EARIC: Exploiting ADC Registers in IoT and Control Systems

Eyasu Getahun Chekole, Rajaram Thulasiraman, and Jianying Zhou

***Abstract*—An analog-to-digital converter (ADC) is a critical part of most computing systems that converts analog signals into quantifiable digital values. Since most digital devices operate only on digital values, the ADC acts as an interface between the digital and analog worlds. As such, ADCs are commonly used in a wide-range of applications, including internet of things (IoT), industrial control systems (ICS), cyber-physical systems (CPS), audio/video devices, medical imaging, digital oscilloscopes, and cell phones, among others. For example, programmable logic controllers (PLCs) in ICS/CPS often make control decisions based on digital values converted from analog signals by ADCs. Due to its crucial role in various applications, ADCs are often targeted by a wide-range of physical and cyber attacks. Attackers often exploit vulnerabilities that could be found in the software/hardware of ADCs. In this work, we first conduct a deeper study in the ADC conversion logic to investigate relevant vulnerabilities that were not well explored by prior works. As a result, we manage to identify exploitable vulnerabilities on certain ADC registers that are involved in the analog-to-digital conversion logic. As a proof of concept, we construct and develop three attack techniques by exploiting the vulnerabilities identified. Finally, we test the attacks on a mini-CPS testbed we designed using IoT devices, analog sensors and actuators. Our experimental results reveal high effectiveness of the proposed attack techniques in misleading PLCs to make incorrect control decisions in CPS. We also analyze the impact of such attacks when launched in real-word CPS testbeds.**

***Index Terms*—ADC Security, ADC Vulnerabilities, ADC Attacks, CPS Security, ICS Security, PLC Attacks, IoT Security**

I. Introduction

A Signal that represents a continuous range of values that varies over time is referred to as an analog signal [1]. Such signals can also be characterized by natural phenomena, such as lightning, earthquake, wind speed, volcano, sound waves, weight measurements, etc. Analog signals are often in the form of electrical energy, such as voltage, current or electromagnetic power. These signals typically come from sound, light, temperature or motion sensors. However, analog signals, which have more than 2 distinct readings, are not compatible in digital computation. This is because, digital devices, such as computers and microcontrollers (MCUs)[1], operate only on binary or digital values, i.e., 0s and 1s. As such, it is required to convert analog signals to digital values (i.e., discrete-time values) in order to process them using digital devices. This is where the analog-to-digital converter (ADC) [2] comes in handy. As the name implies, ADC is a system that converts an analog signal (i.e., continuous voltage values) to digital values, which can be understood by most computers and MCUs for digital computation. Most state-of-the-art MCUs have an inbuilt ADC. Therefore, such binary encoding of analog signals facilitates the interface between digital circuits and the real world.

The transformation from an analog signal domain to a digital signal domain can take place at baseband i.e., using direct conversion receivers or at any intermediate frequency (IF) depending upon the hardware design and its architecture. The analog-to-digital conversion logic of ADC typically involves three steps: sampling and holding (S/H), quantization and encoding [3]. A high-level representation of a typical ADC process is shown in Figure 1.

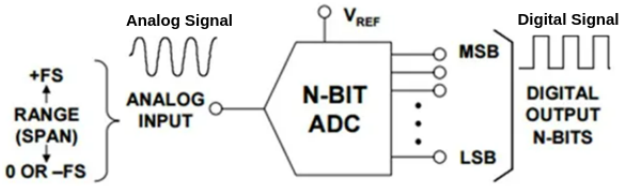

Fig. 1: Input and output definitions of ADC [4]

ADCs are widely used in most digital systems that involve analog signals in its computations. These includes IoT, control systems (e.g., ICS/CPS), image processing, digital multimeters, cell phones, and medical imaging, to name a few. For example, PLCs [5] in ICS/CPS [6], [7] often make control decisions based on the input obtained from analog sensors (e.g., temperature, pressure and force sensors). However, they cannot directly use the analog input as they cannot understand analog signals. Hence, they have inbuilt ADCs that serve to convert the analog signals into digital values. The PLCs will then use these digital values to make control decisions [4].

Since ADC is an integral and critical part of most computing systems, such as ICS/CPS, it has been often targeted by cyber criminals. A wide-range of physical or cyber attackers target ADCs to attack the underlying or hosting systems. These attacks exploit various types of vulnerabilities that can be found in the hardware or software of ADCs. For example, Bolshev et al. [8] has exploited vulnerabilities in the sampling frequency and dynamic range of the ADC conversion logic. There are also attacks that exploited the strong correlation

E. G. Chekole is a Research Fellow with Singapore University of Technology and Design, 487372, Singapore. (e-mail: eyasu_chekole@sutd.edu.sg)

R. Thulasiraman was a master's student with Singapore University of Technology and Design, 487372, Singapore. (e-mail: thulasiraman_rajaram@alumni.sutd.edu.sg)

J. Zhou is with Singapore University of Technology and Design, 487372, Singapore. (e-mail: jianying_zhou@sutd.edu.sg)

[1]https://www.arrow.com/en/research-and-events/articles/engineering-basics-what-is-a-microcontroller

between the ADC digital output codes and the ADC supply current waveforms [9]. Other attacks exploited fast attack automatic gain control (AGC) vulnerability in ADC [10], [11], [12]. Other class of attacks exploited the DAC-to-DAC crosstalk vulnerability in the ADC conversion logic [13], [14], [15]. Numerous side-channel attacks have also exploited various vulnerabilities in ADC [16], [17], [18], [19], [20], [21], [22]. There are also a class of attacks that exploit memory-safety vulnerabilities to corrupt the process memory or execution flow of programs at runtime [23], [24], [25]. Hardware trojan attacks were also launched on the analog circuits of ADCs [26]. Other researchers have conducted a security analysis on the output signals of the ADC datapath and its control unit [27] and ADC power noise measurement attacks [28]. However, we are not aware of existing attack techniques in the literature that specifically exploit vulnerabilities related to ADC registers (the smallest and fastest memory locations that are built into the processor). Hence, this work aims to bridge this gap in ADC security.

In this work, we first conduct a deeper analysis and study on ADCs to explore exploitable vulnerabilities in the analog-to-digital conversion logic. In particular, we study the various types of ADC registers involved in the analog-to-digital conversion process. After systematically analyzing the nature of these registers, we find out that most of them are vulnerable to a manipulation attack. For example, an attacker may modify or clear the default flag values of the registers or the data stored in it. This is because, there are no underlying protections in place to protect them against such unauthorized manipulations. Such attacks can be carried out through malicious code injection or malevolent system configuration to deceive the ADC output. In control systems, such as ICS/CPS, such manipulated ADC outputs can mislead PLCs to make wrong control decisions. This may, in return, result in a disaster to the physical plant of the control system. To the best of our knowledge, there are no prior attacks presented in the literature that specifically targeted ADC registers to deceit the ADC conversion process.

To scrutinize the actual exploitability of the registers, we design EARIC (Exploiting ADC Registers in IoT and Control systems) – a scheme comprising the different attack techniques we designed to manipulate the ADC conversion logic. To achieve this, we target certain critical ADC registers used in the ADC conversion logic, such as the ADC multiplexer selection register (ADMUX), the analog comparator control and status register (ACSR), and the two ADC data registers, namely ADC High register (ADCH) and ADC Low register (ADCL). By exploiting these registers, we managed to fake outputs of the ADC. That means, we force the ADC to return undesirable digital values from analog signals. This is achieved by manipulating certain flags of the registers or by systematically synthesising certain bit values of different registers. By doing so, we design and perform three types of attacks on the ADC conversion logic **(1)** Deceiving the ADC conversion process – changing the expected ADC output into a totally different value; **(2)** Creating denial of service (DoS) in the ADC process – hanging the ADC conversion process and and causing system unavailability; **(3)** Resetting the ADC conversion process – making the ADC to return always an empty outcome. Finally, we assess and evaluate the effectiveness of the proposed attacks using a minimalist CPS (mini-CPS) testbed we designed using IoT devices, such as Arduino (as a soft PLC), analog sensors and actuators. The proposed attacks can also be integrated and tested in systems other than IoT and control systems.

In general, we make the following technical contributions in this work.

1) We conduct a deeper study in the ADC conversion logic and identify vulnerabilities on the ADC registers used in the analog-to-digital conversion process.
2) We design and perform three types of attacks by exploiting the vulnerabilities we identified.
3) We assess and evaluate the effectiveness (in terms of accuracy, efficiency and impact) of the proposed attacks using an IoT-based mini-CPS testbed we designed.

