사물인터넷융복합논문지 Vol. 12, No. 1, pp. 57-65, 2026 https://doi.org/10.20465/KIOTS.2026.12.1.057

Implementation of an Analog Hall-Effect USB Keyboard With an 8-bit AVR Microcontroller

Mingi Hwang[1], Yunjung Lee[2*]
[1]Student, Division of Electrical and Computer Engineering, Stony Brook University
[2]Professor, Dept. of Data Science, Jeju National University

8비트 AVR 마이크로컨트롤러를 사용한 아날로그 홀 효과 USB 키보드 구현

황민기[1], 이윤정[2*]
[1]스토니브룩 대학교 전기컴퓨터공학부 학생, [2]제주대학교 데이터사이언스학과 교수

Abstract Hall-effect keyboards have become a popular choice in premium PC gaming peripherals due to their ability to provide customizable and precise key actuation. This research investigates the feasibility of implementing such an analog hall-effect USB keyboard using a lower-cost 8-bit AVR ATmega32U4, alongside a WebHID-based graphical user interface. Results demonstrate that while the ATmega32U4 can successfully support hall-effect key sensing at an acceptable polling rate, limitations in ADC throughput and computational resources introduce drawbacks compared to more advanced microcontrollers. The source code for this work is available at https://github.com/goonmandu/GoonBoard-HE.

Key Words : Hall effect, ADC, AVR, LUFA, Keyboard, USB, WebHID

요 약 홀 효과를 사용한 키보드는 개인화가 가능하고 정밀한 키 입력 덕분에 고급 PC 게임용 주변기기 중 인기 있는 선택지가 되었다. 이러한 키보드는 많은 양의 아날로그 정보를 신속히 처리할 CPU와 신호 변환 속도가 빠른 ADC가 필요하다. 본 연구는 가격이 상대적으로 저렴한 8비트 AVR 마이크로컨트롤러인 ATmega32U4로도 상용 홀 효과 키보드 수준의 성능과 WebHID 기반 키보드 개인화 GUI의 구현 실용성을 분석하였다. 구현 결과, ATmega32U4로도 게임 용도로 적합한 폴링률로 홀 효과 키를 감지할 수 있었다. 이 보고서와 관련된 모든 소스 코드는 https://github.com/goonmandu/GoonBoard-HE에서 확인할 수 있다.

주제어 : 홀 효과, ADC, AVR, LUFA, 키보드, USB, WebHID

1. Foreword

1.1. Background

The evolution of PC gaming has consistently driven advancements in input technology. Among these innovations, keyboards stand out as one of the most critical peripherals, shaping how players interact with their digital environments. As the demand for precision and customization grows, hall-effect keyboards have emerged as the de facto standard for premier performance, enabling nuanced actuation through magnetic

*교신저자 : 이윤정(rheeyj@jejunu.ac.kr)
접수일 2026년 01월 02일 수정일 2026년 01월 28일 심사완료일 2026년 02월 10일
www.kci.go.kr

field sensing.

In traditional mechanical keyboards, each key switch serves as a convenient method to close an electrical path and allow the microcontroller (MCU) to detect a small current through one of its pins and register a keypress. This method of sensing is trivial to implement, cheap on processing and “good enough” for the vast majority of people: it only has one actuation point (being the point at which the circuit is closed). However, the simple binary switch is not ideal in fast-paced virtual environments where the most subtle inputs are critical to gaining an edge against competitors and/or the environment [1, 2].

1.2 Motivation

Almost all hall-effect keyboards are enabled by high-performance MCUs capable of rapidly sampling and processing analog signals. However, these devices often come at a cost and development overhead that places them outside the reach of some developers, hobbyists, and cost-sensitive designs. The idea behind this work is to question whether the same principles can be achieved with a simpler, more accessible platform: the 8-bit AVR MCU.

By designing and evaluating an analog hall-effect USB keyboard built upon the ATmega32U4, this paper demonstrates both the opportunities and challenges of implementing advanced input mechanisms on modest hardware. It is intended not only as a technical study but also as a contribution to the broader conversation on how resource constraints can drive innovative solutions in cost-effective embedded systems design.

