ELT SENSOR
Environment Leading Technology

Programming Guide
I2C Bus

S-Series, T-Series models

cdg, 2016. 6.

T: +082-32-719-8055
E-mail: cdg@eltsensor.co.kr

www.eltsensor.co.kr

ELT SENSOR
Environment Leading Technology

Contents

Applied models

- T Series model
  - T100, T110

- S Series model
  - S100, S100H, S300, S300E

✓ **Prior to next page**

※ Notes.

1. Co2 module update equation every three seconds.
2. Therefore, it can be ideally once read the data every three seconds.
3. It is recommended to read once or twice within 3 seconds.
4. Although it is possible to read data at a high rate, however the co2 value can lead to errors due to an microcontroller unit overload.

Compare of each models

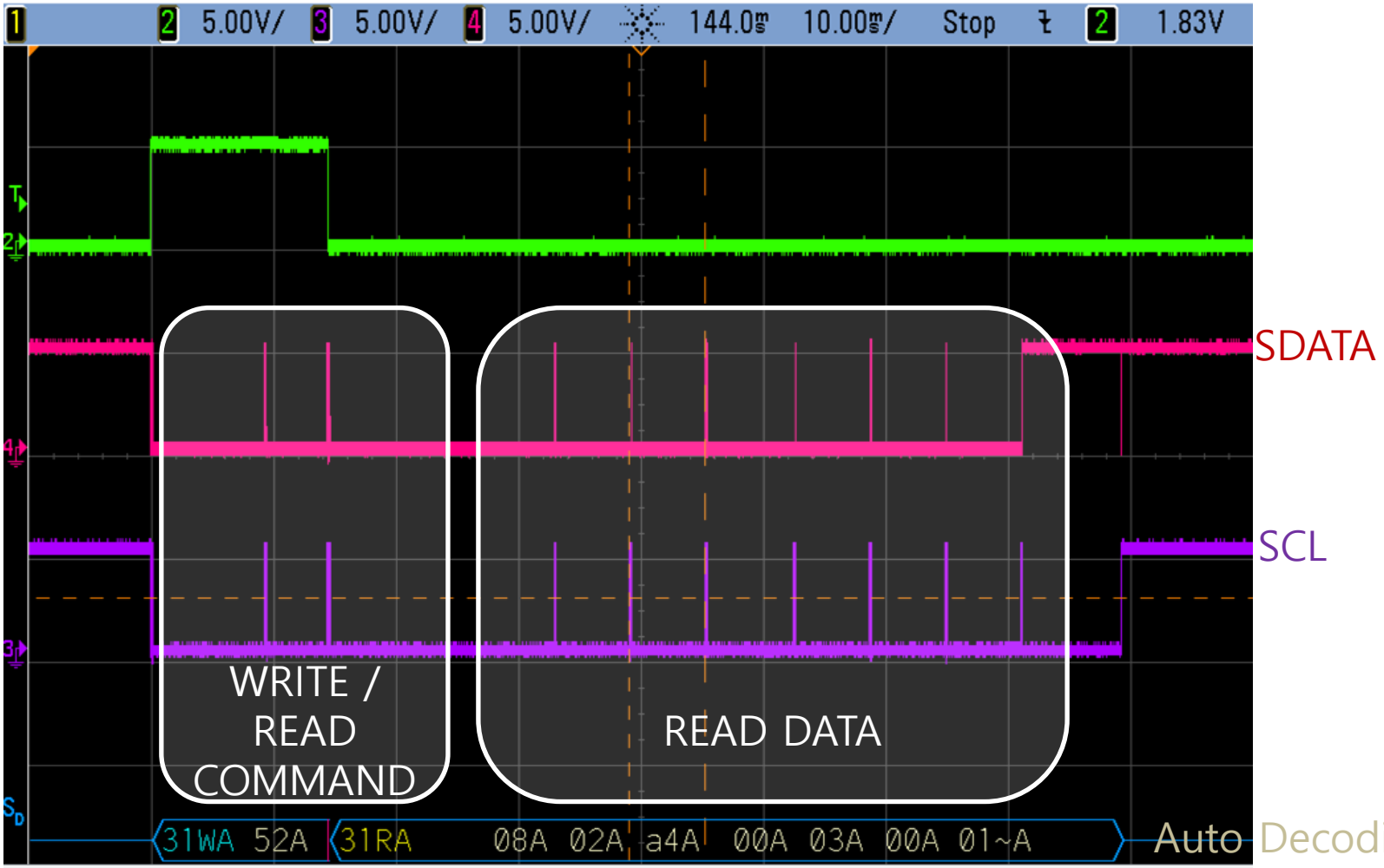
- I2C classified according to the MCU

	ADuC848/ Analog Device	Atmega 16/ Atmel
▪ Applied model	S300, T100, T110	S100, S100H, S300E
▪ Pull-up resister on I2C bus line	MCU's interanll weak pull-up[1]	10KΩ on PCB[2]
▪ Max I2C speed	400kHz[3]	400kHz[4]
▪ Recommend I2C speed	100kHz[5]	100kHz[6]
▪ Interval time delay between I2C byte	10ms[7]	10ms[8]

[1][2] It is need adjust resister value when i2c bus line don't have stable.
[3][4] It is value up to speed of mcu's i2c functionality.
[5][6] If you have some issues then you try better with i2c low speed.
[7][8] It can be without delay but actually need some delay for stable i2c communication.

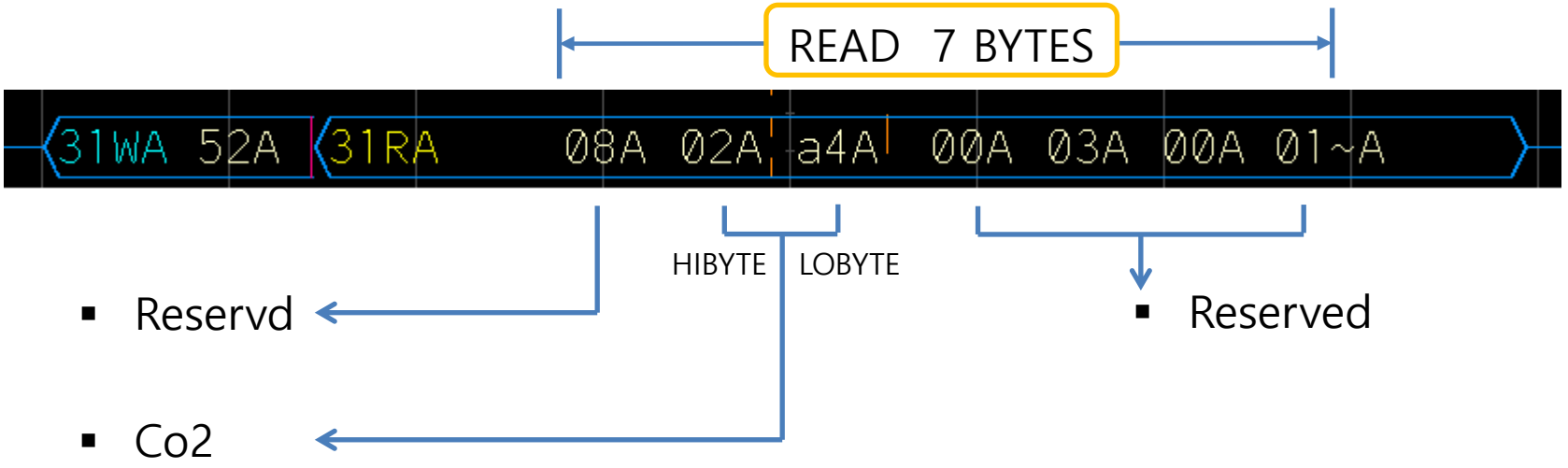
I2C Frame Sequence (1/4)

- All frame patten of oscilloscope
  - Example of 5V MCU models.



