David Mraz, Complete Automation Controls, LLC

Low Latency Parsing of Network Data Using Control Logix PLCs

Abstract:

With a simple software trick, PLCs can be used as low-latency signal conversion devices. This paper shares the experience gained while developing a product that parses dual-homed, 500-byte UDP multicast messages and outputs an analog signal based on that data within 8ms. The project used a Rockwell RSLogix5000 with an L7x series controller.

Many challenges were overcome during development. The first design issue stems from the inability of Rockwell Ethernet module to produce hardware interrupts, forcing the controller to poll the Ethernet module to learn when new data arrives. To resolve this, the project utilizes a periodic task with the sole function of monitoring the receive buffers on the Ethernet modules to determine when new messages are received. The periodic task is uninterruptable by all other tasks, and has the smallest period possible (ideally, 1ms or less) while allowing other tasks to function properly.

The second design issue arises from the algorithm the PLC uses to parse the received messages. Parsing is an expensive process for the PLC controller and it cannot afford to parse both messages entirely. Instead, it is designed to partially parse the network message upon reception. Only enough information is parsed to decide whether the primary or secondary interface is most desirable to fully parse. The decision is based on the packet validity and freshness. The fast multicast receive task described in the paragraph above includes this partial parsing. When the best packet is fully parsed, the task calls an event task, the highest priority task in the whole project, to complete the conversion and output process.

To measure the latency of the signal reliably, Precise Time Protocol (IEEE 1588) or some other synchronization scheme must be utilized to synchronize clocks of the PLC and the output device from which the PLC receives the multicast data. The multicast output device must timestamp the data immediately before transmission, while the PLC must timestamp the data immediately after conversion and output. The difference between the timestamps is used to calculate the full system latency. Trending the latency provides a graphical depiction of system performance over time. With this new system in place, the latency period will be greatly decreased.

Introduction:

> **"***The socket interface, via messaging, is not well suited for real-time control as communication with this method is not scheduled or deterministic."*
>
> -From Performance Considerations section of Rockwell's *EtherNet/IP Socket Interface* manual.

This paper explores the limits of that quote by discussing a project that attempted to prove it wrong. That project aimed to convert digital data from a multicast message to a custom analog signal with low latency. The system was developed to address the U.S. Navy's need for an in-house alternative to an existing third-party conversion device. It was a research project that determined if a PLC-based system would be a viable option. A custom embedded solution or the VxWorks platform were the other platform alternatives considered. The Navy preferred using PLCs, however, due to their track record of high reliability while deployed in the fleet. The biggest design challenge in the development of a PLC-based replacement arose while meeting the strict 8ms maximum latency period between input to and output from the PLC that was required.

The system uses input data in the form of dual-homed UDP multicast messages. The existing PLC code, on which this project is based, reads multicast messages using periodic tasks. These tasks are configured with a period of one half the expected period between incoming messages. This approach has worked well in deployed systems and ensures that the PLC ignores no messages while always using the most current data. For this project, however, the PLC is not allowed an entire period to read, convert and output data – it is allotted only 8ms to accomplish those functions. The baseline project is not designed for this real-time specificity and cannot meet it.

Because the baseline approach is not fast enough to cope with the new latency requirement, a new method was devised. The new approach keeps the periodic task that receives incoming multicast messages, but removes as much other functionality from the task as possible. The functionality necessary to parse, convert and output data moves to an event task. This event is triggered by a multicast receive task that is called when a new message is received. This method reduces the amount of time a message waits in the multicast receive buffer before the PLC controller acknowledges it.

The redundant networking associated with the baseline PLC system parses the same data twice when both networks are active, leading to long function execution times. To reduce the execution time, the new system needs the ability to determine when a message is a duplicate from the other network line and ignore it. To accomplish this, PLC code selectively parses only the data needed to determine the better multicast message of the two received, then fully parse the chosen message. In this case the most recent message is considered the better message. The project architecture, task properties, monitoring statistics and code are detailed throughout this paper to demonstrate the results of the new project compared to the baseline system.