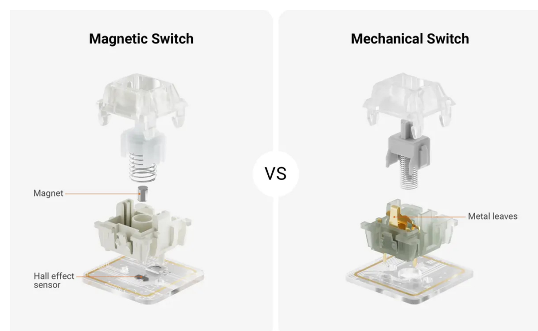
2. Overview

2.1. Existing Implementations

A popular free and open-source implementation on the STM32F4 series of MCUs is libhmk [3]. It incorporates many features traditionally found in commercial hall-effect firmware, including NKRO (N-key rollover; a feature that allows an arbitrary number of keys to be simultaneously pressed), user-defined key mappings, rapid trigger (a feature that decouples actuation and reset points to enable more responsive repeated presses) [1, 2, 4], customizable actuation points, SOCD (simultaneous opposing cardinal directions; a feature that suppresses an active input in software when the opposing direction is newly pressed), automatic analog sensor calibration, gamepad emulation (reporting as a game controller instead of a keyboard), and a web configurator. The work presented in this paper focuses on implementing a core feature set required to achieve a functionally complete hall-effect keyboard: user key mappings and actuation points, rapid trigger, SOCD, automatic sensor calibration, and a web configurator.

2.2 Hall-Effect Sensing

At the core of any hall-effect keyboard are four components: the hall-effect switch, the hall-effect sensor, the ADC (analog-to-digital converter), and the MCU. Instead of two metal plates found in mechanical switches, hall-effect switches embed a small permanent magnet inside of its stem that moves downward as the user presses the switch (Fig. 1). As the magnet moves



[Fig. 1] Exploded view comparisons of the two switch types. [4]

www.kci.go.kr

perpendicular to the sensor placed directly under the switch, it senses a change of magnetic field which affects its output voltage in proportion to its strength [5]. This analog output voltage is converted into the digital domain via the ADC, which the MCU uses to determine the current state of each key [1, 2].

2.3 The ATmega32U4

The MCU used in this design is the ATmega32U4. It incorporates an 8-bit modified Harvard architecture AVR CPU at 16 MHz on a 5 V supply voltage. It has 2 KB of SRAM, 32 KB of flash, and 1 KB of EEPROM for persistent configurations. It also supports serial communication protocols such as UART, I^2C, SPI, and most importantly, USB (up to 2.0) through a built-in dedicated USB controller [6]. This allows the MCU to function as a USB keyboard.

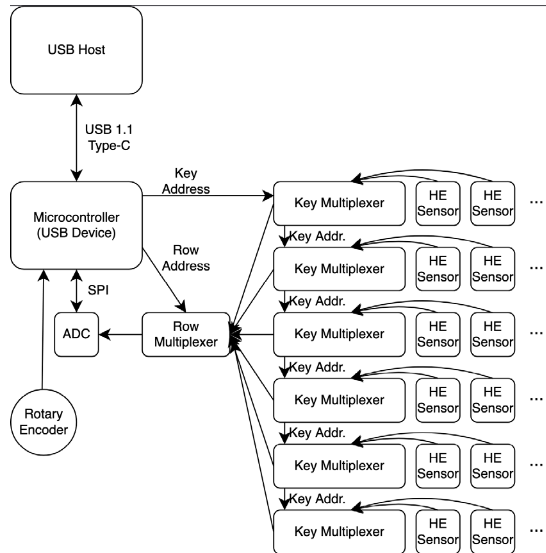
3. System Design

3.1 Architectural Overview

The MCU commands an ADC and two sets of analog multiplexers: the row multiplexer to select the row, and the key multiplexers (one per row) to select the individual sensor in that row. The output from the key multiplexers is fed as inputs to row multiplexer, at which one is selected based on the row address. Once the analog sensor output voltage has propagated up the multiplexers, it is sensed by the ADC where the MCU initiates a conversion. As a successive approximation ADC is used, the signal is acquired then sequentially converted as the MCU clocks the ADC result bits out via SPI. When all hall-effect sensor values have been acquired and processed into key inputs, the HID report is created and transmitted to the USB host via the built-in USB controller.