I2C Frame Sequence (2/4)

- Detail read data format.



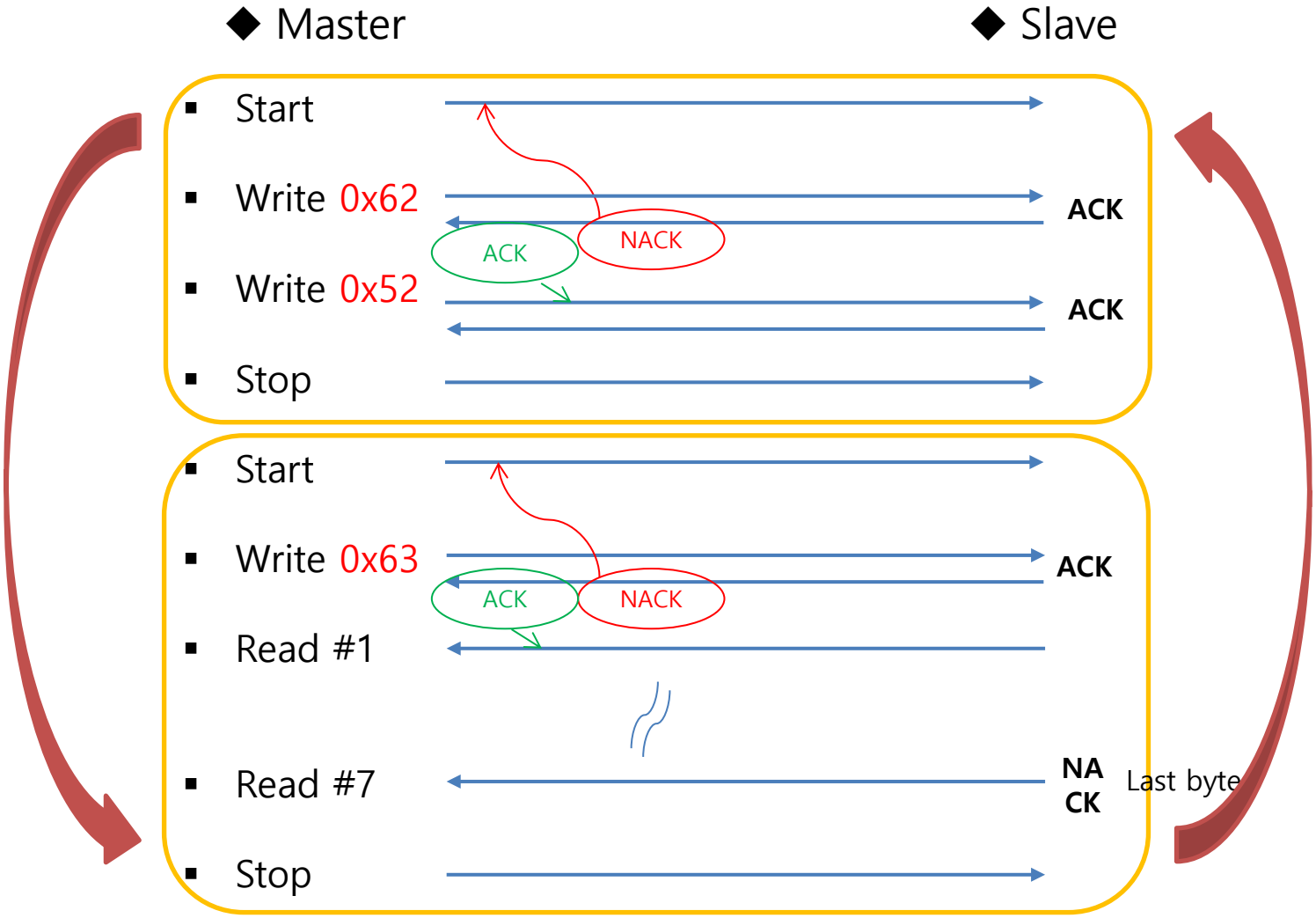
- Co2 Value.
  - Co2 value is word 0x02a4 (Example above in the frame)
  - Co2 default value is 500 (0x1F4)

    ※ If you get co2 value by i2c within 3 second after sensor module power on then you will receive 0x1F4

    (D300 models : 10 second)

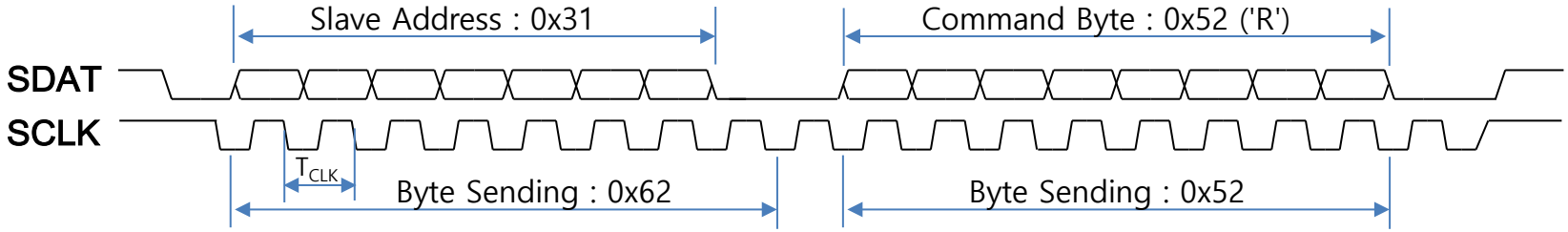
I2C Frame Sequence (3/4)

- All of I2C Command steps.