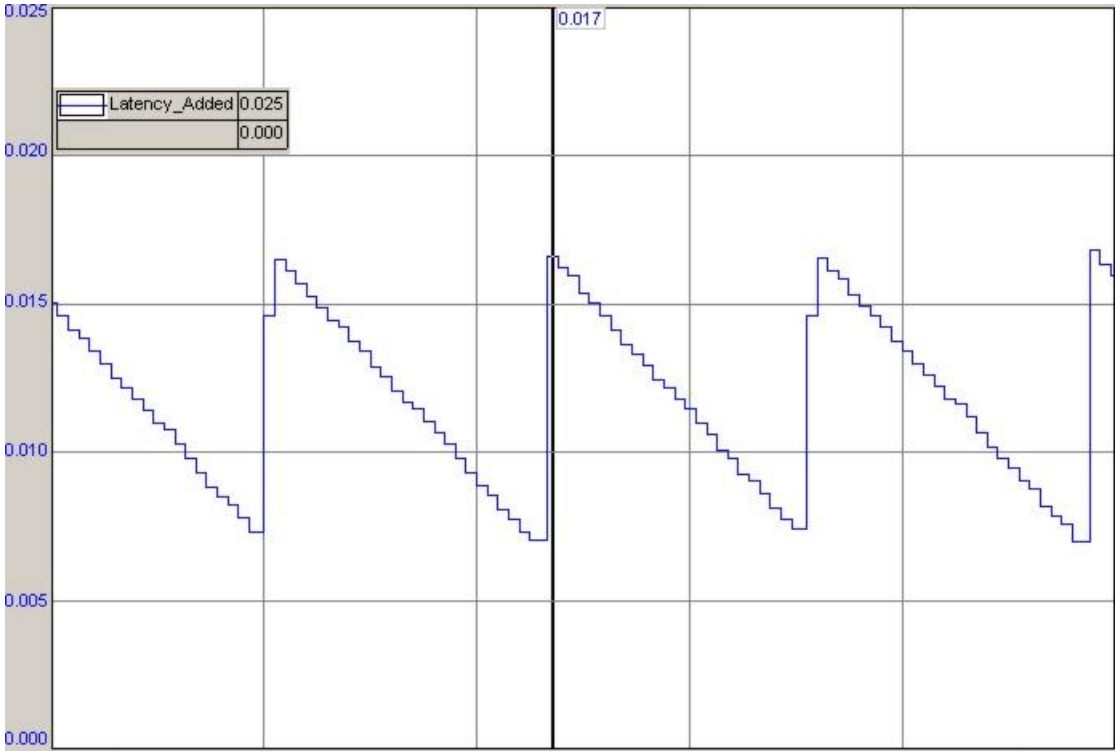


Figure 1: Latency for Baseline System Over Time (in seconds)

Full versus Partial Parsing

As shown in Figure 1, the baseline system does not meet the latency requirements, clocking a maximum latency of 17ms. Its minimum latency of 8ms, however, indicates that meeting the new requirement is possible. Two main issues stand out with the baseline implementation. First, the PLC fully parses both the primary and secondary multicast message interfaces before deciding which one contains the most recent data. Parsing is an expensive operation that hinders speed and increases latency. For this reason, it is faster to partially parse both the primary and secondary messages to decide which is better to send to the final buffer for execution, rather than fully parsing each one to make that determination. This concept is illustrated in the flowcharts shown in Figures 2 through 4.

The baseline system, represented in Figure 2, shows that both messages must be fully parsed before a determination of which message should be executed can be made. This more complex process results in a Max Scan Time of 5030us. The new project, represented in Figure 3, shows that the PLC first parses the portion of each message that shows if it is valid and how recent it is, then based on that data fully parses the better message. This more efficient approach results in a Max Scan Time of 2582us, which is nearly twice as fast as the original method.

