AN3407

Using Matrix Keypad with AVR® Devices

Features

- Basic Overview and Example of Matrix Keypad Operation
- Easy Implementation to Other Applications
- Advanced Low-Power Implementation Example
  - Fully interrupt driven operation
  - Power-Down Sleep mode
  - Well suited in battery-powered applications
- Software Button Debouncing
- Simple Passcode Check Implementation
- LED Passcode Validation Indicator

Introduction

Author: Amund Aune, Microchip Technology Inc.

This application note shows how a general keypad application can be implemented with tinyAVR® and megaAVR® devices. A conceptual overview of the operation of a matrix keypad and two demo applications are presented. One demo shows a simple implementation of a keypad, while the second demo uses more advanced features to make the application more efficient and use less power.

The examples in this application note may easily be changed to interface a smaller or larger matrix keypad, and are easy to implement into another application. The application may be used in all implementations using a matrix keypad, such as access control keypads, keyboards, or remote controls.

The code examples are available through Atmel START:

- Using Matrix Keypad with AVR Devices - Basic
  - https://start.atmel.com/#example/Atmel%3AApplication_AVR_Examples%3A1.0.0%3A%3AApplication%3AUsing_Matrix_Keypad_with_AVR_Devices_-_Basic%3A
- Using Matrix Keypad with AVR Devices - Advanced
  - https://start.atmel.com/#example/Atmel%3AApplication_AVR_Examples%3A1.0.0%3A%3AApplication%3AUsing_Matrix_Keypad_with_AVR_Devices_-_Advanced%3A

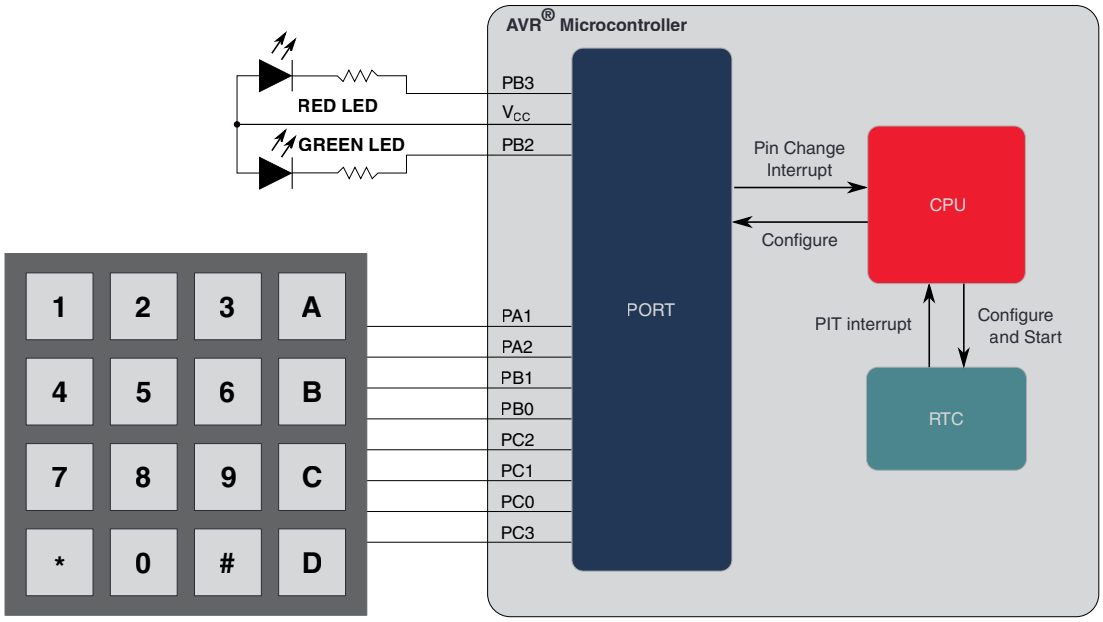
The code examples are also available through GitHub:

Table of Contents

1. Block Diagram

The block diagram below shows an overview of the advanced application example using a 4x4 keypad with the ATtiny1627 Curiosity Nano development board. It shows how the application interacts with the peripherals and CPU of the tinyAVR® 2 device. The keypad and LEDs may be connected to any of the General Purpose Input/Output (GPIO) pins, but in this example, they are connected physically in a row on the ATtiny1627 Curiosity Nano to simplify connection.

