DESCRIPTION

This application note describes the procedure to start-up the BiSS C interface such as a master from the MB100 family. The following identification attributes are available:

- **EDS (electronic data sheet):** The *BiSS* EDS describes the electrical characteristics and operating conditions of a BiSS device and contains information about process data and parameter. The EDS is only once each device present and describes all contained slaves.
- **Device and manufacturer ID:** Every bus participant provides a unique ID to the system, based on manufacturer ID and device ID. Both IDs combined reference to file accessible for the control, which provides all relevant information and is based on XML file format.
- **Profile ID:** Alternatively the communication parameters can be derived from application specific profiles that are referenced by a profile ID that is stored in the slave.

This document describes the start-up based on the EDS.

Within the start-up register communication does read all the EDS (electronic data sheet) parameters of all BiSS bus participants and does optionally reduce the timeout. Based on the read parameter the minimum permitted MA clock frequency, the maximum processing time for sensor data and the minimum cycle time is calculated. After this the data channel parameter of the BiSS master are set.

SEQUENCE

Before initialisation the maximum MA clock frequnecy needs to be set depending on the used cable length. Therefore the following table can be used that is applicable for encoders with common RS422 interface:

Cable length	Clock frequency
up to 10 m	10 MHz
up to 25 m	5 MHz
up to 60 m	2 MHz
up to 100 m	1 MHz
up to 200 m	500 kHz
up to 500 m	200 kHz
up to 1000 m	100 kHz

Table 1: MA clock frequency for RS422 interfaces

The reciprocal frequency as is the clock period on MA is provided to the initialisation routine. If there are no restrictions, the value 0 can be used. Additionally the cycle time is also provided to the initialisation routine The value 0 does indicate here too that this is the shortest possible cycle time

The initialisation routine sequence is described next. The following TCL program code shows an implemenation.

1. First the command BREAK is sent to the *BiSS* master to abort a possible running data transmission. After this the power supply of the connected slaves is turned on. The register communication of all participants should be available after a delay of 1 second. The *BiSS* master revision is verified by the *BiSS* master register read access on the *BiSS* master adresses 0xEA (=234) and ID 0xEB (=235). This application note requires a *BiSS* master revision number of minimum 2 as is equivalent to redesign Z, Y or later. It is also possible to determine the availability of data channels (slaves) in this master by writing and verifying the addresses 0xC0 (=192) to 0xDF (=223) of the *BiSS* master.
2. The MA clock frequency is programmed by the parameter FREQ and is calculated by the parameter and the used system clock of the *BiSS* master. Within the initialisation phase the MA clock frequency should be limited to a maximum of 6.375 MHz, permitting a processing time of up to 40 µs . The resulting MA clock period is stored in the variable T(MA).

3. The parameter MO_BUSY is set dependant on the FREQ parameter to achieve a processing time of minimum 40 µs .
4. The cycle time needs to be calculated depending on the *BiSS* master revision.
   (a) *BiSS* master with a revision number of minimum 4 can execute register communication even when not all sensor data are clocked out. Therefore the parameter HOLDCDM needs to be set. A cycle required minimum 14 MA clocks plus a processing time of 40 µs maximum plus a *BiSS* timeout of 40 µs maximum. The ”14 clock periods” result through a maximium transmission delay of 8 slaves daisy chained each channel plus 5 clock periods for the *BiSS* header and a stop bit before the *BiSS* timeout.
   (b) *BiSS* master with a revision number less than 4 can only execute register communication when the SL data line is low after the last MA clock as is when all data bits are clocked out. Therefore the data length of all data channels are set to the maximum of 64 bit and the CRC verification is deactivated by using the CRC polynome 0x00. A cycle required minimum 14 MA clocks plus 64 clocks each data channel plus a processing time of 40 µs maximum plus a *BiSS* timeout of 40 µs maximum.
5. All data channel parameter are set to 0. Depending on the master revision all data length are programmed to 64 bit or the HOLDCDM flag is set.
   Additionally REGVERS (*BiSS* C register communication) and ENMO (output of the MO programmed processing delay for bus structures) are set. For all channels BISSMOD (*BiSS* C) is set and SELSSI (SSI output format) is reset.
   With SLAVELOC = 1 all (logic) data channel are assigned to the first (physical) channel.
6. After power on it is possible that the data line SL is not automatically ’high’. Therefore the command INIT needs to be executed in the *BiSS* master. After the previously calculated cycle time this INIT command needs to be finished. The flag EOT in the status register needs to be ’1’ and the data line SL need to be ’high’ now. The state of all SL lines is checkable by the ”Statusinformation2” SL1 . . . SL8.
7. Now it is possible to start an endless sequence of cycles by setting the AGS to ”1”. Check the status register after one cycle time for correctness.
8. Next it is tried to read the EDS pointer, the manufacturer ID and the profile ID of all slaves. The highest slave address with a valid register access is stored in the variable MaxSlaveId.
   (a) On a valid EDS pointer the corresponding bank is read and verified. The EDS contains a check sum on the last address. To verify the EDS´s complete content from address 0x00 to 0x3E (=62) is added and divided by 256. The remaining division rest needs to match with the content on address 0x3F(=63).
   (b) The EDS is assigned to its slave addresses by the parameters SL_NUM (address 0x11 (=17)) and SL_OFF (address 0x12 (=18)). Each slave address is only one EDS permitted.
9. In this phase a loop counting backwards through the slave addresses starting at MaxSlaveId does process all parameters of all participants. Therefore the EDS structure is used. An external XML file or the BiSS profile can not be used for this.
   (a) The minimum permitted MA clock period is calculated by the maximum of TMA (address 4 in the EDS, unit 1 ns) of all participants.
   (b) To calculate the maximum timeout the parameter TO_MAX, TOS_MAX and TCLK_MAX are used and the abilities of all participants for a short BiSS timeout are checked to be activatable.
      i. If the in variable T(MA) stored clock period is smaller than the parameter TOS_MIN (address 0x07 (=7), unit 25 ns) then the parameter TOS_MAX (address 0x08 (=8), unit 25 ns) can be used. Therefore the short BiSS timeout needs to be activated in all slaves by individual writing of the value 7 in address 0x7C (=124) to activate this short BiSS timeout.
      As an EDS may carry multiple slaves in one device, the parameter SL_NUM (address 0x11 (=17)) and SL_OFF (address 0x12 (=18)) of all relevant slave IDs are needed to be determined.
      ii. If the in variable T(MA) stored clock period is bigger than the parameter TOS_MIN (address 0x07 (=7), unit 25 ns) then the parameter TO_MAX (address 0x06 (=6), unit 250 ns) needs to be applied.

iii. Additionally the adaptive timeout parameter TCLK_MAX (address 0x0A (=10), unit 25 ns) needs to be considered. The maximum timeout is calculated by the formula:
3 * TCLK_MAX + 1.5 * T(MA).

