BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

EXTENSION SHIELD FOR TESTING OF ARM-BASED EMBEDDED SYSTEMS

ROZŠIŘUJÍCÍ MODUL PRO TESTOVÁNÍ VESTAVĚNÝCH SYSTÉMŮ S PROCESORY ARM

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR PETR VALENTA
AUTOR PRÁCE

SUPERVISOR Ing. VÁCLAV ŠIMEK
VEDOUCÍ PRÁCE

BRNO 2018

Brno University of Technology - Faculty of Information Technology

Department of Computer Systems Academic year 2017/2018

Bachelor's Thesis Specification

For:	**Valenta Petr**
Branch of study:	Information Technology
Title:	**Extension Shield for Testing of ARM-Based Embedded Systems**
Category:	Embedded Systems

Instructions for project work:

1. Investigate Bulldog - a Java library for direct manipulation with low-level I/O interfaces of ARM-based embedded systems.
2. Familiarize yourself with technical details of Raspberry Pi, CubieBoard and BeagleBone Black. Focus on types of I/O interfaces supported by the given platforms.
3. Design an extension shield for testing of the Bulldog library with the specified platforms. Choose suitable components and create detailed electrical schematics of the extension shield.
4. Create a PCB (Printed Circuit Board) according to the circuit diagram and populate it. Design and implement a simple firmware for the extension shield.
5. Extend the test suite of Bulldog library to enable use of the testing shield in order to verify the library's function and behaviour of the I/O interfaces supported by the platform.
6. Evaluate achieved results and propose further improvements.

Basic references:

- According to instructions of the supervisor.

Requirements for the first semester:
Fulfilment of the items 1 to 3 of the assignment.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Bachelor's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor:	**Šimek Václav, Ing.**, DCSY FIT BUT
Beginning of work:	November 1, 2017
Date of delivery:	May 16, 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

[illegible]

Lukáš Sekanina
Professor and Head of Department

Abstract

The purpose of this Bachelor's thesis was to design and create an extension shield for testing of the Bulldog library on an ARM-based embedded system. It covers the Bulldog library, describes its supported platforms and their I/O interfaces. The main part of this thesis is focused on design and manufacture of the extension shield. Its purpose is automating the tedious process of testing hardware.

Abstrakt

Tato bakalářská práce se zabývá návrhem a realizací rozšiřujícího modulu pro testování knihovny Bulldog na vestavěném systému s procesorem ARM. Zabývá se knihovnou Bulldog, popisuje podporované platformy a jejich vstup/výstupní rozhraní. Hlavní část práce je zaměřená na tvorbu rozšiřujícího modulu a doprovodný software. Vytvořený produkt si klade za cíl automatizovat zdlouhavé testování hardware.

Keywords

Bulldog, embedded systems, testing, Raspberry Pi, ATmega328p, I2C, SPI, UART, PWM

Klíčová slova

Bulldog, vestavěné systémy, testování, Raspberry Pi, ATmega328p, I2C, SPI, UART, PWM

Reference

VALENTA, Petr. *Extension Shield for Testing of ARM-Based Embedded Systems.* Brno, 2018. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Václav Šimek

Rozšířený abstrakt

Testování je velmi důležitou součástí každého vývoje. Vývojáři se jej snaží automatizovat jak to jen jde – stejně jako kteroukoliv jinou únavnou a zdlouhavou činnost. A právě tato snaha stojí za touto bakalářskou prací. Bulldog je knihovna v jazyce Java určená k ovládání nízkoúrovňových vstup/výstupních operací na vestavěných systémech s procesory ARM. Mezi podporované platformy patří například známá vývojová deska Raspberry Pi, která se těší oblibě mnoha žáků a učitelů, ale i kutilů a profesionálů. Bulldog dále podporuje desky CubieBoard a BeagleBone Black a klade si za cíl, aby se mohl kód, který byl napsaný pro jednu platformu, pustit na platformě jiné, s co nejmenším počtem úprav. Toho se snaží dosáhnout abstrakcí a objektovou orientací, která je jazyku Java, v němž je Bulldog napsán, vlastní.

Knihovna Bulldog podporuje několik rozhraní:

- Digitální vstup/výstup
- Přerušení
- Pulzně šířková modulace (analogový výstup)
- A/D převodník (analogový vstup)
- I^{2}C
- SPI
- UART
- Aplikační rozhraní pro různá zařízení (např. tlačítko, displej, servo)

Cílem této práce je vyvinout a vytvořit rozšiřující modul, který bude schopen testovat knihovnu Bulldog na hardwarové úrovni (na zmíněných rozhraních). Tento modul by se měl připojit přímo na vývody testované desky a bude z ní také napájen.

Při návrhu rozšiřujícího modulu se ukázalo jako nejvohnější použít mikrokontrolér. Dle několika různých parametrů byl zvolen mikrokontrolér ATmega328p-au. Testování za účelem návrhu schématu zapojení a návrhu desky plošných spojů probíhalo s pomocí mikrokontroléru ATmega328p-pu. Jedná se o téměř stejný mikrokontrolér, na rozdíl od ATmega328p-au není určen k povrchové montáži, ale je vhodný k zasazení do nepájivého pole.

Výsledný modul komunikuje s testovanou platformou pomocí rozhraní I^{2}C. Testovaná platforma přistupuje k I^{2}C registrům rozšiřujícího modulu, zapisuje do nich příkazy, čísla pinů, požadované hodnoty a čte z nich různé informace o stavu modulu. K usnadnění vývoje testů používajících tento modul bylo vytvořeno aplikační rozhraní. Vytvořený test pak může volat metody, které mají podobné názvy jako funkce knohovny Arduino vykonávané použitým mikrokontrolérem.

Kromě základních Arduino funkcí byly použity ještě knihovny `Wire.h` a `SPI.h`.

Schéma zapojení a deska plošných spojů byly navrženy pomocí open-source programu KiCAD. Deska byla zhotovena doma tzv. fotocestou. Fotocesta využívá postup zvaný fotolitografie, při kterém jsou za pomoci UV záření přeneseny obrazce z masky na vrstvu fotorezistu na plošném spoji. Ten je pak rozpuštěn roztokem hydroxidu sodného. Následně je odleptána odhalená měď roztokem chloridu železitého. Takto vyrobená deska pak byla ošetřena pájitelným lakem a navrtána na stojanové vrtačce. Domácí výroba plošného spoje umožnila velmi rychlé iterace ve vývoji rozšiřujícího modulu.

Po osazení součásktami a oživení byl rozšiřující modul testován pomocí logického analyzátoru a open-source programu sigrok.

Rozšiřující modul je možné programovat přímo z testované platformy pomocí utility `avrdude`, nebo pomocí AVR ISP programátoru na vyvedených pinech.

Na přiloženém CD je poskytnut i příklad testu, který využívá funkce vyvinutého rozšiřujícího modulu.

Hlavní cíl této bakalářské práce – vyvinout a vytvořit rozšiřující modul určený k testování knihovny Bulldog byl uspokojivě splněn. Vytvořené API navíc usnadní práci s tímto modulem. Kvůli časové tísni však bohužel nedošlo k vytvoření adaptérů pro připojení modulu k platformám CubieBoard a BeagleBone Black.

Během vývoje tohoto zařízení bylo objeveno mnoho chyb v knihovně Bulldog. Například čtení z I^2C je v porovnání s ostatními knihovnami občas nespolehlivé a při pokusu o PWM výstup dojde dokonce ke zhroucení celého operačního systému.

Ukázalo se tak, že byl projekt užitečný k odhalení různých neduhů knihovny. Do budoucna se nabízí různé vylepšení, zejména pak použití modulu ne k testování, ale k rozšíření vstup/výstupních rozhraní různých platforem.

Extension Shield for Testing of ARM-Based Embedded Systems

Declaration

I hereby declare that I have written this Bachelor's thesis on my own under the supervision of Ing. Václav Šimek. Additional information was provided by Ing. Pavel Macík of the Red Hat Czech s.r.o. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Petr Valenta
May 17, 2018

Acknowledgements

I would like to thank all people who helped me in any way during the development of this thesis. Huge gratitude goes to my parents and my supervisor Ing. Václav Šimek. Special thanks belongs to Ing. Pavel Macík who helped me learn many new skills.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Testing is a crucial part of any development process. Testing is a tedious task, and as any other annoying task, developers strive to automate as much of it as possible.

A similar need for automation is behind this thesis. Bulldog[1] — a Java GPIO library — needs a way to automate testing the I/O interfaces it controls. This thesis covers design and creation of an extension shield for this task.

The thesis is divided to several chapters. The next chapter is a research the Bulldog library. Chapter 3 researches existing solutions and describes design of the device. The final product, including accompanying software for the extension shield itself and API for the Bulldog library, is covered by Chapter 4. Ending with Chapter 5 – Conclusion, which looks at achieved results and possible improvements.

