University College of Southeast Norway

Bachelor Project

Campus Vestfold

University College of Southeast Norway

Bachelor Project

Campus Vestfold

Machine Learning Based Intrusion Detection in Controller Area Networks

Innbrudds-deteksjon på CAN-buss basert på maskinlæring

Project number:	DA-2016-06
Project group:	Roar Elias Georgsen
Submission date:	19.05.2016
Restriction:	By Agreement

Summary:
This project examines the feasibility of machine learning based fingerprinting of CAN transceivers for the purpose of uniquely identifying signal sources during intrusion detection.

A working multi-node CAN bus development environment was constructed, and an OpenCL Deep Learning Python Wrapper was ported to the platform.

Multiple Machine Learning Algorithms were compared Systematically, and two models fully implemented on a SoC ARM/FPGA device, with computationally intensive tasks running as Software Defined Hardware using an OpenCL FPGA interface.

The implementation achieves a higher hit rate than earlier work based on least-mean squares and convolution Digital Signals Processing (DSP). Performance on learning tasks is comparable to high end CPU devices, indicating that FPGA is a cost effective solution for utilizing machine learning in embedded systems.

While statistical methods are not sufficient on their own, these results demonstrate that machine learning based methods are now viable in embedded devices, presenting a useful way to circumvent security issues faced by Controller Area Networks on the protocol level.

University College of Southeast Norway

Bachelor Project

Campus Vestfold

Skjemaet skal leveres sammen med besvarelsen.

Obligatorisk erklæring

Jeg erklærer herved at min:

Eksamensbesvarelse i emnekode:	FE-BAC3000	**Fakultet:**	

1. er utført av undertegnede. Dersom det er et gruppearbeide, blir alle involverte holdt ansvarlig og alle skal undertegne blanketten.
2. ikke har vært brukt til samme/en annen eksamen ved HSN eller et annet institutt/ universitet/høgskole innenlands eller utenlands.
3. ikke er kopi eller avskrift av andres arbeid, uten at dette er korrekt oppgitt.
4. ikke refererer til eget tidligere arbeid uten at dette er oppgitt.
5. har oppgitt alle referanser/kilder som er brukt i litteraturlisten.

Jeg/vi er kjent med at brudd på disse bestemmelsene er å betrakte som fusk og behandles i hht. §18 i Forskrift om eksamen og studierett ved HSN og U-loven Kap. 4 § 4-7.

Dato:	19.05.2016	Sted:	Borre

Underskrift[1]:		Kand.nr.:	6128

University College of Southeast Norway

Bachelor Project

Campus Vestfold

Machine Learning Based Intrusion Detection in Controller Area Networks

Innbrudds-deteksjon på CAN-buss basert på maskinlæring

Forfatternes navn: Roar Elias Georgsen

Veiledernes navn: Thomas Nordli

Kurs/avdeling: ____

Dato: _19.05.2016______________________

Rett til innsyn, kopiering og publisering av bacheloroppgave

Høgskolen ønsker å gjøre gode bacheloroppgaver tilgjengelig ved å publisere dem i papirutgave og legge dem på internett. Høgskolen trenger studentenes tillatelse til dette.

Hovedprosjektet vil fortsatt være forfatterens åndsverk med de rettigheter det gir.

Høgskolens bruk vil ikke omfatte kommersiell bruk av studenters hovedprosjekt.

Tillater du/dere at din/deres hovedprosjekt blir publisert både i papir og nettutgave?

_X_ ja ___ nei

Signatur av alle forfattere:

____

____

____

Preface

This project was inspired by the HSN (formerly HBV) AHMOS [1] [2] Project, a honeypot based intrusion detection system for use with SCADA (Supervisory control and data acquisition) systems in ship engine rooms. However, as this project does not directly focus on a specific engine environment, it should not be considered a direct continuation of AHMOS.

As with the two AHMOS projects, Thomas Nordli has acted as main academic advisor. Rune Langøy and Christian Hovden have provided additional assistance and equipment.

In contrast to AHMOS, the current project focuses on the CAN specification independent of application, with an emphasis on the physical layer in particular. The project began without any prior knowledge of CAN or vehicle networks, but built on former experience in System-on-Chip design.

The CAN (Controller Area Network) bus is increasingly becoming exposed to sophisticated security threats. In the absence of source authentication, the CAN bus is an easy target for even basic attacks. Most approaches to CAN security have centered around cryptography at the protocol layer, but the increased computational and communication overhead does not make this a feasible alternative in many practical scenarios. The CAN frame data field is just 64 bit followed by a 15-bit CRC, and bus load is limited to 1 Mbps, so adding either authentication tags or authentication is not a viable approach.

The aim of this project is to demonstrate source authentication using fingerprinting of signal characteristics on the physical level as means of uniquely identifying each transducer on the network, and to evaluate the viability of this approach in embedded systems.

The CAN specification allows a great degree of flexibility on the physical layer, and thus physical signals are not identical. Even for transducers of the same model from the same manufacturer, the physical characteristics of the component will vary, and unique identification is possible.

A wider aim of this project is to demonstrate the extent to which computationally intensive tasks such as machine learning can now be performed by small and power friendly devices. Part of this is due to the increasing processing power and energy efficiency of traditional CPU architectures such as ARM, combined with an equally dramatic decrease in associated cost. The other part of the equation is the increasing sophistication of tools for configuring field-programmable gate arrays (FPGA), that let developers more easily deploy highly complex software as hardware. An exciting new generation of hybrid architectures are currently being explored across the industry.

New tools have been developed in this project to further enhance the overlap between computer science and electronics engineering. As far as has been uncovered, this is the first project to use a python API to configure an FPGA on-chip, if at all, including functions for deep neural networks. It is the hope of the author that this project and the tools provided will inspire and lead to further work in complex embedded information technology.

All software developed as part of this project will be made available under a 3-clause BSD license.

Contents

1. Introduction

1.1. Motivation and Related Work

1.1.1. AHMOS

In 2012 Buskerud and Vestfold University College (currently HSN) was approached by industry and faculty members with a request to develop an intrusion detection system for the campus maritime engine and automation lab. The motivation was the increasing number of cyber-attacks experienced by the maritime sector. [3]

An initial Bachelor project was completed by Andreas Karlsen Monstad and Diana Charlotte Paulsen in 2013.[1] The project was further developed by Martin Sundhaug as his Bachelor project in 2015.[2]

Both projects involved building a honeypot consisting of a Raspberry Pi with a PICAN CAN-shield. Karlsen Monstad and Paulsen [1] created the configuration for connecting the detection unit to the lab CAN-bus environment. Sundhaug [2] expanded on this by developing and testing the actual honeypot, including the production of data sets for use with future research.

The honeypot concept involves masquerading as a conventional network unit, whereas in reality being a dummy unit designed to detect unauthorized activity. SCADA (Supervisory control and data acquisition) is an industrial control system (ICS) for remote control and monitoring. These kind of systems have evolved from semi isolated industrial systems to increasingly complex networked systems, and have thus become ever more vulnerable to attack.[4] Using one or more honeypots it is possible to test the robustness of the SCADA system and its ability to withstand attacks.[1] This was the goal of AHMOS part 1 and part 2.

There are a few reasons why this project should not be considered AHMOS part 3. Machine learning research requires large data sets for learning and testing. Early on it became apparent that the HSN engine and automation lab was not an ideal setting for high volume sampling and experimentation. For this reason, a research and development environment was constructed from scratch using various CAN modules and controller boards. This allowed a large number of CAN transceivers to be sampled, as well as great flexibility in experimental design. However, this removed SCADA from the setup. Also, the data set produced by Sundhaug [2] was based on protocol level information only, which made it irrelevant when studying CAN on the physical level.

As a result of these differences, the current project cannot be said to directly relate to maritime systems in particular, as its theoretical implications apply to CAN bus systems in general.

1.1.2. Security in Vehicle Networks

While ships may represent high value targets for potential attackers, automotive transportation represents a much larger quantity of potential targets. This includes both commercial heavy transport as well as personal vehicles. In contrast to shipping, where high value cargo tends to encourage prioritizing security, on road transportation tends to value convenience, which increasingly entails connection to various personal devices, often through wireless connections. The growing complexity and interconnectedness of automotive embedded systems means it is becoming impossible for manufacturers to anticipate possible threat scenarios. Recent papers such as Checkoway, et al. [5], Hoppe, et al. [6] and Rouf, et al. [7] demonstrate that absent source authentication, modern vehicle networks are vulnerable even to the most basic attacks.

1.1.3. CAN Protocol Security Challenges

The challenges in securing a CAN bus environment stems largely from legacy issues with the CAN protocol itself. Development began in 1983, was gradually introduced throughout the 1980s, with the current CAN 2.0 standard released in 1991. [8] This predates the wide spread adoption of the Internet, and represents a context in which the current connectedness was more or less unthinkable.

As a consequence, the CAN frame lends itself poorly to expansions for cryptographic security. Each frame carries only up to 64 bit of data, hence there is not much room for adding an authentication tag. With Message Authentication Codes (MACs) commonly at 128 bit, this is not a realistic option for CAN. Adding additional frames for authentication increases bus load, which in CAN is limited to 1 Mbps. With the complexity of modern systems, this is an upper bound which is often already reached. [9]

Van Herrewege, et al. [10] proposes CAN+ to hide authentication bits within the bits of ordinary CAN frames. However, few controllers exist that can handle this protocol, and the prohibitive cost has prevented the solution from becoming mainstream. Szilagyi and Koopman [11] proposes a low cost multicast solution where each node votes on a message's authenticity based on an 8-bit MAC. As discussed above, this is way below standard MAC length, and not a viable solution for real life scenarios.

Clearly, implementing source authentication on CAN protocol level is problematic.

1.1.4. Physical Layer Security and Digital Fingerprinting

An alternative approach is focusing on physical layer security. As noted above, The CAN specification allows a great degree of flexibility on the physical layer, and thus physical signals are not identical. Even for transducers of the same model from the same manufacturer, the physical characteristics of the component will vary.

Hall, et al. [12], [13] has demonstrated radio frequency fingerprinting for intrusion detection in wireless networks using a Bayesian filter, with an average success rate of (94-100%). Romero-Zurita, et al. [14] used beamforming and artificial noise to disrupt eavesdropper activity in both wired and wireless networks. Gerdes, et al. [15] attempted to identify Ethernet cards based on their synchronization signal, with some success. Gerdes is interesting in particular because their experiment failed to reliably discriminate between cards of the same model. This is where the CAN specification may be in a favourable position compared to other technologies with more tightly defined physical specifications, a notion which has some support in the research.

1.1.5. Fingerprinting CAN Transducers Using DSP Filtering

The direct inspiration for the approach taken in this project comes from Murvay and Groza [9] *Source Identification Using Signal Characteristics in Controller Area Networks*. For this reason, a more detailed discussion of this paper is necessary.

Like previous approaches Murvay and Groza [9] used simple digital signal processing to filter the physical signals. This is similar to Hall, et al. [12], [13] except Murvay and Groza [9] used mean squared error (MSE) and convolution instead of a Bayesian filter. In both cases this consisted of comparing an input signal to a previously stored example, either a single sample or an average of several. This was attempted using both signals smoothed using a low pass filter, as well as using raw unprocessed signal data.

Murvay and Groza [9] generated CAN frames using sysWORXX USB-to-CAN devices and ZK-S12-B f development boards. Signal sampling was done using a high end Agilent MSO6012A oscilloscope with a sample rate of 2 GSa/s and a resolution of up to 12 bits.

This use of high end equipment is problematic. The motivation for using such an expensive oscilloscope was to minimize signal misalignment. However, for results to be applicable to practical implementation, they need to be reproducible using cost effective off-the-shelf components. For this reason, the current project uses a standard on-chip 20 MHz Analog-to-Digital Converter (ADC) to sample signals, but that represents a challenge. Below is a comparison between samples of the CAN Arbitration field made by [9] and sampling done as part of this project. Clearly, the high end oscilloscope is far superior when it comes to bit level sampling. To overcome this, it will be necessary to locate additional unused sources of information above the bit level.

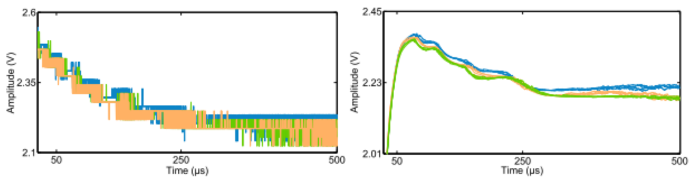


Figure 1 CAN Arbitration field Murvay and Groza

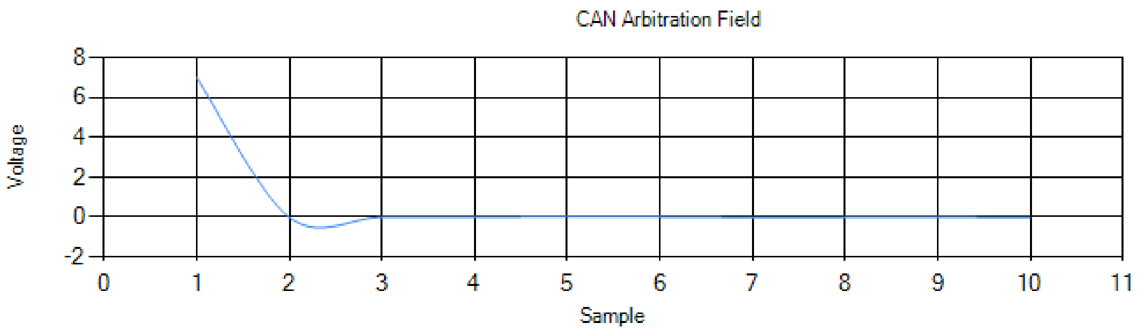


Figure 2 CAN Arbitration Field sampled using Altera De1 SoC 20 MHz ADC

Murvay and Groza [9] found similar results for MSE and convolution. The example below shows MSE values from 10 different PCA82C251 transceivers:

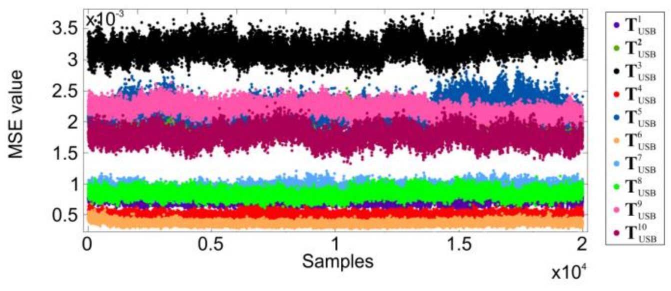


Figure 3 2×10^4 MSE values for PCA transceivers

The tables below show detection accuracy achieved by Murvay and Groza [9] using MSE based detection. Signatures where shown to hold stable over a period of six months. That aspect has not been examined in this study, but is assumed to be valid for these results also. While many tranceivers are easily distinguised, others have very low accuracy.

Interestingly, results appear dependent on CAN ID. Table III uses a different CAN ID, which apparently reduces detection accuracy, yet in some previously difficult cases such as $T^{2'}$ and $T^{3''}$ confusion rate now drops to 0%.

