Journal of Embedded Systems, 2015, Vol. 3, No. 1, 21-27
Available online at http://pubs.sciepub.com/jes/3/1/4

DOI:10.12691/jes-3-1-4

BlueSteps: A Bluetooth Based Stepper Motor Control System

Ifrah Jaffri[1,*], Zeeshan Nafees[1], Shoaib Zaidi[1], Oliver Faust[2]

[1]School of Science and Engineering, Habib University, Karachi, Pakistan
[2]Electrical, Electronic Control Engineering, Sheffield Hallam University, Sheffield, England, UK
*Corresponding author: ifrahjaffri@yahoo.com

Abstract Wireless systems are widely used as a networking technology for the Internet of Things (IOT). Although they were initially designed for voice communication systems, they can be used to exchange control commands and data between machines. In this paper, we present the BlueSteps system which gives a user wireless control over stepper motors. The BlueSteps hardware incorporates a Field Programmable Gate Array (FPGA) coupled to a Bluetooth module and a custom build driver circuit. The FPGA hosts a micro-controller and the control logic for the stepper motors. The Bluetooth module establishes a wireless connection between a mobile device and the micro-controller. Apart from the general systems design, we also conceived the User Interface (UI) software and a driver circuit for the stepper motors. The combination of custom software and custom hardware gave us the flexibility to create a versatile wireless stepper motor control system, which can be used in a wide range of applications.

Keywords: *stepper motors, bluetooth, field programmable gate array, remote control, custom intellectual property cores, custom driver circuit*

Cite This Article: Ifrah Jaffri, Zeeshan Nafees, Shoaib Zaidi, and Oliver Faust, "BlueSteps: A Bluetooth Based Stepper Motor Control System." *Journal of Embedded Systems*, vol. 3, no. 1 (2015): 21-27. doi: 10.12691/jes-3-1-4.

1. Introduction

The Internet of Things (IOT) is the result of a recent idea where objects become smart or indeed smarter through microprocessing and networking [1]. Such smart objects take measurements, do processing and communicate the resulting data through the network [2,3]. It is up to either a central instance or a distributed processing system to make sense of that data [4]. One of the goals of that technology is to remove human decision making almost completely [5]. For mature IOT systems, human interaction happens only on a very high level, where machine based decision making is incapable of resolving a choice. However, such a scenario implies that routinely choices are resolved and actions need to be executed. Actuators are devices which translate decisions into physical actions.

Stepper motors are a special type of actuators: they trans- late electronic commands into precise rotary motion. These electro-mechanical devices are widely used for motion control, because of their low cost and their ability to manipulate a connected electromechanical system [6]. The motor shaft moves in discrete step increments when electrical pulses are applied to it. Stepper motor technology is advantageous in applications where we need to control rotation angles, speed, steps and position. They can work in open loop control, i.e. no feedback channel is necessary. Furthermore, these motors are known for their high reliability and low maintenance. However, stepper motors need a dedicated control unit and they need sophisticated driver circuits which can deliver high currents. Using stepper motors for IOT applications implies that the dedicated control circuit is networked.

The BlueSteps system addresses the need for networked stepper motor control. The system offers remote control for up to two stepper motors. The remote control channel is estab- lished, via a Bluetooth link between a user centric device, such as smart-phone, tablet or PC, and dedicated control hardware. A user controls direction, speed and the number of steps with a User Interface (UI) on the Bluetooth terminal. To establish that functionality, we designed and manufactured a circuit which boosts the control signals from a dedicated embedded control system and interfaces to a Bluetooth module. We configured Field Programmable Gate Array (FPGA) logic as an embedded control system. To be specific, we instantiated a standard microcontroller with peripherals and a custom built stepper motor driver module in the FPGA. The microcontroller runs custom software which generates the UI and at the same time interfaces to the dedicated stepper motor driver module. To integrate these ideas, we prepared software which enabled communication between these components. The BlueSteps system links stepper motors and mobile devices. Such a link is very important for turning the decisions of an IOT system into actions.

The material of the paper is organized as follows. Section II details the individual components used to establish the BlueSteps system. That section presents a block diagram which provides a system overview and it

structures the module description. The discussion section details specific design decisions and it sets the BlueSteps system into a wider context of IOT systems. The paper concludes with Section IV.