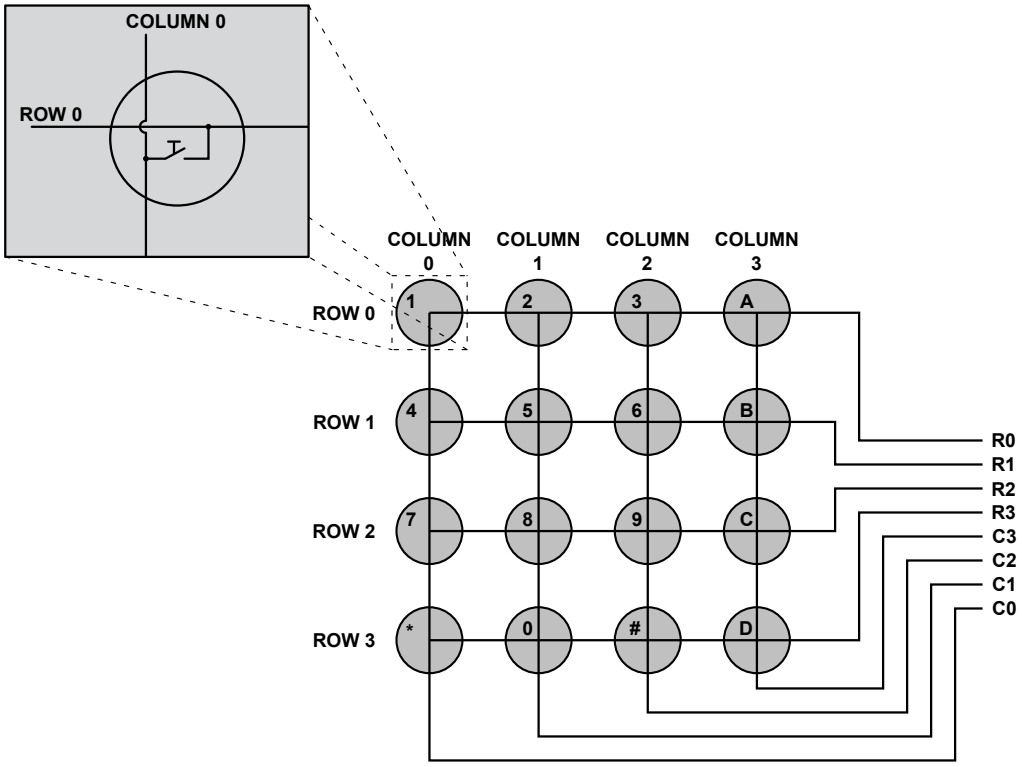
Figure 1-1. Block Diagram



2. Theory of Operation

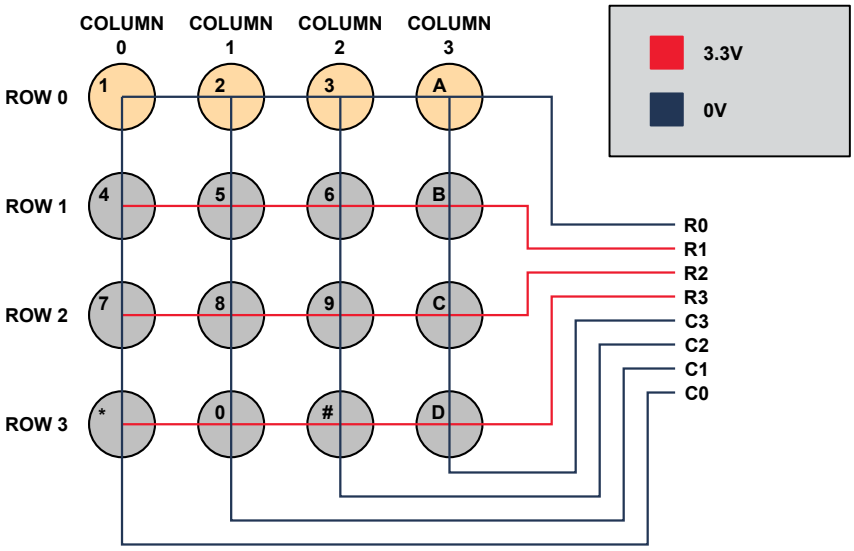
Matrix keypads consist of a grid of buttons with corresponding wires that may be read and interpreted by a microcontroller. As the figure below shows, the number of pins needed by the microcontroller is determined by the number of rows and columns in the button grid, where one wire is needed per row and per column. When a button is pressed, a connection between the corresponding row and column is created. This connection enables the keypad to be interpreted by the microcontroller.

Figure 2-1. Matrix Keypad Schematics



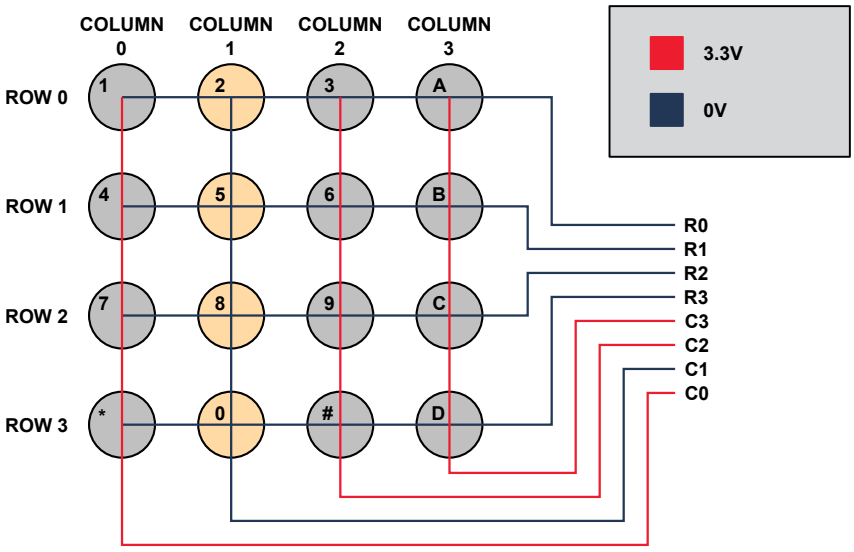
There are multiple ways of scanning a matrix keypad. The simple method used in the example applications is scanning one full axis (that is, the row or the column) at a time. This is done by setting up one of the axes to output a known voltage level and reading the voltage level of the wires of the other axis to find out which wire was pulled to that voltage. Figure 2-2 shows where the columns output 0V and the rows are connected to 3.3V through a pull-up resistor. ROW 0 is shown to be pulled down to 0V because of a connection to the columns, so the pressed button may therefore be assumed to be in ROW 0.

Figure 2-2. Row Scan



After the first axis has been scanned, the setup must be flipped to read the other axis. This is shown in Figure 2-3 where COLUMN 1 is shown to be pulled down to 0V because of a connection to the rows, so the pressed button may therefore be assumed to be in COLUMN 1.

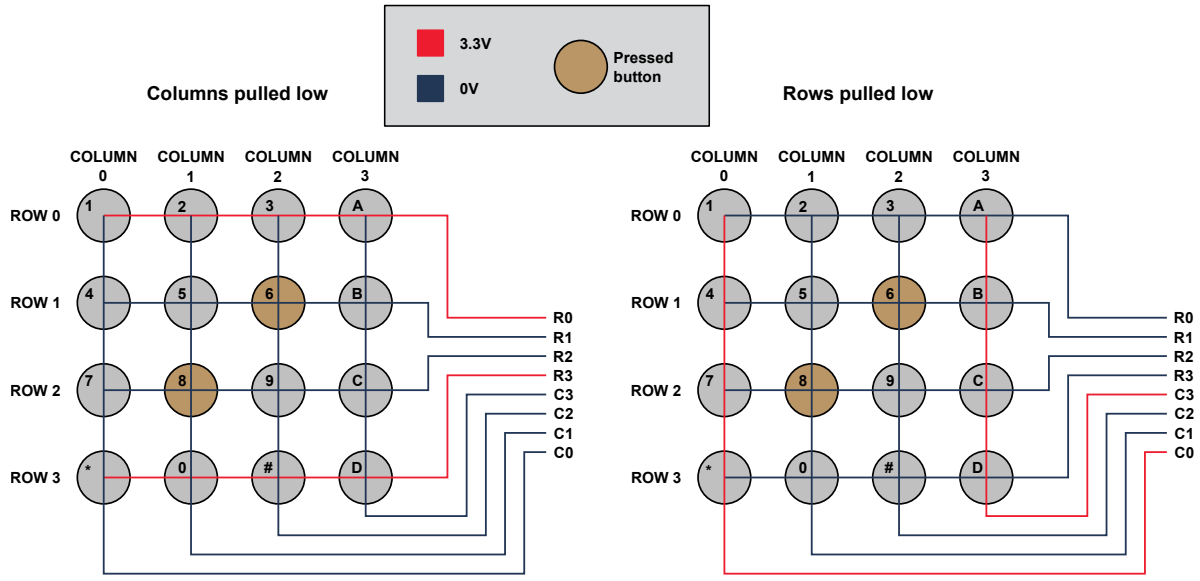
Figure 2-3. Column Scan



When both axes have been scanned, the coordinates of the pressed button in the grid are known, and the button press may be recorded and acted upon. In this example, since ROW 0 and COLUMN 1 are known to include the pressed button, it is understood that button 2 is pressed.

