ABB	**Modbus Subsystem Software Design Description** 4-wire top works Modbus Slave			**SDD**	
Responsibility:	Date: **2016-07-15**	Language: **en**	Filing system :	Revision: **1.3**	Page: **1/21**
Issued by: ZuoChen Wang	Approved:		Released:	Area of validity: ABB.BUI	

Title	**4-wire top works modbus slave: Modbus Subsystem Software Design Description**		
Distribution	Software Archive		
Author	Kunli.Zhou	**Date**	2012-03-29
	Zuochen Wang	**Date**	2016-07-15
Review	Greg Leach	**Date**	2012-04-15
	Andrea Stelter	**Date**	2013-01-22
	Jax Yang	**Date**	2013-03-19
	Georg Horst	**Date**	2016-06-15
Approved	Software Architect	**Date**	
	Project Leader	**Date**	
Remarks	2012-04-15 Reviewed by Greg Leach 2013-01-22 Reviewed by Andrea Stelter, modified by Spring Zhou 2013-03-19 Reviewed by Jax Yang, modified by Spring Zhou		

Contents

1 Context

This Document is a representation of analysis, planning, implementation and decision-making. It consists of design information for the Modbus Slave Subsystem.

The document is divided into two main parts. Firstly there is the subsystem Data Sheet. This chapter lists important issues from a black-box view. Secondly there is a more detailed design description intended for developers who have to maintain or expand the subsystem.

Besides this document there are other documents that describe the requirements [1], test cases and the complete system the subsystem is used in. Links to those documents can be found at the end of the data sheet and in the references section.

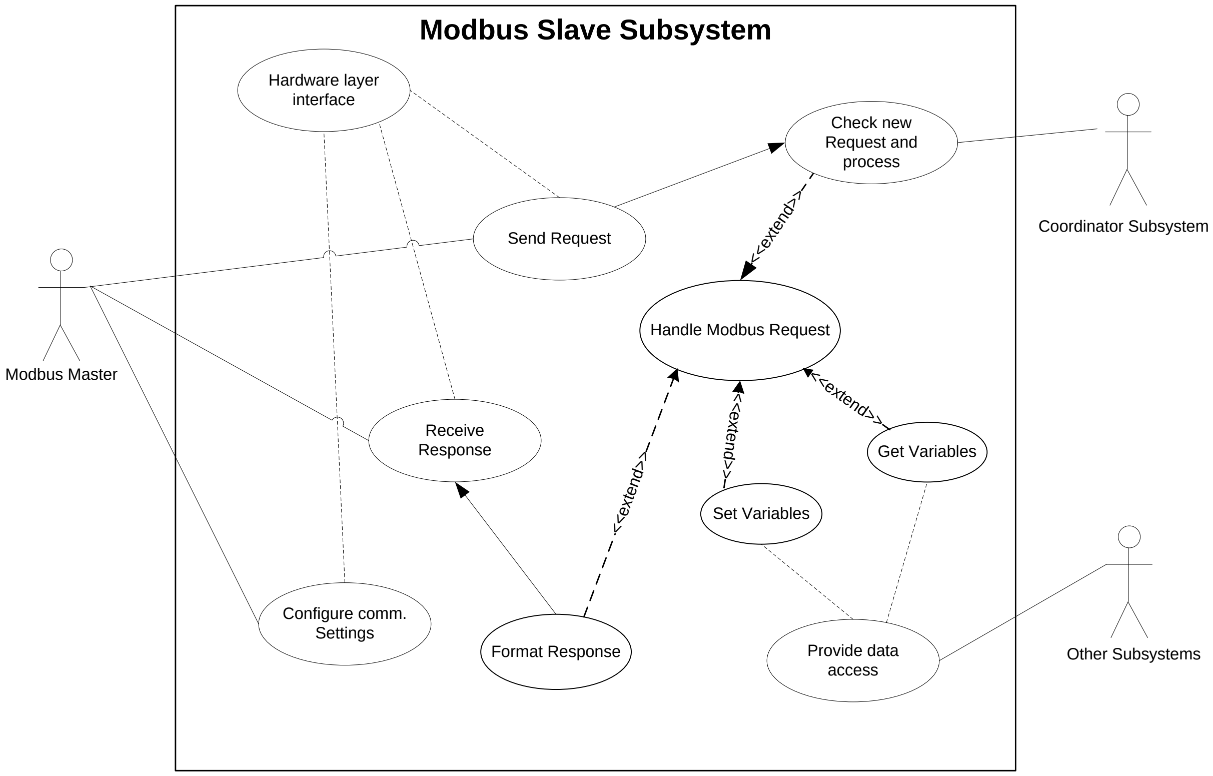


Figure 1: Use Case diagram for Modbus Subsystem

1.1 Definitions, acronyms, and abbreviations

Term	Definition
CRC	Cyclical Redundancy Checking
LRC	Longitudinal Redundancy Checking
PDU	Protocol Data Unit. The MODBUS application protocol defines a simple (PDU) independent of the underlying communication layers, including the Function code and data area.

Term	Definition
CRC	Cyclical Redundancy Checking
LRC	Longitudinal Redundancy Checking
PDU	Protocol Data Unit. The MODBUS application protocol defines a simple (PDU) independent of the underlying communication layers, including the Function code and data area.
RS485	Standard for two wire half duplex serial binary data signal communication between two or more devices over balanced line twisted pair wire bus.
ARM Subsystem	A subsystem which will manage the access rights for different customer level
4WCTW	4 wire common top works
MB	Mother board
FBB	Daughter board

2 Data Sheet

Category	*Item*	*Description*
Development	Version / Status	
	Known Bugs	
	Planned Improvements	
HW-Platform	Type	Renesas Rx210
	Clocking	18.863MHz
SW-Development Environment	Compiler	IAR RX210 IAR C/CE Compiler Version V2.42.3 and IAR ELF Linker V2.42.2
	Operating System	EMBOS for RX210 Version 3.86e
	Case / Code-Generation Tool	Entry Tool 1.1.3 Modbus Gen Tool 1.0.9
Required Resources	Operating System	Subsystem signal semaphore
	HW	RS485 UART, asynchronous half duplex UART12, TXInterrupt, RXInterrupt TMR2, Overflow Interrupt
	RAM	
	NVRAM	Communication configurations parameter
	ROM	Register table and command definition
	Execution Time	<125ms
	Special HW	None
	Subsystems	All subsystem objects need to be configured in modbus register table
	Data Objects	
Standards	Safety	
	Other	- MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3 - MODBUS over Serial Line Specification and Implementation Guide V1.02

Documentation	Requirements / Use Cases	See below Use Case diagram 4WCTW_Modbus Slave Subsystem Requirement Specification.doc
	Public Interface Description	UpdateEXE_MODBUS()
	Test Specification	See test plan

3 Detailed Description

3.1 Static Modelling

3.1.1 Subsystem context

The context of the Modbus Slave Subsystem is shown below in Figure 2 .

