XMC4700 / 4800 MODBUS® RTU/ASCII Client Example

Getting Started V1.0

1 Overview and Requirements

2 Setup

3 Implementing the application

4 How to test

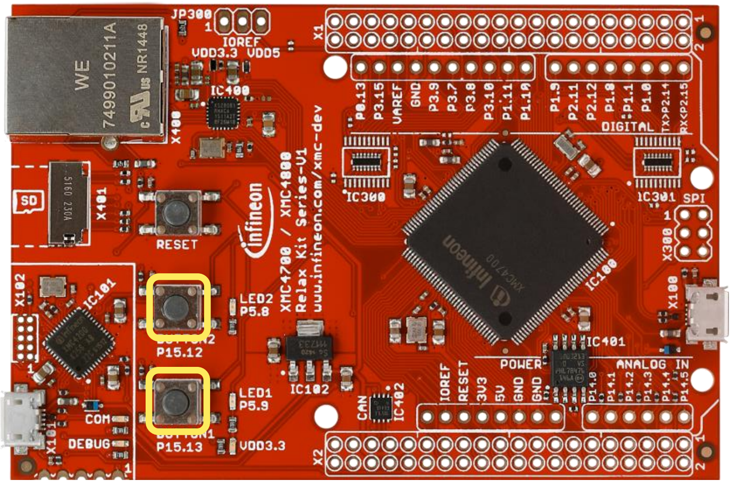
1 Overview and Requirements

2 Setup

3 Implementing the application

4 How to test

Overview



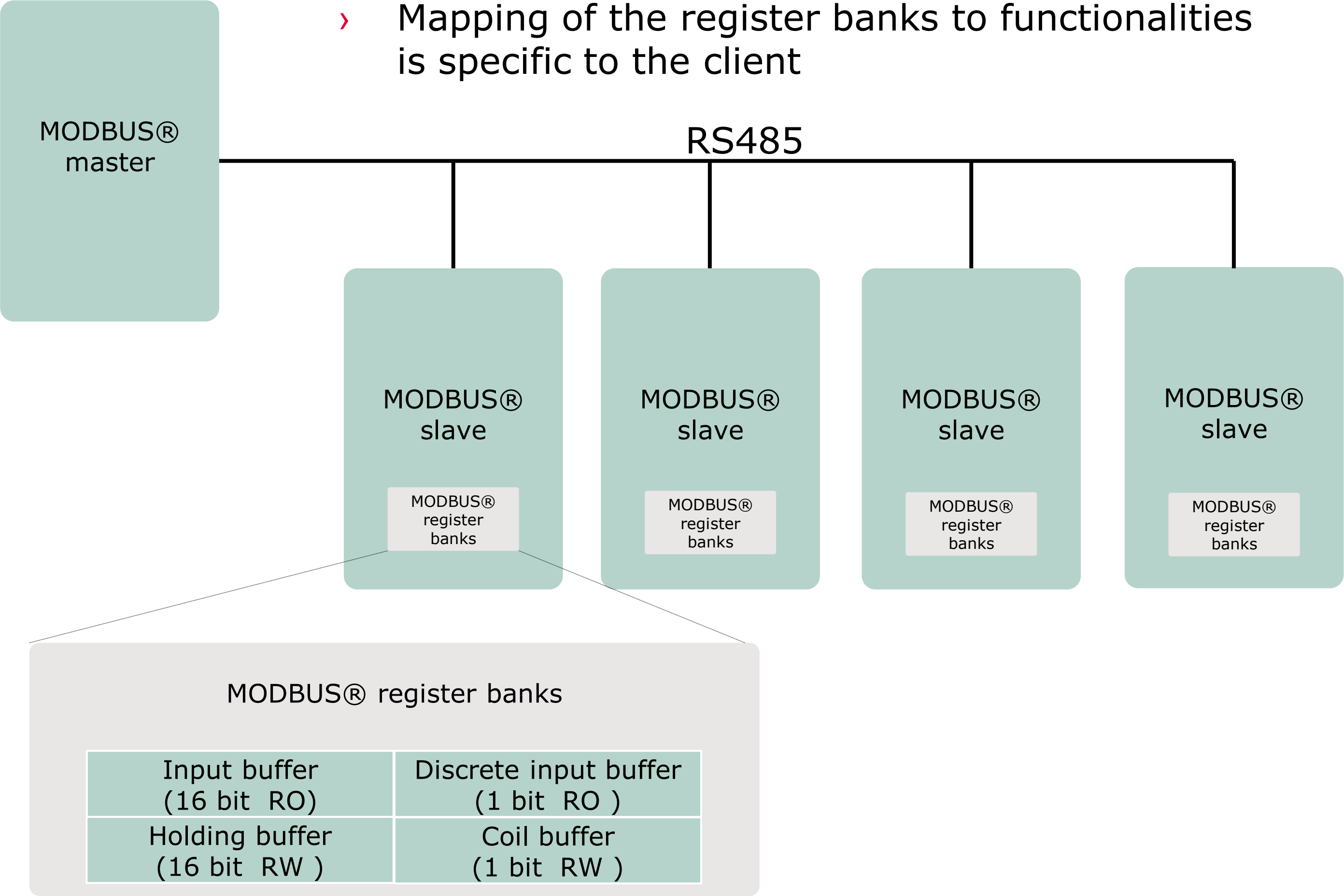
This example demonstrates the implementation of a serial RTU and ASCII Modbus® client. It is based on the FreeMODBUS implementation which is ported to XMC4000 family and provided within this example.

Within this documentation you will be guided through all the building blocks of a typical serial RTU Modbus® application on the XMC4000 family. You will see the input buttons of the board mapped to binary Modbus® discrete input registers and learn to poll their status on your laptop which is used as a Modbus® host.

As a result you will be enabled to implement your own Modbus® client on the XMC4000 family.

Overview – ModBUS®

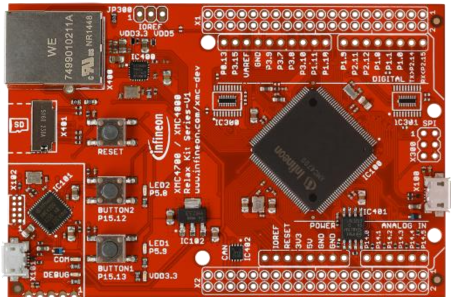
- Master can address each client individually
- Master has access to the register bank inside each client
- MODBUS® protocol defines 4 type of registers
- Mapping of the register banks to functionalities is specific to the client



MODBUS® register banks

Input buffer (16 bit RO)	Discrete input buffer (1 bit RO)
Holding buffer (16 bit RW)	Coil buffer (1 bit RW)

Requirements – hardware



XMC 4700 Relax Kit

or optional

XMC4800 Relax EtherCAT Kit

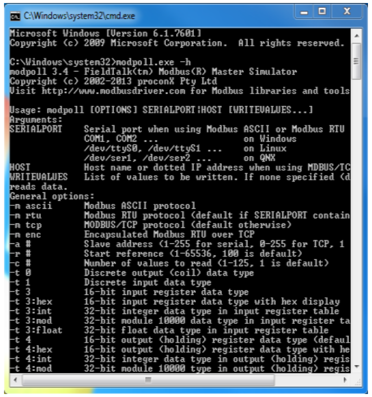
Windows Laptop installed

- DAVE v4 (Version4.1.4 or higher)
- Modpoll command line tool

Micro USB Cable (Debugger and connector)

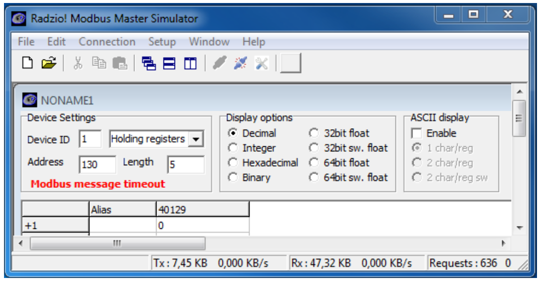
Requirements - free software downloads

DAVE™ (v4.1.4 or higher)
Link: Download DAVE (Version 4)



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>modpoll.exe -h
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools

Usage: modpoll [OPTIONS] SERIALPORT|HOST [WRITEVALUES...]
Arguments:
SERIALPORT    Serial port when using Modbus ASCII or Modbus RTU
              COM1, COM2 ...                on Windows
              /dev/ttyS0, /dev/ttyS1 ...    on Linux
              /dev/ser1, /dev/ser2 ...      on QNX
HOST          Host name or dotted IP address when using MDBUS/TC
WRITEVALUES   List of values to be written. If none specified (d
reads data.
General options:
-m ascii      Modbus ASCII protocol
-m rtu        Modbus RTU protocol (default if SERIALPORT contain
-m tcp        MODBUS/TCP protocol (default otherwise)
-m enc        Encapsulated Modbus RTU over TCP
-a #          Slave address (1-255 for serial, 0-255 for TCP, 1
-r #          Start reference (1-65536, 100 is default)
-c #          Number of values to read (1-125, 1 is default)
-t 0          Discrete output (coil) data type
-t 1          Discrete input data type
-t 3          16-bit input register data type
-t 3:hex      16-bit input register data type with hex display
-t 3:int      32-bit integer data type in input register table
-t 3:mod      32-bit module 10000 data type in input register ta
-t 3:float    32-bit float data type in input register table
-t 4          16-bit output (holding) register data type (defaul
-t 4:hex      16-bit output (holding) register data type with he
-t 4:int      32-bit integer data type in output (holding) regis
-t 4:mod      32-bit module 10000 type in output (holding) regis
```

Modpoll Modbus Master Simulator
Link: modpoll.exe command line tool



Device Settings
Device ID 1 Holding registers
Address 130 Length 5
Modbus message timeout

Display options
● Decimal ○ 32bit float
○ Integer ○ 32bit sw. float
○ Hexadecimal ○ 64bit float
○ Binary ○ 64bit sw. float

ASCII display
□ Enable
● 1 char/reg
○ 2 char/reg
○ 2 char/reg sw

	Alias	40129
+1		0

Tx : 7,45 KB 0,000 KB/s | Rx : 47,32 KB 0,000 KB/s | Requests : 636 0

Optional:
Radzio! Modbus Master Simulator
Link: Download Radzio!

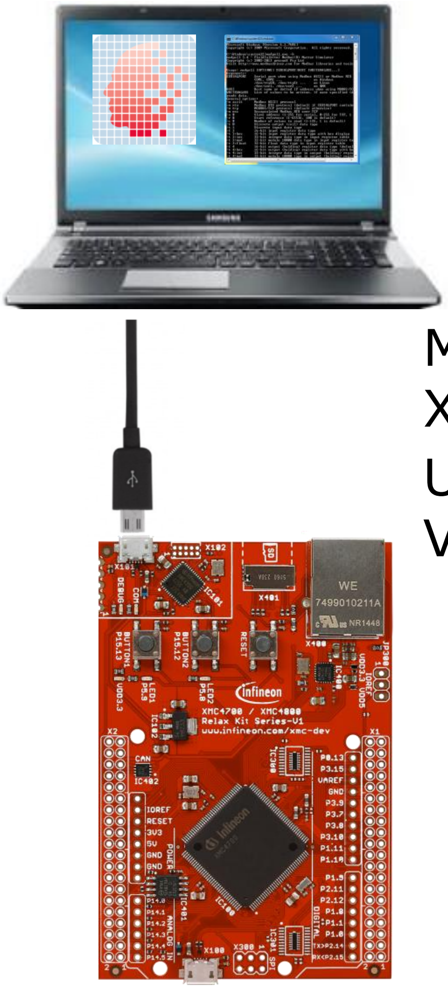
1 Overview and Requirements

2 Setup

3 Implementing the application

4 How to test

Setup – Hardware

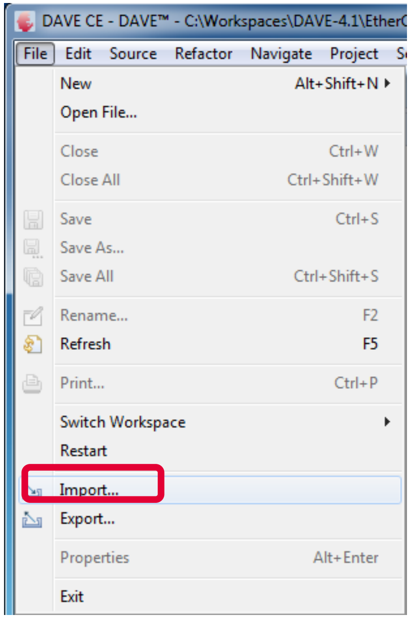


Micro USB cable connected to X101 debug connector.

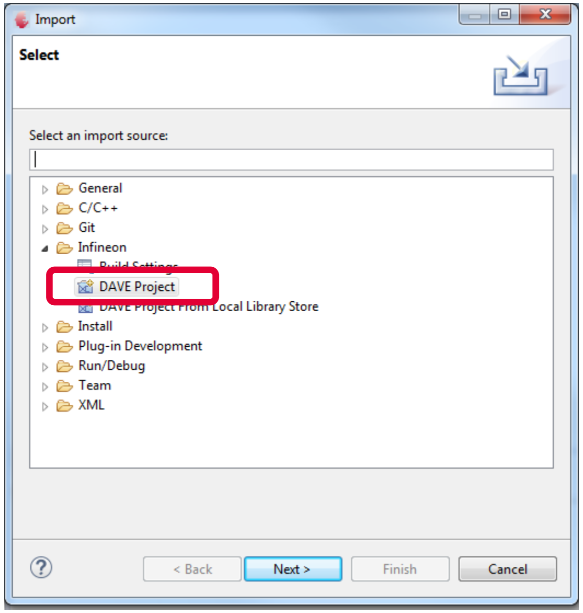
Used as debug interface and VCOM

Setup – Import example project into DAVE™

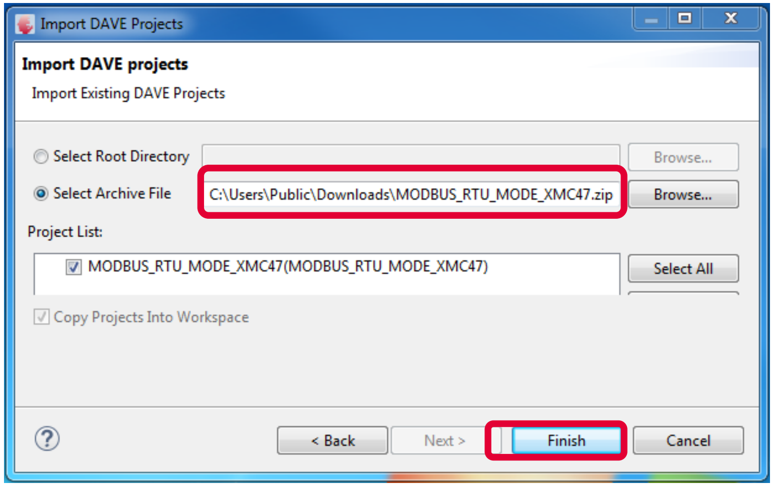
1



2



3



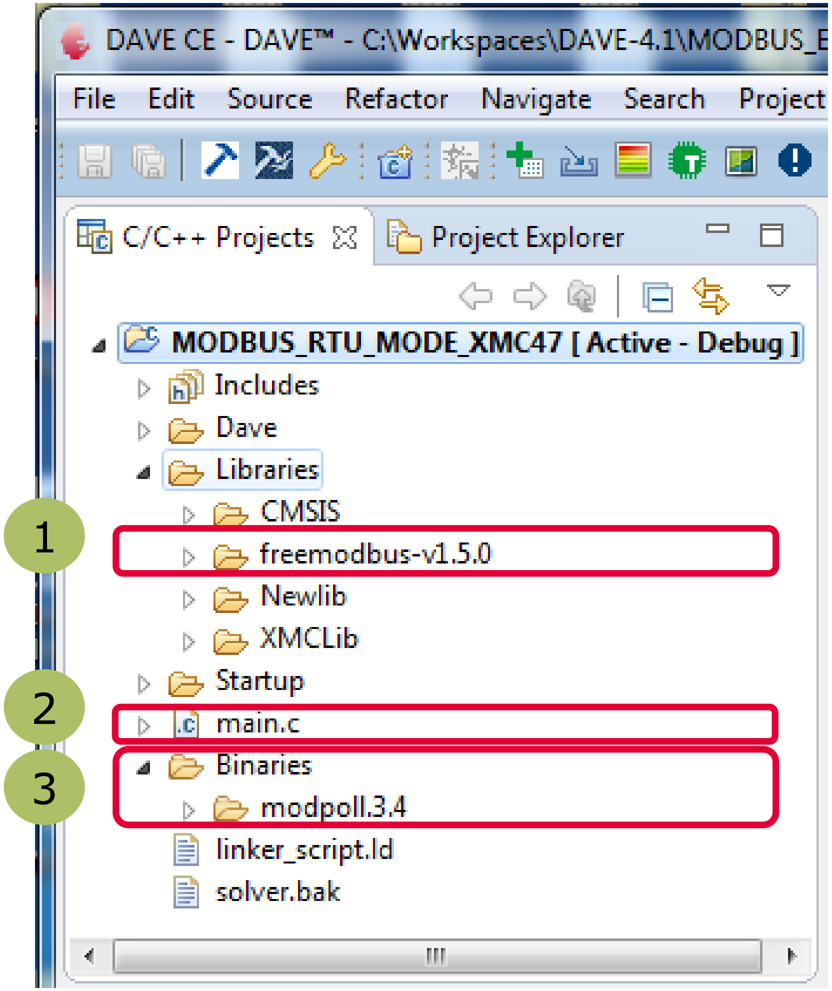
○ Select Root Directory

● Select Archive File C:\Users\Public\Downloads\MODBUS_RTU_MODE_XMC47.zip

☑ MODBUS_RTU_MODE_XMC47(MODBUS_RTU_MODE_XMC47)

☑ Copy Projects Into Workspace

Setup – Import example project into DAVE™



- MODBUS_RTU_MODE_XMC47 [Active - Debug]
  - Includes
  - Dave
  - Libraries
    - CMSIS
    - freemodbus-v1.5.0
    - Newlib
    - XMCLib
  - Startup
  - main.c
  - Binaries
    - modpoll.3.4
  - linker_script.ld
  - solver.bak

Check the folder structure of the imported project.
Beside standard DAVE project items you will find these Modbus® related items:

1 The FreeMODBUS library including the port folder for XMC4000 family

2 The main file implementing the example application

3 Modpoll Modbus Master Simulator. A command line tool to test this example using your laptop as a master

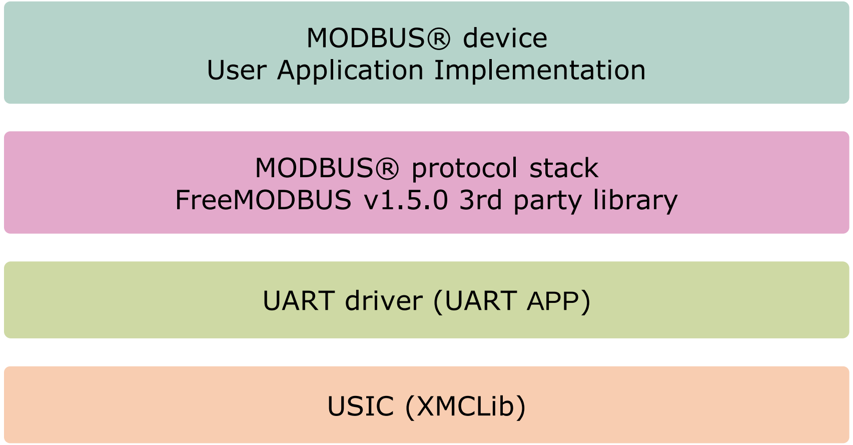
1 Overview and Requirements

2 Setup

3 Implementing the application

4 How to test

Application – Architecture



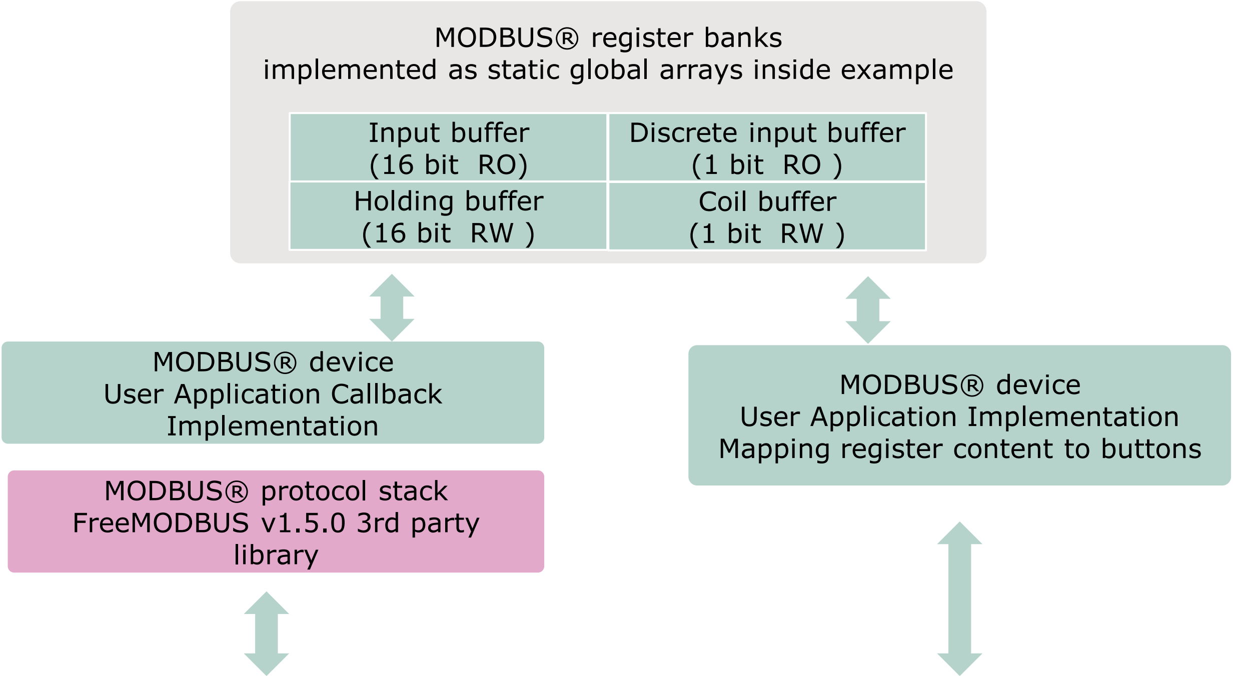
MODBUS® device
User Application Implementation

MODBUS® protocol stack
FreeMODBUS v1.5.0 3rd party library

UART driver (UART APP)

USIC (XMCLib)

Application – Data flow



MODBUS® register banks
implemented as static global arrays inside example

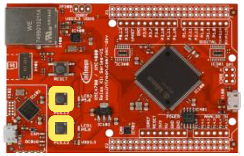
Input buffer (16 bit RO)	Discrete input buffer (1 bit RO)
Holding buffer (16 bit RW)	Coil buffer (1 bit RW)

MODBUS® device
User Application Callback
Implementation

MODBUS® protocol stack
FreeMODBUS v1.5.0 3rd party
library

MODBUS® device
User Application Implementation
Mapping register content to buttons

MODBUS® Master

