Singapore Institute of Technology

BEng (Hons) Telematics (Intelligent Transportation Systems Engineering)

TLM2007 – Operating Systems and Automotive OS Project

Team 8		
S/No	Name	Matrix Number
1	Ahmad Amir Faris Bin Mazlan	1902020
2	Alex Goei Jing En	1902005
3	Choo Xian Zhi	1902026
4	Haziq Isyraq Shah Bin Bismillah Shah	1901990
5	Pham Do Minh Quang	1902021
6	Wilson Lee Kian Yi	1902022

Table of Content

Contents

Introduction

In this project, we are tasked with the development of an automotive odometer. A prototype of the system is to be built based on Erika OSEK/VDX Real-time Operating System, running on a SimulIDE Arduino Simulator.

The main concept of OSEK/VDX system for automotive OSes, is to provide real-time behavior with minimal resources and functions for the automotive vehicle, to ensure the safety of the user and ensure that all tasks/requirements are fulfilled. With this standardization, a specification for a standard software architecture is created, and applicable to the various electronic control units(ECUs) within the car.

Following the given specifications, the odometer will require the listed functions:

- Speed information of 0 to 250 kph, from analog voltage of 0 to 5 volts
- Total distance travelled in kilometers
- Total time of travel in hours and minutes
- Continuous display of the above information on LCD display
- Reset button for clearing mileage information back to zero

We also have intentions of implementing additional functions, such as LCD screen animations, and mode switching for distance unit selection.

To achieve these functionalities, a system design must be created first. This design will consist of the OSEK Implementation Language Configuration, and also involves the various tasks, alarms and interrupt service routines. This will be elaborated in the next section.

System Design

To appropriately design an OSEK/VDX system, we will need to break down the required functionalities first. This can also be separated across hardware and software segments.

Looking at the first function, speed information from analog input, we will require hardware such as a voltage source and analog input pin. Regarding software, the appropriate analog to digital conversion(ADC) library is needed, and further conversion from the ADC value into the required speed value is needed.

Next, regarding the distance and time mileages, these functions only require software as follows. To clock the distance covered, we will need to look at the speed value, and take appropriate measurements at a consistent interval. The time mileage will also require to be recorded at a consistent interval, with counters for seconds, minutes and hours. Thus, the use of a system counter and alarm is suitable, to create the required interval for mileage clocking.

The values discussed so far, will need to be displayed continually on a display for the user. To do this, a third-party LCD display will be used, and the relevant drivers and functions will need to be implemented.

As for the reset of mileage values, hardware-wise we require a button for user input, and the above mentioned distance and time values can be reset to zero upon button press. This can be achieved using an interrupt service routine(ISR) upon button press.

Lastly, for the additional features, the LCD animations can be used to indicate system start up. This animation thus needs to be implemented at the system initialisation stage, such as the startup hook for example. We also wish to implement a mode switch, to change the distance units from kilometers to miles. This can be done using a hardware switch, as well as application modes, a part of the OSEK system.

Based on the above discussion, we are able to define our hardware and software design accordingly.

Hardware

As the prototype is to be built in SimulIDE, an Arduino simulator, the components will be from the available tools within SimulIDE. The LCD display will be a Hitachi LCD controller HD44780, together with the relevant software drivers. Figure 1 below shows a block diagram of our required circuit.

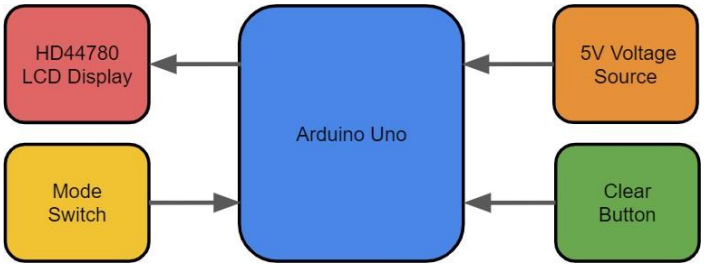

Figure 1: Circuit Block Diagram

Based on the outline from the block diagram, we built the prototype circuit in SimulIDE, as pictured in figure 2 below. An additional reset switch is added to aid in troubleshooting.

