2014 IEEE NINTH INTERNATIONAL CONFERENCE ON INTELLIGENT SENSORS, SENSOR NETWORKS AND INFORMATION PROCESSING (ISSNIP)
SYMPOSIUM ON INTELLIGENT SENSORS
SINGAPORE, 21–24 APRIL 2014

WSN Platform Plug-and-Play (PnP) Customization

Ahmad El Kouche, Hossam S. Hassanein
School of Computing
Queen's University
Kingston, Ontario, Canada
Emails: {elkouche, hossam}@cs.queensu.ca

Khaled Obaia
Oil and Energy

Edmonton, Alberta, Canada
Email: Khaled.Obaia@gmail.com

***Abstract*— Configuring a WSN platform to work seamlessly for various applications is a desirable feature in WSN platforms. Such feature allows expedited design and deployment catering for new applications using existing platforms. This paper describes a Plug-and-Play (PnP) protocol which allows the customizing of WSN platforms for a broad range of applications using sensor modules. Sensor modules are independent sensing devices that attach to a WSN platform and provide an application dependent solution. Sensor modules are automatically detected and configured by Sprouts upon their attachment to one of the available PnP sensor module ports.**

I. INTRODUCTION

A WSN platform is a general purpose development system designed to accelerate research and development of WSN applications. A typical WSN platform combines a number of subsystems, including a wireless communication transceiver, low power microcontroller unit (MCU), energy sources, and a variety of onboard sensors such as temperature, humidity, accelerometer, light, etc. Designing a WSN platform that is low cost, ultra low power, physically small, rugged, multi-standard-compliant, and adaptable across various application domains introduces numerous challenges due to conflicting requirements leading to inevitable tradeoffs in the hardware, middleware, and software architecture.

Configuring a platform to work seamlessly for various applications with little user intervention is not a simple task. In fact, it is difficult to design a platform that can satisfy a wide range of applications without sacrificing performance. Currently on many platforms such as Telos [2], MICA [2], IRIS [2], SunSpot [3], etc., the user has to reprogram the MCU to read a specific digital or analog port connected to a sensing element. Reprogramming the sensor platform's MCU to use a given sensor element could involve a very daunting sequence of tasks including learning the embedded operating system, MCU architecture, middleware, application support layer, etc. Using our Sprouts PnP protocol, the user can simply plug a sensing device to our platform without any software configuration necessary for a given default functionality state. Consequently, customization is moved from the application layer to a simple PnP hardware layer.

A sensor platform which is not customizable for new applications would require the redesign of the sensor platform which would lead to increased costs, time delays, and retesting of the new sensor platform design, all of which are detrimental to any new research project or application. Customization is necessary to adapt the same sensor platform to a various and broad range of applications with little to no user coding effort involved. Thus, our motivation is to support easy customization on WSN platforms using PnP sensor modules, such that we place the development burden on the developers of the sensor modules. However, it allows the user to simply mix and match available sensor modules by simply plugging a desired sensing module suitable for a given application to one of the available ports on the Sprouts platform. Therefore, we present an approach to provide a high level of customization to the Sprouts platform to allow for application-specific layers to be added as PnP sensor modules using a standard serial communication bus.

We use SPROUTS [4] as the candidate WSN platform for our PnP module deployment. Sprouts [4] is a WSN platform designed for deployment in applications for harsh industrial environments [5][6][7]. Sprouts unique attributes include a modular architecture in the hardware and software architecture [8], low power Zigbee modified network [9], rugged metallic enclosure, patch antenna, wireless energy harvesting, remote triggering, and plug-and-play (PnP) sensor module interface. Figure 1 shows some of the optional internal modular layers that compose the Sprouts platform.