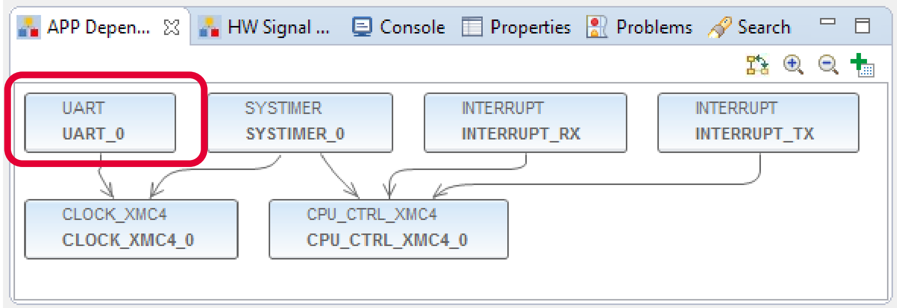


MODBUS® Client

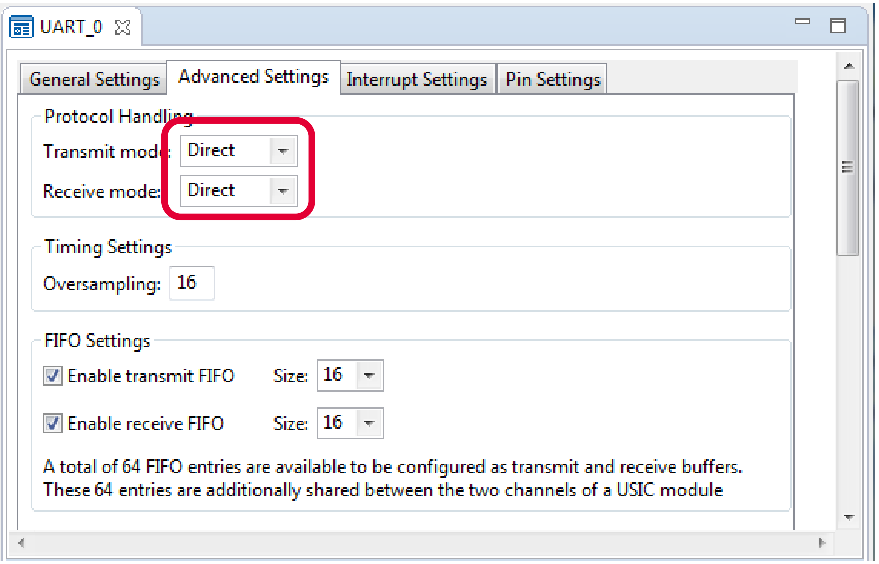
All read/write accesses of the MODBUS® Master to the register banks are handled inside callbacks from the protocol stack. The implementation of these callbacks is specific to the application.
Some of these registers are mapped to input button for demonstration.

APP structure – UART configuration (1/4)

1



2



Protocol Handling

Transmit mode: Direct

Receive mode: Direct

Timing Settings

Oversampling: 16

FIFO Settings

☑ Enable transmit FIFO Size: 16

☑ Enable receive FIFO Size: 16

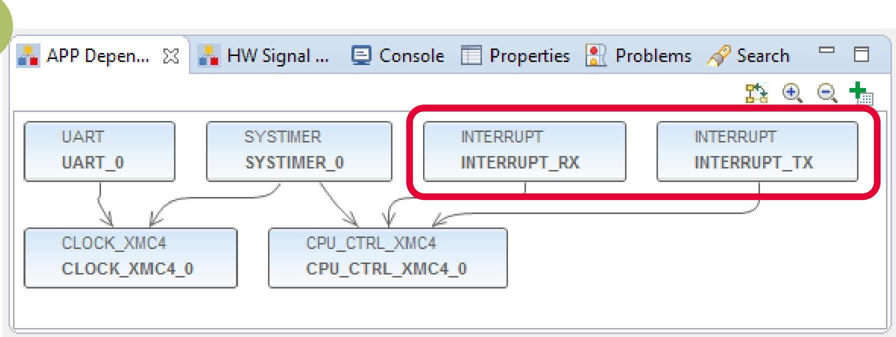
A total of 64 FIFO entries are available to be configured as transmit and receive buffers. These 64 entries are additionally shared between the two channels of a USIC module

1 UART APP is used to configure and implement the USIC driver.

2 Except „Transmit mode" and „Receive mode" all settings are the default UART APP settings. For Protocol Handling the setting „Direct" is used. This allows bytewise handling for receiving and transmitting of the FreeMODBUS library.

APP structure – UART configuration (2/4)

3



UART
UART_0

SYSTIMER
SYSTIMER_0

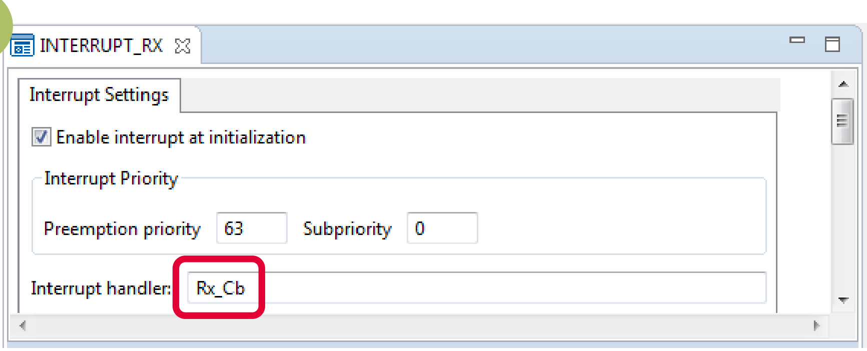
INTERRUPT
INTERRUPT_RX

INTERRUPT
INTERRUPT_TX

CLOCK_XMC4
CLOCK_XMC4_0

CPU_CTRL_XMC4
CPU_CTRL_XMC4_0

4



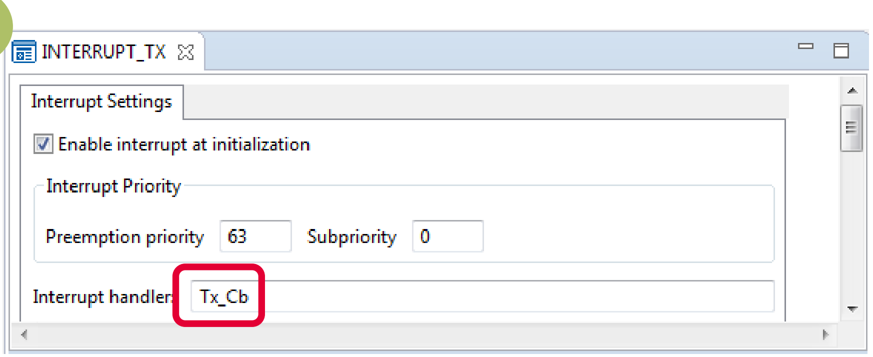
☑ Enable interrupt at initialization

Interrupt Priority

Preemption priority 63 Subpriority 0

Interrupt handler: Rx_Cb

5



☑ Enable interrupt at initialization

Interrupt Priority

Preemption priority 63 Subpriority 0

Interrupt handler: Tx_Cb

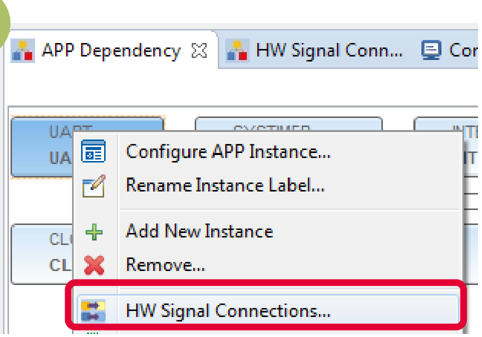
3 For each UART transmission direction an INTERRUPT APP is used.

4 Define callback name of receive interrupt. Implemented inside main.c

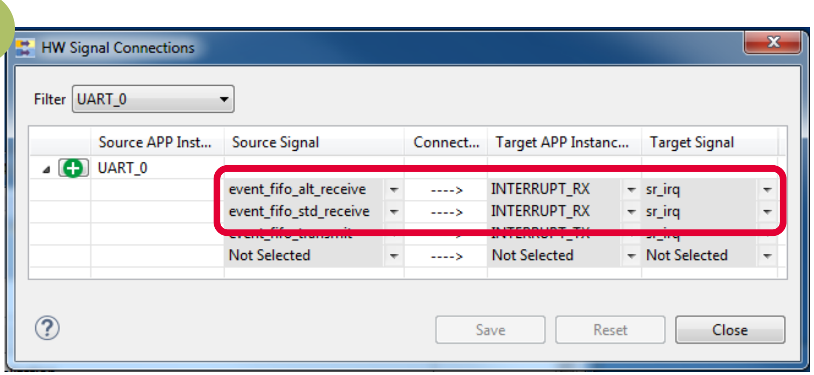
5 Define callback name of transmit interrupt. Implemented inside main.c

APP structure – UART configuration (3/4)

6



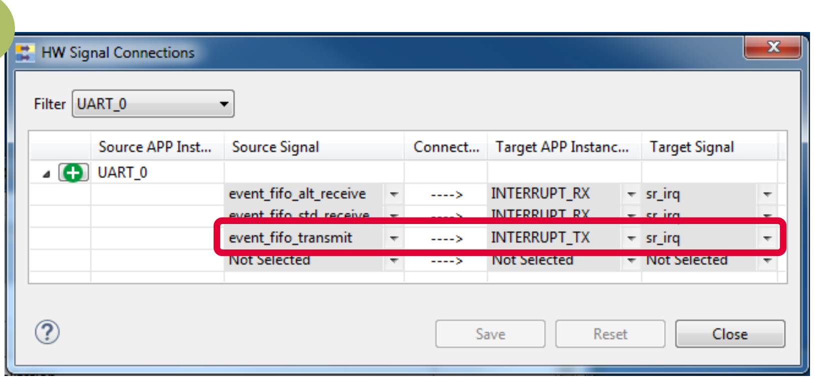
7



Filter UART_0

Source APP Inst...	Source Signal	Connect...	Target APP Instanc...	Target Signal
UART_0				
	event_fifo_alt_receive	---->	INTERRUPT_RX	sr_irq
	event_fifo_std_receive	---->	INTERRUPT_RX	sr_irq
	event_fifo_transmit	---->	INTERRUPT_TX	sr_irq
	Not Selected	---->	Not Selected	Not Selected

8



Filter UART_0

Source APP Inst...	Source Signal	Connect...	Target APP Instanc...	Target Signal
UART_0				
	event_fifo_alt_receive	---->	INTERRUPT_RX	sr_irq
	event_fifo_std_receive	---->	INTERRUPT_RX	sr_irq
	event_fifo_transmit	---->	INTERRUPT_TX	sr_irq
	Not Selected	---->	Not Selected	Not Selected

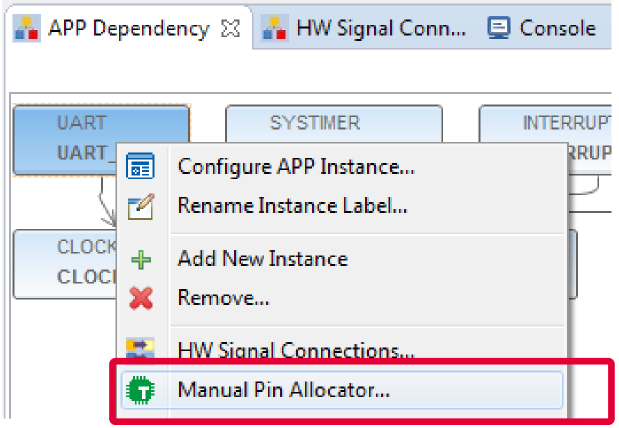
6 Right click on the UART APP. From the context menu select „HW Signal Connections…“

7 Connect all the fifo receive interrupt source signals to your receive INTERRUPT APP.

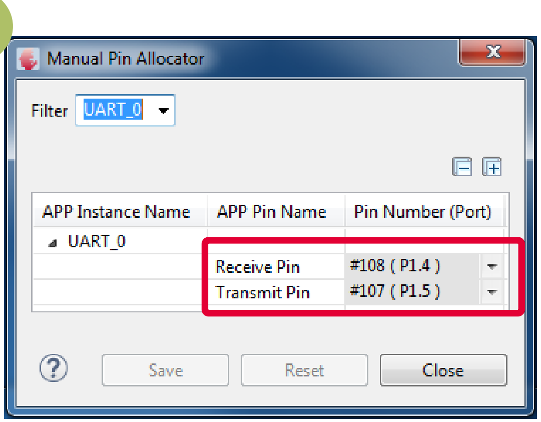
8 Connect all the fifo transmit interrupt source signals to your transmit INTERRUPT APP.

Pinning – UART configuration (4/4)

9



10



Filter UART_0

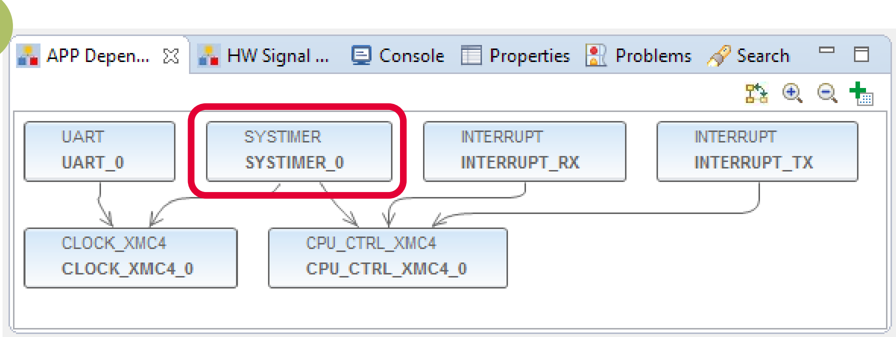
APP Instance Name	APP Pin Name	Pin Number (Port)
UART_0		
	Receive Pin	#108 (P1.4)
	Transmit Pin	#107 (P1.5)

9 Right click on the UART APP. From the context menu select „Manual Pin Allocator‟ to open the pin allocation for the uart module

10 Inside „Manual Pin Allocator‟ you can configure the uart pins for your application. For the example provided, we use those uart pins, which are connected to the VCOM of the debugger chip on XMC4700 / XMC 4800 Relax Kit

APP structure – SYSTIMER

1



UART
UART_0

SYSTIMER
SYSTIMER_0

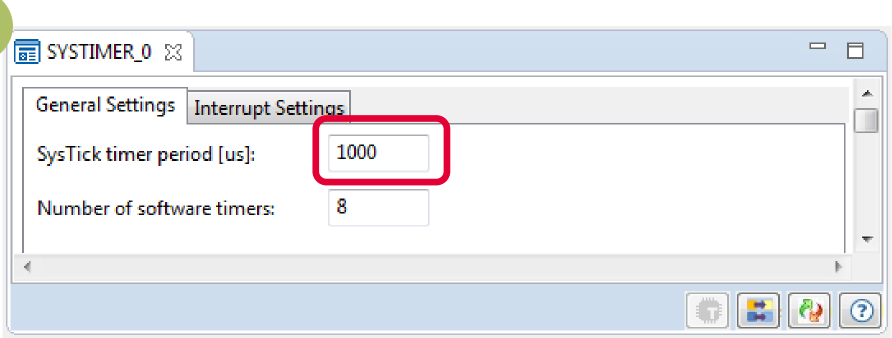
INTERRUPT
INTERRUPT_RX

INTERRUPT
INTERRUPT_TX

CLOCK_XMC4
CLOCK_XMC4_0

CPU_CTRL_XMC4
CPU_CTRL_XMC4_0

2



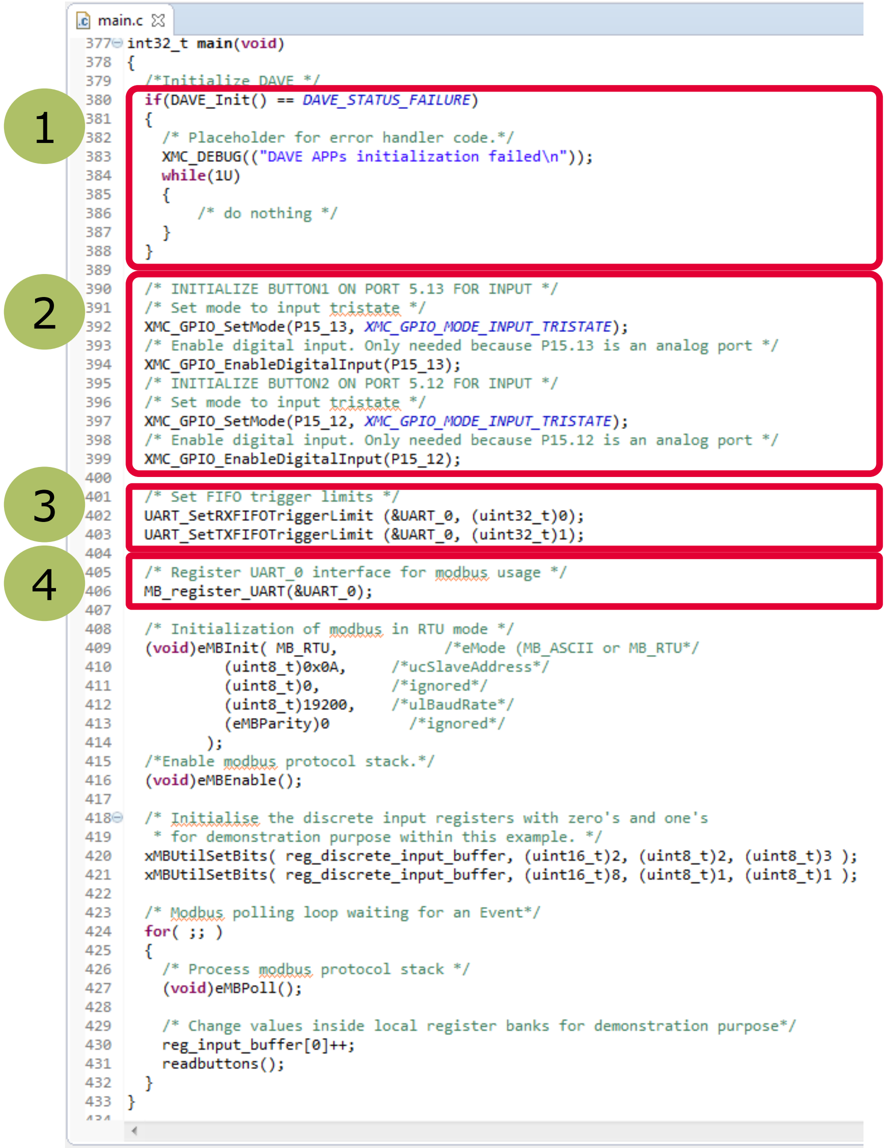
SysTick timer period [us]: 1000

Number of software timers: 8

1 SYSTIMER APP is used inside FreeMODBUS library to trigger timeout when receiving an incomplete request from master.

2 The SysTick timer period is used in it‘s default setting.

Application – Modbus® main application



