ATOMI II - Framework for Easy Building of Object-oriented Embedded Systems

Tero Vallius, Juha Röning
Computer Engineering Laboratory, University of Oulu
P.O. box 4500, FIN-90014 Oulu, Finland
tero.vallius@ee.oulu.fi, juha.roning@ee.oulu.fi

Abstract

Traditionally, an embedded system design process demands a considerable amount of expertise, time and money. This makes developing embedded systems difficult for many companies, and in research facilities it hinders the testing of new research results with real embedded systems. We have earlier presented an easy and fast embedded system development concept based on embedded objects. The embedded object concept (EOC) utilizes common object-oriented methods used in software by applying them in combined Lego-like software-hardware entities. This concept enables fast prototyping with target hardware, incremental device development and high-level device building for non-experts.

The EOC requires a modularly extendable architecture along with mechanical and technical definitions in order to enable physical and electrical interconnectivity with versatile signaling between embedded objects. This paper presents the Atomi II framework, which is our solution for this need. The framework has been tested and implemented with so-called Atomi objects.

1. Introduction

1.1 Motivation

Traditionally, embedded system development consists of the following phases [1]:

- requirement specification,
- partitioning of the design into its software and hardware components,
- iteration and refinement of the partitioning,
- independent hardware and software design tasks,
- integration of the hardware and software components,
- product testing and release, and
- on-going maintenance and upgrading.

The problem with this process is the need of expertise. Usually the system is built from individual components, which provides a huge amount of different possibilities for implementation, thus enabling the designer to create tailored solutions for the systems at hand. At the same time, considerable expertise in embedded system design is required. Iterative or incremental development is not a feasible approach in electronics, since once the hardware is made, it is very expensive to modify. Usually, robust simulation and prototyping is needed in the partitioning phase to confirm the partitioning and component selection decisions. Still, there is room for errors as the prototyping hardware is often just an emulation of the actual implementation hardware, and often the expected electrical characteristics of the design are based on experience with previous projects.

The goal of our research is to make designing embedded systems easier. To reach this goal we have proposed the embedded object concept (EOC), which approaches this problem via object-oriented methods [2]. The EOC is based on embedded objects. Embedded objects are small printed circuit boards (PCBs) that contain both the hardware and software of an object, thus being complete functional entities. Each object contains at least one bus interface and some functional parts, for example an AD-converter or a motor driver. Each object can be connected to any other object, like Legos. These objects are the building blocks of an embedded system. To build new devices with the embedded objects, one only needs to assemble suitable objects together and write some control code.

Object-oriented design principles can be applied for complex systems. Some objects have two or more

buses. These objects can be used to separate the buses between different object groups and to create new interfaces for them. These objects are called class objects, as they represent a new class consisting of the objects connected to them. By dividing the objects into separate buses, the subsystems can be encapsulated into new logical objects that consist of a group of other objects, which is the basic principle of object-oriented complexity growing [3]. This principle produces architectures as depicted in Figure 1. The principles of this concept, the general architecture and our previous implementation have been published in [2], [4] and [5].

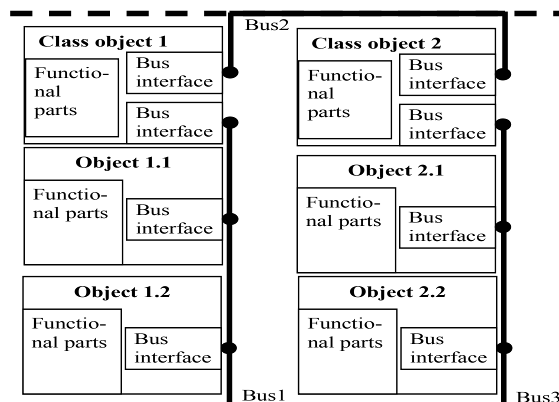

Figure 1 Architectural example of the EOC

The EOC enables fast prototyping of embedded systems. The EOC also provides easy modifications and enhancements of existing systems, thus providing the possibility for incremental development of devices. Furthermore, the architecture of the EOC and related fundamental techniques offer new aspects for reconfigurable and reusable hardware, environmental issues (recycling), test procedures in embedded system production, product maintenance, micro module technology, packaging technologies of integrated circuits (especially state-of-the-art reconfigurable system-on-chip solutions), embedded system co-design, high-level system development and automatic device generation.

