#include <due\_can.h>

#include <genieArduino.h>

#include <Arduino.h>

#include <pins\_arduino.h>

#include <stdint.h>

#include <u8glib.h>

long Boost, EngineSpeed, CoolantTemp, IntakeTemp, VehicleSpeed, ButtonHeld, IgnitionAngle;

//CAN Frames Sent

CAN\_FRAME BP;

CAN\_FRAME RPM;

CAN\_FRAME COL;

CAN\_FRAME IAT;

CAN\_FRAME IA;

CAN\_FRAME VHS;

int x, Brightness;

unsigned char len = 0, flagRecv = 0, Page = 0, Index = 0;

bool NightMode, Ignition, GaugeSweep;

static int gaugeAddVal = 1;

static int gaugeVal = 0;

static int gaugeCurrentValue = 0;

static int defaultBrightness = 17;

Genie genie;

#define RESETLINE 45

void setup()

{

Serial3.begin(256000); //Start serial comms for the LCD display

SerialUSB.begin(115200);

pinMode(DS3, OUTPUT);

digitalWrite(DS3, LOW);

delay(1000);

digitalWrite(DS3, HIGH);

genie.Begin(Serial3); // Use Serial 1 for talking to the Genie Library, and to the 4D Systems display

pinMode(RESETLINE, OUTPUT); // Set D4 on Arduino to Output (4D Arduino Adaptor V2 - Display Reset)

digitalWrite(RESETLINE, 0); // Reset the Display via D4

delay(100);

digitalWrite(RESETLINE, 1); // unReset the Display via D4

Can0.begin(CAN\_BPS\_125K);

Can1.begin(CAN\_BPS\_250K);

CanFrames(); //Declare our various static frames

delay (5000);

// Masks and filters setup for the can interface. We use these to keep resource usage low by only looking for traffic we care about.

Can1.setRXFilter(0, 0x00400021, 0xFFFFFFFF, 1);

Can1.setRXFilter(1, 0x19E00006, 0xFFFFFFFF, 1);

Can1.setRXFilter(2, 0x0100082C, 0xFFFFFFFF, 1);

Can1.setRXFilter(3, 0x19000026, 0xFFFFFFFF, 1);

Can0.watchFor(0x09C050B8);

Can0.setGeneralCallback(sendModifiedFrame);

genie.WriteContrast(0);

}

void loop() //The main loop sends all the various CAN messages to the ECU so we can get data back. It also calls the MessageRecieve and UpdateDisplay functions periodically. This method of 'multitasking' is painful and needs to be rewritten.

{

if (Ignition == 0) {

PowerSaveLoop();

}

if (Ignition) {

UpdateDisplay();

}

//We send different messages at different intervals. Slow updating variables don't need to be polled nearly as fast as fast updating ones.

x++;

if (x > 100000 && (Page == 0)) {

Can1.sendFrame(BP);

x = 0;

}

if (x > 50000 && (Page == 2)) {

Can1.sendFrame(COL);

x = 0;

}

if (x > 10000 && (Page == 1)) {

Can1.sendFrame(IAT);

x = 0;

}

if (x > 3000 && (Page == 3)) {

Can1.sendFrame(IA);

x = 0;

}

MessageRecieveLoop();

}

void UpdateDisplay() { //This function takes the data retrieved in the MessageRecieveLoop and writes it to the LCD.

//Boost Display

if (Page == 0) {

gaugeVal = (Boost - 101.352) \* 1.45;

if (gaugeVal < 0) {

gaugeVal = 0;

}

if (gaugeCurrentValue != gaugeVal) {

if (gaugeCurrentValue > gaugeVal) gaugeAddVal = gaugeVal - gaugeCurrentValue;

if (gaugeCurrentValue < gaugeVal) gaugeAddVal = gaugeVal - gaugeCurrentValue;

gaugeCurrentValue += gaugeAddVal;

genie.WriteObject(GENIE\_OBJ\_IANGULAR\_METER, 0, gaugeCurrentValue);

genie.WriteObject(GENIE\_OBJ\_ILED\_DIGITS, 0, gaugeCurrentValue);

}

}

else if (Page == 1) {

gaugeVal = IntakeTemp;

if (gaugeVal < 0) {

gaugeVal = 0;

}

if (gaugeCurrentValue != gaugeVal) {

if (gaugeCurrentValue > gaugeVal) gaugeAddVal = -1;

if (gaugeCurrentValue < gaugeVal) gaugeAddVal = 1;

gaugeCurrentValue += gaugeAddVal;

genie.WriteObject(GENIE\_OBJ\_IANGULAR\_METER, 1, gaugeCurrentValue);

genie.WriteObject(GENIE\_OBJ\_ILED\_DIGITS, 1, gaugeCurrentValue);

}

}

else if (Page == 2) {

gaugeVal = CoolantTemp;

if (gaugeVal < 0) {

gaugeVal = 0;

