to 25-Mbps data rates with low power consumption and integrated filters for noise-prone applications. C-Grade devices are recommended for lower-speed applications where input noise pulses of less than 6 ns duration must be suppressed, or when low-power consumption is critical.

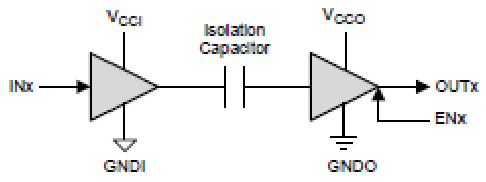
Each isolation channel has a logic input and output buffer separated by a silicon dioxide (SiO_2) insulation barrier. Used in conjunction with isolated power supplies, these devices prevent noise currents on a data bus or other circuits from entering the local ground and interfering with or damaging sensitive circuitry. The devices have TTL input thresholds and can operate from 2.7-V (M-Grade), 3.3-V and 5-V supplies. All inputs are 5-V tolerant when supplied from 3.3-V or 2.7-V supplies.

Device Information[(1)]

PART NUMBER	PACKAGE	BODY SIZE (NOM)
ISO7631FM	SOIC (16)	10.30 mm × 7.50 mm
ISO7631FC		
ISO7641FC		

(1) For all available packages, see the orderable addendum at the end of the datasheet.

Simplified Schematic



Copyright © 2016, Texas Instruments Incorporated

(1) V_{CCI} and GNDI are supply and ground connections respectively for the input channels.

(2) V_{CCO} and GNDO are supply and ground connections respectively for the output channels.

Analog Switches MAX4622 - U12 - U14

19-1497; Rev 0; 8/99

MAXIM

Dual, 5Ω Analog Switches

General Description

The MAX4621/MAX4622/MAX4623 are precision, dual, high-speed analog switches. The single-pole/single-throw (SPST) MAX4621 and double-pole/single-throw (DPST) MAX4623 dual switches are normally open (NO). The single-pole/double-throw (SPDT) MAX4622 has two normally closed (NC) and two NO poles. All three parts offer low 5Ω on-resistance guaranteed to match to within 0.5Ω between channels and to remain flat over the full analog signal range (Δ0.5Ω max). They also offer low leakage (<500pA at +25°C, <5nA at +85°C) and fast switching times (turn-on time <250ns, turn-off time <200ns).

These analog switches are ideal in low-distortion applications and are the preferred solution over mechanical relays in automatic test equipment or applications where current switching is required. They have low power requirements, use less board space, and are more reliable than mechanical relays.

The MAX4621/MAX4622/MAX4623 are pin-compatible replacements for the DG401/DG403/DG405, respectively, offering improved overall performance. These monolithic switches operate from a single positive supply (+4.5V to +36V) or with bipolar supplies (±4.5V to ±18V) while retaining CMOS-logic input compatibility.

Rail-to-Rail is a registered trademark of Nippon Motorola, Ltd.

Features

- ♦ Low On-Resistance: 3Ω (typ), 5Ω (max)
- ♦ Guaranteed R_{ON} Match Between Channels (0.5Ω max)
- ♦ Guaranteed Break-Before-Make Operation (MAX4622)
- ♦ Guaranteed Off-Channel Leakage <5nA at +85°C
- ♦ Single-Supply Operation (+4.5V to +36V) Bipolar-Supply Operation (±4.5V to ±18V)
- ♦ TTL/CMOS-Logic Compatible
- ♦ Rail-to-Rail® Analog Signal Handling Capability
- ♦ Pin Compatible with DG401/DG403/DG405

Applications

Reed Relay Replacement	Military Radios
Test Equipment	PBX, PABX Systems
Communication Systems	Audio-Signal Routing
Data-Acquisition Systems	Avionics

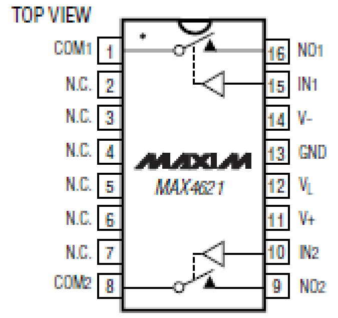
Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX4621CSE	0°C to +70°C	16 Narrow SO
MAX4621CPE	0°C to +70°C	16 Plastic DIP

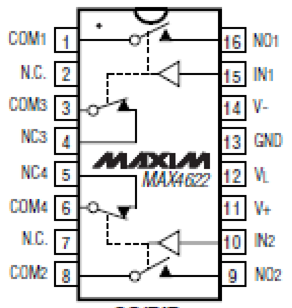
Ordering Information continued at end of data sheet.