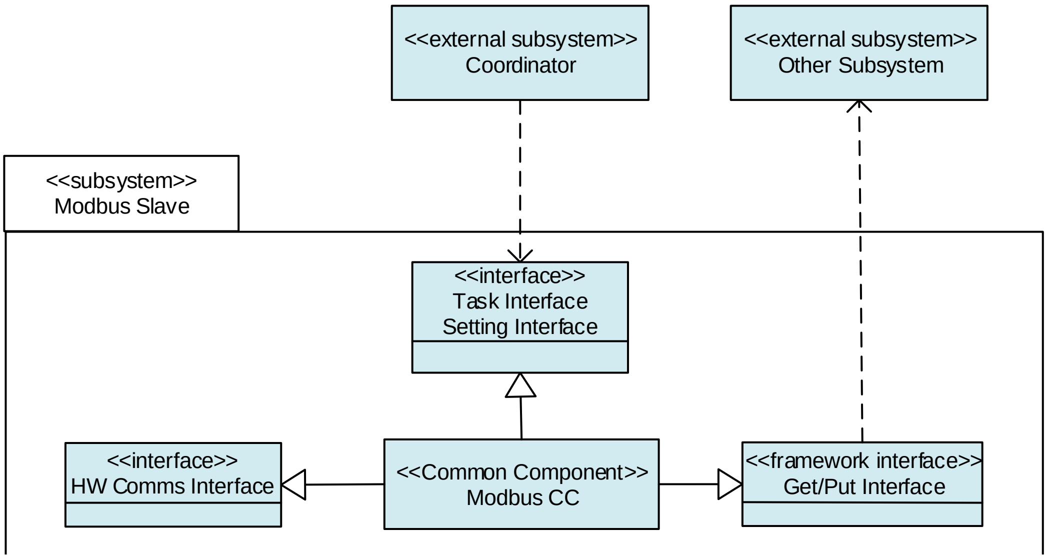


Figure 2 : Context diagram for Modbus Slave Subsystem

The entities are defined as follows:

<<external subsystem>> Coordinator
Plays as a coordinator in this project. It will call the task interface to trigger Modbus Slave subsystem processing the data.

<<external subsystem>> Other subsystem
Other subsystem having the objects that need to put and get through Modbus Slave subsystem.

<<interface>>HW Comms Interface
The communication will use UART 485 as hardware layer. Or USE I2C to communication with Optional MODBUS Module.

<<interface>> Task Interface
Provides an interface UpdateEXE_MODBUS (void) for Task Management to access Modbus Slave subsystem data process.

<<interface>>Setting Interface
Provides an interface UpdateModbusUartSettingEXE_MODBUS (void) for Task Management/Coordinator to Update the Modbus Communication Configuration.
Provides an interface UpdateScanRegister_MODBUS (void) for Task Management/Coordinator to Update the Modbus Scan Registers Table.

<<framework interface>> Get/Put Interface
This interface is the common interface of framework, it realize the relationship between the Modbus Slave subsystem to other subsystem objects set and get.

