MHP: Master-Handoff Protocol for Fast and Energy-Efficient Data Transfer over SPI in Wireless Sensing Systems

Seung-mok Yoo and Pai H. Chou

Serial peripheral interface (SPI) has been identified as a bottleneck in many wireless sensing systems today. SPI is used almost universally as the physical connection between the microcontroller unit (MCU) and radios, storage devices, and many types of sensors. Virtually all wireless sensor nodes today perform up to twice as many bus transactions as necessary to transfer a given piece of data, as an MCU must serve as the bus master in all transactions. To eliminate this bottleneck, we propose the master-handoff protocol. After the MCU initiates reading from the source slave device and writing to the sink slave device, the MCU as a master becomes a slave, and either the source or the sink slave becomes the temporary master. Experiment results show that this master-handoff technique not only cuts the data transfer time in half, but, more importantly, also enables a superlinear energy reduction.

Keywords: Energy efficiency, master-slave switch, SPI bus, wireless sensor network.

I. Introduction

Generally, today's wireless sensor nodes consist of a microcontroller unit (MCU), a number of sensor devices, a radio transceiver, and, possibly, a non-volatile storage device such as flash memory [1]-[9]. A typical program for a sensor node reads data from a sensing device, performs processing if necessary, and sends the data to either the RF module or a flash memory device. Some wireless sensor applications simply transfer data from a data source (for example, sensors or a camera) to a data sink (for example, flash memory) in store-and-forward mode [10]-[12]. For some applications, the total power consumption exceeds the system power estimate because of the data transfer overhead [12]. The overhead can seriously affect the overall performance in such an application. Reducing the overhead can improve the performance in terms of the application execution time and energy efficiency.

Many monitoring applications demand high throughput and lower latency during their active intervals. For example, structural health monitoring based on acceleration requires about 500 to 2,000 samples per second per axis [13]. Electrocardiograms (EKG or ECG) require 125 to 1,000 samples per second per channel. Cameras have also been incorporated into several sensor platforms [2], [11], [14]-[16]. Although local processing can potentially reduce the bandwidth demand, this is not always possible or sufficient. In applications such as cameras, the data after compression may still be relatively large compared to a sporadic sampling of temperature. These applications can easily demand 50 kbps to several hundred kbps when active, regardless of their idle

Manuscript received Apr. 21, 2011; accepted Feb. 24, 2012.

This work was supported in part by the Dual Use Technology Program (Korea), Information and Telecommunication National Scholarship (Korea IITA), the National Science Foundation CAREER Grant CNS-0448668, UC Discovery Grant itl-com05-10154.

Seung-mok Yoo (phone: +82 42 860 5882, yoos@etri.re.kr) is with the IT Convergence Technology Research Laboratory, ETRI, Daejeon, Rep. of Korea.

Pai H. Chou (phchou@uci.edu) is with the Department of Electrical Engineering and Computer Science, UC Irvine, USA, and is also with the Department of Computer Science, Nat'l Tsing Hua University, Hsinchu, Taiwan.

http://dx.doi.rog/10.4218/etrij.12.0111.0209

 © 2012 ETRI

intervals.

A straightforward implementation of a high-throughput or low-latency application has a difficult time achieving its theoretical peak performance. One may be tempted to suggest direct memory access (DMA) and other similar techniques to help perform block data transfers. However, such techniques may not help for several reasons. First, hardware modification may not be an option if a particular platform must be used. Second, even if additional hardware can be added, DMA will only free the MCU from handling the transfer itself, but it will not remove the bottleneck. Third, the number of serial peripheral interface (SPI) transactions to transfer data from the data source to the data sink does not change. We might not expect to see a significant improvement in energy efficiency from DMA, either.

To understand how a bottleneck occurs in today's architecture, it is necessary to understand the interface used by today's MCUs, namely SPI, which works in a master/slave manner. SPI is used as the primary physical interface between the RF module and the MCU in virtually all sensor nodes today. SPI is commonly used for a sensor/ADC-to-MCU or MCU-to-storage interface as well.

Researchers have pointed out that SPI creates a bottleneck in the system [12]. However, SPI itself is not problematic. It's the way SPI is used that is the problem. As a master/slave protocol, SPI by default incurs two separate bus transactions for each piece of sensor data. One transaction is used for the MCU to read from the data source (for example, sensors and flash memory), and the other is used to write the data to a data sink (for example, an RF module or storage device). The reasons for the MCU's involvement may be to packetize the data, perform signal conditioning, compression, or event detection, or any combination thereof. However, the MCU need not always be involved because specialized hardware may already be handling the processing or the data may have already been processed. Even if the MCU must be involved for purposes such as maintaining buffered data for retransmission if the packet is lost, using two serialized transactions adds unnecessary latency because, strictly speaking, the data sink has no data dependency on the MCU. Doubling the number of bus transactions caps the throughput to at most 50% of the throughput as the theoretical maximum, and the latency is at least twice the minimum latency. In both cases, such a design can achieve at most half the data transfer performance and at least double the energy consumption.

To overcome this artificial barrier to high throughput and low latency, we propose a new scheme called the master-handoff protocol (MHP) for bus transactions. According to previous work, the source-to-MCU and MCU-to-sink transactions are serialized because the MCU has to act as a bus master in both cases. In our scheme, the MCU as the master first sets up the transactions for both slaves. Second, the MCU hands off its master role to either the data source or data sink, which temporarily becomes a new master for the payload transfer. At the same time, the MCU has the option of snooping the bus as a slave, allowing it to keep a copy of the data for retransmission if necessary. Third, the new master hands the master role back to the MCU, allowing the MCU to complete the remaining bus transaction.

By eliminating the artificial serialization requirements, the MHP doubles the throughput and halves the latency on the system bus. As a result, energy consumption is reduced, not only because the MHP reduces the number of SPI transactions, but also because the reduced bus utilization makes available more power management opportunities. Another important advantage is that, without hardware modifications, this technique is applicable to not only virtually all existing wireless sensor platforms but also to embedded systems in general that transfer data from a data source to a data sink over SPI. Experiment results show that our technique achieves 173% throughput improvement, 64% latency improvement, and 64% energy improvement when the MHP is applied to an SPI data transfer from the data source to the data sink.

