

Clarkson University Intelligent Cars IV

Final Report

Revision A - December 8, 2017

Written by Team KVM:

James Bruska, Milton Griffin, Robin Kuhns,

Timothy Law, Ryan Shippee, Matthew Strenk

**Table of Contents**

[**Overview**](#_itpauz4np49m) **3**

[**Evaluation and Analysis**](#_d6mhyo8nx2sv) **3**

[Specifications](#_g8vlky9yugan) 3

[Supplemental Capabilities](#_k1obs8p2jv52) 3

[Unique Innovations](#_o13cfsi2e84s) 4

[Problems and Solutions](#_qti0z1opdsjy) 5

[**Testing**](#_98clzl7ddso4) **6**

[Unit Tests](#_yo0rbuib41x5) 6

[System Integration Tests](#_6chwk72phsj3) 7

[Performance Data](#_l0hg14wqjw9) 9

[PID Tuning](#_x33jhm970tu4) 9

[Pixy Tuning](#_2r0bozjwyor4) 11

[**Design Documentation**](#_bxursr9elbw7) **11**

[Zybo](#_1rwuwedmes5m) 11

[C code](#_zdkgprky8xe5) 11

[Main.c](#_1ddhbe8cwoar) 11

[Modes.h](#_4dkdrmulpa2d) 13

[Motor\_and\_steering.h](#_a8n7stwrh53c) 21

[Linescan.h](#_iq705yfpx09l) 22

[Timer\_and\_pwm.h](#_hivn9sea6u0d) 27

[Buttons\_and\_leds.h](#_7bva5lrxoliv) 31

[Pixy.h](#_ro9ubce9ib07) 33

[VHDL](#_qau7i2otz0y3) 34

[Constraints File](#_gaagiwou5njo) 37

[Schematics](#_z82mouueczao) 38

[FRDM](#_iunk1jxap44o) 39

[C code](#_at5l4dl4dn21) 39

[Main.c](#_f27r48y4o9w3) 39

[Modes.h](#_mube3ggyeigg) 41

[Motor\_and\_steering.h](#_tifwvxmp4ny6) 51

[Linescan.h](#_88hyd6ptc8ep) 53

[Timer\_and\_pwm.h](#_9c4ptvkbhywd) 60

[Buttons\_and\_leds.h](#_cs9s7a4mywas) 61

[Pixy.h](#_hnd1gmcvwuuw) 62

[**References**](#_isbhqgwv4u1f) **69**

# Overview

This document serves as the Final Report for the Clarkson University Intelligent Cars IV project as part of the University’s Computer Engineering Senior Lab class. The contents of this document will include an evaluation and analysis of the project, the test results, and include all of the source code used within the project

# Evaluation and Analysis

This section describes the specifications, capabilities, and innovations of this project.

## Specifications

The Concept Document specifications can be described simply as, create a car using the two boards that can perform 4 modes: accuracy mode, speed mode, collision avoidance mode, and discovery mode. In accuracy mode, the car has to move around the track as precisely as possible for 2 laps. In speed mode, the car has to move around the track as quickly as possible for 2 laps. In collision avoidance mode, the car follower car has to catch up the leader car and then follow it for a lap. In discovery mode, the car must find the track when placed within 2.5 feet of it and then locate the finish line. Our products (Zybo car and FRDM car) were able to perform some of these tasks. The FRDM car was able to perform accuracy mode, speed mode, and discovery mode. The Zybo car was able to perform accuracy mode and speed mode. The other modes failed due to various causes. The specific causes of failure can be seen in Section 3 of this report.

## Supplemental Capabilities

There are four supplementary parts of the car that were designed. Three of them were full implemented while the other was partially completed. First, the car was equipped with headlights. This helped to improve the line scan readings. Second, a manual mode for the car was then implemented. This allows for the car to be controlled by any controller that is connected to the car by adjusting the speed or tire position. This was not fully completed though as the bluetooth mechanism to connect the boards to the computer was not implemented. Third, a GUI was implemented on the computer. If the bluetooth module was implemented then this would display the speed and tire position of the car. It would also allow the user on the GUI to control the car if the car was put into manual mode. Finally, the car was stylized with accents in order to make the car more appealing. This includes acrylic plating for the boards, under-board lighting on the acrylic plates, and a Yu-Gi-Oh card attached to the rear of the car. Although, this does not improve the functionality, it does improve the user experience significantly.

## Unique Innovations

The overall system was unique on its own. It was designed using function pointers in order to allow for simplicity upon expansion. For example, if a new mode was needed, it would just use the drivers in order to gather car data and then use the steering and speed functions to control the car movements. Then, it would just be added to the mode changing mechanism and when the car was turned to that mode, the car would operate correctly. This setup also ensures that no variables are being changed by two functions or interrupts “at the same time.”

The line scan mechanism was also creative as it performed anomaly removal before doing analysis on each the line. First it would threshold the values into 0 and 1. Before it processes the information, the algorithm will remove anomalies. This is set by a constant with a value such as 3. This would mean that if the algorithm was looking at a black line section, it would allow up to 3 white pixels before it ended the black line and switch to a white section. After that, it would process all of the information about the line to store in variables such as the line center position, the longest line length, if stop line was seen, and if an intersection was seen.

The Zybo board uses specialized hardware in its FPGA to communicate with the Pixy camera and save cpu time of the Zybo’s ARM processor. This hardware did not have full integration at final testing. This design receives information from the Pixy and stores the data until the full detection if complete. Once the complete detection is stored, an interrupt signal is thrown. The ARM then receives the detection’s signature, x position, and height. These are the only parameters that would be used by the system so, only these signals are sent to reduce bloat.

The PID equation used is unique to its specific tuning for the car. The equation itself used only a proportional and derivative term in its execution, each of which included a calculated and adjusted gain based on Ziegler-Nichols tuning. This PID tuning, in addition to a conditional statement to control the speed of the vehicle, enabled the car to move at much higher speeds in straight areas of the track with a minute amount of self-adjustment, and still handle turns extremely accurately.

The design of the logic for the car’s collision avoidance mode using the CMU Pixy camera was unique. The method that was chosen for the following car to know its distance from the leading car was to use the height attribute of a signature that the camera sees. The visible height will not change as the cars go around the track like the width will. The height of the signature was measured at certain distances from the camera incrementing in three inches, starting at one foot and ending at three feet. A best fit line for the measured data was found and was used for the conversion of height to distance. Another equation was used to solve for the speed that the car should be going at a certain distance. With the two equations in place, it was easy to add bounds so that nothing unexpected occurred.

## Problems and Solutions

There were many problems that occured over the course of the system implementation. One problem that occured was the interrupts not working on the Zybo board for every run. It was determined that if the board was not turned off and back on again in between times that the board was programmed, the interrupts in the program would not occur. This may be due to the global interrupt controller being turned on and then never turned off when the new program is loaded onto the device.

Another problem that we encountered was a buffer overflow from a compiled library that would overwrite exactly 8 bits of out variable with 0s. This took forever to find as it was not due to our code. It was eventually found when the program was progressed through line by line and the memory was being watched at the same time. The problem was temporarily solved by adding an extra variable that was 8 bits in the location that the function overflowed into. This meant that the extra variable was changed and our original variable stayed safe.

One of the car’s functions that was not completed was the bluetooth for both the FRDM car and the Zybo car. This was because the component that we chose to implement bluetooth was a development bluetooth board where the firmware had to be installed to the chip. This added an extra step in the process and required extra time that was not available. If this project was to be done again, the bluetooth module that we would choose would be a component that is easier to implement and use. Once it was realized that bluetooth would not be completed using that component, we decided to switch over to using a BlueSmirf bluetooth component. This component was chosen because we used them in junior lab and the school already had them. There was not anymore success when using the Bluesmirf as we could not maintain a stable connection between the component and the PC. Bluetooth ended up not being used for the sake of time and so that time could be used on more important aspects of the car.

Another one of the car’s functions that was not completed was the CMU Pixy camera on the Zybo car. The hardware and code for operating the Pixy camera and analyzing the data generated by the Pixy camera were but put into place however, the capability for interfacing between the ARM processor and the dedicated hardware was not developed. This caused the car to lack signal detection and collision detection. Once interfacing is implemented, the car will gain the ability to to turn on command signals and avoid collisions with other vehicals.

The last weakness in the car’s design was data logging. The initial plan was to incorporate data logging in with GUI that would be running on the PC. The PC would communicate and transfer the data with the car using bluetooth; but since we could not get the bluetooth to work properly, data logging would not work. The GUI is finished and would work as soon as the bluetooth’s connection was established. In an attempt to get data logging to work on the FRDM car, we started to work on implementing the SD card. This would save certain variables and flags to a log file on the SD card. We were able to get the SD card to have data written to it when the code was not implemented into the main project, but as soon as the code was joined with the rest of the project the file could not be created. We as we debugged the code, we found that it was getting stuck in a function in an event file. We were not able to solve this issue as we ran out of time. If we were to have more time, we would have debugged the code line by line and figured out what was causing the issues that were being experienced.

# Testing

This section contains an explanation for all of the testing that has been performed.

## Unit Tests

The following tables describe, at a high level, all of the testing that took place to determine if each component was functioning. The status “IN PROGRESS” means that the functionality was being worked on but could not be completed due to the time restraint.

|  |  |  |
| --- | --- | --- |
| Capability | FRDM | Zybo |
| Line Following | Yes | Yes |
| Variable Speed | Yes | Yes |
| Mode Selection | Yes | Yes |
| Intersection Detection | Yes | Yes |
| Signal Detection | Yes | IN PROGRESS |
| Turn at Intersection | Yes | Yes |
| Stop Line Detection | Yes | Yes |
| Line Recovery | Yes | Yes |
| Data Logging | IN PROGRESS | No |

Table 1: High Level Unit Testing

|  |  |  |
| --- | --- | --- |
| Capability | FRDM | Zybo |
| Headlights | Yes | Yes |
| Bluetooth | No | No |
| GUI (Monitor)) | Not Integrated | Not Integrated |
| GUI (Manual Mode) | Not Integrated | Not Integrated |

Table 2: Optional Features

## System Integration Tests

The following tables (Table 3 to Table 6) describe the functionality of each car.

|  |  |  |
| --- | --- | --- |
| Test | FRDM | Zybo |
| System Startup | Yes | Yes |
| Mode Accessible | Yes | Yes |
| Follows Track | Yes | Yes |
| Turning | Yes | default left only |
| Track Completion | Yes | Yes |
| Power Off | Yes | Yes |

Table 3: Accuracy Mode

|  |  |  |
| --- | --- | --- |
| Test | FRDM | Zybo |
| System Startup | Yes | Yes |
| Mode Accessible | Yes | Yes |
| Follows Track | Yes | Yes |
| Turning | Yes | default left only |
| Track Completion | Yes | Yes |
| Power Off | Yes | Yes |

Table 4: Speed Mode

|  |  |  |
| --- | --- | --- |
| Test | FRDM | Zybo |
| System Startup | Yes | Yes |
| Mode Accessible | Yes | Yes |
| Follows Track | Yes | Yes |
| Turning | Yes | default left only |
| Catch Leader | Yes | No |
| Track Completion | Yes | Yes |
| Power Off | Yes | Yes |

Table 5: Collision Avoidance Mode

|  |  |  |
| --- | --- | --- |
| Test | FRDM | Zybo |
| System Startup | Yes | Yes |
| Mode Accessible | Yes | Yes |
| Locates Track | Yes | Yes |
| Follows Track | Yes | Yes |
| Turning | Yes | default left only |
| Track Completion | Yes | Yes |
| Power Off | Yes | Yes |

Table 6: Discovery Mode

## Performance Data

### PID Tuning

Tuning the PID equation used to control the automated steering functionality of the car was a heavily documented process which involved many different changes in speed, gain (proportional, integral, and derivative), and overall structure of the equation itself. The effects of the changes to the equation were always noted and kept careful track of. The last iteration of the equation and its tuning notes are included below in Table 7. Bold values were potential candidate values for speed mode. The last values were the ones actually used in the competition

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **a** | **b** | **c** | **Speed** | **Notes** |
| .028 | .000001 | .0003 | 50 / 28 | Too much overshoot |
| .026 | .000001 | .0003 | 50 / 28 | Too much overshoot |
| .026 | .00001 | .0005 | 50 / 28 | Understeers, and overshoots |
| .042 | 0 | 0 | 50 / 28 | Wayyy too much overshoot |
| .026 | 0 | 0 | 50 / 28 | Works well, slightly understeers |
| .028 | 0 | 0 | 50 / 28 | Performs semi-reliably. Recovers well |
| .030 | 0 | 0 | 50 / 28 | Reliable, but overshoots. Need D term |
| .030 | 0 | ,0001 | 50 / 28 | Still overshoots |
| .030 | 0 | ,0002 | 50 / 28 | Loses track of turn sometimes? |
| .028 | 0 | .0002 | 50 / 28 | Runs reliably. Still has overshoot |
| .028 | 0 | .0004 | 50 / 28 | Runs reliably. Overshoot reduced |
| .028 | 0 | .0006 | 50 / 28 | ^ |
| **.028** | **0** | **.0008** | **50 / 28** | **Overshoot is greatly reduced, but at the cost of understeer** |
| **.028** | **0** | **.0007** | **50 / 28** | **Doesn't understeer as much, still has some overshoot** |
| .028 | 0 | .0007 | 55 / 28 | Not reliable. Too much overshoot |
| .028 | 0 | .0007 | 50 / 26 | Reliable. Only some overshoot. Can go faster |
| .028 | 0 | .0008 | 50 / 26 | ^ |
| .028 | 0 | .0009 | 50 / 26 | ^ |
| .028 | 0 | .001 | 50 / 26 | ^ |
| .028 | 0 | .0011 | 50 / 26 | ^ |
| **.028** | **0** | **.0012** | **50 / 26** | **Very consistent. Definitely SM candidate** |
| **.028** | **0** | **.0015** | **50 / 26** | **Gives much less overshoot** |

Table 7: PID Tuning

### Pixy Tuning

To be sure that the Pixy camera would see and recognize the color code signatures that passed by the car the camera’s focus, signatures, and brightness needed to be adjusted. This was done by connecting the Pixy camera to a PC over USB and using the Pixymon software. This software was used to select the colors for each signature and adjust the range for each of the color code signatures, the brightness of the camera, and turning on auto-white balance. These adjustments assured that the car was seeing the color codes and would not get confused when seeing a similar color around the room. What made this difficult was depending on the time of the day the lighting in the room would be different, also the lighting was different on the track than it was at the workstation. To solve this we used a laptop and set the signatures while the car was on the track.

