

# CIP Motion Implementation Considerations

Mark Chaffee  
Senior Principal Engineer  
Rockwell Automation

Presented at the ODVA  
2009 CIP Networks Conference & 13<sup>th</sup> Annual Meeting  
February 25, 2009  
Howey-in-the-Hills, Florida USA

## Abstract:

This paper will present information helpful to those considering implementation of CIP Motion in their future controller or drive products, including techniques to optimize system performance, tools to facilitate implementation, and methods that allow for the full flexibility of the CIP Motion profile from simple VFDs to sophisticated vector servo drives.

System performance considerations will be discussed with attention paid to performance of the various subsystem components. CIP Motion offers an extremely flexible connection data structure that supports dynamic change of data content between the drive and the controller. Learn the implementation details that allow the user to take full advantage of this capability. A number of potential components have been identified to augment CIP Motion ranging from 1588 assist ASICs to WireShark add-on for decoding CIP Motion packets. These components shall be discussed in further detail in the paper.

## Keywords:

CIP Motion  
Drive Profile  
Motion Control

## Definition of terms (optional):

C2D Connection = Control to Device Connection  
D2C Connection = Device to Control Connection  
NIC = Network Interface Component

## Body:

Over the last several years, Rockwell Automation has been developing CIP Motion drive products that integrate with its Logix control system. The project targeted both servo drive products and variable frequency drive products. The initial design work for the project was based on two key CIP Motion specifications, the Motion Axis Object and the CIP Motion Drive Device Profile, that were released in the spring of 2006 as part of the CIP Specification (Volume 1). Much was learned during product development. Opportunities were identified to utilize the full power of the dynamic CIP Motion connection to improve system performance. Tools were developed to provide valuable diagnostics. The purpose of this paper is to share our experience with the CIP community with an eye towards facilitating wide adoption of the CIP Motion standard.

## EtherNet/IP Control System Architecture

It is important to have a general picture of EtherNet/IP control system architecture to appreciate the importance of prioritizing EtherNet/IP connection data as that data passes through the control system components. The system

diagram below illustrates three general classifications of CIP data: Motion, I/O, and Messaging. System performance requirements dictate that this data be identified as High, Medium, and Low priority, respectively. Should system components be subjected to a mix of Motion, I/O, and Messaging data, these components must prioritize data processing to maintain deterministic data delivery. This prioritized data processing strategy, as it applies to each of the control system components, is referred to as End-to-End QoS (Quality of Service).

