www.s2is.org
International Journal on Smart Sensing and Intelligent Systems
ISSN 1178-5608

FEATURES, OPERATION PRINCIPLE AND LIMITS OF SPI AND I^{2}C COMMUNICATION PROTOCOLS FOR SMART OBJECTS: A NOVEL SPI-BASED HYBRID PROTOCOL ESPECIALLY SUITABLE FOR IoT APPLICATIONS

P. Visconti [1], G. Giannotta [2], R. Brama [3 #], P. Primiceri [4], A. Malvasi [5 #], A. Centuori [6 #]
Department of Innovation Engineering, University of Salento, 73100, Lecce, Italy
[#] CMC Labs -A Division of CMC S.r.l.- C.da Pagliarulo sn – 72012, Carovigno (BR), Italy
Emails: paolo.visconti@unisalento.it [1], gmgiannotta@gmail.com [2], r.brama@cmclabs.com [3], patrizio.primiceri@unisalento.it [4]. a.malvasi@cmclabs.com [5], a.centuori@cmclabs.com [6]

Submitted: Feb. 5, 2017 ***Accepted: Apr. 15, 2017*** ***Published: June 1, 2017***

Abstract – ***The Internet of Things (IoT) is an expression, sometimes abused by companies given the absence of an unambiguous meaning, that indicates the upcoming evolution of Internet as it has been known so far. In fact, all objects will have network capabilities which will be exploited to overcome, in certain situations, human intervention. Thanks to the direct cooperation of new class of devices, aware of their operating scenario and interconnected in subnetworks, our life style will be strongly enhanced and simplified. IoT, however, is not yet the "El Dorado" of technology, capable of revolutionizing everyday life: some aspects and open issues have to be carefully analyzed. The huge complexity of this new technology forces companies to select a specific research field: for this reason, they focus only on some features that an IoT device should have to guarantee fulfillment of requirements. In this context, this research work concerns an analysis of features, operation principle and limits of SPI and I^{2}C communication protocols followed by the proposal of a new hybrid protocol suited for embedded systems, named FlexSPI, thought as an evolution of the classic SPI. Thanks to a robust software architecture, it is able to provide many features that can be used by smart objects to enhance their capabilities. In this way, sensors and actuators or, more in general, subsystems, can quickly exchange data and efficiently react to malfunctioning; moreover, number of devices on bus can be safely increased even while smart object is performing operations.***

Index Terms: IoT, embedded system, SPI and I^{2}C communication protocols, smart objects, FlexSPI.

I. INTRODUCTION

The first time the expression "*Internet of Things*" has been used, according to [1], was in 1999, during a presentation for *Procter&Gamble*. The idea, back then, was suggesting that it would have been very groundbreaking to have data gathered and processed without any human intervention, causing a reduction in wastes and costs. To do that, however, it would have been necessary to make computers capable of collecting pieces of information about the physical world by themselves. In the academic world, it is difficult to find a proper and shared definition of this expression; according to [2], this lack is due to the composition of two terms: while the first one, *Internet*, suggests a network-oriented vision, the second one, *things*, focuses the attention on generic objects that should be integrated in a common framework. However, when we put these two words together, something completely new is obtained; semantically speaking, *Internet of Things* can mean a "*world-wide network of interconnected objects uniquely addressable, based on standard communication protocols*" [3]. This definition implies a tremendous number of objects, part of everyone's life, capable of storing pieces of information, exchanging them and behaving according to them. In the wake of this explanation, another interesting definition is given by CISCO in [4], where "*IoT is simply the point in time when more things or objects were connected to the Internet than people*", thus placing between 2008 and 2009, as can be seen in figure 1, the birth of Internet of Things.

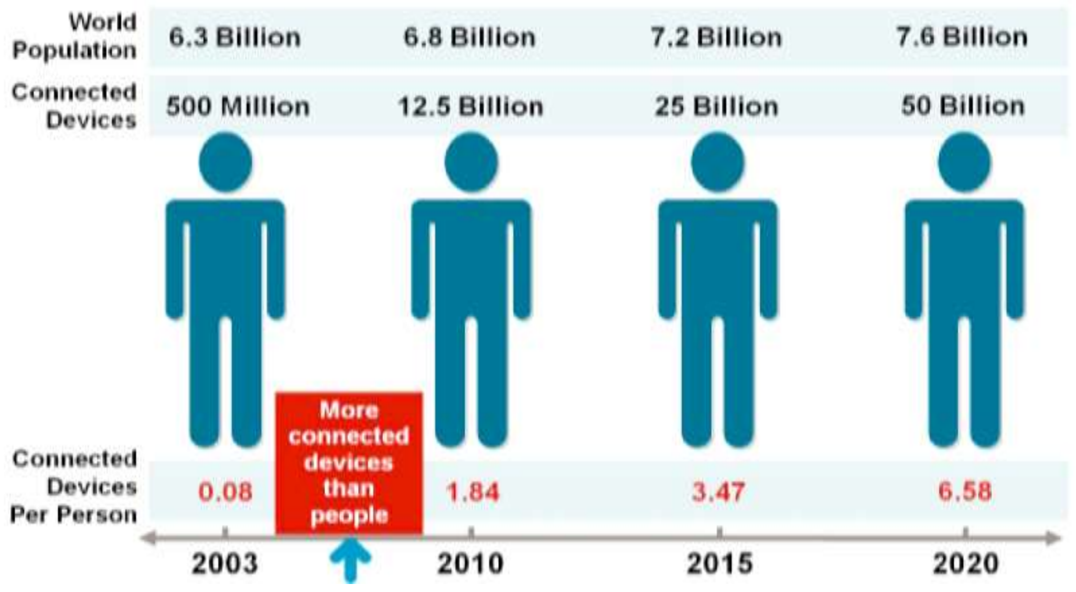


Figure 1. Number of connected devices and human population (Source: Cisco IBSG, 2011).

In [4], it is stressed how the *Internet of Things* is the first real evolution of common Internet: by making it sensory and aware of the physical world, it will improve the way people live, making it easier and more efficient. IoT, in fact, will allow the creation of smart environments thanks to the cooperation of several devices that share data and are aware of the context in which they are used. It is straightforward to observe that designing applications fitting this vision brings a new series of challenges and problems that will be discussed later. The birth of the *Internet of Things*, as described in [5], has been made possible thanks to many factors:

- ✓ the progressive miniaturization of sensors and chips that made easier the integration of several devices on a smaller area;
- ✓ the growth of internet connectivity, allowing devices to create a network in different circumstances;
- ✓ the use of cloud storage as a tool for sharing data and the development of sophisticated analytic models to extract information from them.

Although IoT may be seen as something still far from us, many organizations are already using IoT-based technology for their products. In [5], the experience of a private company like Pirelli is reported: by collecting real-time data from their products, it is possible for fleet managers to efficiently manage fuel costs and increase safety. Another example comes from a public institution, the Boston Police Department, which exploits data gathered from cameras and sensors to efficiently track people or to calculate an evacuation route in case of emergency. IoT technology is changing business models and strategies of the companies that are using it, and it is enhancing services already offered. Again in [5], it is reported the results of a survey on several companies using IoT applications. Results, shown in figure 2, describe a situation in which companies have various kind of advantages in adopting IoT technology: in many cases, the most remarkable result is the possibility of collecting data regarding how a product has been used exploiting this knowledge to adapt and optimize products according to customers.

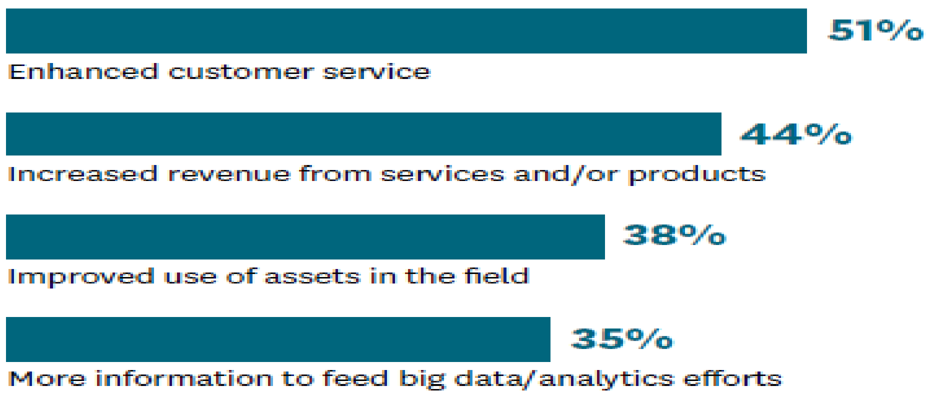


Figure 2. Benefits observed from companies that adopt IoT-based applications.

Despite these results, research on different facets of this technology must go on, in order to satisfy the several requirements that IoT devices should have. This progress can be obtained through many investments, but sometimes stakeholders are not keen to use their money on the

IoT; an emblematic example is given by [6], a report about U.S. Dept. of Defense (Dod) and its usage of the Internet of Things. This paper reports that in the military environment, there is always some reluctance to invest today's budget in a possible future advantage: the estimated long-term savings can be not enough to develop and deploy new technologies and applications, and this is exactly the case of IoT ones. There is also a certain concern regarding the reduction of the human role in military applications: people are more secure if there is a human fallback when technology fails. Despite being very military oriented, this paper stresses that these kind of concerns are not limited to the DoD but can be observed, together with many others, among several companies. Given the benefits for companies and their hesitations on the IoT-based technology, it assumes a great importance to understand which are the features of the devices that should form the Internet of Things, their requirements and the open challenges that hold back stakeholders. This investigation can help to clarify which progresses have been made in this field and what must be done in order to obtain a technology capable to revolutionize the human way of living.

II. FEATURES AND CHALLENGES OF THE INTERNET OF THINGS

A huge flow of data can be generated by rethinking common objects with sensors and connection capabilities, creating an interacting network made by humans and things. In order to do that, it is important that these devices satisfy some requirements and specifications: winning some of these challenges will contribute to increase the interest toward this technology and to attract resources. For this reason, research is investigating different aspects of the IoT architecture: given the complexity of the problem, in fact, it is important to separate and focus on various aspects. Of course, cooperation is mandatory to fully realize the vision proposed by IoT and, therefore, to make the full exploitation of this technology possible. Some of the requirements which IoT devices should have, are described following:

- ✓ ***Scalability***: main purpose of IoT is putting several sensors on objects and connecting them. In order to handle the great amount of generated data, scalability in every layer of network is required; while cognitive radio and spatial reuse can help at physical layer, in upper layers, adoption of 6LoWPAN and IPv6 have been proposed. A different approach is based on exploiting the processing capabilities of devices to decrease generated traffic.
- ✓ ***Affordability***: right now, returns from investments in IoT are not enough to motivate investors. For this reason, companies are trying to exploit mobile phones to collect data since these devices, used by the majority of population, are already full of sensors and

always connected to Internet, thus providing a first generation of applications for IoT. This test bench can be useful to show IoT technology potential to motivate stakeholders.

- ✓ ***Context-awareness***: IoT vision will be eventually reached when it will be possible to derive a physical context through data aggregation coming from different devices. Knowing when, where and why something happened will make possible to react properly to different events; this aspect implies need of a huge development in automation processes.
- ✓ ***Cooperation***: creation of applications should not be made in a vertical and independent way; it would be better to share a common platform that promotes software re-use. Since developing IoT applications is a complex process that requires knowledge of different domains, some kind of cooperation among companies is necessary. In this sense, some frameworks have been created with the aim of separating concerns for every stakeholder.

A list of device capabilities allowing them to properly interact in a network is shown:

- ✓ ***Communications***: devices should not only communicate among themselves, but also with human beings; some proper interfaces must therefore be built.
- ✓ ***Auto identification***: different devices with different capabilities will take part in the creation of a network. A self-describing interface for each device is necessary in order to optimize the management of required tasks.
- ✓ ***Memory and tracking***: in order to maintain settings and historical data, devices must have a persistent memory and they should also be able to properly manage it; the access to cloud storage can be particularly useful too.
- ✓ ***Reasoning & learning***: behavior of interconnected devices should not be static but dynamic. By keeping memory of the past, objects should be able to analyze data thus autonomously performing some actuation. However, the possibility of a human overwrite of some actions must always be possible.
- ✓ ***Maintainability***: upgrading or booting a device should be made in a complete automatic way, although human intervention should be expected.
- ✓ ***Survivability***: adding an object to the network should not cause the collapse of the network itself. The network should also properly handle the failure of a device.

Companies and researchers are already working on these aspects, obtaining important results that make IoT technology more and more widespread. However, there is a different class of aspects and problems that slow down the development of IoT applications and, at the moment, some of them are particularly challenging. In [4] and [6], some of the barriers, which have the potential to stall the progress of IoT, are reported. It must be pointed out, however, that these challenges, analyzed below, cannot be considered insurmountable: "*given the benefits of IoT, these issues will get worked out. It is only a matter of time*" [4].

- ✓ ***Deployment of IPv6***: the availability of IPv4 addresses ended in February 2011. Although this fact had no impact on the general public, it is an issue for IoT: every sensor, in fact, is supposed to have a unique IP address to be uniquely identified. By using more bits to represent an address from 32 to 128, IPv6 ensures a constant availability of identificators. Other benefits of IPv6 include an easier management of networks due to auto configuration and better security features [7].
- ✓ ***Energy harvesting:*** to reach its maximum potential, IoT nodes should be self-sustaining, meaning that they should be able to recharge their batteries by themselves. It is therefore essential to develop new and efficient ways to harvest energy from natural elements such as solar light and wind; devices should also implement some power management procedures in order to maximize their life cycle [8].
- ✓ S***tandard***s: in an IoT vision, sub-networks are tailored to different applications, thus resulting in a heterogeneous global network; packets exchanging among devices should also be able to reach different kind of networks. Companies should agree on a shared standard to ensure the integrity of communications among different kinds of objects.

