

<https://www.halvorsen.blog>



Modbus

With Practical LabVIEW Examples

Hans-Petter Halvorsen

Contents

- Modbus
- Modbus in LabVIEW
- LabVIEW Examples
 - LabVIEW Coils Examples
 - LabVIEW Discrete Input Registers Examples
 - LabVIEW Input Registers Examples
 - LabVIEW Holding Registers Examples



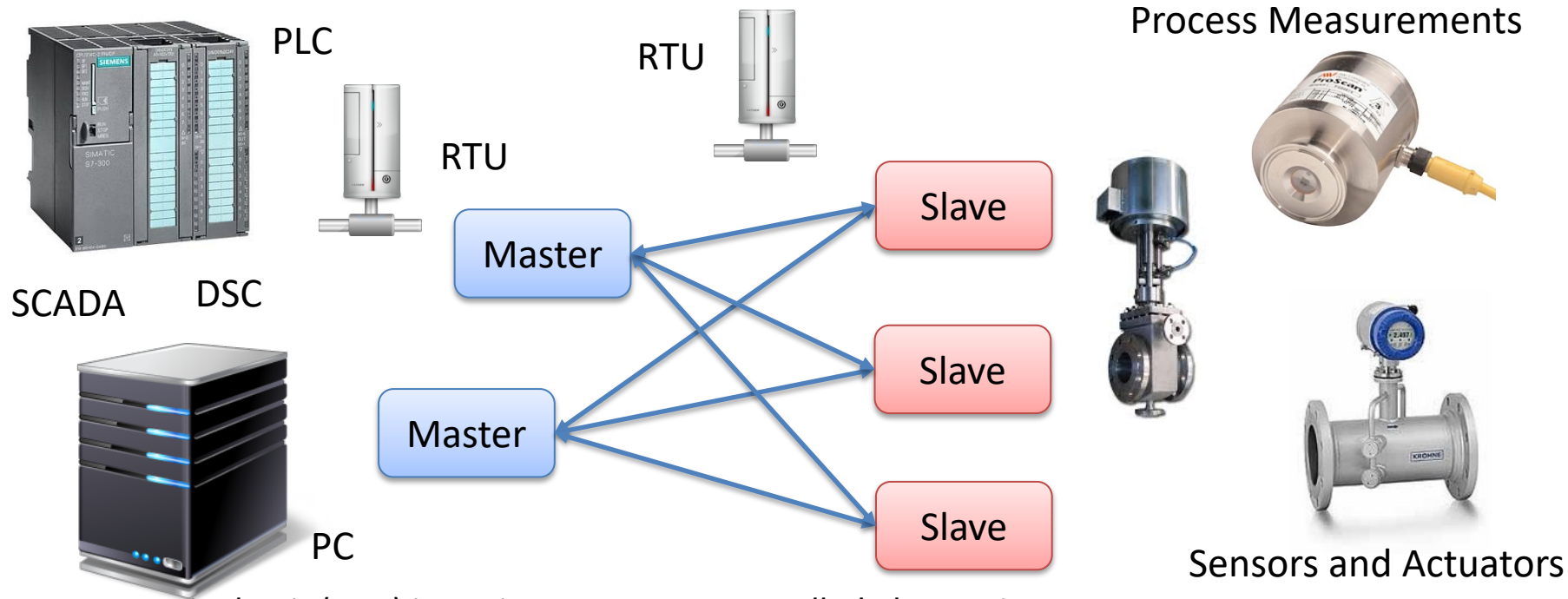
Modbus

What is Modbus?

- **Modbus is a serial communications protocol** originally published by Modicon (now Schneider Electric) in 1979 for use with its programmable logic controllers (PLCs).
- **Simple and robust**, it has since become a de facto standard communication protocol, and it is now a commonly available means of connecting industrial electronic devices
- The development and update of Modbus protocols has been managed by the Modbus Organization since April 2004, when Schneider Electric transferred rights to that organization (<https://modbus.org>)
- Modbus became the first widely accepted fieldbus standard.

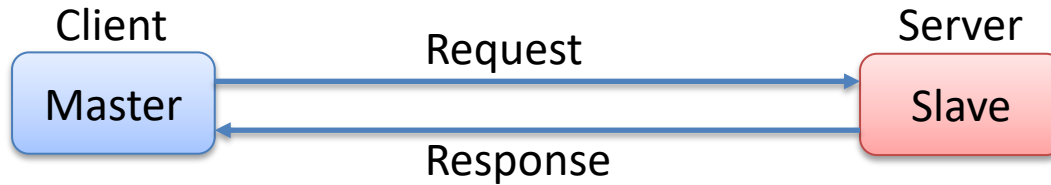
Modbus

The master typically is a PLC (Programmable Logic Controller), PC or DCS (Distributed Control System)



A remote terminal unit (RTU) is a microprocessor-controlled electronic device that interfaces objects in the physical world to a DCS or SCADA System

Master/Slave



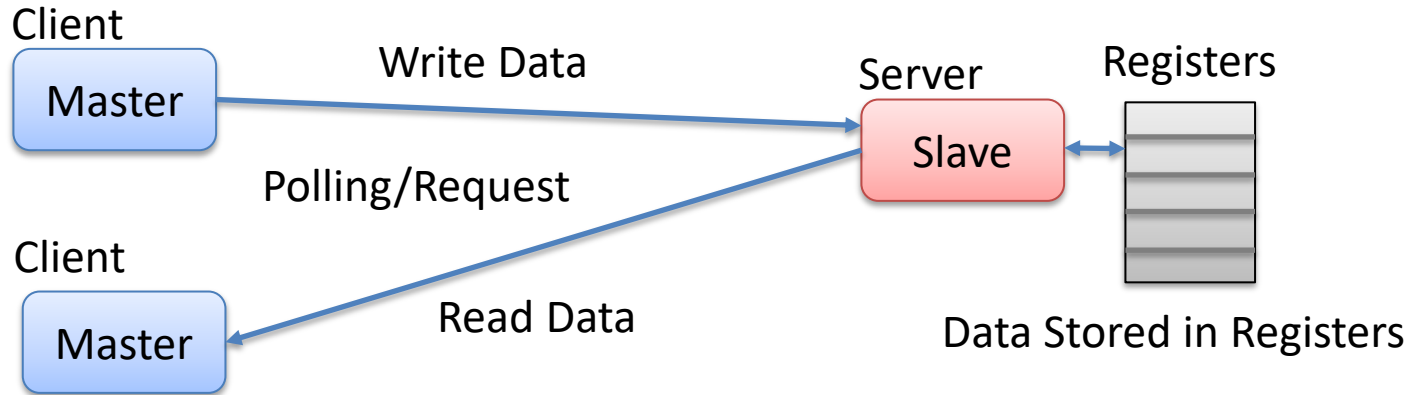
The Modbus protocol follows a Master/Slave (Client/Server) architecture where a Master (Client) transmits a request to a Slave (Server) and waits for the response.

Note! The terms “Master” and “Slave” used in Modbus has been replaced with the terms “Client” and “Server”. The LabVIEW Modbus package still use the old terms, so they will also be used in this Tutorial

Master/Slave

- Modbus protocol is defined as a master/slave protocol, meaning a device operating as a master will poll one or more devices operating as a slave.
- This means a slave device cannot volunteer information; it must wait to be asked for it.
- The master will write data to a slave device's registers and read data from a slave device's registers. A register address or register reference is always in the context of the slave's registers.

Master/Slave



Modbus protocol is defined as a master/slave protocol, meaning a device operating as a master will poll one or more devices operating as a slave. This means a slave device cannot volunteer information; it must wait to be asked for it. The master will write data to a slave device's registers and read data from a slave device's registers. A register address or register reference is always in the context of the slave's registers.

Modbus Register Types

- **Coil** (Discrete Output)
 - Coils are 1-bit registers, used to control discrete outputs, Read or Write
- **Discrete Input** (Read Only)
 - 1-bit registers
- **Input Register** (Read Only)
- **Holding Register** (Read/Write)

Access Levels

- In SCADA systems, it is common for embedded devices to have certain values defined as inputs, such as gains or proportional integral derivative (PID) settings, while other values are outputs, like the current temperature or valve position.
- To meet this need, Modbus data values are divided into four ranges
- In many cases, sensors and other devices generate data in types other than simply Booleans and unsigned integers.
- It is common for slave devices to convert these larger data types into registers. For example, a pressure sensor may **split a 32-bit floating point value across two 16-bit registers.**

Access Levels

| Register Type | Data Type | Master Access | Slave Access |
|-------------------------|---------------|---------------|--------------|
| Coils | Bit (Boolean) | Read/Write | Read/Write |
| Discrete Input | Bit (Boolean) | Read-only | Read/Write |
| Input Register | Unsigned Word | Read-only | Read/Write |
| Holding Register | Unsigned Word | Read/Write | Read/Write |

An **Unsigned Word** is a 16-bit nonnegative Integer Value between 0 – 65535 (2^{16})

Register Addresses

- 0x = **Coil**, Address Range: **00001-09999**
- 1x = **Discrete Input**, Address Range: **10001-19999**
- 3x = **Input Register**, Address Range: **30001-39999**
- 4x = **Holding Register**, Address Range: **40001-49999**

When using the extended referencing, all *number* references must be exactly six digits. This avoids confusion between coils and other entities. For example, to know the difference between holding register #40001 and coil #40001, if coil #40001 is the target, it must appear as #040001.

Register Referencing

40001:7

- This is a commonly used notation for referencing individual bits in a register.
- This example references register 40001 (which is a Holding Register), bit 7.
- Bits are generally numbered starting at bit 0, which is the least significant or right most bit in the field of 16 bits found in a Modbus register.

Modbus Protocols

- Modbus ASCII
- Modbus RTU (Remote Terminal Unit)
 - Modbus RTU uses RS-485 or RS-232
- **Modbus TCP/IP**
 - Modbus TCP uses Ethernet

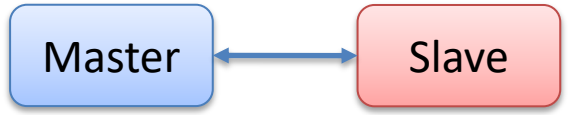
We will focus on Modbus TCP/IP in this Tutorial

Modbus ASCII and Modbus RTU are simple serial protocols that use RS-232 or RS-485 to transmit data packets.

Modbus TCP/IP follows the OSI Network Model and can be used in an ordinary Ethernet network

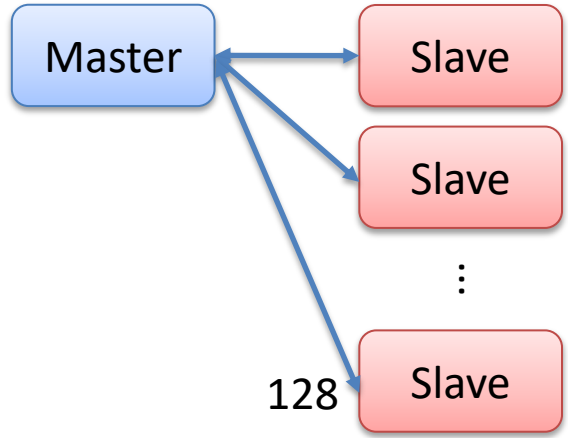
Modbus Communication

RS-232

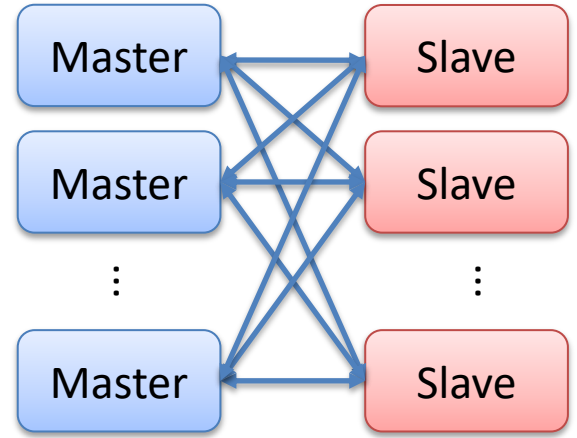


RS-485

Multi-drop network



TCP/IP



Modbus TCP/IP

- Modbus TCP/IP follows the OSI Network Model and can be used in an ordinary Ethernet network
- Modbus TCP requires that you know or define IP addresses on the network
- Modbus TCP/IP uses Port **502**



Modbus in LabVIEW

Modbus in LabVIEW

3 ways to use Modbus in LabVIEW:

- Use a high-level OPC Server
- Use Modbus I/O Server
- Use the LabVIEW Modbus API

“LabVIEW Real-Time Module” or “LabVIEW DSC Module” required

LabVIEW Modbus API

The image displays the LabVIEW Modbus API toolboxes, illustrating the navigation path from the main Data Communication toolbox to the specific Modbus Master and Modbus Slave toolboxes.

Data Communication Toolbox: This is the main toolbox containing various communication-related functions. The **Modbus** icon is highlighted with a red circle. Other visible icons include Shared Variables, Network Streams, Local Variables, Global Variables, Queue Operators, Synchronization, DataSockets, Protocols, Actor Frames, and EPICS.

Modbus Master Toolbox: This toolbox is accessed by clicking the Modbus icon in the Data Communication toolbox. It contains functions for creating and managing Modbus Master objects, such as **Create Master...**, **Close.vi**, and **Property Node**. It also includes functions for reading and writing data, such as **Read Holding Register...**, **Write Single...**, **Write Multiple...**, **Write and Read...**, **Mask Write...**, **Read Coils.vi**, **Write Single...**, **Write Multiple...**, **Read Discrete...**, **Read Input...**, and **Utilities**.