Organization. The rest of the paper is structured as follows. In Section II, we provide relevant background on the ADC conversion logic and CPS. In Section III, we present our threat model. In Section IV, we provide a detailed discussion of the proposed attacks and its respective outcomes. In Section V, we discuss our experimental design. In Section VI, we present our evaluation and discussion of the performed attacks and its impact on real-world CPS systems. In Section VII, we discuss relevant prior works on ADC-based attacks. Finally, Section VIII concludes the paper by outlining future works.

II. Background

In this section, we provide relevant background information to this work. Specifically, we provide a high-level information on the ADC conversion logic and cyber-physical systems (CPS). For easy reference, Table I lists out all the relevant acronyms and notations used in this paper.

A. Overview of ADC

1) Analog and digital signals: As highlighted in the introduction, analog signals are electromagnetic signals that are characterized by a series of continuous values that varies with time. These signals are illustrated in Figure 2. Such signals can be obtained from sound, temperature, light, and motion phenomena using analog sensors.

Analog signals can be used as an input to solve various real-world problems. For example, IoT services and control systems can them to automate or control processes. However, these signals cannot be directly used since digital devices, such as computers and microcontrollers, can read only digital values. Hence, the analog signals need to be first converted to digital signals before it is used by digital devices further computations. Unlike analog signals, which are represented by a sequence of continuous values, digital signals are broken down into a set of discrete values with time series or sampling rates. It usually have only two values – high (1) and low (0). Consequently, all values in digital signal transmissions are in the form of 0's and 1's. Digital signals are illustrated in Figure 2.

TABLE I: Description of acronyms and notations

Notation	Description
ACBG	Analog comparator band gap
ACD	Analog comparator disable
ACIC	Analog comparator input capture enable
ACI	Analog comparator interrupt
ACIE	Analog comparator interrupt enable
ACIS	Analog comparator interrupt mode select
ACME	Analog comparator multiplexer enable
ACO	Analog comparator output
ACSR	Analog comparator control and status register
ADC	Analog-to-digital converter
ADMUX	ADC multiplexer selection register
ADCH	ADC high register
ADCL	ADC low register
ADEN	ADC enable
ADFR	ADC free running
ADIE	ADC interrupt enable
ADIF	ADC interrupt flag
ADPS	ADC pre-scaler selection
ADSC	ADC start conversion
AREF	Analog reference voltage
ADLAR	ADC left adjust result
ADMUX	ADC multiplexer selection register
AIN	Analog input pin
AVCC	Analog voltage common collector
CPS	Cyber-physical systems
DAC	Digital-to-analog converter
DoS	Denial of service
FS	Full scale
GND	Ground
GUI	Graphical user interface
HMI	Human machine interface
ICS	Industrial control systems
IoT	Internet of things
IRQ	Interrupt request
LM35	An analog temperature sensor
LSB	Least significant bit
MCU	Microcontroller
MSB	Most significant bit
MUX	Multiplexer selection register
PCM	Pulse code modulation
PLC	Programmable logic controller
PSA	Power side-channel attack
REFS	Reference selection
S/H	Sampling and holding
SAR	Successive approximation register
SCADA	Supervisory control and data acquisition
SoC	System-on-Chip
SRAM	Static random-access memory
T/C1	Timer/Counter1
VREF	reference voltage

2) Analog to digital conversion: The conversion of analog signals to digital signals is carried out by an analog-to-digital converter (ADC). In other words, ADCs serve to convert continuous-time analog signals to discrete-time digital signals, which will be consumed by digital devices for digital computations. Hence, most digital devices have builtin ADC, integrated with their processors. They can also be connected to an external ADC. A simple ADC conversion schematic is depicted in Figure 4

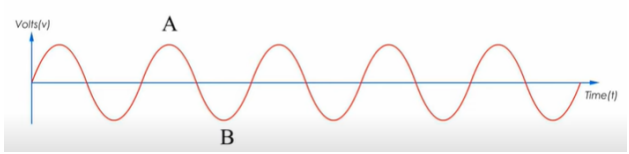

Fig. 2: Analog signals

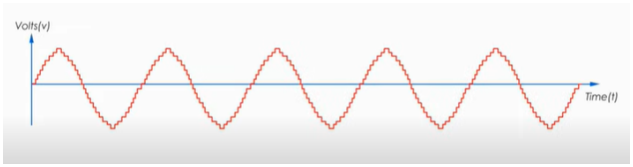

Fig. 3: Digital signals

ADCs convert analog signals to digital signals using pulse code modulation (PCM)[2] method, which involves three main steps – sampling, quantizing and encoding [29], [30]. ADCs on most microcontrollers, e.g., PIC32[3], typically have a 10-bit wide resolution, i.e., with 1024 quantization levels. Most microcontrollers also have multiple analog input channels due to their multiplexed ADC. For example, the PIC32MX460F512L[4] microcontroller has 16 10-bit wide ADC channels. ADCs also involve a wide-range of memory registers that play various roles in the analog-to-digital conversion process. For example, the ADC's output data, i.e., the converted digital value, is stored in a 16-bit double data registers, i.e., ADCH (8-bit size) and ADCL (8-bit size). A high-level architecture of the ADC conversion logic involving the main memory registers is illustrated in Figure 7. A detailed discussion of some of the registers is also provided in Section IV.

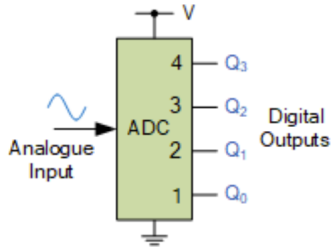

Fig. 4: Simple ADC design [31]

Although it is hard to cover all details on how the ADC conversion logic works, we would like to provide a high-level

[2]https://www.tutorialspoint.com/digital_communication/digital_communication_pulse_code_modulation.htm

[3]https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus/pic32-32-bit-mcus

[4]https://www.microchip.com/en-us/product/PIC32MX460F512L

background on the *ADC analog comparator* below as it is an essential building block in ADC and we also extensive use it in our proposed approach.

3) ADC Analog Comparator: The ADC analog comparator [32] is an electronic circuit that compares two input voltages and produces an output. The analog comparator output (ACO) indicates which of the inputs is greater or lesser. A typical ADC analog comparator consists of a positive input and a negative analog input. The magnitude of two different voltage levels are measured by these 2 inputs. One comparator input uses the voltage input signal (V_{IN}) and the other comparator input uses the reference voltage (V_{REF}). The digital logical output state of the comparator is determined by comparing the two voltage levels "1" or "0" at the input of the comparator. The V_{IN} and V_{REF} are compared and applied to the other inputs. If the V_{IN} is lower than the V_{REF} (i.e., $V_{IN} < V_{REF}$), the output of the comparator will be "OFF". Vice versa, it is "ON" if the V_{IN} is greater than V_{REF}. As a result, the comparator compares the two voltage levels to determine which one is higher. The voltage divider formed by the resistors R1 and R2 (cf. Figure 5) gives V_{REF}. If the values of the R1 and R2 are the same, then the V_{REF} level is clearly half the supply voltage. The output of the open collector output comparator will be HIGH if V_{IN} < V_{REF}/2, and the output will be LOW if V_{IN} > V_{REF}/2. It acts as a 1-bit ADC. Successfully break up the supply voltage by adding a resistor to the voltage network.

A $2^n - 1$ comparator is required to convert to binary output ("n"-bits), where "n" is usually 8 to 16. In Figure 5, one comparator is used based on a single bit ADC (i.e., $2^1 - 1$). If a 2-bit ADC is used, three comparators (i.e., $2^2 - 1$) will be required, because there will be four different voltage levels. Figure 6 depicts a 2-bit ADC analog comparator circuit.

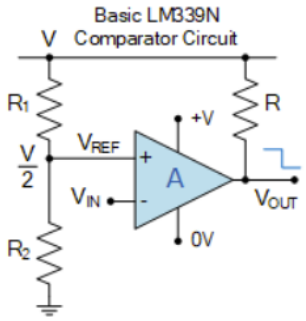

Fig. 5: ADC analog comparator [31]

B. Overview of CPS

Cyber-physical systems (CPS) are engineering systems where computations and communications are firmly integrated with physical entities to automate and control industrial processes through feedback control [6], [7]. It comprises the following main entities [33]: physical plant (the physical system where actual processes take place), sensors (devices that read state information of physical processes), PLCs (embedded devices that issue control commands based on sensor inputs),

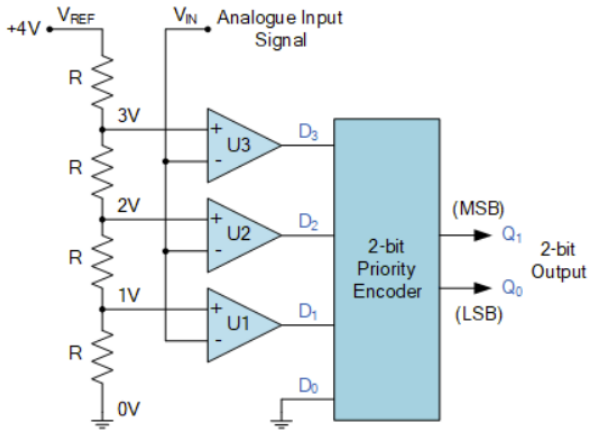

Fig. 6: A 2-bit ADC analog comparator circuit [31]

actuators (physical entities that implement control commands issued by PLCs), SCADA (a software designed for process monitoring and controlling), HMI (a system to display the state information of physical processes), and historian server (a server used to store operational and historical data). An abstraction of a typical CPS model is depicted in Figure 8. A typical CPS is also constrained by stringent real-time and availability requirements [34].

