Proceedings of The National Conference
On Undergraduate Research (NCUR) 2015
Eastern Washington University, Cheney, WA
April 16-18, 2015

Architecture For Plug-And-Play Modular Technology

Michele Pancani
Electrical Engineering and Computer Science (EECS) Department
Milwaukee School of Engineering
1025 N. Broadway
Milwaukee, Wisconsin 53202 USA

Faculty Advisor: Dr. Russell Meier

Abstract

Electronic and computing devices are composed of multiple components performing different tasks. Often, the need to substitute or upgrade a single component leads to the replacement of the entire device. In order to reduce electronic waste, costs and to allow more freedom of customization, the goal of this project was to design and test a peripheral device protocol that allows components, such as sensors and actuators, to operate as plug-and-play interchangeable modules. These modules are connected to the system's control microprocessor through a standard host connector, which is similar to the mikroBUS standard developed by the company MikroElektronika, but with the additional freedom of allowing modules to be interchanged while the program is running, regardless of whether the module is classified as an input or output device. The idea, indeed, is to allow the user to dynamically add a new peripheral device to any available host connector, remove devices while the system is operational, or substitute a peripheral like a fingerprint scanner or a LED flashlight after the system has been programmed and is operational. Furthermore, the technology differs from the Universal Serial Bus (USB) architecture because of the hardware setup and the communication protocol. The project began with the design and development of a first simplified architecture followed by a discussion of a more general and complex interface for plug-and-play interchangeable modules. The peripheral device protocol architecture was described in the VHDL hardware description language and synthesized for implementation in the field-programmable gate array (FPGA) chip of an Altera DE0 development board. Next, testing with an Altera Nios II soft-core processor, instantiated on the same FPGA chip and programmed in C, was completed with simple input/output peripherals implemented in VHDL serving as the test plug-and-play modules. Afterwards, simulation and experimental results were analyzed to ensure that the expectations were met. This concept of interchangeable modules can be applied to connect simple sensors and actuators to a CPU, but also has a wide range of uses in more advanced devices, such as medical equipment or smartphones.

Keywords: Modular Technology, Plug-and-Play, I/O Peripheral Device

1. Introduction

Plug-and-Play technology has been in use for some time in personal computers [1], allowing the use of Universal Serial Bus (USB) devices to grow exponentially over the last few years. However, plug-and-play technology is not currently widespread at the embedded system level and, even though a few different solutions have been proposed to solve different aspects of the problem, there are no specific standards. The biggest issue is that a USB host interface is not practical on an embedded platform for several reasons. Since "*the majority of embedded self-developed USB system software, the underlying drivers, the boundaries for operating systems and applications are often not clear*" [3], in addition to the absence of a high-level operating system, a USB host embedded system cannot be a full application because it would most likely be structured in a way that only supports a specific USB device, losing the universal USB characteristics and advantages [3].

Therefore, the purpose of this project is to provide a simple experimental architecture for embedded systems. The architecture is implemented as a peripheral device and allows the user to dynamically connect and interchange modular sensors and actuators to the system's control microprocessor, after this has been programmed and is operational, through any available standard host connector. In particular, the project is the author's interpretation of the idea of modular technology, as presented on the website of Phonebloks [5] at the end of 2013.

1.1 System Requirements

In order to be able to interface modular peripheral devices like sensors and actuators, which will be referred to as 'modules', to the system, said devices need to be placed, along with a driver chip, on their own board. Of course, I/O pins of the module must match the footprint of the standard host connector. Furthermore, each module must also support the proposed communication protocol, either on the driver chip or on an additional chip, and must have an identification number (ID) unique to the typology of the device (LEDs, DC motor, pushbuttons, digital camera etc...) in order to be properly identified by the system.

The standard host connector is similar to the mikroBUS standard host connector [4] developed by the company MikroElektronika. The mikroBUS simply consists of two parallel rows of pin headers with a defined pinout that includes SPI, I^2C, PWM and power pins [4]. However, due to the hard wired connections, each slot can only support the particular module it has been programmed to. On the other side, the proposed host connector, which has less pins because it interacts with the module through a custom protocol, has the advantage of interchangeability and makes the system more dynamic.