(c) To calculate the maximum processing time for sensor data the based on the parametern TBUSY_S (address 0x0D (=13), unit 250 ns) and BUSY_S (address 0x0E (=14)) the following formula needs to be applied:
TBUSY_S + BUSY_S * T(MA).

(d) The minimum permitted cycle time results on the maximum TCYC (address 0x0B (=11), unit 250 ns) of all participants.

(e) And finally the data channel parameter of all slaves in the EDS are analyzed. The parameter DC_NUM (address 0x10 (=16)) determines the count of data channels of this device.

10. After reading all parameter the MA clock frequency, the processing time and the cycle time can be calculated.

11. Now the master can be configured. Therefore the AGS (Automatic Get Sensor data) should be reset on the AGS bit (address 0xF4 (=244)) or by inhibiting the external GETSENS signal.

12. To test only one cycle is executed. After the cycle time the status register can be read. No error states should be present.

```
##############################################################################
# Procedure   : mb::Eds
# Editor      : Sz
# Date        : 23.02.2012
# Description : Read the EDS of the conected devices and configures the master.
# Example     : mb::Eds -t_ma 400E-09 -t_cycle 31.25E-06
##############################################################################
icdefine_proc_attributes mb::Eds \
  -command_group "mb" \
  -info "Connects␣to␣Hardware." \
  -define_args {
    {-t_ma    "Clockperid␣at␣MA" "" double optional 0.0}
    {-t_cycle "Cycletime"        "" double optional 0.0}
  }
proc mb::Eds args {
  icparse_proc_arguments -args $args Options

  # 1. Determine BiSS-Master
  mb::WriteRegister 244 128 ;# BREAK
  BissPowerOn
  after 1000 ;# Wait 1 sec after Power-On until control communication is available.
  foreach {Revision Identifier} [mb::ReadRegister 234 2] {}
  if {$Identifier>=64 && $Identifier<128} {
    set Master MB[expr $Identifier + 36]
  } elseif {$Identifier==131 || $Identifier==132} {
    set Master MB[expr 1 - $Identifier]
  } else {
    return -code error "ERROR␣unknown␣master."
  }
  if {$Identifier==131} {
    return -code error "$Master␣not␣supported."
  } elseif {($Identifier==132 && $Identifier<1)} {
    return -code error "ERROR␣Master-Revision:␣Need␣1␣or␣newer␣of␣BiSS-Master␣$Master."
  } elseif {($Identifier>=64 && $Identifier<128 && $Revision<1)} {
    return -code error "ERROR␣Master-Revision:␣Need␣1␣or␣newer␣of␣BiSS-Master␣$Master."
  }
  # Determine number of data channels (Slaves)
  for {set Slaves 0} {$Slaves<8} {incr Slaves} {
    mb::WriteRegister [expr 192 + 4*$Slaves] $Slaves
    if {   [mb::ReadRegister [expr 192 + 4*$Slaves]]!=$Slaves
        || [mb::ReadRegister 192]!=0} {
      break
    }
  }
  puts "Info:␣Found␣BiSS-Master␣$Master,␣Revision␣$Revision␣with␣$Slaves␣data␣channels."

  # 2. Calculate clock-frequency at MA
  if {$Options(-t_ma)>12.5E-06} {
    return -code error "Clock-frequency␣at␣MA␣must␣not␣be␣less␣than␣80␣KHz"
  }
  if {$Options(-t_ma)<[expr 40.0E-06 / 255]} {
    # use minimum clock period to guarantee busy time
    set FREQ [expr $mb::Fclk / 2.0 * (40.0E-06 / 255) - 1.0]
    if {$FREQ>15} {
      set FREQ [expr $mb::Fclk / 20.0 * (40.0E-06 / 255) - 1.0]
    }
  } else {
    set FREQ [expr $mb::Fclk / 2.0 * $Options(-t_ma) - 1.0]
    if {$FREQ>15} {
      set FREQ [expr $mb::Fclk / 20.0 * $Options(-t_ma) - 1.0]
    }
  }
```

```
set FREQ [expr int($FREQ)<[format "%.3e" $FREQ] ? int($FREQ) + 1 : int($FREQ)]
if {$FREQ<16} {
  set T(MA) [expr (2.0 * ($FREQ + 1.0)) / $mb::Fclk]
} elseif {$FREQ<32} {
  set T(MA) [expr (20.0 * ($FREQ - 15.0)) / $mb::Fclk]
} else {
  return -code error "ERROR␣FREQ:␣Out␣of␣range␣(FREQ=␣$FREQ)"
}
puts "Info:␣Configure␣initial␣FREQ␣to␣$FREQ␣(F(MA)=␣[format␣"%.2e Hz"␣[expr␣1/$T(MA)]])"

# 3. Calculate MO_BUSY (parameterized processing time = 40 us)
set T(BUSY) 40.0E-06
set MO_BUSY [expr $T(BUSY) / $T(MA)]
set MO_BUSY [expr int($MO_BUSY)<[format "%.3e" $MO_BUSY] ? int($MO_BUSY) + 1 : int(
    $MO_BUSY)]
if {$MO_BUSY<256} {
  set T(BUSY) [expr $MO_BUSY * $T(MA)]
} else {
  return -code error "ERROR␣MO_BUSY:␣Out␣of␣range␣(T(BUSY)=␣[format␣"%.2e sec"␣$T(BUSY)],
      ␣MO_BUSY=␣$MO_BUSY)"
}
puts "Info:␣Configure␣initial␣MO_BUSY␣to␣$MO_BUSY␣(T(BUSY)=␣[format␣"%.2e sec"␣$T(BUSY)])
    "

# 4. Calculate Cycletime (depending on Master Revision)
set T(TIMEOUT) 40.0E-06
if {($Identifier>=128 || $Revision>=4)} {
  # (a) Calculate Cycletime: (14*MA-periods + processing time + timeout)
  set FREQAGS [expr $mb::Fclk / 20.0 * [expr 14.0*$T(MA) + $T(BUSY) + $T(TIMEOUT)] - 1.0]
  if {$FREQAGS>123} {
    set FREQAGS [expr $mb::Fclk / 625.0 * [expr 14.0*$T(MA) + $T(BUSY) + $T(TIMEOUT)] +
        127.0]
  }
} else {
