Final Project

Philopatear Ashraf 15p8202
Nourhan Ayman 15p8167

Automotive Embedded Networking | 5/9/2019

Table of Contents

Introduction

This project is a kind of simulation of a CAN bus Diagnostic tool. One node is the diagnostic tool (node 1) and another one is the ECU that complies with node 1 commands.

In our case, commands to node 2 can be summarized as follows: turning on a specific on-board LED, controlling the state machine direction, and reading both the on-board LED state and the state machine direction, according to a given control session.

Default Control Session provides reading information from CAN state machine (node 2). Diagnostic tool can ask CAN State Machine for either the on-board LED state or the current state machine direction.

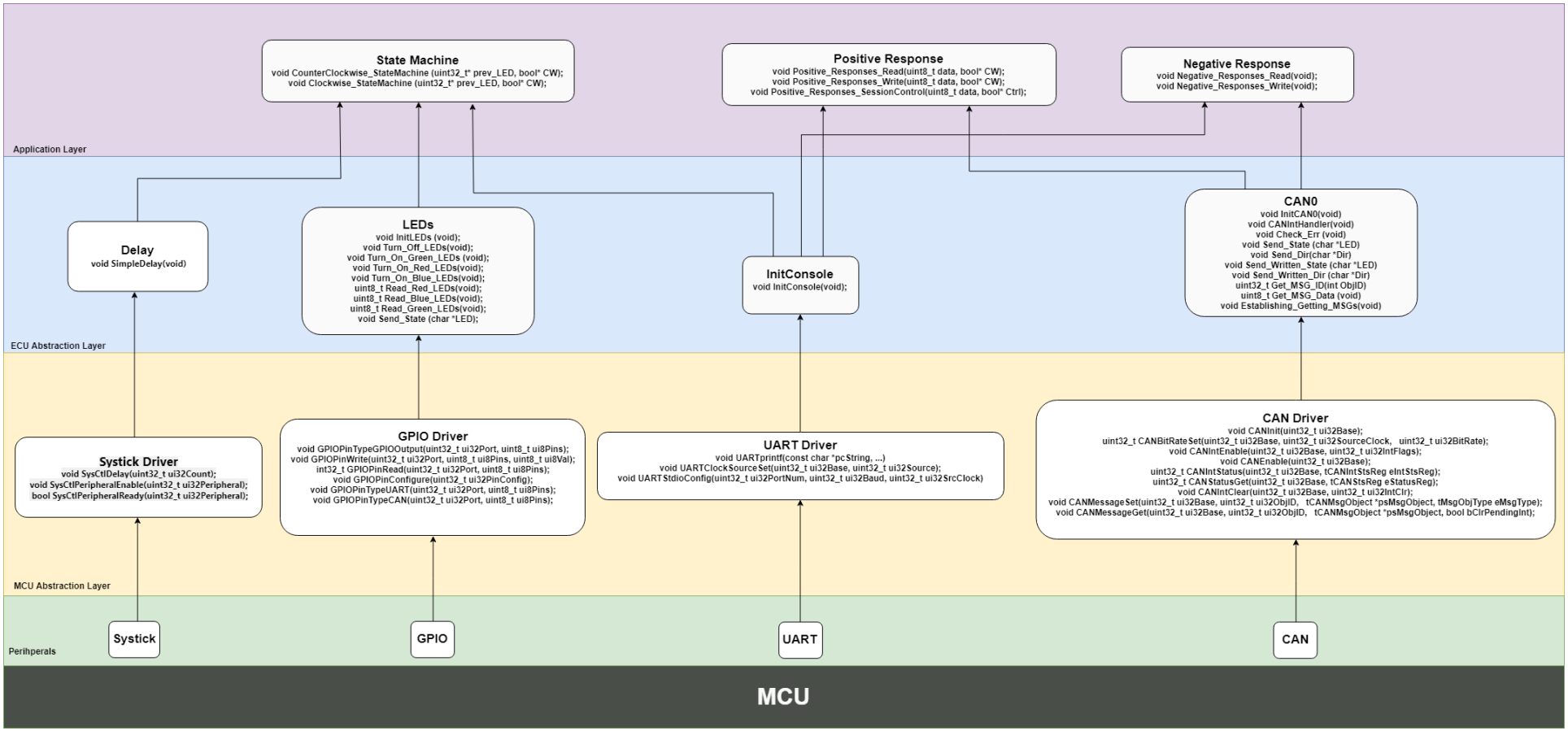
Extended Control Session provides writing on the CAN state machine node. Diagnostic tool can overwrite the LED states and the current state machine direction.

