**Intellitrol Firmware Changes for Version 1.7.0**

Product Numbers Affected:

15051 U17 Main Intellitrol hex file, Last Release 1.6.36

C-020-01 Source Code U17 Main Intellitrol, Last Release 1.6.36

This document will show the changes that were made to the Intellitrol firmware since the release of version 1.6.36.

The following support was added for the SuperTIM:

* Parameters can be read and (some) written by the TAS via the Modbus interface.
* A fault log was added to the SuperTIM memory to record the last 5 sensor or ground faults
* Last load date and time as well as Intellitrol SN and firmware rev are recorded into the SuperTIM.
* Unload mode was added for unload Intellitrols.
* The ability to not permit based on certificate expirations.
* The ability to not permit based on compartment count mismatch.
* Added Modbus error code 0x19 to indicate an attempt to write a read only register.

On startup the firmware revision will be displayed using the compartment and VIP leds. After the leds on the front panel are cycled the firmware version will be displayed using the compartment leds to indicate the decimal value and the VIP leds to indicate what portion of the version is being displayed. For version 1.7.0 the system will light the VIP Authorized led to indicate the major is being displayed and compartment 1 led will be lit. It will then light the VIP Unauthorized led to indicate the minor rev and led 7. The VIP Standby will then light to indicate the revision is being displayed and no compartment leds will light up to display 0. Displaying the firmware revision adds approximately 10 seconds to the Intellitrol start up time.

In previous revisions of firmware if a 2-wire sensor that has been dry either goes wet, open or short it was reported as a wet sensor. The system now checks and reports the correct status.

Dome out logging into the event log is now enabled on startup.

The default dead man open time was changed from 3 seconds to 1 second.

Correctly displays version number in all available locations. Startup LED’s, event log, and ASCII terminal.

Created maintenance error event in event log. This currently indicates unexpected resistance in 5-wire diagnostic line.

Updated function to calculate number of connected probes to be more accurate in worst case situations.

Created modbus command 5B to calculate number of connected probes and return the amount.

Created modbus command 5C to toggle between old and new ADC table for calculating number of probes. Command will return 5C00 for old table and 5C01 for new table.

Created modbus command 5D to determine which ADC table is being used to calculate the number of probes. Command will return 5D00 for old table and 5D01 for new table.

Green permit bar will flash for one second when bypass key is successfully added. This restores functionality as described in current manual.

**Code Changes**

Com\_two.c

Correct reporting of 2 wire probe state

void active\_two\_wire( PROBE\_TRY\_STATE test\_state)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Line 455  if (wet\_pass\_count > N\_CYCLES\_10)  {  check\_shorts\_opens(); /\* This will reset probes\_state and tank\_state if open or short \*/ | if (wet\_pass\_count > N\_CYCLES\_10)  {  if(dry\_once == FALSE)  {  check\_shorts\_opens(); /\* This will reset probes\_state and tank\_state if open or short \*/  } |

This change will cause check\_shorts\_opens to be called on a non-pulsing sensor regardless of whether it’s been dry or not.

comdat.c

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 322  unsigned int S\_TIM\_code; /\* Super TIM supported TRUE/FALSE \*/  unsigned int TIM\_fault\_logged = 0;  unsigned char load\_history\_ptr;  unsigned char cert\_ds\_fails=0;  unsigned int bad\_compartment\_count = 0; | Starting line 322  unsigned int S\_TIM\_code; /\* Super TIM supported TRUE/FALSE \*/ |

This change declares the global variables used for SuperTIM support

comdat.h

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 169  #define ENA\_INTELLITROL2 0x20  #define ENA\_UNLOAD\_TERM 0x40  #define ENA\_CPT\_COUNT 0x80 | Starting line 169  #define ENA\_INTELLITROL2 0x20  #define ENA\_SPARE\_3 0x40  #define ENA\_SPARE\_4 0x80 |
| Starting line 206  #define GROUND\_BYPASS 0x02 /\* missing ground bolt bypass \*/  #define CPT\_COUNT\_BYPASS 0x04 | Starting line 206  #define GROUND\_BYPASS 0x02 /\* missing ground bolt bypass \*/ |
| Starting line 245  #define BVF\_DONE 0x100 /\* Already scanned the EEPROM \*/  #define BVF\_UNLOAD\_EXP 0x200  #define BVF\_CPT\_COUNT 0x400 | Starting line 244  #define BVF\_DONE 0x100 /\* Already scanned the EEPROM \*/ |
| Starting line 483  extern unsigned int S\_TIM\_code; /\* Super TIM supported TRUE/FALSE \*/  extern unsigned int TIM\_fault\_logged;  extern unsigned char load\_history\_ptr;  extern unsigned char cert\_ds\_fails;  extern unsigned int bad\_compartment\_count; | Starting line 480  extern unsigned int S\_TIM\_code; /\* Super TIM supported TRUE/FALSE \*/ |

The first change defines the bits in enable\_soft that enable the unload terminal and compartment count features.

The second change defines the bit in bylevel that indicates a compartment count mismatch is what was bypassed.

The third change defines the bits in badvipflag that indicate the VIP is not authorizing due to unload time expires and compartment count mismatch.

The fourth change declares the global variables used for SuperTIM support.

dallas.c

int read\_TIM\_compartment\_info()

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 1413  if (tim\_block\_read(mem\_ptr, TABLE\_VALID\_ADDR, TABLE\_VALID\_SIZE) != MB\_OK)  {  return FAILED;  }  if ((mem\_ptr[0] != 0x55) || (mem\_ptr[1] != 0xAA))  {  xprintf(142, DUMMY);  return GOOD;  }  if (tim\_block\_read(mem\_ptr, INTELLICHECK\_TYPE\_ADDR, INTELLICHECK\_TYPE\_SIZE) != MB\_OK)  {  return FAILED;  }  if (mem\_ptr[0])  {  printf("\n\r Truck has an IntelliCheck\n\r");  number\_of\_Probes = 0xAA;  return GOOD;  }  if (tim\_block\_read(Truck\_TIM\_Configuration, NUMBER\_OF\_COMPARTMENTS\_ADDR, 1) != MB\_OK)  {  return FAILED;  }  number\_of\_Compartments = (unsigned int)Truck\_TIM\_Configuration[0]; /\* number of compartments stored in the TIM \*/ | Starting line 1405  if (tim\_block\_read(mem\_ptr, VALID\_ENTRIES\_ADR, VALID\_ENTRIES\_SIZE) != MB\_OK)  {  return FAILED;  }  if ((mem\_ptr[0] != 0x55) || (mem\_ptr[1] != 0xAA))  {  xprintf(142, DUMMY);  return GOOD;  }  if (tim\_block\_read(mem\_ptr, INTELLICHECK\_P\_ADR, INTELLICHECK\_P\_SIZE) != MB\_OK)  {  return FAILED;  }  if (mem\_ptr[0])  {  printf("\n\r Truck has an IntelliCheck\n\r");  number\_of\_Probes = 0xAA;  return GOOD;  }  temp\_word = NUMBER\_COMPARTMENTS\_SIZE + COMPARTMENT\_UNIT\_SIZE + (COMPARTMENT\_1\_SIZE \* 4);  if (tim\_block\_read(Truck\_TIM\_Configuration, NUMBER\_COMPARTMENTS\_ADR, temp\_word) != MB\_OK)  {  return FAILED;  }  number\_of\_Compartments = Truck\_TIM\_Configuration[0]; /\* number of compartments stored in the TIM \*/ |

void read\_TIM\_Go\_NoGo\_info()

