Bluetooth Configuration of an FPGA: An Application to Modular Robotics

Semester Project

Summer 2004-2005

Jérôme Maye

Supervisors: Andres Upegui
Prof. Auke Jan Ijspeert

Abstract

Self-reconfigurable modular robotics represents a new approach to robotic hardware. Instead of being designed as a huge centralized system, the “robot” is composed of many simple, identical, interacting modules. With the help of some computation, sensing and communication capabilities, self-reconfiguring robots should be able to adapt their morphology to the environment in order to fulfill a desired task.

The Biologically Inspired Group (BIRG) of the Swiss Federal Institute of Technology in Lausanne (EPFL) has developed a brand new Modular Robot called *YaMoR*. Each module contains an FPGA, a Bluetooth board, a power board, two batteries and a servo-motor. Assembling the modules together, a centralized Java application can supervise the overall movement via Bluetooth.

The aim of the current semester project is to study the unexploited programming potential of the Bluetooth board, containing an ARM core, some Flash and some SRAM. In particular, we show how we can configure the FPGA board wirelessly.

Acknowledgements

First and foremost, I am deeply indebted to my supervisor, Andres Upegui. His encouragement, support, and advice have been immensely valuable. Moreover, he was always available when I needed help or suggestions.

Special thanks go to Alessandro Crespi for his great help in electronics and in practical computing.

I also owe a great debt to Andre Badertscher for his electronic work on the YaMoR modules and for the maintenance of the LSL lounge. Without him, I would have taken much more time on the solvering of the PCBs.

I am also grateful to all the LSL team for their friendship and availability. It is really a pleasure to come every day at the laboratory when the atmosphere is so warm.

A great thanks goes to Professor Auke Jan Ijspeert, who is always motivating people to work hard and consciously, by showing his interest.

Finally, I have to thank all the people who have initiated this project, especially Rico Moeckel, Cyril Jaquier and Kevin Drapel.

Contents

List of Figures

Chapter 1

Introduction

Few things are impossible to diligence and skill. Great works are performed not by strength, but perseverance.
Samuel Johnson (1709 - 1784)

Self-reconfigurable modular robotics represents a new approach to robotic hardware. Instead of designing a new and different mechanical robot for each task, a generic "robot" is composed of many simple, identical, interacting and possibly low-priced modules. The versatility of the robot emerges from the connection of all its components.

Furthermore, given some mechanical abilities, a modular robot can even modify the position of its components to meet the demands of different tasks or different working environments. Recently, some researchers achieved the construction of a modular robot able to replicate itself in a three-dimensional environment, opening a whole world of potential applications.

Modular robots are particularly well-suited to outdoor tasks, away from civilisation, where both mission and geography may be unpredictable. In the future, systems of this kind will be involved in various operations ranging from space exploration (see [YRD+03]) to urban search and rescue (USAR) (see [YDR00a]) in buildings badly damaged by an earthquake or a bomb.

Designing a modular robot involves holding the promises of versatility, adaptability, robustness and low cost. Although the related programming and technical challenges have yet not been fully and satisfactorily overcome, several research projects have reached key milestones, encouraging the scientific community to persevere in that direction.

1.1 Overview of the Projects of Modular Robotics

1.1.1 PolyBot

The Palo Alto Research Center (PARC) has been pursuing research on modular robotics for about ten years. The third generation of a robot called PolyBot (see [YDR00b]) (Figure 1.1) has been released three years ago. The system is composed of several modules containing a microcontroller, a motor, sensors and the ability to attach to other modules. The robot has demonstrated a variety of locomotion, climbing, manipulation as well as manual and self-reconfiguration.

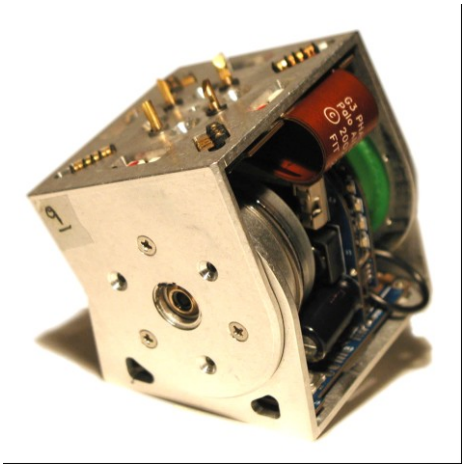


Figure 1.1: A PolyBot module

1.1.2 Molecule

The Dartmouth College Robot Lab has released a modular robot called *Molecule* (Figure 1.2), capable of self-reconfiguration in three-dimensional space. The robot is controlled by low-level assembly code in the onboard processor and high-level code on workstations. There is no clear documentation about this project, but it seems that the communication between the modules and the workstations is performed with cables.

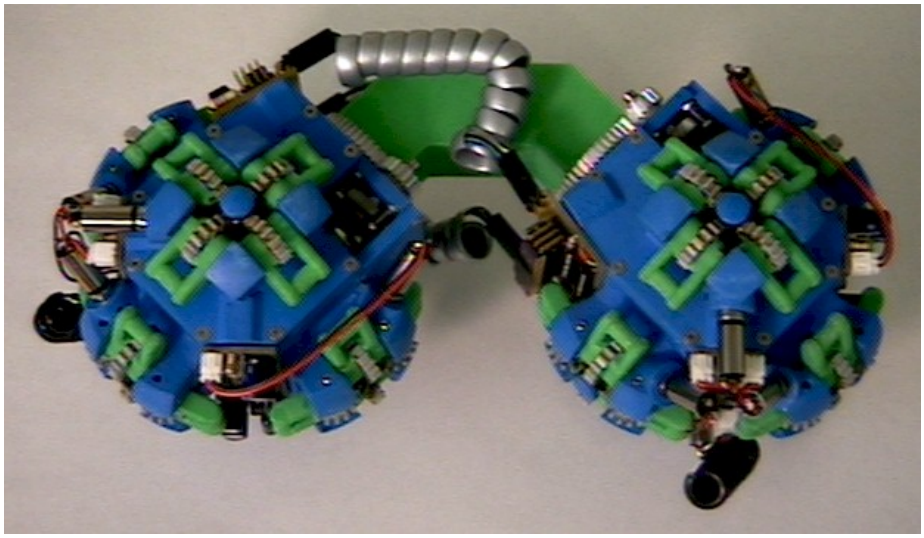


Figure 1.2: The Molecule robot

1.1.3 M-TRAN

The Distributed System Design Research Group of Intelligent Systems Institute, AIST has developed a novel self-reconfigurable modular robot called Modular Transformer (M-TRAN) (Figure 1.3). It has successfully realized multi-mode robotic motion by changing its shape smoothly from a crawler to a four-legged walking robot, which is a world-leading research result.

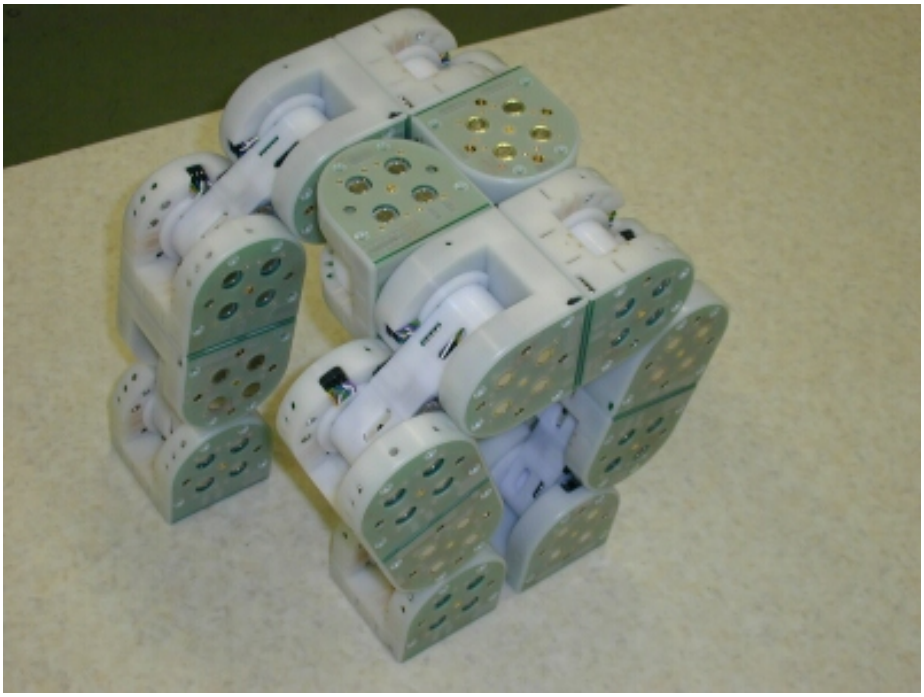


Figure 1.3: The M-TRAN robot

1.2 The YaMoR Project

The Biologically Inspired Group (BIRG) of the Swiss Federal Institute of Technology in Lausanne (EPFL) has developed a brand new Modular robot called *YaMoR* (Figure 1.4), standing for *Yet another Modular Robot*. Each module contains a Field Programmable Gate Array (FPGA), a Bluetooth board, a power board, two batteries and a servo-motor. Assembling the modules together, a centralised Java application can supervise the overall movement via Bluetooth (see [JD05b]).

The behaviour of the module is controlled by the FPGA, into which a Pulse Width Modulation(PWM)[1] generator has been implemented, outputting a signal to the servo-motor. At each time step, the Java application determines the angular position of the mechanical arm attached to the motor and sends it on air to the embedded Blue-

[1] Pulse width modulation (PWM) is an efficient electronic control technique used in stepper motor drivers to set average winding current. PWM is commonly used in high power amplifiers and power supplies.

Figure 1.4: A YaMoR module

tooth board, which in turn simply forwards it through an Universal Asynchronous Receiver/Transmitter (UART)[2] line feeding the PWM module.

The originality of the project arises from the combination of Bluetooth communication and the use of FPGA, two features that will be largely detailed in this report. The FPGA technology fits perfectly into the framework of modular robotics. A single module really becomes generic and can be either totally or partially reconfigured when the circumstances require it. Thanks to the integrated Bluetooth board, the interconnecting wires between the different modules may be totally removed, allowing more freedom in the way the robot organises its physical structure.