The real challenge of this concept is to be able to create embedded systems that can compete in cost and efficiency with devices built with traditional methods. The architecture relies heavily on the interface between the objects, i.e. the physical and logical interconnection. This paper introduces a low-cost, versatile, general-purpose bus framework, some related techniques and physical definitions, which will enable this kind of system architecture.

1.2 Related work

Different object-oriented embedded system approaches exist. For example, the object-oriented methods have been applied to embedded system co-design in the MOOSE (Model-based Object-Oriented System Engineering) [6] and in the unified hardware/software presentation method [7].

Object-Oriented principles have been used for hardware design and hardware synthetization in SystemC [8] and Object-Oriented extensions to VHDL [9, 10]. Some more concrete approaches are OOPic, RoboBricks, SimmSticks and the Tower System. OOPics are PIC-microcontrollers with preprogrammed multitasking objects from a library of software objects, and they are mainly meant for robotic applications [11]. RoboBricks are quite similar to OOPic, but they are complete modules consisting electronics and software. RoboBricks create master-slave type of architectures and they are also meant for robotic applications [12]. SimmSticks are electronic modules that can be inserted into a motherboard to create computer type of embedded systems. They are aimed for hobbyists building quick custom devices. The SimmSticks do not contain any preprogrammed software [13]. Tower system consists of modules that include a foundation board and a set of layers to provide required functions. It is very similar to RoboBricks. The Tower system is meant for modeling a system topology [14, 15].

2. Atomi II Framework

The basic requirements for the framework are based on the ideology of the EOC. We have defined them as follows:

1) The system must be arbitrarily scalable to support architecture that has no fixed amount of objects on a bus.
2) The bus must support a virtual shared memory model for its logical interface. [2]
3) The bus should use a minimal amount of pins to also support small microcontroller units (MCUs).

The first requirement comes directly from the basic idea of the EOC, where the objects are meant to be connected to each other like Lego blocks to create new devices.

The second requirement refers to the virtual shared memory model, which was introduced in [2]. It is derived from object-oriented software, and we have defined it to be the basic communication interface between objects. This model presents the properties, method triggers and event setups of each object as a

variable table in a public memory space available for any object on the same bus. A variable of an object is accessed by the object number of the object and its variable index, as in an example C-code:

```
Object[2].Variable[1]=10;
```

In a typical computer system, each object is located in the same memory space. Since one physical shared memory for an arbitrarily modifiable system is difficult to implement, each object shares part of its internal memory (i.e. its variable table) via the bus. To exchange data via this method we use two functions, GET and SET. The virtual shared memory approach enables modular implementation, and the support for it is a basic requirement for the physical bus and the bus protocol.

The third requirement is directly related to current microcontroller packaging costs. The basic principle of the EOC is that the simplest object performs only one function. Thus, if a function is simple to implement, it requires only a very simple MCU to implement it. Often the simplest MCUs have only a few pins. Thus, if the bus requires too many pins, the MCU can not be necessarily optimally selected for its task, and too large an MCU must be taken. This adds to the cost of the system and reduces its general feasibility accordingly.

2.1 Extended requirements

A generation of embedded objects was created according to the basic requirements, but more requirements have emerged along with the research. In [4] and [5] we introduced our implementation of the first embedded objects, the first generation Atomi objects. The bus implementation of these first generation Atomis was based on a multi-processor communication mode on an asynchronous serial bus, which is a built-in feature of Atmel AVR series microcontrollers [16]. Though we created several functional applications with these Atomis, our studies showed several problems with the architecture, which were all closely related to the bus. To summarize the problems, they were:

a) The bus system required a processor for each object, which is not well suited for simple designs
b) The specialized bus type (multi-processor communication mode on AVR controllers) significantly narrowed the range of suitable components for this architecture
c) The bus speed was low in respect to the clock cycle usage of the processor, making for example polling unusable
d) Predetermined bus speeds were inflexible
e) Arbitration was inadequate; the EOA requires a reliable and fully modular arbitration method

