# SensPnP: Seamless Integration of Heterogeneous Sensors with IoT devices

Sanku Kumar Roy, *Student Member, IEEE,* Sudip Misra, *Senior Member, IEEE,* and Narendra Singh Raghuwanshi

***Abstract*—The increasing growth of Internet of Things (IoT) applications induces the need for easy integration of third-party peripherals, which is restricted in present IoT devices. In this paper, we present a novel plug-and-play (PnP) solution for the above stated problem. The proposed PnP solution, named SensPnP, is the combination of embedded hardware and firmware that has the capability of integrating third-party embedded sensors with the IoT devices without any prior information about the sensors and the Internet. We present an architecture of a PnP-enabled IoT device, which supports heterogeneous embedded peripheral communication protocols. A novel embedded protocol detection approach for enabling seamless integration of sensor with IoT device is proposed. To enable sensor communication, we propose an algorithm for automatic driver management of the connected sensor with an IoT device. This is achieved through a low-cost and low power switching and identification hardware with a light-weight identification and driver management firmware stack. To show the effectiveness of the proposed solution, we practically implemented a prototype in a real test-bed. The analysis of the PnP time, protocol identification time, the memory footprint, lifetime, and overall cost of the proposed PnP solution under different operating conditions establishes superiority of its circuitry and firmware. Experimental results show that SensPnP requires minimal memory footprint, reduced energy consumption, and reduced PnP time compared to the existing solutions. Additionally, the overall cost analysis shows that the cost of SensPnP is noticeably less compared to existing solutions. Thus, the cost-efficiency of SensPnP makes it more acceptable to the consumers when compared with existing solutions.**

***Index Terms*—Internet of Things (IoT), IoT device, Heterogeneous, Plug-and-Play, Device Driver, Protocol Identification.**

## I. Introduction

INTERNET OF THINGS (IoT) encompasses a dynamic global infrastructure, in which physical objects interact with one another, and share their information [1]–[3]. The objects are physical and virtual such as smartphone, vehicle, logistic gadget, home appliance, healthcare gadget, epaper, and ebook. A physical object which is embedded with sensors, actuators, and network connection, is called an IoT device, and is shown in Fig. 1 [4]. IoT devices are constrained in terms of processing capabilities and memory usage and must operate for long periods of time on a tight energy budget. In the physical context, there are a variety of devices which are heterogeneous in terms of hardware design, protocol, specification, semantics and syntax [3], [5]. Also, the integrated peripherals such as sensors, actuators, and radios are heterogeneous in terms of communication protocol, hardware design, and driver.

S. K. Roy and S. Misra are with the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, 721302, India, Email: sankukumarroy@gmail.com and smisra@sit.iitkgp.ac.in.

N S Raghuwanshi is with the Department of Agricultural and Food Engineering, Indian Institute of Technology, Kharagpur, 721302, India, Email: nsr@agfe.iitkgp.ac.in.

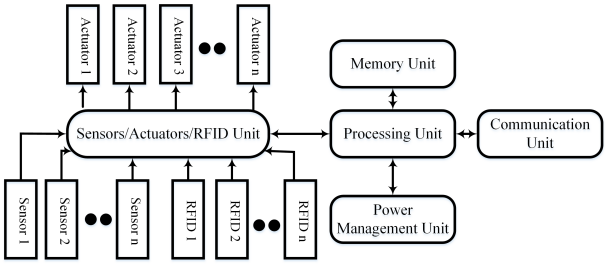



Fig. 1. Block diagram of IoT device

IoT is being applied in real-time, large-scale, and diverse application scenarios which urges the integration of various third-party peripherals with the device. However, peripherals are traditionally manufactured as proprietary, and the integration of peripherals with IoT devices is typically vendor-specific. Addressing PnP integration of heterogeneous peripherals with low-power and low-processing device is a challenging issue in IoT. The complexity of heterogeneous peripheral communication with processor is additionally challenging. The problem is complex — its solution requires addressing different issues concerning plugin, identification, driver management, and data communication.

### A. Motivation

The problem of PnP integration of peripherals in mainstream computing systems has been addressed through standard hardware interfacing protocols such as Universal Serial Bus (USB) [6]. USB [6] provides standard hardware connector, high-speed packet-based communication, auto-configuration, and identification of peripheral devices. However, the approach is inefficient in terms of energy consumption, computation, and memory usage for the resource-constrained IoT devices, as the approach focuses on the performance of peripherals for the mainstream computing system. Moreover, specific and customized chips are required for all peripherals, which increases the cost and energy consumption of every IoT device and effects the implementation of large-scale IoT network of devices. However, there are a number of standard hardware protocols for peripheral interconnection, including Analog to Digital Converter (ADC), Serial Peripheral Interface (SPI) [7], Inter-Integrated Circuit (I2C) [8], and Universal Asynchronous Receiver-Transmitter (UART) [9]. However, the approach does not support PnP device integration and the device type identification lacks the property.

To address the above problem, Yang *et al.* [10] proposed $\mu$PnP solution, where they discussed the solution of peripheral identification, device driver management, and device remote discovery. The $\mu$PnP supports existing peripheral interconnection standard protocols, including ADC, I2C [8], SPI [7], and UART [9]. Using multivibrator circuit, 32 bit unique identifier (UID) for each peripheral is generated, which maps to the global $\mu$PnP address space. Corresponding to the UID, the manufacturer uploads the specification and driver of the peripheral. When the peripheral is plugged to the device, it generates a 32 bit UID, using which the corresponding driver is downloaded from their server for starting the peripheral communication with the IoT device. However, the approach is a vendor platform specific solution and is not generalized for the global IoT network. In addition, the process of driver downloading and installation delays the peripheral communication, and also consumes more power, which effects the life-time of the device. Further, there is a requirement of high bandwidth to download the driver from the server, which may not be feasible for resource-constrained IoT networks.

However, the issue related to seamless integration of peripheral with IoT device is crucial in building a global IoT network of heterogeneous sensors and actuators, which is yet to be adequately addressed.

### B. Contribution

To address the above-discussed issues, we propose a genuine PnP solution for the integration of an third-party embedded peripheral with an IoT device. In brief, the *contributions* in this manuscript are as follows.

- We propose a novel PnP solution for integration of an embedded peripheral with an IoT device, which is a combination of embedded hardware and firmware.
- We present the design of a PnP-enabled IoT device.
- We propose a novel embedded protocol detection technique for enabling seamless integration of peripherals with IoT devices.
- To enable PnP, we propose an algorithm for automatic driver management of an embedded peripheral.
- To show the effectiveness of the proposed solution, we practically implemented the PnP-enabled IoT device to evaluate different parameters such as PnP time, identification time, memory footprint, lifetime, and overall cost.

The remaining part of this paper is organized as follows. Section II discusses the current state-of-the-art of peripheral interfacing solutions in terms of hardware and software, while presenting their limitations. Section III presents the problem scenario and the proposed PnP architecture. Section IV describes the compatible peripheral protocol identification technique. The driver management of the connected peripheral is presented in the Section V, whereas, in Section VI, we present the implementation of SensPnP. Experimental setup and results of the designed system are shown in Section VII. Finally, we conclude the paper and discuss future research directions of the work in Section VIII.