At the current stage, seven modules with static interconnections have been successfully built. The angular positions are tuned manually to obtain a desired behaviour. Some students are working on an extension of the existing Java application, a simulator which will receive exactly the same commands as the robot. The comparison between the model and the reality will certainly brings surprising results.

Although a complete, distributed and autonomous solution remains the final aim of the project, motivating experiments have exhibited interesting emerging movements and showed the difficulty to cope with the coordination of the modules.

1.3 Our Contribution to the YaMoR Project

In the first phase of the project, the power of the different boards has been largely underexploited. The FPGA contains actually a simple PWM, although it supports a complex design including a softcore Microblaze processor and partial dynamic reconfiguration. The Bluetooth board is currently driven by the original firmware provided by the man-

[2] A device, usually an integrated circuit chip, that performs the parallel-to-serial conversion of digital data to be transmitted and the serial-to-parallel conversion of digital data that has been transmitted.

ufacturer of the chip. It only offers the possibility to establish a Bluetooth connection with a remote device and then act as a virtual serial port. All the data received via Bluetooth are simply forwarded on the UART line and conversely.

The aim of the current project is to study the programming potential of the Bluetooth board. In particular, we are interested in the development of a module capable of realizing the configuration of the FPGA, using a bitstream received via Bluetooth and stored either in Flash or external Random Access Memory (RAM).

Although the main advantage of this project is to permit the complete removal of all the wires between the computer and the robot, we also have the strong ambition to give an accurate view of the different boards and technology involved, which could serve as a basis for future projects. Likewise, we think that our work could equally be suitable for other applications than modular robotics.

The rest of the report is organised as follows. Chapter 2 contains an overview of the Bluetooth protocol. Chapter 3 describes the Bluetooth board in details. Chapter 4 gives an insight into the FPGA board. Chapter 5 shows our implementation of the FPGA wireless configuration. Chapter 6 presents some experimental results and discusses future directions for the work on the YaMoR project. Chapter 7 outlines our conclusions on this semester project.

Chapter 2

Overview of the Bluetooth Protocol

You see, wire telegraph is a kind of a very, very long cat. You pull his tail in New York and his head is meowing in Los Angeles. Do you understand this? And radio operates exactly the same way: you send signals here, they receive them there. The only difference is that there is no cat.
Albert Einstein (1879 - 1955)

Since our work partially relies on Bluetooth, this chapter offers a brief overview of the underlying protocol and of its key figures. The experienced reader may skip this part and directly jump to Chapter 3, while the novice will find all the necessary elements to capture the essence of the rest of the report.

2.1 Origin

Bluetooth is the name of a short range radio technology, allowing the creation of a wireless link between any kind of devices. It was originally developed by Ericsson Mobile Communication in 1994, as an alternative to the cables that linked its mobile phones with accessories such as headsets or Personal Digital Assistant (PDA). Although the initial motivation was to unify the telecommunications and computing industries, many other applications have been discovered within the last ten years, ranging from wireless mouses to virtual serial ports.

In order to widespread this revolutionary concept, a group of companies, called the Bluetooth Special Interest Group (SIG), joined their efforts for promoting and defining the Bluetooth specification. This latter specifies precisely the complete system, from the radio right up to the application level. For the sake of compatibility, the use of Bluetooth is limited to the companies that have joined the SIG and therefore strictly respect the specification.

2.2 Protocol

A key feature of the Bluetooth specification is the possibility to connect devices from lots of different manufacturers. To achieve this goal, besides the radio system, a complete software stack (Figure 2.1 is also defined, enabling applications to find other

Bluetooth devices in the area, discover what services they can offer, and use those services.

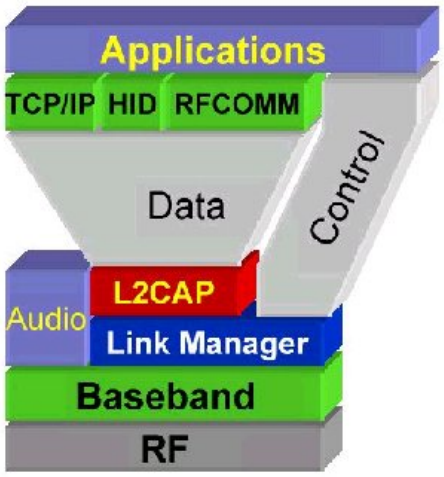


Figure 2.1: Bluetooth stack

Following roughly the classical Open Systems Interconnect (OSI) standard reference model, the Bluetooth protocol stack is separated into a serie of layers, responsible at different levels for the communication.

2.2.1 Radio System

The radio layer modulates and demodulates data for transmission and reception on air. Bluetooth devices operates at 2.4 GHz, in the Industrial, Scientific, and Medical (ISM) band, which is globally available for the applications respecting a basic set of power, spectral emission, and interference specifications. Three different power classes are defined to allow operation ranges of 10 m, 20 m, and 100 m.

As the ISM band is extremely busy, it is not a terribly stable nor reliable medium of communication, due to the high frequency of collisions. To cope with this limitation, various techniques, such as frequency hopping and short data packets, have been deployed.

2.2.2 Frequency Hopping

The operating band is divided into 79 one MHz-spaced channels (23 in some countries). After each packet, both devices retune their radio to a different frequency. This operation is known as *frequency hopping*, and aims to ensure that whenever a packet transmission is compromised on one channel, the retransmission will occur on another channel (potentially clear).

2.2.3 Master/Slave

Frequency hopping relies on the synchronisation of the devices that communicate. Each device can operate in two modes: as a Master, or as a Slave. A Master initiates

the connection and sets the frequency hopping sequence, calculated from its Bluetooth address and Bluetooth clock. The Bluetooth address is composed of 48 bits, identifying each device uniquely. The Bluetooth clock is a 28 bits, free running counter, resetted to zero at power up. When a Slave connects to a Master, this latter transmits these two parameters for the synchronisation of the hopping sequence.

2.2.4 Piconet and Scatternet

The specification allows a Master to be connected simultaneously to a maximum of seven Slaves. This kind of configuration is referred to as a *Piconet*. Some devices may be members of more than one Piconet, forming a *Scatternet*. A device may even act as a Master in a Piconet and as as a Slave in another, as illustrated on Figure 2.2. Increasing the number of independent Piconets or Scatternets in one area is a major source of interference. The retransmissions will be more frequent and hence data rates will dramatically fall.

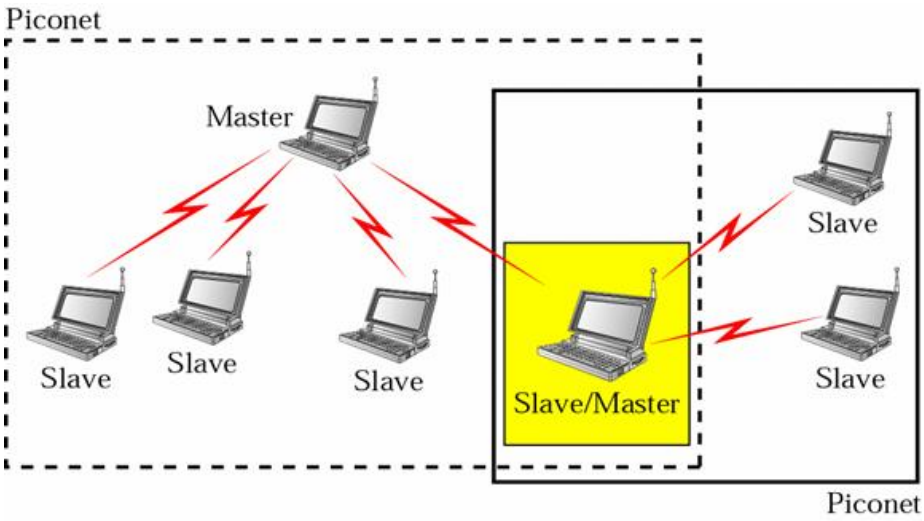


Figure 2.2: Piconet and Scatternet

2.2.5 Data Packets

Different amounts of useful data can be sent in Bluetooth packets, with a maximum of 339 bytes. Packets are transmitted with respect to time slots lasting 625 microseconds. According to the type of packet, the conveyance is distributed into 1, 3, or 5 slots. For instance, a packet containing 339 bytes of payload data consumes 5 slots for the transmission and 1 slot for the acknowledgement, leading to a maximum data rate of 723.2 Kb/s. As the higher layers of the protocol stack use up some of the bandwidth, the maximum data rate falls to 650 Kb/s at the application level.

2.2.6 RFCOMM and SDP

On the Bluetooth protocol stack, two other features are worth mentioning. RFCOMM provides an RS-232[1] like serial interface, which can be used for data transfer in several of the Bluetooth profiles and also as a virtual serial port. Service Discovery Protocol (SDP) lets Bluetooth devices discover what services other Bluetooth devices support and is hence a capital actor in a Bluetooth connection.

2.2.7 Profiles

For the application level, the specification includes a profiles document, describing how a particular end-user function can be implemented with respect to the protocol stack. Profiles ensure interoperability between devices from different manufacturers.

2.2.8 Security

The security of a Bluetooth link is normally guaranteed by the pseudo-random frequency hopping algorithm. In addition, for link encryption and authentification, a strong contemporary cypher algorithm called SAFER+[2] may be employed.

2.3 Use of Bluetooth

Bluetooth is forsooth a good choice for wireless applications. Its low cost, low power consumption, ease of use, and compatibility have turned a novel concept into an ubiquitous technology. For more information about Bluetooth, please refer to [BS00].

[1] RS-232 is a serial communications standard that provides asynchronous communication capabilities, such as hardware flow control, software flow control, and parity check. It has been widely used for decades. Almost all gears, instruments with digital control interface, and communications devices are equipped with the RS-232 interface. The typical transmission speed of an RS-232 connection is 9600 bps over a maximum distance of 15 meters.

[2] In cryptography, SAFER (Secure And Fast Encryption Routine) is the name of a family of block ciphers designed primarily by James Massey (from the ETHZ) on behalf of Cylink Corporation. The early SAFER K and SAFER SK designs share the same encryption function, but differ in the number of rounds and the key schedule. More recent versions -SAFER+ and SAFER++- were submitted as candidates to the AES process and the NESSIE project respectively. All of the algorithms in the SAFER family are unpatented and available for unrestricted use.

