Lab 8 – Team 3

Alice Li, Navjot Singh, Aristos Athens, David LaFehr

**Mechanical hardware Description**

The body of the robot consists of two horizontal platforms. The motors and wheels, castor wheel, and tape sensor secured to the underside of the bottom platform. The Tiva is on the topside of the bottom platform. The top platform holds the breadboard which contains all of our circuitry.

Three pieces of duron separate the two levels, slotted into both platforms. Hot glue was added for improved stability. The tape sensor is glued to the front edge of the bottom platform. We empirically determined which height produced the best signal, and glued it at that height. At the front of the top platform is a column for holding the IR sensor at the same height as the beacon. Numerous holes were cut to allow us to adjust the height of the sensor. A piece of heat shrink was added around the IR sensor to block any signal coming from the sides. This ensures it only reads signals from a single, forward direction.

**Electrical Hardware Description**

The electrical hardware for this assignment can be broken down into four categories: SPI wiring, motor drive, beacon sensing, and tape sensing.

SPI: For the SPI hardware, the SCK, MOSI, MISO, and SS pins of the command generator connect to pins on the TIVA, each appropriately programmed to either receive or transmit. The command generator is the slave, which means that the TIVA provides the serial clock as well as toggling the slave select line.

Motors: The motor drive electronics consist of the use of both halves of an L293. One half of the L293 drives the left motor and the other half drives the right motor. Each motor is driven using drive-brake mode, i.e. enable pulled high, one motor input given a digital signal, and the other given a PWM signal. Each motor input has two snubbing diodes, one going to power and the other to ground.

IR Sensor: The beacon sensing circuit comprises four stages. The first stage is an IR phototransistor, which converts IR light into current, maintaining the signal frequency through this conversion. The second stage is a transimpedance amplifier, implemented with an MCP6294, which transforms the current into a voltage with the same frequency. The third stage is a passive high-pass filter whose corner frequency was selected so as to block ambient light and not attenuate signals with a frequency of 1.45 kHz. The fourth stage is an LM567 tone decoder, a device that outputs a low voltage when its input contains a frequency of interest, and a high voltage otherwise. The frequency of interest, 1.45 kHz in our case, is set by resistors and capacitors external to the chip.

Tape sensor: The tape sensor circuit also had four stages. The first was a QRB 1134 Reflective Object Sensor, which is a combination of IR emitter and IR phototransistor. The emitter was put in series with a resistor and attached to the rails. The phototransistor had its Collector leg attached to Vcc and its emitter leg attached to the next stage. The second stage was a trans-resistive circuit which use an MCP6294 op-amp to amplify the signal. It turns the current signal from the phototransistor into a corresponding inverted voltage. The third stage was a passive low-pass filter, to filter out noise. The last stage was a Schmitt trigger implemented with an LM339 opamp comparator. We tuned the Schmitt trigger based on empirical observations. We found our signal voltage (i.e. line read) was about 0.45\*Vcc, depending on the height of the line-reader. Thus we chose our Schmitt trigger values to be around 0.3\*Vcc. Vref high was 0.36\*Vcc and Vref low was 0.3\*Vcc (see design calculations).

**Software Description**

Our software consists of two modules: one for SPI and another for motor drive, which also includes the code to detect tape and the beacon.

The SPI module allows us to interface with the command generator using the SPI communication protocol. The SPI module queries the command generator every 100ms (sending 0xAA over the MOSI line), and generates an interrupt at the end of transmission. The ISR then reads what the command generator sent back (either 0xFF or a specific command). If it received 0xFF, then it sets a flag to know that the next byte received will be a new command. If it received a command and the flag is set, then the ISR posts the new command to the motor service and clears the flag. Otherwise, the ISR ignores the received byte. In order to use the end of transfer interrupt correctly, the interrupt is disabled at the beginning of the ISR and only enabled again when the service queries the command generator again at the 100ms timer timeout.

The purpose of the motor drive module is to generate signals to the motor inputs of the L293 when a new command is received from the command generator. The SPI module sends the command to the motor drive module, which first disables PWM outputs. Based on whether the command is to look for the beacon, to drive forward until the tape, or to do anything else, a different state is set, and, in the case of beacon and tape commands, digital signals are sent to the motor inputs of the L293, as are PWM signals after the appropriate registers have been set. These signals persist until there is a falling edge on the TIVA pin connected to the output of the beacon sensing circuit or to the output of the tape sensing circuit. After the falling edge, digital signals are sent to the L293 motor inputs, and PWM is disabled, PWM frequency and duty cycle are set and PWM pin mapping is done before PWM signals are sent to the motor inputs of the L293 to stop the motors. The same PWM process happens when a command other than looking for the beacon or driving forward until the tape is received.

**Design calculations**

-L293NE verification: One motor was measured to have a resistance of 7.7 Ω.

(5 V - 0 V) / (7.7 Ω) = 649 mA.

The other motor was measured to have a resistance of 5.7 Ω.

(5 V - 0 V) / (5.7 Ω) = 877 mA.

Each of these currents is below the maximum output current listed on the L293 datasheet, so the L293 is appropriate for the drive of these motors.

-High Pass Filter

We use a high pass filter to remove any DC offset and filter out low frequency noise. For the tone decoder to work well with the IR sensor, we want to ensure that our high pass filter doesn’t filter out the actual 1.45kHz signal. We want the cutoff frequency to be less than 1/5th of the signal frequency, but high enough that noise frequencies don’t interfere (we determined this empirically with testing).

