ADHOC EXTENSIBILITY USING THE STANDARD I2C BUS

Hashini Senaratne[1], Yasura Vithana[1], Udith Gunaratna[1], Pubudu Gunatilaka[1], Chathura de Silva[1], Piyum Fernand [2]

[1] Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka

[2] Singapore University of Technology and Design, Singapore

ABSTRACT

In this paper we present supplementary work carried out on extending the standard I2C communication bus under the project SiFEB—A Simple, Interactive and Extensible Robot Playmate for Kids; in order to be capable of attaching previously unconfigured slave devices into an existing standard I2C communication network and get them functioning by properly allocating valid addresses to them. The approach uses standard I2C hardware and two general purpose I/O lines per a single slave device which are cascaded along the bus in order to give the device a sense of physical position of attachment.

1. INTRODUCTION

Today, the lives of the human beings have been immensely affected by electronic devices. Also the devices have reached a state where they no longer provide a very specific service but a wide collection of services coordinating together. This has been achieved in many cases by a group of low level electronic components which are specialized in a certain task working together inside the device that we see from the outside as one single unit [1, 2].

The requirement of using many different components in order to provide a complex service lead to the introduction of the communication necessity among these individual components. The introduction of various communication protocols to be used with embedded systems like I2C [3], SPI [4], OneWire [5], CAN [6], USB [7], RS-232 [8], SMBus [9], etc. is a result of this. From these different communication protocols, I2C has become a very widely used communication protocol in embedded applications.

The standard I2C bus protocol initially developed by Philips Semiconductor, known today as NXP Semiconductors was intended for the use of coordination between several chips placed separately on the same main board [3]. So basically with the nature of the initially intended application, all the devices that are needed to be part of the communication network was supposed to be known well in advance.

In an I2C network, communication happens in a master slave pattern. In most of the cases I2C protocol is used in single master-multi slave environment although the protocol supports multi master scenarios as well [3]. A single node which is considered to be the master controls the whole communication and also acts as the central point of the communication. The slave devices have addresses unique within the network that they are in and the master is capable of writing to and reading from these devices by calling them with their respective address using the communication bus that all the devices are connected to. If there are no address conflicts, the device with the address in concern will react accordingly.

Due to the simple and reliable interface, I2C has become one of the common features that even a most commonly used microcontroller provides [10]. Also the software support for the use of this protocol is at a very high level in most of the applications [11]. As in most of the embedded communication protocols, if we want to connect a certain device to the communication network we need to make sure that the new device fits the current configuration of the communication network. Basically it is trivial that the signaling specifications need to be compatible. Apart from that, there is one important requirement that needs to be met in order to the device to work properly

and also not to disturb the existing setup, the device needs to have an address that uniquely identifies itself within the communication network.

In this paper we present a portion of the work in the project SiFEB [12]; the part related to inter-module communication. SiFEB is a modularized robot playmate for kids. The self-intelligent modules are connected to a standard I2C bus as slaves and the system is required to be capable of attaching a new module which is not previously configured to the system with a single master which is the main module. When the communication speeds and the other basic requirements are met, assigning an address to the newly connected module is the concern. If the system is capable of assigning a unique address to the new device, then the whole communication network can continue working as a usual I2C communication network. The work carried out that is being reflected in this paper is done to achieve this ad-hoc type extensibility to the standard I2C bus.

2. RELATED WORK

2.1. **Communication Protocols**

In order to handle communication among the modules of SiFEB system while preserving the extensible architecture, we cannot assign a single port per a module. Therefore there was the need of using a communication bus protocol to handle communication between the main module (master) and other modules (slaves). If we select a parallel communication bus protocol, the increased number of lines that will be passed through the modules will add a complexity to the physical structure of the SiFEB system. Therefore the best option was to handle communication using a serial communication bus protocol, even though the speed is less compared to a parallel communication bus protocol. There are several serial bus communication protocols like I2C, SPI, OneWire, CAN, USB, RS-232, SMBus, etc. which can be used with embedded systems.