Chapter 3

The Bluetooth Board

A complex system that works is invariably found to have evolved from a simple system that works.
John Gaule (Middle-Age)

The Bluetooth board was designed by Rico Moeckel, a student from the Eidgenoessische Technische Hochschule Zuerich (ETHZ), during a summer internship in 2004. Unfortunately, due to time constraints, no clear documentation about his admirable work has been compiled yet.

Thanks to his electronic schemas (see Appendix A), to abundant user/reference guides and application notes from the creators of the main chip of the board, we aim at providing a deep understanding of the hardware, as well as the software used. The intent of this chapter is thus to serve as a reference guide for the development of new material for the Bluetooth board.

3.1 Components

Figure 3.1 shows a picture of the Bluetooth board. It contains a Bluetooth Microcontroller, an antenna, some external memories and some input/output pins. In the rest of this section, we describe the relevant elements integrated in the board.

3.1.1 Bluetooth Microcontroller

At the heart of the Bluetooth board lies a ZV4002 (Figure 3.2) chip, designed by Zeevo, a german manufacturer. The ZV4002 allows designers of embedded systems to add wireless connectivity to their products. The chip incorporates the industry standard 32-bit ARM7TDMI Central Processing Unit (CPU) core, with excess processing bandwidth sufficient to support a wide range of embedded applications.

All Radio Frequency (RF) components and digital circuitry are integrated into one solution. The only external components needed are an antenna, crystal, reference resistor, decoupling capacitors, flash memory, and optionally external RAM.

ARM Core

The ZV4002 includes an ARM7TDMI (Figure 3.3), a widespread 32-bit Reduced Instruction Set Computer (RISC) microprocessor core. A combination of high code den-

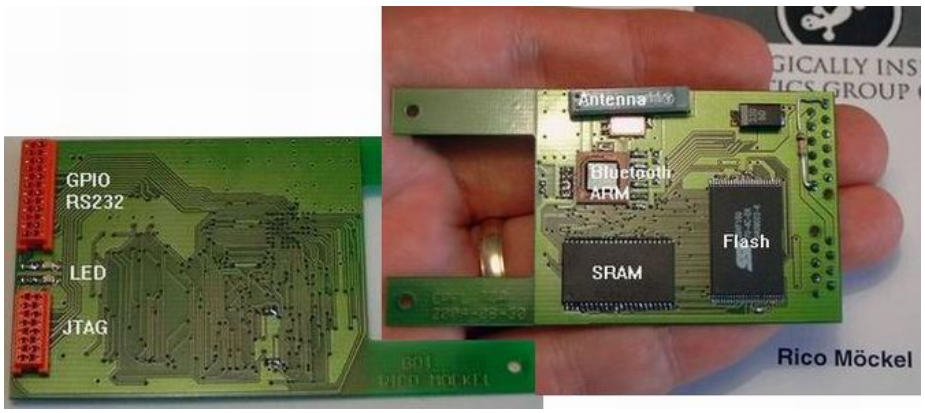


Figure 3.1: The Bluetooth board

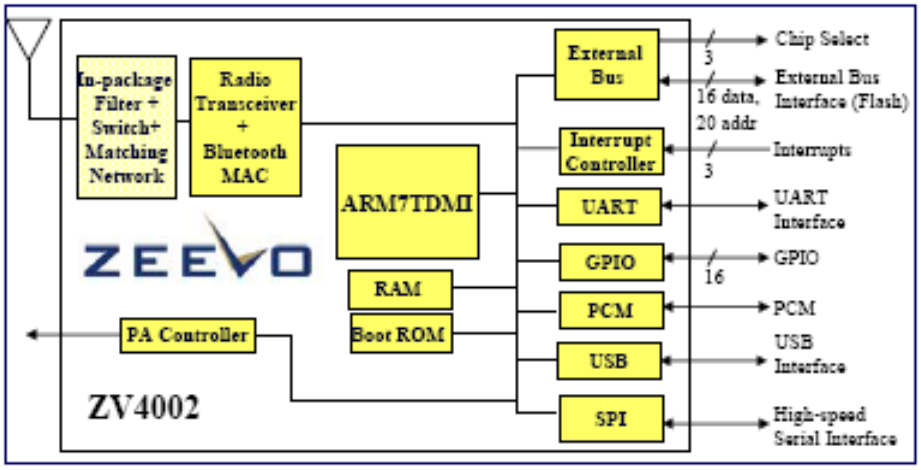


Figure 3.2: ZV4002 schema

sity, fast execution time, minimal system memory size, and low power consumption, are the main ingredients of an uncontested commercial success.

The key features of a RISC architecture is a set of simple but powerful instructions, that execute within a single clock cycle at a high clock speed. The fixed length of the instructions allows the processor to fetch the next one before the end of the decode phase of the current one. The processing may thus be broken into smaller units, forming a pipeline. The memory can only be accessed through load/store instructions and a large general-purpose register file.

The ARM7TDMI uses sixteen 32-bit user registers, as well as a 32-bit unified interface bus, carrying both instructions and data. It is built on a three-stage pipeline architecture and contains a 32-bit Arithmetic Logic Unit (ALU) and a high performance multiplier. Using 16-bit Thumb instruction set may increase code density. In our case, the processor is driven by a 12 MHz external crystal.

The processor communicates with memory and external peripherals through a highly

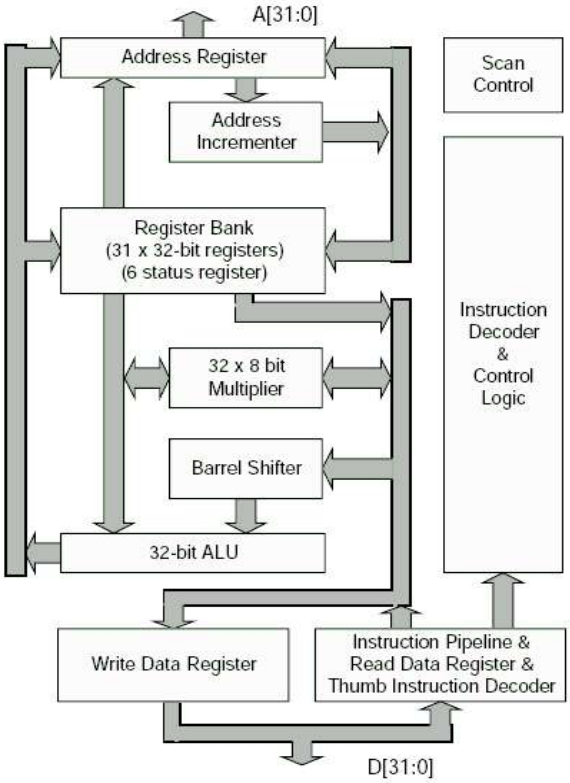


Figure 3.3: ARM7TDMI schema

compatible AMBA bus, supported by a lot of devices. For more information about the ARM core, please refer to [SSW04].

Memory

The ZV4002 also incorporates some on-chip memory. At startup, the processor starts executing instructions at address 0, which is generally mapped to an internal 8 KBytes boot Read Only Memory (ROM) (for debugging purpose, this address can be linked to the external flash). The role of the ROM code is to configure the main UART, to send out a probe character potentially caught by a flash manager, and to jump to the Flash code entry point.

Besides the ROM, two types of RAM are included: 32 KBytes of Internal RAM (IRAM) and 32 KBytes Buffer RAM (BRAM). IRAM is a general-purpose high speed RAM, used for both data and code storage. BRAM is similar to IRAM, except that the on-chip Direct Memory Access (DMA)[1] controllers can have access to it. To avoid conflicts with the DMA, code will generally not run on BRAM.

[1] A method of transferring data from one memory area to another without having to go through the central processing unit.

Peripherals

All ARM peripherals are memory mapped, the programming interface is a set of memory-addressed registers. The address of these registers is an offset from a specific base address. Among the different peripherals visible on Figure 3.2, some deserve further explanation.

In the Bluetooth board, only the TX and RX lines of the UART are necessary to communicate serially with a remote device. The UART is 3.3 V tolerant and thus an RS-232 adapter (Figure 3.4) is needed to set up a connection with an external Personal Computer (PC). The UART is configurable via software with the standard options of the norm (speed, parity, data bits, stop bits). In the ZV4002, there is a 32-byte queue between the UART and the DMA. Depending on the load, this queue might become full and cause a hardware reset.

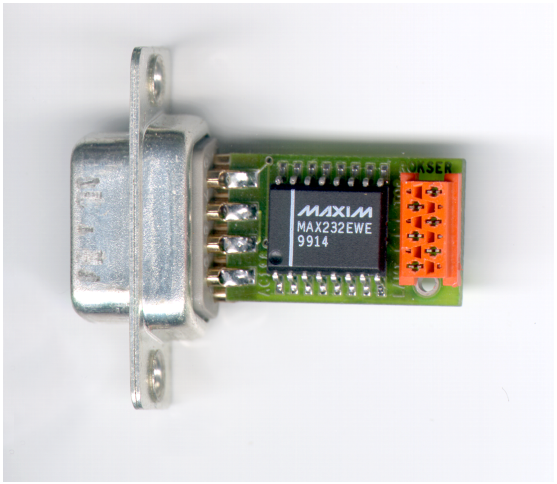


Figure 3.4: RS-232 adapter

The ZV4002 offers sixteen individually programmable General Purpose Input/Output (GPIO), capable of sinking and sourcing 2 mA of I/O current. These terminals are 5 Volts tolerant. GPIO0 to GPIO7 are internally pulled down with 50 KΩ (nominal) resistors, while GPIO8 to GPIO15 are internally pulled up with 50 KΩ (nominal) resistors. In the Bluetooth board, the GPIOs are accessible through Micromatch connectors (or external pins) and output a maximum voltage of 3.3 V for a logical '1'. Only GPIO4, GPIO7, GPIO8, GPIO10, GPIO11 and GPIO12 are reachable. Moreover, GPIO0 is connected on a Pad, outputting the 12 MHz reference oscillator in crystal test mode, for debugging purpose. GPIO4 normally indicates Bluetooth Baseband activity on a green Light Emitting Diode (LED)[2]. GPIO12 may also be used for debugging purpose, since it is linked to a red LED (the output current might thus not be sufficient for other applications). In short, only GPIO7, GPIO8, GPIO10 and GPIO11 might be employed as "real" GPIOs in embedded applications that work in a 3.3 V mode and need a minimal current.