1.2 System Specifications

The architecture, referred to as 'Plug-and-Play modules handler', was implemented in the VHDL hardware description language and was synthesized for testing on the field-programmable gate array (FPGA) chip of an Altera DE0 development board. For the purpose of the project, three host connectors were instantiated and wired to the general purpose pins of the board's pin header GPIO0. Then, a testing module to drive the DE0 board's green LEDs was implemented in VHDL, synthesized on the same FPGA chip and wired to the general purpose pins of the board's pin header GPIO1. Jumper cables were used to make the connections between the testing module and any available host connector on the other pin header of the board. Finally, the 'f' version of a 50MHz 32-bit soft-core Altera Nios II processor was instantiated on the same FPGA chip and programmed in the C programming language in order to control and test the Plug-and-Play modules handler. Interfacing between the processor and the hardware peripheral was done using the Avalon Switch Fabric, which is built automatically by the Altera SOPC builder Qsys used to generate the whole system, shown in Figure 1.

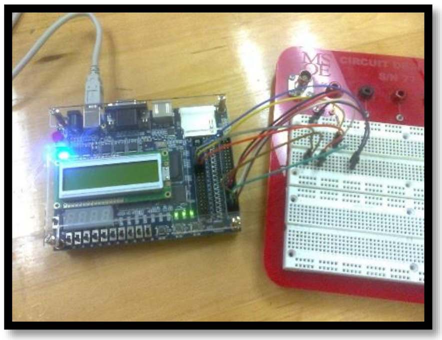


Figure 1. Altera DE0 board used as a testing environment for the Plug-and-Play modules handler peripheral

2. Methodology

The main design was broken down into three main components, as shown in the block schematic of Figure 2, which will be described below in more details.