[1]https://github.com/SilverThings/bulldog

Chapter 2

Bulldog library

This chapter serves as an overview of the Bulldog library (Figure 2.1). It provides description of its various features and introduces supported platforms.

Bulldog is an open source Java library and is a part of the Silverspoon IoT Platform. It is used to control low-level I/O on specific Linux ARM-based single board computers, such as Raspberry Pi [21].

Figure 2.1: Logo of the Bulldog library [21]

Bulldog library can be compared to the Pi4J Project[1]. Both libraries are Java-based, however the supported platforms differ [33]. In a performance test, Bulldog has performed better than Pi4J. I has been able to change digital output with a frequency of 1.080 MHz, whereas Pi4J reached frequency of only 1 kHz [19]. This was not an official test, and it seems inconclusive to me, since I was able to find a benchmark of Pi4J in which it was able to reach frequencies of up to 161 kHz (using OracleJDK 7u10 and HotSpot virtual machine) [34]. However since both of these measurements are over two years old, the results are probably dated. Additionally, the switching frequency will probably never approach the frequency possible with code written in C, using the native library (up to 22 MHz) [30]. This is due to the more complex nature of Java and overhead introduced by the Java Virtual Machine.

[1]http://pi4j.com/

Bulldog aims to be platform agnostic, that means one piece of code can be run on multiple platforms. The library's features include [21]:

- Digital input/output on pins (GPIOs)
- Native Interrupts via epoll (usable on digital input pins)
- Native PWM, ADC
- I^2C
- SPI
- UART
- API for several devices (e.g. button, incremental rotary encoder, servos, LCD)

The Bulldog library is currently tested using JUnit framework. The tests are using mocked I/O [21]. This is great for testing of the library itself, but it can't test the physical I/O. This can be done manually, with an oscilloscope or a logic analyzer, which is very tedious and time consuming task. Thus a need for automated hardware test has arisen. The goal of this thesis is to design and create an extension shield for automated hardware testing of the Bulldog library, addressing the aforementioned problem.

2.1 PWM

PWM (Pulse Width Modulation – Figure 2.2) is a method for transmitting analog signal using digital output pin, which can be used in a wide variety of applications.

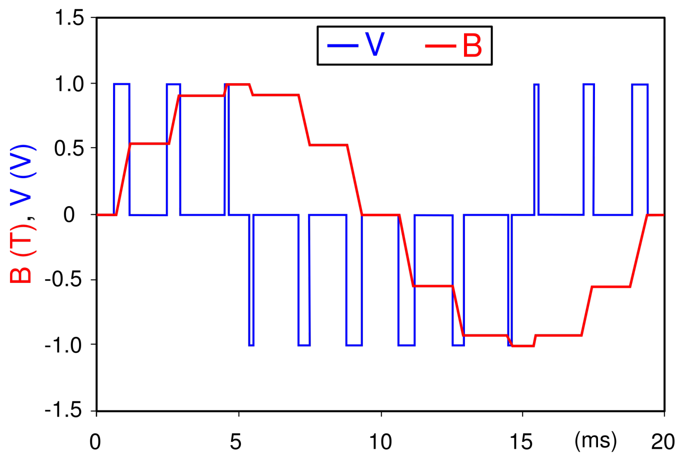


Figure 2.2: Current–Voltage relationship [10]

The square wave signal is created by turning the switch between the supply and the load fully on and off at a constant frequency. The duty cycle (Figure 2.3) – a percentage of time when the signal is high (on), is modulated to encode desired analog signal level. The time during which the supply is applied to the load is also called on-time, as opposed to the off-time when the supply is switched off. Given a sufficiently small period of the signal, any analog value can be encoded with PWM. The signal remains digital all the way from the processor to the controlled system, which minimizes noise effects [17].

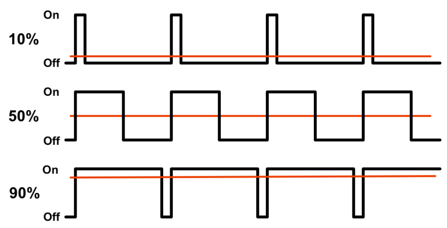


Figure 2.3: PWM duty cycle [31]

Although this modulation was originally developed to improve efficiency and reduce heating in control applications [16], it can also be used to encode information for transmission, where the pulse width corresponds to an encoded data value.

2.2 I^2C

I^2C (Inter Integrated Circuit) bus was developed by Philips Semiconductor (now NXP Semiconductors) in 1980s. It is used primarily for communication between multiple integrated circuits at close distances (usually on a single circuit board) [3].

2.2.1 Physical layer

Apart from ground (low) and supply voltage (high), I^2C uses only two bidirectional lines — SDA (data line) and SCL (clock line). Each line has a pull-up resistor, pulling up the line high. The I^2C driver is open drain, that means the device can pull the line low, but cannot drive it high. This is useful in preventing damage that could happen due to a device driving the line high while another tries to pull it low [35].

In Standard-mode the clock frequency is 100 kHz, however Fast-mode (400 kHz) and Fast-mode Plus (1000 kHz) exist [29].

Maximum pull-up resistor value depends on the operating frequency of the bus and the total capacitance of the bus (which depends mostly on the length and physical properties of wiring and the number of connected devices on the bus). Minimum resistance depends on the supply voltage and the amount of current the pin is able to sink. Example formulas for ATmega168 follow [15]:

$$\text{Freq} < 100\text{kHz} \implies R_{\text{min}} = \frac{V_{cc} - 0.4\text{V}}{3\text{mA}}, R_{\text{max}} = \frac{1000\text{ns}}{C_{\text{bus}}} \tag{2.1}$$

$$\text{Freq} > 100\text{kHz} \implies R_{\text{min}} = \frac{V_{cc} - 0.4\text{V}}{3\text{mA}}, R_{\text{max}} = \frac{300\text{ns}}{C_{\text{bus}}} \tag{2.2}$$

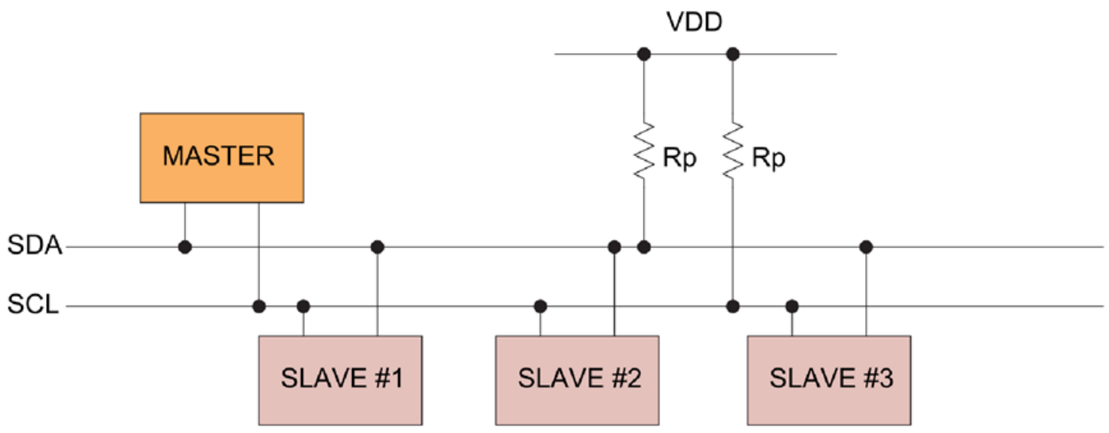


Figure 2.4: Generalized I^{2}C connection diagram [13]

2.2.2 Protocol

Each device connected to the bus can operate as either transmitter or receiver and is recognizable by a unique address. I^{2}C is using a master/slave model of communication with the ability to accommodate multiple master devices on one bus, provided that each master device is a multi-master (i.e. supports busy bus detection and is able to follow arbitration logic). The master both initiates and terminates any transfer. The clock signal is always generated by the current bus master, however in the event known as "clock stretching", some slave devices may drive the clock low if they need more time to prepare data for master or store data from master [29].

Messages consist of one address frame (7-bit address and one bit to select read or write operation) and one or more data frames (8-bit each)(Figure 2.5). One acknowledge bit (sent by receiving device) follows after each frame. SDA can be changed when SCL is low and is read when SCL is high. This has two exceptions, Start and Stop conditions.