Cut-off Frequency:

Fc = 1 / (2pi \* R \* C)

R = 10k

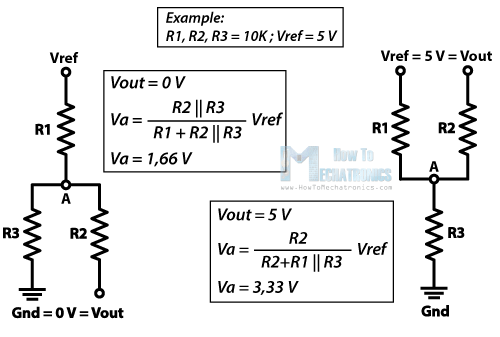
C = 0.1 uF

Fc = 160 Hz

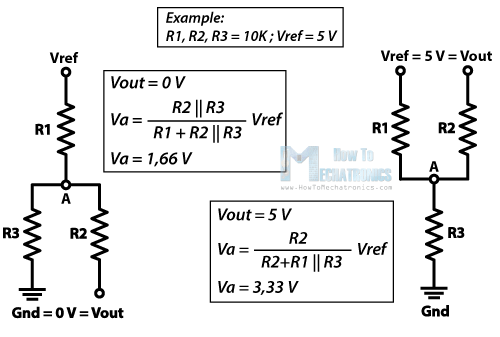
-Schmitt Trigger Values:

Choose R3 = 1M to limit power dissipation.

Low:



High:



Choose R1 and R2 that gets us close to our goal.

R1 = 100k + 100k = 200k

R2 = 47k + 47k = 94k

This gives:

Vref Low = 0.3\*Vcc

Vref High = 0.36\*Vcc

-Tone Decoder

We want our tone decoder to be centered at 1.45kHz, with a relatively narrow bandwidth (but not 0). The fundamental equations (using the suggested LM 567CN layout, which we implemented) are:

**Frequency** = f0 = 1.1/(R1 x C1)  
( 2k <= R1 <= 20k)

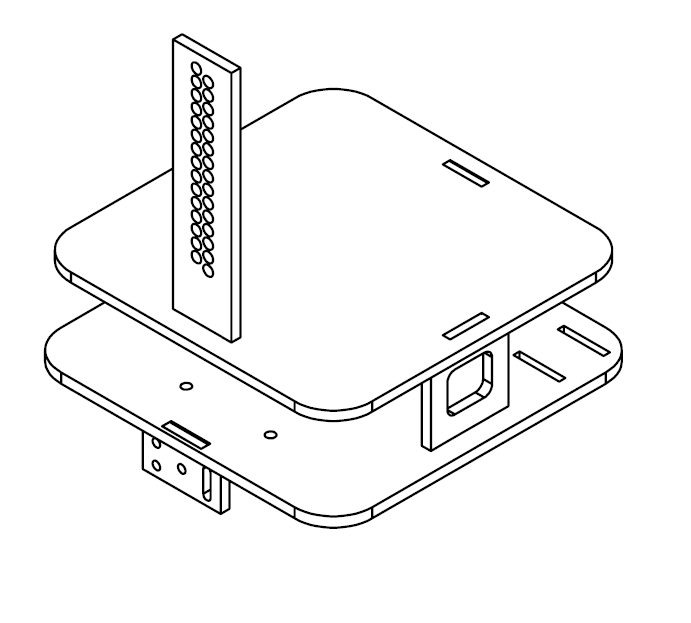
Where R1 is the timing resistor (pin 5) and C1 is the timing capacitor (pin 6). We used 0.1 uF for C1 then found the desired resistor value. We tuned R1 by hand, with a potentiometer. We confirmed the value by proving with a potentiometer.

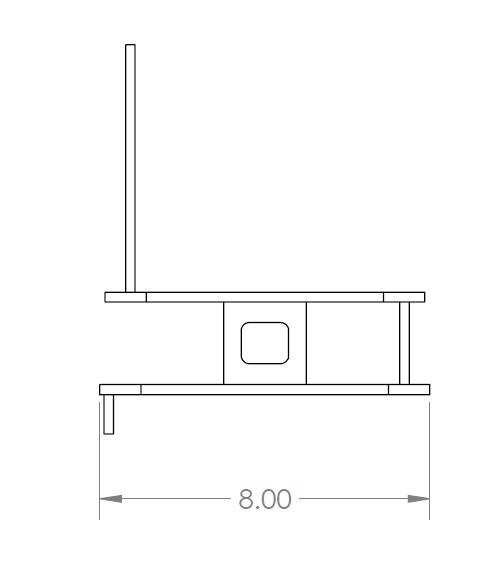
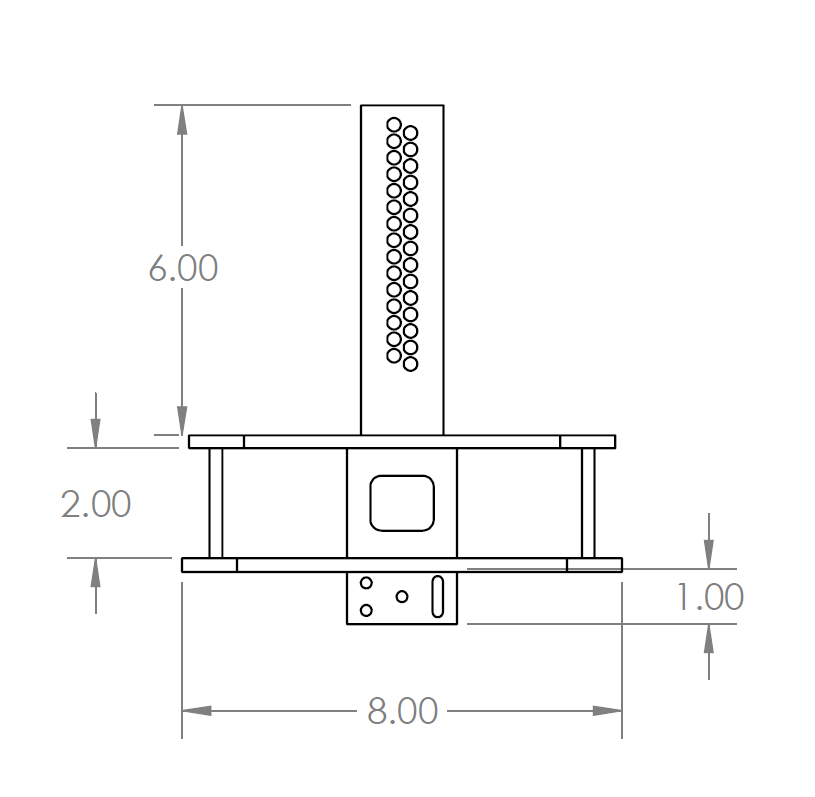
C1 = 0.1 uF