}

if (gaugeCurrentValue != gaugeVal) {

if (gaugeCurrentValue > gaugeVal) gaugeAddVal = -1;

if (gaugeCurrentValue < gaugeVal) gaugeAddVal = 1;

gaugeCurrentValue += gaugeAddVal;

genie.WriteObject(GENIE\_OBJ\_IANGULAR\_METER, 2, gaugeCurrentValue);

genie.WriteObject(GENIE\_OBJ\_ILED\_DIGITS, 2, gaugeCurrentValue);

}

}

else if (Page == 3) {

gaugeVal = (IgnitionAngle);

if (gaugeCurrentValue != gaugeVal) {

if (gaugeCurrentValue > gaugeVal) gaugeAddVal = gaugeVal - gaugeCurrentValue;

if (gaugeCurrentValue < gaugeVal) gaugeAddVal = gaugeVal - gaugeCurrentValue;

gaugeCurrentValue += gaugeAddVal;

genie.WriteObject(GENIE\_OBJ\_IANGULAR\_METER, 3, 50 - (gaugeCurrentValue \* .55));

genie.WriteObject(GENIE\_OBJ\_ILED\_DIGITS, 3, gaugeCurrentValue);

}

}

}

void UpdateBrightness() { //updates the brightness of the OLED when we adjust the dashboard lighting

if (Ignition) {

if (!NightMode) {

genie.WriteContrast(Brightness \* 0.8);

}

else {

if (Brightness > 1) {

genie.WriteContrast(defaultBrightness);

}

else {

genie.WriteContrast(0);

}

}

}

}

void PowerSaveLoop() { //if the ignition message goes low. loop here indefinitely until ignition has come back on. OLED is off and filters are activated to ignore any traffic except the ignition status broadcast.

Can1.setRXFilter(0, 0x00000000, 0xFFFFFFFF, 1);

Can1.setRXFilter(1, 0x19E00006, 0xFFFFFFFF, 1);

Can1.setRXFilter(2, 0x00000000, 0xFFFFFFFF, 1);

Can1.setRXFilter(3, 0x00000000, 0xFFFFFFFF, 1);

while (1 == 1) {

delay(200);

if (Can1.rx\_avail())

{

CAN\_FRAME INCOMING;

Can1.get\_rx\_buff(INCOMING);

if (INCOMING.id == 0x19E00006) {

if ((INCOMING.data.bytes[6] & B01000000) != Ignition) {

Ignition = (INCOMING.data.bytes[6] & B01000000);

Can1.setRXFilter(0, 0x00400021, 0xFFFFFFFF, 1);

Can1.setRXFilter(1, 0x19E00006, 0xFFFFFFFF, 1);

Can1.setRXFilter(2, 0x0100082C, 0xFFFFFFFF, 1);

Can1.setRXFilter(3, 0x19000026, 0xFFFFFFFF, 1);

UpdateIgnition();

break;

}

}

}

}

}

void MessageRecieveLoop() { //check if we have any new messages. If we do, update variables.

while (Can1.rx\_avail())

{

CAN\_FRAME INCOMING;

Can1.get\_rx\_buff(INCOMING);

//SerialUSB.println(INCOMING.id,HEX);

if (INCOMING.id == 0x00400021) {

if (INCOMING.data.bytes[4] == 0x9d) {

Boost = INCOMING.data.bytes[5];

}

if (INCOMING.data.bytes[4] == 0xd8) {

CoolantTemp = INCOMING.data.bytes[5];

CoolantTemp = CoolantTemp \* 0.75 - 48;

}

if (INCOMING.data.bytes[4] == 0xCE) {

IntakeTemp = INCOMING.data.bytes[5];

IntakeTemp = IntakeTemp \* 0.75 - 48;

IntakeTemp = IntakeTemp \* 1.8 + 32;

}

if (INCOMING.data.bytes[4] == 0x36) {

IgnitionAngle = INCOMING.data.bytes[5];

IgnitionAngle = IgnitionAngle \* 191.25 / 255;

}

}

if (INCOMING.id == 0x19E00006) {

if ((INCOMING.data.bytes[6] & B01000000) == !Ignition) {

Ignition = (INCOMING.data.bytes[6] & B01000000);

UpdateIgnition();

}

}

if (INCOMING.id == 0x19000026) {

if (INCOMING.data.bytes[7] == 32) {

if (ButtonHeld < 1) {

ButtonHeld = millis();

}

if ((millis() - ButtonHeld) > 1000 ) {

Page++;

if (Page > 3) {

Page = 0;

}

genie.WriteObject(GENIE\_OBJ\_FORM, Page, 0);

ButtonHeld = millis();

}

}

else {

ButtonHeld = 0;

