Portable Password Bank

A thesis submitted to the
Faculty of the Computer Engineering Technology Program
of the University of Cincinnati
in partial fulfillment of the
requirements for the degree of
Bachelor of Science

in Computer Engineering Technology
at the College of Engineering & Applied Science

by

NIKLAUS PHARO
AND
JUSTIN ROBINSON

Bachelor of Science University of Cincinnati
May 2011

Faculty Advisor: Michael Haas
June 7, 2011

Professor Michael Haas  
University of Cincinnati  
College of Applied Science  
2220 Victory Parkway  
Cincinnati, Ohio 45206

Dear Professor Michael Haas:

Attached is the final report for our senior design project, “Portable Password Bank”, which was approved by the ECET faculty with a deadline of June 7, 2011.

The purpose of this report is to explain the importance of the Portable Password Bank and how it solves the current problems a computer user has with login systems. A timeline and budget for designing and constructing the Portable Password Bank is also given. In addition, a description of the Password Bank’s functionality, universality, and the benefits provided by the Password Bank is conveyed.

Many thanks are owed to a couple of people for the completion of this project. First, thank you Daniel Picard for your assistance in explaining how to properly implement encryption and decryption on the graphical user interface used in our project. Secondly, thank you David Tashjian for your support with the utilization of the programming environment we used and advice given in priorities of what should be accomplished. Also, thanks goes out to Professor Haas for his availability and assistance with the project as the quarters went by.

Thank you for all your time and consideration. If you have any questions, please feel free to contact us at (513) 503-6024 or pharon@email.uc.edu (for Nik) and (513) 300-1835 or robin2jt@email.uc.edu (for Justin).

Sincerely,

Niklaus E. Pharo & Justin T. Robinson
DEDICATION PAGE

We would like to dedicate this project to our families. They have been there for us since we were born and have helped us be where we are today. In addition, we’d like to dedicate this to Professor Michael Haas for his enthusiasm and help throughout the year. He has vastly improved our skills in programming languages such as C#.

Furthermore, we would like to dedicate this project to our brave men and women serving for our freedom. They have enough acronyms to remember. Perhaps this project could help remember a few things for them.
TABLE OF CONTENTS

Abstract 1
Introduction 2
  Problem 2
  Solution 3
  Credibility 3
  Project Goals & Brief Methodology 4
  Overview 5
Discussion 5
  Project Concept 5
  Design Objective 6
  Methodology/Technical Approach 7
  Budget 9
  Timeline 10
  Problems Encountered/Analysis of Problems Solved 10
  Future Recommendations 11
Conclusion 12
References 13
Appendices 14

LIST OF ILLUSTRATIONS AND TABLES

Figure 1: Print a Password Design 4
Figure 2: Edit Password Design 5
Figure 3: AT89STK-05 Kit 7
Figure 4: Schematic 8
Figure 5: Main Menu of GUI 8
Figure 6: Portable Password Bank 9
Table 1: Final Budget 10
Figure 7: Final Schedule 10
ABSTRACT

The Portable Password Bank, PPB for short, is an all-in-one device that relieves computer users the task of memorizing passwords. This adds more simplicity to logging into a program or website. In addition, it aids the user in creating more secure accounts. The PPB will create and store complex passwords to provide users with an easy and secure way of logging in to a website. The Portable Password Bank is mostly hardware based, runs on any Windows XP, Vista, and Windows 7 PC and enters the stored password through USB without the need of any additional drivers. It can be used on any website or login screen, including the Windows login screen.
INTRODUCTION

The purpose of this final report is to summarize the design and construction of the Portable Password Bank. This document depicts how the software and hardware driven Password Bank serves as an all-in-one product that will save time and add security to the task of logging into a program or website.

Currently the only methods to log into a website are memorization and external devices that require proprietary software or drivers installed on the computer in order to operate. The Portable Password Bank will relieve computer users from needing to memorize passwords. In doing so, it will improve the account’s security by increasing the complexity of the password and save the time it takes to enter the password by automatically filling in the password field when the desired password is selected on the device. Most computer users have several login screens that consist of websites and applications. This device can be used on any of the login screens and will easily allow the user to populate the password field with a complex password using the password bank. This report discusses the current problem, methodology, implementation, benefits, as well as providing the work schedule and total budget of this project.

Problem
The average PC, Personal Computer, user has several passwords to remember for various applications. This can be insecure because users often have the same password for everything, forget their passwords, and make their passwords too easy to guess. Passwords are used on several different devices and applications. There are passwords for email accounts, for logging into computers, and for other private data. Users sometimes will use the same password for all of their needs. The obvious advantage of this method is that there is, for instance, one password to remember. With this method, passwords could be complex but it is still insecure because only one password is used. So, if someone hacked one account, that person could hack all of that user’s accounts. Conversely, another method people may use is to make several passwords, one for each application. With this method, passwords are typically made to be less complex so that all of them can be remembered easily. This makes them an easier target to hackers.

It can be a hassle to have to remember a complex password. A complex password must be used to prevent unauthorized access. But for optimal security, there should be a complex password for each application and login screen. With this method, there will be at least five or six complex passwords (depending on the number of login screens and other applications). Using this approach, it will be difficult for the user to remember all of the complex passwords. In addition to it being difficult to remember what each password is, it is also difficult to remember what each password is used for. Using any of these methods, a user may forget their password to the account to add more frustration.
**Solution**

Remembering a complex password and what it is used for can be difficult. Using the Portable Password Bank can abolish this. By using the Portable Password Bank, a user can have the device automatically generate complex passwords initially and then store them into the bank. After changing the password for an account, the user would then simply plug in the device to the computer with that login screen displayed, unlock the “master lock” by pressing a few button combinations, select a password, highlight the password field on the computer, and hit the “paste” button. This device will eliminate the need to remember passwords and can be used for login screens, any website, or other applications. It will also make the user’s accounts more secure by enabling him or her to have more complex passwords and unique passwords for each account.