Modbus Slave Toolbox: This toolbox is accessed by clicking the Modbus Slave icon in the Modbus Master toolbox. It contains functions for creating and managing Modbus Slave objects, such as **Create Slave...**, **Close.vi**, and **Property Node**. It also includes functions for reading and writing data, such as **Read Holding Register...**, **Write Single...**, **Write Multiple...**, **Write and Read...**, **Mask Write...**, **Read Coils.vi**, **Write Single...**, **Write Multiple...**, **Read Discrete...**, **Write Multiple...**, **Read Input...**, **Write Multiple...**, and **Utilities**.

Red arrows indicate the navigation path: from the **Modbus** icon in the Data Communication toolbox to the **Modbus Master** toolbox, and from the **Modbus Slave** icon in the Modbus Master toolbox to the **Modbus Slave** toolbox.



LabVIEW Examples

Modbus LabVIEW Examples

- LabVIEW Coils Examples
- LabVIEW Discrete Input Registers Examples
- LabVIEW Input Registers Examples
- LabVIEW Holding Registers Examples

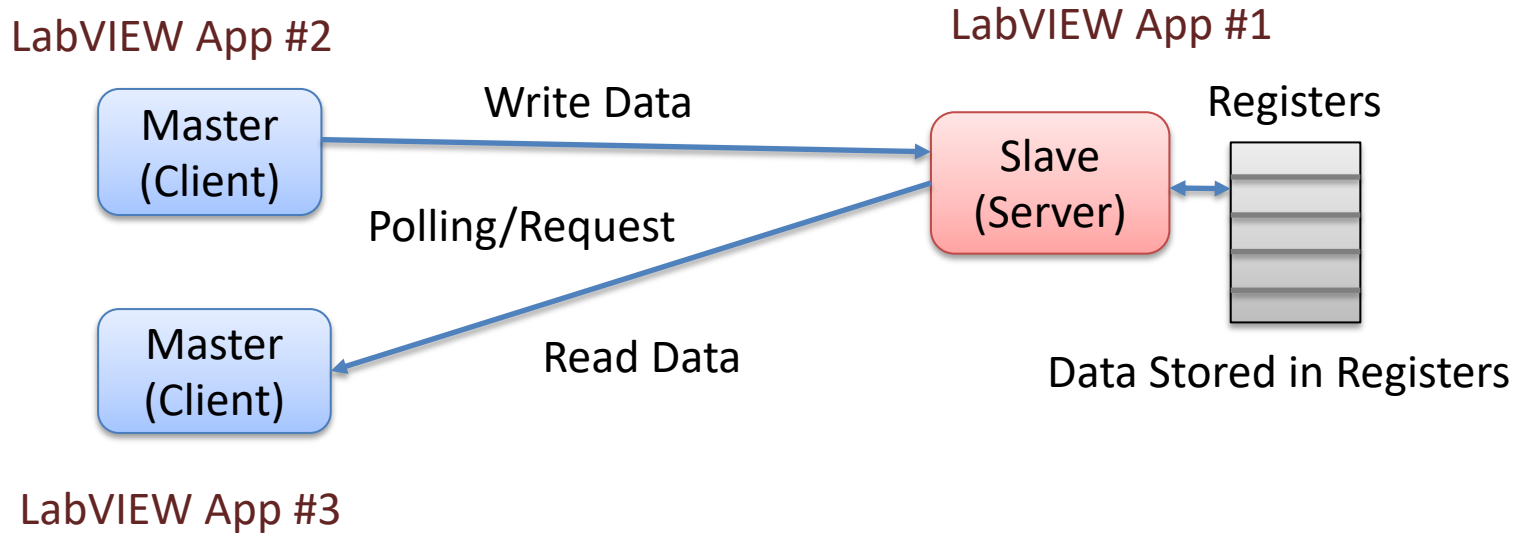
| Memory Type | Data Type | Master Access | Slave Access |
|------------------|---------------|---------------|--------------|
| Coils | Bit (Boolean) | Read/Write | Read/Write |
| Discrete Input | Bit (Boolean) | Read-only | Read/Write |
| Input Register | Unsigned Word | Read-only | Read/Write |
| Holding Register | Unsigned Word | Read/Write | Read/Write |



LabVIEW Coils Examples

LabVIEW Coils Example

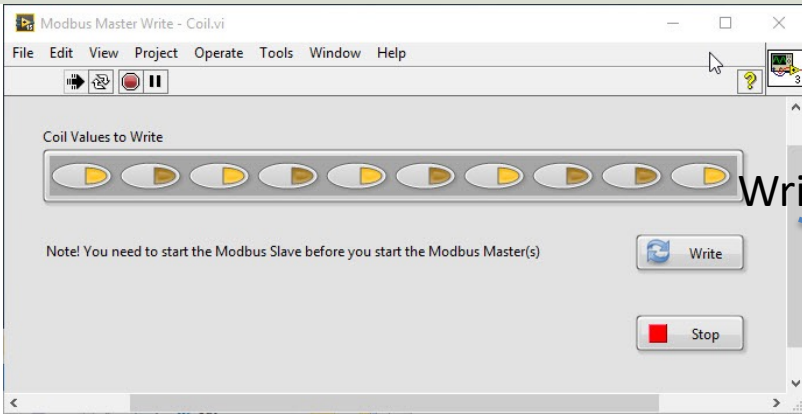
In this Example we Create 3 different LabVIEW Applications:



| Memory Type | Data Type | Master Access | Slave Access |
|-------------|---------------|---------------|--------------|
| Coils | Bit (Boolean) | Read/Write | Read/Write |

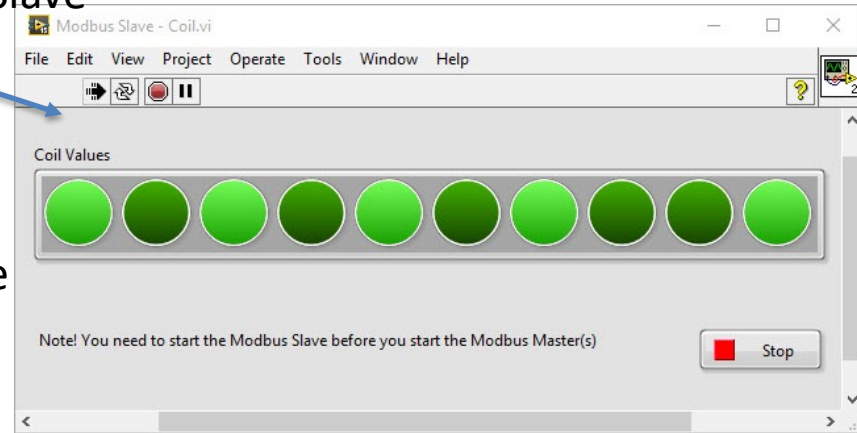
LabVIEW Coils Example

LabVIEW App #2 (Master)



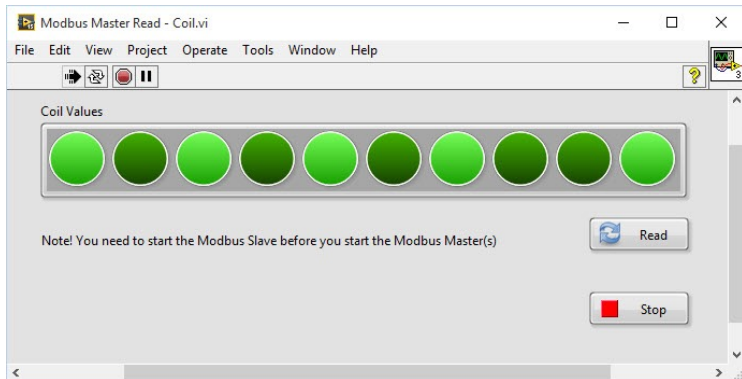
Write Data to Slave

LabVIEW App #1 (Slave)



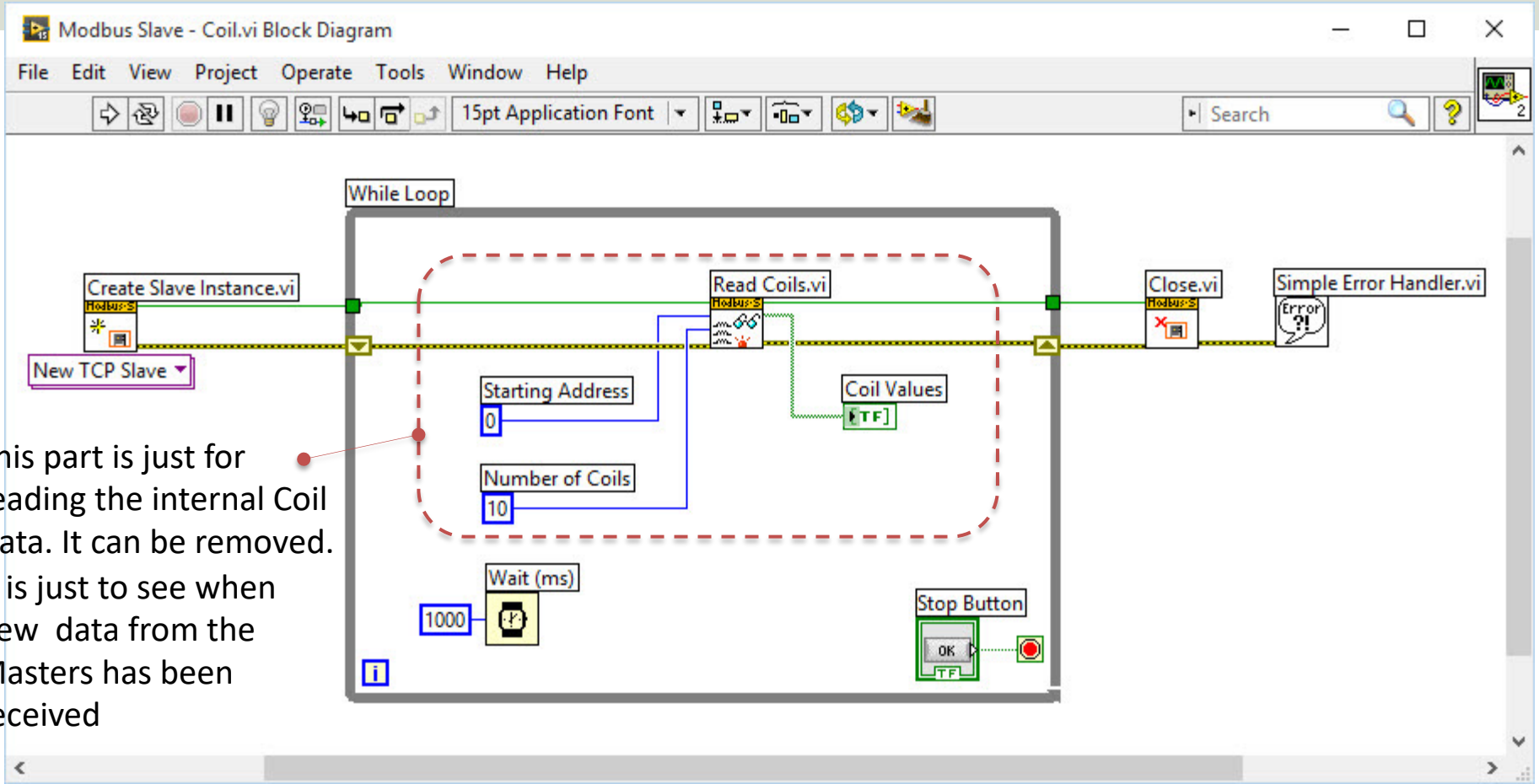
Read Data from Slave

LabVIEW App #3 (Master)