# Design Documentation

This section contains all of the source code and schematics for the embedded processors.

## Zybo

This section contains all of the code and schematics for the Zybo board.

### C code

#### Main.c

#include <stdio.h>

#include <stdint.h>

#include <stdlib.h> // exit

#include <unistd.h> // sleep

#include "platform.h"

#include "xil\_printf.h"

#include "xscugic.h"

#include "xgpio.h"

#include "xparameters.h"

#include "xil\_exception.h"

//#include "sleep.h" // sleep?

#include "modes.h"

#include "buttons\_and\_leds.h"

#include "timer\_and\_pwm.h"

#include "motor\_and\_steering.h"

#include "linescan.h"

#include "lidar.h"

static void initializeSystem();

static void buttonLedInit();

static void motorSteeringInit();

static void lineScanInit();

static void pixyInit();

static void lidarInit();

int main()

{

//int i = 0;

initializeSystem();

while(1){

while(!go\_flag){setMode();}

setMode();

go\_flag = 0;

while(!mode\_finished){

modeAction();

usleep(20\*1000);

}

modeAction = doNothingMode;

modeAction();

track\_laps = 0;

mode\_finished = 0;

}

cleanup\_platform();

return 0;

}

static void initializeSystem(){

init\_platform();

//print("Initializing Car\n\r");

modeAction = doNothingMode;

// Init GIC

GicSetup(&Intc);

buttonLedInit();

motorSteeringInit();

lineScanInit();

pixyInit();

lidarInit();

initializeTimerSystem();

// Enable interrupts in the Processor.

Xil\_ExceptionEnableMask(XIL\_EXCEPTION\_IRQ);

}

static void buttonLedInit(){

// Init Buttons and LEDs

initButtonsAndLeds();

}

static void motorSteeringInit(){

// Init PWMs

initSteerAndMotor();

}

static void lineScanInit(){

// Init PWMs

setupLineScanTimers();

int status = initializeXADC();

if(status != 0){

printf("Error in XADC initialization: %d", status);

exit(1);

}

}

static void pixyInit(){

}

static void lidarInit(){

}

#### Modes.h

#ifndef SRC\_MODES\_H\_

#define SRC\_MODES\_H\_

#include "buttons\_and\_leds.h"

#include "motor\_and\_steering.h"

#include "lidar.h"

#include "linescan.h"

static void setMode();

static void doNothingMode();

static void basicMode();

static void accuracyMode();

static void speedMode();

static void collisionMode();

static void discoveryMode();

static void manualMode();

static void turnRightSlow();

static void turnLeftSlow();

static void turnAroundSlow();

static void turnRightFast();

static void turnLeftFast();

static void turnAroundFast();

void (\*modeAction)(void);

uint8\_t mode\_finished = 0;

int speed = 0;

uint8\_t move\_dir = 0;

uint8\_t marker\_sigs[2] = {1, 0};

extern int intersect\_seen;

uint8\_t turning = 0;

uint8\_t markers\_opposite = 1;

//uint8\_t markers\_opposite = 0;

uint8\_t markers\_visible = 0;

int turn\_count = -1;

int repetitions = -1;

extern int track\_laps;

uint8\_t car\_ahead = 0;

uint8\_t line\_seen = 0;

static void setMode(){

if (mode\_count == 0){

modeAction = doNothingMode;

XGpio\_DiscreteWrite(&LEDInst, 1, 0b0001); //write switch data to the LEDs

}else if (mode\_count == 1){

modeAction = basicMode;

speed = 22;

XGpio\_DiscreteWrite(&LEDInst, 1, 0b0010); //write switch data to the LEDs

}else if (mode\_count == 2){

modeAction = accuracyMode;

speed = 24;

XGpio\_DiscreteWrite(&LEDInst, 1, 0b0011); //write switch data to the LEDs

}else if (mode\_count == 3){

modeAction = speedMode;

speed = 30;

XGpio\_DiscreteWrite(&LEDInst, 1, 0b0100); //write switch data to the LEDs

}else if (mode\_count == 4){

modeAction = collisionMode;

speed = 28;

XGpio\_DiscreteWrite(&LEDInst, 1, 0b0101); //write switch data to the LEDs

}else if (mode\_count == 5){

modeAction = discoveryMode;

XGpio\_DiscreteWrite(&LEDInst, 1, 0b0110); //write switch data to the LEDs

}else if (mode\_count == 6){

modeAction = manualMode;

XGpio\_DiscreteWrite(&LEDInst, 1, 0b0111); //write switch data to the LEDs

}

}

static void turnRightSlow(){

// TODO: change the speeds and the number of repetitions

static int turn\_actions[3][1] = {{-90}, {22}, {150}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 1){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}else{

--repetitions;

}

}else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

static void turnLeftSlow(){

// TODO: change the speeds and the number of repetitions

//static int turn\_actions[3][3] = {{0, -90, 0}, {1, 1, 1}, {2, 100, 2}}; // [[tire angles], [speeds], [repetitions]]

static int turn\_actions[3][1] = {{90}, {22}, {100}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 1){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}else{

--repetitions;

}

}else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

static void turnAroundSlow(){

// TODO: change the speeds and the number of repetitions

static uint8\_t turn\_actions[3][4] = {{0, 90, -90, 0}, {1, 1, 1, 1}, {2, 20, 20, 10}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 4){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (turn\_count == 2){ move\_dir = 1; }

if (turn\_count == 3){ move\_dir = 0; }

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}else{

--repetitions;

}

}else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

static void turnRightFast(){

// TODO: change the speeds and the number of repetitions

static int turn\_actions[3][1] = {{-90}, {22}, {140}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 1){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}else{

--repetitions;

}

}else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

static void turnLeftFast(){

// TODO: change the speeds and the number of repetitions

//static int turn\_actions[3][3] = {{0, -90, 0}, {1, 1, 1}, {2, 100, 2}}; // [[tire angles], [speeds], [repetitions]]

static int turn\_actions[3][1] = {{90}, {22}, {140}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 1){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}else{

--repetitions;

}

}else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

static void turnAroundFast(){

// TODO: change the speeds and the number of repetitions

static uint8\_t turn\_actions[3][4] = {{0, 90, -90, 0}, {1, 1, 1, 1}, {2, 20, 20, 10}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 4){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (turn\_count == 2){ move\_dir = 1; }

if (turn\_count == 3){ move\_dir = 0; }

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}else{

--repetitions;

}

}else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

static void doNothingMode(){

speedSet(0);

steerSet(0);

}

static void basicMode(){

if(turning){

if(marker\_sigs[0]){

if(marker\_sigs[1]){ turnAroundSlow(); }

else{ turnLeftSlow(); }

}else{

if(marker\_sigs[1]){ turnRightSlow(); }

else{ turning = 0; }

}

}else{

steerSet(steerPidOut());

speedSet(speed);

if(markers\_opposite){

if(intersect\_seen){

//markers\_opposite = 0;

if(!markers\_visible){ turning = 1; }

}else{

if(speed > 28){ speed -= 1; }

}

}else{

if(speed < 22){ speed +=1; };

}

}

}

static void accuracyMode(){

if(turning){

if(marker\_sigs[0]){

if(marker\_sigs[1]){ turnAroundSlow(); }

else{ turnLeftSlow(); }

}else{

if(marker\_sigs[1]){ turnRightSlow(); }

else{ turning = 0; }

}

}else{

steerSet(steerPidOut());

speedSet(speed);

if(track\_laps >= 2){ mode\_finished = 1; modeAction = doNothingMode;}

if(markers\_opposite){

if(intersect\_seen){

//markers\_opposite = 0;

if(!markers\_visible){ turning = 1; }

}else{

if(speed > 30){ speed -= 1; }

}

}else{

if(speed < 24){ speed +=1; };

}

}

}

static void speedMode(){

if(turning){

if(marker\_sigs[0]){

if(marker\_sigs[1]){ turnAroundFast(); }

else{ turnLeftFast(); }

}else{

if(marker\_sigs[1]){ turnRightFast(); }

else{ turning = 0; }

}

}else{

steerSet(steerPidOut());

speedSet(speed);

if(track\_laps >= 2){ mode\_finished = 1; }

if(markers\_opposite){

if(intersect\_seen){

//markers\_opposite = 0;

if(!markers\_visible){ turning = 1; }

}else{

if(speed > 30){ speed -= 1; }

}

}else{

if(speed < 26){ speed +=1; };

}

}

}

static void collisionMode(){

if(turning){

if(marker\_sigs[0]){

if(marker\_sigs[1]){ turnAroundSlow(); }

else{ turnLeftSlow(); }

}else{

if(marker\_sigs[1]){ turnRightSlow(); }

else{ turning = 0; }

}

}else{

steerSet(steerPidOut());

if(track\_laps >= 2){ mode\_finished = 1; }

if(markers\_opposite){

if(intersect\_seen){

markers\_opposite = 0;

if(!markers\_visible){ turning = 1; }

}else{

if(speed > 5){ speed -= 1; }

// TODO: possibly put code to avoid collision here

}

}else{

if(speed < 10){ speed +=1; };

if(car\_ahead){

speedSet(colPidOut());

}else{

speedSet(speed);

}

}

}

}

static void discoveryMode(){

// TODO: change the speeds, last tire pos, and the number of repetitions

//static uint8\_t turn\_actions[3][3] = {{0, 90, 45}, {10, 10, 10}, {20, 3, 100}}; // [[tire angles], [speeds], [repetitions]]

static int turn\_actions[3][2] = {{90, 45}, {24, 24}, {180, 500}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 2){

if (turn\_count == 2){ move\_dir = 1; }

if (turn\_count == 3){ move\_dir = 0; }

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}else{

--repetitions;

}

}else{

turn\_count = -1;

repetitions = -1;

modeAction = accuracyMode;

}

if(lineVisible() && turn\_count > 0){ modeAction = accuracyMode; }

}

static void manualMode(){

//steerSet(bluetooth\_tire\_pos());

//speedSet(bluetooth\_speed());

}

#endif /\* SRC\_MODES\_H\_ \*/

#### Motor\_and\_steering.h

#ifndef SRC\_MOTOR\_AND\_STEERING\_H\_

#define SRC\_MOTOR\_AND\_STEERING\_H\_

#include "xparameters.h"

#include "PWM.h"

#include "timer\_and\_pwm.h"

#define PWM0\_BASEADDR XPAR\_PWM\_0\_PWM\_AXI\_BASEADDR

#define PWM1\_BASEADDR XPAR\_PWM\_1\_PWM\_AXI\_BASEADDR

#define PWM\_BASE\_FREQ (100000000U)

#define MOTOR\_PWM\_FREQ (20000U)

#define STEER\_PWM\_FREQ (50U)

#define MOTOR\_PWM\_PERIOD (PWM\_BASE\_FREQ / MOTOR\_PWM\_FREQ)

#define STEER\_PWM\_PERIOD (PWM\_BASE\_FREQ / STEER\_PWM\_FREQ)

static void initSteerAndMotor();

static void updateMotorPwm(double duty\_cycle);

static void updateSteerPwm(double duty\_cycle);

static void speedSet(double speed);

static void steerSet(double angle);

double duty\_cycle\_per\_angle;

static void initSteerAndMotor(){

double pwm0\_duty\_cycle = 0;

double pwm1\_duty\_cycle = .075;

duty\_cycle\_per\_angle = 5.0/180.0;

PWM\_Set\_Period(PWM0\_BASEADDR, MOTOR\_PWM\_PERIOD);

PWM\_Set\_Period(PWM1\_BASEADDR, STEER\_PWM\_PERIOD);

PWM\_Set\_Duty(PWM0\_BASEADDR,(u32)(pwm0\_duty\_cycle\*MOTOR\_PWM\_PERIOD),0);

PWM\_Set\_Duty(PWM1\_BASEADDR,(u32)(pwm1\_duty\_cycle\*STEER\_PWM\_PERIOD),0);

PWM\_Enable(PWM0\_BASEADDR);

PWM\_Enable(PWM1\_BASEADDR);

}

static void updateMotorPwm(double duty\_cycle){

PWM\_Set\_Duty(PWM0\_BASEADDR,(u32)((duty\_cycle/100)\*MOTOR\_PWM\_PERIOD),0);

//printf("motor duty cycle: %d\r\n", duty\_cycle);

}

static void updateSteerPwm(double duty\_cycle){

PWM\_Set\_Duty(PWM1\_BASEADDR,(u32)((duty\_cycle/100)\*STEER\_PWM\_PERIOD),0);

//printf("steer duty cycle: %d\r\n", duty\_cycle);