As discussed above, the PLC is at the heart of the CPS. It issues control commands based on the inputs obtained from sensors. However, the sensors could be digital or analog. In the latter case, the PLC cannot read analog signals like many other digital devices (see the discussion in Section II-A1). Hence, the ADC is required to convert the analog signals to digital values before the PLC uses them to make control decisions. To facilitate the conversion process, most PLCs nowadays come with inbuilt ADCs.

III. Threat Model

In our threat model, we consider threats that may target digital systems, such as IoT and control systems, by exploiting ADC-related vulnerabilities in the devices. In particular, we focus on threats that target certain vulnerabilities of the ADC registers involved in the analog-to-digital conversion logic. We assume that the the targeted digital devices are accessible physically (e.g., via cable connection) or remotely (via internet). Hence, we consider both physical (insider attacks) or remote (cyberattacks) threats in this model.

Nowadays, most digital devices, e.g., PLCs in CPS, has inbuilt ADCs, which serve to convert analog sensor inputs to digital values. Once an attacker has access to the PLC, it can manipulate values of the targeted registers. For example, it can change certain flags of the registers or modify the data stored in it. This can be achieved in various ways. One way is by injecting a malicious code (i.e., a code that changes the default values of the registers) into the PLC firmware. For example, we perform our proposed attacks in Arduino by injecting the malicious code into its firmware (details are provided in Section IV). Another way is, by changing register values (e.g.,

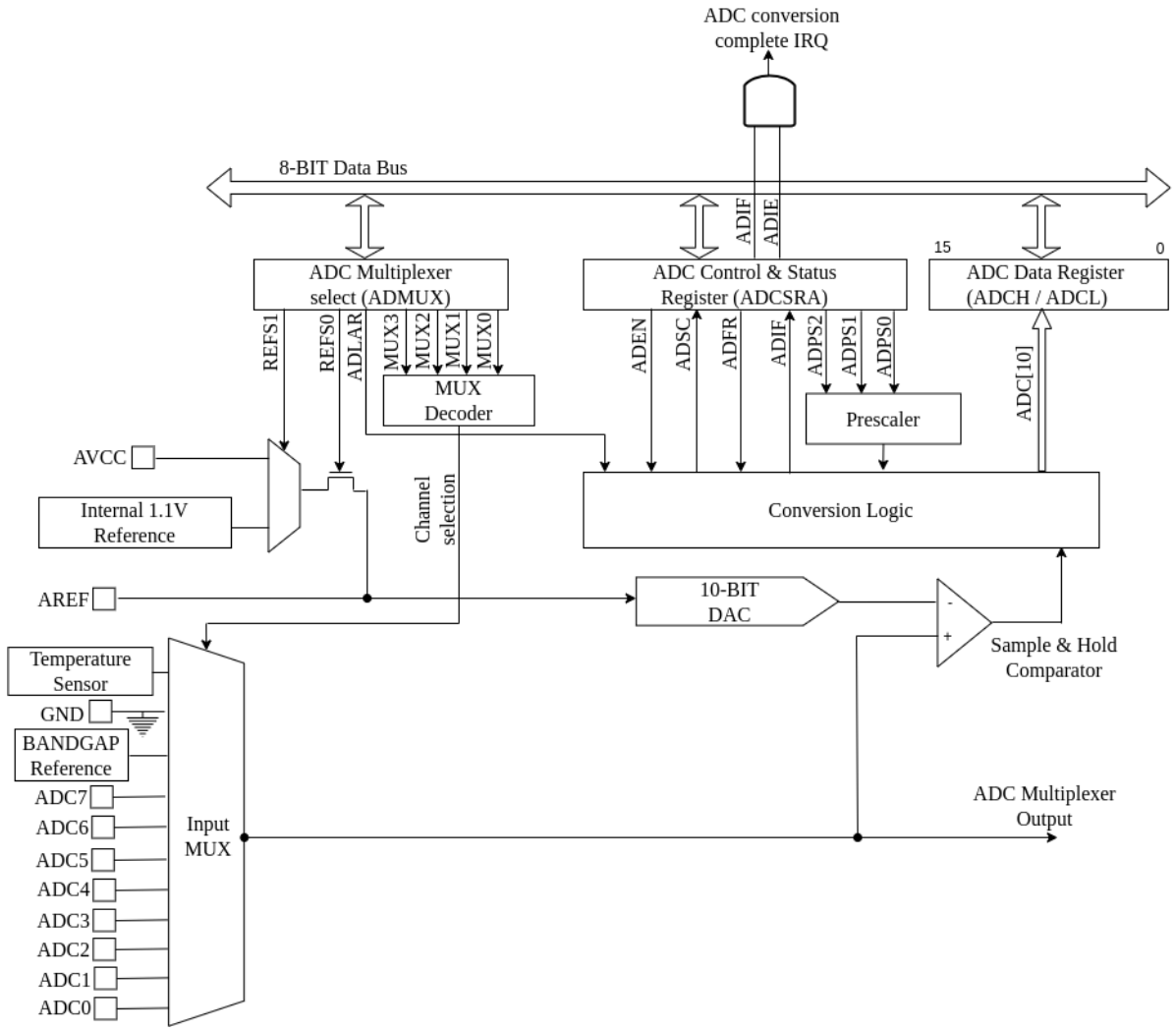

Fig. 7: A high-level architecture of ADC with registers

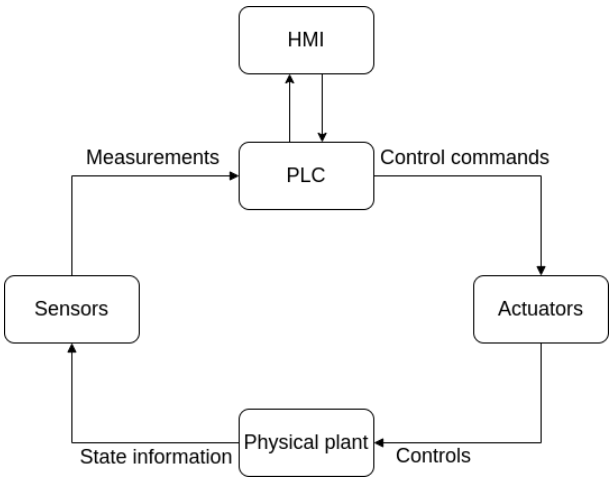

Fig. 8: An abstraction of a CPS model

by turning ON or OFF certain flags) via system configuration. Because, system settings of most PLCs can be configured via command-line or through graphical user interface (GUI). In either way, an attacker may able to manipulate the ADC registers in the targeted digital device.

The attacks could also be performed physically or remotely. In the former case, the attack can be performed by physically modifying the source-code or configuration of the firmware. In the latter case, the attack can be launched by sending a malicious command over internet. Because, it is now possible to upload code to the controller, e.g., to Arduino board, over the Internet by flashing the code on a special bootloader[5].

For example, by manipulating the flags or data values of the ADC registers in PLCs, the following are some of the attack scenarios that a threat can launch on CPS:

- Deceiving outcome of the ADC conversion process by manipulating certain register values, e.g., *Attack 1* in our proposed attacks (cf. Section IV-B1).
- Creating denial-of-service (DoS) attack by systematically synthesizing some register values, e.g., *Attack 2* in our proposed attacks (cf. Section IV-B2).
- Systematically resetting the ADC conversion process e.g., *Attack 3* in our proposed attacks (cf. Section IV-B3).

[5]https://github.com/loathingKernel/ariadne-bootloader

IV. EARIC: The Proposed Attacks

A. Overview

In this section, we introduce EARIC – a scheme comprising the three attack techniques we designed. As discussed in the preceding sections, we propose and develop new ADC attack techniques by exploiting the registers used in the analog-to-digital conversion logic. To simplify the presentation of our proposed attack techniques, it is essential to highlight how the ADC conversion logic works and the relevant registers involved in the process. As such, in addition to the background information provided in Section II, we further discuss details of the ADC conversion logic as below.