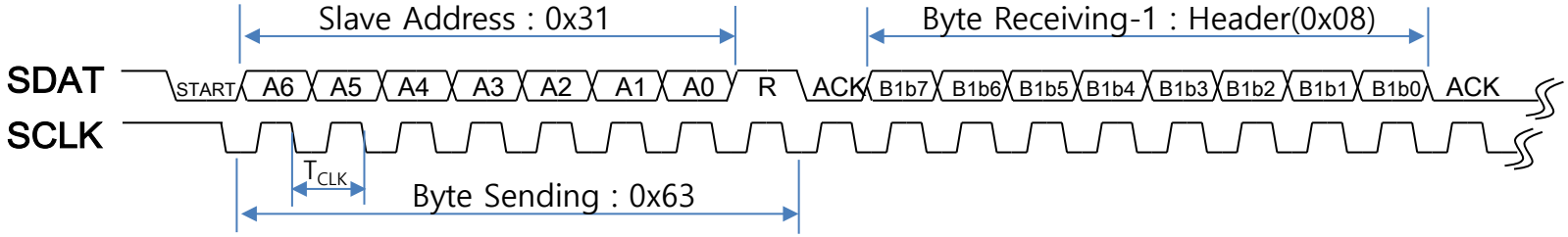


I2C Frame Sequence (4/4)

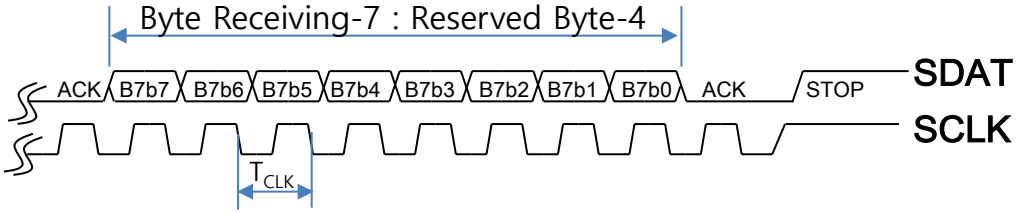
- 'R'(0x52) Command Write



- Read Data(7-Bytes)

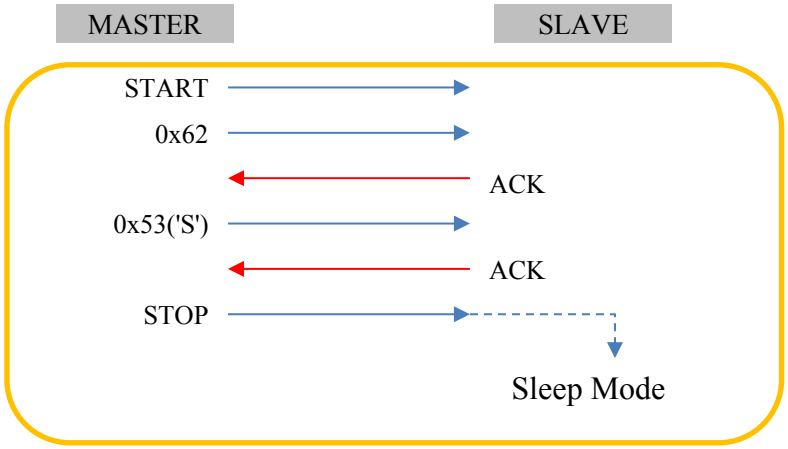


[Continual Bytes]
Receiving Byte-2 : Upper byte of PPM of CO2
Receiving Byte-3 : Lower byte of PPM of CO2
Receiving Byte-4 : Reserved Byte-1
Receiving Byte-5 : Reserved Byte-2
Receiving Byte-6 : Reserved Byte-3



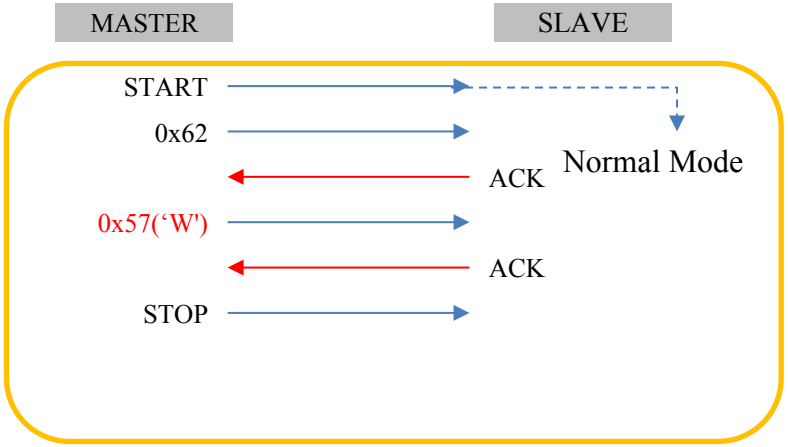
Kind of I2C Command (1/6)

Sleep Mode Command



□ 'S'(0x53)-Command : Sleep Mode Starting

If The 'S'-Command is sent, the slave is insert to sleep mode within 4 seconds

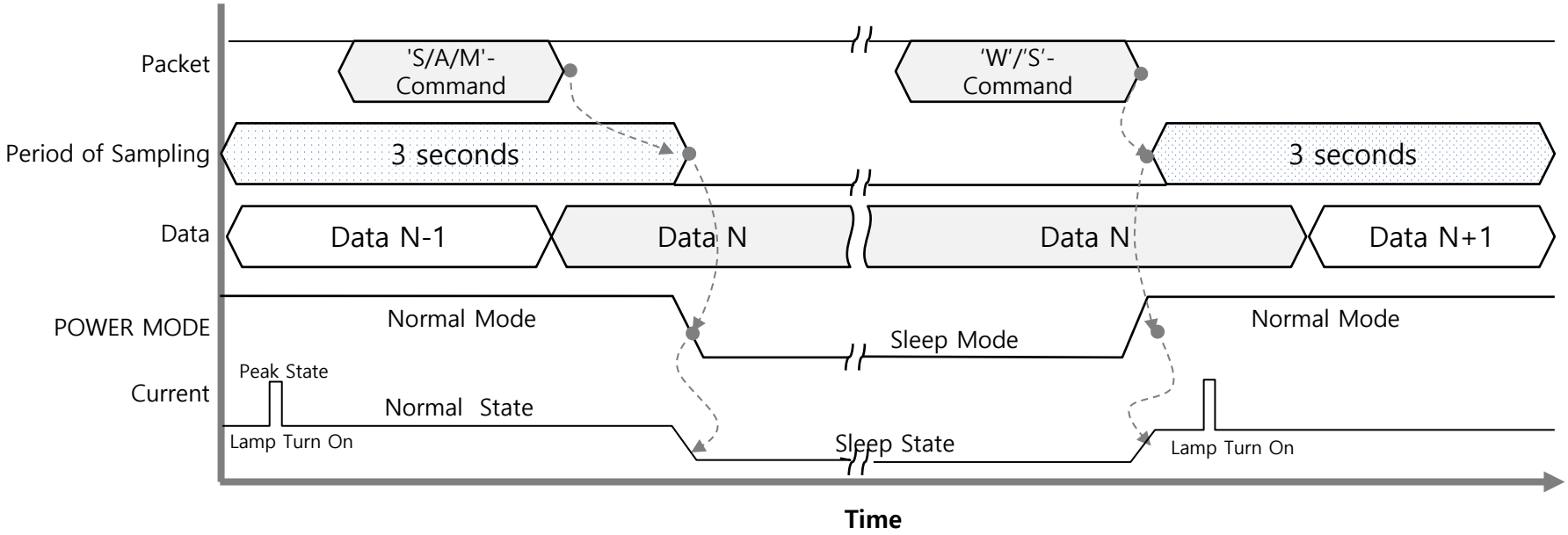


□ 'W'(0x57)-Command : Wake-up

If the 'A'-command is sent, the slave is awoke from the sleep mode. The slave is only check the sub-address in I2C packet.
So, if the 'S'-command is sent in sleep mode, the slave is awoke and re-insert to the sleep mode after 3 seconds.
And you can use the 'R'-command instead of 'A'-command, but you will received the invalid data(old data) from sensor. because of the valid data is reported after 3-seconds.