Murvay and Groza [9] uses this apparent curiosity to argue for strategic selection of CAN ID. This is not a viable strategy however, as there is no simple method of choosing the appropriate setting other than trial and error. Also, with accuracy occasionally dropping as

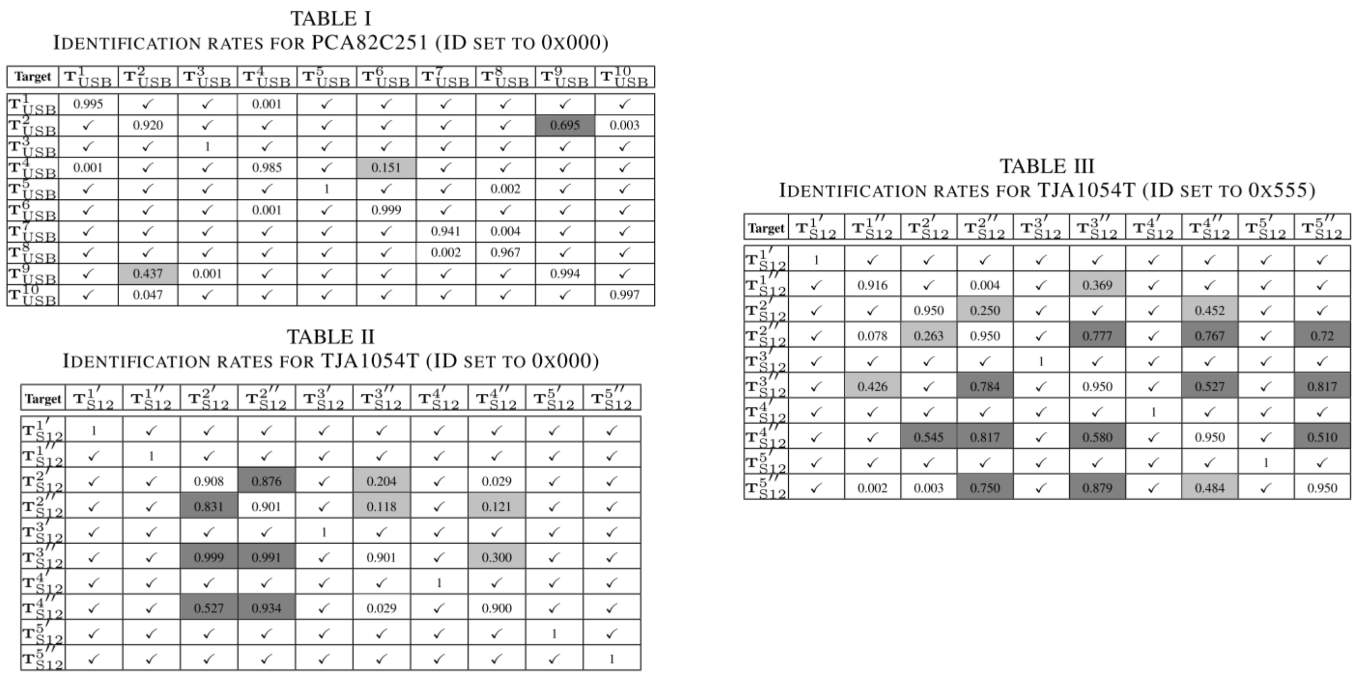


TABLE I
IDENTIFICATION RATES FOR PCA82C251 (ID SET TO 0X000)

Target	T^{1}_{USB}	T^{2}_{USB}	T^{3}_{USB}	T^{4}_{USB}	T^{5}_{USB}	T^{6}_{USB}	T^{7}_{USB}	T^{8}_{USB}	T^{9}_{USB}	T^{10}_{USB}
T^{1}_{USB}	0.995	✓	✓	0.001	✓	✓	✓	✓	✓	✓
T^{2}_{USB}	✓	0.920	✓	✓	✓	✓	✓	✓	0.695	0.003
T^{3}_{USB}	✓	✓	1	✓	✓	✓	✓	✓	✓	✓
T^{4}_{USB}	0.001	✓	✓	0.985	✓	0.151	✓	✓	✓	✓
T^{5}_{USB}	✓	✓	✓	✓	1	✓	✓	0.002	✓	✓
T^{6}_{USB}	✓	✓	✓	0.001	✓	0.999	✓	✓	✓	✓
T^{7}_{USB}	✓	✓	✓	✓	✓	✓	0.941	0.004	✓	✓
T^{8}_{USB}	✓	✓	✓	✓	✓	✓	0.002	0.967	✓	✓
T^{9}_{USB}	✓	0.437	0.001	✓	✓	✓	✓	✓	0.994	✓
T^{10}_{USB}	✓	0.047	✓	✓	✓	✓	✓	✓	✓	0.997

TABLE II
IDENTIFICATION RATES FOR TJA1054T (ID SET TO 0X000)

Target	$T^{1'}_{S12}$	$T^{1''}_{S12}$	$T^{2'}_{S12}$	$T^{2''}_{S12}$	$T^{3'}_{S12}$	$T^{3''}_{S12}$	$T^{4'}_{S12}$	$T^{4''}_{S12}$	$T^{5'}_{S12}$	$T^{5''}_{S12}$
$T^{1'}_{S12}$	1	✓	✓	✓	✓	✓	✓	✓	✓	✓
$T^{1''}_{S12}$	✓	1	✓	✓	✓	✓	✓	✓	✓	✓
$T^{2'}_{S12}$	✓	✓	0.908	0.876	✓	0.204	✓	0.029	✓	✓
$T^{2''}_{S12}$	✓	✓	0.831	0.901	✓	0.118	✓	0.121	✓	✓
$T^{3'}_{S12}$	✓	✓	✓	✓	1	✓	✓	✓	✓	✓
$T^{3''}_{S12}$	✓	✓	0.999	0.991	✓	0.901	✓	0.300	✓	✓
$T^{4'}_{S12}$	✓	✓	✓	✓	✓	✓	1	✓	✓	✓
$T^{4''}_{S12}$	✓	✓	0.527	0.934	✓	0.029	✓	0.900	✓	✓
$T^{5'}_{S12}$	✓	✓	✓	✓	✓	✓	✓	✓	1	✓
$T^{5''}_{S12}$	✓	✓	✓	✓	✓	✓	✓	✓	✓	1

TABLE III
IDENTIFICATION RATES FOR TJA1054T (ID SET TO 0X555)

Target	$T^{1'}_{S12}$	$T^{1''}_{S12}$	$T^{2'}_{S12}$	$T^{2''}_{S12}$	$T^{3'}_{S12}$	$T^{3''}_{S12}$	$T^{4'}_{S12}$	$T^{4''}_{S12}$	$T^{5'}_{S12}$	$T^{5''}_{S12}$
$T^{1'}_{S12}$	1	✓	✓	✓	✓	✓	✓	✓	✓	✓
$T^{1''}_{S12}$	✓	0.916	✓	0.004	✓	0.369	✓	✓	✓	✓
$T^{2'}_{S12}$	✓	✓	0.950	0.250	✓	✓	✓	0.452	✓	✓
$T^{2''}_{S12}$	✓	0.078	0.263	0.950	✓	0.777	✓	0.767	✓	0.72
$T^{3'}_{S12}$	✓	✓	✓	✓	1	✓	✓	✓	✓	✓
$T^{3''}_{S12}$	✓	0.426	✓	0.784	✓	0.950	✓	0.527	✓	0.817
$T^{4'}_{S12}$	✓	✓	✓	✓	✓	✓	1	✓	✓	✓
$T^{4''}_{S12}$	✓	✓	0.545	0.817	✓	0.580	✓	0.950	✓	0.510
$T^{5'}_{S12}$	✓	✓	✓	✓	✓	✓	✓	✓	1	✓
$T^{5''}_{S12}$	✓	0.002	0.003	0.750	✓	0.879	✓	0.484	✓	0.950

Table 1 Murvay and Groza Model Evaluation

low as 0.1%, this method clearly has room for improvement.

This project aims to improve on Murvay and Groza [9] by replacing MSE and convolution filtering with statistical machine learning algorithms. While Murvay and Groza [9] hold that collisions between transceiver patterns are random, the working hypothesis of this project is that it is not.

Also, the modified design will aim for independency from frame content. It will do so by single pulses rather than entire frames.

1.1.6. The Case for Machine Learning on FPGA

Despite the advantages offered by the FPGA for flexibility, power and cooling cost, it has so far proven too cumbersome for many to utilize. Chen and Singh [16] in their paper "*Using OpenCL to Evaluate the Efficiency Of CPUs, GPUs And FPGAs For Information Filtering"* examine how the increased sophistication of software defined hardware is bridging the gap for FPGA when it comes to dynamic and complex computation. With traditional HDL it is necessary to clearly understand the flow through the logic fabric, which essentially made advanced tasks such as machine learning and neural networks exceedingly difficult to implement above the most basic level. With the higher level abstraction of parallel programming languages such as OpenCL, this vastly expands the areas of application where an FPGA might be useful. This is the reason why Intel recently acquired Altera, and why they describe the technology as essential to their core strategy. [17]

1.2. Project Goals

The primary goal of this project is to demonstrate source authentication using machine learning based fingerprinting of signal characteristics on the CAN physical level.

The secondary goal is to demonstrate the viability of machine learning in embedded systems based on FPGA and ARM.

1.3. Research Question and Hypothesis

Based on the previous work referenced above and the project goals stated, a research question can be stated as follows:

Is it possible to reliably separate overlapping CAN transceiver physical signal patterns using Commonly Used Machine Learning Algorithms?

Specifically, this entails:

- **Two-Class Support Vector Machine One-vs-All Multiclass**
- **Multiclass Decision Forest**
- **Multiclass Logistic Regression**
- **Multiclass Decision Jungle**
- **Multiclass Convolutional Neural Network**

The project works from the following hypothesis:

CAN transceiver physical signal patterns contain persistent non-random information that can be extracted with statistical significant reliability above single bit level.

This includes the assumption that the overlap between signals is not random as assumed by Murvay and Groza [9], but can be inferred from information present in the signal.

If this hypothesis is correct, statistical methods should be able to outperform convolution and mean squared error filtering. If any of the examined approaches achieves significantly fewer conflicted cases than was found by Murvay and Groza [9], this would support the hypothesis that additional information is embedded in the signal.

1.4. Project Objectives

Based on the above the project has the following defined objectives:

- **Build a functioning CAN bus with programmable nodes and easily interchangeable transceivers.**
- **Construct SoC device that can sample and store individual CAN signal pulses.**
- **Process data on device using SVM and CNN with ARM/FPGA in combination.**
- **Outperform convolution and mean squared error filtering on hit rate.**
- **Achieve a minimum hit rate of 0.9 for all transceivers.**
- **Achieve mean hit rate of 0.99.**
- **Perform comparably to Nvidia GTX 980Ti GPU**

2. Building a CAN Bus Research Platform

This section describes the development and test environment set up for the project. It was used to collect data to compare performance of Machine Learning algorithms to the findings of Murvay and Groza [9].

Because the nature of the project required high frequency sampling and multiple interchangeable transceivers, it was necessary to build a completely separate environment from the working lab. This included a small CAN bus with multiple programmable nodes with easily replaceable components.

2.1. The CAN Bus

The ISO 11898-1:2003 [18] standard initially outlined the CAN physical level as a bit-level multiple access medium, where access is determined by a succession of dominant and recessive states. ISO 11898-2:2003 further developed the details as to voltage, current and conductors. [19] Beyond this not much is specified about the mechanical aspects of the CAN physical layer, which has resulted in large variation, not only between vendors, but also between models from the same manufacturer, and even batches of the same model, depending on component availability at the time of production.

The network design chosen here relies ISO 11898-2:2003 [19] to achieve noise immunity, by terminating the bus using two 120 Ω transistors. The design was based on the following schematic: [20]

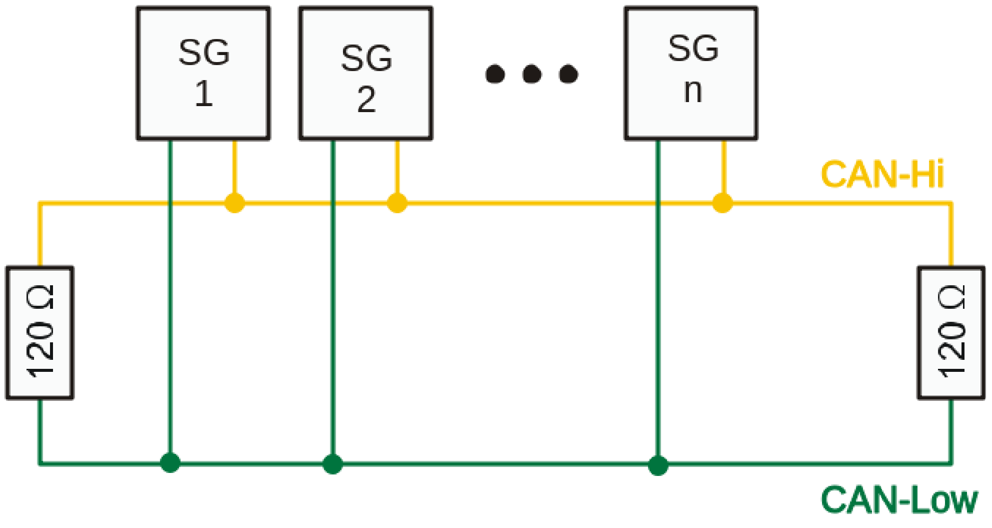


Figure 4 CAN-Bus Topology (Electronic Two Wire Connection) - Wikimedia GFDL

It was implemented as follows:

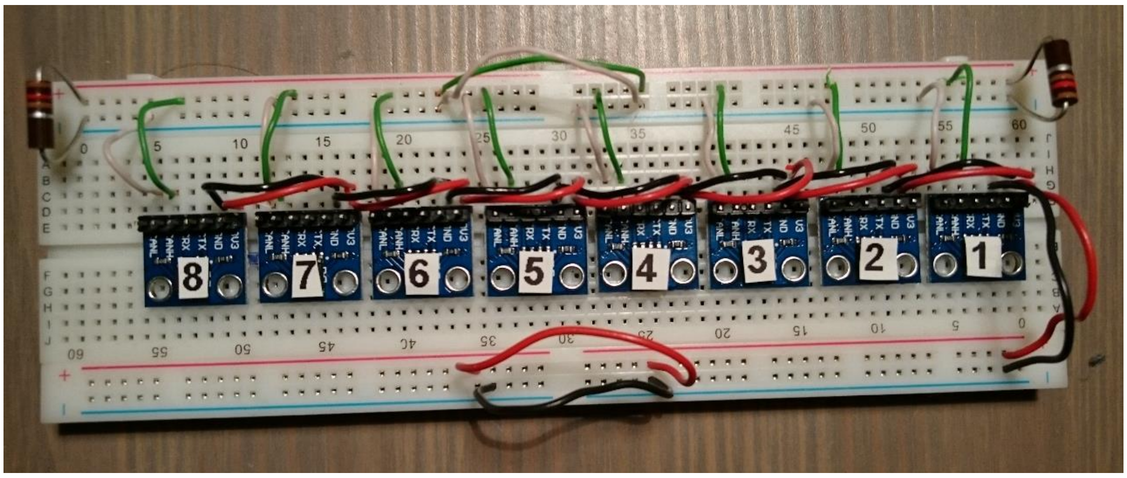


Figure 5 CAN Bus (own photo)

- **Green:** CANH
- **White:** CANL
- **Red:** 3.3 V
- **Black:** Ground

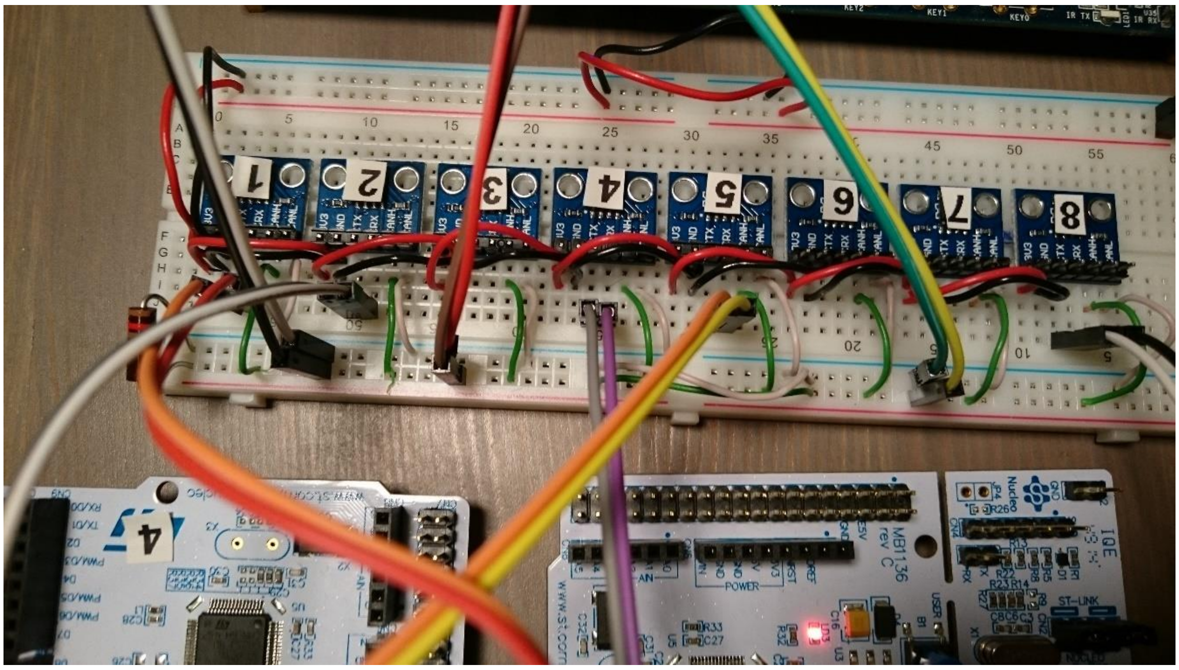


Figure 6 CAN Bus - Connected (own photo)

Intentionally, only the transceiver pins are soldered. The system is deliberately set up to allow fast and flexible reconfiguration.

Because the SN65HVD230 transceiver uses the somewhat less common 3.3 V source, that is also the supply voltage on the network rail. This is not an area that is formally standardized, and typical supply voltages on networks can be as much as 30V. However, since this is a system designed for computer engineers, the lower voltage makes experimentation easier, and makes the system simple to use with inexpensive components such as the SN65HVD230.

2.2. Generating and Detecting CAN frames

Although the protocol level is not the focus of this project, certain aspects of the CAN frame can be taken advantage of. The signal line can be either in a recessive or a dominant state. During a recessive state there is high impedance with respect to both rails, and voltages tend towards roughly half that of the supply voltage for both CANH and CANL. For this particular installation the recessive state voltage is 1.83 V. During a dominant state a low impedance state is induced, CANH is driven towards +5 V and CANL falls towards 0 V.

Most importantly a dominant state can only ever be actively induced, whereas the recessive state is the same as a quiet network.

This means that information is almost exclusively contained in the voltage difference during a dominant state. The differences during a recessive state will be indistinguishable from noise, especially with off-the-shelf equipment as has been deliberately used in this project.

