Automotive Embedded System

There are 2 primary parts of an Automotive System:

- Electronic Control Unit (ECU)
    - Hardware Mother Board (enclosed in car engine bay/ clusters/ chassis/ wheels/ airbags….

- Microcontroller Unit (MCU)
    - Integrated Chip (IC)

MCU internal (on-chip) modules:

- Processor Unit (CPU)
- Clock Generation Unit (CGU)
- Memory Unit
    - ROM
    - FLASH
    - SRAM
    - EEPROM
- Digital Input Output ports (GPIO)
- Timer/ Counters (GPT)
- Analog to Digital Converter (ADC)
- Digital to Analog Converter (DAC)
- Serial Input Output Interfaces
    - Inter IC (I^2C)
    - Serial Peripheral Interface (SPI)
- Serial Communication
    - Universal Asynchronous Receiver Transmitter (UART)
    - Controller Area Network (CAN)

ECU internal (on-board) modules:

- Power Supply Circuitry
- MCU chip mounted
- On-board Oscillator
- On-board Digital IO (DIO) Modules
- On-board Analog Modules
- On-board PWM Modules
- On-board Serial IO Modules
- On-board Serial Communication
    - PC connectivity (UART-USB)
    - CAN Bus

Unit Testing Embedded C Modules

Modules under Test

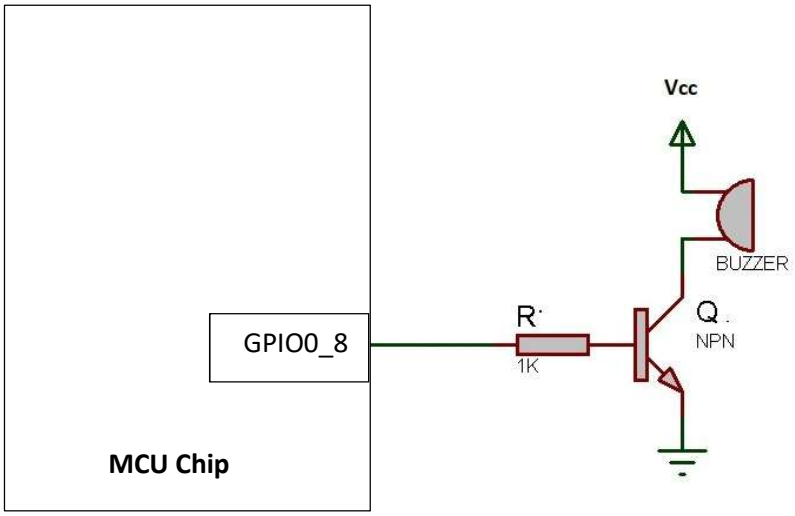
- Digital IO Modules
  - rgb
  - ukeys
  - buz
- Analog Module
  - analog
- Display Module
  - lcd
- Serial IO Modules
  - eeprom
  - dac

ECU on-board module	Source files	Dependent MCU on-chip module	Source Files	EUT Mock
RGB LED	rgb.c rgb.h	GPIO	gpio.c gpio.h	mock_gpio.h
BUZZER	buz.c buz.h	GPIO	gpio.c gpio.h	mock_gpio.h
USER KEYS	ukeys.c ukeys.h	GPIO	gpio.c gpio.h	mock_gpio.h
ANALOG	analog.c analog.h	ADC	adc.c adc.h	mock_adc.h
DAC	dac.c dac.h	SPI GPIO	spi.c spi.h gpio.c gpio.h	mock_gpio.h mock_spi.h
LCD	lcd.c lcd.h	GPIO	gpio.c gpio.h	mock_gpio.h
EEPROM	eeprom.c eeprom.h	I2C	i2c.c i2c.h	mock_i2c.h

One of the major challenges involved in unit testing of embedded software is that the code interacts with the hardware peripherals. In most cases, hardware cannot be accessed during unit tests. Keeping hardware interaction as thin as possible helps in testing most of the code by dividing it into small pieces. These pieces can then be independently tested without hardware interaction. mocks are used to simulate external dependencies of the code under test. This allows the code to be tested in isolation from other parts of the system. CMock is a tool to generate mock functions from C source header files. It generates mocks for each function and puts these into new mock files that are generated at run-time.