〈Table 1〉 Parts used in the breadboard implementation of the keyboard.

Type	Part Number
Microcontroller	ATmega32U4 (“Adafruit Pro Micro”)
ADC	ADS7885
Analog Multiplexer	CD74HC4067M96
Hall-Effect Sensor	OH49E
Hall-Effect Switch	KS-20T
Rotary Encoder	EC11



[Fig. 2] Hardware layout represented as a block diagram.

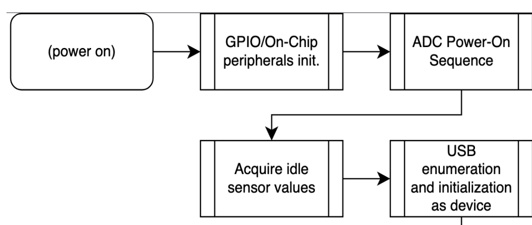
3.2 Libraries

The LUFA library (“Lightweight USB Framework for AVRs”) was used as the hardware abstraction layer for USB communication. In addition, the keyboard demo found in the GitHub repository was used to verify core USB functionality before modifying and extending the demo to include hall-effect-specific and user configurability features [7]. As such, most of the LUFA USB housekeeping routines are unchanged or have merely been extended to allow for additional features. Without this, too much time would need to be spent familiarizing oneself with the details of the USB protocol and the USB module registers, making the implementation more prone to errors.

www.kci.go.kr

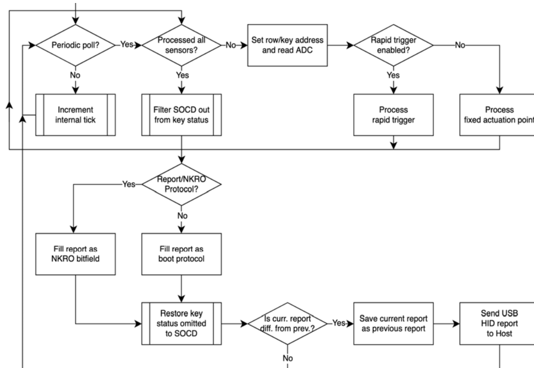
3.3 Firmware Architecture and Flow

First, when the MCU is powered on, it initializes the required GPIO port pins and on-chip peripherals, most notably SPI to interface with the ADC and the USB module for the MCU to serve as a USB device. Then, it outputs the power-up sequence to the ADC. After pausing to stabilize the ADC, it acquires baseline idle sensor values across the matrix to use as reference in the fixed-actuation mode. Finally, the MCU advertises itself as a USB HID keyboard device to the host (Fig. 3).



[Fig. 3] Firmware boot-up sequence.

Each polling cycle starts by examining the internal tick counter to see if a new report should be generated. If not yet, it increments its counter and checks again, processing buffered asynchronous USB requests from the host. When it needs to generate the next report (every 1 ms), it reads each sensor's ADC value, stores it in a global matrix, and uses either the rapid trigger algorithm or a simple threshold check to determine the state of the key. This runs until all sensors have been read from and moves on to suppressing SOCD results from the key states array, if enabled. Next, if in "report protocol," it constructs the HID report as an NKRO bit field, each bit representing one key code. Otherwise ("boot protocol"), it fills in the first six pressed keys. Finally, it compares the newly filled report byte-stream against the previous buffer, updating the previous buffer and sending the HID report only if the current report is different from the one previous. At the end of this cycle, the firmware returns to checking the internal tick counter and waits for the next scan (Fig. 4).



[Fig. 4] Sensor matrix scanning sequence.

3.4 Generic HID and Quality-of-Life Features

While the prototype would have been functionally complete with just a Boot Protocol HID report, extra interfaces and report descriptors are used to communicate with the host for various other purposes. First, a "Consumer" report is sent after the keyboard report to implement host system-agnostic reporting of multimedia keys (e.g. Play/Pause, volume controls, Next/Previous track, etc.) [8]. This is necessary as the key codes recognized by each operating system are different: for example, Windows 10 does not recognize the key codes in the range [0xE8, 0xFF] while Ubuntu 24.04 does [8].

