ME218 B

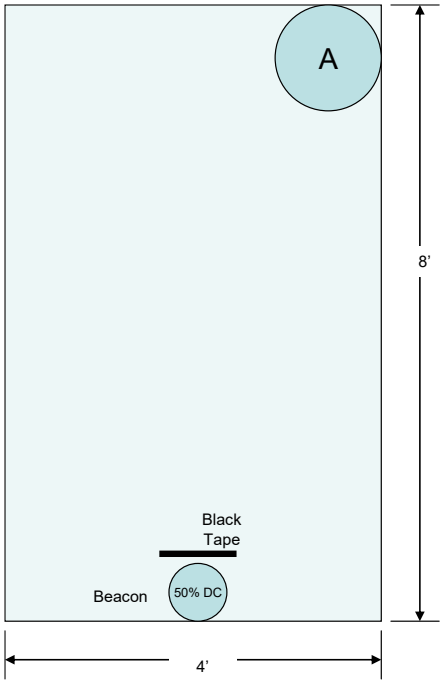
Laboratory Assignment Number 8 for ME 218b

Due by 5:00 pm on February 5, 2026

Part 1: A Simple Mobile Platform

Assignment:

You are to design and build a mobile platform that will be able to begin in a random orientation at position A. There will be a beacon placed approximately as shown in the diagram below. The beacon will pulse infrared light at a frequency of 1427Hz. Your droid should query the Command Generator over SPI (protocol and command descriptions in Appendix A) and follow the commands to position itself. Get a sign-off from a TA/Ed/lab coach after demonstrating your completed vehicle.



❑ 1.1) Your vehicle must be constructed from scratch using the motors and wheels provided and carry the PIC32, Command Generator and other electronics of your choosing. It should use a tether to supply power and may connect to the PC to display debugging messages. The motors and wheels may not be modified and must be returned in good working order at the end of the lab. Note1: the motor leads are somewhat fragile. Please be sure to strain-relieve them to prevent fatiguing the motor connections.

In the report:

The report should include a description of the mechanical hardware design, electrical hardware design (schematics and design calculations), along with state charts and pseudo-code for the software. Additionally, include a narrative description of the hardware and software (brief and to the point, max. 1 page each), and a Highlighted listing of the final software that was implemented. When copying your code into your report, please use the Highlight program to prepare an RTF file using the "bright" color theme. You can then open the RTF file in Word and copy and paste the code into your report and it will maintain the color highlighting. Look at the code that you paste into your report. If the indenting is messed up anywhere, it means that you had tab characters in your source files. If the indenting is messed up, clean this up before running Highlight again and submitting your report.

Schematics

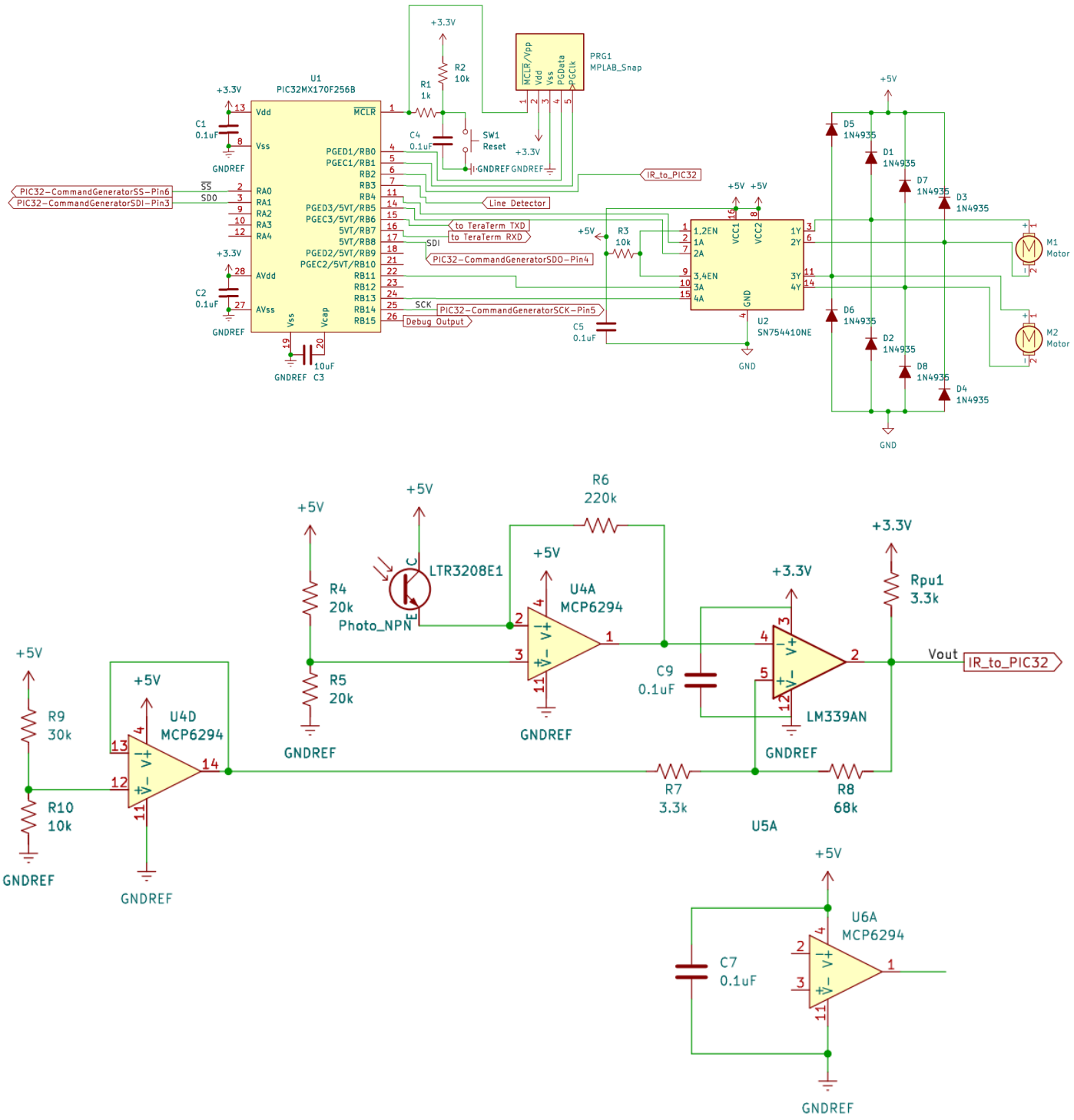

U1
PIC32MX170F256B
PRG1
MPLAB_Snap
U2
SN754410NE
U4A
MCP6294
U4D
MCP6294
U5A
LM339AN
U6A
MCP6294
LTR3208E1
Photo_NPN
SW1
Reset
M1
Motor
M2
Motor
D1 1N4935
D2 1N4935
D3 1N4935
D4 1N4935
D5 1N4935
D6 1N4935
D7 1N4935
D8 1N4935
R1 1k
R2 10k
R3 10k
R4 20k
R5 20k
R6 220k
R7 3.3k
R8 68k
R9 30k
R10 10k
Rpu1 3.3k
C1 0.1uF
C2 0.1uF
C3 10uF
C4 0.1uF
C5 0.1uF
C7 0.1uF
C9 0.1uF
+3.3V
+5V
GNDREF
GND
Vout
IR_to_PIC32
Line Detector
to TeraTerm TXD
to TeraTerm RXD
SDI
SCK
Debug Output
PIC32-CommandGeneratorSS-Pin6
PIC32-CommandGeneratorSDI-Pin3
PIC32-CommandGeneratorSDO-Pin4
PIC32-CommandGeneratorSCK-Pin5

Table of Pins

Motor Output (OC, PWM)

Controlled Motor	Pin# on PIC32	Port on PIC32	PWM Channel PIC32	Pin# on L293NE	Port on L293NE
M1+	11	RB4	1	2	1A
M1-	14	RB5	2	7	2A
M2+	18	RB9	3	10	3A
M2-	24	RB13	4	15	4A

Input/Output

Description	A/D	Pin# on PIC32	Port on PIC32	I/O
Phototransistor (IR)	Digital	6	RB2	Input
Line Detector	Digital	7	RB3	Input
Debug Output	Digital	26	RB15	Output

SPI Pins Configuration

Description	Pin# on PIC32	Port on PIC32
SS	2	RA0
SDO	3	RA1
SDI	17	RB8
SCK	25	RB14