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 1518  if (tim\_block\_read(mem\_ptr, TABLE\_VALID\_ADDR, TABLE\_VALID\_SIZE) != MB\_OK)  {  return;  } | Starting line 1503  if (tim\_block\_read(mem\_ptr, VALID\_ENTRIES\_ADR, VALID\_ENTRIES\_SIZE) != MB\_OK)  {  return;  } |
| Starting line 1549  if (tim\_block\_read((unsigned char\*)&temp\_byte, SCULLY\_SENSORS\_ADDR, SCULLY\_SENSORS\_SIZE) != MB\_OK)  {  return;  } | Starting line 1533  if (tim\_block\_read((unsigned char\*)&temp\_byte, SCULLY\_EQUIPMENT\_ADR, SCULLY\_EQUIPMENT\_SIZE) != MB\_OK)  {  return;  } |

All the changes in dallas.c are for remapping SuperTIM parameter locations

dumfile.c

void report\_tank\_state(void)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 234  else if (!((badvipflag & 0xFEFF) == 0))    {  t3 = LITE; /\* And RED for Failure \*/  t1 = DARK;  t2 = DARK;  } | Starting line 222  else if (!((badvipflag & 0xFF) == 0))  {  t3 = LITE; /\* And RED for Failure \*/  t1 = DARK;  t2 = DARK;  } |
| Starting line 261  else if (!((badvipflag & 0xFEFF) == 0)) /\* Anything else wrong?\*/  {  t3 = LITE; /\* Yes, just plain unauthorized truck \*/  t1 = DARK;  t2 = DARK;  } | Starting line 247  else if (!((badvipflag & 0xFF) == 0)) /\* Anything else wrong?\*/  {  t3 = LITE; /\* Yes, just plain unauthorized truck \*/  t1 = DARK;  t2 = DARK;  } |

These changes were made because we are now using bits in the upper byte of badvipflag to indicate the reason that the VIP did not authorize.

esquared.c

char eeUpdateSys(void)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 178  SysParm.DM\_Warn\_Start = DM\_WARN; /\* Active Deadman Warning time \*/  SysParm.Cert\_Expiration\_Mask = 0x00; /\* Active Deadman Warning time \*/  SysParm.Unload\_Max\_Time\_min = 240; /\* Active Deadman Warning time \*/  return nvSysParmUpdate(); /\* Write SysParm to EEPROM \*/ | Starting line 178  SysParm.DM\_Warn\_Start = DM\_WARN; /\* Active Deadman Warning time \*/  return nvSysParmUpdate(); /\* Write SysParm to EEPROM \*/ |

This change sets the default values for certificate expiration enable and the maximum unload time.

main.c

void main(void)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 588  diagnostics(DIA\_CHK\_INIT); /\* Run Diagnostics/Calibration for following parameters: \*/  /\* Kernel and Flash(Shell) CRC, Dallas Clock, \*/  /\* Reference, open voltages and 6/8 compartment \*/  /\* jumpers, raw, bias, noise voltages, 10/20 voltage \*/  /\* Ground R/Diode, LED Panel, Enable Jumpers \*/    show\_revision();    MBLINK1 = 1;  MBLINK2 = 0;  MBLINK3 = 1; | Starting line 588  diagnostics(DIA\_CHK\_INIT); /\* Run Diagnostics/Calibration for following parameters: \*/  /\* Kernel and Flash(Shell) CRC, Dallas Clock, \*/  /\* Reference, open voltages and 6/8 compartment \*/  /\* jumpers, raw, bias, noise voltages, 10/20 voltage \*/  /\* Ground R/Diode, LED Panel, Enable Jumpers \*/  MBLINK1 = 1;  MBLINK2 = 0;  MBLINK3 = 1; |

This change calls the function that displays the version number.

modcmd.c

static MODBSTS write\_tim(unsigned char tim\_type)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Lines 1925 to 2038  Commented out the routine write\_tim as it is all handled through write\_tim\_block |  |

static MODBSTS mbcRdTrBuilderInfo(void)

Lines 2060 to 2824, Mapped all the tim parameters for the 0x53 writeparameter command

static MODBSTS mbcRdTrBuilderInfo(void)

Lines 2060 to 2824, Mapped all the tim parameters for the 0x53 read parameter command

static MODBSTS mbcWrBuilderInfo(void)

Lines 2903 to 3117, Mapped all thetim parameters for the 0x54 write parameter command

MODBSTS modbus\_decode

(

unsigned char ilen, /\* Input ModBus message length (no CRC) \*/

unsigned char \*icmd, /\* Input ModBus message pointer \*/

unsigned char \*olen, /\* Output ModBus message length (no CRC) \*/

unsigned char \*orsp /\* Output (response) ModBus message pointer \*/

)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 2.38 |
| Starting line 3531  save\_iec0 = IEC0;  IEC0 = 0; /\* Disable heart beat and DMA interrupt \*/  sts = readTIMarea(0x080, 0x0FF);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 9/11/2008 10:35AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* restore interrupts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  IEC0 = save\_iec0; /\* Re-enable Heart Beat and DMA interrupts \*/  break;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 6/22/2009 8:17AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Write into the Scully reserve area  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  case WRITE\_THIRD\_PARTY: /\* 0x56 -- \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 9/11/2008 10:33AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Disable interrupts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  save\_iec0 = IEC0;  IEC0 = 0; /\* Disable heart beat and DMA interrupt \*/  sts = writeTIMarea(0x080, 0x0FF); | Starting line 2977  save\_iec0 = IEC0;  IEC0 = 0; /\* Disable heart beat and DMA interrupt \*/  sts = readTIMarea(0xC00, 0x1FFF);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 9/11/2008 10:35AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* restore interrupts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  IEC0 = save\_iec0; /\* Re-enable Heart Beat and DMA interrupts \*/  break;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 6/22/2009 8:17AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Write into the Scully reserve area  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  case WRITE\_THIRD\_PARTY: /\* 0x56 -- \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 9/11/2008 10:33AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Disable interrupts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  save\_iec0 = IEC0;  IEC0 = 0; /\* Disable heart beat and DMA interrupt \*/  sts = writeTIMarea(0xC00, 0x1FFF); |

