REVERSE ENGINEERING WATERS CHALLENGE 2019

Generated by Doxygen 1.8.13

Wed Jan 13 2021 06:46:49

Contents

Chapter 1

Reverse Engineering Waters Challenge 2019

1.1 Introduction

This project is developed to reverse engineer the Waters Challenge 2019 model and implement it on Jetson TX2 platform using the capability of GPU accelerator. The processing power offered by GPUs and their capability to execute parallel workloads is exploited to execute and accelerate applications related to advanced driver assistance systems.

The challenge consists in analytically master the complex HW/SW system (that will be available as Amalthea model) to answer the following questions:

Response Time Computation:

a) Given an application consisting of a set of dependent tasks and a given mapping, calculate its end-to-end response time.

b) The response time should account for the time for the copy engine to transfer data between the CPU and the GPU.

c) The response time should account for the time for the data transfers between the CPU and the shared main memory considering a read/execute/write semantic.

d) Optionally the off-loading mechanism (synchronous/asynchronous) can be changed to further optimize the end-to-end latencies.

The below sections consists of the steps for headless installation the RT-Linux on a Jetson TX2 platform.

1.2 Installation Setup

The application is implemented on NVIDIA Jetson TX2 board. At a high-level abstraction, embedded heterogeneous SoCs (System of Chip) featuring GP-GPU accelerators are characterized by the following hardware components:

1) CPU

2) Accelerator

3) Memory hierarchy

However, the code can be compiled and executed on any NVIDIA supported hardware device.

The application uses three ARM A57 cores for running the applications parallelly. Each task on the CPU core follows RMS(Rate Monotonic Scheduling) approach. This scheduling approach is core specific. The kernel must support real time time scheduling in order to execute this application.

The installation setup process mentioned below is on x86 Host machine running Ubuntu 18.04 64 bit Linux version.

1.2.1 Jetson TX2 Headless installation

Connect your Jetson TX2 device to your host machine via the USB cable and connect a wired ethernet to the device. Power on the device.

Download the NVIDIA SDK Manager from the NVIDIA website on your host machine.

- Start the sdkmanager and follow the steps to install the packages for the Jetson TX2 device.
- Choose the latest Jetson OS image available.
- The SDK manager will prompt to flash the OS on the Jetson TX2 target device.
- Switch the Jetson TX2 device to recovery mode by pressing and holding the RECOVERY button and then press and release the RESET button.
- Release the RECOVERY button after 2 seconds.
- Wait for the SDK manager to flash the OS on the Jetson TX2 target device.
- Install screen utility on the host Linux machine.
- The USB port for the Jetson TX2 device will appear in the /dev folder of the Host machine with value as ttyACM∗ device.
- Start the screen utility at a baud rate of 115200. screen /dev/ttyACM0 115200.
- The screen prompt will enable for the target device which can be used to configure the device. Follow the steps as prompted.
- Execute ifconfig command to get the IP address. Once the IP address is known, ssh can be used to connect to the Jetson TX2 board.

1.2.2 Installing Real Time Kernel on NVIDIA Jetson TX2 board.

Follow below steps to install the SMP PREEMPT RT on NVIDIA Jetson TX2 board. Login to the NVIDIA Jetson TX2 device through ssh. Create a folder and open a terminal in that folder.

- git clone https://github.com/jetsonhacks/buildJetsonTX2Kernel.git
- checked out the tag vL4T32.1.0 - Same Tegra version as on the board
- Executed the script sudo ./getKernelSources.sh followed by sudo ./makeKernel.sh - Kernel will build successfully.
- cd /usr/src/kernel/kernel-4.9/scripts/rt-patches.sh. Executed the patch as : sudo ./scripts/rt-patches.sh apply-patches. Patch should apply successfully.
- Go to the buildJetsonTX2Kernel folder.
- sudo ./makeKernel.sh
- sudo ./copyImage.sh - Everything compiled successfully and the image is also copied in the /boot folder. -sudo reboot
- Login to the Jetson TX2 board and execute the command uname -a. The printed string must contain SMP PREEMPT RT in it.

1.3 Development Environment

NVIDIA Eclipse Nsight edition is used for the development and debug of the application.

1.3.1 Running the Application

- git clone https://github.com/anand6105/MBSE.git
- cd MBSE
- make all
- jetsonsim executable will generated.
- Run the jetsonsim executable with sudo privileges.

The application prints the time taken by each task in execution on the NVIDIA GPU device.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 detectObject_t Struct Reference

Public Attributes

- int bboxDeviceDetection

  Boundary box device detection.
- int bboxHostDetection

  Boundary box host detection.
- int imageHostDetection

  Image host detection.
- int imageDeviceDetection

  Image device detection.

The documentation for this struct was generated from the following file:

- inc/mbseCuda.h

4.2 plannerData_t Struct Reference

Public Attributes

- int canData

  Can polling data.
- int yawRate

  Yaw rate in x-direction.
- int velocity

  Velocity of the vehicle.
- int xCar

  Position of the vehicle on x co-ordinate.
- int yawCar

  Yaw rate in y-direction.
- int yCar

Position of the vehicle on y co-ordinate.

- int matrixSFM

  Input SFM Data.
- int bBoxHost

  Input detection data.
- int occupancyGrid

  Occupancy Grid.
- int laneBoundary

  Lane boundary.
- int steerObjective

  Steer control data.
- int **speedObjective**

The documentation for this struct was generated from the following file:

- inc/mbseCuda.h

4.3 sfmData_t Struct Reference

Public Attributes

- int matrixSFMHost

  SFM matrix host data.
- int imageSFMHost

  SFM Image host data.