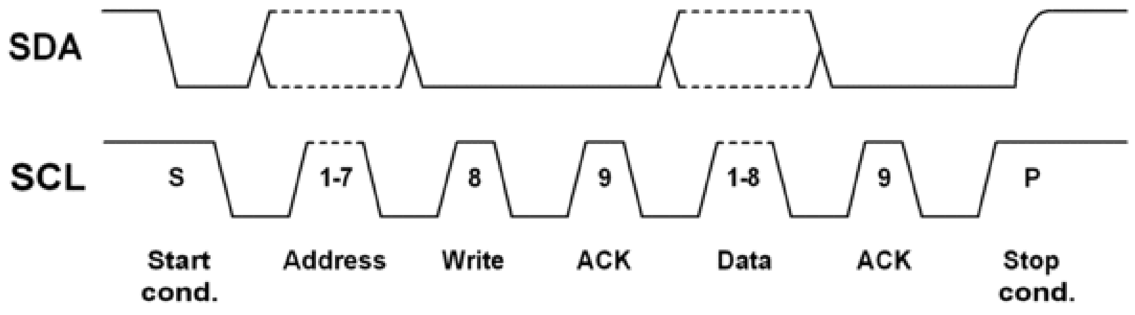


Figure 2.5: I^{2}C protocol example [1]

Figure 2.5 shows an example of a transmission on the I^{2}C bus. The transmission occurs as follows:

- Idle (free bus) — Both data line and clock line are high. No transmission occurs.

- Start condition — Master initiates a transmission by leaving SCL high and pulling SDA low. This notifies any devices on the bus, and the bus is considered busy.
- Transmission — Data is being sent by either master or slave depending on which operation master specified in the last bit of address frame. The data is clocked out in most significant bit (MSB) first manner. After every 8 bits, receiving device responds with either "acknowledge" (ACK) or "not acknowledge" (NACK). ACK is issued by pulling SDA low, NACK occurs when SDA is high during the last clock cycle. NACK can occur automatically, due to lack of any action, thanks to the pull-up resistor on SDA.
- Stop condition — Master terminates the transmission by a low to high transition on SDA line while SCL is high.

2.2.3 TWI bus

TWI (Two Wire Interface) bus is essentially the same as I^2C bus. It is used by Atmel and other vendors to avoid patenting issues. Depending on the device, some I^2C features, such as clock stretching, general broadcast, or 10-bit addressing, might not be supported [6].

2.3 SPI

SPI (Serial Peripheral Interface) bus is a serial communication interface developed by Motorola in the mid 1980s. It is used for a short distance communication between integrated circuits, primarily in embedded systems.

2.3.1 Interface

SPI uses a master/slave model of communication with a single master device and one or more slave devices. The bus uses four signal lines (Figure 2.6): SCLK – Clock signal, generated by the master; MOSI – Master Output Slave Input, data line driven from the master; MISO – Master Input Slave Output, data line driven from the slave and SS – Slave Select, driven from the master to each slave individually.

2.3.2 Operation

The master device initiates all communication with slaves by generating the clock signal and selecting a slave device by pulling corresponding SS line low. Master sends the data on MOSI line, while the data from the slave device are received on MISO line (Figure 2.7). Full-duplex data are transmitted on each clock cycle, even when one-directional data transfer is intended, therefore it is up to each device to consider the data meaningful or not [18]. Master terminates the transmission by pulling SS back to high.

Apart from the clock frequency, the master configures the polarity and phase, using two parameters: CPOL (clock polarity) and CPHA (clock phase). CPOL specifies the clock cycle, while CPHA specifies when the data is propagated and captured.

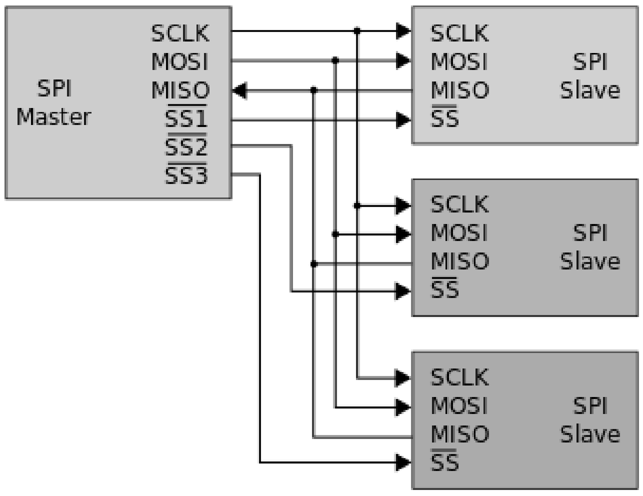


Figure 2.6: SPI bus: master and three slaves [11]

- CPOL = 0: A logic high indicates a clock cycle, clock idle state is logic low
- CPOL = 1: A logic low indicates a clock cycle, clock idle state is logic high
- CPHA = 0: The data is captured on the leading (first) clock edge
- CPHA = 1: The data is captured on the trailing (second) clock edge

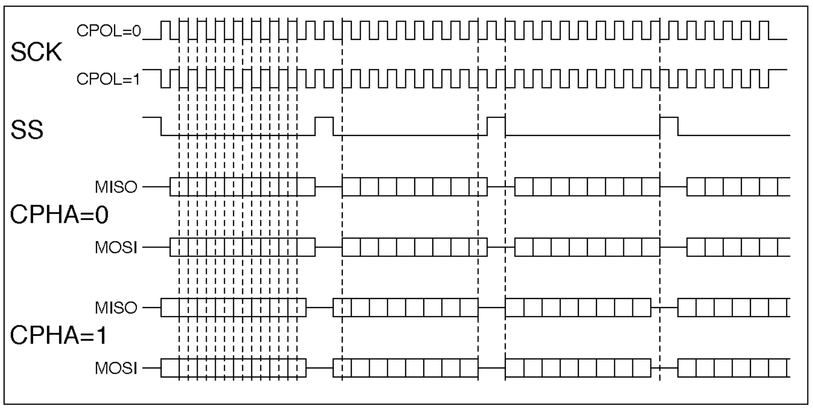


Figure 2.7: A timing diagram showing clock polarity and phase [22]

The combinations of CPOL and CPHA are referred to as modes. If the different slaves are fixed in different modes, the master has to reconfigure itself each time to communicate with a given slave.

2.4 UART

UART (Universal Asynchronous Receiver-Transmitter) is a hardware component that receives and transmits asynchronous serial data, first designed by Gordon Bell of DEC (Digital Equipment Corporation).

Interface

In order to transmit the data from a controlling device, like a CPU to a receiving device, two UART units (Figure 2.8) are used to communicate directly with each other. The transmitting UART gets the parallel data from the data bus and converts it into a serial form, to send it from its Tx (transmit) pin to the Rx (receive) pin of the receiving UART. The receiving UART converts the data back into a parallel form for the data bus of the receiving device [8].

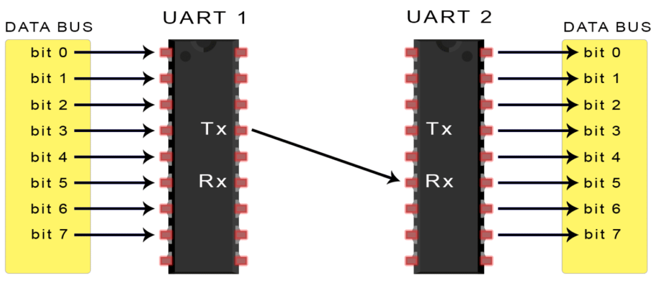


Figure 2.8: UART communication example [8]

Data flow

Because of its asynchronous nature and therefore the lack of a clock signal, both UARTs must operate at the same pre-configured baud rate. The transmitting UART also needs to add a start bit and stop bits, defining the beginning and the end of a data packet. The start bit is normally held high, so a high to low transition indicates the start of a transfer. To stop the transfer, the transmitting UART drives the line from low to high for two bit durations.

Optionally, a data packet (Figure 2.9) can also contain a parity bit. Both a parity bit and start / stop bits are removed when the data is converted back to parallel.

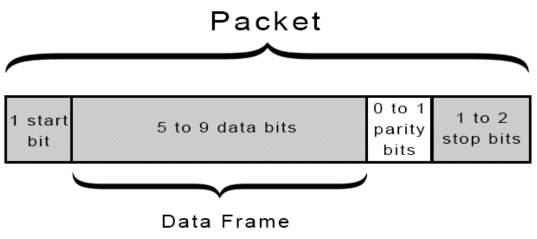


Figure 2.9: UART packet [8]

2.5 Supported platforms

As of April 2018, Bulldog library supports three different embedded Linux platforms — Raspberry Pi, CubieBoard, and BeagleBone Black [21].

2.5.1 Raspberry Pi

Raspberry Pi is a name of single-board computers designed by a UK-based charity – the Raspberry Pi Foundation[2]. The latest iteration — Raspberry Pi 3 Model B+ — is shown in figure 2.10. It uses a Broadcom BCM2837B0 SoC with a 1.4 GHz 64-bit quad-core ARM Cortex-A53 processor. It has 1 GB RAM, 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, four USB 2.0 ports, Ethernet port, HDMI, 40-pin GPIO header and more. Operating system is loaded from a MicroSD card [32]. Most operating systems for Raspberry Pi are Linux based, including the official operating system – Raspbian.

Its main advantages are large community, low price, and availability.