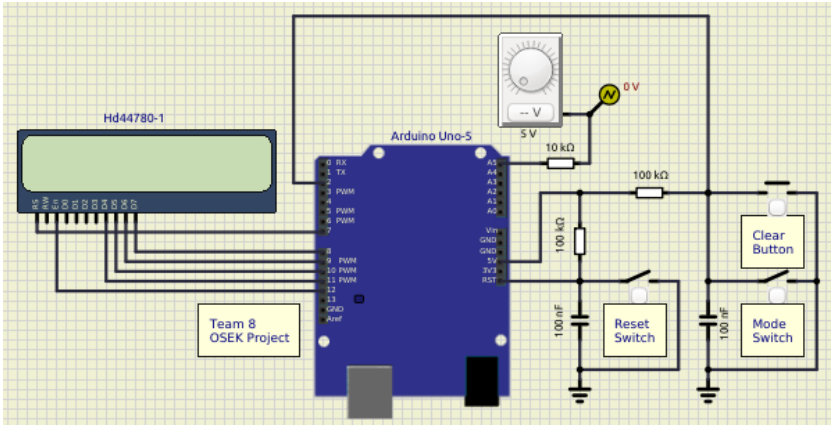

Figure 2: SimulIDE Prototype Circuit

Software

To meet OSEK/VDX system specifications, the software functions must be implemented through tasks. Looking at our required functions, we can easily split them into two tasks, one for data logging, and another for the output of data onto the LCD display. The tasks will be named MileageTask and DisplayTask respectively.

Due to the data logging nature of the MileageTask, it will be considered a critical task, and will be configured to be non-preemptive and higher priority than DisplayTask. As DisplayTask does

not alter the data collected in this system, it is less critical, and is set to be preemptable, so that it does not interfere with data integrity.

The tasks will need to occur on a set interval. This set interval can be created by utilising an alarm generated from a system counter. The alarm interval duration is chosen to be 1 second, as this can be used as a normalised unit of time, ideal for mileage clocking purposes. The alarm can then call the MileageTask at this interval, ensuring consistent clocking of required data.

These two tasks will need to be synchronised to ensure the system is stable and consistent, thus we will be implementing events to trigger the appropriate task. This requires the system to be in extended mode, enabling extended tasks. We will be using Kernel Conformance Class ECC1, which provides extended functionality without multiple task activations. The tasks will also require a private stack each, which we have assigned 128 bytes to each.

The events to be implemented are ClearMileage, ClockMileage, and UpdateLCD. ClearMileage will be used for triggering a reset of the mileage values, through MileageTask. ClockMileage will trigger MileageTask to update the speed, distance and time values, while UpdateLCD will trigger DisplayTask to print the latest data to the LCD display. The task will automatically start at system boot, and enter a waiting state, only performing its functions when triggered by its event.

With events, the system's alarm will set the ClockMileage event and in turn trigger MileageTask. At the end of MileageTask, an UpdateLCD event will be set, to output the new data via DisplayTask. The timing diagram in figure 3 below illustrates this process.

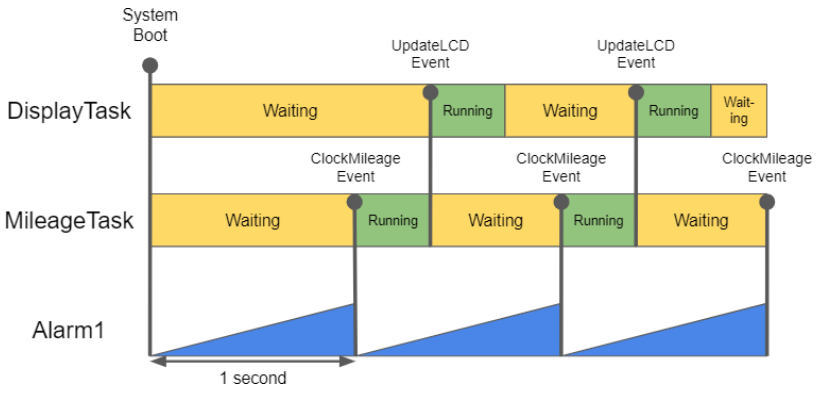

Figure 3: System Alarm/Task Timing Diagram

We also require an interrupt service routine(ISR), as we have implemented a button for the reset of mileage values. The ISR will be named ButtonISR, and is of category type 2 and hardware vector “INT0”.