3.1.2 Subsystem structure

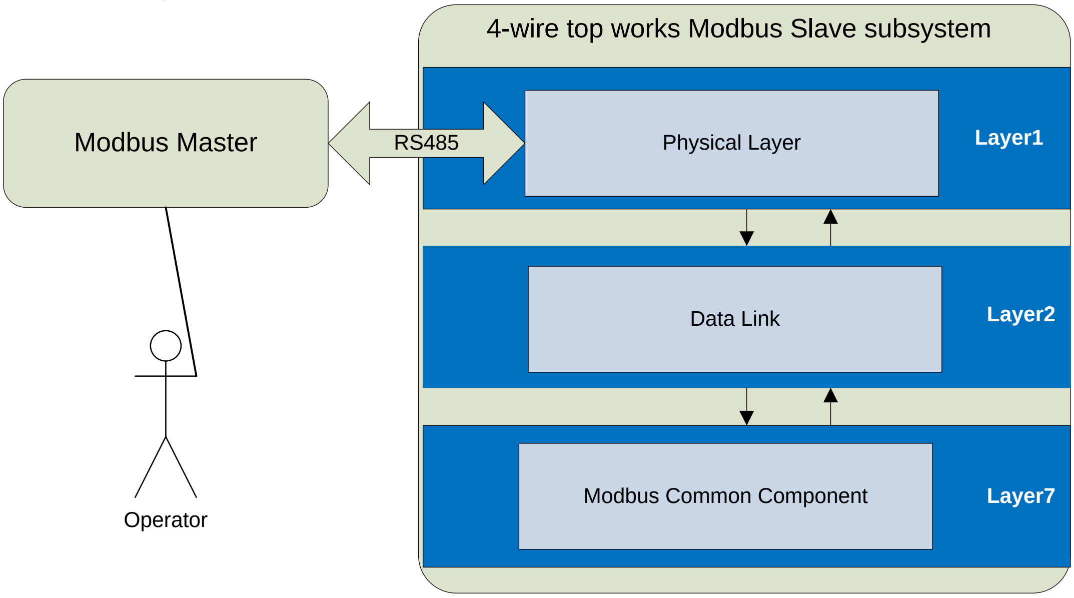


Figure 3 : Modbus Slave Subsystem structure Diagram

3.1.3 Class definition

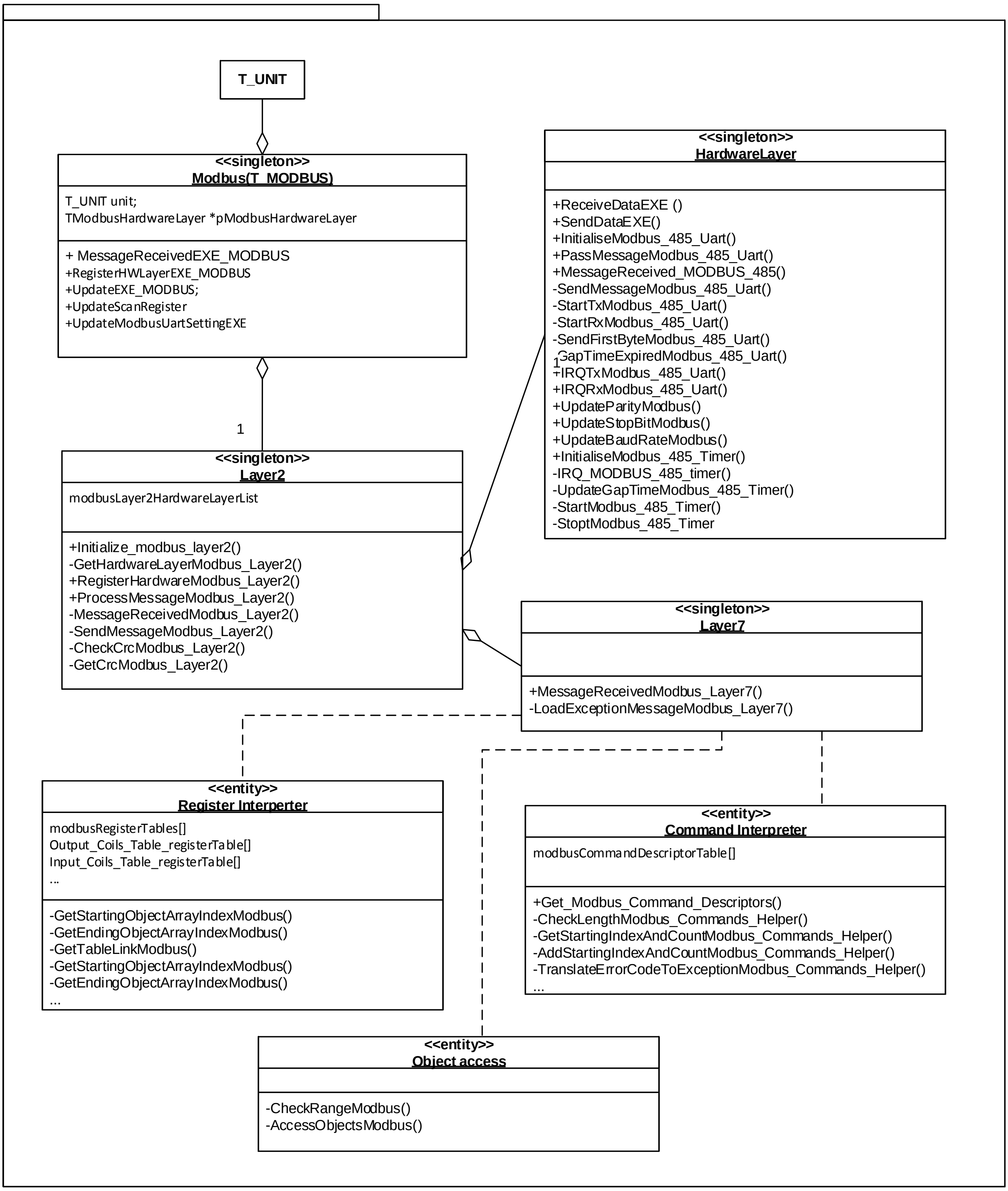

T_UNIT
<<singleton>>
Modbus(T_MODBUS)
T_UNIT unit;
TModbusHardwareLayer *pModbusHardwareLayer
+ MessageReceivedEXE_MODBUS
+RegisterHWLayerEXE_MODBUS
+UpdateEXE_MODBUS;
+UpdateScanRegister
+UpdateModbusUartSettingEXE
1
<<singleton>>
Layer2
modbusLayer2HardwareLayerList
+Initialize_modbus_layer2()
-GetHardwareLayerModbus_Layer2()
+RegisterHardwareModbus_Layer2()
+ProcessMessageModbus_Layer2()
-MessageReceivedModbus_Layer2()
-SendMessageModbus_Layer2()
-CheckCrcModbus_Layer2()
-GetCrcModbus_Layer2()
<<singleton>>
HardwareLayer
+ReceiveDataEXE ()
+SendDataEXE()
+InitialiseModbus_485_Uart()
+PassMessageModbus_485_Uart()
+MessageReceived_MODBUS_485()
-SendMessageModbus_485_Uart()
-StartTxModbus_485_Uart()
-StartRxModbus_485_Uart()
-SendFirstByteModbus_485_Uart()
-GapTimeExpiredModbus_485_Uart()
1
+IRQTxModbus_485_Uart()
+IRQRxModbus_485_Uart()
+UpdateParityModbus()
+UpdateStopBitModbus()
+UpdateBaudRateModbus()
+InitialiseModbus_485_Timer()
-IRQ_MODBUS_485_timer()
-UpdateGapTimeModbus_485_Timer()
-StartModbus_485_Timer()
-StoptModbus_485_Timer
<<singleton>>
Layer7
+MessageReceivedModbus_Layer7()
-LoadExceptionMessageModbus_Layer7()
<<entity>>
Register Interperter
modbusRegisterTables[]
Output_Coils_Table_registerTable[]
Input_Coils_Table_registerTable[]
...
-GetStartingObjectArrayIndexModbus()
-GetEndingObjectArrayIndexModbus()
-GetTableLinkModbus()
-GetStartingObjectArrayIndexModbus()
-GetEndingObjectArrayIndexModbus()
...
<<entity>>
Command Interpreter
modbusCommandDescriptorTable[]
+Get_Modbus_Command_Descriptors()
-CheckLengthModbus_Commands_Helper()
-GetStartingIndexAndCountModbus_Commands_Helper()
-AddStartingIndexAndCountModbus_Commands_Helper()
-TranslateErrorCodeToExceptionModbus_Commands_Helper()
...
<<entity>>
Object access
-CheckRangeModbus()
-AccessObjectsModbus()

Figure 4: Modbus Slave Subsystem class Diagram

3.2 Dynamic Modelling

The following three figures show the state chart and activity chart of modbus slave subsystem transmission and data process.

