

Autonomous Prototype of a Full Dimension Continuous Haulage System

Bruce J. Wells

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mechanical Engineering

Dr. Robert Sturges, Chair
Dr. Michael Deisenroth
Dr. Charles Reinholtz

July 21, 1999
Blacksburg, Virginia

Keywords: Mechatronics, Autonomous Vehicles, Mining
Copyright 1999, Bruce J. Wells

Autonomous Prototype of a Full Dimension Continuous Haulage System

Bruce J. Wells

(ABSTRACT)

Design and development of a 1/10 scale prototype of a Full Dimension Continuous Haulage System manufactured by the Long-Airdox Company. The prototype, which will allow development and testing of path-planning and control algorithms for autonomous navigation and operation in underground coal mines, has been completed. The prototype system, though not an identical copy, clones all full-scale model degrees of freedom and functions necessary for navigation. In addition to the physical structure, a microcontroller-based system was developed for providing the necessary low-level motor controls, data gathering and multiple processor communications. High level software running on a laptop PC with the windows operating system is used for analyzing all measurement data, execution of path-planning and control algorithms and issuing the command data.

Acknowledgements

I would like to thank my committee for support of this project, especially Dr. Robert Sturges. If not for his enthusiasm for the project and good salesmanship, I would not have had the opportunity to gain such diverse experience with electronics and microcontroller-based systems. The experience is greatly appreciated because the project is as close to any real world project as any encountered during my prior employment as a mechanical design engineer for an aerospace contractor.

Our counterparts at the Long-Airdox Company are thanked for sponsoring this project and supporting an excellent working atmosphere. Although development schedules have been grueling at times, they have always been understanding and pleasurable to work with.

I would like to thank the VA Tech Formula SAE Car Project for use of their machine tools and shop space, and for assisting with prototype fabrication during times when my Graduate Teaching Assistant responsibilities were quite time consuming. Specifically, help from Brandon Bagwell, Terje Aasland and Russell Turner has been greatly appreciated. Greatly appreciated was the use of sheetmetal equipment and loaned materials made possible by Johnny Cox and the members of the Mechanical Engineering Machine Shop. David Mayhew is thanked for his help in clarifying how the CRC 16 checksum is calculated by the algorithm provided by SICK Optic Electronic.

I would like to thank Dr. Larry Mitchell for his mentoring throughout my undergraduate and graduate experiences, and with whose support helped me return to graduate school.

And finally, I would like to thank my teammates on this project. Aishwarya “Aish Aish Baby” Varadhan, Amnart “Mike” Kanarat, Yodyot “Yos” Wongwanich and Todd Upchurch have provided a diverse and enjoyable development team. Best wishes are given as they work towards successful completion of this project and their continued education.

Table of Contents

1	Introduction	1
2	Prototype Continuous Haulage System	6
2.1	Prototype Introduction	6
2.2	Prototype Hardware	7
2.3	Prototype Electronics	15
2.4	Prototype Software	20
2.5	Prototype Interface Program	26
2.5.1	Exploration of Software for Prototype Interface	27
2.5.2	LabVIEW Demonstration	28
2.5.3	Prototype Interface Development	30
3	SICK Optic LMS 200	33
3.1	Sensor Background	33
3.2	LMS 200 Telegram Structure	35
3.3	Calculation of the CRC 16 Checksum	37
3.4	Development of LabVIEW Interface for LMS 200	38
4	Full-Scale Continuous Haulage System	40
4.1	Full-Scale Introduction	40
4.2	Full-Scale Electronics Development	40
4.3	Full-Scale Software Development	43
5	Results and Conclusions	46
5.1	Results	46
5.2	Conclusions	48

Appendix A: Prototype CHS Schematics	50
Appendix B: Prototype CHS Electronic System	64
Appendix C: Controller Software	68
Appendix D: LabVIEW Interface and Control Program	89
Appendix E: SICK Optic LMS 200 Interface	99
Appendix F: Full-Scale MBC Hardware and Software	113
References	122
Vita	123

List of Figures

Figure 1	Long-Airbox Full Dimension Continuous Haulage System	1
Figure 2	Depiction of CHS navigating a coal mine	2
Figure 3	CHS used for VA Tech team test driving	4
Figure 4	Rear view of assembled prototype MBC	9
Figure 5	View of prototype MBC deck and dolly travel mechanism	10
Figure 6	MBC/Pig joint during initial assembly	12
Figure 7	MBC/Pig joint after final assembly	13
Figure 8	Prototype CHS featuring two MBCs and a Pig	14
Figure 9	Connection Layout for SPI and SCI Devices	17
Figure 10	Control Hierarchy of a single Prototype MBC	18
Figure 11	MBC master and slave controllers mounted in prototype MBC	20
Figure 12	Flowchart of MBC master controller	22
Figure 13	Flowchart of MBC slave controller	23
Figure 14	Flowchart of MBC master controller with Laser-Video Scanners	24
Figure 15	Flowchart of MBC slave controller with velocity measurements	25
Figure 16	LabVIEW demonstration program	29
Figure 17	Flowchart of interface program using LMS device	30
Figure 18	Front Panel of initial interface VI using LMS 200 device	32
Figure 19	SICK Optic LMS 200 laser measurement device	34
Figure 20	HC11 digital valve driver interface	42
Figure 21	Flowchart of full-scale MBC controller	43
Figure 22	Full-scale communications and control layout	44

Chapter 1: Introduction

Coal mining is an essential source of energy for powering the global economy. As yearly demand increases, coal-mining companies must seek new means to streamline operations and harness more efficient technologies in order to remain competitive. The days of the pick ax and mule or horse cart are long gone.

Underground coal mining typically consists of removing coal from a seam sandwiched between rock or other material, and the seam ranges from 30” to 70” in height. The advent of the Continuous Miner brought the ability to mine large quantities of coal in very short periods of time. However, the bottleneck in mining operations was the removal of coal from the miner to outside the mine. To reduce this bottleneck, the Long-Airdox Company developed and produced a Continuous Haulage System (CHS) for underground coal mining. Figure 1 shows a picture of the Full Dimension Continuous Haulage System ready to enter a mine.



Figure 1. Long-Airdox Full Dimension Continuous Haulage System

The CHS is essentially a driveable conveyor system that is capable of following the Continuous Miner throughout the mine. As the Continuous Miner removes coal from the seam, it is fed to the first unit of the CHS – called the feeder-breaker. The feeder-breaker, as its name implies, breaks the coal into smaller sizes and then sends the coal on its journey through the CHS and out of the mine at rates up to 20 tons of coal per minute, depending upon the model. With this great capacity to move coal quickly, dramatic increases in coal production can be achieved. Figure 2 depicts a multiple unit CHS navigating a typical mine.

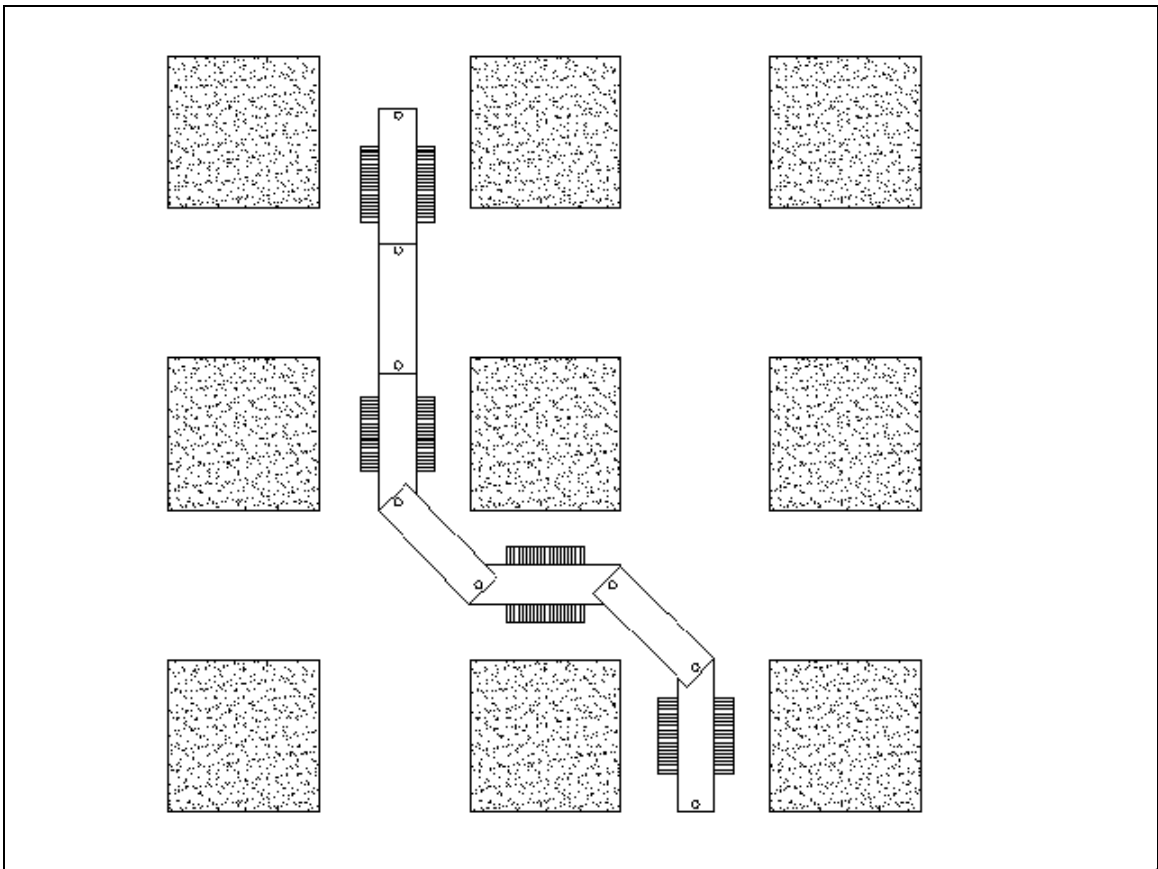


Figure 2. Depiction of a CHS navigating a Coal Mine

The three main parts of the CHS are the MBC (Mobile Bridge Carrier), the Pig (Piggyback Conveyor) and the RFM (Rigid Frame Modular) tail-piece. As its name implies, the MBC is a tracked vehicle supporting the Pigs and has a driver located in the right rear of the MBC. The Pig, which varies in length from 30' to 40' depending upon the CHS configuration, is a rigid conveyor section used to span two MBCs or the last MBC and the RFM. One MBC and a Pig are considered a unit and 5 or more units might

be linked together in a typical mining application. The RFM connects the last MBC to a stationary type conveyor system for the final stage of transferring coal to the surface. This section of conveyor belt is not very mobile and must be dragged into location by an MBC, shuttle car or some other machine.

As hopefully can be inferred from the figures and brief discussion, the CHS requires a skilled team of operators to efficiently traverse the mine. Since coal mining by nature commands high wages, the yearly costs for skilled operators can be quite expensive. These annual costs are overshadowed by the fact that coal mining is a dangerous business. Even though mining safety has been greatly increased, the potential for catastrophe is omnipresent making any reduction in the number of persons necessary in a mine highly desirable. To address these and other concerns, Long-Airbox has expressed the need to automate the Continuous Haulage Systems to increase system efficiency and coal throughput.

As a result, Long-Airbox and VA Tech are working in close collaboration to develop the necessary technologies to automate a Full Dimension Continuous Haulage System. To this end, the VA Tech team is tasked with research, development and testing of the necessary sensing, data analysis, driving rules, control algorithms and hardware re-design. The VA Tech team is responsible for developing the required technologies for automation and providing the necessary technology transfer through documentation.

In order to gain insight to the problem, the team members were able to drive a CHS that was being refurbished for a mining company in early Fall semester of 1998. Figure 3 shows the refurbished CHS that was test-driven by the VT team. Note the first unit is the feeder-breaker, with the wide front that catches coal being fed from the Continuous Miner. After driving the CHS, it was quite apparent that a high degree of skill and cooperation between team members is required to efficiently traverse a mine. The inertia and system response was observed in order to lay a foundation for the automated control system. Armed with a better understanding of the CHS, development of the necessary technologies for automation resumed.

A major focus on automation was path planning; how the CHS would navigate through a mine. Path planning is heavily used in robotics, where a robotic machine must

navigate within some workspace. Typically two means are used for navigation; a robot can either have the layout of a workspace programmed into its memory, or it must



Figure 3. Continuous Haulage System used for VA Tech Team Test Driving

sense its location with respect to its surrounds and navigate accordingly. Because all mines do not share exact layouts and are not typically cut exactly to specifications, requiring that a company operating an automated CHS program the mine layout was not deemed a suitable solution. However, requiring that the automated CHS be capable of sensing its location within a mine and navigating accordingly requires more effort and sophistication in the software algorithms, but is thought to provide a more flexible and intelligent system. Because of the strategy adopted, sensors are needed to measure the distance and incidence of the walls. Outfitting the CHS with enough sensors to fully describe its configuration at a given moment in time is also necessary. All this

information will have to be gathered and processed in order to issue the appropriate position or velocity commands to each MBC in the CHS.

In order to develop, test and demonstrate competency with the sensing, path planning and control algorithms, a suitable test bed is needed. Having production MBCs available for instrumentation and testing at will is not possible, therefore an inexpensive alternative is necessary. The author has been tasked with the development of a 5-unit prototype Continuous Haulage System that will provide continual development and testing of the overall automation strategies. The prototype development includes scaled models of an MBC and Pig, and the low-level microcontroller-based motor controllers necessary to provide motion. Responsibilities have grown from just developing and constructing the prototypes, to developing sensor interfaces and the communications hierarchy necessary for gathering and parsing all the data to a laptop PC for computation of all algorithms. All computations will be performed on an IBM Thinkpad laptop personal computer because it is the most cost-effective means for the prototype. Although Long-Airdox intends to outfit each MBC in a production CHS with a custom designed PLC (Programmable Logic Controller), their estimated \$6000 price tag places them beyond the reach of the initial project budget. Any testing on full-scale production MBCs requires specific hardware and software, though as much of the prototype equipment as possible will be modified for consistency and reduced development times.

Because the authors' work on this research project has been heavily project oriented and has required the creation of much hardware and software, this document serves as an important source of documentation for the remaining team members who will have to use the hardware and software in future testing. In the following sections, overviews of the prototype vehicles, electronics and software are presented. A discussion on the operation and use of the SICK Optic LMS 200 laser measurement device is included. Although the topics are all intertwined, they have been separated in attempt to provide clarity to each subsystem. Finally results from current testing and conclusions/recommendations will be made.

Although the author is somewhat disappointed to be graduating prior to total completion of the project, it is hoped that this document will serve as a useful and beneficial tool for the other team members.

Chapter 2: Prototype Continuous Haulage System

2.1 Prototype Introduction

The prototype Continuous Haulage System has many levels to its development and construction. On the basic level, a properly scaled clone of the production CHS was needed. The main requirements for the prototype structure were rigidity, reasonable weight, consistent scale and proper function. The two main structures to replicate are the MBC and the Pig. Since the Pig is modeled as a rigid link for the purposes of the prototype, the main functions to replicate were the MBC TRAM LEFT, TRAM RIGHT, IN-BY, OUT-BY and the dolly travel. TRAM refers to the controlling the speed of the tracks, while IN-BY and OUT-BY changes the elevation of the front or rear conveyor sections. Since they do not appear to place any requirements on the control system the prototype would not incorporate the IN-BY and OUT-BY functions. Long-Airdox assumed responsibility for developing a separate system for controlling these functions. The dolly travel allows compliance between two MBCs by enabling the front pig pin to slide five feet along the front-to-rear axis of the MBC. This extra compliance is deemed essential for driving the CHS through a mine.

The next level of development has two parts; developing the prototype electronics hardware and the software which includes the microcontroller-based motor controller, multi-processor communications for data gathering and interfacing to a control PC. The electronics system was chosen to provide a scalable system—as more functionality or processing power was needed, extra microcontrollers could be added to perform the required additional function

The final level of prototype development pertains to a high-level interface and control program running on an IBM Thinkpad laptop computer. The interface and control program is responsible for receiving in all sensor data, performing all necessary data analysis, path planning and control algorithms and parsing command velocity data back to the appropriate MBC. Since cost prohibits use of the Long-Airdox PLC, all inter-MBC communications expected between full scale MBC PLCs must be simulated by the interface and control program. Although these levels are heavily intertwined, discussions on their development will be separated in an attempt to provide clarity for each.

2.2 Prototype Hardware

The first step in prototype development was deciding upon a suitable scale for the models. Since an MBC drives much like a military tank, a RC (Radio-Controlled) tank model was viewed as a suitable base for the prototype MBCs. By using RC tanks as the foundation for the prototypes, it was hoped that significant reductions in development time would be realized. As a result, available RC tank models somewhat drove the prototype scale. After reviewing the sparse information on various models, it appeared that most tank models were approximately 7-9% of the full-scale MBC. However, after purchasing two models it was apparent that available radio-controlled tank models had some significant disadvantages.

Although the first tank model purchased was very inexpensive, it was very flimsy being made of plastic and more suited to higher speed operation. Because low speed control is critical, extensive modifications to the gear train for additional speed reduction would be required. Abandoning the first model, a King Tiger Tank model from Tamiya America, Inc. was purchased on recommendation from a RC model dealer because the chassis was made from stamped aluminum and the tracks were metallic. Even though the model is quite expensive, having metallic tracks on the prototype is ideal. However, the models are no longer manufactured with metallic tracks; only plastic tracks are currently produced. Although disappointing, the model was larger and more ruggedly built than the first model. During assembly of the model, it became apparent that the King Tiger Tank model would also require heavy modifications to the powertrain. The modifications would be necessary because it had only one motor controlling both tracks; directional control of the factory model is accomplished by engaging and disengaging clutches, implying the model is incapable of reverse. Therefore, a second motor would be required to provide separate, reversible control of each track. Modifying the powertrain proved to be a rather involved task, necessitating many hours of custom machining. Another concern with the RC models was the uneven scale; typically the width of the model was a desirable scale, but the length was much too great. Because of all the problems encountered with the models, design of custom prototypes was viewed as more cost-effective and a more efficient use of time.

Designing custom prototypes involved a few important considerations, of which scale was again the starting point. Since the both RC tank models were odd sizes, it was decided to make the prototypes an even 10% scale replica of the CHS. This scale would provide a larger platform for supporting the necessary sensors and hardware needed for the project.

The drivetrain and motors would be specified first, and then the chassis would be designed accordingly. The outside-in design methodology was used to keep a consistent scale and to simplify the design; it started with the tracks and worked towards the inside of the model. Since a source for properly sized steel tracks was unavailable, the plastic tracks and drive sprockets from the King Tiger Tank model were incorporated into the design and were purchased from Tamiya America, Inc. Because the drive sprockets had a 2" outer diameter, little ground clearance would be available. Therefore, selecting a motor that would provide enough torque at scaled prototype speeds while providing suitable ground clearance became a significant issue with the design. Using a geartrain or flexible coupling as a means to elevate a large motor and increase ground clearance would add cost and complexity - not a highly desirable option. After scouring the catalogs of many electronic hardware suppliers, some small gearhead dc motors with an offset output shaft were located. Because of the integral gear reduction, these motors had a slow output shaft speed with good torque and would allow direct mounting of the output shaft to the sprocket via a simple, custom-made hub. An added benefit of these particular motors is the integral optical encoders, which allow for position or velocity feedback. The motors were purchased and fitted to a prototype test chassis; it was a compact design, but appeared to be quite feasible.

With the drivetrain and motors specified, the chassis was designed as a simple structure made from 16-gauge mild steel sheetmetal stamped into a U-shape. A lip on top of the chassis is provided as a mounting surface for the canopy. A template was made so that all machining to mount the motors and drivetrain be completed before stamping, allowing the five prototype chassis to be machined at one time. Once machining was completed, the parts were then stamped into final geometry. Figure 4 shows a rear view of an assembled prototype MBC to give a better detail of the motor mounting.



Figure 4. Rear View of Assembled Prototype MBC

Being made from 16-gauge mild steel, the prototype chassis are quite rigid for the application. No extra stiffening is incorporated because the prototype canopy would provide added rigidity when fastened at assembly. With the prototype chassis design completed, the prototype canopy was next.

The deck is designed as a welded assembly. A piece of sheetmetal matching the outline of the chassis forms the base of the canopy. The canopy bridge needs to have the proper scale width and be rigid enough to support the Pigs and any additional sensors or hardware. A piece of sheetmetal was stamped into a channel to provide the necessary strength. The bridge is properly aligned with the base and clamped in place. With a final recheck of location, the two pieces are MIG welded together. The assembly is fastened to the chassis by 4 #10-32UNF screws. With the deck fashioned, the dolly travel mechanism was designed.

Because the dolly travel provides much needed compliance between MBCs to allow the CHS to snake around mine pillars, incorporating the dolly displacement into the

control algorithms is necessary. The production MBC has a dolly travel of five feet, requiring a prototype dolly travel of 6 inches. The initial dolly design incorporated precision ground steel rod and linear bearings. However, this option was quickly discarded in favor of using a precision drawer slide for simplicity and reduced cost. A travel stop was needed since the drawer slide is capable of extending to ten inches. A piece of mild steel stock was welded to the top of the drawer slide as part of the travel stop, and also to provide a mounting point for a measurement device. A bolt and was fastened through the deck at a point six inches from the welded bar, so that travel would be limited by contact between the bar and the bolt. These features provide a simple and effective solution to the design requirements. Figure 5 highlights the canopy, tag-line potentiometer, dolly travel and travel stop.

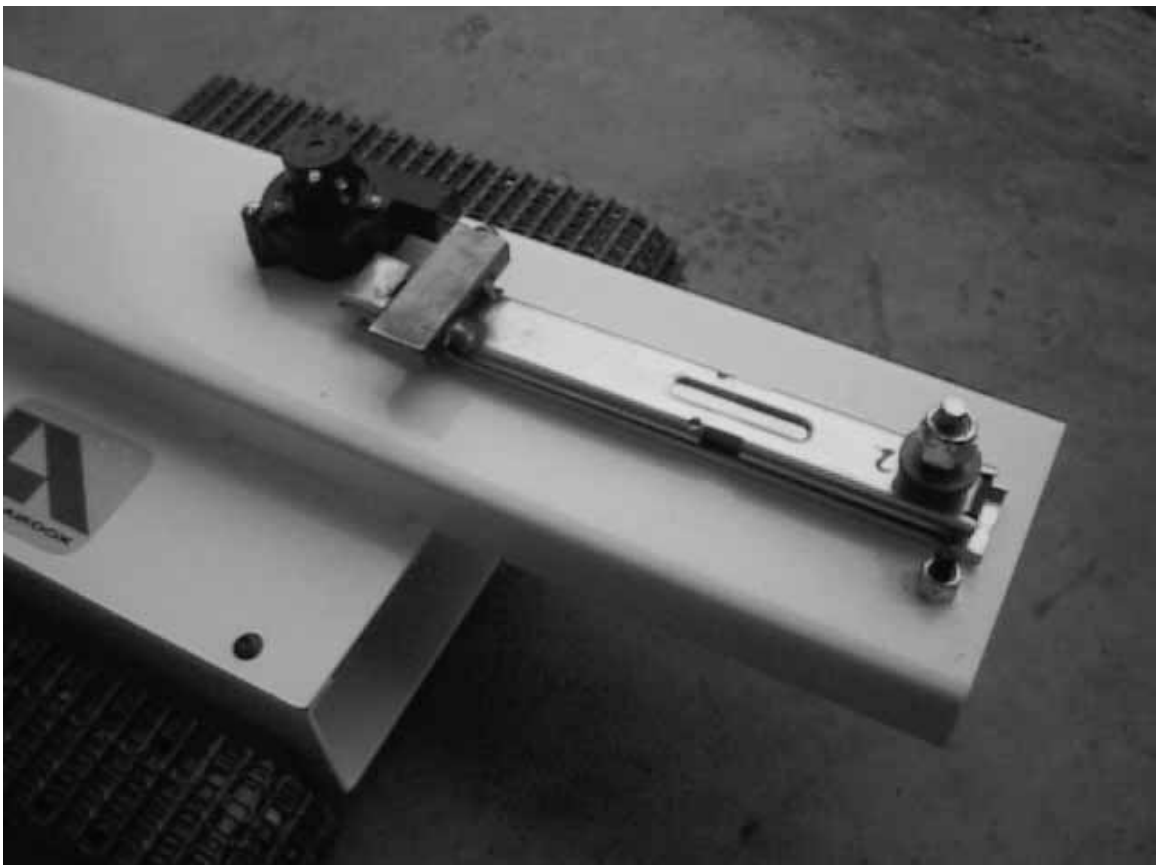


Figure 5. View of Prototype Deck and Dolly Travel Mechanism

Because the Pig is designed as a simple U-shaped channel, stamped from 16-gauge mild steel sheetmetal, the final design consideration for the prototypes was the

development of the pig pin and the associated joint design for coupling the MBC and Pig together. Since the joint would also have to incorporate a rotary potentiometer with a ¼” diameter shaft, a suitable flexible couplings were sought. However, precision flexible couplings turned out to be a rather bulky and expensive option. Therefore, the resulting solution would use ¼” inside diameter rubber fuel hose, small hose clamps and a modified bolt. The Pig would be modified to include a close sliding-fit hole for the modified bolt and the potentiometer mount.

Because the pig pin must sit on top of the dolly slide, the head of a 3/8”-16UNC bolt was machined flat then drilled and tapped for a #10-32UNF screw. This would allow the bolt to be fastened to both the deck and drawer slide without affecting the operation of the drawer slide. The threaded end of the bolt was machined down to a ¼” diameter to provide a pin-like area to insert into the fuel hose upon assembly, the remaining thread would be used for loosely fastening the Pig and MBC together with a teflon locknut.

The potentiometer mount was made from a piece of sheetmetal, stamped into a U-shape and drilled to accept the potentiometer. The potentiometer mounts are fastened to both ends of the Pig, making sure that the potentiometer is inline with the pig pin. This design makes assembly of the joint quite simple. The pig pin is inserted into the clearance hole on the Pig. The nylon locknut is tightened snugly, providing just a slight bit of clearance for rotation. As the potentiometer is fastened to the mount, a piece of fuel hose is slid down the shaft of the potentiometer and then over the machined section of the pig pin. The hose clamps are tightened on the pig pin and the potentiometer shaft. Figures 6 and 7 show the joint before and after the fuel hose is correctly attached.



Figure 6. MBC/Pig Joint Initial Assembly



Figure 7. MBC/Pig Joint After Final Assembly

Some care must be given during assembly of the joint to ensure that the potentiometer is properly adjusted so valid measurements are received by the interface program. Before fully tightening the clamps, the pot must be centered. The potentiometer can be centered by measuring the output voltage with a voltmeter or by monitoring the angular display in interface program. If the potentiometer is correctly adjusted, the output voltage will be 2.5 volts whereas the interface program should read 0 degrees on the appropriate pig angle indicator. Once the potentiometer is adjusted, the pig joint assembly is complete. Figure 8 shows two MBCs and a Pig connected.



Figure 8. Prototype CHS Featuring Two MBCs and a Pig

Please Reference Appendix A for detailed prototype drawings, specifications and hardware on the Prototype Continuous Haulage System. With the MBCs and Pigs designed and constructed, development of the motor controllers and communications hardware and software is the next phase of the prototype development.

2.3 Prototype Electronics

Before any electronic hardware could effectively be specified, it was essential to identify the sensors, measurement devices, and tasks required for developing an autonomously navigating prototype CHS. Since the system must gather and transmit measurement data, a system based on microcontrollers will be used. Analog devices will only be used as passive components in driver and digital circuitry. An educated estimate on the types and number of sensors and the functions to perform is crucial to specifying appropriate and upgradeable microcontrollers for the system. As the computing power required for the project is still uncertain, using a PC for all computations is the most cost-effective solution. As computational requirements are more fully understood, the future test hardware could be modified accordingly.

Itemizing the requirements for the electronics system starts with the basic function of the electronics system; controlling each of the dc motors needed for driving an MBC. The initial plan does not include monitoring of the MBC velocity because a lot of track slippage is expected in mining operations, making accurate measurements quite difficult. However, if deemed necessary at a later date, the MBC must have the capacity to measure the each track velocity. There are three displacements per MBC that need to be measured to determine the CHS configuration, the front and rear pig angles and the dolly travel. All three will be measured using analog potentiometers, requiring the prototype electronic system possess analog-to-digital capabilities. For measuring the mine walls, the prototype electronics must be capable of interfacing with either a SICK Optic LMS 200 laser measurement device or the LVS (Laser-Video Scanner) being developed by the VA Tech Team for the prototype. Interfacing with both of these sensors requires adequate communications capabilities. Finally, the system must also be able to communicate with a central laptop PC that will receive all measured data, perform the data analysis and path planning before sending out command data for velocity control of each MBC in the CHS. Although many requirements of the system have been identified, only testing will determine if these requirements are sufficient. Because of this uncertainty, it is important that the system have the ability to easily add new sensors and

functionality with minimal effort. Therefore, a multiple processor system is envisioned as the best means for achieving a powerful and scalable prototype electronics system.

Networking multiple processors is especially important so that the project does not become limited by hardware. Such a limitation might require a complete revision or redesign of the system to add a new sensor or function. As more sensing is needed, “smart sensors,” or sensors that have their own processors can be added to the network with reasonable effort. Therefore, processors with built in communications capabilities are a must. Given all of these criteria, a suitable processor could be specified.

Because the Motorola 68HC11 microcontroller has on-board communications capabilities and the author had prior experience with the chip, it was investigated as the first choice. The HC11, as commonly referred, has two on-board serial communications subsystems, a UART (Universal Asynchronous Receiver-Transmitter) and a SPI (Serial Peripheral Interface). The UART supports many standards of asynchronous serial communication between devices by using the proper driver. RS-232 and RS-485 are two very common and inexpensive standards. The RS-232 standard, which is found on all PCs, supports point-to-point communication between devices over relatively short distances. The RS-485 standard provides the ability for multiple devices to communicate on a single serial line over much greater distances than capable with RS-232. The SPI is developed for synchronous serial communications between microprocessors and peripherals. Peripherals are typically memory modules, device drivers, or other microprocessors. Reviewing the specifications for the many standards for detailed information is recommended for anyone interested in the subject.

A major distinction between the two protocols is that the SPI is a synchronous receiver/transmitter; all processors connected via a SPI bus share a common clock signal. Sharing a clock signal line creates problems when transmitting over long distances due to noise and other effects. Another difference is that SPI uses a slave select line. When operating in a master-slave layout, the master processor will drive the slave select line to a low state (0 volts) notifying the slave processor to commence data transfer. Because the SPI can be configured in a master-slave relationship between processors, it provides a flexible means to continually upgrade the system to meet growing demand. These differences are shown in Figure 9.

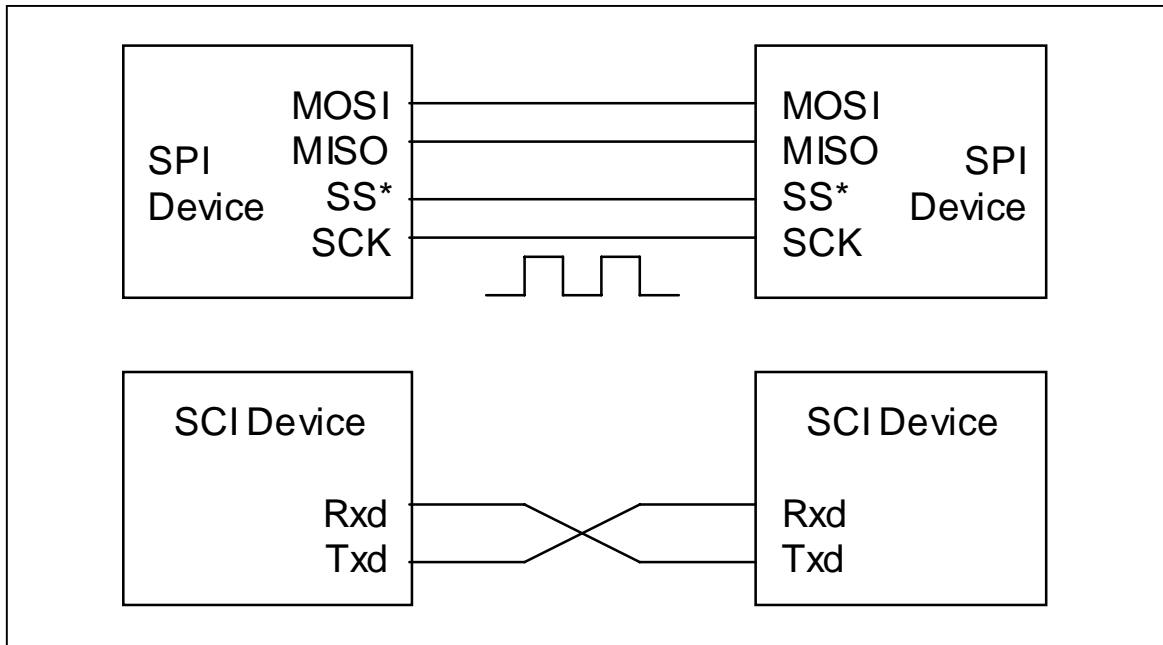


Figure 9. Connection Layout for SPI and SCI Devices

In addition to the serial communication features, the HC11 has an 8 channel, 8 bit ADC (Analog-to-Digital Converter) to measure the potentiometers. The HC11 has timer functions including the OC (Output Compare) function, which can be used for PWM (Pulse Width Modulation) signal generation for motor controls. With an optical encoder attached to each track motor, the IC (Input Capture) feature can be used for velocity measurement of the MBC by measuring the period between successive pulses from the encoder output. Because of these features, the HC11 microcontroller was chosen as the foundation of the prototype electronics system. Instead of developing a custom HC11 controller board, the Motorola MC68HC11EVBU [1] evaluation board was selected as the platform for the HC11 microprocessor for both the master and slave controllers due to its low cost and ease of expandability.

Since the HC11 can function effectively in a master-slave configuration using SPI, it seemed logical that a basic prototype electronics system would contain at least one master and one slave HC11 controller. A slave HC11 controller would be tasked with performing the signal generation for motor controls and if necessary, performing velocity measurements for closed-loop feedback of the motors. The master HC11 controller would be responsible for gathering and sending sensor data to the control PC and then parsing the command data from the control PC to the slave HC11 controller. The HC11

is not expected to be powerful enough for computation of the control algorithms, relying upon a PC for all computations. Figure 10 shows the expandable communications and control hierarchy developed for the prototype.

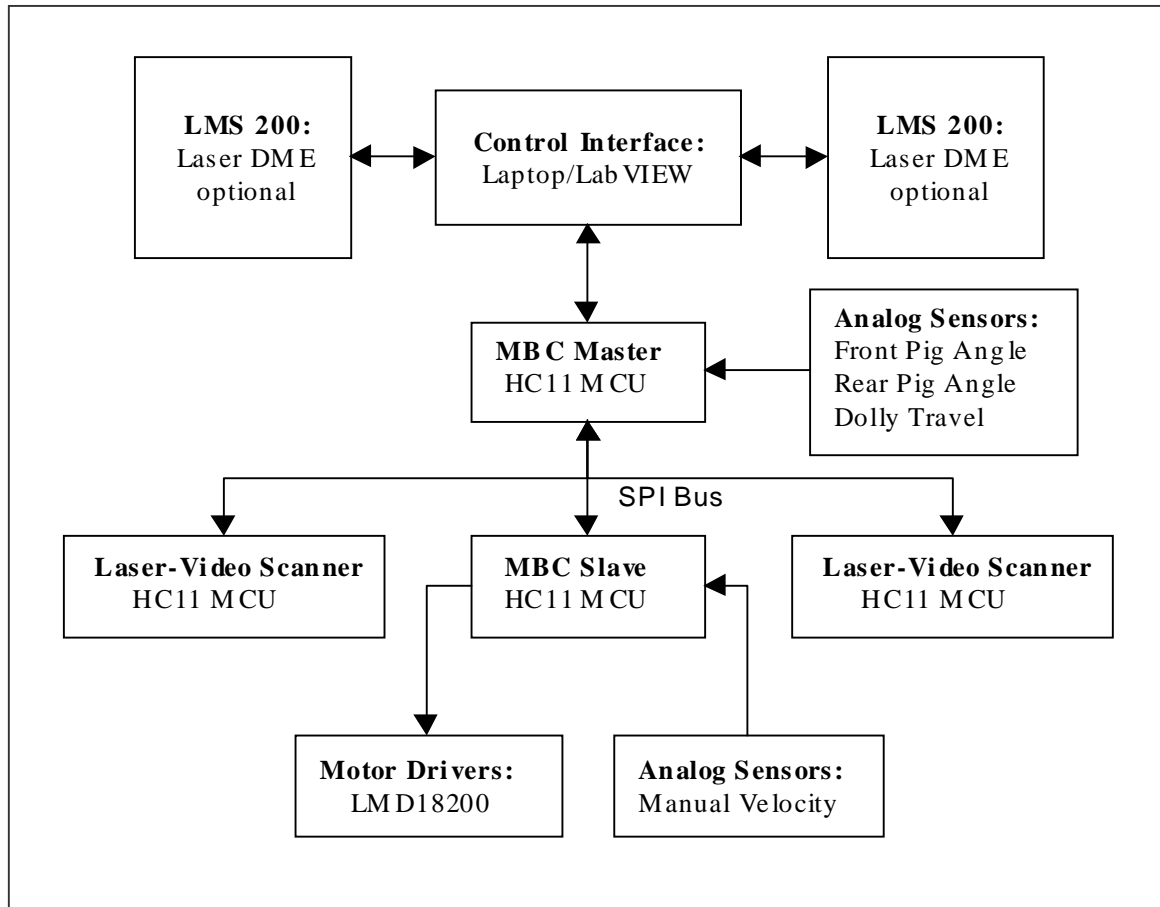


Figure 10. Prototype Control Hierarchy of a Single MBC

In Figure 10, a LMS 200 laser measurement device is shown along with Laser-Video Scanners. Both units will not be operated at the same time on a single MBC, however, it is possible that future testing will use different sensors on different MBCs.

