A Study of Time Triggered Systems

John Gittings

Advisor: Dr. J. Zalewski

FGCU

Ft. Myers, FL 33965

December 11, 2009

Table of Contents

1. Introduction to time-triggered systems 4
2. Problem Description - Anti-lock Breaking System 8
3. Solution Models 14
3.1 Quarter Vehicle Model 14
3.2 Four Wheel Vehicle Model 17
4. Experiments 19
4.1 QVM 21
4.1.1 Experiment A - Undisturbed State 21
4.1.2 Experiment B – Disturbing Individual Nodes 23
4.1.3 Experiment C – Interval Disturbance of Pedal Sensor 27
4.1.4 Experiment D – Disruption of Bus Channel 33
4.2 FVM 34
4.2.1 Experiment E – Undisturbed Model 35
4.2.2 Experiment F – Individual node disturbance 35
4.2.3 Experiment G – Pedal Sensor Disturbance 37
7. Conclusion 39
6. References 42
Appendix A. Overview of development for the TTTech Demonstration Cluster 43
A1. The Time Triggered Protocol 43
A2. Developing Applications for the TTTech Cluster 46
Appendix B. DemoCluster Application Quick Start Guide 48
Appendix D. TTTech TTP Build Quick Start Guide 78
Appendix E. TTTech TTP Load Quick Start Guide 101
Appendix F. TTTech TTP View Quick Start Guide 107

Appendix G. MATLAB Quick Start Guide 112

Appendix H. Experimental Disturbance Node XML Scenarios 125

H1. QVM Node 1 Disturbance XML 125

H2. QVM Node 2 Disturbance XML 126

H3. QVM Node 3 Disturbance XML 128

H4. QVM Node 4 Disturbance XML 130

H5. QVM Pedal Sensor Disturbance (1:5) XML 131

H6. QVM Pedal Sensor Disturbance (1:4) XML 133

H7. QVM Pedal Sensor Disturbance (1:3) 135

H8. QVM Pedal Sensor Disturbance (1:2) XML 137

H9. QVM Pedal Sensor Disturbance (3:4) XML 139

H10. QVM Pedal Sensor Disturbance (4:5) XML 141

H11. QVM Pedal Sensor Disturbance (5:6) XML 143

H12. QVM Pedal Sensor Disturbance (10:11) XML 145

H13. QVM Pedal Sensor Disturbance (20:21) XML 147

H14. QVM Pedal Sensor Disturbance (40:41) XML 150

H15. QVM Pedal Sensor Disturbance Channel A XML 152

H16. FVM Node 1 Disturbance XML 154

H17. FVM Node 2 Disturbance XML 156

H18. FVM Node 3 Disturbance XML 157

H19. FVM Pedal Sensor Disturbance XML 159

Appendix I. Special Development Platform Considerations 162

1. Introduction to time-triggered systems

Real-time computing systems require not only logical correctness of calculations, but that these calculations be available before a requirements defined deadline. If the deadline is exceeded the data is useless. Many real-time systems are also safety critical and failure to meet deadlines can result is catastrophic outcomes. For example, an anti-lock breaking systems may fail to measure the correct wheel speed in time, resulting in excessive application of the brakes which could lead to wheel lockup. At present, there are two primary protocols for relaying data within real-time, safety critical systems. The more commonly used, the event driven system, delivers data across the communication medium (bus) as they are being created. While under ideal conditions, this method of data transmission updates systems on the bus as conditions change. However, during periods of peak activity, messages compete for space on the bus. This can lead to delay, or even loss of data. Contrast this with a time-triggered system, which transmits data at regular intervals, bounded at a maximum time by the deadline requirements of the system. This approach guarantees complete predictability of the system as the worst case execution time mirrors the best case execution time [1]. The system's behavior is predictable and deterministic. The trade off for this approach is a relative level of inflexibility when attempting to add components to the system after the initial design phase.

Time-triggered systems transmit messages based on the global schedule. This global schedule separates real-time into discrete units. This allows units of the system, throughout this report referred to as nodes, to come to consensus that an event, *a*, which happens at time tick *j*, happened for all nodes simultaneously, so long as the duration of the tick is sufficiently large to account for the precision of time synchronization within the system. This global schedule is then separated into rounds where each participating node is allowed to communicated once (Fig 1.1). Rounds are strung together to allow for the transmission of more varied node data [4].

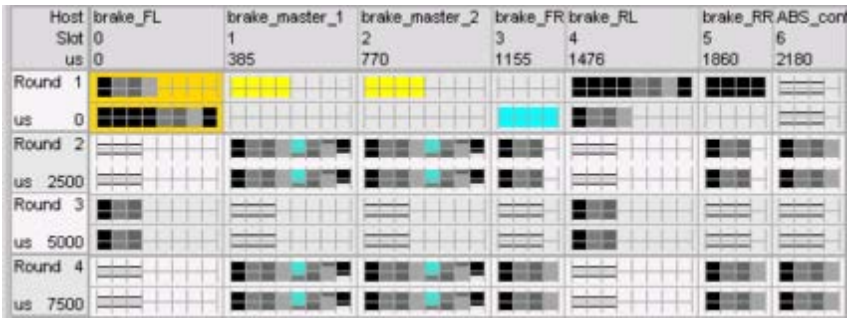

Figure 1.1 Global time-triggered schedule

The time-triggered architecture (TTA) consists of a number of distributed nodes, that model a system and are connected to one another across a real-time communication network [1]. Each node will consist of three components: the host controller that contains the system of interest, a communication controller that serves as the interface between the host and the third element, the communications network interface (CNI), referred to here as the bus. TTA specifies a dual channel bus for fault tolerance and the specification allows for the replication of nodes to ensure data integrity[1]. TTA also specifies methods for generating agreement amongst multiple copies of redundant data. This process, known as replica determinism, allows for communication controllers to handle data validation or pass the responsibility up to the host controller[4].

A key principle of time-triggered systems is the notion of composability. A time-triggered system, being a series of autonomous nodes, connected across the CNI, allows the separation of system level architecture and node level design. At each level of the design process for a time-triggered system, testing and certification can be performed without fear that future integration will create incompatibilities (assuming each component is designed to specification). Initially, system level architecture will specify the exact data and time intervals passed across the CNI. At this point, node level designers have the opportunity to meet the specification required while testing the node level implementation to ensure safety and reliability [4]. Integrating nodes must maintain stability of prior services, as each node will only communicate across the CNI as dictated by the globally shared schedule. In this way, nodes are easily replaceable in instances

of single point failure.

Absolutely vital to the functioning of time-triggered systems is the notion of a global clock. Given that time-triggered systems are intended for safety critical systems, single points of failure, such as a single global external clock, would present significant problems. The TTA specifies that each node contain a local clock along with the global schedule of communication [1]. Each node then generates a functional global clock as it receives messages according to the global schedule by comparing known signal times and times listed in the global schedule. Discrepancies between a node's local clock and the global schedule are corrected such that a de facto global clock is created. So long as each node possesses the same global schedule, agreement amongst all nodes in the system about a global time can occur.

The TTA specification allows for scalability via the introduction of bridge nodes. If a single application becomes too computationally intensive to continue running on a node and still provide its data within the deadline, that node can be expanded with little impact on the system [1]. A bridge node will act as a gateway between two independent time-triggered networks. The computationally heavy application is split among multiple new nodes and an entirely new system is created. The data created in this new time-triggered network are presented to the original network through the bridge as if it were a single node on the original system. In this way, the TTA allows for expansion based on system needs without requiring complete redesigns of the system. This becomes key in the commercial sector as cost and adaptability are critical components of a systems viability.

Many other safeguards exists within the current implementation of the TTA to handle more complex and varied, real-world failure states. Individual nodes contain “babbling idiot” safeguards on the communications controller, such that nodes that fail and send gibberish across the network can be silenced, and restarted, without colliding and disrupting communication [2]. Clique detection, or the detection of sub-groups falling out of sync with the global schedule, and remaining out of sync, relies on the TTA implementation of node membership. During each round

of communication, nodes record whether or not they were able to read the data from the current transmitting node. If they are able, that node is recorded as good, if not, then that node is marked as bad. Each round, when the transmitting node shares its membership listing as part of the overall CRC encoding embedded in the message, nodes that agree in memberships can decode their groups data. So long as other nodes receive and agree with this listing, that node is included in the global membership. If at any time a node finds itself as part of a membership that contains less than half of the total nodes, the node restarts in an attempt to rejoin the system [4].

The time-triggered architecture began development in 1979 at the Technical University of Berlin as part of the MARS project. In 1982, Hemann Kopetz began working on variations of the MARS architecture at the Vienna University of Technology, examining the critical need for hardware supported, fault tolerant clock synchronization in the time-triggered architecture[1]. Development continued through the 1980's and early 1990's. The University commercialized the protocol in 1998 and created the TTTech corporation, which focuses on safe, reliable real-time systems for use in transportation industries. Currently TTTech systems are used in the Airbus A380, Boeing 787 Dreamliner and the next generation of Audi A8 [5].

2. Problem Description - Anti-lock Breaking System

Anti-lock brakes have been around for decades, finding wide use in commercial aircraft in the 1950's[6]. In 1936, Robert Bosch filed the patent for ”Apparatus for preventing lockbraking of wheels in a motor vehicle“ [7]. During the next 40 years, Bosch would work on modifying his ABS system to produce a commercially viable product for cars and light trucks [7]. The Bosch corporation implemented the first commercially produced 4-wheel multi-channel ABS system in the Mercedes S Class in 1978 [6]. Other manufacturers employed anti-lock brakes as well. The 1969 Ford Thunderbird had a rear-wheel anti-lock breaking system, and the 1971 Chrysler Imperial had four-wheel anti-lock brakes[7]. Today, virtually all cars and light trucks have ABS as standard or optional equipment, with many companies providing ABS solutions for automakers. Anti-lock Breaking Systems (ABS) are designed to maximize breaking force during periods of sustained breaking by dramatically decreasing or eliminating wheel slip. Wheel slip (and wheel lock) reduce the braking ability of a vehicle as the dynamic coefficient of friction between tires and the road surface is much lower than the static coefficient of friction. Thus the stopping power of tires in wheel lock is greatly diminished. As the tire contact patch with the road is the only means of transmitting driver control for the vehicle, once the wheels have locked, the driver is no longer able to avoid hazards in the road to prevent a collision. While the implementation of ABS has changed over the years to incorporate advances in embedded systems, the fundamental principles of implementation have remained the same. To demonstrate the mechanics involved in the ABS, a simple quarter vehicle model consisting of a single wheel and break setup can be used (Fig. 2.1).

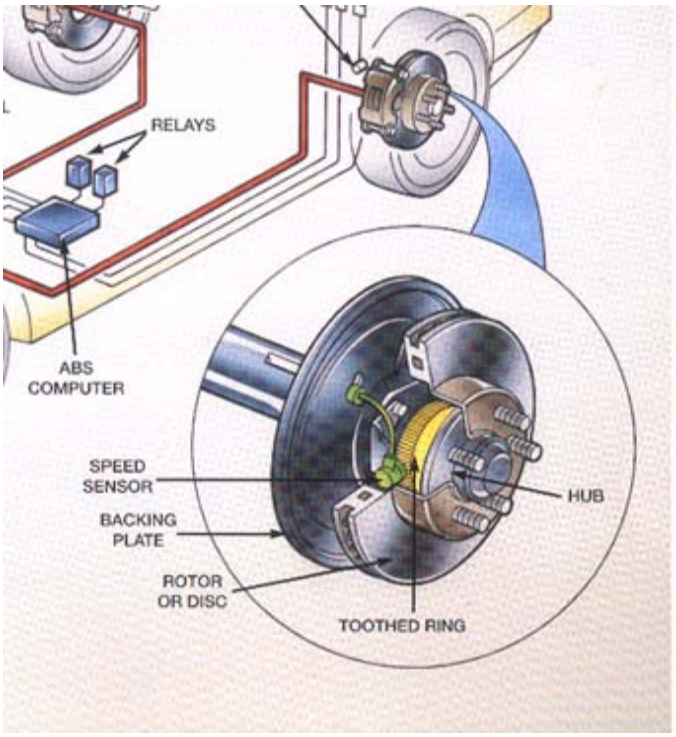

Figure 2.1. Quarter Vehicle Components [9]

At any given wheel, speed sensors will monitor the deceleration rate of the wheel. If the deceleration occurs too quickly and passes a designated threshold, the embedded ABS controller will reduce the braking force on that wheel. This is accomplished with the use of values at the brake caliper that either close the intake valve of the break master cylinder to maintain steady pressure, or open the outlet valve to reduce pressure and allow the wheel to accelerate out of wheel lock[7]. Through the duration of the stop, the ABS controller sends a series of signals to the brake master cylinder and pump to either increase or decrease the fluid pressure at that brake as needed. The driver of the vehicle will feel a series of strong pulses through the brake pedal. To an external viewer, a car undergoing active ABS breaking will appear to have its wheels fluctuate in speed or occasionally stop as the system modifies the breaking force needed to keep the wheels spinning in relation to the road surface.

At a minimum, an ABS system requires two additional components that work in tandem

with a vehicles brakes: the wheel speed sensors and the ABS controller that monitors the speed sensor and modulates the breaking. A single quarter vehicle model of a car, consisting of a single wheel, brake caliper, wheel rotor, speed sensor and controller is adequate to exam the basic functioning of the abs controller program at the wheel level, but does not adequately model the safety needs of such a critical system. This model presents multiple “single points of failure”. The system will fail if the speed sensor fails to transmit or if the bus transmitting the signal from the sensor fails. In order to avoid these failures, it would be necessary to add additional speed sensors to the wheel as well as redundant connections to ensure that adequate data returned to the abs controller. It is also possible to have a failure at the ABS controller. Given the nature of the failure and the implementation of the embedded operating system and control logic, the controller may be able to recover through a system restart, or the system may become inoperable, if only a single controller is used. Thus, the ABS controller would also require a redundant copy. However, in the commercial automotive market, this amount of redundancy would prove difficult to bring to market. It may drive the price high enough to prove noncompetitive, while also adding in additional points of maintenance for the consumer. In practice, the desired redundancy is achieved in the operation of four independent speed sensors, one located at each wheel. Loss of information from a single wheel does not necessarily cause failure of the entire ABS system (Fig. 2.2).

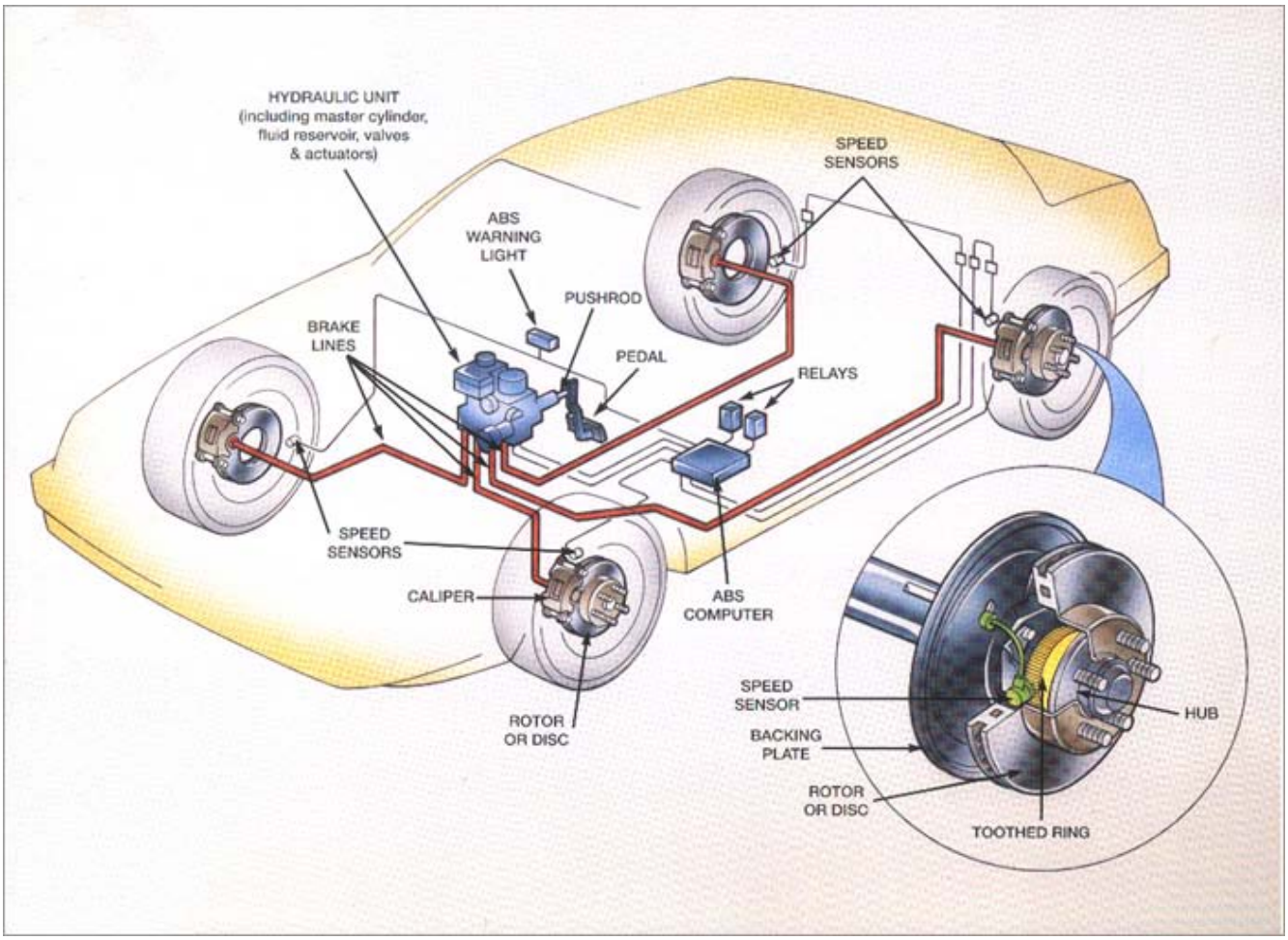

Figure 2.2. Complete Vehicle ABS Diagram [9]

Balancing the needs of safety and the market present interesting challenges. As noted previously, time- triggered systems that implement the time-triggered protocol focus on composability, which leads to cost savings and parts standardization. While not the focus of this study, it would be interesting to examine the safety gains and cost increases of utilizing a hybrid speed sensor with an embedded abs controller system that distributed the abs calculations to four independent nodes within the automobile. Would it be possible to justify the increased costs with the gains achieved in safety and composability? Another area for exploration, and the focus of this study, is fault tolerance within the time-triggered system. Given that the worst case execution

time remains constant within the time-triggered architecture, system designers are presented with some unique opportunities when testing fault tolerance and simulating failures of the system. Certain failures can be examined using the quarter vehicle model. Failure at the speed sensor and bus need not be total to cause significant problems within the ABS. Significant disruption of the speed sensor may prevent it from meeting deadlines updating the wheel speed, which causes anomalous results in the ABS calculations at the controller. At what threshold must the speed sensor provide updates for the ABS controller to maintain adequate and correct brake force for the wheel? Is it possible to fail “gracefully” and still utilize data from the sensor at a less than optimal frequency, and at what point is the data infrequent enough to consider the sensor failed? Once the sensor has failed, the four wheel model can then examine the viability of the vehicle's ABS system with the loss of a single node (wheel), and later examine at what point the system fails entirely (after the lost of two or three nodes).

The TTTech development cluster provides tools that can examine these various fail states. Through the use of a disturbance node connected to the cluster, it is possible to “knock out” or disrupt individual messages, such as wheel speed, as they are being sent across the bus. These disruptions are configurable and allow for the testing of both intermittent (both irregular and regular patterns) and total disruption of a node. What's more, these disruptions can also be targeted at the signals relayed from the ABS controller, simulating failure at the application level (Fig. 2.3). At what minimum frequency are brake updates required to maintain safe control of the vehicle under potential wheel lock conditions. Thus, the quarter vehicle and complete vehicle models can be implemented on the TTTech development cluster and then tested in various states of failure.

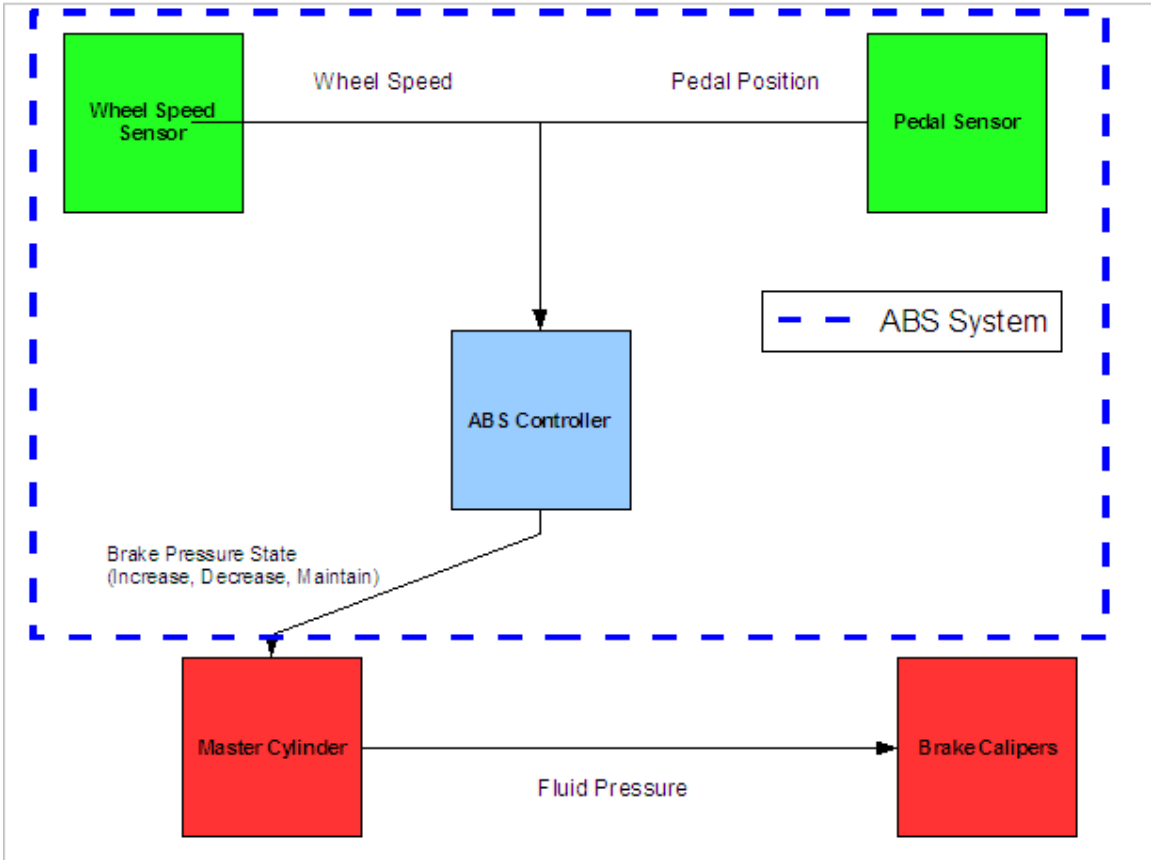

Figure 2.3. ABS control schema

3. Solution Models

3.1 Quarter Vehicle Model

The quarter vehicle model (QVM) is adapted from a model provided by TTTech. TTTech provides the model to mimic a "real-world" application running on the development cluster and demonstrate principles of redundancy and application load distribution. The QVM is divided into three modules: brake force calculation, wheel speed braking model and pedal position sensor. The wheel speed braking module and pedal sensor are each operating in a single node, while the brake force calculations are redundantly performed on three nodes. Every 3.4 ms, pedal position and wheel speed are updated and new braking force is applied to the wheel. Results from that rounds braking are calculated and the ensuing wheel speed is derived for the following round (Fig. 3.1).

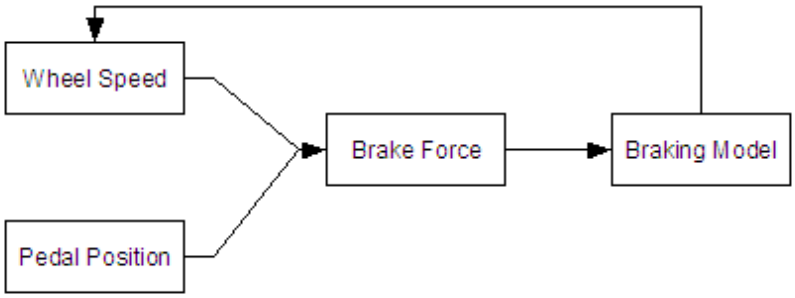

Figure 3.1. TTTech ABS Simulation Model

The algorithm for the brake force calculation is itself composed of two modules, the raw brake force calculation and the ABS control module. Brake force is determined as a simple ratio of the wheel speed, which creates a rough approximation of the necessary brake force in Newtons. (Fig. 3.2).

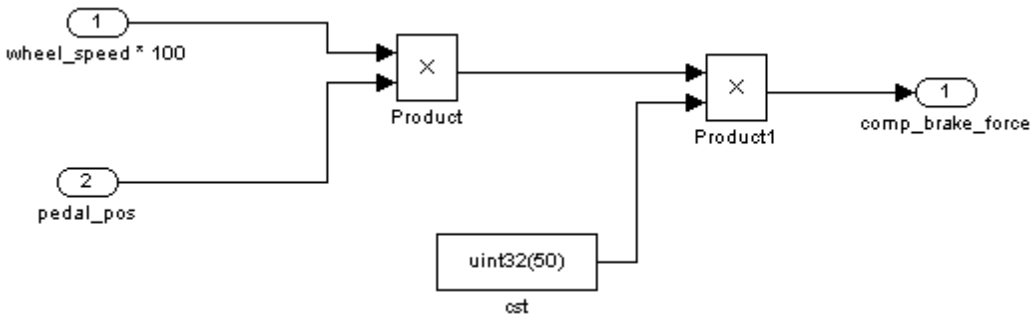

Figure 3.2. MATLAB representation of brake force model

The ABS logic is also straightforward, as brake force is removed from the wheel only when wheel lock occurs (Fig. 3.3). If wheel speed remains above zero, proportional brake force is applied. If wheel lock occurs, the switch condition sends a zero value instead of the calculated brake force.

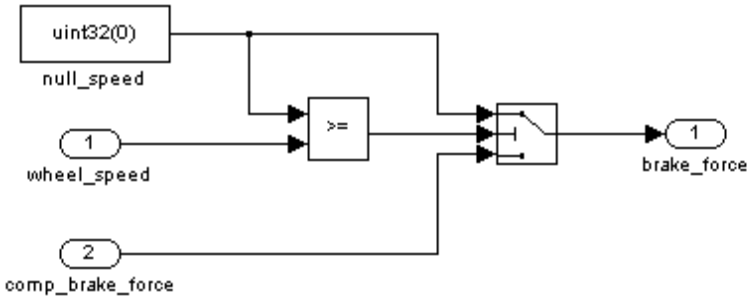

Figure 3.3. MATLAB model of ABS control logic

Once a brake force is applied to the wheel, the resulting deceleration is determined based on the model shown in Figure 3.4. Please note that the presence of the previous round's wheel speed is used only for the purpose of restarting the simulation after wheel speed has reached zero. It is not used in the calculation of the current rounds speed.

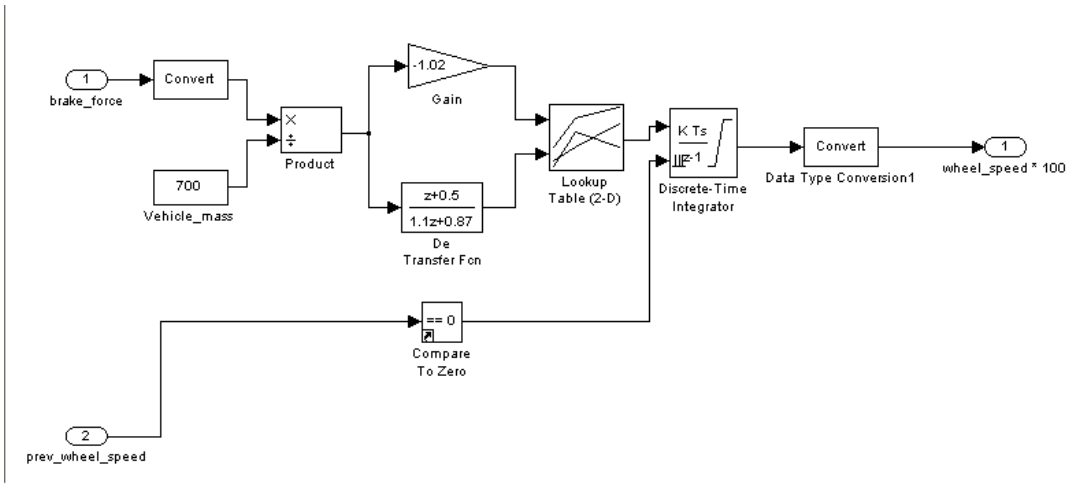

Figure 3.4 MATLAB model for quarter vehicle deceleration.

During simulation, the ABS controller brings the wheel to a halt in 2.3952 seconds (Fig. 3.5) from the initial wheel speed value of 300. The brake force applied to the wheel during the simulation is shown in Figure 3.6.

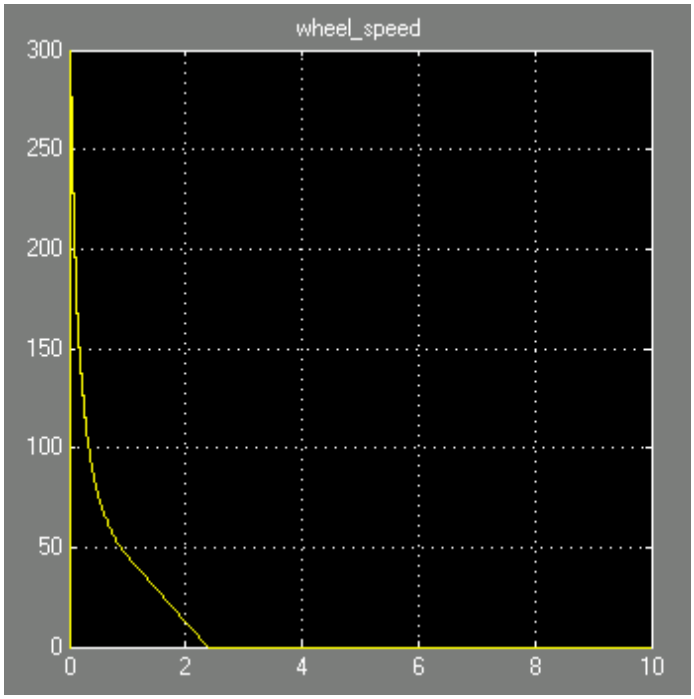

Figure 3.5 Wheel Speed (m/s) vs. Time (sec) in MATLAB Simulation.

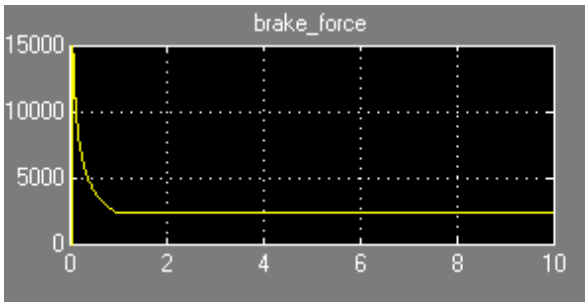

Figure 3.6 Brake Force (N) vs. Time (sec) in MATLAB Simulation.

3.2 Four Wheel Vehicle Model

The four wheel vehicle model (FWM) requires a different approach, given the constraints of working with only the four nodes of the development cluster. Each node needs to simulate wheel speed as braking force is applied. In addition, each node handles an additional subsystem: pedal position, vehicle speed and rotational velocity, ABS control, and average vehicle speed. In order to leverage the QVM deceleration model, the FWM simulates the entire mass of the car through four independent QVM instances that are averaged to create a total vehicle speed. This average is fed back into each wheels speed calculation to simulate the entire vehicle. The system is shown in Figure 3.7. A disadvantage of this system is lack of variation in any wheel creating a simulation that produces too much uniformity. One method to avoid this, would have wheels' starting speed randomized on initialization with a variation of ±1% from the QVM starting value of 300. Using the model shown in Figure 3.7, the starting variation in wheel speed is averaged out quickly as the entire vehicle achieves equilibrium. An alternative to this FVM model does not use the vehicle average speed as feedback to the wheels, instead using the speed averager as a means to report the speed of the vehicle, but allowing for the initial variation in wheel speed to remain in the system throughout braking. This later model is examined during disruption in order to evaluate the validity of the simulations.

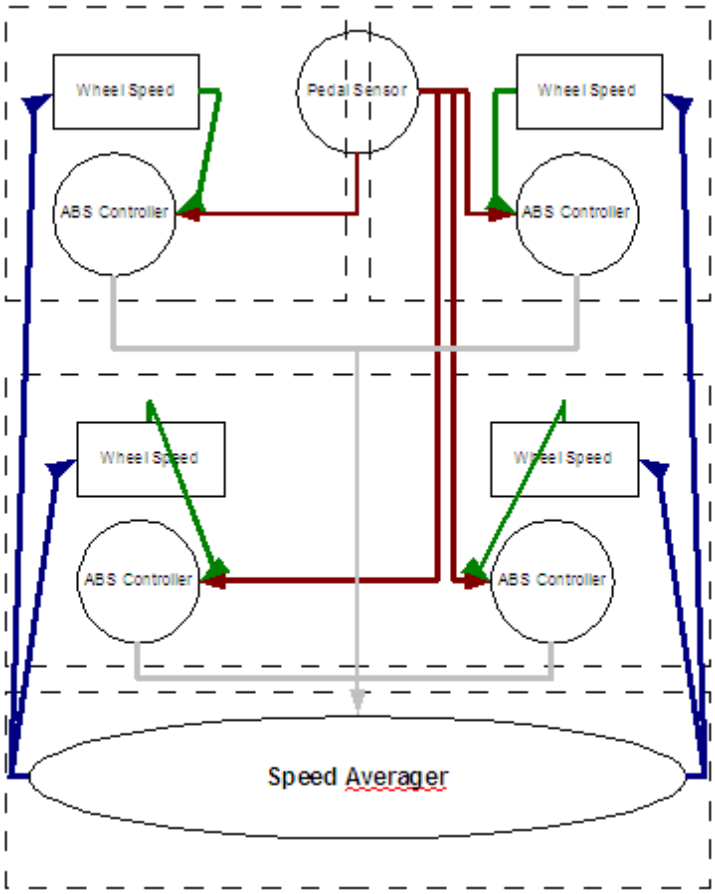

Figure 3.7. Full vehicle ABS model.

4. Experiments

Failure state simulations are run with the use of the TTTech Disturbance Node. This node connects to the bus and is activated through a serial interface with a host PC that transmits the disturbance parameters in an XML file. The XML defines the method, timing and length of interruption to be created. An example of the XML file is shown in Figure 4.1.