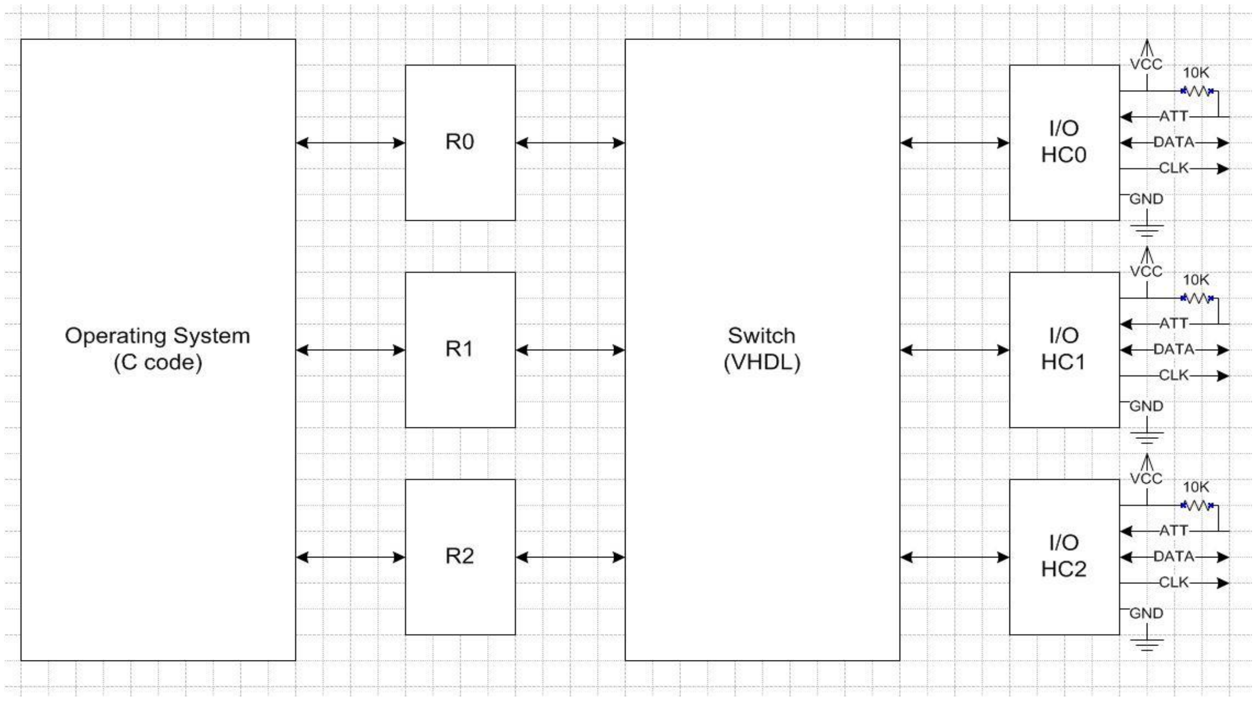


Figure 2. Schematic of the proposed architecture for plug-and-play technology

As previously mentioned, each modular peripheral device needs to be driven by its own driver chip, which includes a set of data and control registers, and needs to have an identification number (ID), unique to the typology of the device. In order to control the modular peripheral device, a set of general purpose registers, R on the left hand side of the schematic shown in Figure 2, is provided. Each one of these registers will need to be programmed to support a specific typology of device, as explained in the *General purpose registers* sub-section, so that, when a module with a matching ID is connected to the system through any available host connector, I/O HC on the right hand side of the schematic shown in Figure 2, the switch component in between will establish a wired connection between such host connector and the register programmed to support that particular module.

After a connection with the module has been established in the way described below, the general purpose register will be synchronized to the register of the driver chip of the associated module. Consequently, writing a value to a general purpose register of the Plug-and-Play modules handler will cause the value at the respective address offset of the register of the associated module to change and vice versa. The synchronization between the modules handler and the modules is event driven, meaning that data is exchanged only when a value on either side changes.

When a new module is connected to the system through any available host connector HC, the active-low attention line ATT goes low to indicate the presence of a new device. At this point, the host connector sends an identification request to the module, as discussed in more details in the *Communication protocol* sub-section, and the module will have to respond by sending back its identification number (ID). Once the module has been identified, the switch component will look for an available register R that supports such typology of device and, if found, will established a wired connection and mark that register as no longer available.

2.1 General Purpose Register

Each set of general purpose registers, three for the purpose of this project, consists in a set of sixteen 32-bit locations. Currently, location 0 corresponds to the ID field, which is the field in which the user needs to write the identification number of a module in order to program the register to support such module. Therefore, from a programming point of view, nothing changes from the classical model of embedded system because the microprocessor knows what general purpose registers to write to or read from in order to interact with a particular module. Indeed the switch component is going to take care of the actual physical location the module is connected to and map the module registers to the pre-programmed general purpose registers. Furthermore, in order to handle the synchronization process that takes place whenever the user writes a new value into a register, a signal goes high to notify the host connector that a specific location has been updated. The signal remains high until a second hand shake signal pulses, meaning that the host connector has processed the new value and synchronized the register of the module.

2.2 Switch Component

The switch component consists of a set of state machines, one for each host connector, that sequentially scan the general purpose registers in order to find one that supports the module connected to the respective host connector. If successful, the state machines establish a wired connection between the two components. More specifically, once a connection request is received from a host connector, the respective state machine, shown in Figure 3 and stuck in the IDLE state until then, starts iterating through a number of states that equal the number of general purpose registers. Assuming that each register has been previously programmed to support one typology of module, each state with the prefix CHECK_R reads the ID field of the respective register to see if it matches the module's ID and checks whether the register is available. If not, the machine moves to the next CHECK_R state, otherwise it locks itself into a CONN_R state, marks the register as not available (for the other state machines running in parallel) and establishes a hardware connection between the respective register (which depends on the number of the CONN_R state) and the respective host connector (which depends on the state machine). Such connection is held until the host connector suspends the connection request. Finally, after a host connector has been wired to a register, a signal goes high to notify the host connector that the connection was accepted and established.

