Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Study of GPIO Expansion Through Multiplexers and Shift Registers

Chandramouli Haldar

chandramoulihaldar@gmail.com

NovaTech Innovative Solutions https://orcid.org/0009-0004-9759-194X

Systematic Review

Keywords: GPIO Expansion, 74HC4051, 74HC595, IoT, Embedded Systems, Arduino Uno, ESP32, Port Manipulation

Posted Date: September 8th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10940138/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

A Study of GPIO Expansion Through Multiplexers and Shift Registers

Chandramouli Haldar
Independent Researcher
NovaTech Innovative Solutions
Kolkata, India
0009-0004-9759-194X

Abstract

Limited GPIO pins are a common problem in microcontrollers used for IoT and embedded systems. As more sensors and devices are connected, the available GPIO pins can quickly become insufficient. The aim of this study is to show how simple and low-cost components can be used to increase the number of inputs and outputs available to a microcontroller, and to study the effect of this expansion on speed, power consumption, and scalability. In this work, a 74HC4051 analog multiplexer and 74HC595 shift registers are used with Arduino Uno and ESP32 boards. The 74HC4051 provides access to eight analog channels using four microcontroller pins, while cascaded 74HC595 devices provide multiple digital outputs using only three microcontroller pins. The experiments compare direct GPIO connections with the expanded configurations and examine GPIO usage, execution time, power consumption, and the effect of adding more shift registers. The results show that the 74HC4051 can reduce the number of pins required for eight analog inputs by 50%, while four 74HC595 devices can provide 32 digital outputs using only three microcontroller pins. Direct GPIO control also reduces execution time significantly compared with standard Arduino GPIO functions, with a maximum measured improvement of 58.2 times on the Arduino Uno. An eight-device 74HC595 setup was tested to control 64 digital outputs using the same three microcontroller pins. The results demonstrate that these simple hardware components can provide an effective way to expand microcontroller I/O while also showing the trade-offs in speed, power consumption, and sequential input reading. The study provides practical information for selecting GPIO expansion methods in IoT and embedded applications.

Keywords GPIO Expansion, 74HC4051, 74HC595, IoT, Embedded Systems, Arduino Uno, ESP32, Port Manipulation

1 Introduction

The rapid growth of Internet of Things (IoT) applications has increased the demand for embedded nodes capable of interfacing with tens to hundreds of sensors and actuators. However, the limited GPIO resources of low-cost microcontrollers can become a significant constraint as system complexity increases. The Arduino Uno, based on the ATmega328P, provides 14 digital I/O pins and 6 analog input channels, while the ESP32 family offers a larger nominal GPIO count, of which several interfaces may be unavailable or constrained by boot-strapping functions, ADC2–Wi-Fi interactions, and flash-bus requirements. Consequently, applications involving multi-channel sensing and actuation can rapidly exhaust the available GPIO resources; for example, a conventional 4×4 keypad requires eight GPIO lines, while a multi-channel relay interface may require substantially more.

Hardware-based GPIO expansion using analog multiplexers and serial-in/parallel-out shift registers provides a low-cost alternative to direct GPIO allocation. Although these techniques are well established individually, their combined impact on GPIO utilization, execution latency, power overhead, and scalability has received limited systematic evaluation within a common experimental framework. This study therefore investigates an integrated architecture based on the 74HC4051 analog multiplexer and 74HC595 shift register, evaluated across Arduino Uno and ESP32 platforms. The principal contributions of this work are as follows:

- A quantitative GPIO Efficiency Ratio (GER) is introduced to characterize the relationship between addressable I/O channels and the number of MCU GPIO pins required by the expansion architecture.
- A controlled cross-platform experimental evaluation is conducted on Arduino Uno and ESP32, covering GPIO utilization, execution latency, power consumption, and scalability across five experimental scenarios.
- The scalability of cascaded 74HC595 devices is experimentally investigated, demonstrating reliable operation up to eight devices, corresponding to 64 digital outputs, under the evaluated conditions.
- Reproducible firmware procedures are developed for multiplexer-based input acquisition, cascaded shift-register control, and direct port manipulation, enabling systematic comparison between high-level and low-level GPIO access.
- The resulting architecture is evaluated for resource-constrained IoT applications, including industrial automation, smart agriculture, medical sensing, robotics, home automation, and edge-AI systems.

2 Background and Related Work

Microcontrollers used in IoT and embedded systems often have a limited number of GPIO pins. When a system needs to connect several sensors, LEDs, displays, relays, or other devices, the available GPIO pins may not be sufficient. One simple way to solve this problem is to use additional hardware that allows several devices to share a smaller number of microcontroller pins.