Design Calculations

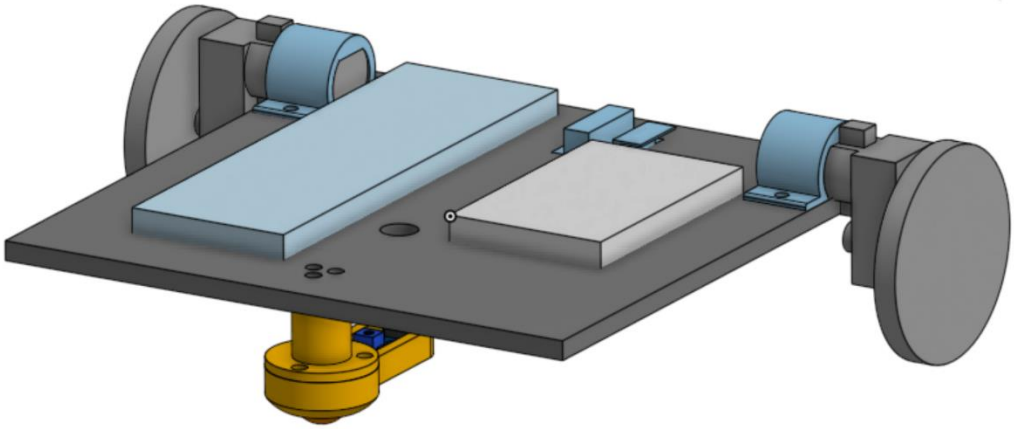
Choosing H-bridge - SN754410**:**

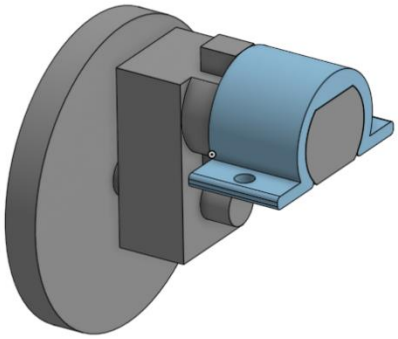
A coil of the DC motor typically has resistance 8.5Ω (by Digital Multimeter measurement), and draws approximately 588mA at 5V, steady state. If it draws as much as 588mA, the voltage will drop below 5V, to at least 3V, which corresponds to 353mA using 8.5Ω motor. Actually the voltage will be higher than 3V, and however the voltage drops the current can be safely supplied. Therefore, the SN754410-driven H-bridge can drive the coils.

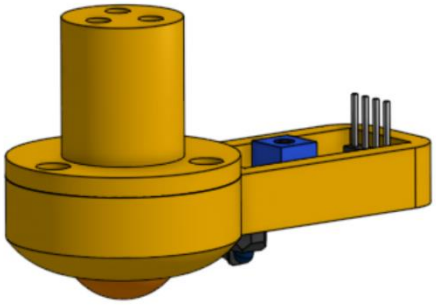
Resistor Value for Amplifying Circuit

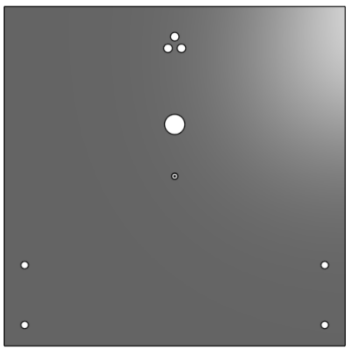
The reference voltage for the first-stage amplifier (using Op-Amp) is 2.5V, so the output of the first-stage amplifier is swinging from 2.5V to 0V. Then the second-stage thresholding (using comparator) uses the reference voltage of around 1.25V. Resistor values are found during field testing.

Mechanical Hardware Design









For Lab 8, we used onshape to design and create a virtual assembly of all our components. The main base for our mobile platform was laser cut out of ¼" thick birch plywood, with the remaining components fabricated using 3D printing. To make assembly easier, all of the hardware connections were standardized to use M4 screws. Starting with the motor clamps, these were 3D printed to roughly fit the shape of the provided motors and to clamp them down using the screws. Next, we also had to design a custom roller ball bearing caster for the front, so that the entire base can be supported. After finding a spare ball bearing, we designed the housing to not only accommodate it, but to also hold the TRFT5000 line sensor that we were using for line detection. Based on the datasheet, the maximal range of the sensor is around 15 mm, which is less than the height of our base; hence why we added the sensor to the front caster assembly. Finally, for the base laser cut drawing, we assembled the components together and edited the base in context to make the proper cutouts/holes for each component. Generally speaking, due to the temporary nature of this lab, miscellaneous components (breadboards, wires, etc.) were fastened to the top using tape.

Software Design

DCMotorService controls 2 DC motors, and takes ES_MOTOR_ACTION_CHANGE event as a trigger for changing PWM states for the DC motors. It takes the desired speed and direction for both motors from the variables stored using the function MotorCommandWrapper called by the caller service (mainly MainLogicFSM) and configure a drive-brake control by setting PWM with the method used in Lab 7.

Beacon detection is done by simply checking if there is a rising edge coming from the phototransistor module (IR input) using **EventCheckers**. If so, it will then post an ES_BEACON_DETECTED event to MainLogicFSM. This method works because no other IR poise is present on the field, and once there is strong enough IR signal, the program believes a beacon has been detected.

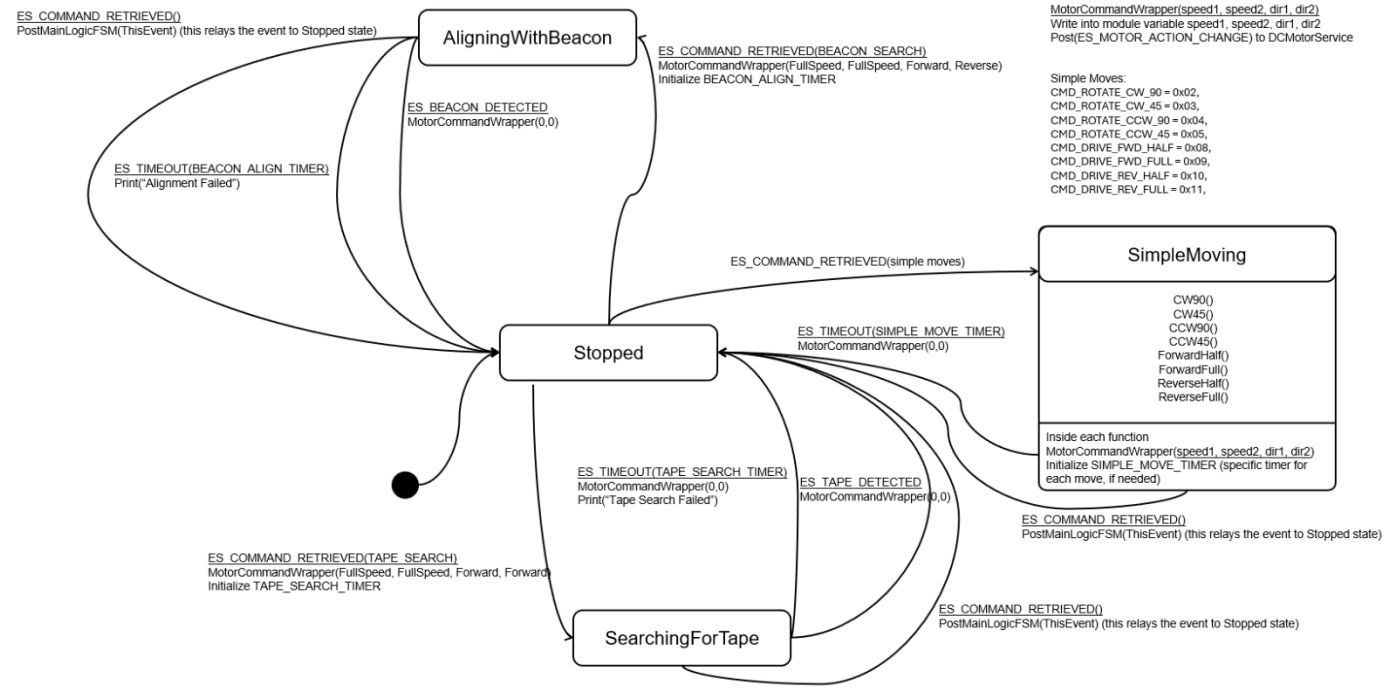
CommandRetrieveService is responsible for periodically polling the SPI-based CommandGenerator and notifying MainLogicFSM when a new command becomes available. This service uses a periodic timer (COMMAND_SPI_TIMER) to initiate polling at fixed intervals. On each timeout: If 0xFF is received, a flag (SawNewCommandFlag) is set. On the subsequent query, if a valid command byte is received: The byte is verified against a lookup table in CommonDefinitions. If valid and different from the previously processed command, an ES_COMMAND_RETRIEVED(commandByte) event is posted to MainLogicFSM. Repeated old command values are ignored to prevent duplicate event posting.

CommonDefinitions.c/CommonDefinitions.h centralizes all shared system constants, command encodings, hardware configuration parameters to ensure consistency and reduce duplication across all motor control services.

Ports.c centralizes all microcontroller pin configuration and low-level digital I/O access, ensuring that hardware-specific TRIS/ANSEL setup and signal reads are abstracted cleanly from higher-level services.