Pin Configurations/Functional Diagrams/Truth Tables

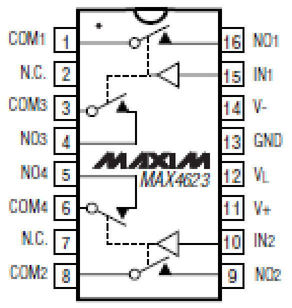
TOP VIEW



SO/DIP



SO/DIP



SO/DIP

MAX4621

LOGIC	SWITCH
0	OFF
1	ON

MAX4622

LOGIC	SWITCHES 1, 2	SWITCHES 3, 4
0	OFF	ON
1	ON	OFF

MAX4623

LOGIC	SWITCH
0	OFF
1	ON

SWITCHES SHOWN FOR LOGIC "0" INPUT

For free samples & the latest literature: http://www.maxim-ic.com, or phone 1-800-998-8800. For small orders, phone 1-800-835-8769.

LT3092

200mA 2-Terminal Programmable Current Source

FEATURES

- **Programmable 2-Terminal Current Source**
- **Maximum Output Current: 200mA**
- **Wide Input Voltage Range: 1.2V to 40V**
- **Input/Output Capacitors Not Required**
- **Resistor Ratio Sets Output Current**
- **Initial Set Pin Current Accuracy: 1%**
- **Reverse-Voltage Protection**
- **Reverse-Current Protection**
- <0.001%/V Line Regulation Typical
- Current Limit and Thermal Shutdown Protection
- Available in 8-Lead SOT-23, 3-Lead SOT-223 and 8-Lead 3mm × 3mm DFN Packages

APPLICATIONS

- 2-Terminal Floating Current Source
- GND Referred Current Source
- Variable Current Source
- In-Line Limiter
- Intrinsic Safety Circuits

DESCRIPTION

The LT®3092 is a programmable 2-terminal current source. It requires only two resistors to set an output current between 0.5mA and 200mA. A multitude of analog techniques lend themselves to actively programming the output current. The LT3092 is stable without input and output capacitors, offering high DC and AC impedance. This feature allows operation in intrinsically safe applications.

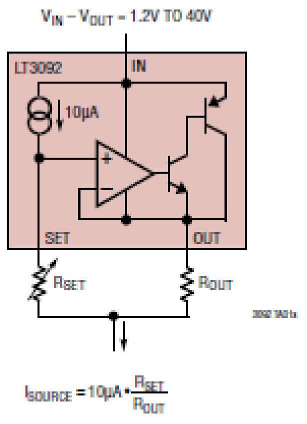
The SET pin features 1% initial accuracy and low temperature coefficient. Current regulation is better than 10ppm/V from 1.5V to 40V.

The LT3092 can operate in a 2-terminal current source configuration in series with signal lines. It is ideal for driving sensors, remote supplies, and as a precision current limiter for local supplies.

Internal protection circuitry includes reverse-battery and reverse-current protection, current limiting and thermal limiting. The LT3092 is offered in the 8-lead TSOT-23, 3-lead SOT-223 and 8-lead 3mm × 3mm DFN packages.

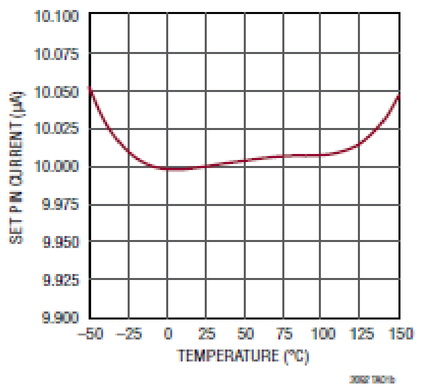
TYPICAL APPLICATION

Adjustable 2-Terminal Current Source



$I_{SOURCE} = 10\mu A \cdot \frac{R_{SET}}{R_{OUT}}$

SET Pin Current vs Temperature



Digital Line Protector DA112S1 - U18

DA108S1
DA112S1

Diode array

Features

- Array of 8 or 12 diodes
- Low input capacitance
- Suitable for digital line protection

Complies with following standards:

- IEC 61000-4-2 Level 4
  - 15kV (air discharge)
  - 8kV (contact discharge)

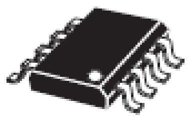

SO-8

Figure 1. Functional diagram: DA108S1

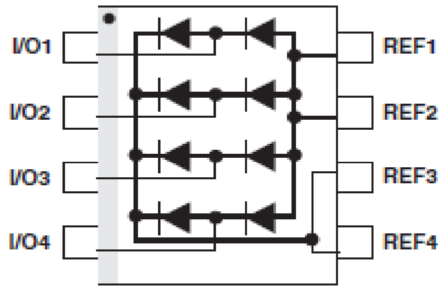
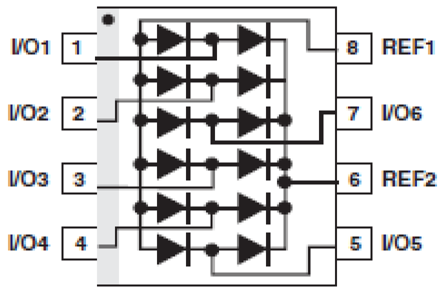