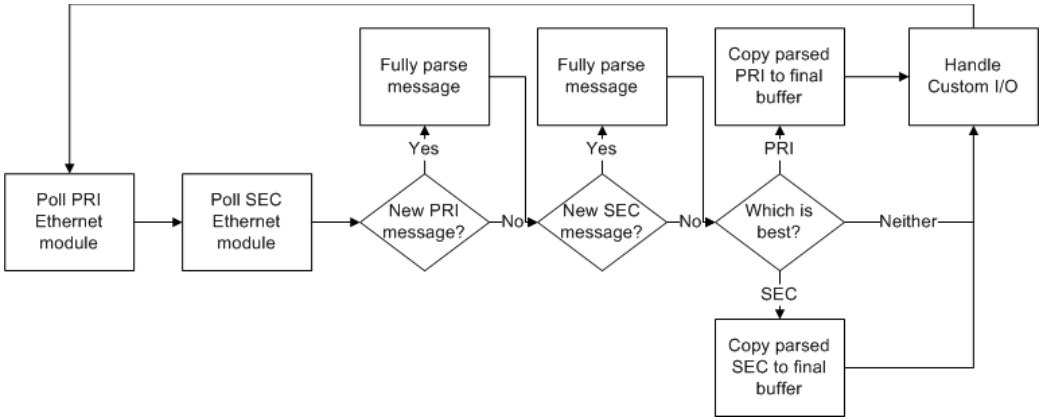


Figure 2: Flowchart for Baseline Parsing Algorithm

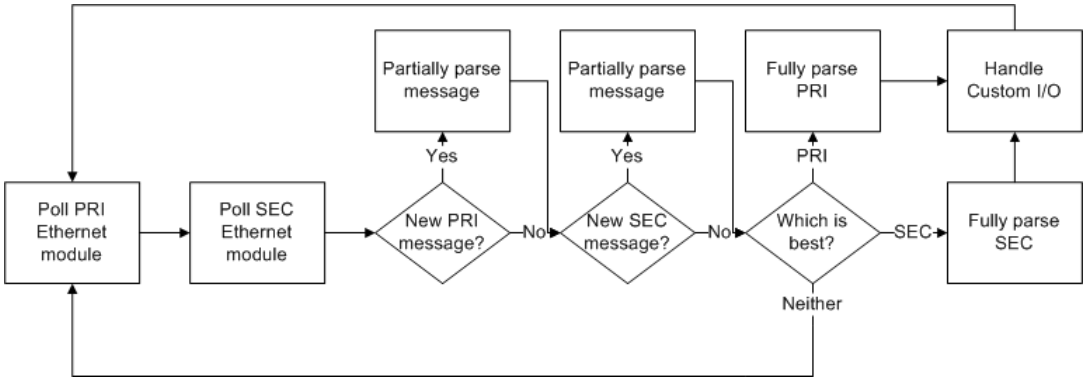


Figure 3: Flowchart for New Parsing Algorithm

Program List for: MSG_MCAST_Read				
	Program Name	**Last Scan**	**Max Scan**	**Program State**
Baseline System	MCAST_Reader	4728 us	5030 us	5030 us
New System	MCAST_Reader	58 us	2582 us	Enabled

Figure 4: Execution Times for Baseline and New Parsing Algorithms

Determination Process

The partial parsing determination capability is made possible in the new system by reading a field in the incoming multicast message that serializes the message. The field is called the unique message ID number field, "UNIQUE_ID_NUM," and is associated with each multicast message that is input into the system. The PLC uses this information to decide which message is better. Because both messages originate from the same source, comparing one unique message ID to the other, in addition to using a watchdog timer to ensure that each interface is active, proves sufficient for the PLC to determine which message is more recent.

A new program flow was implemented to perform the determination process, the ladder logic for which can be found in Appendix A. The important aspect of deciding which received message is better, is abstracted out of the ladder into a subroutine called "Choose_PRI_or_SEC". The decision is based on four input parameters (message ID and interface activity for both messages). The "IsValid" parameter informs the program if a message interface has timed out. The "UNIQUE_ID_NUM" parameter is a message counter that informs the program which message is more recent assuming both interfaces are active. Using these four parameters, the better interface decision is based on the following logic:

1. If one interface is invalid, then use the other one;
2. If one interface is newer, then use the newer one;
3. If both interfaces are the same, then use the primary one.

Latency Evaluation

By utilizing partial parsing, latency decreases by approximately 2.5ms, as seen in Figure 4, but the maximum latency remains at approximately 14.5ms, which does not meet the requirements. The most significant cause of this latency is the 10ms time period used to perform the multicast read task, "MSG_MCAST_Read." While this task does guarantee that every message received at 50Hz will be parsed by the PLC, it can add up to 10ms of latency before the PLC acknowledges that there is a message in the Ethernet modules' receive buffers. Ideally, an Ethernet module would create an interrupt task upon receipt of a message; in reality, though, the only way for the controller to communicate with the Ethernet module is by periodically polling it using CIP messages. Thus reducing the period of the multicast receive task is the only way to reduce latency. Doing this, however, may, interfere with task scheduling. Because multicast parsing has a real-time requirement, the parsing task is a high priority task, and it can cause other lower priority tasks to overlap if the period is reduced too drastically.