  # (b) Calculate Cycletime: ((14+64*Slaves)*MA-periods + processing time + timeout)
  set FREQAGS [expr $mb::Fclk / 20.0 * [expr (14+64*$Slaves)*$T(MA) + $T(BUSY) + $T(
      TIMEOUT)] - 1.0]
  if {$FREQAGS>123} {
    set FREQAGS [expr $mb::Fclk / 625.0 * [expr (14+64*$Slaves)*$T(MA) + $T(BUSY) + $T(
        TIMEOUT)] + 127.0]
  }
}
set FREQAGS [expr int($FREQAGS)<[format "%.3e" $FREQAGS] ? int($FREQAGS) + 1 : int(
    $FREQAGS)]
if {$FREQAGS<128} {
  set T(CYCLE) [expr (20.0 * ($FREQAGS + 1.0)) / $mb::Fclk]
} elseif {$FREQAGS<256} {
  set T(CYCLE) [expr (625.0 * ($FREQAGS - 127.0)) / $mb::Fclk]
} else {
  return -code error "ERROR␣FREQAGS:␣Out␣of␣range␣(T(CYCLE)=␣[format␣"%.2e sec"␣$T(CYCLE)
      ],␣FREQAGS=␣$FREQAGS)"
}
puts "Info:␣Configure␣initial␣FREQAGS␣to␣$FREQAGS␣(T(CYCLE)=␣[format␣"%.2e sec"␣$T(CYCLE)
    ])"

# 5. Write Configuration (depending on Master Revision)
if {($Identifier>=128 || $Revision>=4)} {
  for {set i 0} {$i<$Slaves} {incr i} {
    mb::WriteRegister [expr 192 + 4*$i] {0x00 0x00 0x00 0x00 };# Reset Datachannel
        Parameter
  }
  mb::WriteRegister 229 0x43                    ;# REGVERS=1, ENMO=1, HOLDCDM=1
```

```
  } else {
    for {set i 0} {$i<$Slaves} {incr i} {
      mb::WriteRegister [expr 192 + 4*$i] {0x7F 0x00 0x00 0x00 };# Set Datachannel
          Parameter
    }
    mb::WriteRegister 229 0x42                      ;# REGVERS=1, ENMO=1, HOLDCDM=0
  }
  mb::WriteRegister 230 $FREQ
  mb::WriteRegister 232 $FREQAGS
  mb::WriteRegister 233 $MO_BUSY
  mb::WriteRegister 236 {0x01 0x55 0x55 0x00} ;# SLAVELOC=1, SELSSI=0, BISSMOD=1

  # 6. Execute Init
  mb::WriteRegister 244 16
  after [expr int($T(CYCLE)) + 1] ;# wait one cycletime
  if {[expr [mb::ReadRegister 240] & 0x01]!=1} {
    return -code error "Error␣Init:␣Command␣not␣completed.\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Maybe␣no␣BiSS␣device␣connected."
  }
  # Test SL-Line after Init
  if {[expr [mb::ReadRegister 248] & 0x01]!=1} {
    return -code error "Error␣Init:␣SL-Line␣is␣not␣high␣after␣timeout."
  }

  # 7. start Autocycle
  mb::WriteRegister 244 1
  after [expr int($T(CYCLE)) + 1] ;# wait one cycletime
  if {[expr [mb::ReadRegister 240] | 0x27]!=255} {
    return -code error "Error␣Autocycle:␣BiSS-Frames␣are␣incorrect."
  }
  puts "Info:␣Init␣OK."

  # 8. Read DEVICE-EDS
  mb::WriteRegister 228 0x01  ;# CHSEL=1. ToDo: For all Channels
  set MaxSlaveId -1 ;# Detect number of Slaves
  for {set SlaveId 0} {$SlaveId < 8} {incr SlaveId} {
    if {[catch {set EDS_BANK($SlaveId) [mb::BissReadRegister $SlaveId 65 1]}]} {
      set EDS_BANK($SlaveId) -1
    } elseif {$EDS_BANK($SlaveId)<255} {
      if {[catch {set EDS_CONTENT($SlaveId) [mb::BissReadBank $SlaveId $EDS_BANK($SlaveId)
          ]}]} {
        return -code error "Error␣reading␣EDS-Content␣from␣SlaveId␣$SlaveId␣in␣Device␣
            $Device"
      }
      # (a) Test checksum
      set Checksum 0
      for {set i 0} {$i<63} {incr i} {
        set Checksum [expr $Checksum + [lindex $EDS_CONTENT($SlaveId) $i]]
      }
      if {[expr ($Checksum & 0xFF)]!=[lindex $EDS_CONTENT($SlaveId) 63]} {
        return -code error "Checksum␣Error:␣EDS␣from␣SlaveId␣$SlaveId,␣Bank␣$EDS_BANK(
            $SlaveId)␣in␣Device␣$Device"
      }
      # (b) Assign EDS to SlaveIds
      set SL_NUM [lindex $EDS_CONTENT($SlaveId) 17]
      set SL_OFF [lindex $EDS_CONTENT($SlaveId) 18]
      for {set i 0} {$i < $SL_NUM} {incr i} {
        if {[info exists EDS([expr $SlaveId + $i - $SL_OFF])]} {
          return -code error "Error:␣Found␣two␣EDS␣for␣Slave␣[expr␣$SlaveId␣+␣$i␣-␣$SL_OFF]
              .\n\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣First␣under␣SlaveId␣$EDS([expr␣$SlaveId␣+␣$i␣-␣$SL_OFF])␣and\
    n\
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣and␣second␣under␣SlaveId␣$SlaveId"
        } elseif {[expr $SlaveId + $i - $SL_OFF]<0 || [expr $SlaveId + $i - $SL_OFF]>7} {
          return -code error "Error:␣EDS␣under␣SlavId␣$SlavId␣points␣to␣SlaveId␣[expr␣
              $SlaveId␣+␣$i␣-␣$SL_OFF]"
        } else {
          set EDS([expr $SlaveId + $i - $SL_OFF]) $SlaveId
        }
      }
      if {[expr $SlaveId + $SL_NUM - 1 - $SL_OFF] > $MaxSlaveId} {
        set MaxSlaveId [expr $SlaveId + $SL_NUM - 1 - $SL_OFF]
      }
    } else {
      set EDS_BANK($SlaveId) -1
      if {$SlaveId > $MaxSlaveId} {
        set MaxSlaveId $SlaveId
      }
    }
    if {[catch {set MANUFACTURER_ID($SlaveId) [mb::BissReadRegister $SlaveId 126 2]}]} {
      set MANUFACTURER_ID($SlaveId) -1
    } else {
      set MANUFACTURER_ID($SlaveId) [format "%4X" [expr 256*[lindex $MANUFACTURER_ID(
          $SlaveId) 0] + [lindex $MANUFACTURER_ID($SlaveId) 1]]]
      if {$MANUFACTURER_ID($SlaveId) == "0000" || $MANUFACTURER_ID($SlaveId) == "FFFF"} {
        set MANUFACTURER_ID($SlaveId) -1
      }