Changed ranges to enable the read and write to customer locations on the tim.

modfrc.c

MODBSTS mbxForce

(

unsigned int fbit, /\* "Bit" or function code \*/

unsigned int fval /\* Value/argument for bit/function code \*/

)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 2.38 |
| Starting line 429  if ((badvipflag & 0xFEFF)) /\* VIP/Truck-ID bypass permissable? \*/  {  if ((fval != MODBITON) /\* Turning it on? \*/ | Starting line 428  if ((badvipflag & 0xFF)) /\* VIP/Truck-ID bypass permissable? \*/  {  if ((fval != MODBITON) /\* Turning it on? \*/ |

This change was made because we are now using bits in the upper byte of badvipflag to indicate the reason that the VIP did not authorize.

modreg.c

static unsigned mbrNonPermitReg (void)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 98  if (((badvipflag & 0xFEFF)) /\* Truck ID? \*/  && (!(bylevel & VIP\_BYPASS))) /\* and not already bypassed? \*/  { /\* Yes \*/  hval |= VIP\_BYPASS; /\* Non-Permit due to Unauthorized \*/  } | Starting line 98  if (((badvipflag & 0xFF)) /\* Truck ID? \*/  && (!(bylevel & VIP\_BYPASS))) /\* and not already bypassed? \*/  { /\* Yes \*/  hval |= VIP\_BYPASS; /\* Non-Permit due to Unauthorized \*/  } |

This change was made because we are now using bits in the upper byte of badvipflag to indicate the reason that the VIP did not authorize.

MODBSTS mbrRdReg

(

unsigned int regno, /\* 16-bit ModBus "Register" number \*/

unsigned int \*value /\* Pointer to return 16-bit register data \*/

)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 678  case 0x8: /\* 068 -- VIP status (inc DateStamp) \*/  hval = badvipflag;  break;  case 0x9: /\* 069 -- Service "A" flags \*/  hval = iambroke; /\* Voltage error details, etc. \*/  break;  case 0xA: /\* 06A -- Service "B" flags \*/  hval = iamsuffering; /\* Hard-wired relays etc. \*/  break;  case 0xB: /\* 06D -- bad compartment / probe count \*/  hval = (unsigned)(unsigned char)bad\_compartment\_count;  break;  case 0xC: /\* 06B -- VIP status (inc DateStamp) \*/  hval = (unsigned)(((unsigned)(unsigned char)badvipdscode << 8)  | ((unsigned)(unsigned char)badvipflag & 0xFF));  break;  case 0xD: /\* 06D -- Ground status \*/  hval = (unsigned)(unsigned char)badgndflag;  break;  case 0xE: /\* 06C -- VIP status (inc DateStamp) \*/  hval = badvipflag;  break;  case 0xF: /\* 06C -- VIP status (inc DateStamp) \*/  hval = cert\_ds\_fails;  break; | Starting line 672  case 0x9: /\* 069 -- Service "A" flags \*/  hval = iambroke; /\* Voltage error details, etc. \*/  break;  case 0xA: /\* 06A -- Service "B" flags \*/  hval = iamsuffering; /\* Hard-wired relays etc. \*/  break;  case 0xC: /\* 06C -- VIP status (inc DateStamp) \*/  hval = (unsigned)(((unsigned)(unsigned char)badvipdscode << 8)  | ((unsigned)(unsigned char)badvipflag & 0xFF));  break;  case 0xD: /\* 06D -- Ground status \*/  hval = (unsigned)(unsigned char)badgndflag;  break; |
| Starting line 793  case 0x03: /\* 83 -- Active Deadman Warning time \*/  hval = (SysParm.DM\_Warn\_Start >> 2);  break;  case 0x04: /\* 84 -- Software Feature Enable Unload Term \*/  if(SysParm.EnaSftFeatures & ENA\_UNLOAD\_TERM)  {  hval = 0xFF;  }  else  {  hval = 0;  }  break;    case 0x05: /\* 86 -- Supertim max unload time \*/  hval = (SysParm.Unload\_Max\_Time\_min);  break;  case 0x06: /\* 86 -- Supertim cert date enable mask \*/  hval = (SysParm.Cert\_Expiration\_Mask);  break;  case 0x07: /\* 87 -- Software Feature Enable compartment count check \*/  if(SysParm.EnaSftFeatures & ENA\_CPT\_COUNT)  {  hval = 0xFF;  }  else  {  hval = 0;  }  break;    default: /\* Others are an error \*/  return(MB\_EXC\_ILL\_ADDR);  break; | Starting line 771  case 0x03: /\* 83 -- Active Deadman Warning time \*/  hval = (SysParm.DM\_Warn\_Start >> 2);  break;  default: /\* Others are an error \*/  case 0x03: /\* 83 -- Active Deadman Warning time \*/  hval = (SysParm.DM\_Warn\_Start >> 2);  break;  default: /\* Others are an error \*/  hval = MB\_EXC\_ILL\_ADDR;  break; |
| Starting line 1218  case 0x1: /\* 121 - Stop logging dome out events \*/  hval = (unsigned)disable\_domeout\_logging; /\* \*/  break;    case 0x2: /\* 122 - \*/  hval = (unsigned)TIM\_size; /\* \*/  break;    case 0x3: /\* 123 - \*/  hval = (unsigned)S\_TIM\_code; /\* \*/  break;    default: /\* Others \*/  hval = MB\_EXC\_ILL\_ADDR;  break; | Starting line 1166  case 0x1: /\* 121 - Stop logging dome out events \*/  hval = (unsigned)disable\_domeout\_logging; /\* \*/  break;  default: /\* Others \*/  hval = MB\_EXC\_ILL\_ADDR;  break; |

These changes were made for new modbus registers. We also added lines 1255 to 1470 for the holding registers for all the superTIM parameters.