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
       <definition name="const_signal">
              <source channel="BOTH">
                     <duration>
                            <unsigned_intvalue value="700" />
                     </duration>
                     <signal_low />
              </source>
       </definition>
       <scenario name="set_to_run">
<medl>
020238000018381800283840003138710027389800470000
00000000000000000000000000000000000000000000d66f
d0050066018c0004032001040000f0823006f018c00040320
0104000f000000c8000000600180052c002c000400030004
000000000005001400020000003000000040003001af0965
00000000003f549e0003d14382369ff0d98c1dd400000000
e4e7edece7eee5edf9b5a1b3f9e6e4e4edd43ae09592e0e1
91e290eee0e19292303339383433a31392d6175672d323030
3920202020202020202020202020206a7367697474696e
2020202020202020c247000400000000002fffffff0b78
00111011000000000808080a1009570230411101100000000
808080a00094171f0811101100000000808080a0009451da
0c111011c00080008080c09f00934bde000800000000ffff
fffffffeb000011001108000805848480a10092e35b0411
0011080a080f848480a000915e8508110011081408198484
80a000917a280c110011c00000008080c09f009382cf0011
001100000000828280a1009417120411001100000008282
80a000931d2408110011081e0823848480a000917d2c0c11
0011c05080538282c09f00929c36
</medl>
              <reboot />
              <blackout_detection roundslot="0" />
              <repeat_scenario name="do" loop="2500"/>
       </scenario>
       <scenario name="do">
              <run_scenario name="disturb_a"/>
       </scenario>

       <scenario name="disturb_a">
              <wait_for_roundslot roundslot="0" offset="5" />
              <trigger name="TRIGGER_1" switch="ON" />
              <run_definition name="const_signal" />
```