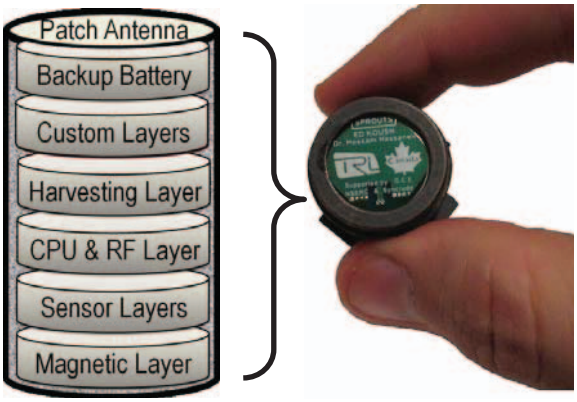

Figure 1 Sprouts Sensor Node showing internal stackable layers

We introduce an interface bus communication protocol between that SPROUTS sensor node and the PnP modules. Our Sprouts PnP protocol provides a unique customization method using sensor modules, which allows the user to implement custom modules for specific applications. Custom sensor modules can be interchanged, reused, or combined to make new applications as needed. To the best of our knowledge such PnP modules and communication protocol do not exist in the WSN platform literature.

978-1-4799-2843-9/14/$31.00 © 2014 IEEE

This paper is organized as follows. In Section II we provide an in-depth critique of alternative bus interface architectures for low-power WSNs. Section III details our PnP communication protocol. In Section IV we evaluate the performance of the SPROUTS PnP protocol. Section V concludes this paper.

II. Related Communication Buses Comparison

We investigate popular serial communication standards available for low power WSNs in order to determine a best fit serial protocol that we can build upon to establish a PnP protocol for the Sprouts platform. Below is a list of prominent serial protocols that are low cost and widely available for low power MCUs:

- **UART:** Universal Asynchronous Receive Transmit is one of the earliest standard serial communication interface, dating back to the early 1970s, created to establish a basic wired communication channel among different devices to exchange information. Enhanced versions of the UART are still widely in use in embedded systems, mobile phones, laptops, PCs, modems, printers, equipment, scanners, etc. The wide adoption of the UART is mainly due to its simplicity and ubiquity. The working principle of UART is to serialize data between two devices, such a serializing a Byte into a stream of 8 serial bits from the transmitting device, and then reconstructing the byte on the receiving device. A bidirectional UART communication can be established with as little as 3 wires (i.e. Transmit, Receive, Common-Ground). An optional hardware flow control (i.e. handshaking) is also available using request to send (RTS) and clear to send (CTS) additional pins. In order to establish a UART communication port, the user must specify the baud-rate (bits/s or bps), data length, parity, and stop bit. There are no standard protocols for UART communication, thus, it is up to developers to define their own.

Advantages: only 3 wires required, simple to use, and available on most low cost MCUs. UART-to-USB dongles and microchips are widely available**.**

Disadvantages: Very low scalability since it was designed for communication between two devices. Requires accurate crystal generated clock sources on both devices. Standard baud rates greater than 19.2Kbps (i.e 28.8 Kbps, 38.4Kbps, 56Kbps, etc.) may generate unreliable bit error rates (BER) if the crystal frequency used is not an integer multiple of the baud rate desired. UART is generally used with low communication speeds and a maximum baud rate approximately 1Mbps.

- **SPI**: Serial Peripheral Interface is a synchronous serial communication interface between one master and one or more slave devices. The master node always initiates the communication and asserts the required clock cycles to drive data flow. Data is commonly transferred in 8-bit segments with one clock pulse asserted by the master for every bit shifted, thus, no baud rate is required. SPI commonly requires 5 wires to operate in full-duplex mode between one master and one slave: master-out-slave-in (MOSI), master-in-slave-out (MISO), clock (CLK), and slave-select (SS), and common ground (GND). Additional slave devices can be added to the bus by dedicating additional lines for each (e.g. SS1, SS2, SS3, SS4, etc.). There are no standard protocols for SPI communication, thus, it is up to developers to define their own.

Advantages: Communication rates up to half the MCU clock source (e.g. 10 Mbps using 20MHz clocked MCU) are possible. No dependency on clock frequency accuracy, thus, high accuracy crystal driven clock sources are not necessary. Full-duplex mode can potentially reduce latency in some applications. Multiple slave devices can be supported.

Disadvantage: Full duplex mode requires 5 wires (MOSI, MISO, CLK, SS, and GND).

- **I^2C**: Inter Integrated Communication is a synchronous serial communication protocol which requires 3 wires to establish communication between one master and multiple slaves: Serial Data (SDA), CLK, and GND. Data transmission is bidirectional on the same data wire, SDA, and initiated by the master. Instead of using a slave-select for each device on the bus as is the case with SPI, I2C uses a unique addressing scheme to connect to a given slave device on the bus. Multiple masters are also allowed, such that pull-up resistors on SDA port (RSDA) and CLK port (RCLK) are used to avoid any damages that may occur due to data collisions. The maximum number of devices supported on the I2C bus is limited to the address space (i.e. 7-bit or 10-bit by design) and to the total bus capacitance (i.e. 400pF by design).