The problem a requires an additional definition for the embedded object architecture (EOA). To support feasible simple designs, the bus must also support objects that do not have a microcontroller, but only some functional circuitry and/or connectors. Because of this, we have now defined two categories for the embedded objects: **active** and **passive**. An active object means the object can initiate a bus communication sequence, i.e. reserve the bus and use it (usually to send a data request). Passive objects cannot initiate the bus communication sequence, but only decode the address and enable its communication lines on an address match. Active objects usually contain an MCU and possibly some function-related circuitry, such as a motor driver or a network controller. Passive objects have only an address decoder and some function-related circuitry, such as I/O pins, an AD-converter or a Bluetooth device which can be used by an active object. Passive objects enable us to create single-processor systems with several inexpensive peripheral objects, thus reducing the cost of the system.

To address the problem b, the bus should be easily implemented with inexpensive electronics or with basic functionality of every MCU, i.e. general I/O pins. This combined with problem c and d suggests the use of an asynchronous flow-controlled basic parallel data bus, which can be easily implemented with general I/O pins with any MCU. On the other hand, using parallel bus requires a lot of IO-pins. Thus, also serial data methods should be available.

Problem e simply requires a reliable arbitration method, but due to the fundamental architecture, the arbitration must also be fully modular and freely scalable.

Based on these findings, we have defined the additional requirements for the bus:

4) The bus enables passive objects, i.e. I/O lines can be used for multiple purposes, not only a virtual shared memory model.
5) I/O lines required by the bus are not tied to any peripheral or MCU, allowing implementation with any brand or type of MCU
6) The bus data access time is very short, i.e. the number of clock cycles required to transfer data between objects is minimal, in order to make polling feasible and to make this architecture competitive with single-processor solutions
7) Data transfer is flow-controlled to enable communication between objects running in different speeds
8) Implementation is simple and inexpensive

The easiest solution would be to use some existing bus as such for interconnection method. The following chapter examines some existing bus types with respect to the requirements for the EOA.

2.2 Typical existing bus technologies

Serial buses can be divided into asynchronous buses (for example CAN [17] and LIN [18]) and synchronous buses (for example I2C [19]). All serial buses have most of the same problems as our first bus implementation regarding the EOA. The clock cycle to bus speed ratio is low, even though synchronized buses usually have a better ratio than asynchronous ones. They do not support passive objects because of the complexity of the logic for accessing objects, and because there are no extra I/O lines available for multipurpose use. MCU selection for active objects is practically restricted to those that have an integrated bus peripheral. The bus speeds must be predetermined in order to maintain compatibility. However, synchronous serial buses give some degree of flexibility to bus speeds, as the bus is synchronized by the clock signal. Still, the transmitter must always know the maximum speed of the recipient, which on its part reduces its feasibility for the EOA. The positive side of all serial buses is that they use only a few pins and some protocols have built-in arbitration methods.

Parallel data buses are commonly used as internal buses of processors [20], modular back plane systems [21, 22, 23] and memory interfaces [24]. Parallel buses commonly consist of data lines, address lines, arbitration lines and control lines. The fundamental problem of parallel buses is that they require many I/O lines. However, there are techniques for reducing the amount of needed lines, such as multiplexing the address and data into the same I/O lines [22], or using a serial bus for the address and arbitration in conjunction with a parallel data bus [25]. Still, parallel buses require vastly more I/O lines than serial buses. The positive sides of parallel buses are that they are fast and relatively simple. With respect to the EOA, a parallel bus can easily be implemented with general I/O –pins, and the clock cycle to bus speed ratio is good (depends on the protocol, though). There are also flow control and arbitration methods available for parallel buses [24].

According to our research on existing buses, there is no bus standard that has an optimal combination of features for the EOA as such. Also, existing arbitration methods require too many pins and too expensive logic for the EOA. For these reasons we have defined a hybrid bus system and developed a proprietary addressing and arbitration method. The next chapter introduces the Atomi II framework, which consists of an EOA-optimized bus definition, addressing method, arbitration method, physical definitions and rules for expanding the design.