If diagnostic tool asked for a reading while in the extended control session or tried to overwrite while in default control session, it'll receive a negative response from the CAN state machine node.

In this document we represent for both nodes (diagnostic tool and CAN state machine) static architecture, API functions' description, data format of sending commands and receiving information, compilation warning report, and MISRA report.

CAN State Machine

STATIC ARCHITECTURE



API FUNCTIONS

ECU Abstraction Layer

CAN0

Function Name	`InitCAN0`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Initializes CAN0 Module.`

Function Name	`CANIntHandler`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Interrupt CAN Handler. Detects if an error occurred, sets receiving flag at successful receiving and clear interrupts at receiving and transmitting.`

Function Name	`Check_Err`
Arguments	`void`
Returned Parameter	`void`
Function Description	`If Error Flag is set by the CAN handler, prints that.`

Function Name	`Send_State`

Arguments	`Message ID (uint32_t MsgID):`
	`MsgID can be any ID for the message to be sent.`
	`LED State (const char *LED):`
	`*LED can be either "Red", "Green", or "Blue".`

Returned Parameter	`void`
Function Description	`Sends a message on CAN bus using CAN0 module with the Message ID at MsgID. Message contains LED State and is sent through Object 1.`

Function Name	Send_Dir
Arguments	Message ID (uint32_t MsgID): MsgID can be any ID for the message to be sent. State Machine Direction (const char *Dir): *Dir can be either Clockwise "CW”, or Counter Clockwise "CCW".
Returned Parameter	void
Function Description	Sends a message on CAN bus using CAN0 module with the Message ID at MsgID. Message contains State Machine direction and is sent through Object1.

Function Name	Send_Session
Arguments	Message ID (uint32_t MsgID): MsgID can be any ID for the message to be sent. Control Session (int Ctrl): Ctrl can be either 0 for Default Session or 1 for Extended Session.
Returned Parameter	void
Function Description	Sends a message on CAN bus using CAN0 module with the Message ID at MsgID. Message contains Control Session required and is sent through Object1.

Function Name	Establishing_Getting_MSGs
Arguments	void
Returned Parameter	void
Function Description	Establish receiving messages of any ID on CAN0 module through object 2.

Function Name	`Get_MSG_ID`
Arguments	`Object number (uint32_t ObjID) is object number of which a message has been received.`
Returned Parameter	`void`
Function Description	`Returns Message ID that's received on object number at ObjID. Can't be called unless Establishing_Getting_MSGs() is called prior.`

Function Name	`Get_MSG_Data`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Returns Message data that’s been received. Can't be called unless Establishing_Getting_MSGs() is called prior.`

LEDs

Function Name	`InitLEDs`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Initialize on-board LEDs.`

Function Name	`Turn_Off_LEDs`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Turn off on-board LEDs.`

Function Name	`Turn_On_Green_LEDs`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Turn on green on-board LEDs.`

Function Name	`Turn_On_Red_LEDs`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Turn on red on-board LEDs`

Function Name	`Turn_On_Blue_LEDs`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Turn on blue on-board LED.`

Function Name	`Read_Red_LEDs`
Arguments	`void`
Returned Parameter	`Red LED State of type uint8_t.`
Function Description	`Returns red on-board LED state.`

Function Name	`Read_Blue_LEDs`
Arguments	`void`
Returned Parameter	`Blue LED State of type uint8_t.`
Function Description	`Returns blue on-board LED state.`

Function Name	`Read_Green_LEDs`
Arguments	`void`
Returned Parameter	`Green LED State of type uint8_t.`
Function Description	`Returns green on-board LED state.`

InitConsole

Function Name	`InitConsole`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Establishes PC-MCU UART Communication.`

Delay

Function Name	`SimpleDelay`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Provides delay for 1 second.`

Application Layer

State Machine

Function Name	`CounterClockwise_StateMachine`

Arguments	`Previous LED state (int* prev_LED):`
	`*prev_LED is a pointer to previous LED color on State Machine.`
	`Direction (uint8_t* CW):`
	`*CW is a pointer to State Machine Direction.`

Returned Parameter	`void`
Function Description	`Sets State Machine Direction to be in a Counter Clockwise Direction.`

Function Name	`Clockwise_StateMachine`

Arguments	`Previous LED state (int* prev_LED):`
	`*prev_LED is a pointer to previous LED color on State Machine.`
	`Direction (uint8_t* CW):`
	`*CW is a pointer to State Machine Direction.`