This device will prevent access from hackers who try to read the passwords from the computer by disabling the “pasting” of passwords until the master lock has been unlocked.

**Credibility**

Even though any computer user can easily operate the Portable Password Bank, skillful and ambitious students were necessary to design, implement, and manage this project. Additionally, it was essential to seek the knowledge of professionals in the field for assistance with proper functionality issues. Our experiences as well as people we consulted with for this project are listed below.

**Qualification**

Over the years, we have gained experience with embedded systems through Co-Ops and school courses. More specifically, through programming courses in C, C#, and Object Oriented Programming (OOP), we have attained the knowledge and skills needed to design and construct the Portable Password Bank.

**Resources**

While researching information about the Portable Password Bank, it was necessary to speak with several University of Cincinnati faculty members. They provided background information and methods on how to interface our device using USB.

With his background in C# and the technical skills he acquired at P&G, Professor Haas, was the main source for technical guidance. David Tashjian, with his knowledge of embedded systems, also provided technical guidance early in the project.
**Project Goals & Brief Methodology**

An Atmel 89C5131 microcontroller, programmed in C, was used to allow the Portable Password Bank to interface to the computer. Once the programming was finished, the microcontroller could emulate a keyboard through USB. The microcontroller essentially sent hex code (as a keyboard normally would) to the computer. By doing this method, our device does not need any additional drivers. For added security, a master lock was used to prevent unauthorized users from using the stored passwords on our device. Initially, the device would not work until the master lock key combination is entered correctly. This key combination was predefined and is only known to the actual user. Figure 1 shows a block diagram of how the Portable Password Bank prints a password to a user’s computer.

![Figure 1: Print a Password Design](image)

A user friendly Graphical User Interface (GUI), programmed in C#, was also included with the Portable Password Bank. The GUI has several menu options to help the user manage his/her passwords. It gives the user the ability to edit the passwords, change the master lock, save all the passwords to their machine, and output/generate a password for a particular slot. The software is only used on the home computer of the user. For improved security, the Password Configuration program is password locked. Only the user knowing the password to the Configuration program can access the options listed above. Figure 2 shows a block diagram on how the Password Configuration program could be used to manage a password. In addition, we added AES 128-bit encryption to a function that would allow the user to save all passwords to the computer, encrypted. This could only be decrypted by the GUI.
Overview
The remainder of this report outlines how the project was completed. The report includes the following sections: project concept, design objectives, methodology, budget, timeline, problems encountered, and future recommendations.

DISCUSSION

This section provides an overview of the Portable Password Bank. A summary of the project objectives will detail the initial goals and their purpose. Also provided are the project budget, schedule, and the problems encountered during the construction of the Portable Password Bank. A description of further recommendations will also be illustrated.

Project Concept
The project idea began when both of us contemplated how insecure passwords can become. This seems to be usual for non-technical people since they often forget about computer security and the possibility that their accounts could be breached by unauthorized people. So the user has a couple of choices; they could make an easy password (so that they can remember it easily), or they could make a complex password and have a difficult time remembering it. In either case, it is more secure to have multiple complex passwords for different logins. It is difficult for anyone to remember 3 or 4 complex passwords. These complex passwords could include: percent signs, numbers etc. Therefore, instead of having to remember a long list of complex passwords, the user could simply use the Portable Password Bank. The Portable Password Bank will even create the complex password for the user. All the user has to do is simply assign the newly created password to a specific slot (button) so that it can be retrieved later. By using the Portable Password Bank, there is no need to ever remember a password.
Design Objective

The Portable Password Bank was meant to meet a number of objectives. While all of the major objectives were met, the fingerprint reader (used as a master lock) became less essential and was replaced by a specific key combination instead. By not implementing the fingerprint reader, the project was completed on time. It would have taken several months and additional funding to develop the software for the fingerprint reader and to integrate it with the Portable Password Bank. For simplicity, a key combination was used.

The first objective was to use an embedded system to interface the Portable Password Bank through a computer’s USB port. This includes emulating a keyboard for the purpose of entering passwords into a password field without needing a driver. With this method, the device could be used on any Windows machine that has a USB device.

The second objective was to allow the user to toggle between passwords using up/down arrows. Instead of this approach, one password was assigned to each button. Since there were four buttons, four passwords could be retrieved. The user would simply hold the button corresponding to the saved password and then press the password button. The buttons were numbered 1-4. At the same time, the menu to edit or print out passwords would be developed.

The next objective was to create a Password Management Graphical User Interface that would allow users and/or system administrators to view and edit the passwords stored on the device. The Password Manager acting as a visual representation of the menu built into the Portable Password Bank would assist the user in editing passwords. This application also needed a password lock to prevent an unauthorized user from accessing saved passwords.

The next objective was to design the Portable Password Bank so that user could retrieve a stored password at any time. The microcontroller needed to be able to read/write to the EEPROM at any time. The software programmed on the microcontroller would read from the EEPROM and know which password belonged to the button desired by the user. Again, since there were only four buttons on the Password Bank, the device could only store four passwords.

