Implementing the RS485 Modbus with PSoC62 using ModusToolbox™

Gintaras Drukteinis,
RUTRONIK Electronics Worldwide, Kaunas, Lithuania

***Abstract* — The Modbus protocol was created in 1979 by Schneider Electric and it is still in use today. It is widely used as a standard open protocol in industrial and commercial environments because of its simplicity, low resources requiring and royalty-free features. New technologies and development environments are emerging and evolving constantly but the need to implement already existing de facto or true standard protocols never disappears. The implementation of Modbus protocol with PSoC6245 microcontroller and RS485 transport layer is described in this application note.**

Index Terms* — *Microcontroller (MCU), Integrated Development Environment (IDE), Berkeley Source Distribution (BSD), Electrostatic discharge (ESD), Printed Circuit Board (PCB), Clear-to-Send (CTS), Real-Time Operational System (RTOS).

I. Introduction

The ModusToolbox IDE is a free development environment provided by Infineon. Though the ModusToolbox has a large variety of third-party software libraries integrated, the Modbus protocol stack is not included yet. Since the Modbus is an open royalty-free protocol, the free-to-use libraries are quite popular. For instance, the "FreeMODBUS" provided by "embedded experts" is available under the BSD license conditions and supports RTU/ASCII transmission modes. The FreeMODBUS supports standard Modbus functions written in C language and comes with many examples for the variety of MCU architectures and development platforms. On the other hand, the PSoC62 microcontrollers are not supported currently, hence the example will be provided by Rutronik for the development platform called RutDevKit-PSoC62.

II. Hardware Considerations

The RS485 communication interface requires a driver with data direction control. The purpose of the driver is to convert single-ended reception and transmission signals to differential signals traveling through the single twisted pair and vice versa. The MaxLinear SP3078EEN-L/TR driver is used on the RutDevKit-PSoC62 board. This driver can support up to 256 devices connected on a single twisted pair. The ability to operate at 3.3V makes this driver ideal for most of the modern MCUs. Although the device has 15kV ESD protection for RS485 pins it is recommended to protect the RS485 additionally with external ESD diodes and resettable fuses in case wiring mistakes occur.

On the PSoC62 side, the data pins are connected to the UART RX TX signals with no additional components attached. The PSoC62 supports hardware data direction control using a CTS signal.

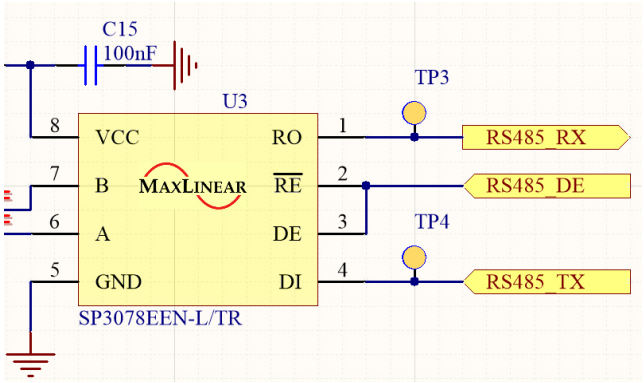

Fig. 1 RS485 driver TX, RX, and DE (CTS) signals

Additional care must be taken with the PCB design while routing the differential traces to the driver and output terminal. The best results would be if the traces are routed according to the differential routing rules while keeping the impedance as close as possible to 120Ω. Moreover, for practical use cases, the traces need to be much wider than impedance match control rules require because the currents are much larger than the fault event occurs. Alternatively, a compromise to meet all the requirements is to have protection devices as close as possible to the output terminal and then smaller traces could continue under the differential and impedance matching rules.

Following the RS485 specification, the last device on the RS485 line must be terminated with a 120Ω resistor. The RutDevKit-PSoC62 has such resistors on every board, only they are not enabled by default. The solder bridge has to be shorted to have the termination enabled. It becomes necessary if a long twisted pair cable is used and the reflected signal from the terminal starts to interfere with communications.

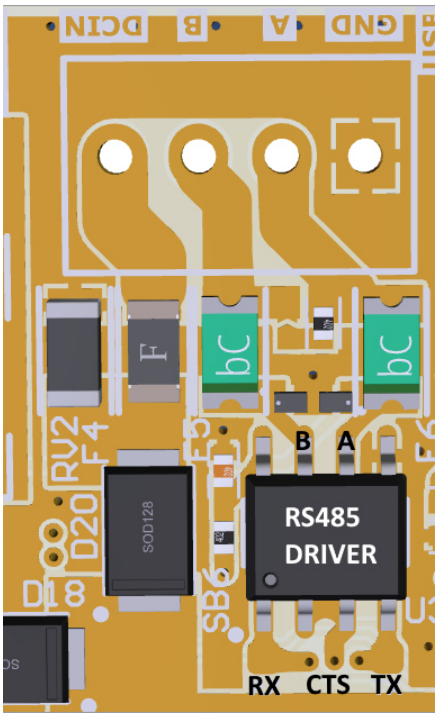

Fig. 2 RS485 routing and placement example.

