Arm Cortex-M4 I/O Implemented 1-Wire Secure Authenticator Demo and Real Time Measurements

MAXREFDES9007

Overview

The MAXREFDES9007 is a reference design that showcases the **DS28E36** and demonstrates how to implement a simple 1-Wire® host using only a microcontroller's GPIO pins. The reference design provides a GPIO-based 1-Wire library designed for an Arm® Cortex® M4 microcontroller, such as the **MAX32660**, with example programs that run the main 1-Wire sequences and calculate how much time it takes for each to run. This design also provides the amount of memory required to store such programs.

Features

- ECC-256 Compute Engine
- FIPS 186 ECDSA P256 Signature and Verification
- ECDH Key Exchange with Authentication Prevents Man-in-the-Middle Attacks
- ECDSA Authenticated R/W of Configurable Memory
- SHA-256 Compute Engine
- FIPS 180 MAC for Secure Download/Boot Operations
- FIPS 198 HMAC for Bidirectional Authentication and Optional GPIO Control
- Two GPIO Pins with Optional Authentication Control
- Open-Drain, 4mA/0.4V
- Optional SHA-256 or ECDSA Authenticated On/Off and State Read
- Optional Set On/Off After Multiblock Hash for Secure Boot/Download
- RNG with NIST SP 800-90B Compliant Entropy Source with Function to Read Out
- Optional Chip-Generated Pr/Pu Key Pairs for ECC Operations
- 17-Bit One-Time Settable, Nonvolatile Decrement-Only Counter with Authenticated Read
- 8Kbits of EEPROM for User Data, Keys, and Certificates
- Unique and Unalterable Factory Programmed 64-Bit Identification Number (ROM ID)
- Optional Input Data Component to Crypto and Key Operations
- Single-Contact 1-Wire Interface Communication with Host at 11.7kbps and 62.5kbps
- Operating Range: 3.3V ±10%, -40°C to +85°C
- 6-Pin TDFN-EP Package (3mm x 3mm)
- Accessory and Peripheral Secure Authentication
- IoT Peripheral Crypto-Protection
- Secure Boot or Download of Firmware and/or System Parameters
- Secure Storage of Cryptographic Keys for a Host Controller

Applications

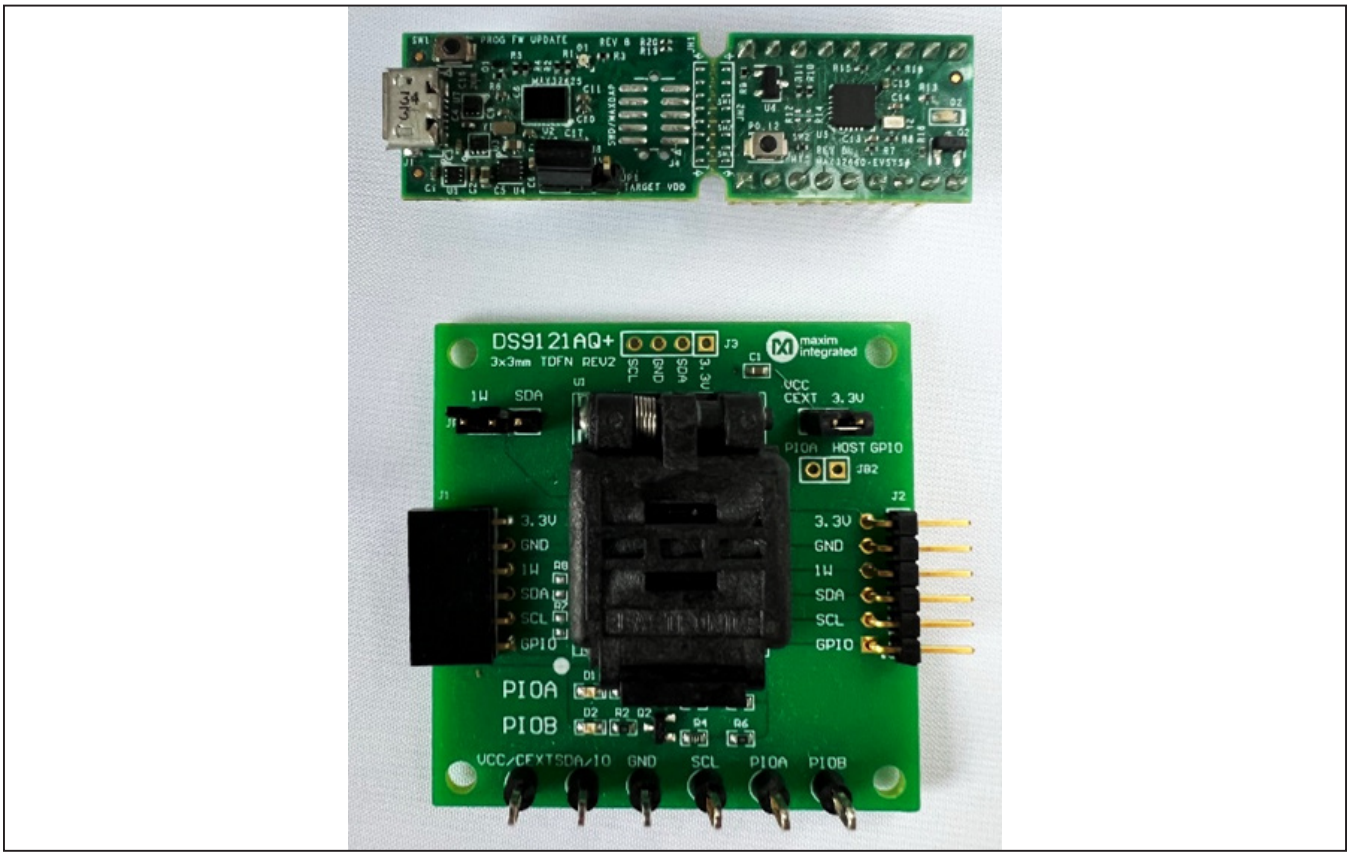
- IoT Peripheral Crypto-Protection
- Accessory and Peripheral Secure Authentication
- Secure Storage of Cryptographic Keys for a Host Controller
- Secure Boot or Download of Firmware and/or System Parameters

Arm and Cortex are trademarks of Arm Limited.
1-Wire is a registered trademark of Maxim Integrated Products, Inc.

One Analog Way, Wilmington, MA 01887 U.S.A. | Tel: 781.329.4700 |

Introduction

A simple, cost-efficient, 1-Wire line using the DS28E36 is demonstrated for a simple host/peripheral communication system for the MAX32660 host microcontroller. This reference design includes the following major components: a MAX32660, DS28E36, and a DS9121AQ+ socket board. This document describes the hardware shown in Figure 1 as well as its supplementing software. It provides a detailed, systematic technical guide to set up and understand the MAXREFDES9007 reference design. The system has been built and tested, details of which follow later in this document.



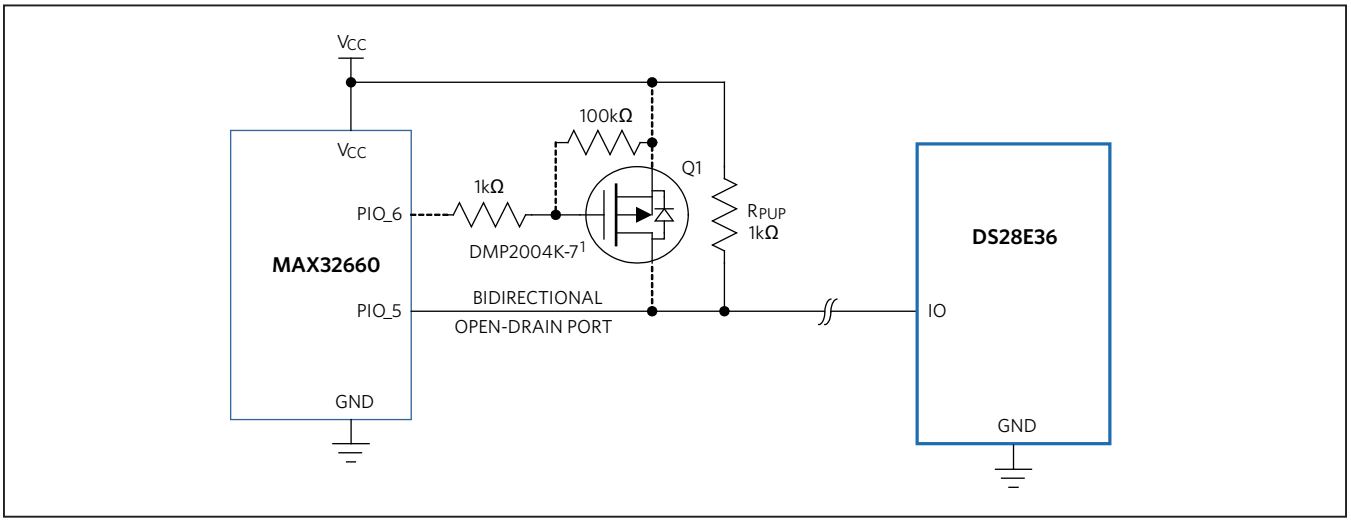


Figure 1. MAXREFDES9008 Hardware.

Quick Start Guide

The reference design is fully assembled and tested. Follow these steps to set up the demo software:

Required Equipment:

- PC with a Windows® 10, Windows 8, or Windows 7 operating system (64 bit or 32 bit), and a spare USB 2.0 or higher port
- Micro USB 2.0 cable
- Maxim Micros SDK (Windows)
- MAXREFDES9007 C-Demo software
- Serial Console Application (such as PuTTY)
- Saleae Logic 2 (optional)
- DS28E36 + DS9121AQ+ (Or the DS28E36 Evaluation System)
- MAX32660-EVSYS

Procedure

1) Download the **ARMCortexToolchain.exe** file.
2) In a file viewer (Figure 2), double-click **MaximMicrosSDK.exe** to begin the installation.
3) Follow the prompts on the setup wizard (Figure 3) to finish the installation.

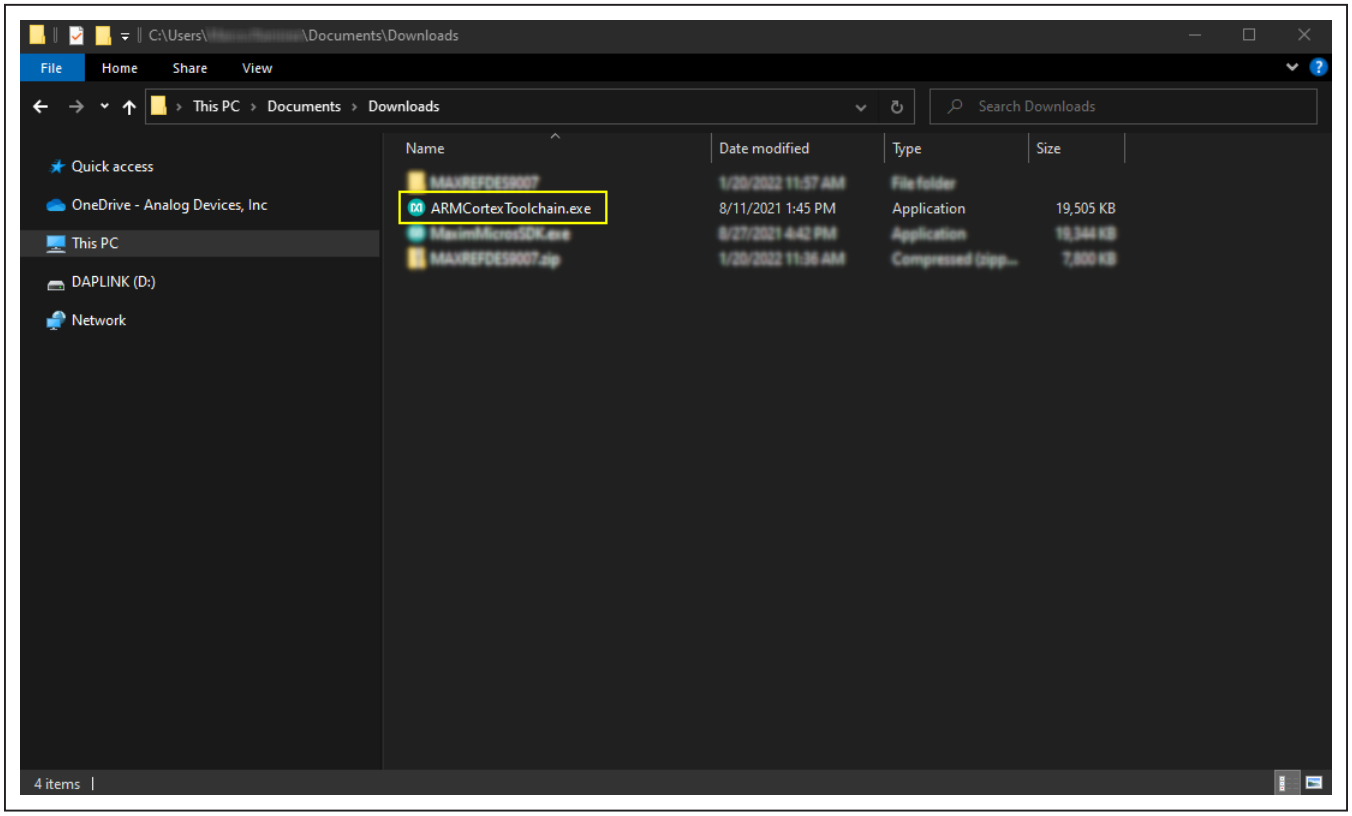


Figure 2. File Viewer with Toolchain.