ECU on-board module: Buzzer

- MCU IO Pin used to interface Buzzer: GPIO0, pin#8
- IO direction: Output
- Circuit design: Active HIGH
    - logic 1 (HIGH) turns the buzzer to ON
    - logic 0 (LOW) turns the buzzer to OFF



R acts as Bias Resistor
Transistor Q acts as a switch

Embedded C module under test: buz.c/ .h
Function Name: BuzSet
Input parameters: 3 options

- 1 (BUZ_OFF)
- 2 (BUZ_ON)
- 3 (BUZ_BEEP)

return value: None

MCU on-chip module header file to mock: gpio.h
Function to mock: GPIO0SetPin
Input Parameters: pin number, set option

- pin num: 8
- set option: LOW/ HIGH/ TOGGLE

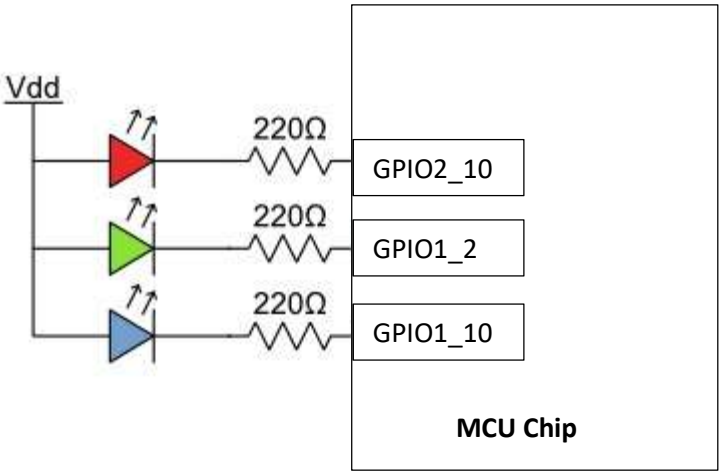
return value: None

Test cases to be developed: test_buz.c
Minimum test cases needed: 3

void test_buz_case_On(void)
void test_buz_case_Off(void)
void test_buz_case_Beep(void)

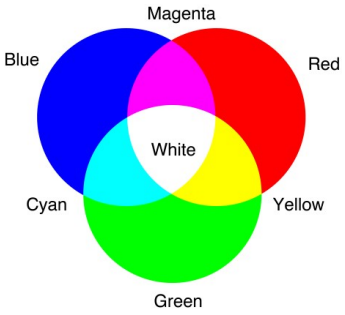
ECU on-board module: RGB LED(s)

- MCU IO Pins used to interface RGB LEDs
  - RED LED: GPIO2, pin#10
  - GREEN LED: GPIO1, pin#2
  - BLUE LED: GPIO1, pin#10
- IO direction: Output
- Circuit design: Active LOW Common Anode LEDs
  - logic 0 (LOW) turns the LED to ON
    - Current flows through LED
  - logic 1 (HIGH) turns the LED to OFF
    - No current flows through LED



Note: Resistor 220 ohm acts as current limiting resistance.

RGB LED Color Output



ECU on-board module under unit test: rgb.c/ .h
Function to test: RGBSetColor
Input Parameters: option (0 to 10)
Return value: None

MCU on-chip module to mock: gpio.h
Functions to mock: GPIO1SetPin, GPIO2SetPin
Input parameters: pin number and option
- pin number: 10 and 2

