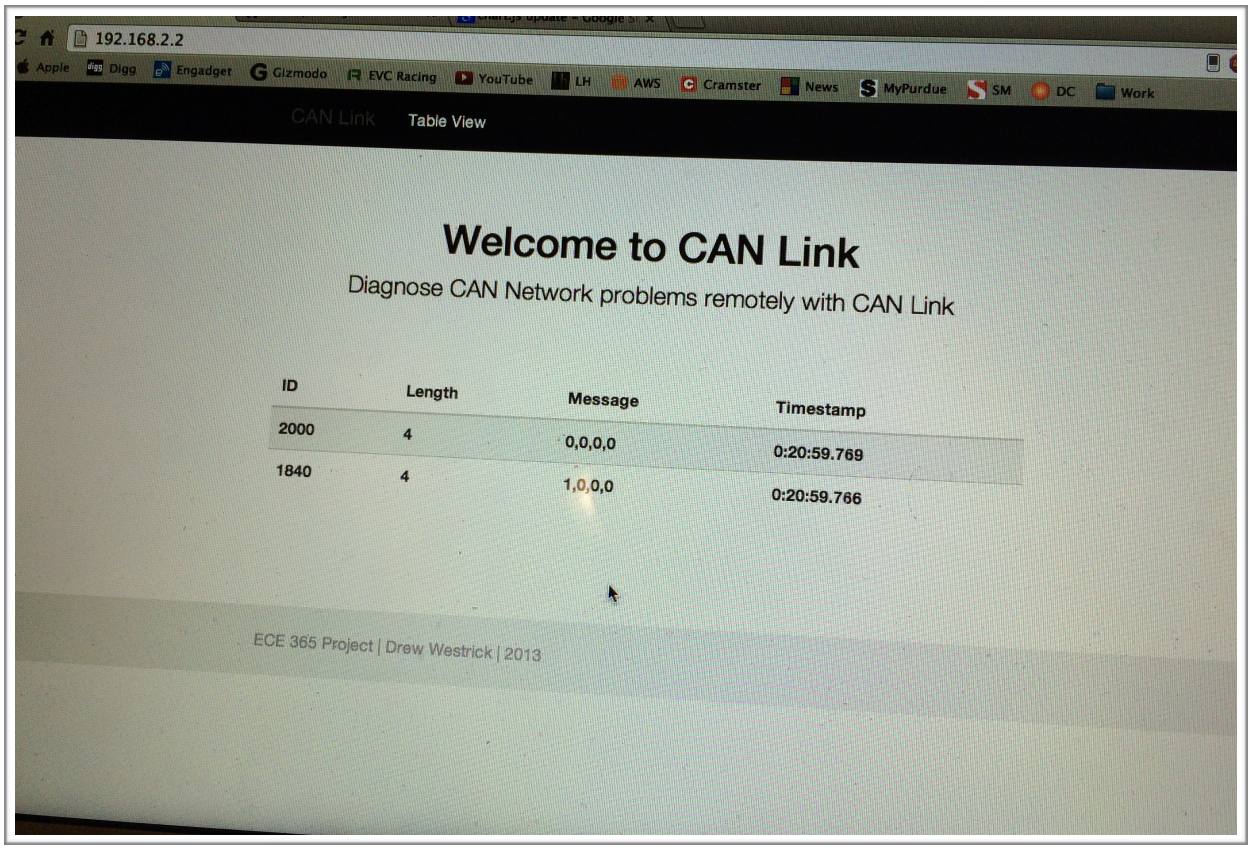
CAN Link

Diagnose CAN network problems remotely with CAN Link



ECE 365 Project | Andrew Westrick

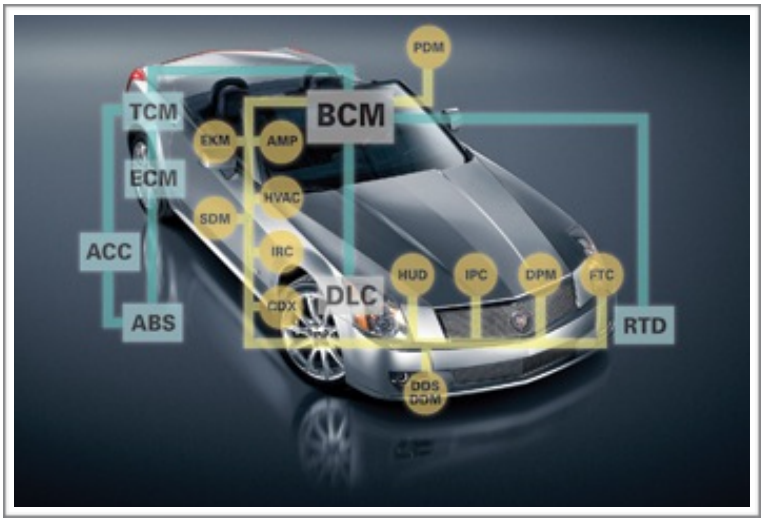
Fall 2014

CAN Link

Diagnose CAN network problems remotely with CAN Link

Introduction

In the last few months I have found myself working on a lot of project that require the use of a CAN Network. Diagnosing CAN problems and viewing information on the network is often an important part of my job. This is why I decided to create a network capable utility that can interface with any CAN 2.0b network.



CAN Network Freescale Processor

At the heart of the CAN Link project is an ARM M4 Freescale processor. This processor acts at the intermediate layer between the Raspberry Pi and the physical CAN network. The development kit that I was working with has an onboard CAN transceiver and decoder. This processor has on onboard FPU so it can perform advanced floating point math very quickly if needed. There are configuration variables in the CAN processor code that allow the CAN data to be scaled and offset as needed. Once the information is captured via the Freescale ARM processor it is decoded, formatted and then passed out via an onboard UART

running at 38,400 baud. This is how the information is transferred from the Freescale processor to the Raspberry Pi application processor.

Application Raspberry Pi Processor

Once the data has been captured and formatted is comes into the Raspberry Pi via its onboard UART transceiver. The main operating system that runs on the raspberry pi is a Debian variant called Raspbian. Running on the operating system is a server side program called NodeJs. NodeJs is a server side javascript engine. Since Node runs on the server side it is able to access and interface with the onboard UART. The nice thing about Node is that it is decided to high throughput asynchronous communications. This makes it a perfect fit as this system will need to handle a larger amount of data that may come at any time over the CAN network. During testing of the system I was able to run multiple CAN messages at 1000 times a second. The Raspberry Pi running Node was more than capable to handle this data load.

NodeJs Web Application

