Freescale Semiconductor
Application Note

AN2396
Rev. 1.0, 12/2005

Servo Motor Control Application on a Local Interconnect Network (LIN)

By **Matt Ruff**
8/16 Bit Division Systems Engineering
Austin, Texas

1.0 Introduction

The main purpose of this application note is to show one way to implement a closed-loop control position servo motor application which utilizes the Local Interconnect Network (LIN) to allow a series of similar motors to be connected together and controlled from a central LIN master controller. The application note explains a basic design which uses a DC brush motor, feedback potentiometer, a Freescale M68HC08 microcontroller, and SmartMOS power components. The software design and messaging strategy are explained step by step to aid in the development of an entire LIN subnetwork of servo motors, complete with full hardware diagnostics capability.

Contents

1.1 What is the Distinction Between Open-Loop and Closed-Loop Control?

Without getting into a detailed explanation of the specifics of control theory, it is possible to make a simple distinction between open-loop and closed-loop systems.

An open-loop system operates with no feedback from the object being controlled. This is sort of a “fire and forget” approach to control. An input stimulus is provided and the controller commands the system to go to a particular location, speed, whatever, and hopes that the system responds accordingly. There is no information from the system under control to indicate that it even received the command, much less acted upon it.

The key to a closed-loop control system is the introduction of feedback. If speed is being controlled, a measure of the current speed is provided back to the controller, allowing it to adjust its commands as the system responds to the commands. Likewise is true with position. Think of a gymnast on a balance beam. They are constantly commanding their muscles to adjust the pressure and position of their feet to maintain a set position. This is done based on inputs from vision, sense of balance, and even tactile feedback from contact with the beam itself. This is a perfect example of a closed-loop system.

The performance of a closed-loop system is partially a function of the speed at which the feedback is returned to the controller. This closing of the loop will always take a set amount of time, and the longer that time is the less responsive the controller will be to fast changing conditions. For example, if the gymnasts are very tired they are not able to perform as adeptly due to slow response to the feedback received and as conditions change more rapidly they are more likely to fall off the bar.

Depending on the performance requirements of the application, either a closed-loop or open-loop control system can be used to control motor position, speed, or other similar application.

1.2 What is a Servo Motor?

Motors come in many different varieties for different applications. The term “servo motor” doesn’t really apply to the motor itself, but rather the way in which the motor is used and controlled. In a position servo motor application (henceforth just ‘servo motor’), the idea is to hold the target load (generally attached to the motor shaft through a series of gears for speed and torque adjustment) in a given position.

To accomplish a servo motor function, positioning information must be obtained from the output of the motor to provide feedback for the control system. This can be in the form of a potentiometer attached somewhere in the gear train, a hall effect sensor monitoring passing teeth on a metal gear, an encoder (optical or magnetic) mounted directly to the motor, or any other such sensor which can provide position feedback of the motor shaft or connected portion of the gear train.

It is also possible to determine motor shaft position by counting the commutation pulses on the terminals of the motor. Freescale SMARTMOS H-Bridge drivers do have current re-copy capability, which allows a fraction of the load current to be output through a reference resistor and measured with an analog-to-digital converter (ADC). In this way, it might be possible to monitor the load current and detect variations in the load current which correspond with the commutation of the brushes in the motor. However, this method requires additional software and hardware complexity which do not justify their inclusion, given that encoders are just as effective and much simpler to implement. This modification allows streamlining of the logic needed at the load to drive it.

1.3 Where would I use a DC Brush Position Servo Motor?

DC brush position servo motors can be used in many different applications. A partial list of possible applications includes:

- Automotive Market:
  - Power mirror positioning
  - Power seats positioning motors
  - Power door and trunk lock mechanisms
  - Windshield wiper motors
  - Heating, Ventilation, and Air Conditioning (HVAC) vent controls
  - Power sliding door, sunroof, and convertible top actuators
  - Headlight positioning and levelling actuators
- Industrial and Consumer Markets:
  - Proportioning valves for gasses and liquids
  - Paper and materials handling equipment
  - HVAC ventilation control
  - Entertainment equipment (powered, remotely controlled volume controls for audio receivers and mixers)

1.4 How Much Control do I Need?

Most automotive body electronics servo motor applications are extremely slow by control systems standards. Response times are often measured in seconds rather than milliseconds, which means that the demands on the control loop are relatively light. Complex high-performance control algorithms such as

Proportional, Integral, Differential (PID) control algorithms are not generally necessary in these types of systems. For more information on digital implementation of PID control algorithms, refer to *16-Bit DSP Servo Control With the MC68HC16Z1* (Freescale document order number AN1213).

Since a simple control algorithm should work, it is best to begin with the simplest solution possible and then add complexity only if needed. In the case of a simple position servo motor in a body system, polling a simple position feedback sensor (potentiometer, encoder, etc.) until the desired position is reached should suffice, provided the polling is fast enough to prevent overshooting the targeted value.

1.5 Where do I Close the Loop? (Network or Local)

Depending on the performance requirements of your system and the performance of your network, you can “close the loop” of feedback at either the load itself, or through the network itself. This means that you either locate the control algorithms locally, at the servo motor, or you pass all the command and feedback data remotely through the network to a central controller module.

Ultimately, the choice of closing the control loop locally or over the network is a function of many variables, including required performance of the servo motor, network speed and architecture, system level cost of the solution, and the number of motors to be controlled over the network.

1.6 How would Network Speed and Protocol Affect Control Performance?

Consider the case of closing the loop through the network. By inserting the network into the control and feedback loops, time delay is introduced. Depending on the speed and nature of the network used, the time delays introduced can become quite significant and could result in destabilization of or oscillatory behavior in the system.

The use of networks which have faster data throughput allows the sampling frequency of the control to increase. Faster updates of the feedback data through the network allow faster sampling rates and therefore increased system response time.

Simply choosing a network with a higher baud rate will not always achieve faster data throughput. The protocol used is a large determining factor. The real issue is how much and how fast that ***data*** can be passed between the controlling node and the node being controlled. In the case of a protocol like Controller Area Network (CAN), message identifiers indicate the priority of a message and dictate bus arbitration. Higher priority messages take precedence on the bus and prevent or delay lower priority messages from being transmitted. Without careful choice of identifiers (often not even an option for automotive system suppliers), this could result in delays or prevention of reception of valuable command input or feedback information. Unpredictability of time delays will result in error in the accuracy and dependability of the feedback data. The longer the delay that is introduced, the “older” the feedback reading becomes, and therefore it becomes much less reliable in a dynamic system.

The effects of bus arbitration can be reduced (but not eliminated) by increasing the speed of the network. If the choice to use a higher-level protocol such as CAN has been made, a CAN capable microcontroller unit (MCU) is required at the motor to provide the communications and could likely handle the motor control as well.

Consider using a network like the Local Interconnect Network (LIN). LIN, albeit much slower than CAN (20 kbps maximum), does provide a time deterministic method of data transfer. LIN also does not require advanced silicon to implement, such as dedicated CAN hardware. It is even possible to implement LIN communication using very simple state machine devices. If the motor control functions can be handled by a central node in the network (the master node in a LIN network), then the only job required at the motor slave node is to decode the network messages, drive the motor according to this data, and provide the feedback data back out to the network when requested.

Using LIN will definitely limit the data throughput of the network, due to its slower speeds, but the resulting simplification of the silicon required at the motors opens up new possibilities for lower performance control architectures. The problem is that as more motors are added to the system, the performance demands on the network change dramatically.

1.7 How would I Choose and Configure the Right Network?

A great many factors affect the choice of a network to communicate between electronic control units (ECU). One factor might simply be the physical location of the ECUs. Very distant nodes might not be well suited to a network requiring a single wire transmission medium. The long wire runs would be more susceptible to signal degradation and noise due to increased resistance and capacitance of the wires.

If the network is located in a hazardous environment, such as inside a tank of explosive or flammable materials, perhaps opto-isolation is called for, using optical fiber and light to transmit data rather than wire.

Another consideration with respect to physical transmission medium is the operating environment of the network. If the network is to be located in an automobile, for instance, ElectroMagnetic Compatibility (EMC) is extremely important. The wiring that serves to pass network messages around the car also serves as a radiating antenna, as well as a receiving antenna to pick up any noise which might be in the air.

EMC is a way to measure how much electromagnetic energy is emitted by the components. If the network is emitting noise in the wrong frequency spectra, it can even interfere with reception of AM/FM radio reception. Customers don’t respond well to a ‘buzzing’ noise while in the middle of listening to their favorite tunes, so auto makers are keenly aware of EMC issues. The physical medium of transmission, as well as how the data is encoded on that medium affect the amount of EMI generated.

Susceptibility measures how well the components deal with noise injected into the system from outside influences. In a gasoline powered car, this can be considerable due to the spark noise from the ignition system.

Safety critical systems need to take the above into account, but also might require extremely good data error checking, such as the Cyclical Redundancy Check (CRC) found in Controller Area Network (CAN). Even better data integrity measures might be required in safety systems, such as redundant wiring and components, as well as time-triggered mechanisms which ensure timely delivery of information in the system.

As the title of this document suggests, LIN will be the network of choice for this application. But why?

The most straightforward target application for many simple position servo motors in an automotive environment is the heating, ventilation, and air conditioning (HVAC) system. The actuators used to control airflow in the vehicle are sometimes extremely basic, serving the function of a “switch” for the air stream with only an on and off type control. Other, more complex actuators have positioning feedback and can be adjusted to precise positions, allowing proportioning of airflow, such as controlling the relative amounts of air from the hot and cold air supplies.

In an HVAC system, the response times required by the actuators are typically extremely long, allowing the designer to implement very basic control algorithms. Keeping the controls and actuators simple allows the cost of the actuators to be kept to a minimum. These factors make LIN an optimum network choice for this system, as it is designed to be an open standard, inexpensive sensor/actuator network which can be implemented using existing silicon solutions.

