Electronics Description

Beacon Sensor

Two beacon sensors are mounted just below the beacon on top of the robot. They face forward and provide stereo vision for finding and tracking the 3 kHz beacon at the side of the field. Each beacon sensor uses a photo-transistor to detect the beacon. The current generated by the photo-transistor is amplified using a 6044 op-amp. The signal is then passed through a high pass filter to remove ambient light differences, a gain stage, a low pass filter, and a second stage of gain. All filtering and gains are done at a base level of 2.5 volts to prevent the inputs to the op-amp from going negative. The high pass is designed with a corner frequency of 1590 Hz. Picking a low capacitor value of 0.001μF yields a resistor value of 100 kΩ. The high pass is designed for a corner frequency of 3400 Hz. Choosing the capacitor value of 47 pF yields a resistor value of 1MΩ. The first gain stage is in a non-inverting configuration with a gain of 23. A 100 kΩ resistor was chosen for the feedback resistor, which yields a pull-down resistor value of 4.7 kΩ. The second gain stage is also non-inverting, but utilizes a variable potentiometer to adjust the gain. The 4.7k pull-down resistor and the 100k feedback resistor combined with the 10k potentiometer provide gains varying from 8 to 23. As a result, the total system gain can vary from 184 to 529. The two stages of gain provide protection from the 6044's awful 100kHz gain bandwidth product limitation.

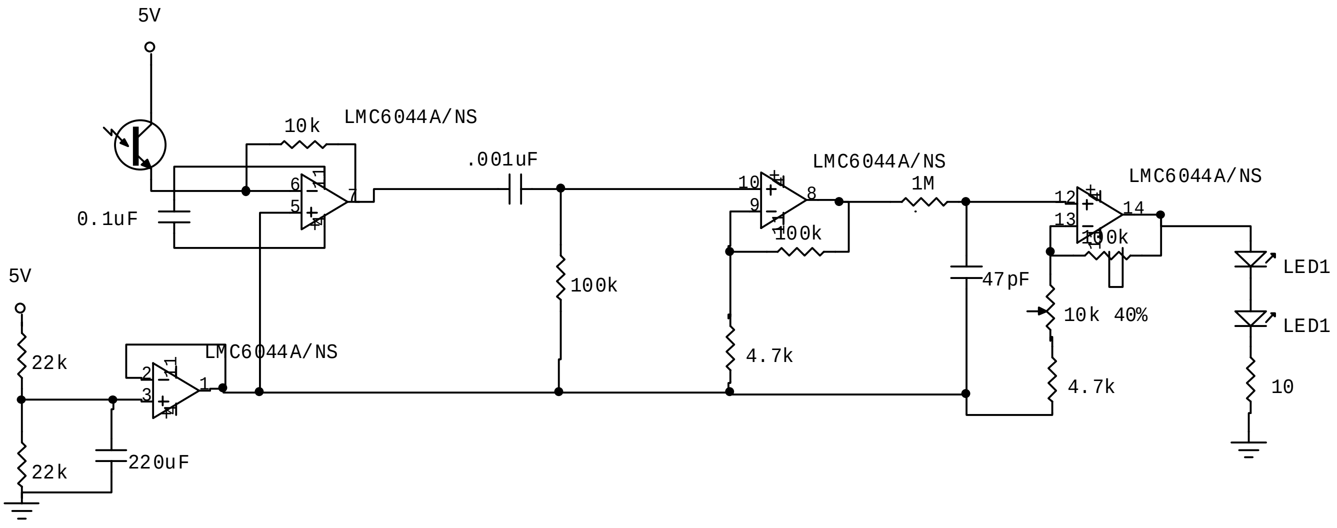
The voltage divider used to create the reference voltage uses 22kΩ resistors to limit current and power.

Bypass capacitors are used around the 6044 and the lower half of the voltage divider for noise protection.

LEDs are utilized to show when the sensor is detecting the beacon. (In general, the level must exceed the forward voltage of the two LEDs in series).

The beacon sensors are extremely reliable and were implemented early. Since we chose not to look for the opposing robot the range of the sensor is only specified for 3kHz. Another option would have been to provide a wider range in the sensor and use software to determine different frequencies.

Beacon Sensor Circuit Schematic



Tape Sensors

Four tape sensors are located on the outside edges near the front of the plow. Two are on the left, two are on the right, and there is 3" of separation between the fore and aft sensors. The 3" separation was intended to determine different tape widths (that is center tape versus edge tape). Eventually only the two front sensors were used in software but they all worked reliably as long as the height was set correctly. Adjustability was provided for both height and direction when the tape sensors were mounted to the platform. It is convenient to have that flexibility but they tend to get tweaked out of position. In retrospect, we would determine the correct height prior to machining the final platform and concern ourselves with keeping the tape sensors pointed directly at the ground.

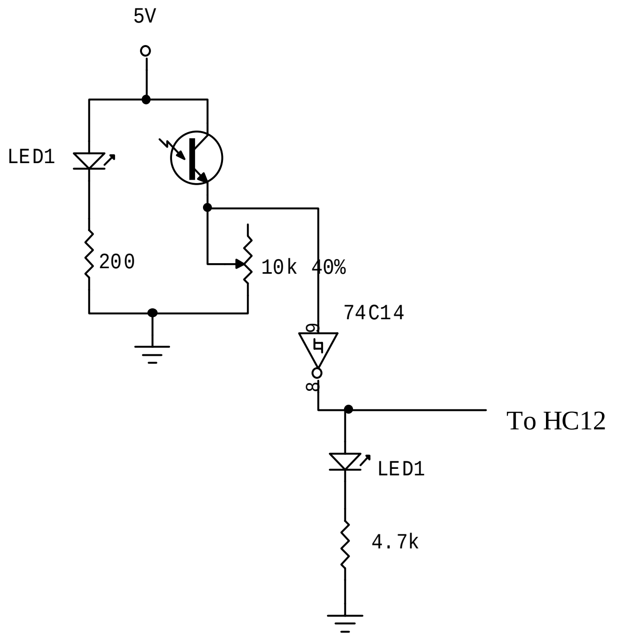
The tape sensors are in a simple Schmitt trigger circuit. The IR Emitter is driven at a constant current of approximately 33mA. (1.7 V drop over the emitter with a 200Ω resistor). The 200Ω resistor limits the current to 25mA even if the voltage drop across the LED were 0 volts.