I2C is a simple, well known, universally accepted, cost effective inter IC serial communication protocol, where it also has the capability of plug and play feature with large portfolio [10]. The main disadvantage that can be seen when it compared with some other serial communication protocols like USB and CAN is the limited speed, but the effect of that can be neglected when we are dealing with shorter distances. Since Serial Peripheral Interface (SPI) bus protocol additionally requires a separate select line for each module which is connected directly to the master, the requirement of three wires to handle communication would not be a good option for our need [4]. Controller Area Network (CAN) bus is a two-wire, secure and fast communication protocol [6]. But since CAN is an expensive and a bit complex protocol which is mainly automotive oriented with limited portfolio, I2C protocol is more applicable in our case. If we use Universal Serial Bus (USB) in this case, we need to make the slave modules more complex to be able to act like both hubs and functions and to have the ability to monitor the other modules connected to one module [7]. Since RS-232 is generally intended for communications between just two devices and due to the increased number of lines compared to I2C in handling two way data transferring, modifying RS-232 for our need would be more complex [8]. Although it would be possible to implement a serial communication protocol with a reduced number of lines with the use of OneWire protocol, less available support and low speeds compared to I2C protocol will make the task difficult [5]. Also, as it is a software controlled protocol, we may end up with complex module programs. I2C would be more flexible than some protocols like SMBus which is a derivation of I2C, because it does not reset slave devices upon a time out whenever the clock goes low for longer duration [9].

Due to the seen advantages over the other available commonly used serial communication bus protocols, I2C can be identified as a protocol which can be used to develop a communication protocol which is capable of assigning addresses to slaves dynamically.

2.2. **Dynamic Address Allocation with I2C**

Dynamically allocating I2C addresses using self-bus switching devices is a one significant methodology presented in a patent in 2004 [13]. In this method, the I2C bus is segmented with the use of I2C self-bus switches, where such a switch is

connected to the bus in between each and every I2C main device which acts like I2C slave and needs to be configured with a new address in the run time. These I2C self-bus switching devices also have I2C addresses, where they respond to their own addresses to on or off the switches which pass bus signals. They also can be turned off independent of the I2C commands, with the use of reset signals. At the resetting stage, all the I2C main devices except the switching devices have a default address and all are the same. At this stage, all the switching devices have a predefined address which are same per a bus line. Since the I2C main devices are disconnected by the I2C self-bus switches, I2C conflicts would not occur. The I2C master starts addressing by turning on one bus at a time, first commanding the switching device to change the address and then to switch on it to expand the bus. Down the line the master will command the connected I2C main device to change the address and repeat the process by changing the address of next switching device. This process is continued until all the devices connected to all the buses attached to master have unique addresses.

This technique allows to change the addresses of the devices at any time upon reset and allows to add devices to the bus at any time. So this technique can be used to create a hot pluggable system with certain restrictions. Since the separate lines handled by the master should have a separate set of switches where the default I2C addresses are different from other lines but equal in the same line, this cannot be seen as a generic solution. Also, since it needs to connect an additional device called switch along with the wanted I2C main device, this technique is not straightforward and simple enough. I2C address changing happens at the switching devices is a clear overhead to the system. So it would be really useful if we can come up with a technique, which can overcome the above mentioned problems.

3. MODIFICATIONS TO THE STANDARD I2C INTERFACE

In this scenario, we are attempting to assign a valid address to a device which is connected to an existing I2C communication network. We can assume two cases. One is that the new device is plugged into the existing network while the existing network is powered on and the other is that the existing network might be powered down when the new device is connected. In the second case we can see that there is a possibility of the system becoming alive at a state where more than one unconfigured device is connected to the communication network. All these devices will appear similar to the network. Since the devices are connected to a continuous bus, there is not a way of identifying the devices considering the physical placement. So our initial step would be to give the devices connected to the network a sense of physical location of themselves. In order to do that we introduced another line to the bus which is a cascaded select line as in Figure 1. With this newly added additional line, devices can be identified according to their physical positions relative to the network.

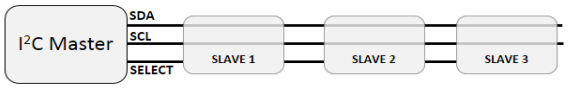

Figure 1: A single branch of the modified I2C bus.

Figure 2: A single slave node.

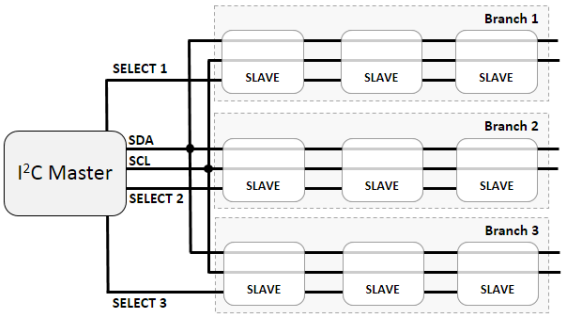

Figure 3: Multibranch network.