1.8 Again, where do I Close the Loop (Network or Local)?

In light of each specific application, it is necessary to revisit the question of where to close the control loop. Taking the application of an HVAC system, the first response is to think that it would be best to close the loop through the network. After all, the actuator response times are already slow and it would greatly simplify the complexity of each actuator design. The problem with closing the loop here lies in the problems introduced to the system design.

Consider the position of an automobile maker, who might have three, four, or perhaps more levels of system complexity for the HVAC system. For each vehicle platform the manufacturer produces, there could be different options in complexity of the HVAC system. The higher end vehicles will have more sophisticated features, such as automatic temperature controls, oscillating vent outputs, instant-on heater, and multiple zone controls for different parts of the passenger compartment.

So what effect does this have on the question at hand? It means that if the control loop is closed through the network, it introduces two distinct problems for the system designer.

First, the response times for each node will increase as more nodes, hence more network communications, are added to the system. It can be seen that if sufficient numbers of nodes are added to the system that require constant control, it is possible that the response times can become unacceptably slow. This also means that messaging might have to be custom tailored for each configuration of the system, requiring a great number of variations in implementations. This leads to the second point, which relates to cross platform compatibility and standardization for the auto maker.

A great deal of money is spent by automotive manufacturers to maintain multiple versions of essentially the same system. If the control loops for HVAC actuators are closed through the network and different messaging and programming is required for each system in each vehicle platform, inventories must be kept of all the different variations, software must be maintained for each of these different variations, etc. In contrast, if the loop is closed locally at each actuator node, the network architecture can be standardized not only through the different option levels with many different numbers of nodes, but also across vehicle platforms, allowing the car maker to use one strategy for all their vehicle lines.

With locally closed control loops, the HVAC system becomes completely modularized and standardized. To increase the complexity of the system, it is only needed that the necessary components be added to the network and perhaps the master node software changed to reflect the new hardware addition (e.g., adding a temperature sensor to the network to facilitate automatic temperature control). For these reasons, the control loops in this system will be closed locally at each actuator device.

1.9 Are There Other Factors that Affect the Control System?

There are other factors to consider in designing the control system for a position servo motor application that are worth mentioning. For the sake of brevity, these will not be discussed in detail here, but are mentioned for consideration.

The first factor is sticking friction of the actuator. When the actuator is at rest, extra force is required to initiate motion. Once the actuator begins to move, it is also possible that it may jump suddenly, depending on how tight the mechanical tolerances are on the actuator. This rapid rate of change should be considered when determining response times required of the control system.

The ratio of output movement to feedback data acquisition is another consideration which greatly affects the accuracy and response of the control system. For example, in a basic actuator the motor and output shaft may be joined by a series of gears and the feedback mechanism might be a potentiometer connected to another gear in the system. The ratio of movement of the output shaft to the movement of the potentiometer shaft is required to relate rate of change of the actuator output to the rate of change of the feedback.

1.10 How do I Choose a Logical Messaging Strategy — Extend the Control Method?

There are many schools of thought on how to design/devise a messaging strategy for a LIN system. Since the HVAC system is relatively simple and self-contained, it may seem simple at first glance to define a messaging strategy. Defining a message strategy that works isn’t the real challenge here; the challenge is defining a messaging strategy that works while allowing the system costs and complexity to be reduced without compromising system flexibility.

The key to reducing system costs is the reduction of slave node cost. In any given LIN system, there is only one master and up to 15 slave devices. The master node must be an MCU with a crystal or other very stable time base, so costs can only be cut so far at the master node. The slave devices; however, can have much lower tolerances on clocking and are simple enough to implement in a variety of simple mechatronic and small microcontroller solutions.

In order to use these types of very simple devices at slave nodes, the messaging should be as simple and straightforward as possible. In the case of this application, we are looking at one specific type of slave device, a position servo motor.

The first step in determining the messaging strategy is to determine what information must flow into and out of each slave device (in this case only one slave device). The main messaging functions for the position servo application can be defined as:

- Input Command — master to slave only:
  - — Target position (8-bit value), which corresponds to the potentiometer reading which will be read when the servo has reached its desired position.
- Node Status Request — master sends ID, node responds with:
  - — Current position (8-bit value)
  - — Current command being acted upon
  - — SMOS load diagnostics
  - — LIN messaging error codes (refer to *LIN Protocol Specification*)
- NVM Programming (upload) command — master to slave only:
  - — Node address (uniquely identifies each node), index to data field of input command — gives 'slot number' to node.
  - — Input command IDs to recognize

Now that we have laid out the basic information flow, we must lock down specific identifiers, bits, etc. of these messages to truly define the messages themselves. Refer to **Appendix A — LIN HVAC System Demonstration Messaging Strategy** and **Appendix B — LIN HVAC System Demonstration Messaging Strategy — Configuration Language Description File** for details on the complete messaging strategy.

1.11 How do I use a LIN to Control a DC Brush Position Servo Motor?

Now that most of the major issues have been discussed regarding how to control a DC brush position servo motor, it is time to actually design this control system.

The hardware for this example was created and designed by another company, so the physical constraints of the mechanics were fixed from the beginning. These constraints can be adjusted to suit most any similar system. Simply, the actuator to be controlled consists of a DC brush motor whose output is connected to a gear train. The output shaft of the actuator is about 950 times slower than the motor shaft. Position feedback is obtained from a 10 kohm potentiometer, with a gear that engages the output shaft with a 2:1 gear ratio.

The control hardware used is the X05 Medium Power demonstration unit from Freescale's Mechatronics group. The X05 is a member of Freescale's Smartconnector Family combining a high performance 8-bit HC08 microcontroller with a SmarTMOS power technology. It includes all the necessary features to meet the requirements for LIN applications. This includes features like an RC oscillator with ±2% accuracy, an enhanced SCI module to allow 13-bit break detection capability, FLASH technology, and the appropriate power stages to drive loads (e.g., lamps, stepper motor, DC motors…). It also provides the necessary inputs for reading the status of Hall sensors or potentiometers. The power H-Bridge stages have protection and diagnostics capabilities.

The control code from this application note could easily be ported to another microcontroller with a suitable power stage and LIN physical interface (such as the MC33399), but the X05 provides a quick and easy platform for developing the algorithm without getting into hardware details which are beyond the scope of this paper.

Figure 1 shows a simplified version of the hardware to provide a reference of the basic components of the hardware.

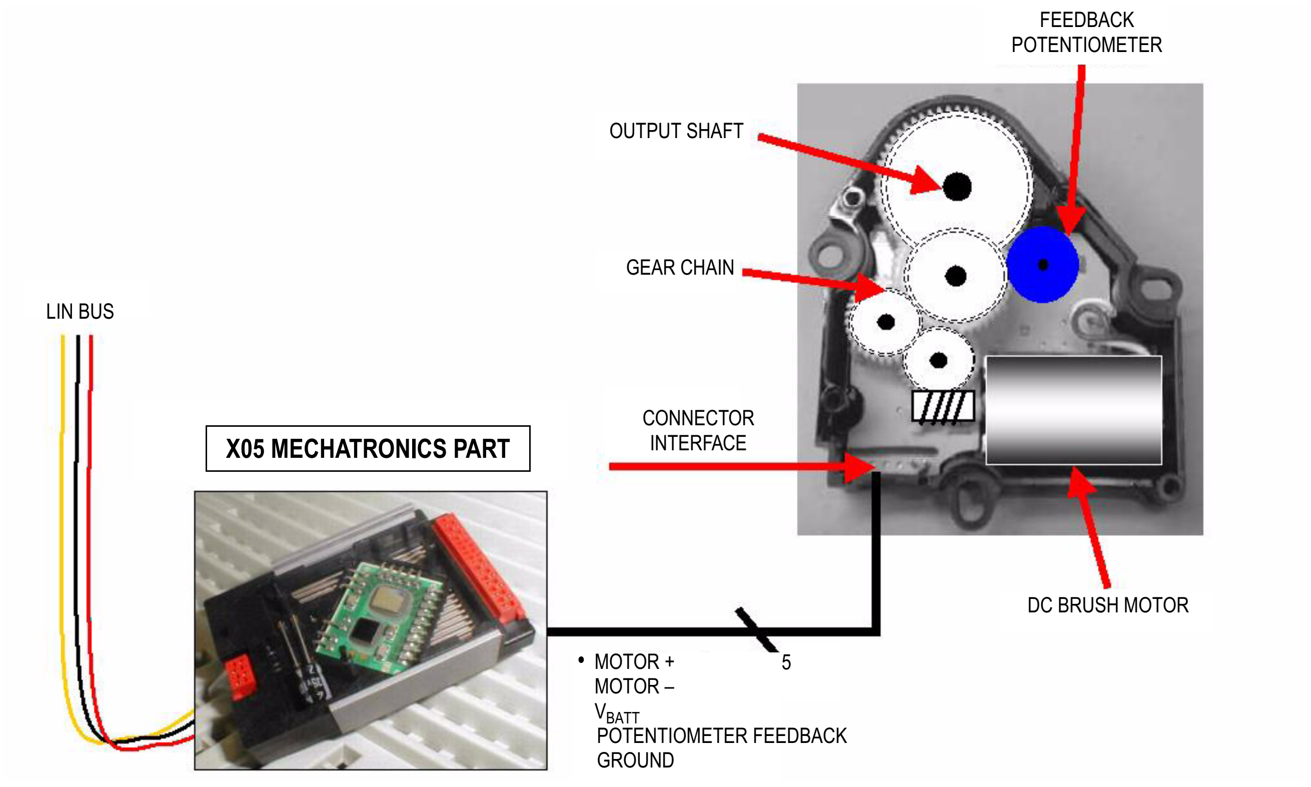

Figure 1. Hardware Connection Block Diagram

To begin designing the application code, it’s necessary to create a basic state machine to show the various stages of the application operation. This exercise gives a high level view of what the actuator can and cannot do. Refer to **Figure 2**.