Based on the ADC conversion logic, major ADC types can be classified as dual slope, flash, pipeline, successive approximation method (SAR), and delta-sigma ($\Delta\sum$). SAR and $\Delta\sum$ are the most widely used ADC types. In this work, we use SAR with 10-bit resolution. As discussed in section II-A, ADC converts the voltage value on the analog input pin and returns a digital value from 0 to 1023 (for a 10-bit wide ADC), relative to the reference value. The analog input channel is selected using an analog multiplexer [35], and the input value is processed in ADC with a reference voltage for certain clock timings. When the analog-to-digital conversion is completed, the output result (often called the "ADC output data") is stored in the two ADC data registers, i.e., ADCH and ADCL (each 8-bit wide). More precisely, for a 10-bit ADC resolution, the ADC output will be stored in the 9^{th} to 0^{th} bits of the ADCH and ADCL data registers (cf. Figure 9). A typical schematic of ADC is illustrated in Figure 7. In Figure 7, ADC0 to ADC7 represents the input pins for the analog input signals. The multiplexer (MUX) selects the input voltage from the pins and transfers it to the registers.

As shown in Figure 7, several registers are involved in the ADC conversion logic. As highlighted in the preceding sections, these registers are vulnerable to attacks since its default values (data or flags) can be manipulated by an attacker. This is because, there are no security mechanisms in place to protect these registers against such malevolent manipulations. In this work, we exploit such weaknesses to perform three types of attacks on the ADC conversion logic. A detailed account of the attacks is provided in the following section.

B. The Proposed Attacks

As mentioned in the preceding sections, we perform three types of attacks on ADC to scrutinize exploitability of its registers. In particular, we perform the attacks by exploiting three of the most critical ADC registers, such as ADMUX, ACSR, and the ADC data registers (i.e., ADCH and ADCL). The attacks are tested using a mini-CPS testbed simulating an alarm system based on an analog temperature sensor. In brief, the system triggers an alarm when the temperature read is beyond a threshold. A detailed account of the testbed is provided in Section V. Below, we discuss each of the proposed attack techniques and its respective outcomes.

1) Deceiving the ADC conversion logic (Attack 1): With the first attack, we deceive the ADC conversion logic by manipulating the ADMUX register. The ADMUX register is used to select the reference voltage as well as to determine which analog input channel is to be chosen. Furthermore, this register is used to determine whether the ADC output data should be left-justified (i.e., the output data is to be read from the left-most bits) or right-justified (i.e., the output data is to be read from right-most bits) with respect to the 16-bit ADC data registers (i.e., ADCH + ADCL). As shown in Table II, the ADMUX register comprises 8 bits. A high-level discussion of the bits is provided as follows.

- *REFS (Reference Selection Bits)*: REFS1 (Bit 7) and REFS0 (Bit 6) are reference selection bits in ADMUX that are used to select the voltage reference for the ADC. The internal voltage reference options may not be used if an external reference voltage is applied to the AREF pin.
- *ADLAR (ADC Left Adjust Result)*: ADLAR (Bit 5) affects the presentation of the ADC output data in the ADC data registers (refer Section IV-B3). Depending on the value set to the ADLAR bit, the ADC output data can be either right-justified (i.e., ADLAR = 0) or left-justified (i.e., ADLAR = 1) in the ADCH and ADCL data registers. The default mode is right-justified. The left-justified mode is not supported by most microcontrollers, including the Arduino board we used in our experimental setup (cf. Section V).
- *MUX3 (Multiplexer)*: MUX3 (Bit 0 to 3) are the analog channel selection bits that are used to select the analog input channel (refer ADC0 to ADC7 in Figure 7). A detailed account of how the analog channel selection bits work in ADC can be found in [36].

Attack synopsis: The default values of the ADMUX register bits are shown in Table II. That is, REFS1 is '1', REFS0 is '1', ADLAR is '0', and MUX0 to MUX3 is '0'. As discussed above, the value of ADLAR affects the presentation of the ADC output data in the ADCH and ADCL data registers. By default, the ADC output data is right-justified (i.e., ADLAR = 0). That means, the output data will be read from the 9^{th} to 0^{th} bits of the ADCH and ADCL data registers (for a 10-bit ADC resolution). The ADCH and ADCL data presentation with respect to the ADLAR value (i.e., '0' or '1') is illustrated in Figure 9. However, as shown in Table III, the ADLAR bit of the ADMUX register can be set to '1' to reverse the ADC output data presentation (i.e., left-justified). Meaning, the ADC output data will be read from the 15^{th} to 6^{th} bits, where the 15^{th} to 10^{th} bits contain garbage (junk) data as shown in Figure 9. When the digital device (e.g., the PLC in CPS) tries to read the ADC output data, it will be referred to the garbage location, which returns an undesirable value (often a very high value). In practice, this attack might be achieved in different ways. For example, it could be launched by sending a malicious ADC command to the PLC at runtime or by systematically synthesising and injecting a malicious code to the PLC firmware. In our case, we follow the latter. We inject the following code into the Arduino firmware, which sets the ADLAR bit to '1'.

$$ADMUX \mid = (1 << 5);$$

TABLE II: ADMUX register bits with its default values

ADMUX Bits	REFS1 (Bit 7)	REFS0 (Bit 6)	ADLAR (Bit 5)	- (Bit 4)	MUX3 (Bit 3)	MUX2 (Bit 2)	MUX1 (Bit 1)	MUX0 (Bit 0)
Read/ Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Default Values	1	1	0	0	0	0	0	0

TABLE III: ADMUX register bits after manipulating the ADLAR bit

ADMUX Bits	REFS1 (Bit 7)	REFS0 (Bit 6)	ADLAR (Bit 5)	- (Bit 4)	MUX3 (Bit 3)	MUX2 (Bit 2)	MUX1 (Bit 1)	MUX0 (Bit 0)
Read/ Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
Bit Values (ADLAR = 1)	1	1	1	0	0	0	0	0

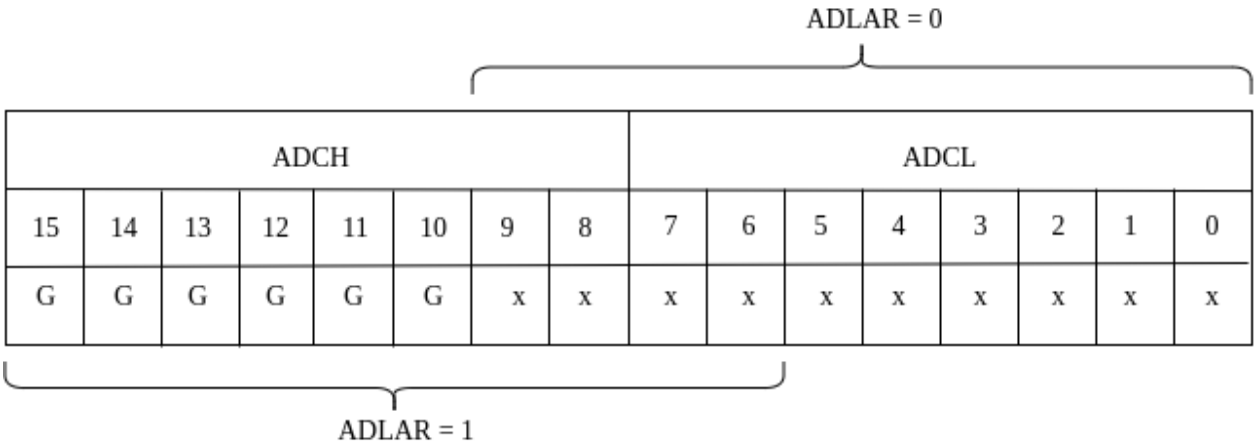

Fig. 9: ADC output data presentation in ADCH and ADCL registers with respect to the ADLAR value
Note: *"G" is for garbage data, "x" (from bit 9 to 0) represents the ADC output data values in binary format, i.e., 0's and 1's. For example, Table IV shows how a temperature reading of 24.49 is stored in the ADCH and ADCL data registers.*

After performing the above attack on our experimental setup, the ADC was forced to return a temperature of 1588.13℃ from the analog temperature sensor even though the actual temperature reading is 24.49℃. The output of this attack is depicted in Figure 11. This misleads the PLC to issue and send a wrong control command (i.e., "ON" command) to the actuator, i.e., a siren alarm set in our experimental setup (refer Section V). As a result, the siren alarm was triggered even though the actual temperature was below the threshold. That means, the wrong ADC read from the garbage location misleads the PLC to make a wrong control decision, which in turn could cause a disaster or damage to the CPS plant.

