Analyzing Bus Load Data Using an FPGA and a Microcontroller

Marcel Dombrowski*, Kenneth B. Kent†, Yves Losier‡, Adam Wilson§ and Rainer Herpers¶

University of New Brunswick
Fredericton, NB, Canada
∗Faculty of Computer Science, Email: marcel.dombrowski@unb.ca
†Faculty of Computer Science, Email: ken@unb.ca
‡Department of Mechanical Engineering, Email: ylosier@unb.ca
§Institute of Biomedical Engineering, Email: awilson@unb.ca
¶Bonn-Rhein-Sieg University of Applied Sciences, Department of Computer Science
Sankt Augustin, Germany, Email: rainer.herpers@h-brs.de

Abstract—In this paper we present the design, implementation, and testing of an evaluation tool for the ongoing development of the Prosthetic Device Communication Protocol (PDCP) which is an open protocol and is featured in the University of New Brunswick’s most recent prosthetic limb research project, the UNB Hand System. This prosthetic device utilizes the CAN bus hardware with the PDCP for passing command and data messages between modules within the prosthetic limb system. The PDCP allows abstraction of the underlying bus system and allows different network topologies depending on particular needs. To be able to analyze communication in the CAN layers as well as in the PDCP layer we present our own solutions utilizing an FPGA for CAN bus bandwidth load monitoring and a microcontroller for PDCP monitoring and analysis.

Keywords—analysis; bus load; can bus; data logging; fpga; microcontroller; monitoring; pdcp; prosthetic limb

I. INTRODUCTION

The University of New Brunswick’s (UNB) Institute of Biomedical Engineering (IBME) is developing together with different research partners a prosthetic hand system, termed the UNB Hand System. For the design of the hand not only the overall cost was relevant, but also the controllability and design modularity. Thus the UNB Hand is capable of recognizing different grasp patterns, with an emphasis on the movements that are controllable by most amputees. As every user has their own requirements the UNB Hand is also modular to fit different sizes [1]. An image of the hand can be seen in Figure 1.

Inside the hand different sensors for measuring the pressure and motors for moving the thumb and fingers are working. These modules are connected via a CAN bus network (ISO OSI layer 1 and 2). On top of this bus system the Prosthetic Device Communication Protocol (PDCP) has been built [2]. It allows an abstraction of the underlying bus system to give engineers more flexibility when designing new components for the hand. The PDCP itself acts on ISO OSI layer 3 and is used for information exchange.

During the implementation stage of additional modules for our prosthetic device, tools are needed that analyze the data on the bus to ensure that messages are transmitted and received properly before these modules are used in a live system. Furthermore, in a live system our engineers need to be able to gather information on how the prosthetic device is used by different patients. Thus we need to be able to monitor and analyze the communication on the bus online and offline.

Section II will briefly introduce the CAN bus as well as the PDCP, Section III will show related work in the field of CAN bus monitoring using FPGA and microcontroller devices, Section IV will introduce our design, which will further be explained in Section V. This paper will conclude in Section VI with future work and a short summary of this paper.

II. BACKGROUND

A. CAN Bus

The Controller Area Network (CAN) bus started as an internal project at Bosch in 1983 and was officially introduced in 1986 [3]. Originally being designed for usage in the automotive industry the CAN bus was soon also used in other areas. The CAN bus operates on the Physical and Data Link layer of the ISO OSI Reference Model. The physical media for transmission was intentionally left out to give the engineers more flexibility during the creation of the system.

For collision detection the CAN bus utilizes the CSMA/CD protocol [4], which is also used in the IEEE 802.3 standard [5]. The CAN bus protocol is a message
based protocol and it specifies four different frames: data frame, remote frame, error frame, and overload frame. Each data frame either has an 11 bit (SID) or a 29 bit (EID) identifier and can support the transmission of up to 8 bytes of data. For error detection in the data field of a data frame it uses the 15 bit CRC algorithm.