Advantages: Only 3 wires are required to establish a multi-master and multi-slave bus. Data is acknowledged after every transfer.

Disadvantages: Standard communication rate is 100kbps with a maximum of 400Kbps. Increased hardware and software design complexity. Careful timing is necessary. Pull-up resistors RSDA and RCLK both waste power for every logic-zero in the packet since the full voltage supply VDD is present across the resistors, thus, power consumption is increased.

- **LIN**: Local Interconnected Network is a serial asynchronous communication protocol which allows one master and up to 16 slave devices to communicate over one bidirectional wire. Thus, including a common ground wire, only 2 wires are needed to establish communication, which greatly reduces deployment costs. LIN emerged as a low cost alternative to the CAN bus in the automotive industry. The master node always initiates communication on the bus, and polls all slaves periodically allowing for deterministic time responses. LIN bus has a maximum baud rate of 19.2Kbps, up to 8 data bytes maximum payload, data acknowledgement, and data checksum.

Advantages: Low cost 2 wire communication (i.e. Data and GND). Multiple slaves are software addressable. Deterministic response times from slave nodes. Baud rate detection is possible using synchronization byte 0x55-hex, thus, allowing the use of inaccurate clock sources. Possible to daisy chain slave-devices to further reduce deployment costs.

Disadvantages: Careful timing is necessary. Limited number of slave devices up to 16 on one bus. Limited communication speed up to 19.2K.

- **CAN**: Controller Area Network is a message-based serial asynchronous communication protocol which focuses on high reliability applications such as industrial and automotive, but not limited to either. Priority arbitration in CAN is similar to a logic-AND gate, such that logic-zeros on the bus are dominant and logic-ones are recessive, which provides contention immunity and grants message IDs with a lower address (i.e. more logic-zeros in the address) higher priority to continue transmission. Nodes with lower priority immediately switch to receive-mode upon contention detection (i.e. when the output bit is not equal to the transmitted bit). CAN specifies an 11-bit or 29-bit extended message identifier (MID), which is a predefined application-specific field that identifies the type of data on the bus (e.g. temperature sensor, motor control, switch, rotary dial, etc.). In other words, the messages sent on the bus are unique, however, the nodes are not. A given application may divide the 11-bit MID field into a unique 6-bit address space and a 5-bit sub-node message identifier. Thus, the maximum number of nodes on a single CAN bus is undefined, but conveniently set 64 nodes (i.e. derived from 6-bit address) for 11-bit MID width. Since all messages are broadcasted on the bus, nodes may individually decide whether to act on the received data or ignore it. A maximum of 8 bytes can be transmitted per packet.

Advantages: Highly reliable physical-layer data bus for electromagnetically noisy environments. No master or slave node concept or limitation, only messages transmitted are uniquely identified. Fault tolerant bus using CRC, contention immunity, differential twisted-pair transmission, and data acknowledgements.

Disadvantages: Costly to implement since it requires at least 3 wires (i.e. differential pair, and GND) and a CAN hardware transceiver. CAN controllers are not available on low cost MCUs, and complex to emulate in software using bit-banding techniques. Higher power consumption is necessary for the differential pair communication. A maximum of 8-byte payload allowed. No addressing space specified by standard.

III. SPROUTS PLUG-AND-PLAY (PNP) PROTOCOL

In order to customize Sprouts for a broad range of applications, we design a PnP protocol which accepts externally attached custom modules, which we refer to as Sensor Modules (SM). A SM can be a generic or an application dependent device, compliant with Sprouts PnP protocol, and may include one or more sensor elements (SEs).

To avoid any naming confusions, we define the following:

- **Sensor Element (SE)**: Elementary type sensor with little or no embedded control. Usually requires an MCU to manage its output. For example, temperature sensor MCP9700A is a SE.
- **Sensor Module (SM)**: Custom solution device that typically manages one or more SEs and is compliant with Sprouts PnP protocol. Can be a generic or application dependent solution.
- **Sensor Node (SN)**: Wireless sensor node in a WSN, such as Sprouts sensor platform.