      if {$SlaveId > $MaxSlaveId} {
        set MaxSlaveId $SlaveId
      }
    }
    if {[catch {set PROFILE_ID($SlaveId) [mb::BissReadRegister $SlaveId 66 2]}]} {
      set PROFILE_ID($SlaveId) -1
    } else {
      set PROFILE_ID($SlaveId) [format "%4X" [expr 256*[lindex $PROFILE_ID($SlaveId) 0] + [
          lindex $PROFILE_ID($SlaveId) 1]]]
      if {$PROFILE_ID($SlaveId) == "0000" || $PROFILE_ID($SlaveId) == "FFFF"} {
        set PROFILE_ID($SlaveId) -1
      }
      if {$SlaveId > $MaxSlaveId} {
        set MaxSlaveId $SlaveId
      }
    }
    puts "Info:␣SlaveId␣$SlaveId:␣EDS_BANK=␣$EDS_BANK($SlaveId),␣ManufacturerId=␣
        $MANUFACTURER_ID($SlaveId),␣ProfileId=␣$PROFILE_ID($SlaveId)"
  }
  # 9. Analyze DEVICE-EDS
  set DC          1  ;# DataChannel counter
  set Device      1  ;# Device counter
  set TMA         0  ;# Minimum permitted clock period at MA
  set TBUSY_S     0  ;# Maximum processing time SCD
  set TO_MAX      0  ;# Maximum BiSS Timeout
  set TCYC        0  ;# Minimum permitted cycle time
  set DataRange   0  ;# Data range length
  set SlaveConfig {} ;# Initialize Slave-Parameter
  for {set SlaveId $MaxSlaveId} {$SlaveId >= 0} {set SlaveId [expr $SlaveId - 1]} {
    if {[info exists EDS($SlaveId)]} {
      # Using EDS
      if {$EDS($SlaveId) == $SlaveId} {
        # (a) Calculate TMA
        if {[expr 1.0E-09 * [lindex $EDS_CONTENT($SlaveId) 4]] > $TMA} {
          set TMA [expr 1.0E-09 * [lindex $EDS_CONTENT($SlaveId) 4]]
        }
        # (b) Set short timeout if T(MA) < TOS_MIN
```

```
if {$T(MA)<[expr 25.0E-09 * [lindex $EDS_CONTENT($SlaveId) 7]]} {
  # in all slaves of the current device
  set SL_NUM [lindex $EDS_CONTENT($SlaveId) 17]
  set SL_OFF [lindex $EDS_CONTENT($SlaveId) 18]
  for {set i 0} {$i < $SL_NUM} {incr i} {
    puts "Info: Configure short timout at SlaveId [expr $SlaveId + $i - $SL_OFF]"
    catch {mb::BissWriteRegister [expr $SlaveId + $i - $SL_OFF] 124 7}
  }
  # Calculate TO_MAX using TOS_MAX
  if {[expr 25.0E-09 * [lindex $EDS_CONTENT($SlaveId) 8]] > $TO_MAX} {
    set TO_MAX [expr 25.0E-09 * [lindex $EDS_CONTENT($SlaveId) 8]]
  }
} else {
  # Calculate TO_MAX using TO_MAX
  if {[expr 250.0E-09 * [lindex $EDS_CONTENT($SlaveId) 6]] > $TO_MAX} {
    set TO_MAX [expr 250.0E-09 * [lindex $EDS_CONTENT($SlaveId) 8]]
  }
}
# Calculate TO_MAX TCLK_MAX (adaptive timeout)
if {[expr 3 * 25.0E-09 * [lindex $EDS_CONTENT($SlaveId) 10] + 1.5 * $T(MA)] >
    $TO_MAX} {
  set TO_MAX [expr 3 * 25.0E-09 * [lindex $EDS_CONTENT($SlaveId) 10] + 1.5 * $T(MA)
      ]
}
# (c) Calculate TBUSY_S using TBUSY_S and BUSY_S
if {[expr 250.0E-09 * [lindex $EDS_CONTENT($SlaveId) 12] + $T(MA) * [lindex
    $EDS_CONTENT($SlaveId) 13]] > $TBUSY_S} {
  set TBUSY_S [expr 250.0E-09 * [lindex $EDS_CONTENT($SlaveId) 12] + $T(MA) * [
      lindex $EDS_CONTENT($SlaveId) 13]]
}
# (d) Calculate TCYC using TCYC
if {[expr 250.0E-09 * [lindex $EDS_CONTENT($SlaveId) 11]] > $TCYC} {
  set TCYC [expr 250.0E-09 * [lindex $EDS_CONTENT($SlaveId) 11]]
}
# (e) Get Data Channel Parameter
puts "Info: Number of data channels in device $Device: [lindex $EDS_CONTENT(
    $SlaveId) 16]"
for {set i 1} {$i<=[lindex $EDS_CONTENT($SlaveId) 16]} {incr i} {
  puts "Info: Bank address for content description of data channel $DC: [lindex
      $EDS_CONTENT($SlaveId) [expr $i*4 + 16]]"
  set DLEN [lindex $EDS_CONTENT($SlaveId) [expr $i*4 + 17]]
  if {[expr [lindex $EDS_CONTENT($SlaveId) [expr $i*4 + 18]] & 0x01]==0} {
    set TYP "Sensor"
  } else {
    set TYP "Actuator"
  }
  if {[expr [lindex $EDS_CONTENT($SlaveId) [expr $i*4 + 18]] & 0x02]==0} {
    set ALIGN "Right"
  } else {
    set ALIGN "Left"
  }
  if {[expr [lindex $EDS_CONTENT($SlaveId) [expr $i*4 + 18]] & 0x08]==0} {
    set STOP 0
  } else {
    set STOP 1
  }
  set CPOLY [lindex $EDS_CONTENT($SlaveId) [expr $i*4 + 19]]
  puts "Info: Datalength of data channel $DC: $DLEN"
  puts "Info: Dataformat of data channel $DC: TYP= $TYP, ALIGN= $ALIGN, STOP=$STOP"
  if {$Revision<5} {set STOP 0}
  if {$DLEN>0} {
    lappend SlaveConfig [expr 128*$STOP + 64 + $DLEN - 1]
```

```
        } else {
          lappend SlaveConfig [expr 128*$STOP]
        }
        set DataRange [expr $DataRange + $DLEN + $STOP]
        puts "Info: Crc-Polynom of data channel $DC: [ic::ToBin $CPOLY 8]1"
        lappend SlaveConfig [expr 128 + $CPOLY]
        if {[expr $CPOLY & 0x40]>0} {
          set DataRange [expr $DataRange + 7]
        } elseif {[expr $CPOLY & 0x20]>0} {
          set DataRange [expr $DataRange + 6]
        } elseif {[expr $CPOLY & 0x10]>0} {
          set DataRange [expr $DataRange + 5]
        } elseif {[expr $CPOLY & 0x08]>0} {
          set DataRange [expr $DataRange + 4]
        } elseif {[expr $CPOLY & 0x04]>0} {
          set DataRange [expr $DataRange + 3]