One example of what the control system can do is to move to the “MOTOR CLOCKWISE” state from the “MOTOR STOP” state. This makes sense, as the motor should be able to be started in either direction from a stopped condition.

However, if the command is to reverse the direction of the motor, it is highly inadvisable to immediately reverse the voltage on a motor to attempt to drive it in the opposite direction. The inductive properties of a DC brush motor mean that the motor is still trying to drive current in the previous direction. Very large reverse voltages can form (usually called “back EMF”) which can cause permanent damage to components in the system. Notice that in the state diagram, it is impossible to go from the “MOTOR CLOCKWISE” state to the “MOTOR COUNTER-CLOCKWISE” state or vice-versa, without first stopping the motor. In this way, the state diagram forces the control system to avoid the potentially dangerous state change of reversing the motor without stopping.

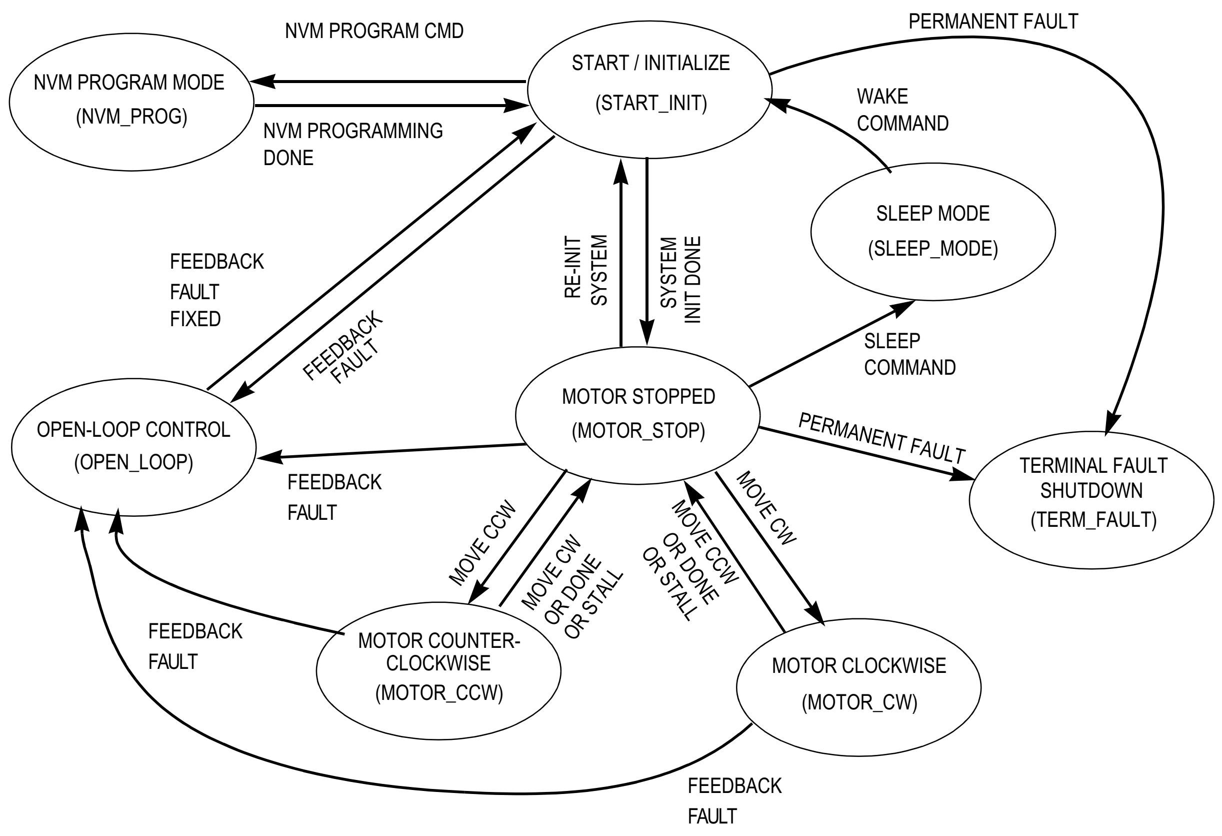

Figure 2. Closed-Loop Control Actuator State Diagram

State transition timing will dictate performance parameters of the system. For example, the speed with which the control system is able to move from a motor stopped state to a motor rotating state governs how fast the actual motor transitions from one state to another. In the application code, these state transitions could simply be placed in a continuously updating loop. This would result in variations in state transitions based on central processor unit (CPU) activity and CPU base clock speed. In order to better regulate the timing of these state transitions, a state clock will be implemented, based on a timer interrupt, which will ensure that state transitions occur on a predictable time base. This is identical to the way in which hardware based state machines operate.

It is necessary to describe specifically what occurs in each state, in order that the behavior of that state is fully understood. SMOS errors are checked for in all states and do not result in a state change, only an update of LIN messaging and a sometimes a difference in behavior of the state. For example, if the motor is supposed to be turning, but an SMOS over-voltage condition exists, the SMOS component automatically cuts power to the load. No change of state is needed, but the errors are reported through LIN messaging.

The following is a detailed description of each state:

- **START_INIT —** The controller defaults to this state upon power-up and will send the motor counter-clockwise to its home (or zero) location. Lacking any input from the LIN network, the motor will remain in this location.
- **MOTOR_STOP —** The controller cuts power to the motor. If the motor was previously turning, then a delay is inserted before continued processing in order to allow the energy in the motor to be dissipated. For this reason, this state must be entered when reversing the motor direction.
- **MOTOR_CW —** The controller turns on the H-bridges to turn the motor clockwise. If the motor is approaching the end of the range of travel, the power to the H-bridges is reduced to minimize the force potentially applied to the physical stop. This has the added effect of decelerating the motor near the stop. Certain SMOS errors will automatically cut power to the H-bridges (e.g., over temperature, over voltage) to prevent damage to components, which might result in the failure of the motor to turn. The controller will remain in this state, but will indicate the fault in LIN messaging and the motor will not turn.
- **MOTOR_CCW —** Same as MOTOR_CW, but opposite direction of rotation and power reduction is still performed, but at other end of range of motion.
- **SLEEP_MODE** (unimplemented) — Puts the controller into low-power sleep mode upon reception of the correct LIN message.
- **OPEN_LOOP** (unimplemented) — If a physical fault is introduced to the feedback loop, such as a discontinuity in the feedback potentiometer, the controller would enter this state. The controller then simply responds to clockwise or counterclockwise commands, and reports whatever value it does see on the potentiometer to the LIN master. This allows the motor to be controlled remotely over the LIN network, albeit at reduced response rates governed by the speed of LIN messaging.
- **NVM_PROG** (unimplemented) — This state is reserved for a bootloader type function, since Freescale FLASH microcontrollers have the ability to be reprogrammed while in the vehicle, this state could be utilized to update the software in a node without removing it from the vehicle.
- **TERM_FAULT** (unimplemented) — This state is reserved for cases when a node has become damaged or faulted to the point that it can no longer function properly and must be shut down.

Now that the basic states are understood, greater detail can be examined on specifically how the control algorithm works, how it moves from state to state, what information is stored, etc.

The state machine is switched based on a timer routine which fires periodically and based on the current state, checks inputs and conditions to determine the next state of the controller. The main program processes the current state, updating messaging, and outputs as needed. To maintain mutual exclusion (often called MUTEX in operating system terminology) of variables, especially the state variable, a blocking semaphore is used to tell the software that the current state is being processed. The next state change can only take effect between cycles of the main routine which is processing the current state. The update rate of the state machine can be controlled by changing the frequency of the timer routine, limited only by the worst case execution time of the loop which processes the current state in main.

As a next step, consider the data flow of the application. This is a description of where information is stored and how it is moved around in the application software. Data flow diagrams are extremely useful in understanding how different portions of the software interact, as well as defining all the specific data storage areas needed to create the application. To facilitate this, the naming conventions to be used in your coding should also be followed when drafting the data flow diagram. The data flow diagram for this application is shown in **Figure 3**. Much of the storage for this application revolves around the LIN messages which control incoming commands and outgoing status messages. This diagram does not concern itself with what occurs in these processes, only the information which enters and leaves them and where that information is stored.

The cloud bubbles indicate some process which manipulates or generates the data in the system. Data storage locations are indicated by text with horizontal lines above and below and as was mentioned use the exact names of the data storage structures used in the code itself (e.g., MOTOR_DIAG_SET). In some cases, these storage areas are simply registers in the MCU memory map, in others, they are the LIN message buffers created in RAM for sending and receiving LIN messages. Data flow is shown as arrows with accompanying text to explain what data is being moved. All text shown in the `COURIER` font corresponds to a variable or constant name used in the C code.

The control flow must allow all of the state changes shown **Figure 2**, and prevent any changes in state which are not shown. The basic control algorithm is shown in **Figure 4**. This diagram gives some insight into more of the specifics of what occurs to process a given state, updating variables, etc. It is really secondary to the state diagram and data flow diagram in importance.