R1 = 7.6k

F0 = 1447 Hz

**Mechanical Drawings**





**Electrical Design**

**IR Sensor**



**Line Reader**

****

****

**Motor Circuit**

****

**Tiva Pins**

|  |  |  |  |
| --- | --- | --- | --- |
| Number | Description | Analog or Digital | Input or Output |
| PA2 | SSI0Clk (clock), alternate function 2 | Digital | Output |
| PA3 | SSI0Fss (slave select), alt func 2 | Digital | Output |
| PA4 | SSI0Rx (MISO), alt func 2 | Digital | Input |
| PA5 | SSI0Tx (MOSI), alt func 2 | Digital | Output |
| PB6 | Speed for left motor (M0PWM0, alt func 4) | Digital | Output |
| PB7 | Speed for right motor (M0PWM1, alt func 4) | Digital | Output |
| PB0 | Direction for left motor | Digital | Output |
| PB1 | Direction for right motor | Digital | Output |
| PB3 | Tape detection circuit output | Digital | Input |
| PB4 | Beacon detection circuit output | Digital | Input |
|  |  |  |  |

**Pseudocode**

*SPIService*

Data private to the module: MyPriority, Query (0xAA), CommandReady (0xFF),

NewCommand (0)

InitSPIService:

Declare ThisEvent

Set MyPriority to Priority

Call TERMIO\_Init

Call InitSPI

Set event type of ThisEvent to ES\_INIT

Return true

End SPIService

RunSPIService:

Declare ReturnEvent

Set event type of ReturnEvent to ES\_NO\_EVENT

Query the command generator

Enable the NVIC interrupt for SSI when starting to transmit (vector #23, interrupt #7)

Return ReturnEvent

End RunSPIService

CommGenISR:

Disable the NVIC interrupt for SSI when transmit finished (vector #23, interrupt #7)

Read the command generator

If command is CommandReady

Set NewCommand to 1

Elseif NewCommand is 1

Declare CommandEvent

Set event type of CommandEvent to COMMAND\_RECEIVED

Set event parameter of CommandEvent to Command

Post CommandEvent to MotorService

Set NewCommand to 0

Endif

Set COMM\_TIMER for QueryTime

End CommGenISR

InitSPI:

Enable the clock to the GPIO port

Enable the clock to the SSI module

Wait for the GPIO port to be ready

Program the GPIO to use the alternate functions on the SSI pins

Set mux position in GPIOPCTL to select the SSI use of the pins

Program the port lines for digital I/O

Program the required data directions on the port lines

Program the pull-up resistor

Wait for the SSI0 to be ready

Disable SSI

Select master mode and TXRIS indicating end of transmit (EOT)

Configure the SSI clock source to the system clock

Configure the clock pre-scaler

Configure the clock rate, phase and polarity, mode, data size

Enable local interrupts

Enable SSI

Query command generator

Enable the NVIC interrupt for SSI when starting to transmit (vector #23, interrupt #7)

End InitSPI

*MotorService*

Data private to the module: MyPriority, CurrentState, TapeFlag, CompareValueGenB

InitMotorDriveService:

Set MyPriority to Priority

Enable the clock to Module 0 of PWM

Enable the clock to Port B

Wait till the clock for Port B is ready

Select the PWM clock as System Clock / 32

Wait till clock for PWM is ready

Set PB0, PB1, PB3, and PB4 as digital pins

Set PB0 and PB1 as output pins

Set PB3 and PB4 as input pins

Set TapeFlag to 0

Set CompareValueGenB to 0

Return true

End InitMotorDriveService

RunMotorDriveService:

Declare ReturnEvent

Set ReturnEvent type to ES\_NO\_EVENT

Disable PWM outputs

If event type is ES\_TIMEOUT and event parameter is ROTATION\_TIMER

Call StopMotors

Elseif event type is COMMAND\_RECEIVED and event parameter is 0x20 (find beacon)

Set CurrentState to LookingForBeacon

Program generators to go to 1 at rising compare and 0 on falling compare

Set PB1 high

Set PB0 high

Elseif event type is COMMAND\_RECEIVED and event parameter is 0x40 (find tape)

Set CurrentState to LookingForTape

Set TapeFlag to 1

Program generators to go to 1 at rising compare and 0 on falling compare

Set PB0 high

Set PB1 low

Set compare value for Gen B to result in a 45% duty cycle

Elseif TapeFlag is 0 and event type is not TAPE\_DETECTED and event type is not BEACON\_DETECTED