Two commonly used components for this purpose are the 74HC4051 multiplexer and the 74HC595 shift register. The 74HC4051 is useful for increasing the number of input channels. It provides eight channels that can be selected using three control pins. The selected channel is connected to a common signal pin, which can then be connected to an analog input of the microcontroller. The device operates from 2–6 V and has a typical on-resistance of about 70 Ω [1]. Since the channels are selected one at a time, the 74HC4051 is suitable when the connected signals do not need to be read simultaneously.

The 74HC595 provides a simple way to increase the number of digital outputs. It receives data serially and provides eight parallel output pins. Only three microcontroller pins are normally required for data, clock, and latch signals. Several 74HC595 devices can also be connected together to provide more outputs while continuing to use the same three microcontroller pins [2]. This makes the device useful for controlling LEDs, displays, relays, and other digital output devices.

The 74HC4051 and 74HC595 solve different GPIO problems. The 74HC4051 is mainly useful when a system has many input signals, while the 74HC595 is mainly useful when a system needs many digital outputs. They can also be used together when a system requires both additional inputs and additional outputs.

Several previous studies have shown the importance of limited GPIO resources in IoT and embedded systems. Kumar et al. [3] developed a low-power Arduino–LoRa system for smart agriculture, where multiple sensors need to be connected to a small microcontroller platform. Rodrigues et al. [4] developed an ECG/PPG system using an Arduino Nano, where the available GPIO resources were also an important consideration. These studies show that GPIO limitations can affect the design of practical embedded systems. However, their main focus was on the target applications rather than on measuring the performance of GPIO expansion hardware.

Other devices can also be used for GPIO expansion. For example, the PCF8574 and MCP23017 provide additional digital I/O through serial communication interfaces [5, 6]. These devices are useful when more digital I/O is required, but the microcontroller must communicate with the expansion device through the corresponding serial bus. Therefore, the choice of GPIO expansion method depends on the number of required pins, update speed, and application requirements.

The way GPIO pins are controlled in software can also affect system performance. Arduino functions such as `digitalWrite()` are easy to use and make programs simpler, but they require additional software operations. The ATmega328P used in the Arduino Uno also allows GPIO pins to be controlled directly through hardware registers such as `PORTx`, `DDRx`, and `PINx` [7]. Direct register control can reduce software overhead and provide faster GPIO operation, although it is more dependent on the specific microcontroller.

Based on these observations, this study focuses on a simple experimental evaluation of GPIO expansion using the 74HC4051 and 74HC595. The experiments are performed using an Arduino Uno and an ESP32. The study compares the number of microcontroller GPIO pins required, execution time, power consumption, and the effect of increasing the number of cascaded shift registers. The results are intended to provide a clear understanding of the advantages and limitations of these two simple GPIO expansion methods for embedded and IoT applications.

3 Proposed Framework

The proposed framework combines a 74HC4051 analog multiplexer and cascaded 74HC595 shift registers to expand MCU input and output capacity while minimizing GPIO usage. The architecture separates analog acquisition, digital output expansion, firmware optimization, and communication, allowing the same hardware concept to be evaluated across Arduino Uno and ESP32 platforms.