Sprouts PnP protocol is established between Sprouts and SMs in order to achieve a high level of customization. Due to the small size constraint of Sprouts, only 4 SMs can be connected at any given time. However, one SM may contain multiple SEs, thus, 4 SMs limitation does not pose any serious constraints. In addition, a custom SM may be designed to act as a PnP expansion module to provide more PnP ports. Moreover, serial communication protocols in industrial applications are driven by cost, such as moving from a 3-wire CAN network to a simpler 2-wire LIN bus which reduces system complexity and the number of physical wires needed. However, our PnP protocol is designed for direct connection with sensor nodes without using cable extensions, which is similar to how a microSD memory card is directly plugged into a smartphone. Therefore, the cost limitation of cables is eliminated allowing us to expand the number of connections needed to design a simple to use PnP serial communication bus. Each one of the four PnP ports on the Sprouts platform has a standard 8-pin rectangular header (1.27mm pitch), such that each pin is described in TABLE I. .

TABLE I. SPROUTS PNP SERIAL PROTOCOL BUS

Pin#	Acr.	Name	Owned by	Description
1	**GND**	Ground	Common to all	Common ground reference necessary to establish a voltage potential.
2	**VDD**	DC Power Supply	Common to all	3.0V DC power provided by Sprouts battery to power external sensor modules. Modules may also have their own power.
3	**MOSI**	Master Out Slave In	Sprouts Node	MOSI is a unidirectional SPI data link from Sprouts to the SM.
4	**MISO**	Master In Slave Out	all Sensor Modules	MISO is a unidirectional SPI data link from the SMs to Sprouts. Only one SM with an active PS-x* may use this pin.
5	**CLK**	Clock	Sprouts Node	Clock signal provided by the master device (i.e. Sprouts) to drive data.
6	**PSx***	Port Select-x*	Sprouts Node	Port Select pin to choose one of four SMs to communicate with Sprouts.
7	**ENx***	Enable-x*	Sprouts Node	External Enable interrupt signal to wake up the sensor module from sleep.
8	**RDx***	Ready-x*	Unique per Sensor Module x*	Module-x* ready to send or receive data. Necessary when a new sensor module is attached for the first time.

x* denotes the PnP port number (or SM number) from 1 to 4. Pins with x* are unique per SM.

When a module is plugged into one of the four ports, the module receives power via GND and VDD, and initiates a registration event with Sprouts by activating the RDx pin followed by a registration request event. Sprouts wakes up via RDx, configures the SPI protocol, sets ENx High causing RDx to go Low, sets PSx High, and requests the module's unique ID via SPI when the RDx pin is set High again. The module replies back with its unique 32-bit ID, such that every reply is accompanied with the request command that triggered the response. Sprouts then sends a request to the sink node requesting the module's configuration parameters. These configuration parameters are set by the module's designer and are unique to each module. The configuration parameters allow for further customization per application requirements. For example, for our ultrasound thickness module [10], we have configuration parameters related to material type, calibration values, gain, acquisition time, average number of samples, reporting intervals, duty cycles, etc. When the configuration setup is complete, the module relays the number of sensor ports available to Sprouts with the corresponding duty cycle of each. At this point, it is the responsibility of Sprouts to request a

sensor reading update from the specified port of the module at the appropriate time interval. Sensor ports represent the various sensor elements that may be present in a single sensor module, as seen in Figure 2. In addition, Figure 3 shows Sprouts attached to four ultrasound sensor modules. For example, a module may include a temperature sensor at port 1, light sensor at port 2, humidity at port 3, and a barometer at port 4. Sprouts communicates with all active module ports to collect readings on all sensor ports, combines the data in one packet, and sends one report packet to the sink node in order to save power consumed by the transceiver. In the case of a critical event detection by a module, interrupts from identified modules are permitted via RDx. These interrupts are served in high priority and reported to the sink immediately. When a module is detached from Sprouts, it times out after three unresponsive requests, or when a new module is attached triggering a registration event. Modules may be recycled from one application to another, allowing for a greater degree of customization per application.

