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:

$$Va = \frac{R2 \,||\, R3}{R1 + R2 \,||\, R3} Vref$$

High:

$$Va = \frac{R2}{R2 + R1 \,||\, R3} Vref$$

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

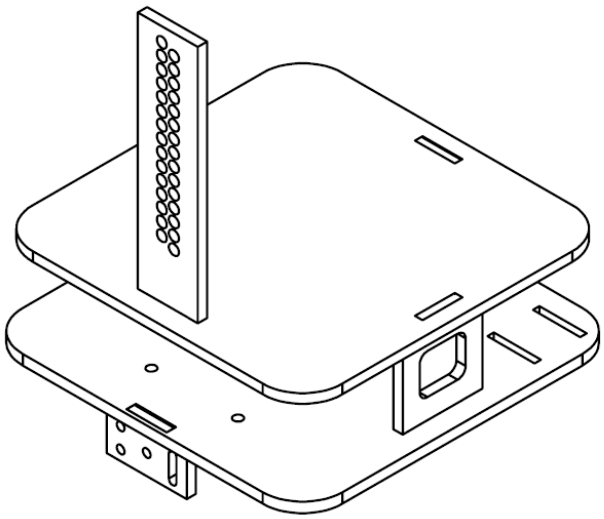
We want our tone decoder to be centered at 1.45kHz, with a relatively narrow bandwidth (but not 0). The fundamental equations (using the suggested LM 567CN layout, which we implemented) are:

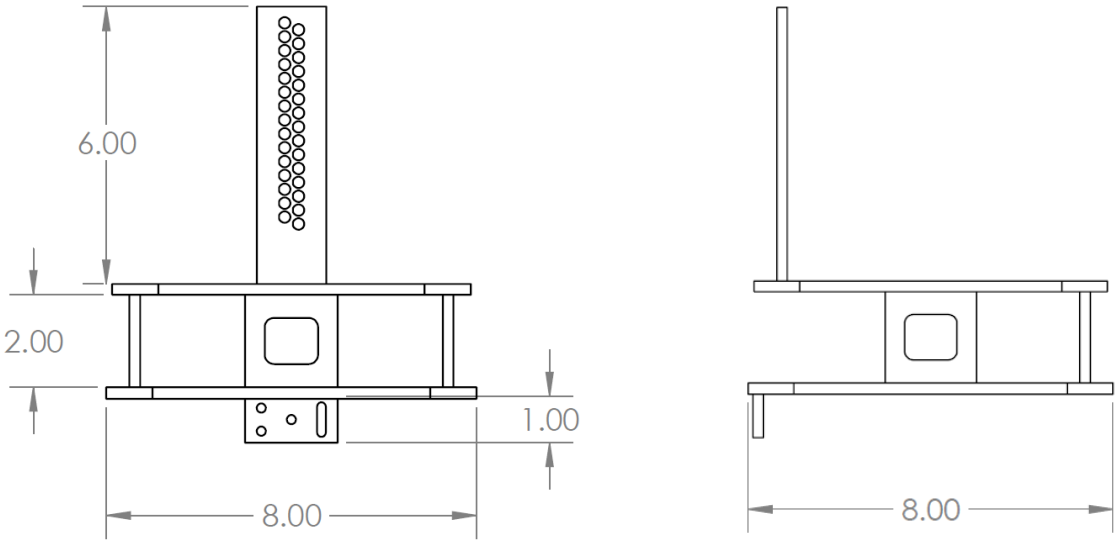
Frequency = f0 = 1.1/(R1 x C1)
(2k <= R1 <= 20k)

