UNIVERSITY OF MINNESOTA

CENTER FOR TRANSPORTATION STUDIES

RESEARCH REPORT

Development of Signal Operations Research Laboratory for Testing and Development of Advanced Control Strategies, Phase I

Eil Kwon
Sangho Kim
Dongsoo Kim
Enha Lee

CTS 02-06

Technical Report Documentation Page

1. Report No.	2.	3. Recipients Accession No.	
CTS 02-06			
4. Title and Subtitle DEVELOPMENT OF SIGNAL OPERATIONS RESEARCH LABORATORY FOR TESTING AND DEVELOPMENT OF ADVANCED CONTROL STRATEGIES, PHASE 1		5. Report Date July 2002 6.	
7. Author(s) Eil Kwon, Sangho Kim, Dongsoo Kim, Enha Lee		8. Performing Organization Report No.	
9. Performing Organization Name and Address Center for Transportation Studies 200 Transportation and Safety Bldg. 511 Washington Avenue SE Minneapolis, MN		10. Project/Task/Work Unit No. 11. Contract (C) or Grant (G) No.	
12. Sponsoring Organization Name and Address Minnesota Department of Transportation 395 John Ireland Boulevard Mail Stop 330 St. Paul, Minnesota 55155		13. Type of Report and Period Covered 14. Sponsoring Agency Code	
15. Supplementary Notes			

16. Abstract (Limit: 200 words)

A virtual intersection environment consisting of a new microscopic traffic simulator and a 2070 traffic controller was developed to provide a platform for a realistic pseudo real-time evaluation of intersection control strategies. The new simulator is based on an object-oriented modeling approach. In particular, the interface between the simulator and the traffic controller was developed using a commonly available digital input/output card and the Windows NT registry. Further, a signal converter was also developed to transform the format of the emulated detector data from the simulation model into the data format acceptable by the traffic controller. The implementation of the intersection control strategies into the 2070 controller was performed using the field I/O manager module provided by the City of Los Angeles DOT. The resulting Hardware-in-loop simulation system was applied to evaluate different control strategies for an intersection, i.e., pre-timed, actuated and adaptive methods.

17. Document Analysis/Descriptors	18. Availability Statement		
Traffic control, Traffic signals, Computer simulation	No restrictions. Document available from: National Technical Information Services, Springfield, Virginia 22161		
19. Security Class (this report)	20. Security Class (this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	184	

DEVELOPMENT OF SIGNAL OPERATIONS RESEARCH LABORATORY FOR TESTING AND DEVELOPMENT OF ADVANCED CONTROL STRATEGIES, PHASE I

Final Report

July 2002

Principal Investigator:
Eil Kwon, Ph.D.
Center for Transportation Studies
University of Minnesota

Graduate Research Assistants:
Sangho Kim, Dongsoo Kim and Enha Lee
Department of Computer Science
University of Minnesota

Center for Transportation Studies
University of Minnesota

CTS 02-06

ACKNOWLEDGEMENTS

This research was supported by the ITS Institute, Center for Transportation Studies, University of Minnesota. The authors deeply appreciate the cooperation from Mr. Verej Janoyan, the Department of Transportation, City of Los Angeles, who provided the interface modules for the advanced traffic controller. Also the technical guidance and cooperation from Dr. Taek Mu Kwon, University of Minnesota Duluth, in developing the intersection control modules with the advanced traffic controller was greatly appreciated. Finally the support from Dr. Max Donath, the director of the ITS Institute, was crucial in completing this research.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This report summarizes the results from the first phase of developing a signal operations research laboratory, which consists of an advanced traffic controller and a personal computer-based, microscopic traffic simulator. For this research, a new microscopic simulator based on an object-oriented approach was developed. The interface between the simulator and the traffic controller was developed using a commonly available digital input/output card and the Windows NT Registry. Further, a signal converter was also developed to transform the format of the output data (i.e., emulated detector signals) from the PC-traffic simulator into the data format that can be accepted by the 2070 traffic controller and vice versa. The implementation of the intersection control strategies into the 2070 controller was performed using the field I/O manager module provided by the Department of Transportation, City of Los Angeles. The open and flexible architecture of the advanced traffic controller makes it possible to implement different types of traffic control strategies directly into the controller. For an example application of the pseudo real-time evaluation system, an adaptive strategy based on a direct control approach with link-wide congestion index was implemented and its performance was compared with that of conventional control strategies. The capability to directly implement new control strategies into the traffic controller and to evaluate them in a realistic real-time simulation environment provides an important step in developing robust and efficient traffic control strategies.

Further, preliminary design of an event-driven microscopic simulator was conducted as the first step to develop more realistic driver behavior models and to examine the capability of the current event handlers of object-oriented compilers. In addition, preliminary work was also performed to develop an object-oriented, dynamic network analysis environment, where a given large network can be divided into multiple sub-networks for efficient evaluation of signal operation strategies in varying scopes. Such a dynamic network experimentation system can be a key component of the future signal operations research laboratory that can handle a large traffic network. Future study needs to expand the hardware-in-loop simulation system for a large traffic network by developing virtual traffic controllers that can emulate the detailed functionalities of the existing controllers. Continuing development of the event-driven microscopic simulator and an efficient network management system that can automatically generate input case files for traffic simulation models is also recommended.

1. INTRODUCTION

1.1 Background and research objectives

Testing and refining new operational strategies in a realistic traffic environment, prior to full-scale implementation, is of critical importance in developing efficient and robust real time traffic management strategies. While traditional simulation models have built-in signal control modules, those pre-programmed controller models can not simulate the detailed features provided by the new generation of advanced controllers, such as the 2070 controller. As a result, the impacts of advanced signal control strategies on the network performance cannot be accurately evaluated with the existing network simulation models, such as CORSIM.

To address the above needs, this research develops a signal operations research laboratory (SORL) where new advanced control concepts can be refined and tested with field controllers in a simulated environment prior to field implementation. The current research, Phase 1, develops a pseudo-real-time testing environment for a single intersection by integrating an advanced traffic controller and the newly developed microscopic traffic simulator. The specific objectives of Phase 1 include:

- Development of a comprehensive framework for the signal operations research laboratory (SORL),
- Development of a virtual-intersection SORL using a hardware-in-loop simulation with a new microscopic simulator and a 2070 traffic controller,
- Development of an interface between a simulator and a 2070 traffic controller for the virtual-intersection SORL, and
- Implementation of the new adaptive intersection control strategy into the 2070 traffic controller in the virtual-intersection SORL.

For this research, an advanced traffic controller, model 2070, was purchased and an interface was developed to convert the digital signals from a digital I/O card in a personal computer to a format that can be accepted by the 2070 controller. The interface also converts the signals from the 2070 controller to the personal computer that has the new microscopic traffic simulator. The resulting virtual intersection environment is used to refine and implement the adaptive control strategy developed in the previous research. Further, a preliminary design of an event-driven microscopic simulator was conducted as the first step to develop more realistic

driver behavior models and to examine the capability of the current event handlers of object-oriented compilers. In addition, a preliminary design is conducted to develop an object-oriented, dynamic network experimentation environment, where a given large network can be divided into multiple sub-networks for efficient evaluation of signal operation strategies in varying scopes. Such a dynamic network experimentation system is a key component of a comprehensive decision-support system for traffic network management.

1.2 Report Organization

Chapter 2 develops a detailed framework for the signal operations research laboratory, which includes a microscopic traffic simulator, a 2070 controller, and the interface between the controller and the PC-based traffic simulator. A detailed description of the microscopic traffic simulator is included in Chapter 3. Chapter 4 develops the controller-simulator interface including the new signal converter. The adaptive control algorithm developed in the previous research is implemented into the 2070 controller in Chapter 5 using the programming environment of the controller. Finally, Chapter 6 develops a preliminary design to develop the object-oriented network experimentation environment and Chapter 7 includes the conclusions and further research needs.

2. DEVELOPMENT OF AN ARCHITECTURE FOR SIGNAL OPERATIONS RESEARCH LABORATORY

2.1 Framework for single-intersection SORL

Developing efficient and robust strategies for traffic network management requires a realistic simulation environment, where alternative signal operational strategies can be directly implemented with field control devices or emulators. Such a pseudo-real time simulation allows detailed evaluation of proposed control strategies in terms of their operational capabilities and their effectiveness in managing congestion prior to field implementation. Figure 2-1 shows the framework of the signal operations research laboratory (SORL), which will be developed for a single intersection in the current phase of this research. Unlike other simulation models that have been used for evaluating traffic control strategies, the SORL will be equipped with an actual field control device that interacts with a real-time traffic simulator whose virtual detectors generate digital detection signals. By implementing alternative traffic control strategies directly into field controllers or through emulators, the feasibility and robustness of any proposed control methods can be evaluated in detail.

2.1 Overview of advanced traffic controller

The development of an advanced traffic controller (ATC) with multitasking capability was initiated by the California Department of Transportation and the City of Los Angeles Department of Transportation in 1992. The main objective of the ATC is to address the shortcomings of existing signal controllers, which are mainly single-purpose control devices. The current technical specifications have been developed by the Steering Committee under the auspices of the FHWA. It is a single, multitasking and multipurpose control equipment, whose hardware architecture was designed with a 'shared hardware manager' concept. The major components of the ATC hardware include multiprocessor design CPU with a Motorola 68360 processor, standard VME bus, asynchronous serial communication module, I/O module and modular plug-in power supply. The modules are coupled through a motherboard (1).

The ATC runs under the OS-9 operating system, which can operate multiple programs simultaneously in real-time. In particular, a series of standardized application interfaces are expected to be developed by the steering committee and adapted as a national standard to ensure the interchangeability, interoperability and ease of transportability of the software applications.

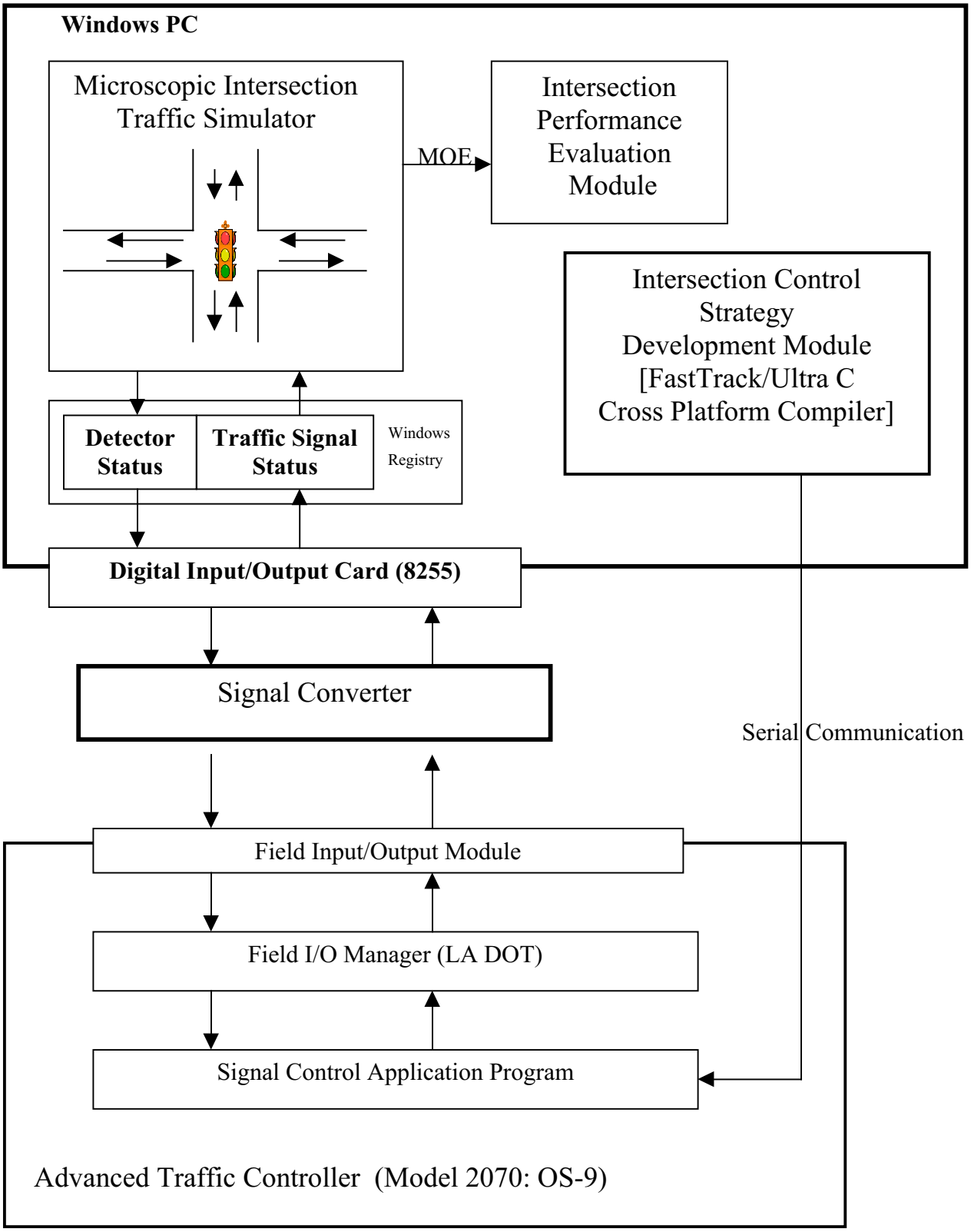


Figure 2-1 Framework of Signal Operations Research Laboratory

2.2 Architecture for signal operations research laboratory with ATC

As indicated in Figure 1, the SORL consists of four major parts that include a Windows PC with an 8255 digital I/O card (2), an advanced traffic controller (ATC) of model 2070, and a signal converter. The current version of ATC purchased for this research is equipped with the Motorola 68360 microprocessor running the OS-9 version 3.0 operating system. It is based on an open architecture that ensures compatibility with off-the-shelf products with standard VME interface modules. In this research, the application program interface provided by the City of Los Angeles Department of Transportation was used to develop the different types of intersection control software into the ATC.

The Windows-based PC has a microscopic traffic simulator for an intersection and the control strategy development module that has a cross-platform C compiler that operates under both the Windows NT and OS-9 operating systems. The intersection traffic simulator was developed in this research using the Visual Basic language that allows object-oriented design of the simulation package. It microscopically simulates the traffic behavior at an intersection responding to the changes in control signals and generates digital detection signals from all the detectors located at the sample intersection. The simulated digital detection signals go to the signal converter through the 8255 digital I/O card. The interface between the digital I/O card and the simulator was developed with the Windows Registry. The signal converter, which is also developed in this research, translates TTL level signals to open-collector outputs that can be handled by the ATC. The converted detector signals go to the field I/O module of the ATC, whose parallel I/O ports have 64 bits each for input and output. The application control program, compiled at the Windows-based PC and downloaded into the ATC through a serial connection, is written using the application program interface developed by the Los Angeles City Department of Transportation. The control application program in the ATC determines the status of the control signal heads with given detector data set and the resulting control signal head data is transferred to the intersection simulator through the signal converter.

The above interaction between the intersection simulator and the ATC continues on a real-time basis for a pre-determined time period. As can be noted, the ATC is used in exactly the same way as a field device, whose functionality and limitations can be directly reflected in developing signal control strategies in the SORL. Figure 2-2 shows the framework for the large-network-based SORL that will be developed in the subsequent phases of this research.

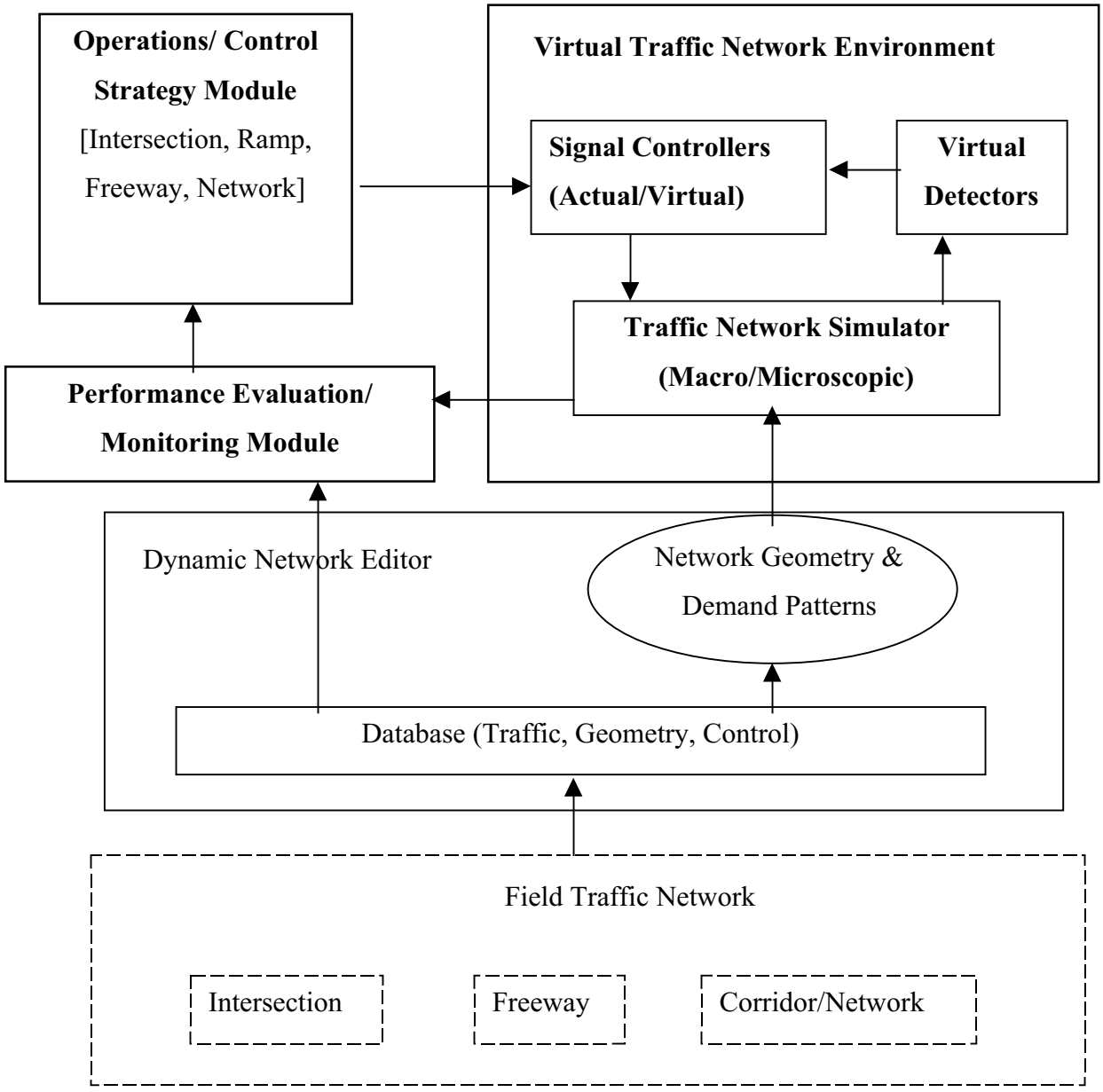


Figure 2-2. Framework for a large-network SORL

3. DEVELOPMENT OF A MICROSCOPIC INTERSECTION SIMULATOR

3.1 Overview of simulator structure

The intersection simulator in the signal operations research laboratory provides a pseudo-real-time environment where various types of control strategies can be evaluated under controlled traffic conditions. The simulator needs to be able to interact with the control device by generating on-line detector signals and receiving control signal settings on a real-time basis. Further, in addition to realistic representation of traffic behavior at an intersection, it should be able to model different types of current and future detection systems, such as GPS-based spatial detectors. In this research, the cellular-automata-based microscopic simulator developed in the previous research (3,4) was enhanced to represent more realistic behavior of vehicle movements at an intersection. In particular, the hopping behavior model was changed to a continuous moving behavior model with detailed lane-changing modeling of drivers approaching the stop-lines at a given intersection. The new simulator was designed with an object-oriented, modular approach using the Visual Basic programming language.

Figure 3-1 shows the structure of the simulator consisting of several major modules. It is based on a time-step simulation scheme, which updates the location and speed of each vehicle in a given roadway system every Δt second using a set of car-following and lane-changing models. The input data files for the simulator can be generated with Excel spreadsheet software, and the graphical output module shows the movement of each vehicle on the monitor screen as simulation progresses. With a time-variant demand pattern for each origin-destination pair in a given network, the vehicle generation module creates vehicles at each source node and assigns a specific path to each vehicle depending on its origin and destination. In the current version, one of three different types of probability distribution pattern can be selected to model the headway distribution of new vehicles generated at each source node, i.e., uniform, Poisson and negative exponential. Each vehicle makes its own decisions regarding lane changing depending on its current location, destination and surrounding traffic conditions. To model lane-changing behavior, an efficient sorting procedure is developed to find surrounding traffic conditions (i.e., locations of neighborhood vehicles and their types/speed values) for each vehicle. Further, each roadway is divided into three sections depending on the lane-changing possibilities, i.e., free, mandatory and no lane-changing areas.

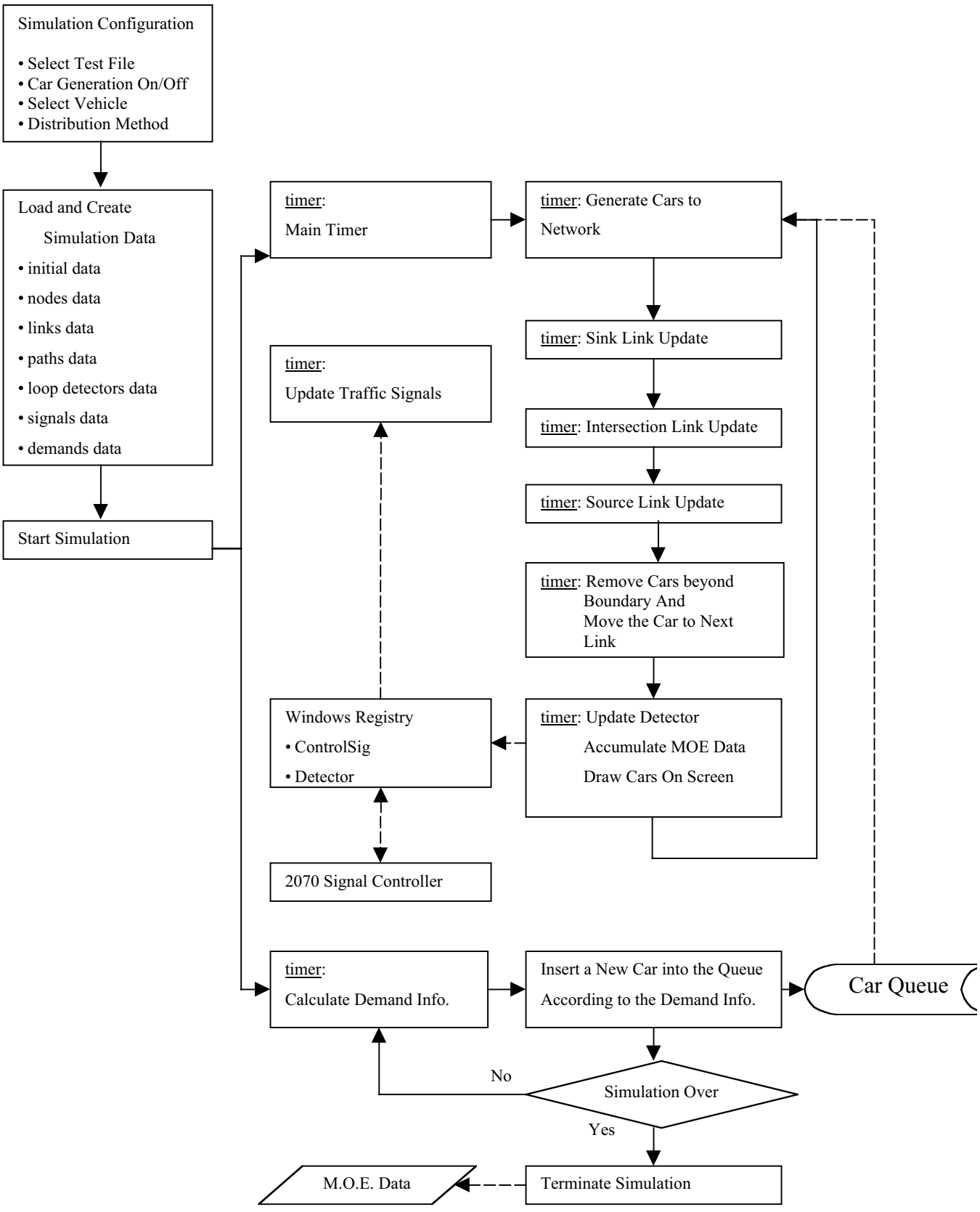
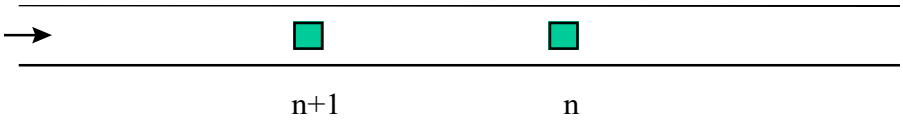


Figure 3-1. Structure and data flow of main simulation model

3.2 Enhancement of car-following model

In this research, the car-following model developed in the previous phase was enhanced to reflect continuous movement of vehicles and improve deceleration behavior. The new enhanced model updates the speed of each vehicle every Δt second as follows:



The speed of the vehicle, n+1, at the next time step, V^{t+1}_{n+1}, is determined as

$$V^{t+1}_{n+1} = V^{t}_{n+1} + \Delta v$$

To determine Δv, find the headway (space), D, from vehicle n+1 to the front vehicle, n;

If $D^t \geq D_{max}$, then $\Delta v = \alpha$,

where D_{max} = headway for maximum acceleration,

α = maximum acceleration per second.

Else, if $V^t_n \geq V^t_{n+1}$, then

$$\Delta v = \text{Min}\ [\ D^t * \alpha\ / D_{max},\ (V^t_n - V^t_{n+1})]$$

where $[V^t_{n+1} + \Delta v] <=$ Maximum Speed Limit

if $V^t_n < V^t_{n+1}$, then

$$\Delta v = -[(V^t_n - V^{t+1}_n) + (V^t_{n+1} - V^t_n)^2 * \Delta t/(D^t - D_{min})]$$

where D_{min} = minimum headway,

In the above procedure, the deceleration model is derived from the assumption that the headway changes between two vehicles, n and n+1, from time t to t+Δt is in proportion to the speed changes of the two vehicles during the same Δt interval. It can be noted that the new model features a small number of parameters with a simplified structure, and this allows efficient calibration with real traffic data.

3.3 Vehicle generation module

At each source node, the vehicle generation module creates new vehicles, i.e., determines the headway between two successive vehicles following a probability distribution selected by user. The time-variant, origin/destination-specific demand provided by the user are used as mean flows for the selected headway distribution, which includes the following three types:

1) Uniform distribution
2) Negative exponential distribution
3) Poisson distribution

The vehicle generation process for each node is as follows:

1) Get demand data for current time interval for each origin-destination pair.
2) Determine arrival time for next vehicle.
3) Specify destination and vehicle characteristics for new vehicle.
4) When simulation time reaches the arrival time, insert new vehicle onto the Vehicle Queue.
5) If the Vehicle Queue is empty, directly put the new vehicle onto a lane that has a space. Otherwise, wait until there is a space in a link.

3.4 Vehicle location update procedure

The basic process of updating location of each vehicle includes the following procedures. Starting from the first vehicle at downstream end of each link,

1) Find the vehicles in the neighborhood of a subjective vehicle,
2) Determine specific location and current speed level of neighborhood vehicles,
3) Determine moving range of the subjective vehicle considering its lane-change possibilities,
4) Find all the vehicles in the moving range,
5) Determine speed and new location of the subjective vehicle depending on its lane-changing options,
6) Move the subjective vehicle to the new location.

Figure 3-2 illustrates the flow chart of the vehicle updating procedure developed in this research. As indicated in this figure, one of the key requirements in updating location of a vehicle every Δt is an efficient procedure to search its neighborhood vehicles. In this research, an array-based

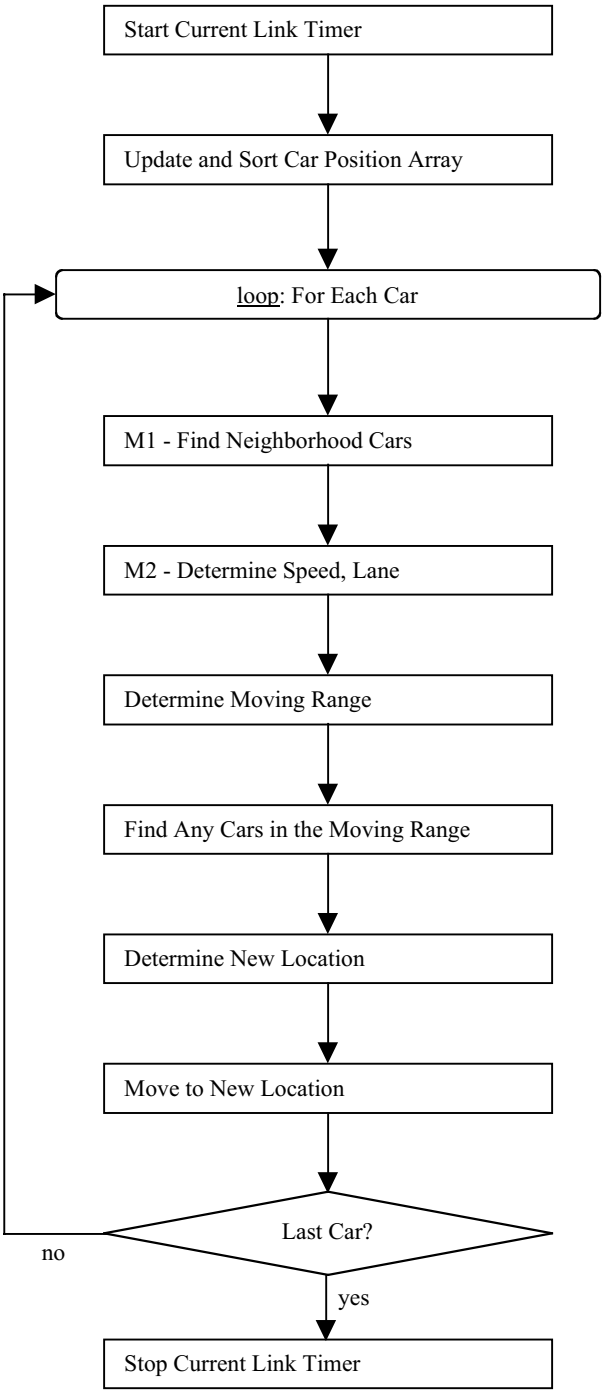


Figure 3-2 Vehicle location update procedure

search method was developed. First, all the vehicles in a link are put into the 'vehicle-position' array and sorted according to the distance of each vehicle from a stop-line. Next, starting from the first vehicle at the downstream end of each link, all the surrounding vehicles of each vehicle are found following the procedure illustrated in Figure 3-3. The next step is to make lane-changing decisions for each vehicle depending on its current location, destination and neighborhood vehicle information. After lane-change decisions are made, the new speed of each vehicle is determined by the car-following model with the current speed and headway information of the front vehicle in a destination lane. Finally the new location of each vehicle is determined with its updated speed and location of any vehicle in front.

Lane-changing model

The lane-changing model developed in this research is an enhanced version of the previous approach, which divides each link into three sections depending on the location from the stop-line, i.e., upstream, middle and downstream sections (3). Each section is assumed to have different lane-changing characteristics as follows:

Upstream section: Free lane-changing zone, where vehicles change lanes if possible.

Middle section: Mandatory lane-changing zone. Vehicles in a middle section change lanes depending on their destinations, e.g., left-turn vehicles change to left lane, while through vehicles change to either right or left lane depending on space availability.

Downstream section: No lane-changing zone. All the vehicles that need to change lanes complete lane-changing maneuver before they reach the downstream end of the middle section.

In this research, detailed lane-changing procedures were developed for vehicles in each section depending on their lane locations. Figure 3-4 shows the flow chart of the lane-changing model for the vehicles located in a middle lane of an upstream section. As indicated in the figure, the lane-changing model considers all the possible configurations in terms of neighborhood vehicle locations. It first determines the candidate destination lane for a vehicle depending on available space. If there is a vehicle behind the subject vehicle in the candidate destination lane, then the space headway between two vehicles is examined. If the headway is greater than the minimum headway pre-specified by user, then the subject vehicle can move to the destination lane. The new speed level of the subject vehicle is determined by applying the car-following model with the speed/headway information of a front vehicle in the destination lane. Flowcharts showing treatment of lane-changing behavior for other situations are included in Appendix A.

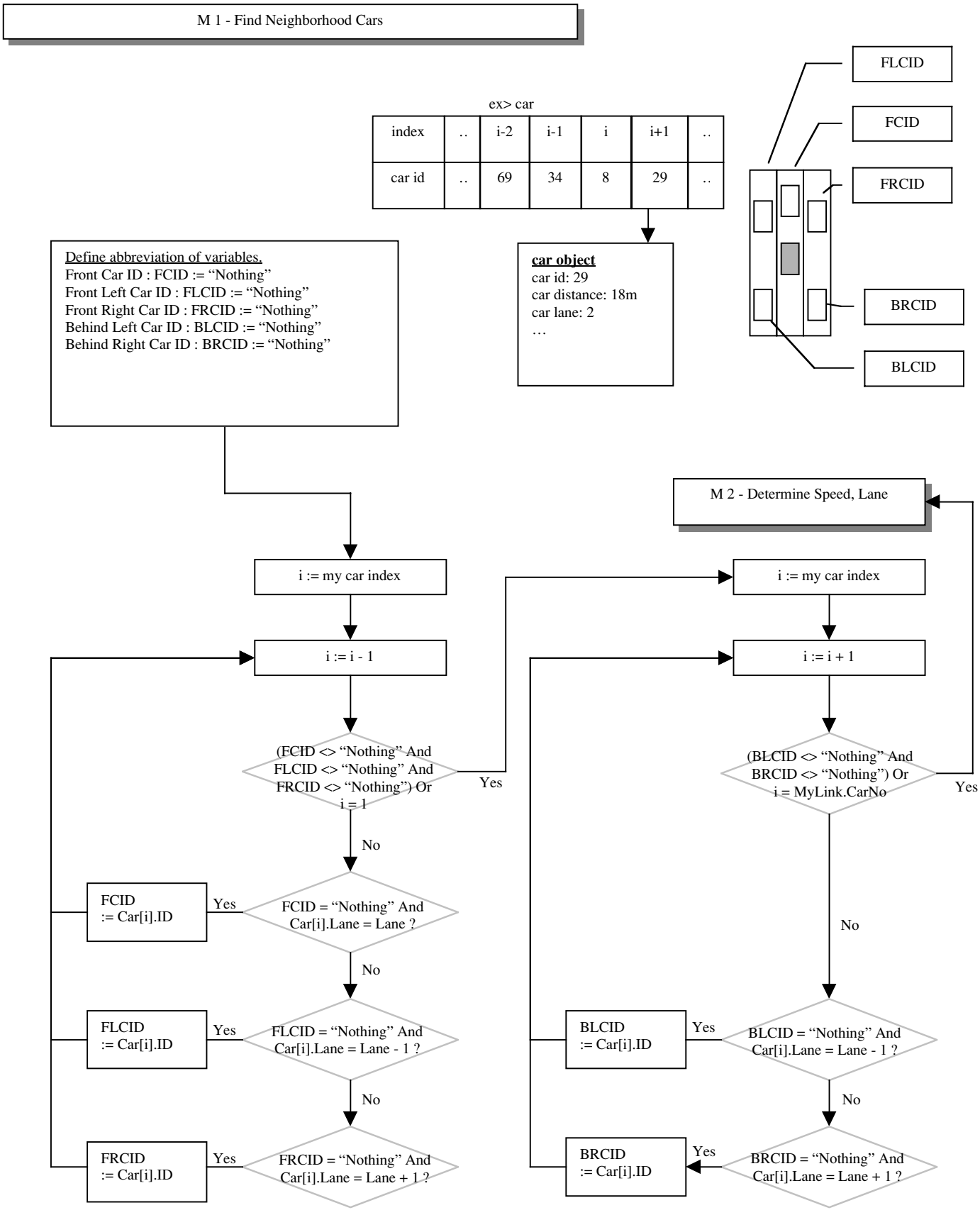


Figure 3-3 Procedure to find information of surrounding cars

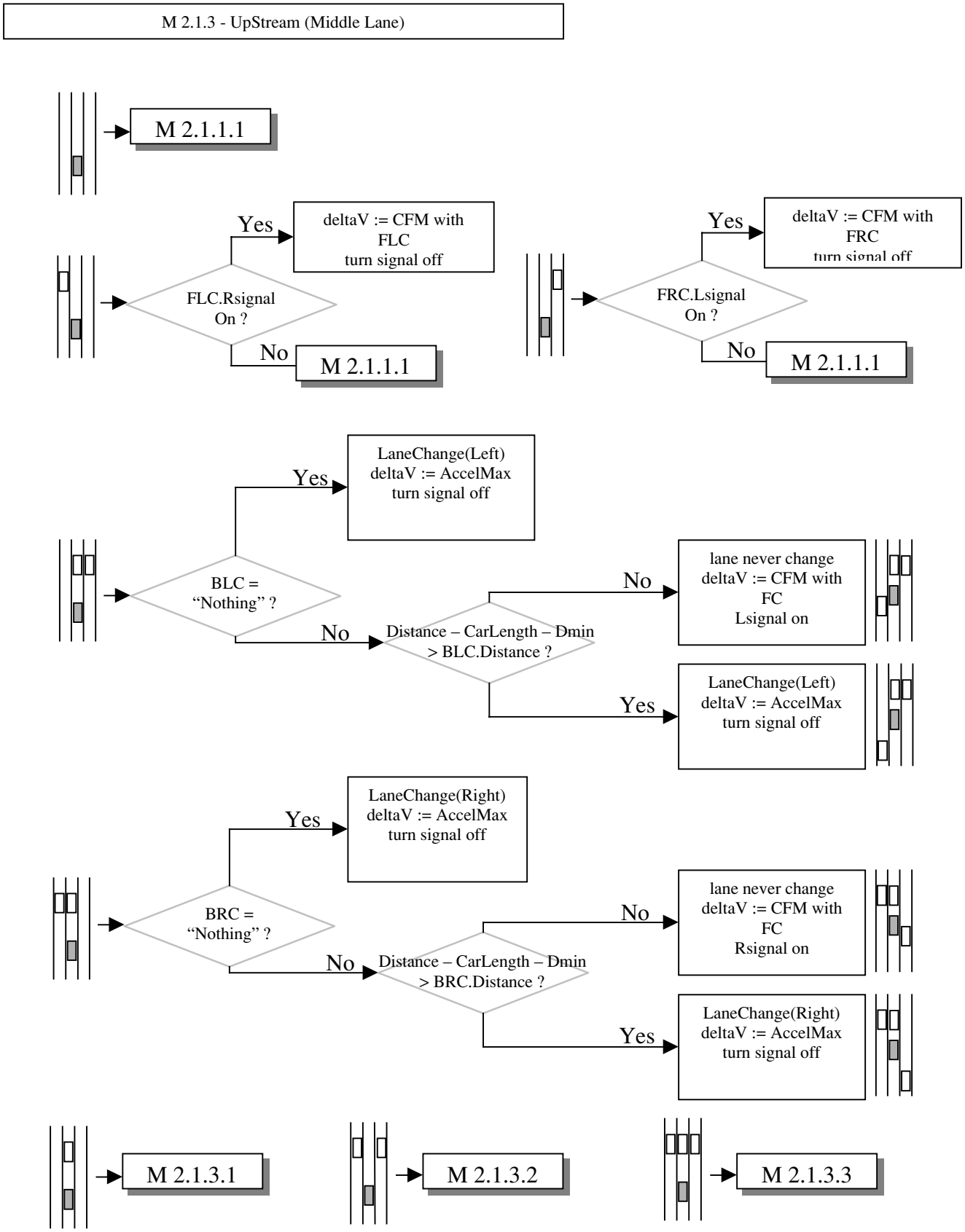


Figure 3-4 Lane-changing procedure for vehicles in upstream section

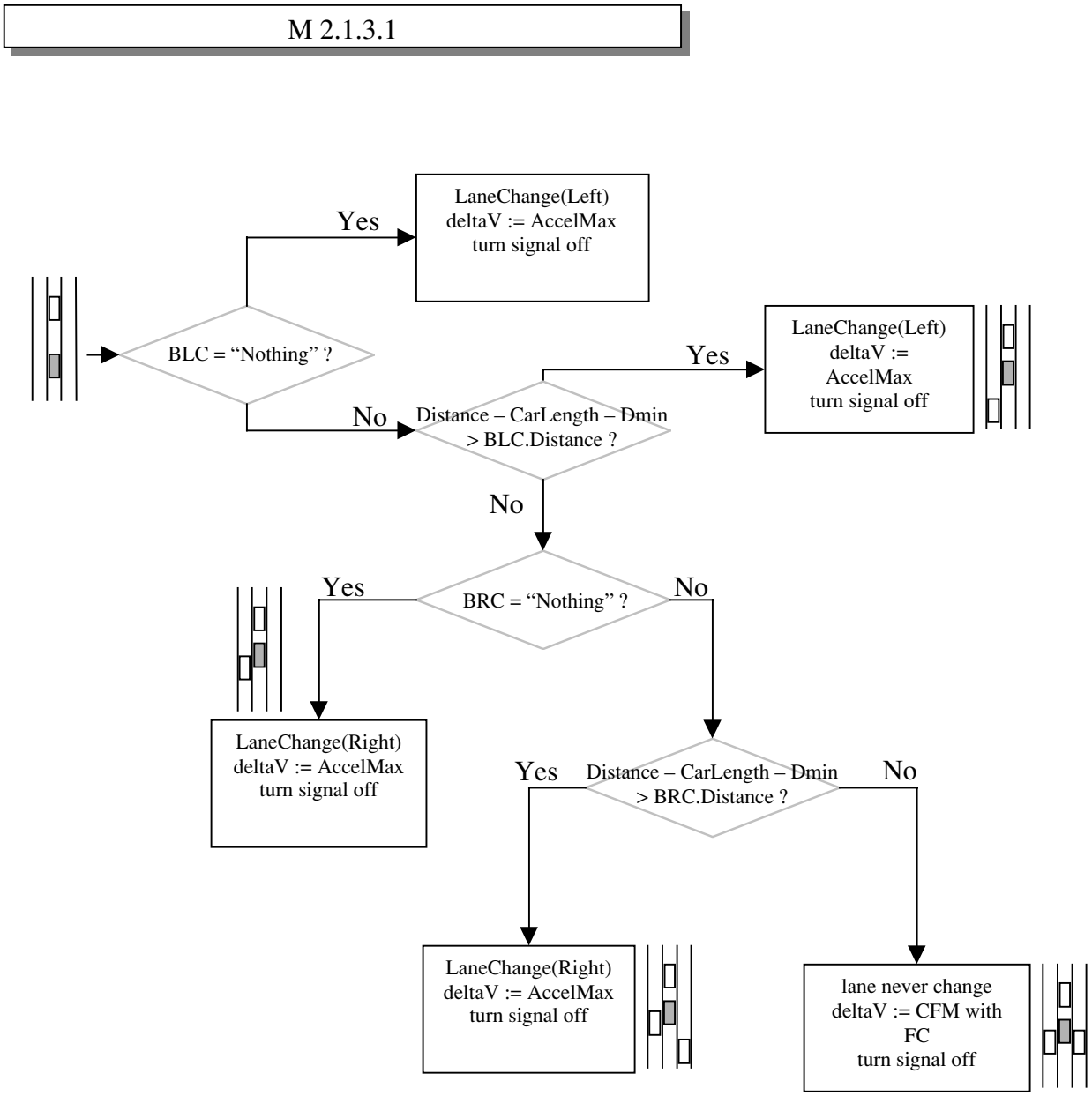


Figure 3-5 Lane-changing procedure for vehicles in upstream section (continued)

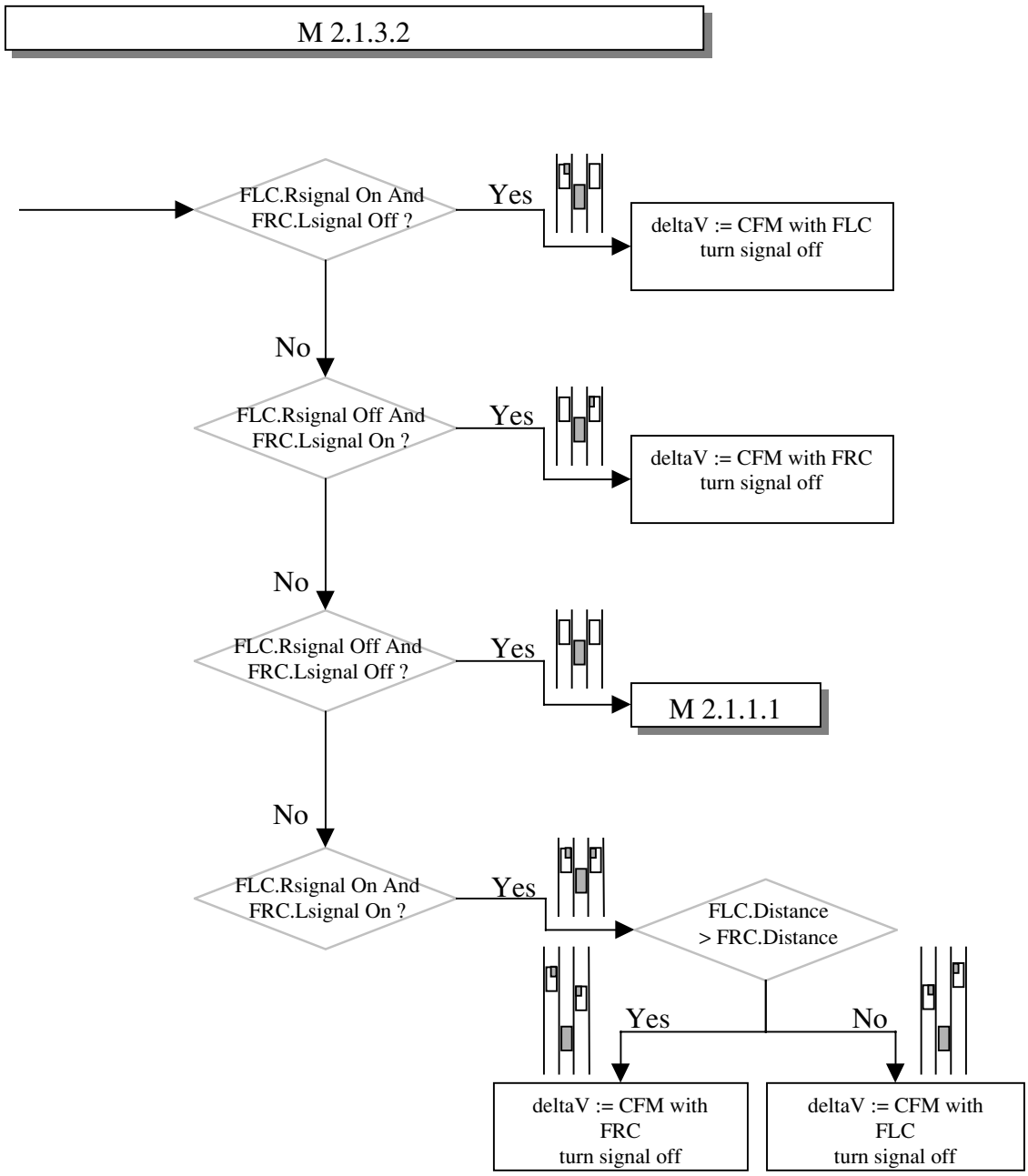


Figure 3-6 Lane-changing procedure for vehicles in upstream section (continued)

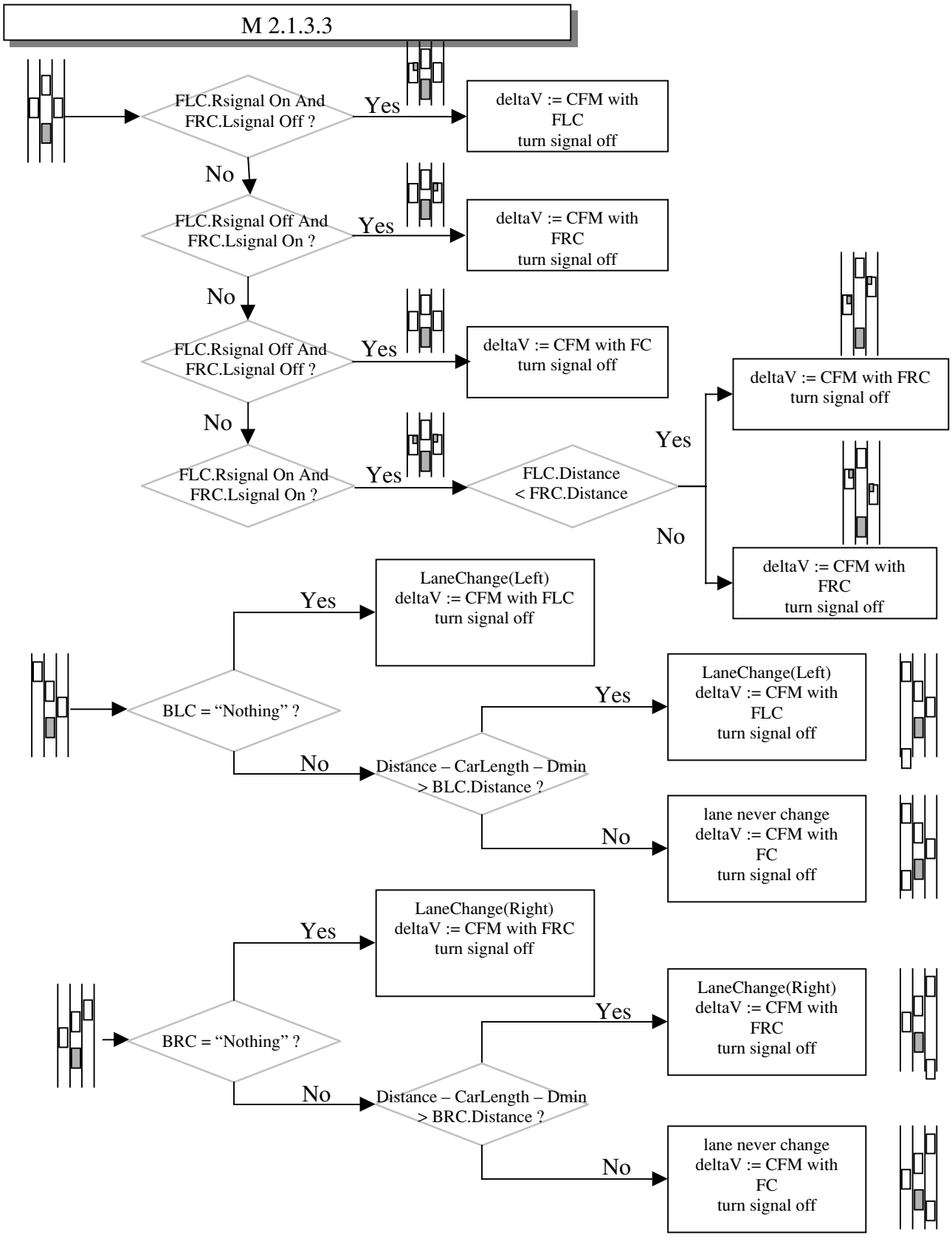


Figure 3-7 Lane-changing procedure for vehicles in upstream section (continued)

3.5 Modeling signal phases and traffic movements within an intersection

The movements of vehicles within an intersection proper were handled by creating sub-links within an intersection depending on the type of signal phase at each simulation time. For an unprotected left-turn, the model first checks if there is any vehicle in the conflicting path within the intersection proper. Further, any vehicle in the opposite link within a pre-specified distance is also checked before the left-turn vehicle is allowed to enter the intersection (Figure 3-8). In the case of a through vehicle, all the sub-links in the opposing left-turn link need to be cleared. Figure 3-9 also shows the types of signal phases modeled in the current version of the simulator.