The remainder of this paper is organized as follows. Section II provides an overview of SPI and describes wireless sensor platforms. Section III describes our proposed MHP in detail. Section IV describes our implementation and presents the experiment results at both the system level and wireless, end-to-end level.

II. Related work

1. SPI Bus Transactions

SPI is a serial interface proposed by Motorola for synchronous communication between a processor and peripheral devices or between processors. SPI is commonly used for both off-chip and on-chip communications in many embedded systems today.

SPI can be expanded to a bus, which consists of three bus wires plus a separate slave select. The three bus wires connect to the SPI clock (SCK), master out slave in (MOSI), and master in slave out (MISO). The master is responsible for generating the SCK, and it must also assert a separate slave select (SS) signal to select the slave device during a bus transaction [17]. Each SPI controller contains a shift register, and the master-slave registers form a loop during a bus transaction. On each SCK pulse, the master and selected slave exchange one bit of data: the master shifts one bit to the slave through the MOSI line, and the slave also shifts one bit to the master through the MISO line. The granularity of an SPI

transaction is a byte, and an SS signal should be asserted throughout the transaction.

2. Wireless Sensor Platforms

Many wireless sensor platforms developed to date use SPI. This section reviews Berkeley motes, Eco, PASTA stack, RISE – Co-S, Stack, TinyNode, and XYZ as representative examples.

The MICA series and Telos series [3], [9], [18], [19] are generally referred to as Berkeley motes and follow the two-SPI topology. In both architectures, the external memory and radio transceivers are connected to the MCU via SPI. However, the difference is that the MICA series uses two separate SPI connections for the flash and RF, whereas the Telos series uses a shared-bus topology instead. In MICA, the only way to transfer data from one SPI device to the other is for the MCU to read data from one SPI bus and to write to the other SPI bus. In Telos, it is assumed that at most one SS is asserted at a time for devices on the same SPI bus. Neither Berkeley mote supports SPI on the expansion connector, even though it would be easy to add more SPI devices to the motes.

The Eco wireless sensor node [8] also uses SPI for both on-chip and on-board connections with peripherals. Eco uses the Nordic nRF24E1 MCU, which contains an 8051 MCU core and an nRF2401 radio transceiver. The MCU accesses the radio transceiver through an internal SPI in a point-to-point topology. The external SPI is used for connecting to the external EEPROM and an additional external SPI device, which can be connected via its 16-pin flexible-PCB expansion interface. The SPI controller on the nRF24E1 is hardwired to work as a master. The only way to transfer data between one SPI device and the RF is for the MCU to explicitly copy data. However, it is possible to add another SPI device to the expansion interface. In this case, the added SPI device can communicate with the EEPROM over SPI.

The PASTA stack [12] is different from other wireless sensor platforms in terms of the philosophy and the platform capability. The PASTA stack is a distributed system, whereas other wireless sensor platforms are processor-centric systems. Each module in the stack has a low-power MCU that manages a power switch and controls the external bus access. The external bus contains two separate SPIs. MCUs on the modules can negotiate to use the SPIs. Any module can transfer data among other modules through the bus.

RISE – Co-S [20] consists of two platforms. The RISE platform contains an 8051MCU core and an RF transceiver. The platform has an SPI bus interface. Devices such as the RISE storage board can be connected through the interface. The Co-S platform is a co-processing system connected to the RISE platform over a serial interface. Co-S has two serial interfaces that can be configured as SPI ports. Any device can be connected to both of the platforms through SPI.

The MIT Stack platform [2] has an SPI port on the expansion connector on the board. The flash memory is connected through the SPI port. Stack also has an RF transceiver. If SPI devices are connected to the expansion connector, the devices can communicate with the flash memory over SPI.

TinyNode [4], [21] has two SPI ports [22]. Similar to Telos, TinyNode also uses a shared bus topology for connecting its flash memory and RF through one SPI port, which is not exposed to the expansion connector. The other SPI port is connected to the expansion connector on the board. SPI devices connected to the other SPI port can communicate over this SPI port. However, it is not easy to transfer data between an SPI device on the first SPI port and another SPI device on the other SPI port.

The XYZ sensor node [6] has one SPI port. The RF transceiver is connected to the MCU through the SPI. The XYZ node has an expansion connector, but the SPI is not part of the expansion connector. It is not easy to add another SPI device to the platform.

If a platform has an expansion connector that contains pins for an SPI bus, it is called *SPI-expandable*, and we can apply MHP to devices on the same SPI bus. Eco, PASTA stack, RISE – Co-S, Stack, and TinyNode are known to be SPI-expandable.

III. MHP

The purpose of the MHP is to enable a data transfer from the data source (for example, sensor or data storage) to the data sink (for example, RF transceiver or flash memory) without incurring double transactions. Because both the source and the sink are SPI slave devices, the MCU must initiate a setup with both devices. The MCU then hands off its master role to either the source or the sink, which becomes an SPI master temporarily so that it can perform a direct transfer. Upon completion of the data transfer, the temporary master hands over the master role back to the MCU as before.

1. Bus Connection

The MHP requires SPI devices to form a bus topology, which is shown in Fig. 1. Another requirement is that the MCU and one of the slaves involved in the handoff must be configurable as either a master or a slave of the SPI bus dynamically. When the slave and the master change roles, their pin directions should be reconfigured, too. Specifically, on the MCU, the SS, MOSI, and SCK pins should be set to input, and the MISO pin should be set to output. On the temporary

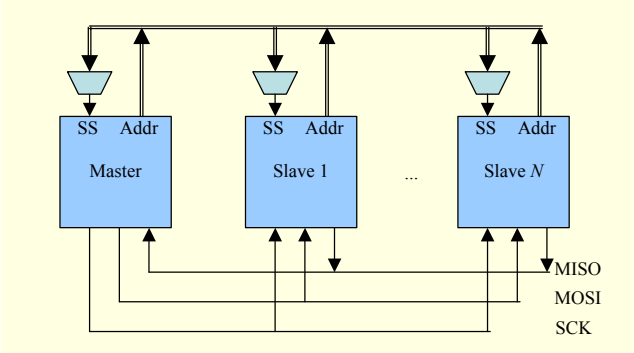

Fig. 1. Abstract SPI bus for MHP.

Table 1. Primitives for MHP.