Another class of problems regarding IoT deals with security and privacy inside networks: due to the confidential content of data exchanged among devices and applications tackled by IoT, network security is of paramount importance. Only by ensuring safety of IoT devices and their respect of users privacy, stakeholders will be pushed to support more and more applications, making possible IoT vision. In many applications, especially military ones, security is vital: an enemy stealing information about troops position, as example, may cause a serious damage. Here some IoT security problems, derived from military needs, but useful for any application.

- ✓ ***Networking***: having a great ubiquitous number of devices interconnected means giving to hackers several potential points of access to the network. Using a strong data encryption and multiple authentication mechanisms can be a solution but it is proving being very challenging in IoT devices: the limited computational capacity and power savings precautions make difficult to implement strong encryption keys.
- ✓ ***Electronic warfare***: IoT devices communicate through wireless RF channels: it is not too complicated to block signals breaking link in the network. Signals can be used as beacon to locate transmitter threatening troops safety in military or users privacy in civil applications.
- ✓ ***Automation***: devices capable of automatic actuations according to sensed data transform

a cyber threat in a physical one; this kind of consequences was tested when two hackers were able to take control of a moving car from their own apartment, showing the degree of damages that can be provoked by exploiting vulnerabilities in the car's firmware.

These aspects have consequences for the privacy of users: IoT attacks do not need that the intruder is physically present near the target to collect information. The classic attacks on networks such as eavesdropping, traffic analysis and data meaning, are still possible and effective; however, given the type of data devices are supposed to transmit, these intrusions can be a serious violation of privacy. Research is active in investigating techniques that can be used to build a safer IoT architecture; the success in granting those features in IoT devices would make possible to overcome companies concerns regarding the IoT security.

- ✓ ***Authorization and access control***: main problem in an IoT scenario is developing access control rules that devices can easily create, understand and manipulate. This aspect is correlated with identification mechanisms creating a central point in research activities.
- ✓ ***Trust management***: since IoT environment is dynamic and therefore uncertain, how can a device trust another? We could identify two different kinds of trust: not only in the interaction between objects, but also in the system from a user perspective. Focusing on the interactions between objects, if trust evaluation could be automated, autonomous, decentralized and transitive, it would overcome constraints introduced by devices [9][10].
- ✓ ***Privacy metric***: sharing collected data between multiple entities makes hard to create a platform that ensures the conformance to privacy regulations. For this reason, creating a privacy metric would allow data owners to properly evaluate a privacy breach and decide if a certain application can be trusted [9].
- ✓ ***Limited access***: in [6], a hierarchical approach has been proposed to enhance complex networks security. This approach limits number of nodes that an attacker may access from any given entry point. In this way, performances of IoT applications would be drastically reduced since there would be a lowering in the data amount that the network can collect.

III. REALIZING IoT: RFID DEVICES vs SMART OBJECTS

Despite problems and challenges still open, several IoT applications have been already developed, proving the great benefits that this technology would have on everyone's life. As stated before, the idea is giving to objects sensing and communicating capabilities in order to create a network that exchanges data and reacts to environmental changes. Two possible ways for obtaining this network are here discussed, focusing on how their features fit IoT requirements. The first one is the adoption of RFID technology; RFID, acronym of Radio-Frequency IDentification, is a technology that exploits the electromagnetic field to identify and track objects. A model of its architecture is shown in figure 3.

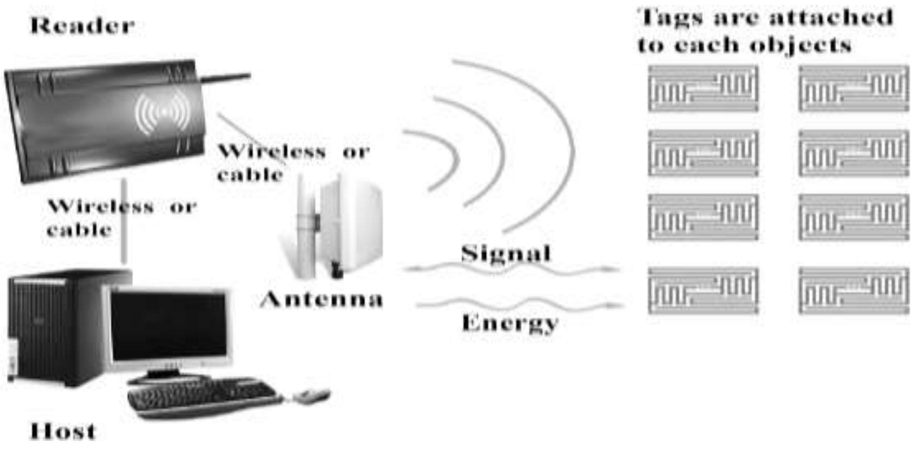


Figure 3. Typical RFID architecture.

This system is composed by a RFID tag, active or passive, that sends the stored information via the electromagnetic field emitted by the reader, a reader, fixed or moving, that receives data sent from tag, eventually modifying information stored in it and other elements used to provide the necessary tools to decode the information acquired from the reader and eventually to send obtained data on cloud. This technology has encountered a great success in the identification market, due to low cost of tags and in many healthcare applications to track patients medical history or in the logistics of deliveries. RFID tags are increasingly widely used to identify pallets, cartons, library books and passports, among other things, and the ability to tag real-world objects is just starting to be persuasive; combined with Wireless Sensor Networks (WSNs), this technology can play a central role in the identification aspects of IoT technologies. Although very promising, this approach shows some limitations when applied to the core concepts of IoT; some of these aspects are pointed out following:

- ✓ it is not well suited for fast evolving context, thus limiting the flexibility of applications;
- ✓ it has poor sensing capabilities, although it can be integrated with WSNs;
- ✓ it could work fine if the number of nodes is not high;
- ✓ it can be a real option when a particular processing of collected data is not required.

If we consider those aspects together with requirements described before, one should note that choosing RFID could not be enough to create vision proposed by IoT. A valid alternative to RFID are smart objects, whose model can be seen in figure 4; it is a single platform able to create, process and share information based on data acquired from sensors and that is also able to communicate wirelessly with other smart objects.

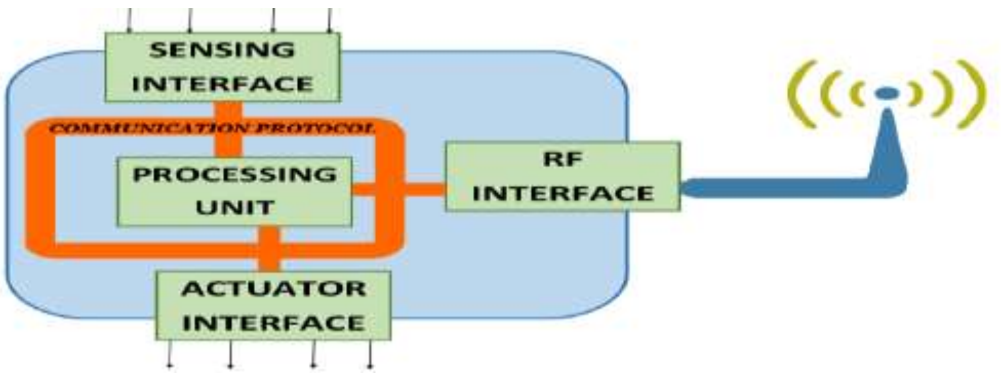


Figure 4. Basic architecture of a smart object.

Each of the functions that a smart object possesses is handled by a particular subsystem that uses a dedicated microprocessor to coordinate and efficiently manage their tasks. Thanks to the progressive miniaturization of the single components, the integration among subsystems deeply different from each other, can be obtained directly at hardware level; in this way, the device itself can do more autonomous processing on collected data, decreasing workload to human operators. This architecture can be very flexible, if it is provided a mechanism to plug other subsystems so enhancing capabilities of smart objects. Given this definition, any object can become smart by simply adding missing components and when smart objects connect their functionalities, they trigger their full potential. This aspect is strengthened in [11]: "*smart objects are everyday physical objects enhanced with sensing, processing and communication abilities*". While RFID technology uses point-to-point communication between tag and reader, smart objects create ad-hoc networks in which they communicate each others while they keep monitoring the assigned targets. They are typically arranged in clusters, with peripheral nodes interacting with each others and with a central node whose task is to deliver the information outside the sub-network [12]. In this way, it is possible to increase the network life cycle, since only one node has to perform power-intensive communications. This is a typical approach used to organize WSNs; strategies in figure 5 have merit to be less power consumptive than classical routing, although a trade-off between latency and energy efficiency has to be found.

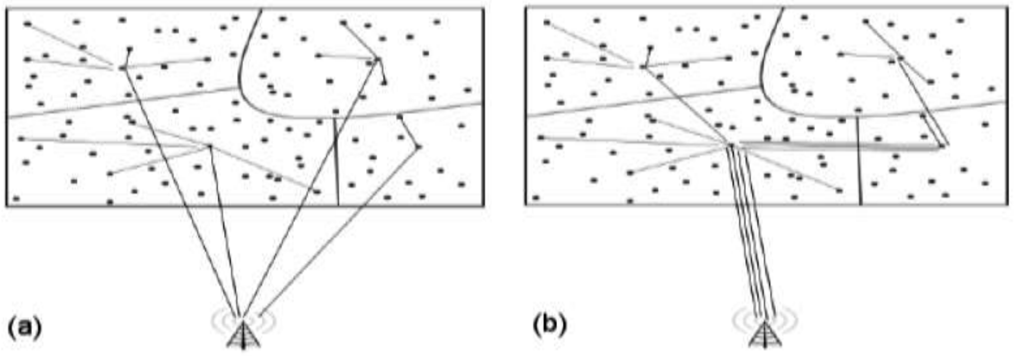


Figure 5. Examples of hierarchical routing strategies: single (a) and multi-hop (b) approaches.

An object, in order to be classified as smart, should satisfy a certain number of requirements:

- ✓ ***Awareness***: it must have the ability to sense and understand events and human activities in its physical domains.
- ✓ ***Identification:*** it must be uniquely identified inside the network; the adoption of IPv6, as said before, plays an important role.
- ✓ ***Actuation***: using self-collected or third party communicated information, it shall take decisions on itself or on other devices.
- ✓ ***Cooperation***: it should be able to inform other smart objects in the network regarding its capabilities useful to perform network adaptation according to available capabilities.
- ✓ ***Human interaction***: as for all applications, an appropriate end-user interface enabling human interaction with the network must be efficiently designed.

Since a smart object is defined as the aggregation of several subsystems, all these results can be obtained if communications between subsystem themselves is efficient; it is important to analyze the standard communication protocols, aiming to extract features suited for efficient smart objects and to propose a valid alternative that could satisfy the desired *IoT* requirements.

IV. PROTOCOLS COMPARISON FOR COMMUNICATIONS INSIDE A SMART OBJECT

In this section, two of the most famous communication protocols in the electronics world, I^{2}C and SPI, will be discussed, stressing their strong points and limitations. This analysis is necessary to understand the behavior of each standard when applied to a smart object, leading to the necessity of conjugating the positive sides of both I^{2}C and SPI. In this way, it is possible to propose a new communication protocol tailored for smart objects.

a. I^{2}C protocol analysis

The *Inter Integrated Circuit* (I^{2}C) protocol, here described according the specification file [13], was developed by Philips in 1982 and it is a serial, single-ended bus, with multi-master support and typically used to connect low speed devices. This protocol had several upgrades with similar buses, like SMBus, to meet requirements of newer and more complex devices on the market. The bus, as shown in figure 6, is obtained with two bidirectional lines connecting the devices, masters and slaves, in an open-drain way: two pull-up resistors are used to mark high the lines recessive state. The two lines that form I^{2}C bus have the following functions:

✓ Serial Data (SDA) to exchange data among the devices thus making I^{2}C an half-duplex bus.

✓ Serial Clock (SCL) used by a master to impose the transmission rate; it can be controlled also by slaves in some advanced procedures.

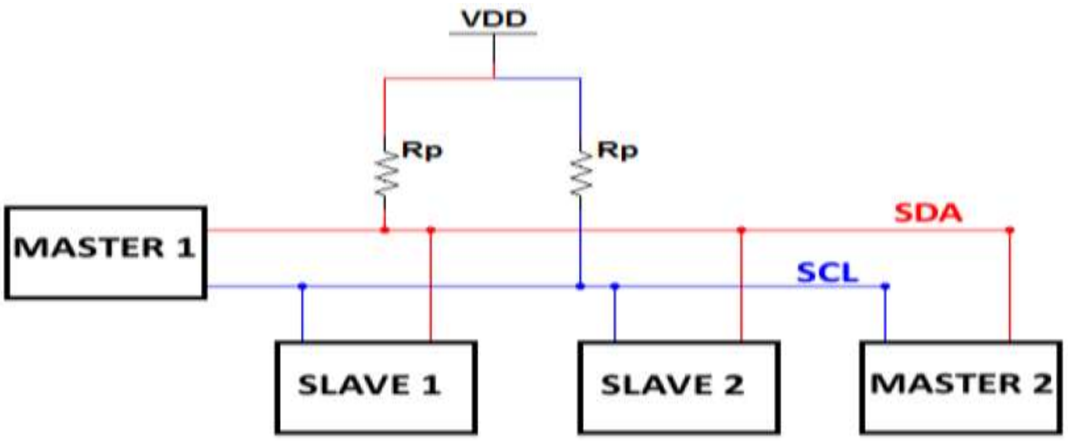


Figure 6. Example of a typical I^{2}C bus.

Since the bus is completely shared, it is possible, for devices, to receive not only unicast transmission but also broadcast messages: for this reason, whenever master wants to write or read data from a particular slave, it will address it first. Addressing bits were originally seven, extended to 10 with the latest reviews to increase supported maximum number of connected devices. I^{2}C bus supports multiple masters on the same bus too, but a proper conflict-solving algorithm has to be implemented. When a master addresses a slave, it will wait for an acknowledgment before executing the desired operation: an ACK/NACK approach is actually adopted to verify result of the transmission of every byte. The frame structure is simple: after starting bit, used to signal the beginning of a transmission, and the address of the slave, the master defines the operating mode, read or write, through a single bit. After having received the acknowledgment from the addressed slave, the appropriate session will begin, with an ACK or a NACK transmitted from the receiver after every byte; an ending bit will be the signal that the transmission has ended. The structure of a frame in the two supported operating mode is reported in figure 7; although non reported, it is also possible to exchange combined messages, i.e. when a master addresses at least two reads and/or writes to one or more slaves.