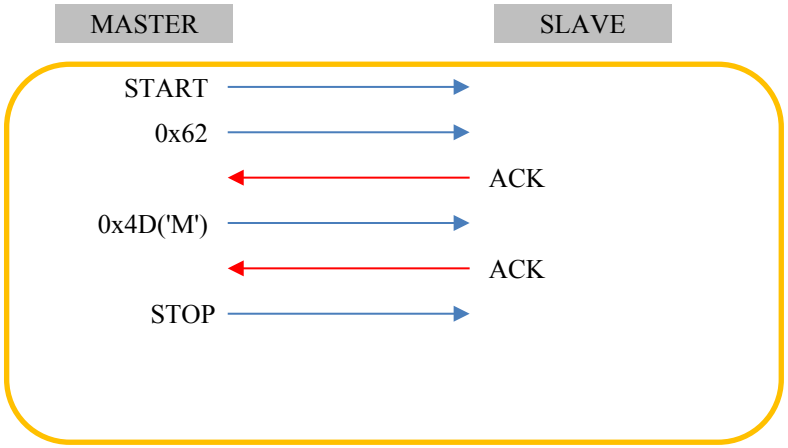
Kind of I2C Command (1/6)

- Timing Diagram of Sleep Mode



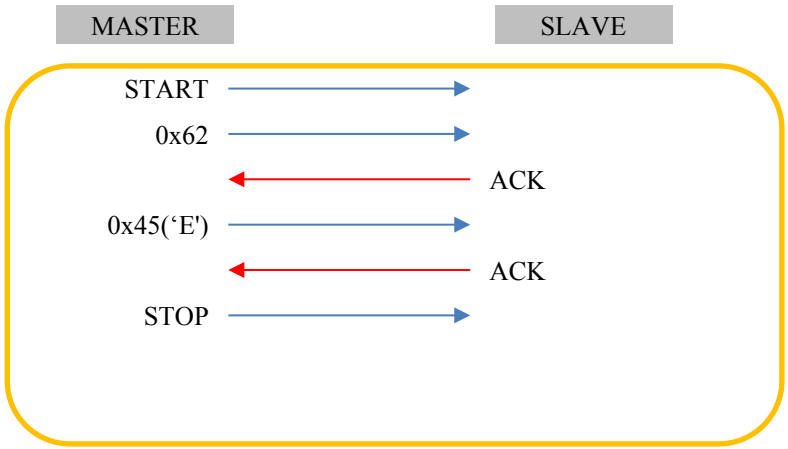
Kind of I2C Command (2/6)

MCDL Command



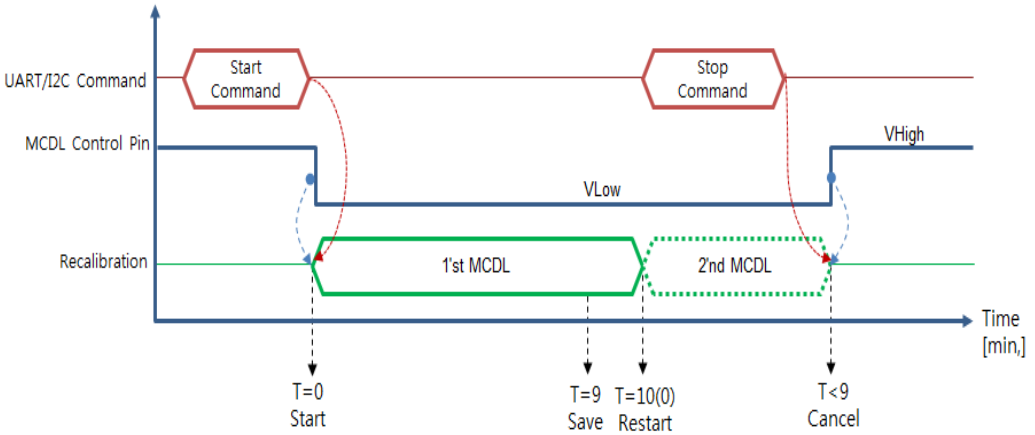
☐ 'M'(0x4D)-Command : MCDL Mode Starting

10' MCDL(10 minute Non-Periodic Manual Calibration) start as sensor receives 'M' command from Main Board and repeated 9 minute Period until the sensor get 'E' command.



☐ 'E'(0x45)-Command : END

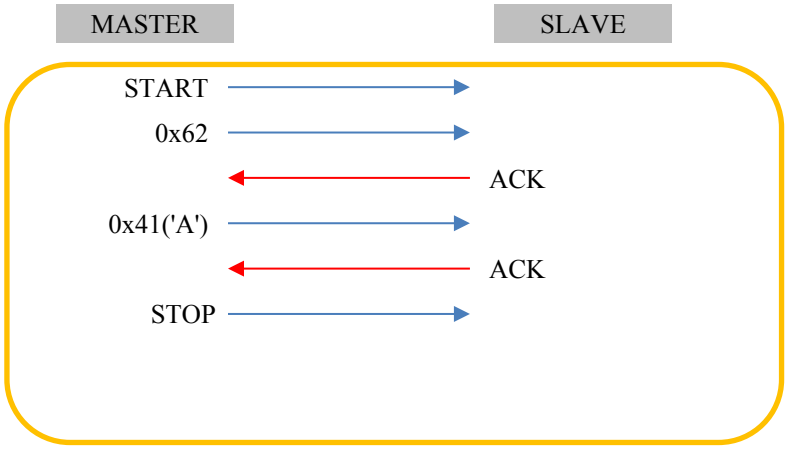
Sensor quit MCDL and store the recalibrated value of 10' MCDL.
CF) Clear command is available to erase the recalibrated value.



Time Diagram of MCDL

Kind of I2C Command (3/6)

