An Embedded Modbus Compliant Interactive Operator Interface for a Variable Frequency Drive Using Rs 485

Adnan Shaout
The Electrical & Computer Engineering Department
The University of Michigan – Dearborn, MI U.S.A.

Khurram Abbas
The Electrical & Computer Engineering Department
The University of Michigan – Dearborn, MI U.S.A.

***Abstract*—The paper proposes the architecture and software design of a Modbus Compliant Operator Interface Panel (MCOIP) for a high speed Variable Frequency Drive (VFD) – a state of the art embedded design that offers several key advantages over the existing proprietary industrial models in use today. The use of serial Modbus RTU communication over RS485 allows an economically feasible, open source, vendor neutral, feature laden, robust and safe operating model. Through the use of an ARM based RISC microcontroller, the low response time of the design makes the human machine interface more real-time and interactive.**

***Keywords*—Modbus RTU; Variable Frequency Drive Operator Panel; Modbus Master VFD**

I. INTRODUCTION

A Variable Frequency Drive is an embedded system that controls the speed and frequency of a motor. The system consists of a drive Operator Interface Panel along with the main drive controller as depicted in Figure 1. A high speed VFD typically requires the user of a remote human machine interface that can control and monitor the drive. Typically, this is done through the use of a panel either mounted on the chassis or remotely through a very short shielded cable. These panels allow a human operator limited functionality mainly run, stop and speed control. The panels usually have a primitive display to indicate drive speed or consist of light emitting diodes to indicate status of the drive. The drive parameters are factory programmed according to the specifications of the customer's motor ratings as indicated by the motor name plate. To operate the VFD with different rated motors, the control board of a drive would have to be shipped back to factory. Furthermore, control of the VFD through applications such as a LabVIEWTM environment to build a sub Virtual Interface is not easy for machine integration due to the proprietary communication interface developed by VFD manufacturing vendors.

High speed VFDs also known as AC Motor Drives are used in very specialized industrial applications such as very high precision grinding and milling. An exhaustive review of commercially available VFD brochures was completed to ensure the need for this design approach. Existing VFDs require an operator to be in close proximity which often poses a safety concern with the acoustic noise and the inhalation of fine particle dust [1, 2].

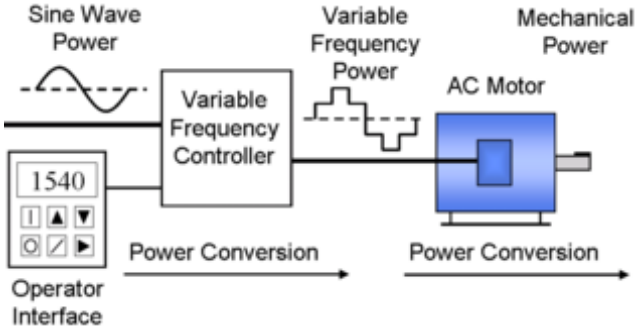

Fig. 1. High level illustration of a Variable Frequency Drive (VFD)

This limitation is mainly due to the fact that existing Operator Interface Panels on VFDs use primitive serial communication such as RS232 and the standard puts restrictions on the distance between the operator interface panel and the actual VFD [3]. The proposed design uses RS485 that can allow the remote Operator Interface Panel to be placed several hundred meters away reliably alleviating many of the safety concerns by allowing the operator to control the VFD from a safe distance or isolated premises.

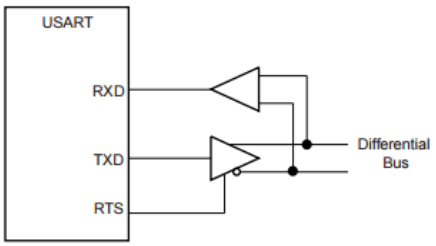

Fig. 2. Connection to a RS485 Interface

The control board of a VFD contains hundreds of drive parameters which define the way the drive manipulates the frequency and voltage to control the motor. An extensive literature review and survey of commercially available VFDs reveals that for a consumer to use a differently rated motor would require factory reprogramming [4, 5, and 6]. A convenient way to modify these drive parameters on the fly would make motor control more customized. Furthermore, the design also opens up the realm of providing access to the complete parameter table stored in the drive.

The afore mentioned RS485 communication illustrated in figure 2 also offers higher bandwidth than RS232 in present day VFD Operator Interface Panels to make this increased flow of bidirectional data between the VFD and the operator station. This would require adding more peripherals namely a graphics display and a full keypad. The use of specialized data structures to handle the read and write requests between the VFD and the Operator Interface Panel is also proposed in this design. This is achieved using Modbus at the application layer of the Open Source Interconnection Model which is an open source industrial protocol suited for this application. This is particularly beneficial as the biggest challenge with VFDs is the lack of standardization and use of proprietary software for communication. It also makes it easy for the user of a VFD to replace an Operator Interface Panel with prepackaged Modbus ready VIs for LabVIEW™ integration. In applications where the VFD powers up a high speed motor to several thousand RPMs in a matter of seconds the added responsiveness of the system makes the speed display (and other parameters) more real time. This is a desired change as in existing VFD Operator Interface Panels which use typical seven segment displays for speed this real time ramp up of speed is not observed.

The paper is organized as follows: section 2 will present requirements for the Modbus Compliant Operator Interface (MCOIP) Panel for a high speed Variable Frequency Drive, section 3 will present the design of the MCOIP, section 4 will present simulation and results, section 5 will present the performance of the system and section 6 will present conclusions.

II. Requirements

A. Functional Requirements

Firstly, although an Operator Interface Panel for a VFD is primarily a communication device it also serves other purposes. The ability to read/write parameters adds complexity to the main loop of the embedded device. It requires the use of various peripherals of the microcontroller to interface with the graphics display, keypad matrix etc. Furthermore, the requirement to meet the responsiveness of the VFD is further constrained if the main loop encompasses other tasks such as display refresh, keypad scanning etc. in addition to the communication routine.