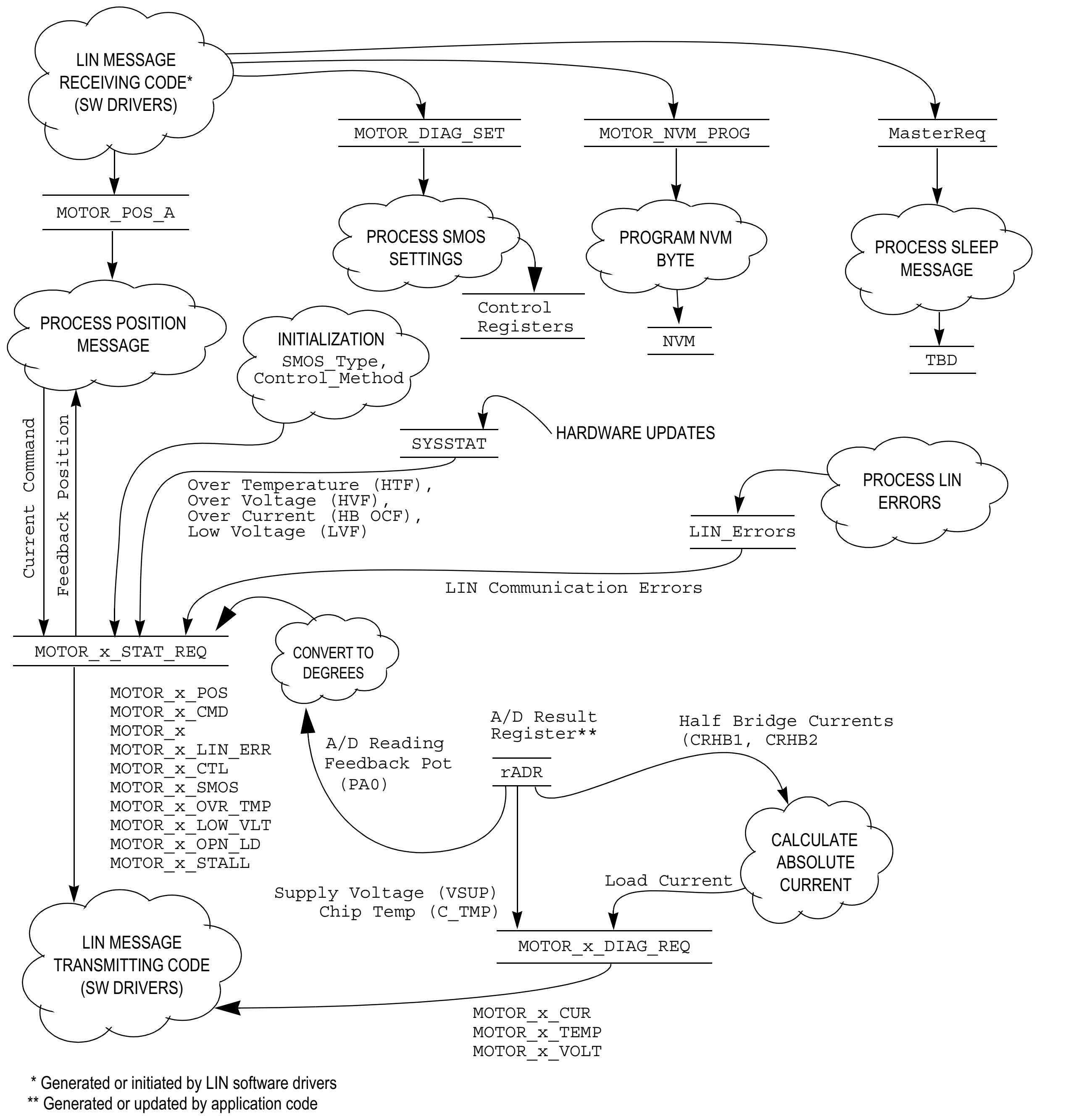

Figure 3. Closed-Loop Control Actuator Data Flow Diagram

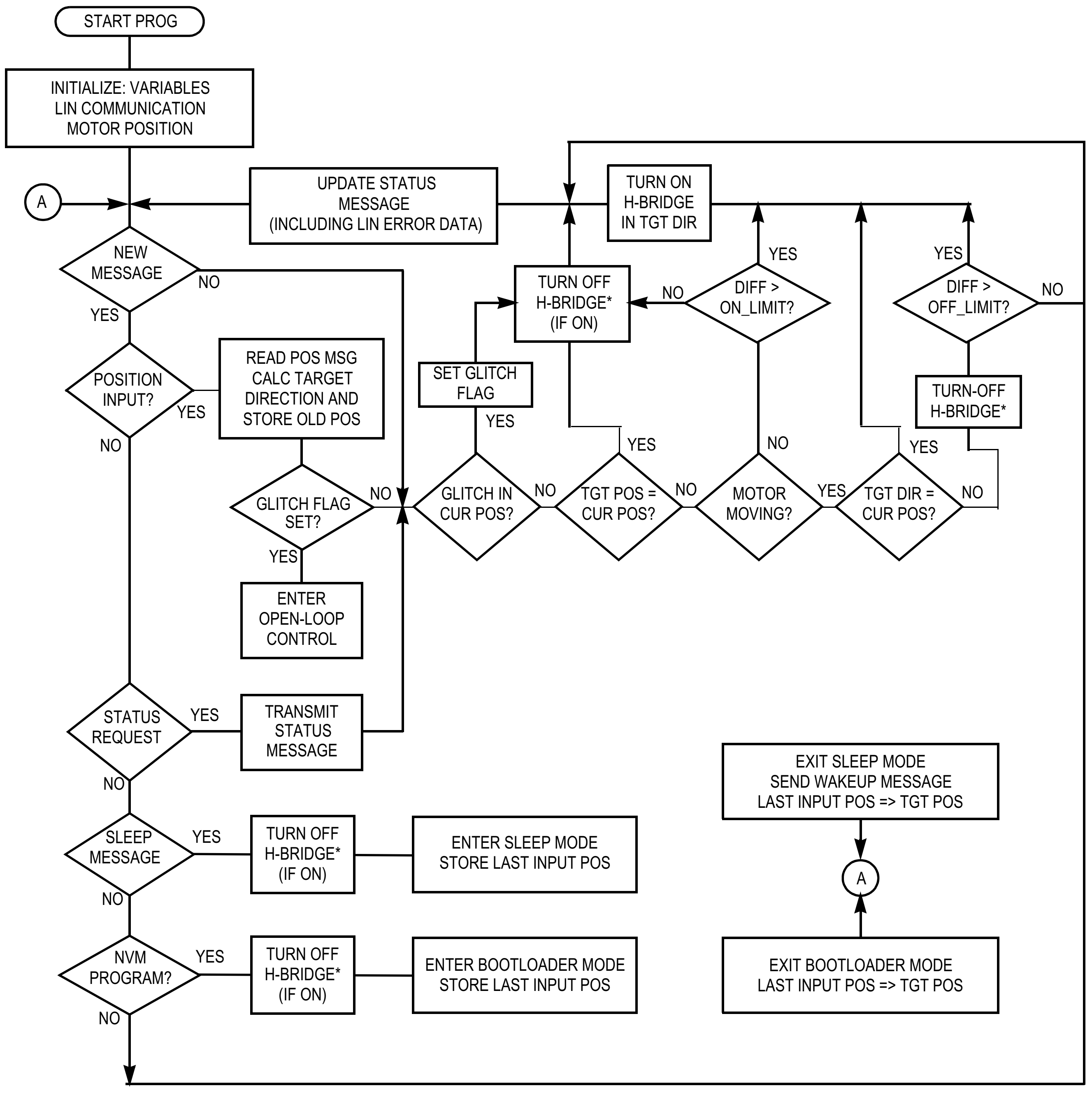

Figure 4. Closed-Loop Control DC Brush Position Servo Motor LIN Slave Node — Software Control Flow Diagram

Throughout the software, certain assumptions have been made:

- Master node will park motor prior to entering sleep mode
- Master node handles all fault conditions, slave node simply reports conditions and fails safe
- Sufficient delay is inserted after turning off H-bridge to allow current to dissipate and avoid back EMF issues
- H-bridge is self-protecting and cannot be activated when error conditions such as insufficient supply voltage exist.

A few details of variables in the control flow diagram need to be explained further. ON_LIMIT and OFF_LIMIT are hysteresis limits on position feedback which allow the user to set boundaries for what is considered “close enough” to the target position to not activate the motor. These serve to prevent oscillation and overshoot if set high enough and can be adjusted to suit the needs of the application. Faster responding systems might need a wider “OFF_LIMIT” to prevent overshoot problems from causing oscillations around the target position, for example.

Similarly, GLITCH_LIMIT is a threshold to limit the amount that the value of the feedback can change between samples. If the limit is exceeded, the algorithm throws a flag and after updating status variables enters open-loop control, where only full-open (0x00) or full-closed (0xFF) commands are accepted. All other position inputs are considered invalid and the motor does not move. In the case where the master node fails to recognize that a slave has entered open-loop mode, the slave will then only respond to full-open (0x00) or full-closed (0xFF) commands. The master node must decide how to handle these situations. The information is present for the master node to actually control the motor through the network, but the response is now limited to the speed of input updates obtained through the LIN network.

In order to simplify additional slave state machines, this module can be used for either open-loop or closed-loop systems. By setting GLITCH_LIMIT to 0x00, the user can force the module to always be in open-loop control mode. The limit could also be set to full scale to allow faster changes in feedback input for faster responding systems. GLITCH_LIMIT has not been implemented in the current version of the software.

1.12 What are the Unimplemented Features?

As mentioned earlier, some features have not been implemented at this time and are left as an exercise for the reader. Among these are:

1. Implement sleep, NVM programming, and terminal fault shutdown states
2. LIN communications error checking
3. Open-load indication flag
4. GLITCH_LIMIT checking of feedback to check for faults in feedback data

2.0 Improvements, Lessons Learned, and Suggestions

As with all designs, there is always room for improvement. Some improvements are for better performance, some might be to simplify the design or to make it more generic. Some of the ways in which this controller could be improved might be:

- Characterize a wide selection of feedback potentiometers to ensure that the translation from ADC reading to real position is accurate. This could even be implemented in a lookup table or piecewise defined equation, depending on available ROM space.
- Initialization sequence to test full range of motion — this feature would allow the dynamic characterization of the feedback potentiometer and set boundary conditions appropriately. This allows the software to be more generic and reusable with multiple different actuators without reprogramming.
- Lock down current limitation for motor used and dial-in parameters to avoid oscillations — by characterizing the motors used, it is possible to fine tune the software parameters more closely to match production systems.
- Back EMF delay based on zero current — rather than using a delay of fixed time to allow the motor current to dissipate while in MOTOR_STOP state, it is possible with Freescale SMOS to measure the current through the low-side switches. If the motor braking is achieved by shorting LS switches, then the delay need only be long enough to ensure that the current drops to zero. This would improve response time when reversing the motor.
- Acceleration/deceleration profiling to improve smoothness of motor operation.
- Memory positions — add programmable locations so that the motor could go to a series of memorized positions. This is especially useful for automotive applications, when multiple drivers might have different preferences of seat position, HVAC settings, etc. For example, in this way a single command could be issued to all motors to go to the position for driver 1. With the current implementation, the memory positions could easily be stored in the master and appropriate inputs sent to each motor. Each method has benefits and detriments.