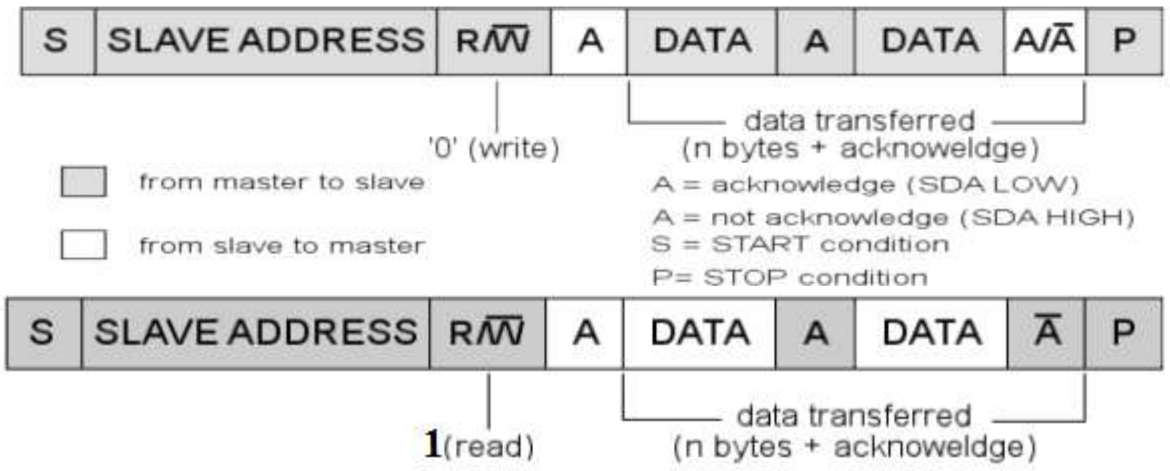


Figure 7. I^{2}C frame format when the master is writing data (a) and when is receiving data (b).

The typical communication session begins when the SCL line is high and SDA is pulled down, configuration that identifies the start bit. In order to transmit the following bits, data are prepared while SCL is low and, as soon as SCL return to its logic high level, the receiver reads the value from the SDA line; after a short delay, the master pulls down again the SCL line to prepare another bit, and so on. A rising edge of the SDA line while SCL is still high represents the stop bit (figure 8). Analyzing this protocol from a circuital point of view, the open-drain connection of devices, detailed in figure 9, implies that they can only pull down the two lines, while they have to wait pull-up resistors action for the logic level to return high.

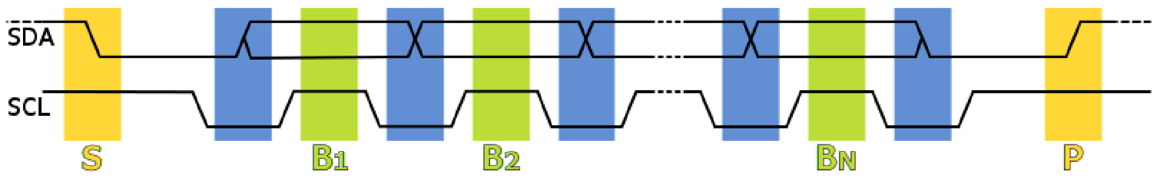


Figure 8. Example of I2C timing diagram.

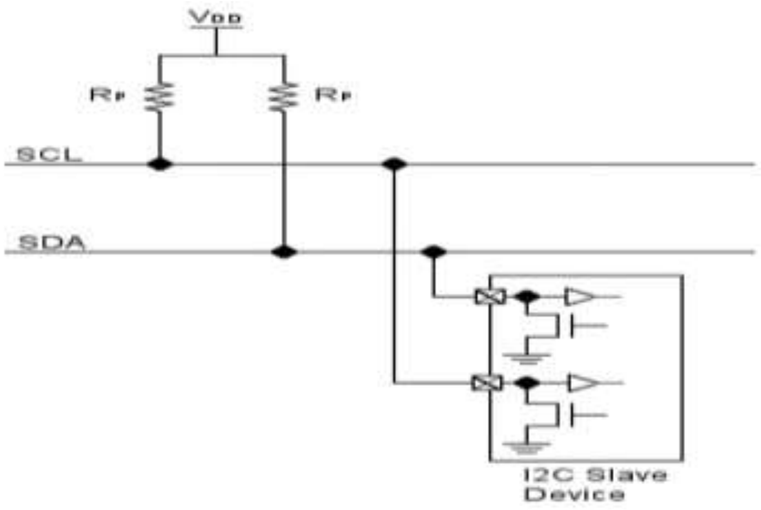


Figure 9. Open-drain connection in the I^{2}C bus.

Two procedures can be adopted by an I^{2}C bus, for extending its functions:

- ✓ ***Clock stretching***: an addressed slave can keep low the SCL line until it properly processes the received byte, forcing the master to wait.
- ✓ ***Arbitration:*** since I^{2}C supports multiple masters, an arbitration rule is necessary. In this protocol, arbitration proceeds bit by bit and the rule is deterministic: the first master that produces a one when the other produces a zero loses the arbitration.

This bus is used in many applications where addressing is necessary: the easiness of the connections and the possibility to define the protocol details via software reduce the design time of an application. However, this protocol has some limitations: the clock stretching procedure has some bad side effects since it reduces the bit-rate in fast devices. Another limitation is in the addressing: seven bits are not enough in systems where a large number of devices are connected, thus restricting the use of this protocol to small systems. At last, I^2C forces bus designers to face a trade-off between speed and power consumption, caused by the sizing of the pull-up resistor. The total bus capacitance, in fact, is given by the number of connected slaves: given a desired rise time of the line, and therefore a bus speed, more devices are connected then smaller the resistance must be, being $\tau = RC$ and therefore $R=1/\tau C$. On the other side, by decreasing the pull-up resistance, the power consumption of the bus increases, since pulling down a line means closing the path between the power source and ground, thus having a current inversely proportional to the pull-up resistor. To decrease the bus capacitance and therefore increase throughput, buffers and multiplexers can be used, dividing the bus in smaller sectors; however, with this approach, the circuital complexity increases.

b. SPI protocol analysis

The *Serial Peripheral Interface* (SPI) protocol, developed by Motorola and now a de facto standard is, as described in the specifications [14], a duplex synchronous, serial communication bus based on a master-slave approach. Four lines connect the master to each slave, whose functions are described below:

- ✓ ***Serial clock (SCL):*** generated by the master, it gives the timing of data exchange.
- ✓ ***Master Output Slave Input (MOSI):*** used to transfer data from the master to the slave.
- ✓ ***Master Input Slave Output (MISO):*** used to transfer data from the slave to the master.
- ✓ ***Slave Select (SS):*** also known as *Chip Select* and usually active low, it is used by the master to address and activate a particular slave in order to start a communication session.

A representation of a SPI bus can be seen in figure 10; while three lines are shared among all the devices on bus, every slave must be connected through a dedicated $\overline{CS}$ line; for this reason, in its standard configuration, SPI is considered a 3+n bus with n number of slaves. When a master wants to begin a communication session, it has to configure clock frequency according to the slave capabilities; the master then selects the slave by lowering the appropriate $\overline{CS}$ line.

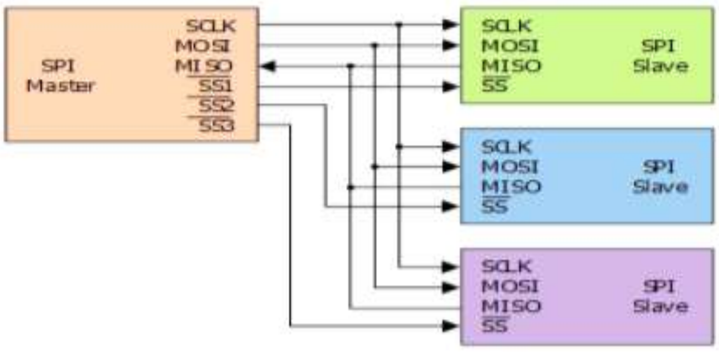


Figure 10. Typical configuration of a SPI Bus.

Data transmission involves two shift registers, typically 8-bit long as reported in figure 11. In almost every application, data are shifted out starting from the MSB and, after every clock cycle, the remaining bits are transmitted. When the register is full, the content of the received data is moved to an appropriate buffer, letting the device ready to receive another byte. When the transmission is completed, the master stops toggling the clock line and deselects slave by rising the CS line. In many microcontrollers, clock generation is tied with the transmission buffer of the master, which eventually must be loaded with dummy data: developers must take into account this aspect when designing a firmware.

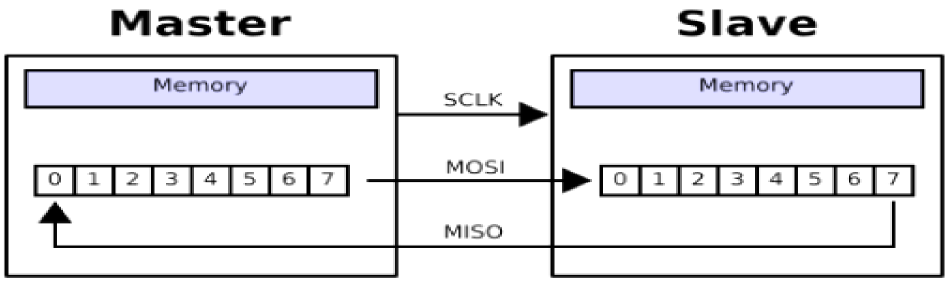


Figure 11. Hardware setup exploiting the two shift registers in an SPI bus.

Among the parameters set by the master, *clock polarity* (CPOL) and *clock phase* (CPHA) are used to set the idle state of the clock and the sampling edge. According to their values, four possible behaviors are possible, affecting not only the sampling edge but also the output timing and obtaining the configurations reported here below:

✓ When CPOL=0, the base value of the clock is 0, meaning that active state is 1; therefore:
- if CPHA=0, bits are sampled on the clock rising edge and in output on the falling edge;
- if CPHA=1, bits are sampled on the clock falling edge and in output on the rising edge.

✓ On the other hand, when CPOL = 1, the clock is active when it is 0; therefore:
- if CPHA=0, bits are sampled on the clock falling edge and in output on the rising edge;
- if CPHA=0, bits are sampled on the clock rising edge and in output on the falling edge.

The combinations of these values produces different timing diagrams reported in figure 12.

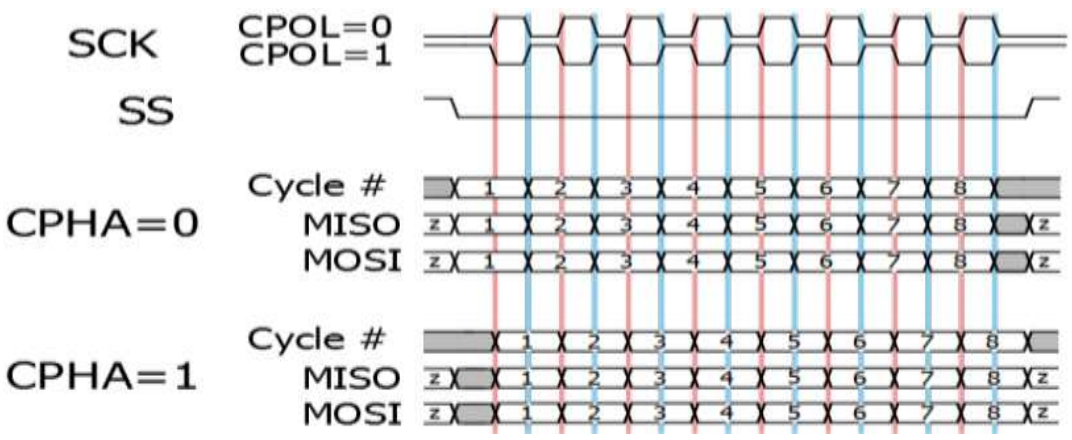


Figure 12. SPI timing diagrams with all the possible clock setups.

Given its simplicity and customizability, SPI protocol can be adapted to different applications: for example, some devices have minor variances from the classical CPOL/CPHA modes, providing further settings. Some chips, instead, combine MOSI and MISO lines into a single data line (SI/SO), obtaining the so-called *three-wire* signaling. Another example of its flexibility, in the connection of the devices, is the daisy-chain configuration that uses only one $\overline{\text{SS}}$ line regardless the number of slaves, as can be seen in figure 13. Bit-rate, as expected, decreases, since data are transmitted to a particular slave by shifting them through the intermediate devices. It must be pointed out, however, that the clock speed can reach several Mhz, thus limiting the impact of the slowing down caused by this configuration.

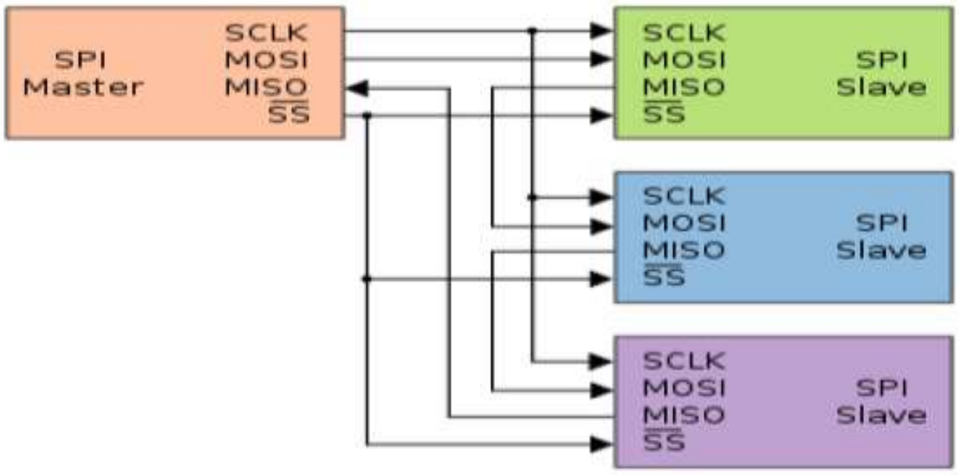


Figure 13. Daisy-chain configuration for an SPI bus.

