I2C-FRTC User's Guide

The I2C-FRTC is an add-on module that provides the Inter-Integrated Circuit – IIC, also commonly known by the acronym I^{2}C or I2C bus, to a Nano-10, a FMD88-10, or a FMD1616-10 PLC. Please refer to the I2C specifications of your device for detailed explanation of the I2C protocol.

The PLC only supports the I2C as a master and operates at 100KHz, which allows it to connect to many off-the-shelve components such as GPS, accelerometers, thermometer, analog and digital I/O chips, etc. The PLC can connect to multiple I2C slaves in a multi-drop I2C bus, which greatly expands its capability. The built-in TBASIC commands also greatly simplify the I2C communication with the slave devices.

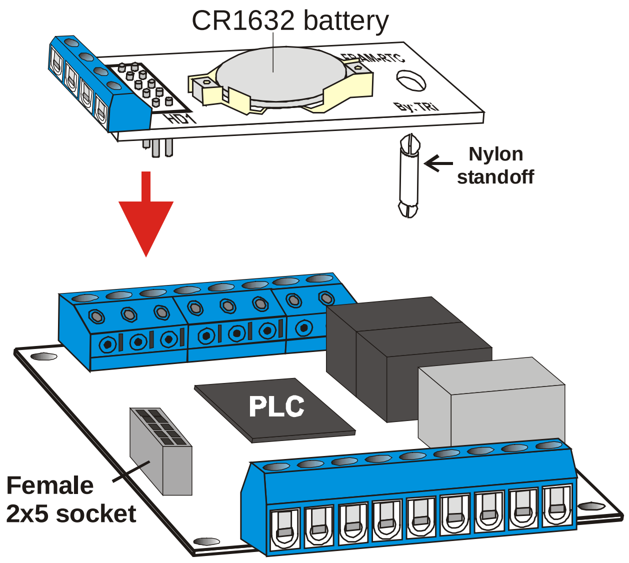
However, you can only use the I2C communication capability provided your Nano or FMD PLC meets **all** the following conditions:

1) You have installed the I2C interface module (such as the I2C-FRTC) on the PLC.

2) You have upgraded your I-TRiLOGI software to version 6.40 or later.

3) The PLC firmware is >= r74

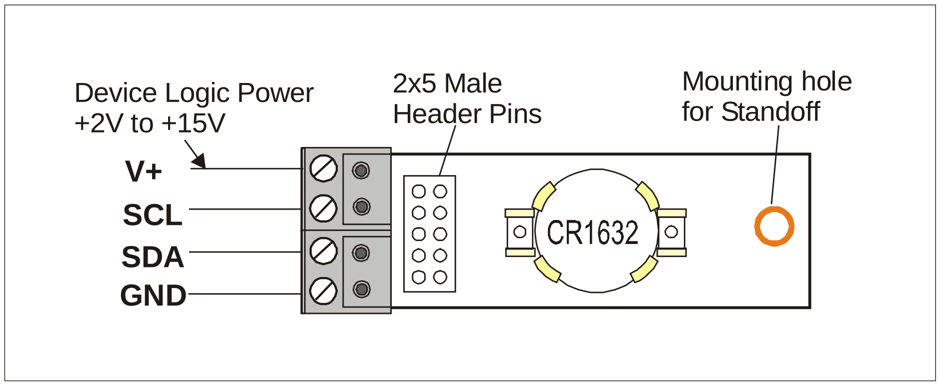
1 Installing the I2C-FRTC Module



To install the I2C-FRTC module, first **ensure that you have TURNED OFF** power to the PLC.

The I2C-FRTC module has a row of 2x5 male header pins that is to be inserted into the single mating 2x5 female socket on the PLC. Please ensure that the pins are aligned correctly with the socket. There is a single mounting hole on other end of the I2C-FRTC module, which provides support to the module via a nylon standoff included in the I2C-FRTC package. You should also dfind a matching hole on the PLC which is aligned with the mounting hole on the I2C-FRTC module. The nylon standoff has two supporting catches that will mate to the mounting holes on both the I2C-FRTC and the PLC and it provides a fairly strong support to the I2C-FRTC module.

2 I2C-FRTC Hardware Overview



The I2C-FRTC modules adds the following hardware to the Nano-10 or FMD PLC

1) I^{2}C communication interface chip

2) 11K words of FRAM memory, Expand program memory to 16K words, DM[1001] to DM[4000] and a Battery-backed Real Time Clock (RTC) *

3) **128K** bytes of I2C EEPROM memory (M24M01) – Expandable to 256K bytes by soldering an additional M24M01 chip next to it on the blank solder pad.