Primitives	Parameters	Descriptions
Read	variable	Read a variable (MCU → source or sink)
Write	variable, value	Update a variable (MCU → source or sink)
SwitchToTX	SinkAddr	Make data source master (MCU → source)
SwitchToRX	SourceAddr	Make data sink master (MCU → sink)
SwitchBack	none	Hands back master role (temp master → MCU)

master, the MOSI, SCK, and address pins should be set to output, and the MISO pin should be set to input. The temporary master should drive the SCK line.

2. Variables for MHP

The data source and data sink have variables, which are used for configuring the MHP. *SPIConfig* contains the SCK speed, clock polarity, clock phase, and bit order configurations. *PayloadSize* represents the length of the sense data from the data source to the data sink in one SPI transaction.

3. Protocol

When the master starts an SPI transaction, it selects a slave by asserting the SS of the slave. Since the SS is an active low signal, "asserting" means setting it to a low voltage. If the slave receives the first byte of the SPI transaction, it invokes a function to interpret the byte, which contains a primitive, as shown in Table 1. Only the current acting master can send the primitives to its slaves.

Variables are accessible by the *Read* and *Write* primitives. *Read* takes a variable name as an argument. The master combines *Read* and a variable name to make a command, and sends the command to a slave. When the slave receives the *Read* primitive, it extracts a variable name from the command. For example, when the master wants to get the value of a variable, named *PayloadSize*, of the data source, the master sends the *Read(PayloadSize)* command to that data source. Depending on the variables, the length of the return value can be different. The names and data lengths of the variables are predefined on all SPI nodes. Similarly, *Write* also takes a variable name and its value as arguments. Upon receiving a *Write* primitive, the slave extracts the variable name and waits for the value from the master. This is because SPI can send only one byte at a time. If the slave receives the value, it updates the variable.

SwitchToTX, *SwitchToRX*, and *SwitchBack* are for the master and slave to switch roles. *SwitchToTX* is for the data source to become a temporary master. Upon receiving the *SwitchToTX (SinkAddr)* command from the MCU, the data source extracts a data sink address from the command and changes its role to master. After the data transfer is completed, it sends *SwitchBack* to the MCU and changes its role back to slave. In the meantime, the MCU waits until it receives the *SwitchBack* command and becomes a master again. *SwitchToRX* is for the data sink to become a temporary master. Similarly, if the data sink receives the *SwitchToRX(SourceAddr)* command from the MCU, it extracts a data source address from the command and becomes the master. After it receives data from the data source, it sends the *SwitchBack* command to the MCU and changes its role back to slave.

Figure 2 shows an example of a data transfer using the MHP. The MCU starts a data transfer by sending *SwitchToTX* to the data source. After sending *SwitchToTX*, the MCU becomes a slave. The data source becomes the temporary master and transfers data to the data sink. If the number of bytes sent is equal to *PayloadSize*, the data source sends *SwitchBack* to the

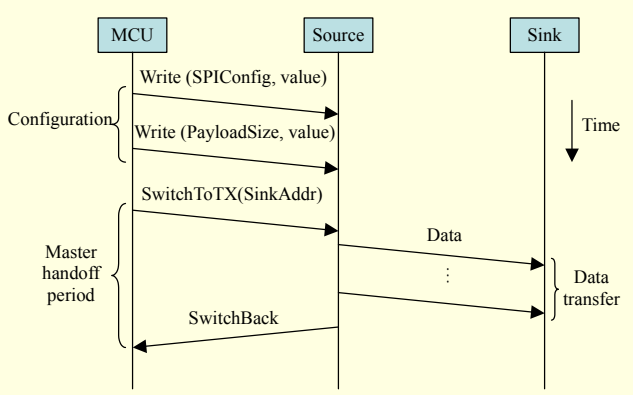

Fig. 2. Example of data transfer using MHP; data source becomes temporary master during data transfer.

MCU and becomes a slave. Upon receiving *SwitchBack*, the MCU becomes the master and the data transfer ends.

4. Snooping the SPI bus

The MHP supports an option for the MCU to snoop the SPI bus while it is acting as a slave device during the source-to-sink transfer. The reason for this option is that the MCU is often responsible for managing the communication protocol. In case of a packet loss, the MCU will need to transmit another copy of the data. By snooping the bus, the MCU can also obtain a copy of the sensor data "for free," without incurring additional bus transactions.

To accomplish snooping, the SPI pin configuration on the MCU should be different from the conventional configuration (Fig. 3). On virtually all MCUs, SPI pins are configurable as GPIO ports by default [17], [22]. When the software enables SPI, these pins are then connected to the SPI port. However, the pin directions are not automatically configured, because they are dependent on the role of SPI. As a slave, the MOSI, SCK, and SS pins are supposed to be set to input, while the MISO is set to output. The pin directions should be configured in the software. For snooping the SPI bus, we must make the MCU a half-duplex device, since the conventional pin configuration as a full-duplex device would result in a bus conflict on the MISO wire. This can be solved in several ways. In general, each SPI pin is connected to an internal bi-directional port with an optional internal pull-up register [17]. There is also a tri-state between the pin and the internal port. By setting the tri-state on the MCU's MISO pin to high impedance, the pin can be disabled, when the MCU is in slave mode. If the MISO pin of the MCU supports an open-drain output (that is, a logical 1 indicates a disconnection from the external resistive pull-up, while a logical 0 indicates a real 0), then the MCU can write a 0xff byte after it receives each byte to accomplish the same effect as a high-impedance output.

IV. Performance Evaluation

1. Application and Assumptions

We test the MHP on a suite of wireless sensing applications that demand high throughput and low latency during active intervals. ECG-type applications typically demand 125 to 1,000 samples per second per channel. The camera module in [16] supports VGA and QVGA resolutions. The MCU on the platform can receive 4.1 frames per second from the camera at QVGA resolution. To transmit such a video image to the base station in real time, the data rate will need to exceed 250 kbps.

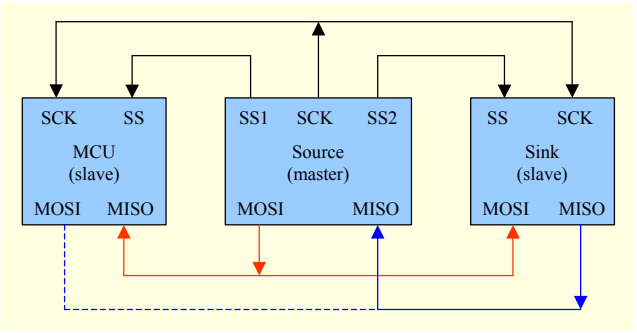

Fig. 3. Block diagram for snooping SPI bus.