For our additional features, the function for the LCD display startup animation will be placed alongside the system initialisation codes, inside of the startup hook. As for the distance unit switching, we will make use of application modes to accomplish this. The modes we have defined are Kilometers and Miles, which represent the distance units as per their own names.

Using the switch in the hardware design, the user is able to select the distance units preferred, which by default will be kilometers if the switch is not pressed. This is checked by the system during initialisation, inside the startup hook. After determining the application mode, the both MileageTask and DisplayTask are able to perform differently based on the active application mode. A declaration is required inside the configurations of both tasks, defining the default mode, in this case Kilometers.

Function Implementations

Having covered the overall system design and configuration, we will now delve deeper into the functions and implementation. We shall be following the sequence of events as per the design defined earlier, starting with the startup hook.

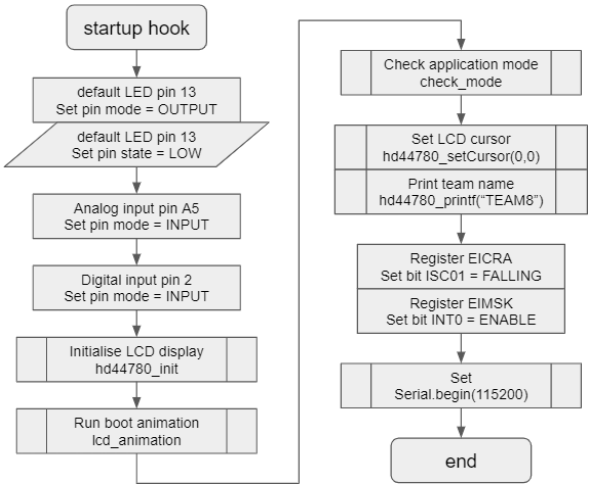

Figure 4: Startup Hook Flowchart

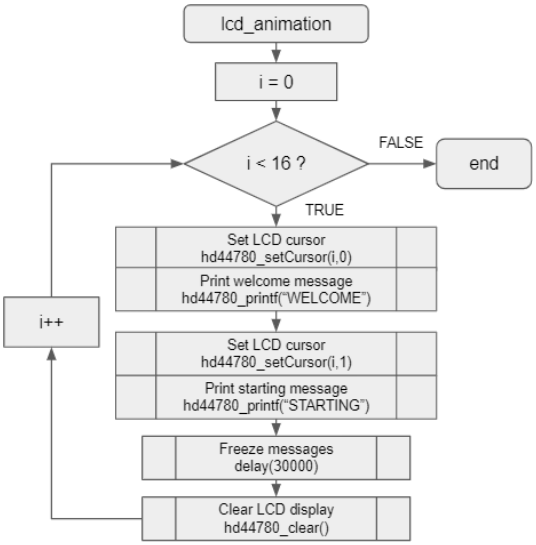

Figure 5: LCD Animation Flowchart

Looking at figure 4 above, we see the flowchart of the startup hook. Basic Arduino pin setup is done first, and this includes the analog input pin for the voltage source input, as well as the input for the button interrupt. Next, the LCD display is initialised, and the boot animation is played. The function of the animation is detailed in the flowchart in figure 5 above, and is utilising a for loop to shift the messages through the LCD display columns. Next, the application mode is checked using the check_mode function, where in this function, the switch state is read, and the mode is set accordingly, while the user sees a LCD message indicating the unit selected.

Next, we look at MileageTask and its many functions, which is detailed in the flowcharts in figure 6 and 7 below. Whenever there is a ClockMileage event, data logging occurs. This involves the speed, distance and time values.