Figure 2. Functional diagram: DA112S1



Applications

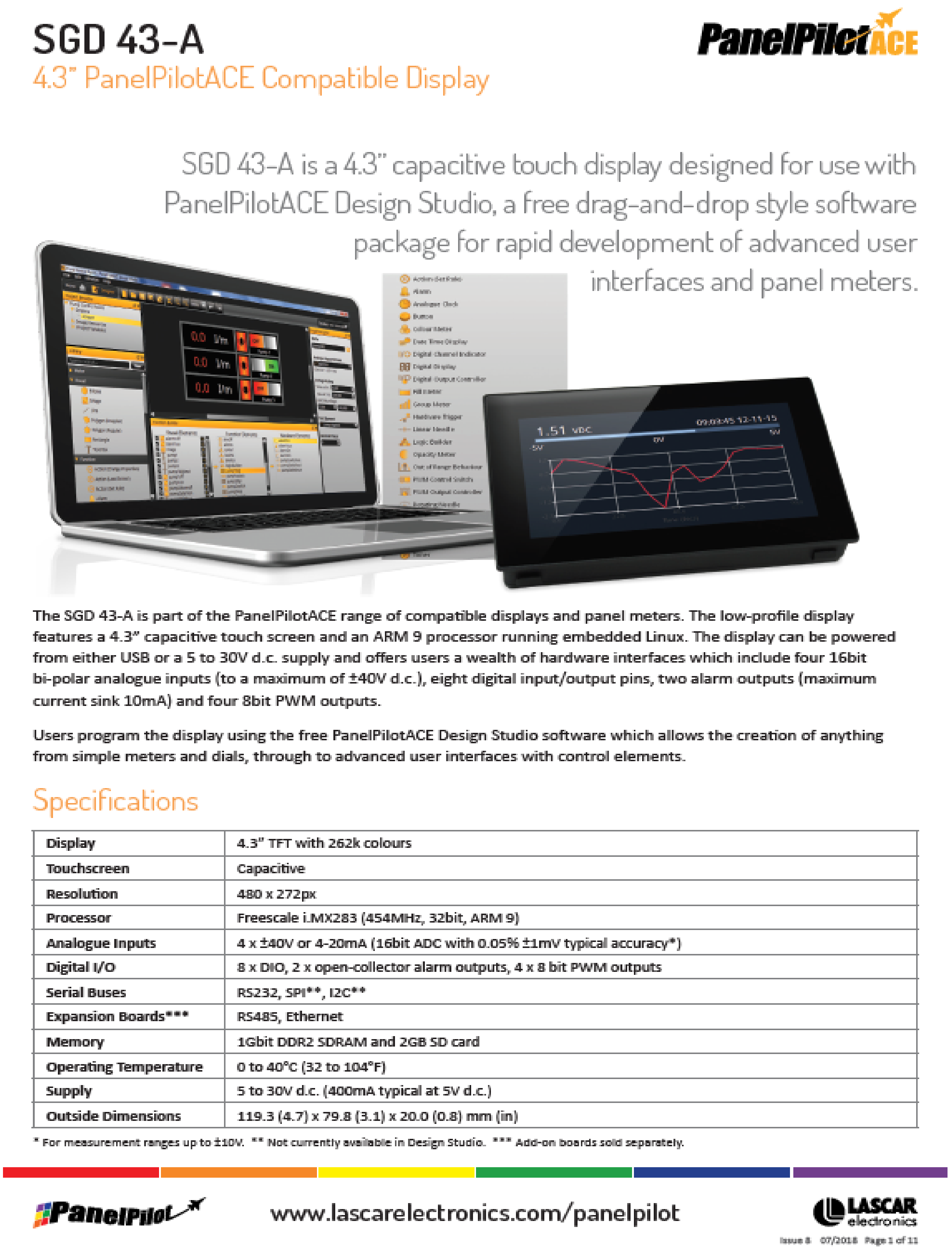
- Protection of logic side of ISDN S-interface
- Protection of I/O lines of microcontroller
- Signal conditioning

Description

Array of 8 or 12 diodes configured by cells of 2 diodes, each cell being used to protect signal line from transient overvoltages by clamping action.