## II. Related Works

This section summarizes the state-of-the-art existing works on the seamless integration of heterogeneous peripherals in IoT. The problem of seamless integration in IoT is categorized into two parts — a) seamless integration with IoT network and b) seamless integration of peripherals with low power and low processing computing devices. The seamless integration with IoT network endorses different programming platforms for accessing and connecting heterogeneous IoT peripheral devices with cyber-physical world [11]. On the other hand, with the emergence of various vendor-specific peripherals, there is a need of seamless integration of these peripheral with the computing devices.

To enable seamless integration with IoT network, researchers proposed different IoT middlewares based on three architectures, i.e., service-oriented, cloud-based, and actor model [11]. In the service-oriented architecture [12], [13], the middleware is augmented with cloud platform to perform seamless integration between IoT devices and the application. A model-driven development (MDD) approach based middleware is proposed in [14] to provide seamless integration between sensing devices and application layer. However, this architecture is not suitable for resource constrained device-to-device communication. In the cloud-based architecture [11], the cloud-based application programming interface (API) acts as middleware to provide seamless communication between IoT peripherals and the application layer. But the IoT peripherals can only be controlled or accessed using cloud supported or vendor provided APIs. The most popular IoT middleware is the actor model architecture, due to its light-weight framework and distributed nature [15]. The actor-based middleware is embedded in all the layers of IoT architecture (sensing/actuating, cloud, and application), which acts as a translator for seamless communication between heterogeneous devices of different layers. The presence of actors host/translators in different layers provide advantages of scalability, latency, and maintainability. In [15], authors proposed a hybrid framework-based of middleware on the actor model and flow based computing. It provides an unified programming model and light-weight running API to enable seamless communication between IoT peripherals and applications.

On the other hand, Sriskanthan *et al.* [16] proposed a protocol for data transfer through Bluetooth in the home network. The protocol supports PnP to connect home appliances in the Bluetooth-based home network. Similarly, Baek *el al.* [17] designed an universal PnP (UPnP) bridge to support communication two different UPnP networks through non-IP channels. Likewise, a software bridge between Bluetooth compatible devices and UPnP networks using a virtual agent is proposed by Jo *et al.* [18]. The software bridge enables communication between non-IP Bluetooth devices and IP based UPnP devices. Therefore, the user can access their Bluetooth devices data on a UPnP network.

The problem of PnP peripheral device integration for mainstream computing device was addressed by USB [6]. USB [6] defines a standard connector and communication protocol for connection and communication between peripheral and

computer. On the other hand, the interconnection technology provides peripheral type identification, auto-configuration, and high-speed communication with a computer. Also, the solution provides a mechanism for peripheral discovery in the embedded operating system (OS) and automatically install the necessary driver for the peripheral. Further, USB [6] supplies more details about the peripheral such as type, manufacturer's name, and manufacturing date. These information are manually stored in the special registers of USB during manufacturing process. This raises extra complexity and cost for manufacturers. Moreover, even a low power USB host chip consumes too much power that is inefficient for resource-constrained IoT network. The USB standard mainly focuses on the communication performance between peripheral and computer, rather than minimizing energy consumption, memory usage, and CPU utilization. However, the process of automatic peripheral configuration is very essential for IoT systems, as a single IoT network consists of thousands of heterogeneous peripherals.

On the other hand, the peripheral interfacing protocols for microcontroller such as UART [9], I2C [8], and SPI [7] target to minimize energy consumption, memory usage, and CPU requirement. SPI and I2C bus allow multiple peripherals to be connected to a single bus, while UART protocol allows only single peripheral. SPI, developed by Motorola, is a synchronous serial communication peripheral interfacing protocol for short range distance, and follows master-slave architecture. On the other hand, I2C compatible peripherals communicate with the bus using 8 bits unique address. UART is asynchronous serial point-to-point protocol using dedicated transmitting and receiving channel. To support these protocols, manufacturers write a manual driver for each peripheral. They neither provide the peripheral type identification nor allow the embedded OS to auto-configure providing the necessary driver of the connected peripheral.

To address the above-mentioned problem, Yang *et al.* [10] proposed $\mu$PnP for peripheral interfacing in the IoT, where they discussed peripheral identification, driver management, and remote discovery. $\mu$PnP supports existing peripheral interconnection standard protocols including ADC, I2C [8], SPI [7], and UART [9]. The authors used multivibrator circuit to generate a time pulse converting into a unique 32-bit address for each peripheral, where the value of passive components such as resistors and capacitors is defined by their online tool and the unique address is mapped with the global $\mu$PnP address space. During peripheral registration, manufacturers upload the details of manufacturer and the corresponding driver, which is downloaded from the server using the Internet when the peripheral is connected to the $\mu$PnP controller board to start the PnP peripheral communication. The process always depends on the Internet connectivity which effects the non-IP communication protocol compatible IoT device to connect with global IoT network. On the other hand, the approach is a vendor platform specific solution and is not generalized for the global IoT network. The process of driver downloading and installation delays peripheral communication, which is also inefficient due to increased power consumption. Besides, there is a requirement of high bandwidth to download the driver from the server, which may not be feasible for resource-constrained IoT networks.

Sakamoto *et al.* [19] proposed a dynamic device connection method between Web apps and peripheral using Web driver to support various kinds of devices with multiple OS for smartphones in the IoT scenario. Further, they demonstrated the feasibility of the proposed method by implementing it in the Android OS. In their solution, nearest peripherals are wirelessly controlled by a smartphone. However, the proposed scheme is limited to smartphone, where sufficient memory, CPU, and energy are available along with the Internet.

*Synthesis*: Critical analysis of the existing works unfold the existence of a research gap in PnP peripheral interfacing with IoT device to build a global IoT network. Some of the existing works [12]–[15] focus on seamless integration of heterogeneous peripherals between IoT devices and application layer, which are not suitable for the seamless integration of third-party embedded sensors with IoT devices. On the other hand, some of the existing works [16]–[18] concentrate on seamless communication among devices, which belong to heterogeneous networks, but these are not applicable for seamless peripheral integration. The existing work [6] primarily focuses on mainstream computing system, which is inefficient in the scenario of IoT network. Some of them [10], [19] tried to solve the problem for IoT, but the solutions are Internet-dependent and vendor-specific, and are hence not general solutions for the global IoT network of billion of peripherals. In this paper, we present a novel PnP solution for the integration of third-party embedded sensors with IoT devices without any prior information about the sensors and the Internet. The proposed PnP solution is the combination of embedded hardware and firmware.

## III. System Model

### A. Problem Scenario

The integration and configuration of a peripheral with a processor is practically a big challenge to enable PnP in the IoT device, which is highlighted in Fig. 2. Available heterogeneous sensors are mainly compatible with the embedded interfacing protocols such as SPI, I2C, UART, ADC, and Digital. These protocols focus on the interfacing of peripherals with low-power, low-processing, and low-memory usage embedded computing processors such as microcontroller, Field-Programmable Gate Array (FPGA), and Digital Signal processing (DSP). But these protocols do not support PnP communication between the peripheral and the embedded processor.

