Test Protocol					
Test Case ID	**Test Case Description**	**Test Case Steps**	**Expected Result**	**Actual Result**	**Pass/Fail**
MCAL Module					
DIO Driver					
TC_DIO_001	**Test DIO initialization**	1. Call DIO_init with valid parameters	EN_DIO_Error_T is returned as DIO_OK	*Matches Expected Result*	Pass
TC_DIO_002	**Test DIO read**	1. Call DIO_init with valid parameters 2. Call DIO_write with a value of 1 3. Call DIO_read	u8_a_value is returned as 1	*Matches Expected Result*	Pass
TC_DIO_003	**Test DIO write**	1. Call DIO_init with valid parameters 2. Call DIO_write with a value of 1	EN_DIO_Error_T is returned as DIO_OK	*Matches Expected Result*	Pass
TC_DIO_004	**Test DIO toggle**	1. Call DIO_init with valid parameters 2. Call DIO_write with a value of 1 3. Call DIO_toggle 4. Call DIO_read	u8_a_value is returned as 0	*Matches Expected Result*	Pass
TC_DIO_005	**Test port initialization**	1. Call DIO_portInit with valid parameters	EN_DIO_Error_T is returned as DIO_OK	*Matches Expected Result*	Pass
TC_DIO_006	**Test port write**	1. Call DIO_portInit with valid parameters 2. Call DIO_portWrite with a value of 0x55 and a mask of **0xFF (DIO_NO_MASK)**	The specified port pins are set to 0x55	*Matches Expected Result*	Pass
TC_DIO_007	**Test port toggle**	1. Call DIO_portInit with valid parameters 2. Call DIO_portWrite with a value of 0x55 and a mask of **0xFF (DIO_NO_MASK**) 3. Call DIO_portToggle with a mask of **0xFF (DIO_NO_MASK**) 4. Call DIO_portWrite with a value of 0xAA and a mask of **0xFF (DIO_NO_MASK**) 5. Call DIO_portToggle with a mask of **0xFF (DIO_NO_MASK)**	The specified port pins are toggled between 0x55 and 0xAA	*Matches Expected Result*	Pass
TC_DIO_008	**Re-Test with invalid data**	Re-test cases from 1 to 7 with invalid data	Return DIO_ERROR	*Matches Expected Result*	Pass
EXI Driver					
TC_EXI_001	**Test EXI enablePIE**	1. Call EXI_enablePIE with valid parameters (Correct interruptId and senseControl)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_EXI_002	**Test EXI enablePIE**	1. Call EXI_enablePIE with invalid parameters (Incorrect interruptId or senseControl)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_EXI_003	**Test EXI disablePIE**	1. Call EXI_disablePIE with valid parameters (Correct interruptId)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_EXI_004	**Test EXI disablePIE**	1. Call EXI_disablePIE with invalid parameters (Incorrect interruptId)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_EXI_005	**Test EXI intSetCallBack**	1. Call EXI_intSetCallBack with valid parameters (Correct interruptId and Valid Pointer)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_EXI_006	**Test EXI intSetCallBack**	1. Call EXI_intSetCallBack with invalid parameters (Incorrect interruptId or Null Pointer)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
TIMER Driver					
TC_TIMER_001	**Test timer0 & timer2 Init**	1. Call TMR_timer0NormalModeInit & TMR_timer2NormalModeInit	The two timers working in the normal mode	*Matches Expected Result*	Pass
TC_TIMER_002	**Test timer0 delay function**	1. Call TMR_timer2Delay with deferent delay ranges	The desired delay is acheived	*Matches Expected Result*	Pass
TC_TIMER_003	**Test timer2 delay function**	1. Call TMR_timer0Delay with deferent delay ranges	The desired delay is acheived	*Matches Expected Result*	Pass
TC_TIMER_004	**Test timer0 & timer2 start**	1. Call TMR_timer0Start & TMR_timer2Start	The two timers start working in normal mode	*Matches Expected Result*	Pass
I2C Driver					
TC_I2C_001	**Test TWI_init**	1. Call TWI_init	TWI module is initialized	*Matches Expected Result*	Pass
