FREE* Security Key Lite Design!
(for Personal/non-commercial use only)

Scalable Security System Using the PIC 16F84

Author: Ken Rieli 1/29/98

Table of Contents

Security Key Project Documentation - Abstract

In this last decade of the 20th century, we've seen steadily decreasing R&D budgets, in both government and private organizations. At the same time, hacking, industrial espionage and theft of intellectual property are on the rise. To counter these threats, security measures must be implemented right down to the personal level.

The least effective electronic security keys use magnetic strips or non-volatile memory chips to store key codes and user data. While this approach may be sufficient for low-level security, only key systems utilizing on-board CPUs can provide more advanced functions such as rolling key codes. This project will address the basics of key code storage and interrogation.

The PIC 16F84 is an ideal processor for highly-effective security systems due to the right mix of RAM, Flash memory, and EEPROM. This low-cost design uses the 16F84, a minimum of passive components and simple software routines to implement security key functions. By powering the key from the host system's serial port, parts count and cost are kept to a minimum.

During programming, a standard 60-byte (480 bit) key code is stored in the 84's EEPROM. Software utilities allow rewriting this memory area with other random codes. By setting the code polling length in the host system's interrogation routine, key code length is scalable from 8 bits to 480 bits -- a bit better than Federal limits of 56 bits for exportable technology. User access level codes can be stored in each hardware key, and later requested for processing.

In the second phase of this project, we add an external serial EEPROM, increasing the key code length to 250 bytes (2,000 bits). These astronomical or long codes provide a solid foundation for building absolute security systems.

This project is easy to build and provides maximum security benefits. Applications include: personal computer access, network access, electronic door locks, etc.

Block Diagram

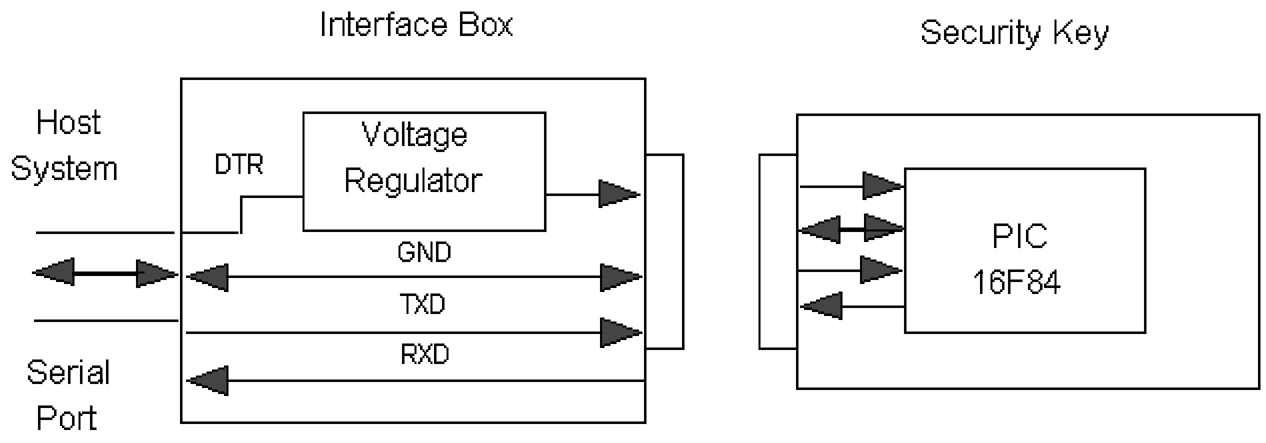

Figure 1

Forward

Security -- it's a subject all of us have to contend with sooner or later. It should be of particular interest to individual entrepreneurs, engineers and designers who work independently from big business, and rely on their designs for survival.

In this last decade of the 20th Century we've witnessed a dramatic increase in intellectual piracy. By some estimates, there is as much business in bootlegged goods as there is in honest, legitimate design. The reason for this is that R&D is very expensive -- the most expensive part of doing business. Major corporations, governments and industries around the world are decreasing money spent for R&D, and shifting their emphasis toward economies of scale -- mass production and sales -- commercial off-the-shelf products. So where will big business get designs for new product if they don't put major resources into R&D? You guessed it -- from small businesses and individuals who aren't aware of how easily their intellectual property is confiscated. By simply logging onto the Internet, you create a two-way street between your computer and any smart hacker bent on pilfering your files. By far, most piracy is committed by somebody within your organization; about 80% of all theft is by employees or someone you know!

As your IT becomes more valuable, and piracy becomes more lucrative, what can you do? Begin today to study about and implement security measures. Even if your operation is very small, it is prudent to set up and build on basic security procedures and technology. It is security technology that this project focuses on.

Background

The simplest form of security involves using a memorized password for access to computers, networks, labs, etc. The main problem with this approach is that passwords must be very short to be remembered, and to allow access to the system before the next Ice Age. Short passwords result in easy breaches of security. More sophisticated security schemes use longer passwords for better protection.

The Federal government here has limited exportable security product bit lengths to 56 bits -- allowing them to break into any system within a relatively short amount of time. The technology being presented here breaks the 56-bit barrier, providing a much longer key code length.

The code length alone places this technology beyond the reach of hackers -- private, corporate or government. Although this feature disqualifies the technology for exportation and foreign sales, it is perfectly legitimate to use within U.S. borders. (Data security technologies are classified as munitions by the Federal government.) The security key project outlined here may be used in conjunction with RSA, PGP, ASA, or any security algorithm to provide very good security for individuals and businesses alike.