Set CurrentState to IgnoringTapeAndBeacon

Endif

If event type is not ES\_TIMEOUT

switch on CurrentState

case CurrentState is IgnoringTapeAndBeacon

If event type is COMMAND\_RECEIVED

If event parameter is 0x00 (stop)

Call StopMotors

Endif

If event parameter is 0x02 (cw 90)

Program generators to go to 1 at rising compare and 0 on

falling compare

Set PB0 high

Set PB1 high

Set ROTATION\_TIMER for DURATION\_90

Endif

If event parameter is 0x03 (cw 45)

Program generators to go to 1 at rising compare and 0 on

falling compare

Set PB0 high

Set PB1 high

Set ROTATION\_TIMER for DURATION\_45

Endif

If event parameter is 0x04 (ccw 90)

Program generators to go to 1 at rising compare and 0 on

falling compare

Set PB0 low

Set PB1 low

Set ROTATION\_TIMER for DURATION\_90

Endif

If event parameter is 0x05 (ccw 45)

Program generators to go to 1 at rising compare and 0 on

falling compare

Set PB0 low

Set PB1 low

Set ROTATION\_TIMER for DURATION\_45

Endif

If event parameter is 0x08 (half forward)

Program generators to go to 1 at rising compare and 0 on

falling compare

Set PB0 low

Set PB1 low

Set compare value for Gen B to result in a 45% duty cycle

Endif

If event parameter is 0x09 (full forward)

Set GenA for 0% duty cycle

Program GenB to go to 1 at rising compare and 0 on

falling compare

Set PB0 high

Set PB1 low

Set compare value for Gen B to result in a 95% duty cycle

Endif

If event parameter is 0x10 (half reverse)

Program generators to go to 1 at rising compare and 0 on

falling compare

Set PB0 low

Set PB1 high

Set compare value for Gen B to result in a 45% duty cycle

Endif

If event parameter is 0x11 (full reverse)

Set GenA for 100% duty cycle

Program GenB to go to 1 at rising compare and 0 on

falling compare

Set PB0 low

Set PB1 high

Set compare value for Gen B to result in a 95% duty cycle

Endif

Break

Endif

case CurrentState is LookingForTape

If event type is TAPE\_DETECTED

Call StopMotors

Set TapeFlag to 0

Endif

Break

case CurrentState is LookingForBeacon

If event type is BEACON\_DETECTED

Call StopMotors

Endif

Break

Endif

Set load value

Set compare value for Gen A as COMPARE\_VALUE\_GENA

Set compare value for Gen B as CompareValueGenB

Enable M0PWM0 output

Enable M0PWM1 output

Select alternate function for PB6

Select alternate function for PB7

Map M0PWM0 to PB6

Map M0PWM1 to PB7

Set PB6 and PB7 as digital pins

Set PB6 and PB7 as output pins

Set up+down count mode, enable PWM generator, and make generate update locally synchronized to zero count

End RunMotorService

StopMotors:

Set GenA for 0% duty cycle

Set GenB for 0% duty cycle

Set PB0 low

Set PB1 low

End StopMotors

Check4Beacon:

Declare ThisEvent

Set event type of ThisEvent to BEACON\_DETECTED

Declare CurrentBeaconState

Set ReturnValue to false

Get current state of PB4

If CurrentBeaconState is 0

Post ThisEvent to MotorService

Set ReturnValue to true

Endif

Return ReturnValue

End Check4Beacon

Check4Tape:

Declare ThisEvent

Set event type of ThisEvent to TAPE\_DETECTED

Declare CurrentTapeState

Set ReturnValue to false

Get current state of PB3

If CurrentTapeState is 0

Post ThisEvent to MotorService

Set ReturnValue to true

Endif

Return ReturnValue

End Check4Tape

**Code**

Source files

SPIService.c

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Module

SPIService.c

Revision

1.0.1

Description

This is a file for implementing a simple service under the

Gen2 Events and Services Framework.

Notes

History

When Who What/Why

-------------- --- --------

01/16/12 09:58 jec began conversion from TemplateFSM.c

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*----------------------------- Include Files -----------------------------\*/

/\* include header files for this state machine as well as any machines at the

next lower level in the hierarchy that are sub-machines to this machine

\*/

#include "ES\_Configure.h"

#include "ES\_Framework.h"

#include "SPIService.h"

#include "MotorService.h"

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "inc/hw\_ssi.h"

#include "inc/hw\_nvic.h"

#include "inc/hw\_gpio.h"

#include "inc/hw\_timer.h"

#include "inc/hw\_sysctl.h"

#include "termio.h"

/\*----------------------------- Module Defines ----------------------------\*/

#define BitsPerNibble 4

#define TicksPerMS 40000

#define QueryTime 100

#define PreScaler 50

/\*---------------------------- Module Functions ---------------------------\*/

/\* prototypes for private functions for this service. They should be functions

relevant to the behavior of this service

\*/

void InitSPI(void);

/\*---------------------------- Module Variables ---------------------------\*/

// with the introduction of Gen2, we need a module level Priority variable

static uint8\_t MyPriority;

static const uint8\_t Query = 0xAA;

static const uint16\_t CommandReady = 0xFF;

static bool NewCommand = 0;

/\*------------------------------ Module Code ------------------------------\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function

InitSPIService

Parameters

uint8\_t : the priorty of this service

Returns

bool, false if error in initialization, true otherwise

Description

Saves away the priority, and does any

other required initialization for this service

Notes

Author

J. Edward Carryer, 01/16/12, 10:00

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool InitSPIService(uint8\_t Priority)

{

ES\_Event\_t ThisEvent;

MyPriority = Priority;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

in here you write your initialization code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