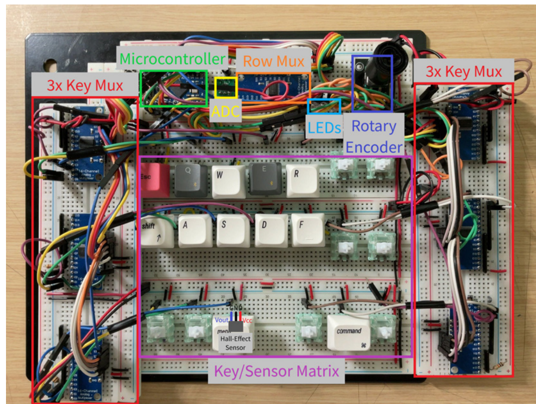
Second, a vendor-defined interface is advertised concurrent to the main keyboard interface to facilitate browser-based WebHID configuration of the keyboard [8, 9]. Without this, such a feature would not have been easily done as some operating system kernels (such as macOS 26 and Ubuntu 24.04) reserve the main keyboard interface, requiring root privileges. In addition to these benefits, this interface lets the keyboard appear to the host as another device so that the keystrokes and configuration commands are logically decoupled.

www.kci.go.kr

4. Implementation

4.1 Breadboard Assembly

The test design was built on breadboards, with one large breadboard for the core components and two smaller breadboards for the key multiplexers (Fig. 5).



[Fig. 5] Labeled photo of the breadboard assembly.

4.2 Firmware Development

4.2.1 Basic Keyboard Functions

Initial development started with simple pushbutton switches to verify the functionality of the keyboard demo in the LUFA library. Then, the function responsible for generating USB HID reports was modified to scan logic levels for each pin that had pushbuttons connected.

4.2.2 Hall-Effect Sensors

The next stage of development began with estimating the number of sensors (and therefore keys) the MCU would be able to support. Initial calculations had the absolute maximum at about 120 keys with 1000 Hz polling rate while leaving some headroom for the firmware to service host requests. This meant that targeting the "75% layout" with 82 keys was the most practical. As the packages were being shipped, a breadboard layout for the sensor matrix was devised to accommodate the additional six key multiplexers.

A test layout was soon constructed to verify the functionality of the ADC, the analog multiplexers, the sensors, and the switches. It consisted of one multiplexer stage (as opposed to the two stages in the final design), four of whose inputs were connected to the hall-effect sensors. Next, the routine to scan the per-key ADC values was written and the fixed actuation detection was tested. With the basics verified working, a rudimentary rapid trigger algorithm was devised by simulating various types of keystrokes. After identifying the errors and weaknesses of the initial rapid trigger implementation, additional state variables were added to improve the algorithm.

4.2.3 Expanding the Sensor Matrix

Next, the sensor array was expanded into a matrix as described in Section 3.1. Initially, the state variables and overall algorithm were reused from the simple one-stage multiplexer design. However, upon implementing the SOCD feature, it became apparent that the time between the row and key addresses being sent out and starting the ADC conversion was shorter than the multiplexers' propagation delays. This forced a rewrite of the sensor scan loop in a way that one iteration effectively reads the ADC, sends the addresses of the key to be scanned next, then processes the sensor values to be converted into key states. Compared to inserting arbitrary delays in between sending the addresses and reading the ADC, this approach saves approximately 72 μs per USB frame by repurposing the algorithm processing time as waiting for propagation delays, a significant amount of time in the 1000 μs budget.

4.2.4 Micro-Optimizations

Moreover, inline assembly was used to further optimize software alignment of the bytes received from the ADC which sent the 8 bits in

www.kci.go.kr

two byte packets; 7 MSbs then 1 LSb [10]. A simple inline assembly section was used to shift/rotate two general-purpose registers to efficiently store them in one 8-bit register.