System Design -- Theory of Operation

The basic principle of this project is very simple: the host computer maintains a number of security key files on disk; on command, the host system interrogates the security key for its stored code via the serial interface. The key code is then compared to the file code for a go/no-go decision and subsequent action.

Host System Flowchart

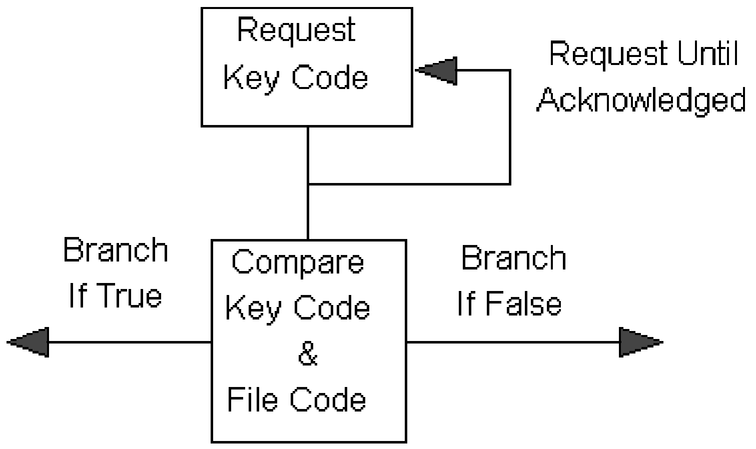

Figure 2

The 16F84 architecture integrates 68 Bytes of RAM, 1K Flash, and 64 Bytes of EEPROM. Using the PIC Basic Compiler (MELABS Inc.), RAM is used for variables, Flash for storing compiled program code, and EEPROM for storing key code data. EEPROM storage space is utilized as follows:

Data Location:			
0	*1-60*	*61*	*62*
Access Level	*Key Code*	*Keylength*	*Key #*

Scalable Key Codes -- While most commercial security products are limited to a fixed 56 or other bit length, this design utilizes a scalable scheme. Scalability allows us to:

a. Decrease the bit length for greater throughput

b. Increase bit length for maximum security

At data rates of 9600 baud, a full-length (480 bit) key code transfers reasonably fast. For speed freaks, a 56-100 bit length may be more suitable. Government and industry prefer maximum security and numerous access levels.

To satisfy everyone's needs, this design stores key codes at a full 480 bits. Code interrogation may be any value between 8 and 480 bits. Expanded systems may store various key lengths; a practical limit is about two thousand bits. Data location 61 stores the key length value.

Using the security key utilities, a security level may be written to each key to determine access capabilities.

System Development

After drafting preliminary specs and schematics, Phase I of the hardware was bread boarded; this allowed us to easily verify the design and change components or pinouts as the software was developed. *(See Photo 1)*

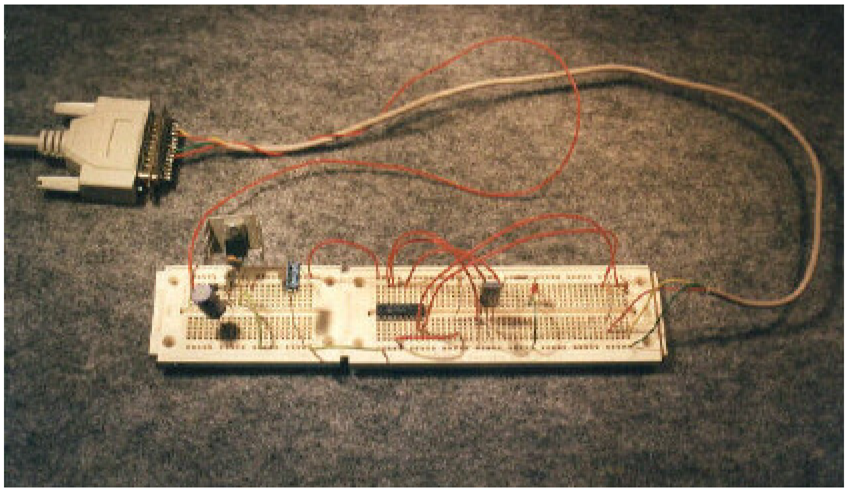

Photo 1 - breadboard

In any project involving hardware and software development, hardware is by far the easiest part of the job. After bread boarding components, we arrive at the job of writing and debugging code. In order to transform this arduous task into an easy and (in my case) fun experience, we use the best tools we can find for the non-professional programmer: Basic!

We've found that, by using a compiler Basic, we can achieve very high performance and throughput. The main benefit is that it dramatically decreases the time and energy usually associated with the development of software -- particularly when debugging. This translates into lower cost of development and shorter time-to-market cycles. For x86 development, we use Robert Zale=s Power Basic; for PIC development, we use PIC Basic Compiler (MELABS Inc.)

After debugging code, Power Basic programs are compiled to EXE files. PIC Basic programs are first compiled to HEX files, then downloaded to the 16F84 using MELABS= EPIC- programmer.

In the second phase of the prototyping, I modified an old PIC Basic 1 stamp board. With a new processor, a serial (I2C) EEPROM and a few passive components, the operation was complete. The interface box was fabricated from a generic plastic project case. *(Photo 2)*.