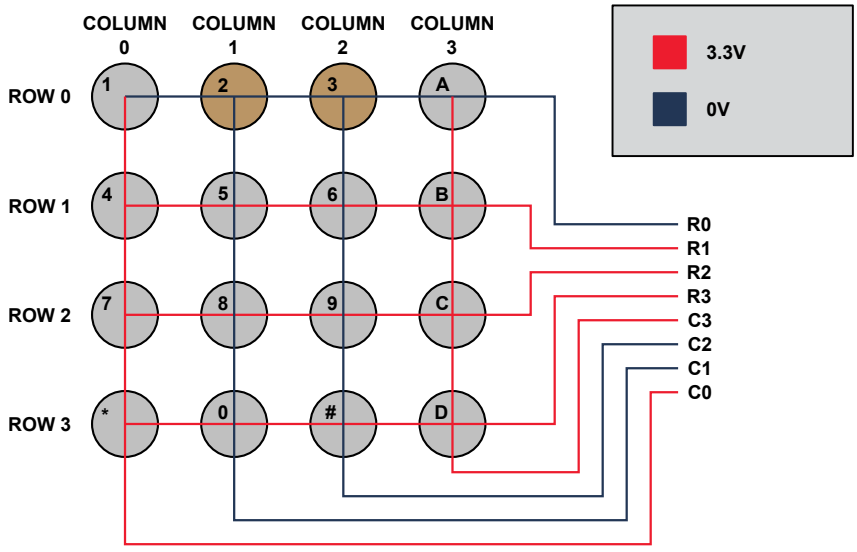
Although this method is simple, the problem with it is that it can only handle one simultaneous button press at a time. Figure 2-4 shows which wires are pulled low when each of the axes are pulled low. As shown in the figure, ROW 1 and 2 and COLUMN 1 and 2 are pressed, leading to four buttons being recorded while only two buttons are really pressed. This effect is called ghosting.

Figure 2-4. Ghosting



Another method is to pull the axis low one wire at a time, reading all the pins on the other axis for each time to find the intersections. This is shown in the figure below, where ROW 0 is driven low, and COLUMN 1 and COLUMN 2 are pulled low by buttons 2 and 3 being pressed. The advantage of this method is the ability to read two button presses at a time, while the disadvantage is that it may add complexity to the application.

Figure 2-5. Single Axis Scan



3. Demo Operation

In these demo applications, a keypad is connected to the ATtiny1627 Curiosity Nano development board to show how an access code can be read and checked for validity. The pin code is written by pressing the alphanumerical buttons on the keypad, reset by pressing star (*), and check for validity by pressing pound (#). The green LED flashes when the pin is valid and the red LED flashes if the code is incorrect. The red LED also flashes if the preset maximum number of characters (20) is reached.

Two demos are presented where one, the basic keypad example, implements the above functionality in the simplest way possible. It is presented to create an understanding of how the keypad works and how one may interface with it. The other demo, the advanced keypad example, implements more advanced features of the tinyAVR® device to improve the efficiency of the application in various ways.

3.1 Hardware Prerequisites

- Microchip ATtiny1627 Curiosity Nano Evaluation Kit
  - https://www.microchip.com/developmenttools/ProductDetails/DM080104
- A 4x4 Matrix Keypad
- Two LEDs
- Two resistors
- Micro-USB cable (Type-A/Micro-B)

3.2 Software Prerequisites

- Atmel Studio 7 (Version 7.0.2397 or later)
- Atmel Studio ATtiny_DFP version 1.4.308 or later

3.3 Running the Example

- Connect the keypad to the ATmega1627 Curiosity Nano as follows:
  - Row 0: PB0
  - Row 1: PB1
  - Row 2: PA2
  - Row 3: PA1
  - Column 0: PC3
  - Column 1: PC0
  - Column 2: PC1
  - Column 3: PC2
- Connect the red LED to PB3 and the green LED to PB2
- Connect the ATtiny1627 Curiosity Nano to a computer using a USB cable
- Download the application as explained in Section 6. Get Code Examples from Atmel START and program it to the ATtiny1627 Curiosity Nano
- The pin code “123ABC” is programmed to be the passcode. Try pressing this code followed by pound (#) to observe the green LED flashing. Try a different code to observe the red LED flashing.

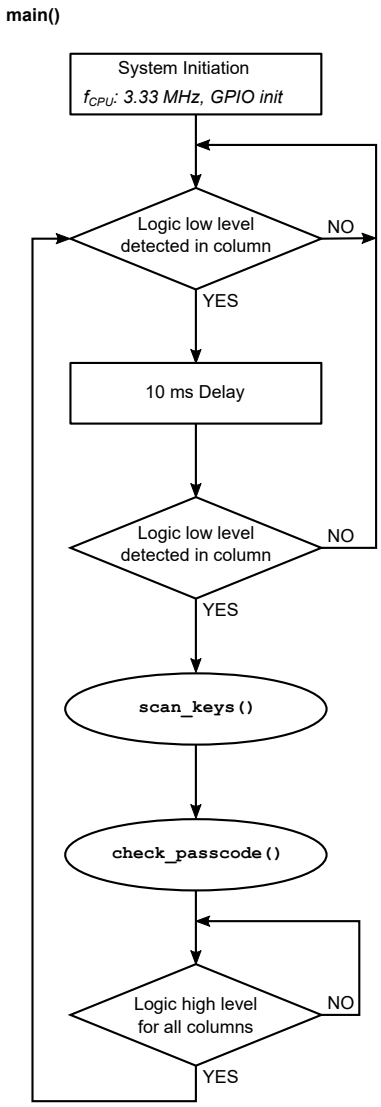
4. Source Code Overview

4.1 Basic Operation

This section presents a basic keypad application where a bare minimum of peripherals and features have been used. It is intended to show the basics of interfacing a keypad with an AVR® microcontroller.

The figure below shows the application main function. It initiates the microcontroller to the default main clock setting and configures the GPIO pins connected to the LED and to the keypad. The keypad pins are initiated with the rows as outputs with a low-output value and the columns as inputs with the internal pull ups enabled.

Figure 4-1. Basic Operation Main Function Overview



As shown in the figure, the device polls the column pins to see whether any of them have been pulled down because of a connection to the row. If a connection is detected, the device busy-waits for 10 ms before checking the column pins again. If one of the column pins are still pulled to the row voltage, the key press is considered valid. This sequence is a simple implementation of a button debouncing technique.