The documentation for this struct was generated from the following file:

- inc/mbseCuda.h

Chapter 5

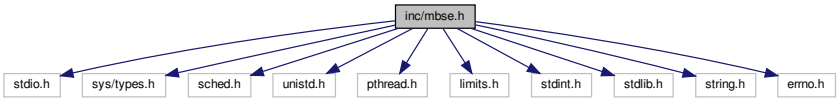
File Documentation

5.1 inc/mbse.h File Reference

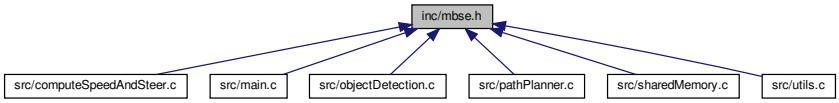
This header file used for all C specific source files.

```
#include <stdio.h>
#include <sys/types.h>
#include <sched.h>
#include <unistd.h>
#include <pthread.h>
#include <limits.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
```

Include dependency graph for mbse.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **_GNU_SOURCE**
- #define **MBSE_NUMBER_OF_THREADS** 6
- #define **MBSE_THREAD_STACK_SIZE** (100 ∗ 1024) /∗ 100 kB is enough for now. ∗/
- #define **NSEC_PER_SEC** (1000 ∗ 1000 ∗ 1000)
- #define **MICRO_SECONDS** 1000
- #define **MILLI_SECONDS** (MICRO_SECONDS ∗ 1000)
- #define **SECONDS** (MILLI_SECONDS ∗ 1000)

Functions

- void error (int at)

  Function to print error message thrown from a particular point in application code.
- void utilAddDelay (uint32_t ms, struct timespec ∗deadline)

  Function to add delay to the task.
- void utilSetThreadPriority (pthread_t threadId, int prio)

  Function to set the thread priority.
- void ∗ pathPlannerCanBusPolling (void ∗args)

  This task is used perform the CAN Bus Polling task functionality.
- void ∗ pathPlannerCalculation (void ∗args)

  This task is used perform the Planner task functionality.
- void ∗ objDetectGetObject (void ∗args)

  This task is used perform the detection functionality.
- void ∗ objDetectStructureFromMotion (void ∗args)

  This task is used perform the Structure-From-Motion functionality.
- void ∗ computeSpeedAndSteer (void ∗args)

  This function implements the DASM task.
- void ∗ computeOSOverhead (void ∗args)

  This task is used to add the OS overhead to the overall tasks based on the Amalthea task model.
- int shmemReadPlannerBufferLabel (unsigned int index)

  Function to read to the Planner Output data buffer.
- int shmemReadLaneBoundaryBufferLabel (unsigned int index)

  Function to read to the lane boundary data buffer.
- int shmemReadGridDataBufferLabel (unsigned int index)

  Function to read to the data grid buffer.
- void shmemWritePlannerDataOutLabel (int offset, int size, int data)

  Function to write to the Planner data buffer.
- void shmemWriteSFMDetectionDataInLabel (int offset, int size, int data)

  Function to write to the SFM and detection Input buffer.
- void shmemWriteGridDataBufferLabel (int offset, int size, int data)

  Function to write to the data grid buffer.
- void shmemWriteLaneBoundaryBufferLabel (int offset, int size, int data)

  Function to write to the lane boundary buffer.
- void shmemWriteDetectionDataOutLabel (int offset, int size, void ∗data)

  Function to write to the Detection data buffer.
- int shmemReadSFMDetectionDataInLabel (unsigned int index)

  Function to read to the Detection and SFM Input data buffer.
- void shmemWriteSFMDataOutLabel (int offset, int size, void ∗data)

  Function to write to the SFM data buffer.
- int shmemReadSFMDataOutLabel (unsigned int index)

  Function to read to the SFM Output data buffer.
- int shmemReadDetectionDataOutLabel (unsigned int index)

  Function to read to the Detection Output data buffer.

5.1.1 Detailed Description

This header file used for all C specific source files.

Author

Anand Prakash

Date

22 April 2020

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.html

5.1.2 Function Documentation

5.1.2.1 computeOSOverhead()

```
void* computeOSOverhead (
            void * args )
```

This task is used to add the OS overhead to the overall tasks based on the Amalthea task model.

This task adds an additional overhead to the overall application. It is used to simulate the overhead that occur in real scenario. It is executed on core number 3 on Jetson TX2 ARM A57 core with the thread priority of 99.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

void

5.1.2.2 computeSpeedAndSteer()

```
void* computeSpeedAndSteer (
            void * args )
```

This function implements the DASM task.

This task computes and establishes the speed and steer that must be effectively employed from the information that is provided by the Path Planner task. It is executed on core number 3 on Jetson TX2 ARM A57 core with the thread priority of 99.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

void

5.1.2.3 error()

```
void error (
            int at )
```

Function to print error message thrown from a particular point in application code.

This function prints the error code from which the error was thrown.

Parameters

in	*at*	Error code.

Returns

void

5.1.2.4 objDetectGetObject()

```
void* objDetectGetObject (
            void * args )
```

This task is used perform the detection functionality.

This task is responsible of detecting and classifying the objects in the road. All the objects detected are visualized and the information produced is sent to the Planner task. It is executed on core number 0 on Jetson TX2 ARM A57 core with the thread priority of 98.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

void

5.1.2.5 objDetectStructureFromMotion()

```
void* objDetectStructureFromMotion (
            void * args )
```

This task is used perform the Structure-From-Motion functionality.

Structure-From-Motion is a method for estimating 3-D structures (depth) from vehicle motion and sequences of 2-D images. This task returns a matrix of points representing the distance with respect the objects of the image. It is executed on core number 0 on Jetson TX2 ARM A57 core with the thread priority of 99.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

void

5.1.2.6 pathPlannerCalculation()

```
void* pathPlannerCalculation (
            void * args )
```

This task is used perform the Planner task functionality.