Across all of these applications, the required throughput ranges from 50 kbps to 100 kbps.

When data is transferred, either a data source or data sink can be a temporary master. No matter which one becomes the temporary master, the number of SPI transactions does not change. For a performance evaluation, we choose the data source as the temporary master without loss of generality.

2. Data Transfer Performance on SPI

A. Configuration for the Experiment

For the MHP performance evaluation, we implement two different versions of a data transfer application and compare the experiment results. To transfer the data, one version uses a baseline (no-MHP) approach and the other uses the MHP. Except for their data transfer approaches, the versions are almost identical. For the baseline approach, the MCU receives data from the data source and sends the data to the data sink. However, the data source cannot spontaneously send sensor data to the MCU through the SPI bus unless the master activates the SPI data transfer. There are two ways that the slave sends data to the master. One is that the master waits for a predefined time and reads the SPI data register on the slave. The other one is that the slave uses one extra signal line. We use the latter one. Even if the latter one needs one more extra signal line, it may be faster than the first one. We refer to the extra signal as *DataReady*. If the data source receives a command to load data to the SPI data register, the data source loads the data and sets *DataReady* to high. If the data on the data source is transferred, the data source sets *DataReady* to low. The baseline approach uses the SPI bus and an I/O signal for *DataReady*, whereas the MHP approach uses only the SPI bus.

MCU: For our experimental platform, we use Atmel's AVR Butterfly, a wearable, multi-sensor platform based on the ATMega169V [23]. We set one up as the sensor node and one as the data source. The ATMega169V, an AVR 8-bit RISC MCU [17], runs at up to 8 MHz. The processor core without

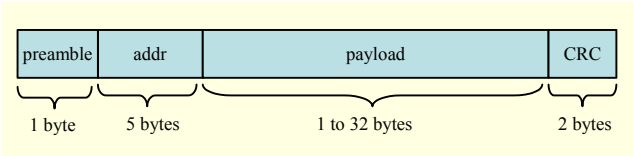

Fig. 4. nRF24L01 RF transceiver packet format.

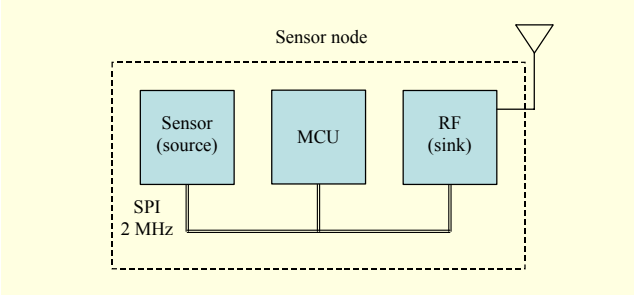

Fig. 5. Block diagram for experimental setup from source.

peripherals on the MCU is equivalent to the core on the ATMega128L, which is the MCU on MICA2 [3]. The ATMega169V MCU is thus representative of many wireless sensor platforms today. The MCU is configured as the bus master for the experiment.

Data Source: We set up a second AVR board as the bus interface to a high-speed data source. Conceptually, the data source has a similar structure to the ADC module of the PASTA stack, which has an 8051 MCU operating in store-and-forward mode on the module [12]. If the data source becomes a temporary master, the data source should know the SCK speed, polarity, and phase, as well as the bit order. The SPI configuration information is stored in the *SPIConfig* variable and given at the configuration stage by the MCU.

Data Sink: For the data sink, we set up one Nordic nRF24L01 RF transceiver module, which uses the 2.4 GHz to 2.527 GHz ISM band [24]. Figure 4 shows the packet format. The packet starts with one byte preamble, which is automatically added by the RF transceiver. The address length is configurable from 3 bytes to 5 bytes. We use a 5-byte address. The payload size in the nRF24L01 packet is configurable from 1 to 32 bytes. We measure the latency and throughput with a payload size of 4 to 32 bytes in 4-byte increments. A 16-bit CRC is added and checked by the RF transceiver. The transceiver has an SPI port on it. The SPI port is handled in hardware. Thus, SPI data can be sent and received without a time delay between consecutive bytes.

For our experiment, the SCK rate is set to 2 MHz (Fig. 5), the maximum rate achievable by the AVR boards. Theoretically, a 2 MHz clock can transmit data at 250 kbps. This clock rate is fast enough to handle both a 2.4 GHz 802.15.4/Zigbee data rate [25] and the Nordic transceiver.

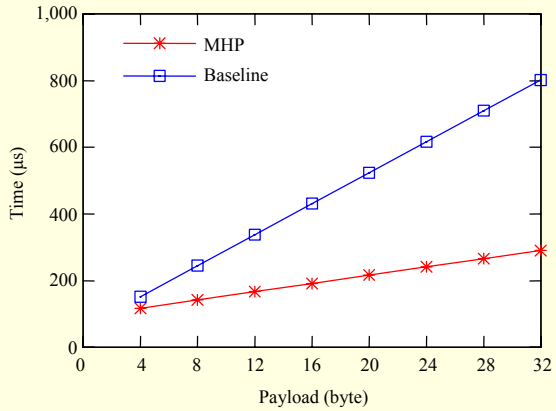

Fig. 6. Latency comparison.

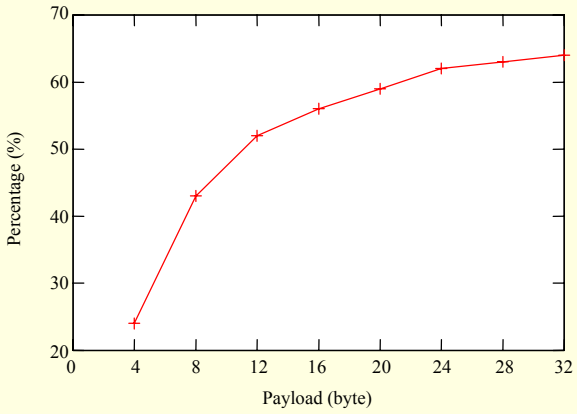

Fig. 7. MHP latency improvement over baseline.