The programmability of the design will allow the system to be used with a wide range of different motor ratings. Depending on the application, there could be well over 200 drive parameters that could be modified. This feature would require interactive communication with the VFD through the use of the operator interface to modify drive motor parameters (individually on a per parameter basis or in bulk).

The design allows a single operator interface to be used with various master units (multiple drives/multiple motors). Various industrial applications require one operator unit and multiple VFD units (e.g. high precision grinding/milling applications). Presently, this is done through daisy chaining and signal mirroring which requires the use of additional hardware. Modbus is suited for such single slave multiple master implementation and the software design techniques to accomplish this will be described in greater detail in the design.

Since the operator panel is always communicating with the VFD in an open control loop, various drive parameters are always being retrieved by the operator interface. This is to ensure that the human operator is always visibly aware of the drive status while navigating through various different menus of the operator interface. Modbus has special commands to read large registers in one go which are purposely placed contiguously.

Lastly, a requirement of the design is to allow the VFD operator to be able to connect the drive several thousand feet away from the actual drive [7]. Existing drives systems use RS232 which imposes restrictions on the distance between the operator interface and the drive controller. This limitation can potentially add risk to the operator in certain applications where high speed, high voltage AC motors are used. Using RS485 which uses a differential communication model can hence offer more safety by allowing the operator interface to be located farther away with very little error rate [8]. Modbus standards limit this distance to 1000m [9]. Furthermore, Modbus also defines the type of connector to be used along with the pin out.

This paper discusses the various aspects of the project progress in conjunction with the waterfall model by describing the requirements, design, implementation, verification (system performance).

B. Non-Functional Requirements

The functional requirements stated add stringent real time constraints on the embedded system. To overcome some of these timing constraints the design would incorporate a phase locked loop with an aptly selected crystal oscillator. The embedded coding would also utilize a highly optimized compiler for code efficiency while keeping in mind the space limitations to keep the cost down.

The use of timer and peripheral interrupts can delay execution of tasks within the main system loop and the execution of the operator interface will require a robust scale of reentrancy.

The paper aims to put forward a design which should be compatible with practically any Modbus compliant controller by simply plugging the operator interface into the port. This non-functional requirement would require some robustness in the design.

This embedded system should leverage the use of a watch dog timer to prevent deadlocks and also offer error handling to ensure safe drive operation. The system model and state design machine should ensure all process deadlines are met. A brownout circuit will also be explored in case of loss of power detection.

The operator interface being a slave Modbus system should be interchangeable with any Modbus compliant control methodology. The software inside the embedded device should be designed such that it can be ported and used in a National Instruments LabVIEW™ environment on a computer.

Table I summarizes the key advantages that the proposed design offers over existing VFD operator panels.

TABLE I. KEY ADVANTAGES OF MCOIP

Features	MCOIP
Interactivity	• Full keypad array (numeric). • Fast response time (more real-time).
Programmability	• Through keypad or any Modbus compliant slave. • High data transmission rates.
Reliability	• Increased noise immunity. • Reliable communication.
Flexibility	• Remote operation up to 1000m. • Vendor neutral.

III. DESIGN

The ATMEL AT91SAM7S series of RISC Processor was chosen for this embedded system. It incorporates the ARM7TDMI ARM Thumb Processor and provides 32 bit RISC architecture. The RISC architecture is suited for Modbus in an industrial environment as this provides the design immunity to electromagnetic interference [10]. Figure 3 shows the peripheral block diagram of the series and illustrates the outlined peripherals to be used this design [11]. This was determined in lieu of the prior requirements set forth while following the software waterfall model.

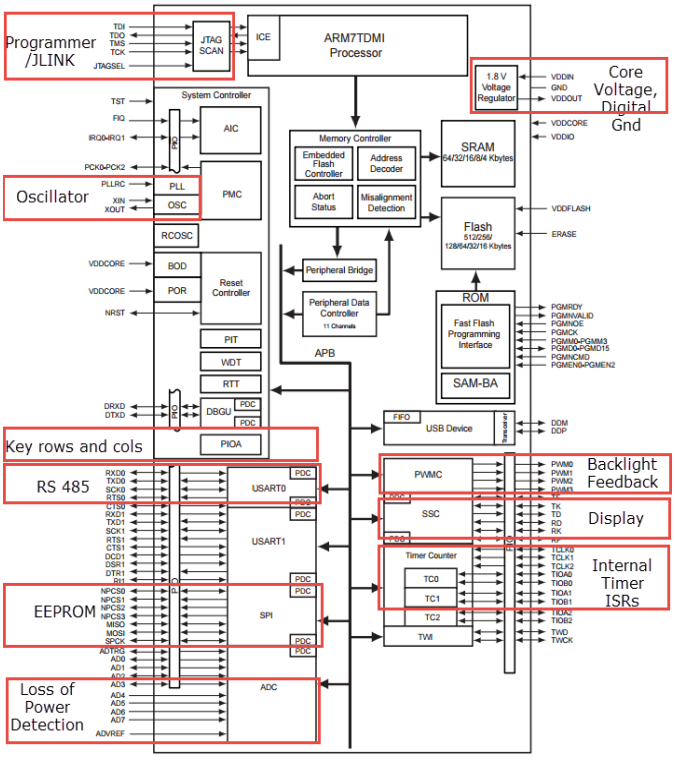

Fig. 3. AT91SAM7S Block Diagram and Pheripherals Used

A readily available oscillator/crystal with a frequency of 18.432MHz can be used [11].

$$Clock\ Frequency = 31.334MHz$$

$$1\ instruction = {}^{1}/_{31334000\ HZ} = 31.91\ \text{ns}$$

$$1\ instruction = {}^{1}/_{31334000\ HZ} = 31.91\ \text{ns}$$

$$Max.Instructions = {}^{50ms}/_{31.91ns} = 1{,}566{,}906$$

The design in this paper focuses primarily on the main functional aspects of the Operator Interface Panel for the VFD although some of the other functions which may be outside the scope of the paper are mentioned. The design was suited to fit the needs of the AT91SAM7S and its available multiplexed peripherals along with a prototype printed circuit board (PCB). However, it has flexibility and portability to be applied to other microcontrollers and a different schematic implementation of a hardware design. Conceptually, the crux of the proposed state of the art design described next is what makes the implementation unique and should be adhered to as much as possible.