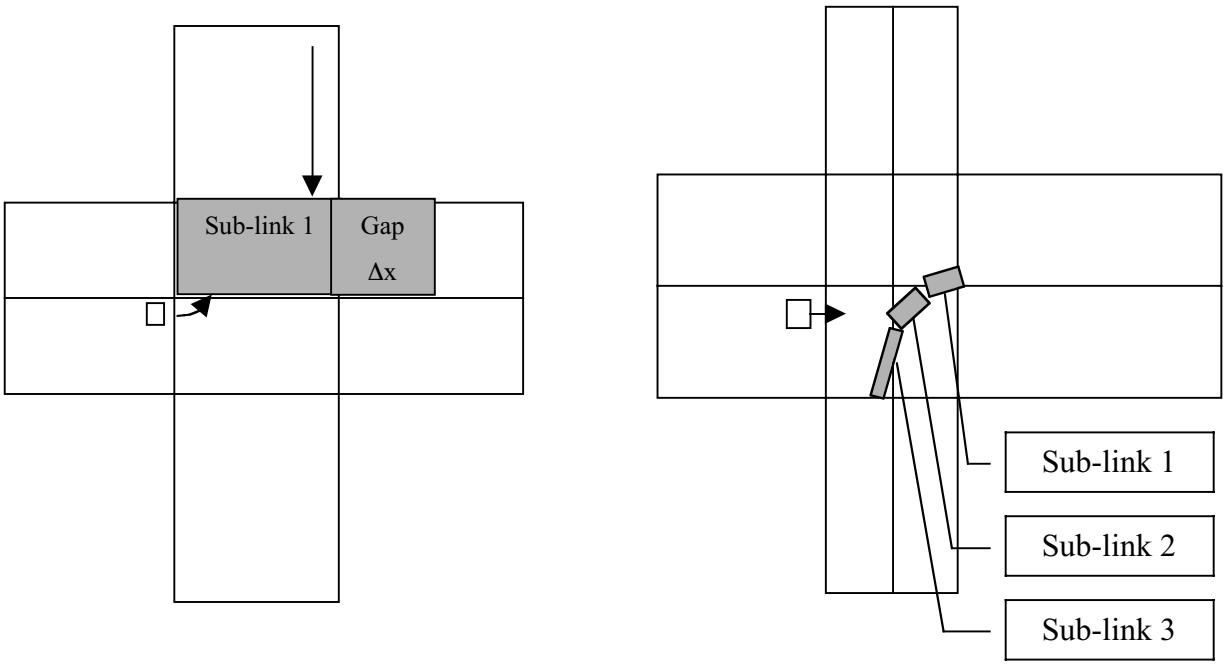


Figure 3-8 Sub-configuration of intersection proper for modeling vehicle movements

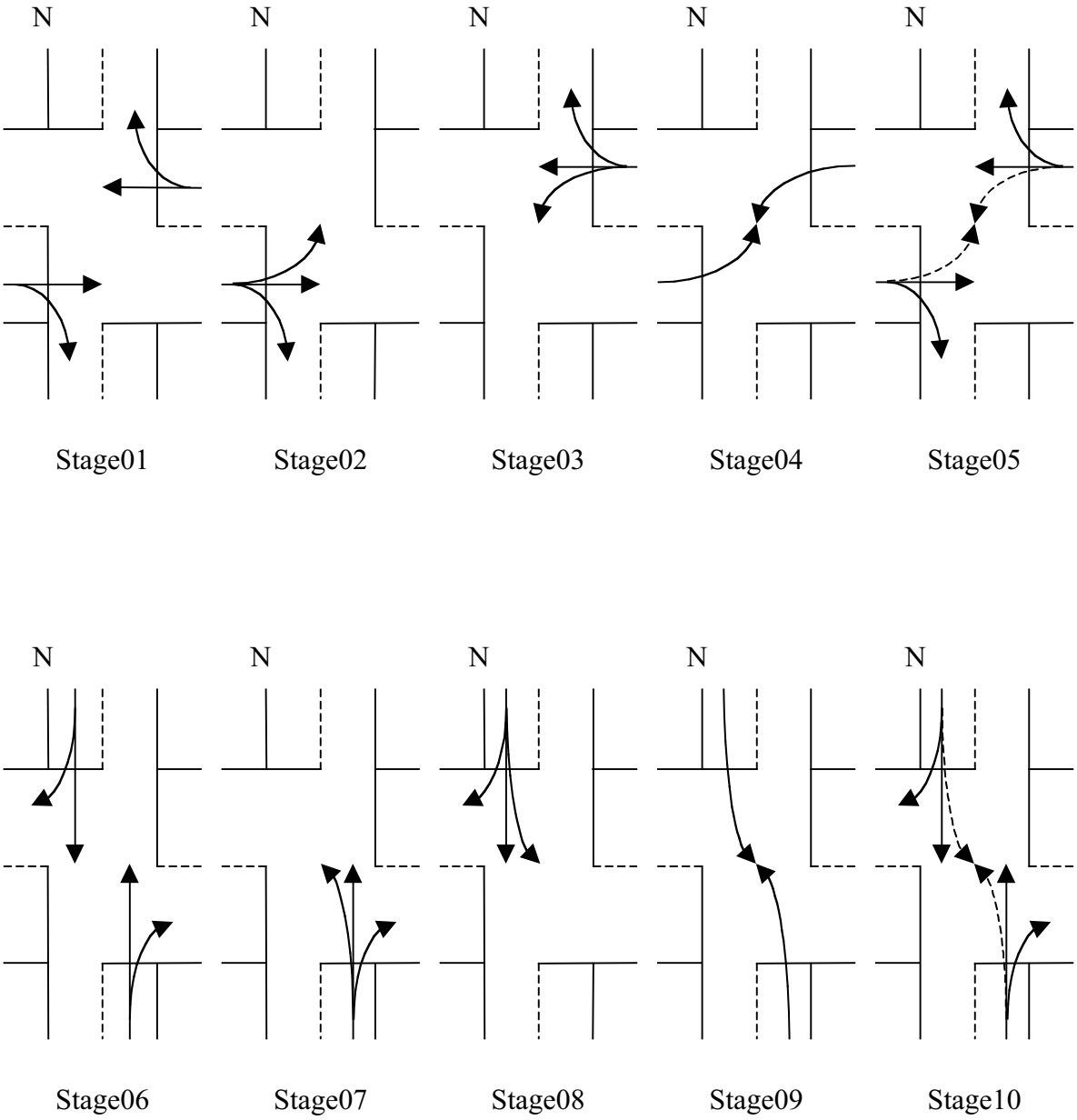


Figure 3-9 Phases modeled in the current simulator

3.6 Development of input/output user interface

The current version of the simulator is limited to a single intersection, but the structure of the model is designed to handle future expansions including multiple intersections and freeways. The types of the input data required for the current version include:

- Intersection geometry data: length, number of lanes, left-turn pocket length for each link
- Detector data: size and location of each loop detector,
- Control signal data: Signal phase sequence (for off-line simulation)
- Traffic demand data for each time interval for each origin/destination pair,
- Vehicle path data for each origin/destination pair.

The output data currently available includes the on-screen animation of vehicle movements and measurement of effectiveness (MOE) data, which consists of vehicle-hour data for each link through time. The development of detailed MOEs, such as link delay and travel time for each origin/destination pair, will be developed in future phases of this research. Figure 3-10 illustrates the output screen showing vehicle-animation with the main menu.

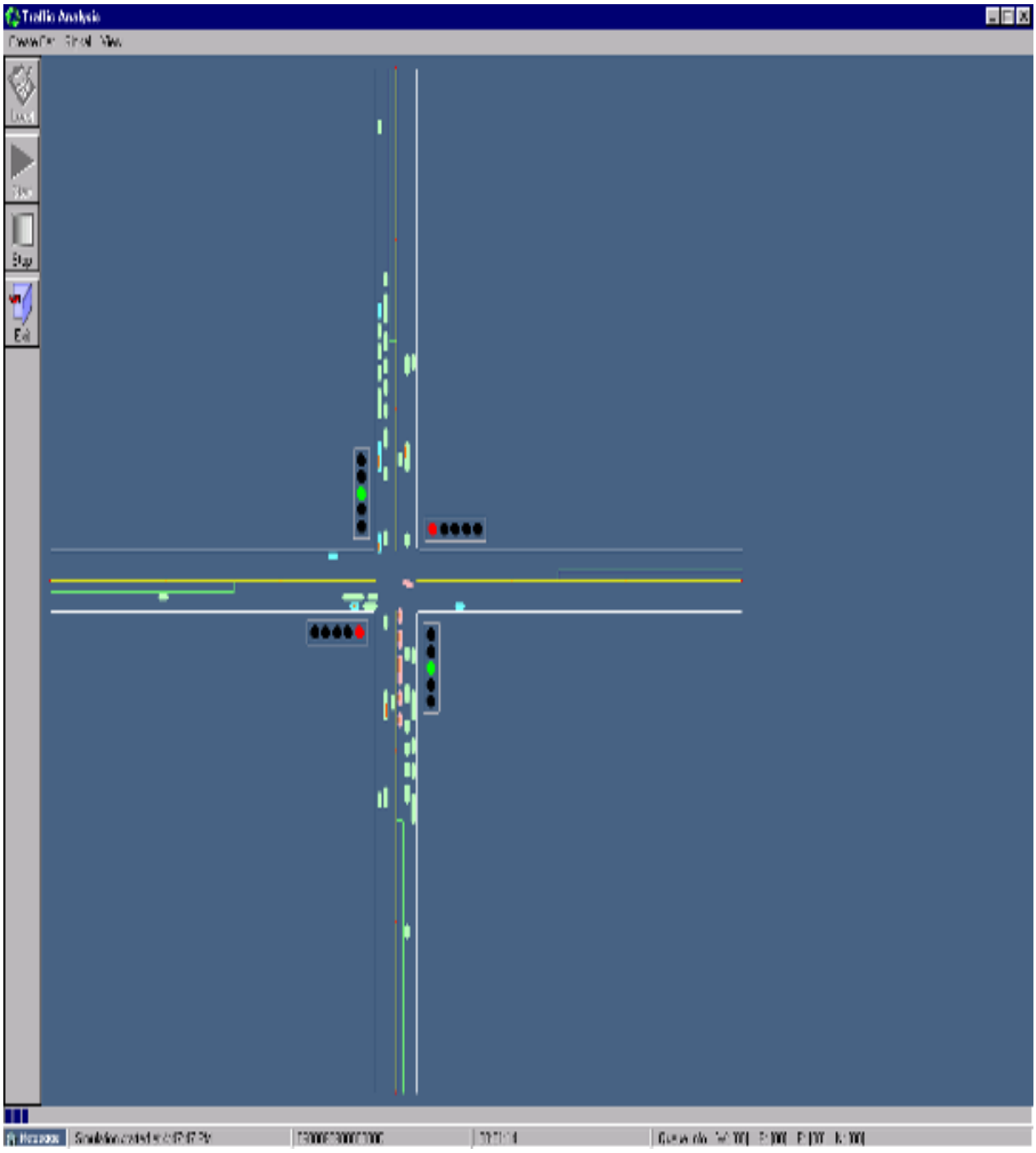


Figure 3-10 Example output screen

4. DEVELOPMENT OF AN INTERFACE BETWEEN TRAFFIC SIMULATOR AND 2070 CONTROLLER

4.1 Overview of interface architecture

The real-time interaction between the traffic simulator and the 2070 controller is the key component of the signal operations research laboratory. Figure 4.1 shows the simplified structure of the interfaces between simulator and controller. The simulated detector signal needs to go to the field data input module of the 2070 controller, whose output, i.e., the status of control signal heads, should be fed back to the traffic simulator in real time. In this research, a digital input/output card is installed into the Windows NT-based personal computer (PC) to send the simulated detector data out to the 2070 controller and also to receive the control signal head data from the controller. Further, a Signal Converter is developed to convert the transistor-transistor-logic (TTL) level signals from the digital I/O card to a format that can be accepted by the 2070 controller. The signal converter also converts the output signals from the 2070 controller to the TTL level signals for the digital I/O card in the PC. The interaction between the traffic simulator and the digital I/O card is handled by the software interface, which was developed by using the Windows Registry. The rest of this chapter describes the structure of the Windows Registry-based interface and the signal converter.

4.2 Interface between traffic simulator and digital I/O card

The digital input/output card selected for this research can be used with any 5V PCI-bus slot in IBM-compatible personal computers. According to the manufacturer's manual (2), the card supports 24 bits of parallel digital input/output on the PCI bus with three 8-bit ports, designated as A, B and C. Further, it contains a type 8255-5 programmable peripheral interface (PPI) chip, which can be programmed to accept inputs or to provide outputs to any of three ports. Each I/O line is buffered by a type 74LS245 tri-state buffer transceiver, which is configured automatically by hardware logic for input or output use according to the direction assignment from a control register in the PPI. The control register is a write-only, 8-bit register used to set the mode and the direction of the ports during initialization. Table 4.1 shows the bit assignment of the control register in the PPI. In this research, Port A was used to control the input/output process, Port B was used to read the data from the 2070 controller, and Port C was used to write the simulated

detector data generated from the traffic simulator. Figure 4.2 illustrates the configuration of the three ports.

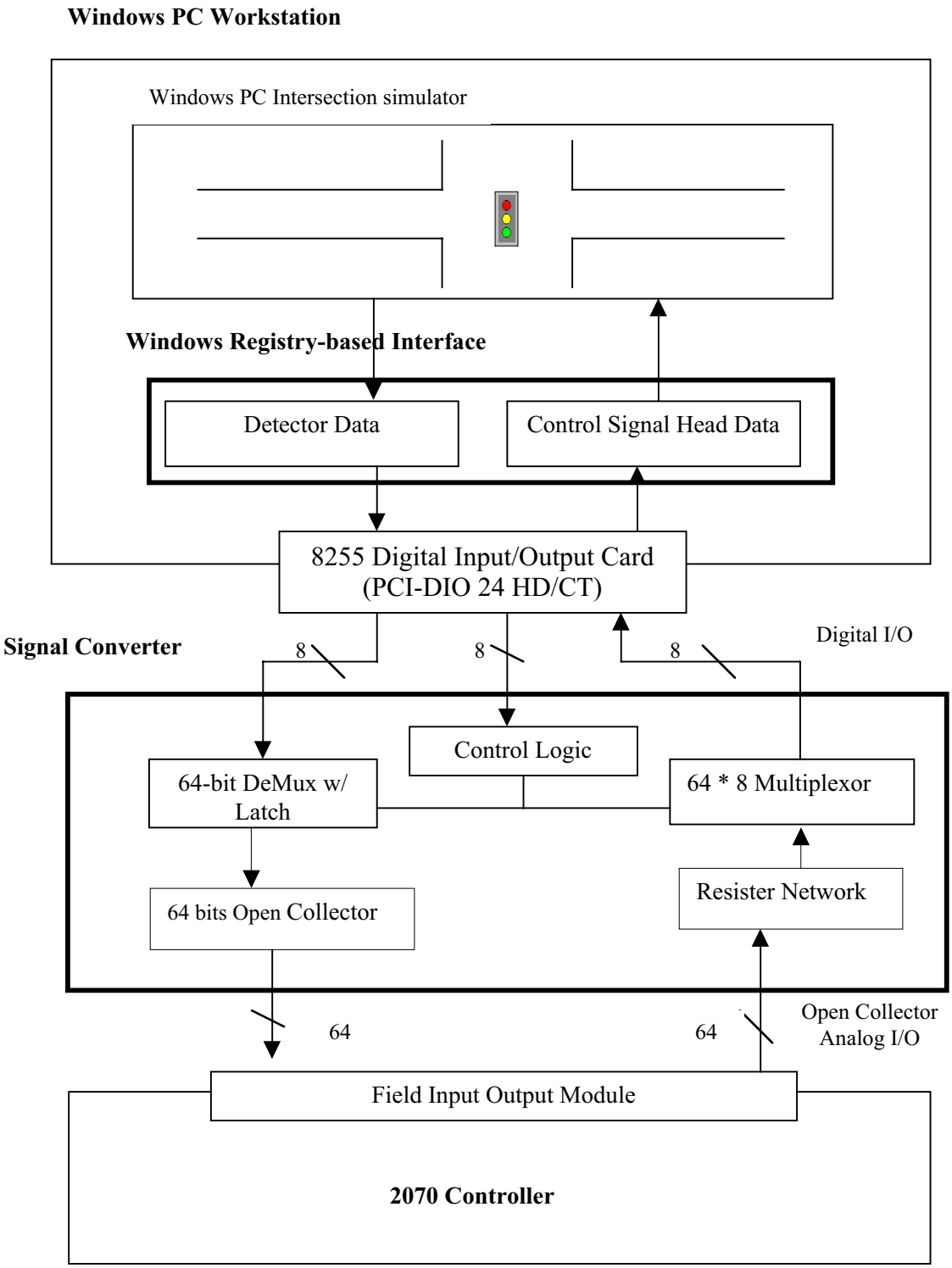


Figure 4.1 Interface structure between PC and 2070 controller

Table 4.1 Control Register Bit Assignment

Bit	Assignment	Code	Value Used
D0	Port C Lo (C0-C3)	1=Input, 0=Output	0
D1	Port B	1=Input, 0=Output	1
D2	Mode Select	1=Mode 1, 0=Mode 0	0
D3	Port C Hi (C4-C7)	1=Input, 0=Output	0
D4	Port A	1=Input, 0=Output	0
D5,D6	Mode Select	00=Mode 0, 01=Mode 1, 1X=Mode 2	00
D7	Mode Set Flag	1=Active	1

* Source: Product Manual (ICS, 1999)

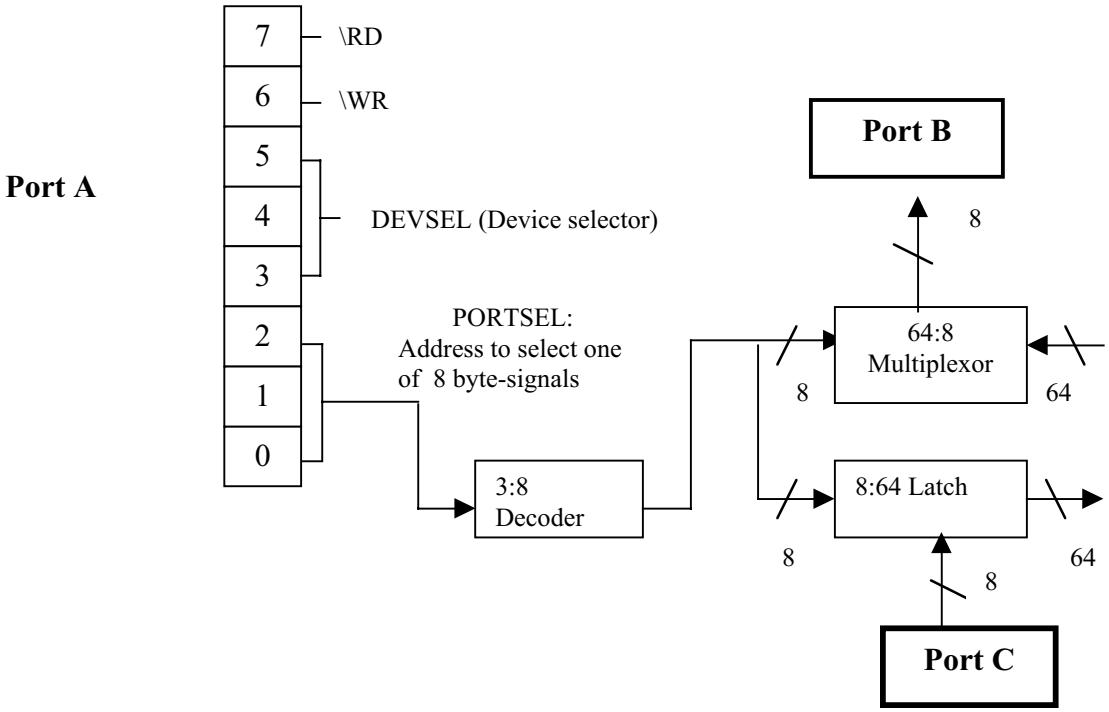


Figure 4.2 Port configuration for I/O card

The data input/output process through the digital I/O card is handled by the interface module, which uses two functions provided with the card. They are:

```
Private Declare Function
OutPortB Lib "ACCES32" Alias "VBOutPortB" (ByVal Port As Long, ByVal Value As
Byte) As Integer

Private Declare Function
InPortB Lib "ACCES32" Alias "VBInPortB" (ByVal Port As Long) As Integer
```

Figure 4.3 shows the flow chart of the interface module. First, the traffic simulator writes a set of the simulated detector data (i.e., 1 or 0 for each detector) every Δt to the designated place in the Windows Registry, which is a database containing the configuration information of a Windows-based PC. The detector data set is then read by the interface, which writes to Port C of the digital I/O card. The interface also reads the data from Port B, i.e., the control signal output from the 2070 controller, and writes them to the designated place in the Windows Registry. The control signal data stored in the Windows Registry is then retrieved by the traffic simulator, which continues simulation with the new set of control data. The data in the Windows Registry are updated every 2 ms by the interface module during simulation.

The implementation of the interface in Visual Basic to read from and write to the Registry was performed with two standard function calls:

GetSetting(*appname*, *section*, *key*[, *default*])

appname	Required. String expression containing the name of the application or project whose key setting is requested.
section	Required. String expression containing the name of the section where the key setting is found.
key	Required. String expression containing the name of the key setting to return.
default	Optional. Expression containing the value to return if no value is set in the key setting. If omitted, *default* is assumed to be a zero-length string ("").

Example> String1 = GetSetting(“TrafficSim”, “2070”, “Detector”, “Error”)

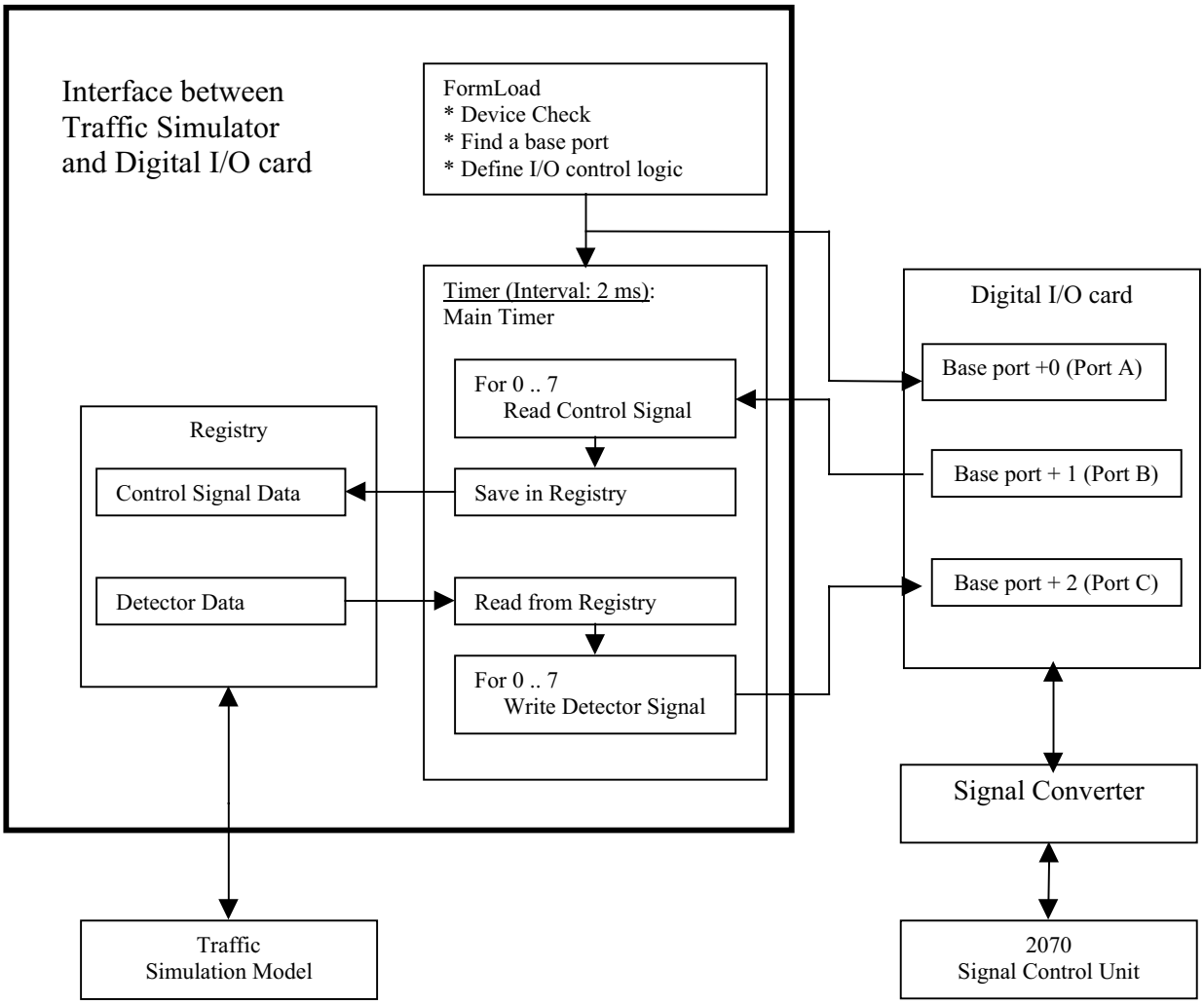


Figure 4.3 Structure of the Interface between Traffic Simulator and Digital I/O card

SaveSetting (*appname*, *section, key, setting)*

appname	Required. String expression containing the name of the application or project to which the setting applies.
key	Required. String expression containing the name of the key setting being saved.
setting	Required. Expression containing the value that *key* is being set to.

Example> SaveSetting "TrafficSim", "2070", "ControlSig", String2

*Real registry values are in the following place in the Registry:

My Computer → HKEY_CURRENT_USER → Software → VB and VBA Program Settings → TrafficSim → 2070

4.3 Interface between digital I/O card and 2070 controller

Figure 4.4 shows the schematic diagram of the signal converter that was developed in this research to connect the digital I/O card and the 2070 controller. The I/O card has 24 bits of parallel input/output ports, while the 2070 controller has 64 bits of input and 64 bits of output ports. As indicated in the figure, the key component of the signal converter is the Control Logic Unit (CLU), which controls the input/output process of the converter, i.e., the process of latching and multiplexing of inputs/outputs from/to the external devices. For example:

If DEVSEL (A3..A5) of Port A matches with RDS (Rear Device Select) and

if \RD is equal to 0, then \OE becomes 0. Otherwise, \OE = 1.

if \WR is equal to 0, then \LE becomes 0. Otherwise, \LE = 1.

The input signals from the I/O card, originating from the traffic simulator in the workstation, are temporarily stored in a latch and sent to the open-collector buffer, which translates the TTL level signals to open-collector outputs that can be handled by the 2070 controller. The process to latch the input signals is controlled by the control byte, i.e., A0-A2 of Port A of the digital I/O card. The 2070 controller generates open-collector outputs that are pulled up by the register network to convert the open-collector outputs to TTL level signals. The TTL level signals from the register network are multiplexed under the control of the digital I/O card. The signal converter also has front port selectors that can be used to monitor any 8-bit set of 64 detector or 64 control signal data. Further, by using the DEVSEL scheme, a single digital I/O card can control up to 8 2070 controllers.

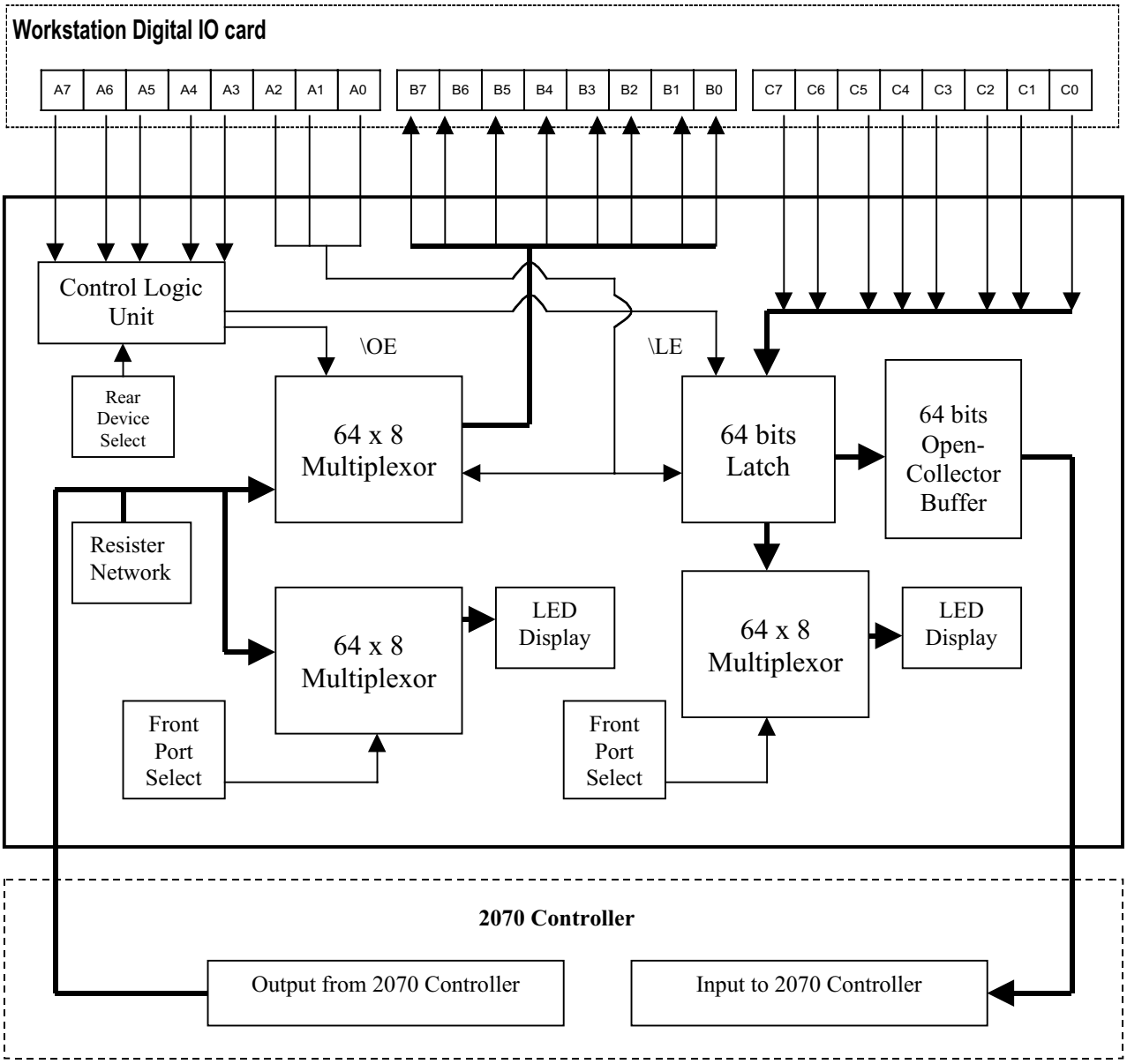


Figure 4.4 Schematic diagram of the signal converter

5. IMPLEMENTATION OF ADAPTIVE CONTROL STRATEGY IN SIGNAL OPERATIONS LABORATORY

5.1 Overview of adaptive control strategy

Effective management of urban traffic networks requires efficient control strategies that can respond to link-wide traffic conditions. While various traffic-responsive control methods have been developed for an intersection control, most existing strategies in operation use only spot measurements, either from stop-line or upstream boundary, that do not fully reflect the link-wide traffic conditions (5,6,7). While some methods employ complex traffic models to estimate link conditions as well as to formulate control functions (8,9), the inherent limitations in modeling and the difficulties in calibrating models for operational traffic conditions substantially limit the effectiveness of the model-based control.

In this research, the non-model-based, adaptive strategy developed in the previous study by the principal investigator is implemented in the signal operations research laboratory. The strategy reflects the link-wide traffic conditions without employing the complicated models of non-linear traffic dynamics. It directly quantifies the link-wide traffic conditions using the newly defined congestion index, which can be estimated with the data from traffic sensors, e.g., machine-vision detectors or conventional loops. The rest of this section summarizes the formulation of the congestion index and the adaptive control procedure.

Formulation of link-congestion index

For an example link, i, shown in Figure 5-1, the new link-congestion index developed in this research defines the level of congestion as

$$C_{i,t} = \sum \beta_{i,j} D_j$$

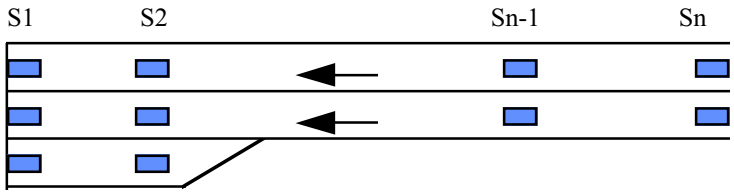


Figure 5-1. Example link (S: Detector location)

Where $C_{i,t}$ = Congestion index for approach or link i at the end of time interval t,

$\beta_{i,j}$ = Weighting parameter for detector location j at approach i,
j = 1, 2, - - n.

$D_j = (P_j + V_j)/(1 + V_j)$;

where P_j = 1.0; if detector j is occupied at the end of time interval t,
= 0.0; otherwise
V_j = number of vehicles crossing detector j during time interval t.

In the above formula:

$D_j = 0$; if $P_j = V_j = 0$; No traffic during the last time interval.
$0 < D_j < 1$; if $P_j = 0$ and $V_j \neq 0$; Congestion level depends on number of vehicles passed.
$D_j = 1$; if $P_j = 1$ and $V_j = 0$; Vehicle did not move during the last time interval.
$V_j \neq 0$; Vehicles passed but detector is still occupied.

As indicated above, the new congestion index quantifies the level of congestion on a 0 to 1 scale by combining occupancy information with the vehicle count data for each time interval. Further, by using different values of $\beta_{i,j}$, i.e., the weighting parameter for detector locations, the user can reflect specific link conditions or control objectives. Also, by adding the congestion indices of the active links in the current and next phases, the level of congestion for the current and next phase can be quantified.

Adaptive signal control procedure with congestion index

The adaptive control strategy implemented in this research uses the cumulative values of the congestion index estimated every one second for the current and next phases. Further, it adopts the simple structure of the stop-line actuated control, currently being operated at the laboratory intersection by the City of Minneapolis. The input to the control includes the phase sequence, minimum/maximum phase duration and extension intervals for each phase. After the minimum duration of the current phase expires, the algorithm compares the cumulative congestion index of the current phase with that of the next phase. If the next-phase congestion index is greater than or equal to the current-phase congestion level, then the control switches to the next phase; otherwise, the current phase is extended until the next extension check time. Figure 5-2 shows the simplified procedure of the new control strategy. Although the current version of the new control uses a fixed phase sequence and constant extension intervals, the use of the congestion index allows the possibility of introducing variable minimum/maximum phase duration and extension intervals depending on the level of congestion at each approach.

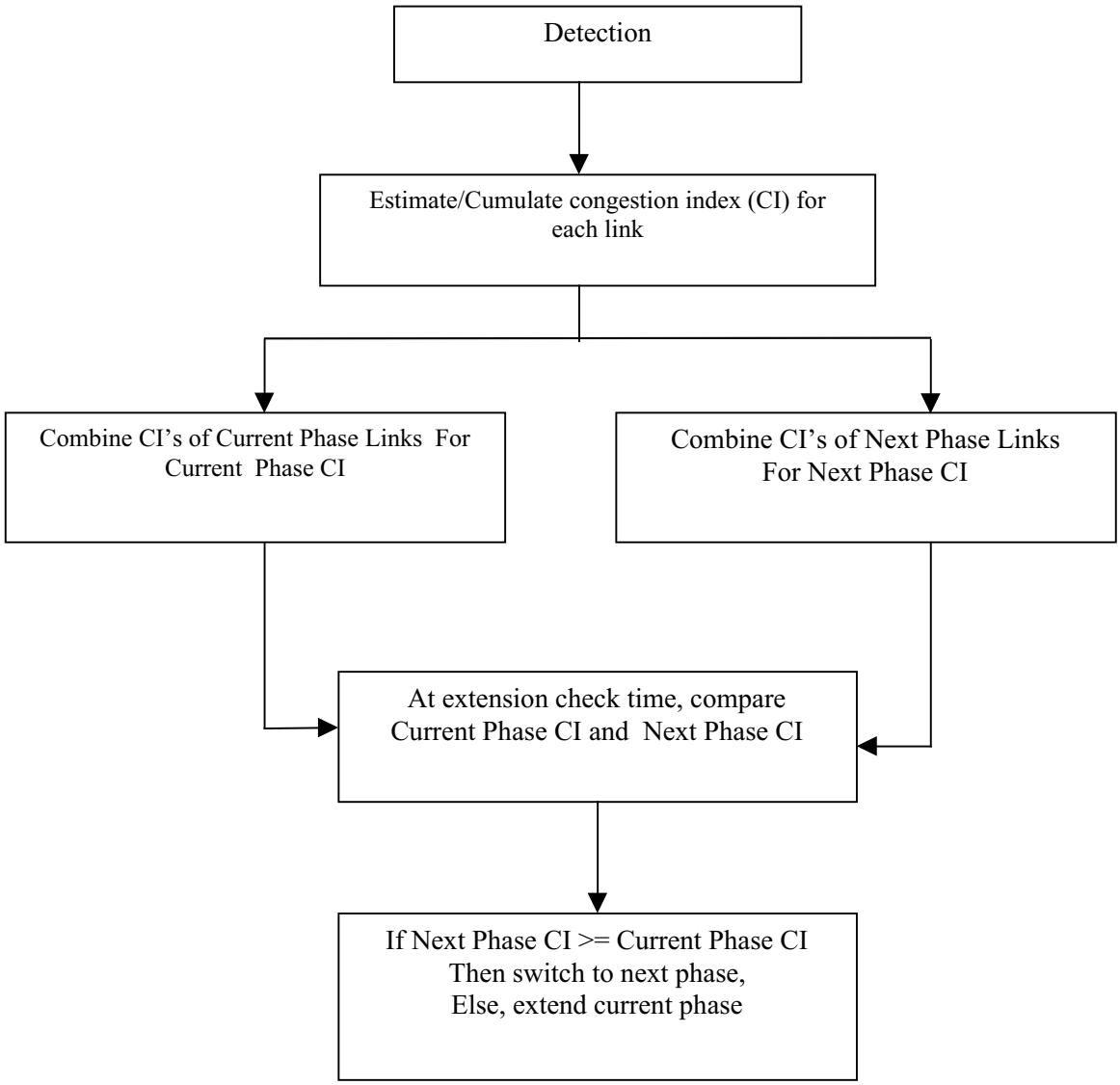


Figure 5-3 Procedure of Adaptive Control Strategy

5.2.1 Implementation and evaluation of adaptive strategy with signal operations laboratory

The adaptive control strategy described in the previous section was implemented and evaluated using the signal operations research laboratory developed in this research. The control algorithm was implemented into the 2070 controller and its performance was simulated and compared with those of the pre-timed and the stop-line actuated control strategies, currently being implemented by the City of Minneapolis, Minnesota. Figure 5-4 shows the schematic diagram of a sample intersection used in this simulation. The geometry of the sample intersection is a simplified version of the actual intersection located near downtown Minneapolis, which is currently controlled with the fully actuated method with stop-line detection using four signal phases as shown in Figure 5-4. For this experimentation, each approach is assumed to have same length and lane configuration. The volume data collected from the actual intersection over a one-hour period from 4 p.m. to 5 p.m. on a typical weekday was used as the input demand pattern for the simulation. Further, the cumulative index of total vehicle-hours for the whole intersection was used as the measure of effectiveness for the evaluation.

To compare the performance of each control strategy, the key control parameters (i.e., the phase sequence and the minimum/maximum duration of each phase) remained the same for all three methods. Further, the maximum duration of each phase was used for the pre-timed control. To implement the adaptive control strategy, three detector stations are placed for each link, i.e., stop-line, upstream boundary and the middle point, to estimate the congestion index. It should be noted that by changing the weights associated with the three locations, the congestion index can reflect different types of control. For example, stop-line actuated control can be emulated by setting the weight parameters of the middle and upstream detectors to zero. Figures 5-5 and 5-6 include the simulation results for three control strategies using a common set of demand data and phase sequence. In this simulation, vehicle generation at the upstream boundary of each approach is assumed to follow two different types of distribution, i.e., uniform and shifted exponential. As indicated in the figure, the adaptive control clearly shows improved performance over the other two conventional methods in reducing total vehicle-hours during the peak period for the same amount of traffic demand. In terms of cumulative vehicle-hours over a one-hour period with uniform distribution, adaptive control resulted in 17% fewer vehicle-hours than actuated control and 23% fewer than pre-timed signal operations.