The last objective was to create a master lock to prevent anyone from using the device except the owner of the device. As mentioned above, a key combination was used to unlock the device initially. There was also a blinking red LED indicating if the master lock was entered correctly. The red LED would blink three times if the master lock combination was entered incorrectly. However, if the combination was correct, the red LED would blink and stay lit for approximately three seconds and then turn off.
Methodology/Technical Approach

To successfully achieve all the project goals, each objective had to be individually developed in order to verify proper operation and to allow for testing of each component, whether it be software or hardware. After finding a suitable development board, the next objective became developing the software for the device itself while also allowing us to test it at any point. Upon interfacing to the computer, the device was controlled by hardware and we later developed the Graphical User Interface to allow for better debugging and a visual representation of where the user was in the device’s menu.

Before any designing began, we studied a lot of examples that showed how one could interface a microcontroller through USB. We also consulted with professors to verify that this project was feasible. In addition to this, we took some time to decide what the best qualities or features in such a project would be. After careful deliberation, we concluded that this project was indeed feasible and that the best attributes for a user would be security, convenience, portability, universality, and low cost. Through further discussions with our advisor and other professors, our main objectives were defined and construction began.

Using open source code and examples from Atmel, we learned how to emulate a keyboard. We used given open source code that detailed proper timing and the necessary communication standards used to implement the USB communication. After successfully emulating a keyboard and having proper setup to communicate through USB, we were able to connect the device to a computer and the computer would recognize it as a keyboard. Figure 4 shows the completed schematic of our device.
Once that was complete, we used examples and developed code that allowed us to print a character to a text field in the computer. Once that was finished, the next step was to add additional push buttons to allow us the capability of navigating through menus we would later build into the project. While doing this, we created additional code for our device that would allow us to select a password. To do this we assigned each of the push buttons a memory location in the EEPROM. Once editing a password, the design we used made the user select one of the password slots, which would notify our program where in memory this password would be stored and later recalled from.

When the menu built into the microcontroller was partially complete, we started developing the “PPB Config”, the Graphical User Interface for the project. Developing this alongside with the Password Bank’s menu allowed us to make on the fly design decisions and quickly set up our menu. Figure 5 shows the Main Menu of the device after unlocking it.
In order for the microcontroller to read from the EEPROM, it was necessary for the microcontroller’s software to call a C program function. The microcontroller needed to be able to read/write to the EEPROM at any time. Every time that a button was held (corresponding to the chosen password) and the password button pressed, the software would call the “read from EEPROM” function and paste the saved password to the user’s login screen. Using the correct timing for USB, the text string was sent through the device’s USB cable to the computer.

The master lock was created by first defining the correct key-combination in the software. We also created a way for the user to change the master lock at any time by first typing in the old master lock and then typing a new one. As shown in Figure 5, the user can select option 2 to change the master lock. Once the user types in the new master lock, the key combination is saved to the EEPROM and this lock can be used even after unplugging the device and plugging it into another device. We wanted a way to show the user if the master lock was entered correctly. In order for this to function, we added a second circuit containing an LED, transistor, and a resistor. We took a wire from the transistor to a port pin on the microcontroller. We were able to turn off/on the light by the C code written for the microcontroller. As mentioned above, if the LED flashes three times, this indicates to the user that the master lock was not entered properly. However, when the LED turns on and stays on for three seconds, then turns off, this indicates that the master lock was entered correctly. A power light (LED) was also added to the device to show the user that the device is getting the 5 volts it needs from the computer through the USB connection. See Figure 6 for an image of the device with its indicator lights.

**Budget**
Table 1 shows an accurate account of the final budget for this project. The name of each component in the left column and its associated cost is directly across. As shown, the most costly component was the Atmel microcontroller development kit, as seen in Figure 3. This was because the development kit came with extra components that were not needed in the project. These components include: RS-232 port, alternate power supply, and misc. small electronic surface parts.
<table>
<thead>
<tr>
<th>Component</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT89STK-05 Development kit</td>
<td>$224.00</td>
</tr>
<tr>
<td>Microsoft Visual C# 2010 Express</td>
<td>Free</td>
</tr>
<tr>
<td>LEDs</td>
<td>$2.00</td>
</tr>
<tr>
<td>Push Buttons</td>
<td>$7.00</td>
</tr>
<tr>
<td>Case</td>
<td>$5.00</td>
</tr>
<tr>
<td>Miscellaneous Parts</td>
<td>$4.00</td>
</tr>
<tr>
<td><strong>Total Cost</strong></td>
<td><strong>$242.00</strong></td>
</tr>
</tbody>
</table>

Table 1: Final Budget

**Timeline**

Figure 7 shows an accurate timeline for all the work performed. Each task relied on the completion of another in order to be integrated properly. It took longer than expected for all of the software to be completed. It also took longer to test/debug the Portable Password Bank. When the project was tested, another flaw was detected and needed to be corrected. In the end, however, the project was completed on time. The Password Bank, by design, also has room to be expanded. It seemed that regulating workloads and assigning specific tasks between both of us allowed our project to be completed on time.

![Figure 7: Final Schedule](image)

**Problems Encountered/Analysis of Problems Solved**

This project was mostly successful but not without difficulties. The item that became the most challenging problem dealt with the memory management of the Password Bank. Another unexpected issue was the use of numerous faulty push buttons. Thanks to examples of memory management from Atmel and open source code as well as Dave Tashjian for his assistance in inspecting the push buttons, we were able to overcome these issues.

First, only having an understanding of memory management in Assembly language, we had relatively no idea how to read from and write to the EEPROM. Thanks to Atmel’s sample code, some open source code and many hours of trial and error, we gained enough of an understanding to gain the ability to store passwords to the EEPROM and
call them when needed. This allowed us to keep the passwords saved upon disconnecting
the power.