Modbus protocol was originally developed and published for industrial PLC communication in the 70s. It supports RS323, RS422, RS485 communication interfaces as well as Ethernet interface (TCP/IP). Figure 4 illustrates the place of Modbus pertaining to the OSI model. Modbus facilitates communication between devices and dictates how devices send requests, responses, handle errors and records. It uses the "big-Endian" for representation of Big-Endian data item addressing i.e. most significant byte sent first if more than a single byte transferred/received.

Register addressing scheme in Modbus will be used to accomplish the latter. A VFD can design the parameter table in any way desired as long as it adheres to Modbus registering format [12]. This makes the Operator Interface Panel (Master) universally compatible with any Modbus Slave VFD.

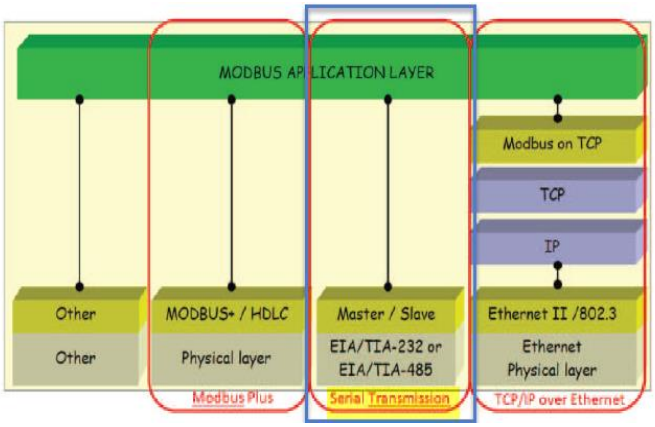

Fig. 4. Modbus and the OSI Layer

The data field is closely related to the function code and can vary with what is sent as shown in Table II.

Modbus also defines the physical connectivity for the serial RTU implementation [9]. It requires the use of RJ45 serial connector as shown on the right on Figure 5. The pin out and the color coding of the signals of the twisted pairs is also standardized as shown and implemented.

TABLE II. MODBUS FUNCITON CODE

Function code	Name	Function
01	Read Coils	Read the current status of coils
02	Read Input Discrete	Read the current status of discrete input
03	Read Multiple Registers	Read the contents of holding registers
04	Read Input Register	Read the contents of Input registers
05	Write Single Coil	Write a single output to either ON or OFF
06	Write Single register	Write a binary number to single holding register

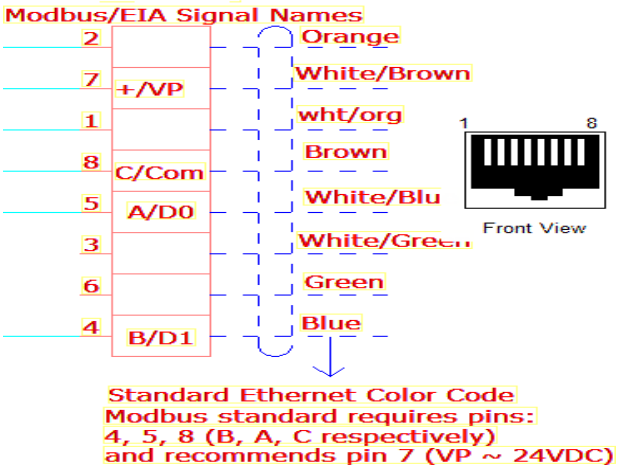

Fig. 5. RJ45 Jack EIA/Modbus Signal Names

This controller allows formation of an asynchronous state machine which features a main polled loop. Figure 6 depicts the high level state machine in terms of a flow chart. Per the non-functional requirements of this design the loop should execute within 50ms. The design is completely embedded and autonomous. It powers up and loads the Parameter Table from the drive. If that process fails the operator interface loads the default Parameter Table from code. In the main loop it performs a check to see if a key was pressed, carries out Modbus communication and refreshes the graphics display. This main polled loop is allotted a maximum total execution time of 50ms. A loop execution time of larger than 50ms will result in the design losing a responsive feel and will render the design non-real time.

A. *Interrupts*

- A timer interrupt set to occur every 1ms is used to implement key scanning to scan and check key entry against every row column combination. It is also used to enough time has passed (50ms) to flash the data entry cursor and then to process the key to determine if a key was pressed by the operator and which one. Figure 7 is a flow chart depicting the ISR for Timer 1.

