PCS RS40 Real Time Clock Application

ABSTRACT

The PCS RS40 is a 40kW Power Conversion System (PCS) intended for battery-based energy storage applications. The real-time clock function provided in the PCS RS40 must be set each time the PCS is powered on. This application note describes the nuances of the real-time clock feature and the resulting application considerations for the system designer.

TABLE OF CONTENTS

INTRODUCTION

The PCS RS40 is a 40kW Power Conversion System (PCS) intended for battery-based energy storage applications. One of the features provided by the PCS is a real-time clock function. The function tracks the date and elapsed time and is used by the data logging module to date and time stamp Fault History logs, Operating logs, and Event logs.

Because the PCS RS40 was designed to be maintenance free, i.e., no field service required, it does not contain a battery to power the real-time clock when bias power is removed. For this reason, it is necessary for the user to set the time and date each time the PCS is powered on. Normally these steps would be incorporated into the standard system initialization routine.

This document describes the details of the real-time clock function and how to properly interface with it. More detailed product information, including control register details, can be found in the User's Manual, UM-0061.

POWER ON/OFF RECOMENDATIONS

The PCS RS40 derives all its internal bias voltages off the high voltage DC battery inputs. This internal bias supply is controlled using the opto-isolated Bias Enable input. While this optoisolated input is asserted by applying 24VDC, the bias supply remains operating. When the input is deasserted, the bias supply will hold up for approximately 2.5 seconds to allow the microprocessor to finish its housekeeping tasks, after which the bias supply is powered off. These housekeeping tasks include saving the present time and date, as well as updating the operating logs in non-volatile memory. Because the Bias Enable input is used to trigger these shutdown housekeeping tasks, it is important that the system designer use the input accordingly.

⚠ CAUTION ⚠

Permanently tying the Bias Enable to 24V and using contactors on the battery inputs to turn the PCS on/off will circumvent the intended shutdown tasks and corrupt the real-time clock and data logs. Furthermore, Trystar recommends a 5 second delay between deasserting Bias Enable and removing battery voltage.

It is permissible to include contactors on the battery inputs, but they should be sequenced with the Bias Enable input. For example, when turning on, the contactors should be closed first, and then the Bias Enable input asserted. More importantly, when shutting down, the Bias Enable input should be deasserted first, followed by a 5 second delay before the contactors are opened.

When the PCS is powered on, it will read the time and date that was stored in non-volatile memory during the last power down (assuming the recommended power-down sequence was followed). These values are loaded into the RTC, and time keeping resumes from this point. If the user does not subsequently set the RTC, all time and date stamped data log events will use this stale information. Although this time and date may not be correct, data log events will at least be stamped in chronological order.

If the recommended power-down sequence is not followed (i.e., if the battery is disconnected without deasserting Bias Enable), the PCS is unable to save the most recent RTC time and date in non-volatile memory. In this situation, the values read by the PCS at power on will be the same. As a result, data log events are not guaranteed to be stamped in chronological order, and the recorded time and dates cannot be used to determine the sequence of logged events.

REAL-TIME CLOCK REGISTER FORMATS

Real-Time Clock – Date Format

The RTC date is stored as a 32-bit data word with the following format:

Bits 31 - 24	Bit 23 – 16	Bit 15 - 14	Bit 13 - 0
Day	Month	n/a	Year

For example, a date of 3/21/2021 would be formatted as follows:

Bits 31 - 24	Bit 23 – 16	Bit 15 - 14	Bit 13 - 0
21	3	0	2021

In hexadecimal format, the 32-bit register value would be 0x150307E5

Real-Time Clock – Time Format

The RTC clock time is stored as a 32-bit data word with the following format:

Bits 31 - 24	Bit 23 – 16	Bit 15 - 8	Bit 7 - 0
n/a	Seconds	Minutes	Hours (0-23)

For example, a time of 1:48pm would be formatted as follows:

Bits 31 - 24	Bit 23 – 16	Bit 15 - 14	Bit 13 - 0
0	0	48	13

In hexadecimal format, the 32-bit register value would be 0x0000300D

SETTING THE DATE AND TIME USING MODBUS

Data Organization

The Modbus protocol is based on reading and writing 16-bit “Holding” registers. As such, 32-bit registers must be read and written to as a pair of registers. Modbus uses a ‘big-Endian’ representation for addresses and data items. This means that bytes are organized such that the most significant byte is sent first. For example, a 16-bit register is organized as follows:

Modbus Register	1															
Byte	5								1							
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Similarly, a 32-bit holding register is organized as two comprising registers as follows:

Modbus Register	1		2	
Byte	0	1	2	3
Bits	31 … 24	23 … 16	15 … 8	7 … 0

Writing the Date and Time

For synchronization purposes, the date is internally latched upon writing to the Set Time register, therefore the date and time must always be set together. **REG 41525 – RTC Set Date** must be written to first, followed by a write to **REG 41527 – RTC Set Time**. If the user fails to write to **REG 41527 – RTC Set Time,** the date will not be set.