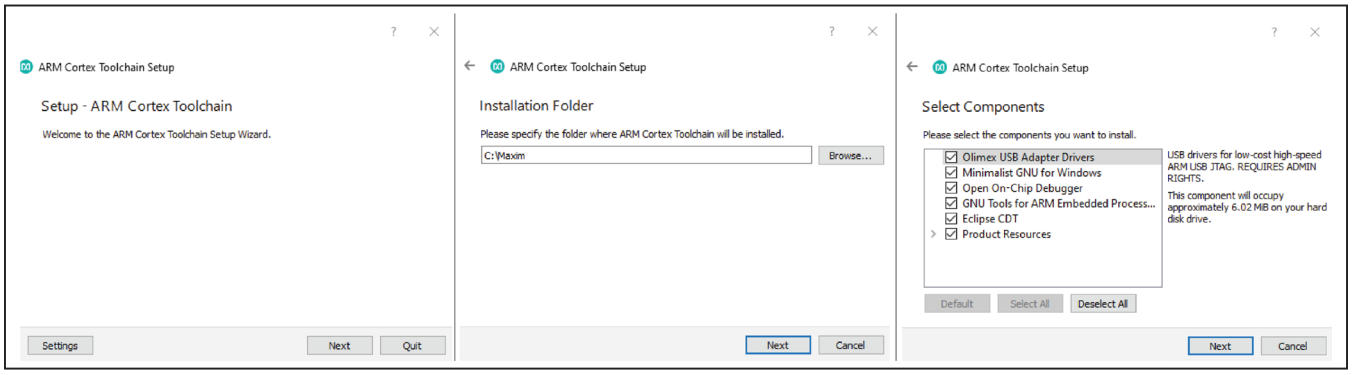


Figure 3. Arm Cortex Toolchain Wizard.

Windows is a registered trademark of Microsoft Corporation.

4) Click on the Windows icon on the bottom left-side of the screen, search for the Maxim Integrated folder, then run the **Eclipse** application (Figure 4).

Alternatively, navigate to the toolchain's install directory, open the Eclipse folder, and run **Eclipse.exe** to launch the Eclipse IDE (Figure 5).

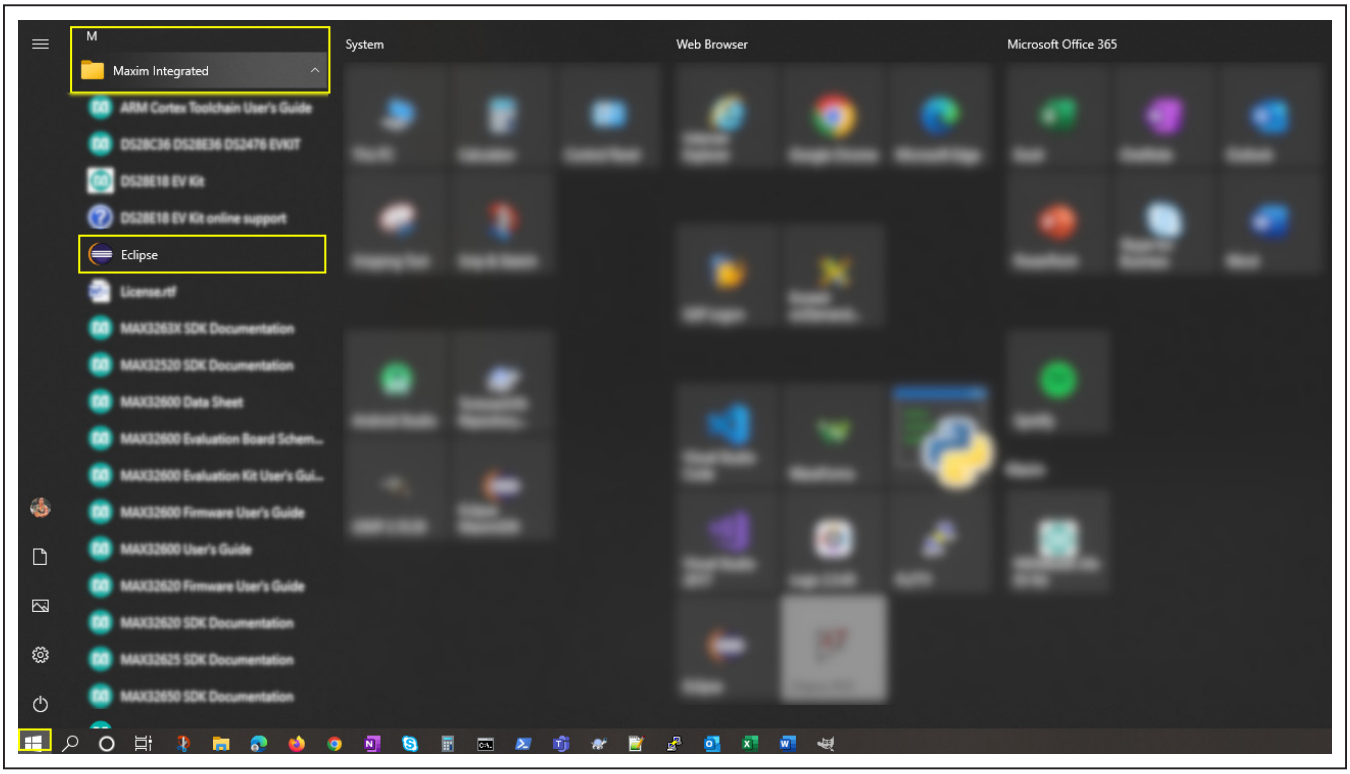


Figure 4. Eclipse application Launch through Windows Menu.

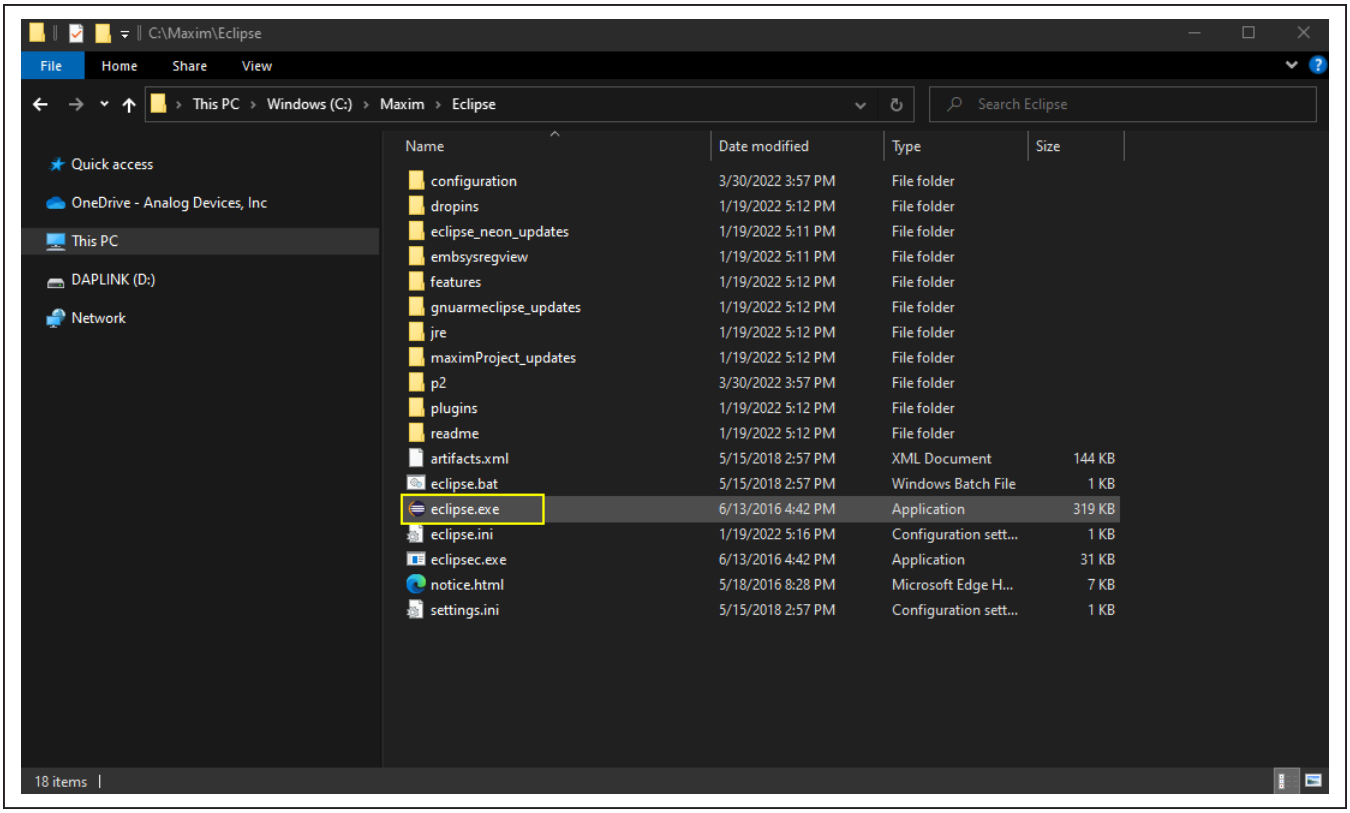


Figure 5. Eclipse application Launch through File Explorer.

5) Create a workspace in the desired location (Figure 6).
6) Download and extract the **MAXREFDES9007-V1.0.0.zip** file to any location (Figure 7).

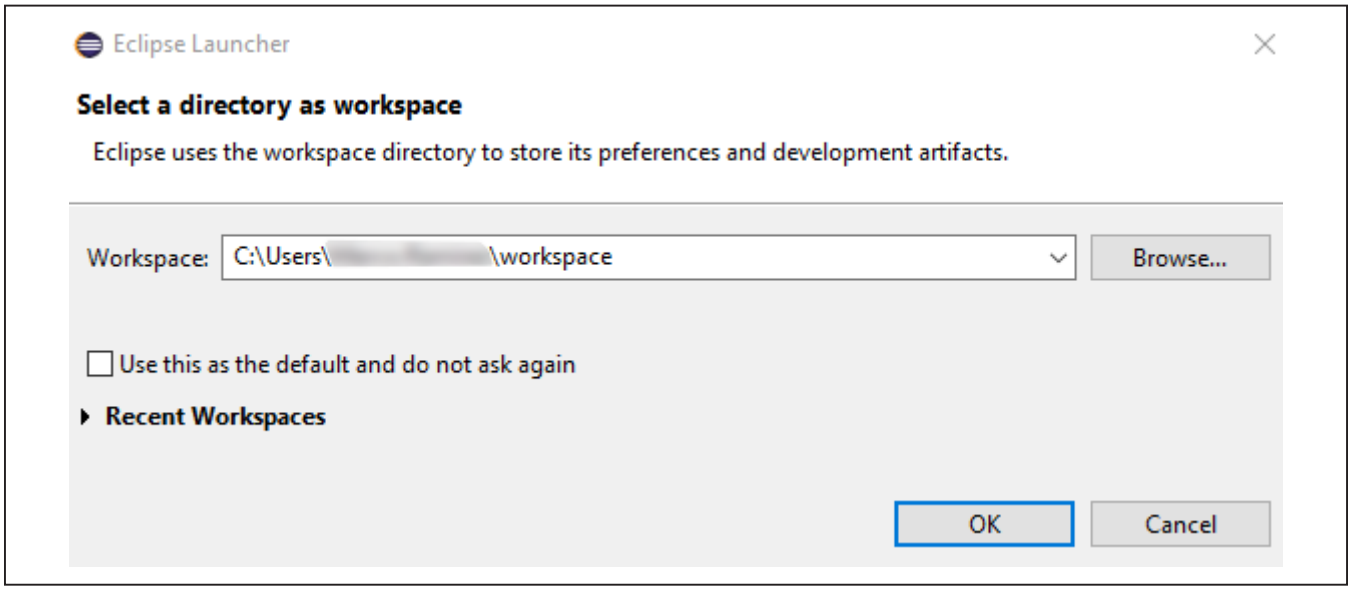


Figure 6. Eclipse Workspace Creation.