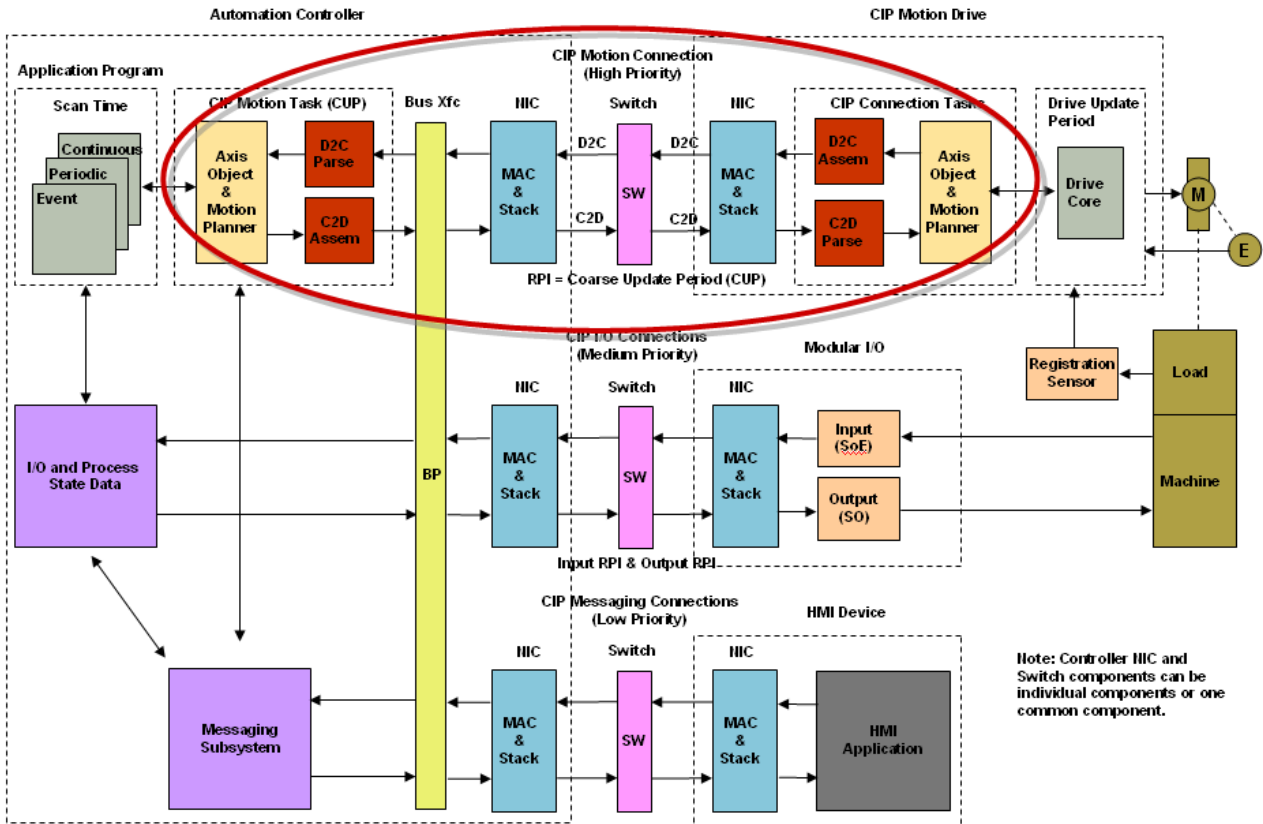


Figure 1 – EtherNet/IP Control System Architecture

### CIP Motion System Components

Our focus will be on the components within the red ellipse that are specifically associated with the CIP Motion system. On the left we have an automation controller that supports CIP Motion compliant drives. The controller executes a high priority, periodic, CIP Motion Task that begins by receiving fresh CIP Motion data from each drive associated with the D2C connection. The Task then performs motion planner calculations to determine the command reference value for each drive axis. Finally, the task assembles the C2D connection data block, including the command reference, and initiates transfer of the data block to the Network Interface Component (NIC) via some form of bus interface. The bus interface mechanism could be DMA, PCI, USB, etc. The NIC runs an EtherNet/IP stack that encapsulates the C2D connection payload in a standard Ethernet UDP packet and transmits the packet to the targeted CIP Motion drive. The NIC typically consists of a CPU to run the stack, a MAC to manage the Ethernet media and a PHY to provide the hardware interface. The CPU to run the stack can be the same as the CPU that runs the controller tasks, or can be a separate communications processor. The MAC can also be a separate device, or as is commonly the case, integrated into the same CPU that runs the stack.

After the NIC transmits the C2D packet on the Ethernet wire, the packet passes through one or more Ethernet switches on its way to the targeted CIP Motion drive. These switches can be stand-alone, managed or unmanaged, switches or they can be 3-port switches integrated into the drive product. Embedding 3-port switches in the drive is particularly attractive in that it eliminates the cost of a separate switch and reduces cabling cost by allowing line or ring network topology.

The packet is received by the drive's NIC, which removes the Ethernet Encapsulation and delivers the CIP Motion C2D payload to the drive's application layer for processing. CIP Motion data processing is typically performed by an interrupt driven CIP Connection Task. The CIP Connection Task processes the C2D connection data, updates various axis object attributes, and establishes a new command reference target for the drive core to control the motion of the associated motor and load.

Shortly before the start of the next update cycle, the CIP Connection Task assembles the latest data from the axis object into a D2C connection data block and initiates transfer of the data block to the NIC via the bus interface. The NIC runs an Ethernet/IP stack that encapsulates the D2C connection payload in a standard Ethernet UDP packet and transmits the packet to the targeted controller. The NIC components on the drive side of the Ethernet interface are the same as those on the controller side. But the communication demands on the controller side far exceed those on the drive side. For this reason it is common for the drive CPU to also include support for the EtherNet/IP stack and an integrated MAC.

Following the same path as the preceding C2D packet, the D2C packet passes back through the Ethernet switches on its way to the targeted CIP Motion controller. The controller NIC receives the D2C packet, strips off the Ethernet encapsulation and initiates transfer of the D2C payload to the controller's memory via the bus interface, thus completing the connection cycle.

Each of these components plays an important role in the overall performance of the CIP Motion system.

### **Performance Considerations**

The Distributed Motion jSIG is currently reviewing a number of proposed changes to the original CIP Motion specification. A significant percentage of those changes were introduced to optimize CIP Motion system performance. To fully appreciate the importance of these changes, a metric of system performance is needed. The motion control industry has generally adopted the "axis per msec" metric as an indicator of motion control system performance. This metric represents the number of axes the system can control for a given connection update period, or cycle time. For a motion control system, the axis per msec performance level is determined by the constraints of the timing model.

The CIP Motion 1-Cycle Timing Model shown in diagram below defines the exchange of data between the motion controller and the drive device. This timing model breaks the update cycle into three equal parts, the input interval, the planner interval, and the output interval. The timing is quite simple:

1. Input data must arrive at the controller before the end of input interval.
2. Planner generated output should be transmitted before the end of the planner interval.
3. Output data must arrive at the drive before the end of the output interval.