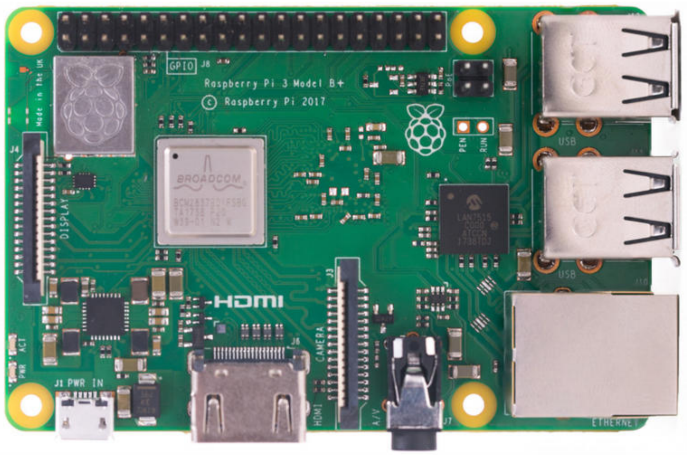


Figure 2.10: Raspberry Pi 3 Model B+ [24]

2.5.2 CubieBoard

CubieBoard[3] is a series of open source hardware single-board computers designed and produced by cubieteam[4] in Zhuhai, China. The newest model — CubieBoard6 (fig 2.11) — is powered by Actions S500[5] SoC with quad-core ARM Cortex-A9 processor. As with the latest Raspberry Pi, CubieBoard6 supports WiFi (802.11b/g/n), Bluetooth (4.0), has a MicroSD card slot and Ethernet port. It is pricier than the Raspberry Pi, but has additional features to justify the cost. These features include 2 GB RAM, USB 3.0, SATA, and an 8 GB eMMC on-board storage [28].

[2]https://www.raspberrypi.org/
[3]http://cubieboard.org/
[4]http://www.cubietech.com/
[5]http://www.actions-semi.com/en/productview.aspx?id=209

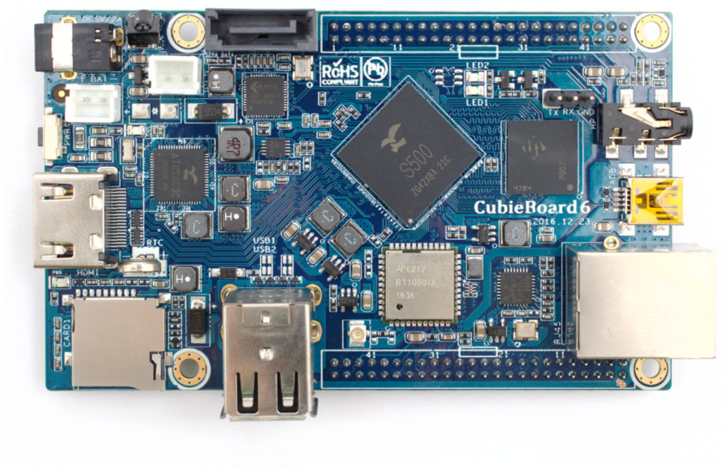


Figure 2.11: CubieBoard6 [23]

2.5.3 BeagleBone Black

BeagleBone Black (fig. 2.12) was first released in 2013 by BeagleBoard.org[6], an US-based non-profit corporation. The latest revision (Rev C) came out in 2014 and uses a Texas Instruments AM3358 SoC with 1GHz Sitara ARM Cortex-A8 processor. It has 4 GB of on-board eMMC storage. Being the oldest of the boards covered in this chapter, its features are lacking compared to them. It has no WiFi or Bluetooth capability, and has only 512 MB of RAM. However, another model of BeagleBone — BeagleBone Wireless — replaces the Ethernet port with 802.11 b/g/n 2.4 GHz WiFi and Bluetooth [7]. Considering its age and features, it is pricey compared to the other boards.

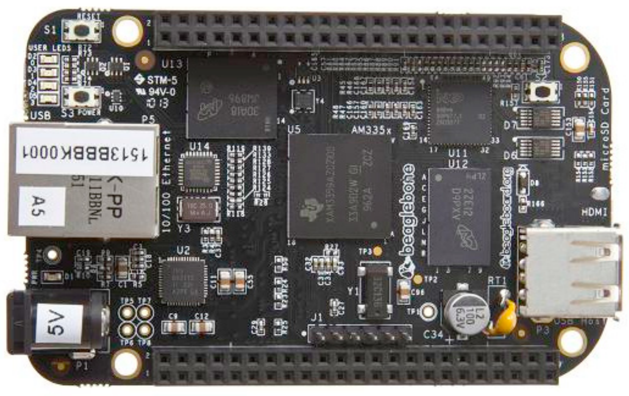


Figure 2.12: BeagleBone Black [25]

[6]https://beagleboard.org/

Chapter 3

Extension shield design

The aim of this thesis is to create a shield for different platforms (Raspberry Pi, CubieBoard, BeagleBone Black). The purpose of this shield is to test the Bulldog library on these platforms.

This chapter looks into existing solutions for testing embedded devices. Afterwards it covers the design of an extension shield for testing the Bulldog library.

3.1 Existing solutions

Currently there aren't many hardware solutions for testing I/O on ARM boards. WiringPi — a C library and a command line utility for controlling Raspberry Pi I/O — was tested using GPIO expanders and shift registers. This approach has a limited speed, because a single bit needs 10 clock pulses to get to the shift registers. It also does not test analog inputs or outputs[1].

In the embedded hardware industry, automated solutions exist in the form of test fixtures. Devices such as thermostats or even mobile phone CPUs can be tested in different test fixtures [12]. Details about specific testing fixtures are difficult to find since most of them are in-house custom-built solutions.

3.2 Hardware

This section will cover the process of designing the shield.

3.2.1 Format

According to the specification of this thesis, its goal is to build a shield (i.e. dautherboard) for Bulldog's supported platforms. The term "shield" was used mainly in the Arduino ecosystem, but has since become more widespread.

[1]http://wiringpi.com/

Raspberry Pi extension boards are called HATs (Hardware Attached on Top). BeagleBone uses the term "cape". An example of Raspberry Pi HAT can be seen in Figure 3.1.

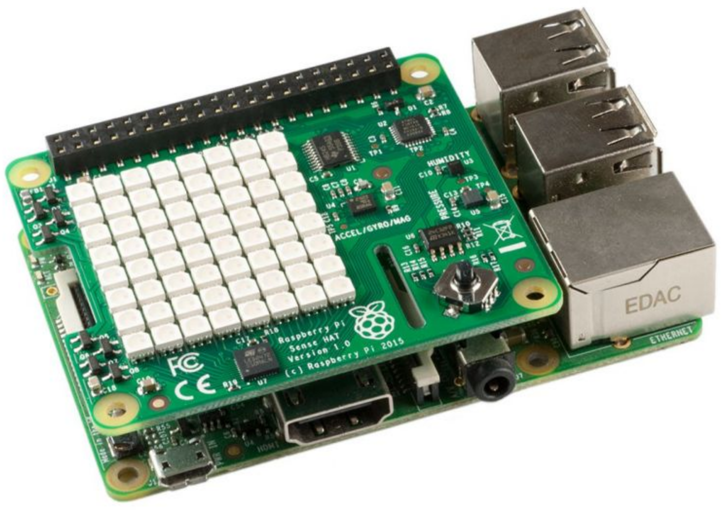


Figure 3.1: Raspberry Pi with Sense HAT attached [4]

Because the platforms don't share one layout, or even one footprint, it would be very difficult if not impossible to design one shield that could fit all of them. To design three different shields would be impractical and uneconomic, therefore my proposed solution was to design a shield that would connect to the boards using wires or adapters.

Wires were ruled out as too complicated and time consuming for the end user. Because Raspberry Pi was available, I have decided to design the shield with its pinout in mind. Adapters for BeagleBone Black, CubieBoard, and any other future platform can be added later.

3.2.2 Concept

The shield should be able to test I/O interfaces that are supported by the Bulldog library. This means finding a way to test GPIO, I^{2}C, SPI, PWM, UART, and ADC. My first idea was using shift registers to store a value that could be read later by the tested board itself. One reason this idea was abandoned was because testing anything but GPIO would be too intricate. The other reason being is that the solution wouldn't be very flexible compared to using a microcontroller, which could be programmed to change the properties of the shield.

Moreover, microcontrollers possess a wide range of different interfaces, many having all the interfaces supported by Bulldog library. The tested board could actively communicate with the microcontroller and control its function (e.g. run different tests). The next step is to find a suitable microcontroller for this task.

3.2.3 Microcontroller selection

In order to be able to test every bulldog feature, the desired microcontroller should have parameters that are described in Table 3.1.