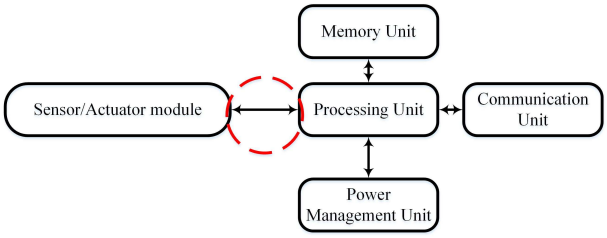


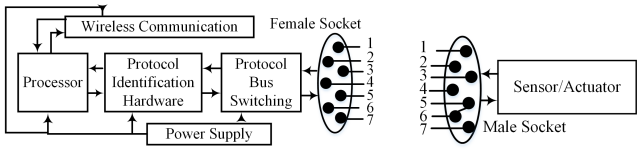

Fig. 2. Problem Scenario

The reason behind the above stated problem is the different hardware and software requirement of each protocol. As an example, SPI bus specifies four logical signals, i.e., Master Output Slave Input (MOSI), Master Input Slave Output (MISO), Serial Clock (SCK), and Slave Select (SS) whereas I2C bus requires two logical signals, i.e., Serial Data Line (SDA) and Serial Clock Line (SCL). Besides, the complex configuration of heterogeneous peripherals is also challenging, and in terms of software, each protocol has different configuration logic. Also, different peripherals with the same protocol or similar peripherals with the same protocol but different manufacturers have different configuration drivers, which make the integration of the peripherals with IoT device more complex. Therefore, the integration procedure of each peripheral is manually configured, which makes the configuration design vendor-specific.

From a consumer point of view, it is non-trivial to configure and start the communication between the peripheral and the processor [20]. To start and configure the communication, consumers should have the knowledge of low-level embedded coding and hardware circuitry. In addition, consumers may want to connect different sensors at different time instants in the same IoT device. For example, at one instant a consumer may want to monitor room temperature, but in later instant, his/her mind may change for monitoring light luminosity. But both of these cases of configuration and connection of different sensors in the same IoT device at different time instants are unfavorable from the consumer's perspective, which affects the implementation of the global IoT network.

### B. Proposed Architecture

In this section, we present the proposed architecture of SensPnP, while describing the different components, as shown in Fig. 3. The proposed architecture of SensPnP is divided into two parts — a) Universal plug-and-play (UniPnP) controller and b) Plug-and-play (PnP) peripheral module. A detailed description of each constituent part is presented below.




(a) UniPnP controller (b) PnP peripheral module

Fig. 3. Proposed architecture of SensPnP

*1) UniPnP Controller:* The UniPnP controller consists of different hardware units such as protocol bus switching unit, protocol identification hardware unit, processing unit, wireless communication unit, and power supply unit, as shown in Fig. 3(a). We separate the sensor/actuator unit from the UniPnP controller to enable PnP. In this paper, we mainly focus on the integration of peripherals with an IoT device, apart from wireless communication and power supply. When a peripheral is connected to the UniPnP controller, the processor gets an interrupt signal from the PnP peripheral module. Thereafter, the processor identifies the compatible protocol of the connected peripheral using both the protocol-bus-switching and protocol identifier units. Once the processor detects the supporting protocol of the connected peripheral, the processor starts searching for the corresponding driver of the peripheral for configuration and initialization. At the end, the processor starts data collection from the peripheral and sends the data to the neighbor IoT devices through wireless medium.

TABLE I
DIFFERENT PROTOCOL BUS MAPPING TO PNP CONNECTOR

| Pin No | UART | I2C | ADC | Digital | SPI |
|---|---|---|---|---|---|
| 1 | RX | SDA | NC | NC | SCK |
| 2 | TX | SCL | NC | NC | MISO |
| 3 | NC | NC | ADC | NC | MOSI |
| 4 | NC | NC | NC | Digital | SS |
| 5 | INT | INT | INT | INT | INT |
| 6 | +VCC | +VCC | +VCC | +VCC | +VCC |
| 7 | GND | GND | GND | GND | GND |

*2) PnP Peripheral Module:* The architecture of the PnP peripheral module is shown in Fig. 3(b). We use a connector having 7 pins to connect the peripheral to the PnP controller. Pins 1-4 of the connector are assigned for data communication between the processor of the PnP controller and the peripheral. Pin 5 is used to generate an interrupt signal. On the other hand, Pins 6 and 7 are assigned for power supply, i.e. $V_{cc}$ and ground (GND), respectively. Pins 5 and 6 are directly connected to generate the rising edge of the interrupt signal. Table I shows how the PnP connector maps to each embedded peripheral communication protocol bus, where NC refers to a not connected pin.

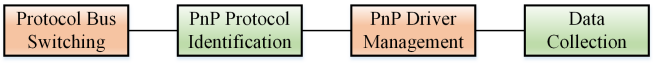



Fig. 4. Software stack of PnP-enabled IoT device

In our design, we propose hierarchical software layers stack to serve the different functionalities of PnP-enabled IoT device, as shown in Fig. 4. The bottom most layer of the stack, protocol bus selection, selects physical path for different embedded interconnects using multiplexer. On the other hand, PnP protocol identification software layer identifies the compatible protocol of the connected peripheral. Similarly, PnP driver management software layer manages the driver of the connected peripheral to collect sensor data.

## IV. PNP HARDWARE PROTOCOL IDENTIFICATION

### A. Protocol Bus Switching

SensPnP supports common microcontroller peripheral protocols such as SPI, I2C, UART, ADC, and Digital. Each protocol requires different number of logical signal pins. If we use dedicated physical path for each peripheral protocol, the number of pins of a connector will be increased. To minimize and optimize the number of pins of the connector, we propose a multiplexer-based protocol bus switching hardware, as shown in Fig. 5. The procedure of selecting each protocol is described in Table II, where 'X' means 'Don't Care'. In the proposed circuit, a multiplexer (MUX) starts its operation when the enable pin of the MUX is in 'Active Low' condition.

TABLE II
TRUTH TABLE OF PROTOCOL BUS SELECTION

| Inputs | | | | | | | Output |
|---|---|---|---|---|---|---|---|
| **D0** | **D1** | **D2 (E0)** | **D5** | **D6 (E2)** | **D7** | **D8 (E3)** | **Protocol Bus** |
| 0 | 0 | 0 | X | 1 | X | 1 | UART |
| 0 | 1 | 0 | X | 1 | X | 1 | I2C |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | SPI |
| X | X | 1 | 0 | 0 | X | 1 | ADC |
| X | X | 1 | X | 1 | 0 | 0 | Digital |

### *B. Protocol Identification*

The protocol identification of a peripheral is a big issue to enable seamless integration with the IoT device, which does not have any prior information about the peripheral such as compatible protocol, unique identification number/address, data format, and device name. We are the first to propose a novel protocol identification technique for a peripheral without having any prior information, which enables PnP, so that a consumer is able to connect any peripheral to a IoT device without any knowledge of low-level embedded coding and hardware circuitry.

When a peripheral is connected to UniPnP controller, it generates an interrupt signal, INR and sets the value of the interrupt flag ($I_F$) to 'True'. Accordingly, the processor starts protocol bus selection with the help of the proposed protocol bus switching hardware circuitry, as shown in Fig. 5. After the selection of a protocol bus, the processor identifies the compatible protocol of the connected peripheral. The identification technique of each protocol is discussed below.