Once the data is read in via the NodeJs server side script it is sent over to the client via a web socket connection. Again this connection is asynchronous and non-blocking. This allows multiple clients to be connected at once and receive data in near real time. One the client web application side the messages that we sent via the web socket from the server side are received. jQuery is then used to parse the data and create the table elements needed to display the messages. If a message already exists it is updated, otherwise a new row in the table is created.

Conclusion

The overall benefit to this system is the fact that the CAN network is now internet connected. With a few modifications to the code this data can be send up to a log server that can save the data for later analysis and use. This system also allows program managers and engineers to monitor test data remotely. Overall I think this CAN Link program will come in use on future projects. It will also be made available via github.com so that other people can benefit from its use.

/**

NodeJs Application Code

```
* Module dependencies.
 */

var express = require('express');
var routes = require('./routes');
var user = require('./routes/user');
var http = require('http');
var path = require('path');
var serialport = require("serialport");
var SerialPort = serialport.SerialPort;

var serialPort = new SerialPort("/dev/ttyAMA0", {
  baudrate: 38400
});

var app     = express();
var server  = require('http').createServer(app);
var io      = require('socket.io').listen(server);

io.set('log level', 1);

function Message(data) {
    this.Id = data[0]+(data[1]<<8);
    this.Len = data[2]
    this.Data = data.slice(3,3+this.Len);
}

serialPort.on("open", function () {
    console.log('open');
    serialPort.on('data', function(data) {
```

```
        var Pkt = new Message(data);
        io.sockets.emit('Pkt', JSON.stringify(Pkt));
    });
});

// all environments
app.set('port', process.env.PORT || 80);
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');
app.use(express.favicon());
app.use(express.logger('dev'));
app.use(express.json());
app.use(express.urlencoded());
app.use(express.methodOverride());
app.use(app.router);
app.use(require('stylus').middleware(path.join(__dirname, 'public')));
app.use(express.static(path.join(__dirname, 'public')));

// development only
if ('development' == app.get('env')) {
  app.use(express.errorHandler());
}

app.get('/', routes.index);
app.get('/users', user.list);

server.listen(app.get('port'));
```