With the basic hierarchy developed, the electronics hardware could be designed. The low-level motor controller using a single HC11 board was the first part developed. The motors are controlled with the PWM signals generated using two OC pins. The OC channels are TTL outputs and can source only 15ma. Because the dc gearmotors selected can draw about 1.5 amps under load, a driver was needed to amplify the PWM signals. There are many options for providing reversible motor control, but a single chip H-Bridge was desired. The LMD18200T H-Bridge from National Semiconductors is

designed for motion control applications and was selected as the motor driver. The integrated circuit is well matched to the power requirements of the motors and requires a simple interface with the HC11. For motor control, the LMD18200T requires a PWM signal, a direction signal that provide either logic high (5 volts) or logic low (0 volts) and a power supply. Because the H-Bridge inputs are TTL compatible, direct connection with the HC11 pins is possible. The schematic for the motor control circuit is given in Appendix B.

The only other hardware associated with the slave controller is various connectors for power supply, motor leads and the SPI bus. A ten-conductor ribbon cable is used as the bus for SPI communications between the master and slave controllers. The master controller has a simpler layout than does the slave controller. Terminal blocks for connecting to the potentiometers, a matching connector for the SPI cable and two additional SPI connectors for communications with the Laser-Video Scanner are the only other additions to the master controller board.

With the master-slave boards completed, the boards were mounted in the belly of the MBC. Figure 11 shows the completed boards mounted in a prototype MBC. Note that the master and slave HC11 controllers are located on the left and right sides of MBC, respectively.

With the master and slave boards completed, the remaining portions of the prototype electronics system pertain to mounting power supplies, wiring of potentiometers to the appropriate boards and ensuring all sensors and processors are connected to the control PC. This information can be found in Appendix B.

A discussion of the prototype software should serve to complete the prototype electronics system, and is explained in the following section.

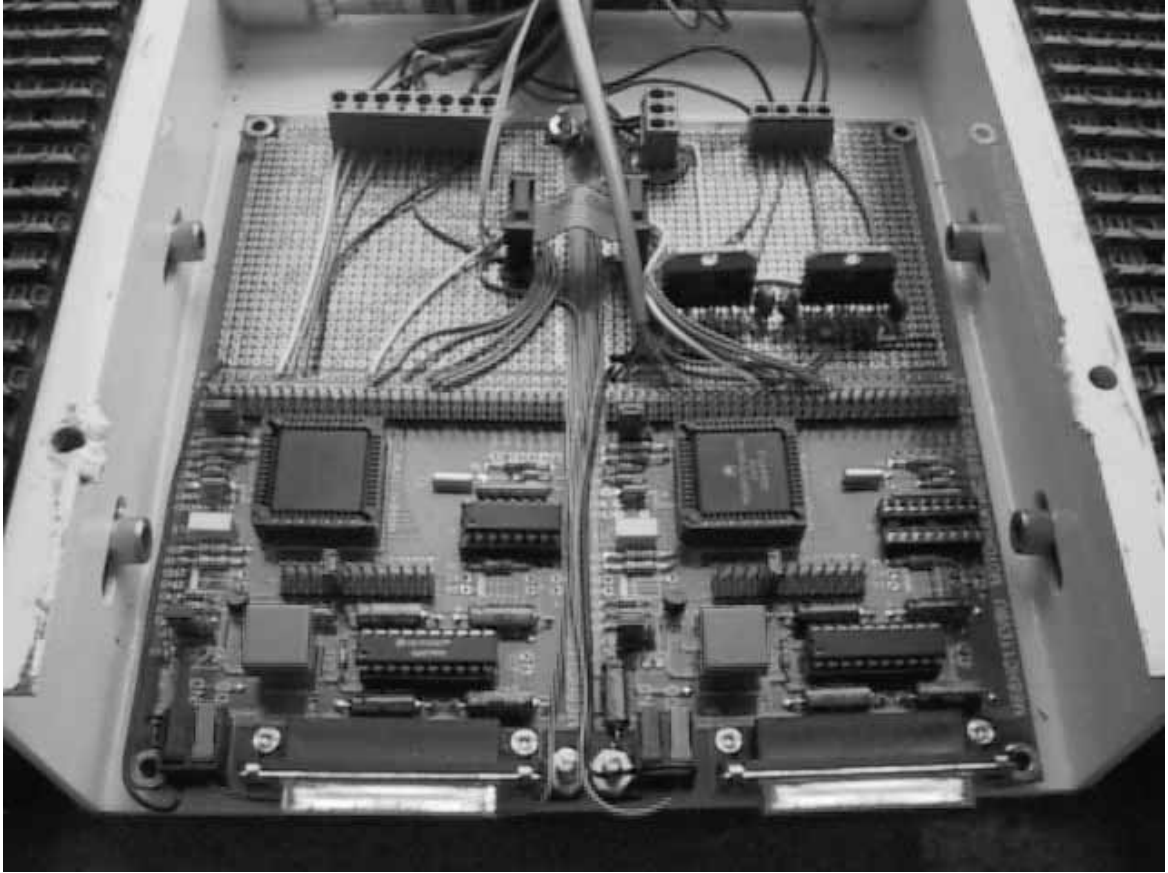


Figure 11. MBC Master and Slave Controllers Mounted in MBC

2.4 Prototype Software

The low-level controller software forms the foundation of the prototype software package. It is somewhat like the kernel in a computer operating system; it provides the low-level interface for handling the various functions and subsystems of the prototype electronics system. For example, the interface and control program will not directly provide the velocity control. After analyzing the data, it will generate velocity commands that are sent to the slave controller. The slave controller will then convert these velocity commands into the actual PWM signals required to drive the motor.

Since each prototype MBC has a master and slave based on the HC11 evaluation board, all programming will be completed in assembly language [2,3,4,5]. Assembly programming is processor specific, meaning that a program written for the HC11 will not likely be compiled for another processor without major modifications. If the programs are written in C, then they could be cross-compiled for other processors with reasonable

effort. However, since the prototype electronics system is different than what is expected for production hardware (Long-Airdox PLC), such portability is not necessary. Any line-finding algorithms that might be offloaded to an HC11 controller should be written in C, as these algorithms can be easily ported to the PLC.

The prototype has two modes of operation—manual and automatic, which are determined by the user setting a toggle switch. In manual mode, the operator will use two slide potentiometers as a joystick to control speed and direction of an MBC. Manual mode is used when trying to navigate the prototype to a test area, or when manually driving the first MBC in a CHS which has other units in automatic mode. The latter scenario is commonly called “follow the leader” because this is how the autonomous CHS is expected to operate; all MBCs will follow the front MBC that has a human operator. When operating in automatic mode, the MBC slave controller will not scan the joystick, instead it receives velocity and direction commands from the control PC via the MBC master controller. While operating in automatic mode, the master controller acts as a “traffic cop;” it is responsible for gathering sensor data, sending the sensor data to the interface and control program, and finally parsing command data to the slave controller. When operating in manual mode, the master controller waits for the operating mode to switch back to automatic mode.

The prototype software has been in continual evolution to meet timelines for testing. The initial software for testing is somewhat different than what is expected for the final prototype. The original plan incorporated sensors and measurement devices developed in-house for use on the prototype, due to the high cost of acquiring similar technologies from commercial sources. Once the necessary control algorithms and sensing was tested and verified on the prototype, development on the full-scale model would commence. Due to the rapid development of the project, two SICK optic LMS 200 units were purchased and delivered before any custom sensors were finished. As a result, parallel development of hardware and software was necessary for both configurations.

The first master-slave controllers developed were for use with the LMS 200 laser measurement device. Figure 12 shows the flowchart of the MBC master controller.

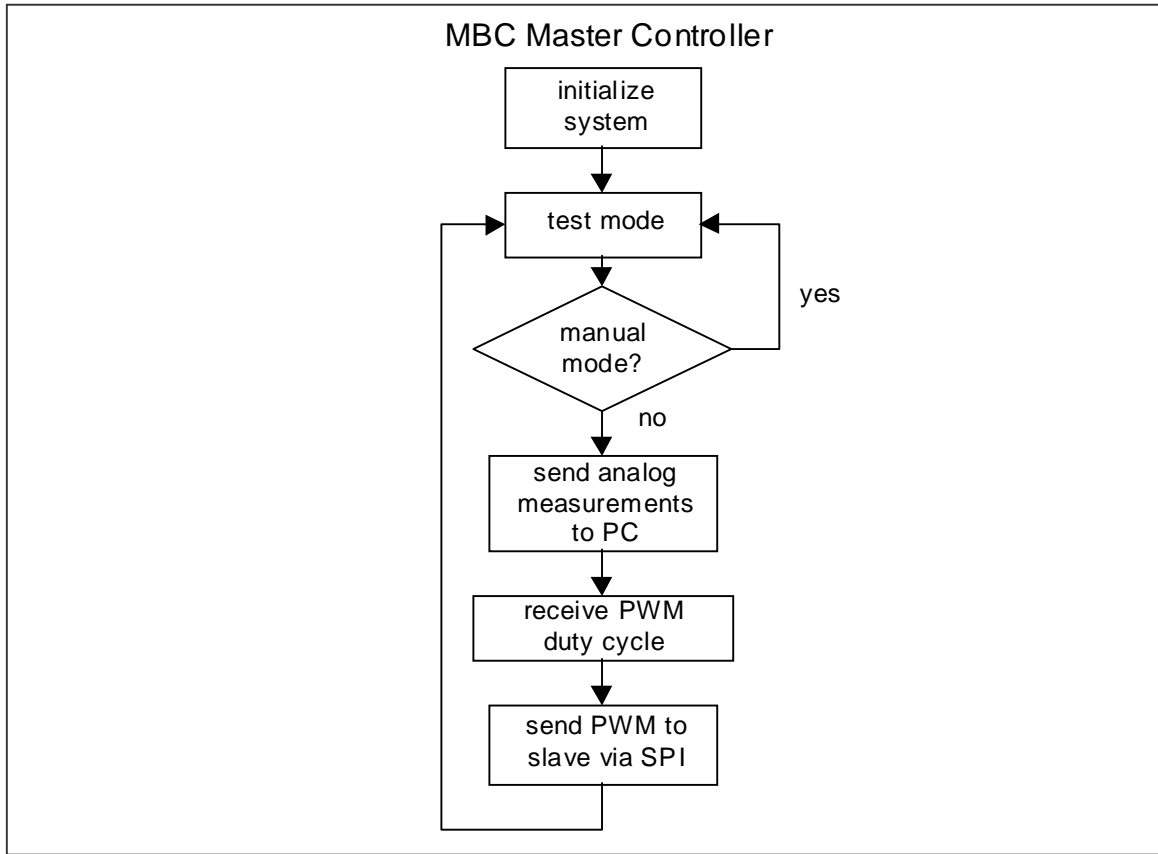


Figure 12. Flowchart of MBC Master Controller

Figure 13 on the following page shows the flowchart of the MBC slave controller.

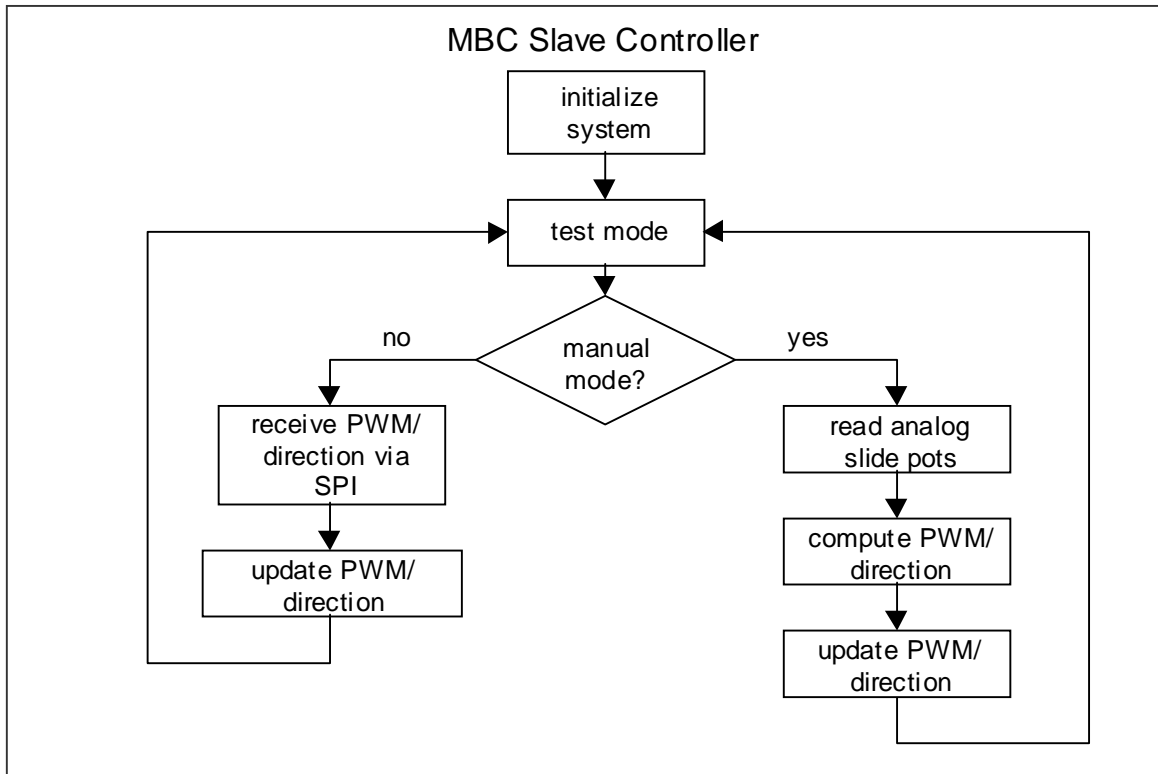


Figure 13. Flowchart of MBC Slave Controller

The LMS 200s are directly connected to the control PC via RS-485 PCMCIA cards, while the MBC master controllers are connected to the serial and parallel ports. Because the LMS 200 measurement device is capable of recording large quantities of data, it takes much less time to send data directly to the PC than if the data would be sent through the master controller first. The direct link also benefits the hardware of the MBC master controller; extra RAM (Random Access Memory) would be needed to provide suitable storage for all the LMS data. When using the parallel port, it is necessary to use a parallel to serial converter to interface correctly with the serial port of the HC11.

A modification to the original MBC master controller software was needed to interface with the Laser-Video Scanners being developed by Todd Upchurch, a member of the VA Tech Team. Since each scanner has an HC11 controlling the scanner, the MBC master controller software needed the addition of SPI communications between the scanners. The master controller commands a scanner to perform a scan cycle and then receives the data. The data is then sent to the control PC. No modifications were

necessary to the MBC slave controller. Figure 14 shows the modified MBC master controller flowchart.

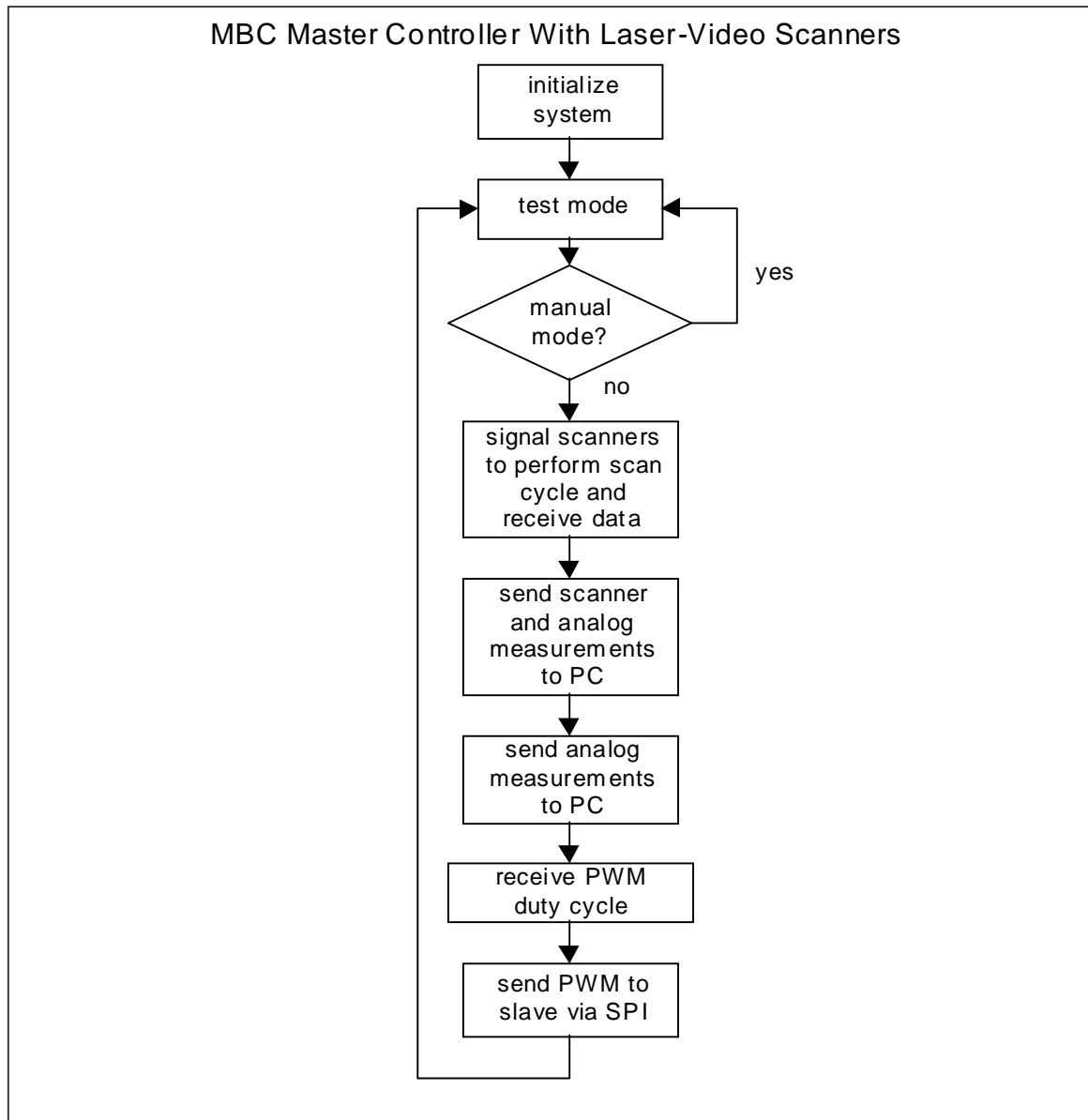


Figure 14. Flowchart of MBC Master Controller with Laser-Video Scanners

Although all initial testing has been performed using the first version of hardware and software, having only two LMS 200 measurement devices necessitates the use of the LV Scanner when a 5-unit prototype CHS will be tested. With two LMS units, a maximum of two automatic MBCs undertaking unidirectional turning can be achieved because the line finding algorithms will be blind to one side of the MBC. For bi-directional navigation, one MBC could be outfitted with both LMS units.

Another version of the MBC master-slave controller software is required for measuring the MBC position or velocity to provide closed-loop feedback for more sophisticated control strategies. The slave controller requires a significant amount of extra software to measure the velocities of both motors. The input capture timer function is used to measure successive transitions of the signal generated by the motor encoders. These period measurements are sent to the control PC through the master controller. The period measurements will be converted as needed by the interface and control program for use in a PID (Proportional Integral Derivative) control algorithm. The master controller software is modified to receive and send the motor velocity measurements. Figure 15 shows the flowchart for the slave controller incorporating motor velocity measurements.

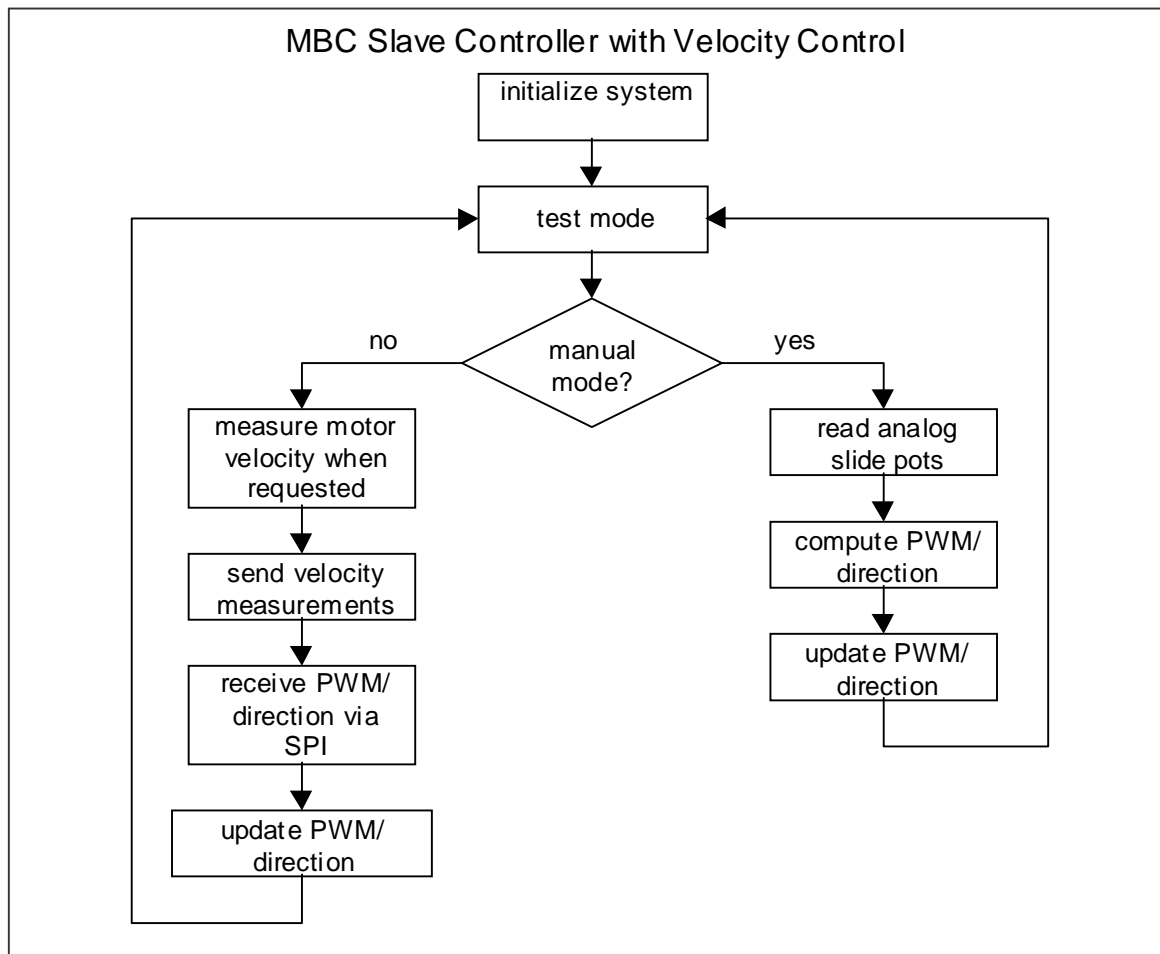


Figure 15. Flowchart of MBC Slave Controller with Velocity Control

Please reference Appendix C for detailed flowcharts and program documentation for the master and slave controllers.

Although the three versions of master-slave controller software packages discussed should meet all current needs, the number of units in a CHS might require some additional software and electronics due to a potential shortage of serial ports on the laptop PC used for the interface and control program. The current laptop PC has 4 RS-485 connections via a pair of dual-port serial communications PCMCIA cards, a parallel port and RS-232 serial port. The parallel port can be modified to become another serial port by using a parallel-to-serial converter. With RS-232-to-485 converters, each MBC master controller can interface with the control PC when using the Laser-Video Scanners. If two or more LMS 200 devices are connected to the serial communications card, then the MBC master controllers must be modified to operate on an RS-485 network. Although extra hardware is needed to change standards, the effect is transparent on the assembly software. However, when multiple master controllers share the same serial line, addressing becomes an issue. With a direct serial link, the master controllers only wait to receive a ready signal before performing their program cycle. With a multi-processor network, each processor must listen for the control PC to broadcast the address of a particular master controller. All master controllers on the network must compare the broadcast address with their own unique address. The master controller that matches the address will then perform the program cycle, while the other master controllers sit idle. Although the software is not currently written for a multi-drop network, it should not present major difficulties. Given that the majority of the master-slave software is in existence, a robust communications routine will be needed for the master controller. It will be required to decode the broadcast address and either perform the program cycle or sit idle until it is addressed at a later time.

2.5 Prototype Interface Program

A discussion of the prototype interface program, from selection of a suitable software package to successful interface with the MBCs is presented in the following sub sections. The interface program acts as the brains for the autonomous CHS.

2.5.1 Exploration of Software for Prototype Interface

A major decision for the prototype CHS involved the selection of a software program that would be suitable for use as the main interface and control program. The interface and control program must run on an IBM Thinkpad laptop PC using the Windows 98 operating system. The interface and control program is responsible for gathering all sensor data and analysis of the data for computation of the path planning and control algorithms in order to issue velocity commands to each MBC in the prototype CHS. Since the MBC processors need to communicate with the control PC via either an RS-232 or RS-485 network, serial communications capabilities are a must. A flexible software package that would allow rapid program development with the ability to visually display data was desired. Early research efforts examined use of the popular Visual Basic and C/C++ programs, and their respective capabilities. Both of these programs are quite powerful and capable, but appear to require serious programming efforts throughout all stages of project development. Using either of these high-level software programs is seen more as an end solution to write very optimized code for the automated full scale CHS.

During this software exploration phase, a new site license agreement between the College of Engineering and National Instruments for use of their LabVIEW software was completed. This was agreement was of great interest because LabVIEW is used by many departments throughout the university in both research and coursework to provide data acquisition and analysis of experiments. Since the license had been paid for by the College of Engineering and could be used for the project without cost, it was researched as a potential interface program.

While other programming systems use text-based languages to create lines of code, LabVIEW uses a graphical programming language called G to create programs in block diagram form[6,7]. LabVIEW is a general-purpose programming system with comprehensive libraries of functions and subroutines for most any programming task, much like C or BASIC. Since extensive libraries for serial communications were found, it appeared to be the flexible and powerful software program needed. Like using any new software package, some time was spent learning how to program in G. After a modest

level of competency was achieved, some simple programs were successfully developed that enabled data transfer between LabVIEW and an HC11 evaluation board.

In addition to the rapid development of programs, it was discovered that LabVIEW has some added features that would make it extremely useful for interfacing and commanding the prototype CHS. LabVIEW has the ability to call or run other software codes using the CIN (Call Interface Node). This ability to call other software programs from within LabVIEW has two main benefits. First, since the VA Tech Team members tasked with writing the path planning and control algorithms are fluent in C, it is of great benefit that learning a new language is not necessary. Secondly, since the algorithms are written in C, they can be ported to the Long-Airdox PLC with reasonable effort. This portability means reduced efforts when converting from prototype to production software. Given all the benefits, LabVIEW was chosen for the interface and control program.

2.5.2 LabVIEW Demonstration

Before proceeding with the development of the interface and control program, a very brief overview of LabVIEW is presented. A LabVIEW program is called a VI, short for Virtual Instrument. A VI has two “windows”; one is called the front panel and the other is called the diagram. The front panel is where the controls and indicators are displayed; it serves as the visual interface for the program. A control is how data and logic is input to block diagram. An example is a numeric control, which allows the user to change a particular numerical input. Other types of controls are boolean, string, arrays and clusters. An indicator is a display showing the output of numeric, boolean or string data from the program. Great flexibility in the appearance of indicators is available; indicators can range from a simple numerical output to a liquid level display of a water holding tank.

The core of LabVIEW programming is conducted on the diagram window. This is where the block diagram is located. Programming in G appears similar to wiring up an electronic circuit. Figure 16 shows the front panel and diagram panel of a demonstration program that displays the speed of a fictitious automobile engine on a tachometer. A

random number generator is used to create random numbers ranging from 0-1. The output of the random number generator is multiplied by a constant of 7000, which simulates a maximum engine speed of 7000 revolutions per minute. A dial indicator displays the resulting engine speed to a dial indicator.

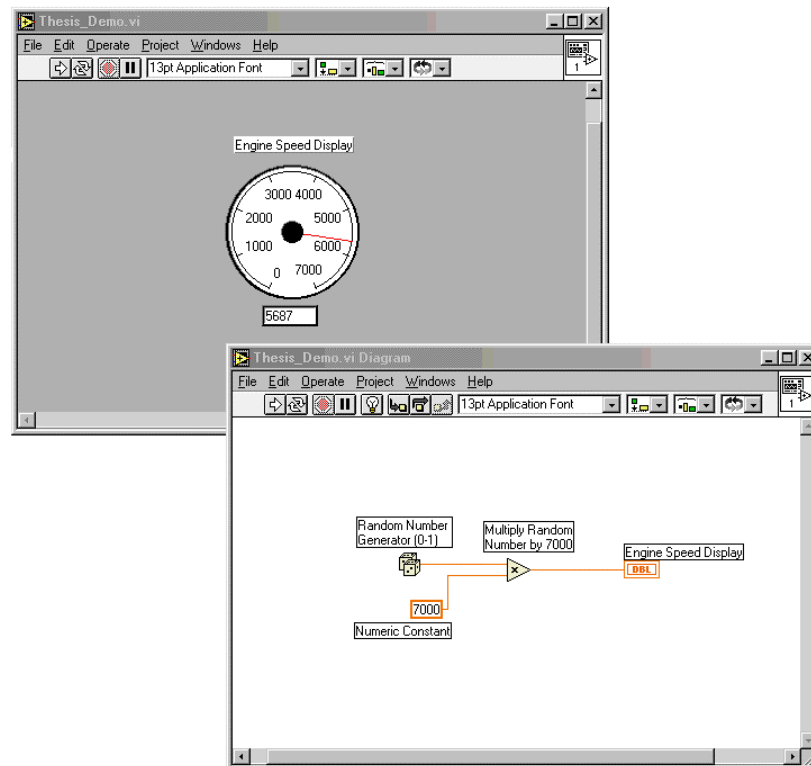


Figure 16. LabVIEW Demonstration Program

Although the demonstration program is a very simple example, it should serve to show the flexibility and power of the G programming language. The ability to rapidly update the interface and control program as project development advances is quite a luxury. As modifications and additional functions are necessary, the programmer simply makes the necessary change and re-wires the affected portions of the block diagram. The debugging and error checking features prevents a programmer from making many mistakes while creating and modifying the block diagrams. Should a VI not produce the desired results, very powerful debugging tools are available to expedite correction of the program. These are all highly desirable traits for the prototype development because of the dynamic nature of software and hardware needs. Only when competency in path

planning and control algorithms has been demonstrated should the efforts shift to writing production hardware-specific software.

2.5.3 Prototype Interface Development

With a better understanding of LabVIEW, a discussion of the interface code is appropriate. Since the LMS 200 laser measurement devices were the first sensors available for measuring the mine walls, the interface and control program was created to interface with the units. Figure 17 shows the flowchart of the interface and control VI using the LMS 200 devices.

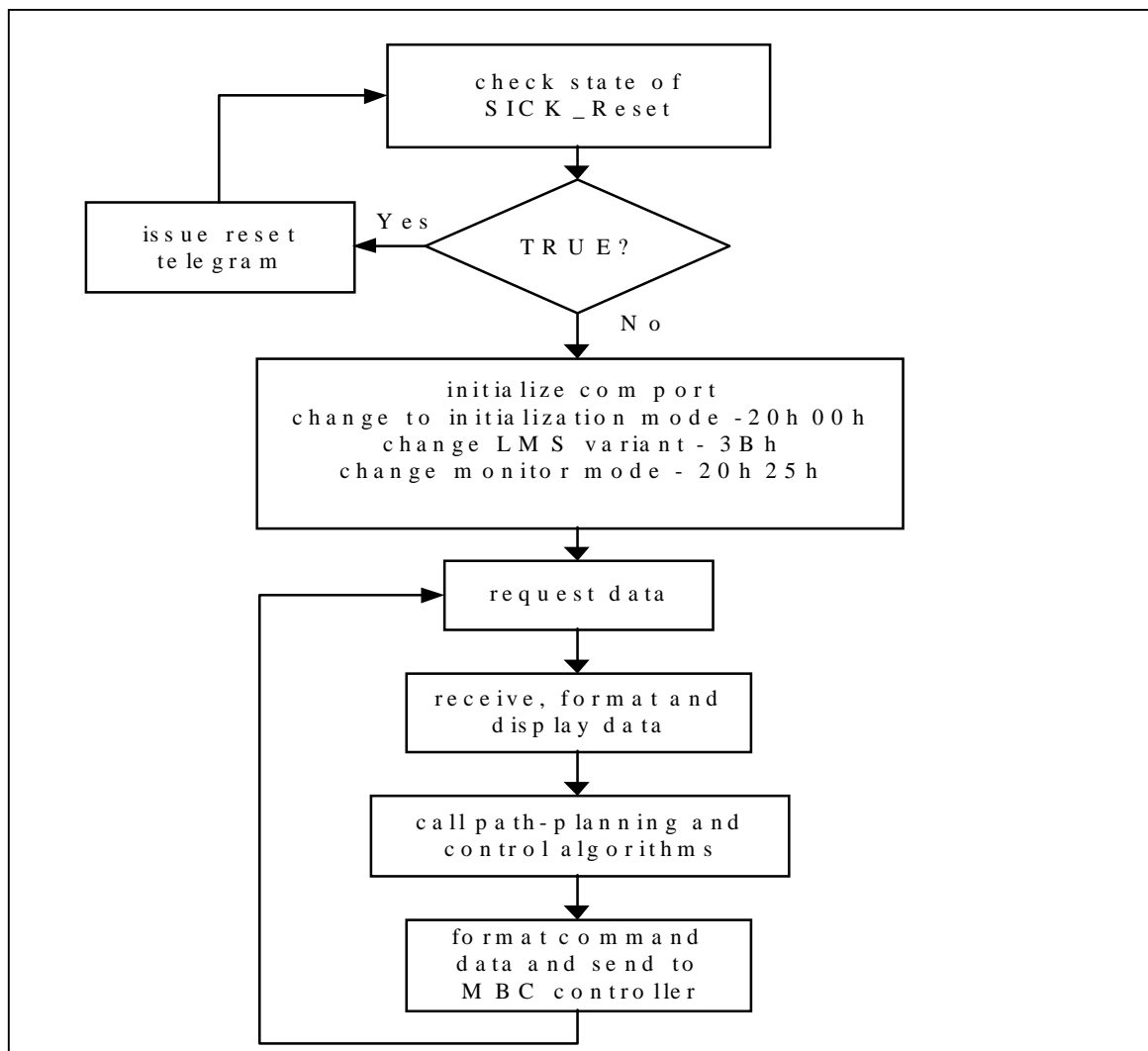


Figure 17. Flowchart of Interface Program Using LMS Devices

When the VI first executes, an initialization loop to configure the LMS is performed; details on interfacing and operating the LMS 200 device will be discussed in great detail in the following section. Once the initialization loop is completed, the LMS devices are ready to perform distance measurements. The VI will request a LMS perform a measurement scan and then wait for the data. After receiving all the LMS data from all units on an MBC, the VI will then address the MBC master controller and request all analog measurements. This process will be completed for each MBC in the prototype CHS until all data has been gathered. Since both the LMS and the MBC master controller transfer data in hexadecimal (base 16), the data must first be formatted into decimal (base 10) before being called by the path planning and control algorithms that are written in C. The control algorithms currently output a value between +1 and -1; +1 is for full forward velocity, -1 is for full reverse velocity and 0 is off. Since the MBC motor controller requires two 16-bit values to define the PWM high and low times for the OC function, the code output must be formatted appropriately before being sent to the MBC master controller. Figure 18 on the following page shows a screen capture of the most recent interface and control VI.