*1) UART Identification:* The UART protocol consists of two pins, i.e. TX and RX. Fig. 5 shows that the RX pin of the processor is connected to the TX pin of the peripheral through the connector and vice versa. The simple UART data transmission technique helps to identify the compatibility of the connected peripheral with the UART protocol. When the UART bus is not transmitting data, it normally holds high voltage level. To start the transfer of data, the transmitting UART pulls the transmission line from high to low for one clock cycle. Before transmitting data, holding high level voltage at UART bus helps to identify the compatibility of the connected peripheral with UART. The detailed identification procedure for UART protocol is described in Algorithm 1. The first step of Algorithm 1 is used to select UART bus using Table II. Then, the TX and RX pins need to be configured as inputs and the initial value of both the pins is logically set as low (0 volt) . After that, Algorithm 1 checks the digital logic of both the pins. If TX and RX pins hold high (VCC) logics, which indicate that the connected peripheral is compatible with UART protocol.

*2) I2C Identification:* The I2C bus follows a master-slave architecture for communication between a processor and a peripheral and uses two wires: SDA and SCL. In SensPnP, the processor always works as a master and the peripheral works as a slave, while considering 7 bit unique address of each slave device. Therefore, the address of the connected peripheral should be between 0 to 127. All the slave devices follow the standard I2C bus protocol for communication. If the peripheral as a slave replies with an acknowledgment

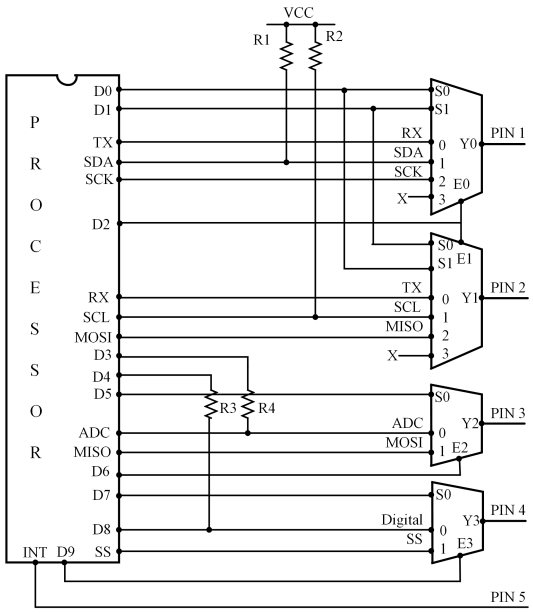



Fig. 5. Protocol bus switching and identification hardware

**Algorithm 1** UART interconnect identification

**INPUT:**
PnP peripheral module
**OUTPUT:**
$\mathcal{U}_{det}$ ▷ UART protocol identification
**PROCEDURE:**
Select UART bus using Table II
Configure TX and RX pins as inputs;
Initial value of TX and RX ← LOW; ▷ Initial condition
Sense TX and RX; ▷ Sensing values after interrupt
**if** TX = HIGH and RX = HIGH **then** ▷ Digital logic values
    $\mathcal{U}_{det}$ = True; ▷ connected to UART bus
**else**
    $\mathcal{U}_{det}$ = False; ▷ Not connected to UART bus

message corresponding to the handshaking message sent by the master device, the peripheral is connected to the I2C bus and is compatible with the I2C protocol, as discussed in Algorithm 2. If the sending address matches the peripheral's own address, the peripheral replies with an acknowledgment, which identifies the address of the connected peripheral with the compatible protocol. The detailed identification procedure for I2C protocol is described in Algorithm 2.

**Algorithm 2** I2C interconnection identification

**INPUT:**
PnP peripheral module
**OUTPUT:**
$\mathcal{I}_{det}$, address ▷ I2C protocol identification
**PROCEDURE:**
Select I2C bus using Table II
Configure I2C bus;
**for** address = 0 to 127 **do**
    **if** Handshaking(address) = True **then** ▷ connected to I2C bus
        $\mathcal{I}_{det}$ = True;
        Return address;
    **else** ▷ Not connected to I2C bus
        $\mathcal{I}_{det}$ = False;

*3) ADC Identification:* Fig. 5 shows that resistor R4 is connected with ADC bus and digital pin D3 of the processor. When the logic of D3 is high (VCC), the resistor R4 works as a pullup resistor for the ADC bus. On the other hand, when the

logic of D3 is low (0 volt), the resistor R4 works as a pulldown resistor for the ADC bus. On the basis of pullup/pulldown resistor, we present an ADC protocol identification algorithm described in Algorithm 3 to identify the compatibility of the connected peripheral with ADC bus. Similar to other protocol identification algorithms, first, Algorithm 3 selects ADC bus using Table II and configures the bus. Then, resistor R4 is configured as a pullup and pulldown. If the sensing values after pullup and pulldown are not equal to VCC and GND, respectively, the peripheral is connected to the ADC bus and is compatible with the ADC protocol. Otherwise, sensing value should be held corresponding to activation condition of R4.

**Algorithm 3** ADC interconnection identification

**INPUT:**
PnP peripheral module
**OUTPUT:**
$\mathcal{A}_{det}$ ▷ ADC protocol identification
**PROCEDURE:**
Select ADC bus using Table II
ADC_Configuration();
Activate resistor R4 as a pullup resistor;
Sense ADC port of the processor, $\mathcal{A}^{\mathcal{U}}$; ▷ $\mathcal{A}^{\mathcal{U}}$ is the sensing value after pullup
Activate resistor R4 as a pulldown resistor;
Sense ADC port of the processor, $\mathcal{A}^{\mathcal{D}}$; ▷ $\mathcal{A}^{\mathcal{D}}$ is the sensing value after pulldown
**if** $\mathcal{A}^{\mathcal{U}}$ = VCC and $\mathcal{A}^{\mathcal{D}}$ = GND **then** ▷ VCC←+5 V and GND←0 V
    $\mathcal{A}_{det}$ = False; ▷ Not connected to ADC bus
**else**
    $\mathcal{A}_{det}$ = True; ▷ Connected to ADC bus

*4) Digital Identification:* We use simple logic to identify the compatibility of the connected peripheral with digital bus like ADC identification, as shown in Fig. 5. According to the value of digital pin D4 of the processor, resistor R3 works as a pullup/pulldown resistor for the digital bus. On the basis of this condition, we present an digital protocol identification technique described in Algorithm 4.

**Algorithm 4** Digital interconnection identification

**INPUT:**
PnP peripheral module
**OUTPUT:**
$\mathcal{D}_{det}$ ▷ Digital interconnect identification
**PROCEDURE:**
Select Digital bus using Table II;
Activate resistor R3 as a pullup resistor;
Sense Digital port D8, $\mathcal{D}^{\mathcal{U}}$; ▷ $\mathcal{D}^{\mathcal{U}}$ is the sensing value after pullup
Activate resistor R3 as a pulldown resistor;
Sense Digital port D8, $\mathcal{D}^{\mathcal{D}}$; ▷ $\mathcal{D}^{\mathcal{D}}$ is the sensing value after pulldown
**if** $\mathcal{D}^{\mathcal{U}}$ = HIGH and $\mathcal{D}^{\mathcal{D}}$ = LOW **then** ▷ Digital logic values
    $\mathcal{D}_{det}$ = False; ▷ Not connected to Digital bus
**else**
    $\mathcal{D}_{det}$ = True; ▷ Connected to Digital bus

*5) SPI Identification:* The SPI bus follows master-slave architecture for communication between a processor and a peripheral and uses four lines: MOSI, MISO, SCK, and CS. In this work, all the connected peripherals always work as a slave and the processor as a master has unidirectional control over all these peripherals. In SensPnP, SPI follows *rising leading edge* and *falling trailing edge* for transferring data to the slave device. Algorithm 5 presents the detailed identification procedure for SPI protocol. After initialization of the SPI bus, the processor sends a data request to the connected peripheral. If the processor receives the value less than 255 it indicates that the peripheral is compatible with SPI bus.