B. Latency and Throughput

The latency and throughput for the baseline are measured from the start of the data source access to the completion of the data transfer to the data sink on the MCU. The latency and throughput for the MHP are measured from the *SwitchToTX* transmission to the *SwitchBack* arrival on the MCU. Figure 6 compares the two latency measurements. When the payload size is 4 bytes, we observe a latency of 152 μs for the baseline, and 117 μs for MHP. The difference is 35 μs. As the payload size increases, both curves grow linearly as does the latency difference. At the maximum payload size of 32 bytes, the latency of the baseline is 802 μs versus MHP's 291 μs. Figure 7 shows the MHP latency improvement against the baseline. When the payload size is 32 bytes, the latency improvement is 64%. If the data size is greater than or equal to 12 bytes, the latency improvement is greater than 50%. In other words, if the size of data transferred is greater than or equal to 12 bytes, the MHP latency is less than half of the baseline latency. If the payload size is greater than or equal to 24 bytes, the

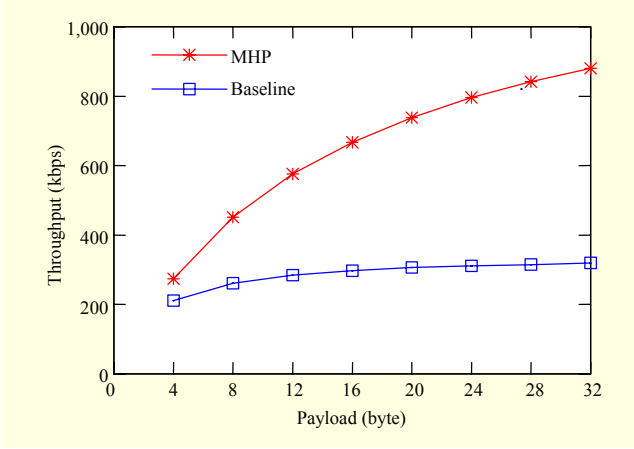

Fig. 8. Throughput comparison.

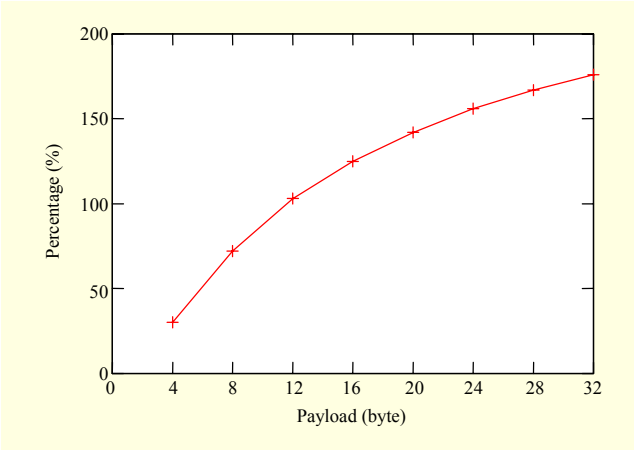

Fig. 9. MHP throughput improvement over baseline.

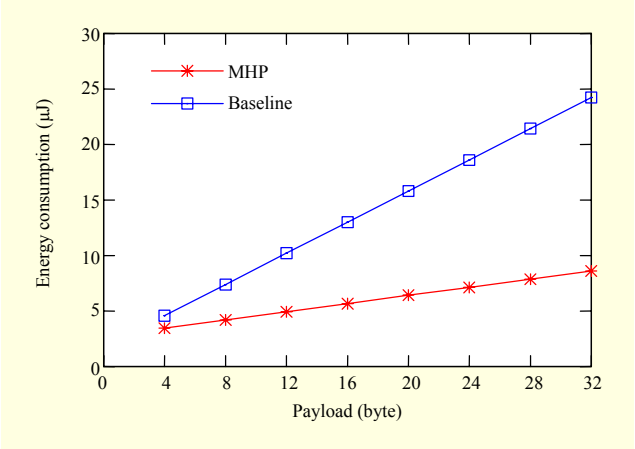

Fig. 10. Energy consumption comparison.

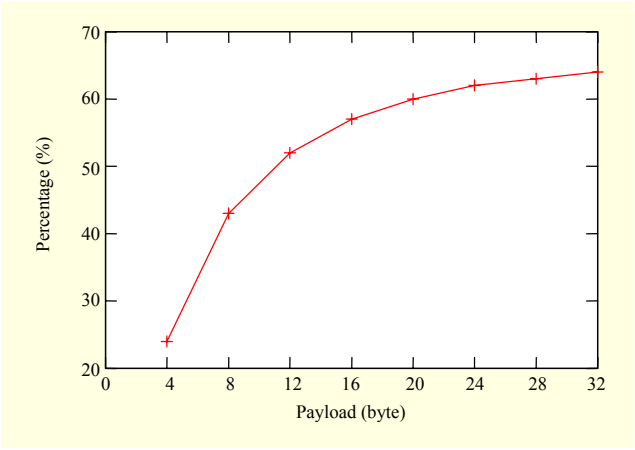

Fig. 11. MHP energy consumption improvement over baseline.

improvement is greater than 60%, and the improvement curve becomes stable. To increase the latency performance, it is more efficient to send more than 24 bytes of data at a time.

Figure 8 shows the measured throughput comparison. At a payload size of 4 bytes, the baseline throughput is 211 kbps versus MHP's 274 kbps, with a difference of 63 kbps. As the payload size grows, the difference also grows. At the maximum payload of 32 bytes, the throughput of the baseline is 319 kbps versus MHP's 880 kbps. The difference is 561 kbps. Figure 9 shows the MHP throughput improvement. If the payload size is greater than or equal to 12 bytes, the MHP throughput improvement is greater than 100%. When the payload size is 32 bytes, the throughput improvement is 176% compared to the throughput of the baseline. Unlike the latency improvement curve, the throughput improvement curve steadily grows up to 32 bytes, although the growth rate goes down.

The super-linear improvement in latency and throughput is due to the fact that SPI transactions between the MCU and the data source take a longer time than those between the MCU and the data sink. This is because the RF transceiver handles SPI data in hardware, whereas on the MCU, software needs to access the SPI hardware register, and this takes more time. Another difference is that in the MHP, the MCU and data source communicate with each other during the role change period. By our measurement, it takes 12 μs for the data source to become a master after the start of the transaction. This is a negligibly small number in terms of the SPI transaction time. Thus, the baseline latency is more than twice the MHP latency, and the baseline throughput is less than half of the MHP throughput.