The Schmitt trigger provides clean edges to the HC12 and works very reliably as long as thresholds are set correctly. Using a 10k pot enables us to adjust to varying light conditions.

Tape Sensor Status Display

Four LEDs provide information on the status of the tape sensors. They are switched in software with every edge detected. The LEDs are driven from the HC12 with high value resistors (270k) to extremely limit the current drawn from the HC12 (0.018 mA even if there is no drop across the LED).

Tape Sensor Circuit Schematic



Puck Sensor

The puck sensor is mounted in the middle of the puck collection system on the underside of the robot. It uses an IR emitter and IR detector pair to detect reflections of IR off of the puck. The emitter runs at 1kHz.

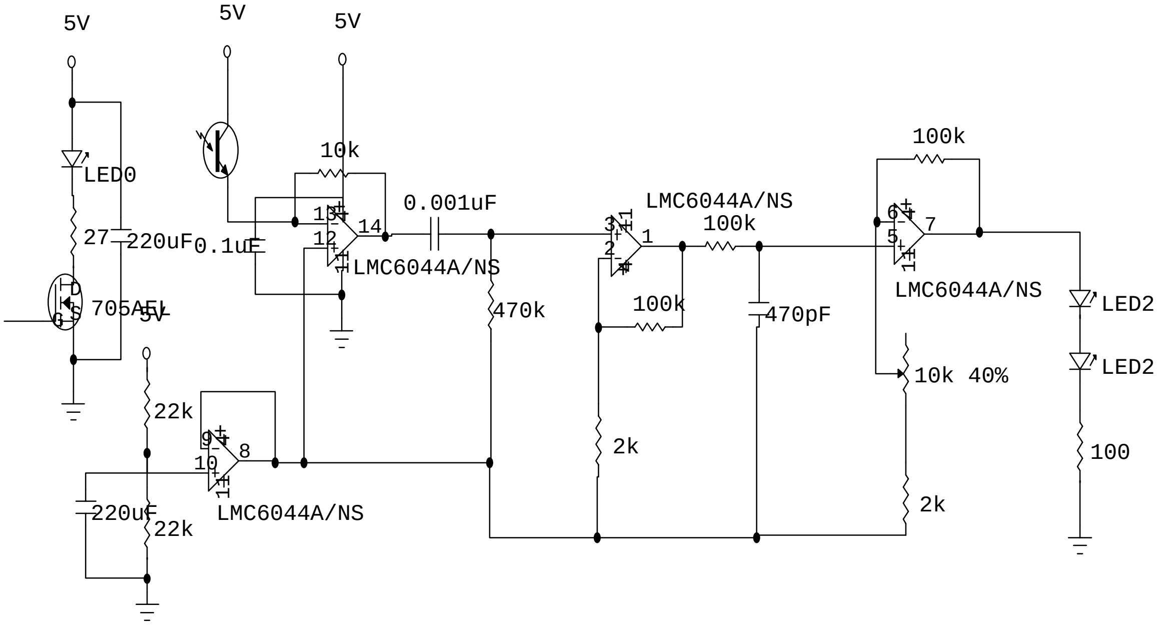
The IR emitter is designed to run at 1kHz with a 30% duty cycle. The low duty cycle allows the emitter to run at 225 mA. (These numbers were determined by examining the beacons). The voltage drop across the emitter was measured to be 1.2V. With a 27Ω resistor, 140mA of current runs through the emitter, which proved to be plenty. Noise is a problem with this much current running through the circuit, so a separate 5 volt regulated power supply is dedicated to the emitter.

The IR detector portion of the puck sensor is the same as the beacon sensor with different corner frequencies and gains. The high pass break frequency is designed to be 340 Hz. Choosing a capacitor value of 0.001 µF results in a resistor value of 470 kΩ. The low pass break frequency is designed to be 3390 Hz. Choosing a capacitor value of 470 pF results in a resistor value of 100 kΩ. The gains were again designed with 100kΩ feedback resistors. The first stage provides a gain of 51 with the 2 kΩ pull-down resistor. The second stage is variable from 9 to 51 using the 10k potentiometer. The overall gain of the circuit varies from 459 to 2601. The two stages of gain again avoid the 100kHz gain bandwidth product limitation of the 6044 op-amp.

Two LEDs in series again give a general idea of when the puck is detected.

The puck sensor has a range of roughly 3-4 feet. Using a 10k potentiometer in the last gain stage allows for easy adjustments and adapting to different environments.

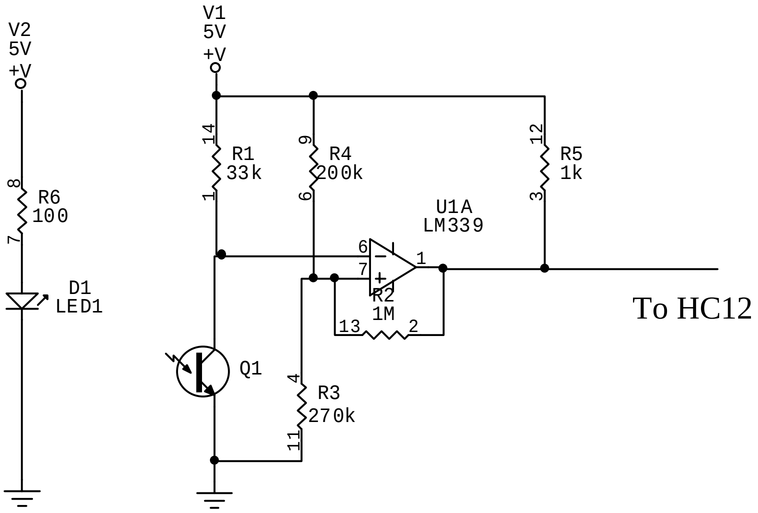
Puck Sensor Circuit Schematic



Puck Counter

The puck counter is a simple IR transmitter/receiver circuit passed through a comparator. The transmitter is mounted on one side of the plow inline with the receiver. The comparator was utilized to allow for varying bandwith but a Schmitt trigger would have been sufficient (especially as we didn't use a potentiometer in the comparator circuit). The output is connected to the T5 line on the HC12.