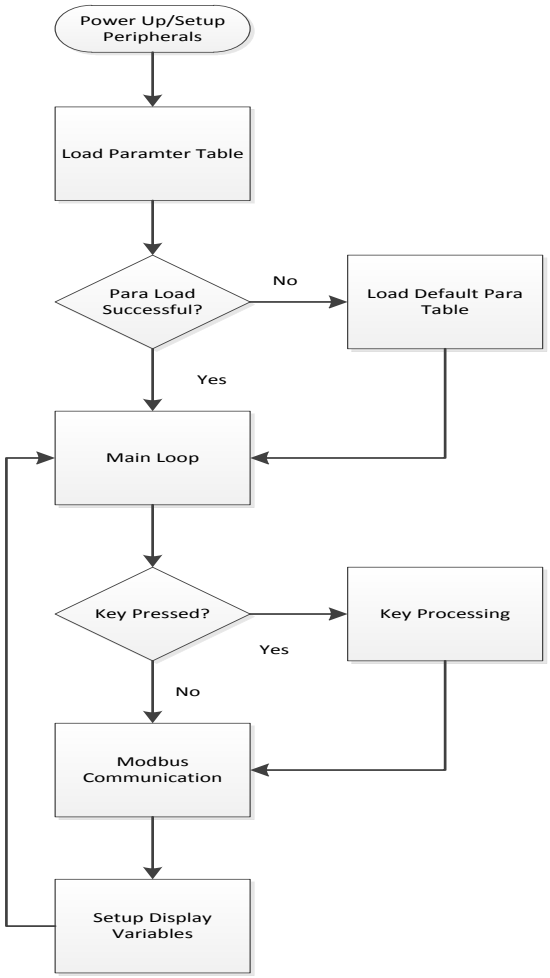

Fig. 6. Operator Panel Interface Main Call Graph

- Another timer interrupt is set to trigger every 250usec. Inside this handler a counter called Frame Silent Interval Counter is used to keep track of a 3.5 character delay. Before transmitting via Modbus this is the counter which is checked to see if that delay has not yet been met. Another counter keeps track of whether enough time has passed to determine whether a message response timeout has occurred or not (meaning the Modbus slave i.e. VFD control board in the VFD did not reply). Figure 8 is a flow chart depicting the ISR for Timer2.

- A Universal Serial Asynchronous Receiver Transceiver timer is set to trigger every time a character is received. The ISR first checks to see if the transmission for this character is complete. It would then go ahead and put the character in the Modbus Receive Buffer provided the buffer size has not exceeded 256 Bytes in conjunction with Modbus stipulations on maximum buffer size [12]. The ISR then resets the counters tracking the response timeout and the frame silent interval. The routine then checks to ensure that a 1.5 character timeout has not occurred again per Modbus stipulations indicating the frame has been received [12]. If it has occurred the interrupt is disabled and the

response time counter is reset along with the frame silent interval counter. An end of frame flag is then set to true indicating successful receives of the Modbus frame. Figure 9 is a flow chart depicting the ISR for the UART Receive.

Since the design primarily happens to be a communication device the highest priority is assigned to the USART Receive Interrupt. An incoming reception from the Modbus Slave (the VFD) is the absolute critical task which should be services immediately. The only exception could potentially be a Analog to Digital Interrupt which could potentially be triggered on a drop in voltage of the main power indicating the loss of power. This can be used in conjunction with a loss of power indication circuit to immediately service some tasks that require immediate attention such as locking EEPROM to prevent corruption, halting master requests gracefully or turning graphics display backlight off. However, these are beyond the scope of the proposed design and in normal operation the Modbus reception of a frame through the use of the USART receive interrupt has the highest priority. Table III depicts the interrupts associated with the design along with their priorities and rate/triggers.

TABLE III. INTERRUPT PRIORITIES

Interrupt	Priority (5 being highest)	Rate/Trigger
USART0	5	Receive character (1 byte)
TIMER0	4	Every 1ms
TIMER2	3	Every 250usec

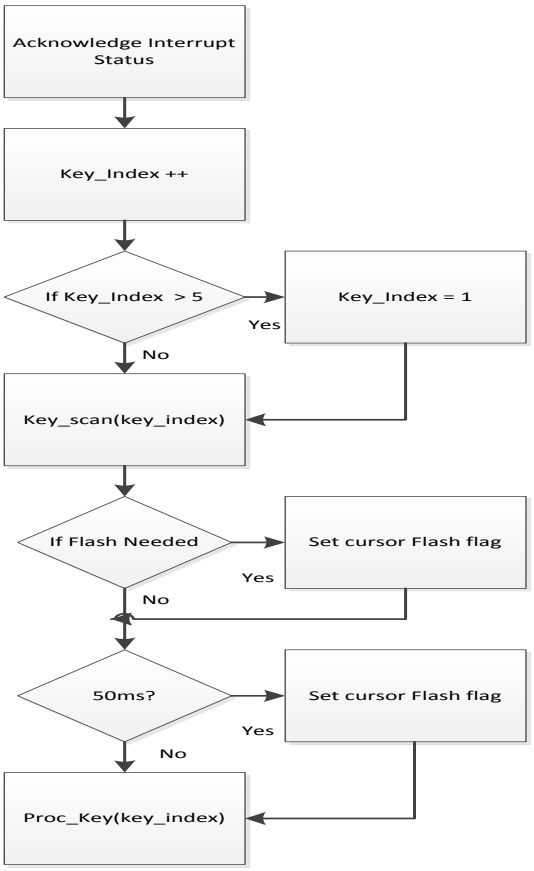

Fig. 7. Timer 1 (1ms) – Interrupt Service Routine

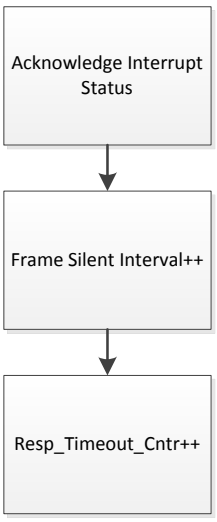

Fig. 8. Timer 2 (250usec) – Interrupt Service Routine

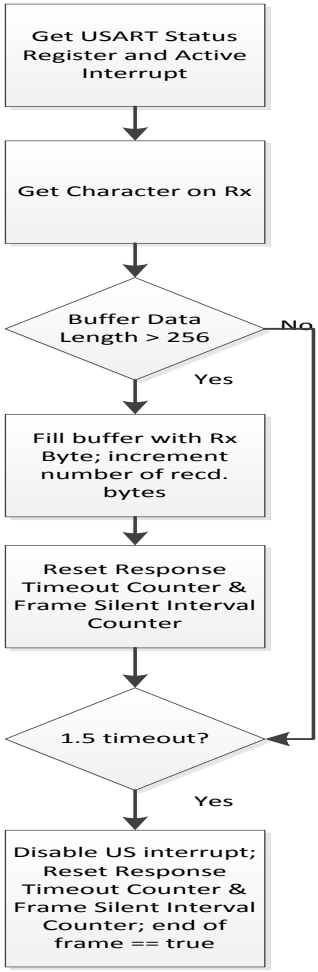

Fig. 9. USART Receive Interrupt Service Routine

B. Modbus Communication

Modbus RTU serial communication is carried out using RS485. This requires using the Universal Synchronous Receiver Transceiver (USART) in RS485 mode. As defined in the Modbus specifications a stop bit, even parity, 8 bit mode is