TERMIO\_Init();

InitSPI();

// post the initial transition event

ThisEvent.EventType = ES\_INIT;

if (ES\_PostToService(MyPriority, ThisEvent) == true)

{

return true;

}

else

{

return false;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function

PostSPIService

Parameters

EF\_Event ThisEvent ,the event to post to the queue

Returns

bool false if the Enqueue operation failed, true otherwise

Description

Posts an event to this state machine's queue

Notes

Author

J. Edward Carryer, 10/23/11, 19:25

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool PostSPIService(ES\_Event\_t ThisEvent)

{

return ES\_PostToService(MyPriority, ThisEvent);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function

RunSPIService

Parameters

ES\_Event\_t : the event to process

Returns

ES\_Event\_t, ES\_NO\_EVENT if no error ES\_ERROR otherwise

Description

add your description here

Notes

Author

J. Edward Carryer, 01/15/12, 15:23

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

ES\_Event\_t RunSPIService(ES\_Event\_t ThisEvent)

{

ES\_Event\_t ReturnEvent;

ReturnEvent.EventType = ES\_NO\_EVENT; // assume no errors

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

in here you write your service code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Query the command generator

HWREG(SSI0\_BASE + SSI\_O\_DR) = Query;

// Enable the NVIC interrupt for the SSI when starting to transmit (vector #23, Interrupt #7)

HWREG(NVIC\_EN0) |= BIT7HI;

return ReturnEvent;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

private functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void CommGenISR(void)

{

// Disable the NVIC interrupt for the SSI when transmit finished (vector #23, Interrupt #7)

HWREG(NVIC\_EN0) &= BIT7LO;

// Read the command generator

uint16\_t Command = HWREG(SSI0\_BASE + SSI\_O\_DR);

// If we receive 0xFF, set NewCommand to be ready for next command

if (Command == CommandReady)

{

NewCommand = 1;

// If not 0xFF and NewCommand ready, post Command to MotorService

}

else if (NewCommand == 1)

{

ES\_Event\_t CommandEvent;

CommandEvent.EventType = COMMAND\_RECEIVED;

CommandEvent.EventParam = (uint8\_t)Command;

PostMotorService(CommandEvent);

NewCommand = 0;

}

ES\_Timer\_InitTimer(COMM\_TIMER, QueryTime);

}

void InitSPI(void)

{

//Enable the clock to the GPIO port

HWREG(SYSCTL\_RCGCGPIO) |= BIT0HI;

// Enable the clock to SSI module

HWREG(SYSCTL\_RCGCSSI) |= BIT0HI;

// Wait for the GPIO port to be ready

while ((HWREG(SYSCTL\_PRGPIO) & SYSCTL\_PRGPIO\_R0) != SYSCTL\_PRGPIO\_R0)

{}

// Program the GPIO to use the alternate functions on the SSI pins

HWREG(GPIO\_PORTA\_BASE + GPIO\_O\_AFSEL) |= (BIT2HI | BIT3HI | BIT4HI | BIT5HI);

// Set mux position in GPIOPCTL to select the SSI use of the pins

HWREG(GPIO\_PORTA\_BASE + GPIO\_O\_PCTL) = (HWREG(GPIO\_PORTA\_BASE + GPIO\_O\_PCTL) & 0xff0000ff) \

+ (2 << (5 \* BitsPerNibble)) + (2 << (4 \* BitsPerNibble)) + \

(2 << (3 \* BitsPerNibble)) + (2 << (2 \* BitsPerNibble));

// Program the port lines for digital I/O

HWREG(GPIO\_PORTA\_BASE + GPIO\_O\_DEN) |= (BIT2HI | BIT3HI | BIT4HI | BIT5HI);

// Program the required data directions on the port lines

HWREG(GPIO\_PORTA\_BASE + GPIO\_O\_DIR) |= ((BIT2HI | BIT3HI | BIT5HI) & BIT4LO);

// If using SPI mode 3, program the pull-up on the clock line

HWREG(GPIO\_PORTA\_BASE + GPIO\_O\_PUR) |= BIT2HI;

// Wait for the SSI0 to be ready

while ((HWREG(SYSCTL\_PRSSI) & SYSCTL\_PRSSI\_R0) != SYSCTL\_PRSSI\_R0)

{}

// Make sure that the SSI is disabled before programming mode bits

HWREG(SSI0\_BASE + SSI\_O\_CR1) &= BIT1LO;

// Select master mode (MS) & TXRIS indicating End of Transmit (EOT)

HWREG(SSI0\_BASE + SSI\_O\_CR1) |= (SSI\_CR1\_EOT & (~SSI\_CR1\_MS));

// Configure the SSI clock source to the system clock

HWREG(SSI0\_BASE + SSI\_O\_CC) = SSI\_CC\_CS\_SYSPLL;

// Configure the clock pre-scaler: max frequency 961kHz

// SSInClk = SysClk / (CPSDVSR \*(1+SCR)), we want CPSDVSR\*(1+SCR) > 42

HWREG(SSI0\_BASE + SSI\_O\_CPSR) = PreScaler;

// Configure clock rate (SCR), phase & polarity (SPH, SPO), mode (FRF), data size (DSS)

HWREG(SSI0\_BASE + SSI\_O\_CR0) |= (SSI\_CR0\_SPH | SSI\_CR0\_SPO | SSI\_CR0\_DSS\_8); // +7 for 8-bit data size

// Locally enable interrupts (TXIM in SSIIM)

HWREG(SSI0\_BASE + SSI\_O\_IM) |= SSI\_IM\_TXIM;