MainLogicFSM is a finite state machine that controls the logic of the program. Refer to the state diagram below:

State Diagram of **MainLogicFSM**:



Pseudocode

1). MainLogicFSM:

Description
Pseudocode for MainLogicService based on the state diagram.
Handles command-driven motion modes and timer-based completion.

Notes
State Summary
- Stopped: idle, waits for commands
- SimpleMoving: executes open-loop moves (rotations/drives)
- SearchingForTape: drive forward until tape detected or timeout
- AligningWithBeacon: spin to align to beacon or timeout

Event
ES_COMMAND_RETRIEVED(commandByte)
ES_TAPE_DETECTED
ES_BEACON_DETECTED
ES_TIMEOUT(SIMPLE_MOVE_TIMER)
ES_TIMEOUT(TAPE_SEARCH_TIMER)
ES_TIMEOUT(BEACON_ALIGN_TIMER)

Action
MotorCommandWrapper(speedLeft, speedRight, dirLeft, dirRight)
Initialize SIMPLE_MOVE_TIMER
Initialize TAPE_SEARCH_TIMER
Initialize BEACON_ALIGN_TIMER

State Chart
Init -> Stopped
Stopped --ES_COMMAND_RETRIEVED(simple moves)--> SimpleMoving
Stopped --ES_COMMAND_RETRIEVED(tape search)--> SearchingForTape
Stopped --ES_COMMAND_RETRIEVED(beacon align)--> AligningWithBeacon

SimpleMoving --ES_TIMEOUT(SIMPLE_MOVE_TIMER)--> Stopped
SearchingForTape --ES_TAPE_DETECTED--> Stopped
SearchingForTape --ES_TIMEOUT(TAPE_SEARCH_TIMER)--> Stopped
AligningWithBeacon --ES_BEACON_DETECTED--> Stopped
AligningWithBeacon --ES_TIMEOUT(BEACON_ALIGN_TIMER)--> Stopped

MainLogicService
InitMainLogicService
  Initialize ports via Ports.c/Ports.h
    InitBeaconInputPin()
    InitTapeSensorPin()
    InitCommandSPIPins()
  Set CurrentState = Stopped
  MotorCommandWrapper(0,0,FORWARD,FORWARD)

RunMainLogicService
  SWITCH(CurrentState)

  State: Stopped
    On ES_COMMAND_RETRIEVED(commandByte)
      CASE commandByte
        0x00: MotorCommandWrapper(0, 0, FORWARD, FORWARD)
        0x02: RotateCW90(); CurrentState = SimpleMoving

0x03: RotateCW45(); CurrentState = SimpleMoving
0x04: RotateCCW90(); CurrentState = SimpleMoving
0x05: RotateCCW45(); CurrentState = SimpleMoving
0x08: DriveForwardHalf(); CurrentState = SimpleMoving
0x09: DriveForwardFull(); CurrentState = SimpleMoving
0x10: DriveReverseHalf(); CurrentState = SimpleMoving
0x11: DriveReverseFull(); CurrentState = SimpleMoving
0x20: AlignWithBeacon(); CurrentState = AligningWithBeacon
0x40: SearchForTape(); CurrentState = SearchingForTape
END CASE

State: SimpleMoving
On ES_TIMEOUT(SIMPLE_MOVE_TIMER)
MotorCommandWrapper(0,0,FORWARD,FORWARD)
CurrentState = Stopped
On ES_COMMAND_RETRIEVED(0x00)
MotorCommandWrapper(0,0,FORWARD,FORWARD)
PostMainLogicFSM(ThisEvent);
CurrentState = Stopped;

State: SearchingForTape
On ES_TAPE_DETECTED
MotorCommandWrapper(0,0,FORWARD,FORWARD)
CurrentState = Stopped
On ES_TIMEOUT(TAPE_SEARCH_TIMER)
MotorCommandWrapper(0,0,FORWARD,FORWARD)
Print("Tape Search Failed")
CurrentState = Stopped
On ES_COMMAND_RETRIEVED(0x00)
MotorCommandWrapper(0,0,FORWARD,FORWARD)
PostMainLogicFSM(ThisEvent);
CurrentState = Stopped

State: AligningWithBeacon
On ES_BEACON_DETECTED
MotorCommandWrapper(0,0,FORWARD,FORWARD)
CurrentState = Stopped
On ES_TIMEOUT(BEACON_ALIGN_TIMER)
MotorCommandWrapper(0,0,FORWARD,FORWARD)
Print("Alignment Failed")
CurrentState = Stopped
On ES_COMMAND_RETRIEVED(0x00)
MotorCommandWrapper(0,0,FORWARD,FORWARD)
PostMainLogicFSM(ThisEvent);
CurrentState = Stopped

SimpleMoves helpers (open-loop)
RotateCW90()
MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, REVERSE)
Initialize SIMPLE_MOVE_TIMER to 6000 ms

RotateCW45()
MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, REVERSE)
Initialize SIMPLE_MOVE_TIMER to 3000 ms

RotateCCW90()
MotorCommandWrapper(FULL_SPEED, FULL_SPEED, REVERSE, FORWARD)
Initialize SIMPLE_MOVE_TIMER to 6000 ms

```
RotateCCW45()
  MotorCommandWrapper(FULL_SPEED, FULL_SPEED, REVERSE, FORWARD)
  Initialize SIMPLE_MOVE_TIMER to 3000 ms

DriveForwardHalf()
  MotorCommandWrapper(HALF_SPEED, HALF_SPEED, FORWARD, FORWARD)
  Initialize SIMPLE_MOVE_TIMER to 1000 ms (optional: remove for continuous)

DriveForwardFull()
  MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, FORWARD)
  Initialize SIMPLE_MOVE_TIMER to 1000 ms (optional: remove for continuous)

DriveReverseHalf()
  MotorCommandWrapper(HALF_SPEED, HALF_SPEED, REVERSE, REVERSE)
  Initialize SIMPLE_MOVE_TIMER to 1000 ms (optional: remove for continuous)

DriveReverseFull()
  MotorCommandWrapper(FULL_SPEED, FULL_SPEED, REVERSE, REVERSE)
  Initialize SIMPLE_MOVE_TIMER to 1000 ms (optional: remove for continuous)

SearchForTape()
  MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, FORWARD)
  Initialize TAPE_SEARCH_TIMER

AlignWithBeacon()
  MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, REVERSE)
  Initialize BEACON_ALIGN_TIMER
```

2). Event Checker for beacon and tape:

```
Description
  Pseudocode for event checkers related to beacon, tape.

Notes
  EventCheckers should detect transitions and post ES_* events to services.
  Use static "last" variables to avoid repeated events while input stays low.

  Event
  ES_BEACON_DETECTED
  ES_TAPE_DETECTED

  Beacon EventChecker
  Check for IR input I/O line (digital input) (initialized in Ports.c/Ports.h)
  IF input transitions from LOW to HIGH
    Post ES_BEACON_DETECTED to main logic

  Tape EventChecker
  Check for tape sensor input (digital or analog threshold) (initialized in Ports.c/Ports.h)
  IF input transitions from LOW to HIGH
    Post ES_TAPE_DETECTED to main logic
```

3). Receiveing command from command generator:

```
Description
```

Pseudocode for SPI command retrieval and posting command events.

```
Event
    ES_COMMAND_RETRIEVED(commandByte)

    InitCommandRetrieveService
      Initialize flag to track when new command is ready
      Configure SPI hardware as leader with appropriate timing settings
      Set up chip select, data input, and data output pins
      Enable SPI communication
      Start timer to poll for commands every 2 seconds
      Post initialization event

    RunCommandRetrieveService
      When timer expires:
        Query the command generator for a new byte
        If received 0xFF (new command ready flag):
          Remember that next byte will be the new command
        Otherwise if we previously saw the ready flag:
          If the byte is a valid command:
            If it differs from the last command posted:
              Post the new command to the main logic state machine
              Remember this command
          Otherwise:
            Log that an invalid command was received
          Clear the new command ready flag
        Restart the polling timer

    QueryCommandGenerator
      Send a dummy byte to clock out a response from the follower
      Read and return the response byte

    IsValidCommandByte
      Check if the byte matches any entry in the valid commands lookup table
      Return true if valid, false otherwise
```

4). Motor driving service:

Description

Pseudocode for controlling two DC motors (left/right) in one service.
Uses array-based motor state for clean, symmetric updates.

Notes

Guiding recommendation:
- Use ONE DCMotorService that owns both motors.
- Represent motors as an array of structs (index 0 = left, index 1 = right)
  so the same code path configures/updates either motor.
- Provide a single wrapper API (MotorCommandWrapper) that sets both motors
  and posts ES_MOTOR_ACTION_CHANGE to DCMotorService.