[2] A semiconductor device that emits light when an electric current passes through it. Used for status lights or bar graph meters in many audio devices, LED's are also found everywhere from watches to laser disc players. Some LED's emit infra red energy and are found in remote control devices and wireless headphone systems.

The External Bus provides access control to external devices such as Flash, static RAM (SRAM)[3], Digital Signal Processor (DSP)[4], ROM/PROM and Universal Serial Bus (USB)[5] host controller. Up to 5 different banks of control with separate chip selects and interrupts for each may be attached. Two banks are dedicated for Flash and SRAM, with interfaces configurable for 8 or 16 bit accesses. Before using the banks, some parameters have to be finely configured via software (chip select and address setup time, chip select and address hold time, write enable width, output enable width, width of bank).

3.1.2 External RAM

For temporary data storage, a K6X8016T3B low power SRAM is included in the board. It is organized as 512K words of 16 bits, addressed by a bus of 19 bits and operates as a standard RAM.

3.1.3 Flash Memory

Embedded devices also need nonvolatile storage for data that have to be preserved when the power is off. Power and space concerns normally lead to Flash technology. This kind of memory reduces the area dedicated to control by restricting writes to fully erased blocks of bytes.

The Bluetooth board contains a SST39VF160 multi-purpose Flash memory, designed by Silicon Storage Technology (SST), with their high performance Complementary Metal Oxide Semi-conductor (CMOS)[6] SuperFlash technology.

The memory is organized as one million of 16 bits words, addressed by a bus of 20 bits. Low-power consumption, hardware and software data protection schemes, short program and erase times, greater than 100 years data retention, and 100000 cycles of endurance are the main features that make this chip really suitable for a modular robot. The memory operation functions of the device are initiated by commands, written using standard microprocessor write sequences.

Read Operation

A Read operation corresponds closely to a standard memory read access and thus it is not worth complementing this part with additional details, except that the operation completes in about 80 nanoseconds.

[3] A kind of random access memory that requires a constant supply of power in order to hold its content, but does not require refresh circuitry as dynamic random access memory (DRAM) does. Each static RAM bit is a flip-flop circuit made of cross-coupled inverters; the activation of transistors controls the flow of current from one side to the other. Unlike read-only memory (ROM), SRAM will lose its content when the power is switched off. Static RAM is usually faster than dynamic RAM, but takes up more space and uses more power.

[4] A specialized digital microprocessor used to efficiently and rapidly perform calculations on digitized signals that were originally analog in form (eg voice). The big advantage of DSP lies in the programmability of the processor, allowing parameters to be easily changed.

[5] A bidirectional, isochronous, dynamically attachable serial interface for adding peripheral devices such as game controllers, serial and parallel ports, and input devices on a single bus.

[6] A widely used type of semiconductor technology that uses both NMOS (negative polarity) and PMOS (positive polarity) circuits. Since only one of the circuit types is on at any given time, CMOS chips require less power than chips using just one type of transistor.

Word-program Operation

The Flash is programmed on a word-by-word basis. The Sector, in which the word will reside, must be totally erased before any programming operation. The complete operation typically lasts about 16 microseconds.

Sector/Block-erase Operation

The device can be erased on a Sector-by-Sector (or Block-by-Block) basis. A set of 2 KWords form a Sector, while 32 KWords represent a Block. The overall operation will end up in about 16 milliseconds.

Chip-erase Operation

In addition, a valuable command can also be issued to expeditiously erase the entire Flash to the "1" state. This operation will go on during about 64 milliseconds.

3.1.4 Memory Map

After a reset, the memory layout is as in Figure 3.5 (sizes are in bytes and the ARM is configured by default in Little Endian[7]). The actual size of the Flash and external RAM devices determines the range of useful addresses. The memory used for the exception vectors can be mapped to ROM, Flash or IRAM. The exception vectors table contains eight 32-bit instructions, directing the processor on what to do when an exception/interrupt occurs. The instruction at address 0 is executed immediately after a hardware reset.

3.2 Included Software

The ZV4002 chip was delivered with a comprehensive working firmware[8] (Zerial), for handling Bluetooth Serial Port Profile (SPP) and some other features. Figure 3.6 shows an overview of the complete software stack, which can be flashed as an object file via the UART port of the ZV4002. For application development, the source code is also provided and may be freely edited.

In addition to the Zerial firmware, another one called Host Controller Interface (HCI) is as well supplied. HCI is a Bluetooth standard interface. A software running on a host CPU issues commands (via USB or UART) to control the ZV4002. As can be observed on Figure 3.7, only the lower layers of the Bluetooth protocol stack run on the ZV4002. Whereas this firmware was not used in the YaMoR project, we rather focus on the Zerial one in the continuation of this chapter.

3.2.1 Zerial Application

Zerial provides a cable replacement type of application for RS-232 communications. The software behaviour of the Zerial interface is similar to a Hayes[9]-compatible mo-

[7] Data storage format in which a multi-byte data value is stored with the least-significant data byte through most-significant data byte in the lowest through highest byte addresses, respectively. This is the storage format used by Intel and Digital processors.

[8] Program kept in semi-permanent storage, such as various types of read only memory or flash. Software is "burned in" on the memory device so that it is non-volatile (will not be lost when power is shut off)

[9] The manufacturer that first defined the Standard AT Command Set for modems

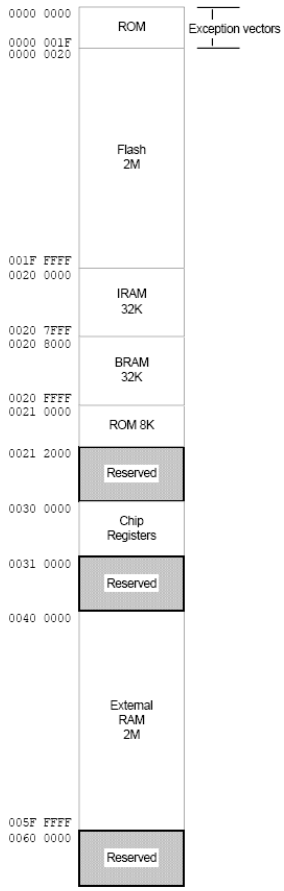


Figure 3.5: Memory map upon reset

dem using an AT[10] command set. A small amount of configuration is performed in command mode, and then the device is placed in bypass mode to function as an RS-232 link. The advantage of Zerial is that neither the Host CPU nor the Application needs to know much about Bluetooth to communicate over the link.

The Zerial interface works as illustrated in Figure 3.8. At startup, the system is in *command mode* and either awaits for commands issued from a host computer through the UART, or listens for a Bluetooth SPP connection from a remote device. Upon connection, the Zerial interface goes into *bypass mode* and the data stream coming from the UART is sent on the air. For more information about the Zerial interface, please refer to [Zeef] and [Zeee].

In a YaMoR module, this application is currently used to control the movement of the servo motor. A Java application, running on a remote computer, establishes a SPP connection with the module. Positions for the motor are then sent on the virtual link and simply forwarded to the FPGA board, which is connected to the UART of the Bluetooth board. Inside the FPGA, a controller interprets the commands and drives a PWM attached to the servo motor.

[10]Modem control language for asynchronous dial-up modems

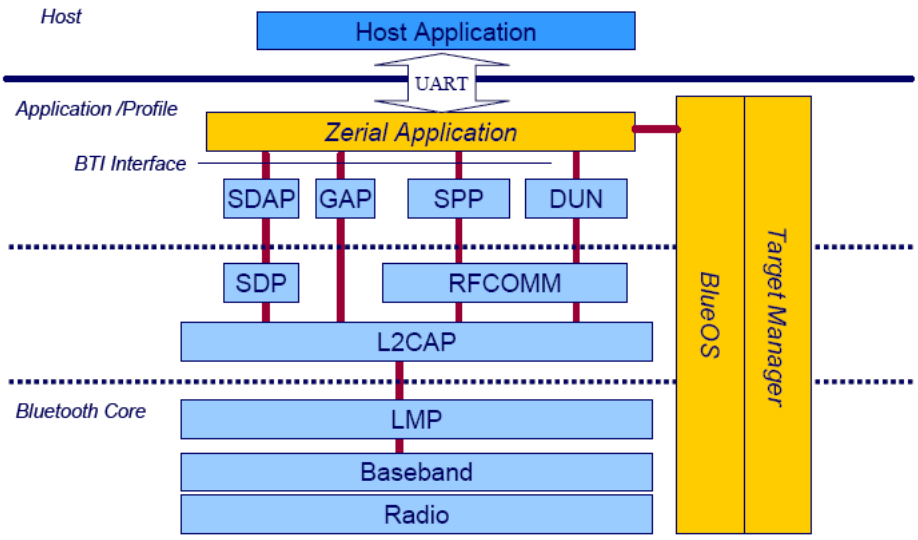


Figure 3.6: Zerial firmware

3.2.2 BlueOS

The BlueOS (see also [Zeea] and [Zeeb]) is a non-preemptive kernel supporting multiple tasks. It is built upon the concept of encapsulated entities called modules. These latters are attached to a backplane, through which they can communicate. Figure 3.9 illustrates this principle.

A module is generally composed of multiple tasks, working together to provide services to other modules. The functionalities of the component are accessible through a function-based interface or a signal-based interface. The BlueOS has a queue of signals, which are sent iteratively to tasks. Figure 3.10 shows how a signal enters a module and dispatches to the tasks.

In BlueOS, tasks may normally only be preempted by interrupts. Therefore, since only one task executes at the time, the kernel just needs a single stack and thus saves RAM space. Moreover, unlike other realtime OS, there is no necessity to use complex primitives like semaphores, mutexes and events for the synchronisation.

The memory is managed either by heaps or by pools. Pools are faster and more efficient since they only handle memory blocks of the same size. Heaps on the other hand are more flexible and can handle memory blocks of different sizes on one heap.

The BlueOS, optimized for the ZV4002, is really an excellent choice for embedded applications. It is fast, efficient, object oriented, easily extensible and lowers power consumption.

