AUTOMOTIVE DOOR CONTROL SYSTEM DESIGN

Embedded Systems Advanced Track

Egyptfwd

Submitted by: Salma Zakaria Mohamed Ibrahim

APIs Table Description

For both µCs:

1. *Memory Manager Module*
2. *CAN Module*
3. *GPIO Module*
4. *Timer Module*

For ECU 1:

1. *Speed Sensor Module*
2. *Light Switch Module (Latch Button)*
3. *Door Sensor*

For ECU 2:

1. *Buzzer Module*

- ***Memory Manager Module***

API	Description	Parameters (In)
`extern void EEPROMIntEnable(uint32_t ui32IntFlags);`	Enables the EEPROM interrupt.	ui32IntFlags indicates which EEPROM interrupt source to enable.
`extern void EEPROMIntDisable(uint32_t ui32IntFlags);`	Disables the EEPROM interrupt.	ui32IntFlags indicates which EEPROM interrupt source to disable.
`extern uint32_t EEPROMIntStatus(bool bMasked);`	Reports the state of the EEPROM interrupt.	bMasked determines whether the masked or unmasked state of the interrupt is to be returned.
`extern void EEPROMIntClear(uint32_t ui32IntFlags);`	This function allows an application to clear the EEPROM interrupt.	ui32IntFlags indicates which interrupt sources to clear.
`extern uint32_t EEPROMInit(void);`	Performs any necessary recovery in case of power failures during write.	None
`extern uint32_t EEPROMSizeGet(void);`	Determines the size of the EEPROM.	None
`extern uint32_t EEPROMProgram(uint32_t *pui32Data, uint32_t ui32Address, uint32_t ui32Count);`	Writes data to the EEPROM.	- pui32Data points to the first word of data to write to the EEPROM. - ui32Address defines the byte address within the EEPROM that the data -ui32Count defines the number of bytes of data that is to be written.
`extern void EEPROMRead(uint32_t *pui32Data, uint32_t`	Reads data from the EEPROM.	- pui32Data points to the first word of data to write to the EEPROM.

`ui32Address, uint32_t ui32Count);`		- ui32Address defines the byte address within the EEPROM that the data -ui32Count defines the number of bytes of data that is to be written.

- ***CAN Module***

API	***Description***	***Parameters (In)***
`extern void CANInit(uint32_t ui32Base);`	Initializes the CAN controller after reset.	ui32Base is the base address of the CAN controller.
`extern void CANEnable(uint32_t ui32Base);`	Enables the CAN controller.	ui32Base is the base address of the CAN controller to enable.
`extern void CANDisable(uint32_t ui32Base);`	Disables the CAN controller.	ui32Base is the base address of the CAN controller to disable.
`extern void CANMessageGet(uint32_t ui32Base, uint32_t ui32ObjID, tCANMsgObject *psMsgObject, bool bClrPendingInt);`	Reads a CAN message from one of the message object buffers.	ui32Base is the base address of the CAN controller. ui32ObjID is the object number to read (1-32). psMsgObject points to a structure containing message object fields. bClrPendingInt indicates whether an associated interrupt should be cleared.
`extern void CANMessageSet(uint32_t ui32Base, uint32_t ui32ObjID, tCANMsgObject *psMsgObject, tMsgObjType eMsgType);`	Configures a message object in the CAN controller.	ui32Base is the base address of the CAN controller. ui32ObjID is the object number to configure (1-32). psMsgObject is a pointer to a structure containing message object settings.

		eMsgType indicates the type of message for this object.

- ***GPIO Module***

API	***Description***	***Parameters (In)***
`extern void GPIOPinTypeGPIOOutput(uint32_t ui32Port, uint8_t ui8Pins);`	Configures pin(s) for use as GPIO outputs.	ui32Port is the base address of the GPIO port. ui8Pins is the bit-packed representation of the pin(s).
`extern void GPIOPinWrite(uint32_t ui32Port, uint8_t ui8Pins, uint8_t ui8Val);`	Writes a value to the specified pin(s).	ui32Port is the base address of the GPIO port. ui8Pins is the bit-packed representation of the pin(s). ui8Val is the value to write to the pin(s).
`extern void GPIOPinTypeGPIOInput(uint32_t ui32Port, uint8_t ui8Pins);`	Configures pin(s) for use as GPIO inputs.	ui32Port is the base address of the GPIO port. ui8Pins is the bit-packed representation of the pin(s).

- ***Timer Module***

API	***Description***	***Parameters (In)***
`extern void TimerEnable(uint32_t ui32Base, uint32_t ui32Timer);`	Enables the timer(s).	- ui32Base is the base address of the timer module. - ui32Timer specifies the timer(s) to enable.