So for the purpose of data reduction, the recessive state will be ignored, as will any voltage difference below + 1 V, including negative differences. That means only 0 bits will be measured, focusing on the shape of the difference in between the voltages, as seen in the image below: [21]

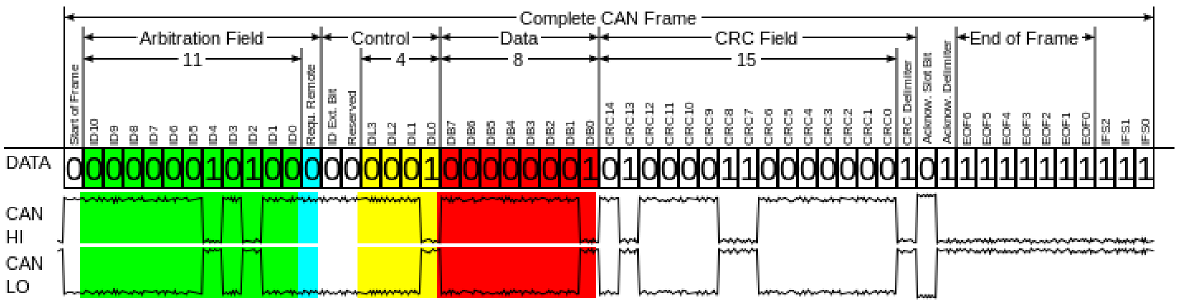


Figure 7 CAN-Bus-frame in base format without stuffbits (wikimedia, Creative Commons)

The motivation behind this choice is not just the limitation of the equipment at hand. By viewing the voltage pattern as essentially a vector of positive numbers, it is possible to treat it as a particularly dense image, which makes it easy to apply algorithms already proven for machine vision, which is a much more mature subject in machine learning than signal analysis. In particular, the data can be formatted according the MNIST format, a flattened 28x28 grayscale image, represented as a single vector with 784 dimensions. [22] [23] [24] [25]

The particulars of the CAN protocol are also useful for synchronization and bit timing without the presence of a clock signal. Synchronization is necessary if one wants to have equivalent samples. Fortunately, the starting bit will always be a logical 0, and hence a dominant state. This project uses a strategy by which the starting bit is listened for as a reference, and then a known frame is used to estimate bit time. This is sufficient for the kind of goal at hand.

2.2.1. TI CAN Bus Physical Layer Transceivers

The TI SN65HVD230 3.3 V CAN transceiver was chosen primarily for its low cost, which made it possible to order in large quantities for experimentation. Despite its low cost the component is compatible with the ISO 11898-2 Standard, and no quality issues arose during the project.

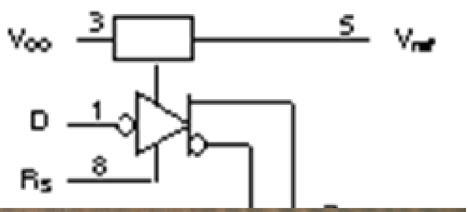


Figure 8 SN65HVD230 Schematic (Texas Instruments)

The transceiver module is incredibly simple, and only implements the CAN physical level, which makes it dependent on a separate controller, which makes it an ideal disposable component in an experimental environment.

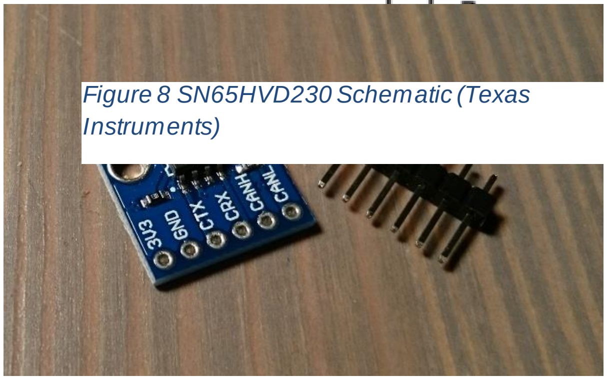


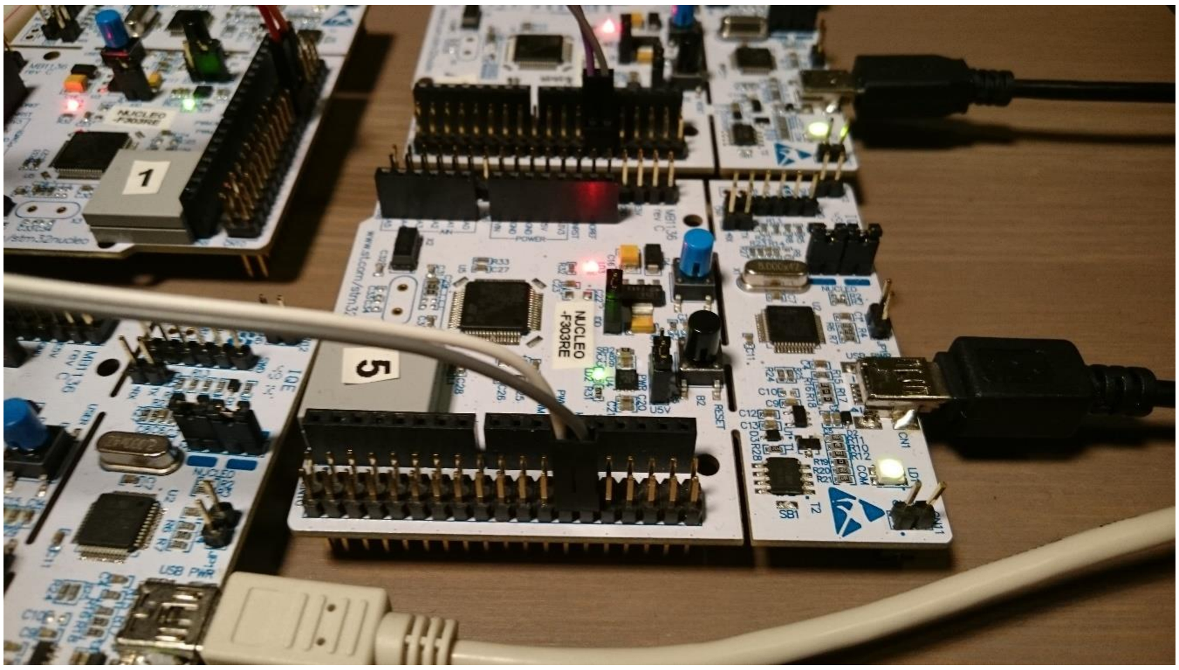
Figure 9 SN65HVD230 Transceiver (own image)

Pins were soldered on so that transceivers could easily be added or removed to the CAN Bus board without any wiring.

2.2.2. Software Controlled Network Nodes

This project was initially intended to be completed v
Lab, but after some review it became apparent that industrial systems are not suited for this kind of development. Part of the motivation for the development platform was to achieve software controlled node behaviour, which is a lot more flexible than the industrial lab, as well as fast.

The STM32 NUCLEO-F303RE board contains a ARM 32-bit Cortex-M4 CPU, and is easily programmed in C/C++ with the ARM mbed platform. Since separate power and ground rails are used in this setup, the board is only connected via CTX/CRX as show below. Programming and power supply are both through as shown USB.



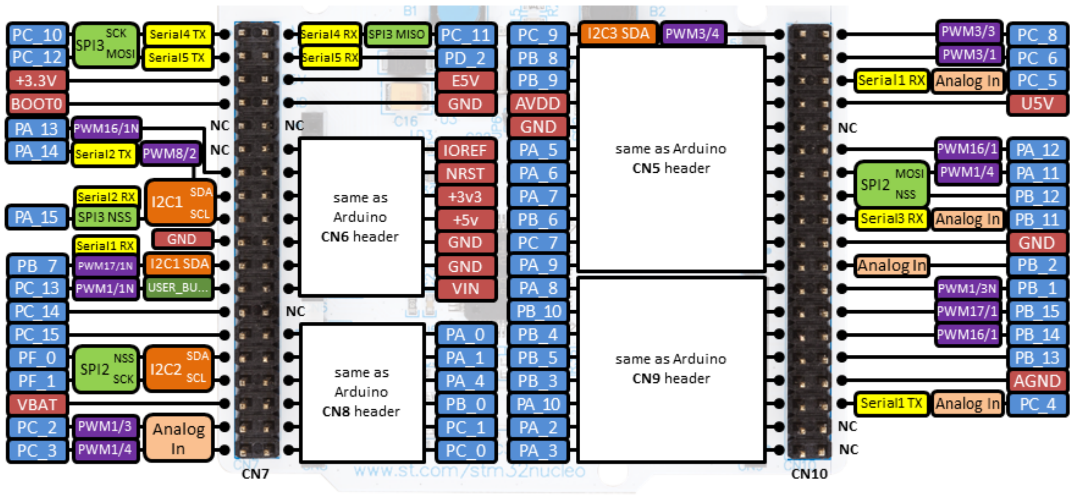
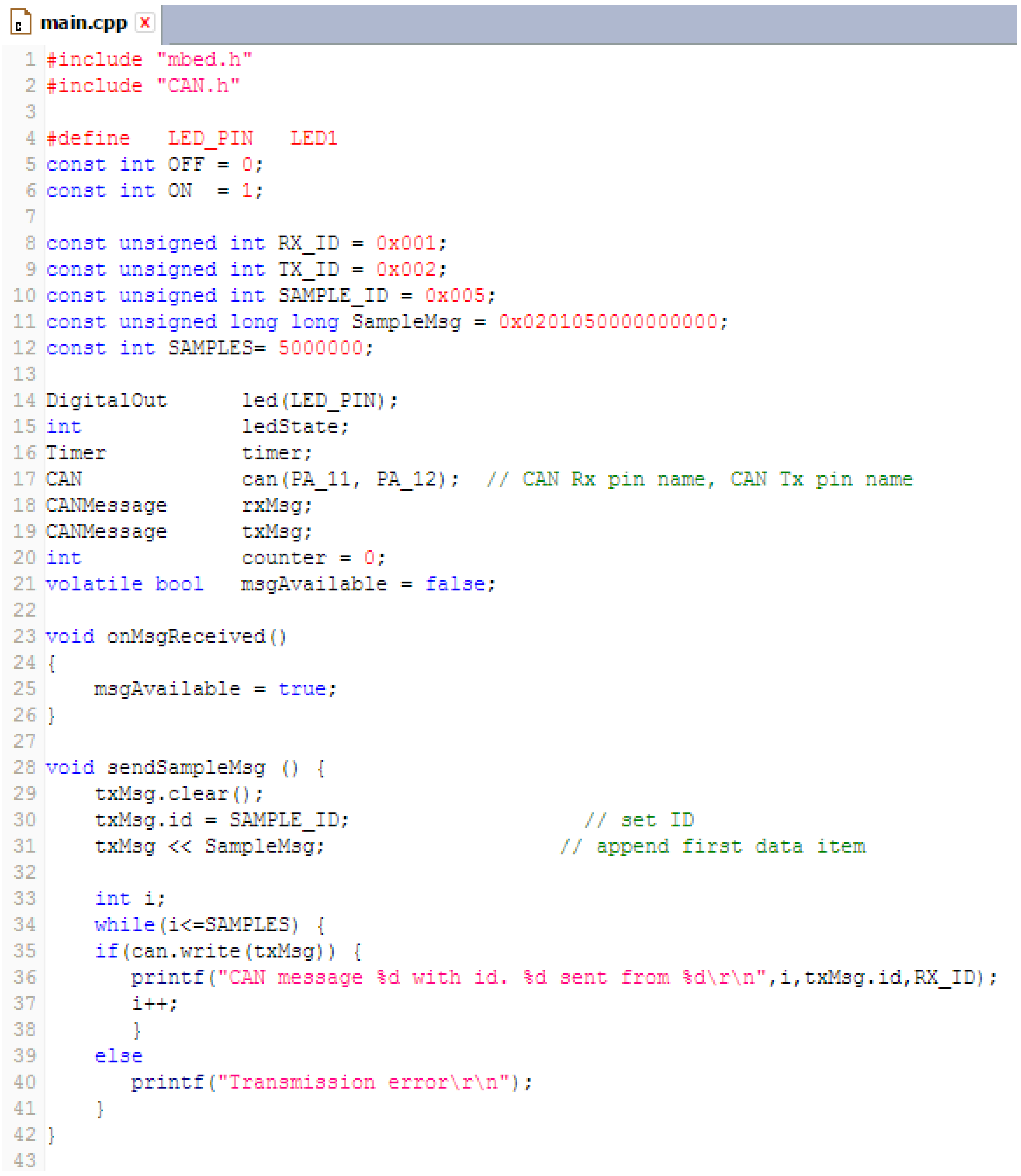


Figure 10 NUCLEO-F303RE Connections (via developer.mbed.org)

RX is connected to the PA_11 pin, TX is connected to PA_11.

By default, the 5 nodes are set up to pass around the same message in an eternal loop, toggling a led. When sampling, a separate function can send a large amount of frames on a single transceiver, before passing the message on to the next in line for the same function to be repeated using a different SN65HVD230. This makes it possible to achieve continuous sampling, even using more than the 8 transceivers that can fit on the bus.

Below is a partial example of the code used for sampling, sending 500 000 identical frames, in this case a coolant temperature request, before passing the message on to the next node in line to take over.


main.cpp

```
#include "mbed.h"
#include "CAN.h"

#define   LED_PIN   LED1
const int OFF = 0;
const int ON  = 1;

const unsigned int RX_ID = 0x001;
const unsigned int TX_ID = 0x002;
const unsigned int SAMPLE_ID = 0x005;
const unsigned long long SampleMsg = 0x0201050000000000;
const int SAMPLES= 5000000;

DigitalOut      led(LED_PIN);
int             ledState;
Timer           timer;
CAN             can(PA_11, PA_12);  // CAN Rx pin name, CAN Tx pin name
CANMessage      rxMsg;
CANMessage      txMsg;
int             counter = 0;
volatile bool   msgAvailable = false;

void onMsgReceived()
{
    msgAvailable = true;
}

void sendSampleMsg () {
    txMsg.clear();
    txMsg.id = SAMPLE_ID;                     // set ID
    txMsg << SampleMsg;                      // append first data item

    int i;
    while(i<=SAMPLES) {
    if(can.write(txMsg)) {
       printf("CAN message %d with id. %d sent from %d\r\n",i,txMsg.id,RX_ID);
       i++;
       }
    else
       printf("Transmission error\r\n");
    }
}
```

Figure 11 Sending 500 000 CAN Frames

2.3. SoC Detection Device

The main device is constructed using a De1 Altera Cyclone V SoC from Terasic. While based on Altera's GHRD (Golden Hardware Reference Design) for SoC, it has been substantially modified for this project, primarily by including additional processors and FIFO shared memory. The full schematics is included in the appendix, and the design files included. Below is a description of the most central components.

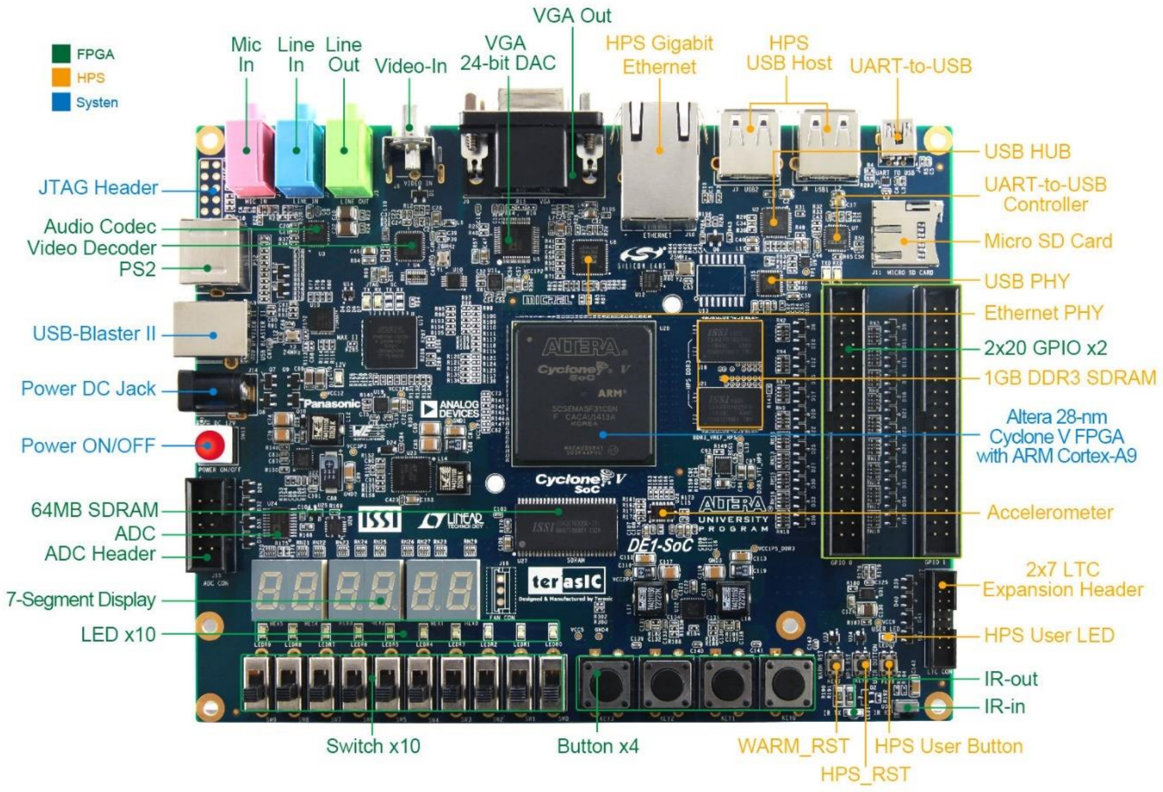


Figure 12 DE1 SoC Layout (provided by Terasic)