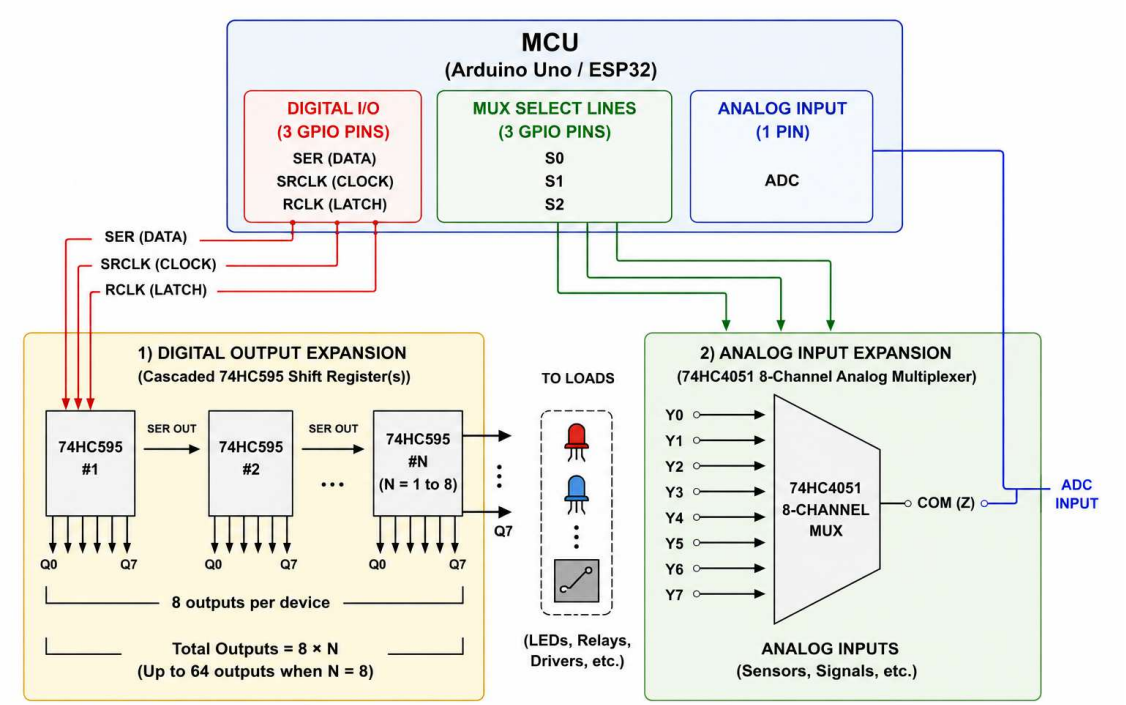


Figure 1: Proposed GPIO expansion architecture integrating the 74HC4051 multiplexer and 74HC595 shift register.

3.1 System Architecture

The framework consists of seven functional layers: power, sensor/actuator, shift-register, multiplexer, MCU, performance optimization, and communication. The power layer provides regulated 3.3-V/5-V supplies with local decoupling. The sensor/actuator layer interfaces with the expanded peripherals. The 74HC595 module provides digital-output expansion through the three-wire SER–$SRCLK$–$RCLK$ interface, while the 74HC4051 module provides eight-channel analog expansion using three address-selection lines and one ADC input. The MCU executes the control firmware, and the performance layer provides low-level GPIO optimization. The communication layer can interface the node with IoT services through MQTT, REST, LoRaWAN, or BLE.

For an n-bit address selection, the number of channels accessible through a multiplexer is

$$C_{\mathrm{MUX}} = 2^n, \tag{1}$$

where n is the number of selection lines. For the 74HC4051, $n = 3$, giving eight selectable channels.

For N cascaded 74HC595 devices, each providing eight parallel outputs, the total digital-output capacity is

$$C_{\mathrm{SR}} = 8N. \tag{2}$$

Since the cascaded devices share the same serial-data, clock, and latch lines, the MCU requires only three GPIO pins for the complete shift-register chain.

3.2 Mathematical Model

3.2.1 GPIO Efficiency

The GPIO Efficiency Ratio (GER) is defined as the number of addressable channels per MCU GPIO pin:

$$GER = \frac{C}{P_{\mathrm{MCU}}}, \tag{3}$$

where C is the number of addressable channels and P_{MCU} is the number of MCU GPIO pins used by the expansion interface.

For an n-select-line multiplexer, n selection pins and one signal pin are required; therefore,

$$GER_{\mathrm{MUX}} = \frac{2^n}{n+1}. \tag{4}$$

For the 74HC4051,

$$GER_{\mathrm{MUX}} = \frac{8}{4} = 2.00. \tag{5}$$

For N cascaded 74HC595 devices,

$$GER_{\mathrm{SR}} = \frac{8N}{3}. \tag{6}$$

Thus, four cascaded devices provide 32 outputs using three MCU GPIO pins, giving

$$GER_{\mathrm{SR}} = \frac{32}{3} = 10.67. \tag{7}$$

The corresponding GPIO saving relative to direct connection is calculated as

$$S_{\mathrm{GPIO}} = \left(1 - \frac{P_{\mathrm{exp}}}{P_{\mathrm{direct}}}\right) \times 100, \tag{8}$$

where P_{exp} and P_{direct} denote the GPIO requirements with and without expansion, respectively.

3.2.2 Timing Model

For N cascaded shift registers, $8N$ bits must be transferred during each complete output update. At a serial clock frequency f_{clk}, the ideal transfer time is

$$T_{\mathrm{shift}} = \frac{8N}{f_{\mathrm{clk}}}. \tag{9}$$

The overall update time can be expressed as

$$T_{\mathrm{update}} = T_{\mathrm{shift}} + T_{\mathrm{fw}} + T_{\mathrm{latch}}, \tag{10}$$

where T_{fw} represents firmware execution time and T_{latch} represents the output-latch operation.