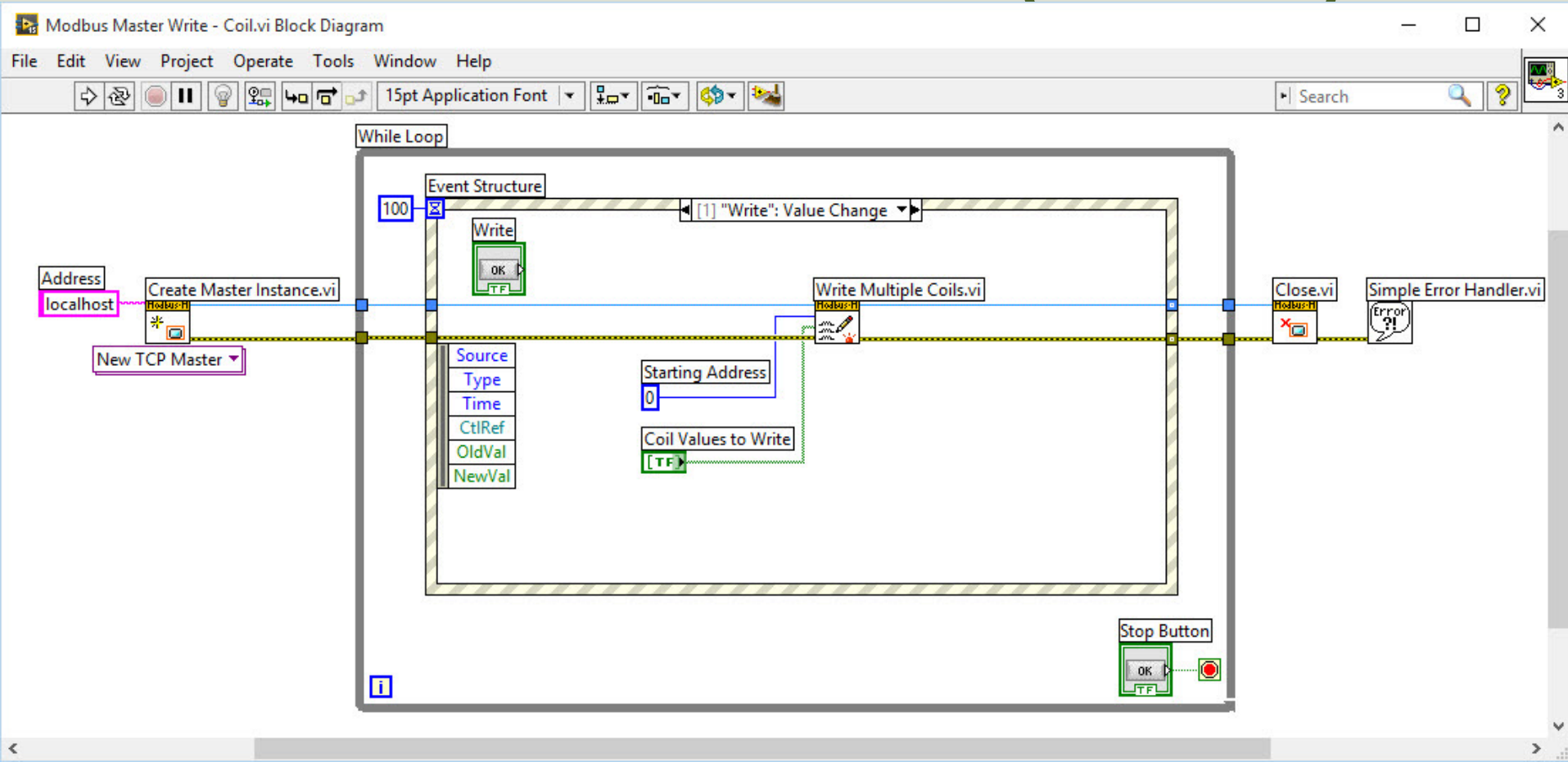
Note! You need to start/run the Modbus Slave App before you start the Modbus Master Apps

Modbus Slave

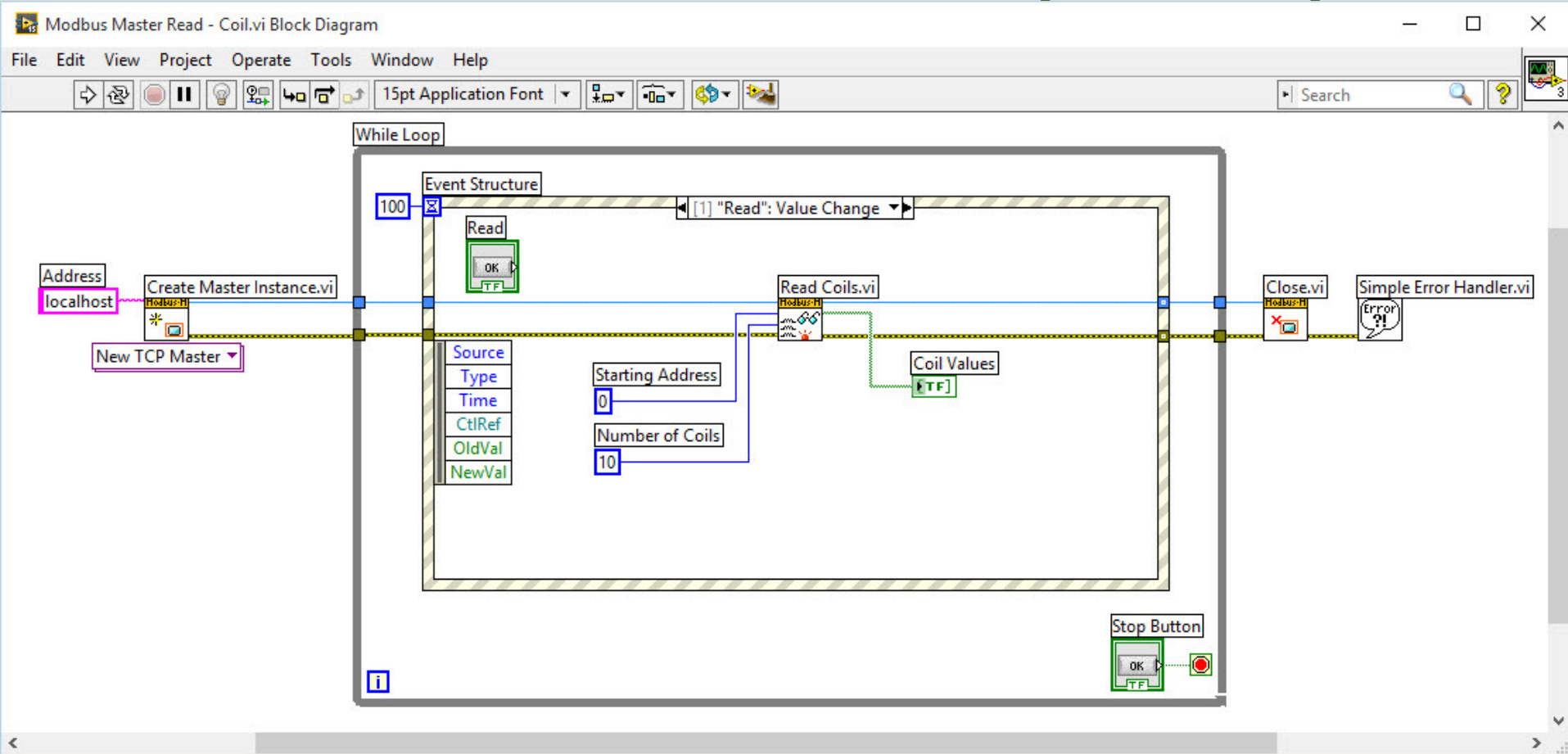


This part is just for reading the internal Coil Data. It can be removed. It is just to see when new data from the Masters has been received

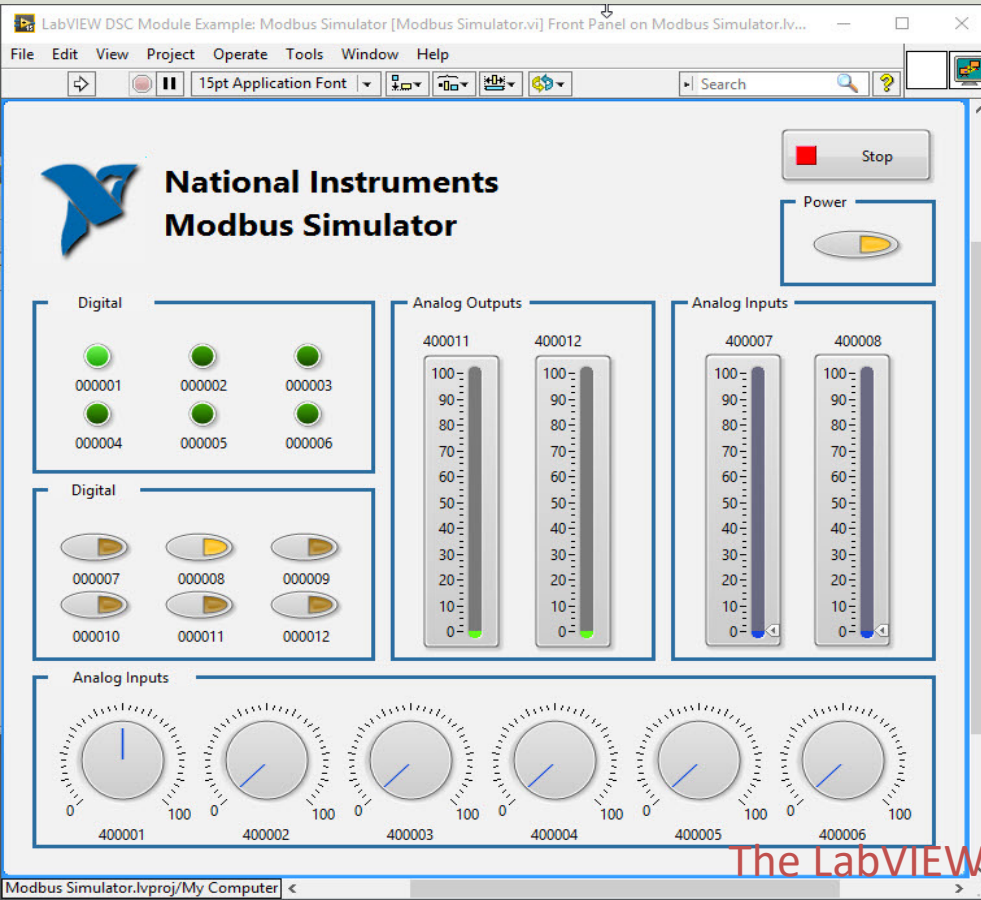
Modbus Master (Write)



Modbus Master (Read)



LabVIEW Modbus Simulator



The LabVIEW Modbus Simulator is integrated with “LabVIEW Real-Time Module” or “LabVIEW DSC Module”

It can be used for test purpose, etc.

The LabVIEW Modbus Simulator is a **Modbus Slave (Server)**

NI Example Finder

The screenshot shows the NI Example Finder application window. The window title is "NI Example Finder". It has two tabs: "Browse" and "Search". The "Search" tab is active. On the left, there is a search input field containing "modbus" and a "Search" button. Below it is a "Double-click keyword(s)" list with "Modbus" selected. At the bottom left, there is a "Hardware" dropdown menu set to "Find hardware" and a checkbox for "Limit results to hardware". The main area displays a list of search results under the heading "Double-click an example to open it." and "4 examples match your search criteria". The results are:


- Modbus Fundamentals.lvproj
- Modbus Simulator.lvproj (highlighted)
- Modbus Library.lvproj
- Redundant Modbus Masters.lvproj

On the right, the "Information" panel shows the "Description" for the selected example: "This LabVIEW example simulates a basic Modbus device. This example demonstrates how to use a Modbus slave to read and write data items by using LabVIEW shared variables and deploying and undeploying a project library programmatically. You can connect to this Modbus device by using a Modbus I/O server or a third-party Modbus client." Below the description, it states: "This example requires the LabVIEW Datalogging and Supervisory Control Module." The "Requirements" section is currently empty. At the bottom of the window, there are buttons for "Add to Favorites", "Setup...", "Help", and "Close".

Find Modbus
Examples with NI
Example Finder

LabVIEW Modbus Simulator Example

LabVIEW DSC Module Example: Modbus Simulator

 **National Instruments
Modbus Simulator**

Stop

Power



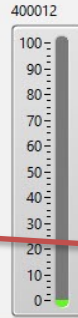
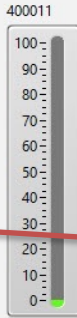
Digital

000001
000004

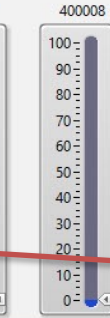
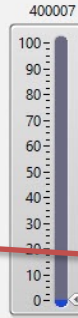
000002
000005

000003
000006

Analog Outputs



Analog Inputs



Digital

000007
000010

000008
000011

000009
000012

Analog Inputs



Modbus Master Write - Coil.vi

File Edit View Project Operate Tools Window Help



Coil Values to Write



Note! You need to start the Modbus Slave before you start the Modbus Master(s)

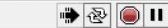
Write

Stop

Modbus Simulator.lvproj/My Computer

Modbus Master Read - Coil.vi

File Edit View Project Operate Tools Window Help



Coil Values



Note! You need to start the Modbus Slave before you start the Modbus Master(s)

Read

Stop

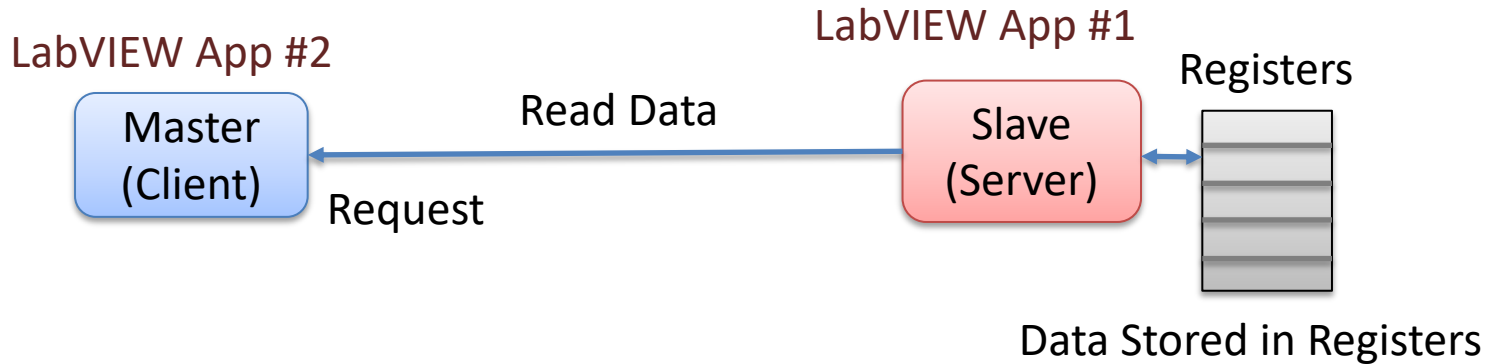
Modbus Simulator.lvproj/My Computer



LabVIEW Discrete Input Registers Examples

LabVIEW Discrete Input Registers Examples

In this Example we Create 2 different LabVIEW Applications:



| Memory Type | Data Type | Master Access | Slave Access |
|----------------|---------------|---------------|--------------|
| Discrete Input | Bit (Boolean) | Read-only | Read/Write |

LabVIEW Discrete Input Registers Examples

Modbus Slave - Discrete Input.vi

File Edit View Project Operate Tools Window Help

Discrete Input Values

Note! You need to start the Modbus Slave before you start the Modbus Master(s)

Stop

This screenshot shows the front panel of a LabVIEW virtual instrument titled "Modbus Slave - Discrete Input.vi". The interface includes a menu bar with "File", "Edit", "View", "Project", "Operate", "Tools", "Window", and "Help". Below the menu bar are standard LabVIEW control icons: a hand cursor, a refresh icon, a red stop button, and a pause icon. The main display area is titled "Discrete Input Values" and contains a horizontal row of ten green circular indicator lights. The fourth light from the left is a brighter shade of green, indicating it is active. At the bottom right of the panel, there is a button with a red square icon and the text "Stop".