**Algorithm 5** SPI interconnection identification

**INPUT:**
PnP peripheral module, $C_{SD}$
**OUTPUT:**
$\mathcal{S}_{det}$ ▷ SPI protocol identification
**PROCEDURE:**
Select SPI bus using Table II
Configure SPI bus; ▷ Configure SPI bus
Begin SPI bus transaction; ▷ Begin SPI bus transaction
temp=SPISend($C_{SD}$); ▷ Send data request
End SPI bus transaction; ▷ End SPI bus transaction
**if** temp $< 255$ **then** ▷ Connected to SPI bus
    $\mathcal{S}_{det}$ = True;
**else** ▷ Not connected to SPI bus
    $\mathcal{S}_{det}$ = False;

## V. PnP Peripheral Driver Management

To communicate with a peripheral, the processor is expected to know a specific driving logic and the compatible interconnection protocol. We first offer a generic and novel driving algorithm for each protocol, so that any peripheral can be supported with the UniPnP control. In SensPnP, the UniPnP control itself manages the corresponding driver for the connected peripheral without any third party involvement, any prior information about the peripheral, and the Internet. Once the processor identifies the compatible protocol, it jumps to the corresponding protocol's driver management algorithm. We consider only heterogeneous sensors as a peripheral in this proposed PnP solution.

**Algorithm 6** UART driver management

**INPUT:**
PnP peripheral module
**OUTPUT:**
$S_D$ ▷ Sensor value
**PROCEDURE:**
UART_Configuration($D_R$); ▷ Initial condition
**while** ($I_F$ = True) **do** ▷ Peripheral is connected
    **if** Time = $S_T$ **then** ▷ Sensing condition
        $S_D$ = UART_Read(); ▷ Reading sensor data

*1) UART Driver Management:* The driver of UART compatible peripheral is presented in Algorithm 6. The communication of UART compatible peripheral does not depend on the request of the processor so that the processor receives the sensor data $S_D$ according to the value of sensing time $S_T$, where $D_R$ is the baud rate.

*2) I2C Driver Management:* For I2C protocol, we design a table to find out the value of access configuration $C_{AC}$, continuous conversion $C_{CC}$, start conversion $C_{SC}$, and sensor data reading $C_{SS}$ command corresponding to the address of the plugged I2C peripheral. The address of the peripheral is received from Algorithm 2. Algorithm 7 describes the detailed driver management procedure of I2C bus compatible sensor.

*3) ADC Driver Management:* We propose a simple algorithm of ADC compatible peripheral in Algorithm 8, where $V_{ref}$ and $R$ are the reference voltage and resolution of ADC, respectively. Each ADC compatible peripheral requires different calibration equations. Therefore, the proposed algorithm generates the sensor output $S_D$ in standard voltage format. In

**Algorithm 7** I2C driver management

**INPUT:**
PnP peripheral module, Address, $C_{AC}$, $C_{CC}$, $C_{SC}$, $C_{SS}$
**OUTPUT:**
$S_D$ ▷ Sensor data
**PROCEDURE:**
Configure I2C bus;
I2C_Configure_Peripheral(address,$C_{AC}$,$C_{CC}$); ▷ Configure the peripheral
I2C_Start_Conversion(address,$C_{SC}$); ▷ Start conversion
**while** ($I_F$ = True) **do** ▷ Peripheral is connected
  **if** Time = $S_T$ **then** ▷ Sensing condition
    I2C_Start_Reading(address,$C_{SS}$); ▷ Start sensing
    $S_D$ = I2C_Read(); ▷ Reading sensor data

**Algorithm 8** ADC driver management

**INPUT:**
PnP peripheral module,
**OUTPUT:**
$S_D$ ▷ Sensor value
**PROCEDURE:**
ADC_Configuration(); ▷ Initial condition
Pulldown resistor R4;
**while** ($I_F$ = True) **do** ▷ Peripheral is connected
  **if** Time = $S_T$ **then** ▷ Sensing condition
    $S_D$ = ADC_Read(ADC_channel); ▷ Read ADC at ADC_channel
    $S_D$ = ( $S_D$*$V_{ref}$)/$2^R$; ▷ Decimal to voltage conversion

our PnP solution, the resolution $R$ and voltage of ADC $V_{ref}$ are 10 bits and 5 Volts, respectively.

*4) Digital driver management:* Algorithm 9 describes the communication logic of Digital compatible peripheral. The peripheral always provides binary logical outputs either 1 (high) or 0 (low). Algorithm 9 initially configures D8 pin and resistor R3 as a digital input and pulldown, respectively. After that, the microcontroller collects sensor data from the digital bus based on the sensing time, while the peripheral should be connected to UniPnP controller. As an example, in the case of proximity sensor, if any hot blooded object comes in the area of the sensor, it generates *high* logic, otherwise always *low* logic.

**Algorithm 9** Digital driver management

**INPUT:**
PnP peripheral module,
**OUTPUT:**
$S_D$ ▷ Sensor value
**PROCEDURE:**
Digital_Configuration(); ▷ Initial condition
Pulldown resistor R3;
**while** ($I_F$ = True) **do** ▷ Peripheral is connected
  **if** Time = $S_T$ **then** ▷ Sensing condition
    $S_D$ = Digital_Read(Digitalchannel); ▷ Read digital logic at digital_channel

*5) SPI driver management:* Algorithm 10 describes the communication logic of SPI compatible peripheral. We follow SPI mode 1, which is discussed in Section IV-B. If the peripheral is connected to the UniPnP controller and the clock time of the processor is equal to sensing time $S_T$, the processor sends a data request instruction to the peripheral. Accordingly, the peripheral as a slave sends sensing data to the processor.