4) Additional Analog output channel #3 and #4 (0-5V only)

* The FRAM memory and battery-backed RTC on the I2C-FRTC is identical to that found on the FRAM-RTC module. i.e. I2C-FRTC = FRAM-RTC + additional I2C hardware.

The I2C interface chip allows the PLC to interface to external I2C devices that are of different logic voltage level from the PLC. You must connect the positive logic voltage of the target device to the "V+" terminal shown in the above diagram and 0V of the target device to the "GND" terminal. Then connect the SCL and SDA signal between the I2C-FRTC module and target device and you are good to go.

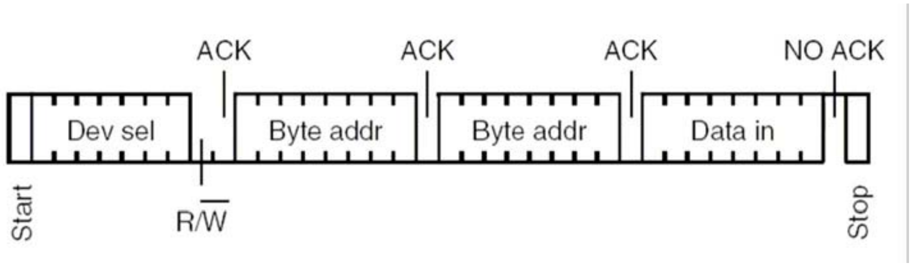
A 1 M bits I2C EEPROM chip (M24M01) is also included in the I2C-FRTC module. This allows you to use the new I2C_READ and I2C_WRITE command (available only in I-TRiLOGI version 6.40 or later) to store and retrieve up to **128K bytes** of non-volatile EEPROM memory to store additional data. This will be described in the next section.

3 New TBASIC Commands: I2C_READ, I2C_WRITE and I2C_STOP

These 3 new TBASIC commands are only available on i-TRiLOGI version **6.40** and above, and they are only enabled on the Nano or FMD PLCs that are installed with I2C-FRTC. If you are still running the older version of i-TRiLOGI, you can get a free update by clicking on the "Help" menu on your production version of I-TRiLOGI software and follow the "Upgrade TRiLOGI" link to download the latest I-TRiLOGI software in order to use these 3 newly added commands.

Both I2C_WRITE and I2C_READ commands use a range of data memory DM[ ] to transmit the data to be written into the device or to be read from the device. The parameters comprise the I2C slave address, the starting index of the DM[] memory location to use and the number of bytes to be sent/received from the slave.

3.1 I2C_WRITE



An **I2C_WRITE** command begins with the master (PLC) sending the START bit, followed by a 7-bit slave address, and then a "R/W" bit set to low, which indicates that it is a WRITE command. If the slave device with the targeted slave address is present, it will send the ACK response to the master on the 9th clock cycle. Otherwise the master sends a STOP bit and quits the I2C_WRITE function.

If the slave does send the ACK bit, the master will then send out a number of data bytes to be written to the slave and the slave will respond with the ACK bit with the completion of each byte it received. After the last data byte has been written to the slave, and if the master is not expecting to read any data from the slave, the master **must** then immediately send the **STOP** bit by executing the **I2C_STOP** command (to be described later) to indicate the End-of-Write to the slave.

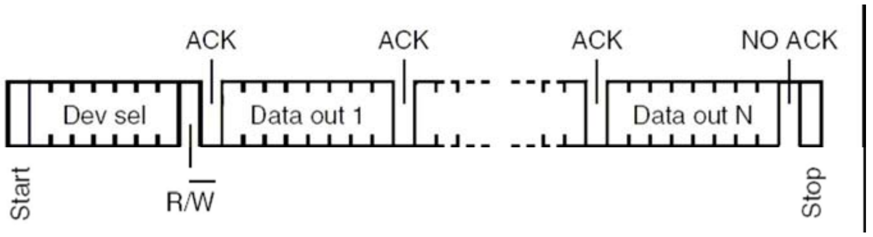
The PLC program can determine if the I2C_WRITE is successful by checking with the STATUS(2) command. The syntax of the I2C_WRITE command is as follow:

I2C_WRITE *i2cslave, dmstart, count*

Purpose	Special command to execute a I2C WRITE out of the PLC's I2C port (if so equipped). The CPU will send a I2C START, followed by the slave address byte (*i2cslave*) and *count* number of data bytes from the DM[*dmstart*] up to DM[*dmstart+count-1*] ***i2cslave -*** The 7-bit slave address that the CPU is writing to. ***dmstart -*** The starting index of the DM[ ] that contains the first data byte ***count -*** number of byte data to send (maximum is dependent on the slave).
Examples	DM[5]= 12: DM[6]= 34 : DM[7]= 56 **I2C_WRITE &H60,5,3** I2C_STOP

Comments	*Following the START bit, the CPU will write the 7-bit slave address &H60 (=110 0000 binary) and a R/W bit set to 0, followed by the byte data stored in DM[5], DM[6] and DM[7].* *The command automatically checks for ACK received from the slave device , and the user program can check the status of this operation by testing the STATUS(2) function. STATUS(2) returns a 1 if ACK is received , and 0 if no ACK is received after time out.* *Note: This command **does not** automatically generate the I2C STOP bit, this is to allow the CPU to perform a I2C_READ following a I2C_WRITE. I2C READ after WRITE is commonly encountered in I2C protocol which requires using the I2C_WRITE to set the internal pointer address in the slave device and then followed by the I2C_READ command.* *Therefore, if your command involves only I2C_WRITE, you must end the WRITE command by executing a I2C_STOP statement.*

3.2 I2C_READ



An **I2C_READ** command begins with the master (PLC) sending the START bit, followed by a 7-bit slave address, and then a “R/W” bit set to high, which indicates that it is a READ command. If the slave device with the targeted slave address is present, it will send the ACK response to the master. Otherwise the master sends a STOP bit and quit the I2C_WRITE function.

If the slave does send the ACK bit, the master will then toggle the SCL (clock) signal and the slave will send the data byte one bit at a time in response to the SCL pulses. After an 8-bit byte has been received, the master will automatically send the ACK bit to the slave and the slave will continue to send the next byte sequentially out to the master.

After the last data byte has been read from the slave, the master will not send the ACK bit but will automatically send the **STOP** bit to the slave. This indicates the End-of-Read to the slave and the communication is complete.

I2C_READ *i2cslave, dmstart, count*

Purpose	Special command to execute a I2C_READ out of the PLC's I2C port (if so equipped). The CPU will send a I2C START bit, followed by the slave address byte (*i2cslave*) with "R/W" bit set to high, and then send out the number of clock pulses required to read *count* number of data bytes from the the slave.

	The data bytes received from the slave will be stored in the memory location DM[*dmstart*] to DM[*dmstart+count-1*]. After receiving all the required data bytes the CPU automatically send the I2C **STOP** bit to the slave to end the communication. ***i2cslave -*** The 7-bit slave address that the CPU is reading from. ***dmstart -*** The starting index of the DM[ ] that is to receive the first data ***count -*** number of byte data bytes to read from the slave.
Examples	`I2C_READ &H0C,21,2   ' read 2 bytes into DM[21] and DM[22]`
Comments	*After sending the START bit, the CPU will write the 7-bit slave address &H60 (=110 0000 binary) and a R/W bit set to 1, followed by 16 clock pulses to read 2 bytes of data and store into DM[21] and DM[22], and then the CPU will generate the STOP bit.* *i.e. there is no need execute the I2C_STOP command after an I2C_READ command.*

3.3 I2C_STOP

This command has no parameter. It sends a STOP bit to the slave and completes the I2C_WRITE command.

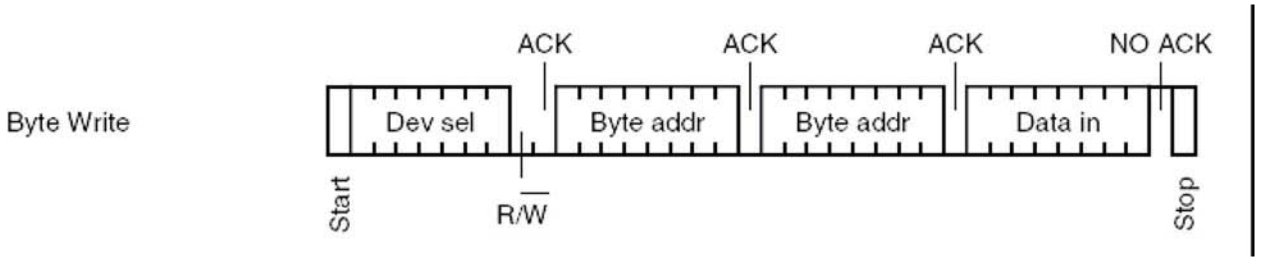
4 Using The I2C Commands To Access M24M01 EEPROM