```c
int32_t main(void)
{
  /*Initialize DAVE */
  if(DAVE_Init() == DAVE_STATUS_FAILURE)
  {
    /* Placeholder for error handler code.*/
    XMC_DEBUG(("DAVE APPs initialization failed\n"));
    while(1U)
    {
        /* do nothing */
    }
  }

  /* INITIALIZE BUTTON1 ON PORT 5.13 FOR INPUT */
  /* Set mode to input tristate */
  XMC_GPIO_SetMode(P15_13, XMC_GPIO_MODE_INPUT_TRISTATE);
  /* Enable digital input. Only needed because P15.13 is an analog port */
  XMC_GPIO_EnableDigitalInput(P15_13);
  /* INITIALIZE BUTTON2 ON PORT 5.12 FOR INPUT */
  /* Set mode to input tristate */
  XMC_GPIO_SetMode(P15_12, XMC_GPIO_MODE_INPUT_TRISTATE);
  /* Enable digital input. Only needed because P15.12 is an analog port */
  XMC_GPIO_EnableDigitalInput(P15_12);

  /* Set FIFO trigger limits */
  UART_SetRXFIFOTriggerLimit (&UART_0, (uint32_t)0);
  UART_SetTXFIFOTriggerLimit (&UART_0, (uint32_t)1);

  /* Register UART_0 interface for modbus usage */
  MB_register_UART(&UART_0);

  /* Initialization of modbus in RTU mode */
  (void)eMBInit( MB_RTU,            /*eMode (MB_ASCII or MB_RTU*/
          (uint8_t)0x0A,     /*ucSlaveAddress*/
          (uint8_t)0,        /*ignored*/
          (uint8_t)19200,    /*ulBaudRate*/
          (eMBParity)0         /*ignored*/
        );
  /*Enable modbus protocol stack.*/
  (void)eMBEnable();

  /* Initialise the discrete input registers with zero's and one's
   * for demonstration purpose within this example. */
  xMBUtilSetBits( reg_discrete_input_buffer, (uint16_t)2, (uint8_t)2, (uint8_t)3 );
  xMBUtilSetBits( reg_discrete_input_buffer, (uint16_t)8, (uint8_t)1, (uint8_t)1 );

  /* Modbus polling loop waiting for an Event*/
  for( ;; )
  {
    /* Process modbus protocol stack */
    (void)eMBPoll();

    /* Change values inside local register banks for demonstration purpose*/
    reg_input_buffer[0]++;
    readbuttons();
  }
}
```

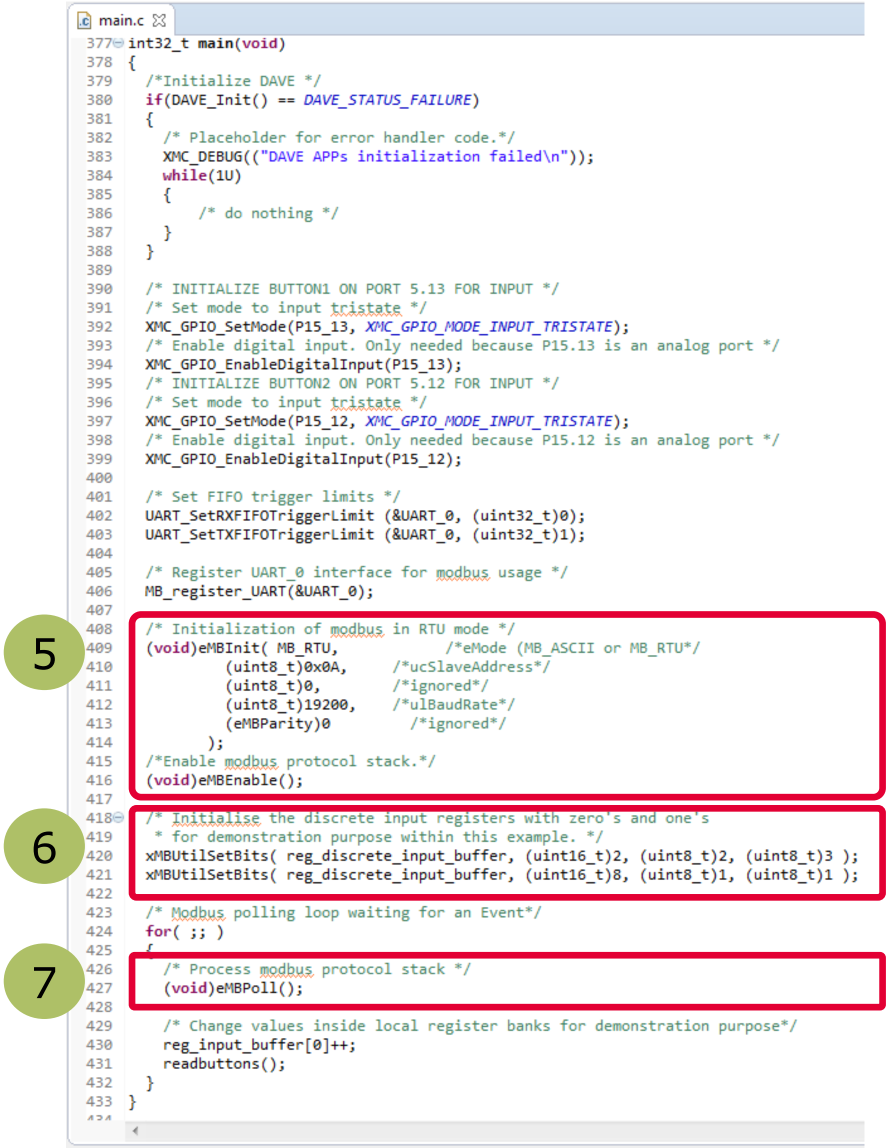
1 Initialize DAVE™ system including all APPs

2 Initialize the Relax Kit input ports of button1 and button2

3 Set uart fifo interrupt trigger levels: When first byte is received (fifo level changes from **0** to 1) and when last byte is transmitted (fifo level changes from **1** to 0)

4 Registering UART APP for use inside FreeMODBUS library

Application – Modbus® main application



```
int32_t main(void)
{
  /*Initialize DAVE */
  if(DAVE_Init() == DAVE_STATUS_FAILURE)
  {
    /* Placeholder for error handler code.*/
    XMC_DEBUG(("DAVE APPs initialization failed\n"));
    while(1U)
    {
        /* do nothing */
    }
  }

  /* INITIALIZE BUTTON1 ON PORT 5.13 FOR INPUT */
  /* Set mode to input tristate */
  XMC_GPIO_SetMode(P15_13, XMC_GPIO_MODE_INPUT_TRISTATE);
  /* Enable digital input. Only needed because P15.13 is an analog port */
  XMC_GPIO_EnableDigitalInput(P15_13);
  /* INITIALIZE BUTTON2 ON PORT 5.12 FOR INPUT */
  /* Set mode to input tristate */
  XMC_GPIO_SetMode(P15_12, XMC_GPIO_MODE_INPUT_TRISTATE);
  /* Enable digital input. Only needed because P15.12 is an analog port */
  XMC_GPIO_EnableDigitalInput(P15_12);

  /* Set FIFO trigger limits */
  UART_SetRXFIFOTriggerLimit (&UART_0, (uint32_t)0);
  UART_SetTXFIFOTriggerLimit (&UART_0, (uint32_t)1);

  /* Register UART_0 interface for modbus usage */
  MB_register_UART(&UART_0);

  /* Initialization of modbus in RTU mode */
  (void)eMBInit( MB_RTU,                /*eMode (MB_ASCII or MB_RTU*/
           (uint8_t)0x0A,       /*ucSlaveAddress*/
           (uint8_t)0,          /*ignored*/
           (uint8_t)19200,      /*ulBaudRate*/
           (eMBParity)0           /*ignored*/
         );
  /*Enable modbus protocol stack.*/
  (void)eMBEnable();

  /* Initialise the discrete input registers with zero's and one's
   * for demonstration purpose within this example. */
  xMBUtilSetBits( reg_discrete_input_buffer, (uint16_t)2, (uint8_t)2, (uint8_t)3 );
  xMBUtilSetBits( reg_discrete_input_buffer, (uint16_t)8, (uint8_t)1, (uint8_t)1 );

  /* Modbus polling loop waiting for an Event*/
  for( ;; )
  {
    /* Process modbus protocol stack */
    (void)eMBPoll();