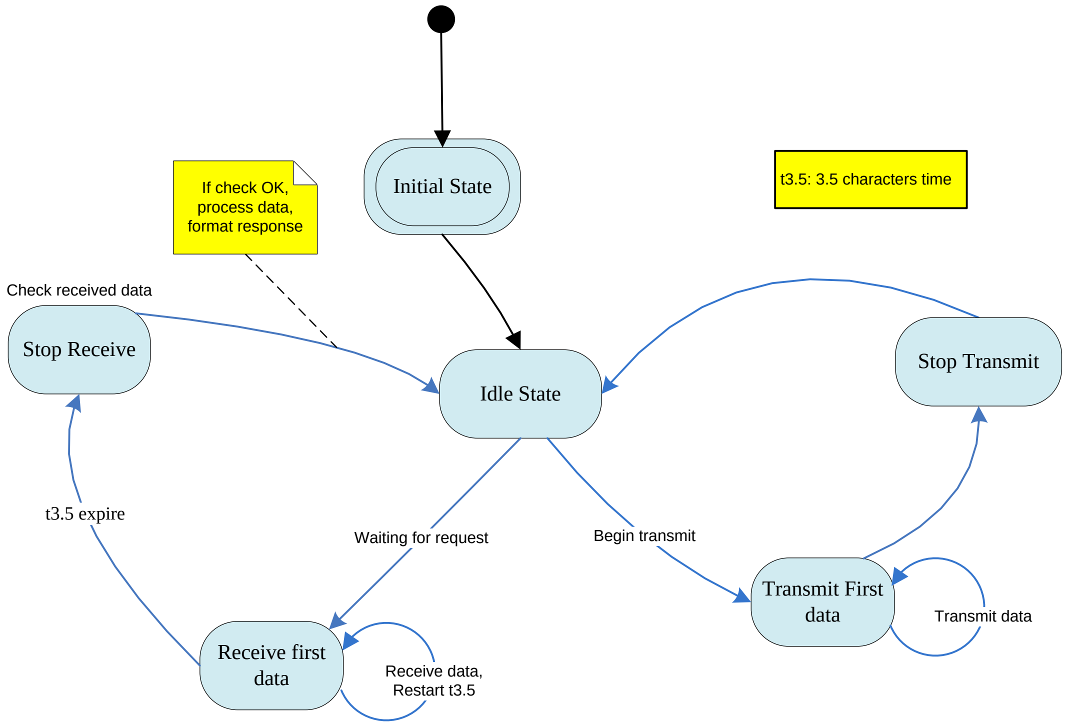


Figure 5: State chart of RTU transmission and timers

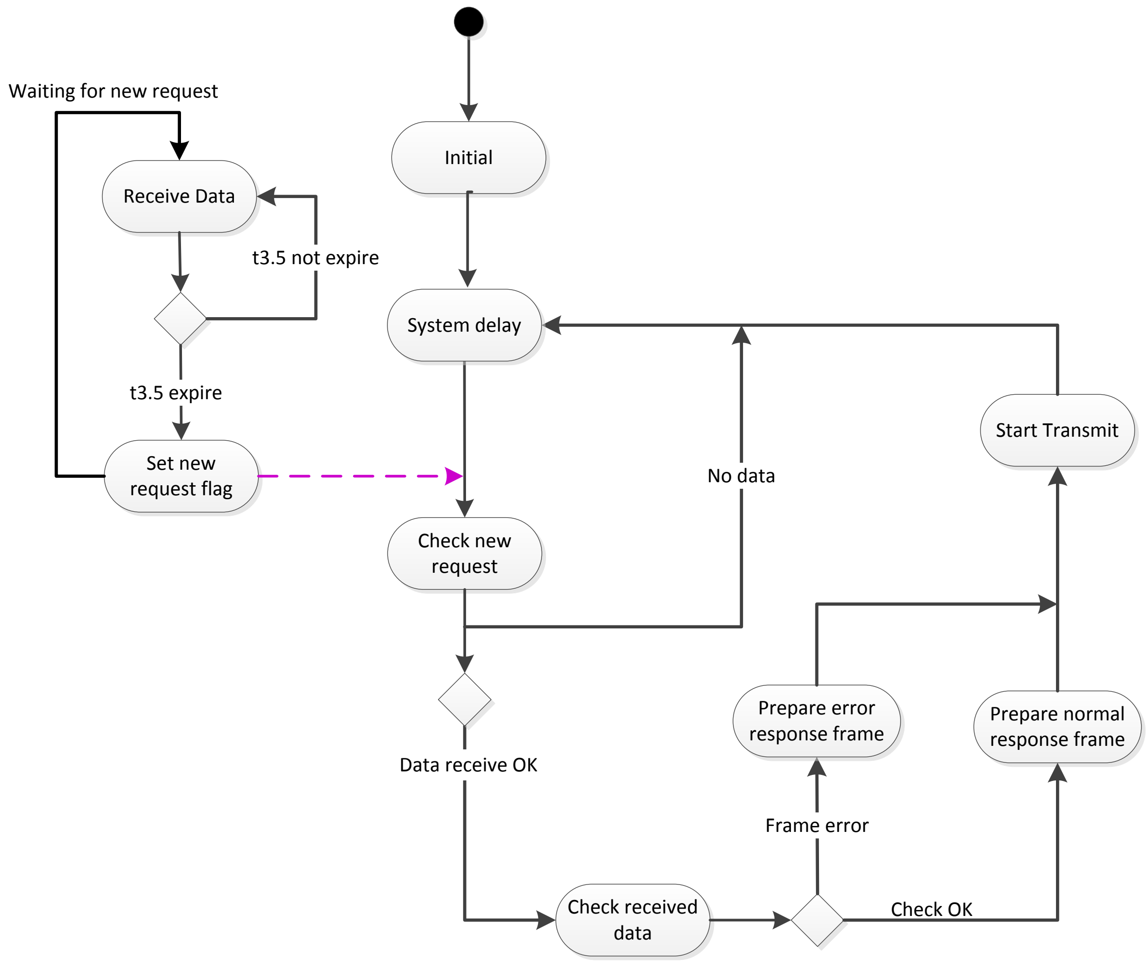


Figure 6: activity diagram of RTU transmission

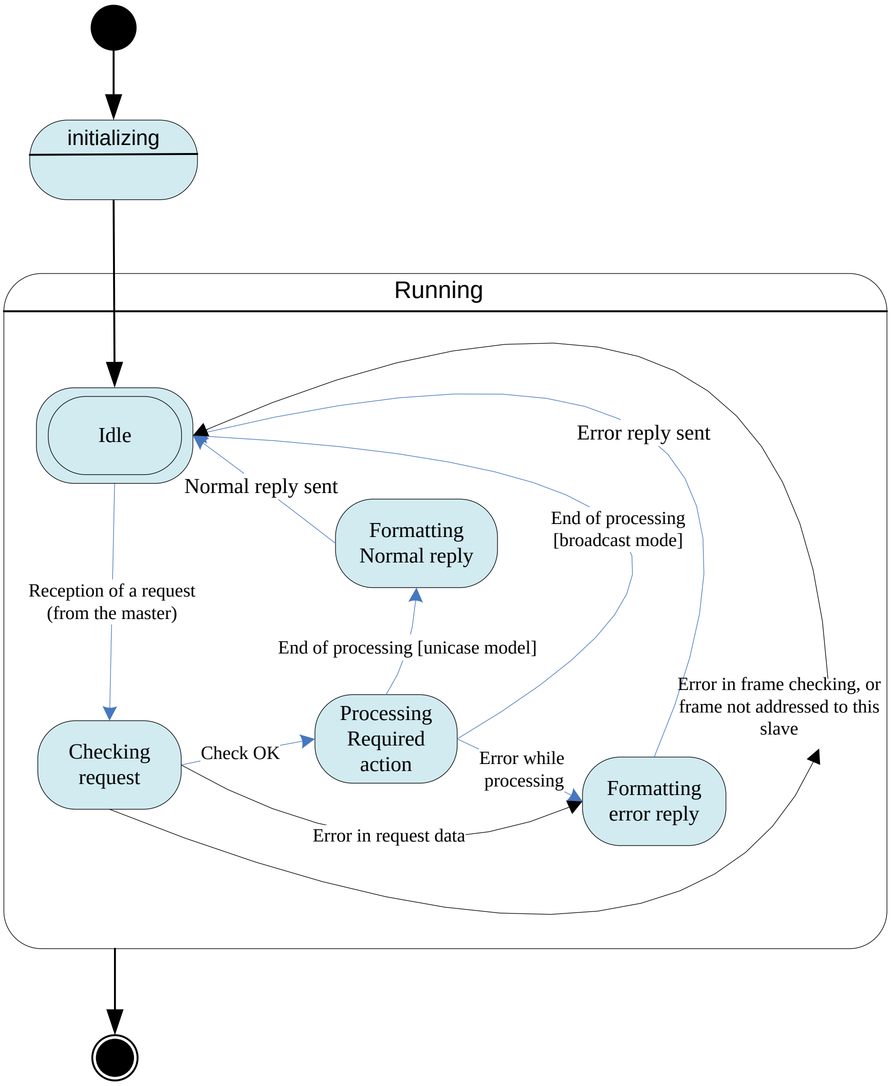
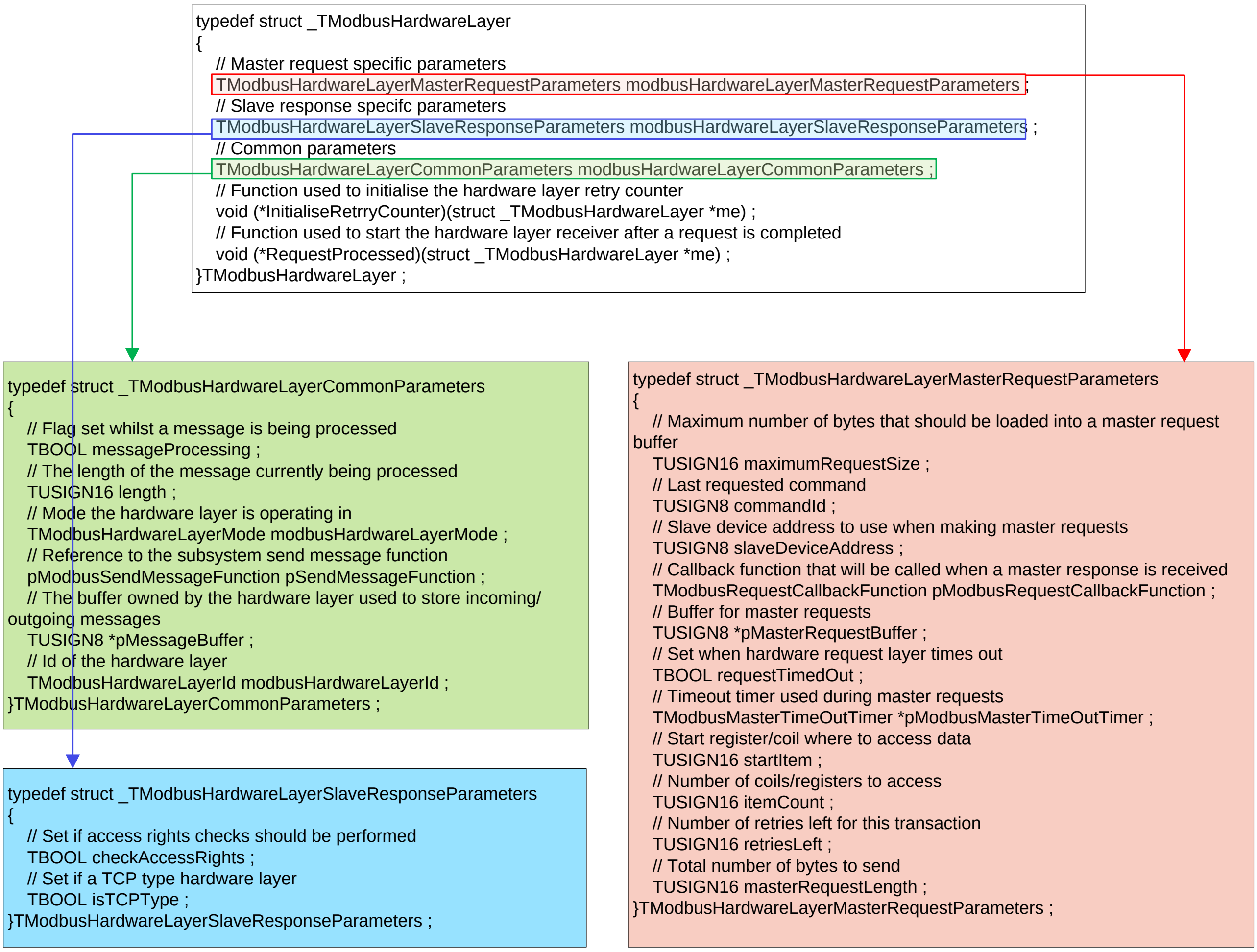

initializing
Running
Idle
Normal reply sent
Error reply sent
Formatting Normal reply
End of processing [broadcast mode]
Reception of a request (from the master)
End of processing [unicase model]
Error in frame checking, or frame not addressed to this slave
Processing Required action
Checking request
Check OK
Error while processing
Formatting error reply
Error in request data

Figure 7: State chart of Data Process

3.3 Class Design

3.3.1 Hardwarelayer structure

The subsystem put all the related hardware parameters and buffers to one whole structure TModbusHardwareLayer.



```
typedef struct _TModbusHardwareLayer
{
   // Master request specific parameters
   TModbusHardwareLayerMasterRequestParameters modbusHardwareLayerMasterRequestParameters ;
   // Slave response specifc parameters
   TModbusHardwareLayerSlaveResponseParameters modbusHardwareLayerSlaveResponseParameters ;
   // Common parameters
   TModbusHardwareLayerCommonParameters modbusHardwareLayerCommonParameters ;
   // Function used to initialise the hardware layer retry counter
   void (*InitialiseRetrryCounter)(struct _TModbusHardwareLayer *me) ;
   // Function used to start the hardware layer receiver after a request is completed
   void (*RequestProcessed)(struct _TModbusHardwareLayer *me) ;
}TModbusHardwareLayer ;
```

```
typedef struct _TModbusHardwareLayerCommonParameters
{
   // Flag set whilst a message is being processed
   TBOOL messageProcessing ;
   // The length of the message currently being processed
   TUSIGN16 length ;
   // Mode the hardware layer is operating in