MODBSTS mbrWrReg

(

unsigned int regno, /\* 16-bit ModBus "Register" number \*/

unsigned int \*value /\* 16-bit "Register" data \*/

)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 1754  case 0x83: /\* 83 -- Active Deadman Warning time \*/  if ((\*value >= 10) && (\*value <= 60))  {  SysParm.DM\_Warn\_Start = (\*value << 2);  (void)nvSysParmUpdate();  }  else  {  return (MB\_EXC\_ILL\_DATA); /\* reject bad values \*/  }  break;    case 0x84: /\* 84 -- Software Feature Enable Code \*/  /\* Enable/Disable Unload Terminal \*/  if (\*value == 0) /\* Disable requested \*/  {  SysParm.EnaSftFeatures =  (unsigned char) (SysParm.EnaSftFeatures & ~ENA\_UNLOAD\_TERM);  }  if (\*value > 0) /\* Enable requested \*/  {  SysParm.EnaSftFeatures =  (unsigned char) (SysParm.EnaSftFeatures | ENA\_UNLOAD\_TERM);  }  // (void)nvSysParmUpdate(); /\* Force EEPROM update \*/  modNVflag++; /\* Request EEPROM update \*/  break;  case 0x85: /\* 85 -- Unload max time \*/  SysParm.Unload\_Max\_Time\_min = (unsigned char)(\*value);  (void)nvSysParmUpdate();  break;    case 0x86: /\* 86 -- Cert DS enable mask \*/  if ((\*value >= 0) && (\*value < 32))  {  SysParm.Cert\_Expiration\_Mask = (unsigned char)(\*value);  (void)nvSysParmUpdate();  }  else  {  return (MB\_EXC\_ILL\_DATA); /\* reject bad values \*/  }  break;    case 0x87: /\* 84 -- Software Feature Enable Code \*/  /\* Enable/Disable Unload Terminal \*/  if (\*value == 0) /\* Disable requested \*/  {  SysParm.EnaSftFeatures =  (unsigned char) (SysParm.EnaSftFeatures & ~ENA\_CPT\_COUNT);  }  if (\*value > 0) /\* Enable requested \*/  {  SysParm.EnaSftFeatures =  (unsigned char) (SysParm.EnaSftFeatures | ENA\_CPT\_COUNT);  }  // (void)nvSysParmUpdate(); /\* Force EEPROM update \*/  modNVflag++; /\* Request EEPROM update \*/  break;  case 0x100: /\* 100 -- High-order system Time-Of-Day \*/  /\* Wait for second half to do the actual write as an atomic operation.  Just "stash" the high half for now; assume low half follows  immediately after. \*/  wtmp = \*value; /\* Just hold on to high order half... \*/  break; | Starting line 1473  case 0x83: /\* 83 -- Active Deadman Warning time \*/  if ((\*value >= 10) && (\*value <= 60))  {  SysParm.DM\_Warn\_Start = (\*value << 2);  (void)nvSysParmUpdate();  }  else  {  return (MB\_EXC\_ILL\_DATA); /\* reject bad values \*/  }  break;    case 0x100: /\* 100 -- High-order system Time-Of-Day \*/  /\* Wait for second half to do the actual write as an atomic operation.  Just "stash" the high half for now; assume low half follows  immediately after. \*/  wtmp = \*value; /\* Just hold on to high order half... \*/  break; |

These changes were made for new modbus registers. We also added lines 1868 to 2387 for the holding registers for all the superTIM parameters.

permit.c

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 383  if ((badvipflag & 0xFEFF)) /\* Unauthorized truck ID ? \*/  { /\* Yes \*/  if ((!(bylevel & VIP\_BYPASS)) /\* VIP (truck ID) bypass set? \*/  && (status)) /\* and not locked out? \*/  {  status = NOTNOW; /\* No -- Bypassable non-permissive \*/  }  } | Starting line 380  if ((badvipflag & 0xFF)) /\* Unauthorized truck ID ? \*/  { /\* Yes \*/  if ((!(bylevel & VIP\_BYPASS)) /\* VIP (truck ID) bypass set? \*/  && (status)) /\* and not locked out? \*/  {  status = NOTNOW; /\* No -- Bypassable non-permissive \*/  }  } |

This change was made because we are now using bits in the upper byte of badvipflag to indicate the reason that the VIP did not authorize.

pod.c

Lines 347 to 444

Added the function void show\_revision(void)

proto.h

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 288  char flash\_panel (void);  void show\_revision(void);  char check\_ref\_volt (void); | Starting line 288  char flash\_panel (void);  char check\_ref\_volt (void); |
| Starting line 388  unsigned char fetch\_serial\_number(unsigned char tim\_type, unsigned char \*tim\_number);  void TIM\_log\_fault(unsigned int fault\_val);  void TIM\_log\_info(void);  char superTIM\_ds\_validate(void);  char check\_unload\_time(void);  void check\_compartment\_count(void); | Starting line 387  unsigned char fetch\_serial\_number(unsigned char tim\_type, unsigned char \*tim\_number); |

These changes are to define the function prototypes for the added global functions.

stdsym.h

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 271  # unsigned int DM\_Warn\_Start; /\* 0x02022C Active Deadman Warning time \*/  unsigned char Cert\_Expiration\_Mask; /\* 0x02022E supertim cert expiration mask \*/  unsigned int Unload\_Max\_Time\_min; /\* 0x02022F Allowable time for unload since load \*/  unsigned char free[0x0D]; /\* //Fogbugz 131 0x020231 Round up to 64 bytes total (23E - current) \*/  unsigned int CRC; /\* 0x02023E G.P. Parameter block CRC \*/ | Starting line 271  unsigned int DM\_Warn\_Start; /\* 0x02022C Active Deadman Warning time \*/  unsigned char free[0x10]; /\* //Fogbugz 131 0x02022E Round up to 64 bytes total (23E - current) \*/  unsigned int CRC; /\* 0x02023E G.P. Parameter block CRC \*/ |
| Starting line 278  #define DM\_OPEN (1\*4) /\* Active Deadman Max open time \*/ | Starting line 276  #define DM\_OPEN (3\*4) /\* Active Deadman Max open time \*/ |

The first change defines the certificate mask and unload time into the SysParmNV structure.

The second change changes the default dead man open time to 1 second instead of 3 seconds.

tim\_util.c

unsigned char fetch\_serial\_number(unsigned char tim\_type, unsigned char \*tim\_number)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 474  case TEST\_TIM :  valid\_address = 0x404;  tim\_address = 0x405;  break;  case ALT\_TIM :  valid\_address = ALT\_TIM\_ID\_VALID\_ADDR;  tim\_address = ALT\_TIM\_ID\_ADDR;  break;  default:  return MB\_EXC\_TIM\_CMD\_ERR; | Starting line 463  case TEST\_TIM :  valid\_address = VALID\_TRUCKTEST\_TIM\_ADR;  tim\_address = TRUCK\_TEST\_TIM\_NUMBER\_ADR;  break;  case ALT\_TIM :  valid\_address = VALID\_ALT\_TIM\_ADR;  tim\_address = ALT\_TIM\_NUMBER\_ADR;  break;  default:  return MB\_EXC\_TIM\_CMD\_ERR; |
| Starting line 497  if ((sts = (unsigned char)tim\_block\_read(tim\_number, TABLE\_VALID\_ADDR, TABLE\_VALID\_SIZE)) != MB\_OK)  {  return (sts);  } | Starting line 486  if ((sts = (unsigned char)tim\_block\_read(tim\_number, VALID\_ENTRIES\_ADR, VALID\_ENTRIES\_SIZE)) != MB\_OK)  {  return (sts);  } |