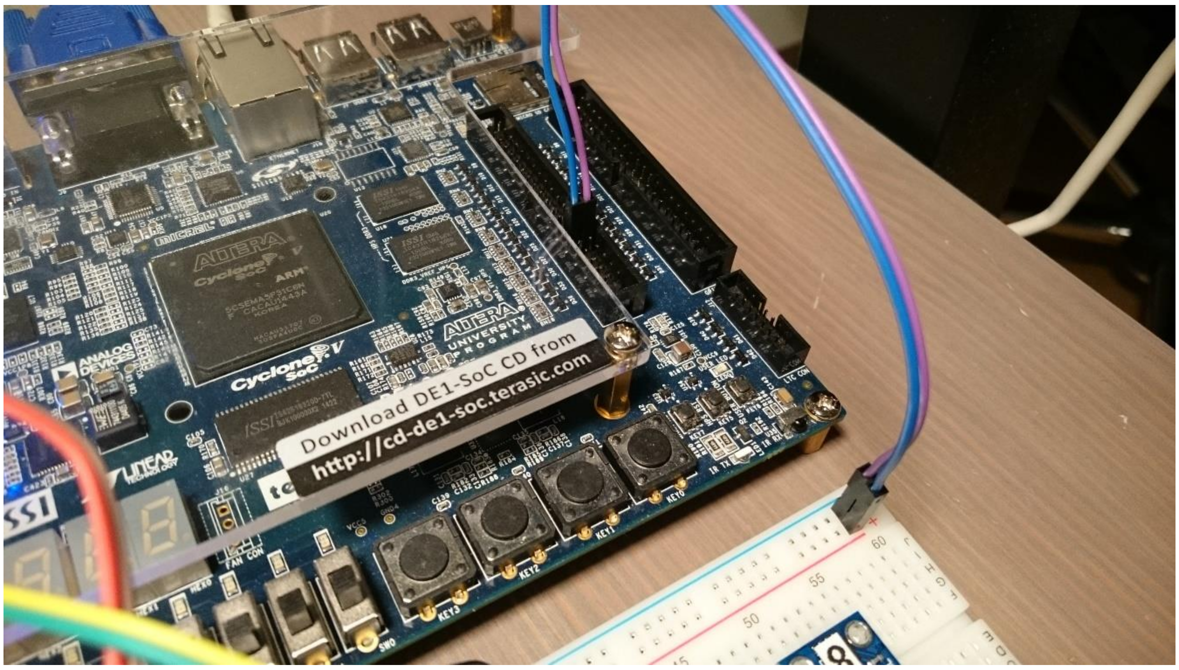


Figure 13 Supply Voltage/Ground provided fron the DE1 SoC

2.3.1. Analog to Digital Converter

The 8 channel 2 bit ADC is a central component, with 20 MHz / 500 KSPS sample rate. This setup uses one channel per bus rail, that is, it samples CANH and CANL on separate channels. In addition, it samples the same signal simultaneously in three different locations on the bus, using a total of six different 12-bit ADC channels to sample the same signal.

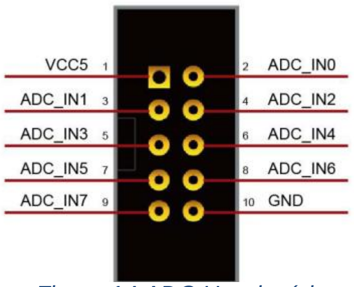


Figure 14 ADC Header (via Terasic)

The motivation behind this configuration is the possibility that additional information might be hidden in the correlations between the six channels, which could outweigh the advantage held by the high end samplers used by Murvay and Groza [9].

The ADC is memory mapped and has its own separate controller set to auto update it can simply be accessed as memory using pointers in C.

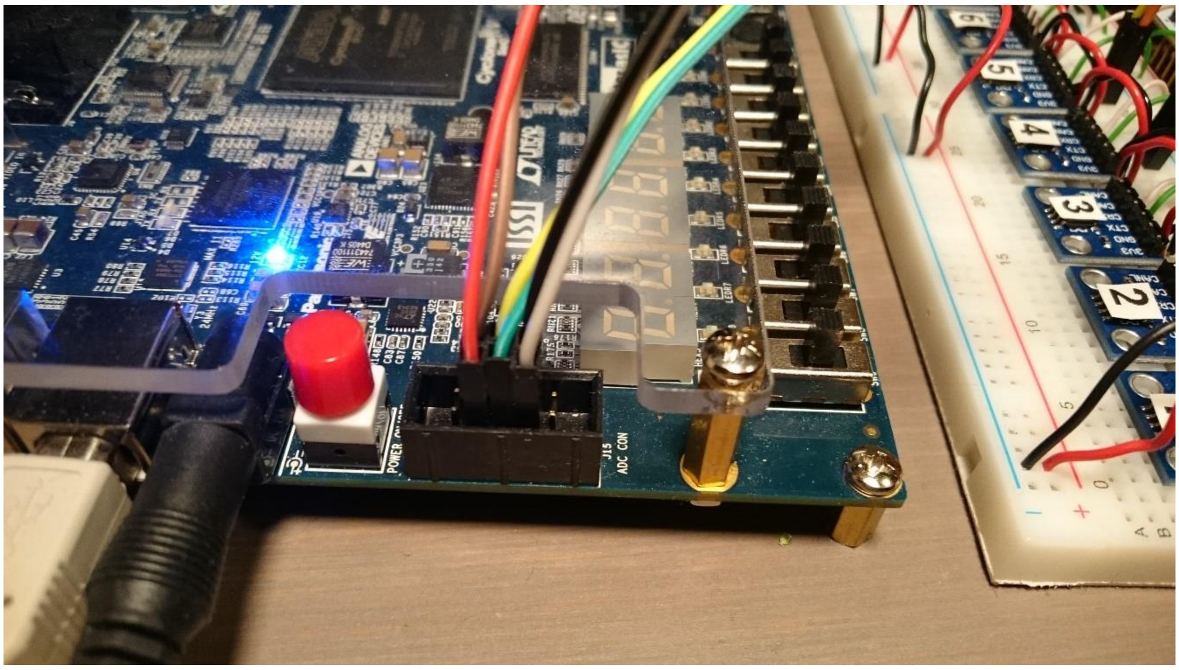
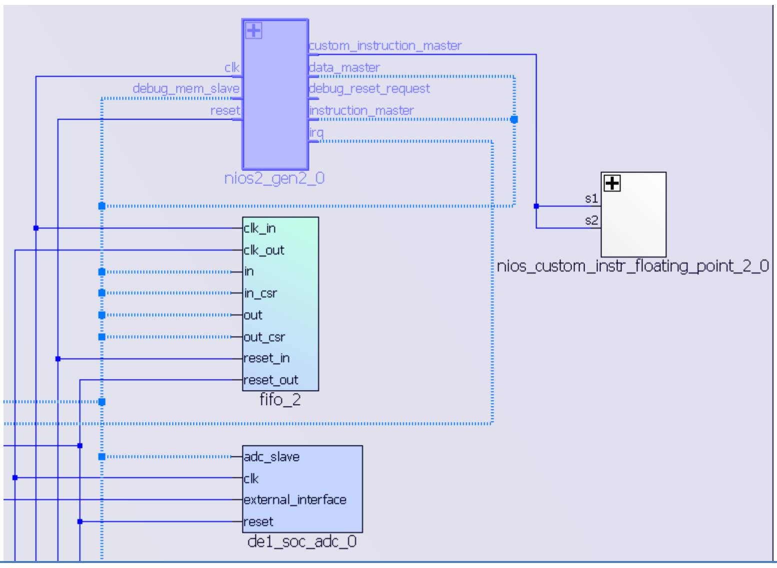


Figure 15 ADC Header DE1 SoC (own photo)

2.3.2. NIOS II Soft Processor



To reserve resources for Linux operations on the ARM CPU, an additional 50 MHz soft processor has been implemented in the logic fabric. It has attached 64 MB RAM, as well as dedicated arithmetic hardware and specialized floating point custom processor.

The main task of the NIOS II is controlling the custom hardware, but it can also be used for pre-processing and software filtering when not wanting to tax the Linux system. Altera provides a high level Hardware Abstraction Level library in c for NIOS

II that is not available from Linux, so hardware control is often accomplished easier and faster from NIOS II.

The soft processor uses the minimal MicroC/OS-II Real Time Operating System, so it is really ideal for keeping the linux installation running smoothly.

2.3.3. ARM Cortex-A9 Hard Processing System (HPS)

The board has a dual core ARM CPU running the lightweight Linux distribution Linaro. The system did initially run on Ubuntu, due to the more sizeable software library, but it became unstable when doing more computationally intensive tasks. System stability improved vastly when changing to Linaro, which is mostly accessed through Eclipse integrated SSH, FTP and gdbserver remote debugging.

The CPU has memory mapped access to the same FPGA connected hardware as does the NIOS II processor, but primarily the role of Linux is networking and OpenCL.

Pre-processed data is passed from NIOS II and the FPGA to the ARM CPU via shared memory, and is distributed by Linux via sockets to either to external recipients or as input to software written in OpenCL used to reconfigure the FPGA.

2.3.4. CAN-Bus Controller

In its current version, the System relies heavily on the ADC for accessing CAN-frames. This is relevant in the context of fingerprinting, but certain tasks would be easier if including on-board CAN controllers. For someone with a background primarily in computer engineering, identifying and timing bits from analogue signals can be a challenge. Using CAN controllers for this seems sensible.

For this reason, two CAN bus controllers are currently implemented in the hardware, connected to the HPS and bridged with the GPIO via the FPGA. However, at the moment accessing these controllers is only possible using memory mapped registers, which is not as convenient as using C on the STM32. Developing a C library for this interface is a future goal for further developing the system.

2.3.5. FIFO Shared Memory

The NIOS II was not designed to be part of an ARM SoC setup, and it is not part of Altera's reference design. One challenge is that the two processors have different type of operating systems and different clock domains. However, the largest challenge was the lack of any conventional communication channel between the two systems. Standard communication interfaces such as RS-232 UART are not compatible between the two systems internally. Altera provides a connection fabric and bridge for access to the other system's peripherals, but direct communication is more complicated.

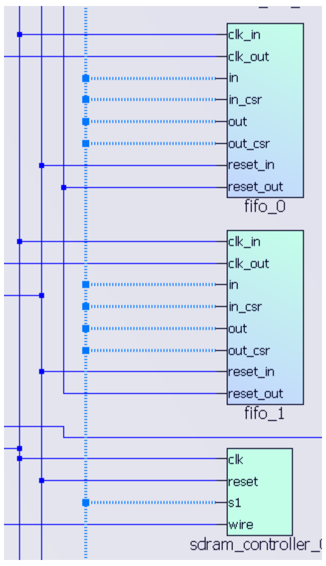


Figure 16 Two FIFO Shared Memory Buffers

The solution for the particular challenge in this project was to set up three large Shared Memory FIFO buffers (up to 8 MB) for data processing. While only NIOS II has a HAL library for the buffer, it can be accessed and controlled from Linux through memory mapping. However, for this reason Linux use is mostly confined to reading processed data.

The FIFO also functions as a clock domain bridge when necessary.

Backpressure is optional. When turned on it ensures data is not unintentionally lost from the buffer, but performance tends to be better if it is turned off. At least at 20 MHz, the challenge is not to keep up with the sampler, but to wait long enough to avoid duplicate data. A good balance for decent performance seems to be allowing possible data loss by letting data flow freely at the sampler's pace, and waiting an additional 2000 microseconds between each reading in Linux.

2.3.6. Altera Cyclone V FPGA

Approximately 30% of the 85K Programmable Logic Elements are taken up implementing custom hardware, leaving the remainder for reconfiguration using OpenCL. Most of the hardware and peripherals are physically connected only to the FPGA, so any access granted the hard processing system need to be justified. A borderline case in this project has been the frame buffer, which uses the FPGA to implement VGA display support, but this was a valuable backup to SSH and serial connection early on in the project when the system was still unstable, as well as for connecting to extra displays. In general, hardware has aggressively been stripped as much as possible to free up as much as possible of the logic fabric for OpenCL.

A likely scenario now that Linux is working properly is to remove the frame buffer, and repurpose the female DE-15 connector now used for VGA as a CAN-bus interface for the HPS.

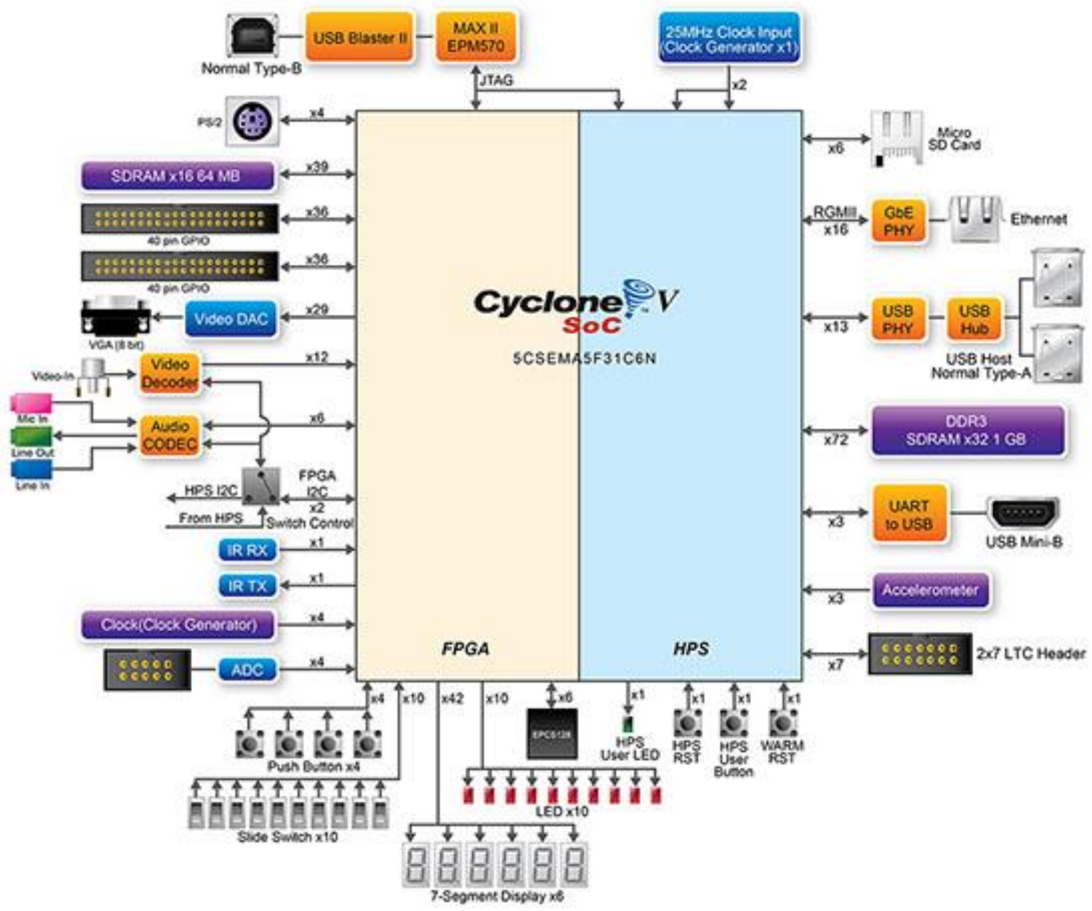


Figure 17 DE1 SoC Layout (via Terasic)