Modbus Master - Discrete Input.vi

File Edit View Project Operate Tools Window Help

Discrete Input Values

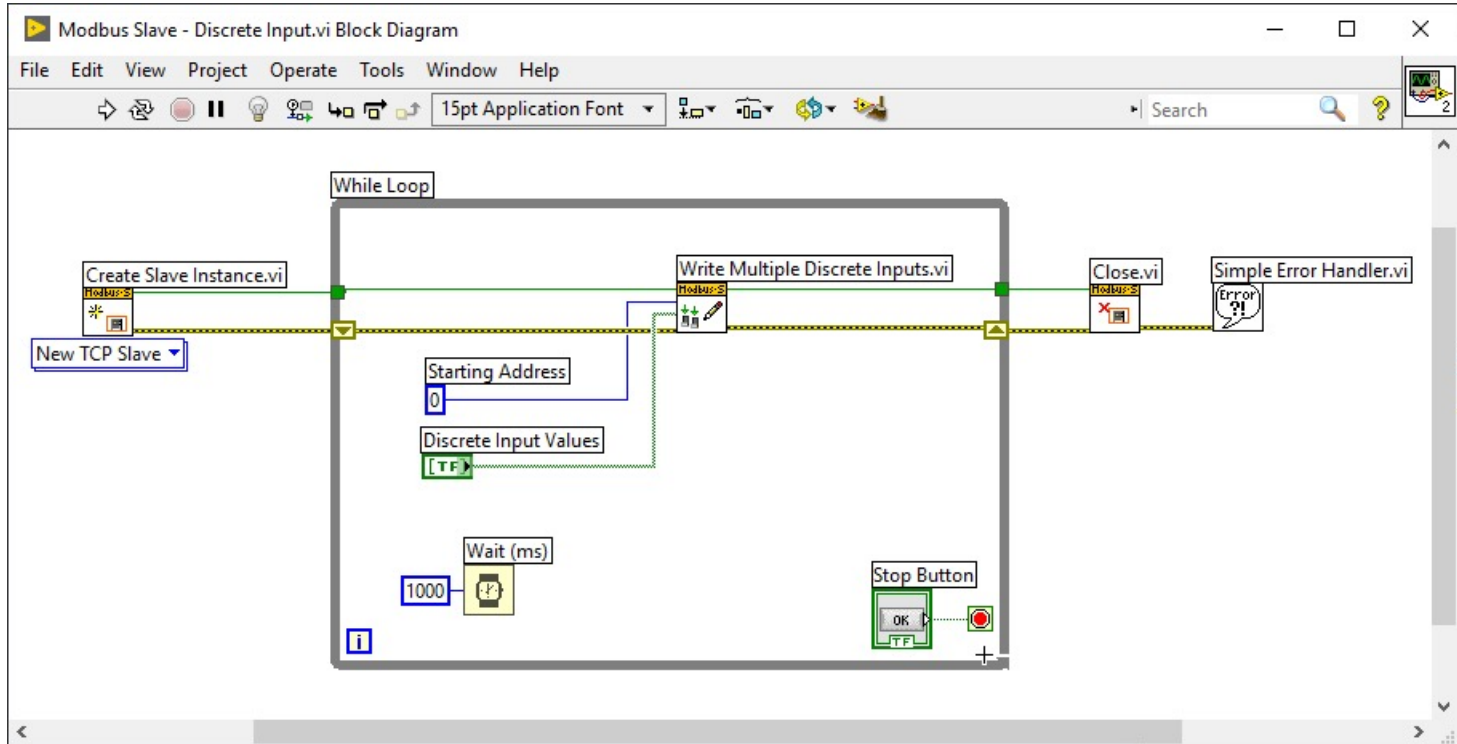
Note! You need to start the Modbus Slave before you start the Modbus Master(s)

Read

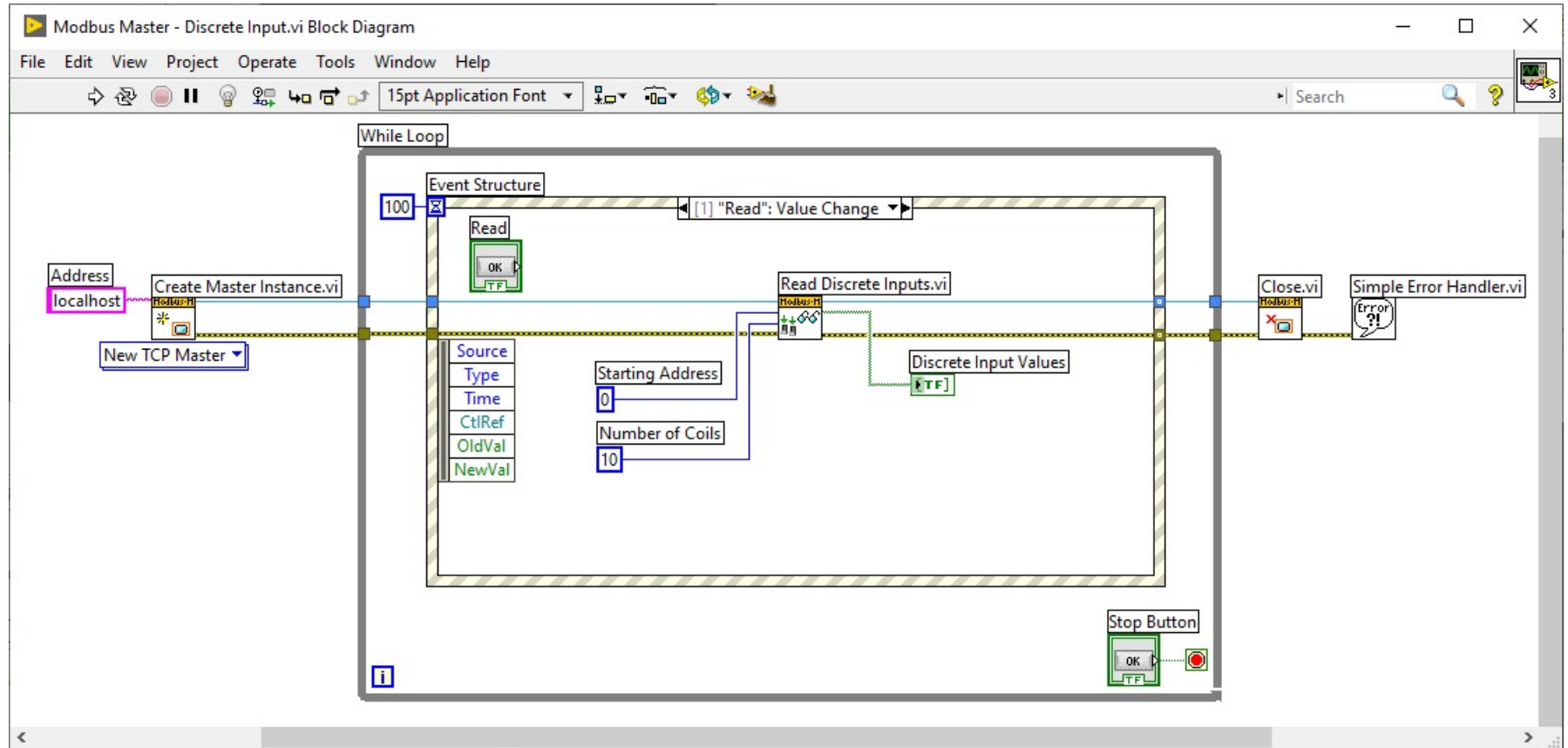
Stop

This screenshot shows the front panel of a LabVIEW virtual instrument titled "Modbus Master - Discrete Input.vi". The interface includes a menu bar with "File", "Edit", "View", "Project", "Operate", "Tools", "Window", and "Help". Below the menu bar are standard LabVIEW control icons: a hand cursor, a refresh icon, a red stop button, and a pause icon. The main display area is titled "Discrete Input Values" and contains a horizontal row of ten green circular indicator lights. The fourth light from the left is a brighter shade of green, indicating it is active. Below the indicator lights, there is a note: "Note! You need to start the Modbus Slave before you start the Modbus Master(s)". To the right of the note are two buttons: one with a blue circular refresh icon and the text "Read", and another with a red square icon and the text "Stop".

Modbus Slave



Modbus Master

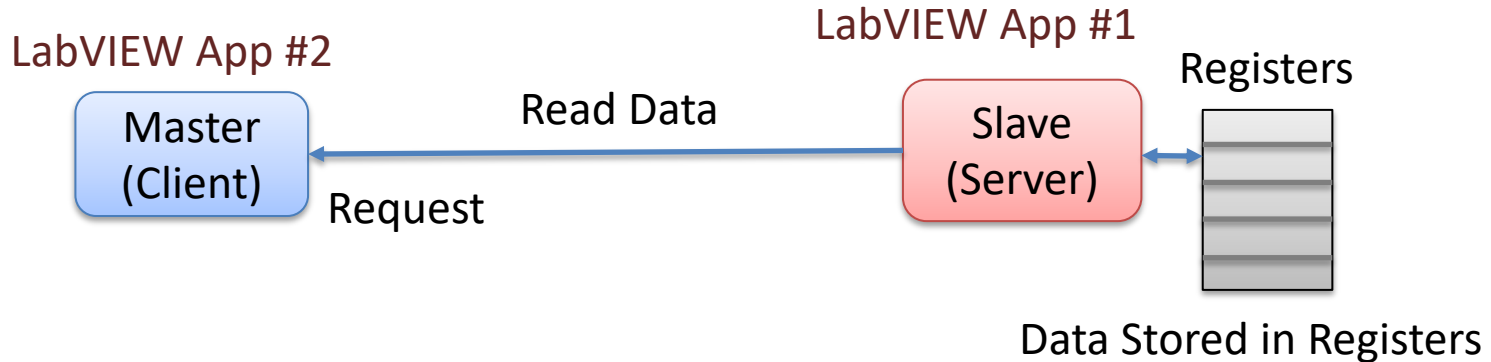




LabVIEW Input Registers Examples

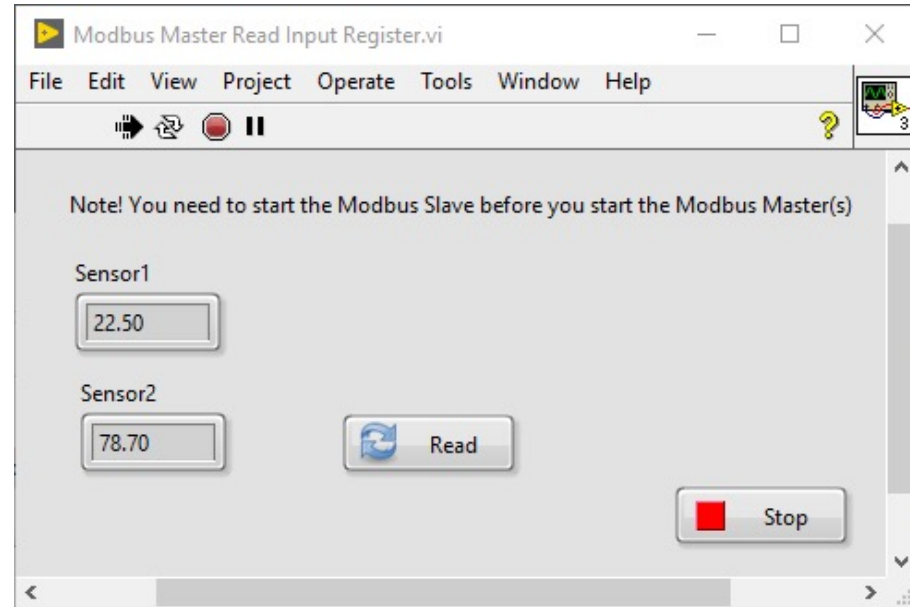
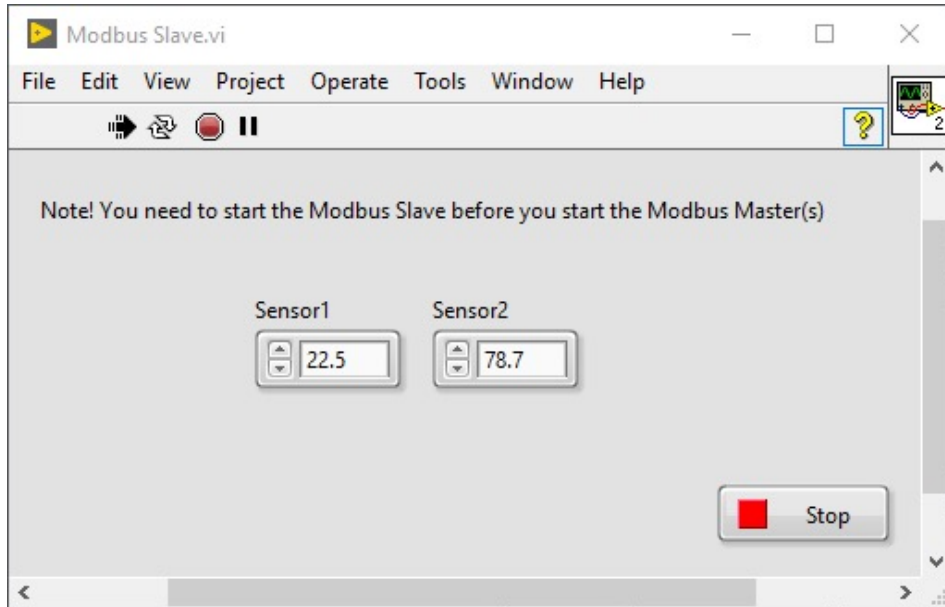
LabVIEW Input Registers Examples