Returned Parameter	`void`
Function Description	`Sets State Machine Direction to be in a Clockwise Direction.`

Positive Response

Function Name	`Positive_Responses_Read`

Arguments	`Data to be read (uint8_t data):`
	`data can be 1 to read LED states or 2 to read direction.`
	`State Machine Direction (const uint8_t* CW):`
	`*CW pointer to State Machine Direction either clockwise or counterclockwise.`

Returned Parameter	`void`
Function Description	`Sets State Machine Direction to be in a Clockwise Direction.`

Function Name	`Positive_Responses_Write`

Arguments	`Data to be read (uint8_t data):`
	`data can be 1 to set red LED or 2 to set blue LED or 3 to set green LED or 4 to set Clockwise direction for State Machine or 5 to set counterclockwise direction for State Machine Direction.`
	`State Machine Direction (uint8_t* CW):`
	`*CW pointer to State Machine Direction either clockwise or counterclockwise.`

Returned Parameter	`void`
Function Description	`Responds to Write Commands from Diagnostic tool if allowed.`

Function Name	`Positive_Responses_SessionControl`

Arguments	`Data to be read (uint8_t data):`
	`data can be 0 for default session or 1 for extended session.`
	`Session Control (uint8_t* Ctrl):`
	`*Ctrl pointer to Session Control either default or extended.`

Returned Parameter	`void`
Function Description	`Responds to Write Commands from Diagnostic tool if allowed.`

Negative Response

Function Name	`Negative_Response_Read`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Responds to Read Commands from Diagnostic tool if not allowed.`

Function Name	`Negative_Response_Write`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Responds to Write Commands from Diagnostic tool if not allowed.`

DATA FROMAT

Command	Data Format	Command		Data Format		Response			
						Read Control Command		Write Control Command	
Read	0x22	State		0		Positive	0x62	Negative	0x22
		Direction		1		Positive	0x62	Negative	0x22
Write	0x2E	State	Red	0	1	Negative	0x2E	Positive	0x6E
			Blue		2	Negative	0x2E	Positive	0x6E
			Green		3	Negative	0x2E	Positive	0x6E
		Direction	Clockwise	1	1	Negative	0x2E	Positive	0x6E
			Counterclockwise		2	Negative	0x2E	Positive	0x6E
Control	0x10	Read		0		Positive	0x50	Positive	0x50
		Write		1		Positive	0x50	Positive	0x50

COMPILATION WARNINGS

No compilation warnings.

MISRA

All **Required** rules are applied, except:

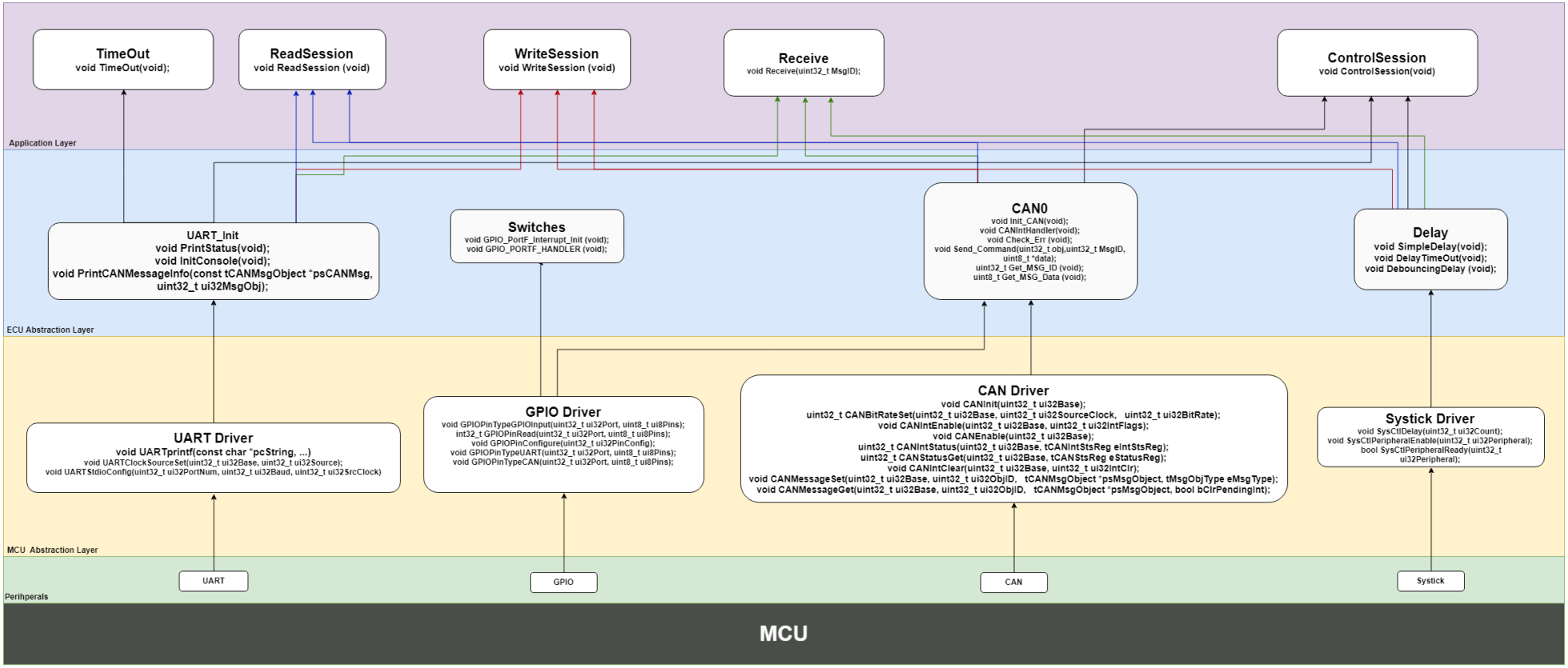
- MISRA C: 2004, 10.6 - A "U" suffix shall be applied to all constants of unsigned type.
  This rule isn't applied to lines 35 & 43 in CANo.c.
  This rule isn't applied to lines 12 & 15 in InitConsole.c.
  This rule isn't applied to lines 16 & 17 in LEDs.c.
  Justification: Tivaware API functions do not apply MISRAC rules.
  Although it can be resolved by replacing
  This line: `SysCtlPeripheralEnable ((uint32_t) SYSCTL_PERIPH_GPIOF)`
  By this one: `SysCtlPeripheralEnable ((uint32_t) 0xf0000805U)`

  We didn't use this solution since we're trying to avoid **hard-coding** as much as possible.
  Note: Modifying the `driverlib/sysctl.h` file didn't stop the error from being generated.
- MISRA-C: 2004. 19.4 - C macros shall only expand to a braced initializer, a constant, a string literal, a parenthesized expression, a type qualifier, a storage class specifier, or a do-while-zero construct.

  Justification: This rule generates the following error: gmake:*** [CANo.obj]Error1 for every c file in the project.
  This error has been represented on Texas Instrument website (by Eng. Sarea Hariri) and **hasn't been resolved** yet.
  Link: http://e2e.ti.com/support/tools/ccs/f/81/t/760304?Compiler-TM4C123GH6PM-can-t-build-the-project-gmake-error

System Diagnostics

STATIC ARCHITECTURE



API FUNCTIONS

ECU Abstraction Layer

CAN0

Function Name	`Init_CAN`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Initializes PortB and CAN0 module and their interrupt.`

Function Name	`CANIntHandler`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Interrupt CAN Handler. Detects if an error occurred, sets receiving flag at successful receiving and clear interrupts at receiving and transmitting.`

Function Name	`Check_Err`
Arguments	`void`
Returned Parameter	`void`
Function Description	`If Error Flag is set by the CAN handler, prints that.`

Function Name	`EstablishingGettingMessage`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Establish receiving messages of any ID on CAN0 module through object 1.`

Function Name	`Send_Command`

Arguments	`Object Number (uint32_t obj):`
	`Obj can be any number.`
	`Message ID (uint32_t MsgID):`
	`MsgID can be any ID for the message to be sent.`
	`Data (uint8_t *data):`
	`Pointer to data address that would be sent.`

Returned Parameter	`void`
Function Description	`Sends a message on CAN bus using CAN0 module with the Message ID at MsgID. Message contains Data to be sent and is sent through Object 2.`

Function Name	`Get_MSG_Data`
Arguments	`void`
Returned Parameter	`data received of data type uint8_t.`
Function Description	`Returns Message Data that's received on object number 1. Can't be called unless EstablishingGettingMessage() is called prior.`

Function Name	`Get_MSG_ID`
Arguments	`void`
Returned Parameter	`Message ID of data type uint32_t.`
Function Description	`Returns Message ID that's received on object number 1. Can't be called unless EstablishingGettingMessage() is called prior.`

UART_Init

Function Name	`InitConsole`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Establishes PC-MCU UART Communication.`