    /* Change values inside local register banks for demonstration purpose*/
    reg_input_buffer[0]++;
    readbuttons();
  }
}
```

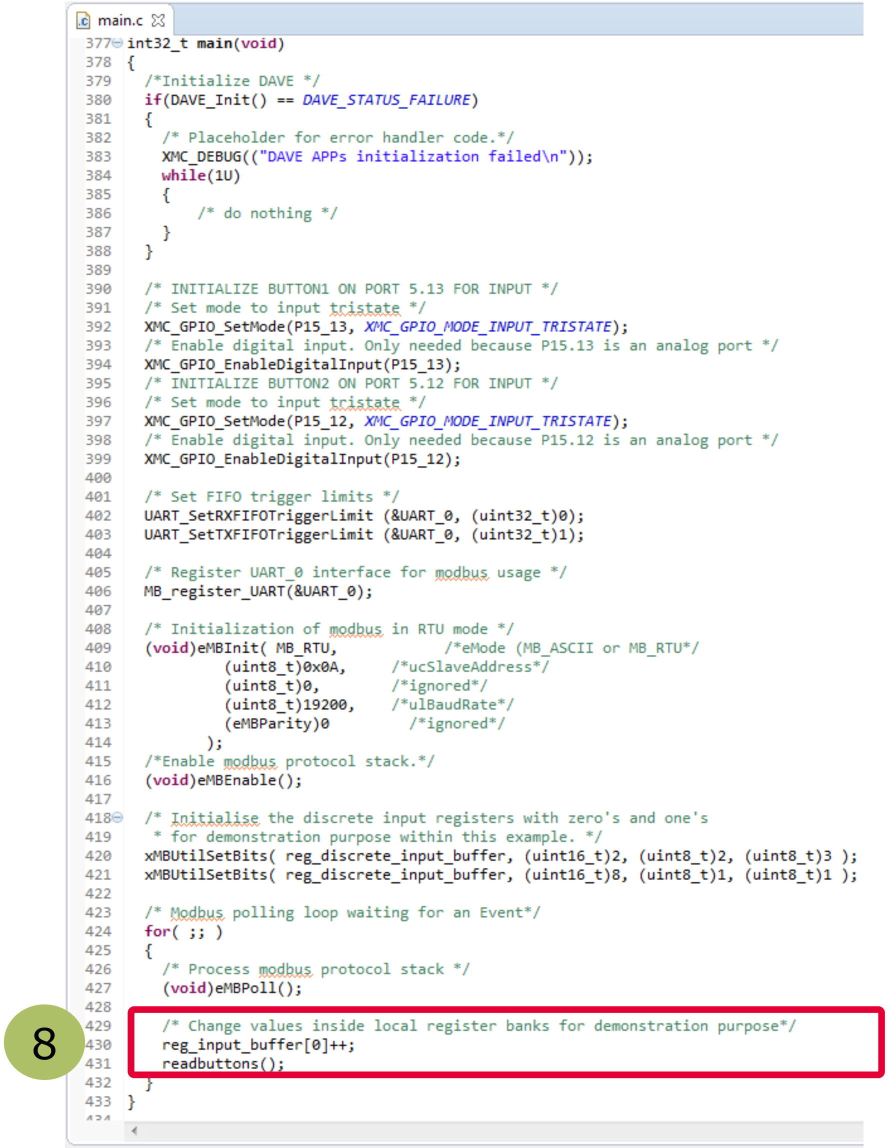
5 Initialize Modbus®:
- RTU or ASCII mode
- slave address
- baudrate

Enable protocol stack

6 Initialize some of the local discrete input buffers. Only for demonstration purpose inside this example

7 Poll ModBus® protocol stack

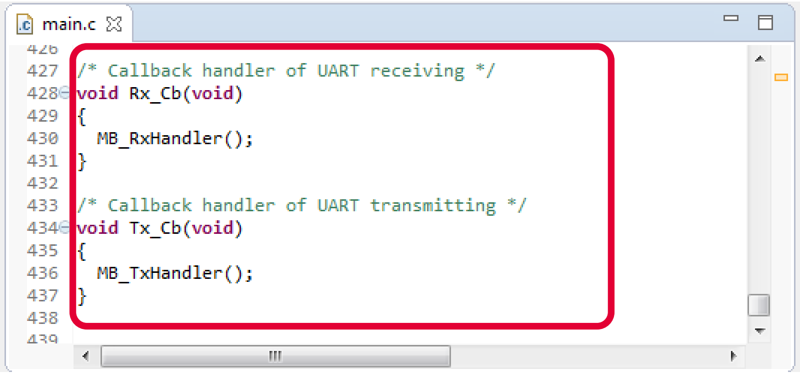
Application – Modbus® main application



```c
int32_t main(void)
{
  /*Initialize DAVE */
  if(DAVE_Init() == DAVE_STATUS_FAILURE)
  {
    /* Placeholder for error handler code.*/
    XMC_DEBUG(("DAVE APPs initialization failed\n"));
    while(1U)
    {
        /* do nothing */
    }
  }

  /* INITIALIZE BUTTON1 ON PORT 5.13 FOR INPUT */
  /* Set mode to input tristate */
  XMC_GPIO_SetMode(P15_13, XMC_GPIO_MODE_INPUT_TRISTATE);
  /* Enable digital input. Only needed because P15.13 is an analog port */
  XMC_GPIO_EnableDigitalInput(P15_13);
  /* INITIALIZE BUTTON2 ON PORT 5.12 FOR INPUT */
  /* Set mode to input tristate */
  XMC_GPIO_SetMode(P15_12, XMC_GPIO_MODE_INPUT_TRISTATE);
  /* Enable digital input. Only needed because P15.12 is an analog port */
  XMC_GPIO_EnableDigitalInput(P15_12);

  /* Set FIFO trigger limits */
  UART_SetRXFIFOTriggerLimit (&UART_0, (uint32_t)0);
  UART_SetTXFIFOTriggerLimit (&UART_0, (uint32_t)1);

  /* Register UART_0 interface for modbus usage */
  MB_register_UART(&UART_0);

  /* Initialization of modbus in RTU mode */
  (void)eMBInit( MB_RTU,             /*eMode (MB_ASCII or MB_RTU*/
          (uint8_t)0x0A,     /*ucSlaveAddress*/
          (uint8_t)0,        /*ignored*/
          (uint8_t)19200,    /*ulBaudRate*/
          (eMBParity)0         /*ignored*/
        );
  /*Enable modbus protocol stack.*/
  (void)eMBEnable();

  /* Initialise the discrete input registers with zero's and one's
   * for demonstration purpose within this example. */
  xMBUtilSetBits( reg_discrete_input_buffer, (uint16_t)2, (uint8_t)2, (uint8_t)3 );
  xMBUtilSetBits( reg_discrete_input_buffer, (uint16_t)8, (uint8_t)1, (uint8_t)1 );

  /* Modbus polling loop waiting for an Event*/
  for( ;; )
  {
    /* Process modbus protocol stack */
    (void)eMBPoll();

    /* Change values inside local register banks for demonstration purpose*/
    reg_input_buffer[0]++;
    readbuttons();
  }
}
```

8 Increment with every cycle the first entry of the local 16bit input buffer.
Inside readbuttons():
Read button1 and button2 state and write button state to bit 0 and bit 1 of discrete input buffer.
Only for demonstration purpose inside this example

Application – FIFO interrupt handling

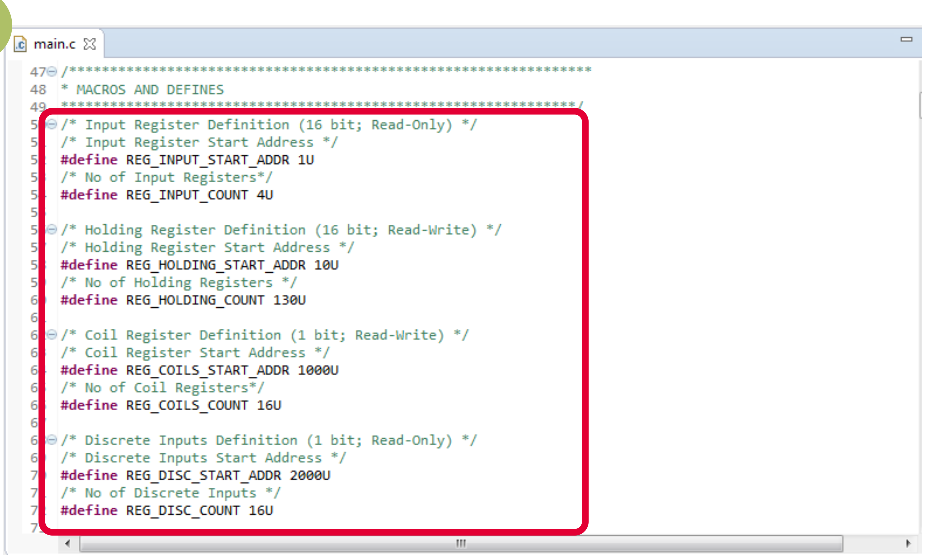


```
/* Callback handler of UART receiving */
void Rx_Cb(void)
{
  MB_RxHandler();
}

/* Callback handler of UART transmitting */
void Tx_Cb(void)
{
  MB_TxHandler();
}
```

Fifo level interrupts have been configured to trigger when first byte was received and when last byte was transmitted.
Here the interrupt handlers MB_RxHandler and MB_TxHandler of FreeMODBUS library port are called to process the ModBus® protocol

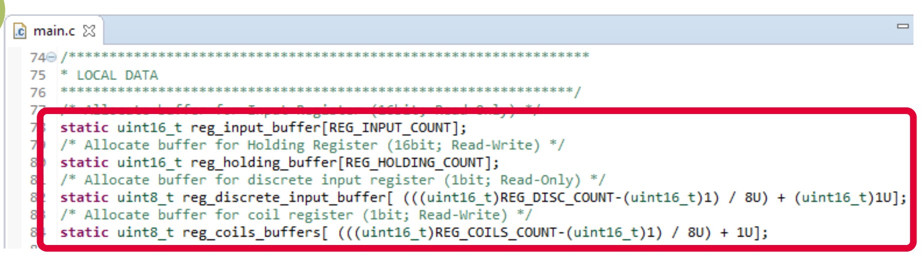
Application – Defining local register banks

1



```
/****************************************************************
 * MACROS AND DEFINES
 ***************************************************************/
/* Input Register Definition (16 bit; Read-Only) */
/* Input Register Start Address */
#define REG_INPUT_START_ADDR 1U
/* No of Input Registers*/
#define REG_INPUT_COUNT 4U

/* Holding Register Definition (16 bit; Read-Write) */
/* Holding Register Start Address */
#define REG_HOLDING_START_ADDR 10U
/* No of Holding Registers */
#define REG_HOLDING_COUNT 130U

/* Coil Register Definition (1 bit; Read-Write) */
/* Coil Register Start Address */
#define REG_COILS_START_ADDR 1000U
/* No of Coil Registers*/
#define REG_COILS_COUNT 16U

/* Discrete Inputs Definition (1 bit; Read-Only) */
/* Discrete Inputs Start Address */
#define REG_DISC_START_ADDR 2000U
/* No of Discrete Inputs */
#define REG_DISC_COUNT 16U
```

2

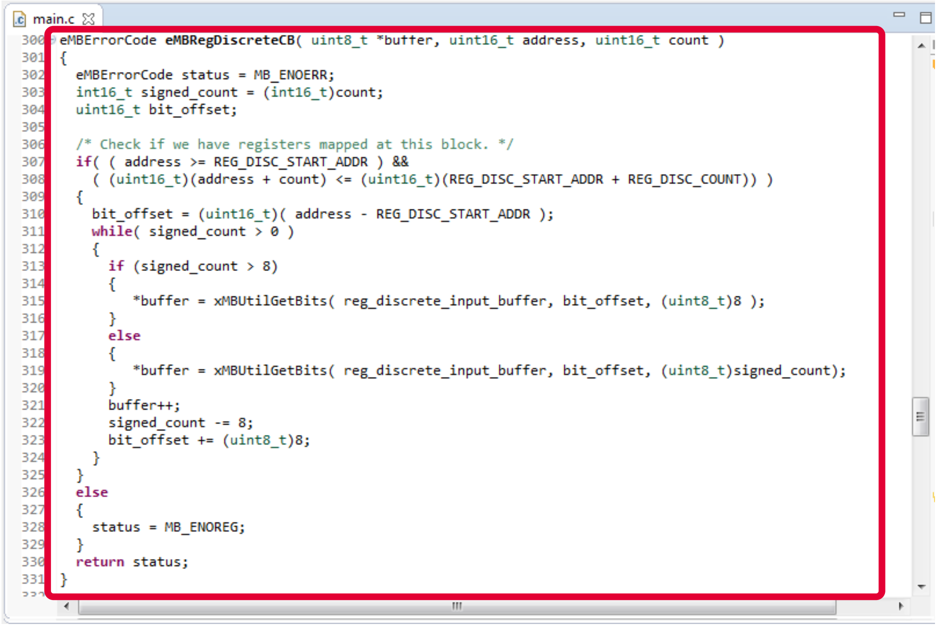


```
/****************************************************************
 * LOCAL DATA
 ***************************************************************/
/* Allocate buffer for Input Register (16bit; Read-Only) */
static uint16_t reg_input_buffer[REG_INPUT_COUNT];
/* Allocate buffer for Holding Register (16bit; Read-Write) */
static uint16_t reg_holding_buffer[REG_HOLDING_COUNT];
/* Allocate buffer for discrete input register (1bit; Read-Only) */
static uint8_t reg_discrete_input_buffer[ (((uint16_t)REG_DISC_COUNT-(uint16_t)1) / 8U) + (uint16_t)1U];
/* Allocate buffer for coil register (1bit; Read-Write) */
static uint8_t reg_coils_buffers[ (((uint16_t)REG_COILS_COUNT-(uint16_t)1) / 8U) + 1U];
```

1 Definition of size and start addresses of local register banks:
input registers (16 bit;RO)
holding registers (16 bit;RW)
coil registers (1 bit;RW)
discrete input registers (1 bit;RO)

2 Allocation of memory for Modbus® register banks as global variables with local scope (static).

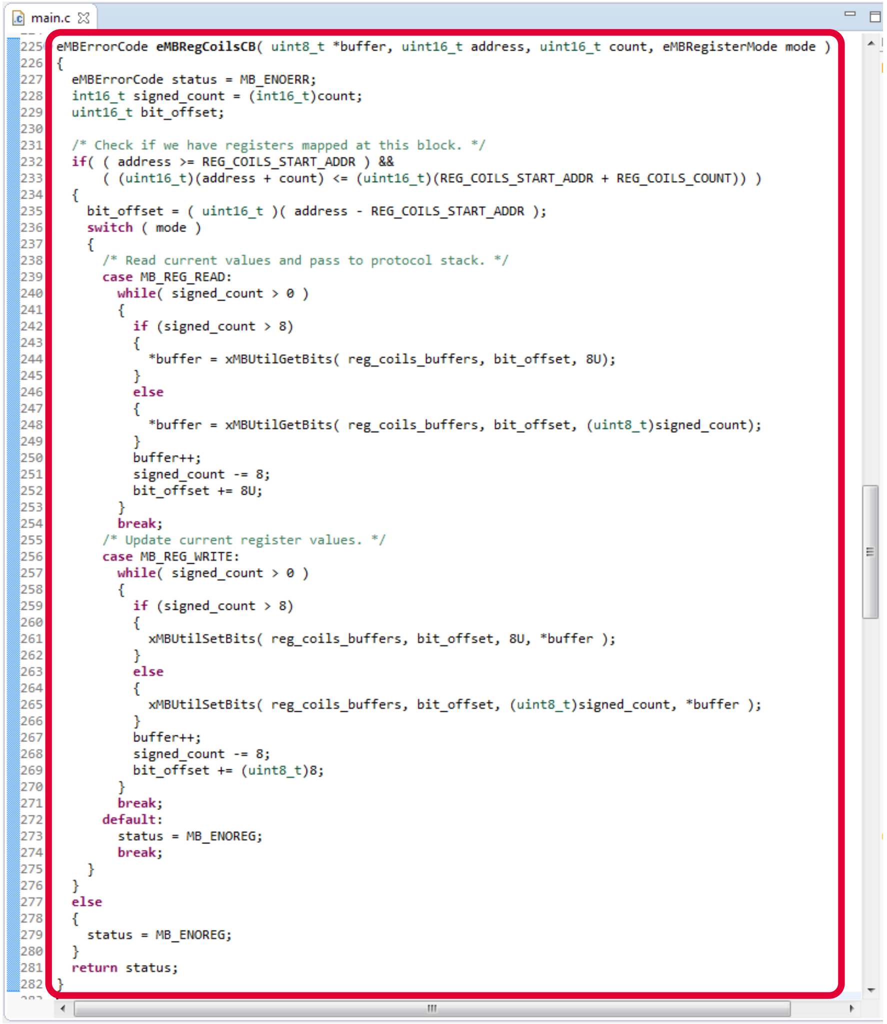
Application – Discrete input registers callback