2.3 Atomi II framework

Atomi II framework is a hybrid of existing techniques appended with a new addressing and arbitration techniques. It presents a compromise solution for the requirements or EOA.

The bus is a 9+3 bit parallel bus, called AtomiBus II. To minimize pin usage of the controlling MCU, all except one of the bus lines are shared between general purpose IO and addressing, data, or control functions. The key elements of the bus are the addressing and arbitration methods and analog connections. The bus contains the following I/O lines:

- shared 9-bit general purpose IO/address line (IO)
- shared general purpose IO/set line (SET)
- shared general purpose IO/acknowledge line (ACK)
- address set line (ADDR)

This makes a total of 12 lines. Additionally, the controlling MCU in an active object needs one I/O line for the arbitration logic (BUS_REQ). Each object does not need to use all the lines of the bus (see chapter 0). The only mandatory line is the ADDR-line. In addition to the control IO-lines, the bus has the following voltage lines:

- Two ground lines
- Operating voltage line, default 3.3 Volts
- Unregulated direct voltage line, which is meant for higher current devices, such as DC-motors. Voltage depends on used power source.

2.4 Bus reservation and arbitration

The schematic of the arbitration logic is presented in Figure 1. The bus is reserved when any of the control lines (ADDR, SET or ACK) are low. Correspondingly, the bus is free when all control lines (ADDR, SET and ACK) are high. The bus is reserved by enabling the bus reservation logic by setting the BUS_REQ signal to high impedance (input mode in MCU). When disabled, the BUS_REQ signal is actively driven low. This logic implements a proprietary daisy chain type arbitration method, which is modular, freely expandable and implements a geographical priority scheme. It is based on the capability of disconnecting the ACK line from the next object in the bus with 2:1 analog MUX gate, thus creating a daisy chain. The daisy chain is used to

provide the priority for simultaneous bus reservations simply by giving it to the object that is the first in line in the bus. The arbitration mechanism uses the control pins, thus saving the I/O lines on the bus. It requires one pin from the MCU for the BUS_REQ signal. The arbitration can be implemented with two 74-series logic gates according to the schematics in Figure 1.

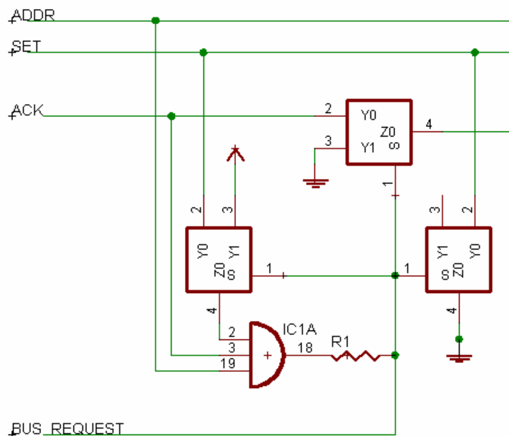

Figure 1 Atomi arbitration schematic

2.5 Addressing

Addressing is the most important operation in the AtomiBus II. The address lines are shared with general purpose IO-lines. The address (or object number) must be latched to the 8-bit bus in the beginning of any bus operation by setting the address on the IO lines and lowering the ADDR line as shown in Figure 2.

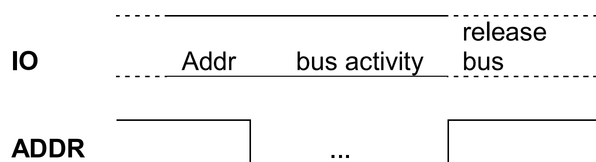

Figure 2 Address selection

The falling edge of the address line signifies that the address is on the bus. After latching the ADDR line low, the IO-pins are free to be used for any purpose. The ADDR line must be kept at logical zero during the whole bus transaction period, thus reserving the bus and signifying that the addressed object must remain online during this time. What happens in the bus activity phase can be different for different object types. For example, this period can be used to measure the input values of the I/O pins of a passive object, or active objects can transfer data with any protocol, serial or parallel.