The propagation delay introduced by a cascaded shift-register chain is

$$T_{\mathrm{pd}} = N t_{\mathrm{pd}}, \tag{11}$$

where t_{pd} is the propagation delay of a single 74HC595 stage. Using the nominal value of 13 ns at 5 V and 25°C reported in the device documentation [2], an eight-device chain introduces approximately 104 ns of cumulative propagation delay.

For multiplexed acquisition, the complete scan time is approximated by

$$T_{\mathrm{MUX}} = C_{\mathrm{MUX}} \left(T_{\mathrm{sel}} + T_{\mathrm{settle}} + T_{\mathrm{ADC}} \right), \tag{12}$$

where T_{sel}, T_{settle}, and T_{ADC} represent channel-selection, signal-settling, and ADC-conversion times, respectively.

3.2.3 Power Model

The total power consumption of the expanded node is considered as

$$P_{\mathrm{total}} = P_{\mathrm{MCU}} + P_{\mathrm{MUX}} + P_{\mathrm{SR}} + P_{\mathrm{load}}. \tag{13}$$

For K simultaneously active outputs, the load component can be approximated by

$$P_{\mathrm{load}} \approx K V_{\mathrm{out}} I_{\mathrm{out}}, \tag{14}$$

where V_{out} and I_{out} are the output voltage and load current, respectively. Dynamic switching power can additionally be represented by

$$P_{\mathrm{dyn}} \approx \alpha C_{\mathrm{L}} V^2 f, \tag{15}$$

where α is the switching activity factor, C_{L} is the effective load capacitance, V is the supply voltage, and f is the switching frequency.

3.3 Hardware Design

The hardware implementation consists of separate analog-input and digital-output expansion modules connected to the host MCU. The 74HC4051 module connects eight analog channels ($Y0$–$Y7$) to a common output (COM). The three select inputs ($S0$–$S2$) are controlled by three MCU GPIOs, while the common output is connected to a single ADC input.

The 74HC595 module uses three MCU GPIOs for serial data (SER), shift clock ($SRCLK$), and storage latch ($RCLK$). Eight LEDs with 330 Ω series resistors are connected to the parallel outputs ($Q0$–$Q7$) of the first device. Additional 74HC595 devices are connected through $Q7'$ to form the cascade.

Table 1: Pin Mapping of the Proposed GPIO Expansion Interface

Signal	**74HC4051**	**74HC595**	**Arduino Uno**	**ESP32**	**Dir.**	**Function**
$S0$	Pin 11	–	D2	GPIO 4	OUT	MUX select bit 0
$S1$	Pin 10	–	D3	GPIO 5	OUT	MUX select bit 1
$S2$	Pin 9	–	D4	GPIO 6	OUT	MUX select bit 2
COM	Pin 3	–	A0	GPIO 36	IN	MUX analog output
SER	–	Pin 14	D5	GPIO 13	OUT	Serial data
$SRCLK$	–	Pin 11	D6	GPIO 14	OUT	Shift clock
$RCLK$	–	Pin 12	D7	GPIO 27	OUT	Storage latch
$Q7'$	–	Pin 9	–	–	OUT	Cascade output

The resulting interface requires four MCU pins for the 74HC4051 and three pins for the 74HC595 interface. The same logical allocation is retained across both platforms to provide a consistent basis for cross-platform evaluation.

3.4 Software Control Algorithms

The firmware was evaluated using three GPIO-control approaches: high-level Arduino functions, dedicated library abstractions, and direct register-level manipulation. The high-level approach uses functions such as `digitalWrite()`, `analogRead()`, and `shiftOut()`, whereas the register-level approach directly accesses MCU GPIO registers to reduce software overhead.

For multiplexer acquisition, the firmware sequentially selects each channel, allows the signal to settle, and then performs an ADC conversion. The procedure is summarized in Algorithm 1.

Algorithm 1 Multiplexer Channel Acquisition

Require: n_{channels}, select pins, ADC input
Ensure: ADC readings for all channels
for $ch = 0$ to $n_{\text{channels}} - 1$ **do**
  for $b = 0$ to $\log_2(n_{\text{channels}}) - 1$ **do**
    $val \leftarrow (ch \gg b)\ \&\ 1$
    Set select pin b to val
  end for
  Wait for the required settling time
  $readings[ch] \leftarrow$ ADC read
end for
return $readings$

For the shift-register interface, the data corresponding to all cascaded devices are shifted sequentially before the latch signal is asserted. This ensures that the outputs of all devices are updated simultaneously, as shown in Algorithm **??**.