        } elseif {[expr $CPOLY & 0x02]>0} {
          set DataRange [expr $DataRange + 2]
        } elseif {[expr $CPOLY & 0x01]>0} {
          set DataRange [expr $DataRange + 1]
        }
        lappend SlaveConfig 0 0
        incr DC
      }
      incr Device
    }
  } elseif {$MANUFACTURER_ID($SlaveId) > 0} {
    # Using external XML-File
    if {[file exists "idbiss${MANUFACTURER_ID($SlaveId)}.xml"]} {
      if {[catch {set DEVICE_ID($SlaveId) [mb::BissReadRegister $SlaveId 120 6]}]} {
        return -code error "Error reading DeviceId fom Slave $SlaveId"
      }
      puts "Info: Configure for Slave $SlaveId via external XML-File"
      return -code error "Error: Configure via external XML-File not implemented"
    } else {
      return -code error "Error: External XMF-File idbiss${MANUFACTURER_ID($SlaveId)}.xml
          not found"
    }
  } elseif {$PROFILE_ID($SlaveId) > 0} {
    # Using ProfileId
    puts "Info: Cnfigure for Slave $SlaveId via ProfileId $PROFILE_ID($SlaveId)"
    # (a) Calculate TMA
    set TMA 100.0E-09
    # (b) Calculate TO_MAX
    set TO_MAX 40.0E-06
    # (c) Calculate TBUSY_S
    set TBUSY_S 40.0E-06
    # (d) Calculate TCYC
    set TBUSY_S .0E-06
    return -code error "Error: Configure via ProfileId not implemented"
  } else {
    return -code error "Error: Missing description for SlaveId $SlaveId"
  }
}
while {[llength $SlaveConfig] < [expr 4*$Slaves]} {lappend SlaveConfig 0 0 0 0}

# 10. Calculate clock-frequency at MA
puts "Info: Minimum permitted clock period at MA= [format "%.2e sec" $TMA]"
set T_MA [expr $Options(-t_ma)==0 ? $TMA : $Options(-t_ma)]
if {$T_MA<$TMA} {
  return -code error "Required T_MA= $T_MA not posible with connected devices!"
}
```

```
set FREQ [expr $mb::Fclk / 2.0 * $T_MA - 1.0]
if {$FREQ>15} {
  set FREQ [expr $mb::Fclk / 20.0 * $T_MA - 1.0]
}
set FREQ [expr int($FREQ)<[format "%.3e" $FREQ] ? int($FREQ) + 1 : int($FREQ)]
if {$FREQ<16} {
  set T(MA) [expr (2.0 * ($FREQ + 1.0)) / $mb::Fclk]
} elseif {$FREQ<32} {
  set T(MA) [expr (20.0 * ($FREQ - 15.0)) / $mb::Fclk]
} else {
  return -code error "ERROR␣FREQ:␣Out␣of␣range␣(FREQ=␣$FREQ)"
}
puts "Info:␣Configure␣final␣FREQ␣to␣$FREQ␣(F(MA)=␣[format␣"%.2e Hz"␣[expr␣1␣/␣$T(MA)]])"

# Calculate MO_BUSY
puts "Info:␣Maximum␣processing␣time␣SCD=␣[format␣"%.2e sec"␣$TBUSY_S]"
set MO_BUSY [expr $TBUSY_S / $T(MA)]
set MO_BUSY [expr int($MO_BUSY)<[format "%.3e" $MO_BUSY] ? int($MO_BUSY) + 1 : int(
    $MO_BUSY)]
if {$MO_BUSY<256} {
  set T(BUSY) [expr $MO_BUSY * $T(MA)]
} else {
  return -code error "ERROR␣MO_BUSY:␣Out␣of␣range␣(T(BUSY)=␣[format␣"%.2e sec"␣$T(BUSY)],
      ␣MO_BUSY=␣$MO_BUSY)"
}
puts "Info:␣Configure␣final␣MO_BUSY␣to␣$MO_BUSY␣(T(BUSY)=␣[format␣"%.2e sec"␣$T(BUSY)])"

# Calculate Cycletime
puts "Info:␣Length␣of␣data␣range=␣$DataRange"
puts "Info:␣Maximum␣BiSS␣Timeout=␣[format␣"%.2e sec"␣$TO_MAX]"
puts "Info:␣Minimum␣permitted␣cycle␣time=␣[format␣"%.2e sec"␣$TCYC]"
# Cycletime = ((6+DataRange)*MA-periods + processing time + timeout)
if {$TCYC<[expr (6+$DataRange)*$T(MA) + $T(BUSY) + $TO_MAX]} {
  set TCYC [expr (6+$DataRange)*$T(MA) + $T(BUSY) + $TO_MAX]
}
set T_CYCLE [expr $Options(-t_cycle)==0 ? $TCYC : $Options(-t_cycle)]
if {$T_CYCLE<$TCYC} {
  return -code error "Required␣T_CYCLE=␣$T_CYCLE␣not␣posible␣with␣connected␣devices!"
}
set FREQAGS [expr $mb::Fclk / 20.0 * $T_CYCLE - 1.0]
if {$FREQAGS>123} {
  set FREQAGS [expr $mb::Fclk / 625.0 * $T_CYCLE + 127.0]
}
set FREQAGS [expr int($FREQAGS)<[format "%.3e" $FREQAGS] ? int($FREQAGS) + 1 : int(
    $FREQAGS)]
if {$FREQAGS<128} {
  set T(CYCLE) [expr (20.0 * ($FREQAGS + 1.0)) / $mb::Fclk]
} elseif {$FREQAGS<256} {
  set T(CYCLE) [expr (625.0 * ($FREQAGS - 127.0)) / $mb::Fclk]
} else {
  return -code error "ERROR␣FREQAGS:␣Out␣of␣range␣(T(CYCLE)=␣[format␣"%.2e sec"␣$T(CYCLE)
      ],␣FREQAGS=␣$FREQAGS)"
}
puts "Info:␣Configure␣final␣FREQAGS␣to␣$FREQAGS␣(T(CYCLE)=␣[format␣"%.2e sec"␣$T(CYCLE)])
    "

# 11. Configure Master
mb::WriteRegister 244 0                  ;# Stop autocycle
mb::WriteRegister 230 $FREQ
mb::WriteRegister 232 $FREQAGS
mb::WriteRegister 233 $MO_BUSY
mb::WriteRegister 192 $SlaveConfig       ;# Config DataChannel in Master
```

```
  # 12. Test
  mb::WriteRegister 244 4                    ;# Start single cycle
  after [expr int($T(CYCLE)) + 1]
  if {[expr [mb::ReadRegister 240] & 0x01]==0} {
    return -code error "Error GETSENS: Command not completed."
  }
  if {[expr [mb::ReadRegister 240] & 0x10]==0} {
    return -code error "Error GETSENS: Single Cycle Data Error."
  }
  if {[expr [mb::ReadRegister 240] & 0x40]==0} {
    return -code error "Error GETSENS: Watchdog Error."
  }
  mb::WriteRegister 244 1                    ;# Start autocycle
  puts "Info: Configuration of BiSS Master finished."
}
```