4.1 Random Write To M24M01 EEPROM

The first M24M01 EEPROM on the I2C-FRTC (U2) has two binary slave device addresses: 101 0000 b (&H50) and 101 0001b (&H51). Device address &H50 is for accessing the first bank of 64K bytes of EEPROM, and address &H51 is for accessing the second bank of 64K bytes of EEPROM.

There is also a blank solder pad on the bottom layer of the I2C-FRTC module, which allows you to solder an additional M24M01 (U3) to the I2C-FRTC PCB. When assembled this second M24M01 chip will assume the binary address of 101 0010b (&H52) and 101 0011b (&H53). Device address &H52 on U3 is for accessing the first bank of 64K bytes while device address &H53 is for accessing the second bank.

Please refer to the M24M01 EEPROM data sheet for the detailed description of the addressing scheme for writing a byte of data to a random EEPROM address. The following picture depict the necessary command:



Example. To write a byte of data XX to the EEPROM address 54321 (&HD431) in first 64K bank, you need to do the following:

```
   DM[11] = &HD4
   DM[12] = &H31
   DM[13] =  xx     ‘ your data byte

   I2C_WRITE &H50, 11, 3   ‘ write 3 bytes of data from DM[11] to DM[13]
   I2C_STOP                ‘ necessary to end the byte write.
```

The data XX will be written to the EEPROM address 54321

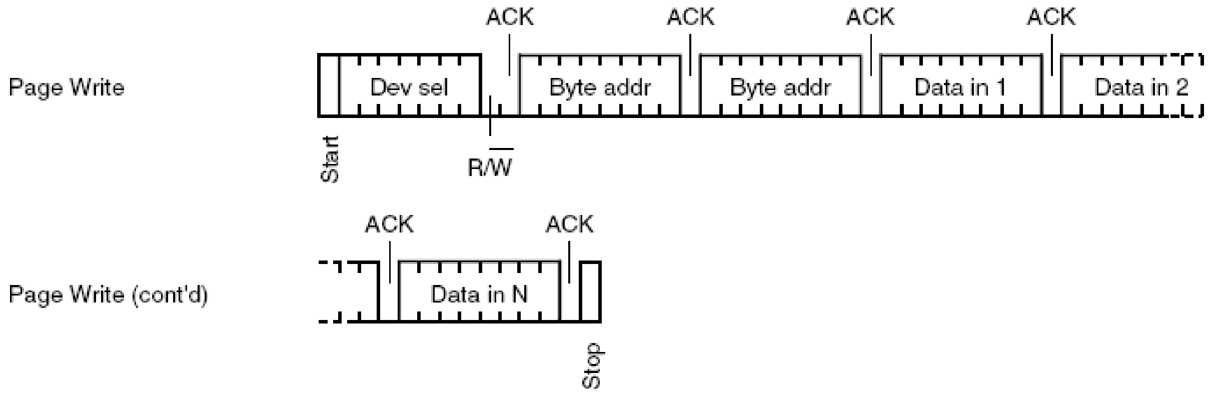
If you want to store the data to second bank of EEPROM address, then replace the I2C_WRITE line with:

```
   I2C_WRITE &H51, 11, 3
```

4.2 Page Write To M24M01 EEPROM

As you can see, writing a single byte of data to a random location involves 4 bytes of data transfer, which is not very efficient. Fortunately, the EEPROM allows you to write more than one byte to the EEPROM and the EEPROM will write to the subsequent location sequentially. This is known as “Page Write” and you can write up to 256 bytes in the same page. A page is defined as the memory location having the same upper address byte (bit 8 to bit 15). E.g. Address &HA011 and &HA0FF are in the same page. But address &HA0FF and &HA100 are **NOT** in the same page even though they are adjacent memory location. So you have to keep the page boundary in mind when performing a page write.