Algorithm 2 Cascaded Shift-Register Update

Require: $data[0 \ldots N-1]$, SER, $SRCLK$, $RCLK$
Ensure: Updated outputs of N cascaded 74HC595 devices
Set $RCLK \leftarrow LOW$
for $ic = N - 1$ down to 0 **do**
  for $b = 7$ down to 0 **do**
    $bit \leftarrow (data[ic] \gg b)\ \&\ 1$
    Set $SER \leftarrow bit$
    Generate rising edge on $SRCLK$
  end for
end for
Set $RCLK \leftarrow HIGH$

The direct register-level implementation is subsequently used to quantify the execution overhead of high-level GPIO functions. This provides the basis for evaluating the relationship between GPIO expansion, firmware execution time, and real-time update requirements in the following experimental section.

4 Experimental Methodology

The proposed GPIO expansion framework was evaluated experimentally on Arduino Uno and ESP32 platforms under controlled hardware and software conditions. The prototypes were assembled on solderless breadboards, with jumper-wire lengths limited to approximately 15 cm to reduce the influence of parasitic effects on signal integrity. Timing measurements were obtained using a Saleae Logic 8 logic analyzer with a 100-MHz sampling capability, while system power consumption was measured using a calibrated 6.5-digit multimeter. Each experiment was repeated 100 times, and the results are reported as the mean with one standard deviation ($\pm\sigma$).

To provide an additional implementation-level check, the experimental procedures were also reproduced in the Wokwi simulation environment for Arduino and ESP32 platforms. The physical measurements were used as the primary basis for performance evaluation, while the simulation results were used to cross-check the observed timing and functional behavior.

Five experimental scenarios were considered to characterize the proposed architecture from GPIO utilization, execution efficiency, and scalability perspectives:

- **S1 – Direct GPIO Baseline:** Eight devices were connected directly to eight MCU GPIO pins to establish the reference configuration.
- **S2 – MUX Expansion:** Eight analog input channels were connected through a single 74HC4051, requiring three selection lines and one ADC input.
- **S3 – Shift-Register Expansion:** Digital-output configurations of 8, 16, and 32 channels were implemented using one, two, and four cascaded 74HC595 devices, respectively, with three MCU GPIO pins shared across the complete chain.
- **S4 – Port-Manipulation Optimization:** The MUX and shift-register configurations were repeated using direct GPIO register manipulation to quantify the execution-time reduction relative to high-level Arduino GPIO functions.
- **S5 – Large-Scale Cascade:** An eight-device 74HC595 chain providing 64 digital outputs was evaluated with respect to signal integrity and power consumption.

The principal hardware and software characteristics of the two experimental platforms are summarized in Table 2. The Arduino Uno uses an ATmega328P-based 8-bit AVR architecture operating at 16 MHz, whereas the ESP32 platform uses a 32-bit dual-core Xtensa LX6 architecture operating at 240 MHz. The different ADC resolutions and software environments were retained as platform-specific characteristics to evaluate the proposed framework under representative MCU conditions.

Table 2: Experimental Hardware and Software Configuration

Parameter	**Arduino Uno**	**ESP32**
MCU / Architecture	ATmega328P / 8-bit AVR	Xtensa LX6 / 32-bit dual-core
Clock frequency	16 MHz	240 MHz
Supply voltage	5 V regulated	3.3 V regulated
Flash / SRAM	32 KB / 2 KB	4 MB / 520 KB
ADC resolution	10-bit (0–1023)	12-bit (0–4095)
Timing measurement	`micros()` + Saleae Logic 8	`esp_timer_get_time()` + Saleae Logic 8
IDE / compiler	Arduino IDE 2.3 / avr-gcc 7.3	Arduino-ESP32 3.0 / xtensa-g++ 8.4

All configurations were evaluated using the same expansion topology and corresponding pin assignments described in Section III. This experimental design provides a consistent basis for comparing direct GPIO access, multiplexer-based input expansion, cascaded shift-register output expansion, and low-level GPIO optimization across the two MCU platforms.

5 Results and Discussion

The experimental evaluation examines the proposed GPIO expansion framework in terms of GPIO utilization, execution time, power consumption, and cascade scalability. The results are discussed for the individual expansion mechanisms and their combined practical implications.

5.1 GPIO Utilization

Table 3 summarizes the GPIO requirements of the evaluated configurations. Directly connecting eight peripherals requires eight MCU pins and gives a baseline GER of 1.00. In contrast, one 74HC4051 multiplexes eight analog channels using three select lines and one ADC input, reducing the GPIO requirement to four pins. This gives a GER of 2.00 and a 50.0% reduction in GPIO usage.