3.2.3 Target Manager

The Target Manager (see also [Zeec] and [Zeed]) is a BlueOS module that provides the interface between the hardware and the software utilities. In the ZV4002, it handles the timers, the UART, the USB, the Flash, the interrupts, the GPIOs, the Deep sleep, and the Fatal control. Some of the Target Manager tasks may receive signals directly from hardware, and others from software.

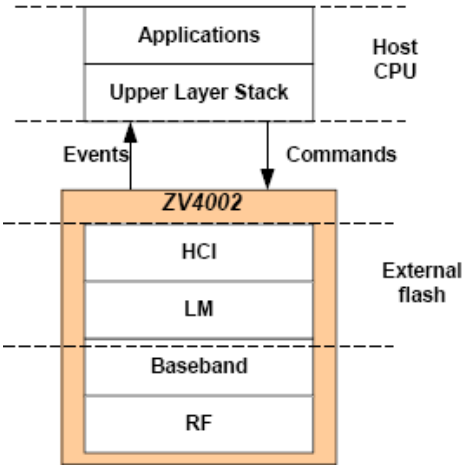


Figure 3.7: HCI firmware

The GPIOs are controlled via three registers, namely *PioDirection*, *PioOutData*, and *PioInData*. *PioDirection* specifies whether the GPIO is an Input or an Output. *PioOutData* contains the values to output on the ports, while *PioInData* stores the values read on the input ports. Although the Target Manager offers some Application Programming Interfaces (API) to facilitate the use of the GPIOs, the code will be more efficient if the registers are directly programmed.

3.3 Extending the Zerial Firmware

This section aims at providing a tutorial for the development of a new module into the BlueOS. It is intended for experienced software developers, who have correctly installed the ARM Developer Suite (including Codewarrior IDE), have a copy of the CD-ROM of the project and an RS-232/UART cable at hand.

3.3.1 Preamble

Some Perl support files need to be added in order to compile the Perl files in ARM's Metrowerks compiler. Please check the following, assuming ARM is installed under C: \ Program Files \ ARM:

- *MWPerl.dll* is in C: \ Program Files \ ARM \ ADSv1_2 \ Bin \ plugins \ compiler
- *MWPerlPanel.dll* is in C: \ Program Files \ ARM \ ADSv1_2 \ Bin \ plugins \ preference panel
- *PerlDLL.dll* is in C: \ Program Files \ ARM \ ADSv1_ 2 \ Bin \ plugins \ support

If these files are not at the right place, copy them from the ARM Developer Suite in the correct directory.

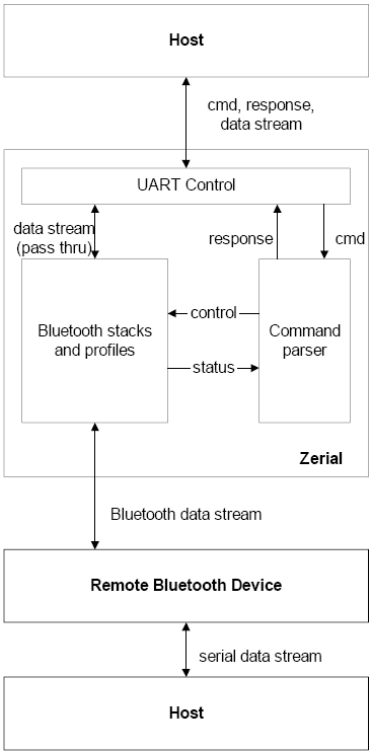


Figure 3.8: Zerial interface overview

3.3.2 First steps

1. Double-click on *Zerial.mcp*, under the directory path ZV4002 \ Zerial_v6_0_1_Development_Kit_Firmware \ BlueZone \ SourceBase \ Project
2. Remove the existing object code if needed by pressing *Ctrl*+-
3. Compile and link the whole project by pressing *F7*. The firmware is now concatenated in a file called *RawFlashCode.bin*
4. Double-click on *ZvDosFlashTool.exe* under the directory path ZV4002 \ Zerial_v6_0_1_Development_Kit_Firmware \ BlueZone \ SourceBase \ Project \ Zerial_Data \ ZerialProduct
5. Connect the RS-232/UART cable from the computer to the Bluetooth board
6. In the Flash tool, select the correct COM port and reset the board
7. Select *Update firmware*. Wait for the end of the process and press *F9*

3.3.3 Create a new module

1. In the Zerial project, right-click on the window and select *Create Group...*
2. In the dialog box, enter *TST* and press return

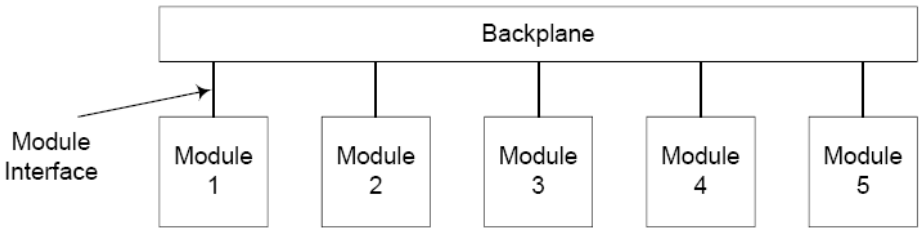


Figure 3.9: The BlueOS backplane

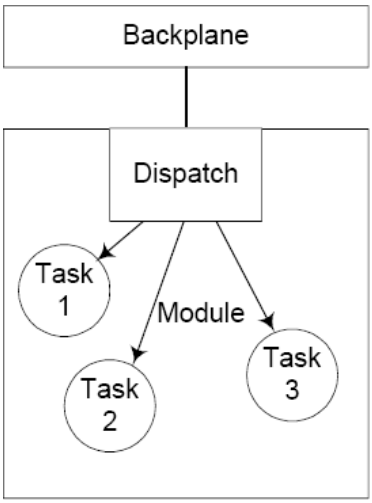


Figure 3.10: The dispatching of a signal in a module

3. Right-click on the TST folder and select *Add Files...*
4. Select the *TST_Module.c* and *TST_Module.h*, under the directory path ZV4002 \ TST
5. Open *Modules.h* in ZV4002 \ Zerial_v6_0_1_Development_Kit_Firmware \ BlueZone \ SourceBase \ ChipBase \ BlueOS
6. Add *BP_DECLARE_MODULE(TST)* in the file and recompile all the project
7. Your module is now working in the BlueOS and you can add it some functionalities

Chapter 4

The FPGA Board

The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man.
George Bernard Shaw (1856 - 1950)

An FPGA is a device mainly composed of an array of programmable logic cells. The particularity of an FPGA is that its functionality is not fixed in advance. It can be programmed hundred of times with a different design and thus offers an outstanding versatility at an acceptable cost. This technology is especially popular for prototyping integrated circuits designs.

In the YaMoR project, each module incorporates an FPGA board. This Printed Circuit Board (PCB) was also designed by Rico Moeckel during his summer internship in 2004.

Starting from his electronic schemas (see Appendix B) and application notes, the purpose of this chapter is to supply a clear documentation about the board and all of its components, in order to ease future development.

4.1 Components

As can be seen on Figure 4.1, the FPGA board contains an FPGA, an external SRAM, some configuration pads and pins, and some GPIOs. In the rest of this section, we deal with these different elements in more details.

4.1.1 FPGA

The core of the PCB is a XC3S400 Spartan-3 FPGA, designed by Xilinx, the world's largest developer and manufacturer of FPGAs. The Spartan-3 family presents an ideal balance between low cost (less than 40 $) and high performance, making it suitable for a wide range of consumer electronics applications. Globally, the architecture of a Spartan-3 FPGA derives from the well-established Virtex-II family.

The XC3S400 contains 8064 logic cells or 400000 system gates. The chip needs three different voltages: 1.2 V for the core, 3.3 V for the I/Os, and 2.5 V for auxiliary purposes. As the 2.5 V voltage is generated from a 3.3 V converter in the FPGA board, serious care must be taken that this component is not damaged. Some internal RAM and multipliers are also included. The system relies on Configurable Logic Blocks

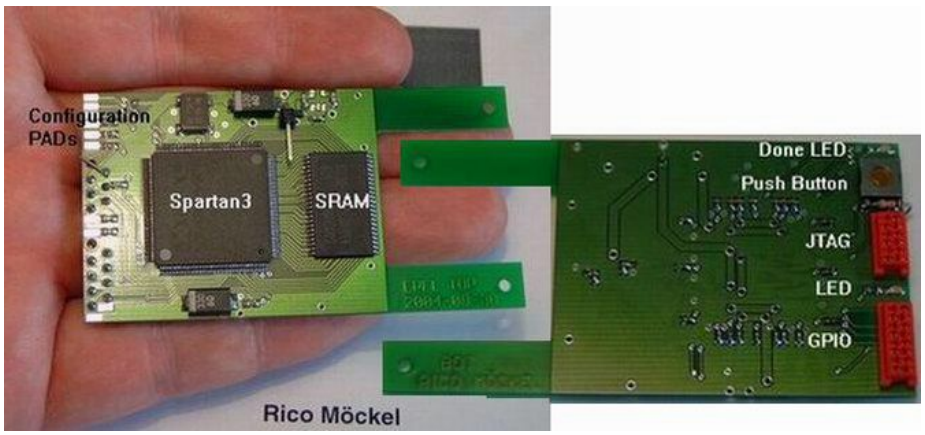


Figure 4.1: The FPGA board

(CLBs), containing RAM-based Look-Up Tables (LUTs). CLBs can be programmed to perform a wide variety of logical functions as well as to store data.

In the YaMoR project, the XC3S400 implements a minimal design. Actually, it only contains an UART and a PWM, which is sufficient for a basic robot controller. The positions of the servo motor are serially sent to the UART and then the PWM receives new duties. In this first phase, the power of the Spartan-3 FPGA is largely underexploited. In fact, it supports several complete 32-bit MicroBlaze softcore processors, functions used by DSP system designers, Dynamic Partial Reconfiguration (DPR), and so on. For more information about the Spartan-3 family, please refer to the complete data sheet [Xil05a].

4.1.2 Pins, Pads, and GPIOs

The FPGA board interacts with the outside world through several pins, also accessible from Micromatch connectors. Some pins can be used as GPIOs (for the UART receive/transmit line or for the PWM out signal) and are 3.3 V tolerant.

