**MMITSS Hardware-in-the-loop Simulation System Setup Guide**

1. **System Architecture:**

The physical Hardware-in-the-loop architecture is shown in Figure 1. This is a basic architecture that is applicable at one single intersection under the control of MMITSS. In this framework, the MMITSS vehicle side co-processor (VSP) will be run on a Raspberry Pi with arm-based architecture wired to an OBU; the MMITSS road-side co-processor (MRP) could be run on a laptop, or an arm-based edge device wired to an RSU. OBU is very similar to the RSU, except the OBUs are designed specifically for integration with vehicle electronics system. Both two devices will be run on the same interface. environment for MRP and VSP are supposed to be Ubuntu 20.04, the MMITSS components are expected to be run in docker containers with the specified docker images. Then, the Raspberry Pi and Laptop should be connected to a windows desktop where VISSIM simulation environment will be running on. The VISSIM simulation software is responsible to simulate the priority and background traffic together with the ASCIII signal controller.

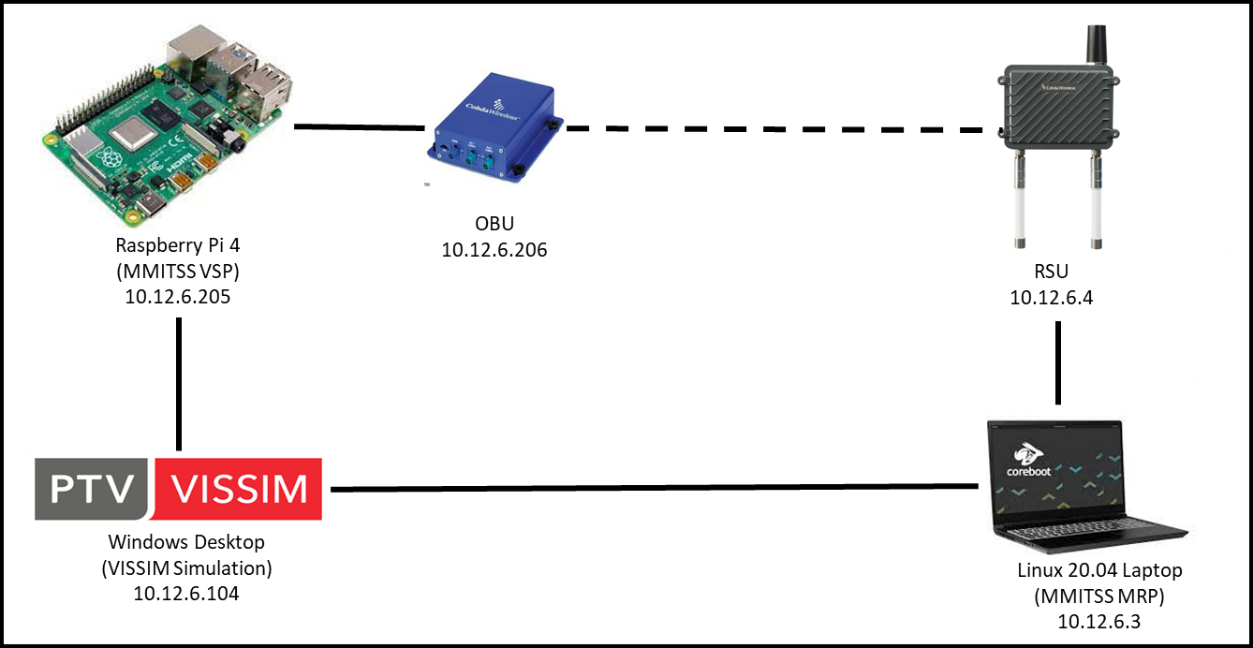


Figure 1: Hardware-in-the-loop Simulation System Architecture

1. **Information Flow:**

**图示

描述已自动生成**

Figure 2: Hardware-in-the-loop Simulation Data Flow

1. **System Setup**

For the simulation desktop, both VISSIM and Econolite ASC/3 signal controller are required to be configurated.