For the speed value, we make use of the ADC value of the voltage source input, via formula of ADC value x (251/1024), based on the speed range of 0 to 250 and ADC range of 0 to 1023. The application mode is also checked, to convert to the correct speed unit accordingly. For the distance covered, the current speed is converted into thousand-th per second of the original value, to be logged under the decimal value, which can be considered meters for kilometer unit and likewise for miles. For the time clocking, we utilise the 1 second duration of the alarm to increment a seconds variable. In the codes, we obey the nested-if technique, and once seconds is 60, it will then be converted into a minute and likewise for minutes to hours.

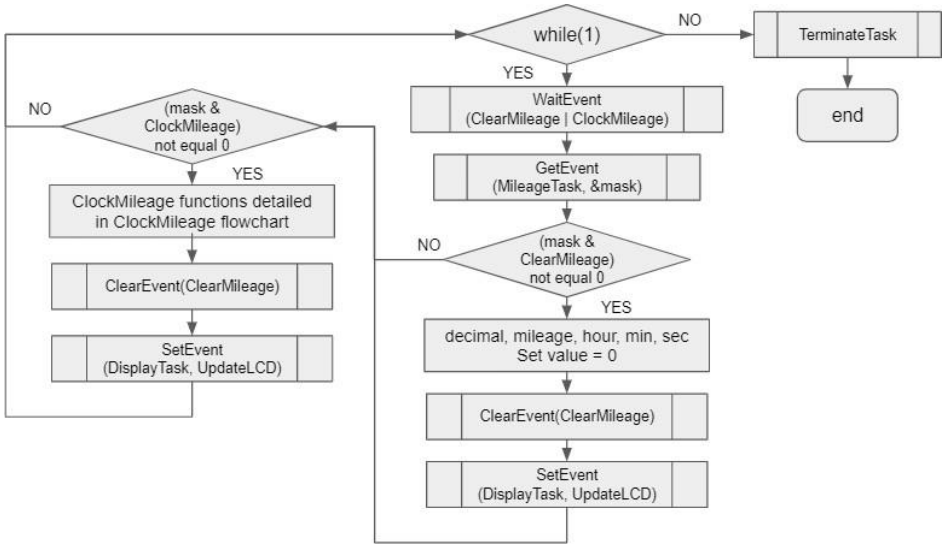

Figure 6: MileageTask Flowchart

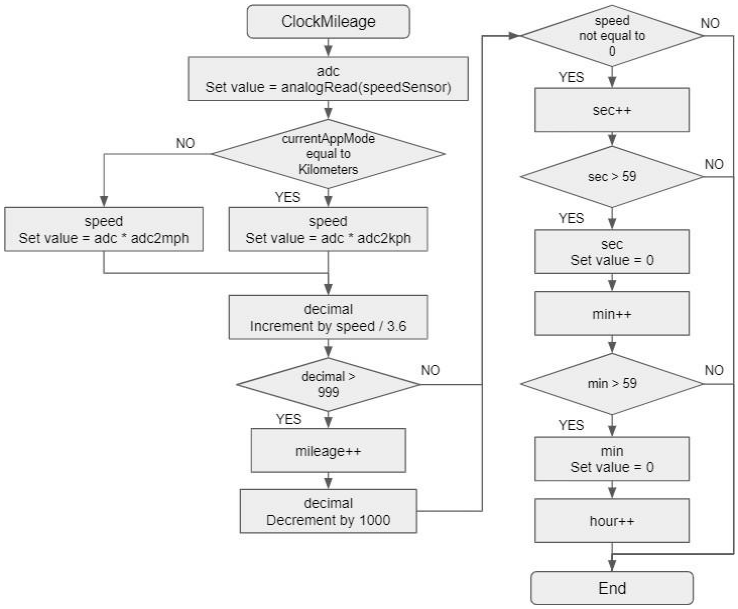

Figure 7: ClockMileage Event Flowchart

Outside of the alarm interval based events, MileageTask also waits for ClearMileage, which is set by our button interrupt. When the user presses the button, the system will enter the interrupt service routine, and the function ButtonISR will set the event ClearMileage. This will trigger MileageTask, and the task will carry out the functions to reset the distance and time value to zero.

After the MileageTask is done with its event, the DisplayTask is called through the UpdateLCD event. In this task, the application mode also checked, and the corresponding display output is printed for kilometers or miles. The flowchart for DisplayTask can be seen below in Figure 8.