TC_I2C_002	**TWI_start**	1. Call TWI_start	TWI module send a start bit.	*Matches Expected Result*	Pass
TC_I2C_003	**TWI_write**	1. Call TWI_write	TWI module read a data/address .	*Matches Expected Result*	Pass
TC_I2C_004	**TWI_readWithAck**	1. Call TWI_readWithAck	TWI module read data with Ack.	*Matches Expected Result*	Pass
TC_I2C_005	**TWI_readWithNAck**	1. Call TWI_readWithNAck	TWI module read data with Not_Ack.	*Matches Expected Result*	Pass
TC_I2C_006	**TWI_stop**	1. Call TWI_stop	TWI module send a stop bit.	*Matches Expected Result*	Pass
SPI Driver					
TC_SPI_001	**Test SPI send / receive**	1. Setup proteus simulation with SPI debugger 2. Setup MASTER loop to send numbers from 0 upwards 3. Setup SLAVE to echo what it recieves	Master recieves what was sent in the previous cycle	*Matches Expected Result*	Pass
TC_SPI_002	**Test SPI restart**	Repeat case 1, adding a SPI_restart() every 8 transmissions	SPI debugger shows restarts, data resumes to be echoed successfuly	*Matches Expected Result*	Pass
TC_SPI_003	**Test SPI stop**	Repeat case 1, but call SPI_stop() after 8 transmissions	No data is sent/recieved after 8 times as slave isn't shifting anything out SPI Debugger shows SPI inactive after 8 cycles	*Matches Expected Result*	Pass
UART Driver					
TC_UART_001	**Test UART initialization**	Call UART_initialization	UART peripheral is initialized with the configurations selected in the .config file	*Matches Expected Result*	Pass
TC_UART_002	**Test UART receiveByte**	Call UART_receiveByte with valid parameters (Correct interruptionMode and Valid Pointer)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_003	**Test UART receiveByte**	Call UART_receiveByte with invalid parameters (Incorrect interruptionMode or Null Pointer)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_004	**Test UART receiveByteBlock**	Call UART_receiveByte with valid parameters (Valid Pointer)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_005	**Test UART receiveByteBlock**	Call UART_receiveByte with invalid parameters (Null Pointer)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_006	**Test UART transmitByte**	Call UART_transmitByte with valid parameters (Correct interruptionMode and transmitByte)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_007	**Test UART transmitByte**	Call UART_transmitByte with invalid parameters (Inorrect interruptionMode or transmitByte)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_008	**Test UART transmitString**	Call UART_transmitString with valid parameters (Valid Pointer)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_009	**Test UART transmitString**	Call UART_transmitString with invalid parameters (Null Pointer)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_010	**Test UART RXCSetCallBack**	Call UART_RXCSetCallBack with valid parameters (Valid Pointer)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_011	**Test UART RXCSetCallBack**	Call UART_RXCSetCallBack with invalid parameters (Null Pointer)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_012	**Test UART UDRESetCallBack**	Call UART_UDRESetCallBack with valid parameters (Valid Pointer)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_013	**Test UART UDRESetCallBack**	Call UART_UDRESetCallBack with invalid parameters (Null Pointer)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_014	**Test UART TXCSetCallBack**	Call UART_TXCSetCallBack with valid parameters (Valid Pointer)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_UART_015	**Test UART TXCSetCallBack**	Call UART_TXCSetCallBack with invalid parameters (Null Pointer)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