3.0 Conclusion

This application note is intended to show one way to implement a closed-loop controlled servo motor application, which can be controlled over a LIN subnetwork. This application can be adapted to suit many different applications. The basic LIN messaging strategy and application code listed here can easily be adapted and modified to suit each application.

This information is also not limited to mechatronics components. The code listed in this application can be adapted to easily work on most M68HC08, M68HC12, or M68HCS12 microcontrollers, depending on the requirements of the application. Additionally, the SMOS H-bridge driver component is available as a separate component that can be driven with essentially the same code from this application note.

4.0 Appendix A — LIN HVAC System Demonstration Messaging Strategy

For ease of use, the LIN HVAC System Demonstration Messaging Strategy documentation is contained in a downloadable spreadsheet format (LIN_HVAC_messaging_strategy_1_00.xls) and can be downloaded from the associated archive file *AN2396.zip* at:

http://freescale.com/semiconductors/

5.0 Appendix B — LIN HVAC System Demonstration Messaging Strategy — Configuration Language Description File

For LIN standardization and integration with standard LIN tools, the LIN HVAC System Demonstration Messaging Strategy has also been captured in the following LIN Configuration Language Description File (LIN_HVAC_messaging_strategy_1_00.ldf), which can be downloaded from the associated archive file *AN2396.zip* at:

http://freescale.com/semiconductors/

And, is also shown on the following pages.

6.0 Appendix C — Closed-Loop Control HVAC Actuator Source Code

Source code for this application note can be downloaded from the associated archive file *AN2396.zip* at:

http://freescale.com/semiconductors/

```
/*******************************************************************************
                                                      Copyright (c) Freescale 2003

File Name        :      LIN_HVAC_messaging_strategy_1_00.ldf

Engineer         :      Matt Ruff

Location         :      OHT

Date Created     :      17 September 2001

Current Revision :      1.00 - 2 Mar 2003

                        0.87 - 6 Feb 2002

                        0.86 - 30 Jan 2002

Notes            :      LIN HVAC system demo - LIN Description File

*******************************************************************************
Freescale reserves the right to make changes without further notice to any
product herein to improve reliability, function or design. Freescale does not
assume any liability arising out of the application or use of any product,
circuit, or software described herein; neither does it convey any license
under its patent rights nor the rights of others. Freescale products are not
designed, intended, or authorized for use as components in systems intended for
surgical implant into the body, or other applications intended to support life,
or for any other application in which the failure of the Freescale product
could create a situation where personal injury or death may occur. Should
Buyer purchase or use Freescale products for any such unintended or
unauthorized application, Buyer shall idemnify and hold Freescale and its
officers, employees, subsidiaries, affiliates, and distributors harmless
against all claims costs, damages, and expenses, and reasonable attorney fees
arising out of, directly or indirectly, any claim of personal injury or death
associated with such unintended or unauthorized use, even if such claim alleges
that Freescale was negligent regarding the design or manufacture of the part.
Freescale and the Freescale logo* are registered trademarks of Freescale
Ltd.*******************************************************************************/

LIN_description_file;
LIN_protocol_version = 1.3;
LIN_language_version = 1.3;
LIN_speed = 9.615 kbps;

Nodes
{
Master: HVAC_CTL, 1 ms, 0.1 ms;
Slaves: MOTOR_0,MOTOR_1,MOTOR_2,MOTOR_3,MOTOR_4,MOTOR_5,MOTOR_6,MOTOR_7,BLOWER,TEMP_SEN;
}

/*____________________________________________________________________Signal Definitions___*/

Signals {
    /* Name            Size   Init  Sender Receiver(s)                        */
    /* ----            ----   ----  ------ -----------                        */

        MOTOR_0_INPUT:   8,                    0,HVAC_CTL, MOTOR_0;
        MOTOR_1_INPUT:   8,                    0,HVAC_CTL, MOTOR_1;
        MOTOR_2_INPUT:   8,                    0,HVAC_CTL, MOTOR_2;
        MOTOR_3_INPUT:   8,                    0,HVAC_CTL, MOTOR_3;
        MOTOR_4_INPUT:   8,                        0,HVAC_CTL, MOTOR_4;
        MOTOR_5_INPUT:   8,                        0,HVAC_CTL, MOTOR_5;
        MOTOR_6_INPUT:   8,                        0,HVAC_CTL, MOTOR_6;
        MOTOR_7_INPUT:   8,                        0,HVAC_CTL, MOTOR_7;

        /* ------------- MOTOR_0 signals ----------------*/

        MOTOR_0_POS:     8,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_CMD:     2,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_LIN_ERR: 3,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_CTL:     1,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_SMOS:    1,      0,                MOTOR_0, HVAC_CTL;
/*      <RESERVED>       1,      0,                MOTOR_0, HVAC_CTL*/
        MOTOR_0_OVR_TMP: 1,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_OVR_VLT: 1,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_LOW_VLT: 1,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_OVR_CUR: 1,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_OPN_LD:  1,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_STALL:   1,      0,                MOTOR_0, HVAC_CTL;
/*      <RESERVED>       1,      0,                MOTOR_0, HVAC_CTL*/
/*      <RESERVED>       1,      0,                MOTOR_0, HVAC_CTL*/

        MOTOR_0_CUR:     8,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_TEMP:    8,      0,                MOTOR_0, HVAC_CTL;
        MOTOR_0_VOLT:    8,      0,                MOTOR_0, HVAC_CTL;

        /* ------------- MOTOR_1 signals ----------------*/

        MOTOR_1_POS:     8,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_CMD:     2,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_LIN_ERR: 3,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_CTL:     1,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_SMOS:    1,      0,                MOTOR_1, HVAC_CTL;
/*      <RESERVED>       1,      0,                MOTOR_1, HVAC_CTL*/
        MOTOR_1_OVR_TMP: 1,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_OVR_VLT: 1,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_LOW_VLT: 1,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_OVR_CUR: 1,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_OPN_LD:  1,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_STALL:   1,      0,                MOTOR_1, HVAC_CTL;
/*      <RESERVED>       1,      0,                MOTOR_1, HVAC_CTL*/
/*      <RESERVED>       1,      0,                MOTOR_1, HVAC_CTL*/

        MOTOR_1_CUR:     8,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_TEMP:    8,      0,                MOTOR_1, HVAC_CTL;
        MOTOR_1_VOLT:    8,      0,                MOTOR_1, HVAC_CTL;

/*      SYSSTAT:         8,      0,                MOTOR_1, HVAC_CTL;// X05 TEST GUI Signals
        HASTAT:          8,      0,                MOTOR_1, HVAC_CTL;
        HB1_Current:     8,      0,                MOTOR_1, HVAC_CTL;
        HB2_Current:     8,      0,                MOTOR_1, HVAC_CTL;
        HB3_Current:     8,      0,                MOTOR_1, HVAC_CTL;
        HB4_Current:     8,      0,                MOTOR_1, HVAC_CTL;
        VSUP:            8,      0,                MOTOR_1, HVAC_CTL;
        CHIP:            8,      0,                MOTOR_1, HVAC_CTL;// --------------------
```

```
*/
        /* ------------- MOTOR_2 signals ----------------*/

        MOTOR_2_POS:     8,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_CMD:     2,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_LIN_ERR: 3,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_CTL:     1,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_SMOS:    1,      0,                     MOTOR_2, HVAC_CTL;
/*      <RESERVED>       1,      0,                     MOTOR_2, HVAC_CTL*/
        MOTOR_2_OVR_TMP: 1,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_OVR_VLT: 1,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_LOW_VLT: 1,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_OVR_CUR: 1,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_OPN_LD:  1,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_STALL:   1,      0,                     MOTOR_2, HVAC_CTL;
/*      <RESERVED>       1,      0,                     MOTOR_2, HVAC_CTL*/
/*      <RESERVED>       1,      0,                     MOTOR_2, HVAC_CTL*/

        MOTOR_2_CUR:     8,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_TEMP:    8,      0,                     MOTOR_2, HVAC_CTL;
        MOTOR_2_VOLT:    8,      0,                     MOTOR_2, HVAC_CTL;

        /* ------------- MOTOR_3 signals ----------------*/

        MOTOR_3_POS:     8,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_CMD:     2,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_LIN_ERR: 3,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_CTL:     1,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_SMOS:    1,      0,                     MOTOR_3, HVAC_CTL;
/*      <RESERVED>       1,      0,                     MOTOR_3, HVAC_CTL*/
        MOTOR_3_OVR_TMP: 1,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_OVR_VLT: 1,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_LOW_VLT: 1,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_OVR_CUR: 1,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_OPN_LD:  1,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_STALL:   1,      0,                     MOTOR_3, HVAC_CTL;
/*      <RESERVED>       1,      0,                     MOTOR_3, HVAC_CTL*/
/*      <RESERVED>       1,      0,                     MOTOR_3, HVAC_CTL*/

        MOTOR_3_CUR:     8,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_TEMP:    8,      0,                     MOTOR_3, HVAC_CTL;
        MOTOR_3_VOLT: 8, 0, MOTOR_3, HVAC_CTL;

        /* ------------- MOTOR_4 signals ----------------*/

        MOTOR_4_POS:     8,      0,                     MOTOR_4, HVAC_CTL;
        MOTOR_4_CMD:     2,      0,                     MOTOR_4, HVAC_CTL;
        MOTOR_4_LIN_ERR: 3,      0,                     MOTOR_4, HVAC_CTL;
        MOTOR_4_CTL:     1,      0,                     MOTOR_4, HVAC_CTL;
        MOTOR_4_SMOS:    1,      0,                     MOTOR_4, HVAC_CTL;
/*      <RESERVED>       1,      0,                     MOTOR_4, HVAC_CTL*/
        MOTOR_4_OVR_TMP: 1,      0,                     MOTOR_4, HVAC_CTL;
        MOTOR_4_OVR_VLT: 1,      0,                     MOTOR_4, HVAC_CTL;
        MOTOR_4_LOW_VLT: 1,      0,                     MOTOR_4, HVAC_CTL;
        MOTOR_4_OVR_CUR: 1,      0,                     MOTOR_4, HVAC_CTL;
        MOTOR_4_OPN_LD:  1,      0,                     MOTOR_4, HVAC_CTL;
        MOTOR_4_STALL:   1,      0,                     MOTOR_4, HVAC_CTL;
```

```
/*      <RESERVED>        1,     0,                  MOTOR_4, HVAC_CTL*/
/*      <RESERVED>        1,     0,                  MOTOR_4, HVAC_CTL*/

        MOTOR_4_CUR:      8,     0,                  MOTOR_4, HVAC_CTL;
        MOTOR_4_TEMP:     8,     0,                  MOTOR_4, HVAC_CTL;
        MOTOR_4_VOLT: 8, 0, MOTOR_4, HVAC_CTL;

        /* ------------- MOTOR_5 signals ----------------*/

        MOTOR_5_POS:      8,     0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_CMD:      2,     0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_LIN_ERR: 3,      0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_CTL:      1,     0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_SMOS:     1,     0,                  MOTOR_5, HVAC_CTL;
/*      <RESERVED>        1,     0,                  MOTOR_5, HVAC_CTL*/
        MOTOR_5_OVR_TMP: 1,      0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_OVR_VLT: 1,      0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_LOW_VLT: 1,      0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_OVR_CUR: 1,      0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_OPN_LD:   1,     0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_STALL:    1,     0,                  MOTOR_5, HVAC_CTL;