// Make sure that the SSI is enabled for operation

HWREG(SSI0\_BASE + SSI\_O\_CR1) |= SSI\_CR1\_SSE;

// Query the command generator

HWREG(SSI0\_BASE + SSI\_O\_DR) = Query;

// Enable the NVIC interrupt for the SSI when starting to transmit (vector #23, Interrupt #7)

HWREG(NVIC\_EN0) |= BIT7HI;

}

/\*------------------------------- Footnotes -------------------------------\*/

/\*------------------------------ End of file ------------------------------\*/

MotorService.c

#include "ES\_Configure.h"

#include "ES\_Framework.h"

#include "ES\_DeferRecall.h"

#include "ES\_ShortTimer.h"

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "inc/hw\_gpio.h"

#include "inc/hw\_sysctl.h"

#include "driverlib/sysctl.h"

#include "driverlib/pin\_map.h"

#include "driverlib/gpio.h"

#include "MotorService.h"

//for PWM definitions

#include "inc/hw\_pwm.h"

#define STOP 0x00

#define CW\_90 0x02

#define CW\_45 0x03

#define CCW\_90 0x04

#define CCW\_45 0x05

#define FORWARD\_HALF 0x08

#define FORWARD\_FULL 0x09

#define REVERSE\_HALF 0x10

#define REVERSE\_FULL 0x11

#define FIND\_BEACON 0x20

#define FIND\_TAPE 0x40

#define DURATION\_90 1900

#define DURATION\_45 1000

#define BITS\_PER\_NIBBLE 4

#define PWM\_PIN\_NUMBER\_LEFT\_MOTOR 6

#define PWM\_PIN\_NUMBER\_RIGHT\_MOTOR 7

//Frequency of 500 Hz

#define LOAD\_VALUE 1250

#define COMPARE\_VALUE\_GENA LOAD\_VALUE >> 1

static uint8\_t MyPriority;

static MotorServiceState\_t CurrentState;

static uint8\_t TapeFlag;

static uint32\_t CompareValueGenB;

bool InitMotorService(uint8\_t Priority)

{

MyPriority = Priority;

//Enable the clock to Module 0 of PWM

HWREG(SYSCTL\_RCGCPWM) |= SYSCTL\_RCGCPWM\_R0;

//Enable the clock to Port B

HWREG(SYSCTL\_RCGCGPIO) |= SYSCTL\_RCGCGPIO\_R1;

//Wait till clock for Port B is ready

while ((HWREG(SYSCTL\_PRGPIO) & SYSCTL\_PRGPIO\_R1) != SYSCTL\_PRGPIO\_R1)

{}

//Select the PWM clock as System Clock / 32

HWREG(SYSCTL\_RCC) = (HWREG(SYSCTL\_RCC) & ~SYSCTL\_RCC\_PWMDIV\_M) |

(SYSCTL\_RCC\_USEPWMDIV | SYSCTL\_RCC\_PWMDIV\_32);

//Wait until the clock has started

while ((HWREG(SYSCTL\_PRPWM) & SYSCTL\_PRPWM\_R0) != SYSCTL\_PRPWM\_R0)

{}

//Set as digital

HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_DEN) |= (BIT0HI | BIT1HI | BIT3HI | BIT4HI);

//Set as outputs

HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_DIR) |= (BIT0HI | BIT1HI);

//Set as inputs

HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_DIR) &= (BIT3LO & BIT4LO);

//Currently not looking for tape

TapeFlag = 0;

//Dummy value

CompareValueGenB = 0;

return true;

}

bool PostMotorService(ES\_Event\_t ThisEvent)

{

return ES\_PostToService(MyPriority, ThisEvent);

}

ES\_Event\_t RunMotorService(ES\_Event\_t ThisEvent)

{

ES\_Event\_t ReturnEvent;

ReturnEvent.EventType = ES\_NO\_EVENT;

//Disable PWM while initializing

HWREG(PWM0\_BASE + PWM\_O\_0\_CTL) = 0;

if ((ThisEvent.EventType == ES\_TIMEOUT) && (ThisEvent.EventParam == ROTATION\_TIMER))

{

StopMotors();

}

else if ((ThisEvent.EventType == COMMAND\_RECEIVED) && (ThisEvent.EventParam == FIND\_BEACON))

{

CurrentState = LookingForBeacon;

//Program generators to go to 1 at rising compare and 0 on falling compare

uint32\_t GenA\_Normal = (PWM\_0\_GENA\_ACTCMPAU\_ONE | PWM\_0\_GENA\_ACTCMPAD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = GenA\_Normal;

uint32\_t GenB\_Normal = (PWM\_0\_GENB\_ACTCMPBU\_ONE | PWM\_0\_GENB\_ACTCMPBD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = GenB\_Normal;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT1HI;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT0HI;

}

else if ((ThisEvent.EventType == COMMAND\_RECEIVED) && (ThisEvent.EventParam == FIND\_TAPE))

{

CurrentState = LookingForTape;

//Currently looking for tape

TapeFlag = 1;

//Program generators to go to 1 at rising compare and 0 on falling compare

uint32\_t GenA\_Normal = (PWM\_0\_GENA\_ACTCMPAU\_ONE | PWM\_0\_GENA\_ACTCMPAD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = GenA\_Normal;

uint32\_t GenB\_Normal = (PWM\_0\_GENB\_ACTCMPBU\_ONE | PWM\_0\_GENB\_ACTCMPBD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = GenB\_Normal;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT0HI;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT1LO;

CompareValueGenB = LOAD\_VALUE - (LOAD\_VALUE \* 45) / 100;