2. Materials and Methods

The project is an integrated system which includes compo- nents and blocks as highlighted in Figure 1 and Figure 2. Figure 1 shows the physical setup of the BlueSteps system. Its upper half shows two connected stepper motors. One motor sits at the base and turns a shaft. The second motor is mounted on the turning shaft and it moves a pointing arm in an angular motion. The stepper motors receive control signals from a driver circuit. The driver circuit boosts the control signals, such that they deliver enough power to drive the stepper motors. That driver circuit is shown in the middle of Printed Circuit Board (PCB). The Darlington driver components are arranged in two distinct columns. The lower portion of the Figure shows the Spartan FPGA LX9 microboard mounted on the PCB. The Bluetooth module is plugged in on the left side of the PCB. That module ensures the up-link between the FPGA logic and a user centric Bluetooth enabled device.

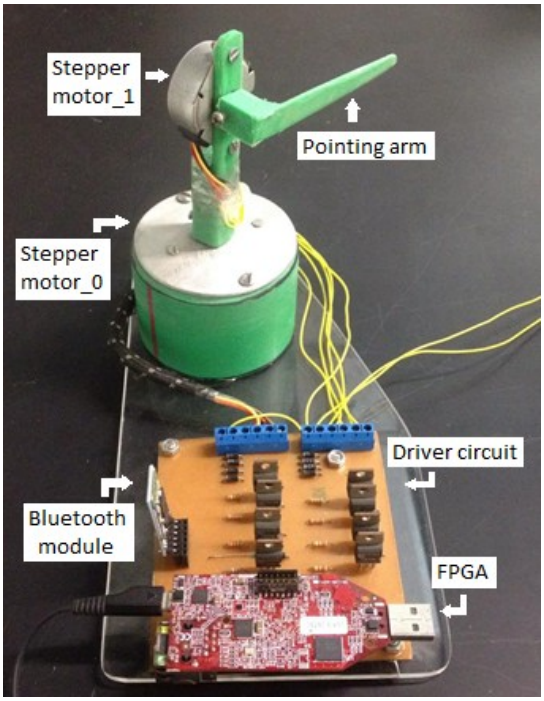

Figure 1. Physical setup of the BlueSteps system

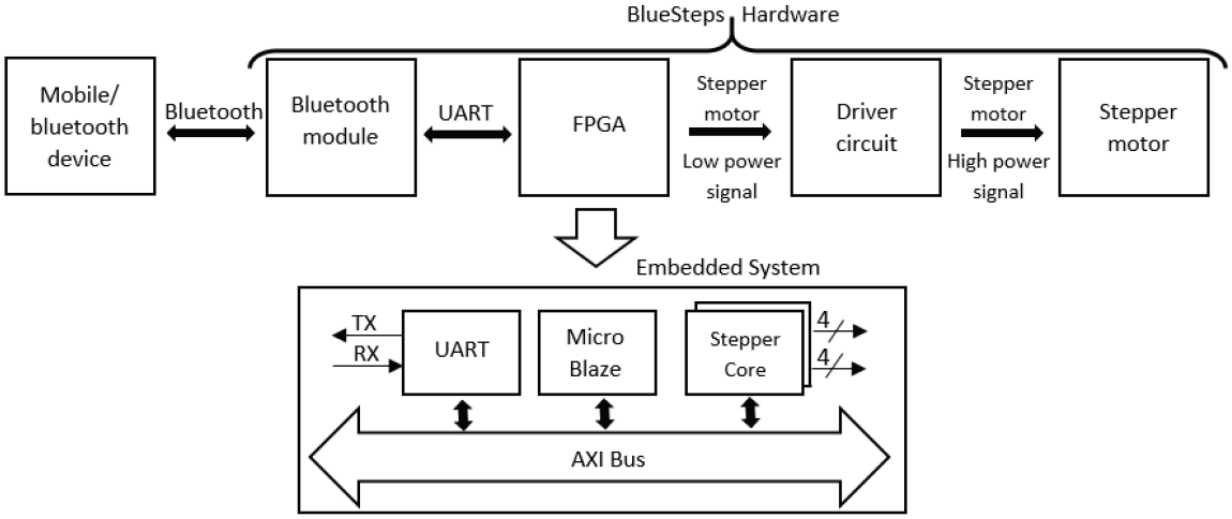

Figure 2. Block Diagram of the BlueSteps system

Figure 2 shows the block diagram of the BlueSteps system. The diagram details both the blocks which establish the mod- ule functionality and the communication links between them. The mobile device is connected to the BlueSteps hardware via Bluetooth. The Bluetooth module translates the wireless communication to wire bound Universal Asynchronous Re- ceiver/Transmitter (UART) signals. These wire bound signals are transmitted and received by the embedded system, imple- mented in FPGA logic. That logic establishes the BlueSteps core functionality by instantiating a MicroBlaze (MB) mi- crocontroller with peripherals as well as a custom stepper Intellectual Property (IP) core. Section II-A describes the functionality of these components in greater detail. Actuator control is exercised when the stepper IP core sends low power control signals to the driver circuit on the PCB. That circuit boosts the control signals such that high power signals are sent to the stepper motors in order to ensure proper operation of such a demanding load.