**3.1 VISSIM configuration**

After completing the setup of VISSIM with background traffic, a configuration of driver\_model need to be modified. The configuration file is located under the path of VISSIM driver model dll, which is a custom driver model built from original VISSIM driver-model to connect the simulated vehicles and the MMITSS applications.

To use the driver-models, a directory named “mmitss\_simulation” needs to be placed in the same directory where the “\*.inpx” file where the simulation exists. This directory is provided in the MMITSS simulation distribution package. The “mmitss\_simulation” directory contains a configuration file named “mmitss\_driver\_model\_config.json” and a subdirectory named “driver\_models” where all four prebuilt driver-models are stored. All four driver-models use the same configuration file and the relative location of the “\*.inpx” configuration file cannot be altered. The configuration file establishes the communication address where the UDP packets are to be sent. In addition, the GPS coordinates of the VISSIM model origin (0,0) is set in the configuration file. This information is required to correctly transform the vehicle’s current location from the VISSIM’s local coordinate system to the GPS coordinate system (WGS-84).

The following is the sample code for this hardware-in-the-loop case for the content of the “mmitss\_driver\_model\_config.json”:

{

""client\_ip": "10.12.6.206",

"client\_port": 20001,

"VISSIM\_origin\_position": {

"elevation\_Meter": 540.11,

"latitude" : {

"Degree" : 33.0,

"Minute" : 50.0,

"Second" : 35.4

},

"longitude" : {

"Degree" : -112.0,

"Minute" : -8.0,

"Second" : -6.1

}

}

}

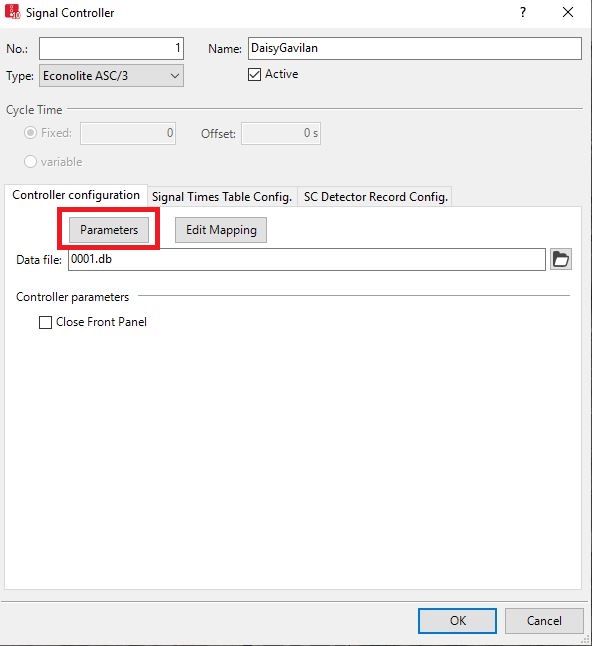
Then the driver model will pack the connected vehicles information from VISSIM as a data structure called “BSM Blob” (referenced by MMITSS development group: <https://github.com/mmitss/mmitss-az/tree/master/src/simulation#readme> ) to the VSP devices for further encoding and distribution.