The interface and control VI requires only modest changes when using the Laser-Video Scanners in lieu of the LMS 200. The overall operations are the same, mainly the volume and format of the data is different. All data must also pass through the MBC master controller.

Going into further depth about operation of the interface and control VI would not be appropriate without first providing a thorough explanation of how the LMS 200 laser measurement device is configured and operated. More detailed explanation is available through reading Chapter 3 which discusses the LMS 200 and also referencing Appendix D for full documentation on the various interface and control programs.

Chapter 3: SICK Optic LMS 200

3.1 Sensor Background

Although it has been assumed from the early stages of the project that measuring the distance and orientation of an MBC with respect to the mine walls is essential to computing a path plan, simulation and dynamic analysis performed Aishwarya Varadhan and Amnart Kanarat have validated this assumption. Their efforts have established a control strategy requiring a line-finding algorithm capable of locating each MBC in the CHS with respect to the mine walls. The evolving algorithm requires measurement devices with the ability to sample mine walls with multiple data points in less than a second. To accomplish this task, either an array of point measurement or swept measurement devices can be used. Ultrasonic sensors and stationary laser devices are typically used for point measurements. Some laser measurement devices that can perform sweeping measurements by deflecting the laser beam by rotating optics are available.

In trying to determine the most suitable technologies, several factors must be examined. Quite prevalent, ultrasonic sensors can require a lot of expertise to ensure accurate and reliable operation. Acoustical reverberation from surrounding structures and cross talk between sensors can be serious problems. Ultrasonic sensors are typically quite cheap to use, so they have a strong economic benefit for the project budget. The performance and benefit of a swept laser measurement device appears proportional to their cost; they are typically quite expensive. The more intelligent the line-finding algorithm needed to become meant the increasing need for a swept laser sensor. Instead of requiring an array of point devices to measure the position of an MBC in the mine, a single swept laser would be capable of measuring all objects within a 180° range of the scanner. A decision on the technology to use was not made for some time, so efforts focused on both developing ultrasonic sensors, developing a swept scanner and procuring a commercial swept laser measurement device.

Even though testing had been done with the ultrasonic sensors, results from continued simulation studies showed that a swept measurement device was ideal for the

line-finding and control algorithms. Therefore, the search for a suitable swept laser measurement device progressed, as did continued development of a prototype swept measurement device.

One of the pioneers in laser measurement equipment is SICK Optic Electronic. They produce many different types laser measurement equipment, with LMS 200 appearing to be the best suited to project needs [8]. The LMS 200 device is capable of producing a 180-degree radial scan with an angular resolution of $\frac{1}{4}$ degree and a range of more than 30 feet. The unit is also exceptionally fast, capable of completing the 180-degree scan in less than 30 milliseconds. Communications between an interface computer and the LMS is accomplished by a serial link. Depending upon how the serial cable is configured, the serial output of the LMS will conform to either the RS-232 or RS-422 standard. Since the RS-485 standard is a superset of RS-422, a direct connection between the RS-485 communications card in the control PC and the LMS is possible. Figure 19 shows the LMS 200 laser measurement device.



Figure 19. SICK Optic LMS 200 Laser Measurement Device

As luck would have it, a LMS 200 laser measurement device is owned by the VA Tech Autonomous Project Team for use on their autonomously navigating vehicles. In order to benchmark the LMS unit, the Autonomous Team agreed to loan the device for

some short duration testing. Preliminary testing and data logging was conducted on the coarse stone buildings around campus, and in a coal mine using software borrowed from the Autonomous Team. The results were very encouraging, as the recorded data proved to be highly accurate. Even though Dr. Sturges and Todd Upchurch were developing a Laser-Video Scanner for the prototypes, the decision was made to purchase two LMS 200 for use on the above ground trials.

3.2 LMS 200 Telegram Structure

With the arrival of the two units, the author became involved in their development for the project because the only software supplied with the LMS was a simple demonstration program. Since it was not capable of meeting the needs of the project, a new program was needed to get the sensor data to the interface and control program. After a complete review of the LMS manuals, it was determined that interfacing with the sensor could be accomplished through a LabVIEW VI. However, developing a successful interface requires a thorough understanding of how to communicate and configure the device.

The LMS comes configured to communicate at 9600 baud and perform 180° scans with a resolution of ½°. The resulting data would only be sent on request from the interface program. Knowing the factory configuration was a good starting point because the preliminary interface program would only have to request the data be sent and then read in the data correctly. Sending a request for data would be accomplished by using what SICK Optic calls a telegram. By using telegrams, the interface program can configure the sensor, set parameters, receive data and perform diagnostics. The various telegrams are listed in the LMS 200 Telegram Listing Manual and are supplied in Appendix E for reference. As discussed in the manual [8,9], a typical telegram structure looks as follows:

STX	ADR	Length	CMD	Data.....	Status	CRC
-----	-----	--------	-----	-----------	--------	-----

Table 1 on the following page gives a description of each parameter, its data width and a brief comment.

Table 1. Description of LMS Telegram

Designation	Data Width (Bits)	Comment
STX	8	Start byte (02h)
ADR	8	Address of LMS contacted LMS adds 80h when responding to host computer
Length	16	Number of following data bytes excluding CRC
CMD	8	Command byte sent to LMS
Data	N x 8	Optional, depends on previous command
Status	8	Optional, LMS transmits its status message only when it transfers data to the host computer
CRC	16	CRC checksum for the entire data package

In order to correctly configure and use the LMS 200, these telegrams must be completely understood and manipulated. The following example is a configuration telegram sets the baud rate to the maximum speed of 500,000 baud. Note that the request telegram and LMS response is given in hexadecimal notation.

Interface Program: 02h/00h/02h/00h/20h/48h/58h/08h

LMS Response: 06h/02h/80h/03h/00h/A0h/00h/10h/16h/0Ah

The request telegram is disassembled and listed in Table 2.

Table 2. Disassembly of Interface Program Request Telegram

STX	02h	Start character for initiation of transmission
ADR	00h	LMS address
LENL/LENH	02h/00h	Length = 2 (2 data bytes follow)
CMD	20h	Select or change operating mode
MODE	48h	Configuration to 500,000 BAUD
CRCL/CRCH	58h/08h	CRC 16 Checksum

The LMS response telegram is disassembled and listed in Table 3.

Table 2. Disassembly of Interface Program Request Telegram

ACK	06h	Acknowledge receipt of telegram
STX	02h	Start character for initiation of transmission
ADR	80h	Host address
LENL/LENH	03h/00h	Length = 3 (3 data bytes follow)
BMACK_TGM	A0h	Response to change of operating mode
BMACK_TGM STATUS	00h	Mode change successful
STATUS	10h	Status byte
CRCL/CRCH	16h/0Ah	CRC 16 Checksum

With a thorough understanding of the telegram structures, virtually any software program with serial communications capabilities can be made to interface with the LMS. Thus, development of the first LabVIEW VI was initiated. Because the LMS was factory configured to send data only on request, the proper telegram to request data was needed. Luckily, the manual listed the necessary telegram in a section discussing the telegram structure. The first interface VI written sent a request to send data to the LMS, and then displayed the hexadecimal data by an indicator. Then the data was manipulated into decimal and displayed through a polar plot. Within a very short time, an understanding of LMS operation had been achieved and a simple interface program was written that enabled more thorough testing of the device, with data acquisition enabling analysis of the results. The only problem encountered was not being able to completely configure and use the LMS due to incorrect calculation of the checksum.

3.3 Calculation of the CRC 16 Checksum

Calculating the correct CRC 16 Checksum is essential for correct processing of interface program requests. The checksum has a unique value for any telegram and is calculated by the LMS with an algorithm using a polynomial generator. When a telegram

is sent to the LMS unit, it is stored in a data buffer. The CRC 16 Checksum algorithm computes a checksum based on the data in the buffer. If the resulting checksum matches the checksum sent with the telegram, the data is valid and an ACK symbol (06h) is returned to the interface program along with the results of the original request telegram. However, if there is an error in the data or the original checksum was incorrect, the LMS will return a NACK symbol (15h). By receiving either the ACK or NACK symbols, the host computer can determine if rebroadcast of the original message is necessary. Although somewhat difficult to follow, the checksum is essential to ensuring valid communications and data transfer between the host and LMS.

Because each telegram has a unique checksum, reconfiguration of the LMS required the proper checksum. Even though a particular telegram was correct except for the checksum, the incorrect checksum would cause the LMS to respond with a NACK. The manual provided the checksum algorithm in ANSI C. Since the initial interface program did not analyze the LMS response for a valid checksum, only the capability to calculate a correct checksum for a given request telegram was needed. Therefore, the checksum algorithm provided in the manual was modified to create an executable program that would compute the checksum for a given telegram. A simple interface that would accept the telegram string and output the checksum was created using LabVIEW. The simple program provided the ability to correctly calculate the checksum for any telegram, removing any remaining roadblocks to complete interfacing with LMS.

3.4 Development of LabVIEW Interface for LMS 200

The first step in refining the operation of the LMS was to reduce the number of data points received. Because the unit was currently configured for a 180° sweep with ½° increments, a total of 733 bytes of data would be sent. The LMS was reconfigured to reduce the current resolution to 1° increments, yielding a decrease in the time required to update the polar plot because the quantity of data had been cut in half. Other measures to increase the speed were investigated. Because the LMS unit is capable of completing a full scan in less than 30 milliseconds, the time required for serial communications can provide a major bottleneck. Therefore, the next performance upgrade was to change the

baud rate of the LMS to 19,200 baud. When compared to the initial VI, the results were dramatic; a 4-fold decrease in update time was now attained making the polar plot appear to update almost instantaneously. However, some problems were encountered with the interface after the VI was stopped. The power supply to the LMS had to be cycled in order to restart the VI. Because the VI changed the baud rate from the default 9600 to 19,200 baud, the VI would communicate at 9600 baud when restarted. However, the LMS would still be expecting communications at 19,200 baud. After cycling the power, the LMS would reboot at 9600 baud. Looking through the telegram listings, a command was found so the baud rate could be changed permanently. Thereafter, the LMS would always reboot at the reconfigured baud rate. The same command could be used if the baud rate needed to be changed back to the default of 9600. Once the LMS was reconfigured to always boot at 19,200 baud the VI worked without problems. The next phase in development of the interface with the LMS incorporated the line-finding algorithms that were being developed.

Since the line-finding algorithms are written in C, the LMS interface program was modified to call the algorithms using the code interface node. With the addition of the line-finding and control algorithms, the new program became the interface and control program for the prototype and full-scale development.

Chapter 4: Full-Scale Continuous Haulage System

4.1 Full-Scale Introduction

Although the prototypes provide a suitable testbed for project development, Long-Airbox is anxious to perform testing on their full-scale models. Such testing requires redirection of efforts and solution of new problems, but is necessary to ensure that development includes solution of any issues pertaining solely to full-scale equipment. Long-Airbox support for testing currently pertains mainly to providing full-scale, production MBCs as units become available. Much scheduling is needed so each phase of testing on the prototypes is recreated on the production models. Since testing both prototypes and production models is expected, minimizing effort to complete the hardware and software for both prototypes and production models is extremely important. Therefore, as much of the existing prototype hardware and software will be used for full-scale testing. Especially since the VA Tech team is responsible for providing a majority of the required hardware for the initial phases of testing. It is expected that future testing of full-scale MBCs might include hardware that is intended for production use, and will be provided and supported by Long-Airbox.

4.2 Full-Scale Electronics Development

Since a hardware and software interface had already been developed and tested on the prototype, the major differences between the full-scale and prototype models had to be investigated in order to determine how much software and hardware could be shared. Since the full-scale TRAM LEFT and TRAM RIGHT functions are controlled by manual operation of lever-actuated valves, a microcontroller-based interface was needed. The Long-Airbox remote controlled MBC was a logical starting point, so the hardware used for the conversion was investigated for possible use in the automation project. In the remote controlled MBC, the manual valve controls are replaced with Apitech Pulsar VS-series digital pressure control valves. These digital valves require a 33 Hz PWM signal for actuation and have a fairly simple operation; increasing the PWM duty cycle

increases the hydraulic flow, causing greater MBC velocity. The valves operate much like the dc motors on the prototype, necessitating only minor changes in the prototype controller software to control the digital valves. To that end, the digital valves were chosen for standardization and ease of interface. Reference Appendix F for manufacturer information on the digital pressure control valves.

The digital valves only draw about 0.45 amps, which is much more current than the HC11 can source. Therefore, a driver was needed to interface the HC11 with the digital valves. The H-Bridge used for the prototype motor control was not chosen because four chips would be needed, and such a configuration is neither cost effective nor efficient. Instead, a driver using an inexpensive darlington array IC was prototyped and for testing on a production MBC. Around the completion of the driver, Apitech digital valve driver modules were supplied by Long-Airdox. Testing on the production model with the darlington driver proceeded because the unit had already been finished. However, after the driver did not function effectively, use of the Apitech modules was deemed a wiser investment of time because they are specifically designed for the digital valves. The Apitech digital valve driver modules were quickly tested to verify their operation by using the joystick to provide a reference signal for the module. By varying the input voltage between 0 and 8 volts, a 33 Hz signal with duty cycles of between 0 and 100% are achieved. The driver modules are self-contained and intrinsically safe units, a MSHA (Mine Safety and Health Administration) mandated requirement for any electronic system in a mine.

In order for the HC11 to interface with the valve driver module, a circuit capable of using digital logic to create a variable output voltage to command the driver module. This task could be accomplished by using digital-to-analog converter (DAC) connected to each of the two 8-bit wide ports on the HC11; TRAM LEFT was connected to Port B and TRAM RIGHT to Port C. Since the DAC is an 8-bit device, writing a value from 0 to 255 in hexadecimal to either port easily changes output signal of the DAC. As the input value is increased, the output voltage increases accordingly. To get the appropriate output voltage range of 0 to 8 volts, standard 741 op-amps (operational amplifiers) were used with the proper gain.

Because two digital valves are required for forward and reverse of each track, a total of four valves would be required to drive an MBC. Because only two DACs can be connected to the HC11 without more complicated circuitry, a simple method of controlling four valves with the two DACs was needed. It was possible to switch the output of the DAC circuit between a pair of valve driver modules, requiring four modules. Another option was to switch the output signal from a driver module between digital valves, requiring only two modules. Since less is assumed better, the solution was to switch the command signal between the forward and reverse digital valves using a double-pole, double-throw relay. Port B would switch between TRAM LEFT forward and reverse, while Port C would switch TRAM RIGHT forward and reverse. The only additional hardware needed for using the relays were two digital I/O (input-output) lines from the HC11 driving 2N2222 npn general purpose transistors to activate the relays. Figure 20 shows the completed HC11 interface and Apitech driver modules. The circuit schematic and documentation is provided in Appendix F.

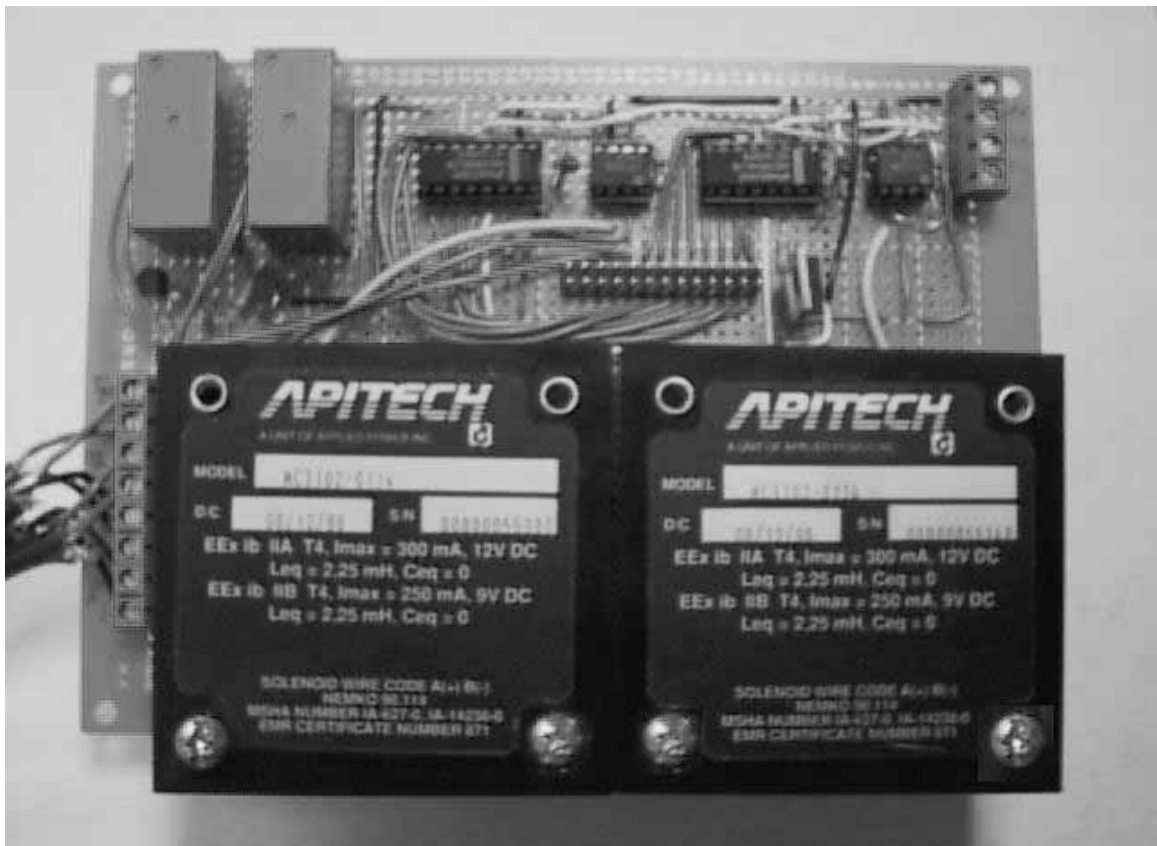


Figure 20. Digital Valve Driver

4.3 Full-Scale Software Development

To finish the full-scale MBC interface, the prototype controller software was modified to reflect the new DAC-based driver module circuit. Since testing of two MBCs with SICK Optic LMS 200 laser measurement devices would be conducted initially, the slave processors were abandoned. Instead, a lone processor would be capable of conducting all necessary data gathering and velocity control, and would still operate in either manual or automatic mode. Should the Laser-Video Scanner be used for above ground testing or additional slave processors become necessary, the typical master-slave hierarchy developed for the prototype could be modified for use. Figure 21 shows the flowchart of the modified controller software for production MBC testing.

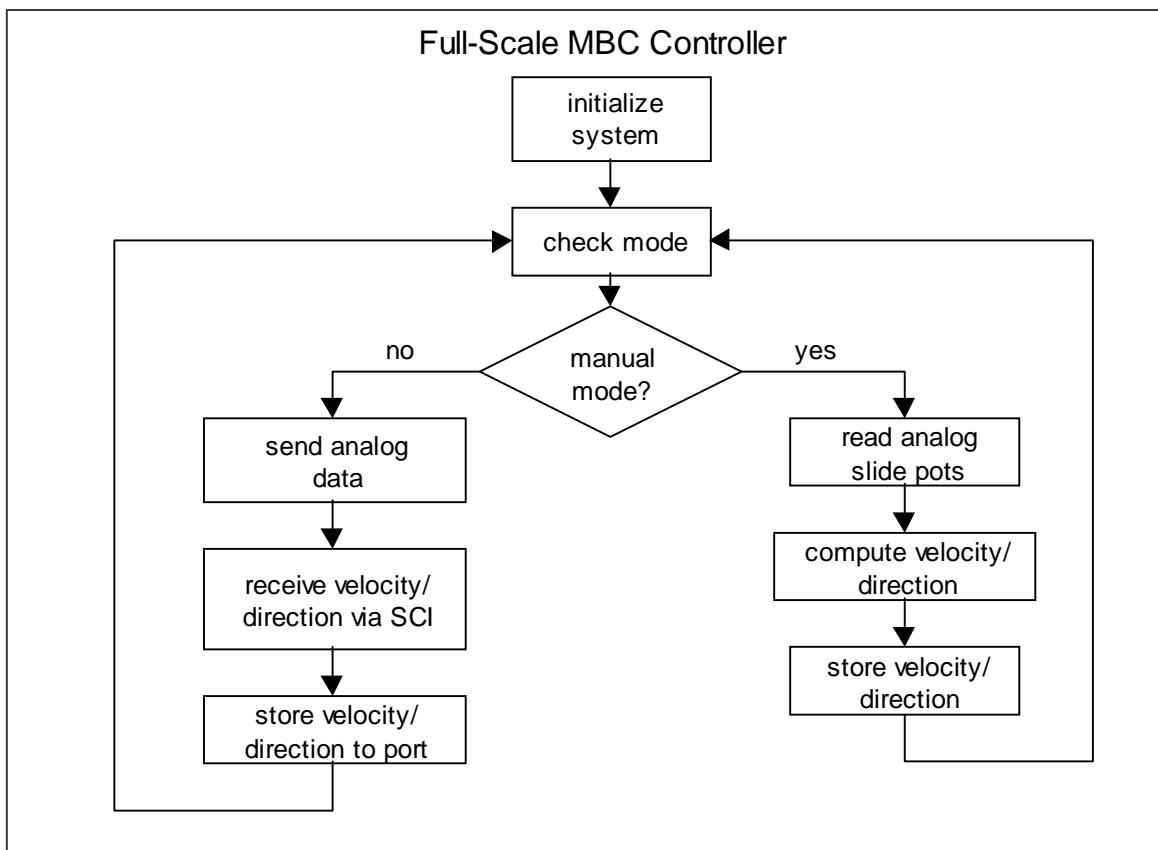


Figure 21. Flowchart of Full-Scale MBC Controller

Analyzing the flowchart in Figure 20, it can be seen that the full-scale MBC controller is a blend of both prototype master and slave controllers. Because the LMS units have a direct link to the control PC, a master processor acting as a “traffic cop” is not necessary. However, this configuration is designed for flexibility and can be readily changed to meet needs.

Upon completion and testing of the full-scale MBC controller, the prototype interface and control VI was modified. Requiring different command output to the HC11 controller is the major difference between the prototype and production MBC interface. Instead of sending 18 bytes of ASCII like the prototype, the output now sent six bytes of ASCII – four bytes for velocity and two bytes for direction. With the exception of the different output to the HC11, the interface program runs identically to the prototype. Figure 22 shows the communications and control layout for testing of one MBC.

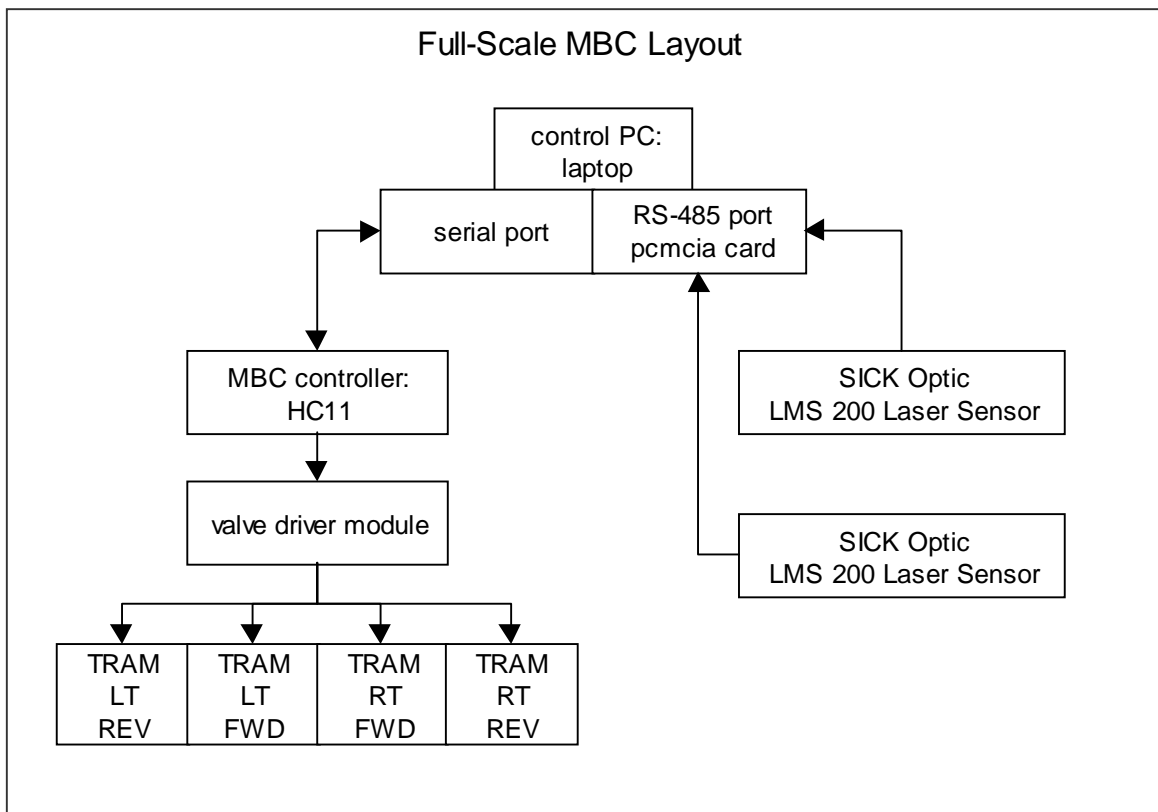


Figure 22. Full-Scale Communications and Control Layout

As with the prototype, adding a second MBC is relatively straightforward. With one MBC and two LMS units, a PCMCIA RS-485 card and the serial port is used.

Adding a second MBC requires using the another PCMCIA RS-485 card and the parallel port with a parallel-to-serial adapter. Additional modifications to the hardware and software will be necessary if more than two automated MBCs are tested. However, it is expected that such testing will be far enough in the development of this project that production hardware, like the Long-Airdox PLC will be used.

Chapter 5: Results and Conclusions

5.1 Results

A 1/10th-scale prototype continuous haulage system was designed and fabricated to provide a test bed for developing path planning and control algorithms, and testing sensor technologies. With the construction of the prototype units, efforts refocused on developing an electronics system capable of providing low-level motor control and communications with multiple processors and a control PC. As a result of the Authors' experience with developing an interface between the MBC controllers and LabVIEW, an interface with the SICK Optic LMS 200 laser measurement was completed. Though success was only achieved after developing a thorough understanding of how the LMS unit operates. As a result of the LabVIEW interface development, current interface and control programs have been based off of the interfaces developed for the MBC controller and the LMS unit. Development of full-scale CHS hardware and software was required for performing above ground trials at a Long-Airdox facility. Much of the prototype control hierarchy was carried to the full-scale design, but a new low-level driver was required to properly interface with the digital control valves on the full-scale MBC.

Although a total of five MBCs and Pigs have been fabricated, the current status of the prototype hardware and software used in testing has involved configurations using either one or two MBCs. As the path-planning and control algorithms advance, more units in the prototype CHS will be used. Preliminary path-planning and control algorithms were conducted on the first prototype MBC completed. Testing has progressed from fairly crude initial runs with the LMS unit, power supply, laptop and cables duct taped to the MBC. The hallway outside the VA Tech Team office was used to test navigation of the overloaded MBC. However, testing was quickly moved to the main hallway because the increased traction provided by the carpeting and the significantly increased weight of the model from the extra sensors and hardware placed too much stress on the plastic tracks. With tile floors, the main hallways provides a more realistic test media because of increased track slippage similar to what is encountered in a mine. After some successful navigation trials through the hallway, a second MBC

operating in manual mode was added. With the manual MBC leading the way, the autonomous second MBC is currently being tested to develop and refine the path-planning and control algorithms. With successful completion of the initial stages of prototype testing, hardware and software modifications were made in order to recreate these tests on the full-scale models.

Long-Airdox secured two full-scale MBCs for use before being shipped to their customer. Behind their Pulaski facility, portions of a mine were laid out using hay bails and black plastic strung between fence posts. Since space was limited, the mine layout would permit the MBC to travel along the wall and turn in one direction. With completion of the mine walls, replacement of manual valves with the digital valves and power supplied to the MBC, testing commenced. Controlling the MBC in manual mode with the joystick completed verification of correct wiring and driver. With the hardware functioning properly, testing of automatic driving proceeded. During the first few runs, the MBC would navigate the course successfully. However, after a short time of testing, the MBC would behave erratically when turning corners. Since this type of behavior had not been experienced with the prototype, there was concern that the algorithms were not robust enough. To properly assess the situation, the VA Tech Team began troubleshooting the system to identify the possible source of the problem. Some additional indicators were added to the interface and control VI in order to observe the command signals while the MBC was operating. As the MBC traversed the mine layout, the added displays showed that the MBC was not reacting to the appropriate command signals. As the MBC would negotiate a turn, the command VI would increase the outside track speed while decreasing the inside track speed. When the algorithms determined that it was necessary to resume straight-ahead travel, the MBC would not respond and would continue to turn. After repeated observation of this behavior, the MBC controller was switched to manual mode and driven in a manner that would attempt to recreate the odd behavior. Recreating this behavior under manual control seemed to indicate that the MBC hydraulics were not operating correctly. After some more debugging, it became evident that there were problems with the hydraulics system. Further testing was postponed until the system could be debugged and fixed.

With the pause in full-scale trials while Long-Airdox employees worked to fix the hydraulics system, efforts resumed on the multiple unit prototype CHS. Because there were some initial problems with weak power supplies and hasty wiring, some time was spent cleanly wiring up new power supplies and putting power buses on each MBC to reduce local wire lengths. With the wiring completed, testing the model resumed with one manual and one automatic MBC. A second LMS device was added to the automatic MBC. Current testing with the MBC continues to refine the path-planning and control algorithms. The addition of closed-loop feedback on the prototype MBC motors has been raised as a necessity and current developmental efforts are looking at the best way to incorporate this motor feedback.

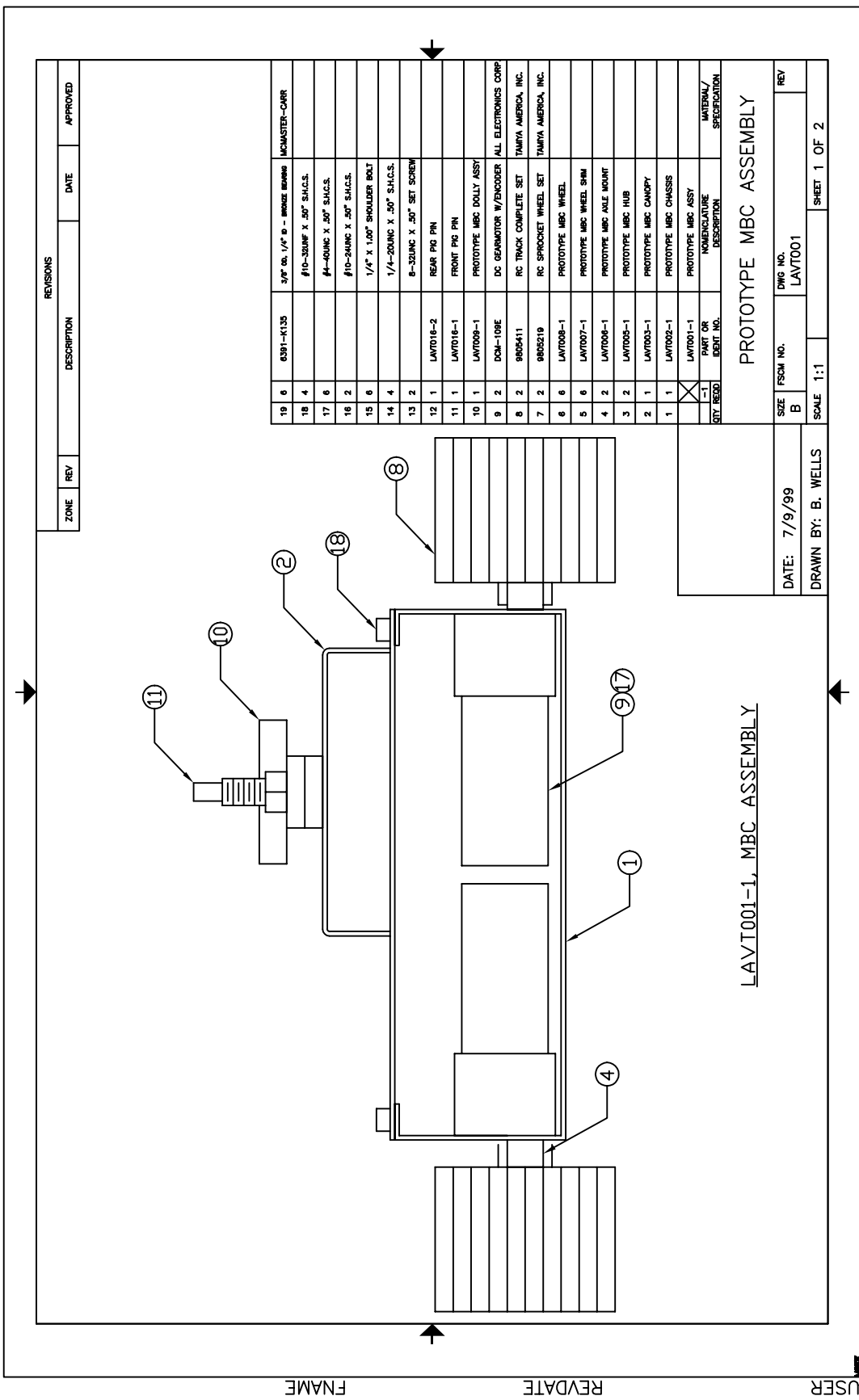
5.2 Conclusions

Although the prototype continuous haulage system seemed to be a long time in the making from the perspective of the author, and probably the other VA Tech Team members, it seems to have met the requirements quite competently. The flexibility and benefits of using LabVIEW for the interface and control program were envisioned; however, not quite to the extent that it has aided the rapid development of this project. Thus far, the hardware has performed effectively, for both the prototype and full-scale models. In fact, the full-scale MBC controller has operated rather robustly in an outdoor environment and has been very reliable.

Being heavily involved with the microcontroller aspect of this project required much review of various electronic products in the marketplace. As a result of this exposure, the use of more powerful microcontrollers or single board computers (SBC) might have been a better solution since the control algorithms are continually growing in size and complexity. This is especially true because the SBCs could effectively serve as a lower cost simulator of the PLC in development by Long-Airdox. However, it is doubtful that a single SBC could be purchased for the price of the combined master and slave controllers, making budget constraints a potential concern. Additionally, all of the current hardware and software should continue to be very functional in its current configuration or with added slave controllers. The motor drivers and Laser-Video

scanners could be used in conjunction with a SBC or more powerful microcontroller, implying that the current electronics system has been an overall safe choice.

Significant efforts are expected to convert the prototype system into a robust system suitable for use in a coal mine. There will be many decisions that will have to be made by Long-Airtox that are hoped to not have a deep impact upon any sensing or control strategies. Only once the path-planning and control algorithms near completion will a final estimate on the computing power and communications needs be truly known. Until then it will continue to be somewhat of a guessing game. By such a time, the Long-Airtox PLC should be closer to completion. Close collaboration with the VA Tech team should help select suitable hardware for the PLC.