The Well-Known HC-05 Bluetooth Module Was Used To Es- Tablish A Host Control Interface (HCI) Bluetooth Link Between The Bluesteps Hardware And The Handheld Mobile Device. It Is A Bluetooth Serial Port Protocol Secure Simple Pairing (SSP) Device, Designed To Establish Transparent Wireless Serial Connections [7]. The HCI Uses The UART Protocol. That Pro- Tocol Establishes A Bi-Directional Communication Link Between The Bluetooth Module And The Embedded System Which Is Implemented In FPGA Logic.

A. Embedded System

To realize the BlueSteps embedded system functionality, we have used a Spartan-6 FPGA LX9 microboard [8] from Avnet. As such, an FPGA is a flexible logic Integrated Circuit (IC) which is configured

by a designer after manufacturing. The specifications, associated with the FPGA board, are given as:

- Board Vendor: Avnet
- Processor frequency: 100 MHz
- Processor: microblaze_0
- Local Memory Size: 8 KB
- Instruction Cache Size: 512 B
- Data Cache Size: 512 B

The BlueSteps embedded system incorporates the so called MicroBlaze (MB) soft processor and not an actual IC core. The cost effective Spartan-6 FPGA LX9 microboard is a good solution for exploring embedded systems with soft processing. The evaluation board comes with several built in systems and permits designers to focus on the software development. The Embedded Development Kit (EDK) provides the Xilinx Platform Studio (XPS) as a hardware development tool and the Software Development Kit (SDK) as an environment for writing and debugging software code. It includes peripherals and expansion interfaces as well [9].

IP cores are the key building blocks for FPGA based systems [10]. An IP core is a logic block or data that is used in making an application for a product. Its importance in interfacing with the devices has been highlighted in many scientific articles [11]. IP cores are instantiated directly in XPS and they can be controlled with the MB. The Xilinx development environment provides a number of IP cores, such as UART, Light Emitting Diode (LED), Ethernet, MB and Memory.

The embedded system is composed from a number of func- tional blocks, as shown in Figure 2. The individual functional components communicate via the Advance Extensible Inter- face (AXI) bus [12]. AXI 4-LITE communication protocol is being used in the BlueSteps system.

B. Stepper Core

The next step was creating our own IP core (Custom core) named as the Stepper IP core. The embedded system contains two instances of the Stepper IP core, labeled steppercore_0 and steppercore_1. Each of these instances controls one stepper motor with four control signals. Figure 2 depicts these two Stepper cores: each instance provides four control output signals. The XPS framework takes care that the Stepper IP cores get unique addresses. Hence, they can be addressed individually from the BlueSteps software program, executed by the MB. The Stepper IP cores are connected to the interconnect bus through AXI IP Interface modules, which provide a quick way to implement the interface between AXI 4 interconnect and the user_logic.

The two stepper motors work as an integrated unit. Stepper motors are not driven by normal excitation, they require a sequential excitation of adjacent phases. There are four stepper motor control signals for each motor. Figure 3 shows the timing diagram of the four stepper motor control signals. Such a signal configuration is called a stepping sequence, because the signals carrying the phases have a step like appearance in the timing diagram. The stepper motors, used in this project, require a so called half step sequence. Half step excitation means alternating single and dual phase operation, which results in steps that are half the basic step angle. Due to a smaller step angle, half step mode provides twice the resolution and a smoother operation [13]. Our system setup ensures that each stepper motor receives independent control signals. Its control behavior can be adapted by altering the control input from the mobile phone.