III. Porting the FreeMODBUS Protocol Stack for PSoC6245

The FreeMODBUS requires a timer and serial port to function properly.

The timer is used for measuring frames duration. It must be running at the frequency of 20kHz - 50µs per tick. The functions that need to be implemented reside in *porttimer.c* file. The stack must be able to initialize, enable, disable the timer. Moreover, it needs a method to detect the timeout of any given period. The most reliable and simple way to do that is to send the signal from the timer timeout interrupt routine.

```
/* Initialize the timer */
BOOL xMBPortTimersInit( USHORT usTim1Timerout50us )
/* Timer Enable */
void vMBPortTimersEnable()
/* Timer Disable */
void vMBPortTimersDisable()
/* Timer Interrupt Routine */
static void mb_timer_isr(void *callback_arg,
cyhal_timer_event_t event)
/* Timeout callback */
(void)pxMBPortCBTimerExpired();
```

The serial UART interface needs to be initialized by the stack as well. The stack requires UART send/receive functions and interrupt control for incoming/outgoing data. The system also must inform the stack when the UART has received a byte and if the transmit buffer is empty. All the functions are ported in the *portserial.c* file.

```
BOOL xMBPortSerialInit(UCHAR ucPORT, ULONG ulBaudRate,
UCHAR ucDataBits, eMBParity eParity);
void vMBPortSerialEnable(BOOL xRxEnable, BOOL xTxEnable);
BOOL xMBPortSerialPutByte(CHAR ucByte);
BOOL xMBPortSerialGetByte(CHAR * pucByte);
static void  uart_event_handler(void *handler_arg,
cyhal_uart_event_t event);
```

If the RTOS is used the additional functions must be implemented in the *portevent.c* file. A task for polling the function `eMBPoll();` in a continuous loop has to be created. The task is supposed to be in a blocked state until the Modbus protocol stack event unblocks it. All events should be placed in a queue. Some event posts are done within the interrupt routine, hence the event post function must be implemented to be able to recognize and use a correct RTOS method to update a queue.

```
BOOL xMBPortEventInit(void);
BOOL xMBPortEventPost(eMBEventType eEvent);
BOOL xMBPortEventGet(eMBEventType * eEvent);
```

IV. Using the FreeMODBUS Protocol Stack

The FreeMODBUS protocol stack does not have any memory allocated for the registers by default, hence the user must assign the registers to the internal or external MCU memory before the protocol stack is enabled. The typical sequence initializing functions is given below:

```
  /*Assign the register to the memory*/
  /* ... */

  /*FreeMODBUS protocol stack initializaton*/
  eMBInit(MB_RTU, 1, 0, 115200, MB_PAR_NONE);
  eMBEnable();

/*Poll the FreeModbus Stack*/
  for(;;)
  {
    eMBPoll();
  }
```

The memory-related with Modbus protocol registers is read or written by the user application or the protocol and sometimes both. It is completely up to the user how the memory reserved by the Modbus protocol is used by the application. There are four callback functions for the Modbus protocol register control:

```
eMBErrorCode eMBRegInputCB(UCHAR * pucRegBuffer, USHORT
usAddress, USHORT usNRegs)

eMBErrorCode eMBRegHoldingCB(UCHAR * pucRegBuffer, USHORT
usAddress, USHORT usNRegs, eMBRegisterMode eMode)

eMBErrorCode eMBRegCoilsCB(UCHAR * pucRegBuffer, USHORT
usAddress, USHORT usNCoils, eMBRegisterMode eMode)

eMBErrorCode eMBRegDiscreteCB(UCHAR * pucRegBuffer,
USHORT usAddress, USHORT usNDiscrete)
```

The callback functions are the interface between the FreeMODBUS protocol stack and user application hardware resources. Although the callback functions are flexible enough to be implemented with various scenarios, the basic

requirements for the data arguments must be followed as it is shown in (Tab. 1).