The implementation for the addressing is simple and low cost. The address can be either an 8-bit data byte or each object can use one of the IO-lines as their own chip-select line. Both addressing types are compatible with each other, and they can both be used at the same bus at the same time. One line select method limits the amount of object on the bus to 11 objects, but it enables selecting several objects at the same time, which can be utilized in some applications. Additionally, it is a very low cost solution: it can be realized with a single latch gate as shown in Figure 3. Using an 8-bit address needs at least one logic gate more to decode the current address, but it enables 256 objects on one bus. For active objects it may be feasible to use 8-bit address since the addressing can be realized completely with software. Software realization, however, brings along timing constraints due to delayed response to falling edge of the ADDR line (see chapter 0).

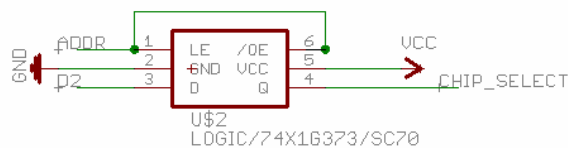

Figure 3 One line address recognition can be implemented with a single latch gate

2.6 Pin usage and data transfers

Due to the protocol independent addressing and analog IO lines, several different data transfer methods can be used for inter-object communication. Each object is required to be connected to only those bus lines that it uses. Typically the active object or objects that control the system must have connection to each line in order to access all objects on the bus, but the passive objects or objects with small MCUs use only those lines that are needed for their communications.

The only mandatory IO-line for each object is the ADDR-line. Any of the rest of the lines can act as data transfer and addressing lines. Thus, even a two pin connection to the bus possible, when for example a one wire serial data transfer is used, and the chip-select is chosen to be the same IO-line that is used for the serial data. Consequently, this bus system meets all the requirements that were previously stated, but not all at the same time. For example, to meet the requirements of the short data access time (requirement 6) and flow controlled data transfer (requirement 7) we have used the Motorola 68K 8-bit parallel data transfer protocol [24]. To meet the requirement of a very low pin count, different serial protocols can be used on the bus. We have tested SPI and asynchronous serial data (as in

RS232) with the bus in our passive Bluetooth and short range radio objects. To maintain the compatibility between objects, we have merely defined fixed IO-pins that are to be used with different protocols.

Since the bus lines are all analog, also analog signaling is possible within certain limits. The analog signal must not exceed the limits of 0 volts and the operating voltage, nor exceed the current limit of the bus connectors or track width. We have tested the analog connection by connecting a passive key matrix to the IO-pins. Furthermore, since there are 12 IO-lines available for use, several electronic modules, for example the popular HD44780 LCD-display, can be used through the bus by merely adding the latch gate for address recognition (meets requirement 4). Furthermore, this system is arbitrarily scalable, IO-lines are not tied to any specific protocol, and implementation is simple and inexpensive, thus meeting all the requirements for the EOA.

2.7 Logical interface

In the previous Atomi implementation all the objects realized the virtual shared memory interfaces that were accessed with general GET and SET functions. This was possible since all objects were active. In Atomi II framework, each active object still realizes the virtual shared memory model, since it can be realized over any physical data transfer method. Passive objects have their own access methods, which presents a problem for the easy-to-use aspect of our system. Our way around this is to have a driver set for each passive object, which is to be included into the software of each active object that is using the passive object in question. The driver interface is similar to active objects, i.e. data can be exchanged via similar GET and SET functions as with active objects. Only difference is that the functions for passive objects have a prefix in their function names, for example BTGET and BTSET for accessing Bluetooth device.

2.8 Physical definitions

Basic size of an Atomi is 48 mm x 14 mm. The 48 mm width is fixed but the 14 mm length can expand if required. Each board connects to each other so that the long sides are parallel to each other. The bus runs through each board via bus connectors. Figure 4 shows an example of two second generation Atomi-objects.

The board size is optimized according the size of a euro PCB panel, which is often used in production. By connecting the boards in parallel, the Atomi-objects will form a prototype of a PCB layout that can be produced as a single PCB. This has also been considered in the bus connector selection. The bus connectors fit directly to the bus tracks without a separate footprint or tracks leading to the footprint. Thus, the layout of Atomi-objects in an application can be manufactured as a single board by copying the layouts together as such according to their positioning in the prototype.