- ACDL Command

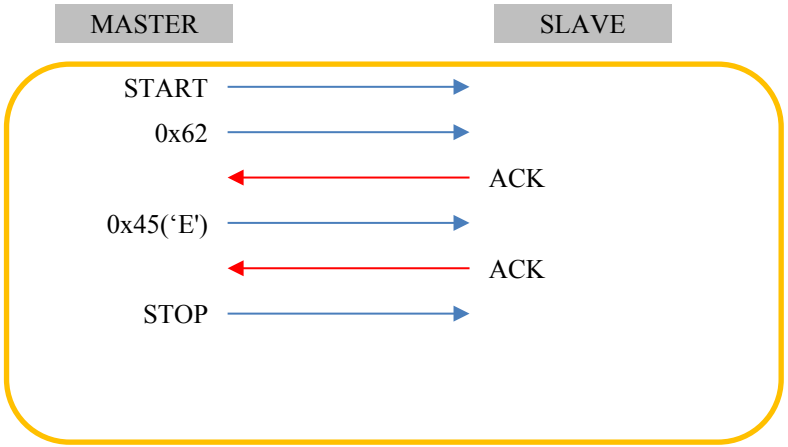


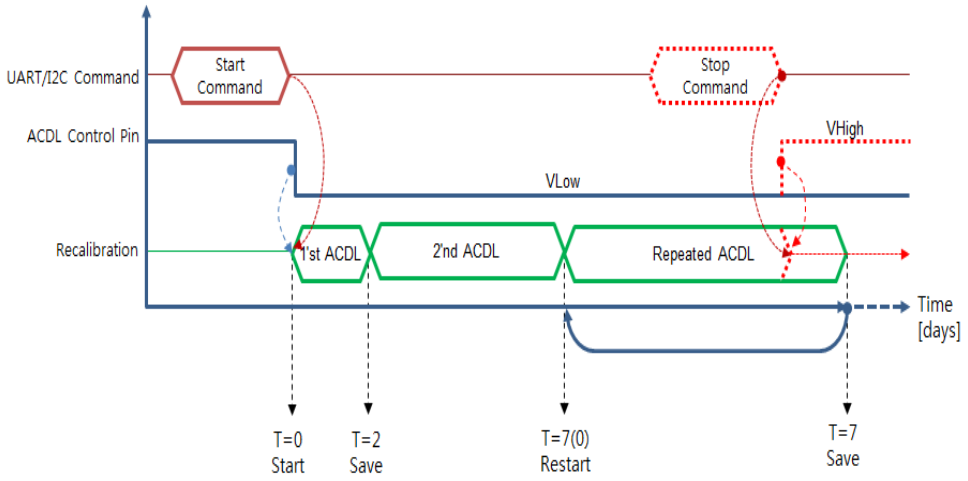
☐ 'A'(0x41)-Command : ACDL Starting

Sensor start weekly Auto-calibration by it self (ACDL) after once in 2 days since power-on until get 'E' command from main board.

☐ 'E'(0x45)-Command : END ACDL

Sensor quit ACDL and store the recalibrated value of ACDL.
CF) Clear command is available to erase the recalibrated value.

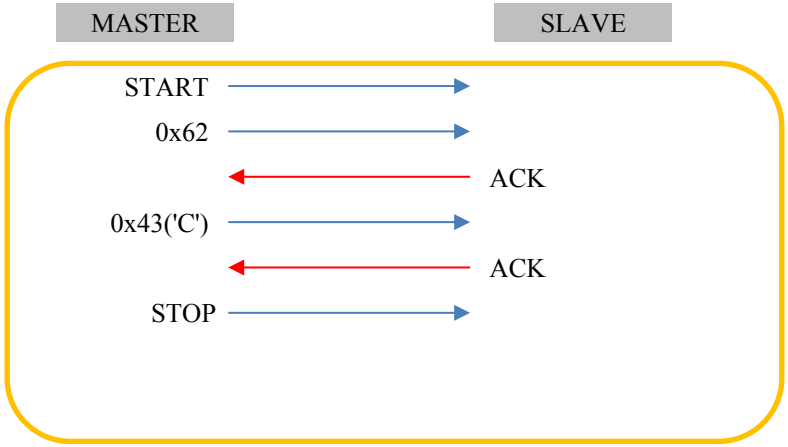




Time Diagram of ACDL

Kind of I2C Command (4/6)

- Clear Command (About MDCL / ACDL)

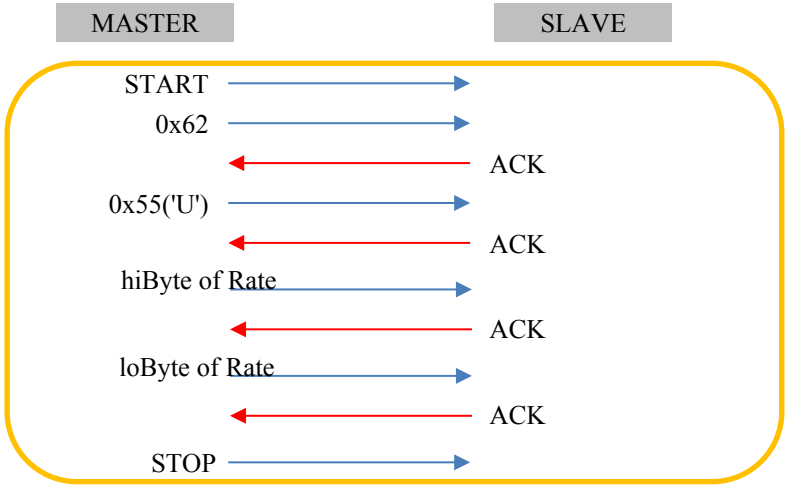


□ 'C'(0x43)-Command : CLEAR Recalibration Factor

Clear command is available to erase the recalibrated value by ACDL or MCDL.

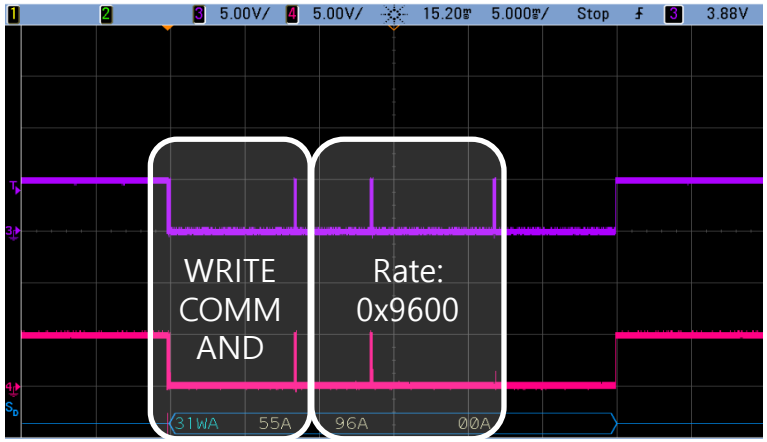
Kind of I2C Command (5/6)

UART RATE Change Command



□ 'U'(0x55)-Command : Uart rate change

- Target uart rate byte : hiByte, loByte
- Available uart rate :
  - 9600
  - 19200
  - 38400
- Target uart rate applied after sensor module power restart.

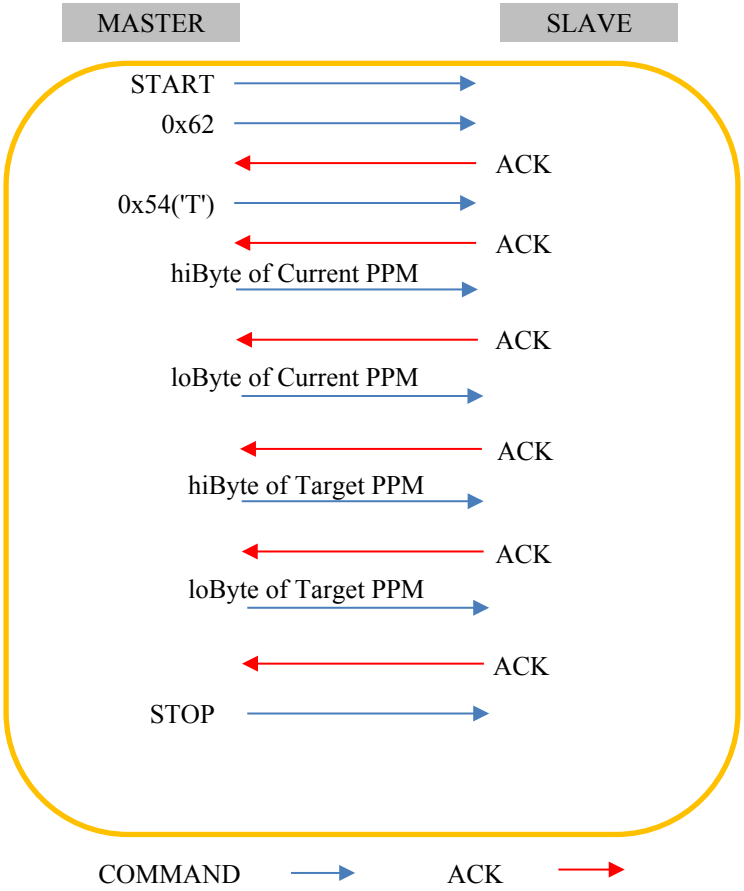


- Timing diagram of oscilloscope (5v mcu)