Since all three boards use output signal voltage of 3.3 V and their inputs are not 5 V tolerant, another important feature is being able to operate at 3.3 V. Otherwise, level shifters would be needed in order to protect the tested board (or the shield) from high voltages. The microcontroller should be powered from the tested board itself, therefore it can't draw too much current as it could potentially damage the power rails or crash the system running on the board. The information about maximum current you can draw from the 3.3 V rail of Raspberry Pi, CubieBoard, and BeagleBone Black is difficult to find and isn't in any official documents. According to on-line forums, the current you can draw from Raspberry Pi 3.3 V power rail is at least 50 mA [27]. Both CubieBoard and BeagleBone Black should be able to support more. This should be easy to achieve, since there is a lot of microcontrollers that draw less than that.

Microcontroller Parameters	
Digital input/output	Required
I^2C as a slave	Required
SPI as a slave	Required
Analog input	Required
PWM	Required
UART	Required
3.3 V source voltage	Not Required
3.3 V logic level	Not Required
Maximum current draw 50 mA	Required
Small footprint	Required
Low cost	Not Required

Table 3.1: Parameters for selecting a microcontroller

ATmega328P-au by Atmel was selected (Figure 3.2). It complies with all required parameters and also not required parameters. It is also available at most electronic components store at low prices. Being a core component in many boards from the Arduino family, extensive information about this microcontroller and its many uses is readily available.

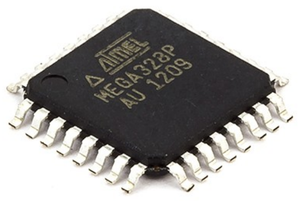


Figure 3.2: ATmega328p-au [5]

SMD (Surface-Mounted Device) version is preferred over the through hole version due to smaller footprint and no need to drill any holes. The QFP (Quad Flat Package) will be easier to solder than other available ATmega328P packages thanks to its extended leads. Some selected parameters are presented in Table 3.2.

ATmega328P-au Parameters	
Operating Voltage	1.8 V – 5.5 V
Peripherals	• Six PWM Channels • 8-channel 10-bit ADC • Two Master/Slave SPI Serial Interface • One Programmable Serial USART • One 2-wire Serial Interface (Philips I^{2}C compatible) • Internal Calibrated Oscillator
Maximum clock frequency	20 MHz
Program Memory Size	32 kB
Power Consumption	0.2 mA (Active at 1 MHz, 1.8 V)
Footprint	TQFP-32 (7 mm x 7 mm)
Price	2 €

Table 3.2: ATmega 328P-au — selected parameters [14, 2]

3.2.4 Connection

The shield will be connected directly to the Raspberry Pi 2x20 male header with female header of the same size. On the shield itself, there will be connections between leads of the microcontroller and corresponding Raspberry Pi pins, and also any additional passive components needed (e.g. pull-up resistors for SDA and SLC lines). Figure 3.3 shows a rough concept.

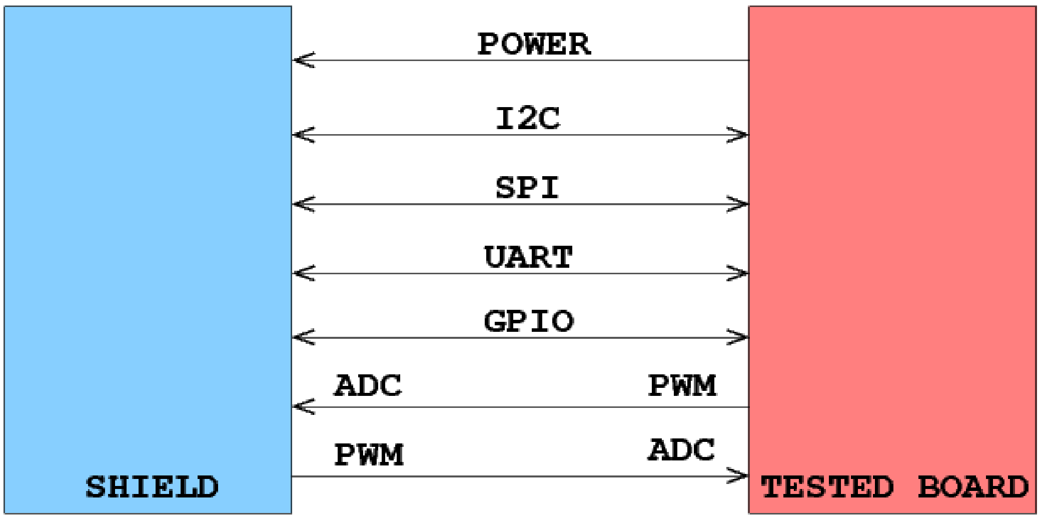


Figure 3.3: Concept of connections between shield and tested board

The shield will be powered from the board itself. To test all the interfaces, it should be able to set digital output for the board to read with its digital input (and vice versa). It should act as an SPI slave, I^2C slave, and communicate with the board using UART. It should also read PWM signal from the board as an analog input and ouptut PWM signal, which will be read by the tested board.

3.2.5 Shield function

There are two main ways the shield could function. It could act as an independent device, with preprogrammed outputs, and always listening (and answering) on the I^2C, SPI, and UART interfaces. However some problems exist. For example the tested board wouldn't know, what value the shield read on one of its pins. Some solution would be needed for this problem (e.g. the shield could read a value on one GPIO pin and then output it on a different pin).

The other option requires a communication protocol between the board and the shield. The board will actively control the actions of the shield by sending messages. The board should also be able to request data from the shield. The tests can therefore be more complex.

The selected approach is a mix of the two mentioned. The shield will have preprogrammed outputs, but can be controlled by the tested board.

3.2.6 Communication

A communication interface has to be selected. Since the shield will already be connected with the board using several interfaces, the selection was narrowed down to UART, SPI, and I^2C.

UART was ruled out because of its asynchronous nature. SPI has the highest data rate of the three, but has no standardized communication protocol. I^2C was selected because it seemed like the easiest to implement and debug, using the register approach. Registers used are shown in Table 3.3.

Name	Access	Offset	Description
STATUS	R	0x00	Status of the shield.
COMMAND	R/W	0x01	Command register (e.g. digital input/output).
PIN_NO	R/W	0x02	Selected pin.
PIN_VALUE	R/W	0x03	Selected pin value.
CONFIG	R/W	0x04	Additional configuration register.
ID	R	0x05	Identification number of the shield's HW.
VERSION	R	0x06	Version of the software.

Table 3.3: Shield I^2C register overview

3.2.7 Prototype

Prototyping, the activity of creating prototypes, is an important element in hardware design. Several prototypes were created using a through hole version of the microcontroller — ATmega328p-pu — which is for our purpose identical to the SMD version. This version is easier to gen in local stores, but its main advantage is being able to insert it into a breadboard. One such prototype can be seen in Figure 3.4.

3.3 Software

In order for the proposed shield to work properly, it has to be accompanied by a program. This program should control I/O interfaces of the shield in such a way that would enable the tested board to interpret results of the tests.

3.3.1 Arduino

Thanks to using Atmega328p, we can take advantage of the Arduino ecosystem and its many libraries[2]. This will make development of the software for the shield considerably easier.

[2]https://www.arduino.cc/en/Reference/Libraries

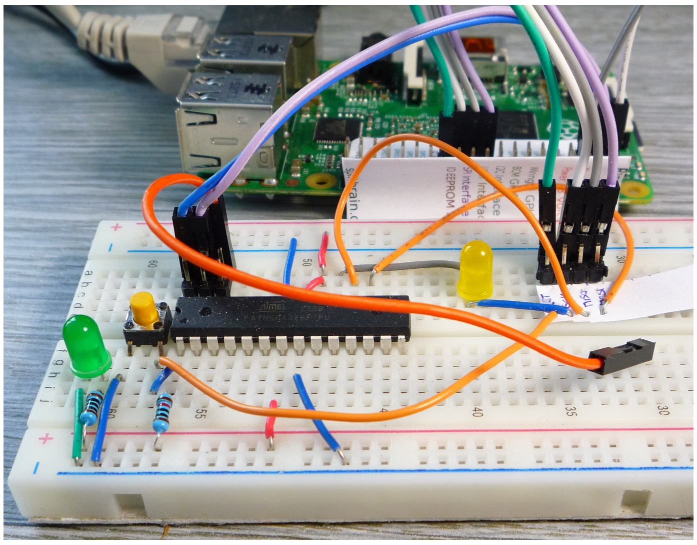


Figure 3.4: Breadboard prototype

3.3.2 Shield as a Bulldog device

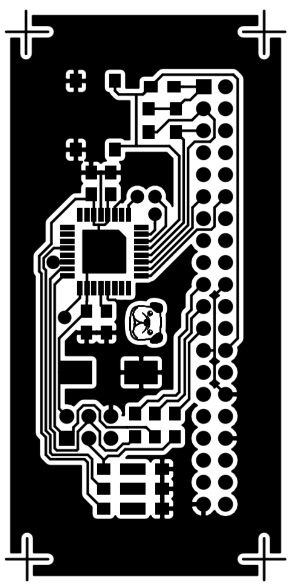
Bulldog library already supports several devices. These are represented as a Java packages with methods that aim to make using these devices with the Bulldog library easier.