These two changes are for the remapped tim parameters

In lines 525 to 814 we added the following functions:

* void TIM\_log\_fault(unsigned int fault\_val)
* void TIM\_log\_info(void)
* char superTIM\_ds\_validate(void)
* char check\_unload\_time(void)
* void check\_compartment\_count(void)

tim\_util.h

There are too many changes to list. All changes were to map and size the new tim parameters.

trukstat.c

static void truck\_idle(void)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Start line 260  print\_once\_msg &= ~UN\_AUTH; /\* Clear all entries in idle loop \*/  TIM\_fault\_logged = 0;  TIM\_info\_logged = 0;  bad\_compartment\_count = 0;  StatusB &= ~STSB\_TRUCK; /\* Clear truck valid \*/ | Start line 258  print\_once\_msg &= ~UN\_AUTH; /\* Clear all entries in idle loop \*/  StatusB &= ~STSB\_TRUCK; /\* Clear truck valid \*/ |
| Start line 712  if ((bypass\_state = bypass\_operation()) != 0)  {  if( (StatusA & STSA\_FAULT) ) // if in system fault do a reset  {  secReset = 1; /\* Force immediate board-level RESET \*/  // for(;;); // let the watchdog reset us  }  if (bypass\_state == 2) /\* good bypass key \*/  {  // printf("\n\r\*\*\*\*\* 07 \*\*\*\*\*\*\n\r");  status = TRUE; | Start line 708  if ((bypass\_state = bypass\_operation()) != 0)  {  if (bypass\_state == 2) /\* good bypass key \*/  {  // printf("\n\r\*\*\*\*\* 07 \*\*\*\*\*\*\n\r");  status = TRUE; |

The first change resets the flags to determine what active super tim functions to call.

The second change does an immediate reset on the hardware when a bypass key is scanned in a system error state.

static void truck\_active(void)

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Start line 1291  StatusO &= ~0x7; /\* Clear VIP output status \*/  ledstate[VIP\_IDLE] = DARK;  ledstate[VIP\_AUTH] = LITE;  ledstate[VIP\_UNAUTH] = DARK;  StatusO |= STSO\_AUTHORIZED;  }  }  }  }  }  //SME  if( TIM\_size == DS28EC20\_SIZE) // Super TIM  {  if( TIM\_info\_logged == 0 )  {  TIM\_log\_info();  TIM\_info\_logged = 1;  }    if( TIM\_fault\_logged == 0 )  {  if( (StatusA & STSA\_PERMIT) == 0)  {  if((badgndflag & GND\_PROBLEMS) || (tank\_state == T\_WET) )  {  fault\_num = 0;  if( badgndflag & GND\_PROBLEMS )  {  fault\_num = 25;  }  else  {  for(i = 0; i < 16; i++)  {  if((probes\_state[i] != P\_UNKNOWN) && (probes\_state[i] != P\_DRY))  {  fault\_num = i + 1;  break;  }  }  }  if( fault\_num != 0 )  {  TIM\_log\_fault(fault\_num);  TIM\_fault\_logged = 1;  }  }  }  }  }    TIM\_timer = (read\_time() + 330); /\* Reset the timer for 1/3 second \*/ | Start line 1280  StatusO &= ~0x7; /\* Clear VIP output status \*/  ledstate[VIP\_IDLE] = DARK;  ledstate[VIP\_AUTH] = LITE;  ledstate[VIP\_UNAUTH] = DARK;  StatusO |= STSO\_AUTHORIZED;  }  }  }  }  }  TIM\_timer = (read\_time() + 330); /\* Reset the timer for 1/3 second \*/ |
| Start line 1383  case OPTIC\_TWO: /\* If here, we have a 2 wire optic truck \*/  xprintf( 45, DUMMY ); /\* as determined in optic\_2\_setup \*/  (void)read\_bypass(TRUE); /\* Read any bypass chip \*/  if ((SysParm.Ena\_Debug\_Func\_1 == 0x32) || (SysParm.Ena\_Debug\_Func\_2 == 0x32)  || (SysParm.Ena\_Debug\_Func\_3 == 0x32) || (SysParm.Ena\_Debug\_Func\_4 == 0x32))  {  debug\_pulse(0x32);  }  check\_compartment\_count();  // (void)compartment\_check(); /\* Validate sensor count with SuperTIM \*/  active\_two\_wire(OPTIC2);  break; | Start line 1314  case OPTIC\_TWO: /\* If here, we have a 2 wire optic truck \*/  xprintf( 45, DUMMY ); /\* as determined in optic\_2\_setup \*/  (void)read\_bypass(TRUE); /\* Read any bypass chip \*/  if ((SysParm.Ena\_Debug\_Func\_1 == 0x32) || (SysParm.Ena\_Debug\_Func\_2 == 0x32)  || (SysParm.Ena\_Debug\_Func\_3 == 0x32) || (SysParm.Ena\_Debug\_Func\_4 == 0x32))  {  debug\_pulse(0x32);  }  // (void)compartment\_check(); /\* Validate sensor count with SuperTIM \*/  active\_two\_wire(OPTIC2);  break; |
| Start line 1399  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5/11/2009 6:52AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* The compartment count can be faked out by a wet probe. So test for  \* return pulse. If the pulse does not return the last probe is wet. If  \* it does return all probes are functioning and valid compartment count  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  check\_compartment\_count();  // (void)compartment\_check();  active\_5wire(); | Starting line 1339  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5/11/2009 6:52AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* The compartment count can be faked out by a wet probe. So test for  \* return pulse. If the pulse does not return the last probe is wet. If  \* it does return all probes are functioning and valid compartment count  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // (void)compartment\_check();  active\_5wire(); |
| Start line 2129  if (S\_TIM\_code) /\* is the TIM a Super TIM? \*/  {  read\_TIM\_Go\_NoGo\_info();  }    if( (SysParm.EnaSftFeatures & ENA\_UNLOAD\_TERM) && (SysParm.Unload\_Max\_Time\_min != 0) )  {  sts = check\_unload\_time();  if( sts )  {  StatusA &= ~STSA\_TRK\_VALID; /\* Truck not authorized! \*/  // badvipflag |= BVF\_DSNOAUTH; /\* No Dallas/IO problems (0x08) \*/  badvipflag |= BVF\_UNLOAD\_EXP; /\* No Dallas/IO problems (0x08) \*/  return; /\* we can't do much more here \*/    }  }    if( SysParm.Cert\_Expiration\_Mask != 0)  {  sts = superTIM\_ds\_validate();  if (sts == (int)DSEXPIRED ) /\* or an invalid terminal \*/  {  /\* DateStamp authoritatively rejects this truck, so his  only hope is a VIP-Bypass operation \*/  badvipflag &= ~BVF\_DSERROR; /\* No Dallas/IO problems \*/  badvipflag |= BVF\_DSNOAUTH; /\* DateStamp says No! \*/  badvipdscode = sts; /\* Detailed rejection code \*/  return;  }  else  {  if (sts) /\* If can't access TIM/file \*/  {  /\* May be "random" I/O error, try again later and maybe  it'll work the next time round \*/  badvipflag |= BVF\_DSERROR; /\* Errors accessing DateStamp \*/  badvipdscode = sts; /\* Detailed error code \*/  val\_state = 0; /\* Enable retry later \*/  return;  }  else  {  badvipflag &= ~BVF\_DSERROR; /\* No error now \*/  badvipdscode = 0; /\* This truck authorized \*/  }  }    }    sts = nvTrkFind (&truck\_SN[0], (word \*)&index); /\* Check local list of good guys \*/ | Start line 2066  if (S\_TIM\_code) /\* is the TIM a Super TIM? \*/  {  read\_TIM\_Go\_NoGo\_info();  }  sts = nvTrkFind (&truck\_SN[0], (word \*)&index); /\* Check local list of good guys \*/ |