CAN Message Decoding Code

```
#include "CAN.h"
#include "CAN_MSG_1840_0x730.h"
#include "Events.h"

CAN_MSG_1840_0x730_Type CAN_MSG_1840_0x730 = {
    .Factor.Signal_1 = 1,
    .Offset.Signal_1 = 0,
    .Factor.Signal_2 = 1,
    .Offset.Signal_2 = 0,
    .Factor.Signal_3 = 1,
    .Offset.Signal_3 = 0,
    .Factor.Signal_4 = 1,
    .Offset.Signal_4 = 0
};
```

```
void CAN_MSG_1840_0x730_Handler(CAN_Queue *QueueIn, LDD_CAN_TMBIndex
BufferIdx) {
LDD_CAN_TFrame CAN_MSG_1840_0x730_Frame = {
.Data = (uint8_t*)&CAN_MSG_1840_0x730.Msg,
};
/* Receive message */
CAN0_ReadFrame(QueueIn->Device, BufferIdx, &CAN_MSG_1840_0x730_Frame);

CAN_MSG_1840_0x730_Pkt.Message.Signal_1 =
((CAN_MSG_1840_0x730.Msg.Signal_1*CAN_MSG_1840_0x730.Factor.Signal_1)+
CAN_MSG_1840_0x730.Offset.Signal_1);

CAN_MSG_1840_0x730_Pkt.Message.Signal_2 =
((CAN_MSG_1840_0x730.Msg.Signal_2*CAN_MSG_1840_0x730.Factor.Signal_2)+
CAN_MSG_1840_0x730.Offset.Signal_2);
CAN_MSG_1840_0x730_Pkt.Message.Signal_3 =
((CAN_MSG_1840_0x730.Msg.Signal_3*CAN_MSG_1840_0x730.Factor.Signal_3)+
CAN_MSG_1840_0x730.Offset.Signal_3);

CAN_MSG_1840_0x730_Pkt.Message.Signal_4 =
((CAN_MSG_1840_0x730.Msg.Signal_4*CAN_MSG_1840_0x730.Factor.Signal_4)+
CAN_MSG_1840_0x730.Offset.Signal_4);

CAN_MSG_1840_0x730_Pkt.ID = CAN_MSG_1840_0x730_Frame.MessageID;
CAN_MSG_1840_0x730_Pkt.Length = CAN_MSG_1840_0x730_Frame.Length;
LDD_TError Error;
Error = UART0_SendBlock(UART0_DeviceData, &CAN_MSG_1840_0x730_Pkt ,
sizeof(CAN_MSG_1840_0x730_Pkt));
}
```

CAN Transmitter Queue Code

```
#include <string.h>
#include "CAN.h"
#include "Events.h"

void CAN_QueueInit(CAN_Queue *Queue) {
    Queue->Front = 0;
    Queue->Back = 0;
    Queue->Size = 0;
    Queue->LostPacket = 0;
}

uint8 CAN_QueueAdd(CAN_Queue* Queue, LDD_CAN_TFrame Frame) {
    if ((Queue->Size) < CAN_QUEUE_SIZE) {
        memcpy(&(Queue->Queue[Queue->Back]),&Frame,sizeof(Queue-
>Queue[Queue->Back]));    /* Copy contents of message to the back of
the CAN queue */
```

```
        Queue->Back++;
                            /* Increment back pointer of the queue */
        Queue->Size++;
                            /* and increase its size by 1 */
        return 0;
    }
    else {
        return 1;
    }
}

uint8 CAN_QueueRemove(CAN_Queue* Queue) {
    if ((Queue->Size) > 0) {
        Queue->Back--;
        Queue->Size--;
        return 0;
    }
    else {
        return 1;
    }
}

uint8 CAN_QueueTransmit(CAN_Queue* Queue) {
    if ((Queue->Size) > 0) {
        int BusyCnt=0;
        while(Queue->SendFrame(Queue->Device, Queue->Buffer, &(Queue-
>Queue[Queue->Front]))==ERR_BUSY){     /* Attempt to retransmit while
bus is busy */
            BusyCnt++;
            if (BusyCnt>CAN_TX_FAULT_TIME) {
                                    /* If bus busy count > TX fault time
then drop message */
                Queue->LostPacket++;
                                    /* and increment lost packet count
*/
                break;
            }
        }
        Queue->Front++;
                                        /* Remove message from queue */
        Queue->Size--;
                                        /* Decrease queue size */
        return Queue->Size;
    }
    else {
        CAN_QueueEmpty(Queue);
                                        /* Empty the queue */
```

```
        Queue->Stats=Queue->GetStats(Queue->Device);
                                    /* Get CAN0 communication statistics
*/
        return 0;
    }
}

void CAN_QueueEmpty(CAN_Queue *Queue) {
    Queue->Front = 0;
    Queue->Back = 0;
    Queue->Size = 0;
}
```