READ BiSS SENSOR DATA

Sensor data is generated with every cycle and transmitted completely within one ***BiSS* frame** as is one Single Cycle Data (SCD) cycle.

To read sensor data one *BiSS* frame needs to be executed. *BiSS* frames can be executed by:

- The *BiSS* master command **GETSENS0** triggers a single BiSS frame. Write the value 0x04 into the command register in address 0xF4 (=244).
- The command **AGS** triggers BiSS frames endlessly on given cycle time. Therefore the cycle time needs to be set with the parameter FREQAGS in register address 0xE8 (=232) and then write the value 0x01 into the command register in address 0xF4 (=244) to start and keep the AGS running.
- The logic input signal respectively input pin **GETSENS** triggers on every 'high' level a single *BiSS* frame. Therefore write into register FREQAGS in address 0xE8 (=232) the cycle time value 0x7F for "INFINITE" and then write the value 0x01 into the command register in address 0xF4 (=244) to start and keep the AGS.

The following errors may occur:

- A BiSS frame can not be started, due to a 'low' SL data line or the last BiSS frame has not ended yet.
  The master sets in this case in the status register the nWDERR-Flag to 'low'.
  The control needs to abort the running frame with a BREAK and wait for 40 µs.
  If the SL data line keeps on staying in the 'low' state, an additional INIT needs to be executed.
  A cause of this error may be a too short cycle time.
- A *BiSS* frame can not be completed.
  The EOT flag of the status register stays in this case 'low'.
  The control needs to terminate the running frame with a BREAK and wait for 40 µs.
  If the SL data line keeps on staying in the 'low' state, an additional INIT needs to be executed.
- The verification one to all data channels CRC check sums failed.
  The master sets in this case the nSCDERR flag to 'low'.
  With the SVALID flags in the addresses 0xF1 and 0xF2 the control can verify which data channel have had a failed CRC verification.
  A root cause for this error may be a misconfigured data channel parameter set.
  On misconfigured data channels the CRC check sums may fail also on all following data channels even if the following data channels are not misconfigured.

BiSS REGISTER ACCESS

The register access is configured in the address range 0xE0 (=224) to 0xE5 (=229).

The following parameters need to be set:

- Set the first address of the address range to be accessed: REGADR = ”start address”.
- Communication direction (e.g. read): WNR = 0.
- Number of registers to read: REGNUM = (number of bytes to read - 1).
- Channel selection (e.g. for channel 1): CHSEL = 0x01.
- Type of control communication ”register access”: CTS = 1.
- Protocol type BiSS C: REGVERS = 1.
- Select slave ID (e.g. for the slave with the slave ID 0): SLAVEID = 0x00.

Before every register access the CDM timeout (minimum of 14 cycles with CDM = ”0”) needs to be executed.
The master indicates the presense of the CDM timeout by the CDMTO (bit 7 in address 0xF3 (=243)).
To start the register access the INSTR in the command register in address 0xF4 (=244) needs to be written with the value 0b100.
As there are multiple *BiSS* frames required to complete a register access with AGS= 1 the automatic or pin triggered sequence of cycles can be activated.
With start of the register communication the master does reset the INSTR to 0b000 (bit 3 downto 1 in address 0xF4 (=244) (=244)) and at the end of the access in the status register (address 0xF0 (=240)) the flags REGEND (bit 2) and with success nREGERR (bit 3) are set.
In case of an error with the register access the count of correct transmitted register bytes is available in REGBYTES (bit 5 to 0 in address 0xF3 (=243)).