The 32-bit **RTC Set Date (REG 41525)** and **RTC Set Time (REG 41527)** registers can be written to using Modbus Function Code 16, “Write Multiple Registers”. The request and response message formats are illustrated below.

Request:

Slave Addr	Func Code	Starting Address	# of Registers	Byte Count	Register Values	CRC
8-bits	8-bits	16-bits	16-bits	8-bits	16-bits x n	16-bits

Response:

Slave Addr	Func Code	Starting Address	# of Registers	CRC
8-bits	8-bits	16-bits	16-bits	16-bits

As an example, to set the time and date to 1:48pm 3/21/2021, the following message can be sent to write the ***RTC Set Date*** and ***RTC Set Time*** registers on Slave 10:

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Slave Address	0A	Slave Address	0A
Function	10	Function	10
Starting Address MSB	A2	Starting Address MSB	A2
Starting Address LSB	34	Starting Address LSB	34
# of Registers MSB	00	# of Registers MSB	00
# of Registers LSB	04	# of Registers LSB	04
Byte Count	08	CRC MSB	crc_h
Register 41524 Value MSB	15	CRC LSB	crc_l
Register 41524 Value LSB	03		
Register 41525 Value MSB	07		
Register 41525 Value LSB	E5		
Register 41526 Value MSB	00		
Register 41526 Value LSB	00		
Register 41527 Value MSB	30		
Register 41527 Value LSB	0D		
CRC MSB	crc_h		
CRC LSB	crc_l		

There are a couple things to note in this example:

- The Modbus “Address” is one less than the register number. So, in this case the starting address is 41524 (0xA234) not 41525.
- Each 32-bit register is updated by writing to two 16-bit registers at consecutive Modbus addresses.
- The write to the **RTC Set Date** register is followed by a write to the **RTC Set Time** register to insure latching of the date register.

Reading the Date and Time

For synchronization purposes, the RTC time is internally latched any time the RTC data register is read. To correctly read the time, ***REG 41529 – RTC Get Date*** must be read first, followed by a read of ***REG 41531 – RTC Get Time.*** These 32-bit registers can be read using Modbus Function Code 3, “Read Holding Registers”. The request and response message formats are illustrated below.

Request:

Slave Addr	Func Code	Starting Address	# of Registers	CRC
8-bits	8-bits	16-bits	16-bits	16-bits

Response:

Slave Addr	Func Code	Data Length	Data … Data		CRC
8-bits	8-bits	8-bits	16-bits	16-bits	16-bits

As an example, the following message can be sent to read the ***RTC Get Date and Get Time*** registers from Slave 10. The returned date and time are 1:48pm 3/21/2021.

Request		**Response**	
Field Name	*(Hex)*	*Field Name*	*(Hex)*
Slave Address	0A	Slave Address	0A
Function	03	Function	03
Starting Address MSB	A2	Byte Count	08
Starting Address LSB	38	Register 41528 Value MSB	15
# of Registers MSB	00	Register 41528 Value LSB	03
# of Registers LSB	04	Register 41529 Value MSB	07
CRC MSB	crc_h	Register 41529 Value LSB	E5
CRC LSB	crc_l	Register 41530 Value MSB	00
		Register 41530 Value LSB	00
		Register 41531 Value MSB	30
		Register 41531 Value LSB	0D
		CRC MSB	crc_h
		CRC LSB	crc_l

As with the write example, there are a couple things to note:

- The Modbus “Address” is one less than the register number. So, in this case the starting address is 41528 (0xA238) not 41529.
- For synchronization purposes, the data and time are read together by first reading the ***RTC Get Date*** register and then the ***RTC Get Time*** register.

IMPORTANT NOTICE

The information provided herein is believed to be reliable; however, Trystar assumes no responsibility for inaccuracies or omissions. Trystar assumes no responsibility for the use of this information, and all use of such information shall be entirely at the user’s own risk. No patent rights or licenses to any of the circuits described herein are implied or granted to any third party. Trystar’s publication of information regarding any third party’s products or services does not constitute Trystar’s approval, warranty, or endorsement thereof. Customers are responsible for their applications using Trystar products.

15765 Acorn Trail / Faribault, MN 55021 / 507.333.3990
Document Number: 1901TN0526