The improvement is more pronounced for digital-output expansion using the 74HC595. A single device provides eight outputs using three MCU pins, while four cascaded devices provide 32 outputs without increasing the MCU interface. The corresponding GER therefore increases from 2.67 to 10.67. With eight cascaded devices, 64 digital outputs are controlled using the same three GPIO pins, resulting in a GER of 21.33 and a 95.3% reduction in GPIO requirements relative to direct wiring.

These results highlight the complementary roles of the two expansion methods. The 74HC595 provides high GPIO efficiency for digital outputs, whereas the 74HC4051 requires an additional ADC input but enables multiplexed analog acquisition. Thus, their benefits depend on the type of I/O resource required by the target application.

Table 3: GPIO Utilization of the Evaluated Configurations

Configuration	Channels	MCU Pins	GER	GPIO Saved
Direct wiring	8	8	1.00	0.0%
74HC4051 ($\times 1$)	8	4	2.00	50.0%
74HC595 ($N = 1$)	8	3	2.67	62.5%
74HC595 ($N = 2$)	16	3	5.33	81.3%
74HC595 ($N = 4$)	32	3	10.67	90.6%
74HC595 ($N = 8$)	64	3	21.33	95.3%
74HC4051 ($\times 2$)	16	5	3.20	68.8%

Table 4: Execution-Time Comparison of High-Level and Direct GPIO Control

Operation	High-Level	Direct	Speedup	Platform
GPIO toggle	7.28 ± 0.12	0.125 ± 0.008	$58.2\times$	Uno
8-bit SR update	432 ± 8.4	7.5 ± 0.3	$57.6\times$	Uno
8-bit SR update	18.5 ± 1.2	0.83 ± 0.05	$22.3\times$	ESP32
8-channel MUX scan	984 ± 15	148 ± 4.2	$6.6\times$	Uno
8-channel MUX scan	112 ± 3.1	18.4 ± 0.9	$6.1\times$	ESP32
32-output SR update	1728 ± 31	30.1 ± 1.1	$57.4\times$	Uno
64-output SR update	3456 ± 62	60.2 ± 2.3	$57.4\times$	Uno

5.2 Execution Time and GPIO Optimization

The execution-time measurements are summarized in Table 4. Direct register-level GPIO control substantially reduces the software overhead associated with high-level GPIO functions. On the Arduino Uno, a single GPIO toggle decreases from 7.28 ± 0.12 μs to 0.125 ± 0.008 μs, corresponding to a $58.2\times$ reduction in execution time. Similarly, an 8-bit shift-register update decreases from 432 ± 8.4 μs to 7.5 ± 0.3 μs.

The same trend is observed for larger output configurations. Updating 32 outputs through four cascaded 74HC595 devices requires 1728 ± 31 μs with the high-level implementation, compared with 30.1 ± 1.1 μs using direct register-level control. For 64 outputs, the corresponding times are 3456 ± 62 μs and 60.2 ± 2.3 μs, respectively.

On the ESP32, the 8-bit shift-register update time decreases from 18.5 ± 1.2 μs to 0.83 ± 0.05 μs, corresponding to a $22.3\times$ reduction. The different improvement observed across the two platforms indicates that the benefit of low-level GPIO control depends on the underlying MCU architecture and software implementation.

For the 64-output configuration, the measured update time is approximately 60.2 μs with direct register-level control, compared with 3.456 ms using the high-level implementation. Under the tested conditions, this difference provides a substantial timing advantage for applications with short output-update intervals.

5.3 Power Consumption

Table 5 presents the measured power consumption of the baseline and expanded configurations. For the Arduino Uno, active power increases from 265 ± 8 mW for the baseline configuration to 285.8 ± 10 mW with eight cascaded 74HC595 devices. This corresponds to an increase of approximately 7.85%. With four shift registers, the increase is approximately 4.68%.

The addition of one 74HC4051 increases the measured active power from 265 ± 8 mW to 267.2 ± 8 mW, corresponding to approximately 0.83%. For the ESP32, adding four 74HC595 devices increases active power from 186 ± 6 mW to 192.4 ± 6 mW, corresponding to approximately 3.44%.

Under the tested conditions, the expansion devices introduce a relatively small increase in system-level power. At higher channel counts, however, the connected load can become the dominant power component. Consequently, the power requirements of the expansion ICs and the external loads should be considered separately when designing a high-density I/O node.