Synchronization cannot be assumed between the firing of the receive task and the receipt of the multicast message. The task monitor for the multicast read task shown in Figure 5 clearly indicates that the interval times between task firings are subject to an almost +/- 0.2ms jitter. This is another issue the new system needed to address since jitter alone accounted for up to 10ms of latency in the baseline project. It would also be unwise to assume that the multicast is reaching the PLC at an exact 50Hz frequency, which could mean that some messages may not be received properly. The concept is illustrated in the time-line in Figure 6, which shows that a late multicast message can lead to a large latency and hinder the system's ability to meet requirements.

Scan Times (Elapsed Time):	
Max:	6.916000 ms
Min:	1.320000 ms
Interval Times (Elapses Time Between Triggers)	
Max:	10.167000 ms
Min:	9.834000 ms
Task Overlap Count:	
0	

Figure 5: Monitoring the Baseline MCAST Read

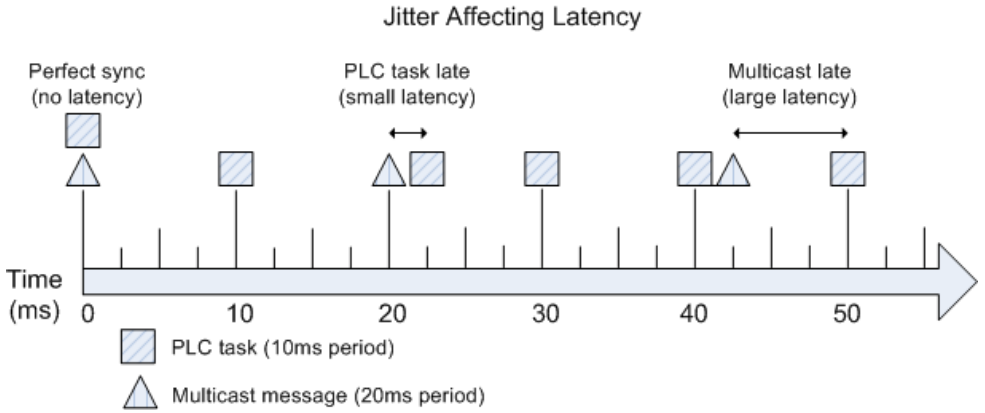


Figure 6: The Effects of Jitter on Latency

Mitigating Latency

Comparing the task monitor in Figure 5 to the latency graph in Figure 1 shows that adding the scan time and the interval time will equal the maximum latency time (7ms + 10ms = 17ms). Based on the time-line above in Figure 6, it can be determined that the maximum latency occurs when the multicast read task executes immediately before the packet arrives. When this happens, it takes approximately 10ms for the PLC program to acknowledge that the Ethernet module's read buffer has received the message. This source of latency must be mitigated by ensuring the multicast read task is executed as often as possible.

There are two available solutions to this problem: either make the multicast read task a continuous task or reduce the period of a periodic task. The first option, using a continuous task, is not ideal because that would make the multicast read task the lowest priority task when it needs to remain one of the highest. This must be avoided because it would cause the timing of other periodic tasks (which, by definition, are all higher priority) to block the continuous task for a significant amount of time. Since reading the multicast message and converting it to a custom output is the process that has the strictest real-time requirement, it should not be dependent upon any other task.

The second option, dividing the reading and converting programs into two separate tasks, would be more effective. The basic configuration of this method is seen in the Controller Organizer image shown in Figure 7. The multicast read task, "MSG_MCAST_Read," polls the Ethernet module as quickly as possible by running the ladder logic shown in Appendix A as its main routine. When it determines a new better message, it fully parses that message and calls the event task, "Custom_IO," that executes the conversion and output functionality. Only executing the event task when there is a new message allows the system to function more efficiently by only running tasks when necessary.