As maximum voltage of each diode is 18 V, maximum input voltage range between two I/Os is either 0 V to 18 V (REF1 = 0 V and REF2 = +18 V) or -9 V to +9 V (REF1 = -9 V and REF2 = +9 V)

Panel Pilot ACE Display



SGD 43-A

4.3" PanelPilotACE Compatible Display

SGD 43-A is a 4.3" capacitive touch display designed for use with PanelPilotACE Design Studio, a free drag-and-drop style software package for rapid development of advanced user interfaces and panel meters.

The SGD 43-A is part of the PanelPilotACE range of compatible displays and panel meters. The low-profile display features a 4.3" capacitive touch screen and an ARM 9 processor running embedded Linux. The display can be powered from either USB or a 5 to 30V d.c. supply and offers users a wealth of hardware interfaces which include four 16bit bi-polar analogue inputs (to a maximum of ±40V d.c.), eight digital input/output pins, two alarm outputs (maximum current sink 10mA) and four 8bit PWM outputs.

Users program the display using the free PanelPilotACE Design Studio software which allows the creation of anything from simple meters and dials, through to advanced user interfaces with control elements.

Specifications

Display	4.3" TFT with 262k colours
Touchscreen	Capacitive
Resolution	480 x 272px
Processor	Freescale i.MX283 (454MHz, 32bit, ARM 9)
Analogue Inputs	4 x ±40V or 4-20mA (16bit ADC with 0.05% ±1mV typical accuracy*)
Digital I/O	8 x DIO, 2 x open-collector alarm outputs, 4 x 8 bit PWM outputs
Serial Buses	RS232, SPI**, I2C**
Expansion Boards* **	RS485, Ethernet
Memory	1Gbit DDR2 SDRAM and 2GB SD card
Operating Temperature	0 to 40°C (32 to 104°F)
Supply	5 to 30V d.c. (400mA typical at 5V d.c.)
Outside Dimensions	119.3 (4.7) x 79.8 (3.1) x 20.0 (0.8) mm (in)

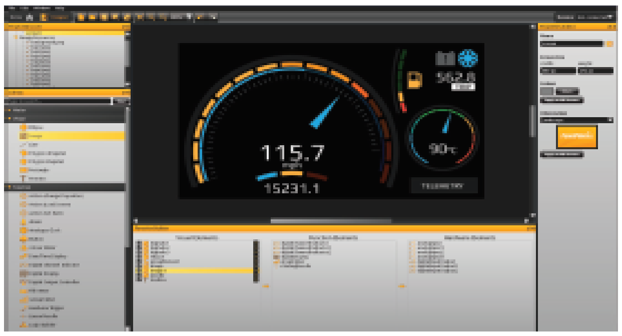
* For measurement ranges up to ±10V. ** Not currently available in Design Studio. *** Add-on boards sold separately.

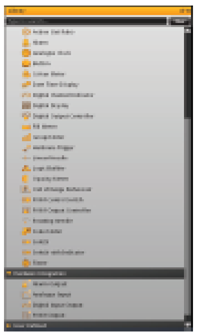
www.lascarelectronics.com/panelpilot

SGD 43-A

4.3" PanelPilotACE Compatible Display

PanelPilotACE Design Studio





Code-Free development for industrial display projects

The PanelPilotACE Design Studio software provides a number of building blocks which allow users to drag-and-drop elements onto the screen to quickly create advanced display interfaces. From images to text elements, analogue style meters, touch screen navigation elements, logic statements, data logging, trend graphing and even complex maths, users can build up multi-screen interfaces without needing to write a line of code.

There is a library of pre-defined elements such as meters, buttons and switches, and users can create their own content by combining elements or importing graphics in a number of formats (including jpg, png, tif, bmp and gif). The software includes support for transparency and multiple layers.

Hardware interfacing is similarly intuitive, with hardware elements being dragged into a function builder where associations with graphical elements (such as a needle on a meter) can be defined. Here users can determine scaling for analogue inputs, define alarm triggers, behaviours for digital inputs and outputs and configure PWM outputs.

PID Control:

Use your display as a PID control unit. Set P, I and D coefficients, measure a process variable and use PanelPilotACE to correct to your desired set point.

Ethernet FTP for Logged Data:

Connect your display to an FTP server via an Ethernet cable allowing quicker and more convenient data collection.

Multi-Channel Data Logging:

10 logging channels allowing logging of any input to the device that can be defined numerically.

Trend Graphs:

Display data on graphs as it is logged. If you use an alarm threshold as one of your data channels, monitor logged data against alarm levels.

Previewing and Uploading Projects

The software includes a 'Preview in Emulator' function which emulates the hardware's inputs and outputs, allowing users to test their projects prior to upload. Projects are uploaded to the SGD 43-A via a mini USB port.

PanelPilotACE Design Studio is compatible with Windows 7, 8 and 10, and can be downloaded free from www.lascarelectronics.com/software/panelpilotace.