Table 5: Measured Power Consumption of Baseline and Expanded Configurations

Configuration	Quiescent	Active	Overhead
Arduino Uno baseline	247 ± 5	265 ± 8	–
Uno + 74HC4051	248.4 ± 5	267.2 ± 8	0.83%
Uno + 1×74HC595	248.8 ± 5	269.1 ± 8	1.55%
Uno + 4×74HC595	250.6 ± 5	277.4 ± 9	4.68%
Uno + 8×74HC595	252.3 ± 6	285.8 ± 10	7.85%
ESP32 baseline	168 ± 4	186 ± 6	–
ESP32 + 4×74HC595	170.2 ± 4	192.4 ± 6	3.44%

Table 6: Experimental and Calculated Cascade Characteristics of the 74HC595

N	Outputs	GPIO	GER	T_{pd}	Basis
1	8	3	2.67	0.013	Experimental
2	16	3	5.33	0.026	Experimental
4	32	3	10.67	0.052	Experimental
8	64	3	21.33	0.104	Experimental
16	128	3	42.67	0.208	Extrapolated
32	256	3	85.33	0.416	Extrapolated

5.4 Scalability and Cascade Behavior

The scalability of the 74HC595 architecture was experimentally evaluated for cascade depths of up to eight devices, corresponding to 64 digital outputs. Larger cascade depths were analytically extrapolated to examine the scaling behavior. As shown in Table 6, the number of outputs and the GER increase linearly with cascade depth, while the MCU interface remains fixed at three GPIO pins.

Using the nominal propagation delay of 13 ns per device, the calculated delay increases from 0.013 μs for one device to 0.416 μs for 32 devices. For the experimentally evaluated eight-device configuration, 64 digital outputs are controlled using three MCU GPIO pins, resulting in a GER of 21.33. The calculated device propagation delay remains small relative to the measured software-based update time, indicating that firmware execution and serial data transfer are the dominant timing factors under the evaluated conditions.

For a 32-device cascade, $32 \times 8 = 256$ serial clock pulses are required for one complete update. At an ideal clock frequency of 16 MHz, the corresponding serial-transfer time is approximately 16 μs, excluding firmware and synchronization overhead. Thus, increasing cascade depth improves GPIO efficiency but also increases the amount of data that must be transferred during each output update. The practical cascade depth should therefore be selected according to the required output-update rate and timing constraints of the target application.

6 Comparative Analysis

The proposed framework is compared with conventional GPIO interfacing and representative hardware-based expansion approaches in terms of GPIO efficiency, interface capability, scalability, cost, and timing suitability. The comparison is summarized in Table 7.

Table 7: Comparison of GPIO Expansion Approaches

Parameter	Direct	I^2C	SPI	74HC4051	74HC595	Proposed
GPIO efficiency	1:1	8:2	16:3	8:4	8:3	32:3
Analog input	Yes	No	No	8-ch	No	Yes
Digital output	Yes	Yes	Yes	No	Yes	Yes
Expansion	None	Addressed	CS-based	Cascade	$Q7'$ chain	Hybrid
Cost / 8 I/O	\$0	∼\$0.50	∼\$0.80	∼\$0.30	∼\$0.20	$<$ \$0.55
Real-time suitability	High	Low	Moderate	Moderate	High	High
Signal integrity	Excellent	Good	Good	Good	Excellent	Good–Excellent

The comparison demonstrates that the individual expansion techniques address different GPIO constraints. Direct GPIO provides the simplest and most deterministic interface but requires one MCU pin per channel. The 74HC4051 reduces the GPIO requirement for analog acquisition by multiplexing eight channels onto a single ADC input, while the 74HC595 provides substantially higher digital-output density because additional devices can be cascaded without increasing the three-wire MCU interface.

I^2C- and SPI-based expanders provide flexible bus-level expansion, although their effective performance depends on bus frequency, transaction overhead, and device architecture. In contrast, the proposed framework combines the 74HC4051 and 74HC595 to support both analog input and high-density digital output expansion within a common architecture. The experimental results further show that firmware implementation has a significant effect on the achievable update rate, with direct GPIO manipulation providing up to a $58\times$ reduction in execution time relative to the Arduino high-level GPIO interface. Thus, the proposed architecture provides a practical trade-off between GPIO efficiency, mixed I/O capability, and execution performance for resource-constrained embedded and IoT nodes.

7 Deployment Applications

The proposed architecture is suitable for embedded and IoT systems where GPIO availability is a primary constraint. Its mixed analog–digital expansion makes it applicable to industrial monitoring, smart agriculture, wearable sensing, and robotics.