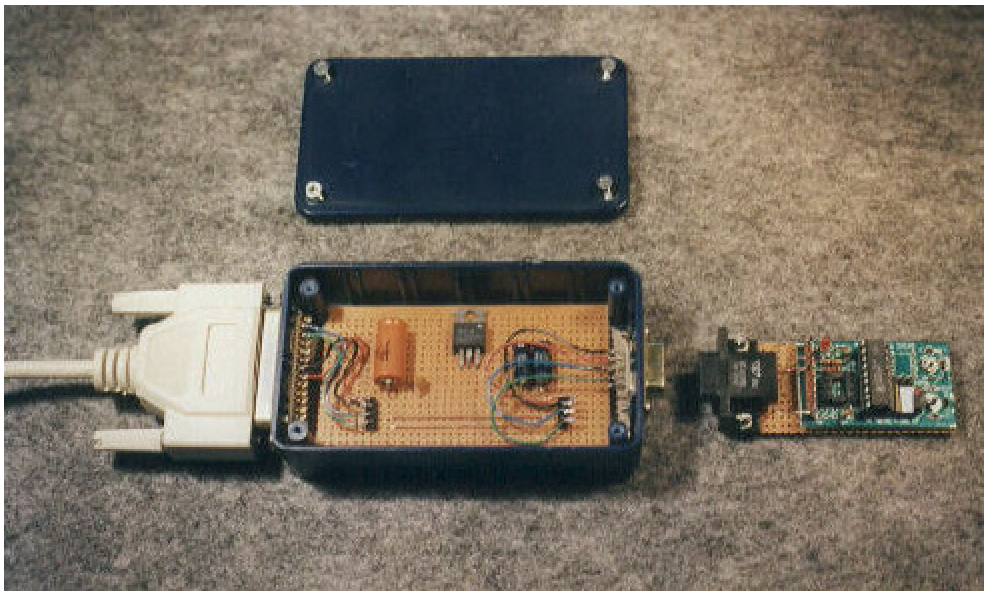

Photo 2 - project prototype

*Note: When using external I2C EEPROM with the MELABS development system, data and clock lines must be assigned to port pins in the PBL.INC file before compiling.

Hardware Description

Schematics 1a and 1b show the complete hardware diagram with component values. As you can see, there are very few parts involved in this design.

Schematic 1a

The serial interface consists of a small enclosure with a voltage regulator clamping the 12-volt DTR line to 5 volts for the 16F84. Ground, transmit and receive lines pass through to the key connector. The serial cable connecting the host system to the serial cable must be a pass-through type. Do not use a null modem cable, since DTR is not passed through.

Schematic 1b

The hardware key enclosure may be anything from a cable end to a small, custom-made Adog tag@. Low-profile parts will allow very thin packages to be utilized. The 16F84 is wired in a typical manner: Pin 11 is data in, Pin 12 is data out. Pin 13 pulses the optional led to show key access activity. A reset button is also optional; it allows resetting the key in case it hangs.

The optional external EEPROM portion of the circuit shows the 256 byte upgrade on the Basic Stamp I board. The clock, data, write protect, and VCC lines of the EEPROM are tied directly to the 16F84 port B, allowing complete control of external memory chips.

Software

Software Software included in this project:

1. SKPIC5.BAS (Security Key -- PIC Basic source code)
2. SKIV5.BAS (Security Key Interrogate -- Power Basic source code)
3. SKUTILS5.BAS (Security Key Utilities -- Power Basic source code)

SKPIC5.BAS -- Embedded code for the 16F84 In reference to Software Flowchart 1a-f, and Software Listing 1: After initializing the 16F84 with a dummy key code, access level #0 and key #0, the CPU waits for a valid command.

> Command 1 -- Read Code After acknowledgment is exchanged with the host system, the stored key code is read from EEPROM and sent via the serial line. EOT is then sent to signal an end of transmission. The optional led on the key is then blinked to indicate a transaction.
>
> Command 2 -- Store Code After sending a "ready" code (ACK), the key I.D. number and new key code are stored in EEPROM. Code is simultaneously returned to the host system for on-screen comparison. EOT is then sent to signal end of transmission. The led again indicates a transaction.
>
> Command 3 -- Store Access Code After sending (ACK), a new access level code is received and stored in EEPROM.
>
> Command 4 -- Key Length The key length is stored to a temporary RAM register; transmission lengths may be modified dynamically with each interrogation.
>
> Command 5 -- Read Key Specs Key code length, key I.D. number, and access level are sent to the host system for further processing.

SKIV5.BAS -- Interrogate Key Routine, for x86 host system The code interrogation demo was developed to make it easy to integrate security keys into any application, particularly new development. A TSR (terminate and stay resident) program can be written to interrogate key codes prior to computer system access. Programmers can embed this routine into their own code.

Referring to Software Flowchart 2 and Software Listing 2, key specs are first requested. After an acknowledgment is exchanged, the key length, I.D. number and access level are received.

Next, a dynamic key length value of 7 (KCL) is sent to Key RAM, limiting the transmission length to 7 bytes (56 bits). This value may be changed by the programmer to any value between 1 and 60.(1-250 for external EEPROM.)

The last part of the routine requests the actual key code and compares it to the disk file for verification. All of the acknowledgment codes, key codes, and key specs are printed to screen in the demo. Audible feedback alerts the user to algorithm progress. This program may be modified to exclude screen and audio feedback when embedding code into other applications.

SKUTILS5.BAS --Utilities for the x86 host system Referring to Flowchart 3 and Software Listing 3, this program is the complement to code embedded in the 16F84 key and provides all of the functions for generating code files, programming, verifying and interrogating hardware keys. Both internal and external EEPROM security key types are supported by this program.