The main purpose of this component is to define and follow a given trajectory. This trajectory is defined as a spline, that is, a line built through polynomial interpolation at times on the map that represents at each point the position and orientation that the car will have to follow. The planner sends the goal state of the vehicle (i.e., target steer and speed) to the DASM task that is in charge of writing the commands in the CAN line the effective steer and speed to apply. It is executed on core number 5 on Jetson TX2 ARM A57 core with the thread priority of 98.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

void

5.1.2.7 pathPlannerCanBusPolling()

```
void* pathPlannerCanBusPolling (
            void * args )
```

This task is used perform the CAN Bus Polling task functionality.

This task snoops the key vehicle information (steer/wheel/break/acceleration status...) from the on-board CAN bus and sends it to the Localization, Planner and EKF tasks. It is executed on core number 5 on Jetson TX2 ARM A57 core with the thread priority of 99.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

void

5.1.2.8 shmemReadDetectionDataOutLabel()

```
int shmemReadDetectionDataOutLabel (
            unsigned int index )
```

Function to read to the Detection Output data buffer.

This function read the Detection output data buffer. This buffer is used as an input to Planner task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.1.2.9 shmemReadGridDataBufferLabel()

```
int shmemReadGridDataBufferLabel (
            unsigned int index )
```

Function to read to the data grid buffer.

This function read the data grid buffer. This buffer is used as an input to Planner task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.1.2.10 shmemReadLaneBoundaryBufferLabel()

```
int shmemReadLaneBoundaryBufferLabel (
            unsigned int index )
```

Function to read to the lane boundary data buffer.

This function read the lane boundary data buffer. This buffer is used as an input to Planner task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.1.2.11 shmemReadPlannerBufferLabel()

```
int shmemReadPlannerBufferLabel (
            unsigned int index )
```

Function to read to the Planner Output data buffer.

This function read the Planner output data buffer. This buffer is used as an input to DASM task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.1.2.12 shmemReadSFMDataOutLabel()

```
int shmemReadSFMDataOutLabel (
            unsigned int index )
```

Function to read to the SFM Output data buffer.

This function read the SFM output data buffer. This buffer is used as an input to Planner task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.1.2.13 shmemReadSFMDetectionDataInLabel()

```
int shmemReadSFMDetectionDataInLabel (
            unsigned int index )
```

Function to read to the Detection and SFM Input data buffer.

This function read to the Detection and SFM Input data buffer. This buffer is used as an input to SFM and Detection task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.1.2.14 shmemWriteDetectionDataOutLabel()

```
void shmemWriteDetectionDataOutLabel (
            int offset,
            int size,
            void * data )
```

Function to write to the Detection data buffer.

This function writes to the Detection data buffer. This buffer is used as an input to Planner task.

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.1.2.15 shmemWriteGridDataBufferLabel()

```
void shmemWriteGridDataBufferLabel (
            int offset,
            int size,
            int data )
```

Function to write to the data grid buffer.

This function writes to the data grid buffer. This buffer is used as an input to Planner task

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.1.2.16 shmemWriteLaneBoundaryBufferLabel()

```
void shmemWriteLaneBoundaryBufferLabel (
            int offset,
            int size,
            int data )
```

Function to write to the lane boundary buffer.

This function writes to the lane boundary buffer. This buffer is used as an input to Planner task

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.1.2.17 shmemWritePlannerDataOutLabel()

```
void shmemWritePlannerDataOutLabel (
            int offset,
```

```
            int size,
            int data )
```

Function to write to the Planner data buffer.

This function writes to the Planner data buffer. This buffer is used as an input to DASM task.

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.1.2.18 shmemWriteSFMDataOutLabel()

```
void shmemWriteSFMDataOutLabel (
            int offset,
            int size,
            void * data )
```

Function to write to the SFM data buffer.

This function writes to the SFM data buffer. This buffer is used as an input to Planner task.

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.1.2.19 shmemWriteSFMDetectionDataInLabel()

```
void shmemWriteSFMDetectionDataInLabel (
            int offset,
            int size,
            int data )
```

Function to write to the SFM and detection Input buffer.

This function writes to the SFM and detection Input buffer. This buffer is used as an input to SFM and detection task.

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.1.2.20 utilAddDelay()

```
void utilAddDelay (
            uint32_t ms,
            struct timespec * deadline )
```

Function to add delay to the task.

This function adds delay to the task by sleeping for the time provided in the argument in terms of millisecond.

Parameters

in	*ms*	Time in millisecond for which the thread needs to sleep.
in,out	*deadline*	Pointer to the timespec for the next deadline

Returns

void

5.1.2.21 utilSetThreadPriority()

```
void utilSetThreadPriority (
            pthread_t threadId,
            int customPrio )
```

Function to set the thread priority.

This task sets the thread priority based on the threadId and the customPrio provided The priority of the thread is set to max priority minus the custom priority

Parameters

in	*threadId*	Thread ID whose priority needs to be set.
in	*customPrio*	Priority offset of the thread from the maximum priority

Returns

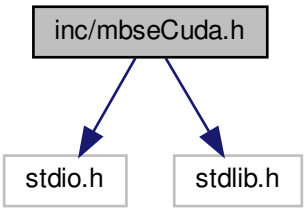
void

5.2 inc/mbseCuda.h File Reference

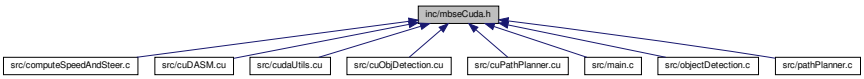
This header file is used for declaring all CUDA specific source files..

```
#include <stdio.h>
#include <stdlib.h>
```

Include dependency graph for mbseCuda.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct detectObject_t
- struct sfmData_t
- struct plannerData_t

Macros

- #define **minVal**(a, b) ((a > b) ? b : a)

Typedefs

- typedef struct detectObject_t **detectObject**
- typedef struct sfmData_t **sfmData**
- typedef struct plannerData_t **plannerData**

Functions

- void cuDetectObject (const char *function, detectObject *objdetected)

  Function to process the Detection task.
- void cuObjDetectSFM (const char *func, sfmData *sfmInputData)

  Function to process the SFM task.
- void cuPlannerFetchCanBusData (const char *func, int *hostCanPollingData)

  Function to process the CAN bus polling task.
- void cuPlannerInterpolatePath (const char *func, plannerData *data)

  Function to process the Planner task.
- void cuProcessDASM (const char *func, int *steer, int *speed)

  Function to process the DASM task.
- void addOSOverhead (const char *func)

  Function to process the OS overhead task.
- void getCudaDeviceProperties (void)

  Function to get the CUDA device properties.