}

static void speedSet(double speed){ // speed between 0 and 100

//updatePwmDutyCycle(speed, timer);

updateMotorPwm(speed);

}

static void steerSet(double angle){ // angle between -90 and 90

//int interval = 5; int center = 7.5;

//int duty\_cycle\_per\_angle = interval / 180;

//int match = duty\_cycle\_per\_angle \* angle + center;

//updatePwmDutyCycle(duty\_cycle\_per\_angle \* angle + 7.5, timer);

updateSteerPwm(duty\_cycle\_per\_angle \* angle + 7.5);

}

#endif /\* SRC\_MOTOR\_AND\_STEERING\_H\_ \*/

#### Linescan.h

#ifndef SRC\_LINESCAN\_H\_

#define SRC\_LINESCAN\_H\_

#include "timer\_and\_pwm.h"

#define SYSMON\_DEVICE\_ID XPAR\_SYSMON\_0\_DEVICE\_ID

#define PS7\_XADC\_DEVICE\_ID XPAR\_PS7\_XADC\_0\_DEVICE\_ID

static int setupLineScanTimers();

static void lineScanSiHandler(void \*CallBackRef);

static void lineScanClkHandler(void \*CallBackRef);

static void line\_eval();

static int steerPidOut();

int initializeXADC();

u32 singleReading();

XTtcPs line\_scan\_si, line\_scan\_clk;

int count = 0;

u32 line\_scan\_data[128];

u32 line\_scan\_temp[128];

u32 address;

int intersect\_seen = 0;

int prev\_intersect\_seen = 0;

int stop\_line\_seen = 0;

int track\_laps = 0;

int min\_val = 5;

int max\_val = 122;

int i;

int num\_black\_sections = 0;

int longest\_black\_section = 0;

int num\_outliers = 3;

int length\_black\_section = 0;

int length\_white\_section = 0;

int in\_black\_section = 0;

int num\_long\_black\_sections = 0;

int long\_section = 15;

int tire\_pos = 0;

int line\_start, line\_end, line\_center;

XAdcPs XADCMonInst;

XAdcPs\_Config \*XADCConfigPtr;

XAdcPs \*XADCInstPtr = &XADCMonInst;

XSysMon SysMonInst;

XSysMon\_Config \*SYSConfigPtr ;

XSysMon\* SysMonInstPtr = &SysMonInst;

static int setupLineScanTimers(){

u16 options;

int ret;

ret = genTimer(40, &line\_scan\_si, &lineScanSiHandler);

if (ret < 0) { return ret; }

XTtcPs\_EnableInterrupts(&line\_scan\_si, XTTCPS\_IXR\_MATCH\_0\_MASK);

options = XTtcPs\_GetOptions(&line\_scan\_si) | XTTCPS\_OPTION\_MATCH\_MODE;

XTtcPs\_SetOptions(&line\_scan\_si, options);

updatePwmDutyCycle(0.135, &line\_scan\_si);

ret = genTimer(20000, &line\_scan\_clk, &lineScanClkHandler);

if (ret < 0) { return ret; }

XTtcPs\_EnableInterrupts(&line\_scan\_clk, XTTCPS\_IXR\_MATCH\_0\_MASK);

options = XTtcPs\_GetOptions(&line\_scan\_clk) | XTTCPS\_OPTION\_MATCH\_MODE;

XTtcPs\_SetOptions(&line\_scan\_clk, options);

updatePwmDutyCycle(50, &line\_scan\_clk);

return 0;

}

static void lineScanSiHandler(void \*CallBackRef)

{

u32 StatusEvent;

StatusEvent = XTtcPs\_GetInterruptStatus((XTtcPs \*)CallBackRef);

XTtcPs\_ClearInterruptStatus((XTtcPs \*)CallBackRef, StatusEvent);

if (0 != (XTTCPS\_IXR\_INTERVAL\_MASK & StatusEvent)) {

XTtcPs\_Start(&line\_scan\_clk);

}

}

static void lineScanClkHandler(void \*CallBackRef)

{

u32 StatusEvent;

StatusEvent = XTtcPs\_GetInterruptStatus((XTtcPs \*)CallBackRef);

XTtcPs\_ClearInterruptStatus((XTtcPs \*)CallBackRef, StatusEvent);

if (0 != (XTTCPS\_IXR\_MATCH\_0\_MASK & StatusEvent)) {

line\_scan\_data[count] = singleReading();

if (line\_scan\_data[count] < 28000 && count > 4 && count < 123){

line\_scan\_data[count] = 0;

}else {

line\_scan\_data[count] = 1;

}

count++;

}

if(count == 129){

XTtcPs\_Stop(&line\_scan\_clk);

line\_eval();

count = 0;

}

}

static int steerPidOut(){

//line\_start = 0, line\_end = 0, line\_center = 0;

while(count != 0){}

line\_start = 0;

for(i = 5; i < 123; ++i){

if((line\_scan\_data[i] == 0) && (line\_start == 0)){

line\_start = i;

}

if((line\_scan\_data[i] == 0) && ((line\_scan\_data[i+1] == 1) || (i == 117))){

line\_end = i;

}

line\_center = ((line\_end - line\_start) / 2) + line\_start;

}

if(line\_center == 0){} //do nothing

else{

tire\_pos = ((59 - line\_center) \* 0.042) / (duty\_cycle\_per\_angle); // TODO: Change

//updateSteerPwm(7.5 + (59 - line\_center) \* 0.042);

//printf(" vals: %d, %d, %d, %d\r\n", line\_end, line\_start, line\_center, tire\_pos);

}

return tire\_pos;

}

static void line\_eval(){

int line\_center\_temp = 0;

int start\_black\_section = 0;

longest\_black\_section = 0;

num\_long\_black\_sections = 0;

for(i = min\_val; i < max\_val; ++i){ // For input (ends cut off)

//printf("%lu", line\_scan\_data[i]); // DEBUG

if(line\_scan\_data[i] == 0){ // If it's a black pixel

++length\_black\_section;

if(!in\_black\_section){ // If it's a white section

if(length\_black\_section > num\_outliers){ // If it's a new black section

in\_black\_section = 1;

++num\_black\_sections;

length\_white\_section = 0;

start\_black\_section = i - num\_outliers;

//printf(" B"); // DEBUG

}

}else{ // Reset length on false alarm from outliers

length\_white\_section = 0;

}

}else{ // If it's a white pixel

++length\_white\_section;

if(in\_black\_section){ // If it's a black section

if(length\_white\_section > num\_outliers){ // If it's a new white section

in\_black\_section = 0;

if(length\_black\_section > longest\_black\_section){ // Update longest black section

longest\_black\_section = length\_black\_section; // TODO: Put in an array of lengths?

//printf(" L (%d)", longest\_black\_section); // DEBUG

line\_center\_temp = ((i - num\_outliers) - start\_black\_section) / 2 + start\_black\_section;

}

if(length\_black\_section > long\_section){ ++num\_long\_black\_sections; }

length\_black\_section = 0;

//printf(" W"); // DEBUG

}

}else{ // Reset length on false alarm from outliers

length\_black\_section = 0;

}

}

if(i == (max\_val-1) && in\_black\_section){ // Ends in a black section

if(length\_black\_section > longest\_black\_section){ // Update longest black section

longest\_black\_section = length\_black\_section; // TODO: Put in an array of lengths?

//printf(" L (%d)", longest\_black\_section); // DEBUG

line\_center\_temp = ((i - num\_outliers) - start\_black\_section) / 2 + start\_black\_section;

}

if(length\_black\_section > long\_section){ ++num\_long\_black\_sections; }

}

//printf(", "); // DEBUG

}

//if(longest\_black\_section > 10 && (line\_center\_temp > 5 && line\_center\_temp < 122) ){ line\_center = line\_center\_temp; }

if(num\_long\_black\_sections == 3 && !intersect\_seen){

if(!prev\_intersect\_seen){

stop\_line\_seen = 1;

++track\_laps;

}

prev\_intersect\_seen = 1;

}else if(longest\_black\_section > 100 && !stop\_line\_seen){

intersect\_seen = 1;

prev\_intersect\_seen = 0;

}else{

intersect\_seen = 0;

stop\_line\_seen = 0;

prev\_intersect\_seen = 0;

}

/\*printf("\nlongest\_black\_section: %d\nnum\_long\_black\_sections: %d\nintersect\_seen: %d\nstop\_line\_seen: %d\n\n",

longest\_black\_section, num\_long\_black\_sections, intersect\_seen, stop\_line\_seen); // DEBUG

\*/

}

int lineVisible(){ return longest\_black\_section > 10 ? 1 : 0; }

int initializeXADC(){

SYSConfigPtr = XSysMon\_LookupConfig(SYSMON\_DEVICE\_ID);

if (SYSConfigPtr == NULL) { return XST\_FAILURE; }

XSysMon\_CfgInitialize(SysMonInstPtr, SYSConfigPtr, SYSConfigPtr->BaseAddress);

address = SYSConfigPtr->BaseAddress;

printf("XADC using Sysmon -> base address %lx \n\r", address);

XADCConfigPtr = XAdcPs\_LookupConfig(PS7\_XADC\_DEVICE\_ID);

if (XADCConfigPtr == NULL) { return XST\_FAILURE+1; }

XAdcPs\_CfgInitialize(XADCInstPtr,XADCConfigPtr,XADCConfigPtr->BaseAddress);

address = XADCConfigPtr->BaseAddress;

printf("XADC using XADPS -> base address %lx \n\r", address);

return 0;

}

u32 singleReading(){

XSysMon\_StartAdcConversion(&SysMonInst);

return(XSysMon\_GetAdcData(&SysMonInst, XADCPS\_AUX14\_OFFSET));

}

#endif /\* SRC\_LINESCAN\_H\_ \*/

#### Timer\_and\_pwm.h

#ifndef SRC\_TIMER\_AND\_PWM\_H\_

#define SRC\_TIMER\_AND\_PWM\_H\_

#include "Xscugic.h"

#include "xttcps.h"

#include "xadcps.h"

#include "xsysmon.h"

#define TTC\_DEVICE\_ID\_0 XPAR\_XTTCPS\_0\_DEVICE\_ID

#define TTC\_INTR\_ID\_0 XPAR\_XTTCPS\_0\_INTR

#define TTC\_DEVICE\_ID\_1 XPAR\_XTTCPS\_1\_DEVICE\_ID

#define TTC\_INTR\_ID\_1 XPAR\_XTTCPS\_1\_INTR

#define INTC\_DEVICE\_ID XPAR\_SCUGIC\_SINGLE\_DEVICE\_ID

static void initializeTimerSystem();

static int genTimer(int freq, XTtcPs \*timer, void (\*Handler)(void \*));

static int genPwm(int freq, double duty\_cycle, XTtcPs \*timer);

static void updatePwmDutyCycle(double duty\_cycle, XTtcPs \*Timer);

static void SetupInterruptTimer(XScuGic \*GicInstancePtr, XTtcPs \*TtcPsInt, unsigned long TTC\_INTR\_ID, void (\*Handler)(void \*));

static void TimerInit(unsigned long TTC\_DEVICE\_ID, u32 freq, XTtcPs \*Timer, unsigned long TTC\_INTR\_ID, void (\*Handler)(void \*));

static void GicSetup(XScuGic \*GicInstancePtr);

static void pwmHandler(void \*CallBackRef);

int used\_timer\_counter, used\_match\_counter;

//int pwm\_wave\_value;

static XScuGic Intc; //GIC

static void initializeTimerSystem(){

used\_timer\_counter = 0;

}

static int genPwm(int freq, double duty\_cycle, XTtcPs \*timer){

u16 options;

int ret;

ret = genTimer(freq, timer, &pwmHandler);

if (ret < 0) { return ret; }

XTtcPs\_EnableInterrupts(timer, XTTCPS\_IXR\_MATCH\_0\_MASK);

options = XTtcPs\_GetOptions(timer) | XTTCPS\_OPTION\_MATCH\_MODE;

XTtcPs\_SetOptions(timer, options);

updatePwmDutyCycle(duty\_cycle, timer);

return 0;

}

static void pwmHandler(void \*CallBackRef)

{

u32 StatusEvent;

StatusEvent = XTtcPs\_GetInterruptStatus((XTtcPs \*)CallBackRef);

XTtcPs\_ClearInterruptStatus((XTtcPs \*)CallBackRef, StatusEvent);

/\*if (0 != (XTTCPS\_IXR\_INTERVAL\_MASK & StatusEvent)) {

//printf("interval interrupt event\n\r");

pwm\_wave\_value = 1;

}

if (0 != (XTTCPS\_IXR\_MATCH\_0\_MASK & StatusEvent)) {

//printf("match interrupt event\n\r");

pwm\_wave\_value = 0;

}

printf("Wave Value: %d\n\r", pwm\_wave\_value);\*/

}

static void updatePwmDutyCycle(double duty\_cycle, XTtcPs \*Timer){

u16 interval; /\* Interval value \*/

interval = XTtcPs\_GetInterval(Timer);

XTtcPs\_SetMatchValue(Timer, 0, (int)(interval\*duty\_cycle/100));

}

static int genTimer(int freq, XTtcPs \*timer, void (\*handler)(void \*)){

switch(used\_timer\_counter){

case 0:

TimerInit(XPAR\_XTTCPS\_0\_DEVICE\_ID, freq, timer, XPAR\_XTTCPS\_0\_INTR, handler);

break;

case 1:

TimerInit(XPAR\_XTTCPS\_1\_DEVICE\_ID, freq, timer, XPAR\_XTTCPS\_1\_INTR, handler);

break;

case 2:

TimerInit(XPAR\_XTTCPS\_2\_DEVICE\_ID, freq, timer, XPAR\_XTTCPS\_2\_INTR, handler);

break;

case 3:

//TimerInit(XPAR\_XTTCPS\_3\_DEVICE\_ID, freq, timer, XPAR\_XTTCPS\_3\_INTR, handler);

break;

case 4:

//TimerInit(XPAR\_XTTCPS\_4\_DEVICE\_ID, freq, timer, XPAR\_XTTCPS\_4\_INTR, handler);

break;

case 5:

//TimerInit(XPAR\_XTTCPS\_5\_DEVICE\_ID, freq, timer, XPAR\_XTTCPS\_5\_INTR, handler);

break;

default:

if (used\_timer\_counter > 5){ return -1; }

else if (used\_timer\_counter < 0){ return -2; }

}

++used\_timer\_counter;

return used\_timer\_counter;

}

static void TimerInit(unsigned long TTC\_DEVICE\_ID, u32 freq, XTtcPs \*Timer, unsigned long TTC\_INTR\_ID, void (\*Handler)(void \*)){

u16 interval;

u8 prescaler;

u16 options;

XTtcPs\_Config \*Config;

XTtcPs\_Stop(Timer); // This may have to go before GicSetup

Config = XTtcPs\_LookupConfig(TTC\_DEVICE\_ID);

XTtcPs\_CfgInitialize(Timer, Config, Config->BaseAddress);

options = XTTCPS\_OPTION\_INTERVAL\_MODE | XTTCPS\_OPTION\_WAVE\_POLARITY;

XTtcPs\_SetOptions(Timer, options);

XTtcPs\_CalcIntervalFromFreq(Timer, freq, &interval, &prescaler);

XTtcPs\_SetInterval(Timer, interval);

XTtcPs\_SetPrescaler(Timer, prescaler);

SetupInterruptTimer(&Intc, Timer, TTC\_INTR\_ID, Handler);

printf("%d\r\n",interval);

printf("%d\r\n",prescaler);

}

static void GicSetup(XScuGic \*GicInstancePtr){

XScuGic\_Config \*IntcConfig; // GIC config

Xil\_ExceptionInit();

// Init GIC

IntcConfig = XScuGic\_LookupConfig(INTC\_DEVICE\_ID);

XScuGic\_CfgInitialize(GicInstancePtr, IntcConfig,

IntcConfig->CpuBaseAddress);

// Connect to the hardware

Xil\_ExceptionRegisterHandler(XIL\_EXCEPTION\_ID\_INT,

(Xil\_ExceptionHandler)XScuGic\_InterruptHandler,

GicInstancePtr);

}

static void SetupInterruptTimer(XScuGic \*GicInstancePtr, XTtcPs \*TtcPsInt, unsigned long TTC\_INTR\_ID, void (\*Handler)(void \*))

{

XScuGic\_Connect(GicInstancePtr, TTC\_INTR\_ID,

(Xil\_ExceptionHandler)Handler, (void \*)TtcPsInt);

XScuGic\_Enable(GicInstancePtr, TTC\_INTR\_ID);

XTtcPs\_EnableInterrupts(TtcPsInt, XTTCPS\_IXR\_INTERVAL\_MASK);

XTtcPs\_Start(TtcPsInt);

}

#endif /\* SRC\_TIMER\_AND\_PWM\_H\_ \*/

#### Buttons\_and\_leds.h

#ifndef SRC\_BUTTONS\_AND\_LEDS\_H\_

#define SRC\_BUTTONS\_AND\_LEDS\_H\_

//#include "xscugic.h"

//#include "xgpio.h"

//#include "xil\_exception.h"

//#include "sleep.h"

// Parameter definitions

#define INTC\_DEVICE\_ID XPAR\_PS7\_SCUGIC\_0\_DEVICE\_ID

#define BTNS\_DEVICE\_ID XPAR\_AXI\_GPIO\_0\_DEVICE\_ID

#define LEDS\_DEVICE\_ID XPAR\_AXI\_GPIO\_1\_DEVICE\_ID

#define INTC\_GPIO\_INTERRUPT\_ID XPAR\_FABRIC\_AXI\_GPIO\_0\_IP2INTC\_IRPT\_INTR

#define BTN\_INT XGPIO\_IR\_CH1\_MASK

int initButtonsAndLeds();

static void BTN\_Intr\_Handler(void \*baseaddr\_p);

static int InterruptSystemSetup(XScuGic \*XScuGicInstancePtr);

static int IntcInitFunction(u16 DeviceId, XGpio \*GpioInstancePtr);

XGpio LEDInst, BTNInst;

XScuGic INTCInst;

static int btn\_value;

int go\_flag = 0;

int mode\_count = 0;

int initButtonsAndLeds(){

int status;

// Initialize LEDs

status = XGpio\_Initialize(&LEDInst, LEDS\_DEVICE\_ID);

if (status != XST\_SUCCESS)

return XST\_FAILURE;

// Initialize Push Buttons

status = XGpio\_Initialize(&BTNInst, BTNS\_DEVICE\_ID);

if (status != XST\_SUCCESS){ return XST\_FAILURE; }

// Set LEDs direction to outputs

XGpio\_SetDataDirection(&LEDInst, 1, 0x00);

// Set all buttons direction to inputs

XGpio\_SetDataDirection(&BTNInst, 1, 0xFF);

// Initialize interrupt controller

status = IntcInitFunction(INTC\_DEVICE\_ID, &BTNInst);

if (status != XST\_SUCCESS){ return XST\_FAILURE; }

return 0;

}

void BTN\_Intr\_Handler(void \*InstancePtr) {

// Disable GPIO interrupts

XGpio\_InterruptDisable(&BTNInst, BTN\_INT);

// Ignore additional button presses

if ((XGpio\_InterruptGetStatus(&BTNInst) & BTN\_INT) != BTN\_INT) {

return;

}

btn\_value = XGpio\_DiscreteRead(&BTNInst, 1);

if (btn\_value == 0b0001){

if (go\_flag == 0){

go\_flag = 1;

} else {

go\_flag = 0;

}

xil\_printf("button 0 pressed\n\r");

}

else if(btn\_value == 0b0010)

{

if (mode\_count == 6) {

mode\_count = 0;

} else {

mode\_count = mode\_count + 1;

}

xil\_printf("button 1 pressed\n\r");

}

(void) XGpio\_InterruptClear(&BTNInst, BTN\_INT);

// Enable GPIO interrupts

usleep(200000);

XGpio\_InterruptEnable(&BTNInst, BTN\_INT);

}

//----------------------------------------------------

// INITIAL SETUP FUNCTIONS

//----------------------------------------------------

int InterruptSystemSetup(XScuGic \*XScuGicInstancePtr) {

// Enable interrupt

XGpio\_InterruptEnable(&BTNInst, BTN\_INT);

XGpio\_InterruptGlobalEnable(&BTNInst);

Xil\_ExceptionRegisterHandler(XIL\_EXCEPTION\_ID\_INT,

(Xil\_ExceptionHandler) XScuGic\_InterruptHandler,

XScuGicInstancePtr);

Xil\_ExceptionEnable();

return XST\_SUCCESS;

}

int IntcInitFunction(u16 DeviceId, XGpio \*GpioInstancePtr) {

XScuGic\_Config \*IntcConfig;

int status;

// Interrupt controller initialisation

IntcConfig = XScuGic\_LookupConfig(DeviceId);

status = XScuGic\_CfgInitialize(&INTCInst, IntcConfig,

IntcConfig->CpuBaseAddress);

if (status != XST\_SUCCESS)

return XST\_FAILURE;

// Call to interrupt setup

status = InterruptSystemSetup(&INTCInst);

if (status != XST\_SUCCESS)

return XST\_FAILURE;

// Connect GPIO interrupt to handler

status = XScuGic\_Connect(&INTCInst, INTC\_GPIO\_INTERRUPT\_ID,

(Xil\_ExceptionHandler) BTN\_Intr\_Handler, (void \*) GpioInstancePtr);

if (status != XST\_SUCCESS)

return XST\_FAILURE;

// Enable GPIO interrupts interrupt

XGpio\_InterruptEnable(GpioInstancePtr, 1);

XGpio\_InterruptGlobalEnable(GpioInstancePtr);

// Enable GPIO and timer interrupts in the controller

XScuGic\_Enable(&INTCInst, INTC\_GPIO\_INTERRUPT\_ID);

return XST\_SUCCESS;

}

#endif /\* SRC\_BUTTONS\_AND\_LEDS\_H\_ \*/

#### Pixy.h

#ifndef SRC\_PIXY\_H\_

#define SRC\_PIXY\_H\_

#endif /\* SRC\_PIXY\_H\_ \*/

### VHDL

-- author: milton griffin

-- email: griffimj@clarkson.edu

-- file: pixy spi

-- description:

-- vhdl inteface for communication with the

-- Pixy Cam over SPI

-- version: 0.10

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity pixy\_spi is

generic(

clock\_speed : integer := 100; -- in MHz

spi\_clock\_speed : integer := 1 -- in MHz

);

Port (

-- internal

clk : in std\_logic;

reset\_n : in std\_logic;

data\_change : out std\_logic;

signature : out std\_logic\_vector(4 downto 0);

xpos : out std\_logic\_vector(8 downto 0);

height : out std\_logic\_vector(7 downto 0);

-- external

spi\_miso : in std\_logic;

spi\_mosi : out std\_logic;

spi\_clk : out std\_logic

);

end pixy\_spi;

architecture Behavioral of pixy\_spi is

type array7x16 is array(6 downto 0) of std\_logic\_vector(15 downto 0);

signal main\_reg : array7x16;

signal reg\_index : integer;

signal mo\_data : std\_logic\_vector(15 downto 0):= x"5a00";

signal shift\_reg : std\_logic\_vector(15 downto 0);

signal div\_clk : std\_logic;

signal div\_clk\_counter : integer;

signal spi\_control : integer;

signal spi\_clk\_ex : std\_logic;

begin

signature <= main\_reg(0)(4 downto 0);

xpos <= main\_reg(1)(8 downto 0);

height <= main\_reg(4)(7 downto 0);

clock\_divider:process(clk,reset\_n)

begin

if(reset\_n = '0')

then

div\_clk <= '0';

div\_clk\_counter <= 0;

else

if(rising\_edge(clk))

then

if(div\_clk\_counter >= clock\_speed/spi\_clock\_speed/2)

then

div\_clk <= not div\_clk;

div\_clk\_counter <= 0;

else

div\_clk\_counter <= div\_clk\_counter + 1;

end if;

end if;

end if;

end process;

data\_builder:process(reset\_n,div\_clk,spi\_control,shift\_reg)

begin

if(reset\_n = '0')

then

main\_reg <= (others => x"0000");

else

if(rising\_edge(div\_clk))

then

if(spi\_control = 19 and reg\_index <= 7)

then

main\_reg(reg\_index - 1) <= shift\_reg;

end if;

if(spi\_control = 20 and reg\_index = 7)

then

data\_change <= '1';

else

data\_change <= '0';

end if;

end if;

end if;

end process;

reg\_index\_controler:process(reset\_n,div\_clk,spi\_control)

begin

if(reset\_n = '0')

then

reg\_index <= 0;

else

if(rising\_edge(div\_clk) and spi\_control = 19)

then

if(reg\_index <= 8)

then

reg\_index <= reg\_index + 1;

elsif(shift\_reg = x"aa56")

then

reg\_index <= 0;

end if;

end if;

end if;

end process;

spi\_controler:process(reset\_n,div\_clk,spi\_control)

begin

if(reset\_n = '0')

then

spi\_control <= 0;

else

if(rising\_edge(div\_clk))

then

if(spi\_control < 22)

then

spi\_control <= spi\_control + 1;

else

spi\_control <= 0;

end if;

end if;

end if;

end process;

spi\_mosi\_control:process(clk,reset\_n,div\_clk,spi\_control)

begin

if(reset\_n = '0')

then

spi\_mosi <= '1';

else

if(falling\_edge(div\_clk))

then

if(spi\_control < 7)

then

spi\_mosi <= mo\_data(spi\_control);

elsif(spi\_control >= 8 and spi\_control <= 16)

then

spi\_mosi <= mo\_data(spi\_control - 9);

else

spi\_mosi <= '1';

end if;

End if;

end if;

end process;

spi\_miso\_control:process(reset\_n,div\_clk,spi\_clk\_ex,spi\_control,shift\_reg)

begin

if(reset\_n = '0')

then

shift\_reg <= x"0000";

else

if(rising\_edge(spi\_clk\_ex))

then

shift\_reg <= shift\_reg(14 downto 0) & spi\_miso;

end if;

end if;

end process;

spi\_clock\_control:process(reset\_n,div\_clk,spi\_control)

begin

if(reset\_n = '0')

then

spi\_clk <= '0';

else

if((spi\_control >= 1 and spi\_control <= 8) or (spi\_control >= 10 and

spi\_control <= 17))

then

spi\_clk\_ex <= div\_clk;

else

spi\_clk\_ex <= '0';

end if;

end if;

end process;

end Behavioral;