C. Energy Consumption

Energy efficiency is one of the important issues in wireless sensor network applications. Figure 10 shows the measured energy consumptions during one data transfer transaction from the source to the sink. For the baseline, the energy consumption is measured for two SPI transactions: one transaction from the source to the MCU and the other transaction from the MCU to the sink. For the MHP, the energy consumption is measured for only one SPI transaction from the source to the sink. When

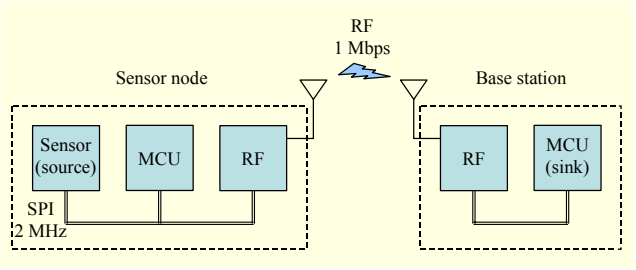

Fig. 12. Block diagram for experimental setup from source to base station.

the payload size is 4 bytes, the energy consumed is 3.46 μJ for the MHP and 4.58 μJ for the baseline. The difference is 1.12 μJ. At the maximum payload size, the energy consumption for the MHP is 8.61 μJ and the energy consumption for the baseline is 24.23 μJ. The difference is 15.62 μJ. Like the latency and throughput graphs, the energy consumption graph grows linearly. Figure 11 shows the MHP energy consumption improvement on the SPI data transfer. The energy consumption of the MHP is 76% of that of the baseline at a payload size of 4 bytes. At the maximum payload size, the energy consumption of the MHP is 36% that of the baseline. This is a 64% improvement in energy consumption.

The shape of the curve in Fig. 11 is quite similar to that of the curve in Fig 7. This is because the *power* consumption of both approaches is nearly identical and the *energy* consumption depends on the latency. If the data size is greater than or equal to 12 bytes, the latency improvement is greater than 50%. If the payload size is greater than or equal to 24 bytes, the improvement is greater than 60% and the improvement curve becomes stable. Like the latency, it is more efficient to send more than 24 bytes of data at a time.

3. Source to Base Station Performance

Localized improvements may not lead to global, end-to-end improvements due to Amdahl's Law. This section presents measurement results that quantify the impact of the MHP, an on-board bus protocol, on the global performance of a data transaction from a sensor on the wireless sensor node to the base station. Our metrics are the throughput, data transfer time, which is the time difference from the start of the i-th packet transmission to the start of the (i+1)th packet transmission, and energy consumption for the data transfer time. The (i+1)th transmission starts once the i-th transmission completely ends.

A. Configuration for the Experiment

We configure one additional node for a base station to evaluate the system-wise performance. The base station has a similar configuration as the sensor node. It uses the same AVR board and nRF24L01 RF transceiver module, but its sensors are not used. Data from the sensor node to the base station is transferred through the transceiver at 1 Mbps in air (Fig. 12).

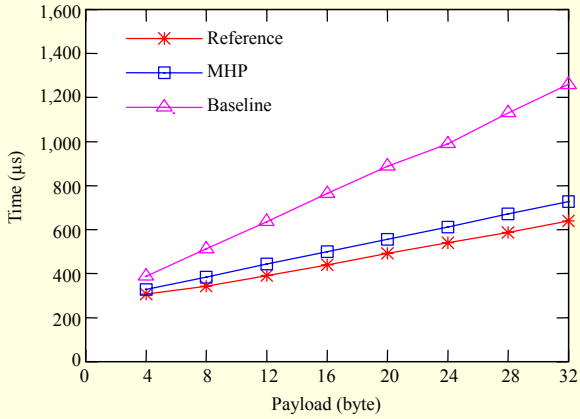

Fig. 13. Data transfer time comparison.

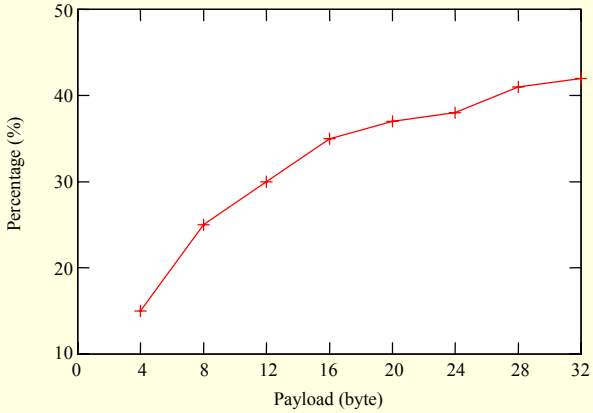

Fig. 14. MHP data transfer time improvement over baseline.

B. Data Transfer Time and Throughput

Figure 13 shows the data transfer time measurements from the data source to the base station. A reference in this case involves the MCU, which continuously transmits dummy data over the RF transceiver to the base station, without any sensor data. The reference can be a lower bound of the data transfer time. At 4 bytes of payload, the data transfer time of the baseline approach (double-transaction) is 388 μs versus MHP's 328 μs, with a difference of 60 μs. As the payload size increases, all three curves grow linearly. At a maximum payload size of 32 bytes, the data transfer time of the baseline is 1,260 μs versus MHP's 728 μs, or a 532 μs difference. If the data transfer time for the MHP is compared to the reference, the reference time is 308 μs at 4 bytes of payload and 640 μs at the maximum payload size. The difference is 20 μs and 88 μs, respectively. The data transfer time for the MHP takes 114% of

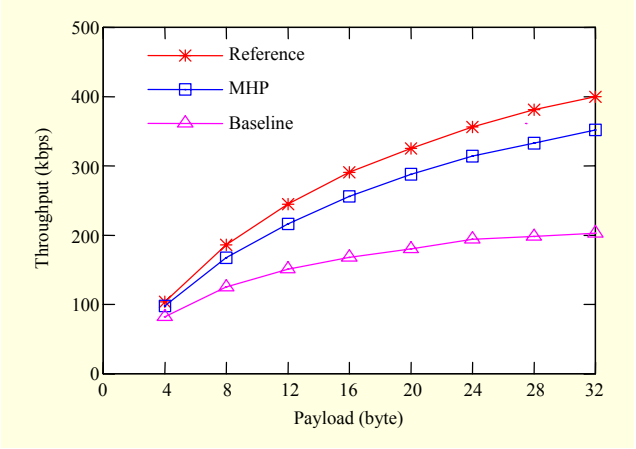

Fig. 15. Throughput comparison.