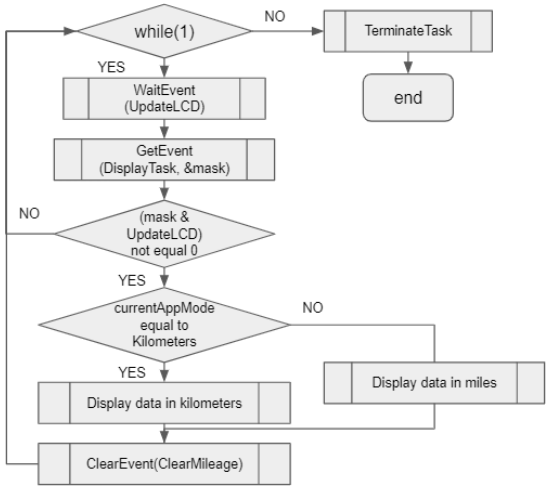

Figure 8: DisplayTask Flowchart

Limitations & Improvements

In this project, it was set up for the tasks to be operated on an event-basis, so if the function or the event is not being called, the event would just be in idle status and may result in a waste of resources for the memory. With the event-based running in the system, there will be a clear sequence of events running in the back-end which means if the code supported is not written or matches via the condition that was set, the whole flow of events would run in the wrong order which would make the system run incorrectly. There could be any implication of a killer performance method such as a break in-between the tasks, to ensure the smoothness of events is running correctly, but in this project, we are able to execute in the correct flow that it was designed for. This could be an improvement for future works to ensure the burst time or the execution time meets the real-world scenario for the car.

Appendix

Table 1: OIL Configuration

OIL Elements	Attributes	Options	Config
OS	STATUS STARTUPHOOK	STANDARD or EXTENDED TRUE or FALSE	EXTENDED TRUE
Kernel Conformance Class	KERNEL_TYPE	BCC1, BCC2,ECC1, ECC2	ECC1
MileageTask	PRIORITY SCHEDULE AUTOSTART APPMODE STACK SIZE ACTIVATION EVENT EVENT	Number FULL or NON TRUE or FALSE User defined strings SHARED or PRIVATE Bytes No. of pending activation User defined strings User defined strings	2 NON TRUE Kilometers PRIVATE 128 1 ClearMileage ClockMileage
DisplayTask	PRIORITY SCHEDULE AUTOSTART APPMODE STACK SIZE ACTIVATION EVENT	Number FULL or NON TRUE or FALSE User defined strings SHARED or PRIVATE Bytes No. of pending activation User defined strings	1 FULL TRUE Kilometers PRIVATE 128 1 UpdateLCD
EVENT	MASK	User defined strings User declared event bits	ClearMileage AUTO
EVENT	MASK	User defined strings User declared event bits	ClockMileage AUTO
EVENT	MASK	User defined strings User declared event bits	UpdateLCD AUTO
APPMODE		User defined strings	Kilometers Miles
COUNTER	MINICYCLE MAXALLOWEDVALUE TICKSPERBASE TYPE SYSTEM TIMER PRIORITY DEVICE SECONDSPERTICK	User defined string Number at minimum Number Maximum, wrap at 0 Number, hardware prescalar HARDWARE or SOFTWARE TRUE or FALSE IRQ priority of HW timer String of hardware name Period in seconds of timer	SystemTimer 1 65535 1 HARDWARE TRUE 1 TIMER1_COMPA 0.001
ALARM	COUNTER ACTION TASK EVENT AUTOSTART ALARMTIME CYCLETIME	User defined string Name of linked counter ACTIVATETASK, SETEVENT, ALARMCALLBACK User defined task name User defined event name TRUE or FALSE Time to set alarm Time to cycle back	Alarm1 SystemTimer SETEVENT MileageTask ClockMileage TRUE 1000 1000
ISR	CATEGORY SOURCE PRIORITY	User defined string Type 1 or 2 String of hardware VECTOR name Interrupt Priority Number	ButtonISR 2 “INTO” <N.A. for Arduino>