Puck Counter Circuit Schematic



PuckSensor

This module contains three functions. One initializes the PWM for the tape sensor at a frequency of 976 Hz. The other function, CheckTape, reads the frequency coming from the tape. Since the PWM is emitting a known frequency if the sensor is positioned over the tape it should read the same frequency. CheckTape returns ON if the frequency being read is the same and OFF if it is not.

Void InitPuckCounter(void)

```
Initializes the puck counter on T5.
    Sets it to capture a falling edge.
    Sets the interrupt flag.
   Enables Timer Subsystem
   Sets Puck_Status to 0, Drive_State to 1, and Puck_State to 1.
```

__interrupt void PuckCounter(void)

```
Increments Puck_Status
Clears Input Compare #5 Flag
```

Void InitPuckPWM(void)

```
Sets clock source for P2 to S1
Enables PWM channel 2
Sets clock period for P2 to 16 counts (15.6kHz/16 = 976Hz)
Set clock S1 prescale to 512 (8e6/512=15.6kHz)
Set duty cycle to 0%
```

Unsigned int GetPuck(void)

```
Based on Puck_State:
    STATE 1:
        Set SHORTTIMER to 10 seconds
        pucks = Puck_Status
        Go directly to state 2
    STATE 2:
        If SHORTTIMER is expired
            Meander until done, then set Puck_State = 1
        Otherwise, FindPuck()
        If FOUNDPUCK, Puck_State = 3
    STATE 3:
        pucks = Puck_Status
        Go directly to state 4
    STATE 4:
        If pucks < Puck_Status
            Puck_State = 1
            increment to pucks
            Set PUCKTIMER to 1/2 second
            Go to Puck_State 5
        Otherwise,  DriveToPuck()
    STATE 5:
            Catch the puck to the left or right based on catch_dir.
            Turn until PUCKTIMER is expired, then reset it for 1/2 second
            and go to State 6.
    STATE 6:
            Go forward until PUCKTIMER is expired, then stop and reset
            Puck_State to 1.
  Returns pucks (The number of pucks)
```

unsigned int Check_Puck_Flag(void)

```
Indicates whether a single puck has been picked up.
```

Static unsigned int CheckPuck(void)

```
Set puck duty cycle to 50%  (Hardcoded to P2)
Analyze Puck Signal to get amplitude
If amplitude > THRESHOLD, status = FOUNDPUCK
Else status = NOPUCK
Return status
```

Static unsigned int FindPuck(void)

```
Set the display to FindPuck
Status = CheckPuck
If there is NOPUCK, spin LEFT SOFT
Otherwise, StopMotor()
Return status
```

Static void DriveToPuck(void)

```
Set Display to Drive to Puck
CheckPuck()
If FOUNDPUCK, ForwardMotor(FULL)
Otherwise based on Drive_State:
   STATE 1:
      Set timer 3 to ONE_SECOND
      Go directly to state 2
   STATE 2:
      RelocatePuck to the LEFT
      If FOUNDPUCK, Drive_State = 1, ForwardMotor(FULL)
      Otherwise, if Timer 3 EXPIRED Drive_State = 3
   STATE 3:
      RelocatePuck to the RIGHT
      If FOUNDPUCK, Drive State = 1, ForwardMotor(FULL)
```

Static unsigned int RelocatePuck(int direction)

```
Status = CheckPuck
  If NOPUCK, Turn in direction, SOFT
  Otherwise, StopMotor()
  Return status
```

External #defines used:

```
SHORTTIMER
FIVE_SECONDS
TMR_EXPIRED
MOTORDONE
TEN_SECONDS
PUCK
LEFT
SOFT
FULL
ONE_SECOND
RIGHT
FOUNDPUCK
PUCK
```

```
#defines created:
        NOPUCK
        FOUNDPUCK
        PUCKACQUIRED
        ALLPUCKSIN
        GOTPUCK

Internal #defines
        THRESHOLD
```

Fire

Module variable: Fire_State – tracks the Firing sub-state machine

Void InitFire(void)

```
Sets Fire_State to 1
Sets the Refind_Flag to 0
```

Unsigned int Fire(void)

```
Fire_Status = 0;
Based on Fire_State:
    STATE 1:
       Go directly to state 2
   STATE 2:
       Set the Display to indicate the Drive to Fan State
       Point to the fans then go to State 3.
   STATE 3:
       If either front tape sensor is ON tape, stop and go to State 4.
       (Okay, so the following part is excluded by the first rule, but hey
       we were sleep deprived!!!)
       Otherwise, if both tape sensors on one side are on:
           Turn softly to the opposite side
           Set Refind_Flag to 1
       Otherwise, if Refind_Flag is 1:
           Go to State 1
           Reset Refind_Flag to 0
            (Okay, so it might be better if it just headed straight after
           getting off the tape....)
       Otherwise (finally!!)
           Drive Forward
   STATE 4:
       Set the Display to the Fire State
       Turn to the opposing goal, based on the knowledge of which side of the field
       it's on:
           If it's on the "RIGHT" side of the field, turn 90 degrees to the left, using
           the wind to determine the turn angle.
           If it's on the "LEFT" side of the field, turn 90 degrees to the right, using
           the wind to determine the turn angle.
       Once the turn is completed, stop and go to State 5.
```

```
STATE 5:
    Drive Forward until either both front tape sensors or both back tape sensors are
    ON tape. Then:
        Stop
        Set the Display to the Fire State
        Fire the Valve  (WOOHOO!!!)
        Pause for 1/2 second
        Go to State 6
STATE 6:
    Back off the tape, then set Fire_Status to FIRED.
DEFAULT:
    Fire_State = 1
    Fire_Status = 0
    Return Fire_Status
```

```
External #defines used:
        SHORTTIMER
        FIVE_SECONDS
        TMR_EXPIRED
        MOTORDONE
        FOUNDBEACON
        ATBEACON
        LF
        RF
        LB
        RB
        ON
        SOFT
        RIGHT
        LEFT
        NINETY_LEFT
        NINETY_RIGHT
        ONE_SECOND
        FULL


#defines created:
        FIRED
```