```
eMBErrorCode eMBRegDiscreteCB( uint8_t *buffer, uint16_t address, uint16_t count )
{
  eMBErrorCode status = MB_ENOERR;
  int16_t signed_count = (int16_t)count;
  uint16_t bit_offset;

  /* Check if we have registers mapped at this block. */
  if( ( address >= REG_DISC_START_ADDR ) &&
    ( (uint16_t)(address + count) <= (uint16_t)(REG_DISC_START_ADDR + REG_DISC_COUNT)) )
  {
    bit_offset = (uint16_t)( address - REG_DISC_START_ADDR );
    while( signed_count > 0 )
    {
      if (signed_count > 8)
      {
         *buffer = xMBUtilGetBits( reg_discrete_input_buffer, bit_offset, (uint8_t)8 );
      }
      else
      {
         *buffer = xMBUtilGetBits( reg_discrete_input_buffer, bit_offset, (uint8_t)signed_count);
      }
      buffer++;
      signed_count -= 8;
      bit_offset += (uint8_t)8;
    }
  }
  else
  {
    status = MB_ENOREG;
  }
  return status;
}
```

Callback which is called from Modbus® protocol stack to read local register banks of 1 bit discrete input registers

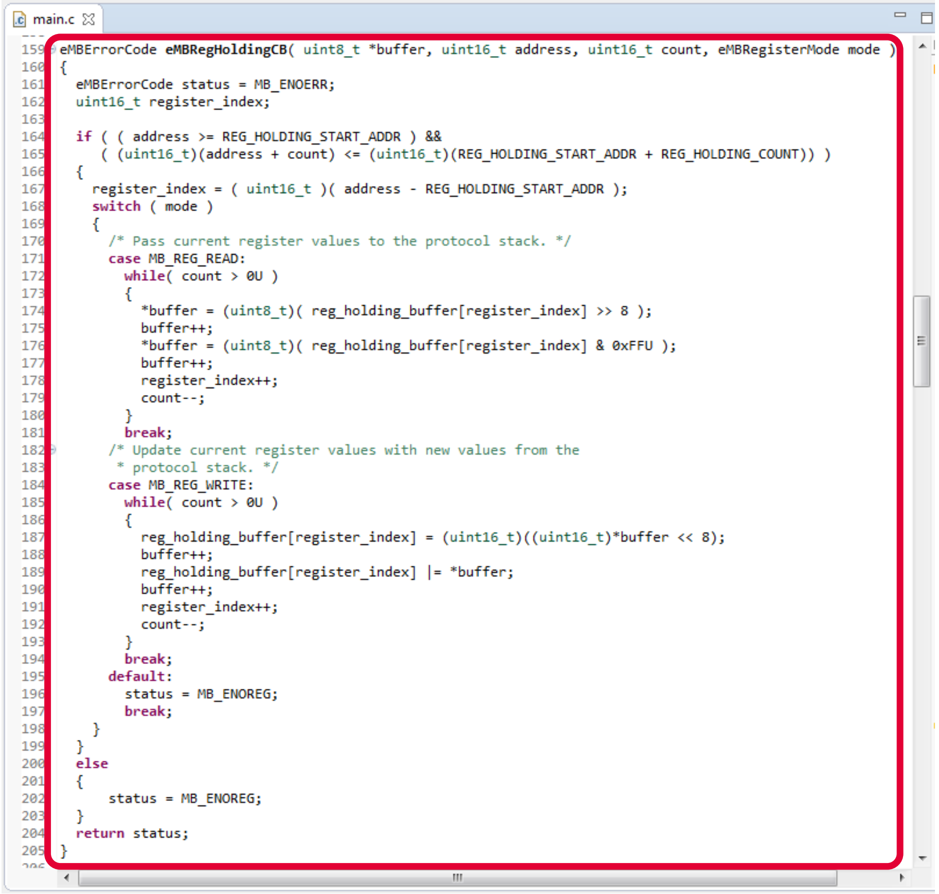
Application – Coil registers callback



```c
eMBErrorCode eMBRegCoilsCB( uint8_t *buffer, uint16_t address, uint16_t count, eMBRegisterMode mode )
{
  eMBErrorCode status = MB_ENOERR;
  int16_t signed_count = (int16_t)count;
  uint16_t bit_offset;

  /* Check if we have registers mapped at this block. */
  if( ( address >= REG_COILS_START_ADDR ) &&
      ( (uint16_t)(address + count) <= (uint16_t)(REG_COILS_START_ADDR + REG_COILS_COUNT)) )
  {
    bit_offset = ( uint16_t )( address - REG_COILS_START_ADDR );
    switch ( mode )
    {
      /* Read current values and pass to protocol stack. */
      case MB_REG_READ:
        while( signed_count > 0 )
        {
          if (signed_count > 8)
          {
            *buffer = xMBUtilGetBits( reg_coils_buffers, bit_offset, 8U);
          }
          else
          {
            *buffer = xMBUtilGetBits( reg_coils_buffers, bit_offset, (uint8_t)signed_count);
          }
          buffer++;
          signed_count -= 8;
          bit_offset += 8U;
        }
        break;
      /* Update current register values. */
      case MB_REG_WRITE:
        while( signed_count > 0 )
        {
          if (signed_count > 8)
          {
            xMBUtilSetBits( reg_coils_buffers, bit_offset, 8U, *buffer );
          }
          else
          {
            xMBUtilSetBits( reg_coils_buffers, bit_offset, (uint8_t)signed_count, *buffer );
          }
          buffer++;
          signed_count -= 8;
          bit_offset += (uint8_t)8;
        }
        break;
      default:
        status = MB_ENOREG;
        break;
    }
  }
  else
  {
    status = MB_ENOREG;
  }
  return status;
}
```

Callback which is called from Modbus® protocol stack to read and write local register banks of 1 bit coil input/output registers

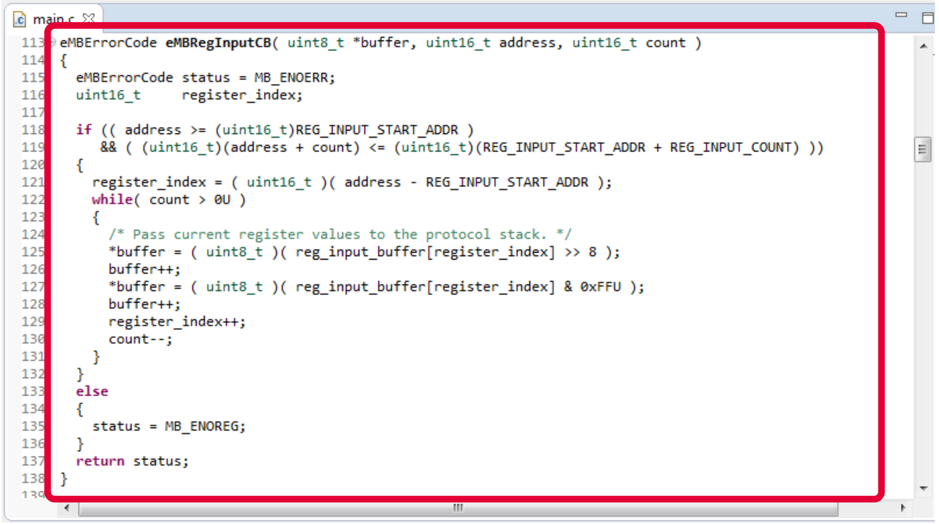
Application – Holding registers callback



```c
eMBErrorCode eMBRegHoldingCB( uint8_t *buffer, uint16_t address, uint16_t count, eMBRegisterMode mode )
{
  eMBErrorCode status = MB_ENOERR;
  uint16_t register_index;

  if ( ( address >= REG_HOLDING_START_ADDR ) &&
     ( (uint16_t)(address + count) <= (uint16_t)(REG_HOLDING_START_ADDR + REG_HOLDING_COUNT)) )
  {
    register_index = ( uint16_t )( address - REG_HOLDING_START_ADDR );
    switch ( mode )
    {
      /* Pass current register values to the protocol stack. */
      case MB_REG_READ:
        while( count > 0U )
        {
          *buffer = (uint8_t)( reg_holding_buffer[register_index] >> 8 );
          buffer++;
          *buffer = (uint8_t)( reg_holding_buffer[register_index] & 0xFFU );
          buffer++;
          register_index++;
          count--;
        }
        break;
      /* Update current register values with new values from the
       * protocol stack. */
      case MB_REG_WRITE:
        while( count > 0U )
        {
          reg_holding_buffer[register_index] = (uint16_t)((uint16_t)*buffer << 8);
          buffer++;
          reg_holding_buffer[register_index] |= *buffer;
          buffer++;
          register_index++;
          count--;
        }
        break;
      default:
        status = MB_ENOREG;
        break;
    }
  }
  else
  {
    status = MB_ENOREG;
  }
  return status;
}
```

Callback which is called from Modbus® protocol stack to read and write local register banks of 16 bit holding input/output registers

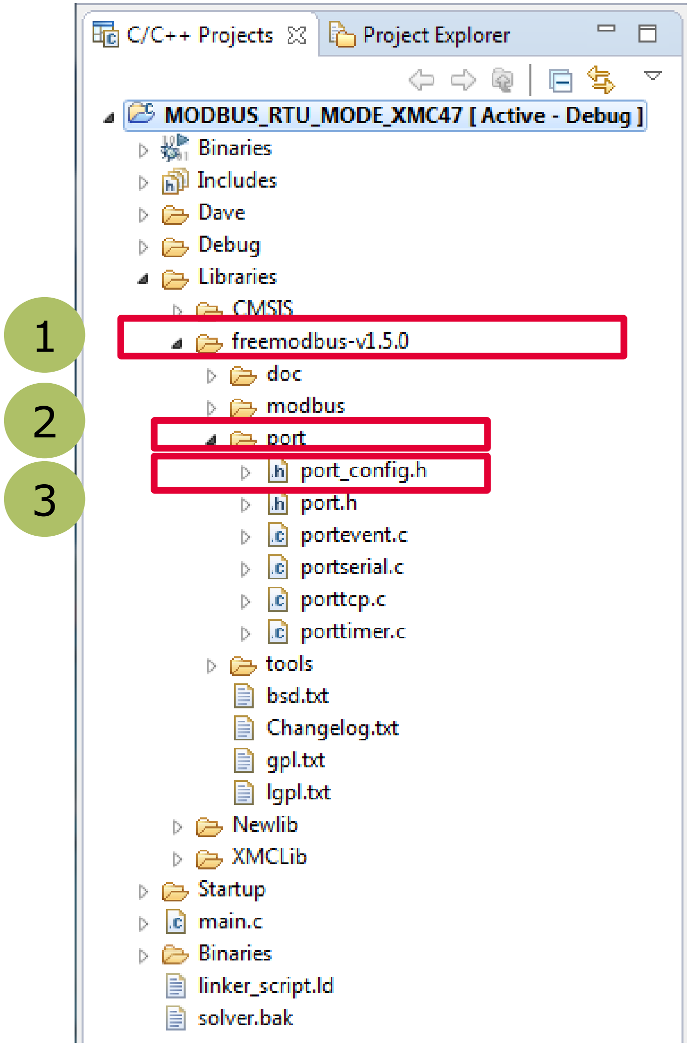
Application – Input registers callback