The select line that is introduced in addition to the clock and data lines used in the standard I2C bus is different from the SPI select pin. Here, only one select pin is connected to the master, assuming

only a single branch of connections is there. Each of the other select lines are only connected with the adjacent devices. Each device has a select line coming into it and another one going out of it as shown in Figure 2. In addition, if the master is capable, we can even have several branches of slaves with the master having separate select lines for each branch as in Figure 3. The master needs to be capable of controlling these lines independently. With the branching, the devices can be located by the branch and their position in the branch. The sense of a sequence has been introduced into this bus by this newly introduced cascaded select line.

4. ADDRESS ASSIGNMENT

The basic idea is for the master to get involved in assigning addresses to the devices connected to the communication bus one by one. The master will select a device and then will assign a valid address to the device and move on to the next device. This hopping from one device to the other has been realized using the select line. The modules individually needs to be capable of assigning themselves the address assigned by the master. Initially we will consider a scenario where the system is powered on with the devices plugged in. The devices need to have volatile memory of their addresses since otherwise they might not be usable in the subsequent attachments. The slave devices will be programmed in such a way that they all will have the ability to change their address to a predefined reserved address (Add1) when the select in line is activated.

4.1. **Master's Workflow**

The master's workflow is very simple. It will have a set of addresses that are to be assigned to the slaves.

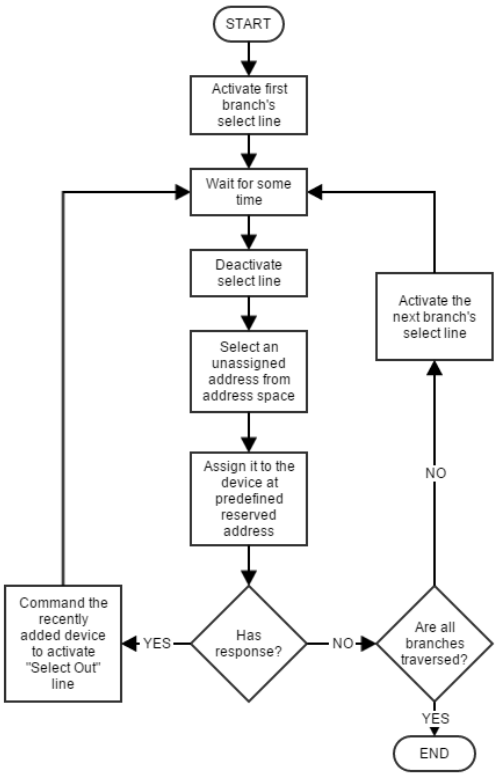

Figure 4: Workflow of the master.

The master will first select a branch from the set of branches and activate the respective select line and wait a short time giving the device time to respond. By this time the use of the select line is expired. So it can be deactivated. Then it will write to the predefined reserved address (Add1) all the data needed by the slave to assign its address. At this moment, the device has been successfully configured to be part of the network. Any application specific data can then be exchanged as if communicating with a standard I2C slave using the newly assigned address. Now the master will have to move to the next level of the branch. This is done by commanding the recently added device to activate its select out line. This procedure is repeated until there is no response from the predefined address (Add1). When there is no response, it means that the master has reached the terminal of that branch. Then the master moves on to the next branch and repeats for all the branches. After the master has finished all the branches the system will have a fully configured set of slaves. The workflow of the master can be demonstrated as shown in Figure 4.

4.2. **Slave's Workflow**

The slave's workflow is very simple compared to the master since master is responsible for the coordination whereas the slave simply executes the requests from the master. Slave will have a program which initially sets its I2C address into a predefined address (Add0) which is not used in any case other than detecting unconfigured devices. Then it will wait for its select in line to be activated. When the select in line is activated, the device changes its address to the predefined address (Add1) that the master uses to communicate address data. The device will wait for the master to send address data. This address data will be used to change the slave device's address which will be used to address the device in the fully configured communication network. Any application specific data can then be exchanged upon the request of the master. After all these are done the slave will fall into the listening mode waiting for the master for any request. All the slave devices need to be able to activate their select out line upon the request of the master. The workflow of the master can be demonstrated as shown in Figure 5.