In sum, the main aim of this attack is deceiving the ADC output data presentation on the ADC data registers (i.e., ADCH and ADCL) by manipulating the ADLAR value on the ADMUX register. Consequently, PLCs will be forced to read undesirable ADC output data, hence misleading them to make wrong control decisions. A high-level illustration of this attack is provided in Figure 10.

2) Creating a DoS attack on the ADC process (Attack 2): In this attack, we create a denial of service (DoS) attack on the ADC conversion process by manipulating the ADC analog comparator control and status register (ACSR). As discussed in Section II-A3, the ADC analog comparator is an essential part of the ADC conversion process. it is managed and controlled by the ACSR register. As depicted in Table V, the ACSR register is represented by 8 bits comprising Analog Comparator Interrupt Mode Select (ACIS0 and ACIS1), Analog Comparator Input Capture Enable (ACIC), Analog Comparator Interrupt Enable (ACIE), Analog Comparator Interrupt (ACI), Analog Comparator Output (ACO), Analog Comparator Band Gap (ACBG) and Analog Comparator Disable (ACD). All the ACSR bits except bit 5 (which is read-only) are readable and writable (R/W). The default value of these bits is '0' except ACO, which is not applicable (NA).

Attack synopsis: Each logical bit in the ACSR register plays different roles and functionalities in the ADC conversion logic, depending on the logical value (i.e., '0' or '1') set to it. For example, the analog comparator will be disabled if the logical bit ACD is set to '1', the analog comparator interruption will be enabled if the logical bit ACIE is set to '1', etc. A

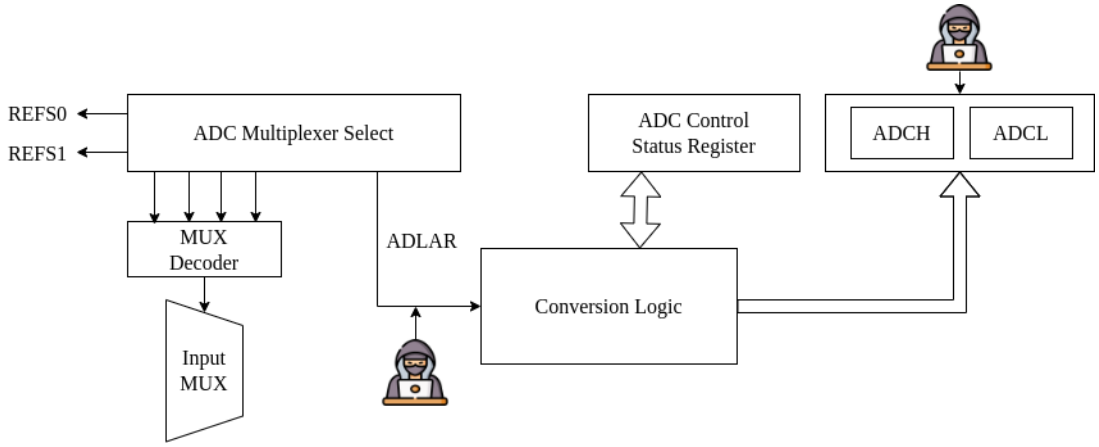

Fig. 10: Attacking the ADMUX register

```
==========================================
Deceiving the ADC conversion logic
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Actual output Temperature: 24.49°C
After the attack Temperature: 1588.13°C
Attack Start Time (milliseconds): 5207
Attack End Time (milliseconds): 5262
==========================================
```

Fig. 11: Output of Attack 1

detailed information regarding the roles and functionalities of the ACSR bits in the ADC conversion logic can be found in [36]. When we simultaneously set the ACD and ACIE bits to '1' in the ACSR register, the ADC conversion process will hang, hence leading to DoS attack. This will render system unavailability, which is a critical concern in time-sensitive systems, such as CPS. Our construction of *Attack 2* (i.e., DoS attack) in the ADC conversion logic is formally captured as follows:

$$DoS\_Attack := (ACD == 1) \wedge (ACIE == 1)$$

In our experimental setup, we perform this attack by injecting the code "*ACSR |= 0b10001000;*" into the Arduino firmware. Here, the 4^{th} bit (i.e., ACIE) and 8^{th} bit (i.e., ACD) of the ACSR register are set to '1', which causes the system to hang (refer the red box in Figure 13). The code snippet of the attack is also shown in Figure 13. Furthermore, a high-level illustration of this attack is depicted in Figure 12.

3) Resetting the ADC process (Attack 3): In this attack, we reset the ADC process by manipulating the ADC data registers, such as ADCH and ADCL. As discussed in the preceding sections, ADC has two 16-bit wide data registers, i.e., ADCH and ADCL. These registers are used to store the ADC digital output obtained from the analog conversion. For example, Table IV shows how our temperature sensor reading of 24.49℃is stored in the ADC data registers. The equation to translate the sensor reading temperature value to binary format and vice versa can be referred in [37].

Attack synopsis: Like the other ADC registers, the ADC data registers (i.e., ADCH and ADCL) can also be manipulated by an attacker. One way to manipulate these registers would be by clearing the ADC outcome data stored in them. However, we cannot directly do that since these registers are read-only. Meaning, we can only read the data stored in these registers, but not modifying it. So, how can we achieve the attack on the ADC data registers? We discuss details of our proposed attack technique as follows.

The ADC output data is read by the the "analogRead()" function – a function (often used in Arduino) that reads the digital value from a specified analog pin. However, there are some implicit tasks to be performed before reading the digital value. First, the analog value (e.g., the voltage between 0 and 5V) from the analog pin will be converted to a digital value between 0 to 1023 (for a 10-bit long ADC). As discussed in the preceding sections, this digital value (i.e., the ADC output) will be then stored in the ADCH and ADCL registers. Then, the "analogRead()" function defines two variables, say "low" and "high", to read the ADC output from the ADCL and ADCH data registers, respectively. That means, the "low" variable reads values from the ADCL register and the "high" variable reads values from the ADCH register. The final ADC output will be a combination of the two variables, i.e., $low = ADCL \;\&\&\; high = ADCH$. However, we can attack this logic by including a malicious script in the device's (the Arduino in our case) firmware, and particularly in the "analogRead()" function. Instead of assigning the ADCL and ADCH register values to the "low" and "high" variables mentioned above, we can maliciously assign '0' to both. That means, we inject the "$low = 0$;" and "$high = 0$;" codes to the source-code of the "analogRead()" function in the Arduino firmware. This might also be done through system configuration. This leads the ADC output to be always '0' instead of the actual result. We tested this attack on our temperature reading setup. Even though the actual temperature was 24.17℃, the temperature reading after launching the attack was always 0℃. The outcome of this attack is shown in Figure 14. Therefore, this attack can also mislead the control decision of PLCs in CPS. In a similar way, more critical and complex attacks can also be performed on the ADC data registers.

TABLE IV: The ADC output data presentation in data registers

ADCH								ADCL							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
G	G	G	G	G	G	0	0	0	1	0	0	1	1	0	0

TABLE V: ACSR register bits

ACSR Bits	ACD (Bit 7)	ACBG (Bit 6)	ACO (Bit 5)	ACI (Bit 4)	ACIE (Bit 3)	ACIC (Bit 2)	ACIS1 (Bit 1)	ACIS0 (Bit 0)
Read/ Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Values	0	0	NA	0	0	0	0	0