## VI. Implementation

To compute the overall effectiveness of SensPnP, a step-wise procedure of simulation and implementation is followed using different temperature sensors compatible with I2C, temperature sensor compatible with SPI, light dependent

**Algorithm 10** SPI driver management

**INPUT:**
PnP peripheral module, $C_{SD}$
**OUTPUT:**
$S_D$ ▷ Sensor data
**PROCEDURE:**
Configure SPI bus; ▷ Configure SPI bus
**while** ($I_F$ = True) **do** ▷ Peripheral is connected
  **if** Time = $S_T$ **then** ▷ Sensing condition
    Begin SPI bus transaction; ▷ Begin SPI transaction
    $S_D$=SPISend($C_{SD}$); ▷ Send data request
    End SPI bus transaction; ▷ End SPI transaction

TABLE III
Experimental Setup

| Parameter | Value |
|---|---|
| Processor | Low power, RISC-based microcontroller [24] |
| Clock frequency | 8 MHz |
| Operational voltage | 5 Volt |
| Protocols | ADC, Digital, I2C, SPI, and UART |
| IoT device price | 20 unit [26] |
| Internet Data cost/byte | 0.001 unit |
| Server maintenance charge/year | 3 unit [26] |

resistor (LDR) sensor with ADC and passive infrared (PIR) proximity sensor with Digital. For protocol bus switching and identification, we used $2 \times 1$ and $4 \times 1$ high speed complementary metaloxidesemiconductor (CMOS) analog multiplexer/demultiplexer [21], [22]. The simulation is done in the real-time embedded simulator [23] and a prototype is built for practical implementation. A Reduced Instruction Set Computer (RISC) architecture-based microcontroller [24] is used in both the cases.

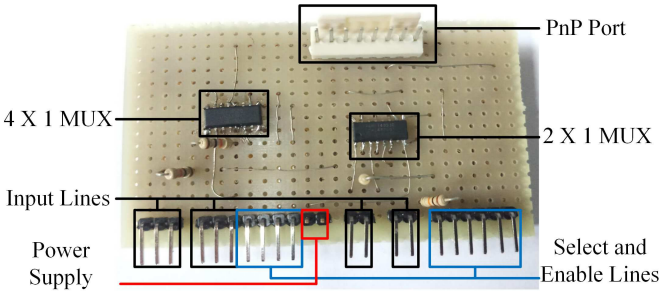



Fig. 6. Hardware design of protocol bus switching and identification

For the implementation purpose, we used a RISC architecture-based microcontroller [24], which offers 8 MHz 8-bit core, 32 KB of flash memory/read-only memory (ROM), and 4 KB of random-access memory (RAM). Fig. 6 shows the designed low-cost and efficient circuit for protocol bus switching and identification. The proposed algorithms are implemented in an embedded C compiler [25]. All the libraries used in the algorithms are written from scratch to make the code efficient.

## VII. Performance Evaluation

### A. Experimental Setup

To evaluate the performance of SensPnP, we conducted different experiments. The experimental setup is shown in Table III, while presenting different parameters. The value of

IoT device price and server maintenance charge (per year) are adopted [26].

We use different performance metrics — PnP time, protocol identification time, the memory footprint, lifetime, and overall cost for characterizing the performance of SensPnP. The overall cost is the combination of the cost for buying IoT device, using the Internet data (per byte), and maintaining the server (per year). We adopt the linear pricing model to compute the overall cost, while considering these parameters [27]. The objective of computing the overall cost is to show how much cost consumer has to bear for enabling PnP of a new peripheral apart from the price of IoT device.

### B. Results and Discussion

In this section, we present the performance of SensPnP over $\mu$PnP [10] and USB host [28] to show the effectiveness of the proposed solution. The computations are performed on the data obtained from the implementation of the prototype.

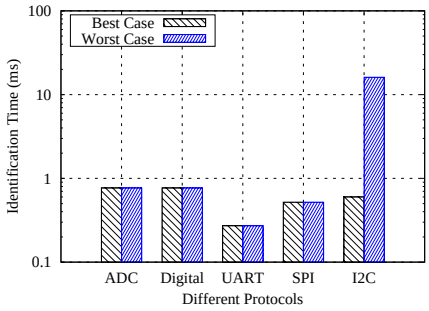



Fig. 7. Protocol identification time

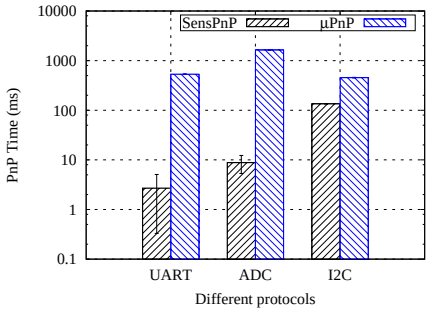



Fig. 8. PnP time for different protocols

*1) PnP Time:* Fig. 8 shows the time required for the complete PnP cycle of a newly connected peripheral considering all the steps starting from plugging in of the peripheral, protocol identification, driver management, single data collection, to plugging out of the peripheral. From the figure, it is evident that the time required for identifying and managing driver of the peripheral in SensPnP is significantly less compared to the $\mu$PnP due to the offline processing and light-weight protocol stack. SensPnP is itself capable of identifying the underlying compatible protocol and driver of the connected peripheral without any prior information and the Internet. However, in case of $\mu$PnP, the driver management process of a new peripheral is done using the Internet, which delays the entire PnP process. In SensPnP, the protocol identification process serially scans for each protocol. The time required for the identification process also depends on the order of arrangement of the protocols are arranged during the scan. It is evident from Fig. 7 that the difference between the best case and worst case protocol identification time of I2C is large, which is due to the time taken for matching the address of the connected peripheral. As the identification process of I2C protocol takes more time compared to other protocols such as ADC, Digital, SPI, and UART, the variation in confidential interval of ADC and UART is more.

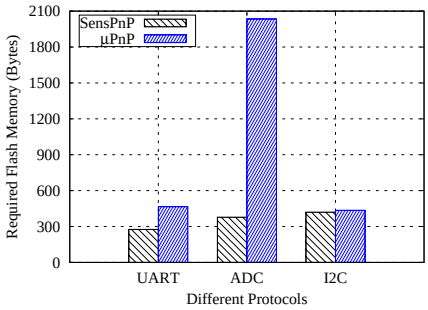



Fig. 9. Memory footprint for different protocols

*2) Flash/ROM Requirement:* We evaluate the resource requirement of different protocols in terms of flash/Read Only Memory (ROM), as shown in Fig. 9, while comparing with $\mu$PnP [10]. Fig. 9 shows that the flash memory requirement of SensPnP for different protocols is significantly less compared to $\mu$PnP. This significant performance is observed due to the development of all the libraries from scratch, which optimizes and reduces the requirement of flash memory of different protocols. For I2C protocol, we design a table to find out the value of $C_{AC}$, $C_{CC}$, $C_{SC}$, $C_{SS}$ corresponding to the address of the plugged I2C peripheral. Thus, the flash memory requirement of I2C protocol is sightly greater than that of UART and ADC.