The second, and most frustrating, issue was faulty push buttons. We initially programmed
our device and tested the push buttons and they worked fine. A few weeks into the
project, they stopped working. We spent nearly two weeks troubleshooting this issue. In
this time, we checked our code, purchased new push buttons, testing and using those,
having those fail as well, and verifying that the current and voltage going through the
push buttons were within proper limits before we found out that it was indeed just the
push buttons that were the problem. In total, we had 14 faulty push buttons. This was due
to using push buttons from a store that was notorious for faulty parts and push buttons
that were left in a dry basement for a few years. After finally solving this problem, we
were able to complete our project according to our schedule.

**Future Recommendations**

One of the design decisions was to use four push buttons, one for each password. Using
this method allowed us to complete our project on time but limits the user to storing only
four passwords. Also, due to the time and budget constraints, we were unable to
incorporate the fingerprint reader as the master lock.

Having only a minimal understanding of how to manage the memory of an embedded
device in C and time constraints, we were unable to incorporate additional password
storage. All that would be required to implement this would be the use of “Up/Down”
arrows and an understanding of how to cycle through memory locations corresponding to
the button presses.

The incorporation of the fingerprint reader would require more funding and time to
develop. Additional time and code would be required to program the embedded
fingerprint reader. To successfully implement this, one would need additional funding,
and about a month or two of time to interface the embedded fingerprint reader to the
microcontroller. Using this, would allow the user to have a more convenient and secure
way of accessing the Portable Password Bank.
CONCLUSION

This project turned out to be a practical solution for people using several passwords for their logins. There are passwords for email accounts, for logging into computers, and for accessing high security databases. It can be difficult for a user to remember a complex password for each application. By using the Portable Password Bank, people have the ability to use complex passwords without the need to remember them. This could be used by anyone with a Windows machine that has a USB port. For optimal security, there should be a complex password for each application and login screen. The Portable Password Bank’s master lock gives the user insurance that the device will be protected when it is not in use.

This project has driven both of us from the beginning. We were able to take our knowledge from our Embedded Systems courses and programming courses, all taken at the University of Cincinnati, and apply it towards our project. We were able to create a design product that is not only practical, but that is also easy to use, portable, universal, and secure. The Senior Design project has proven how much we have learned and how our skills advanced through our education. Through hard work, long nights, research and determination, we were able to construct the Portable Password Bank. In summary, we are both certain, with our current abilities, that we can accomplish any future projects given to us accurately and efficiently.
REFERENCE LIST


APENDIX A: AT89C5131A Hardware Guide

AT89C5131A Starter Kit
Hardware User Guide
## Table of Contents

### Section 1
- Introduction .................................................................................. 1-1
  - 1.1 Features .................................................................................. 1-1

### Section 2
- Hardware Description .................................................................. 2-3
  - 2.1 Block Diagram .......................................................................... 2-3
  - 2.2 Power Supply ........................................................................... 2-4
  - 2.3 C51 Standard Settings ............................................................... 2-5
  - 2.4 Feature Description ................................................................. 2-6
  - 2.5 External Connectors ................................................................. 2-8

### Section 3
- Device Programming ........................................................................ 3-9
  - 3.1 In-System Programming .......................................................... 3-9
  - 3.2 Using a Programmer ............................................................... 3-9

### Section 4
- Appendix ....................................................................................... 4-11
  - 4.1 Electrical Schematics ............................................................. 4-12
  - 4.2 Component Placement .......................................................... 4-16
  - 4.3 Mechanical Outlines .............................................................. 4-17
  - 4.4 Bill of Materials ...................................................................... 4-17
Section 1

Introduction

This document describes the AT89C5131A Starter Kit Evaluation Board dedicated to the AT89C5131A USB microcontroller. This board is designed to allow an easy evaluation of the product using demonstration software (refer to Software Guide).

1.1 Features

The AT89C5131A evaluation board provides the following features:

- Possibility to choose between two packages for the AT89C5131A
  - PLCC 52-pin package
  - VQFP 64-pin package
- On-board power supply circuitry
  - from an external power connector
  - from an external battery
  - from the USB line via the USB on-board connector
- On-board reset, INTO, LEDs, EA, ISP and programming interface
- Power, ALE, RS232 Rx and Tx LEDs
- External system clock connector
- PCA clock connector
- USB, TWI, SPI and RS232 hardware connectors
- Two Connectors available for extended board
Figure 1-1. AT89C5131A Evaluation Board
2.1 Block Diagram

Figure 2-1. AT89C5131A Evaluation Board Components
2.2 **Power Supply**

The on-board power supply circuitry allows various power supply configurations.

The power source can be:
- $V_{BUS}$ from USB (5V)
- $V_{BUS}$ from USB (5V) through the current limiter
- External power supply (from 6 to 12V) or 9V battery

The voltage output can be the direct power source, regulated at 5V or 3.3V.

The power supply selection is performed using the JP2, JP3, JP4 and JP5 jumpers.

The power supply can be turned on/off using the "power" switch (SW6). Once the power is established, the power LED (DS) is lit.