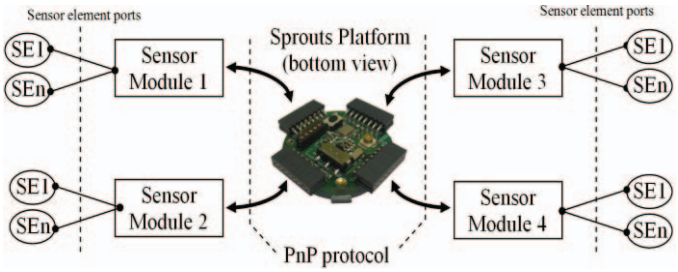

Figure 2 Four PnP modules with each having one or more sensor element (SE) ports allowing easy customization per application

PnP Packet Format

The PnP packet is composed of a 4 bytes header and a variable payload from 1 to 252 bytes.

The PnP header is composed of 4 bytes, as shown in TABLE II. and described in the following sequential order:

1) Start of Frame (SOF): this is a constant 1 byte (0xFE) to indicate the start of frame.
2) Length: calculated length of the entire packet including the header and the payload. MAXIMUM_LENGTH = 256.
3) Message Type (MSG_TYPE): the type of the message sent. For example, the message type might be a REQUEST (0x01) or RESPONSE (0x02).
4) Command identification (CMD_ID): specifies the type of command requested. For example, a READ_SENSOR_PORT (0x01),CONFIG_SENSOR_PORT (0x02), READ_REGISTER (0x03), WRITE_REGISTER (0x03), ACK (0x03), and RESET (0xFF).
5) Payload (PAYLOAD): the payload depends on the MSG_TYPE and CMD_ID. If the MSG_TYPE is REQUEST, and the CMD_ID is to READ_SENSOR_PORT, then the payload will specify which sensor ports to read.

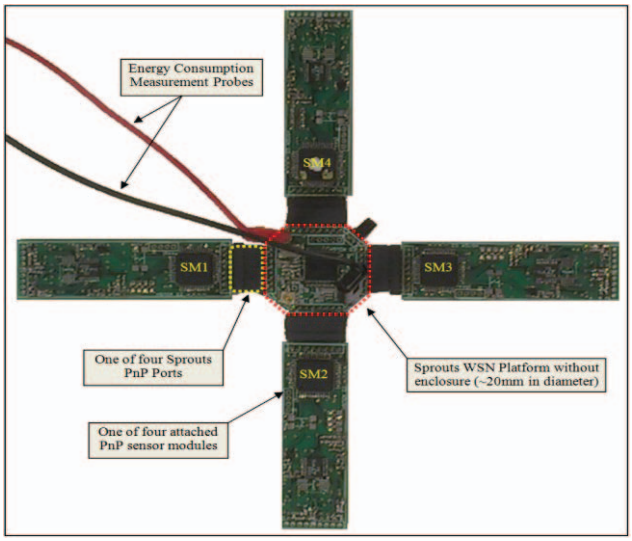

Figure 3 Sprouts node attached to four sensor modules

TABLE II. SPROUT PNP GENERAL PACKET FORMAT

	SOF	LENGTH	MSG_TYPE	CMD_ID	PAYLOAD
size	1 Byte	1 Byte	1 Byte	1 Byte	1 to 252 Byte

For example, using our Ultrasound Thickness Module (UTM) [10], the ultrasound sensor is placed on sensor port 2. Therefore, to obtain an ultrasound thickness reading, the Sprouts node sends the following packet to the UTM:

TABLE III. REQUEST AN ULTRASOUND MEASUREMENT READING EXAMPLE

	SOF	LENGTH	RESPONSE	READ_SENSOR_PORT	PORT_ID
Hex value	0xFE	0x05	0x01	0x01	0x02

The UTM would reply with a RESPONSE to ACK the requested message, and proceed immediately to take thickness measurements. Upon completion of the thickness measurements and before the expiration of a specified timeout period, the UTM sends another RESPONSE packet to the READ_SENSOR_PORT with the ultrasound thickness measurements in the PAYLOAD, as follows:

TABLE IV. RESPONSE TO ULTRASOUND MEASUREMENT REQUEST EXAMPLE

	SOF	LENGTH	RESPONSE	READ_SENSOR_PORT	Ultrasound Readings
Hex value	0xFE	0x05	0x02	0x01	steel_thickness, tungsten_thickness

For increased protocol reliability in a harsh operating environment, and in order to assure that the response packet is directly related to the last requested packet sent, the response packet confirms that the response is related to a READ_SENSOR_PORT command and the first byte in the payload is the Port_ID followed by the actual useful sensor data reply (i.e., steel and tungsten thicknesses).