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

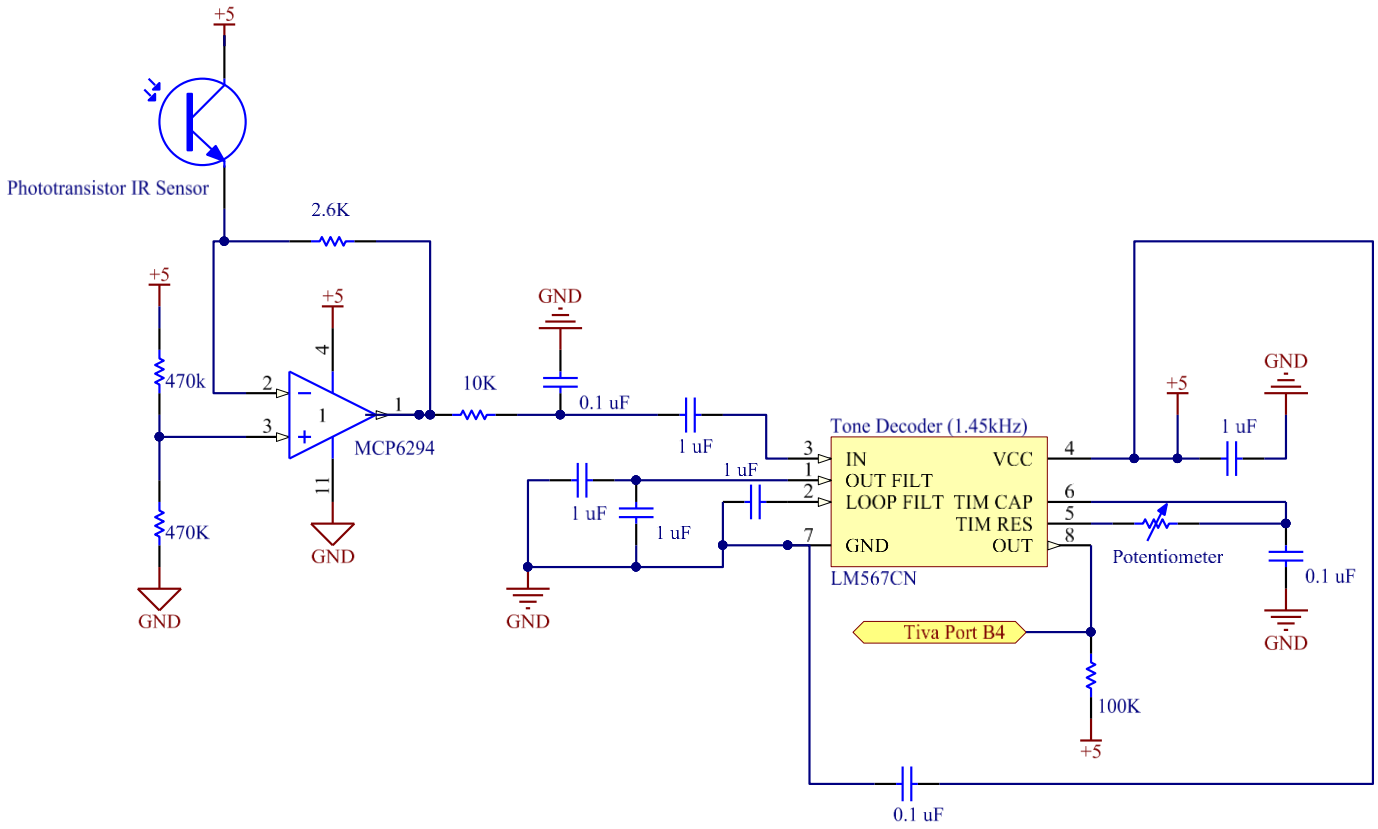
Mechanical Drawings



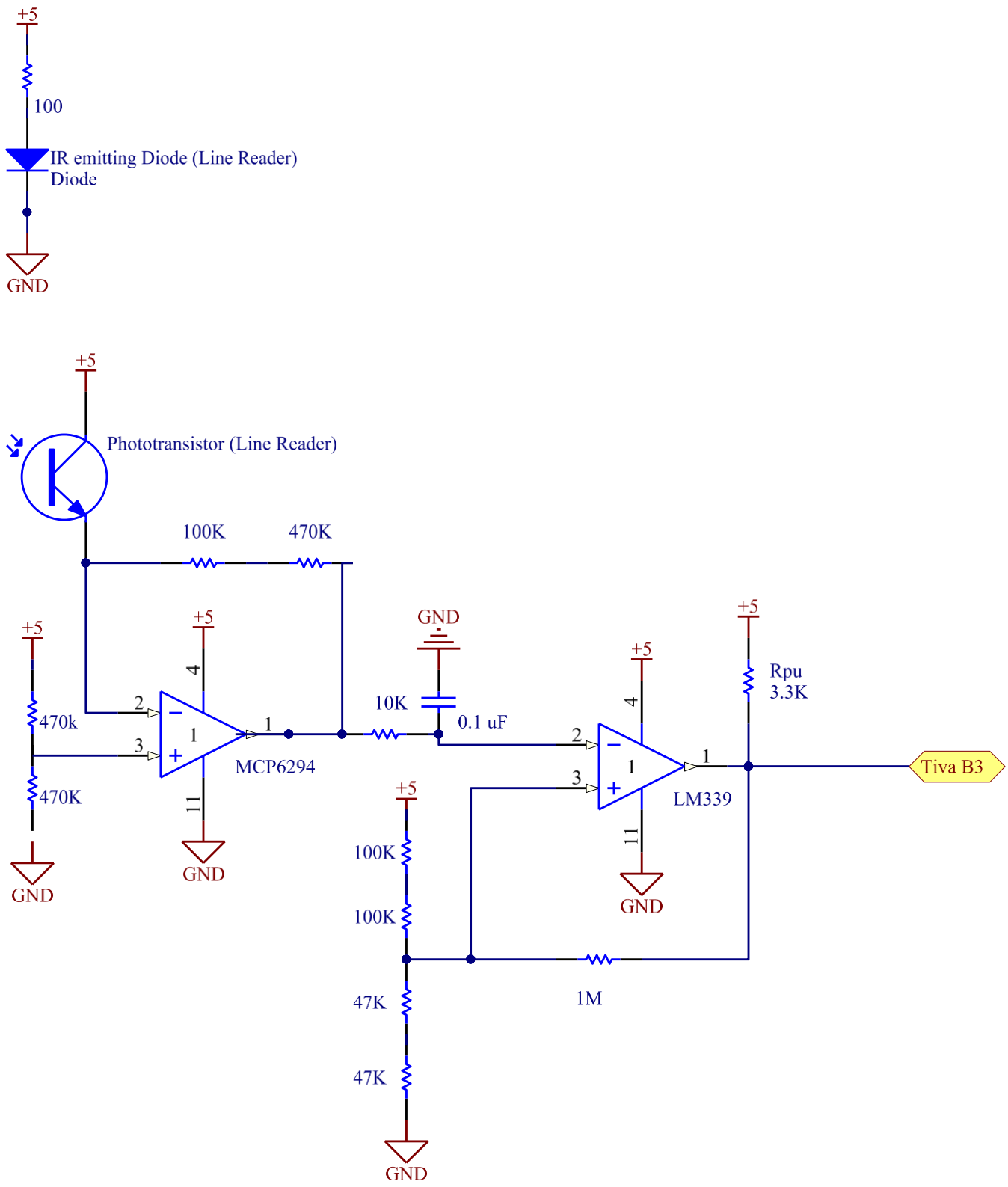


Electrical Design