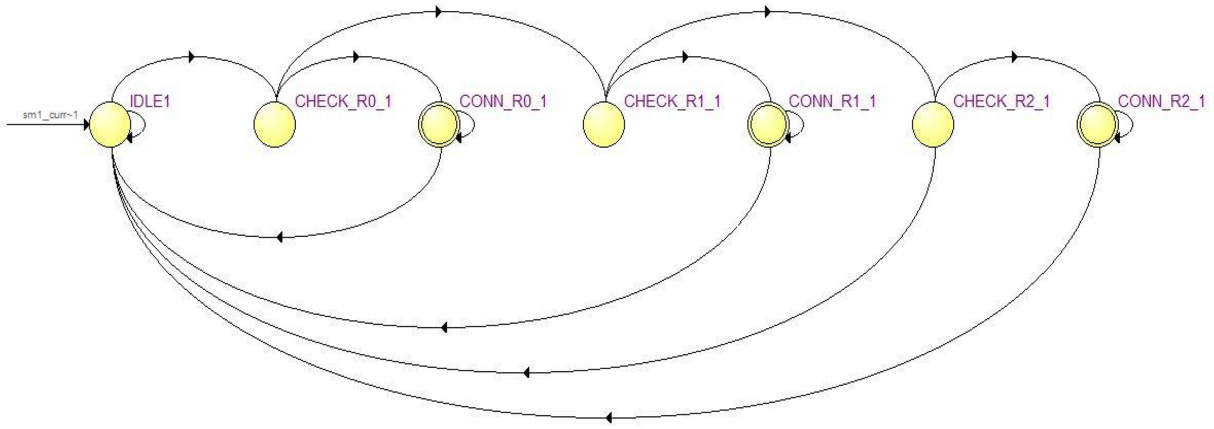


Figure 3. HC0 state machine implemented in the switch component

2.3 Host Connector

The host connector is the component responsible for detecting and identifying new modules, sending connection requests to the switch component and handling the synchronization between the general purpose registers and the modules. Communication with the module is handled serially through the data pins described in the *Pinout* sub-section, whereas data from/to the registers is read/written in parallel. As previously mentioned, when a new module is connected, the attention line ATT goes low and a bit counter and a packet counter in the host connector begin counting to keep track of, respectively, the bit and the packet being transmitted or received. After a module has been detected, the host connector sends an identification request to the module, which will send back its identification number, and forwards the module's ID to the switch component. If the switch component accepts the connection, the host connector begins the synchronization between the module and the register and resets a signal, currently wired to the DE0 board green LEDs, that notifies the user that the connection has been accepted and is no longer pending.

2.3.1 pinout

The standard host connector is wired to the following six I/O pins.

1. VCC: Pin used to provide power to the module (3.3V or 5V)
2. ATT: Input active-low attention line used by the system to detect a module
3. DATA in: Input serial pin to synchronously receive data from the module
4. DATA out: Output serial pin to synchronously transmit data to the module
5. CLK: 5MHz output clock signal to the module
6. GND: Ground pin

The attention line ATT requires an external 10 kΩ pull-up resistor because active-low. Consequently, the ATT line of the module needs to be open drain in order to be able to ground said signal.

2.4 Communication Protocol

The custom made communication protocol between a host connector and a module consists in a stream of data, transmitted least-significant bit (LSB) first, that is divided in packets of 36 bits and described as follows:

- Packet 0: Host connector sends the identification request
- Packet 1: Module responds with the identification number (ID)

After a connection has been established, the communication stream continues as follows, otherwise the alternative identification request is sent until the connection establishes:

- Packet 2: Both the host connector and the module send 0x00 if no information is to be transmitted or 0xCC if the following packet will contain actual valid data
- Packet 3: 32-bit data followed by a 4-bit destination register offset

Packet 0 and 1 are only transmitted during the connection phase, after which the sequence of packet 2 followed by packet 3 repeats until the module is disconnected.

The custom made communication protocol is structured in a way that is somewhat similar to the USB protocol [8] and the interfacing protocol of a SONY Play Station II controller [7].

3. Data

The Plug-and-Play modules handler was first simulated using the Altera Quartus II simulator and, afterwards, it was tested on an Altera DE0 development board with a Nios II soft-core processor. A testing module was implemented in VHDL to drive the board's general purpose LEDs. Furthermore, correct state transitions of the switch component's state machines were verified with the Altera Quartus II RTL viewer.