Beacon
Module variable: Beacon_State

```
Unsigned int FindBeacon( void )
    Get Left Beacon Amplitude
    Get Right Beacon Amplitude
    Status = NOSIGNAL
    If LeftAmp < THRESHOLD and RightAmp < THRESHOLD
        Spin LEFT, SOFT
    Else If LeftAmp > (RightAmp + TOLERANCE)
        Spin LEFT, SOFT
    Else If RightAmp > (LeftAmp + TOLERANCE)
        Spin RIGHT, SOFT
    Else If (LeftAmp < (RightAmp + TOLERANCE)) and (LeftAmp > (RIGHTAMP – TOLERANCE))
        StopMotor()
        Status = FOUNDBEACON
    Return status

Unsigned int FollowBeacon( void )
    Check FrontLeft and FrontRight Tape Sensors
    If both are ON tape
        StopMotor()
        Status = BEACONDONE
    Otherwise,
        Get Left and Right Beacon Amplitudes
        If LbeacAmp > (RbeacAmp + BIGTOL)
            TurnLeft(HARD)
            Status = 0
        Else If LbeacAmp > (RbeacAmp + TOLERANCE)
            TurnLeft(SOFT)
            Status = 0
        Else If RbeacAmp > (LbeacAmp + BIGTOL)
            TurnRight(HARD)
            Status = 0
        Else If LbeacAmp > (LbeacAmp + TOLERANCE)
            TurnRight(SOFT)
            Status = 0
        Else If LbeacAmp and RbeacAmp < THRESHOLD
            Spin LEFT, SOFT
            Status = 0
        Else
            ForwardMotor FULL
            Status = 0
```

External #defines used:

- SHORTTIMER
- FIVE_SECONDS
- TMR_EXPIRED
- LEFTBEACON
- RIGHTBEACON
- MOTORDONE
- LEFT
- SOFT
- RF
- LF
- ON
- OFF
- HARD
- FULL

#defines created:

- NOSIGNAL
- FOUNDBEACON
- ATBEACON

Internal #defines

- LOW
- HIGH
- TOLERANCE
- THRESHOLD
- BIGTOL

Filter

Void InitFilter(void)

```
Sets ATDCTL2 register to:
    Allow the ATD to function normally (ADPU=1)
    Clear flag normally (AFFC = 0)
    Continue to run in wait mode (AWAI = 0)
    Disable ATD interrupt (ASCIE = 0)
Sets ATDCTL5 register to:
    Perform 4 conversions (S8CM = 0)
    Scan continuously (SCAN = 1)
    Use multiple input channels (MULT = 1)
    Use AD0-4 (CD=0, CC=0, CB=0, CA=0)  (CB and CA don't matter)
```

Void AnalyzeSignal(unsigned int *amp, int port)

```
Uses port to determine which AD line to use otherwise the same:
Set FILTERTIMER to WAITPER
Wait until Input > High Threshold or Timer expires
If timer didn’t expire
    Set FILTERTIMER to WAITPER
    Wait until Input < Low Threshold or Timer expires
    If timer didn’t expire
        Set max = reference voltage (2.5 volts)
        Set FILTERTIMER to WAITPER
        Wait for Input to go above Low Threshold or timeout and collect the
        minimum input level
If timer didn’t expire
    Tempamp = (Reference – max(min)) * 255/Reference
Otherwise, Tempamp = 0
*amp = tempamp
```

External #defines used:

- FILTERTIMER
- TMR_EXPIRED
- TMR_NOT_EXPIRED

#defines created:

- LEFTBEACON
- RIGHTBEACON
- PUCK
- _H12_AWAI

Internal #defines

- BEACLO
- BEACHI
- PUCKHI
- PUCKLO
- REFVOLT
- WAITPER

Appendix B – Code
PuckSensor

```
/* PuckSensor.h */

/*-------------------------------Definitions-------------------------------*/
#define NOPUCK 0
#define FOUNDPUCK 1
#define PUCKACQUIRED 1
#define ALLPUCKSIN 6
#define GOTPUCK 23

/*----------------------------Module Prototypes----------------------------*/

void InitPuckCounter( void );
void InitPuckPWM( void );
unsigned int GetPuck( void );
unsigned int Check_Puck_Flag(void);
unsigned int CheckPuck(void);

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


//#define TESTPUCK
//#define DEBUG
//#define SIM
//#define DEBUGPUCK
#ifdef DEBUG
  #include <stdio.h>
  #include <dbprintf.h>
#endif

#ifdef TESTPUCK
  #include <stdio.h>
  #include <dbprintf.h>
#endif
/****************************************************************************
 Module
     PuckSensor.c

 Revision
         1.0

 Description
 * This module contains three functions.  One initializes the PWM for the tape sensor at a
 * frequency of 976 Hz.  The other function, CheckTape, reads the frequency coming from
 * the tape.  Since the PWM is emitting a known frequency if the sensor is positioned
 * over the tape it should read the same frequency.  CheckTape returns ON if the
 * frequency being read is the same and OFF if it is not.



 Notes

 History
 When           Who What/Why
 -------------- --- --------
 2/23/01 13:00  lmf  Modified Lab 9 TapeSensor to PuckSensor. Wrote/modified
                     CheckPuck, FindPuck, DriveToPuck.  Left the PWM set up as it
                     is at 976 HZ ~ 1kHz.
 2/13/01 16:00 jkk  Changed PWM to be momentary.  Added Test Harness. Added
                    #defines for frequencies
****************************************************************************/
/*----------------------------- Include Files -----------------------------*/
#include <me218_912.h>
#include "PuckSensor.h"
#include "MotorDrive.h"
#include "filter2.h"
#include "timer.h"
#include "StateDisplay.h"

/*-------------------------------Definitions-------------------------------*/
#define THRESHOLD 1    // Threshold above which the puck is being detected
```

```
/*----------------------------Module Prototypes----------------------------*/
static unsigned int FindPuck( void );
static void DriveToPuck( void );
static unsigned int RelocatePuck( int direction );

/*---------------------------- Module Variables ---------------------------*/
static unsigned int Puck_Status;
static unsigned int Drive_State;   // State machine indicator for DriveToPuck
static unsigned int Puck_State;
static unsigned int Puck_Flag;          // Flag to indicate when at least
                                        // one puck has been picked up.


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

/****************************************************************************
 Function
    InitPuckCounter

 Parameters

 Returns

 Description
    Initializes the puck counter on T5.
    Sets it to capture a falling edge.
    Sets the interrupt flag.
    Set Puck_Status to 0
    Set Drive_State to 1
    Set Puck_State to 1


 Notes
        The puck sensor uses Port P2
        The puck counter uses Port T5

 Author
        Larissa Fontaine, 3/05/01  18:00
****************************************************************************/

void InitPuckCounter (void)
{
    _H12TCTL3 |= _H12_EDG5B;    // Capture Falling Edge on Pin T5
   #ifndef SIM
    _H12TMSK1 |= _H12_C5I;      // Enable Input Compare Interrupt on IC5
   #endif
    _H12TSCR  |= _H12_TEN;      // Enable Timer
    _H12TFLG1 = _H12_C5F;       // Sets interrupt flag
    Puck_Status = 0;
    Puck_Flag = 0;
}

/****************************************************************************
 Function
    PuckCounter

 Parameters

 Returns

 Description
    Interrupt routine for the puck counter.
    Uses T5, increments the puck counter every time the interrupt is triggered.
    Resets the flag.

 Notes

 Author
        Larissa Fontaine, 3/05/01  18:00
****************************************************************************/
```