```
            <trigger name="TRIGGER_1" switch="OFF" />
            <wait_for_roundslot roundslot="1" offset="5" />
            <trigger name="TRIGGER_1" switch="ON" />
            <run_definition name="const_signal" />
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>
      <scenario name="no_disturb_a">
            <wait_for_roundslot roundslot="0" offset="5" />
            <trigger name="TRIGGER_1" switch="ON" />
            <delay><unsigned_intvalue value="50"/></delay>
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>
</disturbance>
```

Figure 4.1 XML contents of disturbance node configuration.

The XML file is broken up into three sections. At the top is the definition node, which specifies the length and type of signal used for the disturbance. In this example, a low voltage signal will be sent on both channels and will last 700 microseconds. The second section, contained in the middle scenario node, defines the MEDL, which allows the TTTech disturbance node to synchronize with the cluster and send signals during specific rounds in the cluster cycle. This section also defines the number of times the disturbance should loop before completing. In this example, the scenario “do”, which contains one sub-scenario “disturb_a”, will be run 2500 times. Finally, the last series of scenario nodes define the round slot at which the disturbance signal should be sent and if a timing offset is needed before sending the signal. In this example, the scenario “disturb_a” is triggered on round slot 0 and round slot 1 of the TDMA cluster cycle and has a timing offset of 5 microseconds (the disturbance node will wait 5 microseconds before sending the low voltage signal).

The scenarios in use for the following experiments have been developed by the author in order to test features inherent to the TTP protocol. For the purposes of this paper, the disturbances are of two forms: removal of a node and intermittent interruption of messages from a node during the TDMA round. Every scenario's XML file is included in section H of the appendix. Please note that modifications to the existing TTTech QVM braking model are

introduced by the author to facilitate data acquisition during scenario execution. The TTTech braking model is not cyclical, and thus disturbing the entire model from the starting point is prohibitively difficult. By converting the model into a cyclical format, where the system resets itself after the wheel speed reaches zero, it becomes possible to introduce a disturbance at a random point in time but have the disturbance present when the system resets back to its initial state, allowing data collection from a known start state.

4.1 QVM

The first round of tests on the QVM measure the systems response to loss of nodes during operation. The disturbance node removes a node from the model by transmitting a constant low voltage signal on both channels of the bus during the target node's round. This disrupts the signal from the node, removing it from the TDMA group and removing the node's subsystems from the model. The disturbance node must be supplied with the cluster's MEDL, which is sent in the XML, in order to synchronize with the development cluster.

In the QVM, nodes 1, 2 and 3 of the demonstration cluster share the brake force calculation subsystem. Node 3 is also responsible for determining wheel speed based on the brake force applied. Finally Node 4 simulates the pedal sensor.

4.1.1 Experiment A - Undisturbed State

Initial measurements of the control state of the system are shown in Figure 4.2 and 4.3. Brake force and wheel speed follow an inverse logarithmic curve and the values for both come to rest in 767 rounds (at 800 microseconds per round, 613.6 milliseconds). These measurements are for the undisturbed model.

Note that the TTTech model in action performs differently from a traditional ABS braking model in use commercially. In a traditional ABS braking scenario, a driver would apply maximum braking force and the ABS controller would permit the force to continue so long as the wheel

continues to slow at a controlled rate. If the wheel begins to decelerate too quickly, braking force is decreased, preventing the wheel from locking. The resulting graph of brake force differs significantly from that of wheel speed as braking force quickly achieves a maximum value (the maximum force applicable at the edge of tire slip) and oscillates around this point. The TTTech model, however, defines the brake force and wheel speed to be directly proportional, as shown in Figures 4.2 and 4.3. This model, while not analogous to existing ABS systems, affords an easy to use system when applied to the TTTech demonstration cluster. Thus it is used in this study for illustrative purposes of the TTP protocol and demonstration cluster.

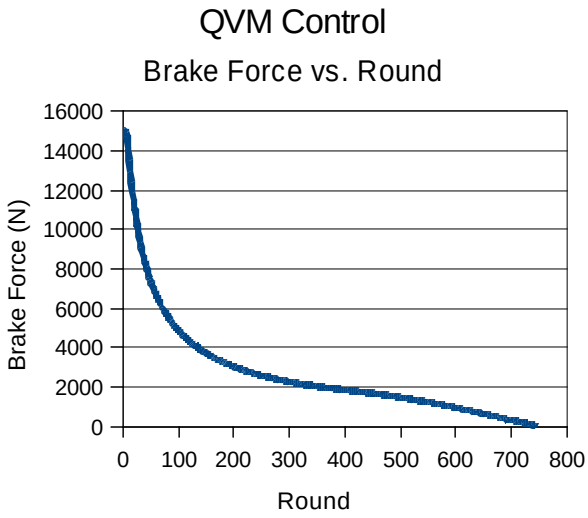

Figure 4.2 QVM control Brake Force vs TDMA Round.

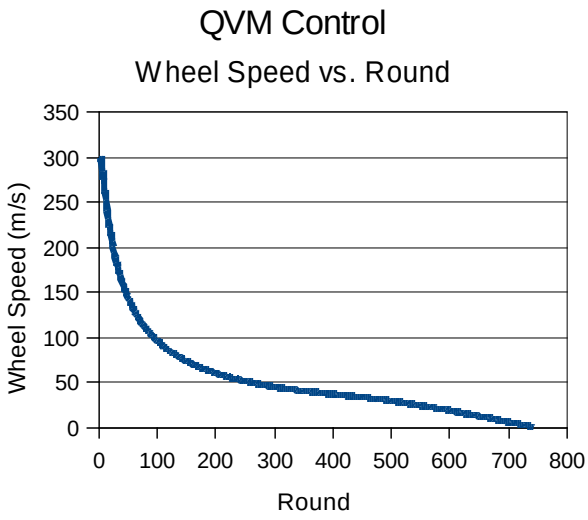

Fig 4.3. QVM control Wheel speed vs. TDMA Round.

4.1.2 Experiment B – Disturbing Individual Nodes

Next nodes 1, 2 and 3 are each disturbed to test the ability of the redundant systems to maintain system integrity during an instance of a node fault, as each system is redundantly calculating and transmitting applied brake force for the round.

Figure 4.5 shows the curve for wheel speed during the disturbance of Node 1. As expected, the wheel speed curve should show no difference from the undisturbed curve in Figure 4.3, as the braking calculation subsystem redundancy protects the system in the event of a single point of failure. Skipping ahead to Figure 4.7, the same pattern is apparent, as again, removing a single system (when three are present) should not affect the system as a whole.

The wheel speed outputs for Node 2, shown in Figure 4.6, shows an unexpected result. The figure has been expanded to demonstrate the disruption of the entire braking model during the disturbance, despite the existence of redundant brake calculation systems. This output shows a fault in the model that requires the brake force message from Node 2 to be present in

order to properly calculate wheel speed. This is most likely due to the author's modifications to the model introducing a timing dependance on the message as this is the last Node to calculate brake force prior to the calculation of wheel speed as shown in Figure 4.4.

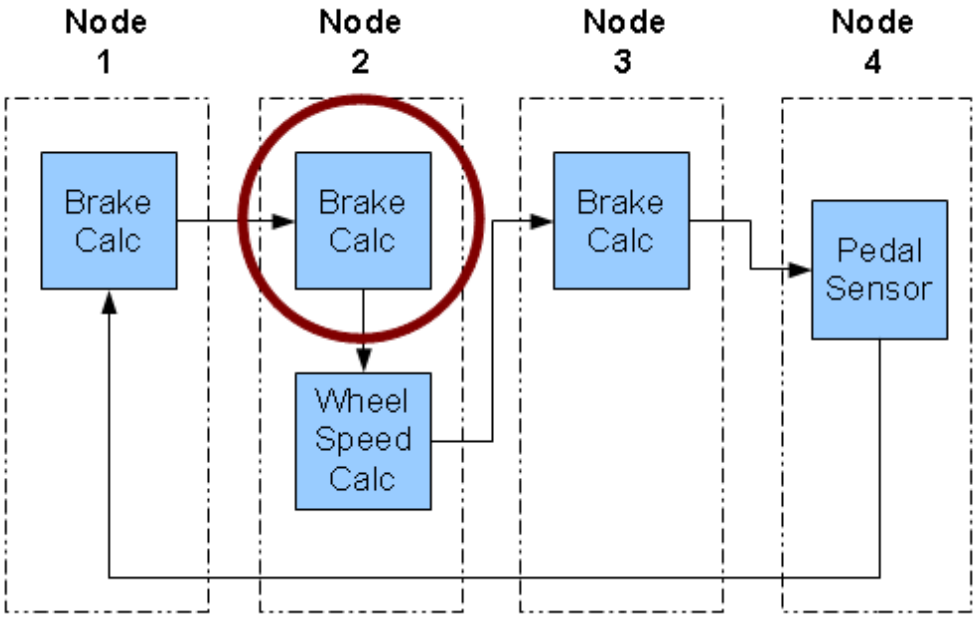

Figure 4.4. Timing diagram for Node 2's brake calculation.

Note that within the cluster schedule, this value for brake force is actually an average created from all the existing known good values of brake force calculated from the previous cluster cycle. Thus, node 2 should not have a heightened importance within the system. Further investigation into the specific failure is warranted.

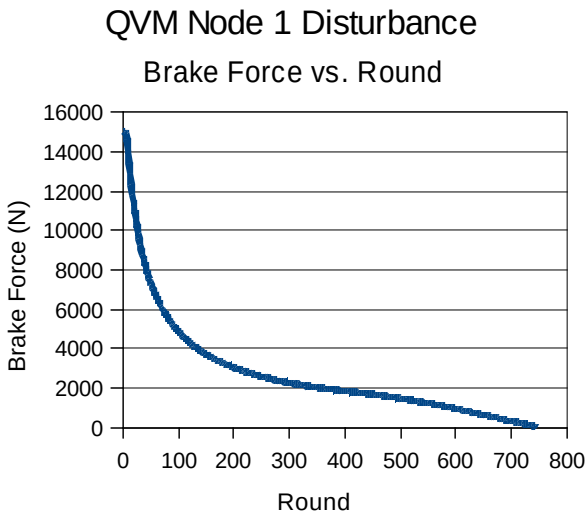

Figure 4.5 Node 1 disturbance of brake force calculation.

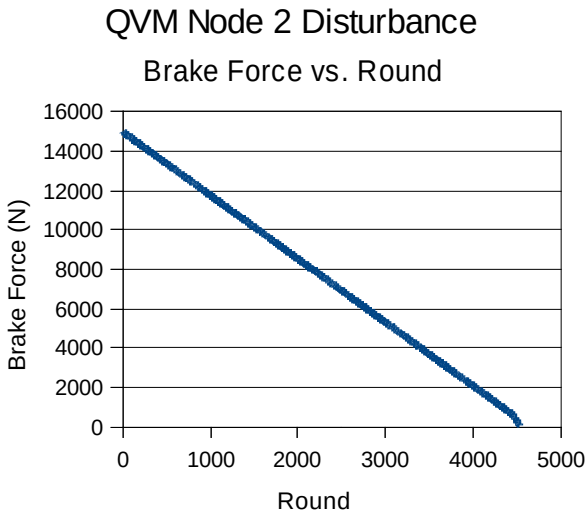

Figure 4.6 Node 2 disturbance of brake force calculation.

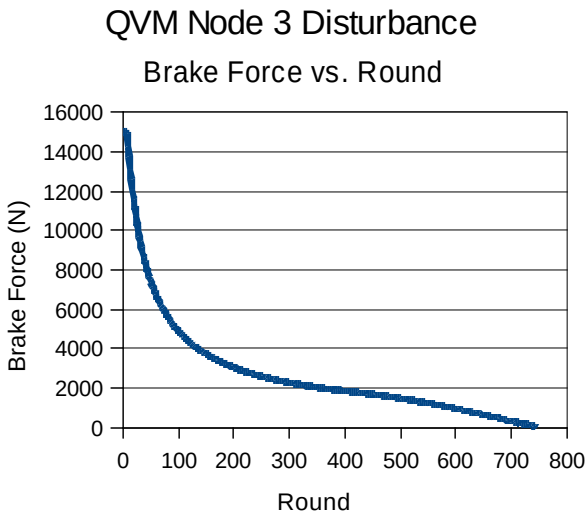

Figure 4.7 Node 3 disturbance of brake force calculation.

Interruption of node 4 removes the output of the pedal sensor from the model. This prevented the application of the brake and the resulting wheel speed graph is shown in Figure 4.8. Without the application of the pedal, there is no brake force applied. Thus the wheel speed gradually moves to zero as a function of the wheel speed model's simulation of friction on the tire.

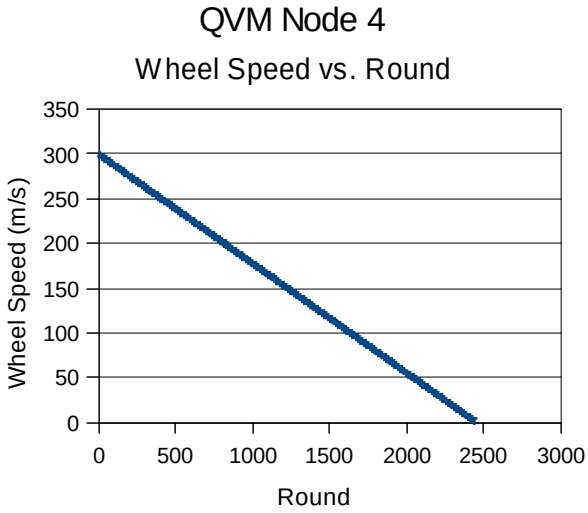

Figure 4.8 Disturbance of pedal sensor in QVM in the Development Cluster.

4.1.3 Experiment C – Interval Disturbance of Pedal Sensor

The following scenarios disrupt the pedal sensor message during transmission at varying intervals. Disturbances are sent across the bus in gradually increasing bursts, and the resulting wheel speed is monitored to determine the ability of the model to apply braking force under non ideal conditions. For each of the scenarios, the amount of disturbance is expressed in a ratio of the following format:

number of disturbances : total number of rounds in this cycle

Please note that the number of rounds in a cycle referenced above does not refer to a TDMA cluster or round, but rather the number of rounds that are disturbed plus the number of rounds that are left undisturbed. Thus a ratio listed as 1:4 would signify one disturbance of the pedal sensor message for every four transmissions of the message. Initially, one disturbance is sent across the bus every 5 rounds, as shown in Figure 4.9, and this produces little change in the graph from the control state shown in section 4.1.1. With the exception of the two scenarios discussed below, the rate of wheel speed reduction and the curve of the graph trended toward the graphs depicting absence of the pedal sensor message. Figures 4.9, 4.11 and 4.13 – 4.18 inclusive show the time needed to come to rest gradually increasing from 700 rounds to over 1800 rounds. Furthermore, as the time needed to stop increases, the rate of change slows as well and the graphs become increasingly linear.

As mentioned above, two graphs do not follow this pattern. Disturbances of 1:4 (Figure 4.10) and 1:2 (Figure 4.12) display an unexpected outcome. Namely, the simulation comes to rest sooner despite a decrease in the number of braking messages sent across the bus. As has been pointed out previously in this paper, the model in use here does not accurately reflect real world conditions and responses. As such, it is the author's opinion that these aberrant results would be eliminated as the model was modified to more closely match a real world environment.

However, for the sake of completeness, these results were included to show that testing was performed at regular intervals beginning at 20% (1:5) disturbance all the way to 97% (40:41) disturbance. Despite the notable weakness of the model, it is encouraging to see that the model was capable of handling the intermittent disturbance without failing outright. At 75% total disturbance, the model was only reduced in effectiveness by 16%, stopping within 800 rounds where the undisturbed model comes to rest within 700 rounds.

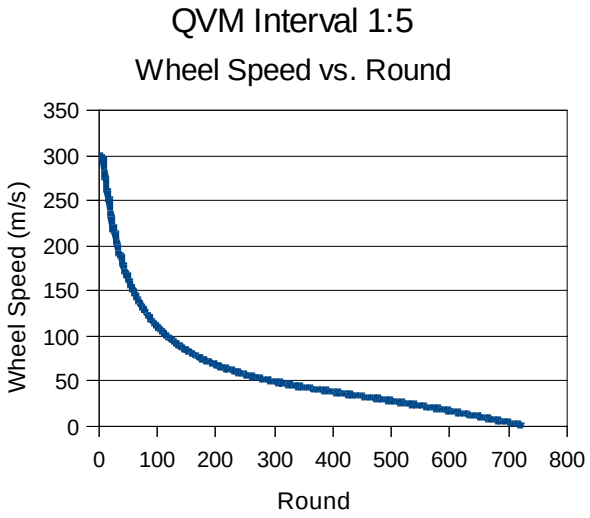

Figure 4.9. Disturbance Interval of 1:5

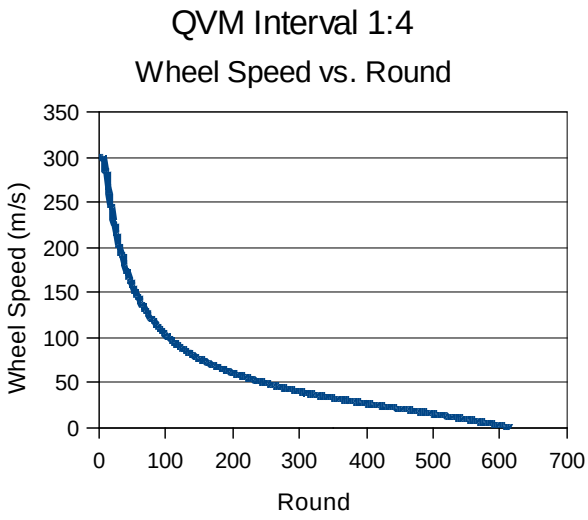

Figure 4.10. Disturbance Interval of 1:4

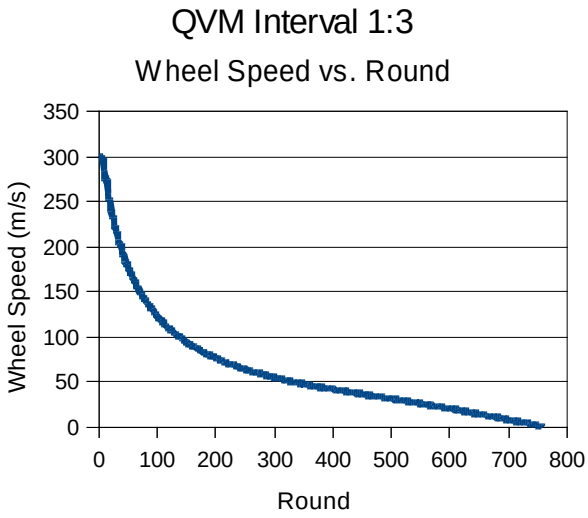

Figure 4.11. Disturbance Interval of 1:3

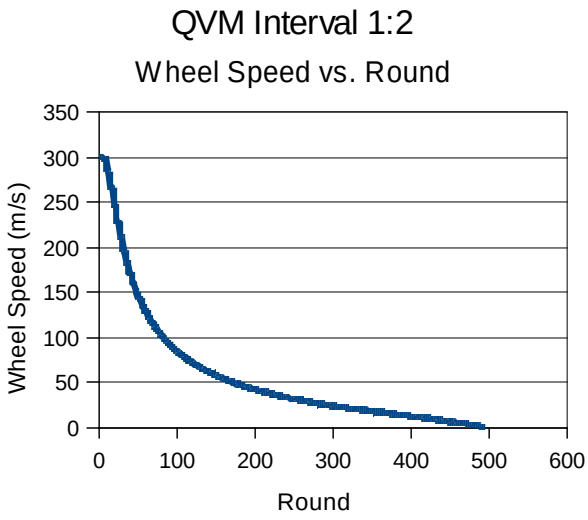

Figure 4.12. Disturbance Interval of 1:2

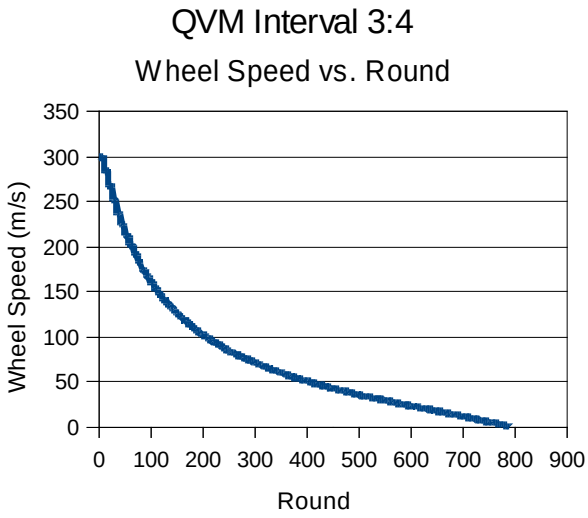

Figure 4.13. Disturbance Interval of 3:4

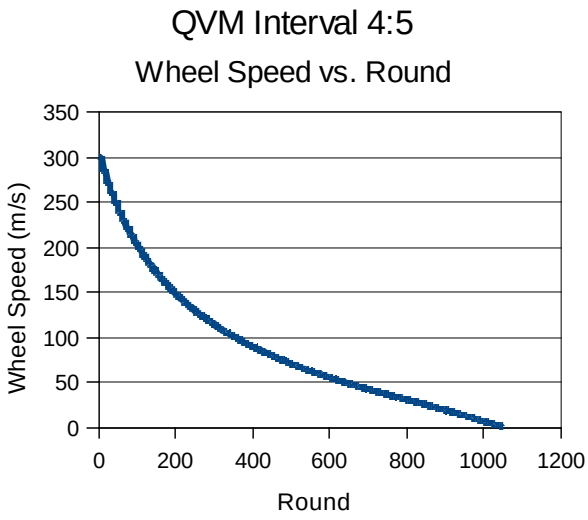

Figure 4.14. Disturbance Interval of 4:5

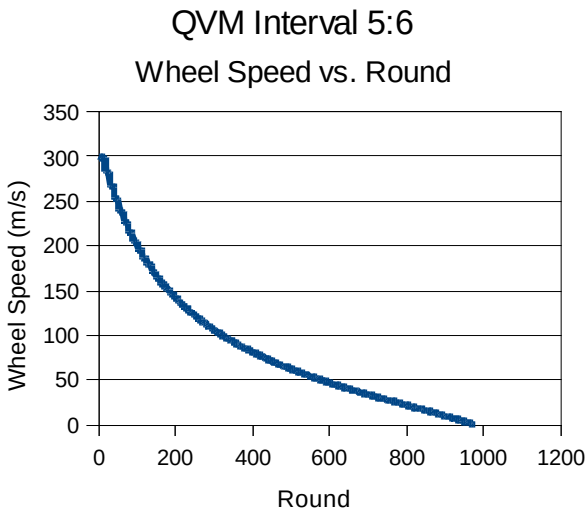

Figure 4.15. Disturbance Interval of 5:6

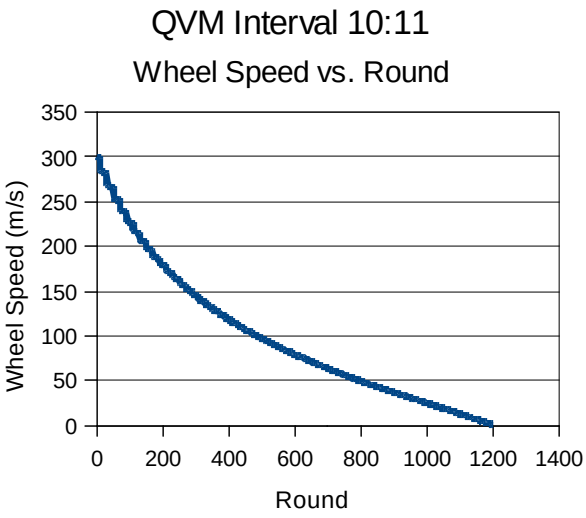

Figure 4.16. Disturbance Interval of 10:11

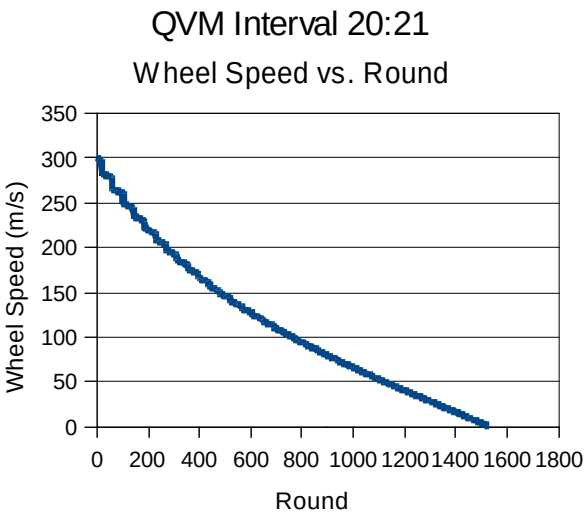

Figure 4.17. Disturbance Interval of 20:21

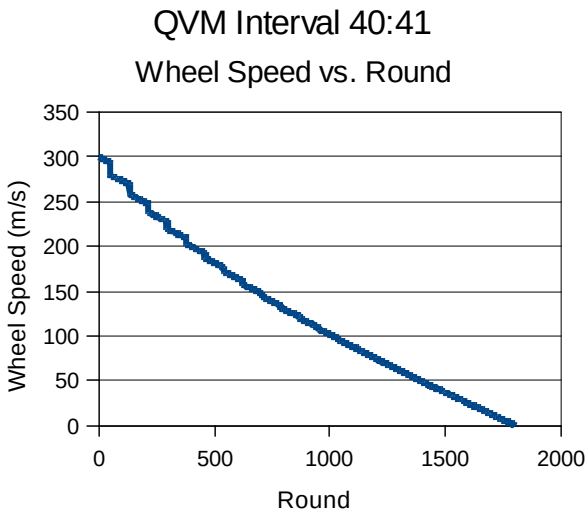

Figure 4.18. Disturbance Interval of 40:41

4.1.4 Experiment D – Disruption of Bus Channel

Finally, the last QVM scenario, demonstrates the ability of the TTTech development cluster to handle signal disruption on one of the two bus channels. The results are shown in Figure 4.19. This test confirms a feature of the TTP protocol that handles all message passing across the the channels of the bus without requiring intervention from the application to handle the error. Once received by the TTP communication layer at the node, the messages from each channel are compared and a final version of the message is passed to the application without any intervention required. As Figure 4.19 shows, there is no functional disruption of the pedal sensor and the curve mirrors that of the undisturbed nodes in section 4.1.1.

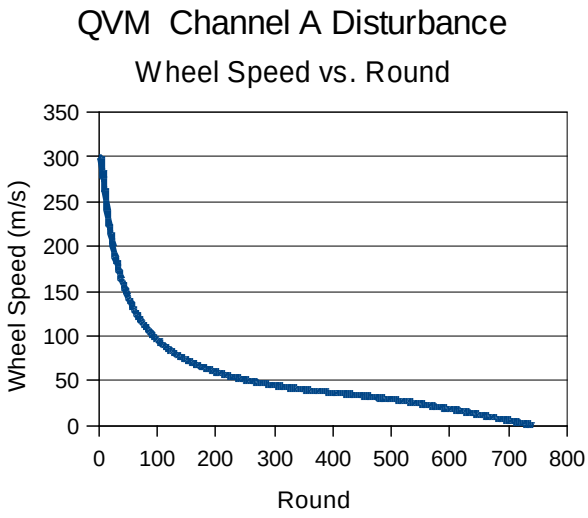

Figure 4.19. Disturbance of pedal sensor (channel A).

4.2 FVM

Development of the FVM builds on the modified version of the QVM from section 4.1. The same cyclical single wheel model is present in four separate instances to simulate the four wheels of a standard passenger vehicle. To determine the speed of the entire vehicle, the speed from each of the four wheels is averaged during each TDMA round. Again, as in the QVM model, the TTTech model is modified in order reset to the beginning state once the wheel speed has reached zero. The 4 QVM models run (Node 1 contains wheel 1, Node 2 contains wheel 2 and Node 3 generates wheels 3 and 4) and calculates the average during the last round of the TDMA cluster on Node 4. This average attempts to handle the disturbance of a single node's output by substituting the previous cluster round's value for averages wheel speed in the absence of input. In this way, the system should adapt to failure of a single wheel without failure of the entire system. Each node is disturbed to test the effects on the calculation of average vehicle speed. Finally, an attempt is made to remove the pedal sensor from the system to examine the effect on the model's average vehicle speed.

4.2.1 Experiment E – Undisturbed Model

The experimental approach to the FVM mirrors that of the QVM. First the undisturbed model is examined. The baseline of the model is run and average wheel speed for the entire vehicle is shown in Figure 4.11. As can be seen in Figure 4.20, the average wheel speed curve is similar to the QVM model's undisturbed output in section 4.1.1.

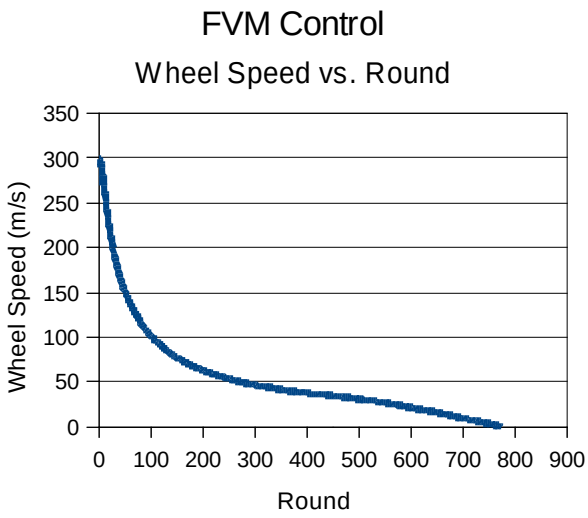

Figure 4.20. FVM control Average Wheel Speed vs. TDMA Round

4.2.2 Experiment F – Individual node disturbance

As in the QVM, the disturbance node disrupts each of the nodes and the average wheel speed for the vehicle is shown in Figure 4.21, 4.22 and 4.23. In each graph, the point of disturbance occurs late in the process, but completely disrupts further calculation of wheel speed for the duration of the disturbance. This results from an error in the FVM model that prevents the average speed from reaching zero with out all four inputs being present. As each QVM requires a zero output from the average wheel speed sensor to reset themselves back to their starting values, each of the wheels runs to near zero and holds until the disturbance ends. In order to

correct this flaw in the model, the averages wheel speed sensor would need to detect when an input value is missing and rather than substitute a number to replace the missing value, should recalculate the average with only the inputs involved.

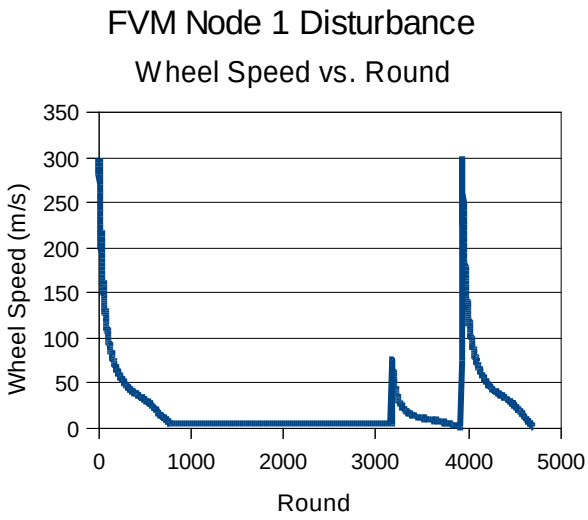

Figure 4.21. FVM Disturbance of Node 1.

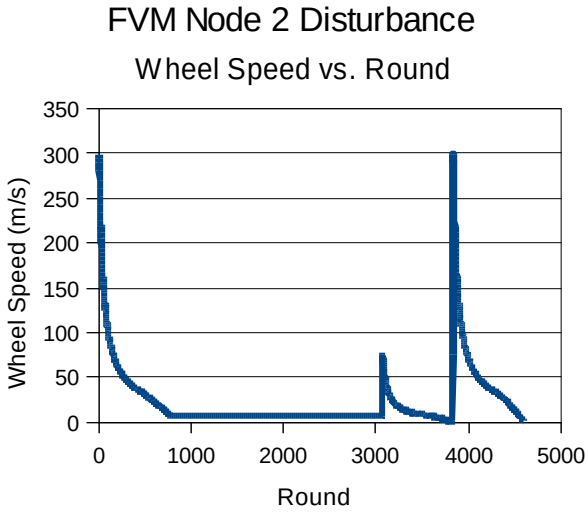

Figure 4.22. FVM Disturbance of Node 2.

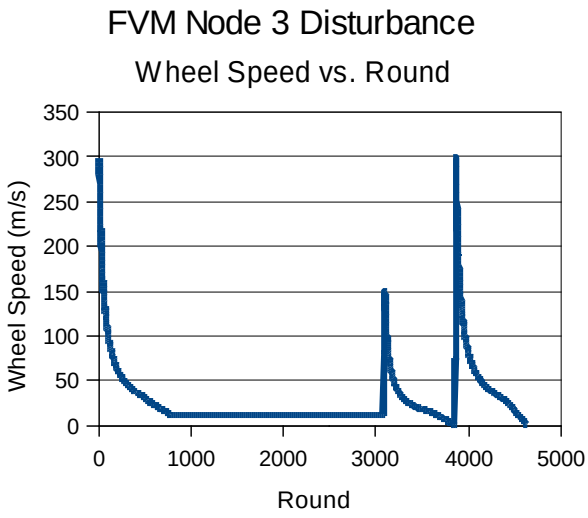

Figure 4.23. FVM Disturbance of Node 3

4.2.3 Experiment G – Pedal Sensor Disturbance

Finally, the attempt is made to mirror the pedal sensor disruption performed on the QVM. However, given that the pedal sensor is redundantly transmitted on both nodes 1 and 2, disrupting both in the same TDMA round generates a clique error within the TTP protocol and forces the development cluster to reset. A clique error is the presence of two or more subgroups within a TDMA cluster that are no longer able to communicate. When a single node is unable to communicate with the cluster, it should recognize that it has entered an error state and restart in an attempt to reintegrate into the cluster. When more than one node partition off into groups, without a clique avoidance algorithm, the nodes in each group are able to communicate with a subset of nodes and thus are unaware of the much larger error involved. The TTP protocol defines a clique avoidance algorithm that monitors the creation of subgroups and always forces the subgroup with a majority of nodes to “win”, causing the other smaller groups to reset and reintegrate. In this experimental scenario, a clique error is generated when two nodes are knocked out, and the entire cluster resets. To avoid the clique error, an attempt is made to

disrupt each node on alternating rounds. The resulting average wheel speed is shown in Figure 4.24. This wheel speed is identical to the undisturbed state as the pedal sensor output is available from at least one node during each round.

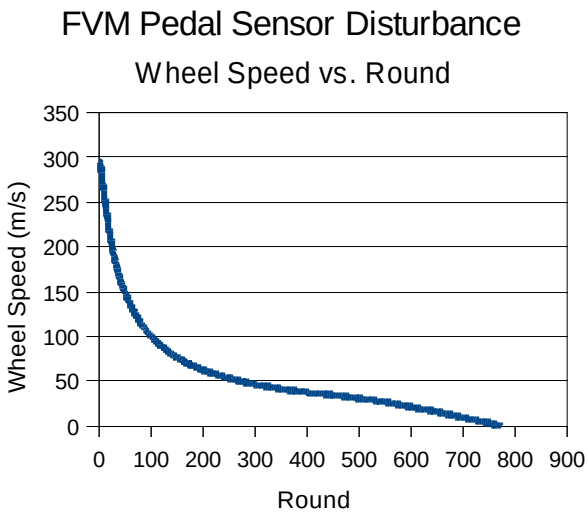

Figure 4.24. FVM Alternating Pedal Sensor Disturbance

7. Conclusion

Anti-lock braking systems are one of the cornerstones in modern automobile safety. Millions of passenger vehicles and light trucks employ ABS as part of a host of critical systems that modern drivers depend on every day. Central to each of these systems are the embedded computers and communications infrastructure that allow each to perform at blinding speed and accuracy, far faster and accurate that a human is capable. Four wheel independent anti-lock braking and traction control systems allow drivers to navigate terrain that previously would be impossible to traverse. Given that all of these systems are absolutely critical in the safe functioning of the vehicle and that each must be able to perform under non optimal conditions, the design and protocol used in these systems must be able to function under duress and must handle failures without compromising safety. Additionally, like any safety critical system, components used must be cost effective, and easily testable prior to deployment.

One protocol that meets these needs is the time-triggered protocol. Developed at the University of Vienna, TTP has undergone over 10 years of research and development. This protocol differs from the more common event driven protocol in that all messages sent across the bus do so based on a schedule. Therefore performance of the system is deterministic and the worst case message transmission rate is equal the average message transmission rate for a given system.. Since the time-triggered protocol requires the entire system and schedule of communication be created at design time, it allows for easy integration of component nodes. This determinism also makes testing and certification a much easier and cost-effective process.. The time-triggered protocol also incorporates many features that layer system safety without any need for additional application code. Communication across the bus is sent on two channels. Subsystems can be redundantly deployed on multiple nodes in the system and the messages they produce are integrated by the TTP controller and presented to the application without any application code becoming involved.