IV. SPROUTS PNP PERFORMANCE ASSESSMENT

Error Detection and Recovery: Redundancy in the protocol is important to detect any errors during the PNP protocol communication, which increases the reliability of packet transmission and error recovery. Our PNP uses an SPI bus to transfer data, which has a unique feature of being a

bidirectional bus. We leverage this unique feature to echo back the bytes received by either device for an added level of error detection, which increases the likelihood of detecting any transmission errors. The Sprouts node checks all the bytes received to verify the correctness of the packet format, and sends an ACK back to the sensor module (i.e. the UTM). If any of the two devices receives a packet format that does not comply, then a PNP_ERROR_COUNT variable is incremented, such that when the total number exceeds 3 errors, an error recovery function is executed to deal with the type of error generated. On the UTM, the *pnp_error_recovery(void)* function simply resets the device which includes the PNP initiation *pnp_init(void)* and configuration *pnp_config(void)* functions upon startup. On the Sprouts node, however, the *pnp_error_recovery(void)* only executes the necessary *pnp_init(void)* and *pnp_config(void)* functions. Resetting the Sprouts node is avoided inside the *pnp_error_recovery(void)* function in order to avoid rejoining the network which is a power intensive procedure in a Zigbee network.

Bit Error Rate (BER): In asynchronous serial communication buses, such as UART, I2C, LIN, or CAN, two communicating devices are assumed to transmit and receive at the same data rate in order to receive the correct stream of data. Hence, when two asynchronous communicating devices are using slightly different clock rates, mostly due to hardware limitations or the crystal frequency of choice, a sampling error occurs due to a timing mismatch. The rate at which this timing mismatch leads to an incorrect sampling of data is referred to as the bit error rate (BER). On the other hand, the SPI serial communication link is a synchronous bus, such that the timing required to transmit and receive data on the SPI bus is determined by the clock (CLK) line which is present on the bus. Therefore, no previous knowledge of the data rate is necessary when using the SPI bus. The Sprouts node generates the clock for the SPI bus, and the sensor nodes use this clock to determine when to sample the channel to read data or transmit data. As long as the data rate on the SPI bus does not exceed the maximum data rate specified per MCU, no transmission or reception errors will occur in the sampling of data during communication on the SPI bus. Thus, the only source of error in communication would be an external source, such as long cable length subject to electromagnetic interferences. Our PnP protocol uses direct plugin modules, thus there are no necessary cables associated with our platform.

Communication Speed: The communication speed of the PnP protocol is limited to a maximum of 4Mbps in bidirectional (full duplex) mode on the Sprouts platform, which is defined by the CC2530 for a running frequency of 32MHz, and operating voltage between 2V and 3.6V. No minimum operating speed is defined for an SPI communication bus, thus very low speeds are possible, such as 1 Kbps. However, such low speeds are undesirable for WSNs, since the MCU has to be in active mode while receiving packets, which will increase the active time cycle, thus, increases power consumption.

Energy Consumption Per Bit: The PnP protocol runs at a full speed of 4Mbps, which means that a single transmitted bit spans 0.25µs. The power consumption of the MCU in active mode running at 32MHz consumes an average of 6.5mA at 3V. Therefore, the energy consumption per bit is approximately 4.875 nJ. The PnP minimum packet size is 5 bytes and the maximum packet size is 256 bytes. Therefore, the minimum energy consumption to send the smallest packet is 195nJ (over 10 µs period), and the maximum energy consumption to send a 256 bytes packet is approximately 9.984 µJ over a 0.512ms time period. A common packet length, such as the ultrasound has a maximum payload length of 12 bytes and a 4 bytes header overhead, thus, the energy consumption to send an ultrasound thickness report packet is approximately 0.624 µJ.