Appendix B – Code
PuckSensor

```
__interrupt void PuckCounter( void )
{

 #ifdef DEBUG
   DB_printf(" # of Pucks = %d\n",Puck_Status);
 #endif
     Puck_Status++;
     Puck_Flag = GOTPUCK;
     _H12TFLG1 = _H12_C5F;
 #ifdef DEBUGPUCK
     DB_printf("\nPuck Interrupt:  # of Pucks = %d\n",Puck_Status);
 #endif


}

/****************************************************************************
 Function
          InitPuckPWM

 Parameters

 Returns

 Description
          Sets clock source for P2 to S1
          Enables PWM channel 2
          Sets clock period for P2 to 16 counts (15.6kHz/16 = 976Hz)
          Set clock S1 prescale to 512 (8e6/512=15.6kHz)
          Set duty cycle to 0%

 Notes
         The puck sensor uses Port P2
         The puck counter uses Port T5

 Author
         Larissa Fontaine, 2/13/01  9:00
****************************************************************************/
void InitPuckPWM(void)
{
    _H12PWPOL |= _H12_PCLK2;                // Sets clock S1 as source for P2
    _H12PWEN |= _H12_PWEN2;                 // Enables P2 as an output
    _H12PWPER2= 15;                         // Sets clock period for Port P2 to 16
    _H12PWSCAL1=255;                        // Sets prescaler to count to 512
    _H12PWDTY2= 15;                          // Inits P2 duty cycle to 0%
    Drive_State = 1;
    Puck_State = 1;
}
/****************************************************************************
 Function
    GetPuck

 Parameters

 Returns
        The number of pucks caught.

 Description
    Based on Puck_State:
        STATE 1:
            Set SHORTTIMER to 10 seconds
            pucks = Puck_Status
            Go directly to state 2
        STATE 2:
            If SHORTTIMER is expired
                Meander until done, then set Puck_State = 1
            Otherwise, FindPuck()
            If FOUNDPUCK, Puck_State = 3
        STATE 3:
            pucks = Puck_Status
                              Go directly to state 4
        STATE 4:
            If pucks < Puck_Status
                Puck_State = 1
                                        increment to pucks
                                        Set PUCKTIMER to 1/2 second
                                        Go to Puck_State 5
            Otherwise,  DriveToPuck()
                    STATE 5:
                              Catch the puck to the left or right based on catch_dir.
                              Turn until PUCKTIMER is expired, then reset it for 1/2 second
                              and go to State 6.
                    STATE 6:
                              Go forward until PUCKTIMER is expired, then stop and reset
                              Puck_State to 1.
    Returns pucks (The number of pucks)

 Notes
        None.

 Author
        Larissa Fontaine, 2/23/01  9:00
****************************************************************************/
unsigned int GetPuck(void)
{
    static unsigned int pucks=0;
    static unsigned int catch_dir = LEFT;  // Direction to turn when getting a puck

   #ifdef SIM
     char ans;
   #endif
    switch (Puck_State)
    {
        case 1 :
            TMR_InitTimer(SHORTTIMER, TEN_SECONDS);
            Puck_State = 2;
            pucks = Puck_Status;
        case 2 :
            if ( TMR_IsTimerExpired(SHORTTIMER) == TMR_EXPIRED) {
                if (Meander()==MOTORDONE) {
                    Puck_State = 1;
                    #ifdef DEBUG
                        DB_printf("Time Out!!!!, State = %d\n",Puck_State);
                    #endif
                }
            }
            else if (FindPuck() == FOUNDPUCK) {
             Puck_State = 3;

              #ifdef DEBUG
                DB_printf("Found Puck!!\n");
              #endif
            }
        break;
        case 3 :
            //TMR_InitTimer(SHORTTIMER, TEN_SECONDS);
            Puck_State = 4;
            pucks = Puck_Status;
        case 4 :
         #ifdef SIM
               DB_printf("Picked up a puck? y/Y:\n");
               ans = getchar();
               if ( (ans == 'y') || (ans == 'Y'))  {
                  Puck_Status++;
         #endif
         #ifndef SIM
            if (pucks < Puck_Status) {  // Picked up a puck
         #endif
               pucks++;
               Puck_State = 1;
                #ifdef  DEBUGPUCK
                    DB_printf("Got a puck!!! Catching It!, #caught = %d, pucks=%d\n",
                              Puck_Status,pucks);
               #endif
               TMR_InitTimer(PUCKTIMER, ONE_SECOND/2);
               Puck_State = 5;
            }
            else {
               DriveToPuck();
            }
        break;
        case 5 :
            if ( TMR_IsTimerExpired(PUCKTIMER) == TMR_NOT_EXPIRED ) {
                if ( catch_dir == LEFT ) {
                    #ifdef  DEBUGPUCK
                        DB_printf("Catching Puck to the LEFT\n");
                    #endif
                    TurnLeft(SOFT);
                    catch_dir = RIGHT;
                }
                else {
                    #ifdef  DEBUGPUCK
                        DB_printf("Catching Puck to the RIGHT\n");
                    #endif
                    TurnRight(SOFT);
                    catch_dir = LEFT;
                }
            }
             else {
               TMR_InitTimer(PUCKTIMER, ONE_SECOND/2);
               Puck_State = 6;
             }
        break;
        case 6 :
            if ( TMR_IsTimerExpired(PUCKTIMER) == TMR_NOT_EXPIRED ) {
               ForwardMotor(FULL);
            }
            else {
               StopMotor();
               Puck_State = 1;
               //_H12PWDTY2= 15;           // PuckSensor duty cycle to 0%

            }
        break;
    }
    return pucks;
}

/***************************************************************************
 Function
   Check_Puck_Flag

 Parameters
         None
 Returns
         Puck_Flag
 Description
    Indicates whether a single puck has been picked up.

 Notes
         None.

 Author

***************************************************************************/

unsigned int Check_Puck_Flag(void)
{
    return Puck_Flag;
```

```
}

/***************************************************************************
 Function
          CheckPuck

 Parameters

 Returns

 Description
    Set puck duty cycle to 50%  (Hardcoded to P2)
    Analyze Puck Signal to get amplitude
    If amplitude > THRESHOLD, status = FOUNDPUCK
    Else status = NOPUCK
    Return status

 Notes
         None.

 Author
         Larissa Fontaine, 2/23/01  9:00
***************************************************************************/

unsigned int CheckPuck(void)
{
    unsigned int Amplitude, status;

    _H12PWDTY2= 7;               // P2 duty cycle to 50%

    AnalyzeSignal(&Amplitude, PUCK);
    if ( Amplitude >= THRESHOLD ) {
    status = FOUNDPUCK;
    }
    else {
          status = NOPUCK;
    }

//    _H12PWDTY2= 15;                          // P2 duty cycle to 0%

    return status;
}

/***************************************************************************
 Function
          FindPuck

 Parameters
          None.
 Returns

 Description
          Set the display to FindPuck
    Status = CheckPuck
    If there is NOPUCK, spin LEFT SOFT
    Otherwise, StopMotor()
    Return status

 Notes
         None.

 Author
         Larissa Fontaine, 2/24/01  14:20
***************************************************************************/
static unsigned int FindPuck( void )
{
    unsigned int status;

    SetDisplay(3);              // Find Puck State
    if ( (status=CheckPuck()) == NOPUCK)  {
        SpinMotor(LEFT, SOFT);
        //TurnLeft(WIDE);
        #ifdef DEBUGJK
            DB_printf("Looking for Puck - Soft Spin Left\n");
        #endif
    }
    else {
        StopMotor();
  #ifdef DEBUGJK
     DB_printf("Found Puck - stop\n");
  #endif
    }
    return status;
}

/****************************************************************************
 Function
          DriveToPuck

 Parameters
          None.
 Returns

 Description
          Set Display to Drive to Puck
    CheckPuck()
    If FOUNDPUCK, ForwardMotor(FULL)
    Otherwise based on Drive_State:
        STATE 1:
            Set timer 3 to ONE_SECOND
            Go directly to state 2
        STATE 2:
            RelocatePuck to the LEFT
            If FOUNDPUCK, Drive_State = 1, ForwardMotor(FULL)
            Otherwise, if Timer 3 EXPIRED Drive_State = 3
        STATE 3:
            RelocatePuck to the RIGHT
            If FOUNDPUCK, Drive State = 1, ForwardMotor(FULL)

 Notes
         None.

 Author
         Larissa Fontaine, 2/24/01  14:20
****************************************************************************/

static void DriveToPuck( void )
{
    unsigned int puck_there;
  #ifdef DEBUGJK
     DB_printf("Drive To Puck: ");
  #endif

    SetDisplay(4);      //Drive to Puck State

    if ( (puck_there=CheckPuck()) == FOUNDPUCK) {
        ForwardMotor(FULL);
        #ifdef DEBUGJK
          DB_printf("  Driving to Puck\n");
        #endif
    }
    else {
        switch ( Drive_State ) {
            case 1 :
              TMR_InitTimer(TURNTIMER,ONE_SECOND);
              Drive_State=2;
                #ifdef DEBUGJK
                    DB_printf("Init: ");
                #endif
```

```
            case 2 :                    // Quick jaunt to left to see if puck is there
                #ifdef DEBUG
                   DB_printf("Lost puck, looking Left, Puck is ");
                   switch ( puck_there )
                   {
                    case FOUNDPUCK :
                        DB_printf("THERE!!!\n");
                    break;
                    case NOPUCK :
                        DB_printf("NOT there\n");
                    break;
                    default:
                        DB_printf("No idea!!!!!!!!!!!!!\n");
                    break;
                   }

                #endif
                if ( RelocatePuck(LEFT)==FOUNDPUCK ) {
                    Drive_State=1;
                    ForwardMotor(FULL);
                }
                else if ( TMR_IsTimerExpired(TURNTIMER)==TMR_EXPIRED)
                {
                    Drive_State=3;
                }
            break;
            case 3 :
                #ifdef DEBUGJK
                   DB_printf("Lost puck, looking right\n");
                #endif
                if ( RelocatePuck(RIGHT)==FOUNDPUCK ) {
                    Drive_State=1;
                    ForwardMotor(FULL);
                }
            break;
        }
    }
}

/****************************************************************************
 Function
   RelocatePuck

 Parameters
    direction - which direction to turn

 Returns
     Integer specifying whether puck is found or not

 Description
    Status = CheckPuck
    If NOPUCK, Turn in direction, SOFT
    Otherwise, StopMotor()
    Return status


 Notes
         None.

 Author
        Jonathan Karpick, 3/6/01  00:47
****************************************************************************/
static unsigned int RelocatePuck( int direction )
{
    unsigned int status;

    if ( (status=CheckPuck()) == NOPUCK) {
        if ( direction == RIGHT)
            TurnRight(SOFT);
        else TurnLeft(SOFT);
    } else {
        StopMotor();    }
    return status;
}



/**************** TEST HARNESS **********************/
#ifdef TESTPUCK

void main( void )
{
    unsigned int pucks;
    char selection;

    InitPuckCounter();
    InitFilter();
    InitPuckPWM();
    InitMotors();
    InitDisplay();
    TMR_Init(TMR_RATE_2MS);
    while(1) {
        DB_printf("Menu:\n");
        DB_printf("1. CheckPuck (FOUNDPUCK or NOPUCK)\n");
        DB_printf("2. FindPuck (Spin until puck is found)\n");
        DB_printf("3. DriveToPuck (picks up a puck)\n");
        DB_printf("4. GetPuck (Finds a puck and goes and gets it)\n");
        DB_printf("5. Get All pucks - Get Puck x6\n");
        selection = getchar();
        switch ( selection )
        {
            case '1' :
                DB_printf("Puck Status = %d, FOUNDPUCK=%d, NOPUCK=%d\n",
                CheckPuck(), FOUNDPUCK, NOPUCK);
            break;
            case '2' :
                while ( FindPuck() == NOPUCK )
                   ;
                StopMotor();
                DB_printf("Find Puck = %d\n", FindPuck());
            break;
            case '3' :
                pucks = Puck_Status;
                while ( pucks == Puck_Status)  {
                    DriveToPuck();
                }
                StopMotor();
                DB_printf("# of Pucks = %d\n", pucks);
            break;
            case '4' :
                pucks = Puck_Status;
                while ( pucks == GetPuck())
                   ;
                DB_printf("Puck = %d\n", pucks);
            break;
            case '5' :
                while (GetPuck() < ALLPUCKSIN)
                    ;
            break;
            default:
                DB_printf("Sorry, try again.\n");
            break;
        }
    }
}
#endif


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

Appendix B – Code
Beacon