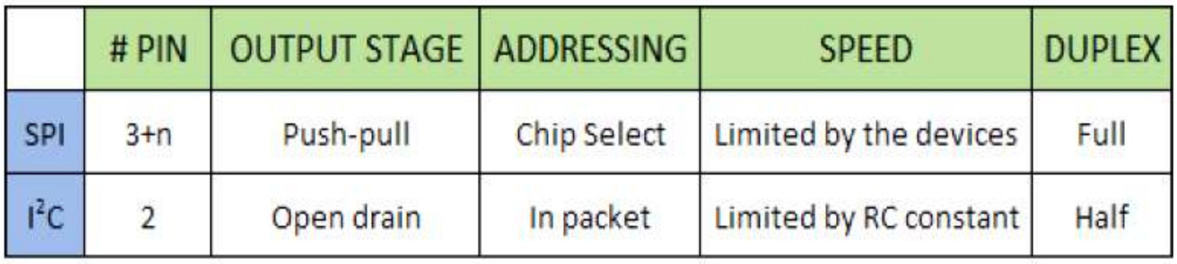
SPI is used to exchange data with a great variety of devices such as sensors, cameras, memories and LCD displays. The full-duplex support makes SPI particularly efficient in applications with a single master and a single slave: this possibility is used to obtain an efficient and fast data stream for applications such as digital audio, digital signal processing, or communications channels. An important feature is the push-pull output stage with two transistors, n-type and p-type, one sourcing current to load from a positive power supply and the other sinking current from load to ground. This output stage is responsible for the high bus speed up to tens of Mhz: the device actively creates the desired logical state by enabling the appropriate transistor, having therefore very short rise and fall times. For this reason, the bus speed is mainly dependent only on the capability of connected devices. Although the many benefits, simplicity of SPI brings some disadvantages: no form of sophisticated addressing and flux control is possible, a multi-master support is not included and it is not possible to hot-plug devices. At last, number of used pins increases linearly with slaves making circuital layout more complicated; therefore number of communicating devices is greatly limited.

V. OVERALL EVALUATION OF SPI AND I^{2}C PROTOCOLS AND PROPOSAL OF THE NEW HYBRID SOLUTION

The overview on two of the most used communication protocols brings some considerations regarding how well they are suited for an IoT application; moreover, this analysis can be used to understand what can be done to improve performances and possibly obtaining a standard that combines the advantages of both of them. I^{2}C has many of the features required by IoT paradigm, with an abstraction layer that provides flexibility and different services. However, although improved with the latest reviews, it still lacks of important features and, above all, its strict connection between number of devices and maximum bit-rate forces to create restricted smart objects. At last, the purpose of limiting energy consumption fails due to the open-drain connection with pull-up resistor.

On the other hand SPI, based on a push-pull output stage, is a protocol less power consuming and with a throughput depending only on the capabilities of the devices on bus. However, its simplicity causes the lack of features necessary to let subsystems communicate in an efficient way, making difficult its implementation in a smart object. The main features of these two protocols, compared in some of the most important aspects, are reported in figure 14. It must be pointed out that, while SPI is a completely physical protocol, I^{2}C is developed with a more complicated software structure; this aspect can be seen as advantage or an obstacle depending on designers preferences. The possibility of conjugating benefits of these two protocols would lead to a communication standard particularly suited for smart objects and IoT applications. A fixed number of used pins would simplify the development of subsystems, making easier to increase capabilities of a smart object; by assigning to every device a long address, it would be possible to avoid multiplexers use decreasing circuital complexity. The protocol should also have capability of recognizing new devices connected to bus; to unlock these features, it is clear that SPI or I^{2}C by themselves are not able.

This analysis leds to the proposal and definition of FlexSPI; built on top of a classic SPI bus, it allows to completely share all its lines among different devices by leaving slaves in high impedance until right before a communication session. By proper addressing and possibility for slaves to signal the need of a communication session, it is possible to obtain an high speed four wired protocol whose power consumption is dependent only on the devices themselves. These features give the possibility of creating a series of procedures thought to obtain a self-sustaining bus particularly suited for smart objects. Since this protocol is thought as a MAC layer over the legacy SPI bus, its implementation requires a further knowledge of advanced software architectures and how to combine them with the available modules of a device [15].



	# PIN	OUTPUT STAGE	ADDRESSING	SPEED	DUPLEX
SPI	3+n	Push-pull	Chip Select	Limited by the devices	Full
I^2C	2	Open drain	In packet	Limited by RC constant	Half

Figure 14.Comparison between I^2C and SPI communication protocols.

VI. OPERATION PRINCIPLE, FUNCTIONALITIES AND HARDWARE SOLUTIONS OF THE NEW FLEXSPI COMMUNICATION PROTOCOL

The purpose that has led to the ideation of this communication protocol is obtaining a fully shared SPI bus, with a fixed amount of wires, without renouncing to the advantages of a push-pull output stage and obtaining an architecture capable of great flexibility. All the four signals of a classic SPI protocol are entirely shared by the slaves on the bus: when a master wants to communicate with a particular device, it will perform an addressing at packet level. All slaves will download the transmission and check if they are the addressed receivers: if so, they will continue to process the packet, otherwise it will be discarded. Thanks to this approach developed in the new FlexSPI protocol, it is possible to provide not only unicast addressing but also multicast and broadcast messaging. It is possible to obtain this behavior by redefining the role of *Chip Select* in the communication session: instead of just bytes, this signal is used to window messaging sessions, to wake up slaves and let them know that a transmission is about to begin. Further transmissions will be handled by slaves according to some built-in finite state machines, updated according to the content of the previous packet.

This approach shows no problems when applied to the SIMO signal but can provoke a serious number of dangerous conflicts when referred to SOMI if an arbitration rule is not applied; moreover, slaves are allowed to use this line to asynchronously request the master attention. Potential malfunctions are prevented by configuring the SOMI pin, when not employed in a data transfer as a high impedance input, while the master uses a pull-up resistor to mark the line in a recessive state. As soon as the master needs to receive data from a particular slave, it explicitly gives the possession of the SOMI line and disables the pull-up resistor, granting only to that device the right to occupy the channel with its transmission. An example of a possible bus using FlexSPI is shown in figure 15; on the same lines, the mastership of channel can be negotiated through a proper packet exchange and, if requested by application, a slave can become master. It is also possible to create partitions of slave devices on FlexSPI bus by having different chip select lines, providing an useful tool to include on bus SPI-only devices; their inclusion, although supported by this protocol, should be carefully implemented.

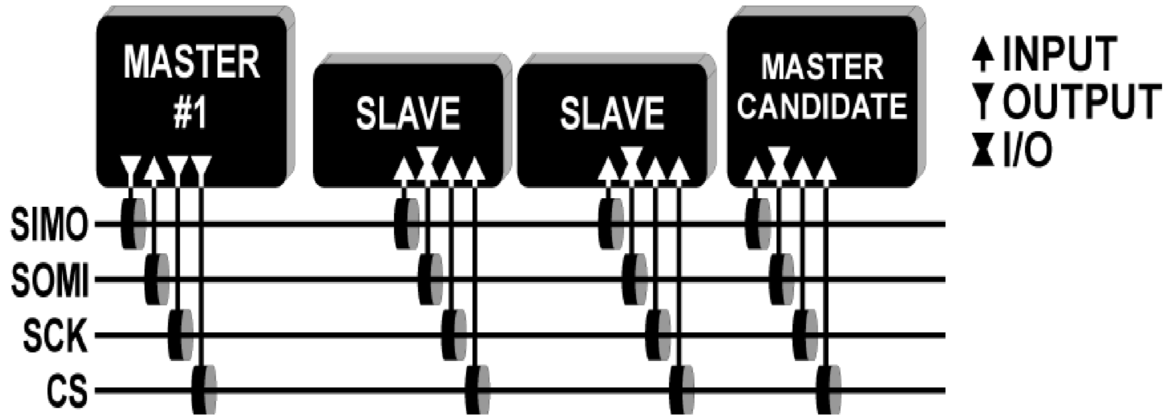


Figure 15. Example of FlexSPI bus functionality.

The exclusive concession of channel from the master does not forbid any form of slave signaling: the presence of the pull-up resistor, in some circumstances, can be exploited to perform an open-drain signaling. This option has been forecast in applications where the master is simply used as a sink collecting data from slaves, e.g. sensors; with this configuration, any device has the possibility to communicate the need of a bus concession to the master, which will perform a procedure to understand which slave produced the signal. Once pinpointed the slave, the master will give to it possession of the channel, downloading the content of the slave's pending queue and eventually transmitting the data. The developed FlexSPI protocol, in fact, is based on SPI and therefore supports full-duplex communications, doubling the available bandwidth of the channel with respect to I^{2}C; this eventuality, however, must be properly signaled in order to let devices to correctly configure their pins and the available memory. It is obviously possible to have just half-duplex data exchanges but, if transmitter is the slave, the master should be made aware of how many clock pulses it must send in order to fully download the data packet.

Analyzing this protocol from a more physical point of view, FlexSPI is taking advantage of both the GPIO and SPI modules available on microcontrollers: the default situation, in fact, features the SIMO and SOMI signals as inputs for both the master and the slaves while the remaining signals are output for the master and input for the slaves. As soon as a particular communication session is desired, the firmware will disconnect the necessary pins from the GPIO module and connect them to the SPI one, performing the required operations. This model of the physical interface is represented in figure 16, showing that the constant multiplexing among the modules is completely transparent to the application layer.

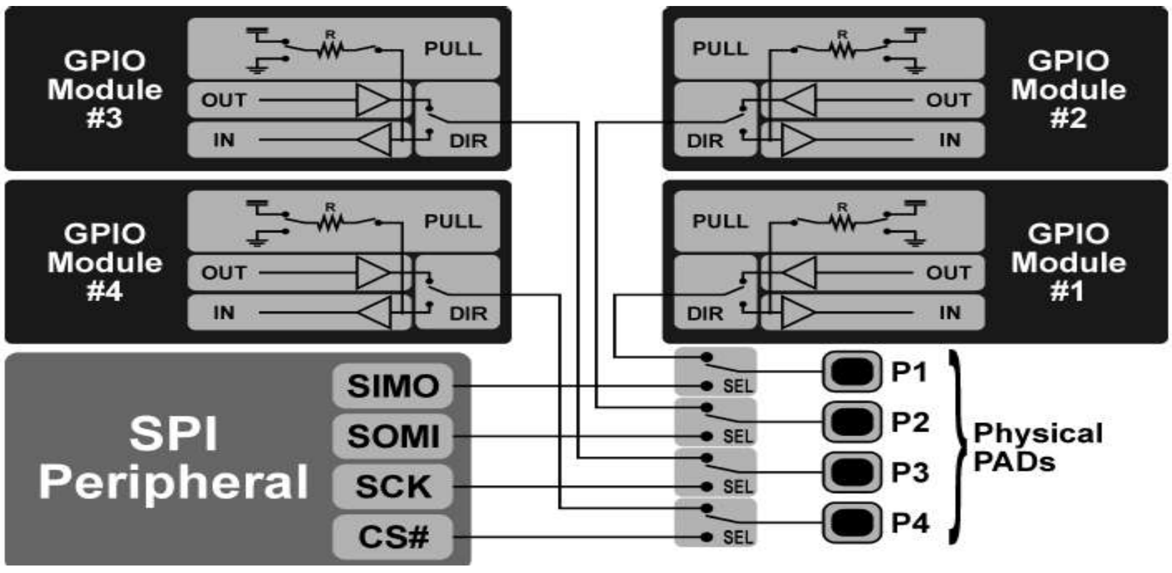


Figure 16. Example of FlexSPI reference physical layer.

An important implementation was adopted in the developed firmware: it is preferable, for the master, to never connect the *Chip Select line* to the SPI module. Since this signal is used to window command session and not bytes, in fact, its behavior would be more unpredictable: it is suggested to let the *Chip Select pin* always connected to the GPIO module, lowering and rising its output logical level when needed in the implementation.

An important module employed to develop the FlexSPI firmware is the Direct Memory Access (DMA) controller; the use of this module is mandatory to perform a robust data exchange among FlexSPI devices, since in this way it is possible to counteract the delay introduced by the computational effort to prepare the communication. Moreover, by using this peripheral, it is possible to temporarily put the device in low power mode until the data transfer is completed, increasing the device life cycle by saving energy.

VII. SMART-OBJECT ARCHITECTURE WITH FLEXSPI BUS COMMUNICATION

Among the many challenges to overcome for designing sensors for the IoT, enhancing device flexibility while keeping low power consumption is of paramount importance. These devices find place in the most different environments, with a wide range of applications; thus it is hard to spot a global optimum [16][17]. The partitioning approach, in such a scenario, is best way to tailor trade-offs involving each slice of the whole problem. Smart-object is a device able to:

- ✓ Interface with the transducers, in order to perform environmental measurements;
- ✓ Collect measurements and, possibly, to elaborate them;
- ✓ Communicate with other smart-objects, allowing to create a WSN.

Each of identified capabilities can be realized by means of a dedicated sub-system, equipped with a microcontroller (µC) allowing to coordinate and communicate with others. To optimize energy consumption, each subsystem implements various sleep modes, allowing to modulate its power demand depending on application needs. The deepest allowable sleep mode is selected as soon as a subsystem contribution is not necessary. The sensor node is thus comprised of the following subsystems: the *Transducer Interface* (TI), the *Measurement Collection and Elaboration* (MCE) and the *Communication Interface* (CI). Despite sensors employed in WSN applications are categorizable in very few types [18], transducers characteristics can be extremely variable, leading to the need for dedicated signal conditioning chains not only by different physical phenomena but also by different applications. For this reason, smart-objects with multiple sensors need more than one TI so that the transduced signals can be properly amplified, filtered, and DC offset before providing them to the Analog to Digital Converter (ADC). It is not uncommon to find architectures exploiting ADCs embedded into each one of the TI-µCs to perform conversion. Despite this approach allows to perform time synchronized acquisitions on multiple channels, it could not be the optimal choice in many battery-operated sensor nodes due to:

- ✓ Increased hardware complexity and costs;
- ✓ Increased communication traffic between MCE and TIs;
- ✓ Increased power consumption.