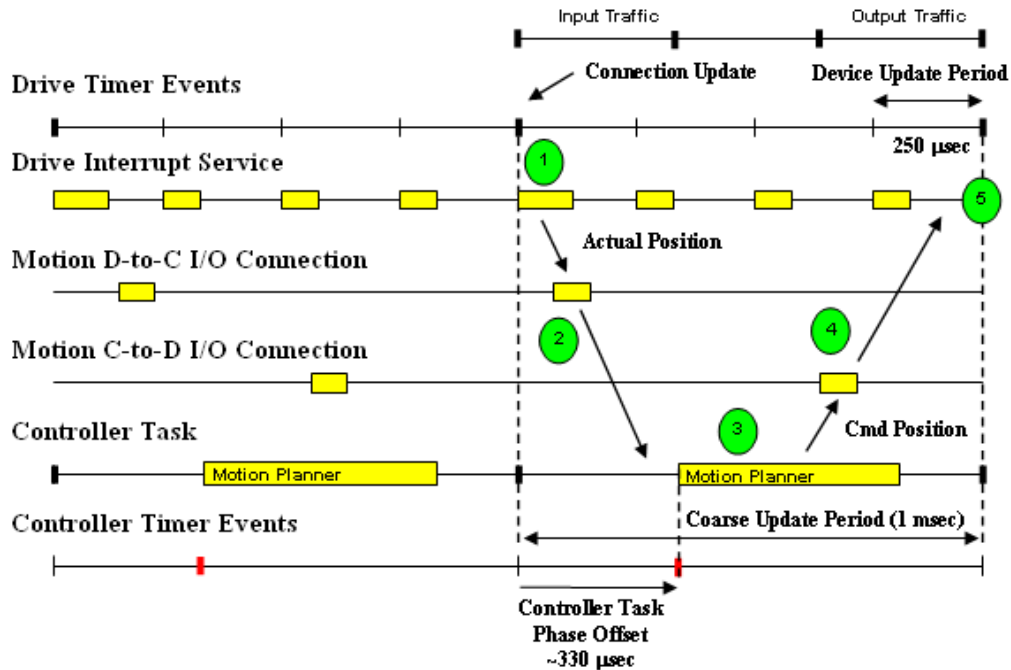


Figure 2 – CIP Motion 1-Cycle Timing Model

By analyzing the data processing and transmission delays through the control system and modeling these through the three intervals of this timing diagram, an estimate of the “axis per msec” performance of the CIP Motion system could be obtained. Through this analysis it became apparent that the input and output communication performance as well as the motion planner performance would have to be optimized to meet system performance requirements.

Optimizing the controller’s NIC stack cut CIP Motion packet processing time in half over initial implementation. Optimization efforts on the drive’s NIC stack were even more rewarding, reducing the packet processing delay by a factor of 10! Stack performance varied considerably based on the specific Ethernet/TCP/UDP/IP stack and CPU selected. With good choices in this regard, high priority EtherNet/IP packets can be processed at wire-speed.

Unlike the NIC stack, optimizing the task of parsing and assembling CIP Motion connection data processing required changes to the CIP Motion specification. While the original specification did well in defining a flexible connection data structure, it did not go far enough to take advantage of that flexibility. The desire to expand the amount of real-time diagnostic information, exchanged between the drive and the controller, resulted in large CIP payloads containing relatively static data.

Connection Header			
Connection Format	Format Revision	Update ID	Node Status
Instance Count	Node Faults/Alarms	Last Received ID	Time Data Set
Device Update Period (4-bytes)			
Device Time Stamp (8-bytes)			
Device Time Offset (8-bytes)			
Timing Diagnostic Data (20-bytes)			
Instance Data Header			
Instance Num	-	Instance Blk Size	Cyclic Blk Size
Cyc. Act. Blk Size	Cyc. Read Blk Size	Event Blk Size	Service Blk Size
Cyclic Data Block			
Control Mode	Feedback Config	Axis Response	Response Status
-	Actual Data Set	Status Data Set	Axis State
Actual Position (4-bytes)			
Fault Data (12-bytes)			
Alarm Data (12-bytes)			
Status Data (20-bytes)			
Cyclic Read Data Block			
Cyclic Write Blk ID	Cyclic Write Status	Cyclic Read Blk ID	Cyclic Read Status
Event Data Block			
Event Checking Status (4-bytes)			

Figure 3 – Un-Optimized CIP Motion D2C Connection Payload (120-bytes)

The fields highlighted in yellow in the above un-optimized data structure are the only values that tend to change every update cycle. The rest of the data structure elements may not change for thousands of updates, some indeed may never change. With minor enhancements to the specification, we were able to take advantage of this behavior using the flexible CIP Motion connection data structure by only passing data when the data value actually changes. As shown in the optimized data structure below, this resulted in a massive 4-to-1 reduction in the CIP payload size and a 3-to-1 reduction in the total time it takes to parse and assemble the CIP payload data.

← 32-bit Word →			
Connection Header			
Connection Format	Format Revision	Update ID	Node Status
Instance Count	Node Faults/Alarms	Last Received ID	Time Data Set
Device Time Stamp (8-bytes)			
Instance Data Header			
Instance Num	-	Instance Blk Size	Cyclic Blk Size
Cyc. Act. Blk Size	Cyc. Read Blk Size	Event Blk Size	Service Blk Size
Cyclic Data Block			
Control Mode	Feedback Config	Axis Response	Response Status
-	Actual Data Set	Status Data Set	Axis State
Actual Position (4-bytes)			

Figure 4 – Optimized CIP Motion D2C Connection Payload (36-bytes)

## Drive Performance

Drive performance also needed optimization. C2D and D2C packet processing was being constrained by the drive timing to occur in the last 3<sup>rd</sup> of the update cycle. When running fast cycle times, there was not be enough time to process both connections. To address this issue, priority must be given to processing the D2C packet during the last 3<sup>rd</sup> of the cycle to insure that the D2C packets transmit by the start of the next cycle. Once the D2C data is

assembled and ready for transmission, the drive can then focus on processing the C2D packet it just received. C2D data processing can extend well into the first 3<sup>rd</sup> of the next cycle if necessary. Such flexibility is made possible by command reference time stamp.

Another optimization opportunity for the drive comes in the form of a CIP Motion Hardware Assist FPGA that offloads EtherNet/IP stack processing from the host CPU. This device would identify inbound CIP Motion C2D packets and extract the C2D connection data blocks from those packets, making the data available to the host CPU for processing. Conversely, the device would also take assembled D2C connection data from the host CPU, encapsulate and transmit that data as an EtherNet/IP packet. This FPGA could also include Hardware Assist for IEEE-1588 and 3-Port Switch functionality to support popular Line and Ring network topologies. A block diagram for such CIP Motion Hardware Assist FPGA is shown in the figure below.