The selection screen is as follows:

1. Interrogate Hardware Key
2. Program Hardware Key
3. Set Keycode Length
4. Set Access Level
5. Verify Hardware Key
6. Generate New Code Files
7. Exit

Programming hardware keys with SKPIC5 or SKPIC5E (expanded) initiates EEPROM space with default values. Interrogating a key before programming it with option (2) will display these defaults. A typical programming session would proceed as follows:

A. Generate New Code Files -- Select option (6). When prompted, enter a file length, enter 60 for a non-expanded key, or 250 for an expanded key. The next prompt requests the number of key files to generate. A small company may need only 10 or 20 keys. Larger companies need hundreds or thousands of keys. This project allows up to 256 keys.
B. Program Hardware Key -- Select option (2) to transfer a key file from disk to a hardware key. When prompted for a disk file number, enter the number of a valid disk file (up to the maximum generated).
C. If access levels are necessary, select option (4) and enter the level number when prompted.
D. Verify Hardware Key -- Select option (5) to compare the key code with the on disk file code. Results are printed to screen as "Key O.K." or "Key Error". (Note: Do not regenerate code files once keys are all programmed, as the file codes are necessary for key verification.)
E. Interrogate Hardware Key -- Select option (1) to view all data written to the security key.

F. Set Keycode Length -- Select option (3) to limit interrogation lengths to any value of Bytes -- 1-60 for non-expanded and 1-250 for expanded systems. Subsequent interrogations will be limited to this length. Since this value is stored in the 16F84 temporary register, the value is lost when the key is powered down.

Conclusions

In conclusion, the PIC 16F84 is an excellent processor for low-cost, low-power security key applications. The quality and sophistication of this project far exceeds the Federal imposition of 56 bits, even with the simple techniques discussed here. The surface mount SOIC version of the 16F84 allows for very small security key packaging.

Future enhancements to the basic project will incorporate rolling key code techniques for better security. Pushing the oscillator clock up to the 10mHz rating of the 16F84-10 will give us baud rates of 23,500. This increase in throughput and performance is essential when using long code lengths.

The external EEPROM located on the Basic Stamp board allows code lengths in excess of 2,000 bits. When used in conjunction with rolling code techniques, these keys are absolutely unbreakable!

Note: If you use the EPIC Programmer with a PIC 16F84-10/P part, use the HS (High Speed) oscillator option -- not XT -- with a crystal or resonator