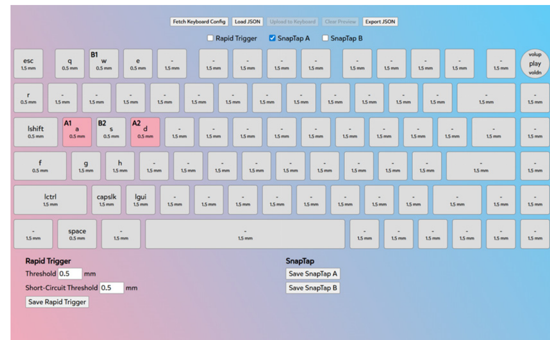
4.2.5 NKRO and Media Keys

The final process of the firmware core was to implement NKRO using a key code bit field instead of appending individual key codes to the report array. Prior to this modification, the keyboard operated on an "extended boot protocol" report, where it would structure the keyboard HID report like a standard boot protocol report, but with 32 key codes instead of the fixed 6. Transitioning away from this "32KRO" to true NKRO also cut down the report generation time by about 150 μs as up to 8 key codes could be processed cheaply with a simple bit mask ORI instruction on the relevant bit field cached in a CPU register.

After the NKRO feature was verified, the last step was to separate out the media keys section in the keyboard report bit field out into the Consumer Controls usage page for host system-agnostic recognition. This involved structuring the Consumer Controls report as a bit field in the same order as the key codes' 8-bit value mappings and setting its report data segment as the last three bytes of the key codes bit field.

4.2.6 Add-Ons and the Web Interface

At the end of the development cycle, support for a rotary encoder was added alongside a WebHID-based GUI and the necessary USB interfaces, endpoints, and report descriptors (Fig. 6) [8. 9]. While all WebHID APIs for communication were written by the author, generative AI (ChatGPT; models "o3-mini" and "5 Thinking") was used to write the HTML/CSS markup and the JavaScript UI logic. Finally, the web configurator was deployed to GitHub Pages as a static web app.



[Fig. 6] The web configurator interface.

5. Evaluation and Analysis

All evaluation and metrics were done with firmware version V3.250826.0, released 26 August 2025.

5.1 Performance Metrics

Table 2 summarizes the performance and timings with 20 sensors installed. "Poll Time" is defined as the time from the start of one USB polling frame until the HID report is sent out of the endpoint. A logic analyzer probing a debug pin was used as the stopwatch.

〈Table 2〉 Performance metrics.

Rapid Trigger	Poll Time	1000 Hz?
Enabled	760 μs	Yes
Disabled	730 μs	Yes

Even though the breadboard only had 20 sensors, performance for the full 82-key configuration can be extrapolated as the firmware still scans the complete number of keys. Because simulated signals took 2 μs less to process versus those coming from real sensors due to pre-filtering, the following (Table 3) can be inferred by simply adding the total difference of 2 μs · 62 = 124 μs per frame:

www.kci.go.kr

〈Table 3〉 Extrapolated timings for 82 keys

Rapid Trigger	20-key	82-key	1000 Hz?
Enabled	760 μs	884 μs	Yes
Disabled	730 μs	854 μs	Yes

While the worst-case metric of 884 μs only leaves ~110 μs for background tasks, it is acceptable for the purposes of this firmware as most of these are configuration commands where the user will likely not be in a time-critical situation while it dips into 500 Hz.

5.2 Resource Utilization

With almost half of flash and SRAM still available, the firmware allows for future improvements without having to worry about optimizing for either code size or compact RAM use.

〈Table 4〉 Resource utilization for the firmware.

Section	Bytes Used	% Used
Flash	16636	58.0% of 28672
SRAM	1455	56.8% of 2560
EEPROM	204	20.4% of 1024

5.3 Comparison versus the STM32F4

While the *functionality* of each module implemented in the firmware is virtually identical to that of libhmk (discussed in Section 2.1), the *number of features* trails behind significantly as there simply is not enough time in between USB frames to fit the necessary background tasks to support those features. For example, analog joystick emulation would be nearly impossible without dropping USB polling rate to 500 Hz as it would need to do additional processing (e.g. dead zone filtering) to construct and send a third HID report alongside the two existing key code reports [11].