   TModbusHardwareLayerMode modbusHardwareLayerMode ;
   // Reference to the subsystem send message function
   pModbusSendMessageFunction pSendMessageFunction ;
   // The buffer owned by the hardware layer used to store incoming/
outgoing messages
   TUSIGN8 *pMessageBuffer ;
   // Id of the hardware layer
   TModbusHardwareLayerId modbusHardwareLayerId ;
}TModbusHardwareLayerCommonParameters ;
```

```
typedef struct _TModbusHardwareLayerSlaveResponseParameters
{
   // Set if access rights checks should be performed
   TBOOL checkAccessRights ;
   // Set if a TCP type hardware layer
   TBOOL isTCPType ;
}TModbusHardwareLayerSlaveResponseParameters ;
```

```
typedef struct _TModbusHardwareLayerMasterRequestParameters
{
   // Maximum number of bytes that should be loaded into a master request
buffer
   TUSIGN16 maximumRequestSize ;
   // Last requested command
   TUSIGN8 commandId ;
   // Slave device address to use when making master requests
   TUSIGN8 slaveDeviceAddress ;
   // Callback function that will be called when a master response is received
   TModbusRequestCallbackFunction pModbusRequestCallbackFunction ;
   // Buffer for master requests
   TUSIGN8 *pMasterRequestBuffer ;
   // Set when hardware request layer times out
   TBOOL requestTimedOut ;
   // Timeout timer used during master requests
   TModbusMasterTimeOutTimer *pModbusMasterTimeOutTimer ;
   // Start register/coil where to access data
   TUSIGN16 startItem ;
   // Number of coils/registers to access
   TUSIGN16 itemCount ;
   // Number of retries left for this transaction
   TUSIGN16 retriesLeft ;
   // Total number of bytes to send
   TUSIGN16 masterRequestLength ;
}TModbusHardwareLayerMasterRequestParameters ;
```

_TModbusHardwareLayer
This structure includes all the hardware layer related issues, including following three structure.