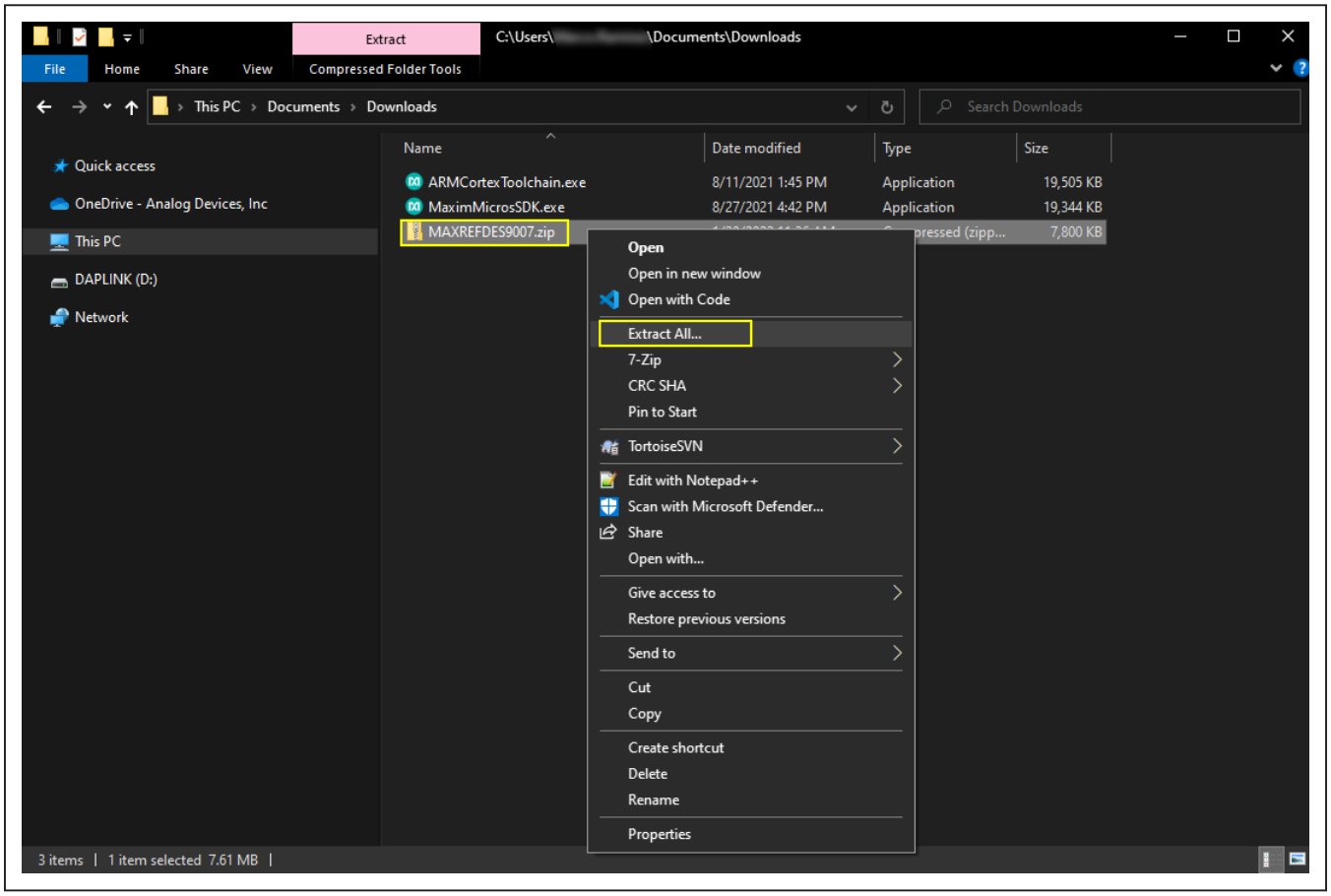


Figure 7. MAXREFDES9007 Extraction.

7) In Eclipse, go to **File->Import** and select **Existing Projects into Workspace** under the General folder (Figure 8). Click **Next >** and then **Browse** to open a directory browser. Navigate to the MAXREFDES9007 C-demo installation directory. Select the extracted folder containing the example programs and click OK. In the panel, check the checkbox next to the **MAXREFDES9007_HMAC** and **MAXREFDES9007_ECDSA** projects to import. Check **Copy projects into workspace** and click **Finish** (Figure 9).

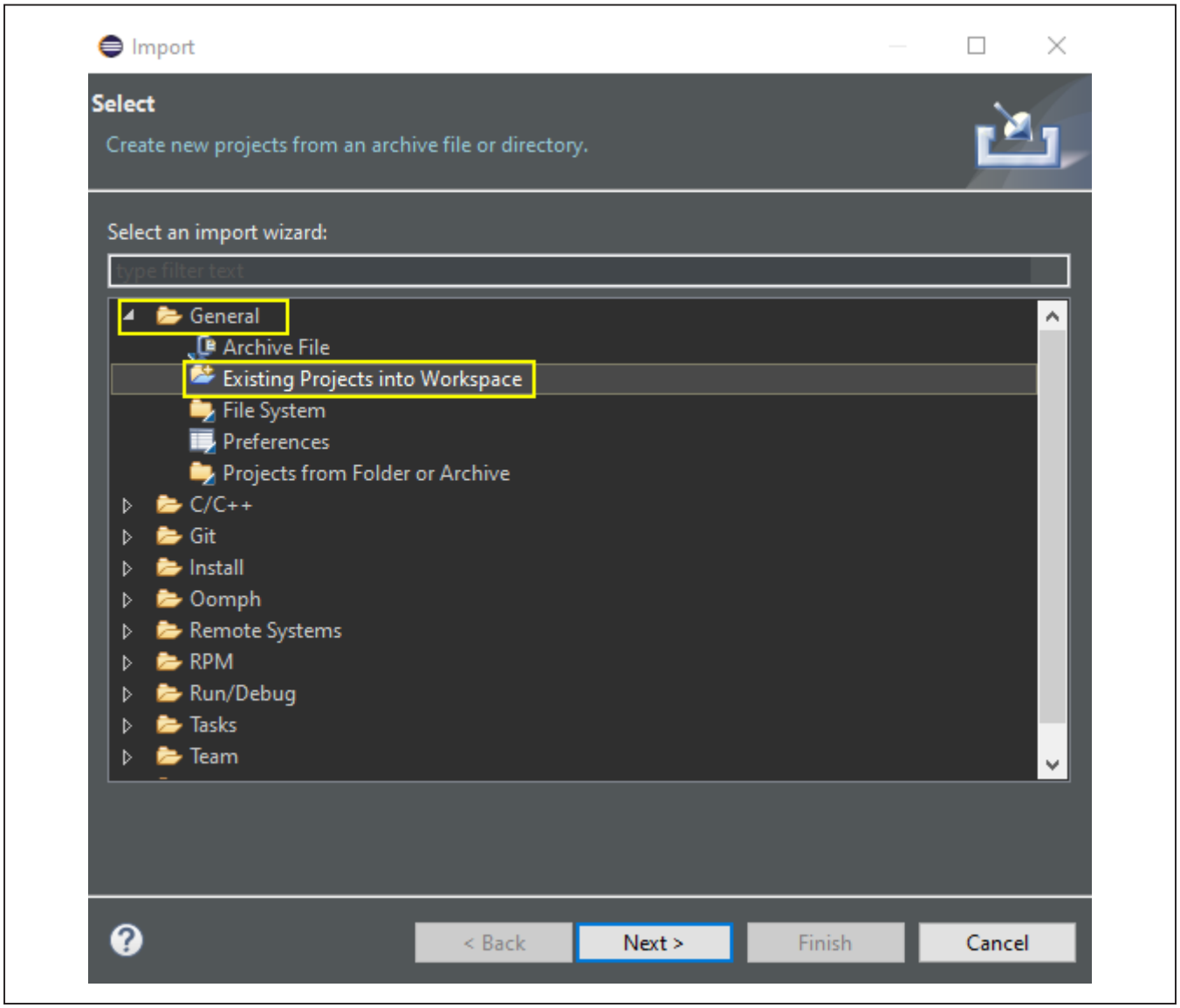


Figure 8. Selecting Eclipse Import Type.

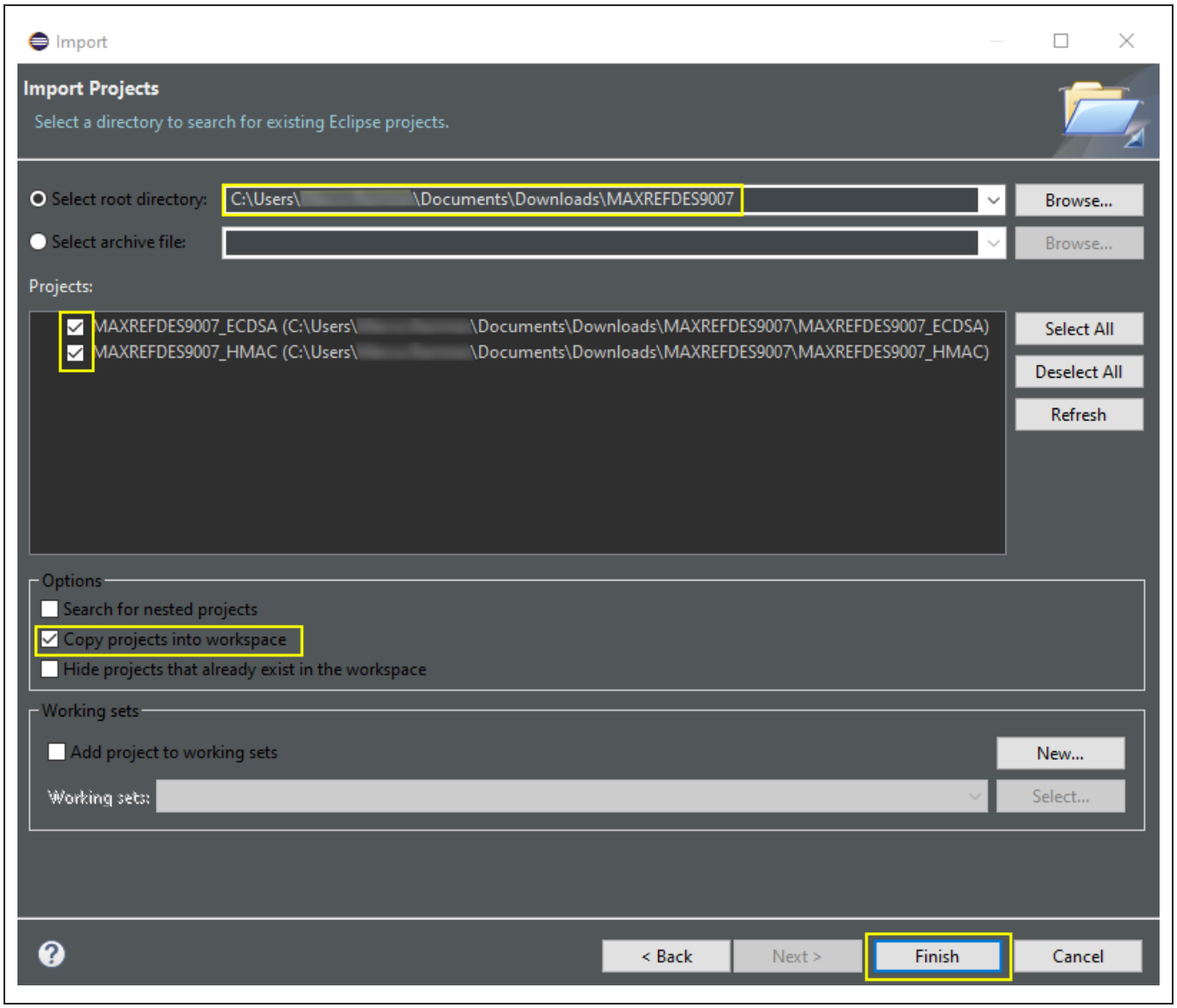


Figure 9. Importing Eclipse Projects.

8) Open a serial console and connect it to the MAX32660's corresponding serial COM port (Figure 10).