Proposed sensor node architecture shown in figure 17 employs a single analog channel shared between TIs; once signal is properly converted into the digital domain, the MCE collects and elaborates measurements. As soon as the application layer requires it, the MCE interacts with CI to allow the sensor node to communicate with other devices. In the described architecture, communication between subsystems plays a key role for reconfigurability and scalability.

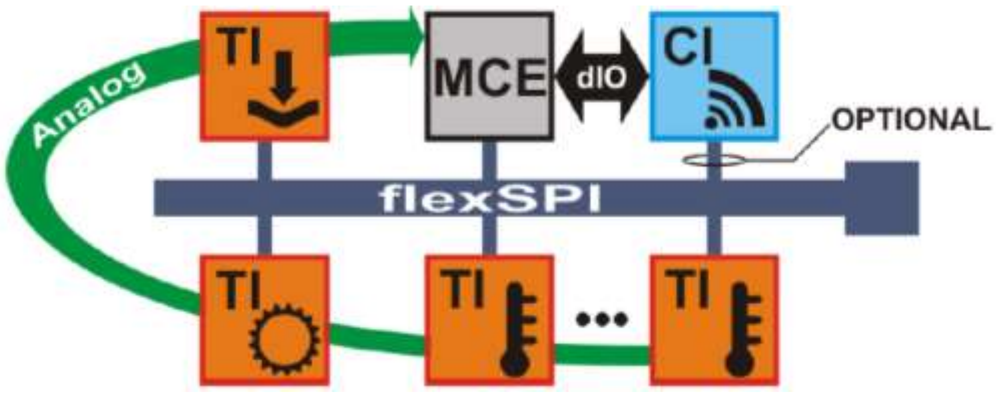


Figure 17. Sensor node modular architecture obtained by partitioning the device into subsystems, communicating with the developed FlexSPI protocol. Communication Interface dedicated digital I/O (dIO) can also be flexSPI enabled, if needed.

Since in a scalable *smart-object* it is not known a-priori how many TIs are present, addressing is mandatory: although latest revisions of I^2C protocol have enhanced its flexibility (easing, as example, the identification of devices by means of a device ID), major issues exist when dealing with the I^2C protocol for advanced subsystems communication:

- ✓ Limited addressing space could rise the need for bus multiplexers;
- ✓ Reduced bandwidth, due to need for line pull-up resistors, usually limits its use only to low-speed applications;
- ✓ Lack of advanced addressing, such as multicast and broadcast messages, limits communication flexibility;
- ✓ Open-drain nature of this bus leads usually to not negligible power consumption.

In this work, we have proposed a new communication protocol, based on SPI peripheral, allowing to tackle needs of smart-objects. The developed approach allows sharing SPI bus between all devices: *Chip Select* ($\overline{\text{CS}}$) signal is used mainly to window messaging and to awake usually sleeping the slave devices, while addressing is performed at packet level. One master device drives the communication and propagates clock through the SCK signal. Slaves normally receive on *Slave Input/Master Output* SIMO line and, instead of configuring it as an output, they keep SOMI line always free, as input, until master device addresses them; when SOMI is not in use, master applies a pull-up resistor to mark the line with a recessive state. This enables flexSPI connected devices to spot broken links and to request master attention generating an IRQ. Unlike I^2C, pull-up resistor is disabled as soon as channel is reserved to a slave device, thus not influencing the peripheral bandwidth and power consumption. With this approach SPI peripheral, instead of being a 3+n wires interface (with n number of slave devices), becomes a 4 wires, full-duplex, hot-pluggable interface bus despite number of connected devices.

VIII. UTILIZED HARDWARE AND SOFTWARE TOOLS FOR FLEX-SPI DEVELOPMENT: TI EXPERIMENTER BOARD AND DMA MODULE FEATURES

The system chosen to test the implementation of FlexSPI firmware is the MSPEXP430F5438 Experimenter Board, shown in figure 18a. This evaluation board is made by Texas Instruments and is equipped with a MSP430F5438A microcontroller, whose capabilities can be easily exploited thanks to the several peripherals connected. The integrated interfaces on the board are a 138 x 110 dot-matrix LCD, a two-axis analog accelerometer, a 5-directional joystick, two push buttons, two LEDs and a full analog signal chain from microphone to audio output jack; it is also possible to exploit an UART communication thanks to the mini-USB connection. At last, wireless communication is also possible via TI wireless evaluation module headers or the EZ430-RF2500T headers. With respect to the photo in figure 18a, the position of these peripherals on the board are shown in figure 18b [19].

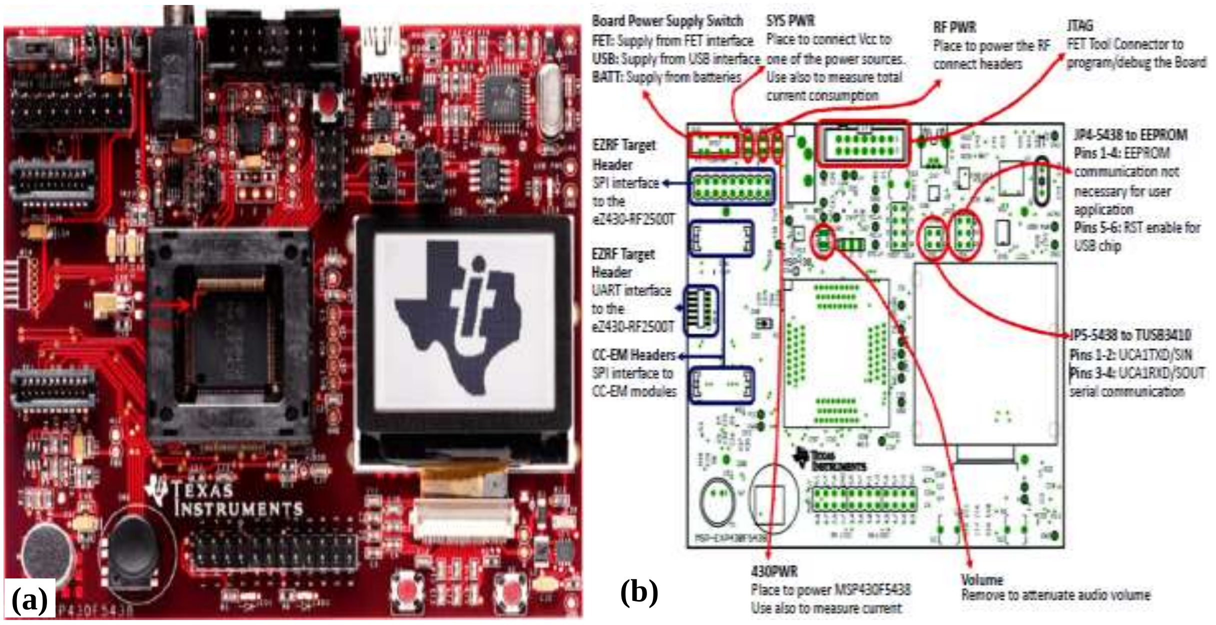
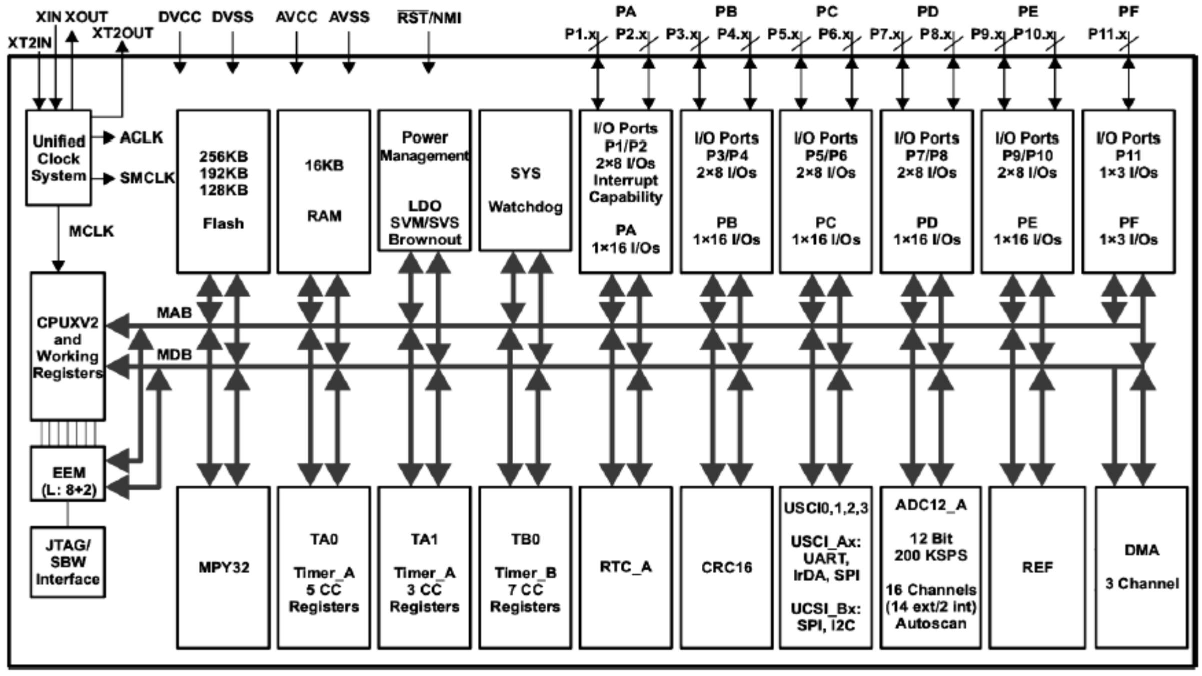

(a)
(b)
Board Power Supply Switch
FET: Supply from FET interface
USB: Supply from USB interface
BATT: Supply from batteries
SYS PWR
Place to connect Vcc to one of the power sources. Use also to measure total current consumption
RF PWR
Place to power the RF connect headers
JTAG
FET Tool Connector to program/debug the Board
EZRF Target Header
SPI interface to the eZ430-RF2500T
EZRF Target Header
UART interface to the eZ430-RF2500T
CC-EM Headers
SPI interface to CC-EM modules
JP4-5438 to EEPROM
Pins 1-4: EEPROM communication not necessary for user application
Pins 5-6: RST enable for USB chip
JP5-5438 to TUSB3410
Pins 1-2: UCA1TXD/SIN
Pins 3-4: UCA1RXD/SOUT serial communication
430PWR
Place to power MSP430F5438
Use also to measure current
Volume
Remove to attenuate audio volume
TEXAS INSTRUMENTS

Figure 18. MSP-EXP430F5438 TI Experimenter Board (a) and its functional overview.

The board is equipped with the Texas Instruments MSP430F5438A microcontroller, very well suited for applications requiring thorough energy management thanks to its architecture that, combined with several low-power modes, makes possible to achieve an extended battery life. Typical applications for this device include analog/digital sensor systems, digital motor control, remote controls, thermostats, digital timers and hand-held meters [20] [21]. Among its features, it is worth mentioning the powerful 16-bit RISC architecture of CPU that, combined with constant generators, contribute to maximize code efficiency with a system clock up to 25 MHz. CPU is highly transparent to the application: all operations, other than program-flow instructions, are performed as they were register operations, thanks to seven addressing modes for source operand and four addressing modes for destination operand [22]. The CPU is equipped with 16 registers that provide reduced instruction execution time: the register-to-register operation execution time is one cycle of the CPU clock. Four of the registers, R0 to R3, are dedicated as program counter, stack pointer, status register, and constant generator, respectively; the remaining registers are general purpose registers [23]. The different peripherals of the microcontroller are connected to the CPU using several kind of buses and can be handled with all instructions, which operate on word and byte data. Another important feature is the digitally controlled oscillator, capable of waking up the device from one of the low-power modes to active mode in about 3-5µs. The MSP430F5438A has one active mode and six software selectable low-power modes of operation: the software can be written so that an interrupt event can wake up the device from any of the low-power modes, perform the required instructions and restore back to the low-power mode.

Regarding its storage capability, MSP430F5438A is equipped with a 256 KB flash memory and 16 KB RAM; memory is organized to store, among other things, code memory and the location of interrupt vectors. CPU can perform single-byte, single-word and long-word writes to the flash memory while, regarding RAM memory, it can power down different sectors to save leakage. At last, this microcontroller includes three 16-bit timers, a high-performance 12-bit ADC, up to four universal serial communication interfaces (USCIs: SPI, I2C, UART and their enhanced versions), a hardware multiplier, DMA, RTC module with alarm capabilities and up to 87 I/O pins. The complete block diagram of MSP430F5438A is shown in figure 19.



XIN XOUT
XT2IN
XT2OUT
DVCC DVSS
AVCC AVSS
RST/NMI
PA PB PC PD PE PF
P1.x P2.x P3.x P4.x P5.x P6.x P7.x P8.x P9.x P10.x P11.x
Unified Clock System
ACLK
SMCLK
MCLK
256KB 192KB 128KB Flash
16KB RAM
Power Management LDO SVM/SVS Brownout
SYS Watchdog
I/O Ports P1/P2 2×8 I/Os Interrupt Capability PA 1×16 I/Os
I/O Ports P3/P4 2×8 I/Os PB 1×16 I/Os
I/O Ports P5/P6 2×8 I/Os PC 1×16 I/Os
I/O Ports P7/P8 2×8 I/Os PD 1×16 I/Os
I/O Ports P9/P10 2×8 I/Os PE 1×16 I/Os
I/O Ports P11 1×3 I/Os PF 1×3 I/Os
CPUXV2 and Working Registers
MAB
MDB
EEM (L: 8+2)
JTAG/ SBW Interface
MPY32
TA0 Timer_A 5 CC Registers
TA1 Timer_A 3 CC Registers
TB0 Timer_B 7 CC Registers
RTC_A
CRC16
USCI0,1,2,3 USCI_Ax: UART, IrDA, SPI UCSI_Bx: SPI, I2C
ADC12_A 12 Bit 200 KSPS 16 Channels (14 ext/2 int) Autoscan
REF
DMA 3 Channel

Figure 19. Functional block diagram of the MSP430F5438A microcontroller.