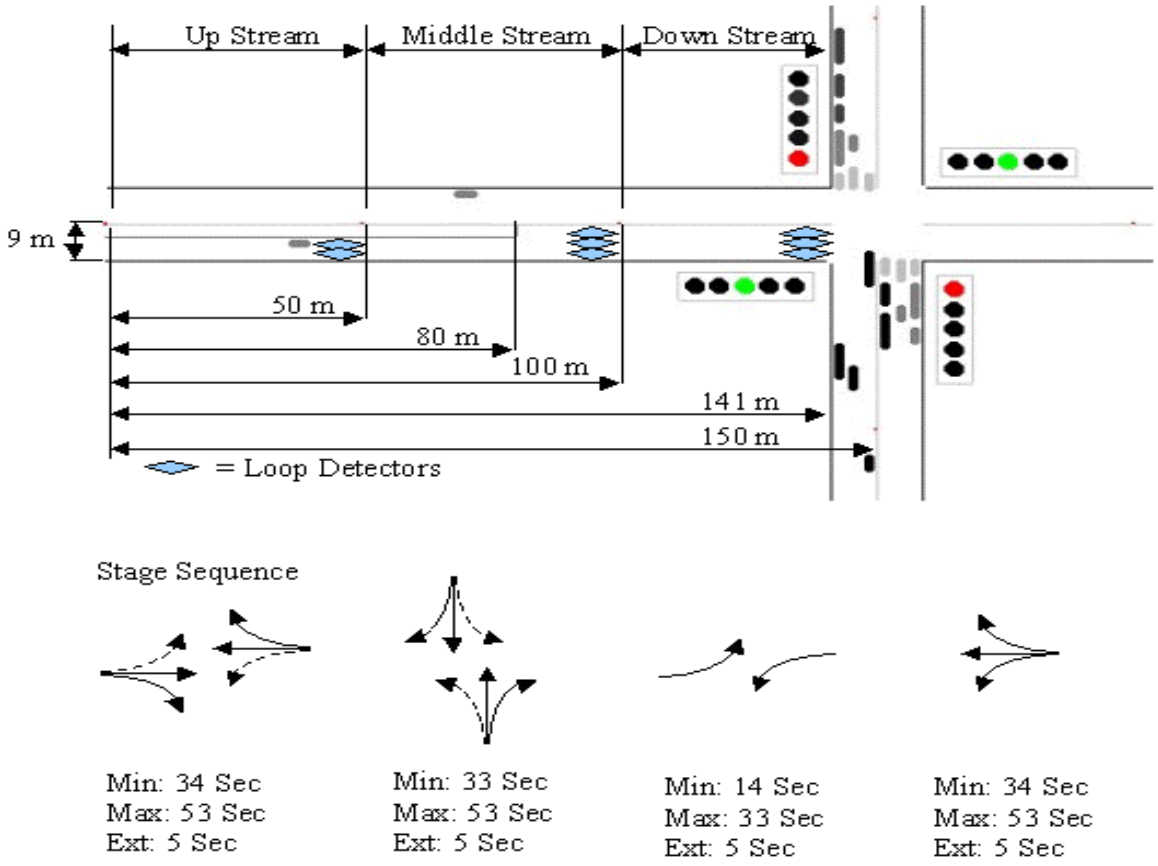


Figure 5-4. Sample intersection

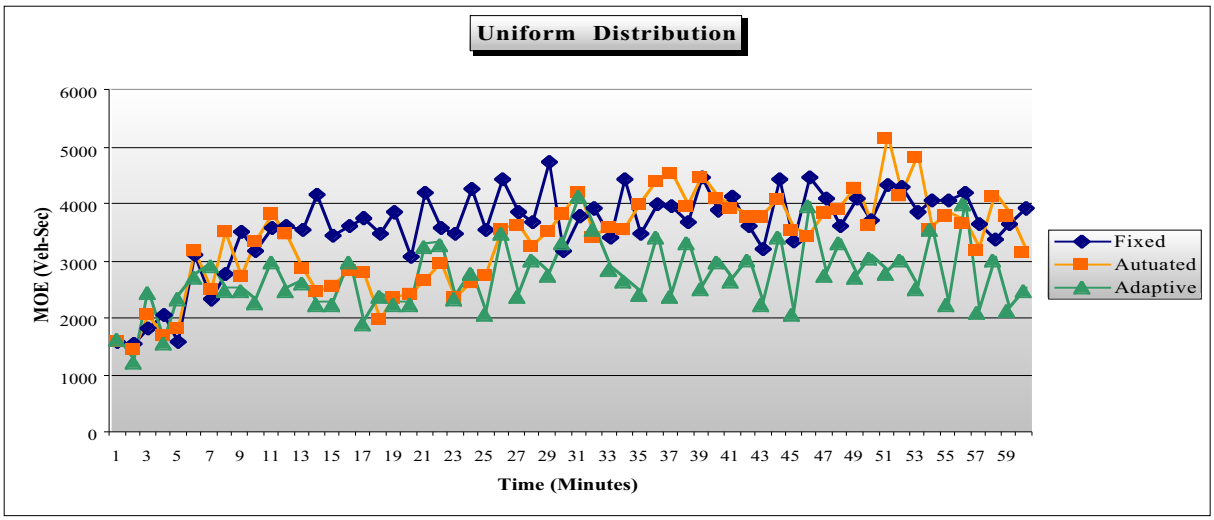


Figure 5-5a Simulation results through time with uniform distribution

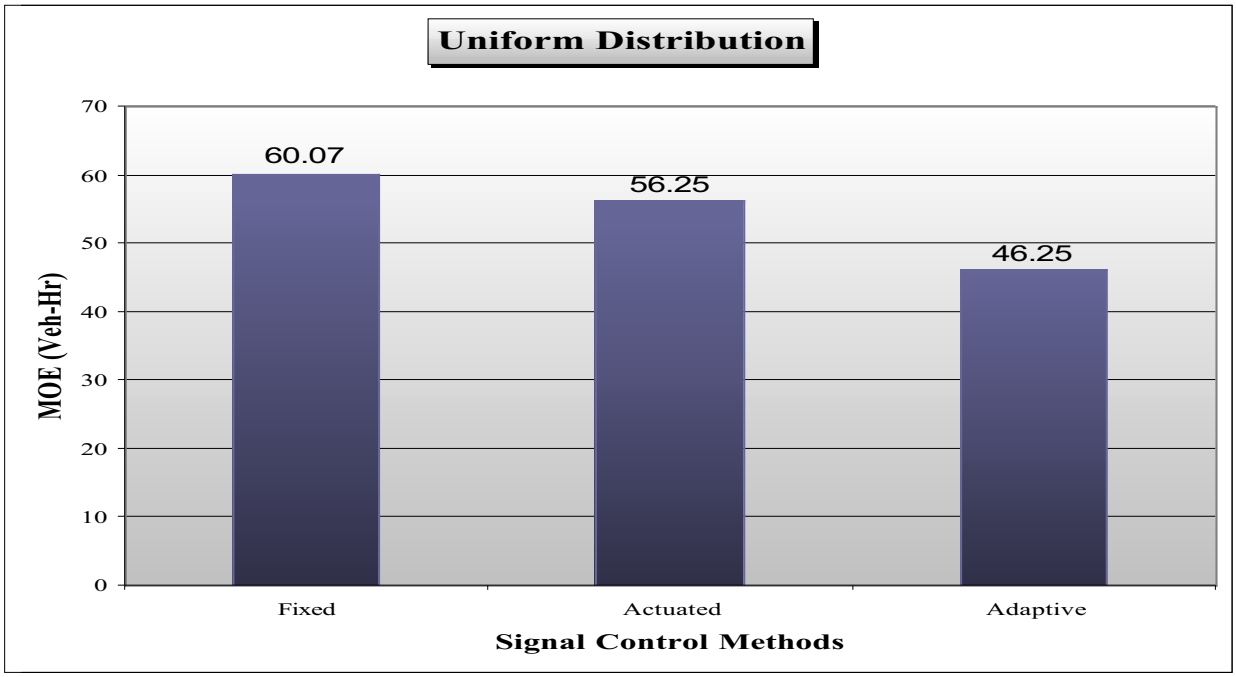


Figure 5-5b Cumulative results for different control methods

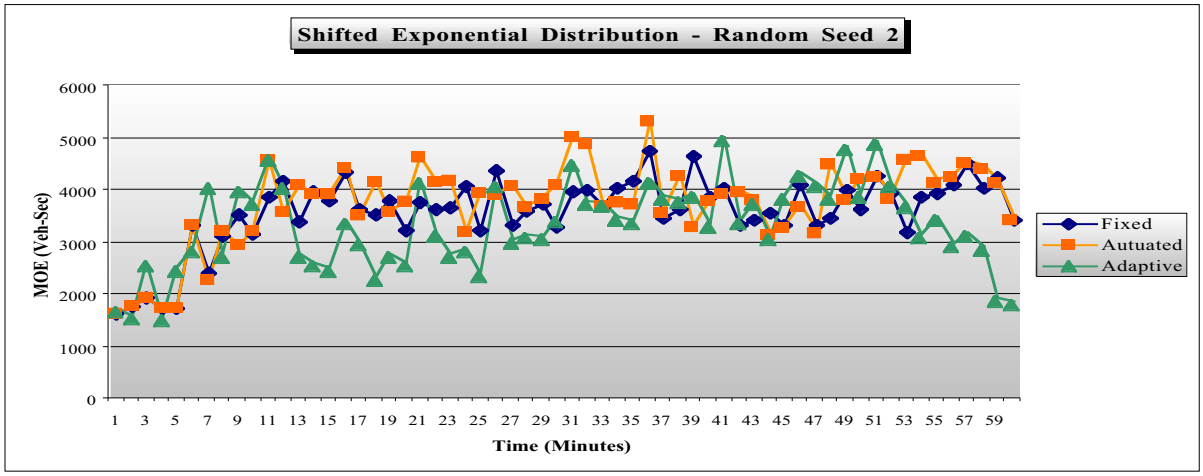


Figure 5-6a Simulation results through time with shifted exponential distribution

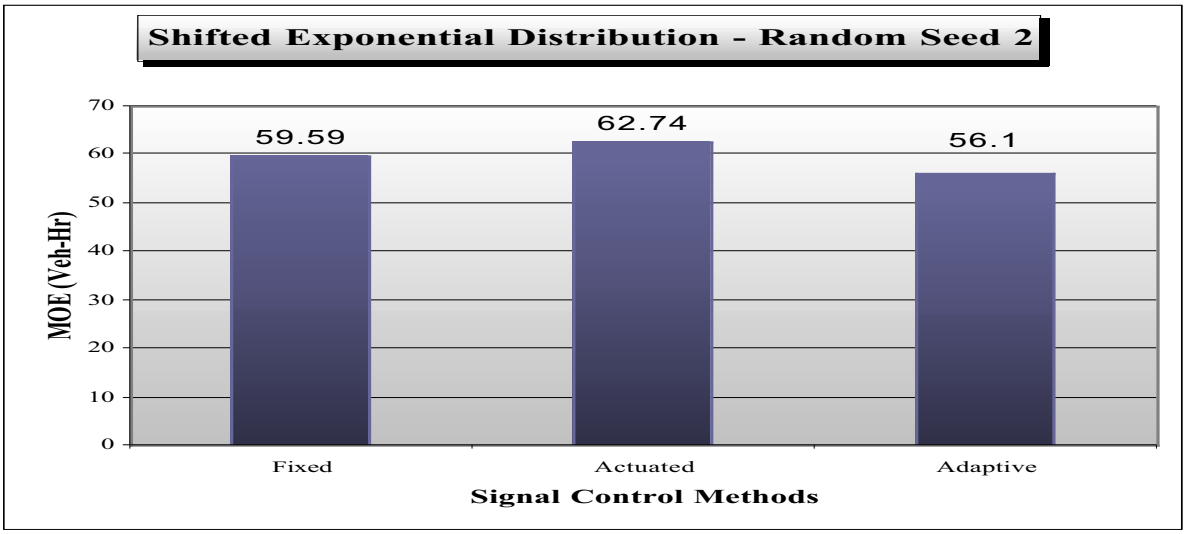


Figure 5-6b Cumulative simulation results with shifted distribution for different control methods

6. CONCLUSIONS AND FUTURE RESEARCH NEEDS

A realistic off-line environment for evaluating traffic control strategies is of critical importance in developing robust operational algorithms for signal systems. In this research, as the first step to develop the signal operations research laboratory, a pseudo-real-time evaluation system for intersection control strategies was developed by combining an advanced traffic controller (ATC) model 2070 and a personal-computer-based traffic simulator. For this research, a microscopic simulation model and a signal converter, which links the controller to the PC-based simulator, were developed. Using the new hardware-in-loop simulation system, an adaptive control strategy was evaluated and compared with traditional control methods in a virtual intersection environment. The major accomplishments of the current phase include:

- Development of a comprehensive framework for signal operations research laboratory (SORL) with a hardware-in-loop simulation architecture,
- Development of a new microscopic traffic simulator for an intersection with Visual Basic,
- Installation of the 2070 advanced traffic controller and a digital input/output card for hardware-in-loop simulation,
- Development of an efficient interface structure between the PC-based traffic simulator and the 2070 traffic controller,
- Development of a Windows Registry-based interface to facilitate data exchange between the traffic simulation model and the external signal converter through the digital I/O card,
- Development of the signal converter that changes the output data, (emulated detector signals) from the PC traffic simulator into a data format that can be accepted by the 2070 traffic controller and vice versa, and
- Refinement and implementation of fixed, fully actuated, and adaptive intersection control strategies into the 2070 controller using the Ultra C cross-platform compiler.

The example simulation of the intersection control methods using a sample intersection demonstrated the capability of the hardware-in-loop simulation system in implementing new control strategies into the traffic controller and evaluating them in a pseudo-real-time simulation environment. Further, preliminary design of an event-driven microscopic simulator was conducted as the first step to develop more realistic driver behavior models and to examine the capability of the current event handlers of object-oriented compilers. In addition, preliminary

work was also performed to develop an object-oriented, dynamic network analysis environment, where a given large network can be divided into multiple sub-networks for efficient evaluation of signal operation strategies in varying scopes. Such a dynamic network experimentation system can be a key component of the future signal operations research laboratory that can handle a large traffic network.

Future study needs to expand the hardware-in-loop simulation system for a large traffic network by developing virtual traffic controllers that can emulate the detailed functionalities of existing controllers. Continuing development of the event-driven microscopic simulator and an efficient network management system that can automatically generate input case files for traffic simulation models is also recommended.

References

[1] Automatic Signal/Eagle Signal, "Model 2070 ATMS controller unit User's Manual", December, 1996.

[2] Industrial Computer Source, "PCDIOB Series product manual", 1998.

[3] Kwon, E., Stephanedes, Y.J. and Liu, X., "Development and application of on-line strategies for optimal intersection control, Phase III", Final Report for Minnesota Department of Transportation, 1996.

[4] Kwon, E., and Stephanedes, Y., "Development of an adaptive control strategy in a live intersection laboratory", Transportation Research Record 1634, National Research Council, pp. 123-129, Washington, D.C., 1998.

[5] Staunton, M., "Vehicle actuated signal controls for isolated intersection", 1976.

[6] Bretherton, R.D. and Bowen, G.T. "Recent enhancements to SCOOT-SCOOT Version 2.4", IEE: Third International Conference on Road Traffic Control, 1-3, pp 95-98, 1990.

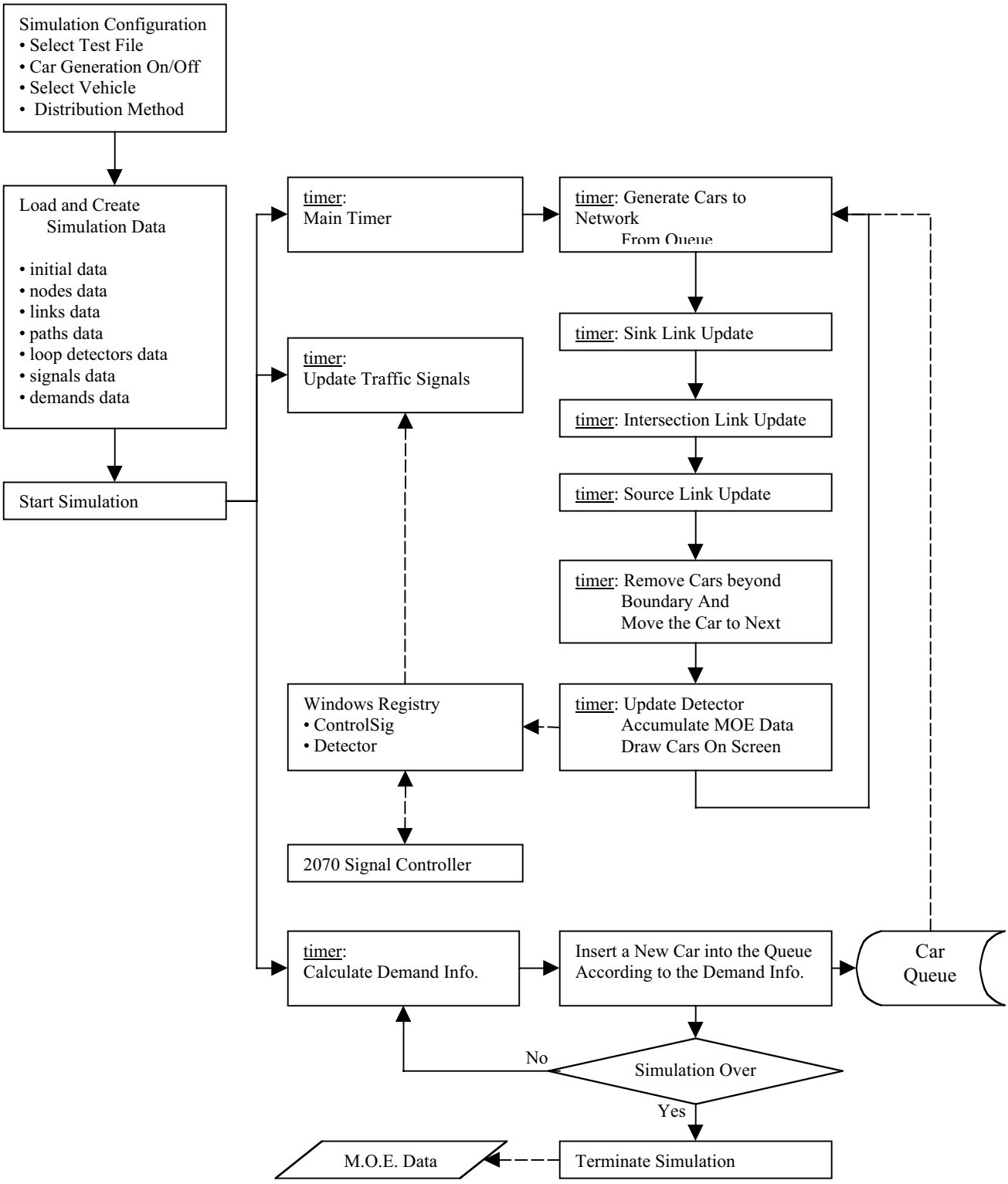
[7] Lowrie, P.R., "SCATS- Sydney Coordinated Adaptive Traffic System: A traffic responsive system for controlling urban traffic", Road and Traffic Authority of N.S.W Sydney, Australia, 1990.

[8] Papageorgiou, M., "Dynamic modeling, assignment, and route guidance in traffic networks", Transportation Research B, vol. 24B, pp 474-495, 1990.

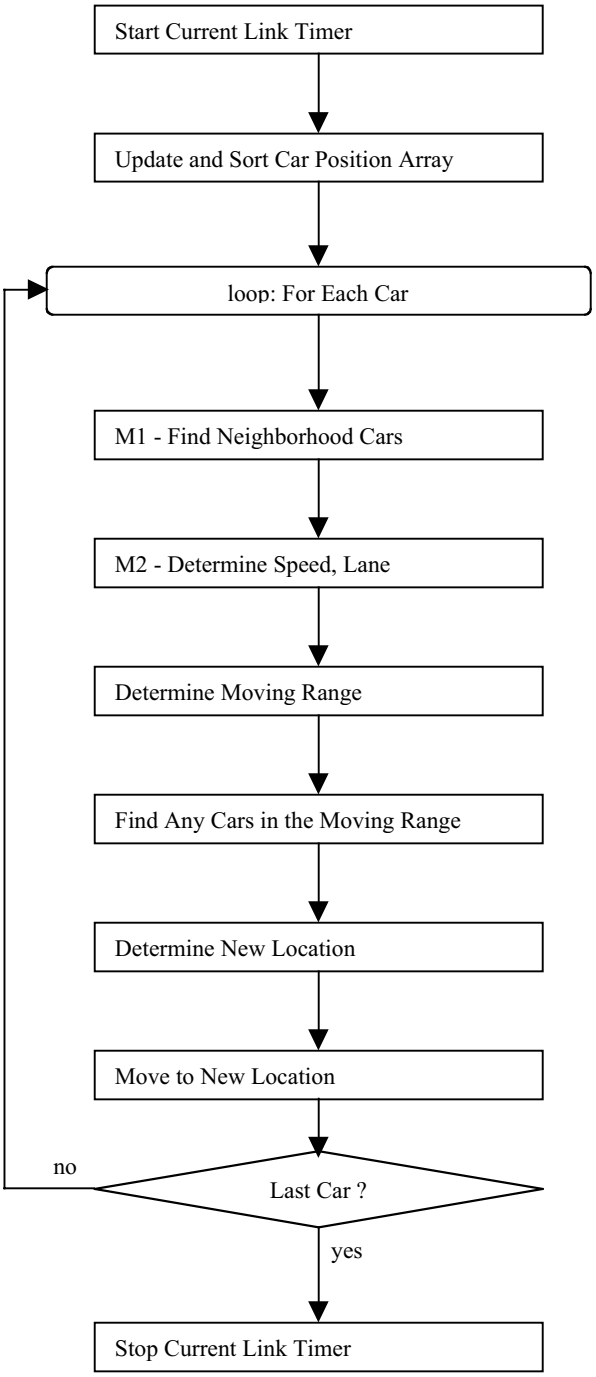
[9] Smith, M.J. and Ghali, M., "The dynamics of traffic assignment and traffic control: a theoretical study", Transportation Research B, vol. 24B, pp. 409-422, 1990.

APPENDIX A
Flow Chart for Microscopic Intersection Simulator

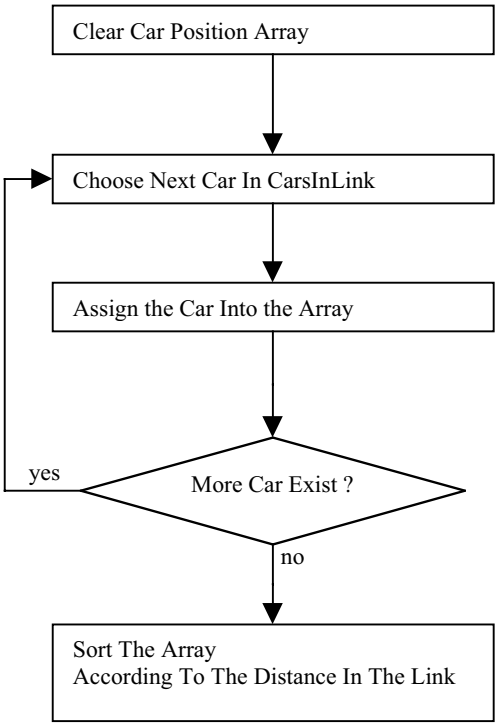
Overall Flow of Simulation Program

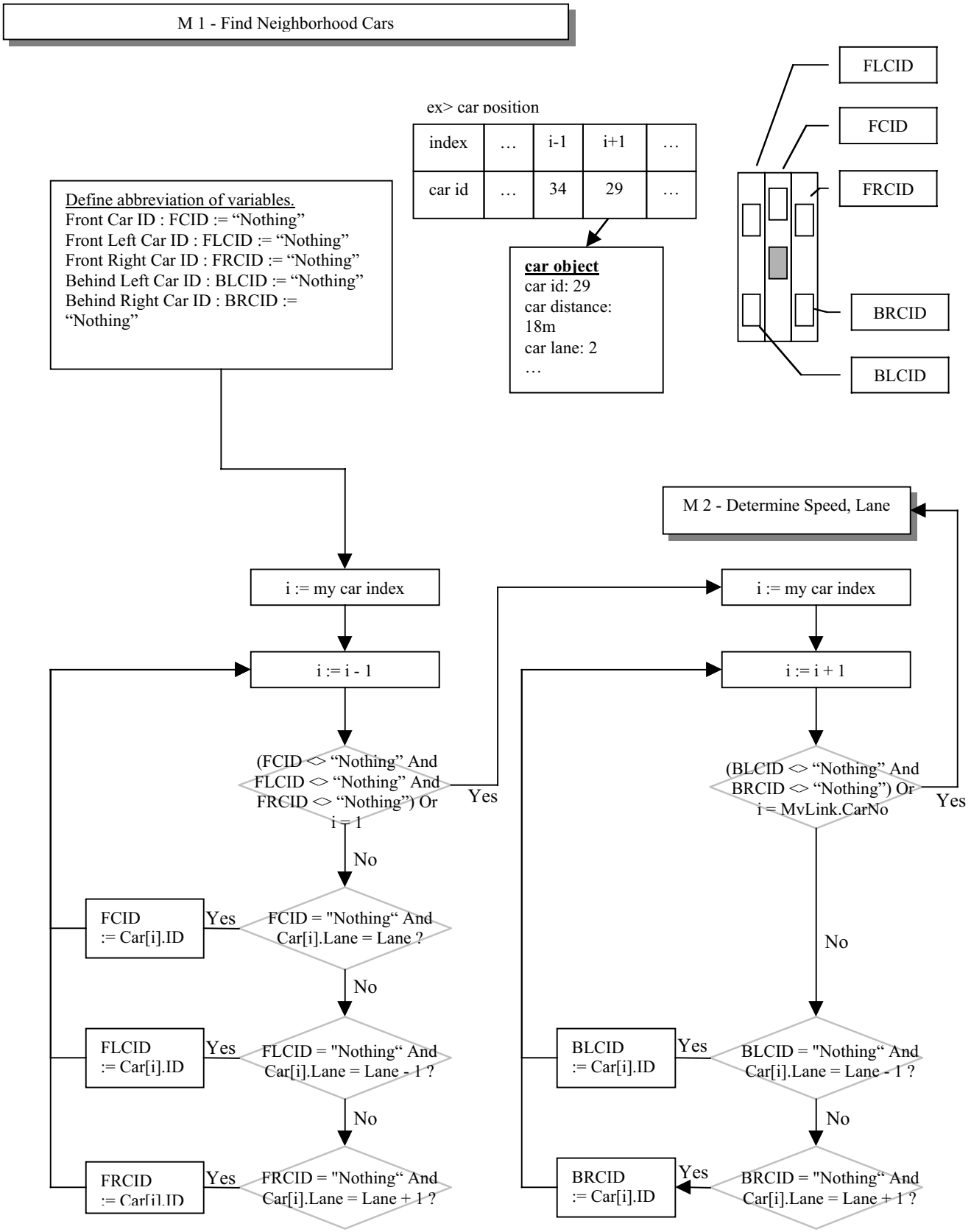


Link Update Flow

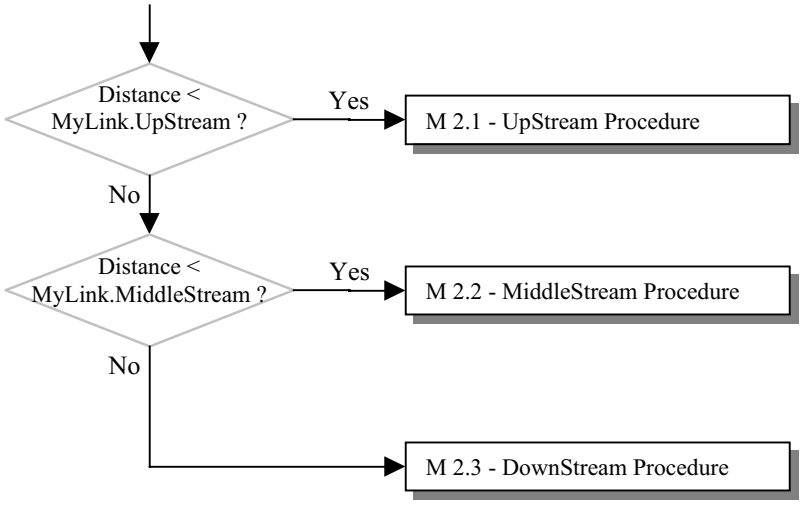


Update and Sort Car Position Array

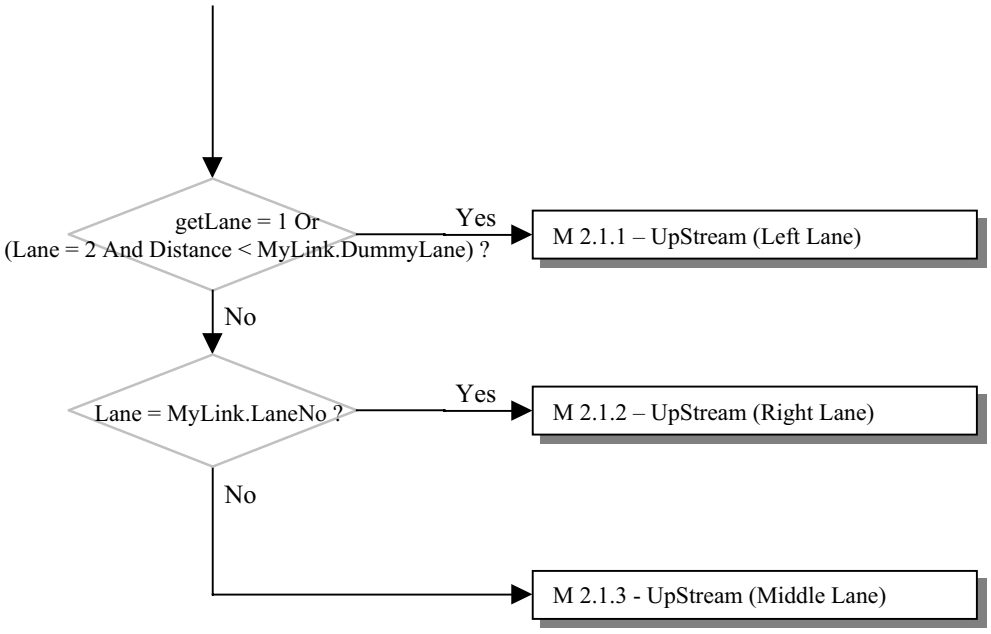



M 1 - Find Neighborhood Cars
ex> car position
index … i-1 i+1 …
car id … 34 29 …
car object
car id: 29
car distance: 18m
car lane: 2
…
FLCID
FCID
FRCID
BRCID
BLCID
Define abbreviation of variables.
Front Car ID : FCID := “Nothing”
Front Left Car ID : FLCID := “Nothing”
Front Right Car ID : FRCID := “Nothing”
Behind Left Car ID : BLCID := “Nothing”
Behind Right Car ID : BRCID := “Nothing”
M 2 - Determine Speed, Lane
i := my car index
i := i - 1
(FCID <> “Nothing” And FLCID <> “Nothing” And FRCID <> “Nothing”) Or i = 1
Yes
No
FCID := Car[i].ID
Yes
FCID = "Nothing“ And Car[i].Lane = Lane ?
No
FLCID := Car[i].ID
Yes
FLCID = "Nothing“ And Car[i].Lane = Lane - 1 ?
No
FRCID := Car[i].ID
Yes
FRCID = "Nothing“ And Car[i].Lane = Lane + 1 ?
i := my car index
i := i + 1
(BLCID <> “Nothing” And BRCID <> “Nothing”) Or i = MvLink.CarNo
Yes
No
BLCID := Car[i].ID
Yes
BLCID = "Nothing“ And Car[i].Lane = Lane - 1 ?
No
BRCID := Car[i].ID
Yes
BRCID = "Nothing“ And Car[i].Lane = Lane + 1 ?

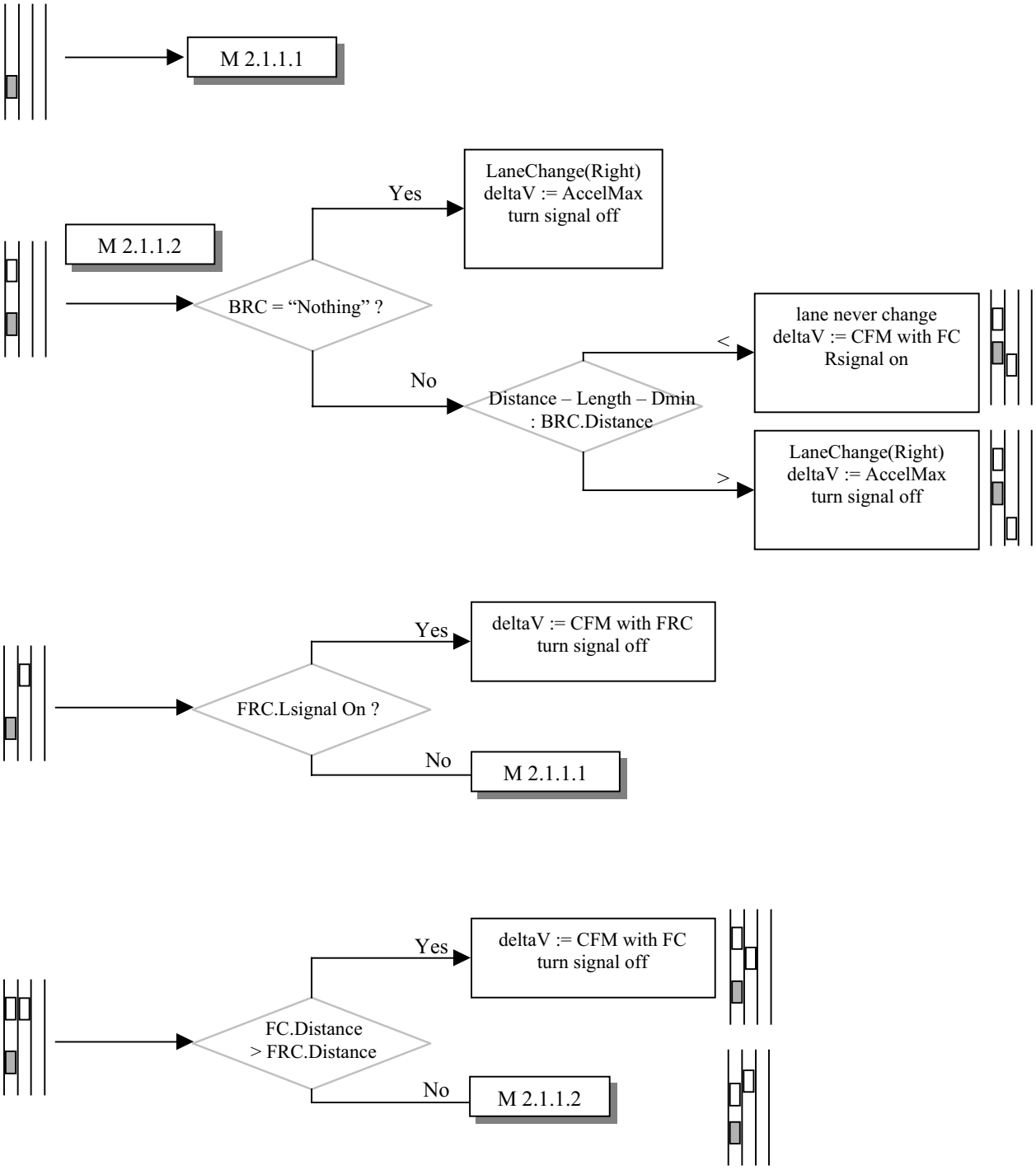
M 2 - Determine Speed, Lane



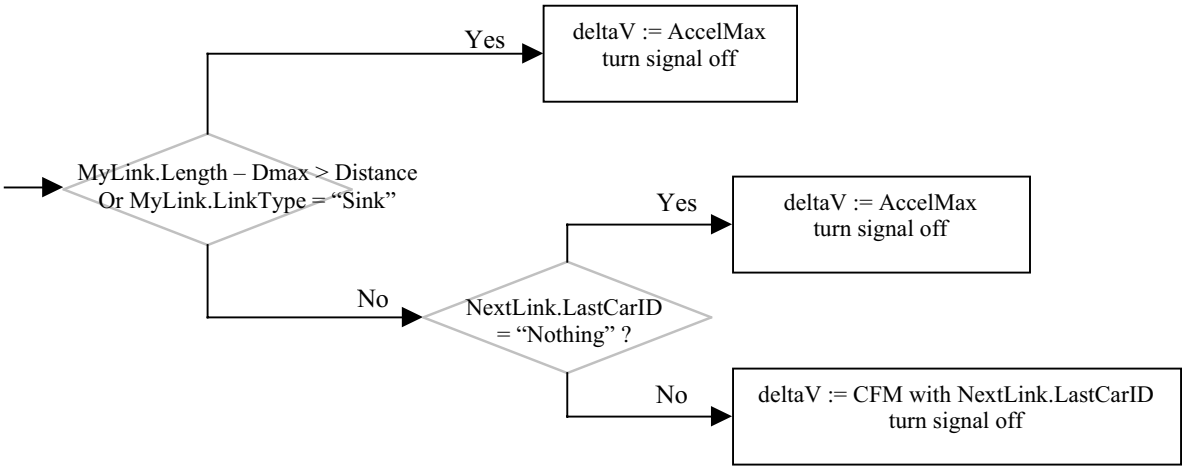
M 2.1 - UpStream Procedure



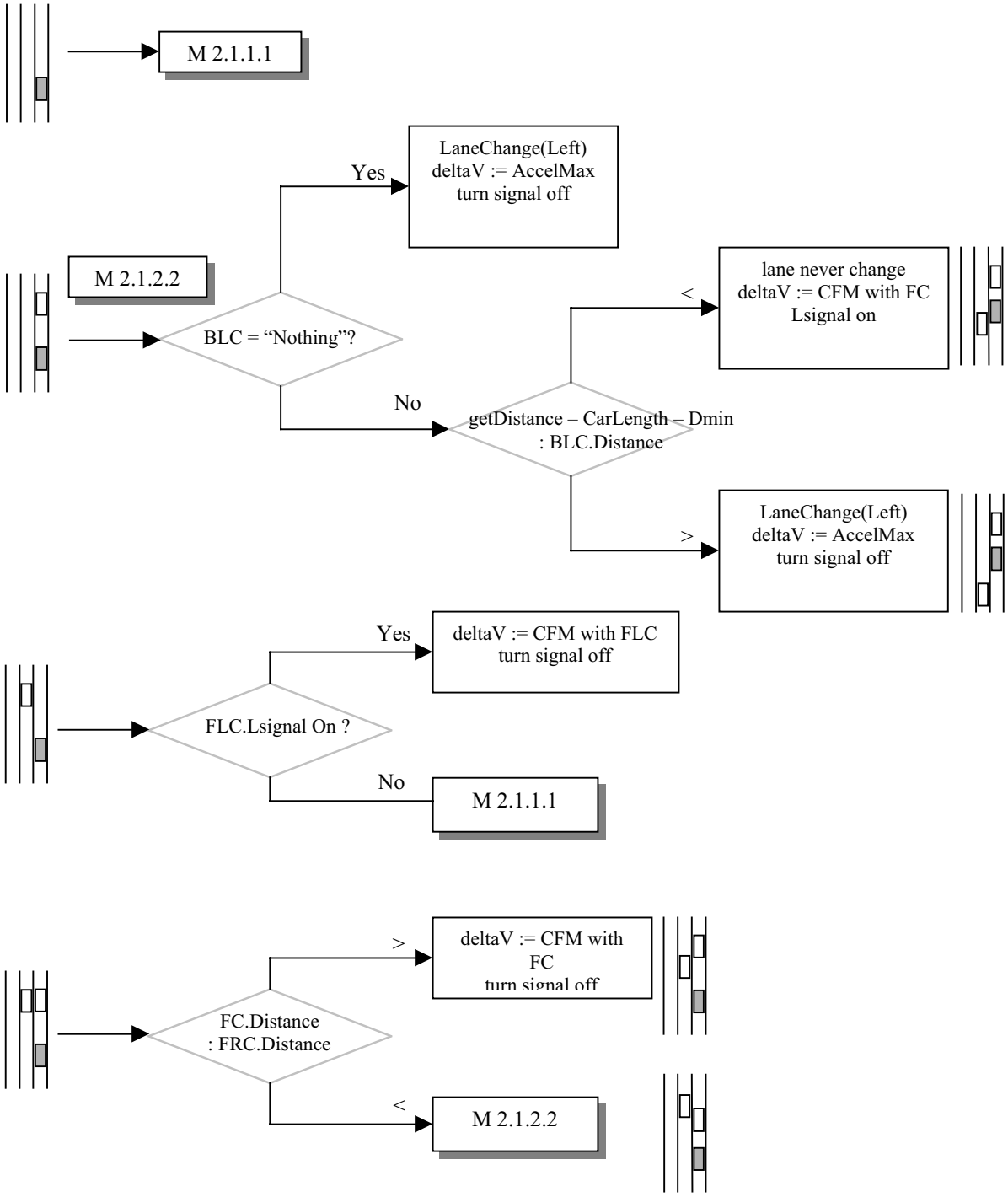
M 2.1.1 – UpStream (Left Lane)

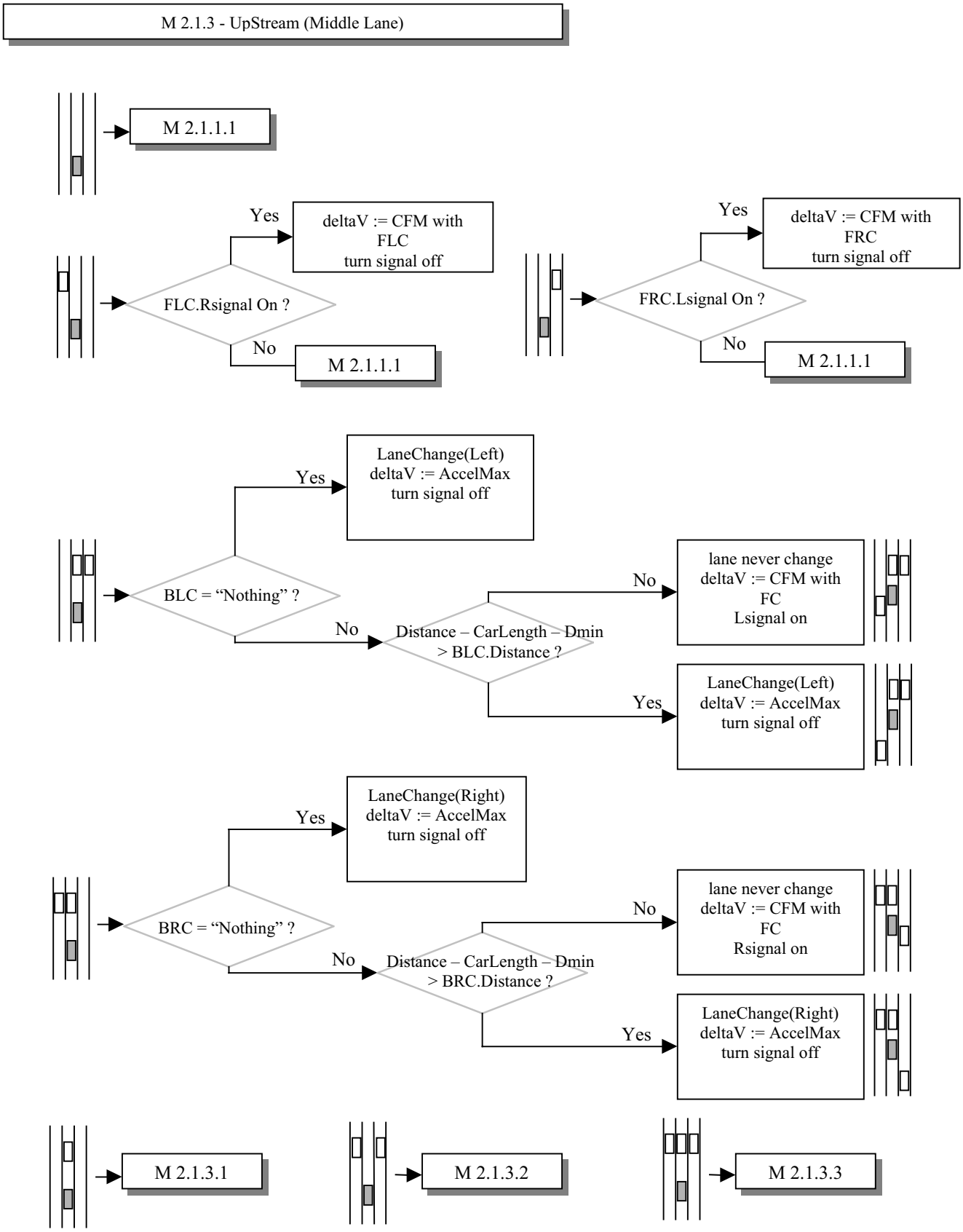


M 2.1.1.1

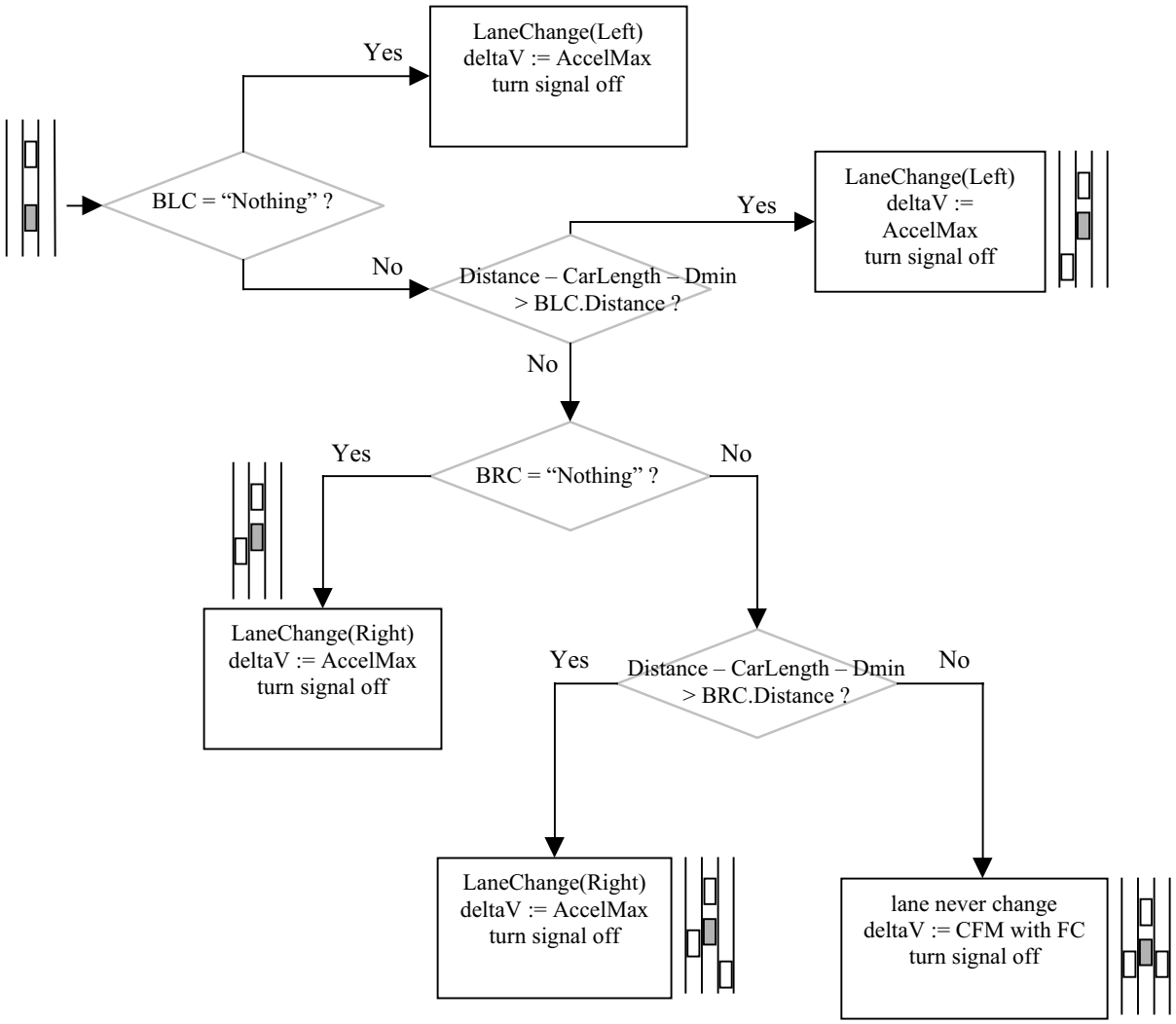

Yes
deltaV := AccelMax
turn signal off
MyLink.Length – Dmax > Distance
Or MyLink.LinkType = “Sink”
Yes
deltaV := AccelMax
turn signal off
No
NextLink.LastCarID
= “Nothing” ?
No
deltaV := CFM with NextLink.LastCarID
turn signal off

M 2.1.2 – UpStream (Right Lane)