The first change handles logging the connection data and fault to the super TIM.

The second and third changes we for calling the compartment count check.

The fourth change calls the unload time check and certificate check if they are enabled

modbus.h

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 60  #define USE\_UPDATED\_ADC\_TABLE 0x5C |  |

Added modbus command 5C to switch between old calc\_tank adc table and new updated table for probe counting.

volts.h

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 224  extern unsigned long lowVolt; |  |

Global variable that stores the lowest adc voltage from calc\_tank until intellitrol is in a permit state.

stsbits.h

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 105  /\* Channel 5 diag line resistance is higher than expected - calc\_tank()\*/  #define CH5\_HIGH\_RESISTANCE 0x0200 |  |

Variable used for storing calc\_tank error in event log.

version.h

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 194  #define MAJVER 01  #define MINVER 07  #define EDTVER 00 | Starting line 194  #define MAJVER 0x01  #define MINVER 0x06  #define EDTVER 36 |

Updated version to 1.7.0 and fixed issue with displaying incorrect version when converting from hex to decimal.

proto.h

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 244  char nvSysDia5Update (unsigned int updatedADCTable); | Starting line 244  char nvSysDia5Update (void); |
| Starting line 316  void logmaintenanceerr(void); |  |

Modified nvSysDia5Update function to allow for selection of old adc table or new table. Created logmaintenanceerr function to push maintenance errors to the event log.

evlog.h

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 161  char future[16]; /\* Reserved for future \*/ | Starting line 161  char future[18]; /\* Reserved for future \*/ |
| Starting line 224  #define EVIMAINTENANCE 0x08  typedef struct  {  unsigned int ch5\_high\_resistance; /\* Channel 5 resistance is higher than expected calc\_tank() \*/  char future[20]; /\* Reserved for future \*/  } EVI\_MAINTENANCE; | Starting line 224  #define EEFMT 0x08  typedef struct  {  char future[22]; /\* Reserved for future \*/  } EEFMT\_INFO; |
| Starting line 249  char future[2]; /\* Reserved for future \*/ |  |
| Starting line 263  char future; /\* Reserved for future \*/ |  |

Created EVI\_MAINTENANCE for displaying maintenance errors to the event log. Added or modified future arrays to correctly size each structure.

stdsym.h

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 309  typedef struct  {  unsigned Reference; /\* "6.759" volt reference level \*/  unsigned PNOffset; /\* Transistor PN-junction bias/offset \*/  unsigned WetVolts[16]; /\* 16-probe "wet" level \*/  unsigned int updatedADCTable; /\* Switch to use updated ADC table for probe counting in calc\_tank, 1 = new table, 0 = old table \*/  /\* DateStamp + SysDia5 = 64 bytes total! \*/  unsigned CRC; /\* SysDia5 block CRC \*/  } SysDia5NV; | Starting line 304  typedef struct  {  unsigned Reference; /\* "6.759" volt reference level \*/  unsigned PNOffset; /\* Transistor PN-junction bias/offset \*/  unsigned WetVolts[16]; /\* 16-probe "wet" level \*/  char free[2]; /\* Round up to 40 bytes total \*/  /\* DateStamp + SysDia5 = 64 bytes total! \*/  unsigned CRC; /\* SysDia5 block CRC \*/  } SysDia5NV; |

Replaced free array with updatedADCTable. updatedADCTable is used to select the old ADC table or the new table to be used in calc\_tank.

permit.c

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 303  lowVolt = 9999; // Used for calc\_tank() probe compartment count |  |

Reset lowVolt to 9999 when in dry permit state.

specops.c

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 342  ledstate[NONPERMIT] = DARK;  ledstate[PERMIT] = LITE;  service\_wait(16); |  |

Flash the green permit led when bypass key is successfully added.

modcmd.c

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 3727  case USE\_UPDATED\_ADC\_TABLE: /\* 0x5C -- Use updated ADC table for probe counting, 1 = new table, 0 = old table \*/  if(pSysDia5->updatedADCTable == 0) {  nvSysDia5Update(1); // Update eeprom block  sts = mbcPutByte (0x01); // Return 01  }    else {  nvSysDia5Update(0); // Update eeprom block  sts = mbcPutByte (0x00); // Return 00  }  break; |  |

Added modbus command 5C to swap to other adc table for use in calc\_tank. Return 5C00 for old table and 5C01 for new table.

main.c

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 1398  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* logmaintenanceerr() -- Create an Event Log entry for maintenance errors  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  void logmaintenanceerr()  {  EVI\_MAINTENANCE maintenance;  maintenance.ch5\_high\_resistance = 1; /\* Channel 5 resistance is higher than expected calc\_tank() \*/    nvLogRepeat (EVIMAINTENANCE, 1, (char \*)&maintenance, (unsigned long)(4 \* 60 \* 60));  } |  |

Created function logmaintenanceerr to push a maintenance error to the event log.