Appendix

Schematics, Flowchart 1a

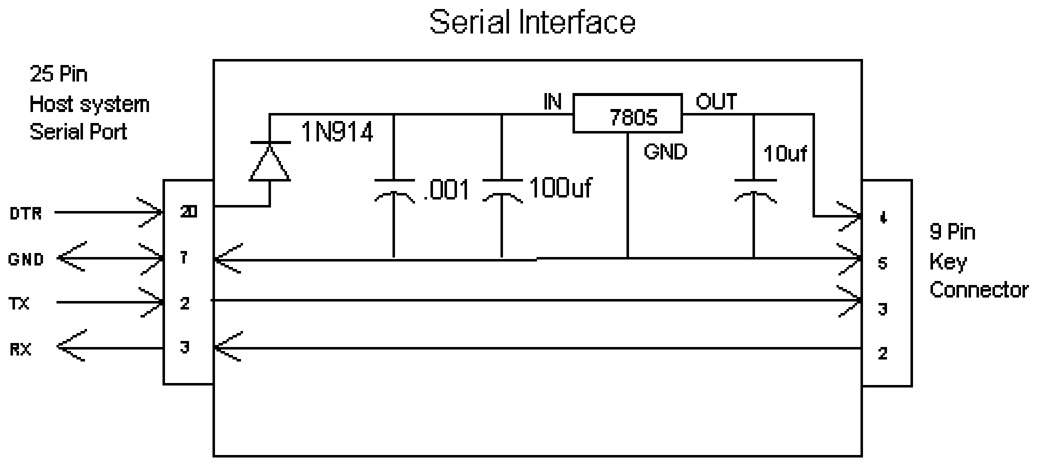

Figure 3
Schematic 1a

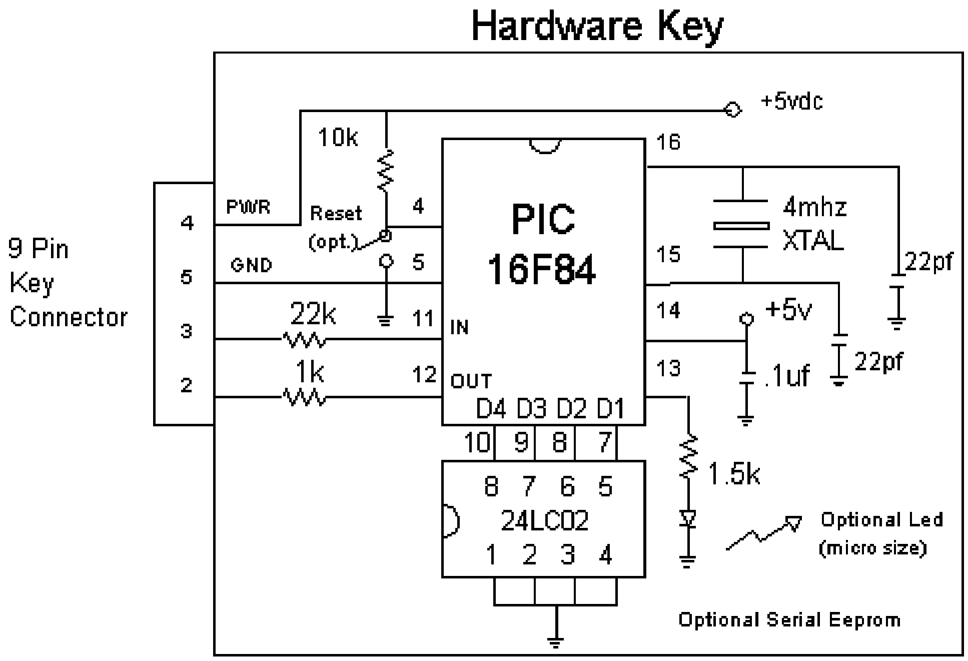

Figure 4
Schematic 1b

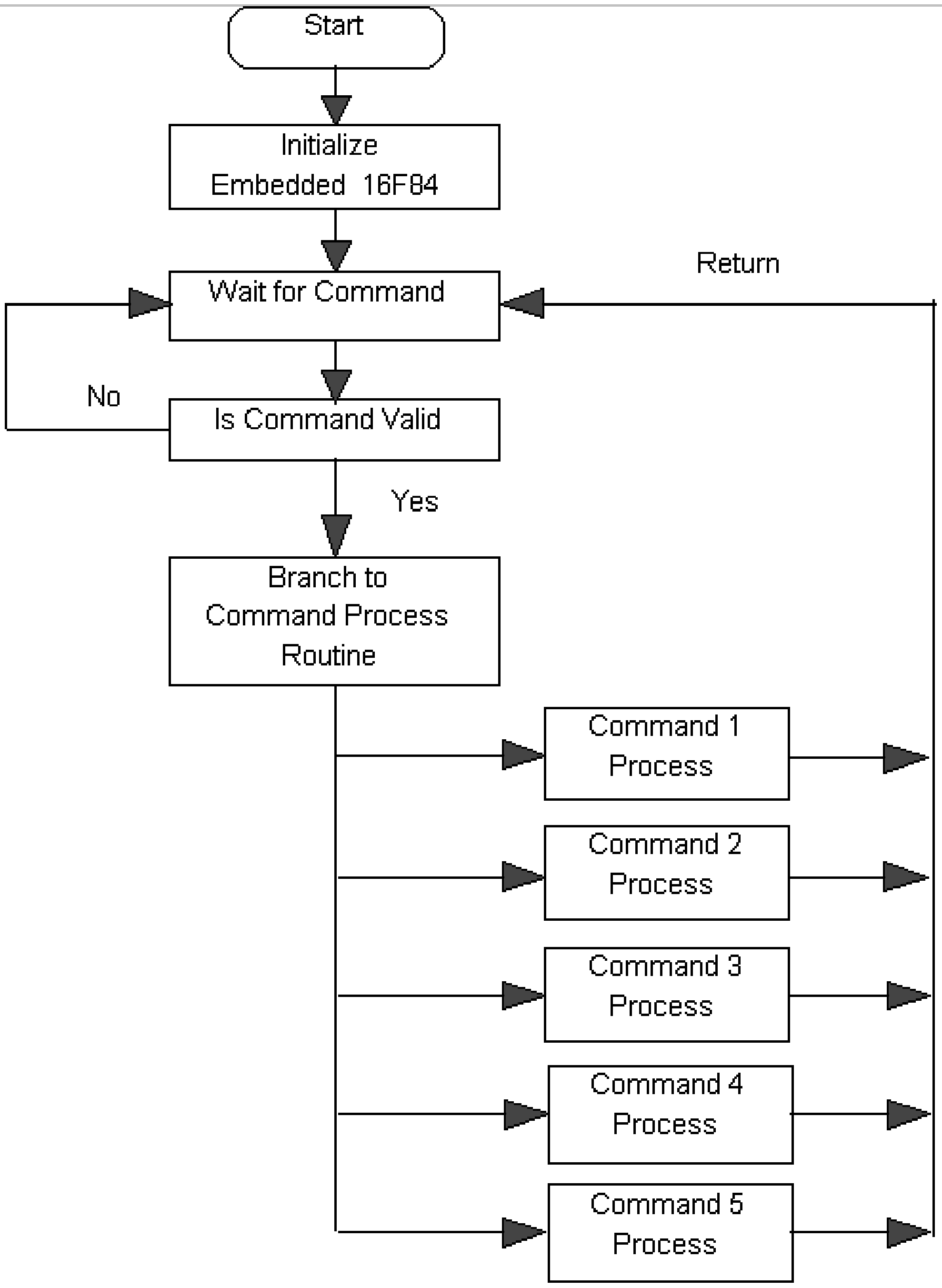

Figure 5
Software Flowchart 1a

Software Flowcharts 1b - 1f

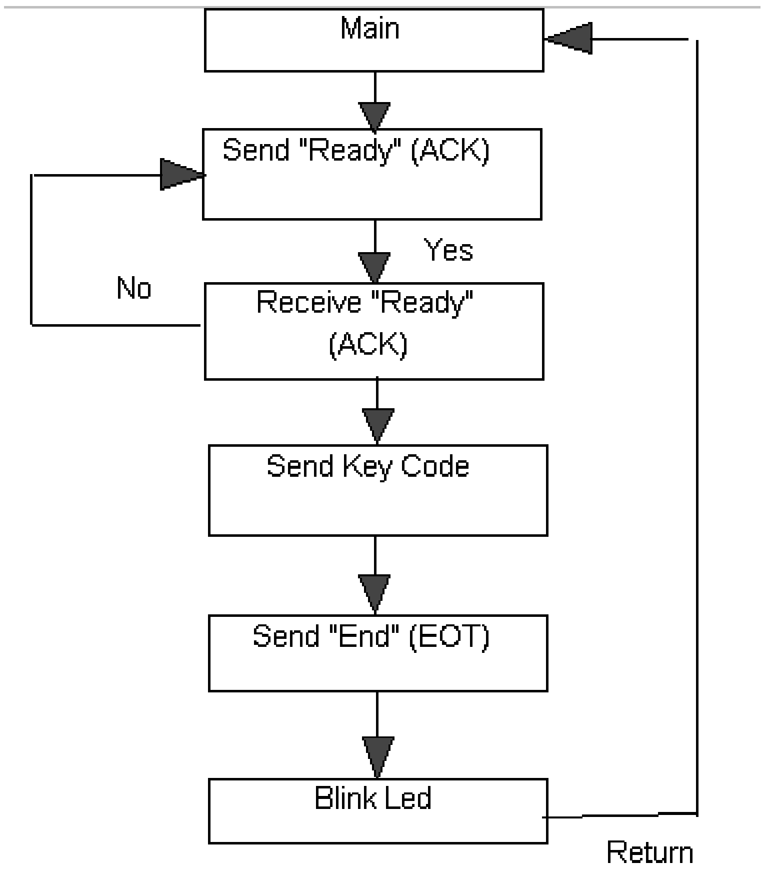