}

}

if (INCOMING.id == 0x0100082C) {

if ((INCOMING.data.bytes[1] & B10000000) != NightMode) {

NightMode = INCOMING.data.bytes[1] & B10000000;

UpdateBrightness();

}

if ((INCOMING.data.bytes[0] & B00001111) != Brightness) {

Brightness = (INCOMING.data.bytes[0] & B00001111);

UpdateBrightness();

}

}

}

}

void UpdateIgnition() {

//This is where we go if the ignition status changes. If it goes high to low (car turns off) then we fade the display out, blank it and put it to sleep for the power save loop. If it goes low to high (car turns on) we wake the display and write a welcome message to the display and proceed to the main loop.

if (Ignition) {

genie.WriteContrast(defaultBrightness);

Page = 0;

genie.WriteObject(GENIE\_OBJ\_FORM, Page, 0);

GaugeSweep = 1;

for (int i = 0; i < 150; i++) {

delay(1);

gaugeVal = i;

if (gaugeCurrentValue != gaugeVal) {

if (gaugeCurrentValue > gaugeVal) gaugeAddVal = -1;

if (gaugeCurrentValue < gaugeVal) gaugeAddVal = 1;

gaugeCurrentValue += gaugeAddVal;

genie.WriteObject(GENIE\_OBJ\_IANGULAR\_METER, 0, gaugeCurrentValue);

genie.WriteObject(GENIE\_OBJ\_ILED\_DIGITS, 0, gaugeCurrentValue);

}

}

GaugeSweep = 0;

}

else {

for (int i = Brightness; i > -1; i--) {

genie.WriteContrast(i);

delay(100);

}

}

}

void CanFrames()

{

BP.extended = 1;

BP.id = 0x000FFFFE;

BP.length = 8;

BP.data.bytes[0] = 0xCD;

BP.data.bytes[1] = 0x7a;

BP.data.bytes[2] = 0xa6;

BP.data.bytes[3] = 0x12;

BP.data.bytes[4] = 0x9d;

BP.data.bytes[5] = 0x04;

BP.data.bytes[6] = 0x00;

BP.data.bytes[7] = 0x00;

RPM.extended = 1;

RPM.id = 0x000FFFFE;

RPM.length = 8;

RPM.data.bytes[0] = 0xCD;

RPM.data.bytes[1] = 0x7a;

RPM.data.bytes[2] = 0xa6;

RPM.data.bytes[3] = 0x10;

RPM.data.bytes[4] = 0x1d;

RPM.data.bytes[5] = 0x01;

RPM.data.bytes[6] = 0x00;

RPM.data.bytes[7] = 0x00;

COL.id = 0x000FFFFE;

COL.extended = 1;

COL.length = 8;

COL.data.bytes[0] = 0xCD;

COL.data.bytes[1] = 0x7a;

COL.data.bytes[2] = 0xa6;

COL.data.bytes[3] = 0x10;

COL.data.bytes[4] = 0xd8;

COL.data.bytes[5] = 0x01;

COL.data.bytes[6] = 0x00;

COL.data.bytes[7] = 0x00;

IAT.extended = 1;

IAT.id = 0x000FFFFE;

IAT.length = 8;

IAT.data.bytes[0] = 0xCD;

IAT.data.bytes[1] = 0x7a;

IAT.data.bytes[2] = 0xa6;

IAT.data.bytes[3] = 0x10;

IAT.data.bytes[4] = 0xce;

IAT.data.bytes[5] = 0x01;

IAT.data.bytes[6] = 0x00;

IAT.data.bytes[7] = 0x00;

IA.extended = 1;

IA.id = 0x000FFFFE;

IA.length = 8;

IA.data.bytes[0] = 0xCD;

IA.data.bytes[1] = 0x7a;

IA.data.bytes[2] = 0xa6;

IA.data.bytes[3] = 0x10;

IA.data.bytes[4] = 0x36;

IA.data.bytes[5] = 0x01;

IA.data.bytes[6] = 0x00;

IA.data.bytes[7] = 0x00;

VHS.extended = 1;

VHS.id = 0x000FFFFE;

VHS.length = 8;

VHS.data.bytes[0] = 0xCD;

VHS.data.bytes[1] = 0x7a;

VHS.data.bytes[2] = 0xa6;

VHS.data.bytes[3] = 0x11;

VHS.data.bytes[4] = 0x40;

VHS.data.bytes[5] = 0x01;

VHS.data.bytes[6] = 0x00;

VHS.data.bytes[7] = 0x00;

}

void sendModifiedFrame(CAN\_FRAME \*incoming) {

if (GaugeSweep) {

CAN\_FRAME modified = \*incoming;

//Can1.read((\*incoming));

//modified.data.bytes[3]=gaugeVal\*1.7;

modified.data.bytes[3] = 0xff;

modified.data.bytes[4] = 0xff;

Can0.sendFrame(modified);

}

}