Return value: None

Minimum number of cases: 11 in test_rgb.c

RGB Control Logic

S No	GPIO2_10	GPIO1_2	GPIO1_10	RGB Output
0	HIGH	HIGH	HIGH	NONE
1	LOW	HIGH	HIGH	RED
2	HIGH	LOW	HIGH	GREEN
3	HIGH	HIGH	LOW	BLUE
4	LOW	LOW	HIGH	YELLOW
5	LOW	HIGH	LOW	MAGENTA
6	HIGH	LOW	LOW	CYAN
7	LOW	LOW	LOW	WHITE
8	TOGGLE	HIGH	HIGH	RED Blink
9	HIGH	TOGGLE	HIGH	GREEN Blink
10	HIGH	HIGH	TOGGLE	BLUE Blink

ECU on-board User Keys

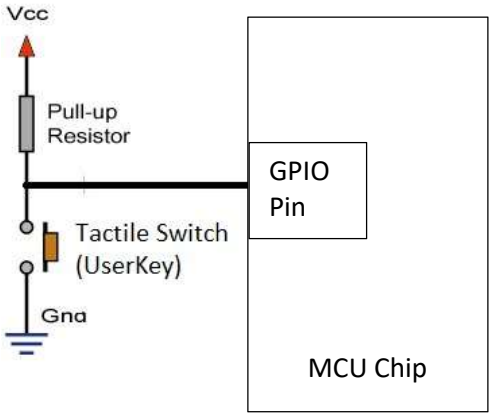
There are 5 user keys (tactile switches) on-board

- KEY_UP interfaced to GPIO0, pin#3
- KEY_DOWN interfaced to GPIO2, pin#6
- KEY_LEFT interfaced to GPIO2, pin#7
- KEY_RIGHT interfaced to GPIO1, pin#4
- KEY_CENTER interfaced to GPIO2, pin#8

IO Direction: Input

Circuit Design: Active LOW logic

- When the user key is un-touched, GPIO pin is at HIGH state
- When the user key is touched, GPIO pin is at LOW state



Embedded C module to unit test: ukey (ukeys.c/ ukey.h)

Functions under test:

```
int UserKeyCGetState(void);    // returns TOUCH or RELEASE
int UserKeyLGetState(void);    // returns TOUCH or RELEASE
int UserKeyRGetState(void);   // returns TOUCH or RELEASE
int UserKeyUGetState(void);   // returns TOUCH or RELEASE
int UserKeyDGetState(void);   // returns TOUCH or RELEASE

void UserKeyCGetClicks(int *count);  // returns the click count
void UserKeyLGetClicks(int *count);  // returns the click count
void UserKeyRGetClicks(int *count);  // returns the click count
void UserKeyUGetClicks(int *count);  // returns the click count
void UserKeyDGetClicks(int *count);  // returns the click count
```

MCU on-chip dependency to mock: gpio (mock_gpio.h)

Functions to mock:

```
int GPIO0GetPinState(int pin); // return value: LOW/HIGH
int GPIO1GetPinState(int pin); // return value: LOW/HIGH
int GPIO2GetPinState(int pin); // return value: LOW/HIGH
```

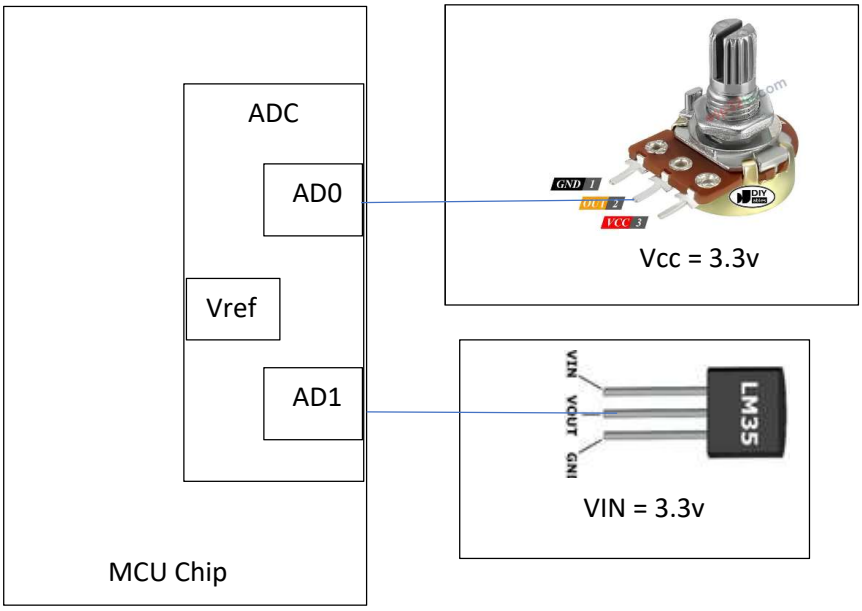
For each user key, there are 2 requirements