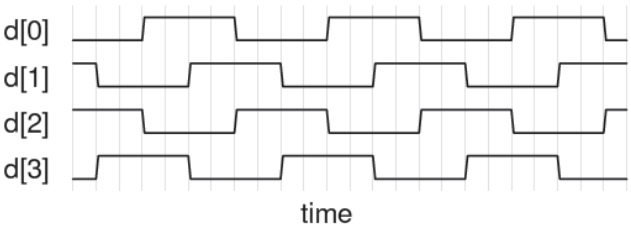

Figure 3. Timing Diagram of the stepper motor controlsignals

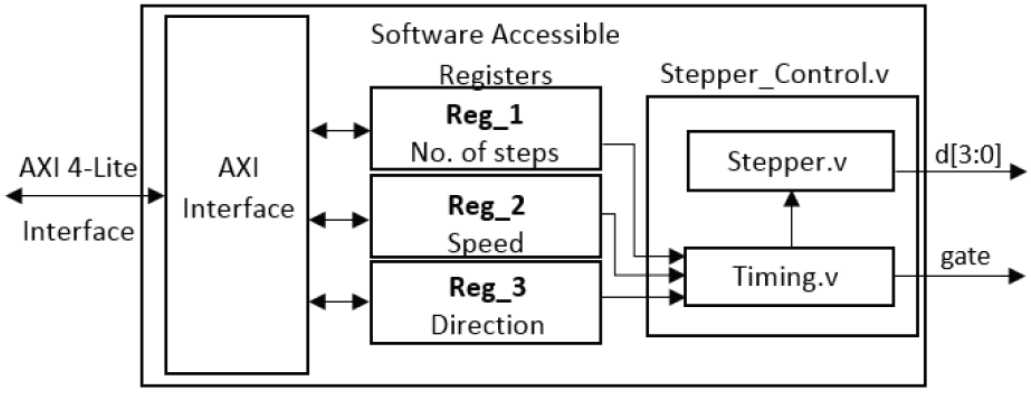

Figure 4. Block Diagram of the User_Logic.v

Figure 4 shows the block diagram of the User_Logic.v module, which resides in every Stepper IP core. That module contains three sub-blocks. The XPS tool creates a Hardware Description Language (HDL) template file. In our case, that template file was called User_Logic.v. We have extended that template file to define our peripheral. Software Acces- sible Registers and Stepper_Control.v in Figure 4 are these extensions. The sub-blocks are described as:

- AXI Interface: This block implements the AXI 4-Lite slave interface for register access and data transfer.
- Software Accessible Registers: Three Software Accessi- ble Registers are instantiated: they are implemented in the slave mode of User_Logic.v. The content of these registers is referred to as one buffer. The MB is a 32 bit processor, hence each of the register holds 32 bits. We can read and write to the Software Accessible Registers. So, the

communication link, between AXI interface and Soft- ware Accessible Registers, is bi-directional. The reading and writing of registers provide adequate functionality. When the peripheral is instantiated, we can access the register by reading and writing to the base address+offset. The base address is unique for each peripheral on the AXI bus. Out of three registers, one is specified for "Direction of the motor" with offset "0" [1], second is for "No. of steps" with offset "4" and third is for specifying the "Speed of the motor" with offset "8".

- Stepper_Control.v: That module contains the functional components of the stepper core. The Timing.v module receives the signals from the Software Accessible Registers and processes the user input. The user input is the number of steps, the time duration for one step and the direction. As soon as the command for direction is received by the Timing.v module, the motor starts to move. The movement is initiated through an output com- mand named "gate", which is input to User_Logic.v. The stepper motors can process only one user command at a time. If the motor/motors are already in motion then it cannot process a new instruction before the current task is accomplished. Hence, the "gate" signal takes care of that issue.

The Stepper.v module receives information from Timing.v. It is responsible for generating the half step control signals for the stepper motor. The four bit wide "dout" register drives these four signals and sends them to the User_Logic.v module. These signals are generated according to the movement, speed and direction of each motor. The information for movement, speed and direction is being provided to it from the Timing.v module.

Stepper_Control.v is the top-level module for the components discussed above. The top-level module sends all input and output signals to the User_Logic module.

C. User Interface (UI)

The previous section described the embedded system hard- ware setup. In this section, we detail the software which controls the hardware. The first step was to import the hard- ware IP configuration to the SDK environment. Within that environment, we used the C language to craft the BlueSteps control software [14].

The BlueSteps control software provides the UI and it translates the user input to commands for the hardware IP. For this paper, we adopt a functional description of the software rather than a source code based description. The description starts with powering up the BlueSteps system. Once there is a stable clock on the MB, the control software takes over and publishes the initial UI main menu over the Bluetooth link to the mobile device.