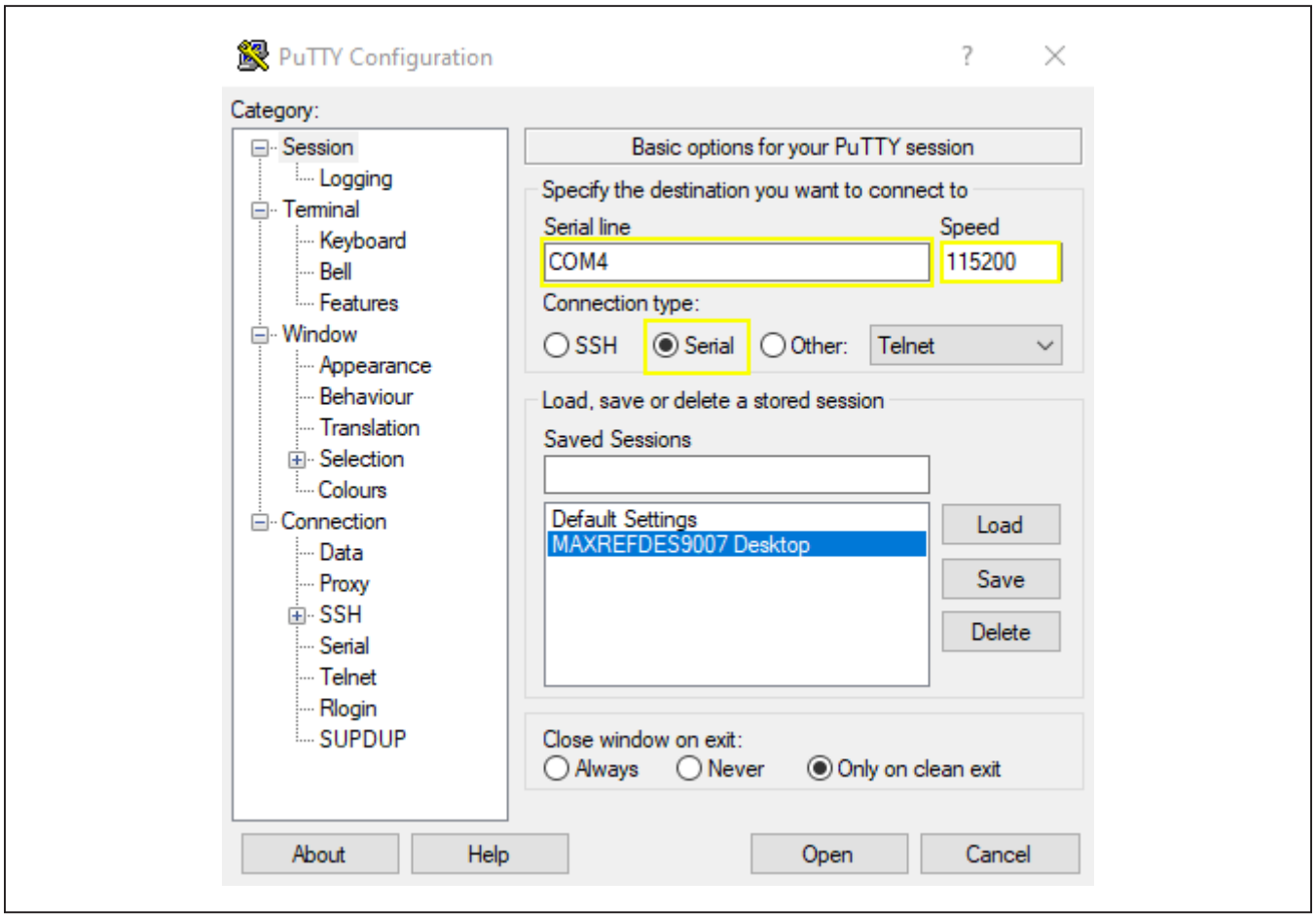


Figure 10. Serial Console Setup (PuTTY).

9) Select an example program to run under the dropdown menu located next to the green Run button (Figure 11). The output is displayed on the serial console (Figure 12).

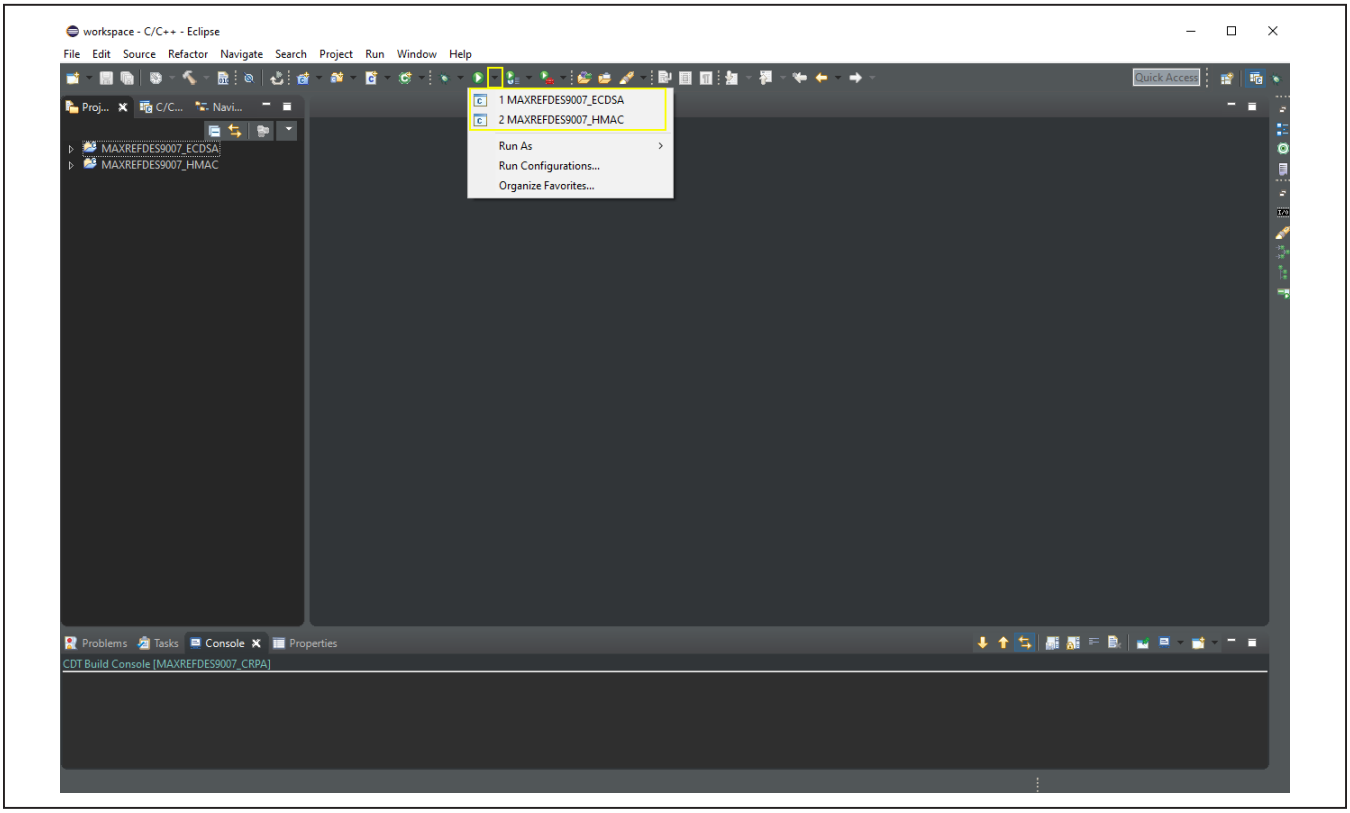


Figure 11. Running One of the MAXREFDES9007 Demos.

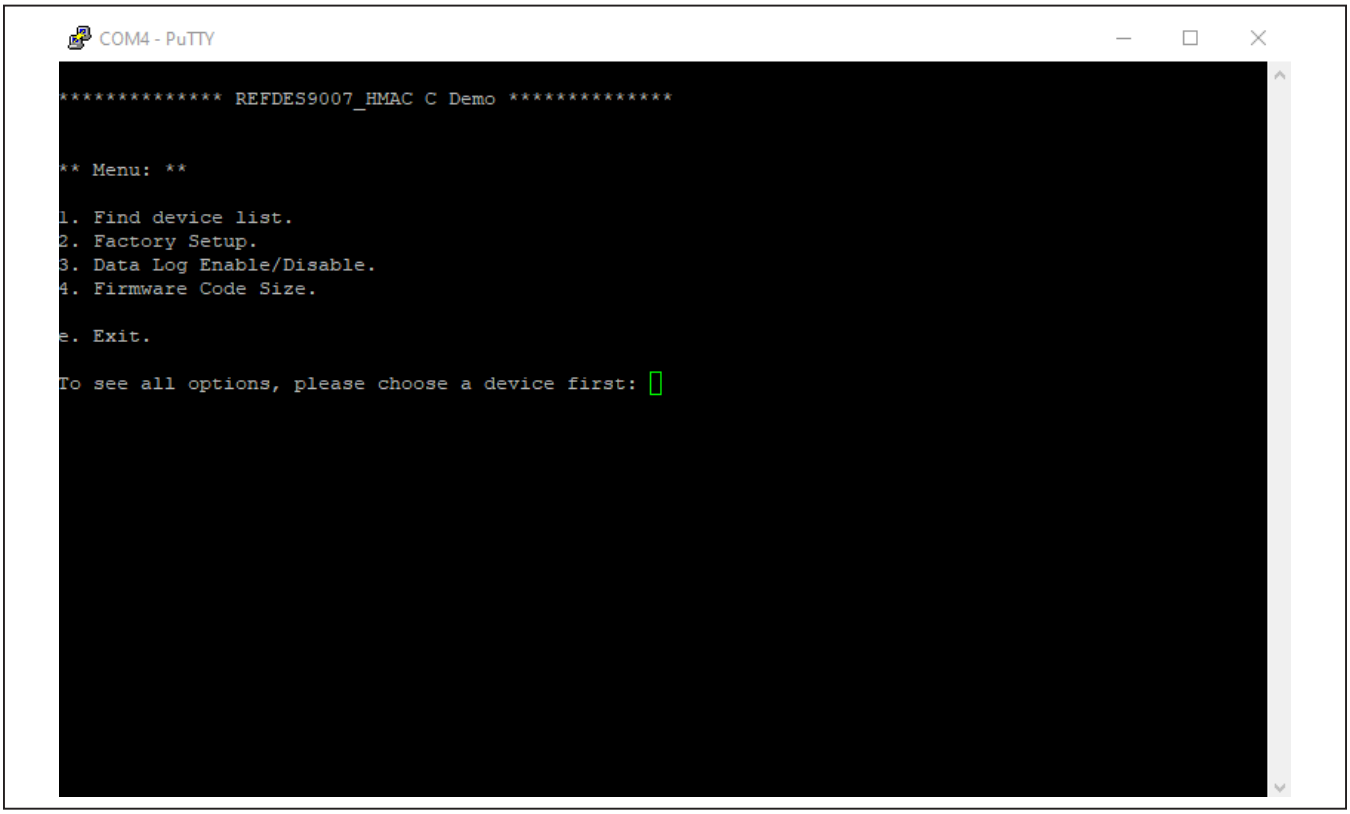


Figure 12. Console Output for MAXREFDES9007 HMAC C Demo.

Detailed Description

Detailed Description of Hardware

Figure 13 shows the main components and connections for the MAXREFDES9007 hardware. This reference design uses the MAX32660 microcontroller as the 1-Wire host. To drive the 1-Wire signal and strength, the MAX32660 uses the GPIO pins, P0_5, and P0_6, respectively. These pins are configured as open-drain outputs to create compatibility with the DS28E36 as the 1-Wire interface is an open-drain design. A low-impedance P-channel MOSFET, Q1, is used to supply the 1-Wire bus with more current when demanded by the DS28E36, such as in an SPU event. The MAXREFDES9007 uses evaluation boards for the MAX32660 and DS28E36 to connect all the main components together. More details for each board are found in the **Design Resources** tab of the MAXREFDES9007 product page.