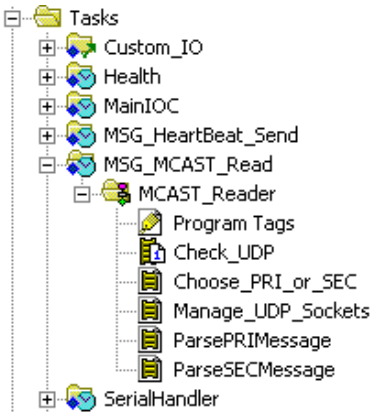


Figure 7: Controller Organizer for New Configuration

Configuration	
Type	Periodic
Period	0.750 ms
Priority	2 (Lower Number Yields Higher Priority)
Watchdog	500.000 ms
☒	Disable Automatic Output Processing to Reduce Task Overhead
☐	Inhibit Task

Figure 8: MSG_MCAST_Read New Configuration Properties

To enable this functionality, the event task, "Custom_IO," must be the highest priority task in the project and the second highest priority task must be the periodic task, "MSG_MCAST_Reader." To minimize latency, the "MSG_MCAST_Reader" task's period should be set to the smallest amount of time that permits all tasks in the project to run properly while also avoiding high latency spikes. This period value must be determined empirically by trending the latency while slowly reducing the period of the "MSG_MCAST_Reader" task. It is important to note that if the periodic task's period is reduced too much, then either the other tasks in the project will begin to overlap, or the overhead associated with task switching will cause latency spikes. The task configuration for the "MSG_MCAST_Read" task used to accomplish this is shown in Figure 8.

By making the "Custom_IO" event task the highest priority task, the program ensures that as soon as the "MSG_MCAST_Reader" periodic task calls the event task, the data conversion and output will occur without interruption. As a side effect, the PLC's task scheduler will attempt to call the periodic task several times while the event task executes causing the Task Overlap Counter to increase for the "MSG_MCAST_Read" task. Creating a new task for the "Custom_IO" program instead of making it a series of subroutines in the "MSG_MCAST_Read" task minimizes the amount of code that sits in a task that accumulates overlaps. The less code that accumulates in a task, the more efficiently the system runs.

Calculating Latency

To calculate the latency time associated with the new system, a number of factors must be taken into account. Simply adding the maximum scan time and interval times will not equal the latency in this configuration because the interval times include time when the task is overlapping. During overlapping parsing, conversion and output work is being performed; the PLC controller does not wait to poll the input buffer. To measure the actual latency time more accurately, the max scan times for the "MSG_MCAST_Read" task and the "Custom_IO" task and the min interval time for the "MSG_MCAST_Read" task must be added (4.3ms + 1.6ms + 0.6ms = 6.5ms).

The graph below shows the latency measured live during the project's execution. The result adheres very closely to the calculation above, displaying a maximum latency of 6ms.

Scan Times (Elapsed Time):	
Max:	4.279000 ms
Min:	0.104000 ms
Interval Times (Elapses Time Between Triggers)	
Max:	4.636000 ms
Min:	0.575000 ms
Task Overlap Count:	
2398	

Figure 9: Monitoring the New MCAST_Read Task

Scan Times (Elapsed Time):	
Max:	1.587000 ms
Min:	1.452000 ms
Interval Times (Elapses Time Between Triggers)	
Max:	22.447000 ms
Min:	18.709000 ms
Task Overlap Count:	
0	