Data structures (module-level in DCMotorService):

```
MotorState {
  speedTicks
  direction
  pwmOcModule
  forwardPin
  reversePin
}
```

```
MotorState Motors[2]  // Motors[LEFT], Motors[RIGHT]

Event
ES_MOTOR_ACTION_CHANGE(desiredSpeed)

InitDCMotorService
  Set MyPriority
  Initialize DesiredSpeed[LEFT_MOTOR] = 0, DesiredSpeed[RIGHT_MOTOR] = 0
  Initialize DesiredDirection[LEFT_MOTOR] = FORWARD, DesiredDirection[RIGHT_MOTOR] = FORWARD
  Call ConfigureDCMotorPins():
    Configure RB4, RB5 (left motor), RB11, RB13 (right motor) as digital outputs
    Initialize all motor pins to LOW
    Map OC1 output to RB4 (left motor PWM)
    Map OC2 output to RB11 (right motor PWM)
  Call ConfigurePWM():
    Configure Timer2 as PWM time base with PRESCALE_2
    Set PR2 = PWM_PERIOD_TICKS
    Configure OC1 and OC2 for PWM mode (OCM = 0b110)
    Set initial duty cycle OC1RS = OC2RS = 0
    Enable OC1 and OC2 modules
    Start Timer2
  Post ES_INIT event

MotorCommandWrapper(speedLeft, speedRight, dirLeft, dirRight)
  DesiredSpeed[LEFT_MOTOR] = speedLeft
  DesiredSpeed[RIGHT_MOTOR] = speedRight
  DesiredDirection[LEFT_MOTOR] = dirLeft
  DesiredDirection[RIGHT_MOTOR] = dirRight
  Post ES_MOTOR_ACTION_CHANGE event to DCMotorService

RunDCMotorService
  On ES_INIT:
    No action needed (initialization done in Init function)
  On ES_MOTOR_ACTION_CHANGE:
    dutyCycle = MapSpeedToDutyCycle(DesiredSpeed[LEFT_MOTOR])
    IF DesiredDirection[LEFT_MOTOR] == FORWARD
      MOTOR_REVERSE_PIN_L = 0
      OC1RS = dutyCycle
    ELSE (REVERSE)
      MOTOR_REVERSE_PIN_L = 1
      OC1RS = PWM_PERIOD_TICKS - dutyCycle + 1
    IF DesiredDirection[RIGHT_MOTOR] == FORWARD
      MOTOR_REVERSE_PIN_R = 0
      OC2RS = dutyCycle
    ELSE (REVERSE)
      MOTOR_REVERSE_PIN_R = 1
      OC2RS = PWM_PERIOD_TICKS - dutyCycle + 1

MapSpeedToDutyCycle(desiredSpeed)
  dutyCycle = desiredSpeed (direct mapping)
  Clamp dutyCycle between DUTY_MIN_TICKS and DUTY_MAX_TICKS
  RETURN dutyCycle
```