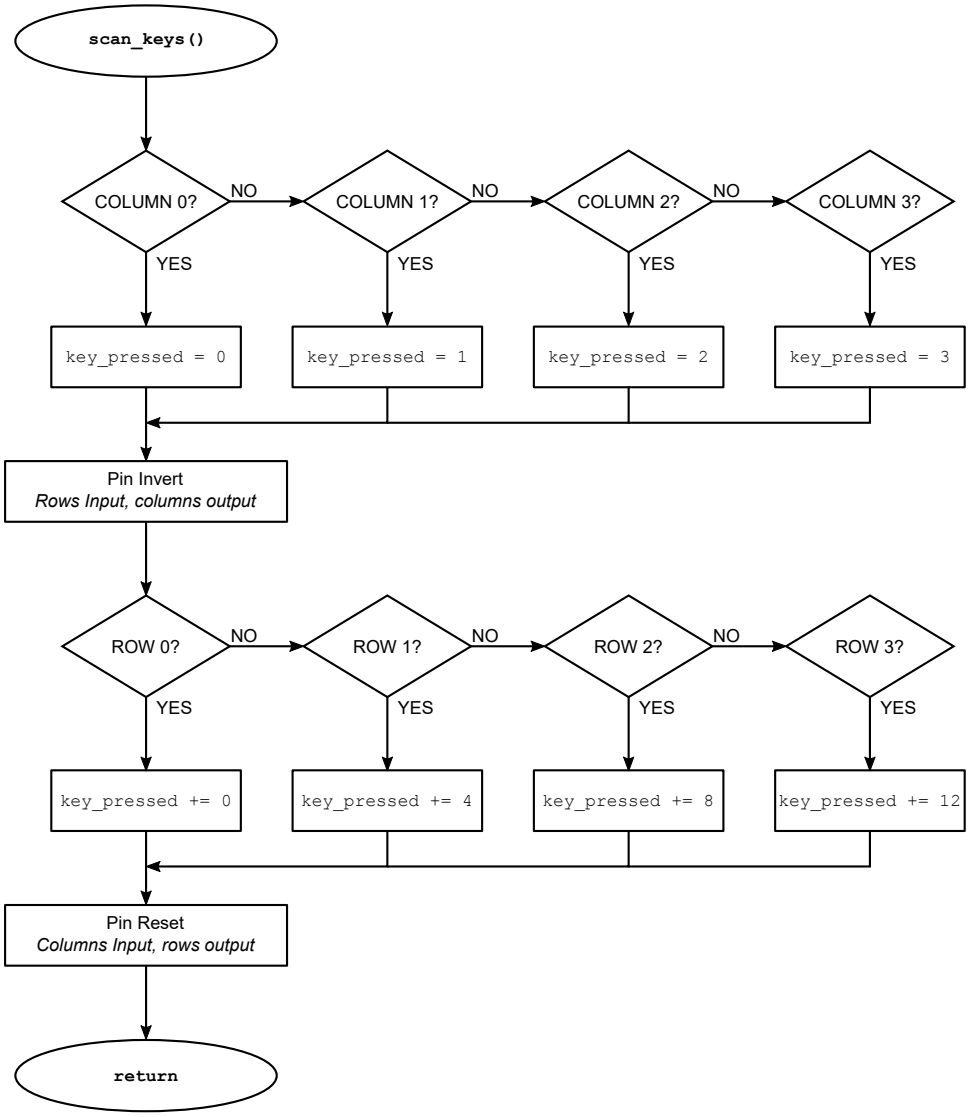
When a valid key press has been detected, the functions `scan_keys()` and `check_passcode()` are run. The two functions are discussed in the following sections.

When the key press has been properly handled by the two functions, the device polls for the button to be released. This is done to prevent one press to be registered multiple times. When the button is released, the device returns to polling for a new key press.

4.1.1 Key Scan

The `scan_keys()` function starts by recording which of the columns contains the pressed button. Each column is assigned a value which is stored in the `key_pressed` variable as shown in the figure below. When the column has been recorded, the row and column pins are inverted in the sense that the rows become inputs and the columns become outputs.

Figure 4-2. Key Scan Function Overview



The `key_pressed` variable is incremented by a value depending on which row is detected to contain the pushed button. If, for example, button **B** is pressed, COLUMN 3 would be recorded first, with a value of 3 being stored into the `key_pressed` variable. Then ROW 1 would be recorded, which would add 4 to the variable, giving the variable a

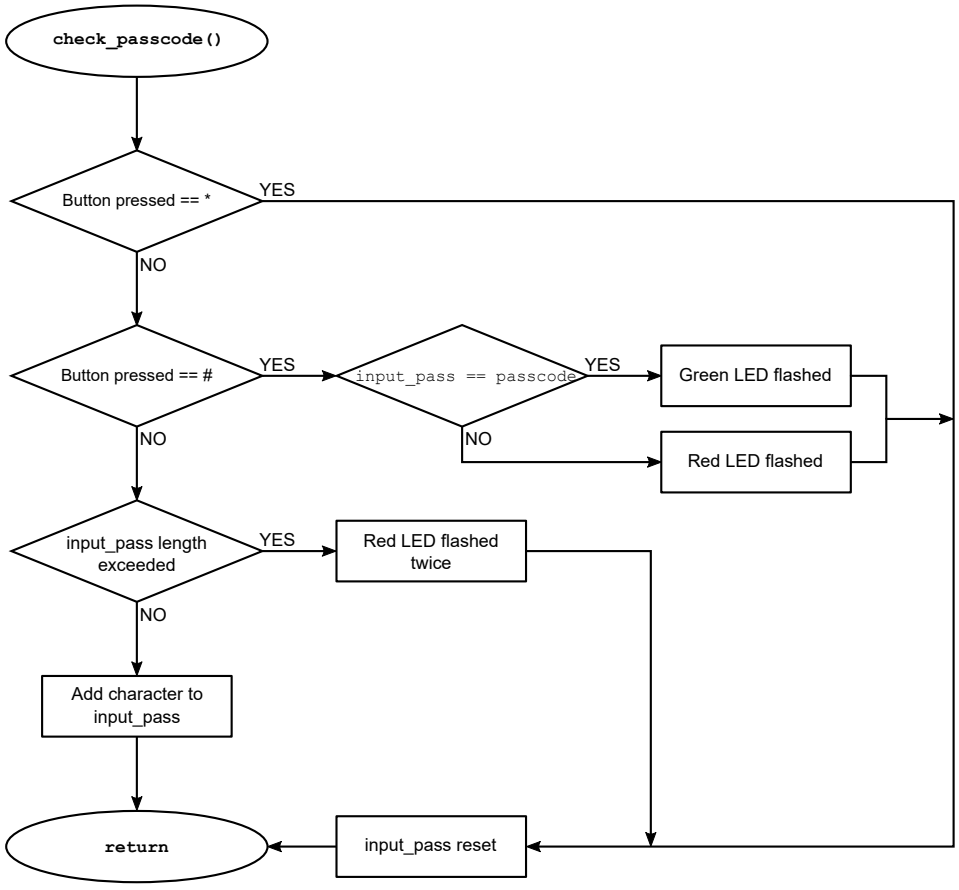
total of 7. Counting the buttons of the keypad left to right and top to bottom, button **B** (with a zero-indexed numbering scheme) is button number 7.

After recording the button press, the pins are reset to the initial state with columns as inputs and rows as outputs.

4.1.2 Passcode Check

In this application example, the star key (*) is used for resetting the input code, while the pound key (#) is used to check whether the input code matches the passcode. Therefore, this function begins by checking whether one of these special characters are pressed. If not, the pressed alphanumeric character is added to the pin code input string.

Figure 4-3. Passcode Processing Overview



If the pound key is pressed and the input code matches the passcode, the green LED is flashed before the input code is reset. If, on the other hand, the input code does not match, the red LED is flashed instead. If the star key is pressed, the passcode is simply reset.

The application implements a fixed length input code of 20 characters. If the length is exceeded, it flashes the red status LED twice and resets the input code.

4.2 Advanced Operation

The tinyAVR® and megaAVR® microcontroller families include advanced features that can make applications more efficient, both in terms of power consumption and CPU utilization. The following sections explain each feature used in the advanced application example.

4.2.1 Sleep

The most important low-power feature that is implemented in the advanced keypad application is putting the MCU to sleep. tinyAVR and megaAVR devices have three available sleep modes. The sleep modes each turn off the clock for different parts of the microcontroller to save power.

When designing an application that uses a sleep mode, one must consider which peripherals and clocks that need to be awake while the CPU sleeps. If, for example, the Real-Time Counter (RTC) should keep track of time while the CPU sleeps, one must select a sleep mode accordingly. Also, the sleep mode must be selected based on how the CPU should wake from sleep. Some interrupt types are unable to wake the device in certain sleep modes. Information regarding peripheral clocks and interrupts in different sleep modes can be found in the *Sleep Controller* section in the device data sheet.