}

else if ((TapeFlag == 0) && (ThisEvent.EventType != TAPE\_DETECTED) && (ThisEvent.EventType != BEACON\_DETECTED))

{

CurrentState = IgnoringTapeAndBeacon;

}

//No need to enter this block if event type is a timeout

if (ThisEvent.EventType != ES\_TIMEOUT)

{

switch (CurrentState)

{

//Code does not respond to pins associated with beacon- and tape-detecting circuits

case IgnoringTapeAndBeacon:

{

if (ThisEvent.EventType == COMMAND\_RECEIVED)

{

if (ThisEvent.EventParam == STOP)

{

//Set GenA and GenB for 0% duty cycle

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = PWM\_0\_GENA\_ACTZERO\_ZERO;

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = PWM\_0\_GENB\_ACTZERO\_ZERO;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT0LO;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT1LO;

}

else if (ThisEvent.EventParam == CW\_90)

{

//Program generators to go to 1 at rising compare and 0 on falling compare

uint32\_t GenA\_Normal = (PWM\_0\_GENA\_ACTCMPAU\_ONE | PWM\_0\_GENA\_ACTCMPAD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = GenA\_Normal;

uint32\_t GenB\_Normal = (PWM\_0\_GENB\_ACTCMPBU\_ONE | PWM\_0\_GENB\_ACTCMPBD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = GenB\_Normal;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT0HI;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT1HI;

//Set timer for length of time it takes to rotate 90 degrees

ES\_Timer\_InitTimer(ROTATION\_TIMER, DURATION\_90);

}

else if (ThisEvent.EventParam == CW\_45)

{

//Program generators to go to 1 at rising compare and 0 on falling compare

uint32\_t GenA\_Normal = (PWM\_0\_GENA\_ACTCMPAU\_ONE | PWM\_0\_GENA\_ACTCMPAD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = GenA\_Normal;

uint32\_t GenB\_Normal = (PWM\_0\_GENB\_ACTCMPBU\_ONE | PWM\_0\_GENB\_ACTCMPBD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = GenB\_Normal;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT0HI;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT1HI;

//Set timer for length of time it takes to rotate 45 degrees

ES\_Timer\_InitTimer(ROTATION\_TIMER, DURATION\_45);

}

else if (ThisEvent.EventParam == CCW\_90)

{

//Program generators to go to 1 at rising compare and 0 on falling compare

uint32\_t GenA\_Normal = (PWM\_0\_GENA\_ACTCMPAU\_ONE | PWM\_0\_GENA\_ACTCMPAD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = GenA\_Normal;

uint32\_t GenB\_Normal = (PWM\_0\_GENB\_ACTCMPBU\_ONE | PWM\_0\_GENB\_ACTCMPBD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = GenB\_Normal;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT1LO;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT0LO;

//Set timer for length of time it takes to rotate 90 degrees

ES\_Timer\_InitTimer(ROTATION\_TIMER, DURATION\_90);

}

else if (ThisEvent.EventParam == CCW\_45)

{

//Program generators to go to 1 at rising compare and 0 on falling compare

uint32\_t GenA\_Normal = (PWM\_0\_GENA\_ACTCMPAU\_ONE | PWM\_0\_GENA\_ACTCMPAD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = GenA\_Normal;

uint32\_t GenB\_Normal = (PWM\_0\_GENB\_ACTCMPBU\_ONE | PWM\_0\_GENB\_ACTCMPBD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = GenB\_Normal;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT1LO;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT0LO;

//Set timer for length of time it takes to rotate 45 degrees

ES\_Timer\_InitTimer(ROTATION\_TIMER, DURATION\_45);

}

else if (ThisEvent.EventParam == FORWARD\_HALF)

{

//Program generators to go to 1 at rising compare and 0 on falling compare

uint32\_t GenA\_Normal = (PWM\_0\_GENA\_ACTCMPAU\_ONE | PWM\_0\_GENA\_ACTCMPAD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = GenA\_Normal;

uint32\_t GenB\_Normal = (PWM\_0\_GENB\_ACTCMPBU\_ONE | PWM\_0\_GENB\_ACTCMPBD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = GenB\_Normal;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT0HI;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT1LO;

CompareValueGenB = LOAD\_VALUE - (LOAD\_VALUE \* 45) / 100;

}

else if (ThisEvent.EventParam == FORWARD\_FULL)

{

//Set Gen A for 0% duty cycle

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = PWM\_0\_GENA\_ACTZERO\_ZERO;

//Program Gen B to go to 1 at rising compare and 0 on falling compare

uint32\_t GenB\_Normal = (PWM\_0\_GENB\_ACTCMPBU\_ONE | PWM\_0\_GENB\_ACTCMPBD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = GenB\_Normal;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT0HI;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT1LO;

CompareValueGenB = LOAD\_VALUE - (LOAD\_VALUE \* 95) / 100;

}

else if (ThisEvent.EventParam == REVERSE\_HALF)

{

//Program generators to go to 1 at rising compare and 0 on falling compare

uint32\_t GenA\_Normal = (PWM\_0\_GENA\_ACTCMPAU\_ONE | PWM\_0\_GENA\_ACTCMPAD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = GenA\_Normal;

uint32\_t GenB\_Normal = (PWM\_0\_GENB\_ACTCMPBU\_ONE | PWM\_0\_GENB\_ACTCMPBD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = GenB\_Normal;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT0LO;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT1HI;

CompareValueGenB = (LOAD\_VALUE \* 45) / 100;

}

else if (ThisEvent.EventParam == REVERSE\_FULL)

{

//Set Gen A for 100% duty cycle

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = PWM\_0\_GENA\_ACTZERO\_ONE;