3.1 Top-Level Schematic

Figure 4 shows the top-level schematic of the Plug-and-Play modules handler. Besides the components shown in Figure 2, the actual design also includes an input and a output header to interface the peripheral device with the Avalon Switch Fabric and a clock divider to slow down the frequency of the system clock from 50MHz to 5MHz because 50MHz experimentally turned out to be too fast to handle the serial communication between the host connector and the modules. Besides, no peripheral device on a module should need a clock signal in the Mega-Hertz range and, if that is the case, the module will need to have its own on-board clock generator.

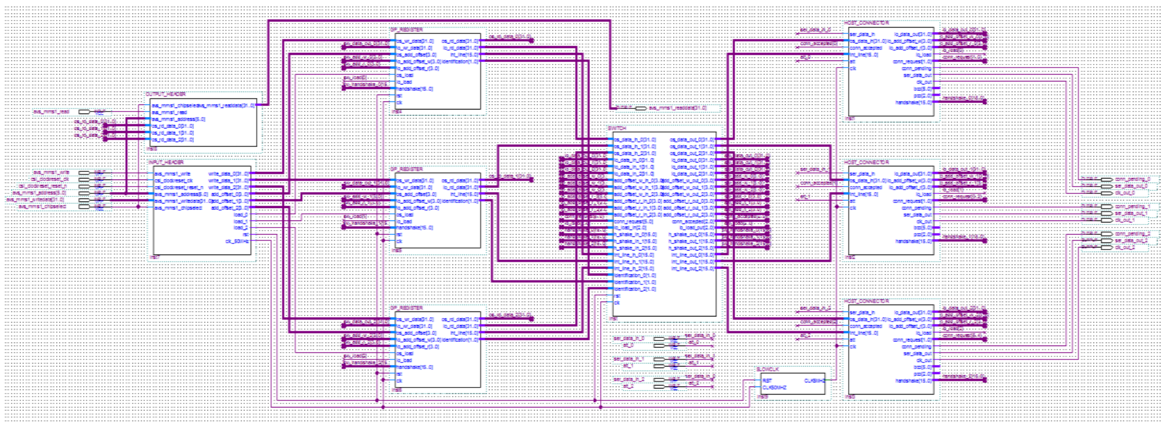


Figure 4. Zoom out of the Top-level schematic of the plug-and-play modules handler

3.2 Simulation Results

Multiple simulations were done over the course of the project, including at the unit level. The result shown in Figure 5 is a simulation of the top level entity shown in Figure 4 that tests three cases in particular, described below.