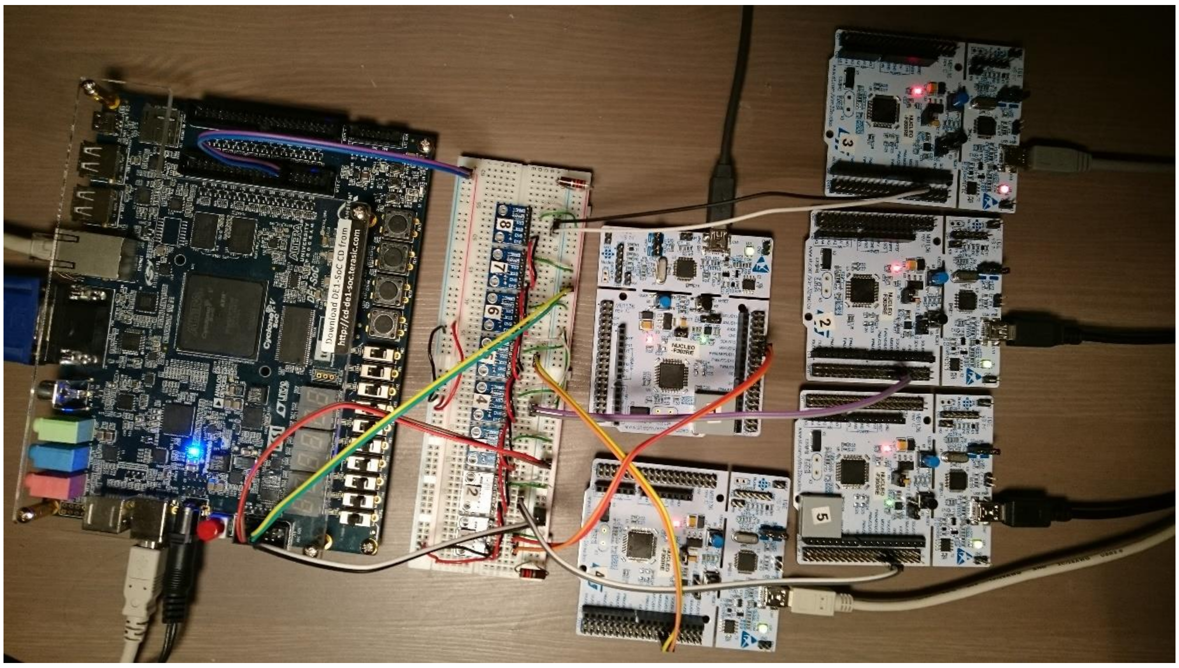
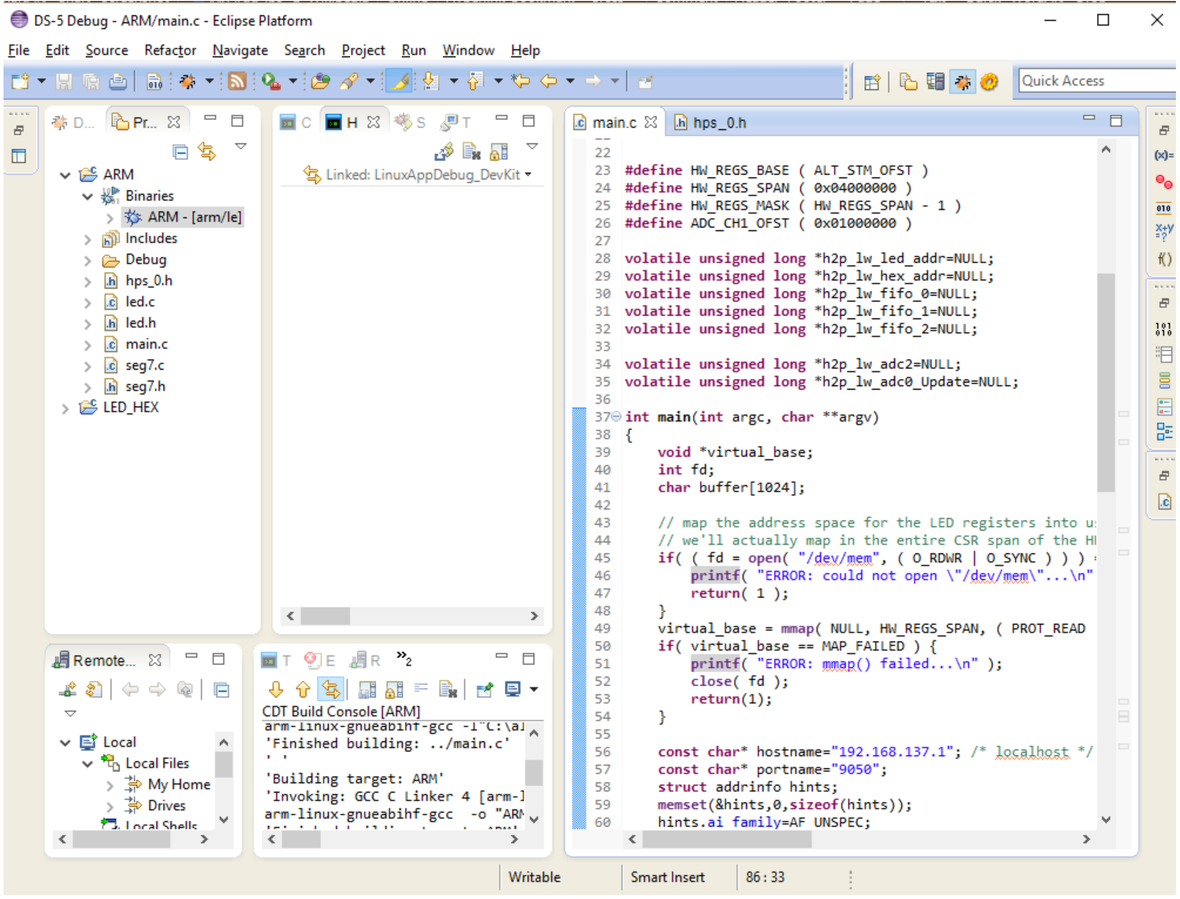


Figure 18 The Complete Development Platform

2.4. PC Development Environment

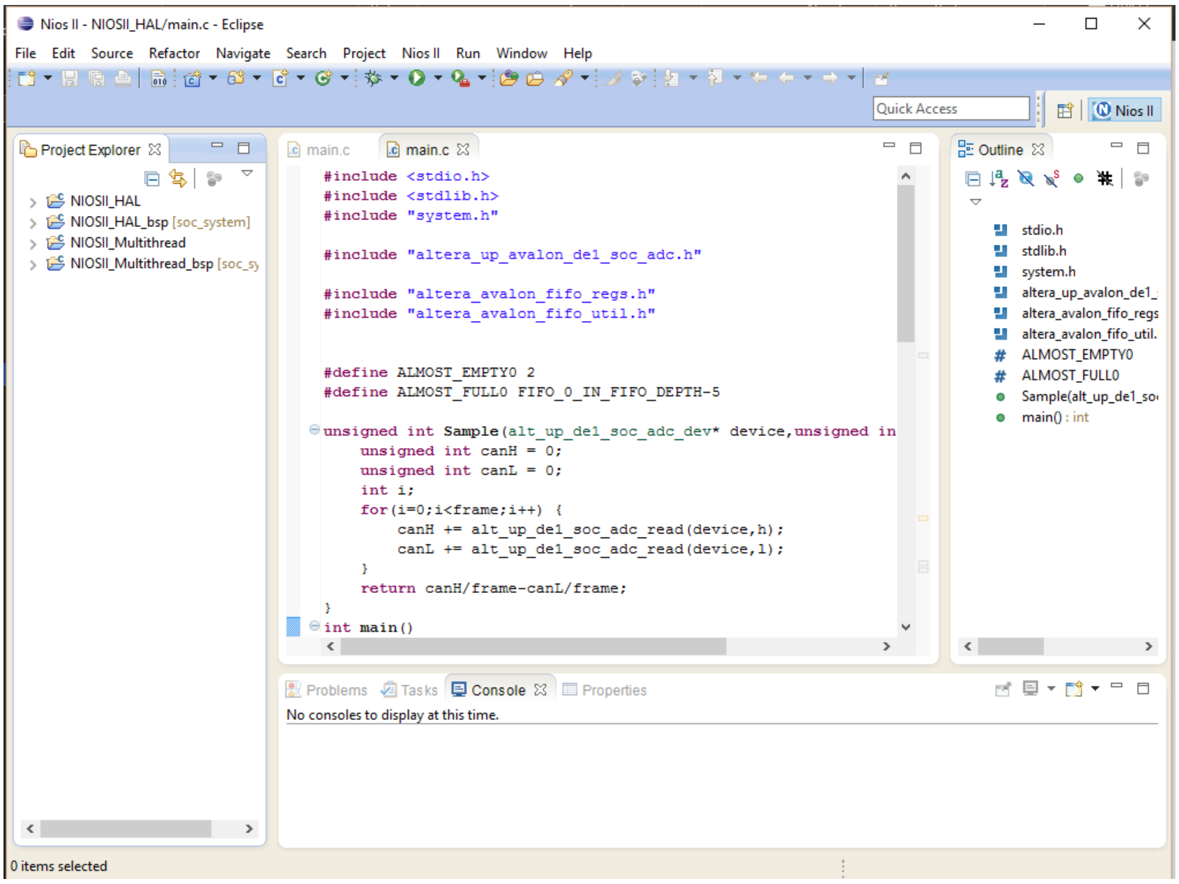
All software programming was done c. NIOS II and ARM Linux software was build using Eclipse with cross compilers and remote debugging. NIOSII code was written both as single thread for the HAL OS and multithreaded for MicroC/RTOS.



```c
#define HW_REGS_BASE ( ALT_STM_OFST )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )
#define ADC_CH1_OFST ( 0x01000000 )

volatile unsigned long *h2p_lw_led_addr=NULL;
volatile unsigned long *h2p_lw_hex_addr=NULL;
volatile unsigned long *h2p_lw_fifo_0=NULL;
volatile unsigned long *h2p_lw_fifo_1=NULL;
volatile unsigned long *h2p_lw_fifo_2=NULL;

volatile unsigned long *h2p_lw_adc2=NULL;
volatile unsigned long *h2p_lw_adc0_Update=NULL;

int main(int argc, char **argv)
{
    void *virtual_base;
    int fd;
    char buffer[1024];

    // map the address space for the LED registers into u
    // we'll actually map in the entire CSR span of the H
    if( ( fd = open( "/dev/mem", ( O_RDWR | O_SYNC ) ) )
        printf( "ERROR: could not open \"/dev/mem\"...\n"
        return( 1 );
    }
    virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ
    if( virtual_base == MAP_FAILED ) {
        printf( "ERROR: mmap() failed...\n" );
        close( fd );
        return(1);
    }

    const char* hostname="192.168.137.1"; /* localhost */
    const char* portname="9050";
    struct addrinfo hints;
    memset(&hints,0,sizeof(hints));
    hints.ai family=AF UNSPEC;
```

CDT Build Console [ARM]

```
arm-linux-gnueabihf-gcc -I"C:\a
'Finished building: ../main.c'
' '
'Building target: ARM'
'Invoking: GCC C Linker 4 [arm-]
arm-linux-gnueabihf-gcc  -o "AR
```



```
#include <stdio.h>
#include <stdlib.h>
#include "system.h"

#include "altera_up_avalon_de1_soc_adc.h"

#include "altera_avalon_fifo_regs.h"
#include "altera_avalon_fifo_util.h"


#define ALMOST_EMPTY0 2
#define ALMOST_FULL0 FIFO_0_IN_FIFO_DEPTH-5

unsigned int Sample(alt_up_de1_soc_adc_dev* device,unsigned in
    unsigned int canH = 0;
    unsigned int canL = 0;
    int i;
    for(i=0;i<frame;i++) {
        canH += alt_up_de1_soc_adc_read(device,h);
        canL += alt_up_de1_soc_adc_read(device,l);
    }
    return canH/frame-canL/frame;
}
int main()
```

```
No consoles to display at this time.
```

3. Machine Learning Algorithms

Despite its recent surge in popularity, machine learning is still mostly the domain of GPUs and data centres. However, academics as well as industry leaders such as Google, Microsoft, IBM and Facebook are increasingly looking to incorporate specialized hardware such as FPGAs into domains that have traditionally been dominated by software. [26] [27] [28] [29]

With frameworks and languages such as OpenCL and OpenCL C becoming ever more accessible and applicable across technologies, the line between hardware and software is blurring. While machine learning in embedded systems and hardware is still in its infancy, there already exists a large selection of well researched and documented methods already developed that can easily be ported.

Zhang, et al. [30] used Rough Set Theory (RST) and Support Vector Machine (SVM) to detect network intrusions. This project does not go into anomaly detection, but it does go into SVM as a classification algorithm. Clustering is also promising field for security applications.

Because of the immaturity of the field, there is still much that can be desired when it comes to efficient development tools. For the sake efficiency, five common approaches to multiclass classification were compared using the compiled CAN bus data set in Microsoft Azure Machine Learning Studio. This is a cloud based IDE that makes it quick and easy to set up multi-tier experiments for comparing different permutations of a model.

Two models where then implemented on the DE1 SoC using the Altera OpenCL SDK.

This section briefly describes the five approaches, and the next section explains the experimental setup used to compare them.

Details of the models will be covered when describing the configuration of the experiment.

3.1. Two-Class Support Vector Machine

Support Vector Machine (SVM) is originally a binary classification model, essentially drawing a line separating two clusters of objects. In that way it is related to common methods in the intersection between linear algebra and statistics, such linear regression. What makes it popular is that it can be extended beyond three dimensions, with dimensions here representing features as opposed to dimensions in space. The idea is that if a solution cannot be found in one dimension, transforming the problem to another dimension could result in a solution. Using linear algebra, it is possible solve for infinite dimensions, taking SVM close to unsupervised learning and clustering. The highly mathematical nature of the approach means that it is one of the few methods where a certain level of proof is possible, but it also makes the model less intuitive. Because it based in generalized linear models, it is also highly suitable for use in programming.

The relevant parameters are lambda and the number of iterations. Lambda is a weight variable used for L1 regularization.

However, despite some impressive results, it generally is only directly applicable in clearly defined binary cases with labelled data.

3.1.1. One-vs-All Multiclass

One way of overcoming the binary restriction of SVM is to combine multiple binary classifications in the same model. In this case, each class of object is compared with all others as being “one of these” or “not one of these”. These types of model are naturally not very scalable.

3.2. Multiclass Decision Forest

A Decision Forest is what is known as an ensemble method, based on a theoretical model of cooperation and competition. The model generates a large number of solutions in the form of decision trees, and then weeds out bad and filters out the good by voting as a method of aggregation. The tree structure is efficient in use of memory, and the model is flexible when it comes to structure.

3.3. Multiclass Logistic Regression

Multiclass or Multinomial Logistic Regression is similar to SVM in that it aims to find a best fit through an n-dimensional feature space, but the mathematics is much less convoluted. The setup solution procedure is exactly the same as with simple linear regression, only with additional feature variables, resulting in a larger matrix.

3.4. Multiclass Decision Jungle

Decision Jungles are an extension on Decision Forests. Both generate and then aggregate decision trees, but with Decision Jungles there is the additional option of allowing branches to merge, resulting in a much reduced memory footprint. Decision Jungles are highly flexible, non-parametric and non-linear, meaning they are also highly noise tolerant.

3.5. Multiclass Convolutional Neural Network

Convolutional Neural Networks are biologically inspired, highly parallel multilevel networks. Nodes in the network perform transformations on data transported through the edges, or tensors. It gets its name from having one or more layers performing convolution transformations on data, but also have additional visible or hidden layers for operations such as reduction and pooling. Often the final layer is a form of logistic regression.

4. Experimental Design

Choosing a model for implementation would be difficult simply using the tools available for embedded development. Altera SDK for OpenCL delivers excellent performance on-chip after compilation, but the experience was seriously degraded by several bugs, especially regarding version compatibility and licence setup. The integration with Embedded Design Suite and DS-5 Eclipse was severely lacking and unstable. The one provided BSP was severely lacking as a workable Linux distribution, and little documentation existed concerning how to compile an OpenCL compatible BSP. It therefore soon became apparent that a better option was to develop on an Nvidia GTX980Ti GPU before recompiling using the Altera compiler through the command line interface. This approach worked fairly well.

Some experimentation was done using MATLAB with FPGA-in-the-loop as well as TensorBoard from Google. Eventually the choice fell on Microsoft Azure Machine Learning Studio for the final comparison. MLS has the benefit of an intuitive drag-and-drop interface for setting up experiment designs, as well as support for Python and R for more low level control. It also has probably the fastest and simplest tools for converting between data set formats. The design was based on the default template for comparing multi-class classifiers, configured for the compiled data set. An additional Deep Neural Net was added later in addition to the standard convolutional neural net.

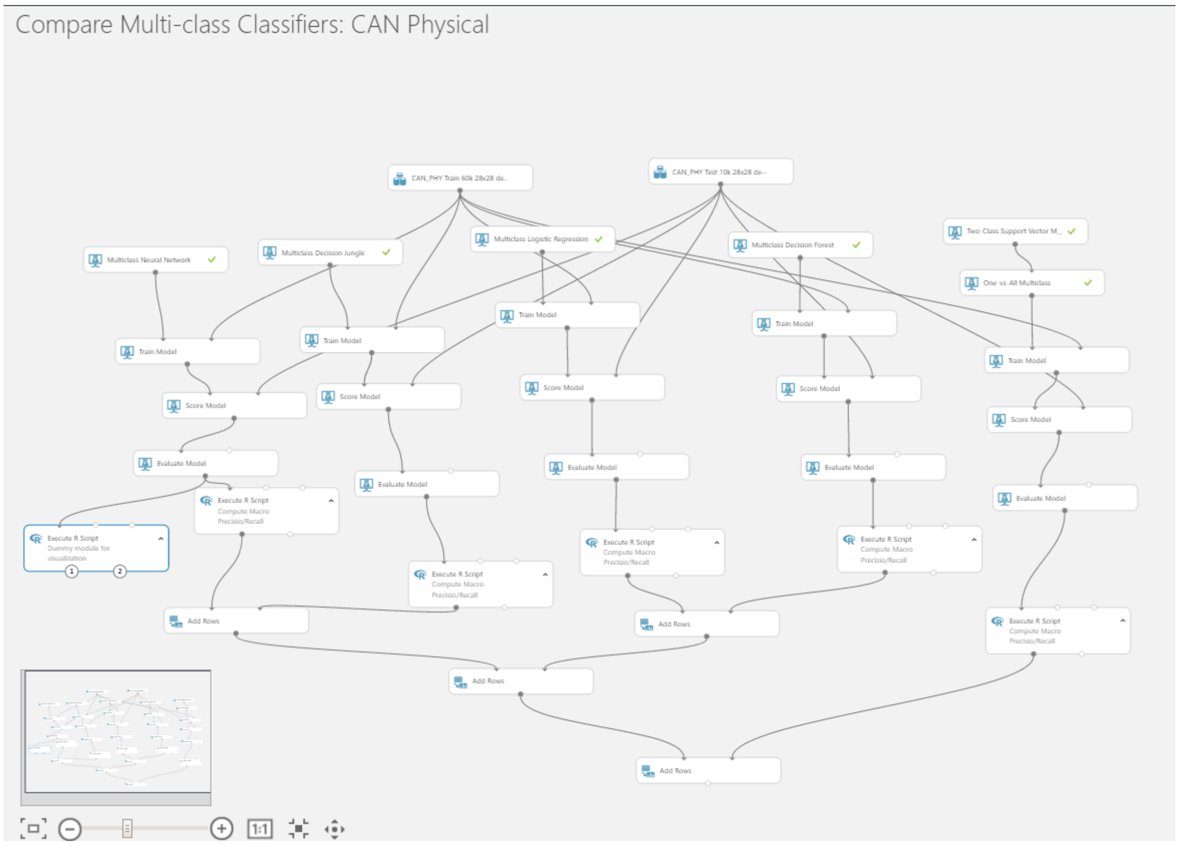
4.1. Data Reduction and Representation

As mentioned previously, focus was placed completely on voltage differences between channels in the dominant state. Also, samples were taken simultaneously at six different locations, three for each rail on the bus. A single sample consisted of a vector 784 values long, based on the MNIST format used for handwritten letter classification. Based on previous research, it was assumed that the exact order was irrelevant as long as it was consistent across the experiment. The adjoining CANH and CANL channels where combined in The FIFO before parsing, leaving three values representing the voltage difference between each of the dual signals. Leaving the first position for the label, that gave the final format for the original comma separated CSV-file:

“$can#,value0,value1,value2…..*261”

i.e. the input vector consisted of 261 sets of 3 values, representing different perspectives of the voltage difference between CANH and CANL at a given time. Value sets within a vector was put in chronological order, whereas the order of input vectors were randomized. 60K input vectors were reserved for training, and 10K was saved for testing and scoring. White space resulting from recessive states was ignored.