Tab. 1

Coil/Contact/Register	Data Addresses	Type	Register Name
1-9999	0x0000 - 0x270E	Read/Write	Discrete Output Coils
10001-19999	0x0000 - 0x270E	Read Only	Discrete Input Contacts
30001-39999	0x0000 - 0x270E	Read Only	Input Registers
40001-49999	0x0000 - 0x270E	Read/Write	Holding Registers

V. Test Results

A combination of the RutDevKit-PSoC62, ModusTollbox IDE, FreeMODBUS protocol, FreeRTOS operational system, and "Radzio! Modbus Master Simulator" software was tested. 2 meters of twisted pair cable and RS485 USB adapter with an FT232RL chip were used to connect a RutDevKit-PSoC62 with a laptop computer. The firmware example provided by Rutronik has only one high-priority task which polls the FreeMODBUS stack constantly. The RS485 was configured to work at a 115200kbit/s baud rate. No errors occurred reading the 6 bytes of initialized "Input Registers" every 10 milliseconds.

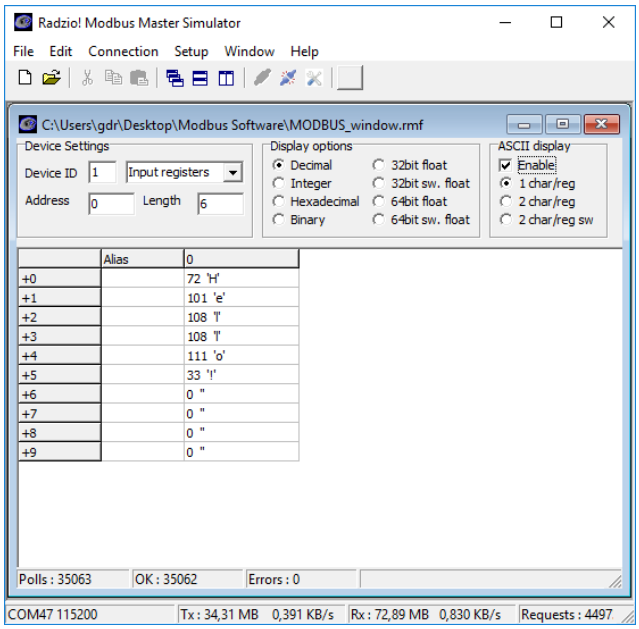

Fig. 3 Modbus master simulator software operating.

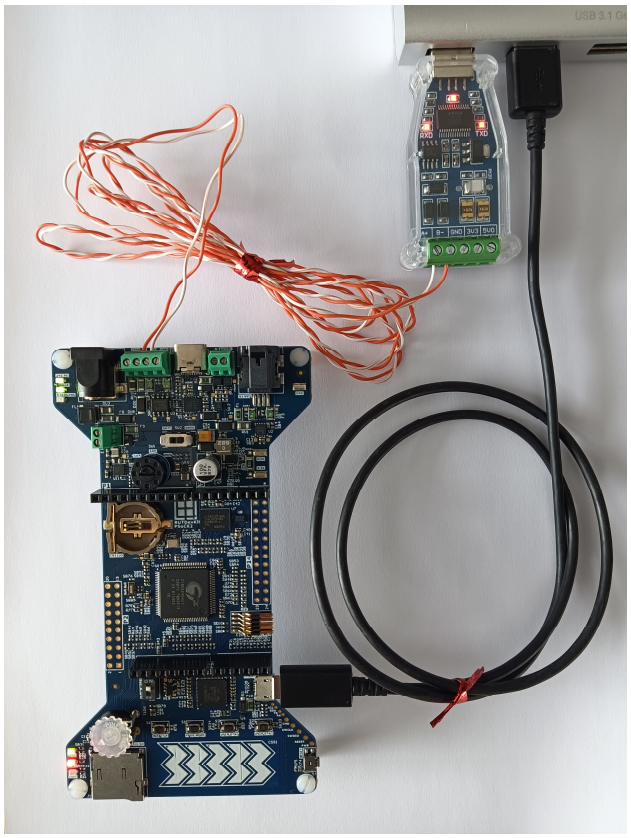

Fig. 4 RutDevKit-PSoC62 and FreeMODBUS protocol stack under test.

VI. Summary

Even though the Modbus protocol has been created almost four decades ago it still has a high potential for being used in today's industrial applications. RS485 interface is ideal in high noise environments where wired communication is not an issue. Combining the high performance of the PSoC62 MCU on the RutDevKit-PSoC62 and RS485 Modbus protocol allows developing modern industrial applications.

References

[1] " PSoC 6 MCU: CY8C6xx5 Architecture Technical Reference Manual (TRM)" User Manual, by Infineon (July 2020).
[2] " PSoC 6 MCU: CY8C62X5 Datasheet" Datasheet, by Infineon (November 2020).
[3] " SP3070E - SP3078E Family" Datasheet, by MaxLinear (February 2020).

Contact:
Gintaras Drukteinis
Technical Support Engineer
RUTRONIK Elektronische Bauelemente GmbH
Jonavos g. 30
44262 Kaunas
Lithuania
gdr@rutronik.com
www.rutronik.com