In this Example we Create 2 different LabVIEW Applications:

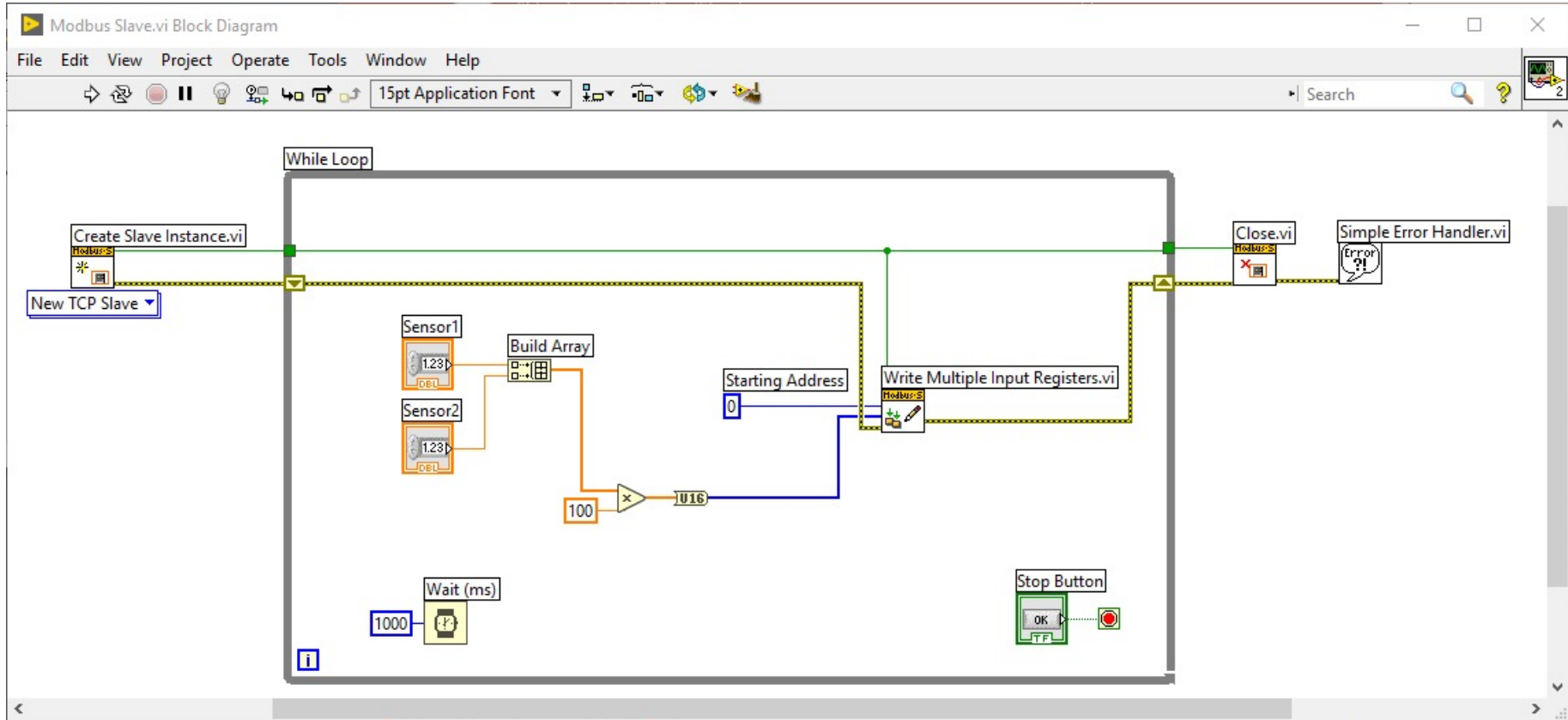


| Memory Type | Data Type | Master Access | Slave Access |
|----------------|---------------|---------------|--------------|
| Input Register | Unsigned Word | Read-only | Read/Write |

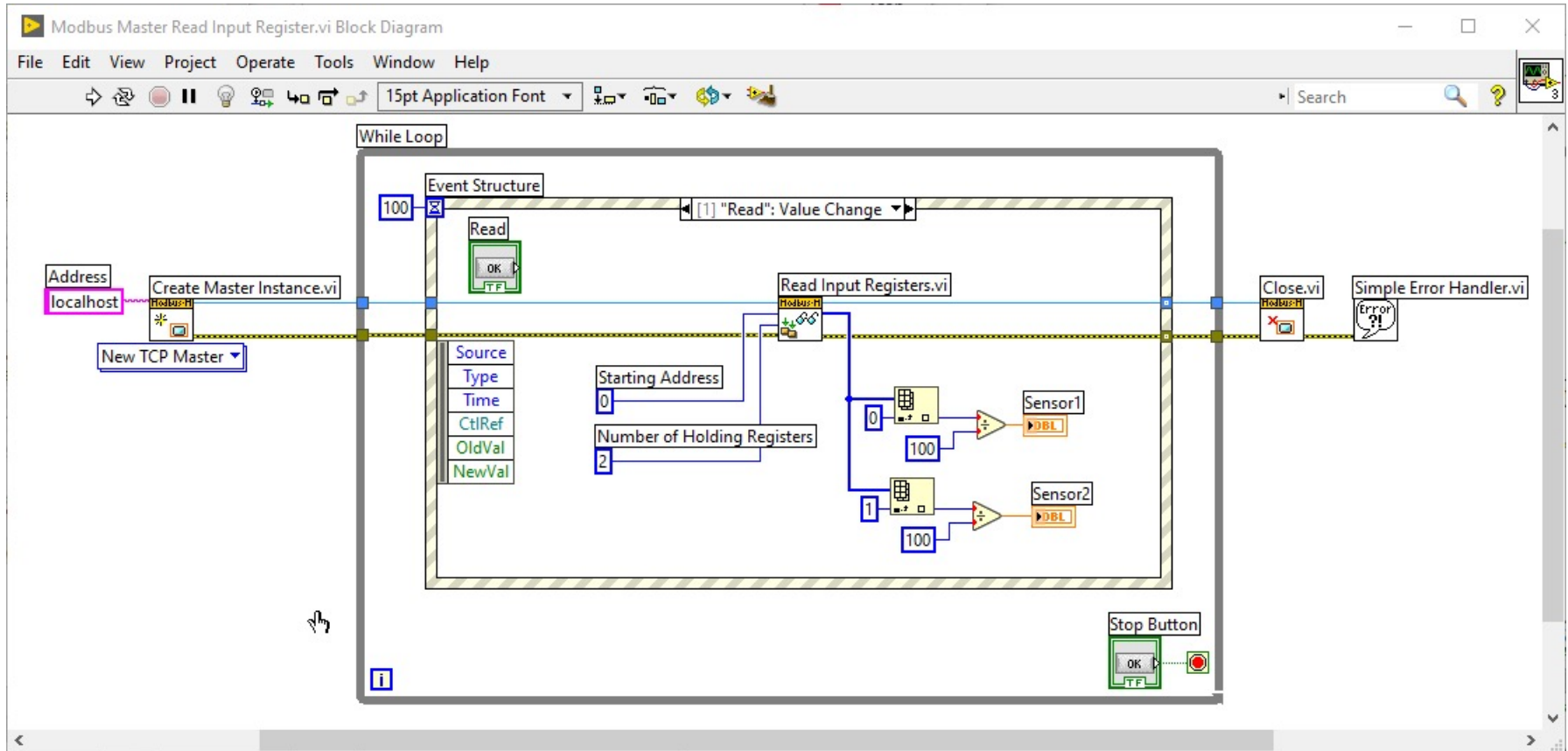
LabVIEW Input Registers Examples



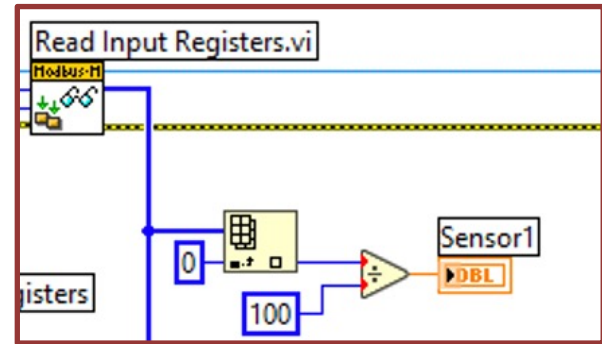
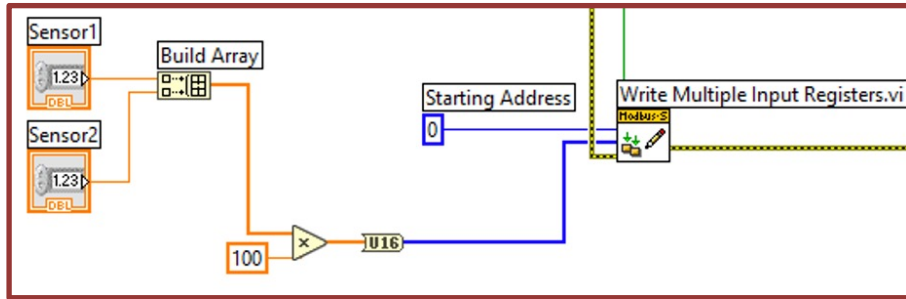
Modbus Slave



Modbus Master Read Input



Decimal/Floating-point Numbers



Memory Type

Data Type

Master Access

Slave Access

Input Register

Unsigned Word

Read-only

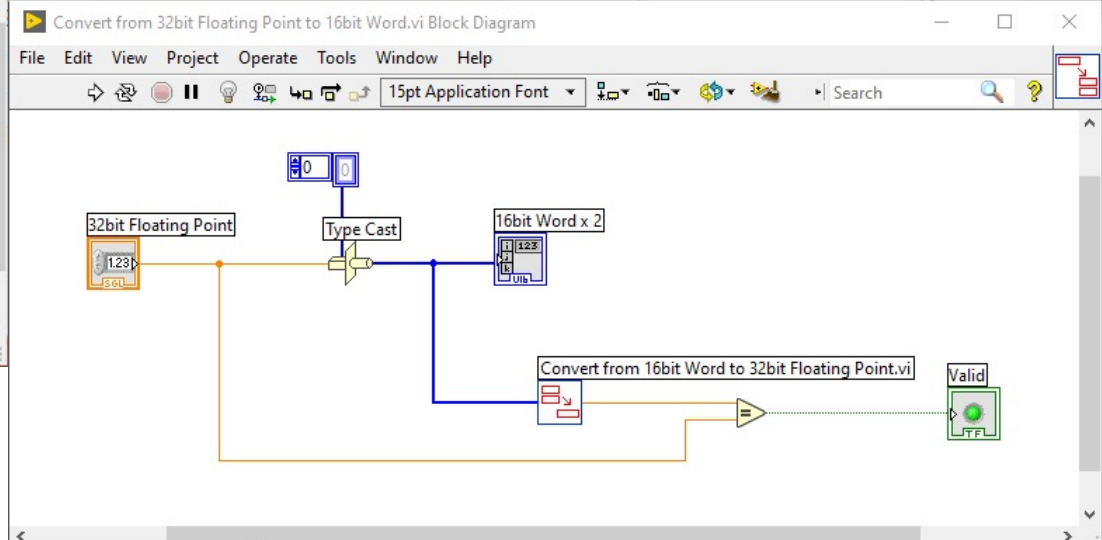
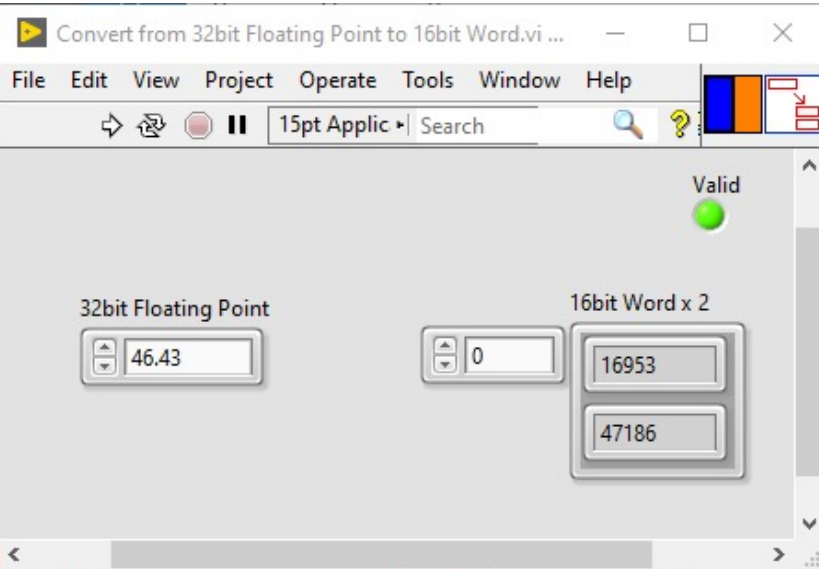
Read/Write