Many tools exist to rapidly prototype and develop for TTTech systems. As shown in this paper, models develop quickly with tools such as MATLAB and can be field tested quickly with hardware like the TTTech development cluster. Even more valuable, failure and problems in the system can be created and tested in a systematic way with the addition of the TTTech Disturbance Node. This is a potent combination of hardware and software that allows system architects to develop, test and finalize complicated new applications that rely on time-triggered technology.

The experiments discussed in this paper sought to demonstrate the various safeguards inherent in the time-triggered protocol, while also demonstrating the operation of a simple ABS model under various levels of stress. In all instances, the TTTech development cluster performed as expected, with the various safety systems performing as per the protocol. In fact, in at least one instance, the ability of the TTTech system to detect more complicated, system level errors prevented the experiment from inadvertently creating a clique situation, where two or more groups of node began working independently from one another. While this made it difficult to test the ABS models functioning under stress for that particular scenario, it did serve as a learning point about the many error detection subsystems built into the TTP controllers.

There is still much to be done with both the TTTech development hardware and the investigation of safety critical systems like the ABS models discussed in this paper. For one, the models used here are largely only useful from a teaching perspective, as they fall well short of what would be necessary if they were to be deployed in a real world environment. While they were quickly created and molded to fit within the four node design of the development cluster, they failed to mirror the data produced by real world ABS systems. In the future, creating a more robust model that achieved maximum braking force quickly and then oscillated about a tires slip point would be the logical next step. There were hurdles to this however, and the amount of time afforded this project prohibited developing a more realistic model. Furthermore, the ability to simulate the road surface tire interaction more realistically is a notoriously difficult and demanding

task, which is often handled by tire manufacturers. Procuring one of these models was again, outside the scope of this project.

Further work can also be done with the models and their error handling ability. The author would like to see more work done on both the QVM and FVM to better handle the disruption of inputs such as the pedal sensor or a single wheels input, rather than fail entirely as the models did under testing. In addition, application code could be written to handle the various high level system events and errors that can occur (such as clique errors and babbling idiots) that would allow the models to recover from these events. Writing this type of application code would require more time spent with the API of the embedded controllers of the demonstration cluster, as this type of code is not generated from the MATLAB rapid development tools.

Ultimately, educating about the time-triggered protocol, ABS systems and the TTTech development cluster has been the goal of this paper. While not producing any new scientific insights, it is the author's opinion that during the process of this project, a great deal of new personal knowledge regarding these systems has been obtained. Much more remains to be accomplished and there is certainly room for genuine scientific inquiry regarding the time-triggered protocol and the development of applications that make use of its unique benefits. With time, this paper could be expanded to meet these needs and it is the author's hope to have the time and resources to make this a reality.

6. References

1. H.Kopetz, The time-triggered architecture, In Proceedings of the 1 st international symposium on object-oriented realtime distributed computing, April 1988, pp. 22-29, IEEE Press.
2. D.A.Gwaltney and J.M. Briscoe, Comparison of Communication Architectures for Spacecraft Modular Avionics Systems, Marshall Space Flight Center, Alabama, 2006.
3. M. Pont, Patterns for Time-Triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers, 2001, Addison Wesley Professional.
4. TTP Seminar, TTP mechanisms: the protocol in detail, June 2009.
5. TTTech Company Overview, http://www.tttech.com/company/overview/company-profile/, 2009.
6. Q&A's: Antilock brakes — cars, trucks, motorcycles, Insurance Institute for Highway Safety, http://www.iihs.org/research/qanda/antilock.html, 2009
7. 25th Bosch ABS Anniversary, http://www.bosch.com/assets/en/company/innovation/theme03.htm ,2003.
8. D. Burton, et.al, Evaluation of Anti-lock Braking System Effectiveness, April, 2004, Royal Automobile Club of Victoria Ltd.
9. Courtesy Goodyear, Tacoma, Washington, http://goodyeartacoma.com/services/brakes/, 2009.

Appendix A. Overview of development for the TTTech Demonstration Cluster

The goal for this document is to give the reader a broad overview of the process and interaction between the software components used in the TTTech Development cluster (TTP-Plan, TTP-Build, TTP-Load and TTP-View). To begin, this appendix goes over the structure for how the TTTech system is normally deployed and the philosophy behind development for TTP systems in the automotive and aerospace industries. Next, it briefly overviews the TTP scheduling system and the premise behind time triggered systems. Finally, it outlines the steps needed to go from a basic application idea to design and implementation on the TTTech development cluster. It must be noted, however, that at no time is this document, or any of the other quick start manuals, a replacement for the TTTech documentation. This appendix is designed to assist in getting a user up to speed with development and not meant as a comprehensive guide to the embedded systems or protocols of the TTTech hardware.

A1. The Time Triggered Protocol

The TTP (time triggered protocol) is used to ensure predictable and reliable communication over real-time, safety critical systems. It differs from the more common event driven systems (where every item on the bus can communicate at once) in that each system on the bus is allowed to communicate only at predetermined time intervals. The communications for systems on a time triggered bus are defined at design time and will not change from that point on. Since many different producers and subcontractors are used in the creation of the final product, it allows for more certainty that when the parts are finally all put together, they will function as

specified. Understanding this issue assists in piecing together all the tools used in the process of creating applications that communicate over the real-time bus. The method of development and deployment in the academic setting is atypical. The normal pattern of development would have a systems architect, who creates the higher level system designs for the project as a whole (for example an automobiles break by wire system) and then creates a communications schedule that each of the component parts must follow. Once the schedule is created, it is handed off to the components subcontractors, who will then build their subsystems to specification. From this, we can see that all of the TTTech tools assist in the scheduling and communication of data over the bus, and are less concerned with the actual applications that consume or produce that data. In addition to developing software in C using the Wind River Diab compiler, TTTech does offer a method for rapid prototyping, that will generate application code for designers using MATLAB. Finally, ESTEREL Technologies offers a model based development environment, SCADE, that specifically focuses on the design, verification and deployment of safety critical, real-time systems.

The core design principle in time triggered technology is the notion that communication **only occurs on the network based on a predetermined schedule**. Systems on the bus (hereafter referred to as "nodes") contain an independent copy of this communication schedule that allows each node to "know" who is communicating on the bus. This allows for very complex and efficient means of error checking, error correction and reliability. In the time triggered protocol, the core unit within a schedule is the **TDMA round**. Within a TDMA round, a node is allowed to send data on the bus once and only once. More importantly, data sent by a node can only ever be viewed once every TDMA round by each of the other nodes. In other words, if the length of a TDMA round is 1000 microseconds, then the fastest update rate a sensor could achieve is 1 update per 1000 microseconds. Nodes do not have to communicate during every TDMA round, however, so depending on the needs of your application, data could be sent once a

round or once per a multiple of rounds. This leads to the notion of the **cluster cycle**. A cluster cycle is a series of TDMA rounds that, taken together will define a schedule for all data sent across the bus. In the following example, we have four nodes in the system. Each TDMA round is 1000 microseconds. Message 1, sent by nodes 1 and 2, needs to be sent every 1000 microseconds. Message 2, sent by node 3, needs to be sent only one every 2000 microseconds. Finally, Message 3, which is sent by node 4, needs to be transmitted every 4000 microseconds. The example schedule is shown in Fig. A1.

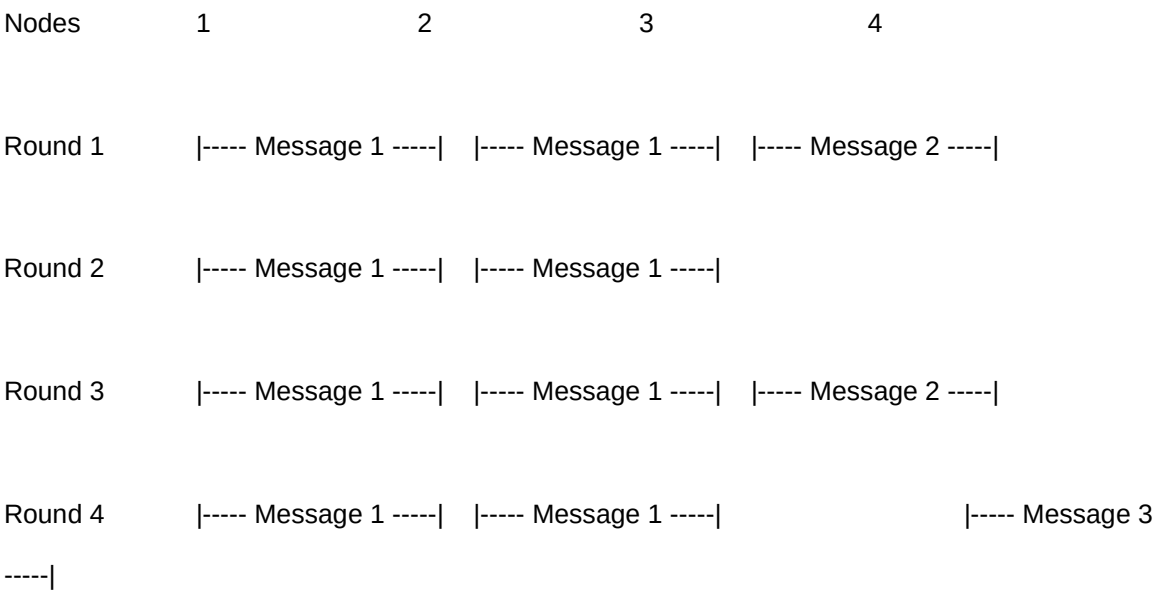

Fig. A1. Example Schedule of a Time Triggered System

__Please Note__ A new version of Message 1 is not sent twice during each round. Think of the message from Node 2 as a backup of Node 1's message. Each node could be connected to

sensors place at nearly the identical location to ensure that data from that location be received, even in the event of hardware failure of a single sensor.

The primary structure of a TTP application follows an object model. The entire system is comprised of nodes. Each node will communicate across the bus via messages. Each node will be running one, or more than one, subsystem that produces and consumes these messages. Each subsystem is comprised of one, or more than one task. Each task contains the user generated applications (reading sensors, making calculations, and so on). Keep this object model in mind, while we discuss the applications used during development.

A2. Developing Applications for the TTTech Cluster

There are three primary means of developing applications for a TTTech system. The first method, which is the most commonly employed, uses the tools provided by TTTech to create a cluster database and schedule, then develops node databases for each node in the system. Next the developer generates the FT-COM layers for each node, and finally the target application for that node. TTP-Plan is the first tool used in this process. With it, the architect of the system can outline when each node should communicate on the bus, the size of those messages, how long each cycle should be and many other parameters that define the interaction between all the nodes. Essentially, TTP-Plan creates a schedule and the subsystems in the object model. Once completed, the architect will generate a cluster database and final schedule for all the nodes. The cluster database is then passed onto each subcontractor for use in configuring the nodes with TTP-Build. A developer will specify each of the messages used on that node during a given application mode. Each message will be associated with a task and the parameters for that task will be defined. Once completed, TTP-Build will generate the FT-COM layer that handles all the bus communication. Finally, the developer will move on to their development environment of choice to create the main application code. Here, the developer will use the TTP-OS API to

interface with the bus for sending and receiving data. Compilers and linkers will need to be micro controller specific in order to generate final binary executable. For the development cluster, make files will be provided with instructions as to their use.

The second and less frequently used method for development is rapid prototyping using MATLAB, Simulink and the Real-Time Workbench. TTTech has provided a number of tools that allow an experienced MATLAB user to develop a model and then output the necessary cluster database, node databases, FT-COM layers and application code, all from within MATLAB. The advantages to this approach are quicker application development and greater flexibility when creating and testing your application model. However, this approach does not allow the degree of fine tuning needed on some systems in order to achieve peak performance. It should be noted, that the two processes are performing the same operations. The tools within MATLAB add another layer of automation to decrease development time. The description of the third method, SCADE, is left for a later date.

Appendix B. DemoCluster Application Quick Start Guide

The goal of this documentation is to assist in getting an application created, from scratch, that will run on the TTTech Development cluster. Please note that this does not go into depth regarding the API available in TTP-OS, but illustrates the minimum necessary code needed to get a program to compile, link and load into the Development Cluster.

1. Please unpack the included zip file (root.zip) into your project directory. It will unpack with the following structure (see Fig. B1):

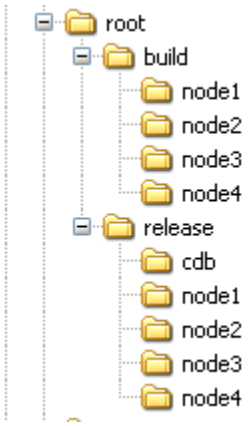

Figure B1. Directory Structure of Build Tree

2. When you create your plan in TTP-Plan, you should save the cluster database file (.cdb extension) into the root\release\cdb directory. During TTP-Plan, when you create the schedule, a new subdirectory will be created in the cdb folder. This folder will be named <application>.ddb, where <application> is the name of your cluster. For example, the following structure is created when you save a plan for a cluster named "test" (see Fig. B2):

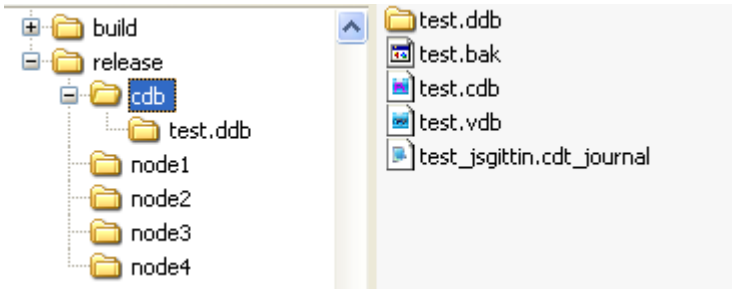

Figure B2. Cluster File Structure

4. As you can see, the test.ddb folder is present (and was created by TTP-Plan). We also have the cluster database file for the test cluster (test.cdb).

5. As for the rest of the file structure, your individual node database filed created in TTP-Build will be saved to their respective node folders under the release directory. So Node1 would be saved to root\release\node1. When you generated the FT-COM code at the end of the TTP-Build process for a node, it will place the header and source files in the same directory of the node database file (see Fig. B3):

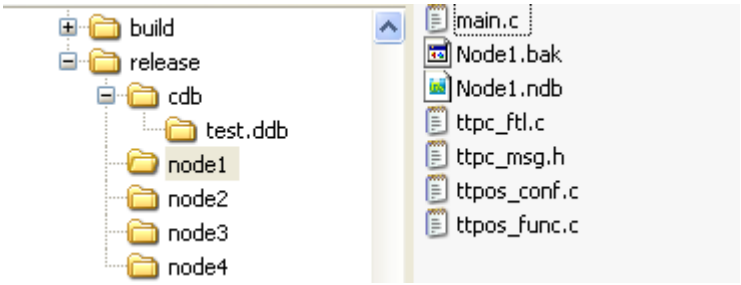

Figure B3. Node file structure

6. The make files needed for the project are located in the root\build directory. There are 4 make files used during compilation and linking (see Fig. B4):

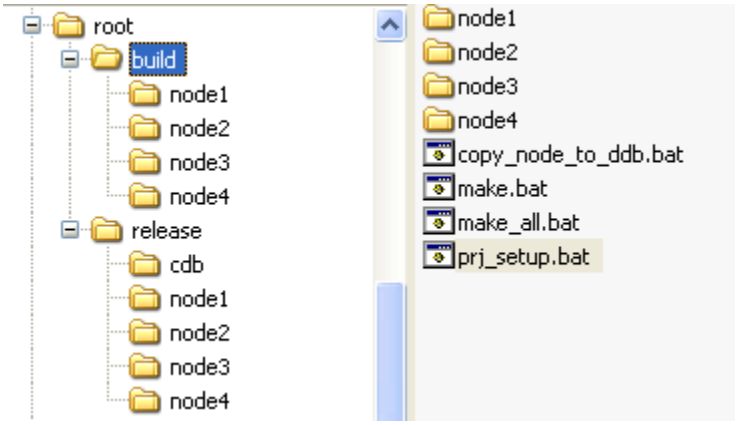

Figure B4. Make files

7. If you maintain the directory structure as outlined and save your cdb and node database files as listed, the only file you need to edit is prj_setup.bat. Once you have named and saved your cluster database file, you must set the CLUSTER_NAME variable to the name of your cluster. This is used during the final copy of your executable. The location to edit is highlighted in red in Fig. B5:

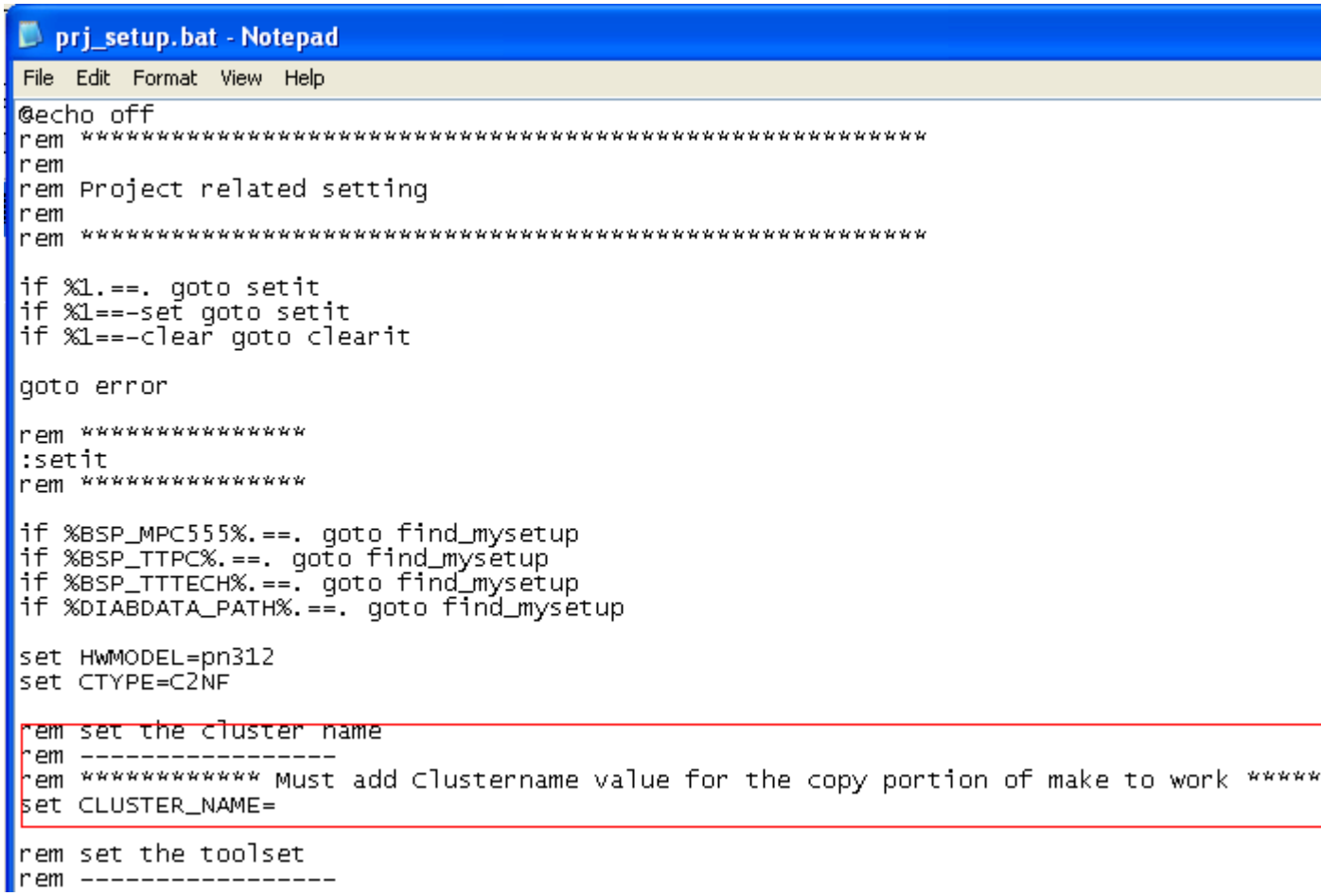

Figure B5. Project setup batch file

8. Once you have generated your FT-COM code for each node, you will need to write your application code for each task in your cluster. This is done in the main.c file within each of the node folders. A main stub file has been supplied for you within each folder. It contains instructions to create the minimum code needed to compile. The instructions and comments will show you where to add code for each task in your cluster.

For TTP-OS API help, please see the TTP-OS manual in

Start->TTTech->TTP-OS-mpc555_AS8202NF-4.11.24->User Manual.

For specific instructions on the MPC-555 chipset, please see the manual in

Start->TTTech->IO Toolbox-MPC555-3.2.1->User Manual.

9. Once you've completed the code in each of your main.c files, you are ready to compile the executables.

Click Start->Run...

In the dialog box type in "cmd".

This will open a command line prompt. Navigate to your root installation and then go to the root\build folder. Type "make_all.bat".

If no errors were reported during compilation and linking, you should be ready to load your application into the development cluster. All of your application files will be in the <application>.ddb folder. See Fig. B6:

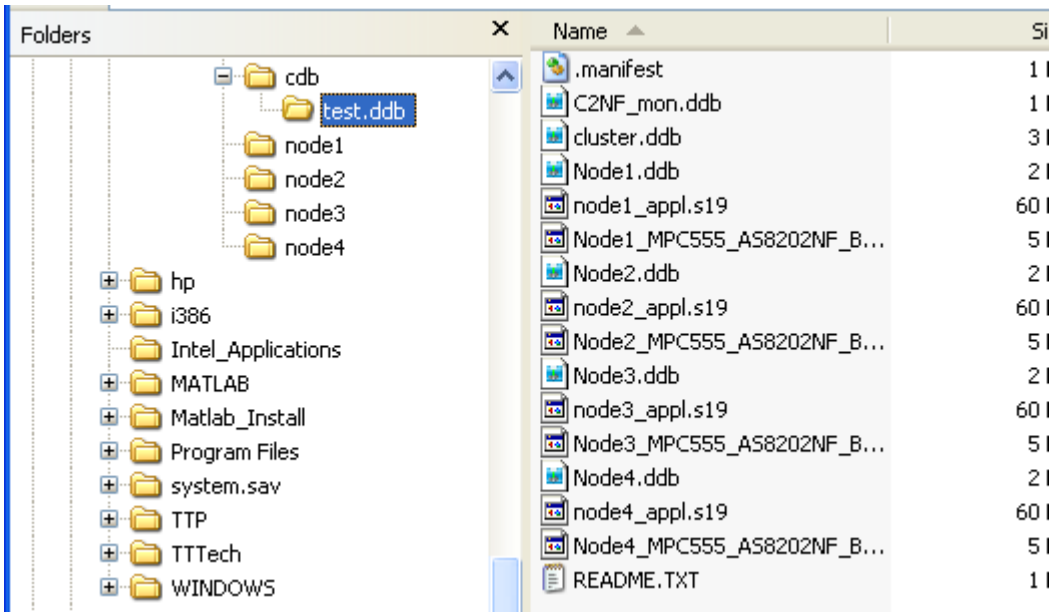

Figure B6. FT-COM files

10. The .s19 files are the executables that will be uploaded into the cluster using TTP-Load.

This is covered in the TTP-Load quickstart appendix

Appendix C. TTTech TTP Plan Quick Start Guide

1.Start TTP-Plan

Start Button -> All Programs -> TTTech -> TTP-Plan-5.5.7 -> TTP-Plan

2. TTP Plan will start in Guide Mode. If it is not in Guide Mode, click the Guide Tab at the top of the page (see Fig. C1):

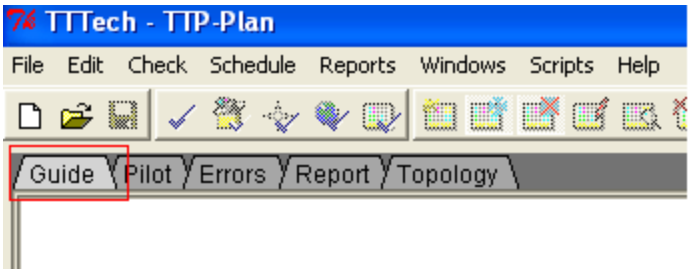

Figure C1. Guide Tab

TTP Plan Guide mode is a guided tour through the definition of a TTP cluster. It leads you through all the steps necessary to create a cluster design. At each point in the design, you can click Next to continue to the next step, Back to return the previous step, and Edit to change values for the current step. Take a moment to save your new cluster definition. This folder will then be the target for all auto generated files within TTP Plan.