The peripherals of the microcontroller used in the FlexSPI firmware have been previously studied and tested with demo codes in order to investigate how they work and how to obtain good performances; these peripherals are the GPIO, SPI and DMA modules [24].

a. General purpose Input Output (GPIO) module features

The general purpose Input Output (GPIO) module is responsible for the settings regarding the direction of every I/O pin available, making them an input for external digital signal or, if output, imposing their logical level. This module gives users the possibility to program individually and independently every pin thanks to the configuration of several registers: it is possible, for example, to set-up pull resistors, both up and down. Two ports, P1 e P2, have also interrupt capabilities that can be used to trigger some events. The MSP430F5438A microcontroller has twelve digital input/output ports, with usually eight pins each. Each line can be set for input or output direction, read or written, left floating or set through pull-up/pull-down resistors; it is also possible to configure the output drive strength, if EMI have to be taken into account. Given the exploitation of this particular port for FlexSPI firmware because of its interrupt capabilities, the block diagram of I/O port P1 is reported in figure 20.

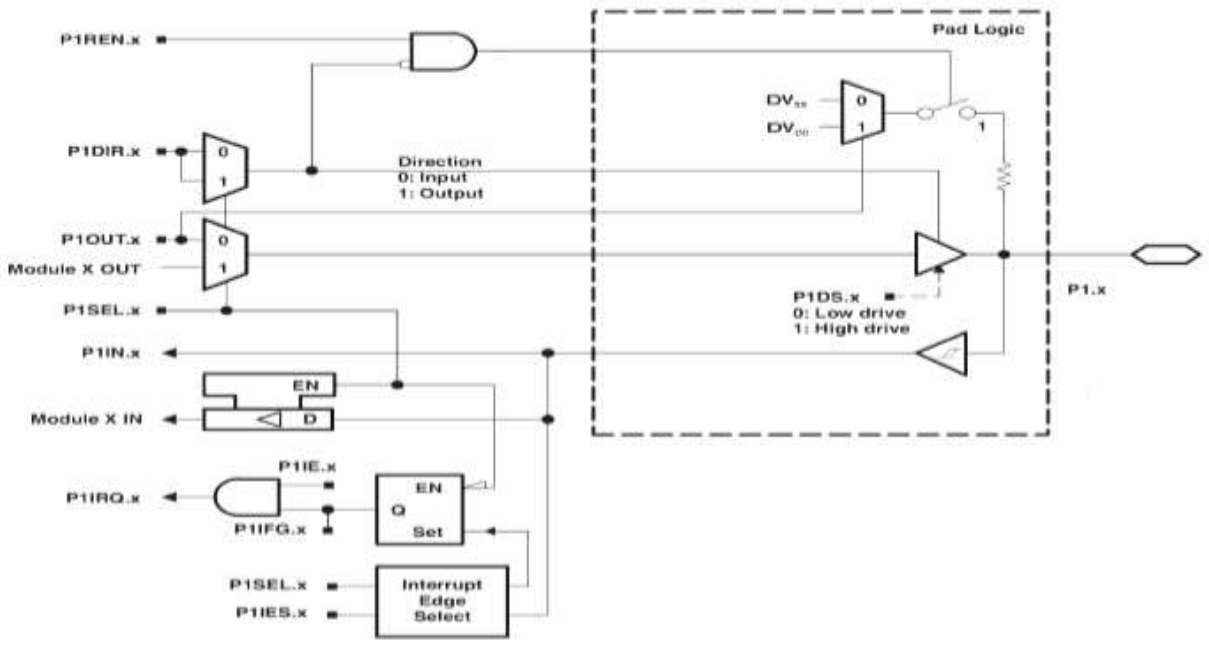


Figure 20. Port P1 block diagram.

The Interrupt Service Routines (ISRs) are crucial in the evaluation of the performances of a firmware and they should be as quick as possible to avoid critical situations, since their activation causes the immediate stop of microcontroller processing. The MSP430F5438A, in order to increase efficiency of the firmware, pre-allocates a specific portion of its memory to the supported interrupt vectors: thanks to pre-processor directive, it is possible to inform the compiler that an ISR for a specific vector, is about to start, and its memory address will be properly copied in the interrupt vector space address.

b. SPI module features

The SPI module is used to implement communication sessions based on the SPI protocol and it can be enabled through the Universal Serial Communication Interface (USCI). The block diagram of this peripheral is here reported in figure 21.

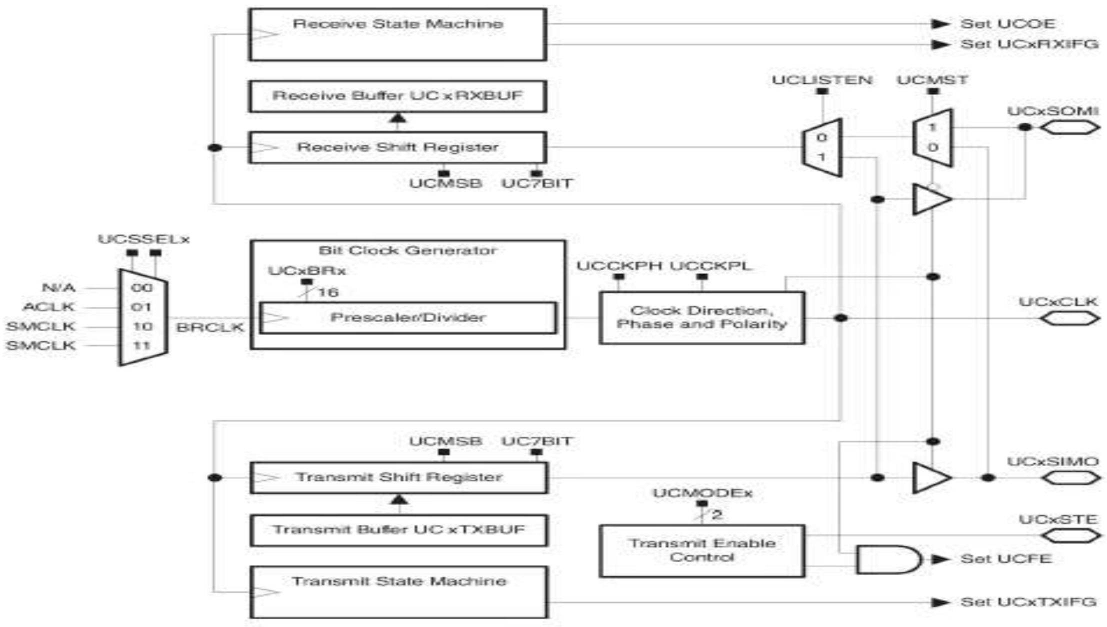


Figure 21. USCI block diagram in SPI Mode.

When configured for the SPI protocol, USCI connects the device to other systems through three or four pins, UCxSIMO, UCxSOMI, UCxCLK and UCxSTE; two SPI modes are in fact available, 3-pin or 4-pin, i.e. transmitting data without or with the *Chip Select* as indicator. Among the other features supported, such as possibility to properly configure SPI parameters, MSP430F5438A has separated buffer registers and independent shift register for transmission and reception and supports continuous Tx/Rx operations. The presence of UCxSTE pin allows multiple masters on the same bus, obtaining a 5-wire protocol; however, for the proposed target of obtaining a 4-pin bus, it has been chosen to configure this pin as GPIO and manually control its value obtaining a *Chip Select* signal and therefore more control on the transmission. The configuration used to test the working principles of SPI communication is reported in figure 22; experimental setup is composed by two *experimenter boards* whose pins, belonging to PORT10, are connected with a modified cable, designed with purpose of easily connecting the appropriate pins on different boards; the logic analyzer is used to track the exchanged signals. Figure 22 shows also another important tool used in all developed codes, the TI MSP-FET430UIF debugger; this device provides a USB interface to program and efficiently debug the firmware written for all the MSP430 family devices; the connection can be obtained through the JTAG interface or by the pin-saving Spy-Bi-Wire (2-wire JTAG) protocol. A useful tool to confirm correct transfer between the two devices is the logic analyzer shown in figure 22; the communication is verified by hooking its probes to the SPI pins, as can be seen in figure 23. The logic analyzer used for this and the other simulations involving communication sessions is the *Acute TravelLogic TL2136* that, after a proper software configuration, displays the exchanged waveforms between master and slave. The software platform of the logic analyzer provides also a trigger mechanism that has been locked to the falling edge of the *Chip Select*. This program can be also configured to decode the captured signals as belonging to an SPI bus, once specified all the parameters of the clock.

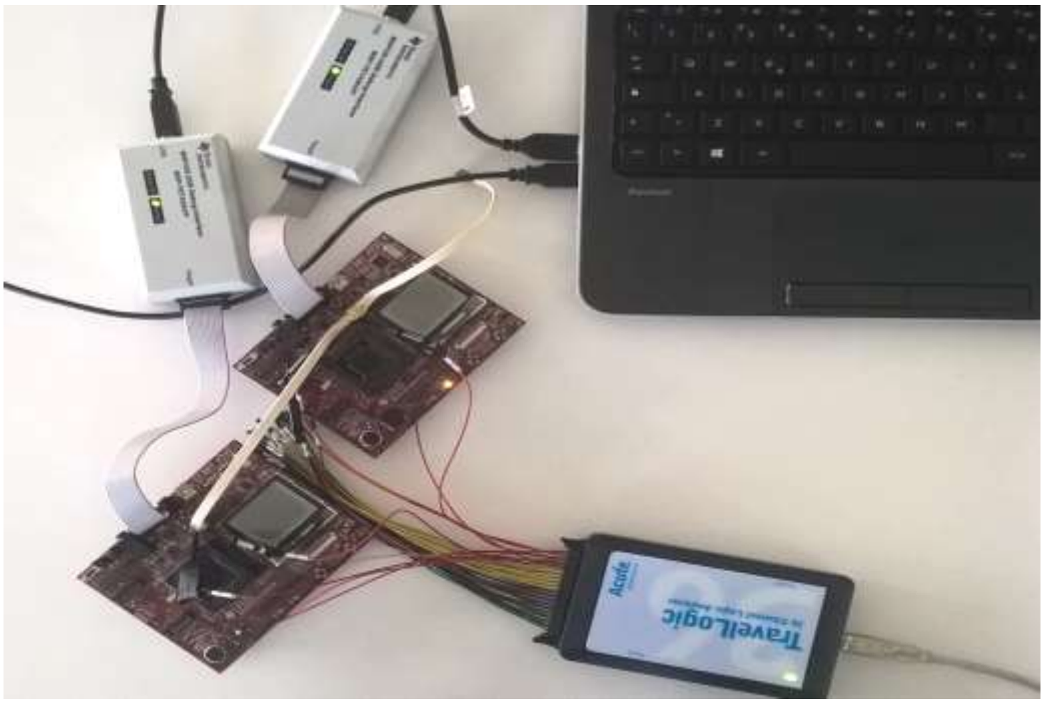


Figure 22. Realized experimental setup for SPI communication.

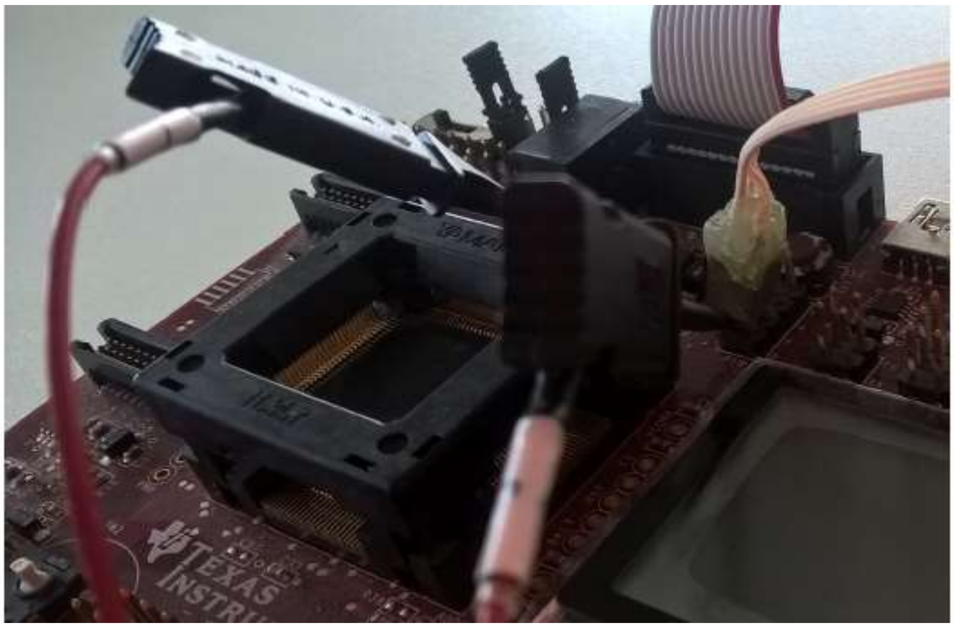


Figure 23. Detail of the hooking of the logic analyzer probes to the MSP430 SPI pins.

The results of developed firmware, as seen from the logic analyzer, are shown in figure 24: the displayed pattern proves correct behavior of the two devices and, in particular, of the slave, which echoes the data received from previous session without losing timing of transmission.