### Constraints File

set\_property PACKAGE\_PIN V15 [get\_ports TTC0\_WAVE0\_OUT]

set\_property PACKAGE\_PIN W15 [get\_ports TTC0\_WAVE1\_OUT]

set\_property PACKAGE\_PIN T11 [get\_ports TTC0\_WAVE2\_OUT]

set\_property IOSTANDARD LVCMOS33 [get\_ports TTC0\_WAVE0\_OUT]

set\_property IOSTANDARD LVCMOS33 [get\_ports TTC0\_WAVE1\_OUT]

set\_property IOSTANDARD LVCMOS33 [get\_ports TTC0\_WAVE2\_OUT]

set\_property PACKAGE\_PIN T10 [get\_ports TTC1\_WAVE0\_OUT]

set\_property PACKAGE\_PIN W14 [get\_ports TTC1\_WAVE1\_OUT]

set\_property PACKAGE\_PIN Y14 [get\_ports TTC1\_WAVE2\_OUT]

set\_property IOSTANDARD LVCMOS33 [get\_ports TTC1\_WAVE0\_OUT]

set\_property IOSTANDARD LVCMOS33 [get\_ports TTC1\_WAVE1\_OUT]

set\_property IOSTANDARD LVCMOS33 [get\_ports TTC1\_WAVE2\_OUT]

set\_property PACKAGE\_PIN N16 [get\_ports Vaux14\_v\_n]

set\_property PACKAGE\_PIN N15 [get\_ports Vaux14\_v\_p]

set\_property IOSTANDARD LVCMOS33 [get\_ports Vaux14\_v\_n]

set\_property IOSTANDARD LVCMOS33 [get\_ports Vaux14\_v\_p]

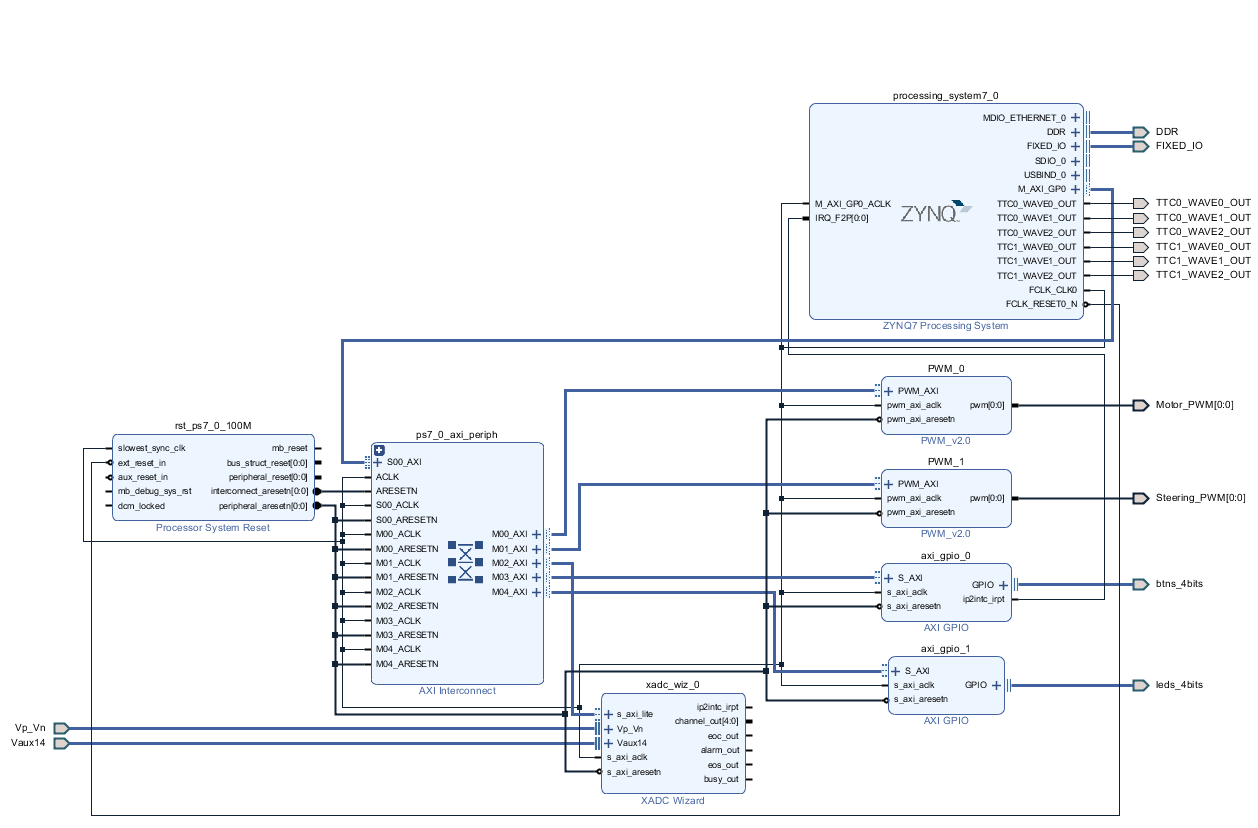
set\_property PACKAGE\_PIN T20 [get\_ports {Motor\_PWM[0]}]

set\_property PACKAGE\_PIN U20 [get\_ports {Steering\_PWM[0]}]

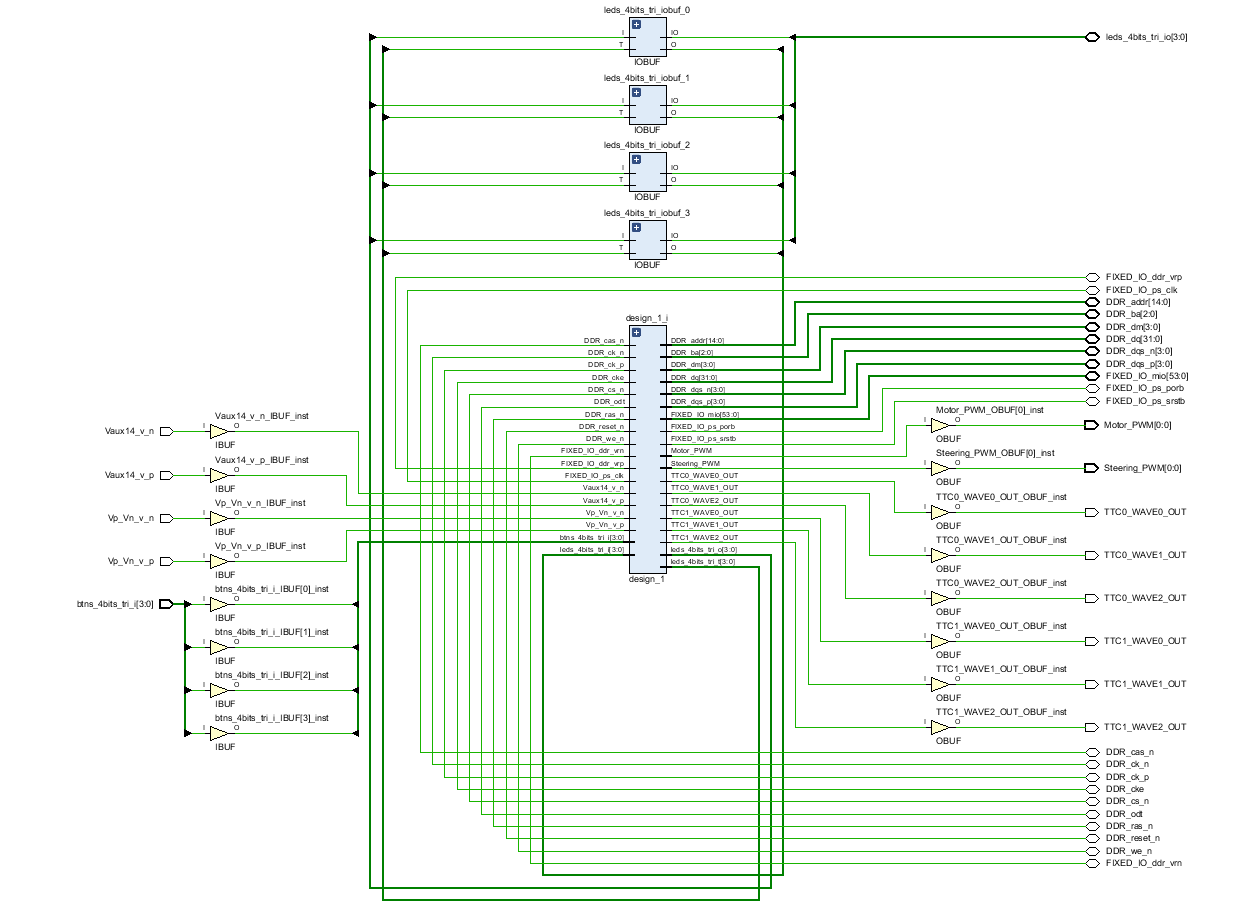
set\_property IOSTANDARD LVCMOS33 [get\_ports {Motor\_PWM[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {Steering\_PWM[0]}]

### Schematics



Block Diagram



Schematic

## FRDM

This section contains all of the code and schematics for the Zybo board.

### C code

#### Main.c

#include <stdio.h>

#include <stdint.h>

#include <stdlib.h> // exit

#include "fsl\_debug\_console.h"

#include "board.h"

#include "pin\_mux.h"

#include "clock\_config.h"

#include "modes.h"

#include "timer\_and\_pwm.h"

#include "motor\_and\_steering.h"

#include "linescan.h"

#include "pixy.h"

#include "lidar.h"

#include "buttons\_and\_leds.h"

//#include "sd\_card.h"

void line\_scan\_init(){

// Init PWMs

setupLineScanTimers();

int status = initializeXADC();

if(status != 0){

printf("Error in XADC initialization: %d", status);

exit(1);

}

}

void motor\_steering\_init(){

// Init PWMs

initSteerAndMotor();

}

void pixy\_init(){

initPixy();

}

void lidar\_init(){

}

void buttons\_and\_leds\_init(){

initButtonAndLEDs();

}

//void sd\_card\_init(){

// //init\_sd\_card();

//}

void initializeSystem(){

//printf("Initializing Car\n\r");

/\* Board pin, clock, debug console init \*/

BOARD\_InitPins();

BOARD\_BootClockRUN();

BOARD\_InitDebugConsole();

//SYSMPU\_Enable(SYSMPU, false);

modeAction = doNothingMode;

motor\_steering\_init();

line\_scan\_init();

pixy\_init();

lidar\_init();

buttons\_and\_leds\_init();

//sd\_card\_init();

initializeTimerSystem();

// Remove warnings

if(mode\_finished){mode\_finished = 1;}

}

int main(void)

{

LED\_RED\_INIT(1U);

LED\_BLUE\_INIT(1U);

LED\_GREEN\_INIT(1U);

int count = 0;

starting = 1;

initializeSystem();

while(1){

while(!go\_flag){setMode();}

setMode();

go\_flag = 0;

while(!mode\_finished){

if(count==5){

modeAction();

count = -1;

}

++count;

pixyRead();

delay(10000);//60000

}

modeAction = doNothingMode;

modeAction();

track\_laps = 0;

mode\_finished = 0;

}

return 0;

}

#### Modes.h

#ifndef SRC\_MODES\_H\_

#define SRC\_MODES\_H\_

#include "pixy.h"

#include "motor\_and\_steering.h"

#include "lidar.h"

#include "linescan.h"

#include "buttons\_and\_leds.h"

static void setMode();

static void doNothingMode();

static void basicMode();

static void accuracyMode();

static void speedMode();

static void collisionMode();

static void discoveryMode();

static void manualMode();

static void turnRightSlow();

static void turnLeftSlow();

static void turnAroundSlow();

static void turnRightFast();

static void turnLeftFast();

static void turnAroundFast();

void (\*modeAction)(void);

uint8\_t mode\_finished = 0;

int speed = 24;

extern int marker\_sigs[2];

extern int intersect\_seen;

uint8\_t turning = 0;

int turn\_count = -1;

int repetitions = -1;

extern int track\_laps;

uint8\_t car\_ahead = 0;

uint8\_t line\_seen = 0;

int starting = 0;

static void setMode(){

if (mode\_count == 0){

modeAction = doNothingMode;

LED\_RED\_ON();

LED\_GREEN\_OFF(); //Red

LED\_BLUE\_OFF();

}else if (mode\_count == 1){

modeAction = basicMode;

speed = 22;

LED\_RED\_ON();

LED\_GREEN\_ON(); //White

LED\_BLUE\_ON();

}else if (mode\_count == 2){

modeAction = accuracyMode;

speed = 24;

LED\_RED\_OFF();

LED\_GREEN\_OFF(); //Blue

LED\_BLUE\_ON();

}else if (mode\_count == 3){

modeAction = speedMode;

speed = 26;

LED\_RED\_OFF();

LED\_GREEN\_ON(); //Green

LED\_BLUE\_OFF();

}else if (mode\_count == 4){

modeAction = collisionMode;

speed = 28;

LED\_RED\_ON();

LED\_GREEN\_ON(); //Yellow

LED\_BLUE\_OFF();

}else if (mode\_count == 5){

modeAction = discoveryMode;

LED\_RED\_ON();

LED\_GREEN\_OFF(); //Purple

LED\_BLUE\_ON();

}else if (mode\_count == 6){

modeAction = manualMode;

LED\_RED\_OFF();

LED\_GREEN\_ON(); //Cyan

LED\_BLUE\_ON();

}

}

void turnRightSlow(){

//static int turn\_actions[3][3] = {{0, -90, 0}, {1, 1, 1}, {2, 100, 2}}; // [[tire angles], [speeds], [repetitions]]

static int turn\_actions[3][1] = {{-90}, {22}, {110}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 1){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}

else{

--repetitions;

}

}

else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

void turnLeftSlow(){

//static int turn\_actions[3][3] = {{0, -90, 0}, {1, 1, 1}, {2, 100, 2}}; // [[tire angles], [speeds], [repetitions]]

static int turn\_actions[3][1] = {{90}, {22}, {110}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 1){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}

else{

--repetitions;