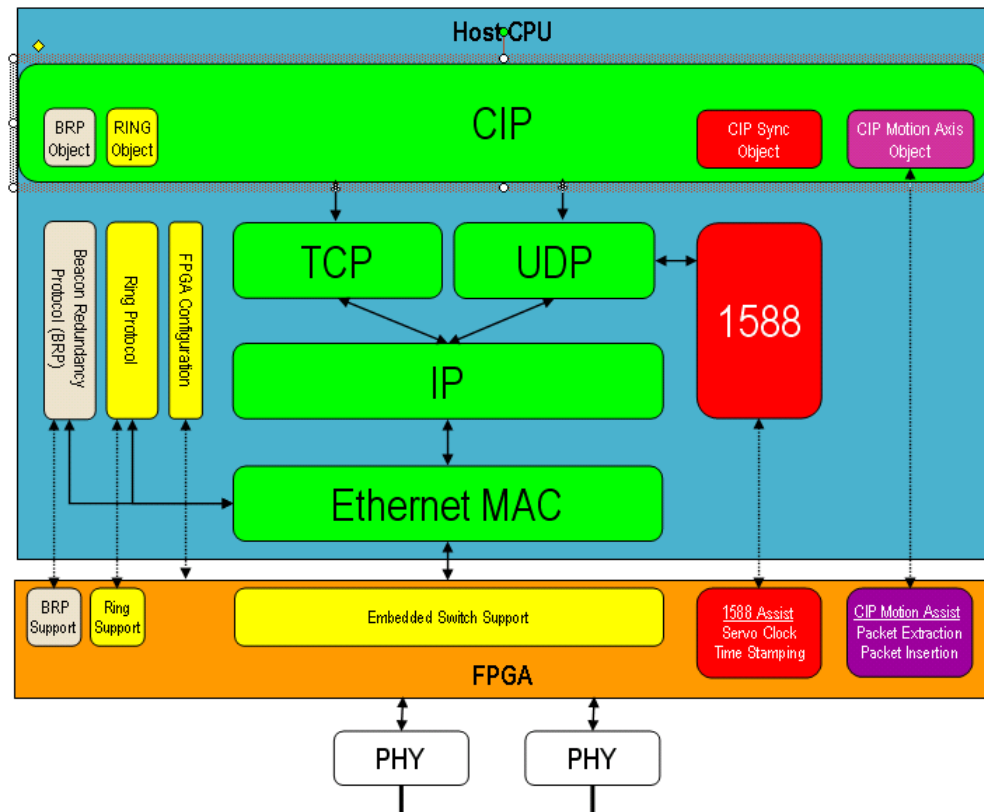


Figure 5 – CIP Motion Hardware Assist FPGA

While not strictly necessary to implement a CIP Motion compliant device, a CIP Motion Hardware Assist FPGA would also facilitate adoption of the CIP Motion interface by drive device vendors. Today's drive architectures typically support multiple motion networks, SERCOS III, PROFINet IRT, EtherCat, etc. To support these motion network interfaces, the drives must include an FPGA or ASIC device (SERCON, ERTEC, FMMU, etc) to handle non-standard Ethernet media access control and process packets. Therefore, since the host processor for these drive products do not need to run a stack to process these packets, the expectation is that some form of external packet processing would be provided for CIP Motion as well. A CIP Motion Hardware Assist FPGA could be developed to allow these vendors to drop the CIP Motion FPGA into their modular network interface architecture and more easily extend their motion network support to include CIP Motion.

### End-to-End QoS

The above performance analysis was based on a private network CIP Motion implementation where there is no non-motion data traffic. This is analogous to our typical SERCOS II interface topology. But unlike SERCOS and other motion network solutions, CIP Motion, and EtherNet/IP in general, allow motion packets to freely mix with non-

motion data packets, e.g. CIP I/O, CIP Messaging, and even non-CIP TCP/IP traffic. Looking back to the EtherNet/IP Control System diagram in Figure 1, this mix of packets may be passing through a common switch, a common NIC, a common bus interface, and even target a common controller or drive end device. To provide the determinism and timely data transfer necessary for motion control, each component in the control system must prioritize data processing. Within the Ethernet infrastructure this requirement is met by components that support Ethernet standard QoS (Quality of Service) protocols.

There are two standard Ethernet QoS protocols that are available. The IEEE-802.1D Tagged Frame packet protocol uses a special three-bit QoS field in the extended Ethernet Frame header to determine the priority of the packet. This Layer 2 (Data Link Layer) protocol adds an additional field to the standard IEEE 802.3 format. As a result, any network components that do not support the Tagged Frame protocol would have to reject this packet as a non-standard format, thus negatively impacting interoperability. For this reason, ODVA strongly recommends the DSCP (Differentiated Service Code Point) protocol as its preferred EtherNet/IP QoS solution. DSCP utilizes the existing Type of Service (ToS) field in the IP header to determine the priority of the packet. The new CIP Motion Drive Device Profile specification has been modified to align with the ODVA recommendation stating that all CIP Motion devices shall implement DSCP marking for CIP Motion packets.

In order to get the most value from DSCP, it must be supported by more than just the Ethernet infrastructure components, i.e. the switches.. What about the NIC transmit and receive stacks? What about the device's internal bus interface to the NIC? What puts the CIP Motion connection data on the fast track through these components versus other non-motion data? Clearly, Ethernet QoS must extend into these components to provide an End-to-End solution. ODVA recently addressed this issue in Chapter 3 of Volume 2, the EtherNet/IP Adaptation of CIP specification by mapping DSCP codes to CIP Priority levels. CIP Motion packets are assigned a CIP Priority of Urgent. When a CIP component is faced with the decision of processing CIP Motion data versus non-motion data in the processing queue, the Urgent priority level trumps all other priority levels, so the CIP Motion data is immediately processed.