Figure 6

Software Flowchart 1b (Command 1)

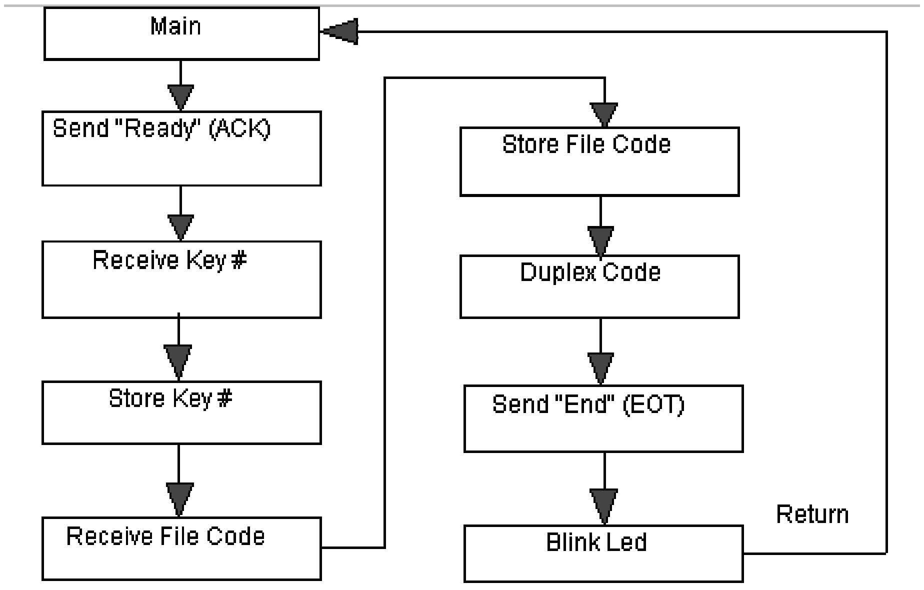

Figure 7

Software Flowchart 1c (Command 2)

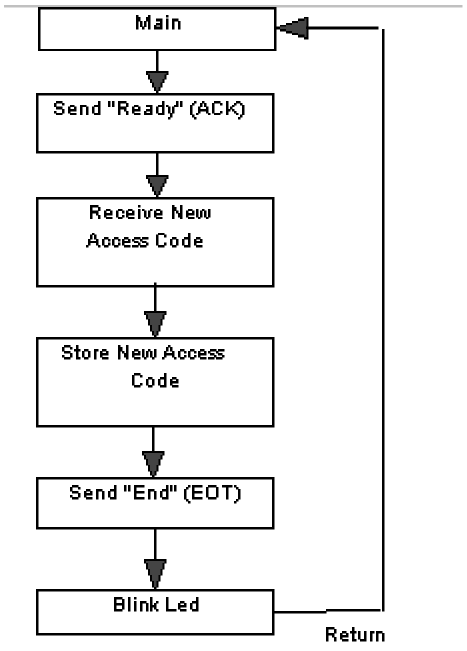

Figure 8

Software Flowchart 1d (Command 3)

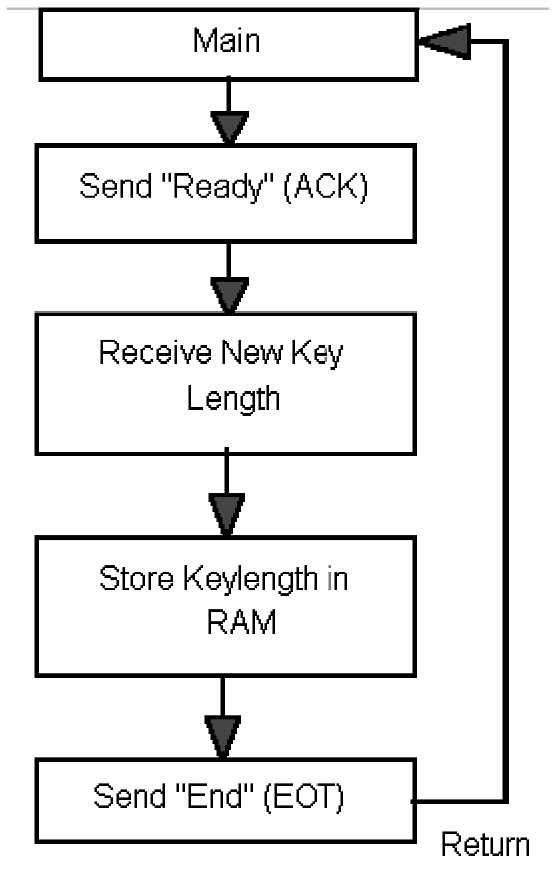

Figure 9

Software Flowchart 1e (Command 4)