5.2.1 Detailed Description

This header file is used for declaring all CUDA specific source files..

Author

Anand Prakash

Date

22 April 2020

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.html

5.2.2 Function Documentation

5.2.2.1 addOSOverhead()

```
void addOSOverhead (
            const char * func )
```

Function to process the OS overhead task.

The functions adds some OS overhead to simulate the real scenario.

Parameters

func[in]	Function name

Returns

void

5.2.2.2 cuDetectObject()

```
void cuDetectObject (
            const char * func,
            detectObject * objdetected )
```

Function to process the Detection task.

Function to detect the object and process the image. The output of this function is provided to the pathPlanner for further processing. It has three runnables.

Parameters

in	*func*	Function name
in,out	*objdetected*	Pointer to structure to detectObject input data

Returns

void

5.2.2.3 cuObjDetectSFM()

```
void cuObjDetectSFM (
            const char * func,
            sfmData * sfmInput )
```

Function to process the SFM task.

The functions process the data received from the input buffer and generates the input data for the planner task. It has three runnables.

Parameters

in	*func*	Function name
in,out	*sfmInput*	Pointer to structure to SFM input data

Returns

void

5.2.2.4 cuPlannerFetchCanBusData()

```
void cuPlannerFetchCanBusData (
            const char * func,
            int * hostCanPollingData )
```

Function to process the CAN bus polling task.

The functions process the data obtained from the global buffer. It provides this value as n input to Planner task.

Parameters

in	*func*	Function name
in,out	*hostCanPollingData*	Pointer to host can polling data.

Returns

void

5.2.2.5 cuPlannerInterpolatePath()

```
void cuPlannerInterpolatePath (
            const char * func,
            plannerData * data )
```

Function to process the Planner task.

The functions process the data received from the SFM, Detection, Can BUs Polling, grid data and lane boundary detection and process the data to generate the input to the DASM task.

Parameters

in	*func*	Function name
in,out	*data*	Pointer to structure to planner input data

Returns

void

5.2.2.6 cuProcessDASM()

```
void cuProcessDASM (
            const char * func,
            int * steer,
            int * speed )
```

Function to process the DASM task.

The functions process the data received from the planner task and generate the steer and speed output. The task consists of one runnable.

Parameters

func[in]	Function name
steer[inout]	Pointer to input steer from planner. Data is modified after processing.
speed[inout]	Pointer to input speed from planner. Data is modified after processing.

Returns

void

5.2.2.7 getCudaDeviceProperties()

```
void getCudaDeviceProperties (
            void  )
```

Function to get the CUDA device properties.

The functions get the CUDA device count and prints the device properties of each CUDA device.

Returns

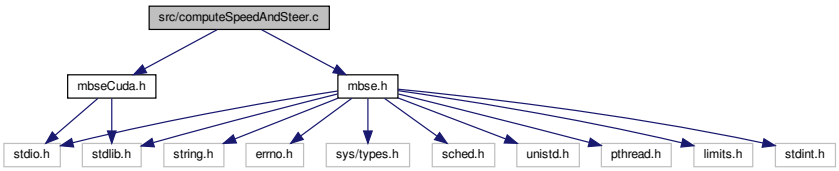
void

5.3 src/computeSpeedAndSteer.c File Reference

This file contains the basic implementation of the DASM and OS overhead tasks.

```
#include "mbse.h"
#include "mbseCuda.h"
```

Include dependency graph for computeSpeedAndSteer.c:



Functions

- void * computeSpeedAndSteer (void *args)

  This function implements the DASM task.
- void * computeOSOverhead (void *args)

  This task is used to add the OS overhead to the overall tasks based on the Amalthea task model.

5.3.1 Detailed Description

This file contains the basic implementation of the DASM and OS overhead tasks.

Author

Anand Prakash

Date

12 May 2020 This task computes and establishes the speed and steer that must be effectively employed from the information that is provided by the Path Planner task. It also implements a task which introduces the OS overhead that may occur during the execution.

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.html

5.3.2 Function Documentation

5.3.2.1 computeOSOverhead()

```
void* computeOSOverhead (
            void * args )
```

This task is used to add the OS overhead to the overall tasks based on the Amalthea task model.

This task adds an additional overhead to the overall application. It is used to simulate the overhead that occur in real scenario. It is executed on core number 3 on Jetson TX2 ARM A57 core with the thread priority of 99.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

void

5.3.2.2 computeSpeedAndSteer()

```
void* computeSpeedAndSteer (
            void * args )
```

This function implements the DASM task.