Figure 10: Monitoring the Custom_IO Task

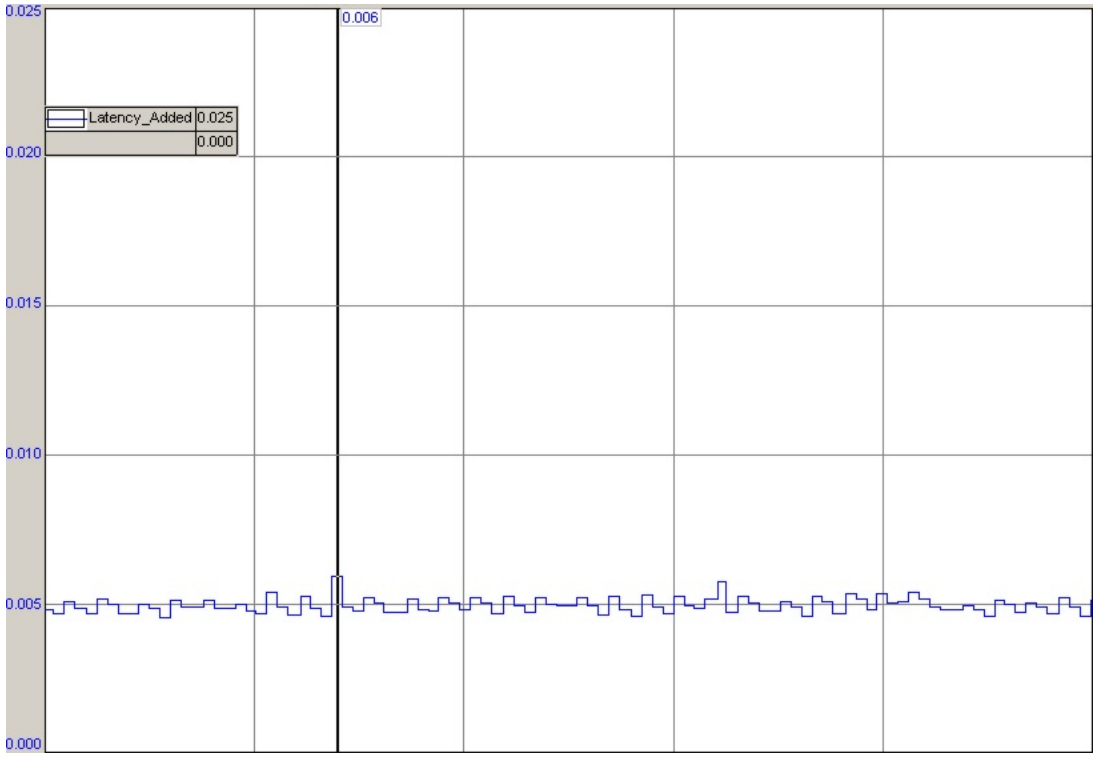


Figure 11: Latency for New System Over Time (in seconds)

CONCLUSION:

Allen Bradley PLCs can prove suitable for real-time conversion applications when reading multicast packets if a project architecture that prioritizes polling the Ethernet module is enabled. A lean periodic task that has the smallest period possible to check the receive buffer and an event task that executes upon receiving new data are the two software features that make low-latency conversion in a PLC possible. With this architecture in place, data processing time is the most significant factor in controlling the latency of the system. Data processing can vary greatly based on packet size, packet format, conversion algorithm and other factors. The multicast packet in this project is approximately 500 bytes.

To reliably measure the latency that the PLC project adds to an overarching system, a time-syncing and time-stamping scheme must be implemented. This project used a Precise Time Protocol (IEEE 1588) CIP-Sync to synchronize the clocks on the multicast transmitting device with the PLC controller clock. A similar setup is recommended for any party developing a system that must measure latency. More information on this subject can be found in Rockwell's Precise Time Protocol manual (Reference 3).

REFERENCES:

1. Allen-Bradley, *EtherNet/IP Network Configuration* (enet-um001)

2. Allen-Bradley, *EtherNet/IP Socket Interface* (enet-at002)

3. Allen-Bradley, *Integrated Architecture and CIP Sync Configuration* (ia-at003)

4. IEEE 1588

ACKNOWLEDGEMENTS:

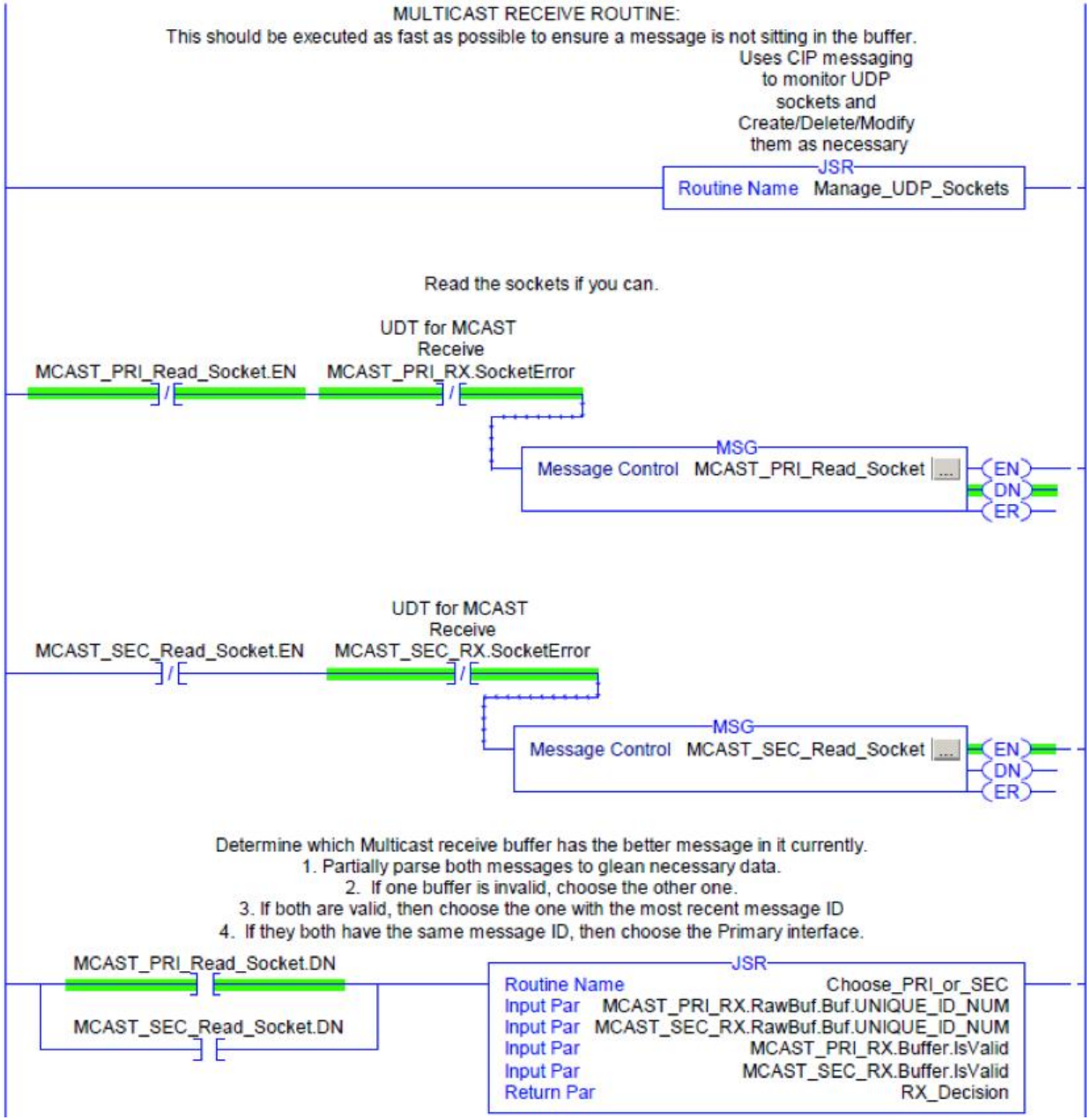
Charles Hermann invented the custom I/O hardware that the PLC interfaces with in this project. He also created the first prototype of this project that interfaced with that hardware.