※ Currnetly supported model
- D300

Kind of I2C Command (6/6)

ReCalibration by Target PPM (with Current PPM) Command



※ Currnetly supported model
- S300 3V, T110 3V

□ 'T'(0x54)-Command : Target PPM

- Current PPM : 2 BYTE WORD (HI, LOW BYTE)
- Target PPM : 2 BYTE WORD (HI, LOW BYTE)
- Recalibration by directly with current and target ppm
- Target values reflects takes about more than 6 seconds after command

□ Recommendations

Directive use range for minimize error to effective sensor value
- PPM RANGE : 0 - 2000 ppm
- Target to up : 1000 ppm within differences, ex, 400 > 1400(max)
- Target to down: 300 ppm within differences, ex 1000 > 700(min)

□ Note

Directive use valid signal of I2C, MCDL, ACDL

- 2 seconds after power on or reset

Sample code (1/8)

```
#define PIN_I2C_CLK      PORTE_Bit4
#define PIN_I2C_DAT     PORTE_Bit5
#define SDA_INPUT       PINE_Bit5
#define SDA_DIRECT      DDRE_Bit5

unsigned char    giReadI2CFlag=0;
unsigned int     giCO2PPM;     //Storage of value of PPM

#ifdef I2C_DEBUGCOUNT
    unsigned int giI2cNack=0; //DEBUGONLY
    unsigned int giI2cCall=0; //DEBUGONLY
#endif

//################################
// File : Header.h
//--------------------------------

#define I2C_DEBUGCOUNT  //DEBUGONLY
#ifdef I2C_DEBUGCOUNT
  extern unsigned int giI2cNack; //DEBUGONLY
  extern unsigned int giI2cCall; //DEBUGONLY
#endif

#define USE_S100_SERIES

//This factor can be changeable by customer.
//I'd like to recommend customer to the 500msec or 1000msec
// S-100 : 1sample/1sec or 1sample/2sec
// S-200 : 2samples/1sec or 1sample/1sec
#define I2C_READ_PERIOD 200 //[msec]

//-----------------------------------
```

```
//################################
// File : Interrupt.c
//--------------------------------
unsigned int count_timer0=0;
unsigned int giReadI2cCount=0;  //DEBUGONLY

#pragma vector = TIMER0_COMP_vect                    // 1ms TIMER
__interrupt void Timer0_OutputCompare_Interrupt(void)
    {
    giReadI2cCount++;
    if(giReadI2cCount>=I2C_READ_PERIOD)
        {
        //After "I2C_READ_PERIOD" msec, the variable "giReadI2CFlag" is incleased to 1 or 2.
        if(giReadI2CFlag<2){giReadI2CFlag++;}else{;}
        giReadI2cCount=0;
        }else{;}
    }
//--------------------------------

/*
    The target board of this source code is ATmega 169p of 8MHz.
    And this code is using only GPIO port(Isn't using the H/W I2C Driver module!)

    The result timing diagram of SCLK is as blow.

    [USE_S100_SERIES : for ATmega16 of S-100]

        Total : 29.76us , 33.6kHz

        13.52us
       +-------+          |
       |       |16.98us|
     ---+       +-------+

    [NON USE_100_SERIES : for ADuC848 of S-200]

        Total : 14.69us, 68.1kHz

         6.05us
       +-------+         |
       |       | 8.64us|
     ---+       +-------+
*/
```

Sample code (2/8)

```
#ifdef USE_S100_SERIES
    //[S-100, S-100H] ATmega16L chip
    //Total time : 1.1msec + 100msec + 4msec
    //Maximum Frequency of SCLK : 33.6 kHz (29.76 us)

    //NOTICE : This setting value can be used for S-200 model (ADuC848 chip)

    #define I2C_DELAY_HALF     4 //Value of half delay time of TL & TH     [us]
    #define I2C_DELAY_FULL     8 //Value of delay time of TL & TH [us]

    // After one byte read or write, The waiting time is needed.
    // Because slave must analyze the receive data in I2C RX interrupt routine or
    // process another higher priority interrupt routine (ex:Timer-0)
    // The length of waiting time is depend slave MPU clock and Firmware
    #define I2C_DELAY_BYTE     100 //Value of waiting time after byte-writing or reading
    #define I2C_DELAY_SETUP 4      //Value of setup time for Start/Stop bits

    //In the S-100 series, The long waiting time is needed Between 'R'-Command Packet
and
    //Reading Packet
    #define I2C_DELAY_ATMEL_ISR 10 //For ATmega16

#else

    //[S-200] ADuC848 chip
    //Total time : 3mS
    //Maximum Frequency of SCLK : 80.9KHz (12.36us)

    //NOTICE : This setting value can't be used for S-200 model (ADuC848 chip)

    //Value of half delay time of TL & TH     [us] (Half = (1,000,000
/FrequencyOfTargetClock)/4-2)
    #define I2C_DELAY_HALF     1

    //Value of delay time of TL & TH [us] (Full = (1,000,000 /FrequencyOfTargetClock)/2-4)
    #define I2C_DELAY_FULL     2

    // After one byte read or write, The waiting time is needed.
    // Because slave must analyze the receive data in I2C RX interrupt routine or
    // process another higher priority interrupt routine (ex:Timer-0)
    // The length of waiting time is depend slave MPU clock and Firmware
    #define I2C_DELAY_BYTE     100 //Value of waiting time after byte-writing or
reading(Min=60us)
    #define I2C_DELAY_SETUP 1      //Value of setup time for Start/Stop bits

#endif
```


```
#ifdef USE_S100_SERIES
    //[S-100, S-100H] ATmega16L chip
    //Total time : 1.1msec + 100msec + 4msec
    //Maximum Frequency of SCLK : 33.6 kHz (29.76 us)

    //NOTICE : This setting value can be used for S-200 model (ADuC848 chip)

    #define I2C_DELAY_HALF     4 //Value of half delay time of TL & TH     [us]
    #define I2C_DELAY_FULL     8 //Value of delay time of TL & TH [us]

    // After one byte read or write, The waiting time is needed.
    // Because slave must analyze the receive data in I2C RX interrupt routine or
    // process another higher priority interrupt routine (ex:Timer-0)
    // The length of waiting time is depend slave MPU clock and Firmware
    #define I2C_DELAY_BYTE     100 //Value of waiting time after byte-writing or reading
    #define I2C_DELAY_SETUP 4      //Value of setup time for Start/Stop bits

    //In the S-100 series, The long waiting time is needed Between 'R'-Command Packet
and
    //Reading Packet
    #define I2C_DELAY_ATMEL_ISR 10 //For ATmega16

#else

    //[S-200] ADuC848 chip
    //Total time : 3mS
    //Maximum Frequency of SCLK : 80.9KHz (12.36us)

    //NOTICE : This setting value can't be used for S-200 model (ADuC848 chip)

    //Value of half delay time of TL & TH     [us] (Half = (1,000,000
/FrequencyOfTargetClock)/4-2)
    #define I2C_DELAY_HALF     1

    //Value of delay time of TL & TH [us] (Full = (1,000,000 /FrequencyOfTargetClock)/2-4)
    #define I2C_DELAY_FULL     2

    // After one byte read or write, The waiting time is needed.
    // Because slave must analyze the receive data in I2C RX interrupt routine or
    // process another higher priority interrupt routine (ex:Timer-0)
    // The length of waiting time is depend slave MPU clock and Firmware
    #define I2C_DELAY_BYTE     100 //Value of waiting time after byte-writing or
reading(Min=60us)
    #define I2C_DELAY_SETUP 1      //Value of setup time for Start/Stop bits

#endif
```