The following picture depict the page write command:



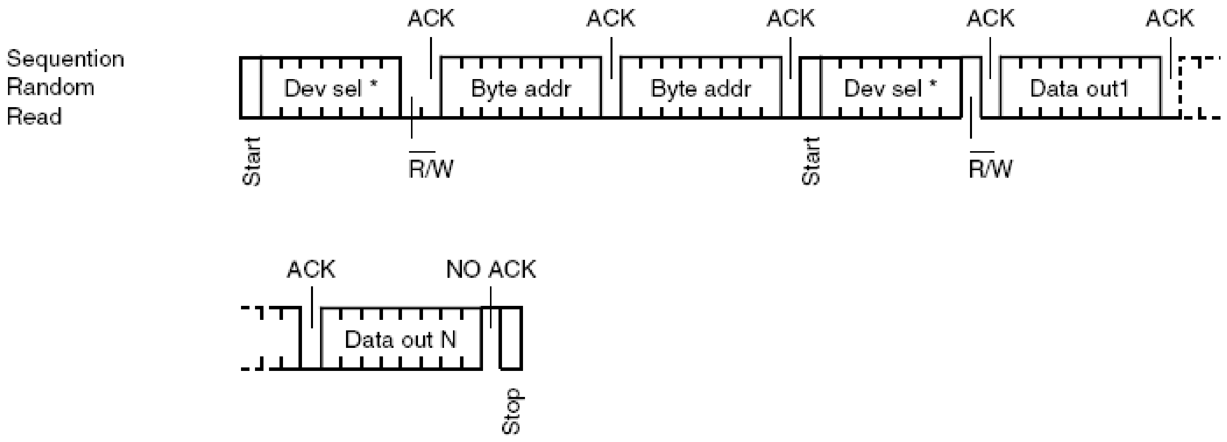
Example. To write 4 byte of data XX to the EEPROM address 19876 to 19879 (&H4DA4) in first 64K bank, you need to do the following:

```
DM[11] = &H4D
DM[12] = &HA4
DM[13] =  xx   ‘ your data byte 1
DM[14] =  yy   ‘ your data byte 2
DM[15] =  zz   ‘ your data byte 3
DM[16] =  ww   ‘ your data byte 4

I2C_WRITE &H50, 11, 6   ‘ write 6 bytes of data from DM[11] to DM[16]
I2C_STOP                ‘ necessary to end the write cycles.
```

The data contained in DM[13] to DM[16] will be written to the EEPROM address &H4DA4 to &H4DA7.

4.3 Random Read From M24M01 EEPROM



Reading data from a random EEPROM location is slightly more involved than writing. You need to first use the I2C_WRITE command to set the memory pointer inside the M24M01 to point to the memory address location, then immediately followed by I2C_READ command to read one or more data bytes starting from the pointer address. After every byte is read the internal pointer will be incremented automatically and point to the next address byte, this allows you to read a large number of data sequentially from the EEPROM with minimum overhead. This can be very useful for “data dump” to the TLServer to rapidly upload the collected data

Example. To Read 100 bytes from EEPROM address 12345 (&H3039) to 12444 in first 64K bank, you need to do the following:

```
DM[11] = &H30
DM[12] = &H39
I2C_WRITE &H50, 11, 2  ‘ write 2 bytes of address in DM[11] to DM[12]
I2C_READ &H50,21,100   ‘ read 100 bytes data into DM[21] to DM[120]
```

The returned data will be stored in the DM[21] to DM[120].

Note: There is no need to execute the I2C_STOP command after the I2C_READ since the I2C_READ command automatically sends a STOP bit after the last byte is read.

4.4 Sequential Read From M24M01 EEPROM

Note that after a random read, the memory pointer inside the M24M01 will be pointed to the next address following the very last read memory address. This means that you could repeatedly execute only the I2C_READ command to read more data sequentially from the EEPROM memory.

Example:

```
DM[11] = &H30
DM[12] = &H39
I2C_WRITE &H50, 11, 2   ‘ write 2 bytes of address in DM[11] to DM[12]
FOR I = 1 to 10
  I2C_READ &H50,21,100  ‘ read 100 bytes data into DM[21] to DM[120]
  CALL Datadump         ‘ call some subroutine to upload data to server.
NEXT
```

In the above example, the I2C_READ command was executed 10 times, each time 100 data point is read into DM[21] to DM[120] and the program then calls another custom function to dump these data points to the server. The loop then continue for another 9 times, and hence altogether 1000 data points from address 12345 to 13344 can be uploaded to the server in a simple FOR..NEXT loop.