Implementation Cost: The SPI serial bus is available on almost all low cost MCUs, which makes it possible to implement a low cost sensor module using any MCU of choice. In addition, since the SPI bus provides a clock signal, sensor modules are not obligated to utilize a crystal to obtain accurate timing for data rates. In fact, no crystal is necessary, which lowers implementation cost. In the case SPI hardware is not present on an ultra low cost MCU, the SPI bus is the simplest communication bus to emulate in software using standard digital ports. Furthermore, implementation cost is also specified by the number of pins needed to establish a basic connection including power and ground. Sprouts PnP protocol utilizes an 8 pin header (i.e. VDD, GND, MISO, MOSI, CLK, PSx, ENx, and RDx). However, our PnP protocol was designed for direct plug-in modules that do not utilize a cable length, only connectors are necessary. A cable may still be used, however, the cost of the cable, depending on its length, will increase implementation cost. The cost of an 8-pin header (with locking) is approximately $0.96 US per sensor module, or $3.84 US on the Sprouts platform, which utilizes four headers.

Development Complexity: The SPI communication bus is the second simplest serial communication bus to develop a protocol, with the simplest being the UART. However, the SPI bus does not have many of the limitations that the UART bus does, such as speed, scalability, or bit error rate. In addition, we keep every communication line on the bus separate from the other, which increases the number of pins required, but reduces development complexity associated with switching between transmit and receive, input and output, or ready and busy signal, which makes it easy to code and easy to debug the protocol while in development. Our PnP protocol also has a minimal packet format that is intuitive to understand and use. In out PnP protocol, we do not pack different variables into individual bits inside a byte, which further simplifies development and reduces code complexity. Every sensor module is treated as a source of sensor readings provided at given ports. To read a sensor port, a simple packet is sent from the Sprouts platform to the sensor modules indicating a reading is required from a specified port. To read a different port, the same packet format is sent again but with a different port number, and so on. A single reading from a single port makes development very simple and straight forward.

Scalability: Scalability is defined as the ability to add more sensor elements to the Sprouts platform with minimal influence on system complexity and performance. Our Sprouts platform was designed to be physically small to accommodate a wide range of applications. Therefore, the physical size of the platform limits the number of sensor module ports to four. However, this does not necessarily limit the number of supported sensor elements. Any sensor module attached to a

Sprouts platform is not limited to a maximum number of sensor ports, hence, the sensor module may utilize as many sensor elements as possible by the sensor module's hardware. Therefore, scalability is dependent on the number of ports supported by sensor module's hardware, and not by the PnP protocol or Sprouts platform.

Sprouts PnP Attributes Comparison

In TABLE V. , we compare the performance and attributes of the most prominent serial communication interfaces and protocols available for WSNs. Except for our PnP protocol, none of the other aforementioned serial interfaces and protocols in Table V take into consideration ultra-low power wireless sensor nodes into consideration, which typically requires an external wakeup signal to exit deep sleep modes, which we enable using ENx. In addition, protocols such as LIN and CAN were designed for automotive sensors where power is abundantly available. Furthermore, LIN and CAN are typically unavailable on low cost MCUs and are process intensive to emulate by software. I2C requires additional power for every logic-zero transmitted due to pull-up resistors on the bus. UART requires a quartz crystal to achieve higher speeds than 19.2Kbps, which becomes costly for low cost sensor modules. SPI requires the most wires to implement, thus, it is most suitable for plug-in devices (e.g. microSD memory cards) as opposed to cable extended devices. The most attractive feature of SPI is the fast transfer rate which does not require any accurate timing since the clock signal is provided by the host. In addition, SPI does not specify a protocol to adhere to, making it a good candidate for developing new protocols. Our unique PnP protocol differs in many ways from the other serial interfaces and protocols in TABLE V. in that it:

TABLE V. ATTRIBUTES COMPARISON OF AVAILABLE SERIAL INTERFACES

Serial Interface	Defines a protocol	Multi-Master	Max payload size (bytes)	Scalability (Max Nodes Supported)	Max Bitrate (10^3 bits/s = 1 Kbps)	Wires/ports needed (with GND)	Software Development Complexity	Requires accurate Timing	Low power wakeup
UART	No	No	Undef.[e]	1	921.6[c] Kbps	3 wires	Very Low	Yes	No
SPI	No	No	Undef.[e]	Limited[a]	8 to 16[d] Mbps	5 pins	Low	No	No
I²C	Yes	Yes	Undef.[e]	127 or 1023	1 Mbps	3 wires	Med	Yes	No
LIN	Yes	No	8	16	19.2 Kbps	2 wires	Med	Yes	No
CAN	Yes	Yes	8	Limited[b] (64 typ.)	1 Mbps	3 wires	High	Yes	No
Sprouts PnP	Yes	No	252	4 sensor modules	4 Mbps	8 pins	V.Low (PnP)	No	Yes

a. Requires 1 extra wire per node; b. limited by bus current drive and capacitance; c. maximum standard bitrate, however, higher non-standard bitrates are possible up to 16 Mbps; d. limited by MCU clock speed; e. limited by available RAM;