The following TCL procedure shows by example the implementation of a register read access:

```
##############################################################################
# Procedure   : mb::BissReadRegister
# Editor      : Sz
# Date        : 14.12.2011
# Description : Executes a BiSS read register access.
# Example     : set Data [mb::BissReadRegister 0 64 8 1]
##############################################################################
icdefine_proc_attributes mb::BissReadRegister  \
  -command_group "mb" \
  -info "Executes␣a␣BiSS␣read␣register␣access." \
  -define_args {
    {SlaveId "Slave␣ID"                       "" integer required}
    {Adr     "Address"                        "" integer required}
    {Number  "Number␣of␣registers␣to␣transfer" "" integer optional 1}
    {-pData  "Pointer␣for␣read␣data"          "" structure optional}
    {-Chsel  "Channel␣select"                 "" integer optional}
    {-Single "Executes␣single␣register␣access" "" boolean optional}
  }
proc mb::BissReadRegister args {
  icparse_proc_arguments -args $args Options

  if {[info exists Options(-pData)]} {
    upvar $Options(-pData) Data
    set Data {}
  }
  # mb::Regsiter(228): CHSEL(8:1)
  # mb::Regsiter(229): CTS=1, REGVERS=1, SLAVEID, -, ENMO, HOLDCDM
  if {[info exists Options(-Chsel)]} {
    mb::WriteRegister 228 [list \
      [expr $Options(-Chsel)] \
      [expr (0xC0 + ($Options(SlaveId) << 3) + 0x$mb::Register(229) & 0x07)] \
    ]
```

```
  } else {
    mb::WriteRegister 229 [expr 0xC0 + ($Options(SlaveId) << 3) + (0x$mb::Register(229) & 0
        x07)]
  }
  if {[info exists Options(-Single)]} {
    set NumRegs 1
  } else {
    set NumRegs $mb::NumRegs
  }
  set Return {}
  for {set i 0} {$NumRegs * $i < $Options(Number)} {incr i} {
    # mb::Regsiter(226): WNR=0, REGADR(6:0)
    # mb::Regsiter(227):  , , REGNUM
    mb::WriteRegister 226 [list \
      [expr $NumRegs * $i + $Options(Adr)] \
      [expr ($Options(Number) - $NumRegs * $i) >= $NumRegs ? $NumRegs - 1 : $Options(Number
          ) - $NumRegs * $i - 1] \
    ]
    # Start register access
    mb::WriteRegister 244 9; # INSTR="100", AGS=1
    # Test REG=0 in INSTRUCTION register
    set mb::Timeout 0
    after 1000 set mb::Timeout 1
    while {[expr [mb::ReadRegister 244] & 0x08] == 1 && $mb::Timeout == 0} {
      update
    }
    after cancel set mb::Timeout 1
    if {$mb::Timeout == 1} {
      mb::ReadRegister
      return -code error "TIMEOUT-ERROR: Can not start register access"
    }
    # Test REGEND=1 in STATUS register
    set mb::Timeout 0
    after 1000 set mb::Timeout 1
    while {[expr [mb::ReadRegister 240] & 0x04] == 0 && $mb::Timeout == 0} {
      update
    }
    after cancel set mb::Timeout 1
    mb::ReadRegister
    if {$mb::Timeout == 1} {
      return -code error "TIMEOUT-ERROR: Invalid register access"
    }
    # Test nREGERR=1 in STATUS register
    if {[expr (0x$mb::Register(240) & 0x08)] == 0} {
      for {set j 0} {$j < [expr 0x$mb::Register(243) & 0x3F]} {incr j} {
        lappend Return [expr 0x$mb::Register([expr 128 + $j])]
        if {[info exists Options(-pData)]} {
          lappend Data [expr 0x$mb::Register([expr 128 + $j])]
        }
      }
      return -code error "ERROR: nREGERR = 0. [llength $Return] bytes correct transfered."
    }
    for {set j 0} {$j < (($Options(Number) - $NumRegs * $i) >= $NumRegs ? $NumRegs :
        $Options(Number) - $NumRegs * $i)} {incr j} {
      lappend Return [expr 0x$mb::Register([expr 128 + $j])]
      if {[info exists Options(-pData)]} {
        lappend Data [expr 0x$mb::Register([expr 128 + $j])]
      }
    }
  }
  return $Return
}
```

REQUIREMENTS

This *BiSS* master initialisation sequence requires some application conditions as are typical applications:

- *BiSS* C sensor, one slave with occupying ID = 0.
- Point to point structure.

By structure there are three levels:

Host

- Controls the *BiSS* master.
- Uses the sensors content:
  - Measurement data
  - Sensor information
  - User information (USER DATA)

***BiSS* master**

- Coodinates the communication to the sensor.
- Controlled by the host.
- Monitored by the host.

***BiSS* Sensor**

- Generates measurement data:
  - Position information
  - Status information
- Provides device information.
- Provides user information (USER DATA).
- Stores user information (USER DATA).

INITIALIZE BiSS MASTER

- Verify host communication with master: write and verify test words in uncritical addresses.
- Verify master version and master revision with approvals (0xEA und 0xEB).
- Load master with default configuration:
  - Configure master:
    - Placement of the slave 1 in SLAVELOC.
    - Activate sensor data ACTnSENS = 0.
    - *BiSS* protocoll (not SSI).
    - *BiSS* C protocoll.
    - Frequency divider FREQ.
    - Frequency divider AutoGetSens(AGS).
    - Start bit delay on MO.
  - Configure sensor:
    - Only one sensor
      - Sensor data length to ”safe superset”: SCDLEN1 = 63.
      - Enable SCD: ENSCD1 = 1.
      - Deaktivate GRAY: GRAYS1 = 0.
      - Deactivate SCD CRC verification: SCRCPOLY = 0x00.
      - Set CRC inverting activ (at that time no function): INVCRCS1 = 1.
    - Deactivate all other additional sensors
      - SCDLENx = 0.
      - ENSCDx = 0.
      - GRAYx = 0.
      - SCRCPOLY = 0.
      - INVCRCSx = 0.
  - Configure control communication:
    - Set start address to 0x78: REGADR.
    - Set communication direction to read: WNR = 0.
    - Count of registers to be read: REGNUM = 0 (equates to 1 byte).
    - Channel selection to channel 1: CHSEL = 0x01.
    - Set control communication type to ”register access”: CTS = 1.
    - Protocoll type *BiSS* C: REGVERS = 1.
    - Addressed slave ID = 0 : SLAVEID = 0x00.
    - Deactivate master out: EnMO = 0.
    - Deactivate CDM output extension: HOLDCDM = 0
- Read master status:
  - Read status word 1: 0xF0.
  - Read status word 2: 0xF8.