Creation of similar API for the shield could lead to simplifying development of tests using this device. This way, the test itself does not have to deal with controlling the communication with the shield, it can use more abstract methods provided by the API.

Chapter 4

Implementation

This chapter covers the manufacture of the final product, and provides information about the developed software. PCB schematics and source files are published at GitHub[1].


(a) Top side

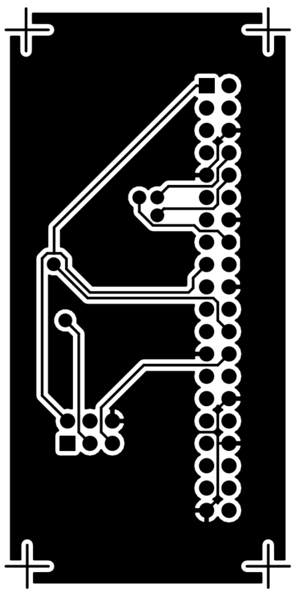

(b) Bottom side

Figure 4.1: Shield PCB schematic

4.1 PCB

The final product is a double sided board designed using an open source software suite KiCad EDA[2]. The board was manufactured at home. This section will cover the creation of the board at home. Ordering a PCB has become affordable, but making the board at home enables fast prototyping. Figure 4.4 shows one of the prototypes.

[1]https://github.com/SilverThings/bulldog-test-shield
[2]http://kicad-pcb.org/

The board was designed according to AN2519 - AVR Microcontroller Hardware Design Considerations[3]. This document was useful for designing layout of the PCB and selecting correct component values. I^2C pull-up resistor value was calculated using 2.1. Figure 4.1 shows both sides of the final design.

4.1.1 Manufacture

There are two main ways of making a PCB at home:

- Milling
- Etching

Milling is done using a CNC (Computer Numerical Control) mill. Using a rotary cutting tool, the mill removes areas of copper in order to create the desired pads and traces [26]. It is easier of the two methods, doesn't require the use of chemicals, but requires more expensive equipment. Therefore etching was chosen.

Similarly to the milling process, **etching** also works in subtractive manner by removing copper. Ferric chloride is widely used for etching copper PCBs [26]. In order to create the traces, the etching chemical must be prevented from accessing areas of the board. At home, this is can be done two ways. The toner transfer method uses a laser printer to print a negative of the PCB design on a paper. The toner is then transfered to the board using heat.

The second method uses a light sensitive material — a photoresist — to coat the board. Using a light-blocking mask, parts of the PCB are exposed to light. A developer is then used to dissolve parts of the photoresist. A diagram of this process is shown in figure 4.2.

The PCB was made using the etching method with presensitized positive photoresist copper-clad board. Mask was printed using a laser printer onto an ordinary printing paper. The mask was then treated with TRANSPARENT 21[4] spray, which makes the paper translucent to UV light. The photoresist was exposed using an UV nail curing lamp for 165 seconds. Because time varies based on used equipment and materials, correct exposure time has been found by trying different exposure times on one board.

[3]http://ww1.microchip.com/downloads/en/AppNotes/AN2519-AVR-Microcontroller-Hardware-Design-Considerations-00002519B.pdf

[4]http://www.kontaktchemie.com/KOC/KOCproductdetail.csp?product=TRANSPARANT%2021

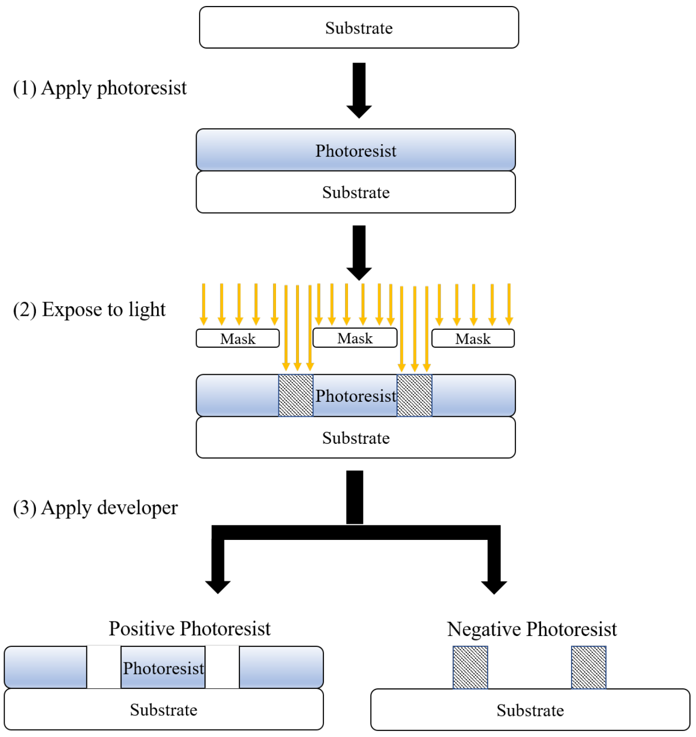


Figure 4.2: Photolithography [9]

Because the final product is a double sided board, this had to be done from both sides. In order to achieve sufficient precision, guiding crosses were added to each corner of the mask. Guiding holes were drilled into the PCB using a 0.5 mm drill bit. A thin wire was then inserted into each hole and used to guide the mask (Figure 4.3).

After exposing both sides of the board with UV lamp, exposed photoresist was washed away in a photoresist developer – 1,5% sodium hydroxide solution. The board then must be cleaned with water in order to remove any excess developer. Next it was placed in etchant – warm ferric chloride solution, which slowly removed the exposed copper. After etching, the board was rinsed off in water and dried. FLUX 10 SK[5] spray was applied to both sides of the PCB to facilitate soldering and protect the copper from corrosion.

4.1.2 Drilling

The PCB had to be drilled in order create vias and mounting holes for the headers. The mounting holes were drilled using a 1 mm PROXXON wolfram vanadium steel drill bit. The vias were drilled with a 0.5 mm PROXXON wolfram vanadium steel drill bit.

I was unable to produce good enough holes with a standard consumer rotary tool. Therefore the board was drilled on a drill press at Industra[6], a makerspace located in Brno.

[5]http://www.kontaktchemie.com/KOC/KOCproductdetail.csp?product=FLUX%20SK%2010
[6]https://industra.space/

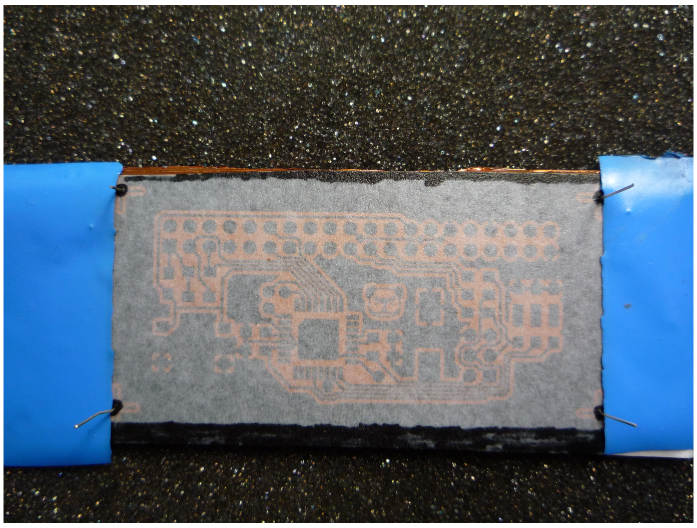


Figure 4.3: PCB with a mask

4.1.3 Assembly

The board was populated with components listed in Table 4.1 using a soldering iron. Because the holes aren't plated, through hole components (the two headers) had to be soldered from both sides of the PCB. Vias were made conductive by inserting a wire and soldering it to both pads.