_TModbusHardwareLayerMasterRequestParameters
This structure lists all the parameters of the master request.

_TmodbusHardwareLayerSlaveResponseParameters
This structure lists the parameters of the slave response

_TmodbusHardwareLayerCommonParameters
This structure lists all the parameters related to communications. Including the SendResponseFunction pointer and the receive/transmit buffer pointer.

3.4 Detailed Design

3.4.1 Initialize

3.4.1.1 hardware layer related parameter

Define the modbusLayer2HardwareLayerList in layer2. For 4-wire top works Modbus Slave subsystem, only one uart 485 is implemented.

Define the TModbusHardwareLayer modbus485HardwareLayer sturcture initial value.

3.4.1.2 Initialize hardware layer

The Modbus Hardwarelayer should be initialized including the UART and timer initialization. The elements should be initialized are listed here: transmission mode, slave address, baud-rate, parity and stop-bit. Gap time.
Register the hardware layer by using the method RegisterHWLayerEXE( ), ensure the hardware layer is not used by other port (this is not compulsory for only one hardware layer will be inplemented).

3.4.2 Update Modbus

In the main task, the modbus subsystem will check if the hardware layer have received request data, if there is a complete frame received, then use the ProcessMessageModbus_Layer2( ) method to process the request.
After processing data and format response, use SendMessageModbus_Layer2 ( ) method to send the response frame.

The activity diagram takes reference to Fig 5.

3.4.3 Update Modbus Communication Configuration

When the configuration of the modbus is changed, the modbus slave subsystem will update the settings using the following functions.

- UpdateModbusUartSettingEXE ()

This function will change the Modbus UART Setting. It includes:

a) Address->Default Setting Value is one.
b) BaudRate->Default Setting Value is 9600 bps.
c) Parity->Default Setting Value is ODD
d) StopBits->Default Setting Value is one stop bit.
e) ResponseDelayTime->Default Setting Value is 10 ms.

- UpdateScanRegister ()

This function is going to update the snap view of registers. The snap view is a collection to collect the values in in different registers address.
At the same time, update the Gap time settings according to these changes.

3.4.4 Request process

The request activity is processed by common component subsystem, see diagram below.

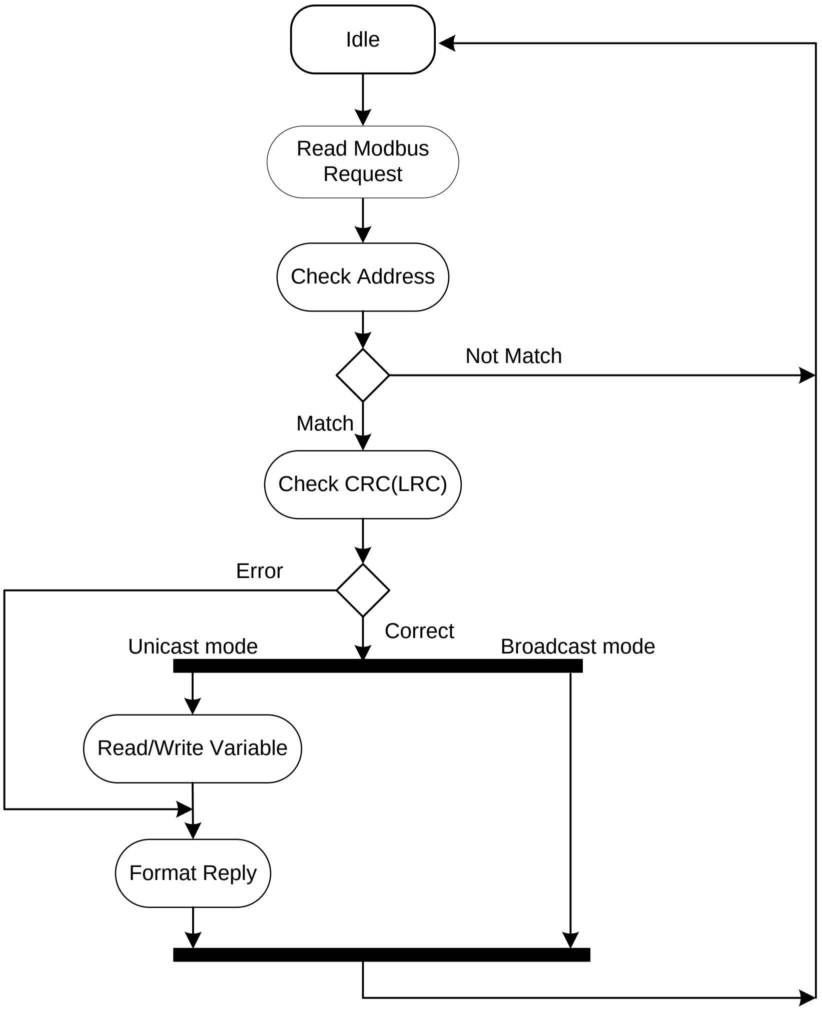


Figure 8: Flow chart for request process

3.4.5 Data Handler of Modbus Common Component

The figure below gives a brief introduction about the data process activity, detailed information please see Common component subsystem design description.