used [12]. In order to meet stringent requirements Direct Memory Access (DMA) is used for transmit. DMA with Modbus receive cannot be used because a received frame can be variable length and indeterminate. The Modbus frame is received through the use of a USART ISR described previously.

The Modbus communication routine strictly adheres to the requirements set forth by the standards both in terms of timing and data structures [12]. Certain status parameters indicate critical information such as the drive run statistics, speed, fault etc. depending on what section of the menu a user is in. These are purposely placed contiguously in the Modbus memory block for easy of retrieving. Modbus uses Cyclic Redundancy Check to ensure error free reception and transmission of frames. This is done by calculating the CRC value that uses a standard 16 bit generated polynomial to check out 16 bit-check code for any length information fields [12]. Any Modbus frames received are also checked for CRC to ensure errors have not occurred [9, 12]. Modbus frames prepared for transmissions are also appended with the checksum calculation to satisfy Modbus requirements and to ensure the slave receives error free frames.

For timing the communication routine ensures that before it sends out a new frame the silent interval has passed. There are certain drive parameters that the Operator Interface Panel is always reading in every execution of the main poll loop which executes the Modbus Communication block. Figure 10 shows the Modbus Slave (VFD) state diagram while Figure 11 shows the Modbus Master (Operator Panel Interface) state diagram.

In case of a transmission error the frame retransmission is tried thrice with a delay of 10ms to avoid bus contention with the Master. If there is a problem that persists then the slave should do back off retries i.e. introducing an extended delay between each retry to reduce processing on the slave (VFD) side. In such a scenario the display would display no data for the status or the parameter being checked. As soon as communication is reestablished the data would appear on the display. The user is still able to browse through the menus and do limited operations on the Operator Interface Panel in the event of no communication from the Slave (VFD). This is a stark improvement to the traditional VFDs in the industry today where an operator panel with lack of interactivity to begin with becomes completely unresponsive until communication is reestablished.

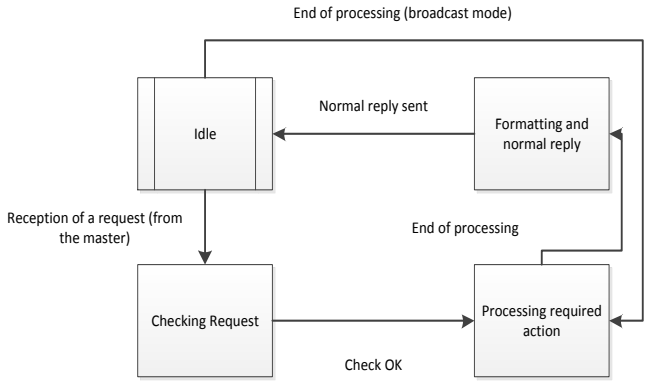

Fig. 10. Modbus Slave State Diagram

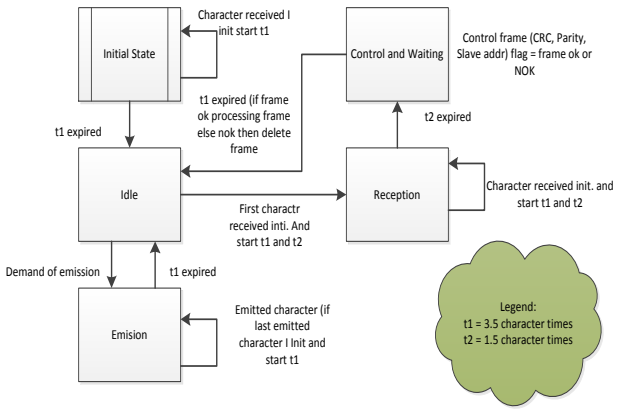

Fig. 11. Modbus Master Transmission Mode State Diagram

C. Keyscanning

A requirement of the Operator Interface Panel required a user to be able to do data entry in order for drive parameter programming. This is made possible through the addition of a 5 x 5 key matrix using tactile switches. This is introduced to meet the requirements to make the Operator Interface Panel programmable and fully interactive with rich human-machine interfacing when it comes to drive parameters.

TABLE IV. KEYPAD ENTRIES

Key	Function
Status	Go to Status Screen
Parameter	Go to Parameter Screen
Setup	Go to Setup Screen
Fault	Go to Fault Screen
Control Panel	Go to Control Panel Screen
0 to 9	Numeric Data Entry
Decimal	Decimal Entry
Clear	Clear Entry
Enter	Accept Entry
Escape	Cancel Data Entry
+/Up	Scroll Up
-/Down	Scroll Down
Start (I)	Start Command
Stop (O)	Stop Command
F1	Future Use
RST	RESET Microcontroller

Key scanning is implemented by putting a software routine to scan the 5 columns rows inside a 1ms timer as indicated in Figure 7. The general purpose I/O pins of the AT91SAM7S translating to 5 rows and 5 column can be used for this purpose and should be defined as internal pull ups. The 5 column pins should be defined as inputs only. The un-driven row pins should be weakly pulled up (open source).

It is assumed that only one key is to be pressed at a time. If there is a second key pressed either at the same time (theoretically) or right after the first one, it will be ignored by the key scanning and key processing routines. In case of a simultaneous press the order of the key scanning process dictates what key takes precedence. Table IV highlights the available options of the 25 keys. This can be customized based on the needs of Control Interface Panel.