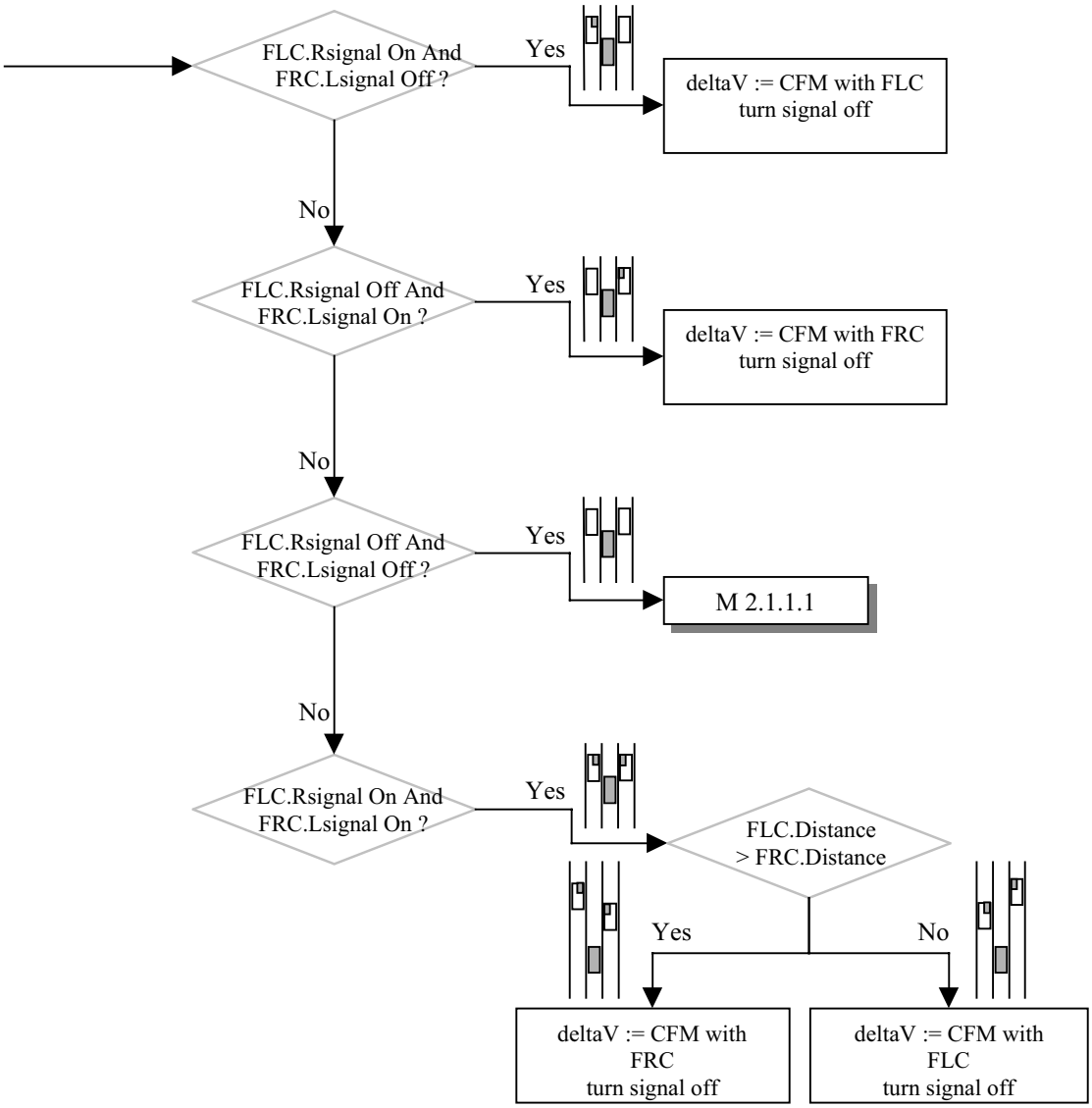


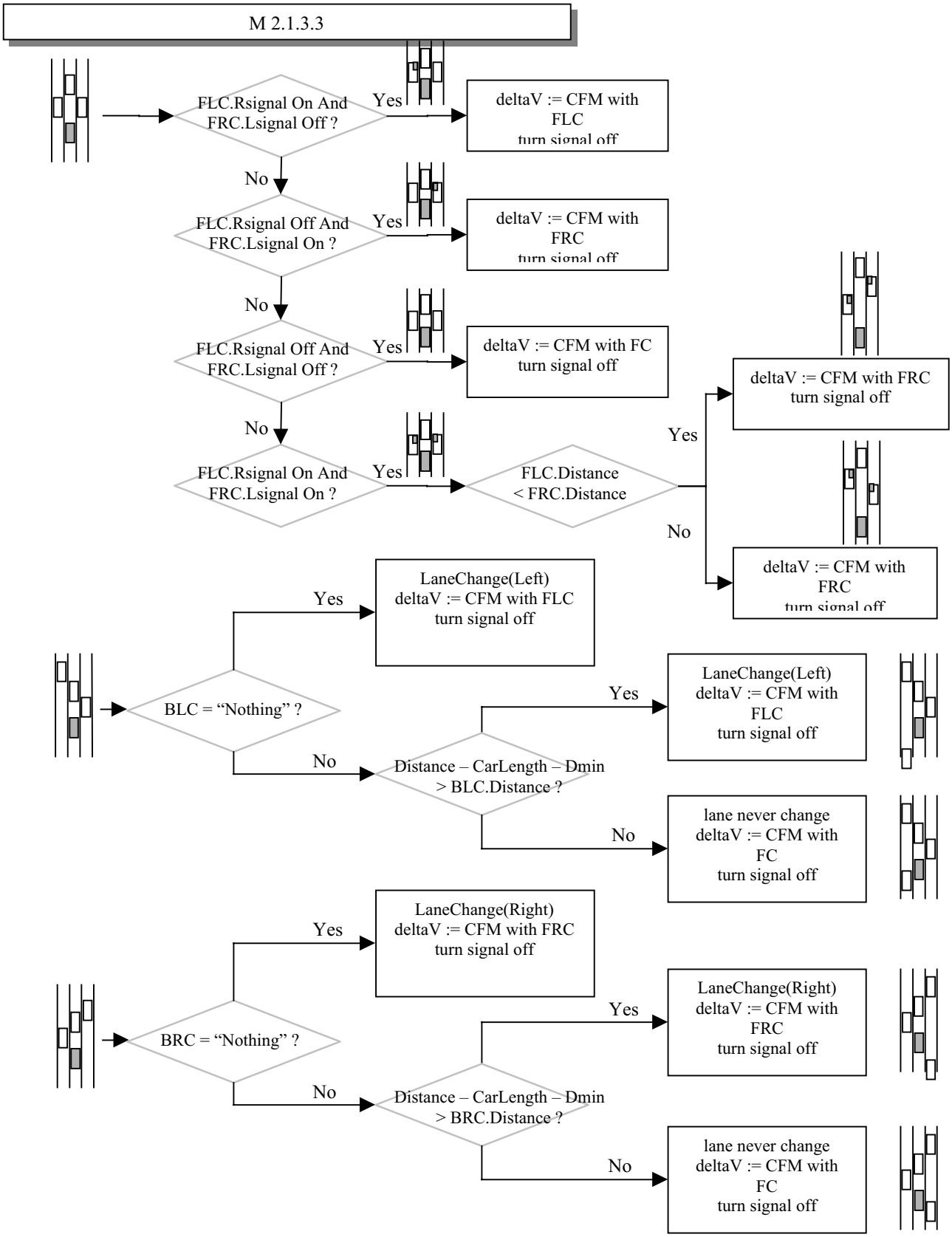

M 2.1.3 - UpStream (Middle Lane)
M 2.1.1.1
Yes
deltaV := CFM with FLC
turn signal off
FLC.Rsignal On ?
No
M 2.1.1.1
Yes
deltaV := CFM with FRC
turn signal off
FRC.Lsignal On ?
No
M 2.1.1.1
LaneChange(Left)
deltaV := AccelMax
turn signal off
Yes
BLC = "Nothing" ?
No
Distance – CarLength – Dmin > BLC.Distance ?
No
lane never change
deltaV := CFM with FC
Lsignal on
Yes
LaneChange(Left)
deltaV := AccelMax
turn signal off
LaneChange(Right)
deltaV := AccelMax
turn signal off
Yes
BRC = "Nothing" ?
No
Distance – CarLength – Dmin > BRC.Distance ?
No
lane never change
deltaV := CFM with FC
Rsignal on
Yes
LaneChange(Right)
deltaV := AccelMax
turn signal off
M 2.1.3.1
M 2.1.3.2
M 2.1.3.3

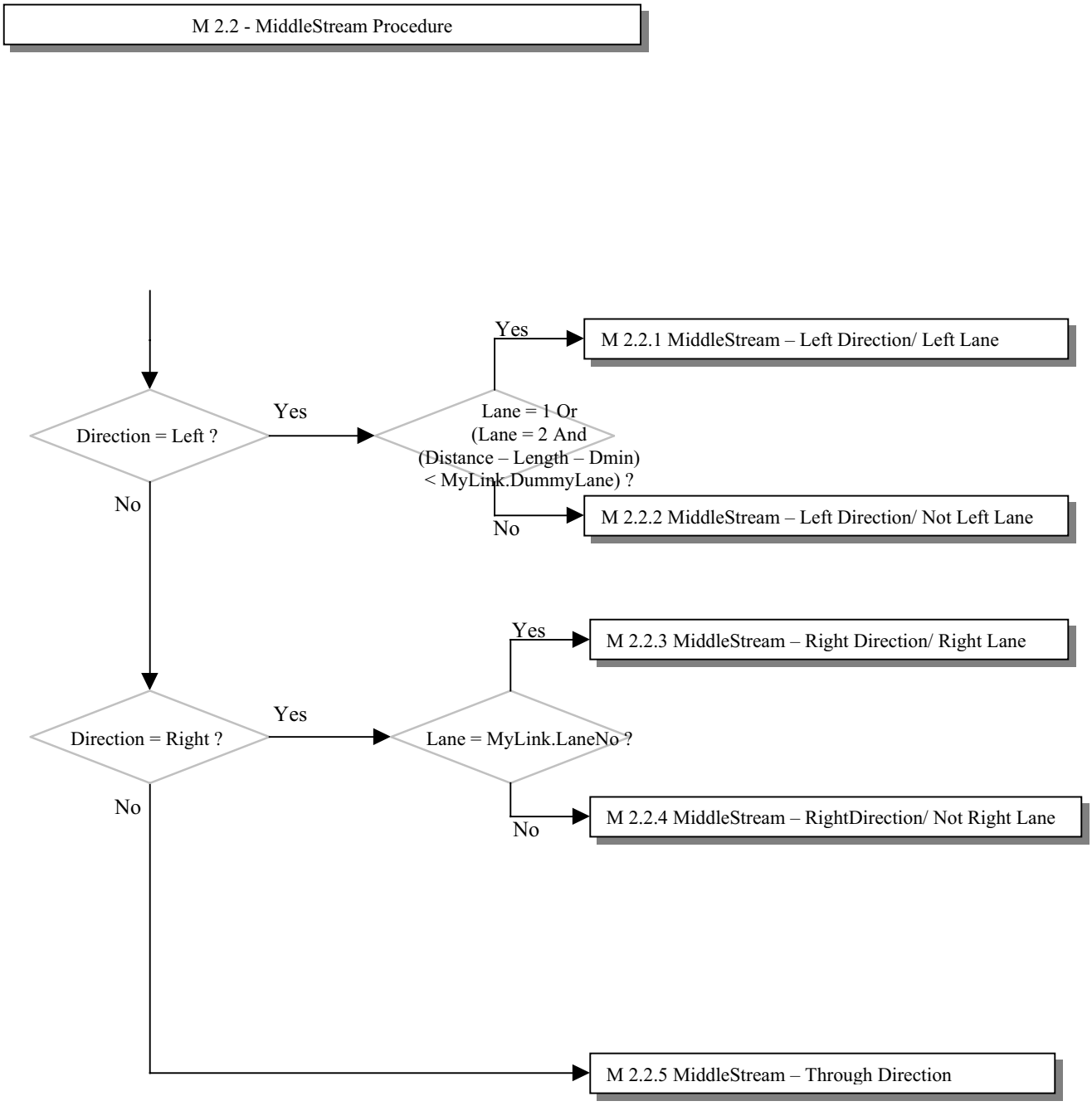
M 2.1.3.1


LaneChange(Left)
deltaV := AccelMax
turn signal off
Yes
BLC = "Nothing" ?
Yes
LaneChange(Left)
deltaV :=
AccelMax
turn signal off
No
Distance – CarLength – Dmin
> BLC.Distance ?
No
Yes
BRC = "Nothing" ?
No
LaneChange(Right)
deltaV := AccelMax
turn signal off
Yes
Distance – CarLength – Dmin
> BRC.Distance ?
No
LaneChange(Right)
deltaV := AccelMax
turn signal off
lane never change
deltaV := CFM with FC
turn signal off

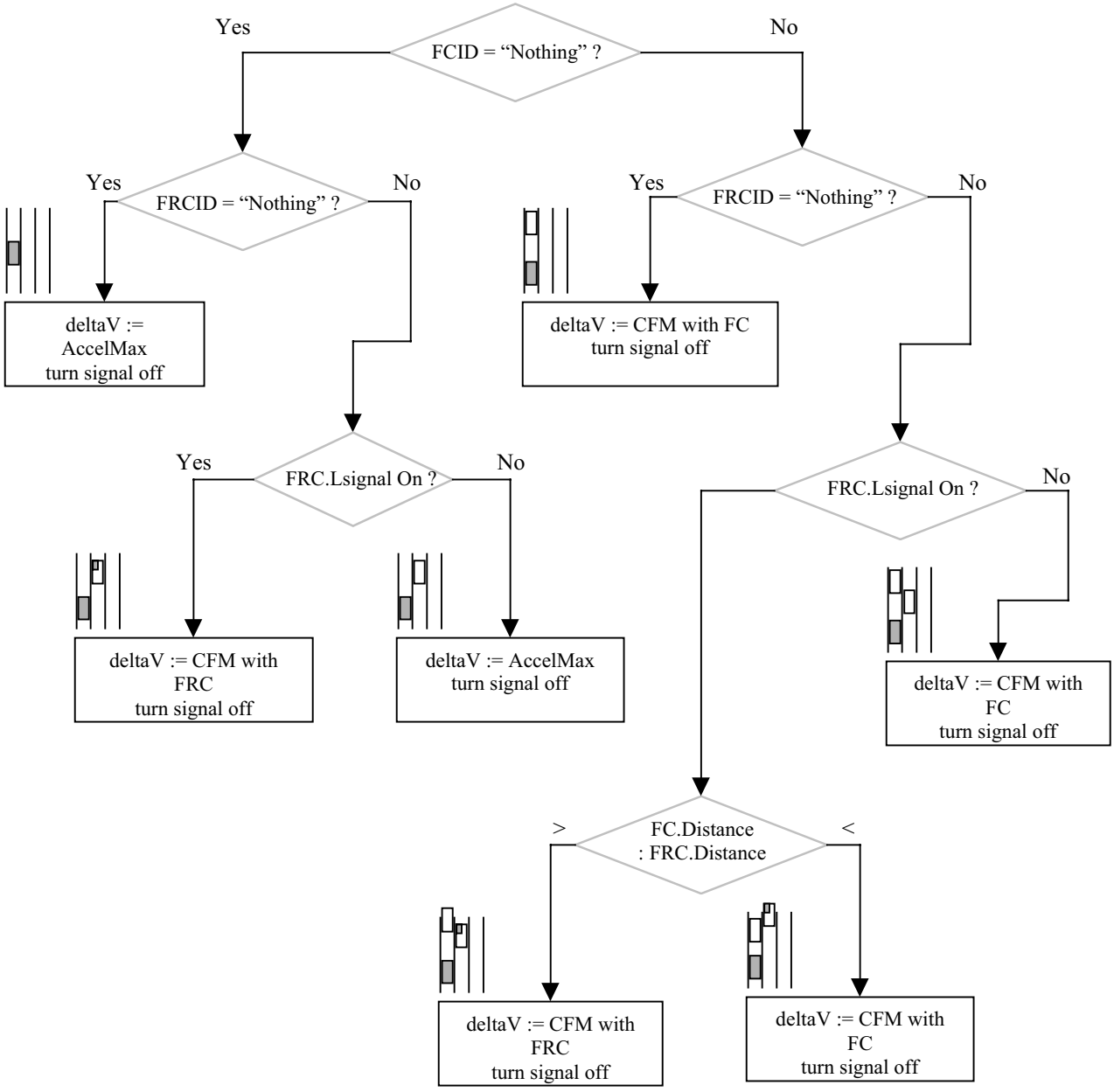
M 2.1.3.2


FLC.Rsignal On And FRC.Lsignal Off ?
Yes
deltaV := CFM with FLC turn signal off
No
FLC.Rsignal Off And FRC.Lsignal On ?
Yes
deltaV := CFM with FRC turn signal off
No
FLC.Rsignal Off And FRC.Lsignal Off ?
Yes
M 2.1.1.1
No
FLC.Rsignal On And FRC.Lsignal On ?
Yes
FLC.Distance > FRC.Distance
Yes
No
deltaV := CFM with FRC turn signal off
deltaV := CFM with FLC turn signal off

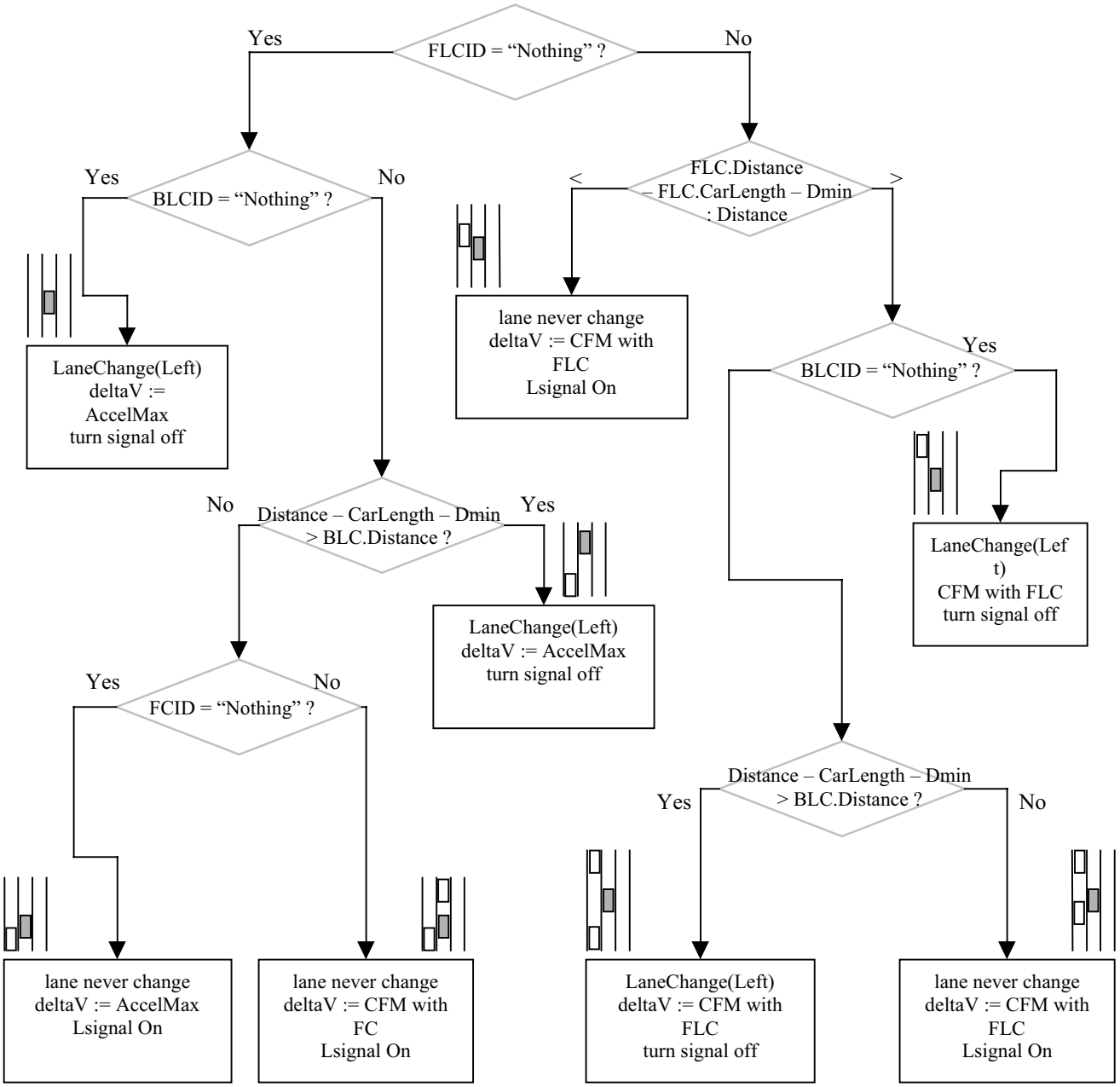

M 2.1.3.3
FLC.Rsignal On And FRC.Lsignal Off ?
Yes
deltaV := CFM with FLC turn signal off
No
FLC.Rsignal Off And FRC.Lsignal On ?
Yes
deltaV := CFM with FRC turn signal off
No
FLC.Rsignal Off And FRC.Lsignal Off ?
Yes
deltaV := CFM with FC turn signal off
No
FLC.Rsignal On And FRC.Lsignal On ?
Yes
FLC.Distance < FRC.Distance
Yes
deltaV := CFM with FRC turn signal off
No
deltaV := CFM with FRC turn signal off
BLC = "Nothing" ?
Yes
LaneChange(Left) deltaV := CFM with FLC turn signal off
No
Distance – CarLength – Dmin > BLC.Distance ?
Yes
LaneChange(Left) deltaV := CFM with FLC turn signal off
No
lane never change deltaV := CFM with FC turn signal off
BRC = "Nothing" ?
Yes
LaneChange(Right) deltaV := CFM with FRC turn signal off
No
Distance – CarLength – Dmin > BRC.Distance ?
Yes
LaneChange(Right) deltaV := CFM with FRC turn signal off
No
lane never change deltaV := CFM with FC turn signal off


M 2.2 - MiddleStream Procedure
Direction = Left ?
Yes
No
Lane = 1 Or
(Lane = 2 And
(Distance – Length – Dmin)
< MyLink.DummyLane) ?
Yes
No
M 2.2.1 MiddleStream – Left Direction/ Left Lane
M 2.2.2 MiddleStream – Left Direction/ Not Left Lane
Direction = Right ?
Yes
No
Lane = MyLink.LaneNo ?
Yes
No
M 2.2.3 MiddleStream – Right Direction/ Right Lane
M 2.2.4 MiddleStream – RightDirection/ Not Right Lane
M 2.2.5 MiddleStream – Through Direction

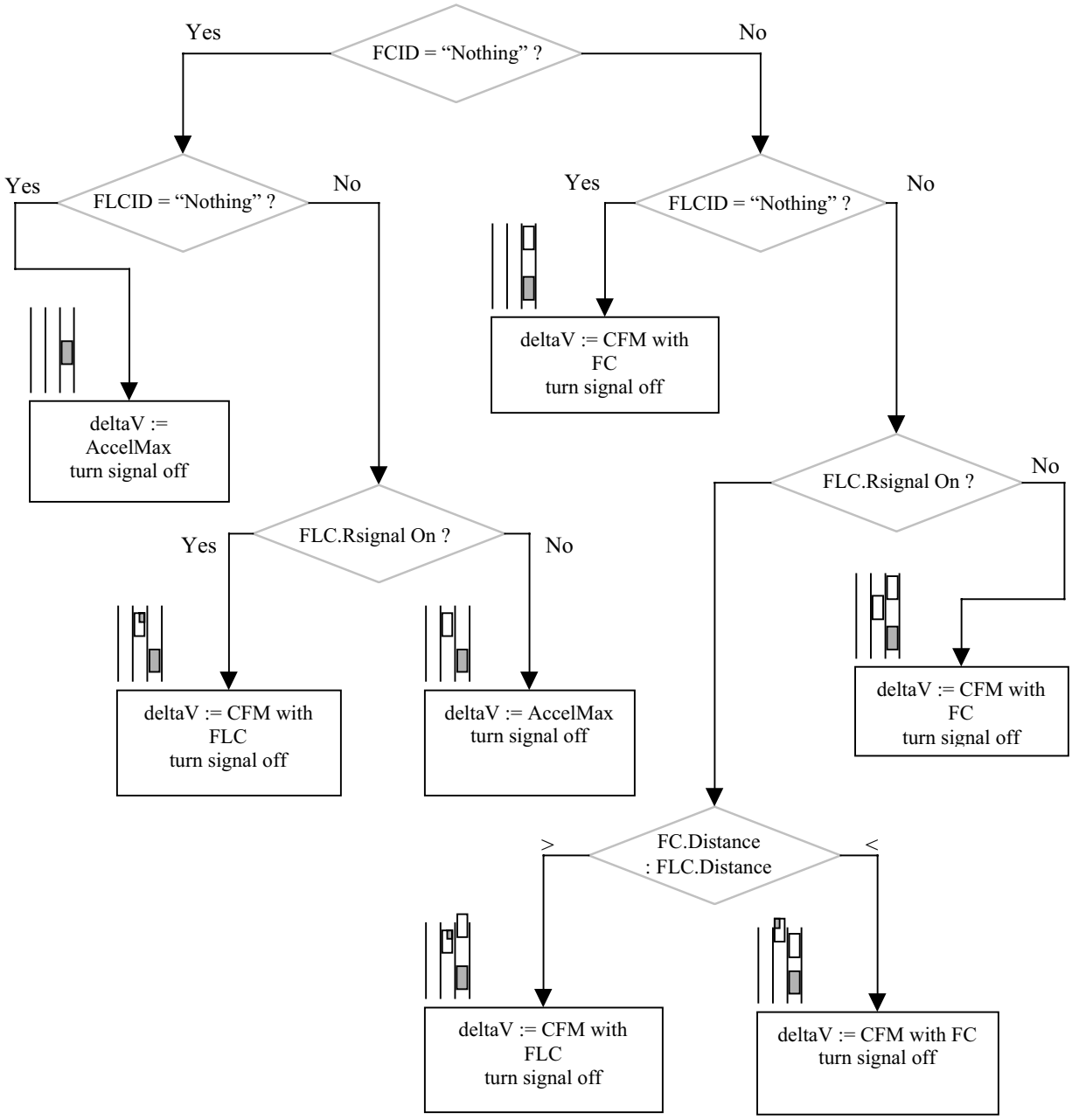
M 2.2.1 MiddleStream – Left Direction/ Left Lane



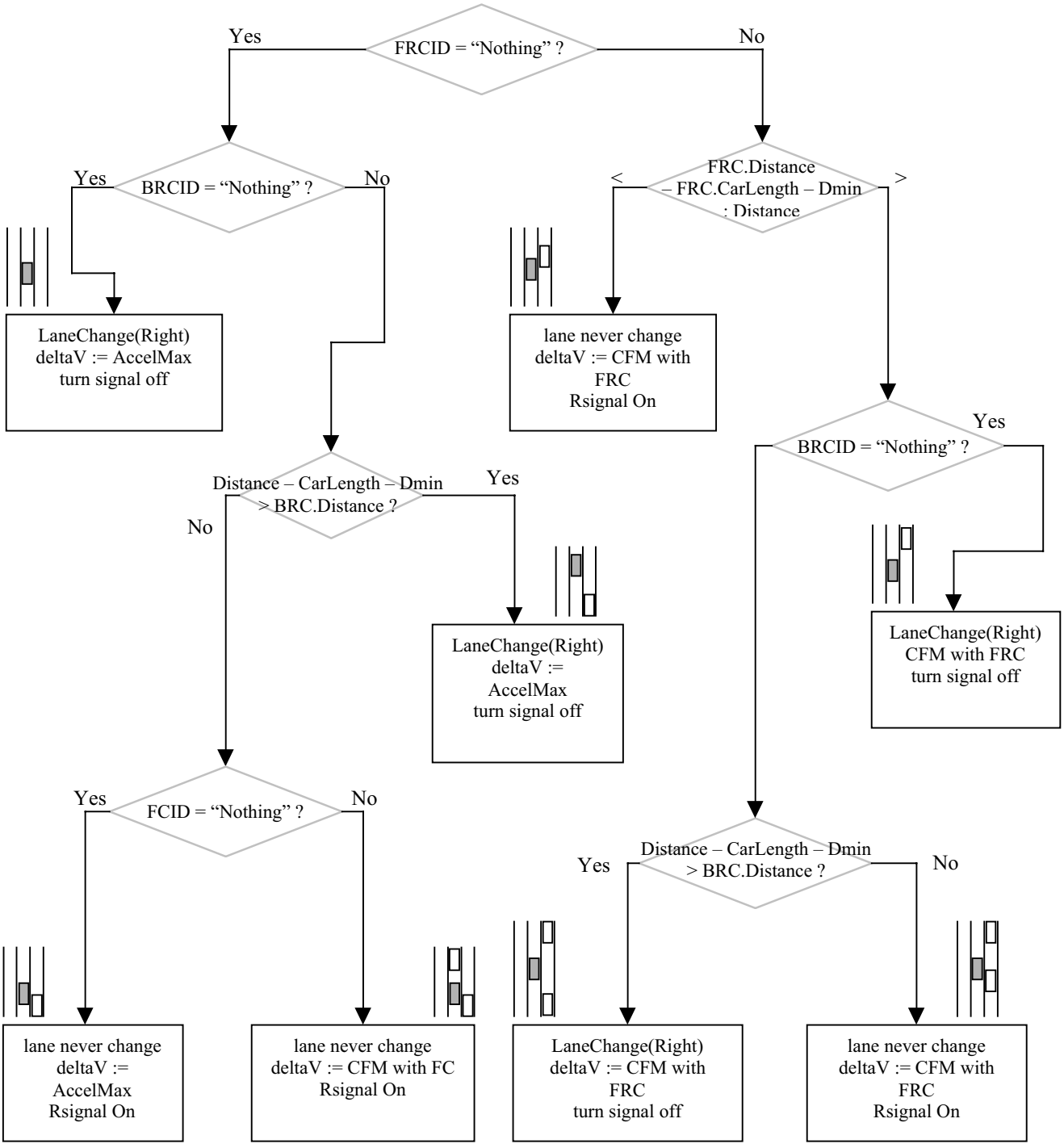
M 2.2.2 MiddleStream – Left Direction/ Not Left Lane



M 2.2.3 MiddleStream – Right Direction/ Right Lane



M 2.2.4 MiddleStream – RightDirection/ Not Right Lane



M 2.2.5 MiddleStream – Through Direction

M 2.2.5 is the same as

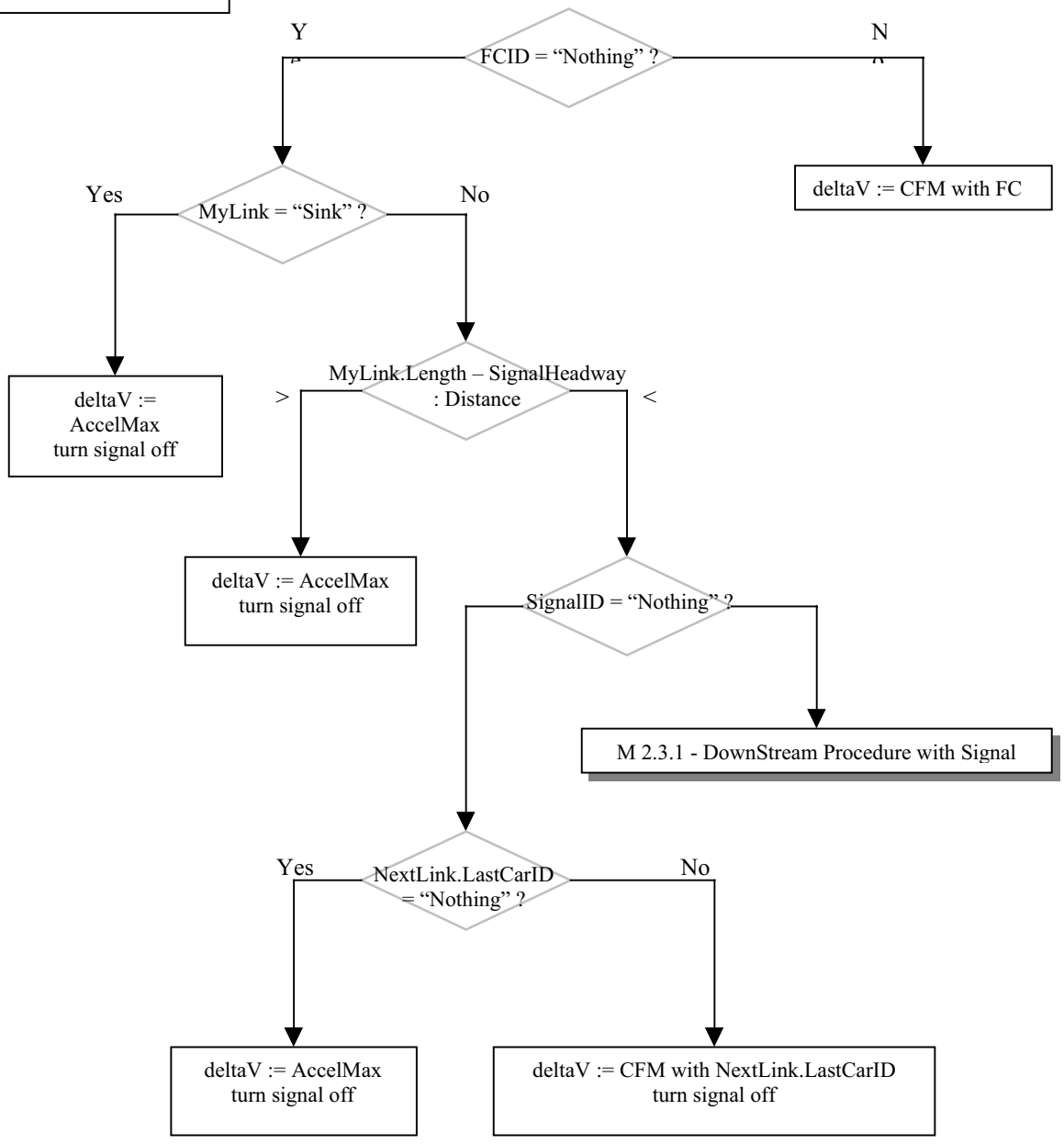
M 2.1.3 - UpStream (Middle Lane)

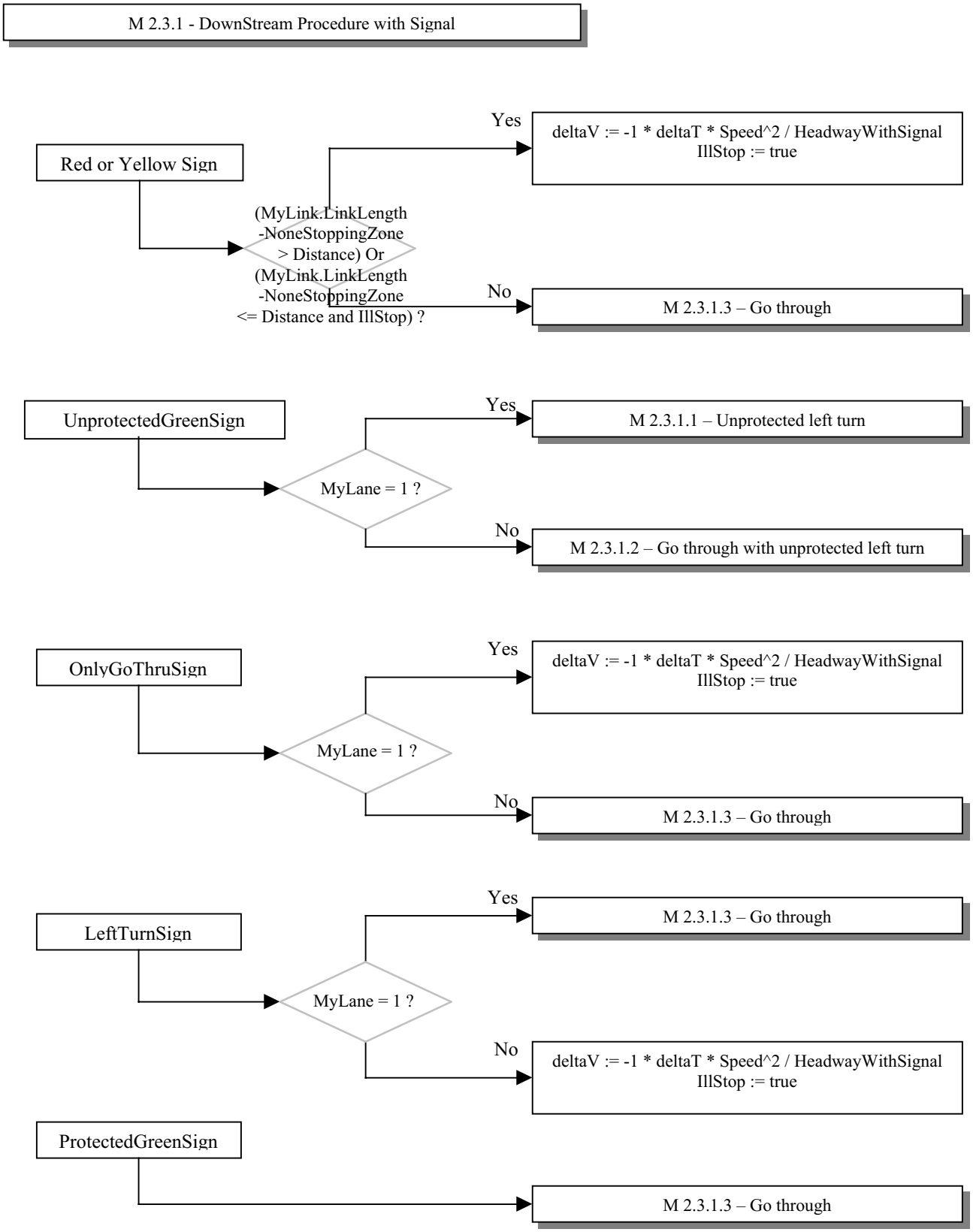
Except MyCar must not go into the left lane.

M 2.3 - DownStream Procedure

We assume that there is no lane changing in DownStream area.
We assume that the signal is at the end of the

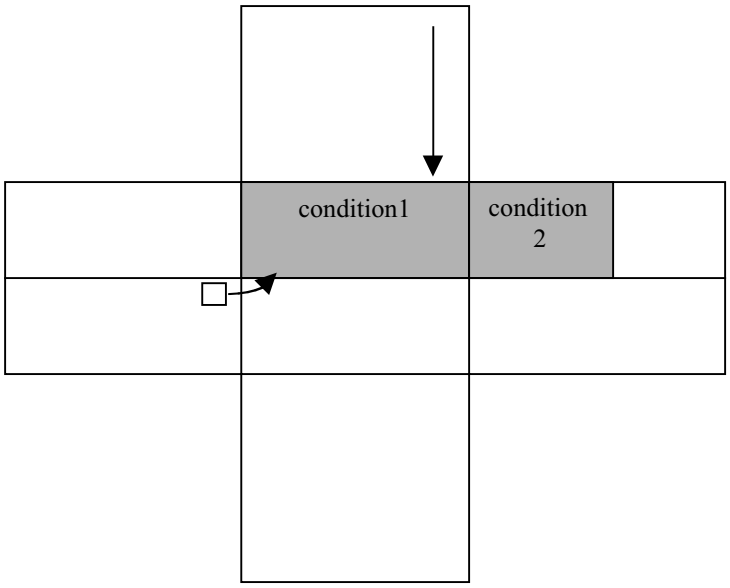
Select Case GetDirection
Case “L” LSignal On
Case “R” LSignal On
Case Else turn signal off
End Select




M 2.3.1 - DownStream Procedure with Signal
Red or Yellow Sign
(MyLink.LinkLength -NoneStoppingZone > Distance) Or (MyLink.LinkLength -NoneStoppingZone <= Distance and IllStop) ?
Yes
deltaV := -1 * deltaT * Speed^2 / HeadwayWithSignal
IllStop := true
No
M 2.3.1.3 – Go through
UnprotectedGreenSign
MyLane = 1 ?
Yes
M 2.3.1.1 – Unprotected left turn
No
M 2.3.1.2 – Go through with unprotected left turn
OnlyGoThruSign
MyLane = 1 ?
Yes
deltaV := -1 * deltaT * Speed^2 / HeadwayWithSignal
IllStop := true
No
M 2.3.1.3 – Go through
LeftTurnSign
MyLane = 1 ?
Yes
M 2.3.1.3 – Go through
No
deltaV := -1 * deltaT * Speed^2 / HeadwayWithSignal
IllStop := true
ProtectedGreenSign
M 2.3.1.3 – Go through

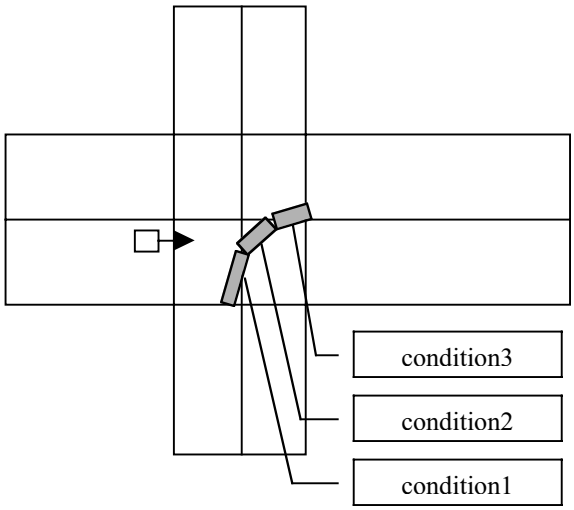
M 2.3.1.1 – Unprotected left turn

For unprotected left turn, two conditions are required.
First condition, there is no car in the intersection.
Second condition, there is no car in the opposite link. (UnprotectedLeftTurnCarefulZone)

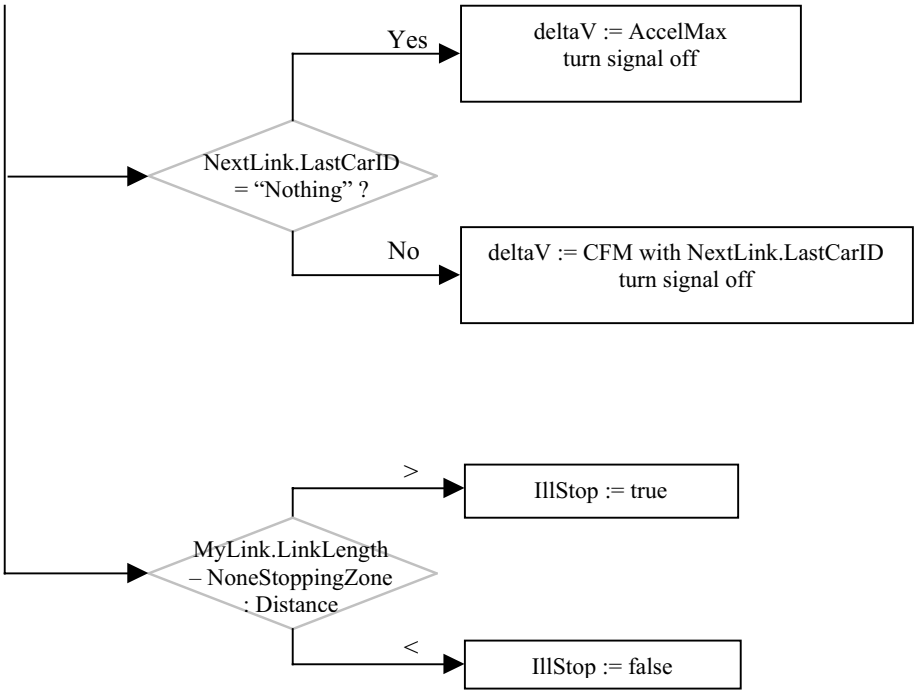


M 2.3.1.2 – Go through with unprotected left turn

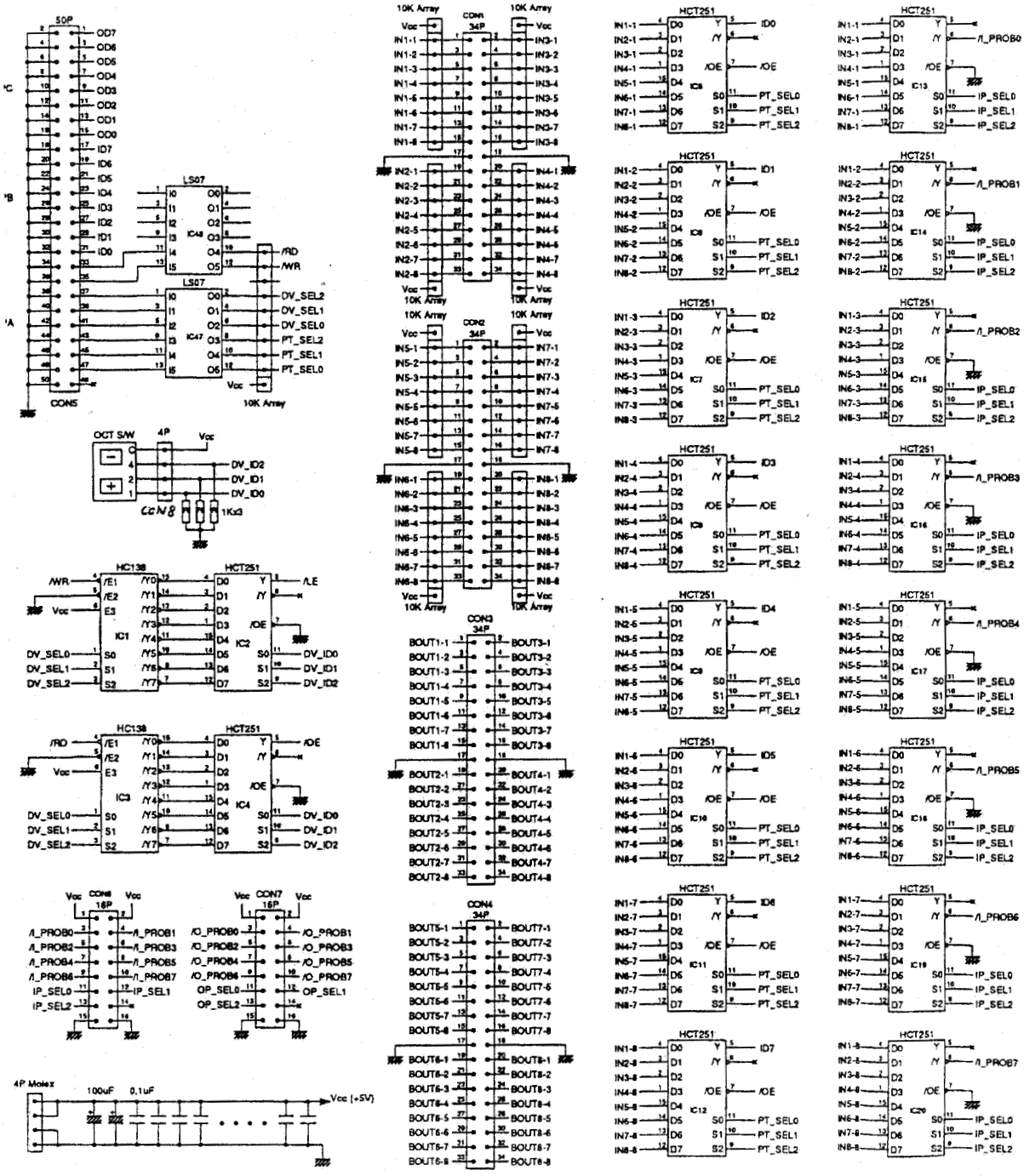
For going through with unprotected left turn, three conditions are required.
For three conditions, there is no car in the intersection.