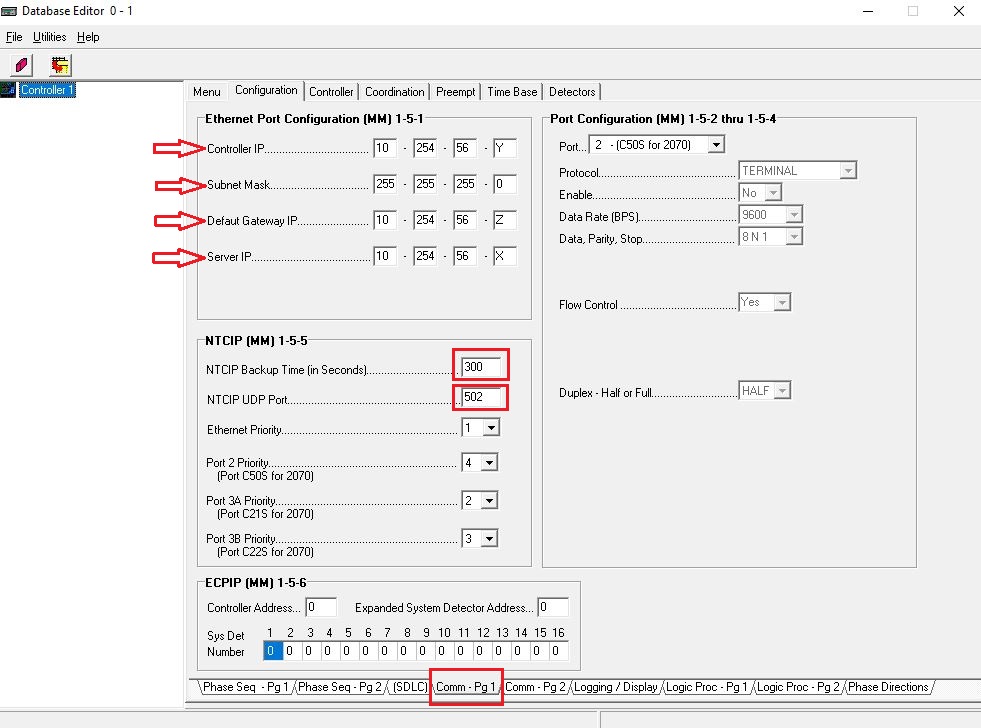
**3.2 Econolite ASC/3 signal controller Configuration**

The following steps are required to configure Econolite ASC/3 signal controller in VISSIM.

(a) Determine the IP address of the MS Windows computer where VISSIM is installed. All the components in this system should be in the same subnet. For example, if the VISSIM device host IP address is: 10.12.6.xx, then the MRP, VSP, OBU and RSU must be 10.12.6.yy, where xx and yy are determined by the user.

(b) Next define the controller IP, subnet mask, default gateway IP, server IP, NTCIP backup time and NTCIP UDP Port in the VISSIM ASC/3 controller. From the VISSIM open each signal controller and select “Parameters” option. Go to the Comm-Pg1 option (MM 1-4) under configuration and define the parameters.





1. **Cohda OBU/RSU Setup**

Both devices need to be installed with the RSU image. The current workable version is: mk5-17.Release.108905-RSU1609-typical.sqsh

Scripts that enables the IMF and real-time forwarding functions needs to be executed on the devices. Before executing the scripts, the devices should be placed where the GSP corrections can be received. Otherwise, the script will not be executed successfully.

The name of the script in the obu is: cw\_rsu41\_setup.sip\_obu.sh

The name of the script in the RSU is: cw\_rsu41\_setup.sip\_rsu.sh

The current scripts support broadcasting and forwarding six types of J2735 messages: BSM, SPaT, MAP, SRM, SSM, and RSM. More message types can be added if needed.

The scripts listen to UDP port 1516 to receive a text string following the USDOT RSU4.1 standard and broadcast through the DSRC interface. The broadcast parameters such as channel, PSID, and mode can be modified through functions \_set\_IMFX() in the script.

The received messages from other devices will be forwarded through UDP to a designated address and port. This address and port can be set up through the parameters FW\_DSRC\_ADDR1 and FW\_DSRC\_PORT1 in the script. Multiple addresses and ports can be set through the \_set\_WSMFwdRxX() functions.

tcpdump can be used to monitor the communications through the UDP (-i eth0) and DSRC (-i wave-raw).

1. **VSP/MRP Setup**

MMITSS can support different type of device with different user requirements. So it has two different versions for vehicle side processor (VSP) and road side processor (MRP). The setup process for VSP and MRP are similar. However, current VSP is supposed to be run on an ARM architecture edge device while the MRP can support both x86 and ARM architecture. A python script named “bsm\_distribution.py”, which is designed to encoder J2735 format BSM from the BSM Blob received from the VISSIM. In the main function of the python script, communication configurations need to be setup for receiving BSM blob from VISSIM device and sending encoded BSM package to OBU and MMITSS VSP port.