The timer interrupt triggering every 1ms shown in Figure 7 does key scanning and processing. The key scanning cycles through the rows and columns to determine what is pressed where as the key processing routine implements de-bouncing and goes ahead and aptly stores the pressed keys into a "First In First Out" (FIFO) buffer. A gear count is also used in case the same key is repeatedly pressed to speed up scrolling processes. The main poll loop checks to see if a key is pressed by reading the FIFO as indicated in Figure 6. There are certain menu screens of the Control Interface Panel where pressing a key can potentially trigger a data entry routine to meet the requirement of making the design programmable. This could be a parameter entry. Hence, every key is handled on a case scenario and special consideration of the state of the Control Interface Panel keeps track of the data entry.

IV. Simulations & Results

A. Results

To test the design the Modbus Slave Simulator is programmed with dummy variables. The Modbus Slave Simulator is set up using the communication settings as shown below which are on par with the design code on the USART peripheral settings i.e. 115Kbps Baud Rate, 8 Data Bits, Even Parity and 1 Stop Bit.

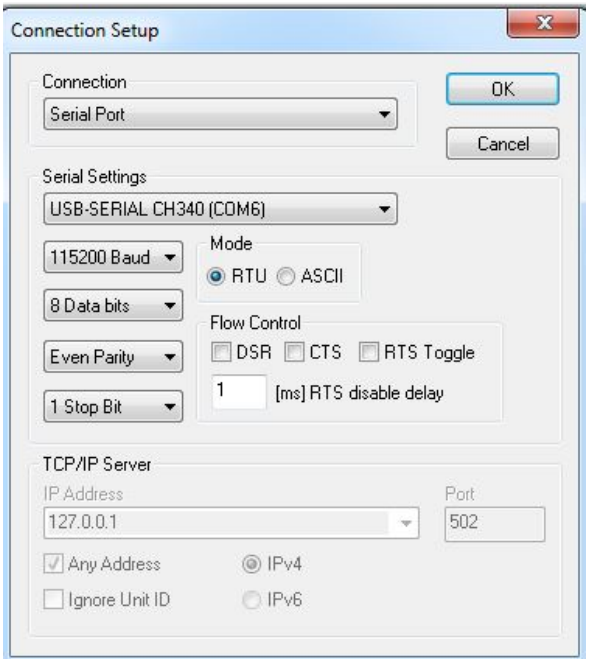

Fig. 12. Slave Communicating Settings

As an example parameter 2.04 of the drive is chosen arbitrarily. From the parameter table 2.04 in figure corresponds to Register Number 40097 as shown in Figure 13.

Modbus	Index	Address	#	Parameter Description	Unit
				Motor Control [All Motors]	
40093	46.5	0x5D	2.01	Voltage Gain	
40095	47.5	0x5F	2.02	Scaling Factor (Current)	
40097	48.5	0x61	2.03	Electrical Frequency	Hz
40099	49.5	0x63	2.04	Frequency Gain	

Fig. 13. Parameter Table Entry of Parameter 2.04

The electrical frequency of a motor is typically a floating point value. To optimize our design floating point variables were not used in the design. Floating point values were handled by using Binary 16 and Binary 32 variables with scaling. For this parameter, entries were defined in the parameter table; a snippet shows in Figure 13. The store radix indicates that the value would be using a 3rd decimal place. The maximum and minimum are merely for a local master bound check for this particular application and do not necessarily represent the actual data structure's maximum and minimum. Hence an out of bound value was retrieved by the master is capped to the maximum applicable value as defined by the parameter table from Figure 13 for this parameter.

A dummy register entry was made in the Modbus Slave Simulator in the register 40097 as shown in Figure 14. The value entered is 23 (0x17).

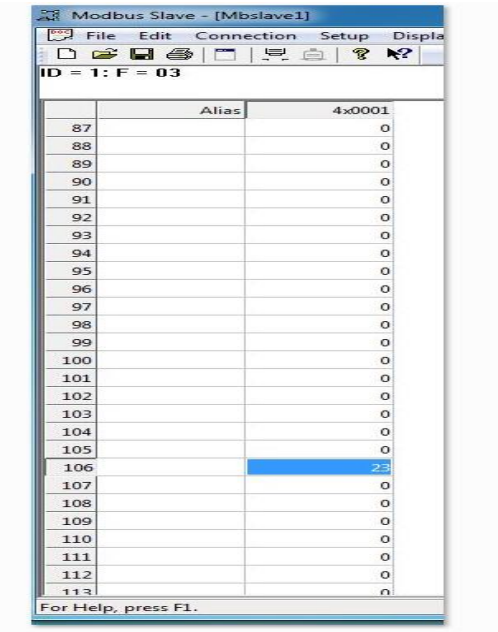

Fig. 14. Slave Communicating Settings

The Modbus poll request for this frame would be:

01 03 00 60 00 02 45 D7
Device Address: 0x01
Function: 0x03 (Read Holding Register)
Point Address: 40097
Point Code: 2
Checksum: No Error

This is verified by the communications log monitoring of the Modbus Slave shown in Figure 15 (or through the use of a Logic Analyzer connected to the RS485 transceiver). The Modbus response to this poll would be:

01 03 04 00 00 00 17 BA 3D
Device Address: 0x01
Function: 0x03 (Read Holding Register)
Point Count: 2 (Implied by byte count of 4)
Point Index 1: 0x00
Point Index 2: 0x17
Checksum: No Error