This task computes and establishes the speed and steer that must be effectively employed from the information that is provided by the Path Planner task. It is executed on core number 3 on Jetson TX2 ARM A57 core with the thread priority of 99.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

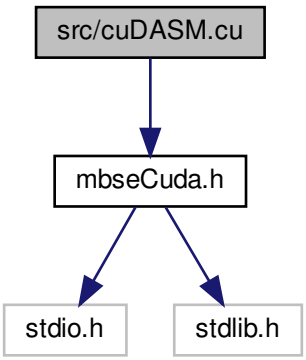
void

5.4 src/cuDASM.cu File Reference

This file contains the CUDA kernel implementation of the DASM and OS overhead tasks.

```
#include "mbseCuda.h"
```

Include dependency graph for cuDASM.cu:



Functions

- __global__ void computeDASM (int *devSpeed, int *devSteer, int step, int size)
  Kernel that executes on the CUDA device to compute the speed and steer for DASM task.
- void cuProcessDASM (const char *func, int *steer, int *speed)
  Function to process the DASM task.
- __global__ void osOverhead (int *A, int *B, int *C, int size)
  Kernel that executes on the CUDA device the OS overhead task.
- void addOSOverhead (const char *func)
  Function to process the OS overhead task.

5.4.1 Detailed Description

This file contains the CUDA kernel implementation of the DASM and OS overhead tasks.

Author

Anand Prakash

Date

12 May 2020 This file implements the runnables and used by the DASM and OS Overhead tasks.

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.html

5.4.2 Function Documentation

5.4.2.1 addOSOverhead()

```
void addOSOverhead (
            const char * func )
```

Function to process the OS overhead task.

The functions adds some OS overhead to simulate the real scenario.

Parameters

func[in]	Function name

Returns

void

5.4.2.2 cuProcessDASM()

```
void cuProcessDASM (
            const char * func,
            int * steer,
            int * speed )
```

Function to process the DASM task.

The functions process the data received from the planner task and generate the steer and speed output. The task consists of one runnable.

Parameters

func[in]	Function name
steer[inout]	Pointer to input steer from planner. Data is modified after processing.
speed[inout]	Pointer to input speed from planner. Data is modified after processing.

Returns

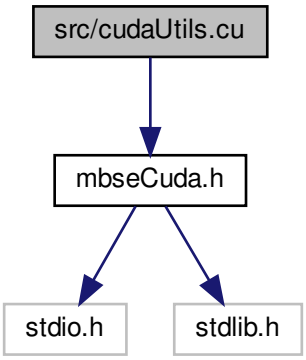
void

5.5 src/cudaUtils.cu File Reference

This file contains the CUDA utility functions.

```
#include "mbseCuda.h"
```

Include dependency graph for cudaUtils.cu:



Functions

- void getCudaDeviceProperties (void)

  Function to get the CUDA device properties.

5.5.1 Detailed Description

This file contains the CUDA utility functions.

Author

Anand Prakash

Date

01 May 2020 This file implements utility functions like getting the CUDA device properties. All the common CUDA specific implementation must be added in this file.

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.↩
html

5.5.2 Function Documentation

5.5.2.1 getCudaDeviceProperties()

```
void getCudaDeviceProperties (
            void  )
```

Function to get the CUDA device properties.

The functions get the CUDA device count and prints the device properties of each CUDA device.

Returns

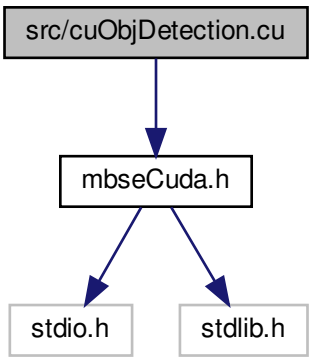
void

5.6 src/cuObjDetection.cu File Reference

This file contains the CUDA kernel implementation of Detection and Structure- from-motion tasks.

```
#include "mbseCuda.h"
```

Include dependency graph for cuObjDetection.cu:



Functions

- __global__ void processImage (int *hostBbox, int *devBbox, int *hostImage, int *devImage, int numElements)

  CUDA Kernel Device code.
- void cuDetectObject (const char *func, detectObject *objdetected)

  Function to process the Detection task.
- __global__ void processSFMData (int *image, int *matrix, int nbin, int step, int nthreads, int nblocks)

  Kernel that executes on the CUDA device to process the SFM data.
- void cuObjDetectSFM (const char *func, sfmData *sfmInput)

  Function to process the SFM task.

5.6.1 Detailed Description

This file contains the CUDA kernel implementation of Detection and Structure- from-motion tasks.

Author

Anand Prakash

Date

17 April 2020 This file implements runnables executed for Detection and Structure-from-Motion tasks.

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.html

5.6.2 Function Documentation

5.6.2.1 cuDetectObject()

```
void cuDetectObject (
            const char * func,
            detectObject * objdetected )
```

Function to process the Detection task.

Function to detect the object and process the image. The output of this function is provided to the pathPlanner for further processing. It has three runnables.

Parameters

in	*func*	Function name
in,out	*objdetected*	Pointer to structure to detectObject input data

Returns

void

5.6.2.2 cuObjDetectSFM()

```
void cuObjDetectSFM (
            const char * func,
            sfmData * sfmInput )
```

Function to process the SFM task.

The functions process the data received from the input buffer and generates the input data for the planner task. It has three runnables.

Parameters

in	*func*	Function name
in,out	*sfmInput*	Pointer to structure to SFM input data

Returns

void

5.6.2.3 processImage()

```
__global__ void processImage (
             int * hostBbox,
             int * devBbox,
             int * hostImage,
             int * devImage,
             int numElements )
```

CUDA Kernel Device code.

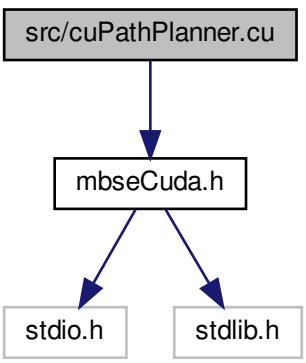
Runnable to Process the image to detect and classify the objects by creating the Boundary Box.