INITIALIZE BiSS INTERFACE

- If EOT = 0 :
  - Send command BREAK = 0x80 in 0xF4.
  - Wait until EOT = 1 or timeout by host.
- Initialisation of the *BiSS* interface by INIT command:
  - Write command INIT 0x10 into command register 0xF4 (INIT).
  - Wait until EOT = 1 in status word 1 or timeout by host.
- Verify if sensor is connected electrically:
  - Check SL1 line level: ”1” required.

IDENTIFY BiSS SENSOR

There are different techniques to identify a *BiSS* sensor.
The following basic information is important for the *BiSS* master:

- SCD data length
  - Position MT, ST
  - Flags nE, nW
  - Lifecycle counter LC
  - Temperature value TEMP
- Applied CRC polynome.
- Inverted CRC transmission.
- Additional data channel.

Depending on the technique host relevant data is also available:

- Device ID
- Serial number
- Temperature range
- Current consumption
- Power supply voltage range
- Tolerances
- Mechanical data of the material measure
- Mechanical limits
- . . .

BiSS sensor does not provide any identification

- Use of the existing configuration.
- Manual configuration.
- ”Trial and error” test of matching configurations,
  might require a move of the sensor/angular position (see *BiSS* AN6) .

The cause of missing identification attributes may be missing or wrong entries in the slave or not supporting register access (e.g. *BiSS* C unidirectional)

BiSS sensor provides BiSS identifier and XML file

- Reading the BiSS identifier 0x78 .. 0x7F.
- Splitting into
  - Manufacturer ID: 0x7E and 0x7F.
  - Device ID: 0x78 ... 0x7D.
- Parsing to the XML file with the *BiSS* identifier as the parsing key.
- Transfer of all contained data into the master configuration.

BiSS sensor provides *BiSS* Profile BPx

- Reading the *BiSS* profile identifier in address 0x42 and 0x43.
- Verification with the in the master defined BiSS profiles and assignment of the relevant parameter.

***BiSS* sensor provides EDS**

- Process EDS common part:
  - Reading the EDS bank 0x41.
  - Selection of the EDS bank in bank select register 0x40.
  - Reading the first bank of the EDS from the mapped bank register address range 0x00 ... 0x3F.
  - Verification the EDS content with the total check sum CHKSUM = 0x3F [common EDS].
  - Verification the EDS version if processable 0x00 [common EDS].
  - Processing of the EDS content:
    - Verify common communication parameter 0x04 ... 0x12 with active configuration, update on demand.
    - Slave 1 communication parameter:
      - Bank1 in 0x14 [common EDS].
      - DLEN1 in 0x15 [common EDS].
      - FORMAT1 in 0x16 [common EDS].
      - CPOLY1 in 0x17 [common EDS].
    - verify additional slaves with empty/deactiated communication.
      - Verify content ”0x00” in 0x18[common EDS] ... 0x33[common EDS].
    - Update of the sensor configuration in the master.

- Process EDS BPx profile specific part:
  - Selection of EDS PBx of bank 1 in the bank selection 0x40
  - Reading the first bank of the EDS BPx from the mapped bank register address range 0x00 ... 0x3F.
  - Verification the BPx specific EDS content with the total check sum CHKSUM = 0x3F [BPx specific EDS].
  - Verification the EDS BPx version if processable 0x00 [common EDS].
  - Verification of the the BiSS profile version with the BiSS profile entry in 0x42 and 0x43.
  - Processing of the EDS BPx content:
    - Read common parameter of the sensor 0x04 ... 0x39 and transfer to the host.

- Read in USER DATA:
  - Data format defined by the end user.
  - Data securing defined by the end user.
  - Data encryption defined by the end user.
  - Data range for USER DATA storage on EDS allowance (bank range) to be managed host.

RECONFIGURE BiSS MASTER TO SENSOR

- Stop of optional cyclic sequences(e.g. AGS or GETSENS trigger signals)
- Configure master
  - Placement of slave 1: SLAVELOC.
  - Activate sensor data: ACTnSENS.
  - BiSS protocol not SSI: SELSSIx.
  - BiSS C protocol: BiSSMODx.
  - Frequency divider: FREQ.
  - Frequency divider AutoGetSens(AGS): FREQAGS.
- Configure sensor 1
  - Set sensor data length SCDLEN1.
  - Enable SCD: ENSCD1 = 1.
  - Deactivate GRAY: GRAYS1 = 0
  - Deactivate SCD CRC verification: SCRCPOLY = 0x00.
  - Activate CRC inverting (no relevant function here): INVCRCS1 = 1.
- Continuing of optional cyclic sequences(e.g. AGS or GETSENS trigger signals)

READING BiSS POSITION DATA

With INSTR command

- If EOT = 0 :
  - Send command BREAK = 0x80 in 0xF4.
  - Wait until EOT = 1 or timeout by host.
- Send command INSTR = 0b100 as a command 0x04 into the command register 0xF4.
- Wait until EOT = 1 or timeout by host.
- Check of nERR
  - The signal is low active: 0 = error state.
  - General error occured.
  - Additional output on NERR.
- Check on nSCDERR
  - An error occured in all received SCD data, CRCs do not match to received data:
    - Errors can be mapped to the sensor data with the SVALID flags in 0xF1 and 0xF2.
    - Apply the SVALID1 in 0xF1 Bit 1 for the sensor data.
- Check on nWDERR
  - A watchdog error occured while using the AGS function:
    - Exceeding the cycle time by too long sensor processing time.
    - Exceeding the cycle time by too long line delay.
    - Exceeding the cycle time by other error.
- Position data in SCDATA 1 valid
  - SCD lowest byte in 0x00.
  - SCD highest byte in 0x07.
  - The received CRC is right aligned available in 0x07 if SCD data lengths < 56 bit.

With AGS

- If EOT = 0 :
  - Send command BREAK = 0x80 in 0xF4.
- Wait until EOT = 1 or timeout by host.
  - Send command INSTR = 0b01 as a command 0x04 into the command register 0xF4.

With GETSENS signals

- Host requests synchronous to the GETSENS signal.
- Host timeout monitoring required.
- No AGS function applicable e.g. watchdog not available.

REVISION HISTORY

Rev	Notes	Pages affected
A1	Initial version	
A2	Wrong address for 192: 0xCD replaced by 0xC0	2