6. Discussion

6.1 AVR Benefits and Trade-Offs

The main benefit of using the AVR MCU was the simplicity in its programming and the author's existing familiarity with the AVR core and toolchain. Due to this, development was overall smooth with no MCU learning curve, barring some USB quirks and limitations that were diagnosed and worked around with Linux terminal commands, WireShark, and a logic analyzer.

There were numerous trade-offs; first, unusual optimizations had to be made such as rearranging the address output and ADC read sequence to fulfill propagation delay requirements. Moreover, the ATmega32U4's built-in ADC was too slow to support 1000 Hz polling rate (13 kS/s · 1 frame / 82 samples = 158.5 frames/s); were it fast enough, the external ADC and the row multiplexer could have been avoided entirely and the outputs from the key multiplexers directly routed to the GPIO pins for the different channels for the internal ADC. This hypothetical would also eliminate the need to waste time waiting for the SPI transfers to finish, reducing the time per matrix scan by roughly 200 μs.

Second, other quality-of-life features had to be omitted to satisfy the 1000 Hz polling rate and the program fitting in the limited on-chip memory. On more complex MCUs with a CPU frequency and SRAM/flash sizes close to two orders of magnitude better than the ATmega32U4, this is not an issue. However, as the AVR only gets about 100 μs per frame to service non-keyboard requests, fitting, let's say, a small 128-by-64 OLED display becomes much trickier both timing- and code size-wise as the driver code, display framebuffer, and bitmaps for icons and fonts would need to be stored in the MCU's memory [12].

6.2 Practical Challenges

Due to the limited time available to complete

www.kci.go.kr

the project necessitating fast development cycles and prototyping, plug-and-play breadboards and breakout boards had to be used instead of surface-mount devices and PCBs. While this was initially beneficial, the later stages of firmware development (when additional hardware features were added) were slowed by the space inefficiency of jumper wires rendering free space on the breadboard unusable.

Also, the Pro Micro development board not offering all of the GPIO pins on the ATmega32U4 could become a point of failure in later production stages as the pins that support pin change interrupts (PCINTx pins) had to serve other functions instead of monitoring the rotary encoder for state changes [6]. This meant that encoder states had to be polled like every other key code source on the keyboard. This could fail if the user turns the dial quick enough (faster than 6 deg/μs) and make the finite state machine miss a state transition. Interrupts would fundamentally solve this problem as they are asynchronous to all USB routines [6, 13].

6.3 Potential Improvements

There are numerous potential improvements that could be made to both the firmware and the hardware layout of the keyboard. First, instead of relying on the singular CONTROL endpoint, OUT endpoints for each USB interface could explicitly be configured to further minimize the chance of the firmware mishandling USB packets [12]. Second, the routine responsible for sending which reports get sent could be cleaned up. Currently, the firmware sends both the key code report and the Consumer Control report if either bit field has changed from the previous buffer. A method would be to have separate buffers for the basic key codes and the media keys, check which buffers changed, and only send those that did.

Hardware layout-wise, the GPIO pin usages and mappings could be redone from the ground-up to put the rotary encoder on the PCINTx pins and allow the firmware to support I^2C devices such as OLED displays as the I^2C SCL/SDA pins are occupied by bits 2 and 3 of the row multiplexer address select line.

7. Conclusion

7.1 Summary of Findings

This work demonstrated that advanced hall-effect keyboard functionality, typically reserved for higher-end MCUs, could be implemented on a modest 8-bit AVR platform. Through careful architectural decisions, optimization of firmware routines, and deliberate trade-offs, the ATmega32U4 was shown to sustain a full 82-key configuration at 1000 Hz polling rate with rapid trigger, SOCD, NKRO, and user configurability. Although not as feature-complete as STM32F4-based designs, the design achieved the main goal of validating hall-effect sensing on accessible, cost-effective hardware. The evaluation confirmed that timing budgets and memory constraints could be managed with practical engineering trade-offs [14].