B. PDCP

The PDCP was developed by the IBME at UNB and is an open communication protocol which serves as an abstraction of the underlying bus system. It enables point-to-point connections of different nodes in the bus by setting up data channel links. If one node wants to directly communicate with another node it creates a data channel link which gets its ID assigned by the master controller, termed the Bus Arbitrator (BA). After connecting a device to the bus it needs to register itself with the BA by sending its vendor ID, product ID, and serial number to the BA. According to this information the BA assigns unique node IDs for the nodes. The node ID is the lower 8 bits of the SID field of a data frame. The top two bits are reserved for the message priority, ranging from 10\textsubscript{2} with the lowest priority to 00\textsubscript{2} with the highest priority. 11\textsubscript{2} is reserved for Bind Device Requests. The third most significant bit of the SID is reserved for the message mode, with 1\textsubscript{2} being reserved for the BA and 0\textsubscript{2} for all other devices. Figure 2 shows this.

![Figure 2: Standard Identifier Field Subsections.](image)

III. RELATED WORK

Much research has been conducted in the field of bus monitoring, and not only limited to the CAN bus. Bochem et al showed in [6] an approach for monitoring the CAN bus using an Altera DE2 development board with two different CAN controllers connected to it. This design has been done in Verilog using different IP cores.

Kashif et al showed in [7] an implementation of a CAN bus analyzer using Verilog on Spartan 3E and Vertex 2 Pro FPGAs. They implemented an 8051 microcontroller with external RAM and two Philips SJA1000 standalone CAN controllers. Furthermore they wanted to be able to inject further data on the bus with their solution.

Various other methods have been used to monitor CAN activity using PCs [8], microcontrollers [8] [9], and CPLDs [9]. These methods only allowed monitoring of bus activity as opposed to bus bandwidth load.

IV. DESIGN

For our design we chose to separate the functionality onto two hardware devices: as we wanted to be able to monitor the CAN bus bandwidth load we use an FPGA device for this purpose. The FPGA is able to transmit messages to the second hardware device, which is a microcontroller. The FPGA is sending the bus load data, as well as the individual module’s bus load to the microcontroller. The microcontroller receives these statistics and generates output files in a human readable format. Furthermore the microcontroller is directly connected to the CAN bus as well and it receives messages and interprets them according to the PDCP. Based on the interpreted PDCP messages and on the a priori knowledge of the devices on the bus it also generates statistics, e.g. which module communicates with another module.

![Figure 3: The Altera DE0-nano Board [10].](image)

For the FPGA we chose to use the Altera DE0-nano board with an LCD connected to its GPIO port. The FPGA board can be seen in Figure 3. The microcontroller is a Microchip dsPIC33E USB Starter Kit (Figure 4(a)) which is attached to the Microchip Multimedia Expansion Board (MEB) (Figure 4(b)). The MEB features an LCD with a touchscreen, which is useful for debugging the application, as well as a microSD card connector, which is used for writing the statistics, so that these can later on be viewed offline on a PC. Furthermore it includes an I/O expansion header and due to the dsPIC33E microcontroller’s reprogrammable pins most of the pins on this header can be reprogrammed for e.g. CAN RX, CAN TX, or UART RX.

The project itself, for both parts, has been designed top-down, meaning that every problem has been split into smaller problems for as often as possible. Once this was done the implementation was done bottom-up, starting from the smallest problems in a hierarchical way to the topmost problem. Nevertheless before being able to implement using this approach we implemented drivers.
V. IMPLEMENTATION
A. Overview and Architecture
The FPGA part of our project has been divided into two main parts: the monitor and the output module. The monitor module is responsible for receiving CAN messages and interpreting them according to their identifier. At every sampling point, that can be set before synthesizing the design, it transmits the load data for every module that is monitored as well as the load data for the overall bus load to the output module. The output module is responsible for the conversion of the load data values into strings for displaying on the LCD as well as for the wrapping and sending of the load data values into UART frames which are then sent over the bus. Figure 5 shows the architecture of our FPGA design.