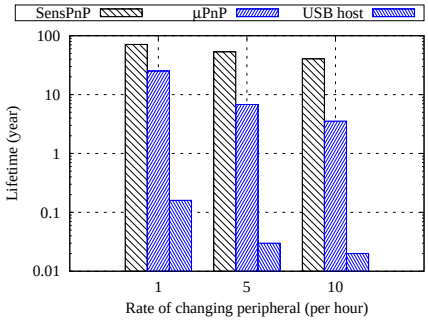
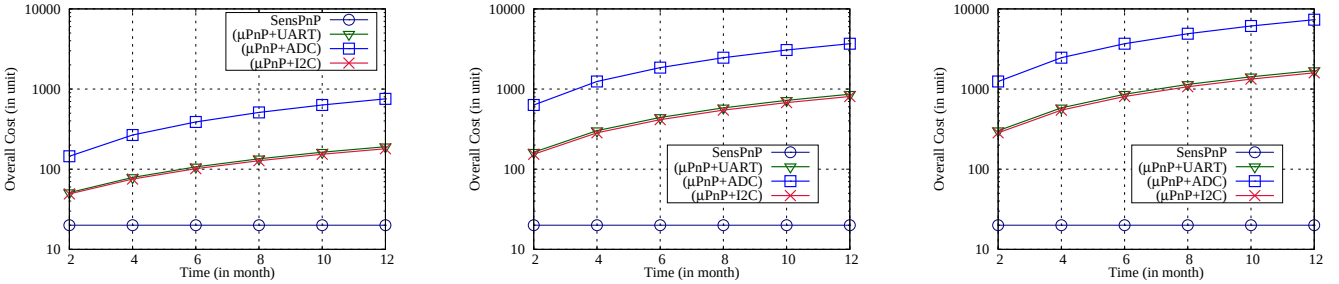



Fig. 10. Lifetime Analysis

*3) Lifetime:* We evaluated the lifetime of the developed protocol bus switching and identification (PBSI) circuitry during protocol identification with respect to the full charge capacity of a battery. We used a 80 mAh lithium-ion battery for this purpose. Fig. 10 shows the lifetime of the PBSI circuitry with respect to the rate of changing peripheral per hour, while




(a) Rate of changing peripheral (per day) = 1 (b) Rate of changing peripheral (per day) = 5 (c) Rate of changing peripheral (per day) = 10

Fig. 11. Overall cost analysis

comparing with $\mu$PnP [10] and USB host [28]. In case of $\mu$PnP and USB, the energy consumption of the identification circuit and the USB host controller is computed considering the minimum and ideal power consumption, respectively. On the other hand, we consider worst-case energy consumption for SensPnP. From the figure, it is clear that the lifetime of PBSI circuitry is significantly more compared to that of $\mu$PnP and USB, even in the worst-case situation. In the identification process of $\mu$PnP, the minimum current rating and identification time are 3.42 mA and 220 ms. Similar figures are seen for USB host which asserts that it is a power ravenous device. However, in the worst-case, the current rate of PBSI circuity is 1.81 mA and the identification time is maximum 18.23 ms. Due to the above mentioned factors, our proposed identification solution operates for longer duration of time compared to $\mu$PnP and USB.

*4) Overall Cost Analysis::* We analyze the overall cost of SensPnP over $\mu$PnP [10] from the consumer's perspective. Fig. 11 presents the overall cost by varying the rate of changing peripheral per day. We adopted the linear pricing model to compute the overall cost, while considering the cost for buying IoT device, using the Internet (per byte), and maintaining the server (per year) , as mentioned in Section VII-A [27]. Fig. 11 shows that the cost of SensPnP is noticeably less compared to $\mu$PnP due to the offline process of the protocol identification and driver management in contrast to that of $\mu$PnP. $\mu$PnP requires downloading of the corresponding driver of the peripheral whenever a new device is connected, which increases the overall cost apart from the hardware cost. In SensPnP, the cost only depends on the hardware cost of the IoT device. Thus, the cost-efficiency of the proposed PnP solution makes it more acceptable to the consumers when compared with $\mu$PnP.

### C. Applicability and Expected Benefits for Consumers

IoT has several application domains such as smart home, smart healthcare, smart farming, smart agriculture, smart industry, and smart cities, which need different sensors integration to IoT device. Consumers normally use the IoT devices for monitoring and controlling different activities of these applications. The proposed PnP solution helps consumer to integrate different sensors for their application. In addition, it provides consumers ease to use the same IoT device for multiple applications by only changing sensor modules, instead of purchasing different IoT devices for different applications. Therefore, SensPnP reduces the overhead cost of purchasing a new device every time with the change in their needs. Also, the complete abstraction of underlying technology allows even a naive consumer to use our PnP solution in the integration of different sensors on the IoT devices. Further, from Section VII-B4, it is evident that the cost-efficiency of the proposed PnP solution makes it more acceptable to the consumers when compared with $\mu$PnP. The proposed solution can be useful other consumer electronics products such that automatic washing machine, digital thermometer, soil meter, and refrigerator, where need to sensor integration.

## VIII. Conclusion

In this paper, we present a novel PnP solution for the integration of third-party embedded sensors with the IoT devices without any prior information about the sensors and the Internet. SensPnP is the combination of embedded hardware and firmware. We present an architecture of a PnP-enabled IoT device, which supports heterogeneous embedded peripheral communication protocols. A novel embedded protocol detection and automatic driver management of the connected sensor with an IoT device are presented. This is achieved through a low-cost and low power switching and identification hardware with a light-weight identification and driver management firmware stack. To show the effectiveness of SensPnP, we practically implemented a prototype in a real test-bed. Experimental results show that SensPnP requires minimal memory footprint, reduced energy consumption, and much reduced PnP time compared to the existing solutions. Additionally, the overall cost analysis shows that the cost of SensPnP is noticeably less compared to existing solutions.

In this work, we have only considered heterogeneous sensors as a peripheral device, but other peripherals such as actuator and wireless communication device were not considered. In IoT network, apart from sensor, these peripherals play an important role to make it global. However, the seamless integration of third-party actuators and wireless communication modules is required. The proposed PnP solution can be extended in future to address this issue.

## References

[1] S. S. Roy, D. Puthal, S. Sharma, S. P. Mohanty, and A. Y. Zomaya, "Building a sustainable Internet of Things: Energy-efficient routing using