7.2 Low-Cost Design Implications

The results highlight the viability of using low-cost MCUs for applications previously assumed to require more powerful devices. By leveraging external components such as faster ADCs and multiplexers, it is possible to extend the relevance of older platforms without abandoning performance-critical requirements like high polling rates [15]. For developers and hobbyists, this approach lowers the barrier to entry for building hall-effect keyboards, promoting experimentation and custom solutions. More broadly, the design illustrates how resource constraints can drive innovation, forcing unorthodox optimization strategies that may not otherwise be considered in resource-rich environments [14].

www.kci.go.kr

7.3 Suggestions for Future Work

Future efforts could focus on expanding the firmware feature set while maintaining performance targets. These include implementing joystick emulation, support for simple displays, or richer configurability options through more robust USB endpoints. Hardware refinements, such as moving away from breadboard prototypes toward custom PCBs, would address reliability issues and open the design up to the public with a ready-to-manufacture gerber file. Further investigation into alternate scanning architectures, more efficient USB handling, or hybrid MCU-ADC configurations could also improve timing margins. Ultimately, porting the design ideas to a wider range of low-cost MCUs would help validate the generalizability of the approach and further bridge the gap between hardware accessibility and high-performance input devices [15].

REFERENCES

[1] Vodden, G. (2025). What Is A Hall Effect Keyboard?. https://www.rtings.com/keyboard/learn/what-is-a-hall-effect-keyboard.

[2] T., O., & L., J. (2024). Hall effect keyboards, a history | LTT Labs. Hall Effect Keyboards, A History. https://www.lttlabs.com/blog/2024/12/31/hall-effect-keyboards-a-history.

[3] Prasertsuk, P. ("peppapighs") (2025). libhmk. https://github.com/peppapighs/libhmk.

[4] Nuphy Studio (2024). What is Magnetic Switch?. https://nuphy.com/blogs/journal/what-is-magnetic-switch.

[5] Ripka, P. (2010). Magnetic sensors and magnetometers. Artech House.

[6] Atmel Corporation. (2016). ATmega16U4/ATmega32U4 | 8-bit MCU with 16/32K bytes of ISP Flash and USB Controller | DATASHEET. https://ww1.microchip.com/downloads/en/devicedoc/atmel-7766-8-bit-avr-atmega16u4-32u4_datasheet.pdf.

[7] Camera, D. ("abcminiuser") (2025). lufa. https://github.com/abcminiuser/lufa.

[8] Bates, B. M., et al. (2004). Universal Serial Bus - HID Usage Tables. https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf.

[9] World Wide Web Consortium. (2023). WebHID API specification. W3C. https://www.w3.org/TR/webhid.

[10] Microchip Technology Inc. (2021). AVR instruction set manual. Microchip Technology Inc. https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-InstructionSet-Manual-DS40002198.pdf.

[11] Texas Instruments. (2008). 10-/8-BIT, 3-MSPS, MICRO-POWER, MINIATURE SAR ANALOG-TO-DIGITAL CONVERTERS. https://www.ti.com/lit/ds/symlink/ads7885.pdf.

[12] Deiss, M. (2025). Master Your Game: How 5-Stop Hall-Effect Clutch Triggers with Hair-Trigger Mode Enhance Precision in Competitive Play. https://uk.turtlebeach.com/blog/5-stop-hall-effect-clutch-triggers-hair-trigger-mode.

[13] Axelson, J. (2015). USB complete: The developer's guide (5th ed.). Lakeview Research.

[14] Barr, M., & Massa, A. (2006). Programming embedded systems: With C and GNU development tools (2nd ed.). O'Reilly Media.

[15] Marwedel, P. (2011). Embedded System Design: Embedded systems foundations of cyber-physical systems (2nd ed.). Springer.

황 민 기(Mingi Hwang) [준회원]

- 2021년 8월 ~ 현재 : 스토니브룩 대학교 컴퓨터공학과 재학

〈관심분야〉
엣지컴퓨팅, 사물인터넷, 임베디드펌웨어, FPGA

이 윤 정(Yunjung Lee) [준회원]

- 2004년 9월 ~ 2022년 2월 : 제주대학교 전산통계학과 교수
- 2022년 3월 ~ 현재 : 제주대학교 데이터사이언스학과 교수

〈관심분야〉
사물인터넷 보안, 정보보안

www.kci.go.kr