- Track the current state (TOUCH or RELEASE)
- Track the click count
    - User Key click means TOUCH and RELEASE

ECU on-board Analog sensors

There are 2 analog inputs on-board

- 0 to 3.3v POT interfaced to MCU on-chip ADC input pin, AD0
- LM35 temperature sensor interfaced to MCU on-chip ADC input pin, AD1



ADC Specification

- Vref = 3.3v (Max voltage)
- 10-bit Resolution
- Analog Input Range: 0 to 3.3v
- Digital Output Range: 0 to 2^{10} - 1 = 0 to 1023
- 1 count of ADC represents 3.22mv

Note: ADC converts the analog quantity to digital count.

LM35 Specification

- Converts the physical temperature to electric voltage
- For every 1 degree raise in temperature, it generates 10mv
  - ADC generates 10mv/ 3.22mv = 3 counts
- Temperature Measurement
  - T = (ADC Count)/3

Embedded C module under test: analog (analog.c/ .h)

Functions under test:

float DtrmVoltage(void); // returns the voltage in the range: 0 to 3.3v

float DtrmTemperature(void); // returns the temperature in the range: 0 to 50 Celsius

Dependent MCU on-chip module to mock: adc (adc.c/ .h)

Function(s) to mock

unsigned short ADCGetA2D(int input);

- Input parameter: ADC channel number (0 and 1)
- Return value: digital count: 0 to 1023

Analog Input: 0 to 3.3v POT

Test Samples

Sample No	ADC Output	Measured Voltage (v)
1	10	10 X 0.00322 = 0.0322
2	100	100 X 0.00322 = 0.322
3	250	250 X 0.00322 = 0.805
4	500	500 X 0.00322 = 1.61
5	800	800 X 0.00322 = 2.576
6	1000	1000 X 0.00322 = 3.22
7	1023	1023 x 0.00322 = 3.3

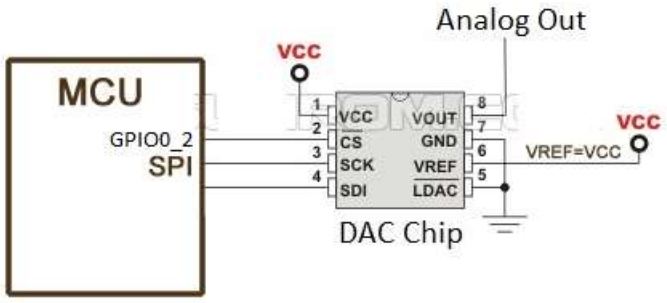
Analog Input: LM35 temperature sensor

Test Samples

Sample No	ADC Output	Measured Room Temperature (C)
1	10	10/3 = 3.33
2	30	30/3 = 10
3	50	50/3 = 16.66
4	80	80/3 = 26.66
5	100	100/3 = 33.33
6	120	120/3 = 40
7	130	130/3 = 43.33
8	150	150/3 = 50

ECU on-board module: Serial DAC

There is a serial DAC chip interfaced through SPI bus on the ECU board.



DAC converts the n-bit digital count to analog voltage.

DAC Specification

- Vref = 3.3v (Max voltage)
- 10-bit Resolution
- Digital Input Range: 0 to 2^{10} -1 = 0 to 1023
- Analog Output Range: 0 to 3.3v
- 1 count of DAC input generates 3.22mv of voltage

For a serial (SPI) DAC, input to be given through serial data transfer.