}

}

else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

void turnAroundSlow(){

// TODO: change the speeds and the number of repetitions

//static int turn\_actions[3][3] = {{0, -90, 0}, {1, 1, 1}, {2, 100, 2}}; // [[tire angles], [speeds], [repetitions]]

static int turn\_actions[3][3] = {{-90, 90, 0}, {20, 25, 1}, {100, 200, 40}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 3){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (turn\_count == 1){

setReverseDir();

}

if (turn\_count == 2){

setForwardDir();

}

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}

else{

--repetitions;

}

}

else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

void turnRightFast(){

//static int turn\_actions[3][3] = {{0, -90, 0}, {1, 1, 1}, {2, 100, 2}}; // [[tire angles], [speeds], [repetitions]]

static int turn\_actions[3][1] = {{-90}, {22}, {140}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 1){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}

else{

--repetitions;

}

}

else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

void turnLeftFast(){

//static int turn\_actions[3][3] = {{0, -90, 0}, {1, 1, 1}, {2, 100, 2}}; // [[tire angles], [speeds], [repetitions]]

static int turn\_actions[3][2] = {{90, -90}, {22, 16}, {140, 30}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 1){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}

else{

--repetitions;

}

}

else{

turn\_count = -1;

repetitions = -1;

turning = 0;

}

}

void turnAroundFast(){

// THIS WILL NOT BE TESTED

turn\_count = -1;

repetitions = -1;

turning = 0;

}

void doNothingMode(){

speedSet(0);

steerSet(0);

}

void basicMode(){

pixyRead();

if(turning){

if(marker\_sigs[0]){

if(marker\_sigs[1]){

turnAroundSlow();

}

else{

turnLeftSlow();

}

}

else{

if(marker\_sigs[1]){

turnRightSlow();

}

else{

turning = 0;

}

}

}

else{

steerSet(steerPidOut());

speedSet(speed);

if(markers\_opposite){

if(intersect\_seen){

markers\_opposite = 0;

if(!markers\_visible){

turning = 1;

}

}

else{

if(speed > 24){

speed -= 1;

}

}

}

else{

if(speed < 26){

speed +=1;

}

}

}

}

void accuracyMode(){

if(turning){

if(marker\_sigs[0]){

if(marker\_sigs[1]){

turnAroundSlow();

}

else{

turnLeftSlow();

}

}

else{

if(marker\_sigs[1]){

turnRightSlow();

}

else{

turning = 0;

}

}

}

else{

steerSet(steerPidOut());

speedSet(speed);

if(track\_laps >= 2){

mode\_finished = 1;

modeAction = doNothingMode;

}

if(markers\_opposite){

if(intersect\_seen){

markers\_opposite = 0;

if(!markers\_visible){

turning = 1;

}

}

else{

if(speed > 22){

speed -= 1;

}

}

}

else{

if(speed < 22){

speed +=1;

};

}

}

}

void speedMode(){

if(turning){

if(marker\_sigs[0]){

if(marker\_sigs[1]){

turnAroundFast();

}

else{

turnLeftFast();

}

}

else{

if(marker\_sigs[1]){

turnRightFast();

}

else{

turning = 0;

}

}

}

else{

steerSet(steerPidOut());

speedSet(speed);

if(track\_laps >= 2){

mode\_finished = 1;

}

if(markers\_opposite){

if(intersect\_seen){

markers\_opposite = 0;

if(!markers\_visible){

turning = 1;

}

}

else{

if(speed > 24){

speed -= 1;

}

}

}

else{

if(isOnStraightAway()){

if(speed < 50){

speed += 1;

}

}

else{

if(speed > 26){

speed -= 1;

}

}

}

}

}

void collisionMode(){

if (starting){

starting = 0;

collisionModeSpeed(100);

}

if(turning){

if(marker\_sigs[0]){

if(!marker\_sigs[1]){

turnLeftSlow();

}

}

else{

if(marker\_sigs[1]){

turnRightSlow();

}

else{

turning = 0;

}

}

}else{

steerSet(steerPidOut());

if(track\_laps >= 2){

mode\_finished = 1;

}

if(markers\_opposite){

if(intersect\_seen){

markers\_opposite = 0;

if(!markers\_visible){

turning = 1;

}

}

}

}

// if (masterRxInfo[0][1] == 28 && masterRxData[0] == 0){

// //collisionModeSpeed(25);

// }

// else if (masterRxInfo[0][1] == 28 || masterRxInfo[1][1] == 28){

// collisionModeSpeed(distance);

// }

}

void discoveryMode(){

// TODO: change the speeds, last tire pos, and the number of repetitions

//static int turn\_actions[3][2] = {{90, 45}, {28, 28}, {180, 500}}; // [[tire angles], [speeds], [repetitions]]

static int turn\_actions[3][2] = {{90, 45}, {28, 28}, {301, 1000}}; // [[tire angles], [speeds], [repetitions]]

if(turn\_count == -1 && repetitions == -1){

turn\_count = 0;

repetitions = turn\_actions[2][0];

}

if (turn\_count < 2){

steerSet(turn\_actions[0][turn\_count]);

speedSet(turn\_actions[1][turn\_count]);

if (repetitions <= 0){

++turn\_count;

repetitions = turn\_actions[2][turn\_count];

}else{

--repetitions;

}

}else{

turn\_count = -1;

repetitions = -1;

modeAction = doNothingMode;

}

if(lineVisible() && turn\_count > 0){

modeAction = accuracyMode;

}

}

void manualMode(){

//steerSet(bluetooth\_tire\_pos());

//speedSet(bluetooth\_speed());

}

#endif /\* SRC\_MODES\_H\_ \*/

#### Motor\_and\_steering.h

#ifndef SRC\_MOTOR\_AND\_STEERING\_H\_

#define SRC\_MOTOR\_AND\_STEERING\_H\_

#include "timer\_and\_pwm.h"

#include <math.h>

/\* Interrupt number and interrupt handler for the FTM instance used \*/

#define SteeringPWM\_HANDLER FTM0\_IRQHandler

ftm\_config\_t ftmInfo3;

ftm\_config\_t ftmInfo4;

ftm\_chnl\_pwm\_signal\_param\_t ftmParam3;

ftm\_chnl\_pwm\_signal\_param\_t ftmParam4;

gpio\_pin\_config\_t dir\_config = { // steering

kGPIO\_DigitalOutput, 0

};

double cSpeed = 30;

void initSteerAndMotor(){

ftmParam3.chnlNumber = kFTM\_Chnl\_1; //Steering Servo

ftmParam3.level = kFTM\_HighTrue;

ftmParam3.dutyCyclePercent = 7U;

ftmParam3.firstEdgeDelayPercent = 0U;

ftmParam4.chnlNumber = kFTM\_Chnl\_4; //Motor

ftmParam4.level = kFTM\_HighTrue;

ftmParam4.dutyCyclePercent = 0U;

ftmParam4.firstEdgeDelayPercent = 0U;

FTM\_GetDefaultConfig(&ftmInfo3);

FTM\_GetDefaultConfig(&ftmInfo4);

ftmInfo3.prescale = kFTM\_Prescale\_Divide\_128; //must be done after default config

/\* Initialize FTM module \*/

FTM\_Init(FTM0, &ftmInfo3);

FTM\_Init(FTM3, &ftmInfo4);

FTM\_SetupPwm(FTM0, &ftmParam3, 1U, kFTM\_EdgeAlignedPwm, 50U, FTM\_SOURCE\_CLOCK);

FTM\_SetupPwm(FTM3, &ftmParam4, 1U, kFTM\_EdgeAlignedPwm, 20000U, FTM\_SOURCE\_CLOCK);

GPIO\_PinInit(GPIOC, 1U, &dir\_config); //Initialize GPIO Pin PTC1

}

void updateMotorPwm(double duty\_cycle){

/\* Disable interrupt to retain current dutycycle for a few seconds \*/

//FTM\_DisableInterrupts(FTM3, kFTM\_Chnl4InterruptEnable);

/\* Disable channel output before updating the dutycycle \*/

FTM\_UpdateChnlEdgeLevelSelect(FTM3, kFTM\_Chnl\_4, 0U);

/\* Update PWM duty cycle \*/

FTM\_UpdateDutyCycleDec(FTM3, kFTM\_Chnl\_4, kFTM\_EdgeAlignedPwm, duty\_cycle);

/\* Software trigger to update registers \*/

//FTM\_SetSoftwareTrigger(FTM3, true);

/\* Start channel output with updated dutycycle \*/

FTM\_UpdateChnlEdgeLevelSelect(FTM3, kFTM\_Chnl\_4, kFTM\_HighTrue);

/\* Enable interrupt flag to update PWM dutycycle \*/

//FTM\_EnableInterrupts(FTM3, kFTM\_Chnl4InterruptEnable);

}

void updateSteerPwm(double duty\_cycle){

/\* Disable interrupt to retain current dutycycle for a few seconds \*/

FTM\_DisableInterrupts(FTM0, kFTM\_Chnl1InterruptEnable);

/\* Disable channel output before updating the dutycycle \*/

FTM\_UpdateChnlEdgeLevelSelect(FTM0, kFTM\_Chnl\_1, 0U);

/\* Update PWM duty cycle \*/

FTM\_UpdateDutyCycleDec(FTM0, kFTM\_Chnl\_1, kFTM\_EdgeAlignedPwm, duty\_cycle);

/\* Software trigger to update registers \*/

FTM\_SetSoftwareTrigger(FTM0, true);

/\* Start channel output with updated dutycycle \*/

FTM\_UpdateChnlEdgeLevelSelect(FTM0, kFTM\_Chnl\_1, kFTM\_HighTrue);

/\* Enable interrupt flag to update PWM dutycycle \*/

FTM\_EnableInterrupts(FTM0, kFTM\_Chnl1InterruptEnable);

}

void speedSet(double speed){ // speed between 0 and 100

updateMotorPwm(speed);

}

void steerSet(double angle){ // angle between -90 and 90

//int interval = 5; int center = 7.5;

//int duty\_cycle\_per\_angle = interval / 180;

//int match = duty\_cycle\_per\_angle \* angle + center;

updateSteerPwm(duty\_cycle\_per\_angle \* angle + 7.5);

}

void SteeringPWM\_HANDLER(void)

{

if ((FTM\_GetStatusFlags(FTM0) & kFTM\_Chnl1Flag) == kFTM\_Chnl1Flag)

{

/\* Clear interrupt flag.\*/

FTM\_ClearStatusFlags(FTM0, kFTM\_Chnl1Flag);

}

}

void setForwardDir(){

GPIO\_WritePinOutput(GPIOC, 1U, 0U);

}

void setReverseDir(){

GPIO\_WritePinOutput(GPIOC, 1U, 1U);

}

void collisionModeSpeed(double distance){

if (distance > 36)

distance = 36;

else if (distance < 15)

distance = 15;

cSpeed = (-0.016 \* pow(distance, 3)) + (1.0895 \* pow(distance, 2)) - (21.06 \* distance) + 124.11;

if (cSpeed > 30)

cSpeed = 30;

else if (cSpeed < 0)

cSpeed = 0 ;

speedSet(cSpeed);

}

#endif /\* SRC\_MOTOR\_AND\_STEERING\_H\_ \*/

#### Linescan.h

#ifndef SRC\_LINESCAN\_H\_

#define SRC\_LINESCAN\_H\_

#include "fsl\_adc16.h"

#include "timer\_and\_pwm.h"

/\* Interrupt number and interrupt handler for the FTM instance used \*/

//#define FTM\_INTERRUPT\_NUMBER FTM3\_IRQn

#define FTM\_LineScan\_Handler FTM3\_IRQHandler

/\* ADC ADC ADC ADC ADC ADC ADC ADC ADC \*/

#define DEMO\_ADC16\_BASE ADC0

#define DEMO\_ADC16\_CHANNEL\_GROUP 0U

#define DEMO\_ADC16\_USER\_CHANNEL 12U /\* PTB2, ADC0\_SE12 \*/

ftm\_config\_t ftmInfo1;

ftm\_config\_t ftmInfo2;

ftm\_chnl\_pwm\_signal\_param\_t ftmParam1;

ftm\_chnl\_pwm\_signal\_param\_t ftmParam2;

ftm\_pwm\_level\_select\_t pwmLevel = kFTM\_LowTrue;

volatile bool ftm\_is\_r\_flag = false;

volatile uint8\_t num\_cycles = 0U;

volatile uint16\_t num\_outputs = 0U;

volatile double updated\_duty\_cycle = 7.5;

volatile double prev\_duty\_cycle = 7.5;

volatile int8\_t line\_error = 0;

volatile int8\_t line\_error\_prev = 0;

volatile int8\_t line\_error\_prev2 = 0;

volatile int8\_t line\_error\_prev3 = 0;

volatile int8\_t line\_error\_prev4 = 0;

volatile double pid\_a = 0.028; // 0.042

volatile double pid\_b = 0;

volatile double pid\_c = 0.0015;

volatile int straight\_away = 0;

volatile uint8\_t line\_start, line\_center, line\_end;

adc16\_config\_t adc16ConfigStruct;

adc16\_channel\_config\_t adc16ChannelConfigStruct;

int count = 0;

int line\_scan\_temp[128];

int line\_scan\_data[128];

int address;

int intersect\_seen = 0;

int prev\_intersect\_seen = 0;

int stop\_line\_seen = 0;

int track\_laps = 0;

int min\_val = 0;

int max\_val = 117;

int num\_black\_sections = 0;

int longest\_black\_section = 0;

int num\_outliers = 3;

int length\_black\_section = 0;

int length\_white\_section = 0;

int in\_black\_section = 0;

int num\_long\_black\_sections = 0;

int long\_section = 10;

int tire\_pos = 0;

int setupLineScanTimers(){

int i;