3. Click Next and go to the Cluster Step.

4. Click Edit. This will bring up a dialog box where you input the cluster name. Enter a name for your cluster and click OK (see Fig. C2).

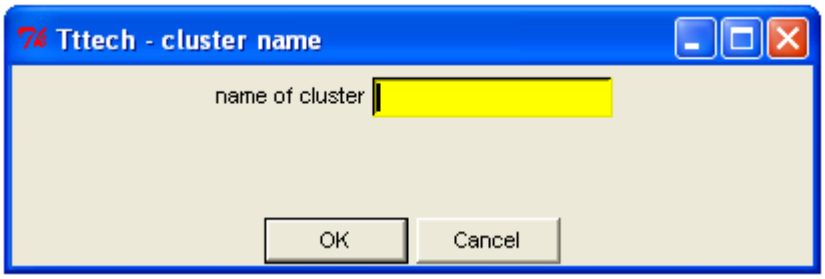

Figure C2. Cluster name dialog

5. Once you've clicked OK, another dialogue box will open (see Fig. C3).

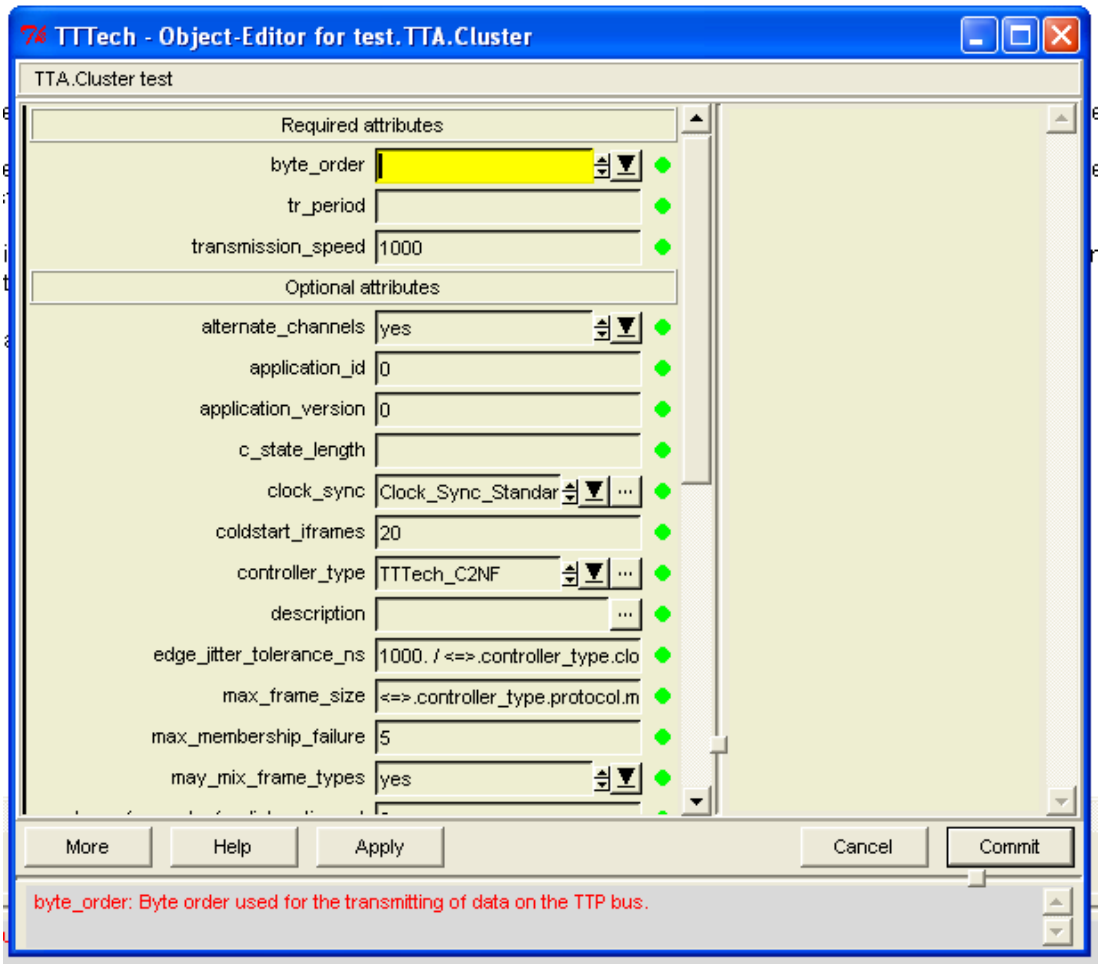

Figure C3. Object editor for cluster

6. Three values are required to continue (byte_order, tr_period and transmission_speed). See page 6 of the TTP Plan Documentation for more information.

Byte_order - for the purposes of creating a test cluster, 32bit big endian (big_32_endian) is sufficient, but your application may require an alternative byte ordering. For a brief overview of endianness, see the following (http://en.wikipedia.org/wiki/Endianness).

tr_period - TDMA round period. The value you place here specifies the lenght of the TDMA round (in microseconds). This value, multiplied by the number of rounds in a cluster, will give you the overall time period of a cluster.

transmission_speed - is the data transmission rate over the network. The value is in kbit/sec, so a value of 1000 is roughly 1Mbit/sec. The value you place here must be allowed by the TTP controllers in use. For the development cluster, a value of 2000 is recommended.

7. For all additional parameters, TTP Plan will insert average default values. If you wish to change these values, please consult the TTP Plan manual (page 56). Click commit when you've entered in the required values. The dialogue will close. A new window will open that will allow you to add more cluster definitions or alter an existing one (see Fig. C4). Click close to continue.

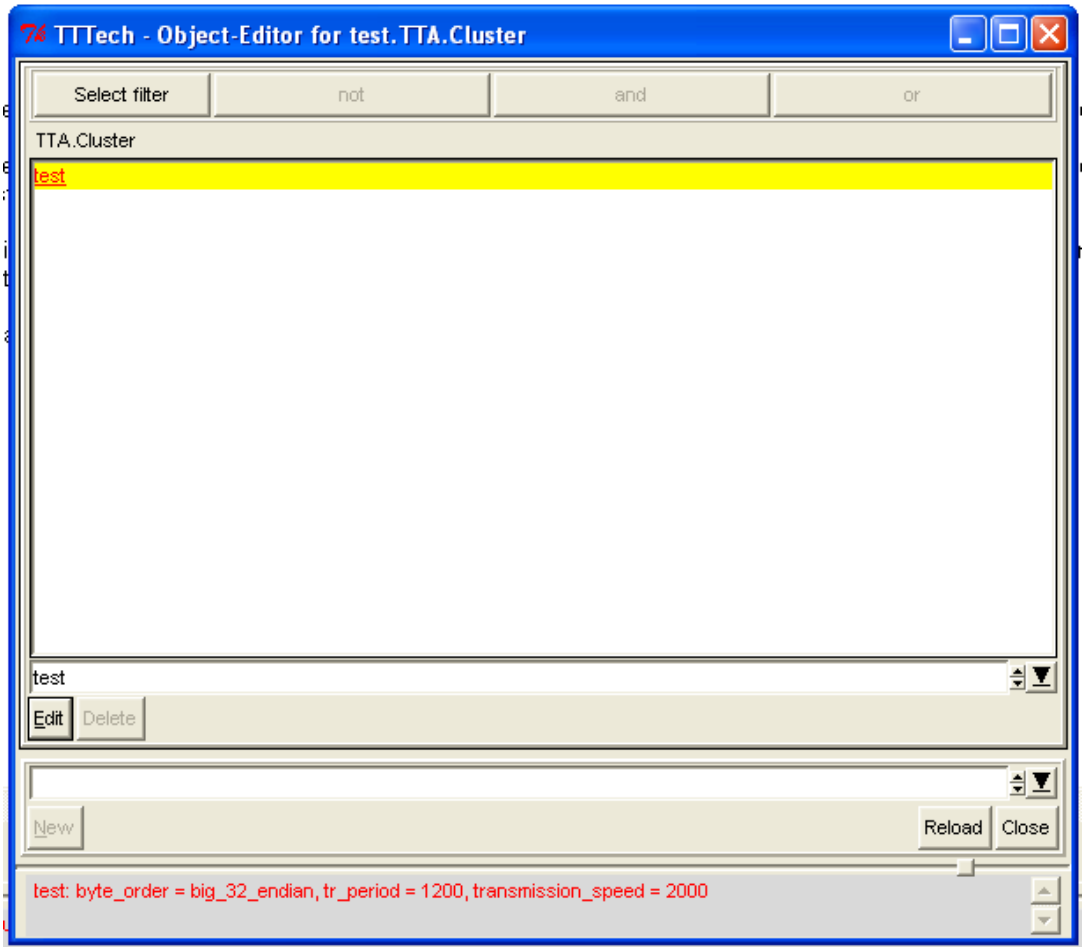

Figure C4. Cluster list

8. Click the Next button in the guide to continue to the next page.

9. The TTA.Host screen is where you will configure your TTP controller nodes (4 are provided in the development cluster). Click the Edit button to begin.

10. The dialogue box that opens allows you to edit existing node configurations and add new ones. When starting a brand new cluster, the dialogue box appears in Fig. C5:

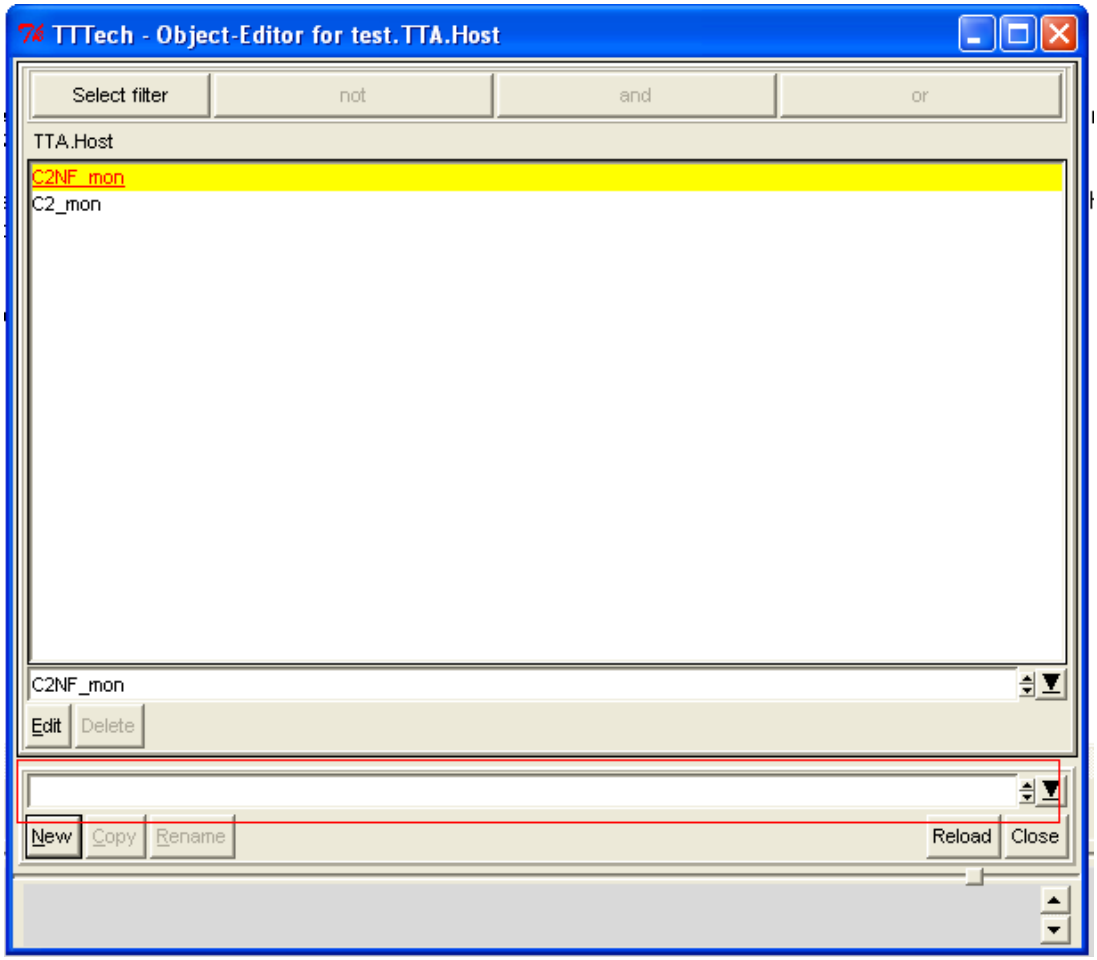

Figure C5. Host list

The C2NF_mon is the configuration information for the monitoring node included in with the development cluster. To add a new node, enter a name for the node in the highlighted textfield and click New.

11. Once you've clicked new, the node will appear in the upper list below the C2_Mon entry. Highlight your node in the list and click the edit button (see Fig C6).

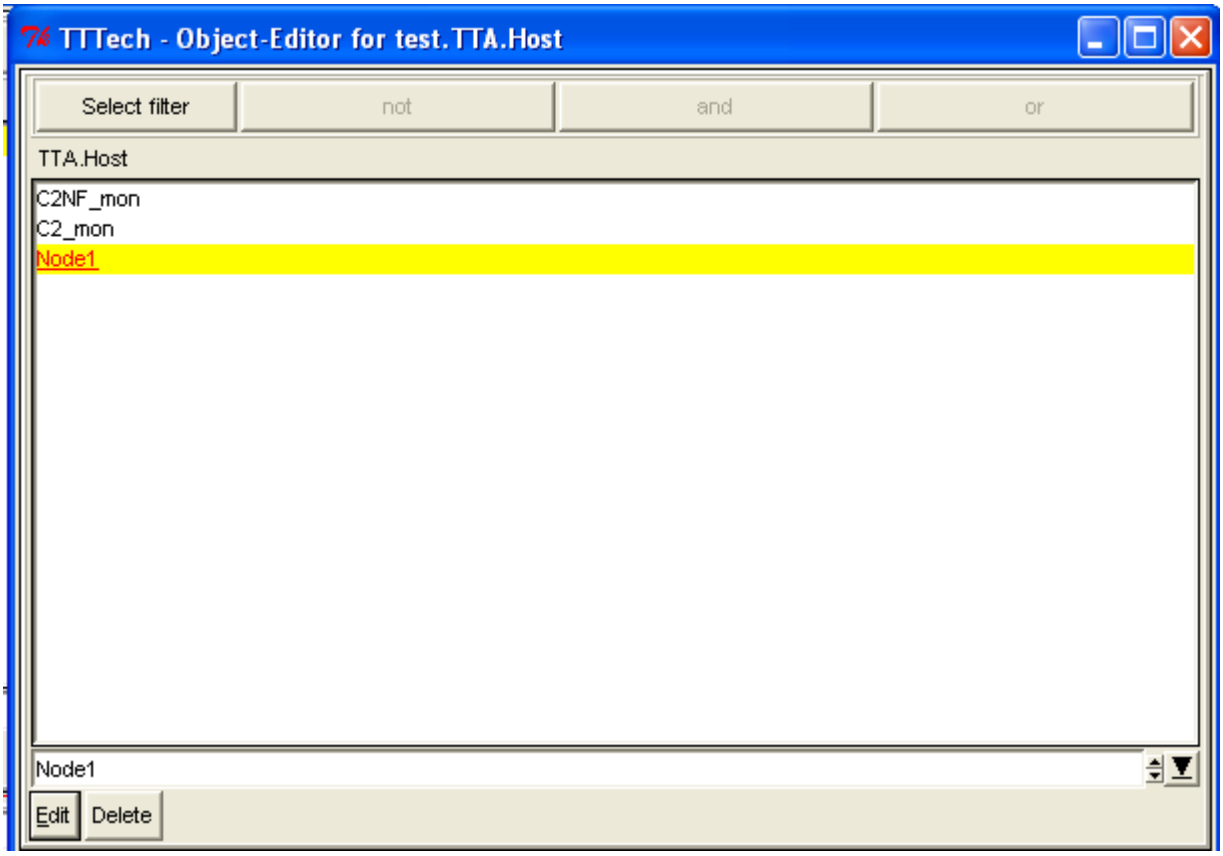

Figure C6. Host list with use created node

12. A new dialogue box will open and you will be allowed to enter values for this node (see Fig. C7).

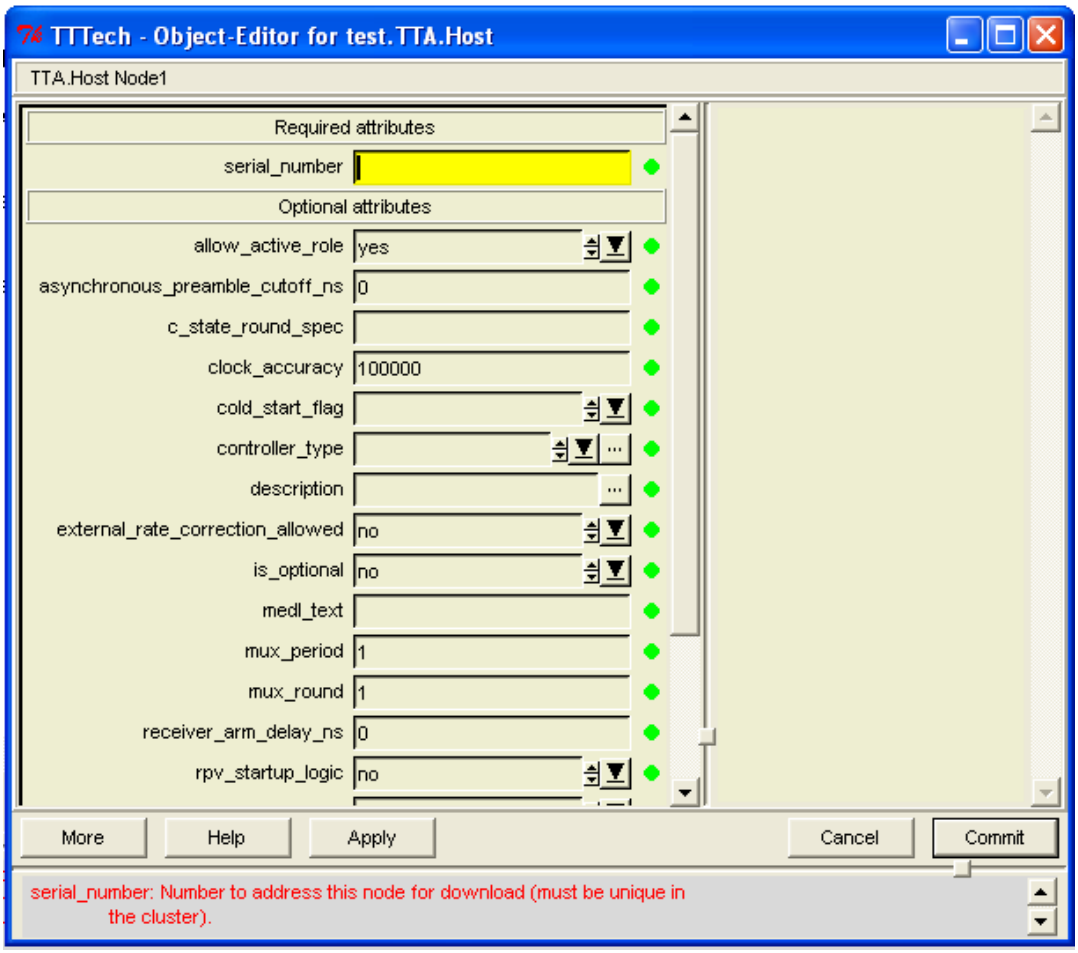

Figure C7. Host properties dialog

13. The only required value is a unique serial number that you assign to the node (page 8). Later in the build process, you will match this serial number to a specific node in the development cluster. Click the Commit button when you've entered a serial number.

14. Repeat this process of creating new nodes and entering in serial numbers until you have the 4 development cluster nodes defined (see Fig C8).

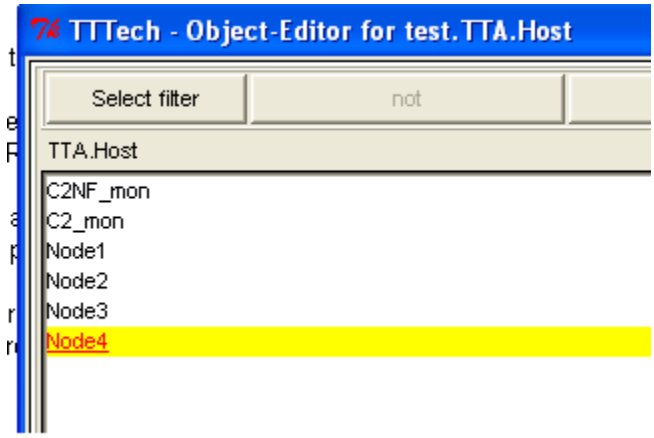

Figure C8. Host list with all four nodes

15. Click close to return to the Guide and then click Next to continue.

16. The TTA.Host_runs_Subsystem_in_Cluster_Mode screen will allow us to define what job will be run on what host. For example, in the demo application, Nodes 1 and 2 both run counter programs to increment different counter values, while Nodes 3 and 4 are defined to read those values and update their LED's.

17. Click the Edit button to begin assigning subsystems (jobs) to hosts (nodes). You will open the following dialogue shown in Fig. C9.

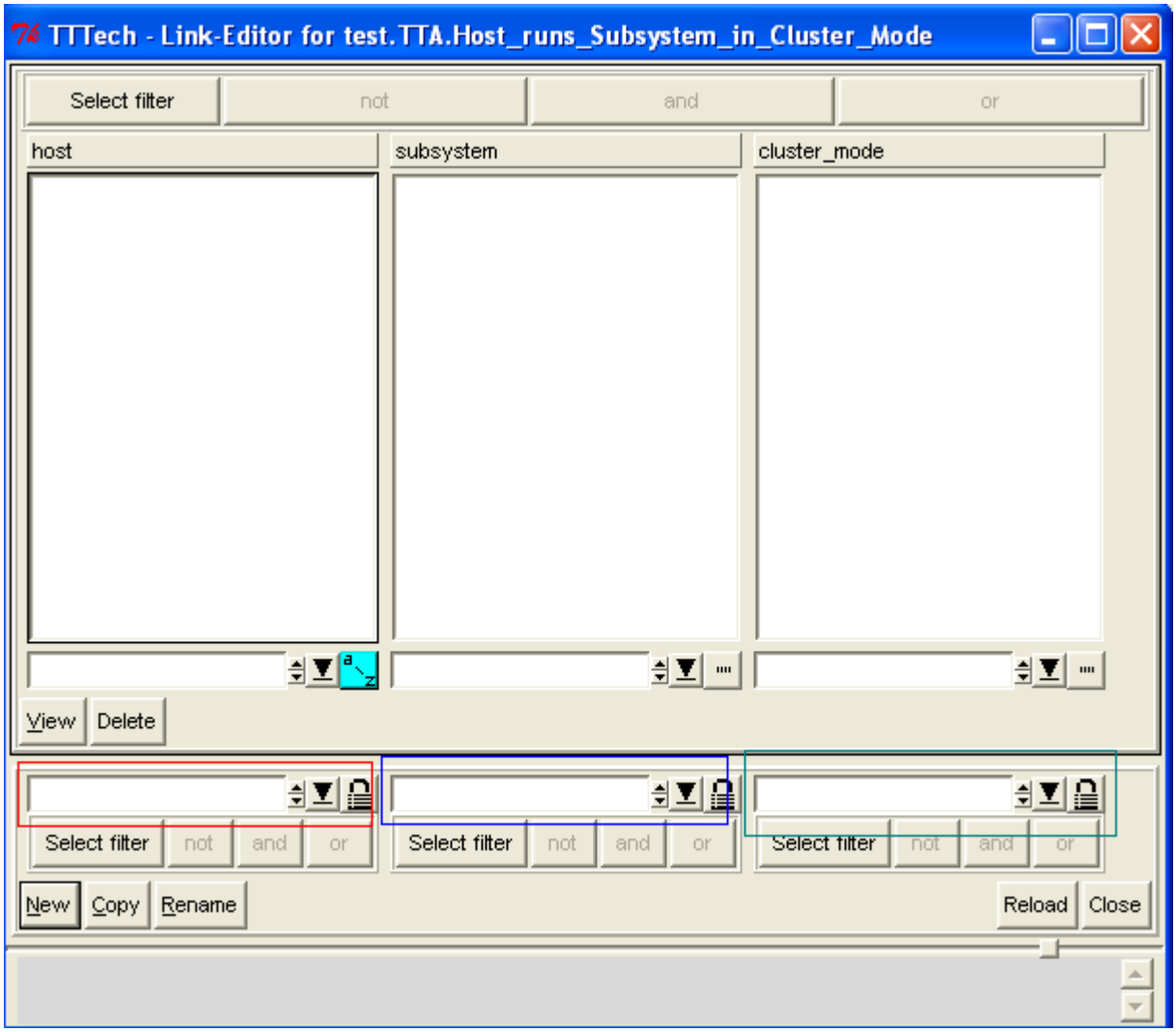

Figure C9. Host, subsystem and cluster mode link dialog

18. We've defined hosts in the previous page, but we haven't defined any subsystems or cluster modes yet. To start, you will select an existing host via the dropdown box highlighted in red. Next, type in the name of a new subsystem within the box highlighted blue. Finally, type in the name of a new cluster mode within the box highlighted green. Once you've filled out all three fields, click the New button at the bottom of the dialogue box. This will create a new link between one of your nodes, a subsystem and a cluster mode. For more infomation, see page 9 in the manual. An example of a possible layout is as follows in Fig. C10:

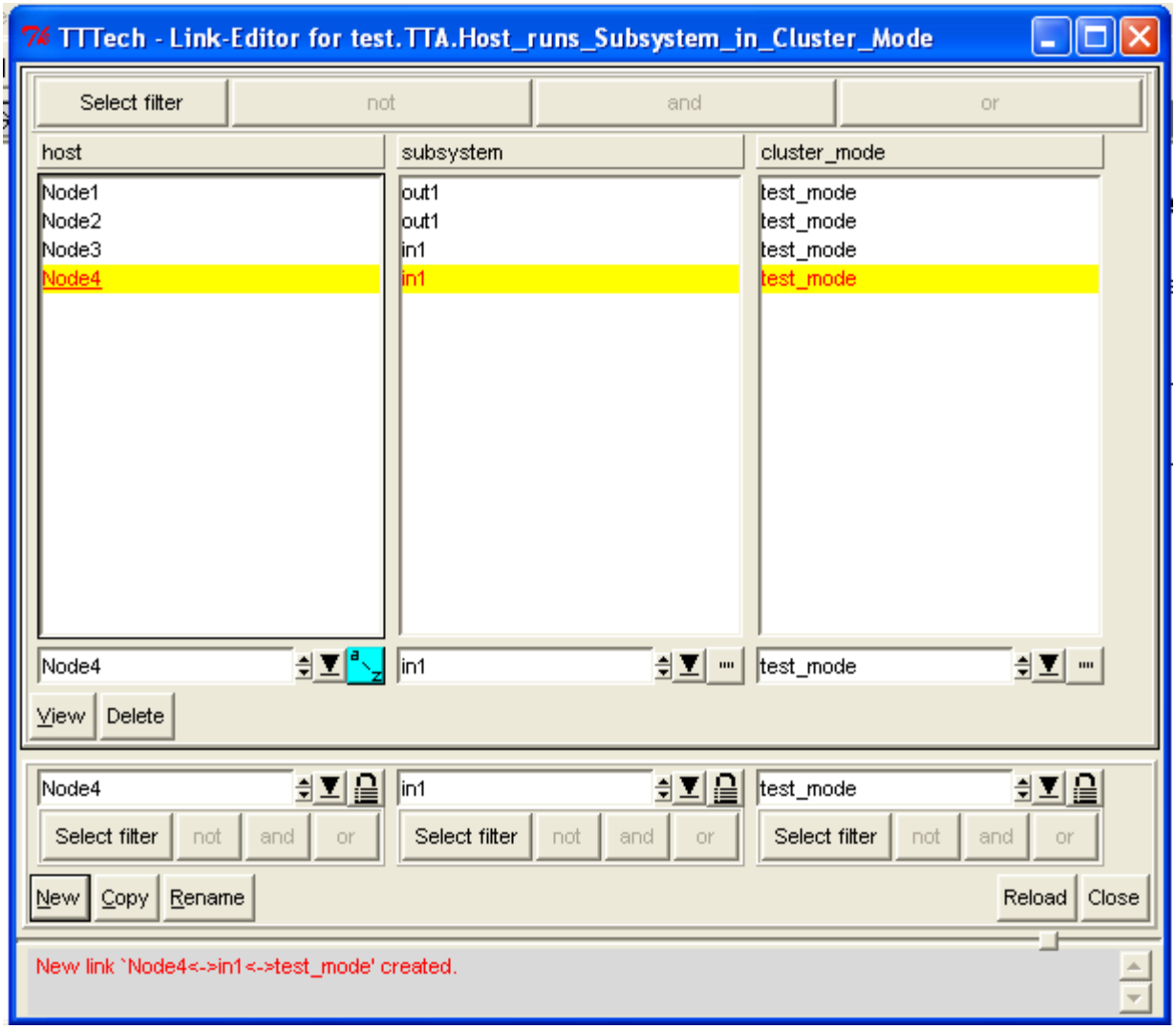

Figure C10. Example host, subsystem and cluster mode links

You can delete exisiting links by highlighting a node in the upper host list and clicking the delete button. Alternatively, if you would like to rename a subsystem or cluster mode, you can highlight it within the appropriate list and click the rename button. Click close on the dialogue when you've created all your subsystem links.

19. Click Next within the Guide to continue to the TTA.Subsystem_sends_Message page.

20. This page allows you to link subsystems and the messages that the subsystem will send. You do not specify message to be recieved, since each subsystem has access to every frame. Click the Edit button to open the dialogue box show in Fig. C11.

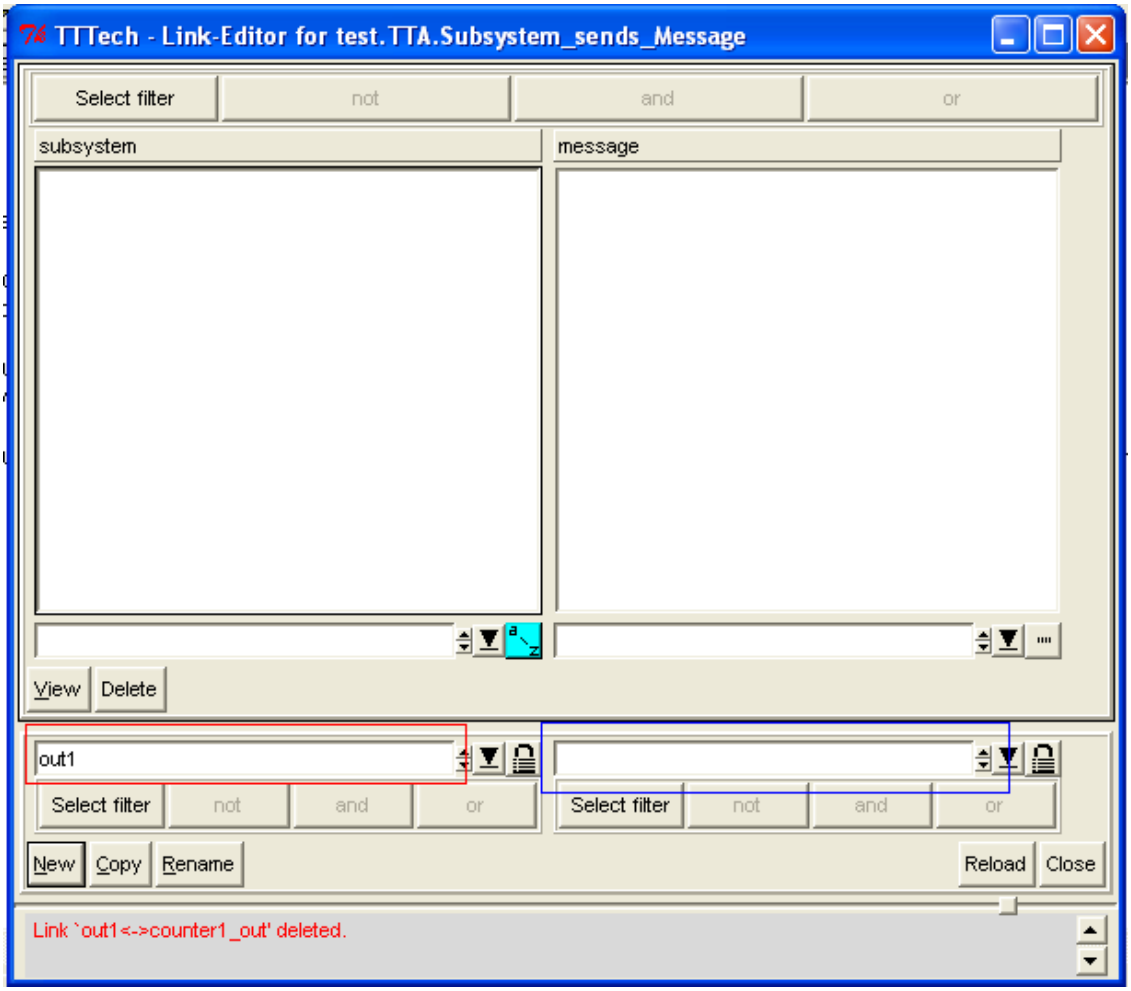

Figure C11. Subsystem and message link editor

21. In the red highlighted box, select a subsystem, and in the blue highlighted box, enter in the name of a message that subsystem sends. Click New to create the link. This will add them to the subsystem and message lists at the top of the dialogue. Please note that each message is sent by exactly one subsystem but each subsystem may send as many messages as necessary. For more information, please see page 11 in the manual. When you have created your links, you should have something similar to the following example. In this example, sub1 sends two messages (msg1 and msg2) while sub2 only sends one message (msg3). See Fig. C12.

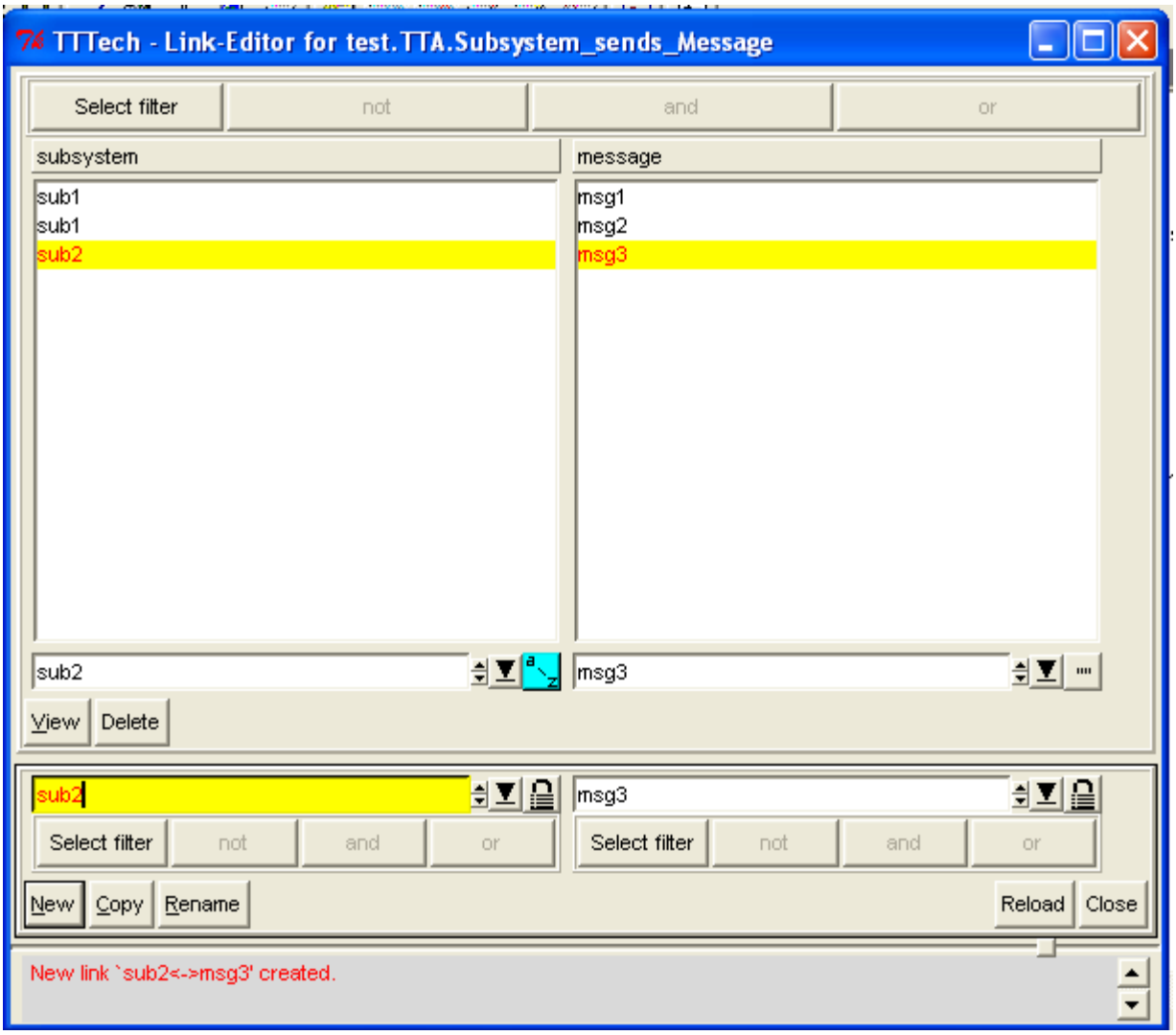

Figure C12. Example subsystem, message links

22. Click close when finished and then click Next in the Guide to continue on to the TTA.Msg_Type_P page. This screen will let you define what types of data will be carried in each message you created in the previous page. You won't be linking your message to a type, however, during this step. You will only be defining what types of data your messages will carry. See page 12 for more information.

23. Click the Edit button to open the following dialogue box. You are presented with a number of default types as well as the opportunity to add your own (see Fig. C13).

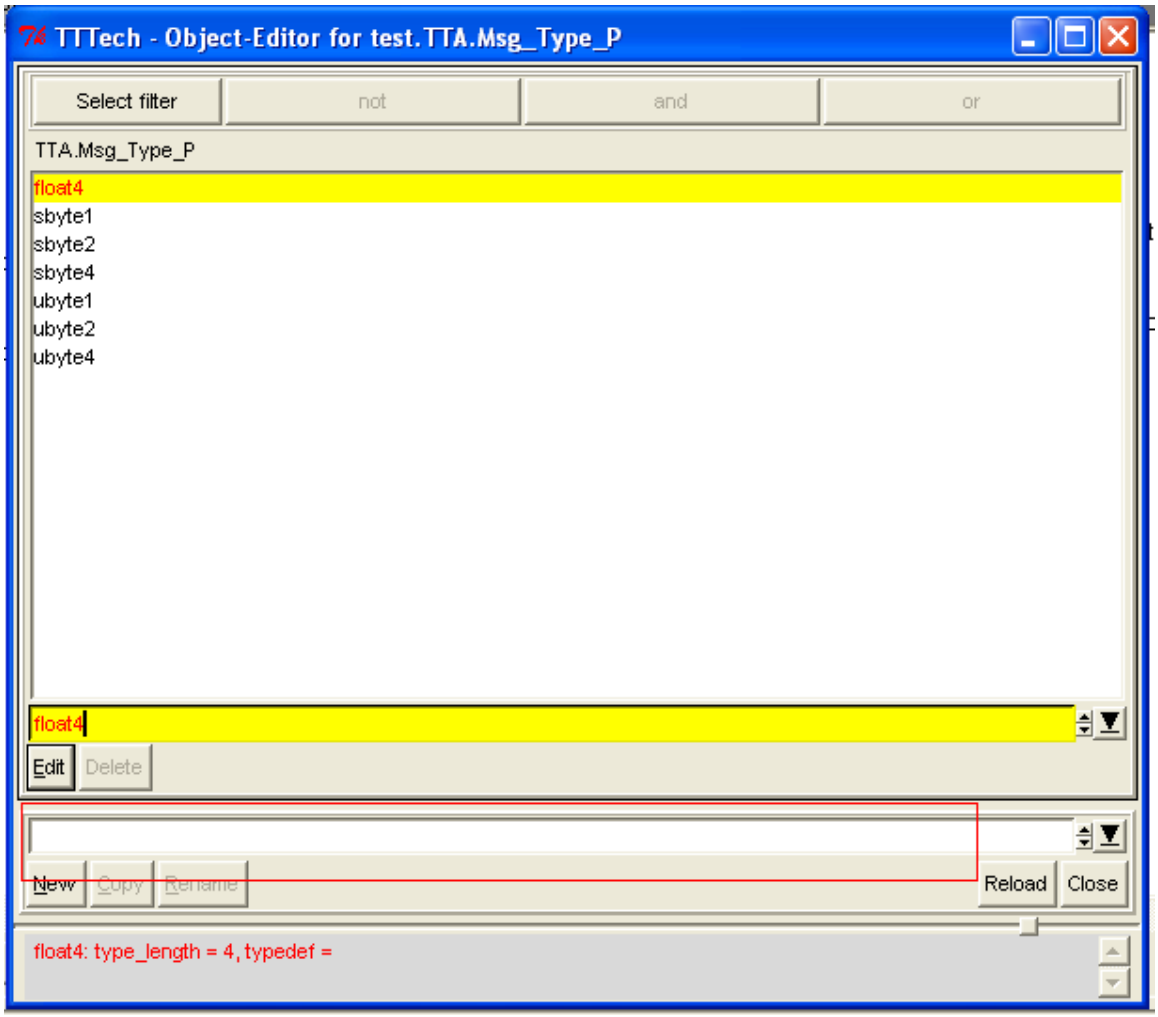

Figure C13. Message types

24. If you do not wish to use one of the predefined types, enter a new name for your type in the red highlighted text field and click new. Then highlight the type in the above list and you can edit the parameters of your type as shown in Fig. C14:

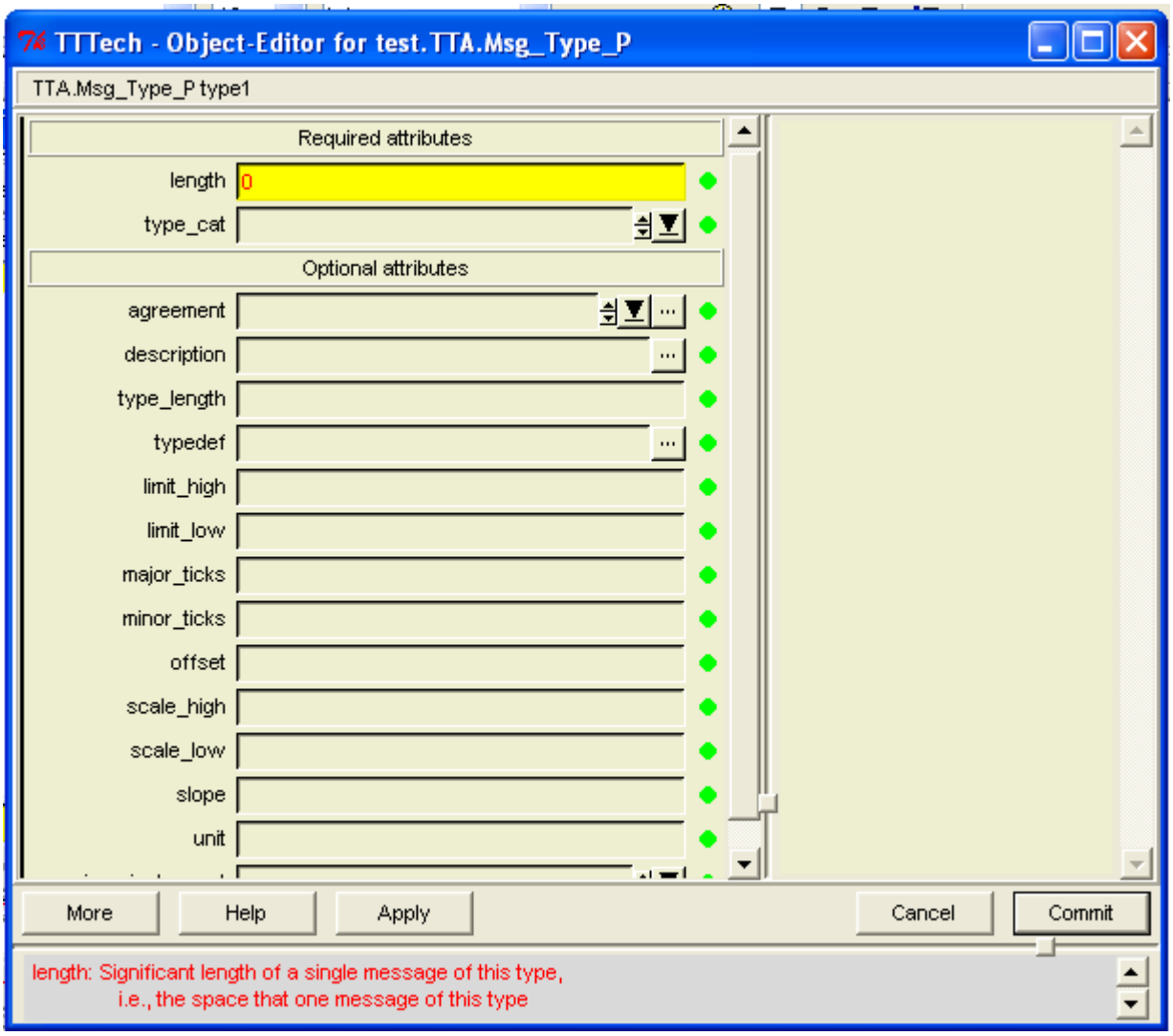

Figure C14. Property editor for message type

25. The required fields are length and type_cat (more information for required values can be found on page 52, optional values page 72):

length - specifies the size of these messages in bits (you can also specify bytes or

combinations of the two. Please see the manual for the formatting).

type_cat - is either INT, for integer values that can be negative, UINT for integer values that are always positive, and REAL for all real values.

26. Once you have completed the required fields, click Commit. Then Close the previous dialogue box once you've added all of your needed message types. Finally, click Next on the Guide to continue to the TTA.Message page.

27. On the TTA.Message Guide page, Click the Edit button. You are now given the option to set preferences for the messages you previously created (see Fig. C15).

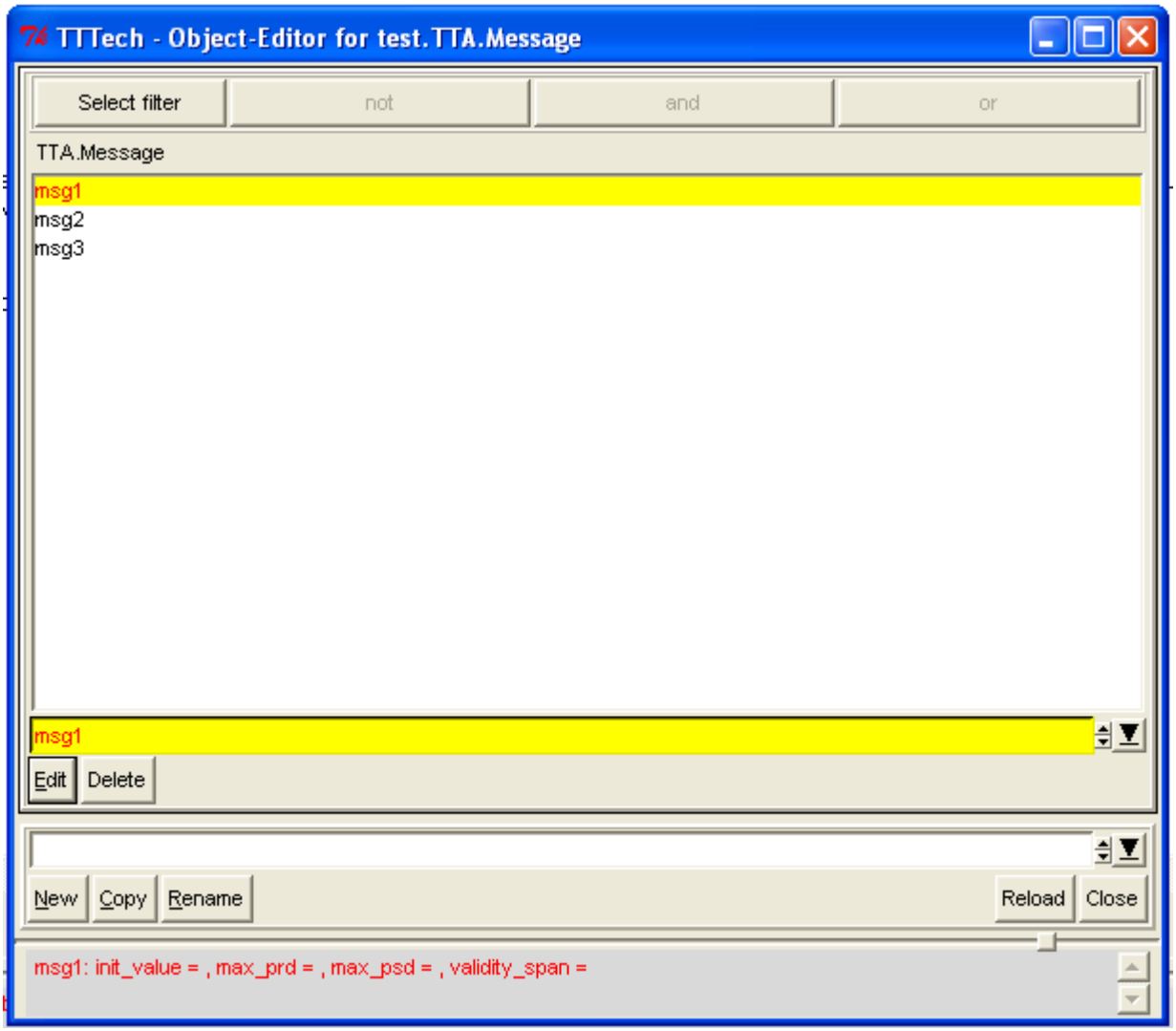

Figure C15. Message object list

28. Highlight each message and click the Edit button.

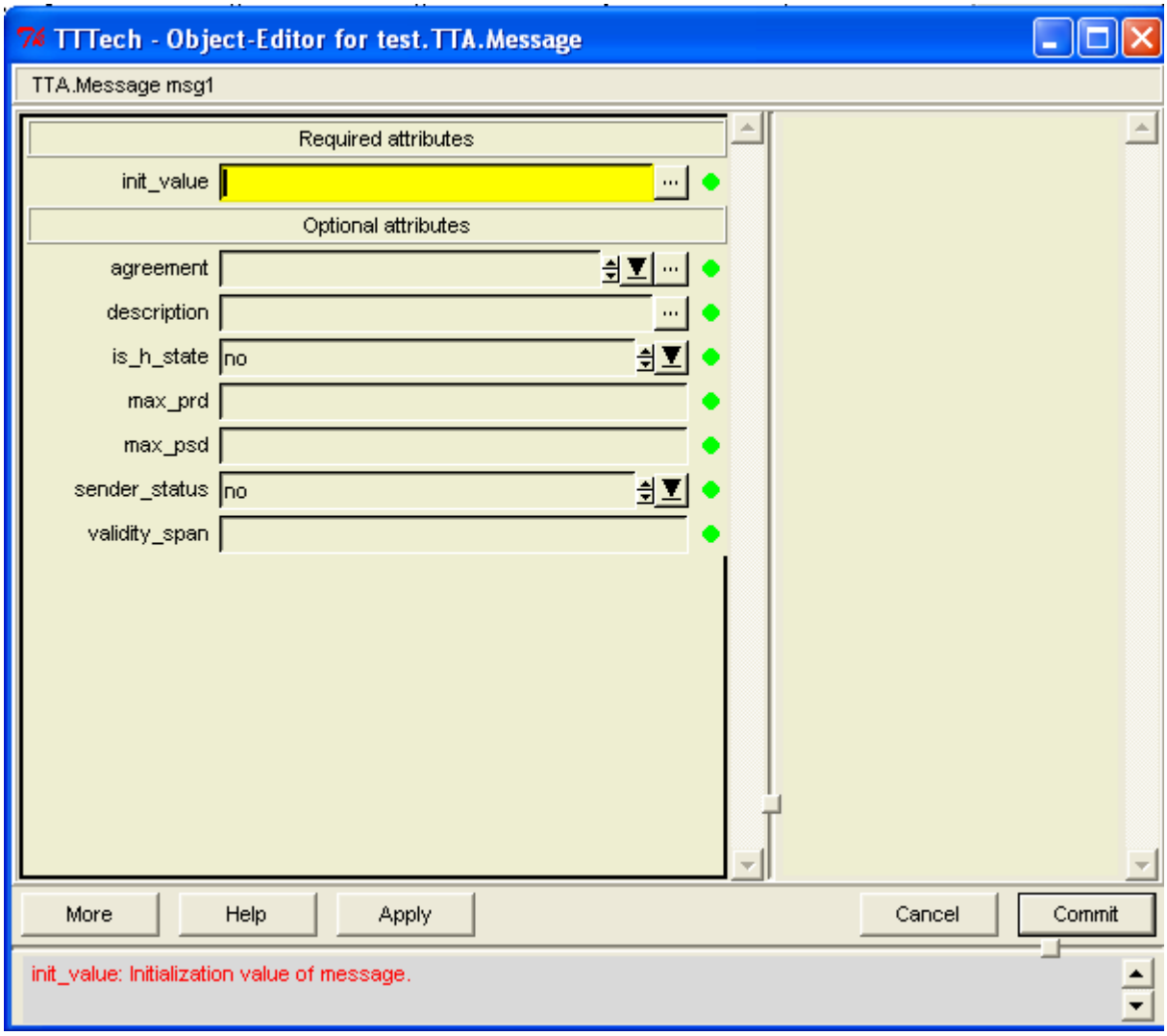

Figure C16. Property editor for a message

29. As shown in Fig. C16, the only required field is init_value. Set this according to the needs of your application. Click Commit to close this window. Finally, set the init values for each of your messages and close the messages list (required field page 12, optional fields page 74).

30. Click Next on the Guide page to continue to the TTA.Message_uses_Msg_Type page. On this page, you will set the type for each of your messages. Click Edit to open the dialogue box (see Fig. C17).

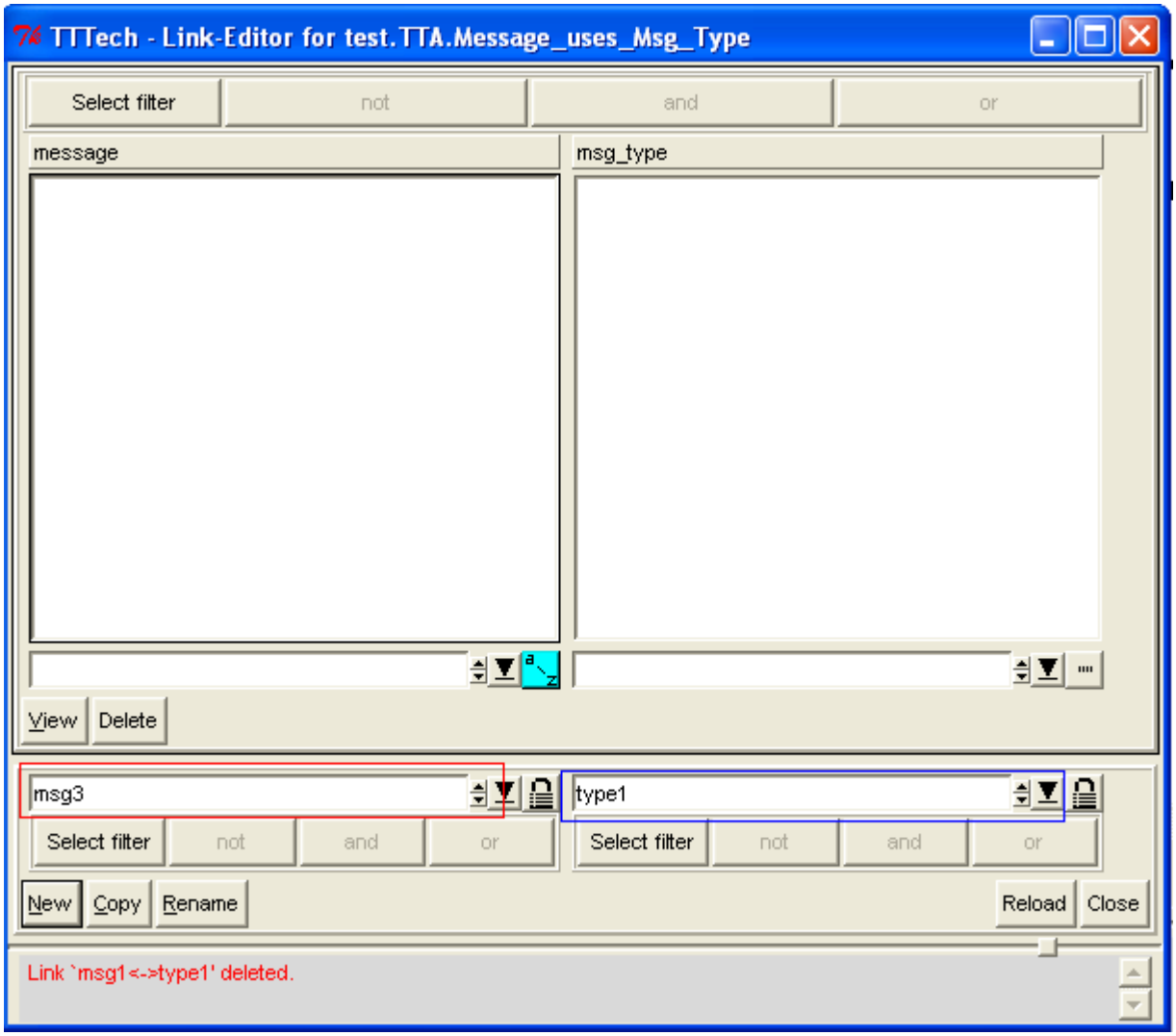

Figure C17. Message, message type link dialog

31. In the red highlighted field, select a message. In the blue highlighted field, select the appropriate type (one of the defaults or a type you previously created). Click the New button to create the link between message and type. Do this for each of your cluster's messages. Click

Close.

32. Click Next in the Guide to continue to the TTA.Cluster_Mode_uses_Message page. In this step, we will be defining how often a given message needs to be transmitted within a given cluster mode. Click Edit (see Fig. C18).

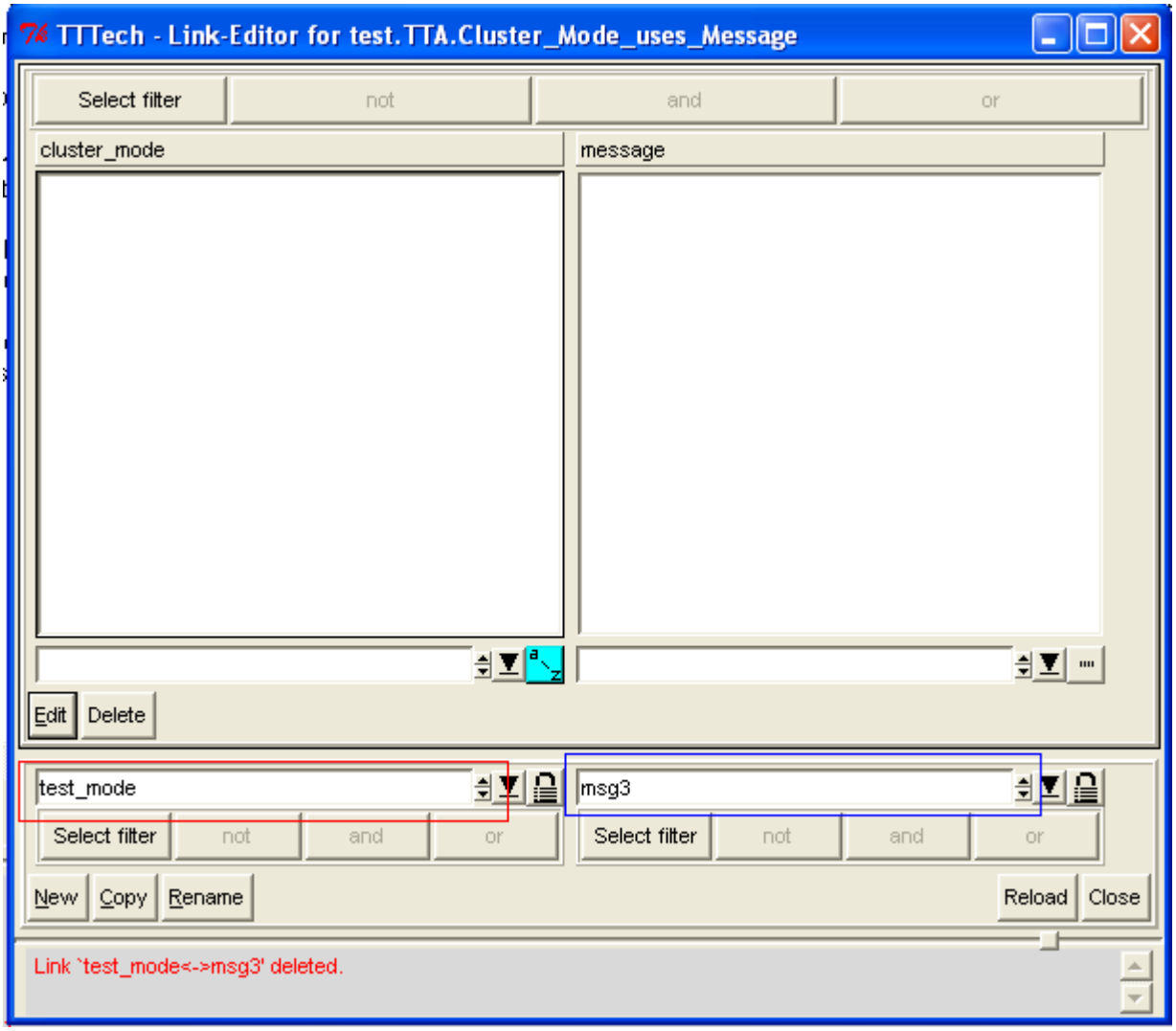

Figure C18. Cluster mode, message linker dialog

33. First select the cluster mode you want to define by selecting it within the red highlighted area. Next, select a message you want transmitted during that cluster mode in the blue highlighted area. Click New and this link will be added to the cluster_mode and message lists above. Finally, select each cluster mode, message link in the list and click Edit. You will be present with the following dialogue in Fig. C19:

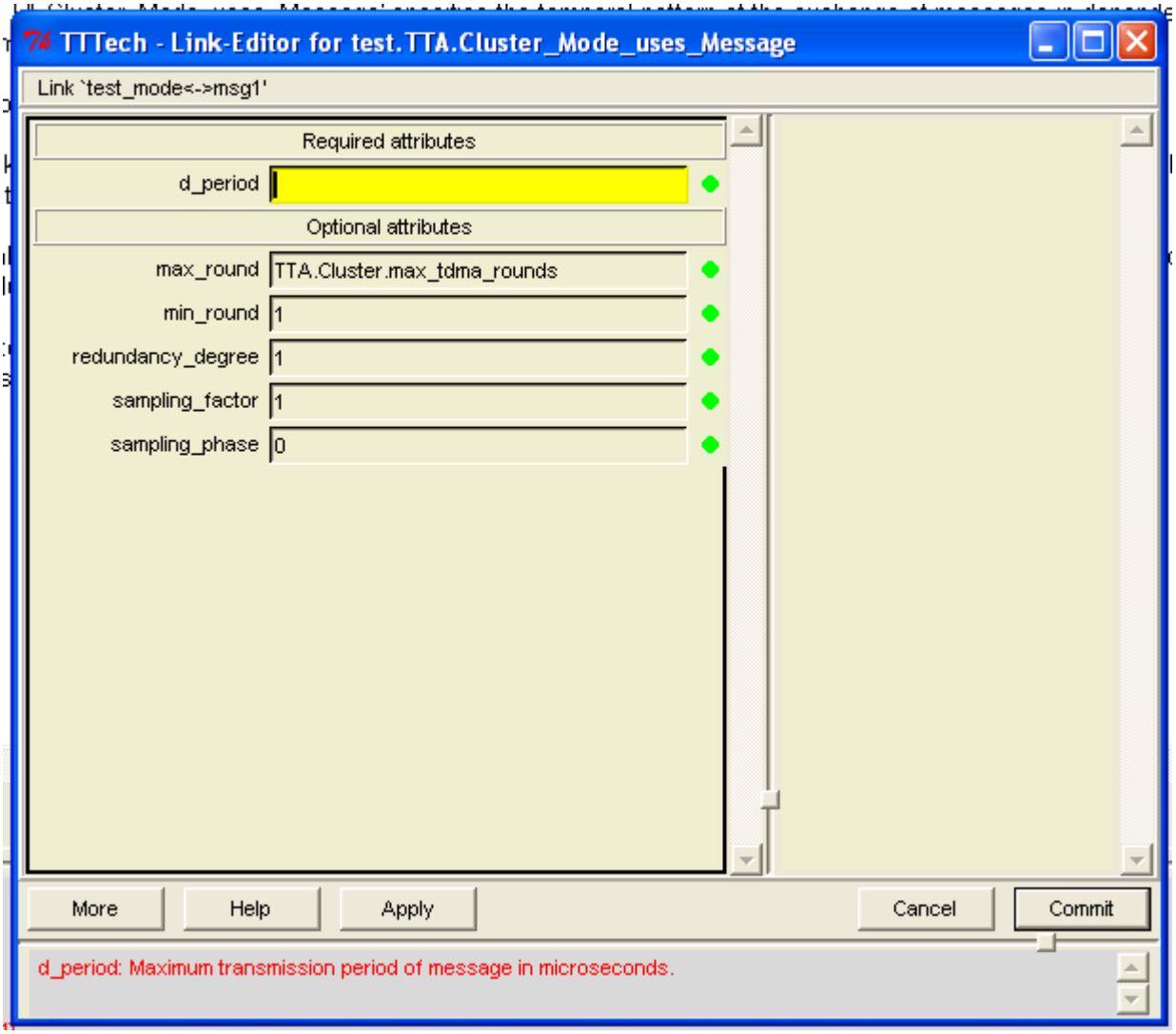

Figure C19. Cluster mode, message link property editor

34. This is where we define the frequency of message transmission. The only required field,

d_period, is the (maximum) number of microseconds that can elapse between transmission of the message (more information on page 15, optional values are defined on page 81). Enter a value and click Commit. Do this for each cluster mode, message link. Finally, click Close on the link editor dialogue box and then click Next in the Guide. You will continue to the Mode change permissions page.

35. The mode change permissions page will let you set which nodes in the system can cold start the system and which nodes are allowed to send xframes during normal transmissions. For the purposes of testing, you can select the Cheat button (where the Edit button is usually placed) to set all nodes available for cold start. Click Next.

36. You are now at the Checking for errors page. Click the Check button (again, where the Edit button has been previously placed) to check for any errors in your configuration. If you encounter any errors, you can click on the arrow to the left of the errors to expand the error type and get a description of the problem. You can then click the links provided to open the dialogue box of the specified configuration to change it as needed. When you have no more errors, click Next.

37 You are now on the Making a schedule page. To compile the plan and make a schedule, click the Schedule button. Once the schedule has compiled, you will get a text output of the results. Click Next in the Guide to move to the Edit Schedule page.

38. On the Edit Schedule page, click the Edit button to get a visual representation of your schedule. Your completed Schedule will look something like Figure C20:

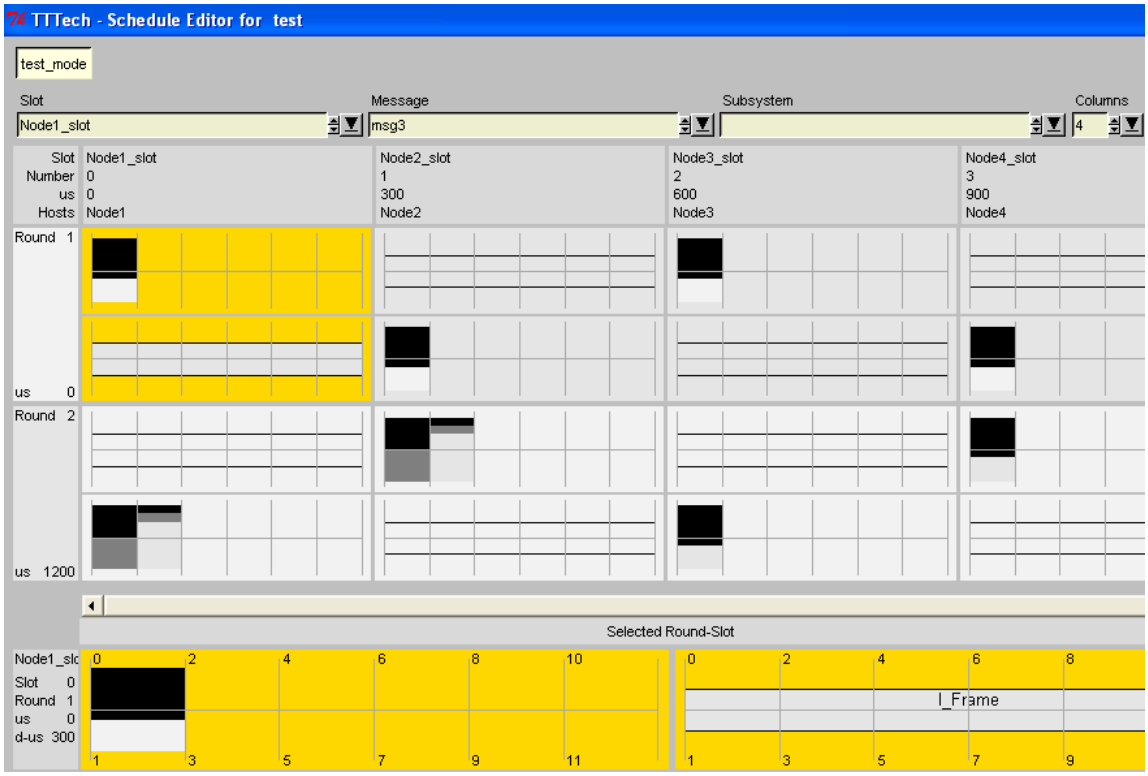

Figure C20. Demonstration schedule

39. If you would like to see a graphical representation of your schedule during any other step in process, click the Edit Schedule button at the top of the application, as shown in Fig. C21.

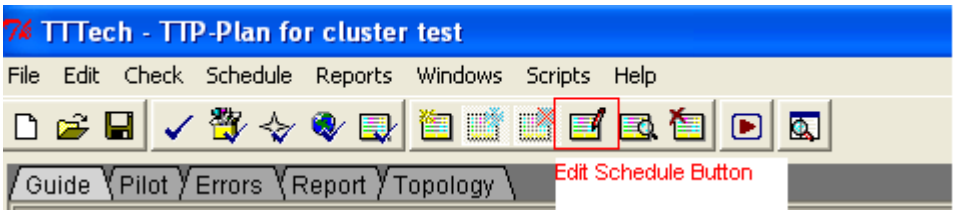

Figure C21. Edit schedule

40. Close the Schedule and then Click the Next button in the Guide.You have finished building the Plan for your TTP application. You will now need to move on to TTP Build to create the code needed to upload into the Nodes. From there you will upload the code using TTP Build. Make sure to save your plan and remember where you are saving the cluster definition. You will need it when you design the nodes in TTP Build. You can save your plan by clicking File->Save.

Appendix D. TTTech TTP Build Quick Start Guide

1.Start TTP-Build

Start Button -> All Programs -> TTTech -> TTP-Build-5.5.40 -> TTP-Build

2. TTP Build will start in Guide Mode (see Fig. D1). If it is not in Guide Mode, click the Guide Tab at the top of the page.

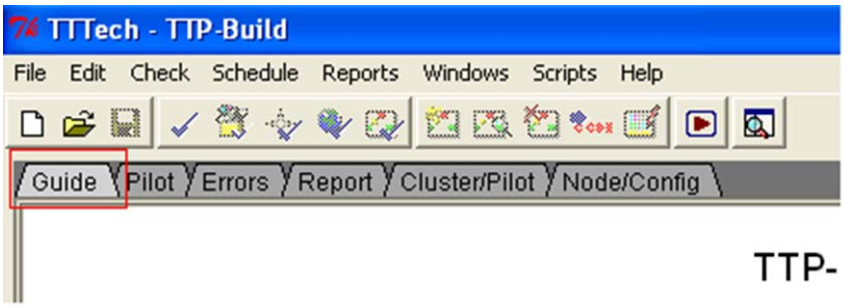

Figure D1. TTP-Build guide mode tab

TTP Build Guide mode is a guided tour through the design of a specific node. It leads you through all the steps necessary to create a node design. At each point in the process, you can click Next to continue to the next step, Back to return the previous step, and Edit to change values for the current step. Click Next to continue.

3. You will start on the Loading a cluster design page. We first need to load the cluster definition created during TTP Plan. Click the Load button at the bottom of the page. You will be presented with a browse file dialogue to locate the *.cdb file of your cluster design. Click the file and then click Open (see Fig. D2).

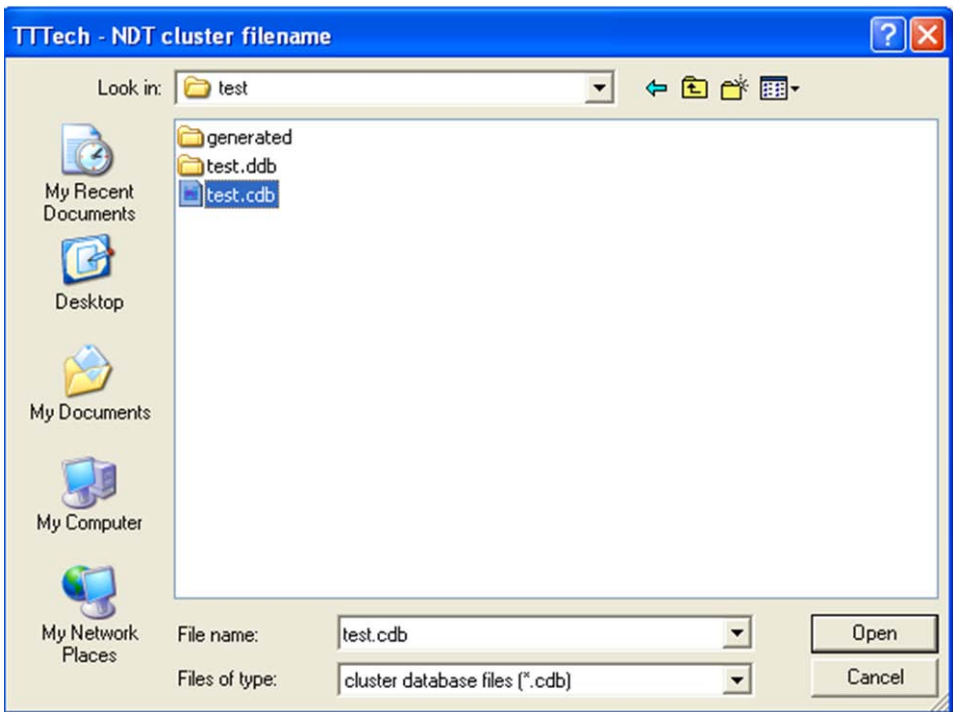

Figure D2. Open file dialog

4. Next you will be asked to select a node to design (see Fig. D3):

Figure D3. Node selection

Select a node and then press OK. At this point you should save your node. If you do not do this now, when you generate your FT-COM layer code, it will be placed in the default location, the working directory of TTP Build, which is ill advised. Once you've saved your node database to your own directory, click Next in the Guide to continue to the TTA.Host page.

5. On the TTA.Host page, you will be configuring an individual node in the cluster. Click Edit and you will open the window shown in Fig. D4:

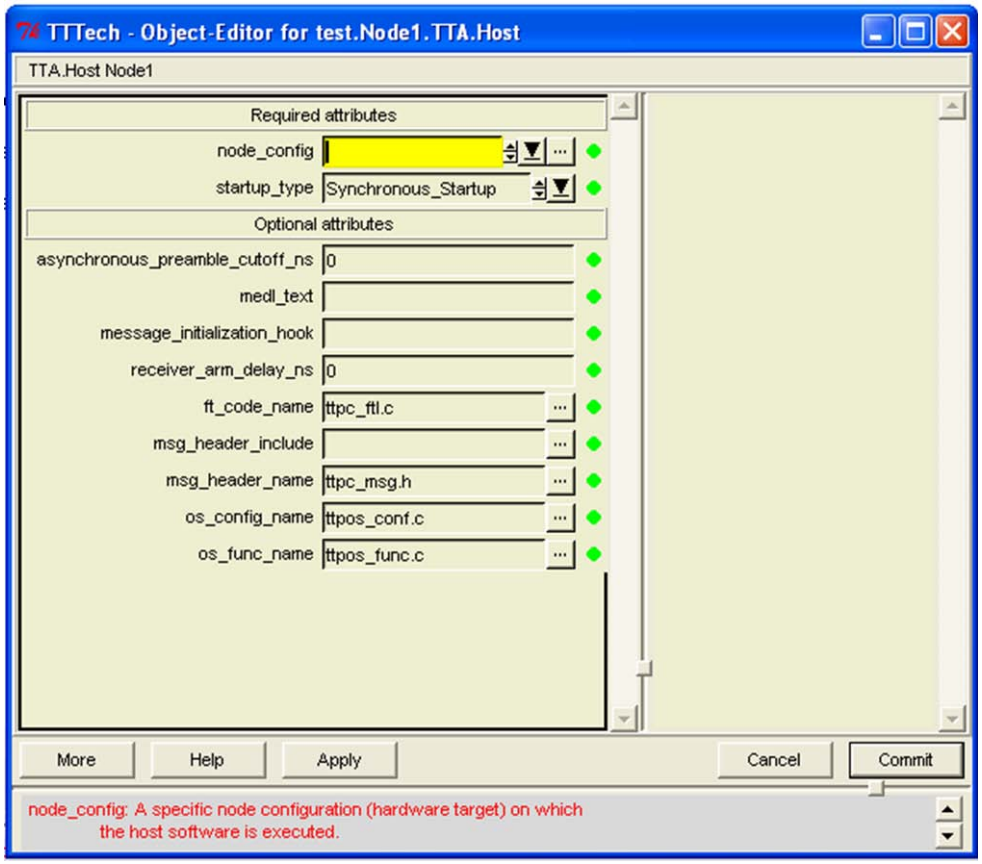

Figure D4. Host property editor

There are two required fields, node_config and startup_type (more information on page 7 of the TTP Build documentation). For the node_config field, select MPC555_AS8202NF as this is the

chipset of all four nodes in the development cluster. For startup_type, select either Synchronous_Startup or Asynchronous_Startup based on the requirements of your application (See page 58 of the TTP OS manual for more details). Click Commit when finished.

6. Click Close on the Object Editor Window (notice that you can only view the other Nodes in your cluster, because TTP Build can only work on one Node at a time during this process). Click Next in the Guide and continue to the TTA.Node.Subsystem_runs_Task page.

7. On the TTA.Node.Subsystem_runs_Task page, you will be linking subsystems and tasks. Tasks are the functional units in the TTP model (in other words the application that you create to use the data transmitted over the TTP bus). Each task is executed by exactly one subsystem, but each subsystem may execute as many tasks as necessary. Click the Edit button to open the following dialogue show in Fig. D5:

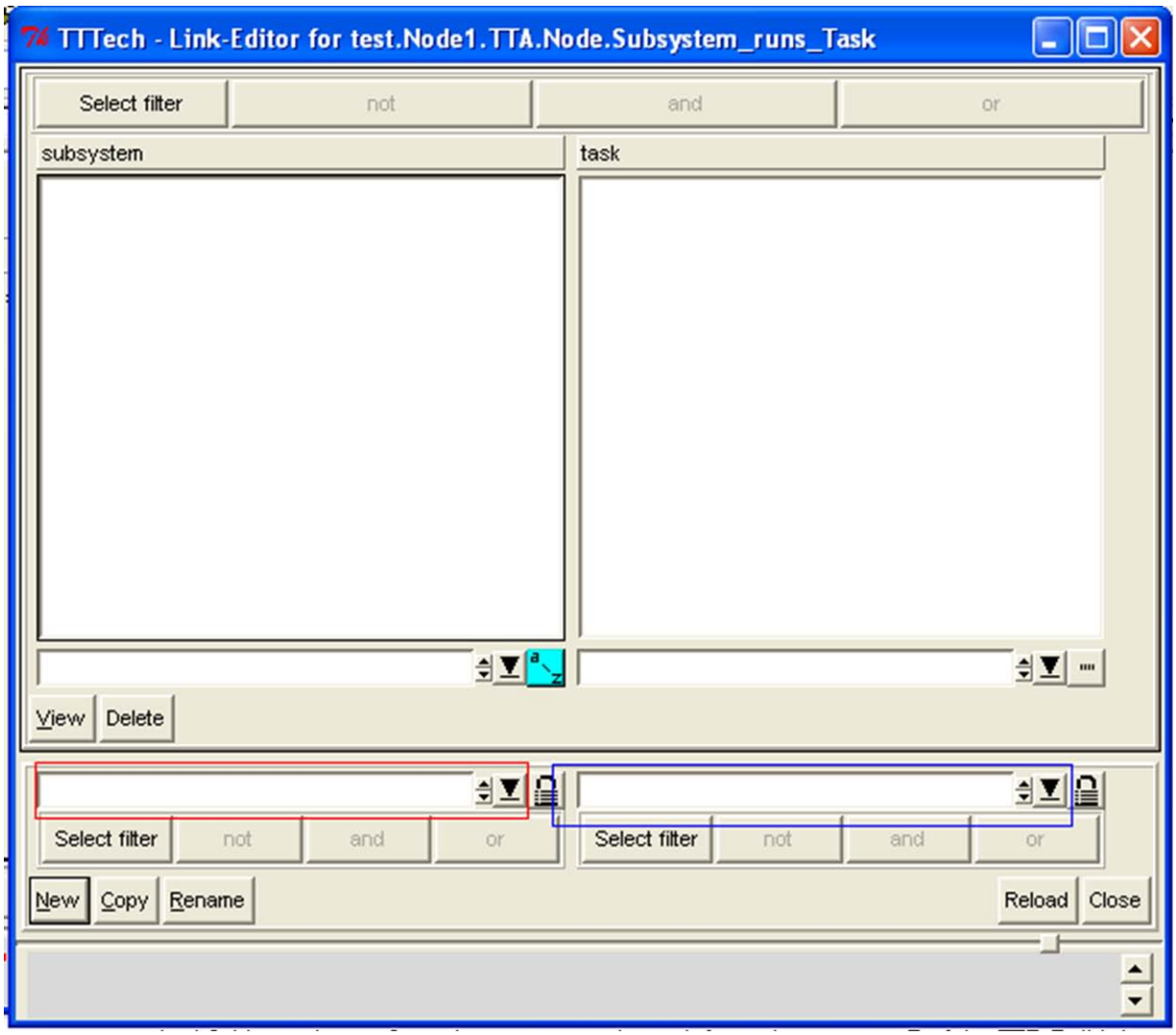

Figure D5. Subsystem, task linker

8. The red highlighted field will contain the subsystems you defined in TTP Plan. The blue field will initially be empty. You will add tasks here by typing their name into the field. Once you've entered a subsystem and task, click New to create the link. When done, your links will look something like those shown in Fig. D6:

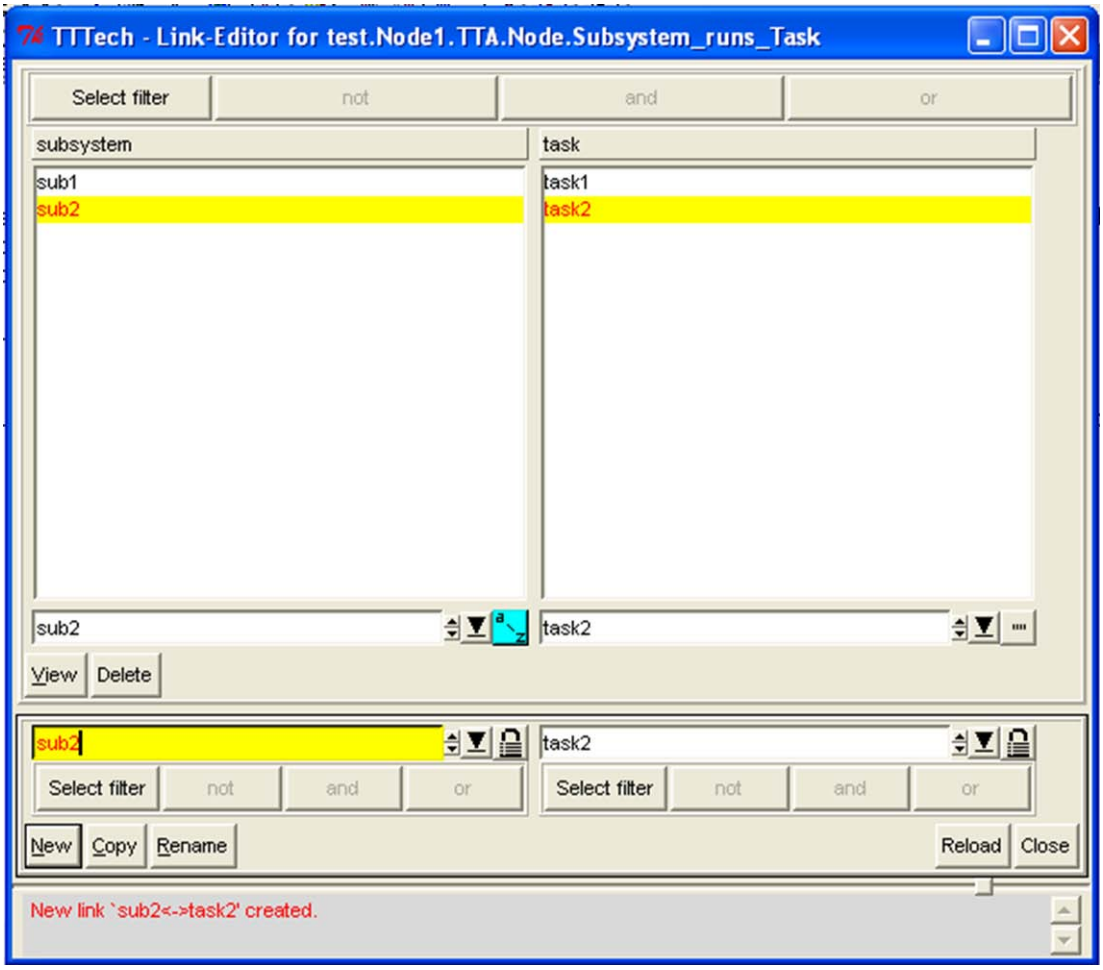

Figure D6. Examples of subsystem, task links

9. Click Close when you've created all your links. Click Next in the Guide to continue to the TTA.Node.App_Task page. You will edit parameters for the tasks you just created in the previous step. Click Edit to bring up a list of the tasks running on this Node (see Fig. D7):

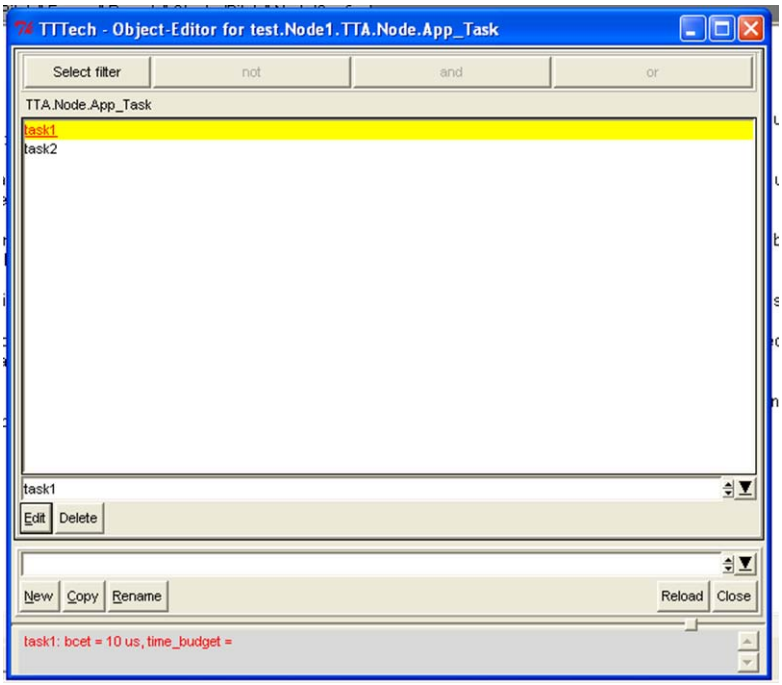

Figure D7. Task editor list

10. Select a task on the list and then click Edit. You will then open the edit task dialogue shown in Fig. D8.

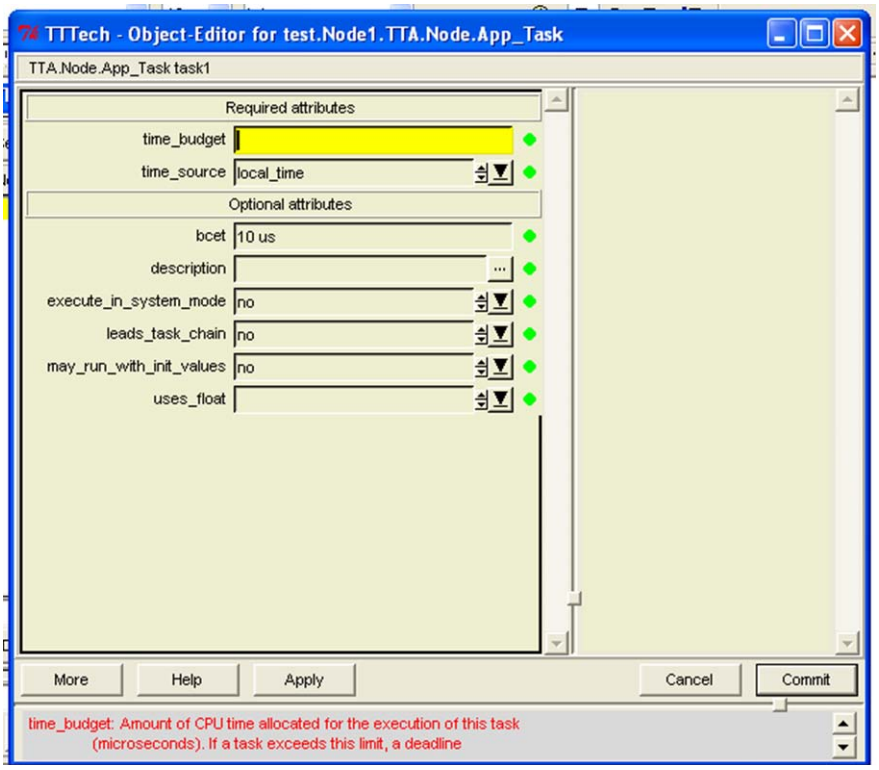

Figure D8. Task property editor

11. The two required fields are time_budget and time_source (see page 10).

time_budget - is implementation specific, but the value should not be larger than the time alloted to the node during a single TDMA round. In fact, it will need to be quite a bit smaller to incorporate the time taken to pull the data from the bus, move it to memory and then reverse the process during transmission.

time_source - depends on the application requirements. If your task needs to be run regardless of whether the global clock is available, then run on local time.

Optional values are described on page 21.

12. Once you've added the required values, click Commit. Then add the required values for any other tasks. Finally, close the Object List Editor with the Close button and click the Next button in the Guide. You will continue to the TTA.Node.Task_uses_Message page.

13. The TTA.Node.Task_uses_Message page lets you link messages to task. Each message is sent by one and only one task, but each task can send multiple messages. To create the links, click the Edit button in the Guide shown in Fig. D9:

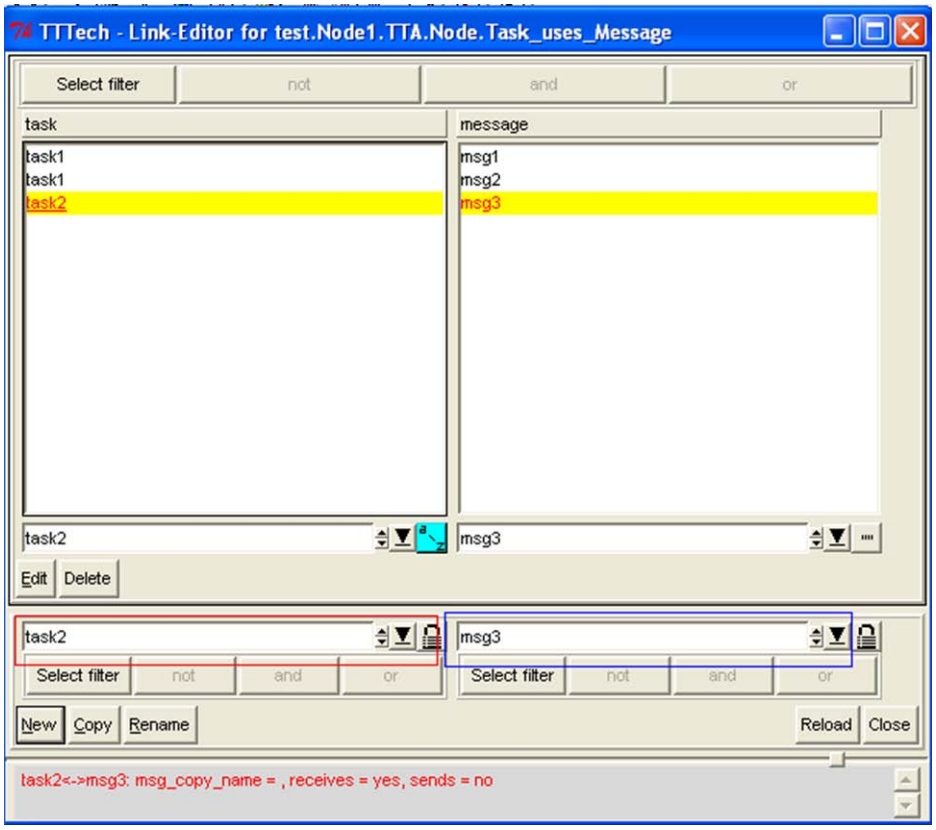

Figure D9. Example of task, message links

14. In the red highlighted field shown in Fig. D9, select one of your previously defined tasks. In the blue highlighted field, type the name of one of your messages that will be sent by the task (please keep in mind that you should stay consistent with your naming convention used in TTP Plan). Click the New button to create the link. Do this for all your tasks and messages. Links will be show above in the task/message lists.

