Chapter 6

Conclusion

This chapter details some challenges that I faced while meeting my goals on the project and an overall summary of my contributions towards the IVDS project.

6.1 Repeater Unit software challenges

During software development on the WMI™, I ran into some challenges that I had to bypass in order to achieve a few of my goals. I detail these challenges in a chronological sequence since I joined the project.

My first goal was to change the method in which messages were displayed on the handheld controller unit. Initially the WMI™ handheld controller, consisting of a user keypad and a LCD display screen, was being used as a resourceful debugging tool. Outbound messages were being streamed to this handheld controller. Debugging software permitted user interference through keypad initiated commands. I found this as adding to the software overhead since the handheld controller was part of the peripheral polling sequence. There was also a limit on the size of the character string that could be sent to this controller. In order to send a text string to the controller, it was necessary to wait until there was three times the text string length space available in the 6805 buffer. This made the debug output look "unclean" and, in some cases, unreadable. I bypassed this by streaming the outbound messages to a dumb PC sitting with an open "Terminal" window open.

Implementing a software interrupt turned out to be more challenging than I anticipated. Having absorbed the necessary knowledge from the Motorola data book [9], I set about searching the kernel for existing interrupt implementations. I first created a test interrupt routine that could be entered and exited without performing any tasks. When this didn't work, I received useful advice from Robert Brickhouse, the author of the WMI™ kernel. He suggested freeing up the timer that was dedicated to processing one of the serial ports. This timer could then be used as an interrupt generator. Freeing up the timer meant that the serial port it serviced could not communicate to the PC at 57.6Kbaud but at any other lower baud. I did not see this as an issue since this only meant that the WMI™ monitoring software would run slower. Since the monitor software is only used during software development and debugging, this does not hamper system performance. However, this restricts this particular serial port (PORTA) to be only used at a maximum baud of 38.4Kbaud for any application. The other serial port (PORTB) can still be used at 57.6Kbaud since it is not tied to the timer on the 68306 microprocessor. I freed up the timer and found out that the interrupt service routine I had written was still not being entered. I ended up using the timer as a down counter where an adjustable pre-loaded count value determines the period of the counter-generated interrupt.

I made use of the Real Time Clock (RTC) chip on-board the WMI™ to time-stamp incoming messages. In order to use this, I needed to initially issue a one-time command to set the time to the local time. The software application that read the real system time was getting accurate data from the RTC chip initially. During late hours in the lab, occasionally, I discovered that the RTC chip would return erroneous system time values. This proves to be extremely critical to the message processing routines which are very time dependent. In particular, values of "04:44:44" or "00:00:00" would be received from the RTC chip in the HH: MM: SS format. I noticed this would happen particularly after working on the WMI™ for extended periods of time. I by-passed this by testing for these specifically returned erroneous values and accepting any other values except these. I quickly realized that this would not help when, in fact, the true system time was 4:44:44. I then tried to read the RTC multiple times, average the time readouts and observe the variance with respect to the observed samples. This worked very well and the problem did not occur again. During later stages, I commented this portion of the code and noticed that the problem did not occur anymore. This software strategy could be reused if the problem resurfaces again. This is only an interim solution. For a better approach, I recommend contacting Robert Brickhouse to see if he encountered similar problems before, looking at possible errata sheets for the RTC manufacturer or contacting the RTC manufacturer. The only problem I see in implementing the "averaging" code is excess latency in reading an external device and performing some math.

While implementing the GPS code, my set up was with the GPS receiver closer to a window so I could receive good GPS data. It took me a few days to figure out that I had the connector from the receiver to the WMI™ in backwards and that the code sample was good. This connector is not polarized and may cause confusion. Having tested the code near the "window" environment, I moved back into the CWT where I could no longer test the code. When the same code was tested out in an IVDS demonstration, I learned that the GPS code was giving out incorrect data. I took the same exact code and tried it again the near the same "window" environment where I developed it and it again worked like a charm. I believe the reason for this to be due to the nature of the received GPS data. Since it is the nature of GPS data to be in an uncompressed or compressed format (data fields are omitted due to lack of variance over time), my software was extracting the geographical information from compressed sentences when there was no such information being transmitted. I modified my software to adjust to the compressed format as well and tested it successfully in a few places.