DAC output to be observed at VOUT pin.

GPIO0, pin#2 is used to enable (chip-select) and disable the DAC chip for serial data transfer.

How to give input to serial (SPI) DAC chip

- Enable (chip-select) the DAC chip for serial data transfer from MCU
  - GPIO0, pin#2 to be set LOW
- Transfer 16-bit data to DAC internal register through SPI bus
  - 16-bit has multiple fields
    - 4-bit Configuration Field
      - Bit#15:0
      - Bit#14:1
      - Bit#13:1
      - Bit#12:1
    - 10-bit DAC data field
      - Bit#11 to Bit#2

    Note: Bit#1 and Bit#0 to be ignored.
- Disable the DAC chip for serial data transfer
  - GPIO0, pin#2 to be set HIGH

ECU on-board module under test: dac (dac.c/ dac.h)

Function(s) to test: void DACWrite(unsigned short val);

Input Parameters: 16-bit data

Return Value: None

Dependent MCU on-chip module: spi (spi.c/ spi.h) and gpio (gpio.c/ .h)

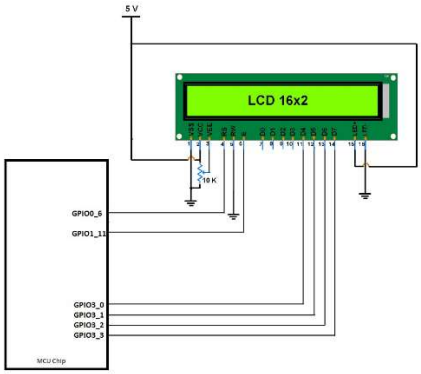
Functions to mock in gpio: GPIO0SetPin

Functions to mock in spi: SPISetTransfer

ECU on-board module: LCD

There is a 16X2 alphanumeric display on ECU board interfaced in 4-bit configuration.
There are 6 GPIO pins used to interface LCD to MCU chip

- 2 control signals
  - LCD_RS is connected to GPIO0, pin#6
  - LCD_EN is connected to GPIO1, pin#11
  - *LCD_RW in connected to Ground.*
- 4 data lines
  - LCD_D4 to LCD_D7 are connected to GPIO3, pin#0 to pin#3
  - *LCD_D0 to LCD_D3 have no connection*



LCD functionality

LCD module has a processor (HITACHI HD44780) and 32 bytes of RAM.

- There are 2 internal 8-bit registers: Command Register and Data Register
  - To program the Command register, LCD_RS pin to be at LOW state
  - To program the Data register, LCD_RS pin to be at HIGH state
- We need to split the 8-bit data (command/ data) into 2 nibbles and higher nibble first.
- To latch the data into LCD registers, a falling edge (HIGH to LOW transition) trigger is needed at LCD_EN pin.

How to write to LCD command register

- LCD_RS (GPIO0_6) to be set to LOW
- Write the higher nibble in GPIO3_0 to GPIO3_3
- Set a falling edge on LCD_EN (GPIO1_11)
  - HIGH to LOW transition
- Write the lower nibble in GPIO3_0 to GPIO3_3
- Set a falling edge on LCD_EN (GPIO1_11)
  - HIGH to LOW transition

How to write to LCD data register

- LCD_RS (GPIO0_6) to be set to HIGH
- Write the higher nibble in GPIO3_0 to GPIO3_3
- Set a falling edge on LCD_EN (GPIO1_11)
  - HIGH to LOW transition
- Write the lower nibble in GPIO3_0 to GPIO3_3
- Set a falling edge on LCD_EN (GPIO1_11)
  - HIGH to LOW transition

Embedded C module for Unit testing: lcd.c/ lcd.h

Functions to unit test

- void LCDSetCmd (unsigned char cmd)
  - Input Parameters: 8-bit command
  - Return Value: None
- char LCDPutChar (char dat)
  - Input Parameters: alphanumeric character
  - Return Value: same as input