An **Unsigned Word** is a 16-bit nonnegative Integer Value between 0 – 65535 (2^{16})

- How do you deal with Decimal/Floating-point Numbers? In Modbus, the default practice is to split a 32-bit floating point value across two 16-bit registers.
- In this example I just Multiply with 100 in the the Slave Application, then I divide by 100 in the Master Application, which work when you deal with numbers with 2 decimals, and you only need one register per number
- Example: $2.56 \Rightarrow 2.56 \times 100 = 256 \Rightarrow 256 / 100 = 2.56$

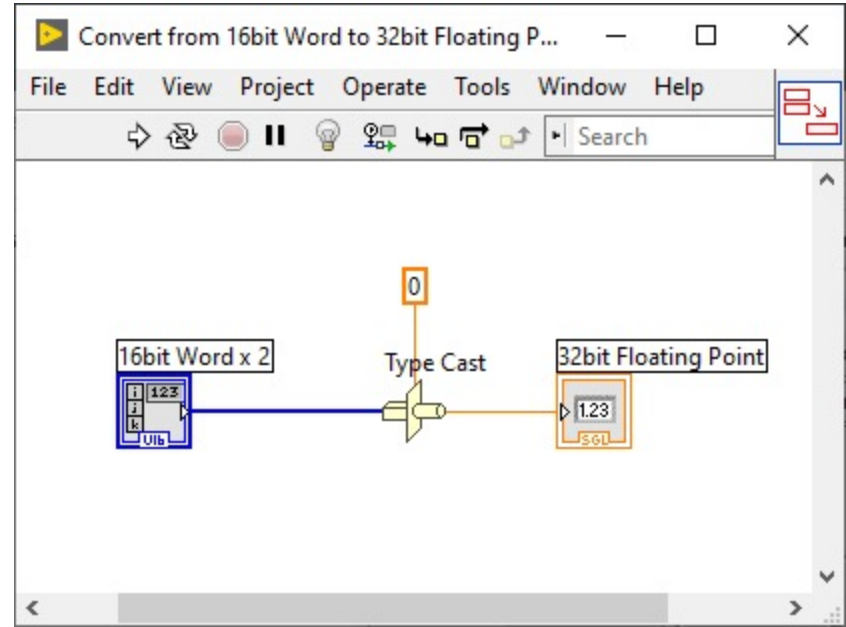
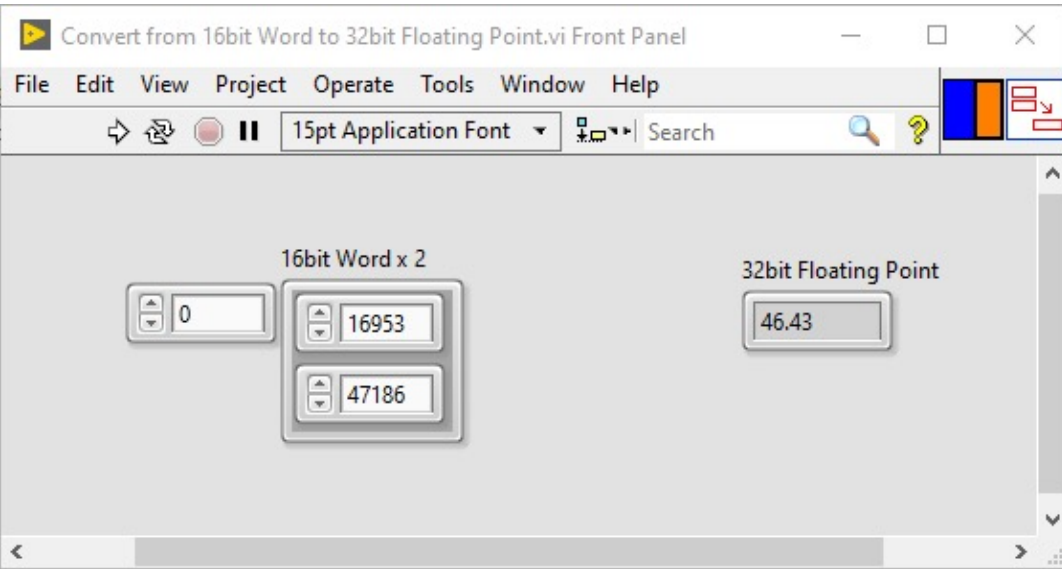
32-bit floating point across two 16-bit registers

Here we have split a 32-bit floating point value across two 16-bit registers



32-bit floating point across two 16-bit registers

Here we get the 32-bit floating point from two 16-bit registers

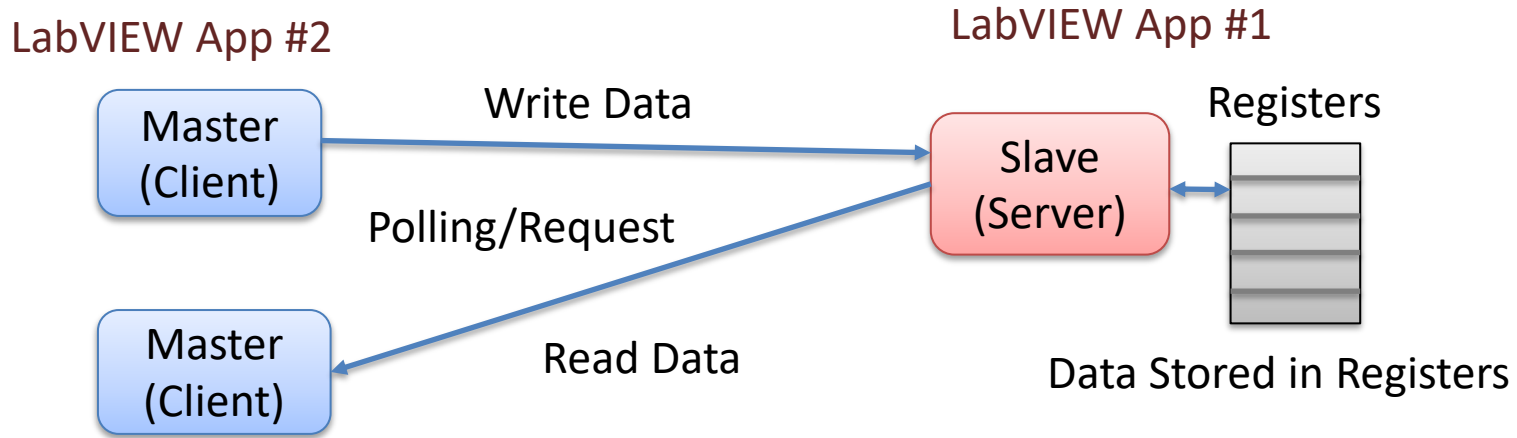




LabVIEW Holding Registers Examples

LabVIEW Holding Registers Example

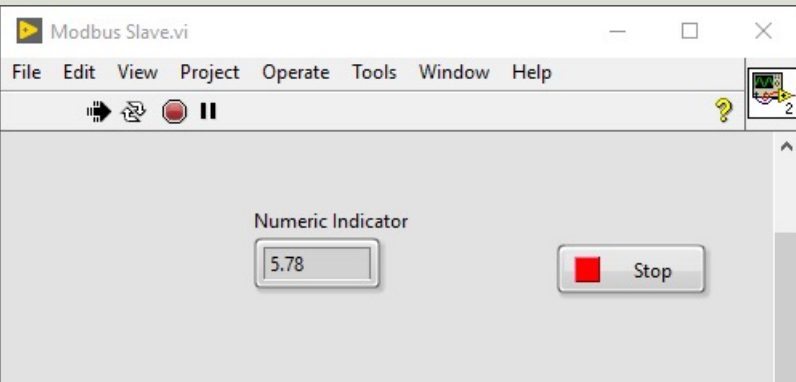
In this Example we Create 3 different LabVIEW Applications:



LabVIEW App #3

| Memory Type | Data Type | Master Access | Slave Access |
|------------------|---------------|---------------|--------------|
| Holding Register | Unsigned Word | Read/Write | Read/Write |

LabVIEW Holding Registers Example



Modbus Slave.vi

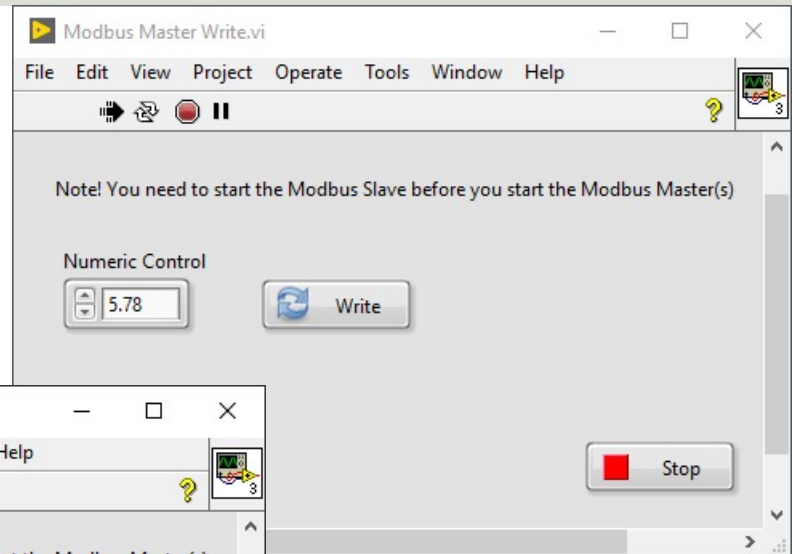
File Edit View Project Operate Tools Window Help

Hand cursor, Refresh, Run, Pause

Numeric Indicator: 5.78

Stop button

Icon: 2



Modbus Master Write.vi

File Edit View Project Operate Tools Window Help

Hand cursor, Refresh, Run, Pause

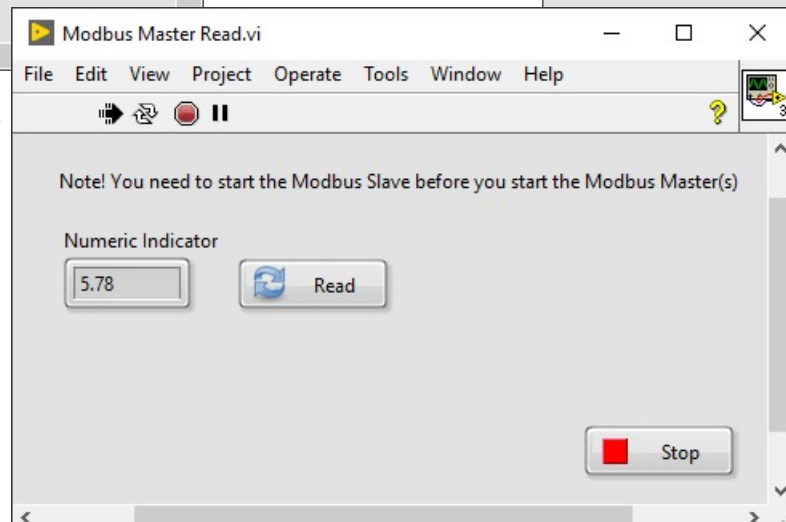
Note! You need to start the Modbus Slave before you start the Modbus Master(s)

Numeric Control: 5.78

Write button

Stop button

Icon: 3



Modbus Master Read.vi

File Edit View Project Operate Tools Window Help

Hand cursor, Refresh, Run, Pause

Note! You need to start the Modbus Slave before you start the Modbus Master(s)