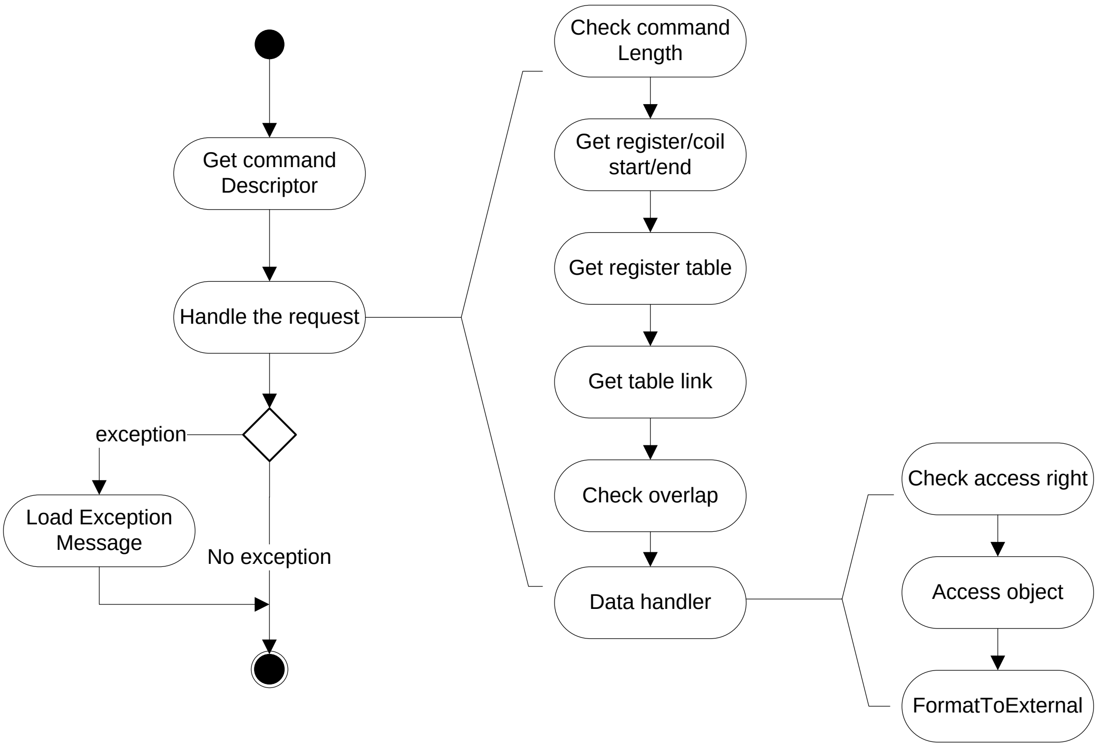


Figure 9: Activity diagram for request process

3.4.6 Modbus Timers

3.4.6.1 Inter frame timer

If the baud rate is higher or equal to 38400bps, then the inter frame timer should fix at 1.75ms.
The interframe time will be updated when the communication configuration variables changes.
If the inter frame timer overflows when receiving data, it will consider that this frame is over and pass the data to layer2.

3.4.6.2 Inter character timer

Do not implement this timer in the subsystem, it is ensured by the modbus master.

3.4.7 Modbus PDU and Register definition

Modbus Serial Line PDU is the data object passed between Remote Comms subsystem and Modbus Stack Interface. In reality, a pointer is used to point to the head of this PDU buffer, when we pass it, we just change the pointer. It’s a common useful method of data transfer. The same method is used in Modbus PDU which is transferred between Modbus Stack Interface and Modbus Slave Stack.

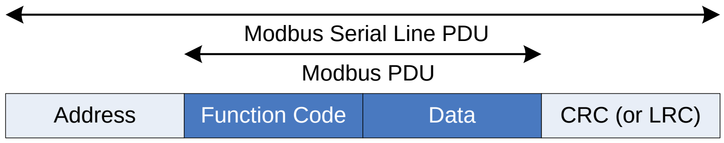


Figure 10: Modbus frame format

(1) Modbus Addressing rules
The Modbus addressing space comprises 256 different addresses. Number 1~247 can be used for this Modbus Slave. NV Default shall be 247.
The Address 0 is reserved as the broadcast address. All slave nodes must recognise the broadcast address. The slave nodes must have an address. This address must be unique on a MODBUS serial bus.

(2) The standard command supported are shown in table below.

Command ID (Hex)	Description
0x1	Read coils
0x2	Read discrete inputs
0x3	Read holding registers
0x4	Read input registers
0x5	Write single coil
0x6	Write single register
0x8	Diagnostics
0xF	Write multiple coils
0x10	Write multiple registers
0x11	Report slave id

Table 1: Standard Modbus commands

The sub-command ids shown in Table should be supported by the Diagnostics command (0x8).

Sub-command ID (Hex)	Description
0x0	Return query data

Table 2: Supported Diagnostics command sub-ids

(3) Data field
Commonly, the Data field includes the register addresses and the number of registers or the preset values. Table 4 is the index table that we can use to map the Modbus registers to the variables. As described in the requirements document, the uppermost level register structure is provided shown in Table 4 below.

Custom Command Address	Device Address	Description
0 .. 09999	1 .. 10000	Coils (outputs)
10000 .. 19999	10001..20000	Input coils
30000 .. 39999	30001.. 40000	Input registers
40000 .. 49999	40001.. 50000	Holding registers

Table 3: Modbus Subsystem register table structure

(4) CRC Check
CRC is used for RTU mode
CRC calculation is implemented in two use-cases: “checking request” after the reception of the request and “formatting reply” after the end of processing.

For CRC, we use the Index table method, the generating polynominal = 1 + x2 + x15 + x16. The two 256B tables reside in ROM.

3.4.8 Communication to Option Card Modbus

4WCTW MB communicate with FBB via I2C interface, the 4WCTW MB is I2C master, Modbus module is I2C slave.

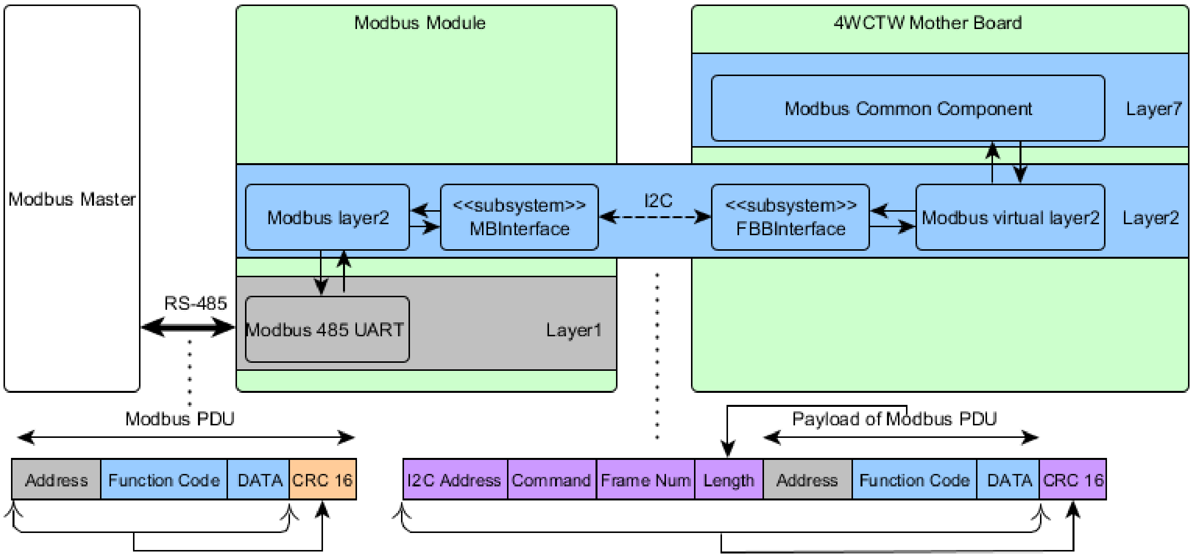


Figure 11: MODBUDS Deployment Diagram

An event-driven task FBBITask to drive the communication between the MODUBS module and 4WCTW mother board Modbus. Task shall drive/call CommunicationCtrlModbus of FBBInterfaceto interact with MODBUS.

3.5 Design Decisions and Limitations

3.6 Hardware Dependencies

This Modbus Slave Subsystem is developed on the Renesas Rx210 microcontroller, use UART12 as it communication hardware and user timer MTU5 to manage the inter frame time.
Power control of this hardware is handled outside the scope of the Modbus Subsystem.
If it work together with FBBInterface. Then the RIIC is used to communicate with optional MODBUS module.

3.7 Data Object Description

3.8 Data Object Type Description

3.8.1 t_data_obj_modbus_diag.c

The data class comply with ABB Framework. It inherit from t_data_obj class. The follow two functions are overload by t_data_obj_modbus_diag.