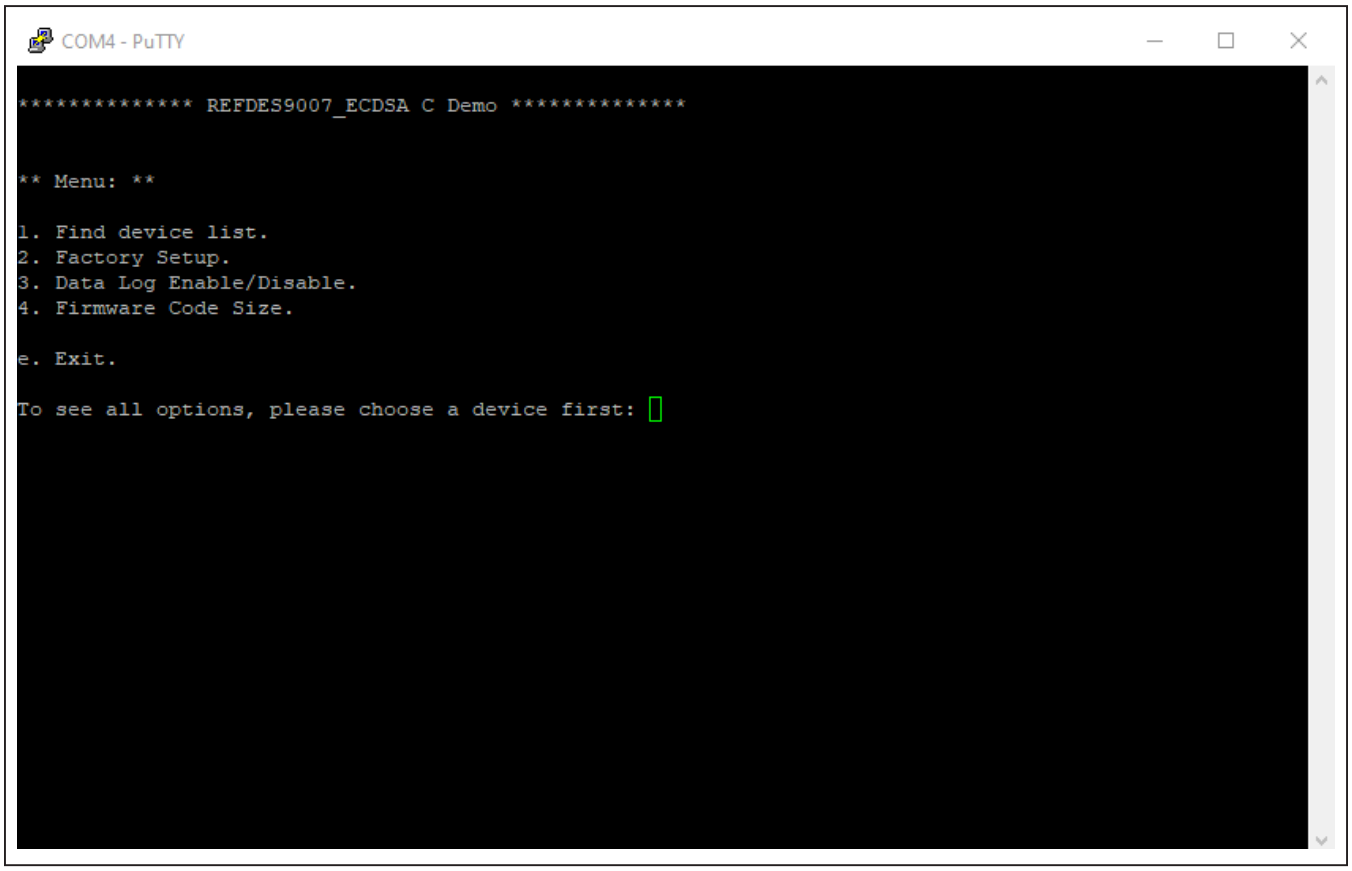


Figure 13. Console Output for MAXREFDES9007 ECDSA C Demo.

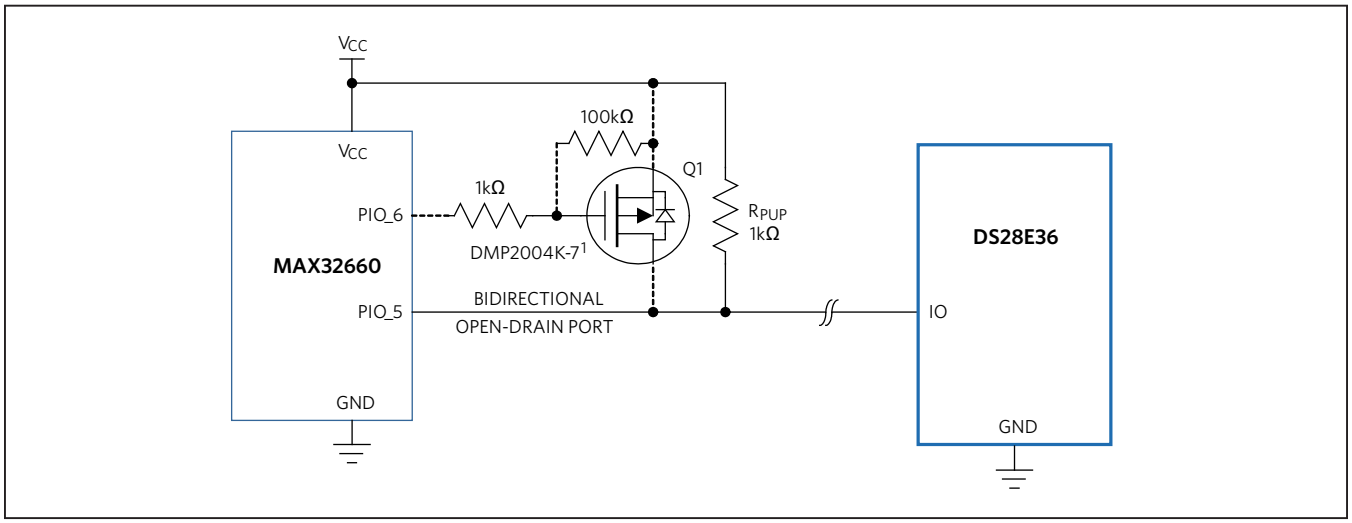


Figure 14. Typical MAXREFDES9007 Hardware Configuration.

Detailed Description of Software

The MAXREFDES9007 software consists of example code written in C used to interface with the hardware. This C-Demo software provides a 1-Wire API developed from bit-banging the MAX32660's GPIO. See Table 1 for an overview of the 1-Wire API. This API allows the MAX32660 to interface with multiple 1-Wire Peripheral devices as it includes all the necessary functions to control host 1-Wire communication for speeds in both standard and overdrive modes. Since it uses a GPIO to drive the 1-Wire line, the MAX32660 needs to operate its GPIO pins accordingly with respect to typical 1-Wire timings. The API does this by utilizing the MAX32660's peripheral timer, Timer1, to generate the precise timings. To set the line high, the microcontroller simply disables the output buffer of the pin corresponding to the 1-Wire bus (P0_5), allowing the external pullup resistor to drive the line high. Similarly, setting the line low can be achieved by simply enabling the output buffer in a low state . Additionally pin 6 (P0_6) is used to enable the P-channel MOSFET by disabling the output buffer that masks that pin, creating a negative voltage difference between the 1-Wire line and the gate of the MOSFET. This allows the flow of current through it, bypassing its parallel resistor and activating the strong pullup. To better understand this concept, see Figure 15. The C-Demo software also provides a DS28E36 API for both the HMAC and ECDSA applications. This API makes it easy to exercise all the features of the DS28E36. Two example programs are provided to demonstrate the complete command sequences used in HMAC and ECDSA applications and to show how much time it takes for each sequence to run (including HMAC and Signature calculations).

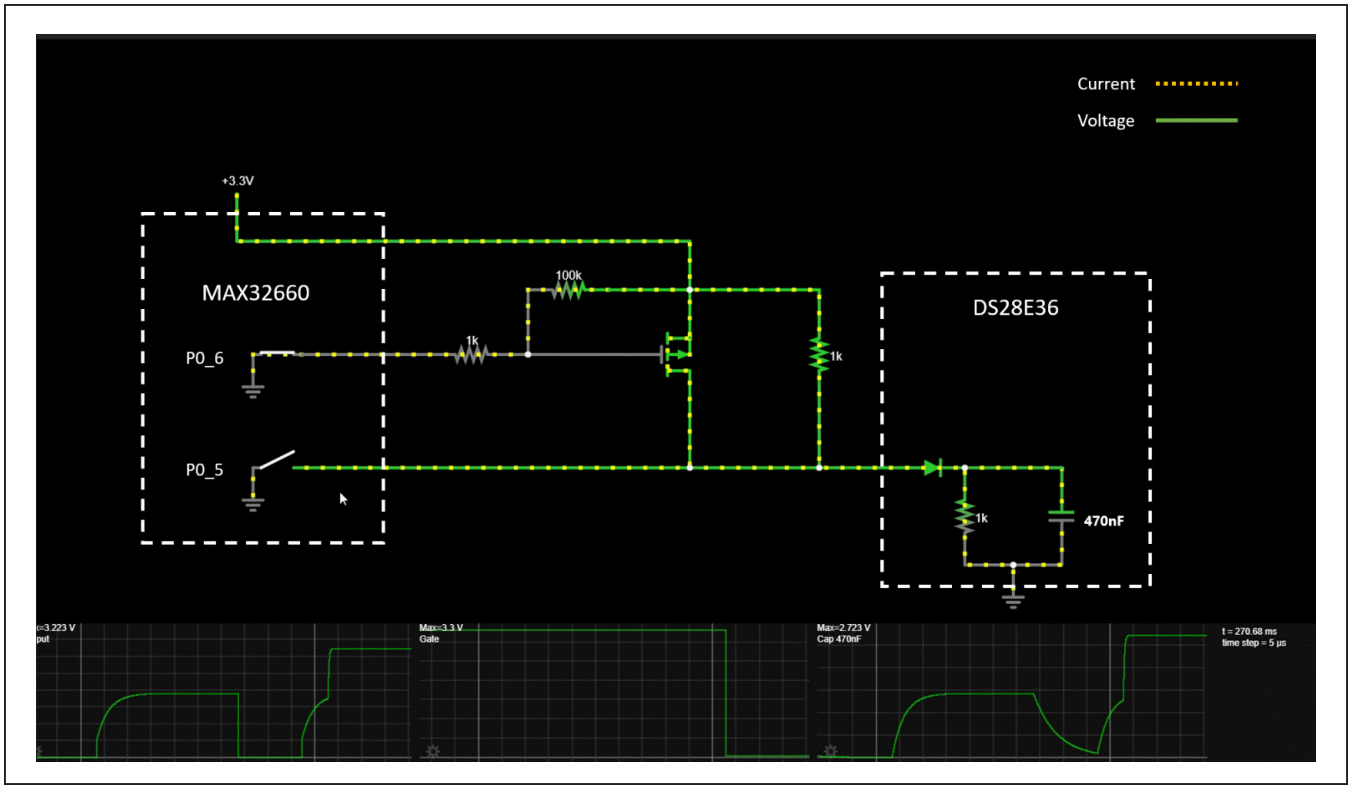


Figure 15. Strong Pullup Assertion Process.

Both the HMAC and ECDSA demos work in a similar manner. When first run, both software welcome the user into an initial menu which allows them to choose a device from the 1-Wire line as well as a Factory Setup' on the selected device. Once a device has been found, selected, and successfully set up, an extended menu is available with options to run different commands and sequences of the DS28E36 secure authenticator.

The MAXREFDES9007 demos make use of the 'TMR0' multiple 32-bit, reloadable timer in 'Capture mode to measure the time between the beginning and end of an HMAC or ECDSA sequence. The timer increments from an initial value until an edge transition occurs on the timer pin (pin 3 of port 0 in alternate function #3). This triggers the 'capture' event, which copies the TMRn_CNT value to the TMRn_PWM.pwm register, resets TMRn_CNT to 0x0000 0001, and continues incrementing.

The capture mode timer period is calculated using the following equation:

$$\text{Capture elapsed time} = \frac{\left(\text{TMR\_PWM} - \text{TMR\_CNT}_{\text{INITIAL\_VALUE}}\right) + \left(\left(\#\ \text{OF ROLLOVER EVENTS}\right) \times \left(\text{TMR\_CMP} - \text{TMR\_CNT}_{\text{INITIAL\_VALUE}}\right)\right)}{f_{CNT\_CLK}}$$

Since no rollover events happen, '# of rollover events = 0'.

$$\text{Capture elapsed time} = \frac{\left(\text{TMR\_PWM} - \text{TMR\_CNT}_{\text{INITIAL\_VALUE}}\right)}{f_{CNT\_CLK}}$$

This software also makes use of the linker-defined symbols such as 'text', 'data', and 'bss' to calculate the project's size. Where 'text' represents the size of the code and constant data that is stored in **FLASH** memory. 'Data' represents the initialized data, which is stored **both** in FLASH and in **RAM** memory and is also added to the size of the FLASH memory alongside the 'text' size. 'Bss' is short for 'Block Started by Symbol' and contains all the uninitialized data that is stored in **RAM** memory. In summary for each of the projects included in the MAXREFDES9007 software package, the FLASH memory can be calculated by adding the 'text' and 'data' sizes. To calculate their RAM is necessary by adding the 'data' and the 'bss' sizes.

The software is compatible with the ADI toolchain, found in the **Design Resources** tab of the MAX32660. This can be directly imported into an Eclipse IDE workspace. See the *Quick Start* section for details on how to set up the C-Demo software.

Table 1. 1-Wire API Overview

FUNCTION	DESCRIPTION
OneWire_Init	Sets up the MAX32660 as a 1-Wire host.
OneWire_ResetPulse	Sends a 1-Wire reset pulse down the 1-Wire bus.
OneWire_WriteByte	Sends a specified byte down the 1-Wire bus.
OneWire_WriteBytePower	Sends a specified byte down the 1-Wire bus and immediately enables the strong pullup (SPU).
OneWire_ReadByte	Requests a byte from the 1-Wire peripheral.
OneWire_Search	Discovers multiple 1-Wire slaves found on the bus.
OneWire_SetSpeed	Sets the 1-Wire master speed between standard and overdrive.