15. For each link, you will need to add a few required values. Highlight a link and then click the Edit button (just above the red highlight). The will open a new dialogue box as shown in Fig. D10.

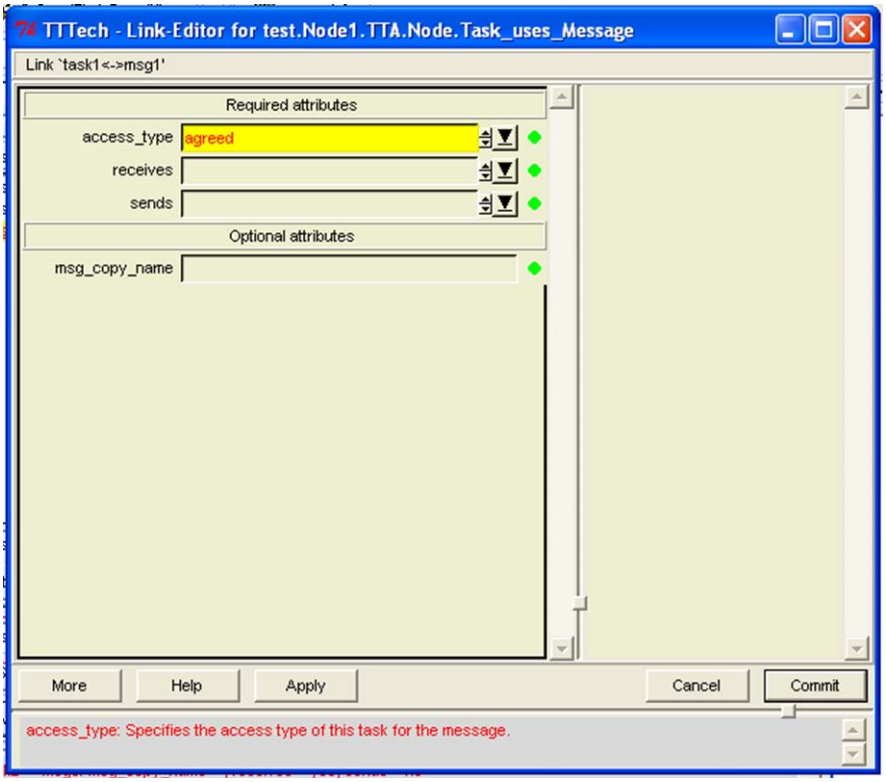

Figure D10. Task, message link property editor

16. The three required fields are (see page 11 for required fields, page 57 for optional fields):

access_type - determines how the values this task recieves are handled. raw states that all messages in are directly passed to task, whereas agreed waits for the messages to be processed by the agreement functions to determine message validity.

receives - set to yes if this task receives this message, no if it does not.

sends - set to yes if this task sends this message, no if it does not.

17. Click Commit when finished and then Edit each of your links before closing the link editor window. Click Next in the Guide to continue to the TTA.Node.IO_Message page.

18. The TTA.Node.IO_Message page lets you create links between task and messages that do not travel over the TTP bus. For example, if you task depended on a sensor reading that entered into the system from outside of the Node, this is where you specify to TTP that this message must be used and accounted for in the timing of the code. Click Edit to bring up the link editor window as shown in Fig. D11:

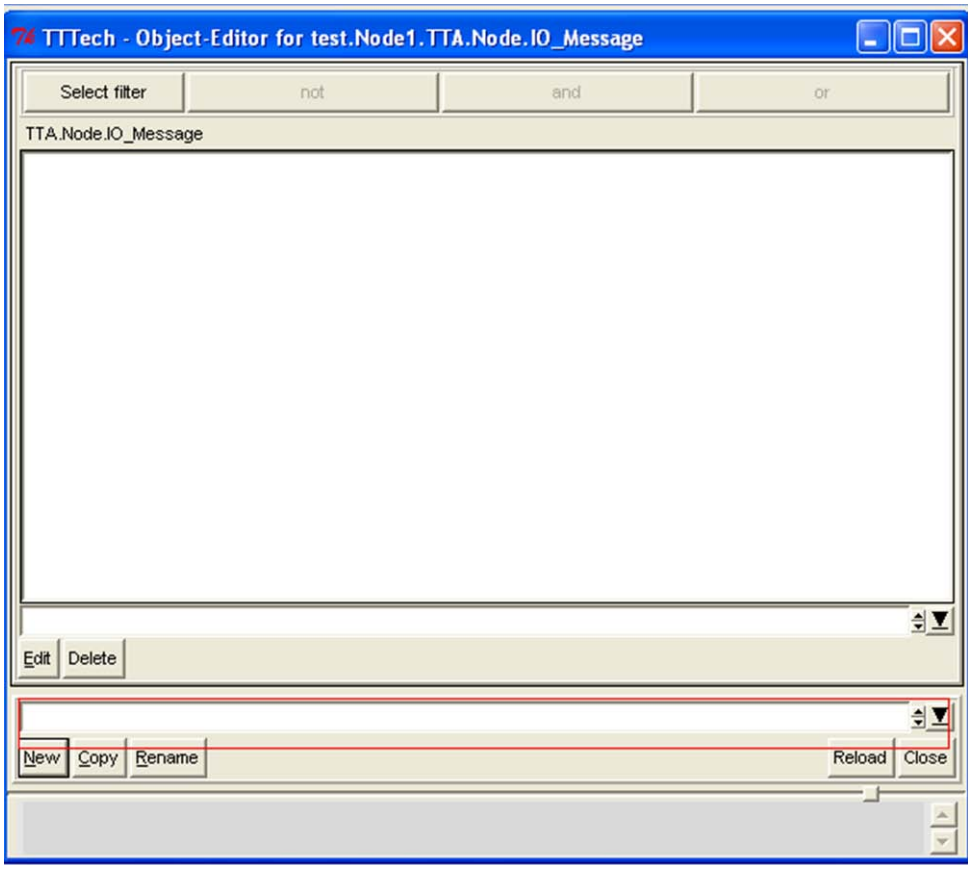

Figure D11. Message list editor

19. Enter the name of the IO message into the red highlighted field and then click the New button. Once the message is in the Node.IO_Message box, select it and then click the Edit button (just above the red highlight). This will open the edit dialogue for that message, as shown in Fig. D12:

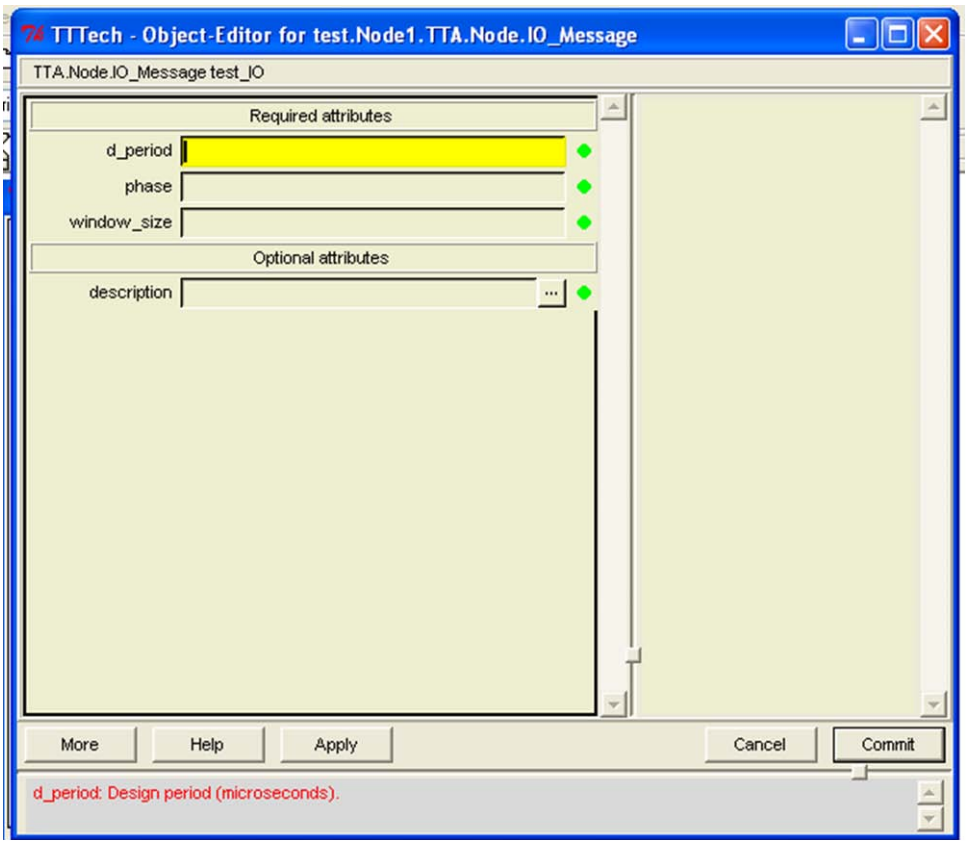

Figure D12. Message property editor

20. There are three required fields (page 13 for required fields, page 59 for optional fields):

d_period - *[from the manual]* defines the time between two consecutive I/O messages. This has to be an integer fraction of the cluster cycle time*[end excerpt]*. This means that you must define how often the task should receive this message and that this must be received at least once per cluster cycle (more is allowed).

phase - defines where in the cluster and I/O message should start.

window_size - time allotment to allow for variation when the message could be received (a phase of 250 microseconds with a window_size of 500 would allow for a message to be received anywhere from 0 to 500 microseconds).

21. When you have added all the required values, click Commit. Make sure to edit values for all additional IO messages. Then click Close to leave the I/O message editor window. Click Next in the Guide to continue to the TTA.Node.App_Mode page.

22. The TTA.Node.App_Mode page will let you define what tasks are run during a specifc mode. Nodes are allowed to run in different functional modes. To edit a mode, click the Edit button (see Fig D13):

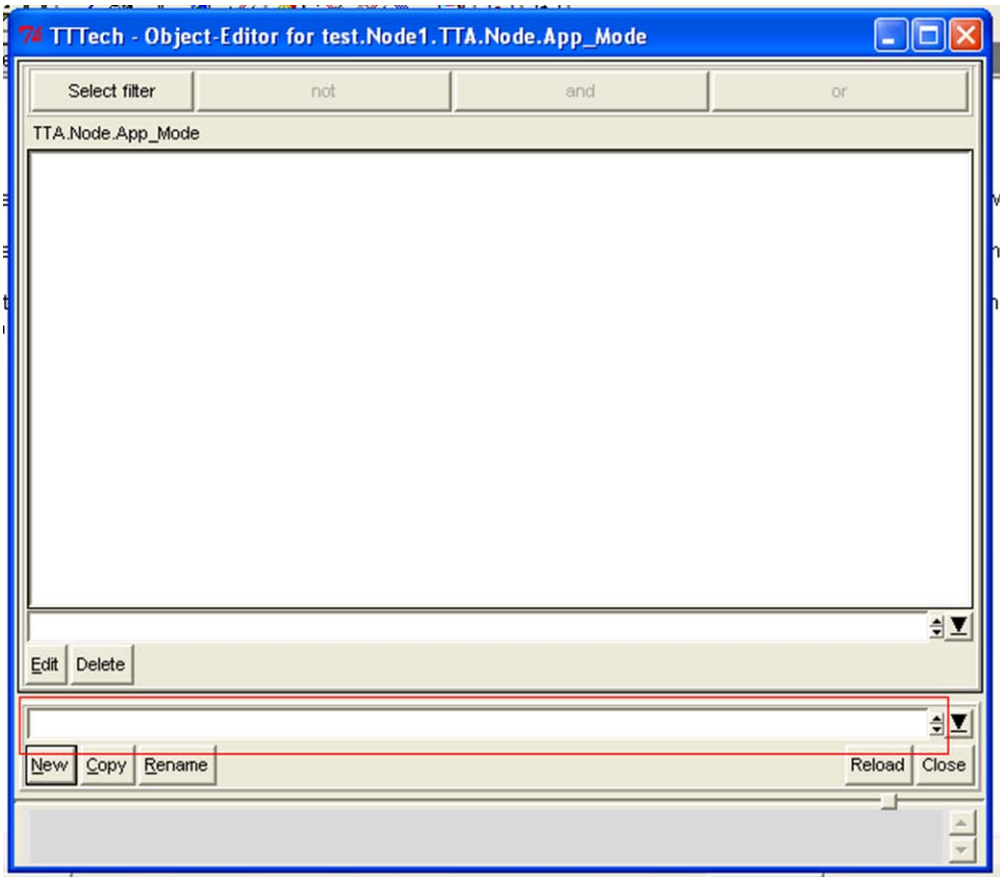

Figure D13. Application mode list editor

23. You enter in a new mode name in the red highlighted box (see Fig D13). Then click New to add it to the list. Once in the list, select the mode, and click the Edit button (above the highlighted field).

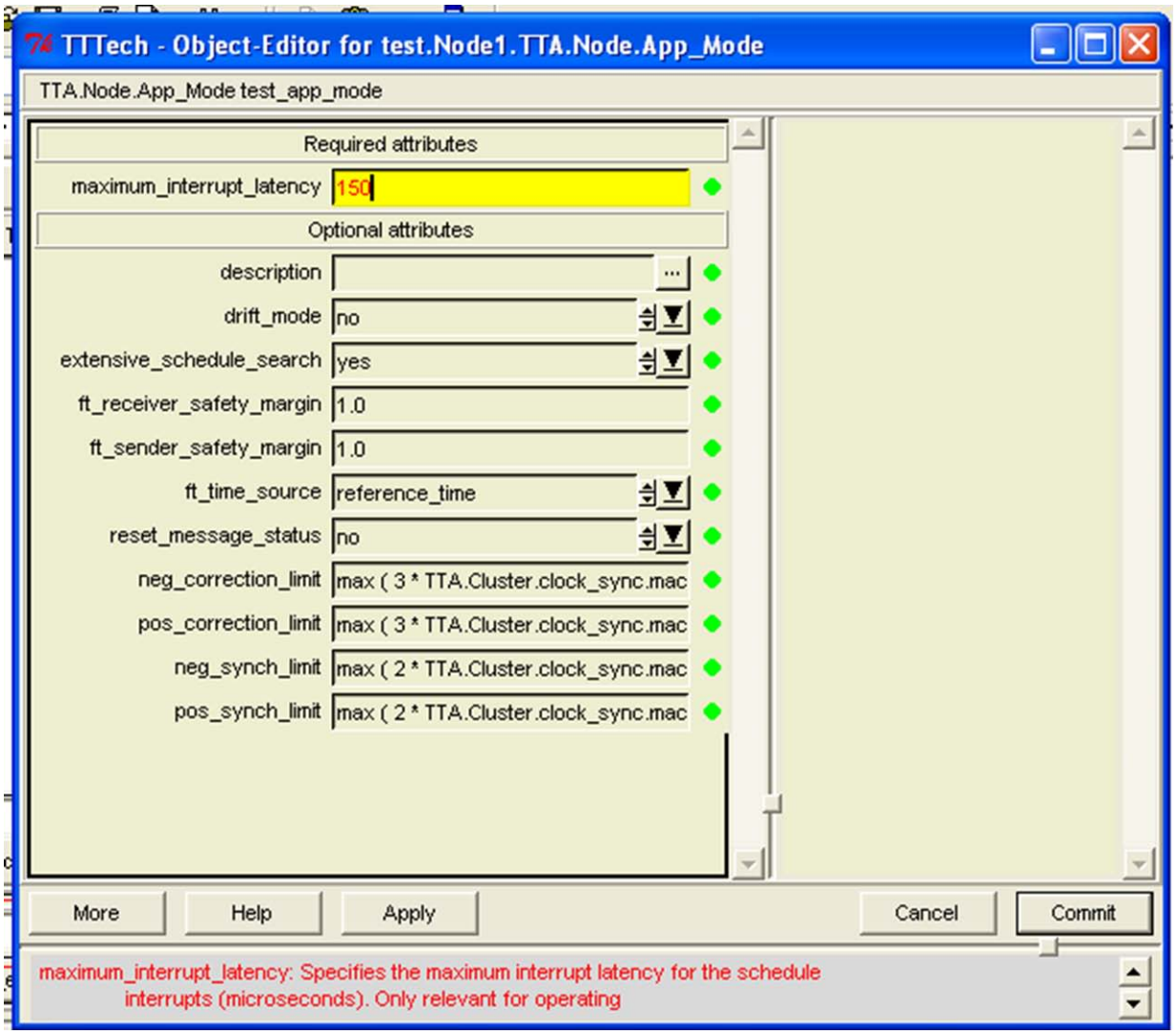

Figure D14. Application mode property editor

24. There is only one required field, maximum_interrupt_latency (see Fig. D14). TTP Build provides a reasonable default value. If you need to alter this value or need more information on the optional values, see section 3.6 of the TTP Build documentation. Click Commit when finished

and then Close the Object Editor box. Finally click Next in the Guide to continue to the TTA.Node.Task_in_App_Mode page.

25. The TTA.Node.Task_in_App_Mode page allows you to define what tasks run in a specific application mode (see Fig. D15). Click Edit to begin.

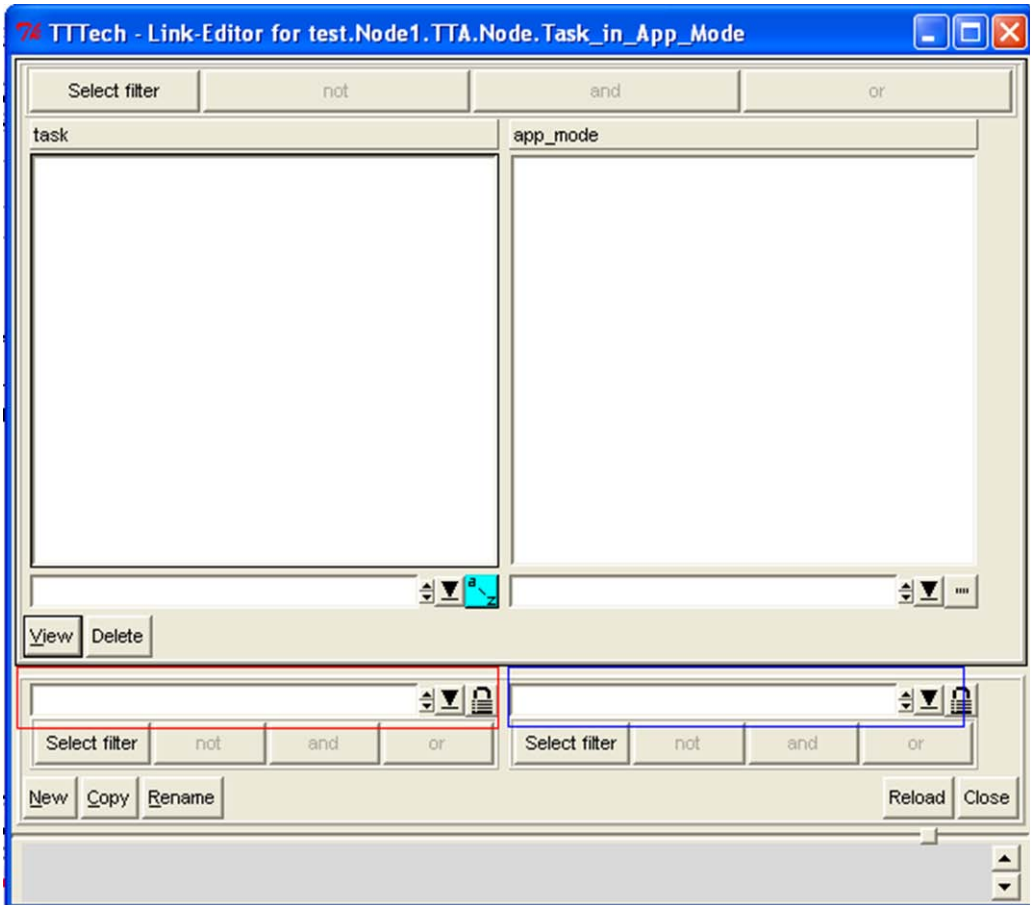

Figure D15. Task, application mode link editor

26. Select a task in the red highlighted field and an application mode (you just defined in the previous step) in the blue highlighted field. Click New to create the link and add them to the list. Do this for every task you wish to run in a given application mode. Click Close when you've added all your links and then click Next in the Guide to continue to the TTA.Node.App_Mode_maps_to_Cluster_Mode page.

27. The TTA.Node.App_Mode_maps_to_Cluster_Mode page lets you create a link between the application mode you've recently created and the cluster mode you created in TTP Plan (see Fig. D16).

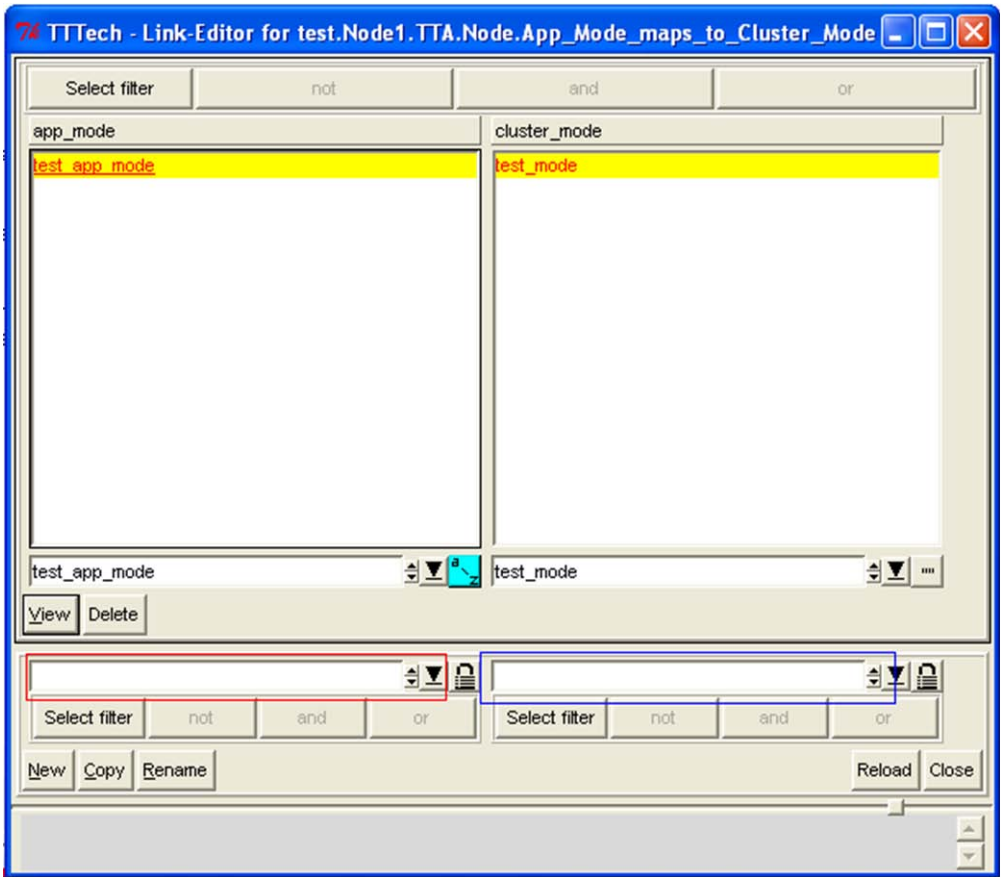

Figure D16. Application mode, cluster mode link editor

28. If the application hasn't already done so, create a link by selecting an app mode in the red highlighted field and a cluster mode in the blue highlighted field. Click New and the link will be created. After creating all your application's links, Click Close to close the Link Editor window and then click Next in the Guide to continue to the Checking for errors page.

29. Similar to TTP Plan, this page builds the node scripts and will alert you to any configuration

errors. Click the Check button and you will receive output similar to Fig. D17:

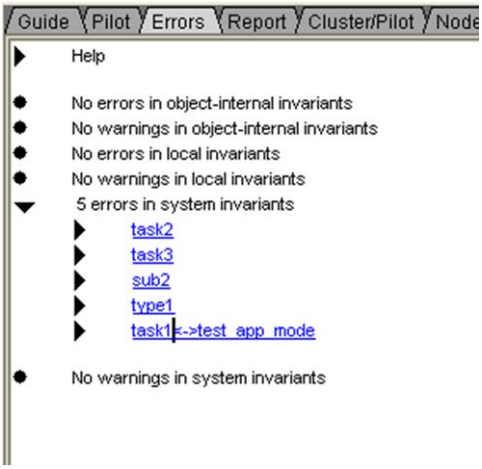

Figure D17. Error check output

30. *Please Note* Prior to generating your schedule, you must make sure that your node is properly configured for the development cluster (PN312) as shown in Fig D18. Go to the Node/Config tab in TTP Build (highlighted in red) and then double click on the Node-Config block (highlighted in blue):

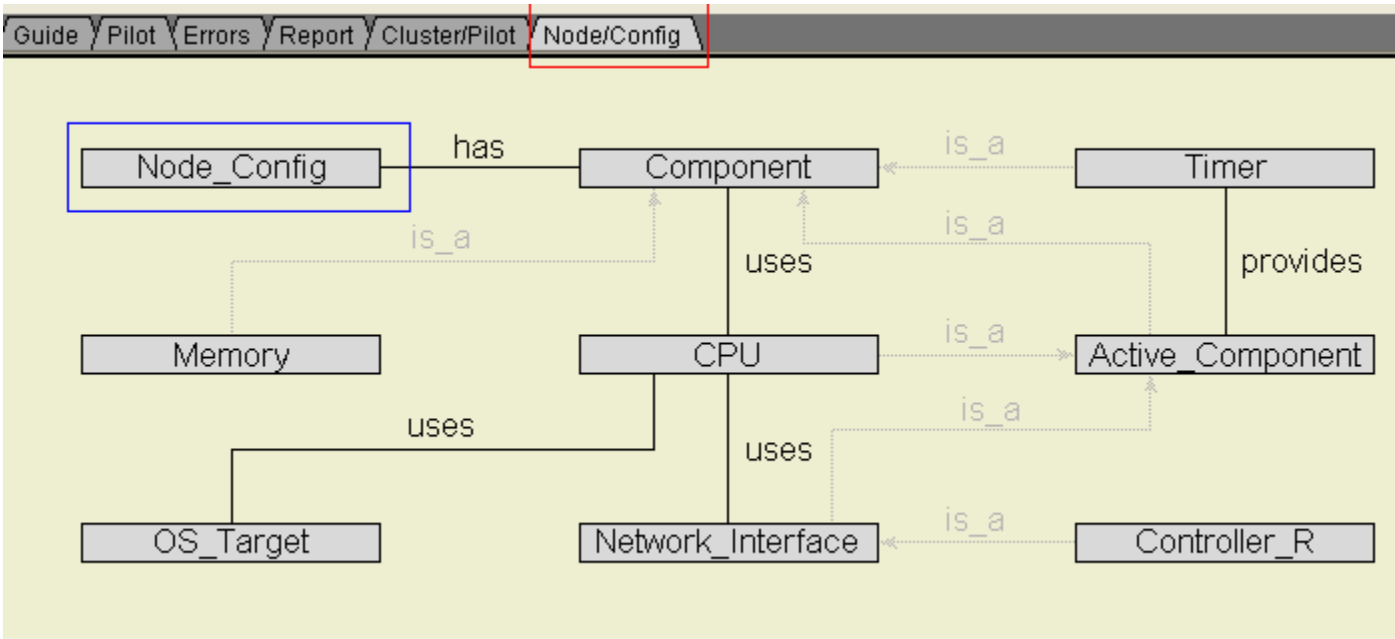

Figure D18. Node configuration tab

31. You will need to make sure that the dialog box shown in Fig. D19 contains the value PN312_External. If it does not, select PN312_External in the dropdown box (highlighted in red) and then click the load button (highlighted in blue):

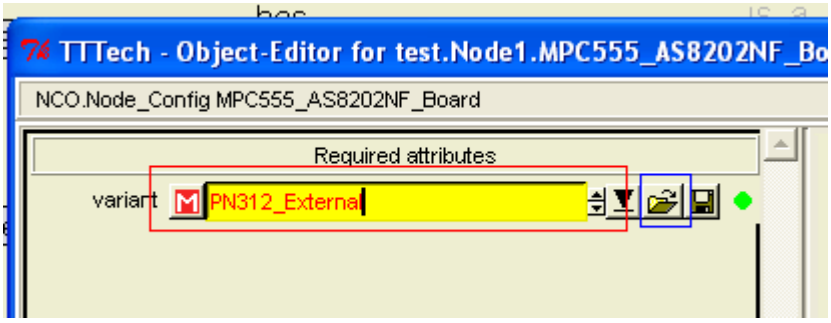

Figure D19. MPC555 property editor

32. Once you have loaded the correct configuration, click Commit and return to the Guide tab within TTP-Build to continue error checking.

33. In order to see detailed information about any errors or warnings, open the various trees and click the links to be directed to the problem configuration page. Once you have resolved all application errors, click the Next button in the Guide to continue to the Making a schedule page.

34. Again, this page is similar to the TTP Plan Making a schedule page. Click the Schedule button and then click Next to continue to the Generate FT-COM code page.

35. The Generate FT-COM code page will create all of the files needed to build the FT-COM layer of the TTP Node. This will create a MEDL and source code for the Node in question. You will still need to run a make file to compile the source code prior to loading into the Node. Click the FT-COM button to generate the code and then click Next to continue to the Congratulations page. You have completed one of the Nodes in the development cluster. Repeat Steps 3 - 32 for each other Node in the development cluster. Once you have completed each node, you will then need to compile your code. After the executable has been built, you will then use TTP Load to upload the MEDL and executable to each of the nodes in turn.

Appendix E. TTTech TTP Load Quick Start Guide

1.Start TTP-Load

Start Button -> All Programs -> TTTech -> TTP-Load-6.5.11-> TTP-Load

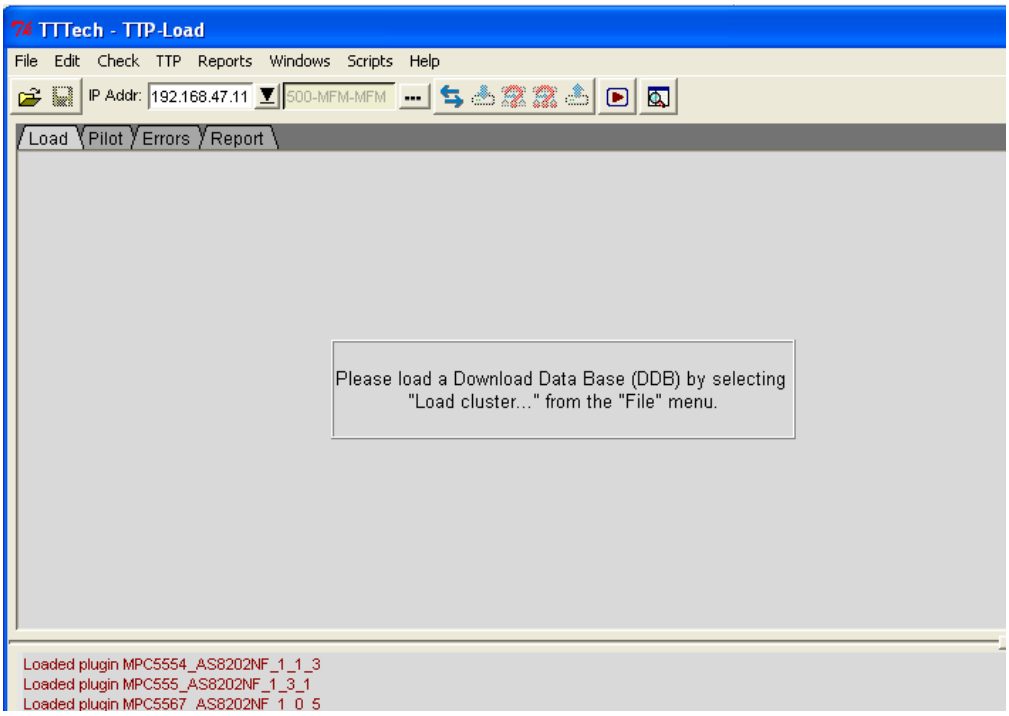

Figure E1. TTP-Load main window

2. First we need to load the cluster definition file. Select File -> Load Cluster (see Fig. E2)

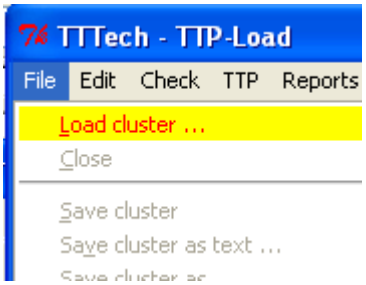

Figure E2. Load cluster menu item

3. Next, navigate the location of your cluster database file (*.ddb) and then click Open (see Fig. E3).

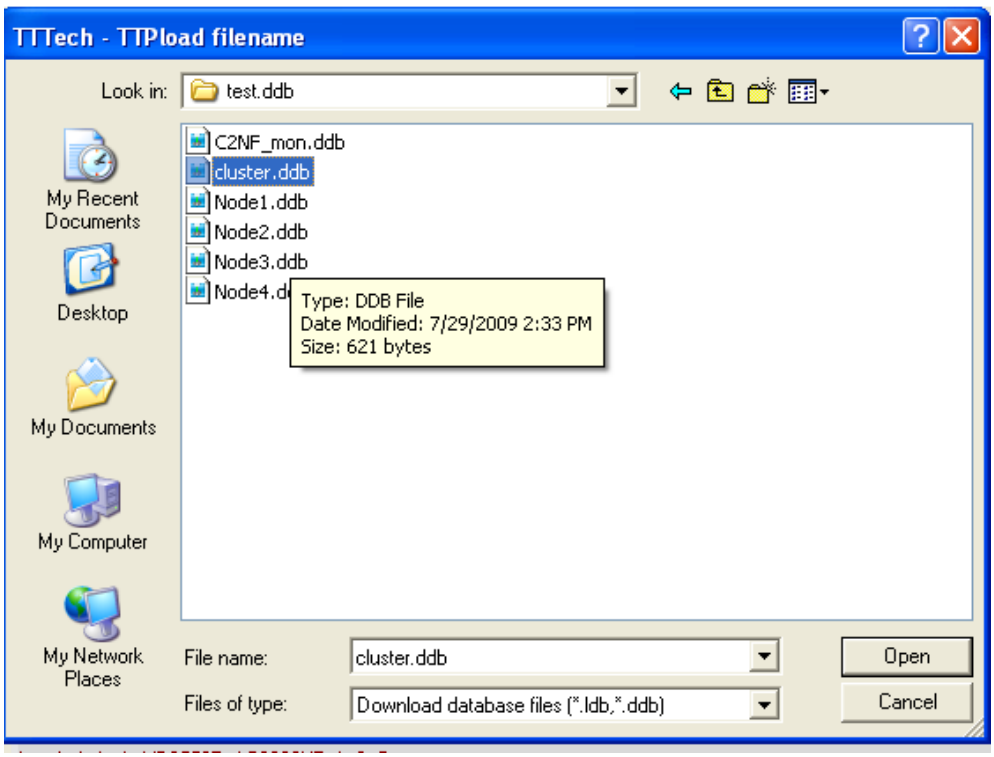

Figure E3. Load cluster dialog

4. Your window should now look like Fig. E4.

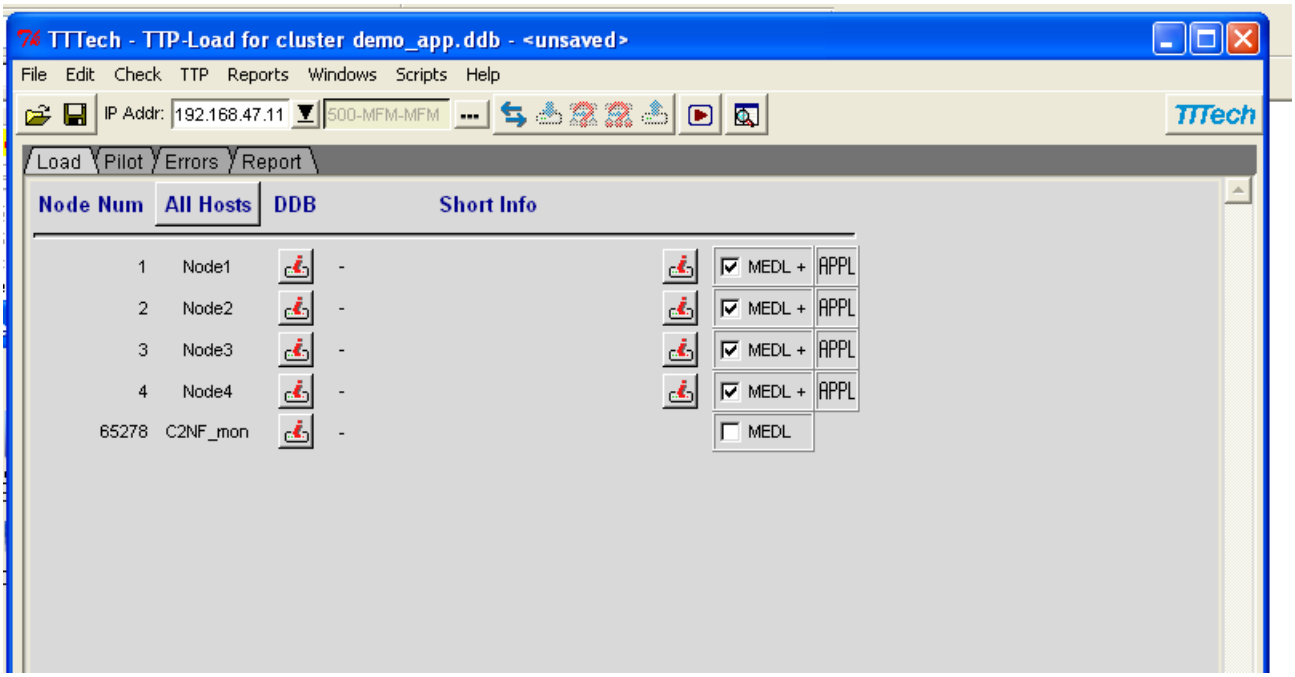

Figure E4. Example of loaded cluster

5. If any of the check boxes are grayed out, it means that those node databases need to be rebuilt against the loaded cluster database (see Fig. E5). Rebuild the grayed out nodes in TTP-Load.

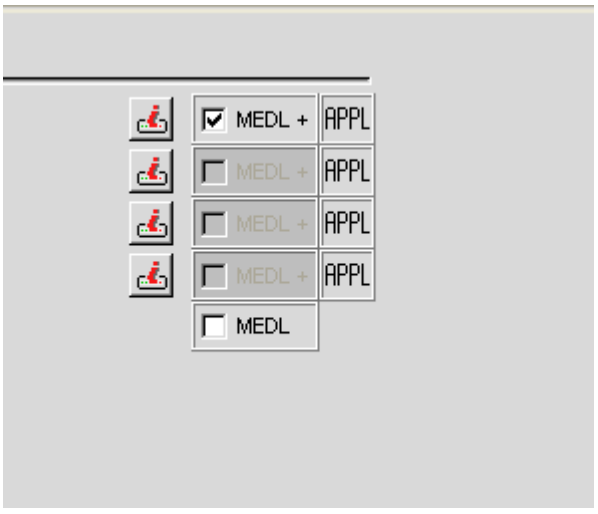

Figure E5. Out of sync MEDLs

6. Next, check the box next to the C2NF_mon MEDL. This represents the monitoring node. It will need a current copy of the MEDL as well, if you intend to view the activity of the development cluster with TTP View.

7. Next, verify that you are connecting to the correct IP address of the monitoring node. Your window should resemble Fig. E6:

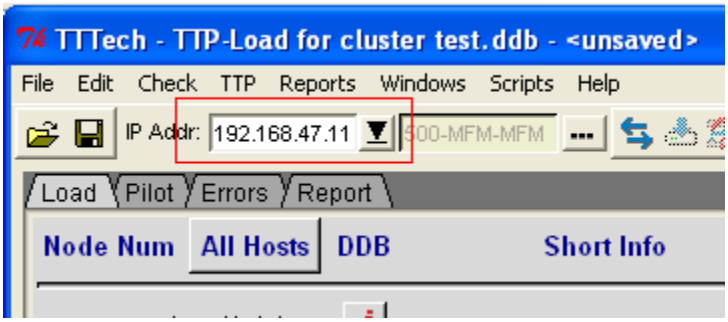

Figure E6. Monitoring node IP address

8. Please note that the monitoring node has been configured to listen at the 192.168.47.11 IP address.

9. Finally, select the TTP.Download ddb button to begin the download process as shown in Fig. E7 *(the term download here refers to downloading the new MEDL into the monitoring node. You are in fact pushing the new node firmware to each node in turn via the monitoring node)*.

Figure E7. TTP download button

10. You will see a series of progress bars. If there were no errors find the compiled .s19 files, you will see the development cluster enter into download mode. Each node will stop the current running task and you will see LED output to signify download mode.

Appendix F. TTTech TTP View Quick Start Guide

1.Start TTP-View

Start Button -> All Programs -> TTTech -> TTP-View-6.5.6-> TTP-View

2. TTP View will start up and you will begin with this window (see Fig. F1).

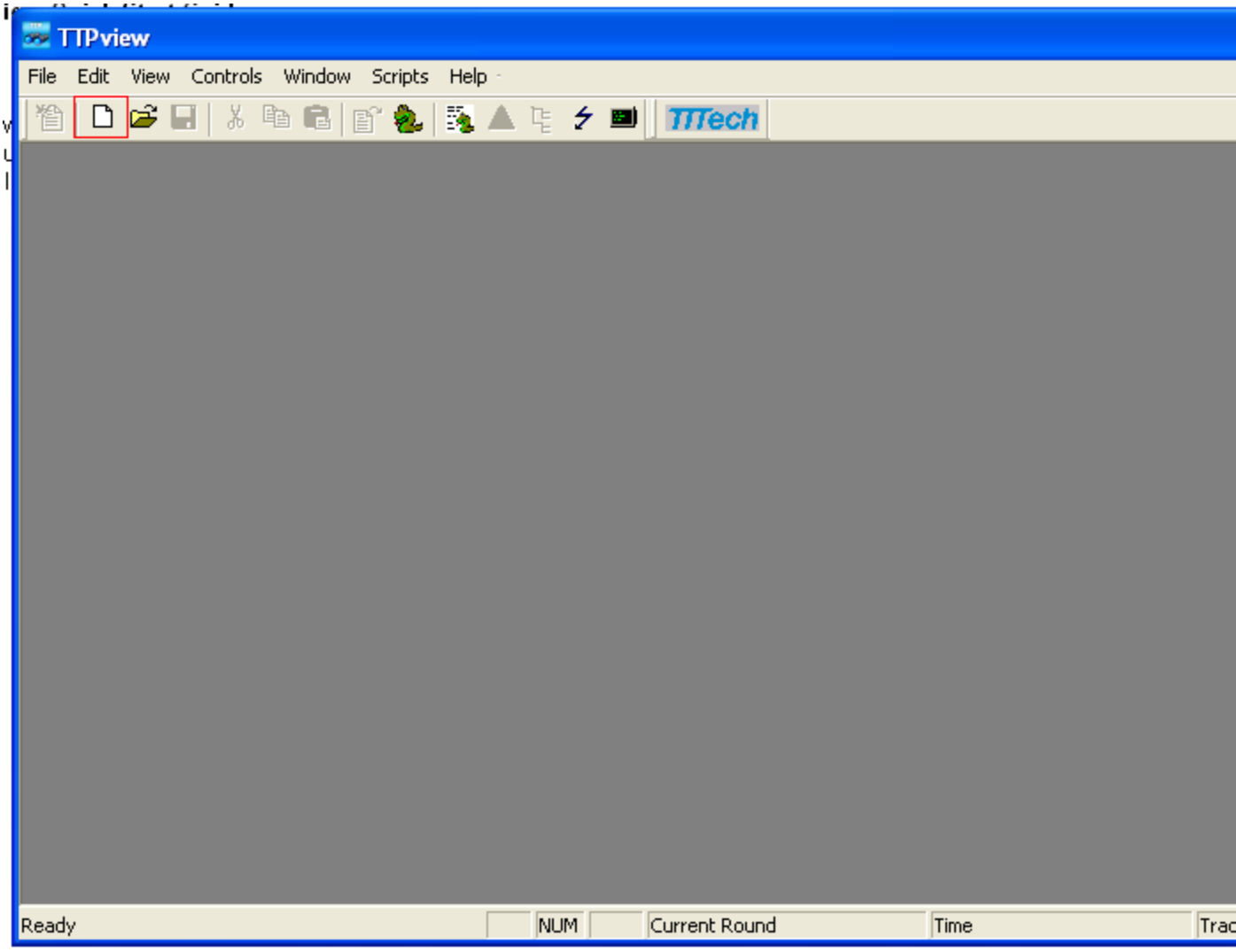

Figure F1. TTP-View main window

3. If this is your first use of TTP View with your cluster, the click the red highlighted button (create a view database from cluster database).

4. This will open a file dialogue box where you should locate your cluster database (see Fig. F2).

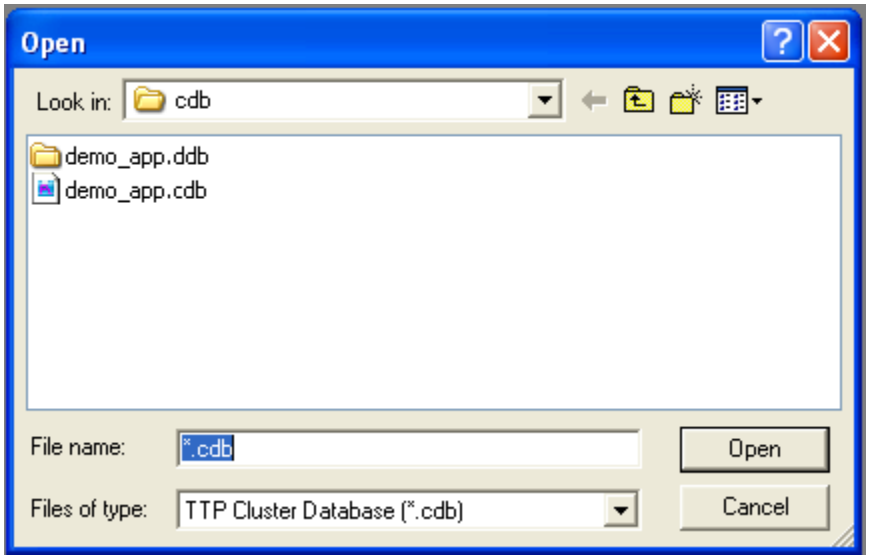

Figure F2. Open file dialog

5. Select your cluster file and then click Open. If you already have a view database, you will then need to answer yes to create a new database or no to update (see Fig. F3).

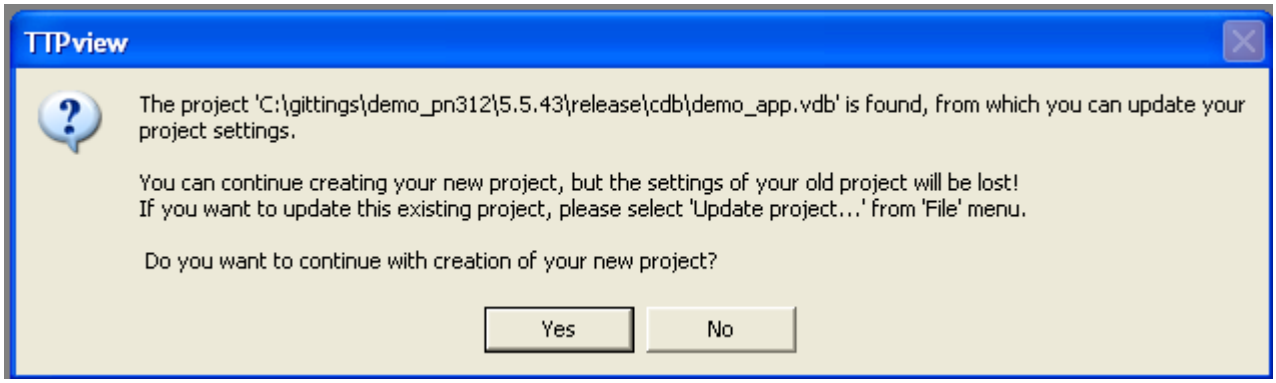

Figure F3. Create new project modal dialog

6.Once you've opened the view database, you should see Fig. F4:

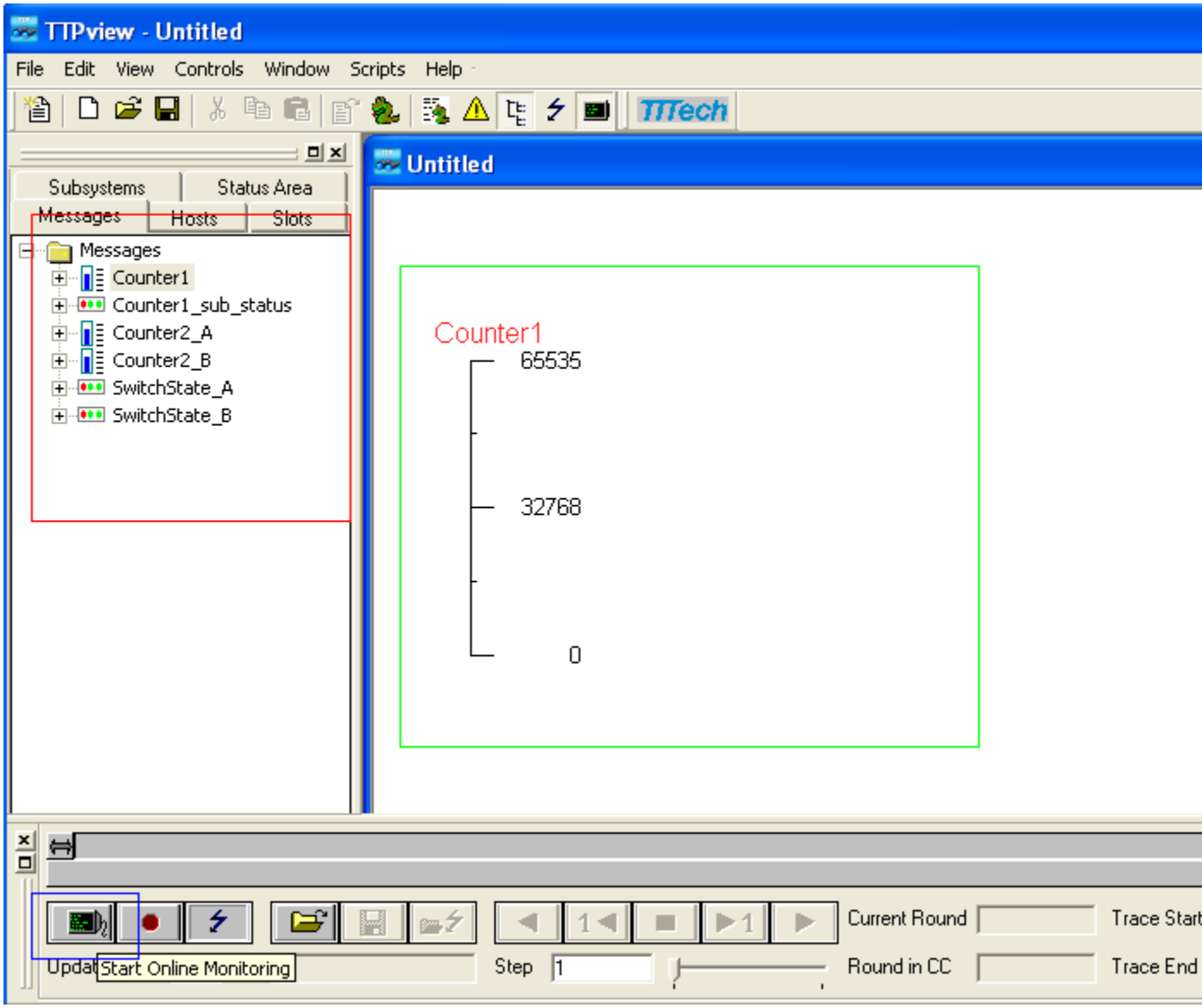

Figure F4. Variables loaded for monitoring

7. In the red highlighted area, your applications variables are listed. In order to begin monitoring your variables, you must click and hold a variable from the red area and drag it to the green area. You will then see a representation of your variable based on its type definition. To begin

monitoring the variable through the monitoring node, click the blue highlighted button (Start Online Monitoring). This will begin collecting data from the monitoring node and store it in memory.

8. To stop monitoring your variables (and begin analyzing them), click the stop button (highlighted in red) as shown in Fig. F5.

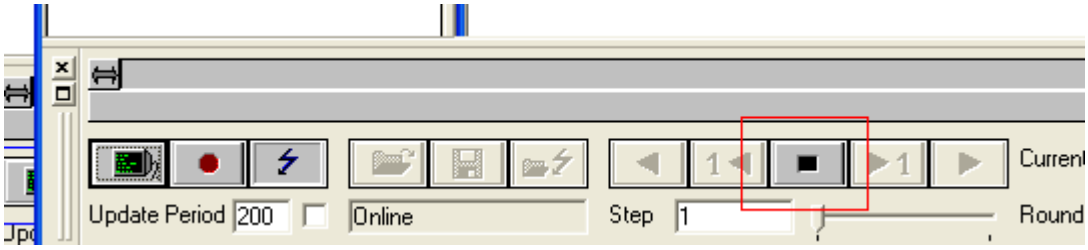

Figure F5. Stop recording button

9. Once you have stopped recording, you can then step or scan through your output using the vcr type controls (see Fig. F6).

Figure F6. VCR controls

10. You can step a single frame at a time forward or back, by clicking on the 1< or >1 buttons. You can also quickly scan along the process by clicking anywhere within the blue timeline bar. Finally, to save your output, click on the disk icon.

Appendix G. MATLAB Quick Start Guide

1.Start MATLAB

Start Button -> All Programs -> MATLAB -> R2008a -> MATLAB R2008a

2. MATLAB will start with the desktop open and the command window ready to accept input as seen in Fig. G1.

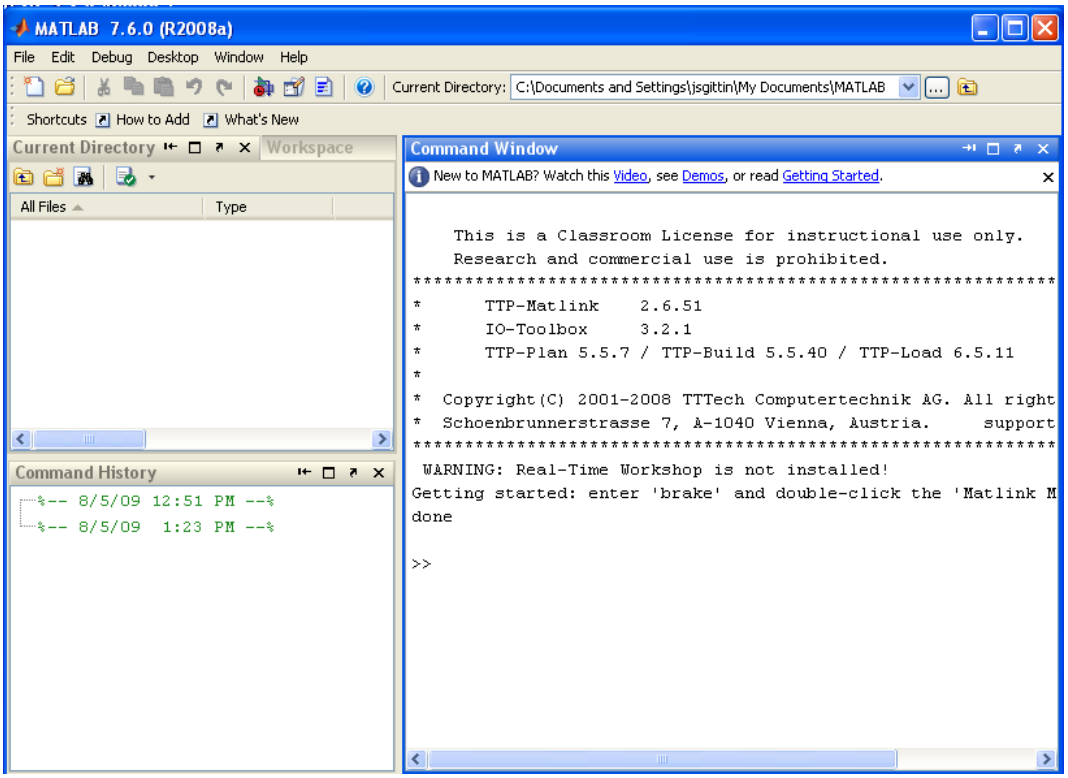

Figure G1. MATLAB main window

3. For an initial basic tutorial of the windows and general functionality of MATLAB, please see the following videos:

Getting Started -

C:\MATLAB\R2008a\toolbox\matlab\demos\html\GettingStartedwithMATLAB.html

Working with the desktop -

http://www.mathworks.com/support/2008a/matlab/7.6/demos/WorkingInTheDevelopmentEnvironment.html

Writing a program -

http://www.mathworks.com/support/2008a/matlab/7.6/demos/WritingAMATLABProgram.html

4. The focus of this appendix is to create models in MATLAB using the TTP Simulink objects for developing applications.

5. Select File ->New->Model (see Fig. C2)

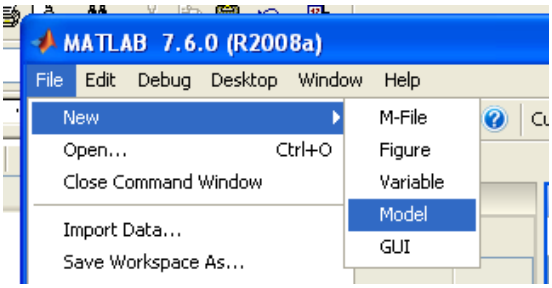

Figure G2. New model menu item

6. A new model window will appear (Fig. C3):

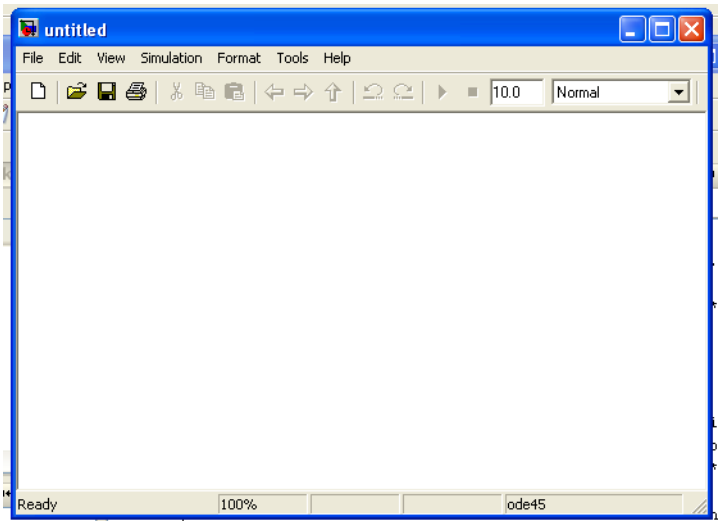

Figure G3. New model window

7. In order to add objects into the model, we will need to open the object Library. Click View -> Library Browser (Fig. C4).

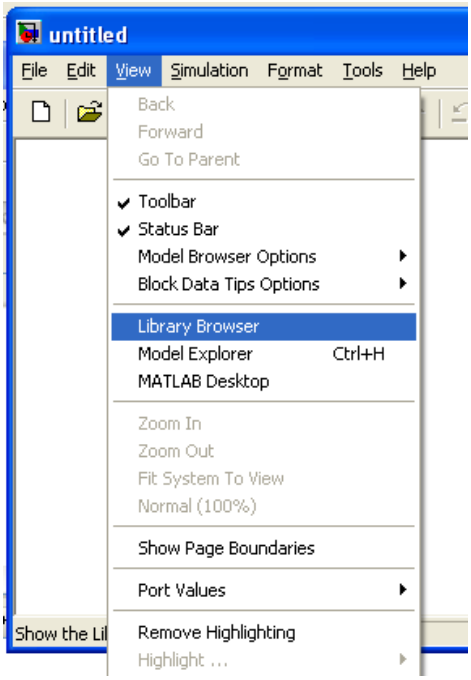

Figure G4. Library browser menu item

8. This will open the symbols library (Fig. C5):

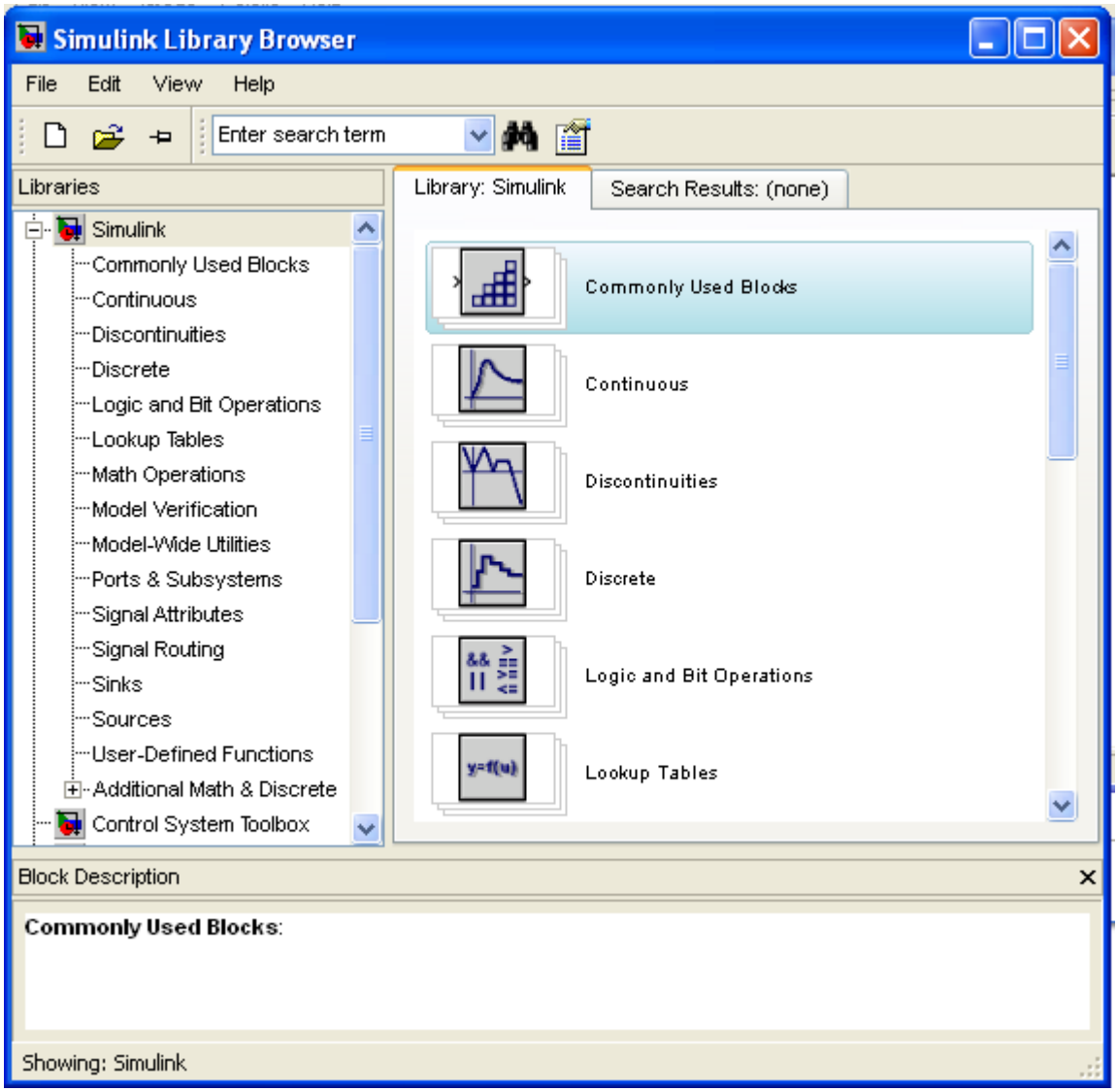

Figure G5. Simulink library browser

9. The first step in creating a TTP application model is adding a TTP Matlink main dialogue object to the model (Fig C6).

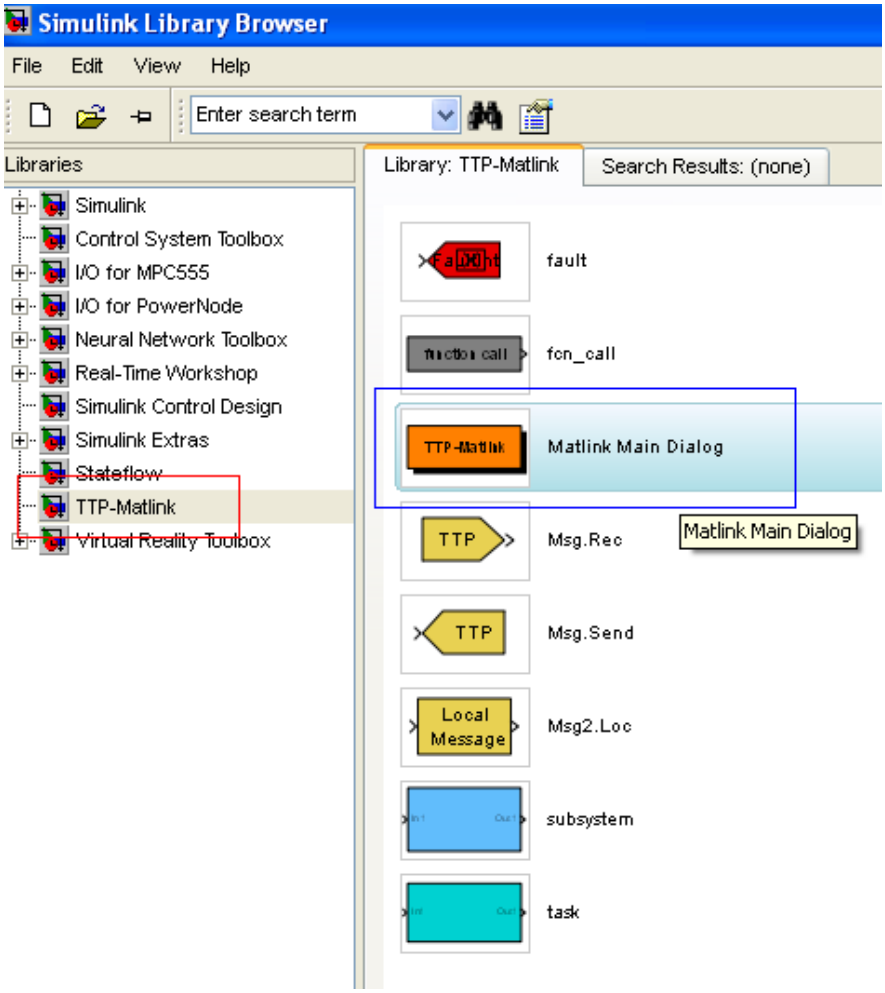

Figure G6. TTP Simulink objects

10. Select TTP-Matlink under the Libraries heading (red highlight) and then click and drag an instance of the Matlink Main Dialog (blue highlight) into the model window as shown in Fig. C6.

11. Your model should now look like this (Fig. C7):

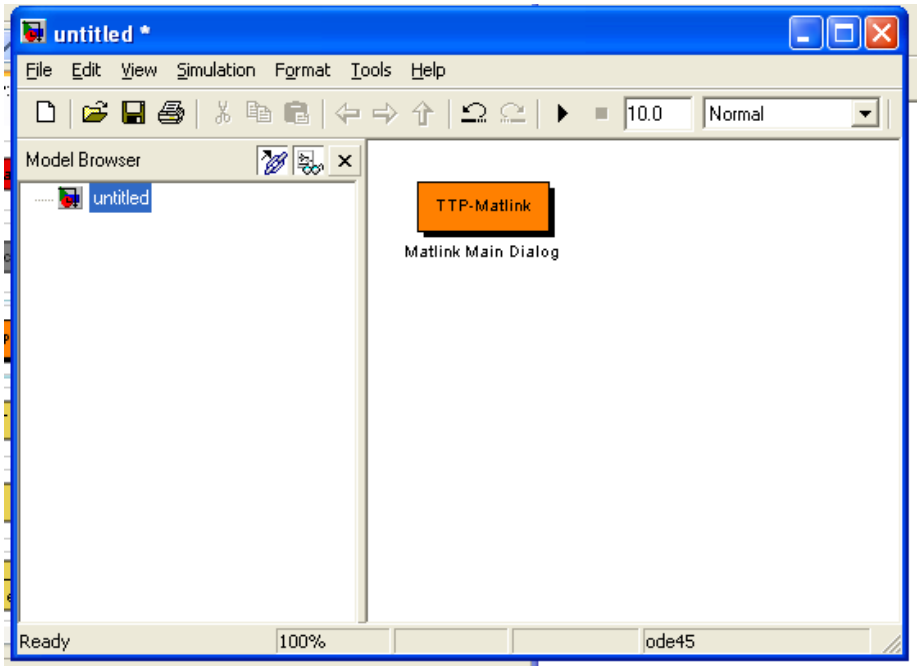

Figure G7. First object in TTP model

12. Now, double click on the orange TTP-Matlink box to bring up the properties dialog (Fig. C8).

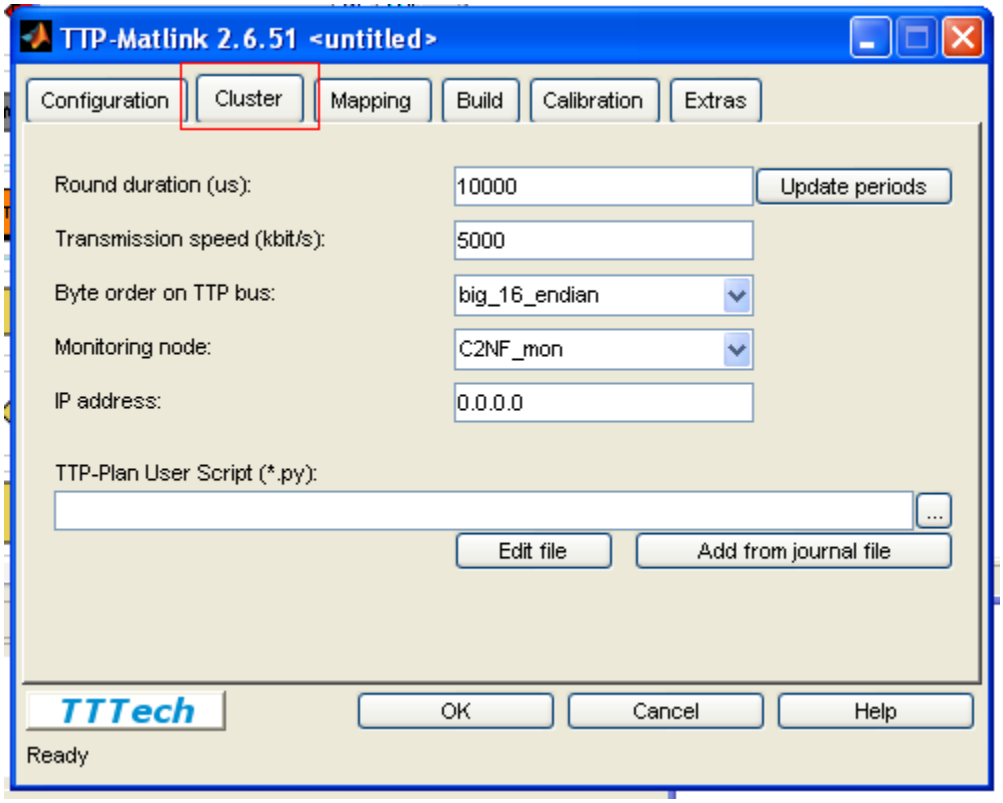

Figure G8. TTP main object properties dialog

13. If it isn't already selected, please select the Cluster tab (highlighted in red). The following fields will need to be completed:

Round Duration - the length of your TDMA round in microseconds.

Transmission Speed - the transmission speed of your TTP bus in kbits/s. A value of 2000 will suffice for the development cluster.

Byte order on TTP bus - this value depends on your application. A value of big_32_endian should suffice as a starting point. For more information on endianness, see the TTP Plan documentation (page 6) or for more general information, http://en.wikipedia.org/wiki/Endianness.

Monitoring node - The monitoring node for the development cluster is the C2NF_mon

IP address - The IP address of the monitoring node is 192.168.47.11

14. Once you've completed these fields, click OK to close the dialog.

15. A small application developed in MATLINK, for the development cluster will look like Fig. C9:

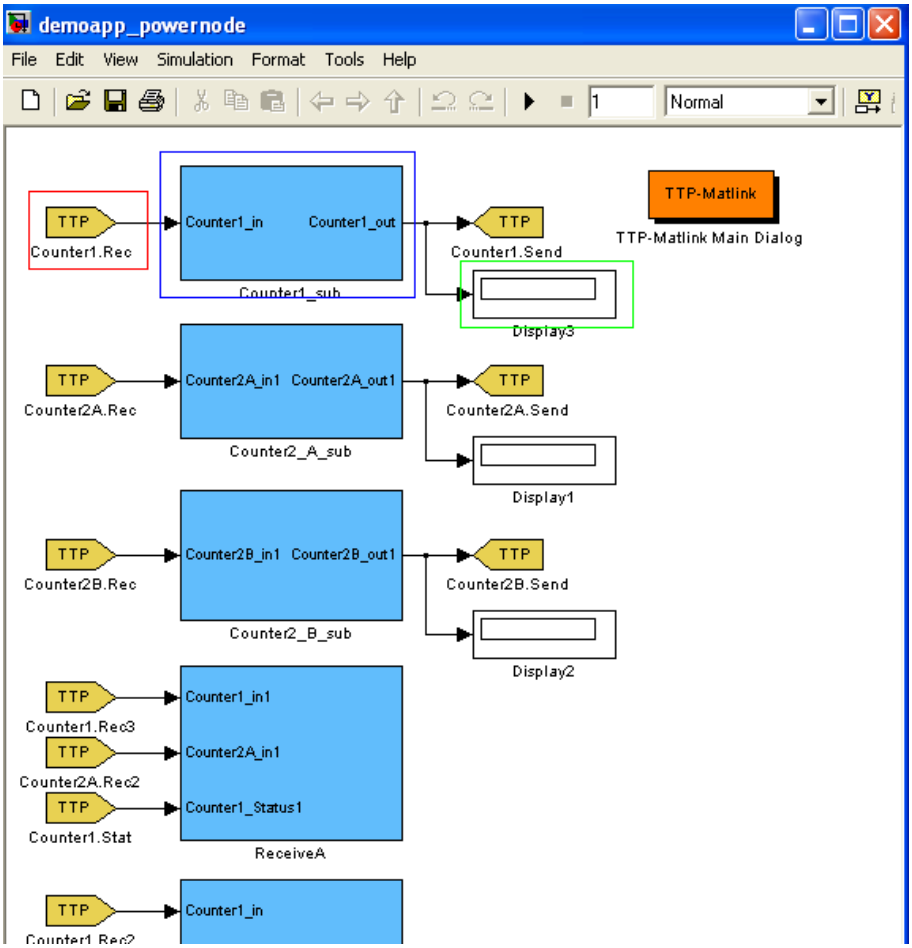

Figure G9. Example model

16. Messages into and out of subsystems are the yellow polygons (red highlight). Subsystems are the blue rectangles (blue highlight) and output is show in the display boxes (highlighted green). Any of these objects can be double clicked to open property dialog boxes for further configuration. Note that subsystems, when double clicked, will display the tasks within that

subsystem.

17. For example, double clicking the Counter1_sub box, will open that subsystems task (Fig. C10):

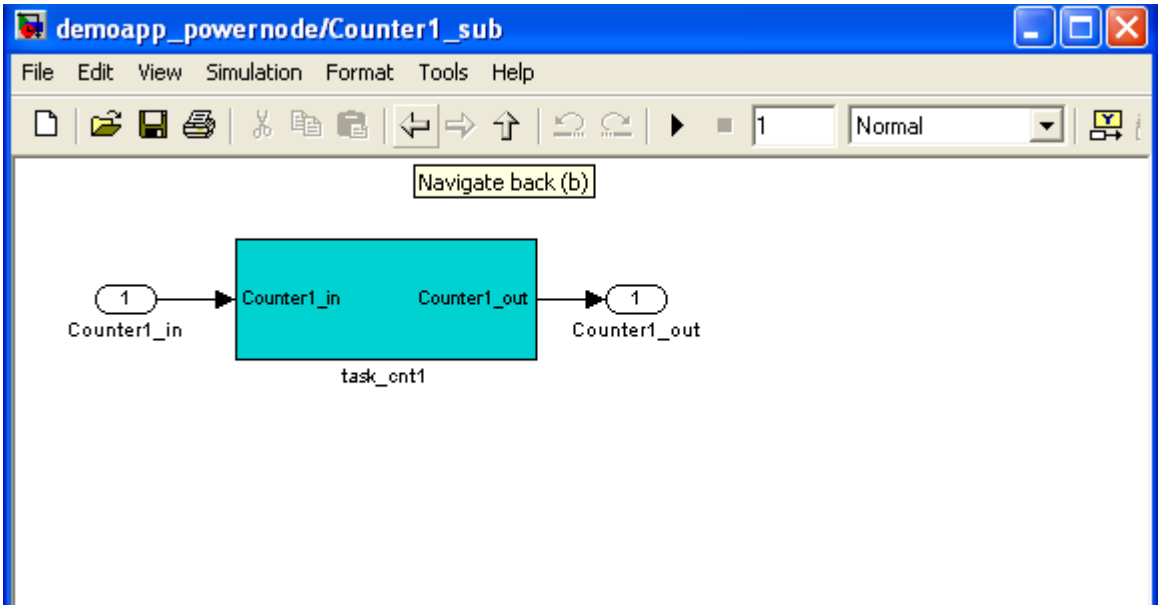

Figure G10. Subsystem task window

18. Double clicking the any task will open the its grouped objects. For example, double clicking on task_cnt1 will reveal Fig. C11:

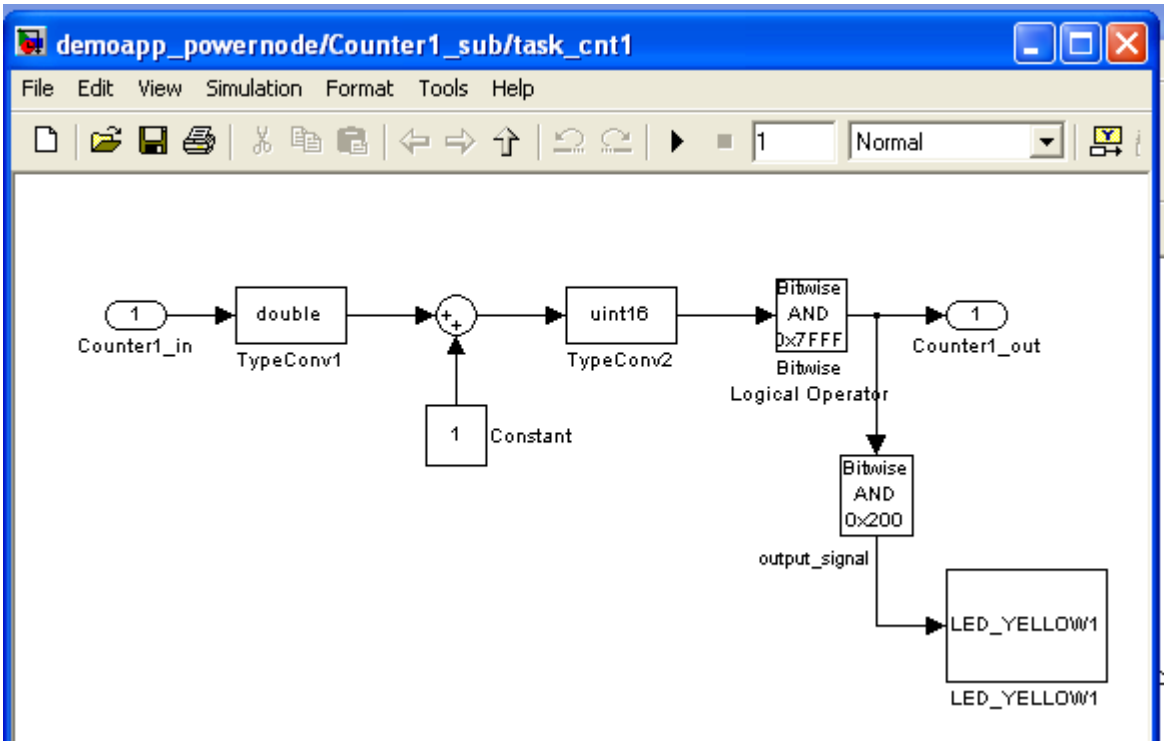

Figure G11. Task example

19. You will know where you are in the object hierarchy by looking at the title of the model window.

20. When you have completed your model and are ready to test it. You can click the Run Simulation button to run the program and view its output via the display boxes (assuming you have them) as shown in Fig. C12.

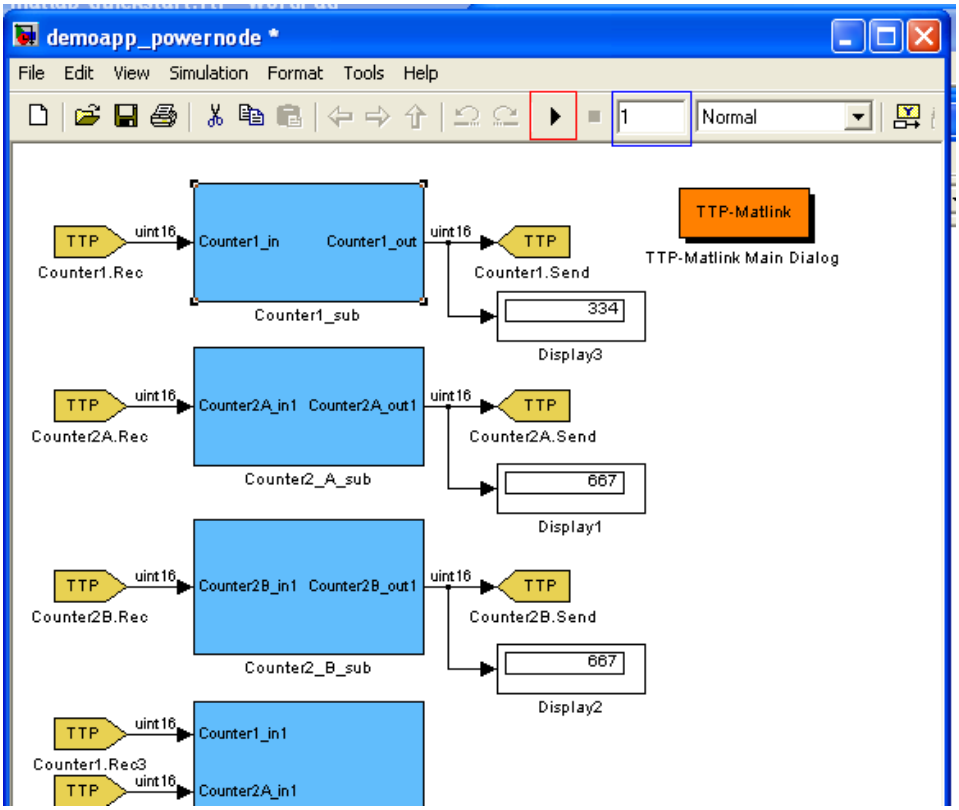

Figure G12. Run simulation example

21. The run simulation button is highlighted in red. The length of time for the simulation to run is highlighted in blue (the value is in seconds). Finally, if you have display boxes, their values will now be shown in the associated text fields.

22. If you are happy with your model and wish to generate all the needed TTP MEDL and runtime files, double click the TTP-Matlink object (orange rectangle) and then select the Build tab as shown in Fig. C13.

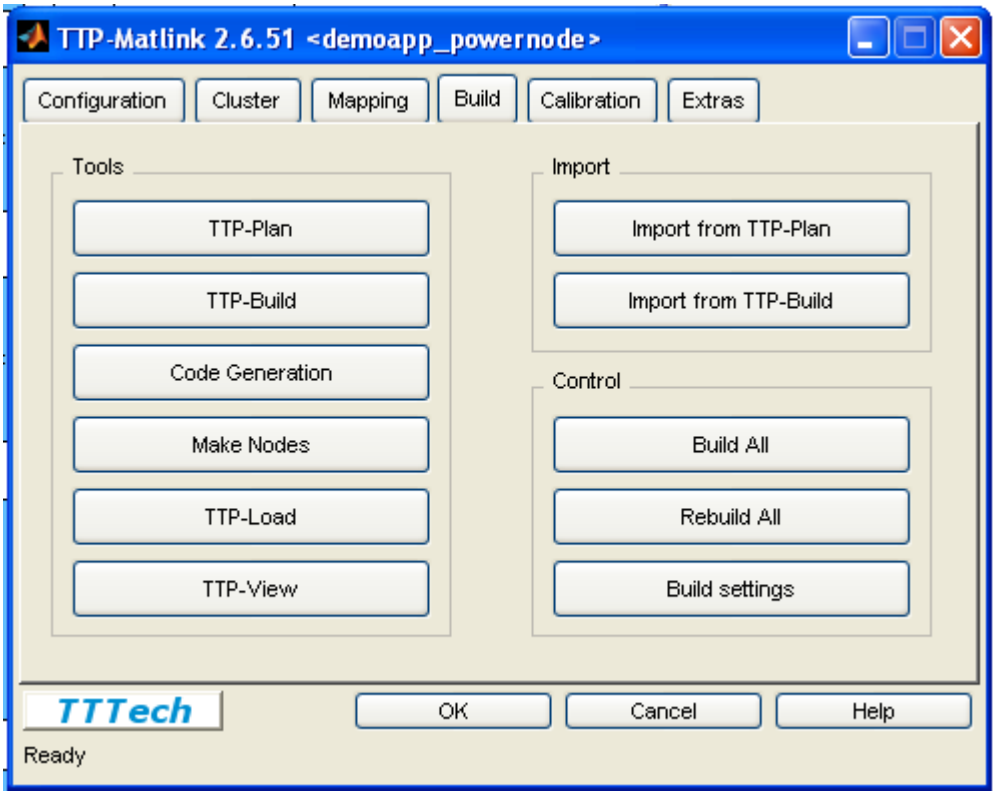

Figure G13. Automated TTP Build dialog window

23. Within this dialog box, you can select output for any step in the process or output everything and automatically push the new code into the nodes. You can also import from TTP Plan or Build if you have started your cluster design in those applications.

Appendix H. Experimental Disturbance Node XML Scenarios

H1. QVM Node 1 Disturbance XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs round 4 of qvm
770 us low voltage signal (brake calc node 1)
5 us offset
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="770" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
***********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
00000000000000000000000000000000000000000000d66f
5b4d008c018c000403200104000f334c00b7018c00040320
0104000f000000c80000006001800554002c000400030004
00000000000500140002d000000300000004000301be09ab
00000000003f800d00038f60225c52fed98c1dd400000000
a3a0aaaba0a9a2aabef2e6f4bea1a3a3aa933aa7d2d5a6aa
aba7aaa9a0d5d5d730333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
2020202020202020982f0004000000000002ffffffff0b78
001110110000000808080a5009990f604111011000000000
808080a500995a5a0811101100000000808080a500991c9f
```

```
0c111011c00080008080c0a500993c87000800000000ffff
ffffffffeb000011001100000000808080a50099f2640411
001100000000808080a5009938c808110011000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
001100000000828280a50098f7c704110011000000008282
80a500983d6b08110011000000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <run_scenario name="do"/>
      </scenario>
<!--
**********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <repeat_scenario name="disturb" loop="2500"/>

      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="4" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>
</disturbance>
```