Traffic Type	CIP Priority	DSCP	CIP Traffic Usage
PTP event (IEEE 1588)	n/a	59 ('111011')	PTP event messages, used by CIP Sync
CIP Class 0 / 1	Urgent (3)	55 ('110111')	CIP Motion
	Scheduled (2)	47 ('101111')	Safety I/O, I/O
	High (1)	43 ('101011')	I/O
	Low (0)	39 ('100111')	Open
CIP UCMM CIP Class 3	All	35 ('100011')	CIP Messaging

Figure 6 – QoS Priority Mapping

We refer to the concept of priority-driven data processing through all components from the originator and target CIP devices as End-to-End QoS. With support for End-to-End QoS, it is possible to determine the maximum delays that CIP Motion data may incur as it passes through the connection components. For example, the maximum delay that a CIP Motion packet may experience in passing through a switch supporting QoS is the time for the switch to process a non-motion packet, which is approximately the transmission time of the packet. Suppose the maximum non-motion packet size is limited to 500-bytes. Based on 100 Mbps Ethernet data rate, the delay is approximately 40 usec, worst case. The effect a 40 usec delay has on overall motion system bandwidth is limited to a small reduction in the maximum axis per msec that the system can support; this is calculated at 1 to 2 axes per msec. Note that

because the CIP Motion information is timestamped using CIP Sync technology, this 40 usec delay for servicing a non-motion packet does not impact the real-time control of the axes.

The beauty of CIP Motion with End-to-End QoS and CIP Sync (IEEE 1588 Precision Time Protocol) is that the quality of motion control is substantially independent of the volume of non-motion traffic on the network components and, yet, this is still standard Ethernet!

## Startup and Synchronization

Details pertaining to the startup and synchronization of a CIP Motion system were worked out during our CIP Motion product development. Unlike other motion network protocols, CIP Motion does not require time slot negotiation as part of the connection startup procedure. The CIP Motion startup procedure proceeds as follows:

1. Forward Open
2. Configuration
3. Synchronization

Like other EtherNet/IP devices, a CIP Motion connection is opened when the originator, the controller in this case, sends a Forward Open service, to the target device, the drive. The RPI (Requested Packet Interval) of the Forward Open for the Class 1 I/O connection is set to the Controller Update Period. Once the C2D and D2C connections have been successfully established using Forward Open service, the controller and the device are each responsible for transmission of data packets at the RPI.

Once the controller is ready to manage the C2D connection, it begins sending configuration data to the drive using set attribute services. Configuration data is first targeted to class attributes of the Motion Device Axis Object, i.e. instance 0. After the class attributes are set, each axis instance is then configured, starting with axis instance 1. We took the approach of configuring the drive one instance at a time, but the specification allows for configuring the instances in parallel. A bit was added to the Node Control field of the C2D connection header for the controller to indicate to the drive that the configuration process is complete.

The final step in the CIP Motion startup process is establishing time synchronization between the controller and the drive devices. Concurrent with the process of opening a CIP Motion connection and configuring the axis instances, the drive device is receiving Time Sync packets from upstream IEEE-1588 time masters and synchronizing its local clock to the networks Grand Master Clock. The controller confirms time synchronization when it receives a “device synchronized” response to a Group Sync service request sent to the class instance of the Motion Device Axis Object.

Once all drives associated with the controller are successfully synchronized, the controller initiates synchronous connection operation by setting the Synchronization Control bit in the C2D Connection header. The drive device is then free to schedule D2C Connection updates based on the received Controller Time Stamp. From this point on, C2D and D2C packets are exchanged according to the CIP Motion timing model.

There were several cases we ran into where the Group Sync response from the drive indicated the device was synchronized when, in actuality, System Time for the device was severely skewed relative to System Time in the controller. One case that created a skew condition was when the drive device and the controller happened to not share the same Grand Master Clock. To address this criterion for the Group Sync “device synchronized” response needed to include confirmation that the drive device and the controller share the same Grand Master Clock.

Another case of unexpected source of clock skew turned out to be due to an unsynchronized intermediate IEEE-1588 boundary clock. While this is an unusual condition, it can be mitigated by having the drive device perform a check when the first D2C Connection update is scheduled. If the computed transmission time is out of tolerance, the drive indicates a clock skew alarm condition and the connection continues to operate in asynchronous mode. If the clock skew condition persists for more than 60 seconds, the drive issues a clock skew fault.

## **Scheduling the Next Update**

To insure that a Device-to-Controller Connection update occurs regardless of having received a Controller-to-Device Connection packet, during every cycle the drive must schedule the time for next cycle's Device-to-Controller Connection update.

For synchronous operation, the drive computes the time for the next cycle's Device-to-Controller Connection update to be enough before the end of the next update cycle to allow time for Device-to-Controller Connection data to be assembled and ready to transmit before the end of the cycle. For timer event interrupt-driven Transmit Tasks, this rescheduling occurs automatically when the local periodic timer reloads. For implementations where the Device-to-Controller Connection is handled by the Drive Task, the rescheduling must be done explicitly by the drive at the conclusion of every Device-to-Controller Connection update.

Not all applications require the drives to be synchronized with the control, for example variable frequency control (V/Hz) operation. For asynchronous operation, the drive initiates a task to assemble and transmit the Device-to-Controller Connection packet immediately after receiving (and parsing) a Controller-to-Device packet. The drive then schedules the transmission of the next Device-to-Controller packet to be 1 Controller Update Period from the time this Controller-to-Device packet was received. Thus, the transmission of that Device-to-Controller packet will take place even if the next Controller-to-Device packet is lost or late.