**Figure 2-2. Different Power Configurations**

<table>
<thead>
<tr>
<th>Power Source Regulation</th>
<th>VBUS</th>
<th>VBUS and Current Limiter</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Input</td>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
<tr>
<td>5V Regulate</td>
<td><img src="image4" alt="Diagram" /></td>
<td><img src="image5" alt="Diagram" /></td>
<td><img src="image6" alt="Diagram" /></td>
</tr>
<tr>
<td>3.3V Regulate</td>
<td><img src="image7" alt="Diagram" /></td>
<td><img src="image8" alt="Diagram" /></td>
<td><img src="image9" alt="Diagram" /></td>
</tr>
</tbody>
</table>
2.3 C51 Standard Settings

2.3.1 Reset 
The external Reset push-button (SW3) is provided to easily generate a warm reset. This button is used for ISP process. The Reset applied is active low.

2.3.2 Clock 
A crystal can be easily installed on the Y1 socket. The clock can also be provided using the J8 connector instead of the crystal.

Note: Remove the clock generators before the using the programmer.

2.3.3 EA 
Place a jumper on the EA connector (J10) to force the EA pin to ground and execute external code. Otherwise internal code will be executed.

Figure 2-3. EA Circuitry

2.3.4 INT0 
In order to use the on-board INT0 circuitry, connect the J7 Jumper to the AT89C5131A. When you press the INT0 button (SW5), the P3.2 pin will go low which induces an interrupt event.

Note: Remove the J7 jumper before using the programmer. Otherwise the programmer will not function.

Figure 2-4. INT0 Circuitry
2.4 Feature Description

2.4.1 RS232
The AT89C5131A evaluation board includes all the required hardware to manage the RS232 communication.

![RS232 Circuitry Diagram]

2.4.2 USB Peripheral
The AT89C5131A evaluation board provides all the required hardware to develop a USB firmware for the AT89C5131A, this includes:
- a USB connector
- 2 test points on D+ and D-
- 1 test point on Vin
- a USB UNLOAD button which allows to disconnect the pull-up on D+ and then to simulate an Attach/Detach of the USB cable

The USB peripheral can also be used to perform an In-System Programming.
2.4.3 TWI Peripheral

The CT3 and CT5 contacts have to be soldered in order to use the SDA and SCL alternate P4.1 and P4.0 port configuration on the SPI connector (J4). In order to use these signals on the J5 extension connector (SDA and SCL), the CT4 and CT6 contacts have to also be soldered.

2.4.4 SPI Peripheral

2.4.5 LED Controller

The AT89C5131A controller includes an LED controller on:
- P3.3 (LED 0)
- P3.5 (LED 1)
- P3.6 (LED 2)
- P3.7 (LED 3)

The on board LEDs can be controlled with the AT89C5131A if the corresponding contacts CT9, CT10, CT11 and CT12 are bypassed.

Figure 2-6. On-board LEDs for LED Controller
2.5 External Connectors

These two external connectors to build a customer extended board easily.

Figure 2-7. Top View of J5 and J6 Connectors

<table>
<thead>
<tr>
<th>J5</th>
<th>J6</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.1</td>
<td>AT</td>
</tr>
<tr>
<td>P1.2</td>
<td>13</td>
</tr>
<tr>
<td>P1.3</td>
<td>14</td>
</tr>
<tr>
<td>P1.4</td>
<td>15</td>
</tr>
<tr>
<td>P1.5</td>
<td>16</td>
</tr>
<tr>
<td>P1.6</td>
<td>17</td>
</tr>
<tr>
<td>P1.7</td>
<td>18</td>
</tr>
<tr>
<td>P1.8</td>
<td>19</td>
</tr>
<tr>
<td>P1.9</td>
<td>20</td>
</tr>
<tr>
<td>P1.10</td>
<td>21</td>
</tr>
<tr>
<td>P1.11</td>
<td>22</td>
</tr>
<tr>
<td>P1.12</td>
<td>23</td>
</tr>
<tr>
<td>P1.13</td>
<td>24</td>
</tr>
<tr>
<td>P1.14</td>
<td>25</td>
</tr>
<tr>
<td>P1.15</td>
<td>26</td>
</tr>
<tr>
<td>P2.16</td>
<td>20</td>
</tr>
<tr>
<td>P3.1.17</td>
<td>21</td>
</tr>
<tr>
<td>P3.1.18</td>
<td>22</td>
</tr>
<tr>
<td>P3.1.19</td>
<td>23</td>
</tr>
<tr>
<td>P3.1.20</td>
<td>24</td>
</tr>
<tr>
<td>P3.1.21</td>
<td>25</td>
</tr>
<tr>
<td>P3.1.22</td>
<td>26</td>
</tr>
<tr>
<td>P3.1.23</td>
<td>27</td>
</tr>
<tr>
<td>P3.1.24</td>
<td>28</td>
</tr>
<tr>
<td>P3.1.25</td>
<td>29</td>
</tr>
<tr>
<td>P3.1.26</td>
<td>30</td>
</tr>
<tr>
<td>P2.1.30</td>
<td>30</td>
</tr>
<tr>
<td>P1.2.30</td>
<td>30</td>
</tr>
<tr>
<td>P1.3.30</td>
<td>30</td>
</tr>
<tr>
<td>P1.4.30</td>
<td>30</td>
</tr>
<tr>
<td>P1.5.30</td>
<td>30</td>
</tr>
<tr>
<td>P1.6.30</td>
<td>30</td>
</tr>
<tr>
<td>P1.7.30</td>
<td>30</td>
</tr>
</tbody>
</table>
Section 3

Device Programming

3.1 In-System Programming
The user memory of the AT89C5131A part can be programmed using the ISP mode of
the device. In order to enter in ISP mode, first select the high pin count mode (PSEN) or
the low pin count mode (P1.0) using the ISP switch (SW2).

