Lockitall LockIT Pro

User Guide

Revision 3

FakePublisher, Nowhere, DNE.

First edition: 20 November 2008
Second edition: 5 January 2010
Third edition: 1 July 2013

10 09 08 07 06 05 04 03 02 01 03 04 05 06 07

Never printed on paper.

Preface

About This Manual

This manual is provided along with your LockIT Pro order. It is written to allow customers to understand how their lock functions, as well as to allow them to add new features to their lock as they desire. We describe features of the MSP430, the MCU on which the LockIT Pro runs, as well as additions we have made to the MCU. We further describe the Hardware Security Modules sent along with this shipment.

FCC Warning

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct this interference.

Contents

1 Introduction

The LockIT Pro is the first of a new series of locks. It is controlled by a MSP430 microcontroller, and is the most advanced MCU-controlled lock available on the market. The MSP430 is a very low-power device which allows the LockIT Pro to run in almost any environment.

The LockIT Pro contains a Bluetooth chip allowing it to communiciate with the LockIT Pro iPhone App, allowing the LockIT Pro to be inaccessable from the exterior of the building.

There is no default password on the LockIT Pro—upon receiving the LockIT Pro, a new password must be set by connecting it to the LockIT Pro iPhone App and entering a password when prompted, and then restarting the LockIT Pro using the red button on the back.

There are several additional attachments to the LockIT Pro which are described in this document: two different Hardware Security Modules, the LockIT Pro Deadbolt, and the LockIT Pro Bluetooth Connector.

2

Overview of the LockIT Pro

2.1 The MSP430

The LockIT Pro is controlled using a modified TI MSP430. The TI MSP430 is one of the world's most popular MCUs. They're extremely inexpensive and can be powered off a lemon, which allows customers to deploy the LockIT Pro in any location without concern for power.

The LockIT Pro comes preprogrammed, but can be reprogrammed, allowing customers to load whatever software they wish on the lock.

The MSP430 is a conventional 16 bit register RISC architecture with a RAM-based stack, 16 registers, and a simple memory map.

A brief introduction to MSP430 assembly is given in Section

2.2 Software Interrupts

Lockitall has extended the MSP430 to support software interrupts, implemented with a callgate at address 0x0010 on the MCU. When the CPU executes the instruction at this address, the CPU begins executing special-purpose LockIT Pro code to perform special-purpose functions, such as printing characters to the console, or requesting terminal input. The interrupt kind is passed in R2, the status register, on the high byte. Arguments are passed on the stack.

The inturrupts are described in detail in Section 4.

2.3 Memory Protection

Some version of the LockIT Pro contain memory protection which allows each of the 256 pages to be either executable or writable, but never both. This prevents many common attacks. There is an interrupt for the LockIT Pro which enables memory protection, and there are interrupts to specify whether a given page should be executable or writable.

3

LockIT Pro External Hardware

3.1 The Door Lock

While the LockIT Pro contains the circuitry to function, it does not contain a physical lock. The door lock must be attached to output pin 7 of the MCU. This enables the CPU to trigger software interrupt 0x7F to directly trigger the door lock to unlock.

The door lock automatically relocks after the door has been opened, no further commands must be sent to it.

3.2 HSM Model 1

Lockitall includes several hardware security module interfaces along with the LockIT Pro . The first of these is the Model 1.

The Model 1 of the hardware security module contains a simple interface which allows the MCU to test if an entered password is valid. By default, the interrupt 0x7D will pass a given password to the HSM, and will set a byte in memory if the password entered matches the stored password.

The stored password can be reset by detaching the HSM from the lock and attaching it to the Model 1 reset device, also included.

3.3 HSM Model 2

The Model 2 of the hardware security module is a more advanced HSM, with the ability to directly trigger the unlock functionality in the lock. The MCU passes the lock a password, and the HSM will trigger the unlock if the password is valid. By default, the interrupt 0x7E will pass a given password to the HSM, and the lock will be opened if the password entered matches the stored password.

The stored password can be reset by detaching the HSM from the lock and attaching it to the Model 2 reset device, also included.

4

Developing for the LockIT Pro

This section provides references for developing for the LockIT Pro . We describe the build process for developing C code to execute on the lock. We have built a small C standard library, which is documented below.

4.1 Installation

Lockitall recommends the msp-gcc compiler for development. It can be installed by running "sudo apt-get install gcc-msp430" or "sudo yum install msp430-gcc".

Developing for the MSP is then straightforward. Listing 1 contains a sample method of unlocking a lock conditioned on a fixed password in memory of the lock.

```
#include <io.c>
#include <lib.c>
int main(void) {
  char* pw = ``password'';
  char buf[16];
  char* ptr = buf;
  puts(``What is the password?'');
  gets(buf, 15);
  while (*pw && *buf)
    if (*pw++ != *buf++) return 1;
```

```
  if (*pw != *buf) return 1;
  INT(0x7F);
  return 0;
}
```

4.2 The C Standard Library

The USB key contained with this shipment contains the object files. We document the methods here.

INT

Declaration:

```
void INT(int arg, ...);
```

Trigger interrupt given by `arg`, passing additional arguments.

Range:

The value of `arg` must be within the range of 0 to 0x7F (inclusive).

gets

Declaration:

```
void gets(char* buf, unsigned int length);
```

Copies at most `length` (including the terminating null byte) bytes from the input device into `buf`.