Source Code

1). MainLogicFSM:

```
/*******************************************************************************
Module
  MainLogicFSM.c

Revision
  0.1

Description
  Main logic state machine for command-driven robot behavior.

Notes
  States:
    - Stopped
    - SimpleMoving
    - SearchingForTape
    - AligningWithBeacon

History
When         Who    What/Why
------------ ---    --------
02/03/26     Tianyu  Initial creation for Lab 8 main logic
*******************************************************************************/

/*---------------------------- Include Files ----------------------------*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_Timers.h"
#include "MainLogicFSM.h"
#include "DCMotorService.h"
#include "CommonDefinitions.h"
#include "dbprintf.h"
#include "Ports.h"
#include "dbprintf.h"

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

/*---------------------------- Module Functions ----------------------------*/
static void RotateCW90(void);
static void RotateCW45(void);
static void RotateCCW90(void);
static void RotateCCW45(void);
static void DriveForwardHalf(void);
static void DriveForwardFull(void);
static void DriveReverseHalf(void);
static void DriveReverseFull(void);
static void SearchForTape(void);
static void AlignWithBeacon(void);

/*---------------------------- Module Variables ----------------------------*/
static MainLogicState_t CurrentState;
static uint8_t MyPriority;

/*---------------------------- Module Code ----------------------------*/
/*******************************************************************************
Function
   InitMainLogicFSM

Parameters
   uint8_t : the priority of this service

Returns
   bool, false if error in initialization, true otherwise

Description
   Initializes the main logic state machine.
```

```
Author
   Tianyu, 02/03/26
****************************************************************************/
bool InitMainLogicFSM(uint8_t Priority)
{
 ES_Event_t ThisEvent;

 MyPriority = Priority;

 /****************************************
  Initialization code for ports and sensors
  ****************************************/
 // TODO: Initialize ports via Ports.c/Ports.h
 InitBeaconInputPin();
 InitTapeSensorPin();
 InitCommandSPIPins();
 InitDebugOutputPin();

 CurrentState = Stopped;

 // Stop motors on startup
 MotorCommandWrapper(0, 0, FORWARD, FORWARD);

 ThisEvent.EventType = ES_INIT;
 if (ES_PostToService(MyPriority, ThisEvent) == true)
 {
  return true;
 }
 return false;
}

/****************************************************************************
Function
   PostMainLogicFSM

Parameters
   ES_Event_t 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

Author
   Tianyu, 02/03/26
****************************************************************************/
bool PostMainLogicFSM(ES_Event_t ThisEvent)
{
 return ES_PostToService(MyPriority, ThisEvent);
}

/****************************************************************************
Function
  RunMainLogicFSM

Parameters
  ES_Event_t : the event to process

Returns
  ES_Event_t, ES_NO_EVENT if no error ES_ERROR otherwise

Description
  State machine for command-driven robot behavior.

Author
   Tianyu, 02/03/26
****************************************************************************/
ES_Event_t RunMainLogicFSM(ES_Event_t ThisEvent)
```

```
{
 ES_Event_t ReturnEvent;
 ReturnEvent.EventType = ES_NO_EVENT;

//  DB_printf("Current State is %d \r\n", CurrentState);

 switch (CurrentState)
 {
  case Stopped:
   if (ThisEvent.EventType == ES_COMMAND_RETRIEVED)
   {
    switch (ThisEvent.EventParam)
    {
     case CMD_STOP:
      MotorCommandWrapper(0, 0, FORWARD, FORWARD);
      break;
     case CMD_ROTATE_CW_90:
      DB_printf("State: Rotating CW 90 deg\r\n");

      RotateCW90();
      CurrentState = SimpleMoving;
      break;
     case CMD_ROTATE_CW_45:
      DB_printf("State:  Rotating CW 45 deg\r\n");

      RotateCW45();
      CurrentState = SimpleMoving;
      break;
     case CMD_ROTATE_CCW_90:
       DB_printf("State: Rotating CCW 90 deg\r\n");
      RotateCCW90();
      CurrentState = SimpleMoving;
      break;
     case CMD_ROTATE_CCW_45:
       DB_printf("State: Rotating CCW 45 deg\r\n");
      RotateCCW45();
      CurrentState = SimpleMoving;
      break;
     case CMD_DRIVE_FWD_HALF:
       DB_printf("State: drive forwards half speed\r\n");
      DriveForwardHalf();
      CurrentState = SimpleMoving;
      break;
     case CMD_DRIVE_FWD_FULL:
       DB_printf("State: drive forwards full speed\r\n");
      DriveForwardFull();
      CurrentState = SimpleMoving;
      break;
     case CMD_DRIVE_REV_HALF:
       DB_printf("State: drive reverse half speed\r\n");
      DriveReverseHalf();
      CurrentState = SimpleMoving;
      break;
     case CMD_DRIVE_REV_FULL:
       DB_printf("State: drive reverse full speed\r\n");
      DriveReverseFull();
      CurrentState = SimpleMoving;
      break;
     case CMD_ALIGN_BEACON:
       DB_printf("State: aligning with beacon\r\n");
      // If already HIGH, the ES_BEACON_DETECTED event will be posted immediately
      if( ReadBeaconInputPin() == true ) {
       ES_Event_t BeaconEvent;
       BeaconEvent.EventType = ES_BEACON_DETECTED;
       BeaconEvent.EventParam = 0;
       PostMainLogicFSM(BeaconEvent);
      }
      else{
       // If not detected, act to look for beacon signal
       AlignWithBeacon();
      }
      CurrentState = AligningWithBeacon;
```

```
        break;
      case CMD_SEARCH_TAPE:
        DB_printf("State: searching for tape \r\n");
       // If already HIGH, the ES_TAPE_DETECTED event will be posted immediately
       if( ReadTapeSensorPin() == true ) {
        ES_Event_t TapeEvent;
        TapeEvent.EventType = ES_TAPE_DETECTED;
        TapeEvent.EventParam = 0;
        PostMainLogicFSM(TapeEvent);
       }
       else{
       // If not detected, act to look for line detect signal
        SearchForTape();
       }
       CurrentState = SearchingForTape;
       break;
      default:
       break;
    }
   }
   break;

  case SimpleMoving:
   if (ThisEvent.EventType == ES_TIMEOUT &&
      ThisEvent.EventParam == SIMPLE_MOVE_TIMER) // movement timer expired after a set amount of time
   {
      DB_printf("Motor Timeout Received while moving\r\n");
    MotorCommandWrapper(0, 0, FORWARD, FORWARD);
    CurrentState = Stopped;
   }
   else if (ThisEvent.EventType == ES_COMMAND_RETRIEVED) // while simple moving, new command received
   {
    DB_printf("New command received while moving\r\n");
    CurrentState = Stopped;
    PostMainLogicFSM(ThisEvent); //go back to stopped list to take action on new command
   }
   break;

  case SearchingForTape:
   if (ThisEvent.EventType == ES_TAPE_DETECTED) // detected tape
   {
      DB_printf("Tape detected\r\n");
    MotorCommandWrapper(0, 0, FORWARD, FORWARD);
    CurrentState = Stopped;
   }
   else if (ThisEvent.EventType == ES_TIMEOUT &&
         ThisEvent.EventParam == TAPE_SEARCH_TIMER) // stop looking for tape after set time
   {
    MotorCommandWrapper(0, 0, FORWARD, FORWARD);
    DB_printf("Tape Search Failed: Timeout");
    CurrentState = Stopped;
   }
   else if (ThisEvent.EventType == ES_COMMAND_RETRIEVED)    // new command received while searching for tape