/\* Configure ftm params \*/

ftmParam1.chnlNumber = kFTM\_Chnl\_7; //Line scan CLK

ftmParam1.level = pwmLevel;

ftmParam1.dutyCyclePercent = 50U;

ftmParam1.firstEdgeDelayPercent = 0U;

ftmParam2.chnlNumber = kFTM\_Chnl\_6; //Line scan SI

ftmParam2.level = pwmLevel;

ftmParam2.dutyCyclePercent = 50U;

ftmParam2.firstEdgeDelayPercent = 0U;

FTM\_GetDefaultConfig(&ftmInfo1);

FTM\_GetDefaultConfig(&ftmInfo2);

/\* Initialize FTM module \*/

FTM\_Init(FTM3, &ftmInfo1);

FTM\_Init(FTM3, &ftmInfo2);

FTM\_SetupPwm(FTM3, &ftmParam1, 1U, kFTM\_EdgeAlignedPwm, 20000U, FTM\_SOURCE\_CLOCK);

FTM\_SetupPwm(FTM3, &ftmParam2, 1U, kFTM\_EdgeAlignedPwm, 20000U, FTM\_SOURCE\_CLOCK);

for(i=0; i<128; ++i){ line\_scan\_data[i] = 100000; }

return 0;

}

static void line\_eval(){

int line\_center\_temp = 0;

int start\_black\_section = 0;

num\_black\_sections = 0;

longest\_black\_section = 0;

num\_long\_black\_sections = 0;

int long\_black\_section\_on\_start = 0;

int long\_black\_section\_on\_end = 0;

for(i = min\_val; i < max\_val; ++i){ // For input (ends cut off)

//printf("%lu", line\_scan\_data[i]); // DEBUG

if(line\_scan\_data[i] == 0){ // If it's a black pixel

++length\_black\_section;

if(!in\_black\_section){ // If it's a white section

if(length\_black\_section > num\_outliers){ // If it's a new black section

in\_black\_section = 1;

++num\_black\_sections;

length\_white\_section = 0;

start\_black\_section = i - num\_outliers;

//printf(" B"); // DEBUG

}

}else{ // Reset length on false alarm from outliers

length\_white\_section = 0;

}

}else{ // If it's a white pixel

++length\_white\_section;

if(in\_black\_section){ // If it's a black section

if(length\_white\_section > num\_outliers){ // If it's a new white section

in\_black\_section = 0;

if(length\_black\_section > longest\_black\_section){ // Update longest black section

longest\_black\_section = length\_black\_section; // TODO: Put in an array of lengths?

//printf(" L (%d)", longest\_black\_section); // DEBUG

line\_center\_temp = ((i - num\_outliers) - start\_black\_section) / 2 + start\_black\_section;

}

if(length\_black\_section > long\_section){

++num\_long\_black\_sections;

if(start\_black\_section < 6){ long\_black\_section\_on\_start = 1; }

}

length\_black\_section = 0;

//printf(" W"); // DEBUG

}

}else{ // Reset length on false alarm from outliers

length\_black\_section = 0;

}

}

if(i == (max\_val-1) && in\_black\_section){ // Ends in a black section

long\_black\_section\_on\_end = 1;

if(length\_black\_section > longest\_black\_section){ // Update longest black section

longest\_black\_section = length\_black\_section; // TODO: Put in an array of lengths?

//printf(" L (%d)", longest\_black\_section); // DEBUG

line\_center\_temp = ((i - num\_outliers) - start\_black\_section) / 2 + start\_black\_section;

}

if(length\_black\_section > long\_section){ ++num\_long\_black\_sections; }

}

//printf(", "); // DEBUG

}

//if(longest\_black\_section > 10 && (line\_center\_temp > 5 && line\_center\_temp < 122) ){ line\_center = line\_center\_temp; }

if(num\_long\_black\_sections == 3 && !intersect\_seen && long\_black\_section\_on\_start && long\_black\_section\_on\_end){

if(!prev\_intersect\_seen){

stop\_line\_seen = 1;

++track\_laps;

}

prev\_intersect\_seen = 1;

}else if(longest\_black\_section > 100 && !stop\_line\_seen){

intersect\_seen = 1;

prev\_intersect\_seen = 0;

}else{

intersect\_seen = 0;

stop\_line\_seen = 0;

prev\_intersect\_seen = 0;

}

/\*printf("\nlongest\_black\_section: %d\nnum\_long\_black\_sections: %d\nintersect\_seen: %d\nstop\_line\_seen: %d\n\n",

longest\_black\_section, num\_long\_black\_sections, intersect\_seen, stop\_line\_seen); // DEBUG

\*/

}

int lineVisible(){ return ((longest\_black\_section > 15) && (line\_error < 10) && (line\_error > -10)) ? 1 : 0; }

void FTM\_LineScan\_Handler(void)

{

num\_cycles++;

if(num\_cycles == 160U)//180

{

FTM\_UpdateChnlEdgeLevelSelect(FTM3, kFTM\_Chnl\_6, 0U);

FTM\_UpdatePwmDutycycle(FTM3, kFTM\_Chnl\_6, kFTM\_EdgeAlignedPwm, 100U);

FTM\_SetSoftwareTrigger(FTM3, true);

FTM\_UpdateChnlEdgeLevelSelect(FTM3, kFTM\_Chnl\_6, kFTM\_LowTrue);

num\_cycles = 0;

ftm\_is\_r\_flag = true;

}

ADC16\_SetChannelConfig(DEMO\_ADC16\_BASE, DEMO\_ADC16\_CHANNEL\_GROUP, &adc16ChannelConfigStruct);

if((num\_outputs < 128U) & (ftm\_is\_r\_flag == true))

{

line\_scan\_temp[num\_outputs] = ADC16\_GetChannelConversionValue(DEMO\_ADC16\_BASE, DEMO\_ADC16\_CHANNEL\_GROUP);

num\_outputs++;

}

else

{

num\_outputs = 0U;

ftm\_is\_r\_flag = false;

for(int i = 5U; i < 123U; i++)

{

if(line\_scan\_temp[i] < 900U)

{

line\_scan\_data[i-5U] = 0U;

}

else

{

line\_scan\_data[i-5U] = 1U;

}

}

line\_eval();

}

if ((FTM\_GetStatusFlags(FTM3) & kFTM\_Chnl7Flag) == kFTM\_Chnl7Flag)

{

/\* Clear interrupt flag.\*/

FTM\_ClearStatusFlags(FTM3, kFTM\_Chnl7Flag);

}

}

int isOnStraightAway(){

return straight\_away;

}

int steerPidOut(){

int i;

line\_start = 0;

line\_end = 0;

line\_center = 0;

while(count != 0){}

for(i = 0; i < 118; ++i){

if((line\_scan\_data[i] == 0) && (line\_start == 0)){

line\_start = i;

}

if((line\_scan\_data[i] == 0) && ((line\_scan\_data[i+1U] == 1U) || (i == 117U))){

line\_end = i;

}

line\_center = (line\_start+line\_end) / 2;

}

line\_error\_prev4 = line\_error\_prev3;

line\_error\_prev3 = line\_error\_prev2;

line\_error\_prev2 = line\_error\_prev;

line\_error\_prev = line\_error;

line\_error = line\_center - 59;

prev\_duty\_cycle = updated\_duty\_cycle;

if(line\_center == 0U)

{

if(prev\_duty\_cycle < 7.5){ updated\_duty\_cycle = 5.0; }

else if(prev\_duty\_cycle > 7.5){ updated\_duty\_cycle = 10.0; }

else{} //do nothing so it still continues the same direction it was

}

else

{

//updated\_duty\_cycle = (7.5 + (59 - line\_center) \* 0.042);

updated\_duty\_cycle = (7.5 - (line\_error \* pid\_a) - ((line\_error + line\_error\_prev + line\_error\_prev2 + line\_error\_prev3 + line\_error\_prev4) \* pid\_b) - ((line\_error - line\_error\_prev) \* pid\_c));

if((line\_error > 4) || (line\_error < -8)){ straight\_away = 0; }

else { straight\_away = 1; }

if(updated\_duty\_cycle > 10.0) { updated\_duty\_cycle = 10.0; }

else if(updated\_duty\_cycle < 5.0){ updated\_duty\_cycle = 5.0; }

}

tire\_pos = (updated\_duty\_cycle - 7.5) / duty\_cycle\_per\_angle;

return tire\_pos;

}

int initializeXADC(){

ADC16\_GetDefaultConfig(&adc16ConfigStruct);

#ifdef BOARD\_ADC\_USE\_ALT\_VREF

adc16ConfigStruct.referenceVoltageSource = kADC16\_ReferenceVoltageSourceValt;

#endif

ADC16\_Init(DEMO\_ADC16\_BASE, &adc16ConfigStruct);

ADC16\_EnableHardwareTrigger(DEMO\_ADC16\_BASE, false); /\* Make sure the software trigger is used. \*/

adc16ChannelConfigStruct.channelNumber = DEMO\_ADC16\_USER\_CHANNEL;

adc16ChannelConfigStruct.enableInterruptOnConversionCompleted = false;

#if defined(FSL\_FEATURE\_ADC16\_HAS\_DIFF\_MODE) && FSL\_FEATURE\_ADC16\_HAS\_DIFF\_MODE

adc16ChannelConfigStruct.enableDifferentialConversion = false;

#endif /\* FSL\_FEATURE\_ADC16\_HAS\_DIFF\_MODE \*/

return 0;

}

#endif /\* SRC\_LINESCAN\_H\_ \*/

#### Timer\_and\_pwm.h

#ifndef SRC\_TIMER\_AND\_PWM\_H\_

#define SRC\_TIMER\_AND\_PWM\_H\_

/\* Get source clock for FTM driver \*/

#define FTM\_SOURCE\_CLOCK CLOCK\_GetFreq(kCLOCK\_BusClk)

#include "fsl\_ftm.h"

/\* The Flextimer instance/channel used for board \*/

//#define BOARD\_FTM\_BASEADDR FTM3

/\* Interrupt to enable and flag to read; depends on the FTM channel used \*/

//#define FTM\_CHANNEL\_INTERRUPT\_ENABLE kFTM\_Chnl0InterruptEnable

//#define FTM\_CHANNEL\_FLAG kFTM\_Chnl0Flag

double duty\_cycle\_per\_angle = 5.0/180.0;

void initializeTimerSystem(){

/\* Enable FTM channel interrupt flag.\*/

FTM\_EnableInterrupts(FTM3, kFTM\_Chnl7InterruptEnable);

FTM\_EnableInterrupts(FTM0, kFTM\_Chnl1InterruptEnable);

/\* Enable at the NVIC \*/

EnableIRQ(FTM3\_IRQn);

EnableIRQ(FTM0\_IRQn);

FTM\_StartTimer(FTM3, kFTM\_SystemClock);

FTM\_StartTimer(FTM0, kFTM\_SystemClock);

}

void FTM\_UpdateDutyCycleDec(FTM\_Type \*base,

ftm\_chnl\_t chnlNumber,

ftm\_pwm\_mode\_t currentPwmMode,

double dutyCyclePercent)

{

uint16\_t cnv, cnvFirstEdge = 0, mod;

mod = base->MOD;

if ((currentPwmMode == kFTM\_EdgeAlignedPwm) || (currentPwmMode == kFTM\_CenterAlignedPwm))

{

cnv = (mod \* dutyCyclePercent) / 100;

/\* For 100% duty cycle \*/

if (cnv >= mod)

{

cnv = mod + 1;

}

base->CONTROLS[chnlNumber].CnV = cnv;

}

else

{

/\* This check is added for combined mode as the channel number should be the pair number \*/

if (chnlNumber >= (FSL\_FEATURE\_FTM\_CHANNEL\_COUNTn(base) / 2))

{

return;

}

cnv = (mod \* dutyCyclePercent) / 100;

cnvFirstEdge = base->CONTROLS[chnlNumber \* 2].CnV;

/\* For 100% duty cycle \*/

if (cnv >= mod)

{

cnv = mod + 1;

}

base->CONTROLS[(chnlNumber \* 2) + 1].CnV = cnvFirstEdge + cnv;

}

}

#endif /\* SRC\_TIMER\_AND\_PWM\_H\_ \*/

#### Buttons\_and\_leds.h

#ifndef SRC\_BUTTONS\_AND\_LEDS\_H\_

#define SRC\_BUTTONS\_AND\_LEDS\_H\_

#include "fsl\_debug\_console.h"

#include "fsl\_port.h"

#include "fsl\_gpio.h"

#include "fsl\_common.h"

#include "board.h"

#include "pin\_mux.h"

#include "clock\_config.h"

#define modeChangeHandler BOARD\_SW3\_IRQ\_HANDLER

#define modeEnableHandler BOARD\_SW2\_IRQ\_HANDLER

/\* Whether the SW button is pressed \*/

int mode\_count = 0;

bool go\_flag = 0;

void modeChangeHandler(void)

{

/\* Clear external interrupt flag. \*/

GPIO\_ClearPinsInterruptFlags(BOARD\_SW3\_GPIO, 1U << BOARD\_SW3\_GPIO\_PIN);

if (mode\_count == 6) {

mode\_count = 0;

} else {

mode\_count = mode\_count + 1;

}

}

void modeEnableHandler(void)

{

/\* Clear external interrupt flag. \*/

GPIO\_ClearPinsInterruptFlags(BOARD\_SW2\_GPIO, 1U << BOARD\_SW2\_GPIO\_PIN);

if (go\_flag == 0){

go\_flag = 1;