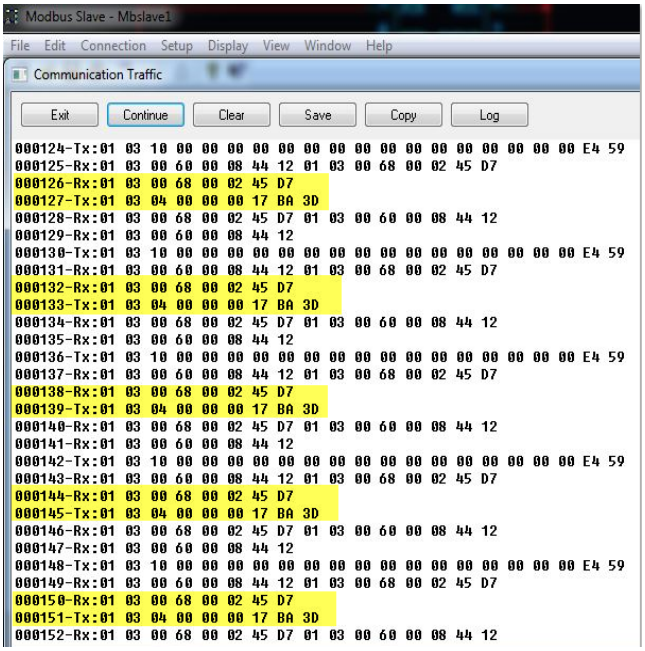

Fig. 15. Modbus Slave Communication Log

This value is translated using the data structure definition in Figure 14 by the case check for the display parameter routine and displayed on the screen as shown in Figure 16 which shows an actual snippet while this was performed.

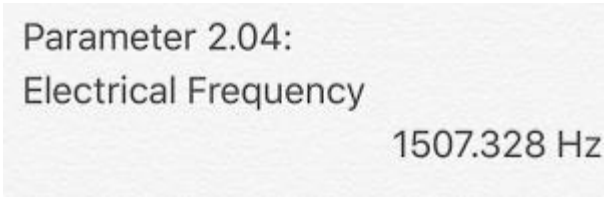

Fig. 16. Parameter 2.04 value retrieved from Slave

B. Simulation

Since Modbus limits the length of the RS485 cable to 1000m a test was done to ensure the design meets the requirements of being remotely deployed. A 1000m CAT5 Ethernet Cable could be used with the design and a count of errors through the internal error log as well as the Modbus Slave Simulator could be performed over time to count CRC errors. Since a commercially available 1000m Ethernet cable is expensive, a signal integrity simulation was performed using Mentor Graphics Hyperlynx to assess the effect of Bit Rate and Frame Delay Variation (also known as Jitter) vs. cable length. This was done through the use of exporting the schematic model available from the Mentor Graphics PADS Schematic to the Hyperlynx simulation product. The schematic is shown in Figure 17.

A differential voltage threshold of -100mV and +100mV was used to demonstrate a practical (and not a perfect) receiver) between pins A/D0 and B/D1. The simulation charted in Figure 18 show found to be within acceptable levels for up to 850 meters which falls slightly short of the 1000m for this design as set forth by Modbus (highlighted in yellow in Figure 18).

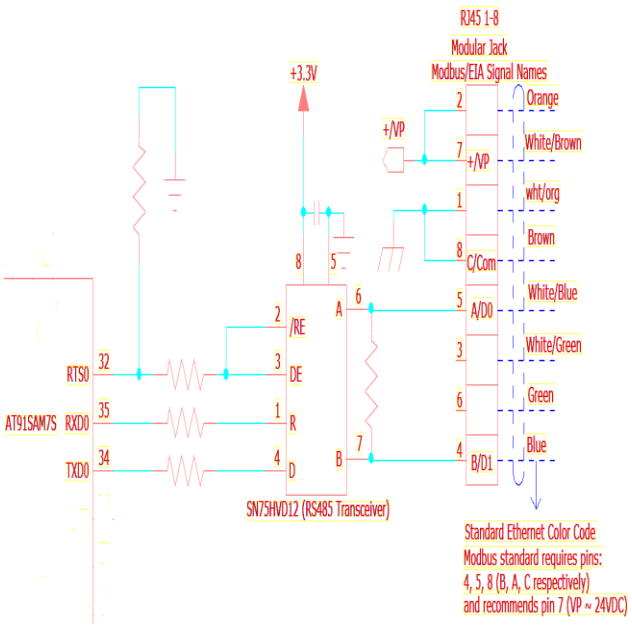

Fig. 17. Schematic of the RS485 circuit

This is a limitation in the Hyperlynx Simulation Tool however the design specifications of Modbus guarantee operation up until 1200m. The Baud Rate used in this design (115Kbps) is also highlighted in purple.

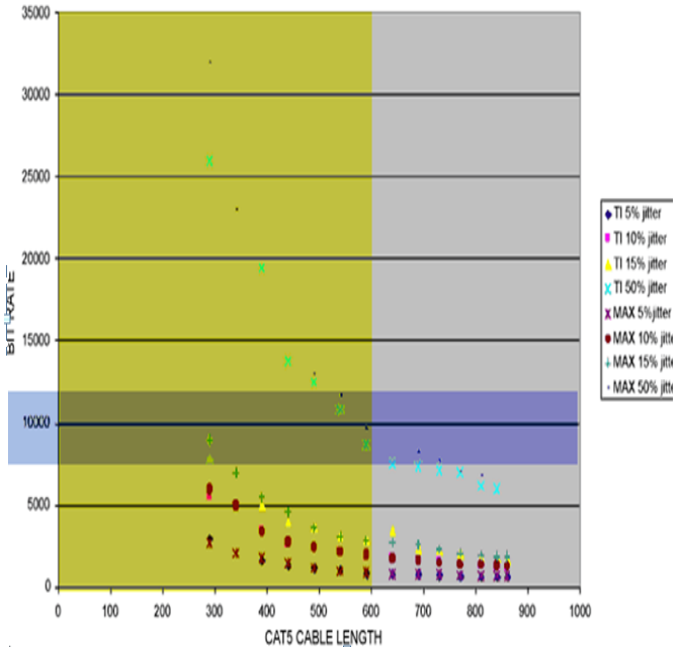

Fig. 18. Chart showing frame delay variation with Bit Rate (Kbps) vs. Cable Length (m)

V. System Performance

Due to the nature of the application the timing requirement of 50ms discussed in the requirements section is critical. Many techniques were used both to measure and modify the system response time to ensure that all the tasks complete without the

required time. With the use of these techniques it was found that the main loop on average operation on a typical screen of the Operator Interface Panel was measured to be:

Worst case execution time for Main Poll Loop = 32ms

Worst case CPU utilization (no retransmissions) = 64%

Some of these techniques to evaluate and improve the system performance of the device are listed below.