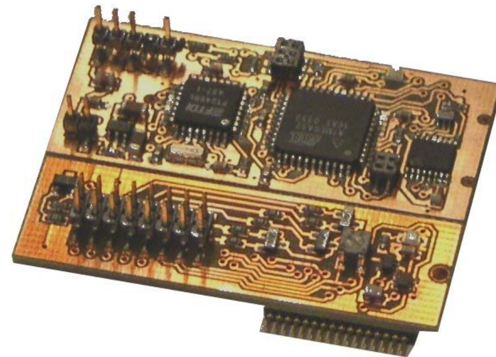

Figure 4 USB- and IO-Atomi

3. Results

3.1 Atomi-objects

We have created several different Atomi-objects with the Atomi II framework. As results we introduce the following Atomi-objects: USB-Atomi, ADC-Atomi, DC-Motor-Atomi, IO-Atomi, Power-Atomi and GPS-Atomi.

USB-Atomi represents both an active Atomi-object and a class object as it has two buses, USB and AtomiBus II. It runs with Atmel ATMega 32 MCU at 8 MHz and uses software address recognition. It contains arbitration circuitry. It is connected to every pin on the bus. It is capable of M68K 8-bit data transfer method, i2c, SPI, synchronous or asynchronous serial transmissions (USART) and direct IO-pin value reading.

ADC-Atomi is another active object with same MCU, address recognition, data transfer and arbitration configuration as USB-Atomi.

DC-Motor-Atomi represents a passive object created with relatively small MCU, ATTiny2313. It runs at 4 MHz and also uses software address recognition. It uses M68K 8-bit data transfer, and connects 11 pins to the bus (8 for data 2 for flow control and the ADDR-line).

IO-Atomi is an example of a passive object with analog connection to the bus. It has IO-pins connected to the bus in key matrix style [27].

GPS-Atomi represents a passive Atomi that uses its own asynchronous serial interface to connect to the AtomiBus II. It serves as an example of how existing modules can be converted into Atomi II framework

USB-, ADC- and DC-Motor-objects use M68K type flow controlled 8-bit parallel data transfer. The tests were run at 3.3 volt operating voltage.

3.2 Addressing

The one line addressing method was tested with USB-Atomi and IO-Atomi. There the IO-Atomi has one of the general purpose IO-lines as chip-select line connected to the latch gates D-input. Figure 5 shows how the chip-select line goes up on the falling edge of the ADDR-line. The response time is less than 10 nanoseconds. Our oscilloscope was not fast enough to give the exact time.

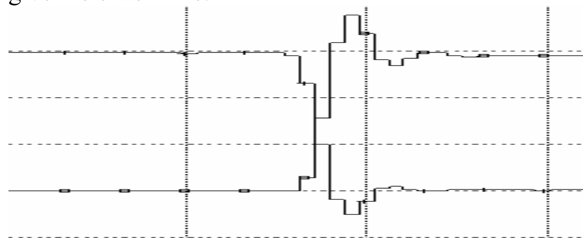

Figure 5 One line addressing. Chip-select line is the line with + signs and ADDR-line with small rectangles. Time grid is 100ns and voltage grid 1 volts.

Software implementation of the address recognition was tested with USB-, ADC- and DC-MotorAtomi-objects. The address recognition was implemented via an interrupt routine triggered by the ADDR-line. This presents a timing constraint: the address must be kept on the IO-line long enough for even the slowest object to be able to check it. The slowest object here is DC-Motor-Atomi, running at 4MHz. The response time for this object is 8 - 12 clock cycles, which corresponds to maximum of 3 microseconds. Thus the address must be kept on the IO-lines for 3 microseconds after lowering the ADDR-line with this configuration. The system was tested to be fully functional by first addressing the DC-MotorAtomi and after 3 microseconds changing the data on a bus to another value. The software address recognition is also used in the test application, the robot (chapter 3.5).