```
/* Beacon.h */

/*-------------------------------Definitions-------------------------------*/
#define NOSIGNAL 20
#define FOUNDBEACON 24
#define ATBEACON 25
/*----------------------------Module Prototypes----------------------------*/
 //unsigned int DriveToBeacon(void);
unsigned int FindBeacon(void);
unsigned int FollowBeacon( void );

/*-----------------END OF FILE---------------------------*/


//#define TESTBEACON
//#define DEBUGBEACON
//#define DEBUG
//#define SIM
#ifdef DEBUG
  #include <stdio.h>
  #include <dbprintf.h>
#endif
#ifdef TESTBEACON
  #include <stdio.h>
  #include <dbprintf.h>
#endif

/****************************************************************************
 Module
     u:\_Teams\helterskelter\ProjectCode\Beacon.c

 Revision
         1.0

 Description
          Module that contains the beacon related funtions:
                    Turns and finds the beacon
                    Drives to the beacon

 Notes
    AD2 - Left Beacon
    AD3 - Right Beacon
 History
 When           Who What/Why
 -------------- --- --------
 2/10/01 16:00  aps  first pass
 2/23/01 13:30  lmf  specify for project needs
****************************************************************************/
/*----------------------------- Include Files -----------------------------*/

#include <me218_912.h>
#include "motordrive.h"
#include "filter2.h"
#include "Beacon.h"
#include "Timer.h"
#include "TapeSensor.h"
#include "StateDisplay.h"

/*-------------------------------Definitions-------------------------------*/
#define LOW 0
#define HIGH 1
#define TOLERANCE 20
#define THRESHOLD 1
#define BIGTOL 30
/*----------------------------Module Prototypes----------------------------*/
/*---------------------------- Module Variables ---------------------------*/

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

/****************************************************************************
 Function
        FindBeacon
 Parameters
        None.
 Returns
        integer signifying FindBeacon completed.
 Description
     Responsible for spinning the robot until it is directly facing the
        beacon. Returns status, FOUNDBEACON or 0.

 Notes
         None.