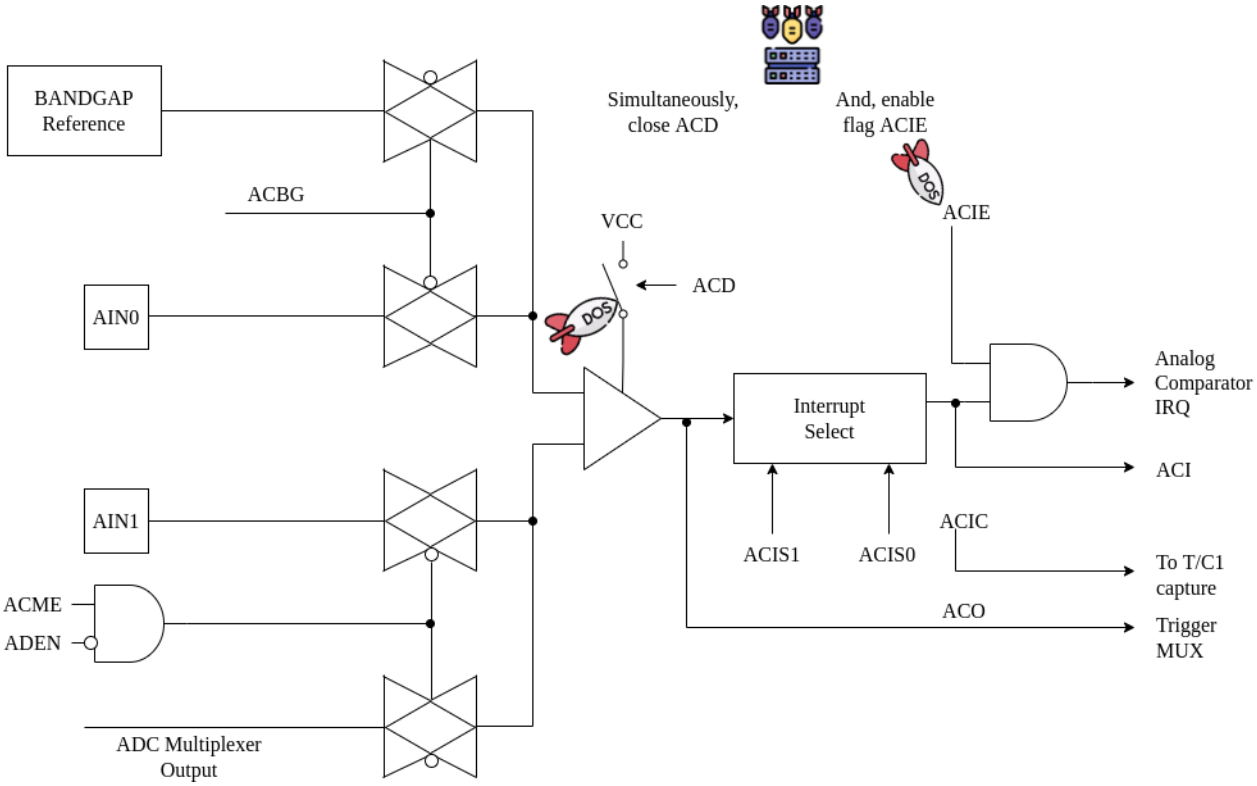

Fig. 12: DoS attack on ADC by manipulating the ACD and ACIE bits of the ACSR register

V. Experimental Design

In this section, we present details of our experimental setup designed to test the proposed attack techniques. Our experimental setup simulates a temperature-based alarm control system. In brief, the system periodically reads the surrounding temperature, and it triggers an alarm when the temperature value is above a threshold, say 30℃.

We design a mini-CPS that simulates a temperature-based alarm control system. The system uses analog temperature sensor, The controller (i.e., PLC) will then trigger a siren alarm when the temperature reading is above the threshold.

To simulate the above process, we design a mini-CPS testbed using IoT devices, sensors and actuators. Specifically, we use Arduino MEGA[6] as a soft PLC, which makes control decisions based on the temperature reading. We use an analog sensor, called LM35[7], to read the surrounding temperature. We use an 8Ω mini speaker[8] (a siren alarm) as an actuator, which activates the alarm when it receives an "ON" command from the PLC. A high-level schemata of the experimental setup is depicted in Figure 15. A more detailed discussion of the experimental setup is provided as follows.

As shown in Figure 15, the analog temperature sensor (LM35) is connected to the Arduino board (via the analog input A0) to read the surrounding temperature. The sensor is also connected to the internal voltage reference 3.3V. The Arduino board has 16 analog input pins and 54 digital input/output pins. It also contains an inbuilt ADC and MCU. The inbuilt

[6]https://store.arduino.cc/products/arduino-mega-2560-rev3

[7]https://www.electronicwings.com/sensors-modules/lm35-temperature-sensor

[8]https://circuit.rocks/mini-metal-speaker-w-wires-8-ohm-0-5w.html

```
202  void loop() {
203
204    Serial.println("\n=========================================");
205    ACSR |= 0b10001000;
206    int actualReading = analogRead(sensorPin);
207    int attackedConversion = attackedConversionReading(getPinByBoard(sensorPin));
208
209    Serial.print("Actual output ");
210    measureTemperature(actualReading);
211
212    int digiValueAttacked = map(attackedConversion, 0, 1023, 0, 3);
213    Serial.print("After the attack ");
214    //Serial.println(digiValueAttacked);
215
216    measureTemperature(attackedConversion);
217    digitalWrite(digiPin,digiValueAttacked);
218    //Serial.print("Read value from digiPin : ");
219    //Serial.println(digitalRead(digiPin));

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

> Executing task: C:\Users\rtram\.platformio\penv\Scripts\platformio.exe device monitor <

--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2file, nocontrol, printable, send_on_enter, time
--- More details at https://bit.ly/pio-monitor-filters
--- Miniterm on COM6  9600,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
```

Fig. 13: The code snippet and output of *Attack 2*

```
==========================================
Disabling conversion attack at ADC
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Actual output Temperature: 24.17°C
After the attack Temperature: 0.00°C
Attack Start Time (milliseconds): 10474
Attack End Time (milliseconds): 10529
==========================================
```

Fig. 14: Output of Attack 3

ADC (integrated in the same electrical circuit board with the MCU) converts the analog temperature values to a discrete-time digital values. The MCU acts as a PLC and makes control decisions, such as triggering the alarm, based on the digital temperature value obtained from the ADC. More specifically, it issues an "ON" or "OFF" control command depending on the the temperature value and the threshold set. The "ON" control command triggers the alarm while the "OFF" control command turns off the alarm. An 8Ω mini speaker (a siren alarm) is connected to the Arduino board to act as an actuator. It activates the alarm when it receives an "ON" command from the PLC, and it turns off the alarm otherwise.

Due to the lack of access, unfortunately, we did not conduct our experiments on real-world CPS testbeds with vendor-supplied PLCs. However, we believe that our experimental setup discussed above is sufficient to evaluate the effectiveness of the proposed attach techniques. Because,

1) Nowadays, Arduino boards are widely used both in experimental and production settings. For example, it is also widely used in most IoT systems. It is often used as a soft PLC in many ICS/CPS systems. Therefore, our Arduino-based experimental setup is substantially sufficient to evaluate the effectiveness of the proposed ADC attacks.
2) More importantly, the analog-to-digital conversion logic and software/hardware design of most ADCs are very similar. Hence, the ADC architecture (including its memory registers) of a real-world PLC and an Arduino-based soft PLC is also expected to be very similar. Therefore, it is highly likely that our proposed ADC attacks will also be effective when applied to real-world PLCs. Nevertheless, we still intend to extend our experiments on real-world CPS testbeds in the future.

VI. Evaluation and Discussion

In this section, we discuss a detailed evaluation of our proposed attacks. In brief, we evaluate the proposed attack techniques along three dimensions: 1) Accuracy 2) Efficiency 3) Impact. Furthermore, we discuss possible countermeasures to prevent such types of ADC attacks.

A. Attack Accuracy

There were no much significant internal or external factors that could influence our experimental results. The only sensible factor or variable is the temperature environment. Hence, we conduct the experiments in different temperature conditions, such as cold (< 16℃), mild (16℃ – 25℃) and hot (> 25℃). In all such circumstances, the proposed attacks always produced the expected results. Meaning, we have not observed any false positive or false negative results in all our experiments. Therefore, the proposed attacks are very accurate in achieving the intended goal.

B. Attack Efficiency

The proposed attack techniques are simple to be launched. The attacks are performed by systematically manipulating the flag or data values of the targeted ADC registers. At runtime, there was not any significant overhead observed, both in CPU

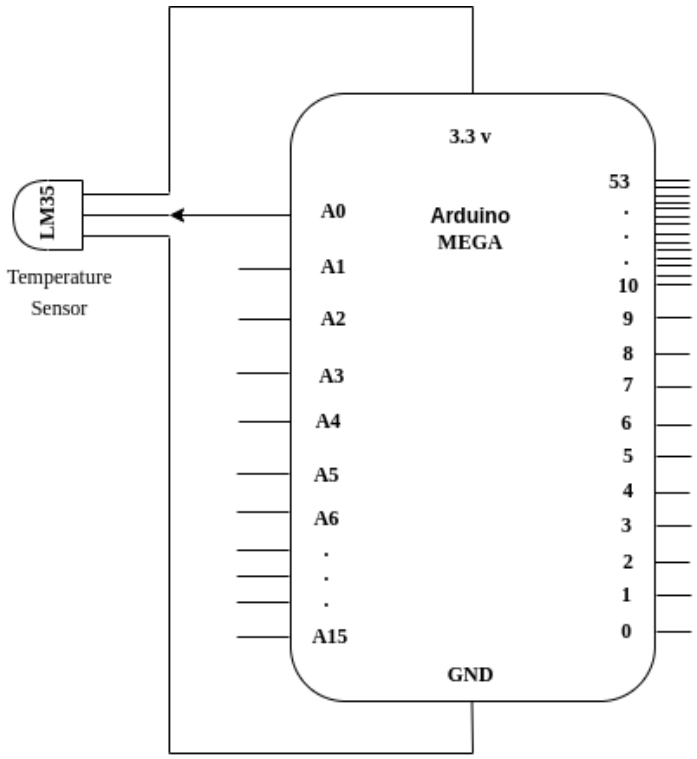

Fig. 15: Schematic diagram of the experimental setup

and memory usage. It takes only a few microseconds to conduct each of the three attacks. To experimentally show the execution time of each attack, we performed 50 simulations for each attack. The experimental results are depicted in Table VI. That means, the execution time of *Attack 1* and *Attack 3* are 60.2μ and 60.3μ, respectively. However, we could not measure the execution time of *Attack 2* since the system immediately hangs after this attack is performed. Therefore, outputs of the attacks are almost instantaneous. Meaning, impacts of the attacks can be reflected in real-time – without any significant delay.