Sample code (3/8)

```
int I2C_ReadCO2Sensor(void);
int I2C_SendRCommand(void);
int I2C_Read7Bytes(void);
void I2C_SendByte(unsigned char data);
unsigned char I2C_ReceiveByte(void);
void I2C_SendStart(void);
void I2C_SendAck(void);
unsigned char I2C_ReceiveAck(void);
void I2C_SendStop(void);
unsigned char I2C_ReceiveAck(void);
void Delay_ms_01(unsigned int time_ms);
void Delay_us_01(unsigned int time_us);

//----------------------------
void main(void)
   {

    //.....

    while(1)
        {

        //.....

        I2C_ReadCO2Sensor();

        //.....

        // Display_giCO2PPM();

        //.....

        }
    }
```

```
//----------------------------
int    giPreReadI2CFlag=0;    //Backup variable of PPM Data

int I2C_ReadCO2Sensor(void)
    {
    unsigned    char    iRet;

    //The period of read is 200msec.
    //This time can be changable by customer.
    if((giPreReadI2CFlag!=giReadI2CFlag)&&(giReadI2CFlag!=0))
        {
        giPreReadI2CFlag=giReadI2CFlag;
        if(giReadI2CFlag>=2){giReadI2CFlag=0;}else{;}
        }else{
            giPreReadI2CFlag=giReadI2CFlag;
            return(0);
            }

    #ifdef I2C_DEBUGCOUNT
        giI2cCall++;//DEBUGONLY
        if(giI2cCall>=100)
            {
            giI2cCall=1;//DEBUGONLY
            }else{;}
        if(giI2cNack>=100)
            {
            giI2cNack=0;//DEBUGONLY
            }else{;}
    #endif
```

Sample code (4/8)

```
    #ifdef USE_S100_SERIES //For S-100

        if(giPreReadI2CFlag==1)
            {
            //In the S-100 series, The 'R'-command packet is must sent at every times
            //before data bytes reading.
            iRet=I2C_SendRCommand();
            if(iRet==0x00){return(0);}else{;}
            return(1);
            }

        //* In here, The waiting time is too long!(100msec)
        //* So I'd like to recommend you to time-scheduler type
        //* I impleneted the time-scheduler using giPreReadI2CFlag and
        //  giReadI2CFlag on this source code.
        //Delay_ms_01(100); //Internal processing time for S-100 : Test only

        else{
            iRet=I2C_Read7Bytes();
            if(iRet==0x00){return(0);}else{;}
            return(7);
            }

    #else    //For S-200
        //The 'R'-command packet is must sent at every times before data bytes
reading.
        iRet=I2C_SendRCommand(); //Excution time : 1.1msec
        if(iRet==0x00){return(0);}else{;}

        // No waiting time in here for S-200

        iRet=I2C_Read7Bytes();    //Excution time : 4msec
        if(iRet==0x00){return(0);}else{;}
        return(7);
    #endif

    }
```

```
//---------------------------
int I2C_SendRCommand(void)
    {
    unsigned    char    iRet;

    //Send 'R'-Command to Sensor
    I2C_SendStart();
    I2C_SendByte(0x62); //Write Mode & Slave Address=0x31
    iRet=I2C_ReceiveAck();
    #ifdef USE_S100_SERIES
        Delay_us_01(I2C_DELAY_ATMEL_ISR);
    #endif
    if(iRet!=0x00)
        {
        #ifdef I2C_DEBUGCOUNT
            giI2cNack++;//DEBUGONLY
        #endif
        giReadI2cCount=0;I2C_SendStop();
        return(0);
        }else{;}
    Delay_us_01(I2C_DELAY_BYTE);

    I2C_SendByte(0x52);    //Send 'R' Command to Slave
    iRet=I2C_ReceiveAck();
    if(iRet!=0x00)
        {
        #ifdef I2C_DEBUGCOUNT
            giI2cNack++;//DEBUGONLY
        #endif
        giReadI2cCount=0;
        I2C_SendStop();
        return(0);
        }else{;}

    I2C_SendStop();
    Delay_us_01(I2C_DELAY_BYTE);

    return(1);

    }
```

Sample code (5/8)

```
//----------------------------
int I2C_Read7Bytes(void)
    {
    unsigned    char     LP01;
    unsigned    char    iRet;
    unsigned    char    TmpBuff[8];
    unsigned    short     CO2Density;
    unsigned    char    MaxRD;

    //Read PPM Data from Sensor
    I2C_SendStart();
    I2C_SendByte(0x63); //Read Mode & Slave Address=0x31
    iRet=I2C_ReceiveAck();
    if(iRet!=0x00)
        {
        #ifdef I2C_DEBUGCOUNT
            giI2cNack++;//DEBUGONLY
        #endif
        giReadI2cCount=0;I2C_SendStop();
        return(0);
        }else{;}

    Delay_us_01(I2C_DELAY_BYTE);

    MaxRD=7;

    for(LP01=0;LP01<MaxRD;LP01++)
        {
        iRet=I2C_ReceiveByte();TmpBuff[LP01]=iRet;
        I2C_SendAck();
        Delay_us_01(I2C_DELAY_BYTE);
        }
    I2C_SendStop();
    Delay_us_01(I2C_DELAY_BYTE);
```

```
//Parse the Received packet
    if((TmpBuff[0] != 0x08)|| //Header Byte
    (TmpBuff[3] == 0xFF)||    //Reserved Byte-1
    (TmpBuff[4] == 0xFF)||    //Reserved Byte-2
    (TmpBuff[5] == 0xFF)||    //Reserved Byte-3
    (TmpBuff[6] == 0xFF))     //Reserved Byte-4
        {
        #ifdef I2C_DEBUGCOUNT
            giI2cNack++;//DEBUGONLY
        #endif
        giReadI2cCount=0;
        return(0);
        }else{;}

    //Save data
    CO2Density=0x00;
    CO2Density=TmpBuff[1]<<8;
    CO2Density|=TmpBuff[2];

    giCO2PPM=CO2Density; //PPM data is saved to global variable.

    return(7);
    }
```