Function Name	`PrintCANMessageInfo`

Arguments	`Message Object (const tCANMsgObject *psCANMsg):`
	`Pointer to message object to be printed.`
	`uint32_t ui32MsgObj:`
	`Object number.`

Returned Parameter	`void`
Function Description	`Prints detailed information of a CAN Message.`

Function Name	`PrintStatus`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Prints Message ID and data that are received on the CAN0 module from node 2.`

Delay

Function Name	`SimpleDelay`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Provides delay for 1 second.`

Function Name	`DelayTimeOut`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Delay for 500 msec.`

Function Name	`DebouncingDelay`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Delay for 500 msec to avoid switches bouncing.`

Switches

Function Name	`Switches_Interrupt_Init`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Initializes on-board switches and their interrupts.`

Function Name	`Switches_HANDLER`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Switches Interrupt Handler, sends state of the switches and checks for errors while message is set.`

Application Layer

Time Out

Function Name	`TimeOut`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Indicates if there is no response for 500ms, prints a time out message and clears the receiving flag.`

Receive

Function Name	`Receive`
Arguments	`Message ID (MsgID):` `MsgID can be any ID.`
Returned Parameter	`void`
Function Description	`Receiving a Response from Node 2 and then Respond to the returned MsgID and Data also it's responsible for applying the current active session either Default session or Extended`

Control Session

Function Name	`ControlSession`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Sends Session Control Command and waits for a Timeout in case of no response or a late one.`

Read Session

Function Name	`ReadSession`
Arguments	`void`
Returned Parameter	`void`
Function Description	`Sends a command to read either the state or the direction. Waits for a specific number of milliseconds before discarding the received message.` `Receives a response and prints it, in case a response is available before the time out.`

Write Session

Function Name	`WriteSession`
Arguments	`void`
Returned Parameter	`void`

Function Description	`Sends a command to overwrite the state or the direction according to the input from user.`
	`Waits for a specific number of milliseconds before discarding the received message.`
	`Receives a response and prints it, in case a response is available before the time out.`

DATA FROMAT

Command	Data Format	Command		Data Format		Response			
						Read Control Command		Write Control Command	
Read	0x22	State		0		Positive	0x62	Negative	0x22
		Direction		1		Positive	0x62	Negative	0x22
Write	0x2E	State	Red	0	1	Negative	0x2E	Positive	0x6E
			Blue		2	Negative	0x2E	Positive	0x6E
			Green		3	Negative	0x2E	Positive	0x6E
		Direction	Clockwise	1	1	Negative	0x2E	Positive	0x6E
			Counterclockwise		2	Negative	0x2E	Positive	0x6E
Control	0x10	Read		0		Positive	0x50	Positive	0x50
		Write		1		Positive	0x50	Positive	0x50

COMPILATION WARNINGS

No compilation warnings.

MISRA

All **Required** and **Advisory** rules are applied, except:

- MISRA C: 2004, 10.6 - A "U" suffix shall be applied to all constants of unsigned type.
  This rule isn't applied to lines 34 & 38 in CANo.c.
  This rule isn't applied to lines 38 in Switches.c.
  This rule isn't applied to lines 18 & 21 in UARTInit.c.
  Justification: Tivaware API functions do not apply MISRAC rules.
  Although it can be resolved by replacing
  This line: `SysCtlPeripheralEnable ((uint32_t) SYSCTL_PERIPH_GPIOF)`
  By this one: `SysCtlPeripheralEnable ((uint32_t) 0xf0000805U)`

  We didn't use this solution since we're trying to avoid **hard-coding** as much as possible.
  Note: Modifying the `driverlib/sysctl.h` file didn't stop the error from being generated.

- MISRA-C: 2004. 19.4 - C macros shall only expand to a braced initializer, a constant, a string literal, a parenthesized expression, a type qualifier, a storage class specifier, or a do-while-zero construct.

  Justification: This rule generates the following error: gmake:*** [CANo.obj]Error1 for every c file in the project.
  This error has been represented on Texas Instrument website (by Eng. Sarea Hariri) and **hasn't been resolved** yet.
  Link: http://e2e.ti.com/support/tools/ccs/f/81/t/760304?Compiler-TM4C123GH6PM-can-t-build-the-project-gmake-error

- MISRA C: 2004, 11.3 - A cast should not be performed between a pointer type and an integral type.
  This rule isn't applied to lines 40, 41, 53 & 54 in Switches.c.
  Justification: Tivaware API functions do not apply MISRAC rules.