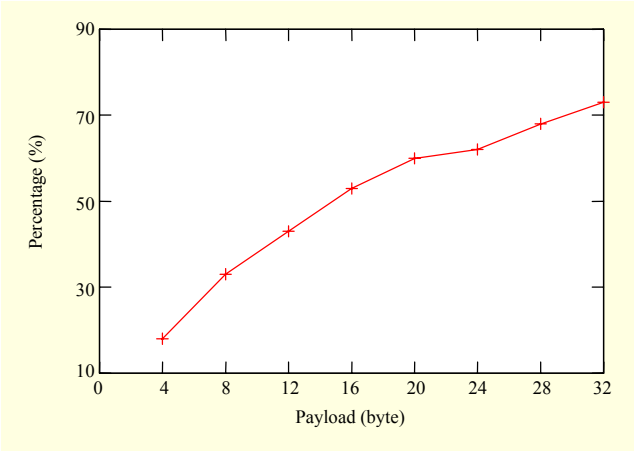

Fig. 16. MHP throughput improvement over baseline.

the data transfer time for the reference, whereas the data transfer time for the baseline takes 197% of the data transfer time for the reference at the maximum payload size.

Figure 15 shows a throughput comparison of the two approaches plotted against the sensorless reference. At a payload size of 4 bytes, the baseline throughput is 82 kbps compared to MHP's 98 kbps, with a difference of 15 kbps. As the payload size grows, the difference also grows. At a maximum payload size of 32 bytes, the baseline throughput is 203 kbps versus MHP's 352 kbps; this difference is 149 kbps. The throughput of the reference is 104 kbps at a payload size of 4 bytes and 400 kbps at the maximum payload size. The MHP throughput is 94% of the reference throughput at a payload size of 4 bytes and 88% of the reference throughput at the maximum payload size. The baseline throughput is 79% of the reference throughput at a payload size of 4 bytes and 51% of the reference throughput at the maximum payload size. Given that the throughput requirement for one camera application exceeds 256 kbps, the MHP is the only way that the system can meet the requirement without a hardware modification.

Figures 14 and 16 show the MHP improvements over the baseline. When the payload size is 20 bytes, both improvement curves slowly grow. At the maximum payload size of 32 bytes, the MHP data transfer time improvement over the baseline is 42%, which leads to a 73% throughput improvement. The reason there is an improvement at the lower platform-level over the SPI-level is simply due to Amdahl's Law: we are not improving the actual over-the-air transmission time, which occurs at 1 Mbps compared to SPI's 2 Mbps. Moreover, the transceiver itself incurs additional overhead per packet, including the formation of packet headers and a CRC, and it transmits a total of 40 bytes for every 32 bytes of payload. The MHP data transfer time on the node is 728 μs, whereas the MHP latency for the SPI data transfer is only 291 μs. This is because 60% of the platform-level latency actually goes into packetizing the payload and the over-the-air transmission. However, compared with the sensorless reference, the MHP achieves 88% of the theoretical maximum, whereas the baseline achieves 51%. In an MHP transaction, only 12% of the data transfer time is used for the MHP, while 88% of the data transfer time is used for wireless communications.

C. Energy Consumption

Figure 17 compares the energy consumption of the two approaches. The energy consumption is measured on the sensor node during one data transfer from the data source to the base station. When the payload size is 4 bytes, the energy consumed is 9.71 μJ for the MHP and 11.72 μJ for the baseline. The difference is 2.01 μJ. At the maximum payload size, the energy consumption is 21.55 μJ for the MHP and 38.05 μJ for the baseline. The difference is 16.50 μJ. Like the latency and throughput graphs, the energy consumption graph grows linearly. A comparison between the MHP energy consumption rate and the baseline is shown in Fig. 18. The energy consumption improvement over the baseline is 17% at a payload size of 4 bytes. When the payload size is 20 bytes, the energy consumption improvement over the baseline is 39%. At the maximum payload size, the improvement is 43%. The shape of the curve is also similar to that of the curve in Fig. 14 as the *power* consumption of both approaches is similar and the *energy* consumption depends on the data transfer time.

The performance of the MHP is better than that of the baseline in energy efficiency as well. Like the SPI data transfer case, as the payload size increases, the MHP improvements regarding data transfer time, throughput, and energy consumption over the baseline also increase.

V. Conclusion

This paper described the MHP for a fast and energy-efficient

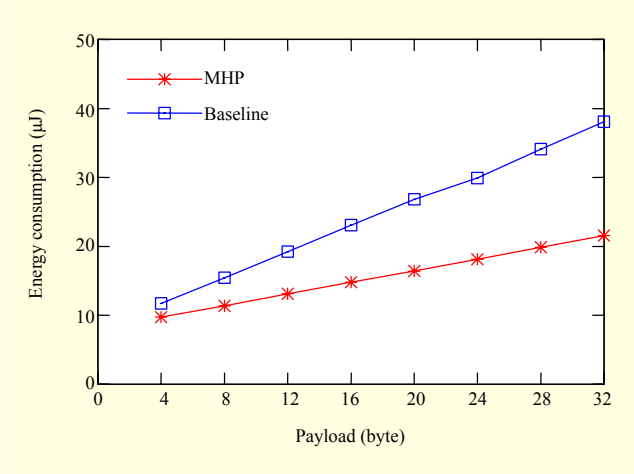

Fig. 17. Energy consumption.