For the sake of understanding, we consider a specific scenario. We assume that user wants to turn stepper motor 0 10 steps clockwise and the pointing arm attached to stepper motor_1, 10 steps anti-clockwise. Figure 5 shows the dynamic communication between the MB and the Mobile device, which is necessary to accomplish that task. When the Bluetooth connection is established for the first time after power up, the MB sends the main menu information to the Mobile device, that action is symbolized by an arrow from the MB to the Mobile which is labeled "Main Menu". Once the main menu is published, the user can choose between seven options. Figure 7 shows a screenshot of a mobile phone displaying the main menu. To complete the task of moving the stepper motors, a user has to select option 5, which is "keyboard motor control". Therefore, the user keys in 5 and sends back the command "5" which is depicted as the arrow labeled "5", from Mobile to MB, as shown in Figure 5. As soon as "5" is received by the MB, a second menu, i.e. first Sub Menu, is communicated to the Mobile. Figure 8 shows the Sub Menu 1. That menu asks the user to specify the number of steps for the motor per key press. The number of steps defines the motor turning angle. The user then specifies it as "10" i.e. the motor turns 10 steps for each key press. Afterwards, Sub Menu 2, shown in Figure 9, appears on the Mobile screen. That menu lets the user choose which motor to move 10 steps in what direction. In our scenario, we assume the user presses "k" followed by "enter" in order to move stepper motor_0 in a clockwise direction and in return the MB performs that action, as shown in Figure 5. Then, to move the pointing arm, a user has to press "s" followed by "enter", which causes the MB to move stepper motor_1 in an anti-clockwise direction. Finally, the user presses "x" with "enter" to exit Sub Menu 2 and in response the MB publishes the Main Menu again. In this way, one cycle of the dynamic communication is accomplished. This is how the specified task is accomplished, a user can move the motors in any direction with whatever turning angle, by selecting the corresponding option issued to the Mobile device in Sub Menus 1 and 2.

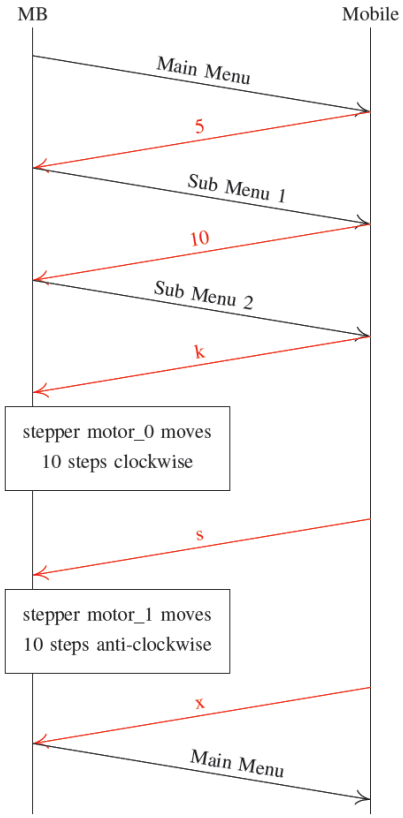

Figure 5. One cycle of dynamic communication between MB and Mobile phone. The messages shown in the diagram will cause stepper motor 0 to do 10 steps clockwise and stepper motor 1 to do 10 steps anti-clockwise

[1] Offset 0 means that the register is located at the base address.

D. Driver Circuit

The driver circuit is shown as the fourth block from the left in row one of the block-diagram, shown in Figure 4. It is acting as an interface between the embedded control system, which is instantiated in the FPGA, and the stepper motors. The purpose of the circuit is to boost both voltage and current of the control signals. The FPGA works at 3.3 V, but the stepper motors require 12 V input. So, an amplifier is required. We realized that amplifier using a Darlington pair in common emitter configuration, as shown in Figure 6. That circuit increases the 3.3 V control signal, from the FPGA, to 12 V for the stepper motor. The circuit also features two free wheel diodes D_1 and D_2. They were included to eliminate flyback, which is a sudden voltage spike, seen across the output of the stepper motor (inductive load) when it tries to stop, i.e. its supply voltage is suddenly reduced or removed. In the circuit, shown in Figure 6, the magnetizing current goes through the diode and dies out.

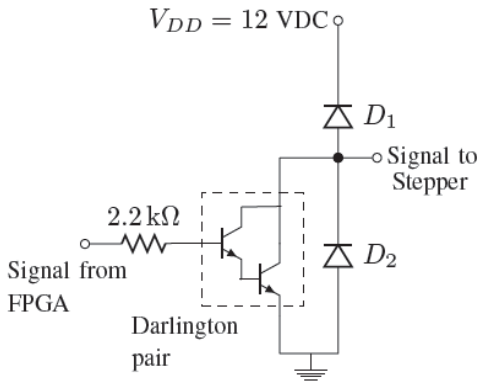

Figure 6. Driver circuit schematic

On the custom PCB, eight such driver circuits are integrated to achieve the complete driver functionality. The driver circuits provide a total of eight power signals to the two stepper motors. Four signals are provided to each motor, as discussed in Section II-B and Figure 3.