//Program Gen B to go to 1 at rising compare and 0 on falling compare

uint32\_t GenB\_Normal = (PWM\_0\_GENB\_ACTCMPBU\_ONE | PWM\_0\_GENB\_ACTCMPBD\_ZERO);

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = GenB\_Normal;

//Set directional pins

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT0LO;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) |= BIT1HI;

CompareValueGenB = (LOAD\_VALUE \* 95) / 100;

}

}

break;

}

case LookingForTape:

{

if (ThisEvent.EventType == TAPE\_DETECTED)

{

StopMotors();

//Currently not looking for tape

TapeFlag = 0;

}

break;

}

case LookingForBeacon:

{

if (ThisEvent.EventType == BEACON\_DETECTED)

{

StopMotors();

}

break;

}

}

}

//Set period

HWREG(PWM0\_BASE + PWM\_O\_0\_LOAD) = LOAD\_VALUE;

//Set value at which PWM edges occur

HWREG(PWM0\_BASE + PWM\_O\_0\_CMPA) = COMPARE\_VALUE\_GENA;

HWREG(PWM0\_BASE + PWM\_O\_0\_CMPB) = CompareValueGenB;

//Enable PWM output

HWREG(PWM0\_BASE + PWM\_O\_ENABLE) |= (PWM\_ENABLE\_PWM0EN | PWM\_ENABLE\_PWM1EN);

//Select an alternate function for PB6 and PB7

HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_AFSEL) |= BIT6HI;

HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_AFSEL) |= BIT7HI;

//Map PWM to PB6. 4 comes from Table 23-5 on Page 1351 of TIVA datasheet

HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_PCTL) = (HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_PCTL) & 0xF0FFFFFF)

+ (4 << (PWM\_PIN\_NUMBER\_LEFT\_MOTOR \* BITS\_PER\_NIBBLE));

//Map PWM to PB7. 4 comes from Table 23-5 on Page 1351 of TIVA datasheet

HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_PCTL) = (HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_PCTL) & 0x0FFFFFFF)

+ (4 << (PWM\_PIN\_NUMBER\_RIGHT\_MOTOR \* BITS\_PER\_NIBBLE));

//Set PB6 and PB7 as digital

HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_DEN) |= (BIT6HI | BIT7HI);

//Set PB6 and PB7 as outputs

HWREG(GPIO\_PORTB\_BASE + GPIO\_O\_DIR) |= (BIT6HI | BIT7HI);

//Set up+down count mode, enable PWM generator, and make generate update locally synchronized to zero count

HWREG(PWM0\_BASE + PWM\_O\_0\_CTL) = (PWM\_0\_CTL\_MODE | PWM\_0\_CTL\_ENABLE | PWM\_0\_CTL\_GENAUPD\_LS

| PWM\_0\_CTL\_GENBUPD\_LS);

return ReturnEvent;

}

//Function to check state of pin connected to output of beacon-detecting circuit

bool Check4Beacon(void)

{

ES\_Event\_t ThisEvent;

ThisEvent.EventType = BEACON\_DETECTED;

uint8\_t CurrentBeaconState;

bool ReturnValue = false;

//Get current state of pin

CurrentBeaconState = (BIT4HI & HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)));

//If pin is low, the beacon has been detected

if (CurrentBeaconState == 0)

{

PostMotorService(ThisEvent);

ReturnValue = true;

}

return ReturnValue;

}

//Function to check state of pin connected to output of tape-detecting circuit

bool Check4Tape(void)

{

ES\_Event\_t ThisEvent;

ThisEvent.EventType = TAPE\_DETECTED;

uint8\_t CurrentTapeState;

bool ReturnValue = false;

//Get current state of pin

CurrentTapeState = (BIT3HI & HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)));

//If pin is low, black tape has been detected

if (CurrentTapeState == 0)

{

PostMotorService(ThisEvent);

ReturnValue = true;

}

return ReturnValue;

}

static void StopMotors(void)

{

//Set GenA and GenB for 0% duty cycle

HWREG(PWM0\_BASE + PWM\_O\_0\_GENA) = PWM\_0\_GENA\_ACTZERO\_ZERO;

HWREG(PWM0\_BASE + PWM\_O\_0\_GENB) = PWM\_0\_GENB\_ACTZERO\_ZERO;

//Set directional pins such that no current flows

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT0LO;

HWREG(GPIO\_PORTB\_BASE + (GPIO\_O\_DATA + ALL\_BITS)) &= BIT1LO;

}

Header files

SPIService.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Header file for SPI service

based on the Gen 2 Events and Services Framework

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef ServSPI\_H

#define ServSPI\_H

#include "ES\_Types.h"

// Public Function Prototypes

bool InitSPIService(uint8\_t Priority);

bool PostSPIService(ES\_Event\_t ThisEvent);

ES\_Event\_t RunSPIService(ES\_Event\_t ThisEvent);

#endif /\* ServSPI\_H \*/

MotorService.h

#ifndef MotorService\_H

#define MotorService\_H

//Event Definitions

#include "ES\_Configure.h" /\* gets us event definitions \*/

#include "ES\_Types.h" /\* gets bool type for returns \*/

//Typedefs for the states

typedef enum

{

IgnoringTapeAndBeacon, LookingForTape, LookingForBeacon

}MotorServiceState\_t;

//Function Prototypes

bool InitMotorService(uint8\_t Priority);

bool PostMotorService(ES\_Event\_t ThisEvent);

ES\_Event\_t RunMotorService(ES\_Event\_t ThisEvent);

bool Check4Beacon(void);

bool Check4Tape(void);

static void StopMotors(void);

#endif