optic5.c

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 319  void optic\_5\_setup( void )  {  unsigned int i;  int probe\_flag = FALSE;  lowVolt = 9999;  // last\_routine = 0x38;  ledstate[OPTIC\_OUT] = DARK; /\* In case where updater has them flashing \*/  ledstate[OPTIC\_IN] = DARK;  set\_main\_state (ACTIVE);  truck\_state = OPTIC\_FIVE;  acquire\_state = IDLE\_I;  optic5\_state = NO5\_TEST;  probe\_try\_state = OPTIC5;  jump\_time = 0; /\* JUMP\_START off always \*/  dry\_timer = 0; /\* dry probe operation only \*/  badgndflag |= GND\_INIT\_TRIAL;  set\_porte( OPTIC\_DRIVE ); /\* assure setup for 5 wire optic pulsing \*/  set\_mux( M\_PROBES ); /\* assure set mux to probes \*/  ops\_ADC( OFF ); /\* shut off ADC timer interrupt (T3) \*/  xprintf( 25, DUMMY );  number\_of\_Probes = 0;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 12/5/2008 7:17AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Test how many probes on this truck might take a few tries  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  for (i = 0; i < 5; i++)  {  DelayMS(30); /\* ensure minimum period \*/  if (try\_five\_wire() == TRUE)  {  probe\_flag = TRUE;  number\_of\_Probes = (unsigned int)(calc\_tank() - 1);  DelayMS(100); /\* wait maximum period \*/  service\_charge(); /\* Appease watchdog \*/  }  }  if (probe\_flag == TRUE) /\* Return pulse seen; dry vehicle \*/  {  if (number\_of\_Probes < 1 || number\_of\_Probes > 16)  {  number\_of\_Probes = 1; /\* Might be a IntelliCheck or no diag line \*/  }    display\_probe();  dry\_5W\_probes();  }  else  {  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 7/22/2011 8:16AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Probes are wet  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  number\_of\_Probes = (unsigned int)(calc\_tank());  }    lowVolt = 9999;  } /\* end of optic\_5\_setup \*/ | Starting line 319  void optic\_5\_setup( void )  {  unsigned int i;  int probe\_flag = FALSE;  // last\_routine = 0x38;  ledstate[OPTIC\_OUT] = DARK; /\* In case where updater has them flashing \*/  ledstate[OPTIC\_IN] = DARK;  set\_main\_state (ACTIVE);  truck\_state = OPTIC\_FIVE;  acquire\_state = IDLE\_I;  optic5\_state = NO5\_TEST;  probe\_try\_state = OPTIC5;  jump\_time = 0; /\* JUMP\_START off always \*/  dry\_timer = 0; /\* dry probe operation only \*/  badgndflag |= GND\_INIT\_TRIAL;  set\_porte( OPTIC\_DRIVE ); /\* assure setup for 5 wire optic pulsing \*/  set\_mux( M\_PROBES ); /\* assure set mux to probes \*/  ops\_ADC( OFF ); /\* shut off ADC timer interrupt (T3) \*/  xprintf( 25, DUMMY );  number\_of\_Probes = 0;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 12/5/2008 7:17AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Test how many probes on this truck might take a few tries  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  for ( i=0; i<10; i++)  {  DelayMS(30); /\* ensure minimum period \*/  if (try\_five\_wire() == TRUE)  {  probe\_flag = TRUE;  number\_of\_Probes = (unsigned int)((unsigned char)calc\_tank()-1);  DelayMS(100); /\* wait maximum period \*/  service\_charge(); /\* Appease watchdog \*/  if ((number\_of\_Probes != 0) &&  (number\_of\_Probes == (unsigned int)((unsigned char)calc\_tank()-1)))  {  break; /\* Break from the loop \*/  }  }  }  if ( probe\_flag == TRUE) /\* Return pulse seen; dry vehicle \*/  {  if (number\_of\_Probes == 0)  {  number\_of\_Probes = 1; /\* Might be a IntelliCheck\*/  }  else  {  if (number\_of\_Probes > 16)  {  number\_of\_Probes = 0x55; /\* Set as an invalid compartment count \*/  }  }  display\_probe();  dry\_5W\_probes();  } else  {  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 7/22/2011 8:16AM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Probes are wet  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  number\_of\_Probes = (unsigned int)((unsigned char)calc\_tank());  //FogBugz 108 unknown\_probes();  // report\_flag = 0;  }  // last\_routine = 0x38;  } /\* end of optic\_5\_setup \*/ |
| Starting line 755  #define TRIALS 10 /\* number or repeats to do the averaging \*/  unsigned long lowVolt = 9999;  unsigned int calc\_tank( void )  {  unsigned long ch5\_volt;  unsigned long ch5\_volt\_oldTable;  unsigned long index;  unsigned int tank\_number;  unsigned long voltList[17] = {6840, 6630, 6380, 6000, 5690, 5385, 5130, 4890, 4660, 4470, 4270, 4105, 3950, 3795, 3675, 3550, 3440};  char error\_found = 0;    StatusA &= ~CH5\_HIGH\_RESISTANCE;    //printf("Using table number %d\n", pSysDia5->updatedADCTable);    DelayMS(30); // ensure minimum period  optic\_5\_pulse(); // Pulse optic probe to get reading  tank\_number = 0;  ch5\_volt = 0;  CH\_TEST5 = 0; // Turn off Ch 5 (DIAG channel)  DIAGNOSTIC\_EN = 0; // Turn on precision DIAG voltage    if (read\_ADC() == FAILED)  {  printf("\n\r3: Trouble reading the Analog Port\n\r");  Init\_ADC();  return 18; // Since we can't read the voltage we call it a invalid probe  }  for (index = 0; index < TRIALS; index++ ) // Average TRIALS  {  if (read\_ADC() == FAILED)  {  printf("\n\r4: Trouble reading the Analog Port\n\r");  Init\_ADC();  return 18; // Since we can't read the voltage we call it a invalid probe  }  else  {  ch5\_volt += probe\_volt[4];  optic5\_table[index] = probe\_volt[4];  }  }    CH\_TEST5 = 1;  DIAGNOSTIC\_EN = 1;  ch5\_volt /= index; // 5-wire-optic diagnostic voltage    // Add offset only for old table  ch5\_volt\_oldTable = ch5\_volt;  ch5\_volt\_oldTable += (unsigned long)pSysDia5->PNOffset;  ch5\_volt\_oldTable \*= (unsigned long)ReferenceVolt;  ch5\_volt\_oldTable /= (unsigned long)1000;    ch5\_volt \*= (unsigned long)ReferenceVolt;  ch5\_volt /= (unsigned long)1000;    // New Table  if(pSysDia5->updatedADCTable == 1)  {  if(ch5\_volt < lowVolt)  {  lowVolt = ch5\_volt;  }    //printf("LOW VOLTAGE: %d\n", (int)lowVolt);  compare\_volts = lowVolt;    for (index = 0; index < 17; index++)  {  if (lowVolt <= voltList[index])  {  tank\_number++;  }  }    if(tank\_number > 1 && tank\_number < 16) {  if (lowVolt > ((((voltList[tank\_number - 1] - voltList[tank\_number]) \* 25) / 100) + voltList[tank\_number])  || (lowVolt >= voltList[tank\_number] && lowVolt < (voltList[tank\_number] + (unsigned long)5))) {  //printf("Please check sensor connection: %d\n", (int)((((voltList[tank\_number - 1] - voltList[tank\_number]) \* 25) / 100) + voltList[tank\_number]));  StatusA |= CH5\_HIGH\_RESISTANCE;  logmaintenanceerr();  }  }    //printf("TABLE VALUE %d: %d\n", tank\_number - 1, (int)voltList[tank\_number - 1]);  //printf("TABLE VALUE %d: %d\n\n", tank\_number, (int)voltList[tank\_number]);  }    // Old Table  else {  lowVolt = 9999;    //printf("CHANNEL 5 VOLTAGE: %d\n", (int)(ch5\_volt\_oldTable));  compare\_volts = ch5\_volt\_oldTable;    for (index = 0; index < 16; index++)  {  tank\_number++;    if (pSysDia5 != 0)  {  if (ch5\_volt\_oldTable > pSysDia5->WetVolts[index])  {  error\_found = 1;  break;  }  }  }  if (error\_found == 0)  {  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1/22/2010 2:52PM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* If no error found then it must be for sizing a truck that has 16  \* compartments. Because the result is subtracted by one we must add one to it  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  tank\_number++; /\* Because the result is subtracted by one we must add one to it \*/  }  }    return(tank\_number);  } | Starting line 760  #define TRIALS 8 /\* number or repeats to do the averaging \*/  unsigned int calc\_tank( void )  {  unsigned long ch5\_volt;  unsigned long index;  unsigned int tank\_number;  char error\_found = 0;  // last\_routine = 0x3C;  DelayMS(30); /\* ensure minimum period \*/  optic\_5\_pulse(); /\* Pulse optic probe to get reading \*/  tank\_number = 0;  ch5\_volt = 0;  CH\_TEST5 = 0; /\* Turn off Ch 5 (DIAG channel) \*/  DIAGNOSTIC\_EN = 0; /\* Turn on precision DIAG voltage \*/  if (read\_ADC() == FAILED)  {  printf("\n\r3: Trouble reading the Analog Port\n\r");  Init\_ADC();  return 18; /\* Since we can't read the voltage we call it a invalid probe \*/  }  for (index=0; index<TRIALS; index++ ) /\* Average TRIALS \*/  {  if (read\_ADC() == FAILED)  {  printf("\n\r4: Trouble reading the Analog Port\n\r");  Init\_ADC();  return 18; /\* Since we can't read the voltage we call it a invalid probe \*/  } else  {  ch5\_volt += probe\_volt[4];  optic5\_table[index] = probe\_volt[4];  }  }  CH\_TEST5 = 1;  DIAGNOSTIC\_EN = 1;  ch5\_volt /= index; /\* 5-wire-optic diagnostic voltage \*/  ch5\_volt += (unsigned long)pSysDia5->PNOffset; /\* Offset by switching transistor \*/  ch5\_volt \*= (unsigned long)ReferenceVolt; /\* Calibrated x1000 (check\_ref\_volt()) \*/  ch5\_volt /= (unsigned long)1000;  for (index=0; index<16; index++)  {  tank\_number++;  if ( pSysDia5 != 0)  {  if ((unsigned)ch5\_volt > pSysDia5->WetVolts[index])  {  error\_found = 1;  break;  }  }  }  if ( error\_found == 0)  {  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1/22/2010 2:52PM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* If no error found then it must be for sizing a truck that has 16  \* compartments. Because the result is subtracted by one we must add one to it  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  tank\_number++; /\* Because the result is subtracted by one we must add one to it \*/  }  return(tank\_number);  } /\* end of calc\_tank \*/ |