3. Discussion

The BlueSteps FPGA logic is designed as an embedded system. Embedded systems are hardware/software systems built into devices that are not necessarily recognized as com- puterized devices, but embedded systems do control the func- tionality of these devices [15]. The importance of embedded systems is growing continuously [16]. This importance comes from the fact that embedded systems are required to provide real-time responses. A real-time system is defined as a system whose correctness depends on the timeliness of its response [17,18]. Examples of these systems are aircraft flight control systems, sensor systems in nuclear reactors and power plants [19]. Embedded systems reside in nearly all of the electronic devices used today, from remote sensing over avionics to the health sector [20]. By now, it is almost impossible to build an electronic system without adding at least a small microprocessor which is controlled by software [21].

Although, the BlueSteps system is an embedded system, it has a number of advantages. But still, there are a few limita- tions associated with the system. Firstly, the BlueSteps system incorporates polling in the hardware control mechanism. The microcontroller polls the registers to check whether or not they contain new data. In the current implementation, hardware control and user interaction are mutually exclusive. This exclu- sivity results in a halt situation. If the stepper motor/motors are moving then the UI is frozen. Conversely, if the UI is active so the motor/motors cannot move. Hence, BlueSteps is a system where both software and hardware interfaces are not active at the same time. This leads a discontinuity in the buffer processing.[2] The discontinuity arises because of the approach we used. The Stepper IP core ensures that, when buffer is processed, it releases a specific bit as “0”. The software ‘looks’ for the bit value and if the bit value is “0”, then the software writes back to the buffer. This scenario causes the stepper motors to be non-functional for the duration it takes for the software to generate the new buffer values. Hence, in the worst case scenario, the motor stops until all buffers are serviced.

Although polling avoids the overhead of interrupt-based systems, polling is generally used only for low level hardware control. Interrupt driven systems are another technique in computer architecture, which has a number of advantages [22]. In short, in polling systems, the device priority is determined by how often it is serviced within the polling loop. Unlike polling systems, devices can be prioritized in interrupt driven systems. So, interrupt driven systems are fast and efficient and they would be better suited to address the halting problem than polling systems. Moreover, to keep the UI unfrozen all the time, there should be an Operating System (OS) [23]. An OS, such as the XILkernel [24], partitions the Central Processing Unit (CPU) processing time. Concurrent lines of execution open up when multiple software threads are instantiated and the OS is allocating CPU processing time to these threads. To solve the stepper motor staling issue of the BlueSteps system, the hardware control and the UI functionality have to reside in separate threads. These threads execute concurrently, which, for that application, is indistinguishable from executing hardware control and the UI functionality in parallel. In case there are still discontinuities in the stepper motor operation, a double buffering technique can be used. Double buffering uses two buffers to ensure a seamless handover of data between two functional entities [25]. Data in one buffer is being processed, while the next set of data is read into the other one.

4. Conclusion

In this paper, we present BlueSteps, a Bluetooth based remote control system for stepper motors. The system con- sists of the BlueSteps hardware and software. The hardware establishes a Bluetooth link between a mobile, user centric, device and an embedded system, instantiated in FPGA logic. Using that link, the embedded system provides the UI, which facilitates user control. To be specific, a user can set speed, number of steps and direction for each of the two stepper motors. Within the embedded system, two custom IP cores translate the commands into stepper motor signals. These signals are

[2] The content of the three user logic registers in the Stepper IP core is called as buffers, as discussed in Sub-Section II-B.

boosted by a driver circuit on a custom made PCB before they cause the stepper motors to move.

Our achievement was the completion of a whole cycle of control between a remote device and the actuator. To establish that control we designed and developed both the BlueSteps hardware and software. That design was challeng- ing, because it involved custom IP cores and custom PCB circuits. Furthermore, the software setup was also developed without supporting frameworks. Hence, the BlueSteps system is an example of hardware software co-design which generates control signals with a precision of 10 ns. As a consequence, our system facilitates the development of innovative actuator control strategies for state of the art IOT applications.

Remote controlling actuators is becoming one of most im- portant factors which determines the feasibility of introducing an IOT environment to manufacturing companies. The most likely way of interfacing with devices in the future will be IP; it is more flexible, scalable and compatible. One of the biggest issues will be to realize usable and accessible devices, with relevant functionality, for all kinds of users. Since this is an ongoing field of investigation, the results of the BlueSteps project are likely to be worthy of further analysis.