   {
    DB_printf("New command received while searching for tape\r\n");
    CurrentState = Stopped;
    PostMainLogicFSM(ThisEvent);
   }
   break;

  case AligningWithBeacon:
   if (ThisEvent.EventType == ES_BEACON_DETECTED) // found direction of beacon
   {
    DB_printf("Found beacon\r\n");
    MotorCommandWrapper(0, 0, FORWARD, FORWARD); // change speed
    CurrentState = Stopped;
   }
   else if (ThisEvent.EventType == ES_TIMEOUT &&
         ThisEvent.EventParam == BEACON_ALIGN_TIMER) // set time passed, stop aligning towards beacon
   {
    MotorCommandWrapper(0, 0, FORWARD, FORWARD);
    DB_printf("Beacon Search Failed: Timeout");
```

```
      CurrentState = Stopped;
    }
    else if (ThisEvent.EventType == ES_COMMAND_RETRIEVED) // new command received while aligning for beacon
    {
      DB_printf("New command received while aligning with beacon\r\n");
      CurrentState = Stopped;
      PostMainLogicFSM(ThisEvent);
    }
    break;

  default:
    break;
 }

 return ReturnEvent;
}

/****************************************************************************
Function
   QueryMainLogicFSM

Parameters
   None

Returns
   MainLogicState_t: the current state of the main logic FSM

Description
   Returns the current state of the main logic FSM

Author
   Tianyu, 02/03/26
****************************************************************************/
MainLogicState_t QueryMainLogicFSM(void)
{
 return CurrentState;
}

/*---------------------------- Helper Functions ---------------------------*/
/****************************************************************************
Function
   RotateCW90

Parameters
   None

Returns
   None

Description
   Open-loop 90 degree clockwise rotation.

Author
   Tianyu, 02/03/26
****************************************************************************/
static void RotateCW90(void)
{
 // Pseudocode:
 // MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, REVERSE)
 // Initialize SIMPLE_MOVE_TIMER to 6000 ms
 MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, REVERSE);
 ES_Timer_InitTimer(SIMPLE_MOVE_TIMER, SIMPLE_MOVE_90_MS);
}

/****************************************************************************
Function
   RotateCW45

Parameters
```

```
   None

Returns
   None

Description
   Open-loop 45 degree clockwise rotation.

Author
   Tianyu, 02/03/26
****************************************************************************/
static void RotateCW45(void)
{
 // Pseudocode:
 // MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, REVERSE)
 // Initialize SIMPLE_MOVE_TIMER to 3000 ms
 MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, REVERSE);
 ES_Timer_InitTimer(SIMPLE_MOVE_TIMER, SIMPLE_MOVE_45_MS);
}

/****************************************************************************
Function
   RotateCCW90

Parameters
   None

Returns
   None

Description
   Open-loop 90 degree counter-clockwise rotation.

Author
   Tianyu, 02/03/26
****************************************************************************/
static void RotateCCW90(void)
{
 // Pseudocode:
 // MotorCommandWrapper(FULL_SPEED, FULL_SPEED, REVERSE, FORWARD)
 // Initialize SIMPLE_MOVE_TIMER to 6000 ms
 MotorCommandWrapper(FULL_SPEED, FULL_SPEED, REVERSE, FORWARD);
 ES_Timer_InitTimer(SIMPLE_MOVE_TIMER, SIMPLE_MOVE_90_MS);
}

/****************************************************************************
Function
   RotateCCW45

Parameters
   None

Returns
   None

Description
   Open-loop 45 degree counter-clockwise rotation.

Author
   Tianyu, 02/03/26
****************************************************************************/
static void RotateCCW45(void)
{
 // Pseudocode:
 // MotorCommandWrapper(FULL_SPEED, FULL_SPEED, REVERSE, FORWARD)
 // Initialize SIMPLE_MOVE_TIMER to 3000 ms
 MotorCommandWrapper(FULL_SPEED, FULL_SPEED, REVERSE, FORWARD);
 ES_Timer_InitTimer(SIMPLE_MOVE_TIMER, SIMPLE_MOVE_45_MS);
}
```

```
/*****************************************************************************
Function
   DriveForwardHalf

Parameters
   None

Returns
   None

Description
   Drive forward at half speed (open-loop).

Author
   Tianyu, 02/03/26
*****************************************************************************/
static void DriveForwardHalf(void)
{
 // Pseudocode:
 // MotorCommandWrapper(HALF_SPEED, HALF_SPEED, FORWARD, FORWARD)
 // Optionally set SIMPLE_MOVE_TIMER
 MotorCommandWrapper(HALF_SPEED, HALF_SPEED, FORWARD, FORWARD);
}

/*****************************************************************************
Function
   DriveForwardFull

Parameters
   None

Returns
   None

Description
   Drive forward at full speed (open-loop).

Author
   Tianyu, 02/03/26
*****************************************************************************/
static void DriveForwardFull(void)
{
 // Pseudocode:
 // MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, FORWARD)
 // Optionally set SIMPLE_MOVE_TIMER
 MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, FORWARD);
}

/*****************************************************************************
Function
   DriveReverseHalf

Parameters
   None

Returns
   None

Description
   Drive reverse at half speed (open-loop).

Author
   Tianyu, 02/03/26
*****************************************************************************/
static void DriveReverseHalf(void)
{
 // Pseudocode:
 // MotorCommandWrapper(HALF_SPEED, HALF_SPEED, REVERSE, REVERSE)
```

```
 // Optionally set SIMPLE_MOVE_TIMER
 MotorCommandWrapper(HALF_SPEED, HALF_SPEED, REVERSE, REVERSE);
}

/*****************************************************************************
Function
   DriveReverseFull

Parameters
   None

Returns
   None

Description
   Drive reverse at full speed (open-loop).

Author
   Tianyu, 02/03/26
*****************************************************************************/
static void DriveReverseFull(void)
{
 // Pseudocode:
 // MotorCommandWrapper(FULL_SPEED, FULL_SPEED, REVERSE, REVERSE)
 // Optionally set SIMPLE_MOVE_TIMER
 MotorCommandWrapper(FULL_SPEED, FULL_SPEED, REVERSE, REVERSE);
}

/*****************************************************************************
Function
   SearchForTape

Parameters
   None

Returns
   None

Description
   Drive forward until tape detected or timeout.

Author
   Tianyu, 02/03/26
*****************************************************************************/
static void SearchForTape(void)
{
 // Pseudocode:
 // MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, FORWARD)
 // Initialize TAPE_SEARCH_TIMER
 MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, FORWARD);
//  ES_Timer_InitTimer(TAPE_SEARCH_TIMER, TAPE_SEARCH_MS);
}

/*****************************************************************************
Function
   AlignWithBeacon

Parameters
   None

Returns
   None

Description
   Spin until beacon detected or timeout.

Author
   Tianyu, 02/03/26
*****************************************************************************/
```

```
static void AlignWithBeacon(void)
{
 // Pseudocode:
 // MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, REVERSE)
 // Initialize BEACON_ALIGN_TIMER
 MotorCommandWrapper(FULL_SPEED, FULL_SPEED, FORWARD, REVERSE);
 ES_Timer_InitTimer(BEACON_ALIGN_TIMER, BEACON_ALIGN_MS);
}
```