$python3 bsm\_distribution.py

The MMITSS VSP is supposed to be built in a docker container with the image from dockerhub. There are two ways to build the MMITSS applications. One is to build a docker container with the predefined docker image and scripts, the other is to build from source. The following are the steps to build the docker container in edge device. To begin with, the main folder that includes scripts and source code of building mmitss needs to be downloaded to a root folder from the github by git clone.

$ git clone <https://github.com/mmitss/mmitss-az.git>

1. **Create configuration file**

It is required to create mmitss-phase3-master-config.json, mmitss-coordination-plan.json, and mmitss-data-external-clients.json configuration files **for the MRP container** and mmitss-phase3-master-config.json, and mmitss-bus-stop-location.json configuration files for **the VSP container**. The mmitss-phase3-master-config.json configuration file contain the IP addresses, UDP ports, and other configuration data which are required to establish communication between the MMITSS software components. The appendix A is the detailed and example for these configurations.

1. **Build docker container**

Run the container-build script to build the docker container. The scripts for building MMITSS VSP/MRP containers is located under the path

“..MMITSS\_ROOT/mmitss-az/build/scripts/deployment/”

There are two scripts: setup-deployment-environment.sh and launch-container.sh in this folder. The first script will setup the required environment variables for running MMITSS applications on the device. Thus, it only needs to be run at the first time to build MMITSS on the devices or change the environment variables. The second script is used for building docker containers. Run the launch-container.sh script to run the docker container.

* The container can be start by executing the following command:

launch-container.sh

A user interface will appear and following information must be provided (An example for VSP container).

Full absolute path of MMITSS configuration directory: <path-to-mmitss>/emergency-vehicle

Name of container image on the Dockerhub: mmitssuarizona/mmitss-vsp-arm:1.1

Name of container: vsp\_container

* To monitor the containers, execute the following command:

docker container exec –it <container name> /bin/bash

* To stop all the containers, execute the following command:

docker container stop <container name>

* To start container the execute the following command:

docker container start <container name>

The following table is the sample variables required for building VSP and MRP.

|  |  |  |
| --- | --- | --- |
|  | VSP | MRP |
| Required configurations | mmitss-phase3-master-config.json | mmitss-phase3-master-config.json  mmitss-coordination-plan.json  mmitss-data-external-clients.json |
| Docker contianer | mmitssuarizona/mmitss-vsp-arm:2.2 | mmitssuarizona/mmitss-mrp-arm:2.2  mmitssuarizona/mmitss-mrp-x86:2.2 |
| Name of container | vsp\_contianer | mrp\_container |
| Net adaptor | Wired adaptor id | Wired adaptor id |
| Time zone | America/New\_York | America/New\_York |

**Appendix A: Example for MMITSS configurations.**

An example of mmitss-phase3-master-config.json is following:

{

"HostIp": "xxx.xxx.xxx.xxx",

"SourceDsrcDeviceIp": "xxx.xxx.xxx.yyy",

"IntersectionName": "xxx",

"MapPayload":001283fe38083020315abe2149d0eecf1800a0000271c4fcbd028280",

"IntersectionID" : XXXX,

"RegionalID" : 0,

"DataCollectorIP": "xxx.xxx.xxx.xyx",

"HMIControllerIP": "xxx.xxx.xxx.yxx",

"MessageDistributorIP": " xxx.xxx.xxx.zzz ",

"PriorityRequestGeneratorServerIP": "xxx.xxx.xxx.zzz",

"VehicleType" : “Transit”,

"Logging" : "True",

"SRMTimedOutTime" : 10.0,

"ScheduleExecutionBuffer": 1.0,

"SystemPerformanceTimeInterval": 300.0,

"ApplicationPlatform": "roadside",

"PeerDataDecoding": false,

"CoordinationPlanCheckingTimeInterval": 300,

"PortNumber":{

"MessageTransceiver":{

"MessageSender": 10003,

"MessageReceiver": 10002,

"MessageEncoder": 10003,

"MessageDecoder": 10002

},

"MessageDistributor": 5000,

"RsmDecoder": 10006,

"OBUBSMReceiver": 10005,

"HostBsmDecoder": 10005,

"TrajectoryAware": 20001,

"PriorityRequestServer": 20002,

"PrioritySolver": 20003,

"PriorityRequestGenerator": 20004,

"TrafficControllerInterface": 20005,

"TrafficControllerCurrPhaseListener": 20006,

"TrafficControllerTimingPlanSender": 20007,

"PerformanceObserver": 20008,

"HMIController": 20009,

"PrioritySolverToTCIInterface": 20010,

"SignalCoordination": 20011,

"MapSPaTBroadcaster": 6053,

"DsrcImmediateForwarder": 1516,

"PriorityRequestServer\_SendSSM": 50003,

"DataCollector": 30001,

"SnmpEngine": 20020,

"SnmpEngineInterface": 20021,

"PriorityRequestGeneratorServer": 20022,

"TrajectoryAware\_MapEngineInterface": 20023,

"MapEngine": 20024,

"LightSirenStatusManager": 20025,

"PeerToPeerPriority": 20026 "SnmpEngine": 20020,

"SnmpEngineInterface": 20021,

"PriorityRequestGeneratorServer": 20022

},

"psid": {

"map": "8002",

"spat": "8002",

"rsm": "8002",

"srm": "8002",

"ssm": "8002",

"bsm": "20"

},

"msgId": {

"map": "0012",

"spat": "0013",

"rsm": "0021",

"srm\_lower": "001d",

"srm\_upper": "001D",

"ssm\_lower": "001e",

"ssm\_upper": "001E",

"bsm": "0014"

},

"TxChannel": {

"map": 172,

"spat": 172,

"rsm": 172,

"srm": 182,

"ssm": 182,

"bsm": 172

},

"TxMode": {

"map": "CONT",

"spat": "CONT",

"rsm": "CONT",

"srm": "ALT",

"ssm": "ALT",

"bsm": "CONT"

},

"SignalController": {

"IpAddress": "xxx.xxx.xxx.yyy",

"NtcipPort": 501,

"TimingPlanUpdateInterval\_sec": 600,

"NtcipBackupTime\_sec": 300,

"Vendor": "Econolite",

"TimingPlanMib": "/nojournal/bin/EconoliteMib.py",

"InactiveVehPhases": [],

"InactivePedPhases": [],

"SplitPhases": {

"1": 6,

"3": 8,

"5": 2,

"7": 4

},

"PermissiveEnabled": {

"1": true,

"3": true,

"5": true,

"7": true

}

},

"IntersectionReferencePoint": {

"Latitude\_DecimalDegree": 33.82249812,

"Longitude\_DecimalDegree": -112.2315621233,

"Elevation\_Meter": 906

},

"DataTransfer": {

"FtpServerPort": 9090,

"StartTime": {

"Hour": 18,

"Minute": 30

},

"EndTime": {

"Hour": 19,

"Minute": 30

},

"MaxRetries": 3

},

"PriorityParameter": {

"EmergencyVehicleWeight": 1.0,

"EmergencyVehicleSplitPhaseWeight": 0.1,

"TransitWeight": 1.0,

"TruckWeight": 1.0,

"DilemmaZoneRequestWeight": 2.0,

"CoordinationWeight": 0.1

}

}

The mmitss-coordination-plan.json file contain the coordination plan parameter and split data to run priority-based coordination system of MMITSS. An example of mmitss-coordination-plan.json file is following:

{

"IntersectionName": "xxx",

"CoordinationParameters": [

{

"CoordinationPlanName": "AM-Plan",

"CoordinationPatternNo": 1,

"SplitPatternNo": 1,

"CycleLength": 90,

"Offset": 0,

"CoordinationStartTime\_Hour": 6,

"CoordinationStartTime\_Minute": 30,

"CoordinationEndTime\_Hour": 9,

"CoordinationEndTime\_Minute": 30,

"CoordinationSplit": 20.0,

"CoordinatedPhase1": 2,

"CoordinatedPhase2": 6,

"SplitPatternData": {

"PhaseNumber": [

1,

2,

3,

4,

5,

6,

7,

8

],

"Split": [

15,

39,

12,

24,

15,

39,

16,

20

]