Acronyms

AXI	Advance Extensible Interface
CPU	Central Processing Unit
EDK	Embedded Development Kit
FPGA	Field Programmable Gate Array
HCI	Host Control Interface
HDL	Hardware Description Language
IC	Integrated Circuit
IOT	Internet of Things
IP	Intellectual Property
LED	Light Emitting Diode
MB	MicroBlaze
OS	Operating System
PCB	Printed Circuit Board
SSP	Secure Simple Pairing
SDK	Software Development Kit
UART	Universal Asynchronous Receiver/ Transmitter
UI	User Interface
XPS	Xilinx Platform Studio

References

[1] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy, "Smart objects as building blocks for the internet of things," *Internet Computing, IEEE*, vol. 14, no. 1, pp. 44-51, 2010.

[2] S. Hong, D. Kim, M. Ha, S. Bae, S. J. Park, W. Jung, and J.-E. Kim, "Snail: an ip-based wireless sensor network approach to the internet of things," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 34-42, 2010.

[3] O. F. Xue-ning Jiang and X. min Xu, "Beatmaster : Software defined clock frequency for system on chip designs," *International Journal of Novel Materials*, vol. 3, pp. 27-32, 2012.

[4] T. S. Lo´ pez, D. C. Ranasinghe, M. Harrison, and D. McFarlane, "Adding sense to the internet of things," *Personal and Ubiquitous Computing*, vol. 16, no. 3, pp. 291-308, 2012.

[5] Z. Ji, J. Ma, and O. Faust, "Formal and model driven design of a high speed data transmission channel," *Journal of Circuits, Systems, and Computers*, vol. 22, no. 10, p. 1340038, 2013.

[6] F. Alidoust Aghdam and S. Saeidi Haghi, "Implementation of high performance microstepping driver using fpga with the aim of realiz- ing accurate control on a linear motion system," *Chinese Journal of Engineering*, vol. 2013, pp. 1-8, 2013.

[7] "Bluetooth module: HC-05," http://wiki.iteadstudio.com/Serial_Port_Bluetooth_Module (Master/Slave)_:_HC-05, accessed: 2015-10-30

[8] X. Spartan, "Fpga lx9 microboard user's manual," *Avnet Incorporated*, vol. 2211, 2006.

[9] "Avnet microboard," http://www.em.avnet.com/en-us/design/drc/Pages/Xilinx-Spartan-6-FPGA-LX9-MicroBoard.aspx, accessed: 2015-10-25.

[10] M. P. By Gang Qu, *Intellectual Property Protection in VLSI Designs: Theory and Practice*. Kluwer Academic, 2003.

[11] S. Aurell, "Remote controlling devices using instant messaging: building an intelligent gateway in erlang/otp," in *Proceedings of the 2005 ACM SIGPLAN workshop on Erlang*. ACM, 2005, pp. 46-51.

[12] AXI, Xilinx, "Reference guide," *Xilinx Inc*, 2011.

[13] V. V. Athani, *Stepper motors: fundamentals, applications and design*. New Age International, 1997.

[14] "Software manual," http://www.xilinx.com/support/documentation/sw manuals/xilinx13 2/edk ctt.pdf, accessed: 2015-10-28.

[15] B. H. C. Sputh, O. Faust, and A. R. Allen, "A versatile hardware-software platform for in-situ monitoring systems." in *CPA*, 2007, pp. 299-311.

[16] B. H. C. Sputh, O. Faust, and A. R. Allen, "Portable csp based design for embedded multi-core systems." In *CPA*, 2006, pp. 123-134.

[17] D. Nathan, B. Sputh, O. Faust, and C. B. Koon, "Real-time decoding and streaming of dab audio frames by a user-space program running on a non-real-time os," *Consumer Electronics, IEEE Transactions on*, vol. 48, no. 2, pp. 313-321, 2002.

[18] O. Faust, W. Yu, and U. R. Acharya, "The role of real-time in biomedical science: A meta-analysis on computational complexity, delay and speedup," *Computers in biology and medicine*, vol. 58, pp. 73-84, 2015.

[19] "Importance of embedded systems," https://en.wikibooks.org/wiki/ Embedded Systems/Embedded Systems Introduction, accessed: 2015-11-03.

[20] Z. Song, Z. Ji, J.-G. Ma, B. H. C. Sputh, U. R. Acharya, and O. Faust, "A systematic approach to embedded biomedical decision making," *Computer methods and programs in biomedicine*, vol. 108, no. 2, pp. 656-664, 2012.