`extern void TimerDisable(uint32_t ui32Base, uint32_t ui32Timer);`	Disables the timer(s).	- ui32Base is the base address of the timer module. - ui32Timer specifies the timer(s) to enable.
`extern void TimerConfigure(uint32_t ui32Base, uint32_t ui32Config);`	Configures the timer(s).	- ui32Base is the base address of the timer module. - ui32Timer specifies the timer(s) to enable.
`extern void TimerLoadSet(uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value);`	Sets the timer load value.	- ui32Base is the base address of the timer module. - ui32Timer specifies the timer(s) to enable. - ui32Value is the load value.
`extern void TimerIntEnable(uint32_t ui32Base, uint32_t ui32IntFlags);`	Registers an interrupt handler for the timer interrupt.	- ui32Base is the base address of the timer module. - ui32Timer specifies the timer(s) to enable. - pfnHandler is a pointer to the function to be called when the timer //! interrupt occurs.

- ***Speed Sensor Module***

  For this module we will use CM3218 Speed Sensor which is connected through I2C Protocol**.**

API	***Description***	***Parameters (In)***
`CM3218Init();`	Initializes the CM3218 driver.	- a pointer to the CM3218 instance data. - a pointer to the I2C driver instance data.

		- the I2C address of the CM3218 device. - the function to be called when the initialization has - a pointer that is passed to the callback function.
`CM3218Read();`	Reads data from CM3218 registers.	- a pointer to the CM3218 instance data. - a pointer to the location to store the data that is read. - the number of register values bytes to read. - the function to be called when data read is complete - a pointer that is passed to the callback function.
`CM3218Write();`	Writes data to CM3218 registers.	- pointer to the CM3218 instance data. - the first register to write. - a pointer to the 16-bit register data to write. - the number of data bytes to write. - the function to be called when the data has been written. - a pointer that is passed to the callback function.
`CM3218DataRead();`	Reads the light data from the CM3218.	- pointer to the CM3218 instance data. - the function to be called when the data has been read - a pointer that is passed to the callback function.

- ***Light Switch Module (Push Button)***

API	*Description*	*Parameters (In)*
extern int32_t CircularButtonMsgProc(tWidget *psWidget, uint32_t ui32Msg, uint32_t ui32Param1, uint32_t ui32Param2);	Handles messages for a circular push button widget.	psWidget is a pointer to the push button widget. ui32Msg is the message. ui32Param1 is the first parameter to the message. ui32Param2 is the second parameter to the message.
extern void CircularButtonInit(tPushButtonWidget *psWidget, const tDisplay *psDisplay, int32_t i32X, int32_t i32Y, int32_t i32R);	Initializes a circular push button widget.	psWidget is a pointer to the push button widget to initialize. - psDisplay is a pointer to the display on which to draw the push button. - i32X is the X coordinate of the upper left corner of the push button. - i32Y is the Y coordinate of the upper left corner of the push button. - i32R is the radius of the push button.

- ***Door Sensor***

In this Design we use infrared sensor.

API	*Description*	*Parameters (In)*
extern uint_fast8_t ISL29023Init(tISL29023 *psInst, tI2CMInstance *psI2CInst,uint_fast8_t ui8I2CAddr,	Initializes the ISL29023 driver.	psInst is a pointer to the ISL29023 instance data. psI2CInst is a pointer to the I2C driver instance data.

`tSensorCallback *pfnCallback, void *pvCallbackData);`		ui8I2CAddr is the I2C address of the ISL29023 device. pfnCallback is the function to be called when the initialization has completed. pvCallbackData is a pointer that is passed to the callback function.
`extern uint_fast8_t ISL29023Read(tISL29023 *psInst, uint_fast8_t ui8Reg, uint8_t *pui8Data, uint_fast16_t ui16Count, tSensorCallback *pfnCallback, void *pvCallbackData); void *pvCallbackData);`	Reads data from ISL29023 registers.	psInst is a pointer to the ISL29023 instance data. ui8Reg is the first register to read. pui8Data is a pointer to the location to store the data that is read. ui16Count is the number of data bytes to read. pfnCallback is the function to be called when the data has been read. pvCallbackData is a pointer that is passed to the callback function.
`extern uint_fast8_t ISL29023Write(tISL29023 *psInst, uint_fast8_t ui8Reg, uint8_t *pui8Data, uint_fast16_t ui16Count, tSensorCallback *pfnCallback, void *pvCallbackData);`	Write register data to the ISL29023.	psInst is a pointer to the ISL29023 instance data. ui8Reg is the first register to write. pui8Data is a pointer to the data to write. ui16Count is the number of data bytes to write. pfnCallback is the function to be called when the data has been written.

		pvCallbackData is a pointer that is passed to the callback function.

- ***Buzzer Module***

API	***Description***	***Parameters (In)***
`void BuzzerInit(uint8 BuzzerPinNo);`	Initialize the buzzer.	BuzzerPinNo Pin Number
`void BuzzerOn(uint8 BuzzerPinNo);`	Enable the buzzer.	BuzzerPinNo Pin Number
`void BuzzerOff(uint8 BuzzerPinNo);`	Disable the buzzer.	BuzzerPinNo Pin Number
`void BuzzerToggle(uint8 BuzzerPinNo);`	Toggle the buzzer.	BuzzerPinNo Pin Number