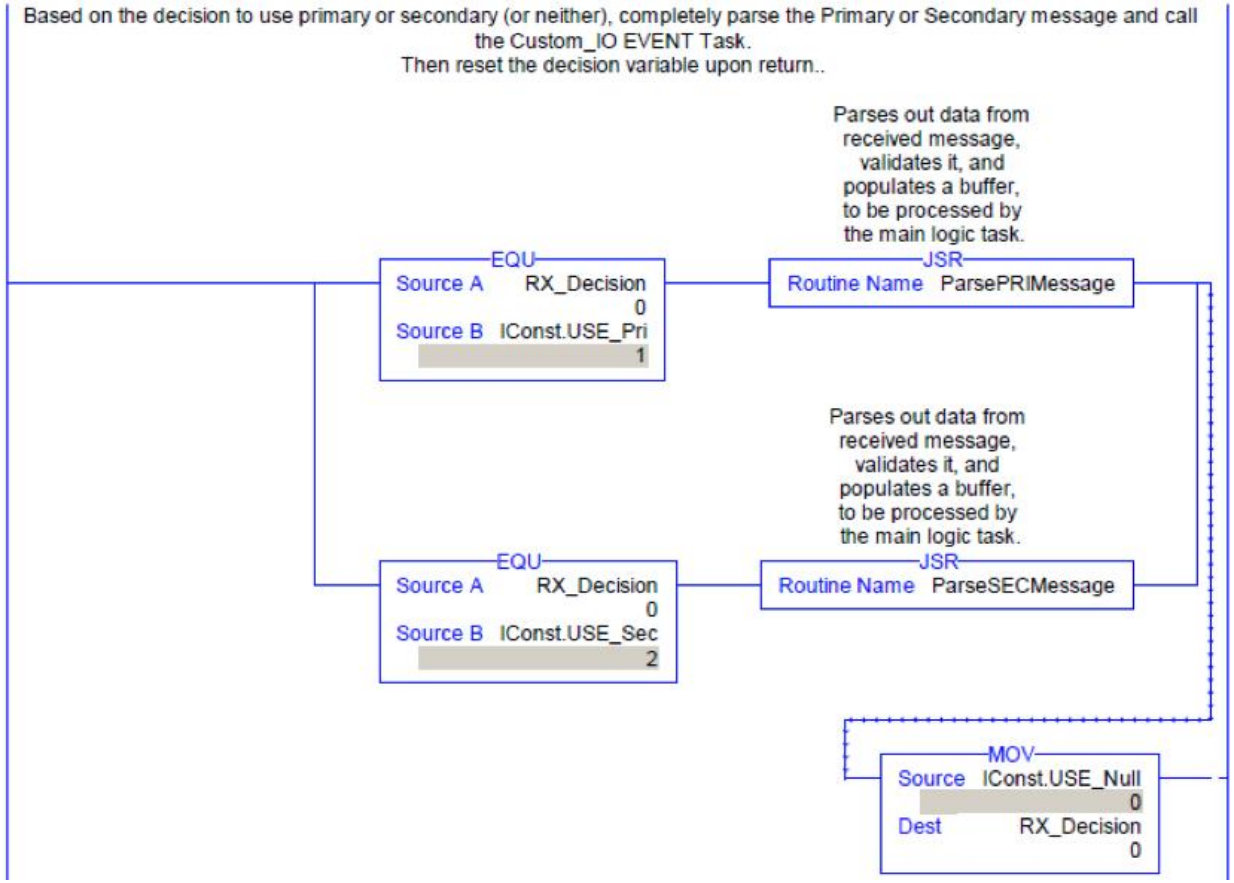
Bartlomiej Kleczynski created the PLC framework including multicast reading and parsing routines that were used as the baseline in this project.

BIO:

David Mraz is an electrical engineer for Complete Automation Controls, LLC. specializing in embedded electronics and PLC control systems. His current main contract is working with the Navy for NSWCCD-SSES Code 967, Steering, Propulsion, and Navigation Systems.

Appendix


MULTICAST RECEIVE ROUTINE:
This should be executed as fast as possible to ensure a message is not sitting in the buffer.
Uses CIP messaging
to monitor UDP
sockets and
Create/Delete/Modify
them as necessary
JSR
Routine Name Manage_UDP_Sockets
Read the sockets if you can.
UDT for MCAST
Receive
MCAST_PRI_Read_Socket.EN
MCAST_PRI_RX.SocketError
MSG
Message Control MCAST_PRI_Read_Socket
EN
DN
ER
UDT for MCAST
Receive
MCAST_SEC_Read_Socket.EN
MCAST_SEC_RX.SocketError
MSG
Message Control MCAST_SEC_Read_Socket
EN
DN
ER
Determine which Multicast receive buffer has the better message in it currently.
1. Partially parse both messages to glean necessary data.
2. If one buffer is invalid, choose the other one.
3. If both are valid, then choose the one with the most recent message ID
4. If they both have the same message ID, then choose the Primary interface.
MCAST_PRI_Read_Socket.DN
MCAST_SEC_Read_Socket.DN
JSR
Routine Name Choose_PRI_or_SEC
Input Par MCAST_PRI_RX.RawBuf.Buf.UNIQUE_ID_NUM
Input Par MCAST_SEC_RX.RawBuf.Buf.UNIQUE_ID_NUM
Input Par MCAST_PRI_RX.Buffer.IsValid
Input Par MCAST_SEC_RX.Buffer.IsValid
Return Par RX_Decision



Appendix A: Ladder Logic for New Parsing Algorithm