/*      <RESERVED>        1,     0,                  MOTOR_5, HVAC_CTL*/
/*      <RESERVED>        1,     0,                  MOTOR_5, HVAC_CTL*/

        MOTOR_5_CUR:      8,     0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_TEMP:     8,     0,                  MOTOR_5, HVAC_CTL;
        MOTOR_5_VOLT:     8,     0,                  MOTOR_5, HVAC_CTL;

        /* ------------- MOTOR_6 signals ----------------*/

        MOTOR_6_POS:      8,     0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_CMD:      2,     0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_LIN_ERR: 3,      0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_CTL:      1,     0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_SMOS:     1,     0,                  MOTOR_6, HVAC_CTL;
/*      <RESERVED>        1,     0,                  MOTOR_6, HVAC_CTL*/
        MOTOR_6_OVR_TMP: 1,      0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_OVR_VLT: 1,      0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_LOW_VLT: 1,      0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_OVR_CUR: 1,      0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_OPN_LD:   1,     0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_STALL:    1,     0,                  MOTOR_6, HVAC_CTL;
/*      <RESERVED>        1,     0,                  MOTOR_6, HVAC_CTL*/
/*      <RESERVED>        1,     0,                  MOTOR_6, HVAC_CTL*/

        MOTOR_6_CUR:      8,     0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_TEMP:     8,     0,                  MOTOR_6, HVAC_CTL;
        MOTOR_6_VOLT:     8,     0,                  MOTOR_6, HVAC_CTL;

        /* ------------- MOTOR_7 signals ----------------*/

        MOTOR_7_POS:      8,     0,                  MOTOR_7, HVAC_CTL;
        MOTOR_7_CMD:      2,     0,                  MOTOR_7, HVAC_CTL;
        MOTOR_7_LIN_ERR: 3,      0,                  MOTOR_7, HVAC_CTL;
        MOTOR_7_CTL:      1,     0,                  MOTOR_7, HVAC_CTL;
        MOTOR_7_SMOS:     1,     0,                  MOTOR_7, HVAC_CTL;
/*      <RESERVED>        1,     0,                  MOTOR_7, HVAC_CTL*/
        MOTOR_7_OVR_TMP: 1,      0,                  MOTOR_7, HVAC_CTL;
        MOTOR_7_OVR_VLT: 1,      0,                 MOTOR_7, HVAC_CTL;
        MOTOR_7_LOW_VLT: 1,      0,                 MOTOR_7, HVAC_CTL;
        MOTOR_7_OVR_CUR: 1,      0,                 MOTOR_7, HVAC_CTL;
        MOTOR_7_OPN_LD:  1,      0,                 MOTOR_7, HVAC_CTL;
        MOTOR_7_STALL:   1,      0,                 MOTOR_7, HVAC_CTL;
/*      <RESERVED>       1,      0,                 MOTOR_7, HVAC_CTL*/
/*      <RESERVED>       1,      0,                 MOTOR_7, HVAC_CTL*/

        MOTOR_7_CUR:     8,      0,                 MOTOR_7, HVAC_CTL;
        MOTOR_7_TEMP:    8,      0,                 MOTOR_7, HVAC_CTL;
        MOTOR_7_VOLT:    8,      0,                 MOTOR_7, HVAC_CTL;

        MOTOR_0_SET_A    8,      0,                 HVAC_CTL, MOTOR_0;
        MOTOR_0_SET_B    8,      0,                 HVAC_CTL, MOTOR_0;

        MOTOR_1_SET_A    8,      0,                 HVAC_CTL, MOTOR_1;
        MOTOR_1_SET_B    8,      0,                 HVAC_CTL, MOTOR_1;

        MOTOR_2_SET_A    8,      0,                 HVAC_CTL, MOTOR_2;
        MOTOR_2_SET_B    8,      0,                 HVAC_CTL, MOTOR_2;
        MOTOR_3_SET_A    8,      0,                 HVAC_CTL, MOTOR_3;
        MOTOR_3_SET_B    8,      0,                 HVAC_CTL, MOTOR_3;

/*-- SPECIAL CASE: <subscribed_by> of next 8 signals depends on who's powered on bus at time
---*/

        MOTOR_NODE_ADDR  8,      0,                 HVAC_CTL,
MOTOR_0,MOTOR_1,MOTOR_2,MOTOR_3,MOTOR_4,MOTOR_5,MOTOR_6,MOTOR_7,BLOWER,TEMP_SEN;
        MOTOR_CMD_ID     8,      0,                 HVAC_CTL,
MOTOR_0,MOTOR_1,MOTOR_2,MOTOR_3,MOTOR_4,MOTOR_5,MOTOR_6,MOTOR_7,BLOWER,TEMP_SEN;
        MOTOR_STAT_ID    8,      0,                 HVAC_CTL,
MOTOR_0,MOTOR_1,MOTOR_2,MOTOR_3,MOTOR_4,MOTOR_5,MOTOR_6,MOTOR_7,BLOWER,TEMP_SEN;
        MOTOR_VAR_1      8,      0,                 HVAC_CTL,
MOTOR_0,MOTOR_1,MOTOR_2,MOTOR_3,MOTOR_4,MOTOR_5,MOTOR_6,MOTOR_7,BLOWER,TEMP_SEN;
        MOTOR_VAR_2      8,      0,                 HVAC_CTL,
MOTOR_0,MOTOR_1,MOTOR_2,MOTOR_3,MOTOR_4,MOTOR_5,MOTOR_6,MOTOR_7,BLOWER,TEMP_SEN;
        MOTOR_VAR_3      8,      0,                 HVAC_CTL,
MOTOR_0,MOTOR_1,MOTOR_2,MOTOR_3,MOTOR_4,MOTOR_5,MOTOR_6,MOTOR_7,BLOWER,TEMP_SEN;
        MOTOR_VAR_4      8,      0,                 HVAC_CTL,
MOTOR_0,MOTOR_1,MOTOR_2,MOTOR_3,MOTOR_4,MOTOR_5,MOTOR_6,MOTOR_7,BLOWER,TEMP_SEN;
        MOTOR_VAR_5      8,      0,                 HVAC_CTL,
MOTOR_0,MOTOR_1,MOTOR_2,MOTOR_3,MOTOR_4,MOTOR_5,MOTOR_6,MOTOR_7,BLOWER,TEMP_SEN;

/*-- end SPECIAL CASE ---*/

        MOTOR_0_STALL_LMT8,      0,                 HVAC_CTL, MOTOR_0;
        MOTOR_1_STALL_LMT8,      0,                 HVAC_CTL, MOTOR_1;
        MOTOR_2_STALL_LMT8,      0,                 HVAC_CTL, MOTOR_2;
        MOTOR_3_STALL_LMT8,      0,                 HVAC_CTL, MOTOR_3;
        MOTOR_4_STALL_LMT8,      0,                 HVAC_CTL, MOTOR_4;
        MOTOR_5_STALL_LMT8,      0,                 HVAC_CTL, MOTOR_5;
        MOTOR_6_STALL_LMT8,      0,                 HVAC_CTL, MOTOR_6;
        MOTOR_7_STALL_LMT8,      0,                 HVAC_CTL, MOTOR_7;

//      SYS_SLEEP        8,      0,                 HVAC_CTL,
MOTOR_0,MOTOR_1,MOTOR_2,MOTOR_3,MOTOR_4,MOTOR_5,MOTOR_6,MOTOR_7,BLOWER,TEMP_SEN;
}
```

```
/*______________________________________________________________________Frame Definitions____*/
Frames {
    /* FrameName    Id  Sender                                                          */
    /* ---------    --  ------                                                          */
MOTOR_POS_A:      58, HVAC_CTL, 8 {
        /* Signal     Offset                                                            */
        /* ------     ------                                                            */
        MOTOR_0_INPUT,    0;
        MOTOR_1_INPUT,    8;
        MOTOR_2_INPUT,   16;
        MOTOR_3_INPUT,24;
        MOTOR_4_INPUT,32;
        MOTOR_5_INPUT,40;
        MOTOR_6_INPUT,48;
        MOTOR_7_INPUT,56;
        }
MOTOR_0_STAT_REQ:0x10, MOTOR_0, 3 {
        /* Signal     Offset                                                            */
        /* ------     ------                                                            */
        MOTOR_0_POS,      0;
        MOTOR_0_CMD,      8;
        MOTOR_0_LIN_ERR,10;
        MOTOR_0_CTL,     13;
        MOTOR_0_SMOS,    14;
        /*<reserved><reserved>1 15 */
        MOTOR_0_OVR_TMP,16;
        MOTOR_0_OVR_VLT,17;
        MOTOR_0_LOW_VLT,18;
        MOTOR_0_OVR_CUR,19;
        MOTOR_0_OPN_LD,  20;
        MOTOR_0_STALL,   21;
        /* <reserved>SMOS diag1 22 */
        /* <reserved>SMOS diag1 23 */
        }
MOTOR_0_DIAG_REQ:0x20, MOTOR_0, 3 {
        /* Signal     Offset                                                            */
        /* ------     ------                                                            */
        MOTOR_0_CUR, 0;
        MOTOR_0_TEMP, 8;
        MOTOR_0_VOLT,16;
        }
MOTOR_1_STAT_REQ:0x11, MOTOR_1, 3 {
        /* Signal     Offset                                                            */
        /* ------     ------                                                            */
        MOTOR_1_POS,      0;
        MOTOR_1_CMD,      8;
        MOTOR_1_LIN_ERR,10;
        MOTOR_1_CTL,     13;
        MOTOR_1_SMOS,    14;
        /*<reserved><reserved>1 15 */
        MOTOR_1_OVR_TMP,16;
        MOTOR_1_OVR_VLT,17;
        MOTOR_1_LOW_VLT,18;
        MOTOR_1_OVR_CUR,19;
        MOTOR_1_OPN_LD,  20;
        MOTOR_1_STALL,   21;
        /* <reserved>SMOS diag1 22 */
        /* <reserved>SMOS diag1 23 */
        }
MOTOR_1_DIAG_REQ:0x21, MOTOR_1, 3 {
        /* Signal     Offset                                              */
        /* ------     ------                                              */
        MOTOR_1_CUR, 0;
        MOTOR_1_TEMP, 8;
        MOTOR_1_VOLT,16;
        }
MOTOR_2_STAT_REQ:0x12, MOTOR_2, 3 {
        /* Signal     Offset                                              */
        /* ------     ------                                              */
        MOTOR_2_POS,      0;
        MOTOR_2_CMD,      8;
        MOTOR_2_LIN_ERR,10;
        MOTOR_2_CTL,     13;
        MOTOR_2_SMOS,    14;
        /*<reserved><reserved>1 15 */
        MOTOR_2_OVR_TMP,16;
        MOTOR_2_OVR_VLT,17;
        MOTOR_2_LOW_VLT,18;
        MOTOR_2_OVR_CUR,19;
        MOTOR_2_OPN_LD,  20;
        MOTOR_2_STALL,   21;
        /* <reserved>SMOS diag1 22 */
        /* <reserved>SMOS diag1 23 */
        }
MOTOR_2_DIAG_REQ:0x22, MOTOR_2, 3 {
        /* Signal     Offset                                              */
        /* ------     ------                                              */
MOTOR_2_CUR, 0;

MOTOR_2_TEMP, 8;

MOTOR_2_VOLT,16;

}
MOTOR_3_STAT_REQ:0x13, MOTOR_3, 3 {
        /* Signal     Offset                                              */
        /* ------     ------                                              */
        MOTOR_3_POS,      0;
        MOTOR_3_CMD,      8;
        MOTOR_3_LIN_ERR,10;
        MOTOR_3_CTL,     13;
        MOTOR_3_SMOS,    14;
        /*<reserved><reserved>1 15 */
        MOTOR_3_OVR_TMP,16;
        MOTOR_3_OVR_VLT,17;
        MOTOR_3_LOW_VLT,18;
        MOTOR_3_OVR_CUR,19;
        MOTOR_3_OPN_LD,  20;
        MOTOR_3_STALL,   21;
/* <reserved>SMOS diag1 22 */
/* <reserved>SMOS diag1 23 */
}
MOTOR_3_DIAG_REQ:0x23, MOTOR_3, 3 {
        /* Signal     Offset                                                   */
        /* ------     ------                                                   */
        MOTOR_3_CUR, 0;
        MOTOR_3_TEMP, 8;
        MOTOR_3_VOLT,16;
        }
MOTOR_4_STAT_REQ:0x14, MOTOR_4, 3 {
        /* Signal     Offset                                                   */
        /* ------     ------                                                   */
        MOTOR_4_POS,      0;
        MOTOR_4_CMD,      8;
        MOTOR_4_LIN_ERR,10;
        MOTOR_4_CTL,     13;
        MOTOR_4_SMOS,    14;
        /*<reserved><reserved>1 15 */
        MOTOR_4_OVR_TMP,16;
        MOTOR_4_OVR_VLT,17;
        MOTOR_4_LOW_VLT,18;
        MOTOR_4_OVR_CUR,19;
        MOTOR_4_OPN_LD,  20;
        MOTOR_4_STALL,   21;
        /* <reserved>SMOS diag1 22 */
        /* <reserved>SMOS diag1 23 */
        }
MOTOR_4_DIAG_REQ:0x24, MOTOR_4, 3 {
        /* Signal     Offset                                                   */
        /* ------     ------                                                   */