Table 2. DS28E36 Sequences API Overview

FUNCTION	DESCRIPTION
Initial Setup	
find_and_select_device_call	Sends a 1-Wire reset pulse down the 1-Wire bus.
factory_setup	Sends a specified byte down the 1-Wire bus.
dlog_disable	Sends a specified byte down the 1-Wire bus and immediately enables the strong pullup (SPU).
print_size	Requests a byte from the 1-Wire peripheral.
HMAC Sequences	
compute_and_read_page_authentication_ Sequence	Collects and computes the necessary components to run the ‘compute and read page authentication’ command.
authenticated_write_memory_Sequence	Collects and computes the necessary components to run the ‘authenticated write memory’ command.
encrypted_authenticated_write_memory_ Sequence	Collects and computes the necessary components to run the ‘encrypted authenticated write memory’ command
encrypted_read_memory_Sequence	Collects and computes the necessary components to run the ‘encrypted read memory’ command.
ECDSA Sequences	
compute_and_read_page_authentication_ Sequence	Collects and computes the necessary components to run the ‘compute and read page authentication’ command.
authenticated_write_memory_Sequence	Collects and computes the necessary components to run the ‘authenticated write memory’ command.
encrypetd_authenticated_write_ memory_Sequence	Collects and computes the necessary components to run the ‘encrypted authenticated write memory’ command.
encrypted_read_memory_Sequence	Collects and computes the necessary components to run the ‘encrypted read memory’ command.
Supporting Sequences	
write_buffer_command	Collects and computes the necessary components to run the ‘write buffer’ command.
read_buffer_command	Collects and computes the necessary components to run the ‘read buffer’ command.
write_memory_command	Collects and computes the necessary components to run the ‘write memory’ command.
read_memory_command	Collects and computes the necessary components to run the ‘read memory’ command.
read_page_protection_command	Collects and computes the necessary components to run the ‘read page protection’ command.
set_page_protection_command	Collects and computes the necessary components to run the ‘set page protection’ command.
decrement_counter_command	Collects and computes the necessary components to run the ‘decrement counter’ command.
read_random_number_generator_ command	Collects and computes the necessary components to run the ‘read random number generator’ command.
compute_and_lock_secret_command	Collects and computes the necessary components to run the ‘compute and lock secret’ command.
generate_ecc256_key_pair_command	Collects and computes the necessary components to run the ‘generate ECC256 key pair’ command.
authenticate_ecdsa_public_key_command	Collects and computes the necessary components to run the ‘authenticate ECDSA public key’ command.

Table 2. DS28E36 Sequences API Overview (continued)

FUNCTION	DESCRIPTION
Utility Methods	
secrets_setup	
hmac_message	Collects the necessary data to generate an HMAC message for multiple purposes.
write_memory_message	Collects the necessary data to generate an HMAC message for the write memory sequences.
encrypted_read_memory_message	Collects the necessary data to generate an HMAC message for the encrypted read memory sequences.
hash_compare	Compares hashes to validate HMACs.
select_validation	
fill_data	Fills page data buffers with requested data.
print_data	Prints a specified number of data bytes.
ECDSA Utility Methods	
cert_sig_message	Collects the necessary data to generate a certificate signature message for multiple purposes.
sw_verifyECDSASignature	Collects the necessary data to validate an ECDSA signature for multiple purposes.
sw_computeECDSASignature	Collects the necessary data to compute an ECDSA signature for multiple purposes.
sw_verifyECDSACert	Collects the necessary data to Verify an ECDSA certificate for multiple purposes.
sw_computeECDHKey	Collects the necessary data to compute an ECDH signature for multiple purposes.

Table 3. DS28E36 Low-Level API Overview

FUNCTION	DESCRIPTION
rom_cmd_bundle	Constructs the ROM-level command to be sent based on the selected ROM cmd.
write_memory	Sends the ‘write memory’ command down the 1-Wire bus and calculates the CRCs.
read_memory	Sends the ‘read memory’ command down the 1-Wire bus and calculates the CRCs.
write_buffer	Sends the ‘write buffer’ command down the 1-Wire bus and calculates the CRCs.
read_buffer	Sends the ‘read buffer’ command down the 1-Wire bus and calculates the CRCs.
read_page_protection	Sends the ‘read_page_protection’ command down the 1-Wire bus and calculates the CRCs.
set_page_protection	Sends the set page protection’ command down the 1-Wire bus and calculates the CRCs.
decrement_counter	Sends the ‘decrement counter’ command down the 1-Wire bus and calculates the CRCs.
read_rng	Sends the ‘read random number generator’ command down the 1-Wire bus and calculates the CRCs.
encrypted_read_memory	Sends the ‘encrypted read memory’ command down the 1-Wire bus and calculates the CRCs.
compute_and_read_page_authentication	Sends the ‘compute and read page authentication’ command down the 1-Wire bus and calculates the CRCs.
authenticated_sha_write_memory	Sends the authenticated sha write memory’ command down the 1-Wire bus and calculates the CRCs.
compute_and_lock_sha_secret	Sends the compute and lock sha secret’ command down the 1-Wire bus and calculates the CRCs.
generate_ecc_key_pair	Sends the ‘generate ECC key pair’ command down the 1-Wire bus and calculates the CRCs.

Table 3. DS28E36 Low-Level API Overview (continued)

FUNCTION	DESCRIPTION
authenticate_ecdsa_public_key	Sends the 'authenticate ECDSA public key' command down the 1-Wire bus and calculates the CRCs.
authenticated_ecdsa_write_memory	Sends the 'authenticated ECDSA write memory' command down the 1-Wire bus and calculates the CRCs.
find_device_list	Creates a list of the available devices on the 1-Wire bus by running the 1-Wire 'search' and 'next' commands.
select_device	Selects a device from the 'find_device_list' of found devices with which to communicate.
print_devices_list	Prints a list of the found devices by the 'find_device_list' command.
readcrc16	Reads two CRC bytes during the 1-Wire communication sequences.
calculateCrc16Byte	Calculates a CRC16 byte at a time.
calculateCrc16Block	Calculates a block of CRC16 bytes by calling the 'calculateCrc16Byte'.
comp_CRC16	Compares the calculated CRC16 block with the received CRC16 block.
protection_validation	Validates the selected protection by the user is valid in certain instances.
pageCheck	Checks that the page number provided for the commands is setup with the correct format.
dataCheck	Checks that the page data provided for the commands is setup with the correct format.
print_protect	Gets the number of the associated protection passed on certain commands and prints its alphanumeric equivalent (used for the user interface).

Example Program's User Interface Overview

The following section walks through how to run the example programs, their features, and outputs.

For both demos, the process for selecting and setting up the device are exactly the same so their use will be demonstrated using the HMAC demo. The first step is to select **Search** and select a device on the 1-Wire line. Next, a factory setup for the selected device is required since it generates the necessary SHA2 secrets as well as sets the required protections for the demos to work. After these steps have been completed, an Extended Menu unlocks in order for the user to run the cryptographic command sequences of the DS28E36.

To start using the demos, the user must select option 1, **Find Device List** in the Main menu (or starting menu) and then select the desired device. Press **1** followed by **Enter** to get into the **Find device list** submenu as shown in Figure 16.

The **Find device list** submenu shows the number of DS28E36 devices found on the 1-Wire line and displays their ROM IDs as well. They are highlighted in green in Figure 17. To select a device, enter the number of desired devices based on a list of displayed devices (highlighted in yellow in Figure 15) and click **Enter**. After these actions are completed, the program returns to the Main menu.

Note 1: A different device can be selected at any time but a Factory Setup must be run right after the selection.

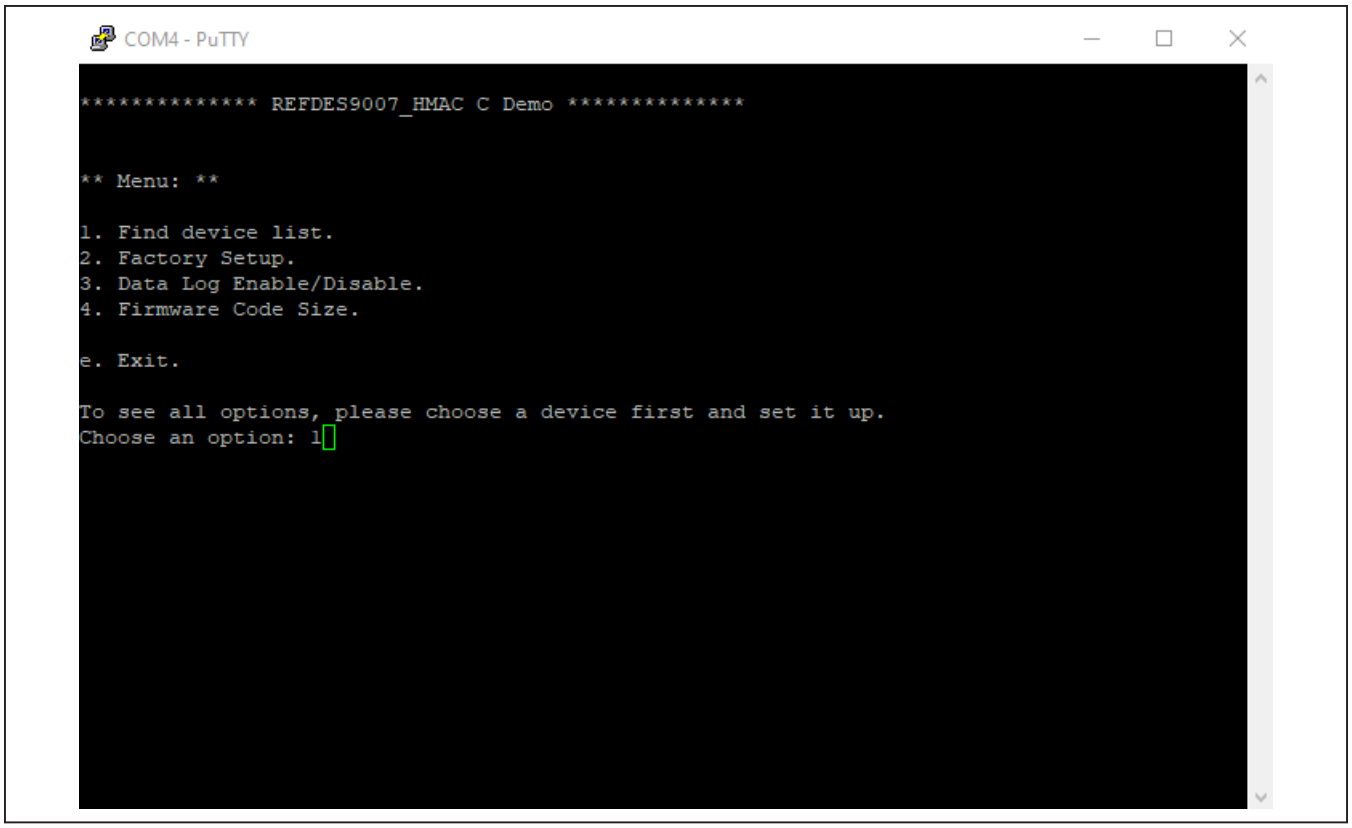


Figure 16. Selecting a Device in the HMAC Demo.

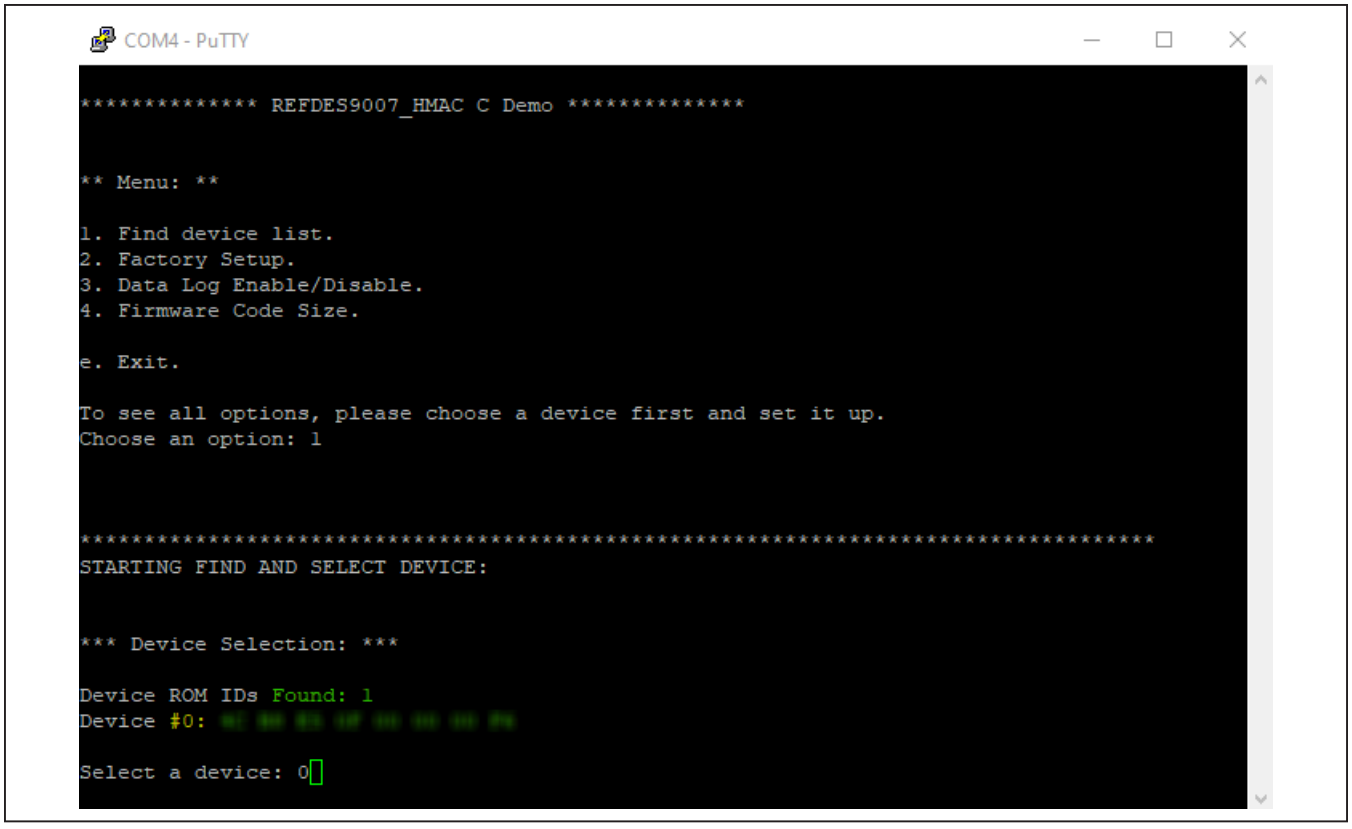


Figure 17. Find Device List Submenu in HMAC Demo.