HAL Module					
BUZ Driver					
TC_BUZ_001	**Test buzzer init**	Call BUZZER_init()	The Buzzer module should be initialized	*Matches Expected Result*	Pass
TC_BUZ_002	**Test buzzer on**	Call BUZZER_on()	The buzzer should be turned on	*Matches Expected Result*	Pass
TC_BUZ_003	**Test buzzer off**	Call BUZZER_off()	The buzzer should be turned off	*Matches Expected Result*	Pass
MBTN Driver					
TC_BTN_001	**Test Init**	Call the MBTN_init passing the button configuration	Btn pins are set as inputs, enabling pullup resistor	*Matches Expected Result*	Pass
TC_BTN_002	**Test not pressed**	Call getBtnState with button not pressed	function returns NOT_PRESSED	*Matches Expected Result*	Pass
TC_BTN_003	**Test short press**	Call getBtnState with button pressed and released shortly	return is RELEASED	*Matches Expected Result*	Pass
TC_BTN_004	**Test long press/release**	Call getBtnState with button pressed then after 2 seconds released	function breaks after 2 seconds exactly and returns LONG_RELEASED	*Matches Expected Result*	Pass
TC_BTN_005	**Test long press no release for 2s**	Call getBtnState with button pressed then after more than 2 seconds released	function breaks after 2 seconds exactly and returns LONG_RELEASED	*Matches Expected Result*	Pass
LCD Driver					
TC_LCD_001	**LCD Initialization**	Call LCD_init() function.	The LCD module should be initialized and ready to display data.	*Matches Expected Result*	Pass
TC_LCD_002	**Sending Command**	Call LCD_sendCommand() function with a valid command.	The command should be successfully sent to the LCD controller and executed.	*Matches Expected Result*	Pass

TC_LCD_003	**Sending Character**	Call LCD_sendChar() function with a valid character.	The character should be displayed on the LCD screen.	*Matches Expected Result*	Pass
TC_LCD_004	**Sending String**	Call LCD_sendString() function with a valid null-terminated string.	The string should be displayed on the LCD screen. If the character '\n' is encountered, the cursor should move to the beginning of the next line.	*Matches Expected Result*	Pass
TC_LCD_005	**Setting Cursor Position**	Call LCD_setCursor() function with valid line and column values.	The cursor should be set to the specified line and column on the LCD screen.	*Matches Expected Result*	Pass
TC_LCD_006	**Storing Custom Character**	Call LCD_storeCustomCharacter() function with a valid bitmap pattern and location.	The custom character should be successfully stored in the CGRAM of the LCD module.	*Matches Expected Result*	Pass
TC_LCD_007	**Clearing Display**	Call LCD_clear() function.	The LCD screen should be cleared and the cursor should be returned to the home position (line 0, column 0).	*Matches Expected Result*	Pass
			KPD Driver		
TC_KPD_001	**KPD_initKPD**	Call the KPD_initKPD	Rows Pins are set as *Output*, Columns Pins are set as *Input* (enabling pullup resist	*Matches Expected Result*	Pass
TC_KPD_002	**KPD_enableKPD**	Call KPD_enableKPD	Rows Pins are set as Output (enable or re-enable KPD)	*Matches Expected Result*	Pass
TC_KPD_003	**KPD_disableKPD**	Call KPD_disableKPD	Rows Pins are set as *Input* (disable KPD)	*Matches Expected Result*	Pass
TC_KPD_004	**KPD getPressedKey**	Call KPD_getPressedKey with valid parameters (Valid Pointer)	STD_OK is returned as u8 errorState	*Matches Expected Result*	Pass
TC_KPD_005	**KPD getPressedKey**	Call KPD_getPressedKey with invalid parameters (Null Pointer)	STD_NOK is returned as u8 errorState	*Matches Expected Result*	Pass
			EEPROM Driver		
TC_EEPROM_001	**Write on EEPROM**	1.Call the EEPROM_writeByte function.	one byte is written inside the ext_memory	*Matches Expected Result*	Pass