Four of these pins are dedicated for the Join Test Action Group (JTAG) configuration of the FPGA and are 2.5 V tolerant. The JTAG programmation can also be performed through pads that are 3.3 V tolerant, thanks to serial resistors (see [Xil05b]). The other pads serve for the Slave Serial configuration of the FPGA and are also 3.3 V tolerant.

4.2 Configuration

Generally, a circuit is designed using a high level language, such as Verilog or Very high speed integrated circuits Hardware Description Language (VHDL). The resulting files of this process are compiled into a *bitstream* file, defining which logic function each cell will implement, and how they will be interconnected together and with the I/Os. A complete bitstream for the XC3S400 contains 1′699′136 bits.

Providing a bitstream was successfully created, the FPGA board currently supports two methods of configuration: Slave Serial and JTAG. The first one is the simplest and

the most limited, while the second one is more elaborate and thus offers several useful options.

4.2.1 Slave Serial

The Slave Serial mode depends on four dedicated pads of the FPGA board: PROG_B, INIT_B, CCLK, and DIN. The configuration memory of the FPGA is first entirely cleared. The bitstream is then serially loaded on DIN (Most Significant Bit first). DIN is sampled at each rising edge of CCLK (external clock).

Using the Slave Serial mode, the FPGA can only be programmed completely after an erasure of the current configuration memory. DPR is not supported in this mode, as well as the readback of the configuration.

4.2.2 JTAG

JTAG is a standard from the Institute of Electrical and Electronics Engineers (IEEE). It initially provides a mean to ensure the integrity of individual board-level components and their interconnections, through instructions issued on Test Access Ports (TAP). In addition, a device can implement a set of user-defined instructions, such as configure and verify.

FPGAs are generally fully compliant with the JTAG standard, which can thus be used to perform several interesting functions on the chip. The mandatory elements of the norm include the TAP, the TAP controller, the instruction register, the instruction decoder, the boundary-scan register, and the bypass register. Some details of the JTAG architecture are nonetheless specific to the device.

The TAP of the XC3S400 are accessible through the pads (3.3 V) or the pins (2.5 V) of the FPGA board. They are called TCK (Test Clock), TMS (Test Mode Select), TDI (Test Data In), and TDO (Test Data Out). The sequence of states through the TAP controller (Figure 4.2) is determined by the value of TMS on the rising edge of TCK. TDI is a serial input (sampled on the rising edge of TCK) for all JTAG instruction and data registers. TDO is a serial output (changes state on the falling edge of TCK) for all JTAG instruction and data registers.

A JTAG configuration sequence starts in the Test-Logic-Reset state. Moving to the SHIFT-IR state, a configuration instruction (CFG_IN) is loaded in the instruction register (6 bits for the XC3S400). The next step is to move to the SHIFT-DR state, where the bitstream is shifted in. Finally, a last instruction (JSTART) is loaded in the SHIFT-IR state, to launch the startup sequence.

Once the configuration completes and the system is correctly working, other JTAG instructions may be applied. For instance, the FPGA may be partially reconfigured. In this case, the special configuration sequence does not clear the configuration memory and only a small part of the FPGA is modified, while the system is still running.

4.2.3 Current Method in Use

In the YaMoR project, the FPGA is configured via JTAG. The following elements are needed:

- Xilinx iMPACT software running on a PC
- Xilinx JTAG/Parallel Download Cable (Figure 4.3) connected to the parallel port of the PC

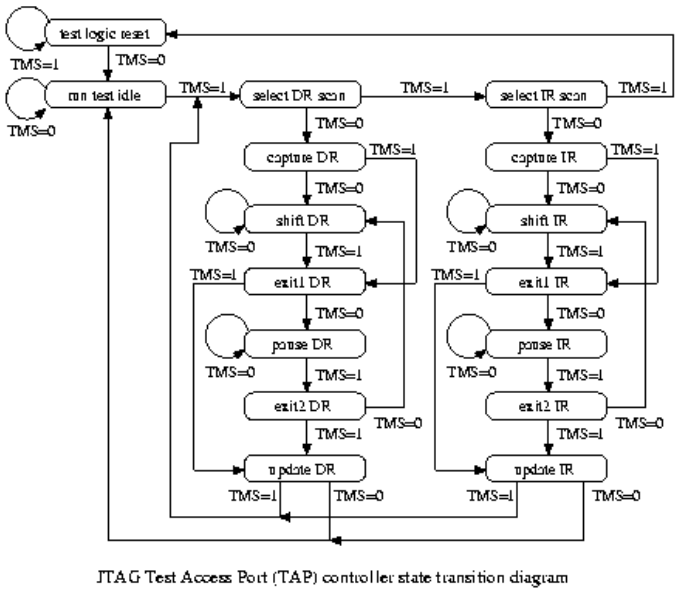


Figure 4.2: The TAP controller

- TCK, TMS, TDI, TDO pins of the FPGA connected to the Download Cable
- A reference voltage, identical to the one of the configuration pins, connected to the Download Cable

Figure 4.3: The Xilinx programmer

Starting from a bitstream file, the iMPACT software automatically generates the JTAG signals for configuring the FPGA. This method reveals some limitations, since many modules have to be configured each time the system is powered. In Chapter 5, an elegant alternative is proposed, exploiting the unused potential of the Bluetooth board to perform a wireless configuration.

Chapter 5

Wireless Configuration of the FPGA

Nothing in the world can take the place of Persistence. Talent will not; nothing is more common than unsuccessful men with talent. Genius will not; unrewarded genius is almost a proverb. Education will not; the world is full of educated derelicts. Persistence and determination alone are omnipotent. The slogan 'Press On' has solved and always will solve the problems of the human race.
Calvin Coolidge (1872 - 1933)

The FPGA configuration in the YaMoR project is achieved with a download cable that is plugged into each module one at the time. As the chip configuration memory is cleared at poweroff of the device, the overall process has to be frequently repeated. Furthermore, in the future we might want to implement an independent *on-chip* controller in each module, that will be partially reconfigured depending on the situation. Using the "classical" method seems to be rather clumsy and tedious, since plugging a cable into a moving robot is not possible without disturbing the complete system.

Providing that each robot module is equipped with a Bluetooth board, we may take advantage of it for performing a wireless configuration of the FPGA. This chapter firstly aims at chronologically exposing the key features of this implementation. In addition, the procedure for setting up a working system is fully explained.

5.1 CNF Module in Zerial

Since the YaMoR project is currently working with the Zerial firmware on the Bluetooth board, the idea of our implementation is to keep its basic functionalities and build upon it. A Configuration (*CNF*) Module is added in the BlueOS backplane and interacts with the system as illustrated in Figure 5.1.

The CNF module can be seen as a layer between the Bluetooth stack and the Zerial application. The Bluetooth serial stream goes through the CNF module, inside which each byte is analysed. As soon as two consecutive identification bytes are recognised, the CNF module starts the reception of a CNF packet and stops the transmission of the stream to the Zerial application.

Providing the Bluetooth stream does not unintentionally contains the identification bytes, this modification of the Zerial firmware has limited impact on the existing use

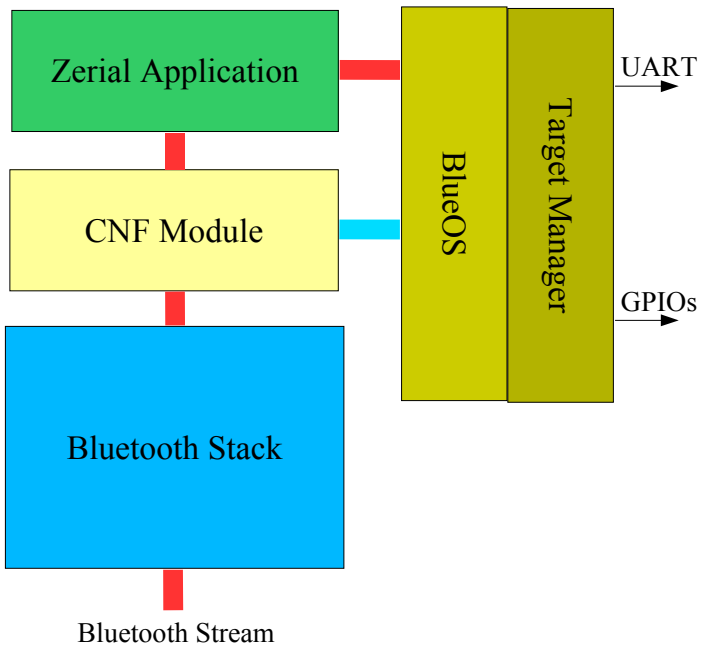


Figure 5.1: The CNF module in Zerial

of the SPP profile. Actually, it is just disabled for a short time, when the CNF module receives a packet and its command is executed.

5.2 CNF Protocol

At the lowest level of the Bluetooth stack, a bitstream is divided into several data packets of various lengths, it is thus not sent as a single packet. In the ZV4002, a signal is sent each time a packet is received, containing a pointer on the data and the number of bytes received. In order to keep track of the course of the reception, a protocol has been created. Actually, it just establishes the notion of a uniform CNF packet (Figure 5.2), into which all the functionalities of the CNF module can be encoded. In the continuation of this section, we explain the possible values for the different fields.

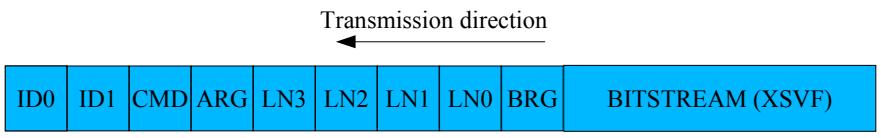


Figure 5.2: A CNF packet

5.2.1 IDs

ID0 has the default value $0x43$, while ID1 has $0x57$. This fields serve for the identification of a CNF packet and their values have been randomly chosen (it can easily be changed when needed). When these two bytes are recognised in the Bluetooth stream, a reception state machine is launched and waits for the next fields.

5.2.2 CMD

The CMD field is used to specify what function the CNF module should execute. It can take the following values (this set of commands is easily extensible) :

- $0x00$: Retrieve informations about the bitstreams currently stored in the Bluetooth board (not yet implemented)
- $0x01$: Program the FPGA
- $0x02$: Store a bitstream
- $0x03$: Return an XRAM memory pool to the BlueOS

5.2.3 ARG

The ARG field contains the parameters for the commands above. It is organised as in Figure 5.3.