ZONE	REV	DESCRIPTION	DATE	APPROVED
18	6	6381-K135	3/8" dia. 1/4" B - BRIDGE BEARING	MOHAMMED-CARR
18	4		#10-32UNC X .50" SH.C.S.	
17	6		#4-40UNC X .50" SH.C.S.	
16	2		#10-24UNC X .50" SH.C.S.	
15	6		1/4" X 1.00" SHOULDER BOLT	
14	4		1/4-20UNC X .50" SH.C.S.	
13	2		5-32UNC X .50" SET SCREW	
12	1	LAVT016-2	REAR PHD PIN	
11	1	LAVT016-1	FRONT PHD PIN	
10	1	LAVT009-1	PROTOTYPE MBC DOLLY ASSY	
9	2	DCM-108E	DC GEARMOTOR W/ENCODER	ALL ELECTRONICS CORP
8	2	9806411	RC TRACK COMPLETE SET	TAMIYA AMERICA, INC.
7	2	9806219	RC SPROCKET WHEEL SET	TAMIYA AMERICA, INC.
6	6	LAVT008-1	PROTOTYPE MBC WHEEL	
5	6	LAVT007-1	PROTOTYPE MBC WHEEL SPRM	
4	2	LAVT006-1	PROTOTYPE MBC AXLE MOUNT	
3	2	LAVT005-1	PROTOTYPE MBC HUB	
2	1	LAVT003-1	PROTOTYPE MBC CHASSIS	
1	1	LAVT002-1	PROTOTYPE MBC ASSY	
			PART OR NOMENCLATURE DESCRIPTION	
			QTY REQD	
			DEPT NO.	
				MATERIAL SPECIFICATION

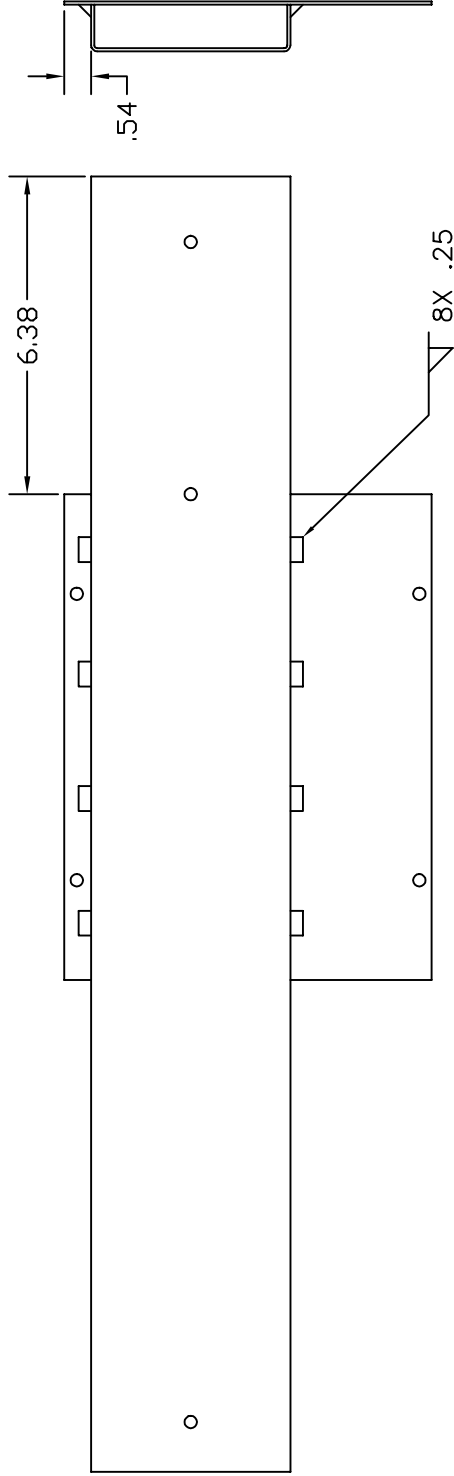
LAVT001-1, MBC ASSEMBLY

PROTOTYPE MBC ASSEMBLY

DATE: 7/9/99	SIZE: B	FSKM NO.: LAVT001	DWG NO.: LAVT001
DRAWN BY: B. WELLS	SCALE: 1:1	SHEET 1 OF 2	

REVISIONS		
ZONE	REV	DESCRIPTION

DATE	APPROVED



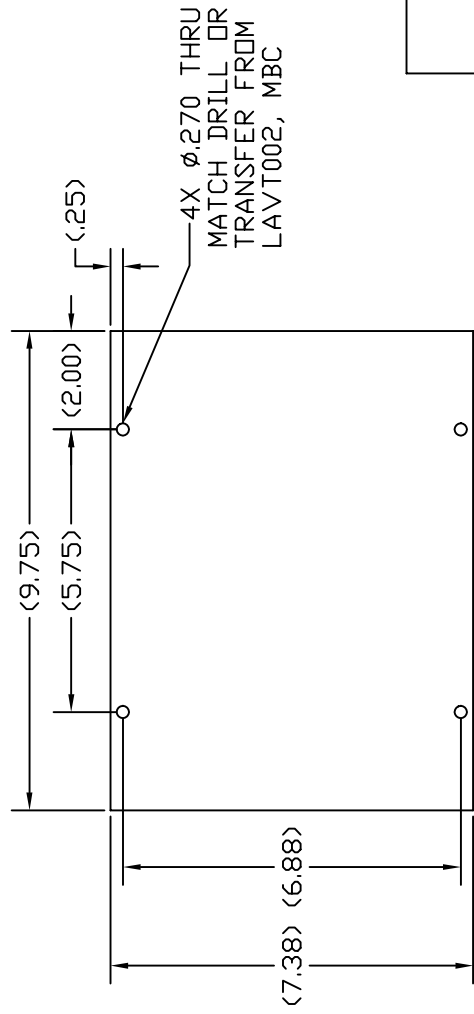
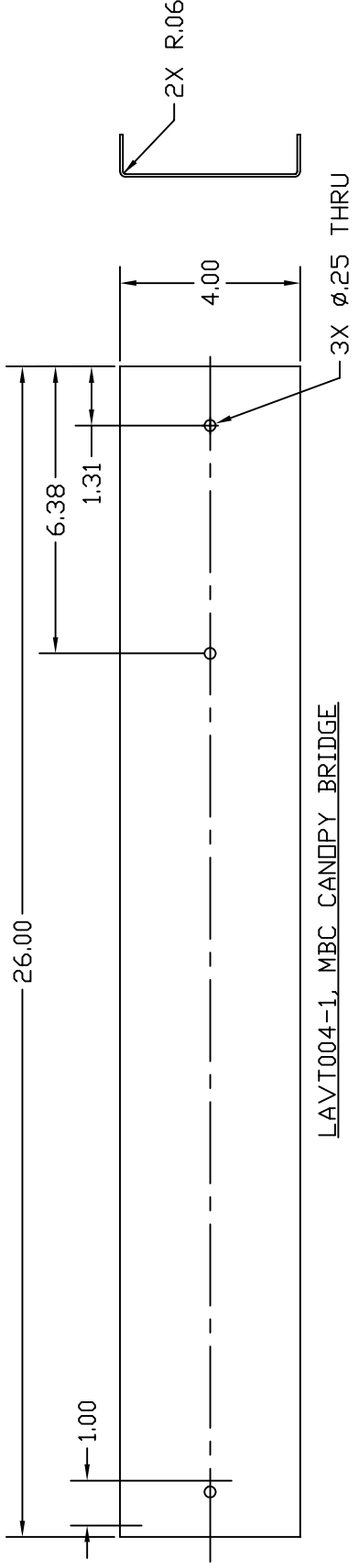
LAVT003-1, MBC CANOPY ASSEMBLY

QTY	REQD	PART OR IDENT. NO.	NOMENCLATURE DESCRIPTION	MATERIAL/SPECIFICATION
1		LAVT004-2	MBC CANOPY DECK	16-GUAGE MILD STEEL
1		LAVT004-1	MBC CANOPY BRIDGE	16-GUAGE MILD STEEL

PROTOTYPE MBC CANOPY ASSY

DATE: 7/9/99	SIZE: B	FSCM NO.:	DWG NO. LAVT003	REV
DRAWN BY: B. WELLS	SCALE: 1:2			SHEET 1 OF 1

REVISIONS			DATE	APPROVED
ZONE	REV	DESCRIPTION		

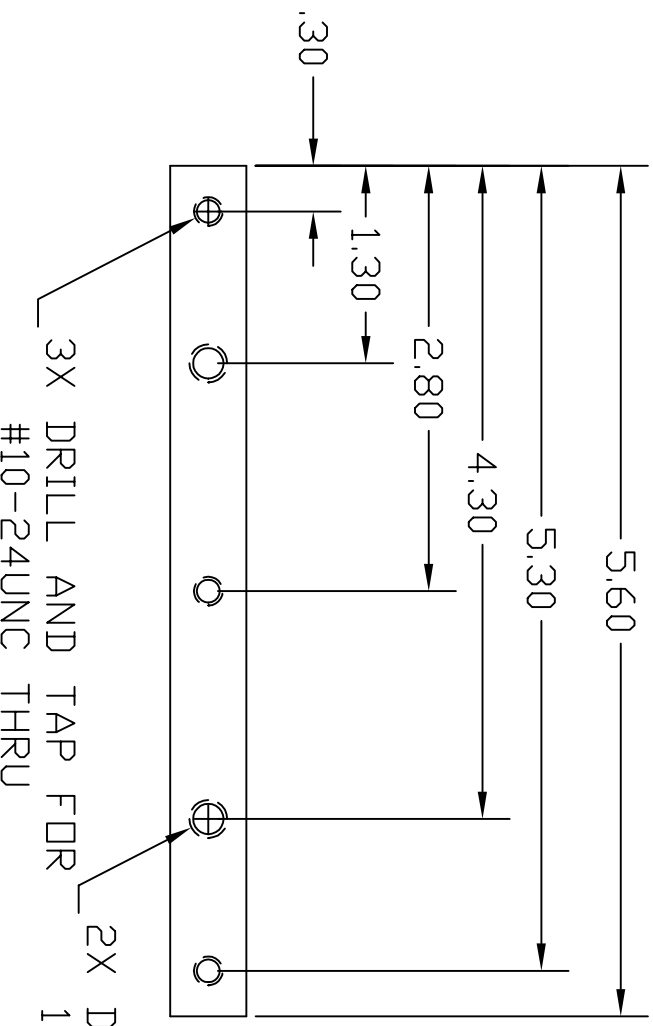


QTY REQD	1	LAVT004-2	MBC CANOPY DECK	16-GUAGE MILD STEEL
		LAVT004-1	MBC CANOPY BRIDGE	16-GUAGE MILD STEEL
		PART OR IDENT. NO.	NOMENCLATURE DESCRIPTION	MATERIAL/SPECIFICATION

PROTOTYPE MBC CANOPY PARTS

DATE:	7/9/99	FSCM NO.	DWG NO.	REV
DRAWN BY:	B. WELLS		LAVT004	
		SCALE	1:2	SHEET 1 OF 1

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED

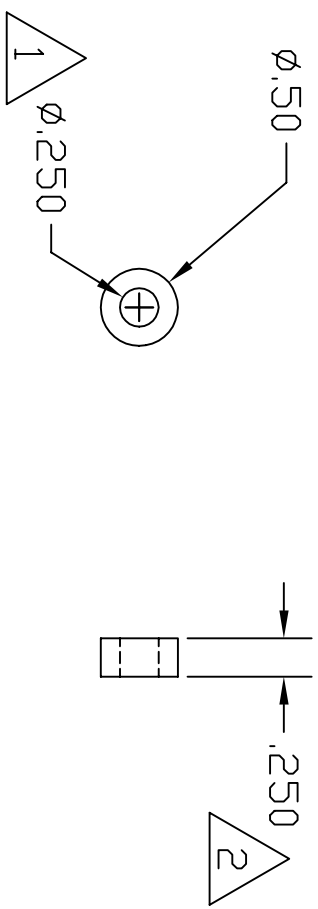


3X DRILL AND TAP FOR #10-24UNC THRU

2X DRILL AND TAP FOR 1/4-20UNC THRU

DATE: 7/9/99		DRAWN BY: B. WELLS	
SIZE A		SCALE 1:1	
FSCM NO.		DWG NO.	
LAVT006-1		LAVT006	
PART OR IDENT NO.		MBC AXLE MOUNT	
QTY REQD		NOMENCLATURE DESCRIPTION	
-1		ALUMINUM OR STEEL	
MATERIAL / SPECIFICATION			
PROTOTYPE MBC AXLE MOUNT			
REV		SHEET 1 OF 1	

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED



1 FOR CLOSE SLIDING FIT WITH (.250") DIA. SHOULDER BOLT

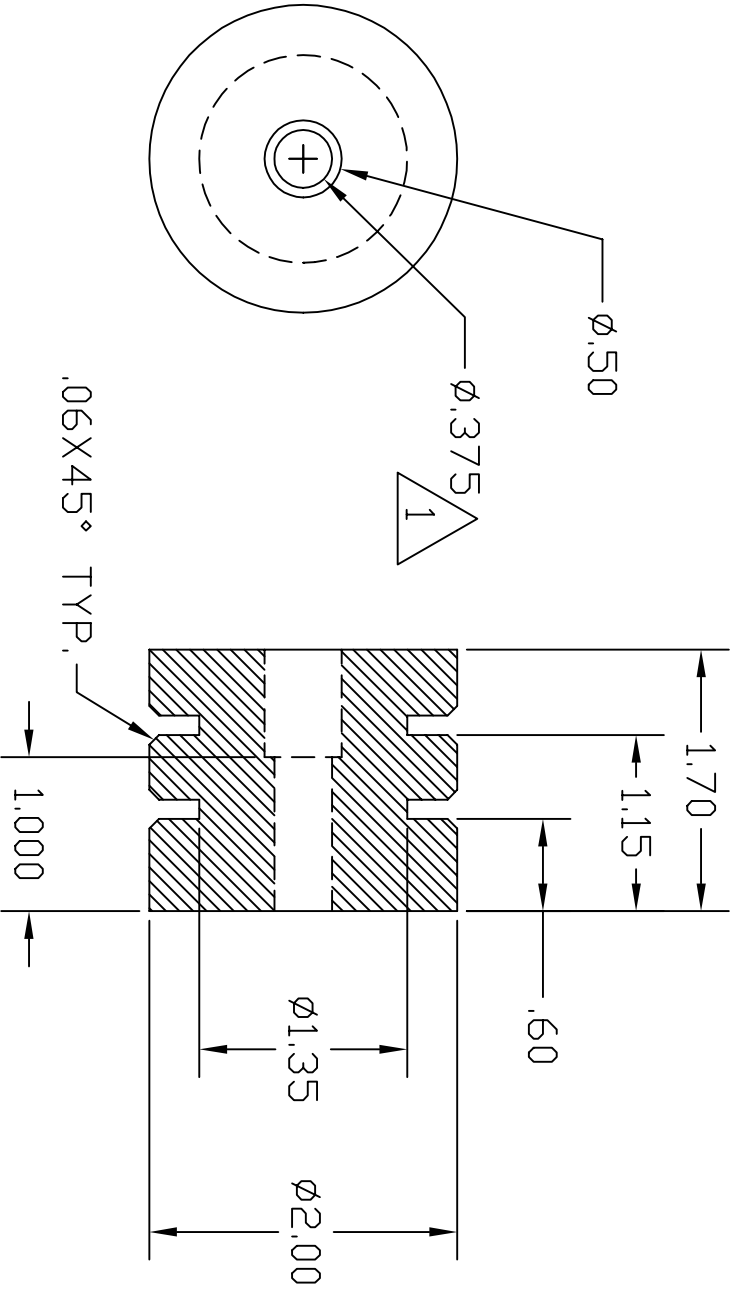
2 FILE AS NECESSARY FOR CLEARANCE BETWEEN WHEEL AND AXLE MOUNT

DATE: 7/9/99		SIZE: A		FSCM NO.		DWG NO. LAVT007		REV	
DRAWN BY: B. WELLS		SCALE: 1:1		LAVT007-1		WHEEL SHIM		ALUMINUM OR STEEL	
		-1		PART OR IDENT NO.		NOMENCLATURE DESCRIPTION		MATERIAL / SPECIFICATION	
		QTY REQD				WHEEL SHIM			

WHEEL SHIM

SHEET 1 OF 1

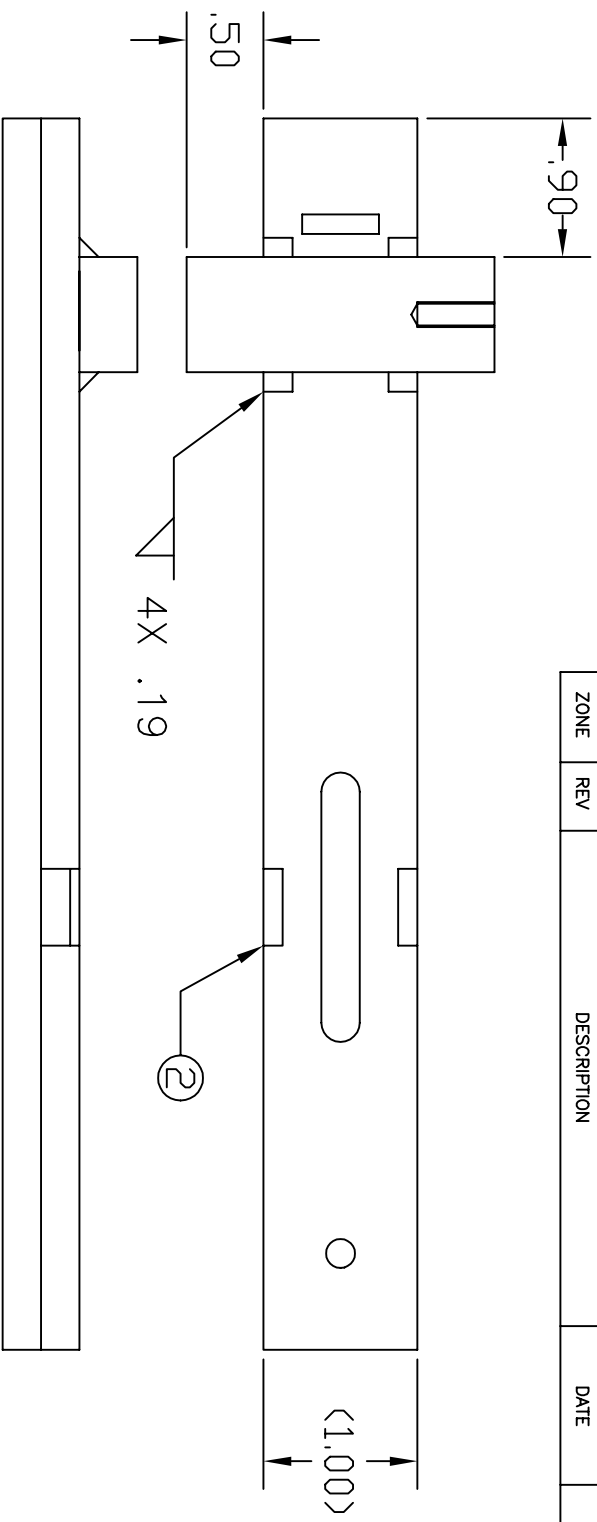
REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED



1 FOR LIGHT PRESS FIT WITH (.375") DIA. BRONZE BEARING

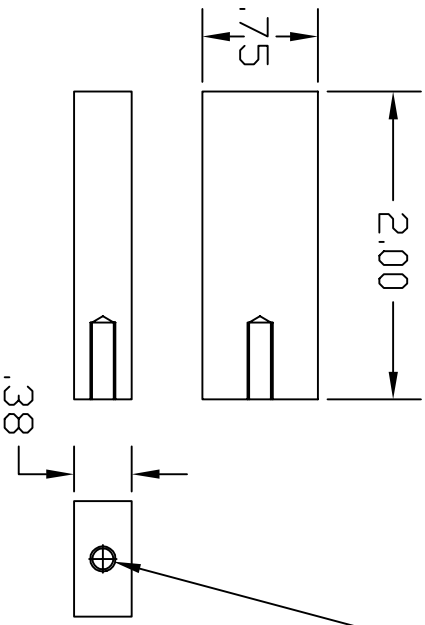
DATE: 7/9/99		SIZE: A	FSCM NO.:	DWG NO.:	REV:
DRAWN BY: B. WELLS		SCALE: 1:1		LAVT008	
LAVT008-1		WHEEL		ALUMINUM OR STEEL	
PART OR IDENT NO.		NOMENCLATURE DESCRIPTION		MATERIAL / SPECIFICATION	
QTY REQD: -1		PROTOTYPE MBC WHEEL			

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED



LAVT009-1, DOLLY ASSY

DRILL AND TAP #8-32UNC .5 DEEP



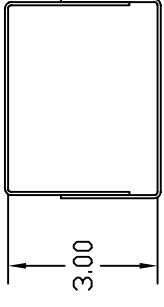
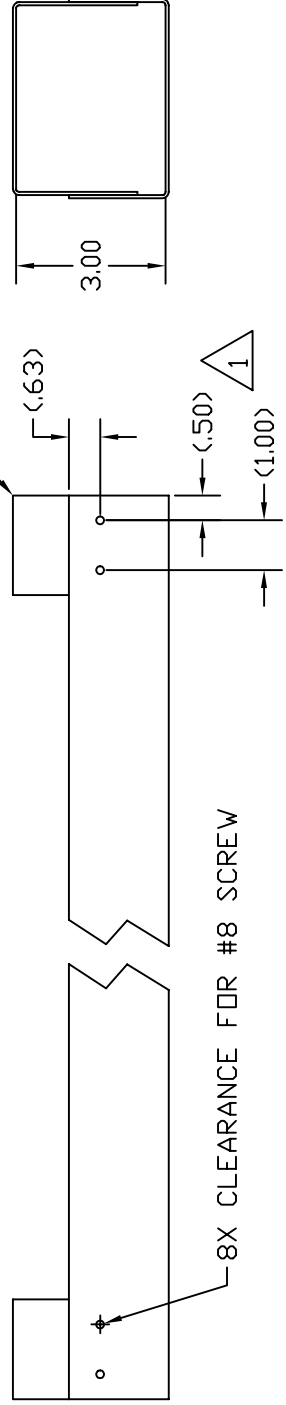
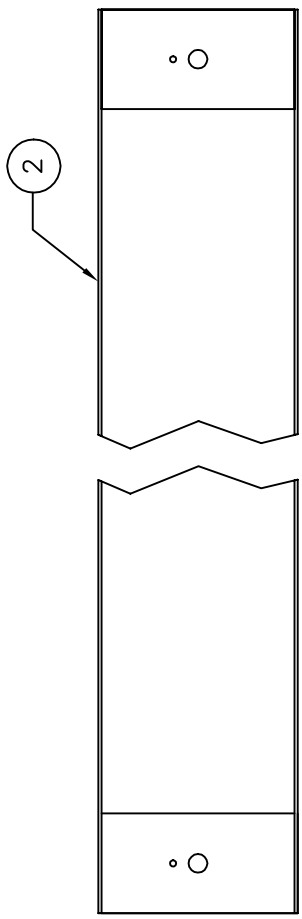
LAVT009-3, DOLLY BLOCK

QTY	REQD	PART OR IDENT NO.	NOMENCLATURE DESCRIPTION	MATERIAL / SPECIFICATION
1		LAVT009-3	DOLLY BLOCK	STEEL STOCK - .375" x .750
1		LAVT009-2	DRAWER SLIDE	MCMASTER-CARR #11965-A181
	X	LAVT009-1	DOLLY ASSEMBLY	
	-1			

PROTOTYPE MBC DOLLY ASSY

DATE: 7/9/99	SIZE: A	FSCM NO.	DWG NO. LAVT009	REV
DRAWN BY: B. WELLS	SCALE: 1:1			

REVISIONS		
ZONE	REV	DESCRIPTION
		DATE
		APPROVED



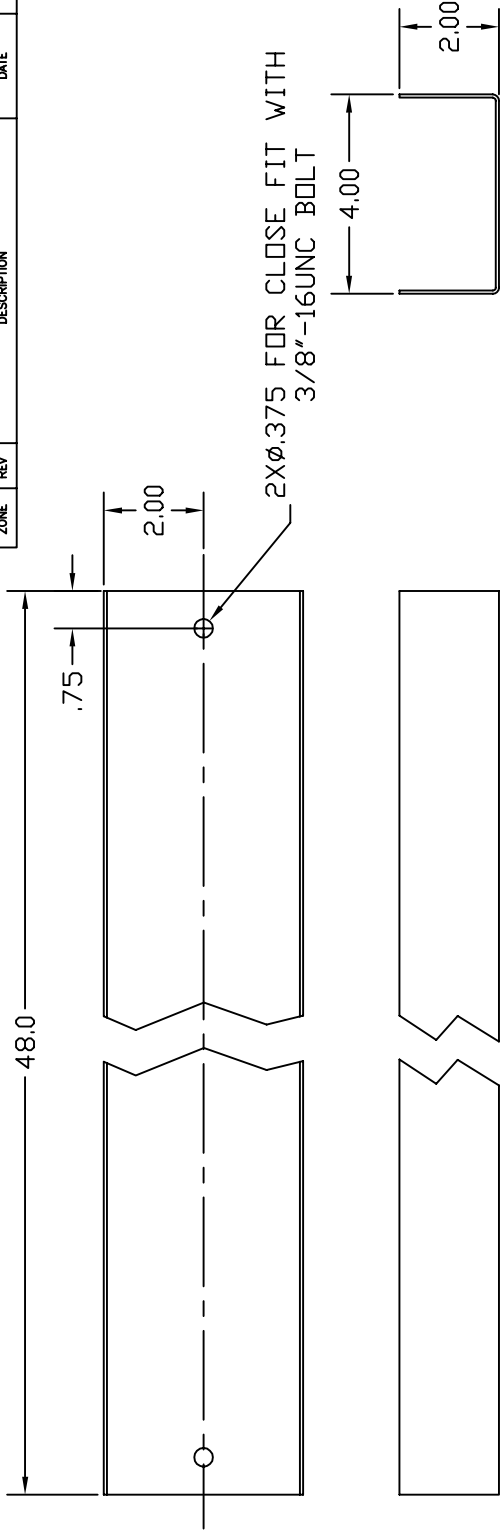
LAVT010-1, Pig Assembly

△ 1 DIMENSIONS INDICATED THUS TO BE MATCH DRILLED ON ASSY WITH LAVT010-3

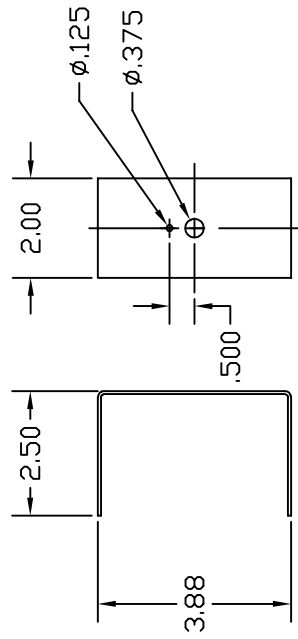
4	#8-32UNC x .50 Long screw	16-GAUGE MILD STEEL
2	LAVT010-3 POTENTIOMETER MOUNT	16-GAUGE MILD STEEL
1	LAVT010-2 PIG	16-GAUGE MILD STEEL
1	LAVT010-1 PIG ASSEMBLY	16-GAUGE MILD STEEL
1	PART OR IDENT. NO.	MATERIAL/ SPECIFICATION
1	QTY. REQD.	

PROTOTYPE PIG ASSEMBLY		
SIZE	FSCM NO.	DWG NO.
B		LAVT010
DATE:	7/9/99	REV
DRAWN BY:	B. WELLS	SHEET 1 OF 2
SCALE:	1:2	

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED



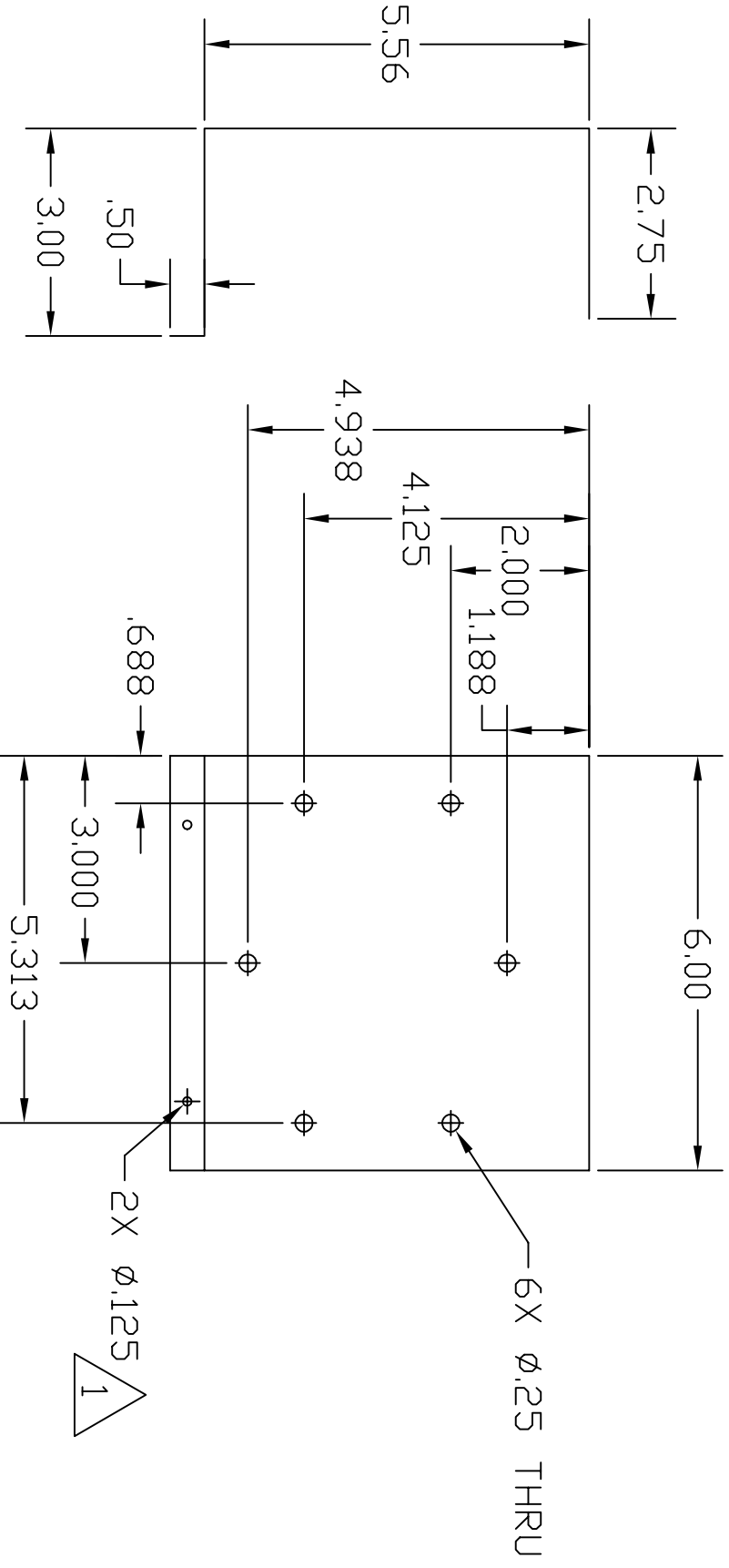
LAVT010-2, PIG



LAVT010-3, POTENTIOMETER MOUNT

<input checked="" type="checkbox"/>	LAVT010-1	PIG ASSEMBLY	16-GAUGE MILD STEEL
<input type="checkbox"/>	PART OR IDENT. NO.	NOMENCLATURE DESCRIPTION	MATERIAL SPECIFICATION
<input type="checkbox"/>	QTY. REQD.		
PROTOTYPE PIG ASSEMBLY			
DATE: 7/9/99	FSCM NO. B	DWG NO. LAVT010	REV
DRAWN BY: B. WELLS	SCALE 1:2		SHEET 1 OF 2

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED

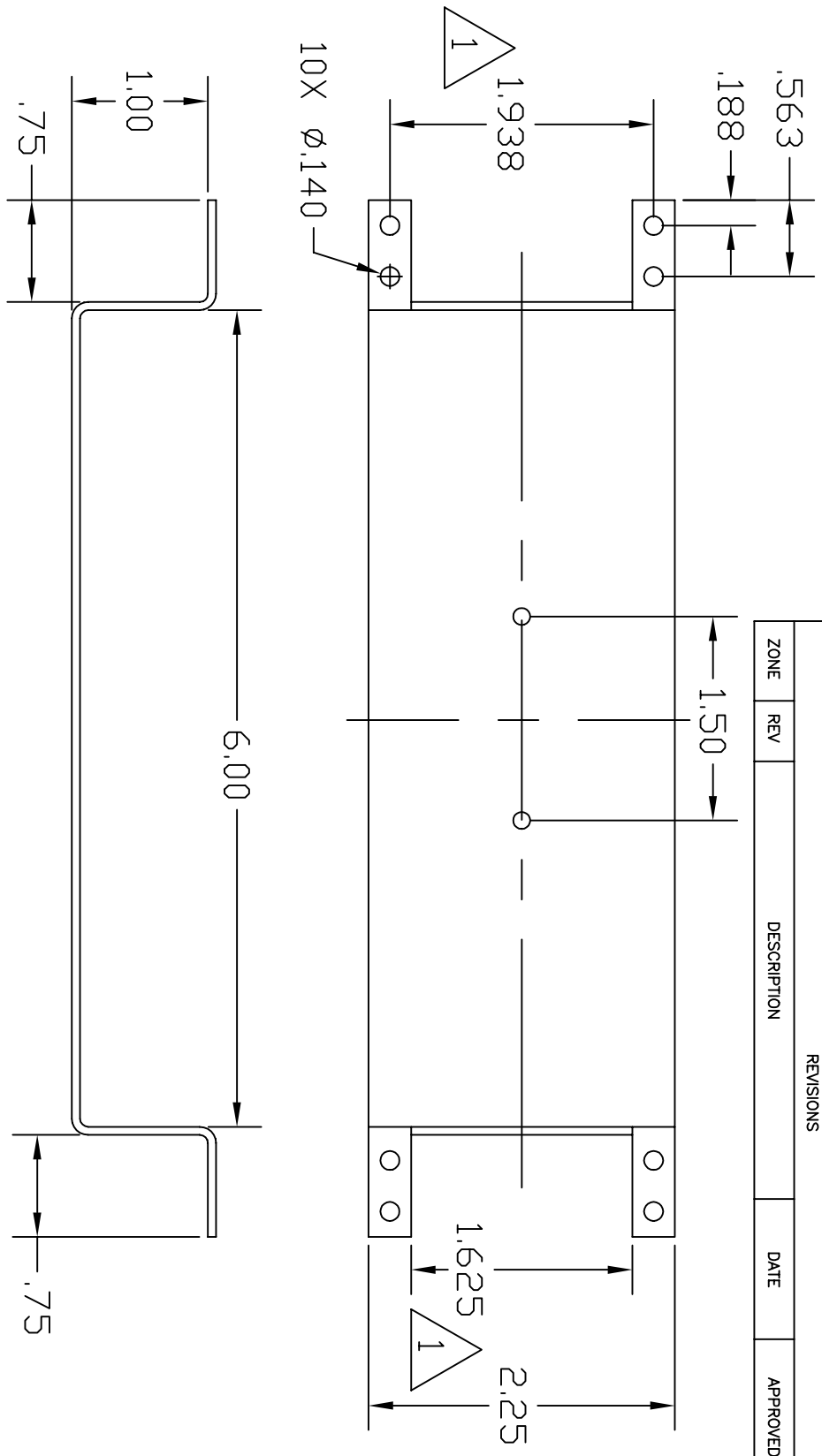


1 MATCH DRILL WITH MBC
CANDPY, LAVT003-1

DATE: 7/9/99		SIZE: A		FSCM NO.:		DWG NO. LAVT014		REV	
DRAWN BY: B. WELLS		SCALE: 1:2		SICK OPTIC MOUNT		NOMENCLATURE DESCRIPTION		16-GUAGE MILD STEEL	
		-1		LAVT014-1		PART OR IDENT NO.		MATERIAL / SPECIFICATION	
		QTY RECD							
PROTOTYPE MBC SICK OPTIC MOUNT									

MARK

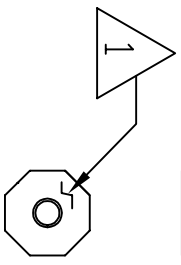
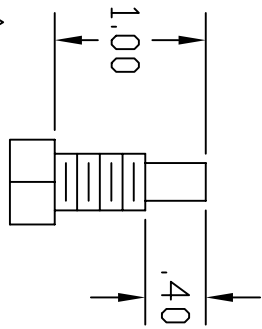
REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED



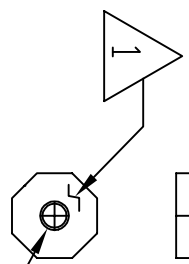
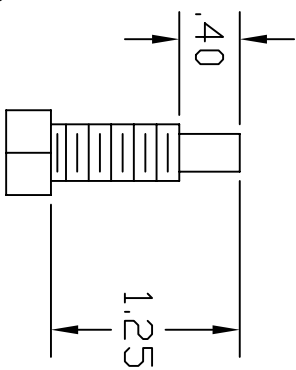
1
 DIM. MUST PROVIDE
 CLOSE SLIDING FIT
 WITH SERVO MOTOR

DATE: 7/9/99		SIZE: A		FSCM NO.:		DWG NO. LAVT015		REV:	
DRAWN BY: B. WELLS		SCALE: 1:1		LAVT015-1		LASER-VIDEO SCANNER MOUNT		16-GUAGE MILD STEEL	
		QTY REQD: -1		PART OR IDENT NO.		NOMENCLATURE DESCRIPTION		MATERIAL/SPECIFICATION	
PROTOTYPE MBC LASER-VIDEO SCANNER MOUNT									

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED



LAVT016-1, FRONT PIG PIN



DRILL AND TAP FOR #10-32UNF .50 DEEP

LAVT016-2, REAR PIG PIN

1 MACHINE BOLT HEAD FLAT

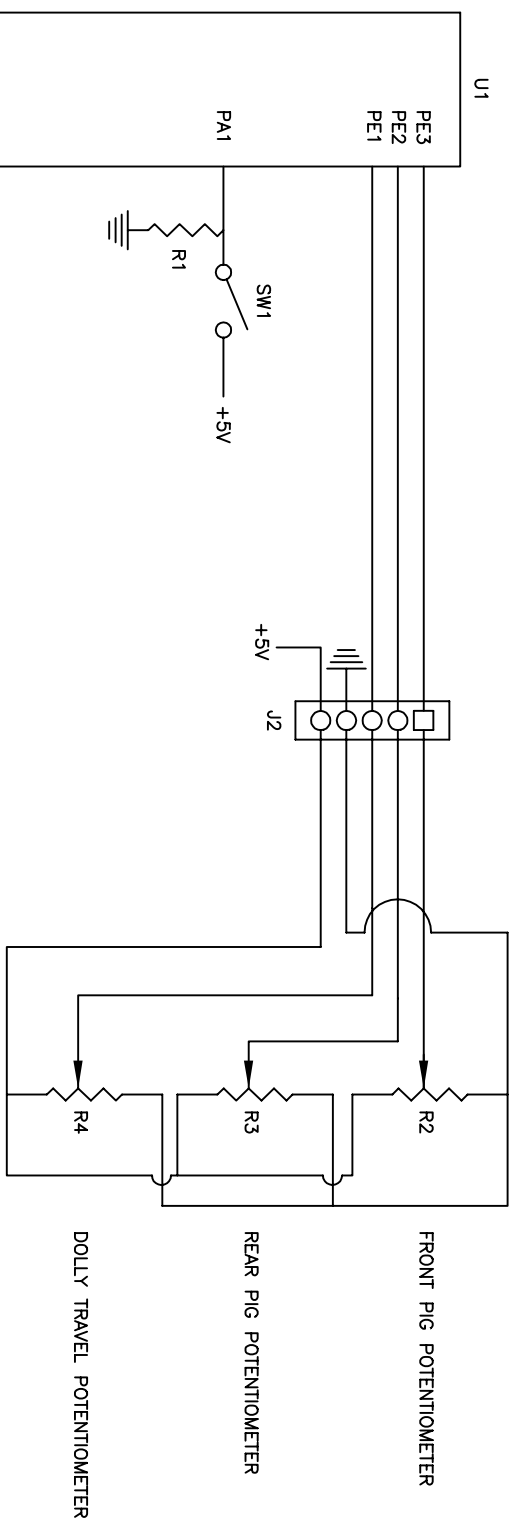
QTY	RECD	PART OR IDENT NO.	NOMENCLATURE DESCRIPTION	MATERIAL / SPECIFICATION
1		LAVT016-2	REAR PIG PIN	3/8"-16UNC X 1.25" BOLT
1		LAVT016-1	FRONT PIG PIN	3/8"-16UNC X 1.00" BOLT
-1				

PIG PIN

DATE: 7/9/99	SIZE: A	FSCM NO.	DWG NO. LAVT016	REV
DRAWN BY: B. WELLS	SCALE: 1:1		SHEET 1 OF 1	

Appendix B: Prototype CHS Electronic System

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED

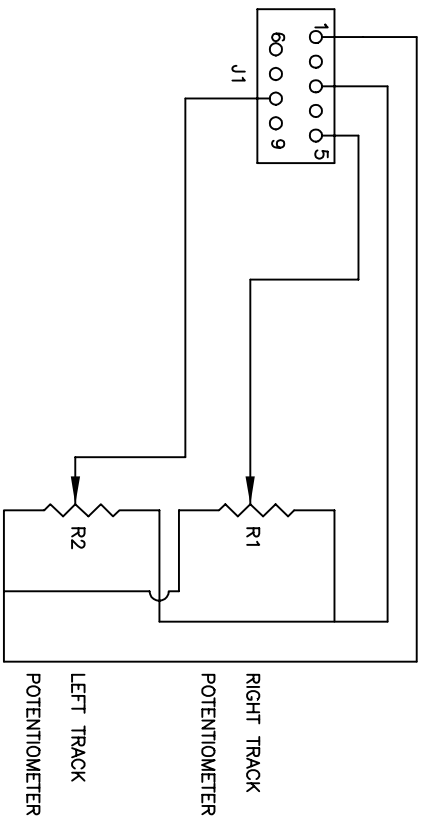


1	SW1	SPST SWITCH		
3	R2,R3,R4	5K POTENTIOMETER	DIGIKEY#	
1	R1	1K PULL-DOWN RESISTOR		
2	J2	.200" VERT. TERMINAL BLOCKS		
1	J1	10-PIN .100" ST. MALE HEADER		
1	U1	MCU BOARD - MC68HC11EVB	MOTOROLA	
-1				
QTY	REQD	PART OR IDENT NO.	NOMENCLATURE DESCRIPTION	MATERIAL / SPECIFICATION

PROTOTYPE MASTER CONTROLLER SCHEMATIC

DATE:	7/9/99	SIZE	FSCM NO.	DWG NO.	LAVT011	REV
DRAWN BY:	B. WELLS	SCALE	1:1	SHEET	1 OF 1	

REVISIONS				
ZONE	REV	DESCRIPTION	DATE	APPROVED



QTY	READ	PART OR IDENT NO.	NOMENCLATURE DESCRIPTION	MATERIAL / SPECIFICATION
2		R1,R2	5K SLIDE POTENTIOMETERS	
1		J1	DB-9 CONN. - MALE	
-1				

PROTOTYPE SLAVE CONTROLLER JOYSTICK

DATE: 7/9/99	SIZE: A	FSCM NO.	DWG NO. LAVT013	REV
DRAWN BY: B. WELLS	SCALE: 1:1		SHEET 1 OF 1	

MARK

Appendix C: Controller Software

B.1 modtest1.asm

*Bruce Wells

*4/12/99

*This program is for use with 1 SICK optic laser scanner and a laptop -
*SICK hooked to RS232 com port and this HC11 controller to parallel
*port via parallel/serial converter. Program loaded into MBC slave
*controller board

*MBC2 will have 2 modes, manual and automatic determined by polarity of
*PA1

*manual mode - read slide pots and calculate speed and direction

*according to DUTY routine.

*automatic - poll SCI data register for data from master. Once

*received, stop interrupts store new values to OC4/5 ram variables and

*then resume interrupts.

```
JTOC4    EQU    $00D6          ;interrupt vectors for OC4 and OC5
JTOC5    EQU    $00D3

RAM      EQU    $01D0

        ORG    $B600
        LDS    #01CF
        LDX    #REGS

SETUP

        BSET   PACTL,X,%10001000 ;PORTA PIN 7 CONFIGURED AS OUTPUT
        LDAA  #$FF
        STAA  DDRC,X           ;PORTC as OUTPUT
        LDAA  #$7E
        STAA  JTOC4            ;setup OC4 pseudovector
        STAA  JTOC5            ;setup OC5 pseudovector
        LDD   #RTOC4
        STD   JTOC4+1
        LDD   #RTOC5
        STD   JTOC5+1
        JSR   SCI_INIT
        JSR   INITAD
        JSR   INITOC
        LDD   #$FF
        LDY   #$FFFF
        STD   OC4HI            ;both outputs initially off..
        STY   OC4LO
        STD   OC5HI
        STY   OC5LO

OPER_MODE
        BRSET PORTA,X,$01,MANUAL ;CHECK PA0, If SET MANUAL MODE
        BRA   AUTO             ;OTHERWISE OPERATE IN AUTOMATIC

MODE

MANUAL
```

```

        LDAA    ADR4,X                ;LEFT SIDE TRACK-PE3
        LDAB    #$80
        JSR     DUTY
        STD     OC4HI
        STY     OC4LO
        LDAA    ADR3,X                ;RIGHT SIDE TRACK-PE2
        LDAB    #70
        JSR     DUTY
        STD     OC5HI
        STY     OC5LO
        BRA     MANUAL

AUTO
AUTO1   JSR     SCI_REC
        JSR     PWM_CHNG
        JSR     STOR_PWM
        BRA     AUTO1

*****
*SCI_INIT - Initialize the Serial Communications Interface
*****
SCI_INIT
        LDAA    #$30
        STAA    BAUD,X              ;BAUD REGISTER
        LDAA    #$00
        STAA    SCCR1,X            ;SCCR1 SCI CONTROL REG 1 SET UP
        LDAA    #$0C
        STAA    SCCR2,X            ;SCCR2 SCI CONTROL REG 2 SET UP
        LDAA    SCSR,X              ;PURGE RECEIVE FLAGS
        LDAA    SCDR,X              ;AND RECEIVE DATA
        RTS

*****
*SUBROUTINE INITAD
*
*INITIALIZES A/D SYSTEM
*****
INITAD
        BSET    OPTION,X,%10000000
        BCLR    OPTION,X,%01000000
        BSET    ADCTL,X,%00110000
        BCLR    ADCTL,X,%00001111
        RTS

*****
*Subroutine INITOC
*
*Initializes timer output OC4 & OC5 for PWM output,
*interrupt driven
*****
INITOC
        LDD     TCNT,X              ;START PWM GENERATION @TCNT
        STD     TOC4,X
        ADDD    #$64                ;START PWM GENERATION @TCNT+100 CLOCK

TICKS
        STD     TI4OC5,X

```

```

        LDAA    #%00001111        ;OM5:OL5=OM4:OL4=1:1 TO SET OC4&5 HIGH
FIRST TIME
        STAA    TCTL1,X
        LDAA    #%00011000        ;CLEAR I4/O5F & OC4F IF SET
        STAA    TFLG1,X
        STAA    TMSK1,X          ;SET OC5I & OC4I TO ENABLE INTERUPT
        CLI
        RTS
*****
*SCI_REC - RECEIVE SERIAL DATA(18 BYTES)
*****
SCI_REC
        LDX     #REGS
        LDAB    #$00
        LDY     #SCIDAT
NOCHNG  BRCLR   SCSR,X,$20,NOCHNG ;Test for RDRF receive character in
        LDAA    SCDR,X          ;SCSR register, 0 no new character.
        STAA    0,Y
        INY
        INCB
        CMPB    #$12
        BNE     NOCHNG
        RTS
*****
*PWM_CHNG - Convert SCI data from ASCII into 8 bit hex values
*           and stores the new values into PWMDAT
*****
PWM_CHNG
        PSHX
        LDX     #SCIDAT
        LDY     #PWMDAT
        LDAA    #$00
        STAA    COUNT
PCHNGLP LDAA    0,X
        INX
        LDAB    0,X
        JSR     TO_HEX
        STAA    0,Y
        INX
        INY
        INC     COUNT
        LDAA    COUNT
        CMPA    #$08
        BNE     PCHNGLP
        LDAA    0,X          ;store LT & RT directions without converting
        STAA    LTDIR      ;from ASCII - should get $30(0) & $46(F)
        INX
        LDAA    0,X
        STAA    RTDIR
        PULX
        RTS
*****
*Converts 2 ASCII characters to 1 hex byte returned in ACCA
*requires the ASCII chars representing the hi and low bits
*to be in ACCA and ACCB, respectively...
*****
TO_HEX

```

```

HIBIT   CMPA   #$39
        BLE   HIBIT1
        SUBA   #$07
HIBIT1  LSLA
        LSLA
        LSLA
        LSLA
        ANDA   #$F0   ;mask lower half
LOWBIT  CMPB   #$39
        BLE   LOWBIT1
        SUBB   #$07
LOWBIT1 ANDB   #$0F   ;mask upper half
        ABA
        RTS
*****
*STOR_PWM - stop interrupts, load SCIDAT and store to
*          OC4HI,OC4LO,OC5HI,OC5LO.....
*****
STOR_PWM
        SEI           ;HALT INTERUPTS
        LDAB          #$00
        LDX           #OC4HI
        LDY           #PWMDAT
STOR_LP  LDAA          0,Y
        STAA          0,X
        INY
        INX
        INCB
        CMPB          #$08           ;#$0A
        BNE          STOR_LP
        LDX          #REGS
        LDAA          LTDIR           ;check for motor direction.....
        CMPA          #$30           ;$30 = FWD, $46 = REV
        BNE          LTREVD
        BSET          PORTA,X,%10000000
        BRA          NEXTD1
LTREVD  BCLR          PORTA,X,%10000000
NEXTD1  LDAA          RTDIR
        CMPA          #$30
        BNE          RTREVD
        BSET          PORTA,X,%01000000
        BRA          DIREND
RTREVD  BCLR          PORTA,X,%01000000
DIREND  CLI           ;RESUME INTERUPTS
        RTS
*****
*SUBROUTINE DUTY
*
*Calculates the duty cycles for OC4 & OC5 and the directions of the
motors
*****
DUTY
        CMPA          #$96           ;COMPARE TO VALUE OF 150
        BHS          REV
        CMPA          #$64           ;COMPARE TO VALUE OF 100
        BLS          FWD

```

```

LDD    #$FF
LDY    #$FFFF
RTS

REV    CMPB    #$80
      BEQ     LTREV
      BRA     RTREV
LTREV  BCLR    PORTA,X,%10000000 ;THIS SETS REV DIRECTION FOR H-BRIDGE
      BRA     NEXT1
RTREV  BCLR    PORTA,X,%01000000 ;THIS SETS REV DIRECTION FOR H-BRIDGE
NEXT1  CMPA    #$F0
      BHS     DC80
      CMPA    #$E1
      BHS     DC70
      CMPA    #$D2
      BHS     DC60
      CMPA    #$C3
      BHS     DC40
      BRA     DC30

FWD    CMPB    #$80
      BEQ     LTFWD
      BRA     RTFWD
LTFWD  BSET    PORTA,X,%10000000 ;THIS SETS FWD DIRECTION FOR H-BRIDGE
      BRA     NEXT2
RTFWD  BSET    PORTA,X,%01000000 ;THIS SETS FWD DIRECTION FOR H-BRIDGE
NEXT2  CMPA    #$0E
      BLS     DC80
      CMPA    #$1C
      BLS     DC70
      CMPA    #$2A
      BLS     DC60
      CMPA    #$38
      BLS     DC50
      CMPA    #$46
      BLS     DC40
      BRA     DC30

DC80   LDD    #$640
      LDY    #$190
      RTS

DC70   LDD    #$578
      LDY    #$258
      RTS

DC60   LDD    #$4B0
      LDY    #$320
      RTS

DC50   LDD    #$3E8
      LDY    #$3E8
      RTS

DC40   LDD    #$320
      LDY    #$4B0
      RTS

DC30   LDD    #$258
      LDY    #$578
      RTS

```

```

*****
*SUBROUTINES RTOC4 & RTOC5
*
*DRIVES OC4 & OC5 OUTPUT FOR PWM BY SCHEDULING TIME DELAY FOR
*NEXT EDGE. ALSO CONFIGURES NEXT EDGE OPPOSITE TO THAT OF CURRENT
*EDGE. WILL NOT WORK PROPERLY WITH DUTY CYCLES CLOSE TO 0
*OR 100%. ADAPTED FROM SPASOV.....
*
*EXECUTED AFTER TOC4=TCNT AND TOC5=TCNT+100 OCCURS
*****

RTOC4    LDX    #REGS
         BRCLR  TCTL1,X,$04,GETOC4LO ;CHECK IF OL4 IS HIGH OR LOW
         LDD    OC4HI
         BRA    NEWTOC4
GETOC4LO LDD    OC4LO
NEWTOC4  ADDD   TOC4,X
         STD    TOC4,X
         LDAA  TCTL1,X                ;INVERT OL4 TO TOGGLE NEXT
         EORA  #%00000100            ;OC4 EDGE BY UPDATING CONTROL REG
         STAA  TCTL1,X
         BCLR  TFLG1,X,%11101111    ;CLEAR FLAG OC4F
         RTI
*****
RTOC5    LDX    #REGS
         BRCLR  TCTL1,X,$01,GETOC5LO
         LDD    OC5HI
         BRA    NEWTOC5
GETOC5LO LDD    OC5LO
NEWTOC5  ADDD   TI4OC5,X                ;CHECK IF OL5 IS HIGH OR LOW
         STD    TI4OC5,X
         LDAA  TCTL1,X                ;INVERT OL5 TO TOGGLE NEXT
         EORA  #%00000001            ;OC5 EDGE BY UPDATING CONTROL REG
         STAA  TCTL1,X
         BCLR  TFLG1,X,%11110111    ;CLEAR FLAG OC5F
         RTI

$INCLUDE 'HC11REG.H'

*****
* RAM data area
*****
         ORG    RAM
SCIDAT  RMB    18
PWMDAT  RMB    8
LTDIR   RMB    1
RTDIR   RMB    1
COUNT  RMB    1
OC4HI   RMB    2
OC4LO   RMB    2
OC5HI   RMB    2
OC5LO   RMB    2

```

B.2 modmstr2_SICK.asm

*Bruce Wells

*4/27/99

*modmstr2.asm programmed in HC11 MBC master controller, and is used
*with modslav2.asm programmed in HC11 MBC slave controller. Both
*programs are for use with the HC11E9 version. Use with SICK LMS 200
*laser measurement devices.

```
RAM      EQU      $01C1

          ORG      $B600
          LDS      #$01C0

SETUP
          LDX      #REGS
          BSET     PACTL,X,%10001000 ;PORTA PINS 3,7 as OUTPUT
          JSR      SCI_INIT
          JSR      SPI_INIT
          JSR      INITAD

MAIN

OPER_MODE
          BRSET    PORTA,X,$02,MANUAL ;If PA1 SET, then MANUAL MODE
          BRA      AUTO ;otherwise AUTOMATIC MODE

MANUAL   JSR      SCI_REC ;added this to tell hc11 when to go
          CMPA     #$06
          BNE     MANUAL
          JSR      ADREAD ;get A/D values first
          JSR      SEND_AD ;then send to interface program
          BRA      OPER_MODE ;check OPER_MODE

AUTO     JSR      SCI_REC
          CMPA     #$06
          BNE     AUTO
          BSET     PORTA,X,%10000000 ;SS* HIGH-slave not selected
          JSR      ADREAD
          JSR      SEND_AD
          JSR      REC_DATA ;get PWM values from interface program
          JSR      PWM_CHNG ;convert and send PWM values to slave
          JSR      PWM_SEND
          BRA      OPER_MODE ;check OPER_MODE
```

*SCI_INIT - Initialize the Serial Communications Interface

SCI_INIT

```
          LDAA     #$30
          STAA     BAUD,X ;BAUD REGISTER
          LDAA     #$00
          STAA     SCCR1,X ;SCCR1 SCI CONTROL REG 1 SET UP
          LDAA     #$0C
          STAA     SCCR2,X ;SCCR2 SCI CONTROL REG 2 SET UP
          LDAA     SCSR,X ;PURGE RECEIVE FLAGS
          LDAA     SCDR,X ;AND RECEIVE DATA
```

```

        RTS
*****
*SPI_INIT - Initialize the Serial Peripheral Interface
*****
SPI_INIT
        LDAA    #%00111000    ;MOSI,SS*,SCK ARE OUTPUT - OTHERS INPUTS
        STAA    DDRD,X
        LDAA    #%01010111    ;ENABLE SPI AS MASTER, CPOL=0, CPHA=1, E/16
        STAA    SPCR,X
        RTS

*****
*SUBROUTINE INITAD
*
*INITIALIZES A/D SYSTEM
*****
INITAD
        BSET    OPTION,X,%10000000
        BCLR    OPTION,X,%01000000
        BSET    ADCTL,X,%00110000
        BCLR    ADCTL,X,%00001111
        RTS

*****
*SUBROUTINE ADREAD
*
*CHECKS FOR COMPLETED CONVERSION
*****
ADREAD

ADREAD1 BRCLR    ADCTL,X,%10000000,ADREAD1
        LDAA    ADR4,X                      ;PIN PE3
        STAA    PIG_FRT
        LDAA    ADR3,X                      ;PIN PE2
        STAA    PIG_REAR
        LDAA    ADR2,X                      ;PIN PE1
        STAA    DOLLY
        RTS

*****
*SCI_REC - RECEIVE SERIAL DATA
*****
SCI_REC
NOCHNG1 BRCLR    SCSR,X,$20,NOCHNG1 ;Test for RDRF receive character in
        LDAA    SCDR,X                      ;SCSR register, 0 no new character.
        RTS

*****
*SEND_AD - SEND POTENTIOMETER MEASUREMENTS
*****
SEND_AD
        LDAA    PIG_FRT
        JSR    OUTLHLF
        LDAA    PIG_FRT
        JSR    OUTRHLF
        LDAA    PIG_REAR
        JSR    OUTLHLF
        LDAA    PIG_REAR
        JSR    OUTRHLF

```

```

        LDAA  DOLLY
        JSR   OUTLHLF
        LDAA  DOLLY
        JSR   OUTRHLF
        RTS

*****
*REC_DATA - RECEIVE SERIAL DATA(18 BYTES)
*****
REC_DATA
        LDX   #REGS
        LDAB  #$00
        LDY   #SCIDAT
NOCHNG  BRCLR  SCSR,X,$20,NOCHNG ;Test for RDRF receive character in
        LDAA  SCDR,X           ;SCSR register, 0 no new character.
        STAA  0,Y
        INY
        INCB
        CMPB  #$12 ;get 16 bytes for PWM DC + 2 for direction
        BNE   NOCHNG
        RTS

*****
*PWM_CHNG - Convert SCI data from ASCII into 8 bit hex
*           values and stores the new values into PWMDAT
*****
PWM_CHNG
        PSHX
        LDX   #SCIDAT
        LDY   #PWMDAT
        LDAA  #$00
        STAA  COUNT
PCHNGLP LDAA  0,X
        INX
        LDAB  0,X
        JSR   TO_HEX
        STAA  0,Y
        INX
        INY
        INC   COUNT
        LDAA  COUNT
        CMPA  #$08 ;#$08 ADDED 2 BYTES FOR DIRECTION
        BNE   PCHNGLP

        LDAA  0,X ;store LT & RT directions without converting
        STAA  LTDIR ;from ASCII - should get $30(0) & $46(F)
        INX
        LDAA  0,X
        STAA  RTDIR

        PULX
        RTS

```

```

*****
*Converts 2 ASCII characters to 1 hex byte returned in ACCA
*requires the ASCII chars representing the hi and low bits
*to be in ACCA and ACCB, respectively...
*****
TO_HEX

HIBIT   CMPA   #$39
        BLE   HIBIT1
        SUBA   #$07
HIBIT1  LSLA
        LSLA
        LSLA
        LSLA
        ANDA   #$F0   ;mask lower half
LOWBIT  CMPB   #$39
        BLE   LOWBIT1
        SUBB   #$07
LOWBIT1 ANDB   #$0F   ;mask upper half
        ABA
        RTS

*****
*PWM_SEND - Sends PWM values to the slave controller
*****
PWM_SEND
        LDX   #REGS
        BCLR  PORTA,X,$80   ;drive PA7 for slave select
        LDAB  #$00
        LDY   #PWMDAT
PSNDLP  LDAA  0,Y
        JSR   SPI_SEND
        INY
        INCB
        CMPB  #$0A           ;send 10 bytes
        BNE  PSNDLP
        BSET  PORTA,X,$80
        RTS

*****
*SPI_SEND - SEND SPI DATA
*****
SPI_SEND
NAK     BRSET PORTA,X,$01,NAK ;check PA0 for slave status(ready or stop)
        STAA  SPDR,X
POLL    TST   SPSR,X         ;CHECK TO SEE IF ANY FLAGS ARE SET
        BPL  POLL
        RTS

*****
*OUTRHLF(), OUTLHLF(), OUTA()
*Convert A from binary to ASCII and output.
*Contents of A are destroyed..
*****

OUTLHLF LSRA                   ;shift data to right
        LSRA
        LSRA
        LSRA
OUTRHLF ANDA   #$0F           ;mask top half

```

```

        ADDA      #$30                ;convert to ascii
        CMPA      #$39
        BLE       OUTA                ;jump if 0-9
        ADDA      #$07                ;convert to hex A-F
OUTA    JSR       SCI_SEND            ;output character
        RTS
*****
*SCI_SEND - Sends a byte through UART
*****
SCI_SEND
        LDX       #REGS
TBNMT   BRCLR    SCSR,X,$80,TBNMT    ;Loop til xmitter output buffer empty
        STAA      SCDR,X
        RTS

```

```
$INCLUDE 'HC11REG.H'
```

```

*****
* RAM data area
*****

```

```

        ORG       RAM

SCIDAT  RMB      12
PWMDAT  RMB      8
LTDIR   RMB      1
RTDIR   RMB      1
COUNT  RMB      1
PIG_FRT RMB      1
PIG_REAR RMB     1
DOLLY   RMB      1

```

B.3 modmstr2_PMS.asm

```

*Bruce Wells
*6/20/99

```

```

*modmstr2_PMS.asm programmed in HC11 MBC master controller, and is used
*with modslav2.asm programmed in HC11 MBC slave controller. Both
*programs are for use with the HC11E9 version. For use with the Laser-
*Video Scanner developed by Todd Upchurch.

```

```

RAM     EQU      $0100

        ORG      $B600
        LDS      #$00FF

SETUP
        LDX      #REGS
        BSET     PACTL,X,%10000000    ;PORTA PINS 7 as OUTPUT
        BCLR     PACTL,X,%00001000    ;PA3 INPUT
        BSET     PORTA,X,%01100000    ;SS* is HIGH - slave not selected
        JSR      SCI_INIT
        JSR      SPI_INIT
        JSR      INITAD

```

```
OPER_MODE
```

```

        BRSET  PORTA,X,$02,MANUAL      ;If PA1 SET, then MANUAL MODE
        BRA    AUTO                    ;otherwise AUTOMATIC MODE
MANUAL  BRA    MANUAL                  ;MANUAL loop does nothing
AUTO    JSR    SCI_REC                 ;wait until receive the ACK symbol from
        CMPA  #$06                    ;control PC - can be used for address
        BNE   MAIN
        BCLR  PORTA,X,%01100000      ;initiate both scanners
        JSR   ADREAD                  ;get analog measurements

        BSET  PORTA,X,%00100000      ;deselect RTSCANNER
LTW8    BRCLR  PORTA,X,%00001000,LTW8 ;wait for scanner READY,PA3 HIGH
        LDY   #LTSCANNER              ;load LTSCANNER RAM address for indexed
        JSR   SCANNER_REC             ;addressing
        BSET  PORTA,X,%01000000      ;done, deselect LTSCANNER
        BCLR  PORTA,X,%00100000      ;deselect RTSCANNER
RTW8    BRCLR  PORTA,X,%00000100,RTW8 ;wait for scanner READY,PA2 HIGH
        LDY   #RTSCANNER              ;load RTSCANNER ram address for indexed
        JSR   SCANNER_REC             ;addressing
        BSET  PORTA,X,%00100000      ;done, deselect RTSCANNER

        JSR   SEND_DATA ;send data - both scanners & analog
        JSR   REC_DATA  ;get PWM values from interface program
        JSR   PWM_CHNG  ;convert and send PWM values to slave
        JSR   PWM_SEND
        BRA   AUTO                    ;restart loop

```

```

*****
*SCI_INIT - Initialize the Serial Communications Interface
*****

```

```

SCI_INIT

```

```

        LDAA  #$30
        STAA  BAUD,X      ;BAUD REGISTER
        LDAA  #$00
        STAA  SCCR1,X     ;SCCR1 SCI CONTROL REG 1 SET UP
        LDAA  #$0C
        STAA  SCCR2,X     ;SCCR2 SCI CONTROL REG 2 SET UP
        LDAA  SCSR,X      ;PURGE RECEIVE FLAGS
        LDAA  SCDR,X      ;AND RECEIVE DATA
        RTS

```

```

*****
*SPI_INIT - Initialize the Serial Peripheral Interface
*****

```

```

SPI_INIT

```

```

        LDAA  #%00111000 ;MOSI,SS*,SCK ARE OUTPUT - OTHERS INPUTS
        STAA  DDRD,X
        LDAA  #%01010111 ;ENABLE SPI AS MASTER, CPOL=0, CPHA=1, E/16
        STAA  SPCR,X
        RTS

```

```

*****
*SUBROUTINE INITAD
*

```

```

*INITIALIZES A/D SYSTEM

```

```

*****
INITAD

```

```

        BSET  OPTION,X,%10000000
        BCLR  OPTION,X,%01000000
        BSET  ADCTL,X,%00110000

```

```

        BCLR      ADCTL,X,%00001111
        RTS

*****
*SUBROUTINE ADREAD
*
*CHECKS FOR COMPLETED CONVERSION
*****
ADREAD  LDX      #REGS
ADREAD1 BRCLR   ADCTL,X,%10000000,ADREAD
        LDAA     ADR4,X                      ;PIN PE3
        STAA     PIG_FRT
        LDAA     ADR3,X                      ;PIN PE2
        STAA     PIG_REAR
        LDAA     ADR2,X                      ;PIN PE1
        STAA     DOLLY
        RTS

*****
*SCI_REC - RECEIVE SERIAL DATA
*****
SCI_REC
NOCHNG1 BRCLR   SCSR,X,$20,NOCHNG1 ;Test for RDRF receive character in
        LDAA     SCDR,X                      ;SCSR register, 0 no new character.
        RTS

*****
*SEND_AD - SEND POTENTIOMETER MEASUREMENTS IN ASCII
*****
SEND_AD
        LDAA     PIG_FRT
        JSR      OUTLHLF
        LDAA     PIG_FRT
        JSR      OUTRHLF
        LDAA     PIG_REAR
        JSR      OUTLHLF
        LDAA     PIG_REAR
        JSR      OUTRHLF
        LDAA     DOLLY
        JSR      OUTLHLF
        LDAA     DOLLY
        JSR      OUTRHLF
        RTS

*****
*SCANNER_REC - RECEIVE SPI DATA FROM SCANNER - 49 BYTES/SCANNER
*****
SCANNER_REC
        LDX      #REGS
        CLRB

REC_LP
        STAA     SPDR,X
POLL    TST      SPSR,X                      ;CHECK TO SEE IF ANY FLAGS ARE SET
        BPL      POLL
        LDAA     SPDR,X
        STAA     0,Y
        INY
        INCB
        CMPB     #$31                        ;receive 49 bytes per scanner

```

```

        BNE      REC_LP
        RTS

*****
*SEND_DATA - Sends data to control PC through SCI
*****
SEND_DATA
        LDX      #REGS
        LDY      #LTSCANNER      ;PIG_FRT
        CLR      CLR      B
SEND_LP  LDAA     0,Y
TBNMT   BRCLR   SCSR,X,$80,TBNMT ;Loop til xmitter output buffer empty
        STAA     SCDR,X
        INCB
        INY
        CMPB    #!98              ;send 49+49+3 bytes = 101bytes(65h)
        BNE     SEND_LP
        RTS

*****
*REC_DATA - RECEIVE SERIAL DATA(18 BYTES)
*****
REC_DATA
        LDX      #REGS
        LDAB     #$00
        LDY      #SCIDAT
NOCHNG  BRCLR   SCSR,X,$20,NOCHNG ;Test for RDRF receive character in
        LDAA     SCDR,X ;SCSR register, 0 no new character.
        STAA     0,Y
        INY
        INCB
        CMPB    #$12 ;get 16 bytes for PWM DC + 2 for direction
        BNE     NOCHNG
        RTS

*****
*PWM_CHNG - Convert SCI data from ASCII into 8 bit hex
*           values and stores the new values into PWMDAT
*****
PWM_CHNG
        PSHX
        LDX      #SCIDAT
        LDY      #PWMDAT
        LDAA     #$00
        STAA     COUNT
PCHNGLP LDAA     0,X
        INX
        LDAB     0,X
        JSR      TO_HEX
        STAA     0,Y
        INX
        INY
        INC      COUNT
        LDAA     COUNT
        CMPA    #$08 ;#$08 ADDED 2 BYTES FOR DIRECTION
        BNE     PCHNGLP
        LDAA     0,X ;store LT & RT directions without converting
        STAA     LTDIR ;from ASCII - should get $30(0) & $46(F)
        INX
        LDAA     0,X

```

```

        STAA     RTDIR
        PULX
        RTS
*****
*Converts 2 ASCII characters to 1 hex byte returned in ACCA
*requires the ASCII chars representing the hi and low bits
*to be in ACCA and ACCB, respectively...
*****
TO_HEX
HIBIT   CMPA     #$39
        BLE     HIBIT1
        SUBA     #$07
HIBIT1  LSLA
        LSLA
        LSLA
        LSLA
        ANDA     #$F0      ;mask lower half
LOWBIT  CMPB     #$39
        BLE     LOWBIT1
        SUBB     #$07
LOWBIT1 ANDB     #$0F      ;mask upper half
        ABA
        RTS
*****
*PWM_SEND - Sends PWM values to the slave controller
*****
PWM_SEND
        LDX     #REGS
        BCLR    PORTA,X,$80    ;drive PA7 for slave select
        LDAB   #$00
        LDY     #PWMDAT
PSNDLP  LDAA     0,Y
        JSR     SPI_SEND
        INY
        INCB
        CMPB   #$0A           ;send 10 bytes
        BNE    PSNDLP
        BSET   PORTA,X,$80
        RTS
*****
*SPI_SEND - SEND SPI DATA
*****
SPI_SEND
        LDX     #REGS
NAK     BRSET   PORTA,X,$01,NAK ;check PA0 for SS*(ready or stop)
        STAA   SPDR,X
POLL2   TST     SPSR,X           ;CHECK TO SEE IF ANY FLAGS ARE SET
        BPL    POLL2
        RTS
*****
*OUTRHLF(), OUTLHLF(), OUTA()
*Convert A from binary to ASCII and output.
*Contents of A are destroyed..
*****
OUTLHLF LSRA                       ;shift data to right
        LSRA

```

```

        LSRA
        LSRA
OUTRHLF ANDA      #$0F          ;mask top half
        ADDA      #$30          ;convert to ascii
        CMPA      #$39
        BLE       OUTA          ;jump if 0-9
        ADDA      #$07          ;convert to hex A-F
OUTA    JSR       SCI_SEND      ;output character
        RTS
*****
*SCI_SEND - Sends a byte through UART
*****
SCI_SEND
        LDX       #REGS
TBNMT2 BRCLR     SCSR,X,$80,TBNMT2 ;Loop til xmitter output buffer empty
        STAA      SCDR,X
        RTS

$INCLUDE 'HC11REG.H'

*****
* RAM data area
*****
        ORG       RAM

SCIDAT  RMB      12
PWMDAT  RMB      8
LTDIR   RMB      1
RTDIR   RMB      1
COUNT  RMB      1
PIG_FRT RMB      1
PIG_REAR RMB     1
DOLLY   RMB      1
LTSCANNER RMB    !49
RTSCANNER RMB    !49

```

B.4 modslav2.asm

```

*Bruce Wells
*4/25/99

```

```

*modmstr2.asm programmed in HC11 MBC master controller, and is used
*with modslav2.asm programmed in HC11 MBC slave controller. Both
*programs are for *use with the HC11E9 version.
*MODSLAV2 will have 2 modes, manual and automatic determined by
polarity of PA1

```

```

*manual mode - read slide pots and calculate speed and direction
*according to DUTY routine automatic - poll SPI data register for data
*from master. Once received, stop interrupts.