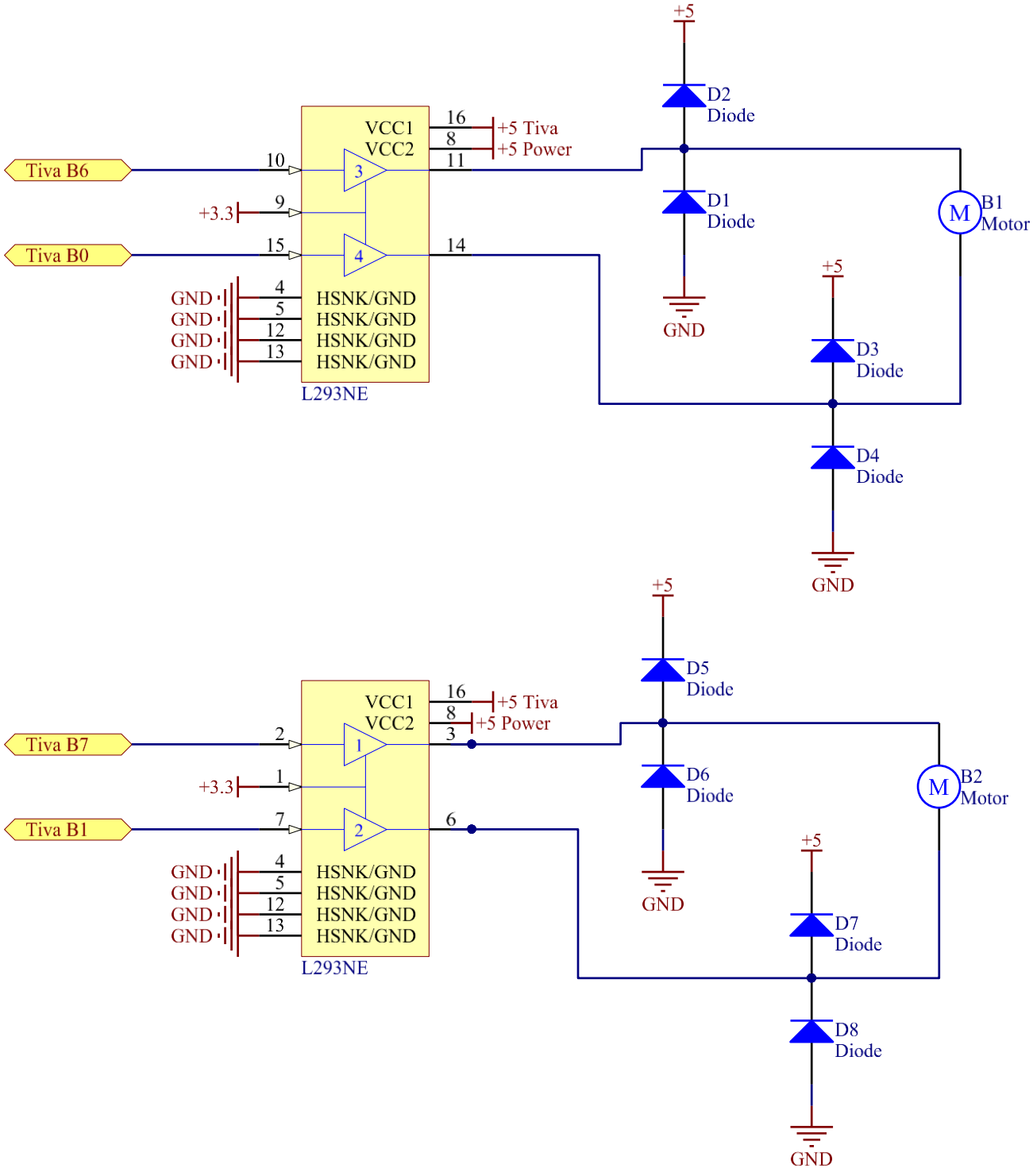
IR Sensor



Line Reader


+5
100
IR emitting Diode (Line Reader)
Diode
GND
+5
Phototransistor (Line Reader)
100K
470K
+5
470k
470K
GND
+5
4
2
3
1
11
MCP6294
GND
10K
GND
0.1 uF
+5
4
2
3
1
11
LM339
GND
+5
Rpu
3.3K
1
Tiva B3
+5
100K
100K
47K
47K
GND
1M

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
```