Part Number	Description	Footprint	Quantity
1	Tantalum Capacitor 100 µF	7343	1
2	Capacitor 100 nF	0805	4
3	LED (various colors, forward voltage 2 V)	0805	3
4	2x3 male header	2.54 mm	1
5	2x20 female header	2.54 mm	1
6	Resistor 68R	0805	3
7	Resistor 10k	0805	1
8	Resistor 2K7	0805	2
9	Resistor 330R	0805	3
10	Tactile switch	7 mm x 7 mm	1
11	ATmega328p-au	TQFP-32	1

Table 4.1: Bill of Materials

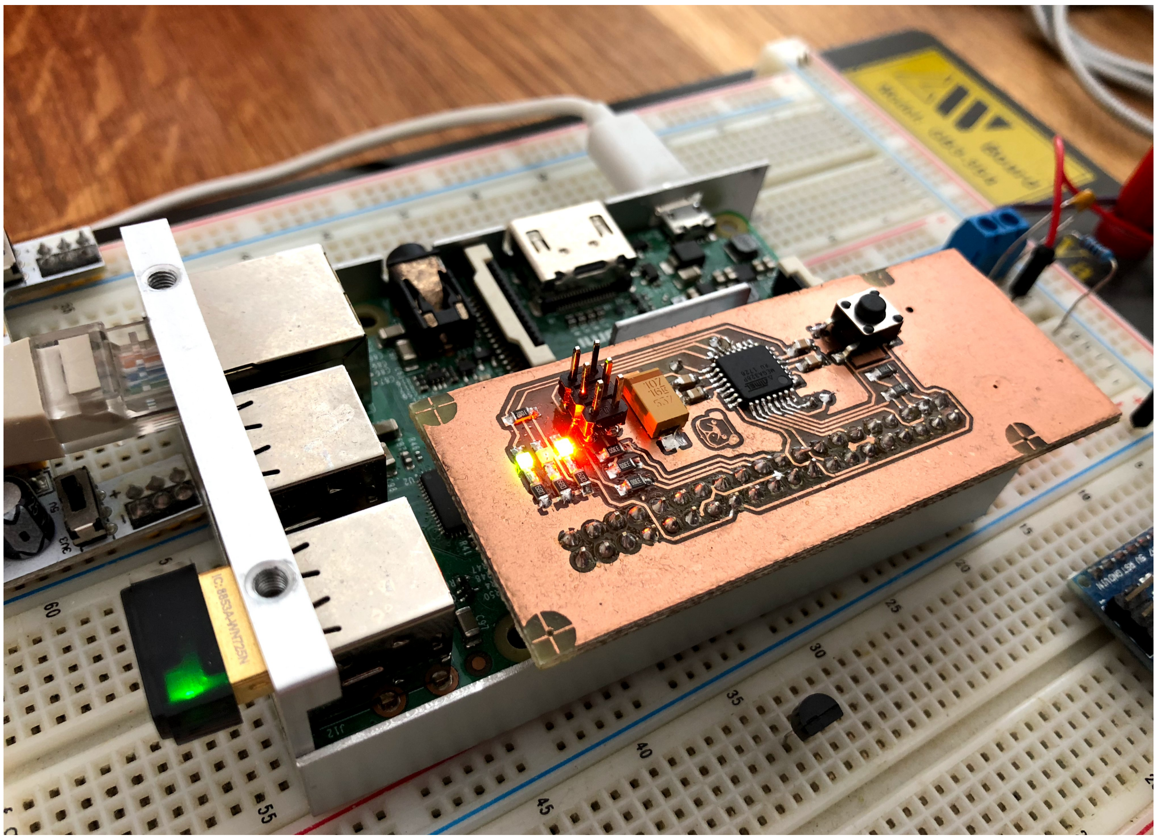


Figure 4.4: Powered on Raspberry Pi with a final prototype of the shield

4.2 Software

This section will provide information about the software powering the shield and describe corresponding methods in the API for Bulldog.

4.2.1 Shield

The code was written in C++ using Visual Studio Code. The program was built using `arduino-mk` and flashed with `avrdude`. It is using Arduino libraries `Wire.h` and `SPI.h`.

Initialization

After power up, global variables are set to default values and function `setup()` is executed, which initializes the shield. Most importantly it starts listening to events on the I^{2}C lines. The current state of the program is stored in registers (described in Table 3.3), which are accessible through I^{2}C.

Afterwards, the program loops consecutively in the function `loop()`. Without capturing any I^{2}C events, a "demo output" takes place. The program outputs on all tested interfaces (i.e. digital output, PWM, UART). This is useful for testing interfaces (i.e. digital input, ADC, UART) of the tested board without communicating with the shield. However to use the shield to its full potential, the board must communicate with the shield via I^{2}C.

Communication

After an I^{2}C write is captured, function `receiveEvent()` is executed. In this function, the shield reads data sent by a master (to a maximum length of `MAX_SENT_BYTES`) and saves them into an array. Based on the size of the array, the shield determines if the master wants to write data to the shield's registers or select a register to read from. If the master sends only one byte of data, it means the master wants to select a register to read from. The offset of this register is the only received byte of data.

If the shields receives more bytes of data, the first byte is an offset and the rest is the data to be written. The data is written in temporary registers and flags are set. The flags are checked after each block in `loop()`. If any flags are set, the main register is updated with newly received values and the `loop()` is reset. This ensures atomicity of the I/O functions.

Function `requestEvent()` is executed after an I^{2}C read occurs. The shield checks if it has any new relevant data, and updates its registers accordingly. Next it starts sending data byte after byte over I^{2}C, starting at register offset selected by the master in previous I^{2}C write event. Data is being sent until the master stops the transmission by generating a Stop condition, or until all data is sent.

Main loop

The shield performs different actions based on commands received through I^{2}C. Commands and corresponding actions are described in Table 4.2.

Value	Name	Description
0x00	INPUT	Sets mode of a digital pin.
0x01	OUTPUT	Sets mode of a digital pin.
0x20	NO_MODE	Initial state.
0x21	DIGITAL_INPUT	Reads from PIN_NO to PIN_VALUE.
0x22	DIGITAL_OUTPUT	Writes PIN_VALUE to PIN_NO.
0x23	ANALOG_INPUT	Reads from PIN_NO to PIN_VALUE.
0x24	ANALOG_OUTPUT	Writes PIN_VALUE to PIN_NO.
0x25	SPI	Starts/stops SPI based on PIN_VALUE.
0x26	UART	Starts/stops UART based on PIN_VALUE.
0x27	DEMO	Starts/stops demo output based on PIN_VALUE.

Table 4.2: List of COMMAND register values

4.2.2 Bulldog

An API was created to simplify usage of the shield. The shield is represented by an instance of Java class `Atmega328P`, which extends class `I2cDevice`. Methods such as `digitalWrite()` and `analogRead()` can be utilized by the test itself without the need of managing the I^{2}C connection. An example of such a test is located in directory `bulldog-examples` on the included CD.

4.2.3 Programming

The shield can be programmed using its ISP 6 pin connector with an AVR programmer. Another option is to program it directly from the Raspberry Pi. This capability is possible thanks to connecting the reset line to the Raspberry Pi. The reset line can be pulled down by the Pi, in order to reset the shield. The board is programmed using `avrdude` with special configuration file.

4.2.4 Testing

The shield was tested using a logic analyzer and sigrok[7] – an open-source signal analysis software suite.

[7]https://sigrok.org/

Chapter 5

Conclusion

The goal of this Bachelor's thesis was to design and create a testing device for the Bulldog library. A shield capable of testing low level I/O interfaces of single-board Linux computers. First part of the thesis researches the Bulldog library, explores its features and supported platforms.

Second part of the thesis briefly discusses existing solutions, covers the design of the extension, describes the developed board and accompanying software.

The design and implementation was successful with satisfactory results. The developed shield is based on ATmega328p microcontroller and is able to perform all required functions.

No adapters for BeagleBone Black or CubieBoard were designed due to time constraints. The developed API facilitates creation of tests using the shield.

5.1 Bulldog issues

In the process of designing the shield, several issues with the Bulldog library were discovered. I^2C can fail quite often compared to other libraries. In the current build, PWM causes a file system error that crashes the whole operating system.

5.2 Improvements

In the future, this project could improve by supporting additional I/O libraries and boards.

Another interesting area to explore would be performance testing of the I/O interfaces. This might prove difficult to achieve with the used hardware.

The project could also pivot – the developed board is in it's nature an I^2C I/O expander. With a different pinout it could find use in areas other than testing Bulldog supported platforms. This functionality could be improved by redesigning the layout of the board and improving the provided API.

Bibliography

[1] Addressing. [Online; visited 2018-04-26].
Retrieved from: https://www.i2c-bus.org/addressing/

[2] ATMEGA328P-AU. [Online; visited 2018-05-01].
Retrieved from: https://eu.mouser.com/ProductDetail/Microchip-Technology-Atmel/ATMEGA328P-AU

[3] I^{2}C. [Online; visited 2018-04-26].
Retrieved from: https://en.wikipedia.org/wiki/I%C2%B2C

[4] Raspberry Pi Sense HAT. [Online; visited 2018-04-29].
Retrieved from: https://www.modmypi.com/raspberry-pi/sensors-1061/orientationaccelerometers-1067/raspberry-pi-sense-hat

[5] TQFP ATMEGA328P-AU ATMEGA328P AU IC FOR ARDUINO. [Online; visited 2018-05-10].
Retrieved from: http://hallroad.org/product/tqfp-atmega328p-au-atmega328p-au-ic-for-arduino/

[6] Introduction into TWI. Feb 2016. [Online; visited 2018-04-27].
Retrieved from:
https://www.nongnu.org/avr-libc/user-manual/group__twi__demo.html

[7] Meet BeagleBone™ Black Wireless, the newest board in the BeagleBone™ family. Sep 2016. [Online; visited 2018-05-02].
Retrieved from:
https://beagleboard.org/blog/2016-09-26-meet-beaglebone-black-wireless/