4.2. Comparing Multi-class Classifiers


Compare Multi-class Classifiers: CAN Physical
CAN_PHY Train 60k 28x28 de...
CAN_PHY Test 10k 28x28 de...
Multiclass Neural Network
Multiclass Decision Jungle
Multiclass Logistic Regression
Multiclass Decision Forest
Two Class Support Vector M...
One-vs-All Multiclass
Train Model
Score Model
Evaluate Model
Execute R Script
Dummy module for visualization
Compute Macro Precision/Recall
Add Rows

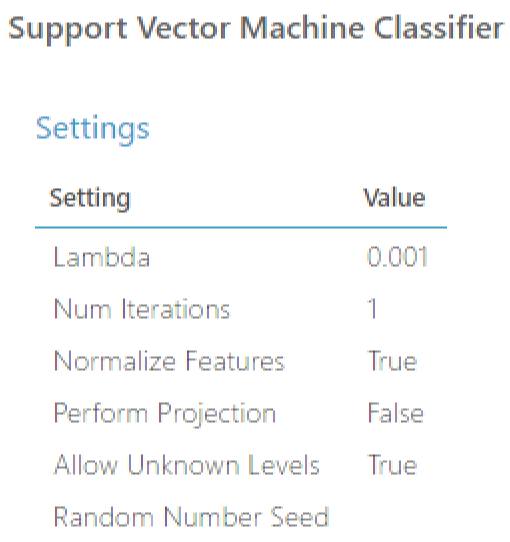
A model was constructed from the default template. Each node represents a process, and data and activity flows downwards. Each column represents a different classifier. The final layers are R script pooling and data visualization.

4.2.1. Two-Class Support Vector Machine

The first level in the model defines the settings for the SVM. It was kept at single parameter setting with a Lambda of 0.001. The module I was set to normalize the data, and it was set to accept infinite dimensional and hidden solutions.

The second level does exactly the same thing for the One-vs-All SVM.

Data is them passed to the training module for learning.


Support Vector Machine Classifier

Settings

Setting	Value
Lambda	0.001
Num Iterations	1
Normalize Features	True
Perform Projection	False
Allow Unknown Levels	True
Random Number Seed	

4.2.2. Multiclass Decision Forest

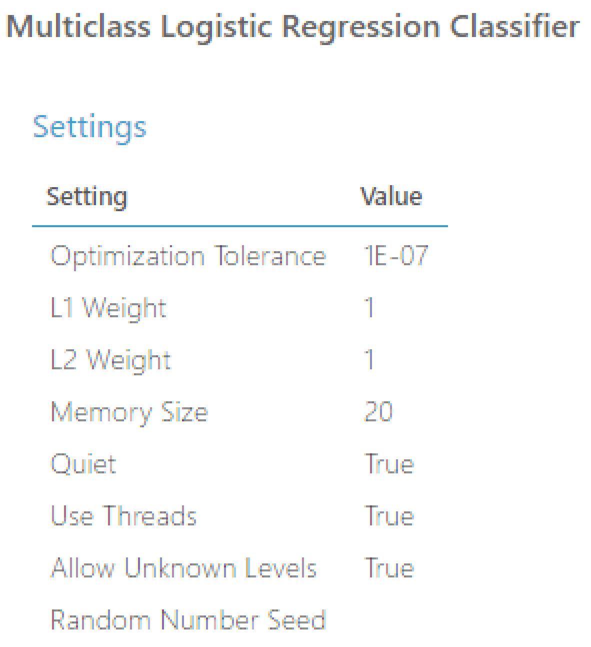
The Decision Forest is also kept at single parameter. The max depth of any decision tree is kept at 32. Bagging is set as the resampling method. This means that resampling is with a new random sample of the original dataset, as opposed to simple replication.

4.2.3. Multiclass Logistic Regression

4.2.4. Multiclass Decision Jungle

The Decision Jungle is set to exactly the same settings as the Decision Forest.

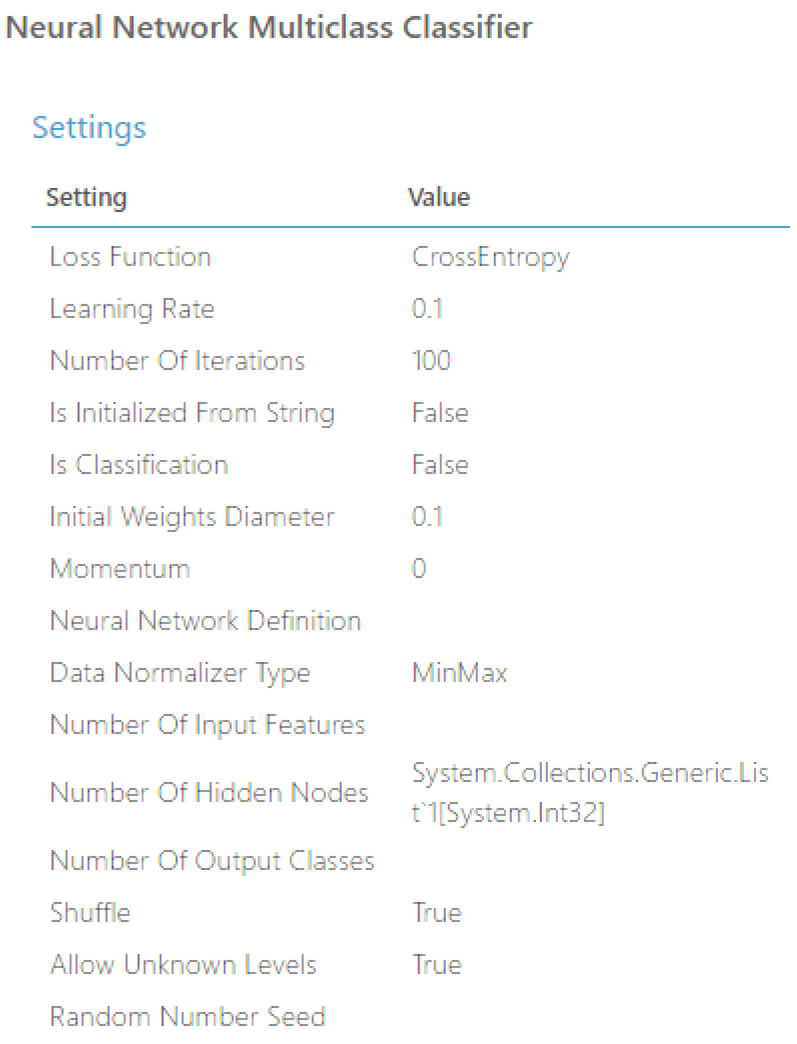
4.2.5. Multiclass



Multiclass Logistic Regression Classifier

Settings

Setting	Value
Optimization Tolerance	1E-07
L1 Weight	1
L2 Weight	1
Memory Size	20
Quiet	True
Use Threads	True
Allow Unknown Levels	True
Random Number Seed	



Neural Network Multiclass Classifier

Settings

Setting	Value
Loss Function	CrossEntropy
Learning Rate	0.1
Number Of Iterations	100
Is Initialized From String	False
Is Classification	False
Initial Weights Diameter	0.1
Momentum	0
Neural Network Definition	
Data Normalizer Type	MinMax
Number Of Input Features	
Number Of Hidden Nodes	System.Collections.Generic.List`1[System.Int32]
Number Of Output Classes	
Shuffle	True
Allow Unknown Levels	True
Random Number Seed	

Convolutional Neural Network

The simple CNN was set as a fully connected net with 100 hidden nodes, 100 iterations for learning, and MIN-MAX as normalisation method.

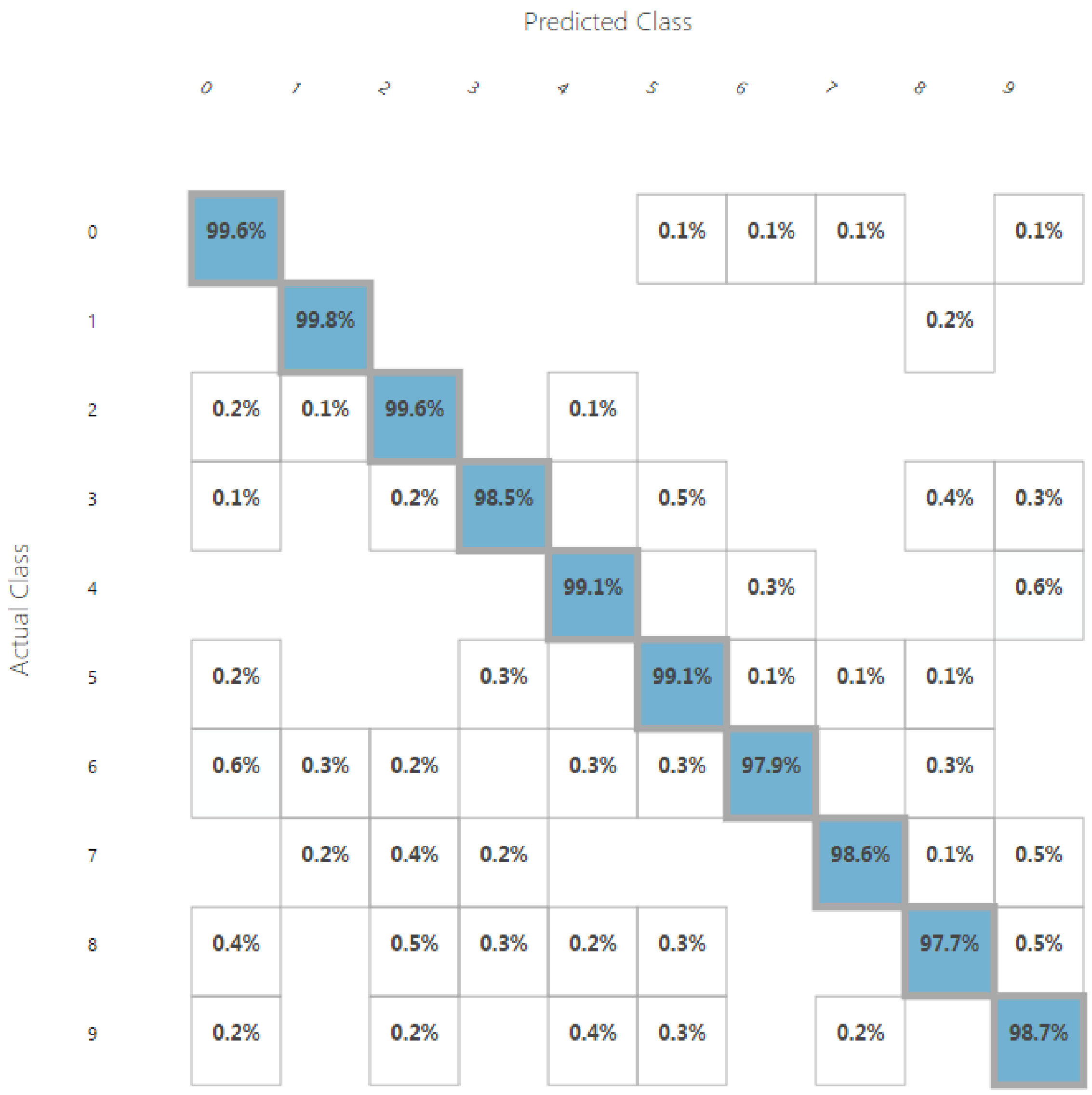
A second, deeper network was set up using R (source code provided), but here the number of iterations where kept at 20 due to processing requires.

This model was also run in a separate session, but on the same data set.

5. Results

Not surprisingly, the deep net outperformed the other models. A more surprising result was the poor performance of SVM.

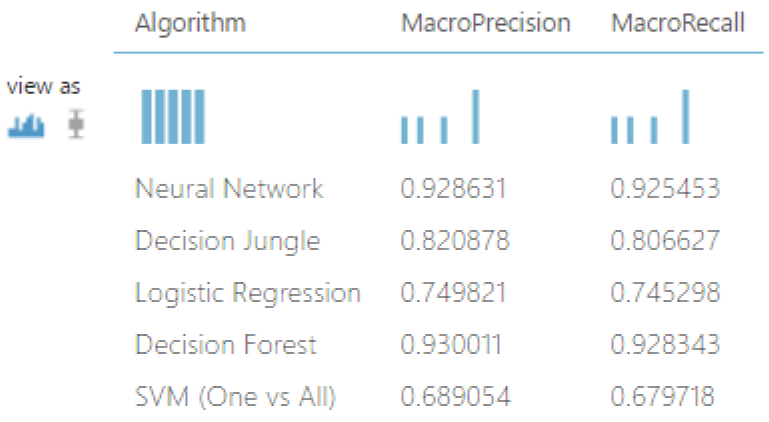
The Matrix below show results for the deeply connected network analogue to the to the results Murvay and Groza [9] presented. Clearly the deep network out-performed Mean Square Error and Convolution. However, this required several minutes of cloud based GPU processing, so it can hardly be expected to be running on embedded units any time soon.


Predicted Class
Actual Class

	0	1	2	3	4	5	6	7	8	9
0	99.6%					0.1%	0.1%	0.1%		0.1%
1		99.8%							0.2%	
2	0.2%	0.1%	99.6%		0.1%					
3	0.1%		0.2%	98.5%		0.5%			0.4%	0.3%
4					99.1%		0.3%			0.6%
5	0.2%			0.3%		99.1%	0.1%	0.1%	0.1%	
6	0.6%	0.3%	0.2%		0.3%	0.3%	97.9%		0.3%	
7		0.2%	0.4%	0.2%				98.6%	0.1%	0.5%
8	0.4%		0.5%	0.3%	0.2%	0.3%			97.7%	0.5%
9	0.2%		0.2%		0.4%	0.3%		0.2%		98.7%

6. Conclusions

The results support the hypothesis that more information is present in an analogue signal than is immediately available and obvious. Unfortunately, the only network able to perform at the desired standard is one that is nowhere close to be running on an FPGA or ARM system.



Algorithm	MacroPrecision	MacroRecall
Neural Network	0.928631	0.925453
Decision Jungle	0.820878	0.806627
Logistic Regression	0.749821	0.745298
Decision Forest	0.930011	0.928343
SVM (One vs All)	0.689054	0.679718

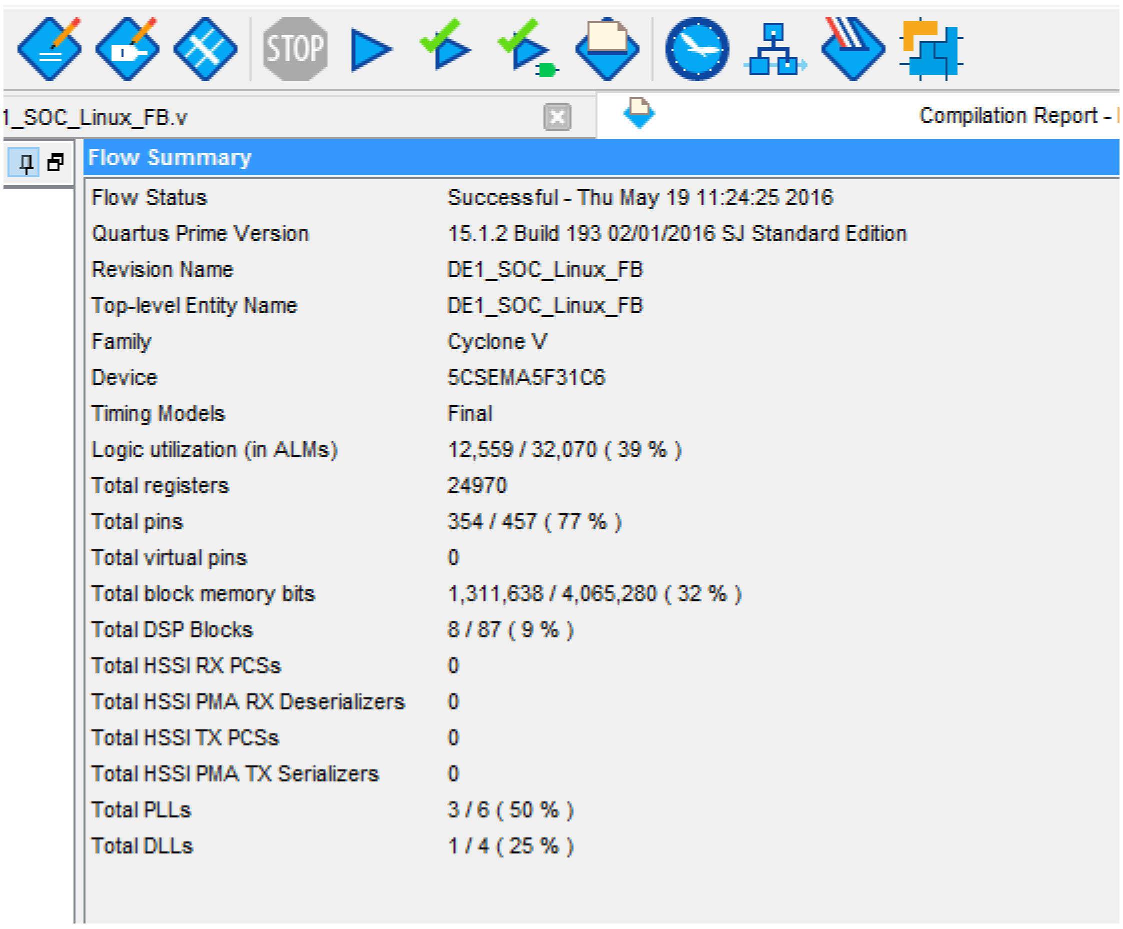
However, that predictions even in the low 90s is achievable is very impressive.