} else {

go\_flag = 0;

}

}

int initButtonAndLEDs(void)

{

/\* Define the init structure for the input switch pin \*/

gpio\_pin\_config\_t sw\_config = {

kGPIO\_DigitalInput, 0,

};

/\* Init input switch GPIO. \*/

PORT\_SetPinInterruptConfig(BOARD\_SW3\_PORT, BOARD\_SW3\_GPIO\_PIN, kPORT\_InterruptFallingEdge);

EnableIRQ(BOARD\_SW3\_IRQ);

GPIO\_PinInit(BOARD\_SW3\_GPIO, BOARD\_SW3\_GPIO\_PIN, &sw\_config);

/\* Init input switch GPIO. \*/

PORT\_SetPinInterruptConfig(BOARD\_SW2\_PORT, BOARD\_SW2\_GPIO\_PIN, kPORT\_InterruptFallingEdge);

EnableIRQ(BOARD\_SW2\_IRQ);

GPIO\_PinInit(BOARD\_SW2\_GPIO, BOARD\_SW2\_GPIO\_PIN, &sw\_config);

}

#endif /\* SRC\_BUTTONS\_AND\_LEDS\_H\_ \*/

#### Pixy.h

#ifndef SRC\_PIXY\_H\_

#define SRC\_PIXY\_H\_

#include "fsl\_device\_registers.h"

#include "fsl\_debug\_console.h"

#include "fsl\_dspi.h"

#include "board.h"

#include <math.h>

#include "pin\_mux.h"

#include "clock\_config.h"

#define EXAMPLE\_DSPI\_MASTER\_BASEADDR SPI0

#define EXAMPLE\_DSPI\_MASTER\_CLK\_SRC DSPI0\_CLK\_SRC

#define EXAMPLE\_DSPI\_MASTER\_CLK\_FREQ CLOCK\_GetFreq(DSPI0\_CLK\_SRC)

#define EXAMPLE\_DSPI\_MASTER\_IRQ SPI0\_IRQn

#define EXAMPLE\_DSPI\_MASTER\_PCS kDSPI\_Pcs0

#define EXAMPLE\_DSPI\_MASTER\_IRQHandler SPI0\_IRQHandler

#define TRANSFER\_SIZE 1U /\*! Transfer dataSize \*/

#define TRANSFER\_BAUDRATE 115200U /\*! Transfer baudrate - 500k \*/

uint16\_t masterRxData[TRANSFER\_SIZE] = {0U};

uint8\_t masterTxData[TRANSFER\_SIZE] = {0U};

volatile uint32\_t masterTxCount;

volatile uint32\_t masterRxCount;

volatile uint32\_t masterCommand;

uint32\_t masterFifoSize;

dspi\_master\_handle\_t g\_m\_handle;

volatile bool isTransferCompleted = false;

int16\_t masterRxInfo[2][7] = {0U};

char masterRxString [4];

int poleNum = -1;

int poleAttr = -1;

uint8\_t useless = 0;

int markers\_visible = 0;

int markers\_opposite = 0;

int marker\_sigs[2] = {-1,-1};

int half\_width = 128;

double distance = 0.0;

int height = 0;

dspi\_command\_data\_config\_t commandData;

uint32\_t srcClock\_Hz;

uint32\_t errorCount;

uint32\_t i;

dspi\_master\_config\_t masterConfig;

static void delay(volatile uint32\_t nof) {

while(nof!=0) {

\_\_asm("NOP");

nof--;

}

}

void createArray(int masterRxCount){

if (strcmp(masterRxString,"aa55") == 0){

if (masterRxInfo[1][1] == 0){

markers\_visible = 0;

}

else if (masterRxInfo[1][1] != 0){

markers\_visible = 1;

if(masterRxInfo[0][1] > 0 && masterRxInfo[1][1] > 0){ // check if there are two markers

if(masterRxInfo[0][2] < half\_width && masterRxInfo[1][2] > half\_width){ // marker 1 on left and marker 2 on right

markers\_opposite = 1;

if(masterRxInfo[0][1] == 10){

marker\_sigs[0] = 0;

}

else if(masterRxInfo[0][1] == 11){

marker\_sigs[0] = 1;

}

if(masterRxInfo[1][1] == 10){

marker\_sigs[1] = 0;

}

else if(masterRxInfo[1][1] == 11){

marker\_sigs[1] = 1;

}

}

else if(masterRxInfo[0][2] > half\_width && masterRxInfo[1][2] < half\_width){ // marker 1 on right and marker 2 on left

markers\_opposite = 1;

if(masterRxInfo[0][1] == 10){

marker\_sigs[1] = 0;

}

else if(masterRxInfo[0][1] == 11){

marker\_sigs[1] = 1;

}

if(masterRxInfo[1][1] == 10){

marker\_sigs[0] = 0;

}

else if(masterRxInfo[1][1] == 11){

marker\_sigs[0] = 1;

}

}

else{

markers\_opposite = 0;

}

}

}

if (marker\_sigs[1] == 1 && marker\_sigs[0] == 0){//LEFT [RED]

LED\_RED\_ON();

LED\_GREEN\_OFF();

LED\_BLUE\_OFF();

}

else if (marker\_sigs[1] == 0 && marker\_sigs[0] == 1){//RIGHT [GREEN]

LED\_RED\_OFF();

LED\_GREEN\_ON();

LED\_BLUE\_OFF();

}

else if (marker\_sigs[1] == 1 && marker\_sigs[0] == 1){//U TURN [BLUE]

LED\_RED\_OFF();

LED\_GREEN\_OFF();

LED\_BLUE\_ON();

}

else if (marker\_sigs[1] == 0 && marker\_sigs[0] == 0){//STRAIGHT [WHITE]

LED\_RED\_ON();

LED\_GREEN\_ON();

LED\_BLUE\_ON();

}

poleNum = 0;

//masterRxInfo[1][1] = 0;

}

else if (strcmp(masterRxString,"aa56") == 0 && poleNum != -1){

poleAttr = 0;

}

else if (masterRxData[0] == 0 && poleAttr == 0){

markers\_visible = 0;

}

else if (poleNum >= 0 && poleAttr >= 0){

masterRxInfo[poleNum][poleAttr] = masterRxData[masterRxCount];

poleAttr++;

}

if (poleAttr > 6){

poleAttr = 0;

poleNum++;

}

if (poleNum > 1){

poleNum = -1;

}

// if (masterRxInfo[0][1] == 28 && masterRxData[0] == 0){

// collisionModeSpeed(25);

// }

if (masterRxInfo[0][1] == 28){

height = masterRxInfo[0][5];

distance = 684.16 \* pow(height,-0.987);

collisionModeSpeed(distance);

}

else if (masterRxInfo[1][1] == 28){

height = masterRxInfo[0][5];

distance = 684.16 \* pow(height,-0.987);

collisionModeSpeed(distance);

}

}

void EXAMPLE\_DSPI\_MASTER\_IRQHandler(void)

{

if (masterRxCount < TRANSFER\_SIZE)

{

while (DSPI\_GetStatusFlags(EXAMPLE\_DSPI\_MASTER\_BASEADDR) & kDSPI\_RxFifoDrainRequestFlag)

{

masterRxData[masterRxCount] = DSPI\_ReadData(EXAMPLE\_DSPI\_MASTER\_BASEADDR);

itoa(masterRxData[masterRxCount],masterRxString,16);

createArray(masterRxCount);

++masterRxCount;

DSPI\_ClearStatusFlags(EXAMPLE\_DSPI\_MASTER\_BASEADDR, kDSPI\_RxFifoDrainRequestFlag);

if (masterRxCount == TRANSFER\_SIZE)

{

masterRxCount = 0;

break;

}

}

}

if (masterTxCount < TRANSFER\_SIZE)

{

while ((DSPI\_GetStatusFlags(EXAMPLE\_DSPI\_MASTER\_BASEADDR) & kDSPI\_TxFifoFillRequestFlag) &&

((masterTxCount - masterRxCount) < masterFifoSize))

{

if (masterTxCount < TRANSFER\_SIZE)

{

EXAMPLE\_DSPI\_MASTER\_BASEADDR->PUSHR = masterCommand | masterTxData[masterTxCount];

++masterTxCount;

}

else

{

break;

}

/\* Try to clear the TFFF; if the TX FIFO is full this will clear \*/

DSPI\_ClearStatusFlags(EXAMPLE\_DSPI\_MASTER\_BASEADDR, kDSPI\_TxFifoFillRequestFlag);

}

}

/\* Check if we're done with this transfer.\*/

if ((masterTxCount == TRANSFER\_SIZE) && (masterRxCount == TRANSFER\_SIZE))

{

/\* Complete the transfer and disable the interrupts \*/

DSPI\_DisableInterrupts(EXAMPLE\_DSPI\_MASTER\_BASEADDR,

kDSPI\_RxFifoDrainRequestInterruptEnable | kDSPI\_TxFifoFillRequestInterruptEnable);

}

}

void initPixy(){

useless = 0;

/\* Master config \*/

masterConfig.whichCtar = kDSPI\_Ctar0;

masterConfig.ctarConfig.baudRate = TRANSFER\_BAUDRATE;

masterConfig.ctarConfig.bitsPerFrame = 16;

masterConfig.ctarConfig.cpol = kDSPI\_ClockPolarityActiveHigh;

masterConfig.ctarConfig.cpha = kDSPI\_ClockPhaseFirstEdge;

masterConfig.ctarConfig.direction = kDSPI\_MsbFirst;

masterConfig.ctarConfig.pcsToSckDelayInNanoSec = 1000000000U / TRANSFER\_BAUDRATE;

masterConfig.ctarConfig.lastSckToPcsDelayInNanoSec = 1000000000U / TRANSFER\_BAUDRATE;

masterConfig.ctarConfig.betweenTransferDelayInNanoSec = 5000000000U / TRANSFER\_BAUDRATE;

masterConfig.whichPcs = EXAMPLE\_DSPI\_MASTER\_PCS;

masterConfig.pcsActiveHighOrLow = kDSPI\_PcsActiveLow;

masterConfig.enableContinuousSCK = false;

masterConfig.enableRxFifoOverWrite = false;

masterConfig.enableModifiedTimingFormat = false;

masterConfig.samplePoint = kDSPI\_SckToSin0Clock;

srcClock\_Hz = EXAMPLE\_DSPI\_MASTER\_CLK\_FREQ;

DSPI\_MasterInit(EXAMPLE\_DSPI\_MASTER\_BASEADDR, &masterConfig, srcClock\_Hz);

isTransferCompleted = false;

/\* Enable the NVIC for DSPI peripheral. \*/

EnableIRQ(EXAMPLE\_DSPI\_MASTER\_IRQ);

/\* Start master transfer\*/

commandData.isPcsContinuous = false;

commandData.whichCtar = kDSPI\_Ctar0;

commandData.whichPcs = EXAMPLE\_DSPI\_MASTER\_PCS;

commandData.isEndOfQueue = false;

commandData.clearTransferCount = false;

masterCommand = DSPI\_MasterGetFormattedCommand(&commandData);

masterFifoSize = FSL\_FEATURE\_DSPI\_FIFO\_SIZEn(EXAMPLE\_DSPI\_MASTER\_BASEADDR);

masterTxCount = 0;

masterRxCount = 0;

DSPI\_StopTransfer(EXAMPLE\_DSPI\_MASTER\_BASEADDR);

DSPI\_FlushFifo(EXAMPLE\_DSPI\_MASTER\_BASEADDR, true, true);

DSPI\_ClearStatusFlags(EXAMPLE\_DSPI\_MASTER\_BASEADDR, kDSPI\_AllStatusFlag);

DSPI\_StartTransfer(EXAMPLE\_DSPI\_MASTER\_BASEADDR);

/\*Fill up the master Tx data\*/

while (DSPI\_GetStatusFlags(EXAMPLE\_DSPI\_MASTER\_BASEADDR) & kDSPI\_TxFifoFillRequestFlag)

{

if (masterTxCount < TRANSFER\_SIZE)

{

DSPI\_MasterWriteData(EXAMPLE\_DSPI\_MASTER\_BASEADDR, &commandData, masterTxData[masterTxCount]);

++masterTxCount;

}

else

{

break;

}

/\* Try to clear the TFFF; if the TX FIFO is full this will clear \*/

DSPI\_ClearStatusFlags(EXAMPLE\_DSPI\_MASTER\_BASEADDR, kDSPI\_TxFifoFillRequestFlag);

}

/\*Enable master RX interrupt\*/

DSPI\_EnableInterrupts(EXAMPLE\_DSPI\_MASTER\_BASEADDR, kDSPI\_RxFifoDrainRequestInterruptEnable);

}

void pixyRead(){

DSPI\_MasterWriteData(EXAMPLE\_DSPI\_MASTER\_BASEADDR, &commandData, masterTxData[masterTxCount]);

}

#endif /\* SRC\_PIXY\_H\_ \*/

# References

[1] KVM, *Detailed Design Document*, 2017. [Online]. Clarkson University. [Accessed: 11- Nov- 2017]

[2] "PID controller", *En.wikipedia.org*, 2017. [Online]. Available: https://en.wikipedia.org/wiki/PID\_controller. [Accessed: 11- Nov- 2017].

[3] "Discrete PID Controller", *Portal.ku.edu.tr*, 2017. [Online]. Available: http://portal.ku.edu.tr/~cbasdogan/Courses/Robotics/projects/Discrete\_PID.pdf. [Accessed: 11- Nov- 2017].