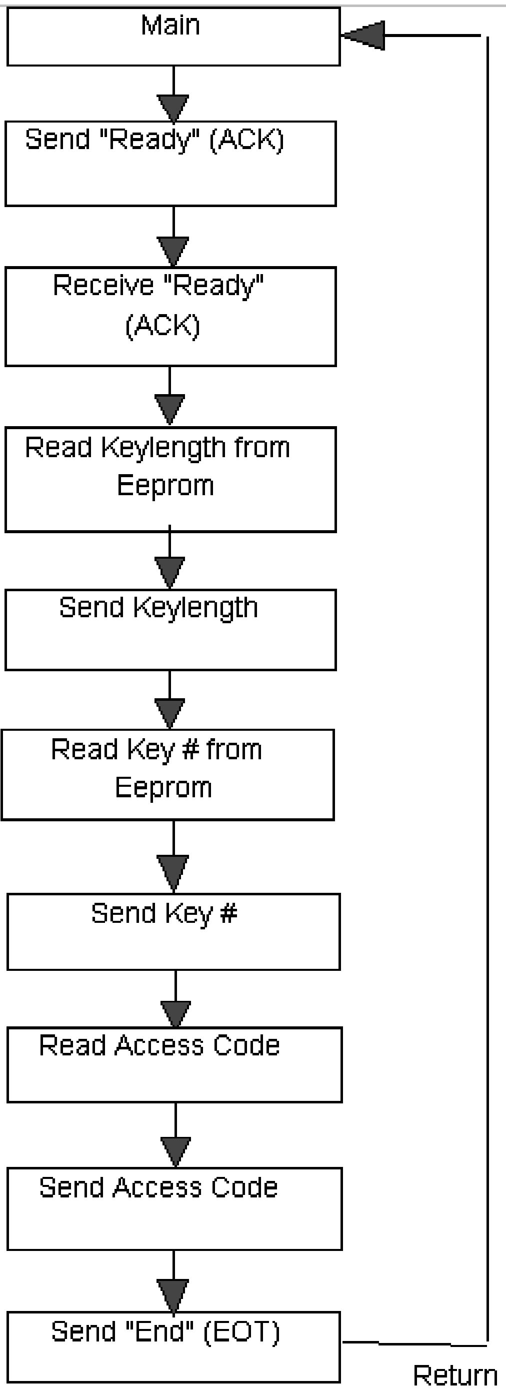

Figure 10

Software Flowchart 1f (Command 5)