It is fully possible that with additional experience with writing code specifically for the FPGA, rather than simply recompiling code written for the GPU, additional improvement can be seen in the current generation of hardware.

In hindsight, the most rewarding part of the project was not getting code to run on the DE1 SoC, but the experience of building a working CAN-bus research environment.

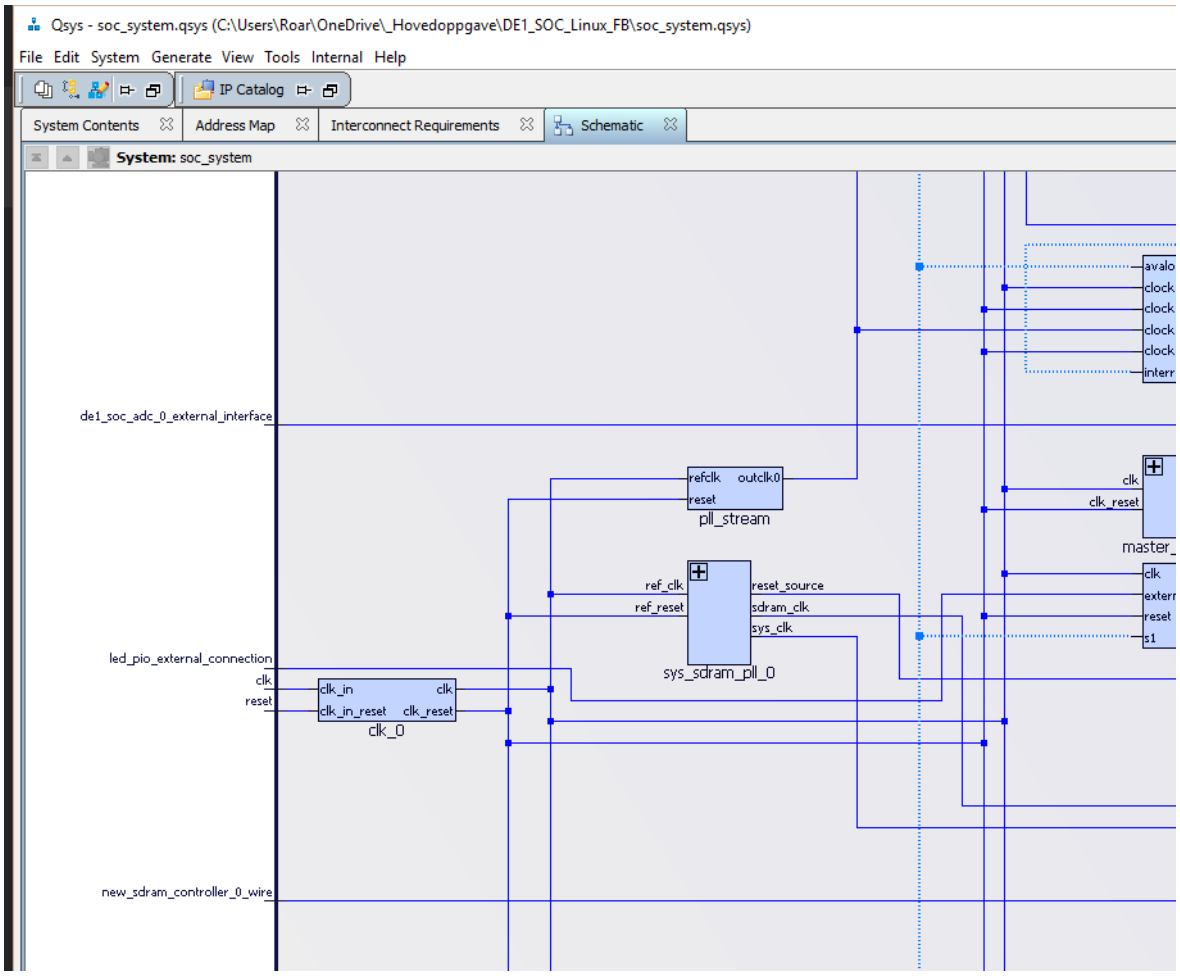
7. SoC Hardware Design

Hardware was designed primarily in Altera Qsys, and compiled for FPGA using Quartus Prime.



Flow Summary

Flow Status	Successful - Thu May 19 11:24:25 2016
Quartus Prime Version	15.1.2 Build 193 02/01/2016 SJ Standard Edition
Revision Name	DE1_SOC_Linux_FB
Top-level Entity Name	DE1_SOC_Linux_FB
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	12,559 / 32,070 (39 %)
Total registers	24970
Total pins	354 / 457 (77 %)
Total virtual pins	0
Total block memory bits	1,311,638 / 4,065,280 (32 %)
Total DSP Blocks	8 / 87 (9 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	3 / 6 (50 %)
Total DLLs	1 / 4 (25 %)

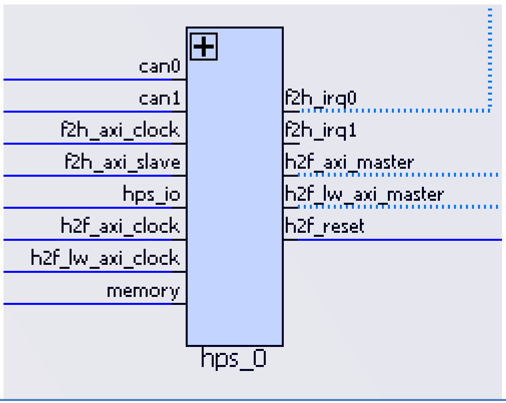

Qsys - soc_system.qsys (C:\Users\Roar\OneDrive_Hovedoppgave\DE1_SOC_Linux_FB\soc_system.qsys)
File Edit System Generate View Tools Internal Help
IP Catalog
System Contents
Address Map
Interconnect Requirements
Schematic
System: soc_system
de1_soc_adc_0_external_interface
led_pio_external_connection
clk
reset
new_sdram_controller_0_wire
clk_in
clk
clk_in_reset
clk_reset
clk_0
refclk
outclk0
reset
pll_stream
ref_clk
ref_reset
reset_source
sdram_clk
sys_clk
sys_sdram_pll_0
clk
clk_reset
master_
avalo
clock
interr
exter
s1

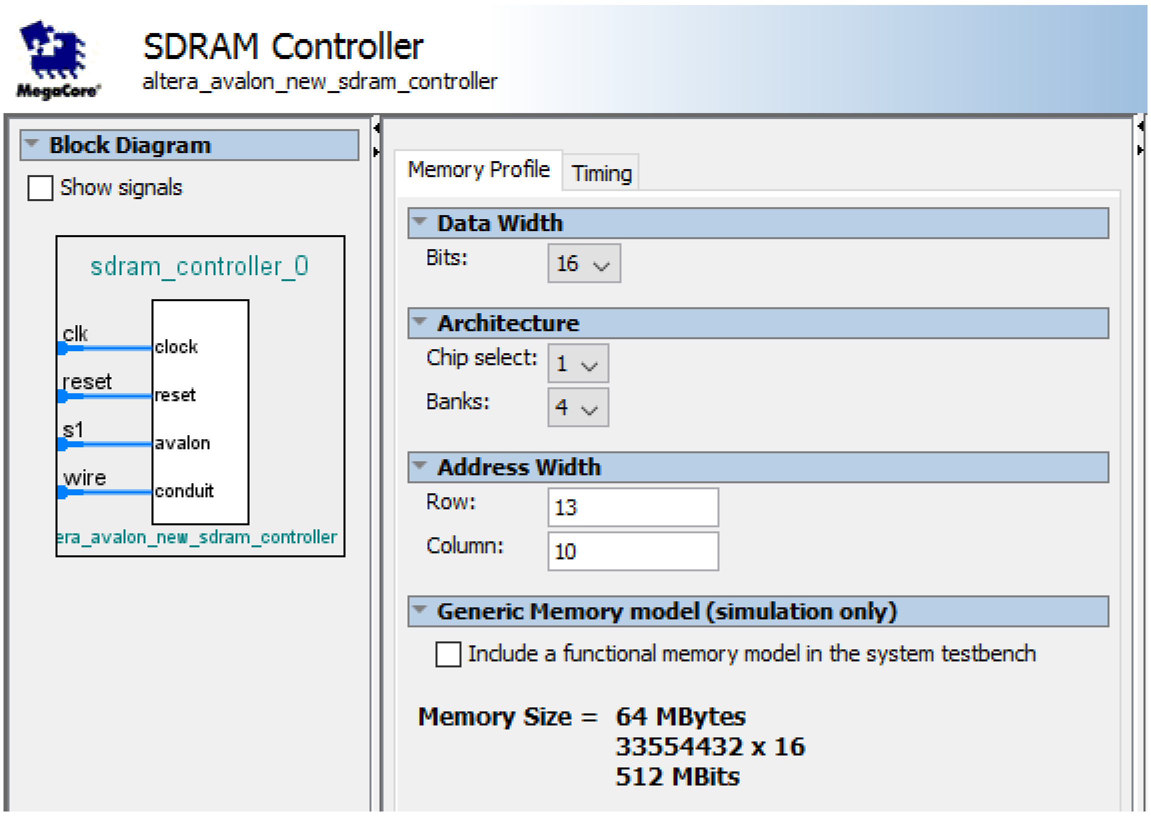
7.1. HPS/FPGA Interface

The interface between the ARM CPU and the FPGA is kept largely as per the Terasic/Altera reference design. The only modification has involving removing an SPI interface to make room for the CAN-interface. The two CAN interfaces each has their own controller, and each connection seen on the schematic has both an RX and a TX pin.

7.2. Replacing on-chip Memory for SDRAM

The reference design for NIOS II is based on on-chip memory. For this design that was replaced with 64 MB of SDRAM. This required installing an additional controller and a separate clock domain for the NIOS subsystem. For this reason, it was necessary to implement the FIFO shared memory with a dual clock system. The SDRAM time had to be set to be board specific.




SDRAM Controller
altera_avalon_new_sdram_controller

Data Width

Bits:	16

Architecture

Chip select:	1
Banks:	4

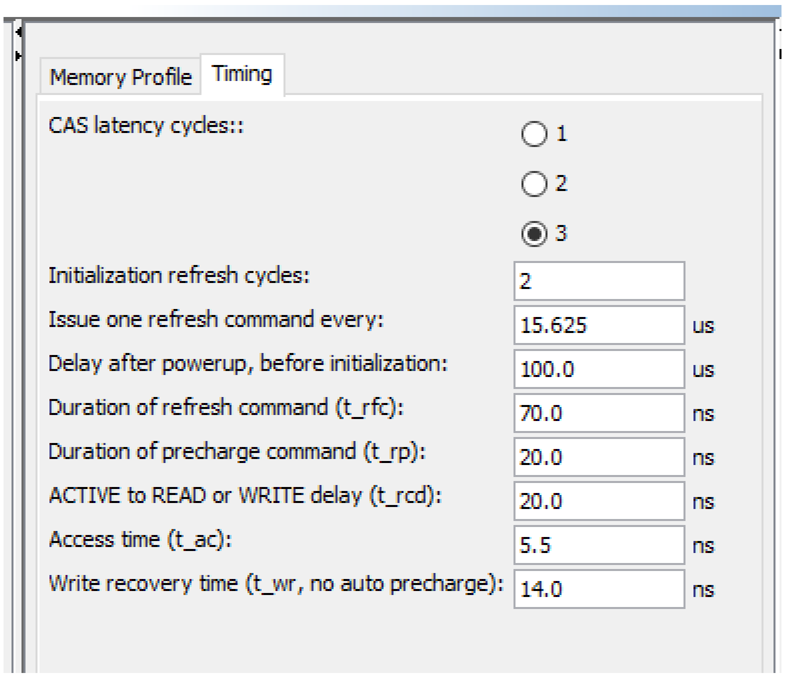
Address Width

Row:	13
Column:	10

Generic Memory model (simulation only)

☐ Include a functional memory model in the system testbench

Memory Size = 64 MBytes
33554432 x 16
512 MBits



CAS latency cycles:: ○ 1 ○ 2 [✓ ● 3]

Initialization refresh cycles:	2	
Issue one refresh command every:	15.625	us
Delay after powerup, before initialization:	100.0	us
Duration of refresh command (t_rfc):	70.0	ns
Duration of precharge command (t_rp):	20.0	ns
ACTIVE to READ or WRITE delay (t_rcd):	20.0	ns
Access time (t_ac):	5.5	ns
Write recovery time (t_wr, no auto precharge):	14.0	ns

8. DeepCL

While OpenCL is a vast improvement on using HDL, it is still far from going mainstream. Few programmers ever get much experience with programming in a fully parallel paradigm, so there is clearly a need for higher level tools to bridge the gap.

The neural network below is implemented in the DeepCL Python API.

DeepCL is a set of publically available wrappers for OpenCL with both a Python, LUA and C++ API using the OpenCL 1.1 standard. If built from source it should in theory run using any OpenCL driver. Several attempts at porting DeepCL to Windows 10 for use in this project failed, but it was successfully built on Linux Mint Debian Edition and used to compile the included source code to executable aoclx format. The network worked as suggested, but performance was not measured.

Altera has showcased several OpenCL examples that take advantage of the flexible memory arrangement in FPGAs, and their neural networks in particular take advantage of this, allowing kernels to be much more independent of the host than is possible on the GPU.

Unfortunately, Altera has not yet released the source code for their networks, and have recently removed the whitepaper describing their method from their web site.

So as of yet, while it may be optimized for the GPU, DeepCL remains a workable alternative.

The DeepCL Git repository can be found here:

https://github.com/hughperkins/DeepCL

9. Hardware Implementation of Algorithms

9.1. Support Vector Machines

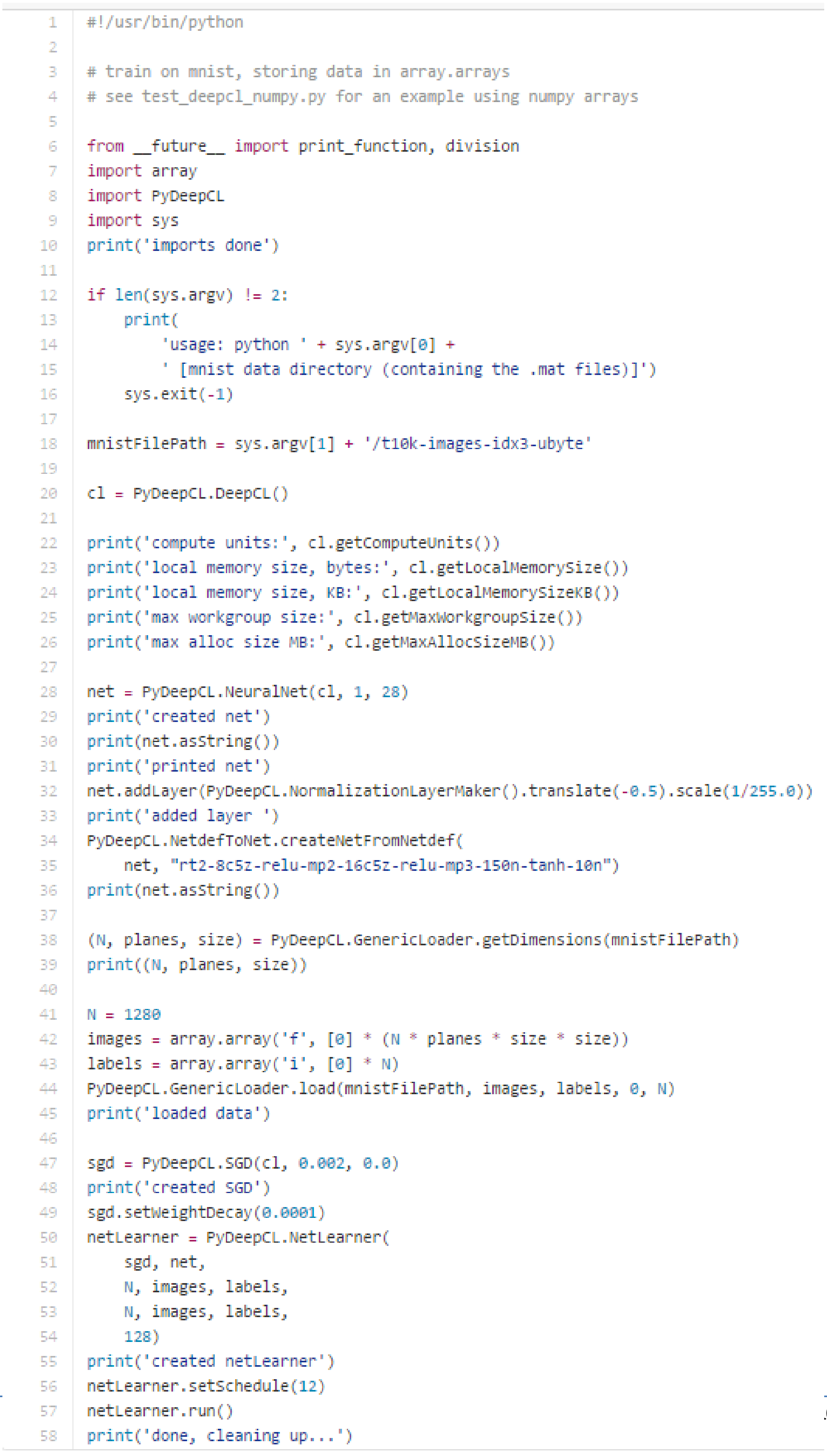
The following example is originally published as a case study by AMD on their developer network to demonstrate effective memory allocation on the GPU.