 Author
         Larissa Fontaine 14:23 pm, 2/24/01
****************************************************************************/
unsigned int FindBeacon( void )
{
    unsigned int LeftAmplitude, RightAmplitude, status;


  #ifdef TESTBEACON
     DB_printf("Getting Values:\n");
  #endif

    SetDisplay(1);                  // Find Beacon State
    AnalyzeSignal(&LeftAmplitude, LEFTBEACON);
    AnalyzeSignal(&RightAmplitude, RIGHTBEACON);
    status = NOSIGNAL;
  #ifdef TESTBEACON
     DB_printf("Left = %d, Right = %d\n,LeftAmplitude, RightAmplitude");
  #endif
    if ((LeftAmplitude < THRESHOLD) && (RightAmplitude < THRESHOLD)) {  //No Signal
        SpinMotor(LEFT, SOFT);
        status = NOSIGNAL;
     #ifdef DEBUG
        DB_printf("LeftBeac = 0, RightBeac = 0, Spin SOFTLY!!!\n");
     #endif
        }
    else if (LeftAmplitude > (RightAmplitude + TOLERANCE))  {   //Beacon to left
        SpinMotor(LEFT, SOFT);
        status = NOSIGNAL;
     #ifdef DEBUG
        DB_printf("LeftBeac > RightBeac, Spin LEFT SOFTLY!!!\n");
     #endif
    }
    else if (RightAmplitude > (LeftAmplitude + TOLERANCE))  {   //Beacon to right
        SpinMotor(RIGHT, SOFT);
        status = NOSIGNAL;
     #ifdef DEBUG
        DB_printf("RightBeac > LeftBeac Spin RIGHT SOFTLY!!!\n");
     #endif
    }
  else if ( (LeftAmplitude > THRESHOLD) && (RightAmplitude > THRESHOLD) )  {  //Both pretty
much equal
        StopMotor();                   // but non-zero
        status = FOUNDBEACON;
     #ifdef DEBUG
        DB_printf("LeftBeac = RightBeac, STOP!!\n");
     #endif
     }
     #ifdef DEBUG
        DB_printf("LeftBeac= %d, RightBeac = %d, Beacon = %d, FOUNDBEACON=%d, NOSIGNAL=%d\n",
                   LeftAmplitude, RightAmplitude, status, FOUNDBEACON, NOSIGNAL);
     #endif

    return status;
}

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

Appendix B – Code

Beacon

```
 Function
        FollowBeacon
 Parameters
        None.
 Returns
        integer signifying FollowBeacon completed.
 Description
     A function responsible for moving the robot towards the beacon.
 Notes
         Checks to see that both back tape sensors are off tape.
            Gets the amplitude of the signal coming into both sensors.
         Turns hard or sorft depending on comparisons between the left
            and right signals.
         Moves forwards when both amplitudes are equal.
         If both beacon signals are lost spins left until found again.
         When both front tape sensors hit tape, stops and returns LF-flag, ON.

 Author
         Larissa Fontaine 14:23 pm, 2/24/01
****************************************************************************/

unsigned int FollowBeacon( void )
{
    unsigned int lBeaconAmp, rBeaconAmp, status, tapestatus;

    #ifdef TESTBEACON
        DB_printf("Following Beacon: ");
          #endif

     SetDisplay(5);      //FollowBeacon State

     if ( (CheckTape(RF) == ON) && (CheckTape(LF) == ON) ) {
        tapestatus = 1;
        StopMotor();
        status = ATBEACON;
    #ifdef TESTBEACON
        DB_printf("Both Front on Tape, Stop!!!\n");
          #endif
    }
    else {
        AnalyzeSignal(&lBeaconAmp, LEFTBEACON);
        AnalyzeSignal(&rBeaconAmp, RIGHTBEACON);
        tapestatus = 0;
    #ifdef TESTBEACON
        DB_printf("Left Beac = %d, Right Beac = %d: ",lBeaconAmp, rBeaconAmp);
          #endif
                    if( lBeaconAmp > (rBeaconAmp + BIGTOL)) {
                              TurnLeft(HARD);
            status = 0;
    #ifdef TESTBEACON
          DB_printf("Hard Left\n");
          #endif
        }
        else if( lBeaconAmp > (rBeaconAmp + TOLERANCE)) {
               TurnLeft(SOFT); //turns left until beacon ampl's approx equal
                status = 0;
    #ifdef TESTBEACON
          DB_printf("Soft Left\n");
          #endif
        }
        else if( rBeaconAmp > (lBeaconAmp + BIGTOL)) {
                              TurnRight(HARD);
                    status = 0;
        #ifdef TESTBEACON
        DB_printf("Hard Right\n");
        #endif
        }
        else if( rBeaconAmp > (lBeaconAmp + TOLERANCE)) {
                              TurnRight(SOFT); // turns right until beacon ampl's approx equal
                status = 0;
    #ifdef TESTBEACON
          DB_printf("Soft Right\n");
          #endif
        }
        else if ((lBeaconAmp < THRESHOLD) && (rBeaconAmp < THRESHOLD)) {
                              SpinMotor(LEFT,SOFT);
                status = 0;
    #ifdef TESTBEACON
          DB_printf("Lost, spinning Left\n");
          #endif
        }
        else {
            ForwardMotor(FULL);
            status = 0;
    #ifdef TESTBEACON
          DB_printf("FORWARD CHARGE!!!!\n");
          #endif
        }
    }
     #ifdef TESTBEACON
        DB_printf("Tape Status = %d\n",tapestatus);
        DB_printf("Beacon Status = %d\n",status);
     #endif

     return status;
}


/*********************** TEST HARNESS ********************/
#ifdef TESTBEACON

main()
{
    //unsigned int status;              // error "not used in block"
    char selection;

    while(1)  {
        InitMotors();
        InitTapeSensors();
        InitFilter();
        TMR_Init(TMR_RATE_2MS);
        DB_printf("Menu:\n");
        DB_printf("1. FindBeacon (Turn to Beacon)\n");
        DB_printf("2. FollowBeacon (Drive towards Beacon until tape is hit)\n");
        selection = getchar();
        switch ( selection )
        {
            case '1' :
                while ( FindBeacon()==NOSIGNAL )  {
                    DB_printf("Looking for Beacon \n");
                }
            break;
            case '2' :
                while ( FollowBeacon() != ATBEACON ) {
                    DB_printf("Following Beacon \n");
                }
                #ifdef TESTBEACON
                    DB_printf("Beacon Status = %d\n",FollowBeacon());
                #endif
            break;
            default:
                DB_printf("Don't recognize that number\n");
            break;
        }  //Switch
    }  //Inf Loop
}
#endif


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