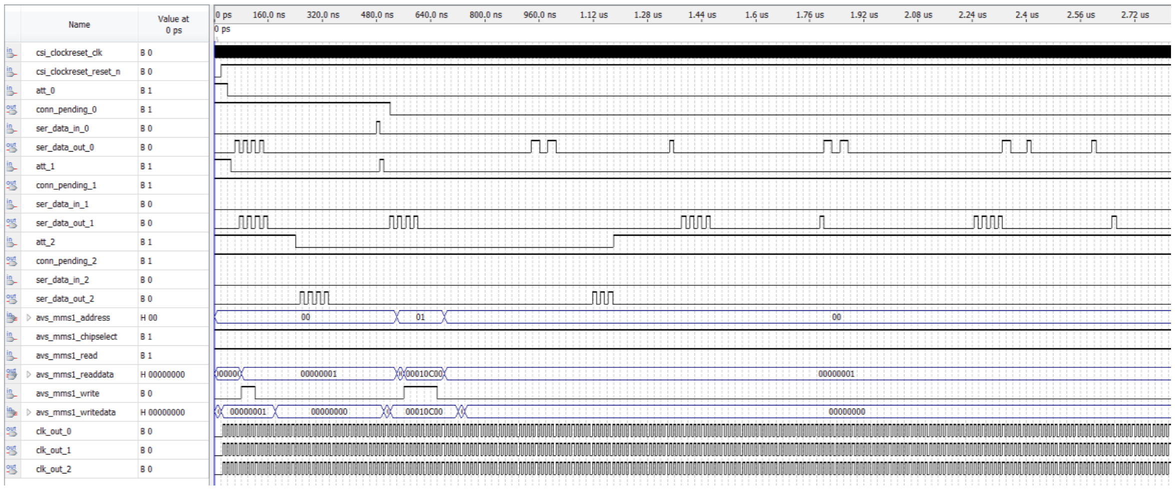


Figure 5. Simulation result of the plug-and-play modules handler

3.2.1 *simulation case 0*

For the first simulation case, register 0 was programmed to support a device of typology (ID) 0x1 and an arbitrary value of 0x10C00 was written at arbitrary offset 0x1. The result shows that, after the attention line (att_0) of host connector 0 goes low, connector 0 (ser_data_out_0) sends out the identification request 0xAA, the module (ser_data_in_0) responds with the ID 0x1 and, shortly after that, the connection pending 0 line (conn_pending_0) goes low, meaning that the connection was accepted.

Afterwards, the synchronization process begins and the host connector (ser_data_out_0) sends out the string 0xCC to indicate that valid data is following. The first package of valid data is an echo of the typology of the module, 0x1, whereas the second package of valid data, proceeded by 0xCC again, is the pattern 0x10C00, followed right after by the destination offset of register 0, 0x1.

3.2.2 *simulation case 1*

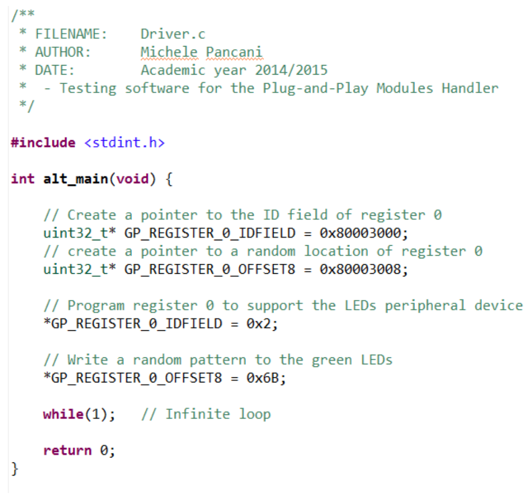
For the second simulation case, the same setup of the first simulation case could be assumed. Once again, register 0 was programmed to support a device of typology (ID) 0x1 and an arbitrary value of 0x10C00 was written at arbitrary offset 0x1. However, in this simulation case, the attention line (att_1) of host connector 1 goes low a few clock cycles after the attention line of host connector 0. Therefore, since module 0 successfully connects to register 0 earlier, the connection requested by host connector 1 will not be accepted because the register is no longer available. The result is that host connector 1 (ser_data_out_1) will keep sending out the alternate identification request 0x8000000AA for the rest of the simulation.

3.2.3 *simulation case 2*

The last simulation case shows a peripheral (ser_data_in_2) that does not respond. Indeed, since no connection could be made, host connector 2 send out the alternate identification request 0x8000000AA until the module is disconnected. The simulation also shows how the host connector interrupts all communications as soon as the attention line goes back high, meaning that the module was unplugged.

3.3 Hardware Implementation

The final part of the project consisted in the actual hardware implementation. In order to do it, the system described in the *System specifications* section was implemented and instantiated on the DE0 board. Then, the testing software shown in Figure 6 was run on the Nios II processor. The signals between the host connectors and the modules were intercepted and monitored on an oscilloscope.



```
/**
 * FILENAME:    Driver.c
 * AUTHOR:      Michele Pancani
 * DATE:        Academic year 2014/2015
 *   - Testing software for the Plug-and-Play Modules Handler
 */

#include <stdint.h>

int alt_main(void) {

    // Create a pointer to the ID field of register 0
    uint32_t* GP_REGISTER_0_IDFIELD = 0x80003000;
    // create a pointer to a random location of register 0
    uint32_t* GP_REGISTER_0_OFFSET8 = 0x80003008;

    // Program register 0 to support the LEDs peripheral device
    *GP_REGISTER_0_IDFIELD = 0x2;

    // Write a random pattern to the green LEDs
    *GP_REGISTER_0_OFFSET8 = 0x6B;

    while(1);   // Infinite loop

    return 0;
}
```

Figure 6. Testing C code used to program general purpose register 0

4. Conclusion

The project was meant to propose an architecture for embedded systems, implemented as a hardware peripheral device, to interface a microprocessor to modular interchangeable plug-and-play sensors and actuators through a standard host connector. The architecture, referred to as Plug-and-Play modules handler, was tested and simulation and experimental results were analyzed to ensure that the expectations were met.