To enter in ISP mode, press both the RESET (SW3) and ISP (SW4) buttons simultaneously. First release the RESET button and then the ISP button. The device enters in
ISP mode.

ISP can then be performed using the USB bus (or with the peripheral corresponding
with the bootloader version). The user may need to re-enumerate the USB bus using the
USB UNLOAD button (SW1) if the USB cable is already connected.

3.2 Using a Programmer
The AT89C5131A microcontroller can also be programmed using a programmer with
the J3 connector. Connect all required signals between the programmer and the J3 con-
nector and remove the J7 jumper to disconnect the EA circuitry. No clock should be
enabled on the board, except the clock coming from the J3 connector.

Figure 3-1. J3 Connector Schematic

![Diagram of J3 connector]
4.2 Component Placement

Figure 4-1. AT89CS131A Evaluation Board Overview

Figure 4-2. AT89CS131A Evaluation Board Component Implementation
Appendix

4.3 Mechanical Outlines

Figure 4-3. AT89C5131A Evaluation Board Mechanical Outlines

<table>
<thead>
<tr>
<th>LEFT</th>
<th>TOP</th>
<th>RIGHT</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.35 mm</td>
<td>2.54 mm</td>
<td>5.08 mm</td>
</tr>
</tbody>
</table>

99.08 mm

93.98 mm (37*2.54 mm)

Top view

C51 Generic Board's Left connector

C51 Generic Board's Right connector

7.00 mm

2.54 mm

BOTTOM

4.4 Bill of Materials

Table 4-1. Bill of Materials

<table>
<thead>
<tr>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>C11, C12</td>
<td>22 pF</td>
</tr>
<tr>
<td>C2</td>
<td>2.2 nF</td>
</tr>
<tr>
<td>C1, C10, C14</td>
<td>10 nF</td>
</tr>
<tr>
<td>C9, C13</td>
<td>100 nF</td>
</tr>
<tr>
<td>C3, C4, C5, C6, C7, C8, C15, C17, C20, C21</td>
<td>0.1 µF</td>
</tr>
<tr>
<td>C16, C19</td>
<td>10 µF</td>
</tr>
<tr>
<td>R4, R5</td>
<td>27</td>
</tr>
<tr>
<td>R3</td>
<td>100</td>
</tr>
<tr>
<td>R15</td>
<td>180</td>
</tr>
<tr>
<td>R1, R6, R13, R19</td>
<td>1K</td>
</tr>
<tr>
<td>R7, R8, R9</td>
<td>4.7K</td>
</tr>
<tr>
<td>R11, R12, R14</td>
<td>10K</td>
</tr>
</tbody>
</table>
Table 4-1. Bill of Materials  (Continued)

<table>
<thead>
<tr>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>D2</td>
<td>LED GREEN</td>
</tr>
<tr>
<td>D1, D7, D9</td>
<td>LEDs RED</td>
</tr>
<tr>
<td>D3, D4, D5, D6, D8</td>
<td>LEDs PWR GREEN</td>
</tr>
<tr>
<td>D8</td>
<td>MRA4007</td>
</tr>
<tr>
<td>D11</td>
<td>SMBJ9.0A</td>
</tr>
<tr>
<td>U1</td>
<td>MAX202ECSE</td>
</tr>
<tr>
<td>U8</td>
<td>DF005S</td>
</tr>
<tr>
<td>TP1, TP2, TP3, TP4, TP5, TP6</td>
<td>TEST POINTS</td>
</tr>
<tr>
<td>J7, J10, J12</td>
<td>JUMPER</td>
</tr>
<tr>
<td>J13</td>
<td>CONNECTOR JACK PWR</td>
</tr>
<tr>
<td>J8, J9</td>
<td>CONNECTORS BNC</td>
</tr>
<tr>
<td>P1</td>
<td>SUB-D9 FEMALE</td>
</tr>
<tr>
<td>J1</td>
<td>USB B</td>
</tr>
<tr>
<td>J6, J5</td>
<td>HEADER 24X2</td>
</tr>
<tr>
<td>J11</td>
<td>CONNECTOR SIP2</td>
</tr>
<tr>
<td>J4</td>
<td>CONNECTOR SIP4 RA</td>
</tr>
<tr>
<td>J2</td>
<td>CONNECTOR SIP8 RA</td>
</tr>
<tr>
<td>J3</td>
<td>CONNECTOR HE10</td>
</tr>
<tr>
<td>SW2, SW6</td>
<td>SW KEY SPDT</td>
</tr>
<tr>
<td>U3</td>
<td>AT80C5131A_52</td>
</tr>
<tr>
<td>U5</td>
<td>AT89C5131A_VQFP64</td>
</tr>
<tr>
<td>CT1, CT2, CT3, CT4, CT5, CT6, CT7, CT8, CT9, CT10, CT11, CT12, CT13, CT14</td>
<td>CONTACT</td>
</tr>
<tr>
<td>C18</td>
<td>10 μF TANTAL</td>
</tr>
<tr>
<td>D10</td>
<td>1N4002</td>
</tr>
<tr>
<td>JP1</td>
<td>A16_Buzz Jumper</td>
</tr>
<tr>
<td>JP2</td>
<td>Limiter Jumper</td>
</tr>
<tr>
<td>JP3</td>
<td>Power Source Jumper</td>
</tr>
<tr>
<td>JP4</td>
<td>Regulator Jumper</td>
</tr>
<tr>
<td>JP5</td>
<td>VCC Level jumper</td>
</tr>
<tr>
<td>R20</td>
<td>121.1%</td>
</tr>
<tr>
<td>R21</td>
<td>365.1%</td>
</tr>
<tr>
<td>R22</td>
<td>196.1%</td>
</tr>
<tr>
<td>SW1</td>
<td>CONTACT BREAKER</td>
</tr>
<tr>
<td>Reference</td>
<td>Part</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
</tr>
<tr>
<td>SW3, SW4, SW5</td>
<td>PUSH-BUTTON</td>
</tr>
<tr>
<td>U2</td>
<td>MAX706CSA</td>
</tr>
<tr>
<td>U6</td>
<td>TPS2041AD</td>
</tr>
<tr>
<td>U9</td>
<td>LM1084/TO263</td>
</tr>
<tr>
<td>Y1</td>
<td>CRYSTAL</td>
</tr>
</tbody>
</table>
APENDIX B: AT89C5131A Software Guide