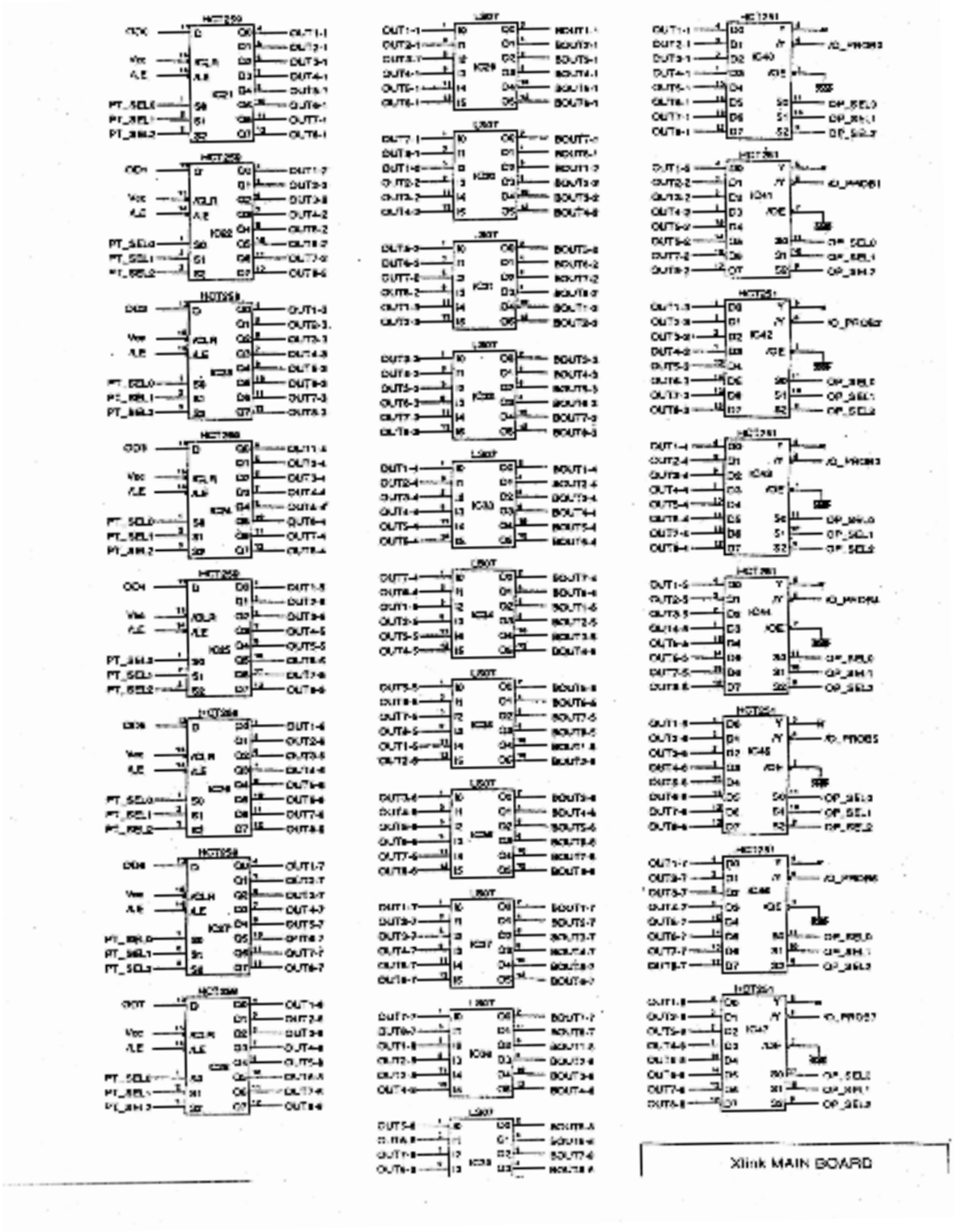


M 2.3.1.3 – Go through



APPENDIX B
Circuit Diagram for Signal Converter




Xilink MAIN BOARD

APPENDIX C
Code for Intersection Control Strategies for 2070 Controller

APPENDIX C: Code for Intersection Control Strategies for 2070 Controller

```
/* Traffic Signal Control Program for 2070 controller for fixed,
   actuated and adaptive control strategies. This code is written with the
   Field I/O manager module provided by the City of Los Angeles.

*/

/* ------------------------------------------------
** include some libraries
** ------------------------------------------------ */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <fioman.h>
#include <fiomod.h>
#include <cglob.h>

/* ------------------------------------------------
** define some fixed variable
** ------------------------------------------------ */

/* if you want to debug information, define DEBUG */
#define DEBUG

/* when you send data from 2070 controller to PC,
   you have to send (flush) data periodically.
   or the 2070 controller reset sent data to 0.
   I send data at 10 ticks interval. (10 ticks = 10 / 120 second)
*/
#define STAGE_FLUSH_ALARM           220
#define STAGE_FLUSH_INTERVAL  10     /* unit: ticks (10 ticks = 1/12 sec) */
#define STAGE_CHANGE_ALARM          222
#define DETECT_POLL_ALARM           221
#define DETECT_POLL_INTERVAL  2      /* unit: ticks (2 ticks = 1/60 sec) */
#define VOLOCC_REPORT_ALARM         223
#define     VOLOCC_REPORT_INTERVAL  7200 /* unit: ticks (7200 ticks = 1 min) */
#define LEN_IO                          8
#define LEN_REGISTRY                16
#define LEN_BUF                         1024
#define LEN_FILENAME                256
#define LEN_DETECTOR_VALUE          32
#define MAX_STAGE                   17
#define MAX_STAGE_SEQ               30
#define LINK_NO                         4
#define DETECTOR_NO_IN_LINK         8
#define     WEST                        0
#define     SOUTH                       1
#define     EAST                        2
#define     NORTH                       3
#define SUCCESS                         1
#define FAILURE                         0
#define TRUE                            1
#define FALSE                           0
```

```
#define FIXED                              1
#define ACTUATED                    2
#define ADAPTIVE                    3

/* ------------------------------------------------
** function definition
** ------------------------------------------------ */
void initialize();
void simple_sighandler(register int signal);
void finilize();
int load_signal_registry_definition(char *filename);
int lookup_stage_registry_value(int si, char *registry_value);
int load_signal_seq(char *filename);
int load_detector_weight(char *filename);
int extension_by_actuated_control();
int extension_by_adaptive_control();
double get_congestion_index(int si);
void do_sleep(int sec);
void do_polling();
int atox(char *s);
void set_signal_registry_value(char *signal_value);
void decode_detector_data();
void increase_current_stage_index();
void report_volocc();
void set_next_stage();

/* ------------------------------------------------
** define types
** ------------------------------------------------ */
typedef struct Stage
{
      int stageid;
      char registry_value[ LEN_REGISTRY + 1];
} Stage;

typedef struct StageSeq
{
      int stageid;
      int min_duration;
      int max_duration;
      int ext_interval;
} StageSeq;

typedef struct Link
{
      double congestion_index;
      int old_detector[ DETECTOR_NO_IN_LINK];
      int detector[ DETECTOR_NO_IN_LINK];
      int volume[ DETECTOR_NO_IN_LINK];
      int accumulated_occupancy_time[ DETECTOR_NO_IN_LINK];
      int volume_for_adaptive[ DETECTOR_NO_IN_LINK];
      double beta[ DETECTOR_NO_IN_LINK];
} Link;

/* ------------------------------------------------
** global variable
** ------------------------------------------------ */
```

```
/* fio device pointer */
TFio_dev *fio_dev;

/* alarm id */
alarm_id stage_flush_alarmid;
alarm_id stage_change_alarmid;
alarm_id detect_poll_alarmid;
alarm_id volocc_report_alarmid;

u_char ctrl_sig_buf[LEN_IO];
u_char detector_buf[LEN_IO];
u_char detector_buf_old[LEN_IO];

int current_stage_index;
Stage stage[MAX_STAGE];
StageSeq stage_seq[MAX_STAGE_SEQ];
Link link[LINK_NO];
int stage_seq_no;
char case_name[LEN_FILENAME + 1];
int extension_type;
int total_extension_interval;
FILE *result_file;
int volocc_count;
int started;
int terminated;

/* ------------------------------------------------
** main
** ------------------------------------------------ */
void main(int argc, char *argv[])
{

      /* if the number of argument is not 2, print out an usage message */
      if (argc != 3)
      {
            printf("Usage: 2070 test_case_name extension_type (1: fixed, 2:
actuated, 3: adaptive) \n");
            exit(0);
      }

      /* receive the test case_name, extension_type from the argument */
      strcpy(case_name,  argv[1]);
      extension_type = atoi(argv[2]);

      if (extension_type != 1 &&
            extension_type != 2 &&
            extension_type != 3)
      {
            printf("Usage: extension type must be 1 or 2 or 3. (1: fixed, 2:
actuated, 3: adaptive) \n");
            exit(0);
      }

      printf("------------------------------------------------\n");
      printf("If you want to terminate simulation, hit any key.\n");
      printf("------------------------------------------------\n");
```

```
	printf("loop started...\n");

	/* initialize program */
	initialize();

	while((ready_key()==FALSE) && (terminated == FALSE))
	{
		do_sleep(1);
	}
	printf("loop terminated...\n");

	/* finilize program */
	finilize();
}

/* ------------------------------------------------
** initialize
** : initialize program
** ------------------------------------------------ */
void initialize()
{
	int i;
	int j;
	char data_file_name[ LEN_FILENAME + 1];
	error_code ecode;
	char result_file_name[ LEN_FILENAME + 1];

	/* initialize variable */
	for (i=0; i<LINK_NO; i++)
	{
		link[ i].congestion_index = 0.0;

		for (j=0; j<DETECTOR_NO_IN_LINK; j++)
		{
			link[ i].old_detector[ j] = 0;
			link[ i].detector[ j] = 0;
			link[ i].volume[ j] = 0;
			link[ i].accumulated_occupancy_time[ j] = 0;
			link[ i].volume_for_adaptive[ j] = 0;
			link[ i].beta[ j] = 0.0;
		}
	}

	/* load signal registry definition */
	if (load_signal_registry_definition("registry.cfg") == FAILURE)
	{
		printf("registry.cfg file read failed.\n");
		finilize();
		exit(1);
	}

	/* load signal sequence */
	/* append file name with .dat */
	strcpy(data_file_name, case_name);
	strcat(data_file_name, ".dat");
	if (load_signal_seq(data_file_name) == FAILURE)
	{
```

```
            /* print out message */
            printf(strcat(data_file_name, " file read failed.\n"));
            finilize();
            exit(1);
      }

      /* load_detector weight */
      if (load_detector_weight("dw.cfg") == FAILURE)
      {

            /* print out message */
            printf("dw.cfg file read failed\n");
            finilize();
            exit(1);
      }

      /* link with field I/O manager */
      if (fioman_link(&fio_dev))
      {
            printf("Fioman link failed\n");
            finilize();
            exit(1);
      }

      /* create result file */
      strcpy(result_file_name, case_name);
      strcat(result_file_name, "_result.dat");
      if ((result_file = fopen(result_file_name, "w")) == NULL)
      {
            printf("result_file open failed.");
            exit(1);
      }

      volocc_count = 1;
      started = TRUE;
      terminated = FALSE;

      /* _os_intercept installs a signal intercept routine */
      /* simple_sighandler will work peoriodically like a timer */
      _os_intercept(simple_sighandler, _glob_data);

      _os_alarm_cycle(&stage_flush_alarmid, STAGE_FLUSH_ALARM,
STAGE_FLUSH_INTERVAL);
      _os_alarm_cycle(&detect_poll_alarmid, DETECT_POLL_ALARM,
DETECT_POLL_INTERVAL);
      _os_alarm_cycle(&volocc_report_alarmid, VOLOCC_REPORT_ALARM,
VOLOCC_REPORT_INTERVAL);

      /* start first stage */
      set_next_stage();
}

/* ------------------------------------------------
** simple_sighandler
** : it is triggered periodically like a timer
** ------------------------------------------------ */
void simple_sighandler(register int signal)
```

```
{
	switch (signal)
	{
		case STAGE_FLUSH_ALARM :
			fio_sig_outputs(fio_dev, &ctrl_sig_buf, 0);
			break;

		case DETECT_POLL_ALARM :
			do_polling();
			break;

		case STAGE_CHANGE_ALARM :
		set_next_stage();
			break;

		case VOLOCC_REPORT_ALARM :
			report_volocc();
		break;

		default:
		break;
	}
	_os_rte();
}

/* ---------------------------------------------------
** finilize
** --------------------------------------------------- */
void finilize()
{
	time_t ltime;
	char buffer[ LEN_BUF + 1];
	error_code ecode;

	/* delete os alarm */
	ecode = _os_alarm_delete(stage_flush_alarmid);
	if (ecode)
		printf("alarm delete Fail for STAGE_FLUSH_ALARM\n");

	ecode = _os_alarm_delete(detect_poll_alarmid);
	if (ecode)
		printf("alarm delete Fail for DETECT_POLL_ALARM\n");

	ecode = _os_alarm_delete(stage_change_alarmid);
	ecode =_os_alarm_delete(volocc_report_alarmid);

	/* unlink fioman */
	if (fioman_unlink())
		printf("fioman unlink failed\n");

	/* write the program closing time */
	time(&ltime);
	strcpy(buffer, "\nprogram closed: ");
	strcat(buffer, ctime(&ltime));
	printf(buffer);

	/* result_file close */
```

```
	fclose(result_file);
}

/* ------------------------------------------------
** load_signal_registry_definition
** : read the signal registry value from the file
** ------------------------------------------------ */
int load_signal_registry_definition(char *filename)
{
	FILE *fp;
	char buffer[ LEN_BUF + 1];
	char temp_filename[ LEN_FILENAME + 1];
	int i=0;

	/* tab is seperator for the file */
	/* be careful! : never make an empty line in the file. */
	char seperators[]   = "\t\n";
	char *registry_value;

	/* file open */
	if ((fp = fopen(filename, "r")) == NULL)
		return FAILURE;

	/* read and analyze every line */
	while (fgets(buffer, LEN_BUF, fp) != NULL)
	{
		/* retrieve stage name and registry value from the buffered line.
*/
		stage[ i].stageid = atoi(strtok(buffer, seperators));
		registry_value = strtok(NULL, seperators);
		strcpy(stage[ i].registry_value, registry_value);
		i++;
	}

	/* close file pointer */
	fclose(fp);

	/* print out message */
	strcpy(temp_filename, filename);
	printf(strcat(temp_filename, " reading ...Done\n"));
	return SUCCESS;
}

/* ------------------------------------------------
** lookup_stage_registry_value
** : lookup the signal registry value by stageid
** ------------------------------------------------ */
int lookup_stage_registry_value(int si, char *registry_value)
{
	int i;

	for (i=0; i<MAX_STAGE; i++)
	{
		if (si == stage[ i].stageid)
		{
			strcpy(registry_value, stage[ i].registry_value);
			return SUCCESS;
```

```
            }
      }

      return FAILURE;
}

/* ------------------------------------------------
** load_signal_seq
** : read the signal sequence from the file
** ------------------------------------------------ */
int load_signal_seq(char *filename)
{
      FILE *fp;
      char buffer[LEN_BUF + 1];
      char temp_filename[LEN_FILENAME + 1];

      /* tab is seperator for the file */
      /* be careful! : never make an empty line in the file. */
      char seperators[]   = "\t\n";

      /* file open */
      if ((fp = fopen(filename, "r")) == NULL)
            return FAILURE;

      /* read and analyze every line */
      stage_seq_no = 0;
      while (fgets(buffer, LEN_BUF, fp) != NULL)
      {
            /* retrieve stage name and registry value from the buffered line.
*/
            stage_seq[stage_seq_no].stageid      = atoi(strtok(buffer,
seperators));
            stage_seq[stage_seq_no].min_duration = atoi(strtok(NULL,
seperators));
            stage_seq[stage_seq_no].max_duration = atoi(strtok(NULL,
seperators));
            stage_seq[stage_seq_no].ext_interval = atoi(strtok(NULL,
seperators));
            stage_seq_no++;
      }

      /* close file pointer */
      fclose(fp);

      /* print out message */
      strcpy(temp_filename, filename);
      printf(strcat(temp_filename, " reading ...Done\n"));
      return SUCCESS;
}

/* ------------------------------------------------
** load_detector_weight
** : read the load detector weight (beta) from the file
** ------------------------------------------------ */
int load_detector_weight(char *filename)
{
      FILE *fp;
```

```
	char buffer[ LEN_BUF + 1] ;
	char temp_filename[ LEN_FILENAME + 1] ;
	int i=0;
	int temp_link_no = 0;

	/* tab is seperator for the file */
	/* be careful! : never make an empty line in the file. */
	char seperators[ ]    = "\t\n";
	char *detectorid;

	/* file open */
	if ((fp = fopen(filename, "r")) == NULL)
		return FAILURE;

	/* read and analyze every line */
	while (fgets(buffer, LEN_BUF, fp) != NULL)
	{
		/* retrieve detector id and its weight value from the buffered
line. */
		detectorid = strtok(buffer, seperators);
		link[ temp_link_no] .beta[ i]  = atof(strtok(NULL, seperators));

		i++;
		if ((i % DETECTOR_NO_IN_LINK) == 0)
		{
			temp_link_no++;
			i = 0;
		}
	}

	/* close file pointer */
	fclose(fp);

	/* print out message */
	strcpy(temp_filename, filename);
	printf(strcat(temp_filename, " reading ...Done\n"));
	return SUCCESS;
}

/* ---------------------------------------------------
** extension_by_actuated_control
** : actuated signal control extension type.
** --------------------------------------------------- */
int extension_by_actuated_control()
{
	int temp_ext_interval;
	int result;

	decode_detector_data();

	temp_ext_interval = stage_seq[ current_stage_index] .ext_interval;
	result = 0;

	switch (stage_seq[ current_stage_index] .stageid)
	{
		/* Stage01 */
		case 1:
```

```
                  /* if there is any car at the end of the lane 2,3 */
                  /* in either west or east link */
                  if ((link[ WEST].detector[ 1] == 1) ||
                        (link[ WEST].detector[ 2] == 1) ||
                        (link[ EAST].detector[ 1] == 1) ||
                        (link[ EAST].detector[ 2] == 1))
                  {
                        /*printf("actuated stage01\n");*/
                        result = temp_ext_interval;
                  }
                  break;

            /* Stage02 */
            case 2:
                  /* if there is any car at the end of the lane 1,2,3 in west
link */
                  if ((link[ WEST].detector[ 0] == 1) ||
                        (link[ WEST].detector[ 1] == 1) ||
                        (link[ WEST].detector[ 2] == 1))
                  {
                        /*printf("actuated stage02\n");*/
                        result = temp_ext_interval;
                  }
                  break;

            /* Stage03 */
            case 3:
                  /* if there is any car at the end of the lane 1,2,3 in east
link */
                  if ((link[ EAST].detector[ 0] == 1) ||
                        (link[ EAST].detector[ 1] == 1) ||
                        (link[ EAST].detector[ 2] == 1))
                  {
                        /*printf("actuated stage03\n");*/
                        result = temp_ext_interval;
                  }
                  break;

            /* Stage04 */
            case 4:
                  /* if there is any car at the end of the lane 1 in either
west or east link */
                  if ((link[ WEST].detector[ 0] == 1) ||
                        (link[ EAST].detector[ 0] == 1))
                  {
                        /*printf("actuated stage04\n");*/
                        result = temp_ext_interval;
                  }
                  break;

            /* Stage05 */
            case 5:
                  /* if there is any car at the end of the lane 1,2,3 in either
west or east link */
                  if ((link[ WEST].detector[ 0] == 1) ||
                        (link[ WEST].detector[ 1] == 1) ||
                        (link[ WEST].detector[ 2] == 1) ||
```

```
                        (link[ EAST].detector[ 0]  == 1) ||
                        (link[ EAST].detector[ 1]  == 1) ||
                        (link[ EAST].detector[ 2]  == 1))
                  {
                        /*printf("actuated stage05\n");*/
                        result = temp_ext_interval;
                  }
                  break;

            /* Stage06 */
            case 6:
                  /* if there is any car at the end of the lane 2,3 in either
south or north link */
                  if ((link[ SOUTH].detector[ 1]  == 1) ||
                        (link[ SOUTH].detector[ 2]  == 1) ||
                        (link[ NORTH].detector[ 1]  == 1) ||
                        (link[ NORTH].detector[ 2]  == 1))
                  {
                        /*printf("actuated stage06\n");*/
                        result = temp_ext_interval;
                  }
                  break;

            /* Stage07 */
            case 7:
                  /* if there is any car at the end of the lane 1,2,3 in south
link */
                  if ((link[ SOUTH].detector[ 0]  == 1) ||
                        (link[ SOUTH].detector[ 1]  == 1) ||
                        (link[ SOUTH].detector[ 2]  == 1))
                  {
                        /*printf("actuated stage07\n");*/
                        result = temp_ext_interval;
                  }
                  break;

            /* Stage08 */
            case 8:
                  /* if there is any car at the end of the lane 1,2,3 in north
link */
                  if ((link[ NORTH].detector[ 0]  == 1) ||
                        (link[ NORTH].detector[ 1]  == 1) ||
                        (link[ NORTH].detector[ 2]  == 1))
                  {
                        /*printf("actuated stage08\n");*/
                        result = temp_ext_interval;
                  }
                  break;

            /* Stage09 */
            case 9:
                  /* if there is any car at the end of the lane 1 in either
south or north link */
                  if ((link[ SOUTH].detector[ 0]  == 1) ||
                        (link[ NORTH].detector[ 0]  == 1))
                  {
                        /*printf("actuated stage09\n");*/
```

```
                        result = temp_ext_interval;
                  }
                  break;

            /* Stage10 */
            case 10:
                  /* if there is any car at the end of the lane 1,2,3 in either
south or north link */
                  if ((link[ SOUTH].detector[ 0] == 1) ||
                        (link[ SOUTH].detector[ 1] == 1) ||
                        (link[ SOUTH].detector[ 2] == 1) ||
                        (link[ NORTH].detector[ 0] == 1) ||
                        (link[ NORTH].detector[ 1] == 1) ||
                        (link[ NORTH].detector[ 2] == 1))
                  {
                        /*printf("actuated stage10\n");*/
                        result = temp_ext_interval;
                  }
                  break;
      }

      return result;
}

/* ------------------------------------------------
** extension_by_adaptive_control
** : adaptive signal control extension type.
** ------------------------------------------------ */
int extension_by_adaptive_control()
{
      int next_stage_index;
      double current_congestion_index = 0.0;
      double next_congestion_index = 0.0;
      int found;

      /* get current congestion index */
      current_congestion_index
            = get_congestion_index(stage_seq[ current_stage_index].stageid);

      /* get next stage index */
      /* this while loop is used to skip yellow and all red sign. */
      found = 0;
      next_stage_index = current_stage_index;
      while(found == 0)
      {
            next_stage_index++;
            if (next_stage_index == stage_seq_no)
                  next_stage_index = 0;

            if ((stage_seq[ next_stage_index].stageid >= 0) &&
                  (stage_seq[ next_stage_index].stageid <= 10))
                  found = 1;
      }

      /* get next congestion index */
      next_congestion_index
            = get_congestion_index(stage_seq[ next_stage_index].stageid);
```

```
      /* determine extending or not */
      /* if current congestion index is greater than next stage congestion
index */
      /* extend signal duration. */
      if (current_congestion_index > next_congestion_index)
            return stage_seq[ current_stage_index].ext_interval;

      else
            return 0;
}

/* ------------------------------------------------
** get_congestion_index
** : get congestion index according to the stage id
** ------------------------------------------------ */
double get_congestion_index(int si)
{
      int i = 0;
      int j = 0;
      double d = 0.0;
      double p = 0.0;
      double result = 0.0;

/*printf("\n--- get_congestion_index ------------------------\n");*/
      for (i=0; i<LINK_NO; i++)
            for (j=0; j<DETECTOR_NO_IN_LINK; j++)
      {
            /* p is 1.0, if detector j is occupied at the end of time intervla
t */
            if (link[ i].detector[ j] == 1)
                  p = 1.0;
            else
                  p = 0.0;

/*if (i==1)
      printf("link[ %d].detector[ %d]=[ %d]\n", i, j, link[ i].detector[ j]);*/

            /* apply formulation of link-congestion index */
            d = (p + link[ i].volume_for_adaptive[ j]) / (1 +
link[ i].volume_for_adaptive[ j]);
            link[ i].congestion_index += link[ i].beta[ j] * d;

/*if (i==1)
      printf("d=[ %f], ci=[ %f], accumulated_ci=[ %f]\n", d, (link[ i].beta[ j] *
d), link[ i].congestion_index);                                    */
      }

      /* stage id */
      /* congestion index is calculated according to the relevant links */
      switch (si)
      {
            case  1: result = link[ WEST].congestion_index +
link[ EAST].congestion_index; break;
            case  2: result = link[ WEST].congestion_index; break;
            case  3: result = link[ EAST].congestion_index; break;
```

```
            case  4: result = link[ WEST].congestion_index +
link[ EAST].congestion_index; break;
            case  5: result = link[ WEST].congestion_index +
link[ EAST].congestion_index; break;
            case  6: result = link[ SOUTH].congestion_index +
link[ NORTH].congestion_index; break;
            case  7: result = link[ SOUTH].congestion_index; break;
            case  8: result = link[ NORTH].congestion_index; break;
            case  9: result = link[ SOUTH].congestion_index +
link[ NORTH].congestion_index; break;
            case 10: result = link[ SOUTH].congestion_index +
link[ NORTH].congestion_index; break;
            default: 0.0;
      }

      /* initialize variables */
      for (i=0; i<LINK_NO; i++)
            link[ i].congestion_index = 0.0;

/*printf("stage%2d congestion Index =[ %7.3f]\n", si, result);*/
      return result;
}

/* ------------------------------------------------
** do_sleep
** : sleep for a given time without waking up by signals
** ------------------------------------------------ */
void do_sleep(int sec)
{
      u_int32 tcount;
      error_code ecode = 0;

      /* convert second to ticks */
      tcount = sec * 120;

      while (tcount > 0)
            ecode = _os9_sleep(&tcount);
}

/* ------------------------------------------------
** do_polling
** : retrieve and accumulate volume of vehicles
** ------------------------------------------------ */
void do_polling()
{
      int i;
      int j;

      decode_detector_data();

      /* count the number of vehicles crossing detectors during time interval
*/
      for (i=0; i<LINK_NO; i++)
            for (j=0; j<DETECTOR_NO_IN_LINK; j++)
      {
            /* if detector value is converted from 1 to 0, */
            /* it is considered as one vehicle passing */
```

```
            if ((link[ i] .detector[ j]  == 0) && (link[ i] .old_detector[ j]  == 1))
            {
                  link[ i] .volume[ j] ++;
                  link[ i] .volume_for_adaptive[ j] ++;
            }

            /* if detector value is 1 */
            /* it is considered as occupied detector */
            if (link[ i] .detector[ j]  == 1)
                  link[ i] .accumulated_occupancy_time[ j]  +=
DETECT_POLL_INTERVAL;

            /* copy current value to old value variables */
            link[ i] .old_detector[ j]  = link[ i] .detector[ j] ;
      }
}

/* ------------------------------------------------
** atox
** : read a string as a hexdecimal
** ------------------------------------------------ */
int atox(char *s)
{
      int sum = 0;
      for (; *s; s++) {
            if ( *s >= '0' && *s <= '9')
                  sum = sum * 16 + *s - '0';
            else if ( *s >= 'a' && *s <= 'f')
                  sum = sum * 16 + *s - 'a' + 10;
            else if ( *s >= 'A' && *s <= 'F')
                  sum = sum * 16 + *s - 'F' + 10;
      }
      return sum;
}

/* ------------------------------------------------
** set_signal_registry_value
** ------------------------------------------------ */
void set_signal_registry_value(char *signal_value)
{
      int i;
      char s[ 3] ;

      /* convert character to hexadecimal value */
      for (i=0; i<LEN_IO; i++)
      {
            strncpy(s, signal_value + i*2, 2);
            s[ 2]  = '\0';
            ctrl_sig_buf[ i]  = atox(s);
      }

      /* send data from 2070 controller to PC */
      fio_sig_outputs(fio_dev, &ctrl_sig_buf, 0);
}

/* ------------------------------------------------
** decode_detector_data
```

```
** ------------------------------------------------- */
void decode_detector_data()
{
      int i, j;
      int element;
      int temp;
      int converted_value;

      /* read input signal */
      fio_sig_inputs(fio_dev, &detector_buf, 0);

      /*************************************************************************
********/
      /* simulation termination test */
      /*************************************************************************
********/
      temp = 1;
      for (i=0; i<LEN_IO; i++)
      {
            /* this line will convert the input signal to the correct value */
            element = 255 - detector_buf[ i] ;

            if (element != 255)
            {
                  temp = 0;
                  break;
            }
      }

      /* if all values are 255, then termiate simulation. */
      if (temp == 1)
            terminated = TRUE;

      /*************************************************************************
********/
      /* data conversion
      /*************************************************************************
********/
      /* each element of detector_buf has the integer value of 0 ~ 255 */
      /* the integer value will be converted to binary format */
      /* for example, it one element is 7, it is 00000111*/
      /* therefore, each element can represent 8 detectors by binary format */
      /* the total possible number of detectors is 8 * 8 = 64 */

      /* this loop is for WEST, SOUTH, EAST, NORTH */
      for (i=0; i<LINK_NO; i++)
      {
            element = 255 - detector_buf[ i] ;

            /* this loop is for decoding decimal into binary format */
            for (j=0; j<8; j++)
            {
                  temp = element % 2;
                  element = (element - temp) / 2;
                  link[ i] .detector[ 7 - j]  = temp;

            }
```

```
	}

	/*************************************************************************
********/
	/* display detector data for test
	/*************************************************************************
********/
	/*
	temp = 0;
	for (i=0; i<LINK_NO; i++)
	{
		if (detector_buf[ i]  != detector_buf_old[ i] )
		{
			temp = 1;
			break;
		}
	}

	if (temp == 1)
	{
		for (i=0; i<LINK_NO; i++)
		{
			element = 255 - detector_buf[ i] ;

			if (element < 10)
				printf("00%d", element);
			else if (element < 100)
				printf("0%d", element);
			else
				printf("%d", element);

			detector_buf_old[ i]  = detector_buf[ i] ;
		}
		printf("\n");

		for (i=0; i<LINK_NO; i++)
		{
			for (j=0; j<8; j++)
				printf("%d", link[ i] .detector[ j] );

			printf(" -- ");
		}
		printf("\n");
	}
	*/
}

/* -------------------------------------------------
** increase_current_stage_index
** ------------------------------------------------- */
void increase_current_stage_index()
{
	/* increase current stage index */
	if (current_stage_index == (stage_seq_no - 1))
		current_stage_index = 0;
	else
		current_stage_index++;
```

```
}

/* ------------------------------------------------
** report_volocc
** : report volume every VOLOCC_REPORT_INTERVAL ticks  (set to 1 min)
** ------------------------------------------------ */
void report_volocc()
{
      int i;
      int j;
      int volume;
      double occupancy;
      double speed;
      char buffer[ LEN_BUF] ;
      char temp[ LEN_BUF] ;

      sprintf(buffer, "%03d min\t\t", volocc_count++);

      for (i=0; i<LINK_NO; i++)
            for (j=0; j<DETECTOR_NO_IN_LINK; j++)
            {
                  volume = link[ i] .volume[ j] ;
                  occupancy = link[ i] .accumulated_occupancy_time[ j]  /
VOLOCC_REPORT_INTERVAL;

                  /* calculation of speed */
                  /* the length of standard loop detector is 1.8 meter */
                  /* speed(m/s) = 1.8 * volume / occupied time */
                  if (link[ i] .accumulated_occupancy_time[ j]  != 0)
                        speed = (1.8 * volume) /
(link[ i] .accumulated_occupancy_time[ j]  / 120);
                  else
                        speed = 0.0;

                  /* write the result onto the result file */
                  sprintf(temp, "\t%03d\t%4.1f\t%4.1f\t", volume, occupancy,
speed);
                  strcat(buffer, temp);

                  link[ i] .volume[ j]  = 0;
                  link[ i] .accumulated_occupancy_time[ j]  = 0;
            }

      strcat(buffer, "\n");
      fputs(buffer, result_file);

}

/* ------------------------------------------------
** set_next_stage
** : determine whether the signal sign must be extended.
** : change the signal according to the determination.
** ------------------------------------------------ */
void set_next_stage()
{
      char registry_value[ LEN_REGISTRY + 1] ;
      int extension_interval;
```

```
	int temp;
	int i;
	int j;

	/* when simulation is started, */
	/* the process goes into this block only one time */
	if (started == TRUE)
	{
		current_stage_index = 0;
		extension_interval = stage_seq[ current_stage_index] .min_duration;
		total_extension_interval = extension_interval;
		started = FALSE;
		temp = 0;
	}
	else
	{
		/* extension must be considered  */
		/* when extension type is fixed or extension interval is 0,
		 then always go to the next stage. */
		if ((extension_type == FIXED) ||
			(stage_seq[ current_stage_index] .ext_interval == 0))
		{
			increase_current_stage_index();
			extension_interval =
stage_seq[ current_stage_index] .min_duration;
			total_extension_interval = extension_interval;
		}

		/* when extension type is actuated or adaptive */
		else
		{
			switch (extension_type)
			{
				case ACTUATED:
					temp = extension_by_actuated_control();
					break;
				case ADAPTIVE:
					temp = extension_by_adaptive_control();
					break;
				default: temp = 0;
			}

			/* if the return value temp is greater than 0 and
			   total_extension_interval is less than max_duration,
			   current stage must be extended. */
			if ((temp > 0) &&
			(total_extension_interval <
stage_seq[ current_stage_index] .max_duration))
			{
				extension_interval = temp;
				total_extension_interval += temp;
			}

			/* else go to the next stage */
			else
			{
				increase_current_stage_index();
```

```
                        extension_interval =
stage_seq[current_stage_index].min_duration;
                        total_extension_interval = extension_interval;

                        /* re-initialize adaptive volume variable */
                        for (i=0; i<LINK_NO; i++)
                              for (j=0; j<DETECTOR_NO_IN_LINK; j++)
                        {
                              link[i].volume_for_adaptive[j]  = 0;
                        }
                  }
            }
      }

#ifdef DEBUG
      if ((temp > 0) &&
            (total_extension_interval <
stage_seq[current_stage_index].max_duration))
            printf("looping--> [stage%2d]-[%2d]-[%2d]-extended\n",
            stage_seq[current_stage_index].stageid, extension_interval,
total_extension_interval);
      else
            printf("looping--> [stage%2d]-[%2d]-[%2d]\n",
            stage_seq[current_stage_index].stageid, extension_interval,
total_extension_interval);
#endif

      /* look up the current stage registry value */
      lookup_stage_registry_value(stage_seq[current_stage_index].stageid,
registry_value);

      /* change the signal value */
      set_signal_registry_value(registry_value);

      /* reset timer according to the new duration time */
      /* convert min_duration (second) to ticks */
      _os_alarm_set(&stage_change_alarmid, STAGE_CHANGE_ALARM,
            (u_int32)(extension_interval * 120));
}
```

APPENDIX D
Simulation Procedure for Intersection Control Strategies with Signal Operations Laboratory

APPENDIX D: Simulation Procedure for Intersection Control Strategies with Signal Operations Laboratory

D.1 Operating procedure for 2070 Controller from Windows NT PC

The 2070 ATC Control program is written in the C programming language, which is compiled with the FasTrak cross-platform compiler.

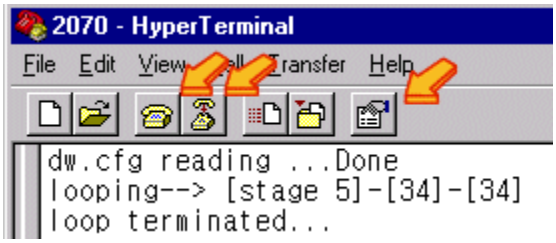
Ex> C:\Mwos\DOS\bin\os9make -o -f "2070.mak"

Once compiled, the executable file can be uploaded into the 2070 Controller using Hyper Terminal.

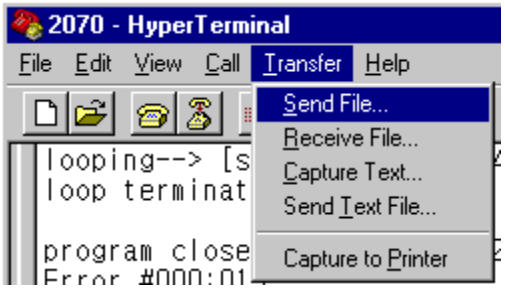
Procedure for booting up the OS/9 operating system and starting simulation.

OS/9:

- Turn on the power switch inside the 2070 Controller.
- Execute Hyper Terminal program connected with COM1 or COM2.
- Initial baud rate for OS/9 is 9600 bps.
- For fast uploading the executable file, change the baud rate to 38400 bps.
  Ex> $ tmode baud=38400
- Change the Hyper Terminal Setting to 38400 and disconnect and connect.
-



- Execute the field I/O manager interface, 'fioman', as background.
  Ex> $ fioman &
- Change directory to f0
  Ex> $ chd /f0
- Delete existing 2070 control program
  Ex> $ del 2070
- Upload 2070 Control program using kermit mode.
  Ex> $ kermit rdi8



- Change the 2070 program to executable mode.
  Ex> $ attr –pe –e 2070

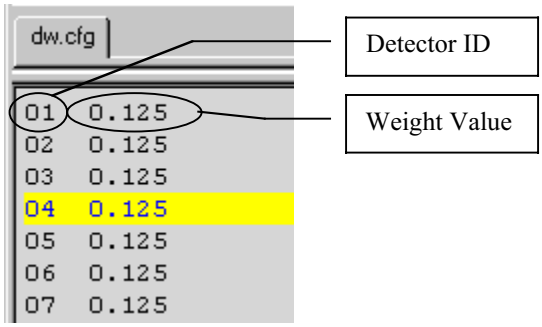
Windows:
- Execute TrafficAnalysys.exe and InterfaceThread.exe in Windows NT.
- Start traffic simulation corresponding to the first control stage.

Starting the simulation:
- Execute 2070 program. (Last argument: 1=FIXED, 2=ACTUATED, 3=ADAPTIVE)
  Ex>$ 2070 test1 1

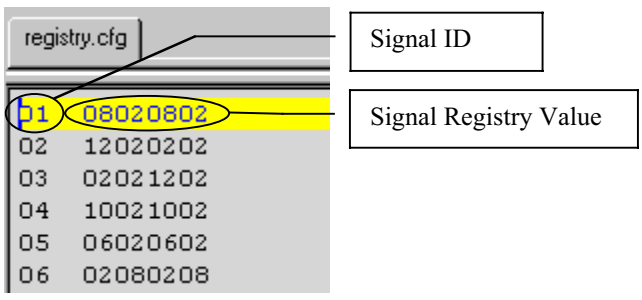
D.2 Configuration file format for 2070 control application programs

- To upload text file into the 2070 controller.
  Ex> $ kermit rd8

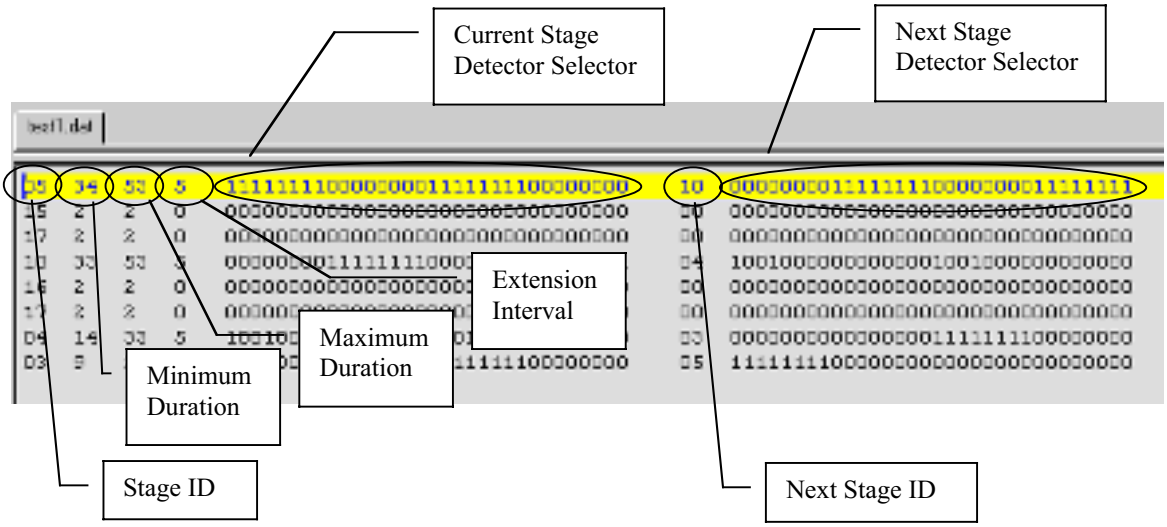
- dw.cfg (detector weight configuration file)
  In the example application, each link has 8 detectors, which have.
  the same weight value (1/8 = 0.125) in below example.



- registry.cfg
  Each signal registry value is predefined in the traffic simulation program.
  Public Const RedSign = 2
  Public Const YellowSign = 4
  Public Const UnprotectedGreenSign = 6
  Public Const OnlyGoThru = 8
  Public Const LeftTurnSign = 10
  Public Const ProtectedGreenSign = 12



- test1.dat (Data file for Example Application of Adaptive Strategy)
  Minimum Duration: The minimum amount of time to stay in this stage.
  Maximum Duration: The maximum amount of time to stay in this stage.
  Extension Interval: The amount of time to extend at each signal extension.
  Detector Selector: Value 1 is calculated. Value 0 is skipped.
  (In first row of below example, all west, east link detectors are calculated. (11111111))
  (The order of four links is West, South, East, North.)



D.3 Control strategies implemented with 2070 controller

Fixed Control

- There is no extension according to the traffic situation.
- Therefore, each stage is executed with minimum duration.

Actuated Control

- Stage extension is applied according to the detectors located at the end of each link, i.e., stop-line.
- At least one vehicle is on the detector at the end of one link when next stage is calculated, current stage will be extended.

Adaptive Control

- Stage extension is based on the adaptive control theory.
- This approach can apply overall traffic situation to signal control.
- Adaptive Control Theory
  - $C_{i,t} = \sum \_{i,j} D_j$
    Where $C_{i,t}$ = Congestion index for approach or link I at the end of time interval t.
    $\_{i,j}$ = Weighting parameter for detector location j at approach i.
    $D_j = (P_j + V_j) / (1 + V_j)$

where

P_j = 1.0; if detector j is occupied at the end of time interval t,

= 0.0; otherwise

V_j = number of vehicles crossing detector j during time interval t.

- Compare current stage and next stage. If current stage's congestion index is greater than next stage's congestion index, current stage will be extended.

D.4 Data and configuration files for example simulation of control strategies

The data files for three control strategies, i.e., FIXED, ACTUATED, ADAPTIVE, that are evaluated in this research are as follows:

- dw.cfg

```
01  0.125
02  0.125
03  0.125
04  0.125
05  0.125
06  0.125
07  0.125
08  0.125
09  0.125
10  0.125
11  0.125
12  0.125
13  0.125
14  0.125
15  0.125
16  0.125
17  0.125
18  0.125
19  0.125
20  0.125
21  0.125
22  0.125
23  0.125
24  0.125
25  0.125
26  0.125
27  0.125
28  0.125
29  0.125
30  0.125
31  0.125
32  0.125
```

- registry.cfg

```
01  08020802
02  12020202
03  02021202
04  10021002
05  06020602
06  02080208
07  02120202
08  02020212
09  02100210
10  02060206
11  04020202
12  02040202
13  02020402
14  02020204
15  04020402
16  02040204
17  02020202
```

- test1.dat

```
05  34      53      5       11111111000000001111111100000000        10
    00000000111111110000000011111111
15  2       2       0       00000000000000000000000000000000        00
    00000000000000000000000000000000
17  2       2       0       00000000000000000000000000000000        00
    00000000000000000000000000000000
10  33      53      5       00000000111111110000000011111111        04
    10010000000000001001000000000000
16  2       2       0       00000000000000000000000000000000        00
    00000000000000000000000000000000
17  2       2       0       00000000000000000000000000000000        00
    00000000000000000000000000000000
04  14      33      5       10010000000000001001000000000000        03
    00000000000000001111111100000000
03  9       28      5       00000000000000001111111100000000        05
    11111111000000000000000000000000
```

D.5 Frequently used OS/9 and FasTrack Commands

- **OS-9 command**

```
$command -? : command syntax
$tmode baud=38400 : change the communication baudrate
$chd /f0 : change directory
$del 2070 : delete file
$kermit rdi8 : for uploading binary file
```

$kermit rd8 : for uploading text file
$kermit sdi8 : for downloading binary file
$kermit sd8 : for downloading text file
$attr -pe -e 2070 : change the attribute to executable
$dir -e : list up file and directory in current directory
$pd : display the presently working directory
$procs : list up currently running process
$kill 12 : kill the process id 12
$fioman& : backgroudn execusion of command
$mfree -e : display the amount of unused RAM
$copy file newfile : copy file to new file
$makdir abc : make abc directory
$deldir abc : delete abc directory

Tip: 1 second = 120 ticks

- **FasTrack Commands for Source Code Compiling**

  - Makefile must be made before compiling
    Start -> Programs -> FasTrak -> Makefile Editor
    Adjust setting
      a. Mode: Extended ANSI C
      b. Processor Family: 68X
      c. Chip: MC68000/08/10/12/70
      d. Settings: Data REferences
      e. Options: 16 Bit
      f. Debug: Source Level
      g. Stack Checking: Checked
      h. Link Type: O-Code
      i. Link With: Standard
      j. Directories for Includes:
        C:\ATC2070\INCLUDE
        C:\MWOS\SRC\DEFS
        C:\MWOS\OS9\SRC\DEFS
      k. Directories for Default Libraries:
        C:\ATC2070\INCLUDE
        C:\MWOS\SRC\DEFS
        C:\MWOS\OS9\SRC\DEFS
    Save as "2070.mak".
    Generate All
  - Execute command window. (Run "cmd")
  - Change the directory to the current directory
    which contains 2070.c source code.
  - Run C:\Mwos\DOS\bin\os9make -o -f "2070.mak"
    Or You can add Path setting.
  - If compiling is successful, 2070 binary code will be created.

APPENDIX E
Preliminary Design for the Network Analysis System and Event-driven Microscopic Simulator

E-1. Overview of Dynamic Analysis System for Traffic Networks

This section summarizes the preliminary design results to develop a Dynamic Analysis System for Traffic Networks (DASTN), which can be used as the graphical user interface for the network-based signal operations research laboratory. The main goal of the DASTN is to provide an integrated graphical environment, where preparation for traffic simulation and analysis for network performance can be efficiently managed. Figure C.1 shows the framework of the proposed system, whose main modules include a graphical network editor, traffic database, data analysis and network performance evaluation modules. The proposed system adopts an object-oriented approach in modeling traffic networks and utilizes an object-oriented database, POET, which can be integrated into a C++ program. Due to the time and budget limitation, only the design of the main framework and basic traffic objects were conducted in this research.