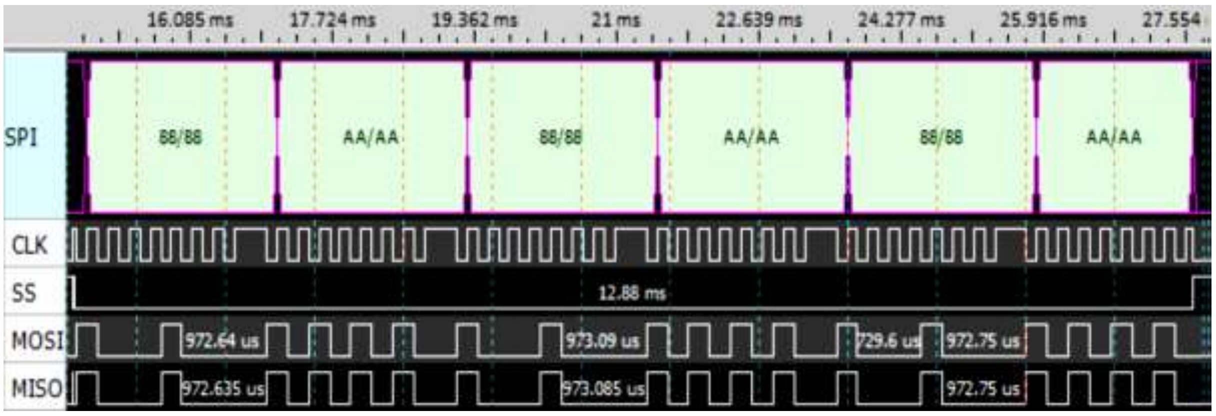


Figure 24. Exchanged data between master and slave after the first training session.

Figure 24, however, describes the limit of relying only on the SPI module for data transfer: from one byte to the other, in fact, the delay introduced by the loading of transmission register can be seen. This behavior decreases the bit-rate and therefore efficiency of the transmission; for this reason, advanced bus implementations exploit advanced mechanism that, triggering data transfers independently from CPU, increases the speed and saves power consumption.

c. DMA module features

Direct Memory Access (DMA) controller is a circuital component that enables the transfer of data from one memory address to another without any CPU intervention. The use of DMA can increase the throughput of all communication modules and reduce system power consumption by allowing CPU to remain in low-power mode until data is simply moved from the peripheral into RAM. The typical architecture of a system equipped with a DMA is reported in figure 25.

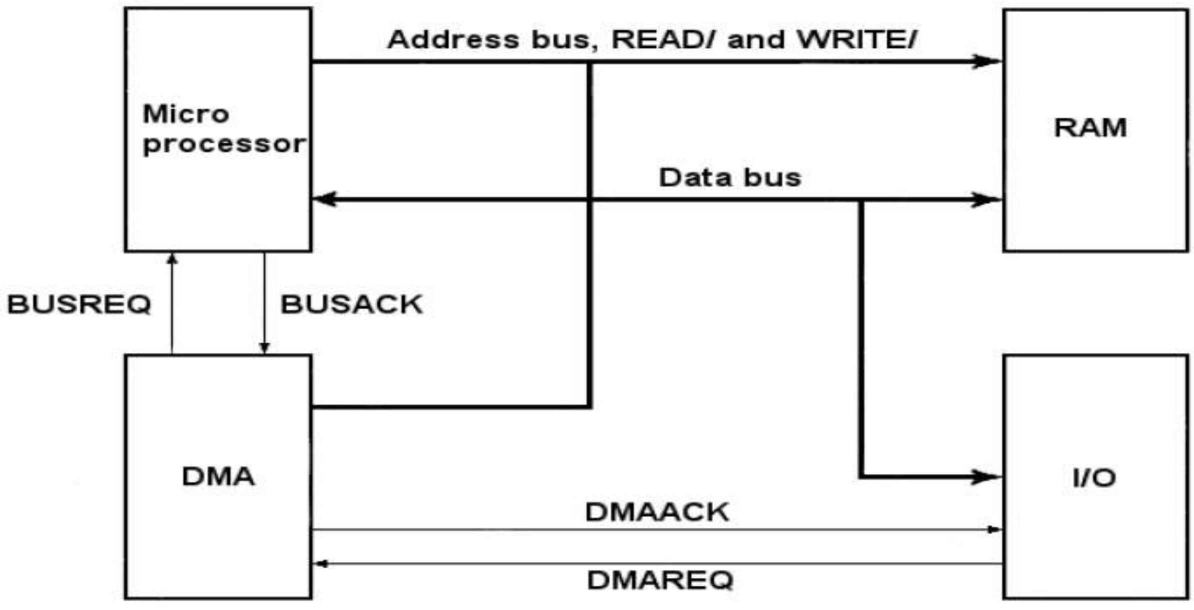


Figure 25. Architecture of a system, smart device or microcontroller with DMA.

Thanks to the many registers typically present in any microcontroller, it is possible to fully configure DMA via software, specifying peripherals involved in the transfer and all required parameters: data, in fact, can be moved as bytes or words, with the possibility of performing a single or a block transfer, eventually repeated. When the I/O peripheral needs to move data to RAM, it asserts a DMAREQ signal to DMA, that successively will require bus availability to the microprocessor through a BUSREQ command. As soon as microprocessor has no pending operations and therefore is ready to give up the bus, it will inform the DMA with a BUSACK signal that grants to DMA the possibility to place the targeted address into the bus, starting a read/write operation that does not involve the CPU.

Since only one device can use DMA, several copies of DMA circuitry called *channels* are usually available permitting simultaneous operations with the same module. MSP430F5438A, equipped with three DMA channels, is designed to perform automatic full data transfer via several communication protocols thanks to the support for different triggers, i.e. events that cause new data to be shifted out or copied in the memory. Its behavior can be set in two different ways:

- ✓ The DMA can be edge-triggered, meaning that a new transfer will be performed after a certain value has changed its logical level.
- ✓ The DMA can be also level-triggered and it will keep moving the data until the chosen parameter will maintain the desired logical level.

MSP430F5438A microcontroller supports also DMA interrupts that can be exploited to perform actions after all data have been transferred; it is also possible to perform data transfer between fixed memory locations or to increment the source and/or destination addresses by locking the increment to an interrupt. Since the example code written for this module is based on a single transfer, in figure 26 its state diagram with all the registers is reported.

The example firmware written to investigate the benefits of DMA adoption for data transfer implements a simple and fixed data exchange between master and slave via the SPI protocol. Since it is desirable to link the shift of data with events related to SPI module, DMA has been configured to trigger when a specific flag is set or reset: since not all USCI modules support these features, the chosen USCI module has been UCB0, not only for DMA support but also for the proximity of all the necessary pins, assigned to PORT3. After the configuration of SPI parameters, the DMA is set: as reported in the portion of code in figure 27, the first aspect that has been configured is the event that triggers activation of the DMA and hence the movement of data from the source address to destination address. After that, all other parameters of DMA module are set, starting from the type of transfer that will be performed to quantity of memory that DMA must increment to find a new byte. At last, two more aspects must be taken into account: first, enabling of interrupts that, with an appropriate ISR, will end the transmission and second, the number of bytes that will be transferred, because once this register becomes zero it can be used as an interrupt event. The DMA settings of the device acting as slave are the same of the master's ones, therefore they have not been reported.

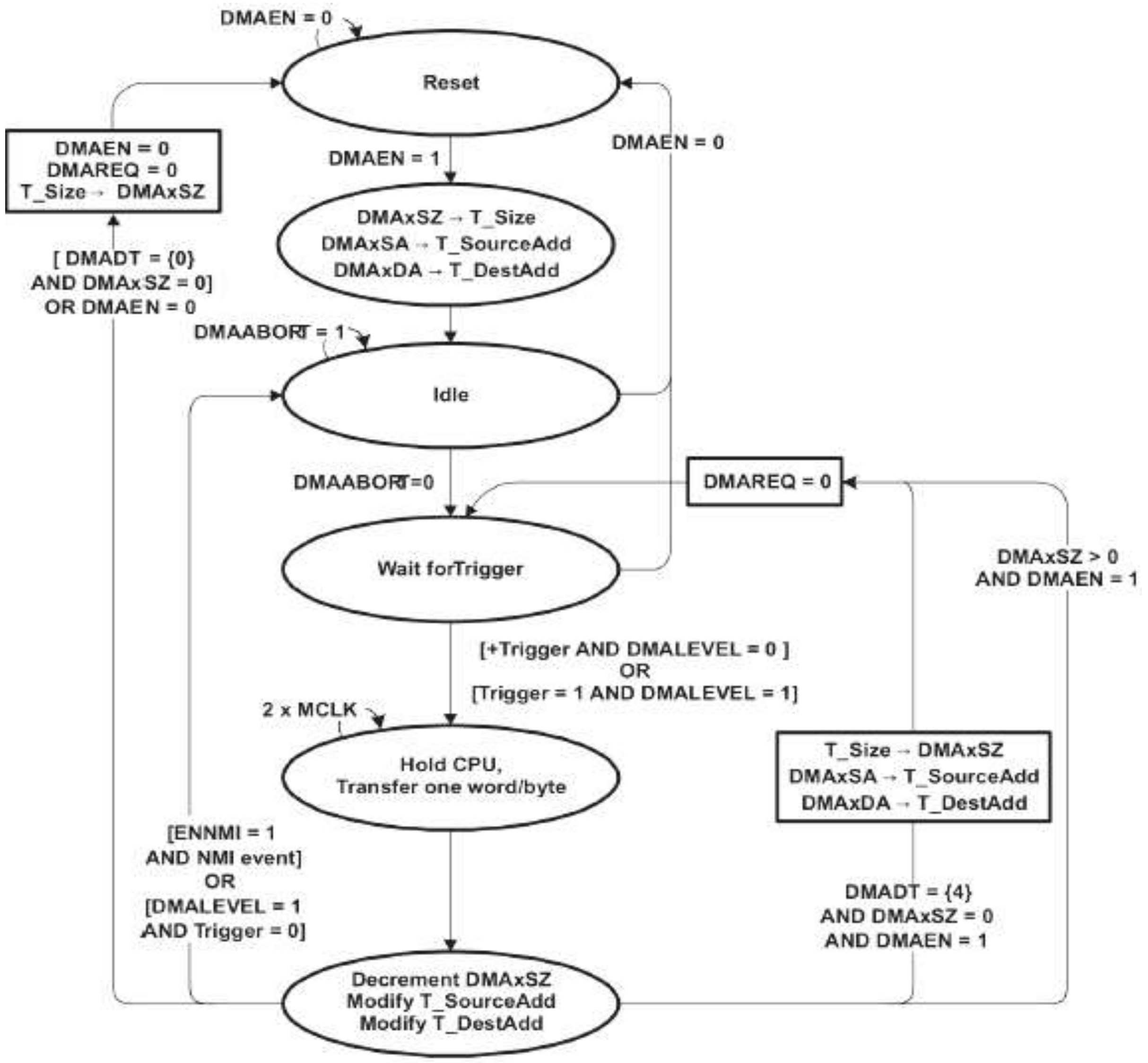
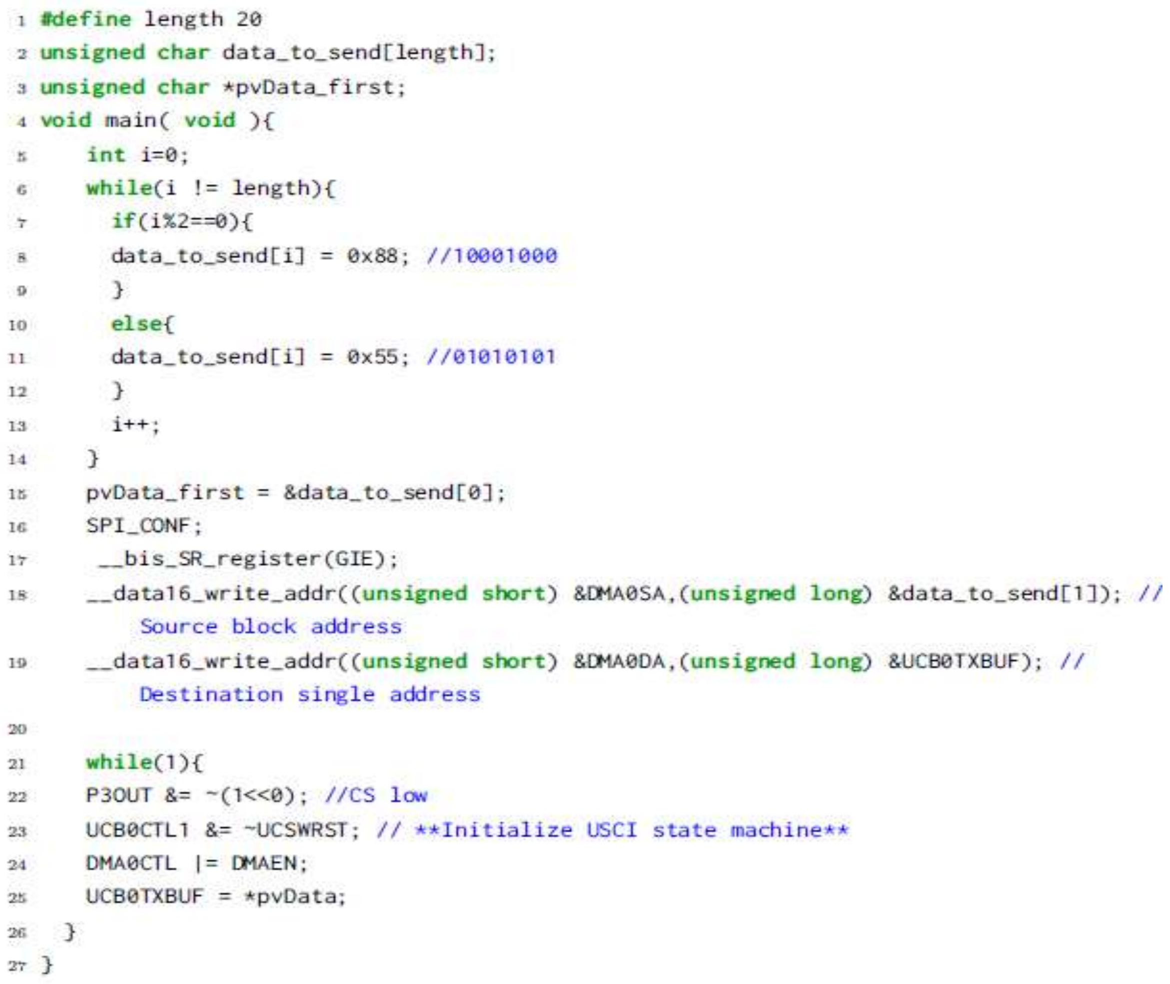


Figure 26. DMA single transfer state diagram.

```
DMACTL0 = DMA0TSEL_19; //UCB0 tx
DMA0CTL = DMADT_0 + DMASRCINCR_3 + DMAIE+DMADSTBYTE+DMASRCBYTE; // DMA transfer: source
    word to destination byte; source address get increased of 1 byte after every tx
DMA0CTL &= ~DMAIFG;
DMA0SZ = 19;// Block size (number of bytes)
```

Figure 27. Developed code related to DMA master configuration and settings.