```

```

JTOC4   EQU      $00D6          ;interrupt vectors for OC4 and OC5
JTOC5   EQU      $00D3

```

```

RAM      EQU      $01D0

          ORG      $B600
          LDS      #$01CF
          LDX      #REGS

SETUP

          BSET     PACTL,X,%10001000          ;PORTA Pin 7 as OUTPUT
          LDAA     #$7E
          STAA     JTOC4                      ;setup OC4 pseudovector
          STAA     JTOC5                      ;setup OC5 pseudovector
          LDD      #RTOC4
          STD      JTOC4+1
          LDD      #RTOC5
          STD      JTOC5+1
          JSR      SPI_INIT
          JSR      INITAD
          JSR      INITOC

          LDD      #$FF
          LDY      #$FFFF
          STD      OC4HI                      ;both outputs initially off..
          STY      OC4LO
          STD      OC5HI
          STY      OC5LO

OPER_MODE
          BRSET    PORTA,X,$01,MANUAL          ;If PA0 SET, then MANUAL MODE
          BRA      AUTO                        ;otherwise AUTOMATIC MODE

MANUAL
          LDAA     ADR4,X                      ;left side track-PE3
          LDAB     #$80
          JSR      DUTY
          STD      OC4HI
          STY      OC4LO
          LDAA     ADR3,X                      ;right side track-PE2
          LDAB     #70
          JSR      DUTY
          STD      OC5HI
          STY      OC5LO
          BRA      OPER_MODE

AUTO
          BCLR     PORTA,X,$20 ;PA3 LOW-notify master SPI xfer can begin
AUTO1    BRSET    PORTD,X,%00100000,AUTO1
          JSR      SPI_REC
          JSR      STOR_PWM
          BRA      OPER_MODE

```

```

*****
*SPI_INIT - configures the SPI interface as slave
*****

```

```

SPI_INIT
          LDAA     #%00000100          ;MISO=1, all others INPUT
          STAA     DDRD,X
          LDAA     #%01000111          ;enable SPI as SLAVE, CPOL=0, CPHA=1,
          STAA     SPCR,X
          RTS

```

```

*****
*SUBROUTINE INITAD
*
*INITIALIZES A/D SYSTEM
*****
INITAD
    BSET    OPTION,X,%10000000
    BCLR    OPTION,X,%01000000
    BSET    ADCTL,X,%00110000
    BCLR    ADCTL,X,%00001111
    RTS

*****
*Subroutine INITOC
*
*Initializes timer output OC4 & OC5 for PWM output,
*inteRrupt driven
*****
INITOC
    LDD     TCNT,X          ;START PWM GENERATION @TCNT
    STD     TOC4,X
    ADDD    #$64           ;START PWM GENERATION @TCNT+100 CLOCK TICKS
    STD     TI4OC5,X
    LDAA    #%00001111    ;OM5:OL5=OM4:OL4=1:1 TO SET OC4&5 HIGH
FIRST TIME
    STAA    TCTL1,X
    LDAA    #%00011000    ;CLEAR I4/O5F & OC4F IF SET
    STAA    TFLG1,X
    STAA    TMSK1,X      ;SET OC5I & OC4I TO ENABLE INTERUPT
    CLI
    RTS

*****
*SPI_REC-receives the PWM & direction from the master
*****
SPI_REC LDX     #REGS
        LDAB    #$00
        LDY     #SPIDAT
SPIW8   TST     SPSR,X
        BPL     SPIW8
        LDAA    SPDR,X
        STAA    0,Y
        INY
        INCB
        CMPB    #$0A                ;10 bytes received??
        BNE     SPIW8
        RTS

*****
*STOR_PWM - stop interrupts, load SPIDAT and store to
*
*          OC4HI,OC4LO,OC5HI,OC5LO.....
*****
STOR_PWM
    SEI                ;HALT INTERUPTS
    LDAB    #$00
    LDY     #SPIDAT
    LDX     #OC4HI
STOR_LP LDAA    0,Y
        STAA    0,X
    INY

```

```

        INX
        INCB
        CMPB    #$08            ;#$0a
        BNE     STOR_LP
        LDX     #REGS
        LDAA    LTDIR            ;check for motor direction.....
        CMPA    #$30            ;$30 = FWD, $46 = REV
        BNE     LTREVD
        BSET    PORTA,X,%10000000
        BRA     NEXTD1
LTREVD  BCLR    PORTA,X,%10000000
NEXTD1  LDAA    RTDIR
        CMPA    #$30
        BNE     RTREVD
        BSET    PORTA,X,%01000000
        BRA     DIREND
RTREVD  BCLR    PORTA,X,%01000000
DIREND  CLI     ;RESUME INTERRUPTS
        RTS

*****
*SUBROUTINE DUTY
*
*Calculates duty cycles for OC4 & OC5 and the directions of the motors
*****
DUTY
        CMPA    #$96            ;COMPARE TO VALUE OF 150
        BHS     REV
        CMPA    #$64            ;COMPARE TO VALUE OF 100
        BLS     FWD
        LDD     #$FF
        LDY     #$FFFF
        RTS

REV     CMPB    #$80
        BEQ     LTREV
        BRA     RTREV
LTREV  BCLR    PORTA,X,%10000000    ;THIS SETS REV DIRECTION FOR H-
BRIDGE
        BRA     NEXT1
RTREV  BCLR    PORTA,X,%01000000
NEXT1  CMPA    #$F0
        BHS     DC80
        CMPA    #$E1
        BHS     DC70
        CMPA    #$D2
        BHS     DC60
        CMPA    #$C3
        BHS     DC40
        CMPA    #$B4
        BHS     DC30
        BRA     DC20
FWD    CMPB    #$80
        BEQ     LTFWD
        BRA     RTFWD
LTFWD  BSET    PORTA,X,%10000000
        BRA     NEXT2
RTFWD  BSET    PORTA,X,%01000000

```

```

NEXT2  CMPA    #$0E
        BLS    DC80
        CMPA    #$1C
        BLS    DC70
        CMPA    #$2A
        BLS    DC60
        CMPA    #$38
        BLS    DC50
        CMPA    #$46
        BLS    DC40
        CMPA    #$54
        BLS    DC30
        BRA    DC20
DC80    LDD     #$640
        LDY    #$190
        RTS
DC70    LDD     #$578
        LDY    #$258
        RTS
DC60    LDD     #$4B0
        LDY    #$320
        RTS
DC50    LDD     #$3E8
        LDY    #$3E8
        RTS
DC40    LDD     #$320
        LDY    #$4B0
        RTS
DC30    LDD     #$258
        LDY    #$578
        RTS
DC20    LDD     #$190
        LDY    #$640
        RTS

```

```

*****
*SUBROUTINES RTOC4 & RTOC5
*
*DRIVES OC4 & OC5 OUTPUT FOR PWM BY SCHEDULING TIME DELAY FOR
*NEXT EDGE. ALSO CONFIGURES NEXT EDGE OPPOSITE TO THAT OF CURRENT
*EDGE. WILL NOT WORK PROPERLY WITH DUTY CYCLES CLOSE TO 0
*OR 100%. ADAPTED FROM SPASOV.....
*
*EXECUTED AFTER TOC4=TCNT AND TOC5=TCNT+100 OCCURS
*****
RTOC4   LDX     #REGS
        BSET   PORTA,X,$20 ;PA5 HI to notify master to halt SPI xfers
        BRCLR  TCTL1,X,$04,GETOC4LO ;CHECK STATE OF OL4
        LDD    OC4HI
        BRA    NEWTOC4
GETOC4LO LDD    OC4LO
NEWTOC4  ADDD   TOC4,X
        STD    TOC4,X
        LDAA  TCTL1,X           ;INVERT OL4 TO TOGGLE NEXT
        EORA  #%00000100       ;OC4 EDGE BY UPDATING CONTROL REG
        STAA  TCTL1,X
        BCLR  TFLG1,X,%11101111 ;CLEAR FLAG OC4F

```

```

        BCLR   PORTA,X,$20           ;resume SPI transfers
        RTI
*****
RTOC5   LDX    #REGS
        BSET   PORTA,X,$20 ;PA5 HI to notify master to halt SPI xfers
        BRCLR  TCTL1,X,$01,GETOC5LO ;CHECK STATE OF OL5
        LDD    OC5HI
        BRA    NEWTOC5
GETOC5LO LDD    OC5LO
NEWTOC5  ADDD   TI4OC5,X
        STD    TI4OC5,X
        LDAA   TCTL1,X             ;INVERT OL5 TO TOGGLE NEXT
        EORA   #%00000001         ;OC5 EDGE BY UPDATING CONTROL REG
        STAA   TCTL1,X
        BCLR   TFLG1,X,%11110111 ;CLEAR FLAG OC5F
        BCLR   PORTA,X,$20         ;resume SPI transfers
        RTI

```

\$INCLUDE 'HC11REG.H'

```

*****
* RAM data area
*****

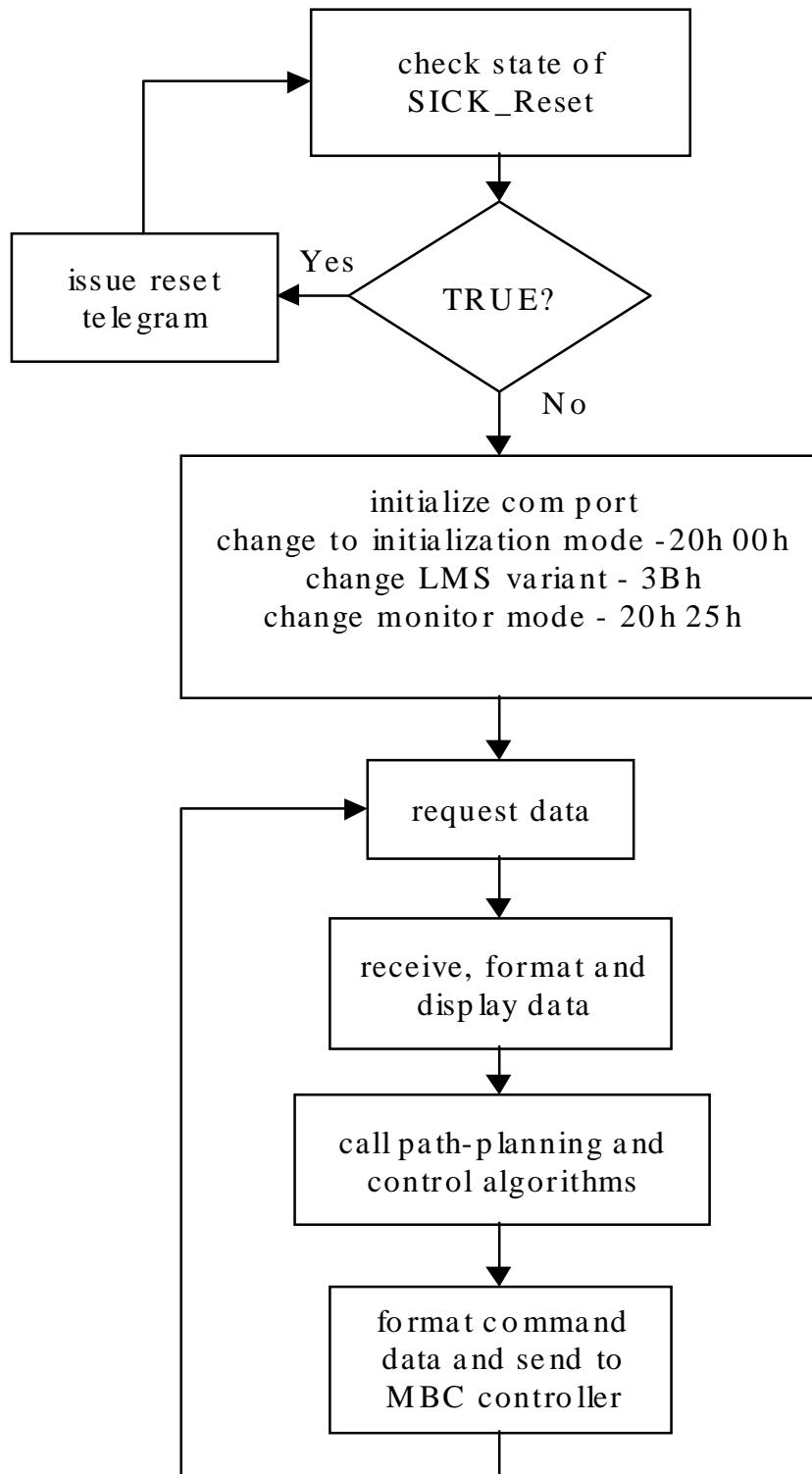
```

```

        ORG    RAM
SPIDAT  RMB    $8
LTDIR   RMB    1
RTDIR   RMB    1
OC4HI   RMB    2
OC4LO   RMB    2
OC5HI   RMB    2
OC5LO   RMB    2

```

Appendix D: LabVIEW Interface and Control Program



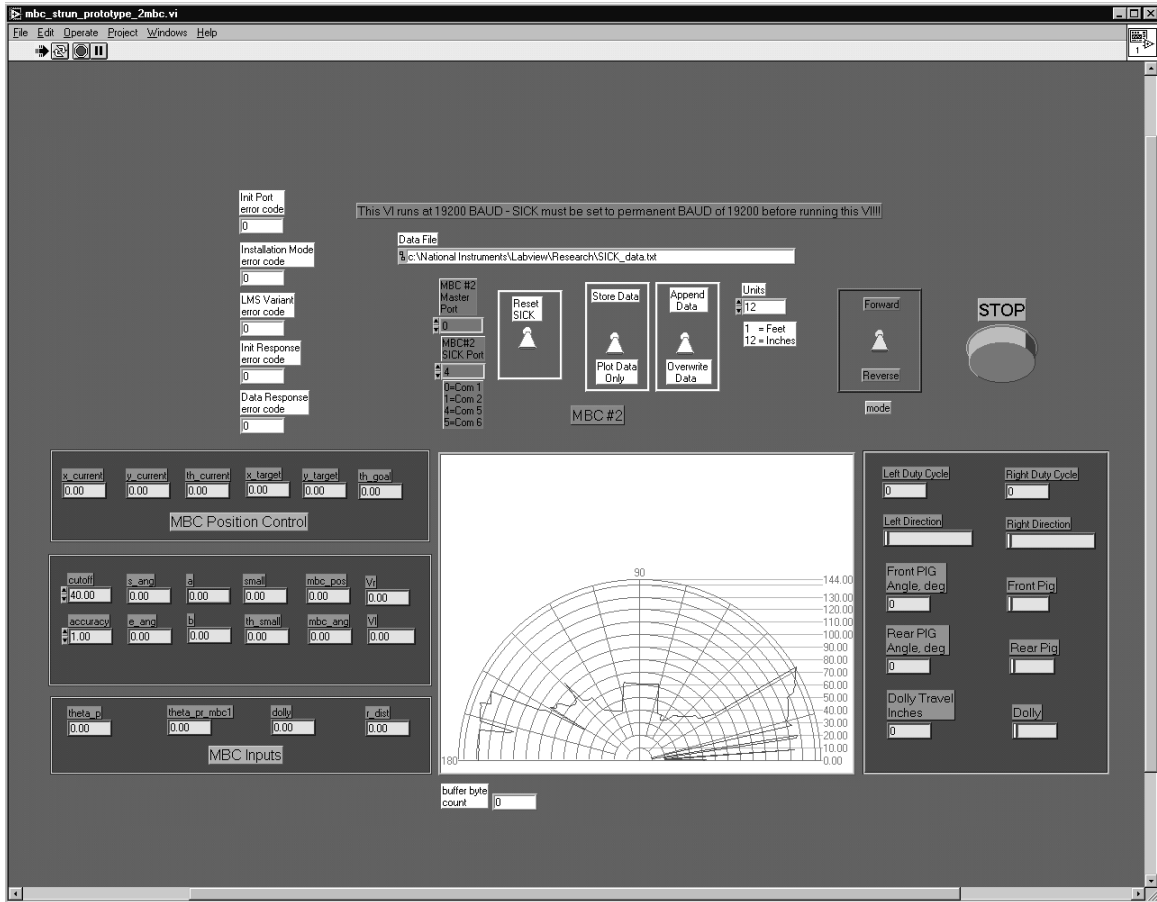


Figure D-1. Front Panel Snapshot of MBC_STRUN_PROTOTYPE_2MBC.VI

Inputs:

- Communications ports for MBC master controller and SICK Optic LMS 200
- Data storage: file name and path, toggle switch to determine storage or display, toggle switch to append or overwrite data.
- Direction control through a toggle switch.
- STOP button commands the MBCs to stop by setting PWM = 0%

Outputs:

- Polar plot of LMS 200 data
- Indicators of MBC configuration: front and rear pig angles, and dolly travel measurement
- Indicators of path-planning algorithms
- Indicators of MBC velocity commands

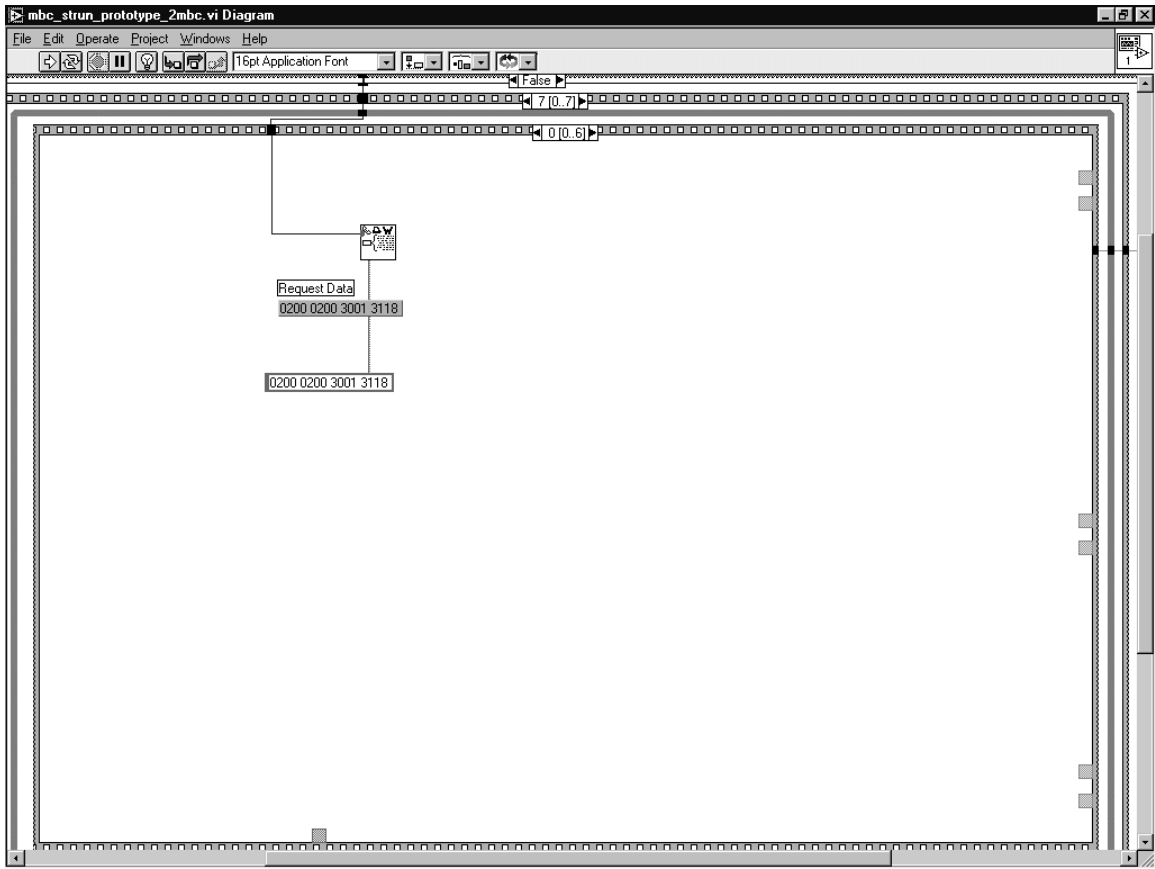


Figure D-2. Diagram Panel Snapshot 1 of 7

VI is same as that for SICK_Interface.VI described in Appendix E, except in frame 7 of 7. The 4 frame sequence structure embedded in frame 7 of 7 was modified to call manipulated LMS data for the path-planning and control algorithms, format the command output into PWM duty cycles and MBC direction to be sent to the MBC master controller.

Frame 0 of 6: Use 'Serial Port Write.vi' to send telegram 30h01h requesting data be sent computer. Complete telegram is given by the string constant '0200 0200 3001 3118'.

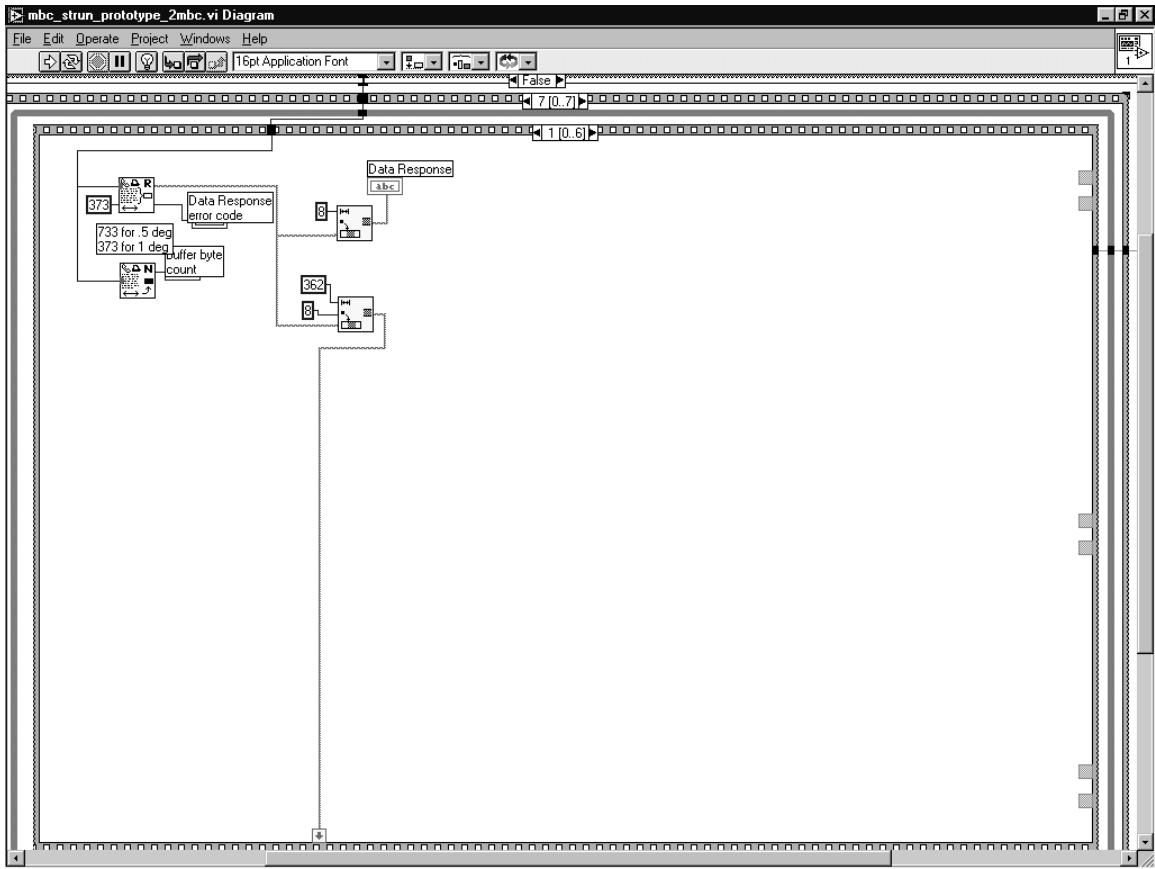


Figure D-3. Diagram Panel Snapshot 2 of 7

Frame 1 of 6: Use 'Serial Port Read.vi' to receive 373 bytes of data. The first 8 bytes are removed using the 'String Subset.vi' because these are the response from the LMS device. The next 362 bytes correspond to 181 16-bit distance measurements. These measurements are removed from the response string and wired to a sequence local to pass data to the next frame.

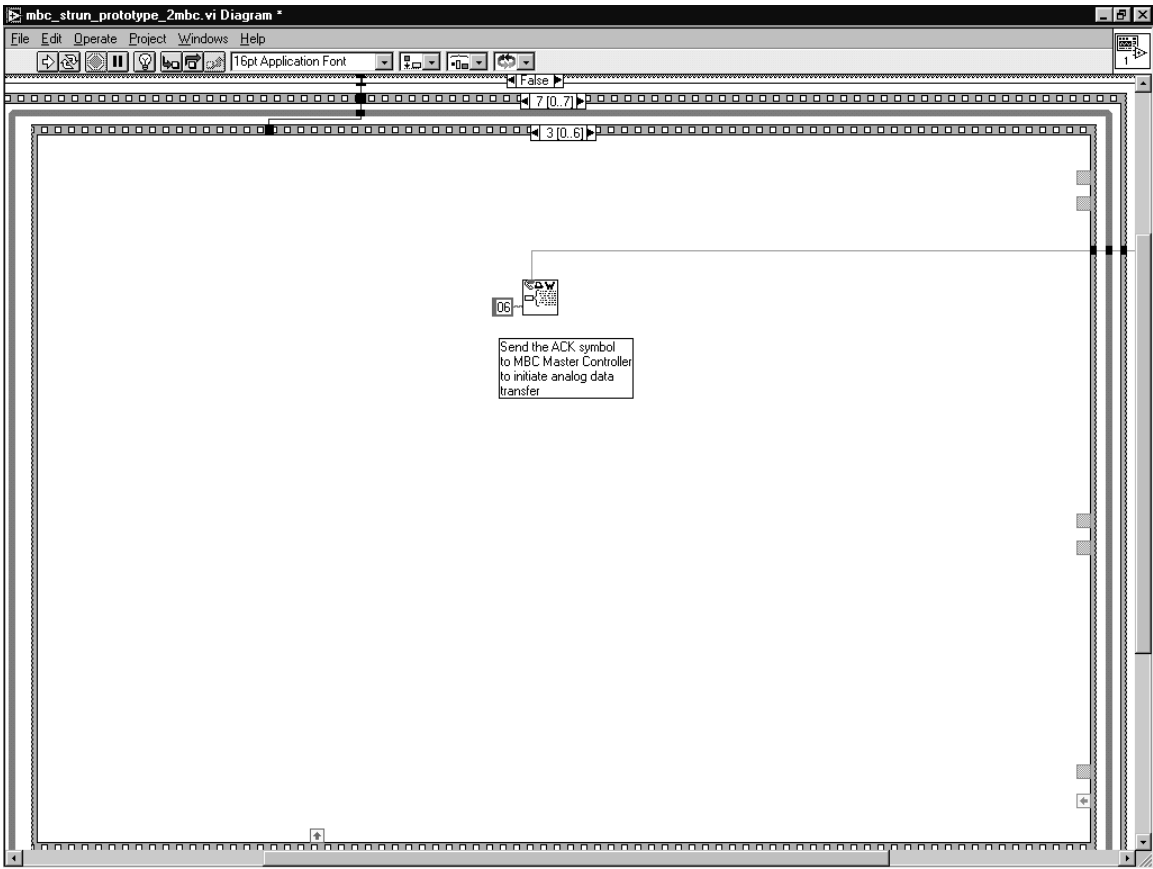


Figure D-5. Diagram Panel Snapshot 4 of 7

Frame 3 of 6: Instruct the MBC master controller begin its program cycle by sending the ACK symbol, which is 06 in hexadecimal. The 'Serial Port Write.vi' subvi is used.

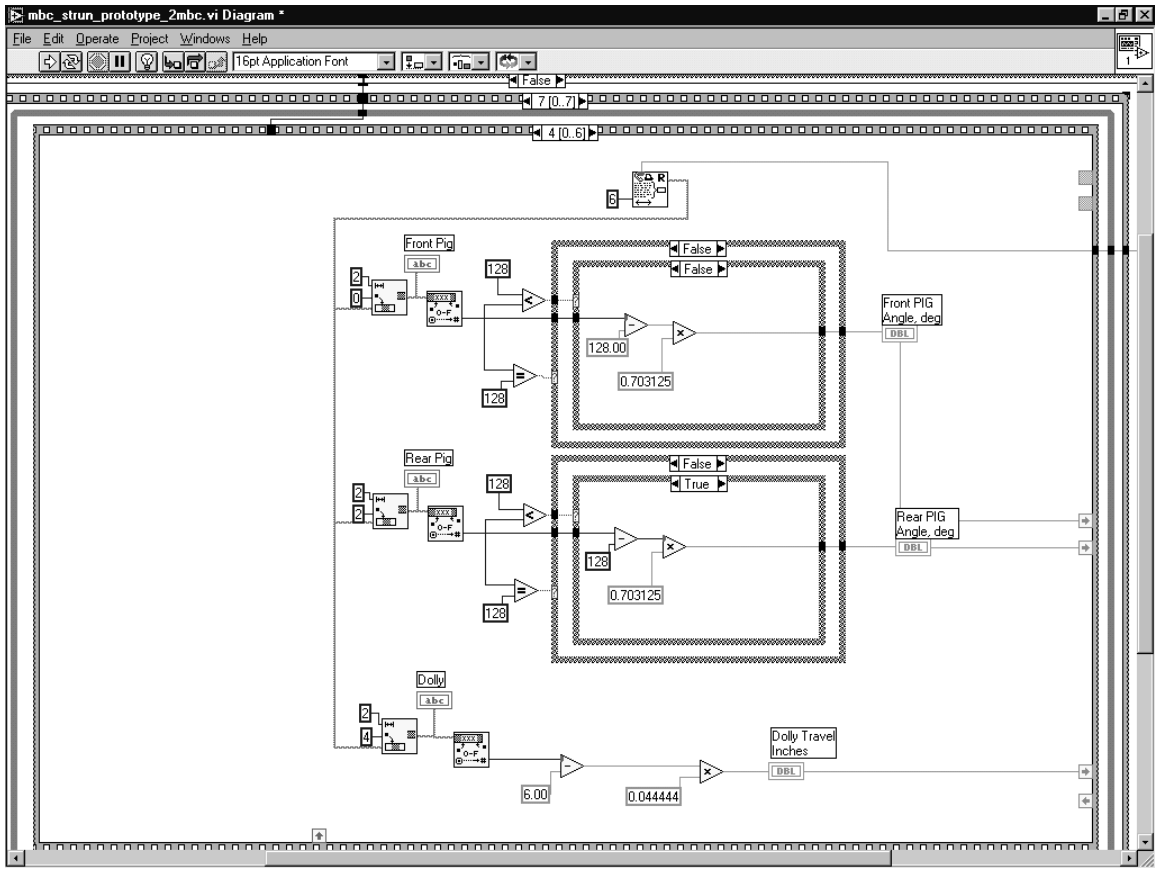


Figure D-6. Diagram Panel Snapshot 5 of 7

Frame 4 of 6: The front pig and rear pig angles, and the dolly travel distance are received by the interface program using the 'Serial Port Read.vi' subvi. Each 8-bit measurement is represented by two bytes of ASCII in order to use the 'From Hexadecimal.vi' to convert data into decimal. Both the front and rear pig angles use two case structures in order to determine the angle. The measured value is compared to determine if it is equal to 128. If equal to 128, the outer case structure is TRUE and the resulting angle is 0. If not equal to 128, the angle is given by $(\text{ANGLE VALUE} - 128) * .703125$. This is true of both front and rear pig angle measurements. The dolly travel measurement is converted into decimal. A constant, equal to 6.00 in this case, is subtracted in order to calibrate the dolly travel to fully closed position. The resulting value is multiplied by the constant .044444 to convert into inches. Both the calculated angles and dolly travel distance are wired to sequence locals to transfer the data to the next frame.

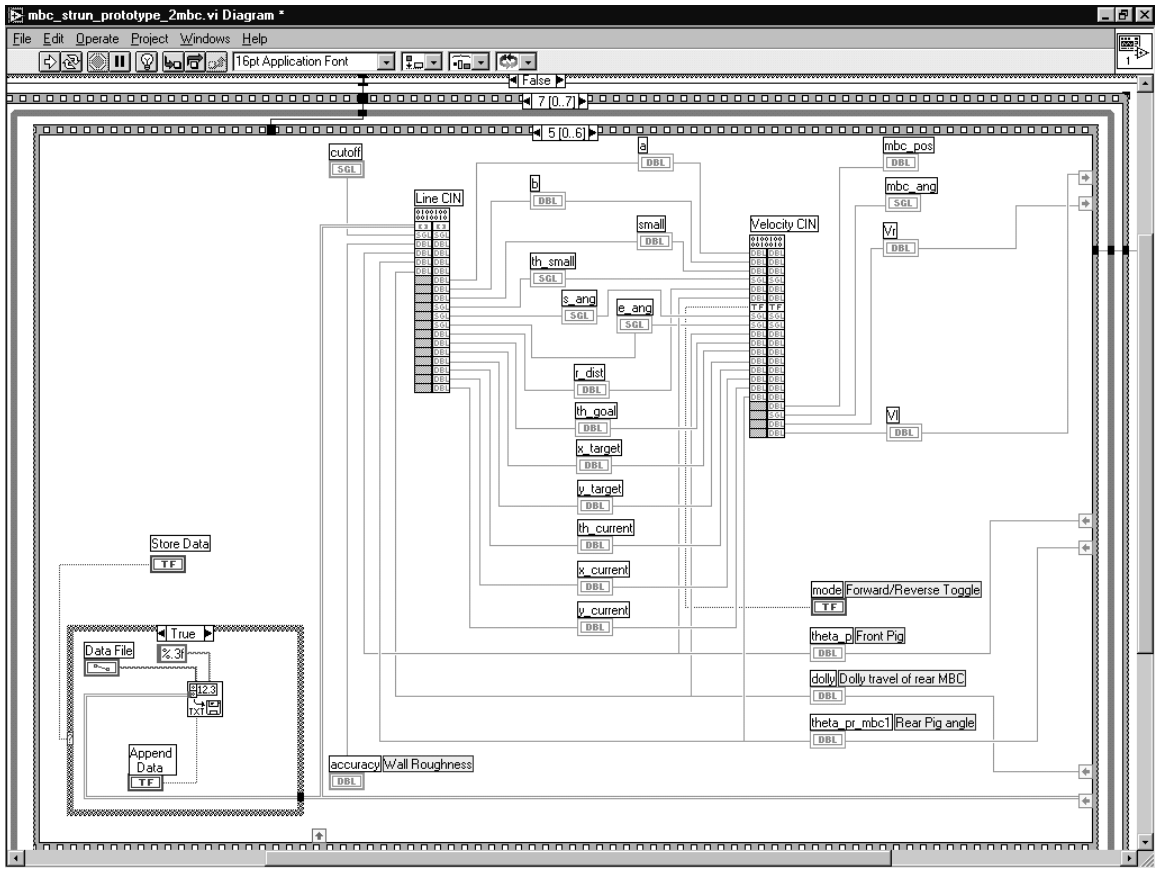


Figure D-7. Diagram Panel Snapshot 6 of 7

Frame 5 of 6: Call the Line.c and Velocity.c using the Call Interface Node (CIN) in order to compute the path-plan and issue the velocity and direction commands. The commands are wired to sequence locals in order to pass the data to the next frame. These algorithms were written by Aishwarya Varadhan and Amnart Kanarat.