TABLE VI: The average execution time of the attacks for 50 simulations

Attack 1	***Attack 2***	***Attack 3***
60.2µs	Not applicable since the system hangs after the attack	60.3µs

C. Attack Impact

ADCs are commonly used in a wide-range of critical systems, such as ICS, CPS, and IoT, among others. Hence, manipulating the ADC conversion logic may result in a catastrophic impact to the systems. For example, the ADC outcome (i.e., the converted digital value) is a crucial input to the PLC to make control decisions in CPS. If the ADC outcome is manipulated, it will mislead the PLC to make wrong control decisions. This will result in incorrectly controlling the physical process in CPS. Hence, the entire CPS system could be severely impacted, including destruction of the physical plant.

Although the proposed attacks are tested on an Arduino-based soft PLC, we believe that it can also be applied and tested on real-word CPS systems (please refer the discussion in Section V).

D. Proposed Countermeasures

As discussed in the preceding sections, attacking the analog-to-digital conversion process can result a catastrophic impact on various systems and infrastructures. In particular, manipulating the flags and data values of ADC registers is a critical stealthy attack that might not even be easily detected. Therefore, it is essential to design appropriate countermeasures against these attacks. In this work, we highlight possible countermeasures and research directions to overcome such security concerns in ADCs.

a) Enforcing write-protected policy to ADC registers: As discussed, the register manipulation attacks are carried out by overwriting the exiting data or flag values of certain critical registers in ADC, such as the ADC multiplexer selection register (ADMUX), the ADC analog comparator and status register (ACSR), and the ADC data registers (ADCL and ADCH), among others. One possible direction to address such attacks is by enforcing a "write-protected" policy to critical ADC registers and other memory locations. Such measures would help to prohibit an unauthorized writing or overwriting of ADC registers, hence preventing manipulation attacks in ADC registers.

b) Decoupling the ADC process into subroutines: The other mitigation approach we would like to propose is, decoupling the ADC conversion into a separate ADC subroutine program and to allow changes only to the subroutine. This

is because the main program will validate the subroutine before executing the code. The first validation will take place whenever there is change in the source-code of the ADC subroutine. The outputs are validated every time the subroutine is triggered to check the values are within the prescribed norms. This helps to prevent an unauthorized injection of malicious code in the PLC's firmware or control software, hence preventing attacks from manipulating ADC registers via malicious code injection.

c) Authorizing and tracking firmware updates: Properly authenticating and authorizing PLCs would be another approach to prevent ADC-based attacks. Only authorized users must be allowed to make edits to the firmware or control software of PLCs. A logging system should also be in place that tracks and traces all authorized and unauthorized activities, including providing input to the system. The track and trace system must clearly identify all software/firmware edits made or inputs provided to the system.

VII. Related work

In this section, we discuss the relevant prior works that are closely related to the security concerns of ADC. In particular, we discuss prior attacks performed on the ADC conversion logic.

As discussed in the introduction, ADCs have been targeted by various types of attackers. Attackers often target certain vulnerabilities that can be discovered in the hardware or software of ADCs. Bolshev et al. [8] has conducted an extensive study both on the hardware and software based vulnerabilities of ADCs. They then developed an attack technique by exploiting vulnerabilities in the sampling frequency and dynamic range of the ADC conversion logic. There are also side-channel attacks that exploited the strong correlation between the ADC digital output codes and the ADC supply current waveforms [9]. If the power side-channel attack (PSA) of the ADC is exploited, it can expose the private signal change data [11]. When applied to a successive approximation register (SAR) without PSA protection, the power supply current waveforms of the SAR are attacked. Other side-channel attacks have been also developed by exploiting various vulnerabilities in ADC [16], [17], [38], [18], [19], [20], [21], [22].

Other class of attacks have exploited fast attack automatic gain control (AGC) vulnerability in ADC to deceive the outcome of the analog to digital conversion [10], [11], [12]. Some other attacks exploited the DAC-to-DAC crosstalk vulnerability in the ADC conversion logic [13], [14], [15]. However, we are not aware of any existing attack techniques that exploit vulnerabilities related to ADC registers.

In CPS, an attacker who has access to the PLCs can generate a signal with a frequency that is interpreted as being valid by the ADC, when in reality it can cause serious damage to the physical process [12]. In spite of ADCs having anti-aliasing filter that restricts the bandwidth of a signal, these filters do not prevent frequency attacks. The attacks must be customized for targeted ADCs.

Another ADC-related attack involves manipulating the device's input and output (I/O) at a low level, which allows the attacker to control the PLC without triggering any alarms [38].

System-on-Chip (SoC) integrators may design a Hardware Trojan with the intention of perturbing the ADC from malfunctioning by manipulating input or output signals or by affecting the modulator's output bit [15]. By applying the wrong version of the input signal instead of original version of input signal in modulator sometimes, change to one third of the healthy signal amplitude. The output signal also can be manipulated through noise signal making it through inverted, attenuated or by doing both. A maliciously sized static random-access memory (SRAM) cell can be controlled locally or globally to supply inversed output bits from the modulator to the digital filter block at any time by controlling the supply voltage of its cell. Also, the output signal can be manipulated by triggering capacitance by including trojan circuit to change the structure and size of the components based on the attacker restrictions. Another stealthy hardware trojan attack was also recently launched on the analog integrated circuits (ICs) of ADCs [26]. However, all these attacks did not specifically target the ADC registers.

Memory corruption attacks are another common threats against PLCs of industrial control or cyber-physical systems. They typically exploit memory-safety vulnerabilities, such as buffer overflows and dangling pointers, that could be found in the firmware of PLCs to corrupt the process memory or execution flow of programs at runtime [23], [24], [25]. However, these attacks target the runtime process memory of PLCs, not specifically the ADC memory registers.

In summary, there are several types of ADC-related attacks presented in the literature. To the best of our knowledge, non of them specifically target the ADC registers. In this work, we rather identify and exploit certain ADC registers used in the analog-to-digital conversion process, which appear to be the unexplored attack surfaces in ADC.

VIII. Conclusion

ADCs are integral components in most critical systems, such as IoT and control systems. However, ADCs have been targeted by a wide range of physical or cyber attacks. The attackers may exploit various types of vulnerabilities that could be found in the software or hardware of ADCs. In this work, we first conducted a more in-depth study of the ADC conversion logic to investigate ADC vulnerabilities that were not well explored by previous work. Consequently, we managed to find relevant vulnerabilities related to the registers used in the ADC conversion logic. To demonstrate its exploitability, we developed three types of ADC attacks and tested it in an IoT-based mini-CPS environment.

By manipulating the ADC registers, we showed that it is possible to deceive the ADC outcome or maliciously halt the analog-to-digital conversion process. The ADC process can be forced to return an output that is much different from the expected result. This is carried out by changing the flag in the ADC multiplexer selection register, called ADMUX. An attack can also be carried out by manipulating the analog comparator control and status registers, called ACSR. We managed to maliciously hang the ADC conversion process by simultaneously enabling the ACD (analog comparator disable)

and ACIE (analog comparator input enable) bits of the ACSR register, which resulted in system unavailability. We also showed that the ADC conversion process can be rendered useless by setting its output values to zero. This is achieved by resetting the data reader when it reads the ADC output from the ADCH and ADCL data registers. This was an attempt to show that ADC registers can most definitely be manipulated if no underlying protection mechanism is set for the ADC conversion process.