[21] M. Barr and A. Massa, *Programming embedded systems: with C and GNU development tools*. " O'Reilly Media, Inc.", 2006.

[22] D. Jen and A. Lotan, "Processor interrupt system," Jan. 29 1974, uS Patent 3,789,365.

[23] B. H. C. Sputh, O. Faust, E. Verhulst, and V. Mezhuyev, "Opencomrtos: A runtime environment for interacting entities." in *CPA*, 2009, pp. 173-184.

[24] G. Uğurel and C. F. Bazlamac¸ci, "Context switching time and memory footprint comparison of xilkernel and µc/os-ii on microblaze," in *Elec- trical and Electronics Engineering (ELECO), 2011 7th International Conference on*. IEEE, 2011, pp. II–62.

[25] O. Faust, B. Sputh, D. Endler, and A. R. Allen, "Chaining communications algorithms with process networks," *Communicating Process Architectures 2004*, vol. 62, p. 325, 2004.

Appendix

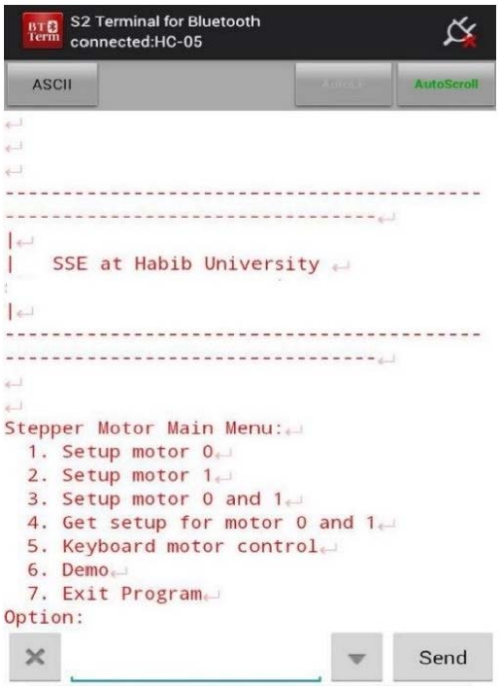

Figure 7. Main Menu Information

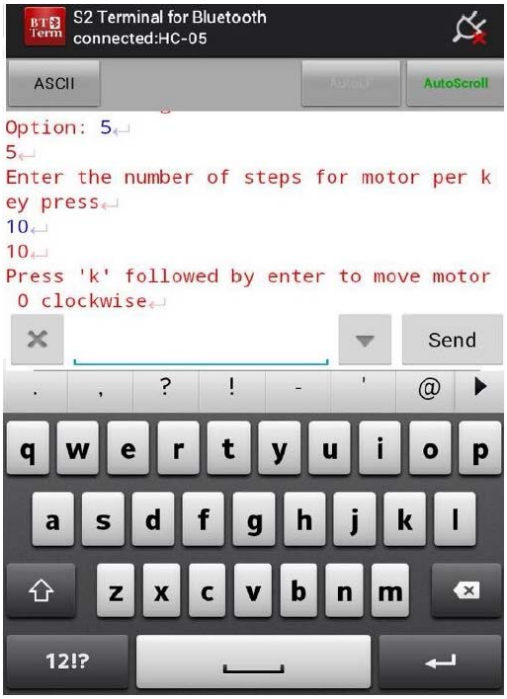

Figure 8. Sub Menu 1 Information

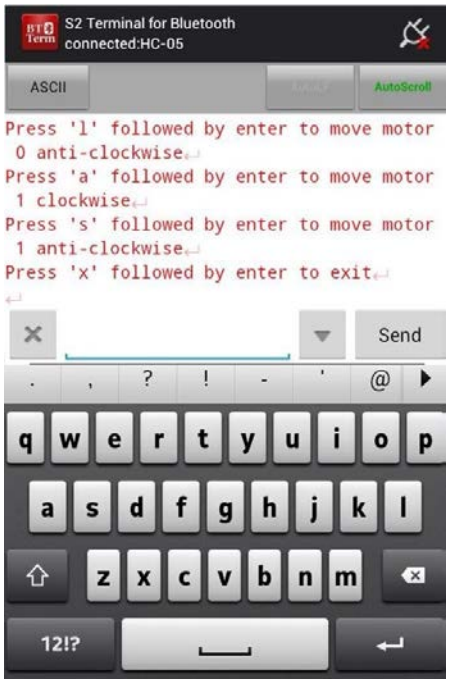

Figure 9. Sub Menu 2 Information