Numeric Indicator: 5.78

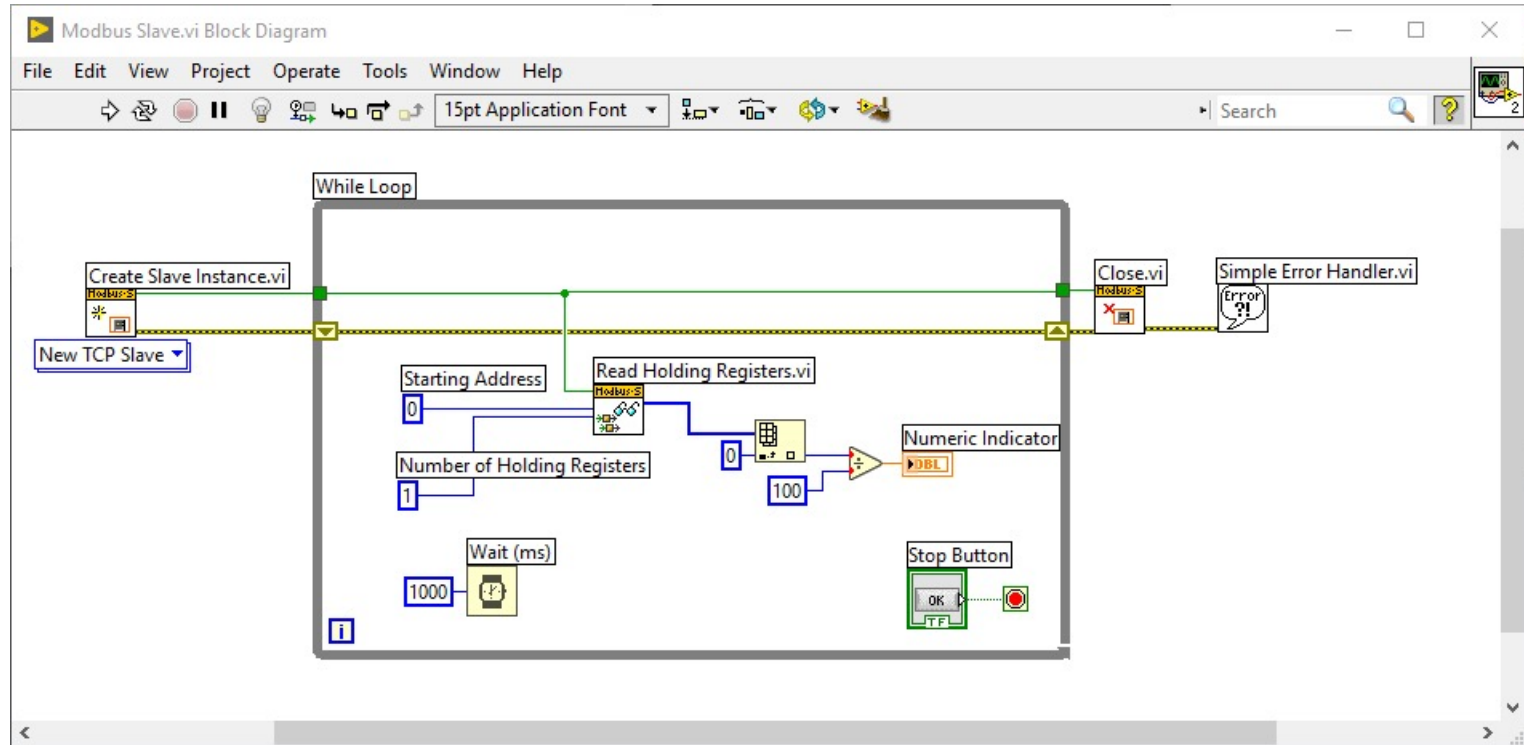
Read button

Stop button

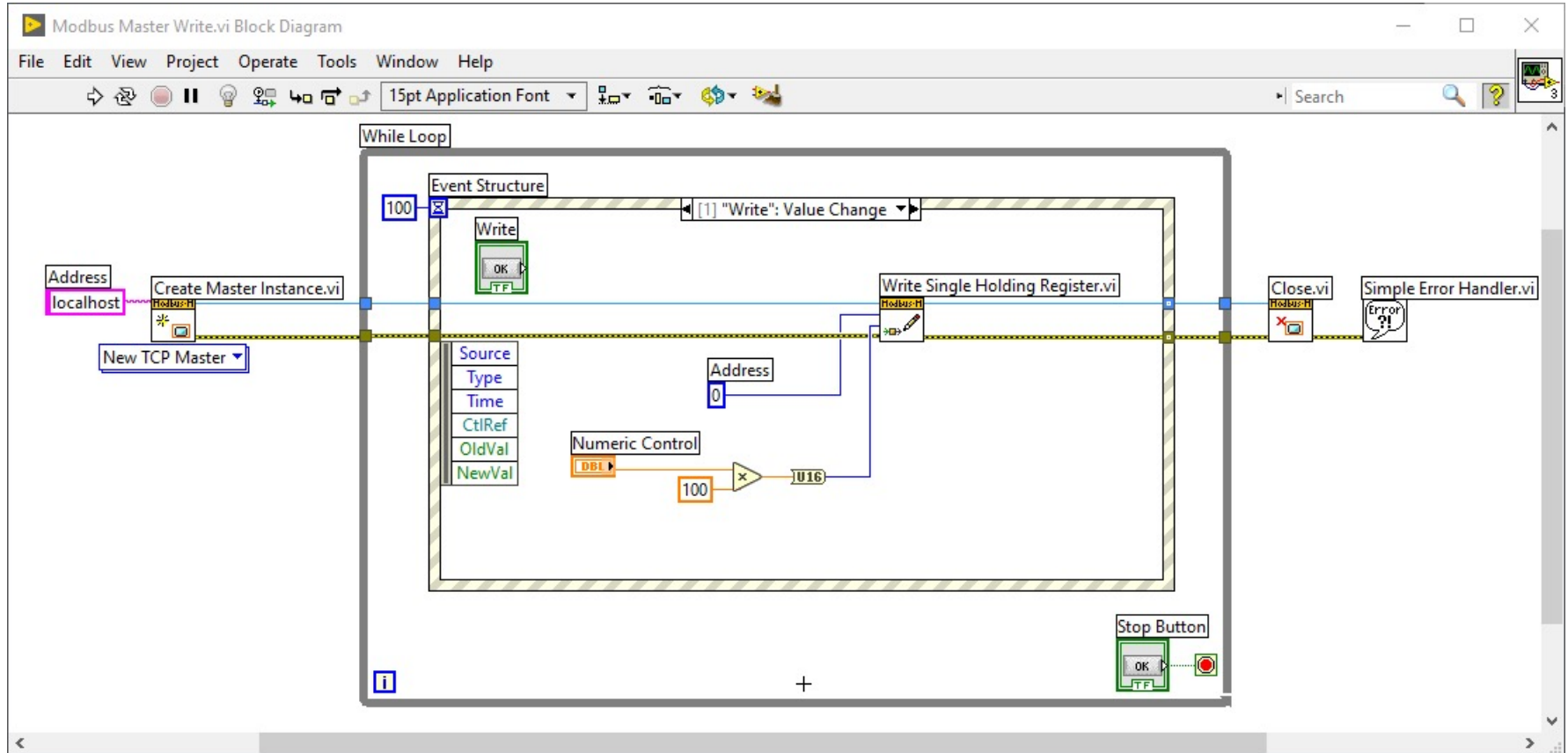
Icon: 3



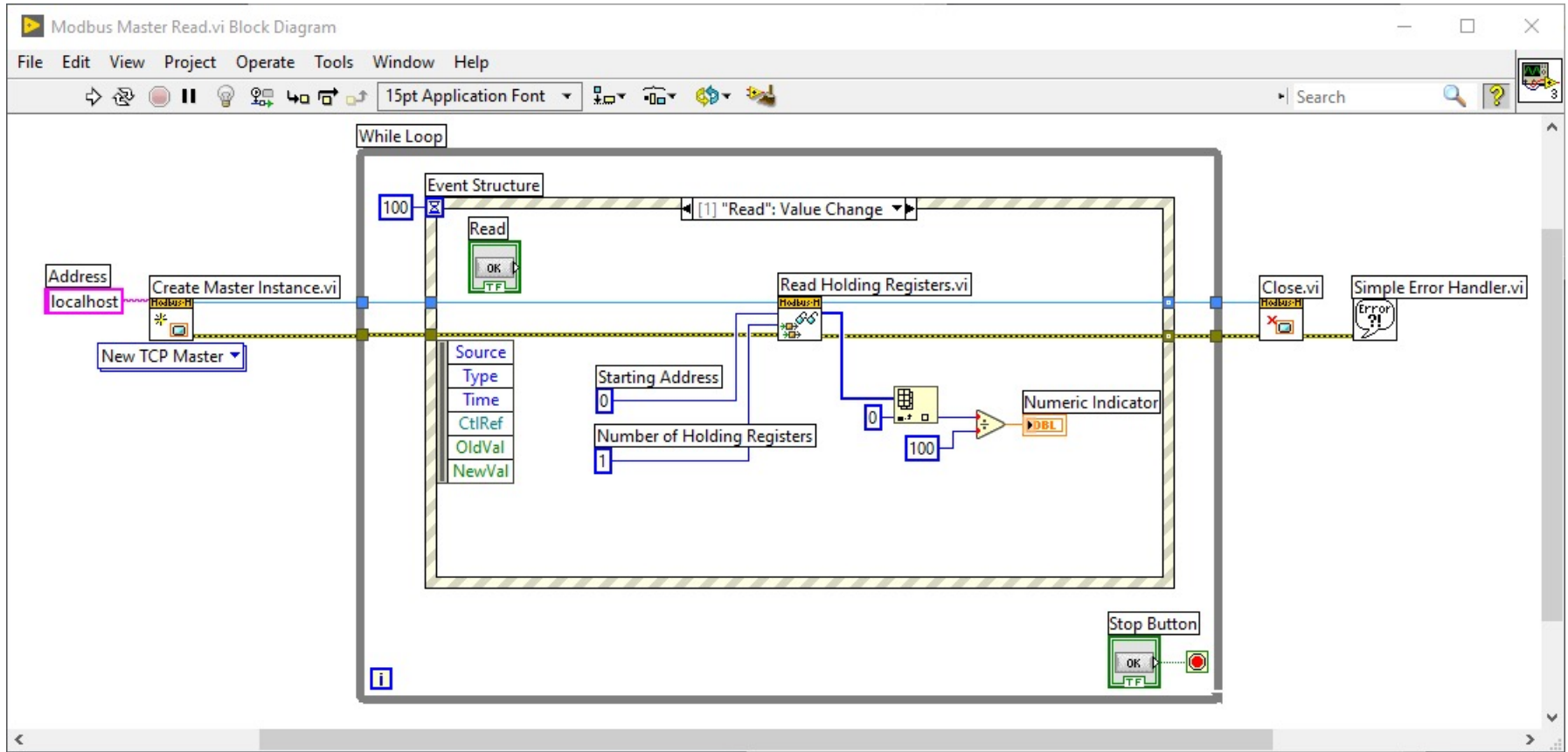
Modbus Slave



Modbus Master (Write)



Modbus Master (Read)

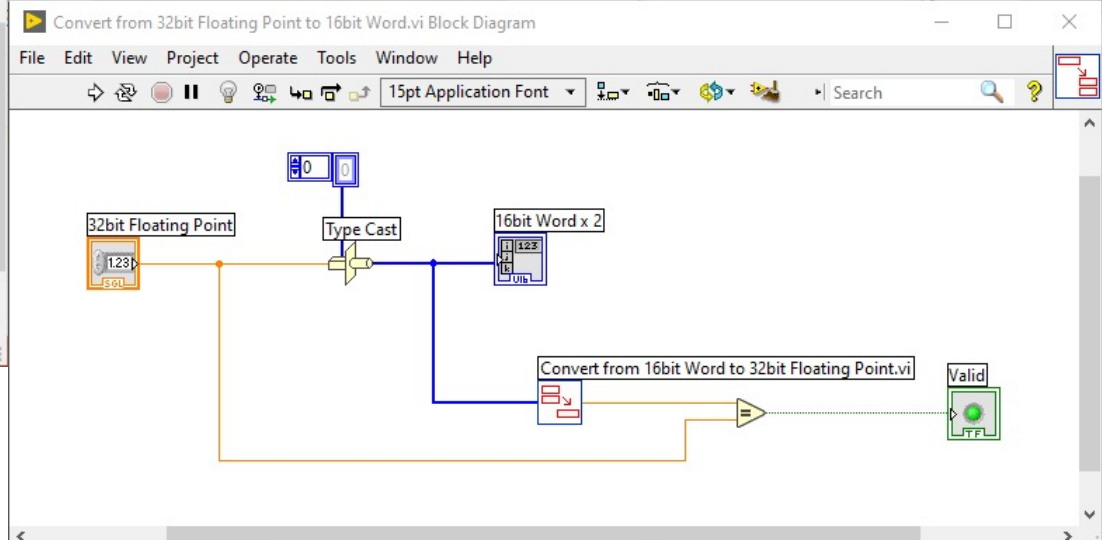
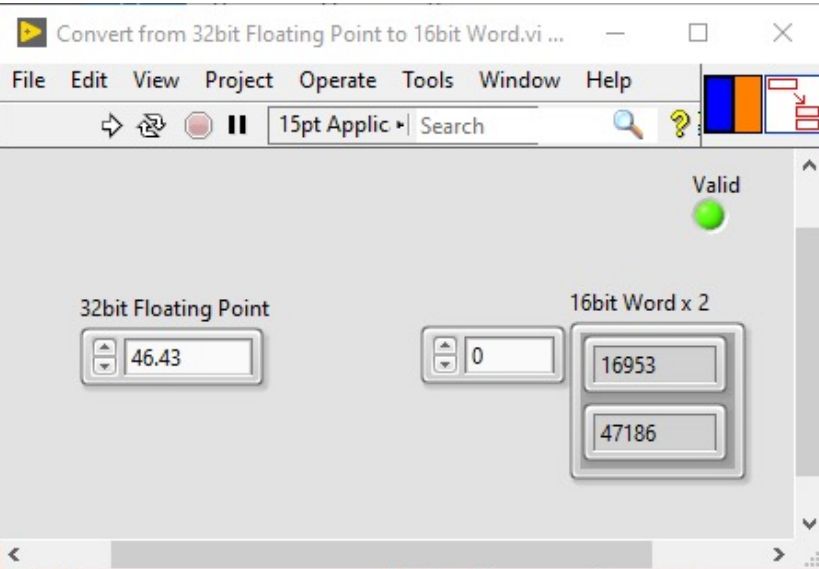