TC_EEPROM_002	**Read from EEPROM**	1.Call EEPROM_readByte function	one byte is returned from the specific address	*Matches Expected Result*	Pass
TC_EEPROM_003	**Initializing the EEPROM**	1.Call EEPROM_init function	the TWI module is initialized	*Matches Expected Result*	Pass
			APP Module		
			ATM ECU APP		
TC_ATM_APP_001	**Inputs before trigger signal**	Test all keypad keys and ENTER/0 Button before trigger signal	No action	*Matches Expected Result*	Pass
TC_ATM_APP_002	**Enter wrong PIN 3 times**	Insert Card, then enter user pin 3 times wrong in a row	System lock, Alarm ON until reset	*Matches Expected Result*	Pass
TC_ATM_APP_003	**Enter Invalid PIN (pin length)**	Try to enter invalid PIN length	No action, no more digits to be entered on LCD / System	*Matches Expected Result*	Pass
TC_ATM_APP_004	**Enter correct PIN**	Insert Card -> Enter correct PIN number	System logs user in -> ask for transaction amount	*Matches Expected Result*	Pass
			Card ECU APP		
TC_CARD_APP_001	**PAN > 19**	Enter PAN that is more than 19 chars	Fail, Wrong PAN [Not in Valid Range]	*Matches Expected Result*	Pass
TC_CARD_APP_002	**PAN < 16**	Enter PAN that is less than 16 chars	Fail, Wrong PAN [Not in Valid Range]	*Matches Expected Result*	Pass
TC_CARD_APP_003	**PAN with alpha chars**	Enter PAN with alphabetic characters	Fail, Wrong PAN [Non Numeric]"	*Matches Expected Result*	Pass
TC_CARD_APP_004	**Correct PAN**	Enter correct PAN in Valid Range (16 -> 19) [Numeric]	Pass, PAN is saved successfully	*Matches Expected Result*	Pass
TC_CARD_APP_005	**PIN < 4**	Enter PIN number all numeric but less than 4 characters	Fail, Wrong PIN [Not 4 Digits]	*Matches Expected Result*	Pass
TC_CARD_APP_006	**PIN > 4**	Enter PIN number all numeric but more than 4 characters	Fail, Wrong PIN [Not 4 Digits]	*Matches Expected Result*	Pass
TC_CARD_APP_007	**PIN with alpha**	Enter PIN number with correct length but with alphanumeric characters	Fail, Wrong PIN [Non Numeric]	*Matches Expected Result*	Pass
TC_CARD_APP_008	**Different confirmation PIN**	Enter different PIN confirmation that the first PIN	Fail, Wrong PIN [Non Identical]	*Matches Expected Result*	Pass
TC_CARD_APP_009	**Correct PIN**	Enter correct PIN 4 chars in length [Numeric]	Pass, Will ask for the Confirmation PIN	*Matches Expected Result*	Pass
TC_CARD_APP_010	**Correct PIN confirmation**	Enter correct PIN 4 chars in length [Numeric and Identical to first PIN]	Pass, PIN is saved successfully	*Matches Expected Result*	Pass
			User Stories		
TC_ATM_APP_001	**Fraud Card**	Insert a card that has no entry on the ATM database	System lock, FRAUD CARD message is displayed, Alarm ON until reset	*Matches Expected Result*	Pass
TC_USER_STR_002	**Stolen Card**	Insert a card that has an entry in ATM DB but the card status is BLOCKED	System lock, STOLEN CARD message is displayed, Alarm ON until reset	*Matches Expected Result*	Pass
TC_USER_STR_003	**Max limit exceeded**	Insert a valid card, try to make a transaction that is above the limit (5000.0)	Rejected, max limit exceeded, reset to insertion of amount again	*Matches Expected Result*	Pass
TC_USER_STR_004	**Insufficient fund**	Insert a valid card, try to make a transaction that is not above the limit but the user account doesn't have enough funds for it	Rejected, insufficient exceeded, reset to insertion of amount again	*Matches Expected Result*	Pass
TC_USER_STR_005	**Approved Transaction**	Insert a valid card, try to make a transaction that is not above the limit and the user account balance covers it	Approved, Rem. balance is: xxxx, Ejecting card, reset system	*Matches Expected Result*	Pass