}

},

{

"CoordinationPlanName" :"PM-Plan",

"CoordinationPatternNo": 2,

"SplitPatternNo": 2,

"CycleLength": 90,

"Offset": 10,

"CoordinationStartTime\_Hour": 15,

"CoordinationStartTime\_Minute": 30,

"CoordinationEndTime\_Hour": 19,

"CoordinationEndTime\_Minute": 0,

"CoordinationSplit": 20.0,

"CoordinatedPhase1": 2,

"CoordinatedPhase2": 6,

"SplitPatternData": {

"PhaseNumber": [

1,

2,

3,

4,

5,

6,

7,

8

],

"Split": [

19,

35,

12,

24,

19,

35,

16,

20

]

}

}

]

}

The mmitss-bus-stop-location.json file is required only for the transit vehicles. Transit vehicle dwells in the bus stop for boarding and taking off passengers. MMITSS software component (priority-request-generator) is designed to send priority request after passing the bus stop (for transit vehicle). The mmitss-bus-stop-location.json file. It contains the information of the bus stop location for each transit vehicle (depends on the travel route). An example of mmitss-bus-stop-location.json file is following:

{

"NoOfBusStop": 5,

"BusStopInformation": [

{

"IntersectionName": "xx-yy",

"IntersectionID": xy,

"TravelDirection": "EastBound",

"ApproachNo": 3,

"Latitude\_DecimalDegree": 33.142863,

"Longitude\_DecimalDegree": -110.134406,

"Elevation\_Meter": 739

},

{

"IntersectionName": "yy-xx",

"IntersectionID": yx,

"TravelDirection": "EastBound",

"ApproachNo": 5,

"Latitude\_DecimalDegree": 32.250825,

"Longitude\_DecimalDegree": -112.416047,

"Elevation\_Meter": 960

},

{

"IntersectionName": "yy-zz",

"IntersectionID": yz,

"TravelDirection": "WestBound",

"ApproachNo": 1,

"Latitude\_DecimalDegree": 30.289396,

"Longitude\_DecimalDegree": -112.112475,

"Elevation\_Meter": 687

},

{

"IntersectionName": "zz-yy",

"IntersectionID": zy,

"TravelDirection": "WestBound",

"ApproachNo": 1,

"Latitude\_DecimalDegree": 34.346593,

"Longitude\_DecimalDegree": -108.183794,

"Elevation\_Meter": 546

},

{

"IntersectionName": "xx-zz",

"IntersectionID": xz,

"TravelDirection": "WestBound",

"ApproachNo": 6,

"Latitude\_DecimalDegree": 35.9643024,

"Longitude\_DecimalDegree": -114.934988,

"Elevation\_Meter": 839

}

]

}

For the MRP container, “HostIp”, “SourceDsrcDeviceIp”, “IntersectionName”, “MapPayload”, “IntersectionID”, signal controller “IpAddress”, “NtcipPort”, and “NtcipBackupTime\_sec”, “Vendor”, “TimingPlanMib” are required. The “HostIP” must match the ip address of the connected vehicle co-processor (CVCP). The “SourceDsrcDeviceIp” must be the RSU IP. The map payload can be obtained by creating an intersection map using USDOT map tool (https://webapp.connectedvcs.com/isd/).

For the VSP container, “HostIp”, “SourceDsrcDeviceIp”, “VehicleType” are required. The vehicle type must be one of following strings:

i. “Transit”

ii. “Truck”

iii. “EmergencyVehicle”

Create a log folder which must be placed in the same directory structure of the mmitss-phase3-master-config.json file. To log the data, specify “Logging”: “True” in the mmitss-phase3-master-config.json file otherwise specify it as “False”.