H2. QVM Node 2 Disturbance XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs round 5 of qvm
760 us low voltage signal (brake calc node 2)
5 us offset
sent on both channels
repeated 2500 times
```

```
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="760" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
***********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000d66f
5b4d008c018c000403200104000f334c00b7018c00040320
0104000f000000c80000006001800554002c000400030004
000000000000500140002d0000003000000004000301be09ab
00000000003f800d00038f60225c52fed98c1dd400000000
a3a0aaaba0a9a2aabef2e6f4bea1a3a3aa933aa7d2d5a6aa
aba7aaa9a0d5d5d730333938343a31392d6175672d323030
39202020202020202020202020202020206a7367697474696e
2020202020202020982f000400000000000002ffffffff0b78
00111011000000000808080a5009990f60411101100000000
808080a500995a5a0811101100000000808080a500991c9f
0c111011c00080008080c0a500993c870008000000000ffff
ffffffffeb0000110011000000000808080a50099f2640411
0011000000000808080a5009938c808110011000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
0011000000000828280a50098f7c704110011000000008282
80a500983d6b08110011000000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <run_scenario name="do"/>
      </scenario>
<!--
***********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
```

```
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <repeat_scenario name="disturb" loop="2500"/>

      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="5" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>
</disturbance>
```

H3. QVM Node 3 Disturbance XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs round 6 of qvm
760 us low voltage signal (brake calc node 3)
5 us offset
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="760" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
************************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
```

```
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
02023800001838180028384000313871002738980047000
00000000000000000000000000000000000000000000d66f
5b4d008c018c00040320010400 0f334c00b7018c00040320
0104000f000000c800000060018005540 02c000400030004
000000000005001400 2d000000300000004000301be09ab
00000000003f800d00038f60225c52fed98c1dd400000000
a3a0aaaba0a9a2aabef2e6f4bea1a3a3aa933aa7d2d5a6aa
aba7aaa9a0d5d5d730333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
2020202020202020982f00040000000000 02ffffffff0b78
0011101100000000808080a5009990f60411101100000000
808080a500995a5a0811101100000000808080a500991c9f
0c111011c00080008080c0a500993c87000800000000ffff
ffffffffeb000011001100000000808080a50099f2640411
0011000000000808080a5009938c8081100110000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
0011000000000828280a50098f7c7041100110000000008282
80a500983d6b0811001100000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <run_scenario name="do"/>
      </scenario>
<!--
**************************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <repeat_scenario name="disturb" loop="2500"/>

      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="6" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
```

```
	</scenario>
</disturbance>
```