Rewrote calc\_tank to fix probe counting errors. Old function and ADC table only work with perfect conditions and do not take real life variables into account. Created new ADC table to better match the real world and will work with best case scenarios while improving performance in worst case situations. lowVolt was added to only using the lowest voltage seen instead of using the live voltage. Using the lowest voltage gives much more consistent results and we found that the voltage can only go so low in all situations, while the highest voltage is widely unpredictable in worst case situations. This update in conjunction with regularly cleaning the socket greatly increases the accuracy in worst case situations. Modbus command 5C was added to allow the user to switch back to the old calc\_tank function if desired. Modbus command 5B allows the user to run the calc\_tank function at any time and have the number of probes returned to them.

nvsystem.c

|  |  |
| --- | --- |
| Rev 1.7.0 | Rev 1.6.36 |
| Starting line 74  1, /\* Switch to use updated ADC table for probe counting in calc\_tank, 1 = new table, 0 = old table \*/ | Starting line 74  {0}, /\* Reserved, MBZ \*/ |
| Starting line 133  sts = nvSysDia5Update(1); | Starting line 133  sts = nvSysDia5Update(); |
| Starting line 403  char nvSysDia5Update (unsigned int updatedADCTable)  {  unsigned int crc;  char sts;  memcpy((char \*)&SysNonV.Dia5Block, (char \*)&SysDia5Default, sizeof(SysDia5Default));  SysNonV.Dia5Block.updatedADCTable = updatedADCTable;  crc = modbus\_CRC ((unsigned char \*)&SysNonV.Dia5Block,  sizeof(SysDia5NV) - 2,  INIT\_CRC\_SEED);  SysNonV.Dia5Block.CRC = crc;  sts = eeBlockWrite((unsigned long)SysDia5Adr + 0x100, (unsigned char \*)&SysNonV.Dia5Block.Reference, sizeof(SysDia5Default));  // last\_routine = 0x21;  return (sts); /\* Propagate success/failure \*/  } | Starting line 403  char nvSysDia5Update (void)  {  unsigned int crc;  char sts;  memcpy((char \*)&SysNonV.Dia5Block, (char \*)&SysDia5Default, sizeof(SysDia5Default));  crc = modbus\_CRC ((unsigned char \*)&SysNonV.Dia5Block,  sizeof(SysDia5NV) - 2,  INIT\_CRC\_SEED);  SysNonV.Dia5Block.CRC = crc;  sts = eeBlockWrite((unsigned long)SysDia5Adr + 0x100, (unsigned char \*)&SysNonV.Dia5Block.Reference, sizeof(SysDia5Default));  // last\_routine = 0x21;  return (sts); /\* Propagate success/failure \*/  } |

Modified nvSysDia5 structure and nvSysDia5Update function to allow for selection of old or new ADC voltage table for use in calc\_tank function.