In the future, we plan to extend our experiments on real-world CPS testbeds using vendor-supplied PLCs. We also intend to conduct additional research to further explore key ADC vulnerabilities. Proposing and developing appropriate countermeasures for register-based ADC attacks is also left as a future work.

REFERENCES

[1] D. K. Mynbaev and L. L. Scheiner, *Analog Signals and Analog Transmission*, 2020, pp. 103–201.

[2] B. Le, T. Rondeau, J. Reed, and C. Bostian, "Analog-to-digital converters," *IEEE Signal Processing Magazine*, vol. 22, no. 6, pp. 69–77, 2005.

[3] T. Satoh, K. Takahashi, H. Matsui, K. Itoh, and T. Konishi, "10-gs/s 5-bit real-time optical quantization for photonic analog-to-digital conversion," *IEEE Photonics Technology Letters*, vol. 24, no. 10, pp. 830–832, 2012.

[4] M. Lab, "Analog to digital converter – how adc works and types?" 2017, online, available at https://microcontrollerslab.com/analog-to-digital-adc-converter-working/.

[5] E. R. Alphonsus and M. O. Abdullah, "A review on the applications of programmable logic controllers (plcs)," *Renewable and Sustainable Energy Reviews*, vol. 60, pp. 1185–1205, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032116000551

[6] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363–369.

[7] S. Zanero, "Cyber-physical systems," *Computer*, vol. 50, no. 4, pp. 14–16, 2017.

[8] A. Bolshev, J. Larsen, M. Krotofil, and R. Wightman, "A rising tide: Design exploits in industrial control systems," in *10th USENIX Workshop on Offensive Technologies (WOOT 16)*. Austin, TX: USENIX Association, Aug. 2016. [Online]. Available: https://www.usenix.org/conference/woot16/workshop-program/presentation/bolshev

[9] T. Jeong, A. P. Chandrakasan, and H.-S. Lee, "S2adc: A 12-bit, 1.25ms/s secure sar adc with power side-channel attack resistance," in *2020 IEEE Custom Integrated Circuits Conference (CICC)*, 2020, pp. 1–4.

[10] analog.com, "Ad9364 register map reference manual," 2021, white Paper, available at https://www.analog.com/media/cn/technical-documentation/user-guides/ad9364_register_map_reference_manual_ug-672.pdf.

[11] T. Jeong, "Secure analog-to-digital conversion against power side-channel attack," May 2020, online, available at https://dspace.mit.edu/handle/1721.1/127018.

[12] E. Kovacs, "Adc attacks can cause damage in industrial environments," Nov 2016, online, available at https://www.securityweek.com/adc-attacks-can-cause-damage-industrial-environments.

[13] docs.rs online.com, "8-channel, 12-bit, configurable adc/dac with on-chip reference, i2c interface," 2014, online, available at https://docs.rs-online.com/1e6a/0900766b813daba4.pdf.

[14] R. Langmann and M. Stiller, "The plc as a smart service in industry 4.0 production systems," *Applied Sciences*, vol. 9, no. 18, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/18/3815

[15] S. Taheri and J.-S. Yuan, "Mixed-signal hardware security: Attacks and countermeasures for $\delta\sum$ adc," *Electronics*, vol. 6, no. 3, 2017. [Online]. Available: https://www.mdpi.com/2079-9292/6/3/60

[16] T. Miki, N. Miura, H. Sonoda, K. Mizuta, and M. Nagata, "A random interrupt dithering sar technique for secure adc against reference-charge side-channel attack," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 1, pp. 14–18, 2020.

[17] T. Miki and M. Nagata, "Countermeasures against physical security attacks on ICs utilizing on-chip wideband ADCs," *Japanese Journal of Applied Physics*, vol. 61, no. SC, p. SC0803, Mar 2022. [Online]. Available: https://doi.org/10.35848/1347-4065/ac4823

[18] R. Munny and J. Hu, "Power side-channel attack detection through battery impedance monitoring," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[19] N. Gattu, M. N. Imtiaz Khan, A. De, and S. Ghosh, "Power side channel attack analysis and detection," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–7.

[20] M. Ashok, E. V. Levine, and A. P. Chandrakasan, "Randomized switching sar (rs-sar) adc protections for power and electromagnetic side channel security," in *2022 IEEE Custom Integrated Circuits Conference (CICC)*, 2022, pp. 1–2.

[21] R. Chen, H. Wang, A. Chandrakasan, and H.-S. Lee, "Ram-sar: A low energy and area overhead, 11.3fj/conv.-step 12b 25ms/s secure random-mapping sar adc with power and em side-channel attack resilience," in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2022, pp. 94–95.

[22] D. R. E. Gnad, J. Krautter, and M. B. Tahoori, "Leaky noise: New side-channel attack vectors in mixed-signal iot devices," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 3, p. 305–339, May 2019. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8297

[23] Y. Geng, Y. Chen, R. Ma, Q. Wei, J. Pan, J. Wang, P. Cheng, and Q. Wang, "Defending cyber-physical systems through reverse engineering based memory sanity check," *IEEE Internet of Things Journal*, pp. 1–1, 2022.

[24] E. G. Chekole, S. Chattopadhyay, M. Ochoa, H. Guo, and U. Cheramangalath, "Cima: Compiler-enforced resilience against memory safety attacks in cyber-physical systems," *Computers & Security*, vol. 94, p. 101832, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820301061

[25] E. G. Chekole, S. Chattopadhyay, M. Ochoa, and G. Huaqun, "Enforcing full-stack memory safety in cyber-physical systems," in *Proceedings of the International Symposium on Engineering Secure Software and Systems (ESSoS'18)*, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-94496-8_2

[26] M. Elshamy, G. Di Natale, A. Pavlidis, M.-M. Louërat, and H.-G. Stratigopoulos, "Hardware trojan attacks in analog/mixed-signal ics via the test access mechanism," in *2020 IEEE European Test Symposium (ETS)*, 2020, pp. 1–6.

[27] S. Taheri, J. Lin, and J.-S. Yuan, "Security interrogation and defense for sar analog to digital converter," *Electronics*, vol. 6, no. 2, 2017. [Online]. Available: https://www.mdpi.com/2079-9292/6/2/48

[28] T. Wadatsumi, T. Miki, and M. Nagata, "A dual-mode successive approximation register analog to digital converter to detect malicious off-chip power noise measurement attacks," *Japanese Journal of Applied Physics*, vol. 60, no. SB, p. SBBL03, feb 2021. [Online]. Available: https://doi.org/10.35848/1347-4065/abde26

[29] G. Prathiba, M. Santhi, and A. Ahilan, "Design and implementation of reliable flash adc for microwave applications," *Microelectronics Reliability*, vol. 88-90, pp. 91–97, 2018, 29th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis (ESREF 2018). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026271418306577

[30] A. Grami, "Chapter 5 - analog-to-digital conversion," in *Introduction to Digital Communications*, A. Grami, Ed. Boston: Academic Press, 2016, pp. 217–264. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124076822000053

[31] "Analogue to digital converter," Dec 2020, online, available at https://www.electronics-tutorials.ws/combination/analogue-to-digital-converter.html.

[32] P. Li, X. Yi, X. Liu, D. Zhao, Y. Zhao, and Y. Wang, "All-optical analog comparator," *Scientific Reports*, vol. 6, 08 2016.

[33] E. G. Chekole, J. H. Castellanos, M. Ochoa, and D. K. Y. Yau, "Enforcing memory safety in cyber-physical systems," in *Katsikas S. et al. (eds) Computer Security. SECPRE 2017, CyberICPS 2017*, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-72817-9_9

[34] E. G. Chekole and G. Huaqun, "Ics-sea: Formally modeling the conflicting design constraints in ics," in *Proceedings of the Fifth Annual Industrial Control System Security (ICSS) Workshop*, ser. ICSS. New York, NY, USA: Association for Computing Machinery, 2019, p. 60–69. [Online]. Available: https://doi.org/10.1145/3372318.3372325

[35] R. R. Jogdand, P. K. Dakhole, and P. Palsodkar, "Low power flash adc using multiplexer based encoder," in *2017 International Conference*

on Innovations in Information, Embedded and Communication Systems (ICIIECS), 2017, pp. 1–5.

[36] M. Mitescu and I. Susnea, "Interfacing to analog signals," *Microcontrollers in Practice*, pp. 93–106, 2005.

[37] adafruit.com/, "Using a temp sensor," 2022, online, available at https://learn.adafruit.com/tmp36-temperature-sensor/using-a-temp-sensor.

[38] E. Kovacs, "Plcs vulnerable to stealthy pin control attacks," Nov 2016, online, available at https://www.securityweek.com/plcs-vulnerable-stealthy-pin-control-attacks.