## **Fault & Alarm Log**

On the subject of improved drive diagnostics, CIP Motion introduces a novel way of providing vast amounts of fault and alarm information to the controller without consuming connection bandwidth. CIP Motion supports up to 128 distinct fault and alarm conditions per axis. That compares to 32 distinct faults and warnings defined for SERCOS. With SERCOS, these real-time faults and warnings must be transmitted to the controller every update cycle. This is inefficient given that, during normal operation there are no active faults or warnings. A CIP Motion device, on the other hand, sends no fault or alarm information during operation. When a fault or alarm condition occurs, the device records the condition in a local Fault Log or Alarm Log with an associated timestamp. When the next scheduled Drive to Controller connection update occurs, the device checks the Fault Log and Alarm Log for any new records. If a new record is found it is loaded into the Cyclic Data Block of the Drive to Controller connection data structure. The controller receives the Fault and Alarm record from the drive and applies it to its own Fault Log and Alarm log, thus replicating the Fault Log and Alarm Log in the drive. The controller's Fault Log and Alarm Log can be easily accessed by configuration software or HMI device to provide a powerful diagnostic tool. The figure below is representative example of a Fault Log displayed by configuration software.

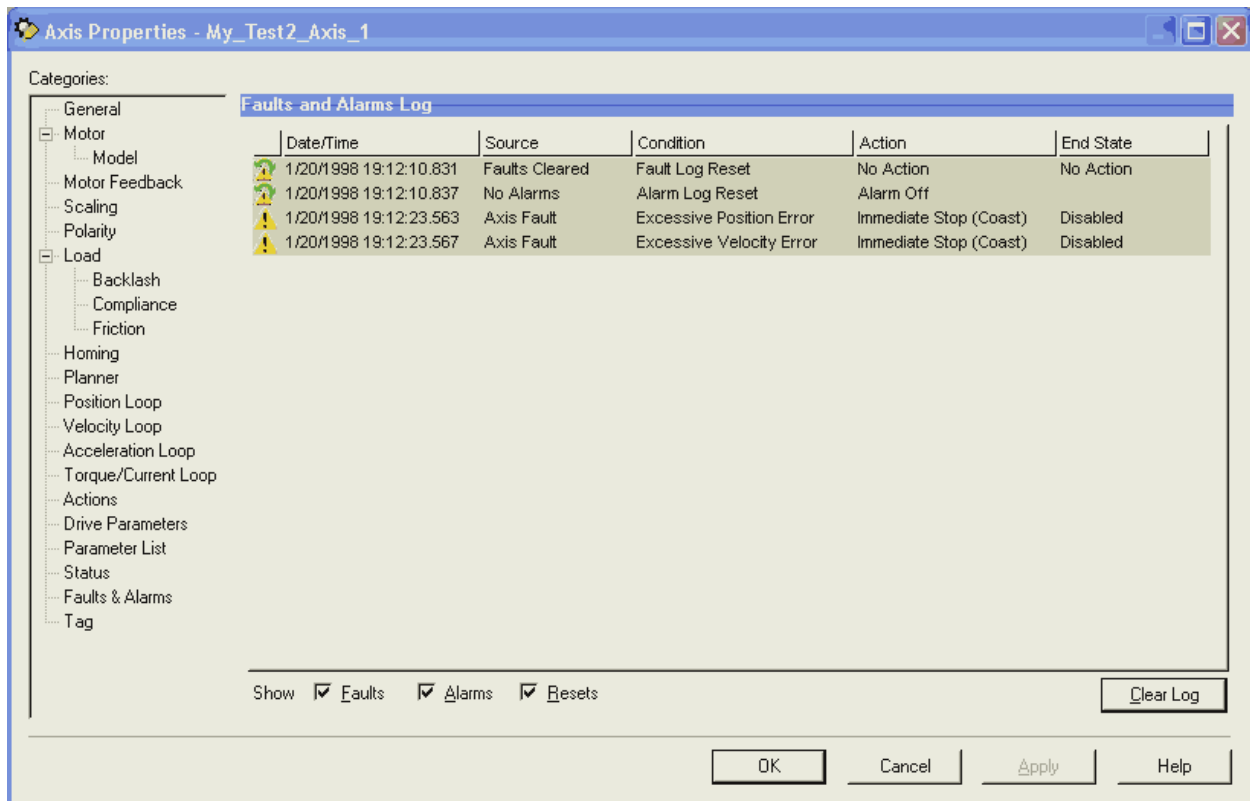


Figure 7 – Fault & Alarm Log

## Diagnostic Tools

One tool we found to be invaluable in developing and debugging our CIP Motion components is the freeware WireShark network protocol analyzer. WireShark can be customized to support many network protocols through Add-ins. We wrote an Add-in for WireShark that includes CIP Motion's connection data formatting rules. The result of this customization was that we could examine all elements of the CIP Motion connection data structure associated with captured CIP Motion packets traversing the network. This is in stark contrast to other application layer protocols where only the end nodes are capable of parsing the connection (I/O) data. Below is an example of a typical CIP Motion packet parsed by WireShark.

```

[-] Frame 7471 (126 bytes on wire, 126 bytes captured)
[-] Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: Allen-Br_
[-] Internet Protocol, Src: 192.168.1.40 (192.168.1.40), Dst: 192.168.1.33
[-] User Datagram Protocol, Src Port: 2222 (2222), Dst Port: 2222 (2222)
[-] EtherNet/IP (Industrial Protocol)
[-] Common Industrial Protocol, Motion
    CIP Class 1 Sequence Number: 3651
    [-] Connection Header
        Connection Format: Variable Device-to-Controller (7)
        Format Revision: 2
        Update Id: 47
    [-] Node Status: 0x07
        Instance Count: 1
        Node Faults and Alarms: 0
        Last Update Id: 47
    [-] Time Data Set: 0x00
        Device Time Stamp: 883613606412553490
        Lost Updates: 0
        Late Updates: 0
        Data Received Time Stamp: 883613873989856795
        Data Transmitted Time Stamp: 883613873991598175
    [-] Instance Data Header - Instance: 1
        Instance Number: 1
        Instance Block Size: 7 words
        Cyclic Block Size: 5 words
        Cyclic Data Block Size: 3 words
        Cyclic Read/Write Block Size: 2 words
        Event Block Size: 0 words
        Service Block Size: 0 words
    [-] Cyclic Data Block
        Control Mode: Position Control (1)
        Feedback Config: Motor Feedback (2)
        Axis Response: No Acknowledge (0)
        Response Status: Success (0)
    [-] Actual Data Set: 0x01
        .... ..1 = Actual Position: 1
        .... ..0. = Actual Velocity: 0
        .... .0.. = Actual Acceleration: 0
        .... 0... = Actual Torque: 0
        ...0 .... = Actual Current: 0
        ..0. .... = Actual Voltage: 0
        .0.. .... = Actual Frequency: 0
        Actual Position: 1319880960
    [-] Status Data Set: 0x00
        Axis State: Stopped (2)
    [-] Cyclic Read Data Block