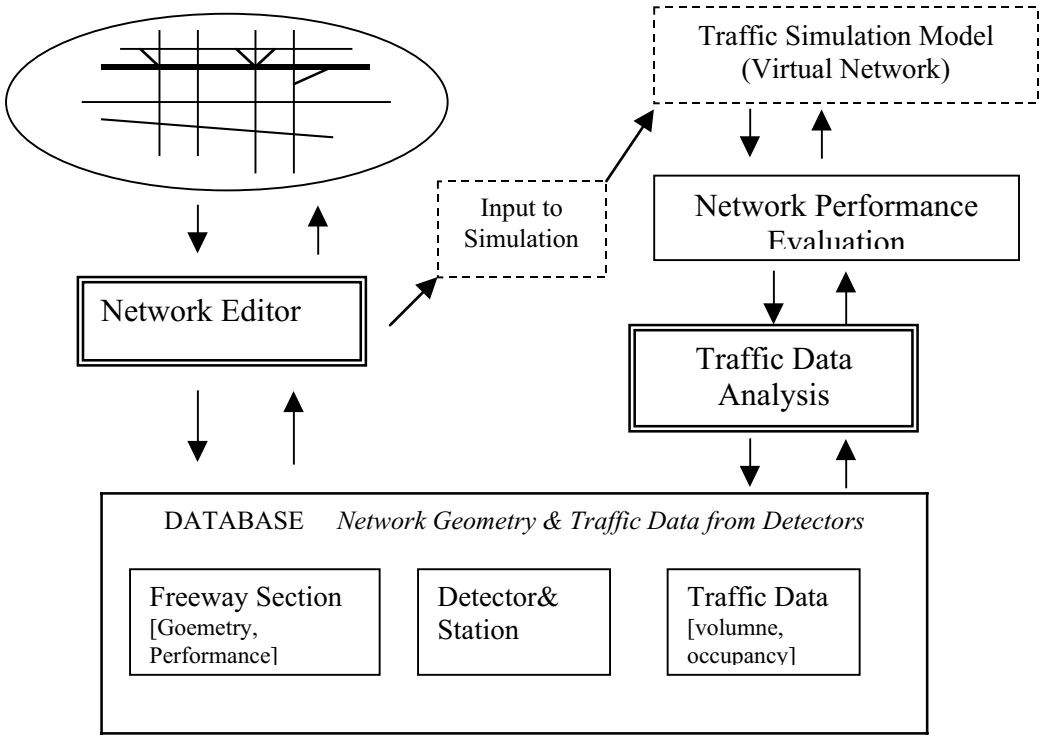
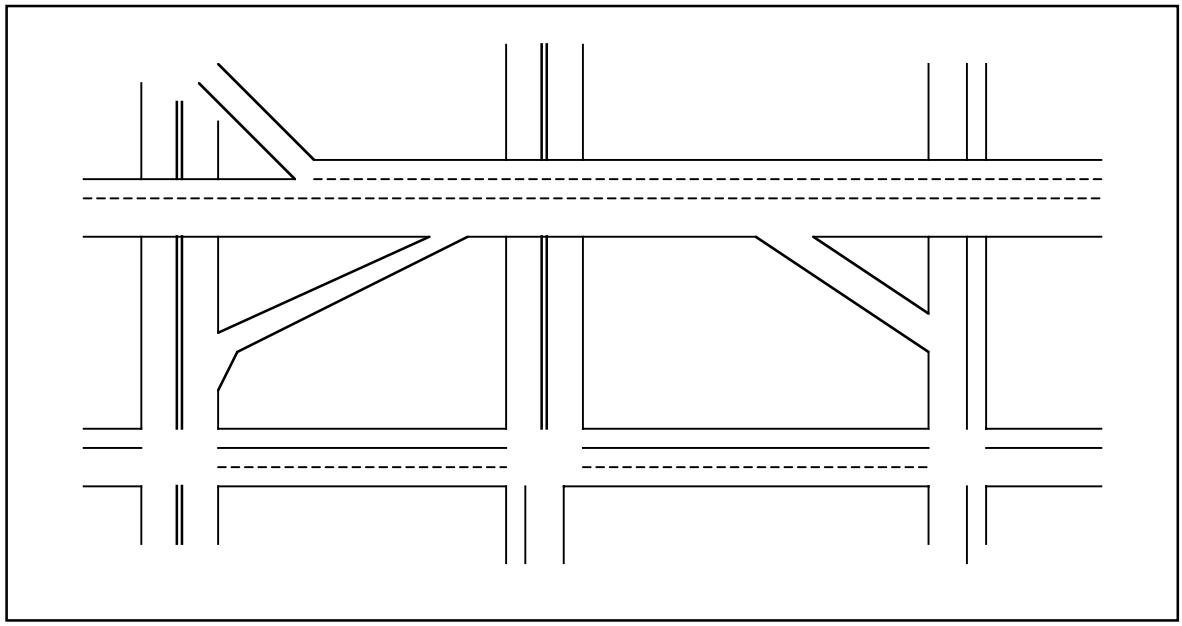


Figure E.1 Framework for Dynamic Network Analysis System

E-2 Development of Network and Traffic Objects

E-2-1 Definition of Objects

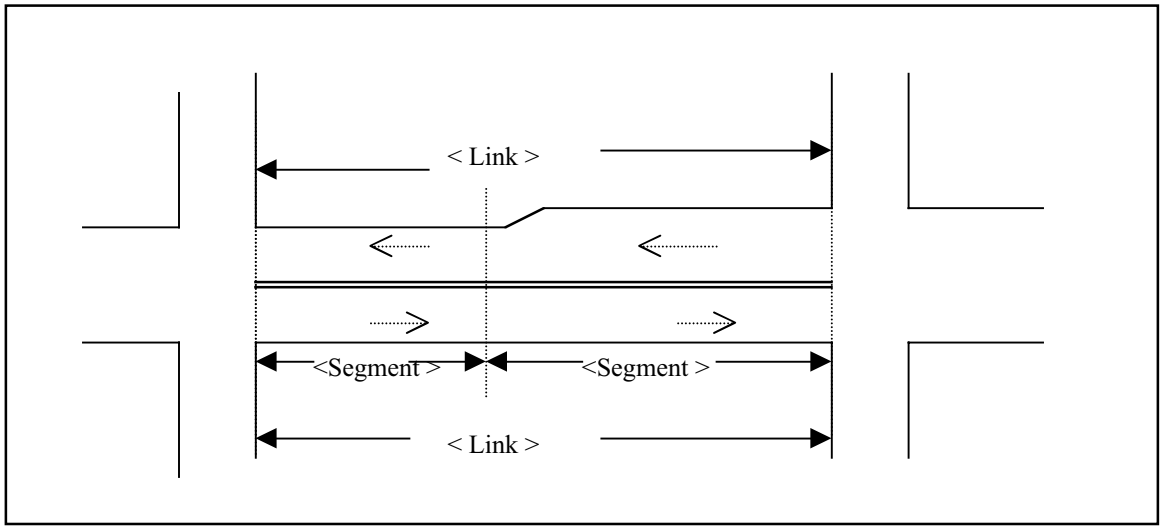
Network
Network is a collection of data representing specific real or virtual world of road where all kind of traffic activities like vehicle movement signal change take place. Network object is consisted of Segment, Node and other traffic Control object.



Example of Network (with Highway and Local Street)

Link Collection of connected segments in same direction between two adjacent nodes.

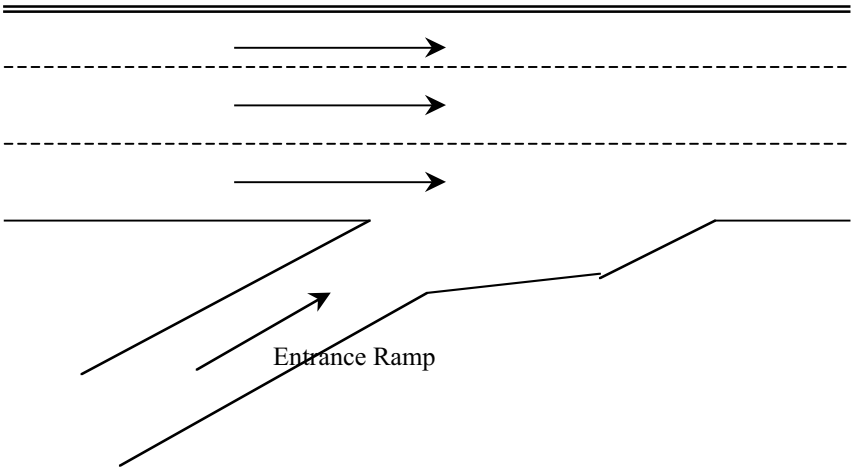
Segment Unit road element with uniform geometry and/or common traffic characteristics.



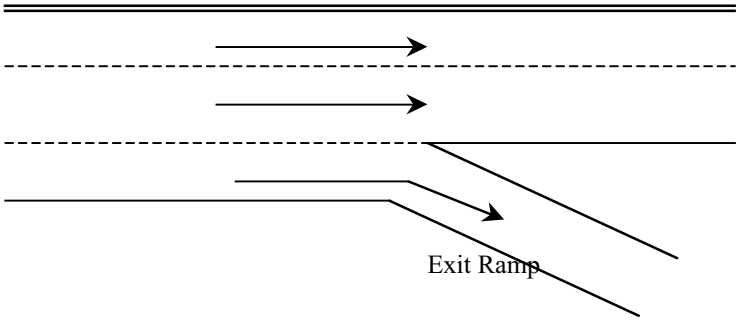
Link and Segment

Freeway Node

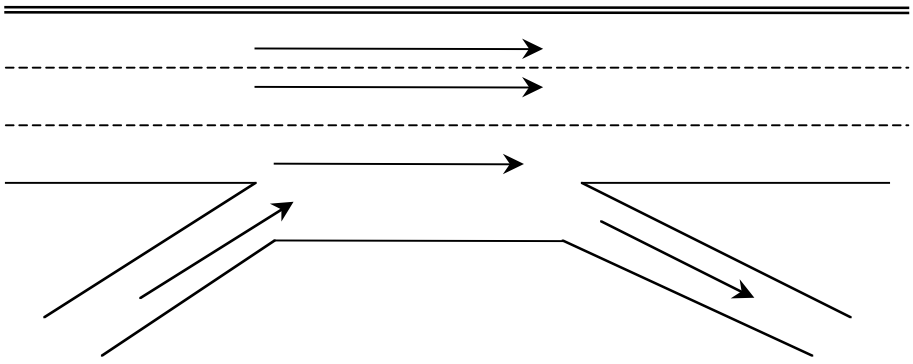
On-Ram, *Off-Ramp* and *Weave Nodes*



On-Ramp Node

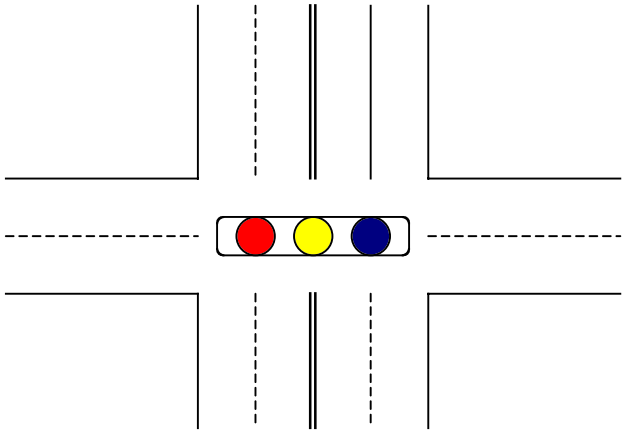


Off-Ramp Node



Weave Node

Local Node An Intersection area in local streets with Signal Object. Routing decisions take place in Local Nodes.



Local Node

Traffic Objects

Signal Control: Signal controller at intersections with timing plans
Traffic Control: Objects that control vehicle movements, e.g., Stop sign, Variable Message Sign, etc.
Detector: A device on the road counting vehicles and the time it is occupied by vehicles.
Detector Station: A group of detectors that are located at a same position of road but different lanes.

E-3. Detail Design of Geometric and Traffic Objects

E-3-1 Segment

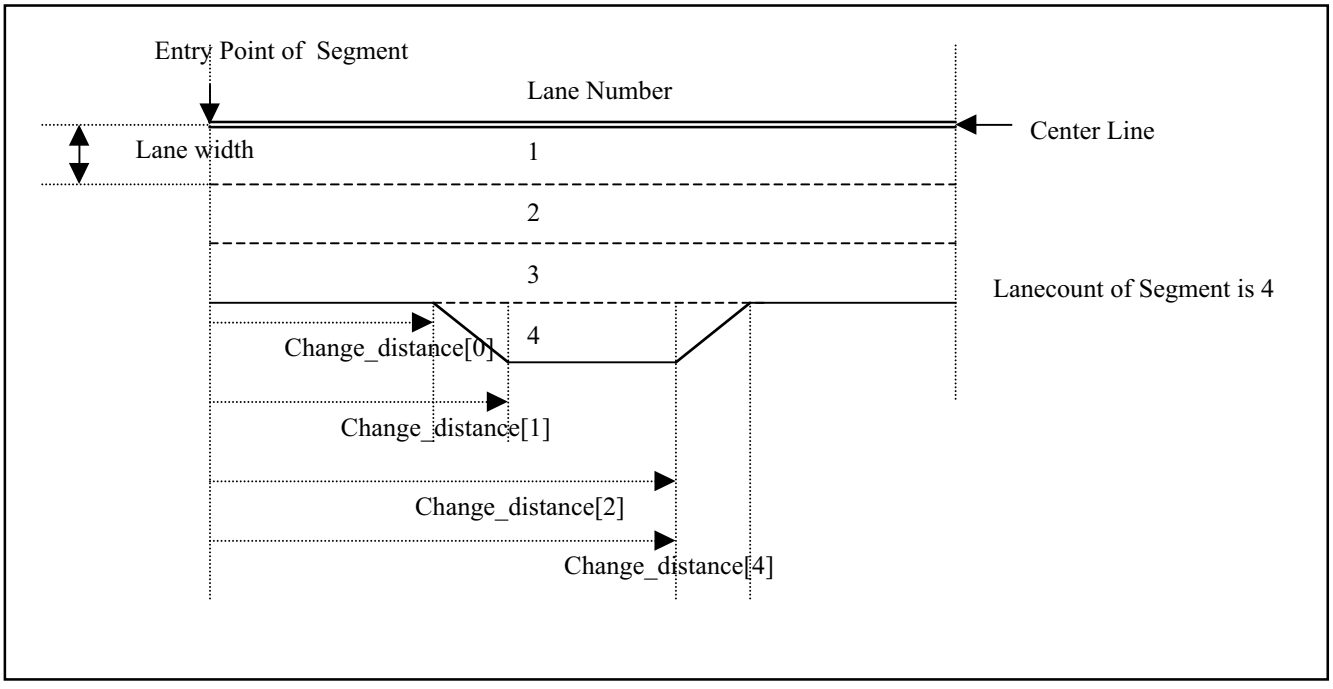
Attributes of Segment

- OBJECT_COLOR_ENUM shapecolor :
  Enum type data to draw on screen each object by differrent color

```
enum OBJECT_TYPE_ENUM
{
    segment,
    localnode,
    freewaynode,
    onrampnode,
    offrampnode,
};
```

- SEGMENT_STYLE shapestyle :
  enum type data to contain segment is on two way road or onw way road.

```
enum SEGMENT_STYLE
{
        oneway,
        twoway,
};
```



Attributes of Segment

- LANE laneArry[MAX_LANE];
  Array of LANE structure. Each lane has lane character and traffic contraint value
  And lane change point.

```
struct LANE
        {
        LANE_CHARACTER    character;
        int     lane_number;
        float    width;
        float    change_distance[MAX_CHANGE_POINT];
        //Traffic Constraint
        float       capacity;
        float       speed_limit;
        float       design_speed;
        };

        enum LANE_CHARACTER
        {
                Normal,
                HOV,
                Toll,
                HOT,
                Ex_Bus,
                Auxiliary,
        };
```

- cset<ondemand<Control>> controlList :
  cset of reference of Contro object. 'cset' type id provided by POET database.
  Set type can handle its reference by Get(), Append(),Delete() functions of POET.
  Ondemand<Control> is same as Control* (pointer). For the difference of pointer and ondemand refer 'POET Programmer's Guide' about Set.

- cset<Station*> stationList;
  cset of reference of Staion object.

- cset<Detector_struct> DetectorStructSet;
  cset of structure Detector_struct. Detector_struct has value only when Segment or FreewayNode has detectors without Station. The Segment or FreewayNode in highway has detectors with Station. In those case detectors are handled through cset<Station*> not through cset<Detector_struct>.

```
struct Detector_struct
{
public:
    Detector* pDet;
    int lane_number;
};
```

- PtString segmentID;
  Character value of Segment ID, segmentID is unique.
  PtString type is provide by POET and it is equivalent with CString in MFC of Visual C++.

PtString handles string whole value, not character by character like char*.

- PtString linkID;
  Link ID of the Segment belongs to.

- PtString streetName;
  Street name of the Segment belongs to.
  Ex) 'Snelling', 'University', '15th Ave'

- float entry_x;
- float entry_y;
  X-Y cordinate value of entry point of Segment.

- float exit_x;
- float exit_y;
  X-Y cordinate value of exit point of Segment

- int lanecount;
  Number of lane

- float length;
  The length from entry point toexit point of segment.

Functions of Segment

Contruct Function

```
Segment();
Segment(int lanecount, float x1, float y1, float x2, float y2);
```

Access Function

```
void SetEntry( float X1, float Y1){ entry_x=X1; entry_y=Y1; };
void SetExit( float X2, float Y2){ exit_x=X2; exit_y=Y2; };
void SetLaneCount( int lc){ lanecount=lc;};
void SetLength( float len ){ length=len; };
void SetSegmentID( PtString sid){ segmentID=sid; };
void SetLinkID( PtString lid){ linkID = lid; };
void SetStreetName( PtString st){ streetName = st; };
void SetArc( int a){ arc=a; };
float GetEntryX(){ return entry_x; };
float GetEntryY(){ return entry_y; };
float GetExitX(){ return exit_x; };
float GetExitY(){ return exit_y; };
PtString GetSegmentID(){ return segmentID; };
PtString GetLinkID(){ return linkID; };
int GetLaneCount(){ return lanecount; };
float GetLength(){ return length; };
```

E-3-2 FreewayNode

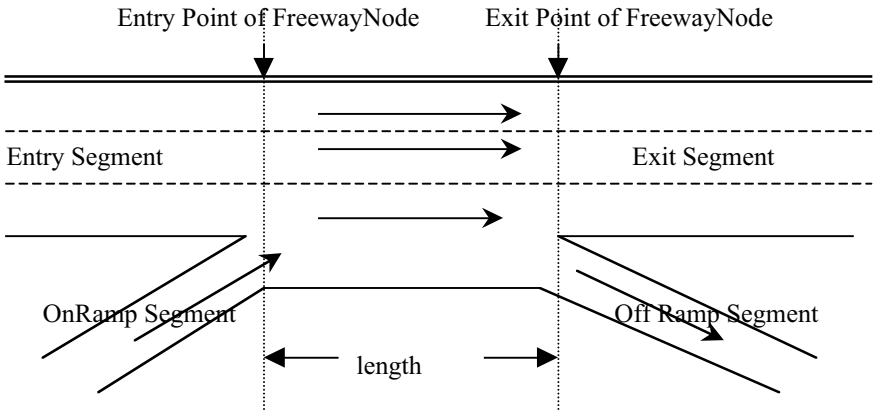
Attributes of FreewayNode

FreewayNode is almost same but it doesn't have linkID, streetName and Detector_struct.
It has pointer of entry segment ,exit segment and INTERSECTION_STYLE
It is a base object for OnRampNode, OffRampNode and WeaveNode.

- PtString FnodeID;
- INTERSECTION_STYLE style;
  Enum value of intersection type

- float entry_x;
- float entry_y;

- float exit_x;
- float exit_y;
- float length;
- int lanecount;
- LANE laneArry[MAX_LANE];
- Segment* entry_segment;
- Segment* exit_segment;



Attribute of FreewayNode,

Functions of Freeway Node

Contruct Function

FreewayNode();
FreewayNode(int lanecount, float x1, float y1, float x2, float y2);

Access Function

void SetEntry(float X1, float Y1){ entry_x=X1; entry_y=Y1; };

```
void SetExit( float X2, float Y2){ exit_x=X2; exit_y=Y2; };
void SetLaneCount( int lc){ lanecount=lc; };
void SetLength( float len ){ length=len; };
void SetFnodeID( PtString fid){ FnodeID = fid; };
float GetEntryX(){ return entry_x; };
float GetEntryY(){ return entry_y; };
float GetExitX(){ return exit_x; };
float GetExitY(){ return exit_y; };
PtString GetFnodeID(){ return FnodeID; };
int GetLaneCount(){ return lanecount; };
float GetLength(){ return length; };
```

On RampNode, Off RampNode, Weave Node – Inherited from Freeway Node

OnRampNode

```
persistent class OnRampNode : public FreewayNode
{
public:
        //Reference to entrance ramp segment
        Segment* on_ramp_segment;

        //Construct Function
        OnRampNode();
        OnRampNode(int lanecount, float x1, float y1, float x2, float y2);

        //Deconstruct Function
        virtual ~OnRampNode();
};
```

OffRampNode

```
persistent class OffRampNode : public FreewayNode
{
public:
        //Reference to entrance ramp segment
        Segment* off_ramp_segment;

        //Construct Function
        OffRampNode();
        OffRampNode(int lanecount, float x1, float y1, float x2, float y2);

        //Deconstruct Function
        virtual ~OffRampNode();
};
```

WeaveNode

```
persistent class WeaveNode : public FreewayNode
{
public:
```

```
        Segment* on_ramp_segment;
        Segment* off_ramp_segment;
        WeaveNode();
        WeaveNode(int lanecount, float x1, float y1, float x2, float y2);
        virtual ~WeaveNode();
};
```

E-3-3 Local Node

Attributes and Functions of Local Node Object

LocalNode is a object which conect more than three segment. It has its own geometry area and vehicle movement.

```
persistent class LocalNode
{
public:
        PtString  LnodeID;
        INTERSECTION_STYLE style;

        float center_x;
        float center_y;
        float WS_point_x;
        float WS_point_y;

        cset<ondemand<Segment>> segmentlList;
    SignalTimingPlan  signal;
        void Remove();
        LocalNode();
        LocalNode( float x1, float y1);
        virtual ~LocalNode();
};
```

- PtString LnodeID;
  Character value of LocalNode ID

- INTERSECTION_STYLE style;
  Enum value of intersection type

```
        enum INTERSECTION_STYLE
        {
                three_way,
                four_way,
                five_way,
                merge_way,
                split_way,
                merge_split_way,};
```

- float center_x;
- float center_y;
  X-Y cordinate value of center point.

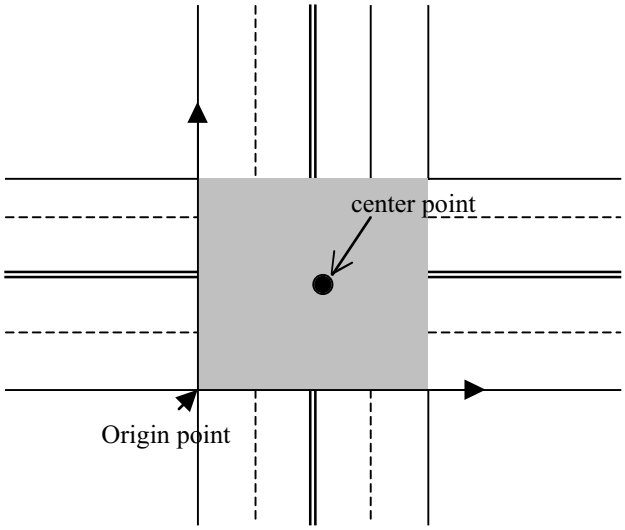
- float WS_point_x;
- float WS_point_y;
  X-Y cordinate value of origin point in LocalNoce

- cset<ondemand > segmentlList;
  Reference of segment which are linked to localNode. It includes both ongoing segment and outgoing segment.
  The number of segments linked is variable, it depends intersection style(three way, four way) and segment style(oneway,twoway road) .

- SignalTimingPlan signal;
  SignalTimingPlan object control vehicle movement at LocalNode(intersection).
  Refer 5.1 Signal Control for more detail.

- void Remove();
  delete itself function

- LocalNode();
- LocalNode(float x1, float y1);
- virtual ~LocalNode();
  Contruct/Decontruct function.



Attribute of Local Node – fourway style

E-3-4 Link

Attributes and Functions of Link Object

```
persistent class Link
{
public:
        //Contruct function
        Link();
```

```
        //Set LinkID function
        void SetLinkID(PtString lid){ linkID=lid; };

        //Add reference of segment to segmentList
        void AddSegment(Segment* pSegment);

private:
        PtString  linkID;

        //Reference of segment which are in this link
        lset<ondemand<Segment>> segmentList;
};
```

E-3-5 Network

Network Object is a unit for simulation and a consistent view of road for User.

Attributes and Functions of Network Object

```
persistent class Network
{
public:
        lset<Segment*> segmentList;
        lset<FreewayNode*> nodeList;
        Network();
        Network(PtString nID);
        ~Network();
        PtString getID(){ return networkID; };
private:
        PtString  networkID;
};
```

```
lset<Segment*> segmentList;
        lset of all Segment pointer which are in this network

lset<FreewayNode*> nodeList;
        lset of all Freeway pointer which are in this network

Network();
Network(PtString nID);
        Consruct function

PtString getID(){ return networkID; };
        Access function

PtString  networkID;
        Character value of Network ID, networkID is unique in all networks in same
        database.
```

E-3-6 Signal Control Object

SIGNAL_PHASE

```
struct SIGNAL_PHASE
{
        int phase_number;
        int SW,SN,SE,NE,NS,NW,WN,WE,WS,ES,EW,EN;
        int min_duration;
        int max_duration;
};
```

phase_number
: integer value, it is unique and iterate within a SignalTimingPlan like as phase#1,phase#2,,,,

int SW,SN,SE,NE,NS,NW,WN,WE,WS,ES,EW,EN;
: Vehicle passing is allowd if true.
Ex) SW == TRUE → Vehicle passing from South to West is allowed.
NE == TRUE → Vehicle passing from Nouth to East is allowed.
EW == FALSE → Vehicle passing from East to West is not allowed

int min_duration;
: Signal phase(a consistent moment of light) cannot be less than min_duration. Unit is a second.

int max_duration;
: Signal phase can not exceed more than max_duration. Unit is a second.

SignalTimingPlan Object

```
persistent class SignalTimingPlan
{
public:
        SignalTimingPlan();
        ~SignalTimingPlan();
private:
        PtString  control_type;
        int number_of_pahse;
        SIGNAL_PHASE  phases[MAX_PHASE_NUM];
        int cycle_length;
};
```

PtString control_type;
: Character value of signal control type
Ex) “Permitted”,”Actuated”, “Adaptive-A”, etc.

int number_of_pahse;
: total number of SIGNAL_PHASE

SIGNAL_PHASE phases[MAX_PHASE_NUM];
: Array of SIGNAL_PHASE

E-3-7 Traffic Control Object

Attributes of Control Object

```
persistent class Control
{
public:
   CONTROL_DEVICE_TYPE  device;
   float              distance;

            Control();
            virtual ~Control();

};
```

CONTROL_DEVICE_TYPE device;

Enum type value of Control type.

```
enum CONTROL_DEVICE_TYPE
{
            StopSign,
            VMS,
            LaneChangeRst,
            NormalMeter,
        HOVMeter,
};
```

E-3-8 Detector and Station Objects

Attributes and functions of Detector Object

```
persistent class Detector
{
public:
// Constructor
        Detector(unsigned long id, bool real = true) :
                _id(id),
                _mySeg((Segment*)0),
                _myLNode((LocalNode*)0),
                _myFNode((FreewayNode*)0),
                _real(real) {}

// Other public functions
        unsigned long getId() { return _id; }
        void addDayStat(DetectorDayStat* dayStat);
        void addDescription(Description& description);
        void addStation(Station* station);
        void reportDailyPeakVolumes();
        QueryResponse getPeak(PtDate queryDate, int startHour, int stopHour,
                unsigned long* peakHour, unsigned long* peakVolume, unsigned long* dailyVolume);
        static void getDetector(PtBase* db, unsigned long id, Detector*& detector);
```

```
        virtual ~Detector();
// public data
        cset<Description> _descriptions;

private:
// Functions
        Detector();
        Detector(const Detector& other);
        Detector& operator=(const Detector& other);
//Data
        unsigned long _id;
        Segment* _mySeg;
        LocalNode* _myLNode;
        FreewayNode* _myFNode;
        depend cset<ondemand<DetectorDayStat> > _dayStats;
        ondemand<Station> _station;
        bool _real;

        useindex DetectorIndex;
};
```

Attributes/functions of DetectorDayStat and DetectorStat Object

DetectorStat is a five minute interval data of volume and occupancy.
DetectordayStat is a one day worth of volume and occupancy data. DetectorDayStat has a set of DetectorStat.

```
_persistent class DetectorDayStat  // do not build an AllSet for DetectorDayStat
{
public:
        DetectorDayStat(const PtDate& id) : _id(id) {}
        void addStat(DetectorStat& stat);
        PtDate getId() { return _id; }
        void reportPeakVolume();
        void getPeakHourVolume(unsigned long* peakHour, unsigned long* peakVolume,
                                                                        unsigned long*
dailyVolume);

        virtual ~DetectorDayStat();

private:
        DetectorDayStat();
        DetectorDayStat(const DetectorDayStat& other);
        DetectorDayStat& operator=(const DetectorDayStat& other);

        PtDate _id;
        cset<DetectorStat> _stats;
};

class DetectorStat
{
public:
        DetectorStat() : _volume(0), _occupancy(0.0f), _statusAndFlag(0) {}
        DetectorStat(unsigned char volume, float occupancy, unsigned char statusAndFlag) :
```

```
            _volume(volume), _occupancy(occupancy), _statusAndFlag(statusAndFlag) {}
        DetectorStat(const DetectorStat& other)
        {
            _occupancy = other._occupancy;
            _volume = other._volume;
            _statusAndFlag = other._statusAndFlag;
        }
        DetectorStat& operator=(const DetectorStat& other)
        {
            if (this != &other)
            {
                _occupancy = other._occupancy;
                _volume = other._volume;
                _statusAndFlag = other._statusAndFlag;
            }

            return *this;
        }

        unsigned char getVolume() { return _volume; }
        float getOccupancy() { return _occupancy; }
        unsigned char getStatusAndFlag() { return _statusAndFlag; }

        virtual ~DetectorStat() {}

private:
        float _occupancy;
        unsigned char _volume;
        unsigned char _statusAndFlag;
};
```

Attributes and functions Description Object

```
class Description
{
public:
// Constructor
        Description();
        Description(char* typePtr, char* routePtr, char* dirPtr,
                                    float mile = 0, char* altRoutePtr = "", char* altDirPtr = "");
        Description(const Description& other);
        Description& operator=(const Description& other);
        void reportDescription();
        void getType(char* type){ strcpy(type, _type); }
        void getRouteID(char* routeID){ strcpy(routeID, _routeID); }
        void getAltRouteID(char* altRouteID){ strcpy(altRouteID, _altRouteID); }
        void getDir(char* dir){ strcpy(dir, _dir); }
        void getAltDir(char* altDir){ strcpy(altDir, _altDir); }
        float getMilepost()  { return _milepost; }
        bool isType(char* type);

//  Destructor
        virtual ~Description() {}
```

```
private:
//Data
        char _type[3];      //location of detector
        char _routeID[17];
        char _altRouteID[17];  //only type Q identifies 2 route names
        char _dir[6];  //usually EB,WB, etc. Reversible lanes can be EB/WB
        char _altDir[3];  //only type Q can have 2 directions
        float _milepost;    //optional
};
```

APPENDIX F
Preliminary Design of Event-drive Microscopic Intersection Simulator

APPENDIX F: Preliminary Design of Event-Driven Microscopic Intersection Simulator