Figure 5.3: The ARG field

When the FLSH bit is active, the operations will be based on the Flash memory, otherwise on XRAM. The two POSs bits specify a bitstream memory slot (Flash or XRAM).

5.2.4 LNs

The four LNs bytes contain the length (in bytes) of the bitstream field (MSB in LN3). After the reception of the bitstream, these four bytes will be prepended to it, in order to keep track of its length when it is stored. If this field is null, the rest of the packet is ignored and the reception is stopped. For instance, it happens when we want to program the FPGA with a stored bitstream.

5.2.5 BRG

The BRG byte serves for storing informations about the bitstream. In the current implementation, it is used with the value $0x80$ when we want the FPGA to be configured with this bitstream at next startup. This byte will also be mixed with the bitstream when it is stored.

5.2.6 BITSTREAM (in XSVF)

The BITSTREAM field contains the configuration data for the FPGA (MSB first). Further details about the XSVF format will be given in the next section.

5.3 FPGA Configuration

In the FPGA board, only the JTAG configuration method allows the partial reconfiguration of the device. This feature offers many attractive applications in the framework of modular robotics and therefore has to be available.

Using four pins of the Bluetooth board, the JTAG programmation can be performed "manually" by applying the correct sequence on the TAPs, with respect to the specification. However, this solution is not elegant and cruelly lacks of flexibility.

5.3.1 SVF and XSVF

The Serial Vector Format (SVF), was jointly developed by Texas Instruments and Teradyne in 1991. SVF is a standard ASCII format for expressing high-level JTAG chain operations in a compact and portable fashion. The need for SVF arose from the desire to have vendor-independent JTAG test patterns that are transportable across a wide selection of simulation software and test equipment. Complicated programming algorithms can be translated into generic SVF instructions, requiring no special knowledge of the target device.

An SVF file can be generated from iMPACT, using a *.bit* file. The JTAG operations are stored in the form of a set of SVF statements, that can be read, modified, or written manually in any text editor. Globally, they define the information that have to be shifted in the JTAG chain.

Since an ASCII format wastes too much memory space in embedded applications, the Xilinx SVF (XSVF) has been developed. It offers the functionalities of SVF in a compact, binary format. Its use is nonetheless restricted to Xilinx devices, for which it is optimized. XSVF files can be directly generated from iMPACT or translated from an SVF file. For more information about SVF and XSVF format, please refer to [Xil02].

5.3.2 Implementation

Xilinx supplies a C-code algorithm (see [Xil04a]) that interprets the XSVF files and provides the required JTAG TAP stimulus. Originally, the source code is written for a Windows platform (stimulus are sent through the parallel port of the PC) and has thus to be adapted for the ZV4002.

Mainly, four functions specific to the platform have to be implemented:

- A function for assigning a value to TDI, TCK and TMS
- A function for reading TDO
- A function providing a byte of the XSVF data each time it is called
- A function for waiting a certain amount of time

In a Spartan-3, the last function has to be understood as a number of TCK cycles. In the original code, the meaning of this function was to wait some microseconds with

respect to the frequency of the processor. As this is not extremely well documented, it took quite a long time to realize it.

The Xilinx code uses a special structure to store information about the current SVF instruction. A parameter specifies the maximum number of bytes an instruction may contain. Originally, a conservative value has been chosen (7000) and has to be lowered (128) to fit on the small ZV4002 stack memory.

When a task runs for a too long time, the BlueOS resets the system for two main reasons:

- Watchdog timer[1] reached 0
- Unhandled interrupts accumulate on a queue

To cope with this problem, during the FPGA configuration, the watchdog timer and the external interrupts have to be disabled.

5.4 Setting up a Working System

The intent of this section is to describe the different steps needed to build a platform for configuring the FPGA via Bluetooth. We assume the user has some knowledge in electronics and is in possession of the following elements:

- A PCB with a ZV4002, a Flash, an external SRAM, 4 GPIOs pins and 2 UART pins
- A PCB with a Xilinx FPGA and four 3.3 V tolerant JTAG configuration pads
- A PC equipped with Bluetooth
- The CD-ROM of the project
- The *Arm Development Suite* for changing parameters if needed (including cable for re-flashing)

5.4.1 Electronic Connections

The first step is to connect the four GPIOs of the ZV4002 to the JTAG TAPs of the FPGA. In the compiled firmware provided in the CD-ROM, the pins have been assigned as follows:

- GPIO7 to TDO
- GPIO8 to TDI
- GPIO10 to TCK
- GPIO11 to TMS

Ensure the GPIOs supply enough current to configure the FPGA. In the first implementation, GPIO12 was used to drive a JTAG pin, but as it was also connected to a LED, the system did not work.

[1] A watchdog timer is a piece of hardware that can be used to automatically detect software anomalies and reset the processor if any occur. Generally speaking, a watchdog timer is based on a counter that counts down from some initial value to zero. The embedded software selects the counter's initial value and periodically restarts it. If the counter ever reaches zero before the software restarts it, the software is presumed to be malfunctioning and the processor's reset signal is asserted. The processor (and the embedded software it's running) will be restarted as if a human operator had cycled the power.

5.4.2 Software

If this is not already done, flash the firmware provided in the CD-ROM (refer to Chapter 3 for a tutorial). Providing Bluetooth is correctly installed on your PC, the FPGA can now be managed remotely using two kinds of software.

Bluemove

Bluemove is a Java application that was originally developed to control the movement of YaMoR modules through a user-friendly Graphical User Interface (GUI). In June 2005, the CNF protocol has been incorporated as a new feature. A bitstream is intuitively assigned to each module and sent via Bluetooth. The main advantage of this solution is its simplicity and its portability. However, all the functionalities of the CNF protocol are not yet provided. For more information about Bluemove, please refer to [JD05a].

Writing a Small Program

An example of a small program for configuring the FPGA via Bluetooth is provided in the CD-ROM, under the directory *Configure*. This program may be freely edited and works as follows:

- In Windows, launch a research of Bluetooth peripherals
- Connect to the YaMoR module using the PIN 1234
- A virtual serial port is assigned to the module (*com9* for instance)
- In the program, modify the statement that opens the serial port
- Change the different parameters with respect to the CNF protocol
- Build everything and launch

Chapter 6

Experimental Results and Further Work

The result justifies the deed. (Exitus acta probat)
Ovid (43 BC - 17 AD)

During the last phase of the project, thorough experiments have been conducted to evaluate the quality of our wireless FPGA configuration solution. The limitations inherent to the YaMoR framework are first discussed, followed by the description and results of a partial reconfiguration application. Finally, some guidelines for future work are proposed.

6.1 Hardware Limitations

6.1.1 Memory

The external RAM of the Bluetooth board can store 1 MByte, which is allocated from pools. In the current implementation, there are a pool for the bitstreams and a pool for the flash temporary data. From the bitstream pool, three memory blocks of 215'000 bytes can be allocated simultaneously. From the flash pool, one memory block of 300'000 bytes is available. If the bitstream file size is bigger, the system has to be adapted and we may lose some flexibility.

The Flash memory of the Bluetooth board provides around 600'000 bytes that can be dynamically used. As the area is divided in two, we may have to use some tricks to store bitstreams larger than 300'000 bytes.

6.1.2 Connections

At the end of the semester, the wireless configuration solution has been successfully implemented in only two modules. Other modules have been tested, but it seems that the connections are too weak and do not deliver enough current to the FPGA pads. As the GPIOs only supply 2 mA, the 100 Ω-resistors on the FPGA pads may be removed.

6.2 Description of the Experiments

A totally independent controller for the servo motor of the YaMoR has been designed. It is composed of two parts:

- A PWM
- A table for storing the input values of the PWM

The two modules communicate through a bus macro. The PWM is a fixed area and the table may be modified while the system is running. This promising technique is referred as Dynamic Partial Reconfiguration (see [Xil04b]). In this experiment, we apply the difference-based style (i.e. a partial bitstream is generated from two complete one, differing in only some look-up tables).

A complete bitstream (210 KB) implementing a sinusoidal movement is sent as the initial configuration of the FPGA and the servo motor is powered. Partial bitstreams (30 KB and 50 KB) implementing triangular and square movements are then sent without shutting down the FPGA. Furthermore, the Flash and XRAM storage are also tested.

6.3 Results and Critics

The experiments were conducted with success and the following timings were obtained:

- 2 seconds for the Bluetooth transmission of a complete bitstream
- 5 seconds for the flashing of a complete bitstream
- 14 seconds for the full configuration of the FPGA
- 3 seconds for the dynamic partial reconfiguration of the FPGA

Globally, the experiment is satisfactorily, as we could demonstrate that our solution is perfectly working. However, the partial reconfiguration of the device should take less time and be performed more smoothly. Actually, in this experiment the system is paused during the reconfiguration, to avoid strange behaviours when some parts of the table are already reconfigured and others not. A solution could be to have two copies of the table in the FPGA and to use the copy while we are reconfiguring.

Since the management of the Flash is quite a complex task, an intensive usage may sometimes lead to lost bitstreams. In this case, it is recommended to restart the Bluetooth board and to reflash the entire bitstream. These problems appears nevertheless only on rare occasions and should not be harmful to the FPGA.

6.4 Further Development

For the continuation of the project, we consider the following ideas:

- Further testing and improvements on the firmware
- Adding some Flash and XRAM on the Bluetooth board
- Development of a complete GUI for managing the platform configuration via Bluetooth

- Integration and adaptation of our solution into another platform
- Combination of our solution with a Microblaze design

We hope that we will have the opportunity to work on these projects in the future and especially in the continuation of YaMoR. We imagine a totally independent robot equipped with sensors, capable of self-reconfiguring its shape and its controller in a smoothly way. Several milestones have to be reached before we can see this kind of modular robot evolve in our laboratory.