3.3 Analog IO on the bus

The analog connection was tested with USB-Atomi connected to an IO-Atomi. USB-Atomi contains the library software of the IO-Atomi and it polls the IO-lines of the IO-Atomi at approximately 50 kHz frequency. To test if the key matrix of the IO-Atomi corrupts the data for other data transfer on the bus, we connected also the DC-Motor-Atomi to the bus at the same time. Additionally to test driving LEDs with the key matrix of the IO-Atomi, we set four pins to serve as key matrix inputs and other four pins to drive LEDs. Test software was made into a PC to control the new device. The test software read the key values, changed the led outputs and read the name variable of DC-Motor-Atomi in a continuous loop. The test did not show any faults. Each data transfer was successful and the key matrix successfully read key input and drove LEDs. Figure 6 shows the test setup.

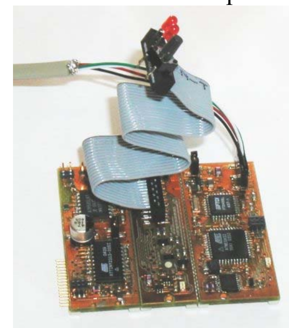

Figure 6 Analog bus test setup with IO-, DC-Motor-, and USB-Atomis.

3.4 Passive object

An example of a passive object is a GPS-object (Figure 7). Its Atomi-board consists basically of the latch gate addressing chip with its chip-select output connected to a 4 x analog switch gate (regular 74x4066). Via this circuitry it connects Lassen iQ GPS-module to the bus. [28]

Figure 7 GPS- and USB-Atomi

The module was tested from PC via USB-object. The USB-object was incremented with software that reads the current longitude and latitude information and sets them available for reading through USB-interface. The system was fully functional.

3.5 Two wheel standing robot application

As an example application of using the Atomi II framework we have made a two wheel standing robot (Figure 8). It consists of a body that has two wheels and a castor wheel attached to a joint. The joint does not support the robot, but it measures the tilt angle of the robot. The wheels are driven by DC-Motors with encoders. For each DC-Motor there is a DC-Motor-Atomi controlling it. The standing angle of the robot is measured by a simple potentiometer attached to the joint of the castor wheel pole. The tilt measuring potentiometer is attached into an ADC-Atomi. The robot has a battery pack, which is connected into a Power-Atomi. The system is controlled by the USB-Atomi, which contains balancing software in addition to the basic USB-Atomi software. It encapsulates the robot control software to one bigger object, whose interface is shown through the USB-connection. This interface exposes only the balancing parameters of the robot to be adjustable.We used a PID-control algorithm to keep the robot balanced. The Atomi-objects operated successfully in this robot

Figure 8 Two wheel robot test case

4. Discussion

The presented framework meets the requirements set by the EOC. It is a flexible framework for Lego-like building of new systems. The electronics required by this framework are inexpensive, and many electronic modules on the market can be easily integrated to this framework. Thus this framework suits very well for the EOC. With this framework the EOC will introduce a lot of benefits, such as:

- clear object-oriented architecture
- easy maintainability
- easy modifications and reconfiguring possible
- virtually unlimited expansions to existing designs
- re-use of existing objects
- enables incremental design and testing process
- shortened design time

However, the framework of the EOC has also some inherent downsides. The idea of building blocks requires a bus and the blocks that connect via the bus. Due to the block idea, this system obviously cannot produce highly integrated single chip devices. Furthermore, a bus consumes more power than highly integrated systems, leaving this framework unsuitable for very low power solutions, for example a watch. The best target for this kind of a framework is in replacing computers in control applications, for example in robotics or automation industry. Particularly usable this framework is in research and development phase of embedded systems. In R&D, different device setups and algorithms must be tested on prototypes, and the prototypes should be quickly and easily reconfigurable, modifiable or extendable. Often the prototypes are built on starter kits or prototyping boards, which contain an MCU with some amount of fixed peripherals. There the Atomi II framework provides a far more flexible alternative provided that the bus based architecture is suitable for the application.

5. Conclusion

This paper presented the Atomi II framework which is specially tailored for the Embedded Object Concept. The characteristics of this framework are versatility, support for passive objects via a simple low-cost addressing scheme, suitability for microcontrollers, minimal use of I/O lines via line sharing, full modularity, and simple arbitration. The bus was tested in respect to its suitability for the EOC. The bus provides a feasible platform for the EOC.

6. Acknowledgments

This research was partially funded by Infotech Oulu Graduate School, the Finnish Academy and the University of Oulu, Robotics Group.