4.3. **Hot Plugging of Devices**

When the device configuration changes while the system is powered on, the situation is a little bit different. Identifying a missing device is trivial since any request from the master will not be acknowledged. The method used to identify newly connected devices would be dependent on the application. In project SiFEB, the main module stays idle most of the time in the mode which anticipates hot plugging of modules. The process followed there is checking for the availability of the default address used by the devices. If that is not the case, the terminal devices can be instructed to activate their select out line and see if the predefined address (Add1) is available. If such a device exists, that device needs to be configured. Polling for this will work in this scenario. But in certain cases, the select line might be used in order to convey the message to the master that a new device is connected.

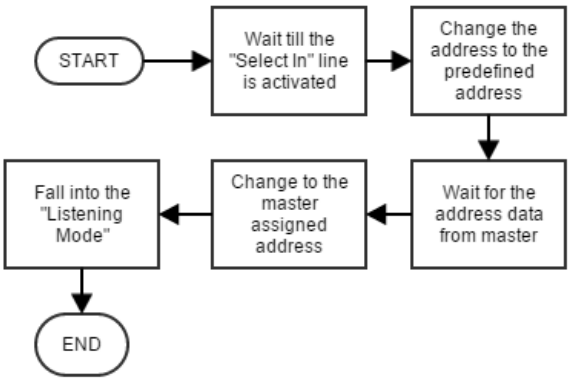

Figure 5: Workflow of a slave.

5. CONCLUSION AND FUTURE WORK

Since this work has been carried out as supporting work for a project, some aspects of the work has been specific to the application. For example, the auto detection of hot plugged devices can be achieved through an interruption process rather than through polling. Polling method can be used only in cases where the master is kept idle for a considerable portion of the uptime. Or else the terminal modules can be assigned the responsibility to be alert on these new additions.

With the success of this implementation, the requirement for modular communication in the project SiFEB was fulfilled at the prototyping stage. The actual implementation of this protocol in SiFEB actually is a derivation of this where an optical signaling mechanism was used as the select line. As future work, this solution can be improved by making it more generalized and by exploring the possibility of eliminating the need of a separate select line thus reducing the total width of the communication bus.

6. REFERENCES

1) S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita and S. Kokaji, "M-TRAN: self-reconfigurable modular robotic system," Mechatronics, IEEE/ASME Transactions, vol 7, no. 4, pp. 431–441, Dec. 2002.

2) J. Sadler, K. Durfee, L. Shluzas, P. Blikstein, "Bloctopus: A Novice Modular Sensor System for Playful Prototyping," in Proc. of

the 9th International Conf. on Tangible, Embedded and Embodied Interaction, 2015, pp. 347–354.

3) NXP Semiconductors., “UM10204.” I2C-Bus Specification and User Manual. 2014. [Online]. Available: http://www.nxp.com/documents/user_manual/UM10204.pdf.

4) Microchip Technology Inc., “Section 23. Serial Peripheral Interface (SPI).” PIC32 Family Reference Manual. 2011. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/61106G.pdf.

5) Texas Instruments., “1-Wire Enumeration.” 2013. [Online]. Available: http://www.ti.com/lit/an/spma057a/spma057a.pdf.

6) Texas Instruments., “Introduction to the Controller Area Network (CAN).” 2008. [Online]. Available: http://www.ti.com/lit/an/sloa101a/sloa101a.pdf

7) Universal Serial Bus Specification. Available: http://sdphca.ucsd.edu/Lab_Equip_Manuals/usb_20.pdf

8) Dallas Semiconductor.“Fundamentals of RS–232 Serial Communications.” 2001. [Online]. Available: http://ecee.colorado.edu/~mcclurel/dan83.pdf.

9) Freescale Semiconductor., “SMBus Quick Start Guide.” 2012. [Online]. Available: http://cache.freescale.com/files/32bit/doc/app_note/AN4471.pdf.

10) Philips Semiconductors., “AN10216-01 I2C Manual.” 2003. [Online].

11) Available: http://www.nxp.com/documents/application_note/AN10216.pdf.

12) Arduino., “Wire Library.” 2015. [Online]. Available: http://arduino.cc/en/reference/Wire.

13) H. Senaratne, P. Gunatilaka, U. Gunaratna, Y. Vithana, C. de Silva and P. Fernando, “SiFEB - A Simple, Interactive and Extensible Robot Playmate for Kids,” in Proc. 4th International Conference on Artificial Intelligence with Applications in Engineering and Technology, 2014.

14) M. Barenys, R. Faust and J. Goodwin,"Dynamically Allocating I2C Addresses Using Self Bus Switching Device." U.S. Patent 6,745,270, issued June 01, 2004. [Online]. Available: https://www.google.com/patents/US6745270