The monitor module consists of the CAN driver which receives messages from the CAN bus and interprets them according to the Bosch CAN specification [3]. After a full message has been received it is passed on to the load calculation module. In there each CAN message is checked for their identifier, which can be either the 11 bit SID or the 29 bit EID. According to the identifier the length of the received CAN message will be added to the corresponding module. At every sampling point the load values that have been calculated in that sampling cycle will be passed on to the output module. The output module then does the calculation of the actual load of every module as well as of the overall bus load and sends a formatted string with the relative load data compared to the maximum possible bus load to the LCD driver and also wraps the IDs and corresponding load values into UART frames and sends them to the UART driver. Furthermore our design is capable of meeting hard real-time requirements, as this is needed in a CAN bus setup.

The second part of our project is implemented on the microcontroller side. For this the microcontroller needs to implement low-level drivers for the CAN bus in order to receive and transmit messages, as well as a UART driver for the reception of messages via UART and a SD card driver in order to write data to a microSD card. On top of this high-level functionality is needed, which takes the received CAN messages, interprets them, and generates statistics based on these messages. The microcontroller architecture can be seen in Figure 6.

The microcontroller needs to implement the CAN bus specification in order to receive and transmit messages from and to the bus. Once a new message arrives it is interpreted by our PDCP interpretation module. With the interpreted messages the microcontroller is then able to generate statistics based on the PDCP commands. Furthermore the microcontroller is also capable of receiving messages from the FPGA via UART. The generated statistics and the received load data from the FPGA is then written in a human readable format to the microSD card. As the microcontroller is also capable of writing to the CAN bus it needs to follow the PDCP as well. Thus the microcontroller needs to be connected to a PDCP network after the BA has been powered on else it cannot bind itself to the bus. An image of the microcontroller program can be seen in Figure 7.
B. Application

As the PDCP has been designed to be an open protocol other applications could also implement it. For instance in the automotive industry the PDCP could be used to make it easier for engineers to establish communication between different nodes. Nodes could create communication channels with every node that they need to talk to and then communicate only using these channel IDs. With this the receiving node directly knows which node has sent the message. Using our tools the engineer can directly connect these devices to the CAN bus and specify the configuration parameters as well as the list of the devices on the bus. The engineer would then be able to see online what is happening on the bus in real-time and if the devices are known also see which devices are communicating. Furthermore different messages can also be specified, so that if a node sends a specific bit combination in the data field it is displayed as e.g. proximity_sensor1 to braking_system: obstacle in 20m. As every action can be logged it is also possible to have different runs either in simulation or in real-life usage and then review the communication as CAN messages as well as interpreted messages offline on a PC for tuning system or node parameters.

VI. Future Work and Conclusions

Our project can be further extended in terms of configurability, so that it is possible for the engineer to specify the CAN bus bitrate in the configuration file, as well as to set the system clock differently, as using different bus systems might require a system clock higher than 32 MHz. Furthermore also having an option to specify which bus system is used might be helpful, so that the project does not need to be rebuilt every time a different bus system is used.

As most bus analyzers are only capable of displaying raw message data we introduced our analyzer which can be configured depending on the nodes that are connected to the system as well as on the messages that are sent on this network. Our analyzer displays messages during runtime on a built-in LCD and saves the messages received in the network according to pre-defined filters on a microSD card, so that the logged data can later be viewed offline on a PC. Our FPGA solution can be used if only the bus load of different modules is of interest and our microcontroller solution if interpretation of the messages is needed.

ACKNOWLEDGMENT

The authors would like to thank the Institute of Biomedical Engineering (University of New Brunswick), CMC Microsystems as the tool provider, NSERC, and the Atlantic Canada Opportunities Agency for their contribution to the UNB Hand project.

REFERENCES