The following code is the results from the preliminary design to develop an object-oriented, event-driven microscopic simulator for an intersection using the MODSIM simulation language.

```
(*
    DControlMedia.mod
*)

DEFINITION MODULE ControlMedia;
FROM GrpMod         IMPORT  BTreeObj;
FROM Observe        IMPORT  ObserveSetObj, ALL ObserveType;
FROM Image          IMPORT  ImageObj;
FROM GTypes         IMPORT  PointType, ALL ColorType;

CONST
    OnRedColor = Red;           {Color of Lamp when it is ON or OFF}
    OnYellowColor = Yellow;
    OnGreenColor = Green;
    OffRedColor = DarkRed;
    OffYellowColor = DarkYellow;
    OffGreenColor = DarkGreen;
    OutBackColor = Khaki;       {Other Colors of Traffic Signal Device}
    InBackColor = DarkGrey;
    OffColor = Black;           {Size factor of Traffic Lamp}
    SizeFactor = 0.3;
    MAXCONTROL = 64;    {Number of Control Signal for Eagle 2070}
                        {Need to Update for 2070N}
TYPE

LampObj = OBJECT(ImageObj);
    type : ObserveType;
    ASK METHOD SetType(IN m : ObserveType); {Set the Color and/or Direction}
    TELL METHOD SetOn;  {Real-time interface must use these functions }
    TELL METHOD SetOff; {to schedule the events of traffic lamp's ON and OFF}

END OBJECT;

(*************************************************************************
    Base Object for control sign, stopline, traffic signal and
    variable message
*************************************************************************)
ControlMediaObj = OBJECT(ImageObj);
    id : STRING;                    {Key to order}
    location : PointType;           {Local coordinate on the given section}
    displaylocation : PointType;    {Local coordinate on the given section}
    message : ObserveType;          {one message at a time}
    ASK  METHOD SetID(IN a : STRING);
    ASK  METHOD SetCount(IN n : INTEGER);   {Pure Virtual Function }
    ASK  METHOD Init(IN plocation, dlocation : PointType);
    ASK  METHOD InitMessage(IN sm : STRING);
    ASK  METHOD SetMessage(IN m : ObserveType);
END OBJECT;

(*************************************************************************
    Derived object for sign board
*************************************************************************)
ControlMediaSign = OBJECT(ControlMediaObj);
OVERRIDE
    ASK  METHOD InitMessage(IN sm : STRING);
END OBJECT;


(*************************************************************************
    Derived object for stopline
*************************************************************************)
ControlMediaStopLine = OBJECT(ControlMediaObj);
OVERRIDE
    ASK  METHOD Init(IN plocation, dlocation : PointType);
```

```
END OBJECT;

(*************************************************************************
    Derived object for traffic signal
*************************************************************************)
ControlMediaSignal = OBJECT(ControlMediaObj);
    numberoflamp : INTEGER;
    lamps : ARRAY INTEGER OF LampObj;
PRIVATE
    idx : INTEGER;
OVERRIDE
    ASK  METHOD SetCount(IN n : INTEGER);
    ASK  METHOD InitMessage(IN sm : STRING);
    ASK  METHOD SetMessage(IN m : ObserveType);
END OBJECT;

(*************************************************************************
    Derived object for variable message
*************************************************************************)
ControlMediaMessage = OBJECT(ControlMediaObj);
OVERRIDE
    ASK  METHOD SetMessage(IN m : ObserveType);
END OBJECT;

(*************************************************************************
    Collector for Control Media
*************************************************************************)
ControlMediaDbaseObj = OBJECT(BTreeObj[ ANYOBJ:ControlMediaObj] );
OVERRIDE
    ASK METHOD Rank (IN a, b : ControlMediaObj) : INTEGER ;
    ASK METHOD Key (IN a : ControlMediaObj) : STRING;
END OBJECT;

VAR
    Lamps : ARRAY INTEGER OF LampObj;
PROCEDURE InitLamps;
END MODULE.


(*
    IControlMedia.mod
*)

IMPLEMENTATION MODULE ControlMedia;
FROM MathMod        IMPORT  pi;
FROM Observe        IMPORT  ObserveSetObj, Str2Observe, ALL ObserveType;
FROM Image          IMPORT  ImageObj;
FROM Graphic        IMPORT  GraphicLibObj;
FROM Line           IMPORT  PolylineObj;
FROM Sect           IMPORT  NormalLineColor;
FROM GTypes         IMPORT  PointArrayType, ALL LineStyleType, ALL FillStyleType;
FROM Fill           IMPORT  PolygonObj, CircleObj;
FROM Util           IMPORT  StrTok;

TYPE
    ArrowType = (NoArrow, LeftArrow, RightArrow);

PROCEDURE CreateCircle(IN color : ColorType; IN arrow : ArrowType) : ImageObj;
VAR
    c : CircleObj;
    vc : ImageObj;
    a : PolygonObj;
    p, q : PointArrayType;
    S2 : REAL;
BEGIN
    NEW(vc);
    NEW(c);
    NEW(p, 1..2);
    p[ 1] .x := 0.0;                  p[ 1] .y := 0.0;
    p[ 2] .x := 3.0 * SizeFactor;     p[ 2] .y := 0.0;
    ASK c TO SetPoints(p);
```

```
    ASK c TO SetStyle(SolidFill);
    ASK vc TO AddGraphic(c);
    IF arrow = NoArrow
        ASK c TO SetColor(color);
    ELSE
        ASK c TO SetColor(OffColor);
        NEW(a);
        NEW(q, 1..7);
        S2 := 0.8;
        q[ 1].x := -5.0 * SizeFactor * S2;   q[ 1].y := 0.0;
        q[ 2].x := -1.0 * SizeFactor * S2;   q[ 2].y := -5.0 * SizeFactor * S2;
        q[ 3].x := -1.0 * SizeFactor * S2;   q[ 3].y := -3.0 * SizeFactor * S2;
        q[ 4].x :=  4.0 * SizeFactor * S2;   q[ 4].y := -3.0 * SizeFactor * S2;
        q[ 5].x :=  4.0 * SizeFactor * S2;   q[ 5].y :=  3.0 * SizeFactor * S2;
        q[ 6].x := -1.0 * SizeFactor * S2;   q[ 6].y :=  3.0 * SizeFactor * S2;
        q[ 7].x := -1.0 * SizeFactor * S2;   q[ 7].y :=  5.0 * SizeFactor * S2;
        ASK a TO SetPoints(q);
        ASK a TO SetColor(color);
        ASK a TO SetStyle(SolidFill);
        ASK vc TO AddGraphic(a);
        DISPOSE(q);
    END IF;
    DISPOSE(p);
    RETURN vc;
END PROCEDURE;

OBJECT LampObj;
    ASK METHOD SetType(IN m : ObserveType);
    VAR
        onlamp, offlamp : ImageObj;
    BEGIN
        type := m;
        CASE (type)
        WHEN RedSignal:
            onlamp  := CreateCircle(OnRedColor, NoArrow);
            offlamp := CreateCircle(OffRedColor, NoArrow);
        WHEN RedLeftSignal:
            onlamp  := CreateCircle(OnRedColor, LeftArrow);
            offlamp := CreateCircle(OffRedColor, LeftArrow);
        WHEN RedRightSignal:
            onlamp  := CreateCircle(OnRedColor, RightArrow);
            offlamp := CreateCircle(OffRedColor, RightArrow);
        WHEN YellowSignal:
            onlamp  := CreateCircle(OnYellowColor, NoArrow);
            offlamp := CreateCircle(OffYellowColor, NoArrow);
        WHEN GreenSignal:
            onlamp  := CreateCircle(OnGreenColor, NoArrow);
            offlamp := CreateCircle(OffGreenColor, NoArrow);
        WHEN GreenLeftSignal:
            onlamp  := CreateCircle(OnGreenColor, LeftArrow);
            offlamp := CreateCircle(OffGreenColor, LeftArrow);
        WHEN GreenRightSignal:
            onlamp  := CreateCircle(OnGreenColor, RightArrow);
            offlamp := CreateCircle(OffGreenColor, RightArrow);
        OTHERWISE
            OUTPUT("WARNING. Unknown Signal Type");
            RETURN;
        END CASE;
        ASK onlamp  TO SetHidden(TRUE);
        ASK offlamp TO SetHidden(FALSE);
        ASK SELF TO AddChild(onlamp,   "ON", 1);
        ASK SELF TO AddChild(offlamp, "OFF", 0);
    END METHOD;

    TELL METHOD SetOn;
    VAR
        onlamp, offlamp : ImageObj;
    BEGIN
        onlamp := ASK SELF TO Child("ON", 1);
        offlamp := ASK SELF TO Child("OFF", 0);
        IF (onlamp = NILOBJ) OR (offlamp = NILOBJ)
```

```
            OUTPUT("WARNING. CODE 120");
            RETURN;
        END IF;
        ASK onlamp TO SetHidden(FALSE);
        ASK offlamp TO SetHidden(TRUE);
        ASK SELF TO Draw();
    END METHOD;

    TELL METHOD SetOff;
    VAR
        onlamp, offlamp : ImageObj;
    BEGIN
        onlamp := ASK SELF TO Child("ON", 1);
        offlamp := ASK SELF TO Child("OFF", 0);
        IF (onlamp = NILOBJ) OR (offlamp = NILOBJ)
            OUTPUT("WARNING. CODE 120");
            RETURN;
        END IF;
        ASK onlamp TO SetHidden(TRUE);
        ASK offlamp TO SetHidden(FALSE);
        ASK SELF TO Draw();
    END METHOD;
END OBJECT;

OBJECT ControlMediaObj;
    ASK  METHOD SetID(IN a : STRING);
    BEGIN
        id := a;
    END METHOD;

    ASK  METHOD SetCount(IN n : INTEGER);   { Pure Virtual Function }
    BEGIN
        OUTPUT("WARNING! : ControlMediaObj::SetCount() is a pure virtual function.");
    END METHOD;


    ASK  METHOD Init(IN plocation, dlocation : PointType);
    BEGIN
        location := plocation;
        displaylocation := dlocation;
    END METHOD;

    ASK METHOD InitMessage(IN sm : STRING);
    BEGIN
        message := Str2Observe(sm);
    END METHOD;

    ASK  METHOD SetMessage(IN m : ObserveType);
    BEGIN
        message := m;
    END METHOD;

END OBJECT;

OBJECT ControlMediaSign;
    ASK METHOD InitMessage(IN sm : STRING);
    VAR
        lib : GraphicLibObj;
        imagename : STRING;
    BEGIN
        INHERITED InitMessage(sm);
        NEW(lib);
        ASK lib TO ReadFromFile("sign.sg2");
        LoadFromLibrary(lib, sm);
        DISPOSE(lib);
    END METHOD;
END OBJECT;

OBJECT ControlMediaStopLine;
    ASK  METHOD Init(IN plocation, dlocation : PointType);
    VAR
```

```
        line : PolylineObj;
        p : PointArrayType;
    BEGIN
        INHERITED Init(plocation, dlocation);
        NEW(line);
        NEW(p, 1..2);
        p[1].x := 0.0;
        p[1].y := 0.0;
        p[2].x := location.y - displaylocation.y;
        p[2].y := -(location.x - displaylocation.x);
        ASK line TO SetPoints(p);
        ASK line TO SetColor(NormalLineColor);
        ASK line TO SetStyle(SolidLine);
        ASK SELF TO AddGraphic(line);
        DISPOSE(p);
    END METHOD;
END OBJECT;

OBJECT ControlMediaSignal;
    ASK  METHOD SetCount(IN n : INTEGER);
    VAR
        r : PolygonObj;
        p : PointArrayType;
        sizex, sizey, gap : REAL;
    BEGIN
        numberoflamp := n;
        NEW(lamps, 1..n);
        sizex := SizeFactor * 16.0;
        sizey := SizeFactor * FLOAT(n+1) * 8.0;
        NEW(p, 1..4);
        NEW(r);
        p[1].x := 0.0;      p[1].y := 0.0;
        p[2].x := sizex;    p[2].y := 0.0;
        p[3].x := sizex;    p[3].y := sizey;
        p[4].x := 0.0;      p[4].y := sizey;
        ASK r TO SetPoints(p);
        ASK r TO SetStyle(SolidFill);
        ASK r TO SetColor(OutBackColor);
        ASK r TO SetRotation(-0.5 * pi);
        ASK SELF TO AddGraphic(r);

        NEW(r);
        gap := 2.0 * SizeFactor;
        p[1].x := gap;          p[1].y := gap;
        p[2].x := sizex - gap;  p[2].y := gap;
        p[3].x := sizex - gap;  p[3].y := sizey - gap;
        p[4].x := gap;          p[4].y := sizey - gap;
        ASK r TO SetPoints(p);
        ASK r TO SetStyle(SolidFill);
        ASK r TO SetColor(InBackColor);
        ASK r TO SetRotation(-0.5 * pi);
        ASK SELF TO AddGraphic(r);
    END METHOD;

    ASK METHOD InitMessage(IN sm : STRING);
    VAR
        ltype: STRING;
        addr : INTEGER;
        p : PointType;
    BEGIN
        ltype := StrTok(sm,",");
        addr := STRTOINT(sm);
        INC(idx);
        p.x := 8.0 * SizeFactor;
        p.y := FLOAT(numberoflamp - idx + 1) * 8.0 * SizeFactor;
        NEW(lamps[idx]);
        ASK lamps[idx] TO SetType(Str2Observe(ltype));
        ASK lamps[idx] TO SetTranslation(p.y, -p.x);
        ASK lamps[idx] TO SetRotation(-0.5 * pi);
        Lamps[addr] := lamps[idx];
        ASK SELF TO AddGraphic(lamps[idx]);
```

```
    END METHOD;

    ASK  METHOD SetMessage(IN m : ObserveType);
    VAR
        i : INTEGER;
    BEGIN
        FOR i := 1 TO numberoflamp
            IF lamps[ i] .type = m
                TELL lamps[ i]  TO SetOn;
            ELSE
                TELL lamps[ i]  TO SetOff;
            END IF;
        END FOR;
    END METHOD;
END OBJECT;

OBJECT ControlMediaMessage;
    ASK  METHOD SetMessage(IN m : ObserveType);
    BEGIN
    END METHOD;
END OBJECT;

OBJECT ControlMediaDbaseObj;
    ASK METHOD Rank (IN a, b : ControlMediaObj) : INTEGER ;
    BEGIN
        IF a.location.y < b.location.y
            RETURN -1;
        END IF;
        IF a.location.y > b.location.y
            RETURN 1;
        END IF;
        RETURN 0;
    END METHOD;

    ASK METHOD Key (IN a : ControlMediaObj) : STRING;
    BEGIN
        RETURN a.id;
    END METHOD;
END OBJECT;

PROCEDURE InitLamps;
VAR
    i : INTEGER;
BEGIN
    NEW(Lamps,0..MAXCONTROL-1);
    FOR i:=0 TO MAXCONTROL-1
        Lamps[ i]  := NILOBJ;
    END FOR;
END PROCEDURE;

END MODULE.


(*
    DDetector.mod
*)
DEFINITION MODULE Detector;
FROM GrpMod     IMPORT  BTreeObj;
FROM Image      IMPORT  ImageObj;
FROM GTypes     IMPORT  PointType, ALL FillStyleType, ALL ColorType;
FROM Fill       IMPORT  PolygonObj;

CONST
    DetectorFillStyle = MediumCrosshatchFill;
    DetectorColor     = Wheat;

TYPE
  DetectorStationObj = OBJECT; FORWARD;

  DetectorObj = OBJECT(ImageObj);
    id : STRING;
```

```
    location : PointType;
    width, height: REAL;
    station : DetectorStationObj;
  PRIVATE
    status : BOOLEAN;
    shape  : PolygonObj;
    ASK METHOD SetID(IN a : STRING);
    ASK METHOD SetLocation(IN p : PointType);
    ASK METHOD SetSize(IN w, h : REAL);
    ASK METHOD SetStation(IN u : DetectorStationObj);
    ASK METHOD BeginPassing;
    ASK METHOD EndPassing;
  OVERRIDE
    ASK METHOD  ObjInit();
  END OBJECT;

  DetectorArrayType = ARRAY INTEGER OF DetectorObj;

  DetectorDbaseObj = OBJECT(BTreeObj[ANYOBJ:DetectorObj]);
  OVERRIDE
    ASK METHOD Rank (IN a, b : DetectorObj) : INTEGER ;
    ASK METHOD Key (IN a : DetectorObj) : STRING;
  END OBJECT ;

  DetectorStationObj = OBJECT;
    id : STRING;
    detectors : DetectorArrayType;
    ASK METHOD SetID(IN s : STRING);
    ASK METHOD SetSize(IN n : INTEGER);
    ASK METHOD AddDetector(IN d : DetectorObj);
    ASK METHOD SetOn();
    ASK METHOD SetOff();
  PRIVATE
    idx : INTEGER;
    count : INTEGER;
  END OBJECT;

  DetectorStationDbaseObj = OBJECT(BTreeObj[ANYOBJ:DetectorStationObj]);
  OVERRIDE
    ASK METHOD Rank (IN a, b : DetectorStationObj) : INTEGER;
    ASK METHOD Key (IN a : DetectorStationObj) : STRING;
  END OBJECT;

END MODULE.


(*
    IDetector.mod
*)

IMPLEMENTATION MODULE Detector;
FROM GrpMod     IMPORT  BTreeObj;
FROM Image      IMPORT  ImageObj;
FROM GTypes     IMPORT  PointType, PointArrayType;

OBJECT DetectorObj;
    ASK METHOD  SetID(IN a : STRING);
    BEGIN
        id := a;
    END METHOD;

    ASK METHOD  SetLocation(IN p : PointType);
    BEGIN
        location := p;
    END METHOD;

    ASK METHOD  SetSize(IN w, h : REAL);
    VAR
        p : PointArrayType;
    BEGIN
        width  := w;
```

```
        height := h;
        NEW(p, 1..4);

        p[1].x := height / 2.0;     p[1].y := width / 2.0;
        p[2].x := height / 2.0;     p[2].y := -width / 2.0;
        p[3].x := -height / 2.0;    p[3].y := -width / 2.0;
        p[4].x := -height / 2.0;    p[4].y := width / 2.0;
        ASK shape TO SetPoints(p);
    END METHOD;

    ASK METHOD  SetStation(IN u : DetectorStationObj);
    BEGIN
        station := u;
    END METHOD;

    ASK METHOD BeginPassing;
    BEGIN
        ASK station TO SetOn;
    END METHOD;

    ASK METHOD EndPassing;
    BEGIN
        ASK station TO SetOff;
    END METHOD;

    ASK METHOD ObjInit();
    VAR

    BEGIN
        INHERITED ObjInit;
        status := FALSE;
        NEW(shape);
        ASK shape TO SetStyle(DetectorFillStyle);
        ASK shape TO SetColor(DetectorColor);
        AddGraphic(shape);
    END METHOD;
END OBJECT;

OBJECT DetectorDbaseObj;
    ASK METHOD Rank (IN a, b : DetectorObj) : INTEGER ;
    BEGIN
        IF a.location.y < b.location.y
            RETURN -1;
        END IF;
        IF a.location.y > b.location.y
            RETURN 1;
        END IF;
        RETURN 0;
    END METHOD;

    ASK METHOD Key (IN a : DetectorObj) : STRING;
    BEGIN
        RETURN a.id;
    END METHOD;
END OBJECT ;

OBJECT DetectorStationObj;
    ASK METHOD SetID(IN s : STRING);
    BEGIN
        id := s;
    END METHOD;

    ASK METHOD SetSize(IN n : INTEGER);
    BEGIN
        NEW(detectors, 1..n);
        idx := 0;
    END METHOD;

    ASK METHOD AddDetector(IN d : DetectorObj);
    BEGIN
        IF idx < HIGH(detectors)
```

```
            INC(idx);
            detectors[ idx]  := d;
            ASK d TO SetStation(SELF);
        ELSE
            OUTPUT("WARNING!. Overflow in DetectorStation");
        END IF;
    END METHOD;

    ASK METHOD SetOn;
    BEGIN
        IF count < HIGH(detectors)
            INC(count);
        ELSE
            OUTPUT("WARNING. Overflow in SetOn");
        END IF;
    END METHOD;

    ASK METHOD SetOff;
    BEGIN
        IF count > 0
            DEC(count);
        ELSE
            OUTPUT("WARNING. Underflow in SetOff");
        END IF;
    END METHOD;

END OBJECT;

OBJECT DetectorStationDbaseObj;
    ASK METHOD Rank (IN a, b : DetectorStationObj) : INTEGER ;
    BEGIN
        IF a.id < b.id
            RETURN -1;
        END IF;
        IF a.id > b.id
            RETURN 1;
        END IF;
        RETURN 0;
    END METHOD;

    ASK METHOD Key (IN a : DetectorStationObj) : STRING;
    BEGIN
        RETURN a.id;
    END METHOD;
END OBJECT ;

END MODULE.


(*
    DGlobal.mod
*)
DEFINITION MODULE Global;
FROM MathMod    IMPORT pi;
FROM Window     IMPORT WindowObj;
FROM Image      IMPORT ImageObj;
FROM ControlMedia   IMPORT ControlMediaSignal;

CONST
    SIMTIC = 0.5;   {simulation time tick}
    MPH2MTPS = 1600.0/3600.0;   {  1 mile = 1600 m,  1 hr= 3600 sec }
    METERPERFOOT = 0.3;
    METERPERKILOMETER = 1000.0;
    METERPERMILE = 1600.0;
    GRAMPERKILOGRAM = 1000.0;
    GRAMPERPOUND = 496.0;
    GRAMPERTON = 1000000.0;
    RADIANPERDEGREE = pi / 180.0;
    SECPERMINUTE = 60.0;
    SECPERHOUR = 3600.0;
VAR
```

```
    ConvertLength,  {conversion factor for length unit}
    ConvertWeight,  {conversion factor for weight unit}
    ConvertAngle,   {conversion factor for angle unit}
    ConvertSpeed,   {conversion factor for speed unit}
    ConvertTime : REAL; {conversion factor for time unit}
    window : WindowObj; {global window object}
    world  : ImageObj;  {global world object}
    viewxlo, viewylo, viewxhi, viewyhi : REAL;{global viewpoints}

PROCEDURE InitializeGlobal;
PROCEDURE DisposeGlobal;
END MODULE.


(*
    IGlobal.mod
*)

IMPLEMENTATION MODULE Global;
PROCEDURE InitializeGlobal;
BEGIN
    ConvertLength := 1.0;
    ConvertWeight := 1.0;
    ConvertAngle := 1.0;
    ConvertSpeed := 1.0;
    ConvertTime := 1.0;
    viewxlo := 0.0;
    viewylo := 0.0;
    viewxhi := 100.0;
    viewyhi := 100.0;

    NEW(window);
    ASK window TO SetTitle("Microscopic Traffic Modelling");
    ASK window TO SetSize(100.0, 100.0);
    NEW(world);
END PROCEDURE;

PROCEDURE DisposeGlobal;
BEGIN
{     DISPOSE(world);
    DISPOSE(window);
}
END PROCEDURE;

END MODULE.


(*
    DInterface.mod
 *)
DEFINITION MODULE Interface;

PROCEDURE InitializeConnection:INTEGER; NONMODSIM;
PROCEDURE CloseConnection:INTEGER;      NONMODSIM;
PROCEDURE SetStart:INTEGER;             NONMODSIM;
PROCEDURE SendDetectSignal (IN addr, status:INTEGER):INTEGER;   NONMODSIM;
PROCEDURE RecvControlSignal(IN addr, status:INTEGER):INTEGER;

END MODULE.


(*
    IInterface.mod
 *)

IMPLEMENTATION MODULE Interface;
FROM ControlMedia   IMPORT  LampObj, Lamps;

PROCEDURE RecvControlSignal(IN addr, status:INTEGER):INTEGER;
BEGIN
    IF (addr >= MAXCONTROL) OR (Lamps[ addr]  = NILOBJ)
```

```
        RETURN 0;
    END IF;
    IF (status > 0)
        TELL Lamps[ addr]  TO SetOn;
    ELSE
        TELL Lamps[ addr]  TO SetOff;
    END IF;
    RETURN 1;
END PROCEDURE;

END MODULE.



(*
    DMenus.mod
 *)
DEFINITION MODULE Menus;
FROM Menu IMPORT MenuBarObj;
TYPE
    menutype = OBJECT(MenuBarObj)
        OVERRIDE
        ASK METHOD BeSelected;
    END OBJECT;
    PROCEDURE InitMenuBar;
VAR
    menubar : menutype;
END MODULE.


(*
    IMenus.mod
 *)

IMPLEMENTATION MODULE Menus;
FROM SimMod     IMPORT  Timescale;
FROM Graphic    IMPORT  GraphicLibObj;
FROM Image      IMPORT  ImageObj;
FROM Form       IMPORT  DialogBoxObj;
FROM Menu       IMPORT  MenuBarObj, MenuObj, MenuItemObj;
FROM Button     IMPORT  ButtonObj;
FROM Value      IMPORT  ValueBoxObj;
FROM GrpMod     IMPORT  QueueObj;
FROM Global     IMPORT  window;

CONST
    menufile = "menus.sg2";
VAR
    menulib : GraphicLibObj;

PROCEDURE InitMenuBar;
BEGIN
    NEW(menulib);
    ASK menulib TO ReadFromFile(menufile);
    NEW(menubar);
    ASK menubar TO LoadFromLibrary(menulib, "MenuBar");
    ASK window  TO AddGraphic(menubar);
    ASK menubar TO Draw;
END PROCEDURE;

OBJECT menutype;
    ASK METHOD BeSelected;
    (* asynchronous menu handling routine *)
    VAR
        item      : MenuItemObj;
        menu      : MenuObj;
        temp      : ANYOBJ;

    BEGIN
        CASE ASK LastPicked ReferenceName
        WHEN "Exit" :
```

```
            OUTPUT("EXIT");
        WHEN "Start" :
            OUTPUT("Start");
        WHEN "Stop" :
            OUTPUT("Stop");
        WHEN "SetAddress" :
            OUTPUT("SetAddress");
        END CASE;
    END METHOD;
END OBJECT;
(*********************************************************************)
END MODULE.


(*
    DNetwork.mod
      logical vertics and links to find routes
      and to estimate the distance from a source to a destination
*)
DEFINITION MODULE Network;
FROM GrpMod     IMPORT  RankedObj, BTreeObj, QueueObj;
FROM Sect       IMPORT  Section;
FROM GTypes     IMPORT  PointType;

TYPE
    EdgeObj = OBJECT;
        v : VertexObj;
        distance : REAL;    {can be any index, Dijksta's algorithm applied based on this value}
        ASK METHOD SetVertex(IN a : VertexObj);
        ASK METHOD SetDistance(IN a : REAL);
        ASK METHOD GetWeight() : REAL;
    END OBJECT;

    VertexListObj = OBJECT(BTreeObj[ANYOBJ:EdgeObj]);
    OVERRIDE
        ASK METHOD Rank(IN a, b : EdgeObj) : INTEGER;
    END OBJECT;

    VertexObj = OBJECT;
        id : STRING;
        entrysection, exitsection : Section;
        location : PointType;       {global location point}
        adjacent : VertexListObj;
        {  BEGIN : Internal Usage }
            previous : VertexObj;
            weighttosource : REAL;
            {visited  : BOOLEAN;}
            {ASK METHOD SetVisited(IN v : BOOLEAN);}
            ASK METHOD SetWeightToSource(IN w : REAL);
            ASK METHOD SetPreviousVertex(IN v : VertexObj);
        {  END   : Internal Usage }
        ASK METHOD SetID(IN s : STRING);
        ASK METHOD SetSections(IN a, b : Section);
        ASK METHOD SetLocation(IN p : PointType);
        ASK METHOD GetDistance(IN v : VertexObj) : REAL;
        ASK METHOD ObjInit;
    END OBJECT;

    VertexArrayType = ARRAY INTEGER OF VertexObj;

    NetworkObj = OBJECT(BTreeObj[ANYOBJ:VertexObj]);
        ASK METHOD FindShortestPath(IN srcname, dstname : STRING;
                                    INOUT path : VertexArrayType);
        ASK METHOD Print();
    OVERRIDE
        ASK METHOD Rank(IN a, b : VertexObj) : INTEGER;
        ASK METHOD Key (IN a : VertexObj) : STRING;
    END OBJECT;

VAR
    NetworkDbase : NetworkObj;
```

```
END MODULE.

(*
    INetwork.mod

*)
IMPLEMENTATION MODULE Network;
FROM GrpMod     IMPORT  RankedObj, BTreeObj, QueueObj;
FROM Sect       IMPORT  Section;
FROM GTypes     IMPORT  PointType;
FROM MathMod    IMPORT  SQRT;
FROM IOMod      IMPORT  StreamObj, ALL FileUseType;
CONST
    MAXREAL = 1.7E+308;
    INFINITY = MAXREAL + MAXREAL;
TYPE
    PriorityQueueObj = OBJECT(RankedObj[ANYOBJ:VertexObj]);
    OVERRIDE
        ASK METHOD Rank(IN a, b : VertexObj) : INTEGER;
    END OBJECT;

OBJECT EdgeObj;
    ASK METHOD SetVertex(IN a : VertexObj);
    BEGIN
        v := a;
    END METHOD;

    ASK METHOD SetDistance(IN a : REAL);
    BEGIN
        distance := a;
    END METHOD;

    ASK METHOD GetWeight() : REAL;
    BEGIN
        RETURN distance;
    END METHOD;

END OBJECT;

OBJECT VertexListObj;
    ASK METHOD Rank(IN a, b : EdgeObj) : INTEGER;
    BEGIN
        IF a.GetWeight() < b.GetWeight()
            RETURN -1;
        ELSIF a.GetWeight() = b.GetWeight()
            RETURN 0;
        ELSIF a.GetWeight() > b.GetWeight()
            RETURN -1;
        ELSE
            OUTPUT("Error : Unknow RANK");
        END IF;
    END METHOD;
END OBJECT;

OBJECT VertexObj;
    ASK METHOD ObjInit;
    BEGIN
        NEW(adjacent);
    END METHOD;

    ASK METHOD SetID(IN s : STRING);
    BEGIN
        id := s;
    END METHOD;

    ASK METHOD SetSections(IN a, b : Section);
    BEGIN
        entrysection := b;
        exitsection := a;
    END METHOD;
```

```
    ASK METHOD SetLocation(IN p : PointType);
    BEGIN
        location := p;
    END METHOD;

    ASK METHOD GetDistance(IN v : VertexObj) : REAL;
    VAR
        dx, dy : REAL;
    BEGIN
        dx := location.x - v.location.x;
        dy := location.y - v.location.y;
        RETURN SQRT(dx * dx + dy * dy );
    END METHOD;

    ASK METHOD SetWeightToSource(IN w : REAL);
    BEGIN
        weighttosource := w;
    END METHOD;

    ASK METHOD SetPreviousVertex(IN v : VertexObj);
    BEGIN
        previous := v;
    END METHOD;
END OBJECT;

OBJECT PriorityQueueObj;
    ASK METHOD Rank(IN a, b : VertexObj) : INTEGER;
    BEGIN
        IF a.weighttosource < b.weighttosource
            RETURN -1;
        ELSIF a.weighttosource = b.weighttosource
            RETURN 0;
        ELSIF a.weighttosource > b.weighttosource
            RETURN -1;
        ELSE
            OUTPUT("Error : Unknow RANK");
        END IF;
    END METHOD;
END OBJECT;

OBJECT NetworkObj;
    (*
        OVERRIDE METHOD : Rank
    *)
    ASK METHOD Rank(IN a, b : VertexObj) : INTEGER;
    BEGIN
        IF a.id < b.id
            RETURN -1;
        ELSIF a.id = b.id
            RETURN 0;
        ELSIF a.id > b.id
            RETURN -1;
        ELSE
            OUTPUT("Error : Unknow RANK");
        END IF;
    END METHOD;

    (*
        OVERRIDE METHOD : Key
    *)
    ASK METHOD Key (IN a : VertexObj) : STRING;
    BEGIN
        RETURN a.id;
    END METHOD;

    ASK METHOD Print();
    VAR
        e : EdgeObj;
        v : VertexObj;
        fout : StreamObj;
    BEGIN
```

```
    NEW(fout);
    ASK fout TO Open("stdout", Output);
    FOREACH v IN SELF
        ASK fout TO WriteString(v.id);
        ASK fout TO WriteString(" => ");
        FOREACH e IN v.adjacent
            ASK fout TO WriteString(e.v.id);
            ASK fout TO WriteString(" ");
        END FOREACH;
        ASK fout TO WriteLn;
    END FOREACH;
END METHOD;


(*
    Find Shortest Path from src to dst.
*)
ASK METHOD FindShortestPath(IN srcname, dstname : STRING;
                            INOUT path : VertexArrayType);
VAR
    src, dst, v, u : VertexObj;
    queue : PriorityQueueObj;
    cweight : REAL;
    w : EdgeObj;
    cnt : INTEGER;
BEGIN
    path := NILARRAY;
    src := Find(srcname);
    IF src = NILOBJ
        OUTPUT("Can not find the source : ", srcname);
        RETURN;
    END IF;
    dst := Find(dstname);
    IF dst = NILOBJ
        OUTPUT("Can not find the destination : ", dstname);
        RETURN;
    END IF;

    FOREACH v IN SELF
        {ASK v TO SetVisited(FALSE);}
        ASK v TO SetWeightToSource(INFINITY);
        ASK v TO SetPreviousVertex(NILOBJ);
    END FOREACH;
    NEW(queue);

    { Dijkstra's Shortest Path Algorithm }
    ASK queue TO Add(src);
    v := ASK queue TO First();
    ASK v TO SetWeightToSource(0.0);

    WHILE v <> NILOBJ
        v := ASK queue TO Remove();
        {ASK v TO SetVisited(TRUE);}
        FOREACH w IN v.adjacent
            cweight := v.weighttosource + w.GetWeight();
            IF cweight < w.v.weighttosource
                ASK w.v TO SetWeightToSource(cweight);
                ASK w.v TO SetPreviousVertex(v);
            END IF;
            ASK queue TO Add(w.v);
        END FOREACH;
        v := ASK queue TO First();
    END WHILE;

    cnt := 1;
    v := dst;
    WHILE (v <> NILOBJ) AND (v <> src)
        INC(cnt);
        v := v.previous;
    END WHILE;
```

```
        IF v = NILOBJ
            RETURN;        { path was initialized to NILARRAY }
        END IF;

        NEW(path, 1..cnt);
        v := dst;
        WHILE (v <> NILOBJ) AND (v <> src)
            path[ cnt] := v;
            DEC(cnt);
            v := v.previous;
        END WHILE;
        path[ cnt] := v;
    END METHOD;
END OBJECT;
END MODULE.


(*
    DObserve.mod

*)

DEFINITION MODULE Observe;
TYPE
    ObserveType = (
        FirstObserve,

        {Type of Signal}
        RedSignal, RedLeftSignal, RedRightSignal,
        YellowSignal,
        GreenSignal, GreenLeftSignal, GreenRightSignal,

        {Type of On-Road Line}
        StopLine, LeftAllowedLine, ThruAllowedLine, RightAllowedLine,

        {Type of Sign}
        StopSign, YieldSign, LeftYieldSign,

        {Type of Vehicle Observed}
        FrontClear, FrontVeryClear,
        BackClear, BackVeryClear,
        LeftClear, LeftVeryClear,
        RightClear, RightVeryClear,

        LastObserve
    );

    {
        Set of Observe Type
    }
    ObserveSetObj = OBJECT
        pool : ARRAY ObserveType OF BOOLEAN;
        ASK METHOD Cardinal : INTEGER;
        ASK METHOD Add(IN a : ObserveType);
        ASK METHOD Subtract(IN a : ObserveType);
        ASK METHOD Includes(IN a : ObserveType) : BOOLEAN;
        ASK METHOD ObjPrint : STRING;
        ASK METHOD ObjInit;
    END OBJECT;

    {Find a match of a string to ObserveType
     Note:Use MODSIM functions(eg. OUTPUT()) to print a string of
     observe variable}
    PROCEDURE Str2Observe(IN s : STRING) : ObserveType;
END MODULE.
(*
    IObserve.mod

*)

IMPLEMENTATION MODULE Observe;
```

```
FROM Util       IMPORT CompactString;

CONST
    fstring = "*****************";

OBJECT ObserveSetObj;
    (************************************************)
    ASK METHOD Cardinal : INTEGER;
    VAR
        i : ObserveType;
        c : INTEGER;
    BEGIN
        c := 0;
        FOR i := LOW(pool) TO HIGH(pool)
            IF pool[ i]
                INC(c);
            END IF;
        END FOR;
        RETURN c;
    END METHOD;

    (************************************************)
    ASK METHOD Add(IN a : ObserveType);
    BEGIN
        pool[ a]  := TRUE;
    END METHOD;

    (************************************************)
    ASK METHOD Subtract(IN a : ObserveType);
    BEGIN
        pool[ a]  := FALSE;
    END METHOD;

    (************************************************)
    ASK METHOD Includes(IN a : ObserveType) : BOOLEAN;
    BEGIN
        RETURN pool[ a] ;
    END METHOD;

    (************************************************)
    ASK METHOD ObjPrint : STRING;
    VAR
        i : ObserveType;
        s : STRING;
        total, cnt : INTEGER;
    BEGIN
        total := Cardinal();
        cnt := 0;
        s := "{ ";
        FOR i := LOW(pool) TO HIGH(pool)
            IF pool[ i]
                INC(cnt);
                IF cnt = total
                    s := s + SPRINT(i) WITH fstring + "} ";
                ELSE
                    s := s + SPRINT(i) WITH fstring + ",";
                END IF;
            END IF;
        END FOR;
        CompactString(s);
        RETURN s;
    END METHOD;

    (************************************************)
    ASK METHOD ObjInit;
    VAR
        i : ObserveType;
    BEGIN
        NEW(pool, FirstObserve..LastObserve);
        FOR i := FirstObserve TO LastObserve
            pool[ i]  := FALSE;
```

```
        END FOR;
    END METHOD;
END OBJECT;



PROCEDURE Str2Observe(IN s : STRING) : ObserveType;
BEGIN
    IF s="FirstObserve"
        RETURN FirstObserve;
    ELSIF s="RedSignal"
        RETURN RedSignal;
    ELSIF s="RedLeftSignal"
        RETURN RedLeftSignal;
    ELSIF s="RedRightSignal"
        RETURN RedRightSignal;
    ELSIF s="YellowSignal"
        RETURN YellowSignal;
    ELSIF s="GreenSignal"
        RETURN GreenSignal;
    ELSIF s="GreenLeftSignal"
        RETURN GreenLeftSignal;
    ELSIF s="GreenRightSignal"
        RETURN GreenRightSignal;
    ELSIF s="StopLine"
        RETURN StopLine;
    ELSIF s="LeftAllowedLine"
        RETURN LeftAllowedLine;
    ELSIF s="ThruAllowedLine"
        RETURN ThruAllowedLine;
    ELSIF s="RightAllowedLine"
        RETURN RightAllowedLine;
    ELSIF s="StopSign"
        RETURN StopSign;
    ELSIF s="YieldSign"
        RETURN YieldSign;
    ELSIF s="LeftYieldSign"
        RETURN LeftYieldSign;
    ELSIF s="FrontClear"
        RETURN FrontClear;
    ELSIF s="FrontVeryClear"
        RETURN FrontVeryClear;
    ELSIF s="BackClear"
        RETURN BackClear;
    ELSIF s="BackVeryClear"
        RETURN BackVeryClear;
    ELSIF s="LeftClear"
        RETURN LeftClear;
    ELSIF s="LeftVeryClear"
        RETURN LeftVeryClear;
    ELSIF s="RightClear"
        RETURN RightClear;
    ELSIF s="RightVeryClear"
        RETURN RightVeryClear;
    ELSIF s="LastObserve"
        RETURN LastObserve;
    ELSE
        OUTPUT("Unknow Observe String");
        RETURN LastObserve;
    END IF;
END PROCEDURE;

END MODULE.


(*
    DSect.mod

*)

DEFINITION MODULE Sect;
```

```
FROM Image          IMPORT  ImageObj;
FROM GTypes         IMPORT  PointType, PointArrayType, ALL ColorType;
FROM ControlMedia   IMPORT  ControlMediaObj, ControlMediaDbaseObj;
FROM Detector       IMPORT  DetectorObj, DetectorDbaseObj,
                            DetectorStationObj, DetectorStationDbaseObj;
FROM GrpMod         IMPORT  BTreeObj;

TYPE
    Lane = OBJECT
        number : INTEGER;
        minspeed, maxspeed : REAL;
        width : REAL;
        ASK METHOD SetNumber(IN n : INTEGER);   {Set the index of the lane}
        ASK METHOD SetWidth(IN w : REAL);
        ASK METHOD SetSpeed(IN smin, smax : REAL);
    END OBJECT;

    NormalLane = OBJECT(Lane)   {Shoulder also use the same type as normal lane}
    END OBJECT;

    VariableLane = OBJECT(Lane)
        p1, p2, p3, p4 : REAL;  {pointers in ascending order}
        {p1 and p4 are attached in other lanes,
         p2 and p3 are away from the lanes}
        ASK METHOD SetPoints(IN x1, x2, x3, x4 : REAL);
    END OBJECT;

    NormalLaneArrayType = ARRAY INTEGER OF NormalLane;
    VariableLaneArrayType = ARRAY INTEGER OF VariableLane;

    Section = OBJECT(ImageObj)
        id : STRING;
        origin : PointType;
        orient : REAL;
        controls : ControlMediaDbaseObj;
        detectors : DetectorDbaseObj;
        stations  : DetectorStationDbaseObj;

        ASK METHOD SetID(IN name : STRING);
        ASK METHOD SetOrigin(IN cx, cy : REAL);
        ASK METHOD SetOrientation(IN rad : REAL);
        ASK METHOD GetWorldBounds(INOUT p : PointArrayType);
        ASK METHOD GetNominalSpeed : REAL;
        ASK METHOD ObjPrint() : STRING;
        ASK METHOD CreateImage;
        ASK METHOD GlobalToLocal(IN p : PointType; INOUT q : PointType);
        ASK METHOD LocalToGlobal(IN p : PointType; INOUT q : PointType);
        ASK METHOD GetXYByLW(IN lr, wr : REAL) : PointType;
        ASK METHOD AddControl(IN c : ControlMediaObj);
        ASK METHOD AddDetector(IN d : DetectorObj);
        ASK METHOD AddDetectorStation(IN d : DetectorStationObj);
    PRIVATE
        ASK METHOD DeleteImage;
    OVERRIDE
        ASK METHOD ObjInit;
    END OBJECT;

    SectionArrayType = ARRAY INTEGER OF Section;

    LinearSection = OBJECT (Section)
        length : REAL;
        normallane : NormalLaneArrayType;
        variablelane : VariableLaneArrayType;
        leftshoulder, rightshoulder : NormalLane;
        ASK METHOD SetLength(IN l : REAL);
        ASK METHOD SetNormalLanes(IN lanes : NormalLaneArrayType);
        ASK METHOD SetVariableLanes(IN lanes : VariableLaneArrayType);
        ASK METHOD SetShoulder(IN ls, rs : NormalLane);
        ASK METHOD Print;
        ASK METHOD GetWidth(IN flane, tlane : INTEGER) : REAL;
    OVERRIDE
```

```
        ASK METHOD GlobalToLocal(IN p : PointType; INOUT q : PointType);
        ASK METHOD LocalToGlobal(IN p : PointType; INOUT q : PointType);
        ASK METHOD ObjPrint() : STRING;
        ASK METHOD ObjInit;
        ASK METHOD GetWorldBounds(INOUT p : PointArrayType);
        ASK METHOD GetXYByLW(IN lr, wr : REAL) : PointType;
        ASK METHOD CreateImage;
    END OBJECT;

    LinearSectionArrayType = ARRAY INTEGER OF LinearSection;

    RegionalSection = OBJECT(Section)
        (* p[ 2] of the first outboundsection is the origin ??? *)
        outboundsections, inboundsections : SectionArrayType;
        ASK METHOD SetSections(IN is, os : SectionArrayType);
    OVERRIDE
        ASK METHOD ObjInit;
        ASK METHOD CreateImage;
    END OBJECT;

    VirtualSection = OBJECT(Section)
        mean : REAL;
        ASK METHOD SetMean(IN m : REAL);
    OVERRIDE
        ASK METHOD ObjInit;
        ASK METHOD CreateImage;
    END OBJECT;

    OriginSection = OBJECT(VirtualSection)
        ASK METHOD ConnectTo(IN s : Section);
    OVERRIDE
        ASK METHOD ObjInit;
        ASK METHOD CreateImage;
    END OBJECT;

    DestinSection = OBJECT(VirtualSection)
        ASK METHOD ConnectFrom(IN s : Section);
    OVERRIDE
        ASK METHOD ObjInit;
        ASK METHOD CreateImage;
    END OBJECT;

SectionDbaseObj = OBJECT (BTreeObj[ ANYOBJ:Section]);
OVERRIDE
    ASK METHOD Rank (IN a, b : Section) : INTEGER ;
    ASK METHOD Key (IN a : Section) : STRING;
END OBJECT ;

VAR
    BackgroundColor, RoadColor, ShoulderColor,
    NormalLineColor, CenterLineColor,
    OriginColor, DestinColor : ColorType;
    SectionDbase : SectionDbaseObj;
END MODULE.

(*
    ISect.mod

*)
IMPLEMENTATION MODULE Sect;
FROM MathMod    IMPORT SIN, COS, pi;
FROM Image      IMPORT ImageObj;
FROM Graphic    IMPORT GraphicLibObj;
FROM GTypes     IMPORT PointType, PointArrayType,
                       FillStyleType(SolidFill),
                       LineStyleType(SolidLine, DashedLine);
FROM Line       IMPORT PolylineObj;
FROM Fill       IMPORT PolygonObj;
FROM GrpMod     IMPORT RankedObj;
FROM Util       IMPORT SortedPoints;
FROM Vector     IMPORT VectorObj;
```

```
(*********************************************************
  Object : Lane
 *********************************************************)
OBJECT Lane;
    ASK METHOD SetNumber(IN n : INTEGER);
    BEGIN
        number := n;
    END METHOD;

    ASK METHOD SetWidth(IN w : REAL);
    BEGIN
        width := w;
    END METHOD;

    ASK METHOD SetSpeed(IN smin, smax : REAL);
    BEGIN
        minspeed := smin;
        maxspeed := smax;
    END METHOD;
END OBJECT;

(*********************************************************
  Object : VariableLane
 *********************************************************)
OBJECT VariableLane;
    ASK METHOD SetPoints(IN x1, x2, x3, x4 : REAL);
    BEGIN
        p1 := x1; p2 := x2;
        p3 := x3; p4 := x4;
    END METHOD;
END OBJECT;

(*********************************************************
  Object : Section
 *********************************************************)
OBJECT Section;
    ASK METHOD AddControl(IN c : ControlMediaObj);
    BEGIN
        ASK controls TO Add(c);
    END METHOD;

    ASK METHOD AddDetector(IN d : DetectorObj);
    BEGIN
        ASK detectors TO Add(d);
    END METHOD;

    ASK METHOD AddDetectorStation(IN d : DetectorStationObj);
    BEGIN
        ASK stations TO Add(d);
    END METHOD;

    ASK METHOD SetID(IN name : STRING);
    BEGIN
        id := name;
    END METHOD;

    ASK METHOD SetOrigin(IN cx, cy : REAL);
    BEGIN
        origin.x := cx;
        origin.y := cy;
        SetTranslation(cx, cy);
    END METHOD;

    ASK METHOD SetOrientation(IN rad : REAL);
    BEGIN
        orient := rad;
        SetRotation(orient);
    END METHOD;

    ASK METHOD ObjPrint() : STRING;
```

```
    BEGIN
        RETURN id +
                " (" + REALTOSTR(origin.x) + "m, " + REALTOSTR(origin.y) + "m) " +
                REALTOSTR(orient) + "rad";
    END METHOD;

    ASK METHOD ObjInit;
    BEGIN
        INHERITED ObjInit;
        NEW(controls);
        NEW(detectors);
        NEW(stations);
    END METHOD;

    ASK METHOD CreateImage;        { Virtual Function }
    BEGIN
    END METHOD;

    ASK METHOD DeleteImage;
    VAR
        x, y : ANYOBJ;
    BEGIN
        x := FirstGraphic();
        WHILE (x <> NILOBJ)
            y := x;
            x := NextGraphic(x);
            RemoveThisGraphic(y);
        END WHILE;
    END METHOD;

    ASK METHOD GetWorldBounds(INOUT p : PointArrayType);
    BEGIN
        { Virtual Function }
    END METHOD;

    ASK METHOD GetXYByLW(IN lr, wr : REAL) : PointType;
    VAR
        p : PointType;
    BEGIN
        p.x := 0.0;
        p.y := 0.0;
        RETURN p;
        { Virtual Function }
    END METHOD;


    ASK METHOD GlobalToLocal(IN p : PointType; INOUT q : PointType);
    VAR
        t : PointType;
    BEGIN
        t.x := p.x - origin.x;
        t.y := p.y - origin.y;
        q.x := t.x * COS(-orient) - t.y * SIN(-orient);
        q.y := t.x * SIN(-orient) + t.y * COS(-orient);
    END METHOD;

    ASK METHOD LocalToGlobal(IN p : PointType; INOUT q : PointType);
    BEGIN
        q.x := p.x * COS(orient) - p.y * SIN(orient) + origin.x;
        q.y := p.x * SIN(orient) + p.y * COS(orient) + origin.y;
    END METHOD;

    ASK METHOD GetNominalSpeed : REAL;
    BEGIN
        RETURN 5.0;
    END METHOD;
END OBJECT;

(*******************************************************
  Object : LinearSection
 *******************************************************)
```

```
OBJECT LinearSection;
    ASK METHOD GlobalToLocal(IN p : PointType; INOUT q : PointType);
    VAR
        t : PointType;
    BEGIN
        INHERITED GlobalToLocal(p, t);
        q.x := -t.y;     q.y := t.x;
    END METHOD;

    ASK METHOD LocalToGlobal(IN p : PointType; INOUT q : PointType);
    VAR
        t : PointType;
    BEGIN
        t.x := p.y;     t.y := -p.x;
        INHERITED LocalToGlobal(t, q);
    END METHOD;

    ASK METHOD CreateImage;
    VAR
        w, ws : REAL;
        p, tp, dp : PointArrayType;
        a : PolygonObj;
        h : PolylineObj;
        i, k, lcnt, rcnt, tcnt : INTEGER;
        lpoints, rpoints : SortedPoints;

    BEGIN
        DeleteImage;
        IF (length = 0.0) OR (normallane = NILARRAY)
            RETURN;
        END IF;

        lcnt := 2; rcnt := 2;
        IF variablelane <> NILARRAY
            FOR i := LOW(variablelane) TO HIGH(variablelane)
                IF variablelane[ i] .number = 0
                    INC(lcnt, 4);
                ELSIF variablelane[ i] .number = HIGH(normallane)+1
                    INC(rcnt, 4);
                END IF;
            END FOR;
        END IF;
        NEW(lpoints);
        ASK lpoints TO SetSize(lcnt);
        ASK lpoints TO SetSide(0);  (* Left *)
        NEW(rpoints);
        ASK rpoints TO SetSize(rcnt);
        ASK lpoints TO SetSide(1);  (* Right *)
        {OUTPUT("lcnt=", lcnt);}
        {OUTPUT("rcnt=", rcnt);}

        w := 0.0;
        FOR i :=  LOW(normallane) TO HIGH(normallane)
            w := w + ASK normallane[ i]  TO width;
        END FOR;
        NEW(p, 0..3);

        p[ 0] .x := 0.0;      p[ 0] .y := 0.0;
        p[ 1] .x := length;   p[ 1] .y := 0.0;
        p[ 2] .x := length;   p[ 2] .y := -w;
        p[ 3] .x := 0.0;      p[ 3] .y := -w;
        ASK lpoints TO Add(p[ 0] );
        ASK lpoints TO Add(p[ 1] );
        ASK rpoints TO Add(p[ 2] );
        ASK rpoints TO Add(p[ 3] );

        IF variablelane <> NILARRAY
          FOR i := LOW(variablelane) TO HIGH(variablelane)
            IF variablelane[ i] .number = 0
                p[ 0] .y := 0.0;  p[ 1] .y := 0.0;
                p[ 2] .y := ASK variablelane[ i]  TO width;
```

```
        p[ 3] .y := ASK variablelane[ i]  TO width;
        p[ 0] .x := length * ASK variablelane[ i]  TO p1;
        p[ 3] .x := length * ASK variablelane[ i]  TO p2;
        p[ 2] .x := length * ASK variablelane[ i]  TO p3;
        p[ 1] .x := length * ASK variablelane[ i]  TO p4;
        FOR i:=LOW(p) TO HIGH(p)
            ASK lpoints TO Add(p[ i] );
        END FOR;
    ELSIF variablelane[ i] .number = HIGH(normallane)+1
        p[ 0] .y := -w;   p[ 1] .y := -w;
        p[ 2] .y := -w - ASK variablelane[ i]  TO width;
        p[ 3] .y := -w - ASK variablelane[ i]  TO width;
        p[ 0] .x := length * ASK variablelane[ i]  TO p1;
        p[ 3] .x := length * ASK variablelane[ i]  TO p2;
        p[ 2] .x := length * ASK variablelane[ i]  TO p3;
        p[ 1] .x := length * ASK variablelane[ i]  TO p4;
        FOR i:=LOW(p) TO HIGH(p)
            ASK rpoints TO Add(p[ i] );
        END FOR;
    ELSE
        OUTPUT("I am sorry.  You ask Unsupported specs");
        OUTPUT("We allow only the first left variable lane and right variable lane");
    END IF;
  END FOR;
END IF;
DISPOSE(p);
ASK lpoints TO Sort;
ASK rpoints TO Sort;
tcnt := HIGH(lpoints.data)-LOW(lpoints.data)+1
        +HIGH(lpoints.data)-LOW(lpoints.data)+1;
NEW(tp, 1..tcnt);
k := 1;
dp := ASK lpoints TO data;
FOR i := LOW(dp) TO HIGH(dp)
    tp[ k] .x := dp[ i] .x;
    tp[ k] .y := dp[ i] .y;
    INC(k);
END FOR;
dp := ASK rpoints TO data;
FOR i := HIGH(dp) DOWNTO LOW(dp)
    tp[ k] .x := dp[ i] .x;
    tp[ k] .y := dp[ i] .y;
    INC(k);
END FOR;

NEW(a);          (* base shape *)
ASK a TO SetPoints(tp);
ASK a TO SetColor(RoadColor);
ASK a TO SetStyle(SolidFill);
AddGraphic(a);
DISPOSE(tp);

IF leftshoulder <> NILOBJ
    NEW(a);
    ws := ASK leftshoulder TO width;
    dp := ASK lpoints TO data;
    NEW(tp, 1..(HIGH(dp)-LOW(dp)+1)*2);
    k := 1;
    FOR i := LOW(dp) TO HIGH(dp)
        tp[ k] .x := dp[ i] .x;
        tp[ k] .y := dp[ i] .y;
        INC(k);
    END FOR;
    FOR i := HIGH(dp) DOWNTO LOW(dp)
        tp[ k] .x := dp[ i] .x;
        tp[ k] .y := dp[ i] .y+ws;
        INC(k);
    END FOR;
    ASK a TO SetPoints(tp);
    ASK a TO SetColor(ShoulderColor);
    ASK a TO SetStyle(SolidFill);
```

```
        AddGraphic(a);
        DISPOSE(tp);
    END IF;

    IF rightshoulder <> NILOBJ
        NEW(a);
        ws := ASK rightshoulder TO width;
        dp := ASK rpoints TO data;
        NEW(tp, 1..(HIGH(dp)-LOW(dp)+1)*2);
        k := 1;
        FOR i := LOW(dp) TO HIGH(dp)
            tp[ k] .x := dp[ i] .x;
            tp[ k] .y := dp[ i] .y;
            INC(k);
        END FOR;
        FOR i := HIGH(dp) DOWNTO LOW(dp)
            tp[ k] .x := dp[ i] .x;
            tp[ k] .y := dp[ i] .y-ws;
            INC(k);
        END FOR;
        ASK a TO SetPoints(tp);
        ASK a TO SetColor(ShoulderColor);
        ASK a TO SetStyle(SolidFill);
        AddGraphic(a);
        DISPOSE(tp);
    END IF;

    NEW(p, 0..1);
    p[ 0] .x := 0.0;      p[ 0] .y := 0.0;
    p[ 1] .x := length;   p[ 1] .y := 0.0;
    NEW(h);
    ASK h TO SetPoints(p);
    ASK h TO SetColor(NormalLineColor);
    ASK h TO SetStyle(DashedLine);
    AddGraphic(h);
    w := 0.0;
    FOR i :=  LOW(normallane) TO HIGH(normallane)   (* Draw lines *)
        w := w + ASK normallane[ i]  TO width;
        p[ 0] .y := -w;       p[ 1] .y := -w;
        NEW(h);
        ASK h TO SetPoints(p);
        ASK h TO SetColor(NormalLineColor);
        ASK h TO SetStyle(DashedLine);
        AddGraphic(h);