After a device has been selected, a “Factory Setup” is required (Figure 18). This can be achieved by selecting the second option on the main menu **Factory Setup**. By selecting this option, the software will issue the required commands to generate the necessary Unique Secrets and Page Protections. This will also store any required user data in the device.

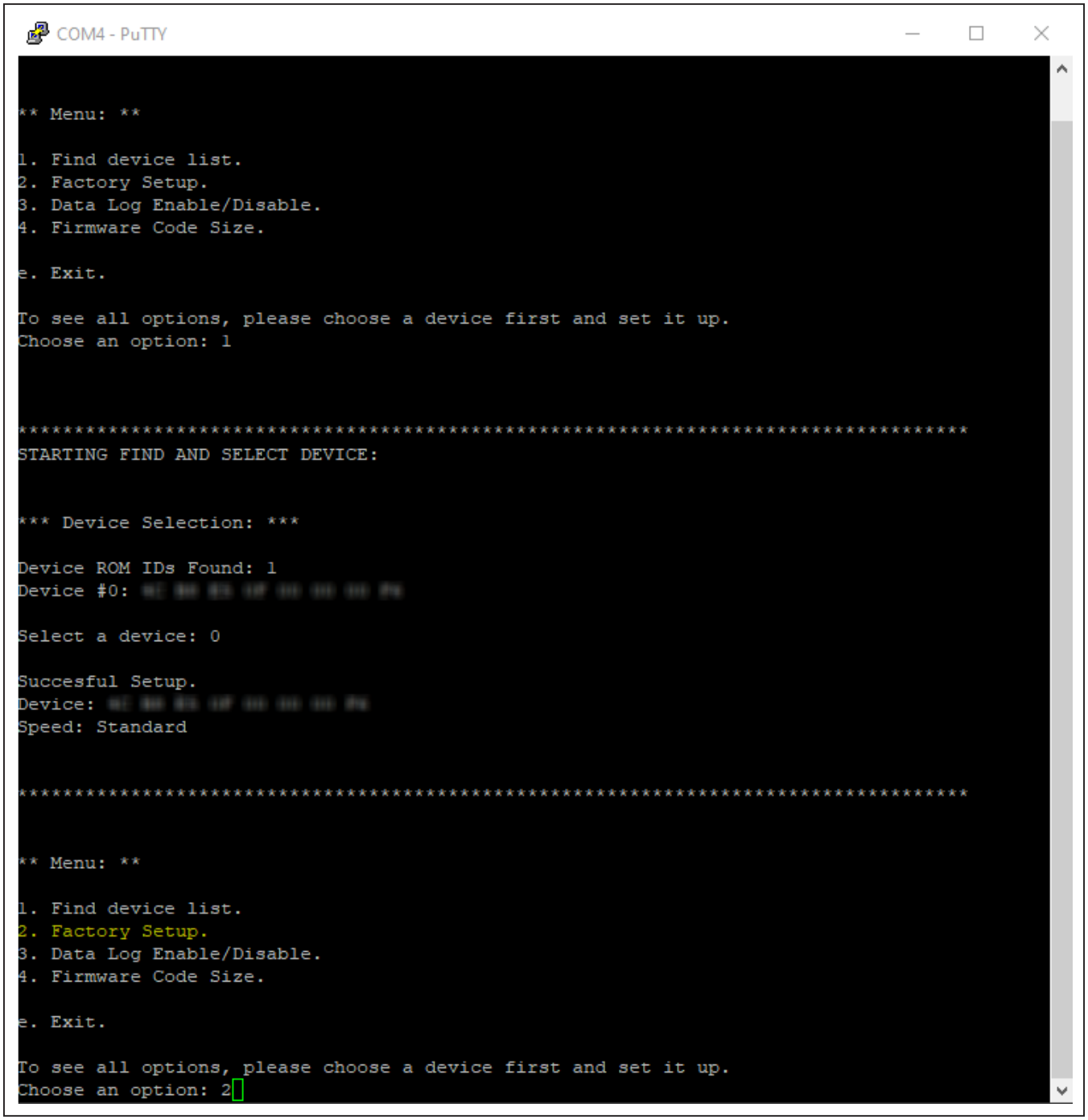


Figure 18. HMAC Demo Factory Setup Selection.

Note 2: *Text logs of the HMAC Demo’s Factory Setup command, as well as for all the other HMAC and ECDSA commands, can be found inside the MAXREFDES9007-V1.0.0.zip software package. The source code contains a directory with text logs of the outputs from the MAXREFDES9007 HMAC Demo and the MAXREFDES9007 ECDSA Demo. It also contains a document describing the data path in the system for each cryptographic command as well as other useful resources to better understand the hardware and software elements of this reference design.*

If the program is to setup incorrectly or the device has already different protections, the Factory Setup will return an error and exit the program.

Once the device has been selected and successfully set up, an extended menu will appear right after the Factory Setup has been completed as shown in Figure 19 (highlighted on green). Before discussing HMAC Sequences, it is important to note the **Data Log Enable/Disable** mode or Debugging mode. The data log is enabled by default., By disabling it, this allows the program to measure the real time it takes to run each of the sequences. Figure 20 shows the log when **Data Log Enable/Disable** is selected.

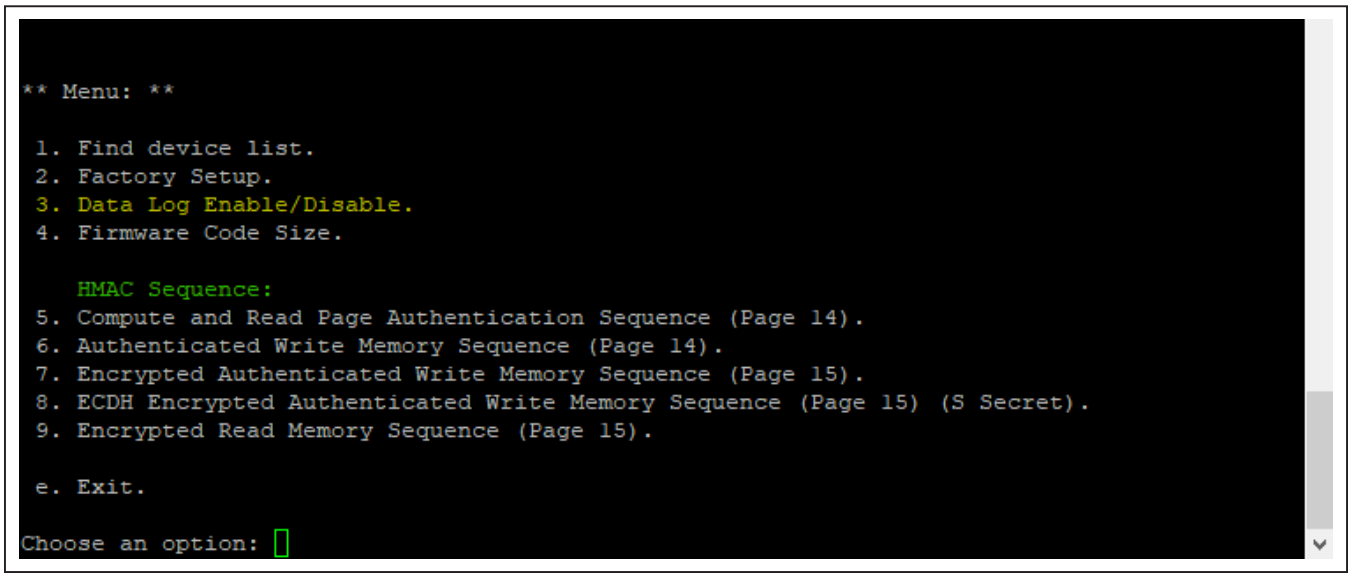


Figure 19. HMAC Demo Extended Menu.

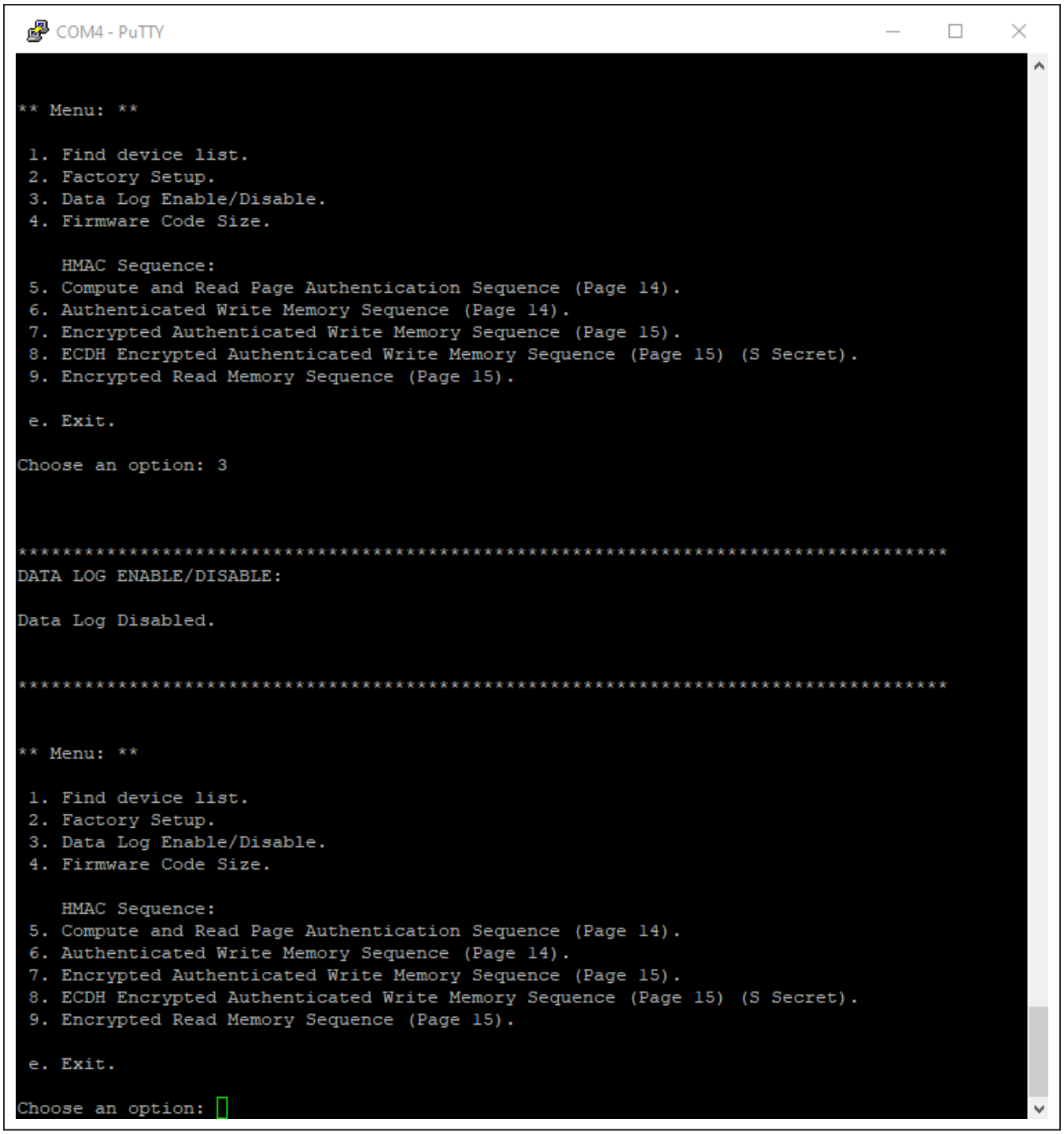


Figure 20. HMAC Demo Data Log Enable/Disable Log.