AT89C5131A Starter Kit
Software User Guide
# Table of Contents

## Section 1
Introduction ................................................................. 1-1
  1.1 Abbreviations ....................................................... 1-1

## Section 2
Getting Started ............................................................... 2-3
  2.1 Hardware Requirements ........................................... 2-3
  2.2 Software Requirements ........................................... 2-3
  2.3 Default Hardware Settings ......................................... 2-3
  2.4 FLIP Software ....................................................... 2-4

## Section 3
HID Keyboard Demonstration Program ................................. 3-7
  3.1 Stand-alone Application ........................................... 3-7
  3.2 C51 Generic Board Application ................................. 3-7

## Section 4
Software Description ...................................................... 3-9
  3.1 Architecture Overview ........................................... 3-9
  3.2 Application Description .......................................... 3-10
  3.3 Libraries Description ............................................. 3-11
Section 1

Introduction

This document describes the AT89C5131A HID keyboard demonstration application, as well as the In-System Programming Tool (FLIP), dedicated to the AT89C5131A microcontroller. This software demonstration is an implementation example of a HID keyboard which is USB Chapter 9 compliant.

1.1 Abbreviations

- USB: Universal Serial Bus
- HID: Human Interface Device
- ISP: In-System Programming
- DFU: Device Firmware Upgrade
Section 2
Getting Started

2.1 Hardware Requirements
The demonstration application requires the following hardware:
- AT89C5131A evaluation board
- AT89C5131A microcontroller (includes a USB bootloader)
- A-B USB cable
- PC running Windows® (98, Me, 2000 or XP) or Linux® with a 1.1 or 2.0 USB Host

2.2 Software Requirements
The following software is necessary to use the demonstration program. The software can be found in the accompanying CD-ROM. Updated FLIP software is available on the Atmel web site.
- Flexible In-System Programming (FLIP) software tool
- usb_hid_kbd.hex file

2.3 Default Hardware Settings
The AT89C5131A evaluation board must be configured as follows:
- Power supply: from VBUS through the limiter and the 3.3V regulator.
- Quartz frequency: 16 MHz
- EA jumper: Off
- INT0 jumper: On

2.3.1 Setting ISP Mode
1. When BLJB is set (= 0) (default setting), the ISP mode is enabled. Plug the USB cable into the PC. The bootloader will automatically execute.
2. When the BLJB is reset (= 1), the ISP is not set. The following procedure must be executed:
   - Press the ISP button (SW4) and plug the USB cable into your PC, or
   - Press both the Reset (SW3) and ISP (SW4) buttons. First release the Reset button and then the ISP button. The device enters ISP mode. The user needs to re-enumerate the device using the USB Unload button (SW1).
2.4 FLIP Software

FLIP software runs on Windows® (98/2000, NT, XP). FLIP supports In-System programming of Flash CS1 devices through RS-232. The latest version of FLIP software can be found on the Atmel web site. A Linux version of FLIP is also available.

Note: See FLIP User documentation for USB Driver Installation Procedure.

2.4.1 Flash Programming

The HID keyboard hex file (usb_hid_kbd.hex) must be programmed in the user Flash memory of the AT89C5131A. There are two different ways to perform the Flash programming:

- Use a programmer which supports the AT89C5131A part
- Use the Atmel FLIP tool (on a PC connected to the evaluation board using a USB cable)

In this section the user will program the AT89C5131A microcontroller via USB using FLIP software. The following procedure will guide you through the programming of the demonstration program.

1. Run FLIP software (see Figure 2.2)
2. From the Device Menu, choose "Select" and select the device (AT80C5131A) that is connected to the evaluation board.
3. Click the “Set Communication” button.

4. Initialize the communication by selecting the “Open” button in the USB Port Connection pop-up window. If the connection is successful, the FLIP window should look like Figure 2-3.
5. In the File menu, select “Load HEX” and choose the demonstration program “usb_kbd.hex”.

6. The message “HEX file usb_kbd.hex loading done” is displayed at the bottom of the FLIP window.

7. Ensure the following check boxes are selected in the Operations Flow section of FLIP:
   - Erase
   - Blank Check
   - Program
   - Verify

These are the operations that will be performed on the microcontroller.

8. Press the “Run” button. Programming is executed. The “Memory Verify Pass” message confirms programming is successful and that the microcontroller has been programmed.

9. Ensure the BLJB box is unchecked. Press “Set”, then “Read” to verify that the BLJB is blank (=1), in order to boot the demonstration program after the next reset.

10. Ensure the “With Reset” box is checked, then press the “Start Application” button.
Section 3

HID Keyboard Demonstration Program

The purpose of the HID Keyboard demonstration program is to send numeric data via the keypad through the AT89C5131A microcontroller to a host PC via USB.

The HID Keyboard demonstration program is used with the AT89C5131A board (Stand-alone Application).