[8] Basics of UART Communication. Apr 2017. [Online; visited 2018-04-18].
Retrieved from: http://www.circuitbasics.com/basics-uart-communication/

[9] Photoresist. Apr 2018. [Online; visited 2018-04-29].
Retrieved from: https://en.wikipedia.org/wiki/Photoresist

[10] Pulse-width modulation. April 2018. [Online; visited 2018-04-25].
Retrieved from: https://en.wikipedia.org/wiki/Pulse-width_modulation

[11] Serial Peripheral Interface Bus. April 2018. [Online; visited 2018-04-20].
Retrieved from:
https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

[12] Test fixture. Apr 2018. [Online; visited 2018-04-22].
Retrieved from: https://en.wikipedia.org/wiki/Test_fixture

[13] Afzal, S.: I^{2}C Primer: What is I^{2}C? (Part 1). [Online; visited 2018-04-26].
Retrieved from: http://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html

[14] Atmel: *ATmega328/P Datasheet Complete*. November 2016. [Online; visited 2018-04-25].
Retrieved from: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

[15] Atmel: *ATmega48/V / 88/V / 168/V Datasheet Complete*. November 2016. [Online; visited 2018-04-22].
Retrieved from: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2545-8-bit-AVR-Microcontroller-ATmega48-88-168_Datasheet.pdf

[16] Ball, S. R.: *Analog interfacing to embedded microprocessors: real world design*. Newnes. 2001. 252–253 pp.

[17] Barr, M.; Massa, A. J.: *Programming embedded systems: with C and GNU development tools*. OReilly. 2010. 254–256 pp.

[18] Barr, M.; Massa, A. J.: *Programming embedded systems: with C and GNU development tools*. OReilly. 2010. 248–250 pp.

[19] Bunčiak, S.: Bulldog: A surprisingly fast GPIO library. Apr 2016. [Online; visited 2018-04-18].
Retrieved from: https://opensource.com/life/16/4/bulldog-gpio-library

[20] Bunčiak, S.; Macík, P.: Silverspoon. [Online; visited 2018-03-15].
Retrieved from: https://github.com/SilverThings/silverspoon

[21] Datenheld; Bunčiak, S.: Bulldog. [Online; visited 2018-03-10].
Retrieved from: https://github.com/SilverThings/bulldog

[22] Dennis, A.: Introduction to Arduino SPI Library with LTC1286 and DAC714. Jul 2015. [Online; visited 2018-04-20].
Retrieved from: https://www.allaboutcircuits.com/projects/arduino-spi-library-ltc1286-dac714/

[23] Halfacree, G.: CubieBoard 6. Jul 2017. [Online; visited 2018-05-01].
Retrieved from: https://www.flickr.com/photos/120586634@N05/sets/72157685713607676

[24] Heath, N.: Raspberry Pi 3 Model B : See all the new features on the board. Mar 2018. [Online; visited 2018-04-29].
Retrieved from: https://www.techrepublic.com/pictures/raspberry-pi-3-model-b-see-all-the-new-features-on-the-board/

[25] jkrinder: BeagleBone Black. Sep 2016. [Online; visited 2018-05-01].
Retrieved from: https://beagleboard.org/Products/BeagleBoneBlack

[26] Khandpur, R. S.: *Printed circuit boards: design, fabrication, assembly and testing.* McGraw-Hill. 2006.

[27] klricks: Current limitation for 3v3 and 5v power rails - Raspberry Pi Forums. [Online; visited 2018-04-28].
Retrieved from: `https://www.raspberrypi.org/forums/viewtopic.php?t=103354`

[28] Lee, A.: CubieBoard6 is released to the overseas users. Dec 2017. [Online; visited 2018-05-11].
Retrieved from: `http://cubieboard.org/2017/12/27/cubieboard6-is-released-to-the-overseas-users/`

[29] NXP Semiconductors: *I2C-bus specification and user manual*. April 2014. rev. 6. [Online; visited 2018-04-22].
Retrieved from: `https://www.nxp.com/docs/en/user-guide/UM10204.pdf`

[30] Pihlajamaa, J.: Benchmarking Raspberry Pi GPIO Speed. Jul 2012. [Online; visited 2018-04-19].
Retrieved from:
`http://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/`

[31] Rampelt, J.: PWM programming activity. [Online; visited 2018-04-25].
Retrieved from: `http://www.siriusmicro.com/chrp3/pwm-c.html`

[32] Raspberry Pi Foundation: *Raspberry Pi 3 Model B+*. [Online; visited 2018-04-29].
Retrieved from: `https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf`

[33] Savage, R.: The Pi4J Project – Home. [Online; visited 2018-04-10].
Retrieved from: `http://pi4j.com/`

[34] Savage, R.: Raspberry Pi - Java GPIO Frequency Benchmarks. Jan 2013. [Online; visited 2018-04-18].
Retrieved from: `http://www.savagehomeautomation.com/projects/raspberry-pi-java-gpio-frequency-benchmarks.html`

[35] SFUptownMaker: I^{2}C. [Online; visited 2018-04-24].
Retrieved from: `https://learn.sparkfun.com/tutorials/i2c`

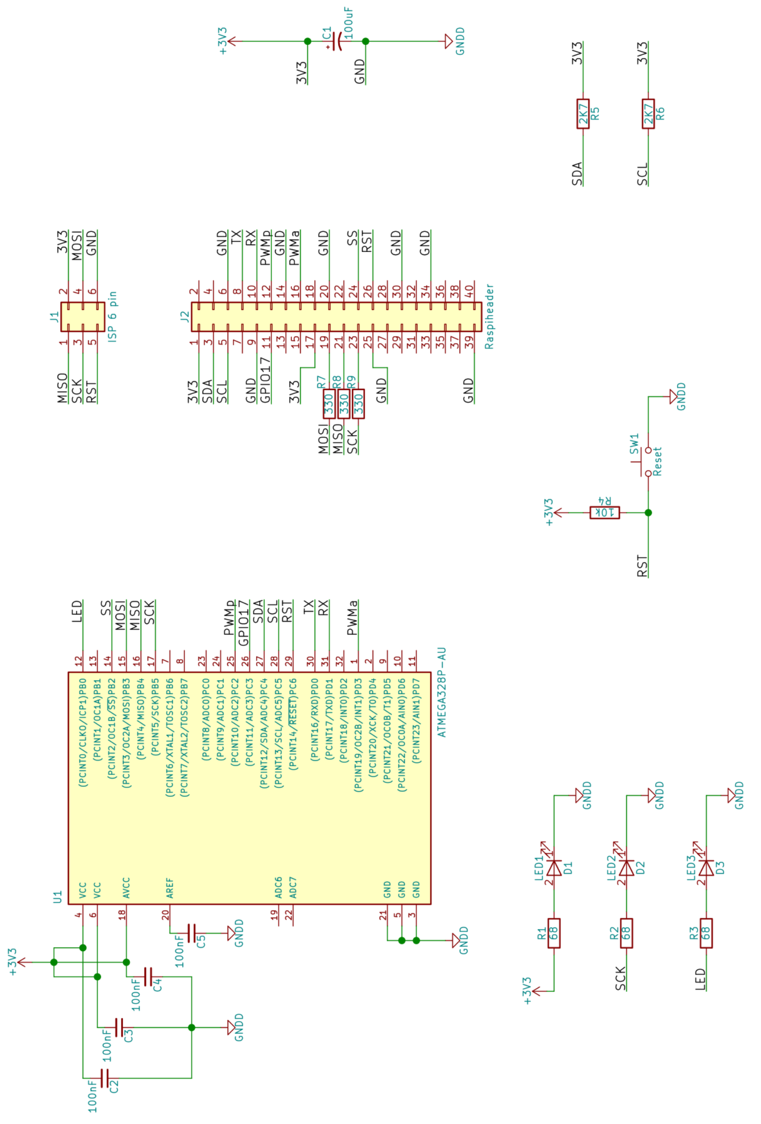
Appendix A

CD Contents

```
CD_xvalen20
├── bulldog ........................................ Copy of the Bulldog library
├── bulldog-test-shield
│   ├── src ................................................... Source code
│   └── testshield ............................................ Schematics
├── bulldog-examples ..................................... Test example
└── thesis
    ├── src
    └── xvalen20-bulldog-test-shield.pdf
```

Appendix B

Schematic



+3V3
C1
100uF
3V3
GND
GNDD
R5
2K7
R6
2K7
SDA
SCL
J1
ISP 6 pin
MISO
SCK
RST
MOSI
J2
Raspiheader
GPIO17
TX
RX
PWMp
PWMa
SS
R7
R8
R9
330
SW1
Reset
10k
U1
ATMEGA328P-AU
VCC
AVCC
AREF
ADC6
ADC7
LED
C2
C3
C4
C5
100nF
R1
R2
R3
68
D1
D2
D3
LED1
LED2
LED3