1) allows sensor modules to be plugged into our Sprouts platform at any given time without user configuration,

2) supports sensor modules with ultra low power sleep modes, which can only wakeup using an external trigger.

3) allows easy incorporation into sensor modules since all signal lines have separate functionality. In addition, the packet format and protocol are simple, yet versatile, which further simplifies software development.

4) works with large payload size which can support up to 252 bytes of data exchange. This can be essential for future protocol enhancements and large memory transfer.

5) is not affected by accurate data transfer timing, thus, the bit-error-rate (BER) is not an issue.

V. CONCLUSION

In this paper we provide the design and implementation of Sprouts Plug-and-Play communication protocol to achieve seamless customization. Easy user customization is a key feature of WSN platforms necessary to achieve maximum applicability across a broad range of applications. We have successfully used our PnP modules for a multiplicity of applications, including in the oil industry and smart grid monitoring. Based on our experiences with building WSN platforms, we have found that the most successful implementation of a WSN platform is one that requires the least amount of modification to adapt for various applications. Our PnP protocol enabled efficient customization of the SPROUTS platform [4] for a multiplicity of application deployments [5-10] requiring only changing sensor modules to achieve desired monitoring and reporting requirements. Indeed, using a module drop-in solution greatly accelerates product to market time, reduces system complexity, and allows developers or users to focus more time on the application layer.

REFERENCES

[1] P. Dutta, J. Taneja, J. Jeong, X. Jiang, and D. Culler. "A building block approach to sensornet systems," in the 6th ACM conference on Embedded network sensor systems, 2008, pp. 267-280.

[2] Memsic WSN sensor nodes: MicaZ, Mica2, IRIS, TelosB, last modified: November 2012.

[3] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White, "Java™ on the bare metal of wireless sensor devices: the squawk Java virtual machine," in the 2nd ACM international conference on Virtual execution environments (VEE), 2006, pp. 78-88.

[4] A. El Kouche, "Towards a wireless sensor network platform for the Internet of Things: Sprouts WSN platform," in the IEEE International Conference on Communications (ICC), 2012, pp.632-636.

[5] A. El Kouche, L. Al-Awami, H. Hassanein, K. Obaia, "WSN application in the harsh industrial environment of the oil sands," in the 7th IEEE International Wireless Communications and Mobile Computing Conference (IWCMC), 2011, pp. 613-618.

[6] A. El Kouche, H. Hassanein, K. Obaia, "Monitoring the reliability of industrial equipment using wireless sensor networks," in the 8th IEEE International Wireless Communications and Mobile Computing Conference (IWCMC), 2012, pp. 88-93.

[7] A. El Kouche, A. Alma'aitah, H. Hassanein, K. Obaia, "Monitoring Operational Mining Equipment Using Sprouts Wireless Sensor Network Platform," in the 9th IEEE International Wireless Communications and Mobile Computing (IWCMC), 2013, pp.1388-1393.

[8] A. El Kouche, L Al-Awami, H. Hassanein, "Dynamically Reconfigurable Energy Aware Modular Software (DREAMS) Architecture for WSNs in Industrial Environments," Procedia Computer Science, vol. 5, pp. 264-271, 2011.

[9] A. El Kouche, M. Rashwan, H. Hassanein, "Energy Consumption Measurements and Reduction of Zigbee based Wireless Sensor Networks," to appear, IEEE International Global Communications (GLOBECOM) Conference, 2013, pp. 1-6.

[10] El Kouche, H. Hassanein, "Ultrasonic Non-Destructive Testing (NDT) Using Wireless Sensor Networks," Procedia Computer Science, vol. 10, pp. 136-143, 2012.