low-power sensors will meet the need," *IEEE Consum. Electron. Mag.*, vol. 7, no. 2, pp. 42–49, Mar. 2018, doi: 10.1109/MCE.2017.2776462.
[2] E. Rubio-Drosdov, D. Díaz-Sánchez, F. Almenárez, P. Arias-Cabarcos, and A. Marín, "Seamless human-device interaction in the Internet of Things," *IEEE Trans. Consum. Electron.*, vol. 63, no. 4, pp. 490–498, Nov. 2017, doi: 10.1109/TCE.2017.015076.
[3] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, Oct 2010, doi: 10.1016/j.comnet.2010.05.010.
[4] H. Thapliyal, "Internet of Things-Based Consumer Electronics: Reviewing existing consumer electronic devices, systems, and platforms and exploring new research paradigms," *IEEE Consum. Electron. Mag.*, vol. 7, no. 1, pp. 66–67, Jan. 2018, doi: 10.1109/MCE.2017.2755219.
[5] D. C. Yacchirema and C. E. Palau, "Smart IoT gateway for heterogeneous devices interoperability," *IEEE Latin America Trans.*, vol. 14, no. 8, pp. 3900–3906, Aug. 2016, doi: 10.1109/TLA.2016.7786378.
[6] "Universal Serial Bus," USB Implementers Forum, Inc., Accessed: Oct. 2018. [Online]. Available: www.usb.org
[7] "SPI Block Guide v3.06," Motorola, Inc., Accessed: Oct. 2018. [Online]. Available: https://web.archive.org/web/20150413003534/http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf
[8] "I2C-Bus," i2c-bus.org, Accessed: Oct. 2018. [Online]. Available: www.i2c-bus.org
[9] A. Osborne, *An Introduction to Microcomputers : Basic Concepts*, 2nd edition, Ed. McGraw-Hill, 1980.
[10] F. Yang, N. Matthys, R. Bachiller, S. Michiels, W. Joosen, and D. Hughes, "$\mu$PnP: Plug and Play peripherals for the Internet of Things," in *Proc. of the Tenth European Conf. on Computer Systems*, New York, USA, 2015, pp. 25:1–25:14, doi: 10.1145/2741948.2741980.
[11] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "IoT middleware: A survey on issues and enabling technologies," *IEEE Internet of Things J.*, vol. 4, no. 1, pp. 1–20, Feb 2017, doi: 10.1109/JIOT.2016.2615180.
[12] M. Eisenhauer, P. Rosengren, and P. Antolin, "A development platform for integrating wireless devices and sensors into ambient intelligence systems," in *Proc. 2009 6th IEEE Annu. Commun. Society Conf. on Sensor, Mesh and Ad Hoc Commun. and Netw. Workshops*, June 2009, pp. 1–3, doi: 10.1109/SAHCNW.2009.5172913.
[13] D. T. Neves, M. Santos, and M. Pinto, "ReActOR: A middleware as a service to interact with objects remotely," in *Proc. 2015 IEEE Int. Conf. Ind. Technol.*, Mar. 2015, pp. 2433–2439, doi: 10.1109/ICIT.2015.7125456.
[14] H. Khaleel, D. Conzon, P. Kasinathan, P. Brizzi, C. Pastrone, F. Pramudianto, M. Eisenhauer, P. A. Cultrona, F. Rusinà, G. Lukàč, and M. Paralic, "Heterogeneous applications, tools, and methodologies in the car manufacturing industry through an IoT approach," *IEEE Syst. J.*, vol. 11, no. 3, pp. 1412–1423, Sept 2017, doi: 10.1109/JSYST.2015.2469681.
[15] P. Persson and O. Angelsmark, "Calvin merging cloud and IoT," *Procedia Computer Science*, vol. 52, pp. 210 – 217, 2015, doi: 10.1016/j.procs.2015.05.059.
[16] N. Sriskanthan, D. Tandon, and K. K. Lee, "Protocol for plug and play in Bluetooth based home networks," *IEEE Trans. Consum. Electron.*, vol. 50, no. 2, pp. 457–462, May 2004, doi: 10.1109/TCE.2004.1309408.
[17] Y. M. Baek, S. C. Ahn, and Y. Kwon, "Upnp network bridge for supporting interoperability through non-ip channels," *IEEE Trans. Consum. Electron.*, vol. 56, no. 4, pp. 2226–2232, Nov. 2010, doi: 10.1109/TCE.2010.5681094.
[18] T. Jo, Y. You, H. Choi, and H. Kim, "A bluetooth-UPnP bridge for the wearable computing environment," *IEEE Trans. Consum. Electron.*, vol. 54, no. 3, pp. 1200–1205, Aug. 2008, doi: 10.1109/TCE.2008.4637607.
[19] T. Sakamoto and K. Nimura, "Dynamic connection management between web apps and peripheral devices by web driver," in *Proc. 2016 IEEE Int. Conf. on Pervasive Comput. and Commun. Workshops*, Mar 2016, pp. 1–6, doi: 10.1109/PERCOMW.2016.7457152.
[20] N. Matthys, F. Yang, W. Daniels, S. Michiels, W. Joosen, D. Hughes, and T. Watteyne, "$\mu$PnP-Mesh: The plug-and-play mesh network for the Internet of Things," in *Proc. 2015 IEEE 2nd World Forum on Internet of Things*, Dec 2015, pp. 311–315, doi: 10.1109/WF-IoT.2015.7389072.
[21] "Triple 2-channel analog multiplexer/demultiplexer," nexperia.com, Accessed: Oct. 2018. [Online]. Available: https://www.nexperia.com/products/logic/i-o-expansion-logic/analog-switches/
[22] "Dual 4-channel analog multiplexer/demultiplexer," nexperia.com, Accessed: Oct. 2018. [Online]. Available: https://www.nexperia.com/products/logic/i-o-expansion-logic/analog-switches/
[23] "Software for circuit simulation," labcenter.com, Accessed: Oct. 2018. [Online]. Available: www.labcenter.com
[24] "Low power microcontroller," microchip.com, Accessed: Oct. 2018. [Online]. Available: https://www.microchip.com/design-centers/8-bit
[25] "C compiler for microcontroller," mikroe.com, Accessed: Oct. 2018. [Online]. Available: https://www.mikroe.com/mikroc-avr
[26] S. Misra, S. Chatterjee, and M. S. Obaidat, "On theoretical modeling of sensor cloud: A paradigm shift from wireless sensor network," *IEEE Syst. J.*, vol. 11, no. 2, pp. 1084–1093, June 2017, doi: 10.1109/JSYST.2014.2362617.
[27] C. Zhu, X. Li, V. C. M. Leung, L. T. Yang, E. C. H. Ngai, and L. Shu, "Towards pricing for sensor-cloud," *IEEE Trans. Cloud Comput.*, pp. 1–13, Jan 2017, doi: 10.1109/TCC.2017.2649525.
[28] "USB Peripheral/Host Controller with SPI Interface," www.maximintegrated.com, Accessed: Oct. 2018. [Online]. Available: https://www.maximintegrated.com/en/products/interface/controllers-expanders.html

**Sanku Kumar Roy** (S'15) received the B.Tech. degree in electronics and communication engineering from the Maulana Abul Kalam Azad University of Technology, India, in 2012. He is presently pursuing the Master of Science (by Research) degree in the Department of Computer Science and Engineering, Indian Institute of Technology (IIT) Kharagpur, India.

From 2014 to 2017, he was a Junior Research Fellow at Indian Institute of Technology Kharagpur. His current research interests include Internet of Things, Wireless Sensor Networks, and Wireless Body Area Networks.

**Sudip Misra** (SM'11) received the Ph.D. degree in Computer Science from Carleton University, in Ottawa, Canada, in 2005.

He is a Professor in the Department of Computer Science and Engineering at the Indian Institute of Technology Kharagpur, India. He has published 9 books in the areas of wireless ad hoc networks, wireless sensor networks, wireless mesh networks, communication networks and distributed systems, and network reliability and fault tolerance, published by reputed publishers such as Cambridge University Press, Springer, Wiley, and World Scientific.

Prof. Misra is a fellow of the National Academy of Science (India). Currently, he is serving as the Associate Editor of the IEEE Transactions on Mobile Computing, and IEEE Systems Journal. He is also an Editor of the IEEE Transactions on Vehicular Technology.

**Narendra Singh Raghuwanshi** received the Ph.D. degree from University of California, Davis, USA, in 1994.

He is a Professor in the Department of Agricultural and Food Engineering at Indian Institute of Technology Kharagpur, India. Currently, he is the director of National Institute of Technology (NIT) Bhopal and Indian Institute of Information Technology, Bhopal. He has published more than 100 peer reviewed papers, 2 book chapters and has 7 copyrights to his credit.

Prof. Raghuwanshi is the recipient of NAAS Recognition Award, Fellow of INAE, Rotary International Fellowship, Fulbright Nehru Senior Research Fellowship, and Dr. P. S. Kankhoje Memorial Gold Medal.