2).Event Checker for beacon and tape detect:

```
/*******************************************************************************
Module
 EventCheckers.c

Revision
 0.1

Description
 Event checkers for beacon, tape, and keyboard input.

Notes
 Use static variables to detect transitions.

History
When         Who    What/Why
-------------- ---    --------
02/03/26     Tianyu  Initial creation for Lab 8 event checkers
*******************************************************************************/

/*---------------------------- Include Files ----------------------------*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_Events.h"
#include "ES_PostList.h"
#include "ES_ServiceHeaders.h"
#include "ES_Port.h"
#include "EventCheckers.h"
#include "CommonDefinitions.h"
#include "dbprintf.h"
#include "Ports.h"

/*---------------------------- Module Code ----------------------------*/
/*******************************************************************************
Function
 Check4Keystroke

Parameters
 None

Returns
 bool: true if a new key was detected & posted

Description
 checks to see if a new key from the keyboard is detected and, if so,
 retrieves the key and posts an ES_NEW_KEY event to TestHarnessService0

Notes
 The functions that actually check the serial hardware for characters
 and retrieve them are assumed to be in ES_Port.c

Author
 J. Edward Carryer, 08/06/13, 13:48
*******************************************************************************/
bool Check4Keystroke(void)
{
 if (IsNewKeyReady())   // new key waiting?
 {
   ES_Event_t ThisEvent;
   ThisEvent.EventType   = ES_NEW_KEY;
   ThisEvent.EventParam  = GetNewKey();
   ES_PostAll(ThisEvent);
   return true;
 }
 return false;
}

/*******************************************************************************
Function
 Check4TapeDetected
```

```
Parameters
  None

Returns
  bool: true if a new tape event was detected & posted

Description
  Checks for a low-going transition on the tape sensor input.

Author
  Tianyu, 02/03/26
****************************************************************************/
bool Check4TapeDetected(void)
{
 static bool LastTapeState = true;
 bool CurrentTapeState = ReadTapeSensorPin();

 if ((CurrentTapeState == false) && (LastTapeState == true))
 {
   ES_Event_t ThisEvent;
   ThisEvent.EventType = ES_TAPE_DETECTED;
   ThisEvent.EventParam = 0;
   PostMainLogicFSM(ThisEvent);
   LastTapeState = CurrentTapeState;
   return true;
 }

 LastTapeState = CurrentTapeState;
 return false;
}

/****************************************************************************
Function
  Check4CommandAvailable

Parameters
  None

Returns
  bool: true if a new command event was detected & posted

Description
  Placeholder event checker. Command retrieval is handled by the
  CommandRetrieveService (SPI polling/interrupts).

Author
  Tianyu, 02/03/26
****************************************************************************/
bool Check4CommandAvailable(void)
{
 // TODO: If moving command retrieval into an event checker, implement here.
 return false;
}

/****************************************************************************
Function
  Check4BeaconDetected

Parameters
  None

Returns
  bool: true if IR input is HIGH

Description
  Beaon event checker. The IR beacon sensor is active HIGH.

Author
```

```
  Tianyu, 02/04/26
****************************************************************************/
bool Check4BeaconDetected(void)
{
 static bool LastBeaconState = false;
 bool CurrentBeaconState = ReadBeaconInputPin();

 if ((CurrentBeaconState == true) && (LastBeaconState == false))
 {
   DEBUG_OUTPUT_PIN_LAT = 1;
   ES_Event_t ThisEvent;
   ThisEvent.EventType = ES_BEACON_DETECTED;
   ThisEvent.EventParam = 0;
   PostMainLogicFSM(ThisEvent);
   LastBeaconState = CurrentBeaconState;
//    printf("Posting ES_BEACON_DETECTED event.\r\n");
   DEBUG_OUTPUT_PIN_LAT = 0;
   return true;
 }

 LastBeaconState = CurrentBeaconState;
 return false;
}
```

3). Receiveing command from command generator:

```
/*******************************************************************************
Module
 CommandRetrieveService.c

Revision
 0.1

Description
 Service for polling SPI and posting ES_COMMAND_RETRIEVED events when a new
 command is available from the CommandGenerator.

Notes
 CommandGenerator is an SPI follower and SPI32 is the leader.
 Query behavior:
  - When a new command is ready, the next query returns 0xFF.
  - The query following that 0xFF returns the new command byte.
  - Subsequent queries return the same command byte until a new command arrives.

 This service should post ES_COMMAND_RETRIEVED(commandByte) to MainLogicService
 when a valid command byte is received after a 0xFF flag.

History
When         Who    What/Why
-------------- ---    --------
02/03/26     Tianyu  Initial creation for Lab 8 command retrieval
*******************************************************************************/

/*---------------------------- Include Files ----------------------------*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_Timers.h"
#include "CommandRetrieveService.h"
#include "CommonDefinitions.h"
#include "PIC32_SPI_HAL.h"
#include "MainLogicFSM.h"
#include "dbprintf.h"
#include <xc.h>
#include <sys/attribs.h>


/*---------------------------- Module Defines ----------------------------*/
#define SPI_POLL_INTERVAL_MS 2000
SPI_Module_t Module = SPI_SPI1;

/*---------------------------- Module Functions ----------------------------*/
static uint8_t ReadSPICommandByte(void);
static bool IsValidCommandByte(uint8_t commandByte);

/*---------------------------- Module Variables ----------------------------*/
static uint8_t MyPriority;
static bool SawNewCommandFlag;
static uint8_t LastCommand;

uint8_t QueryCommandGenerator(void);

/*---------------------------- Module Code ----------------------------*/
/*
void __ISR(_SPI_1_VECTOR, IPL4SOFT) SPI1Handler(void){
  uint8_t data = 0x0;
  data = (uint8_t) SPIOperate_ReadData(Module);
  IEC1CLR = _IEC1_SPI1RXIE_MASK | _IEC1_SPI1TXIE_MASK | _IEC1_SPI1EIE_MASK;

  CurrentCommand = data;
}
* */
/*******************************************************************************
Function
   InitCommandRetrieveService
```

```
Parameters
   uint8_t : the priority of this service

Returns
   bool, false if error in initialization, true otherwise

Description
   Initializes the SPI command retrieval service.

Author
   Tianyu, 02/03/26
****************************************************************************/
bool InitCommandRetrieveService(uint8_t Priority)
{
 ES_Event_t ThisEvent;

 MyPriority = Priority;
 SawNewCommandFlag = false;

 /*****************************************
  Initialization code for SPI command retrieval
  *****************************************/
 // TODO: Configure SPI32 as leader (mode, clock, chip select)
 // TODO: Initialize Ports/SPI pins via Ports.c if needed

  SPI_SamplePhase_t SamplePhase = SPI_SMP_MID;
  uint32_t DesiredClock_ns = 10000;
  SPI_Clock_t ClockIdle = SPI_CLK_HI;
  SPI_ActiveEdge_t ChosenEdge = SPI_FIRST_EDGE;
  SPI_XferWidth_t DataWidth = SPI_8BIT;


  SPI_PinMap_t SSPin = SPI_RPA0; // Chip Select Pin
  SPI_PinMap_t SDOPin = SPI_RPA1; // Chip Output Pin
  SPI_PinMap_t SDIPin = SPI_RPB8; // Chip Input Pin


  TRISAbits.TRISA0 = 0;
  TRISAbits.TRISA1 = 0;

  ANSELAbits.ANSA0 = 0;
  ANSELAbits.ANSA1 = 0;

  SPI1CONbits.SRXISEL = 0b01;

  SPISetup_BasicConfig(Module);
  SPISetup_SetLeader(Module, SamplePhase);
  SPISetup_SetBitTime(Module, DesiredClock_ns);
  SPISetup_MapSSOutput(Module, SSPin);
  SPISetup_MapSDOutput(Module, SDOPin);
  //SPISetup_MapSDInput(Module, SDIPin);
  SDI1R = 0b0100;
  TRISBbits.TRISB8 = 1;

  SPISetup_SetClockIdleState(Module, ClockIdle);
  SPISetup_SetActiveEdge(Module, ChosenEdge);
  SPISetup_SetXferWidth(Module, DataWidth);
  SPISetEnhancedBuffer(Module, true);

  SPISetup_EnableSPI(Module);
  /* Set up interrupt */
  /*
  INTCONbits.MVEC = 1;
  IPC7bits.SPI1IP = 4;
  IPC7bits.SPI1IS = 0;

  IFS1CLR =  _IFS1_SPI1RXIF_MASK;
  IEC1SET = _IEC1_SPI1RXIE_MASK;
```

```
 */

// Start a periodic poll timer if polling is used
ES_Timer_InitTimer(COMMAND_SPI_TIMER, SPI_POLL_INTERVAL_MS);
__builtin_enable_interrupts();
// Post the initial transition event
ThisEvent.EventType = ES_INIT;
if (ES_PostToService(MyPriority, ThisEvent) == true)
{
  return true;
}
return false;
}

/****************************************************************************
Function
   PostCommandRetrieveService

Parameters
   ES_Event_t ThisEvent , the event to post to the queue

Returns
   bool, false if the enqueue operation failed, true otherwise

Description
   Posts an event to this service's queue

Author
   Tianyu, 02/03/26
****************************************************************************/
bool PostCommandRetrieveService(ES_Event_t ThisEvent)
{
 return ES_PostToService(MyPriority, ThisEvent);
}

/****************************************************************************
Function
  RunCommandRetrieveService

Parameters
  ES_Event_t : the event to process

Returns
  ES_Event_t, ES_NO_EVENT if no error ES_ERROR otherwise

Description
  Service state machine for SPI command retrieval.

Author
   Tianyu, 02/03/26
****************************************************************************/
ES_Event_t RunCommandRetrieveService(ES_Event_t ThisEvent)
{
 ES_Event_t ReturnEvent;
 ReturnEvent.EventType = ES_NO_EVENT;

 switch (ThisEvent.EventType)
 {
  case ES_INIT:
    // No action needed beyond initialization
    break;

  case ES_TIMEOUT:
  {
    if (ThisEvent.EventParam != COMMAND_SPI_TIMER)
    {
      break;
    }
    // Pseudocode: poll SPI follower for command bytes
    // Read byte from SPI (leader initiates query)
```

```
    // IF byte == 0xFF
    //    SawNewCommandFlag = true
    // ELSE IF SawNewCommandFlag == true
    //    IF byte is valid command
    //       Post ES_COMMAND_RETRIEVED(commandByte) to MainLogicService
    //    ELSE
    //       Ignore or log invalid command
    //    SawNewCommandFlag = false
    // ELSE
    //    Ignore (repeated old command value)


     uint8_t commandByte = (uint8_t) QueryCommandGenerator();
     DB_printf("Received Command byte: 0x%x\r\n", commandByte);
     if (commandByte == 0xFF)
     {
      SawNewCommandFlag = true;
     }
     else if (SawNewCommandFlag == true)
     {
      if (IsValidCommandByte(commandByte))
      {
       ES_Event_t CommandEvent;
       if(commandByte != LastCommand) {
          CommandEvent.EventType = ES_COMMAND_RETRIEVED;
          CommandEvent.EventParam = commandByte;
          PostMainLogicFSM(CommandEvent);
          LastCommand = commandByte;
       }

      }
      else
      {
       DB_printf("Invalid command byte: 0x%x\r\n", commandByte);
      }
      SawNewCommandFlag = false;
      LastCommand = 0xFF;
     }

    ES_Timer_InitTimer(COMMAND_SPI_TIMER, SPI_POLL_INTERVAL_MS);
    break;
   }
   default:
    break;
 }

 return ReturnEvent;
}

/*----------------------------- Module Helpers -----------------------------*/
/****************************************************************************
Function
   ReadSPICommandByte

Parameters
   None

Returns
   uint8_t command byte read from SPI

Description
   Reads a byte from the SPI follower.

Author
   Tianyu, 02/03/26
****************************************************************************/
uint8_t QueryCommandGenerator(void)
{
 // TODO: Implement SPI read (leader initiates query)
 // Pseudocode:
```

```
  //   Assert chip select
  //   Transfer dummy byte and read response
  //   Deassert chip select
  //   Return response byte
    if(!SPI1STATbits.SPITBF) {
        SPIOperate_SPI1_Send8Wait(0xAA);
    }
    uint8_t data = (uint8_t) SPIOperate_ReadData(Module);
    return data;
}

/*******************************************************************************
Function
    IsValidCommandByte

Parameters
    uint8_t commandByte

Returns
    bool, true if the command byte is valid

Description
    Validates a command byte against the lookup table.

Author
    Tianyu, 02/03/26
*******************************************************************************/
static bool IsValidCommandByte(uint8_t commandByte)
{
  // TODO: Implement command validation using CommonDefinitions lookup table
    bool returnVal = false;
    uint8_t index = 0;

    for(index; index < sizeof(validCommandBytes)/sizeof(validCommandBytes[0]); index++) {
        if(commandByte == validCommandBytes[index]) {
            returnVal = true;
        }
    }
    return returnVal;
}
```