Range:

The buffer `buf` must contain at least `length` bytes of space.

putchar

Declaration:

```
int putchar(int char);
```

Writes the character `char` to the terminal display attached to the lock. Returns the character printed, or -1 if an error occurred.

getchar

Declaration:

```
int getchar();
```

Gets a character from the input device.
Returns the character received, or -1 if no character is available.

puts

Declaration:

```
void puts(char* buf);
```

Writes the characters in `buf` one by one until a zero byte is reached.

printf

Declaration:

```
void printf(char* str, ...);
```

Prints formatted output to the console. The string `str` is printed as in `puts` except for conversion specifiers. Conversion specifiers begin with the `%` character.

Conversion Character	Output
s	The argument is of type `char*` and points to a string.
x	The argument is an `unsigned int` to be printed in base 16.
c	The argument is of type `char` to be printed as a character.
n	The argumentis is of type `unsigned int*`. Saves the number of characters printed thus far. No output is produced.

4.3 Interrupt Listing

The LockIT Pro has an augmented MSP430 CPU with a callgate at address 0x10 causing a software interrupt. The interrupts are described below.

INT 0x00.

The `putchar` interrupt: sends a single byte to the display.

Takes one argument with the character to print.

INT 0x01.

The `getchar` interrupt: reads a single byte of buffered input.

Takes no arguments.

INT 0x02.

The `gets` interrupt: read a specific number of bytes to standard input.

Takes two arguments. The first is the address to place the string, the second is the maximum number of bytes to read. Null bytes are not handled specially null-terminated.

INT 0x10.

Turn on DEP: pages are either executable or writable but never both.

Takes no arguments.

INT 0x11.

Mark as a page as either only executable or only writable.

Takes two one arguments. The first argument is the page number, the second argument is 1 if writable, 0 if executable.

INT 0x20.

The `rand` interrupt: request a random 16-bit number.

Takes no arguments.

INT 0x7D.

Interface with the HSM-1. Set a flag in memory if the password passed in is correct.

Takes two arguments. The first argument is the password to test, the second is the location of a flag to overwrite if the password is correct.

INT 0x7E.

Interface with the HSM-2. Trigger the deadbolt unlock if the password is correct.

Takes one argument: the password to test.

INT 0x7F.

Interface with deadbolt to trigger an unlock if the password is correct.

Takes no arguments.

5

Introduction to MSP430 Assembly

We provide below a brief introduction to MSP430 assembly for those readers famliar only with C programming.

Assembly programs are made of individual instructions. Instructions generally take the form:

```
opcode source, destination
```

Where `source` and `destination` refer to registers, constants, or memory locations. For example, one of the most common lines may read something like this:

```
add #10, r15
```

Which simply means,

```
r15 = r15 + 10
```

Instructions can operate on the following:

- Registers, the working state of the processor.
- The program counter, a special register that identifies the address of the next instruction to run.

- The stack pointer, another special register that identifies a specific region of memory carved out for temporary storage.
- Memory, which stores most of the state of the program.
- The CPU flags, which record things like whether the last instruction produced the value zero, or set the "sign" flag, and are used to implement logic

There are just a two kinds of instructions:

- Arithmetic instructions, like "add", compute values and store their results.
- Control transfer instructions, which decide what the next instruction is going to be.

Arguments to instructions in MSP430 can have one of the following forms:

- **Rx** Reference the value stored in `Rx` directly.
- **@Rx** Reference the value in memory stored at the address indicated by `Rx`.
- **@Rx+** Reference the value in memory stored at the address indicated by `Rx`, and *then* incremement the value of `Rx`.
- **#c** The constant `c`.
- **c** Reference the value in memory stored at offset `r0+c`.

6

MSP430 Assembly Reference

MOV arg1 arg2 → `arg2 = arg1`

ADD arg1 arg2 → `arg2 += arg1`

SUB arg1 arg2 → `arg2 -= arg1`

AND arg1 arg2 → `arg2 &= arg1` (arg1 AND arg2)

XOR arg1 arg2 → `arg2 = arg1` (arg1 XOR arg2)

BIS arg1 arg2 → `arg2 |= arg1` (arg1 OR arg2)

BIC arg1 arg2 → `arg2 &=  arg1` (clear the bits in arg1 from arg2)

CMP arg1 arg2 → compute `arg1 - arg2`, set the flags, and discard the result.

BIT arg1 arg2 → compute `arg1 & arg2`, set the flags, and discard the results (like TEST on x86)

PUSH arg1 → push arg1 onto the stack; subtract 2 bytes from the SP register (r1) and store arg1 in the resulting location in memory.

POP arg1 → `MOV @r1+, arg1`; move the value located at the stack pointer in to `arg1`, and add 2 to the stack pointer.

JMP arg1 jumps to the location arg1 points to.

CALL arg1 jumps to arg1, but first pushes the next address in memory (the return address) to the stack.

RET is thus just MOV @r1+, r0.

Jxx arg1 for xx NZ (not zero), Z (zero), LO (lower), HS (higher or same), N (negative), GE (greater or equal), and L (less) are the conditional jumps, which decide based on the flag bits Z (zero), N (negative), and C (carry).

Bibliography

[1] *The MSP430x1xx Family User's Guide* http://www.ti.com/lit/ug/slau049f/slau049f.pdf 2006.