In the advanced keypad application, no peripheral is needed to be kept running between button presses. Therefore, Power-Down Sleep mode is suitable. Looking at wake-up sources for this sleep mode, one can see that a pin change interrupt will be able to wake the CPU. This is also suitable for the application since a pin change (that is, a button press), is what the application is waiting for. It is important to note that some pin change interrupt types rely on a clock signal to trigger. Since the relevant clock is turned off in Power-Down Sleep mode, an asynchronous (independent of a clock signal) trigger must be chosen for the pin change interrupt. In the application, the BOTHEDGES trigger (as described in Section 4.2.2 Interrupt Operation) is chosen.

Power-Down Sleep mode initialization is shown in the following code snippet:

```
/* Set Power Down Sleep Mode and enable sleep */
SLPCTRL.CTRLA = SLPCTRL_SMODE_PDOWN_gc | SLPCTRL_SEN_bm;
```

4.2.2 Interrupt Operation

The basic keypad application makes the CPU continuously poll the column pins to see if a button has been pressed. Polling buttons with the CPU is inefficient use of both power and CPU resources.

To solve this problem, interrupts can be used. The tinyAVR® and megaAVR® devices feature pin sense interrupts for all of their GPIO pins. With pin sense interrupts enabled, the PORT peripheral will look over the pins and automatically notify the CPU if a change has been detected. This frees the CPU to perform different tasks while waiting for a key press, or to go to sleep to save power.

The pin sense interrupt features the following triggers:

- FALLING: Triggers when the logic input level of the pin has dropped from high to low
- RISING: Triggers when the logic input level of the pin has risen from low to high
- BOTHEDGES: Triggers both when the logic input level of the pin has dropped from high to low or risen from low to high
- LEVEL: Triggers whenever the pin is detected to be low

When combining the pin sense interrupts with a sleep mode, further considerations must be made. Not all pin sense interrupt triggers are compatible with all sleep modes. In the low-power keypad application, the lowest possible power consumption is desired and, therefore, a trigger that can wake the device from Power-Down Sleep mode must be used. The BOTHEDGES and the LEVEL triggers are able to wake the device in the deepest sleep mode. In the application, the BOTHEDGES trigger has been selected.

Configuration of the BOTHEDGES interrupt trigger for the columns of the keypad is shown in the following code snippet:

```
/* Enable BOTHEDGES interrupts for columns */
PORTC.PIN0CTRL = PORT_ISC_BOTHEDGES_gc;
PORTC.PIN1CTRL = PORT_ISC_BOTHEDGES_gc;
PORTC.PIN2CTRL = PORT_ISC_BOTHEDGES_gc;
PORTC.PIN3CTRL = PORT_ISC_BOTHEDGES_gc;
```

4.2.3 Timer Delays

Delaying the application flow by counting clock cycles with the CPU similar to the basic keypad example is inefficient. Instead of holding the CPU to something trivial as counting clock cycles, it can go to sleep to save power or do other useful work.

The ATtiny1627 microcontroller used in this application example features a Real-Time Counter (RTC) which can keep track of delays independent of the CPU. It can run on the internal 32.768 kHz RC oscillator, which means it will run in any sleep mode. The RTC features a Periodic Interrupt Timer (PIT) which can interrupt the CPU from any sleep mode after a set number of RTC clock cycles.

PIT initialization is shown in the code snippet below:

```
/* Set interrupt to fire every 32 RTC clock cycles and enable PIT */
RTC.PITCTRLA = RTC_PERIOD_CYC32_gc | RTC_PITEN_bm;
/* Enable PIT interrupts */
RTC.PITINTCTRL = RTC_PI_bm;
```

4.2.4 Pin Setup

When an application leaves some pins of the microcontroller unused, special considerations should be made to keep power consumption down. When the device is in a sleep mode, the power consumption from unused and incorrectly configured pins may be significant.

The most important aspect to consider is avoiding floating pins. Floating pins are pins that are not tied to a defined voltage level. They may behave sporadically based on outside and internal interference. This sporadic behavior may lead to the voltage threshold of transistors being crossed which leads to dynamic power consumption.

Floating pins are avoided by enabling internal pull ups for the pins. This pulls the voltage to a constant logic high level through an internal pull-up resistor. One may also disable the digital input buffer of the pin. This disconnects the digital input circuitry, further lowering the power consumption.

In all applications configured using Atmel START, unused pins are automatically initialized as inputs with the internal pull up enabled. In the advanced keypad application, the unused pins are also initialized in START to disable the digital input circuits.

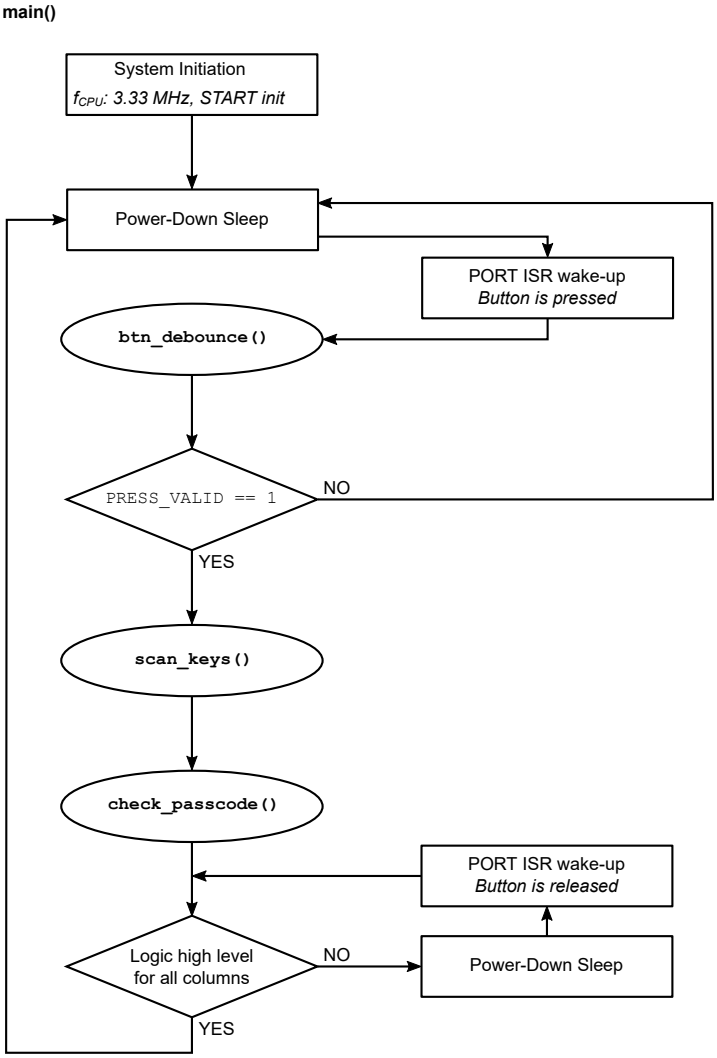
The following code snippet configures pin PA0 to an input with pull up enabled and the digital input circuit disabled:

```
/* Set the pin to input */
PORTA.DIRCLR = PIN0_bm;
/* Enable pullup and disable digital input buffer */
PORTA.PIN0CTRL = PORT_PULLUPEN_bm | PORT_ISC_INPUT_DISABLE_gc;
```