H4. QVM Node 4 Disturbance XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs round 7 of qvm (pedal sensor)
790 us low voltage signal
5 us offset
sent on both channels
repeated 2500 times
-->

	<definition name="const_signal">
	<!-- Disturbs a slot by sending constant signals on channel A. --
>
		<source channel="BOTH">
			<!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
			<duration>
				<unsigned_intvalue value="790" />
			</duration>
			<signal_low />
		</source>

	</definition>
<!--
***********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
	<scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
00000000000000000000000000000000000000000000d66f
5b4d008c018c000403200104000f334c00b7018c00040320
0104000f000000c80000006001800554002c000400030004
00000000000500140002d000000300000004000301be09ab
00000000003f800d00038f60225c52fed98c1dd400000000
a3a0aaaba0a9a2aabef2e6f4bea1a3a3aa933aa7d2d5a6aa
aba7aaa9a0d5d5d730333938343a31392d6175672d323030
39202020202020202020202020202020206a7367697474696e
20202020202020200982f0004000000000002ffffffff0b78
0011101100000000808080a5009990f60411101100000000
808080a500995a5a08111011000000000808080a500991c9f
0c111011c00080008080c0a500993c870008000000000ffff
ffffffffeb000011001100000000808080a50099f2640411
```

```
00110000000808080a5009938c808110011000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
00110000000828280a50098f7c704110011000000008282
80a500983d6b0811001100000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <run_scenario name="do"/>
      </scenario>
<!--
************************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <repeat_scenario name="disturb" loop="2500"/>

      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>
</disturbance>
```

H5. QVM Pedal Sensor Disturbance (1:5) XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
5 us offset
every fourth cluster cycle
sent on both channels
repeated 2500 times
-->
```

```
      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="740" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
***********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
0000000000000000000000000000000000000000000d66f
6a900080018c000403200104000f9a8b00c6018c00040320
0104000f000000c800000060018005540002c000400030004
00000000000500140002d00000030000000040000301be09ab
00000000003f4a7100038427c95a1ccad98c1dd400000000
a8aba1a0aba2a9a1b5f9edffb5aaa8a8a1983aacd9dea0ac
dbaddba2a9deacaf30333938343a31392d6175672d323030
3920202020202020202020202020206a7367697474696e
2020202020202020209a8c0004000000000000002ffffffff0b78
001110110000000000808080a5009990f60411101100000000
808080a500995a5a0811101100000000808080a500991c9f
0c111011c00080008080c0a500993c870008000000000ffff
ffffffffeb000011001100000000828280a50098f7c70411
001100000000808080a5009938c808110011000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
001100000000828280a50098f7c7041100110000000008282
80a500983d6b0811001100000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
***********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
```

```
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <run_scenario name="disturb"/>
            <repeat_scenario name="no_disturb" loop="4"/>
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="no_disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <delay><unsigned_intvalue value="50"/></delay>

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

</disturbance>
```

H6. QVM Pedal Sensor Disturbance (1:4) XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
5 us offset
every fourth cluster cycle
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
```

```
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="780" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
************************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000000d66f
5b4d008c018c000403200104000f334c00b7018c00040320
0104000f000000c800000060018005540O2c000400030004
000000000000500140002d000000300000004000301be09ab
00000000003f800d00038f60225c52fed98c1dd400000000
a3a0aaaba0a9a2aabef2e6f4bea1a3a3aa933aa7d2d5a6aa
aba7aaa9a0d5d5d730333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
202020202020202020982f0004000000000002fffffff0b78
001110110000000808080a5009990f60411101100000000
808080a500995a5a0811101100000000808080a500991c9f
0c111011c00080008080c0a500993c87000800000000ffff
ffffffffeb000011001100000000808080a50099f2640411
001100000000808080a5009938c8081100110000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
001100000000828280a50098f7c704110011000000008282
80a500983d6b08110011000000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
************************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
```

```
            <run_scenario name="disturb"/>
            <repeat_scenario name="no_disturb" loop="3"/>
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="no_disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <delay><unsigned_intvalue value="50"/></delay>

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

</disturbance>
```

H7. QVM Pedal Sensor Disturbance (1:3)

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
5 us offset
every other cluster cycle
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
```

```
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="750" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
************************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000000d66f
6a900080018c000403200104000f9a8b00c6018c00040320
0104000f000000c800000060018005540002c000400030004
000000000000500140002d000000300000004000301be09ab
00000000003f4a7100038427c95a1ccad98c1dd400000000
90939998939a91998dc1d5c78d92909099a03a94e1e69894
e597939a95e1e59230333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
202020202020209adc0004000000000002fffffff0b78
001110110000000808080a5009990f60411101100000000
808080a500995a5a0811101100000000808080a500991c9f
0c111011c00080008080c0a500993c87000800000000ffff
fffffffeb000011001100000000828280a50098f7c70411
001100000000808080a5009938c808110011000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
001100000000828280a50098f7c704110011000000008282
80a500983d6b08110011000000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
************************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <run_scenario name="disturb"/>
            <run_scenario name="no_disturb"/>
```

```
            <run_scenario name="no_disturb"/>
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="no_disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <delay><unsigned_intvalue value="50"/></delay>

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

</disturbance>
```

H8. QVM Pedal Sensor Disturbance (1:2) XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
5 us offset
every other cluster cycle
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
```

```
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="780" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
**********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000000d66f
5b4d008c018c0004032001040000f334c00b7018c00040320
0104000f000000c800000060018005540002c000400030004
000000000050014000 2d0000003000000040003 01be09ab
00000000003f800d00038f60225c52fed98c1dd400000000
a3a0aaaba0a9a2aabef2e6f4bea1a3a3aa933aa7d2d5a6aa
aba7aaa9a0d5d5d730333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
2020202020202020982f0004000000000002fffffff0b78
001110110000000808080a5009990f60411101100000000 0
808080a500995a5a0811101100000000808080a500991c9f
0c111011c00080008080c0a500993c87000800000000ffff
fffffffeb000011001100000000808080a50099f2640411
001100000000808080a5009938c8081100110000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
001100000000828280a50098f7c704110011000000008282
80a500983d6b0811001100000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
**********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <run_scenario name="disturb"/>
            <run_scenario name="no_disturb"/>
      </scenario>
```

```
      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="no_disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <delay><unsigned_intvalue value="50"/></delay>

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>
</disturbance>
```

H9. QVM Pedal Sensor Disturbance (3:4) XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
5 us offset
removes pedal signal 3 out of 4 rounds cluster cycle
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
```

```
                        <unsigned_intvalue value="740" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
***********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000d66f
6a900080018c000403200104000f9a8b00c6018c00040320
0104000f000000c8000000600180055400 2c000400030004
000000000000500140002d000000300000004000301be09ab
00000000003f4a7100038427c95a1ccad98c1dd400000000
90939998939a91998dc1d5c78d92909099a03a94e1e69894
e597939a95e1e59230333938343a31392d6175672d323030
39202020202020202020202020202020206a7367697474696e
2020202020202020209adc00040000000000002ffffffff0b78
00111011000000000808080a5009990f60411101100000000
808080a500995a5a08111011000000000808080a500991c9f
0c111011c00080008080c0a500993c8700080000000ffff
ffffffffeb00001100110000000000828280a50098f7c70411
00110000000000808080a5009938c8081100110000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
00110000000000828280a50098f7c7041100110000000008282
80a500983d6b08110011000000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
***********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="no_disturb"/>
      </scenario>
```

```
	<!-- Runs the disturbance scenario. -->
	<scenario name="disturb">
	<!--Specifies the slot where the disturbance starts.  -->
		<wait_for_roundslot roundslot="7" offset="5" />

		<!-- Output trigger for oscilloscope  -->
		<trigger name="TRIGGER_1" switch="ON" />

		<!-- Starts the disturbance. -->
		<run_definition name="const_signal" />

		<!-- Resets the output trigger signal to low.  -->
		<trigger name="TRIGGER_1" switch="OFF" />
	</scenario>

	<!-- Runs the disturbance scenario. -->
	<scenario name="no_disturb">
	<!--Specifies the slot where the disturbance starts.  -->
		<wait_for_roundslot roundslot="7" offset="5" />

		<!-- Output trigger for oscilloscope  -->
		<trigger name="TRIGGER_1" switch="ON" />

		<delay><unsigned_intvalue value="50"/></delay>

		<!-- Resets the output trigger signal to low.  -->
		<trigger name="TRIGGER_1" switch="OFF" />
	</scenario>

</disturbance>
```

H10. QVM Pedal Sensor Disturbance (4:5) XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
5 us offset
removes pedal signal 4 out of 5 rounds cluster cycle
sent on both channels
repeated 2500 times
-->

	<definition name="const_signal">
	<!-- Disturbs a slot by sending constant signals on channel A. --
>
		<source channel="BOTH">
			<!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
```

```
                  <duration>
                        <unsigned_intvalue value="740" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
***********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
00000000000000000000000000000000000000000000d66f
6a900080018c000403200104000f9a8b00c6018c00040320
0104000f000000c80000006001800554002c000400030004
00000000000500140002d000000300000004000301be09ab
00000000003f4a7100038427c95a1ccad98c1dd400000000
90939998939a91998dc1d5c78d92909099a03a94e1e69894
e597939a95e1e59230333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
20202020202020209adc0004000000000002ffffffff0b78
0011101100000000808080a5009990f60411101100000000
808080a500995a5a0811101100000000808080a500991c9f
0c111011c00080008080c0a500993c87000800000000ffff
ffffffffeb000011001100000000828280a50098f7c70411
001100000000808080a5009938c808110011000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
001100000000828280a50098f7c704110011000000008282
80a500983d6b08110011000000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
***********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="no_disturb"/>
```

```
	</scenario>

	<!-- Runs the disturbance scenario. -->
	<scenario name="disturb">
	<!--Specifies the slot where the disturbance starts.  -->
		<wait_for_roundslot roundslot="7" offset="5" />

		<!-- Output trigger for oscilloscope  -->
		<trigger name="TRIGGER_1" switch="ON" />

		<!-- Starts the disturbance. -->
		<run_definition name="const_signal" />

		<!-- Resets the output trigger signal to low.  -->
		<trigger name="TRIGGER_1" switch="OFF" />
	</scenario>

	<!-- Runs the disturbance scenario. -->
	<scenario name="no_disturb">
	<!--Specifies the slot where the disturbance starts.  -->
		<wait_for_roundslot roundslot="7" offset="5" />

		<!-- Output trigger for oscilloscope  -->
		<trigger name="TRIGGER_1" switch="ON" />

		<delay><unsigned_intvalue value="50"/></delay>

		<!-- Resets the output trigger signal to low.  -->
		<trigger name="TRIGGER_1" switch="OFF" />
	</scenario>

</disturbance>
```

H11. QVM Pedal Sensor Disturbance (5:6) XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
5 us offset
removes pedal signal 5 out of 6 rounds cluster cycle
sent on both channels
repeated 2500 times
-->

	<definition name="const_signal">
	<!-- Disturbs a slot by sending constant signals on channel A. --
>
		<source channel="BOTH">
```

```
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="740" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
**************************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000000d66f
6a900080018c0004032001040000f9a8b00c6018c00040320
0104000f000000c800000060018005540002c000400030004
000000000050014000 2d0000003000000040003 01be09ab
00000000003f4a7100038427c95a1ccad98c1dd400000000
90939998939a91998dc1d5c78d92909099a03a94e1e69894
e597939a95e1e59230333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
20202020202020209adc0004000000000002fffffff0b78
001110110000000808080a5009990f60411101100000000
808080a500995a5a0811101100000000808080a500991c9f
0c111011c00080008080c0a500993c87000800000000ffff
fffffffeb000011001100000000828280a50098f7c70411
001100000000808080a5009938c8081100110000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
001100000000828280a50098f7c704110011000000008282
80a500983d6b0811001100000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
**************************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
```

```
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="no_disturb"/>
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="no_disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <delay><unsigned_intvalue value="50"/></delay>

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

</disturbance>
```

H12. QVM Pedal Sensor Disturbance (10:11) XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
5 us offset
removes pedal signal 10 out of 11 rounds cluster cycle
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
```

```
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="740" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
************************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000000d66f
6a900080018c000403200104000f9a8b00c6018c00040320
0104000f000000c800000060018005540O2c000400030004
000000000000500140002d000000300000004000301be09ab
00000000003f4a7100038427c95a1ccad98c1dd400000000
90939998939a91998dc1d5c78d92909099a03a94e1e69894
e597939a95e1e59230333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
202020202020209adc0004000000000002fffffff0b78
001110110000000808080a5009990f60411101100000000
808080a500995a5a0811101100000000808080a500991c9f
0c111011c00080008080c0a500993c87000800000000ffff
fffffffeb000011001100000000828280a50098f7c70411
001100000000808080a5009938c808110011000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
001100000000828280a50098f7c704110011000000008282
80a500983d6b08110011000000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
************************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
```

```
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="no_disturb"/>
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="no_disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <delay><unsigned_intvalue value="50"/></delay>

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

</disturbance>
```

H13. QVM Pedal Sensor Disturbance (20:21) XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
```

```
5 us offset
removes pedal signal 10 out of 11 rounds cluster cycle
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="740" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
***********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
00000000000000000000000000000000000000000000d66f
6a900080018c000403200104000f9a8b00c6018c00040320
0104000f000000c8000000600180055 4002c000400030004
000000000000500140002d0000003000000004000301be09ab
00000000003f4a7100038427c95a1ccad98c1dd400000000
90939998939a91998dc1d5c78d92909099a03a94e1e69894
e597939a95e1e59230333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
2020202020202020209adc0004000000000000002ffffffff0b78
0011101100000000808080a5009990f60411101100000000
808080a500995a5a081110110000000080808 0a500991c9f
0c111011c00080008080c0a500993c87000800000000ffff
ffffffffeb00001100110000000828280a50098f7c70411
0011000000000808080a5009938c80811001100000000 8282
80a500987bae0c110011c00000008080c0a50099f5960011
0011000000000828280a50098f7c70411001100000000 8282
80a500983d6b0811001100000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
```

```
<!--
************************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="no_disturb"/>
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="no_disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <delay><unsigned_intvalue value="50"/></delay>
```

```
            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

</disturbance>
```

H14. QVM Pedal Sensor Disturbance (40:41) XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
5 us offset
removes pedal signal 10 out of 11 rounds cluster cycle
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="740" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
************************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
00000000000000000000000000000000000000000000d66f
6a900080018c000403200104000f9a8b00c6018c00040320
0104000f000000c80000006001800554002c000400030004
00000000000500140002d00000030000000400030 1be09ab
00000000003f4a7100038427c95a1ccad98c1dd400000000
90939998939a91998dc1d5c78d92909099a03a94e1e69894
e597939a95e1e59230333938343a31392d6175672d323030
```

```
392020202020202020202020202020206a7367697474696e
20202020202020209adc0004000000000002ffffffff0b78
001110110000000000808080a5009990f604111011000000000
808080a500995a5a081110110000000000808080a500991c9f
0c111011c00080008080c0a500993c87000800000000ffff
ffffffffeb00001100110000000000828280a50098f7c70411
0011000000000808080a5009938c80811001100000000008282
80a500987bae0c110011c00000008080c0a50099f5960011
0011000000000828280a50098f7c70411001100000000008282
80a500983d6b081100110000000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
***********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
```

```
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="disturb"/>
            <run_scenario name="no_disturb"/>
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="no_disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <delay><unsigned_intvalue value="50"/></delay>

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

</disturbance>
```

H15. QVM Pedal Sensor Disturbance Channel A XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
```

```
disturbs roundslot 7 of qvm
780 us low voltage signal (pedal sensor)
5 us offset
every fourth cluster cycle
sent on chan A
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="CH_A">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="780" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
**************************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000000d66f
5b4d008c018c00040320010400 0f334c00b7018c00040320
0104000f000000c800000060018005540 02c000400030004
000000000050014000 2d0000003000000040003 01be09ab
00000000003f800d00038f60225c52fed98c1dd400000000
a3a0aaaba0a9a2aabef2e6f4bea1a3a3aa933aa7d2d5a6aa
aba7aaa9a0d5d5d730333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
2020202020202020982f0004000000000002fffffff0b78
001110110000000808080a5009990f60411101100000000
808080a500995a5a0811101100000000808080a500991c9f
0c111011c00080008080c0a500993c87000800000000ffff
fffffffeb000011001100000000808080a50099f2640411
001100000000808080a5009938c8081100110000000 08282
80a500987bae0c110011c00000008080c0a50099f5960011
001100000000828280a50098f7c704110011000000008282
80a500983d6b0811001100000000828280a500987bae0c11
0011c05080538282c0a50098eb6f
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
```

```
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
*****************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <run_scenario name="disturb"/>
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="7" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>

</disturbance>
```

H16. FVM Node 1 Disturbance XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs node 2 of fvm
750 us low voltage signal (wheel 1)
5 us offset
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="750" />
                  </duration>
                  <signal_low />
```

```
            </source>

      </definition>
<!--
**********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000000d66f
d0050066018c0004032001040000f0823006f018c00040320
0104000f000000c8000000600180052c002c000400030004
000000000500140002d000000300000004000301af0965
00000000003f549e0003d14382369ff0d98c1dd400000000
e4e7edece7eee5edf9b5a1b3f9e6e4e4edd43ae09592e0e1
91e290eee0e1929230333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
2020202020202020c24700040000000000002fffffff0b78
0011101100000000808080a10095702304111011000000000
808080a00094171f0811101100000000808080a0009451da
0c111011c00080008080c09f00934bde000800000000ffff
fffffffeb000011001108000805848480a10092e35b0411
0011080a080f848480a000915e850811001108140819848 4
80a000917a280c110011c00000008080c09f009382cf0011
001100000000828280a1009417120411001100000008282
80a000931d2408110011081e0823848480a000917d2c0c11
0011c05080538282c09f00929c36
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <run_scenario name="do"/>
      </scenario>
<!--
**********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <repeat_scenario name="disturb" loop="2500"/>

      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_slot slot="0" offset="5" />
```

```
            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>
</disturbance>
```

H17. FVM Node 2 Disturbance XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs node 2 of fvm
750 us low voltage signal (wheel 2)
5 us offset
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="750" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
**********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000000d66f
d0050066018c0004032001040000f0823006f018c00040320
0104000f000000c8000000600180052c002c000400030004
000000000050014000 2d0000003000000040003 01af0965
00000000003f549e0003d14382369ff0d98c1dd400000000
```

```
e4e7edece7eee5edf9b5a1b3f9e6e4e4edd43ae09592e0e1
91e290eee0e1929230333938343a31392d6175672d323030
39202020202020202020202020202020206a7367697474696e
2020202020202020c247000400000000000002ffffffff0b78
00111011000000000808080a10095702304111011000000000
808080a00094171f0811101100000000808080a0009451da
0c111011c00080008080c09f00934bde000800000000ffff
ffffffffeb000011001108000805848480a10092e35b0411
0011080a080f848480a000915e85081100110814081981848 4
80a000917a280c110011c00000008080c09f009382cf0011
00110000000828280a10094171204110011000000008282
80a000931d2408110011081e0823848480a000917d2c0c11
0011c05080538282c09f00929c36
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <run_scenario name="do"/>
      </scenario>
<!--
**********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <repeat_scenario name="disturb" loop="2500"/>

      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_slot slot="1" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>
</disturbance>
```

H18. FVM Node 3 Disturbance XML

```
<disturbance start_scenario="set_to_run" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
```

```
disturbs node 3 of fvm
730 us low voltage signal (wheel 3 and 4)
5 us offset
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="730" />
                  </duration>
                  <signal_low />
            </source>

      </definition>
<!--
**********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
00000000000000000000000000000000000000000000d66f
d0050066018c00040320010400 0f0823006f018c00040320
0104000f000000c8000000600180052c002c000400030004
000000000050014000 2d0000003000000040003 01af0965
00000000003f549e0003d14382369ff0d98c1dd400000000
e4e7edece7eee5edf9b5a1b3f9e6e4e4edd43ae09592e0e1
91e290eee0e1929230333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
2020202020202020c2470004000000000002ffffffff0b78
0011101100000000808080a1009570230411101100000000
808080a00094171f0811101100000000808080a0009451da
0c111011c00080008080c09f00934bde000800000000ffff
ffffffffeb000011001108000805848480a10092e35b0411
0011080a080f848480a000915e85081100110814081984 84
80a000917a280c110011c00000008080c09f009382cf0011
0011000000 0828280a1009417120411001100000000 8282
80a000931d2408110011081e0823848480a000917d2c0c11
0011c05080538282c09f00929c36
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <run_scenario name="do"/>
```

```
      </scenario>
<!--
**********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <repeat_scenario name="disturb" loop="2500"/>

      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_slot slot="2" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>
</disturbance>
```

H19. FVM Pedal Sensor Disturbance XML

```
<disturbance start_scenario="set_to_run"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Distu_XML_Schema.xsd">
<!--
disturbs roundslot 1 and 2 of fvm
770 us low voltage signal (pedal sensor)
5 us offset
every other cluster cycle
sent on both channels
repeated 2500 times
-->

      <definition name="const_signal">
      <!-- Disturbs a slot by sending constant signals on channel A. --
>
            <source channel="BOTH">
                  <!-- Sends a constant differential high signal on
both channels for 100 microseconds. -->
                  <duration>
                        <unsigned_intvalue value="700" />
                  </duration>
                  <signal_low />
```

```
            </source>

      </definition>
<!--
**********************************************************************
-->
<!-- Configures all the settings needed to run the disturbance (relay,
MEDL, etc.). -->
      <scenario name="set_to_run">

<!-- Generated File -->
<medl>
020238000018381800283840003138710027389800470000
000000000000000000000000000000000000000000000d66f
d0050066018c0004032001040000f0823006f018c00040320
0104000f000000c8000000600180052c002c000400030004
000000000000500140002d000000300000004000301af0965
000000000003f549e0003d14382369ff0d98c1dd400000000
e4e7edece7eee5edf9b5a1b3f9e6e4e4edd43ae09592e0e1
91e290eee0e1929230333938343a31392d6175672d323030
392020202020202020202020202020206a7367697474696e
2020202020202020c24700040000000000002fffffff0b78
001110110000000808080a10095702304111011000000000
808080a00094171f0811101100000000808080a0009451da
0c111011c00080008080c09f00934bde000800000000ffff
fffffffeb000011001108000805848480a10092e35b0411
0011080a080f848480a000915e850811001108140819848 4
80a000917a280c110011c00000008080c09f009382cf0011
001100000000828280a1009417120411001100000008282
80a000931d2408110011081e0823848480a000917d2c0c11
0011c05080538282c09f00929c36
</medl>
            <reboot />
            <!-- Specifies the time slot where the protocol performs
the clique avoidance algorithm (optional, default roundslot = "0"). -->
            <blackout_detection roundslot="0" />
            <repeat_scenario name="do" loop="2500"/>
      </scenario>
<!--
**********************************************************************
-->
      <!-- Runs the disturbance scenario. -->
      <scenario name="do">
            <!-- destroy to consecutive rounds to trigger ack failure -
->
            <run_scenario name="disturb_a"/>
            <run_scenario name="no_disturb_a"/>
      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="disturb_a">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="0" offset="5" />
```

```
            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />

            <wait_for_roundslot roundslot="1" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <!-- Starts the disturbance. -->
            <run_definition name="const_signal" />

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />

      </scenario>

      <!-- Runs the disturbance scenario. -->
      <scenario name="no_disturb_a">
      <!--Specifies the slot where the disturbance starts.  -->
            <wait_for_roundslot roundslot="0" offset="5" />

            <!-- Output trigger for oscilloscope  -->
            <trigger name="TRIGGER_1" switch="ON" />

            <delay><unsigned_intvalue value="50"/></delay>

            <!-- Resets the output trigger signal to low.  -->
            <trigger name="TRIGGER_1" switch="OFF" />
      </scenario>
</disturbance>
```

Appendix I. Special Development Platform Considerations

This appendix outlines the current hurdles to application development for the TTTech Development cluster using MATLAB. As of December 2009, all of the major software components have valid licenses that are installed and working except for one, the Wind River Diab compiler, version 5.1.7. FGCU does possess a valid license for version 5.7 of the Diab compiler, but this has not proved easy to integrate into the existing MATLAB, Real-time Code Generator and TTTech tools. Until a more permanent solution has been achieved, there does exists a workaround to continue using the evaluation license for the Diab compiler that expired in November 2009. After logging into the development PC for the first time, make sure the system clock is set to the current date and time. Open MATLAB and perform a complete build of your project. This will fail at the code generation step with a license expired error. Minimize MATLAB and set the system clock to before November 17, 2009. Your build should now complete without error. Note that the system needs to perform that initial failed code generation under the current date and time, in order to locate the license file. If this step is not performed, you will receive a different error in which MATLAB fails to locate the license entirely.

A second problem that is frequently encountered during development and testing is the lockup of TTP View. Many times during active monitoring of the cluster, if a disturbance is created that crosses round slot boundaries, TTP View will freeze. The only remedy is to close TTP View and restart that monitoring session. In addition, you must also reboot the monitoring node in the cluster. Failure to reboot the monitoring node will cause an error in TTP View during the next attempt to record the data being transmitted on the bus, even after TTP View has been restarted.

Finally, take note that if you are unable to transmit a file to the disturbance node, first reboot the node (by unplugging the device). Many times, after repeated and prolonged communications, the node will enter a failure state in which it is unable to accept new, properly formatted XML. Cycling the power corrects this behavior.