As the code complexity increased, unexplainable latencies cropped up in the WMI™ system during debugging stages. The WMI™ took longer to initialize and sometimes ended up in the wrong code locations and started crashing more often during debugging. This happened only when I dealt with external devices like the handheld display unit, serial ports, ICs external to the 68306 etc. The temporary "fix" I came up with was to introduce dummy wait states before I read any data or addressed any external hardware on the platform. This eliminated the latencies and decreased the code crashes during debugging. When I burned the

EEPROM devices, I commented out these "stabilization" delay strategies and no problems were encountered.

Since PPP was new to me, it took me a while to absorb the guts of the protocol which only led to other sub-protocols. I started out my PPP implementation as a stand-alone software application without involving any message handling software. My initial set up consisted of the PC-MODEM -ISP configuration. I wrote application software that would dial the ISP using the AT command set for the modem using one of the PC COM ports. Having done this, I noticed that the peer (Internet Service Provider, in this case) sent me a "Login/Password" prompt. I then built more software functions that sent out the Login/Password information. Having done this, I started receiving LCP packets from the peer in the HDLC framing format detailed in Chapter 4. I next developed another layer of software that was an implementation of the Password Authentication Protocol, having studied the Protocol field in the incoming LCP packets. Once I got this far in the PPP implementation procedure, I transported the bulk of the software on to the WMI™-MODEM-ISP environment. Though I was able to dial into the modem pool, I was not receiving the "Login/Password" prompt from the ISP. Observing the display LEDs on the modem, I noticed that no data was being transmitted from the modem to the WMI™. After a certain time out period, the connection to the ISP was reset. The only reasoning I could think of at the time was that there was a difference in the serial port configuration on the WMI™ setup versus the PC setup. It is pointed out in one of the RFC references [2] that on the data link layer the modem communication signals DTR, DCD, RTS, CTS are **not** needed. It does not state, however, whether these signals are needed on the physical layer to let the modems interact with their DTEs. This could be further investigated by probing the voltage levels on the input to the modem after the WMI™ dials into the modem pool. This might give a clue as to where the real problem lies - in the ISP to modem path or the modem to WMI™ path. Another suggestion I think might help any future expansions of the project is that there is an existing PCM-CIA memory card slot that is not being used currently. It is possible that this can be used for the extra RAM that would be needed to implement an Internet based protocol. Also there is a memory expansion slot on the WMI™ that is unused. Placing external ROM on this slot can be investigated to provide a larger application software space.

It was during my development of the PPP on the WMI™ that the IVDS project lost financial support. At that point in time, the fundamental concept of the IVDS system had been proven in concept, several system components had been researched, implemented and modified as per the project needs, and along the way the implemented components had been successfully tested individually and integrally.

6.2 Conclusions

During the course of my work on the IVDS project, my primary accomplishments involved the modification and expansion of the existing repeater

unit software. The following list summarizes my key contributions to this project and the state of the software when I left the IVDS project:

- The message structure was modified to make the software algorithms easier for the user control subsystem.

- The throughput between the decoder board and the motherboard was improved by using the bit-stuffing algorithm.

- The buffer structure for the decoder boards was changed to a static buffer that would hold 10,000 IVDS messages during peak incoming message time.

- Incoming messages were time-stamped upon reception.

- The queue structure was modified for Priority7 messages. Duplicates of these messages were retransmitted as well.

- A software interrupt had been implemented to speed up the system response to incoming message traffic.

- GPS software was written to receive NMEA format messages and extract latitude and longitude information.

- A new retransmission algorithm was written that gave more fairness to the priority scheme.

- The Password Authentication Protocol software was written for the WMI™ to connect and communicate to the ISP.