Although the project was simple, the concept of plug-and-play technology at the embedded system level was proven to be efficient and of great potential. Indeed, future implementations of the project may include a much greater number of general purpose registers, in order to support a wider variety and number of potential modules, a greater number of host connectors, to allow more modules to operate simultaneously, and may also include a revision of the communication protocol and of the synchronization system. A more efficient communication protocol and synchronization system are meant to better handle particular cases, more complex modular devices and to make the system more reliable.

Eventually, potential applications may include a wide variety of devices, including, but not limited to, robotic platforms, medical equipment and smartphones, as also proven by the implementation of the Phonebloks idea [5] by Motorola with its Project Ara [6]. Indeed, the realization of plug-and-play devices would be greatly beneficial to reduce electronic waste, costs and to allow more freedom of customization because upgrading single components when needed prevents the replacement of the entire device, which could be very expensive when thinking about hi-tech machines used in the professional world.

5. Acknowledgments

The author wishes to express his appreciation to:

- Faculty advisor Dr. Russ Meier for the support and the precious advises
- Milwaukee School of Engineering (MSOE) Undergraduate Research director Dr. Cory Prust and the MSOE Undergraduate Research committee for the opportunity given
- Milwaukee School of Engineering EECS Tech Support for providing all of the equipment needed, including, but not limited to, wire kits, breadboards, a multi-meter and an oscilloscope to monitor the communication signals between the Plug-and-Play modules handler and the modules themselves

6. References

1. Pitzek, S.; Elmenreich, W., "Plug-and-play: bridging the semantic gap between application and transducers," *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on* , vol.1, no., pp.8 pp.,806, 19-22 Sept. 2005
2. Ylianttila, M.; Harjula, E.; Koskela, T.; Kassinen, O.; Riekki, J., "Mobile Plug-and-Play Architecture for Collaborative Hybrid Peer-to-Peer Applications," *Congress on Services Part II, 2008. SERVICES-2. IEEE* , vol., no., pp.81,87, 23-26 Sept. 2008
3. Gaohua Liao; Quanguo Lu; Weizhong Zhang, "Design of Reusable Software for USB Host Driver in Embedded System," Computing, Control and Industrial Engineering (CCIE), 2010 International Conference on , vol.1, no., pp.312,315, 5-6 June 2010
4. MikroElektronkia, “mikroBUS, a rocking new pinout standard!”, mikroe.com, http://www.mikroe.com/mikrobus/, Last access: May 22, 2015
5. Phonebloks, “Phoneblocks, a phone worth keeping”, phonebloks.com, https://phonebloks.com/en
6. Motorola, projectara.com, http://www.projectara.com/, Last access: May 22, 2015
7. Curious Inventor, “Interfacing a PS2 (Play Station 2) Controller”, couriousinventor.com, http://store.curiousinventor.com/guides/PS2/, Last access: May 22, 2015
8. Wikipedia, “USB”, Wikipedia.org, http://en.wikipedia.org/wiki/USB, Last access: May 22, 2015
9. Mikhaylov, K.; Huttunen, M., "Modular wireless sensor and Actuator Network Nodes with Plug-and-Play module connection," SENSORS, 2014 IEEE , vol., no., pp.470,473, 2-5 Nov. 2014
10. Juvva, A.; Gurkan, D.; Gumudavelli, S.; Wang, R., "Demonstration of the plug and play of smart sensor nodes," Sensors Applications Symposium (SAS), 2010 IEEE , vol., no., pp.265,268, 23-25 Feb. 2010
11. Bodenburg, S.; Lunze, J., "Plug-and-Play control - Theory and implementation," Industrial Informatics (INDIN), 2013 11th IEEE International Conference on , vol., no., pp.165,170, 29-31 July 2013