Alt Solution

- How do you deal with Decimal/Floating-point Numbers?
- Previously we implemented a simple solution by multiplying and dividing with 100, which worked fine for 2 decimal numbers
- In Modbus, the default practice is to split a 32-bit floating point value across two 16-bit registers.
- The disadvantage is that we need to use 2 Modbus register for representing one number

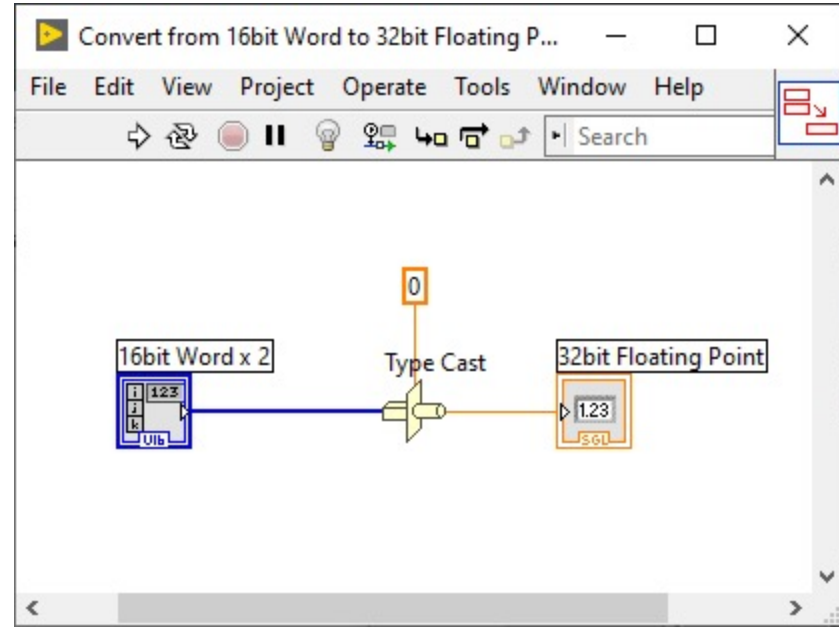
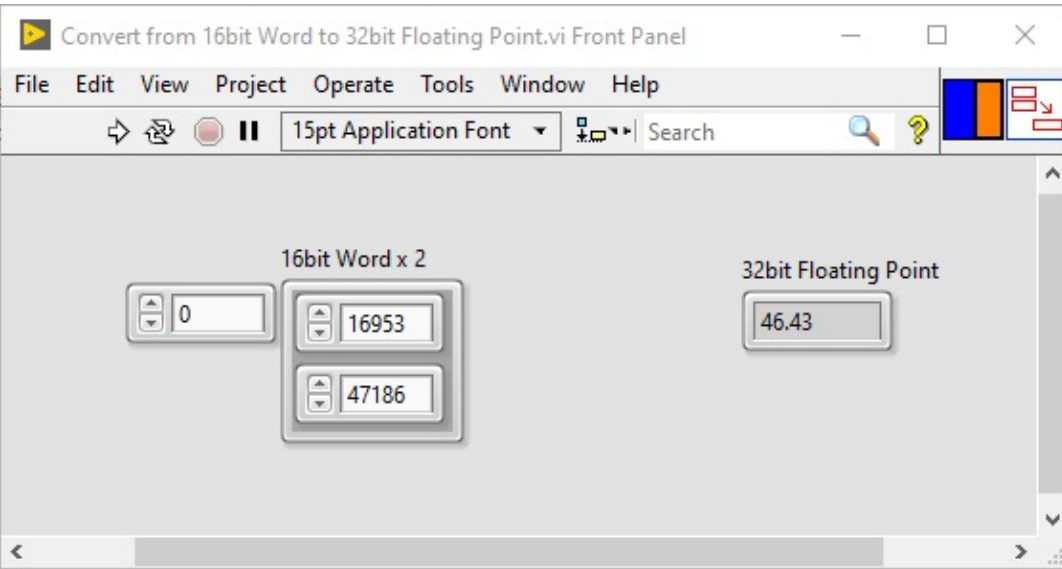
32-bit floating point across two 16-bit registers

Here we have split a 32-bit floating point value across two 16-bit registers

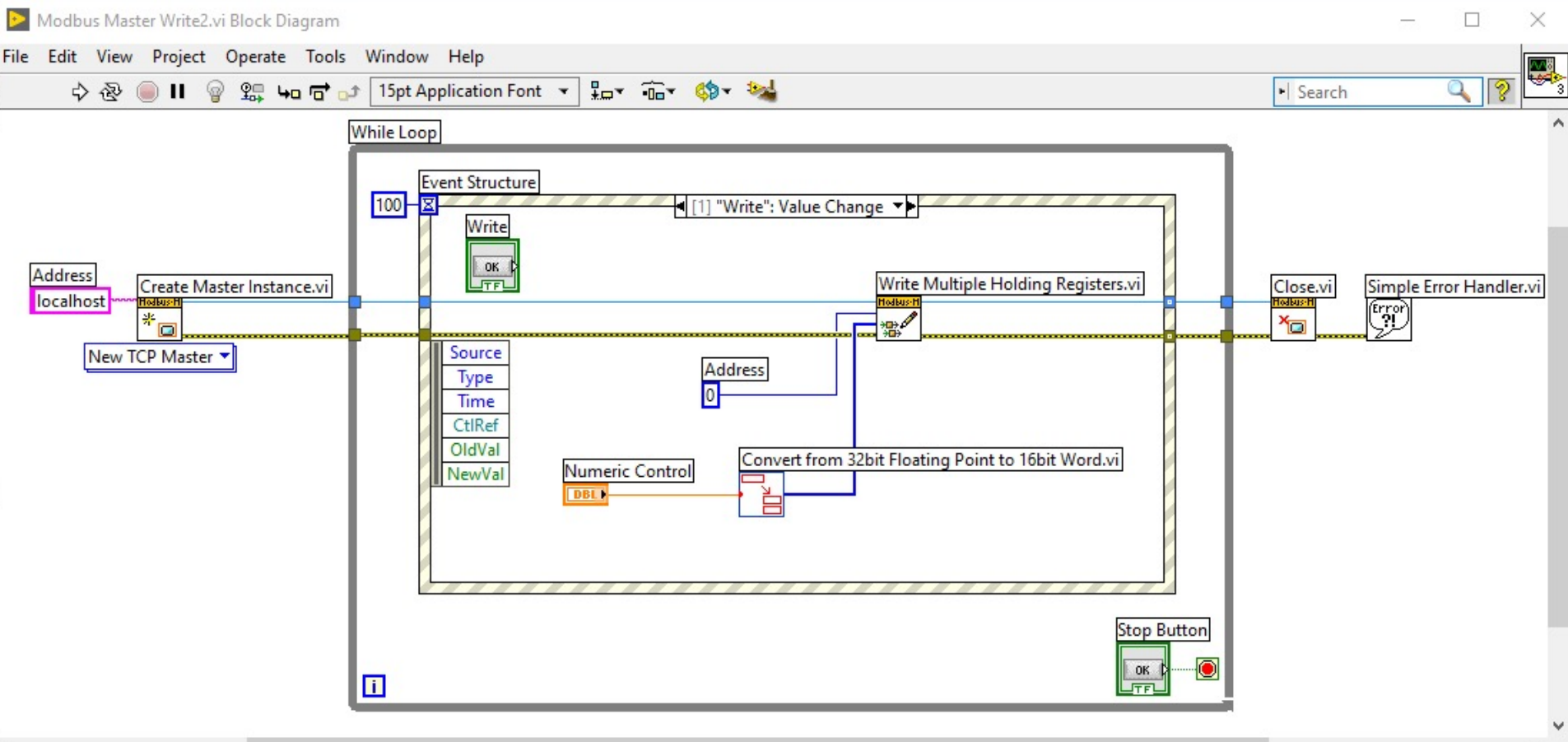


32-bit floating point across two 16-bit registers

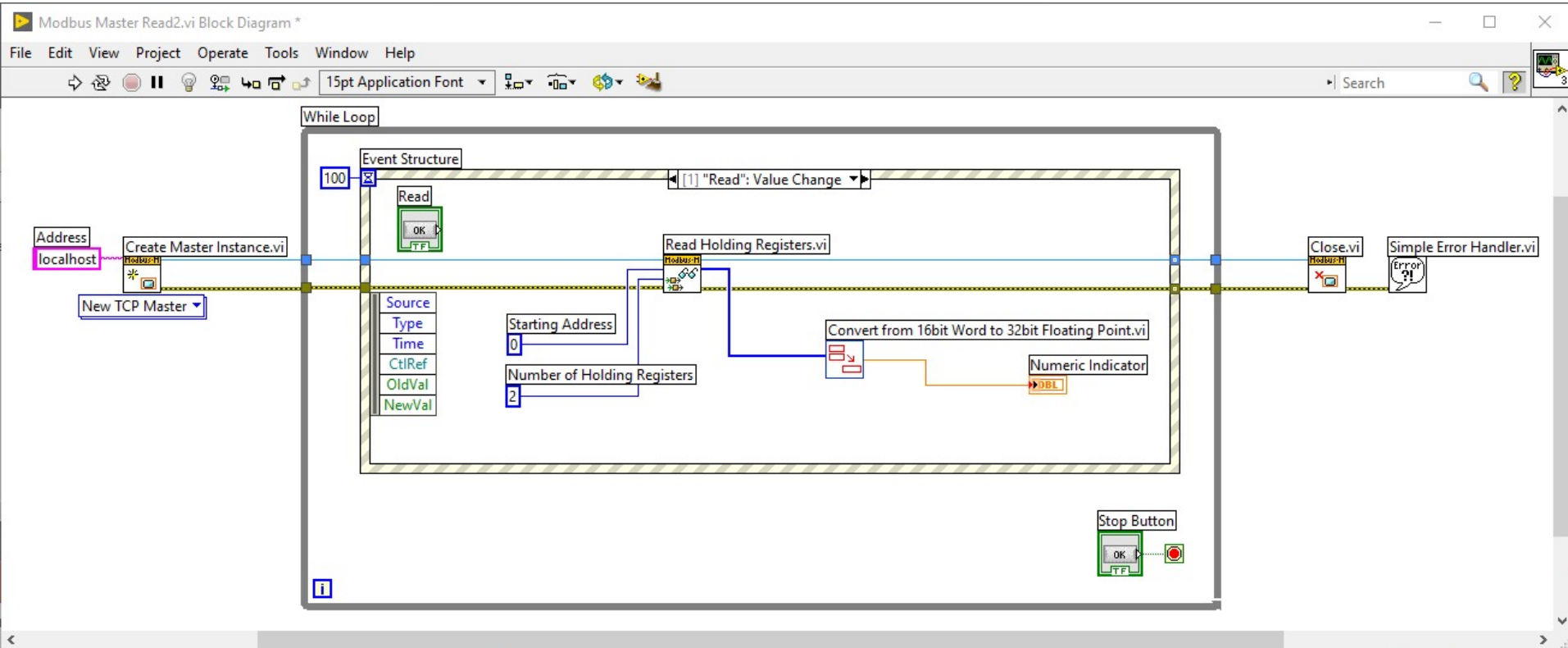
Here we get the 32-bit floating point from two 16-bit registers



Modbus Master (Write)



Modbus Master (Read)



Modbus Registers Summary

| Register Type | Data Type | Master Access | Slave Access |
|-------------------------|---------------|---------------|--------------|
| Coils | Bit (Boolean) | Read/Write | Read/Write |
| Discrete Input | Bit (Boolean) | Read-only | Read/Write |
| Input Register | Unsigned Word | Read-only | Read/Write |
| Holding Register | Unsigned Word | Read/Write | Read/Write |

An **Unsigned Word** is a 16-bit nonnegative Integer Value between 0 – 65535 (2^{16})

References

- Modbus Organization: <http://www.modbus.org>
- Modbus (Wikipedia): <https://en.wikipedia.org/wiki/Modbus>
- Introduction to Modbus (National Instruments):
<http://www.ni.com/white-paper/7675/en/>
- Connect LabVIEW to Any PLC With Modbus (National Instruments):
<http://www.ni.com/tutorial/13911/en/>
- Modbus 101 - Introduction to Modbus:
http://www.csimn.com/CSI_pages/Modbus101.html
- Modbus TCP/IP: <http://www.rtaautomation.com/technologies/modbus-tcpip/>
- Modbus RTU: <http://www.rtaautomation.com/technologies/modbus-rtu/>
- Using Modbus for Process Control and Automation (PDF):
http://www.miinet.com/Portals/0/articles/Using_MODBUS_for_Process_Control_and_Automation.pdf

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: <https://www.halvorsen.blog>