Chapter 7

Conclusion

When I'm working on a problem, I never think about beauty. I think only how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong.
R. Buckminster Fuller (1895 - 1983)

At the end of this semester project, we are proud to present a working system that performs the Bluetooth configuration of an FPGA in a YaMoR module. Its main features are:

- Full configuration in 14 seconds
- Storage of bitstreams in Flash or XRAM
- Configuration from Flash at startup

Furthermore, the system has been successfully integrated in the Bluemove application, dramatically reducing the time to set up a complete modular robot. A standalone application will also soon be created to use the system in other domains than modular robotics. Finally, this report should serve as a reference guide for new students working on the YaMoR project.

Throughout this project, an immensely valuable knowledge has been acquired, ranging from practical electronics to extreme C programming. Working with electronics is always a challenge, since a small detail may cause a failure that can only be solved after hours of intensive debugging. However, the final satisfaction is the best salary of all the investements.

Appendix A

Bluetooth Board Schema

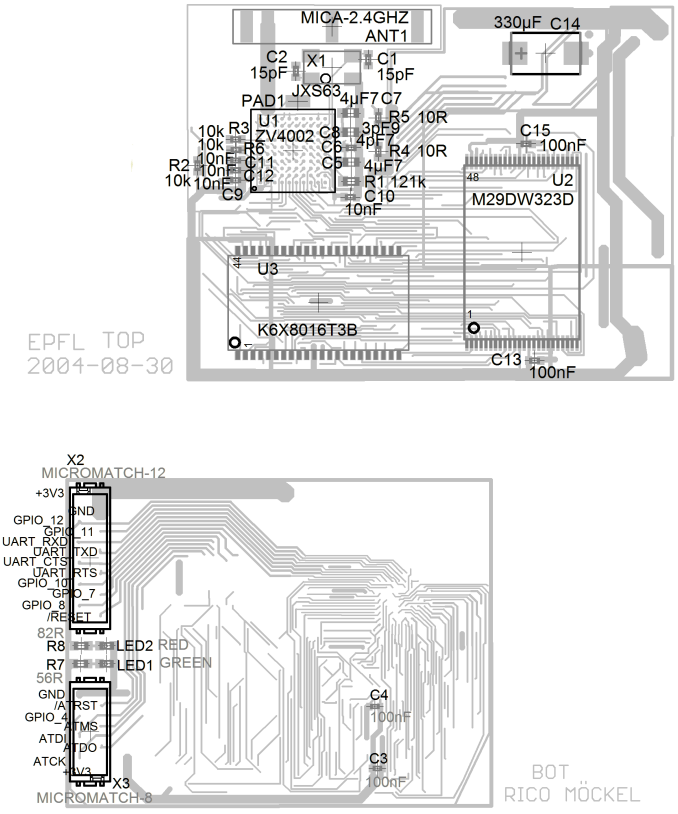
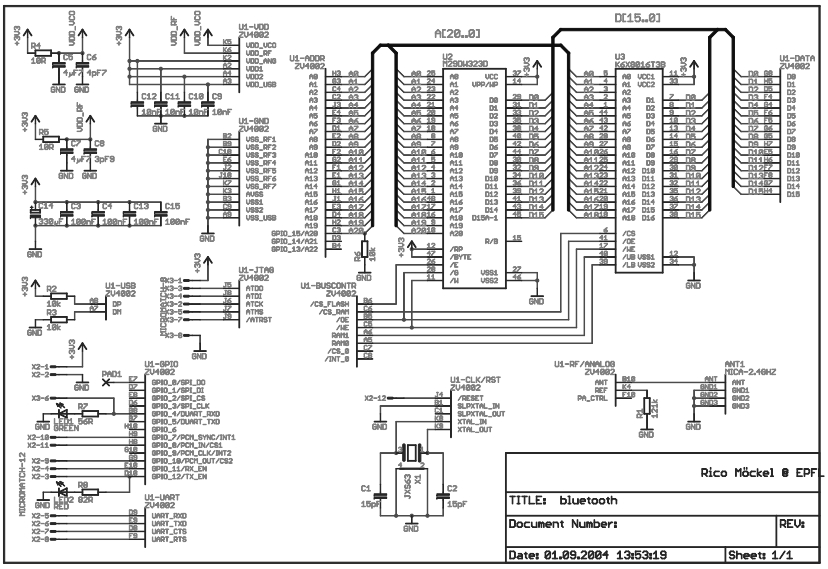


Figure A.1: Bluetooth board PCB



23.11.2004 21:49:43 D:\BLUETOOTH\EAGLE\files\bluetooth.sch (Sheet: 1/1)

Figure A.2: Bluetooth board schematic

Appendix B

FPGA Board Schema

FPGA

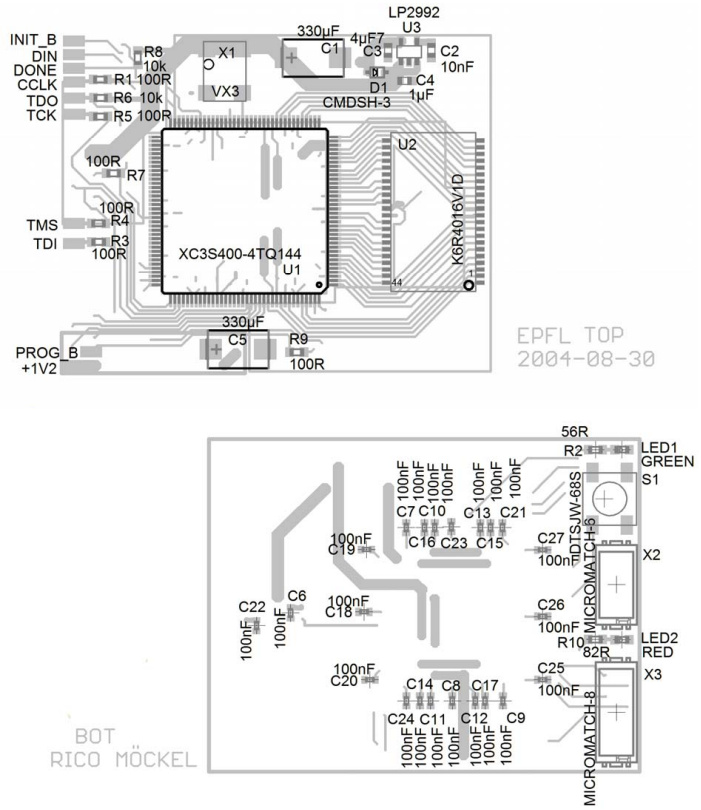
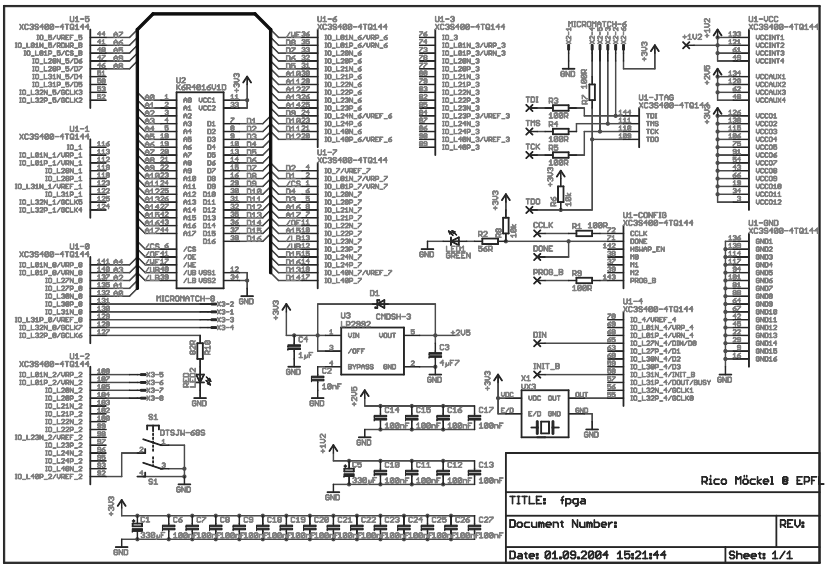


Figure B.1: FPGA board PCB



23.11.2004 21:51:20 D:\BLUETOOTH\EAGLE\files\fpga.sch (Sheet: 1/1)

Figure B.2: FPGA board schematic

Bibliography

[BS00] Jennifer Bray and Charles Sturman. *Bluetooth: Connect Without Cables*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[JD05a] Cyril Jaquier and Kevin Drapel. *Towards an improved framework for YaMoR*. PhD thesis, Swiss Federal Institute of Technology, Lausanne, 2005.

[JD05b] Cyril Jaquier and Kevin Drapel. *Using Bluetooth to Control a YaMoR Modular Robot*. PhD thesis, Swiss Federal Institute of Technology, Lausanne, 2005.

[SSW04] Andrew Sloss, Dominic Symes, and Chris Wright. *ARM System Developer's Guide*. Morgan Kaufmann, 2004.

[Xil02] Xilinx. Xapp503: Svf and xsvf file formats for xilinx devices. Technical report, 2002.

[Xil04a] Xilinx. Xapp058: Xilinx in-system programming using an embedded microcontroller. Technical report, 2004.

[Xil04b] Xilinx. Xapp290: Two flows for partial reconfiguration: Module based or difference based. Technical report, 2004.

[Xil05a] Xilinx. Ds099: Spartan-3 fpga family: Complete data sheet. Technical report, 2005.

[Xil05b] Xilinx. Xapp453: The 3.3v configuration of spartan-3 fpgas. Technical report, 2005.

[YDR00a] Mark Yim, David Duff, and Kimon Roufas. Modular reconfigurable robots, an approach to urban search and rescue. *1st Intl. Workshop on Human-friendly Welfare Robotics Systems*, 2000.

[YDR00b] Mark Yim, David Duff, and Kimon Roufas. Polybot: A modular reconfigurable robot. In *ICRA*, pages 514–520, 2000.

[YRD+03] Mark Yim, Kimon Roufas, David Duff, Ying Zhang, Craig Eldershaw, and Samuel B. Homans. Modular reconfigurable robots in space applications. *Auton. Robots*, 14(2-3):225–237, 2003.

[Zeea] Zeevo. Blueos reference guide. Technical report.

[Zeeb] Zeevo. Bluos user guide. Technical report.

[Zeec] Zeevo. Target manager reference guide. Technical report.

[Zeed] Zeevo. Target manager user guide. Technical report.

[Zeee] Zeevo. Zerial interface reference guide. Technical report.

[Zeef] Zeevo. Zerial interface user guide. Technical report.