3.8.1.1 Put_MODBUS_DIAG

```
TUSIGN16 Put_MODBUS_DIAG(const T_DATA_OBJ SLOW*me,
                    void FAST* ptrValue,
                    TINT16 attributeIndex,
                    OS_RSEMA FAST* ptrDataSemaphore)
```

This method blocks any put request, as the diagnostic bytes are read only, write request is not acceptable. Any calling of this function will received a READ_ONLY_ERR result.

Parameters:

ptrValue ->the new object or attribute value;
attributeIndex-> Set to -1 to operate on the whole object. Set the value above -1 will locate a specify attribute.
ptrDataSemaphore-> Pointer to the resource-semaphore of the used data-memory

3.8.1.2 Get_MODBUS_DIAG

```
TUSIGN16 Get_MODBUS_DIAG(const T_DATA_OBJ SLOW*me,
                    void FAST* ptrValue,
                    TINT16 attributeIndex,
                    OS_RSEMA FAST* ptrDataSemaphore)
```

This method extracts the active diagnostic alarm information from diagnostic sub-system, and stored the value in ptrValue to pass it to the external interface.

Parameters:

ptrValue -> the new object or attribute value will be returned in ptrValue
attributeIndex-> Set to -1 to operate on the whole object. Set the value above -1 will locate a specify attribute.
ptrDataSemaphore-> Pointer to the resource-semaphore of the used data-memory

The return value will be OK or ILLEGAL_ATTRIB_IDX according the operate result.

3.8.2 t_data_obj_modbus_diag_history.c

The data class comply with ABB Framework. It inherit from t_data_obj class. The follow two functions are overload by t_data_obj_modbus_diag.

3.8.2.1 Put_MODBUS_DIAG_HIS

```
TUSIGN16 Put_MODBUS_DIAG_HIS(const T_DATA_OBJ SLOW*me,
                    void FAST* ptrValue,
                    TINT16 attributeIndex,
                    OS_RSEMA FAST* ptrDataSemaphore)
```

This method blocks any put request, as the diagnostic history bytes are read only, write request is not acceptable.Any calling of this function will received a READ_ONLY_ERR result.

Parameters:

ptrValue ->the new object or attribute value;
attributeIndex-> Set to -1 to operate on the whole object. Set the value above -1 will locate a specify attribute.
ptrDataSemaphore-> Pointer to the resource-semaphore of the used data-memory

3.8.2.2 Get_MODBUS_DIAG_HIS

```
TUSIGN16 Get_MODBUS_DIAG_HIS(const T_DATA_OBJ SLOW*me,
                void FAST* ptrValue,
                TINT16 attributeIndex,
                OS_RSEMA FAST* ptrDataSemaphore)
```

This method extracts the alarm history information from diagnostic sub-system, and return these values to the caller.

Parameters:

ptrValue -> the new object or attribute value will be returned in ptrValue
attributeIndex-> Set to -1 to operate on the whole object. Set the value above -1 will locate a specify attribute.
ptrDataSemaphore-> Pointer to the resource-semaphore of the used data-memory

The return value will be OK or ILLEGAL_ATTRIB_IDX according the operate result.

3.9 References

Ref.	Document
[1]	4-wire top works Modbus Subsystem Requirements Specification
[2]	Modbus common component design description
[3]	4WCTW Comport design description
[4]	Modbus_Application_Protocol_V1_1b.pdf
[5]	Modbus Over Serial Line V1.02
[6]	V0.9_4WCTW Mother Board and Field Bus Board Protocol.docx
[7]	4WCTW Fieldbus Module Firmware Integration Guide.doc
[8]	V0.2_4WCTW Modbus Module System Design Description.docx

4 Revision History

Rev.	Description of Version/Changes	Primary Author(s)	Date
0.1	Initial revision	Spring Kunli.Zhou	2012-02-10
0.2	Modification to design according to requirements and guidance under Greg Leach	Spring Kunli.Zhou	2012-04-15
1.0	Do not use Comport subsystem as the communication interface, the subsystem will use the hardware UART and timer directly.	Spring Kunli.Zhou	2013-01-17
1.1	Modified after review by Andrea Stelter	Spring Kunli.Zhou	2013-01-23
1.2	Modified after review by Jax Yang	Spring Kunli.Zhou	2013-03-19
1.3	Modified according to the reviewed comments under Georg Horst	ZuoChen Wang	2016-07-15

5 Design Review

5.1 Decision of the Review:

Decision	*next steps*
❑ Inspection passed ***without restrictions***	Phase finished
■ Inspection passed ***with restrictions***	some changes must be done
❑ Inspection ***not*** passed	Inspection must be repeated

5.2 Check list:

		yes	no
1.	Is the software architecture distinct and documented?	**Y**	
2.	Fit the modules together?	**Y**	
3.	Are complex algorithms/procedures explained?	**Y**	
4.	Is a strategy for error handling designated?	**Y**	
5.	Is the configuration management system well prepared?	**Y**	
6.	Are all open issues transferred to the defects table?	**Y**	

5.3 Remarks:

5.4 Defects

No.	Checkpoint	Description	Major defect	done Date
1	3.2	Do not need to implement the inter-character timer. Change the dynamic diagram	N	2012-4-15
2	3.2,3.4.4.1	Only need to use Read () function to get the communication data, do not need to process the timeover which will be implemented in the comport subsystem. Change the description and the diagram.	N	2012-4-15
3		Delete all the customized command definition, common command 0x8 only support one subcommand.	N	2012-4-15
4	3.4.6.1	‘If the baud rate is higher than 38400bps’ should change to ‘If the baud rate is higher than or equal to 38400bps’		2013-1-22
5	3.4.8	“two 256B tables reside in EPPROM” should be “ROM”	N	2013-1-22
6	3.2	Figure 6: activity diagram of RTU transmission. “Set new request flag” should be connected to the entry before the “Check new request ” block.	N	2013-3-19
7	3.4.7	The abbreviation “PDU” need to be explained in details.	N	2013-3-19
8	3.4.8	Section “(4)CRC Check” The CRC polynomial should be specified.	N	2013-3-19

9	4	Under Data Sheet (2): - SW-Development Environment to be extended by IAR RX210 IAR C/CE Compiler Version V2.42.3 and IAR ELF Linker V2.42.2 - Operating System extend by embOS for RX210 Version 3.86e	N	2016-06-15
10	4	Scan Registers are not described in the Design Spec.	Y	2016-06-15
11	4	Update Modbus Uart Settings are not described in the Design Spec.	Y	2016-06-15
12	Other	Under Acronyms and Abbreviations (1.3): No 4-wire related definitions.	N	2016-06-15
13	Other	Interface to Option Card Modbus is missing as rough description and as reference.	Y	2016-06-15
14	Other	Detailed description of t_data_obj_modbus_diag & t_data_obj_modbus_diag_history missing	N	2016-06-15

5.5 Changes are proved:

The Reviewer confirms that all changes are done:

Proved Rev:	Updated to Rev:	Date:	Reviewer:
Spring Zhou	0.2	2012-4-15	Greg Leach
Spring Zhou	1.1	2013-1-23	Andrea Stelter
Spring Zhou	1.2	2013-3-19	Jax Yang