Sample code (6/8)

```
//----------------------------
void I2C_SendByte(unsigned char data)
    {
    unsigned char     i;
    unsigned char iMask;
    iMask=0x80;
    PIN_I2C_CLK = LOW;
    SDA_DIRECT = OUTPUT;

    for(i = 0;i < 8;i++)
        {
        PIN_I2C_CLK = LOW;
        Delay_us_01(I2C_DELAY_HALF); //ADuc848: TL/2=1us (TL>1.3us)
        if((data & iMask)!=0x00)
            {
            PIN_I2C_DAT = HIGH;
            } else{
                PIN_I2C_DAT = LOW;
                }
        Delay_us_01(I2C_DELAY_HALF); //ADuc848: TL/2=1us (Tdsu>100nS, TL>1.3us)
        PIN_I2C_CLK = HIGH;
        iMask>>=1;
        Delay_us_01(I2C_DELAY_FULL); //ADuc848: Th=2us (Min=0.6us)
        }
    PIN_I2C_CLK = LOW;
    SDA_DIRECT    = INPUT;    //ADuC848: Tdhd<9usec , ATmega16 : Thd;dat<3.45us
    Delay_us_01(I2C_DELAY_FULL); //ADuc848: TL/2=1us (Tdhd>0.9us, TL>1.3us)

    #ifdef USE_S100_SERIES
        Delay_us_01(I2C_DELAY_ATMEL_ISR); //For Atmega ISR of I2C
    #endif

    }
```

```
//----------------------------
unsigned char I2C_ReceiveByte(void)
    {
    unsigned char i;
    unsigned char iMask;
    unsigned char iRet;
    iRet=0x00;
    iMask=0x80;

    PIN_I2C_CLK = LOW;
    SDA_DIRECT = INPUT;
    Delay_us_01(I2C_DELAY_FULL); //ADuc848: TL=2us (Tdsu>100nS, TL>1.3us)

    for(i = 0;i < 8;i++)
        {
        PIN_I2C_CLK = HIGH;
        Delay_us_01(I2C_DELAY_HALF); //ADuc848: TL/2=1us (Tdsu>100nS, TH>0.6us)
        if(SDA_INPUT!=0x00)
            {
            iRet|=iMask;
            }else{;}
        iMask>>=1;
        Delay_us_01(I2C_DELAY_HALF); //ADuc848: TL/2=1us (Tdsu>100nS, TH>0.6us)
        PIN_I2C_CLK = LOW;
        Delay_us_01(I2C_DELAY_FULL); //ADuc848: TL=21us (Tdhd>0.9us, TL>1.3us)
        }

    #ifdef USE_S100_SERIES
        Delay_us_01(I2C_DELAY_ATMEL_ISR); //For Atmega ISR of I2C
    #endif

    return(iRet);
    }
```

Sample code (7/8)

```
//----------------------------
void I2C_SendStart(void)
    {

    SDA_DIRECT    = OUTPUT;
    PIN_I2C_DAT = HIGH; //ADuc848: 250nS (one instruction time)
    PIN_I2C_CLK = HIGH; //ADuc848: 250nS
    Delay_us_01(I2C_DELAY_FULL); //ADuc848: Tbuf=2+0.25 us (Min : 1.3us), It's also used for Trsu
    PIN_I2C_DAT     = LOW;
    Delay_us_01(I2C_DELAY_SETUP); //ADuc848: Tshd=1+0.25 us (Min : 0.6us)
    PIN_I2C_CLK     = LOW;
    Delay_us_01(I2C_DELAY_FULL); //ADuc848: TL=2+0.25 us (Min : 1.3 us)

    #ifdef USE_S100_SERIES
        Delay_us_01(I2C_DELAY_ATMEL_ISR); //For Atmega ISR of I2C
    #endif

    }

//----------------------------
void I2C_SendAck(void)
    {
    PIN_I2C_CLK = LOW;
    SDA_DIRECT = OUTPUT;
    PIN_I2C_DAT = LOW;
    Delay_us_01(I2C_DELAY_FULL); //ADuc848: TL=2us (Tdsu>100nS, TL>1.3us)
    PIN_I2C_CLK = HIGH;
    Delay_us_01(I2C_DELAY_FULL); //ADuc848: Th=2us (Min=0.6us)
    PIN_I2C_CLK = LOW;
    SDA_DIRECT = INPUT;     //ADuC848: Tdhd<9usec , ATmega16 : Thd;dat<3.45us
    Delay_us_01(I2C_DELAY_FULL); //ADuc848: TL=2us (Tdhd>0.9us, TL>1.3us)

    #ifdef USE_S100_SERIES
        Delay_us_01(I2C_DELAY_ATMEL_ISR); //For Atmega ISR of I2C
    #endif

    }
```

```
//----------------------------
unsigned char I2C_ReceiveAck(void)
    {
    unsigned char iRet;
    iRet=0x00;

    PIN_I2C_CLK = LOW;
    SDA_DIRECT = INPUT;
    Delay_us_01(I2C_DELAY_FULL); //ADuc848: TL=2us (Tdsu>100nS, TL>1.3us)

    PIN_I2C_CLK = HIGH;
    Delay_us_01(I2C_DELAY_HALF); //ADuc848: TL/2=1us (Tdsu>100nS, TH>0.6us)
    if(SDA_INPUT!=0x00)
        {
        iRet=1;
        }else{;}
    Delay_us_01(I2C_DELAY_HALF); //ADuc848: TL/2=1us (Tdsu>100nS, TH>0.6us)
    PIN_I2C_CLK = LOW;
    Delay_us_01(I2C_DELAY_FULL); //ADuc848: TL=21us (Tdhd>0.9us, TL>1.3us)

    #ifdef USE_S100_SERIES
        Delay_us_01(I2C_DELAY_ATMEL_ISR); //For Atmega ISR of I2C
    #endif

    return(iRet);
    }
```

Sample code (8/8)

```
//----------------------------
void I2C_SendStop(void)
    {
    PIN_I2C_DAT      = LOW;
    SDA_DIRECT     = OUTPUT;
    Delay_us_01(I2C_DELAY_HALF); //ADuc848: TL/2=1us (Tdsu>100nS, TL>1.3us)
    PIN_I2C_CLK      = HIGH;
    Delay_us_01(I2C_DELAY_SETUP); //ADuc848: Tpsu=1+0.25 us (Min : 0.6us)
    PIN_I2C_DAT      = HIGH;
    Delay_us_01(I2C_DELAY_FULL); //ADuc848: Tbuf=2+0.25 us (Min : 1.3us)

    #ifdef USE_S100_SERIES
        Delay_us_01(I2C_DELAY_ATMEL_ISR); //For Atmega ISR of I2C
    #endif
    SDA_DIRECT     = INPUT;

    }

//----------------------------
void Delay_ms_01(unsigned int time_ms)/* time delay for ms */
   {
    unsigned int i;

    for(i = 0; i < time_ms; i++)
        {
        Delay_us_01(250);
        Delay_us_01(250);
        Delay_us_01(250);
        Delay_us_01(250);
        }
   }

//----------------------------
void Delay_us_01(unsigned int time_us)/* time delay for us */
   {
    unsigned int i;

    for(i = 0; i < time_us; i++)// 4 cycle +
        {
        asm (" PUSH     R0 ");// 2 cycle +
        asm (" POP      R0 ");// 2 cycle +
        }
   }
```