Software Flowcharts 2 & 3

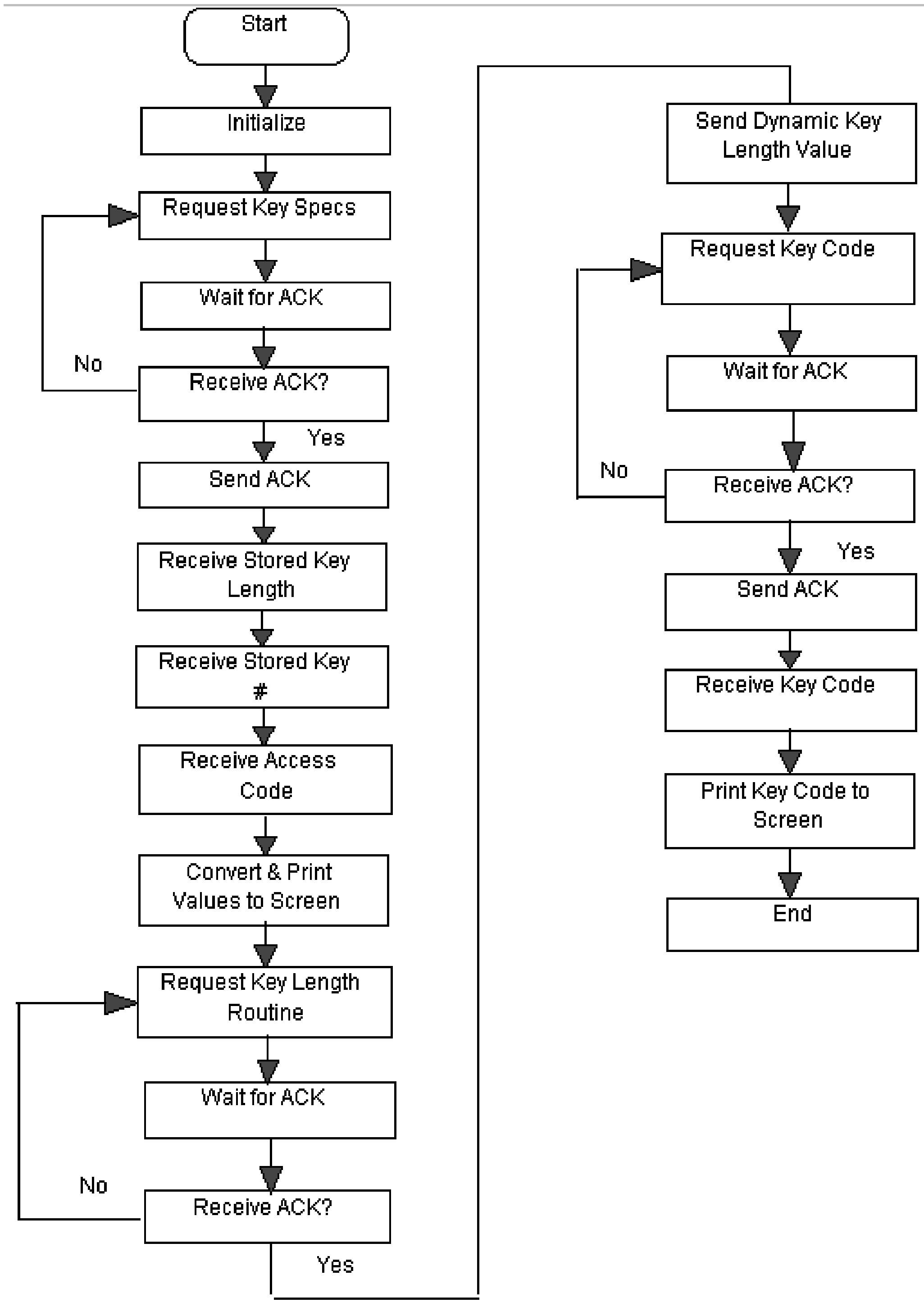

Figure 11

Software Flowchart 2

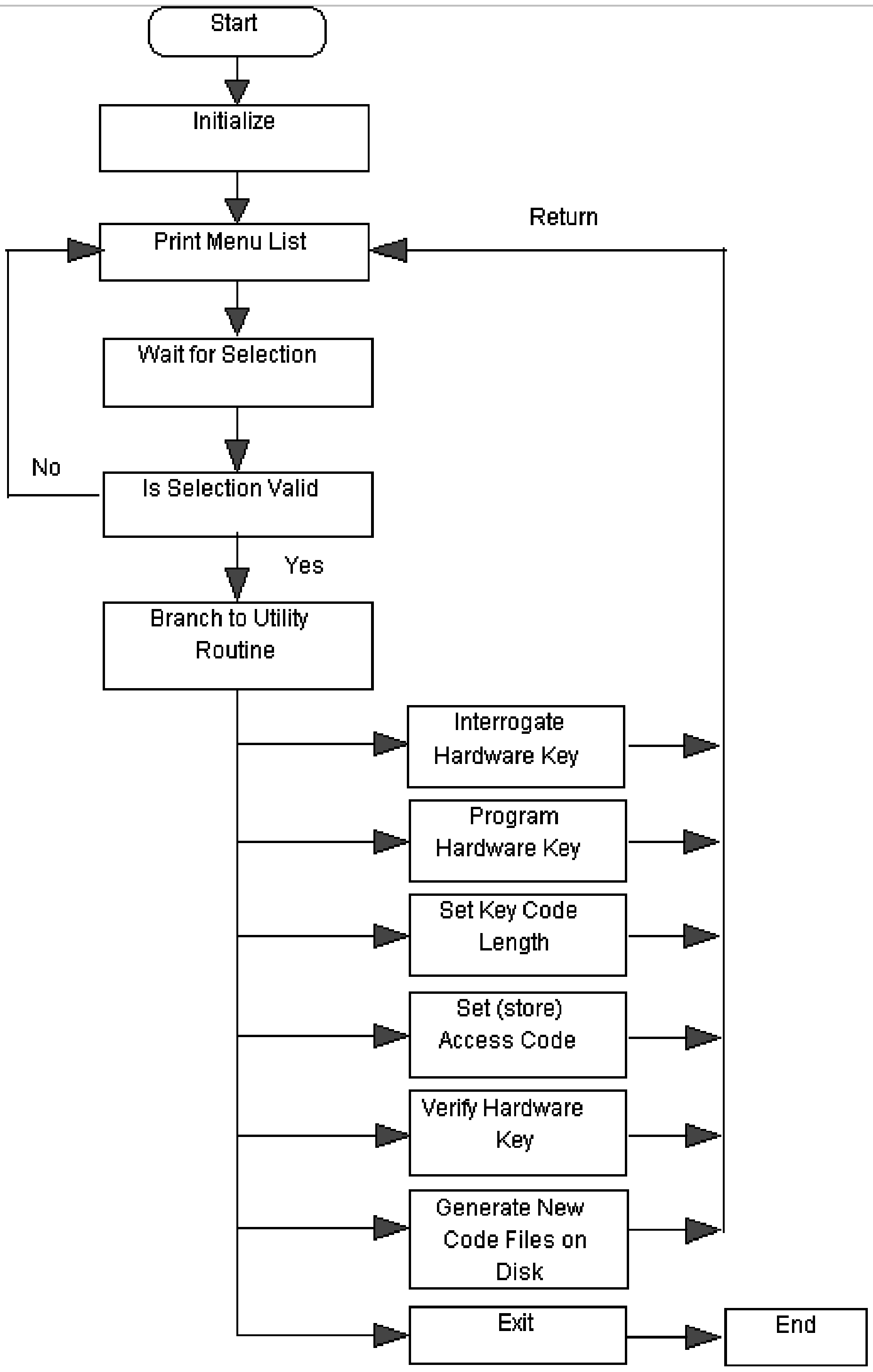

Figure 12

Software Flowchart 3