4.2.5 Advanced Keypad Implementation

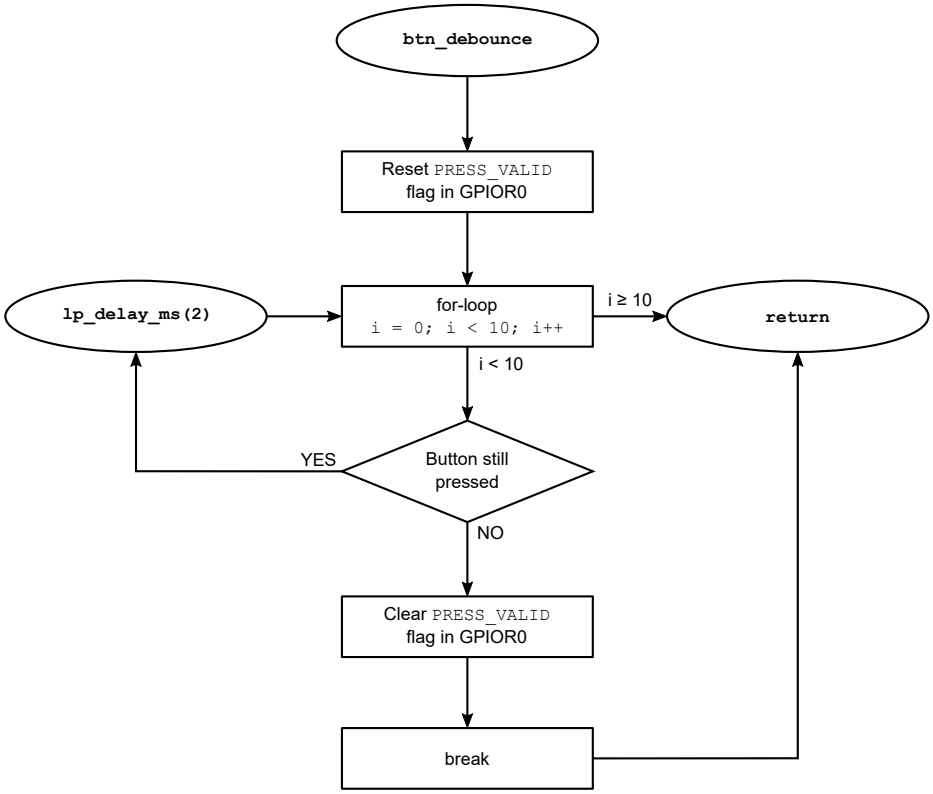
The main application flow of the advanced keypad application is shown in the figure below. It shows that the device is set to sleep when waiting for a keypress. A pin change interrupt wakes up the CPU and lets it scan the button that was pressed.

Figure 4-4. Main Application Flow



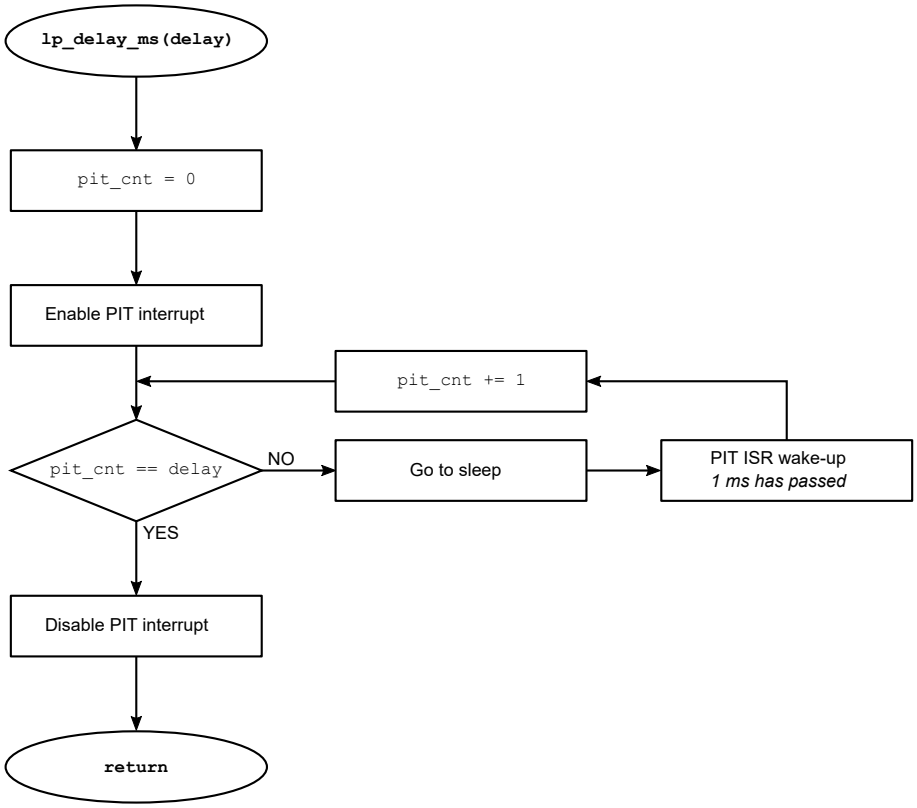
After wake up from sleep, the button press is debounced. This is performed by running a loop ten times, in which the button is checked to be pressed as shown in the chart below. If the button is found to be pressed all ten times, the button press is considered valid. If the button is not read as pressed while iterating through the loop, the validation flag is cleared and the CPU exits the loop.

Figure 4-5. btn_debounce()



The debounce function uses a 2-ms delay between the iterations of the loop. This delay is implemented using a timer delay as shown in the figure below. The Periodic Interrupt Timer (PIT) is programmed to trigger an interrupt once every millisecond. When the interrupt is triggered, a counter is incremented, and when this counter reaches the desired number of microseconds, the CPU returns from the function.

Figure 4-6. lp_delay_ms()



The `scan_keys()` and `check_passcode()` functions are identical to those of the basic keypad example, except that they use the low-power timer delay function when timing the LED indicators.

5. Power Consumption

5.1 Basic Operation

When scanning the keypad in the basic application example, the CPU continuously polls the I/O lines and busy-waits for delays. This is reflected in the relatively high-power consumption as can be seen in the figures below.