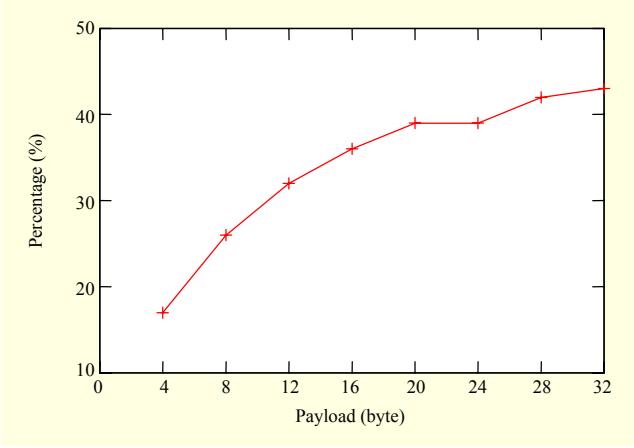

Fig. 18. MHP energy consumption improvement over baseline.

data transfer over SPI in wireless sensor platforms. This eliminates a well-known bottleneck, namely, the *double transaction* problem, which caps the throughput and latency at half of the theoretical peak performance. Our MHP breaks the single, fixed bus master assumption in conventional SPI architectures using a simple, temporary exchange of the master/slave roles. The protocol is simple to implement, incurs low overhead, and is applicable to many conventional, low-cost hardware architectures for sensor networks. It also eliminates an artificial data dependency on the MCU. Experiment results show that our implementation achieves significant improvement in latency (64%), throughput (176%), and energy consumption (64%), not only locally at the SPI-bus level, but, more importantly, globally at the end-to-end level, from a sensor device to the base station (73% and 43%). The reduced load can make available even more power management opportunities and time for useful computation, as well as significantly extend battery life for many emerging sensing applications.

References

[1] I.F. Akyildiz et al., "A Survey on Sensor Networks," *IEEE Commun. Mag.*, vol. 40, no. 8, Aug. 2002. pp. 102-114.

[2] A.Y. Benbasat and J.A. Paradiso, "A Compact Modular Wireless Sensor Platform," *Proc. IPSN*, 2005, pp. 410-415.

[3] Crossbow Technology Inc., "MPR-MIB Users Manual Revision B," June 2006.

[4] H. Dubois-Ferrière et al., "Tinynode: A Comprehensive Platform for Wireless Sensor Network Applications," *Proc. IPSN*, 2006, pp. 358-365.

[5] H. Lee et al., "Wearable Personal Network Based on Fabric Serial Bus Using Electrically Conductive Yarn," *ETRI J.*, vol. 32, no. 5, Oct. 2010, pp. 713-721.

[6] D. Lymberopoulos and A. Savvides, "XYZ: A Motion-Enabled, Power Aware Sensor Node Platform for Distributed Sensor Network Applications," *Proc. IPSN*, 2005, pp. 449-454.

[7] G. Mathur et al., "Ultra-Low Power Data Storage for Sensor Networks," *Proc. IPSN*, 2006, pp. 374-381.

[8] C. Park, J. Liu, and P.H. Chou, "Eco: An Ultra-Compact Low-Power Wireless Sensor Node for Real-Time Motion Monitoring," *Proc. IPSN*, 2005, pp. 398-403.

[9] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling Ultra-low Power Wireless Research," *Proc. IPSN*, 2005, pp. 364-369.

[10] P. Juang et al., "Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet," *SIGOPS Oper. Syst. Rev.*, vol. 36, 2002, pp. 96-107.

[11] G. Mathur et al., "A Storage-Centric Camera Sensor Network," *Proc. SenSys*, 2006, pp. 337-338.

[12] B. Schott et al., "A Modular Power-Aware Microsensor with >1000x Dynamic Power Range," *Proc. IPSN*, 2005, pp. 469-474.

[13] C. Park, P.H. Chou, and M. Shinozuka, "DuraNode: Wireless Networked Sensor for Structural Health Monitoring," *Proc. 4th IEEE Int. Conf. Sensors*, Oct. 2005.

[14] S. Hengsteler and H. Aghajan, "Wisnap: A Wireless Image Sensor Network Application Platform," *Proc. COGnitive Systems Interactive Sensors*, 2006.

[15] M. Rahimi et al., "Cyclops: In Situ Image Sensing and Interpretation in Wireless Sensor Networks," *Proc. SenSys*, 2005, pp. 192-204.

[16] T. Teixeira et al., "Address-Event Imagers for Sensor Networks: Evaluation and Modeling," *Proc. IPSN*, 2006, pp. 458-466.

[17] ATMel, "8-bit AVR Microcontroller with 16K Bytes In-System Programmable Flash," July 2006.

[18] J. Hill and D. Culler, "Mica: A Wireless Platform for Deeply Embedded Networks," *IEEE Micro*, vol. 22, Nov./Dec. 2002, pp. 12-24.

[19] Moteiv Corporation, "Tmote Sky: Datasheet," Feb. 2006.

[20] A. Banerjee et al., "RISE – Co-S: High Performance Sensor Storage and Co-Processing Architecture," *Proc. SECON*, 2005,

pp. 1-12.
[21] Shockfish SA., "TinyNode User's Manual," Rev 1.1, Nov. 2005.
[22] Texas Instruments Inc., "MSP430x15x, MSP430x16x, MSP430x161x Mixed Signal Microcontroller" Rev. E, Aug. 2006.
[23] ATMel, "AVR Butterfly." Available: http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3146
[24] Nordic Semiconductor, Available: http://www.nordicsemi.no/files/Product/data_sheet/nRF24L01_prelim_prod_spec_1_2.pdf
[25] IEEE Standard Department, "Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)," IEEE Standard for Information Technology, IEEE Std 802.15.4-2003, IEEE Press, New York, NY, USA, Oct. 2003.

Seung-mok Yoo received his BS and MS in computer engineering from Kyungpook National University, Daegu, Rep. of Korea in 1994 and 1996, respectively. He received his PhD in electrical and computer engineering from UC Irvine, Irvine, CA, USA, in 2007. He was a researcher at the Agency for Defense Development, Rep. of Korea from 1996 to 2001. He is currently a senior engineer at ETRI, Daejeon, Rep. of Korea. His research interests include node architecture design, MAC and routing protocol design, and distributed real time system design and implementation in wireless sensor networks and embedded systems.

Pai H. Chou is an associate professor in electrical engineering and computer science at UC Irvine, Irvine, CA, USA and in computer science at the National Tsing Hua University, Hsinchu City, Taiwan. He received his AB in computer science from UC Berkeley, Berkeley, CA, USA, in 1990 and his MS and PhD in computer science and engineering from the University of Washington, Seattle, WA, USA, in 1993 and 1998, respectively. His research interests include wireless sensing systems, low-power design, energy harvesting, and system synthesis. He is a recipient of the NSF CAREER Award.