        MOTOR_4_CUR, 0;
        MOTOR_4_TEMP, 8;
        MOTOR_4_VOLT,16;
        }
MOTOR_5_STAT_REQ:0x35, MOTOR_5, 3 {
        /* Signal     Offset                                                   */
        /* ------     ------                                                   */
        MOTOR_5_POS,  0;
        MOTOR_5_CMD,  8;
        MOTOR_5_LIN_ERR,10;
        MOTOR_5_CTL,  13;
        MOTOR_5_SMOS, 14;
        /*<reserved><reserved>1 15 */
        MOTOR_5_OVR_TMP,16;
        MOTOR_5_OVR_VLT,17;
        MOTOR_5_LOW_VLT,18;
        MOTOR_5_OVR_CUR,19;
        MOTOR_5_OPN_LD,20;
        MOTOR_5_STALL,21;
        /* <reserved>SMOS diag1 22 */
        /* <reserved>SMOS diag1 23 */
        }
MOTOR_5_DIAG_REQ:0x25, MOTOR_5, 3 {
        /* Signal     Offset                                              */
        /* ------     ------                                              */
        MOTOR_5_CUR, 0;
        MOTOR_5_TEMP, 8;
        MOTOR_5_VOLT,16;
        }

MOTOR_6_STAT_REQ:0x36, MOTOR_6, 3 {
        /* Signal     Offset                                              */
        /* ------     ------                                              */
        MOTOR_6_POS,      0;
        MOTOR_6_CMD,      8;
        MOTOR_6_LIN_ERR,10;
        MOTOR_6_CTL,     13;
        MOTOR_6_SMOS,    14;
        /*<reserved><reserved>1 15 */
        MOTOR_6_OVR_TMP,16;
        MOTOR_6_OVR_VLT,17;
        MOTOR_6_LOW_VLT,18;
        MOTOR_6_OVR_CUR,19;
        MOTOR_6_OPN_LD,  20;
        MOTOR_6_STALL,   21;
        /* <reserved>SMOS diag1 22 */
        /* <reserved>SMOS diag1 23 */
        }

MOTOR_6_DIAG_REQ:0x26, MOTOR_6, 3 {
        /* Signal     Offset                                              */
        /* ------     ------                                              */
        MOTOR_6_CUR, 0;
        MOTOR_6_TEMP, 8;
        MOTOR_6_VOLT,16;
        }

MOTOR_7_STAT_REQ:0x37, MOTOR_7, 3 {
        /* Signal     Offset                                              */
        /* ------     ------                                              */
        MOTOR_7_POS,      0;
        MOTOR_7_CMD,      8;
        MOTOR_7_LIN_ERR,10;
        MOTOR_7_CTL,     13;
        MOTOR_7_SMOS,    14;
        /*<reserved><reserved>1 15 */
        MOTOR_7_OVR_TMP,16;
        MOTOR_7_OVR_VLT,17;
        MOTOR_7_LOW_VLT,18;
        MOTOR_7_OVR_CUR,19;
        MOTOR_7_OPN_LD,  20;
        MOTOR_7_STALL,   21;
        /* <reserved>SMOS diag1 22 */
        /* <reserved>SMOS diag1 23 */
        }
MOTOR_7_DIAG_REQ:0x27, MOTOR_7, 3 {
        /* Signal     Offset                                                 */
        /* ------     ------                                                 */
        MOTOR_7_CUR, 0;
        MOTOR_7_TEMP, 8;
        MOTOR_7_VOLT,16;
        }
MOTOR_DIAG_SET: 0x1A, HVAC_CTL, 8 {
        /* Signal     Offset                                                 */
        /* ------     ------                                                 */
        MOTOR_0_SET_A, 0;
        MOTOR_0_SET_B, 8;
        MOTOR_1_SET_A,16;
        MOTOR_1_SET_B,24;
        MOTOR_2_SET_A,32;
        MOTOR_2_SET_B,40;
        MOTOR_3_SET_A,48;
        MOTOR_3_SET_B,56;
        }
MOTOR_NVM_PROG: 0x3B, HVAC_CTL, 8 {
        /* Signal     Offset                                                 */
        /* ------     ------                                                 */
        MOTOR_NODE_ADDR, 0;
        MOTOR_CMD_ID, 8;
        MOTOR_STAT_ID,16;
        MOTOR_VAR_1,24;
        MOTOR_VAR_2,32;
        MOTOR_VAR_3,40;
        MOTOR_VAR_4,48;
        MOTOR_VAR_5,56;
        }
MOTOR_STALL_A: 0x09, HVAC_CTL, 8 {
        /* Signal     Offset                                                 */
        /* ------     ------                                                 */
        MOTOR_0_STALL_LMT 0;
        MOTOR_1_STALL_LMT 8;
        MOTOR_2_STALL_LMT16;
        MOTOR_3_STALL_LMT24;
        MOTOR_4_STALL_LMT32;
        MOTOR_5_STALL_LMT40;
        MOTOR_6_STALL_LMT48;
        MOTOR_7_STALL_LMT56;
        }
```

```
//      MasterReq:    0x3C, HVAC_CTL {
//        /* Signal    Offset                                                      */
//        /* ------    ------                                                      */
//               SYS_SLEEP,      0;
//      }

/*      SlaveResp:          0x3D */
/*      <reserved>          0x3E */

/*      <LIN reserved>    0x3F */
}

/*__________________________________________________________Schedule Table Definitions____*/
Schedule_tables {

        DEMO_SCHED {
                MOTOR_POS_A     delay 20.00      ms;
                MOTOR_0_STAT_REQdelay13.00       ms;
                MOTOR_0_DIAG_REQdelay13.00       ms;
                MOTOR_1_STAT_REQdelay13.00       ms;
                MOTOR_1_DIAG_REQdelay13.00       ms;
                MOTOR_2_STAT_REQdelay13.00       ms;
                MOTOR_2_DIAG_REQdelay13.00       ms;
                MOTOR_3_STAT_REQdelay13.00       ms;
                MOTOR_3_DIAG_REQdelay13.00       ms;
                MOTOR_POS_A     delay 20.00      ms;
                MOTOR_4_STAT_REQdelay13.00       ms;
                MOTOR_4_DIAG_REQdelay13.00       ms;
                MOTOR_5_STAT_REQdelay13.00       ms;
                MOTOR_5_DIAG_REQdelay13.00       ms;
                MOTOR_6_STAT_REQdelay13.00       ms;
                MOTOR_6_DIAG_REQdelay13.00       ms;
                MOTOR_7_STAT_REQdelay13.00       ms;
                MOTOR_7_DIAG_REQdelay13.00       ms;
                MOTOR_STALL_A   delay            20.00ms;
                //MOTOR_DIAG_SET_Adelay20.00     ms;
        }

        FAST_SCHED {

                MOTOR_POS_A     delay 20.00      ms;
                MOTOR_0_STAT_REQdelay13.00       ms;
                MOTOR_1_STAT_REQdelay13.00       ms;
                MOTOR_POS_A     delay 20.00      ms;
                MOTOR_2_STAT_REQdelay13.00       ms;
                MOTOR_3_STAT_REQdelay13.00       ms;
                MOTOR_POS_A     delay 20.00      ms;
                MOTOR_4_STAT_REQdelay13.00       ms;
                MOTOR_5_STAT_REQdelay13.00       ms;
                MOTOR_POS_A     delay 20.00      ms;
                MOTOR_6_STAT_REQdelay13.00       ms;
                MOTOR_7_STAT_REQdelay13.00       ms;
        }

        BASIC_SCHED{
                MOTOR_POS_A     delay 20.00      ms;
                MOTOR_0_STAT_REQdelay13.00       ms;
                MOTOR_1_STAT_REQdelay13.00      ms;
                MOTOR_2_STAT_REQdelay13.00      ms;
                MOTOR_3_STAT_REQdelay13.00      ms;
                MOTOR_4_STAT_REQdelay13.00      ms;
                MOTOR_5_STAT_REQdelay13.00      ms;
                MOTOR_6_STAT_REQdelay13.00      ms;
                MOTOR_7_STAT_REQdelay13.00      ms;
        }
        MOTOR_1_SCHED{
                MOTOR_POS_A     delay 20.00     ms;
                MOTOR_1_STAT_REQdelay13.00      ms;
                MOTOR_1_DIAG_REQdelay13.00      ms;
        }
}
/*__________________________________________________________Signal Encoding Types_________*/
Signal_encoding_types {
   Boolean     {
                logical_value, 0, "False";
                logical_value, 1, " True";
        }
        Position {
                physical_value, 0, 255, 1.00, 0, " degree";
        }
        Command   {
                logical_value, 0, "Not Moving";
                logical_value, 1, "Rotating CW";
                logical_value, 2, "Rotating CCW";
                logical_value, 3, "SMOS Error";
        }
        LIN_Error {
                logical_value, 0, "No Error";
                logical_value, 1, "Bit-Error";
                logical_value, 2, "Checksum-Err";
                logical_value, 3, "ID-Parity-Error";
                logical_value, 4, "Slave-Not-Resp";
                logical_value, 5, "Inconsist-Syn-Fld";
                logical_value, 6, "No-Bus-Activity";
                logical_value, 7, "UNDEFINED";
        }
        Control_Method {
                logical_value, 0, "OPEN LOOP";
                logical_value, 1, "CLOSED LOOP";
        }
        SMOS_Type {
                logical_value, 0, "MUX3";
                logical_value, 1, "X05";
        }
        Current {
                physical_value, 0, 255, 0.0019607, 0, " amps";
        }
        Temperature {
                physical_value, 0, 255, 1.435, -98.14, " deg. C";
        }
        Voltage {
                physical_value, 0, 255, 0.1064, 0, " volts";
        }
        System_Messages {
                logical_value, 0, "System Sleep";
                physical_value, 1, 127, 1, 0, " RESERVED";
                physical_value, 128, 255, 1, 0, " USER DEFINED";
        }
}
Signal_representation {
Boolean: MOTOR_0_OVR_TMP, MOTOR_0_OVR_VLT, MOTOR_0_LOW_VLT, MOTOR_0_OVR_CUR, MOTOR_0_OPN_LD,
MOTOR_0_STALL, MOTOR_1_OVR_TMP, MOTOR_1_OVR_VLT, MOTOR_1_LOW_VLT, MOTOR_1_OVR_CUR,
MOTOR_1_OPN_LD, MOTOR_1_STALL, MOTOR_2_OVR_TMP, MOTOR_2_OVR_VLT, MOTOR_2_LOW_VLT,
MOTOR_2_OVR_CUR, MOTOR_2_OPN_LD, MOTOR_2_STALL, MOTOR_3_OVR_TMP, MOTOR_3_OVR_VLT,
MOTOR_3_LOW_VLT, MOTOR_3_OVR_CUR, MOTOR_3_OPN_LD, MOTOR_3_STALL, MOTOR_4_OVR_TMP,
MOTOR_4_OVR_VLT, MOTOR_4_LOW_VLT, MOTOR_4_OVR_CUR, MOTOR_4_OPN_LD, MOTOR_4_STALL,
MOTOR_5_OVR_TMP, MOTOR_5_OVR_VLT, MOTOR_5_LOW_VLT, MOTOR_5_OVR_CUR, MOTOR_5_OPN_LD,
MOTOR_5_STALL, MOTOR_6_OVR_TMP, MOTOR_6_OVR_VLT, MOTOR_6_LOW_VLT, MOTOR_6_OVR_CUR,
MOTOR_6_OPN_LD, MOTOR_6_STALL, MOTOR_7_OVR_TMP, MOTOR_7_OVR_VLT, MOTOR_7_LOW_VLT,
MOTOR_7_OVR_CUR, MOTOR_7_OPN_LD, MOTOR_7_STALL;

Position:MOTOR_0_INPUT, MOTOR_1_INPUT, MOTOR_2_INPUT, MOTOR_3_INPUT, MOTOR_4_INPUT,
MOTOR_5_INPUT, MOTOR_6_INPUT, MOTOR_7_INPUT, MOTOR_0_POS, MOTOR_1_POS, MOTOR_2_POS,
MOTOR_3_POS, MOTOR_4_POS, MOTOR_5_POS, MOTOR_6_POS, MOTOR_7_POS;

Command:MOTOR_0_CMD, MOTOR_1_CMD, MOTOR_2_CMD, MOTOR_3_CMD, MOTOR_4_CMD, MOTOR_5_CMD,
MOTOR_6_CMD, MOTOR_7_CMD;

LIN_Error:MOTOR_0_LIN_ERR, MOTOR_1_LIN_ERR, MOTOR_2_LIN_ERR, MOTOR_3_LIN_ERR, MOTOR_4_LIN_ERR,
MOTOR_5_LIN_ERR, MOTOR_6_LIN_ERR, MOTOR_7_LIN_ERR;

Control_Method: MOTOR_0_CTL, MOTOR_1_CTL, MOTOR_2_CTL, MOTOR_3_CTL, MOTOR_4_CTL, MOTOR_5_CTL,
MOTOR_6_CTL, MOTOR_7_CTL;

SMOS_Type: MOTOR_0_SMOS, MOTOR_1_SMOS, MOTOR_2_SMOS, MOTOR_3_SMOS, MOTOR_4_SMOS, MOTOR_5_SMOS,
MOTOR_6_SMOS, MOTOR_7_SMOS;

Current: MOTOR_0_CUR, MOTOR_1_CUR, MOTOR_2_CUR, MOTOR_3_CUR, MOTOR_4_CUR, MOTOR_5_CUR,
MOTOR_6_CUR, MOTOR_7_CUR;

Temperature: MOTOR_0_TEMP, MOTOR_1_TEMP, MOTOR_2_TEMP, MOTOR_3_TEMP, MOTOR_4_TEMP,
MOTOR_5_TEMP, MOTOR_6_TEMP, MOTOR_7_TEMP;

Voltage: MOTOR_0_VOLT, MOTOR_1_VOLT, MOTOR_2_VOLT, MOTOR_3_VOLT, MOTOR_4_VOLT, MOTOR_5_VOLT,
MOTOR_6_VOLT, MOTOR_7_VOLT;

//      System_Messages: SYS_SLEEP;

}
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2396
Rev. 1.0
12/2005