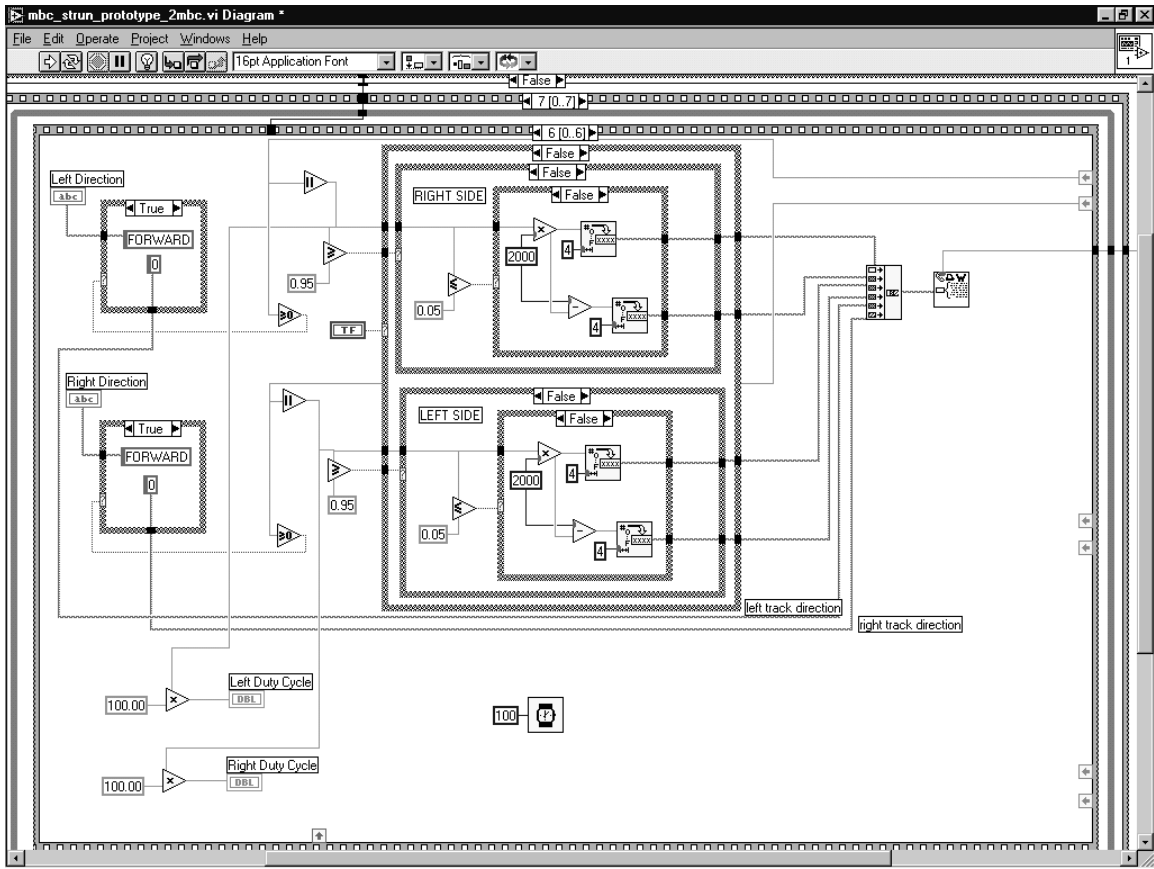


Figure D-8. Diagram Panel Snapshot 7 of 7

Frame 6 of 6: This frame takes the command data and formats into the required values for the PWM motor control generated by the MBC slave controller. The control algorithm generates an output that varies between +1 and -1; +1 is full forward, -1 is full reverse and 0 is off. The command data for each motor is compared to see if greater than or equal to zero. If greater than, the direction case structure is TRUE and the FORWARD command, given by the string constant '0', is output to a build string node. Otherwise, the direction case structure is FALSE and the REVERSE command, given by the string constant 'F', is output to a build string node. The PWM duty cycles are calculated within embedded case structures. The main case structure that contains both the left and right side case structures is controlled by the 'STOP' button on the Front Panel of the VI. When the 'STOP' button is depressed, the case structure is TRUE. The resulting string constants of '00FF' and 'FFFF' are wired to a build string node in order to specify the motors are off. Reference Appendix C for information on PWM operation. If 'STOP' is FALSE, then comparisons are conducted on the absolute value of the command data. If the command is greater than .95, then the string constants '076C' and '0064' are output to build string node. These values are greatest values that can be used for the PWM duty cycle without causing erratic operation of the controller due to the time

required to complete the interrupt service routine. If the values are less than .95, but greater than .05, then the PWM duty cycle is calculated as follows:

$$\text{16-bit PWM high time} = \text{abs|COMMAND|} * 2000$$

$$\text{16-bit PWM low time} = 2000 - \text{abs|COMMAND|}$$

Both of these values are converted into hexadecimal using 'To Hexadecimal' and the resulting hexadecimal values are wired to the build string node. If the command value is less than .05, the string constants '00FF' and 'FFFF' are output to the build string node to essentially force the motors to be OFF. Again, this is due to the minimum time required to complete the interrupt service routine. The build string node is constructed in the following manner: left PWM high time, left PWM low time, right PWM high time, right PWM low time, left direction and right direction. This string is then written to the serial port to be received by the MBC master controller.

Appendix E: SICK Optic LMS 200 LabVIEW Interface

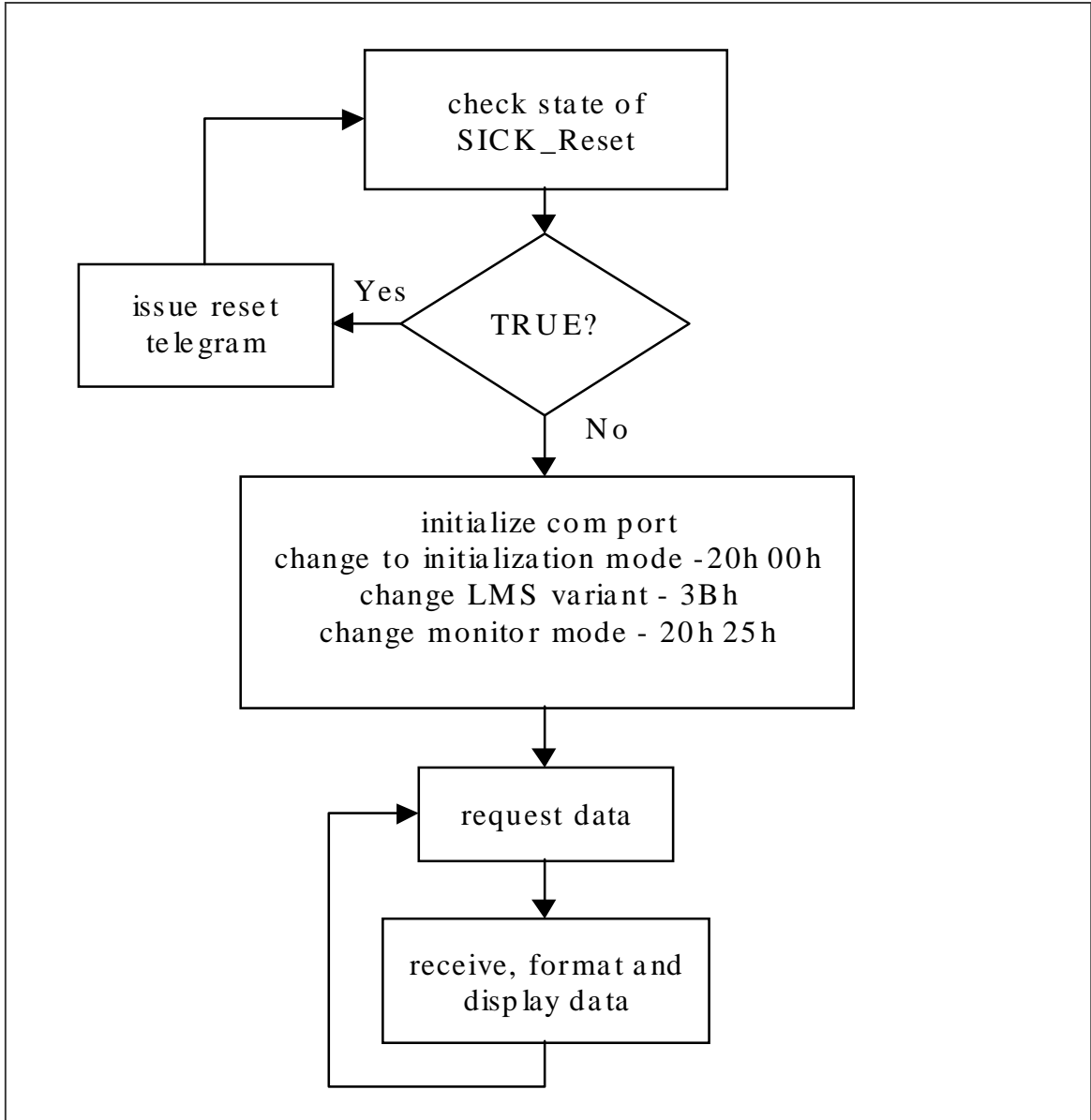


Figure E-1. Flowchart of LabVIEW Interface Program

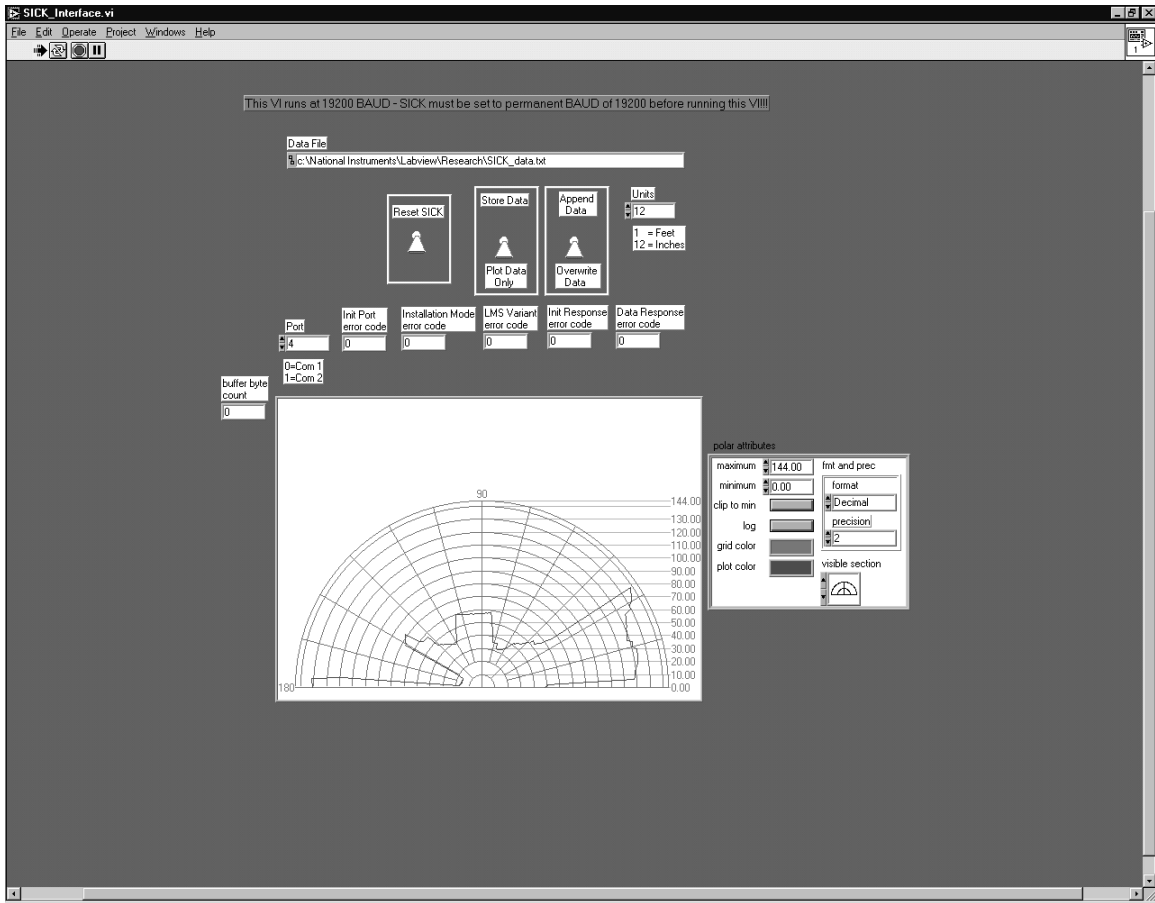


Figure E-2. Front Panel Snapshot of SICK_Interface.vi

Inputs:

Communications port for SICK Optic LMS 200

Data storage: file name and path, toggle switch to determine storage or display, toggle switch to append or overwrite data.

Polar graph attributes.

Outputs:

Polar graph of LMS 200 data

Serial port error indicators

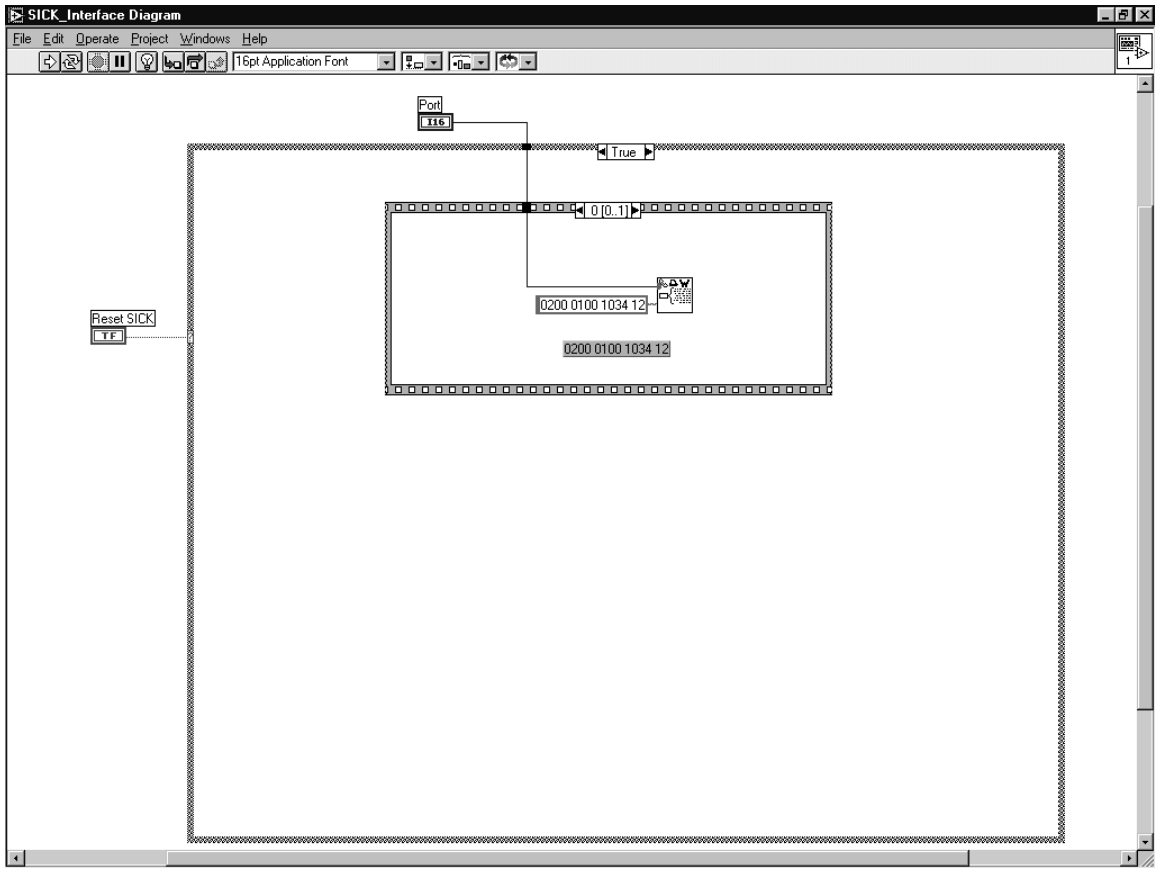


Figure E-3. Diagram Panel Snapshot 1 of 13

If toggle switch 'Reset SICK' TRUE, reset the LMS 200 using a 2 frame sequence structure.

Frame 0 of 1: Initialize and reset LMS by writing telegram number 10H to Port using 'Serial Port Write.vi'. Complete telegram is given by string constant '0100 0100 1034 12'.

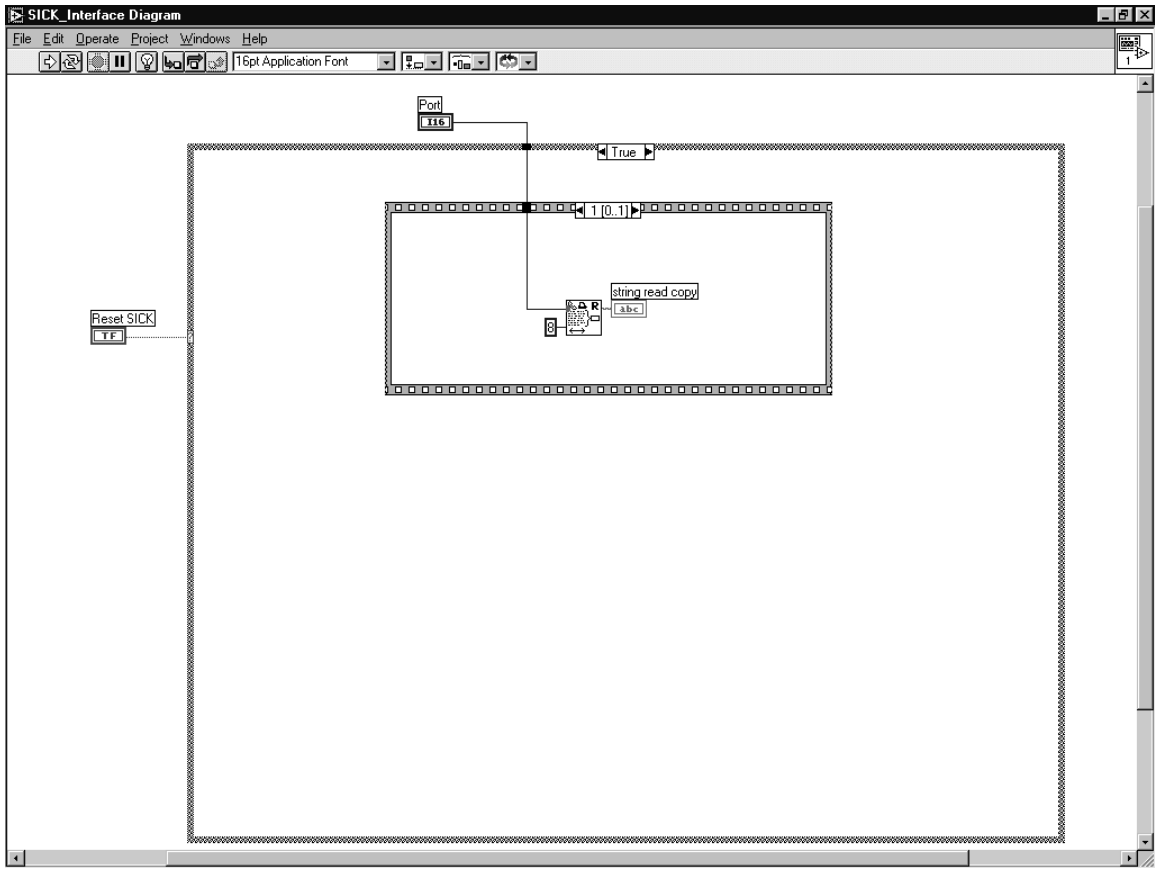


Figure E-4. Diagram Panel Snapshot 2 of 13

Frame 1 of 1: Receive results of previous telegram to Initialize and reset. The response string from LMS is 8 bytes, and is received using 'Serial Port Read.VI'. This frame also clears COM buffer.

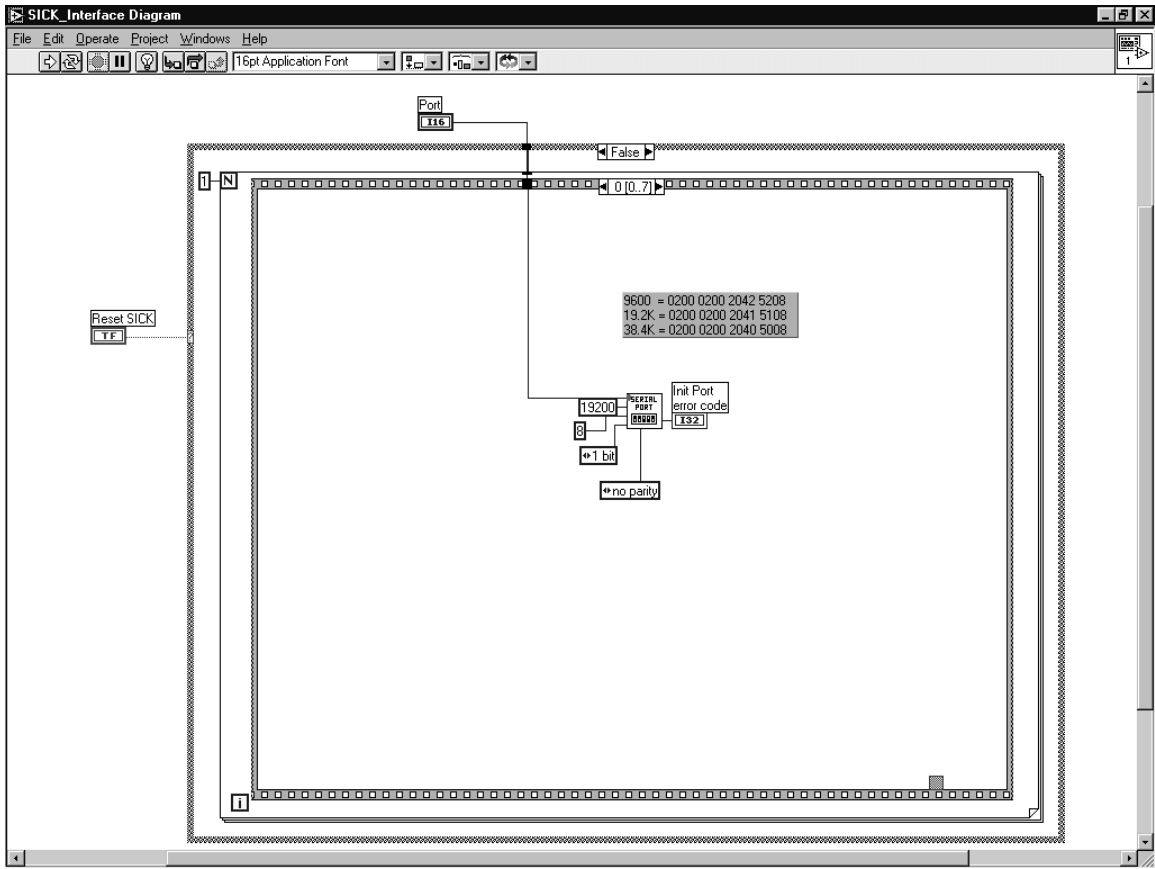


Figure E-5. Diagram Panel Snapshot 3 of 13

If 'Reset SICK' switch FALSE, then case structure is FALSE and the 8 frame sequence structure operates. Frames 0 through 7 are contained in a 'For Loop' which will only cycle once to initialize the LMS 200.

Frame 0 of 7: Initialize COM Port settings using 'Serial Port Init.VI'.

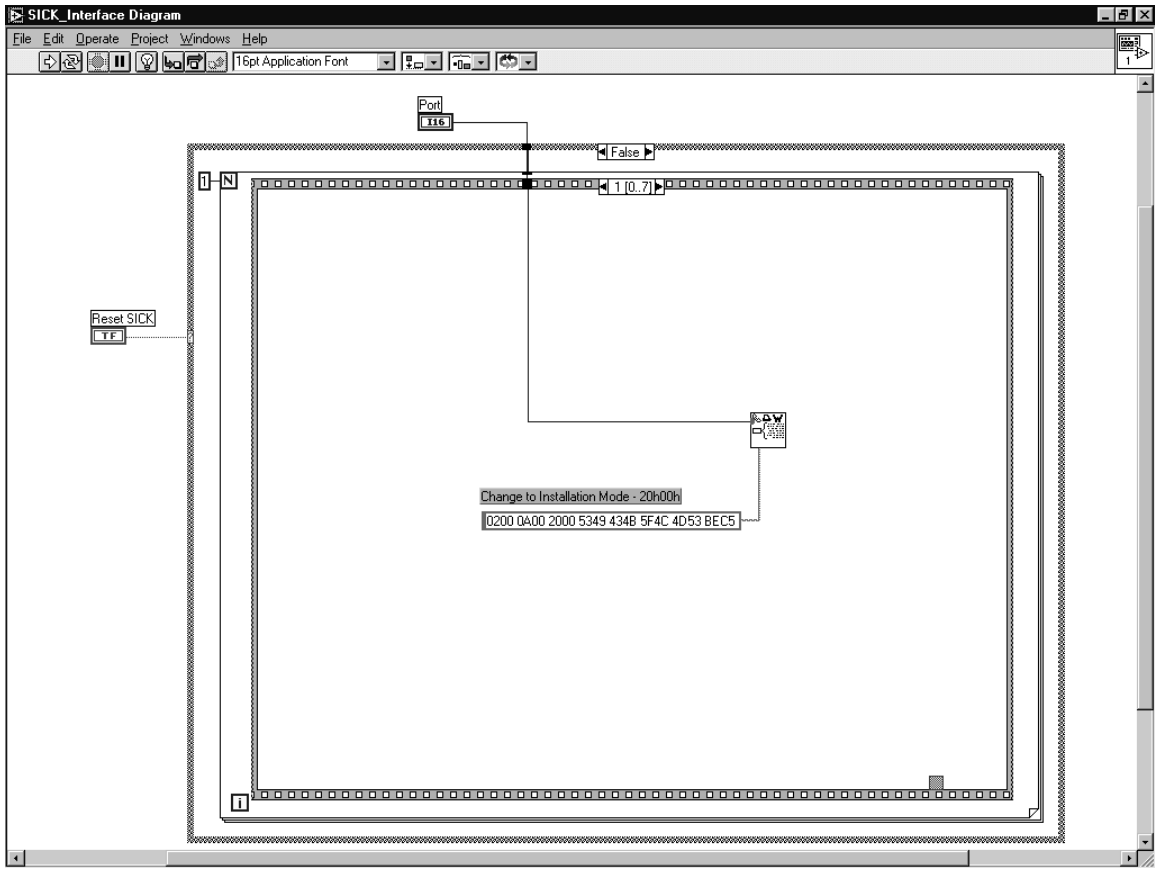


Figure E-6. Diagram Panel Snapshot 4 of 13

Frame 1 of 7: Change to Installation Mode (Telegram number 20h, Mode 00h) by writing string to 'Serial Port Write.VI'. Installation mode must be selected in order to use certain configuration telegrams. Complete telegram is given by string constant '0200 0A00 20005349 434B 5F4C 4D53 BEC5'.

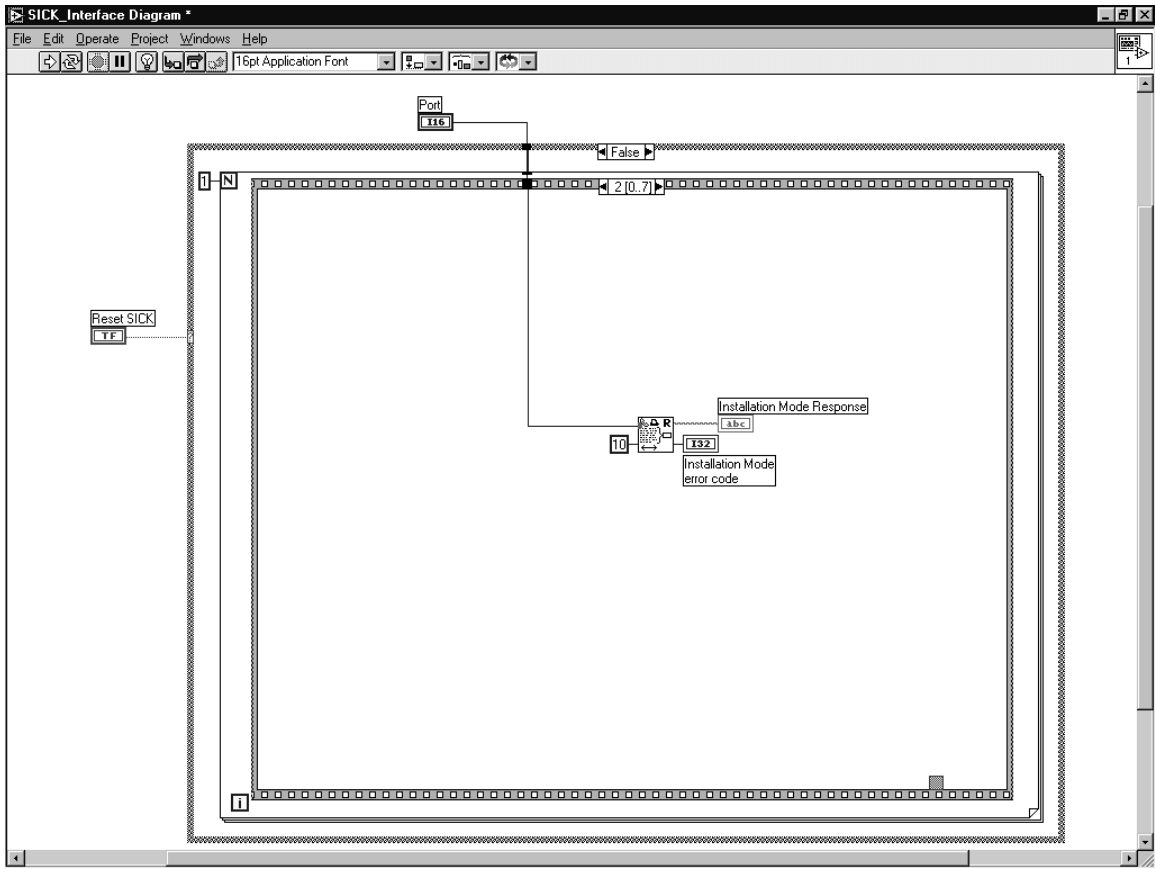


Figure E-7. Diagram Panel Snapshot 5 of 13

Frame 2 of 7: Receive LMS response to telegram using 'Serial Port Read.VI'. By reading the COM Port, the COM buffer is cleared. The response could be used for validation of operation. Total of 10 bytes received through COM Port.

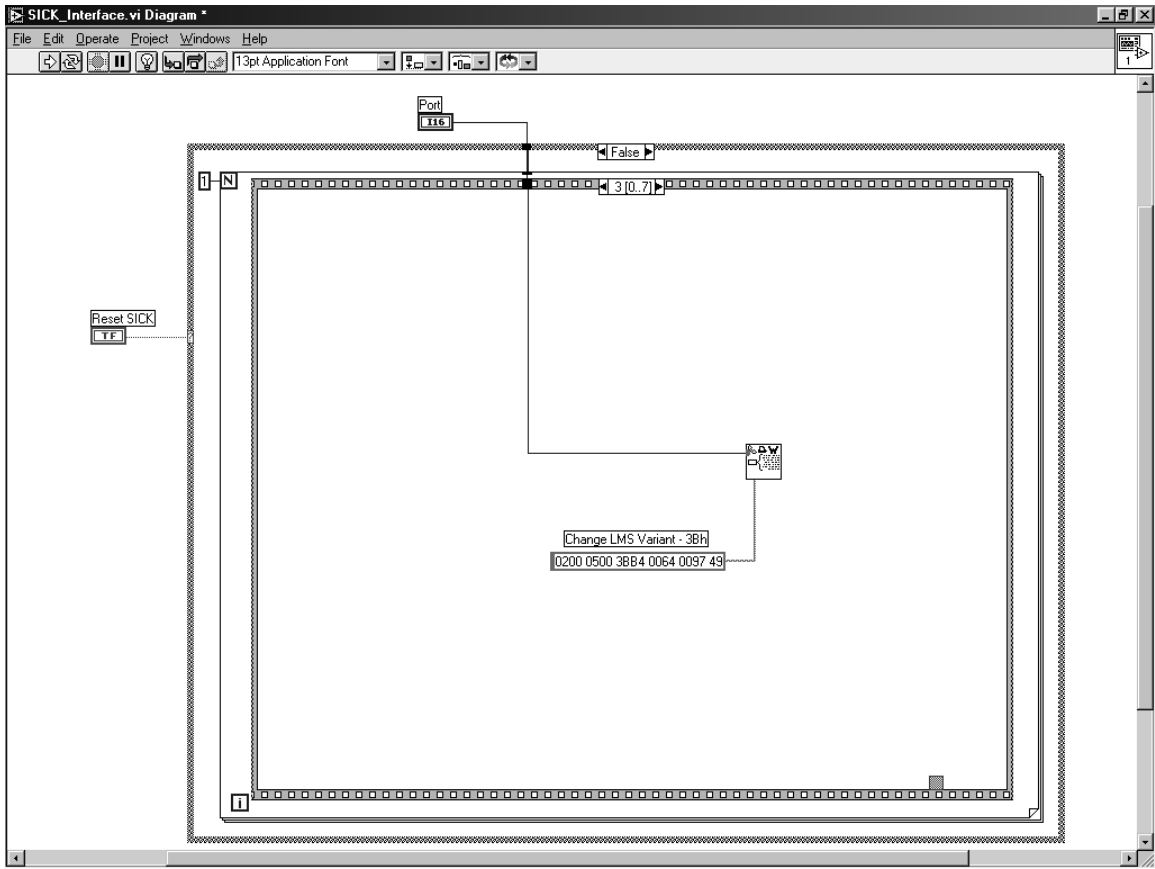


Figure E-8. Diagram Panel Snapshot 6 of 13

Frame 3 of 7: Change LMS Variant (Telegram number 3Bh, angle of scan = 180°, single-shot resolution = 1°) using 'Serial Port Write.VI'. Configure the LMS to have a 180° scan angle with a resolution of 1° for a total of 181 measurement points. Complete telegram is given by string constant '0200 0500 3BB4 0064 0097 49'.

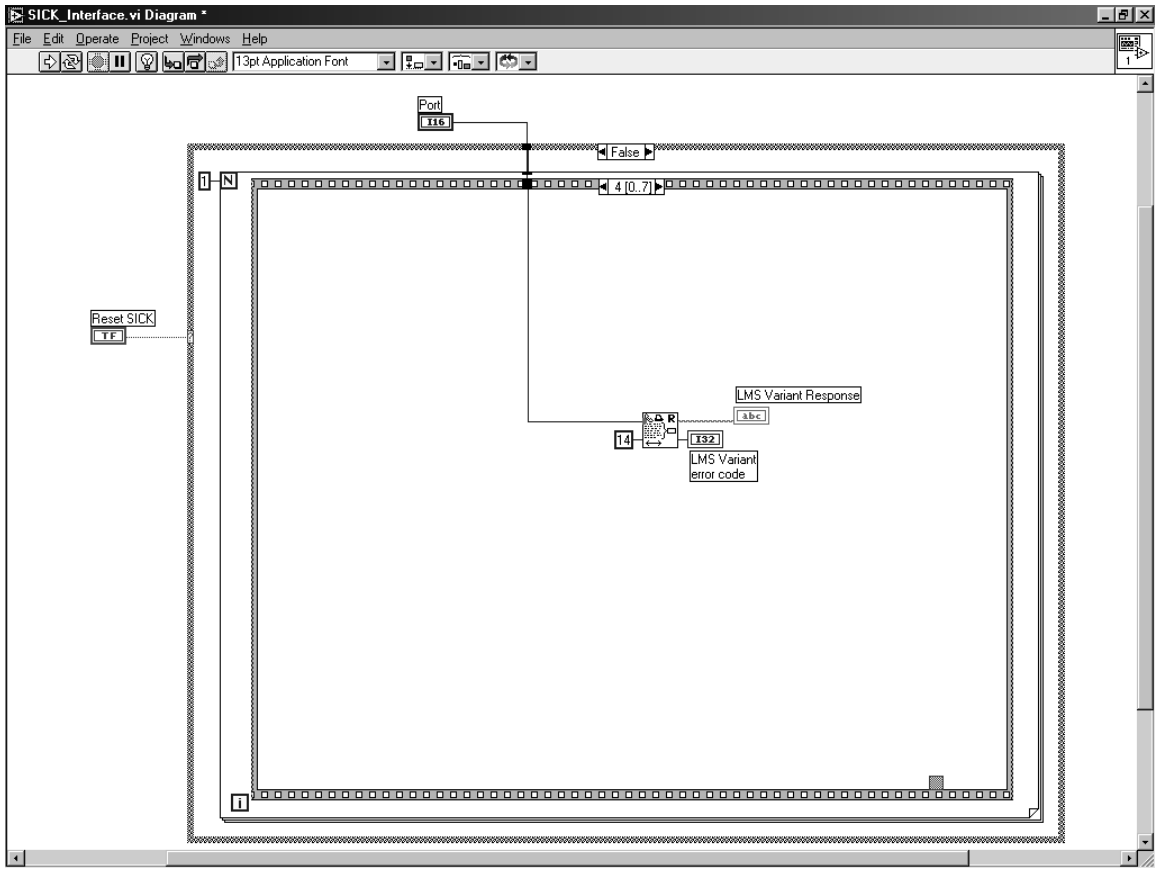


Figure E-9. Diagram Panel Snapshot 7 of 13

Frame 4 of 7: Receive LMS response to previous telegram by using 'Serial Port Read.VI'. By reading the COM Port, the COM buffer is cleared. The response could be used for validation of operation. Total of 14 bytes received through COM Port.