Figure 5-1. Power Analysis, Basic Operation Without Key Press

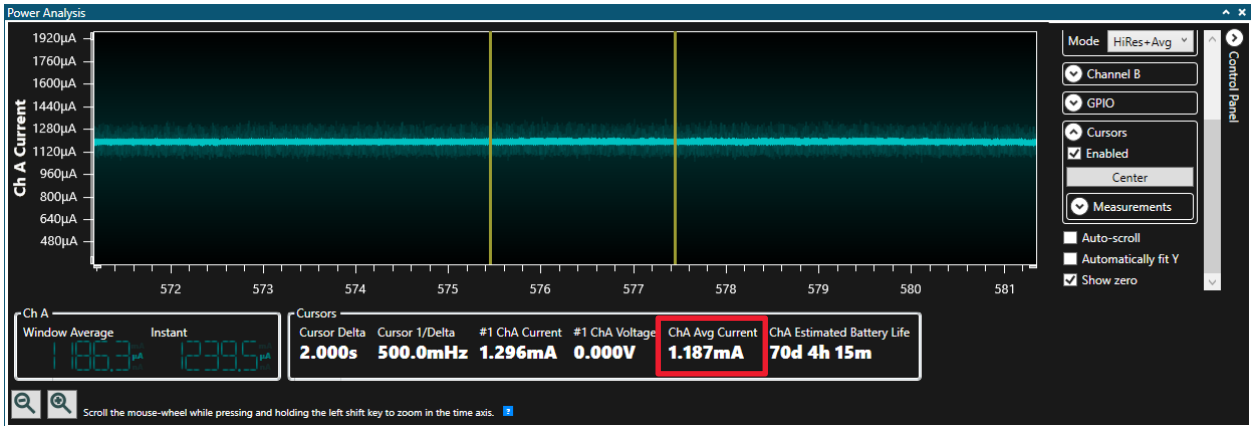
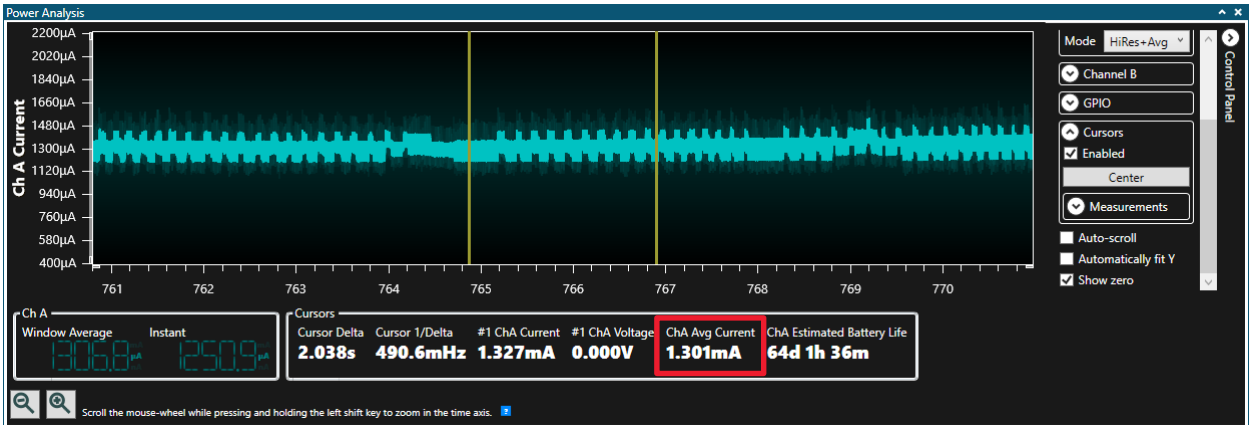


Figure 5-2. Power Analysis, Basic Operation With Key Press

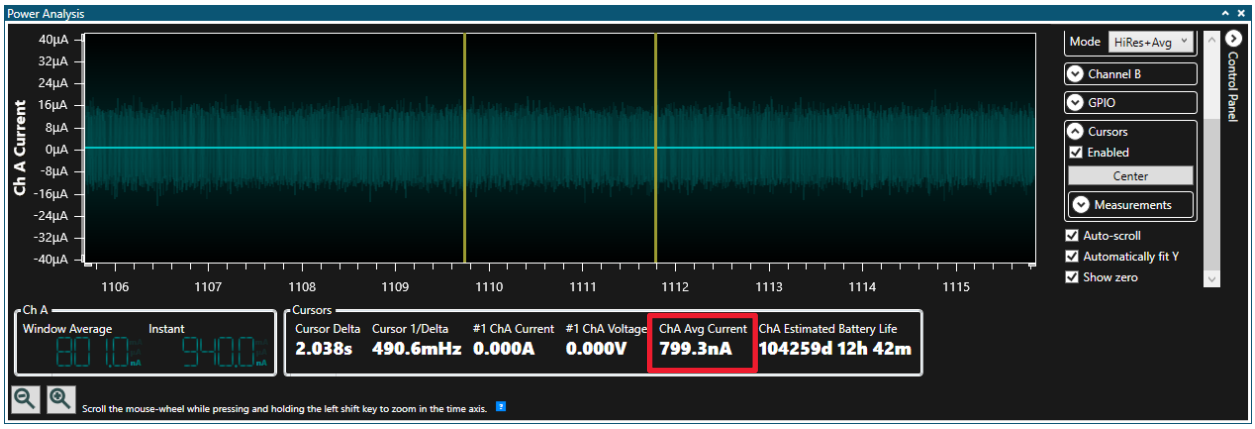


In addition to using much power, unused pins are not handled correctly by enabling the internal pull up, which may be seen in the instability of the power consumption. When touching the other pins of the device while measuring current, one can clearly see that the results are affected.

5.2 Advanced Operation

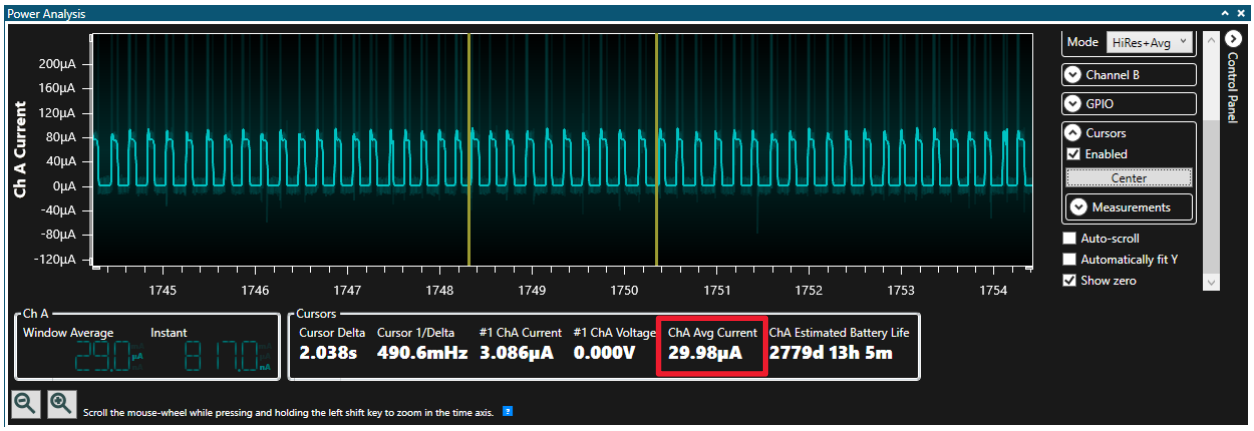
Using the features described in Section 4.2 Advanced Operation, the power consumption is very low as shown in the figures below. When the device is sleeping and no buttons are pressed, it uses ~0.8 µA.

Figure 5-3. Power Analysis, Advanced Operation Without Key Press



When buttons are repeatedly pressed and scanned by the device as seen in the second figure, power consumption is significantly higher than when the CPU is sleeping, but is still very low at ~30 µA on average and ~80 µA when holding down a button.

Figure 5-4. Power Analysis, Advanced Operation With Key Press



5.3 Plotting Current Data

The following instructions show how to analyze power consumption using the Power Debugger and Data Visualizer.