Web Application Code

```
!!! 5
html(lang='en')
head
    meta(charset='utf-8')
    meta(http-equiv='X-UA-Compatible', content='IE=edge')
    meta(name='viewport', content='width=device-width, initial-
scale=1.0')
    meta(name='description', content='')
    meta(name='author', content='')
    link(rel='shortcut icon', href='../../docs-assets/ico/
favicon.png')
    title CAN Link | ECE 365 Project
    link(href='/stylesheets/bootstrap.css', rel='stylesheet')
    link(href='/stylesheets/default.css', rel='stylesheet')
    script(src='/socket.io/socket.io.js')
    script(src='/javascripts/Chart.min.js')
  body
    script
        var messages = new Array();
        var socket = io.connect(location.host);
        socket.on('Pkt', function (data) {
            var d = new Date();
            Pkt = $.parseJSON(data);
            if ($.inArray(Pkt.Id, messages)==-1) {
                messages.push(Pkt.Id);
                console.log(Pkt);
                $("#tablecan tbody").append('<tr id="T'+Pkt.Id+'"></
tr>');
                $("#T"+Pkt.Id).append("<th>"+Pkt.Id+"</th>");
                $("#T"+Pkt.Id).append("<th>"+Pkt.Len+"</th>");
                $("#T"+Pkt.Id).append("<th>"+Pkt.Data+"</th>");
                $("#T"+Pkt.Id).append("<th>"+d.getHours()
+":"+d.getMinutes()+":"+(d.getSeconds()+(d.getMilliseconds()/
1000)).toFixed(3)+"</th>");
```

```
            }
            else {
                var d = new Date();
                $("#T"+Pkt.Id).empty()
                $("#T"+Pkt.Id).append("<th>"+Pkt.Id+"</th>");
                $("#T"+Pkt.Id).append("<th>"+Pkt.Len+"</th>");
                $("#T"+Pkt.Id).append("<th>"+Pkt.Data+"</th>");
                $("#T"+Pkt.Id).append("<th>"+d.getHours()
+":"+d.getMinutes()+":"+(d.getSeconds()+(d.getMilliseconds()/
1000)).toFixed(3)+"</th>");
            }
        });
    .navbar.navbar-inverse.navbar-fixed-top(role='navigation')
      .container
        .navbar-header
          button.navbar-toggle(type='button', data-toggle='collapse',
data-target='.navbar-collapse')
            span.sr-only Toggle navigation
            span.icon-bar
            span.icon-bar
            span.icon-bar
          a.navbar-brand(href='#') CAN Link
        .collapse.navbar-collapse
          ul.nav.navbar-nav
            li.active
              a(href='/') Table View
    .container
      .starter-template
        h1 Welcome to CAN Link
        p.lead
          | Diagnose CAN network problems remotely with CAN Link
    .container
        table#tablecan.table.table-striped
            thead
                tr
                    th ID
                    th Length
                    th Message
                    th Timestamp
            tbody
    script(src='/javascripts/jquery-1.10.2.min.js')
    script(src='/javascripts/bootstrap.min.js')
    #footer
        .container
            p.text-muted.credit
                | ECE 365 Project | Drew Westrick | 2013
```