```

Figure 8 – CIP Motion Packet Captured by WireShark Network Protocol Analyzer

Another powerful diagnostic tool associated with CIP Motion utilizes the timing diagnostics built into the CIP Motion connection header. Using these timing diagnostics it is possible to provide CIP Motion data delivery statistics. Below is a representative example of a Transmission Statistics dialog using this information. Using this diagnostic tool, the user can quickly determine if the network components are overloaded.

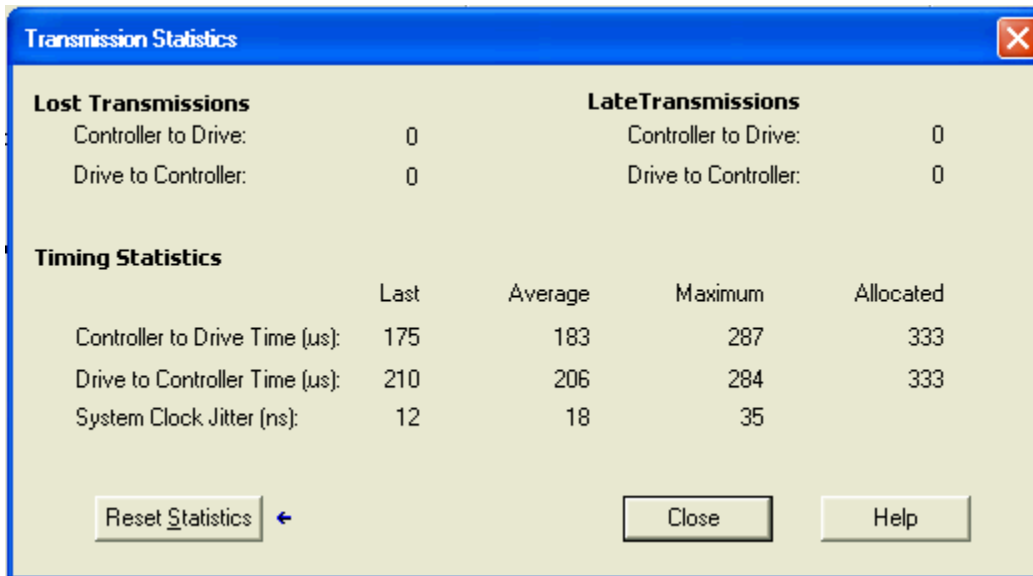


Figure 9 –Transmission Statistics

### Scalable Drive Profile

The CIP Motion Drive Profile is designed to cover drives ranging from simple Variable Frequency Drives (VFDs) to sophisticated Position Control Servo Drives. From a user interface perspective this creates some challenges to manage the configuration of these drives. The Drive Profiles for other motion networks are limited to servo drives so configuration software exposes all the attributes associated with the drives. Clearly, this is not the case with the new CIP Motion profile. With this highly scalable device profile, the difference between the attribute set applicable to a Frequency Control drive and the attribute set applicable to a Position Control drive is enormous.

This issue can be addressed by keying off the Device Control Codes listed in the Motion Axis Object specification. Device Control Codes are used in the attribute tables of the object specification to determine when the attributes are applicable. A section of this table is shown in the figure below.

Instance Attribute			Implementation by Device Control Code					
Attr. ID	Acc. Rule	Attribute Name	N	F	C - PI Vector Ctrl			Conditional Implementation
					V	P	T	
460	Set	Kaff	-	-	R	-	-	
461	Set	Kvp	-	-	R	R	-	
462	Set	Kvi	-	-	R	R	-	
464	Set	Kdr	-	R	O	R	-	
465	Set	Velocity Error Tolerance	-	-	O	O	-	
466	Set	Velocity Error Tolerance Time	-	-	O	O	-	
467	Set	Velocity Integrator Control	-	-	R	R	-	
468	Set	Velocity Integrator Preload	-	-	R	R	-	
469	Set	Velocity Low Pass Filter BW	-	-	O	O	-	
470	Set	Velocity Threshold	-	O	R	R	-	
471	Set	Velocity Lock Tolerance	-	O	R	R	-	
472	Set	Velocity Standstill Window	-	R	R	R	-	
480	Get	Acceleration Command	-	-	O	O	O	
481	Set*	Acceleration Trim	-	-	O	O	O	
482	Get	Acceleration Reference	-	-	O	O	O	
483	Get	Acceleration Feedback	R	-	R	R	O	
490	Get	Torque Command	-	-	R	R	R	
491	Set*	Torque Trim	-	-	R	R	R	
492	Get	Torque Reference	-	-	R	R	R	

Figure 10 – Attribute Implementation table excerpt based on Device Control Code

By embedding these rules in the configuration software and the controller, we were able to effectively manage the large attribute set and only expose attributes to the user if they were applicable. The two configuration screens shown below illustrate how the configuration tree changes dramatically depending on the Axis Configuration setting. On the left, the associated drive is operating as a VFD (Frequency Control), on the right as a position control servo drive (Position Loop).