```c
eMBErrorCode eMBRegInputCB( uint8_t *buffer, uint16_t address, uint16_t count )
{
  eMBErrorCode status = MB_ENOERR;
  uint16_t     register_index;

  if (( address >= (uint16_t)REG_INPUT_START_ADDR )
     && ( (uint16_t)(address + count) <= (uint16_t)(REG_INPUT_START_ADDR + REG_INPUT_COUNT) ))
  {
    register_index = ( uint16_t )( address - REG_INPUT_START_ADDR );
    while( count > 0U )
    {
      /* Pass current register values to the protocol stack. */
      *buffer = ( uint8_t )( reg_input_buffer[register_index] >> 8 );
      buffer++;
      *buffer = ( uint8_t )( reg_input_buffer[register_index] & 0xFFU );
      buffer++;
      register_index++;
      count--;
    }
  }
  else
  {
    status = MB_ENOREG;
  }
  return status;
}
```

Callback which is called from Modbus® protocol stack to read local register banks of 16 bit input registers

Library – FreeMODBUS v1.5.0 protocol stack



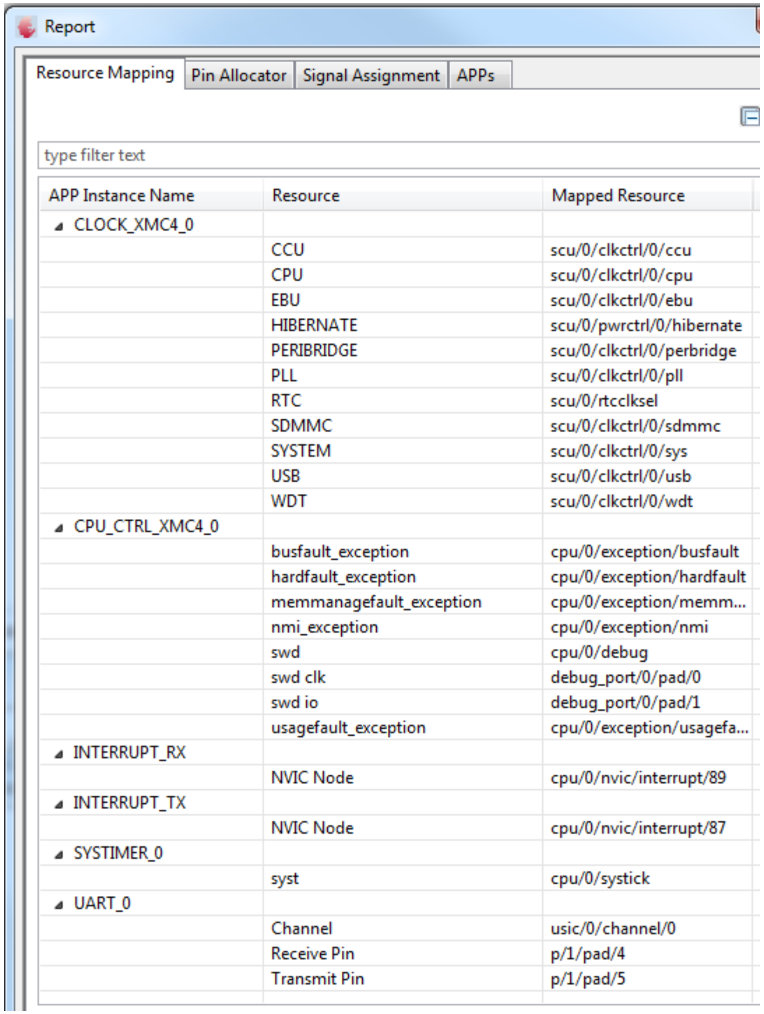
```
MODBUS_RTU_MODE_XMC47 [ Active - Debug ]
    Binaries
    Includes
    Dave
    Debug
    Libraries
        CMSIS
        freemodbus-v1.5.0
            doc
            modbus
            port
                port_config.h
                port.h
                portevent.c
                portserial.c
                porttcp.c
                porttimer.c
            tools
            bsd.txt
            Changelog.txt
            gpl.txt
            lgpl.txt
        Newlib
        XMCLib
    Startup
    main.c
    Binaries
    linker_script.ld
    solver.bak
```

1 The FreeMODBUS protocol stack library is located inside the Libraries folder

2 Source code for porting the library to XMC4000 family is located inside port folder

3 You can modify port_config.h for customized configurations. For example here you can disable modules (ASCII or RTU) of the library to reduce code size.

Application – Overview on used HW resources



APP Instance Name	Resource	Mapped Resource
CLOCK_XMC4_0		
	CCU	scu/0/clkctrl/0/ccu
	CPU	scu/0/clkctrl/0/cpu
	EBU	scu/0/clkctrl/0/ebu
	HIBERNATE	scu/0/pwrctrl/0/hibernate
	PERIBRIDGE	scu/0/clkctrl/0/perbridge
	PLL	scu/0/clkctrl/0/pll
	RTC	scu/0/rtcclksel
	SDMMC	scu/0/clkctrl/0/sdmmc
	SYSTEM	scu/0/clkctrl/0/sys
	USB	scu/0/clkctrl/0/usb
	WDT	scu/0/clkctrl/0/wdt
CPU_CTRL_XMC4_0		
	busfault_exception	cpu/0/exception/busfault
	hardfault_exception	cpu/0/exception/hardfault
	memmanagefault_exception	cpu/0/exception/memm...
	nmi_exception	cpu/0/exception/nmi
	swd	cpu/0/debug
	swd clk	debug_port/0/pad/0
	swd io	debug_port/0/pad/1
	usagefault_exception	cpu/0/exception/usagefa...
INTERRUPT_RX		
	NVIC Node	cpu/0/nvic/interrupt/89
INTERRUPT_TX		
	NVIC Node	cpu/0/nvic/interrupt/87
SYSTIMER_0		
	syst	cpu/0/systick
UART_0		
	Channel	usic/0/channel/0
	Receive Pin	p/1/pad/4
	Transmit Pin	p/1/pad/5

1 Overview and Requirements

2 Setup

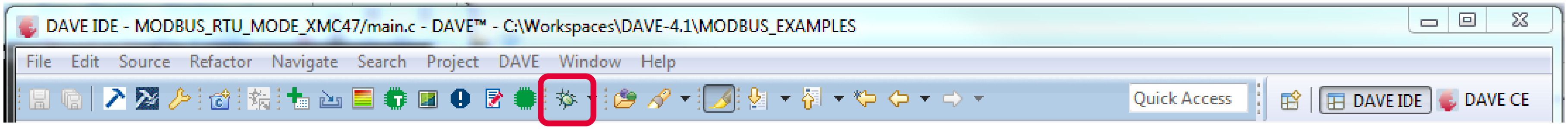
3 Implementing the application

4 How to test

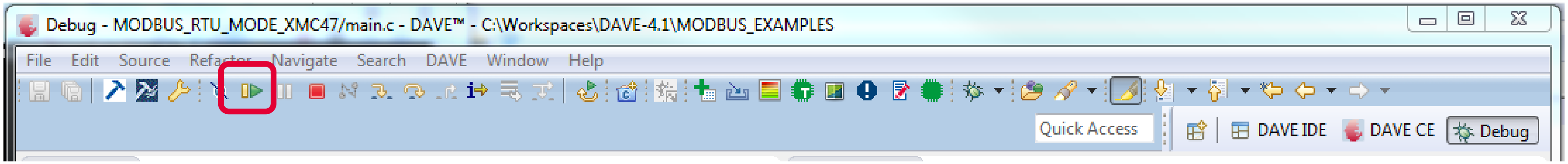
How to test – Build and start the Modbus® client

ACTIONS

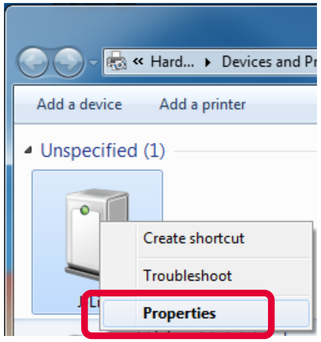
1. Build and download the example application software to the XMC4700 / XMC4800 Relax Kit and start the debugger


DAVE IDE - MODBUS_RTU_MODE_XMC47/main.c - DAVE™ - C:\Workspaces\DAVE-4.1\MODBUS_EXAMPLES
File
Edit
Source
Refactor
Navigate
Search
Project
DAVE
Window
Help
Quick Access
DAVE IDE
DAVE CE

2. Start the software by the run button


Debug - MODBUS_RTU_MODE_XMC47/main.c - DAVE™ - C:\Workspaces\DAVE-4.1\MODBUS_EXAMPLES
File
Edit
Source
Refactor
Navigate
Search
DAVE
Window
Help
Quick Access
DAVE IDE
DAVE CE
Debug

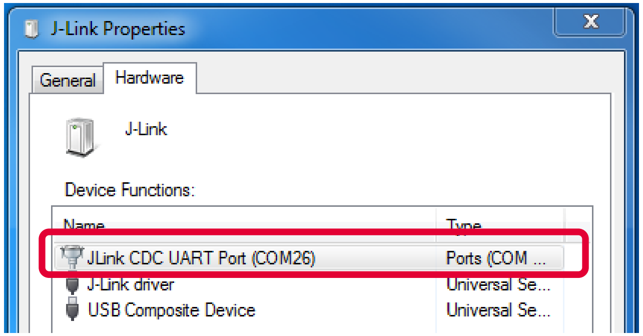
3. START >> Devices and Printers
Open context menu of J-Link Device and select „Properties"



How to test – Build and start the Modbus® client

OBSERVATIONS

Check the COM port number inside the J-Link properties
→ Here it is COM26



J-Link Properties

J-Link

Device Functions:

Name	Type
JLink CDC UART Port (COM26)	Ports (COM ...
J-Link driver	Universal Se...
USB Composite Device	Universal Se...

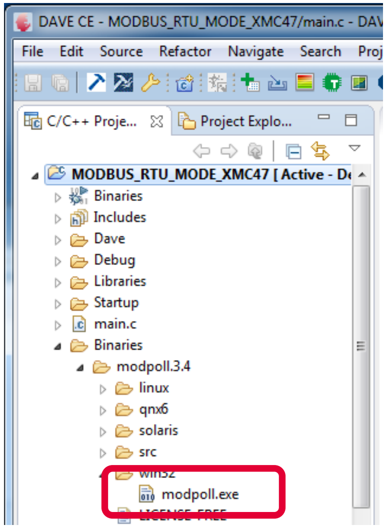
ATTENTION: In the proceeding of this documentation „COM26“ will be used. For your own testing, make sure to replace „COM26“ with the COM port number you have found here

How to test – Read 16 bit input registers in RTU mode

ACTIONS

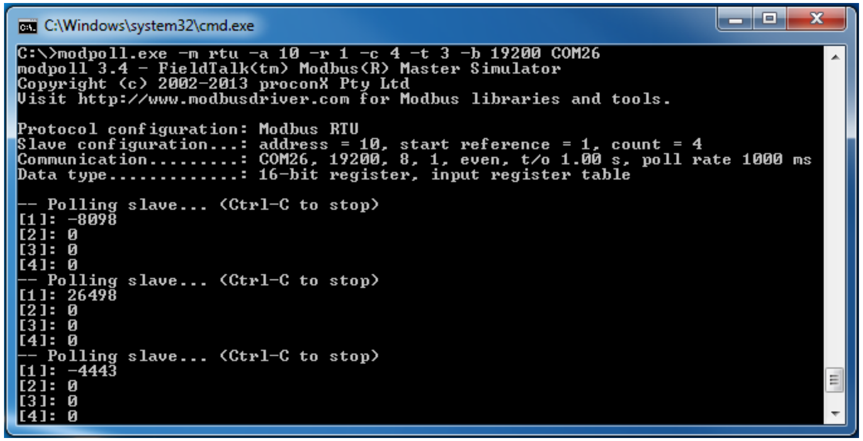
Use „Modpoll Modbus Master Simulator“ which is provided inside example project to poll input registers 1 to 4:

```
modpoll.exe -m rtu -a 10 -r 1 -c 4 -t 3 -b 19200 COM26
```



OBSERVATIONS

Input registers 1 to 4 are polled every second by master.
The value of register 1 is random non-static, because it is incremented inside the endless loop of the example application



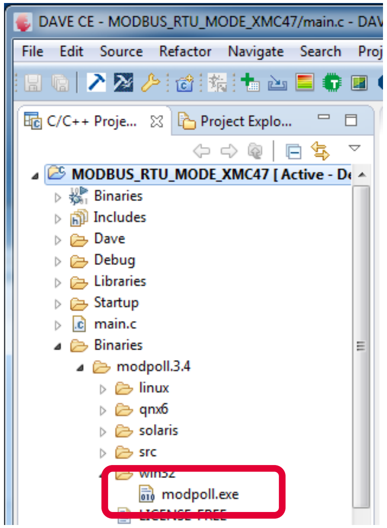
```
C:\>modpoll.exe -m rtu -a 10 -r 1 -c 4 -t 3 -b 19200 COM26
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: Modbus RTU
Slave configuration...: address = 10, start reference = 1, count = 4
Communication.........: COM26, 19200, 8, 1, even, t/o 1.00 s, poll rate 1000 ms
Data type.............: 16-bit register, input register table