    END FOR;
    DISPOSE(p);

    dp := ASK lpoints TO data;
    NEW(h);                          (* Draw center line *)
    ASK h TO SetPoints(dp);
    ASK h TO SetColor(CenterLineColor);
    ASK h TO SetStyle(SolidLine);
    AddGraphic(h);

    dp := ASK rpoints TO data;
    NEW(h);                          (* Draw shoulder line *)
    ASK h TO SetPoints(dp);
    ASK h TO SetColor(NormalLineColor);
    ASK h TO SetStyle(SolidLine);
    AddGraphic(h);

    INHERITED CreateImage;
END METHOD;     (* CreateImage *)

ASK METHOD SetLength(IN l : REAL);
BEGIN
    length := l;
END METHOD;

ASK METHOD SetNormalLanes(IN lanes : NormalLaneArrayType);
BEGIN
```

```
    normallane := lanes;
END METHOD;

ASK METHOD SetVariableLanes(IN lanes : VariableLaneArrayType);
BEGIN
    variablelane := lanes;
END METHOD;

ASK METHOD SetShoulder(IN ls, rs : NormalLane);
BEGIN
    leftshoulder := ls;
    rightshoulder := rs;
END METHOD;

ASK METHOD GetWorldBounds(INOUT p : PointArrayType);
VAR
    i : INTEGER;
    w : REAL;
    q : PointArrayType;
BEGIN
    w := GetWidth(LOW(normallane), HIGH(normallane));
    NEW(q, 1..4);
    q[ 1].x := 0.0;      q[ 1].y := 0.0;
    q[ 2].x := 0.0;      q[ 2].y := -w;
    q[ 3].x := length;   q[ 3].y := 0.0;
    q[ 4].x := length;   q[ 4].y := -w;

    IF variablelane <> NILARRAY
      FOR i := LOW(variablelane) TO HIGH(variablelane)
        IF (variablelane[ i].p1 = 0.0) AND (variablelane[ i].p2 = 0.0)
            IF variablelane[ i].number = 0
                q[ 1].y := q[ 1].y + variablelane[ i].width;
            ELSE
                q[ 2].y := q[ 2].y - variablelane[ i].width;
            END IF;
        END IF;
        IF (variablelane[ i].p3 = 1.0) AND (variablelane[ i].p4 = 1.0)
            IF variablelane[ i].number = 0
                q[ 3].y := q[ 3].y + variablelane[ i].width;
            ELSE
                q[ 4].y := q[ 4].y - variablelane[ i].width;
            END IF;
        END IF;
      END FOR;
    END IF;

    FOR i := 1 TO 4
        (* Rotation by orient and  translate by (origin.x, origin.y) *)
        p[ i].x := q[ i].x * COS(orient) - q[ i].y * SIN(orient) + origin.x;
        p[ i].y := q[ i].x * SIN(orient) + q[ i].y * COS(orient) + origin.y;
    END FOR;
    DISPOSE(q);
END METHOD;

ASK METHOD GetWidth(IN flane, tlane : INTEGER) : REAL;
VAR
    i : INTEGER;
    sum : REAL;
BEGIN
    sum := 0.0;
    FOR i := flane TO tlane
        sum := sum + normallane[ i].width;
    END FOR;
    RETURN sum;
END METHOD;

ASK METHOD GetXYByLW(IN lr, wr : REAL) : PointType;
VAR
    p : PointType;
    lane : INTEGER;
BEGIN
```

```
        lane := TRUNC(wr);
        p.x := GetWidth(1, lane-1) + (wr-FLOAT(lane)) * GetWidth(lane,lane);
        p.y := lr * length;
        RETURN p;
    END METHOD;


    ASK METHOD Print;
    BEGIN
        OUTPUT("ID : ", id);
        OUTPUT("ORIGIN : (", origin.x, ",", origin.y, ")");
        OUTPUT("ORIENT : ", orient);
    END METHOD;

    ASK METHOD ObjPrint : STRING;
    BEGIN
        RETURN INHERITED ObjPrint + ", length:" + REALTOSTR(length);
    END METHOD;

    ASK METHOD ObjInit;
    BEGIN
        INHERITED ObjInit;
    END METHOD;
END OBJECT;

(*************************************************************************
  Regioanl Section
 *************************************************************************)
OBJECT RegionalSection;
    ASK METHOD SetSections(IN is, os : SectionArrayType);
    VAR
        idx1, idx2 : INTEGER;
        p : PointArrayType;
        v : VectorObj;
    BEGIN
        inboundsections := is;
        outboundsections := os;
        idx1 := LOW(outboundsections);
        idx2 := LOW(inboundsections);
        NEW(p, 1..4);
        NEW(v);

        IF outboundsections[ idx1] <> NILOBJ
            ASK outboundsections[ idx1]  TO GetWorldBounds(p);
            ASK v TO Init(p[ 1] .x-p[ 2] .x, p[ 1] .y-p[ 2] .y);
            ASK SELF TO SetOrigin(p[ 2] .x, p[ 2] .y);
            ASK SELF TO SetOrientation(v.Angle);
        ELSIF inboundsections[ idx2] <> NILOBJ
            ASK inboundsections[ idx2]   TO GetWorldBounds(p);
            ASK v TO Init(p[ 4] .x-p[ 3] .x, p[ 4] .y-p[ 3] .y);
            ASK SELF TO SetOrigin(p[ 3] .x, p[ 3] .y);
            ASK SELF TO SetOrientation(v.Angle);
        ELSE
            OUTPUT("Error : Cannot make an intersection because both of the first sections are
NIL");
        END IF;
        DISPOSE(v);
        DISPOSE(p);
    END METHOD;

    ASK METHOD CreateImage;
    VAR
        i, k, cnt : INTEGER;
        a : ANYOBJ;
        p, q : PointArrayType;
        shape : PolygonObj;
    BEGIN
        cnt := 0;
        FOR i:= LOW(inboundsections) TO HIGH(inboundsections)
            IF inboundsections[ i] <> NILOBJ
                INC(cnt);
```

```
            END IF;
        END FOR;
        FOR i:= LOW(outboundsections) TO HIGH(outboundsections)
            IF outboundsections[ i] <> NILOBJ
                INC(cnt);
            END IF;
        END FOR;
        cnt := 2*cnt;
        NEW(p, 1..cnt);     (* number of valid sections *)
        NEW(q, 1..4);
        k := 1;
        FOR i:= LOW(inboundsections) TO HIGH(inboundsections)
            IF outboundsections[ i] <> NILOBJ
                ASK outboundsections[ i] TO GetWorldBounds(q);
                GlobalToLocal(q[ 2], p[ k]);  INC(k);
                GlobalToLocal(q[ 1], p[ k]);  INC(k);
            END IF;
            IF inboundsections[ i] <> NILOBJ
                ASK inboundsections[ i]  TO GetWorldBounds(q);
                GlobalToLocal(q[ 3], p[ k]);  INC(k);
                GlobalToLocal(q[ 4], p[ k]);  INC(k);
            END IF;
        END FOR;

        DeleteImage;
        NEW(shape);
        ASK shape TO SetPoints(p);
        ASK shape TO SetColor(RoadColor);
        ASK shape TO SetStyle(SolidFill);
        AddGraphic(shape);
    END METHOD;

    ASK METHOD ObjInit;
    BEGIN
        INHERITED ObjInit;
    END METHOD;
END OBJECT;

(*************************************************************************
  Virtual Section
 *************************************************************************)
OBJECT VirtualSection;
    ASK METHOD SetMean(IN m : REAL);
    BEGIN
        mean := m;
    END METHOD;

    ASK METHOD CreateImage;
    BEGIN
        { Virtual Function }
    END METHOD;

    ASK METHOD ObjInit;
    BEGIN
        INHERITED ObjInit;
    END METHOD;

END OBJECT;

OBJECT OriginSection;
    ASK METHOD ConnectTo(IN s : Section);
    VAR
        p : PointArrayType;
    BEGIN
        NEW(p, 1..4);
        ASK s TO GetWorldBounds(p);
        SetOrigin(0.5*(p[ 1].x+p[ 2].x), 0.5*(p[ 1].y+p[ 2].y));
        SetOrientation(ASK s TO orient);
        DISPOSE(p);
    END METHOD;
```

```
    ASK METHOD CreateImage;
    VAR
        shape   : ImageObj;
        lib     : GraphicLibObj;
        { text    : TextObj;}
    BEGIN
        NEW(lib);                   (* Load library created using SIMDRAW      *)
        ASK lib TO ReadFromFile("mobile.sg2");
        NEW(shape);
        ASK shape TO LoadFromLibrary(lib, "origin");
        AddGraphic(shape);
        DISPOSE(lib);
    END METHOD;

    ASK METHOD ObjInit;
    BEGIN
        INHERITED ObjInit;
    END METHOD;
END OBJECT;

OBJECT DestinSection;
    ASK METHOD ConnectFrom(IN s : Section);
    VAR
        p : PointArrayType;
    BEGIN
        NEW(p, 1..4);
        ASK s TO GetWorldBounds(p);
        SetOrigin(0.5*(p[ 3] .x+p[ 4] .x), 0.5*(p[ 3] .y+p[ 4] .y));
        SetOrientation(ASK s TO orient);
        DISPOSE(p);
    END METHOD;

    ASK METHOD CreateImage;
    VAR
        shape   : ImageObj;
        lib     : GraphicLibObj;
        { text    : TextObj;}
    BEGIN
        NEW(lib);                   (* Load library created using SIMDRAW      *)
        ASK lib TO ReadFromFile("mobile.sg2");
        NEW(shape);
        ASK shape TO LoadFromLibrary(lib, "destin");
        AddGraphic(shape);
        DISPOSE(lib);
    END METHOD;

    ASK METHOD ObjInit;
    BEGIN
        INHERITED ObjInit;
    END METHOD;
END OBJECT;

OBJECT SectionDbaseObj;
    ASK METHOD Rank(IN a, b : Section) : INTEGER;
    BEGIN
        IF a.id < b.id
            RETURN -1;
        END IF;
        IF a.id > b.id
            RETURN 1;
        END IF;
        RETURN 0;
    END METHOD;

    ASK METHOD Key(IN a : Section) : STRING;
    BEGIN
        RETURN a.id;
    END METHOD;
END OBJECT;
END MODULE.
```

```
(*
    DSimIO.mod
 *)
DEFINITION MODULE SimIO;
PROCEDURE ReadSimFile(IN filename : STRING) : BOOLEAN;
END MODULE.


(*
    ISimIO.mod

*)

IMPLEMENTATION MODULE SimIO;
FROM IOMod      IMPORT  StreamObj, FileUseType(Input);
FROM Network    IMPORT  NetworkDbase, VertexObj, EdgeObj;
FROM MathMod    IMPORT  SQRT;
FROM Window     IMPORT  WindowObj;
FROM Animate    IMPORT  DynImageObj, DynDClockObj;
FROM Graphic    IMPORT  GraphicLibObj;
FROM Image      IMPORT  ImageObj;
FROM SimMod     IMPORT  ResetSimTime;
FROM GTypes     IMPORT  PointType, PointArrayType,
                        ALL ColorType, ALL FillStyleType, ALL LineStyleType,
                        WorldXhi, WorldYhi;
FROM Line       IMPORT  PolylineObj;
FROM Fill       IMPORT  PolygonObj;
FROM Sect       IMPORT  NormalLane, NormalLaneArrayType, VariableLaneArrayType,
                        Section, LinearSection, RegionalSection,
                        SectionArrayType, OriginSection, DestinSection,
                        BackgroundColor, RoadColor, ShoulderColor,
                        NormalLineColor, CenterLineColor,
                        OriginColor, DestinColor,
                        SectionDbase;
FROM Util       IMPORT  IsEmptyString, RemoveComment, StrTok, IsSpace,
                        ArrowPolylineObj;
FROM ControlMedia IMPORT  ControlMediaObj, ControlMediaSign,
                          ControlMediaStopLine, ControlMediaSignal;
FROM Observe    IMPORT  ALL ObserveType;
FROM Detector   IMPORT  DetectorObj, DetectorStationObj;
FROM Global     IMPORT  SIMTIC,
                        METERPERFOOT, METERPERKILOMETER, METERPERMILE,
                        GRAMPERKILOGRAM, GRAMPERPOUND, GRAMPERTON,
                        RADIANPERDEGREE, SECPERMINUTE, SECPERHOUR,
                        ConvertLength,ConvertWeight,ConvertAngle,
                        ConvertSpeed,ConvertTime,
                        window,
                        world,
                        viewxlo, viewylo, viewxhi, viewyhi;

VAR
    gridspace : REAL;
    gridcolor, LogicalColor : ColorType;
    ShowVirtual : BOOLEAN;

PROCEDURE GridDraw;
VAR
    a : PolylineObj;
    p : PointArrayType;
    ruler : ImageObj;
    step : REAL;
BEGIN
    NEW(ruler);
    NEW(p, 1..2);

    step := gridspace * FLOAT(TRUNC(viewxlo/gridspace));

    WHILE step <= viewxhi
        NEW(a);
        p[1].x := step;     p[1].y := viewylo;
```

```
        p[ 2].x := step;     p[ 2].y := viewyhi;
        ASK a TO SetPoints(p);
        ASK a TO SetColor(gridcolor);
        ASK ruler TO AddGraphic(a);
        step := step + gridspace;
    END WHILE;

    step := gridspace * FLOAT(TRUNC(viewylo/gridspace));

    WHILE step <= viewyhi
        NEW(a);
        p[ 1].x := viewxlo; p[ 1].y := step;
        p[ 2].x := viewxhi; p[ 2].y := step;
        ASK a TO SetPoints(p);
        ASK a TO SetColor(gridcolor);
        ASK ruler TO AddGraphic(a);
        step := step + gridspace;
    END WHILE;
    ASK world TO AddGraphic(ruler);
END PROCEDURE;

PROCEDURE ProcessStraightSegment(IN line : STRING);
VAR
    id : STRING;
    x0, y0, orient, length : REAL;
    nlane, nvlane : INTEGER;
    widthnormal : ARRAY INTEGER OF REAL;
    widthlshoulder, widthrshoulder : REAL;
    vid : INTEGER;
    widthvlane, p1, p2, p3, p4 : REAL;
    i : INTEGER;
    section : LinearSection;
    lanes  : NormalLaneArrayType;
    vlanes : VariableLaneArrayType;
    ls, rs : NormalLane;
BEGIN
    id := StrTok(line, ":");
    x0 := STRTOREAL(StrTok(line,",")) * ConvertLength;
    y0 := STRTOREAL(StrTok(line,":")) * ConvertLength;
    orient := STRTOREAL(StrTok(line,":")) * ConvertAngle;
    length := STRTOREAL(StrTok(line,":")) * ConvertLength;
    nlane := STRTOINT(StrTok(line, ":"));
    IF nlane > 0
        NEW(widthnormal, 1..nlane);
        FOR i:=1 TO nlane-1
            widthnormal[ i]  := STRTOREAL(StrTok(line,",")) * ConvertLength;
        END FOR;
        widthnormal[ nlane]  := STRTOREAL(StrTok(line,":")) * ConvertLength;
    END IF;
    widthlshoulder := STRTOREAL(StrTok(line,",")) * ConvertLength;
    widthrshoulder := STRTOREAL(StrTok(line,":")) * ConvertLength;

    NEW(lanes, 1..nlane);
    FOR i:=1 TO nlane
        NEW(lanes[ i] );
        ASK lanes[ i]  TO SetWidth(widthnormal[ i] );
        ASK lanes[ i]  TO SetNumber(i);
    END FOR;

    NEW(ls); NEW(rs);
    ASK ls TO SetWidth(widthlshoulder);
    ASK rs TO SetWidth(widthrshoulder);

    NEW(section);
    ASK section     TO SetID(id);
    ASK section     TO SetOrigin(x0, y0);
    ASK section     TO SetOrientation(orient);
    ASK section     TO SetLength(length);
    ASK section     TO SetNormalLanes(lanes);
    ASK section     TO SetShoulder(ls, rs);
```

```
    { Create a section at this time }
    nvlane := STRTOINT(StrTok(line, ":"));
    IF nvlane > 0
        NEW(vlanes, 1..nvlane);
    END IF;
    FOR i:=1 TO nvlane
        vid := STRTOINT(StrTok(line, ","));
        widthvlane := STRTOREAL(StrTok(line,",")) * ConvertLength;
        p1 := STRTOREAL(StrTok(line,",")) * ConvertLength / length;
        p2 := STRTOREAL(StrTok(line,",")) * ConvertLength / length;
        p3 := STRTOREAL(StrTok(line,",")) * ConvertLength / length;
        p4 := STRTOREAL(StrTok(line,":")) * ConvertLength / length;
        NEW(vlanes[ i] );
        ASK vlanes[ i]  TO SetWidth(widthvlane);
        ASK vlanes[ i]  TO SetNumber(vid);
        ASK vlanes[ i]  TO SetPoints(p1, p2, p3, p4);
    END FOR;
    ASK section TO SetVariableLanes(vlanes);
    ASK SectionDbase TO Add(section);
END PROCEDURE;

PROCEDURE ProcessIntersection(IN line : STRING);
VAR
    id, segid : STRING;
    i, nseg : INTEGER;
    isect, osect : SectionArrayType;
    r : RegionalSection;
BEGIN
    id := StrTok(line, ":");
    nseg := STRTOINT(StrTok(line, ":"));
    NEW(isect, 1..nseg);
    FOR i:=1 TO nseg
        IF i = nseg
            segid := StrTok(line, ":");
        ELSE
            segid := StrTok(line, ",");
        END IF;
        isect[ i]  := ASK SectionDbase TO Find(segid);
    END FOR;
    NEW(osect, 1..nseg);
    FOR i:=1 TO nseg
        IF i = nseg
            segid := StrTok(line, ":");
        ELSE
            segid := StrTok(line, ",");
        END IF;
        osect[ i]  := ASK SectionDbase TO Find(segid);
    END FOR;

    NEW(r);
    ASK r   TO SetID(id);
    ASK r   TO SetSections(isect, osect);
    ASK SectionDbase TO Add(r);
END PROCEDURE;

PROCEDURE ProcessOriginZone(IN line : STRING);
VAR
    zone : OriginSection;
BEGIN
    NEW(zone);
    ASK zone TO SetID(StrTok(line, ":"));
    ASK zone TO ConnectTo(ASK SectionDbase TO Find(StrTok(line, ":")));
    ASK zone TO SetMean(STRTOREAL(StrTok(line,":")));
    ASK SectionDbase TO Add(zone);
END PROCEDURE;

PROCEDURE ProcessDestinZone(IN line : STRING);
VAR
    zone : DestinSection;
BEGIN
    NEW(zone);
```

```
    ASK zone TO SetID(StrTok(line, ":"));
    ASK zone TO ConnectFrom(ASK SectionDbase TO Find(StrTok(line, ":")));
    ASK zone TO SetMean(STRTOREAL(StrTok(line,":")));
    ASK SectionDbase TO Add(zone);
END PROCEDURE;


PROCEDURE ProcessSection(IN command, line : STRING);
BEGIN
    IF STRLEN(command) <> 2
        RETURN;
    END IF;
    CASE SCHAR(command,2)
    WHEN 'S' :          { Straight Segment}
        ProcessStraightSegment(line);
    WHEN 'I' :          { Intersection }
        ProcessIntersection(line);
    WHEN 'O' :          { Origin Zone }
        ProcessOriginZone(line);
    WHEN 'D' :          { Destin Zone }
        ProcessDestinZone(line);
    END CASE;
END PROCEDURE;

PROCEDURE ProcessControl(IN command, line : STRING);
VAR
    i, cnt : INTEGER;
    sign : ControlMediaSign;
    cl : ControlMediaStopLine;
    signal : ControlMediaSignal;
    controlname : STRING;
    sect : Section;
    p, q : PointType;
    message : ObserveType;
BEGIN
    IF STRLEN(command) <> 2
        RETURN;
    END IF;
    CASE SCHAR(command, 2)
    WHEN 'N' :  { Sign }
        NEW(sign);
        ASK sign TO SetID(StrTok(line,":"));
        ASK sign TO InitMessage(StrTok(line, ":"));
        sect := ASK SectionDbase TO Find(StrTok(line, ":"));
        p.x := STRTOREAL(StrTok(line, ","));
        p.y := STRTOREAL(StrTok(line, ":"));
        ASK sign TO Init(p, p);
        ASK sect TO AddControl(sign);
    WHEN 'S' :  { Signal }
        NEW(signal);
        ASK signal TO SetID(StrTok(line,":"));
        cnt := STRTOINT(StrTok(line, ":"));
        ASK signal TO SetCount(cnt);
        FOR i:=1 TO cnt
            ASK signal TO InitMessage(StrTok(line,":"));
        END FOR;
        sect := ASK SectionDbase TO Find(StrTok(line, ":"));
        p.x := STRTOREAL(StrTok(line, ","));
        p.y := STRTOREAL(StrTok(line, ":"));
        ASK signal TO Init(p, p);
        ASK sect TO AddControl(signal);
    WHEN 'L' :  { Control Stop Line }
        NEW(cl);
        ASK cl TO SetID(StrTok(line,":"));
        ASK cl TO InitMessage(StrTok(line, ":"));
        sect := ASK SectionDbase TO Find(StrTok(line, ":"));
        p.x := STRTOREAL(StrTok(line, ","));
        p.y := STRTOREAL(StrTok(line, ":"));
        q.x := STRTOREAL(StrTok(line, ","));
        q.y := STRTOREAL(StrTok(line, ":"));
        ASK cl TO Init(q, p);
```

```
        ASK sect TO AddControl(cl);
    WHEN 'M' :  { Message }
        RETURN;
    END CASE;

END PROCEDURE;

PROCEDURE ProcessDetector(IN command, line : STRING);
VAR
    d : DetectorObj;
    u : DetectorStationObj;
    s : Section;
    p : PointType;
    h, w : REAL;
    i, n : INTEGER;
BEGIN
    IF STRLEN(command) <> 2
        RETURN;
    END IF;
    CASE SCHAR(command, 2)
    WHEN 'L' :  { Loop Detector }
        NEW(d);
        ASK d TO SetID(StrTok(line, ":"));
        s := ASK SectionDbase TO Find(StrTok(line,":"));
        p.x := STRTOREAL(StrTok(line, ","));
        p.y := STRTOREAL(StrTok(line, ":"));
        w := STRTOREAL(StrTok(line, ","));
        h := STRTOREAL(StrTok(line, ":"));
        ASK d TO SetLocation(p);
        ASK d TO SetSize(w, h);
        ASK s TO AddDetector(d);
    WHEN 'S' : {  Detector Station }
        NEW(u);
        ASK u TO SetID(StrTok(line, ":"));
        n := STRTOINT(StrTok(line, ":"));
        ASK u TO SetSize(n);
        FOR i:=1 TO n
            s := ASK SectionDbase TO Find(StrTok(line, ","));
            d := ASK s.detectors TO Find(StrTok(line, ":"));
            ASK u TO AddDetector(d);
        END FOR;
        ASK s TO AddDetectorStation(u);
    OTHERWISE
        OUTPUT("Warning!  Unknown Detector Type", command, ":", line);
    END CASE;
END PROCEDURE;

PROCEDURE ProcessGlobal(IN command, line : STRING);
VAR
    unit : STRING;
    wx, wy : REAL;
    ch  : CHAR;
BEGIN
    IF STRLEN(command) <> 2
        RETURN;
    END IF;
    CASE SCHAR(command,2)
    WHEN 'L' :
        unit := StrTok(line, ":");
        IF unit = "m"
            ConvertLength := 1.0;
        ELSIF unit = "ft"
            ConvertLength := METERPERFOOT;
        ELSIF unit = "km"
            ConvertLength := METERPERKILOMETER;
        ELSIF unit = "mi"
            ConvertLength := METERPERMILE;
        ELSE
            OUTPUT("WARNING! INVALID LENGTH UNIT");
        END IF;
    WHEN 'W' :
```

```
        unit := StrTok(line, ":");
        IF unit = "g"
            ConvertWeight := 1.0;
        ELSIF unit = "kg"
            ConvertWeight := GRAMPERKILOGRAM;
        ELSIF unit = "lb"
            ConvertWeight := GRAMPERPOUND;
        ELSIF unit = "ton"
            ConvertWeight := GRAMPERTON;
        ELSE
            OUTPUT("WARNING! INVALID WEIGHT UNIT");
        END IF;
    WHEN 'A' :
        unit := StrTok(line, ":");
        IF unit = "rad"
            ConvertAngle := 1.0;
        ELSIF unit = "deg"
            ConvertAngle := RADIANPERDEGREE;
        ELSE
            OUTPUT("WARNING! INVALID ANGLE UNIT");
        END IF;
    WHEN 'S' :
        unit := StrTok(line, ":");
        IF unit = "mps"
            ConvertSpeed := 1.0;
        ELSIF unit = "kph"
            ConvertSpeed := METERPERKILOMETER / SECPERHOUR;
        ELSIF unit = "mph"
            ConvertSpeed := METERPERMILE / SECPERHOUR;
        ELSE
            OUTPUT("WARNING! INVALID SPEED UNIT");
        END IF;
    WHEN 'T' :
        unit := StrTok(line, ":");
        IF unit = "sec"
            ConvertTime := 1.0;
        ELSIF unit = "min"
            ConvertTime := SECPERMINUTE;
        ELSIF unit = "hr"
            ConvertTime := SECPERHOUR;
        ELSE
            OUTPUT("WARNING! INVALID TIME UNIT");
        END IF;
    WHEN 'X' :
        viewxlo := STRTOREAL(StrTok(line, ","));
        viewylo := STRTOREAL(StrTok(line, ","));
        viewxhi := STRTOREAL(StrTok(line, ","));
        viewyhi := STRTOREAL(StrTok(line, ":"));
        {OUTPUT(viewxlo, ",", viewylo, ",", viewxhi, ",", viewyhi);}
    WHEN 'G' :
        gridspace := STRTOREAL(StrTok(line, ":"));
    WHEN 'V' :
        ch := SCHAR(StrTok(line,":"), 1);
        IF ch = 'T'
            ShowVirtual := TRUE;
        ELSE
            ShowVirtual := FALSE;
        END IF;
    END CASE;

END PROCEDURE;

PROCEDURE ProcessLogicalNetwork(IN command, line : STRING);
VAR
    v, u : VertexObj;
    a, b : Section;
    p : PointType;
    dstr : STRING;
    d : REAL;
    e : EdgeObj;
BEGIN
```

```
    IF STRLEN(command) <> 2
        RETURN;
    END IF;
    CASE SCHAR(command,2)
    WHEN 'V' :          { Logical Vertex }
        NEW(v);
        ASK v TO SetID(StrTok(line, ":"));
        a := ASK SectionDbase TO Find(StrTok(line, ":"));
        b := ASK SectionDbase TO Find(StrTok(line, ":"));
        ASK v TO SetSections(a, b);
        p.x := STRTOREAL(StrTok(line, ","));
        p.y := STRTOREAL(StrTok(line, ":"));
        ASK v TO SetLocation(p);
        ASK NetworkDbase TO Add(v);
    WHEN 'E' :          { Logical Edge }
        u := ASK NetworkDbase TO Find(StrTok(line, ":"));
        v := ASK NetworkDbase TO Find(StrTok(line, ":"));
        dstr := StrTok(line, ":");
        IF IsEmptyString(dstr)
            d := ASK u TO GetDistance(v);
        ELSE
            d := STRTOREAL(dstr);
        END IF;
        NEW(e);
        ASK e TO SetVertex(v);
        ASK e TO SetDistance(d);
        ASK u.adjacent TO Add(e);
    END CASE;
END PROCEDURE;

PROCEDURE ProcessCommandLine(IN buffer : STRING);
VAR
    command : STRING;
BEGIN
    command := StrTok(buffer, ":");
    CASE SCHAR(command,1)
    WHEN 'S' : { Segment, Intersection, Origin Zone, Destin Zone }
        ProcessSection(command, buffer);
    WHEN 'D' : { Detector}
        ProcessDetector(command, buffer);
    WHEN 'C' : { Control, Sign}
        ProcessControl(command, buffer);
    WHEN 'G' : { Global constant}
        ProcessGlobal(command, buffer);
    WHEN 'L' : { Logical Network}
        ProcessLogicalNetwork(command, buffer);
    END CASE;
END PROCEDURE;

PROCEDURE PrintConverter;
BEGIN
    OUTPUT("Length : ", ConvertLength);
    OUTPUT("Weight : ", ConvertWeight);
    OUTPUT("Angle  : ", ConvertAngle);
    OUTPUT("Speed  : ", ConvertSpeed);
    OUTPUT("Time   : ", ConvertTime);
END PROCEDURE;

PROCEDURE DrawLogicalNetwork;
VAR
    vertex  : VertexObj;
    edge    : EdgeObj;
    p       : PointArrayType;
    line    : ArrowPolylineObj;
BEGIN
    NEW(p, 1..2);
    FOREACH vertex IN NetworkDbase
        p[ 1] := vertex.location;
        FOREACH edge IN vertex.adjacent
            p[ 2] := edge.v.location;
            NEW(line);
```

```
            ASK line TO SetPoints(p);
            ASK line TO SetColor(LogicalColor);
            ASK world TO AddGraphic(line);
        END FOREACH;
    END FOREACH;
    ASK window TO Draw();
END PROCEDURE;

PROCEDURE InitialWindow();
VAR
    lib : GraphicLibObj;
    clockimage  : DynDClockObj;
BEGIN
    { Error : ASK window TO SetInitialState(MaximizedWindowState);}
    ResetSimTime(0.0);
    NEW(lib);                    (* Load library created using SIMDRAW       *)
    ASK lib TO ReadFromFile("mobile.sg2");
    NEW(clockimage);
    ASK clockimage TO LoadFromLibrary(lib, "clock");
    ASK clockimage TO SetTranslation(0.75*WorldXhi, 0.75*WorldYhi);
    ASK clockimage TO SetTimeScale(1.0 / 3600.0);
    ASK clockimage TO StartMotion;
    ASK window TO AddGraphic(clockimage);
    DISPOSE(lib);
END PROCEDURE;

PROCEDURE SetDefaultValues();
BEGIN
    BackgroundColor := Green;
    RoadColor := Grey;
    ShoulderColor := LightGrey;
    NormalLineColor := White;
    CenterLineColor := Yellow;
    LogicalColor := Red;
    gridcolor := Blue;
    ShowVirtual := FALSE;
END PROCEDURE;

PROCEDURE AdjustWindow();
VAR
    xratio, yratio : REAL;
    xdist, ydist : REAL;
BEGIN
    xdist := viewxhi - viewxlo;
    ydist := viewyhi - viewylo;
    IF xdist < ydist
        xdist := ydist;
    ELSE
        ydist := xdist;
    END IF;
    ASK world   TO SetWorld(viewxlo, viewylo, viewxlo+xdist, viewylo+ydist);
    xratio := -viewxlo / xdist;
    yratio := -viewylo / ydist;
    ASK world   TO SetTranslation(WorldXhi*xratio, WorldYhi*yratio);
    IF gridspace > 0.1
        GridDraw;
    END IF;
END PROCEDURE;

PROCEDURE InitializeSection;
VAR
    section : Section;
    c : ControlMediaObj;
    d : DetectorObj;
    stat : DetectorStationObj;
    lp, gp : PointType;
BEGIN
    FOREACH section IN SectionDbase
        ASK section TO CreateImage;
        ASK world   TO AddGraphic(section);
    END FOREACH;
```

```
    FOREACH section IN SectionDbase
        FOREACH c IN section.controls
            ASK section TO LocalToGlobal(c.displaylocation, gp);
            ASK c TO SetTranslation(gp.x, gp.y);
            ASK c TO SetRotation(section.orient);
            ASK world   TO AddGraphic(c);
        END FOREACH;
    END FOREACH;

    FOREACH section IN SectionDbase
        FOREACH d IN section.detectors
            ASK section TO LocalToGlobal(d.location, gp);
            ASK d TO SetTranslation(gp.x, gp.y);
            ASK d TO SetRotation(section.orient);
            ASK world   TO AddGraphic(d);
        END FOREACH;
    END FOREACH;

END PROCEDURE;

PROCEDURE ReadSimFile(IN filename : STRING) : BOOLEAN;
VAR
    fin : StreamObj;
    buffer : STRING;
    orimage : ImageObj;
    lib : GraphicLibObj;
BEGIN
    SetDefaultValues();
    InitialWindow();

    NEW(SectionDbase);
    NEW(NetworkDbase);

    ASK window  TO AddGraphic(world);

    NEW(fin);
    ASK fin TO Open(filename, Input);
    WHILE NOT (ASK fin eof)
        ASK fin TO ReadLine(buffer);
        RemoveComment(buffer);
        IF NOT IsEmptyString(buffer)
            ProcessCommandLine(buffer);
        END IF;
    END WHILE;
    ASK fin TO Close;
    DISPOSE(fin);

    AdjustWindow;

    InitializeSection;

    IF ShowVirtual
        DrawLogicalNetwork;
    END IF;

    ASK window TO Draw;
    RETURN TRUE;
END PROCEDURE;

END MODULE.



(*
    DUtil.mod

*)

DEFINITION MODULE Util;
FROM MathMod    IMPORT  pi;
```

```
FROM GTypes     IMPORT  PointType, PointArrayType, ColorType;
FROM Line       IMPORT  PolylineObj;
FROM Image      IMPORT  ImageObj;

TYPE
    SortedPoints = OBJECT
        number, size, side : INTEGER;
        data : PointArrayType;
        ASK METHOD SetSize(IN n : INTEGER);
        ASK METHOD SetSide(IN s : INTEGER);
        ASK METHOD Add(IN p : PointType);
        ASK METHOD Sort;
    END OBJECT;

    ArrowPolylineObj = OBJECT(ImageObj)
        ASK METHOD SetPoints(IN p : PointArrayType);
    OVERRIDE
        ASK METHOD SetColor(IN color : ColorType);
        ASK METHOD ObjInit;
    END OBJECT;

PROCEDURE IsEmptyString(IN s : STRING) : BOOLEAN;
PROCEDURE RemoveComment(INOUT s : STRING);
PROCEDURE StrTok(INOUT s : STRING; IN delimeter : STRING) : STRING;
PROCEDURE IsSpace(IN ch : CHAR) : BOOLEAN;
PROCEDURE CompactString(INOUT s : STRING);
PROCEDURE PTRTOSTR(IN p : PointType) : STRING;
END MODULE.


(*
    IUtil.mod

*)
IMPLEMENTATION MODULE Util;
FROM MathMod    IMPORT  pi, ATAN, COS, SIN;
FROM GTypes     IMPORT  PointType, PointArrayType, ColorType,
                        ALL FillStyleType;
FROM Fill       IMPORT  PolygonObj;
FROM Vector     IMPORT  VectorObj;


CONST
    TAB = 8;
    SPC = 32;
    CR  = 13;
    LF  = 10;
    CM  = '#';

OBJECT SortedPoints;
    ASK METHOD SetSize(IN n : INTEGER);
    BEGIN
        size := n;
        number := 0;
        NEW(data, 1..size);
    END METHOD;

    ASK METHOD SetSide(IN s : INTEGER);
    BEGIN
        side := s;
    END METHOD;

    ASK METHOD Add(IN p : PointType);
    BEGIN
        IF size = number
            OUTPUT("SortedPoints is full");
            RETURN;
        END IF;
        INC(number);
        data[ number]  := p;
    END METHOD;
```

```
    ASK METHOD Sort;
    VAR
        i, k, cnt, mi : INTEGER;
        mp : PointType;
        cx : REAL;
        temp : PointArrayType;
    BEGIN

        FOR i := 1 TO number
            mi := i;        mp := data[ mi] ;
            FOR k := i+1 TO number
                IF (data[ k] .x < mp.x)
                    mi := k;    mp := data[ mi] ;
                ELSIF (data[ k] .x = mp.x) AND (side = 0) AND (data[ k] .y < mp.y)
                    mi := k;    mp := data[ mi] ;
                ELSIF (data[ k] .x = mp.x) AND (side = 1) AND (data[ k] .y > mp.y)
                    mi := k;    mp := data[ mi] ;
                END IF;
            END FOR;
            IF (mi <> i)
                data[ mi]  := data[ i] ;
                data[ i]  := mp;
            END IF;
        END FOR;

        cx := data[ 1] .x;
        cnt := 1;
        FOR i := 2 TO number
            IF data[ i] .x <> cx
                INC(cnt);
                cx := data[ i] .x;
            END IF;
        END FOR;
        {OUTPUT("cnt=", cnt);}

        NEW(temp, 1..cnt);
        temp[ 1]  := data[ 1] ;
        k := 1;
        FOR i := 1 TO number
            IF data[ i] .x <> temp[ k] .x
                INC(k);
                temp[ k]  := data[ i] ;
            END IF;
        END FOR;

        DISPOSE(data);
        data := temp;
        number := cnt;
    END METHOD;

END OBJECT;

OBJECT ArrowPolylineObj;
    ASK METHOD ObjInit;
    VAR
        line : PolylineObj;
        head : PolygonObj;
    BEGIN
        INHERITED ObjInit;
        NEW(line);
        AddChild(line, "line", 0);
        NEW(head);
        AddChild(head, "head", 0);
    END METHOD;

    ASK METHOD SetPoints(IN p : PointArrayType);
    VAR
        q : PointArrayType;
        head : PolygonObj;
        line : PolylineObj;
```

```
        h : INTEGER;
        angle : REAL;
        v : VectorObj;

    BEGIN
        h := HIGH(p);
        NEW(v);
        ASK v TO Init(p[h-1].x - p[h].x, p[h-1].y - p[h].y);
        angle := ASK v TO Angle;

        NEW(q, 1..4);
        q[1] := p[h];
        q[2].x := p[h].x + 2.0 * COS(angle + pi / 12.0);
        q[2].y := p[h].y + 2.0 * SIN(angle + pi / 12.0);
        q[3].x := p[h].x + 1.5 * COS(angle);
        q[3].y := p[h].y + 1.5 * SIN(angle);
        q[4].x := p[h].x + 2.0 * COS(angle - pi / 12.0);
        q[4].y := p[h].y + 2.0 * SIN(angle - pi / 12.0);
        head := Child("head", 0);
        ASK head TO SetPoints(q);
        ASK head TO SetStyle(SolidFill);
        line := Child("line", 0);
        ASK line TO SetPoints(p);
        DISPOSE(q);
        DISPOSE(v);
    END METHOD;

    ASK METHOD SetColor(IN color : ColorType);
    VAR
        head : PolygonObj;
        line : PolylineObj;
    BEGIN;
        line := Child("line", 0);
        ASK line TO SetColor(color);
        head := Child("head", 0);
        ASK head TO SetColor(color);
    END METHOD;

END OBJECT;

PROCEDURE IsEmptyString(IN s : STRING) : BOOLEAN;
VAR
    i : INTEGER;
BEGIN
    IF STRLEN(s) = 0
        RETURN TRUE;
    END IF;

    FOR i := 1 TO STRLEN(s)
        IF NOT IsSpace(SCHAR(s,i))
            RETURN FALSE;
        END IF;
    END FOR;
    RETURN TRUE;
END PROCEDURE;

PROCEDURE RemoveComment(INOUT s : STRING);
VAR
    n : INTEGER;
BEGIN
    IF STRLEN(s) = 0        {  It is null string}
        RETURN;
    END IF;

    IF SCHAR(s,1) = CM      {  It is a wholly comment}
        s := "";
        RETURN;
    END IF;

    FOR n := 1 TO STRLEN(s)
        IF SCHAR(s,n) = CM
```

```
            s := SUBSTR(1,n-1,s);
            RETURN;
        END IF;
    END FOR;
END PROCEDURE;

PROCEDURE StrTok(INOUT s : STRING; IN delimeter : STRING) : STRING;
VAR
    r : STRING;
    p : INTEGER;
BEGIN
    IF (STRLEN(s) = 0) OR (STRLEN(delimeter) = 0)
        RETURN "";
    END IF;

    p := POSITION(s, delimeter);
    IF p = 0
        r := s;
        s := "";
    ELSE
        IF p = 1
            r := "";
        ELSE
            r := SUBSTR(1, p-1, s);
        END IF;
        IF p = STRLEN(s)
            s := "";
        ELSE
            s := SUBSTR(p+1, STRLEN(s), s);
        END IF;
    END IF;
    RETURN r;
END PROCEDURE;

PROCEDURE IsSpace(IN ch : CHAR) : BOOLEAN;
VAR
    val : INTEGER;
BEGIN
    val := ORD(ch);
    IF (val = TAB) OR (val = SPC) OR (val = CR) OR (val = LF)
        RETURN TRUE;
    END IF;
    RETURN FALSE;
END PROCEDURE;

(*************************************************************)
PROCEDURE CompactString(INOUT s : STRING);
VAR
    t : ARRAY INTEGER OF CHAR;
    i, pos, code : INTEGER;

BEGIN
    NEW(t, 1..STRLEN(s)+1);
    pos := 1;
    FOR i:= 1 TO STRLEN(s)
        code := ORD(SCHAR(s,i));
        IF (code <> TAB) AND (code <> SPC)
            t[ pos]  := SCHAR(s,i);
            INC(pos);
        END IF;
    END FOR;
    t[ pos]  := CHR(0);
    s := CHARTOSTR(t);
    DISPOSE(t);
END PROCEDURE;

PROCEDURE PTRTOSTR(IN p : PointType) : STRING;
BEGIN
    RETURN "[ " + REALTOSTR(p.x) + "," + REALTOSTR(p.y) + "]";
END PROCEDURE;
```

```
END MODULE.


(*
    DVector.mod

*)

DEFINITION MODULE Vector;
TYPE
    VectorObj = OBJECT
        x, y : REAL;
        ASK METHOD Init(IN xi, yi : REAL);
        ASK METHOD Value    : REAL;
        ASK METHOD Angle    : REAL;
        ASK METHOD AngleByDegree : REAL;
        ASK METHOD Stretch(IN s : REAL);
        ASK METHOD ObjPrint() : STRING;
    END OBJECT;
END MODULE.

(*
    IVector.mod
*)
IMPLEMENTATION MODULE Vector;
FROM MathMod IMPORT SQRT, SIN, COS, ATAN, pi;
OBJECT VectorObj;
    {Initialize a vector}
    ASK METHOD Init(IN xi, yi : REAL);
    BEGIN
        x := xi;    y := yi;
    END METHOD;

    {return its absolute value}
    ASK METHOD Value : REAL;
    BEGIN
        RETURN SQRT(x*x + y*y);
    END METHOD;

    ASK METHOD Angle : REAL;
    BEGIN
        IF x = 0.0
            IF y > 0.0
                RETURN 0.5 * pi;
            ELSIF y < 0.0
                RETURN -0.5 * pi;
            ELSE
                RETURN 0.0;
            END IF;
        END IF ;
        IF x < 0.0
            RETURN ATAN(y/x)+pi;
        ELSIF y < 0.0
            RETURN ATAN(y/x)+2.0*pi;
        ELSE
            RETURN ATAN(y/x);
        END IF;
    END METHOD;

    ASK METHOD AngleByDegree : REAL;
    BEGIN
        RETURN Angle * 180.0 / pi;
    END METHOD;

    ASK METHOD Stretch(IN s : REAL);
    VAR
        val, theta : REAL;
    BEGIN
        val := Value + s;
        theta := Angle;
        Init(val*COS(theta), val*SIN(theta));
```

```
    END METHOD;

    ASK METHOD ObjPrint() : STRING;
    BEGIN
        RETURN "<" + REALTOSTR(x) + "," + REALTOSTR(y) + ">";
    END METHOD;


END OBJECT;
END MODULE.


(*
    DVehicle.mod

*)

DEFINITION MODULE Vehicle;
FROM GrpMod     IMPORT  BTreeObj;
FROM Animate    IMPORT  DynImageObj;
FROM Sect       IMPORT  Section;
FROM Vector     IMPORT  VectorObj;
FROM GTypes     IMPORT  PointType, PointArrayType;
FROM Network    IMPORT  VertexObj, VertexArrayType;


TYPE
    StatusType = (
        Stopped,
        Following,
        ShiftingLeft,
        ShiftingRight,
        TurningLeft,
        TurningRight
    );

    VehicleObj = OBJECT(DynImageObj);
        id  : STRING;
        sect : Section;
        location : PointType;      { Global coordinate on the given section }
        velocity, accelerate : VectorObj;
        direction : REAL;
        state : StatusType;
        length, width, weight : REAL;
        path : VertexArrayType;
        pathidx : INTEGER;

        ASK  METHOD SetID(IN s : STRING);
        TELL METHOD TravelThrough(IN srcname, dstname : STRING; IN lane : REAL);
        TELL METHOD SpeedUpTo(IN speed, acc : REAL);
        TELL METHOD SpeedUpFor(IN time, acc : REAL);
        TELL METHOD SpeedDownTo(IN time, acc : REAL);
        TELL METHOD SpeedDownFor(IN time, acc : REAL);
        TELL METHOD MoveFor(IN time : REAL);
{
        TELL METHOD StopAt(IN x, y : REAL);
        TELL METHOD LaneShiftLeft;
        TELL METHOD LaneShiftRight;
        TELL METHOD FollowLinkTraffic;
        TELL METHOD WaitRedSignal;
        TELL METHOD TurnLeft;
        TELL METHOD TurnRight;
        TELL METHOD SendSignal;
}
        ASK  METHOD Initialize(IN loc : VertexObj; IN lane : REAL);
{
        ASK  METHOD RemainTravelLength : REAL;
        ASK  METHOD RemainTravelTime : REAL;
        ASK  METHOD ObserveFront;
        ASK  METHOD ObserveBack;
        ASK  METHOD ObserveRightFront;
```

```
        ASK  METHOD ObserveRightBack;
        ASK  METHOD ObserveLeftFront;
        ASK  METHOD ObserveLeftBack;
        ASK  METHOD ObserveHighway;
        ASK  METHOD ObserveRamp;
        ASK  METHOD ObserveControl;
        ASK  METHOD ChangeRoute;
}
    PRIVATE
        TELL METHOD UpdateLocation(IN locations : PointArrayType);
        TELL METHOD TravelBetweenVertex(IN here, there : VertexObj);
        ASK  METHOD SetNextSpeed;
        ASK  METHOD SetSpeedByPolar(IN length, angle : REAL);
    OVERRIDE
        ASK  METHOD ObjInit;
    END OBJECT;

VehicleDbaseObj = OBJECT (BTreeObj[ANYOBJ:VehicleObj]);
OVERRIDE
    ASK METHOD Rank (IN a, b : VehicleObj) : INTEGER;
    ASK METHOD Key (IN a : VehicleObj) : STRING;
END OBJECT;


END MODULE.

(*
    IVehicle.mod

*)

IMPLEMENTATION MODULE Vehicle;
FROM Window     IMPORT  WindowObj;
FROM Animate    IMPORT  DynImageObj, DynDClockObj;
FROM Image      IMPORT  ImageObj;
FROM Graphic    IMPORT  GraphicLibObj;
FROM MathMod    IMPORT  pi AS PI, COS, SIN;
FROM Vector     IMPORT  VectorObj;
FROM GTypes     IMPORT  PointType, PointArrayType,
                        ALL ColorType,
                        ALL FillStyleType,
                        ALL LineStyleType;
FROM Line       IMPORT  PolylineObj;
FROM MathMod    IMPORT  ATAN;
FROM Sect       IMPORT  Section;
FROM Network    IMPORT  VertexObj, NetworkDbase, VertexArrayType;
FROM Global     IMPORT  SIMTIC;

OBJECT VehicleObj;
    ASK METHOD SetID(IN s : STRING);
    BEGIN
        id := s;
    END METHOD;

    TELL METHOD TravelThrough(IN srcname, dstname : STRING; IN lane : REAL);
    VAR
        p : PointType;
        speed, theta : REAL;
        v, src : VertexObj;
    BEGIN
        ASK NetworkDbase TO FindShortestPath(srcname, dstname, path);
        IF path = NILARRAY
            OUTPUT("I am sorry I cannot find a path from ",
                    srcname, " to ", dstname);
            RETURN;
        END IF;
        Initialize(ASK NetworkDbase TO Find(srcname), lane);
        (*
            1.  Find a path from src to dst, possibly the shortest path.
            2.  a := First(path);   b := Next(path);
            2.  while not arrived {
```

```
                    move it from a to b
                    a := b;     b := Next(path);
                }
            3.  Collect statistical data
        *)
        WAIT FOR SELF TO TravelBetweenVertex(path[ 1] , path[ 2] ) END WAIT;
        {
        FOR pathidx := 2 TO HIGH(path)
            TravelBetweenVertex(path[ pathidx-1] , path[ pathidx] );
        END FOR;
        }
    END METHOD;

    TELL METHOD TravelBetweenVertex(IN here, there : VertexObj);
    BEGIN
{       WAIT FOR SELF TO MoveFor(2.0) END WAIT;}
        WAIT FOR SELF TO SpeedDownTo(0.0, 5.0) END WAIT;
        WAIT FOR SELF TO SpeedUpTo(20.0, 5.0) END WAIT;
    END METHOD;

    ASK  METHOD Initialize(IN loc : VertexObj; IN lane : REAL);
    VAR
        speed : REAL;
    BEGIN
        ASK accelerate TO Init(0.0, 0.0);
        sect := loc.entrysection;
        speed := ASK sect TO GetNominalSpeed;
        direction := ASK sect TO orient;
        SetSpeedByPolar(speed, direction);

        ASK sect TO LocalToGlobal(sect.GetXYByLW(0.0, lane), location);
        SetRotation(direction);
        SetTranslation(location.x, location.y);
        Draw();
        DisplayAt(location.x, location.y);
    END METHOD;

    (*
        Asynchronous function : UpdateLocation
        Restriction : For the given global locations, Section shouldn't be changed.
    *)
    TELL METHOD UpdateLocation(IN locations : PointArrayType);
    VAR
        i : INTEGER;
    BEGIN
        FOR i:= 1 TO HIGH(locations)
            WAIT DURATION SIMTIC
                location := locations[ i] ;
            END WAIT;
        END FOR;
    END METHOD;

    (*
        Asynchronous function : MoveFor
        MOve the vehicle for 'time' second with the current speed and direction
        Restriction : For the given time, Section shouldn't be changed.
    *)
    TELL METHOD MoveFor(IN time : REAL);
    VAR
        gp, lp : PointArrayType;
        i, np : INTEGER;
        newx, newy, clock : REAL;
    BEGIN
        IF velocity.Value = 0.0
            RETURN;
        END IF;
        np := TRUNC(time/SIMTIC);
        NEW(gp, 0..np);

        gp[ 0]  := location;
```

```
    FOR i:= 1 TO np
        gp[ i] .x := gp[ i-1] .x + velocity.x * SIMTIC + accelerate.x * SIMTIC*SIMTIC;
        gp[ i] .y := gp[ i-1] .y + velocity.y * SIMTIC + accelerate.y * SIMTIC*SIMTIC;
    END FOR;

    WAIT FOR SELF TO FollowPath(gp) END WAIT;
END METHOD;

TELL METHOD SpeedUpTo(IN speed, acc : REAL);
VAR
    orient, apt : REAL;
    gp  : PointType;
BEGIN
    IF (acc <= 0.0) OR (velocity.Value >= speed)
        RETURN;
    END IF;

    apt := acc * SIMTIC;    { acceleration per SIMTIC }

    IF velocity.Value = 0.0
        orient := direction;
    ELSE
        orient := velocity.Angle;
    END IF;
    ASK accelerate TO Init(apt*COS(orient), apt*SIN(orient));
    gp := location;
    WHILE velocity.Value < speed
        SetNextSpeed;
        gp.x := gp.x + velocity.x * SIMTIC;
        gp.y := gp.y + velocity.y * SIMTIC;
        WAIT FOR SELF TO MoveTo(gp.x, gp.y)
            location := gp;
        END WAIT;
    END WHILE;
END METHOD;

TELL  METHOD SpeedUpFor(IN time, acc : REAL);
VAR
    orient, newx, newy : REAL;
    apt, clock : REAL;
BEGIN
    IF time <= 0.0
        RETURN;
    END IF;

    apt := acc * SIMTIC;    { acceleration per SIMTIC }

    IF velocity.Value = 0.0
        orient := direction;
    ELSE
        orient := velocity.Angle;
    END IF;
    ASK accelerate TO Init(apt*COS(orient), apt*SIN(orient));

    clock := 0.0;
    WHILE clock < time
        SetNextSpeed;
        newx := location.x + velocity.x * SIMTIC;
        newy := location.y + velocity.y * SIMTIC;
        WAIT FOR SELF TO MoveTo(newx, newy);
            location.x := newx;
            location.y := newy;
        END WAIT;
        clock := clock + SIMTIC;
    END WHILE;
END METHOD;

(*
    Asynchronous function : SpeedDownTo
*)
TELL  METHOD SpeedDownTo(IN speed, acc : REAL);
```

```
    VAR
        orient : REAL;
        apt, currspeed : REAL;
        gp : PointType;
    BEGIN
        IF (acc <= 0.0) OR (velocity.Value <= speed)
            RETURN;
        END IF;

        apt := -acc * SIMTIC;    { acceleration per SIMTIC }
        orient := velocity.Angle;
        ASK accelerate TO Init(apt*COS(orient), apt*SIN(orient));
        currspeed := velocity.Value;
        WHILE currspeed > speed
            IF velocity.Value < (acc*SIMTIC)
                ASK accelerate TO Init(0.0, 0.0);
                EXIT;           { Escape from WHILE loop }
            ELSE
                SetNextSpeed;
            END IF;

            gp.x := location.x + velocity.x * SIMTIC;
            gp.y := location.y + velocity.y * SIMTIC;

            WAIT FOR SELF TO MoveTo(gp.x, gp.y);
                location := gp;
            END WAIT;
            currspeed := velocity.Value;
        END WHILE;
    END METHOD;

    TELL  METHOD SpeedDownFor(IN time, acc : REAL);
    BEGIN
        SpeedUpFor(time, -acc);
    END METHOD;

    ASK  METHOD SetSpeedByPolar(IN length, angle : REAL);
    BEGIN
        ASK velocity TO Init(length*COS(angle), length*SIN(angle));
        SetSpeed(length);
    END METHOD;

    ASK  METHOD SetNextSpeed;
    VAR
        dx, dy : REAL;
    BEGIN
        dx := velocity.x + accelerate.x * SIMTIC;
        dy := velocity.y + accelerate.y * SIMTIC;
        ASK velocity TO Init(dx, dy);
        SetSpeed(velocity.Value);
    END METHOD;


    ASK METHOD ObjInit;
    BEGIN
        INHERITED ObjInit;
        NEW(velocity);
        NEW(accelerate);
    END METHOD;
END OBJECT;

OBJECT VehicleDbaseObj;
    ASK METHOD Rank (IN a, b : VehicleObj) : INTEGER;
    BEGIN
        IF a.id < b.id
            RETURN -1;
        END IF;
        IF a.id > b.id
            RETURN 1;
        END IF;
        RETURN 0;
```

```
    END METHOD;
    ASK METHOD Key (IN a : VehicleObj) : STRING;
    BEGIN
        RETURN a.id;
    END METHOD;
END OBJECT;

END MODULE.


(*
    MSejong.mod
 *)

MAIN MODULE Sejong;
FROM GProcs     IMPORT  HandleEvents;
FROM Graphic    IMPORT  GraphicLibObj;
FROM SimIO      IMPORT  ReadSimFile;
FROM SimMod     IMPORT  StartSimulation;
FROM Vehicle    IMPORT  VehicleObj;
FROM Network    IMPORT  NetworkDbase;
FROM Global     IMPORT  world, InitializeGlobal, DisposeGlobal;
FROM Interface  IMPORT  InitializeConnection, CloseConnection, SetStart;
FROM Menus      IMPORT  InitMenuBar, menubar;
FROM Menu       IMPORT  MenuItemObj;
FROM ControlMedia   IMPORT  InitLamps, Lamps, LampObj;

VAR
    fname : STRING;
    rval : INTEGER;
    item : MenuItemObj;

PROCEDURE Generate;
VAR
    veh1, veh2 : VehicleObj;
    lib : GraphicLibObj;
BEGIN
    NEW(lib);                   (* Load library created using SIMDRAW       *)
    ASK lib TO ReadFromFile("mobile.sg2");

    NEW(veh1);
    ASK veh1 TO LoadFromLibrary(lib, "passenger");
    ASK world   TO AddGraphic(veh1);
    ASK  veh1 TO SetID("440-LMM");
    TELL veh1 TO TravelThrough("AA1", "AB5", 1.5);

    NEW(veh2);
    ASK veh2 TO LoadFromLibrary(lib, "passenger");
    ASK world   TO AddGraphic(veh2);
    ASK veh2 TO SetID("Knight");
    TELL veh2 TO TravelThrough("AA2", "AB6", 2.5);

    DISPOSE(lib);
END PROCEDURE;

(* MAIN MODULE *)
BEGIN
    InitializeGlobal;
    InitLamps;
    rval := InitializeConnection;
    fname := "simple.rod";
    IF ReadSimFile(fname)
        {ASK NetworkDbase TO Print();}
        InitMenuBar;
        ASK menubar TO Erase;
        REPEAT
            item := ASK menubar TO AcceptInput();
        UNTIL (item.ReferenceName = "Start");
        Generate;
        rval := SetStart;
        StartSimulation;
```

```
        REPEAT
            item := ASK menubar TO AcceptInput();
        UNTIL (item.ReferenceName = "Exit");
    ELSE
        OUTPUT("Can not find file ", fname);
    END IF;
    rval := CloseConnection;
    {DisposeGlobal;}
END MODULE.
```