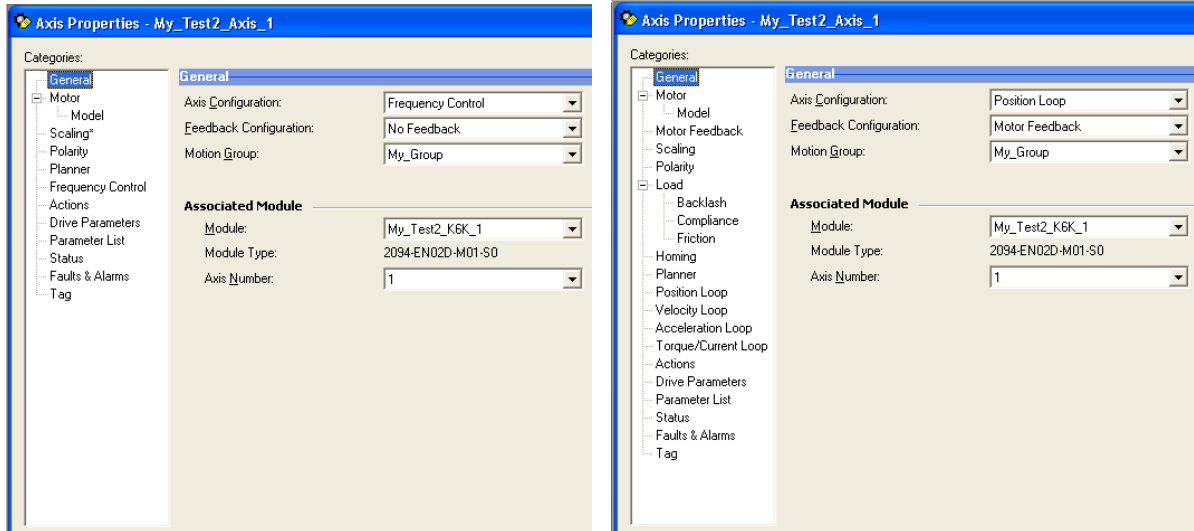


Figure 11 –Configuration Dialogs for VFD (left) and Position Control Servo Drive (right)

## Conclusion and Trends

We have learned a great deal in the process of developing a viable CIP Motion system. First and foremost we learned how we could best utilize the CIP Motion flexible data connection to optimize connection performance and reduce data processing burden on the drive and controller without compromising the rich data exchange between the devices. With contemporary off the shelf component technology CIP Motion can provide comparable performance to other motion network solutions without compromising standard Ethernet protocol and media access control (IEEE 802.3). Second, we learned how End-to-End QoS is the key to maintaining that performance level in a system where motion data traffic is freely mixed with non-motion data traffic.

We discovered that the current CIP Motion specification was inadequate in explaining how to start-up and synchronize a CIP Motion drive. The Distribute Motion jSIG is presently working to update the CIP Motion specifications to provide this detail.

Finally some important changes were proposed to improve connection diagnostics and we developed an extension to the WireShark network protocol analyzer that completely decodes CIP Motion data blocks. We enhanced the fault and alarm interface to a full featured Fault/Alarm Log with time-stamped fault and alarm conditions. These diagnostic tools are invaluable in developing and testing CIP Motion compliant products.

To assist developers in adopting CIP Motion technology, there are a number of tools available to facilitate their efforts. One tool, discussed earlier, is the freeware Wireshark, which is readily downloadable from the internet. A CIP Sync hardware assist FPGA toolkit is already available from one vendor which also supports two Ethernet ports and the Device Level Ring functionality recently released in the ODVA EtherNet/IP Specification. A combo CIP Motion/CIP Sync hardware assist FPGA is also possible, which would further facilitate CIP Motion adoption. As discussed earlier, these FPGA's could be viewed as analogous to those that are required to support other motion network solutions using SERCON, ERTEC, and FMMU chips. However, a CIP Motion/CIP Sync hardware assist mechanism is not required, and if used, would not break compliance with IEEE 803.2 standard unmodified Ethernet.

---

\*\*\*\*\*  
The ideas, opinions, and recommendations expressed herein are intended to describe concepts of the author(s) for the possible use of CIP Networks and do not reflect the ideas, opinions, and recommendation of ODVA per se. Because CIP Networks may be applied in many diverse situations and in conjunction with products and systems from multiple vendors, the reader and those responsible for specifying CIP Networks must determine for themselves the suitability and the suitability of ideas, opinions, and recommendations expressed herein for intended use. Copyright ©2009 ODVA, Inc. All rights reserved. For permission to reproduce excerpts of this material, with appropriate attribution to the author(s), please contact ODVA on: TEL +1 734-975-8840 FAX +1 734-922-0027 EMAIL odva@odva.org WEB www.odva.org

CIP, Common Industrial Protocol, CIP Motion, CIP Safety, CIP Sync, CompoNet, CompoNet CONFORMANCE TESTED, ControlNet, ControlNet CONFORMANCE TESTED, DeviceNet, EtherNet/IP, EtherNet/IP CONFORMANCE TESTED are trademarks of ODVA, Inc. DeviceNet CONFORMANCE TESTED is a registered trademark of ODVA, Inc. All other trademarks are property of their respective owners.