7. References

[1] A. Berger: *Embedded Systems Design – An Introduction to Processes, Tools, and Techniques*, CMP Books, Lawrence, Kansas, USA, 2002.

[2] T.Vallius, J. Haverinen and J. Röning: "Object-Oriented Embedded System Development Method for Easy and Fast

Prototyping”, *International Conference on Machine Automation 2004 Nov 24- Nov 26, proceedings*, Osaka, Japan.

[3]J. Martin and J.J. Odell: *Object-Oriented Analysis and Design*, Prentice Hall, 1992.

[4] T. Vallius and J. Röning: “Implementation of the “Embedded Object” concept and an Example of Using it with UML”, *6th IEEE International Symposium on Computational Intelligence in Robotics and Automation, proceedings*, Espoo, Finland, June 27-30, 2005.

[5] T. Vallius and J. Röning: “Embedded Object Architecture”, *8th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, proceedings*, Porto, Portugal, August 30th - September 3rd, 2005.

[6] M. Edwards and P. Green: “An Object-oriented Design Method for Reconfigurable Computing Systems”, *Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings*, 27-30 March 2000, Page(s): 692 -696.

[7] S. Kumar, J. H. Aylor, B. W. Johnson and WM. A. Wulf: *The codesign of Embedded Systems: A Unified Hardware/Software Representation*, Kluwer Academic Publishers, 1996.

[8] E. Grimpe and F. Oppenheimer: “Object-oriented high level synthesis based on SystemC”, *Electronics, Circuits and Systems, 2001. ICECS 2001. The 8th IEEE International Conference on, Volume: 1*, 2-5 Sept. 2001, Page(s): 529 -534 vol.1.

[9] W. Nebel and G. Schumacher: “Object-oriented hardware modelling-where to apply and what are the objects*?”, Design Automation Conference, 1996, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96,* European, 16-20 Sept. 1996, Page(s): 428 -433.

[10] G. Schumacher and W. Nebel: “Object-oriented modelling of parallel hardware systems”, *Design, Automation and Test in Europe, Conference, 1998, Proceedings*.

[11] Object-oriented programmable integrated circuits, http://www.oopic.com . [Accessed 22.7.2005]

[12] The RoboBricks Project, http://www.gramlich.net/projects/robobricks/ [Accessed 22.7.2005]

[13] The SimmSticks, http://www.simmstick.com/ [Accessed 22.7.2005]

[14] T. Gorton: Tangible Toolkits for Reflective Systems Modeling. Masters of Engineering Thesis. Massachusetts Institute of Technology. May 21, 2003.

[15] The Tower System Web-pages, http://gig.media.mit.edu/projects/tower/ [accessed 15.2.2005]

[16] Atmel: Atmega32 datasheet pages 155-156. http://www.atmel.com/ [accessed 29.11.2005]

[17] CAN specification Version 2.0, Bosch, 1991.

[18] LIN www-pages, http://www.lin-subbus.org/ [accessed 15.2.2005]

[19] The I2C-bus specification Version 2.1, Philips, 2000.

[20] AMBA specification Rev. 2.0, ARM, 2001

[21] IEEE standard for a simple 32-bit backplane bus: NuBus, ANSI/IEEE Std 1196-1987 , 8 Aug. 1988.

[22] PCI specification Rev. 2.1, PCI-SIG, 1995.

[23] Plug and Play ISA specification Version 1.0a, Intel Corporation and Microsoft Corporation, 1994.

[24] M68000 8-/16-/32-Bit Microprocessor User’s Manual, Rev. 8, Motorola, 1994.

[25] J. M. Bennet, P.V. Chefurka: MicroNet: a low-cost local area network for microcomputers, Proceedings of the 1983 ACM SIGSMALL symposium on Personal and small computers, San Diego, California, USA 1983.

[26] (This reference will reveal the writers identity)

[27] Microchip: Application Note 529 - Multiplexing LED Drive and a 4x4 Keypad Sampling. http://www.microchip.com [accessed 29.11.2005]

[28] Trimble: Lassen iQ GPS Module Datasheet, http://www.trimble.com [accessed 29.11.2005]