Dependent MCU on-chip module to mock: gpio.c/ .h (mock_gpio.h)
Functions to mock

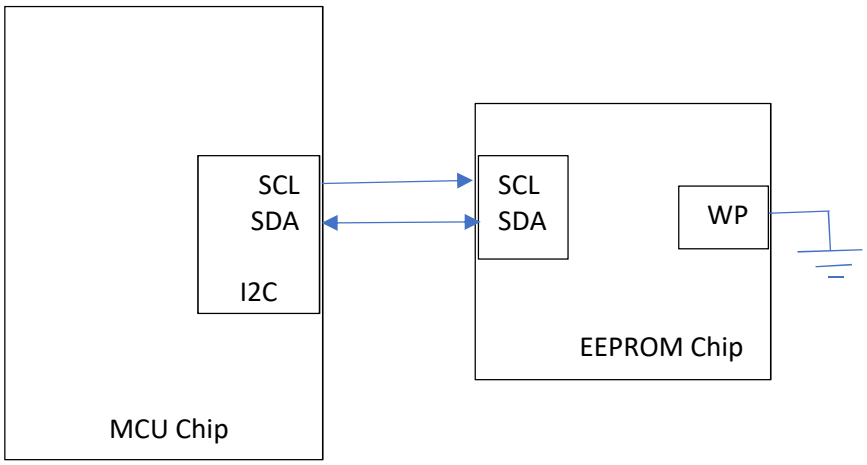
- GPIO3Set4Bit(dat)
- GPIO0SetPin(pin, option)
- GPIO1SetPin(pin, option)

ECU on-board module: Serial EEPROM

There is a serial EEPROM (24C08) on-board interfaced on I2C bus.

I2C is 2-wire synchronous half-duplex master slave organized serial IO bus:

- SCL (Serial Clock)
- SDA (Serial Data)



EEPROM internal memory locations can be accessed for data read/ write through I2C serial data transfer.

EEPROM internal memory map: 0x00 to 0xff

Data read or write is byte oriented.

- Data is written or read byte-wise

ECU on-board module for Unit testing: eeprom (eeprom.c/ .h)

Functions for unit testing

void EEPROMWrite (int addr, unsigned char *buf, int size);

- Input Parameter(s)
  - EEPROM Internal location (0x00 to 0xff)
  - Array of data bytes to write
  - No of bytes
- Return Value: None

void EEPROMRead (int addr, unsigned char *buf, int size);

- Input Parameter(s)
  - EEPROM Internal location (0x00 to 0xff)
  - Array of data bytes to read/ store
  - No of bytes
- Return Value: None

MCU on-chip module to mock: i2c (mock_i2c.h)

Functions to mock:

void I2CStart(void);

void I2CSelectSlave(unsigned char slaveID, int dir);

void I2CWriteByte(unsigned char byte);

void I2CStop(void);

unsigned char I2CReadByte(void);

void I2CSetMasterAckNack(int option);

I2C protocol for Serial EEPROM Write Bytes

- Phase 1: Start
- Phase 2: Select or Enable EEPROM for write
    - 2 Parameters: (I2C_ID, IO Direction)
        - EEPROM I2C Slave ID: A0
        - 0 for Write
- Phase 3: Select EEPROM internal location (00 to FF)
- Phase 4: Write Byte(s)
- Phase 5: Stop

I2C protocol for Serial EEPROM Read Bytes

- Phase 1: Start
- Phase 2: Select or Enable EEPROM for write
    - 2 Parameters: (I2C_ID, IO Direction)
        - EEPROM I2C Slave ID: A0
        - 0 for Write
- Phase 3: Select EEPROM internal location (00 to FF)
- Phase 4: Restart
- Phase 5: Select or Enable EEPROM for read
    - 2 Parameters: (I2C_ID, IO Direction)
        - EEPROM I2C Slave ID: A0
        - 1 for read
- Phase 6a:
    - Issue Master ACK
    - Read Byte(s)
- Phase 6b:
    - Issue Master NACK
    - Read Last Byte
- Phase 7: Stop