Additionally, we have the Firmware Code Size menu option. By selecting it, the program displays the amount of FLASH and RAM memory used. See the *Detailed Description of Software* section of this document for more information on how they are calculated. Figure 21 shows the log of option 4, **Firmware Code Size**.

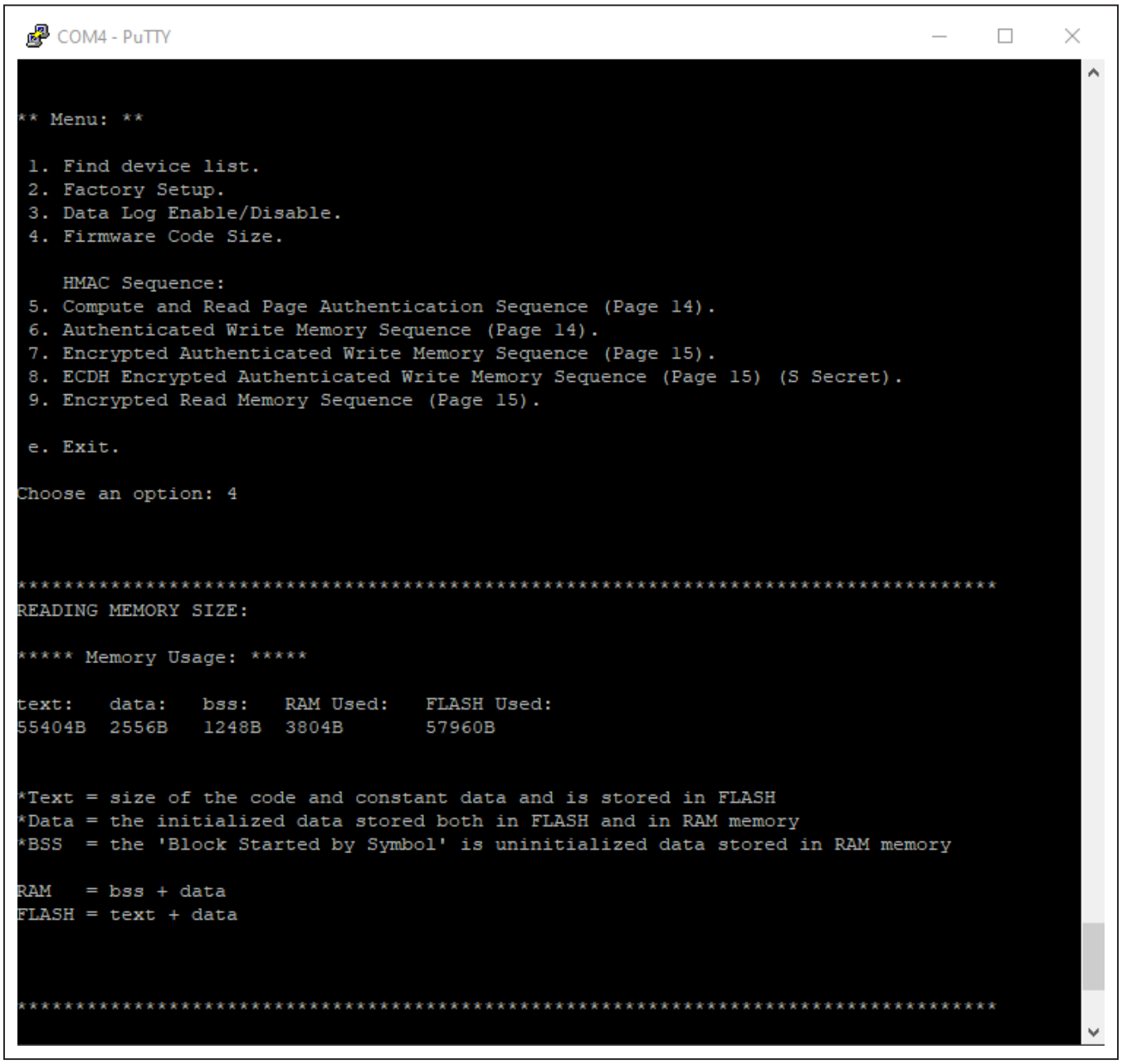


Figure 21. HMAC Demo Firmware Code Size Log.

Lastly, we demonstrate how one of the cryptographic commands from the “extended menu” works to measure its running time with and without debugging data log.

Figure 22 shows the HMAC Encrypted Read Memory command sequence that is run when selecting menu option 9 of the Extended menu. Note that in debugging mode (with Data Log enabled), the Decryption HMAC as well as the HMAC message ingredients are displayed. This **SHOULD NOT BE THE CASE** in real-life applications. In Figure 22 and Figure 23, the real time measurement is highlighted in yellow. All sequences will display a success message after being run successfully . This is highlighted in green inn Figure 22 and Figure 23. To get more information on the individual HMAC and ECDSA command sequences and their Data Logs, see Note 2.

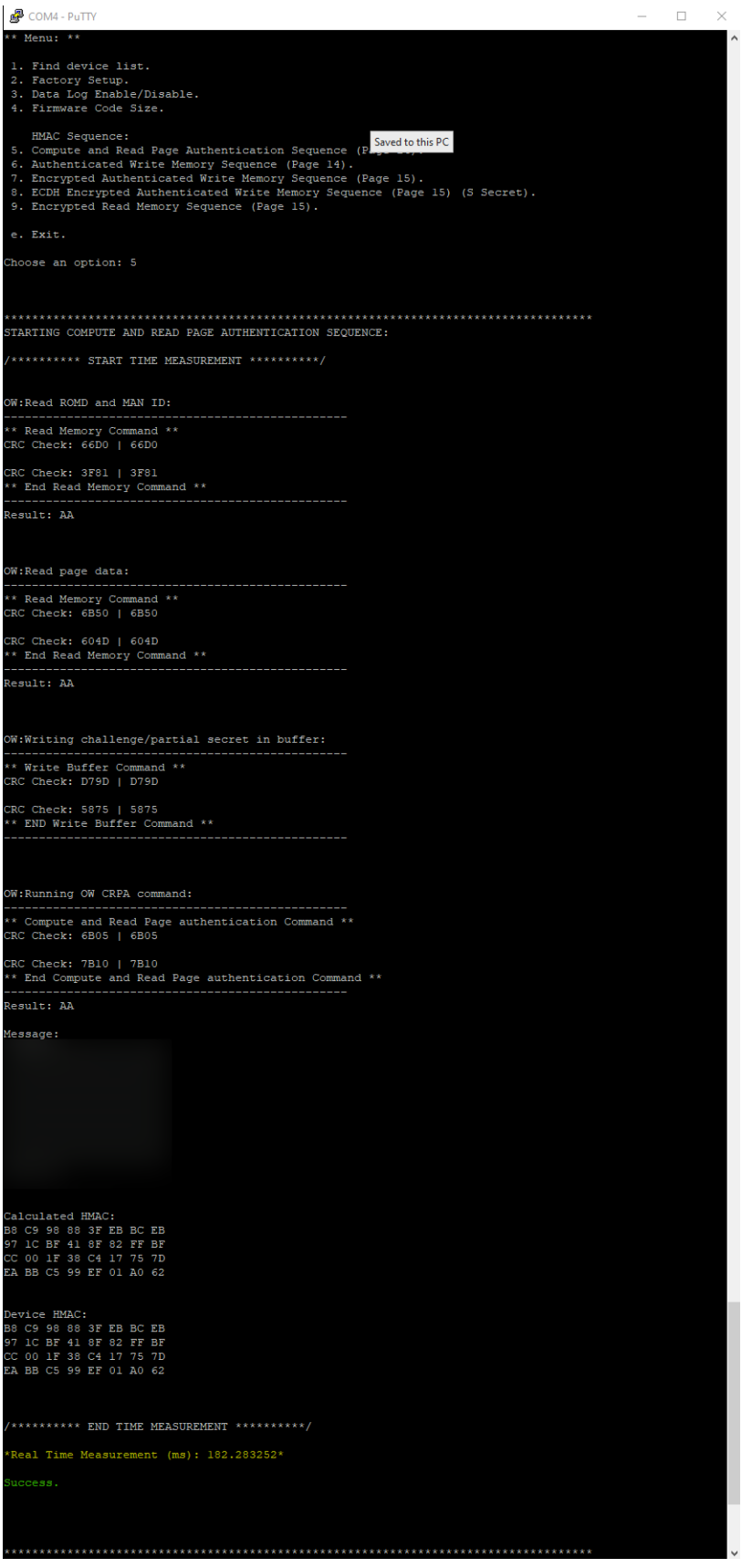


Figure 22. HMAC Encrypted Read Memory with Data Log Enabled.

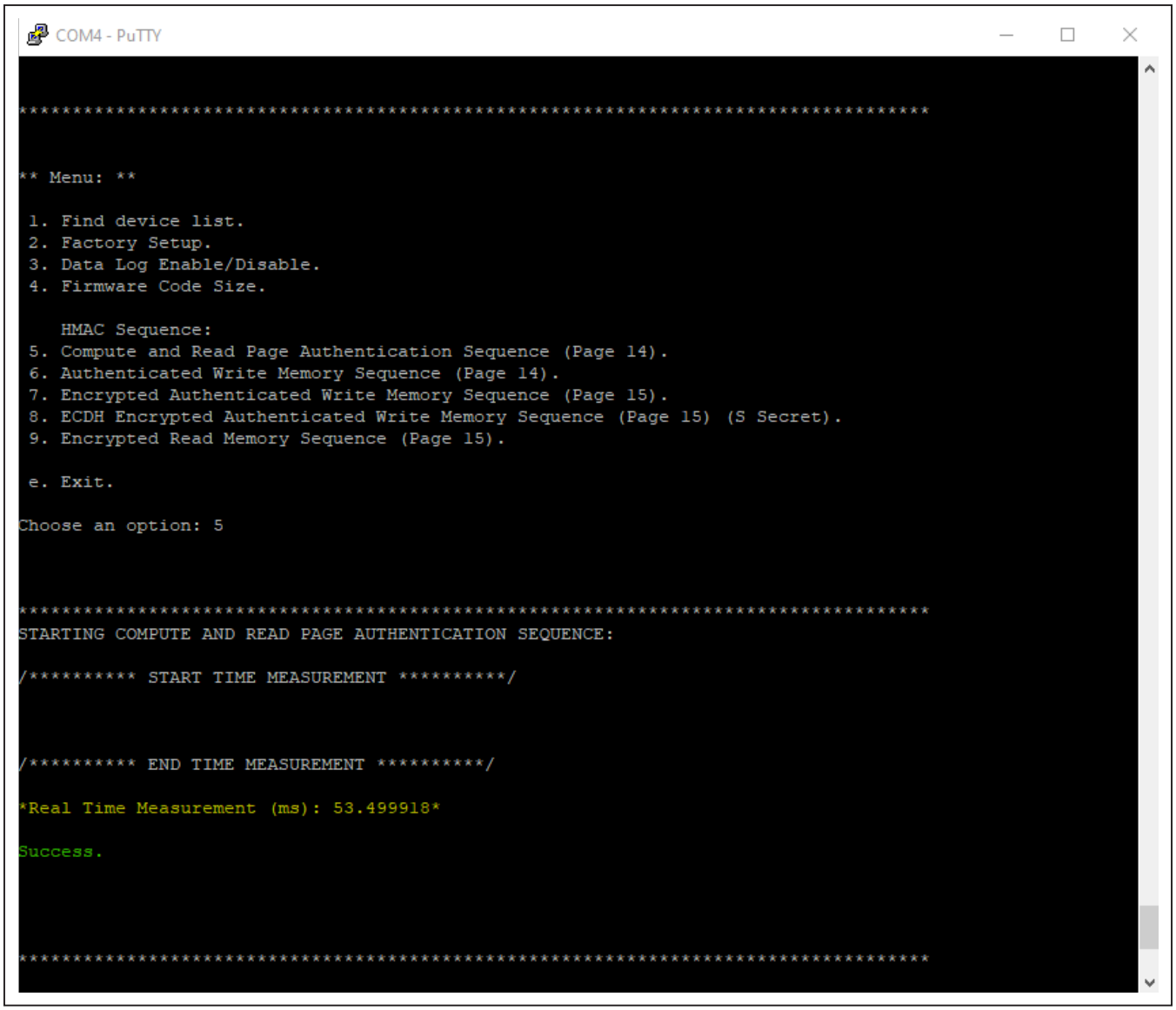


Figure 23. HMAC Encrypted Read Memory with Data Log Disabled.

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	4/22	Initial release	—
1	7/23	Updated Overview, Features, Applications, Introduction, and Detailed Description; Updated Figure 2, Figure 6, Figure 7, Figure 9, Figure 17, Figure 18, and Figure 22; Table 1 and Table 2; Deleted Text Logs; Deleted Appendix A	1, 2, 3, 5, 7, 10, 11, 12, 13, 16-23, 24, 27, 29-99