3.1 Stand-alone Application

The AT89C5131A board can be used to transmit a message stored in the MCUs Flash memory and display the message on a PC text editor.

This demonstrates the AT89C5131A microcontrollers “Plug & Play” and “Hotplug” capability for any USB application.

1. Ensure the USB cable is connected between the AT89C5131A evaluation board and the PC.
2. Open the Notepad application or any text editor on the PC.
3. Click the INTO button of the AT89C5131A evaluation board. The message “Welcome to the HID keyboard” is displayed on the text editor.

3.2 C51 Generic Board Application

1. Plug the USB cable into the USB host socket on one end and to the AT89C5131A evaluation board on the other end.
   The message “AT89C5131A demo” appears on the first line of the LCD. The LED 0 of the AT89C5131A evaluation board blinks. This indicates that the USB cable is correctly connected and the SOF messages are correctly sent by the Host controller and understood by the AT89C5131A microcontroller.
2. The PC Operating System may ask for a driver, specify the proper directory as indicated in the Section “Note on Windows Drivers” below. Once the correct driver is loaded, the PC sends a “SET CONFIGURATION” USB message and the “Enumeration Pass” message appears on the third line of the LCD.
3. Open a calculator or a text editor. When the buttons of the C51 generic board keypad are keyed, the number or the operation is displayed in the text editor or calculator.
HID Keyboard Demonstration Program

Notes: 1. Ensure the Num Lock key is activated on your PC keyboard number pad. The demonstration program is supporting QWERTY keyboard configuration. All other configuration (such as AZERTY) will result in wrong display of some characters.

3.2.1 Note on Windows Drivers

The HID keyboard example can directly interface with native drivers under Windows 98, Me, 2000 and XP. After initial USB connection, Windows may ask for drivers. Indicate the following path:

- `<Windows hard drive>\WINDOWS\inf` (for Windows 98 and Me)
- `<Windows hard drive>\WINNT\inf` (for Windows 2000 or XP)
Section 4
Software Architecture

4.1 Architecture Overview

The HID keyboard demonstration firmware is based on a scheduler in the free running mode.

The main program only enables the interrupts and launches the scheduler.

The first process of the scheduler is the initialization of all the peripherals and of the associated variables.

Once the initialization process is complete, the scheduler launches each task one after the other. The first task is the USB task. This task manages the Default Control Endpoint for the enumeration process and the HID keyboard control.

The second task called by the scheduler is the USB LCD task that manages all the LCD display in accordance with the USB bus status (connected or not, enumeration process passed or not, etc.).

The third and last task called by the scheduler is the USB KBD task that transmits the keys pressed to the USB controller.
4.2 Application Description

4.2.1 Configuration
The USB HID keyboard configuration is performed according to the “USB Device Class Definition for Human Interface Device – Firmware Specification” version 1.1 (4/7/99).

4.2.2 Implementation

4.2.2.1 usb_task_init()
The `usb_task_init()` function is called during the initialization phase. It enables the USB controller, configures the PLL in order to generate the 48 MHz required by the USB controller, and enables and configures the Endpoint 0. In addition, this function performs a USB Detach/Attach in order to be re-enumerated by the Host. This could be necessary after a Start Application is performed by the bootloader.

This function also initializes the LED controller of the AT89C5131A part.

4.2.2.2 usb_task()
The `usb_task` manages the USB events: Suspend, Resume, USB Reset and Start of frame. When the USB bus is in Suspend state, the LED 3 is turned On. If a USB Resume occurs, the firmware turns Off this LED.

Each time a Start of Frame occurs on the USB bus, a counter is increased. When this counter reaches 255, this counter is reset and the LED 1 is toggled.

Figure 4-1. USB HID Keyboard Firmware Architecture Overview
When a SETUP token is detected on the Endpoint 0, the usb_task launches the enumeration process routine. Once the Control Transaction has been completed, the enumeration process routine exits.

A Transmit Complete flag (TXCMPL) detection on the Endpoint 1 (IN endpoint for HID keyboard) means that a HID report has been successfully transmitted to the Host. The usb_task then clears the Transmit Complete flag in order for the USB keyboard application to send the next HID report.

4.2.2.3 usb_lcd_task() The usb_lcd_task() displays the status of the USB on the LCD:
- USB Connected
- USB Suspend
- Enumeration Process Passed

4.2.2.4 usb_kbd_task() This usb_kbd_task() determines if a new report has to be sent to the Host in function of the keyboard scan. This function is also in charge of translating the keyboard scan result into HID comprehensive bytes. The key codes sent correspond to the USB HID Usage Tables document for a QWERTY keyboard.

4.3 Libraries Description

4.3.1 USB The USB management uses two different libraries:
- One for the USB enumeration process
- One for the low level of the USB controller

The enumeration process management is contained in the usb_hid_enum.c and usb_hid_enum.h files. As it is written in the file names, this enumeration process is specific for this application because some HID specific messages require the default control endpoint. However, this enumeration process management can easily be adapted for other applications.

The low level library gives an easy and comprehensive access to the USB controller. This library allows to manage USB events (USB Reset, Suspend/Resume, Start Of Frame), to configure the USB controller and the endpoints, and to send or receive messages over each endpoint. These drivers are contained in the usbDrv.c and usbDrv.h files.

4.3.2 LED The 5131_drv.h file includes the on-chip LED controller driver. This library allows to configure the LED controller, to turn On/Off or to toggle each or every LED.
4.3.3 Keyboard

The keyboard management example has been written for a 4 x 4 keyboard.