5.7 src/cuPathPlanner.cu File Reference

This file contains the CUDA kernel implementation of Planner and CAN Bus Polling tasks.

```
#include "mbseCuda.h"
```

Include dependency graph for cuPathPlanner.cu:



Functions

- __global__ void getCanBusData (int *canData, int size, int nthreads, int nblocks)

  Kernel that executes on the CUDA device to process the CAN Bus data.
- void cuPlannerFetchCanBusData (const char *func, int *hostCanPollingData)

  Function to process the CAN bus polling task.
- __global__ void pathPlan (int *devSpeed, int *devSteer, int size)

  Kernel that executes on the CUDA device to process the path planner task.
- void cuPlannerInterpolatePath (const char *func, plannerData *data)

  Function to process the Planner task.

5.7.1 Detailed Description

This file contains the CUDA kernel implementation of Planner and CAN Bus Polling tasks.

Author

Anand Prakash

Date

12 May 2020 This file implements runnables executed for Planner and CAN Bus Polling tasks.

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.html

5.7.2 Function Documentation

5.7.2.1 cuPlannerFetchCanBusData()

```
void cuPlannerFetchCanBusData (
            const char * func,
            int * hostCanPollingData )
```

Function to process the CAN bus polling task.

The functions process the data obtained from the global buffer. It provides this value as n input to Planner task.

Parameters

in	*func*	Function name
in,out	*hostCanPollingData*	Pointer to host can polling data.

Returns

void

5.7.2.2 cuPlannerInterpolatePath()

```
void cuPlannerInterpolatePath (
            const char * func,
            plannerData * data )
```

Function to process the Planner task.

The functions process the data received from the SFM, Detection, Can BUs Polling, grid data and lane boundary detection and process the data to generate the input to the DASM task.

Parameters

in	*func*	Function name
in,out	*data*	Pointer to structure to planner input data

Returns

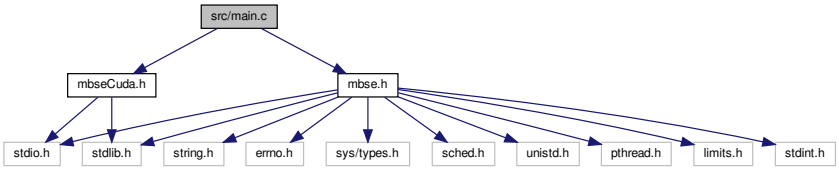
void

5.8 src/main.c File Reference

This file contains the entry point of the applications and consists of the initialization of memory, threads of the application.

```
#include "mbse.h"
#include "mbseCuda.h"
```

Include dependency graph for main.c:



Typedefs

- typedef void ∗(∗ **threadPool_t**) (void ∗)

Functions

- int **main** (int argc, char ∗argv[ ])

5.8.1 Detailed Description

This file contains the entry point of the applications and consists of the initialization of memory, threads of the application.

Author

Anand Prakash

Date

11 April 2020 This is the entry point of the application. The application can be executed only with root/superuser privileges and get the CUDA device properties along with initializing the threads and start their execution. The tasks are executed on three ARM A57 cores and uses RMS scheduling approach.

See also

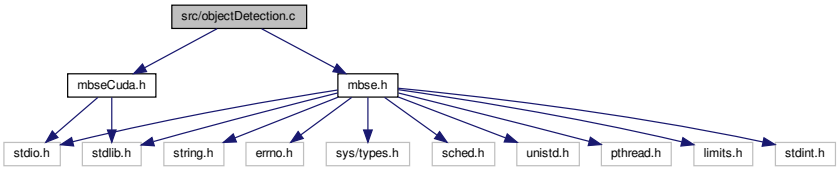
https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.↩
html

5.9 src/objectDetection.c File Reference

This file contains the basic implementation of the Detection and Structure from motion tasks.

```
#include "mbse.h"
#include "mbseCuda.h"
```

Include dependency graph for objectDetection.c:



Functions

- void ∗ objDetectGetObject (void ∗args)

  This task is used perform the detection functionality.
- void ∗ objDetectStructureFromMotion (void ∗args)

  This task is used perform the Structure-From-Motion functionality.

5.9.1 Detailed Description

This file contains the basic implementation of the Detection and Structure from motion tasks.

Author

Anand Prakash

Date

22 April 2020.

The detection task is responsible of detecting and classifying the objects in the road. It uses a machine learning approach. All the objects detected are visualized and the information produced is sent to the Planner task.

Structure-From-Motion task is a method for estimating 3-D structures (depth) from vehicle motion and sequences of 2-D images. This task returns a matrix of points representing the distance with respect the objects of the image.

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.html

5.9.2 Function Documentation

5.9.2.1 objDetectGetObject()

```
void* objDetectGetObject (
            void * args )
```

This task is used perform the detection functionality.

This task is responsible of detecting and classifying the objects in the road. All the objects detected are visualized and the information produced is sent to the Planner task. It is executed on core number 0 on Jetson TX2 ARM A57 core with the thread priority of 98.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

void

5.9.2.2 objDetectStructureFromMotion()

```
void* objDetectStructureFromMotion (
            void * args )
```

This task is used perform the Structure-From-Motion functionality.

Structure-From-Motion is a method for estimating 3-D structures (depth) from vehicle motion and sequences of 2-D images. This task returns a matrix of points representing the distance with respect the objects of the image. It is executed on core number 0 on Jetson TX2 ARM A57 core with the thread priority of 99.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

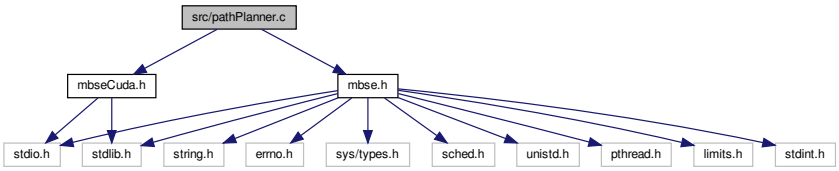
void

5.10 src/pathPlanner.c File Reference

This file contains the basic implementation of the Planner task and CAN Bus polling task.

```
#include "mbse.h"
#include "mbseCuda.h"
```

Include dependency graph for pathPlanner.c:



Functions

- void * pathPlannerCalculation (void *args)

  This task is used perform the Planner task functionality.
- void * pathPlannerCanBusPolling (void *args)

  This task is used perform the CAN Bus Polling task functionality.