In figure 28, the main file for the master device is also reported; the slave shares a very similar structure. The first step must always be the data assignment: after having created a proper vector, the pointer to its first value is assigned to DMA0SA, i.e. the source address of channel 0 of DMA (memory location will be incremented according to explained configuration); similarly, the destination address is the transmission buffer of UCB0 SPI peripheral. The functions used for these operations are provided from the header that the IAR Embedded Workbench (an IDE useful to easely create firmware for a large number of microcontrollers) includes according to the used microcontroller (the MSP430F5438A) to test the firmware.



```
#define length 20
unsigned char data_to_send[length];
unsigned char *pvData_first;
void main( void ){
    int i=0;
    while(i != length){
      if(i%2==0){
      data_to_send[i] = 0x88; //10001000
      }
      else{
      data_to_send[i] = 0x55; //01010101
      }
      i++;
    }
    pvData_first = &data_to_send[0];
    SPI_CONF;
     __bis_SR_register(GIE);
    __data16_write_addr((unsigned short) &DMA0SA,(unsigned long) &data_to_send[1]); //
        Source block address
    __data16_write_addr((unsigned short) &DMA0DA,(unsigned long) &UCB0TXBUF); //
        Destination single address

    while(1){
    P3OUT &= ~(1<<0); //CS low
    UCB0CTL1 &= ~UCSWRST; // **Initialize USCI state machine**
    DMA0CTL |= DMAEN;
    UCB0TXBUF = *pvData;
  }
}
```

Figure 28. Developed code related to DMA - Master main file and data assignment.

It is possible to note that the first element of the source address is not the first element of the array, but the second. This choice has been done because, in order to trigger DMA, the first data must be sent manually, i.e. by properly loading the transmission buffer with its value. This operation, after the lowering of Chip Select and initialization of USCI module, completes the infinite loop wherein the microcontroller is stuck into. At last, an interrupt service routine, activated when the DMA register size becomes zero, ends the communication session: chip select is raised and USCI module register set. These last operations are reported in the DMA interrupt service routine in the listing of figure 29, where the *even_in_range* intrinsic provides a hint to the compiler when generating switch statements for interrupt vector routines.

```
#pragma vector=DMA_VECTOR
__interrupt void DMA_ISR(void){
  switch(__even_in_range(DMAIV,16)) {
    case 2: // DMA0IFG = DMA Channel 0
        P3OUT |= (1<<0);
        UCB0CTL1 |= UCSWRST;
        break;
    default: break;
  }
}
```

Figure 29. Developed code related to DMA - Master main file and DMA interrupt vector.

The pattern of the transmission, recorded with the logic analyzer, is shown in figure 30: the graphic stresses the benefits of the DMA module, that allows a data transfer with a higher bit-rate thanks to the lack of delays caused by the SPI buffer recharge.

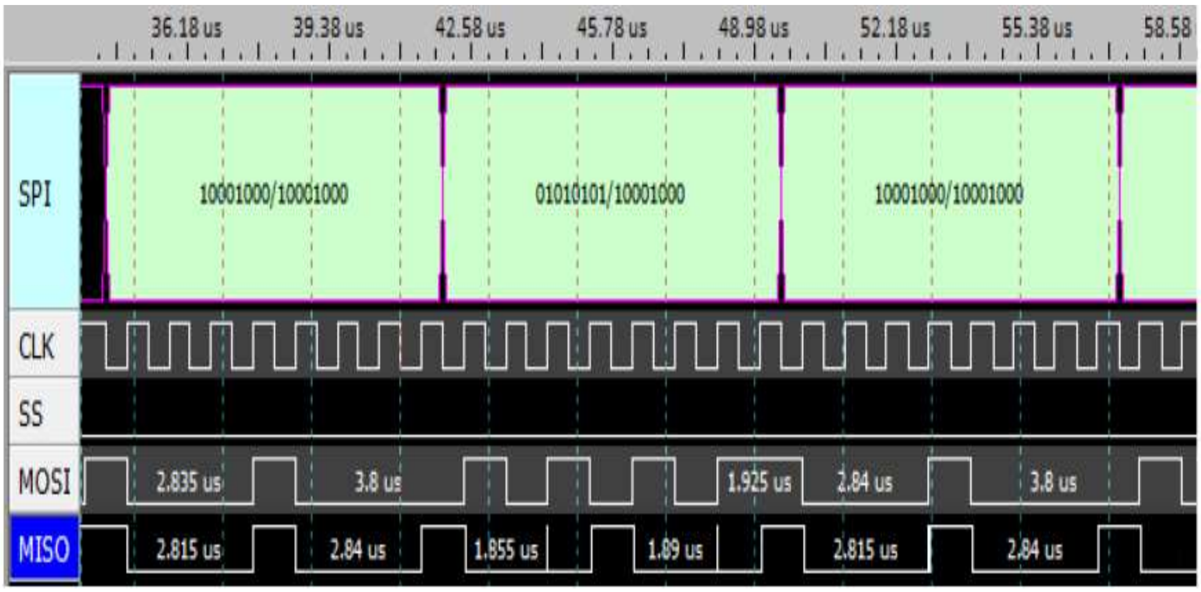


Figure 30. Data exchange recorded with the logic analyzer on the SPI bus when the DMA module is exploited.

Given the possibility of putting the device in low-power mode when the transfer occurs, when this module is supported by a dynamic software architecture it is possible to achieve great power saving benefits, following the direction pointed by the requirements that IoT applications should satisfy. This is one of the reason for which the implementation of FlexSPI has been made with an appropriate software structure.

IX. CONCLUSIONS

In this paper, we investigated communication needs for smart-objects suitable for the IoT; in order to enhance flexibility, maintainability and reliability while keeping low power consumption, we have developed and described a novel efficient and flexible inter-devices communication protocol, named Flex-SPI. Built on top of well-known SPI protocol, the proposed approach allows reducing routing overhead and enabling modularity by exploiting packet-oriented messaging. In order to achieve these improvements, SPI bus is completely shared between connected devices avoiding conflicts by keeping free (i.e. in input direction) the *Slave Output/Master Input* SOMI signal and occupying it only after advanced addressing. Low power and high bandwidth characteristics of SPI bus are maintained while enabling smart features such as hot-pluggability, link diagnosis, devices discovery, synchronization and master solicitation. In conclusion, the flexSPI proposal, together with extremely flexible unicast, multicast and broadcast messaging, presents a full range of advanced features among which dynamic slave device discovery, address leasing, master device synchronization and solicitation while keeping an extremely low routing complexity. Implemented in prototyping hardware, the developed flexSPI protocol, thanks to its flexibility, will be a solid and successful way to improve communication between multiple sub-systems.

REFERENCES

[1]. K. Ashton: That "Internet of Things" Thing. RFiD Journal, Vol.22 (7), pp. 97-114, (2009).

[2]. D. Bandyopadhyay, J. Sen: Internet of things: Applications and challenges in technology and standardization. Wireless Personal Communications, Vol. 58 (Issue 1), pp. 49-69. (2011).

[3]. . INFSO D.4 Networked Enterprise & RFID INFSO G.2 Micro & Nanosystems in co-operation with RFID Working Group of the European technology platform on smart systems integration (EPOSS), "Internet of Things in 2020, a roadmap for the future" (2008).

[4]. D. Evans: The internet of things: How the next evolution of the internet is changing everything. CISCO White Paper, 1, 14 (2011).

[5]. J. Williams, Internet of Things: Science Fiction or Business Fact?. Harvard business review Analytic Services, sponsored by Verizon. (2014).

[6]. D. Zheng, W. A. Carter: Leveraging the Internet of Things for a More Efficient and Effective Military. Rowman & Littlefield. (2015).

[7]. W.Peng, S.Fei, H.Bohong, L.Wenpeng: Ami based sensing architecture for smart grid in IPV6 networks. Int.J. on Smart Sensing and Intelligent Systems,Vol.9 (4), pp.2111-2130 (2016).

[8]. P. Visconti, R. Ferri, M. Pucciarelli, E. Venere: Development and Characterization of a solar-based energy harvesting and power management system for a WSN node applied to optimized goods transport and storage. Int. J. on Smart Sensing and Intelligent Systems, Vol. 9 (4), pp. 1637-1667, http://s2is.org/Issues/v9/n4/ (December 2016).

[9]. C. Hu: Secure data storage mechanism for integration of Wireless Sensor Networks and Mobile Cloud. International Journal on Smart Sensing and Intelligent Systems. Vol.9 (3), pp. 1563-1591, https://s2is.org/Issues/v9/n3/ (2016).

[10]. P. Raghu Vamsi, K. Kant: Trust aware data aggregation and intrusion detection system for Wireless Sensor Networks. International Journal on Smart Sensing and Intelligent Systems, Vol. 9 (2), pp. 537-562, https://s2is.org/Issues/v9/n2/ (2016).

[11]. K. O. Mahony, J. Liang, K. Delaney : Real-time information profiling for smart objects. Proc. of IEEE Int. Conference on Virtual Environments Human-Computer Interfaces and Measurement Systems, DOI: 10.1109/VECIMS.2011.6053854 (2011).

[12]. P.Visconti, P. Primiceri, C. Orlando: Solar Powered Wireless Monitoring System of Environmental Conditions for Early Flood Prediction or Optimized Irrigation in Agriculture. ARPN Journal of Engineering and Applied Sciences, Vol. 11 (Issue 7), pp. 4623-4632, (2016).

[13]. I^{2}C-bus specification and user manual - Rev. 6, NXP Semiconductors, http://www.nxp.com/documents/user_manual/UM10204.pdf (4 April 2014).

[14]. SPI Block Guide V03.06, Motorola Inc., ftp://ztchs.p.lodz.pl/Int_w_sm/SPI.pdf (2003).

[15]. Y. Zhao, H. Ding, Y. Guo, J. Nan, S. Zhou, S. Yao, Q. Liu: Analysis of multi-channel two-dimensional probability CSMA ad-hoc network protocol based three-way handshake mechanism. Int. J. on Smart Sensing and Intelligent Systems, Vol. 10 (1), pp. 88-107 (2017).

[16]. P.Visconti, C.Orlando, P. Primiceri: Solar Powered WSN for monitoring environment and soil parameters by specific app for mobile devices usable for early flood prediction or water savings. IEEE 16th Int. Conf. on Environment and Electrical Engineering EEEIC, DOI: 10.1109/EEEIC.2016.7555638, SCOPUS = 2-s2.0-84988418455 (September 2016).

[17]. P. Primiceri, P.Visconti: Solar-powered LED-based lighting facilities: an overview on recent technologies and embedded IoT devices to obtain wireless control, energy savings and quick maintenance. Journal of Engineering and Applied Sciences ARPN, Vol. 12 (Issue 1), pp. 140 - 150, http://www.arpnjournals.com/jeas/volume_01_2017.htm, (January 2017).

[18]. M. Beigl, A. Krohn, T. Zimmer, C. Decker: Typical sensors needed in ubiquitous and pervasive computing, Proceedings of First International Workshop on Networked Sensing Systems (INSS), Tokyo (Japan), pp. 153-158, (2004).

[19]. P.Primiceri, P.Visconti, A.Melpignano, A.Vilei, G.M.Colleoni: Hardware and software solution developed in ARM mbed environment for driving and controlling DC brushless motors based on ST X-NUCLEO development boards. Int. J. on Smart Sensing and Intelligent Systems, Vol. 9 (3), pp. 1534 - 1562, http://s2is.org/Issues/v9/n3/papers/paper17.pdf (2016).

[20]. P.Visconti, P. Primiceri, P.Costantini, G.Colangelo, G. Cavalera, "Measurement and control system for thermo-solar plant and performance comparison between traditional and nanofluid solar thermal collectors"; Int. Journal on Smart Sensing and Intelligent Systems, Vol. 9 (Issue 3), pp. 1220 – 1242, http://s2is.org/Issues/v9/n3/papers/paper3.pdf (2016).

[21]. P.Visconti, R. Ria, G. Cavalera: Development of smart PIC–based electronic equipment for managing and monitoring energy production of photovoltaic plan with wireless transmission unit. ARPN Journal of Engineering and Applied Sciences, Vol. 10 (Issue n. 20), pp. 9434 - 9441, http://www.arpnjournals.com/jeas/volume_20_2015.htm, (November 2015).

[22]. P. Visconti, A. Lay-Ekuakille, P. Primiceri, G. Cavalera: Wireless Energy Monitoring System of Photovoltaic Plants with Smart Anti-Theft solution integrated with Household Electrical Consumption's Control Unit Remotely Controlled by Internet. Int. Journal on Smart Sensing and Intelligent Systems, Vol. 9 (Issue 2), pp. 681 – 708 (June 2016).

[23]. P.Visconti, R.de Fazio, P.Primiceri, A.Lay-Ekuakille: A solar-powered White LED-based UV-VIS spectrophotometric system managed by PC for air pollution detection in faraway and unfriendly locations. Int. Journal on Smart Sensing and Intelligent Systems, Vol. 10 (1), pp. 18 - 48, http://s2is.org/Issues/v10/n1/, SCOPUS = 2-s2.0-85014108211 (2017).

[24]. P. Visconti, A. Lay-Ekuakille, P. Primiceri, G. Ciccarese, R. de Fazio; "Hardware design and software development for a White LED-based experimental spectrophotometer managed by a PIC-based control system" IEEE Sensors Journal, ISSN: 1530-437X, Vol. 17 (Issue 8), pp. 2507 - 2515, DOI: 10.1109/JSEN.2017.2669529 (2017).