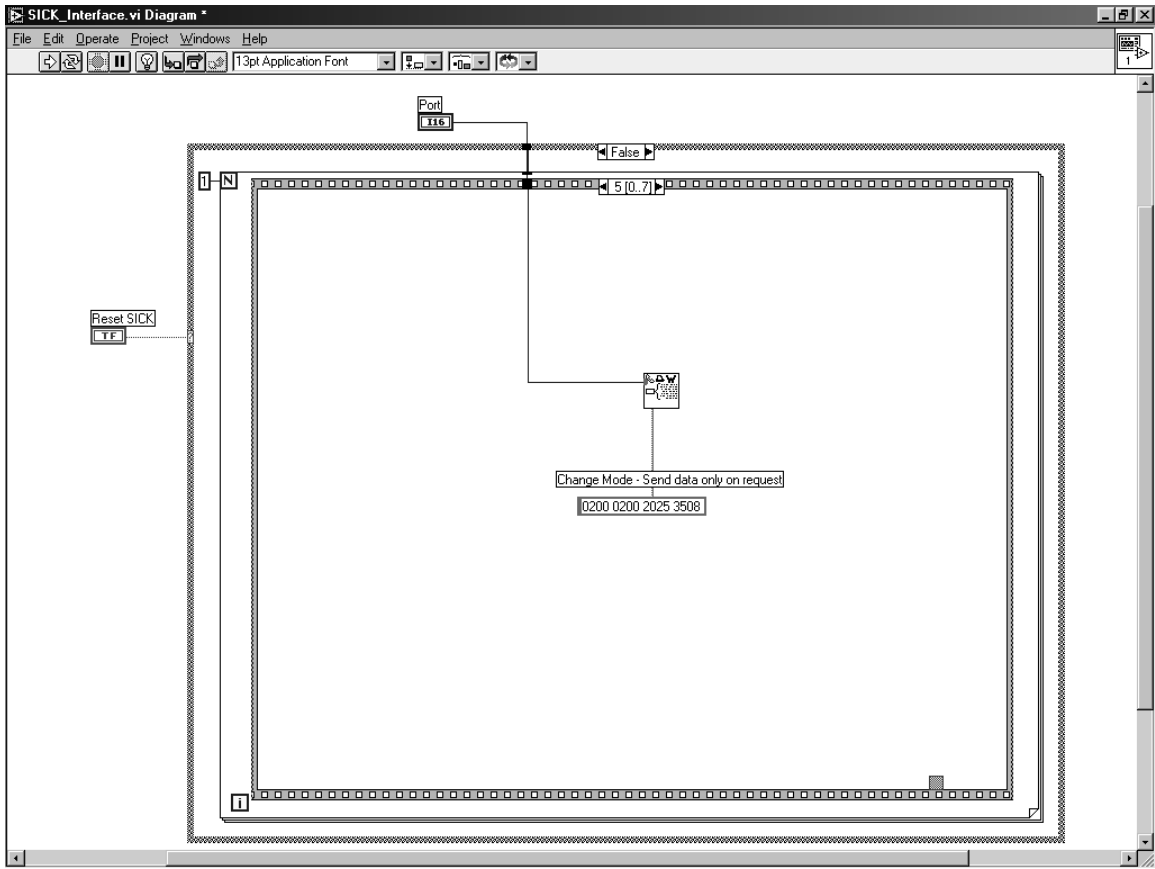


Figure E-10. Diagram Panel Snapshot 8 of 13

Frame 5 of 7: Change LMS monitoring mode (Telegram number 20h, Mode 25h) using 'Serial Port Write.VI'. This telegram configures the LMS to send measured values only on request. Complete telegram is given by string constant '0200 0200 2025 3508'.

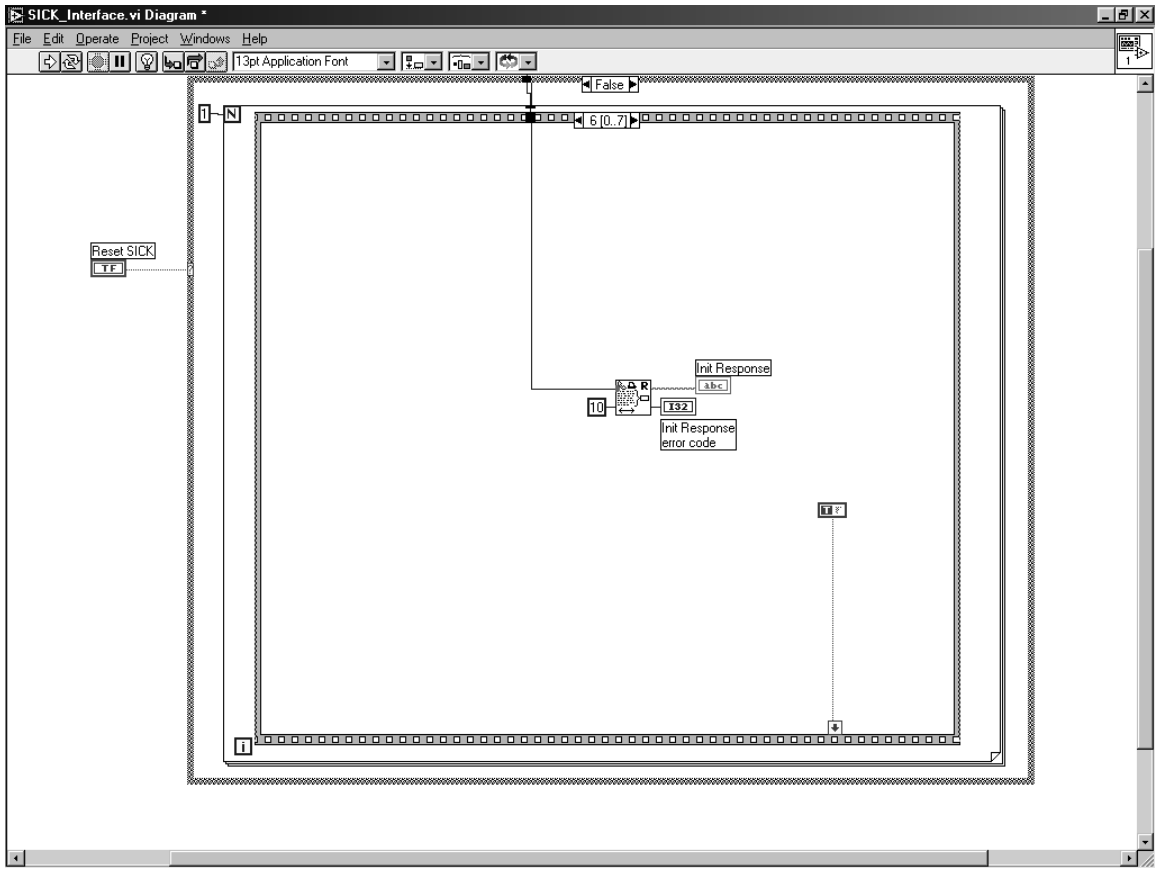


Figure E-11. Diagram Panel Snapshot 9 of 13

Frame 6 of 7: Receive LMS response to previous telegram by using 'Serial Port Read.VI'. By reading the COM Port, the COM buffer is cleared. The response can be used for validation of operation. Total of 10 bytes received through COM Port. A boolean constant is wired to a sequence local that passes TRUE to the 'While Loop' in the next frame. Upon execution of frame 7, the VI will continually execute the embedded sequence structure' located in frame 7 until the VI is stopped.

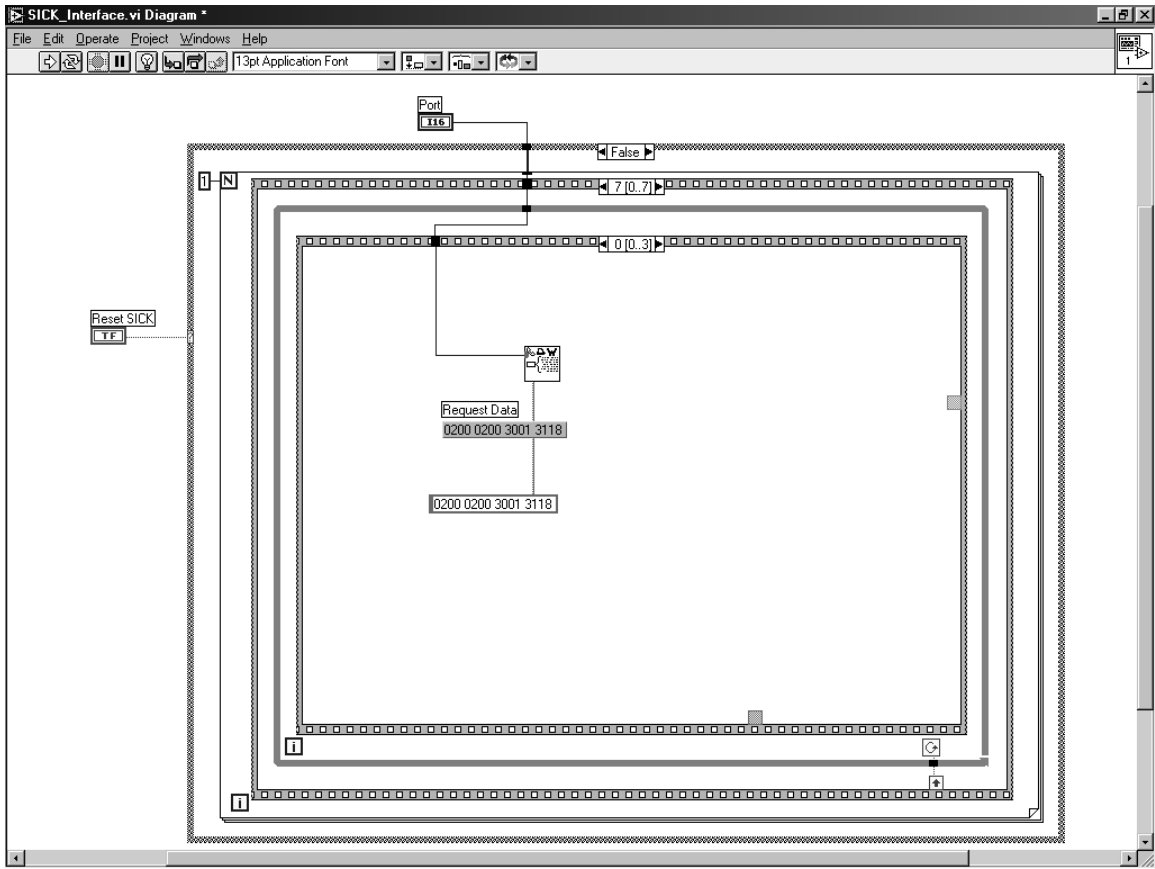


Figure E-12. Diagram Panel Snapshot 10 of 13

Frame 7 of 7: embedded 4 frame sequence structure located inside 'While Loop'.

Frame 0 of 3: Use 'Serial Port Write.vi' to send telegram 30h01h requesting data be sent computer. Complete telegram is given by the string constant '0200 0200 3001 3118'.

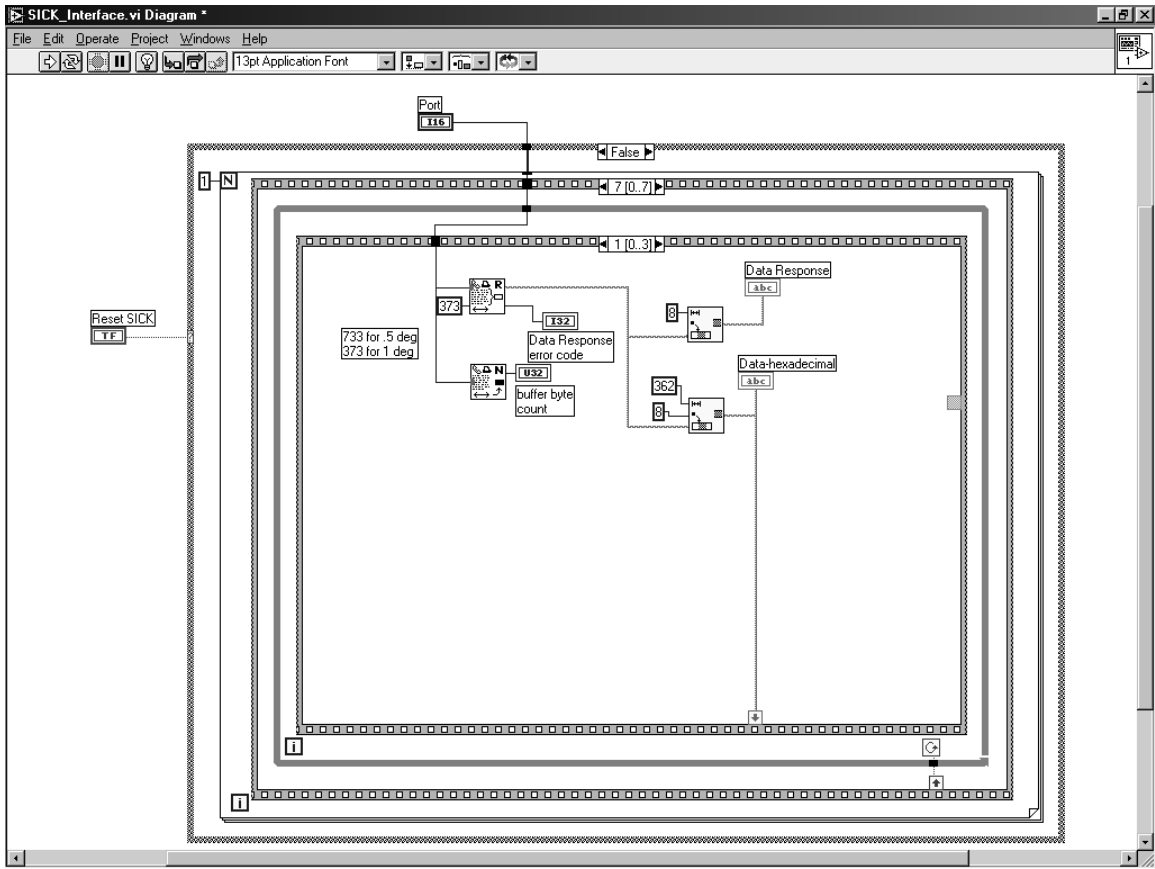


Figure E-13. Diagram Panel Snapshot 11 of 13

Frame 1 of 3: Use 'Serial Port Read.vi' to receive 373 bytes of data. The first 8 bytes are removed using the 'String Subset.vi' because these are the response from the LMS device. The next 362 bytes correspond to 181 16-bit distance measurements. These measurements are removed from the response string and wired to a sequence local to pass data to the next frame.

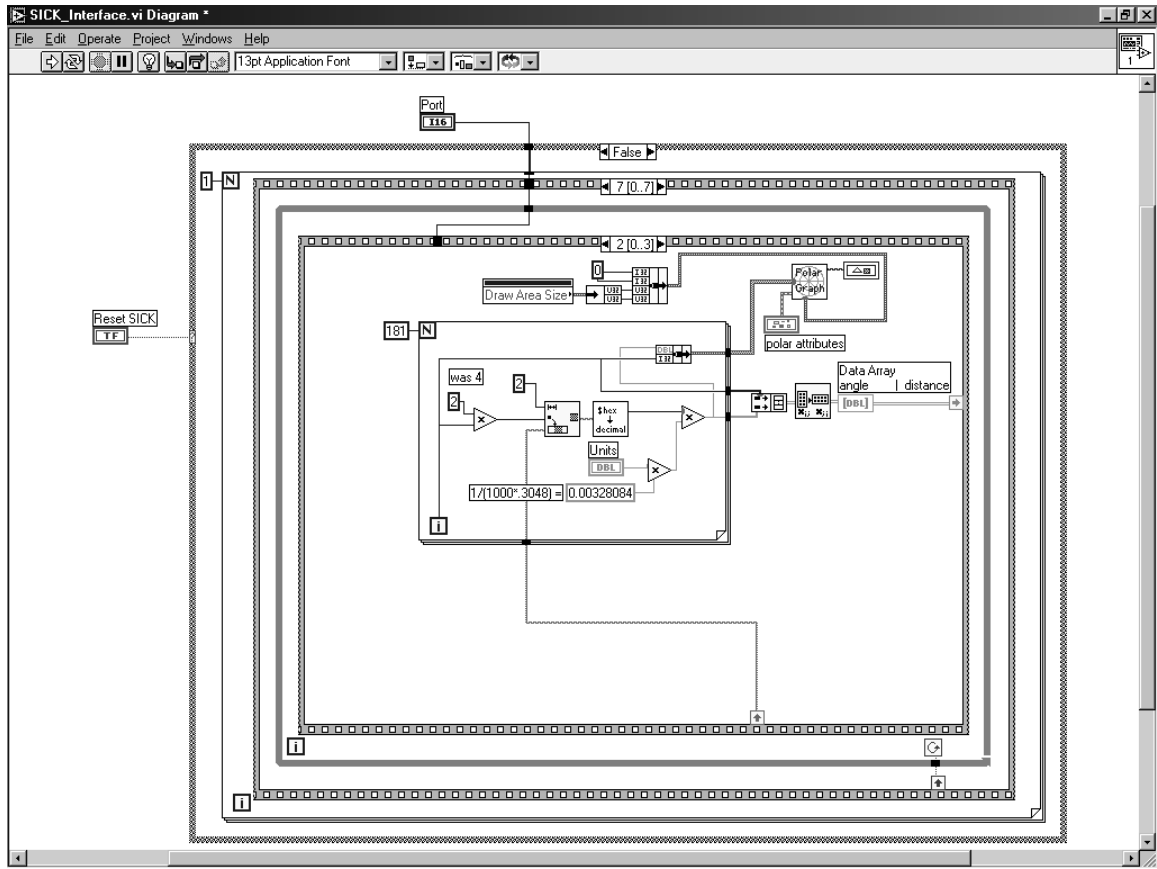


Figure E-14. Diagram Panel Snapshot 12 of 13

Frame 2 of 3: Use FOR LOOP to manipulate the 181 measurement values. The 'For Loop' uses a 'Multiply' node wired to 'String Subset.vi' to index through the data with each increment of loop. The indexed string values are converted from 16-bit hexadecimal into decimal using 'Hex to Decimal Example.vi' subvi. The for loop is also used to create the angular values. Since the LMS sends only measured data, the interface program must assume the angle from the first data value. Because the LMS is sending measurements at 1° increments, the increment counter of the FOR LOOP becomes the angle. Both values are stored in an array, which is wired to sequence local to pass data to next frame. The angle and measurement values are stored to a cluster in order to provide the data to the 'Polar Graph.vi' subvi.

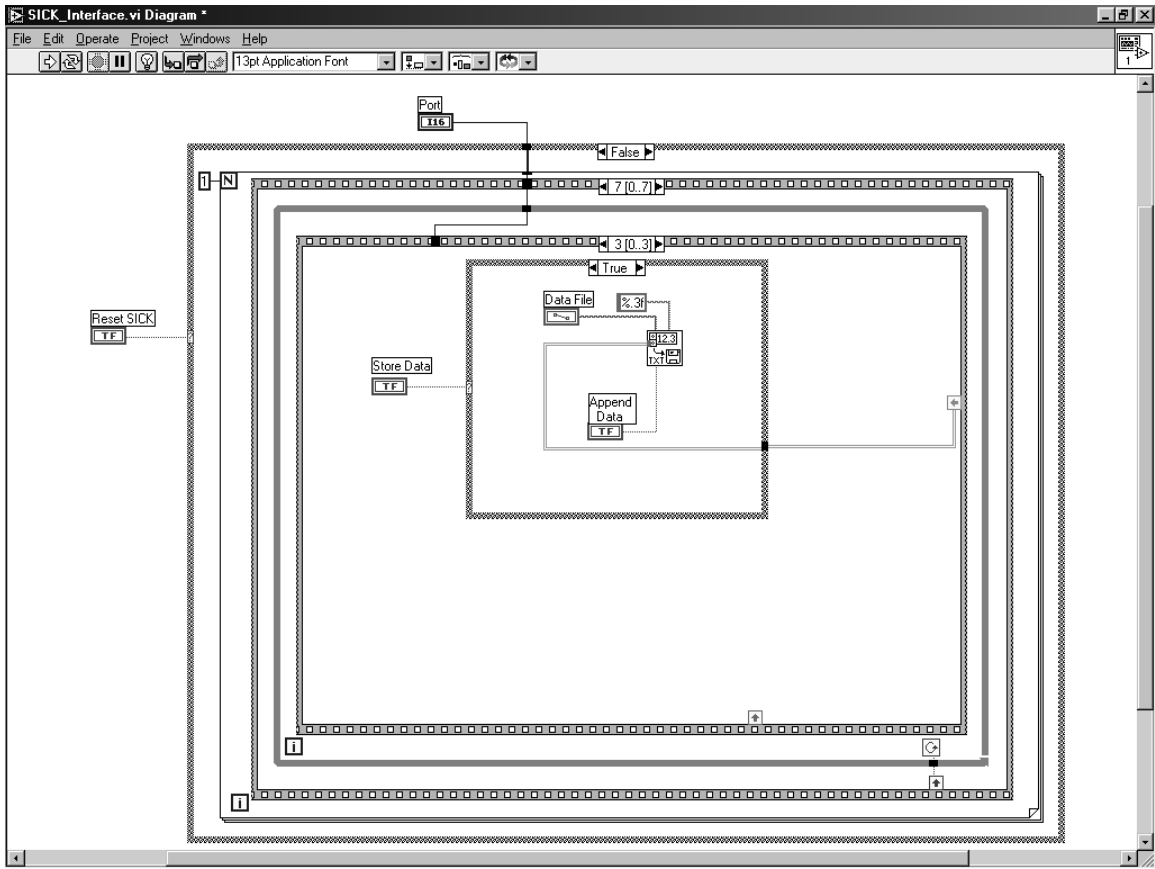


Figure E-15. Diagram Panel Snapshot 13 of 13

Frame 3 of 3: This frame completes the 4-frame sequence that requests, formats and displays the LMS measurement data. If the 'Store Data' boolean toggle switch is TRUE, then the measurement and angle data is stored to a text file. The file path is given by the entering the path into the 'File Path Control' labeled 'Data File.' Operation of the VI will loop to the first frame in the sequence because the 'While Loop' is always TRUE until the VI is stopped.

Appendix F: Full-Scale MBC Hardware and Software

F.1: Fulltest2.asm

*This program interfaces with two 0808 DAC chips for control of APITECH
*PWM drivers reads two slide pots and outputs 50-100% duty cycle, or
*off. Runs in either manual or automatic modes - manual reads slide
*pots and calculates duty cycle, automatic receives 4 bytes from SCI
*interface - left and right speed values(1 byte each) and lt and rt
*direction values(1 byte each).

```
RAM      EQU      $01F0

          ORG      $B600
          LDS      #$01D0
          LDX      #REGS
          BSET     PACTL,X,%10001000    ;PORTA PINS 3,7 CONFIGURED AS

OUTPUT
          LDAA     #$FF
          STAA     DDRC,X                ;PORTC as OUTPUT
          JSR      INITAD
          JSR      SCI_INIT
          LDAA     #$00
          STAA     PORTB,X              ;make sure both tracks are off
          STAA     PORTC,X

OPER_MODE
          BRSET   PORTA,X,$01,MANUAL    ;CHECK PA0,HIGH-MANUAL MODE
          BRA     AUTO                  ;OTHERWISE OPERATE IN AUTOMATIC MODE

MANUAL
          LDAA     ADR4,X              ;LEFT SIDE TRACK-PE3
          LDAB     #$80
          JSR      DUTY
          STAB     PORTB,X
          LDAA     ADR3,X              ;RIGHT SIDE TRACK-PE2
          LDAB     #70
          JSR      DUTY
          STAB     PORTC,X
          BRA     OPER_MODE

AUTO
          BRCLR   SCSR,X,$20,AUTO ;wait here for start signal(06 hex)
          LDAA     SCDR,X
          CMPA     #$06
          BNE     AUTO

          LDAA     ADR2,X
          STAA     PIGFRT
          LDAA     ADR1,X
          STAA     PIGREAR

          JSR      SEND_AD
          JSR      SCI_REC
          JSR      PWM_CHNG
          JSR      DUTY_CHNG
```

```

        BRA      OPER_MODE
*****
*SCI_INIT - Initialize the Serial Communications Interface
*****
SCI_INIT
        LDAA    #$30
        STAA    BAUD,X      ;BAUD REGISTER
        LDAA    #$00
        STAA    SCCR1,X     ;SCCR1 SCI CONTROL REG 1 SET UP
        LDAA    #$0C
        STAA    SCCR2,X     ;SCCR2 SCI CONTROL REG 2 SET UP
        LDAA    SCSR,X      ;PURGE RECEIVE FLAGS
        LDAA    SCDR,X      ;AND RECEIVE DATA
        RTS

*****
*SUBROUTINE INITAD-INITIALIZES A/D SYSTEM
*****
INITAD
        BSET    OPTION,X,%10000000
        BCLR    OPTION,X,%01000000
        BSET    ADCTL,X,%00110000
        BCLR    ADCTL,X,%00001111
        RTS

*****
*SEND_AD - SEND POTENTIOMETER MEASUREMENTS
*****
SEND_AD
        LDAA    PIGFRT
        JSR    OUTLHLF
        LDAA    PIGFRT
        JSR    OUTRHLF
        LDAA    PIGREAR
        JSR    OUTLHLF
        LDAA    PIGREAR
        JSR    OUTRHLF
        RTS

*****
*OUTRHLF(), OUTLHLF(), OUTA()
*Convert A from binary to ASCII and output.
*Contents of A are destroyed..
*****

OUTLHLF LSRA                ;shift data to right
        LSRA
        LSRA
        LSRA
OUTRHLF ANDA    #$0F        ;mask top half
        ADDA    #$30        ;convert to ascii
        CMPA    #$39
        BLE     OUTA        ;jump if 0-9
        ADDA    #$07        ;convert to hex A-F
OUTA    JSR     SCI_SEND    ;output character
        RTS

*****
*SCI_SEND - Sends a byte through UART
*****
SCI_SEND

```

```

        LDX    #REGS
TBNMT   BRCLR   SCSR,X,$80,TBNMT   ;Loop til xmitter output buffer empty
        STAA   SCDR,X
        RTS

*****
*SCI_REC - RECEIVE SERIAL DATA(6 BYTES)
*****
SCI_REC
        LDX    #REGS
        LDAB   #$00
        LDY    #SCIDAT
NOCHNG  BRCLR   SCSR,X,$20,NOCHNG   ;Test for RDRF receive character in
        LDAA   SCDR,X               ;SCSR register, 0 no new character.
        STAA   0,Y
        INY
        INCB
        CMPB   #$06                 ;was 4
        BNE    NOCHNG
        RTS

*****
*PWM_CHNG - Convert SCI data from ASCII into 8 bit hex values
*           and stores the new values into PWMDAT
*****
PWM_CHNG
        PSHX
        LDX    #SCIDAT
        LDY    #LTVAL
        LDAA   #$00
        STAA   COUNT
PCHNGLP LDAA   0,X
        INX
        LDAB   0,X
        JSR    TO_HEX
        STAA   0,Y
        INX
        INY
        INC    COUNT
        LDAA   COUNT
        CMPA   #$02                 ;#$08 ADDED 2 BYTES FOR DIRECTION
        BNE    PCHNGLP

        LDAA   0,X                 ;store LT & RT directions without converting
        STAA   0,Y                 ;from ASCII - should get $30(0) & $46(F)
        INX
        INY
        LDAA   0,X
        STAA   0,Y

        PULX
        RTS

*****
*Converts 2 ASCII characters to 1 hex byte returned in ACCA
*requires the ASCII chars representing the hi and low bits
*to be in ACCA and ACCB, respectively...
*****
TO_HEX

```

```

HIBIT   CMPA   #$39
        BLE   HIBIT1
        SUBA   #$07
HIBIT1  LSLA
        LSLA
        LSLA
        LSLA
        ANDA   #$F0   ;mask lower half
LOWBIT  CMPB   #$39
        BLE   LOWBIT1
        SUBB   #$07
LOWBIT1 ANDB   #$0F   ;mask upper half
        ABA
        RTS
*****
*SUBROUTINE DUTY_CHNG
*
*Stores the duty cycles and the directions of the motors
*****
DUTY_CHNG

        LDAA  LTVAL
        STAA  PORTB,X
        LDAA  RTVAL
        STAA  PORTC,X
        LDAA  LTDIR           ;check for motor direction.....
        CMPA  #$30           ;$30 = FWD, $46 = REV
        BNE  LTREVD
        BSET  PORTA,X,%10000000
        BRA  NEXTD1
LTREVD  BCLR  PORTA,X,%10000000
NEXTD1  LDAA  RTDIR
        CMPA  #$30
        BNE  RTREVD
        BSET  PORTA,X,%01000000
        BRA  DIREND
RTREVD  BCLR  PORTA,X,%01000000
DIREND  RTS

*****
*SUBROUTINE DUTY
*
*Calculates the duty cycles for OC4 & OC5 and the directions of the
motors
*****
DUTY
        CMPA  #$9B           ;COMPARE TO VALUE OF 155
        BHS  REV
        CMPA  #$64           ;COMPARE TO VALUE OF 100
        BLS  FWD
        LDAB  #$00
        RTS

REV
        CMPB  #$80
        BEQ  LTREV

```

```

        BRA    RTREV
LTREV  BCLR   PORTA,X,%10000000 ;THIS SETS REV DIRECTION FOR H-BRIDGE
        BRA    NEXT1
RTREV  BCLR   PORTA,X,%01000000
NEXT1  CMPA   #$F7
        BHS   DC100
        CMPA   #$EE
        BHS   DC95
        CMPA   #$E6
        BHS   DC90
        CMPA   #$DE
        BHS   DC85
        CMPA   #$D6
        BHS   DC80
        CMPA   #$CD
        BHS   DC75
        CMPA   #$C5
        BHS   DC70
        CMPA   #$BD
        BHS   DC65
        CMPA   #$B4
        BHS   DC60
        CMPA   #$AC
        BHS   DC55
        CMPA   #$A4
        BHS   DC50
        BRA    DC45
FWD    CMPB   #$80
        BEQ   LTFWD
        BRA   RTFWD
LTFWD  BSET   PORTA,X,%10000000
        BRA   NEXT2
RTFWD  BSET   PORTA,X,%01000000
NEXT2  CMPA   #$08
        BLS   DC100
        CMPA   #$11
        BLS   DC95
        CMPA   #$19
        BLS   DC90
        CMPA   #$22
        BLS   DC85
        CMPA   #$2A
        BLS   DC80
        CMPA   #$32
        BLS   DC75
        CMPA   #$3B
        BLS   DC70
        CMPA   #$43
        BLS   DC65
        CMPA   #$4B
        BLS   DC60
        CMPA   #$53
        BLS   DC55
        CMPA   #$5C
        BLS   DC50
        BRA   DC45

```

```

DC100 LDAB  #B8
      RTS
DC95  LDAB  #A0
      RTS
DC90  LDAB  #90
      RTS
DC85  LDAB  #88
      RTS
DC80  LDAB  #80
      RTS
DC75  LDAB  #70
      RTS
DC70  LDAB  #68
      RTS
DC65  LDAB  #58
      RTS
DC60  LDAB  #50
      RTS
DC55  LDAB  #48
      RTS
DC50  LDAB  #40
      RTS
DC45  LDAB  #35
      RTS

```

```
$INCLUDE 'HC11REG.H'
```

```

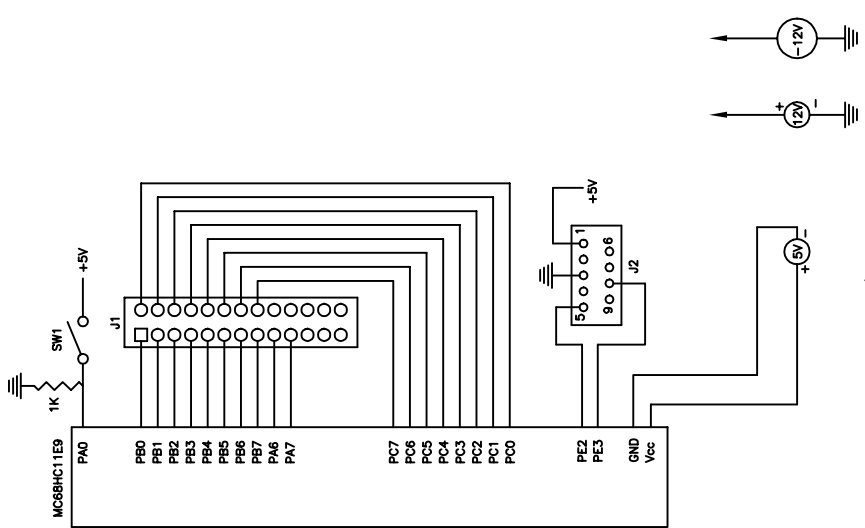
*****
* RAM data area
*****

```

	ORG	RAM
SCIDAT	RMB	6
LTVAL	RMB	1
RTVAL	RMB	1
LTDIR	RMB	1
RTDIR	RMB	1
LEFT	RMB	1
RIGHT	RMB	1
COUNT	RMB	1
PIGFRT	RMB	1
PIGREAR	RMB	1

REVISIONS		
ZONE	REV	DATE

DESCRIPTION	DATE	APPROVED



2	MC3102-0114	APITECH DRIVER MODULE	APITECH
2	OMI-SS-212D	12V DPDT RELAY	RADIO SHACK
2	J4	.200" VERT. TERMINAL BLOCKS	
2	J2	DB-9 CONN.- FEMALE	
2	J1,J3	10 PIN .100" MALE HEADER	
2	U3,U5	DAC8888 DIG.-TO-ANALOG CONV.	NATIONAL SEMI.
2	U2,U4	LM741 OP-AMP	NATIONAL SEMI.
	U1	MC68HC11EBU	MOTOROLA
		PART OR IDENT. NO.	
		MANUFACTURE DESCRIPTION	
			MATERIAL/ SPECIFICATION

FULL-SCALE MBC VALVE CONTROLLER

DATE: 7/9/99	DWG NO. LAVT017	REV
DRAWN BY: B. WELLS	SCALE 1:1	SHEET 1 OF 2

References

- 1 Motorola, Inc., "M68HC11EVBU, Universal Evaluation Board User's Manual," M68HC11EVBU/AD2, Motorola Literature Distribution Center, Phoenix, Arizona, 1992.
- 2 Motorola, Inc., "MC68HC11 Reference Manual," M68HC11RM/AD REV 3, Motorola Literature Distribution Center, Phoenix, Arizona, 1991.
- 3 Motorola, Inc., "MC68HC11 E Series Technical Data," M68HC11E/D REV 1, Motorola Literature Distribution Center, Phoenix, Arizona, 1995.
- 4 Motorola, Inc., "MC68HC11 E Series Programming Reference Guide," M68HC11ERG/AD, Motorola Literature Distribution Center, Phoenix, Arizona, 1991.
- 5 Spasov, Peter. "Microcontroller Technology: The 68HC11 – 2nd ed.," Prentice Hall, Ohio, 1996.
- 6 National Instruments Corp., "LabVIEW User Manual," Part number 320999B-01, Published by National Instruments Corp., Austin, Texas, 1998.
- 7 National Instruments Corp., "LabVIEW Online Reference," LabVIEW 5.0, National Instruments Corp., Austin, Texas, 1998.
- 8 SICK Optic Electronic. "LMS 200 Laser Measurement System: Technical Description," Germany.
- 9 SICK Optic Electronic. "LMS/LMI 400: Telegram Listing," Germany.

Vita

Bruce J. Wells was born in 1973 in a small Mediterranean fishing village south of Naples, Italy. After moving to Virginia, he spent his formative years fishing on the Chesapeake Bay, restoring classic cars and participating in soccer, wrestling and football. Attending VA Tech seemed a logical choice after graduation from high school because of the breadth of academic majors found at VA Tech and its reputation as a top party school. Although originally undecided, Bruce chose to enter the Department of Mechanical Engineering as a means to gain greater knowledge of building racecars. As a result, he became heavily involved with the VA Tech Formula SAE car project; a project in which students design, construct, test and compete in an open-wheeled, 'Formula-1' style racecar. After receiving a B.S.M.E. in May of 1995, Bruce went on to work as a rocket scientist for almost two years before deciding to return to graduate school. Since returning, he has become heavily involved with electronics and microprocessor-controlled gizmos, in addition to strengthening his mechanical design abilities. However, his dedication to graduate school had a large impact upon his fishing and outdoor adventures. Thus, Bruce will be looking forward to working near a large river on which he can kayak and fish until his heart is content.