5.10.1 Detailed Description

This file contains the basic implementation of the Planner task and CAN Bus polling task.

Author

Anand Prakash

Date

22 April 2020 The main purpose of Planner component is to define and follow a given trajectory. This trajectory is defined as a spline, that is, a line built through polynomial interpolation at times on the map that represents at each point the position and orientation that the car will have to follow. The spline can be enriched with additional information such as speed to hold, stop, priorities, etc. The planner sends the goal state of the vehicle (i.e., target steer and speed) to the DASM task that is in charge of writing the commands in the CAN line the effective steer and speed to apply.

The CAN Bus polling task snoops the key vehicle information (steer/wheel/break/acceleration status...) from the on-board CAN bus and sends it to the Planner task.

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.↩
html

5.10.2 Function Documentation

5.10.2.1 pathPlannerCalculation()

```
void* pathPlannerCalculation (
            void * args )
```

This task is used perform the Planner task functionality.

The main purpose of this component is to define and follow a given trajectory. This trajectory is defined as a spline, that is, a line built through polynomial interpolation at times on the map that represents at each point the position and orientation that the car will have to follow. The planner sends the goal state of the vehicle (i.e., target steer and speed) to the DASM task that is in charge of writing the commands in the CAN line the effective steer and speed to apply. It is executed on core number 5 on Jetson TX2 ARM A57 core with the thread priority of 98.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

void

5.10.2.2 pathPlannerCanBusPolling()

```
void* pathPlannerCanBusPolling (
            void * args )
```

This task is used perform the CAN Bus Polling task functionality.

This task snoops the key vehicle information (steer/wheel/break/acceleration status...) from the on-board CAN bus and sends it to the Localization, Planner and EKF tasks. It is executed on core number 5 on Jetson TX2 ARM A57 core with the thread priority of 99.

Parameters

in	*args*	Optional argument. Currently not in use.

Returns

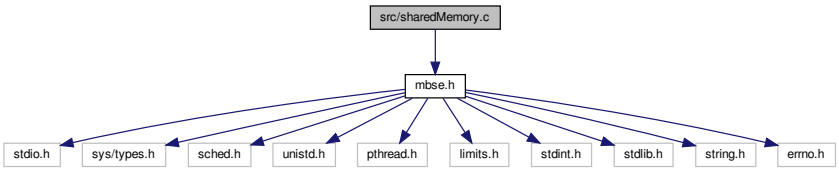
void

5.11 src/sharedMemory.c File Reference

This file declares and implement the read and write operation of all the shared memory.

```
#include "mbse.h"
```

Include dependency graph for sharedMemory.c:



Macros

- #define **DATA_IN_SFM_DETECTION_BUFFER** ((2 ∗ 1024 ∗ 1024) / 4)
- #define **DATA_OUT_SFM_BUFFER** ((24 ∗ 1024) / 4)
- #define **DATA_OUT_DETECTION_BUFFER** ((750 ∗ 1024) / 4)
- #define **DATA_OUT_PLANNER_BUFFER** ((1 ∗ 1024) / 4)
- #define **DATA_GRID_BUFFER** ((1024 ∗ 512) / 4)
- #define **DATA_LANE_BOUNDARY_BUFFER** (256 / 4)

Functions

- int shmemReadSFMDetectionDataInLabel (unsigned int index)

  Function to read to the Detection and SFM Input data buffer.
- int shmemReadSFMDataOutLabel (unsigned int index)

  Function to read to the SFM Output data buffer.
- int shmemReadDetectionDataOutLabel (unsigned int index)

  Function to read to the Detection Output data buffer.
- int shmemReadPlannerBufferLabel (unsigned int index)

  Function to read to the Planner Output data buffer.

- int shmemReadGridDataBufferLabel (unsigned int index)

  Function to read to the data grid buffer.
- int shmemReadLaneBoundaryBufferLabel (unsigned int index)

  Function to read to the lane boundary data buffer.
- void shmemWriteDetectionDataOutLabel (int offset, int size, void *data)

  Function to write to the Detection data buffer.
- void shmemWriteSFMDataOutLabel (int offset, int size, void *data)

  Function to write to the SFM data buffer.
- void shmemWritePlannerDataOutLabel (int offset, int size, int data)

  Function to write to the Planner data buffer.
- void shmemWriteSFMDetectionDataInLabel (int offset, int size, int data)

  Function to write to the SFM and detection Input buffer.
- void shmemWriteGridDataBufferLabel (int offset, int size, int data)

  Function to write to the data grid buffer.
- void shmemWriteLaneBoundaryBufferLabel (int offset, int size, int data)

  Function to write to the lane boundary buffer.

5.11.1 Detailed Description

This file declares and implement the read and write operation of all the shared memory.

Author

Anand Prakash

Date

05 May 2020 It consists a common input buffer for Detection and Structure for Motion task which is 2MB in size. Output data buffer for Structure for motion and Detection task with size 24KB and 750KB respectively. The planner, data grid and lane boundary buffers are 1KB, 512KB and 256 bytes in size respectively.

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.html

5.11.2 Function Documentation

5.11.2.1 shmemReadDetectionDataOutLabel()

```
int shmemReadDetectionDataOutLabel (
            unsigned int index )
```

Function to read to the Detection Output data buffer.