4). Motor Running Service:

```
/****************************************************************************
Module
 DCMotorService.c

Revision
 1.0.1

Description
 This service controls two DC motors (left/right) using PWM outputs.

Notes
  When initializing the DC Motor Service:
  Configure & initialize I/O pins connected with DC motor for PWM
  Map OC output to I/O pin, connected to one of the motor control pins
     Disable the PWM Output Compare module
  Disable the PWM timer
  Set the TMRy prescale value and enable the time base by setting TON (TxCON<15>) =1
  Set the PWM period by writing to the selected timer period register (PRy).
  IF (read from direction I/O pin, and it is LOW)
     Set the PWM duty cycle ticks by writing to the OCxRS register
     Set the other motor control pin to LOW
  ELSE
     Set the PWM duty cycle ticks to (PWM period – duty cycle ticks + 1) by writing to the OCxRS register
     Set the other motor control pin to HIGH
  Write the OCxR register with the initial duty cycle ticks.
  Configure the Output Compare module for one of two PWM Operation modes by writing to the Output Compare mode bits, OCM<2:0> (OCxCON<2
Turn ON the Output Compare module
  Turn the timer to the PWM system on.

  On ES_MOTOR_ACTION_CHANGE event:
     Get desiredSpeed from event parameter
  Map the desired speed to duty cycle ticks
  IF (read from direction I/O pin, and it is LOW)
     Set the other motor control pin to LOW
  ELSE
     Set the new PWM duty cycle ticks to (PWM period – duty cycle ticks + 1)
     Set the other motor control pin to HIGH
  Clamp duty cycle ticks to safe range
  Write new duty cycle ticks to OCxRS



History
When         Who    What/Why
-------------- ---    --------
01/21/26     Tianyu  Updated for Lab 6 motor speed control
01/15/26     Tianyu  Fixed position wrapping logic for unsigned type
01/14/26     Tianyu  Initial creation for Lab 5
****************************************************************************/
/*---------------------------- Include Files ----------------------------*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ES_Timers.h"
#include "DCMotorService.h"
#include "CommonDefinitions.h"
#include "dbprintf.h"
#include <xc.h>

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

// Port definitions
#define MOTOR_FORWARD_PIN_L   LATBbits.LATB4
#define MOTOR_REVERSE_PIN_L   LATBbits.LATB5
//#define DIRECTION_PIN      PORTBbits.RB8  // Direction input pin

//Right Wheel Port definitions
#define MOTOR_FORWARD_PIN_R   LATBbits.LATB11
#define MOTOR_REVERSE_PIN_R   LATBbits.LATB13
```

```
// PWM configuration (period defined in CommonDefinitions.h)
#define INITIAL_DUTY_TICKS 0  // Initial duty cycle in ticks
#define ENABLE_POT_AD

/*---------------------------- Module Functions ----------------------------*/
/* Prototypes for private functions for this service */
static void ConfigureTimeBase(uint8_t prescale);
static void ConfigurePWM(void);
static void ConfigureDCMotorPins(void);
static uint16_t MapSpeedToDutyCycle(uint16_t desiredSpeed);

/*---------------------------- Module Variables ----------------------------*/
// Module level Priority variable
static uint8_t MyPriority;
static uint16_t DesiredSpeed[2];
static uint8_t DesiredDirection[2];

/*---------------------------- Module Code ----------------------------*/
/****************************************************************************
Function
   InitDCMotorService

Parameters
   uint8_t : the priority of this service

Returns
   bool, false if error in initialization, true otherwise

Description
   Initializes the DC Motor Service


Author
   Tianyu, 01/14/26
****************************************************************************/
bool InitDCMotorService(uint8_t Priority)
{
 ES_Event_t ThisEvent;

 MyPriority = Priority;
 DesiredSpeed[LEFT_MOTOR] = 0;
 DesiredSpeed[RIGHT_MOTOR] = 0;
 DesiredDirection[LEFT_MOTOR] = FORWARD;
 DesiredDirection[RIGHT_MOTOR] = FORWARD;

 /*****************************************
  Initialization code for DC Motor Control system
  *****************************************/
 // Initialize Output Compare Pins used
 // TODO: Expand ConfigureDCMotorPins for both motors
 ConfigureDCMotorPins();

 // Configure PWM module (includes timer configuration)
 ConfigurePWM();

 // Post the initial transition event
 ThisEvent.EventType = ES_INIT;
 if (ES_PostToService(MyPriority, ThisEvent) == true)
 {
  return true;
 }
 else
 {
  return false;
 }
}

/****************************************************************************
Function
   PostDCMotorService
```

```
Parameters
   ES_Event_t 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
   Tianyu, 01/14/26
****************************************************************************/
bool PostDCMotorService(ES_Event_t ThisEvent)
{
 return ES_PostToService(MyPriority, ThisEvent);
}

/****************************************************************************
Function
  RunDCMotorService

Parameters
  ES_Event_t : the event to process

Returns
  ES_Event_t, ES_NO_EVENT if no error ES_ERROR otherwise

Description
  Handles events to control the DC motor

Author
  Tianyu, 01/14/26
****************************************************************************/
ES_Event_t RunDCMotorService(ES_Event_t ThisEvent)
{
 ES_Event_t ReturnEvent;
 ReturnEvent.EventType = ES_NO_EVENT; // Assume no errors

 switch (ThisEvent.EventType)
 {
  case ES_INIT:
    // Initialization already done in Init function
    break;

  case ES_MOTOR_ACTION_CHANGE:
  {
    // Pseudocode:
    // FOR each motor in Motors[]
    //   Map desired speed to duty ticks
    //   Clamp duty ticks to safe range
    //   IF motor.direction == FORWARD
    //     Set forward pin HIGH, reverse pin LOW
    //     Write duty ticks to OCxRS
    //   ELSE (REVERSE)
    //     Set forward pin LOW, reverse pin HIGH
    //     Write duty ticks to OCxRS (or inverted if hardware requires)
    // END FOR

    uint16_t dutyCycle = MapSpeedToDutyCycle(DesiredSpeed[LEFT_MOTOR]);

    if (DesiredDirection[0] == 0)
    {
      MOTOR_REVERSE_PIN_L = 0;
      OC1RS = dutyCycle;
      DB_printf("dutyCycle left 0:%u\r\n", dutyCycle);
    }
    else
    {
```

```
      MOTOR_REVERSE_PIN_L = 1;
      OC1RS = PWM_PERIOD_TICKS - dutyCycle + 1;
      DB_printf("dutyCycle left 1:%u\r\n", OC1RS);
    }

    // Hardware motor already inversed for right motor, when forward means same current go through left and right motor
    if (DesiredDirection[1] == 0)
    {
      MOTOR_REVERSE_PIN_R = 0;
      OC2RS = dutyCycle;
      DB_printf("dutyCycle right 0:%u\r\n", dutyCycle);
    }
    else
    {
      MOTOR_REVERSE_PIN_R = 1;
      OC2RS = PWM_PERIOD_TICKS - dutyCycle + 1;
      DB_printf("dutyCycle right 1:%u\r\n", OC2RS);
    }

    break;
  }

  default:
    break;
 }

 return ReturnEvent;
}

/*****************************************************************************
Function
   MotorCommandWrapper

Parameters
   uint16_t speedLeft, speedRight
   uint8_t dirLeft, dirRight

Returns
   None

Description
   Writes desired speeds/directions into module variables and posts
   ES_MOTOR_ACTION_CHANGE events to DCMotorService.

Author
   Tianyu, 02/03/26
*****************************************************************************/
void MotorCommandWrapper(uint16_t speedLeft, uint16_t speedRight,
                         uint8_t dirLeft, uint8_t dirRight)
{
 ES_Event_t ThisEvent;

 DesiredSpeed[LEFT_MOTOR] = speedLeft;
 DesiredSpeed[RIGHT_MOTOR] = speedRight;
 DesiredDirection[LEFT_MOTOR] = dirLeft;
 DesiredDirection[RIGHT_MOTOR] = dirRight;

 ThisEvent.EventType = ES_MOTOR_ACTION_CHANGE;
 ThisEvent.EventParam = 0;
 PostDCMotorService(ThisEvent);

 DB_printf("DesiredSpeed:%u %u, DesiredDirection: %u %u\r\n", DesiredSpeed[0], DesiredSpeed[1], DesiredDirection[0], DesiredDirection[1]);
}

/*****************************************************************************
Private Functions
*****************************************************************************/

/*****************************************************************************
```

```
Function
  ConfigureTimeBase

Parameters
  None

Returns
  None

Description
  Configures the timer used as the time base for PWM operation

Author
  Tianyu, 01/21/26
*****************************************************************************/
static void ConfigureTimeBase(uint8_t prescale)
{
 // Clear the ON control bit to disable the timer
 T2CONbits.ON = 0;
 // Clear teh TCS control bit to select the internal PBCLK source
 T2CONbits.TCS = 0;
 // Select the desired timer input clock prescale
 T2CONbits.TCKPS = PrescaleLookup[prescale];
 // Clear the timer register TMRx
 TMR2 = 0;
 // Enable the timer by setting the ON control bit
 T2CONbits.ON = 1;
}

/*****************************************************************************
Function
  ConfigurePWM

Parameters
  None

Returns
  None

Description
  Configures the PWM Output Compare module for PWM operation
Notes
  This function configures both the timer base and the Output Compare module.
  The timer configuration is done first to ensure proper initialization order.

Author
  Tianyu, 01/21/26
*****************************************************************************/
static void ConfigurePWM(void)
{
 // Step 1: Configure the timer base (must be done before OC config)
 ConfigureTimeBase(PRESCALE_2);
 // Keep timer off during configuration to avoid unintended pulses
 T2CONbits.ON = 0;
 // Disable the PWM Output Compare module before configuration
 OC1CONbits.ON = 0;
 OC2CONbits.ON = 0;

 // Set the PWM period by writing to the timer period register
 PR2 = PWM_PERIOD_TICKS;
 // Write the OCxR register with the initial duty cycle
 OC1RS = INITIAL_DUTY_TICKS;
 // Configure the Output Compare module for PWM mode
 OC1CONbits.OCM = 0b110; // PWM mode on OCx; Fault pin disabled
 // Turn ON the Output Compare module
 OC1CONbits.ON = 1;

 OC2CONbits.OCM = 0b110;  // PWM Mode
 OC2CONbits.OCTSEL = 0;   // Also use Timer 2
 OC2RS = INITIAL_DUTY_TICKS;
```

```
 OC2CONbits.ON = 1;

 // Start the timer after PWM configuration is complete
 TMR2 = 0;      // Clear timer register for clean start
 T2CONbits.ON = 1;
}

/*****************************************************************************
Function
   ConfigureDCMotorPins

Parameters
   None

Returns
   None

Description
   Configures the I/O pins for DC motor control as outputs

Author
   Tianyu, 01/21/26
*****************************************************************************/
static void ConfigureDCMotorPins(void)
{
 // Configure pins as digital outputs (all pins here don't have analog functions)
 TRISBbits.TRISB4 = 0;  // MOTOR_FORWARD as output
 TRISBbits.TRISB5 = 0;  // MOTOR_REVERSE as output
 TRISBbits.TRISB11 = 0;  // MOTOR_FORWARD_R as output
 TRISBbits.TRISB13 = 0;  // MOTOR_REVERSE_R as output

 // Initialize all pins to low
 MOTOR_FORWARD_PIN_L = 0;
 MOTOR_REVERSE_PIN_L = 0;
 MOTOR_FORWARD_PIN_R = 0;
 MOTOR_REVERSE_PIN_R = 0;

 // Map OC1 output to RB4
 RPB4R = 0b0101;
 // Map OC2 output to RB11
 RPB11R = 0b0101;
}

/*****************************************************************************
Function
   MapSpeedToDutyCycle

Parameters
   uint16_t desiredSpeed: desired speed value (0 to ADC_MAX_VALUE)

Returns
   uint16_t: duty cycle value (0 to PWM_PERIOD_TICKS), clamped to safe range

Description
   Maps the desired speed from the ADC range to PWM duty cycle range
   and clamps the result to ensure it stays within valid bounds.

Notes
   Uses ADC_MAX_VALUE from CommonDefinitions.h as the source of truth for the
   ADC range to ensure consistency across services.

Author
   Tianyu, 01/21/26
*****************************************************************************/
static uint16_t MapSpeedToDutyCycle(uint16_t desiredSpeed)
{
 // Map the desired speed to duty cycle
 // desiredSpeed range: [0, ADC_MAX_VALUE] → dutyCycle range: [0, PWM_PERIOD_TICKS]
 // uint32_t dutyCycle = ((uint32_t)desiredSpeed * PWM_PERIOD_TICKS) / ADC_MAX_VALUE;
```

```
  uint16_t dutyCycle = desiredSpeed;

  // Clamp duty cycle to safe range
  if (dutyCycle > DUTY_MAX_TICKS)
  {
    dutyCycle = DUTY_MAX_TICKS;
  }
  else if (dutyCycle < DUTY_MIN_TICKS)
  {
    dutyCycle = DUTY_MIN_TICKS;
  }

  return (uint16_t)dutyCycle;
}

/*------------------------------- Footnotes -------------------------------*/
/*------------------------------- End of file -------------------------------*/
```