A. Probing

Various routines inside the main polled loop were individually measured to determine the compositions. This was easily implemented through the use of probing an unused I/O pin. The multiplexed I/O pin could be set before the start of the routine and then cleared after the routine is finished. A digital oscilloscope (a logic analyzer could be used here as well) was used to determine the high time of the wave form indicating the time taken for the routine. The response times of the interrupts were also evaluated to ensure no unnecessary latency exists.

Decisions to use the SSC peripheral for updating the display and DMA for USART Transmit were primarily based on the results of the probing. SSC and DMA access the RAM independently without bogging down the processor allowing plenty of headway into the 50ms main polled loop.

B. Communication Response Times

Modbus communication response times from a simulation Modbus Slave Tool are exported and sent over to an open source software to perform an analysis [13]. Status 1 screen has the most amount of communication and represents the worst case scenario. For this reason it is chosen for testing response times and results depicted in Table V for the average and maximum response times (all under 50ms).

TABLE V. STATUS 1 RESPONSE TIMES

τ	Avg. RT (ms)	Max. RT (ms)
1	43.238	45.642
2	41.875	42.327
3	41.056	43.293
4	40.240	43.549

C. Inline Assembly Code

In scenarios instances involving initialization of peripherals where exact timing is necessary; inline assembly was used to ensure complete control over timing through the use of modules based on no operations instructions.

D. Optimized Compiler

An IAR Embedded Workbench Compiler for ARM processors was used to compile, debug and build the code. A Segger J-Link JTAG was used to flash the microcontroller. A balanced approach was taken with regards to optimization for speed vs. code space.

E. Dissassembly

In order to ensure optimized compiler behavior was not causing undesired results the disassembly was used as a means to verify correct operation. This was especially useful in the ISR to ensure it is short and simple and does not increase latency.

VI. CONCLUSION

The paper has presented the architecture and a software design of a Modbus Compliant Operator Interface Panel (MCOIP) for a high speed Variable Frequency Drive (VFD). The use of serial Modbus RTU communication over RS485 allows an economically feasible, open source, feature laden, robust and safe operating model. The use of specialized data structures to handle the read and write requests between the VFD and the Operator Interface Panel was proposed in this design. This is achieved using Modbus at the application layer of the Open Source Interconnection Model which is an open source industrial protocol suited for this application. The programmability of the design allowed the system to be used with a wide range of different motor ratings through the use of an ARM based RISC microcontroller. The low response time of the design made the human machine interface more real-time and interactive. Based on this proposal, future implementations can be designed to allow functionality with other variants of Modbus (namely TCP/IP) and other vendor neutral open source communication protocols.

REFERENCES

[1] N. Hashemi, L. R. Lisner, D. Holmes. “Accoustic Noise Reduction for an Inverter-Fed Three-Phase Induction Motor,” Conference Record of the IEEE, 2004, pp. 2030-2035 vol 3.

[2] “Dust and its control,” Occupational Safety and Health Administration.“https://www.osha.gov/dsg/topics/silicacrystalline/dust/chapter_1.html (last accessed 10/27/2015).

[3] B. Carlson. “Maximum transmission distance for RS-232 Computer Cables,” Pulp and Paper Ind. Tech. Conf., 1988, pp. 86-93.

[4] J. Kimbrell. “Top 10 Tips for Specifying VFDs,” Design Engineering, 2014, pp. 30-33 vol 60 issue 1.

[5] [12] B. MB and Vishwanath, “Hegde Energy Conservation using Variable Frequency Drives of a Humidification Plant in a Textile Mill”, 2015 International Conf. on Power and Advanced Control Engineering (ICPACE).

[6] [13] G. Morris and B. Weiss, “Driving Energy Efficiency with Design Optimization of a Centrifugal Fan Housing System for Variable Frequency Drives”, Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2012 13th IEEE Intersociety Conf.

[7] G. Guarese, F. Sieben, T. Webber, M. Dillenburg, C. Marcon. “Exploiting Modbus Protocol in Wired and Wireless Multilevel Communication Architecture,” Brazilian Symp. on Comput. Sys. Eng., Natal, 2012, pp. 13-18.

[8] V. Kumar. “Overcoming Data Corruption in RS485 Communication,” IEE Int. Conf. on Electromagnetic Interference and Compatibility, 1995, pp. 9-12.

[9] “Modbus over Serial Line Specifications and Implementation Guide V1.02,” www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf (last accessed 10/27/2015).

[10] K. Wang, D. Peng, L. Song, H. Zhang. “Implementation of Modbus Communication Protocol Based on ARM Cortex-M0,” IEEE Int. Conf. on Systems Sci. and Eng., Shangai China, 2014, pp. 69-73.

[11] “AT91SAM ARM-based Flash MCU,” ATMEL Corporation, http://www.atmel.com/images/doc6175.pdf (last accessed 10/24/2015).

[12] “Modbus Application Protocol Specifications V1.1b3,” www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf (last accessed 10/24/2015).

[13] G Kunzel, C. Pereira. “A Tool for Response Time and Schedulability Analysis in Modbus Serial Communications,” IEEE Int. Conf. on Ind. Informatics, 2014, pp. 446-451.