While that is not directly relevant for FPGA, it is an elegant piece of OpenCL C that was compiled without difficulty on the DE1 Cyclone V SoC.

9.2. Convolutional Neural Networks

The following example DeepCL Python script was successfully used to compile and run an aoclx executable on a Cyclone V DE1 SoC ARM/FPGA board:

```
float sum(float4 in) {
  return dot(in, (float4)(1.0f, 1.0f, 1.0f, 1.0f));
}

__kernel
void svm_kernel(__global float* data,
                 __const int pitch,
                 __const int nPoints,
                 __const int nDim,
                 __const int high,
                 __const int low,
                 __const float gamma,
                 __global float* high_kernel,
                 __global float* low_kernel
                 ) {
  __local float4 l_high[256];
  __local float4 l_low[256];

  int local_index = get_local_id(0);
  if(local_index < nDim) {
    l_high[local_index] = vload4(high, data + pitch * local_index);
    l_low[local_index] = vload4(low, data + pitch * local_index);
  }
  barrier(CLK_LOCAL_MEM_FENCE);
  int global_index = get_global_id(0);

  __global float* x_i = data + global_index * 4;
  float4 high_accumulator = 0;
  float4 low_accumulator = 0;
  for(int d = 0; d < nDim; d++) {
    float4 x_i_d = vload4(0, x_i);
    float4 x_high_d = l_high[d];
    float4 high_diff = x_i_d - x_high_d;
    high_accumulator += high_diff * high_diff;
    float4 x_low_d = l_low[d];
    float4 low_diff = x_i_d - x_low_d;
    low_accumulator += low_diff * low_diff;
    x_i += pitch;
  }

  float high_kernel = exp(-gamma * sum(high_accumulator));
  high_kernel[global_index] = high_kernel;

  float low_kernel = exp(-gamma * sum(low_accumulator));
  low_kernel[global_index] = low_kernel;

  global_index += get_global_size(0);
}
```



```python
#!/usr/bin/python

# train on mnist, storing data in array.arrays
# see test_deepcl_numpy.py for an example using numpy arrays

from __future__ import print_function, division
import array
import PyDeepCL
import sys
print('imports done')

if len(sys.argv) != 2:
    print(
        'usage: python ' + sys.argv[0] +
        ' [mnist data directory (containing the .mat files)]')
    sys.exit(-1)

mnistFilePath = sys.argv[1] + '/t10k-images-idx3-ubyte'

cl = PyDeepCL.DeepCL()

print('compute units:', cl.getComputeUnits())
print('local memory size, bytes:', cl.getLocalMemorySize())
print('local memory size, KB:', cl.getLocalMemorySizeKB())
print('max workgroup size:', cl.getMaxWorkgroupSize())
print('max alloc size MB:', cl.getMaxAllocSizeMB())

net = PyDeepCL.NeuralNet(cl, 1, 28)
print('created net')
print(net.asString())
print('printed net')
net.addLayer(PyDeepCL.NormalizationLayerMaker().translate(-0.5).scale(1/255.0))
print('added layer ')
PyDeepCL.NetdefToNet.createNetFromNetdef(
    net, "rt2-8c5z-relu-mp2-16c5z-relu-mp3-150n-tanh-10n")
print(net.asString())

(N, planes, size) = PyDeepCL.GenericLoader.getDimensions(mnistFilePath)
print((N, planes, size))

N = 1280
images = array.array('f', [0] * (N * planes * size * size))
labels = array.array('i', [0] * N)
PyDeepCL.GenericLoader.load(mnistFilePath, images, labels, 0, N)
print('loaded data')

sgd = PyDeepCL.SGD(cl, 0.002, 0.0)
print('created SGD')
sgd.setWeightDecay(0.0001)
netLearner = PyDeepCL.NetLearner(
    sgd, net,
    N, images, labels,
    N, images, labels,
    128)
print('created netLearner')
netLearner.setSchedule(12)
netLearner.run()
print('done, cleaning up...')
```

10. Bibliography

[1] A. Karlsen Monstad and D. C. Paulsen, "Intrusion Detection for Ship's Engine Room," Bachelor of Engineering, Computer Science Bachelor Thesis, Faculty for Technology and Maritime Subjects (TEKMAR), Buskerud and Vestfold University College, 2013.

[2] M. Sundhaug, "Intrusion Detection for Ship's Engine Room - Part 2," Bachelor of Engineering, Computer Science Bachelor Thesis, Faculty for Technology and Maritime Subjects (TEKMAR), Buskerud and Vestfold University College, 2015.

[3] D. Cimpean, J. Meire, V. Bouckaert, S. Vande Casteele, A. Pelle, and L. Hellebooge, "Analysis of Cyber Security Aspects in the Maritime Sector," 2011.

[4] R. L. Krutz, *Securing SCADA Systems*: Wiley, 2006.

[5] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage*, et al.*, "Comprehensive experimental analyses of automotive attack surfaces," presented at the Proceedings of the 20th USENIX conference on Security, San Francisco, CA, 2011.

[6] T. Hoppe, S. Kiltz, and J. Dittmann, "Security threats to automotive CAN networks—Practical examples and selected short-term countermeasures," *Reliability Engineering & System Safety,* vol. 96, pp. 11-25, 1// 2011.

[7] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xua*, et al.*, "Security and Privacy Vulnerabilities of In-Car Wireless Networks: A Tire Pressure Monitoring System Case Study," in *Proceedings of USENIX Security Symposium*, 2010.

[8] (May 10). *History of CAN technology*. Available: http://www.can-cia.org/can-knowledge/can/can-history/

[9] P.-S. Murvay and B. Groza, "Source Identification Using Signal Characteristics in Controller Area Networks," *Signal Processing Letters, IEEE,* vol. 21, pp. 395-399, 2014.

[10] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "CANAuth - A Simple, Backward Compatible Broadcast Authentication Protocol for CAN bus," *ECRYPT Workshop on Lightweight Cryptography 2011,* 2011.

[11] C. Szilagyi and P. Koopman, "Low cost multicast authentication via validity voting in time-triggered embedded control networks," presented at the Proceedings of the 5th Workshop on Embedded Systems Security, Scottsdale, Arizona, 2010.

[12] J. Hall, M. Barbeau, and E. Kranakis, "Enhancing intrusion detection in wireless networks using radio frequency fingerprinting," in *Communications, Internet, and Information Technology*, 2004, pp. 201-206.

[13] J. Hall, M. Barbeau, and E. Kranakis, "Radio frequency fingerprinting for intrusion detection in wireless networks," *IEEE Transactions on Defendable and Secure Computing,* 2005.

[14] N. Romero-Zurita, D. McLernon, M. Ghogho, and A. Swami, "PHY layer security based on protected zone and artificial noise," *Signal Processing Letters, IEEE,* vol. 20, pp. 487-490, 2013.

[15] R. M. Gerdes, M. Mina, S. F. Russell, and T. E. Daniels, "Physical-layer identification of wired Ethernet devices," *Information Forensics and Security, IEEE Transactions on,* vol. 7, pp. 1339-1353, 2012.

[16] D. Chen and D. Singh, "Invited paper: Using OpenCL to evaluate the efficiency of CPUS, GPUS and FPGAS for information filtering," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, 2012, pp. 5-12.

[17] B. Krzanich. (2016, 22.04). *Brian Krzanich: Our Strategy and The Future of Intel*. Available: https://newsroom.intel.com/editorials/brian-krzanich-our-strategy-and-the-future-of-intel/

[18] "Road vehicles -- Controller area network (CAN) " in *Part 1: Data link layer and physical signalling* vol. 11898, ed: ISO, 2003.

[19] "Road vehicles -- Controller area network (CAN) " in *Part 2: High-speed medium access unit* vol. 11898, ed: ISO, 2003, p. 21.

[20] Stefan-Xp, "CAN-Bus Topology (Electronic Two Wire Connection)," C.-B. E. Zweidrahtleitung.svg, Ed., ed. commons.wikimedia.org, 2008.

[21] Endres~commonswiki, "CAN-Bus-frame in base format without stuffbits," C.-B.-f. i. b. f. w. stuffbits.svg, Ed., ed. commons.wikimedia.org, 214.

[22] E. Kussul and T. Baidyk, "Improved method of handwritten digit recognition tested on MNIST database," *Image and Vision Computing,* vol. 22, pp. 971-981, 2004.

[23] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine,* vol. 29, pp. 141-142, 2012.

[24] Y. Mizukami, K. Tadamura, J. Warrell, P. Li, and S. Prince, "CUDA implementation of deformable pattern recognition and its application to MNIST handwritten digit database," in *Pattern Recognition (ICPR), 2010 20th International Conference on*, 2010, pp. 2001-2004.

[25] D. Keysers, "Comparison and combination of state-of-the-art techniques for handwritten character recognition: topping the mnist benchmark," *arXiv preprint arXiv:0710.2231,* 2007.

[26] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating deep convolutional neural networks using specialized hardware," *Microsoft Research Whitepaper,* vol. 2, 2015.

[27] D. Bianchi, G. C. Cardarilli, A. Del Re, A. Malatesta, and M. Re, "FPGA implementation of a general purpose HMM processor based on token passing algorithm," in *Circuit Theory and Design, 2005. Proceedings of the 2005 European Conference on*, 2005, pp. I/285-I/288 vol. 1.

[28] C. Gonzalez-Concejero, V. Rodellar, A. Álvarez-Marquina, D. Icaya, E. Martínez, and P. Gomez-Vilda, "A Soft-Core for Pattern Recognition," in *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, 2007, pp. 830-834.

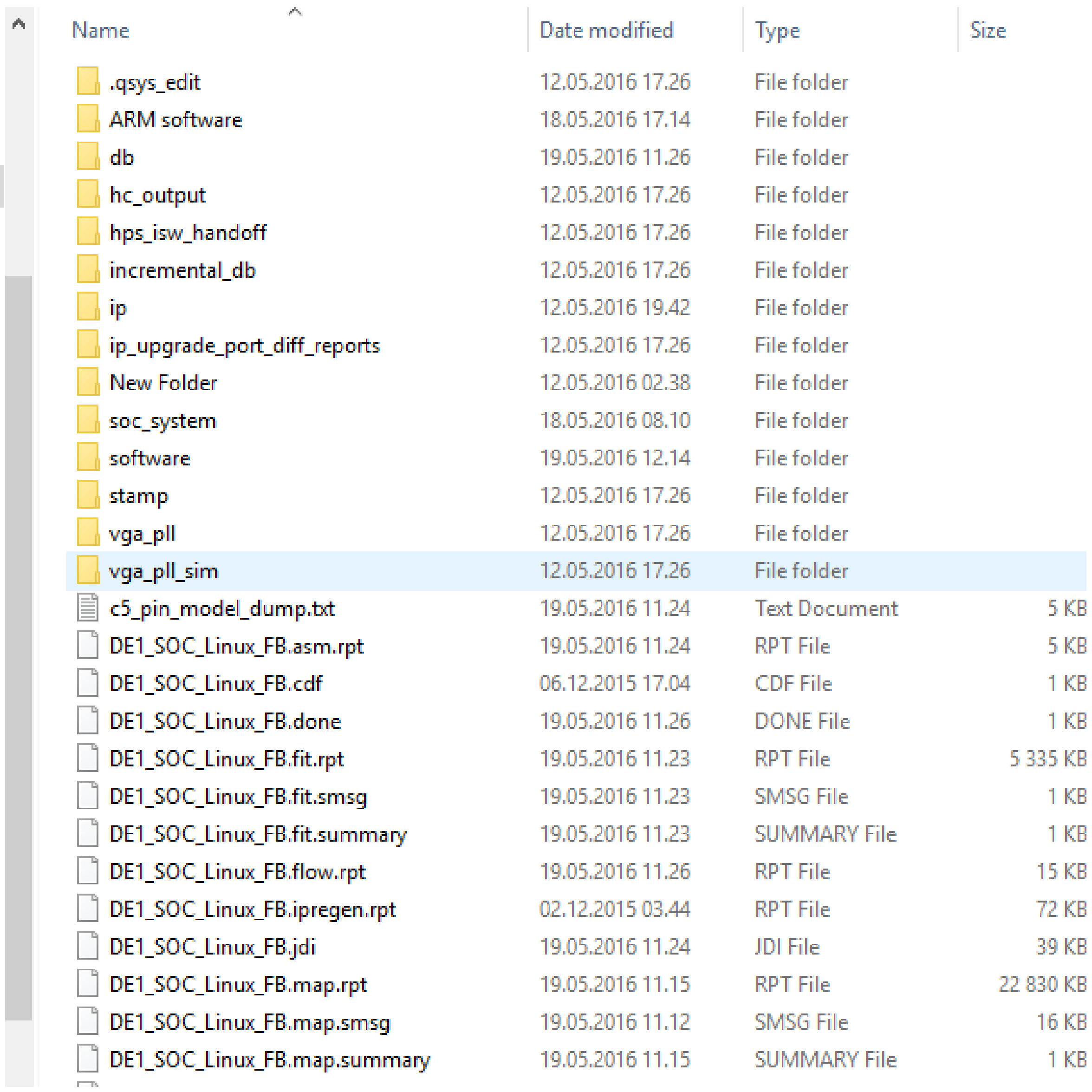
[29] V. Rodellar, C. Gonzalez-Concejero, P. Carretero, A. Alvarez-Marquina, and P. Gomez-Vilda, "A reusable HMM soft-core for isolated word recognition," in *Signals, Circuits and Systems, 2005. ISSCS 2005. International Symposium on*, 2005, pp. 303-306.

[30] X. Zhang, L. Jia, H. Shi, Z. Tang, and X. Wang, "The application of machine learning methods to intrusion detection," in *Engineering and Technology (S-CET), 2012 Spring Congress on*, 2012, pp. 1-4.

11. Term list

12. Attachments

A compressed project folder is included containing both hardware and software design files. The folder names containing software should be self-eplanatory.



Name	Date modified	Type	Size
.qsys_edit	12.05.2016 17.26	File folder	
ARM software	18.05.2016 17.14	File folder	
db	19.05.2016 11.26	File folder	
hc_output	12.05.2016 17.26	File folder	
hps_isw_handoff	12.05.2016 17.26	File folder	
incremental_db	12.05.2016 17.26	File folder	
ip	12.05.2016 19.42	File folder	
ip_upgrade_port_diff_reports	12.05.2016 17.26	File folder	
New Folder	12.05.2016 02.38	File folder	
soc_system	18.05.2016 08.10	File folder	
software	19.05.2016 12.14	File folder	
stamp	12.05.2016 17.26	File folder	
vga_pll	12.05.2016 17.26	File folder	
vga_pll_sim	12.05.2016 17.26	File folder	
c5_pin_model_dump.txt	19.05.2016 11.24	Text Document	5 KB
DE1_SOC_Linux_FB.asm.rpt	19.05.2016 11.24	RPT File	5 KB
DE1_SOC_Linux_FB.cdf	06.12.2015 17.04	CDF File	1 KB
DE1_SOC_Linux_FB.done	19.05.2016 11.26	DONE File	1 KB
DE1_SOC_Linux_FB.fit.rpt	19.05.2016 11.23	RPT File	5 335 KB
DE1_SOC_Linux_FB.fit.smsg	19.05.2016 11.23	SMSG File	1 KB
DE1_SOC_Linux_FB.fit.summary	19.05.2016 11.23	SUMMARY File	1 KB
DE1_SOC_Linux_FB.flow.rpt	19.05.2016 11.26	RPT File	15 KB
DE1_SOC_Linux_FB.ipregen.rpt	02.12.2015 03.44	RPT File	72 KB
DE1_SOC_Linux_FB.jdi	19.05.2016 11.24	JDI File	39 KB
DE1_SOC_Linux_FB.map.rpt	19.05.2016 11.15	RPT File	22 830 KB
DE1_SOC_Linux_FB.map.smsg	19.05.2016 11.12	SMSG File	16 KB
DE1_SOC_Linux_FB.map.summary	19.05.2016 11.15	SUMMARY File	1 KB