- **Industrial IoT:** Four 74HC595 devices provide 32 digital outputs using three MCU pins, while two 74HC4051 devices provide 16 analog inputs using five pins. Thus, a 48-channel mixed-I/O node can be implemented using eight GPIO resources.
- **Smart Agriculture:** The same 32-output and 16-input configuration can support valve/relay control together with soil and environmental sensing, making it suitable for compact irrigation and monitoring controllers.
- **Medical Wearables:** The 74HC4051 can multiplex multiple low- and moderate-bandwidth analog biosignals onto a single ADC input. However, the sequential nature of acquisition makes the architecture unsuitable where simultaneous sampling is required.
- **Robotics and Edge-AI:** Two 74HC4051 devices and four 74HC595 devices provide 16 analog inputs and 32 digital outputs using eight MCU GPIO resources, leaving additional resources for communication and computational peripherals.

8 Limitations

The proposed framework provides substantial GPIO savings with low hardware overhead, while several practical design considerations remain. The 74HC4051 performs sequential channel acquisition, so the achievable sampling rate depends on channel switching, settling, and ADC conversion time. For cascaded 74HC595 devices, output capacity increases linearly without additional MCU pins, while the serial update time increases with cascade depth:

$$T_{\text{shift}} = \frac{8N}{f_{\text{clk}}}. \tag{16}$$

Direct GPIO manipulation further improves execution speed but requires MCU-specific implementation. At higher channel counts, output loading also becomes important; for example, 64 outputs at $10\ \text{mA}$ each correspond to a theoretical load of $640\ \text{mA}$. These factors can be accommodated through appropriate sampling, firmware, cascade, and power-distribution design according to the target application.

9 Conclusion

This study presented and experimentally evaluated a compact GPIO expansion framework that combines the 74HC4051 analog multiplexer with cascaded 74HC595 shift registers and firmware-level GPIO optimization. The results demonstrate that the approach can substantially increase I/O capacity without a proportional increase in MCU pin usage. The 74HC4051 provided eight analog channels using four MCU pins, achieving a GER of 2.00, while four cascaded 74HC595 devices provided 32 digital outputs using three MCU pins, corresponding to a GER of 10.67 and 90.6% GPIO savings relative to direct wiring. Direct GPIO manipulation further reduced execution time by up to $58.2\times$ on the Arduino Uno, demonstrating that firmware-level optimization can be as important as the hardware expansion mechanism for timing-sensitive applications.

The measured power overhead remained comparatively low across the tested configurations, while the eight-device 74HC595 configuration demonstrated the practical feasibility of expanding the digital interface to 64 outputs without increasing the three-pin serial interface. The combined 74HC4051–74HC595 architecture further enables mixed analog and digital expansion; in the evaluated configuration, 16 analog inputs and 32 digital outputs can be addressed using eight MCU GPIO resources. These results indicate that the proposed framework offers a practical and low-cost solution for GPIO-constrained embedded and IoT nodes, particularly in applications requiring a combination of sensing and actuation.

The framework also involves predictable design trade-offs. Multiplexed analog acquisition is sequential, while increasing the shift-register cascade increases serial update time. In addition, low-level GPIO optimization is platform dependent and higher output counts require appropriate power distribution. These considerations provide clear design criteria for selecting the expansion configuration according to the timing, sampling, and loading requirements of a target system.

Future work will focus on hardware-accelerated shift-register control using SPI/DMA, portable low-level GPIO abstractions, and larger-scale validation on application-specific IoT platforms. Further investigation of adaptive GPIO scheduling and hybrid expansion architectures could also improve the flexibility of the framework for systems with dynamic I/O requirements.

References

[1] Texas Instruments. SN74HC4051 8-Ch Analog MUX Datasheet. `https://www.ti.com/lit/ds/symlink/sn74hc4051.pdf`, 2015. Datasheet.

[2] Nexperia. 74HC595 8-Bit Shift Register Datasheet. `https://assets.nexperia.com/documents/data-sheet/74HC_HCT595.pdf`, 2020. Datasheet.

[3] R. Kumar, V. Singh, and P. Sharma. Low-power iot node for smart agriculture using arduino and lora. *IEEE Internet of Things Journal*, 8(11):9123–9135, 2021.

[4] T. Rodrigues, F. Alves, and M. Costa. Compact wearable ecg/ppg acquisition using arduino nano with analog multiplexing. *Sensors*, 21(7):2310, 2021.

[5] NXP Semiconductors. PCF8574/PCF8574A I2C I/O Expander Datasheet, 2019. Revision 8, Datasheet.

[6] Microchip Technology. MCP23017 16-Bit I/O Expander Datasheet. Technical Report DS20001952C, Microchip Technology, 2016.

[7] Microchip Technology. ATmega328/P Datasheet. Technical Report DS-40002061B, Microchip Technology, 2018.