-- Polling slave... (Ctrl-C to stop)
[1]: -8098
[2]: 0
[3]: 0
[4]: 0
-- Polling slave... (Ctrl-C to stop)
[1]: 26498
[2]: 0
[3]: 0
[4]: 0
-- Polling slave... (Ctrl-C to stop)
[1]: -4443
[2]: 0
[3]: 0
[4]: 0
```

How to test – Read invalid 16 bit input registers in RTU mode

ACTIONS

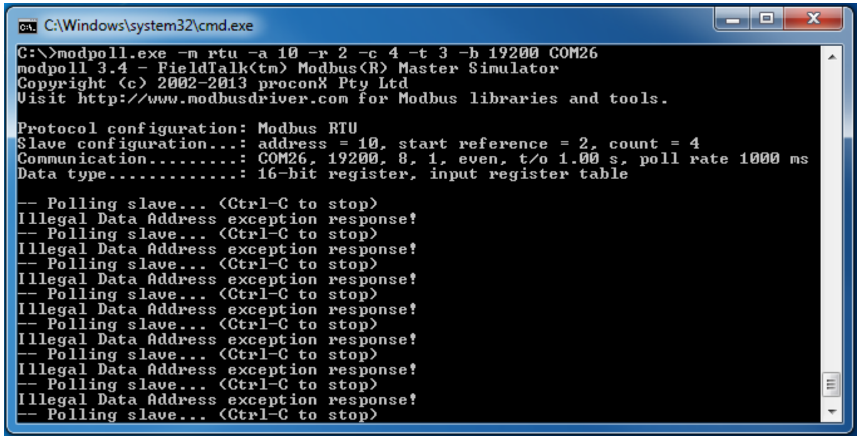
Use „Modpoll Modbus Master Simulator“ which is provided inside example project to poll input registers 2 to 5:

```
modpoll.exe -m rtu -a 10 -r 2 -c 4 -t 3 -b 19200 COM26
```



OBSERVATIONS

Modbus® client returns error code because register address 5 is not defined inside example application



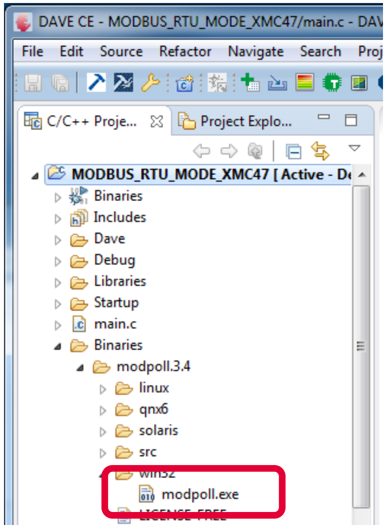
```
C:\>modpoll.exe -m rtu -a 10 -r 2 -c 4 -t 3 -b 19200 COM26
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: Modbus RTU
Slave configuration...: address = 10, start reference = 2, count = 4
Communication.........: COM26, 19200, 8, 1, even, t/o 1.00 s, poll rate 1000 ms
Data type.............: 16-bit register, input register table

-- Polling slave... (Ctrl-C to stop)
Illegal Data Address exception response!
-- Polling slave... (Ctrl-C to stop)
Illegal Data Address exception response!
-- Polling slave... (Ctrl-C to stop)
Illegal Data Address exception response!
-- Polling slave... (Ctrl-C to stop)
Illegal Data Address exception response!
-- Polling slave... (Ctrl-C to stop)
Illegal Data Address exception response!
-- Polling slave... (Ctrl-C to stop)
Illegal Data Address exception response!
-- Polling slave... (Ctrl-C to stop)
Illegal Data Address exception response!
-- Polling slave... (Ctrl-C to stop)
```

How to test – Read 1 bit discrete input registers in RTU mode

ACTIONS

1. Use „Modpoll Modbus Master Simulator" which is provided inside example project to poll discrete input bits on address 2000 to 2001:

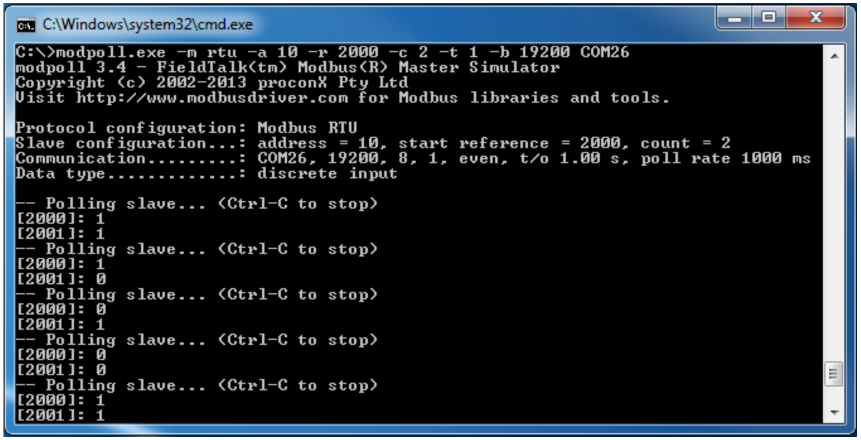
```
modpoll.exe -m rtu -a 10 -r 2000 -c 2 -t 1 -b 19200 COM26
```



2. Push button1 and button2 while polling

OBSERVATIONS

Discrete input registers on address 2000 to 2001 are polled every second. Their value reflects the state of button1 and button2



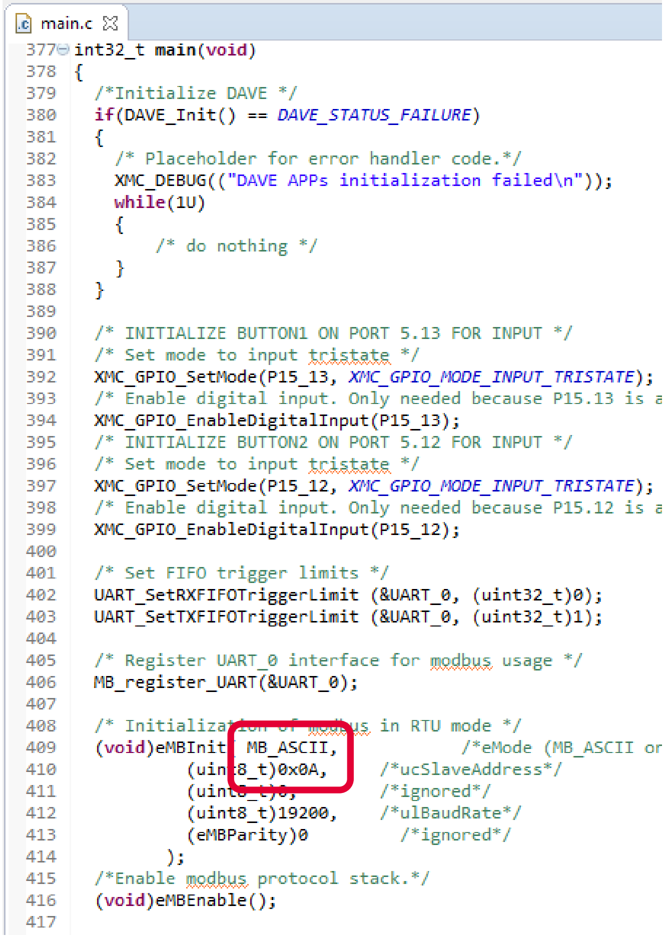
```
C:\>modpoll.exe -m rtu -a 10 -r 2000 -c 2 -t 1 -b 19200 COM26
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: Modbus RTU
Slave configuration...: address = 10, start reference = 2000, count = 2
Communication.........: COM26, 19200, 8, 1, even, t/o 1.00 s, poll rate 1000 ms
Data type.............: discrete input

-- Polling slave... (Ctrl-C to stop)
[2000]: 1
[2001]: 1
-- Polling slave... (Ctrl-C to stop)
[2000]: 1
[2001]: 0
-- Polling slave... (Ctrl-C to stop)
[2000]: 0
[2001]: 1
-- Polling slave... (Ctrl-C to stop)
[2000]: 0
[2001]: 0
-- Polling slave... (Ctrl-C to stop)
[2000]: 1
[2001]: 1
```

How to test – Switch from RTU to ASCII mode

ACTIONS

1. Configure the example application for ASCII mode

2. Rebuild and download the example to target



main.c

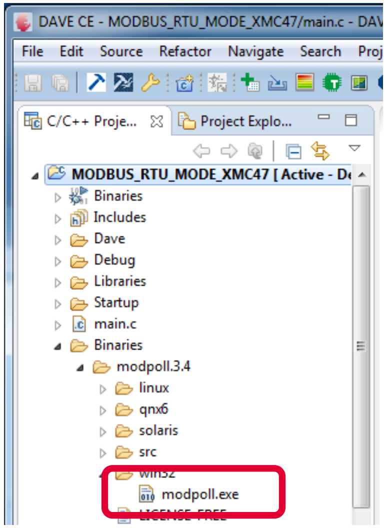
```c
int32_t main(void)
{
  /*Initialize DAVE */
  if(DAVE_Init() == DAVE_STATUS_FAILURE)
  {
    /* Placeholder for error handler code.*/
    XMC_DEBUG(("DAVE APPs initialization failed\n"));
    while(1U)
    {
        /* do nothing */
    }
  }

  /* INITIALIZE BUTTON1 ON PORT 5.13 FOR INPUT */
  /* Set mode to input tristate */
  XMC_GPIO_SetMode(P15_13, XMC_GPIO_MODE_INPUT_TRISTATE);
  /* Enable digital input. Only needed because P15.13 is a
  XMC_GPIO_EnableDigitalInput(P15_13);
  /* INITIALIZE BUTTON2 ON PORT 5.12 FOR INPUT */
  /* Set mode to input tristate */
  XMC_GPIO_SetMode(P15_12, XMC_GPIO_MODE_INPUT_TRISTATE);
  /* Enable digital input. Only needed because P15.12 is a
  XMC_GPIO_EnableDigitalInput(P15_12);

  /* Set FIFO trigger limits */
  UART_SetRXFIFOTriggerLimit (&UART_0, (uint32_t)0);
  UART_SetTXFIFOTriggerLimit (&UART_0, (uint32_t)1);

  /* Register UART_0 interface for modbus usage */
  MB_register_UART(&UART_0);

  /* Initialization of modbus in RTU mode */
  (void)eMBInit( MB_ASCII,                /*eMode (MB_ASCII or
           (uint8_t)0x0A,     /*ucSlaveAddress*/
           (uint8_t)0,        /*ignored*/
           (uint8_t)19200,    /*ulBaudRate*/
           (eMBParity)0         /*ignored*/
         );
  /*Enable modbus protocol stack.*/
  (void)eMBEnable();
```

How to test – Read 16 bit input registers in ASCII mode

ACTIONS (continued)

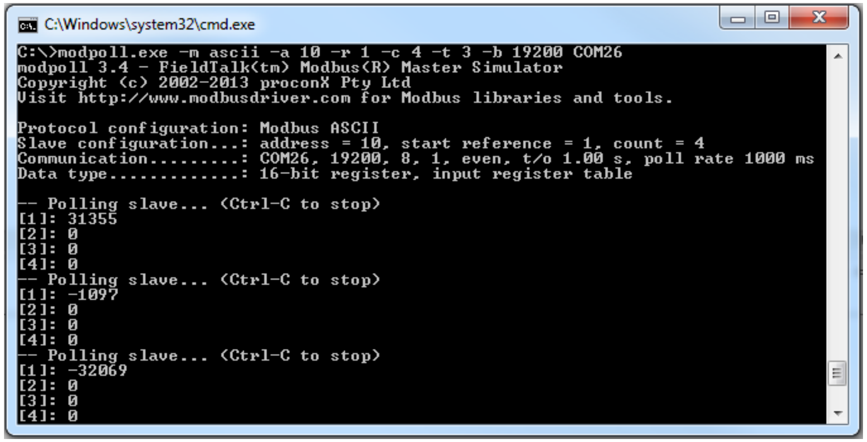
3. Use „Modpoll Modbus Master Simulator" which is provided inside example project to poll input registers 1 to 4:

```
modpoll.exe -m ascii -a 10 -r 1 -c 4 -t 3 -b 19200 COM26
```



OBSERVATIONS

Input registers 1 to 4 are polled every second by master.
The value of register 1 is random non-static, because it is incremented inside the endless loop of the example application



```
C:\>modpoll.exe -m ascii -a 10 -r 1 -c 4 -t 3 -b 19200 COM26
modpoll 3.4 - FieldTalk(tm) Modbus(R) Master Simulator
Copyright (c) 2002-2013 proconX Pty Ltd
Visit http://www.modbusdriver.com for Modbus libraries and tools.

Protocol configuration: Modbus ASCII
Slave configuration...: address = 10, start reference = 1, count = 4
Communication.........: COM26, 19200, 8, 1, even, t/o 1.00 s, poll rate 1000 ms
Data type.............: 16-bit register, input register table

-- Polling slave... (Ctrl-C to stop)
[1]: 31355
[2]: 0
[3]: 0
[4]: 0
-- Polling slave... (Ctrl-C to stop)
[1]: -1097
[2]: 0
[3]: 0
[4]: 0
-- Polling slave... (Ctrl-C to stop)
[1]: -32069
[2]: 0
[3]: 0
[4]: 0
```

Part of your life. Part of tomorrow.