This function read the Detection output data buffer. This buffer is used as an input to Planner task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.11.2.2 shmemReadGridDataBufferLabel()

```
int shmemReadGridDataBufferLabel (
            unsigned int index )
```

Function to read to the data grid buffer.

This function read the data grid buffer. This buffer is used as an input to Planner task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.11.2.3 shmemReadLaneBoundaryBufferLabel()

```
int shmemReadLaneBoundaryBufferLabel (
            unsigned int index )
```

Function to read to the lane boundary data buffer.

This function read the lane boundary data buffer. This buffer is used as an input to Planner task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.11.2.4 shmemReadPlannerBufferLabel()

```
int shmemReadPlannerBufferLabel (
            unsigned int index )
```

Function to read to the Planner Output data buffer.

This function read the Planner output data buffer. This buffer is used as an input to DASM task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.11.2.5 shmemReadSFMDataOutLabel()

```
int shmemReadSFMDataOutLabel (
            unsigned int index )
```

Function to read to the SFM Output data buffer.

This function read the SFM output data buffer. This buffer is used as an input to Planner task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.11.2.6 shmemReadSFMDetectionDataInLabel()

```
int shmemReadSFMDetectionDataInLabel (
            unsigned int index )
```

Function to read to the Detection and SFM Input data buffer.

This function read to the Detection and SFM Input data buffer. This buffer is used as an input to SFM and Detection task.

Parameters

in	*index*	Index at which the data is to be written.

Returns

The value at the requested index, -1 in case the index is out of bounds.

5.11.2.7 shmemWriteDetectionDataOutLabel()

```
void shmemWriteDetectionDataOutLabel (
            int offset,
            int size,
            void * data )
```

Function to write to the Detection data buffer.

This function writes to the Detection data buffer. This buffer is used as an input to Planner task.

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.11.2.8 shmemWriteGridDataBufferLabel()

```
void shmemWriteGridDataBufferLabel (
            int offset,
            int size,
            int data )
```

Function to write to the data grid buffer.

This function writes to the data grid buffer. This buffer is used as an input to Planner task

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.11.2.9 shmemWriteLaneBoundaryBufferLabel()

```
void shmemWriteLaneBoundaryBufferLabel (
            int offset,
            int size,
            int data )
```

Function to write to the lane boundary buffer.

This function writes to the lane boundary buffer. This buffer is used as an input to Planner task

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.11.2.10 shmemWritePlannerDataOutLabel()

```
void shmemWritePlannerDataOutLabel (
            int offset,
            int size,
            int data )
```

Function to write to the Planner data buffer.

This function writes to the Planner data buffer. This buffer is used as an input to DASM task.

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.11.2.11 shmemWriteSFMDataOutLabel()

```
void shmemWriteSFMDataOutLabel (
            int offset,
            int size,
            void * data )
```

Function to write to the SFM data buffer.

This function writes to the SFM data buffer. This buffer is used as an input to Planner task.

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

void

5.11.2.12 shmemWriteSFMDetectionDataInLabel()

```
void shmemWriteSFMDetectionDataInLabel (
            int offset,
            int size,
            int data )
```

Function to write to the SFM and detection Input buffer.

This function writes to the SFM and detection Input buffer. This buffer is used as an input to SFM and detection task.

Parameters

in	*offset*	Offset of the buffer.
in	*size*	Length of data.
in	*data*	Actual data that needs to be copied.

Returns

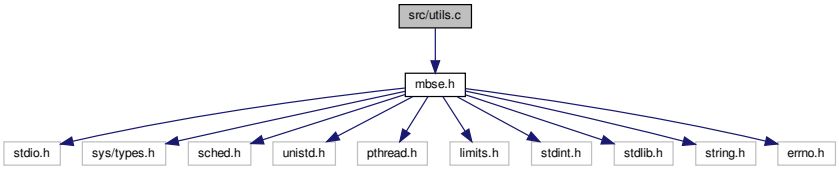
void

5.12 src/utils.c File Reference

This file declares and implement the utility functions used in the application.

```
#include "mbse.h"
```

Include dependency graph for utils.c:



Functions

- void utilSetThreadPriority (pthread_t threadId, int customPrio)

  Function to set the thread priority.
- void utilAddDelay (uint32_t ms, struct timespec *deadline)

  Function to add delay to the task.
- void error (int at)

  Function to print error message thrown from a particular point in application code.

5.12.1 Detailed Description

This file declares and implement the utility functions used in the application.

Author

Anand Prakash

Date

22 April 2020 It consists of functions setting the thread priority, adding delay to the tasks and printing errors. All the C utility functions must be added to this file.

See also

https://www.ecrts.org/archives/fileadmin/WebsitesArchiv/ecrts2019/waters/index.html

5.12.2 Function Documentation

5.12.2.1 error()

```
void error (
            int at )
```

Function to print error message thrown from a particular point in application code.

This function prints the error code from which the error was thrown.

Parameters

in	*at*	Error code.

Returns

void

5.12.2.2 utilAddDelay()

```
void utilAddDelay (
            uint32_t ms,
            struct timespec * deadline )
```

Function to add delay to the task.

This function adds delay to the task by sleeping for the time provided in the argument in terms of millisecond.

Parameters

in	*ms*	Time in millisecond for which the thread needs to sleep.
in,out	*deadline*	Pointer to the timespec for the next deadline

Returns

void

5.12.2.3 utilSetThreadPriority()

```
void utilSetThreadPriority (
            pthread_t threadId,
            int customPrio )
```

Function to set the thread priority.

This task sets the thread priority based on the threadId and the customPrio provided The priority of the thread is set to max priority minus the custom priority

Parameters

in	*threadId*	Thread ID whose priority needs to be set.
in	*customPrio*	Priority offset of the thread from the maximum priority

Returns

void

Index