Notes:

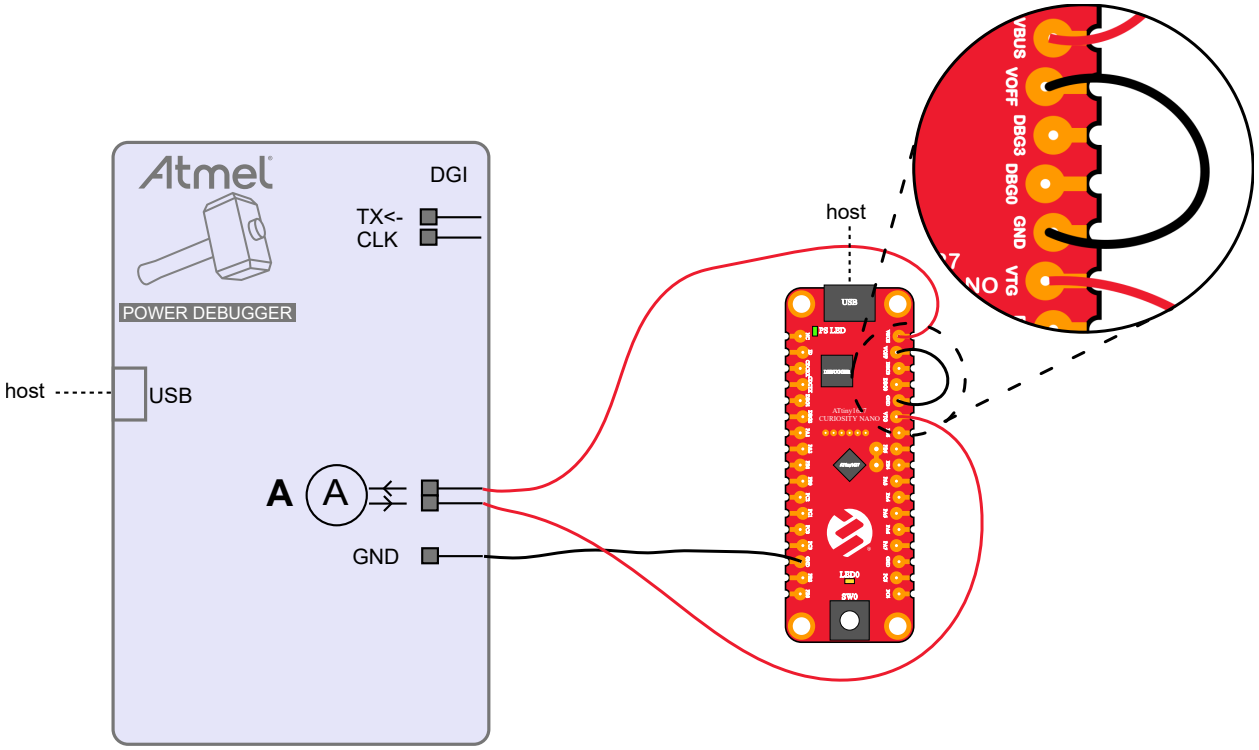
- For detailed information on the Power Debugger, refer to the Power Debugger User's Guide.
- For detailed information on the Data Visualizer, refer to the Data Visualizer User's Guide.

Power Debugger connection:

1. Connect one of the ground reference pins of the Power Debugger to a ground pin on the Curiosity Nano board.
2. Connect the VOFF pin on the Curiosity Nano board to a ground pin to turn off the connection between the USB power supply and the tinyAVR device.
3. Connect the VBUS pin on the Curiosity Nano board to the the input current pin of measurement circuit A on the Power Debugger.
4. Connect the output current pin of measurement circuit A on the Power Debugger to the VTG pin on the Curiosity Nano board.

Measurement circuit A of the Power Debugger is now connected in series between the USB voltage source and the tinyAVR microcontroller, and can measure the total current used.

Figure 5-5. Power Debugger Connection



Data Visualizer setup:

1. Open **Data Visualizer**.
2. Open *Configuration > Visualization > Power Debugging* in Data Visualizer.
3. On the **DGI Control Panel** pane, select *Power Debugger Data Gateway > Connect*.
4. Check the **Power** check box.
5. Expand the **Channel A** dropdown in the **Control Panel** in the **Power Analysis** pane.
6. Drag the plug from **A Current** in the **DGI Control Panel** pane to the **Current** socket in the **Power Analysis** pane.
7. Press **Start** in the **DGI Control Panel** pane.
8. Optional: Enable cursors by checking the **Enabled** check box in the **Cursors** dropdown.

Figure 5-6. Power Debugging Graph in Data Visualizer

6. Get Code Examples from Atmel START

The code examples are available through Atmel START, which is a web-based tool that enables the configuration of the application code through a Graphical User Interface (GUI). The code can be downloaded for Atmel Studio MPLAB X and IAR Embedded Workbench® via the direct example code link below or the **Browse Examples** button on the Atmel START front page.

The Atmel START webpage: start.atmel.com/.

Code Examples

- Using Matrix Keypad with AVR Devices - Basic
  - https://start.atmel.com/#example/Atmel%3AApplication_AVR_Examples%3A1.0.0%3A%3AApplication%3AUsing_Matrix_Keypad_with_AVR_Devices_-_Basic%3A
- Using Matrix Keypad with AVR Devices - Advanced
  - https://start.atmel.com/#example/Atmel%3AApplication_AVR_Examples%3A1.0.0%3A%3AApplication%3AUsing_Matrix_Keypad_with_AVR_Devices_-_Advanced%3A

Click **User Guide** in Atmel START for details and information about example projects. The **User Guide** button can be found in the example browser, and by clicking the project name in the dashboard view within the Atmel START project configurator.

Atmel Studio

Download the code as an `.atzip` file for Atmel Studio from the example browser in Atmel START by clicking **Download Selected example**. To download the file from within Atmel START, click **Export project** followed by **Download pack**.

Double click the downloaded `.atzip` file, and the project will be imported to Atmel Studio 7.0.

MPLAB X

Download the code as an `.atzip` file for MPLAB X IDE from within Atmel START by clicking **Export project** followed by **Download pack**.

To open the Atmel START example in MPLAB X, select from the menu in MPLAB X, *File > Import > START MPLAB Project* and navigate to the `.atzip` file.

IAR Embedded Workbench

For information on how to import the project in IAR Embedded Workbench, open the Atmel START User Guide, select **Using Atmel Start Output in External Tools**, and **IAR Embedded Workbench**. A link to the Atmel START User Guide can be found by clicking *Help* from the Atmel START front page or **Help And Support** within the project configurator, both located in the upper right corner of the page.

7. Get Code Examples from GitHub

The code examples are available through GitHub, which is a web-based server that provides the application codes through a Graphical User Interface (GUI). The code examples can be opened in both Atmel Studio and MPLAB X. To open the Atmel Studio project in MPLAB X, select from the menu in MPLAB X, *File > Import > Atmel Studio Project* and navigate to `.cproj` file.

The GitHub webpage: GitHub.

Code Examples

Download the code as a `.zip` file from the example page on GitHub by clicking the **Clone** or **download** button.

8. Revision History

Revision	Date	Description
A	06/2020	Initial document release

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with

your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

ISBN: 978-1-5224-6262-0

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820